De Montfort University

FPGA Neural Controller for Three-Phase Sensorless Induction Motor Drive Systems

Andrei Dinu

June 2000

A thesis submitted in partial fulfilment of the requirements of
De Montfort University
for the degree of Doctor of Philosophy

ABSTRACT

This thesis presents the research work carried out in the area of design, simulation and implementation of sensorless induction motor drive systems. State-of-the-art control strategies for induction motors are reviewed and the neural network concept is discussed with a view to its application to control systems.

The control strategy includes an improved current control algorithm and a novel sensorless speed control algorithm. The strategy is based on an equivalent three-phase circuit of the induction motor that contains a resistor R, an inductor L and an internal voltage source 'e' on each phase. The circuit is considered symmetrical so the resistances and the inductances on the three phases are equal. The current control method is enhanced by an original on-line induction estimation algorithm, which determines the inductance in the equivalent R-L-e circuit. This information allows the optimisation of the switching process of the PWM inverter in order to minimise the current ripple and to maximise the transient response speed. The current control method is first algebraically analysed and then expressed in geometrical terms using space vectors in the two-dimensional complex plane. The geometrical form of the algorithm is suitable for hardware implementation using neural networks and the corresponding implementation approach turns out to be superior to the implementation methodology that involves only classical digital circuits.

The new sensorless speed control algorithm uses space vectors expressed in polar co-ordinates instead of rectangular co-ordinates in order to reduce the amount of algebraic calculations compared with the classical space vector control method. The implementation strategy developed leads to a reduced hardware complexity controller by transferring part of the control tasks to neural networks performing trigonometric calculations.

A new algorithm for neural network hardware implementation is developed which uses only basic logic gates. It is mathematically analysed and proven superior to other relevant algorithms for a certain class of applications. The algorithm converts the network neurone by neurone and then minimises the gate count by eliminating the redundant logic structures. The implementation process has been automated by means of a set of C++ programs that transforms the matrix description of a feed-forward neural network into a VHDL model of the corresponding logic gate implementation.

The controller model is developed using VHDL in such a manner that it can be easily rescaled according to the size of the FPGA devices available and to the accuracy/performance requirements of the electrical drive application. Practical test results on a 0.5 kW electrical drive that includes an XC4010XL FPGA controller are presented and discussed.

The outcome is a novel FPGA controller is developed for a VSI-PWM power inverter system for induction motors variable speed drive system. The new approach involves an original control algorithm and uses hardware implemented feed forward neural networks in conjunction with classical digital structures.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Dr. Marcian Cirstea for his invaluable guidance and constant support both as my first supervisor and as a friend. This work would not have been possible without him. I am also greatly indebted to my second supervisor, Prof. Malcolm McCormick for his excellent advice throughout the course of my research.

Special thanks are due to Dr. Antonio Ometto and Prof. Nicola Rotondale from the Department of Electrical Engineering at L'Aquila University in Italy. The collaboration with them and their feedback on this research are highly appreciated.

I must acknowledge the stimulating e-mail contacts I have enjoyed with Dr. Valeriu Beiu, researcher at Los Alamos National Laboratory, USA. He generously offered up-to-date information in the field of hardware implemented neural networks. I will always be grateful to my colleague Dr. J.G. Khor for his help with information and advice concerning the practical experiments. Many thanks are due to others academics at DMU and to the technical support and administrative staff, especially Dilip Chauhan, Tim O'Mara and Sheila Hayto.

I also wish to thank my parents Mr. and Mrs. Dinu for their unfailing support and encouragement. Last, but not in any way least, I am particularly grateful to my wife Anca for her faith in me and her patience during all these years.

CONTENTS

1 INTRODUCTION

1.1 THE RESEARCH AREA

Electric motors are major users of electricity in industrial plants and commercial premises. Motive power accounts for almost half of the total electrical energy used in the UK and nearly two-thirds of industrial electricity use. It is estimated that over ten million motors, with a total capacity of 70 GW, are installed in UK industry alone [10]. Although many motor types are currently in use (synchronous motors, PM synchronous motors, DC motors, DC-brushless motors, switched reluctance motors, stepping motors), most of the industrial drives are powered by three-phase induction motors. The majority of them is rated up to 300 kW and can be classified as illustrated by Fig. 1-1.

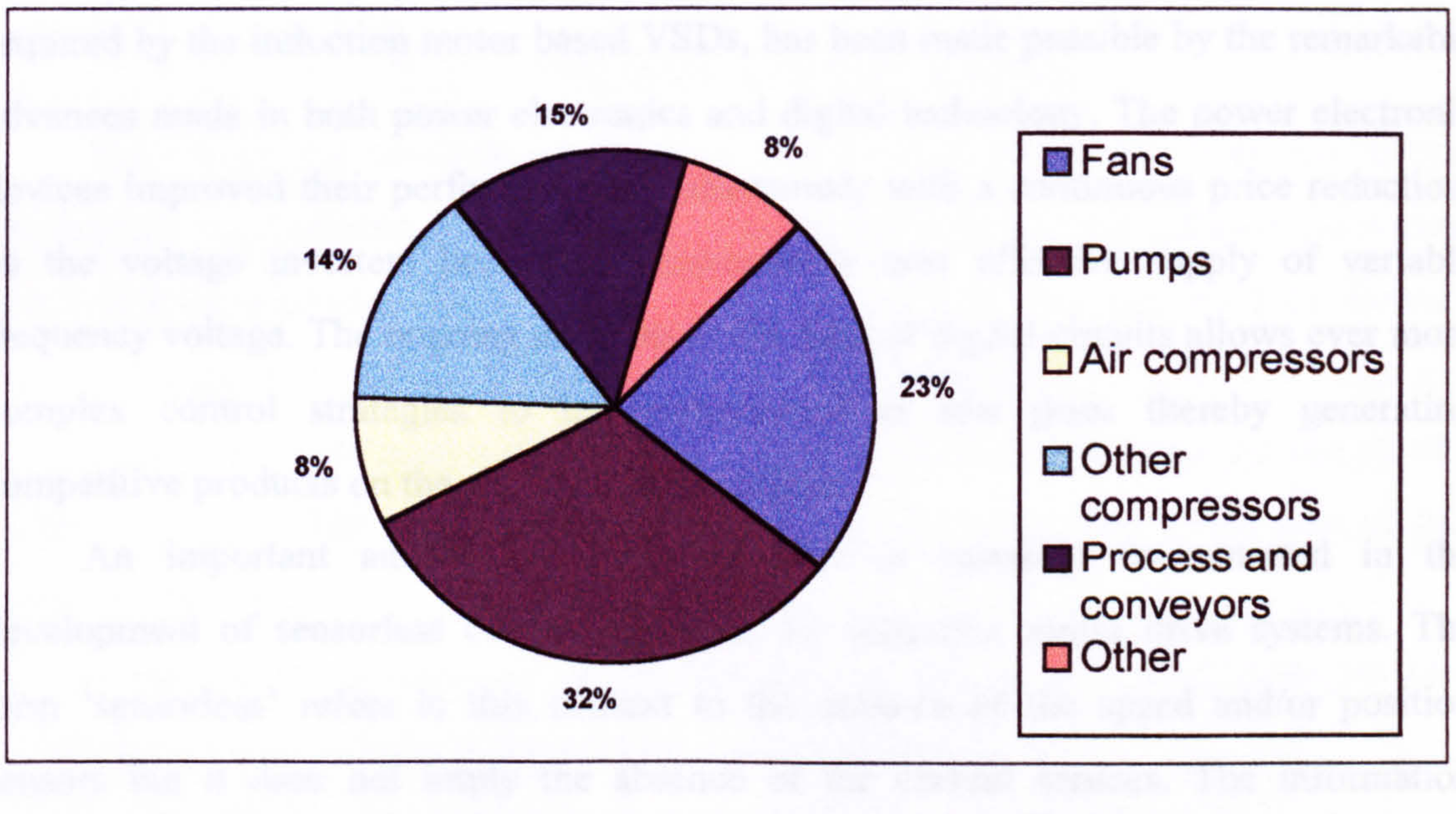

Fig. 1-1 - Energy consumption by induction motors up to 300 kW in industry

The large industrial use of induction motors has been stimulated over the years by their low prices and reliability. Although initially used as fixed speed motors, the advanced control strategies developed in the last four decades made it possible to use induction motors for high performance variable speed drives (VSD), replacing many of the more expensive and less reliable DC motors previously used. Moreover, induction motor-based VSDs are now used for applications that traditionally involved fixed speed

drives. For instance, VSDs replace the old solution of using adjustable nozzles in applications involving fans or pumps. An adjustable nozzle can ensure a variable flow of fluid, but at the cost of decreasing the motor efficiency. A VSD is capable of performing the same task while maintaining the motor efficiency at high levels. This is an essential factor because the price of the electricity consumed by the motor is much larger than its purchase price. For instance, a modest-sized 11 kW induction motor costs as little as £300 to buy, but it can accumulate running costs of over £30,000 in ten years [10]. Therefore, even small efficiency improvements may produce impressive cost savings. In addition to the potential for saving energy, the he use of VSDs has several important benefits including:

- improved process control and hence enhanced productivity
- soft starting, soft stopping and regenerative braking
- unity power factor
- wide range of speed, torque and power
- good dynamic response (comparable with DC drives)

The successful implementation of the sophisticated non-linear control algorithms required by the induction motor based VSDs, has been made possible by the remarkable advances made in both power electronics and digital technology. The power electronic devices improved their performance simultaneously with a continuous price reduction, so the voltage inverters became an increasingly cost effective supply of variable frequency voltage. The ongoing progress in the field of digital circuits allows ever more complex control strategies to be implemented at low price thereby generating competitive products on the electrical drive market.

An important amount of research effort is currently concentrated in the development of sensorless control strategies for induction motor drive systems. The term 'sensorless' refers in this context to the absence of the speed and/or position sensors but it does not imply the absence of the current sensors. The information normally supplied by the speed sensor is in this situation replaced by the result of calculations based on the value of the stator currents and voltages. The sensorless control approach increases the difficulty of the control task but in some practical situations, there are strong reasons to eliminate the speed sensor due to both economical and technical reasons. For example, the pumps used in oilrigs to pump out the oil have to work under the surface of the sea, sometimes at depths of 50 meters. Obtaining the speed measurement data up to the surface means extra cables, which is extremely

expensive, therefore reducing the number sensors and measurement cables provides a major cost reduction [12].

One of the most promising approaches for the control of complex and non-linear systems is the use of artificial neural networks (ANN). Neural networks are information processing systems that are composed of a large number of interconnected basic units named neurones. The operation and the structure of the constituent neurones are inspired from their biological counterparts. The neural paradigm has two main advantages:

- flexibility and the adaptability of the control system, generated by the learning capability of the neural networks.
- tremendous data processing speed, made possible by the massive parallel structure of the neural networks.

Most of the current control applications involve software implementations and exploit the learning capability of the neural networks, but only the hardware implementations are capable to take advantage of the parallel data processing advantage of the neural networks [74].

1.2 THESIS OVERVIEW

The aim of the research presented in this thesis is to develop a controller that implements an improved current control strategy and a simple but efficient sensorless speed control algorithm for induction motors. Hardware implemented feed forward neural networks are used in order to maximise the operation speed of the controller and avoiding the use of external look-up tables which unnecessarily increase the complexity of the controller. The main objectives to be achieved within the stated aim of this research work are:

- The theoretical development of the new current and speed control strategies in a manner that allows efficient hardware implementation.
- The identification of a new optimal methodology for neural network implementation into digital circuits.
- The hardware design of the neural controller implementing the two control strategies.
- The controller performance assessment by simulations and practical experiments.

The new controller is included in a typical sensorless induction motor drive system (Fig. 1-2). The three-phase motor is supplied with variable frequency and

variable amplitude voltage by a three-phase VSI-PWM inverter, which is fed with DC voltage generated by an controlled rectifier via a low-pass gamma filter. The controller receives the reference speed information, calculates the necessary stator currents of the motor and generates the appropriate control signals to the PWM inverter, so that the required currents are achieved. The stator currents are calculated in a manner that ensures that the actual speed of the rotor follows the reference value as closely as possible.

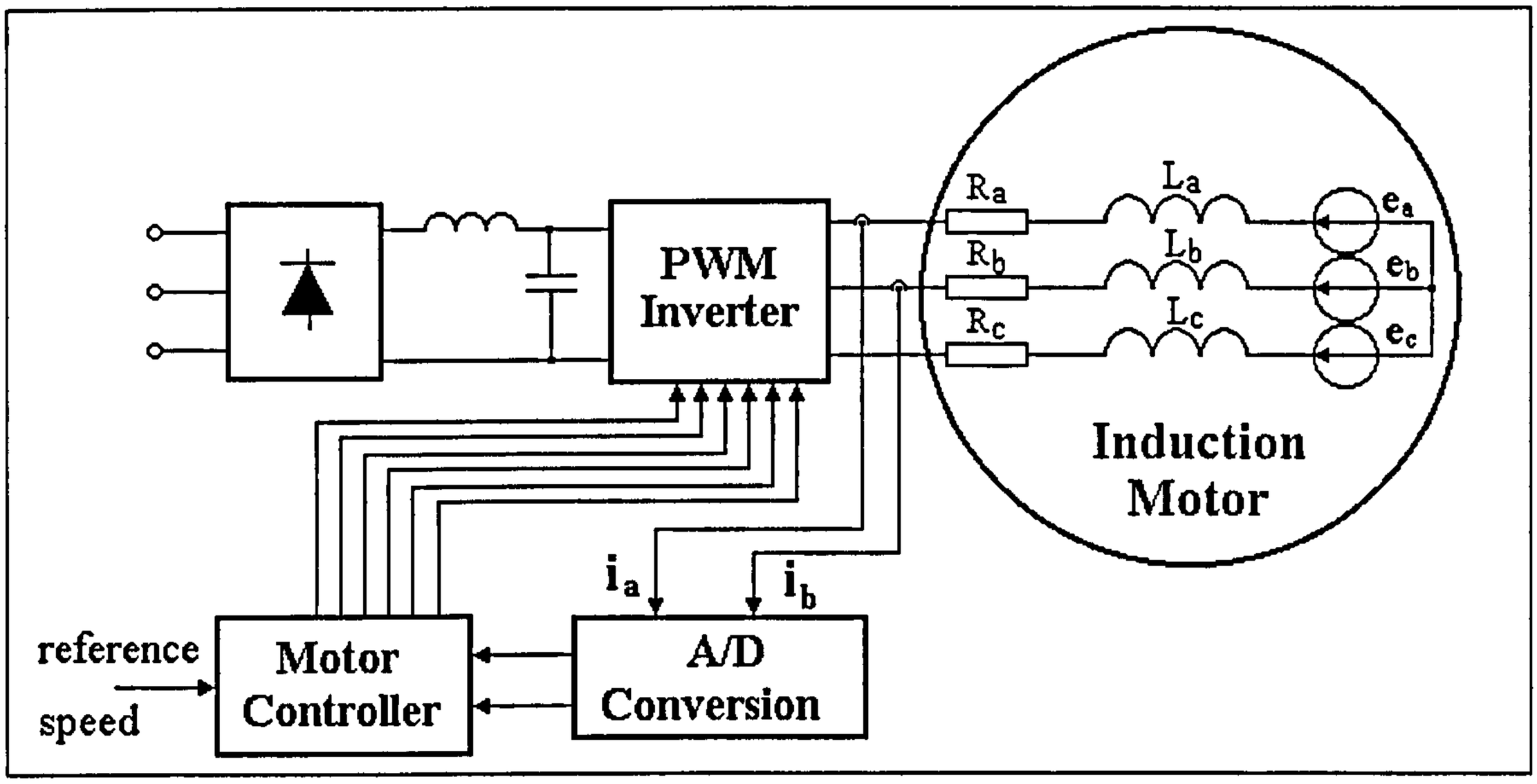

Fig. 1-2 – The block diagram of the drive system that includes novel motor controller

The control principles implemented by the new motor controller rely on an equivalent R-L-e circuit of the induction motor that contains a resistor R, an inductor L, and an internal voltage source on each phase. The controller uses the information regarding the voltage across the motor and the currents through the stator windings to perform an on-line estimation of the equivalent inductance L. The equivalent resistance R is considered a known quantity. The estimated value of the inductance is used both to optimise the control of the stator current and to calculate the internal voltages e_a, e_b and e_c. These voltages are used to determine the motor speed, thereby eliminating the need for a speed sensor.

The content of the thesis is divided in 8 chapters. While this chapter introduces the subject of the research work, chapter 2 presents the space-vector model of the three-phase induction motors and the most important speed control strategies developed so far. It also demonstrates that many of these strategies need to include stator current control algorithms, and reviews the most important of them. The chapter ends with a presentation of the hardware implementation techniques for electrical drive control

strategies and their relation to the design methodologies. Chapter 3 introduces basic elements concerning neural networks and their application to control systems, and discusses the hardware implementation methods available nowadays. Chapter 4 describes in detail the improved current control strategy and the new sensorless speed control method. The description contains thorough mathematical demonstrations and highlights the importance of each parameter of the two control algorithms. A new FPGA implementation method for neural networks is presented in chapter 5. The method is compared with other relevant implementation algorithms from the hardware complexity perspective and its superiority for a certain class of applications is demonstrated. This chapter also describes the design and the implementation of the neural network that is used by the induction motor controller. Chapter 6 presents the architecture and the operation of the motor controller and shows the place of the neural network among the other digital structures included in the controller. The practical test results are presented and discussed in chapter 7, while chapter 8 formulates a list of conclusions and shows possible further developments of this research work.

1.3 ORIGINAL CONTRIBUTIONS OF THE THESIS

The original achievements of the present research work can be summarised as follows:

- The development of a neural network hardware implementation algorithm that uses only AND, OR and NOT logic gates and minimises the generated hardware structure.
- The automation of the implementation algorithm by means of C++ programs that start with the mathematical description of the neural network and generate the optimised VHDL model of the corresponding logic architecture.
- The development of a flexible current control strategy that is suitable for neural network implementation and which allows good control over the ratio between the operation precision and the complexity of the hardware implementation. This ratio can be modified by altering the number of neurones in the corresponding neural network. This strategy can be applied to a large range of three-phase power systems including induction motors.
- The design of an original on-line inductance estimation algorithm that can be combined with the current control strategy to generate an universal current control

structure that automatically adjusts the PWM switching process to the parameters of the load.

- The optimal implementation of the induction estimation algorithm using a feed-forward neural network implemented into digital hardware.
- The development of a new speed estimation algorithm for induction motors, using space vectors defined in polar co-ordinates instead of rectangular co-ordinates. The new approach requires a smaller amount of calculations than other algorithms and it is appropriate for implementation into low complexity hardware by using neural networks. The neural network approach allows the modification of the implementation complexity in accordance with the hardware resources available.
- The development of a sensorless induction motor control strategy that uses the polar co-ordinate approach and includes the previously mentioned speed estimation algorithm.
- The design of a digital sinewave generator with adjustable frequency that uses the differential modulation technique to minimise the size of the associated look-up table.
- The VHDL design of numerous other digital structures that are included in the novel motor controller.

2 THE OPERATION AND CONTROL OF INDUCTION MOTORS - REVIEW

The replacement of DC motors with induction motors in many industrial plants has stimulated the research in modelling and control of induction motors since 1960s. This chapter presents the space vector model of the induction motor, which is the most appropriate mathematical model for drive system design. Based on this model, the main speed control methods available today are classified and analysed. The chapter also discusses different current control algorithms that can be used in conjunction with the speed control methods, underlining their advantages and disadvantages. In the end, the implementation solutions for motor control applications are presented and compared in terms of speed and price.

2.1 THE SPACE VECTOR CONCEPT IN ELECTRICAL POWER SYSTEMS

The space vector concept originated in the study of Y-connected induction motors but it can be extended to describe all three-phase electric system regardless of their exact nature: electrical generators, electrical motors, transformers etc. The basic principle is to transform the scalar electromagnetic quantities describing the system (currents, voltages and magnetic fluxes) into two-dimensional vectors named space vectors. One single space vector replaces a set of three scalar quantities of the same type, thereby generating a more compact notation for the mathematical equations. Therefore, space vectors are largely used to analyse the operation of three-phase electrical machines [93], [109], [132], [134].

If 'A' is an electromagnetic quantity then A_a, A_b and A_c are the three values corresponding to the three system phases. They are initially associated with two-dimensional vectors situated on three directions 120^0 apart in a plane: $\vec{A}_a$, $\vec{A}_b$, and $\vec{A}_c$ as illustrated in Fig. 2-1. Adding the three vectors together, a single two-dimensional vector is obtained according to equation (2-1). $\vec{A}$ is the space vector associated with

scalar quantities A_a, A_b and A_c. The vector components on the real axis (axis 'd') and on the imaginary axis (axis 'q') are given in (2-2).

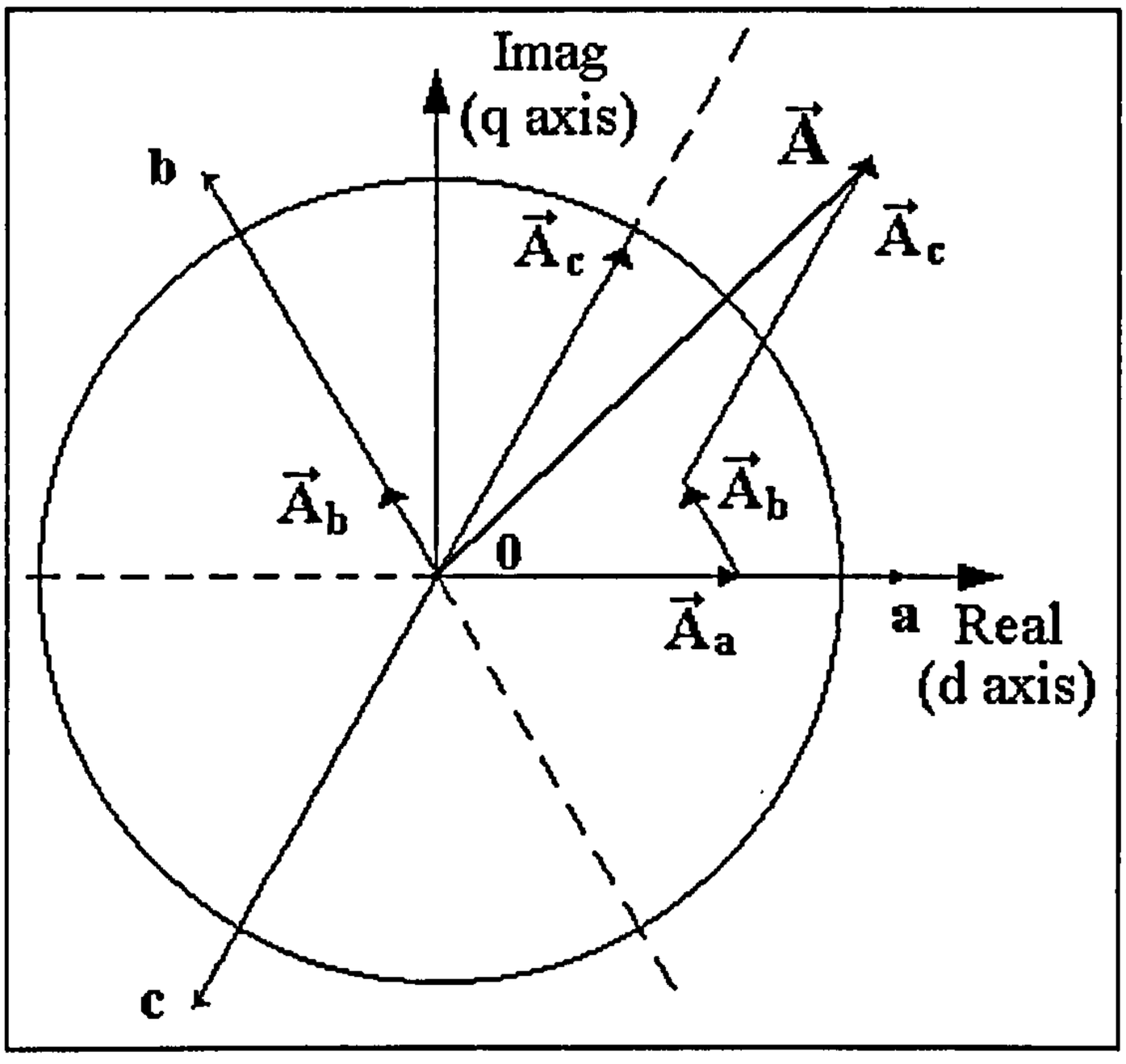

Fig. 2-1 -The relation between phase quantities and the corresponding space vector

$$\vec{A} = \vec{A}_a + \vec{A}_b + \vec{A}_c \tag{2-1}$$

$$\begin{cases} A_d = A_a \cdot \cos(0) + A_b \cdot \cos\left(\frac{2\pi}{3}\right) + A_c \cdot \cos\left(\frac{4\pi}{3}\right) = A_a - \frac{1}{2}A_b - \frac{1}{2}A_c \\ A_q = A_a \cdot \sin(0) + A_b \cdot \sin\left(\frac{2\pi}{3}\right) + A_c \cdot \sin\left(\frac{4\pi}{3}\right) = \frac{\sqrt{3}}{2}A_b - \frac{\sqrt{3}}{2}A_c \end{cases} \tag{2-2}$$

In practical calculations, the space vectors are represented either by 2x1 matrices or by complex quantities. Using matrix notation, equation (2-2) becomes (2-3) while (2-4) describes the complex number approach to space vector calculation (2-1). Two-dimensional vectors like the one in (2-1) are distinguished from the equivalent complex numbers by means of notation. Underlined symbols stand for complex values while vectors are represented by symbols placed under an arrow. Thus, $\underline{A}$ is a complex number while $\vec{A}$ is a vector.

$$\begin{pmatrix} A_d \\ A_q \end{pmatrix} = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \cdot \begin{pmatrix} A_a \\ A_b \\ A_c \end{pmatrix} \tag{2-3}$$

$$\begin{cases} \underline{A} = A_a + \varepsilon \cdot A_b + \varepsilon^2 \cdot A_c \\ \varepsilon = \cos\left(\frac{2\pi}{3}\right) + j \cdot \sin\left(\frac{2\pi}{3}\right) \end{cases} \quad (2\text{-}4)$$

The transformation of the set of three scalar variables into a space vector is equivalent to a transformation from a three-phase system into a two-phase system. The inverse transformation can be calculated based on the property that the algebraic sum of the three scalar values is always null. This property is shared by all electromagnetic quantities related to individual phases (currents, voltages and magnetic fluxes) if the power supply generates symmetric voltages and the load is symmetric and Y-connected.

$$A_a + A_b + A_c = 0 \quad (2\text{-}5)$$

Combining (2-5) with equation (2-2), the system (2-6) is generated from which (2-7) is derived. The system (2-7) describes the inverse transformation of a space vector into the corresponding set of three scalar phase quantities.

$$\begin{cases} A_d = A_a - \frac{1}{2}A_b - \frac{1}{2}A_c \\ A_q = \frac{\sqrt{3}}{2}A_b - \frac{\sqrt{3}}{2}A_c \\ A_a + A_b + A_c = 0 \end{cases} \quad (2\text{-}6)$$

$$\begin{cases} A_a = \frac{2}{3} \cdot A_d \\ A_b = -\frac{1}{3} \cdot A_d + \frac{1}{\sqrt{3}} \cdot A_q \\ A_c = -\frac{1}{3} \cdot A_d - \frac{1}{\sqrt{3}} \cdot A_q \end{cases} \quad (2\text{-}7)$$

2.2 THE SPACE VECTOR MODEL OF THE THREE-PHASE INDUCTION MOTOR

The mathematical models of the electrical machines are classified as lumped-parameter circuit models and distributed-parameter models. The latter are more complex but more accurate than the former. The distributed-parameter models are used for very precise calculations necessary for optimal machine design. They allow an exact calculation of the electromagnetic field and heat distribution inside the machine. The lumped-parameter models can be obtained as a simplification of the distributed-parameter models. They are used for control system design where only global quantities

like currents, torque and speed are important. Their internal distribution inside the machine is not relevant when designing controllers to govern the evolution of speed, torque and power consumption according to the particular application requirements. Furthermore, the lumped-parameter circuit model is simpler and therefore more convenient to use in the study of electric drives. The space-vector model of the induction motor is the lumped-parameter model with the largest use in the study and design of electrical drive applications.

It is common to consider as a first approximation that the rotor windings and the stator windings have a sinusoidal distribution inside the motor and no magnetic saturation is present [32], [93]. Therefore, the magneto-motive force (mmf) space harmonics and slot harmonics are neglected. Although saturation is not taken into account, the model is considered to yield acceptable results for the study of common electric drive applications [32], [93], [132].

The induction motor space vector model is derived from the basic electrical equations describing each of the stator windings and each of the rotor windings. The stator windings equations are given in (2-8) where u_{as}, u_{bs} and u_{cs} are the phase voltages, i_{as}, i_{bs} and i_{cs} are the phase currents, while Ψ_{as}, Ψ_{bs} and Ψ_{cs} are the phase magnetic fluxes.

$$\begin{cases} u_{as} = R_s i_{as} + \dfrac{d\Psi_{as}}{dt} \\ u_{bs} = R_s i_{bs} + \dfrac{d\Psi_{bs}}{dt} \\ u_{cs} = R_s i_{cs} + \dfrac{d\Psi_{cs}}{dt} \end{cases} \tag{2-8}$$

The associated space vectors (expressed as complex numbers) are obtained by multiplying the second equation in (2-8) with ε and the third with ε^2, after which all the three equations are added together. The complex number ε is defined in (2-4). The conversion of the three scalar equations into one space vector equation is illustrated by (2-9) and (2-10).

$$\begin{cases} u_{as} = R_s i_{as} + \dfrac{d\Psi_{as}}{dt} \\ \varepsilon u_{bs} = R_s \cdot \varepsilon i_{bs} + \varepsilon \dfrac{d\Psi_{bs}}{dt} \\ \varepsilon^2 u_{cs} = R_s \cdot \varepsilon^2 i_{cs} + \varepsilon^2 \dfrac{d\Psi_{cs}}{dt} \end{cases} \tag{2-9}$$

$$\begin{cases} \underline{u}_s^s = u_{as} + \varepsilon \cdot u_{bs} + \varepsilon^2 \cdot u_{cs} \\ \underline{i}_s^s = i_{as} + \varepsilon \cdot i_{bs} + \varepsilon^2 \cdot i_{cs} \\ \underline{\Psi}_s^s = \Psi_{as} + \varepsilon \cdot \Psi_{bs} + \varepsilon^2 \cdot \Psi_{cs} \end{cases} \Rightarrow \underline{u}_s^s = R_s \underline{i}_s^s + \frac{d\underline{\Psi}_s^s}{dt} \qquad (2\text{-}10)$$

Different reference frames (still or rotating) can be used to calculate the co-ordinates of the electromagnetic space vectors [32]. Equations (2-10) are written in the stator reference frame. Any rotating reference frame is defined by the electrical angle function $\theta(t)$ that indicates the relative position to the still reference frame. Alternatively, it can be defined by the electrical rotation speed $\omega_e(t)$ and the initial electrical angle $\theta(0)$. For a general rotating frame (illustrated by Fig. 2-2) the equations (2-10) are transformed into (2-11).

$$\begin{cases} \underline{u}_s^\theta = \underline{u}_s \cdot e^{-j\theta} \\ \underline{i}_s^\theta = \underline{i}_s \cdot e^{-j\theta} \\ \underline{\Psi}_s^\theta = \underline{\Psi}_s \cdot e^{-j\theta} \\ \underline{u}_s^\theta \cdot e^{j\theta} = R_s \underline{i}_s^\theta \cdot e^{j\theta} + \frac{d}{dt}\left(\underline{\Psi}_s^\theta \cdot e^{j\theta}\right) \end{cases} \qquad (2\text{-}11)$$

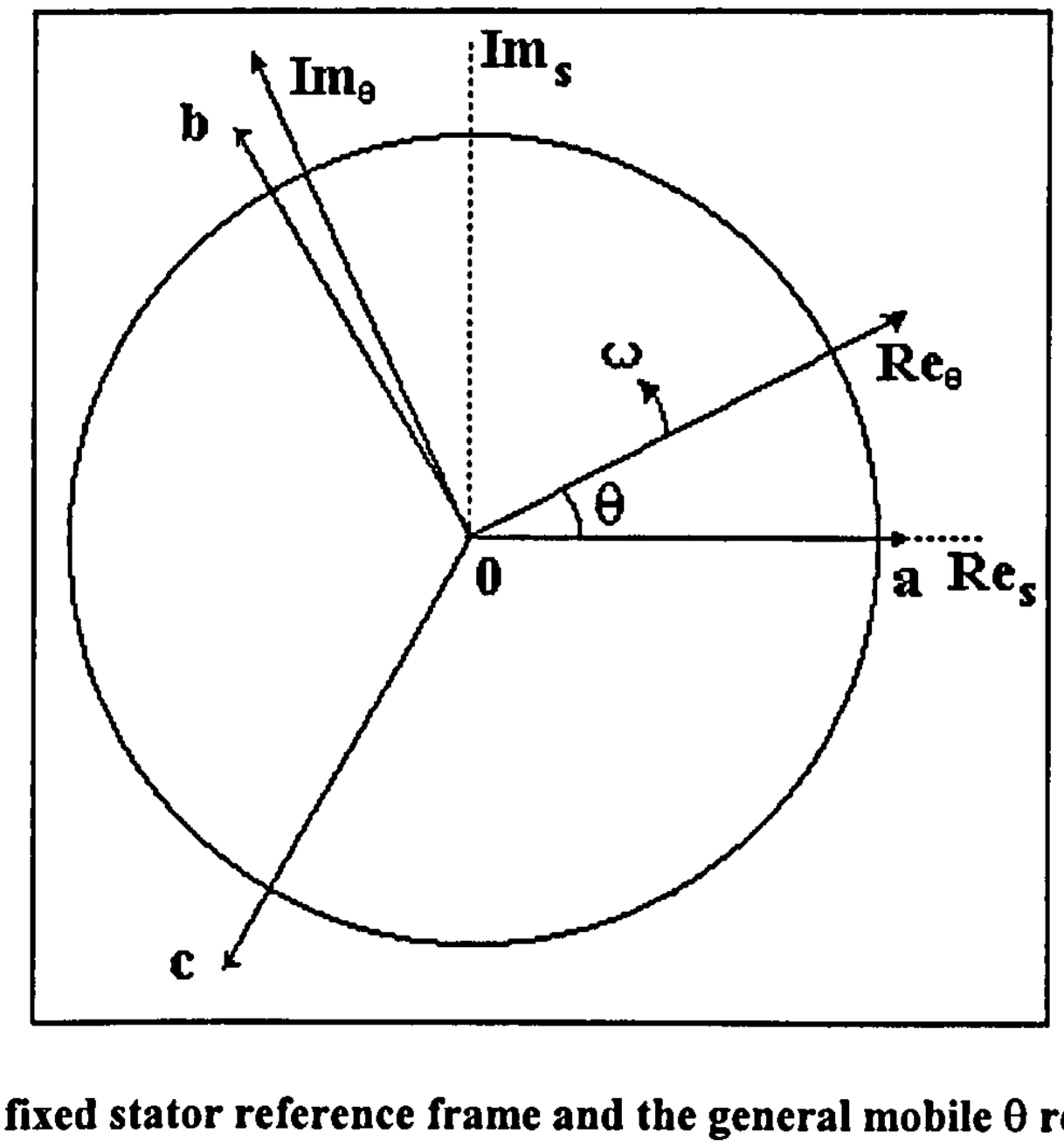

Fig. 2-2 - The fixed stator reference frame and the general mobile θ reference frame

The fourth equation in (2-11) can be rewritten as (2-12). Equation (2-13) is eventually obtained by dividing (2-12) with $e^{j\theta}$.

$$\underline{u}_s^\theta \cdot e^{j\theta} = R_s \underline{i}_s^\theta \cdot e^{j\theta} + \frac{d\underline{\Psi}_s^\theta}{dt} \cdot e^{j\theta} + \underline{\Psi}_s^\theta \cdot e^{j\theta} \cdot j\frac{d\theta}{dt} \qquad (2\text{-}12)$$

$$\underline{u}_s^\theta = R_s \underline{i}_s^\theta + \frac{d\underline{\Psi}_s^\theta}{dt} + j\omega_e \cdot \underline{\Psi}_s^\theta \qquad (2\text{-}13)$$

A similar complex equation describes the rotor circuit with the difference that the reference frame rotation speed relative to the rotor is ω_e-ω_{er} instead of ω_e (ω_{er} is the electrical rotor angular speed). Moreover, the rotor voltage is always zero for squirrel cage induction motors.

$$\underline{u}_r^{\theta} = R_s \underline{i}_r^{\theta} + \frac{d\underline{\Psi}_r^{\theta}}{dt} + j(\omega_e - \omega_{er}) \cdot \underline{\Psi}_r^{\theta} = 0 \tag{2-14}$$

Equations (2-15) describe the relation between the electrical stator angular frequency ω_{es} and the stator current frequency f_s on the one hand, and the relationship between the rotor angular speed ω_{er} and the rotor mechanical speed ω_r on the other hand. The variable 'p' is the number of pairs of stator poles.

$$\begin{cases} \omega_{es} = p \cdot \omega_s = p \cdot 2\pi f_s \\ \omega_{er} = p \cdot \omega_r \end{cases} \tag{2-15}$$

The individual phase fluxes that are used to calculate the magnetic flux vectors, are each composed of six components. The flux components are generated by the electromagnetic interaction between the three rotor windings and the three stator windings.

$$\begin{cases} \Psi_{sa} = \Psi_{sasa} + \Psi_{sasb} + \Psi_{sasc} + \Psi_{sara} + \Psi_{sarb} + \Psi_{sarc} \\ \Psi_{sb} = \Psi_{sbsa} + \Psi_{sbsb} + \Psi_{sbsc} + \Psi_{sbra} + \Psi_{sbrb} + \Psi_{sbrc} \\ \Psi_{sc} = \Psi_{scsa} + \Psi_{scsb} + \Psi_{scsc} + \Psi_{scra} + \Psi_{scrb} + \Psi_{scrc} \\ \Psi_{ra} = \Psi_{rasa} + \Psi_{rasb} + \Psi_{rasc} + \Psi_{rara} + \Psi_{rarb} + \Psi_{rarc} \\ \Psi_{rb} = \Psi_{rbsa} + \Psi_{rbsb} + \Psi_{rbsc} + \Psi_{rbra} + \Psi_{rbrb} + \Psi_{rbrc} \\ \Psi_{rc} = \Psi_{rcsa} + \Psi_{rcsb} + \Psi_{rcsc} + \Psi_{rcra} + \Psi_{rcrb} + \Psi_{rcrc} \end{cases} \tag{2-16}$$

In equation (2-16) each flux component is identified by four indices: the first two indicate the winding where the magnetic flux is measured while the last two indicate the winding that generates it. For instance, Ψ_{sarb} is the flux generated into stator winding 'a' by rotor winding 'b'.

The flux components related to stator phase 'a' are described by (2-17). The names and the significance of the symbols are as follows:

1) l_{msr} - the mutual inductance between stator and rotor. It is proportional to the flux created by one rotor phase into one stator phase.
2) $m_{\sigma s}$ – the stator mutual leakage inductance between two stator phases. It is proportional to the flux produced by one stator phase into another stator phase

without influencing the rotor. It therefore models the magnetic field lines that intersect two stator windings without intersecting the rotor.

3) l_{ms} - the mutual inductance between stator phases. It is proportional to the flux created by one stator phase into another stator phase through the rotor. It models the magnetic field lines that are created by one stator phase but intersect both the rotor and the other stator winding.

4) $l_{\sigma s}$ - the stator phase leakage inductance. It is proportional to the stator phase own leakage magnetic flux. The corresponding magnetic field lines do not intersect any winding other than the stator winding which produces them.

5) α - the angle between the stator d-axis and the rotor d-axis.

$$\begin{cases} \Psi_{sasa} = (l_{\sigma s} + l_{ms}) \cdot i_{sa} \\ \Psi_{sasb} = \left[m_{\sigma s} + l_{ms} \cdot \cos\left(\frac{2\pi}{3}\right) \right] \cdot i_{sb} \\ \Psi_{sasc} = \left[m_{\sigma s} + l_{ms} \cdot \cos\left(\frac{4\pi}{3}\right) \right] \cdot i_{sc} \\ \Psi_{sara} = (l_{msr} \cdot \cos\alpha) \cdot i_{ra} \\ \Psi_{sarb} = l_{msr} \cdot \cos\left(\alpha + \frac{2\pi}{3}\right) \cdot i_{rb} \\ \Psi_{sarc} = l_{msr} \cdot \cos\left(\alpha + \frac{4\pi}{3}\right) \cdot i_{rc} \end{cases} \quad (2\text{-}17)$$

The magnetic coupling between different windings is influenced by their relative position. The coupling is maximal when the angle between the two windings is zero and it is null at 90^0. This geometric factor can be expressed by simple cosine functions due to the assumption that the magnetic field has a sinusoidal distribution. Adding the six components from (2-17) the result is:

$$\Psi_{sa} = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} l_{ms} \right) \cdot i_{sa} + \frac{3}{2} l_{msr} \cdot \cos\alpha \cdot i_{ra} + \frac{\sqrt{3}}{2} l_{msr} \cdot \sin\alpha \cdot (i_{rc} - i_{rb}) \quad (2\text{-}18)$$

Equation (2-18) is obtained based on the property that the sum of the three phase currents is zero. Similar results are obtained for stator phases 'b' and 'c':

$$\begin{cases} \Psi_{sb} = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} l_{ms} \right) \cdot i_{sb} + \frac{3}{2} l_{msr} \cdot \cos\alpha \cdot i_{rb} + \frac{\sqrt{3}}{2} l_{msr} \cdot \sin\alpha \cdot (i_{ra} - i_{rc}) \\ \Psi_{sc} = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} l_{ms} \right) \cdot i_{sc} + \frac{3}{2} l_{msr} \cdot \cos\alpha \cdot i_{rc} + \frac{\sqrt{3}}{2} l_{msr} \cdot \sin\alpha \cdot (i_{rb} - i_{ra}) \end{cases} \quad (2\text{-}19)$$

Eventually the stator flux space vector is calculated multiplying equations (2-18) and (2-19) with 1, ε and ε^2 and adding them together. The flux has two components: one depends on the stator currents and the other depends on the rotor currents.

$$\begin{cases} \underline{\Psi}_s^s = \underline{\Psi}_{ss}^s + \underline{\Psi}_{sr}^s \\ \underline{\Psi}_{ss} = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} l_{ms} \right) \cdot \underline{i}_s^s \\ \underline{\Psi}_{sr} = \frac{3}{2} l_{msr} \cos\alpha \cdot \underline{i}_r + \frac{\sqrt{3}}{2} l_{msr} \sin\alpha \cdot \left(-i_{rb} + i_{rc} + \varepsilon i_{ra} - \varepsilon i_{rc} - \varepsilon^2 i_{ra} + \varepsilon^2 i_{rb} \right) \end{cases} \quad (2\text{-}20)$$

The expression describing the flux component Ψ_{sr} can be further transformed using the mathematical properties (2-21). The results are presented in (2-22), (2-23) and (2-24). Equation (2-24) becomes (2-25) in a general reference frame given by angle θ.

$$\begin{cases} \varepsilon - \varepsilon^2 = j \cdot \sqrt{3} \\ -1 + \varepsilon^2 = j \cdot \sqrt{3} \cdot \varepsilon \\ 1 - \varepsilon = j \cdot \sqrt{3} \cdot \varepsilon^2 \end{cases} \quad (2\text{-}21)$$

$$\underline{\Psi}_{sr}^s = \frac{3}{2} l_{msr} \cos\alpha \cdot \underline{i}_r + \frac{\sqrt{3}}{2} l_{msr} \sin\alpha \cdot j\sqrt{3} \cdot \left(i_{ra} + \varepsilon i_{rb} + \varepsilon^2 i_{rc} \right) \quad (2\text{-}22)$$

$$\underline{\Psi}_{sr}^s = \frac{3}{2} l_{msr} \cos\alpha \cdot \underline{i}_r^r + j \cdot \frac{3}{2} l_{msr} \sin\alpha \cdot \underline{i}_r^r = \frac{3}{2} l_{msr} \cdot \underline{i}_r^r \cdot e^{j\alpha} = \frac{3}{2} l_{msr} \cdot \underline{i}_r^s \quad (2\text{-}23)$$

$$\underline{\Psi}_s^s = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} \cdot l_{ms} \right) \cdot \underline{i}_s^s + \frac{3}{2} \cdot l_{msr} \cdot \underline{i}_r^s \quad (2\text{-}24)$$

$$\underline{\Psi}_s^\theta = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} \cdot l_{ms} \right) \cdot \underline{i}_s^\theta + \frac{3}{2} \cdot l_{msr} \cdot \underline{i}_r^\theta \quad (2\text{-}25)$$

The rotor flux expression is similar to the stator flux expression but each stator inductance is replaced by the corresponding rotor inductance. Thus, induction motor equations, formulated for a reference frame defined by the angle θ(t) and the rotation speed ω(t), are:

$$\begin{cases} \underline{u_s^\theta} = R_s \underline{i_s^\theta} + \dfrac{d\underline{\Psi_s^\theta}}{dt} + j\omega_e \underline{\Psi_s^\theta} \\ \underline{u_r^\theta} = R_r \underline{i_r^\theta} + \dfrac{d\underline{\Psi_r^\theta}}{dt} + j(\omega_e - \omega_{er})\underline{\Psi_r^\theta} \\ \underline{\Psi_s^\theta} = \left(l_{\sigma s} - m_{\sigma s} + \dfrac{3}{2}\cdot l_{ms}\right)\cdot \underline{i_s^\theta} + \dfrac{3}{2}\cdot l_{msr}\cdot \underline{i_r^\theta} \\ \underline{\Psi_r^\theta} = \left(l_{\sigma r} - m_{\sigma r} + \dfrac{3}{2}\cdot l_{mr}\right)\cdot \underline{i_r^\theta} + \dfrac{3}{2}\cdot l_{msr}\cdot \underline{i_s^\theta} \end{cases} \quad (2\text{-}26)$$

The magnetic flux expressions in (2-26) are complicated because seven different inductances are involved. The mathematical technique of referring the rotor quantities to the stator is usually applied to the equations given in (2-26) in order to simplify the flux equations. The basic principle of referring the rotor quantities to the stator is to multiply rotor quantities with constant values in such a manner that the power transfer between stator and rotor is not altered. Thus, if the rotor current is multiplied by constant k then the rotor voltage and the rotor flux are multiplied by 1/k. On the other hand, the rotor resistance and the rotor inductance are multiplied by $1/k^2$. The constant k that generates the simplest transformation of system (2-26) is given by (2-27) while the corresponding referred rotor quantities are (2-28). The equation linking all the referred quantities is (2-29).

$$k = \frac{l_{msr}}{l_{ms}} \quad (2\text{-}27)$$

$$\begin{cases} \underline{i_r'^\theta} = k\cdot \underline{i_r^\theta} = \dfrac{l_{msr}}{l_{ms}}\cdot \underline{i_r^\theta} \\ \underline{u_r'^\theta} = \dfrac{1}{k}\cdot \underline{u_r^\theta} = \dfrac{l_{ms}}{l_{msr}}\cdot \underline{u_r^\theta} \\ \underline{\Psi_r'^\theta} = \dfrac{1}{k}\cdot \underline{\Psi_r^\theta} = \dfrac{l_{ms}}{l_{msr}}\cdot \underline{\Psi_r^\theta} \\ R_r' = \dfrac{1}{k^2}\cdot R_r = \dfrac{l_{ms}^2}{l_{msr}^2}\cdot R_r \end{cases} \quad (2\text{-}28)$$

$$\underline{u_r'^\theta} = R_r \underline{i_r'^\theta} + \frac{d\underline{\Psi_r'^\theta}}{dt} + j(\omega_e - \omega_{er})\underline{\Psi_r'^\theta} \quad (2\text{-}29)$$

The referred rotor flux can be expressed as a function of the stator current and the referred rotor current vector (see (2-30)).

$$\underline{\Psi_r'^\theta} = \frac{l_{ms}}{l_{msr}}\underline{\Psi_r^\theta} = \frac{l_{ms}^2}{l_{msr}^2}\left(l_{\sigma r} - m_{\sigma r} + \frac{3}{2}l_{mr}\right)\cdot \underline{i_r'^\theta} + \frac{3}{2}l_{ms}\cdot \underline{i_s^\theta} \quad (2\text{-}30)$$

The inductances l_{ms}, l_{mr} and l_{msr} are always related by equation (2-31). This relationship allows the rewriting of equation (2-30) as (2-32) and (2-33).

$$l_{ms} \cdot l_{mr} = l_{msr}^2 \tag{2-31}$$

$$\underline{\Psi}_r^{'\theta} = \frac{l_{ms}^2}{l_{msr}^2}\left(l_{\sigma r} - m_{\sigma r}\right) \cdot \underline{i}_r^{'\theta} + \frac{3}{2} l_{ms} \cdot \underline{i}_r^{'\theta} + \frac{3}{2} l_{ms} \cdot \underline{i}_s^{\theta} \tag{2-32}$$

$$\underline{\Psi}_r^{'\theta} = \left(L'_{\sigma r} + L_m\right) \cdot \underline{i}_r^{'\theta} + L_m \cdot \underline{i}_s^{\theta} = L'_r \cdot \underline{i}_r^{'\theta} + L_m \cdot \underline{i}_s^{\theta} \tag{2-33}$$

The significance of the symbols in the previous equations is:

1) $L'_{\sigma r} = \frac{l_{ms}^2}{l_{msr}^2}\left(l_{\sigma r} - m_{\sigma r}\right)$ – the total referred rotor leakage inductance

2) $L_m = \frac{3}{2} l_{ms}$ – the resulting stator-rotor mutual inductance

3) $L'_r = L'_{\sigma r} + L_m$ – the total referred rotor inductance

Substituting the first equation (2-28) in (2-25), the stator flux can be written as:

$$\underline{\Psi}_s^{\theta} = \left(l_{\sigma s} - m_{\sigma s} + \frac{3}{2} \cdot l_{ms}\right) \cdot \underline{i}_s^{\theta} + \frac{3}{2} \cdot l_{ms} \cdot \underline{i}_r^{'\theta} \tag{2-34}$$

$$\underline{\Psi}_s^{\theta} = \left(L_{\sigma r} + L_m\right) \cdot \underline{i}_r^{\theta} + L_m \cdot \underline{i}_s^{\theta} = L_s \cdot \underline{i}_s^{\theta} + L_m \cdot \underline{i}_r^{'\theta} \tag{2-35}$$

The significance of the symbols is:

1) $L_{\sigma s} = l_{\sigma s} - m_{\sigma s}$ – the total stator leakage inductance

2) $L_s = L_{\sigma s} + L_m$ – the total stator inductance

Thus, (2-36) is the compact format of the induction motor equations initially presented in (2-26). This system of equations expresses the space vector model of the induction motor [93].

$$\begin{cases} \underline{u}_s^{\theta} = R_s \underline{i}_s^{\theta} + \dfrac{d\underline{\Psi}_s^{\theta}}{dt} + j\omega \underline{\Psi}_s^{\theta} \\ \underline{u}_r^{'\theta} = R'_r \underline{i}_r^{'\theta} + \dfrac{d\underline{\Psi}_r^{'\theta}}{dt} + j(\omega_e - \omega_{er}) \underline{\Psi}_r^{'\theta} = 0 \\ \underline{\Psi}_s^{\theta} = L_s \underline{i}_s^{\theta} + L_m \underline{i}_r^{'\theta} \\ \underline{\Psi}_r^{'\theta} = L'_r \underline{i}_r^{'\theta} + L_m \underline{i}_s^{\theta} \end{cases} \tag{2-36}$$

NOTE: Usually, to simplify the notation, the apostrophe symbols are not included in the equations. Yet, the rotor quantities are implicitly referred to the stator. No

apostrophe symbol is used in the rest of this thesis but they are implied whenever a rotor equation or a rotor parameter is mentioned.

2.3 INDUCTION MOTOR CONTROL STRATEGIES

During the first one hundred years after its invention, the induction motor was known as a constant speed electrical machine. The advent of electrical power converters in the 1960's made possible the use of the induction motor as a variable speed machine. The recent development of the digital technology created the possibility of implementing complex control algorithms yielding high dynamic performance [134].

Correct control over the motor torque is a prerequisite of all the speed control strategies. The torque equation can be derived from power-based considerations and can be expressed as a function of the current and voltage space vectors. The total power consumed by the motor has three components: the power dissipated by the winding resistances P_R, the power stored in the internal magnetic fields P_μ and the mechanical power P_M. The motor torque is proportional to the mechanical power and inversely proportional to the rotor speed (2-37). The total motor power is the power consumed by all six stator and rotor windings so it can be calculated as in equation (2-38). Elementary algebraic calculations show that the rotor power and the stator power can be calculated as indicated by (2-39). The calculations can be performed in any reference frame defined by the time function $\theta(t)$.

Now as

$$\begin{cases} P = P_R + P_\mu + P_M \\ T = \dfrac{P_M}{\omega_r} = p \cdot \dfrac{P_M}{\omega_{er}} \end{cases} \tag{2-37}$$

and therefore

$$P = P_s + P_r = u_{sa} i_{sa} + u_{sb} i_{sb} + u_{sc} i_{sc} + u_{ra} i_{ra} + u_{rb} i_{sb} + u_{rc} i_{sc} \tag{2-38}$$

where

$$\begin{cases} P_s = u_{sa} i_{sa} + u_{sb} i_{sb} + u_{sc} i_{sc} = \dfrac{2}{3} \mathrm{Re}\left\{ \underline{u}_s^s \cdot \underline{i}_s^{s*} \right\} = \dfrac{2}{3} \mathrm{Re}\left\{ \underline{u}_s^\theta \cdot \underline{i}_s^{\theta*} \right\} \\ P_r = u_{ra} i_{ra} + u_{rb} i_{rb} + u_{rc} i_{rc} = \dfrac{2}{3} \mathrm{Re}\left\{ \underline{u}_r^r \cdot \underline{i}_r^{r*} \right\} = \dfrac{2}{3} \mathrm{Re}\left\{ \underline{u}_r^\theta \cdot \underline{i}_r^{\theta*} \right\} \end{cases} \tag{2-39}$$

The equations (2-40) are obtained by substituting general equations (2-36) into (2-39). Thus, the three power components are calculated according to (2-41).

$$\begin{cases} P_s = \frac{2}{3}\mathrm{Re}\left\{\left[R_s \underline{i}_s^{\theta} + \frac{d\underline{\Psi}_s^{\theta}}{dt} + j\omega_e \cdot \underline{\Psi}_s^{\theta}\right] \cdot \underline{i}_s^{\theta *}\right\} \\ P_r = \frac{2}{3}\mathrm{Re}\left\{\left[R_r \underline{i}_r^{\theta} + \frac{d\underline{\Psi}_r^{\theta}}{dt} + j(\omega_e - \omega_{er}) \cdot \underline{\Psi}_r^{\theta}\right] \cdot \underline{i}_r^{\theta *}\right\} \end{cases} \quad (2\text{-}40)$$

$$\begin{cases} P_R = \frac{2}{3}\mathrm{Re}\left\{R_s \underline{i}_s^{\theta} \cdot \underline{i}_s^{\theta *} + R_r \underline{i}_r^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \\ P_\mu = \frac{2}{3}\mathrm{Re}\left\{\frac{d\underline{\Psi}_s^{\theta}}{dt} \cdot \underline{i}_s^{\theta *} + \frac{d\underline{\Psi}_r^{\theta}}{dt} \cdot \underline{i}_r^{\theta *}\right\} \\ P_M = \frac{2}{3}\mathrm{Re}\left\{j\omega_e \cdot \underline{\Psi}_s^{\theta} \cdot \underline{i}_s^{\theta *} + j(\omega_e - \omega_{er}) \cdot \underline{\Psi}_r^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \end{cases} \quad (2\text{-}41)$$

The imaginary number 'j' in the expression of the mechanical power component P_M can be eliminated using the general algebraic property (2-42).

$$\mathrm{Re}\{j \cdot z\} = -\mathrm{Im}\{z\} \quad (2\text{-}42)$$

Thus, the equation defining P_M is simplified as

$$P_M = -\frac{2}{3}\mathrm{Im}\left\{\omega_e \cdot \underline{\Psi}_s^{\theta} \cdot \underline{i}_s^{\theta *} + (\omega_e - \omega_{er}) \cdot \underline{\Psi}_r^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \quad (2\text{-}43)$$

The two components of the imaginary part in (2-43) can be rewritten as in (2-44), so that the mechanical power equation becomes (2-45).

$$\begin{cases} \mathrm{Im}\left\{\omega_e \cdot \underline{\Psi}_s^{\theta} \cdot \underline{i}_s^{\theta *}\right\} = \mathrm{Im}\left\{\omega_e \cdot \left(L_s \underline{i}_s^{\theta} + L_m \underline{i}_r^{\theta}\right) \cdot \underline{i}_s^{\theta *}\right\} = \omega_e L_m \cdot \mathrm{Im}\left\{\underline{i}_r^{\theta} \cdot \underline{i}_s^{\theta *}\right\} \\ \mathrm{Im}\left\{(\omega_e - \omega_{er}) \cdot \underline{\Psi}_r^{\theta} \cdot \underline{i}_r^{\theta *}\right\} = (\omega_e - \omega_{er}) L_m \cdot \mathrm{Im}\left\{\underline{i}_s^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \end{cases} \quad (2\text{-}44)$$

$$P_M = -\frac{2}{3} L_m \left[\omega_e \cdot \mathrm{Im}\left\{\underline{i}_r^{\theta} \cdot \underline{i}_s^{\theta *}\right\} + (\omega_e - \omega_{er}) \cdot \mathrm{Im}\left\{\underline{i}_s^{\theta} \cdot \underline{i}_r^{\theta *}\right\}\right] \quad (2\text{-}45)$$

Based on the mathematical property (2-46) the equation (2-45) is further transformed into (2-47).

$$\mathrm{Im}\left\{\underline{x} \cdot \underline{y}^*\right\} + \mathrm{Im}\left\{\underline{y} \cdot \underline{x}^*\right\} = 0 \quad (2\text{-}46)$$

$$P_M = \frac{2}{3}\omega_{er} L_m \cdot \mathrm{Im}\left\{\underline{i}_s^{\theta} \cdot \underline{i}_r^{\theta *}\right\} = \frac{2}{3} p\omega_r L_m \cdot \mathrm{Im}\left\{\underline{i}_s^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \quad (2\text{-}47)$$

Therefore, the motor torque may be expressed by (2-48). It is seen that the motor torque depends only on the rotor current vector and on the stator current vector.

$$T = \frac{2}{3} p L_m \cdot \mathrm{Im}\left\{\underline{i}_s^{\theta} \cdot \underline{i}_r^{\theta *}\right\} \quad (2\text{-}48)$$

Alternatively, the torque can be expressed as equivalent functions of the stator magnetic flux and/or the rotor magnetic flux as shown in (2-49), where δ is the angle between the stator flux and the rotor flux.

$$\begin{cases} T = \frac{2}{3} p \cdot \mathrm{Im}\left\{ \underline{\Psi}_r^{\theta} \cdot \underline{i}_r^{\theta *} \right\} & \text{(a)} \\ T = \frac{2}{3} p \cdot \mathrm{Im}\left\{ \underline{i}_s^{\theta} \cdot \underline{\Psi}_s^{\theta *} \right\} & \text{(b)} \\ T = \frac{2}{3} p \cdot \frac{L_m}{L_s L_r - L_m^2} \cdot \mathrm{Im}\left\{ \underline{\Psi}_s^{\theta} \cdot \underline{\Psi}_r^{\theta *} \right\} = \frac{2}{3} p \cdot \frac{L_m}{L_s L_r - L_m^2} \cdot \left| \underline{\Psi}_s^{\theta} \right| \cdot \left| \underline{\Psi}_r^{\theta} \right| \cdot \sin \delta & \text{(c)} \end{cases} \quad \textbf{(2-49)}$$

Relations (2-48) and (2-49) directly or indirectly underlie all induction motor control strategies. They can be classified as scalar control and vector control strategies (Fig. 2-3). The scalar control operates utilising simplified equations derived from the general space vector model (2-36). This approach involves only the space vector amplitudes and their corresponding frequencies and the simplified equations are valid only in steady state operation. Consequently, scalar control is simple but generates poor response during transient operation [132]. In contrast, vector control operates directly with the space-vector model of the motor and implements the equations given in (2-49). Therefore, it offers good results in both steady-state operation and transient operation. The group of vector control algorithms includes the Direct Torque Control (DTC) method and the class of field oriented control strategies. The theory of field oriented control was developed by researchers at Siemens in 1968-1969. Since this time, researchers all over the world have implemented increasingly efficient practical systems based on this theory [134].

The actual motor speed is the most important information for any speed control algorithm. As illustrated in Fig. 2-3, there are two possible approaches to obtain this measure: either to use a speed sensor or to calculate the speed based on the electrical motor quantities. These two approaches are applicable to scalar control methods as well as to vector control methods but the use of vector control ensures better dynamic response. The interest in speed sensorless control emerged from practical applications where high control quality is required but the speed sensor is either difficult to use due to technical reasons, or too expensive. The speed sensorless control of the induction motor is currently one of the most intensively research fields in electrical drives [133].

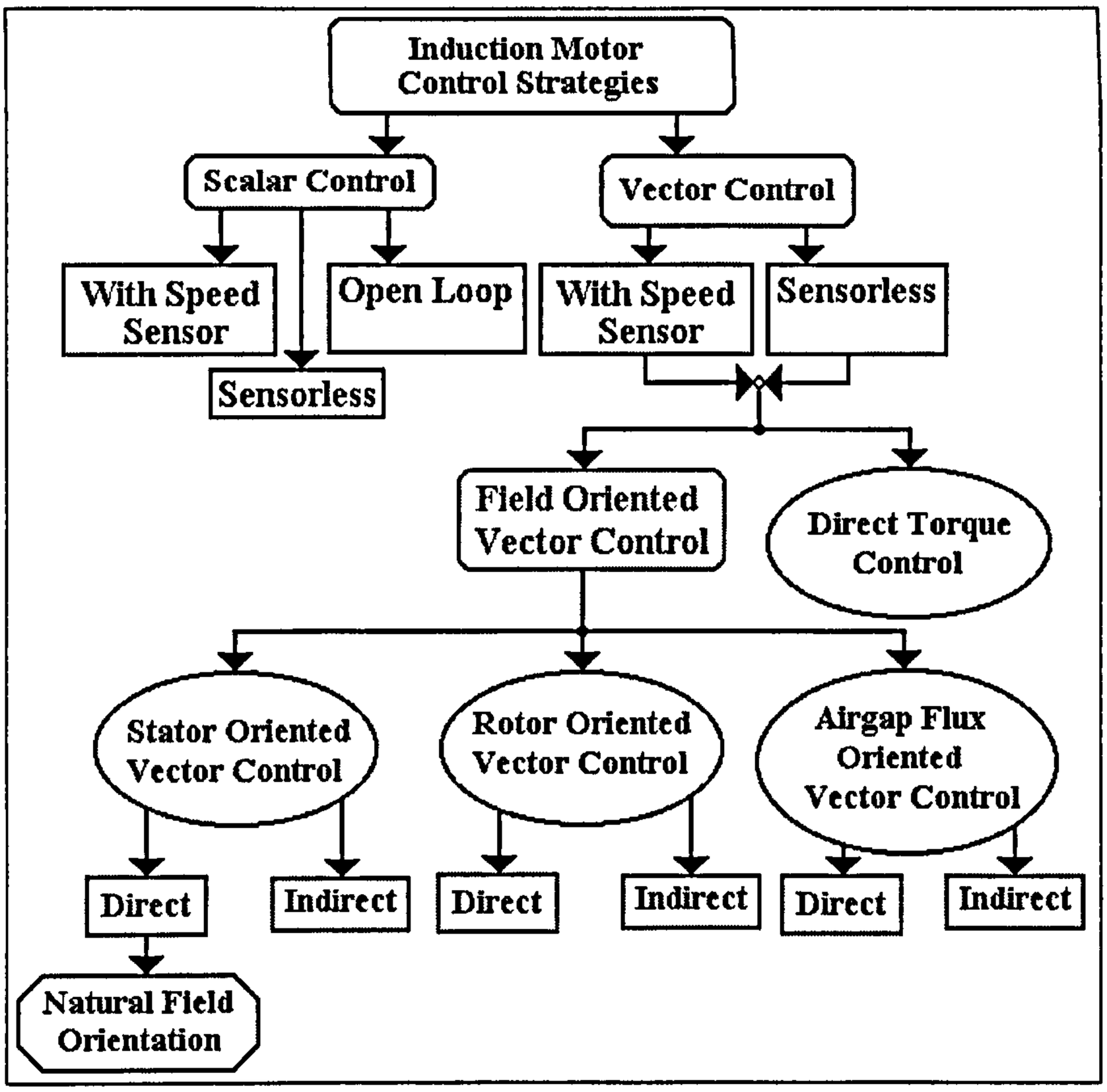

Fig. 2-3 – The classification of the induction control strategies

2.3.1 Scalar Control

Scalar control uses the stator voltage amplitude $U_s=2/3\cdot|\underline{u}_s|$ and the stator frequency f_s as input quantities and works well in steady state and slow transient operation. This strategy varies the stator voltage and the stator frequency according to a function $U_s(f_s)$ so that the maximum torque available is large (and almost constant) at any stator angular frequency ω_{es}.

In steady-state operation, the rotor flux has constant amplitude. Therefore, the rotor equation in rotor co-ordinates is:

$$R_r \underline{i}_r^r + \frac{d\underline{\Psi}_r^r}{dt} = R_r \underline{i}_r^r + j(\omega_{es} - \omega_{er})\underline{\Psi}_r^r = R_r \underline{i}_r^r + j(\omega_{es} - \omega_{er})\cdot\left(L_r \underline{i}_r^r + L_m \underline{i}_s^r\right) = 0 \qquad (2\text{-}50)$$

Under these conditions, the rotor current depends on the stator current space vector and on the slip angular frequency (the difference ω_{es}-ω_{er}) as indicated in (2-51).

$$\underline{i}_r^r = \frac{-j(\omega_{es} - \omega_{er})L_m \underline{i}_s^r}{R_r + j(\omega_{es} - \omega_{er})L_r} \qquad (2\text{-}51)$$

The initial motor torque expression (2-48) can be modified by substituting (2-51) in (2-48) which yields equation (2-52). Therefore, the motor torque is proportional to the stator current module squared.

$$T=\frac{2}{3}pL_m^2\cdot Im\left\{\frac{j(\omega_{es}-\omega_{er})\underline{i}_s^r\cdot\underline{i}_s^{r*}}{R_r+j(\omega_{es}-\omega_{er})L_r}\right\}=\frac{2}{3}pL_m^2\left|\underline{i}_s^r\right|^2\cdot Im\left\{\frac{j(\omega_{es}-\omega_{er})}{R_r+j(\omega_{es}-\omega_{er})L_r}\right\} \quad (2\text{-}52)$$

It is seen that the stator current depends on the stator voltage as indicated by (2-53), (2-54), (2-55), while the dependency between the torque and the stator voltage is obtained by combining equations (2-52) and (2-55) to give relationship (2-56).

$$\underline{u}_s^r=R_s\underline{i}_s^r+j\omega_{er}\underline{\Psi}_s^r+j(\omega_{es}-\omega_{er})\underline{\Psi}_s^r=R_s\underline{i}_s^r+j\omega_{es}\underline{\Psi}_s^r \quad (2\text{-}53)$$

$$\underline{u}_s^r=R_s\underline{i}_s^r+j\omega_{es}\left[L_s\underline{i}_s^r-\frac{j(\omega_{es}-\omega_{er})L_m^2\cdot\underline{i}_s^r}{R_r+j(\omega_{es}-\omega_{er})L_r}\right] \quad (2\text{-}54)$$

$$\left|\underline{i}_s^r\right|=\frac{\left|\underline{u}_s^r\right|}{\left|R_s+j\omega_{es}L_s+\frac{\omega_{es}(\omega_{es}-\omega_{er})L_m^2}{R_r+j(\omega_{es}-\omega_{er})L_r}\right|} \quad (2\text{-}55)$$

$$T=\frac{3}{2}pL_m^2\cdot\frac{U_s^2}{\left|R_s+j\omega_{es}L_s+\frac{\omega_{es}(\omega_{es}-\omega_{er})L_m^2}{R_r+j(\omega_{es}-\omega_{er})L_r}\right|^2}\cdot Im\left\{\frac{j(\omega_{es}-\omega_{er})L_m^2}{R_r+j(\omega_{es}-\omega_{er})L_r}\right\} \quad (2\text{-}56)$$

Fig. 2-4 presents the torque-speed characteristic calculated according to (2-56) for a three-phase induction motor with the parameters R_s=0.371Ω; R_r=0.415Ω; $L_{s\sigma}$=2.72mH; $L_{r\sigma}$=3.3mH; L_m=84.33mH; p=1; P=11.1kW. The motor is supplied by a three-phase 240V/50Hz supply. As the figure shows, the motor torque is zero at synchronous speed and has its maximum at a relatively high angular speed ω_M as compared to the rated stator angular frequency (314rad/s). The motor normally operates at speeds between the synchronous angular speed and ω_M.

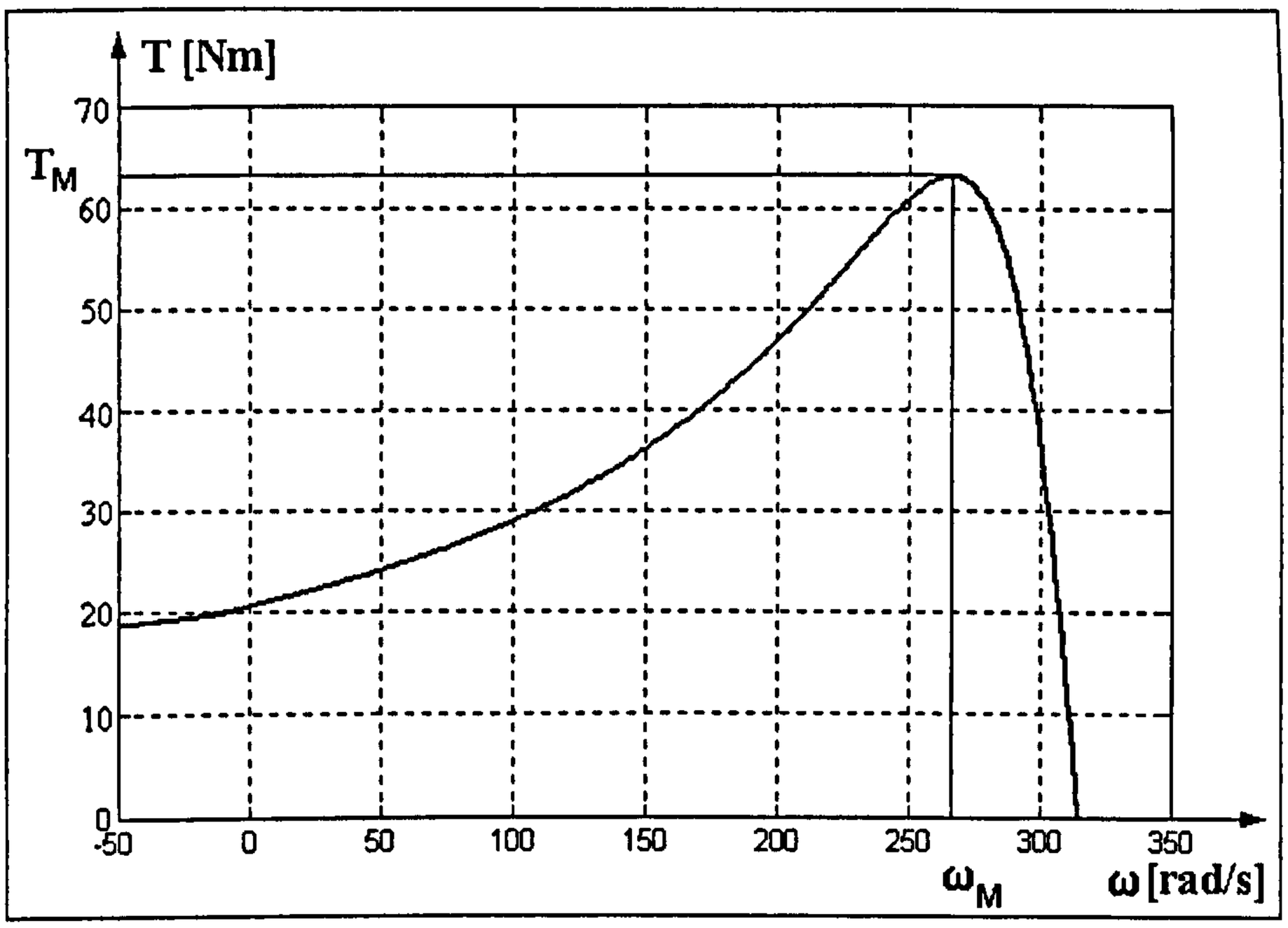

Fig. 2-4 – Induction motor mechanical characteristic (P=11.1 kW)

At high stator angular frequency, around the rated value, the stator resistance is negligible, thus, $|\underline{i}_s^r|$ in (2-55) depends only on the slip angular frequency ($\omega_{slp}=\omega_{es}-\omega_{er}$) and on the voltage-angular frequency ratio (U_s/ω_s). If this ratio is kept constant then the stator current amplitude and the motor torque depend solely on the slip angular frequency. Therefore, the maximum motor torque T_M is independent of the stator angular frequency ω_{es}. At low frequencies however, the stator resistance has an important influence on the stator current and leads to a diminished maximum torque, with negative effects on the motor operation. The effect of the stator resistance on the motor torque can be counteracted by raising the stator voltage to compensate for the stator resistance. The function $U_s(\omega_s)$ that maintains T_M constant at all frequencies can be derived from (2-56). The solution is a non-linear expression, difficult to implement into hardware. A linear approximation of this function is usually adopted in practical situations. The linear approximation $U_s(\omega_s)$ is defined by two points corresponding to the zero stator frequency and to the rated stator frequency:

1. At zero stator frequency, the stator voltage has to generate a current equal to the stator current at rated stator angular frequency (314 rad/s) and maximum torque.
2. At the rated stator frequency, the voltage attains its rated value.

The stator voltage amplitude is therefore defined by (2-57) where 'p' is the number of stator pole pairs. This approximate solution does not provide a perfectly constant T_{max} but restricts its variation within a narrow interval.

$$U_s = R_s I_{s(max\,T)} + \omega_{es} \frac{U_{sMAX} - R_s I_{s(max\,T)}}{p \cdot 2\pi \cdot 50} = U_{s0} + \Phi \cdot f_s \qquad (2\text{-}57)$$

Speeds over the rated value can be obtained by increasing the stator frequency over the 50 Hz limit but in this case the voltage is maintained constant at its maximum value U_{sMAX}. As a result, the maximum available torque decreases (it is inversely proportional to the frequency squared) and very high speeds cannot be obtained using this method. For instance, the maximum torque decreases to as much as 25% from the rated value if the stator frequency is 100Hz.

The open-loop scalar control implements the strategy illustrated by (2-57). This offers an approximate control over the motor speed but the effects of the load torque variations cannot be compensated for due to the lack of any feedback information. A compensation of the average slip angular frequency can be performed instead so that the rotor speed equals the reference speed for the most frequent load torque value.

The control scheme can be implemented with a controlled rectifier as presented in Fig. 2-5, or with an uncontrolled rectifier. In the first case, the PWM inverter controls only the frequency of the output voltage, while the rectifier determines the output voltage amplitude. In the second case, the switching pattern inside the inverter is more complex and determines both the frequency and the amplitude of the output voltage.

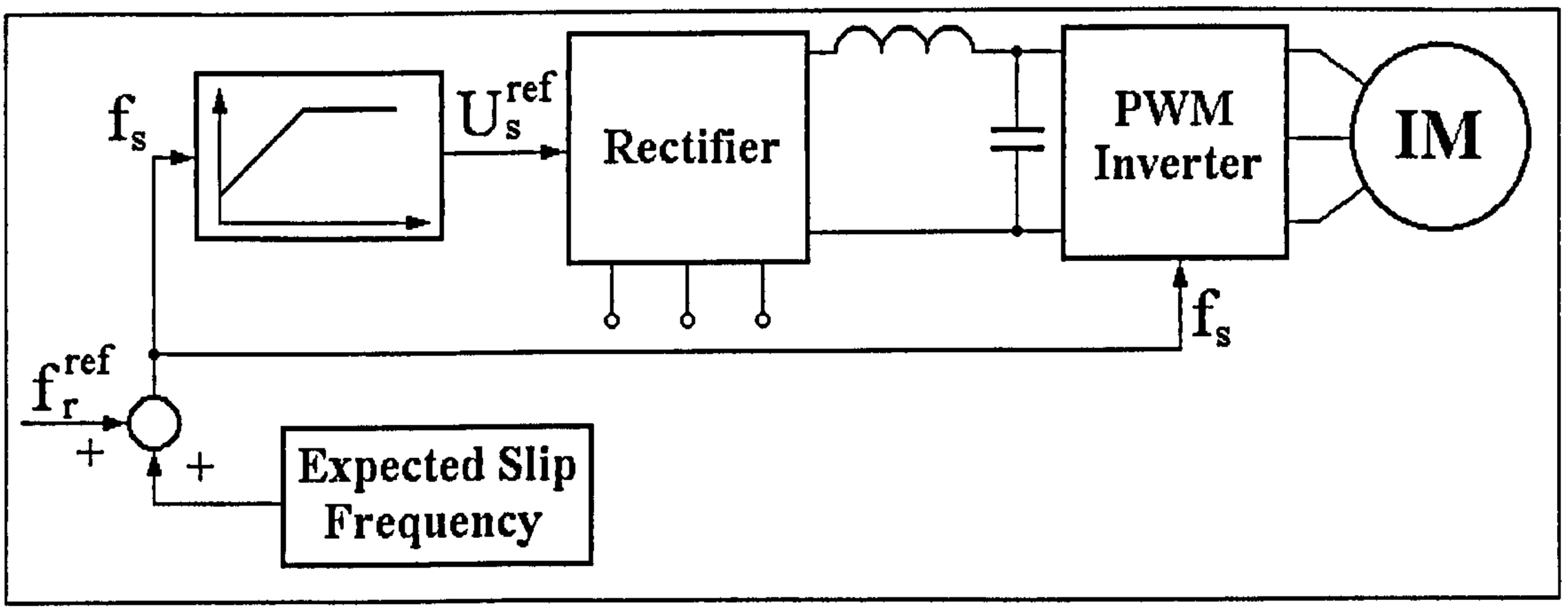

Fig. 2-5 – Open Loop Scalar Control Scheme

The scalar control strategy with speed sensor can be implemented as in Fig. 2-6 using a controlled rectifier and a PWM inverter. As in the previous section, the controlled rectifier can be replaced by an uncontrolled rectifier if the inverter controls both the frequency and the amplitude of the output voltage. The voltage control loop modifies the DC voltage according to the required speed profile while the optimal slip frequency is calculated as a function of the current absorbed by the motor: the slip

increases with the absorbed current. This type of slip-current correlation limits the current variations in the DC link during the transient motor operation.

An increase of the resistive load increases the current absorbed by the motor and decreases its speed. This lowers the DC link voltage. The speed controller responds by increasing the reference voltage while the slip calculator increases the motor slip. As demonstrated by equation (2-56) the motor torque increases with the increase of the stator voltage and with the increase of the slip angular frequency. On the other hand, the stator current depends on the stator voltage in the manner indicated in (2-55). Therefore, a torque increase can be obtained with a diminished current change if the slip angular frequency is changed accordingly. Conversely, when the load torque decreases the current drop in the DC link is limited and the temporary transformation of the motor into a generator is avoided, thereby reducing the strain on the power transistors in the PWM inverter.

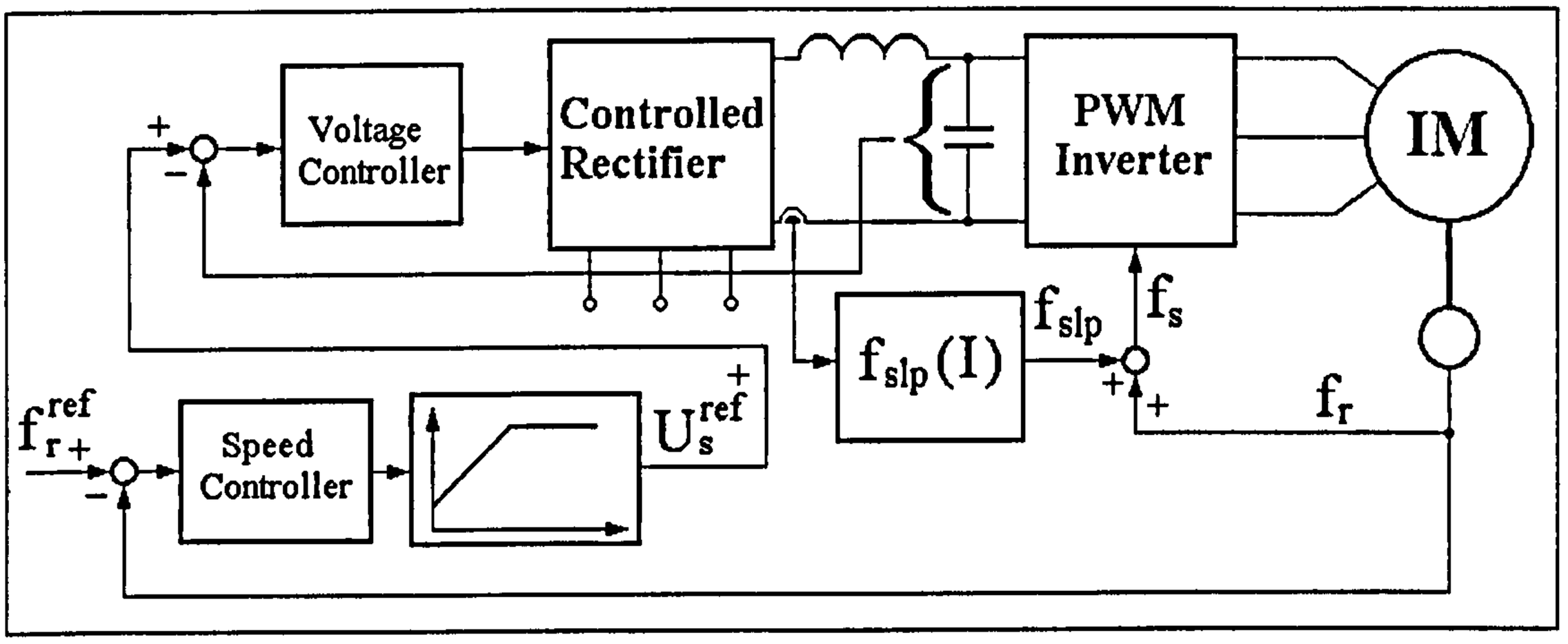

Fig. 2-6 –Scalar control scheme with speed sensor

The sensorless scalar control strategy is based on the possibility of calculating the slip frequency as a function of the stator frequency and the current in the DC link between the rectifier and the PWM inverter [109]. The equation underlying the slip angular frequency calculation can be derived from (2-55). The stator angular frequency is determined as the sum of the slip angular frequency and the calculated rotor angular speed corresponding to the actual voltage across the DC link. In general, the large DC link capacitor prevents the amplitude of the AC voltage from being increased as rapidly as the frequency, which is developed with practically no delay by simply feeding the right triggering pulses to the inverter transistors. Hence, it is customary to calculate the frequency control to the voltage control loop in the manner shown in Fig. 2-7 to prevent the motor from ever receiving the inappropriate voltage-frequency ratio.

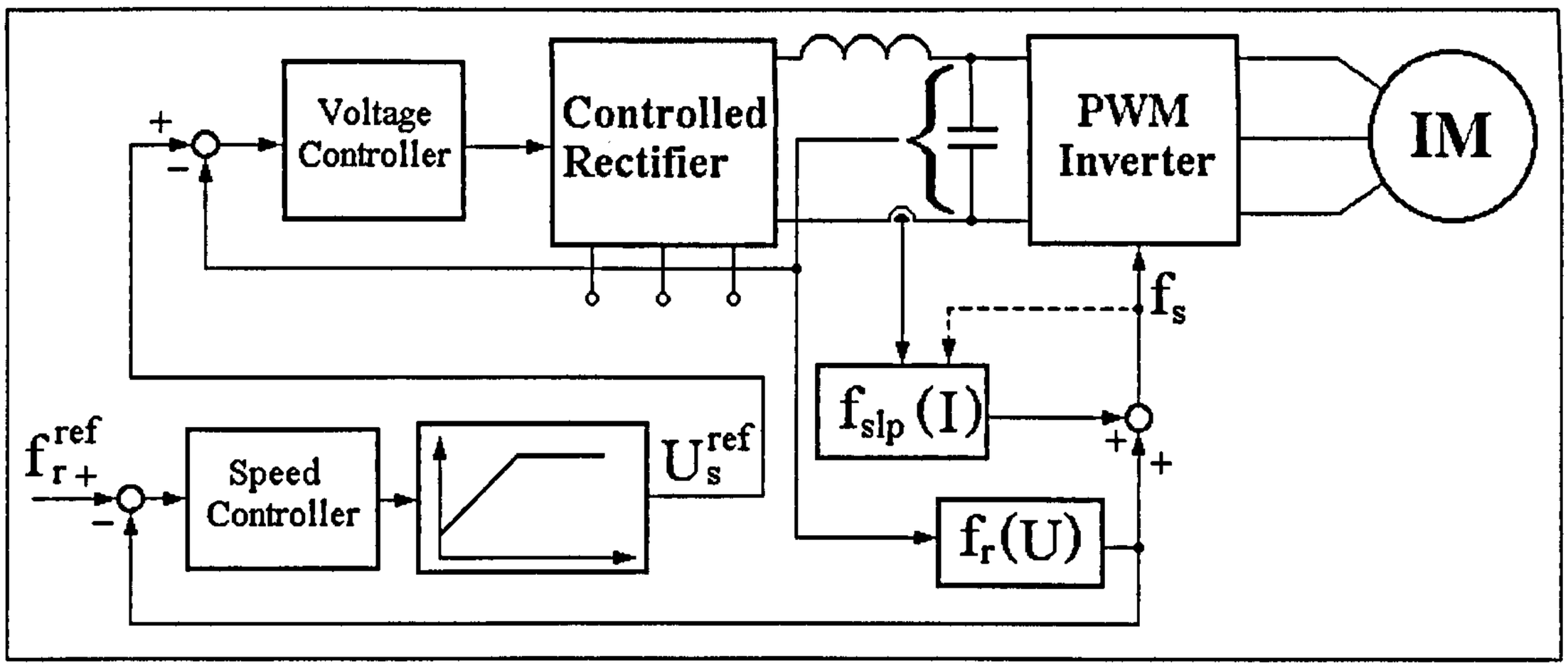

Fig. 2-7 – Sensorless Scalar Control Scheme

Of the two control parameters, frequency control is by far the most sensitive as small changes in frequency produce large changes of slip frequency and hence large changes in current and torque. By slaving the frequency command to the DC bus voltage, the rate of frequency change is generally limited to a value to which the motor can respond without drawing excessive current or without regenerating.

2.3.2 Vector Control

Vector control strategies use the space vector model of the induction motor to accurately control the speed and torque both in steady-state operation and in fast transient operation. The dynamic performance achieved by vector control strategies equals the dynamic performance offered by DC motor drives. In fact, with vector control, induction motor drives outperform DC drives because of higher transient current capability, increased speed range, and lower rotor inertia [33].

The class of vector control strategies encompasses field oriented control methods and direct torque control methods. Field oriented control methods use the rotor oriented reference frame, the airgap oriented reference frame or the stator oriented reference frame (see Fig. 2-3). In each case, the reference frame real axis (axis 'd') is oriented along the direction indicated by the corresponding magnetic flux. The rotor oriented vector control simplifies the control system structure and generates very fast transient response. However, systems working with the stator flux vector or with the airgap flux vector have been successfully implemented as well [44], [32].

2.3.2.1 Rotor Flux Orientation

In the rotor flux oriented reference frame, the rotor flux vector has no imaginary part so that the torque expression (2-49-a) can be written as (2-58). The rotor flux and the rotor current depend to one another in the manner indicated by the equations in (2-59).

$$T = \frac{2}{3} p \cdot \mathrm{Re}\left\{\underline{\Psi}_r^{\theta}\right\} \cdot \mathrm{Im}\left\{\underline{i}_r^{\theta^*}\right\} = -\frac{2}{3} p \cdot \Psi_{rd} \cdot i_{rq} \tag{2-58}$$

$$\begin{cases} \Psi_{rd} = L_r i_{rd} + L_m i_{sd} \\ \Psi_{rq} = L_r i_{rq} + L_m i_{sq} = 0 \\ R_r i_{rd} + \dfrac{d\Psi_{rd}}{dt} - \left(\omega_{e\Psi_r} - \omega_{er}\right)\Psi_{rq} = R_r i_{rd} + \dfrac{d\Psi_{rd}}{dt} = 0 \\ R_r i_{rq} + \dfrac{d\Psi_{rq}}{dt} + \left(\omega_{e\Psi_r} - \omega_{er}\right)\Psi_{rd} = R_r i_{rq} + \left(\omega_{e\Psi_r} - \omega_{er}\right)\Psi_{rd} = 0 \end{cases} \tag{2-59}$$

Equations (2-60) and (2-61) can be derived from the previous system. They illustrate the influence of the stator current components over the rotor flux and on the rotor current component on axis 'q' (i_{rq}). Thus, the modification speed of the rotor flux is limited by the rotor time constant $T_r=L_r/R_r$, while the rotor current component i_{rq} can be changed rapidly as no time constant is involved in (2-61).

$$\frac{L_r}{R_r}\frac{d\Psi_{rd}}{dt} + \Psi_{rd} = L_m i_{sd} \tag{2-60}$$

$$i_{rq} = -\frac{L_m}{L_r} \cdot i_{sq} \tag{2-61}$$

As demonstrated by (2-60) and (2-61), the two quantities influencing the torque can be independently controlled by two uncoupled control loops. For high dynamic performance, the torque is controlled by keeping the rotor flux Ψ_{rd} constant while varying the rotor current component i_{rq}. Keeping the rotor flux constant implies maintaining i_{sd} at a constant value while the rotor current component i_{rq} is controlled by the stator current component i_{sq}.

The control strategy requires the rotor flux orientation to be determined in order to calculate i_{sd} and i_{sq}. The direct vector control method estimates the magnetic flux vector as a function of the stator voltage, the stator current and the rotor speed. There are three types of rotor flux estimators differing by the input data they use: the current-speed

estimator (I_s, ω_{er}), the current-voltage estimator (I_s, U_s) and the current-voltage-speed estimator (I_s, U_s, ω_{er}). The indirect vector control method is simpler as it calculates only the argument θ of the rotor flux as a function of i_{sd} and i_{sq}. The direct vector control is more robust than the indirect vector control but its performance depends on the type of flux estimator used.

The current-speed estimator is derived from the basic rotor equation and from rotor flux expression as shown in (2-62), (2-63) and (2-64). The rotor flux is the solution of the integral equation (2-65). This estimator works well at low speeds but it is not precise at high speeds because in this case the speed measuring errors have a big influence on the calculation results.

$$\begin{cases} 0 = R_r \underline{i}_r^s + \dfrac{d\underline{\Psi}_r^s}{dt} - j\omega_{er}\underline{\Psi}_r^s \\ \underline{\Psi}_r^s = L_m \underline{i}_s^s + L_r \underline{i}_r^s \end{cases} \tag{2-62}$$

$$0 = R_r \cdot \frac{\underline{\Psi}_r^s - L_s \underline{i}_s^s}{L_r} + \frac{d\underline{\Psi}_r^s}{dt} - j\omega_{er}\underline{\Psi}_r^s \tag{2-63}$$

$$\frac{d\underline{\Psi}_r^s}{dt} = \left(-\frac{1}{T_r} + j\omega_{er}\right)\cdot \underline{\Psi}_r^s + \frac{L_s}{T_r}\underline{i}_s^s \tag{2-64}$$

$$\underline{\Psi}_{r(1,\omega)}^s = \int_0^t \left[\left(-\frac{1}{T_r} + j\omega_{er}\right)\cdot \underline{\Psi}_r^s + \frac{L_s}{T_r}\underline{i}_s^s\right]dt \tag{2-65}$$

The current-voltage flux estimator is derived from the stator equation and the stator flux expression (see (2-66), (2-67), (2-68)). Therefore, the equation defining the current-voltage flux estimator is (2-69). This method offers accurate results at high speeds but the precision at low speeds is low.

$$\begin{cases} \underline{u}_s^s = R_s \underline{i}_s^s + \dfrac{d\underline{\Psi}_s^s}{dt} \\ \underline{\Psi}_s^s = L_s \underline{i}_s^s + L_m \underline{i}_r^s = L_s \underline{i}_s^s + \dfrac{L_m}{L_r}\left(\underline{\Psi}_r^s - L_m \underline{i}_s^s\right) \end{cases} \tag{2-66}$$

$$\underline{u}_s^s = R_s \underline{i}_s^s + L_s \frac{d\underline{i}_s^s}{dt} + \frac{L_m}{L_r}\left(\frac{d\underline{\Psi}_r^s}{dt} - L_m \frac{d\underline{i}_s^s}{dt}\right) \tag{2-67}$$

$$\frac{d\underline{\Psi}_r^s}{dt} = \frac{L_r}{L_m}\left(\underline{u}_s^s - R_s \underline{i}_s^s\right) + \frac{L_s L_r - L_m^2}{L_m}\cdot \frac{d\underline{i}_s^s}{dt} \tag{2-68}$$

$$\underline{\Psi}^{s}_{r(I,U)} = \frac{L_r}{L_m} \cdot \int_0^t \left(\underline{u}^{s}_{s} - R_s \underline{i}^{s}_{s}\right) dt + \frac{L_s L_r - L_m^2}{L_m} \cdot \underline{i}^{s}_{s} \qquad (2\text{-}69)$$

The current-voltage-speed estimator (I_s, U_s, ω_{er}) combines the previous two solutions: equation (2-65) and equation (2-69). It generates good rotor flux estimates both at low speeds and high speeds.

$$\underline{\Psi}^{s}_{r(I,U,\omega)} = \frac{\underline{\Psi}^{s}_{r(I,\omega)} + \underline{\Psi}^{s}_{r(I,U)}}{2} \qquad (2\text{-}70)$$

The rotor flux is the original choice for field orientation because in this reference frame the equations corresponding to the two axes ((2-60) and (2-61)) are completely independent. As a result, this control method generates the best dynamic performance. On the other hand, the stator flux orientation has the advantage that the torque calculation uses the stator flux instead of the rotor flux as illustrated by (2-71) which is a consequence of (2-49-b). The stator magnetic flux is much easier to calculate than the rotor magnetic flux because it depends on stator quantities (currents, voltages and resistance) that can be directly measured.

$$T = \frac{2}{3} p \cdot \mathrm{Im}\left\{\underline{i}^{\theta}_{s}\right\} \cdot \mathrm{Re}\left\{\underline{\Psi}^{\theta *}_{s}\right\} = \frac{2}{3} p \cdot \Psi_{sd} \cdot i_{sq} \qquad (2\text{-}71)$$

$$\underline{\Psi}^{\theta}_{s} = \int_0^t \left(\underline{u}^{\theta}_{s} - R_s \underline{i}^{\theta}_{s}\right) dt \qquad (2\text{-}72)$$

A typical direct rotor field oriented control scheme (see Fig. 2-8)) contains two closed loops: one for i_{sd} (controlling the motor magnetic flux) and the other for i_{sq} (controlling the motor torque). The rotor flux orientation exploits the advantage that the two quantities can be controlled independently: the value of one stator current component does not have any influence over the value of the other current component. This property simplifies the control structure and generates good dynamic performance. One of the three flux observers previously described is used to determine the rotor magnetic flux. This information is used to calculate the reference frame transformations: from the stator reference frame to rotor reference frame, and from the rotor reference to stator reference frame.

The flux generating current component (i_{sd}) is maintained constant for speeds under the rated value but is decreased for speeds above the rated value (in the so-called field weakening region). Regardless of the vector control strategy, it can be

demonstrated that maintaining the magnetic flux constant at different stator frequencies implies that the stator voltage amplitude is approximately proportional to the stator frequency. As in the case of scalar control, the stator voltage amplitude is given by an equation similar to (2-57). Therefore, for speeds larger than the rated value the magnetic flux value cannot be kept constant because that would require high voltages that may damage the motor. High speeds are obtained at the expense of the field weakening which decreases the efficiency of the motor.

The torque generating current component (i_{sq}) is calculated as a function of the required motor torque and the motor field. The reference current i_{sq}^{ref} is proportional to the torque-to-field ratio. The torque is calculated in turn as a function of the difference between the reference speed and the actual speed of the motor.

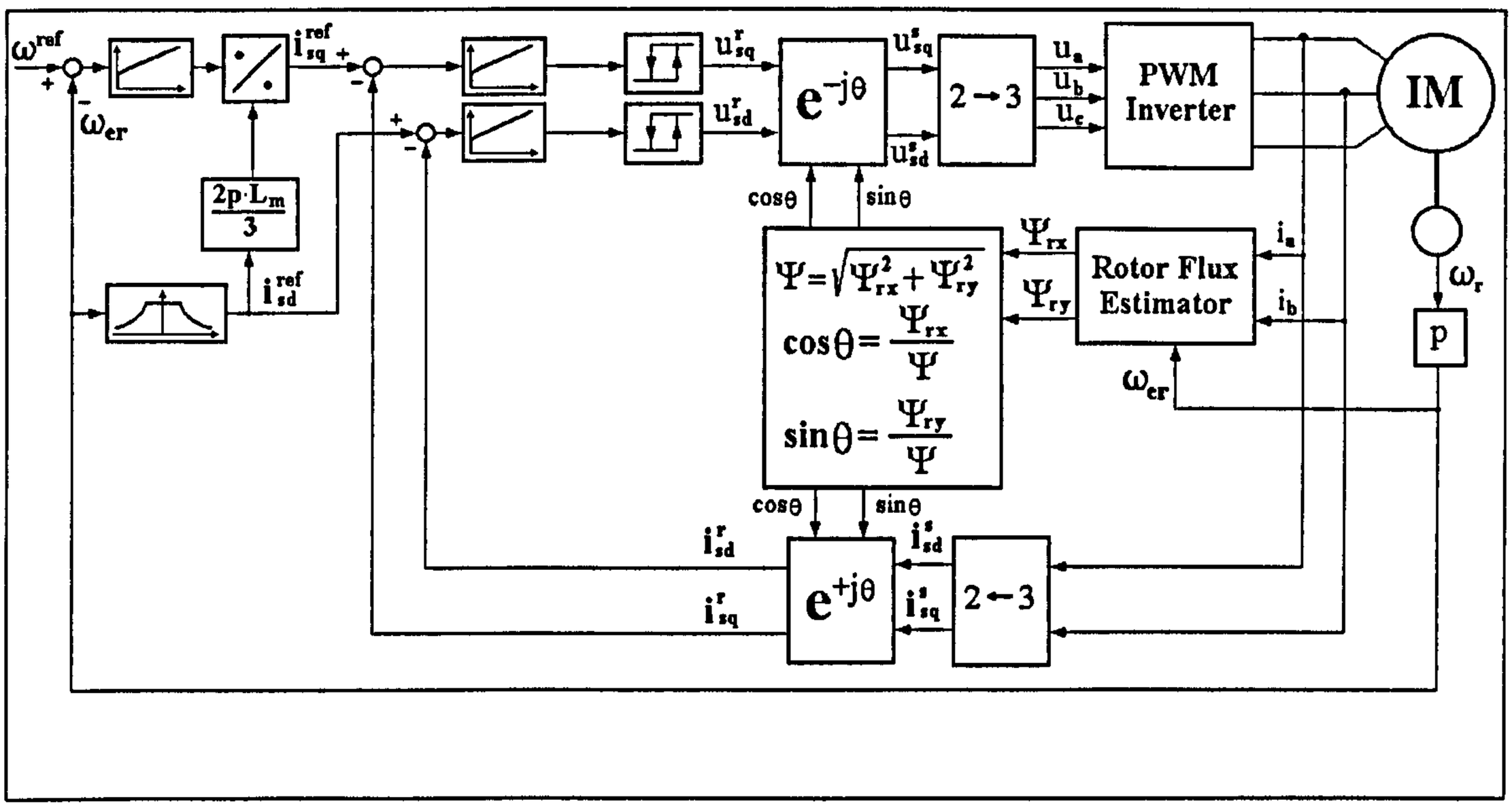

Fig. 2-8 –Direct rotor field oriented control scheme

In the case of indirect rotor field orientation, the flux orientation is calculated by integrating the stator angular frequency (2-73). The slip angular frequency is estimated as shown by equation (2-76) which is derived from the basic equations governing the rotor circuit (see (2-74) and (2-75)). In (2-76) it is implicit that the rotor flux amplitude is constant due to very good current controllers providing very fast (ideally instantaneous) dynamic response. Parameter detuning leads to a loss of rotor field orientation and to a deterioration of the system dynamic response. The rotor time constant T_r especially should be updated through an estimator [101].

$$\theta(t) = \int_0^t (\omega_{slp} + \omega_r)dt = \int_0^t \omega_s \cdot dt \qquad (2\text{-}73)$$

$$0 = R_r \underline{i}_r^r + \frac{d\underline{\Psi}_r^r}{dt} + j(\omega_{es} - \omega_{er}) \cdot \underline{\Psi}_r^r \tag{2-74}$$

$$\begin{cases} 0 = R_r i_{rq} + \omega_{slp} \Psi_{rd} + \dfrac{d\Psi_{rq}}{dt} = R_r i_{rq} + \omega_{slp} \Psi_{rd} = R_r i_{rq} + \omega_{slp} \Psi_r \\ 0 = R_r i_{rd} - \omega_{slp} \Psi_{rq} + \dfrac{d\Psi_{rd}}{dt} \end{cases} \tag{2-75}$$

$$\omega_{slp} = -\frac{R_r i_{rq}}{\Psi_r} = \frac{L_m R_r}{L_r \Psi_r} \cdot i_{sq} \approx \frac{1}{T_r} \cdot \frac{i_{sq}}{i_{sd}} \tag{2-76}$$

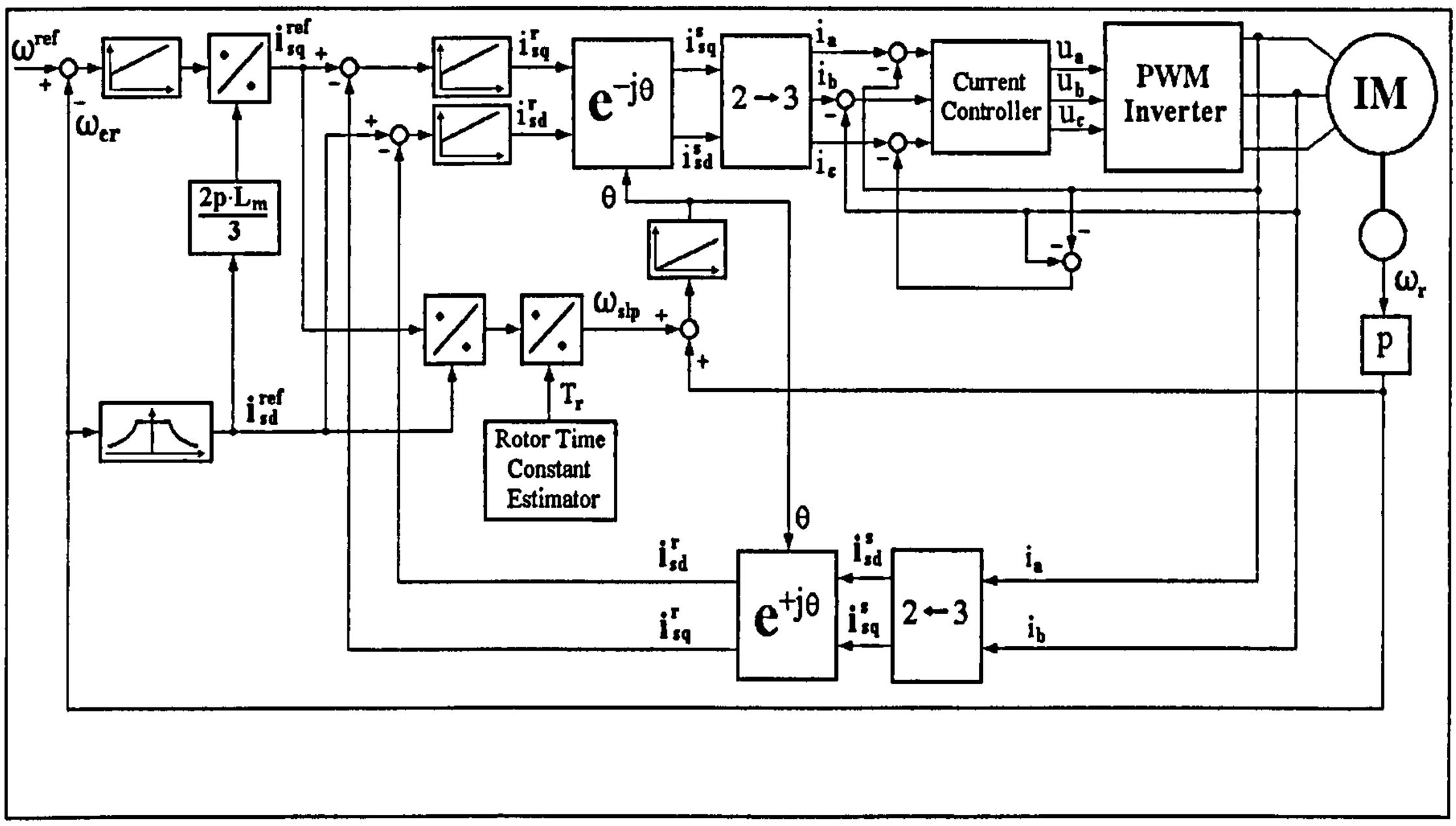

Fig. 2-9 - Indirect rotor field oriented control scheme

2.3.2.2 Stator and Air-Gap Flux Orientation

In the case of stator flux orientation, the flux equations take the form presented in (2-77). The magnetic flux vector and the stator current vector are the solutions of two coupled equations: (2-78) and (2-79) derived from (2-77). Therefore, the magnetic flux and the torque-generating current component cannot be controlled independently as in the case of rotor orientation. Here any modification of the magnetic flux has effects on the torque-generating current component. This slows the system transient response unless special compensation blocks are added to the control scheme.

$$\begin{cases} \Psi_{sd} = L_s i_{sd} + L_m i_{rd} \\ \Psi_{sq} = L_s i_{sq} + L_m i_{rq} = 0 \\ R_s i_{sd} + \dfrac{d\Psi_{sd}}{dt} - \omega_{e\Psi_s} \Psi_{sq} = R_s i_{sd} + \dfrac{d\Psi_{sd}}{dt} = u_{sd} \\ R_s i_{sq} + \dfrac{d\Psi_{sq}}{dt} + \omega_{e\Psi_s} \Psi_{sd} = R_s i_{sq} + \omega_{e\Psi_s} \Psi_{sd} = u_{sq} \end{cases} \quad (2\text{-}77)$$

$$\frac{d\Psi_{sd}}{dt} = u_{sd} - R_s i_{sd} \quad (2\text{-}78)$$

$$i_{sq} = \frac{u_{sq} - \omega_{e\Psi_s} \Psi_{sd}}{R_s} \quad (2\text{-}79)$$

The equations underlying the air-gap flux orientation are identical to (2-78) and (2-79) but are expressed in a different reference frame. Both the stator and the air-gap oriented vector control strategy are similar to the rotor oriented vector control in that the magnetic flux vector is kept constant for speeds below the rated value, while the torque is varied by modifying the corresponding current component (i_{sq} in this case).

Fig. 2-10 presents an example of stator field orientation. The control method is similar to the rotor flux orientation but contains an additional flux controller. The flux controller is added to diminish the effects of the interaction between the magnetic flux vector and the torque-generating stator current component.

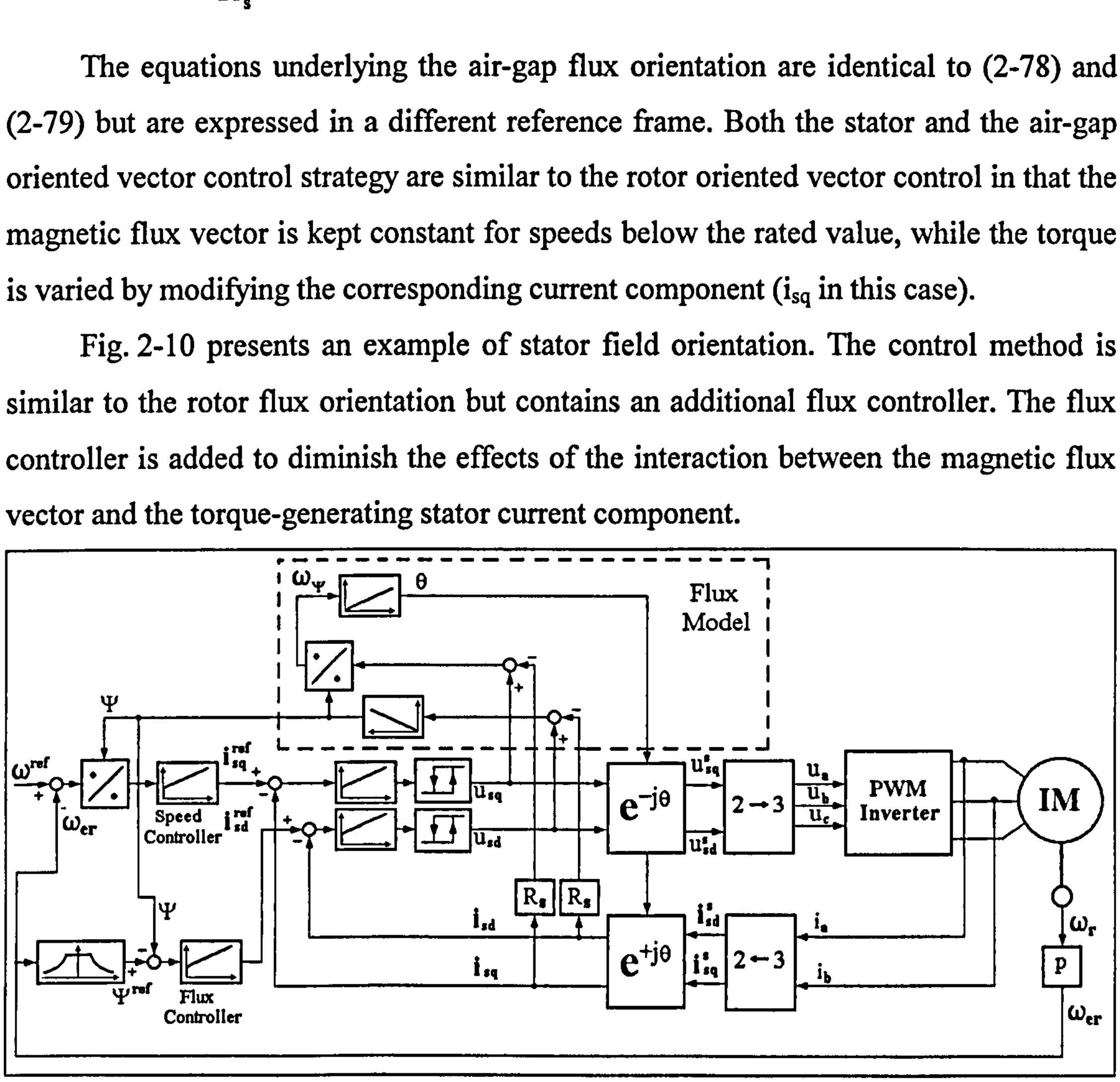

Fig. 2-10 –Direct stator field oriented control scheme

2.3.2.3 Direct Torque Control

In a PWM inverter-fed machine, the vector $\underline{\Psi}_r$ is more filtered than $\underline{\Psi}_s$ and therefore $\underline{\Psi}_r$ rotates more smoothly. The motion of $\underline{\Psi}_s$, dictated by the stator voltage, is discontinuous, but the average velocity is the same with that of $\underline{\Psi}_r$ in steady state. The direct torque control (DTC) method is based on relation (2-49-c). Therefore, the torque is controlled by varying the angle δ between the two flux vectors. Any DTC implementation contains a flux control loop and a torque control loop. The reference torque value is calculated by a speed controller, while the flux reference is determined as a function of the reference speed ω^{ref}.

The machine voltages and currents are sensed to estimate the torque and the stator flux vector. The flux vector estimation gives information about the 60^0 sector where $\underline{\Psi}_s$ is located. The errors E_Ψ and E_T generate digital signals through the respective hysteresis-band comparators. A three-dimensional look-up table then selects the most appropriate voltage vector (u_a, u_b, u_c) to satisfy the flux and torque demands.

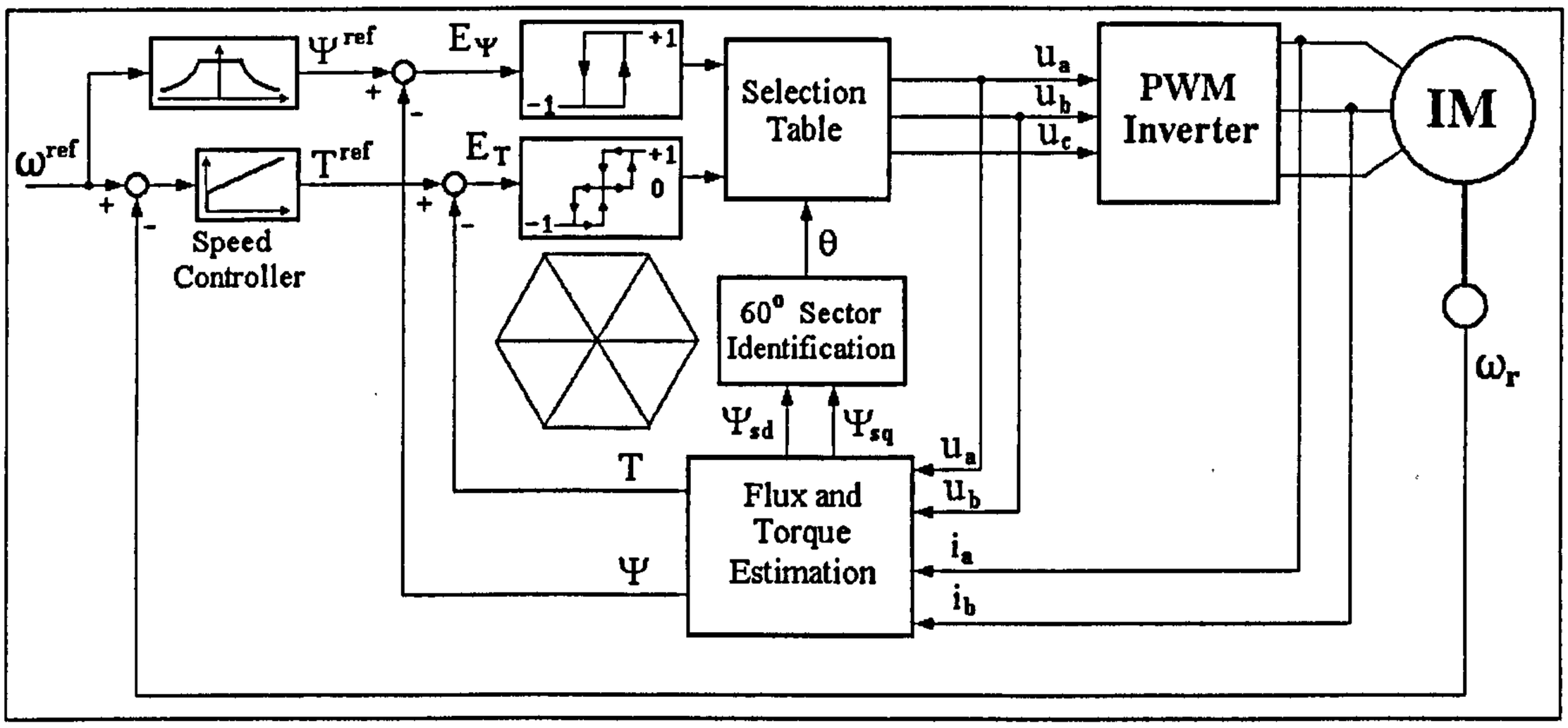

Fig. 2-11 – Direct torque control with speed sensor

DTC ensures fast transient response and generates simple implementations due to the absence of the closed loop current control, traditional PWM algorithm and the vector transformations. It can be implemented with speed sensor as well as in sensorless configurations. However, the drawbacks of DTC are the pulsating torque, pulsating flux and the increased harmonic loss [33]. Recently a large number of papers have been published concerned with improving DTC control [96], [69], [70], [75], [40], [18], [39], [69].

2.3.2.4 Sensorless Vector Control Schemes

The speed estimation methods for induction motors are based on the possibility to calculate the rotor speed as a function of stator currents and stator voltages. Therefore the physical speed sensor is replaced by software or hardware implemented module that performs the necessary calculations. The relation between the voltage and current is influenced by both the motor speed and the winding parameters. These parameters are subject to alterations during the motor operation due to heating and magnetic saturation. Consequently, on-line parameter estimation procedures need to be implemented alongside speed estimation algorithms to ensure correct results under various operation conditions.

Complex mathematical methods have been developed to integrate the speed estimation with the electrical parameter estimation process and to achieve high accuracy and independence of the motor parameter variations. These methods combine the classical field orientation approach with extended Kalman filters [14], [98], Luenberger Observers [88], [111], neural networks [31], [131], fuzzy logic [136], [27]. A different approach makes use of the effects of the rotor saliencies on the stator currents and voltages [122] or the parasitic effects that originate from the discrete winding structure of a cage rotor. In both these two cases, the stator currents contain harmonics that depend on the rotor speed so that Fourier transforms are involved in the speed calculation. Most of these methods are more accurate at high speeds than at low speeds. As a result, the lowest speed at which the system works correctly is an important performance indicator.

The Kalman filter (KF) was developed by R. Kalman and R. Bucy in the early 1960s [79], [80]. The standard KF [129] is a recursive state estimator for multiple-input/multiple-output systems with noisy measurement data and with process noise (stochastic plant model). It uses the inputs and the outputs of the plant together with a state-space model of the system, to give optimal estimates of the system state. The space state model is described by equation (2-80) where vector x is the state of the system and vector u contains the system inputs. The system output is given by (2-81). The matrices v and w, known as the spectral density matrices, model the noise processes. The noise is supposed to be *white* and *gaussian*.

$$\dot{x} = Ax + Bu + Fv \tag{2-80}$$

$$y = C \cdot x + w \tag{2-81}$$

The filter equation is given in (2-82), where K is the gain matrix of the filter. K is calculated as a function of the matrices v and w that describe the statistical properties of the noise processes. Equation (2-82) has the general form of a linear state-space observer. Thus, the KF is an optimal observer because it calculates the vector x as a function of vector u in such a manner that the adverse effect of the noise is minimised.

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - C\hat{x}) \qquad \textbf{(2-82)}$$

In the standard linear form, the Kalman filter can only estimate the stator current d-q components, and the rotor current d-q components. To estimate the rotor speed and/or the rotor resistance (the critical electrical parameter for most of the control strategies), the time-varying variable is treated as a state variable. Consequently, a non-linear system model is generated. To use a non-linear model with the standard Kalman filter, the model must be linearized around the current operating point, giving a linear perturbation model. The result is the extended Kalman filter (EKF). A comparison of the performances of KF and EKF is presented in [98]. The applications using KFs and EKFs are very popular nowadays although they impose high computational demands on the digital equipment involved [116].

The sensorless vector control of induction motors continues to be investigated by many authors and several improvements have been proposed in the recent years [94], [65], [81], [84], [112], [120], [135]. Many companies have launched their own sensorless vector control products [20]. The most representative products are shown in Table 2-1.

Table 2-1 - Representative AC Sensorless Vector Control Products

Company	Product	Ratings kW	Vac input	Speed Reg (±%)	Torque Reg (±%)	Min. Speed at 100% cont. torque
ABB	ACS 600	2.2-600	380-690	0.1-0.3	2	2 Hz
Allen-Bradley	1336 Impact/ Force AC Drive	0.75-485	230-600	0.5	5	0.5 Hz
Baldor Electric	17H Encoderless	0.75-373	180-660	10% of slip	3.5	100 rpm

Company	Product	Ratings kW	Vac input	Speed Reg (±%)	Torque Reg (±%)	Min. Speed at 100% cont. torque
Cutler-Hammer	AF93	1.5-15	340-528	0.5	N/A	50 rpm
MitsubishiElectr. America	A200E A024/A044	0.4-55 0.1-3.7	230-575 230/460	1.0 1-3	N/A N/A	<1 Hz 3 Hz
Siemens E&A	Master Drive 6SE70	to 1,500	208-690	0.1	<2.5	0
Square D	Altivar 66SV	0.75-220	208-460	1.0	N/A	0.5 Hz
Yaskawa Electr. America	VS-616G5	0.4-800	200-600	0.1	3	0.5 Hz
NFO Control AB	NFO Sinus Switch	0.37-5.5	230-400	1	1	1

The Natural Field Orientation (NFO) method, invented and patented by the Swedish company named NFO Control AB, is one of the simplest and most efficient sensorless motor control strategies so far. NFO Control AB implemented this method into hardware alongside an improved PWM switching strategy and sell it under the name "NFO Sinus Switch". NFO is derived from the stator field oriented vector control and it can be implemented with both speed sensor and sensorless but its advantages are fully exploited in the sensorless configuration. The corresponding control circuit is a simplification of the control scheme in Fig. 2-10. The essence of NFO is that the magnitude of the stator flux is not calculated by integration as in the case of stator flux orientation. The flux is set in open loop as a reference quantity that may be subject to change for field weakening [77], [78]. Thus, both the flux controller and the divider, that are present in Fig. 2-10 inside the speed control loop, are eliminated.

NFO can be implemented in several forms beginning with the basic configuration without current controllers, shown in Fig. 2-13, applicable to small drives. In this case, the voltage component u_{sq} is determined by the speed controller while the voltage component u_{sd} is calculated only as a function of the magnetising current i_{ms} so that the

correct stator magnetic flux is generated. The stator magnetising current is defined by (2-83).

$$\underline{\Psi}_s = L_s \underline{i}_s + L_m \underline{i}_r = L_m \underline{i}_{sm} \tag{2-83}$$

The stator equations can be written using the quantity i_{ms} as in system (2-84). One of the features of NFO is that the control scheme operates so that the modulus of i_{sm} equals i_{sd} (see Fig. 2-12). Therefore, the reference voltages are calculated according to (2-85).

$$\begin{cases} L_m \dfrac{di_{sm}}{dt} = u_{sd} - R_s i_{sd} \\ \omega_{sm} L_m i_{sm} = u_{sq} - R_s i_{sq} \end{cases} \tag{2-84}$$

$$\begin{cases} u_{sd}^{ref} = R_s i_{sm} \\ u_{sq}^{ref} = R_s i_{sq} + \omega_{ms} L_m i_{sm} \end{cases} \tag{2-85}$$

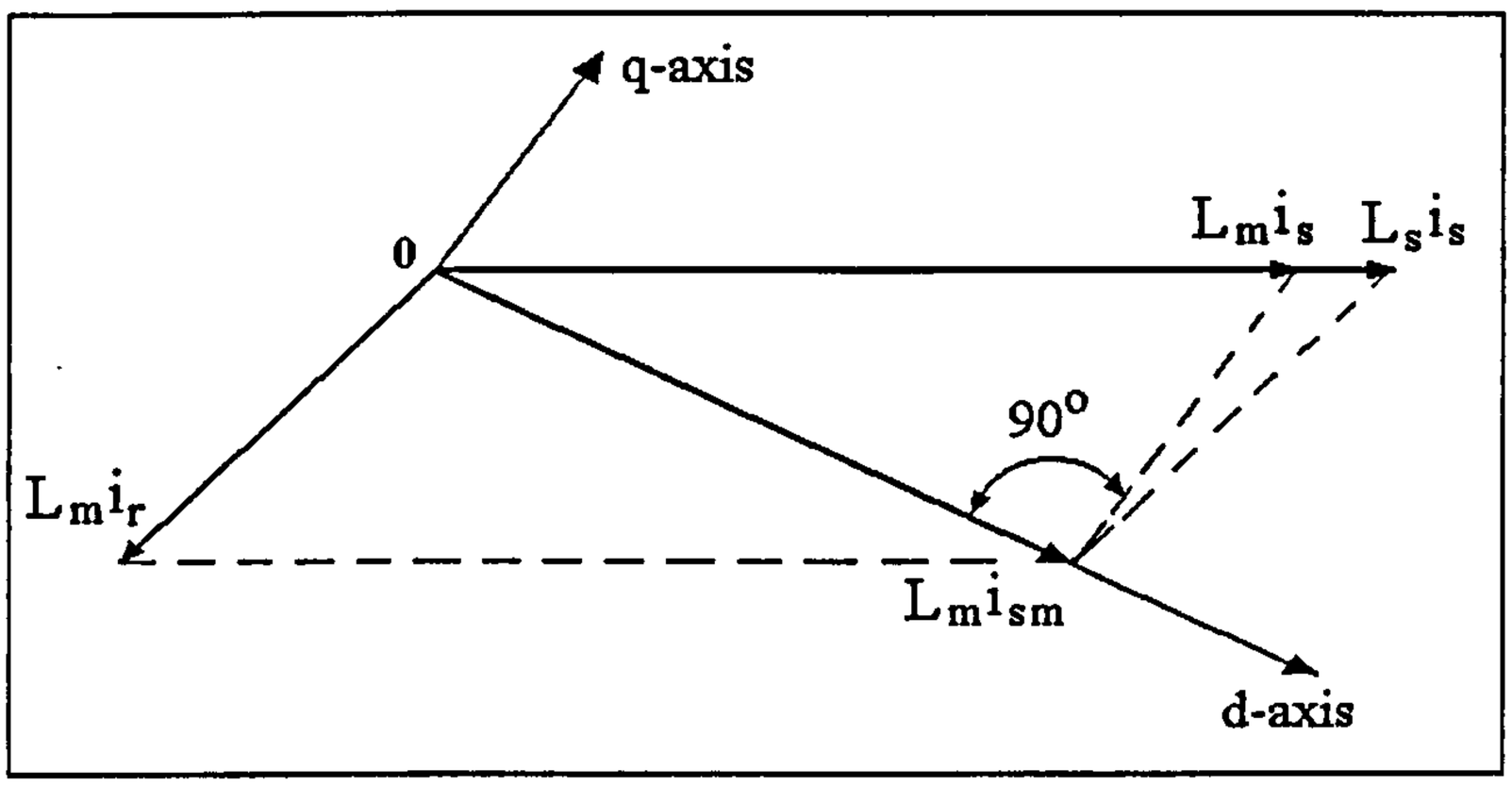

Fig. 2-12 – The stator and rotor current vectors in case of natural field orientation

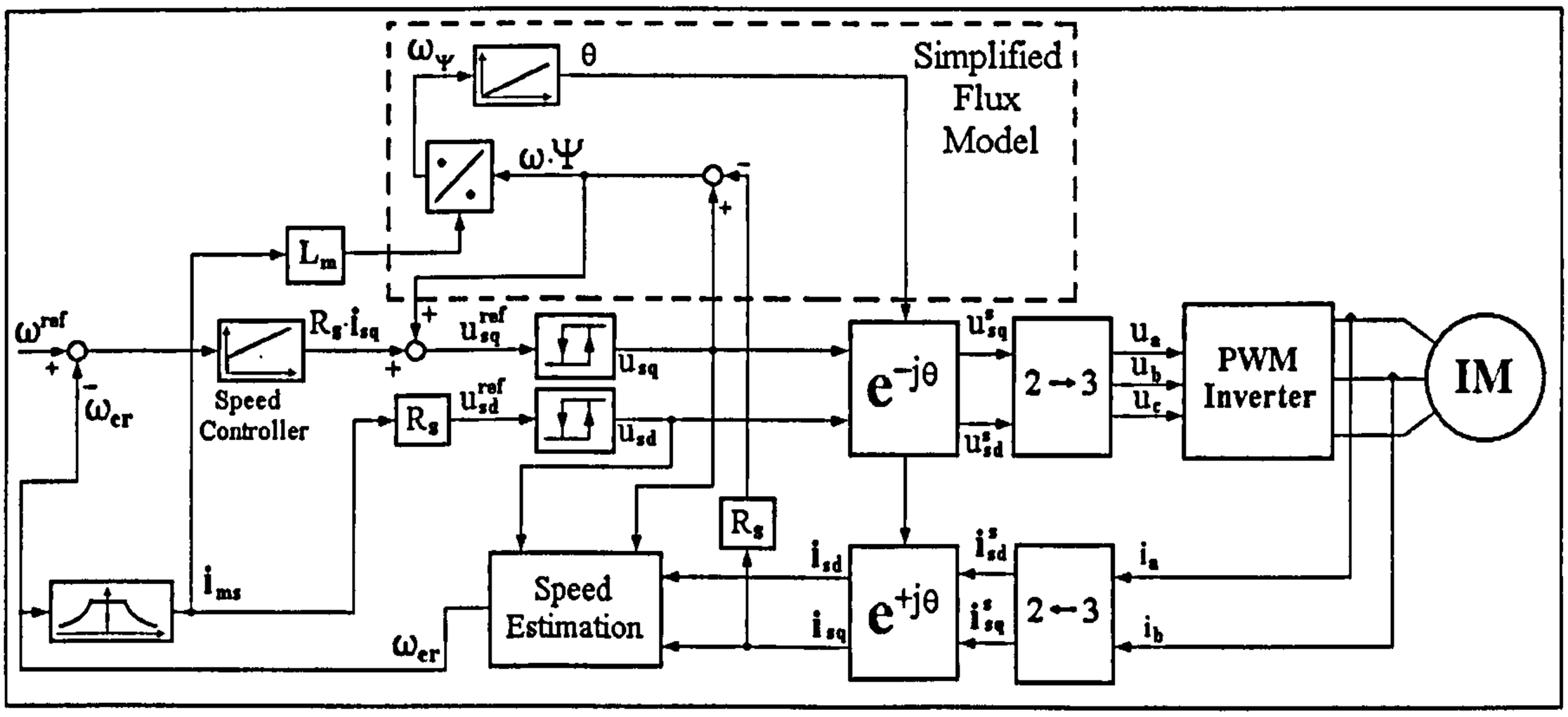

Fig. 2-13 – Natural field orientation (NFO)

The speed estimation is based on an inner voltage vector $\underline{e}_s$ defined according to (2-86). It is demonstrated [78] that the motor speed can be calculated using $\underline{e}_s$ as indicated in (2-87). This equation is valid whether or not i_{sm} equals i_{sd}.

$$\begin{cases} \underline{e}_s = \underline{u}_s - \left(R_s + R_r \dfrac{1+\sigma_s}{1+\sigma_r} \right) \underline{i}_s - \sigma \dfrac{d\underline{i}_s}{dt} \\ \sigma_s = \dfrac{L_{\sigma s}}{L_m} \\ \sigma_r = \dfrac{L_{\sigma r}}{L_m} \\ \sigma = \dfrac{L_s L_r - L_m^2}{L_m^2} \end{cases} \tag{2-86}$$

$$\omega_r = \frac{e_{sq}}{pL_s \cdot \left(\dfrac{i_{sm}}{1+\sigma_s} - \sigma \cdot i_{sd} \right)} \tag{2-87}$$

At low speed, the magnitude of $\underline{e}$ is small. Therefore small errors in measuring the motor currents will lead to large relative errors in calculating the vector $\underline{e}$ (2-86) that will in turn reflect into large relative errors of the estimated rotor speed. Thus, the speed estimation precision is minimal at very small rotor speed. Most sensorless control strategies face the same problem that is why the minimal speed that the system can efficiently control is one of the key parameters used in measuring the control system performance.

The space vector concept has been described alongside the space vector model of the three-phase induction motor. These concepts have been used to describe the main techniques for the control of induction motor drives. In the next chapter, neural network theory is briefly presented and neural control is considered with a view to assessing its applicability to produce efficient control systems for the envisaged induction motor applications.

2.4 COMMON CURRENT CONTROL SOLUTIONS REVIEW

The control of induction motor variable speed drives often requires an accurate control over the motor currents [132]. This is most often achieved by means of a voltage source inverter. Such an inverter is supplied with DC voltage and it generates three-phase PWM voltage with adequately controlled harmonic content. The standard thre-phase inverter configuration contains 6 power transistors connected into 3 pairs (A, B

and C) as illustrated in Fig. 2-14. Each pair belongs to one inverter leg. The normal inverter operation is a series of stable states separated by fast transients [55]. Only one transistor in each pair can be switched on, during a stable state. If both transistors in the same pair are switched on in the same time, short-circuit occurs and the inverter is irreversibly damaged.

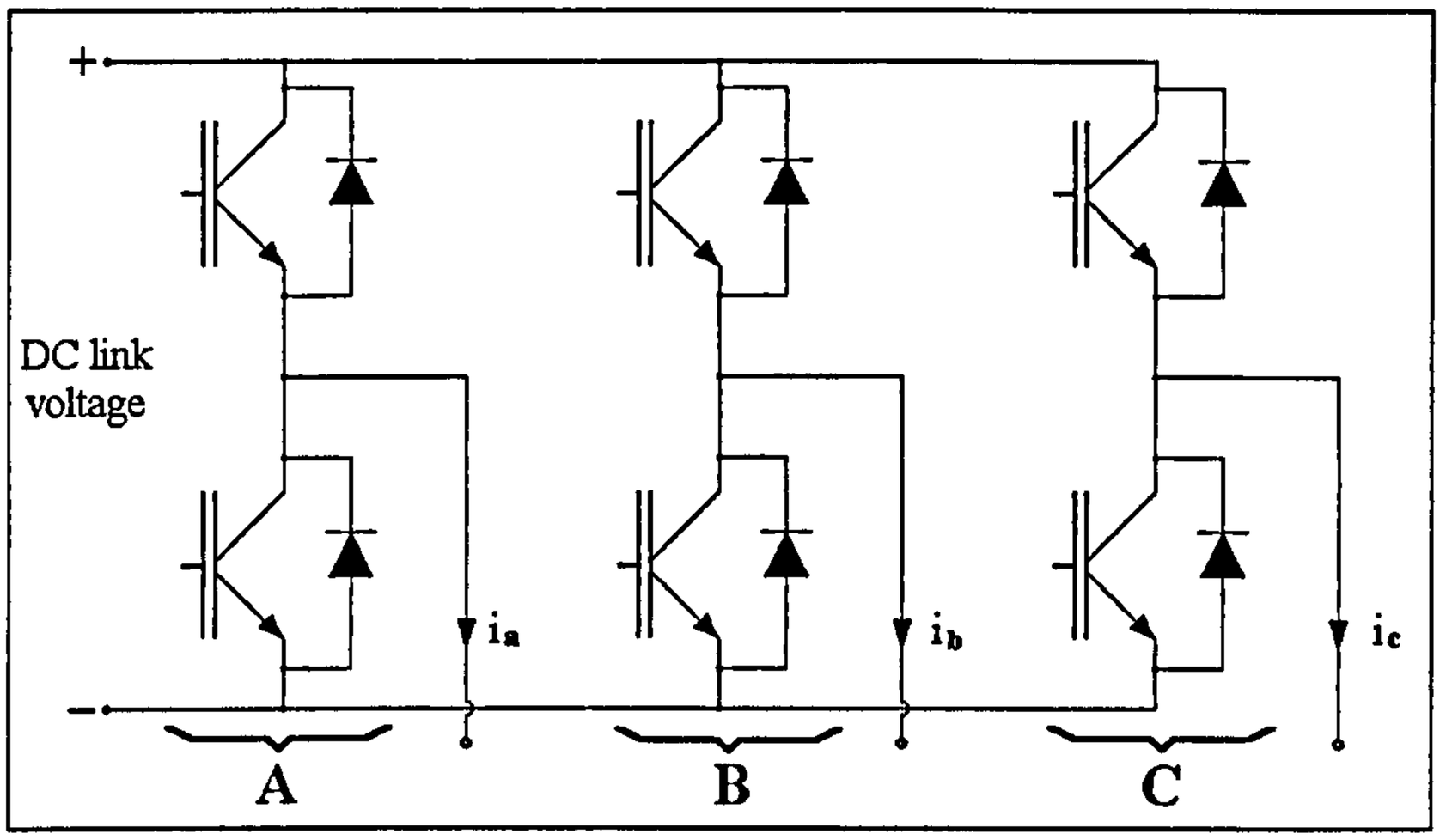

Fig. 2-14 – Three-phase PWM inverter

The inverter states are described by six bits, each bit taking value '1' when the corresponding transistor is turned on, and '0' when the transistor is turned off. The bits related to transistors in the same inverter leg have complementary values during the stable states. Therefore, the stable states can be described as sets of only three bits, each bit describing the operation of the upper transistor in the corresponding inverter leg. However, the transistor switching process is not instant so that during each transient, the first transistor has to be turned off before turning on the second one. Consequently, there are short time intervals when both bits related to one inverter leg are simultaneously '0', as illustrated in Fig. 2-15.

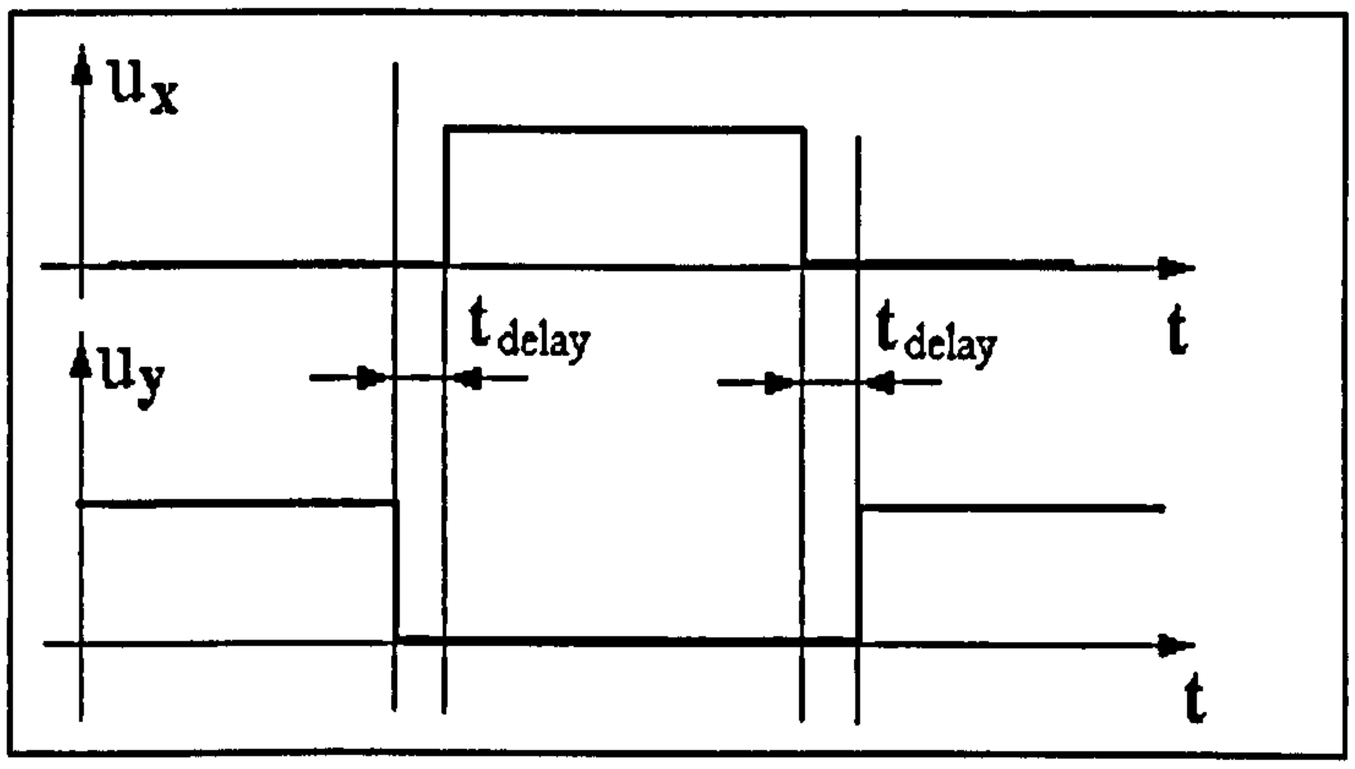

Fig. 2-15 - The control signals to the transistors in the same inverter leg

Usually, the inverter switching is controlled via an interface circuit that has three binary input signals and generates the optimal sequence of control signals on each of the six transistors. In this case, the switching process generating the three PWM voltage signals is mathematically described as a simple time series of bit triplets corresponding to the stable states of the inverter. The eight possible inverter states are related to only seven output voltages. This is due to an identical result corresponding to states (1,1,1) and (0,0,0) when the voltage across the load is zero, as shown in Fig. 2-16.

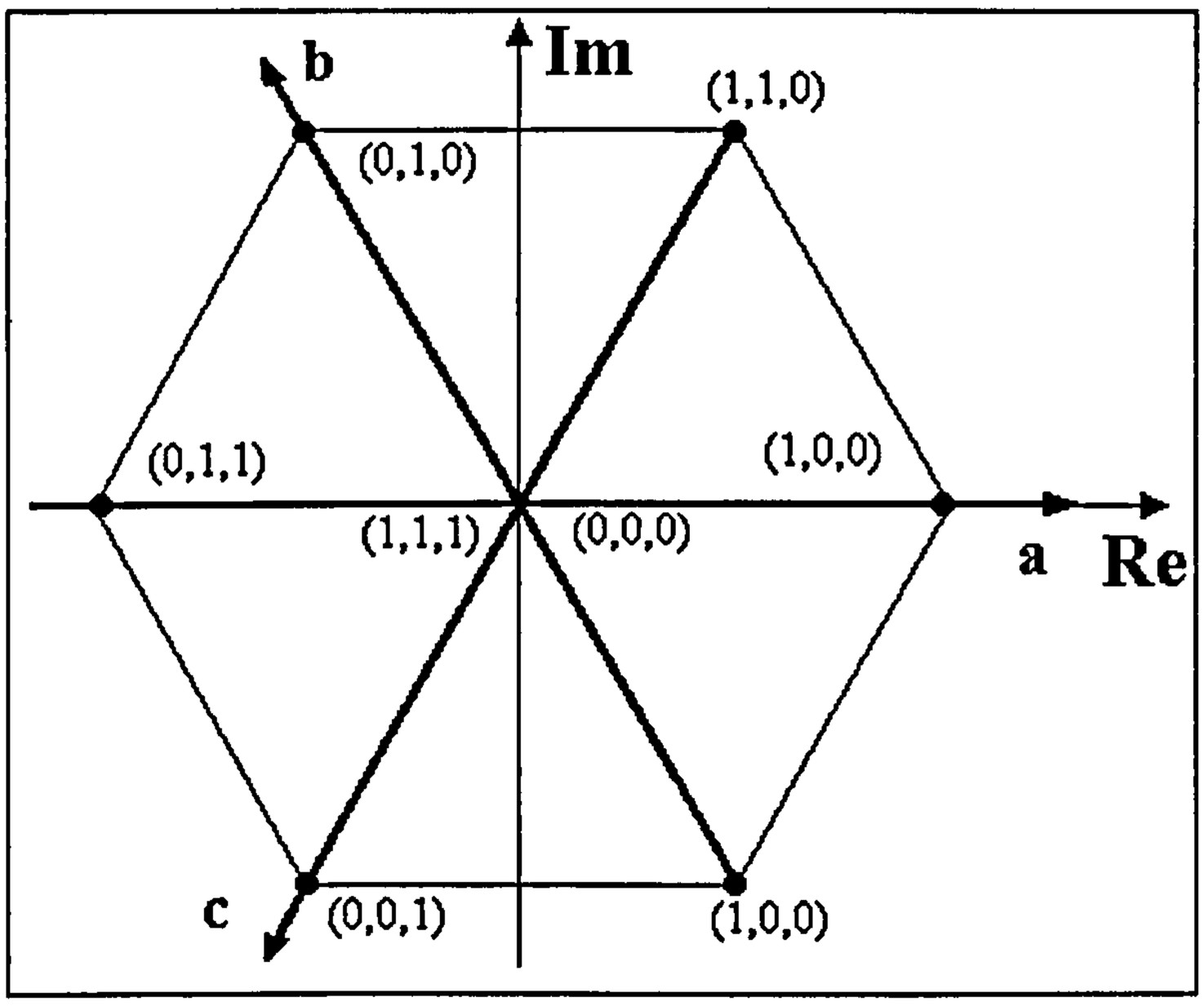

Fig. 2-16- The Inverter output voltage space vectors

The current control techniques presented in the literature fall in three categories: feedback control using ramp comparison PWM [56], hysteresis control [57], and predictive control [36], [83]. The first method involves the generation of a PWM voltage using the classical comparison between a triangular waveform (the carrier) and a sinewave (the modulator) [64]. The amplitude of the modulator is corrected based on the difference between the reference current amplitude and the actual current amplitude. The main drawback of this method is the slow current response.

The hysteresis current control method uses a set of three hysteresis controllers, as presented in Fig. 2-17. Each controller is included in a separate feed-back loop and therefore acts independently. The three controller outputs are binary signals that control the switching of the three inverter legs. This is the fastest control method that can be obtained with simple hardware resources. The main disadvantage is the variable switching frequency that depends on the load parameters and load operation conditions. The irregular switching conditions also affect the inverter efficiency and the reliability

due to the overrating of the power transistors. Some versions of this control method involve limiting the switching frequency to adequate values by using adaptive strategies to modify the hysteresis cycle width or by simply limiting the number of switches per second [82]. In spite of their high switching frequency, the hysteresis current controllers generate larger current ripples than predictive controllers operating at similar frequency.

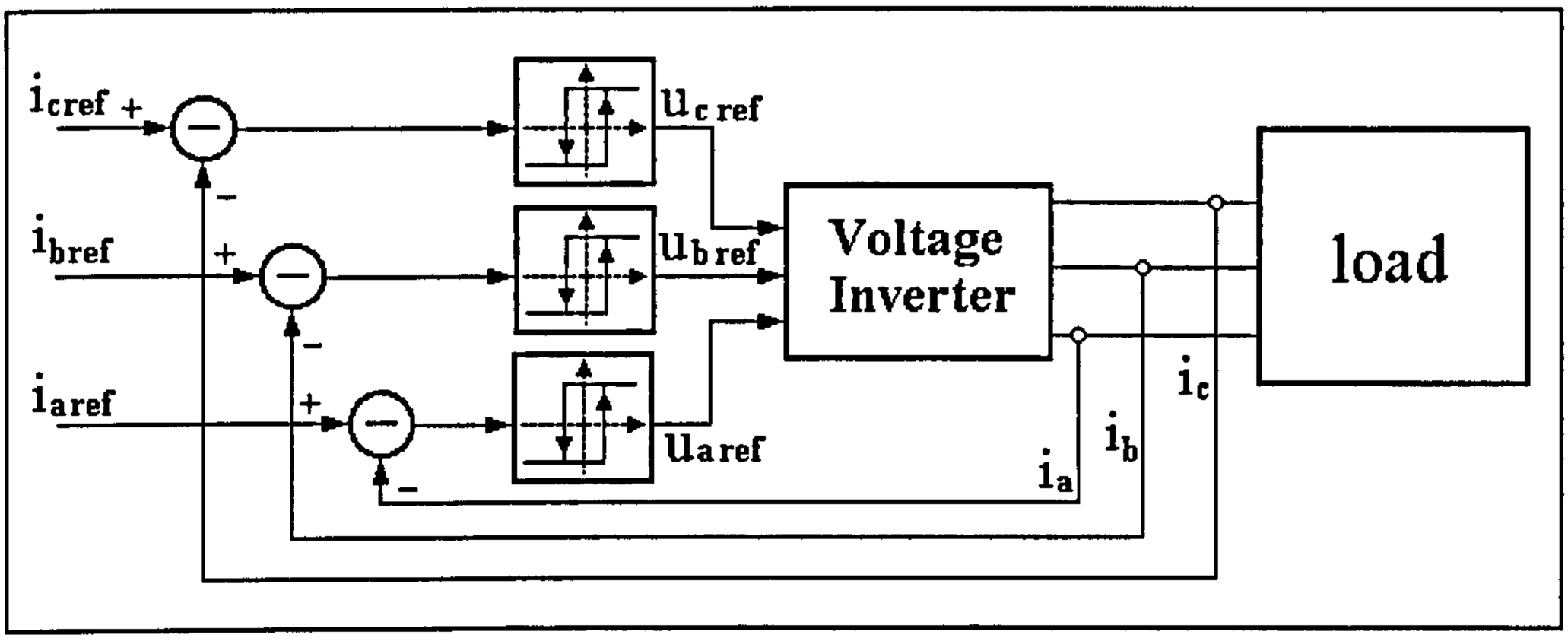

Fig. 2-17 - Current Control Strategy Using Hysteresis Controllers

The class of predictive current controllers performs better control by anticipating the future current response of the load as a function of the inverter voltage, and selecting the optimal inverter output according to the reference current. This approach is used to minimise the harmonic content of the current and the switching frequency [66], [87] or to improve the transient response speed [92], [59]. A combination of the two approaches is also possible [106]. This type of current control uses a load model consisting of a three-phase R-L-e circuit illustrated in Fig. 2-18 where e_a, e_b and e_c are voltage supplies.

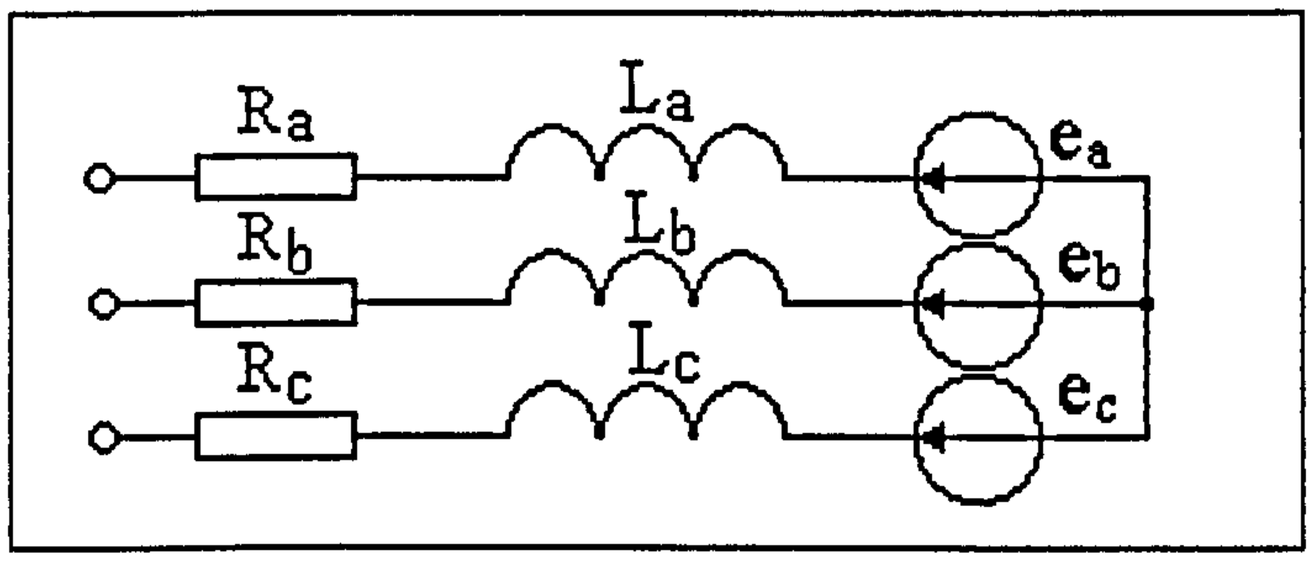

Fig. 2-18 – The equivalent circuit of the induction motor

The circuit is considered symmetrical, that is $R_a=R_b=R_c=R$, $L_a=L_b=L_c=L$ and therefore, the electrical quantities in the equivalent circuit are related by the space vector equation

$$\underline{u}(t) = R\underline{i}(t) + L\frac{d\underline{i}(t)}{dt} + \underline{e}(t) \qquad (2\text{-}88)$$

Controlling the current requires the calculation of the internal voltage $\underline{e}$, which depends on the nature of the load and on its operation mode. It is also necessary to measure or to estimate the resistance R and the inductance L in the equivalent circuit. Therefore, the predictive current control has two important drawbacks: the computational complexity and the requirement for information on the parameters in the equivalent circuit. The computational complexity can be overcome using the latest fast digital electronic devices (DSPs, FPGAs, ASICs) [76]. Methods that operate with approximate information on the circuit parameters have also been developed and validated by experimental tests [106]. The simplified approach adopted in [106] considers that the voltage component $R\underline{i}$ is negligible. This assumption is valid for a large category of electrical motors but it is not true for small power induction motors where the stator resistance is large. In this situation, equation (2-88) becomes

$$L\frac{d\underline{i}(t)}{dt} = \underline{u}(t) - \underline{e}(t) \tag{2-89}$$

Therefore, the current space vector moves on the trajectory whose direction depends on the expression $\underline{u}(t)$-$\underline{e}(t)$. The current control process is thereby transformed into a geometrical problem. Thus, the inverter voltage needs to be generated in such a manner that the space vector $\underline{u}(t)$-$\underline{e}(t)$ is situated on the same direction as $\underline{i}_{ref}(t)$-$\underline{i}(t)$, where $\underline{i}_{ref}(t)$ is the reference current.

Furthermore, the control solution presented in [106] requires only to determine which of the six equilateral triangles in Fig. 2-16 includes the vertex of vector $\underline{e}$. To achieve this, only the inverter voltage $\underline{u}(t)$ and the signs of the three load current derivatives are used. The adequate inverter output voltage is then determined considering that $\underline{e}$ is located in the middle of the corresponding triangle. The method operates with two alternative switching modes: a quick response mode, which avoids the inverter states (0,0,0) and (1,1,1), and a harmonic suppression mode, which includes these states. The method has been tested experimentally and proven superior to the hysteresis control method in terms of harmonic content.

This method has the advantage of a relatively simple implementation but it is not optimal because of the two simplifications it uses: the resistance R is neglected and the estimated vector $\underline{e}$ can have only six discrete positions. Calculations that are more accurate are performed by the predictive current controller proposed in [66].

An important criterion in selecting the appropriate current control strategy for a particular application is the level of inverter losses. The losses can be limited by

decreasing the switching frequency of the transistors but this method cannot be used in any circumstances because it increases the current harmonic content. The simplest solution applicable to predictive current control methods is to generate the two equivalent states (1,1,1) and (0,0,0) selectively, depending on the previous inverter state. If the previous state had two bits 1 and one bit 0, while the next voltage needs to be zero, then state (1,1,1) is generated. Otherwise, state (0,0,0) is generated [123]. A number of other methods that are capable to improve the current harmonic content without increasing the switching frequency have also been reported [83], [128], [59], [115].

An improved version of the control strategy initially proposed in [106] will be presented in chapter 4. The new strategy, does not neglect the effect of the resistance R as in [106], and it incorporates an on-line inductance estimation strategy allowing a more accurate calculation of the of the internal voltage e. Therefore, the strategy is applicable to all types of electrical motors and not only to electrical motors where the stator resistance is negligible.

2.5 IMPLEMENTATION SOLUTIONS FOR ELECTRICAL DRIVE CONTROL STRATEGIES

2.5.1 General Hardware Resources

The control systems can be implemented using several types of electronic equipment:

1) General purpose microprocessors
2) Transputers
3) Microcontrollers
4) Digital Signal Processors (DSPs)
5) Application Specific Integrated Circuits (ASICs)
6) Field Programmable Logic Arrays (FPGAs)

The microprocessors were invented in 1971 as universal VLSI circuits for general applications. Since the first years of their existence, they were used to implement high efficiency control strategies for electrical drives. The advances in the digital technology generated a series of different VLSI circuits whose architecture is adapted to specific tasks as opposed to the universality of the initial microprocessors.

The word transputer is derived from TRANSmitter and comPUTER. It is a microprocessor initially created by Inmos Ltd. UK. Compared to other microprocessors

the transputer has two very special features: it has on chip serial links for communicating to other transputers, and it has hardware support for timesharing. The serial communication links are used to connect several transputers in a network. The result is a parallel multiple-instruction-multiple-data system. The microcontrollers are microprocessors with the internal structure adapted to embedded system applications. Their instruction set is optimised for control applications, while the chip structure includes on-chip RAM and ROM memory, serial communication ports, timers, and a large number of internal registers. Although general microprocessors, transputers and microcontrollers have been used for motor control applications, ([35], [140], [17]) the DSPs, ASICs and FPGAs are the most commonly used hardware resources nowadays.

DSPs are general-purpose data processors initially created for applications that process large amounts of data such as data acquisition, image enhancement and processing, remote sensing, voice synthesis and recognition, telecommunications. The DSP architecture is adapted to handle mathematical problems in real-time. It implements functions such as Finite Impulse Filters (FIR), Infinite Impulse Filters (IIR)), Fast Fourier Transforms (FFT), convolutions, etc. The application of DSPs has now been extended to electric drive control because they extensively use many of the typical DSP functions as part of the speed and torque control algorithms.

Most of the DSP functions require the incoming data to be multiplied and added with various quantities generated by internal feedback mechanisms. This feature is generally called Multiply/ACcumulate. To increase performance, most general-purpose DSP processors perform a multiply/accumulate function in a single clock cycle. The hardware to perform this function is called a Multiply/Accumulator (MAC). Most DSPs have a fixed-point MAC while some have a more expensive floating-point MAC. Every processor is capable of performing signal-processing algorithms because they are all capable to perform additions and multiplications. However, a DSP performs this operation faster than a general-purpose microprocessor because it contains hardware resources optimised for this kind of calculations.

A relevant comparison of the performance of a typical FIR implementation using different technologies is provided in [85]. Each tap of a digital filter requires one multiply/accumulate cycle. A standard Pentium™ processor requires 11 clock cycles to perform a single multiply/accumulate operation whereas most DSPs require just a single cycle. A 50 MHz fixed-point DSP performs a multiply/accumulate cycle in only 20 ns while a 133 MHz Pentium processor requires 1.3 μs to perform the same function. As a

result, a 133 MHz Pentium processor has only 24% signal processing power of a 50 MHz DSP for the filter function shown in Fig. 2-19.

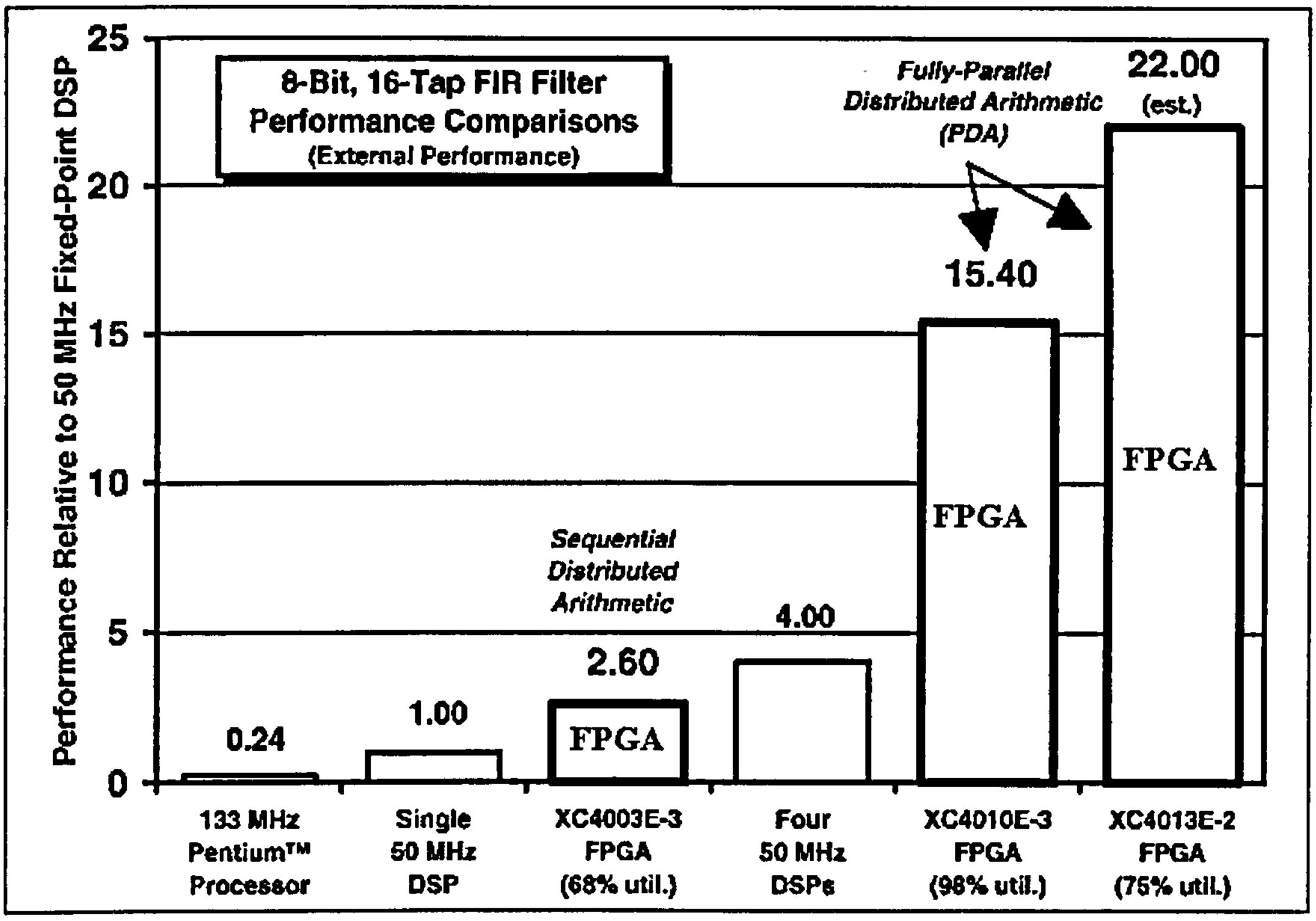

Fig. 2-19 - Relative speed for various implementations of an 8-bit, 16-tap FIR filter compared to a 50 MHz fixed-point DSP processor

Because most DSPs only have one MAC unit, each tap is processed sequentially, slowing the overall system processing speed. Some of the more powerful, DSPs have multiple MACs but they are more expensive. These DSPs perform multiple MACs in one clock cycle. The same goal is accomplished by using several single-MAC DSPs with shared high-speed memory. However, multiple processors require complex real-time multiprocessor code that is difficult to develop and debug. The millions of MACs per second that are possible to be achieved with multiple processors imply a high development effort.

ASIC technology is used whenever the application requires performance beyond the abilities of current DSPs or when the expected production volumes justify a semicustom design solution. Because DSPs are cheap devices, the use of ASICs is cost effective only in case of mass production. However, typical DSP functions can be very efficiently implemented into an ASIC architecture that is optimised for a target application, offering thereby higher processing speed. The ASIC approach is very efficient for example, in case of a complex digital-filtering algorithm requiring

numerous multiply/accumulate cycles. An ASIC implementation of the filter might have a large number of MACs so that all the taps can be processed in parallel.

The programmable logic (FPGA approach) provides a third solution that combines the best of both DSP and ASIC technology without their respective limitations. Like a general-purpose DSP, FPGAs are programmable and changeable. The designer can make changes quickly without the additional cost and long lead-time of an ASIC. On the other hand, FPGAs have sufficient complexity to host several MACs and other basic calculation units into a single device. As a result of the lower price of FPGAs compared with the ASICs, the FPGA implementation is an economically viable solution for a larger class of products than the ASIC approach.

Not only is the FPGA implementation faster than DSPs, but it offers good trade-offs between system density and performance. The first FPGA implementation of the 16-tap filter in Fig. 2-19 uses 68% of an XC4003E-3 FPGA, or roughly 1500 gates [54]. This implementation is 15 times faster than a 133MHz Pentium and outperforms a single 50 MHz DSP by a factor of 2.6. The key to its efficiency is the Sequential Distributed Arithmetic (SDA) algorithm [108], [53]. This algorithm takes advantage of the XC4000E architectural features. The multiply functions are mapped into the FPGA's function generators, the adders and accumulators use the XC4000E fast carry logic, and the serial shift registers are efficiently built in on-chip RAM [7]. The highest performance FPGA implementation uses about 75% of an XC4013E-2 FPGA, or about 9750 gates. Though roughly seven times larger than the first version (which was a space-efficient version), the high-performance implementation is 22 times faster than a Pentium processor. Even higher performance is possible if the application can tolerate the extra data latency caused by pipelining.

A broad range of alternate FPGA implementations is available. The trade-offs between density and performance are shown in Fig. 2-20. Each implementation can be tailored to the speed, density and cost requirements of the target application. Serial sequential arithmetic is the most efficient but also the slowest. Parallel Distributed Arithmetic (PDA) is the fastest but uses the most logic. SDA is a good compromise of speed and density, depending on system requirements.

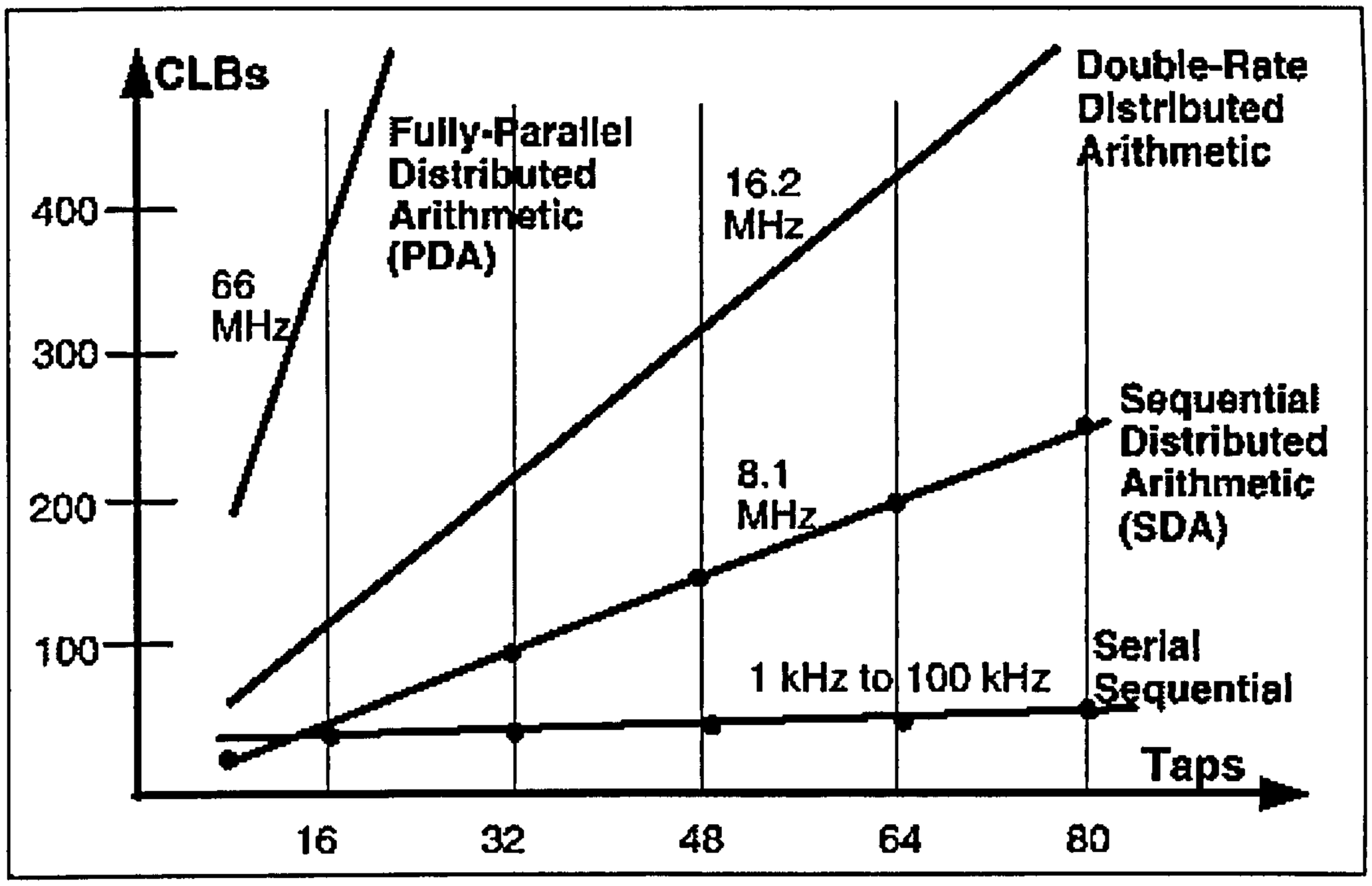

Fig. 2-20 - Performance of different Distributed Arithmetic (DA) FIR filter implementations and their relative silicon efficiency in XC4000E FPGA logic blocks (CLBs)

2.5.2 Implementation Solutions for Induction Motor Drives

The performance of a single DSP processor is adequate for a large class of standard drive control applications. As a result, DSPs have been largely used for drive system control applications involving the use of PWM inverters [51], [99],[126], [89], [135]. They have the advantage of flexibility and adaptation to different applications due to software control strategy.

However, generating PWM gating signals and implementing the current control loops require a high sampling rate to achieve a wide bandwidth performance. Therefore, a large amount of DSP computation resources must be devoted to generating the PWM signals and executing the motor current control algorithms [127].

If the control system combines sophisticated current control methods with other complex control algorithms and/or parameter identification, then the general DSP limitations previously presented can create design and development problems. Moreover, the software code for control algorithms is not optimally implemented in general-purpose DSP architectures. A typical control algorithm contains many repetitive feedback loops and parallel structures. Typically, about 20-40% of the DSP's code utilises 60-80% of the DSP processing power [85].

Although the employment of a further DSP can solve the problem, the additional hardware and software for such a dual-DSP controller will complicate the design process [124]. Consequently, the performance gain that comes with the use of additional DSP processors is small when compared to the increase in the product time-to-market and the supplementary financial burden involved by the complicated design process. Using FPGA-implemented accelerators in conjunction with one single DSP that monitors the operation of the system is the optimal solution for high-performance industrial plants controlled in real time. The FPGA-based DSP accelerator concept is similar to a floating-point coprocessor working with a general-purpose microprocessor. The repetitive data processing is performed at a high speed by FPGAs for each element to be controlled in the system. A fast DSP processor is used to handle the peak performance of a small piece of code. It monitors the overall activity in the system and implements the general control strategy. Efficient DSP/FPGA-based control structures for AC drives have already been reported in literature [125].

FPGAs will probably never completely replace general-purpose DSP processors. The current generation of FPGAs addresses the fixed-point DSP portion of the market. General purposes DSPs still dominate in floating-point performance as they have the advantage of using familiar software methods. Thus, the designer can implement the DSP algorithm using a programming language like C and compile the code for a specific DSP processor. On the other hand, the FPGA and the ASIC approaches require a radically different design approach due to the differences between the software and the hardware paradigms. These design differences tend to decrease nowadays but they still limit the number of applications developed using FPGA technology.

2.5.3 Modern ASIC/FPGA Design Methodologies

VLSI technology has recently moved into the submicron era and the integration level increases very fast (the transistor count doubles every 18 months). Consequently, increasingly complex circuits can be integrated on a single chip, while the design process is ever more difficult. The traditional design methods are not adequate to the complexity of the present electronic systems and to the time-to-market requirements. Moreover, the technology advances so fast that, in many situations, by the time a certain electronic equipment is designed and tested, the underlying implementation technology is already obsolete.

The first answer to the design methodology crisis was the development of Hardware Description Languages (HDL). They offer technology independent design

methods, consisting of abstract descriptions of the circuit functionality, in a programming language format. Synthesis software tools bridge the gap between the high-level abstract HDL descriptions and the low-level hardware implementation details specific to each technology. The use of abstract HDLs increases the design productivity very much, as compared with the traditional vendor proprietary tools for designing integrated circuits, which were based on specific technologies, and functioned primarily at the gate level.

The most popular HDL is VHDL whose evolution began with a mandate set by the Department of Defence (DoD) of the USA back in the early 1980's, as part of the Very High Speed Integrated Circuit (VHSIC) Program. This resulted in the adoption and initial release of an IEEE Standard 1076 in December 1987, which has been superseded by VHDL's IEEE Std 1076-1993 Language Reference Manual (LRM). The LRM was approved by the IEEE Standards Board in September 1993, and published in 1994. VHDL was developed as a flexible hardware description language, capable to handle hierarchical circuit models containing different levels of modelling abstraction (behavioural, structural, mixed) [110], [107]. Nowadays, VHDL is supported by al major Computer Aided Engineering (CAE) platforms.

The second answer to the design methodology crisis was the invention of the FPGA chips. Using FPGAs, fast prototyping techniques can be used in VLSI design thereby dramatically decreasing the time-to-market for the new digital products. The FPGA design cycle consists of several interrelated steps (Fig. 2-21) that usually involve the use of a hardware description language: VHDL or another HDL (Verilog, Abel, etc).

First, the abstract HDL circuit model is generated with a text editor, and then it is compiled and simulated. The simulation results are compared to the design requirements, and corrections in the initial model are performed if necessary. Once an adequate circuit model is obtained, it is synthesised generating a netlist description of the circuit. The netlist generation process takes into account the target FPGA technology and the imposed timing and area constraints. In the last stage, the netlist description is optimised and mapped onto the specific FPGA device used for implementation. The final result is a bitstream file that can be downloaded into the FPGA chip for practical verifications. The design cycle can be repeated in case the practical results are not satisfactory. After the final verification, the production of the new equipment can be achieved using either FPGA or ASIC technology.

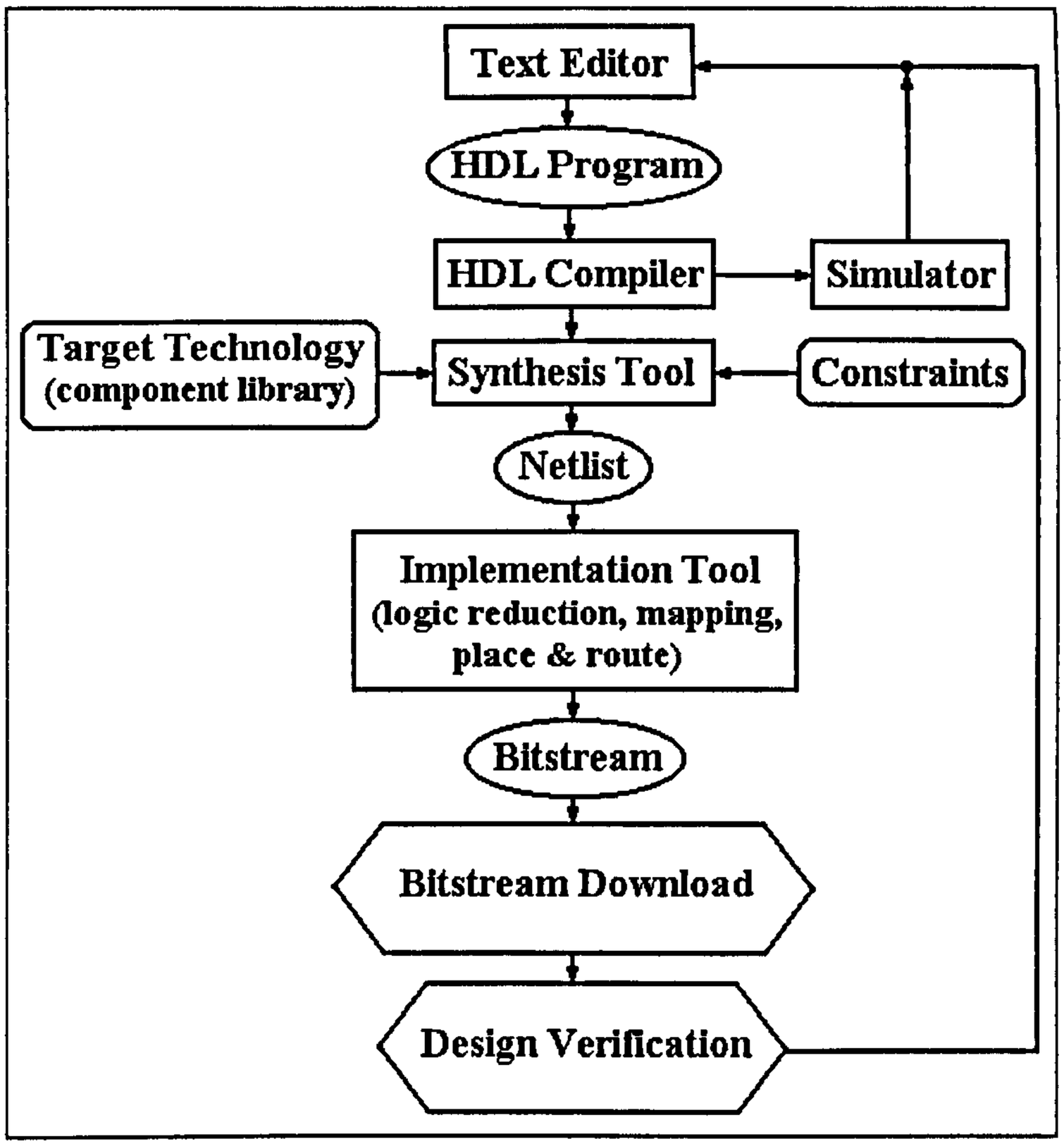

Fig. 2-21 – FPGA design flow from HDL to final implementation

One of the latest significant shifts in design methodology is the principle of design reuse. The only efficient way to create complex sub-micron ICs is to employ large blocks that have been previously designed, and then to integrate them into an ASIC architecture which also includes the original sections of the new design. This way, the designer can focus on areas of the architecture where it truly adds value in terms of the product's target market, and leave the rest to pre-designed blocks that deliver the more routine functions in a predictable manner. The pre-designed blocks are usually named Intellectual Property Blocks (IPs) or 'cores'.

Today, intellectual property exists in a variety of types: hard, soft, and firm, each with its own advantages and disadvantages [42], [117]. The advantage of the hard layout is small size, high performance and other optimisations such as low power. Another plus is that the designer knows the timing across the core, since gates and interconnect have been specified. One drawback is that the core must be used as-it-is with no changes. Moreover, the designer is limited with respect to the manufacture technology of the larger design if these cores are used. Firm cores offer a bit more flexibility in that they exist as optimised, synthesised netlists. Their advantage is that they can be optimised for timing during the final place and route. However, the core cannot be combined with

surrounding logic to reduce the total design gate count. Soft cores offer the greatest flexibility since they are supplied in the form of high-level description language that can be synthesised with surrounding logic. Thus, it can be optimised during synthesis to reduce gate count and achieve some desired level of performance versus area. However, its inherent flexibility is also a disadvantage, since the abstract logic of the soft core must be verified along with the surrounding logic. Developing a test bench to achieve this result is the most difficult part of using a soft core.

3 ELEMENTS OF NEURAL CONTROL

Neural control is a branch of the general field of intelligent control, which is based on the concept of artificial intelligence (AI). AI can be defined as computer emulation of the human thinking process. The AI techniques are generally classified as expert systems (ES), fuzzy logic (FL), artificial neural networks (ANN).

The classical expert systems are based on Boolean algebra and use precise calculations while fuzzy logic systems involve calculations based on an approximate reasoning. Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth - truth values between "completely true" and "completely false" [45]. It was introduced by Dr. Lotfi Zadeh of UC/Berkeley in the 1960's as a means to model the uncertainty of natural language. The truth of a logical expression in fuzzy logic is a number in the interval [0,1]. Fuzzy Logic has emerged as a profitable tool for the control of complex industrial processes and systems. It is used for processes with no simple mathematical model, for highly non-linear processes, or if the processing of linguistically formulated knowledge is to be performed. Although it was invented in the United States, the rapid growth of this technology started from Japan and has now again reached the USA and Europe. The controllers based on this mathematical approach are known as fuzzy controllers.

The use of artificial neural networks (ANN) is the most powerful approach in AI. ANNs are information processing structures whose architecture and operation are inspired biological nervous tissue. Any ANN is a system made up of several basic entities (named neurones) which are interconnected and operate in parallel transmitting signals to one another in order to achieve a certain processing task [139]. One of the most outstanding features of ANNs is their capability to simulate the learning process. They are supplied with pairs of input and output signals from which general rules are automatically derived so that the ANN will be (in certain conditions) capable of generating the correct output for a signal that was not previously used. The neural

approach can be combined with the fuzzy logic generating neuro-fuzzy systems that combine the advantages of the two control paradigms.

3.1 Neurone Types

The operation of the artificial neurones is inspired by their natural counterparts [62]. Each artificial neurone has several inputs (corresponding to the synapses of the biological neurones) and one single output, the axon. Each input is characterised by a certain weight indicating the influence of the corresponding signal over the neurone output. The neurone calculates an equivalent total input signal as the weighted sum of the individual input signals (3-1).

$$net = \sum_{i=1}^{n} w_i \cdot x_i \qquad (3\text{-}1)$$

The resulting quantity is then compared with a constant value named the threshold level and the output signal is calculated as a function of their difference (net-t). This function is named the transfer function or the activation function. The input weights, the threshold level and the activation function are the parameters which completely describe an artificial neurone. Depending on the type of the artificial neurone the activation function may have several forms [63]. There are analogue neurones using continuous real activation functions and discrete neurones whose activation functions are discontinuous. Bipolar neurones generate both positive and negative outputs while unipolar ones generate only positive values.

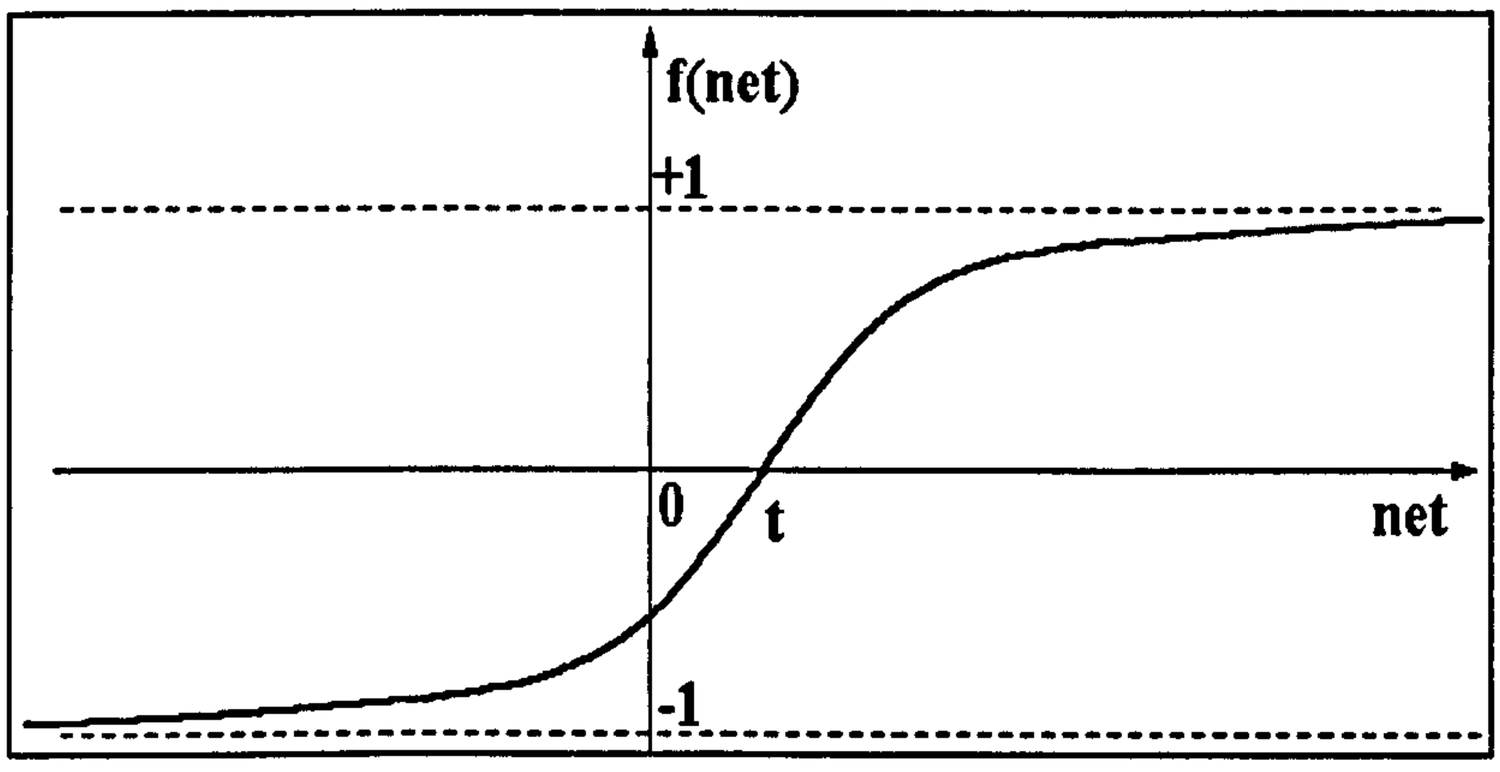

Fig. 3-1 - Sigmoidal activation function of bipolar analogue neurones (λ=1)

In case of bipolar analogue neurones, the most popular activation function is given by equation (3-2) [140]. The output varies continuously between -1 and +1, depending on the input signals, that can have any real value (Fig. 3-1). Parameter λ is a constant controlling the slope of the activation function's graph. Some authors consider λ=1 to simplify the calculations while others operate with the more general format presented in

(3-2) but the fundamental results and properties of the corresponding ANNs remain valid in both situations.

The function in (3-2) is part of a larger transfer function class called "sigmoidal functions". What they have in common is the graph shape and the property to be derivable, which is essential in some applications.

$$f_1(net) = \frac{2}{1+\lambda \cdot e^{-(net-t)}} - 1 \qquad (3\text{-}2)$$

An alternative activation function is presented in (3-3). It is part of the sigmoidal functions group and, as shown in (3-4), it has the same limit values as function f_1.

$$f_2(net) = \frac{1-\lambda \cdot e^{-(net-t)}}{1+\lambda \cdot e^{-(net-t)}} \qquad (3\text{-}3)$$

$$\begin{cases} \lim\limits_{net \to +\infty} f_2(net) = +1 \\ \lim\limits_{net \to -\infty} f_2(net) = -1 \end{cases} \qquad (3\text{-}4)$$

Unipolar analogue neurones are similar to bipolar ones with the difference that the output signals can only take values between 0 and +1 (Fig. 3-2). Their activation function is described by (3-5).

$$f_3(net) = \frac{1}{1+\lambda \cdot e^{-(net-t)}} \qquad (3\text{-}5)$$

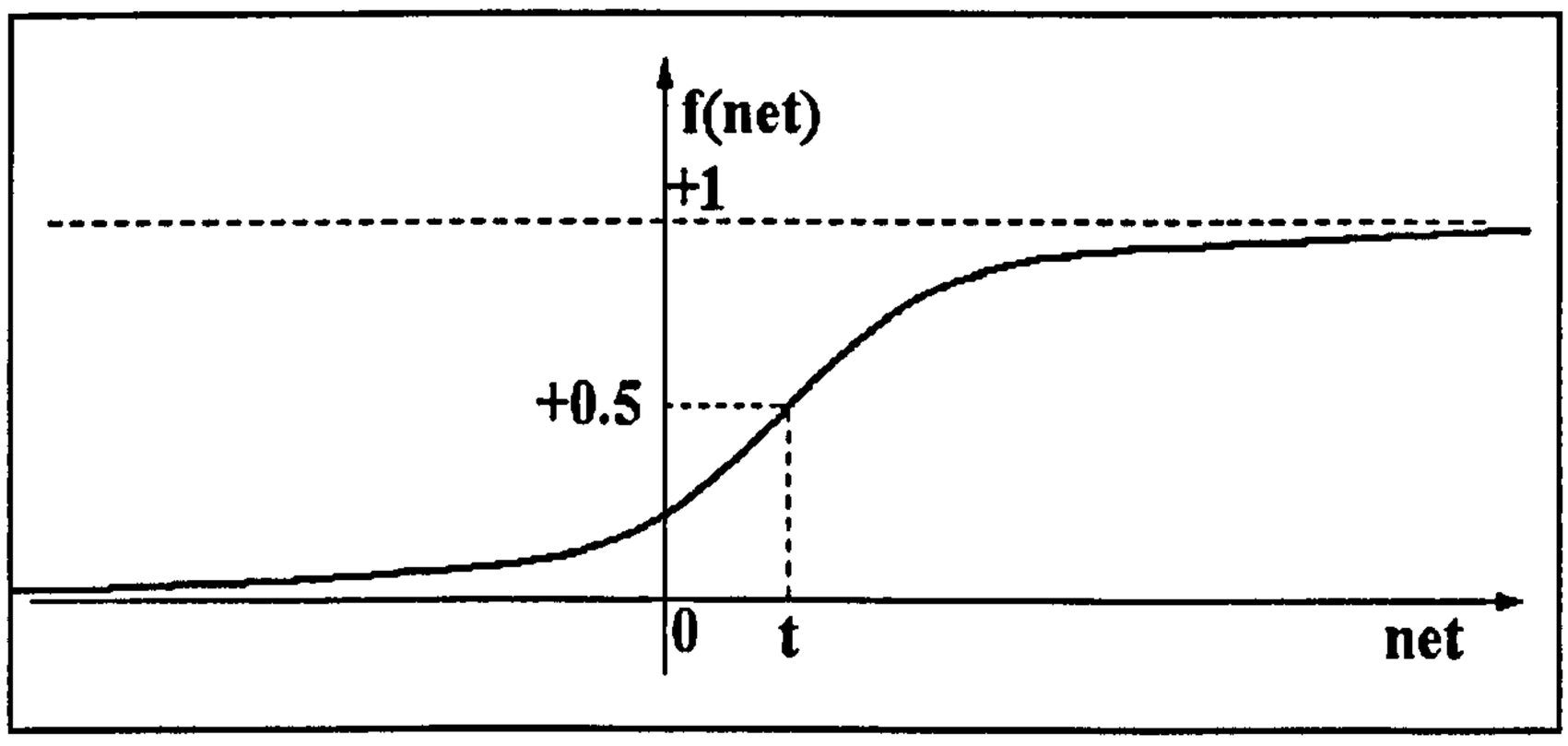

Fig. 3-2 - Sigmoidal activation function for unipolar analogue neurones (λ=1)

Not all continuous activation functions are sigmoidal functions [90], [139]. For instance the stepwise-linear activation function presented in (3-6) is not derivable in two points: net=t-1.0 and net=t+1.0 (Fig. 3-3).

$$f_4(net) = \begin{cases} -1 & \text{if } net < t - 1.0 \\ net - t & \text{if } net \in [t-1.0;\, t+1.0] \\ +1 & \text{if } net > t + 1.0 \end{cases} \qquad (3\text{-}6)$$

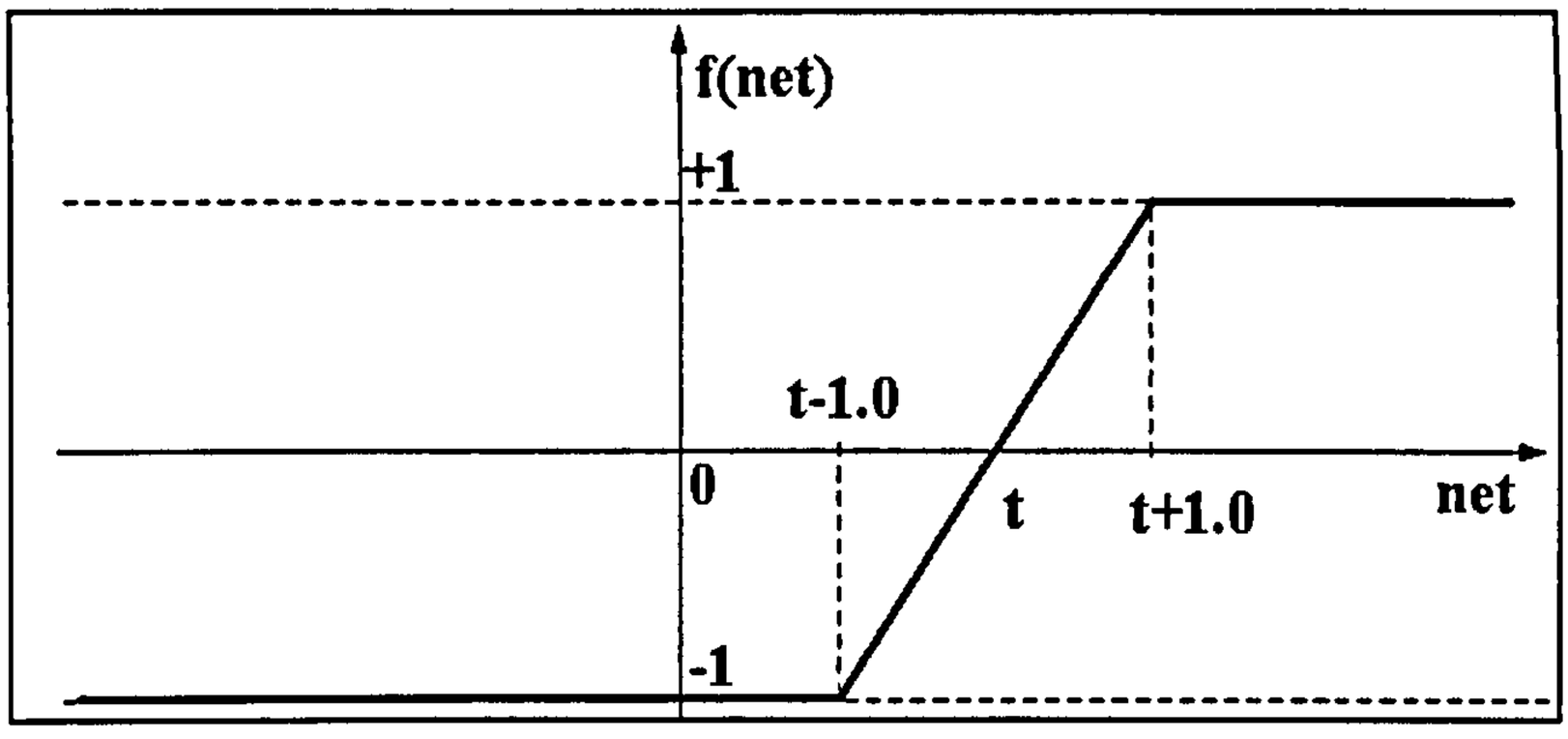

Fig. 3-3 - Non-sigmoidal transfer function

Discrete neurones use threshold type activation functions. The bipolar discrete sort is associated with the activation function described in (3-7) while the unipolar type uses the activation function illustrated by (3-8). These two functions can be considered limiting cases ($\lambda \to \infty$) of the sigmoidal transfer functions presented in (3-2) and (3-5).

$$f_5(net) = \begin{cases} 1 & net \geq t \\ -1 & net < t \end{cases} \tag{3-7}$$

$$f_6(net) = \begin{cases} 1 & net \geq t \\ 0 & net < t \end{cases} \tag{3-8}$$

Over the last few years, more sophisticated types of neurones and activation functions have been introduced in order to solve different sorts of practical problems. In particular, radial basis neurones have proved very useful for many control system and system identification applications [74], [139]. These neurones use so called radial basis activation functions. Equation (3-9) presents the most often used form for such a function, where 'x' is the n-dimensional vector of input signals and 't' a constant vector of the same dimension while $\|\cdot\|$ is the Euclidean norm in the n-dimensional space.

$$f_7(x) = \exp\left(-\|x - t\|^2\right) \tag{3-9}$$

Practically f_7 shows how close vector 'x' is to vector 't' in this n-dimensional space. The closer x is to t, the larger is $f_7(x)$; if x=t then $f_7(x)=1$. The classical Gaussian bell is obtained for the unidimensional case while the two-dimensional case is illustrated by Fig. 3-4. Obviously, such a neurone type is very far from the biological model, but this is irrelevant since it proves useful for certain technical applications.

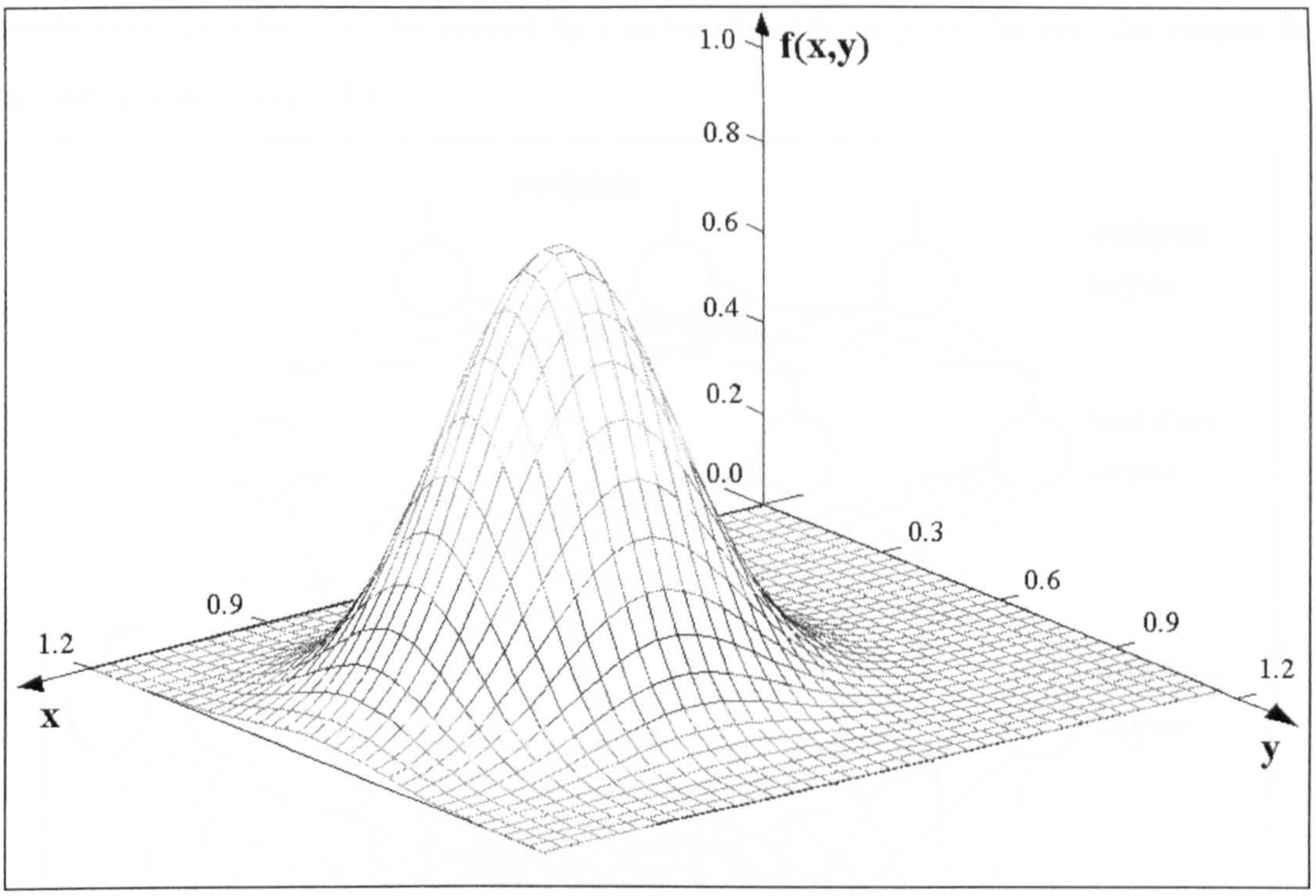

Fig. 3-4 - Radial basis activation function: two-dimensional case

3.2 Architectures of Artificial Neural Networks

Artificial neural networks differ by the type of neurones they are made of and by the manner of their interconnection. There are two major classes of neural networks: feed-forward ANNs and recurrent ANNs. Feed-forward artificial neural networks (FFANN) are organised into cascaded layers of neurones. Each layer contains neurones receiving input signals from the neurones in the previous layer and transmitting outputs to the neurones in the subsequent layer. The neurones within a layer do not communicate to one another. The first network layer is named the input layer, while the last one is named the output layer. All the other neurone layers are known as the hidden layers of the neural network.

FFANN do not have any memory of the past inputs so that they are used for applications where the output is only a function of the present inputs. Therefore, each input vector is simply associated with an output vector. If step activation functions are used, several analogue or discrete input vectors can be associated with a single discrete output vector. Such neural networks are used to solve classification problems. In a classification problem, the set of all possible input vectors is divided into several arbitrary subsets. Each subset is a class. The problem consists of finding out to which class a given input vector belongs. The neural network associates each class with a binary vector and generates the corresponding code for any input vector. Any

classification problem can be solved by a network with only two layers (the output layer plus one hidden layer) [43].

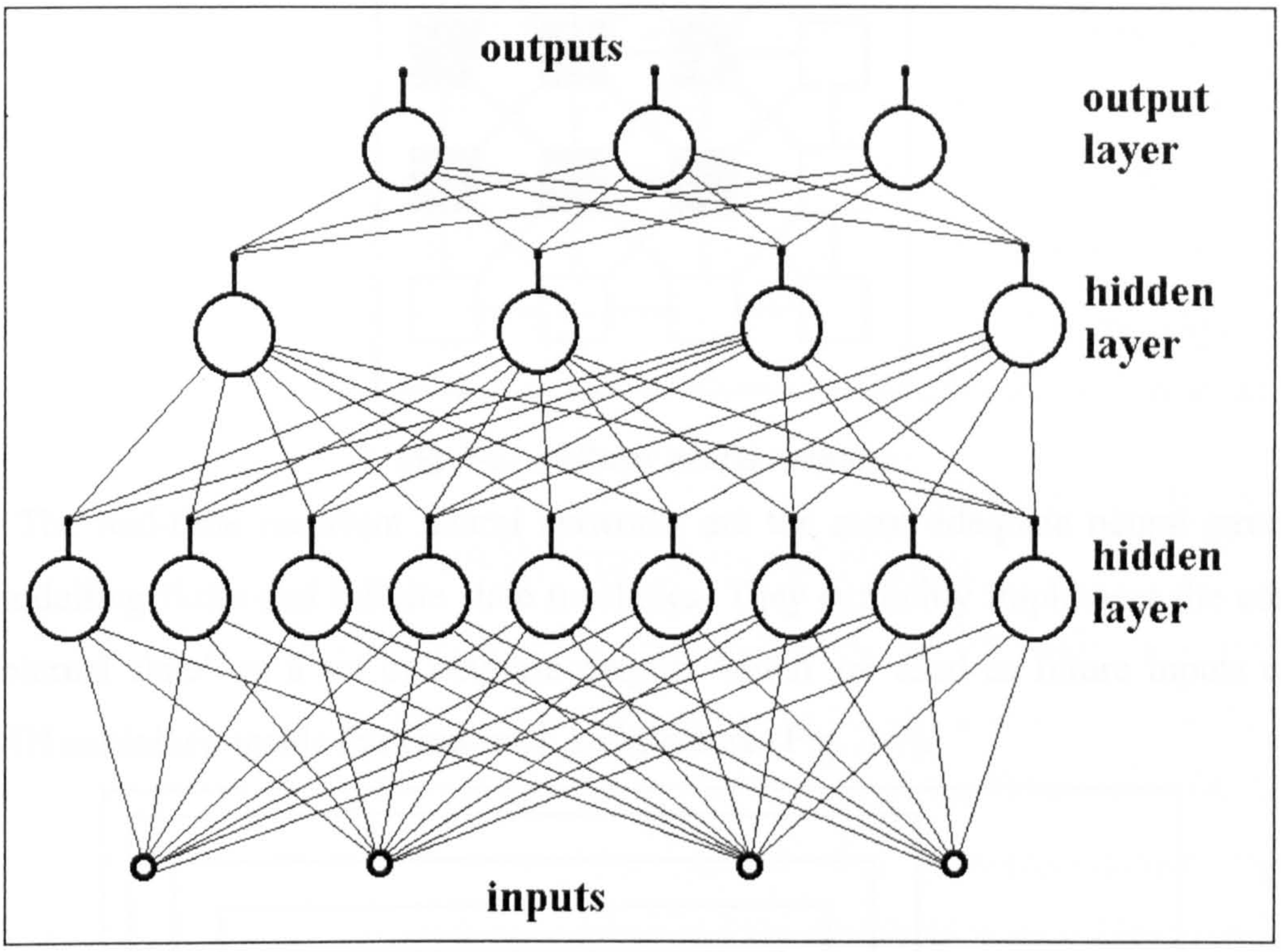

Fig. 3-5 - Feed- forward neural network architecture

Recurrent artificial neural networks include architectures where neurones in the same layer communicate (cellular neural networks) or architectures where some of the outputs of a FFANN are used as inputs (real-time recurrent networks, Hopfield networks). These neural architectures can be described either by continuous time models or by discrete time models.

The concept of cellular neural network (CNN) was first introduced by Chua and Yang (1988). They are a special class of recurrent neural networks, which consist of cells connected only to the cells their neighbourhood (Fig. 3-6). Thus, the main feature of CNNs is the fact that information is directly exchanged just between neighbouring cells. Due to this local interconnection property, CNNs have been considered particularly suitable for VLSI implementations for high-speed parallel signal processing. CNNs are used in several application areas: image processing, artificial vision, associative memories, biological systems modelling, etc.

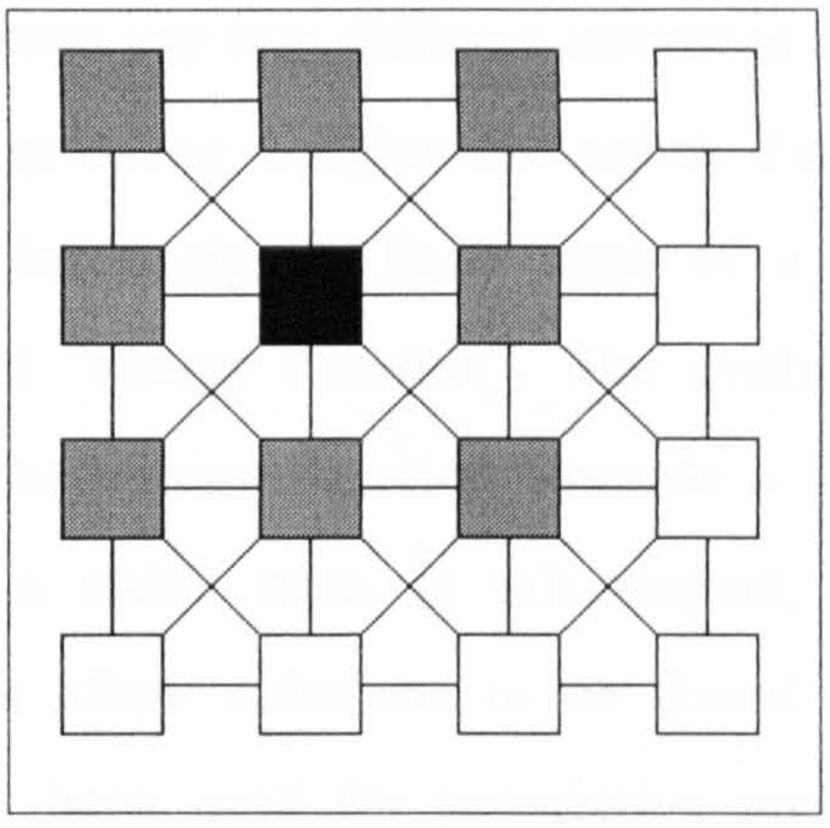

Fig. 3-6 – Cellular neural network

The real-time recurrent neural networks are the most adequate neural structures for modelling finite and infinite state machines. They explicitly implement the concept of "internal state" as a set of neurone outputs which are used as future inputs of the FFANN contained inside the feed-back architecture (Fig. 3-7).

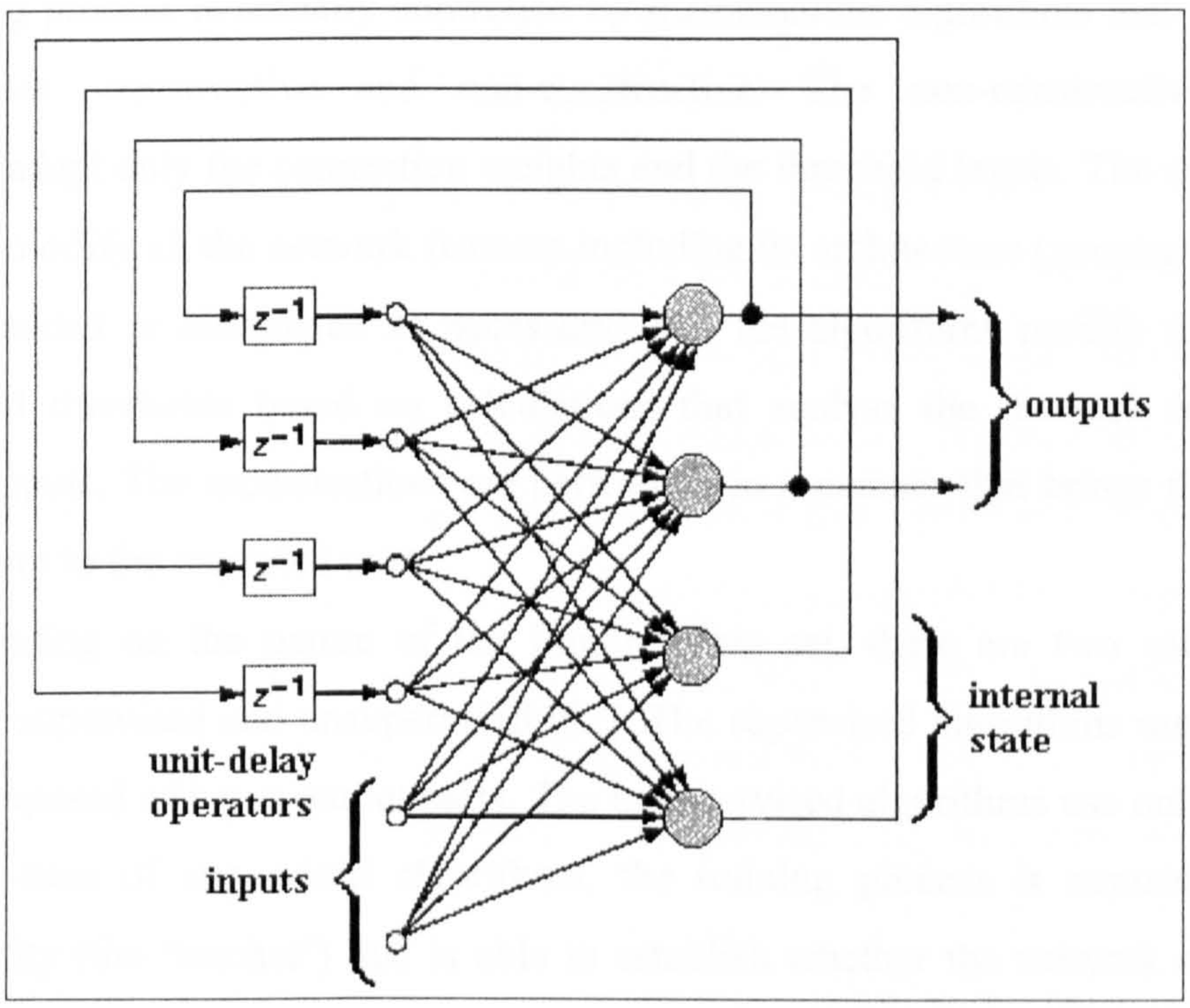

Fig. 3-7 – One-layer real-time recurrent neural network (discrete time model)

The discrete time models contain delay units on the feed-back connections, while the continuous-time models contain low-pass filters (usually first-order elements). Finite state machines are modelled by discrete-time models involving neurones with step activation functions, while infinite state machines are modelled by continuous time models containing neurones with sigmoidal activation functions.

Hopfield networks are a particular case of recurrent neural networks that contain only one layer of neurones and there is no feed-back loop between any neurone and

itself. The connections between any two different neurones are symmetrical in Hopfield networks, that is the corresponding weights are equal. Furthermore, each neurone is connected to an external input signal. Each state of a Hopfield network can be characterised by a so-called "energy function". The evolution of the network's state determines a decrease of the energy function towards a local minimum. Each local minimum is associated to a stable state. In this respect, Hopfield networks can be configured in manners that allow solutions to be found for particular optimisation problems. This feature has been used for associative memory applications, and the optimisation of the power dispatch in the power systems [49].

3.3 Training Algorithms

One of the most important feature of neural networks is their ability to learn (to be trained) and improve their operation using a set of examples named training data set. The training process is actually controlled by mathematical algorithms that fall in two main classes: constructive and non-constructive. The non-constructive training algorithms adapt only the connection weights and the threshold levels. The constructive algorithms modify all the network features including its architecture (neurones and even layers are added or eliminated as necessary). All the algorithms modify the neurone weights and thresholds based on calculations that analyse the network response to particular inputs. The modifications are performed in a manner that brings the network outputs closer to the expected ones.

Depending on the nature of the training data set, there are two categories of algorithms: supervised and unsupervised [62]. The supervised algorithms use a training data set composed of input-output pairs. The unsupervised algorithms use only the input vectors. In case of supervised algorithms, the training process is controlled by an external entity (the 'teacher') that is able to establish whether the network outputs are adequate to the inputs and what is the size of the error. Then the network parameters are modified according to the particular correction method defining each training algorithm. In case of unsupervised methods (the Hebbian rule, the "winner takes all" algorithm, etc), there are no means to know what the expected outputs are. The network evolves as a result of the "experience" gained from the previous input vectors. The weight values converge to a set of final values dictated by the input values used as training data set in conjunction with the particular training algorithm.

The unsupervised family of training algorithms is mainly used for signal and image processing, where pattern classification, data clustering or compression

algorithms are involved. The control engineering problems are better tackled by supervised training methods, as the relationship between inputs and desired outputs is better defined and easier to control.

3.3.1 The Error Back-Propagation Algorithm

The most popular supervised training algorithm is the one named "error back-propagation", or simply "back-propagation". It involves training a FFANN structure made up of sigmoidal activation function neurones. The back-propagation algorithm is a gradient method aiming to minimise the total operation error of the neural network. The total error is a function defined by equation (3-10) where O_i^{ref} is the column vector of the reference outputs and O_i is the column vector of the actual network outputs corresponding to the input pattern number 'i'. The total error Err is the sum of the errors corresponding to all n_p input patterns.

$$Err = \sum_{i=1}^{n_p} \left(O_i^{ref} - O_i\right)^T \cdot \left(O_i^{ref} - O_i\right) = \sum_{i=1}^{n_p} \left\| O_i^{ref} - O_i \right\|^2 \qquad (3\text{-}10)$$

For each training step, the vector of all neurone weights and threshold weights (W) is updated in such a way that the total error Err is decreased. The vector W can be associated to a point in a N_W-dimensional space (the parameter space), where N_W is the total number of weights and thresholds in the neural network. The most efficient way to perform the update is to shift the point W along the curve indicated by the gradient of the total error (∇Err). This principle is illustrated by equation (3-11), where W(t) is the parameter vector during the current training cycle, W(t+1) is the parameter vector for the next training cycle and η is the learning-rate constant. Ideally, the algorithm stops when the total error is zero. In practice, it is stopped when the error is considered negligible.

$$W(t+1) = W(t) - \eta \cdot \nabla Err = W(t) - \eta \cdot \left(\frac{\partial Err}{\partial w_1} \quad \frac{\partial Err}{\partial w_2} \quad \frac{\partial Err}{\partial w_3} \quad \cdots \quad \frac{\partial Err}{\partial w_{N_W}} \right)^T \qquad (3\text{-}11)$$

For the practical calculation of the error gradient ∇Err, the components in the vector W are usually rearranged as a three-dimensional matrix. The matrix has a number of rectangular layers equal to the number of neurone layers in the neural network. Each rectangular layer is a two-dimensional matrix containing one line for each neurone in the corresponding layer of the neural network. Each line includes the input weights and its threshold level of a neurone. Therefore, the element w_{jkm} in the three-dimensional

matrix is the weight 'm' of the neurone 'k' situated in the layer 'j' inside the neural network. The threshold level corresponds to the last element in each line and is not treated any differently to the input weights because it can be considered as an extra weight supplied with a constant input signal –1.

In case of a one-layer network, the components of the error gradient are calculated according to (3-12) where the input signal x_m is the corresponding input signal.

$$\frac{\partial Err}{\partial w_{1km}} = \frac{\partial}{\partial w_{1km}} \sum_{i=1}^{n_p} \left\| O_i^{ref} - O_i \right\|^2 = -\sum_{i=1}^{n_p} 2\left(o_{ik}^{ref} - o_{ik}\right) \cdot \frac{df}{d\left(net_{ik} - t_k\right)} \cdot x_m \tag{3-12}$$

As demonstrated in [140], bipolar sigmoidal activation functions given by (3-2) have the property (3-13), so that the equation (3-12) can be transformed into (3-14).

$$\frac{df}{d(net - t)} = \frac{1}{2}\left(1 - f^2\right) \tag{3-13}$$

$$\frac{\partial Err}{\partial w_{1km}} = -\sum_{i=1}^{n_p} \left(o_{ik}^{ref} - f_{i1k}\right) \cdot \left(1 - f_{i1k}^2\right) \cdot x_m \tag{3-14}$$

If the network has more than one layer then the input signal x_m is actually generated by the neurone 'm' in the layer two so that x_m has to be replaced with f_{2m}. Equations (3-16) and (3-17) illustrate the calculations for the neurones in layers two and three.

$$\frac{\partial Err}{\partial w_{1km}} = -\sum_{i=1}^{n_p} \left(o_{ik}^{ref} - f_{i1k}\right) \cdot \left(1 - f_{i1k}^2\right) \cdot f_{2m} \tag{3-15}$$

$$\frac{\partial Err}{\partial w_{2km}} = -\sum_{i=1}^{n_p} \sum_{x} \left(o_{ix}^{ref} - f_{i1x}\right) \cdot \left(1 - f_{i1x}^2\right) \cdot w_{1xk} \cdot \left(1 - f_{i2k}^2\right) \cdot f_{i3m} \tag{3-16}$$

$$\frac{\partial Err}{\partial w_{3km}} = -\sum_{i=1}^{n_p} \sum_{x} \left(o_{ix}^{ref} - f_{i1x}\right) \cdot \left(1 - f_{i1x}^2\right) \cdot \sum_{y} w_{1xy} \cdot \left(1 - f_{i2y}^2\right) \cdot w_{2yk} \cdot \left(1 - f_{i3k}^2\right) \cdot f_{4m} \tag{3-17}$$

The previous calculations can be generalised for any number of layers [90], [140]. Each component of the gradient is determined following the iterative process (3-18). Similar results are obtained for all types of sigmoidal activation functions. These equations justify the name of the training algorithm: the output errors of the FFANN affect the calculations referring to any weight because their influence propagates back to the inputs from one layer to the next in accordance with (3-18).

$$
\begin{cases}
\dfrac{\partial Err}{\partial w_{jkm}} = \sum_{i=1}^{n_p} \sum_{x} \left(o_{ix}^{ref} - f_{i1x} \right) \cdot \delta_{i1x} \\
\delta_{i1x} = \left(1 - f_{i1x}^2 \right) \cdot \sum_{y} w_{1xy} \cdot \delta_{i2y} \\
\delta_{i2x} = \left(1 - f_{i2x}^2 \right) \cdot \sum_{y} w_{2xy} \cdot \delta_{i3y} \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\delta_{i(k-1)x} = \left(1 - f_{i(k-1)x}^2 \right) \cdot \sum_{y} w_{(k-1)xy} \cdot \delta_{iky} \\
\delta_{ikx} = \begin{cases} 0 & x \neq m \\ \left(1 - f_{i(k-1)x}^2 \right) \cdot f_{km} & x = m \end{cases}
\end{cases}
\tag{3-18}
$$

The back-propagation algorithm faces the well-known problem of any non-linear optimising algorithm using the gradient method: it can become stuck in a local minimum of the objective function (the function 'Err' in this case). Therefore, the back-propagation algorithm is not guaranteed to generate a satisfactory solution for all input-output association problems and FFANN architectures. The training result depends on several factors [140]:

1) Network architecture (number of layers, number of neurones in each layer)

2) Initial parameter values W(0)

3) The details of the input-output mapping

4) Selected training data set (pairs of inputs and corresponding desired outputs).

5) The learning-rate constant η

Back-propagation is not a constructive algorithm; the network architecture has to be chosen in advance. Unfortunately, there is no clearly defined set of rules to be followed in order to decide which is the most appropriate architecture for a problem. Choosing the architecture is a result of a trial and error process supported by previous experience. However, it was mathematically demonstrated that any input-output mapping can be learned by a FFANN with only one hidden layer, provided that the number of neurones in the hidden layer is large enough for the problem to be solved [103]. This means that if a neural network proves incapable of learning how to perform a certain task, than one possible solution is to increase the number of neurones in the hidden layer or layers.

A different solution is to restart the algorithm with another set of initial parameters W(0). This solution is based on the assumption that the previous failure was generated by stopping at a local minimum. The trajectory of vector W in the parameter

space is dependent on its starting point W(0), therefore, the situation may be avoided by changing the initial weights and thresholds.

Another important aspect is choosing an adequate training data set, so that if the number of different input values is finite, the training data set may cover all the possibilities. Nevertheless, if this number is infinite (as it happens when the inputs are analogue signals), or if the number is too large, then only a selection of input combinations will be used to train the neural network. The quality of the training process is influenced by the way the training data set is generated. If the training data set adequately covers all the aspects of the input-output mapping, then the network will be able to generate correct answers for inputs that were not used during the training period.

This property is called "generalisation" and is made possible by the fact that any FFANN actually performs an interpolation in an n-dimensional space (where is 'n' the length of the input vectors) [68]. The interpolation is carried out based on the information provided by the input vectors used during the training period. If the input-output mapping is continuous and smooth, then the network will easily generalise and yield correct answers as a result of a training performed with only few input vectors. If the input-output mapping is rugged and complicated, then a large number of input vectors is required for an adequate training process.

The training process may require hundreds, thousands or even millions of steps of the type described by (3-11). The actual number depends on the nature of the input-output mapping and on the learning-rate constant η. A large value for η accelerates the training process but also increases the chance that the vector W oscillates around the final solution without ever reaching it. A small η increases the chances to obtain the desired solution but also increases the necessary number of training cycles.

3.3.2 Algorithms Derived from the Back-Propagation Method

A series of new algorithms have been derived in the last two decades from the classical back propagation method. They bring improvements to the training process by accelerating the convergence and improving the chances of finding a good solution for particular application types. The improvements proposed can be summarised as:

1) The learning-rate constant η is varied after each training cycle. It starts with a large value that is progressively diminished during the training process. Therefore, the training process is fast at the beginning but the final oscillations are avoided because η decreases during the training process.

2) Every adjustable network parameter has its own learning-rate constant η_i. The back-propagation algorithm may be slow, because the use of a unique learning-rate parameter may not suit all the complicated error variations in the N_W-dimensional parameter space. Thus, a learning-rate value that is appropriate for the adjustment of one weight is not necessarily appropriate for the adjustment of another. Thus, the learning algorithm is described by equation (3-19).

$$W(t+1) = W(t) - \left(\eta_1 \frac{\partial Err}{\partial w_1} \quad \eta_2 \frac{\partial Err}{\partial w_2} \quad \eta_3 \frac{\partial Err}{\partial w_3} \quad \cdots \quad \eta_{N_w} \frac{\partial Err}{\partial w_{N_w}} \right)^T \tag{3-19}$$

3) If one learning-rate is associated with each network parameter, then all the learning-rates are allowed to vary from one training cycle to the next. The variance may be calculated according to point 1). More sophisticated methods may calculate the learning-rate constants based on the error function partial derivatives. Therefore, η_i is large if the influence of w_i over the error is small and η_i is small otherwise (3-20).

$$\eta_i = \frac{1}{\left| \frac{\partial Err}{\partial w_i} \right| + K} \qquad \text{where } K > 0 \tag{3-20}$$

4) If the sign of the error derivative $\partial Err/\partial w_i$ oscillates for several consecutive iterations, the corresponding learning-rate parameter η_i is decreased.

5) The convergence of the training is accelerated by supplementing the current weight adjustment with a fraction of the previous weight adjustment, as shown in equation (3-21). This algorithm is named the momentum method [140] and the second term indicating the fraction of the most recent weight adjustment is called the momentum term. The momentum term α is a user-selected constant with values between 0.1 and 0.8.

$$W(t+1) = W(t) - \eta \cdot \nabla Err + \alpha [W(t) - W(t-1)] \tag{3-21}$$

Real-time recurrent neural networks need to be trained in such a manner that they learn a certain temporal correlation between inputs and outputs. A promising training method applicable to such situations and named the dynamic back-propagation training [63] has been derived from the classical one. The main feature of the new method is that input vectors are not applied randomly, but in rigorously defined series. The expected outputs depend both on the current input and on past inputs, while the error calculation is performed globally for the entire temporal series of input vectors.

3.3.3 Training Algorithms for Neurones with Step Activation Functions

If the activation functions of the neurones in FFANN are not sigmoidal, the back-propagation algorithm cannot be used because the error function cannot be derived. However, two other recursive methods presented in (3-22) and (3-23) are applicable to the FFANNs with only one layer. These are recursive methods like the back-propagation algorithm, but in this case, the training process always has finite number of cycles, provided that the desired input-output relation can be learned by a one layer network.

$$w_{jk}^{t+1} = w_{jk}^{t} + \eta \cdot \sum_{i=1}^{n_p} x_k \cdot \left(o_{ij}^{ref} - f_{ij}\right) \tag{3-22}$$

$$w_{jk}^{t+1} = w_{jk}^{t} + \eta \cdot \sum_{i=1}^{n_p} x_k \cdot \left(o_{ij}^{ref} - net_{ij}\right) \tag{3-23}$$

Finding the correct weights for a multilayer FFANN with step activation functions is a complicated problem. The previous two methods cannot be generalised for such networks. Either constructive methods or genetic algorithms need to be used instead.

3.3.4 The Voronoi Diagram Algorithm

The Voronoi diagram is a constructive algorithm applicable to FFANNs composed of neurones with step activation function [34]. As previously mentioned, any FFANN containing step activation function neurones solves a classification problem. The Voronoi diagram is a graphical representation of the classification problem to be implemented by the FFANN. Let us consider the m-dimensional space of the input data and a set of points in this space, corresponding to a given set of input vectors. The Voronoi diagram (also known as Thiessen polygons or Drichelet tessallation) is a partitioning of the m-dimensional space into convex regions called Voronoi cells, each of which defines the region of influence of one given point in its interior. Any Voronoi cell can be defined as the intersection of a finite number of half-spaces and is therefore delimited by a finite number of hyperplanes.

Each hyperplane can be modelled by one neurone with a step activation function such as (3-7) or (3-8). In the unipolar situation, the neurone generates the output signal '1' for the inputs corresponding to points on a given side of the hyperplane, while '0' is generated for all the other inputs. As illustrated by (3-24), there is a one-to-one correspondence between the algebraic parameters defining the hyperplane and the

neurone parameters. The same applies to bipolar neurones, but the output '0' is replaced by '-1'.

$$\left(\sum_{j=1}^{m} w_j \cdot x_j\right) - t \geq 0 \Leftrightarrow net - t \geq 0 \Leftrightarrow f(net) = 1$$
$$\left(\sum_{j=1}^{m} w_j \cdot x_j\right) - t < 0 \Leftrightarrow net - t < 0 \Leftrightarrow f(net) = 0 \qquad (3\text{-}24)$$

A Voronoi cell is defined by its borders. Consequently, a point in the input data space belongs to a certain Voronoi cell only if all the corresponding neurones simultaneously generate the required outputs. Thus, the set of convex cells in a Voronoi diagram can be modelled by a FFANN with two layers. The input layer contains the neurones modelling the hyperplanes and the second layer contains one neurone for each convex cell. All the neurones defining the borders of a particular cell feed the corresponding neurone in the second layer.

The classes defined by a classification operation are not necessarily convex. Therefore, one class may be the union of several Voronoi cells. As a result, a third layer is necessary in the corresponding neural network. The third layer contains one neurone for each class of input vectors. Each neurone is connected only to those neurones in the second layer implementing Voronoi cells that are part of the given input vector class.

Fig. 3-8 illustrates a Voronoi diagram example built for a neural network with two inputs and one output. Thus, the diagram is two-dimensional and the hyperplanes are straight lines. The shaded areas cover the Voronoi cells that belong to the class '1', the other cells are part of class '0'. There are four Voronoi cells in Fig. 3-8: r_a, r_b, r_c, r_d that belong to class '1', and they are bounded by 9 lines modelled by neurones n1 through n9. Therefore, the first neurone layer contains 9 neurones, the second contains 4 neurones (one neurone for each of the Voronoi cell) whereas the third layer contains a single 4-input neurone (Fig. 3-9). The outputs of the neurone in the third layer is '1' when X_1 and X_2 correspond to a point in one of the shaded areas and it is '0' for all the other cases.

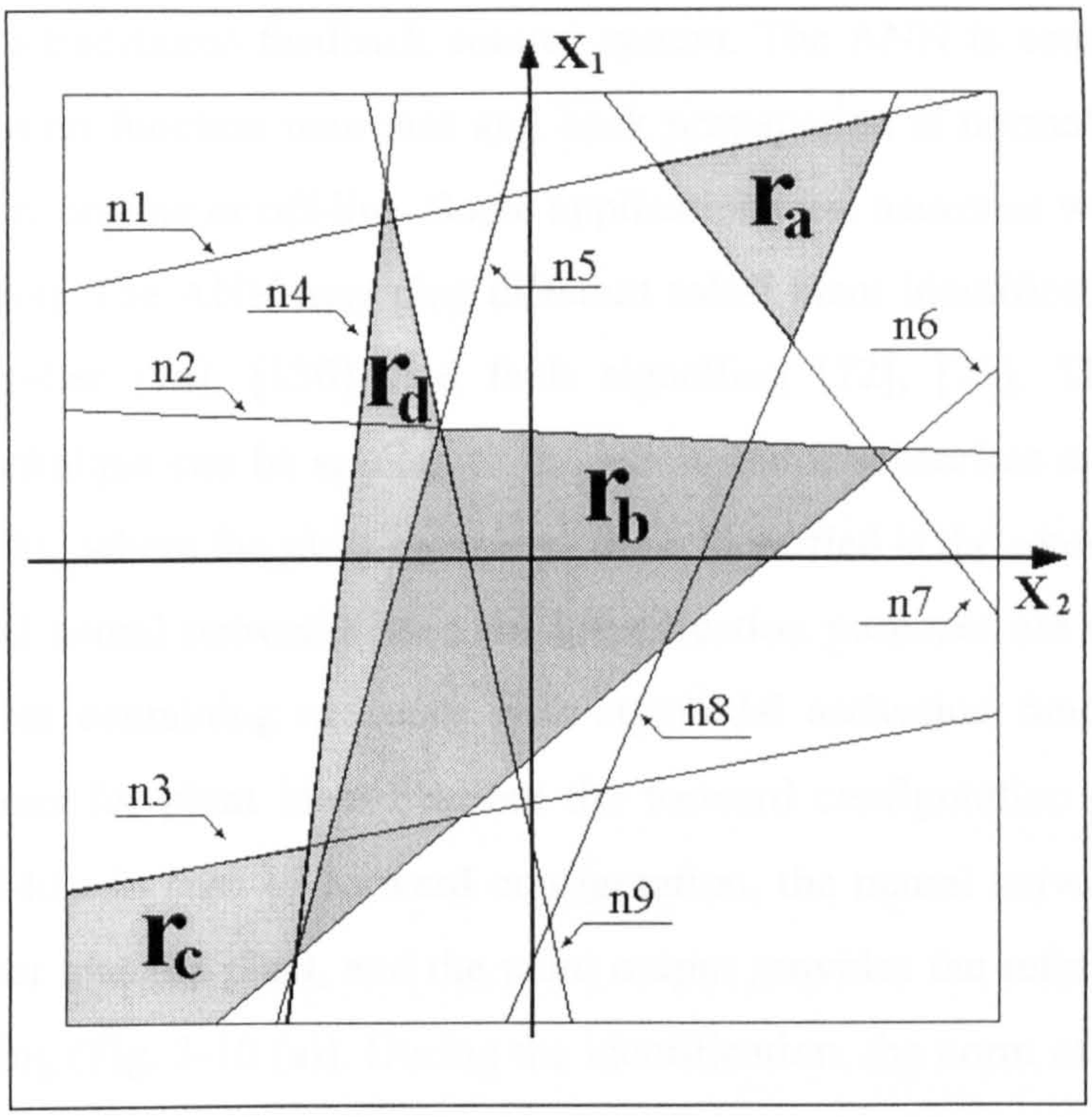

Fig. 3-8- The Voronoi diagram for a 2D example

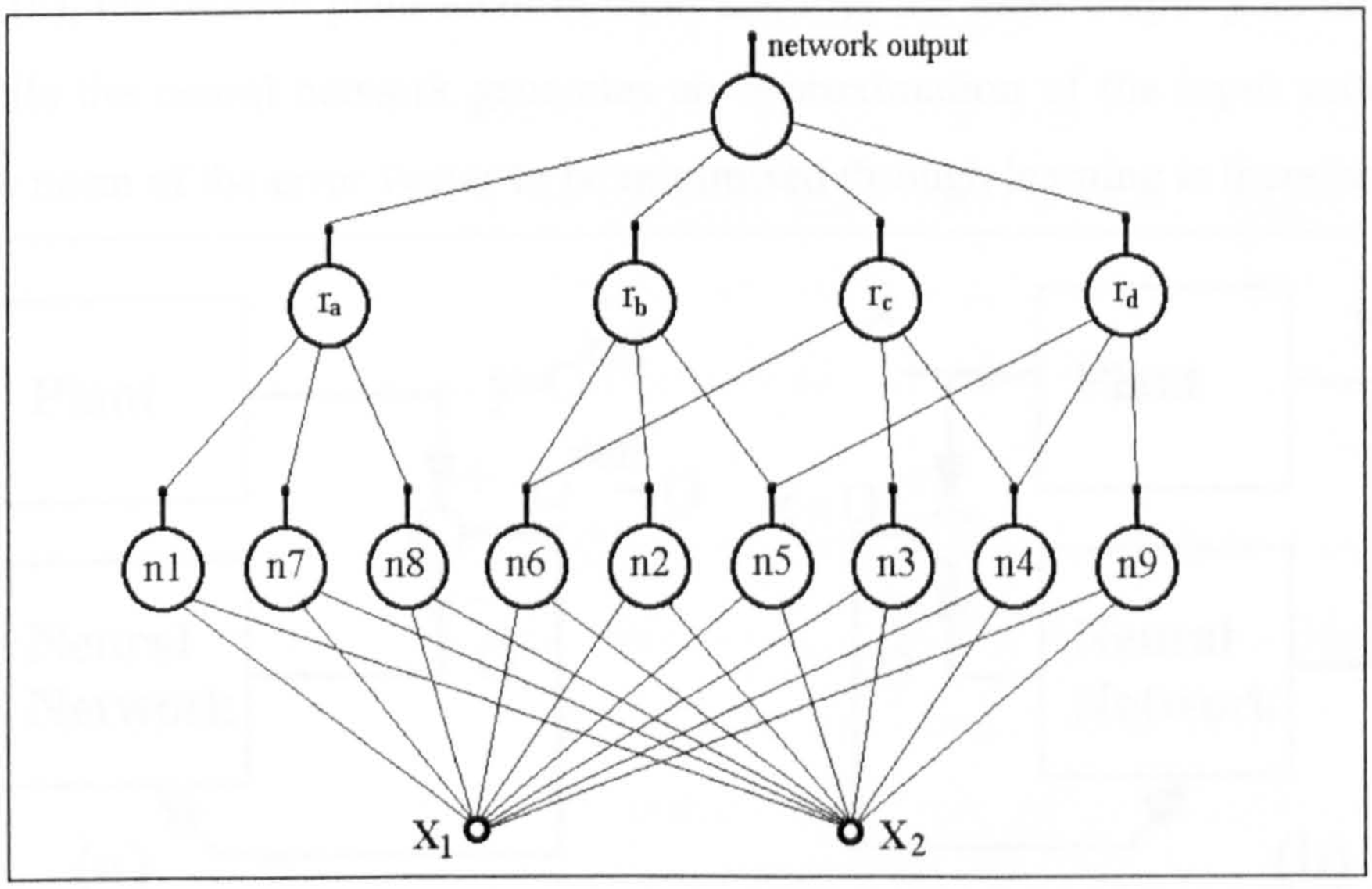

Fig. 3-9 - The neural architecture based on the Voronoi diagram in Fig. 3-8

Very efficient computer algorithms have been developed for the construction of Voronoi diagrams in high dimensional space [46]. They are able to solve this class of problems in linear time and this performance aspect provides tremendous impetus for further research on this topic.

3.4 Control Applications of ANNs

In the recent years, neural solutions have been suggested for many industrial systems using either feed-forward or recurrent neural networks. Most of the published papers describe control system applications built around a feed-forward neural network

included inside a traditional feedback control system. The ANN is usually made up of sigmoidal activation function neurones and back propagation is normally used to train the network either on-line or off-line. Some applications use neurones with a radial base activation function. The ANN may play different roles: plant identification [58], [119], non-linear controller [74], [130], and fault signalling [72], [71]. The neural plant identification technique can be applied to induction motor sensorless speed estimation, for example in [31] where the plant parameter to be identified is the rotor speed.

The typical neural networks used for identification purposes are multilayer feed-forward structures containing neurones with sigmoidal activation function. There are two configurations for plant identification: the forward configuration and the inverse configuration [140]. In case of forward configuration, the neural network receives the same input vector x as the plant, and the plant output provides the reference output O^{ref} during the training (Fig. 3-10 (a)). During the identification, the norm of the error vector $\|O^{ref}-O\|$ is minimised using the back-propagation algorithm. As illustrated in Fig. 3-10 (b), the inverse plant identification employs the plant output y as the network input, while the neural network generates an approximation of the input vector of the plant. The norm of the error vector to be minimised through learning is therefore $\|x-O\|$.

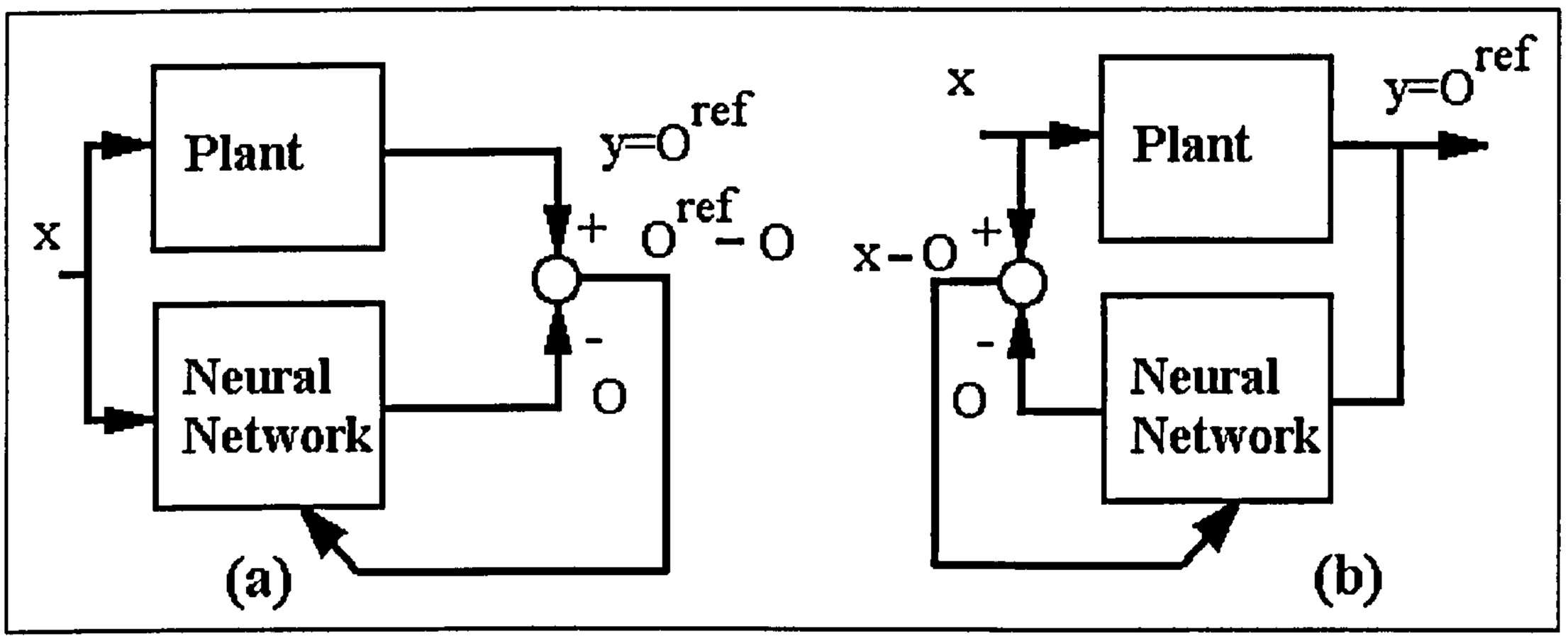

Fig. 3-10 – Neural network configuration for plant identification: (a) forward plant identification; (b) inverse plant identification

Feed-forward neural networks generate instantaneous response. Thus, they can model the steady-state operation of the plant but are not directly capable of modelling its dynamic behaviour. To account for the plant dynamics, the FFANN has to be supplied with a series of past inputs of the plant. Such an approach requires that the neural network is interfaced with a shift register that stores the time series of input vectors (see Fig. 3-11). The shift register is updated at each operation step. An update consists of storing the most recent input vector and discarding the oldest input vector.

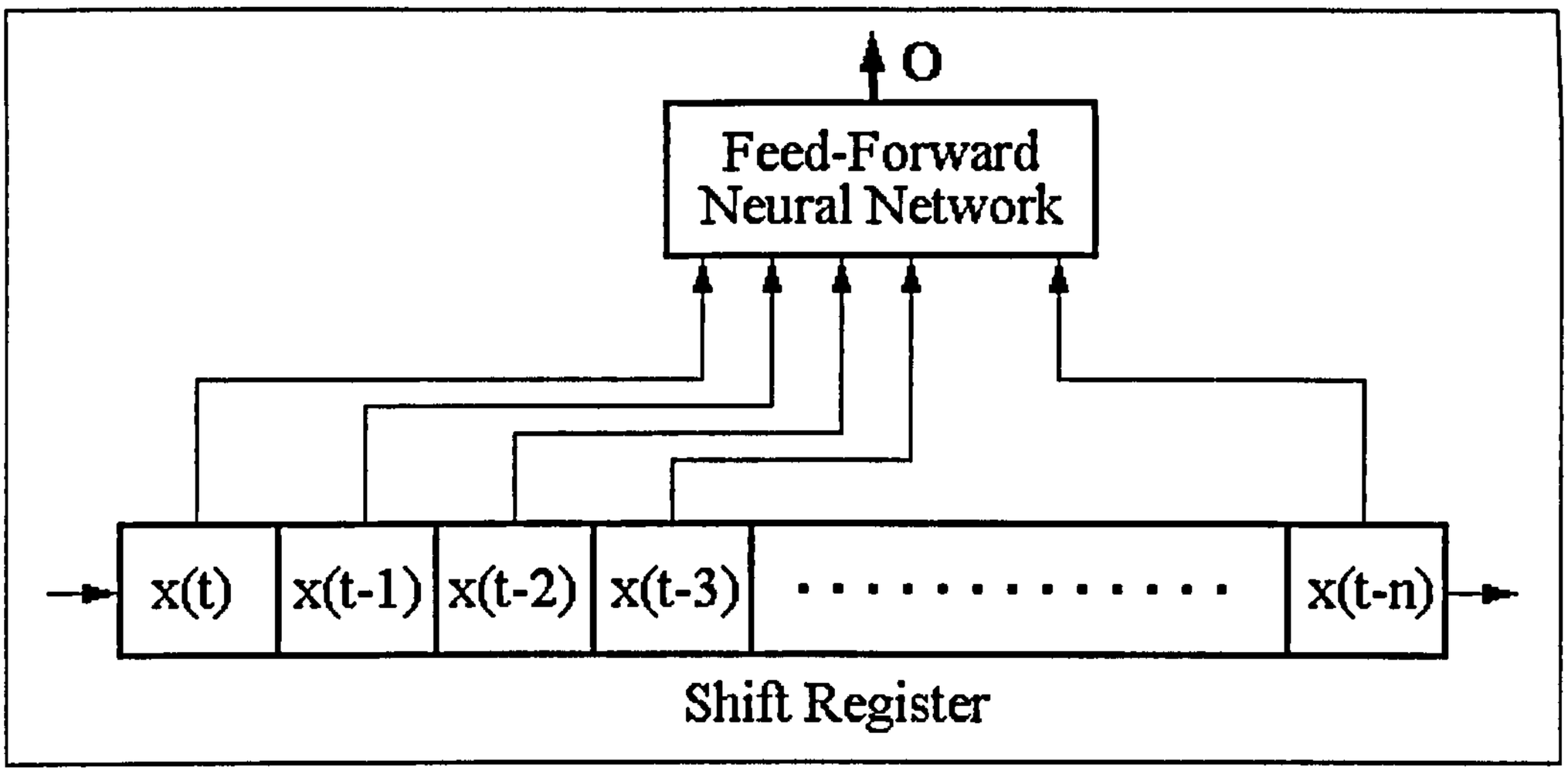

Fig. 3-11 – Neural network interfacing for modelling the plant dynamics

An alternative solution is to use recurrent neural networks. This solution is purely neuronal in that it does not require a shift register. However, most of the control systems have used the first solution so far, because the dynamic back-propagation algorithm requires more computation resources than its static counterpart.

Both identification configurations have advantages and disadvantages. Forward plant identification is always feasible, but it does not immediately allow for the construction of the plant control. In contrast, plant inverse identification facilitates simple plant control. However, the identification itself is not always feasible because in some cases more than one vector x corresponds to a certain vector y (or series of such vectors).

Fig. 3-12 presents a basic control system using a neurocontroller. There are two alternatives: either the neural network is trained only off-line in an inverse identification configuration, as presented in Fig. 3-10 (b), or it is initially trained off-line but the training continues on-line in the control system. Shift registers are used, both during the off-line identification process and inside the control system, to enable the modelling of the dynamic plant behaviour. The neurocontroller input consists of the most recent plant outputs plus the output reference for the current time. Therefore, at each operation step, it generates a control vector O that causes the plant to produce the expected output y^{ref}.

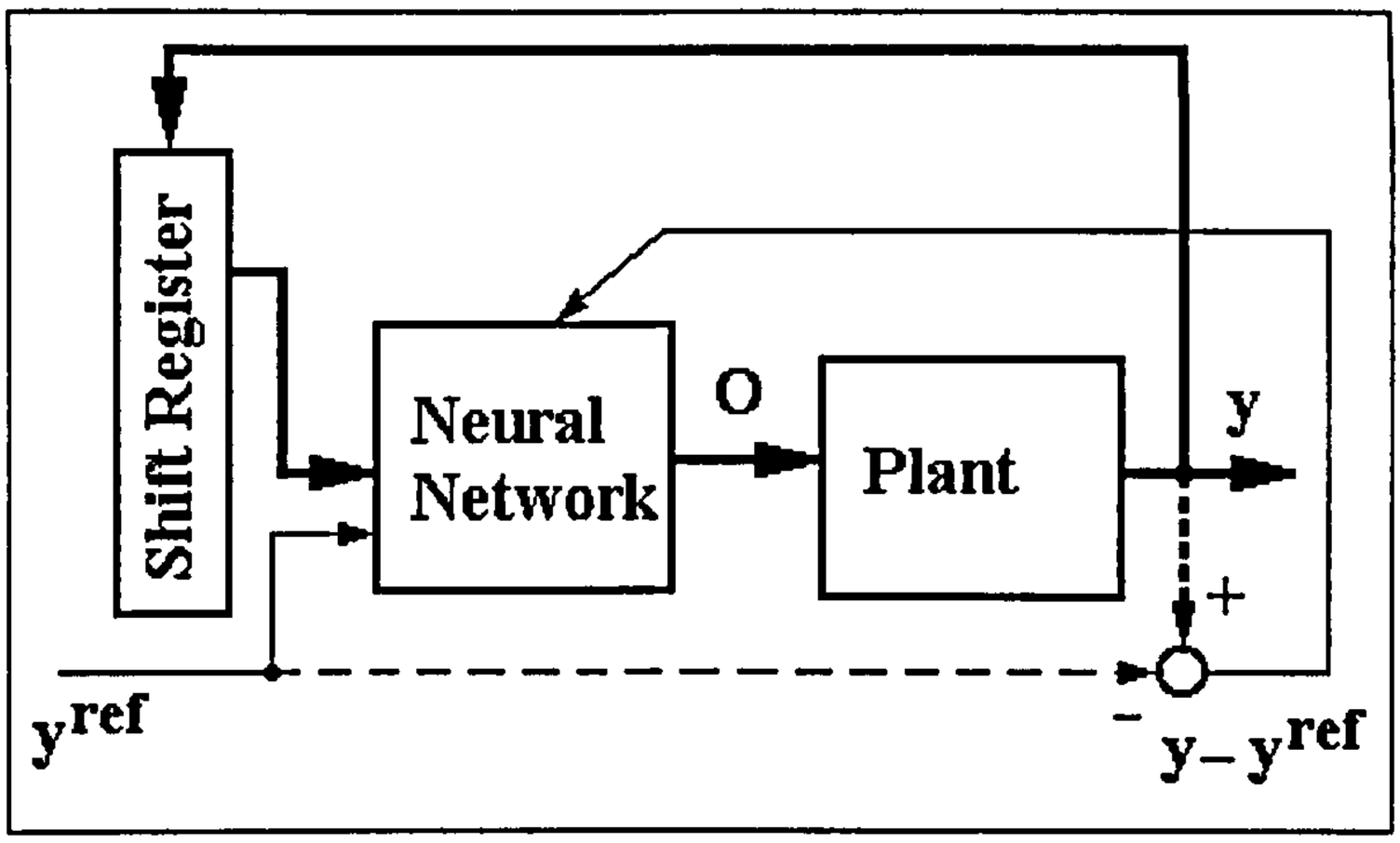

Fig. 3-12 – Basic control system configuration using a feed-forward neurocontroller

The fault signalling applications are part of the larger class of classification applications. The task of the neural network is to analyse the input data and to generate information about the operation of the plant: normal operation, or abnormal operation. In the second case, it may give further details about the abnormality: short-circuit, surpassing voltage or speed limits etc. The neural network is of the feed-forward type and is trained off-line using experimental data that reflects all possible operation modes of the plant.

3.5 NEURAL NETWORK IMPLEMENTATION METHODS

Hardware implemented neural networks are essentially arrays of interconnected processing units that operate concurrently. Each unit has a simple internal structure that, in some cases, includes a small amount of local memory. The most important design issues concerning any neural network hardware implementation are the degree of parallelism, the information processing performance, the flexibility and the silicon area. There are several categories of neural network hardware implementation [95]:

1. Analogue implementation
2. Digital implementation
3. Hybrid implementation
4. Optical implementation

3.5.1 Analogue Hardware Implementation

Analogue neural networks can exploit physical properties of silicon devices to perform network operations obtaining very high processing speed. However, analogue design can be very difficult because of the need to compensate for parameter variations with temperature, manufacturing conditions, etc. One approach is to implement

neurones using common operational amplifiers and resistors [140]. The operational amplifier implements the activation function, while the resistors determine the weight values (Fig. 3-13). The amplifier output voltage V_{out} depends on the input voltages V_+ and V_- that, in turn, depend both on the input voltages and on the resistors values. Ohm laws are used to perform all the necessary calculations.

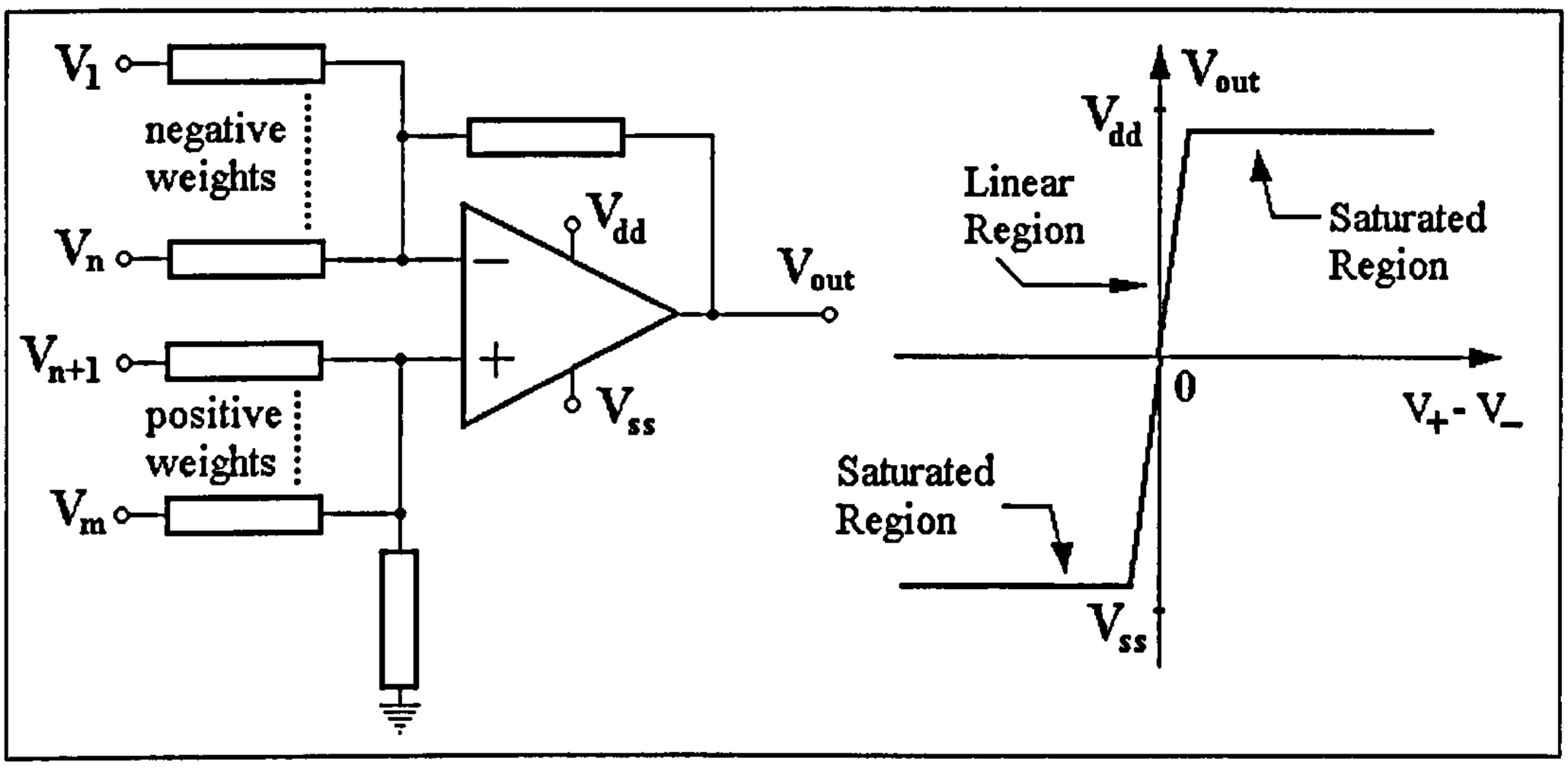

Fig. 3-13 – Neurone implementation using operational amplifiers and resistors

The implementation style using resistors ensures very good linearity but it is not flexible because the weight values are set during the manufacturing process and they cannot be altered afterwards. Creating a changeable analogue synapse involves the complication of analogue weight storage. The simplest approach is to replace the fixed value resistors by MOS transistors that can operate as voltage adjustable switches. Each transistor is controlled by a voltage V_{gs} produced by the charge stored on a capacitor. The charge has to be periodically refreshed. The influence of V_{gs} upon the resistance between the source and the drain of each transistor is illustrated in Fig. 3-14. Thus, the dependence between I_{ds} and V_{ds} is not linear but it can be used as an acceptable approximation of a linear function within certain ranges of currents and voltages. More sophisticated multiplication mechanisms (such as Gilbert multipliers) need to be used if very good linearity is required over a large range of voltages.

The number of operational amplifiers that can be integrated on a chip is limited. Therefore, the implementation methods that use operational amplifiers are applicable only to small-scale neural networks. To obtain high integration densities, the implementation of the activation function is performed with very simple circuits. A minimalist design style is adopted in the analogue approach described in [15]: each activation function is modelled by a circuit containing a single MOS transistor. The design methodology is based on current-mode subthreshold CMOS circuits, according

to which the signals of interest are represented as currents. The current mode approach offers signal processing at the highest bandwidth for a given power consumption. In [102] a different approach is described: the basic building block is a transconductance amplifier (Fig. 3-15). In its basic form, the amplifier contains three MOS transistors and transforms a differential input voltage $V_{in}=V_1-V_2$ into a differential output current $I_{out}=I_1-I_2$. The relation between input and output is non-linear and is a good approximation of a sigmoidal activation function.

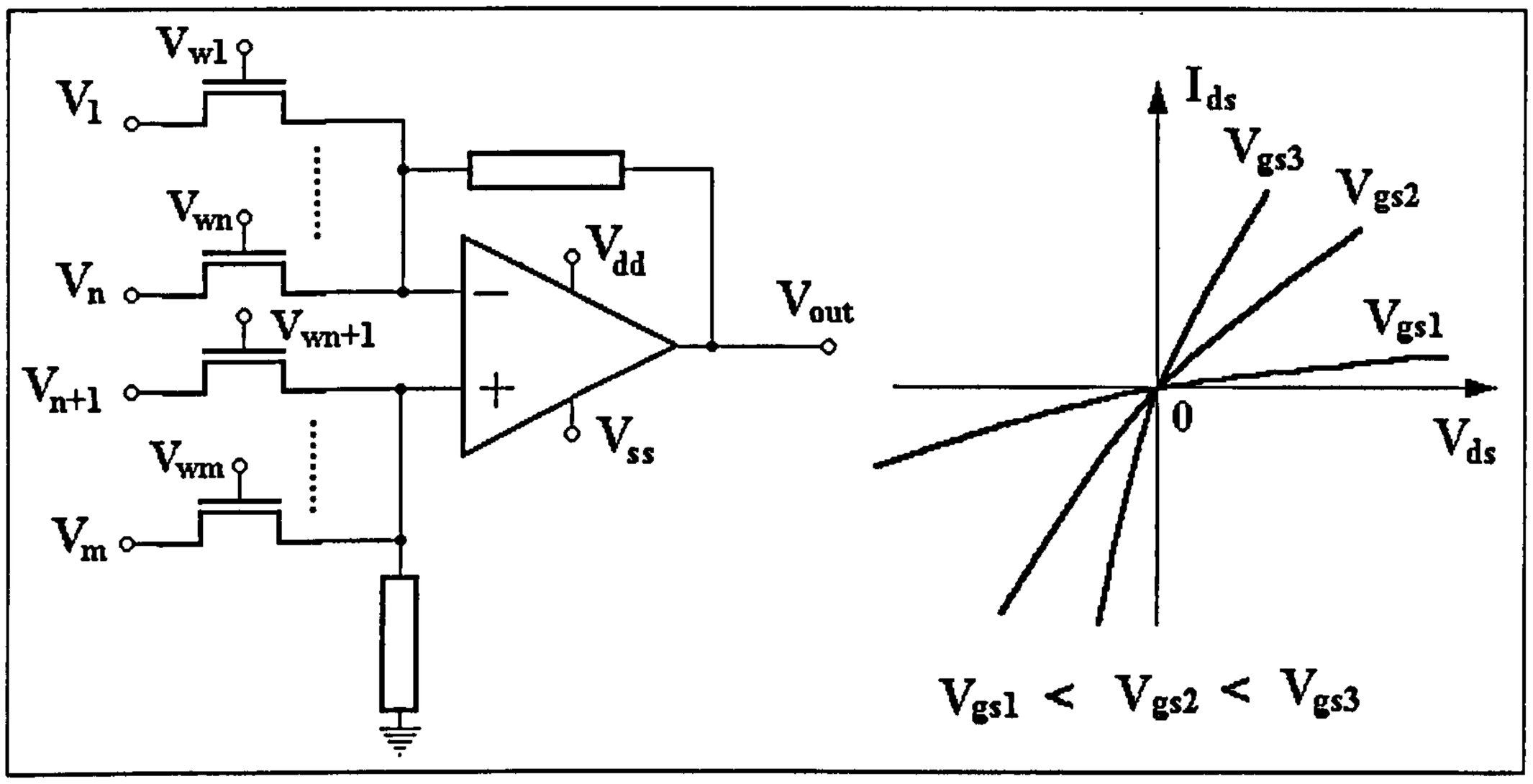

Fig. 3-14 – Neurone implementation with electrically tuneable weights

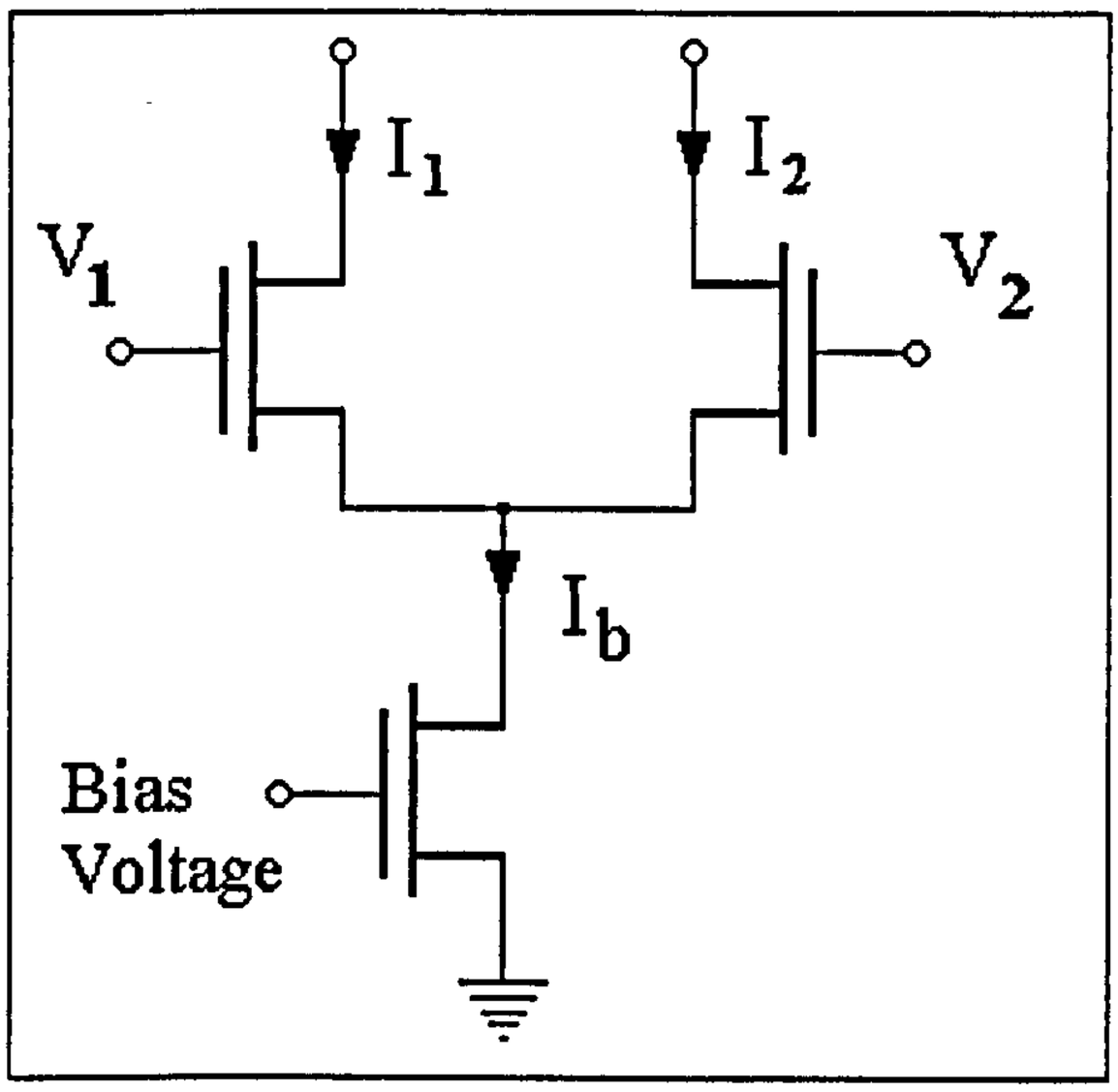

Fig. 3-15 – Circuit diagram of a differential transconductance amplifier

The first analogue commercial chip was the Intel 80170NW ETANN (Electrically Trainable Analogue Neural Network) [11]. It contains 64 neurones and 10280 weights. The non-volatile weights are stored as charge on floating transistor gates and a Gilbert multiplier provides 4-quadrant multiplication. A flexible design, including internal feedback and division of the weights into 64x80 banks, allows multiple configurations

including 3-layers of 64 neurones/layer, and 2-layers with 128 inputs and 64 neurones/layer. No on-chip training was provided, so the connection with a PC is necessary. The PC performs the training process and then transmits the resulting weight values to the neural chip.

New implementation technologies and possible applications of analogue neural chips continue to be investigated and several successes have been reported in literature [104], [91], [97], [61], [100].

3.5.2 Digital Hardware Implementation

The digital neural network category encompasses many sub-categories including slice architectures, single instruction multiple data approach (SIMD), systolic array devices, radial basis function architectures (RBF), ASIC and FPGA implementations. For designers, digital technology has the advantage of mature fabrication techniques and digital chips are easily embedded into most applications. However, digital calculations are usually slower than in analogue systems, especially when performing the multiplications between weights and input signals. Moreover, analogue inputs must first be converted into digital format. The most common performance rating used to compare digital neural implementations is the Connection-Per-Seconds (CPS), which is defined as the rate of multiplication and accumulate operations during normal operation.

Slice architectures for neural networks provide basic building blocks to construct networks of arbitrary size and precision. For example, the NeuroLogix NLX-420 Neural Processor Slice has 16 processing elements and a speed of 300MCPS. A common 16-bit input bus is multiplied by different weights in each parallel processing element. The weights are initially read from outside the chip. The 16-bit weights and inputs can be selected by the user as sixteen 1-bit values, four 4-bit values, two 8-bit values or one 16-bit value. The 16 neuronal inputs are processed by a user-defined piecewise continuous activation function to produce a 16-bit output. Internal feedback allows the implementation of multi-layer networks. Multiple chips can be interconnected to build large networks.

A far more elaborate approach is to place many small processors on a chip. Two architectures dominate such designs: single instruction with multiple data (SIMD) and systolic arrays. For SIMD design, each processor executes the same instruction in parallel, but on different data. In systolic arrays, the basic processors are connected in a matrix architecture. Each processor does one calculation step before passing its result on to the next processor in a pipelined manner. A systolic array system can be built with

Siemens MA-16. The MA-16 provides fast matrix operations using 4x4 processor matrices with a 16-bit interconnecting buss. The overall performance is 400MCPS. The multiplier and accumulator outputs have 48-bit precision. Weights are stored on-chip and neurone activation functions are generated off-chip via lookup tables. Multiple chips can be cascaded.

The networks with RBF neurones provide fast learning and straightforward implementation. The comparison of input vectors to stored training vectors can be done quickly if non-Euclidean distances (such as the Manhattan norm shown in (3-25)) are calculated with no multiplication. One of the commercial available products is Nestor NI1000 chip. The Nestor NI1000, developed jointly by Intel and Nestor, contains 1024 stored vectors of 256 5bit elements. The chip has two on-chip learning algorithms, but it is relatively slow: 40kCPS.

$$\|X-Y\|_{Manhat\,tan} = \sum_i |x_i - y_i| \qquad (3\text{-}25)$$

Digital ASIC and FPGA solutions require that the ANN is fully designed and trained for a particular application before its actual hardware implementation. The operation of the ANN is usually described in terms of Boolean functions or in terms of logic operations and threshold gates (TG). The threshold gate is a more general concept than a logic gate. Any logic gate can be considered a particular case of a TG but TGs can perform more complex information processing tasks than logic gates. They have inputs with different integer weights that makes them very suitable for neurone hardware implementation. Unfortunately, the technology limits the weights to small integer values: 0, ±1, ±2, ±3. The direct use of TGs to implement neurones generates compact hardware structures, but this approach can only be used for a limited number of ASIC technologies. It cannot be used for FPGA implementation because they are not available inside the Complex Logic Blocks (CLBs) of FPGA chips. However, the indirect use of TGs is possible because a TG can be emulated by a digital structure composed of no more than a few AND, OR and NOT interconnected logic gates.

Designing an ANN for a specific application involves the use of either training algorithms or constructive algorithms. Constructive algorithms are the preferable approach in many situations because they are able to determine both the network architecture and the neurone weights and are guaranteed to converge in finite time. A large number of constructive algorithms, reviewed in [29] have been developed in the last decade. They are divided into three categories: geometric ([34], [113]), network-

based [118] and algebraic [67]. Several VLSI friendly algorithms have been created in order to bring closer the design stage and the implementation stage. These algorithms consider some specific aspects of VLSI implementation technology: the precision of the input weights and the neurone fan-in. These factors lead to important limitations that need to be taken into account when designing a neural network. One of the first VLSI friendly algorithms uses the concept of "adaptive tree network" [16]. The research in this direction has been extended by using a combination of AND gates and OR gates, alongside with Threshold Gates (TGs) [19].

3.5.3 Hybrid Implementation Techniques

Hybrid design attempts to combine the advantages of analogue and digital techniques. The use of analogue implementation is attractive for reasons of compactness, speed and the absence of quantization effects. The advantage of digital signals is their robustness. These signals are not affected by disturbances and the calculations performed in digital format always yield precise results.

The pulse modulation technique is one of the most promising principles that can be used to develop efficient hybrid architectures. Using pulse modulation, the internal signals of the neural network are modelled as pulse streams whose parameters are varied in accordance with the neurone states. Depending on the parameter that is varied, there are three theoretical alternatives: the pulse-amplitude modulation, the pulse-width modulation and the pulse-frequency modulation [62].

1. In case of pulse-amplitude modulation, the amplitude of the pulses is modulated in time in a manner that reflects the variation of the corresponding neurone signal. This technique is not satisfactory in neural networks because the information is transmitted as analogue voltage levels, which makes it susceptible to processing errors due to circuit parameter variations.
2. The pulse-width modulation method alters the pulses duration according to the amplitude of the neural signal. The pulse-width modulated signal is robust since the information is coded as a set of time intervals and no analogue voltage is used. However, if several signals in the neural network have almost similar values, then a large number of pulse edges occur almost simultaneously. The existence of this synchronism represents a drawback in VLSI networks since many synapses tend to draw current from the internal supply lines simultaneously. It follows that the internal supply lines have to be oversized to accommodate the high instantaneous currents that may be produced by the use of pulse-width modulation.

3. Pulse-frequency modulation maintains both the amplitude and the width of the pulses constant but modifies the frequency of the pulses. This modulation scheme generates robust signals as well. Moreover, different signals modelling the equal analogue quantities are usually phase-shifted, which leads to avoiding the synchronism of the pulse edges. Thus, the power requirement is averaged in time as a result of using pulse-frequency modulation. Hybrid neural networks combining pulse-frequency modulation and neurones implemented in analogue technology have been successfully designed and implemented [105], [41].

Another reason for producing hybrid neural network implementations is the need to interface the neural architectures with the existing digital equipment. In such a situation, the external inputs and outputs are digital, to facilitate the integration into the digital systems, while internally some or all of the processing is done in analogue technology. The AT&T ANNA chip, for example, is externally digital and all the internal signals are in digital format, but it uses capacitor charge to store the neurone weights [114]. The charge is periodically refreshed by a specialised internal mechanism. The chip structure includes Multiplying Digital-to-Analogue Converters (MDAC). There are electronic devices capable to multiply a digital value with an analogue signal. The MDACs are used to perform the multiplications between the weights and the input signals of each neurone. Conversely, the Bellcore CLNN-32 chip uses digital 5-bit weights, but the neurone inputs and outputs are analogue signals [13]. As in the case of the ANNA chip, the multiplications between weights and signals involve the use of MDACs. The overall performance of the Bellcore chip is 100MCPS. Thus, the MDACs allow the neural network designer to freely combine analogue and digital technologies in an optimal fashion for a given application problem.

3.5.4 Software Versus Hardware Implementations

The software implementation uses a classical von Neuman machine (a general-purpose microprocessor or a DSP). This approach can be used to implement any kind of neural network structures and any training algorithm. However, neural networks simulated on Von Neuman machines run in a series fashion, which does not allow them to be used in real time applications. The operation speed of the neural network is inverse proportional to the number of neurones. Consequently, very large neural networks can only be efficiently software implemented if special hardware resources are also available: either large general-purpose parallel machines or cheaper alternatives such as specialised co-processors, or accelerator cards for personal computers.

The hardware approach overcomes the speed limitations of the software implemented neural networks. True parallel operation mode is achieved in this case, making the calculation speed independent of the network complexity. The actual speed of hardware implementation solutions depends on the technology. The highest speed is achieved using optical implementations while the lowest speed is obtained with the electronic digital architectures. Several optical neural processors have been reported in literature [50], [47]. However, the optical technology has not attained its maturity yet. This approach is still too expensive, too imprecise and too rigid, so that electronic implementations are preferred in most cases.

The training process is faster in case of specialised chips as compared with software implementations, but only relatively simple training strategies are currently implemented into hardware. Thus, a limited number of training algorithms can be performed on-line. If the practical application does not require on-line training, the training process can be performed off-line in a software system, and then the resulting weights can be downloaded into the neural chip. Alternatively, the obtained weights can be used to produce an ASIC or FPGA implementation. FPGA implementations are preferable as they allow fast prototyping. Furthermore, some FPGA chips are capable to change their structure on-line. This feature supports the design of a large range of new on-line training algorithms for digital implemented neural networks.

4. DEVELOPMENT OF A NOVEL INDUCTION MOTOR SENSORLESS CONTROL STRATEGY

This chapter presents the mathematical principles and algorithms underlying the adopted sensorless control strategy for three-phase cage induction motors. The control method comprises two elements: the stator current control strategy and the sensorless speed control strategy. Both of them are based on an equivalent three-phase R-L-e circuit whose parameters are derived from the space vector model of the induction motor. Induction motor speed estimation strategies are investigated and compared in terms of accuracy and hardware implementation complexity. Several new sensorless speed control methods are formulated and tested by computer simulations.

4.1 THE INDUCTION MOTOR EQUIVALENT CIRCUIT

The proposed motor control strategy uses the classical sensorless drive system structure with the motor supplied by a VSI-PWM inverter, which is controlled by a digital circuit based only on the stator current feed-back information. As mentioned in chapter 2, the predictive current control method uses an equivalent R-L-e circuit of the load modelled by the equation

$$\underline{u}(t) = R\underline{i}(t) + L\frac{d\underline{i}(t)}{dt} + \underline{e}(t) \qquad (4\text{-}1)$$

The R-L-e equivalent circuit parameters for an induction motor can be derived from its general space vector model

$$\begin{cases} \underline{u}_s^\theta = R_s \underline{i}_s^\theta + \dfrac{d\underline{\Psi}_s^\theta}{dt} \\ \underline{u}_r^\theta = R_r \underline{i}_r^\theta + \dfrac{d\underline{\Psi}_r^\theta}{dt} - j(\omega_e - \omega_{er})\underline{\Psi}_r^\theta = 0 \\ \underline{\Psi}_s^\theta = L_s \underline{i}_s^\theta + L_m \underline{i}_r^\theta \\ \underline{\Psi}_r^\theta = L_r \underline{i}_r^\theta + L_m \underline{i}_s^\theta \end{cases} \quad (4\text{-}2)$$

particularised for the stator reference frame. Thus, the parameters defining the reference frame are $\theta=0$ and $\omega=\omega_{es}=0$, which yields the equation system

$$\begin{cases} \underline{u}_s^s = R_s \underline{i}_s^s + \dfrac{d\underline{\Psi}_s^s}{dt} \\ \underline{u}_r^s = R_r \underline{i}_r^s + \dfrac{d\underline{\Psi}_r^s}{dt} - j\omega_{er}\underline{\Psi}_r^s = 0 \\ \underline{\Psi}_s^s = L_s \underline{i}_s^s + L_m \underline{i}_r^s \\ \underline{\Psi}_r^s = L_r \underline{i}_r^s + L_m \underline{i}_s^s \end{cases} \quad (4\text{-}3)$$

The two fluxes can be eliminated from the equations giving:

$$\begin{cases} \underline{u}_s^s = R_s \underline{i}_s^s + L_s \dfrac{d\underline{i}_s^s}{dt} + L_m \dfrac{d\underline{i}_r^s}{dt} \\ 0 = R_r \underline{i}_r^s + L_m \dfrac{d\underline{i}_s^s}{dt} + L_r \dfrac{d\underline{i}_r^s}{dt} - j\omega_{er}\left(L_m \underline{i}_s^s + L_r \underline{i}_r^s\right) \end{cases} \quad (4\text{-}4)$$

Therefore the derivative of the rotor current vector is

$$\frac{d\underline{i}_r^s}{dt} = \frac{1}{L_r}\left[-R_r \underline{i}_r^s - L_m \frac{d\underline{i}_s^s}{dt} + j\omega_{er}\left(L_r \underline{i}_r^s + L_m \underline{i}_s^s\right)\right] \quad (4\text{-}5)$$

Substituting this in (4-4) gives

$$\underline{u}_s^s = R_s \underline{i}_s^s + \left(L_s - L_m \frac{L_m}{L_r}\right)\frac{d\underline{i}_s^s}{dt} + \frac{L_m}{L_r}\left[-R_r \underline{i}_r^s + j\omega_{er}\left(L_r \underline{i}_r^s + L_m \underline{i}_s^s\right)\right] \quad (4\text{-}6)$$

Identifying this result with the fundamental equation (4-1), the parameters of the equivalent circuit are determined as follows:

$$\begin{cases} L = \dfrac{L_s L_r - L_m^2}{L_r} \\ R = R_s \\ \underline{e} = \dfrac{L_m}{L_r}\left[-R_r \underline{i}_r^s + j\omega_{er}\left(L_r \underline{i}_r^s + L_m \underline{i}_s^s\right)\right] = \dfrac{L_m}{L_r}\left(-R_r \underline{i}_r^s + j\omega_{er}\underline{\Psi}_r^s\right) \\ \underline{u} = \underline{u}_s \\ \underline{i} = \underline{i}_s \end{cases} \quad (4\text{-}7)$$

Consequently, the voltage space vector $\underline{u}$ is the voltage supplying the motor, the current $\underline{i}$ is the stator current, while the internal voltage $\underline{e}$ is a quantity bearing information on the motor operation parameters (speed and rotor current).

Table 4.1 presents the electrical parameters of five different three-phase cage induction motors. It reflects the influence of the motor power on other parameter values. Thus, the stator and the rotor resistances are larger at lower powers and smaller at higher powers. On the other hand, the leakage inductances are always small compared to the mutual inductance. In a well-designed motor, the total leakage inductance $L_{\sigma s}+L_{\sigma r}$ does not surpasses 10% of the mutual inductance L_m.

Table 4.1- Electrical parameters of three-phase induction motors

Quantity	(a)	(b)	(c)	(d)	(e)
R_s	0.0114Ω	0.371Ω	0.79Ω	2.89Ω	5.9Ω
R_r	0.011Ω	0.415Ω	0.76Ω	2.39Ω	4.62Ω
$L_{s\sigma}$	0.32mH	2.72mH	1.57mH	11mH	22mH
$L_{r\sigma}$	0.36mH	3.3mH	1.59mH	6mH	24mH
L_m	11.68mH	84.33mH	65mH	214mH	809mH
p (pole pairs)	2	1	2	1	1
P	110kW	11.1kW	5kW	3kW	2kW

Under these conditions, the expression of the equivalent inductance L can be transformed as shown in (4-8) and it can be approximated by the sum of the two leakage inductances. The result is the approximate equivalent circuit illustrated in Fig. 4-1.

$$L = \frac{(L_{\sigma s}+L_m)\cdot(L_{\sigma r}+L_m)-L_m^2}{L_m+L_{\sigma r}} = \frac{L_{\sigma s}L_{\sigma r}+L_m(L_{\sigma s}+L_{\sigma r})}{L_m+L_{\sigma r}} \approx L_{\sigma s}+L_{\sigma r} \qquad (4\text{-}8)$$

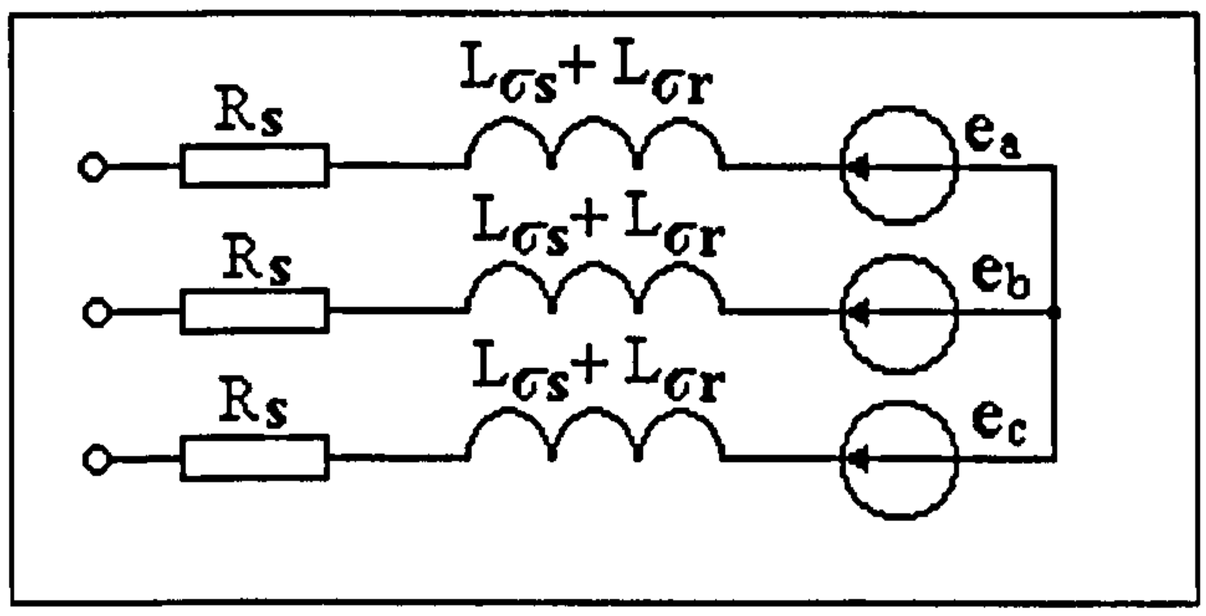

Fig. 4-1 – The approximate R-L-e equivalent circuit of a three-phase induction motor

Thus, despite the large number of turns in the motor windings, the equivalent inductance L is relatively small due to the tight magnetic coupling between stator and rotor. However, the precise circuit contains a slightly larger equivalent inductance that can be calculated according to (4-7).

The parameters of the 11.1 kW motor presented on column (b) in Table 4.1 are used to illustrate all the control principles formulated in this chapter. Using the same parameters for all simulations and calculations facilitates meaningful comparisons between alternative control strategies.

4.2 THE CURRENT CONTROL ALGORITHM

4.2.1 The Switching Strategy

The predictive current control strategy proposed in this thesis involves the concept of non-inductive voltage, which is defined as the sum of the resistive voltage component $R\underline{i}$ and the internal voltage component $\underline{e}$. This quantity, denoted by $\underline{V}_{ni}$, excludes the inductive voltage component $Ld\underline{i}/dt$ from the total voltage, hence the name of **non-inductive voltage.** This can be calculated using one of the two expressions:

$$\underline{V}_{ni} = R\underline{i} + \underline{e} = \underline{u} - L\frac{d\underline{i}}{dt} \tag{4-9}$$

The second formulation is more profitable as it does not use the value of the internal voltage $\underline{e}$, which is difficult to calculate.

The digital implementation of the control algorithm requires that all the quantities are sampled at equal time intervals. If the sampling process is taken into account then equation (4-1) becomes (4-10), where T_s is the sampling period and k is the index of the samples:

$$\underline{u}(kT_s) = R\underline{i}(kT_s) + \frac{L}{T_s}\left[\underline{i}(kT_s) - \underline{i}((k-1)T_s)\right] + \underline{e}(kT_s) + Err(k, T_s) \tag{4-10}$$

The function $Err(k,T_s)$ represents the calculation error generated by replacing the derivative in (4-1) with the approximation calculated based on the difference between two consecutive current values. The calculation error decreases with increasing frequency of the PWM and is negligible at the frequencies commonly used in induction motor drive systems (2 kHz to 20 kHz). Under these conditions, equation (4-10) can be written as

$$\underline{u}(kT_s) \cong R\underline{i}(kT_s) + \frac{L}{T_s}\left[\underline{i}(kT_s) - \underline{i}((k-1)T_s)\right] + \underline{e}(kT_s) \tag{4-11}$$

The notation can be simplified by replacing the time argument 'kT_s' with the sample index k, thereby transforming equation (4-11) into the equivalent form

$$\underline{u}(k) \cong R\underline{i}(k) + \frac{L}{T_s}\left[\underline{i}(k) - \underline{i}(k-1)\right] + \underline{e}(k) \tag{4-12}$$

Based on (4-9) and (4-12), the non-inductive voltage can be approximated as

$$\underline{V}_{ni}(k) \cong \underline{u}(k) - \frac{L}{T_s}\left[\underline{i}(k) - \underline{i}(k-1)\right] \tag{4-13}$$

The inverter output voltage is constant between two consecutive switching transients and if the PWM period is sufficiently short, the internal voltage $\underline{e}$ can be considered constant as well ($\underline{u}(t)=\underline{U}$), $\underline{e}(t)=\underline{E}$). This assumption substantially simplifies the current calculations. Thus, the relation (4-1) becomes

$$\underline{U} = R\underline{i}(t) + L\frac{d\underline{i}(t)}{dt} + \underline{E} \tag{4-14}$$

and has the solution

$$\underline{i}(t) = \frac{\underline{U}-\underline{E}}{R} + \left(\underline{i}(0) - \frac{\underline{U}-\underline{E}}{R}\right)\cdot e^{-\frac{Rt}{L}} \tag{4-15}$$

The real and the imaginary part of the current space vector are

$$\begin{cases} x(t) = \mathrm{Re}\left\{\underline{i}(t)\right\} = \mathrm{Re}\left\{\dfrac{\underline{U}-\underline{E}}{R}\right\} + \mathrm{Re}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}\cdot e^{-\frac{Rt}{L}} \\ y(t) = \mathrm{Im}\left\{\underline{i}(t)\right\} = \mathrm{Im}\left\{\dfrac{\underline{U}-\underline{E}}{R}\right\} + \mathrm{Im}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}\cdot e^{-\frac{Rt}{L}} \end{cases} \tag{4-16}$$

Combining the two equations (4-16), the result is

$$y(t) = \mathrm{Im}\left\{\frac{\underline{U}-\underline{E}}{R}\right\} + \frac{\mathrm{Im}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}}{\mathrm{Re}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}}\cdot\left(x(t) - \mathrm{Re}\left\{\underline{i}(0) - \frac{\underline{U}-\underline{E}}{R}\right\}\right) \tag{4-17}$$

which can be reduced to

$$y(t) = \frac{\mathrm{Im}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}}{\mathrm{Re}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}}\cdot x(t) + \mathrm{Im}\left\{\underline{i}(0)\right\} = a\cdot x(t) + b \tag{4-18}$$

The constants 'a' and 'b' in (4-18) are defined by

$$\begin{cases} a = \dfrac{\mathrm{Im}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}}{\mathrm{Re}\left\{\underline{i}(0) - \dfrac{\underline{U}-\underline{E}}{R}\right\}} \\ b = \mathrm{Im}\{\underline{i}(0)\} \end{cases} \tag{4-19}$$

Equation (4-18) is that of a straight line. This entails that the vertex of the current space vector $\underline{i}$ shifts with a variable speed along a straight trajectory. Due to the linear relationship between vectors $\underline{V}_{ni}$ and $\underline{i}$, the trajectory of $\underline{V}_{ni}$ is straight as well. Furthermore, according to (4-20), the vertices of the two space vectors shift along parallel trajectories whose direction is indicated by the argument ε calculated according to (4-21).

$$\underline{V}_{ni}(t) = R \cdot \underline{i}(t) + \underline{E} \Rightarrow d\underline{V}_{ni} = R \cdot d\underline{i} \tag{4-20}$$

$$\varepsilon = \arg(d\underline{V}_{ni}) = \arg(d\underline{i}) = \arg(\underline{i}(+\infty) - \underline{i}(0)) = \arg(\underline{V}_{ni}(+\infty) - \underline{V}_{ni}(0)) \tag{4-21}$$

The value $\underline{V}_{ni}(+\infty)$ is the non-inductive voltage after an infinitely long period of time and can be calculated as:

$$\underline{V}_{ni}(+\infty) = \lim_{t\to\infty}(R\underline{i}(t) + \underline{E}) = \lim_{t\to\infty}\left[R \cdot \frac{\underline{U}-\underline{E}}{R} + \left(\underline{i}(0) - \frac{\underline{U}-\underline{E}}{R}\right) \cdot e^{-\frac{Rt}{L}} + \underline{E}\right] = \underline{U} \tag{4-22}$$

As a result, the angle ε is

$$\varepsilon = \arg(\underline{U} - \underline{V}_{ni}(0)) \tag{4-23}$$

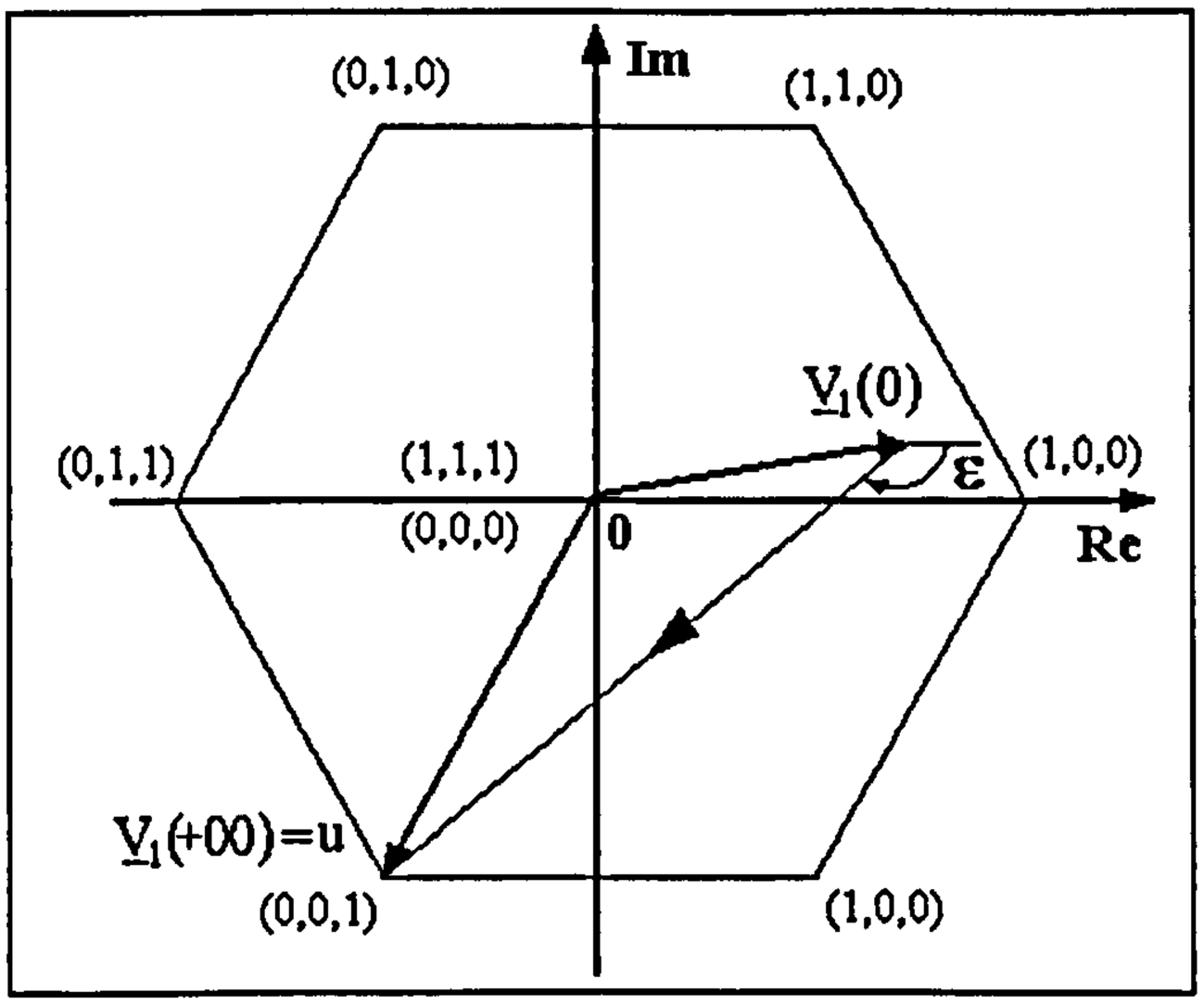

Fig. 4-2 - The trajectory of the vertex of $\underline{V}_{ni}$ space vector

Thus, as illustrated in Fig. 4-2, the trajectory of vector $\underline{V}_{ni}$ is a straight line oriented on the direction that links the point corresponding to the inverter output voltage with the initial value of the non-inductive voltage $\underline{V}_{ni}(0)$. These considerations can be used to determine the direction of the current trajectory in the complex plane without using the value of the internal voltage $\underline{e}$.

The control voltages to the transistors in the PWM inverter need to be produced in such a way that the inverter output voltage maintains the required currents across the load. The required current modification during one sampling period is a complex quantity defined by the argument "$\arg\{\underline{i}_{ref}(k+1)-\underline{i}(k)\}$" and the module $|\underline{i}_{ref}(k+1)-\underline{i}(k)|$. These two parameters are often impossible to achieve simultaneously because only 7 inverter output voltages are available, which means that only 7 different current shifts can be performed at a given moment. Therefore, there are two alternative switching strategies:

- Minimising the module of the current error $|\underline{i}_{ref}(k+1)-\underline{i}(k+1)|$.
- Minimising the angle between the direction of the required current trajectory and the direction of the actual current trajectory $|\arg\{\underline{i}_{ref}(k+1)-\underline{i}(k)\}-\arg\{\underline{i}(k+1)-\underline{i}(k)\}|$.

The first alternative generates optimal control results but requires that equation (4-14) is solved for all seven possible output voltages and the results are compared. The most important disadvantage of this method is that the internal voltage $\underline{E}$ needs to be determined first. The calculation can be performed according to the equation

$$\underline{E}(k)=\underline{V}_{ni}(k)-R\underline{i}(k)\cong\underline{u}(k)-\frac{L}{T_s}[\underline{i}(k)-\underline{i}(k-1)]-R\underline{i}(k) \qquad (4\text{-}24)$$

but this involves the value of the stator resistance R that needs to be determined on-line because it changes during the motor operation. Therefore, this method necessitates complicated calculations that make it unpractical.

The inverter switching strategy adopted in this thesis uses the second alternative. This approach yields good control results without requiring the value of the stator resistance, because it involves $\underline{V}_{ni}$ instead of $\underline{E}$ in the calculations. Thus, the directions ε_j (j=1,2,3, ...7) of the possible current trajectories are first determined according to equation (4-23). Then the output voltage $\underline{u}_j$ that minimises the expression $\left|\varepsilon_j-\arg\{\underline{i}_{ref}(k+1)-\underline{i}(k)\}\right|$ is generated during the next sampling period.

This current control strategy is illustrated by the example in Fig. 4-3. The current error vector $\Delta\underline{i}_{ref}=\underline{i}_{ref}(k+1)-\underline{i}(k)$ indicates a direction in the complex plane which is not

identical to any of the directions that can be achieved using the available voltages. However, the voltage coded as (0,1,0) is capable of producing a current change in a direction that is much closer to the reference one than the other six possibilities (including the zero output voltage). As a result, this voltage is generated during the next sampling period T_s. A consequence of the fast voltage switching is that the non-inductive voltage vector $\underline{V}_{ni}$ never reaches the final value $\underline{V}_{ni}(+\infty) = \underline{u}$ during any of the sampling periods and therefore $\underline{V}_{ni}$ is always inside the hexagon in Fig. 4-3.

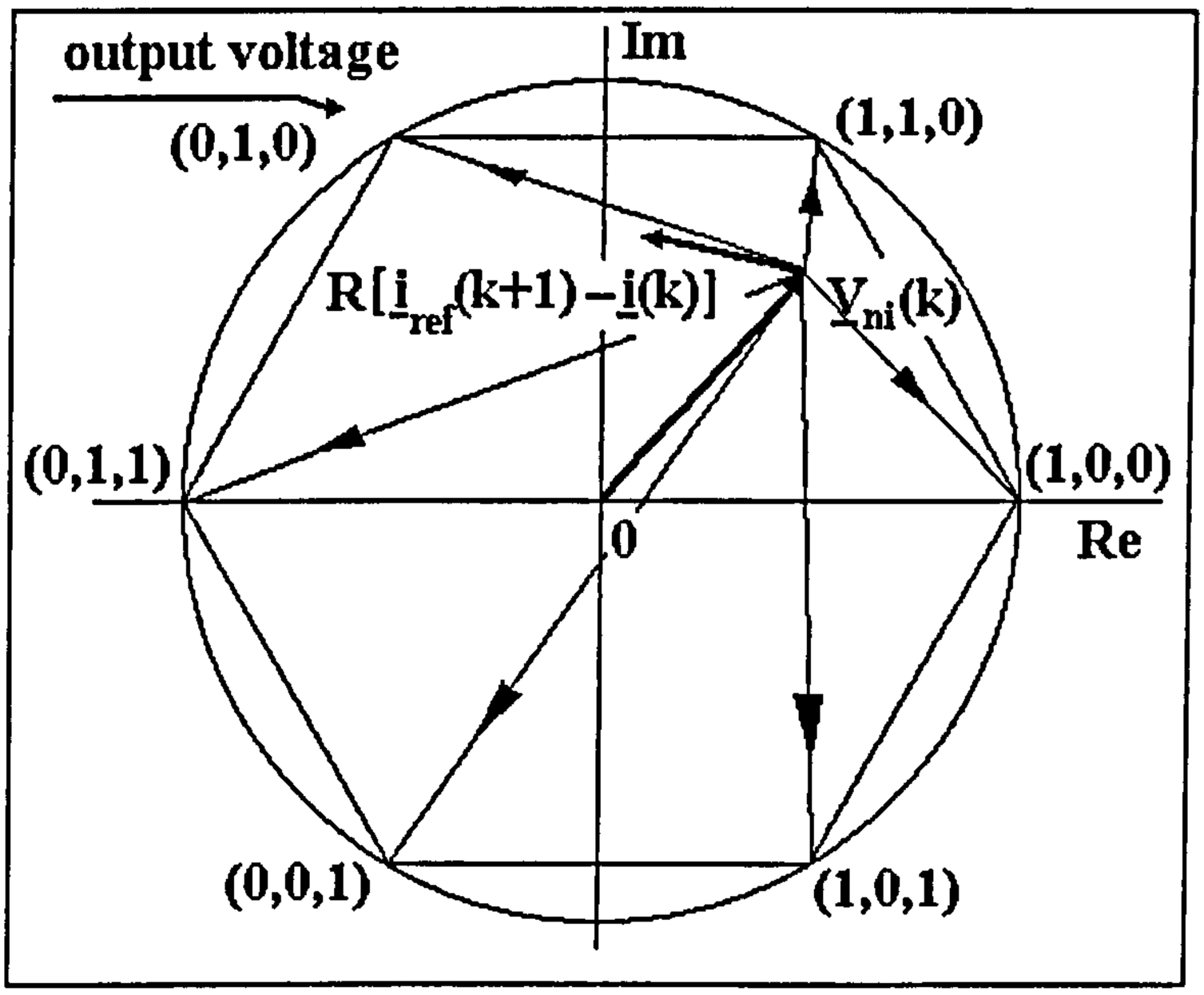

Fig. 4-3- The graphic representation of the PWM current control principle

The switching strategy can include the null voltage generated by the inverter, thereby increasing the control flexibility, or it can exclude it improving the current response speed. Including the null output voltage in the switching strategy improves the current harmonic content [106], but presents the disadvantage that the current response is slow if the motor current is small. The current change rate $|d\underline{i}/dt|$ when $\underline{u}=0$ is governed by equation

$$\left|\frac{d\underline{i}}{dt}\right| = -\frac{1}{L}\left|R\underline{i} + \underline{e}\right| \tag{4-25}$$

that is derived from (4-1). As demonstrated by (4-7), the internal voltage $\underline{e}$ is proportional to the motor currents, and therefore the module of vector $\underline{e}$ is approximately proportional with the module of the equivalent current vector (the stator current). In this case, the current change rate can be considered proportional to the module of the current vector, which means that the system response is infinitely slow when the motor currents tend to zero.

$$\left|\frac{d\underline{i}}{dt}\right| \cong -K|\underline{i}| \Rightarrow |\underline{i}| \cong i(0)\cdot e^{-Kt} \tag{4-26}$$

If the inverter generated voltage is not zero then the current change rate is given by

$$\left|\frac{d\underline{i}}{dt}\right| = -\frac{1}{L}|R\underline{i} + \underline{e} - \underline{u}| = -\frac{1}{L}|\underline{V}_{ni} - \underline{u}| \tag{4-27}$$

This ensures high response speed because the vector $\underline{V}_{ni}$ lies always inside the voltage hexagon and therefore $|\underline{V}_{ni}-\underline{u}|$ is always much larger than zero. As a result, the control method excluding the zero output voltage has to be adopted when the value of $|\underline{V}_{ni}|$ is below a critical limit $|\underline{V}_{ni}|_{crt}$. The zero voltage can be involved in the switching strategy when $|\underline{V}_{ni}|$ is above this limit. The value of the critical limit is chosen based on the required current response and the parameters in the equivalent circuit. Consequently, the control method that always excludes the zero voltage can be considered a particular case of the general control strategy. This particular case is defined by $|\underline{V}_{ni}|_{crt}=+\infty$ so that $|\underline{V}_{ni}|<|\underline{V}_{ni}|_{crt}$ in all situations.

Therefore, the adopted current control strategy in this basic form is flexible as it allows adjusting the relationship between the response speed and the harmonic content. A small $|\underline{V}_{ni}|_{crt}$ ensures a better harmonic content while a large $|\underline{V}_{ni}|_{crt}$ generates faster transient response. The algorithm presented in [106] offers a similar type of flexibility but it uses the magnitude of the current error as the criterion for using or rejecting the zero voltage. The new current control strategy uses the value of $|\underline{V}_{ni}|$ instead.

The new current control strategy generates voltage pulses defined by widths that are integer multiples of the sampling period T_s. The motor currents are sampled before each switching process. Every time, the last two sets of current samples are used in the calculations for the next voltage. Therefore, the output voltage can only change at definite moments in time given by

$$t_k = kT_s + \delta t \qquad k = 0,1,2,3,.... \tag{4-28}$$

where δt is the time required for the calculation process to be fulfilled (Fig. 4-4).

The classical PWM signals are generated by comparing a sinewave (the modulator) with a triangular wave (the carrier). The widths of the generated voltage pulses varies continuously between zero and the period of the triangular carrier T_c. The frequency of the voltage pulses f_{PWM} equals the carrier frequency f_c. An alternative method is the space vector PWM. It changes the inverter voltage between the seven possible values in such a manner that the average voltage over several switching periods

equals the reference voltage space vector. In both cases, the PWM frequency has to be rigorously controlled because it influences the quality of the generated voltage signal, but it is also approximately proportional to the losses in the inverter. Thus, it has to be limited to an acceptable value by adopting an appropriate carrier frequency. The PWM frequency used in common drive systems varies between 2 kHz and 20 kHz.

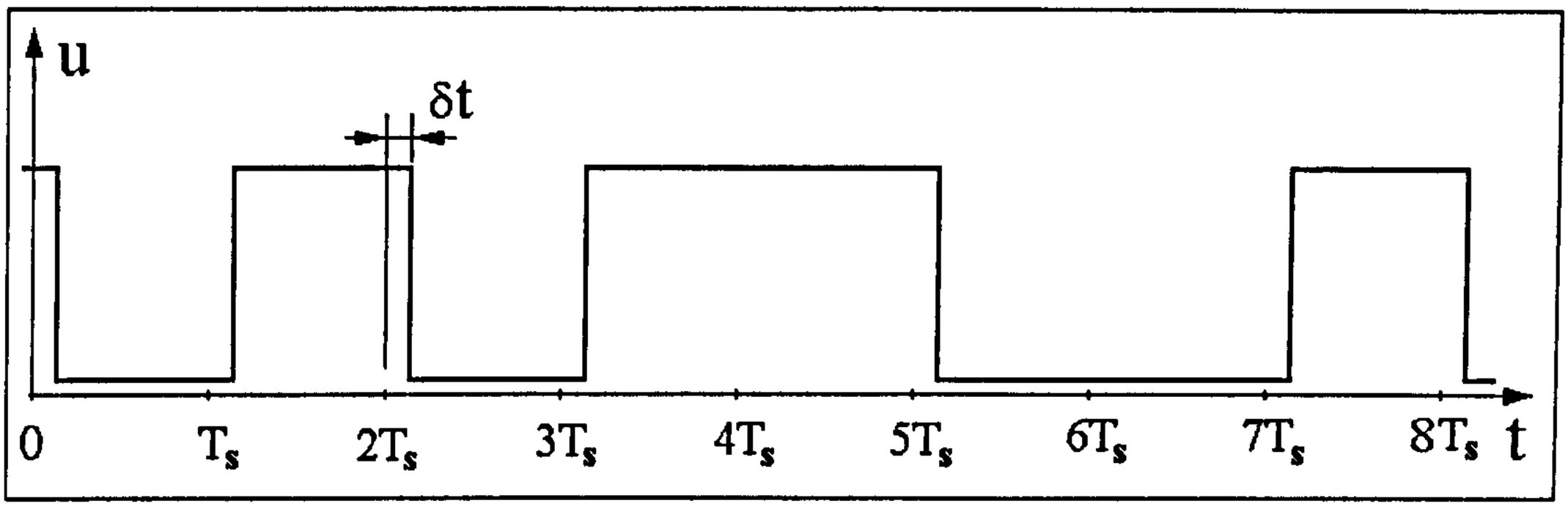

Fig. 4-4 – The PWM voltage signal generated by the basic version of the new control algorithm

The PWM frequency generated by the new control method is not constant, but is influenced by the motor operation conditions and it varies inside the interval $[0; 1/(2T_s)]$. The maximal number of switching processes per second is restricted by equation (4-28). This number needs to be large so that a voltage change can be generated at approximately the moment it is required. This consideration leads to the necessity of a short sampling period. Nonetheless, a short T_s is equivalent to an increased upper limit of the PWM frequency. Therefore, an additional restriction needs to be imposed on the current control algorithm in order to limit the frequency of the PWM voltage while maintaining a short sampling period. A given PWM frequency f_{PWM} can be imposed if only two switching processes are allowed during a time interval of $T_{PWM}=1/f_{PWM}$. If the voltage has already been switched twice during a certain time period, then the switching process is inhibited until a new period begins. Thus, T_s and T_{PWM} are now independent quantities and T_s can be set at much smaller values than T_{PWM}. Typically, T_{PWM} is 10 to 40 times longer than T_s. As the PWM frequency can be as high as 20 kHz, it means that the sampling frequency can be up to 800 kHz, which requires fast A/D converters. This is the enhanced version of the current control algorithm. It is more flexible than the basic algorithm version because it allows a supplemental adjustment of the inverter losses beside the control over the current harmonics.

4.2.2 The On-Line Inductance Estimation

For a correct current control, the value of inductance L needs to be either measured or estimated. An original on-line estimation method has been integrated into the switching algorithm to transform it into a universal control strategy, which does not require any previous information about the motor parameters.

The on-line estimation starts with an initial inductance $\hat{L}(0)$. The inductance estimation $\hat{L}(0)$ is then incrementally updated and progressively more accurate estimations $\hat{L}(1)$, $\hat{L}(2)$, $\hat{L}(3)$ are calculated until the correct value is found. The algorithm convergence is guaranteed for any initial inductance value, but for reasons of simplicity, it is considered that $\hat{L}(0)=0$. Each incremental correction is performed in parallel with one current control step (one output voltage being determined). The effect of using an estimated inductance instead of the exact value is that the non-inductive voltage $\underline{V}_{ni}$ cannot be calculated exactly, but an estimated value $\hat{\underline{V}}_{ni}$ is determined instead. Equation (4-13) can be therefore rewritten as

$$\hat{\underline{V}}_{ni}(k)=\underline{u}(k)-\frac{\hat{L}}{T_s}[\underline{i}(k)-\underline{i}(k-1)]=\underline{u}(k)-\frac{\hat{L}}{T_s}\cdot\Delta\underline{i}(k) \qquad (4\text{-}29)$$

The estimated inductance is given by the relation $\hat{L}=L+\Delta L$ where L is the real inductance and ΔL is the estimation error. As a result, equation (4-29) becomes

$$\hat{\underline{V}}_{ni}(k)=\underline{u}(k)-\frac{L+\Delta L}{T_s}\cdot\Delta\underline{i}(k)=\underline{V}_{ni}(k)-\frac{\Delta L}{T_s}\cdot\Delta\underline{i}(k) \qquad (4\text{-}30)$$

During the next sampling period the current varies according to equation

$$\underline{u}(k+1)\cong R\underline{i}(k+1)+\underline{e}(k+1)+\frac{L}{T_s}\Delta\underline{i}(k+1) \qquad (4\text{-}31)$$

Adding and subtracting $\underline{V}_{ni}(k)=R\underline{i}(k)+\underline{e}(k)$ gives

$$\underline{u}(k+1)\cong\underline{V}_{ni}(k)-R\underline{i}(k)-\underline{e}(k)+R\underline{i}(k+1)+\underline{e}(k+1)+\frac{L}{T_s}\Delta\underline{i}(k+1) \qquad (4\text{-}32)$$

With the notation $\Delta\underline{e}(k+1)=\underline{e}(k+1)-\underline{e}(k)$, (4-32) can be written as

$$\underline{u}(k+1)\cong\underline{V}_{ni}(k)+R\Delta\underline{i}(k+1)+\Delta\underline{e}(k+1)+\frac{L}{T_s}\Delta\underline{i}(k+1) \qquad (4\text{-}33)$$

and as

$$\underline{u}(k+1)-\underline{V}_{ni}(k)\cong\left(R+\frac{L}{T_s}\right)\cdot\Delta\underline{i}(k+1)+\Delta\underline{e}(k+1) \tag{4-34}$$

Substituting (4-30) in (4-34) gives:

$$\underline{u}(k+1)-\hat{\underline{V}}_{ni}(k)-\frac{\Delta L}{T_s}\cdot\Delta i(k)\cong\left(R+\frac{L}{T_s}\right)\cdot\Delta\underline{i}(k+1)+\Delta\underline{e}(k+1) \tag{4-35}$$

or

$$\hat{\underline{V}}_{\Delta}(k+1)\cong\left(R+\frac{L}{T_s}\right)\cdot\Delta\underline{i}(k+1)+\frac{\Delta L}{T_s}\cdot\Delta i(k)+\Delta\underline{e}(k+1) \tag{4-36}$$

where

$$\hat{\underline{V}}_{\Delta}(k+1)=\underline{u}(k+1)-\hat{\underline{V}}_{ni}(k) \tag{4-37}$$

The internal voltage $\underline{e}$ has been shown to be a function of the motor currents, which have a rate of change limited by the motor inductances. Therefore, the change of the internal voltage $\underline{e}$ is similarly limited and $|\Delta\underline{e}(k+1)|$ decreases with the increase of the sampling frequency $f_s=1/T_s$. As a result, in many practical applications $|\Delta\underline{e}(k+1)|$ is much smaller than the module of the other two terms in equation (4-36).

If conditions

$$\begin{cases}\left|\Delta\underline{e}(k+1)\right|<<\left(R+\dfrac{L}{T_s}\right)\cdot\left|\Delta\underline{i}(k+1)\right|\\ \left|\Delta\underline{e}(k+1)\right|<<\dfrac{\left|\Delta L\cdot\Delta\underline{i}(k)\right|}{T_s}\end{cases} \tag{4-38}$$

are fulfilled then equation (4-36) can be simplified as

$$\hat{\underline{V}}_{\Delta}(k+1)\cong\left(R+\frac{L}{T_s}\right)\cdot\Delta\underline{i}(k+1)+\frac{\Delta L}{T_s}\cdot\Delta\underline{i}(k) \tag{4-39}$$

The on-line induction estimation is based on the approximate equation (4-39) and on geometrical properties of the set of 3 space vectors involved in it. Thus, if the estimation error ΔL is positive then the space vector $\hat{\underline{V}}_{\Delta}(k+1)$ is situated between $\Delta\underline{i}(k+1)$ and $\Delta\underline{i}(k)$ as illustrated by Fig. 4-5 (a). On the other hand, if ΔL is negative then $\Delta\underline{i}(k+1)$ lies between $\Delta\underline{i}(k)$ and $\hat{\underline{V}}_{\Delta}(k+1)$. In case $\Delta L=0$, the direction of $\hat{\underline{V}}_{\Delta}(k+1)$ will be the same as the direction of $\Delta\underline{i}(k+1)$.

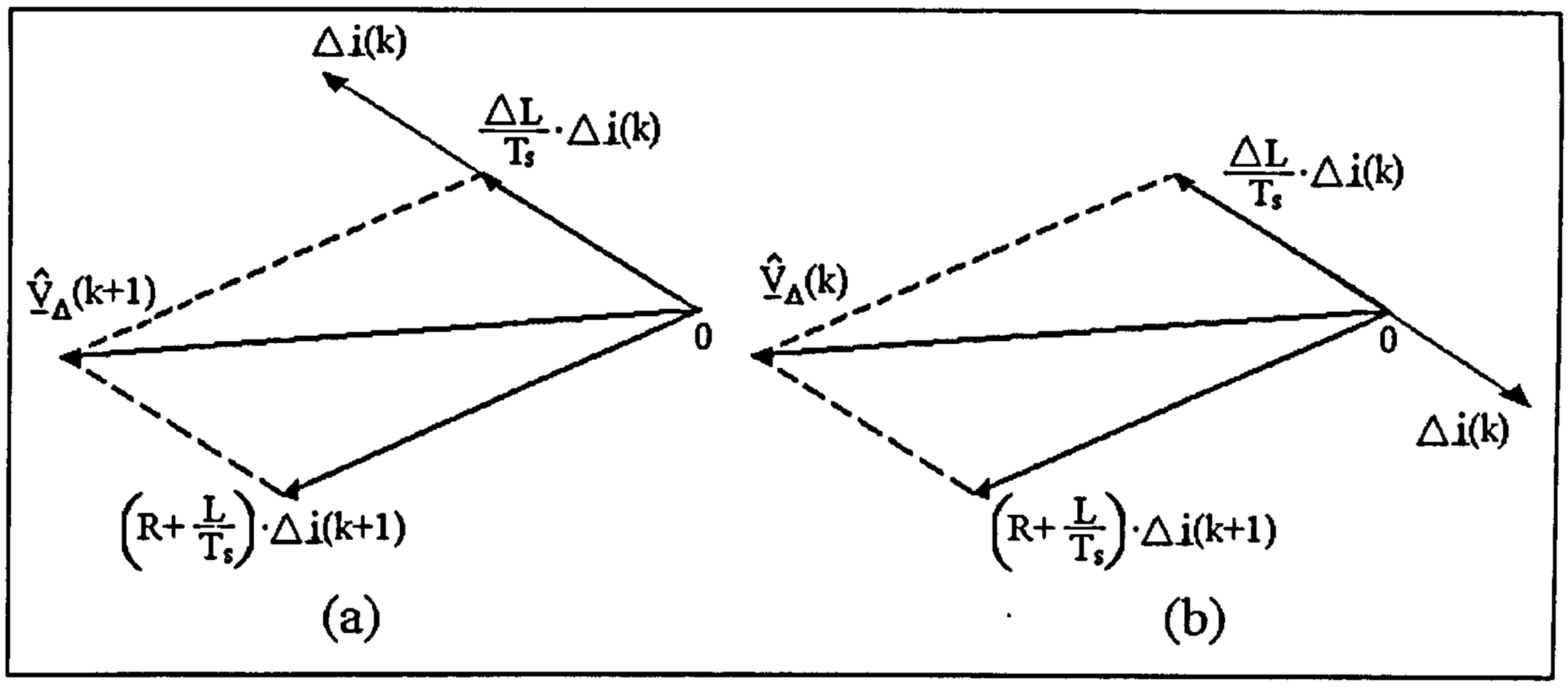

Fig. 4-5 - Inductance estimation principle

The estimated inductance $\hat{L} = L + \Delta L$ needs to be corrected by increasing it whenever the situation in Fig. 4-5 (b) occurs, and by decreasing it in the situation illustrated by Fig. 4-5 (a). The algorithm is concisely expressed as

$$\begin{cases} \hat{L}(k+1) = \hat{L}(k) + \delta L & \text{if } \Delta\underline{i}(k+1) \text{ is between } \underline{\hat{V}}_{\Delta}(k+1) \text{ and } \Delta\underline{i}(k) \\ \hat{L}(k+1) = \hat{L}(k) - \delta L & \text{if } \underline{\hat{V}}_{\Delta}(k+1) \text{ is between } \Delta\underline{i}(k) \text{ and } \Delta\underline{i}(k+1) \end{cases} \tag{4-40}$$

where the increment step δL is a small positive quantity.

The presented algorithm operates correctly only if $|\Delta\underline{e}(k+1)|$ is negligible. The validity conditions (4-38) for induction estimation are a prerequisite for obtaining accurate estimation values. The larger the value of $|\Delta\underline{e}(k+1)|$ the larger the estimation errors. Other factors that influence the induction estimation process are the quantisation error of the A/D converters and the value of the step δL. Due to the quantisation error, the vectors $\underline{\hat{V}}_{\Delta}(k+1)$ and $\Delta\underline{i}(k+1)$ are not always on the same direction even if the inductance estimation is correct. This causes small fluctuations of the estimated value after the approximate inductance has already been calculated. The amplitude of the fluctuations is proportional to the increment step size δL and therefore it has to be small to ensure good estimation precision.

4.2.3 The Conditions For Accurate Current Control

As previously demonstrated, the PWM current controller operates correctly if the sampling frequency is high. Two conditions need to fulfilled:

1) The sampling frequency has to be sufficiently high to ensure that the approximate expression (4-13) of the non-inductive voltage $\underline{V}_{ni}$ is valid.

2) The sampling frequency needs to be high enough to ensure that the variations of the internal voltage $|\Delta \underline{e}(k+1)|$ fulfil the conditions (4-38) for accurate inductance estimation.

This section investigates the limitations imposed by these conditions in the particular case when the inverter load is an induction motor. The calculations presented involve a series of approximations without diminishing the generality of the conclusions. The values of the motor parameters in Table 4.1 are used as a guide to determine the validity of the approximations.

4.2.3.1 The Accurate Non-Inductive Voltage Calculation

Relation (4-13) yields accurate results only if the stator current derivative can be considered constant during a single sampling period T_s. This condition ensures an almost linear current variation during one sampling period and allows approximating the current derivative with the current variation as shown in (4-41).

$$\frac{d\underline{i}_s^s}{dt} \approx \frac{\Delta \underline{i}_s^s}{T_s} \tag{4-41}$$

The variation of the stator current derivative can be expressed as the Taylor series

$$\frac{d\underline{i}_s^s}{dt}(t+T_s) = \frac{d\underline{i}_s^s}{dt}(t) + \frac{T_s}{1!}\cdot\frac{d^2\underline{i}_s^s}{dt^2}(t) + \frac{T_s^2}{2!}\cdot\frac{d^3\underline{i}_s^s}{dt^3}(t) + ... \tag{4-42}$$

If only the first two terms of the series are taken into account, the accuracy condition (4-41) can be mathematically expressed as

$$\frac{d\underline{i}_s^s}{dt}(t) \approx \frac{d\underline{i}_s^s}{dt}(t+T_s) \Leftrightarrow T_s \cdot \left|\frac{d^2\underline{i}_s^s}{dt^2}\right| << \left|\frac{d\underline{i}_s^s}{dt}\right| \tag{4-43}$$

Consequently, the general equation system (4-4) needs to be solved first in order to determine the stator current as a function of the stator voltage. The rotor inertia keeps the motor speed almost constant during one sampling period T_s. As a consequence, the differential equation system in (4-4) can be considered linear in these conditions. The elimination procedure, which is used to solve linear differential equation systems, requires that the system is written in terms of differential operators 'D', so that (4-4) becomes

$$\begin{cases} \underline{u}_s^s = D_{ss}\,\underline{i}_s^s + D_{sr}\,\underline{i}_r^s \\ 0 = D_{rs}\,\underline{i}_s^s + D_{rr}\,\underline{i}_r^s \end{cases} \tag{4-44}$$

The four differential operators used in (4-44) are defined as:

$$\begin{cases} D_{ss} = L_s \dfrac{d}{dt} + R_s \\ D_{sr} = L_m \dfrac{d}{dt} \\ D_{rs} = L_m \dfrac{d}{dt} - j\omega_{er} L_m \\ D_{rr} = L_r \dfrac{d}{dt} + R_r - j\omega_{er} L_r \end{cases} \tag{4-45}$$

According to the elimination method, the stator current is the solution of the differential equation

$$\left(D_{ss} D_{rr} - D_{sr} D_{rs}\right) \cdot \underline{i}_s^s = D_{rr} \underline{u}_s^s \tag{4-46}$$

which is equivalent to

$$\begin{cases} a \cdot \dfrac{d^2 \underline{i}_s^s}{dt^2} + b \cdot \dfrac{d\underline{i}_s^s}{dt} + c \cdot \underline{i}_s^s = \left(R_r - j\omega_{er} L_r\right) \cdot \underline{u}_s^s \\ a = L_s L_r - L_m^2 \\ b = L_s R_r + L_r R_s - j\omega_{er}\left(L_s L_r - L_m^2\right) \\ c = R_s R_r - j\omega_{er} L_r R_s \end{cases} \tag{4-47}$$

The general form of the solution is illustrated by

$$\underline{i}_s^s = K_1 \cdot e^{p_1 \cdot t} + K_2 \cdot e^{p_2 \cdot t} + \frac{\left(R_r - j\omega_{er} L_r\right) \cdot \underline{u}_s^s}{R_s R_r - j\omega_{er} L_r R_s} = K_1 \cdot e^{p_1 \cdot t} + K_2 \cdot e^{p_2 \cdot t} + \frac{\underline{u}_s^s}{R_s} \tag{4-48}$$

where K_1 and K_2 are constants depending on the initial conditions, while p_1 and p_2 are the solutions of the polynomial characteristic equation (4-49) attached to (4-47).

$$a \cdot p^2 + b \cdot p + c = 0 \tag{4-49}$$

Thus, p_1 and p_2 are given by

$$\begin{cases} p_1 = \dfrac{-L_s R_r - L_r R_s + j\omega_{er} L L_r - \sqrt{\Delta}}{2 \cdot L L_r} \\ p_2 = \dfrac{-L_s R_r - L_r R_s + j\omega_{er} L L_r + \sqrt{\Delta}}{2 \cdot L L_r} \\ \Delta = \left(L_s R_r + L_r R_s - j\omega_{er} L L_r\right)^2 - 4 \cdot L L_r \left(R_s R_r - j\omega_{er} L_r R_s\right) \end{cases} \tag{4-50}$$

The first and second derivatives of the stator current are

$$\begin{cases} \dfrac{d\underline{i}_s^s}{dt} = p_1 K_1 \cdot e^{p_1 t} + p_2 K_2 \cdot e^{p_2 t} \\ \dfrac{d^2\underline{i}_s^s}{dt^2} = p_1^2 K_1 \cdot e^{p_1 t} + p_2^2 K_2 \cdot e^{p_2 t} \end{cases} \tag{4-51}$$

Expressions (4-51) demonstrate that p_1 and p_2 strongly influence the relationship between the values of the two derivatives. They are the essential factors that determine the minimal acceptable switching frequency that validates condition (4-43). This is fulfilled if p_1 and p_2 have such values that the following relations are simultaneously true:

$$\begin{cases} T_s \cdot \left| p_1^2 K_1 \cdot e^{p_1 t} \right| << \left| p_1 K_1 \cdot e^{p_1 t} \right| \quad \Leftrightarrow \quad T_s \cdot p_1 << 1 \\ T_s \cdot \left| p_2^2 K_2 \cdot e^{p_1 t} \right| << \left| p_2 K_2 \cdot e^{p_1 t} \right| \quad \Leftrightarrow \quad T_s \cdot p_2 << 1 \end{cases} \tag{4-52}$$

They can be expressed as a single condition:

$$T_s \cdot \max\left\{ |p_1|, |p_2| \right\} << 1 \tag{4-53}$$

Fig. 4-6 illustrates the dependency of p_1 and p_2 on the rotor speed calculated for the 11.1 kW induction motor in Table 4.1. Thus, at very low speeds, the imaginary parts in (4-50) can be neglected while the absolute real parts are given by

$$\begin{cases} \lim\limits_{\omega_{er} \to 0} \left| \mathrm{Re}\{p_1\} \right| = \left| \dfrac{-L_s R_r - L_r R_s - \sqrt{(L_s R_r + L_r R_s)^2 - 4 \cdot L L_r R_r R_s}}{2 \cdot L L_r} \right| \\ \lim\limits_{\omega_{er} \to 0} \left| \mathrm{Re}\{p_2\} \right| = \left| \dfrac{-L_s R_r - L_r R_s + \sqrt{(L_s R_r + L_r R_s)^2 - 4 \cdot L L_r R_r R_s}}{2 \cdot L L_r} \right| \end{cases} \tag{4-54}$$

According to the considerations presented in section 4.1, the equivalent load induction L is small in comparison with L_r and L_s. Consequently,

$$4 \cdot L L_r R_r R_s << (L_s R_r + L_r R_s)^2 \tag{4-55}$$

and the first absolute real part in (4-54) is much larger than the second absolute real part, as shown in Fig. 4-6 and demonstrated by (4-56).

$$\begin{cases} \lim\limits_{\omega_{er} \to 0} \left| \mathrm{Re}\{p_1\} \right| \approx \dfrac{L_s R_r + L_r R_s}{2 \cdot L L_r} \\ \lim\limits_{\omega_{er} \to 0} \left| \mathrm{Re}\{p_2\} \right| \approx 0 \end{cases} \tag{4-56}$$

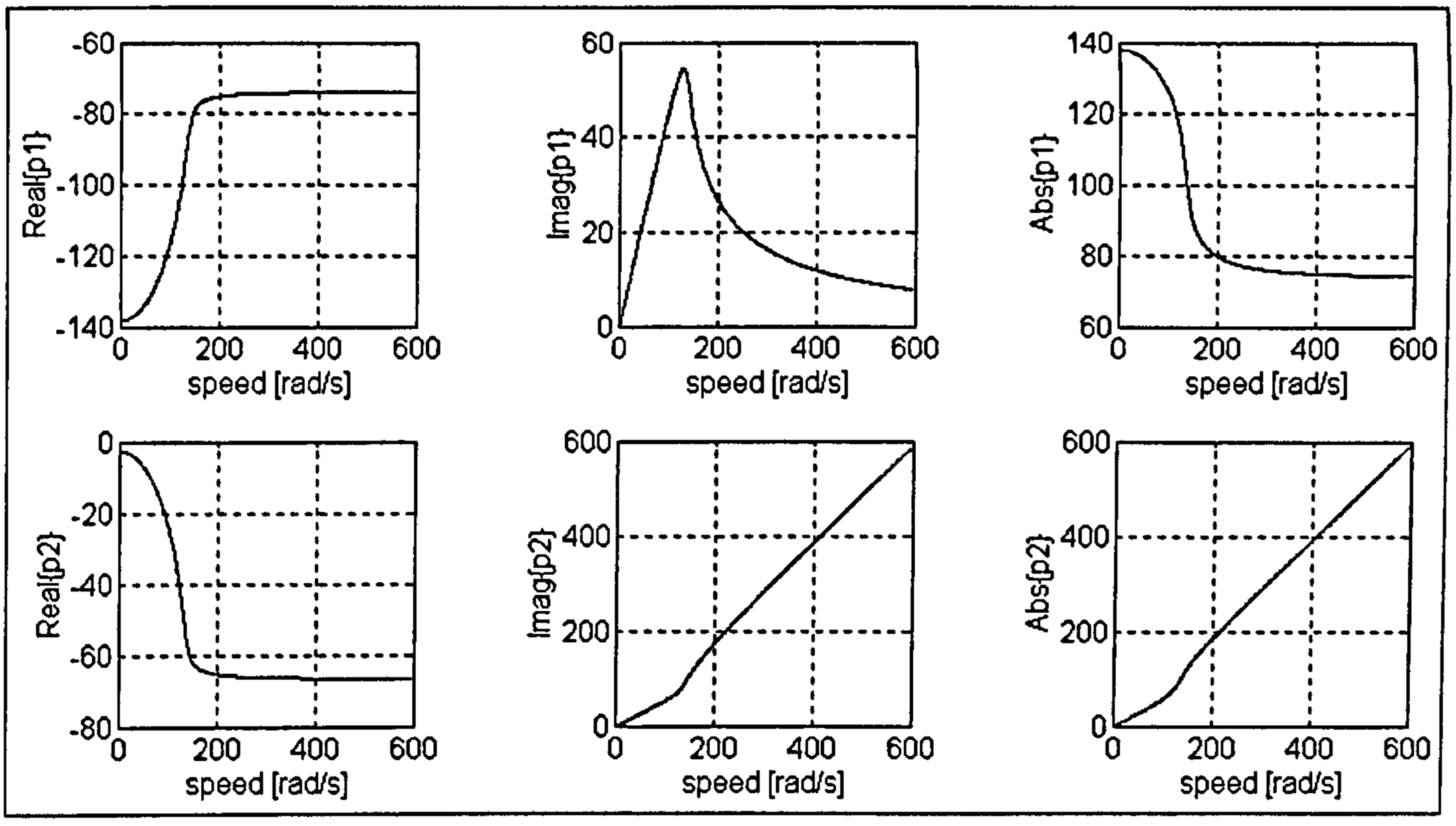

Fig. 4-6 The variation of p_1 and p_2 with the motor speed (calculations for a motor of 11.1 kW)

At high speeds, the imaginary parts cannot be neglected. At speeds around the motor rated value or higher, the absolute imaginary part of p_2 is approximately proportional to the rotor speed ω_{er}, while the absolute real parts are almost equal and have low values. To demonstrate this, the expression of Δ is transformed by substituting the expression (4-7) of L, the result being

$$\Delta = \left(L_s R_r + L_r R_s - j\omega_{er} L L_r\right)^2 - 4L_s L_r R_s R_r + 4L_m^2 R_s R_r + 4j\omega_{er} L L_r^2 R_s \qquad (4\text{-}57)$$

Using basic algebraic calculations, equation (4-57) can be written as

$$\Delta = \left(-L_s R_r + L_r R_s + j\omega_{er} L L_r\right)^2 + 4L_m^2 R_s R_r \qquad (4\text{-}58)$$

The quantity between brackets is very large when the rotor speed is high and the square root of Δ is well approximated by

$$\lim_{\omega_{er}\to\infty} \sqrt{\Delta} = \left(-L_s R_r + L_r R_s + j\omega_{er} L L_r\right) \qquad (4\text{-}59)$$

Therefore, the values of p_1 and p_2 at high motor speed are given by

$$\begin{cases} \lim\limits_{\omega_{er}\to\infty} p_1 = \dfrac{-L_s R_r - L_r R_s + j\omega_{er} L L_r - \lim\limits_{\omega_{er}\to\infty} \sqrt{\Delta}}{2 \cdot L L_r} = \dfrac{-L_r R_s}{L L_r} = -\dfrac{R_s}{L} \\ \lim\limits_{\omega_{er}\to\infty} p_2 = \dfrac{-L_s R_r - L_r R_s + j\omega_{er} L L_r + \lim\limits_{\omega_{er}\to\infty} \sqrt{\Delta}}{2 \cdot L L_r} = \dfrac{-L_s R_r + j \cdot \omega_{er} L L_r}{L L_r} \end{cases} \qquad (4\text{-}60)$$

Table 4.1 shows that the stator and the rotor resistances are always the same order of magnitude. Moreover, in many situations they have close values. Based on this observation, the real and the imaginary parts at high speed are approximated by

$$\begin{cases} \lim\limits_{\omega_{er}\to\infty}\left|\mathrm{Re}\{p_1\}\right| \approx \lim\limits_{\omega_{er}\to\infty}\left|\mathrm{Re}\{p_2\}\right| \approx \lim\limits_{\omega_{er}\to 0}\left|\mathrm{Re}\{p_1\}\right| \\ \lim\limits_{\omega_{er}\to\infty}\left|\mathrm{Im}\{p_1\}\right| = 0 \\ \lim\limits_{\omega_{er}\to\infty}\left|\mathrm{Im}\{p_2\}\right| = \omega_{er} \end{cases} \quad (4\text{-}61)$$

The maximum value of modules $|p_1|$ and $|p_2|$ can be calculated as a function of the results in (4-56) and (4-61) as follows:

$$\max\{|p_1|,|p_2|\} < \sqrt{[\max\{\mathrm{Re}(p_1),\mathrm{Re}(p_2)\}]^2 + [\max\{\mathrm{Im}(p_1),\mathrm{Im}(p_2)\}]^2} \quad (4\text{-}62)$$

$$\max\{|p_1|,|p_2|\} < \sqrt{\left(\lim_{\omega_{er}\to 0}\mathrm{Re}\{p_1\}\right)^2 + \left(\lim_{\omega_{er}\to\infty}\mathrm{Im}\{p_2\}\right)^2} = \sqrt{\frac{(L_sR_r + L_rR_s)^2}{4L^2L_r^2} + \omega_{er}^2} \quad (4\text{-}63)$$

Therefore the maximal value of $|p_1|$ and $|p_2|$ is obtained when the motor attains its maximal speed ω_{max}. The angular frequency squared $\omega_{er}^2 = \omega_{max}^2$ is much larger than the other term in (4-63) Therefore the relationship given in (4-63) can be approximated as

$$p_{max} = \max_{\omega_{er}}\{|p_1|,|p_2|\} = \omega_{max}\cdot\sqrt{\frac{(L_sR_r + L_rR_s)^2}{4L^2L_r^2\cdot\omega_{max}^2} + 1} \approx \omega_{max} \quad (4\text{-}64)$$

Using the result in (4-64), the condition (4-53) becomes

$$T_s\cdot\omega_{max} << 1 \quad (4\text{-}65)$$

The maximum motor speed does not surpass twice the rated speed in normal drive systems, which means the sampling frequency f_s must comply with the condition

$$f_s = \frac{1}{T_s} >> \omega_{max} = 2\pi\cdot 100 = 628\,\mathrm{Hz} \quad (4\text{-}66)$$

4.2.3.2 The Mathematical Conditions for Accurate Induction Estimation

The initial form (4-38) of the conditions for accurate inductance estimation can be divided by T_s and transformed into

$$\begin{cases} \dfrac{|\Delta \underline{e}(k+1)|}{T_s} << \left(R + \dfrac{L}{T_s}\right) \cdot \dfrac{|\Delta \underline{i}(k+1)|}{T_s} \\ \dfrac{|\Delta \underline{e}(k+1)|}{T_s} << \dfrac{\Delta L}{T_s} \cdot \dfrac{|\Delta \underline{i}(k)|}{T_s} \end{cases} \tag{4-67}$$

If the condition (4-66) is fulfilled then the variation of vectors $\underline{e}$ and $\underline{i}$ is almost linear during a sampling period. Thus, the vector variations can be well approximated using the vector derivatives, and the inequalities between vector variations become inequalities between vector derivatives.

$$\begin{cases} \left|\dfrac{d\underline{e}}{dt}\right| << \left(R + \dfrac{L}{T_s}\right) \cdot \left|\dfrac{d\underline{i}}{dt}\right| \\ \left|\dfrac{d\underline{e}(kT_s + T_s)}{dt}\right| << \dfrac{\Delta L}{T_s} \cdot \left|\dfrac{d\underline{i}(kT_s)}{dt}\right| \end{cases} \tag{4-68}$$

Based on the basic equation of the R-L-e circuit (4-1), and on the property that $d\underline{u}/dt=0$ during one sampling period T_s, the derivative of the internal voltage $\underline{e}$ is obtained as follows:

$$\frac{d\underline{e}}{dt} = -R\frac{d\underline{i}}{dt} - L\frac{d^2\underline{i}}{dt^2} - \frac{d\underline{u}}{dt} = -R\frac{d\underline{i}_s^s}{dt} - L\frac{d^2\underline{i}_s^s}{dt^2} \tag{4-69}$$

Using the results in (4-51), the relationship between the first and the second derivative of the stator current can be written as

$$\left|\frac{d^2\underline{i}_s^s}{dt}\right| < \max\{|p_1|, |p_2|\} \cdot \left|\frac{d\underline{i}_s^s}{dt}\right| = \omega_{max} \cdot \left|\frac{d\underline{i}_s^s}{dt}\right| \tag{4-70}$$

Therefore, the module of the internal voltage derivative complies with

$$\left|\frac{d\underline{e}}{dt}\right| = \left|R_s\frac{d\underline{i}_s^s}{dt} + L\frac{d^2\underline{i}_s^s}{dt^2}\right| \le \left|R_s\frac{d\underline{i}_s^s}{dt}\right| + \left|L\frac{d^2\underline{i}_s^s}{dt^2}\right| < \left|R_s\frac{d\underline{i}_s^s}{dt}\right| + \left|L\omega_{max} \cdot \frac{d\underline{i}_s^s}{dt}\right| \tag{4-71}$$

and

$$\left|\frac{d\underline{e}}{dt}\right| < \left(R_s + L\omega_{max}\right) \cdot \left|\frac{d\underline{i}_s^s}{dt}\right| \tag{4-72}$$

Thus, the first condition for accurate induction estimation becomes

$$\left|\frac{d\underline{e}}{dt}\right| < \left(R_s + L\omega_{max}\right) \cdot \left|\frac{d\underline{i}_s^s}{dt}\right| << \left(R_s + \frac{L}{T_s}\right) \cdot \left|\frac{d\underline{i}_s^s}{dt}\right| \Leftrightarrow \frac{1}{T_s} >> \omega_{max} \tag{4-73}$$

which is equivalent to condition (4-66).

The second condition involves two derivatives at two different moments in time. To simplify the mathematical calculations, it can be replaced with the more restrictive condition (4-74) that contains simultaneous derivatives.

$$\left|\frac{d\underline{e}(kT_s+T_s)}{dt}\right| << \frac{\Delta L}{T_s}\cdot\left|\frac{d\underline{i}(kT_s)}{dt}\right| \Leftarrow \max\left\{\left|\frac{d\underline{e}}{dt}\right|\right\} << \frac{\Delta L}{T_s}\cdot\min\left\{\left|\frac{d\underline{i}}{dt}\right|\right\} \quad (4\text{-}74)$$

The minimal current derivative can be estimated on the base of equation

$$L\left|\frac{d\underline{i}}{dt}\right| = \left|\underline{u} - R\underline{i} - \underline{e}\right| = \left|\underline{u} - \underline{V}_{ni}\right| \quad (4\text{-}75)$$

derived from (4-1). To perform the calculations it is necessary to estimate the average module of the quantities involved.

The stator and rotor currents depend mostly on the first harmonic of the voltage because the effects of the higher harmonics are filtered out by the motor inductances. The internal voltage depends on the motor currents, therefore its average module is also determined by the voltage fundamental harmonic. The average module of the non-inductive voltage can be determined by rewriting equation (4-1) in terms of fundamental harmonics:

$$\underline{U}_s = j\omega_{es}L\underline{I}_s + R_s\underline{I}_s + \underline{E} \quad (4\text{-}76)$$

The internal voltage depends on the amplitude of the motor currents, on the rotor angular frequency ω_{er} and on the parameters R_r, L_r, L_s as shown in (4-7). The inductance L in the equivalent circuit is much smaller than L_s and L_r, so that the corresponding voltage component in (4-76) is much smaller than $\underline{E}$. Thus, the amplitude of the non-inductive voltage approximates the amplitude of the stator voltage fundamental harmonic:

$$\left|R_s\underline{i}_s + \underline{E}\right| = \left|\underline{U}_s - L\omega_{es}\underline{I}_s\right| \cong \left|\underline{U}_s\right| \quad (4\text{-}77)$$

The instantaneous value of the internal voltage is proportional to the motor currents. Therefore, the vector $\underline{e}$ suffers small but high frequency oscillations. However these oscillations do not significantly affect the amplitude $|\underline{E}|$ of the internal voltage. On the other hand, due to the PWM modulation, the amplitude of the fundamental harmonic does not surpass the amplitude of the PWM square pulses. This conclusion is illustrated in (4-78) where $k_{PWM}<1$ is the PWM modulation index.

$$k_{PWM}\cdot\left|\underline{u}_s^s\right| = \left|\underline{U}_s\right| \quad (4\text{-}78)$$

The minimal and maximal stator current values are obtained when the stator voltage and the non-inductive voltage are parallel and anti-parallel, respectively.

$$\begin{cases} \min\left|\frac{d\underline{i}_s}{dt}\right| = \frac{1}{L}\left|\underline{U}_s - k_{PWM}\underline{U}_s\right| = \frac{1}{L}\left(1-k_{PWM}\right)\cdot\left|\underline{u}_s^s\right| \\ \max\left|\frac{d\underline{i}_s}{dt}\right| = \frac{1}{L}\left|\underline{U}_s + k_{PWM}\underline{U}_s\right| = \frac{1}{L}\left(1+k_{PWM}\right)\cdot\left|\underline{u}_s^s\right| \end{cases} \quad (4\text{-}79)$$

Combining the second relation (4-79) with (4-72), the result is

$$\max\left|\frac{d\underline{e}}{dt}\right| < \left(R_s + L\omega_{max}\right)\cdot\max\left|\frac{d\underline{i}_s^s}{dt}\right| = \left(\frac{R_s}{L} + \omega_{max}\right)\left(1+k_{PWM}\right)\cdot\left|\underline{u}_s^s\right| \quad (4\text{-}80)$$

Therefore, the second condition for accurate induction estimation is transformed into

$$\left(\frac{R_s}{L} + \omega_{max}\right)\cdot\left(1+k_{PWM}\right)\cdot\left|\underline{u}_s^s\right| << \frac{\Delta L}{L\cdot T_s}\min\left|\frac{d\underline{i}_s^s}{dt}\right| = \frac{\Delta L}{L\cdot T_s}\left(1-k_{PWM}\right)\cdot\left|\underline{u}_s^s\right| \quad (4\text{-}81)$$

The sampling frequency f_s which fulfils this condition is given by

$$f_s = \frac{1}{T_s} >> \frac{1+k_{PWM}}{1-k_{PWM}}\cdot\frac{R_s + L\omega_{max}}{\Delta L} \quad (4\text{-}82)$$

The accepted inductance estimation error ΔL is an important factor that influences the minimal sampling frequency. Very accurate inductance estimations imply small ΔL that, according to (4-82), require high sampling frequencies. Less accurate inductance estimation can be performed at lower sampling frequency. For example, the inductance estimation with a precision of 0.5 mH for the 11.1 kW motor in Table 4.1, when k_{PWM}=0.7, requires that the f_s is much larger than 46.2 kHz. Therefore, an adequate sampling frequency is 450 kHz. Even larger sampling frequencies are required if either k_{PWM} has larger values or if higher estimation precision is required. The practical solution to limit the sampling frequency f_s while obtaining accurate induction estimation is to perform the estimation process at small stator angular frequency ω_{es}.

In conclusion, the lower limit for the sampling frequency has been calculated. It is defined by conditions (4-66), (4-73) and (4-82) which must be fulfilled simultaneously. The first two conditions are less restrictive as they depend only on the maximal motor speed. The last condition is more restrictive and depends on the motor speed, on its electrical parameters and on the PWM modulation index. Consequently, condition (4-82) alone can be used for practical calculations as any sampling frequency determined based on (4-82) also fulfils conditions (4-66) and (4-73).

4.2.4 Current Control Implementation Methods

The implementation of the two interrelated algorithms can be performed using DSPs or using specialised digital architectures (ASICs or FPGAs). The DSPs approach is simple because the corresponding software development is straightforward. However, the two algorithms imply a large number of time-consuming mathematical operations to be performed for each PWM pulse and therefore they use most of the DSP resources, limiting its capability to perform the speed control task. The use of specialised digital structures ensures fast operation and allows the speed control and the current control algorithms to be performed in parallel.

There are two possible software implementations for the current control algorithm: the direct implementation of calculating all the angles, and the indirect implementation that uses scalar products between vectors to find the optimal output voltage. The direct implementation implies calculating the six necessary angles using "arctan" trigonometric function as shown in (4-83), and then comparing the results.

$$\alpha = \begin{cases} \arctan\left(\dfrac{\mathrm{Im}\{\underline{u}_{INV} - \underline{V}_1\}}{\mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\}}\right) & \text{when } \mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\} > 0 \\ \pi - \arctan\left(\dfrac{\mathrm{Im}\{\underline{u}_{INV} - \underline{V}_1\}}{\mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\}}\right) & \text{when } \mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\} < 0 \\ \dfrac{\pi}{2} & \text{when } \begin{cases} \mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\} = 0 \\ \mathrm{Im}\{\underline{u}_{INV} - \underline{V}_1\} > 0 \end{cases} \\ -\dfrac{\pi}{2} & \text{when } \begin{cases} \mathrm{Re}\{\underline{u}_{INV} - \underline{V}_1\} = 0 \\ \mathrm{Im}\{\underline{u}_{INV} - \underline{V}_1\} < 0 \end{cases} \end{cases} \quad (4\text{-}83)$$

The "arctan" function can be implemented as a look-up table thereby accelerating the calculations but the sequentially performed subtractions, divisions and comparisons required by (4-83), considerably slow down the calculation process. A realistic estimate of the computational effort can be obtained considering that the first two possibilities in (4-83) are equally probable while the last two are unlikely to be fulfilled due to the exact equalities which are involved. In this case between 12 and 18 subtractions (depending how many times case (a) and case (b) occur in (4-83)) are requested. Additionally, a further 6 divisions and between 6+6=12 and 9+6=15 comparisons have to be performed for each sampling period.

There are DSPs containing on-chip RAM and on-chip maskable ROM (for instance TMS320C5x) [8]. Consequently, both the control program and the look-up table can reside either on-chip or off-chip in an external EPROM. On-chip configuration

is advantageous because it is compact, reliable and simplifies the PCB design. Unfortunately, the on-chip ROM memory space is limited. A total of $8 \cdot 2^{10}$ memory words are available for TMS320C51 and twice as much for TMS320C53 [8]. If the complete motor control program is large then there might be no available space for a look-up table, which will have to be placed in an external EPROM.

In case a simple and compact hardware implementation is aimed, alternative algorithms must be used to eliminate the need for the external EPROM memory chip. As a result, the trigonometric function calculations have to be avoided because any specialised routine for calculating such functions is a huge time consumer. The indirect implementation avoids trigonometric functions by utilising scalar products, which are performed between each of the 6 vectors and the reference vector. By definition the scalar product between two n-dimensional vectors $\bar{a}$ and $\bar{b}$ is given by

$$\bar{a} \cdot \bar{b} = \left|\bar{a}\right| \cdot \left|\bar{b}\right| \cdot \cos\varphi \tag{4-84}$$

The smaller the angle φ between the two vectors, the larger the result. Thus, the correct output voltage is chosen by maximising the corresponding scalar product. The calculations are relevant only if the 6 vectors have the same module. To fulfil this condition, the six vectors need to be normalised. Consequently the 6 scalar products p_j (where j=1,2,...6) have to be calculated according to

$$p_j = \frac{\mathrm{Re}\{\Delta i_{ref}\} \cdot \mathrm{Re}\{u_{INV}^{(j)} - \underline{V}_1\} + \mathrm{Im}\{\Delta i_{ref}\} \cdot \mathrm{Im}\{u_{INV}^{(j)} - \underline{V}_1\}}{\sqrt{\mathrm{Re}\{u_{INV}^{(j)} - \underline{V}_1\}^2 + \mathrm{Im}\{u_{INV}^{(j)} - \underline{V}_1\}^2}} \tag{4-85}$$

where $u_{INV}^{(j)}$ is one of the inverter output voltages and Δi_{ref} is the current error vector: $\Delta i_{ref} = i_{ref}(k+1) - i(k)$. To avoid square root calculations, the equation (4-85) can be replaced by

$$p_j^2 = \frac{\left(\mathrm{Re}\{\Delta i_{ref}\} \cdot \mathrm{Re}\{u_{INV}^{(j)} - \underline{V}_1\} + \mathrm{Im}\{\Delta i_{ref}\} \cdot \mathrm{Im}\{u_{INV}^{(j)} - \underline{V}_1\}\right)^2}{\mathrm{Re}\{u_{INV}^{(j)} - \underline{V}_1\}^2 + \mathrm{Im}\{u_{INV}^{(j)} - \underline{V}_1\}^2} \tag{4-86}$$

The actual voltage generated by the inverter is $u_{INV}^{(J_{max})}$, where the index J_{max} is determined from the condition

$$p_{J_{max}}^2 = \max_j \{p_j^2\} \tag{4-87}$$

A total of 30 multiplications, 12 subtractions, and 12 additions are required to perform the respective calculations. Clearly, the second approach requires much more resources than the first one.

The software implementation of the inductance estimation algorithm necessitates that the relative positions of three vectors $\Delta\underline{i}(k)$, $\Delta\underline{i}(k+1)$ and $\underline{\hat{V}}_{\Delta}(k+1)$ is expressed in algebraic terms. Three different vectors in a plane are never linearly independent, one is always a linear combination of the other two. Thus, the real and the imaginary part of $\underline{\hat{V}}_{\Delta}(k+1)$ can be written as:

$$\begin{cases} \mathrm{Re}\{\underline{V}_{\Delta}(k+1)\} = \alpha \cdot \mathrm{Re}\{\Delta\underline{i}(k)\} + \beta \cdot \mathrm{Re}\{\Delta\underline{i}(k+1)\} \\ \mathrm{Im}\{\underline{V}_{\Delta}(k+1)\} = \alpha \cdot \mathrm{Im}\{\Delta\underline{i}(k)\} + \beta \cdot \mathrm{Im}\{\Delta\underline{i}(k+1)\} \end{cases} \quad \textbf{(4-88)}$$

where α and β are two real quantities. The vector $\underline{\hat{V}}_{\Delta}(k+1)$ is situated between $\Delta\underline{i}(k)$ and $\Delta\underline{i}(k+1)$ if and only if $\alpha>0$ and $\beta>0$, while $\Delta\underline{i}(k+1)$ lies between $\Delta\underline{i}(k)$ and $\underline{\hat{V}}_{\Delta}(k+1)$ if and only if $\alpha<0$ and $\beta>0$. The two cases are illustrated by Fig. 4-7 where $\underline{\hat{V}}_{\Delta 1}(k+1)$ corresponds to first situation ($\alpha_1>0$) and $\underline{\hat{V}}_{\Delta 2}(k+1)$ corresponds to the second situation ($\alpha_2<0$).

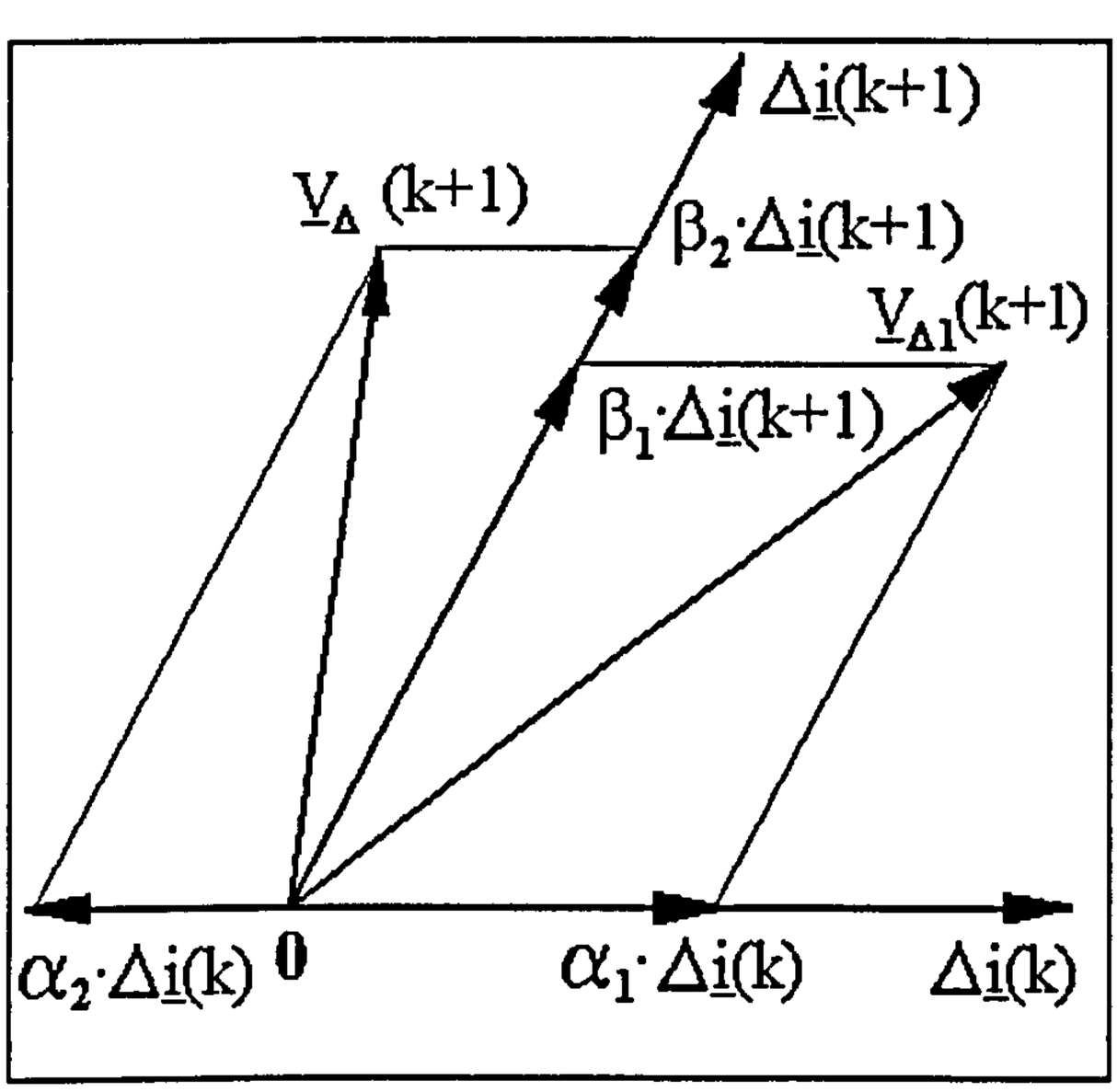

Fig. 4-7 - Three vector problem: graphical representation ($\alpha_1>0$; $\alpha_2<0$)

The equation system (4-88) has to be solved in order to determine the relative position of the three vectors as requested by the inductance estimation algorithm. The system solution is

$$\begin{cases} \alpha = \dfrac{\mathrm{Re}\{\underline{V}_2(k+1)\}\cdot\mathrm{Im}\{\Delta\underline{i}(k+1)\} - \mathrm{Im}\{\underline{V}_2(k+1)\}\cdot\mathrm{Re}\{\Delta\underline{i}(k+1)\}}{\mathrm{Re}\{\Delta\underline{i}(k)\}\cdot\mathrm{Im}\{\Delta\underline{i}(k+1)\} - \mathrm{Im}\{\underline{i}(k)\}\cdot\mathrm{Re}\{\Delta\underline{i}(k+1)\}} \\ \beta = \dfrac{\mathrm{Re}\{\Delta\underline{i}(k)\}\cdot\mathrm{Im}\{\underline{V}_2(k+1)\} - \mathrm{Im}\{\Delta\underline{i}(k)\}\cdot\mathrm{Re}\{\underline{V}_2(k+1)\}}{\mathrm{Re}\{\Delta\underline{i}(k)\}\cdot\mathrm{Im}\{\Delta\underline{i}(k+1)\} - \mathrm{Im}\{\underline{i}(k)\}\cdot\mathrm{Re}\{\Delta\underline{i}(k+1)\}} \end{cases} \quad (4\text{-}89)$$

As a result, the inductance estimation algorithm requires 6 multiplications 2 divisions and 3 subtractions to be performed in order to determine α and β. Depending on the sign of two quantities, the decision to correct the present inductance estimation is eventually taken according to

$$\hat{L}(k+1) = \begin{cases} \hat{L}(k) + \delta L & \text{when} \alpha < 0 \text{ and } \beta > 0 \\ \hat{L}(k) - \delta L & \text{when} \alpha > 0 \text{ and } \beta > 0 \end{cases} \quad (4\text{-}90)$$

which may imply another addition or subtraction. Consequently, a total of 11 or 12 mathematical operations are needed at each estimation step.

The combined software implementation of the current control and inductance estimation algorithms requires a very large number of mathematical operations to be performed. In case where the direct current control implementation is employed, then up to 12+24=36 algebraic calculations and 15+2=17 comparisons are requested for each algorithm step. The use of the indirect current control implementation in order to eliminate the need for look-up tables requires up to 12+54=66 algebraic operations and 5+2=7 comparisons for each algorithm step. In a typical situation where the PWM frequency of 20 kHz and T_{PWM}=10·Ts, the algorithm requires that 200,000 calculation steps are performed each second. This amounts to a total of 7,200,000 algebraic operations plus 3,400,000 comparisons per second in the case of the direct implementation. The indirect implementation requires a computational effort of 13,200,000 algebraic operations and 1,400,000 comparisons per second.

A DSP program created to perform all these operations contains supplementary instructions: reading operands from memory, writing results to memory, program control instructions (jumps) and so on. An optimistic estimate is that the calculation and comparison instruction number is approximately equal to the number of all other instructions in the program. That means that a speed of at least 30,000,000 instructions per second (30 MIPS) is actually required for on-line operation.

The DSPs commonly used in drive system applications belong to the TMS320C3x and TMS320C5x series because they are fast and inexpensive processors. The of 16-bit, fixed-point DSPs TMS320C5x generation perform up to 50 MIPS while some of the

TMS320C3x perform 30 MIPS but have a separate 60 MFLOPS floating point arithmetic unit, which increases the total computation power tremendously. The latest DSP devices offer much larger calculation speeds (TMS320C6x generation offers 1600 MIPS) but their price is still too high for inexpensive drive control applications. Thus, the computation effort required by the adopted current control strategy can be handled by an inexpensive DSP. However, the algorithm consumes a significant part of its total resources (up to 33% if a DSP from the TMS320C3x generation is used and up to 60% if a TMS320C5x is used). Therefore, a complex induction motor control strategy including the presented current control method can be difficult to implement using a single common inexpensive DSP. The resistance estimation algorithms plus the flux and speed control procedures involve a large number of calculations and the total requirements can easily surpass the calculation power of such a chip.

Multiprocessor DSP-based control systems are therefore not a practical solution, and hardware implementation using ASIC or FPGA technologies proved to be an adequate alternative strategy for a fast and efficient control system capable of providing high performance. The high speed is achieved by adapting the hardware architecture to the algorithm specific data flow requirements. In addition, pipelining and parallel processing can be used on large scale to exploit all the opportunities offered by the specific calculation algorithms. The more parallelisms that can be found in one algorithm the faster the operation of its hardware implementation can be.

Calculation parallelism is best exploited by hardware implemented neural networks containing tens, or hundreds of elementary processing units co-operating to solve a particular problem. The neural approach is flexible as the neurone number can be increased or decreased, the calculation precision varying accordingly. The neural network size and architecture is determined based on the necessary calculation precision and the available hardware resources. The motor controller structure developed in this research work uses FPGA implemented neural networks alongside pipelined digital structures to carry out the computationally intensive task of controlling the stator current. This solution is fast, inexpensive and eliminates the timing problems related to the sequential operation of a DSP processor. Using this approach, the complexity of the control tasks performed is not significantly limited by the hardware operation speed. The only important limitation is given by the available amount of hardware resources.

4.2.5 Current Control Simulation

The drive system simulation approach combines the modelling flexibility of the VHDL software tools with the graphical capabilities of MATLAB. Thus, the simulation results have been generated in a numerical format using Workview Office 7.31 produced by Viewlogic, and then imported in MATLAB to generate the corresponding graphs.

A VHDL model of a three-phase induction motor has been created using the mathematical space vector model of the motor. A separate simplified model of the PWM inverter has been developed considering all power transistors as ideal switches. The two modules were combined with an abstract VHDL description of the adopted control strategy to generate a model of the entire drive system. This has been used to analyse the current control principles presented in section 4.2. The system operation has been simulated with different parameter values and the simulation results validated the current control principles previously presented.

As shown in the VHDL Code Fragment 4.1, the motor model is an entity having two input ports (the stator voltage and load torque) and two output ports (the stator currents and the rotor angular frequency). All data regarding the motor operation during the simulation is stored in an output ASCII file (motor.txt). The file contains numerical data in matrix format, which is compatible with MATLAB. Each line in the matrix contains the set of quantities that characterise the motor operation at a certain moment in time: currents, voltages, speed and torque.

```
-- Code Fragment 4.1
LIBRARY math;
USE math.complex_basic.all;
USE math.mathtyx.all;
USE std.textio.all;

ENTITY motor IS
  PORT(us: IN COMPLEX;
       Tload: IN REAL;
       ist: OUT COMPLEX;
       omegar: OUT REAL);
END motor;

ARCHITECTURE arch_motor OF motor IS
  CONSTANT Rs: REAL :=0.371;
  CONSTANT Rr: REAL :=0.415;
  CONSTANT Ls: REAL :=0.08705;
  CONSTANT Lr: REAL :=0.08763;
  CONSTANT Lm: REAL :=0.08433;
  CONSTANT Jr: REAL :=0.1;
  CONSTANT p: REAL :=2.0;
  CONSTANT deltat: TIME :=50 ns;
  CONSTANT dt: REAL := 5.0e-8;
```

```
  SIGNAL next_step: INTEGER :=1;
  FILE outf : TEXT IS OUT "c:\andrei\motor.txt";
BEGIN
  PROCESS(next_step)
    VARIABLE my_line: LINE;
    VARIABLE ist1,ist2,ir1,ir2,Fist1,Fist2,Fir1,Fir2,z:
             COMPLEX :=(0.0,0.0);
    VARIABLE T,omegar1,omegar2: REAL :=0.0;
    CONSTANT d_space: STRING :="  ";
  BEGIN
    IF next_step=1 THEN
      WRITE(my_line,us.re);
      WRITE (my_line,d_space);
      WRITE(my_line,us.im);
      WRITE (my_line,d_space);
      WRITE(my_line,ist1.re);
      WRITE (my_line,d_space);
      WRITE(my_line,ist1.im);
      WRITE (my_line,d_space);
      WRITE(my_line,ir1.re);
      WRITE (my_line,d_space);
      WRITE(my_line,ir1.im);
      WRITE (my_line,d_space);
      WRITE(my_line,T);
      WRITE (my_line,d_space);
      WRITE(my_line,omegar1);
      WRITELINE(outf,my_line);
  END IF;
  ist1:=ist2;
  ir1:=ir2;
  Fist1:=Fist2;
  Fir1:=Fir2;
  omegar1:=omegar2;
  Fist2:=Fist1+(us-Rs*ist1)*dt;
  Fir2:=Fir1+(j*omegar1*Fir1-Rr*ir1)*dt;
  ist2:=(Lr*Fist2-Lm*Fir2)/(Lr*Ls-Lm*Lm);
  ir2:=(Ls*Fir2-Lm*Fist2)/(Lr*Ls-Lm*Lm);
  z:=ist1*conj(ir1);
  T:=3.0/4.0*p*Lm*(z.im);
  omegar2:=omegar1+(T-Tload)/Jr;
  IF next_step<1000 THEN
    next_step<=next_step+1 AFTER deltat;
  ELSE
    next_step<=1 AFTER deltat;
  END IF;
  ist<=ist1;
    omegar<=omegar1;
  END PROCESS;
END arch_motor;

CONFIGURATION conf_motor OF motor IS
  FOR arch_motor
  END FOR;
END conf_motor;
```

The PWM inverter is modelled as a simple VHDL process. The sensitivity list of the process contains the 6-bit vector 'abcdef' containing the control signals to the six power transistors. The first three bits uniquely define the inverter output voltage. They

are used as the selection criterion in the CASE statement that generates the corresponding voltage space vector 'us'.

```
-- Code Fragment 4.2
process(abcdef(5 downto 3))
     constant U0: REAL :=
  begin
    case abcdef(5 downto 3) is
      when "100"=> us<=U0*(1.0,0.0);
      when "110"=> us<=U0*(0.5,0.866);
      when "010"=> us<=U0*(-0.5,0.866);
      when "011"=> us<=U0*(-1.0,0.0);
      when "001"=> us<=U0*(-0.5,-0.866);
      when "101"=> us<=U0*(0.5,-0.866);
      when others=> us<=(0.0,0.0);
    end case;
  end process;
```

The motor parameters used for the simulations presented in Fig. 4-8, Fig. 4-9 and Fig. 4-10 are given in the column (b) of Table 4.1. Thus, according to equation (4-8) the inductance in the equivalent circuit is L=5.9 mH. The parameters defining the operation of the current controller are as follows:

1. The module of the reference stator current space vector: 10 A
2. The reference frequency: 50 Hz.
3. The inductance updating step δL: 0.05 mH.
4. The PWM frequency: 20 kHz
5. The sampling frequency: 450 kHz.
6. $|\underline{V}_{ni}|_{crt}=\infty$

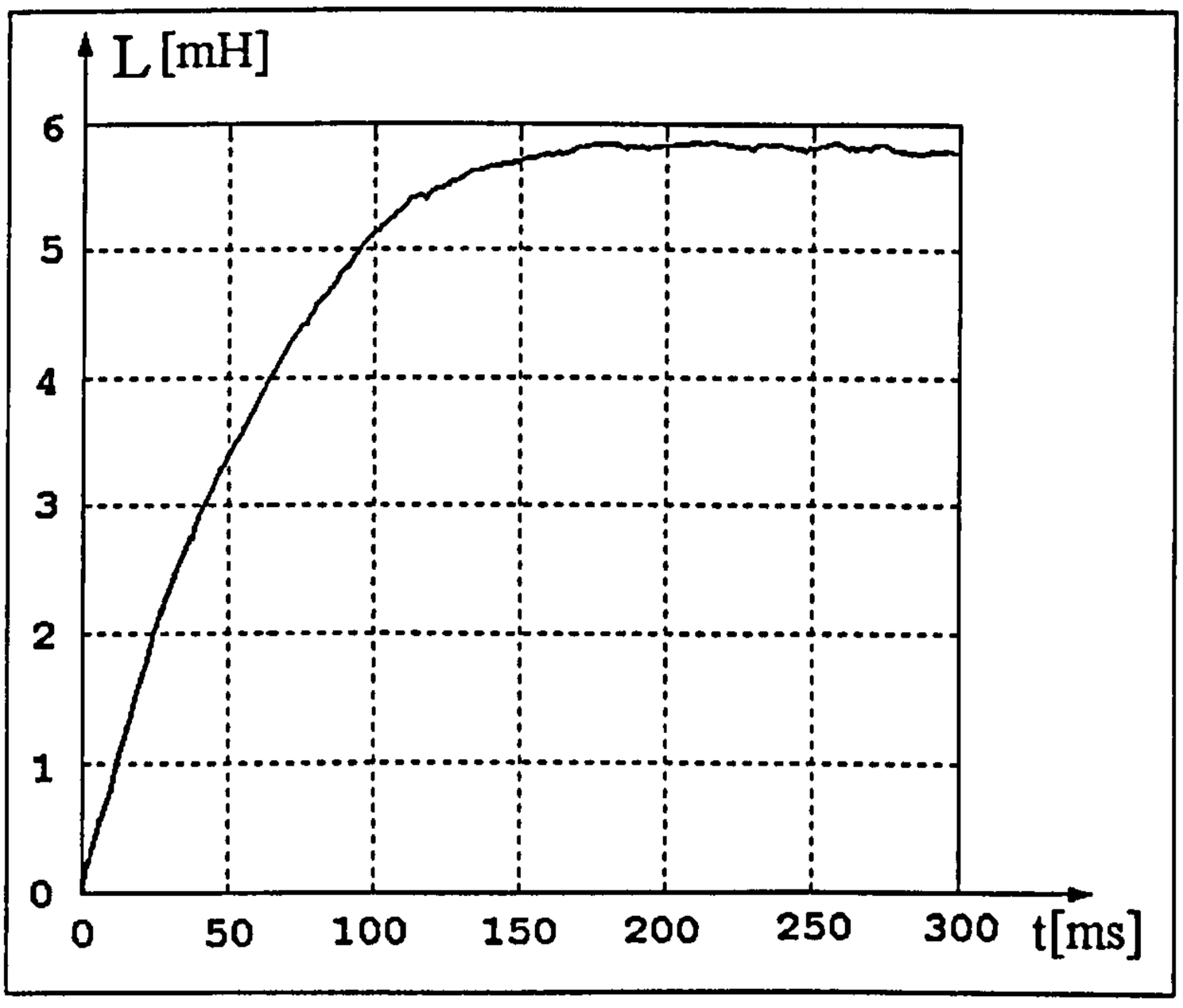

Fig. 4-8 - Inductance estimation values

As shown in Fig. 4-8, the correct inductance value is obtained in a short time interval (about 200 ms). The initial induction estimation error is very large causing very large errors in the calculation of the non-inductive voltage vector $\underline{V}_{ni}$. The increasing accuracy of the inductance estimate is reflected in the decreasing ripples of estimated $\underline{V}_{ni}$ shown in Fig. 4-9.

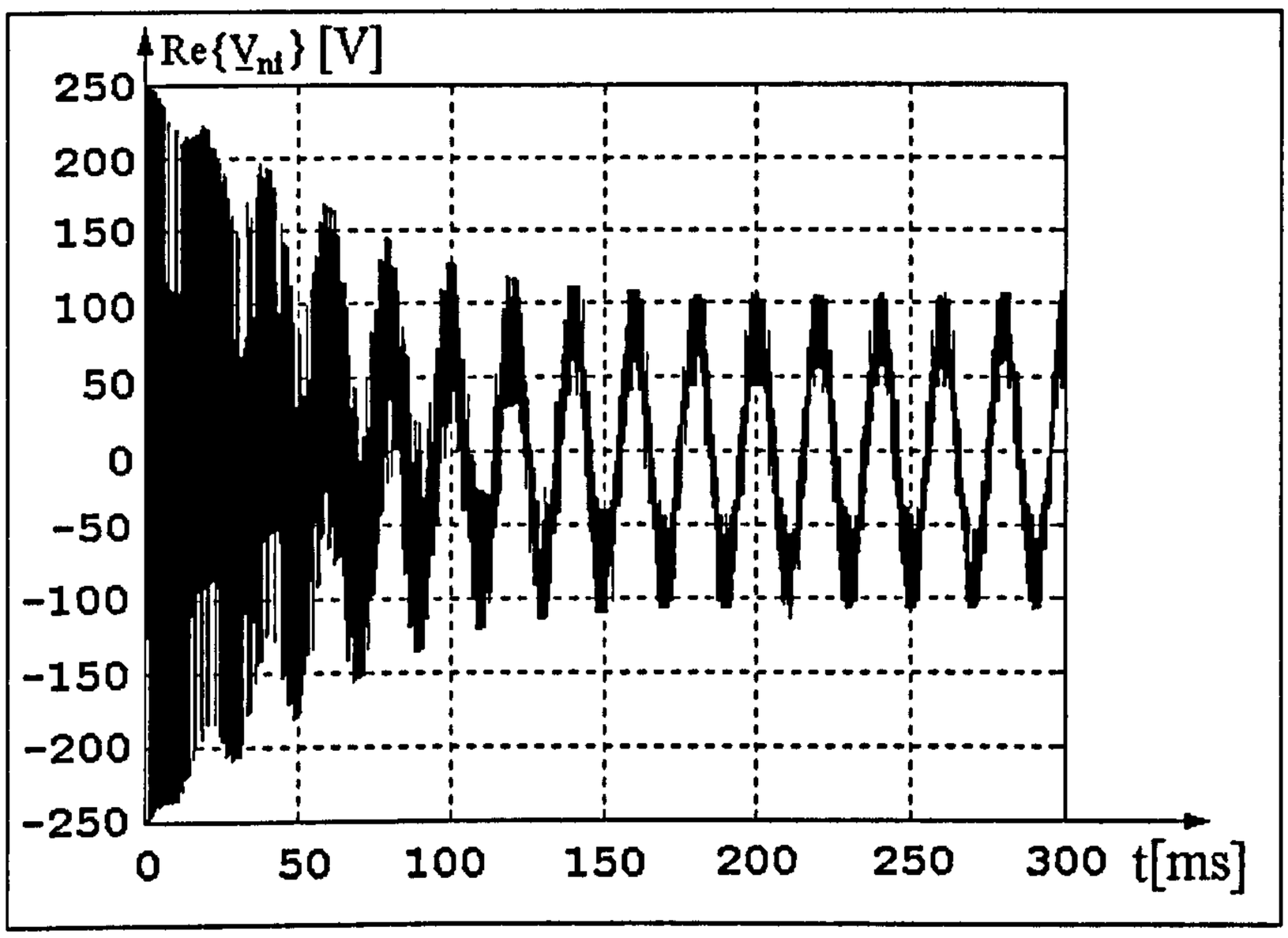

Fig. 4-9 - The non-inductive voltage estimation

Fig. 4-10 presents the trajectory in the complex plane of the actual current space vector across the stator winding. It demonstrates that the current ripples are maintained at low levels.

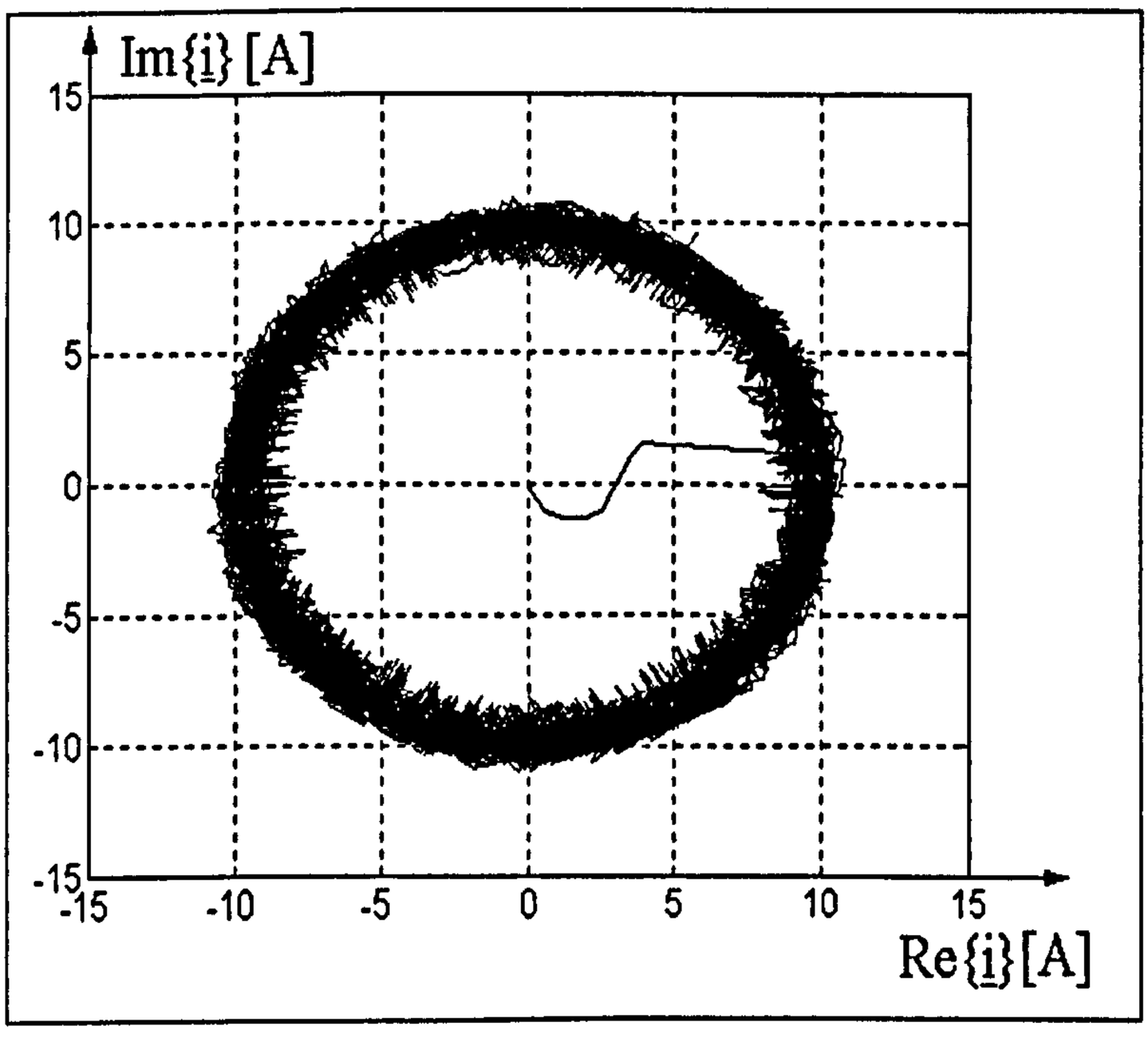

Fig. 4-10 - The load current space vector

Similar simulations have been performed for a current control algorithm version that always uses all the seven output voltages ($|\underline{V}_{ni}|_{crt}=0$). Similar results have been obtained but the inductance estimation process has been demonstrated to be much slower (about ten times slower). Moreover, the variations of the inductance estimation value in steady-state operation were twice the amplitude of the result of Fig. 4-8. On the other hand, the current ripple amplitude decreased to approximately 70% compared to Fig. 4-10 after the inductance estimation process was finished. Thus, although using the zero voltage decreases the current ripples, it also increases the inductance estimation errors, thereby affecting the accuracy of the information on the motor operation. Consequently, this control method version decreases the precision of any sensorless speed control strategy based on the equivalent R-L-e circuit.

4.3 THE NEW SENSORLESS INDUCTION MOTOR CONTROL STRATEGY

The common sensorless induction motor control strategies are derived from the sensor-based field oriented control methods that have been extended to include speed estimation algorithms. All field orientation methods require several transformations of the electromagnetic quantities from the stator reference frame into the flux reference frame, and back from the flux reference frame into stator reference frame. Thus, a general stator quantity '$\underline{A}$' is transformed from fixed stator co-ordinates into mobile flux co-ordinates, using equation (4-91). The inverse transformation is carried out according to (4-92) where $\theta(t)$ indicates the angle of the flux vector (the rotor flux, the stator flux or the airgap flux) and is a function of time. The complete control algorithms require several other mathematical calculations to be carried out: integrations, divisions, multiplications and square roots.

$$\begin{pmatrix} A_d^{\theta} \\ A_q^{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta(t) & \sin\theta(t) \\ -\sin\theta(t) & \cos\theta(t) \end{pmatrix} \cdot \begin{pmatrix} A_d^{s} \\ A_q^{s} \end{pmatrix} \tag{4-91}$$

$$\begin{pmatrix} A_d^{s} \\ A_q^{s} \end{pmatrix} = \begin{pmatrix} \cos\theta(t) & -\sin\theta(t) \\ \sin\theta(t) & \cos\theta(t) \end{pmatrix} \cdot \begin{pmatrix} A_d^{\theta} \\ A_q^{\theta} \end{pmatrix} \tag{4-92}$$

The sensorless speed control strategies are usually software implemented using DSPs or microcontrollers, because the hardware implementation is difficult to achieve due the large number of different mathematical operations to be implemented. Attempts have been made to combine the software and the hardware approaches [52]. This

strategy requires the implementation of a custom mathematical processor alongside specialised control modules, RAM memory, and ROM memory (to store the control programs). The specialised modules implement routine tasks such as the PWM signal generation or the A/D converter control, while the mathematical processor carries out all the complex mathematical calculations and updates the operation parameters for the specialised modules. Such an approach combines the flexibility of software implementation and the speed of the hardware implementation. On the other hand, it requires large integrated circuits and complex design procedures including simulation tools capable to check the operation of the mixed software-hardware control block.

An inexpensive, simple and compact hardware implementation requires the calculation to be minimised so that the software component of the control algorithm can be eliminated. Therefore, simpler speed estimation methods and simpler control strategies need to be devised.

The calculation complexity can be much reduced if quantities that are invariant at reference frame transformations are used so that matrix equations like (4-91) and (4-92) can be eliminated. The two types of quantities having such a property are the space vector modules and the phase shifts between the space vectors. A control algorithm that operates with such quantities is more suitable to using polar co-ordinates than the classical rectangular co-ordinates. Consequently, new speed estimation algorithms and speed control algorithms need to be developed and they need to be expressed as simple equations in polar co-ordinates. The novel speed control strategy proposed in this thesis can operate in conjunction with the current control method described in section 4.2, or it can be implemented independently. In both situations the speed control strategy is based on two principles:

1) The speed information is extracted by analysing the magnitude and/or the phase shift between two space vectors $\underline{A}$ and $\underline{B}$, chosen from the electromagnetic variables in the equivalent R-L-e circuit ($\underline{u}$, $\underline{V}_{ni}$, $\underline{e}$, $\underline{i}$).
2) The motor speed is controlled by modifying the amplitude and the angular speed of the stator current vector.

Using only quantities that are invariant at reference frame transformations (phase shifts and amplitudes) implies that the choice over the reference frame does not change the form of the speed estimation method or the form of the speed control algorithm. All reference frames are equivalent. However, the mathematical demonstration of the principles underlying the new control strategy is simpler in rectangular co-ordinates than

in polar co-ordinates. Furthermore, some reference frame orientations are preferable to others. For simplification reasons, the most appropriate approach is to define the reference frame orientation using the vector $\underline{A}$ involved in the motor speed estimation (the real axis of the rectangular co-ordinates is maintained parallel to this vector as illustrated in Fig. 4-11). In such a situation, the phase shift α_{BA} between $\underline{B}$ and $\underline{A}$ is calculated using only the rectangular components of vector $\underline{B}$:

$$\begin{cases} \mathrm{Re}\{\underline{B}_\theta\} = B \cdot \cos\alpha_{BA} \\ \mathrm{Im}\{\underline{B}_\theta\} = B \cdot \sin\alpha_{BA} \end{cases} \Rightarrow \alpha_{BA} = \arctan\left(\frac{\mathrm{Im}\{\underline{B}_\theta\}}{\mathrm{Re}\{\underline{B}_\theta\}}\right) \tag{4-93}$$

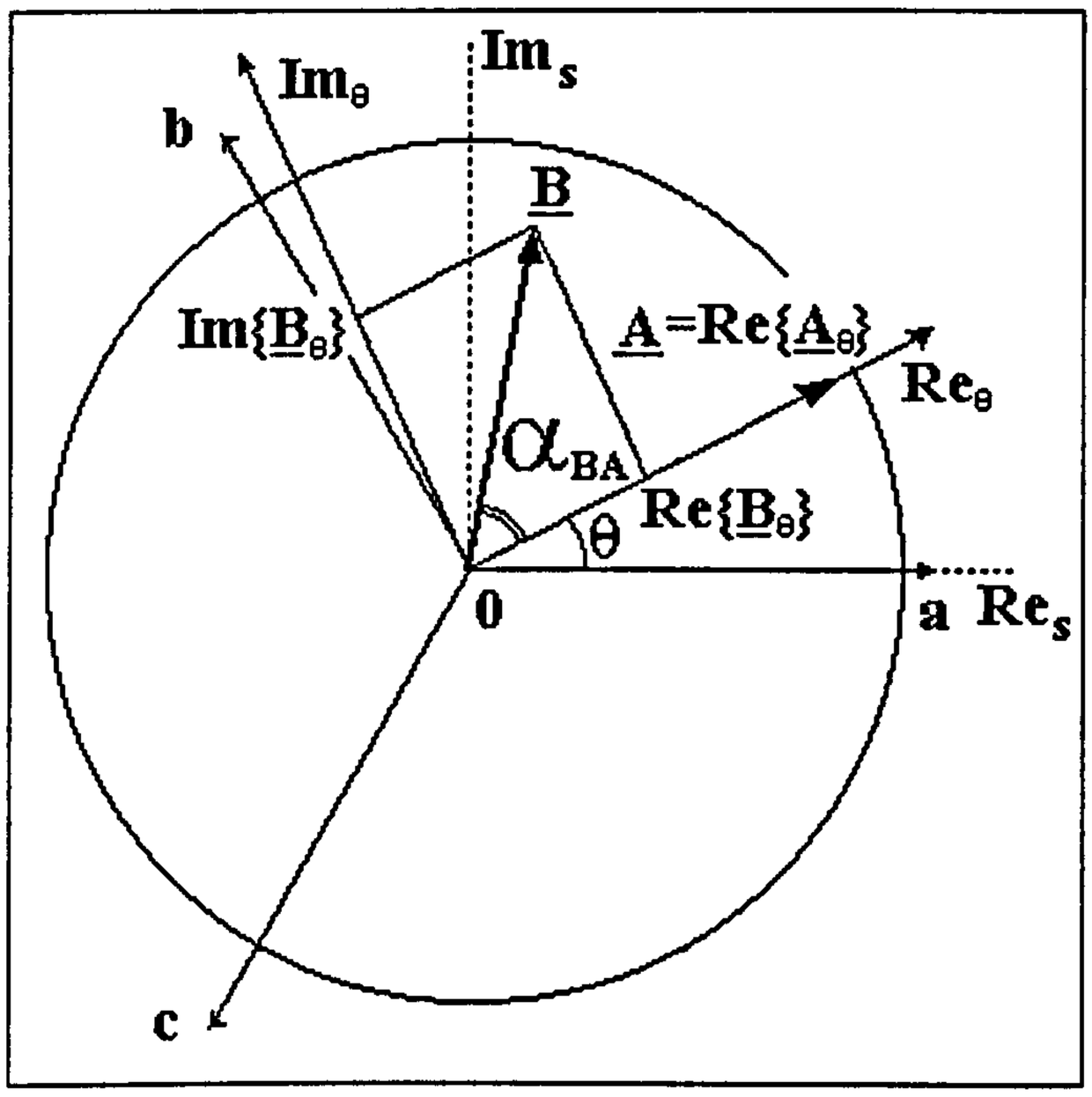

Fig. 4-11 – The reference frame oriented on vector $\underline{A}$

In the next chapters it is demonstrated that the calculation of the space vector arguments can be efficiently carried out by hardware implemented neural networks. The result is that the phase shift calculation is reduced to subtracting the space vector arguments, thereby avoiding trigonometric calculations and reducing the total chip area of the controller. Therefore, equations like (4-93) are used only for theoretical analysis but they do not have to be implemented directly into hardware.

4.3.1 Speed Estimation Algorithms

Several estimation methods can be developed depending on the vectors $\underline{A}$ and $\underline{B}$ that are chosen from the quantities available in the R-L-e circuit ($\underline{u}$, $\underline{V}_{ni}$, $\underline{e}$, $\underline{i}$). The methods have different degrees of accuracy and imply different calculation complexity levels. The most straightforward solution is operating with the voltage $\underline{u}$ and the current $\underline{i}$ because they are directly measurable quantities. The non-inductive voltage $\underline{V}_{ni}$ is a

good option if the speed estimation is performed by a controller that uses the current control strategy presented in section 4.2. The vector $\underline{V}_{ni}$ is calculated for the current control algorithm but the information can also be transferred to the speed estimator thereby decreasing the computation effort. The use of the internal voltage $\underline{e}$ requires the largest number of calculations because its value needs to be derived from the space vectors $\underline{u}$ or $\underline{V}_{ni}$.

The class of estimation methods defined by $\underline{A}=\underline{i}_s$ is analysed in a stator current oriented reference frame. Due to the stator current orientation, the imaginary part of the stator current vector is zero and the reference frame rotates with the angular speed ω_{es}, which corresponds to the synchronism speed in steady state operation. Throughout this section, the superscript 'syn' is attached to space vectors expressed in the synchronous stator current oriented reference frame. The conditions defining the chosen reference frame are mathematically described by

$$\begin{cases} \theta = \omega_{es} \cdot t \\ \underline{i}_s = I_s \end{cases} \tag{4-94}$$

where I_s designates a real quantity. The induction motor space vector model in a stator current oriented reference frame is expressed as

$$\begin{cases} \underline{u}_s^{syn} = R_s I_s + \dfrac{d\underline{\Psi}_s^{syn}}{dt} + j\omega_{es}\underline{\Psi}_s^{syn} \\ \underline{u}_r^{syn} = R_r \underline{i}_r^{syn} + \dfrac{d\underline{\Psi}_r^{syn}}{dt} + j(\omega_{es} - \omega_{er}) \cdot \underline{\Psi}_r^{syn} \\ \underline{\Psi}_s^{syn} = L_s I_s + L_m \underline{i}_r^{syn} \\ \underline{\Psi}_r^{syn} = L_r \underline{i}_r^{syn} + L_m I_s \end{cases} \tag{4-95}$$

If the last two equations in (4-95) are substituted in the first two, the result is

$$\begin{cases} \underline{u}_s^{syn} = R_s I_s + L_m \dfrac{d\underline{i}_r^{syn}}{dt} + L_s \dfrac{dI_s}{dt} + j\omega_{es}(L_s I_s + L_m \underline{i}_r^{syn}) \\ 0 = R_r \underline{i}_r^{syn} + L_r \dfrac{d\underline{i}_r^{syn}}{dt} + L_m \dfrac{dI_s}{dt} + j\omega_{slp}(L_m I_s + L_r \underline{i}_r^{syn}) \end{cases} \tag{4-96}$$

The quantity 'ω_{slp}' in (4-96) is the 'slip angular frequency' and represents the difference between the stator and the rotor electrical angular frequencies.

$$\omega_{slp} = \omega_{es} - \omega_{er} \tag{4-97}$$

The slip angular frequency is related to the motor slip 's' as follows

$$\omega_{slp} = s \cdot \omega_{es} \tag{4-98}$$

The calculation of ω_{slp} is a prerequisite for the rotor speed estimation. The relation between ω_{slp} and the rotor speed is described by

$$\omega_r = \frac{\omega_{es} - \omega_{slp}}{p} = \omega_s - \frac{\omega_{slp}}{p} \tag{4-99}$$

Therefore, the speed estimation methods can be reduced to methods of estimating the slip angular frequency. The possible slip estimation methods are first analysed in steady state operation and then in transient operation.

4.3.1.1 Steady-State Analysis

The general differential equation describing the evolution of the rotor current vector is

$$\frac{d\underline{i}_r^{syn}}{dt} + \left(\frac{R_r}{L_r} + j\omega_{slp}\right)\underline{i}_r^{syn} = -\frac{L_m}{L_r}\cdot\left(j\omega_{slp} I_s + \frac{dI_s}{dt}\right) \tag{4-100}$$

This equation is a consequence of the system (4-96). In steady-state operation, the motor currents are sinusoidal and have constant frequency, which entails circular trajectories for the corresponding space vectors. Thus, in the synchronous reference frame, the rotor current and the stator current space vectors are constant during the steady state. Therefore, the stator current derivative and the rotor current derivative are zero in (4-100). The rotor current space vector during steady-state is:

$$\underline{i}_r^{syn} = \frac{-j\omega_{slp} L_m I_s}{R_r + j\omega_{slp} L_r} \tag{4-101}$$

Based on equation (4-101), all the steady-state electromagnetic quantities of the R-L-e equivalent circuit can be calculated. First the internal voltage $\underline{e}$ is calculated and then the $\underline{V}_{ni}$ and $\underline{u}$ are derived from its expression. Thus, the general equation (4-7) becomes

$$\underline{e}^{syn} = \frac{L_m}{L_r}\left[-R_r \underline{i}_r^{syn} + j\omega_{er}\left(L_r \underline{i}_r^{syn} + L_m \underline{i}_s^{syn}\right)\right] = \frac{L_m}{L_r}\left[-R_r \underline{i}_r^{syn} + j\omega_{er}\underline{\Psi}_r^{syn}\right] \tag{4-102}$$

Substituting (4-101) in (4-102) yields

$$\underline{e}^{syn} = \frac{L_m}{L_r}\left[\frac{j\omega_{slp} L_m R_r I_s}{R_r + j\omega_{slp} L_r} + j\omega_{er}\left(\frac{-j\omega_{slp} L_m L_r I_s}{R_r + j\omega_{slp} L_r} + L_m I_s\right)\right] \tag{4-103}$$

The result can be incrementally transformed as follows:

$$\underline{e}^{syn} = \frac{L_m}{L_r}\left(\frac{j\omega_{slp}L_m R_r I_s}{R_r + j\omega_{slp}L_r} + j\omega_{er}\cdot\frac{R_r L_m I_s}{R_r + j\omega_{slp}L_r}\right) \tag{4-104}$$

$$\underline{e}^{syn} = \frac{L_m}{L_r}\cdot\frac{j(\omega_{slp}+\omega_{er})L_m R_r I_s}{R_r + j\omega_{slp}L_r} \tag{4-105}$$

$$\underline{e}^{syn} = \frac{L_m}{L_r}\cdot\frac{j\omega_{es}L_m R_r I_s}{R_r + j\omega_{slp}L_r} \tag{4-106}$$

$$\underline{e}^{syn} = \frac{L_m}{L_r}\cdot\frac{\omega_{es}\omega_{slp}L_m L_r R_r I_s}{R_r^2 + j\omega_{slp}^2 L_r^2} + \frac{L_m}{L_r}\cdot\frac{j\omega_{es}L_m R_r^2 I_s}{R_r^2 + j\omega_{slp}^2 L_r^2} \tag{4-107}$$

The calculation of the real and the imaginary part of the internal voltage in steady-state yields

$$\begin{cases} \mathrm{Re}\{\underline{e}^{syn}\} = \dfrac{\omega_{es}\omega_{slp}L_m L_r R_r I_s}{R_r^2 + \omega_{slp}^2 L_r^2}\cdot\dfrac{L_m}{L_r} \\ \mathrm{Im}\{\underline{e}^{syn}\} = \dfrac{\omega_{es}L_m R_r^2 I_s}{R_r^2 + \omega_{slp}^2 L_r^2}\cdot\dfrac{L_m}{L_r} \end{cases} \tag{4-108}$$

The non-inductive voltage space-vector $\underline{V}_{ni}$ depends on $\underline{e}$ according to general equation $\underline{V}_{ni}=R\underline{i}+\underline{e}$ which, in the synchronous reference frame, is transformed into $\underline{V}_{ni}^{syn} = R_s I_s + \underline{e}^{syn}$. Therefore, the real and imaginary parts of $\underline{V}_{ni}$ are:

$$\begin{cases} \mathrm{Re}\{\underline{V}_{ni}^{syn}\} = R_s I_s + \dfrac{\omega_{es}\omega_{slp}L_m L_r R_r I_s}{R_r^2 + \omega_{slp}^2 L_r^2}\cdot\dfrac{L_m}{L_r} \\ \mathrm{Im}\{\underline{V}_{ni}^{syn}\} = \dfrac{\omega_{es}L_m R_r^2 I_s}{R_r^2 + \omega_{slp}^2 L_r^2}\cdot\dfrac{L_m}{L_r} \end{cases} \tag{4-109}$$

To calculate the voltage vector $\underline{u}^{sys}$ the equation (4-1) is rewritten as

$$\underline{u}^{sys}\cdot e^{j\theta^{syn}} = R\cdot\underline{i}^{sys}\cdot e^{j\theta^{syn}} + L\cdot\frac{d}{dt}\left(\underline{i}^{sys}\cdot e^{j\theta^{syn}}\right) + \underline{e}^{sys}\cdot e^{j\theta^{syn}} \tag{4-110}$$

in the synchronous reference frame. It can be then further transformed into

$$\underline{u}_s^{syn} = RI_s + L\frac{dI_s}{dt} + j\omega_{es}L\cdot I_s + \underline{e}_s^{syn} \tag{4-111}$$

The stator current vector has constant module during steady-state operation so that (4-111) becomes

$$\underline{u}_s^{syn} = RI_s + j\omega_{es}L\cdot I_s + \underline{e}_s^{syn} = \underline{V}_1^{syn} + j\omega_{es}L\cdot I_s \tag{4-112}$$

Substituting (4-108) into (4-112), the voltage vector $\underline{u}_s^{syn}$ components can be expressed as:

$$\begin{cases} \mathrm{Re}\left\{\underline{u}_s^{syn}\right\} = R_s I_s + \dfrac{\omega_{es}\omega_{slp}L_m L_r R_r I_s}{R_r^2 + \omega_{slp}^2 L_r^2} \cdot \dfrac{L_m}{L_r} \\ \mathrm{Im}\left\{\underline{u}_s^{syn}\right\} = \dfrac{L_s L_r - L_m^2}{L_r} \cdot \omega_{es} I_s + \dfrac{\omega_{es} L_m R_r^2 I_s}{R_r^2 + \omega_{slp}^2 L_r^2} \cdot \dfrac{L_m}{L_r} \end{cases} \quad \textbf{(4-113)}$$

All the space vectors given in (4-108), (4-109) and (4-113) can be used to calculate the slip angular frequency. Both the vector amplitudes and the phase shift between these vectors and $\underline{i}_s$ contain information about ω_{slp}. Nonetheless, the alternative methods of slip estimation imply different degrees of precision, mathematical complexity and implementation difficulty. The next two sections analyse the alternatives from the hardware implementation perspective.

4.3.1.1.1 Slip Estimation Methods Based on Vector Amplitude

Using the real and imaginary components in (4-108), the absolute value of the internal voltage is

$$\left|\underline{e}^{syn}\right| = \mathrm{Re}^2\left\{\underline{e}^{syn}\right\} + \mathrm{Im}^2\left\{\underline{e}^{syn}\right\} = \mathrm{Re}^2\left\{\underline{e}^{s}\right\} + \mathrm{Im}^2\left\{\underline{e}^{s}\right\} = \frac{\omega_{es} L_m R_r I_s}{\sqrt{R_r^2 + \omega_{slp}^2 L_r^2}} \cdot \frac{L_m}{L_r} \quad \textbf{(4-114)}$$

From (4-114) ω_{slp} can be derived as follows:

$$\omega_{slp} = \sqrt{\frac{\omega_{es}^2 L_m^4 R_r^2 I_s^2}{\left|\underline{e}^{syn}\right|^2 \cdot L_r^4} - \frac{R_r^2}{L_r^2}} \quad \textbf{(4-115)}$$

The solution has a complicated non-linear mathematical form so it is not suitable for efficient hardware implementation. Linear equations can be implemented as interconnected adders and multipliers. The non-linear relationship (4-115) necessitates an additional divider plus a square root calculator. The dividers, and particularly the square root calculators, are bulky hardware structures. Positive results have been obtained in the last decade in developing efficient square root calculators ([23], [24], [25], [73]) and efficient dividers ([21], [22]). However, the structure of their implementation is still large so that any means of avoiding such hardware structures are preferable when compact hardware implementations are needed.

An alternative solution is the use of look-up tables to determine the correspondence between $\underline{e}$ and the motor slip. There are three main factors that

influence the value of the slip angular frequency: $|\underline{e}|$, ω_{es} and I_s, but the calculation results are also dependent on the rotor resistance, which is variable with the temperature. Fig. 4-12 illustrates the variation of $\underline{e}$ against ω_{slp} at rated voltage and rated frequency, calculated for the 11.1 kW induction motor. The figure demonstrates that the effect of rotor resistance variations is not negligible. Thus, the solution requires a four-dimensional look-up table which needs a large memory space. If, for instance, only 15 values were taken into account for each table dimension, then the table would have 50625 entries. Similar mathematical considerations apply to the implementation strategies that use space vectors $\underline{u}$ or $\underline{V}_{ni}$ instead of $\underline{e}$.

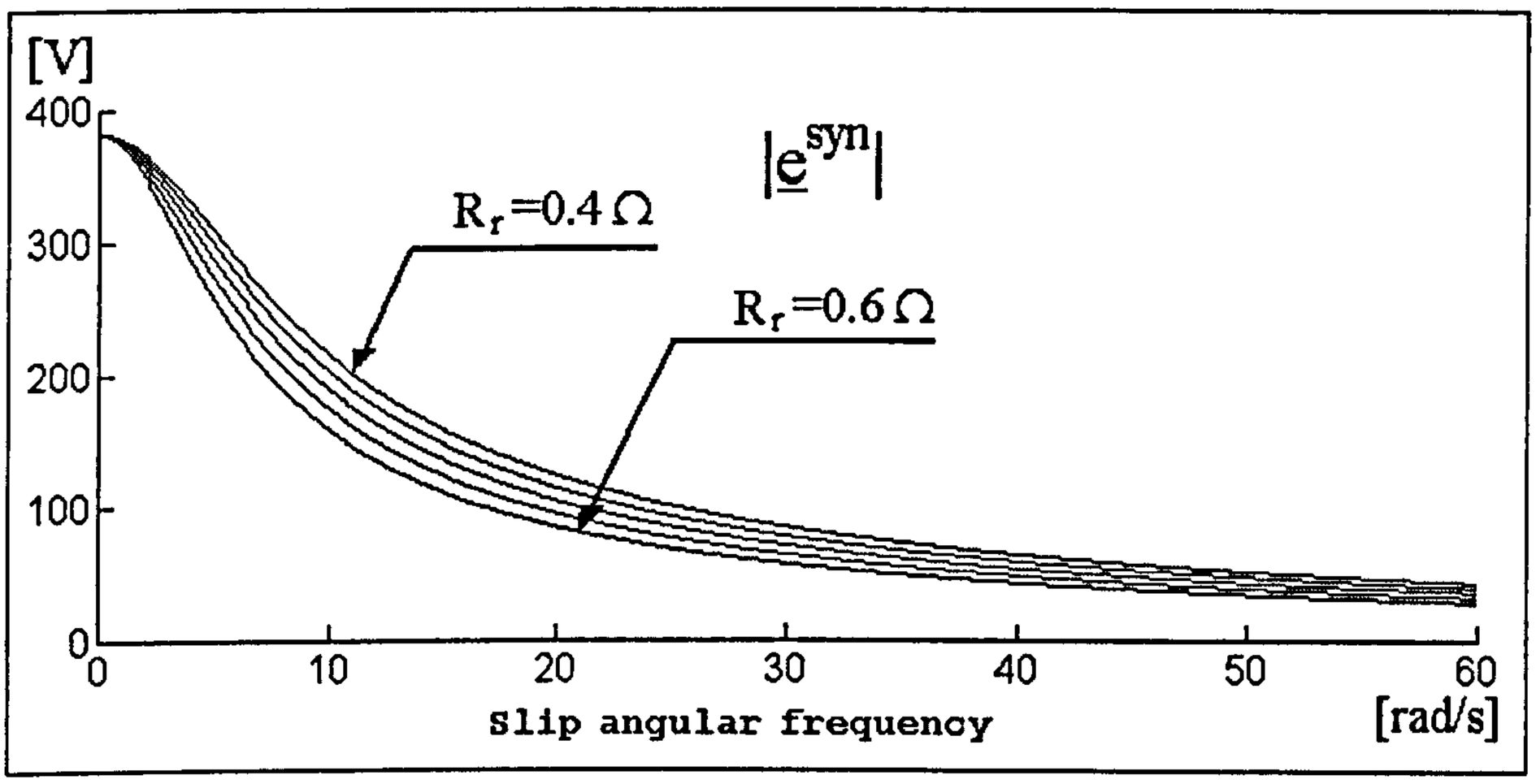

Fig. 4-12 - The variation of $\underline{e}$ against the slip angular frequency at rated stator frequency.

Small tables can be implemented in the same digital chip as the control circuitry thereby reducing the manufacturing cost. Large look-up tables can only be implemented as external EPROM chips, thereby complicating the PCB layout, increasing the power consumption and the size of the controller, decreasing its reliability, etc. In conclusion, the vector amplitude approach is not suitable for hardware implemented speed estimation.

4.3.1.1.2 Slip Estimation Methods Based on Phase Shift

The second possible approach to the motor speed estimation is to process the information contained in the value of the angle α between the stator current vector $\underline{i}$ and one of the other vectors: $\underline{e}$, $\underline{u}_s$ or $\underline{V}_{ni}$. The calculations yield the following results:

$$\tan^{-1}\alpha_{ui} = \tan^{-1}\left(\arg\left\{\underline{u}_s^{syn}\right\}\right) = \frac{R_s L_r\left(R_r^2 + \omega_{slp}^2 L_r^2\right) + \omega_{es}\omega_{slp} L_m^2 L_r R_r}{\left(L_r L_s - L_m^2\right)\cdot\left(R_r^2 + \omega_{slp}^2 L_r^2\right)\cdot\omega_{es} + L_m^2 R_r^2 \omega_{es}} \qquad (4\text{-}116)$$

$$\tan^{-1}\alpha_{V_{ni}} = \frac{\mathrm{Re}\{\underline{V}_{ni}^{syn}\}}{\mathrm{Im}\{\underline{V}_{ni}^{syn}\}} = \frac{\omega_{slp}L_r}{R_r} + \frac{L_r R_s\left(R_r^2 + \omega_{slp}^2 L_r^2\right)}{\omega_{es}L_m^2 R_r^2} \tag{4-117}$$

$$\tan^{-1}\alpha_{ei} = \frac{\mathrm{Re}\{\underline{e}^{syn}\}}{\mathrm{Im}\{\underline{e}^{syn}\}} = \frac{\omega_{slp}L_r}{R_r} \tag{4-118}$$

Relation (4-116) is very complicated and non-linear. It is not suitable to hardware implemented estimation of the slip angular speed. The result in (4-117) is simpler than (4-116), it contains one term that is proportional with ω_{slp} and another term proportional with the slip angular frequency squared. If the stator angular frequency is high ($\omega_{es}\cong 314$ rad/s), and the slip is small (it normally is during typical motor operation) then the last term in (4-117) can be neglected and an almost linear relationship between ω_{slp} and $\tan^{-1}(\alpha_{V_{ni}})$ is obtained:

$$\frac{\mathrm{Re}\{\underline{V}_{ni}^{syn}\}}{\mathrm{Im}\{\underline{V}_{ni}^{syn}\}} \approx \frac{\omega_{slp}L_r}{R_r} \Rightarrow \omega_{slp} \approx \frac{R_r}{L_r}\cdot\tan^{-1}\alpha_{V_{ni}} \tag{4-119}$$

Fig. 4-13 presents the numerical calculation results obtained for the 11.1 kW motor at a stator angular frequency ω_s=314 rad/s. It can be seen that the $\mathrm{Re}(\underline{V}_{ni})/\mathrm{Im}(\underline{V}_{ni})$ characteristics are almost straight lines at this rotor speed. Unfortunately, their curvature increases with the decrease of speed so that the relationship (4-119) is not valid for speeds much below the rated speed.

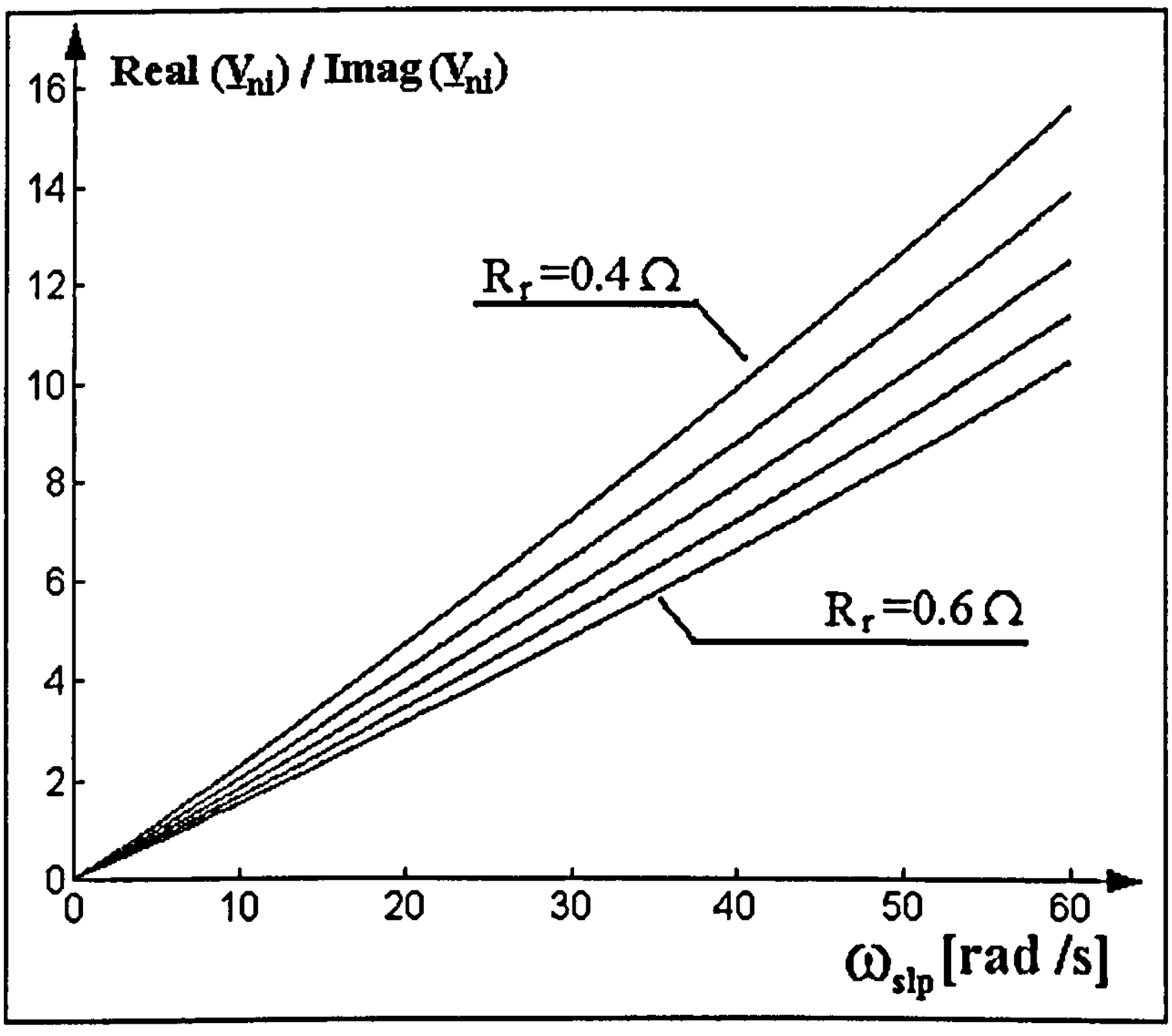

Fig. 4-13 Quasi-linear dependency between ω_{slp} and $\mathrm{Re}(\underline{V}_{ni})$ / $\mathrm{Im}(\underline{V}_{ni})$ ratio (ω_{es}=314 rad/s)

The approximate relationship (4-119) is appropriate for hardware implementation together with the current control strategy previously presented in section 4.2, because it provides the value of $\underline{V}_{ni}$. This version of the slip estimator is based on equation

$$\omega_{slp} = \tan^{-1}\left(\alpha_{V_{ni}i}\right)\cdot\frac{R_r}{L_r} = \tan^{-1}\left[\arg\left(\underline{V}_{ni}^{s}\right)-\arg\left(\underline{i}_s^{s}\right)\right]\cdot\frac{R_r}{L_r} \qquad \textbf{(4-120)}$$

However, such an estimation method can be used only in a limited number of practical applications, where the motor speed is variable but always high. The correct slip estimation at any speed can only be performed using the equation

$$\omega_{slp} = \tan^{-1}\left(\alpha_{ei}\right)\cdot\frac{R_r}{L_r} = \tan^{-1}\left[\arg\left(\underline{e}^{s}\right)-\arg\left(\underline{i}_s^{s}\right)\right]\cdot\frac{R_r}{L_r} \qquad \textbf{(4-121)}$$

that is derived from (4-118).

The internal voltage vector $\underline{e}$ can be calculated either as a function of $\underline{V}_{ni}$, $\underline{i}_s$ and R_s, or based on $\underline{u}_s$, R_s, ω_{es} and $\underline{i}_s$. The two alternatives are:

$$\begin{cases}\underline{e}_s^{s} = \underline{V}_1^{s} - R_s\underline{i}_s^{s}\\ \underline{e}_s^{s} = \underline{u}_s^{s} - R_s\underline{i}_s^{s} - \dfrac{L_sL_r - L_m^2}{L_r}\cdot\dfrac{d\underline{i}_s^{s}}{dt}\end{cases} \qquad \textbf{(4-122)}$$

The choice made depends upon the electrical quantities available. If the new current control method is used, then the $\underline{V}_{ni}$-based estimator is optimal in terms of hardware implementation. Otherwise, the $\underline{u}$-based estimator is the better option. Thus, the two alternative estimators operate based on the equivalent equations

$$\begin{cases}\omega_{slp} = \tan^{-1}\left[\arg\left(\underline{V}_1^{s} - R_s\underline{i}_s^{s}\right)-\arg\left(\underline{i}_s^{s}\right)\right]\cdot\dfrac{R_r}{L_r}\\ \omega_{slp} = \tan^{-1}\left[\arg\left(\underline{u}_s^{s} - R_s\underline{i}_s^{s} - \dfrac{L_sL_r - L_m^2}{L_r}\cdot\dfrac{d\underline{i}_s^{s}}{dt}\right)-\arg\left(\underline{i}_s^{s}\right)\right]\cdot\dfrac{R_r}{L_r}\end{cases} \qquad \textbf{(4-123)}$$

These two estimators are superior to the amplitude-based estimators presented in the previous section because the division and the square root calculation are eliminated. The $\underline{V}_{ni}$-based estimator is particularly simple, as it requires only two multiplications and one subtraction. The $\underline{u}$-based estimator is slightly more complicated because it requires one additional multiplication and two additional subtractions (the current derivative being approximated by the difference of the last two current samples). The vector argument calculations in the stator reference frame can be performed by a hardware implemented neural network. Alternatively, the function $\tan^{-1}$ can be easily

implemented as a small look-up table because it is a periodical and symmetrical function and only its values between 0^{o} and 90^{o} need to be stored. Such a small table is implementable into the same chip as the rest of the speed controller.

Due to their mathematical simplicity and hardware implementation advantages, the estimators based on the phase-shift information are adopted in this work as the optimal solution to the speed calculation problem.

4.3.1.2 The Transient Analysis of the Slip Estimation Process

The previous mathematical results are valid only in steady-state operation because they are based on the steady-state solution (4-101) of the differential equation (4-100). Further investigations are carried out in this section to analyse the magnitude of the slip estimation errors if the slip estimator uses these equations during the transient operation of the motor. Equation (4-100) is a first order linear differential equation. This class of equations has the general form

$$\frac{d\underline{i}_r^{syn}}{dt} + \underline{a}(t)\cdot\underline{i}_r^{syn} = \underline{b}(t) \tag{4-124}$$

where $\underline{a}(t)$ and $\underline{b}(t)$ are complex functions of time, while the solution is

$$\underline{i}_r(t) = e^{-\int \underline{a}(t)dt}\cdot\left[\int e^{\int \underline{a}(t)dt}\cdot\underline{b}(t)dt + K\right] \tag{4-125}$$

where K is a constant that is calculated based on the initial conditions. In the particular case of equation (4-100) the two time functions $\underline{a}(t)$ and $\underline{b}(t)$ are given by

$$\begin{cases}\underline{a}(t) = \dfrac{R_r}{L_r} + j\omega_{slp}(t)\\ \underline{b}(t) = -\dfrac{L_m}{L_r}\cdot\left(j\omega_{slp}(t)\cdot I_s + \dfrac{dI_s}{dt}\right)\end{cases} \tag{4-126}$$

so that the general solution is

$$\underline{i}_r(t) = e^{-\frac{R_r}{L_r}t - j\int\omega_{slp}(t)dt}\cdot\left[-\frac{L_m}{L_r}\cdot\int e^{\frac{R_r}{L_r}t + j\int\omega_{slp}(t)dt}\cdot\left[j\omega_{slp}(t)I_s + \frac{dI_s}{dt}\right]\cdot dt + K\right] \tag{4-127}$$

The general rotor current solution is very complicated. To simplify the calculations, a rule of thumb can be used: given the same initial state, the average slip estimation errors are smaller during the slow transients, than during the fast transients. The rule is justified by two facts:

1. The induction motor is a stable system (during any transient its parameters tend to change to stable values and after the transient ends they remain unchanged). Any set of inputs corresponds to a set of stable motor quantities.
2. The variation in speed of any mechanical or electromagnetic quantity is limited by finite time constants, that is they cannot change instantaneously.

All the motor characteristic quantities closely follow the corresponding steady-state values during a slow transient. The difference is larger during fast transients because the motor quantities lag behind the steady-state values due to the speed limitations imposed by the time constants. The accuracy of the previously analysed slip estimators depends on how close the motor quantities are to the steady-state values. Thus, fast transients will tend to generate larger estimation errors than slow transients. In conclusion, to assess the maximal magnitude of the slip estimation errors it is necessary to analyse the fastest motor transients.

If the PWM inverter supplying the motor is controlled by a current controller as the one described in section 4.2, then the steady state operation is best described by two basic parameters: the stator current angular frequency ω_{es} and the stator current vector amplitude I_s. Thus, the motor transients can be divided in three categories:

1) Transients caused by the alteration of ω_{es}.
2) Transients generated by the variation of the stator current amplitude I_s.
3) Combined transients created by a simultaneous variation of the two quantities.

4.3.1.2.1 The Effects of Altering the Stator Current Frequency

The fastest transient between two stator angular frequencies is caused by a step change of its value. In practical situations, the rotor inertia generates large mechanical time constants (larger than the electromagnetic time constants of the motor). Therefore, the rotor speed change is small during the relatively short transient caused by a step change of the stator angular frequency. To simplify the calculations, the rotor speed is considered constant during the transient. If the rotor speed is constant then, according to (4-97), a step change of ω_{es} implies a step change of ω_{slp}. The transient response for the rotor current is the solution (4-129) of the differential equation (4-128) which is a particular case of (4-124) because the functions $\underline{a}(t)$ and $\underline{b}(t)$ are in this case complex constants.

$$\frac{d\underline{i}_r^{syn}}{dt} + \left(\frac{R_r}{L_r} + j\omega_{slp}\right)\underline{i}_r^{syn} = -j\frac{L_m}{L_r}\cdot\omega_{slp} I_s \qquad (4\text{-}128)$$

$$\underline{i}_r^{syn} = Ke^{-(\frac{R_r}{L_r}+j\omega_{slp}^{[2]})t} - \frac{j\omega_{slp}^{[2]}L_m I_s}{R_r + j\omega_{slp}^{[2]}L_r} \tag{4-129}$$

The symbol $\omega^{[1]}$ denotes the initial slip angular frequency, $\omega^{[2]}$ is the final slip angular frequency, and K is a constant whose value has to be calculated taking into account the initial conditions. The initial rotor current is:

$$\underline{i}_r(0) = \underline{i}_r^{syn}(\omega_{slp}^{[1]}) = \frac{-j\omega_{slp}^{[1]}L_m I_s}{R_r + j\omega_{slp}L_r} \tag{4-130}$$

Consequently, the expression describing the rotor current dynamics is

$$\underline{i}_r^{syn} = \left(\frac{j\omega_{slp}^{[2]}L_m I_s}{R_r + j\omega_{slp}^{[2]}L_r} - \frac{j\omega_{slp}^{[1]}L_m I_s}{R_r + j\omega_{slp}^{[1]}L_r}\right)e^{-(\frac{R_r}{L_r}+j\omega_{slp}^{[2]})t} - \frac{j\omega_{slp}^{[2]}L_m I_s}{R_r + j\omega_{slp}^{[2]}L_r} \tag{4-131}$$

which can be transformed into

$$\underline{i}_r^{syn} = \underline{i}_r^{syn[2]} - \left(\underline{i}_r^{syn[2]} - \underline{i}_r^{syn[1]}\right)\cdot e^{-\left(\frac{R_r}{L_r}+j\omega_{slp}^{[2]}\right)t} = \underline{i}_r^{syn[2]} - \Delta\underline{i}_r^{syn}\cdot e^{-\left(\frac{R_r}{L_r}+j\omega_{slp}^{[2]}\right)t} \tag{4-132}$$

To calculate the evolution of vector $\underline{e}$ during the transient, relation (4-132) is substituted into (4-102). The result is illustrated by equations

$$\underline{e}^{syn} = \frac{L_m}{L_r}\left[\left(-R_r + j\omega_{cr}L_r\right)\cdot\left(\underline{i}_r^{syn[2]} - \left(\underline{i}_r^{syn[2]} - \underline{i}_r^{syn[1]}\right)\cdot e^{-\left(\frac{R_r}{L_r}+j\omega_{slp}^{[2]}\right)t}\right) + j\omega_{cr}L_m I_s\right] \tag{4-133}$$

and

$$\underline{e}^{syn} = \underline{e}^{syn[2]} - \left(\underline{e}^{syn[2]} - \underline{e}^{syn[1]}\right)\cdot e^{-\left(\frac{R_r}{L_r}+j\omega_{slp}^{[2]}\right)t} = \underline{e}^{syn[2]} - \Delta\underline{e}^{syn}\cdot e^{-\left(\frac{R_r}{L_r}+j\omega_{slp}^{[2]}\right)t} \tag{4-134}$$

where $\underline{e}^{syn[1]}$ is the initial internal voltage and $\underline{e}^{syn[2]}$ is the final internal voltage. Therefore, the trajectory of the internal voltage is a spiral of the type illustrated by Fig. 4-14.

The real part and the imaginary part of the internal voltage are:

$$\begin{cases} \mathrm{Re}\{\underline{e}^{syn}\} = \mathrm{Re}\{\underline{e}^{syn[2]}\} - \left(\mathrm{Re}\{\Delta\underline{e}^{syn}\}\cos\omega_{slp}^{[2]}t + \mathrm{Im}\{\Delta\underline{e}^{syn}\}\sin\omega_{slp}^{[2]}t\right)e^{-\frac{R_r}{L_r}t} \\ \mathrm{Im}\{\underline{e}^{syn}\} = \mathrm{Im}\{\underline{e}^{syn[2]}\} - \left(\mathrm{Im}\{\Delta\underline{e}^{syn}\}\cos\omega_{slp}^{[2]}t - \mathrm{Re}\{\Delta\underline{e}^{syn}\}\sin\omega_{slp}^{[2]}t\right)e^{-\frac{R_r}{L_r}t} \end{cases} \tag{4-135}$$

Thus, the stator speed step change produces electromagnetic oscillations in the rotor that can be sensed back in the stator due to the corresponding oscillations of induced electromotive force emf (an electromagnetic echo). All the previously analysed

slip estimation methods based on the R-L-e model produce erroneous results during the transient operation of the motor because they are affected by electromagnetic echoes.

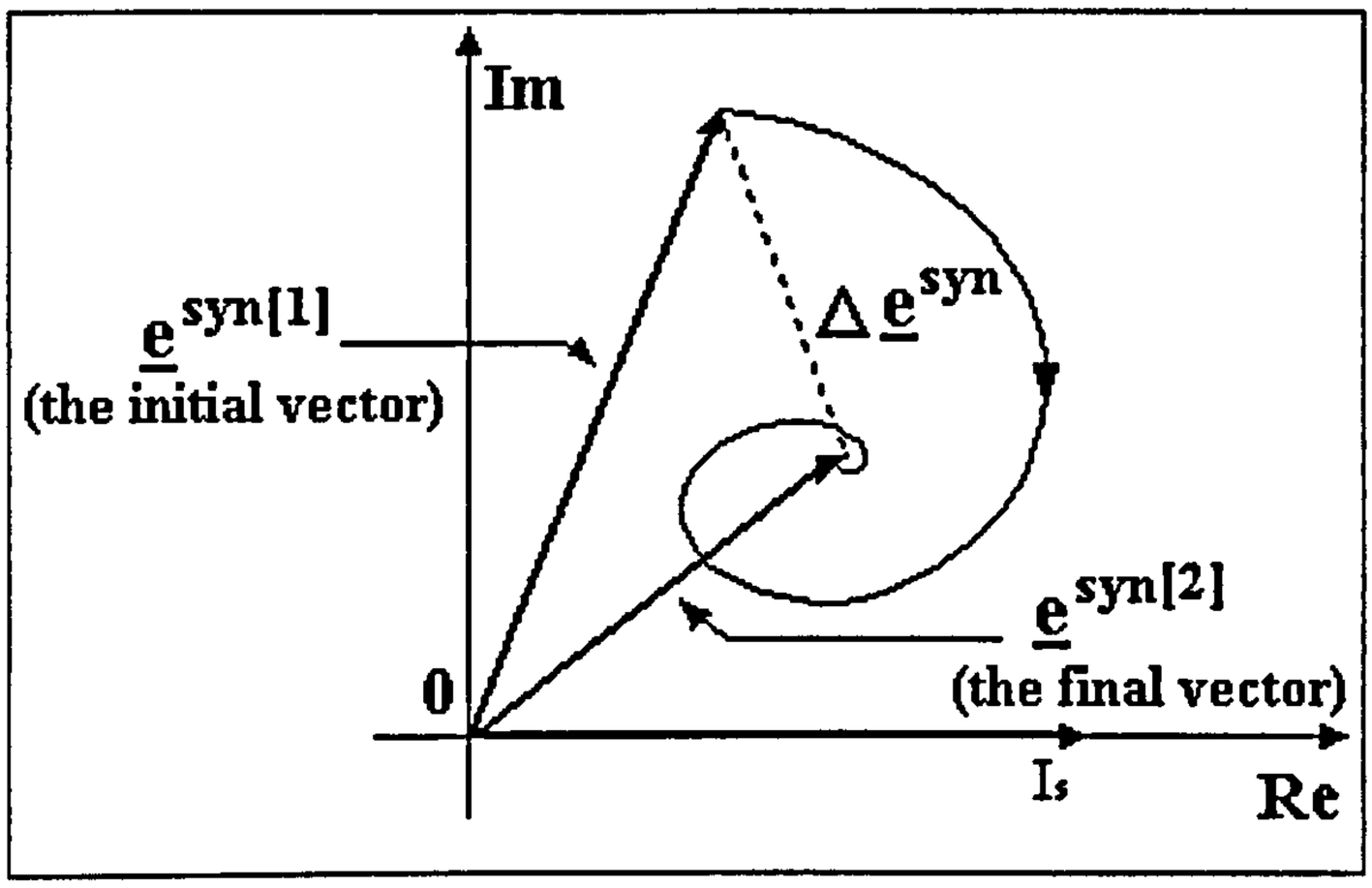

Fig. 4-14 - The evolution of the rotor current space vector in the synchronous frame of co-ordinates

The two hardware implementable estimators discussed (the $\underline{u}$-based estimator and the $\underline{V}_{ni}$-based estimator) generate transient estimates according to equation

$$\hat{\omega}_{slp} = \frac{R_r}{L_r} \cdot \frac{Re\left\{\underline{e}^{syn[2]}\right\} - \left(Re\left\{\Delta\underline{e}^{syn}\right\}\cos\omega_{slp}^{[2]}t + Im\left\{\Delta\underline{e}^{syn}\right\}\sin\omega_{slp}^{[2]}t\right)e^{\frac{R_r}{L_r}t}}{Im\left\{\underline{e}^{syn[2]}\right\} - \left(Im\left\{\Delta\underline{e}^{syn}\right\}\cos\omega_{slp}^{[2]}t - Re\left\{\Delta\underline{e}^{syn}\right\}\sin\omega_{slp}^{[2]}t\right)e^{\frac{R_r}{L_r}t}} \qquad (4\text{-}136)$$

The slip angular frequency estimates undergo damped oscillations at a frequency equal to the slip frequency. This causes oscillatory estimation errors described by

$$err_\omega(t) = \frac{R_r}{L_r} \cdot \frac{Re\left\{\underline{e}^{syn[2]}\right\} - \left(Re\left\{\Delta\underline{e}^{syn}\right\}\cos\omega_{slp}^{[2]}t + Im\left\{\Delta\underline{e}^{syn}\right\}\sin\omega_{slp}^{[2]}t\right)e^{\frac{R_r}{L_r}t}}{Im\left\{\underline{e}^{syn[2]}\right\} - \left(Im\left\{\Delta\underline{e}^{syn}\right\}\cos\omega_{slp}^{[2]}t - Re\left\{\Delta\underline{e}^{syn}\right\}\sin\omega_{slp}^{[2]}t\right)e^{\frac{R_r}{L_r}t}} - \omega_{slp}^{[2]} \qquad (4\text{-}137)$$

Equation (4-108) demonstrate that $Re\{\underline{e}^{syn}\}$ and $Im\{\underline{e}^{syn}\}$ are proportional to I_s and to ω_{es} so that the effects of these two factors cancel out in equations (4-136) and (4-137) because both the numerator and the denominator are proportional to I_s and to ω_{es}. Thus, the estimated slip and the estimation errors depend only on the initial slip angular frequency $\omega_{slp}^{[1]}$ and on the final slip angular frequency $\omega_{slp}^{[2]}$. Alternatively, the errors can be defined as a function of $\omega_{slp}^{[1]}$ and the slip angular frequency change $\Delta\omega_{slp} = \omega_{slp}^{[2]} - \omega_{slp}^{[1]}$.

On the other hand the magnitude of the estimation errors can be calculated as a function of the internal voltage variation $|\Delta\underline{e}^{syn}|$ which is the distance in the complex plane between the initial internal voltage vector $e^{syn[1]}$ and the final internal load vector

$e^{syn[2]}$ (equation (4-134)). In other words, $|\Delta \underline{e}^{syn}|$ is the difference of two steady-state vectors: the final one and the initial one.

Elementary algebraic calculations based on (4-108) prove that the locus of $\underline{e}^{syn}$ in steady state operation is defined by

$$\left[\mathrm{Im}\left(\underline{e}^{syn}\right)-\frac{\omega_{es}L_m^2 I_s}{2L_r}\right]^2+\mathrm{Re}\left(\underline{e}^{syn}\right)^2=\frac{\omega_{es}^2 L_m^4 I_s^2}{4L_r^2} \tag{4-138}$$

This equation can be transformed into

$$\begin{cases}\left(\mathrm{Re}\{\underline{e}^{syn}\}\right)^2+\left(\mathrm{Im}\{\underline{e}^{syn}\}-R\right)^2=R^2\\ R=\dfrac{L_m^2}{2L_r}\cdot|\omega_{es}|\cdot I_s\end{cases} \tag{4-139}$$

proving that the locus is a set of circles whose position and radius depend on the stator current amplitude I_s and the stator angular frequency ω_{es}. According to (4-139), all the circles are tangent to the real axis in the point of co-ordinates (0,0). The sign of $\mathrm{Im}\{\underline{e}^{syn}\}$ depends on $\mathrm{sign}\{\omega_{es}\}$, so that there are circles both above and below the real axis. The upper circles correspond to positive stator angular frequency, while the lower circles correspond to negative stator angular frequency.

Fig. 4-15 presents the internal voltage locus in steady state for the 11.1 kW motor at positive stator angular frequency and positive slip angular frequency. Any steady state internal voltage $\underline{e}^{syn}$ is situated at the intersection between a radius (corresponding to ω_{slp}) and a circle (corresponding to $I_s \times |\omega_{es}|$). When the slip angular frequency changes, the vector $\underline{e}^{syn}$ undergoes a transient that ends in a point situated at another intersection of a circle with a radius. If the transient is due to a change of the rotor speed then the final circle is identical to the initial circle, as the stator angular frequency is unchanged. If the transient is generated by a change of the stator angular frequency then both the new radius and the new circle are different as compared to the initial ones. Fig. 4-15 demonstrates that, at the same initial ω_{slp}, the internal voltage variation $|\Delta \underline{e}|$ and therefore the slip estimation errors increase with the magnitude of the step change $\Delta\omega_{slp}$.

Fig. 4-16 presents MATLAB simulation results obtained for the 11.1 kW induction motor. The stator angular frequency undergoes step changes described by (4-140). The step change Ω_s ranges between 3 and 10 rad/s and the rotor inertia has been considered infinite so that the rotor speed variation is null during the entire transient.

$$\omega_s(t)=\begin{cases}0 \text{ when } t<0\\ \Omega_s \text{ when } t\geq 0\end{cases} \tag{4-140}$$

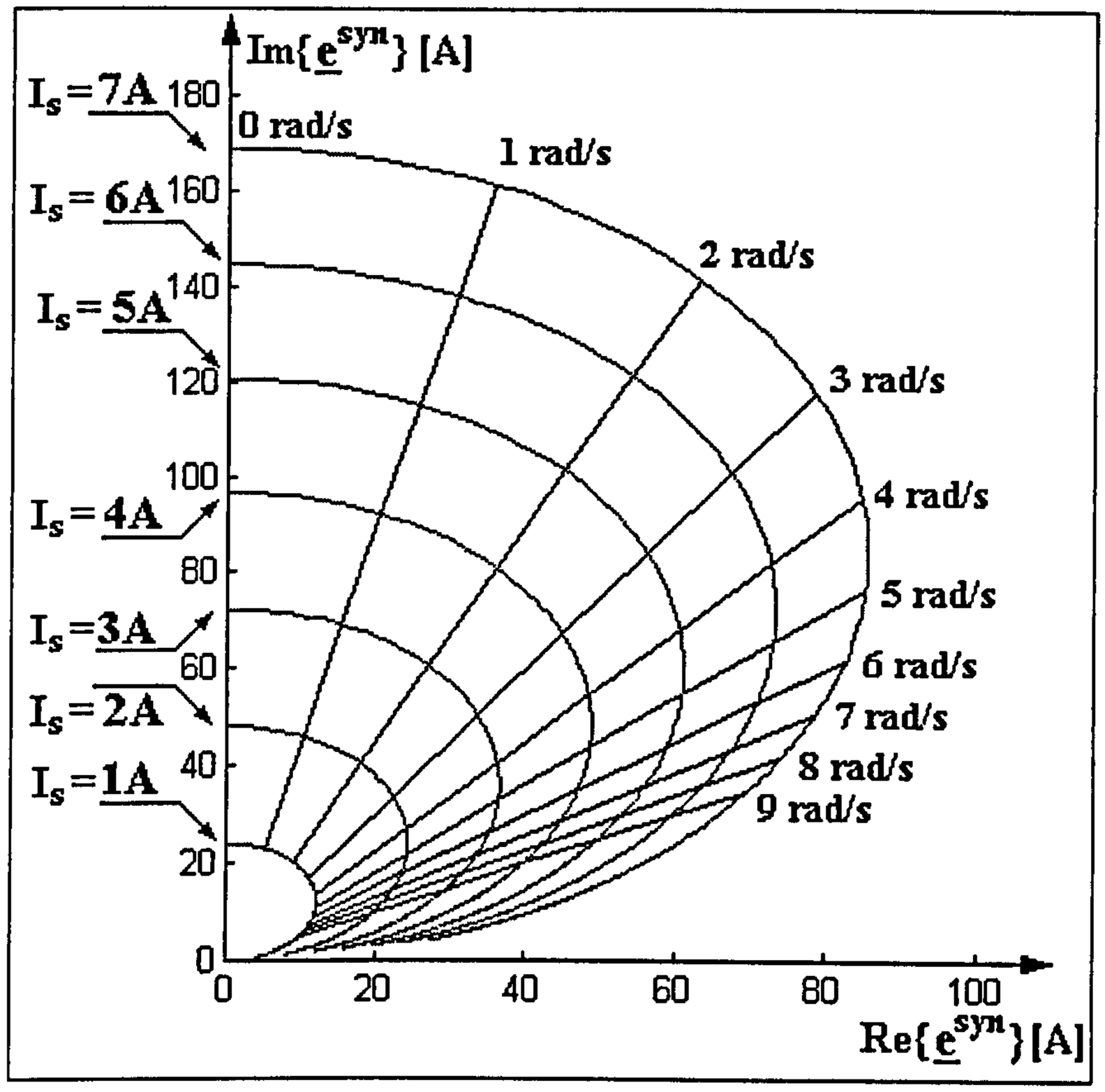

Fig. 4-15 – Internal voltage locus during steady-state (ω_{es}=314 rad/s)

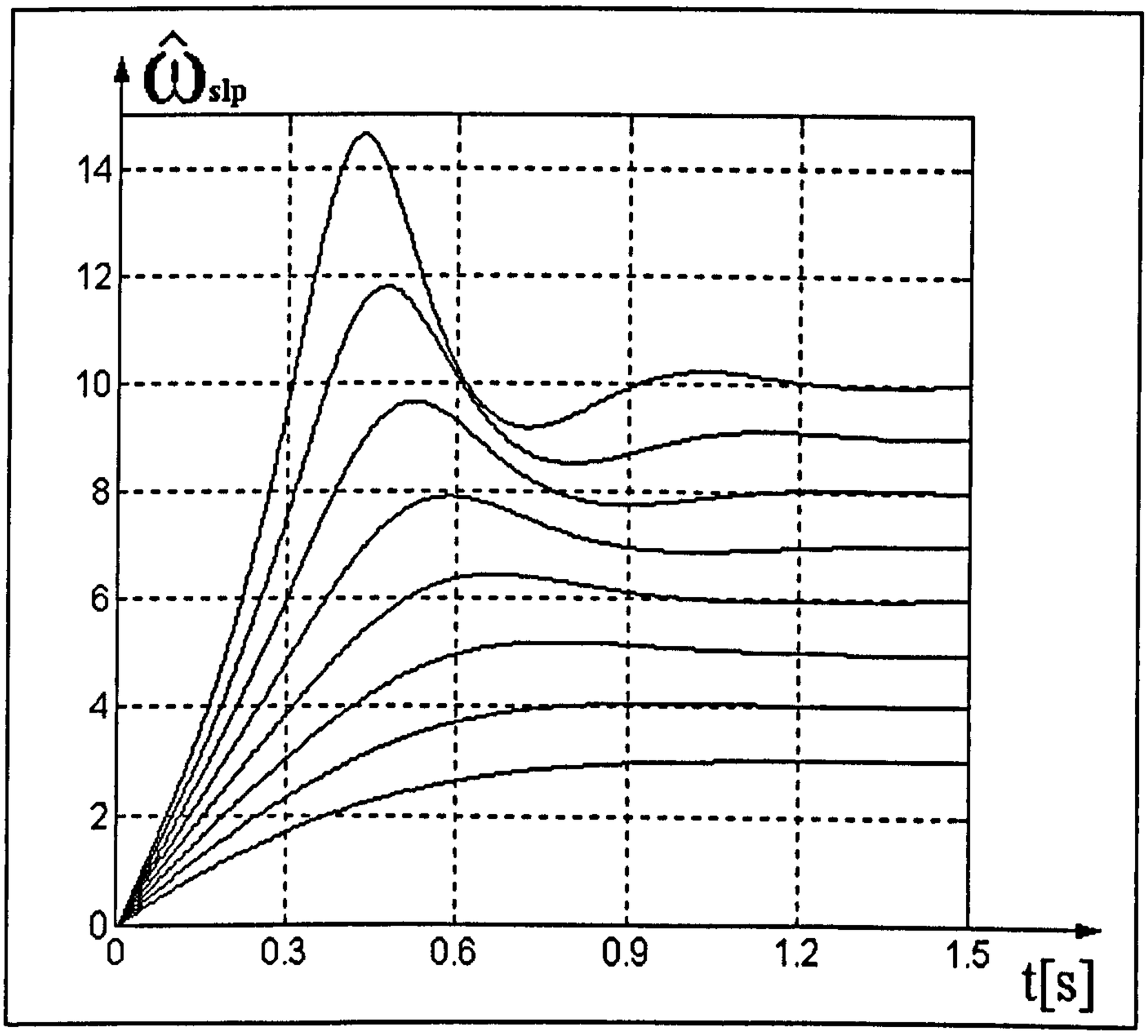

Fig. 4-16 - The transient response of the slip angular frequency estimator (slip modification)

At low slip frequency (smaller than 5 rad/s in this case), $|\Delta e^{syn}|$ is small and the oscillations are damped before the end of the first period so that the oscillatory character of the estimator response is obscured. It becomes apparent at higher slip angular frequencies when both $|\Delta e^{syn}|$ and the oscillation frequency are higher so that the oscillating errors are damped only after several periods.

The initial slip angular frequency ω_{slp} before the transient has an important influence as well. Thus, the error calculated in (4-137) can attain very large values if the denominator approaches zero, while the numerator has large values. This happens when the amplitude of the oscillatory component of the denominator in (4-137) is almost equal to $Im\{\underline{e}^{syn[2]}\}$. Therefore, the magnitude of the estimation errors depends on $Im\{\underline{e}^{syn[2]}\}$, which in turn depends on the initial ω_{slp} and on $\Delta\omega_{slp}$. As shown in Fig. 4-15, the value of $Im\{\underline{e}^{syn[2]}\}$ is small when the initial ω_{slp} is large, thereby amplifying the estimation errors. Moreover, if the initial ω_{slp} and/or $\Delta\omega_{slp}$ are sufficiently high, then the spiral trajectory crosses the real axis, situation illustrated in Fig. 4-17, and the denominator in (4-137) becomes zero. In this case, the errors are infinitely large. In practice, these catastrophic situations are generated by extreme transients: fast and large changes of the stator angular frequency associated with motor reversals or extremely fast speed changes. In these situations the corresponding spiral trajectories have a large diameter and often cross the real axis several times.

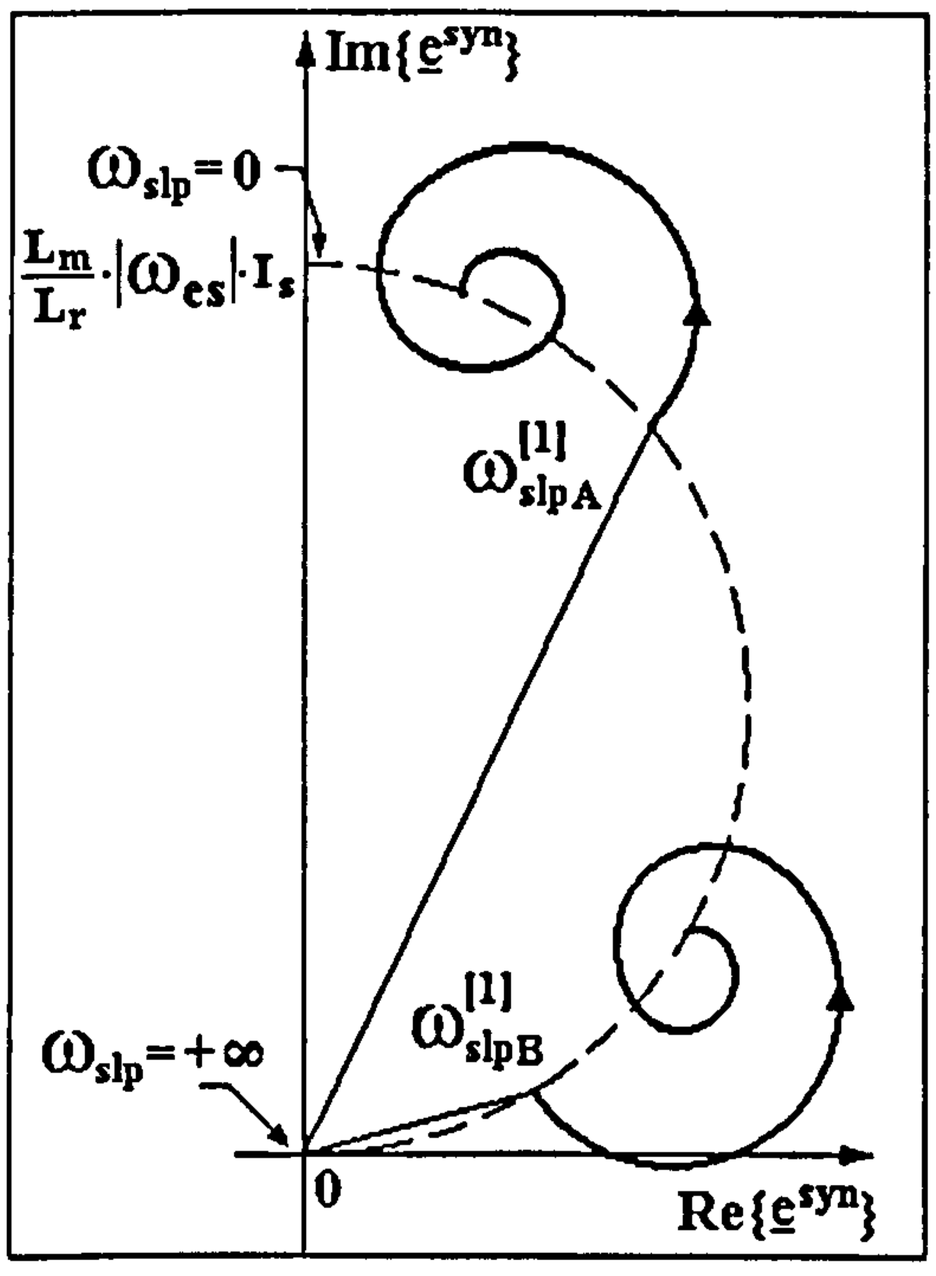

Fig. 4-17 – Internal voltage transients at two slip angular frequency ($\omega_{slpB}^{[1]} >> \omega_{slpA}^{[1]}$)

Conversely, the estimation errors are small when $\mathrm{Im}\{\underline{e}^{syn[2]}\}$ has a large value, which is equivalent to a small ω_{slp}. Thus, to minimise the estimation errors, it is important that the internal load vector does not approach the real axis of the synchronous reference system. This is equivalent to maintaining ω_{slp} to low values.

All the previous theoretical results have been obtained for infinite rotor inertia. If the rotor inertia is small then the slip angular frequency is not constant during the transient. The slip is a maximum at the beginning of the transient and then it decreases, as the rotor speed tends to follow the stator changes. This means that the vector $\underline{e}^{syn}$ does not rotate around a fixed point but around a moving point. This point shifts towards the initial position of vector $\underline{e}$, the effect being an accelerated decrease of the spiral radius (hence a smaller spiral) and a shorter transient. A smaller spiral trajectory in the complex plane implies smaller slip estimation errors than in the case of the infinite inertia motors. Therefore, the results analysed so far represent the maximal estimation errors that can be obtained during the transients when ω_{slp} is variable but I_s is constant.

4.3.1.2.2 The Effects of Altering the Stator Current Amplitude

Equation (4-100) cannot be solved for a step change of the stator current amplitude because in this situation the stator current derivative would be infinite. The step change however, can be considered a limit case of a very fast linear increase followed by a constant value (4-141). The two operation conditions can be studied separately integrating the corresponding differential equations and obtaining two time functions: $\underline{i}_{r1}(t)$ and $\underline{i}_{r2}(t)$. The concatenation of the two solutions describes the complete behaviour of the system:

$$I_s(t)=\begin{cases} I_{s0}+K_I t & \text{if} < T_I \\ I_{s0}+K_I \cdot T_I & \text{if } t \geq T_I \end{cases} \qquad \textbf{(4-141)}$$

The period T_I (when the current amplitude undergoes a ramp variation) is short which implies that the motor speed can be considered constant due to the rotor inertia. In these conditions, the general equation (4-100) becomes

$$\frac{d\underline{i}_r^{syn}}{dt}+\left(\frac{R_r}{L_r}+j\omega_{slp}\right)\underline{i}_r^{syn}=-\frac{L_m}{L_r}\cdot\left[j\omega_{slp}\left(I_{s0}+K_I\cdot t\right)+K_I\right] \qquad \textbf{(4-142)}$$

where I_{s0} is the initial current amplitude and K_I is a constant equal to the derivative of the current amplitude. The general solution of equation (4-142) is (4-143), where the parameters K, p, α, and β are complex constants. The expression $K\cdot e^{pt}$ is the general

solution of the homogenous equation derived from (4-142), while $\alpha t+\beta$ is a particular solution of the non-homogenous equation, whose parameters can be calculated by substituting this expression in (4-142).

$$\underline{i}_r = K \cdot e^{pt} + \alpha t + \beta \tag{4-143}$$

The constant K is determined considering the initial condition when the rotor current value corresponded to the initial steady-state operation:

$$K \cdot p + \beta = I_{r0} = \frac{-j\omega_{slp} L_m I_{s0}}{R_r + j\omega_{slp}} \tag{4-144}$$

Therefore the values of the constants are:

$$\begin{cases} \alpha = \dfrac{-j\omega_{slp} L_m K_I}{R_r + j\omega_{slp}} \\ \beta = -\dfrac{L_m\left(j\omega_{slp} I_{so} + K_I\right)}{R_r + j\omega_{slp}} + \dfrac{j\omega_{slp} L_m L_r K_I}{\left(R_r + j\omega_{slp}\right)^2} \\ p = -\left(\dfrac{R_r}{L_r} + j\omega_{slp}\right) \\ K = \dfrac{L_m L_r K_I}{\left(R_r + j\omega_{slp}\right)^2} + \dfrac{j\omega_{slp} L_m L_r^2 K_I}{\left(R_r + j\omega_{slp}\right)^3} \end{cases} \tag{4-145}$$

Thus, the variation of vector $\underline{i}_r$ during the transient has two components: a linear component varying with the speed α (proportional with the current derivative K_I), and an oscillatory component whose frequency equals the slip frequency ω_{slp}. In accordance with the general equations (4-7), the corresponding variations of the vector $\underline{e}$ are:

$$\underline{e}^{syn} = (-R_r + j\omega_{er} L_m) \cdot \left(Ke^{pt} + \alpha t + \beta\right) + j\omega_{er}\left(K_I t + I_{s0}\right) \cdot \frac{L_m^2}{L_r} \tag{4-146}$$

The mathematical form of this equation determines the type of slip estimation errors during the first part of the transient. When the slip angular frequency is small, the oscillatory character of the estimation error is not visible because period T_I is short (it is shorter than the oscillation period). If the motor slip is large, the error oscillation has a large frequency as well, and the oscillatory character of the estimation error becomes apparent (see the MATLAB simulation results in Fig. 4-18 and Fig. 4-19). The ramp variation is followed by a period of time when the stator current amplitude is constant. The result of this is a second transient during which the rotor current settles to a stable value. The equation describing the second transient is identical to (4-128), only the

initial conditions are different. Thus, the slip estimation oscillations during the second transient are similar to those generated by the stator angular frequency transients analysed in the previous section.

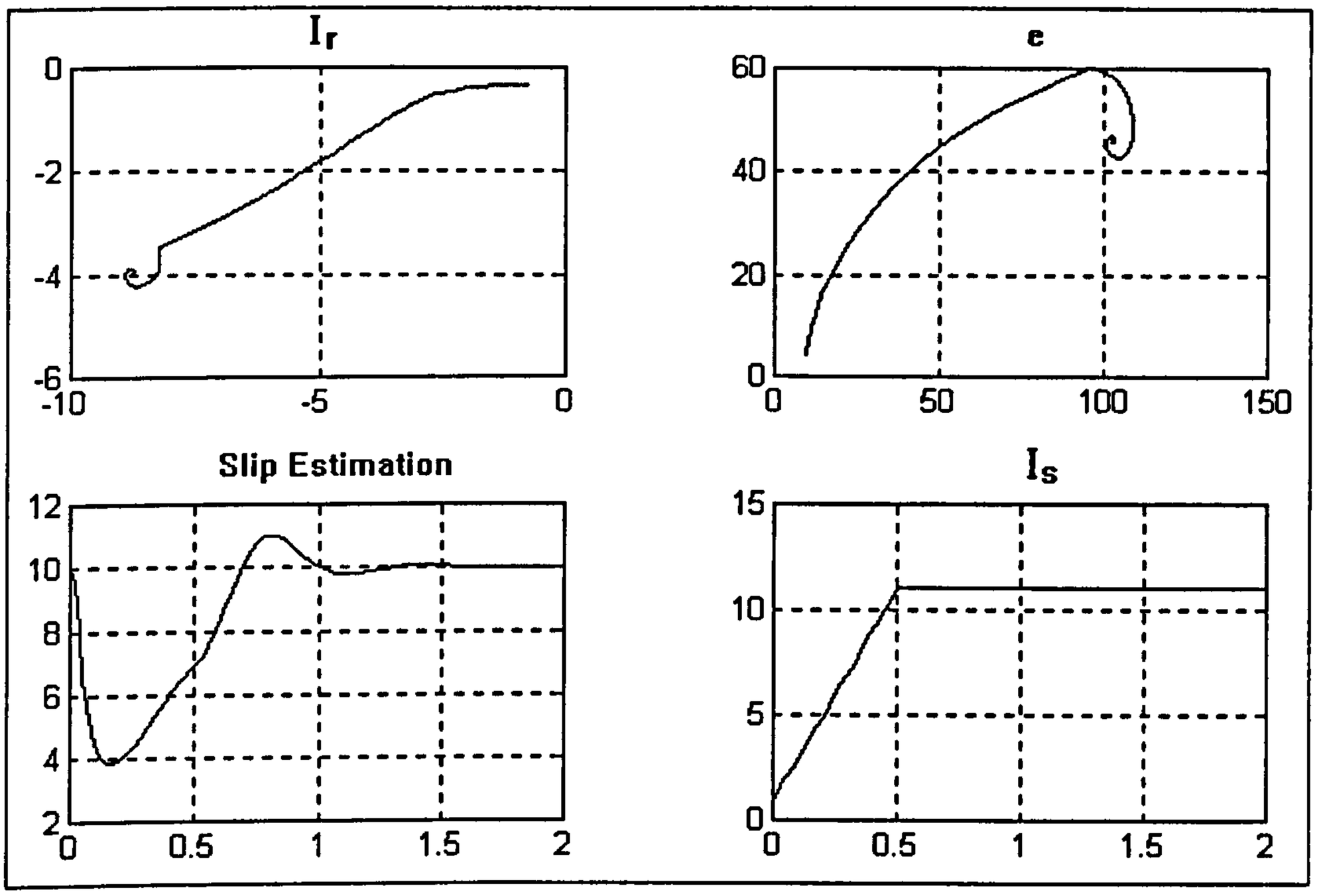

Fig. 4-18 - The slip estimator transient response (I_s modification). ω_{slp}=10 rad/s; T_I=0.5s

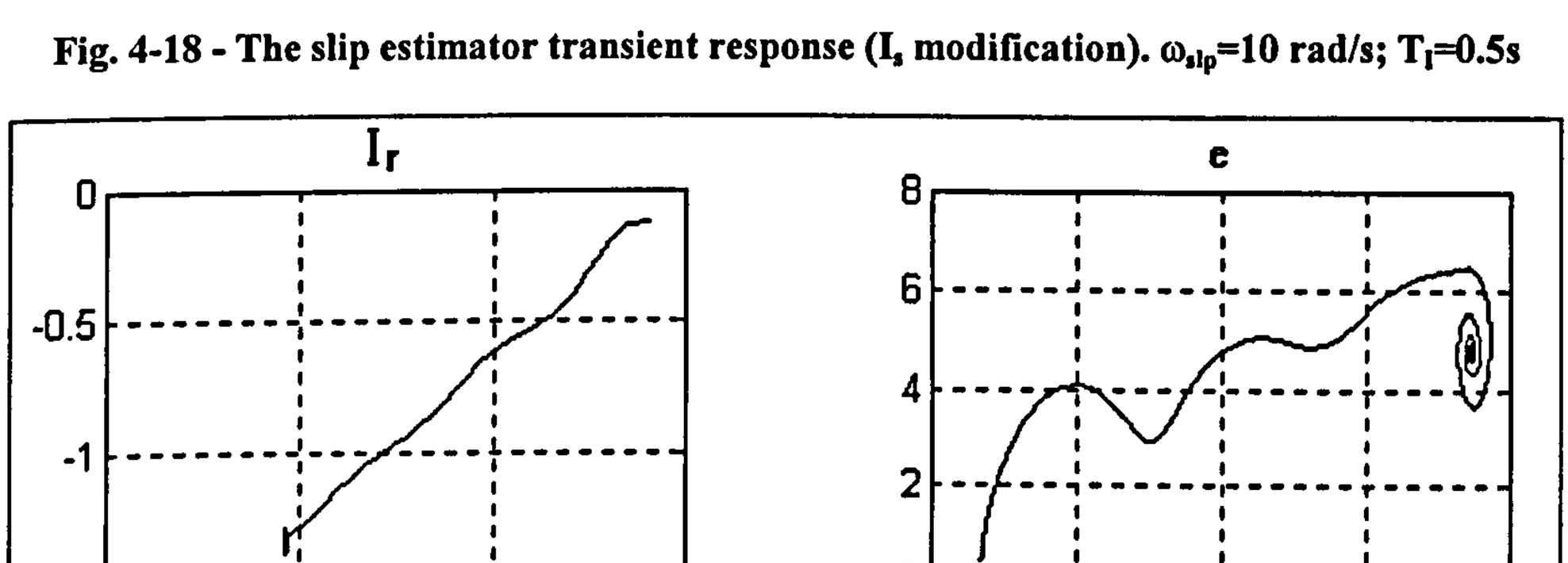

Fig. 4-19 - The slip estimator transient response (I_s modification). ω_{slp}=35 rad/s; T_I=0.5 s

If the stator current increase is quasi-instantaneous, then the T_I is very small and K_I is very large. The transient rotor current cannot follow the fast evolution of the

equilibrium value given by (4-101), but its trajectory is almost linear and the amplitude of the error oscillations is relatively small. During the second part of the transient, the rotor current $\underline{i}_r$ and the internal voltage $\underline{e}^{syn}$ have spiral trajectories in the complex plane, generating larger transients. The initial radius of these spirals increases with the speed of the first transient and so does the amplitude of the estimation errors.

These considerations are supported by the MATLAB simulation results in Fig. 4-18, Fig. 4-19, Fig. 4-20 and Fig. 4-21. The first two figures present slow transients (T_I=0.5 s) while the last two present fast transients (T_I=0.25 s). The rotor inertia has been considered infinite so that the rotor speed is constant. Fig. 4-18 can be directly compared to Fig. 4-20 as they involve the same slip angular frequency (ω_{slp}=10 rad/s) while Fig. 4-19 can be directly compared with Fig. 4-21. Analysing these figures it results that the global estimation errors increase with the transient speed. The errors during the first part are affected only by the transient speed, while the errors during the second part are affected by the slip angular frequency as well. The estimation errors can be significantly larger when ω_{slp} is large. This is due to the different positions of the $\underline{e}^{syn}$ spiral trajectory in the complex plane (the distance to the real axis).

Using the previous observations and calculations, an important conclusion can be drawn: the estimation errors can be decreased by restricting the transient speed (parameter K_I) and by maintaining the slip angular frequency at small values. Furthermore, the influence of ω_{slp} is more important than the influence of K_I.

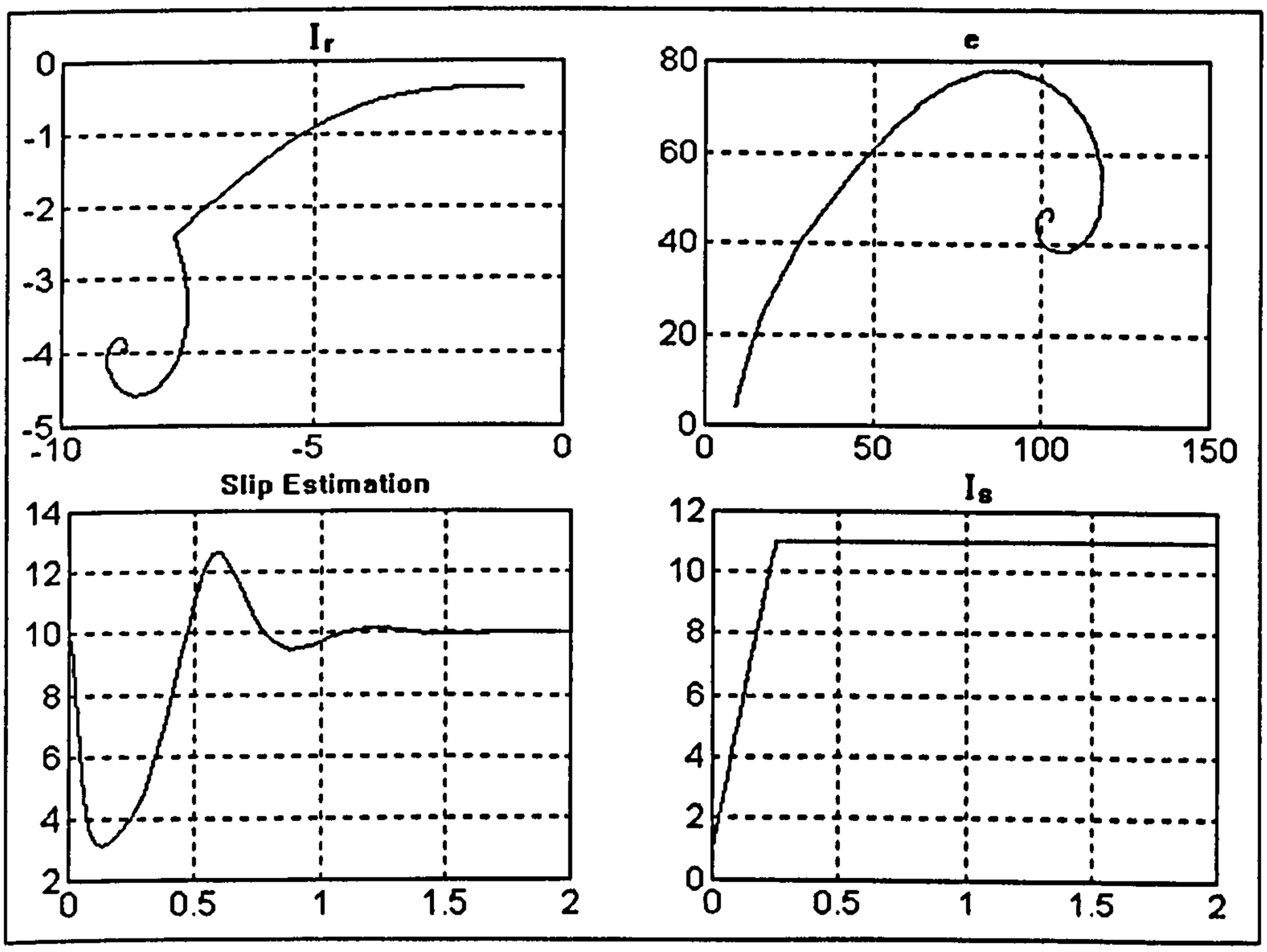

Fig. 4-20 - The slip estimator transient response (I_s modification). ω_{slp}=10 rad/s; T_I=0.25s

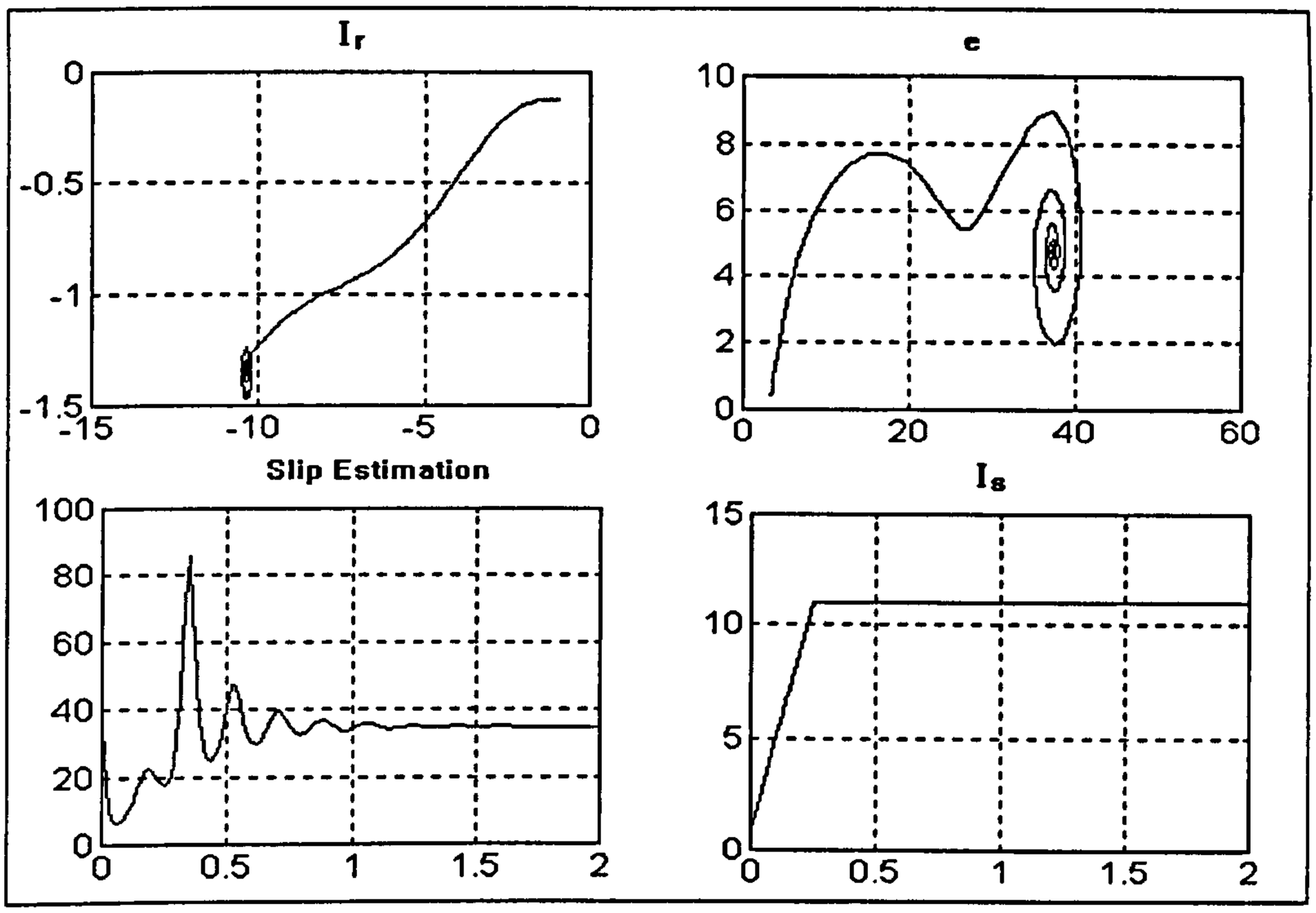

Fig. 4-21 - The slip estimator transient response (I_s modification). ω_{slp}=35 rad/s; T_l=0.25 s

4.3.1.2.3 General Transient Effects

Simultaneous changes of both the amplitude of the stator current I_s and the stator angular frequency ω_{es} produce combined electromagnetic echoes. The effect of the two changes can be enhanced or diminished internal voltage oscillations, depending on I_s and ω_{slp} variation in time. If the oscillations are diminished then the slip estimation errors are decreased during the transient operation. Amplified oscillations imply less accuracy in slip estimation.

The general variation of the rotor current corresponding to a pair of functions $I_s(t)$ and $\omega_{slp}(t)$ is given by (4-127). Substituting the rotor current $\underline{i}_r$ in (4-102) the function $\underline{e}^{syn}(t)$ can be calculated, thereby assessing the transient slip estimation errors. The exact correlation between these functions is very convoluted. However, there are a few practical rules applicable to all transients in the three categories analysed. These rules are derived based on the simulation results and can be used as guidelines to estimate the outcome of a certain transient in terms of slip estimation errors.

1. If the errors contain an oscillatory component the oscillation frequency is approximately equal to the motor slip frequency $f_{slp}=\omega_{slp}/2\pi$.
2. The average errors are large during fast transients and small during slow transients.

3. Given the same initial state and the same transient speed, the errors increase with the distance between the initial internal voltage vector $\underline{e}^{syn[1]}$ and the final internal voltage vector $\underline{e}^{syn[2]}$.
4. The estimation errors are much bigger during a transient at large ω_{slp} than the errors during a transient at small ω_{slp} because in the first case $\underline{e}^{syn}$ is situated closer to the real axis in the real plane and therefore $\tan^{-1}(\arg\{\underline{e}^{syn}\})$ undergoes larger variations.
5. For the same initial conditions, decreasing the stator current amplitude with ΔI_s generates larger estimation errors than increasing the current with the same ΔI_s.
6. For the same initial conditions, increasing the stator angular frequency with $\Delta\omega_{es}$ generates larger estimation errors than decreasing the angular frequency with the same $\Delta\omega_{es}$.

Rules 4, 5 and 6 can be explained based on geometrical considerations. The principles underlying the rule 4 have been discussed in a previous paragraph (The Effects of Altering the Stator Current Frequency), and they have been illustrated in Fig. 4-17. Rules 5 and 6 are justified by the fact that both the increase of stator angular frequency ω_{slp} and the decrease of the stator current amplitude I_s bring the vector $\underline{e}^{syn}$ closer to the real axis in Fig. 4-15. The opposite changes move $\underline{e}^{syn}$ further away from the real axis.

These conclusions are validated by the MATLAB simulation results in Fig. 4-22, Fig. 4-23, Fig. 4-24 and Fig. 4-25. All the four simulations imply equal variations of the slip angular frequency and of the stator current, but the signs of these variations and the initial values are different. The errors are maximal in Fig. 4-24 where ω_{slp} increases and I_s decreases. This combination of factors brings the centre of the internal voltage spiral trajectory close to the origin of the co-ordinate system. As a result, the trajectory intersects the real axis several times and a series of infinite slip estimation errors is produced. The error oscillations are smaller in the other situations where either ω_{slp} decreases (Fig. 4-23 and Fig. 4-25) or the increase of ω_{slp} is counterbalanced by the increase of I_s (Fig. 4-22).

The first important conclusion based on these simulation results is that the direct transitions never generate the same errors as the reverse transitions. The transient in Fig. 4-25 is the opposite of the transient in Fig. 4-22 and Fig. 4-24 is the opposite of Fig. 4-23 but the corresponding slip estimations are totally different. The second conclusion is that the position of the internal voltage spirals in the complex plane is more important than the size of the spirals. A spiral trajectory with a small radius placed

close to the real axis in the complex plane produces much larger estimation errors than a large spiral situated at a big distance from the real axis.

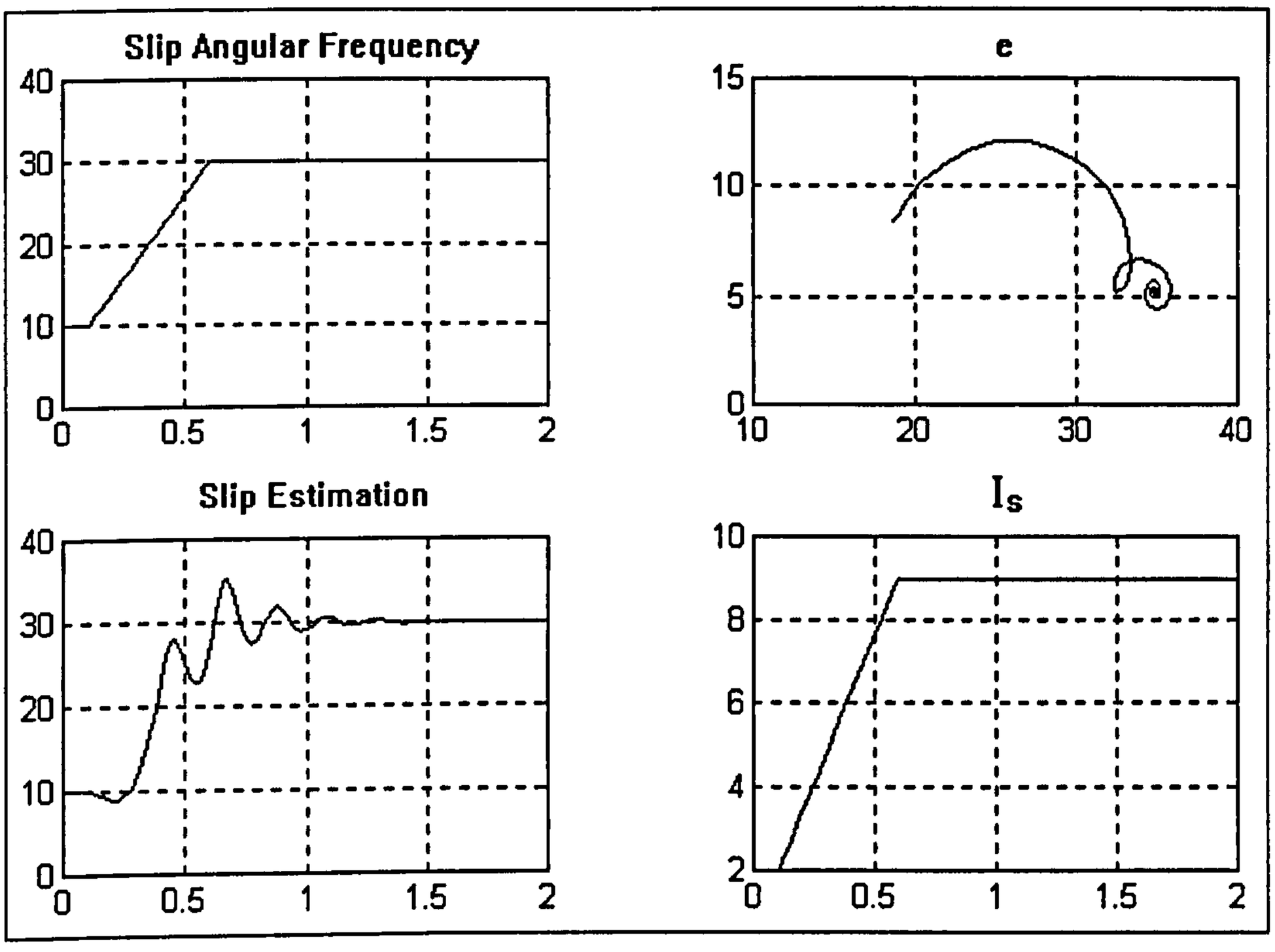

Fig. 4-22 – The effects of the simultaneous increase of ω_{slp} and I_s

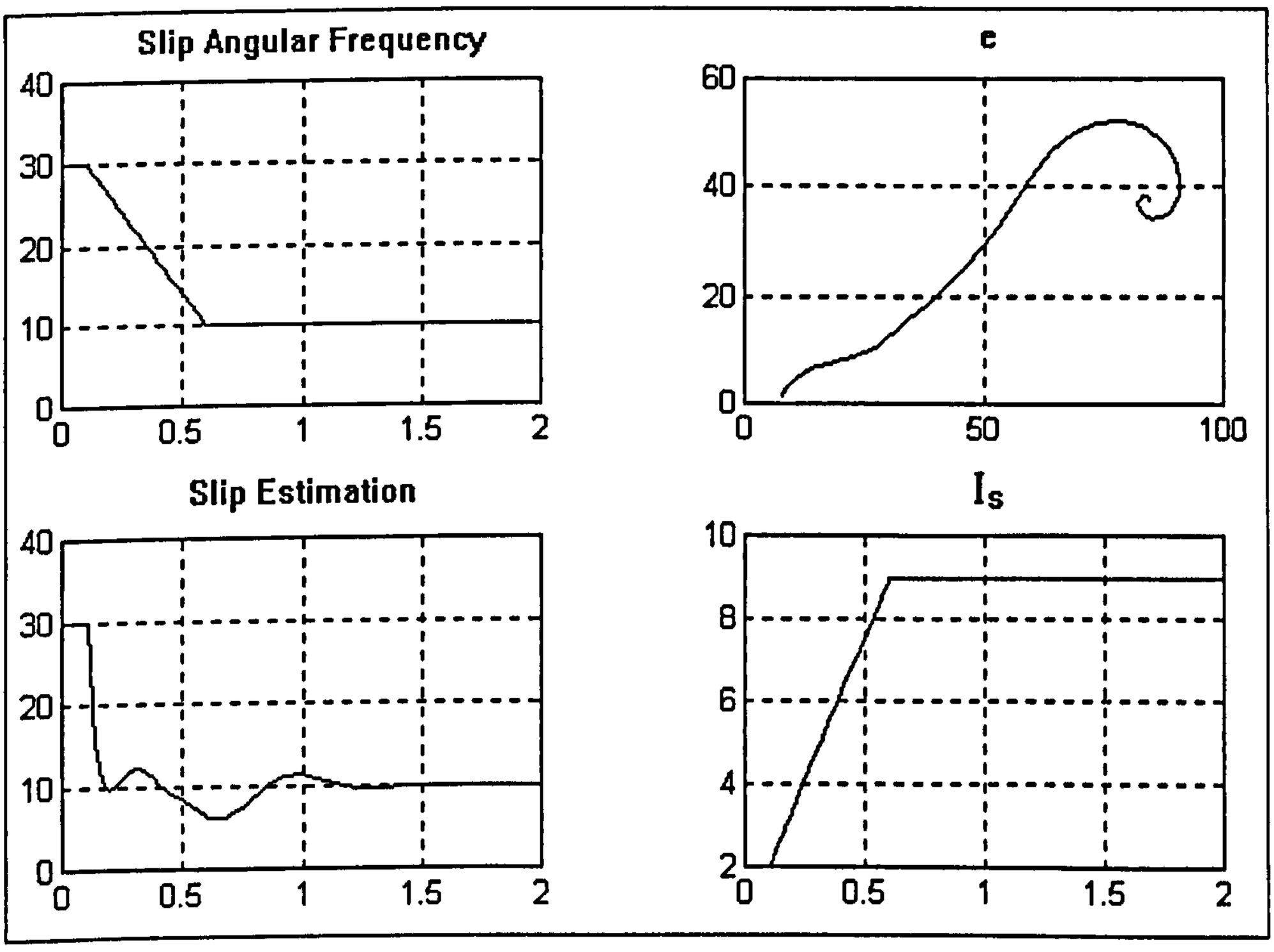

Fig. 4-23 – The effects of decreasing ω_{slp} while increasing I_s

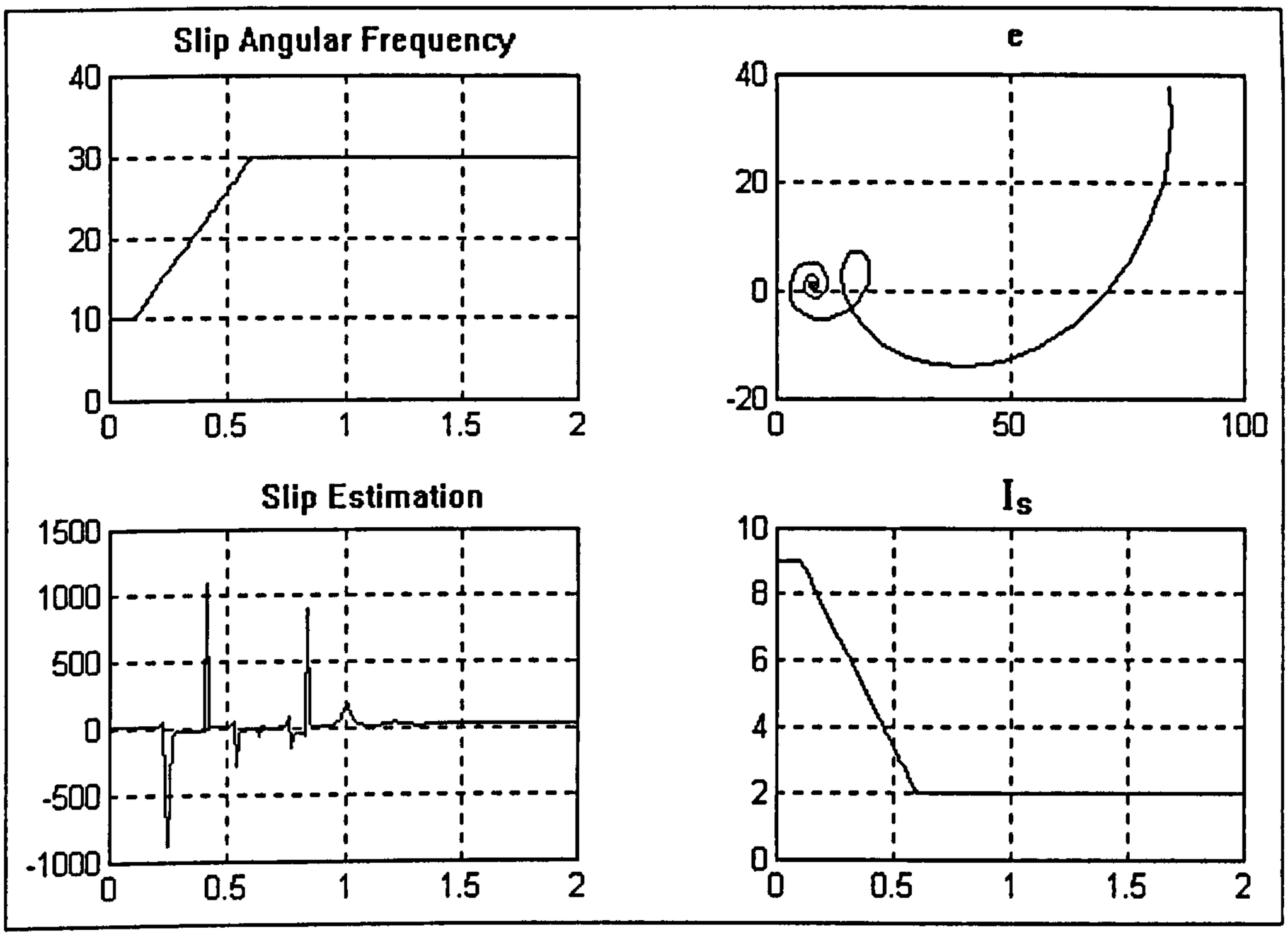

Fig. 4-24 - The effects of increasing ω_{slp} while decreasing I_s

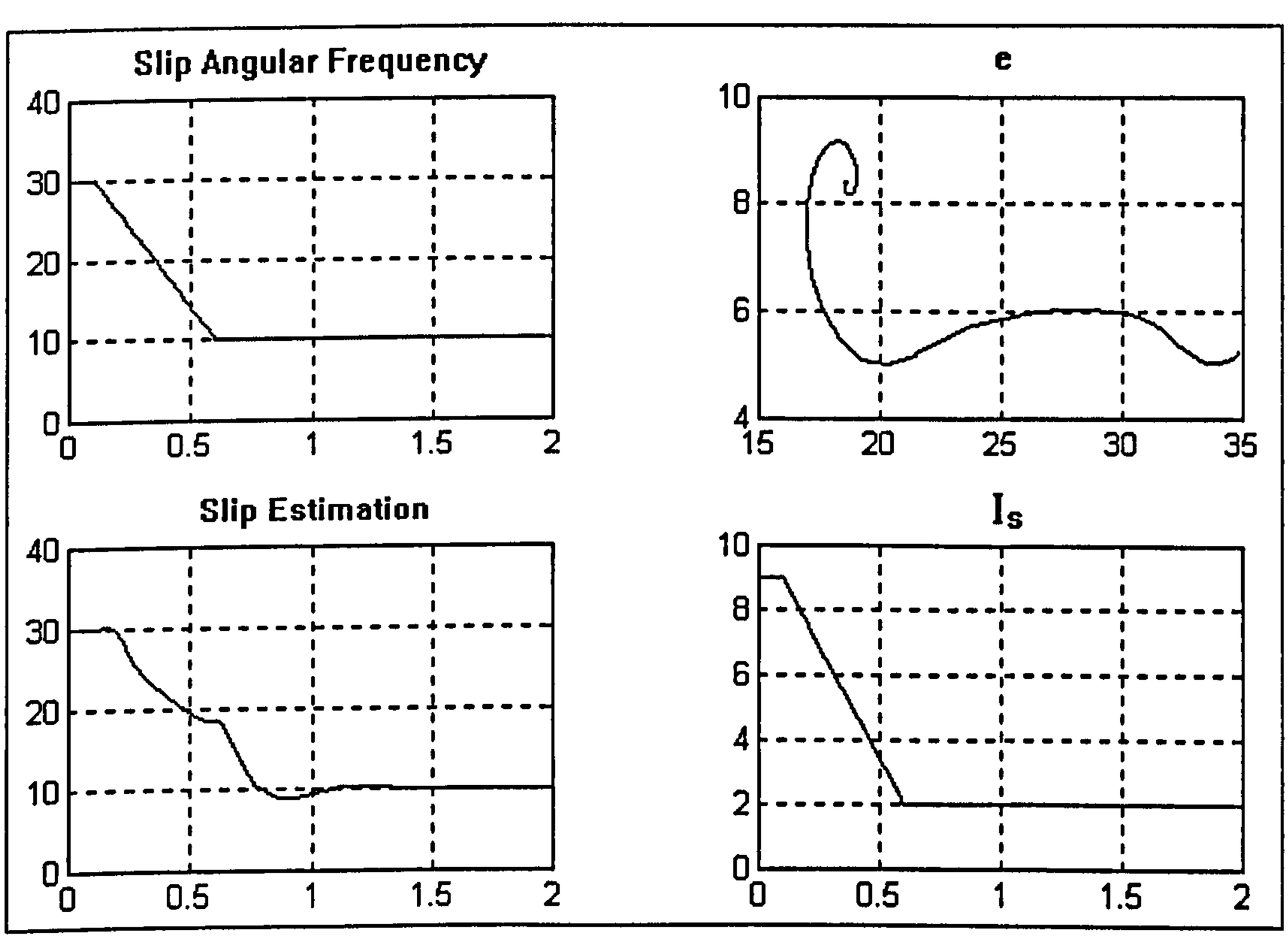

Fig. 4-25 – The effects of the simultaneous decrease of ω_{slp} and I_s

Therefore, the slip estimator based on the phase-shift between vectors $\underline{e}$ and $\underline{i}_s$ is accurate only in steady state motor operation. The slip estimations during transient operation are not reliable. Using such an estimator requires a special speed control

strategy that takes into account the large transient errors and compensates their effects. The design of the speed control strategy must be based on the six practical rules previously formulated.

4.3.2 The Novel Speed Control Algorithm

In accordance with the general principles exposed at the beginning of section 4.3, a novel speed control algorithm is proposed which can be expressed as a set of simple mathematical equations written in polar co-ordinates. The proposed speed control strategy incorporates the slip estimator based on the phase-shift between the vectors $\underline{e}$ and $\underline{i}_s$. The new method simultaneously carries out two interrelated tasks:

1) Controlling the rotor speed ω_r so that it follows the reference speed ω_{ref}.
2) Maintaining the slip angular frequency at a constant value: $\omega_{slp}=\Omega_{slp}$.

The two tasks are performed by controlling the angular frequency and the amplitude of the stator current. Thus, the speed controller contains two control loops. The slip control loop determines the stator current amplitude I_s in such a manner that ω_{slp} is maintained as close as possible to the reference value Ω_{slp}, while the speed control loop calculates the stator angular frequency ω_{es}.

4.3.2.1 The Slip Control Loop

The slip control loop implements a non-linear control strategy to keep ω_{slp} constant by modifying the stator current amplitude I_s. The stator current controls the rotor current and the interaction of the two generates the motor torque, which in turn affects the slip angular frequency. The induction motor torque is given by the general equation

$$T=\frac{2}{3}L_m\cdot \mathrm{Im}\left\{\underline{i}_s\cdot\underline{i}_r^*\right\} \tag{4-147}$$

If the expression of the rotor current for steady-state operation (4-101) is substituted in (4-147) then the steady-state motor torque is obtained as a function of current amplitude and slip angular frequency:

$$T(I_s,\omega_{slp})=L_m\cdot \mathrm{Im}\left\{\frac{j\omega_{slp}L_m I_s^2}{R_r-j\omega_{slp}L_r}\right\} \tag{4-148}$$

This expression can be further simplified as follows:

$$T(I_s,\omega_{slp}) = L_m \cdot \mathrm{Im}\left\{\frac{j\omega_{slp}L_m I_s^2\left(R_r + j\omega_{slp}L_r\right)}{R_r^2 + \omega_{slp}^2 L_r^2}\right\} \tag{4-149}$$

$$T(I_s,\omega_{slp}) = \frac{\omega_{slp}L_m^2 I_s^2 R_r}{R_r^2 + \omega_{slp}^2 L_r^2} \tag{4-150}$$

Thus, the motor torque increases with the stator current squared but has a non-linear variation against the slip speed. Fig. 4-26 illustrates the torque-slip characteristics for the steady-state operation of the 11.1 kW induction motor. The critical slip angular frequency at which the torque attains its maximum corresponds to the null torque derivative:

$$\frac{\partial T(I_s,\omega_{slp})}{\partial \omega_{slp}} = \frac{L_m^2 I_s^2 R_r\left(R_r^2 - \omega_{slp}^2 L_r^2\right)}{\left(R_r^2 + \omega_{slp}^2 L_r^2\right)^2} = 0 \tag{4-151}$$

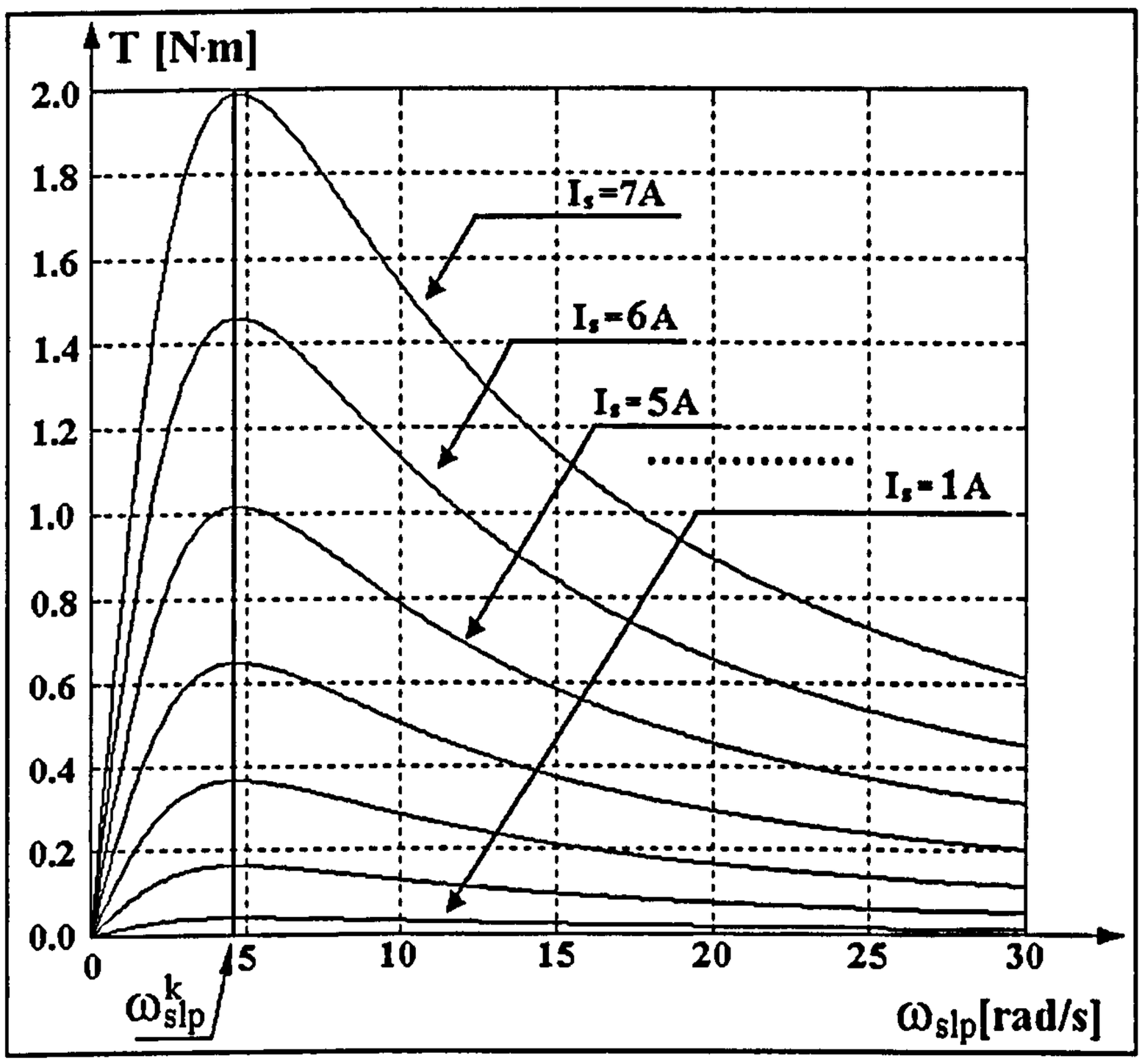

Fig. 4-26 - Steady-state torque variation for stator currents between 1A and 7A

From (4-151), the critical slip angular frequency ω_{slp}^k is calculated as

$$\omega_{slp}^k = \frac{R_r}{L_r} = \frac{1}{T_{er}} \tag{4-152}$$

The motor windings heat up during the operation. The result is a progressive increase of the stator and rotor resistances, which entails an increase of the rotor electrical time

constant T_{er}, and therefore an increase of the critical slip angular frequency. Thus, ω_{slp}^{k} is independent of the stator current amplitude I_s but depends on the rotor temperature. The actual variation of ω_{slp}^{k} during the motor operation depends on the construction details of the motor and on its operation mode.

In practical applications, the load torque T_l decreases with the decrease of the motor speed ($\partial T_l/\partial \omega_r>0$). The stability of the motor operation is ensured only if the motor torque T and the load torque T_l comply with condition

$$\text{sign}\left\{\frac{\partial T}{\partial \omega_r}\right\} = -\text{sign}\left\{\frac{\partial T_l}{\partial \omega_r}\right\} \Leftrightarrow \text{sign}\left\{\frac{\partial T}{\partial \omega_{slp}}\right\} = -\text{sign}\left\{\frac{\partial T_l}{\partial \omega_{slp}}\right\} \quad \text{(4-153)}$$

As a result, the motor speed is stable only if the slip angular frequency is in the interval $[0; \omega_{slp}^{k})$. Therefore, the reference slip angular frequency Ω_{slp} has to be set to a value situated inside this interval. According to Fig. 4-26, the motor slip can be varied at constant load torque by controlling the stator current amplitude. The slip control loop needs to increase the stator current amplitude I_s when the slip angular frequency ω_{slp} is larger than the reference Ω_{slp}, and to decrease it when the slip angular frequency is smaller then Ω_{slp}. The process requires information on the actual motor slip. To calculate this information, the control loop incorporates the slip estimation principles based on the phase shift between $\underline{e}^{syn}$ and $\underline{i}^{syn}$. Thus, maintaining a constant slip angular frequency during the steady-state operation is equivalent to maintaining a constant angle between $\underline{e}^{syn}$ and $\underline{i}^{syn}$.

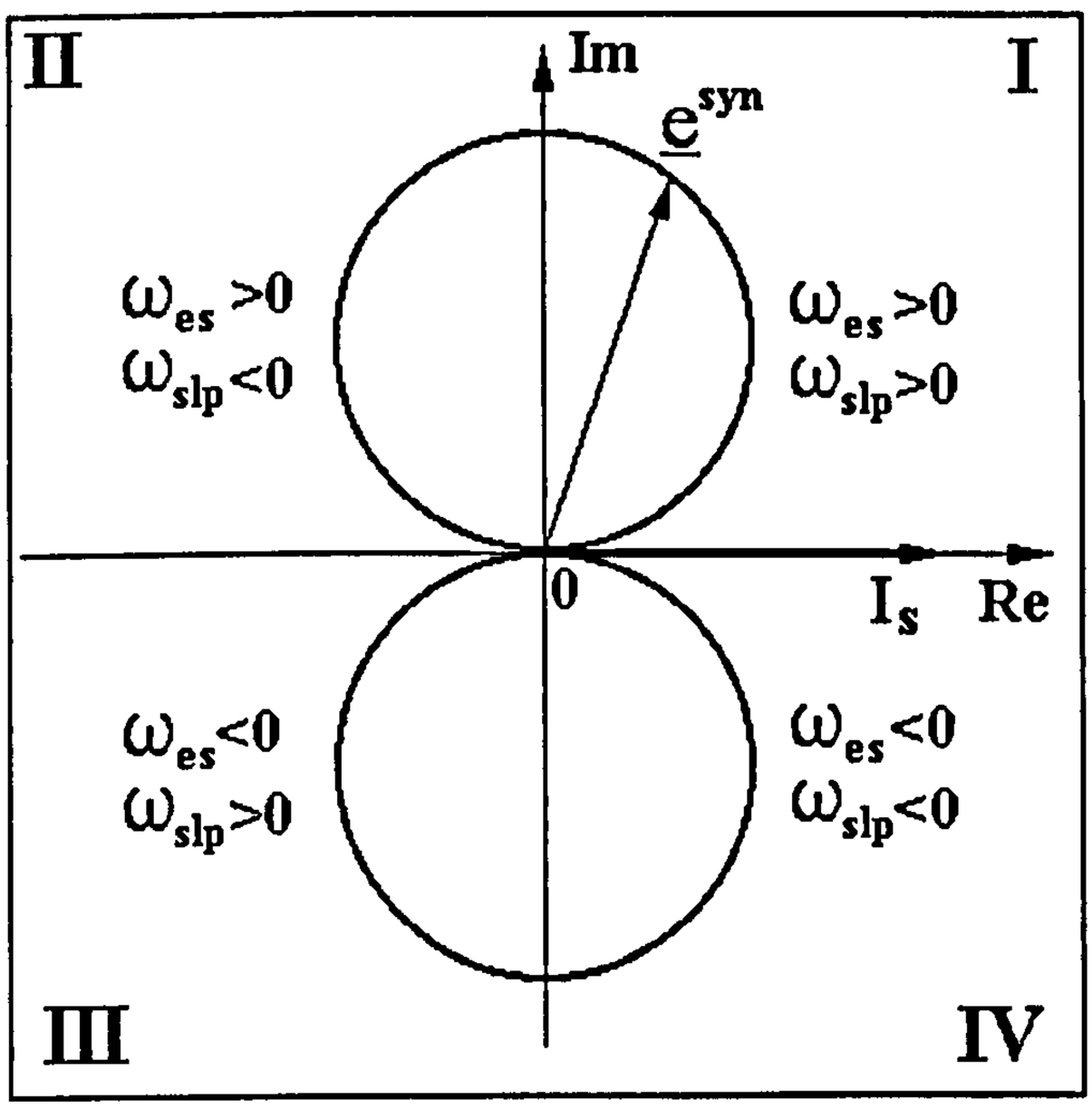

Fig. 4-27 - Internal load voltage locus in the complex plane

According to equations (4-138) and (4-139), the locus of $\underline{e}^{syn}$ is a set of circles tangent to the real axis of the rectangular synchronous reference frame. The stator current amplitude I_s is proportional to the circle radius so that for a given stator current amplitude the locus is made up of the two circles illustrated in Fig. 4-27.

As demonstrated by the system (4-154) derived from (4-108), the quadrant where the internal voltage $\underline{e}^{syn}$ is situated, depends on the sign of the stator angular frequency ω_{es} and on the sign of the slip angular frequency ω_{slp}:

$$\begin{cases} \text{sign}\{\text{Re}\{\underline{e}^{syn}\}\} = \text{sign}\{\omega_{es}\} \cdot \text{sign}\{\omega_{slp}\} \\ \text{sign}\{\text{Im}\{\underline{e}^{syn}\}\} = \text{sign}\{\omega_{es}\} \end{cases} \quad \textbf{(4-154)}$$

In most practical applications, the load torque opposes the motor shaft rotation. In this situation, the absolute value of the rotor speed is smaller than the absolute value of the magnetic field speed. Therefore ω_{es} and ω_{slp} have the same sign and $\underline{e}^{syn}$ is situated either in quadrant I or in quadrant IV of Fig. 4-27.

$$|\omega_{es}| > |\omega_{er}| \Rightarrow \text{sign}(\omega_{slp}) = \text{sign}(\omega_{es}) \quad \textbf{(4-155)}$$

There is one category of applications where the torque may not be opposed to the shaft rotation: the cranes and the elevators. When an elevator is moving down, its weight creates a torque that tends to accelerate the shaft rotation. As a result, the rotor moves faster than the motor magnetic field, and the slip angular frequency sign is the opposite of the stator angular frequency sign. In this situation, the vector $\underline{e}^{syn}$ is situated in either quadrant II or in quadrant III.

$$|\omega_{es}| < |\omega_{er}| \Rightarrow \text{sign}(\omega_{slp}) = -\text{sign}(\omega_{es}) \quad \textbf{(4-156)}$$

In conclusion, the sign of the reference slip angular frequency Ω_{slp} must be dependent on the stator angular frequency sign and on the nature of the load. It has to be positive for the motor operation in quadrants I and III and it is negative otherwise. The motor operation in the four quadrants corresponds to four different internal voltage vectors for steady-state operation: $\underline{e}^{syn}(\Omega_{slp1})$, $\underline{e}^{syn}(\Omega_{slp2})$, $\underline{e}^{syn}(\Omega_{slp3})$, $\underline{e}^{syn}(\Omega_{slp4})$ where $|\Omega_{slp1}|=|\Omega_{slp2}|=|\Omega_{slp3}|=|\Omega_{slp4}|$. The values of the four reference slip values Ω_{slp} have to be smaller in absolute value than the module of the critical slip angular frequency $|\omega_{slp}^{k}|$. If equation (4-152) is substituted in (4-118), the result is:

$$\tan^{-1}\alpha_{ei} = \frac{\omega_{slp}}{\omega_{slp}^{k}} \quad \textbf{(4-157)}$$

Consequently, the internal voltage vectors corresponding to the motor operation at critical slip are placed at 45° with regard to the reference frame axes. At slip values smaller in absolute values than $|\omega_{slp}^{k}|\tan^{-1}\alpha_{ei}$ decreases and if ω_{slp} is null then $\tan^{-1}\alpha_{ei}$ is null as well. As shown in Fig. 4-28, the vectors $\underline{e}^{syn}(\Omega_{slp1})$, $\underline{e}^{syn}(\Omega_{slp2})$, $\underline{e}^{syn}(\Omega_{slp3})$, $\underline{e}^{syn}(\Omega_{slp4})$ need to be situated in the sectors limited by the imaginary axis of the synchronous reference system and by the vectors $\underline{e}^{syn}(\omega_{slp1})$, $\underline{e}^{syn}(\omega_{slp2})$, $\underline{e}^{syn}(\omega_{slp3})$, $\underline{e}^{syn}(\omega_{slp4})$. Based on these considerations, the slip control principle can be formulated as follows:

A) When the internal voltage vector $\underline{e}^{syn}$ lies in one of the shaded areas in Fig. 4-28, the controller decreases the stator current amplitude in order to increase the absolute value of the slip angular frequency $|\omega_{slp}|$.

B) When the internal voltage vector lies outside the shaded sectors the speed controller needs to decrease the stator current amplitude in order to increase the $|\omega_{slp}|$.

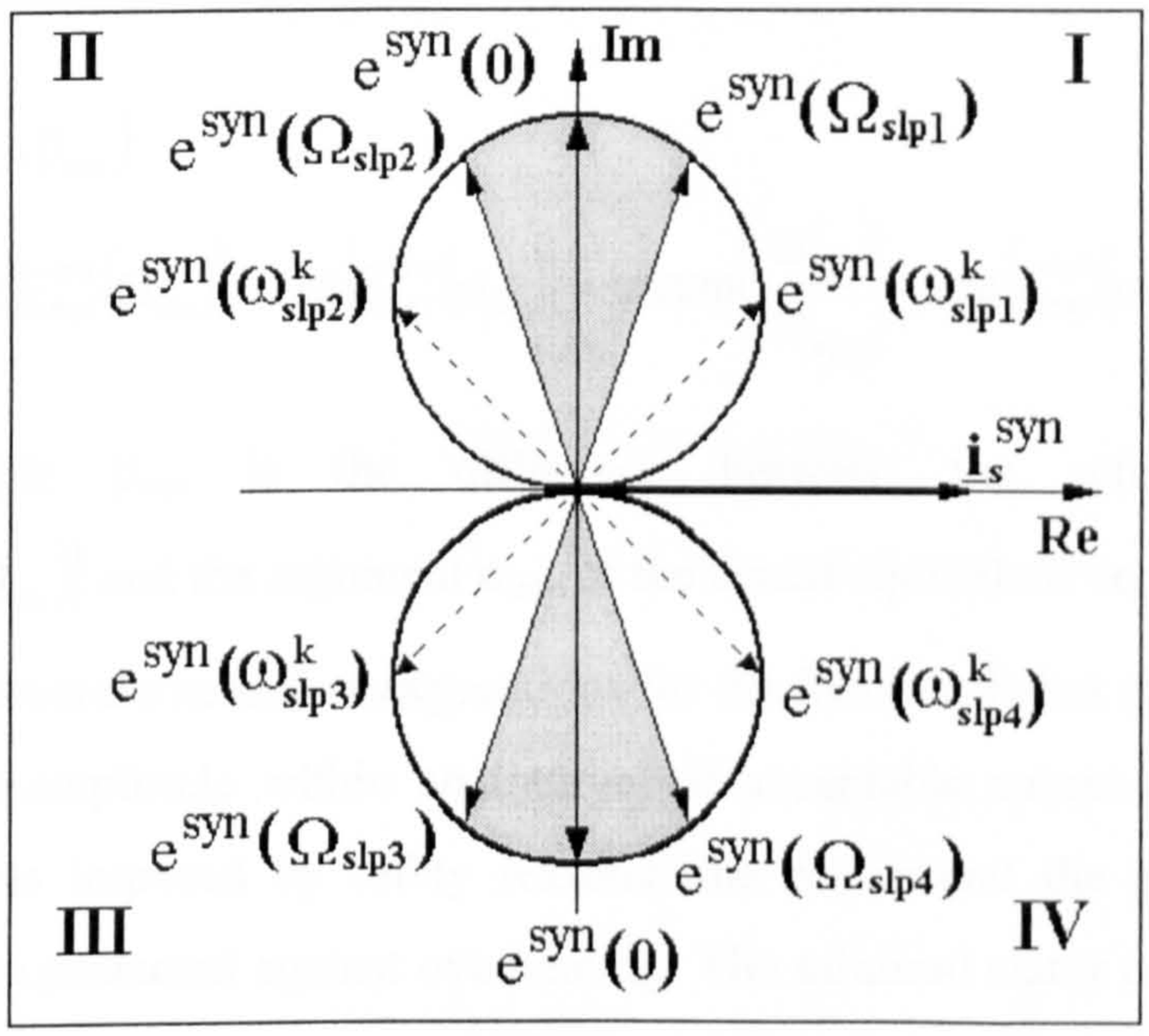

Fig. 4-28 - Characteristic points on the $\underline{e}^{syn}$ locus and the corresponding slip angular frequencies

Due to the symmetry in Fig. 4-28 the calculations referring to four quadrants can be reduced to equivalent calculations in only one quadrant. The transformation from four quadrants to one is carried out by replacing the real and imaginary parts of vector $\underline{e}^{syn}$ with their absolute values. The result is an equivalent vector $\underline{E}_{eqv}^{syn}$ given by

$$\underline{E}_{eqv}^{syn} = \left|\mathrm{Re}\left\{\underline{e}^{syn}\right\}\right| + j\cdot\left|\mathrm{Im}\left\{\underline{e}^{syn}\right\}\right| = \left|\underline{E}_{eqv}^{syn}\right|\cdot\left[\cos\left(\alpha_{eqv}^{ref}\right) + j\cdot\sin\left(\alpha_{eqv}^{ref}\right)\right] \qquad \textbf{(4-158)}$$

that is always situated in the first quadrant, as illustrated in Fig. 4-29.

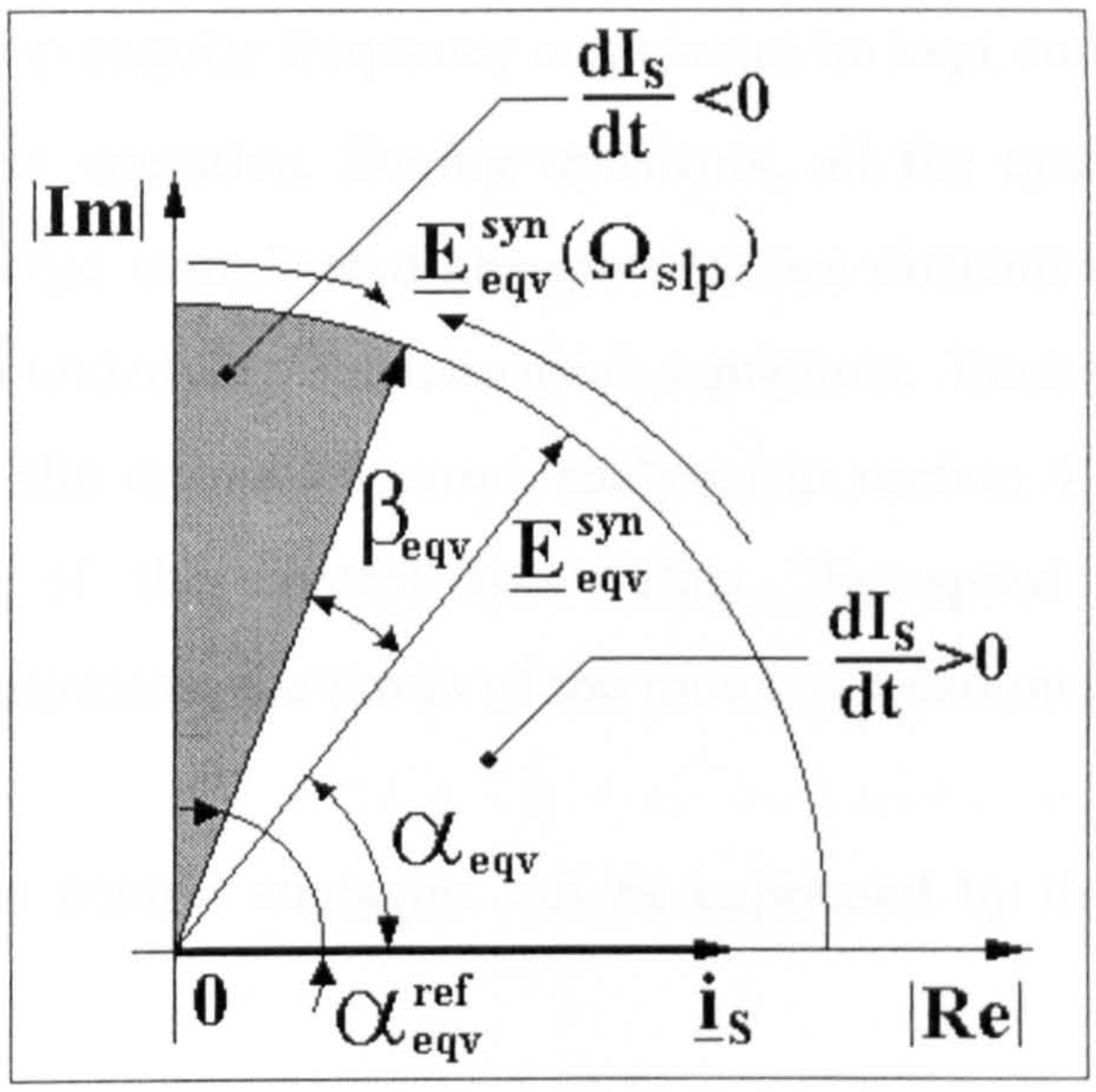

Fig. 4-29 – The reduction of the four quadrants to one

The rules A) and B) concerning the control of the stator current amplitude can therefore be expressed as the differential equation

$$\begin{cases} \dfrac{dI_s}{dt} = F_I\left(I_s, \beta_{eqv}\right) \\ \beta_{eqv} = \arg\left\{\underline{E}_{eqv}^{syn}\left(\Omega_{slp}\right)\right\} - \arg\left\{\underline{E}_{eqv}^{syn}\left(\omega_{slp}\right)\right\} = \arctan\left(\dfrac{\omega_{slp}^{k}}{\Omega_{slp}}\right) - \arg\left\{\underline{E}_{eqv}^{syn}\left(\omega_{slp}\right)\right\} \end{cases} \quad \textbf{(4-159)}$$

where the angle β_{eqv} is the difference between the reference argument $\alpha_{eqv}^{ref} = \arg\left\{\underline{E}_{eqv}^{syn}\left(\Omega_{slp}\right)\right\}$ and the argument α_{eqv} of the actual equivalent vector $\underline{E}_{eqv}^{syn}$.

There are several alternative expressions for the function F_I but all of them have to limit the current amplitude within an interval of acceptable values $[I_{smin}; I_{smax}]$. The maximum limit is imposed by safety reasons: the motor and the power electronics circuitry has to be protected against overheating. The minimal stator current is imposed so that the internal voltage amplitude $|\underline{e}^{syn}|$ does not decrease under the limit where its argument cannot be calculated. Several versions of function F_I are analysed in section 4.3.2.3 and the corresponding motor control performance is assessed.

4.3.2.2 The Speed Control Loop

If the slip angular frequency ω_{slp} is maintained constant then the steady-state relation between the rotor mechanical speed and the stator electrical angular frequency is linear:

$$\omega_{es} = \Omega_{slp} + \omega_{er} = \Omega_{slp} + p \cdot \omega_r \quad \textbf{(4-160)}$$

However, the slip angular frequency ω_{slp} cannot be kept constant at the reference value Ω_{slp} in transient operation. During transients, all the quantities describing the motor operation undergo complicated changes that are difficult to control due to the non-linearity of the underlying mathematical equations. Furthermore, the transient operation causes the slip estimation errors analysed in section 4.3.1.2, thereby raising the difficulty level of the control task. Thus, the speed control loop needs simultaneously to compensate the errors of the motor slip estimation and to control the rotor speed.

All the possible control strategies can be expressed by the general differential equations

$$\begin{cases} \omega_{es}^{ref} = \text{sign}\{\omega_r^{ref}\} \cdot \Omega_{slp} + \omega_{er}^{ref} = \text{sign}\{\omega_r^{ref}\} \cdot \Omega_{slp} + p \cdot \omega_r^{ref} \\ \dfrac{d\omega_{es}}{dt} = F_\omega\left(\omega_{es}^{ref}, \omega_{es}, I_s, \beta_{eqv}\right) \end{cases} \quad \textbf{(4-161)}$$

where the function 'sign' is defined by

$$\text{sign}(x) = \begin{cases} +1 \text{ when } x > 0 \\ 0 \text{ when } x = 0 \\ -1 \text{ when } x < 0 \end{cases} \quad \textbf{(4-162)}$$

Individual strategies rely on different forms of the function F_ω.

4.3.2.3 Alternative Sensorless Speed Control Strategies

Any speed control strategy can be defined by the two functions F_ω and F_I involved in equations (4-159) and (4-161). The simplest control version is defined by the functions

$$\begin{cases} F_I\left(I_s, \beta_{eqv}\right) = \begin{cases} K_I \cdot \beta_{eqv} & \text{when } I_s \in \left(I_{s-min}; I_{s-max}\right) \\ 0 & \text{when } I_s \notin \left(I_{s-min}; I_{s-max}\right) \end{cases} \\ F_\omega\left(\omega_{es}^{ref}, \omega_{es}, I_s, \beta_{eqv}\right) = K_\omega \cdot \text{sign}\left(\omega_{es}^{ref} - \omega_{es}\right) \end{cases} \quad \textbf{(4-163)}$$

where K_I and K_ω are proportionality constants. In this case, the derivative of the current amplitude is proportional to the angular error β_{eqv}, while the derivative of the angular frequency depends on the sign of the stator frequency error. Therefore, the motor control is linear and uses two P controllers operating in an independent manner, as the two functions F_I and F_ω are calculated based on different parameters. This type of control is appropriate when the application requirements do not include fast transient operation.

Fig. 4-30 and Fig. 4-31 present the 11.1 kW motor response to a step change of the reference speed for two different values of the parameter K_ω. In the first simulation K_ω=240 s^{-1}, while in the second simulation K_ω=2x10^4 s^{-1}, so the stator frequency varies much faster than in the first case. The rest of the simulation parameters are the same in both cases: K_I=50 A/s·rad, I_{min}=0.5 A, Imax=24.5 A, Ω_{slp}=1.31 rad/s, J=0.015 N·m^2. The load torque is considered proportional to the rotor speed. Although the stator angular frequency varies faster in Fig. 4-30, the motor attains the reference speed in a shorter time period in the situation presented in Fig. 4-30. Moreover, the average stator current amplitude is lower when K_ω has a lower value. This implies that the motor efficiency is better in Fig. 4-30 than in Fig. 4-31.

Using this control strategy, the motor behaves similarly to a synchronous machine with a start-up cage rotor:

- It is able to generate a constant speed for a certain range of load torque values.
- The rotor speed accurately follows the variations of the stator frequency if this variation is slow.
- If the variations of the stator frequency are too fast, they take the rotor out of synchronism and the speed response becomes relatively slow.

The simulation results prove that the transient slip estimation errors do not affect the stability of the drive system operation. The errors cause oscillations of the angles α_{eqv} and β_{eqv} but the stator current amplitude is given by

$$I_s(t) = K_I \cdot \int \beta_{eqv}(t) \cdot dt \qquad (4\text{-}164)$$

so that the effect of these oscillations is filtered out by integration. However, control strategies of increased complexity are required to obtain a fast system the system response step changes of ω_{ref}. Very fast induction motor transient responses are typically obtained using the rotor field oriented control strategy. The new speed control strategy can be improved by finding two functions F_I and F_ω that emulate the behaviour of a rotor field oriented controller. Thus, the field generating current component i_{sd} needs to be maintained constant while modifying the torque generating current component according to the speed error. This requires the calculation of the position $\theta_\psi(t)$ of the rotor flux vector $\underline{\psi}_r$ and the equation system (4-165) to be solved.

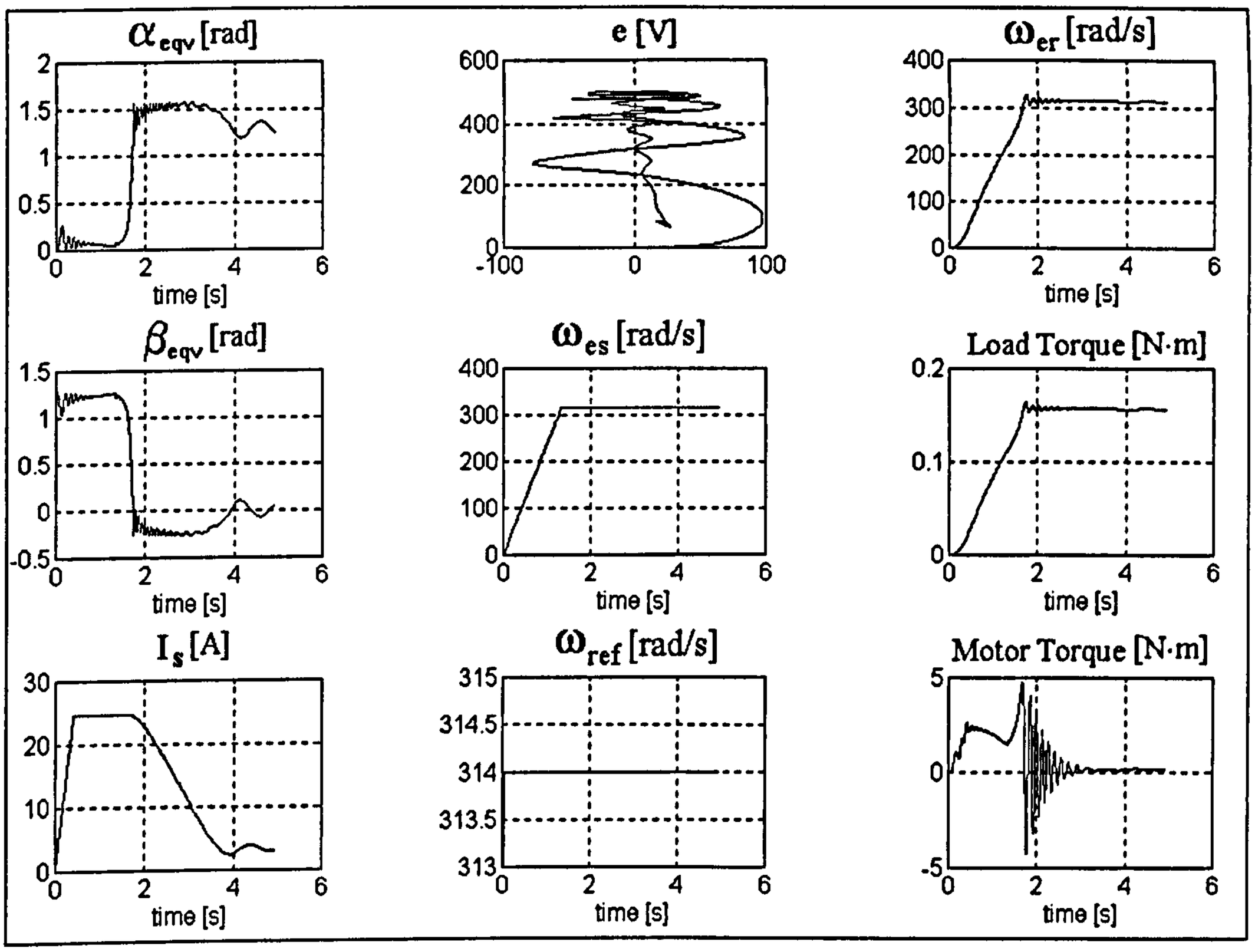

Fig. 4-30 - Motor Response to Gradual Stator Angular frequency Change

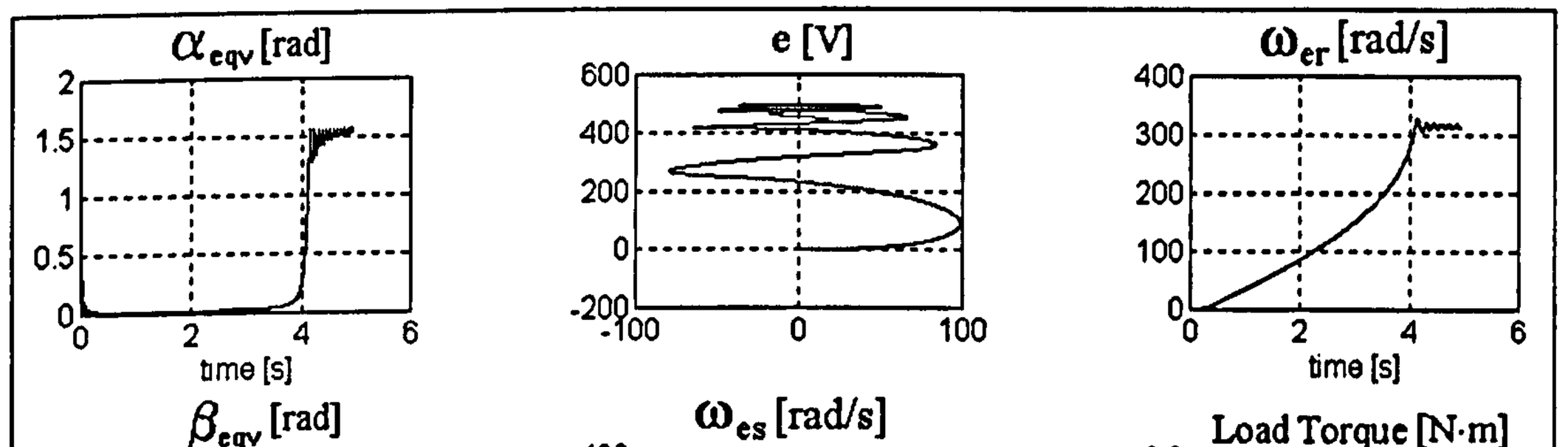

Fig. 4-31 – Motor Response to Sudden Stator Angular frequency Change

$$\begin{cases} \underline{i}_{sd} = I_s(t)\cdot\cos\left(\int_0^t \omega_{es}(t)\cdot t\cdot dt - \theta_\psi(t)\right) = \text{const.} \\ \underline{i}_{sd} = I_s(t)\cdot\sin\left(\int_0^t \omega_{es}(t)\cdot t\cdot dt - \theta_\psi(t)\right) = f(\omega_{ref} - \omega_r) \end{cases} \quad (4\text{-}165)$$

The rotor flux vector is not calculated by the new speed control strategy in order to minimise the calculation amount. On the other hand, solving the equation system (4-165) would increase the hardware implementation complexity to an unacceptable level. However, this strategy can approximate the position of $\underline{\Psi}_r$ using the position of $\underline{e}^{syn}$ for large and medium power motors. The internal voltage vector is defined by (4-102). If the speed is larger than a few revolutions per second, then $\underline{e}^{syn}$ is approximately perpendicular on the rotor flux vector $\underline{\Psi}_r$ because the rotor resistance can be neglected as compared to motor reactance. Under these conditions, it can be used to determine the position of vector $\underline{\Psi}_r$ in the complex plane.

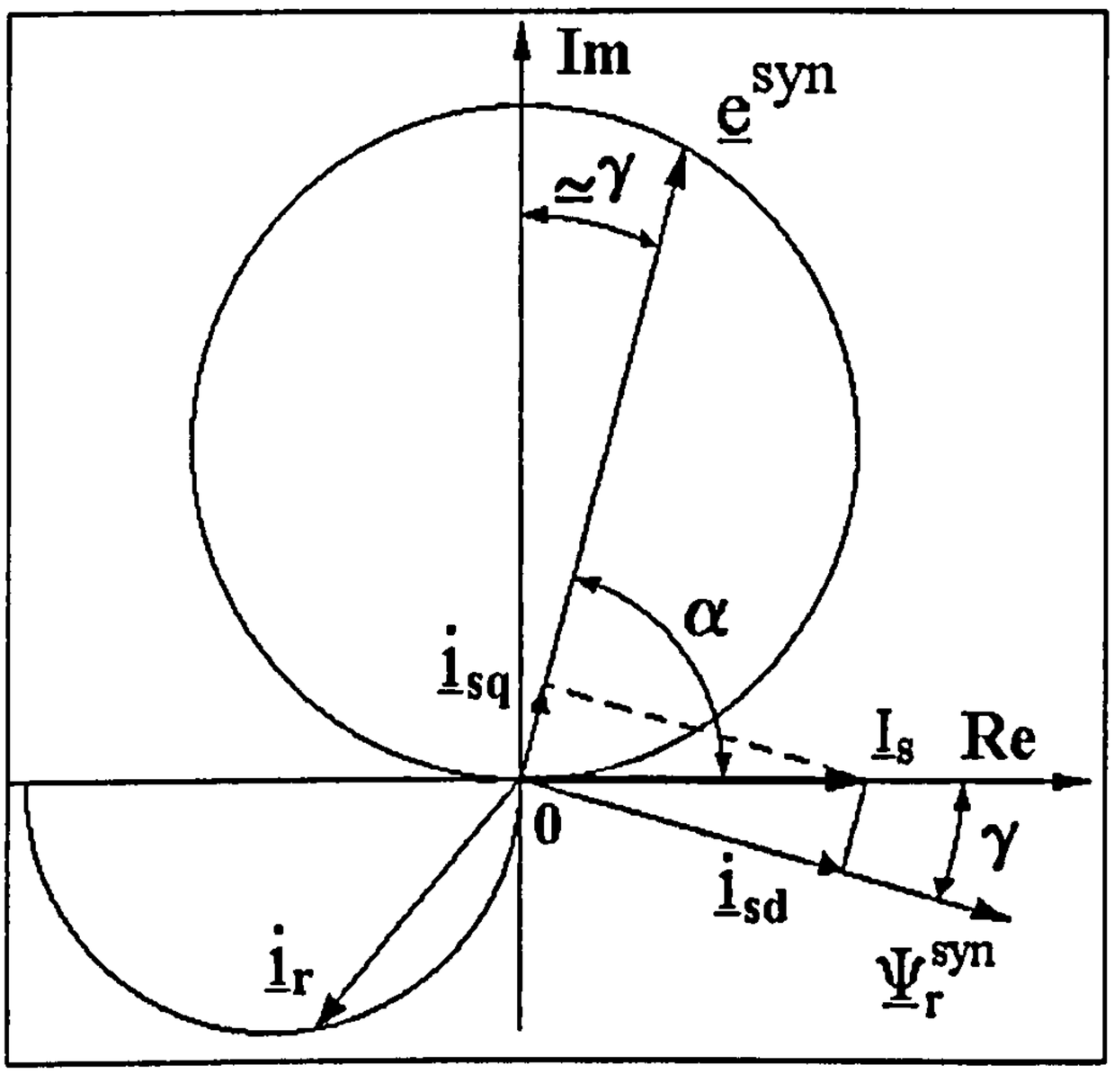

Fig. 4-32 – The relative position of vectors $\underline{e}$, i and $\underline{\Psi}_r$ in the synchronous reference frame

Fig. 4-32 indicates the typical positions of the vectors $\underline{e}$, i and $\underline{\Psi}_r$ in the synchronous reference frame. Modifying the motor speed requires a modification of the motor torque. The field orientation solution is to alter the stator current component i_{sq} while keeping i_{sd} constant. According to the new control method, the task is achieved by simultaneously changing the stator angular frequency ω_{es} and the stator current amplitude I_s. The two stator current components are given by

$$\begin{cases} i_{sd} = I_s\cdot\cos\gamma \\ i_{sq} = I_s\cdot\sin\gamma \end{cases} \quad (4\text{-}166)$$

where the angle γ is indicated in Fig. 4-32, while the derivatives of the two components are

$$\begin{cases} \frac{di_{sd}}{dt} = \frac{dI_s}{dt}\cos\gamma - I_s \sin\gamma \cdot \frac{d\gamma}{dt} = 0 \\ \frac{di_{sq}}{dt} = \frac{dI_s}{dt}\sin\gamma + I_s \cos\gamma \cdot \frac{d\gamma}{dt} \end{cases} \quad (4\text{-}167)$$

The derivative di_{sd}/dt is ideally null during the motor speed change and therefore the variation of the stator current amplitude I_s depends on γ according to

$$\frac{dI_s}{dt} = I_s \cdot tg\gamma \cdot \frac{d\gamma}{dt} \quad (4\text{-}168)$$

which is derived from the first equation (4-167). Substituting (4-168) into the second equation (4-167), the result is

$$\frac{dI_s}{dt} = \frac{di_{qs}}{dt} \cdot \frac{tg\gamma}{tg\gamma \cdot \sin\gamma + \cos\gamma} \quad (4\text{-}169)$$

which demonstrates that the stator current component i_{sq} increases with the increase of I_s. In the same time, any variation of I_s has to comply with the condition (4-168). Consequently, any increase of the stator current amplitude I_s has to be simultaneous with an increase of the angle $\gamma=\pi/2-\arg\{\underline{e}^{syn}\}$. As demonstrated by the simulation results, the variation of I_s generates an initial increase of $\arg\{\underline{e}^{syn}\}$ followed by a decrease. This variation is reflected in the opposite alteration of the slip estimation results and of the angle γ (therefore an unwanted result). To maintain the correct relation between I_s and γ in accordance with (4-168), the stator angular frequency ω_{es} needs to be altered simultaneously with I_s so that the effects of ω_{es} variations compensate the unwanted effects over angle γ. It was proven in section 4.3.1.2 that increasing the slip angular frequency $\omega_{slp}=\omega_{es}-\omega_{er}$ leads to oscillations starting with an initial decrease of $\arg\{\underline{e}^{syn}\}$. This decrease can cancel out the unwanted increase caused by the modification of I_s. The subsequent oscillations of angle γ resulting from the modification of I_s need to be cancelled out by suitable variations of ω_{es}. The exact analytical solution to this problem is very difficult to find and the corresponding hardware implementation is too complex due to the non-linearity of the solution. However, simplified solutions, equivalent to **quasi-field oriented control methods**, can be investigated based on a few principles derived from the previous considerations and from the rules governing the slip estimation process. The principles are:

- The value of $\arg\{\underline{e}^{syn}\}$ needs to be maintained at values close to 90° (Ω_{slp} has to be small) to maintain the slip estimation errors at acceptable levels.
- The rotor speed changes are always initiated by the speed control loop according to equations (4-160) and (4-161). The stator current variations compensate for the unwanted oscillations of angle γ, which can be calculated as a function of β_{eqv} and α_{eqv}^{ref}.
- The angle γ has to be allowed to increase during the speed changes simultaneously with the increase of I_s. This is equivalent with a simultaneous increase of β_{eqv} and I_s, which can be easily achieved if $\partial F_I/\partial\beta_{eqv}>0$.
- If the angles γ and β_{eqv} become too large, the stator frequency variation speed has to be limited in order to reduce the motor slip and the error slip estimations. On the other hand, the stator frequency is allowed to undergo fast speed changes as long as the motor slip has small values.

One of the simplest solutions that complies with the above principles is given by

$$\begin{cases} F_I(I_s,\beta_{eqv})=\begin{cases} K_I\cdot\beta_{eqv} & \text{if} \quad I_s\in(I_{s-min};I_{s-max}) \\ 0 & \text{if} \quad I_s\notin(I_{s-min};I_{s-max}) \end{cases} \\ F_\omega(\omega_{es}^{ref},\omega_{es},I_s,\beta_{eqv})=\begin{cases} \text{sign}(\omega_{es}^{ref}-\omega_{es})\cdot(K_{\omega1}-\beta_{eqv}\cdot K_{\omega2}) & \text{if } \beta_{eqv}<\beta_{max} \\ \text{sign}(\omega_{es}^{ref}-\omega_{es})\cdot(K_{\omega1}-\beta_{max}\cdot K_{\omega2}) & \text{if } \beta_{eqv}\geq\beta_{max} \end{cases} \end{cases} \tag{4-170}$$

where F_ω is a piecewise linear function (Fig. 4-33) defined by the constants K_I, $K_{\omega1}$, $K_{\omega2}$, β whose optimal values depend on the motor parameters. The functions (4-170) represent the basic version of the new sensorless speed control algorithm proposed in this thesis.

This control solution has been tested by simulations on the 11.1 kW motor using different values for the constants in (4-170). The MATLAB simulation results demonstrated substantially improved dynamic response, as exemplified in Fig. 4-34. In the same time, the method is capable of maintaining the rotor speed constant despite load torque variations (Fig. 4-35). The system response speed is approximately proportional to $K_{\omega1}$, but increasing $K_{\omega1}$ over a certain limit actually deteriorates the system response. This phenomenon is illustrated in Fig. 4-36, which can be compared with Fig. 4-34.

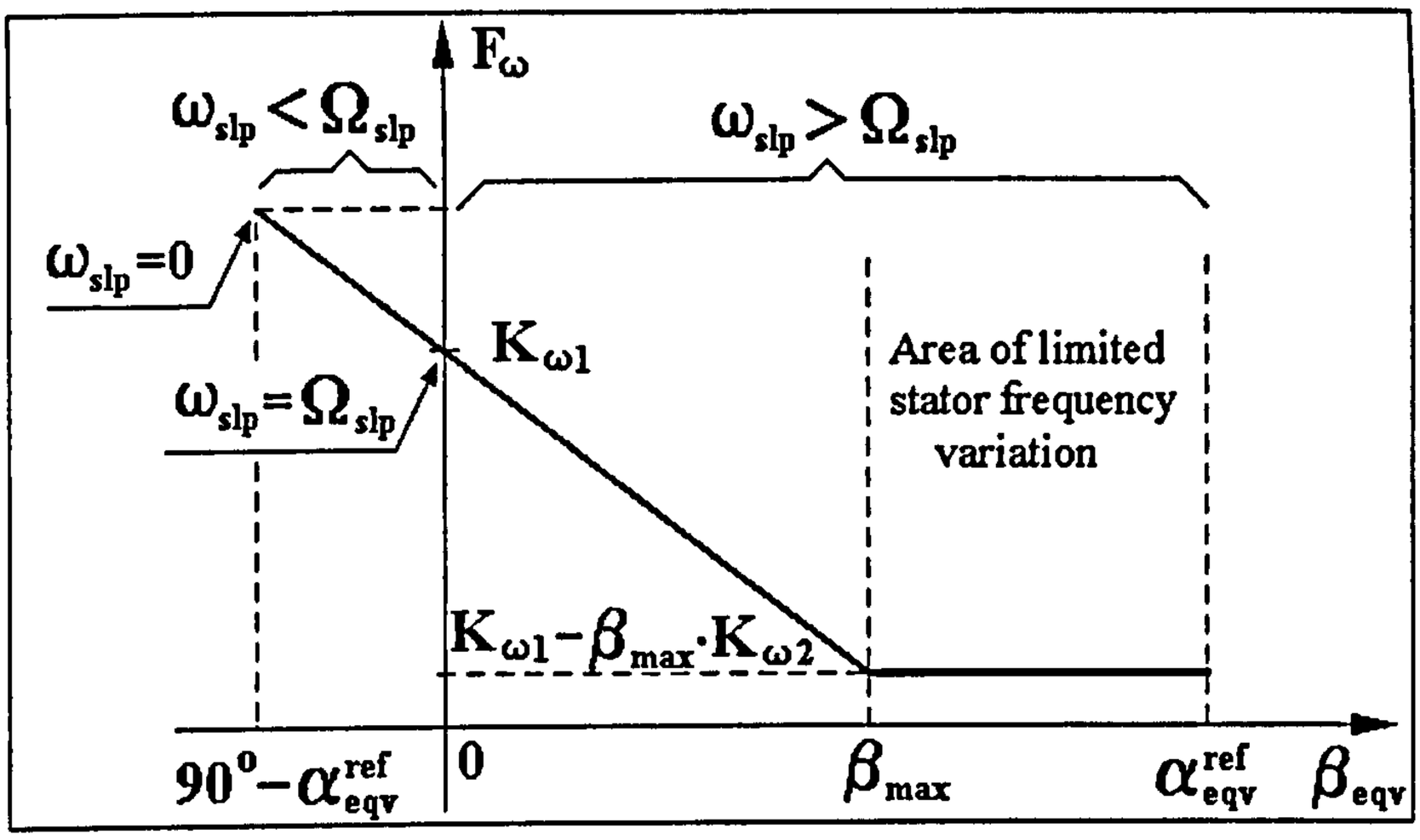

Fig. 4-33 – The variation of F_ω with β when $\omega_{cs}^{ref} > \omega_{cs}$

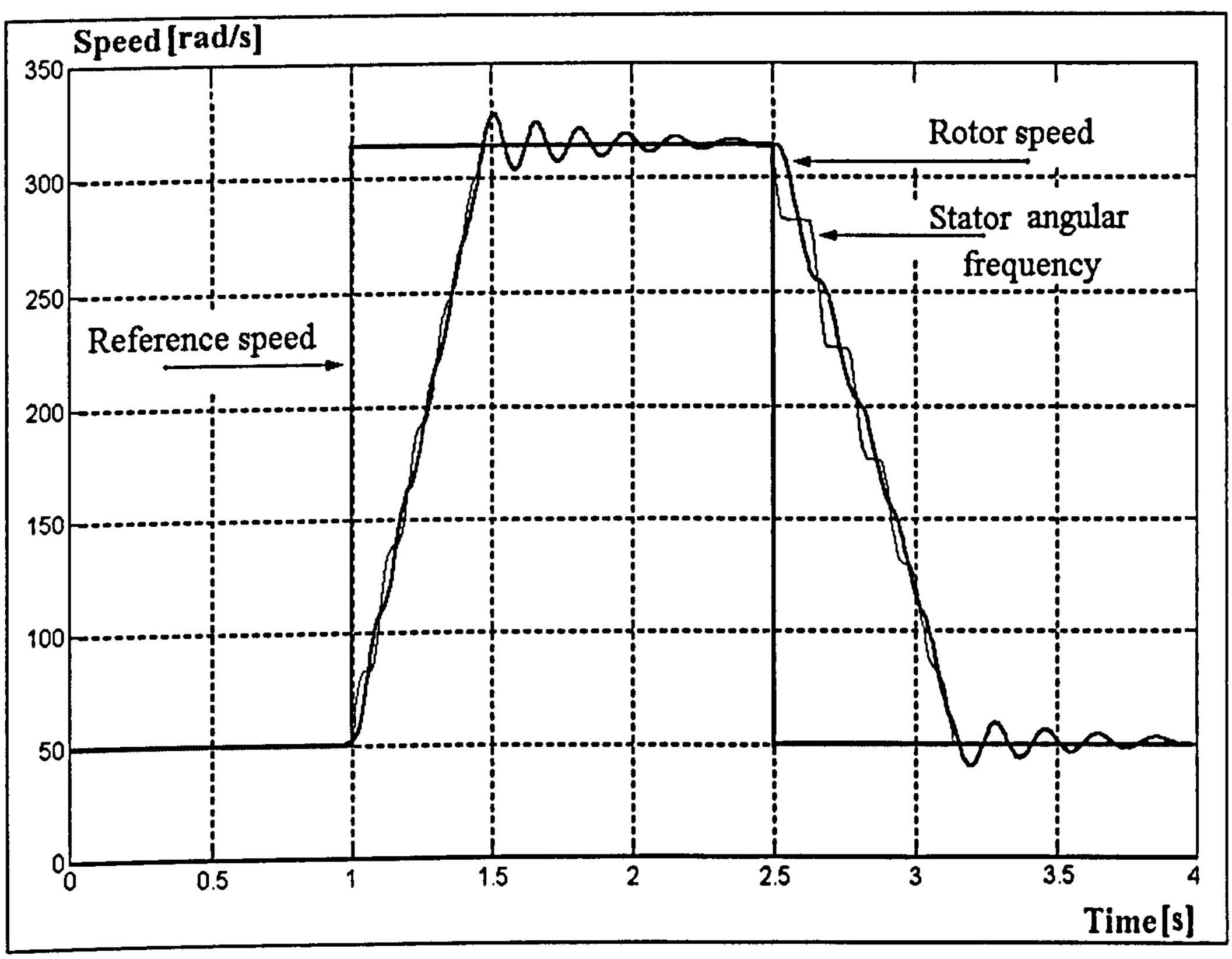

Fig. 4-34 – Quasi-field oriented control method results ($K_{\omega 1}$=1000 s^{-1})

As shown by these simulation results, the stator angular frequency undergoes non-linear variation caused by the non-linear mathematical model (4-170) of the control strategy. Again, the motor behaves in a similar manner to a synchronous motor: the rotor speed follows the stator frequency changes only if the speed of these changes is below a critical limit depending on the rotor inertia and on the load torque.

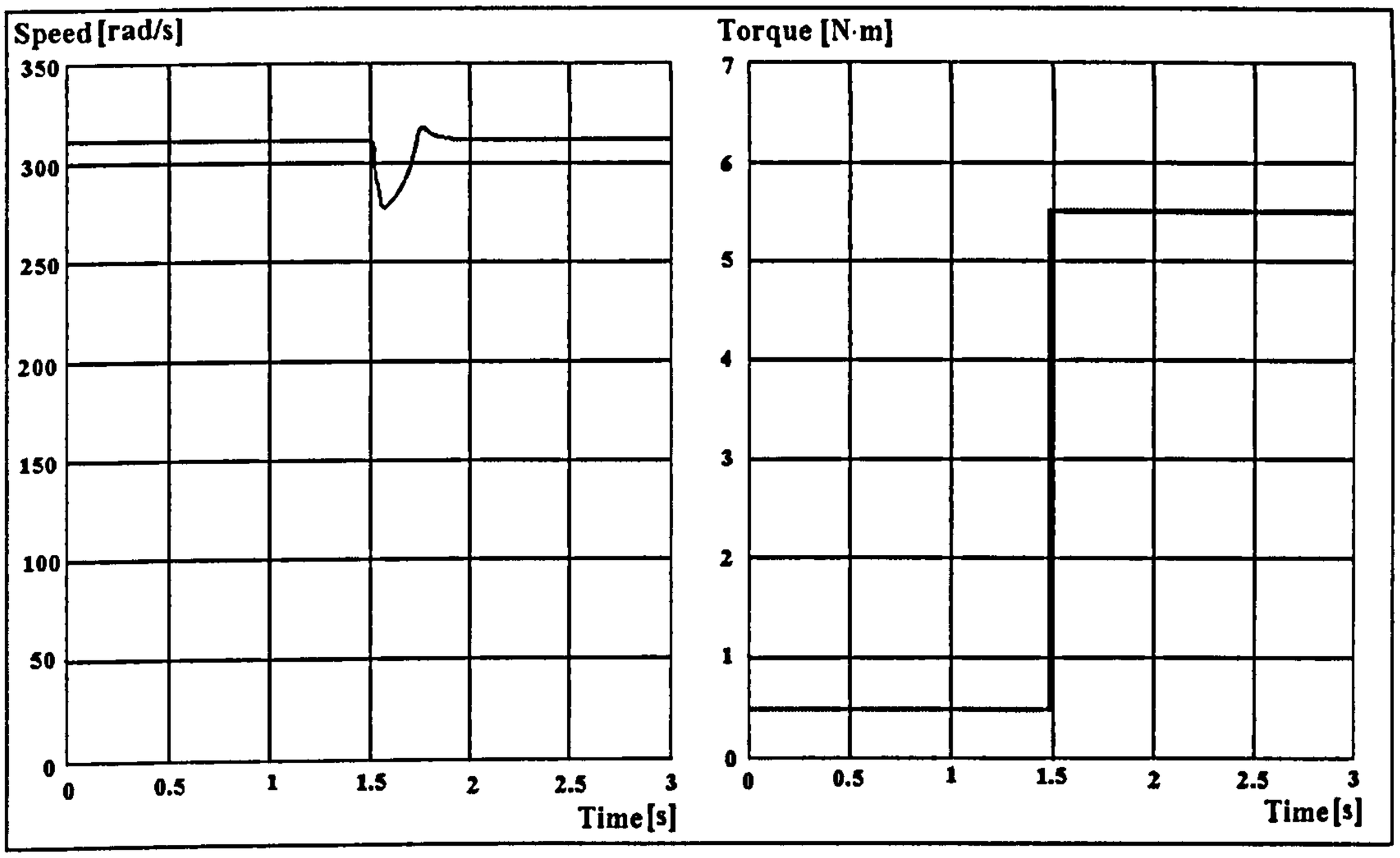

Fig. 4-35 – The motor speed variation during a step increase of the load torque ($K_{\omega 1}$=1000 s^{-1})

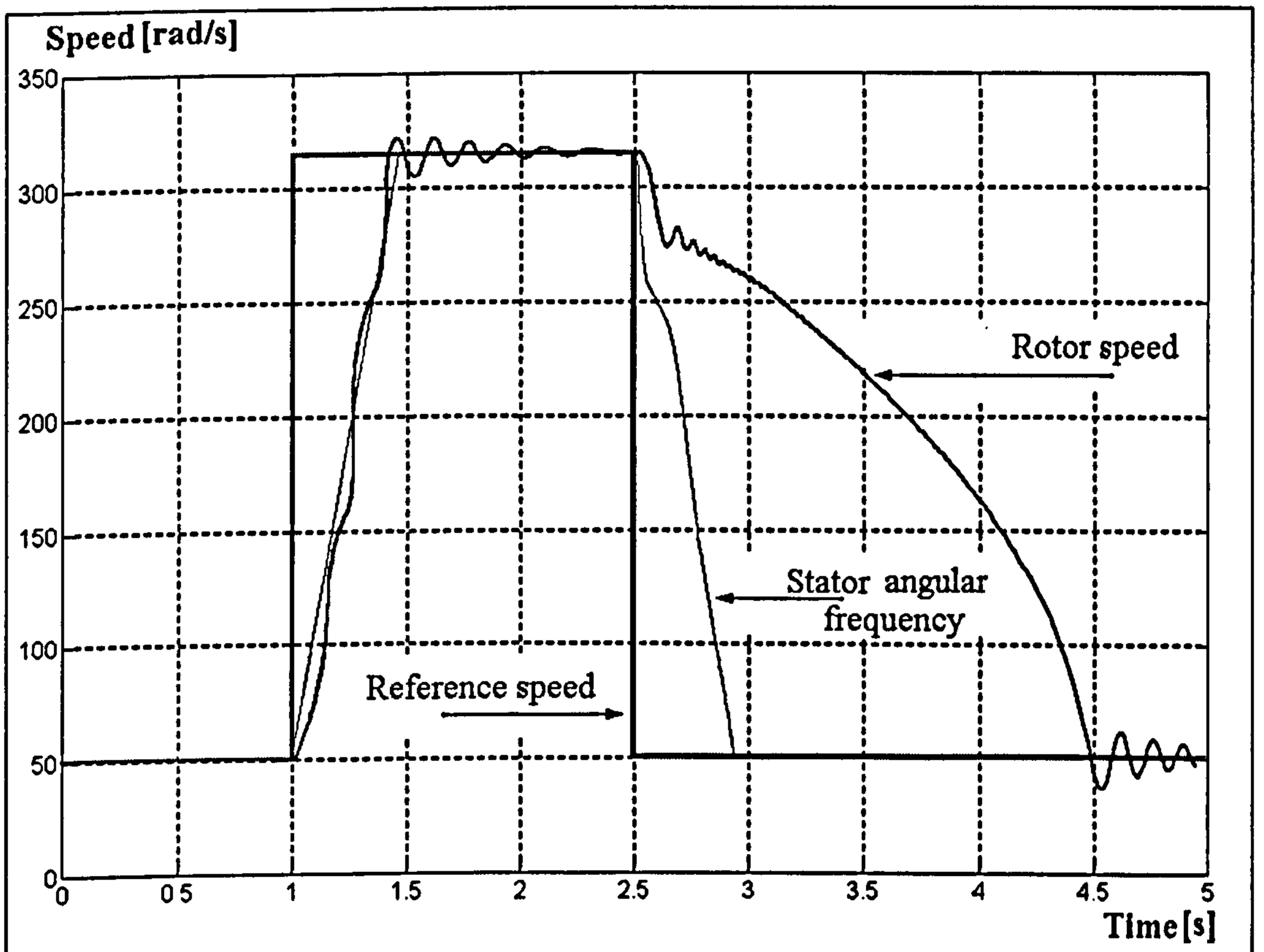

Fig. 4-36 - Quasi-field oriented control method results ($K_{\omega 1}$=3000 s^{-1})

The speed control strategy can be further refined by using different values for the constant K_I depending on the stator current amplitude I_s. A single value K_I cannot be optimal for all the motor currents in the range ($I_{s\text{-}min}$; $I_{s\text{-}max}$) because, as demonstrated by equation (4-150), the motor torque is proportional to I_s squared, and the same derivative

dI_s/dt produces different torque variations at different stator current amplitudes. The effect is a slow dynamic response of the motor when the current is close to $I_{s\text{-}min}$ and a very fast one when the current is close to $I_{s\text{-}max}$. Thus, to optimise the motor response, an improved function F_I needs to be found, which ensures the same dynamic parameters both at small stator currents and at large stator currents. This requires that the motor derivative does not depend on the stator current amplitude. The time derivative of the torque is:

$$\frac{dT}{dt}=\frac{\partial T(I_s,\omega_{slp})}{\partial I_s}\cdot\frac{dI_s}{dt}=\frac{2\omega_{slp}L_m^2 I_s R_r}{R_r^2+\omega_{slp}^2 L_r^2}\cdot\frac{dI_s}{dt} \tag{4-171}$$

Consequently, the torque derivative dT/dt is independent of the current amplitude I_s if dI_s/dt is inversely proportional to I_s. To include this improvement, the quasi-field oriented control strategy initially formulated in (4-170) can be is transformed into

$$\begin{cases}\dfrac{dI_s}{dt}=F_I\left(I_s,\beta_{eqv}\right)=\begin{cases}K_I\cdot\beta_{eqv}/I_s & \text{if} \quad I_s\in\left(I_{s-min};I_{s-max}\right)\\ 0 & \text{if} \quad I_s\notin\left(I_{s-min};I_{s-max}\right)\end{cases}\\ \dfrac{d\omega_{es}}{dt}=F_\omega\left(\omega_{es}^{ref},\omega_{es},I_s,\beta_{eqv}\right)=\begin{cases}\text{sign}\left(\omega_{es}^{ref}-\omega_{es}\right)\cdot\left(K_{\omega 1}-\beta_{eqv}\cdot K_{\omega 2}\right) & \text{if}\ \beta_{eqv}<\beta_{max}\\ \text{sign}\left(\omega_{es}^{ref}-\omega_{es}\right)\cdot\left(K_{\omega 1}-\beta_{max}\cdot K_{\omega 2}\right) & \text{if}\ \beta_{eqv}\geq\beta_{max}\end{cases}\\ \omega_{es}^{ref}=\Omega_{slp}+\omega_{er}^{ref}=\Omega_{slp}+p\cdot\omega_r^{ref}\end{cases} \tag{4-172}$$

In case the limited hardware resources available do not allow the implementation of a supplementary division block (it consumes a significant amount of chip area), the division by I_s can be replaced by a stepwise approximation. Therefore, the unique constant K_I is actually replaced by a stepwise approximation that uses a set of different constants K_{I1}, K_{I2}, K_{I3}, ... depending on the value of I_s. In this case, the parameters of the electrical drive dynamic response depend on the quality of the approximation, which in turn depends on the amount of available hardware resources. The functions (4-172) represent the enhanced version of the new sensorless speed control algorithm proposed in this thesis.

4.4 THE COMPLETE CONTROL SCHEME

The complete sensorless induction motor control scheme generated in this chapter includes a speed controller that operates according to (4-172), a current controller that implements the new method described in section 4.2, and a conversion block that interfaces the two controllers (Fig. 4-37). The conversion block transforms the quantities ω_{es}^{ref} and I_s^{ref} into the reference current $\underline{i}_s$ for the current controller.

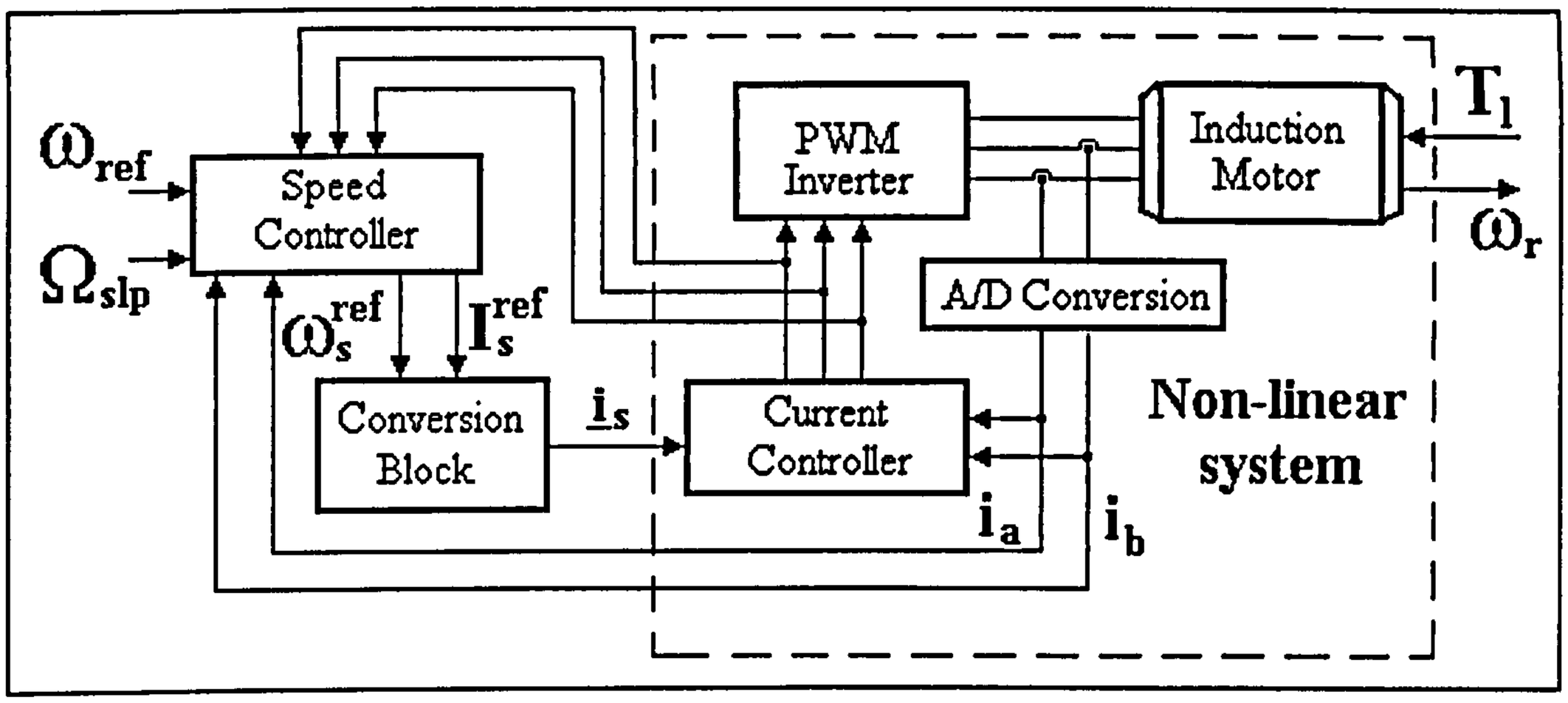

Fig. 4-37 - The Block Diagram of the Sensorless Control Scheme

The current control principles formulated represent a generalisation of the method presented in [106] that leads to superior control performance. The new method requires a big computational effort that can only be performed with the aid of hardware implemented neural networks. The combined effect of the speed controller non-linearity and the slip estimation errors during the transients are very difficult to analyse mathematically but the overall system behaviour can be studied using computer simulations. MATLAB simulations presented prove that the drive system operates without significant speed oscillations, without stationary errors and with good dynamic performance.

5. THE FPGA NEURAL CONTROL APPROACH

This chapter describes the new strategy of implementing neural networks into digital hardware using logic gates and determines the resulting implementation complexity to prove its superiority as compared to relevant results previously presented in the literature. The strategy is illustrated by a complete implementation example: the neural network controlling the current through the stator windings of the induction motor. Experimental results are presented to demonstrate the validity of the adopted design and implementation principles.

5.1 THE NEURAL NETWORK DESIGN AND IMPLEMENTATION STRATEGY

The FFANN design and implementation manner adopted in this thesis is adapted to applications that require high operation speed, accurate control over the network outputs, low cost digital hardware and fast prototyping. FPGA chips are ideal for fast prototyping but the low cost versions still have a limited number of available logic gates. Therefore, the amount of required hardware resources needs to be minimised by optimising the number of neurones and by a compact implementation of each neurone. The classical FFANN design method using neurones with sigmoidal activation function and the back-propagation training algorithm is not appropriate in this context because the resulting number of neurones is large and the sigmoidal activation function requires a considerable amount of hardware resources for implementation. Therefore, neural networks designed with the constructive Voronoi algorithm and consisting of neurones with step activation functions were used instead. The constructive algorithm ensures the minimisation of the neurone number, while the step activation function simplifies the implementation size of each neurone.

5.1.1 General Implementation Principles

The hardware resources offered by FPGA chips are limited to logic gates and flip-flops. The implementation strategy developed in this thesis uses exclusively logic gates to transform any FFANN into a digital hardware structure. The strategy exploits the equivalence between the operation of logic gates and the operation of particular types of neurones. N-input AND gates and n-input OR gates are assimilated to n-input unipolar binary neurones (the input and output values can only be '0' or '1') having positive input weights. The difference between the two logic gate types consists in the relationship between their input weights and the threshold level.

An OR gate output is activated whenever at least one of the inputs is active (is '1'). Thus, the threshold level of the corresponding neurone is positive, but lower than the smallest input weight, as illustrated by (5-1).

$$0 < t_{OR} \leq \min_{i}\{w_i\} \tag{5-1}$$

The output of an n-input AND logic gate is activated only when all the 'n' inputs are active. Therefore, the threshold level can be in this case as large as the total sum of all the input weights. However, it cannot be higher than this sum because otherwise the output cannot be activated in any conditions at all.

$$t_{AND} \leq \sum_{i=1}^{n} w_i \tag{5-2}$$

On the other hand, the threshold level of the corresponding neurone has to be higher than the total sum of any combination of 'n-1' input weights. This last condition is expressed by relation (5-3).

$$t_{AND} > -\min\{w_i\} + \sum_{i=1}^{n} w_i \tag{5-3}$$

As a result, the threshold levels for the two sorts of neurones are confined within the interval limits shown in (5-4). Conversely, any neurone with binary input signals ('0' and '1') whose parameters comply with one of two conditions (5-4), behaves either as an AND gate or as an OR gate.

$$\begin{cases} t_{AND} \in \left(-\min_i\{w_i\} + \sum_{i=1}^{n} w_i ; \sum_{i=1}^{n} w_i \right] & \text{(a)} \\ t_{OR} \in \left(0 ; \min_i\{w_i\} \right] & \text{(b)} \\ w_i > 0 \forall i = 1,2,...n \end{cases} \qquad \text{(5-4)}$$

Neurones whose parameters do not comply with any of the two relations (5-4) can be implemented as a configuration containing several interconnected logic gates. The details of the hardware configuration depend on the relationship between the input weights and the threshold level. The number of necessary gates increases with the complexity of this relationship. To simplify the logical analysis, the adopted implementation strategy decomposes the complex neurones into a pyramidal structure of simpler subneurones. Each subneurone can be further decomposed into higher-order subneurones until each of them can be implemented with a small number of logic gates.

As explained in chapter 3, the Voronoi algorithm produces a FFANN with up to three layers of neurones with step activation functions. The algorithm version that produces unipolar neurones is adopted because unipolar neurones are more adequate for hardware implementation than bipolar neurones. The network accepts analogue input signals but generates digital output signals. The neurones in the input layer have analogue inputs and binary outputs, while the rest of the neurones operate only with binary signals. Therefore, the neurones in layers two and three are appropriate for direct digital hardware implementation. The neurones in the first layer need to be converted first into a digital form that uses bit patterns as inputs instead of analogue signals.

The most appropriate binary codification to be used for neurone input quantities is the complementary code (also named "two's complement" and symbolised by C_2). It is very largely used in computer technology for integer number representations, but it can be readily adapted for real values in the interval [-1; +1).

Considering a n-bit representation "$b_{n-1}b_{n-2}b_{n-3}....b_1b_0$", the corresponding integer value (I_n) is given by:

$$I_n = -2^{n-1} \cdot b_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot b_i \qquad \text{(5-5)}$$

The largest positive number, which can be represented on 'n' bits, is 2^{n-1}-1 while the smallest number is -2^{n-1}. Real values between -1.0 and +1.0 can be represented dividing all the integer values I_n by 2^{n-1}. Thus, equation (5-6) illustrates the complementary code extended to real numbers:

$$R_n = \frac{I_n}{2^{n-1}} = -b_{n-1} + \sum_{i=0}^{n-2} 2^{-n+1+i} \cdot b_i \qquad (5\text{-}6)$$

The large-scale utilisation of complementary code in digital technology is due to the advantages of simple hardware implementation of addition and subtraction. A hardware implemented neural control system contains not only neural networks but also traditional digital structures. Therefore, the use of the same codification manner for the two modules is an important advantage because it simplifies the interface between them.

Thus, the new implementation strategy consists of two parts. In the first phase, the initial FFANN mathematical model is digitised, so that the neurones in the input layer operate only with binary signals. The input signals of the converted FFANN consist of bit strings coding the values of the initial analogue inputs. In the second phase, all the neurones are converted into a set of interconnected logic gates. The implementation into logic gate structures is performed neurone by neurone. Each neurone corresponds to a hardware configuration containing at least one logic gate.

5.1.2 Model Digitisation

The equations underlying the conversion of the analogue neurones into equivalent digital neurones can be demonstrated by decomposing this process in two successive stages. The first stage is to replace the analogue input signals by binary patterns. The second stage brings additional corrections to the neurone mathematical model, so that the resulting neurones use the complementary code extended to real numbers described by (5-6).

The principles underlying the digitisation process involve two basic concepts: the codification style and the neurone behaviour. The codification style, illustrated in Fig. 5-1, is defined as the correspondence between the initial analogue input signals and the binary input codes used by the digital neurone. On the other hand, the neurone behaviour is described by the relationship between the analogue inputs and the neurone output signal. The initial neurone behaviour has to be maintained unchanged during the two stages of the digitisation process. To achieve this, the neurone parameters (input weights and the threshold levels) need to be modified at each conversion stage, in a manner that counteracts the effects of replacing the analogue input signals with binary patterns.

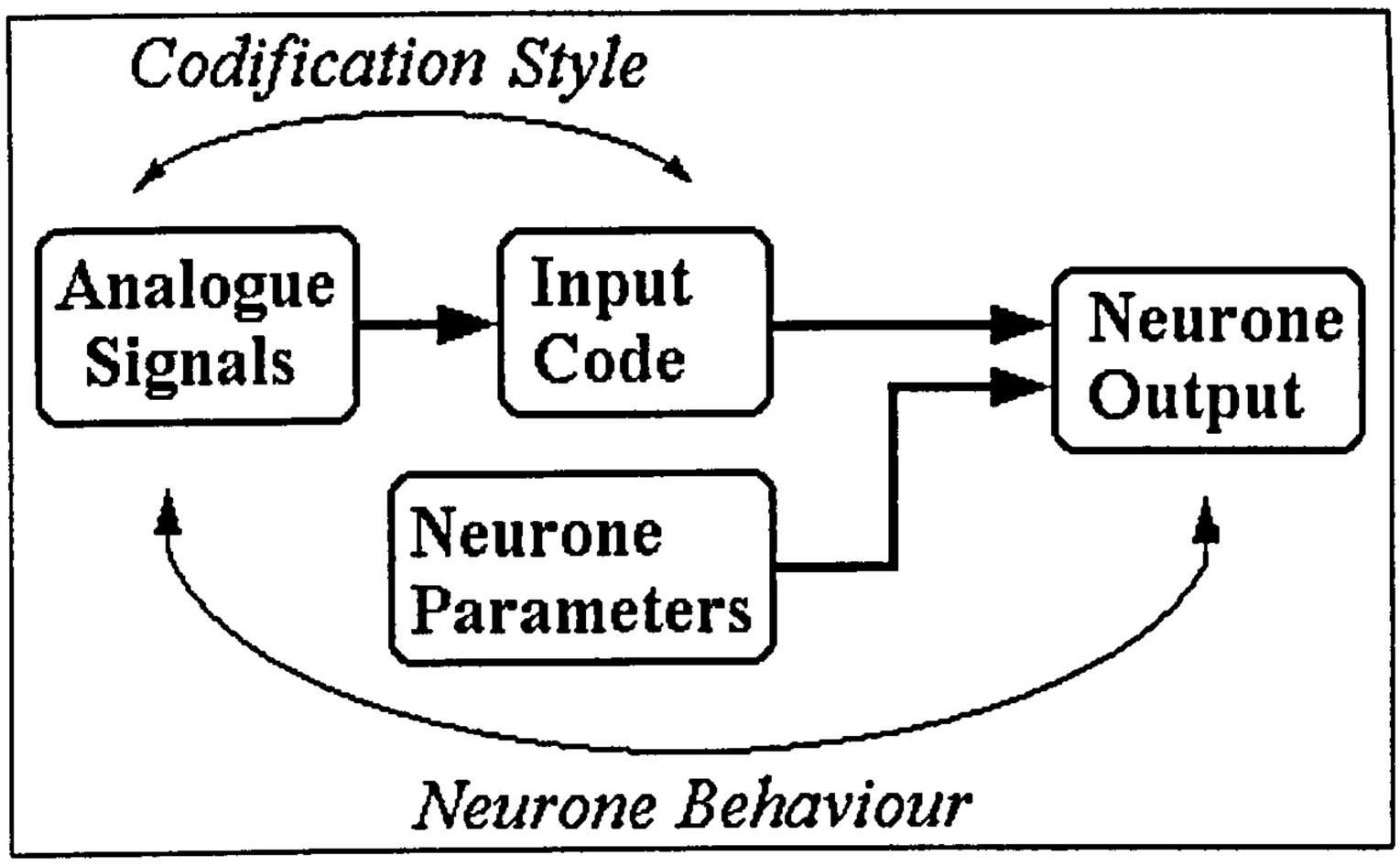

Fig. 5-1 – Basic concepts related to the neurone digitisation process

The minimal condition to attain this aim is to perform the changes such that the sign of the activation function argument is kept constant. This principle is expressed by equation

$$\operatorname{sign}\left(\sum_{i=1}^{m} w_i \cdot x_i - t\right) = \operatorname{sign}(net - t) = constant \tag{5-7}$$

However, for reasons of mathematical simplicity, a more restrictive condition is used instead, namely the argument "net-t" of the activation function is kept itself constant rather than only the sign of it:

$$\sum_{i=1}^{m} w_i \cdot x_i - t = net - t = constant \tag{5-8}$$

5.1.2.1 Conversion Stage One

The first step, illustrated in Fig. 5-2, transforms the analogue neurones generated by means of Voronoi algorithm into digital neurones. The newly obtained neurones receive binary patterns on their inputs instead of analogue signals. The task is achieved by keeping the threshold level unchanged while splitting each input defined by its initial weight w_{ij} into n_b subinputs, whose weights w_{ijp} ($p=0,1,\ldots n_b-1$) are calculated as follows:

$$\begin{cases} w_{ijp}^{(1)} = \dfrac{2^{p+1}}{2^{n_b}} \cdot w_{ij} \; \forall p < n_b - 1 \\ w_{ij(n_b-1)}^{(1)} = -w_{ij} \\ t_i^{(1)} = t_i \end{cases} \tag{5-9}$$

The superscript '(1)' in equations (5-9) shows that the corresponding quantities have been calculated during the first conversion stage. Likewise, the superscript '(2)' identifies the quantities calculated during the second conversion stage.

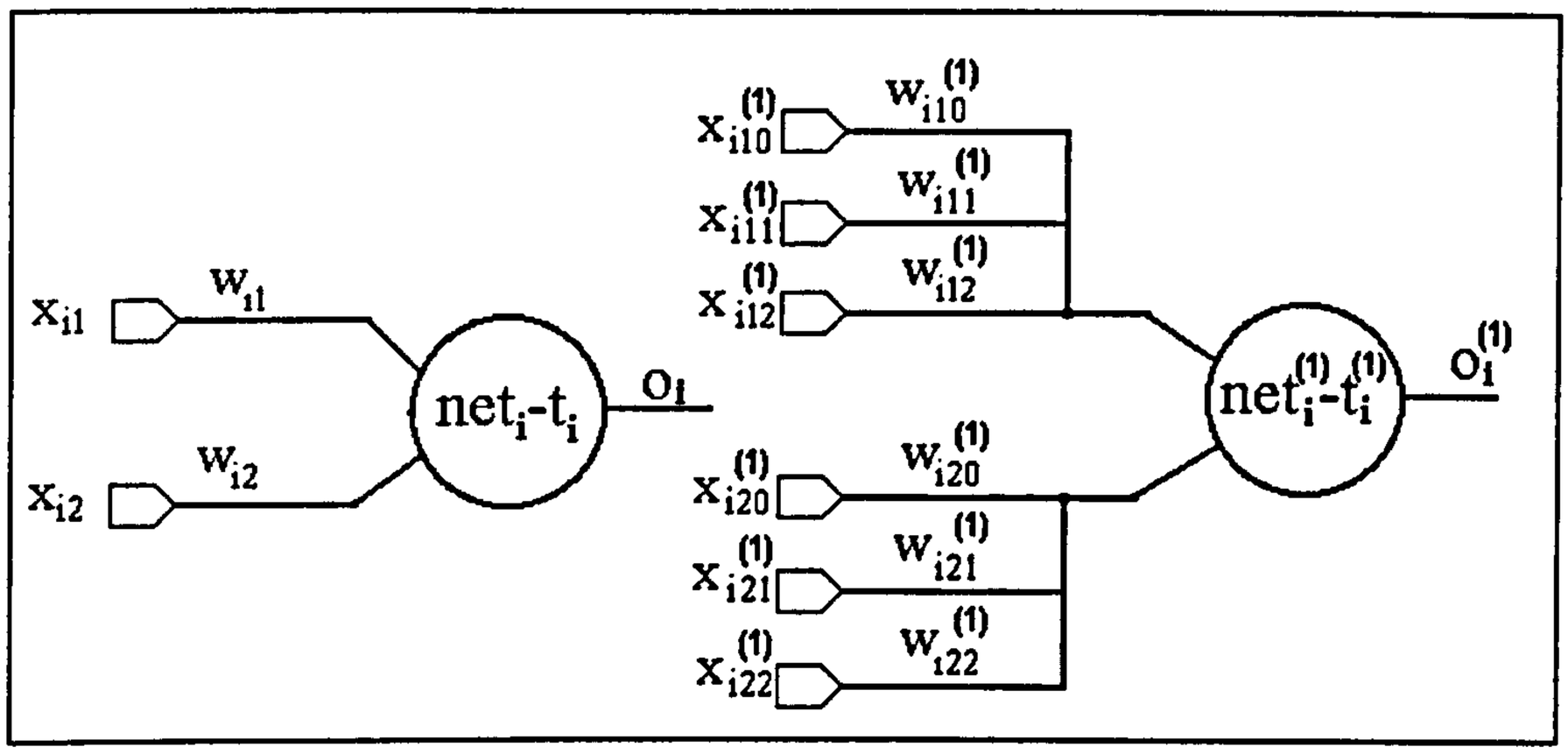

Fig. 5-2- The neurone model before and after stage one of the conversion

The result of the previous calculations is that the initial 'm' inputs are turned into 'm' input clusters, each cluster containing 'n_b' subinputs. The symbol 'w_{ij}' stands for the weight number 'j' of the neurone 'i' in the network, while '$w_{ijp}^{(1)}$' represents the weight of subinput 'p' in cluster 'j' pertaining to neurone 'i'. The index p=0 corresponds to the least significant binary figure, while $p=n_b-1$ corresponds to the most significant one.

According to the previous considerations, only those neurone parameter changes that maintain argument "net_i-t_i" of the activation function constant are allowed. The argument corresponding to the neurone after the first conversion stage is calculated as

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{jp}^{(1)} - t_i^{(1)} = \sum_{j=1}^{m}\left(-w_{ij}\cdot x_{jp}^{(1)} + \sum_{p=0}^{n_b-2} w_{ij}\cdot\frac{2^{p+1}}{2^{n_b}}\cdot x_{jp}^{(1)}\right) - t_i^{(1)} \quad (5\text{-}10)$$

where $x_{jp}^{(1)}$ $(p=0,1,2,\ldots n_b-1)$ are the bits of the binary code received by each new neurone input.

Equation (5-10) can be transformed into

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m} w_{ij}\cdot\left(-x_{j(n_b-1)}^{(1)} + \sum_{p=0}^{n_b-2} 2^{-n_b+p+1}\cdot x_{jp}^{(1)}\right) - t_i^{(1)} \quad (5\text{-}11)$$

The expression between parentheses corresponds to the extended complementary code definition given in equation (5-6). Therefore, (5-11) is further transformed into

$$net_i^{(1)} - t_i^{(1)} = \sum_{j=1}^{m} w_{ij}\cdot x_j - t_i^{(1)} = \sum_{j=1}^{m} w_{ij}\cdot x_j - t_i = net_i - t_i \quad (5\text{-}12)$$

where x_j is an analogue input value of the initial neurone. This proves that the condition expressed by (5-7) is fulfilled. Thus, during conversion stage one the codification style based on the complementary code has been introduced and the required modifications of the neurone parameters have been performed so that the neurone behaviour has been maintained unchanged.

5.1.2.2 Conversion Stage Two

The conversion of the neural network into logic gate architecture is based on the relations (5-4) and on the possibility to transform any neurone into an equivalent structure containing interconnected elements that comply with (5-4). Such transformations are possible only if all the neurone weights are positive. The stage one neurones may have both positive and negative weights. The second conversion stage aims to replace these neurones with equivalent ones having only positive weights. The simplest way to eliminate negative input weights is to use only the module of their values. Consequently, the relationship between stage one neurone weights and their stage two counterparts is expressed by

$$w_{ijp}^{(2)} = \left| w_{ijp}^{(1)} \right| \tag{5-13}$$

Adopting this method means that supplementary parameter alterations are required in order to counteract the neurone behaviour alteration which is caused by changing the sign of some input weights. As the weight values have already been changed according to (5-13), the neurone behaviour can be corrected by changing the threshold level and/or the codification style.

It can be demonstrated that no change of the threshold level can counteract the effect of the input weight alterations. Thus, the change of the threshold level needs to be carried out in such a manner that equation

$$net_i^{(2)} - t_i^{(2)} = \sum_{j=1}^{m} \sum_{p=0}^{n_b-1} \left| w_{ijp}^{(1)} \right| \cdot x_{ijp}^{(2)} - t_i^{(2)} = \sum_{j=1}^{m} \sum_{p=0}^{n_b-1} w_{ijp}^{(1)} \cdot x_{ijp}^{(1)} - t_i^{(1)} = net_i^{(1)} - t_i^{(1)} \tag{5-14}$$

is fulfilled for any input bits x_{ijp}. However, if the input signals to the stage two neurones are the same as the inputs to stage one neurones ($x_{ijp}^{(2)} = x_{ijp}^{(1)}$), then there is no constant value $t_i^{(2)}$ that allows (5-14) to be valid for any combination of input signals. To prove this, the value of $t_i^{(2)}$ can be calculated as

$$t_i^{(2)} = t_i^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left(\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}\right)\cdot x_{ijp}^{(1)} \qquad (5\text{-}15)$$

which is derived from equation (5-14). The value calculated according to (5-15) is dependent on the input bits $x_{ijp}^{(1)}$ and therefore is not a constant as the threshold level should be.

Equation (5-15) demonstrates that no acceptable solution exists when the codification style of stage one neurone is identical to the codification style of stage two neurone. Therefore, the codification style needs to be altered as well. A simple solution to this problem can be found if the input bits corresponding to negative input weights at stage one neurones are reversed at stage two neurones. The modification can be readily implemented into hardware with NOT logic gates as shown in Fig. 5-3.

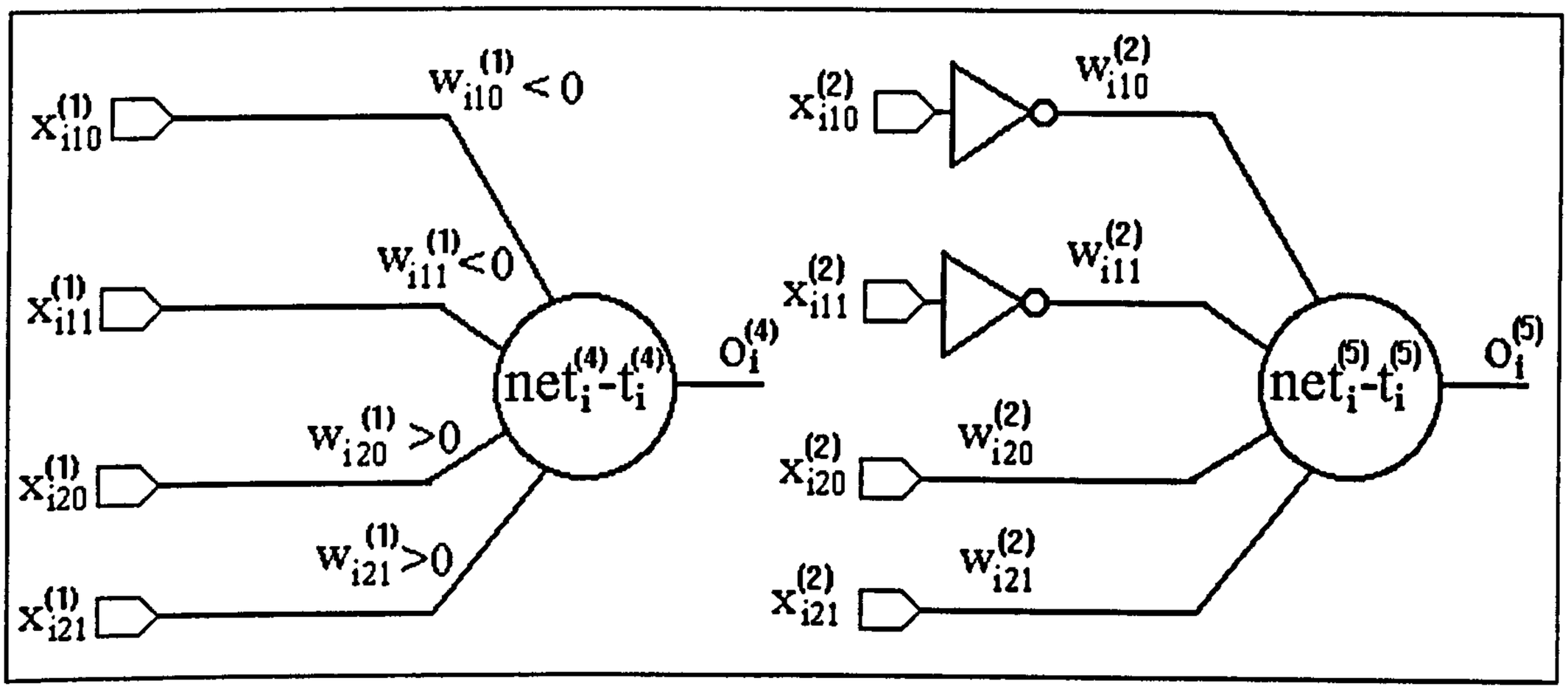

Fig. 5-3 - The neurone "i" before and after stage two of the conversion

The relationship between the input bits of stage two neurones and stage one neurones is expressed by function

$$x_{ijp}^{(2)} = \begin{cases} x_{ijp}^{(1)} \text{ if } w_{ijp}^{(1)} > 0 \\ 1 - x_{ijp}^{(1)} \text{ if } w_{ijp}^{(1)} < 0 \end{cases} \qquad (5\text{-}16)$$

The two situations in (5-16) can be compressed into equation

$$x_{ijp}^{(2)} = \frac{1 - \text{sign}\left(w_{ijp}^{(1)}\right)}{2} + \text{sign}\left(w_{ijp}^{(1)}\right)\cdot x_{ijp}^{(1)} \qquad (5\text{-}17)$$

where the 'sign' function is defined by

$$\text{sign}(x) = \begin{cases} +1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \qquad (5\text{-}18)$$

Using (5-13) and (5-17), the argument of the transfer function for stage two neurones can be calculated as

$$net_i^{(2)} - t_i^{(2)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left[\frac{\left|w_{ijp}^{(1)}\right| - sign\left(w_{ijp}^{(1)}\right)\cdot\left|w_{ijp}^{(1)}\right|}{2} + sign\left(w_{ijp}^{(1)}\right)\cdot\left|w_{ijp}^{(1)}\right|\cdot x_{ijp}^{(1)}\right] - t_i^{(2)} \qquad (5\text{-}19)$$

and

$$net_i^{(2)} - t_i^{(2)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)}\cdot x_{ijp}^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left(\frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2}\right) - t_i^{(2)} \qquad (5\text{-}20)$$

Given the requirement of equality between the two activation function arguments, the threshold level can be calculated based on equation

$$\sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)}\cdot x_{ijp}^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\left(\frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2}\right) - t_i^{(2)} = \sum_{j=1}^{m}\sum_{p=0}^{n_b-1} w_{ijp}^{(1)}\cdot x_{ijp}^{(1)} - t_i^{(1)} \qquad (5\text{-}21)$$

Therefore, the result is

$$t_i^{(2)} = t_i^{(1)} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\frac{\left|w_{ijp}^{(1)}\right| - w_{ijp}^{(1)}}{2} \qquad (5\text{-}22)$$

The threshold level $t_i^{(2)}$ is constant in equation (5-22) because it depends exclusively on constant quantities. The parameters of stage one neurones depend on the initial parameters as described by (5-9). Consequently, (5-22) can be successively transformed as:

$$t_i^{(2)} = t_i + \sum_{j=1}^{m}\frac{\left|w_{ij}\right| + w_{ij}}{2} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-2}\frac{\frac{2^{p+1}}{2^{n_b}}\cdot\left|w_{ij}\right| - \frac{2^{p+1}}{2^{n_b}}\cdot w_{ij}}{2} \qquad (5\text{-}23)$$

$$t_i^{(2)} = t_i + \sum_{j=1}^{m}\frac{\left|w_{ij}\right| + w_{ij}}{2} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-2}\frac{2^p}{2^{n_b}}\cdot\left(\left|w_{ij}\right| - w_{ij}\right) \qquad (5\text{-}24)$$

$$t_i^{(2)} = t_i + \sum_{j=1}^{m}\frac{\left|w_{ij}\right| - w_{ij}}{2} + \sum_{j=1}^{m}\sum_{p=0}^{n_b-2}\frac{2^p}{2^{n_b}}\cdot\left(\left|w_{ij}\right| - w_{ij}\right) + \sum_{j=1}^{m} w_{ij} \qquad (5\text{-}25)$$

$$t_i^{(2)} = t_i + \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\frac{2^p}{2^{n_b}}\cdot\left|w_{ij}\right| - \sum_{j=1}^{m}\sum_{p=0}^{n_b-1}\frac{2^p}{2^{n_b}}\cdot w_{ij} + \sum_{j=1}^{m} w_{ij} \qquad (5\text{-}26)$$

$$t_i^{(2)} = t_i + \frac{2^{n_b}-1}{2^{n_b}} \cdot \sum_{j=1}^{m} |w_{ij}| - \frac{2^{n_b}-1}{2^{n_b}} \cdot \sum_{j=1}^{m} w_{ij} + \sum_{j=1}^{m} w_{ij} \quad (5\text{-}27)$$

$$t_i^{(2)} = t_i + \left(1 - 2^{-n_b}\right) \cdot \sum_{j=1}^{m} |w_{ij}| + 2^{-n_b} \cdot \sum_{j=1}^{m} w_{ij} \quad (5\text{-}28)$$

Thus, the parameters of the final digital neurones can be calculated as a function of the initial analogue neurone parameters by combining (5-28) with (5-13) and (5-9), the result being

$$\begin{cases} w_{ijp}^{(2)} = \frac{2^{p+1}}{2^{n_b}} \cdot |w_{ij}| \; p = 0,1,2,..n_b - 1 \\ t_i^{(2)} = t_i + \left(1 - 2^{-n_b}\right) \cdot \sum_{j=1}^{m} |w_{ij}| + 2^{-n_b} \cdot \sum_{j=1}^{m} w_{ij} \end{cases} \quad (5\text{-}29)$$

As shown in Fig. 5-4, the final implementation solution uses a codification style that involves two binary codes. The first one is the complementary code. This code is transformed by a set of NOT gates into the second code, which is directly used by the neurone obtained after the second conversion stage. This neurone model has only positive input weights so that it can be transformed into a digital structure containing exclusively AND logic gates and OR logic gates.

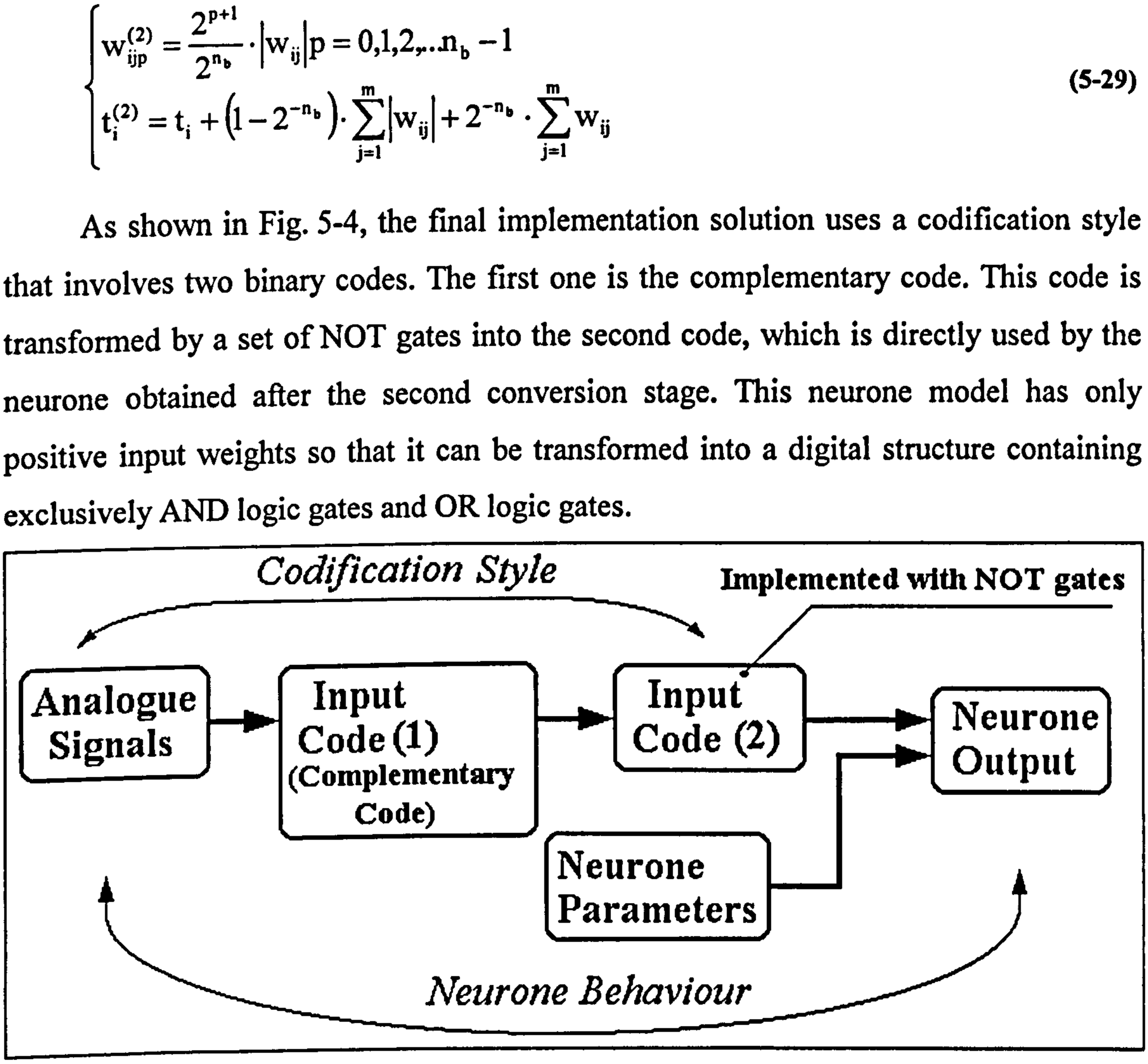

Fig. 5-4 – Neurone conversion solution

5.1.3 Digital Model Implementation Using Logic Gates

The FFANN implementation into a hardware structure is performed separately for each neurone. The implementation method requires that at first the array of input weights $w_{ijp}^{(2)}$ is sorted in descending order. The sorted array contains a total number of $A=m \times n_b$ elements $(w_1^s, w_2^s, w_3^s ... w_A^s)$, where 'm' is the number of analogue input

signals and n_b is the number of bits used for each input code. The sorted weights correspond to the input signals $x_1^s, x_2^s, \ldots x_A^s$. An iterative conversion procedure is used to analyse the input weights and to generate the corresponding netlist description of the logic gate implementation. As mentioned in section 5.1.1, the iterative procedure decomposes the initial neurone into a pyramidal structure of interconnected subneurones. The structure comprises a top subneurone, a layer of first-order subneurones, a layer of second-order subneurones, etc. The subneurones have all the properties of normal neurones, but they have fewer inputs than the initial neurone. Some subneurones are implementable by very simple logic gates configurations. The rest are further decomposed into second-order and third-order subneurones until all them are implemented.

5.1.3.1 Preliminary Considerations

A series of interrelated basic concepts need to be defined before describing the iterative hardware implementation process: **terminal weight group, group threshold level, dominant weight, cumulated weight, critical weight, non-critical weight, significant weight, insignificant weight.**

A **terminal weight group** (or simply a **terminal group**) is a set of weights comprising the last N consecutive elements in the sorted array. Therefore any **terminal weight group** can be uniquely identified by the symbol $G_t(F)$ where 'F' is the index of its first element. There are a number of A overlapping **terminal weight groups** in the sorted array: $G_t(1)$, $G_t(2)$, $G_t(3)$, ...,$G_t(A)$. **Terminal weight group** $G_t(1)$ encompasses all the weights in the array. The weights of each first-order subneurone generated by the iterative implementation algorithm are the weights of a **terminal group**. However, not any terminal group generates a first-order subneurone in the final implementation. Thus, the number of first-order subneurones in the pyramidal logic gate structure is situated in the interval [0; A].

The **group threshold level** T_t is a quantity calculated by the conversion algorithm for each **terminal group of weights** that is to be converted into a subneurone. The **group threshold level** equals the threshold level of the subneurone to be generated. The same terminal group can be analysed by the implementation algorithm more than once in different contexts. Each time it can be associated with a different threshold level.

If a weight value is larger than the **group threshold level,** then it is named a **dominant weight** of the corresponding **terminal group. Any dominant weight** is

related to a **dominant input** that, if active, is able to activate the neurone output signal (force it to '1'), even if all the other input signals are inactive ('0'). The **dominant weights** in a subneurone are always the first in the corresponding **terminal group,** because the initial array of weights was sorted in descending order. Consequently, the number D of dominant inputs can be determined using condition (5-30), and if the largest weight in a **terminal group** is not dominant, no weight is dominant in that **terminal group.**

$$\begin{cases} w^s_{F+i} \geq T_t & \forall\, 0 \leq i < D \\ w^s_{F+i} < T_t & \forall\, i \geq D \end{cases} \tag{5-30}$$

The **cumulated weight** of a terminal group $G_t(F)$ is defined as the sum of all its component weights. The cumulated weight equals the 'net' value of the neurone when all its inputs are active ('1') in the same time. This is the maximum 'net' value of the corresponding subneurone. If the **cumulated weight** is smaller than the **group threshold level,** then the subneurone output is always inactive, regardless of the input signals.

$$W_t(F) = \sum_{i=F}^{A} w^s_i = \max\left\{ \sum_{i=F}^{A} w^s_i \cdot x^s_i \right\} \tag{5-31}$$

The output of a subneurone can be activated either by **dominant inputs** or, if no **dominant input** is active, by combinations of several non-dominant inputs. Some of these non-dominant inputs are included in all the combinations capable of activating the output. They are named **critical inputs** and they correspond to **critical weights.** Activating these inputs does not necessarily ensure that the subneurone output is active. They only bring the 'net' value of the subneurone close to the **group threshold,** so that the output can be activated in conjunction with less important input signals (the importance of an input signal is proportional to its corresponding weight). As the initial array was sorted in descending order, the **critical weights** always follow the **dominant weights** in any terminal group.

Thus, the **critical weights** can be determined by subtracting all the **dominant weights** from the **cumulated weight.** The result has to be larger than the **group threshold.** Each of the remaining weights are then subtracted from the previous result, obtaining a series of increasing values. Those values that are smaller than the **group threshold level** correspond to **critical weights.** This method is summarised in (5-32)

where D is the number of **dominant weights** and C is the number of **critical weights** in the given **terminal group.**

$$\begin{cases} W_t(F) - \sum_{i=0}^{D-1} w_{F+i}^s > T_t \\ W_t(F) - w_{F+D+j}^s - \sum_{i=0}^{D-1} w_{F+i}^s < T_t \quad \forall\, 0 \le j < C \\ W_t(F) - w_{F+D+j}^s - \sum_{i=0}^{D-1} w_{F+i}^s \ge T_t \quad \forall\, j \ge C \end{cases} \tag{5-32}$$

Thus, if all **dominant inputs** are '0' and at least one of the **critical weights** is '0' in the same time, then the neurone output cannot be active. On the other hand, the subneurone output can be active when all the **dominant inputs** are inactive, but all the **critical inputs** are active.

In some cases, the **critical inputs** are sufficient to activate the neurone output. In other cases, the **critical inputs** can activate the output only in conjunction with certain combinations of less important inputs, because the sum of the **critical weights** is lower than the threshold level. These less important inputs, involved in activating the subneurone output, are named **non-critical inputs** and they correspond to **non-critical weights.** As opposed to **critical inputs,** none of the **non-critical inputs** is essential for the subneurone activation. If a **non-critical input** is inactive, its task can be performed by groups of other **non-critical inputs,** so that the 'net' value is maintained above the threshold level and the subneurone is kept active. However, if all **non-critical inputs** are deactivated in the same time, the subneurone output is deactivated as well. A subneurone with D dominant weights and C critical weights has **non-critical weights** as well, if and only if the conditions (5-33) are fulfilled. These conditions signify that the neurone output can be activated by non-dominant inputs but the task cannot be performed by **critical inputs** alone.

$$\begin{cases} \sum_{i=D}^{A} w_{F+i}^s \ge T_t \\ \sum_{i=0}^{C-1} w_{F+D+i}^s < T_t \end{cases} \tag{5-33}$$

The three previous input categories (**dominant, critical** and **non-critical**) are unequally important for the subneurone operation, but all influence the output signal. These types of inputs have **significant weights. Insignificant inputs** do not influence the subneurone output at all. The **insignificant inputs** have **insignificant weights,**

which are very small and do not affect the relation between the subneurone 'net' value and the **group threshold level,** regardless of the corresponding input signals. Consequently, these inputs are not implemented into hardware.

The effect of sorting the initial array of input weights is that the weights of the same type are grouped together. Furthermore, the groups are arranged in a standard sequence: dominant, critical, non-critical, and insignificant, as illustrated by Fig. 5-5 on the particular case of a neurone with 12 arbitrarily chosen input weights.

One or several weight types can be absent from the sequence. For instance, a neurone complying with condition (5-4)-(a) is implementable with an AND logic gate and has only critical weights, because if one of the AND inputs is '0' (inactive) the logic output is '0' as well. Similarly, the neurones complying with condition (5-4)-(b) are implementable with OR logic gates and have only dominant input weights.

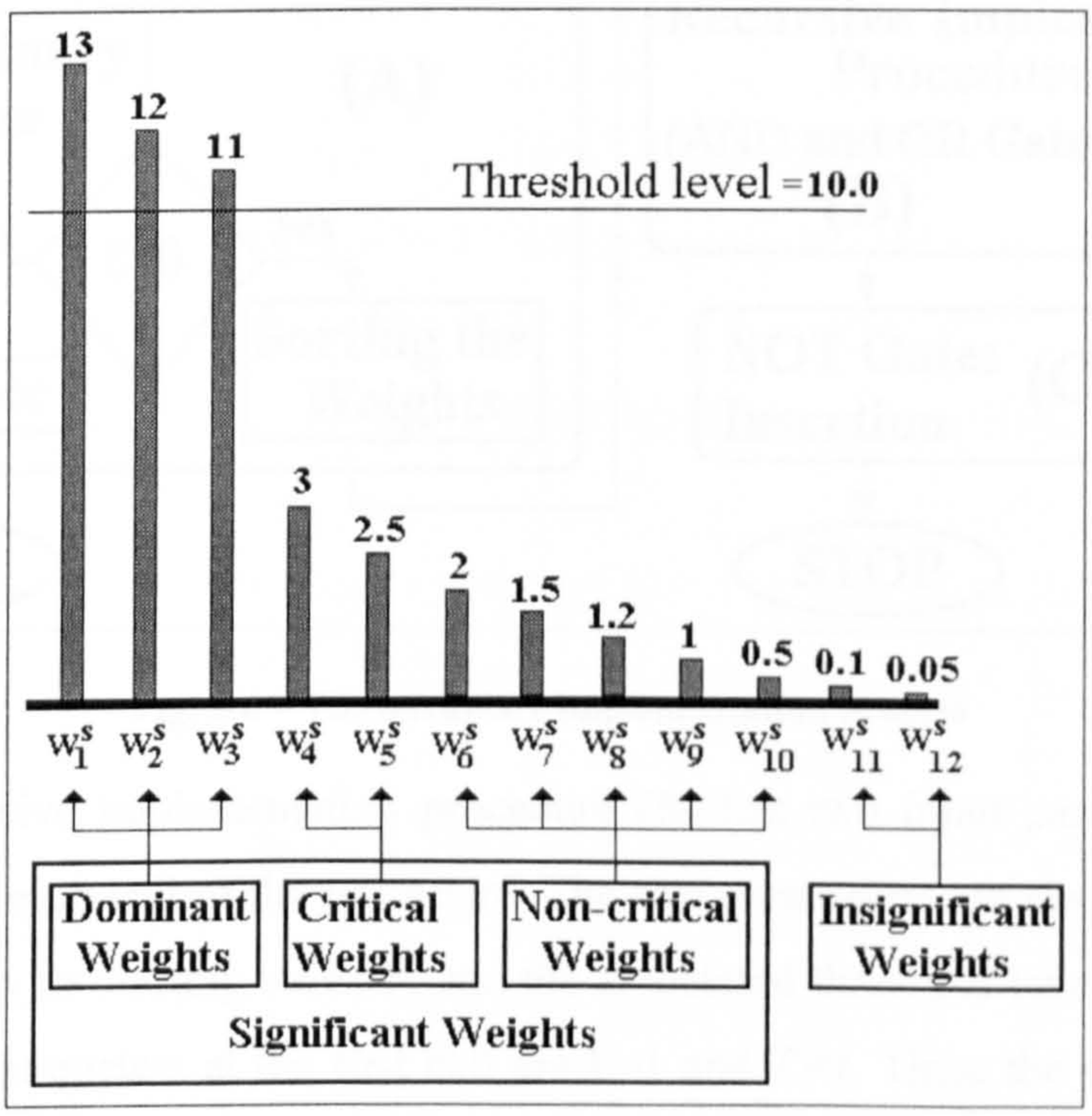

Fig. 5-5 – The neurone weight types and their relative position in the sorted array of weights

5.1.3.2 The Implementation Process – Detailed Description

In this section, the hardware implementation of the digital neurones is described in detail, using the concepts and the formulas from the previous section. The implementation process is divided into three procedures (Fig. 5-6):

(A) The first one carries out a preliminary neurone check. It analyses the sign of its threshold level 't'. If the sign is negative or zero, the neurone output is always

active regardless of the input signals and the neurone implementation is a simple connection between Vcc (+5V) and its output.

(B) If the threshold level is positive, the array of weights is sorted in descending order and then the second procedure is called. This one is a recursive implementation procedure that repeatedly calls itself and builds the required pyramidal structure, gate by gate.

(C) Eventually the third procedure is called which, according to the principles discussed in section 5.1.2 (at conversion stage two), attaches inverter gates to those inputs in the sorted array that correspond to negative weight values at conversion stage one ($w_x^s \Leftrightarrow w_{ijp}^{(1)} < 0$).

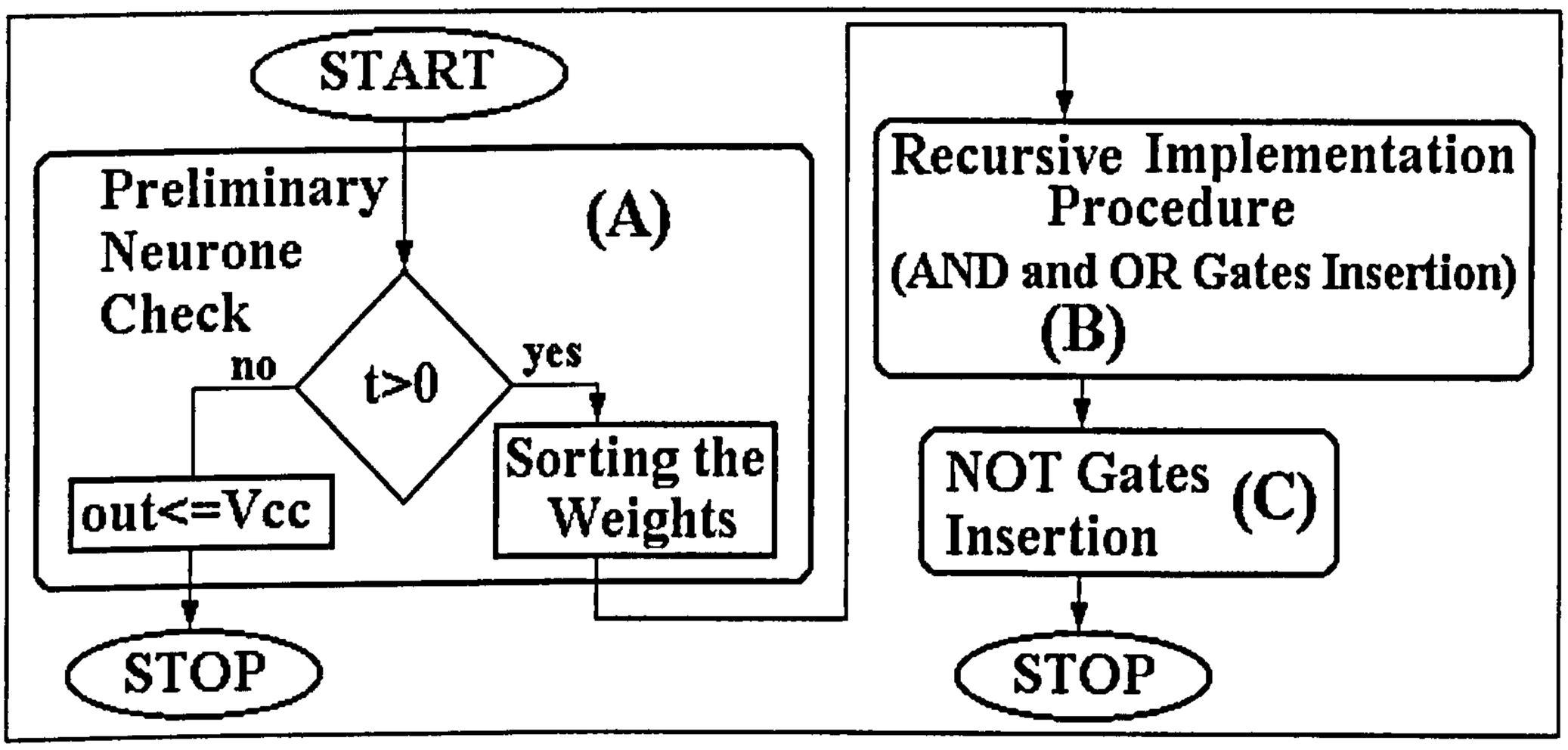

Fig. 5-6 – The hardware implementation process

The recursive implementation procedure (B) has two input parameters that are recalculated for each call of the procedure. The two parameters are the current terminal group defined by its starting index F, and the associated threshold level of the terminal group T_t. The parameters at the first call are F=1 and T_t=t. Thus, the process starts by analysing the terminal group $G_t(F)=G_t(1)$, which comprises all the weights in the array in conjunction with the neurone threshold level 't'. The operation of the recursive implementation procedure can be described in 10 steps.

Step 1) The number D of dominant inputs and the number C of critical inputs are calculated by means of (5-30) and (5-32). Condition (5-33) is used to determine whether the subneurone has non-critical inputs. Table 5-1 presents all the possible situations and the next algorithm step to be performed in each case.

Table 5-1 – Subneurone implementation cases

Dominant inputs	Critical inputs	Non-critical inputs	Next algorithm step
D=0	C=0	N=0	step 2
D>0	C=0	N=0	step 3
D=0	C>0	N=0	step 4
D>0	C>0	N=0	step 5
D=0	C=0	N>0	step 7
D>0	C=0	N>0	step 8
D=0	C>0	N>0	step 6
D>0	C>0	N>0	step 9

Step 2) The neurone has no significant input and therefore its output is always inactive. The hardware implementation reduces to a simple connection between the neurone output and the circuit ground. End of the procedure (B).

Step 3) The subneurone has only dominant inputs and it is implemented as a D-input OR gate. End of the procedure (B).

Step 4) The subneurone has only critical inputs and it is implemented as a D-input AND gate. End of the procedure (B).

Step 5) The subneurone can be activated either by one of the dominant inputs or by all the critical inputs together. Therefore, the current subneurone can be decomposed into a simpler subneurone plus one higher-order subneurone. The first has D+1 dominant inputs and is connected to the D dominant inputs of the initial subneurone, while input D+1 is fed by the second subneurone. The output of the second subneurone is activated only when the initial subneurone is activated due to the critical input signals. Therefore, it is implemented as a C-input AND logic gate. End of the procedure (B).

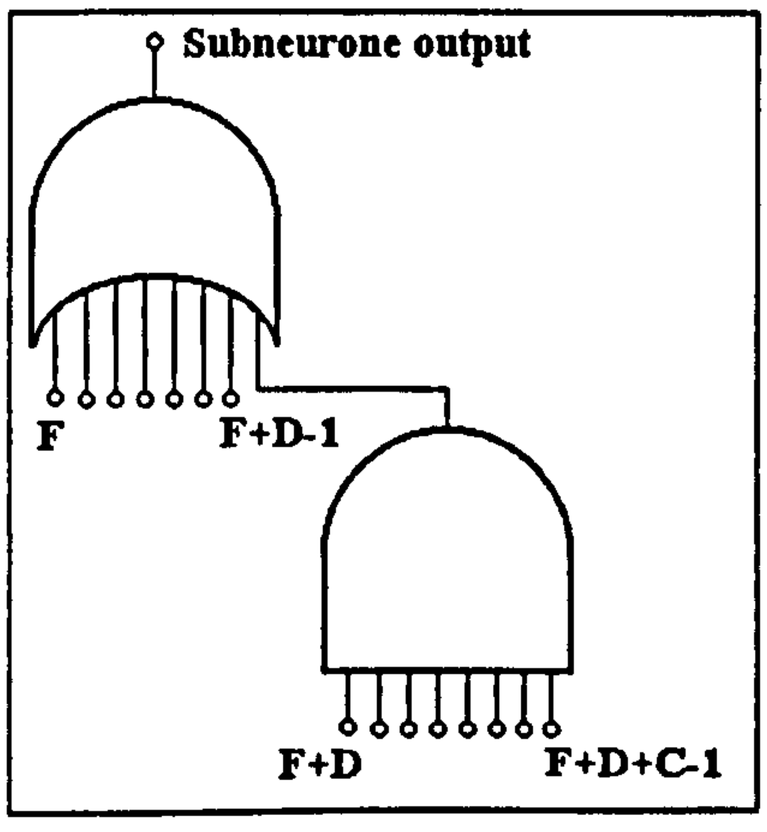

Fig. 5-7 – Subneurone implementation at step 5

Step 6) The subneurone has critical and non-critical inputs. Therefore, the subneurone output is active if all the critical inputs are active simultaneously with certain combinations of non-critical inputs. The subneurone can be decomposed into a higher-order subneurone supplying a simple subneurone implementable as an AND gate with C+1 inputs. The first C gate inputs are connected to the current subneurone critical inputs, while the last input is connected to the output of the higher-order subneurone which analyses the remaining input combinations.

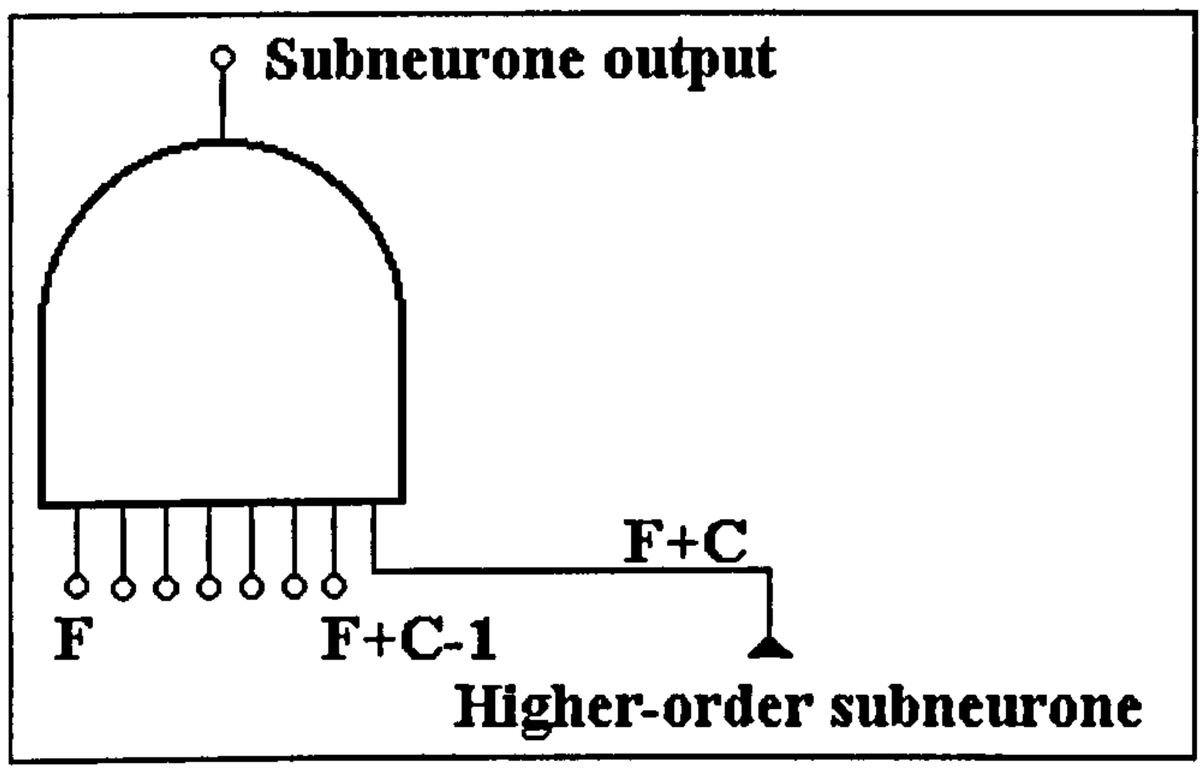

Fig. 5-8 – Subneurone implementation at step 6

The recursive implementation procedure needs to be recalled to generate the implementation of the higher-order subneurone. The new parameters are given in (5-34). The new threshold level is lower than the previous one because the remaining input signals need to cover only the difference between the previous threshold and the sum of the C critical weights already implemented by the AND gate. Go to step 10.

$$\begin{cases} F = F + C \\ T_t = T_t - \sum_{i=0}^{C-1} w_{F+i}^s \end{cases} \tag{5-34}$$

Step 7) The subneurone has only non-critical inputs. Thus, there are several combinations of input signals capable to activate the subneurone output. The combinations are classified into a number of categories. Each category is associated with a terminal group and comprises all the combinations that involve the first input in the given terminal group. In some terminal groups, the different combinations share only the first input but in others, they share more than one input. It is necessary to calculate the number K of combination categories and the number S of shared inputs, apart from the first one in each category. The first requirement is achieved, as shown in (5-35), by calculating the cumulated weight of the smaller terminal groups included in the current one and comparing the result with the current threshold level.

$$\begin{cases} W_t(F+i) \geq T_t & \forall\, 0 \leq i < K \\ W_t(F+i) < T_t & \forall\, i \geq K \end{cases} \tag{5-35}$$

For each terminal group $G_t(F+j)$ (j=0,1,2,...K-1), the number S(j) of shared inputs is determined according to (5-36). To calculate S(j), individual weights are subtracted from the cumulated weight of the group and the result is compared with T_t. One input is shared by all the combinations in the current category, if and only if the subtraction result is smaller than the threshold level. Otherwise, there are input combinations capable to boost the 'net' value of the neurone above the threshold level without using the tested input. The number S(j) does not include the first input in the corresponding terminal group. According to the definition, this input is implicitly used by all the combinations in the same category, so that the input weight w^s_{F+j+i} is not even tested in (5-36).

$$\begin{cases} W_t(F+j) - w^s_{F+j+i+1} < T_t & \forall\, 0 \leq i < S(j) \;\; [\text{if } S(j) > 0] \\ W_t(F+j) - w^s_{F+j+i+1} \geq T_t & \forall\, i \geq S(j) \end{cases} \tag{5-36}$$

Therefore, the current subneurone is implemented as an OR gate with K inputs as illustrated in Fig. 5-9. The OR gate inputs are fed by AND gates with S(j)+2 inputs (j=0,1,2,...K-1) that model the K different combination categories. The first S(j)+1 inputs of each AND gate are connected to **all** the shared inputs of the combinations in the respective category (including this time the first input in the corresponding terminal group). Input S(j)+2 is connected to the output of a higher-order subneurone that analyses the contribution of the remaining inputs to the total net value. Go to step 10.

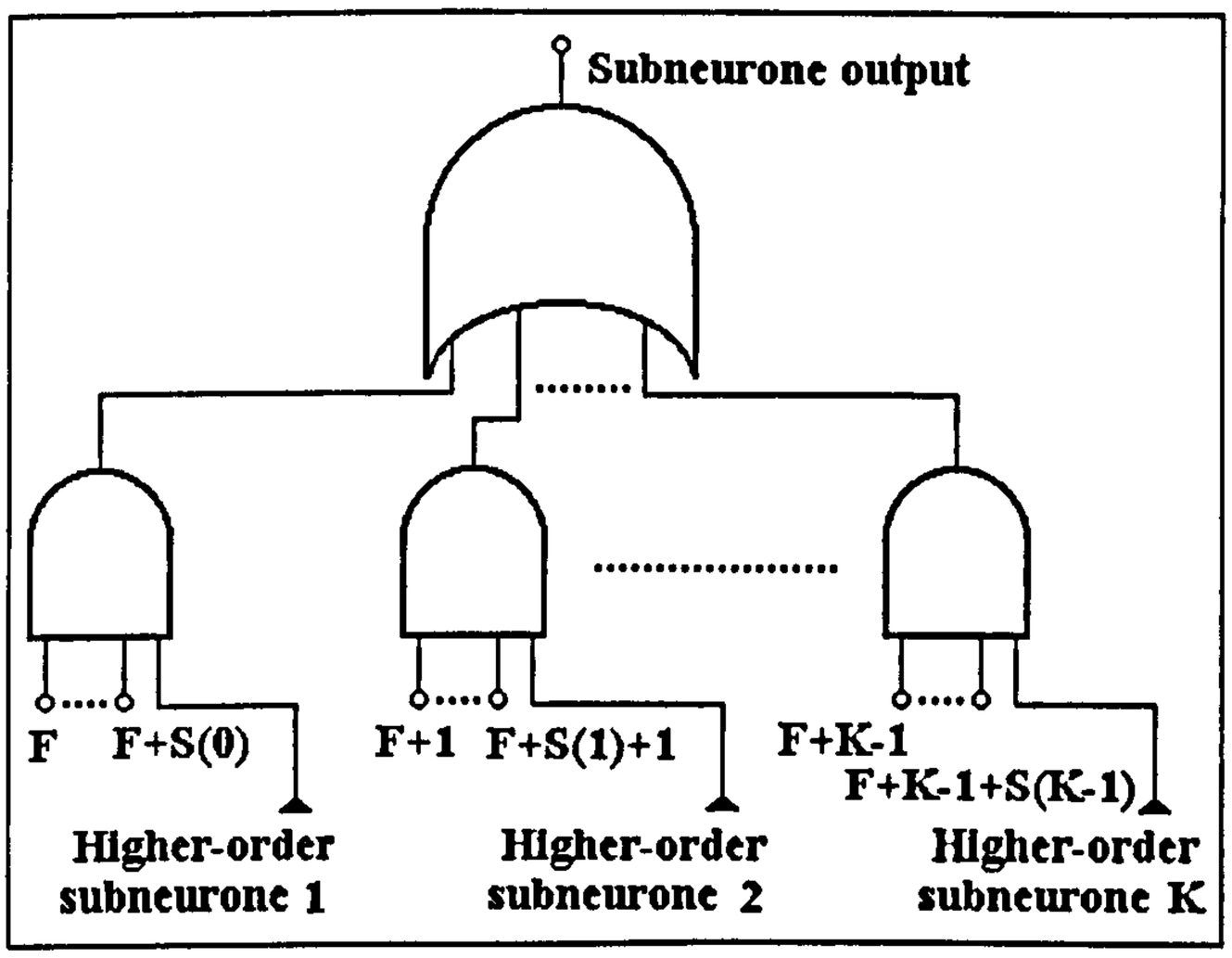

Fig. 5-9 - Subneurone implementation at step 7

The recursive implementation procedure is recalled K times for each high-order subneurone. The parameters for each call are calculated according to (5-37). The principles that underlie these calculations are similar to those applying to the parameters in (5-34). Go to step 10.

$$\begin{cases} F_j = F + S(j) + 1 \\ T_{tj} = T_t - \sum_{i=0}^{S(j)} w^s_{F+j+i} \end{cases} \quad (5\text{-}37)$$

Step 8) The neurone has dominant inputs and non-critical inputs. The combinations of non-critical inputs able to activate the neurone output fall into a number of K categories. Number K is determined using method (5-38), which is similar to (5-35) but takes into account the existence of the D dominant inputs. Thus, the index of the first non-critical input is, in this case, F+D instead of F, so that the initial index F+i in (5-35) has to be replaced with F+D+i.

$$\begin{cases} W_t(F+D+i) \geq T_t & \forall\, 0 \leq i < K \\ W_t(F+D+i) < T_t & \forall\, i \geq K \end{cases} \quad (5\text{-}38)$$

Similarly, the number S(j) of shared inputs in each category of input combinations is calculated according to method (5-39), which is derived from (5-36) by replacing each 'F' with 'F+D' to take into account the existence of the dominant inputs.

$$\begin{cases} W_t(F+D+j) - w^s_{F+D+j+i+1} < T_t & \forall\, 0 \leq i < S(j) \quad [\text{if } S(j) > 0] \\ W_t(F+D+j) - w^s_{F+D+j+i+1} \geq T_t & \forall\, i \geq S(j) \end{cases} \quad (5\text{-}39)$$

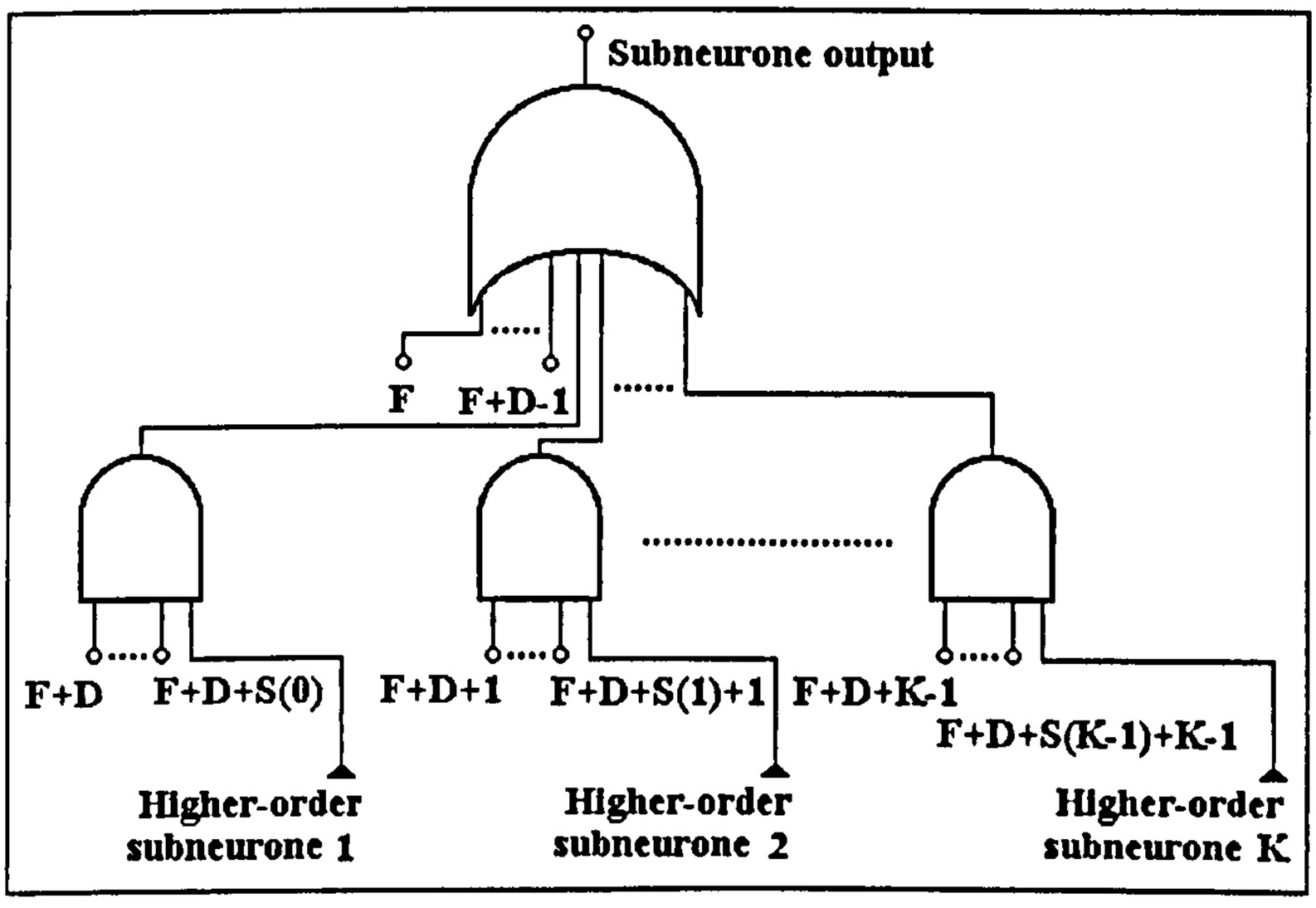

Fig. 5-10 – Subneurone implementation at step 8

As shown in Fig. 5-10, the neurone is implemented by an OR gate with D+K inputs interconnected with K AND gates. The first D inputs of the OR gate are connected to the subneurone dominant inputs, while the rest of the inputs are supplied by the AND gates.

As in the previous cases, the recursive procedure is recalled K times to implement the K higher-order subneurones in Fig. 5-10. The parameters for each call are given in (5-40). Go to step 10.

$$\begin{cases} F_j = F + D + S(j) + 1 \\ T_{tj} = T_t - \sum_{i=0}^{S(j)} w^s_{F+D+j+i} \end{cases} \tag{5-40}$$

Step 9) The neurone contains all three types of significant inputs: dominant, critical and non-critical. It is implemented by a D+1-input OR logic gate cascaded with a C+1-input AND gate and a higher-order subneurone as shown in Fig. 5-11. The higher-order subneurone analyses the combinations of non-critical inputs and activates the current subneurone output when a valid combination is received on the inputs simultaneously with all the critical inputs being active. To generate the higher-order subneurone implementation, the recursive procedure is called with the parameters calculated in (5-41). Go to step 10.

$$\begin{cases} F = F + D + C \\ T_t = T_t - \sum_{i=0}^{C-1} w^s_{F+D+i} \end{cases} \tag{5-41}$$

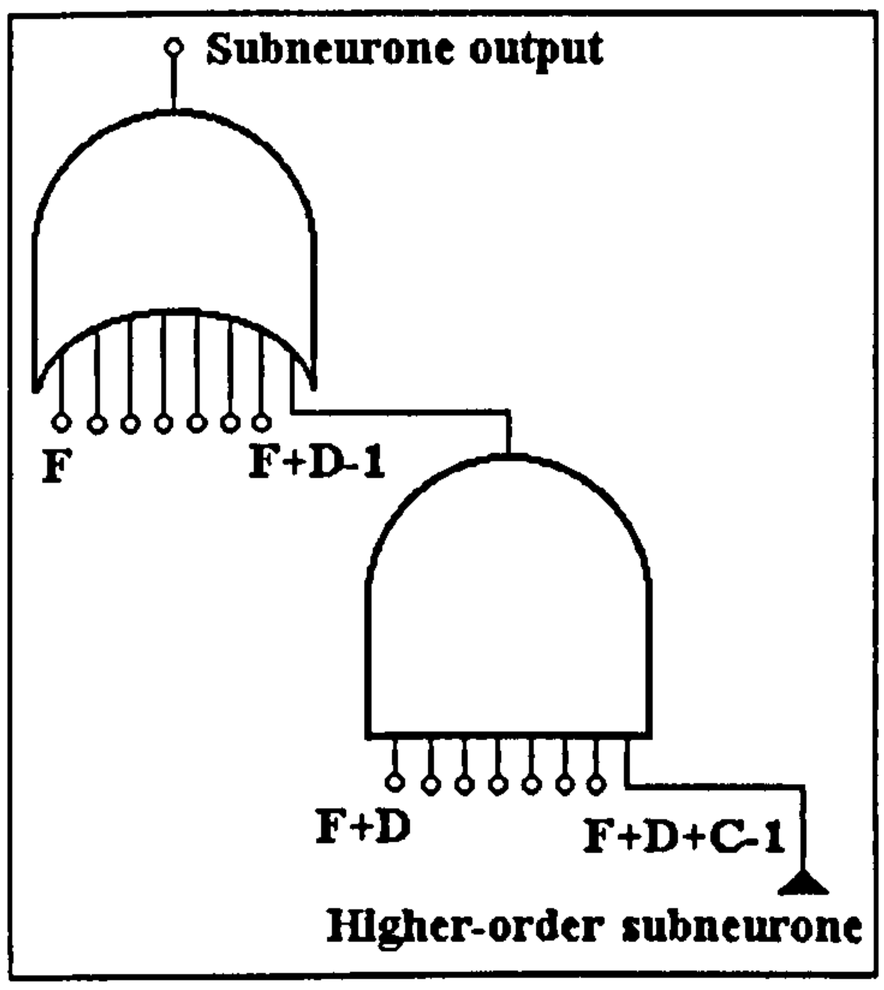

Fig. 5-11 – Subneurone structure at the step 9

Step 10) The execution of the implementation process returns to the point where the present procedure call has been performed. This point can be inside this procedure at

steps 6, 7, 8 or 9, or it can be at the stage where the recursive process itself was initiated. In the first case, according to computer programming principles, the old parameters F and T_t are restored and the execution resumes at the stage where this call was initiated. In the second case, the execution of the present procedure stops.

5.1.3.3 Neurone Implementation Example

For a better understanding of the implementation algorithm, a complete example is presented in Fig. 5-13. The neurone has A=12 input weights and positive threshold level 't'. The weights are sorted in descending order and the recursive implementation procedure is initiated with parameters F=1 and $T_t = t = 10$, as shown in Fig. 5-12. The number of dominant and critical inputs is calculated at step 1) of the recursive implementation procedure. The result is D=3, C=0. The three dominant inputs correspond to the dominant weights w_1^s, w_2^s, w_3^s in Fig. 5-13. Condition (5-33) is used to demonstrate that the neurone has non-critical inputs as well. Thus, according to Table 5-1, the next step to be performed is step 8). The number K of non-critical input combinations is calculated using relations (5-38). The result is K=3. The first two groups contain weight combinations sharing only one input each, while in the third group, four inputs are shared. Therefore, the output of the neurone implementation is generated by the 6-input OR gate g1 connected to the 3 dominant inputs and to 3 AND gates (g2, g3, g4). Gates g2 and g3 have two inputs while g4 has five inputs.

As illustrated in Fig. 5-12, the iterative procedure recalls itself three times to generate the subneurones corresponding to the three previously mentioned groups of weights. First, the procedure is recalled with parameters F=5 and $T_t = t - w_4^s = 1.9$ to generate the implementation of the subneurone connected to gate g2. This subneurone has four dominant inputs (related to the weights $w_5^s, w_6^s, w_7^s, w_8^s$), one critical input (corresponding to $w_9^s = w_3^{(2)}$) and two non-critical inputs (corresponding to w_{10}^s and w_{11}^s). Step 9) is carried out and gates g5 and g6 are inserted into the hardware structure. The remaining three inputs belong to a higher-order subneurone that requires the iterative procedure to be called for the third time.

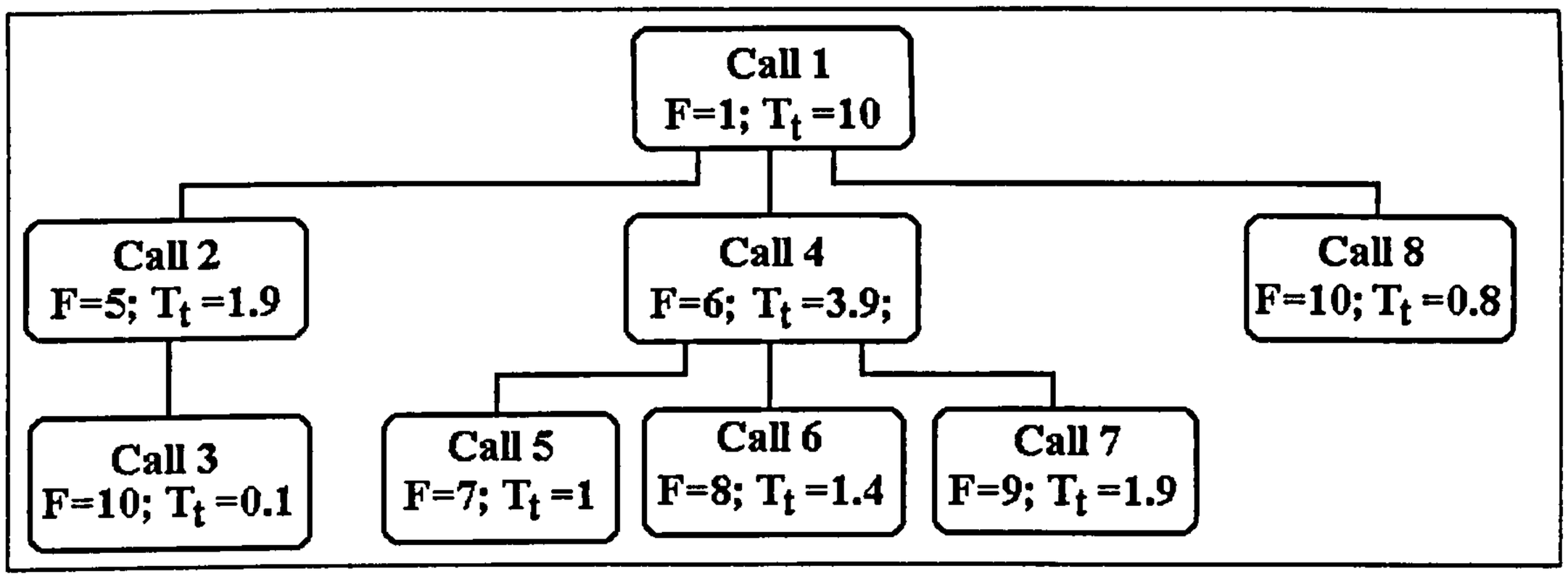

Fig. 5-12 – The recursive implementation process for the neurone in Fig. 5-13

The procedure parameters are redefined as F=10 and $T_t = t - 1.9 - w_9^s = 0.1$ during call number three. As a result, the corresponding subneurone contains two dominant inputs plus one insignificant input (corresponding to w_{12}^s) and it is implemented by the 2-input OR gate g7. At this stage, calls number 2 and 3 of the iterative procedure are finished. The control is handed over to call number 1 which initiates the call number 4 with the parameters are F=6, T_t=t- w_5^s =3.9 in order to generate the implementation of the subneurone connected to the AND gate g3. The new subneurone has only non-critical weights falling in K=3 categories and it is implemented by logic gates g8, g9, g10 and g11. This subneurone is connected to three third-order subneurones, which are analysed during procedure calls 5, 6 and 7, and their implementations contain the gates g12 to g15.

The end of procedure call 7 brings procedure call 4 to an end as well. The control is passed back to procedure call 1, which initiates the call number 8 with parameters F=10 and T_t=t- $w_6^s - w_7^s - w_8^s - w_9^s$ =0.8, and implements the subneurone connected to the AND gate g4. This second-order subneurone has two dominant inputs (corresponding to w_{10}^s and w_{11}^s) and one insignificant input (related to w_{12}^s), so that it is implemented by the 2-input OR gate g16.

The end of procedure call number 8 is followed by the end of procedure call number 1, which stops the entire recursive process. At this point the third procedure is called (procedure C in Fig. 5-6), and inverter gates are connected to the inputs related to the weight w_9. After this stage is finished, the neurone hardware implementation is complete. It is seen that the weight w_{12} is insignificant due to its small value and therefore the corresponding input was not necessary in any combination of inputs. Thus,

the neurone implementation consists of 8 subneurones and requires a total of 18 logic gates arranged on 6 layers.

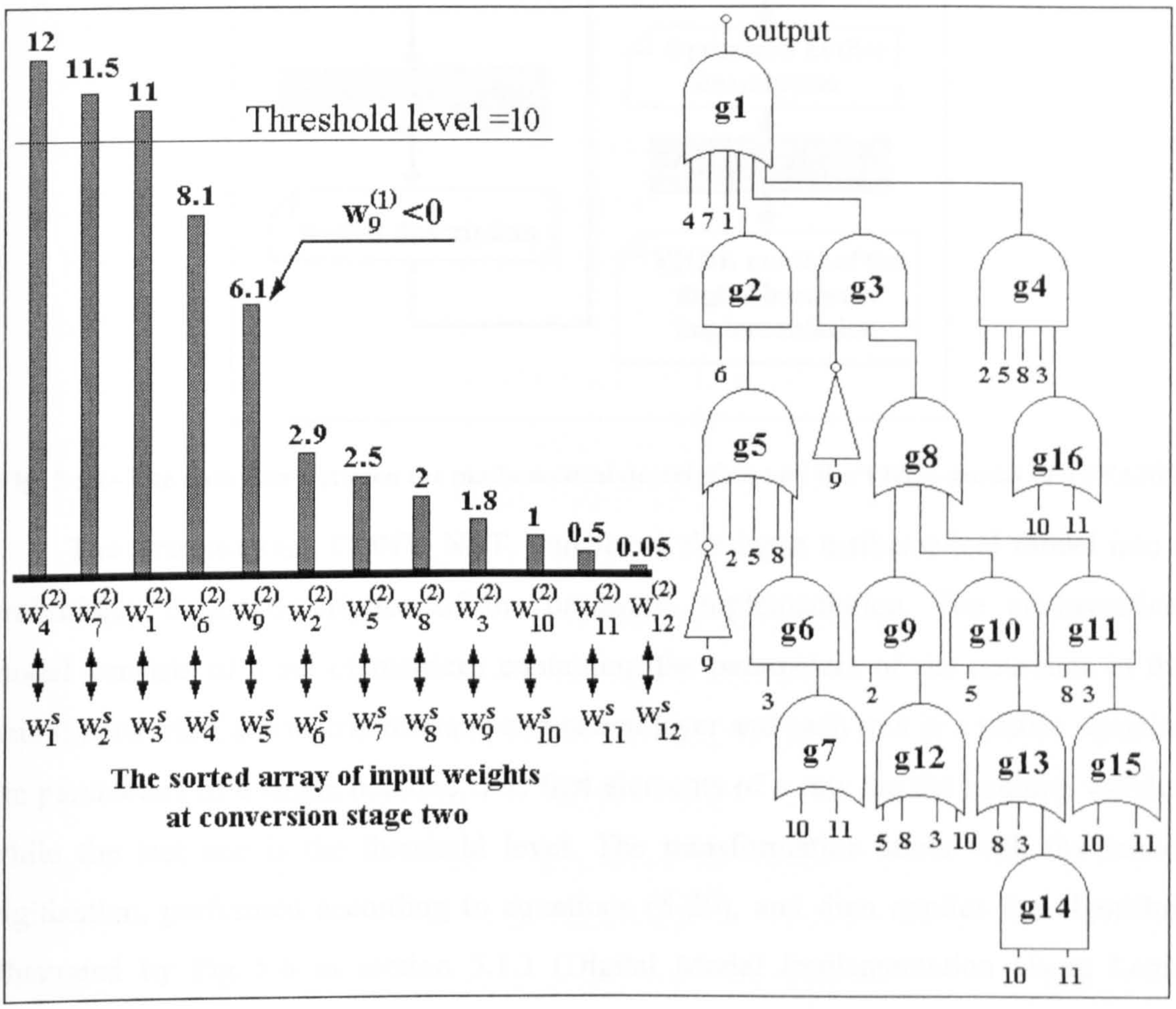

Fig. 5-13 - Digital mathematical model to gate structure conversion example

This example illustrates the complicated calculations necessary to transform even a simple neurone model into a system of interconnected logic gates. ANNs containing several neurones require an amount of calculations that can be efficiently performed only by specialised software instruments. Such instruments have been developed and they are presented in the next section.

5.2 UNIVERSAL PROGRAMS FOR FFANN HARDWARE IMPLEMENTATION

The solution adopted in this thesis is universal. It implies a three stage automatic analysis of the FFANN mathematical model and the generation of a VHDL model describing the corresponding hardware structure. The task is carried out by a set of three interconnected C++ programs, given in Appendix A and illustrated in Fig. 5-14, which communicate by means of simple ASCII files.

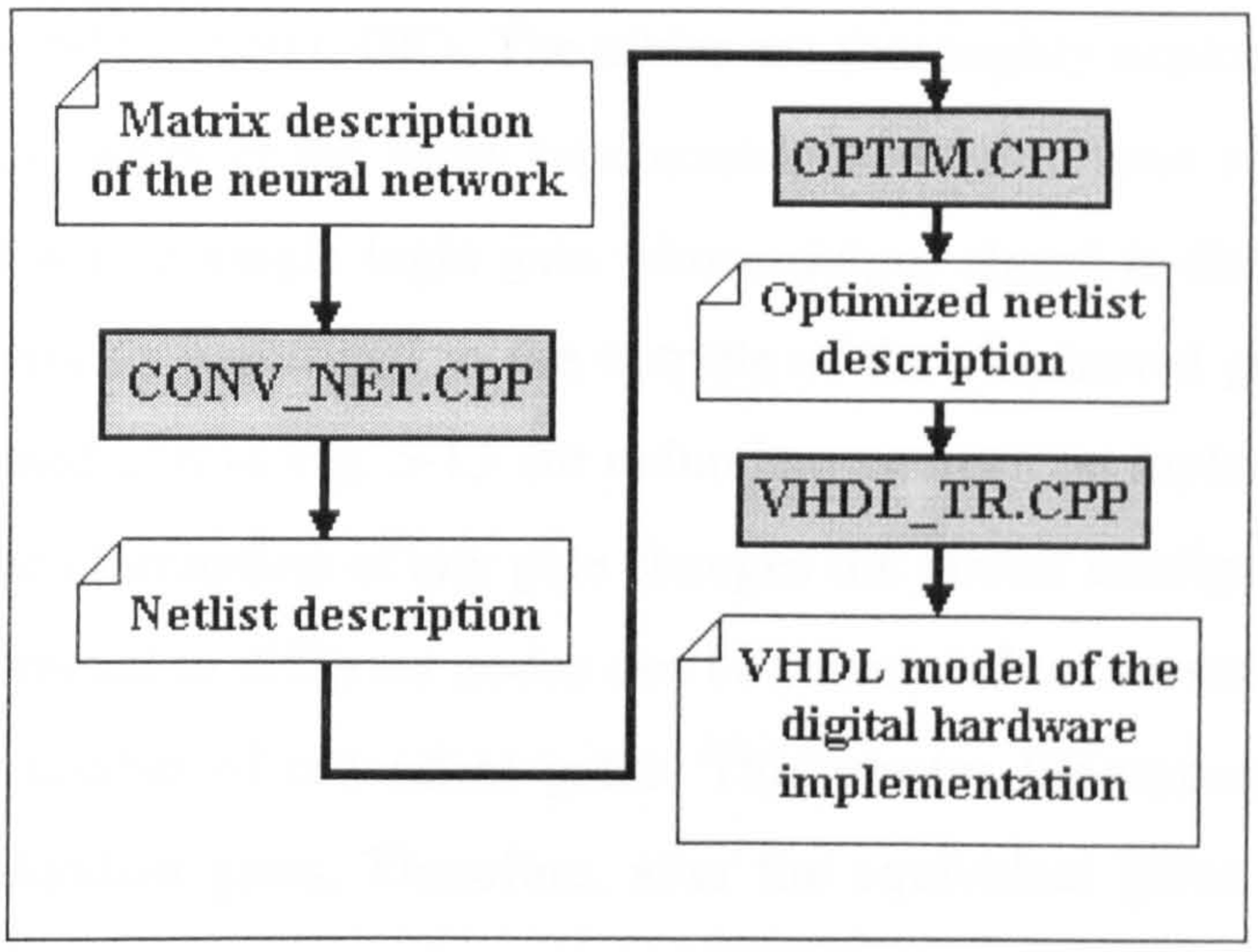

Fig. 5-14 - The data flow between the mathematical description and the VHDL model of a FFANN

The first program, CONV_NET, transforms the input mathematical model into a preliminary netlist description of the hardware implementation. The mathematical model consists of a set of matrices containing the parameters of the neurones in the neural network. Each matrix refers to one neural layer and each row in a matrix contains the parameters of a single neurone. The first elements of a row are the neurone weights while the last one is the threshold level. The transformation starts with the model digitisation, performed according to equations (5-29), and then applies the algorithm illustrated by Fig. 5-6 in section 5.1.3 (Digital Model Implementation Using Logic Gates). The program allows the user to set the number of bits used by the analogue inputs, and the maximal number of inputs per logic gate. If a larger number of inputs are required at a certain stage of the conversion, a pyramidal interconnection of simpler gates will be used to replace the required gate (Fig. 5-15).

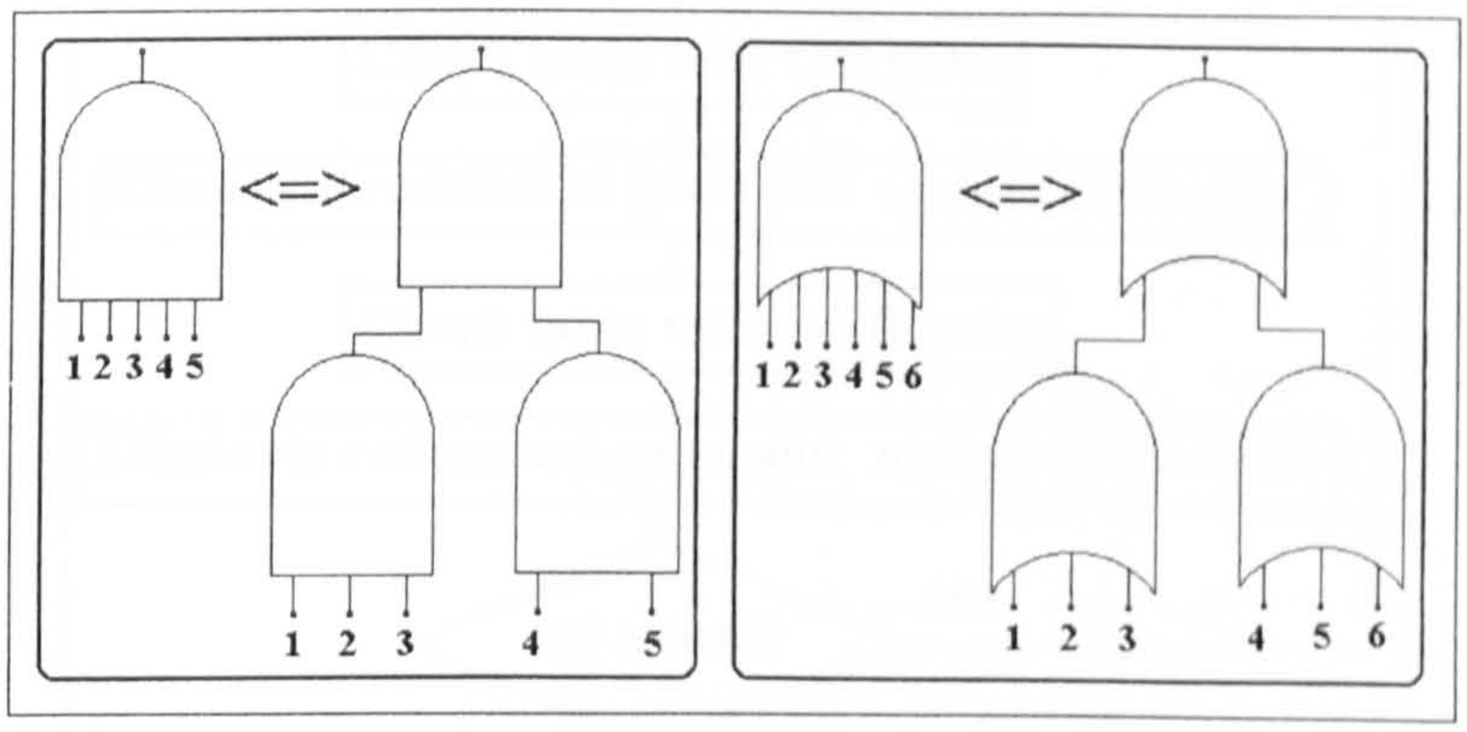

Fig. 5-15 – Examples of fan-in reduction using interconnections of simpler logic gates

The second program, OPTIM, minimises the netlist description by eliminating the redundant components. The netlist optimisation requires that three memory tables containing the circuit nodes and gates are built. Each table contains data about a specific

type of logic gate (NOT, AND, OR). The tables are thoroughly explored to find groups of redundant gates (gates of the same type connected to the same input nodes). Each group is replaced with a single logic gate whose output signal is distributed to all the circuit nodes previously connected to the outputs of the eliminated gates. For instance the gates g7, g15 and g16 in Fig. 5-13 are redundant and can be replaced by a single 2-input OR gate. The elimination of any gate changes the circuit configuration. Gates that were initially connected to different nodes can be connected to the same nodes after the elimination of a number of redundant gates. This creates the opportunity for further elimination of redundant gates. Therefore, after the equivalent gates are removed, the tables are updated and the process is restarted as shown in Fig. 5-16. The optimisation process stops only when no additional modification can be made in any of the three tables.

The third program, VHDL_TR, transforms the optimised netlist description into a VHDL model of the hardware implemented neural network. The obtained VHDL file can be synthesised using any commercially available software package specialised in FPGA design. The file contains a single VHDL entity (the network) whose corresponding architecture comprises a number of internal signals and a list of assignment statements. Each statement models one or several identical logic gates by associating a logical expression either with an internal signal or with an output signal.

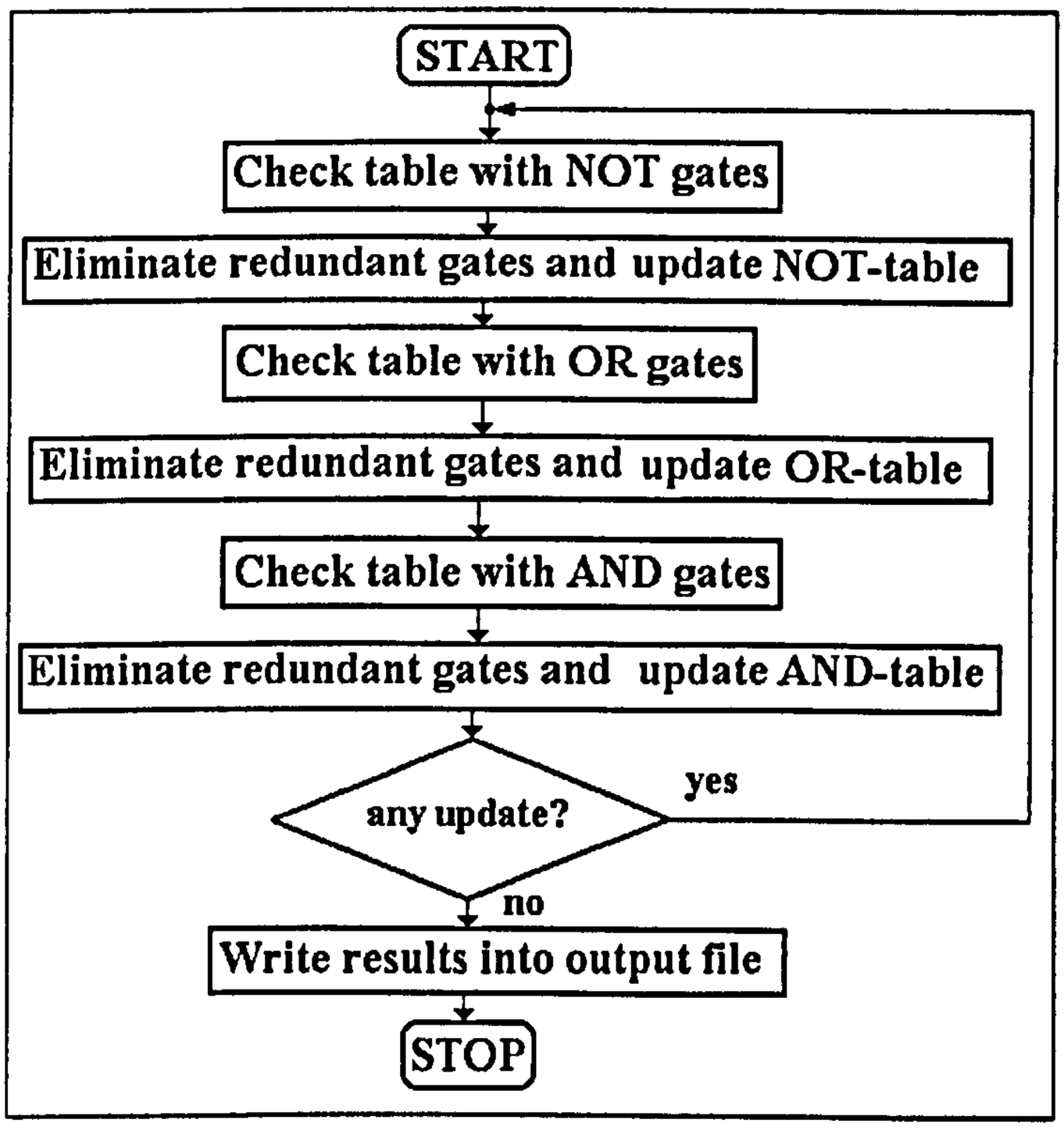

Fig. 5-16 – The general flow-chart underlying the optimisation program

For exemplification, the VHDL model of the neurone in Fig. 5-13 is presented below. The model has been generated using the three universal C++ programs. It is important to note that the index of the components inside the input port 'd_in' vary between 0 and 11 instead of 1 to 12 as it was in Fig. 5-13.

```
-- Code Fragment 5.1
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY network1 IS
  PORT(d_in : IN std_logic_vector(11 DOWNTO 0); --the 12 input signals
       d_out: OUT std_logic_vector(0 DOWNTO 0));--the single output
END network1;

ARCHITECTURE arch_network1 OF network1 IS
    SIGNAL n1,n2,n3,n4,n5,n7,n8,n9,n10,
           n11,n12,n13,n16: std_logic;
BEGIN
n16<= NOT d_in(8);                                          -- the NOT gate
n1<= d_in(5) AND n4;                                        -- gate g2
n2<=n16 AND n7;                                             -- gate g3
n8<= d_in(1) AND n11;                                       -- gate g9
n9 <= d_in(4) AND n12;                                      -- gate g10
n5<= d_in(2) AND n15;                                       -- gate g6
n13 <= d_in(9) AND d_in(10);                                -- gate g14
n7<= n8 AND n9 AND n10;                                     -- gate g8
n10 <=d_in(7) AND d_in(2) AND n15;                          -- gate g11
n3<= d_in(1) AND d_in(4) AND d_in(7) AND d_in(2) AND n15;
n15<= d_in(9) OR d_in(10);                                  -- gates g7, g15, g16
n12 <= d_in(7) OR d_in(2) OR n13;                           -- gate g13
n11 <= d_in(4) OR d_in(7) OR d_in(2) OR d_in(9);            -- gate g12
n4<= n16 OR d_in(1) OR d_in(4) OR d_in(7) OR n5;            -- gate g5
d_out(0)<=d_in(3) OR d_in(6) OR d_in(0) OR n1 OR n2 OR n3;
END arch_network1;

CONFIGURATION conf_network1 OF network1 IS
  FOR arch_network1
  END FOR;
END conf_network1;
```

The optimisation performed by OPTIM has three important effects on the previous VHDL model:

- A single expression models the group of the redundant gates g7, g15 and g16. The other logic gates are modelled by individual logic expressions.
- The internal signals n6, n14 and n15 are absent in the list at the beginning of the network architecture. They have been removed alongside with the gates g7, g15, and g16.
- The three types of logic operators (NOT, AND, OR) occur in three distinct sections of the network architecture description. Inside each section, the logic expressions are sorted in ascending order according to the number of logic operators involved. This

feature is just a side effect of the optimisation algorithm but it simplifies the inspection of the obtained VHDL model (for instance, counting the total number of gates of a certain type or with a certain fan-in).

5.3 THE HARDWARE IMPLEMENTATION COMPLEXITY ANALYSIS

There are two important cost functions characterising the ANN hardware implementations: the input-output delay and the required chip area. For most applications, the delay time is satisfactory but the chip area is critical because the hardware resources are always limited. The input-output delay is approximately proportional to the implementation depth, which is defined as the number of layers of elementary circuit units: TGs or logic gates. Several approximate methods have been proposed to determine the required chip area of an ANN, depending on the envisaged implementation technology. They imply the calculation of the number of neurones, the number of implementation units (logic gates or threshold gates, depending on the technology) [29], the total input number of all implementation units [60], the sum of all input weights and thresholds [29], etc. In the case of FPGA implementations, the total number of gates is the most suitable means to determine the implementation complexity.

It is difficult to calculate in advance the number of logic gates required by a pyramidal logic structure with n inputs, like the one in Fig. 5-13, because the result depends on the fan-in of each individual logic gate. The calculations are simple only if it is possible to achieve the implementation with logic gates having the same fan-in Δ. In this case, the implementation complexity is given by equation (5-42), where $\lceil x \rceil$ is the ceiling function (the smallest integer greater or equal than x).

$$N_{LG\Delta} = \left\lceil \frac{n-1}{\Delta - 1} \right\rceil \qquad (5\text{-}42)$$

Usually the implementation algorithms require logic gates with different fan-ins, so that (5-42) is applicable only to a limited number of practical situations. However, any Δ-input AND gate or Δ-input OR gate can be replaced by a number of Δ gates of the same type, but having only two inputs. If the fan-in is restricted to Δ=2, then equation (5-42) is simplified as (5-43).

$$N_{LG2} = n - 1 \qquad (5\text{-}43)$$

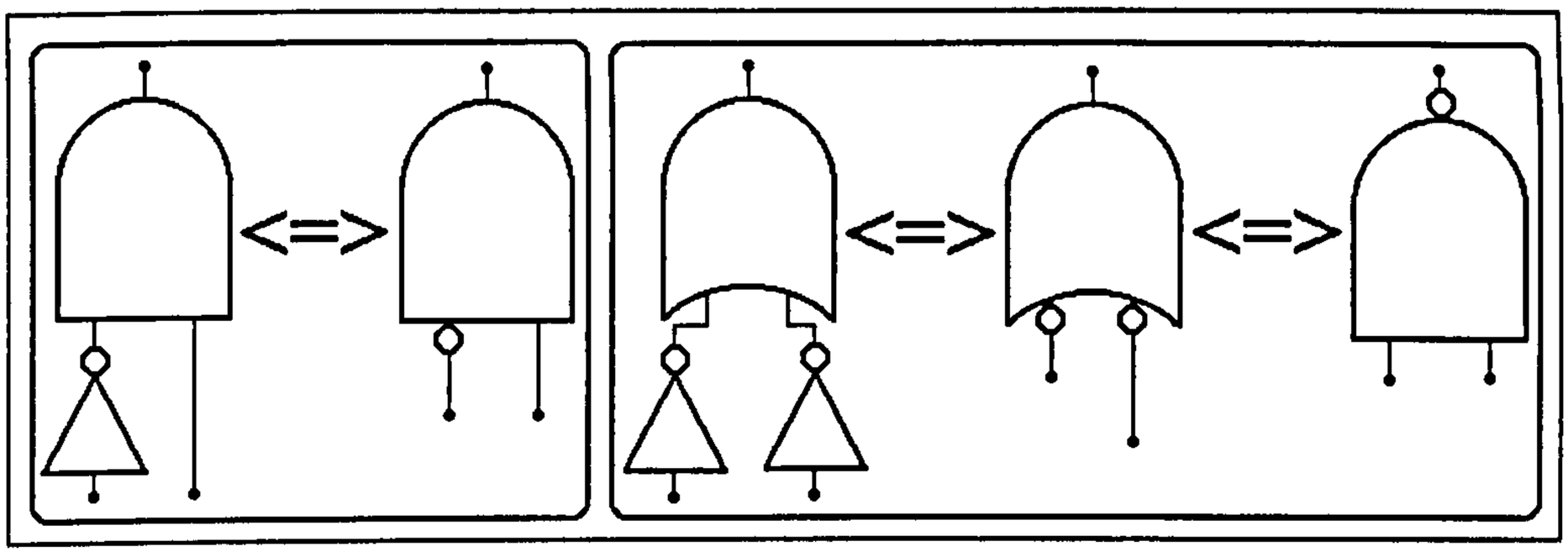

Fig. 5-17 – The integration of NOT operator in complex logic gates performing NOT-AND and NOT-OR operations

Any logic circuit can be built using exclusively 2-input logic gates. Therefore, the number of equivalent 2-input gates in the neural network implementation is a universal measure of its hardware complexity and offers a means to compare different implementation algorithms. As opposed to the rest of the logic gates, the NOT gates always have Δ=1. However, they are not taken into account when estimating the implementation complexity because the NOT logic operator can be integrated into 2-input logic gates as shown in Fig. 5-17. Note that the total number of inputs 'n' in (5-42) and (5-43) is larger than the number of the neurone binary inputs ($A=n_a \times n_b$) because some input signals drive more than one gate in the pyramidal structure.

5.3.1 Results Previously Reported in the Literature

An efficient neural network implementation strategy is one that minimises the number of equivalent 2-input gates in the corresponding digital circuit. Only a few complexity minimising algorithms have been developed so far for digital hardware implementations. The most relevant two of them are proposed in [121] and [30] and lead to the same order of implementation complexity but generate different circuit depths. The results presented in [30] are converted here in numbers of 2-input gate and then a comparison is performed between these results and the hardware complexity generated by the new implementation strategy proposed in this thesis.

The neural network is treated in [30] as a set of k Boolean functions $F_{n,m,i}$ (i=1,2,...k), with n inputs and a cumulated total of m groups of '1' in the truth table. In the one-dimensional case a group of '1' is a set of successive n-bit input strings, whose corresponding function outputs are all '1'. The approach can be extended from one dimension to several dimensions, as shown in Fig. 5-18. The number of truth table dimensions equals the number n_a of inputs of the analogue neurone modelled.

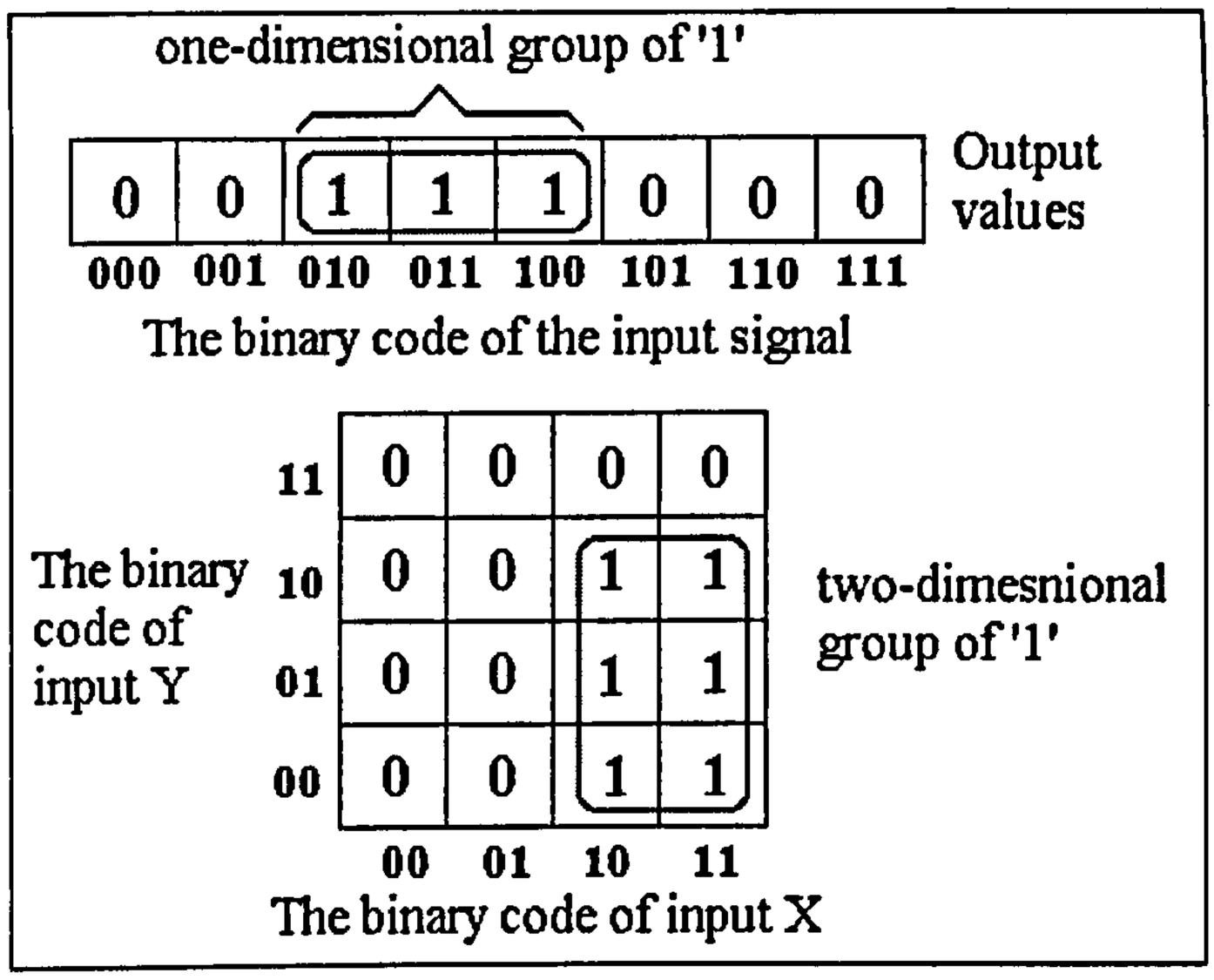

Fig. 5-18 – Groups of '1' in one-dimensional and two-dimensional truth tables

The groups of '1' are generated with a constructive method so that they optimally cover those points in the d-dimensional input data space that have to activate the outputs of the corresponding network. Therefore, the constructive method used in [30] is a particular case of Voronoi algorithm, where all the Voronoi cells are hypercubes bounded by hyperplanes parallel to the axes of the input data space. Three implementation alternatives are compared:

(i) The direct function implementation in disjunctive normal form (DNF) (initially proposed in [26]).

(ii) A more sophisticated strategy which involves the use of n-bit comparators alongside with AND gates and OR gates. The comparators model the n_a-dimensional hyperplanes parallel to n_a-1 axes of the input data space. Each of them performs comparisons between one of the n_a analogue input signals and a constant. The second layer is made up of $2n_a$-input AND gates. Each AND gate implements a hypercube-shaped Voronoi cell corresponding to a group of '1'. The third layer is made up of OR gates combining the information provided by different AND gates.

(iii) A synthesis of the previous two methods that replaces some of the comparators with DNF terms of the Boolean function. This method analyses the size of the groups of '1'. The small cells are more efficiently implemented in DNF format, while large groups are better implemented by comparators. Thus, some of the comparators are replaced by a number of AND gates and NOT gates.

Any n-bit comparator between a variable quantity and a constant value can be implemented with up to 'n-1' 2-input gates [28]. A neural network with n_a analogue

inputs coded on n_b bits each requires up to $n_a \cdot (2^{n_b} - 1)$ comparators, which is equivalent to $n_a \cdot (n_b - 1) \cdot (2^{n_b} - 1)$ 2-input logic gates. The redundancy across different comparators can be reduced by optimisation algorithms. The optimisation is limited by the number of comparators. The comparators outputs are independent signals and therefore are generated by separated logic gates. Thus, if there are $n_a \cdot (2^{n_b} - 1)$ comparators then the complexity of the circuit cannot be decreased below $n_a \cdot (2^{n_b} - 1)$ 2-input logic gates.

The set of Boolean functions $F_{n,m,i}$ (i=1,2,...k) contains a total of m groups of '1' in the truth tables. Consequently, the implementation complexity of the second neurone layer in the corresponding ANN is up to $(2 \cdot n_a - 1) \cdot m$ equivalent 2-input logic gates. On the other hand, the hardware complexity of the third layer is up to $m - k + C_k^2$. This result is a generalisation of the particular cases illustrated in Fig. 5-19. Thus, if the neural network has only one output (k=1) then the third layer is implemented as a pyramid of OR logic gates with the complexity of m-1 equivalent two-input gates. If the neural network has two outputs then the situation is more complex. Thus, the outputs are generated by two different pyramids that can share some of the 'm' inputs (Fig. 5-19-(c)) or they can be completely separate (Fig. 5-19-(b)). When the two pyramids share part of the 'm' inputs the resulting implementation contains three subpyramids and two extra OR-gates generating the actual output signals. As a result, the hardware complexity is larger than in Fig. 5-19-(b). When the number neural network outputs is larger than two, several situations are possible depending on the number of shared clusters of input signals between different OR-gate pyramids. Two possibilities are illustrated in Fig. 5-19-(d) and Fig. 5-19-(e) for the situation when k=3. Generally, the hardware complexity corresponding to the third neural network layer increases with the number of shared clusters of inputs. The maximal number of input clusters is C_k^2. Therefore, a number of $k + C_k^2$ subpyramids are contained in the corresponding hardware implementation. Furthermore, each output is generated by a pyramidal OR-gate structure with k inputs and a complexity of k-1 two-input logic gates. The total complexity of the third layer can be therefore calculated as

$$m - k - C_k^2 + k \cdot (k - 1) = m - k + C_k^2 \tag{5-44}$$

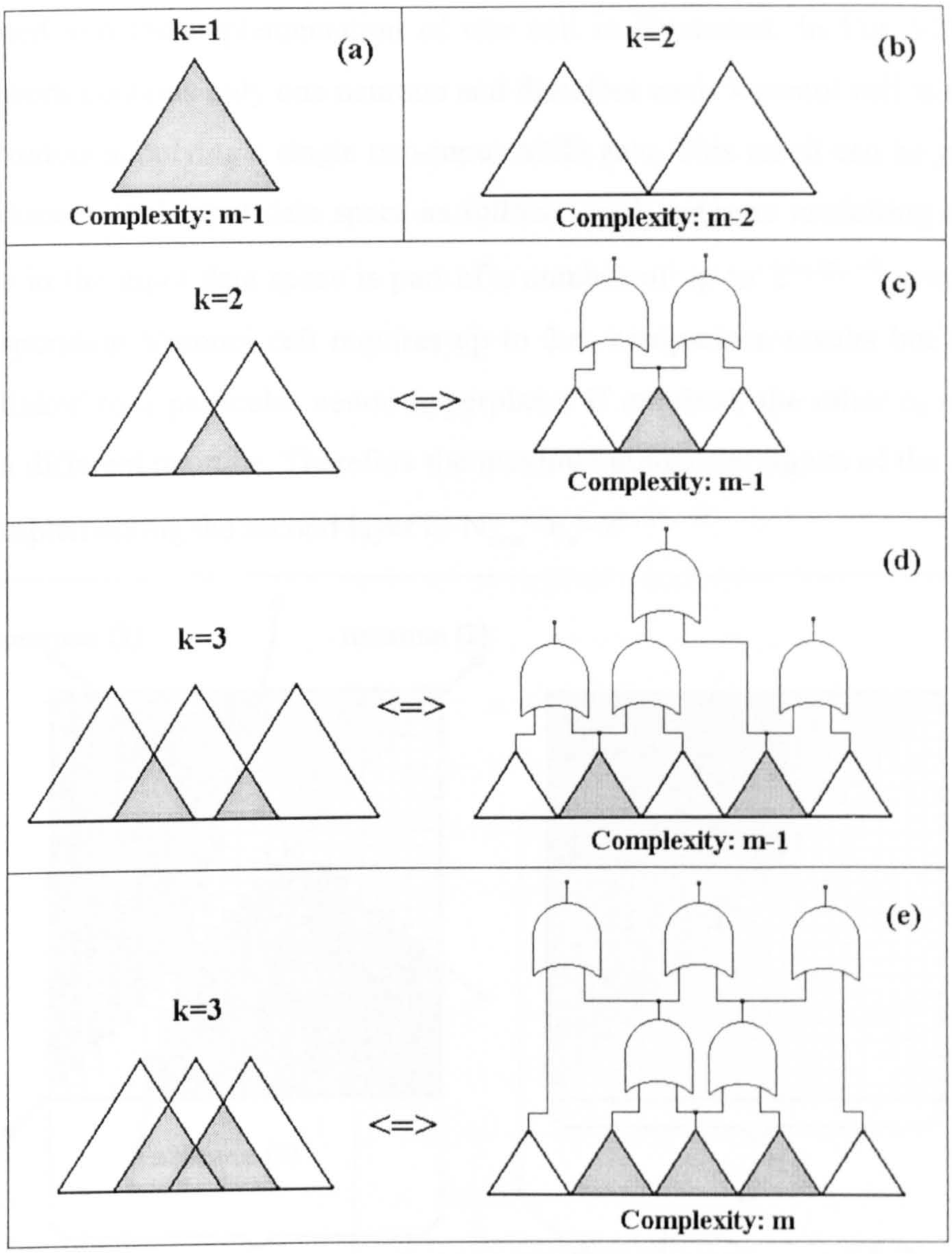

Fig. 5-19 – Analysis of the third layer complexity (typical situations)

Thus, the upper limit of the implementation complexity for method (ii) in [30] is:

$$N_{LG2(ii)} = n_a \cdot \left(2^{n_b} - 1\right) + \left(2 \cdot n_a - 1\right) \cdot m + m - k + \frac{k(k-1)}{2} \qquad \textbf{(5-45)}$$

It is demonstrated in [30] that the hybrid method (iii) generates implementations with up to four times less complexity. The complexity level given in (5-45) is thereby reduced to:

$$N_{LG2(iii)} = \frac{n_a \cdot \left(2^{n_b} - 1\right)}{4} + \frac{n_a \cdot m}{2} + \frac{k(k-1)}{8} - \frac{k}{4} \qquad \textbf{(5-46)}$$

The complexity of the second layer can be calculated as a function of the number N_{neur} of neurones in the first layer of the ANN generating the Boolean functions $F_{n,m,i}$ (i=1,2,...k). Two different situations are illustrated in Fig. 5-20 (a) and (b). In Fig. 5-20 (a), the decomposition of the central region of the diagram into Voronoi cells is

demonstrated and the implementation of one cell is illustrated. In Fig. 5-20 (b), the neural network contains only one neurone and therefore each Voronoi cell is defined by two comparators supplying a single two-input AND gate. This result can be generalised for an n_a-dimensional input data space as follows: each neurone modelling an oblique hyperplane in the input data space is part of a number of up to $2^{n_b \cdot (n_a - 1)}$ groups of '1'. The corresponding Voronoi cell requires up to $2 \cdot n_a$ comparison results but only n_a of them are linked to a particular neural hyperplane. If required, the other n_a signals are related to a different neurone. Therefore the maximal number of inputs of the AND-gate structure implementing the second layer is $N_{neur} \cdot n_a \cdot 2^{n_b \cdot (n_a - 1)}$.

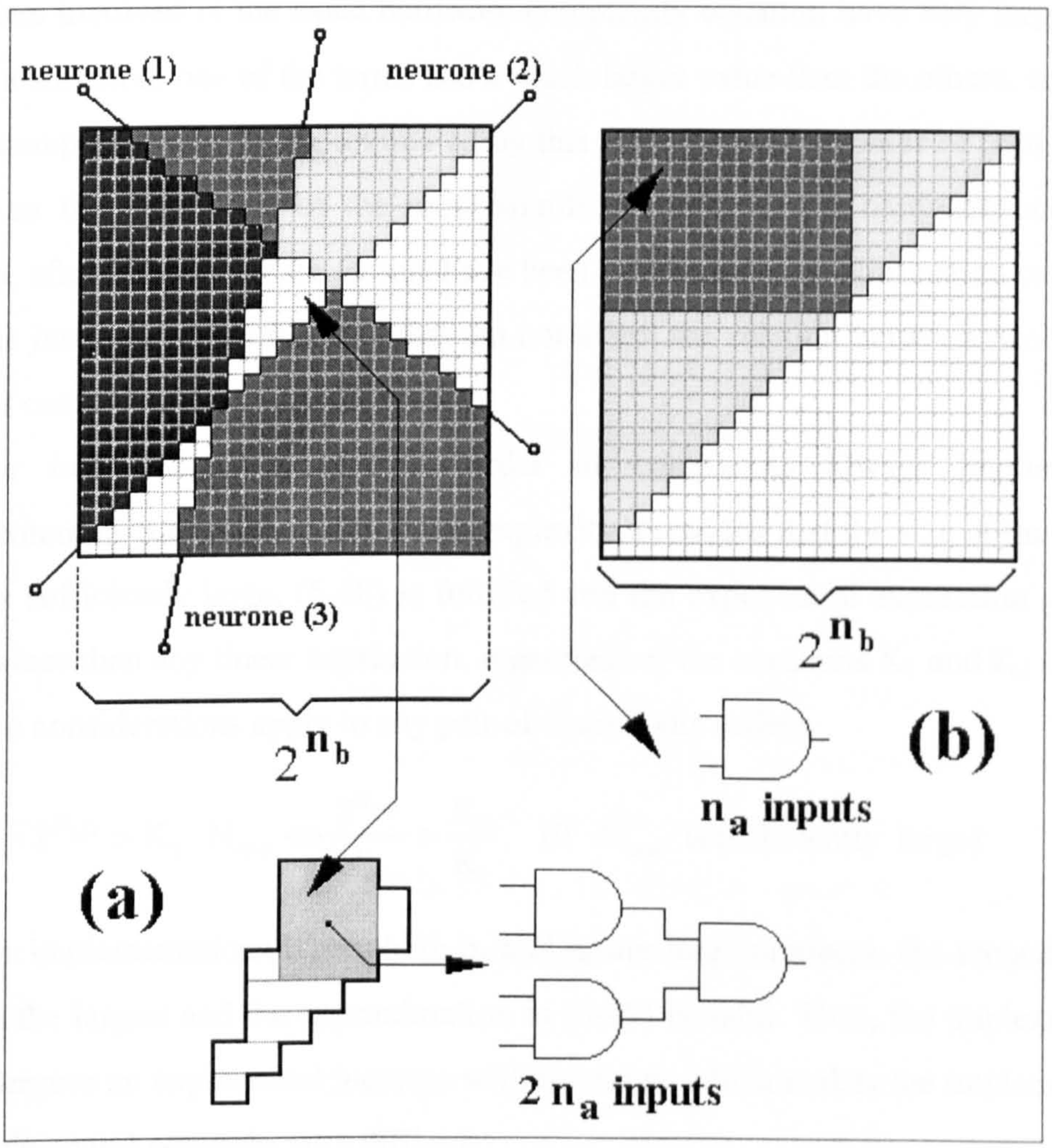

Fig. 5-20 – The estimation of the second layer complexity in a two-dimensional case

In these conditions, equation (5-46) becomes (5-47). The result in (5-47) is an upper limit because it is possible that more than one neurone includes a certain group of '1'.

$$N_{LG2oblique} = \frac{n_a \cdot \left(2^{n_b} - 1\right)}{4} + \frac{N_{neur} \cdot n_a \cdot 2^{n_a n_b - n_b} - 1}{2} + \frac{k(k-1)}{8} - \frac{k}{4} \qquad \textbf{(5-47)}$$

The most general parameter used to compare the general properties of the implementation algorithms is the **order of complexity** of the generated hardware structure. The order of complexity is a concept initially used in software engineering and computer sciences, but it has been extended for assessing the size of neural hardware implementations [28], [29], [30]. The order of complexity associated with an implementation algorithm is an expression that shows how the implementation complexity varies with the increase of the network parameters (number of neurones, number of interconnections, etc.). The increase can be linear, polynomial, exponential, factorial, etc. The order of complexity is obtained considering that all network parameters involved in the exact hardware complexity equation have very large values. In such a situation, one of the terms has a much larger value than the others, so that the overall complexity can be approximated by this term alone. The order of complexity is defined as the expression of the most significant term in the hardware complexity equation, after all the constant factors have been eliminated. The elimination of constant factors is justified by the fact that they do not affect the relation between two different orders of complexity.

For instance, an exponential order of complexity always implies larger implementations than a linear order of complexity. Provided that the size of the network (N_{neur}) is sufficiently large, (5-48) is fulfilled and the exponential expression generates larger values than any linear expression, regardless of the constants K_1 and K_2 involved. The same considerations apply to any pair of complexity orders.

$$K_1 \cdot 2^{N_{neur}} > K_2 \cdot N_{neur} \Leftrightarrow \frac{2^{N_{neur}}}{N_{neur}} > \frac{K_2}{K_1} \quad (\text{if } N_{neur} \text{ is sufficiently large}) \qquad (5\text{-}48)$$

For implementations where both n_a and n_b are large numbers, the second term in (5-47) is the largest and the approximation in (5-49) is valid. Thus, the implementation size undergoes an exponential increase with n_a and n_b which makes the implementation of sizeable neural networks very difficult.

$$N_{LG2oblique} \approx \frac{N_{neur} \cdot n_a \cdot 2^{n_a n_b - n_b}}{2} \Leftrightarrow O\left(N_{neur} \cdot n_a \cdot 2^{n_a n_b - n_b}\right) \qquad (5\text{-}49)$$

5.3.2 The Analysis of the New Implementation Method

The logic structure generated by the implementation algorithm adopted in this thesis, initially presented in section 5.1.3.2, is analysed now from a geometrical point of view, in order to determine the corresponding hardware complexity. The analysis is first

performed for neurones with two analogue inputs (X and Y), and then the results are generalised for any number of analogue inputs. The hardware complexity is initially assessed without taking into account any hardware optimisation. Then the improvements brought about by the optimisation algorithm presented in section 5.2 are considered as well, and the hardware complexity after optimisation is discussed.

5.3.2.1 Implementation Without Optimisation

If the neurone has only two analogue inputs then the input data space is two-dimensional and the hyperplanes are reduced to simple lines. The input data space is divided in 4 quadrants depending on the values of the most significant bits in X and Y input binary codes. The half-space where the neurone output is active covers a number of one, two, three or four of these quadrants. The quadrants can be either partially covered or totally covered. Each quadrant is in its turn divided into other four subquadrants defined by the second significant bit in each input code. The division process can be carried out for n_b times because each input code contains n_b bits.

Each division in four subquadrants corresponds to a subneurone inside the complete hardware implementation. The subneurone models the boundary between the active region and the inactive region inside the subquadrant. If the symmetrical situations are ignored, there are only eight types of relative positions between the hyperplane and the four quadrants. They are analysed in Fig. 5-21 alongside with the corresponding subneurone implementations. The presented results apply to the subneurones of orders larger than one but smaller than n_b. The analysis of the first-order subneurones generates results similar to the findings shown in Fig. 5-21, but the bits '0' and '1' in the truth tables are reversed. This situation is caused by the use of two's complement codification where the most significant bit of positive numbers is '0', while for negative numbers it is '1'.

Therefore, all subneurones of order $i<n_b$ in the pyramidal structure are fed with the signals of zero, one, two or three higher-order subneurones. The subneurones of order n_b are not fed by other subneurones because there are no more bits available in the input codes to generate such subneurones. In this case, as shown in Fig. 5-22, the higher-order subquadrants are either completely included or completely excluded from the active region of the current subquadrant. Such subquadrants are named elementary subquadrants because they cannot be further divided into higher-order subquadrants. If the area bounded by the hyperplane is more than half the surface of a higher-order

subquadrant then it is completely included in the active region. Otherwise, it is completely excluded. There are five types of n_b-order subneurones defined by the number of subquadrants that are included in the active region (Fig. 5-22). Two of them have the hardware complexity N_{LG2}=1 (H and J), while the other three have the complexity N_{LG2}=0 (G, I, K), because they do not require any logic gate for their implementation.

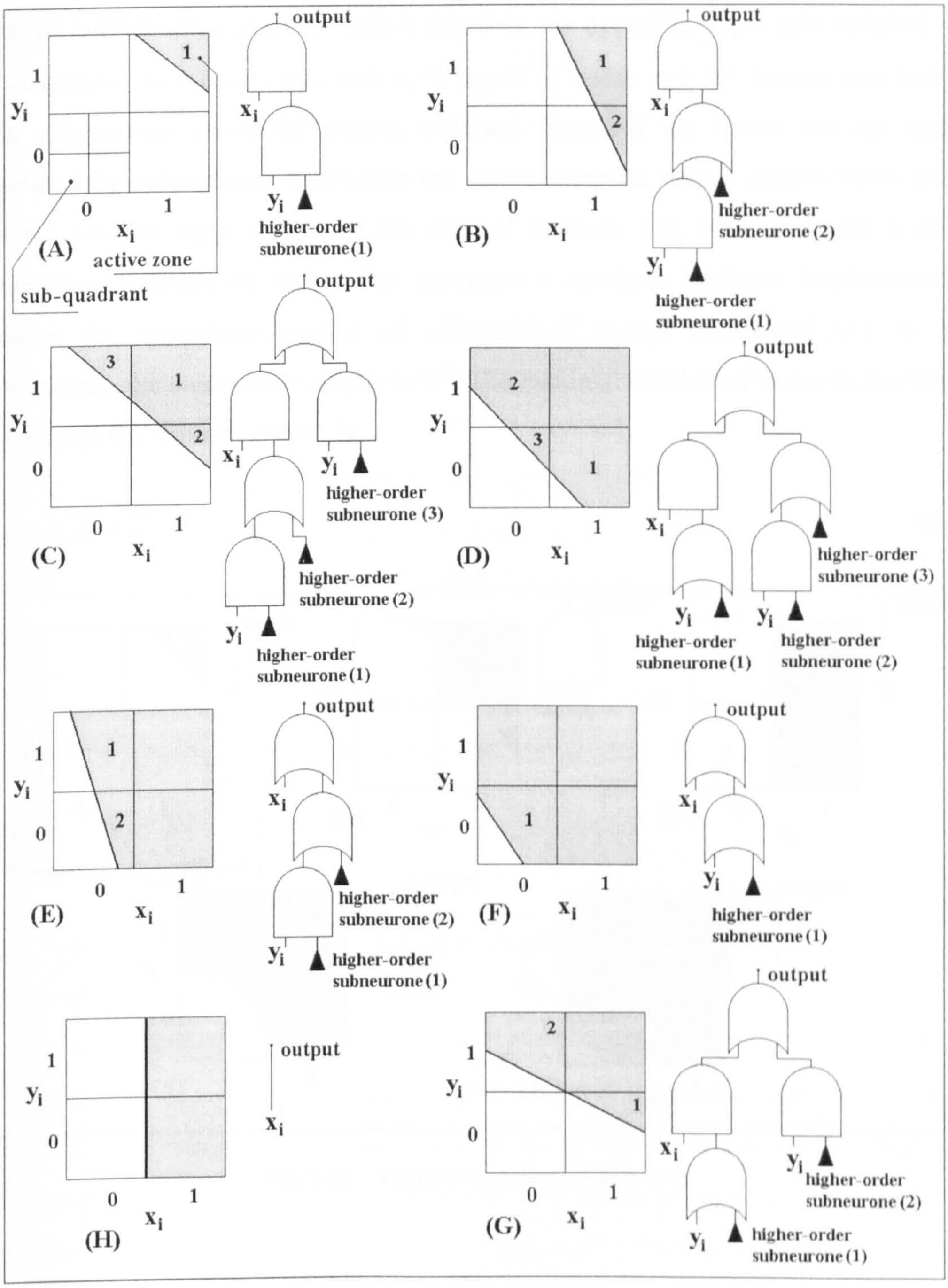

Fig. 5-21 – The division of a (i-1)-order quadrant in i-order quadrants and the corresponding subneurone implementations ($1<i<n_b$)

In a data space with more than two dimensions, there are more subneurone types than in Fig. 5-21. As Fig. 5-21 demonstrates, the number of subneurones equals every time the number of subquadrants crossed by the hyperplane. Therefore, the highest hardware complexity is obtained when the number of subquadrants crossed by the hyperplane is maximal. A hyperplane in a n_a-dimensional space can cross up to $2^{n_a}-1$ quadrants. Therefore, this is the maximum number of higher-order subneurones that can feed the current subneurone. In such a situation, the pyramidal logic gate structure can be considered as a binary tree with n_a+1 layers of nodes and 2^{n_a} leaves: one leaf for each subquadrant inside the current quadrant. Some of the leaves use the signals generated by subneurones to produce the correct neurone output signals while others directly use the input signals of the original neurone. Fig. 5-23 illustrates a three-dimensional subneurone (n_a=3) that generates a maximal hardware implementation because the hyperplane crosses all subquadrants except (x_i,y_i,z_i)=(1,1,1). In this subquadrant, the subneurone output is '1'. The maximal number of nodes in the binary tree, except the neurone output, is:

$$\sum_{i=1}^{n_a} 2^i = 2^{n_a+1} - 2 \qquad \textbf{(5-50)}$$

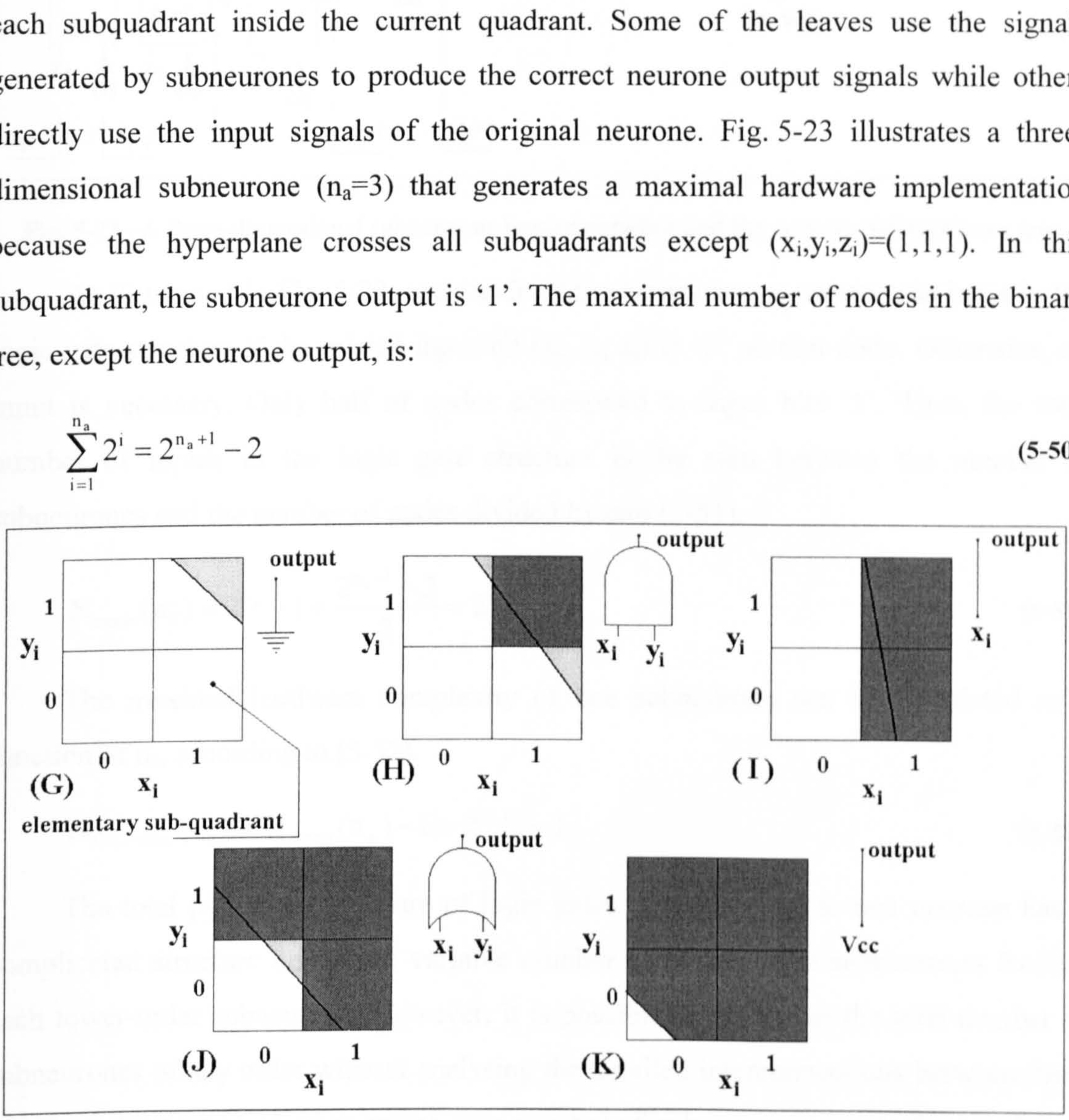

Fig. 5-22 – Types of n_b-order subneurones

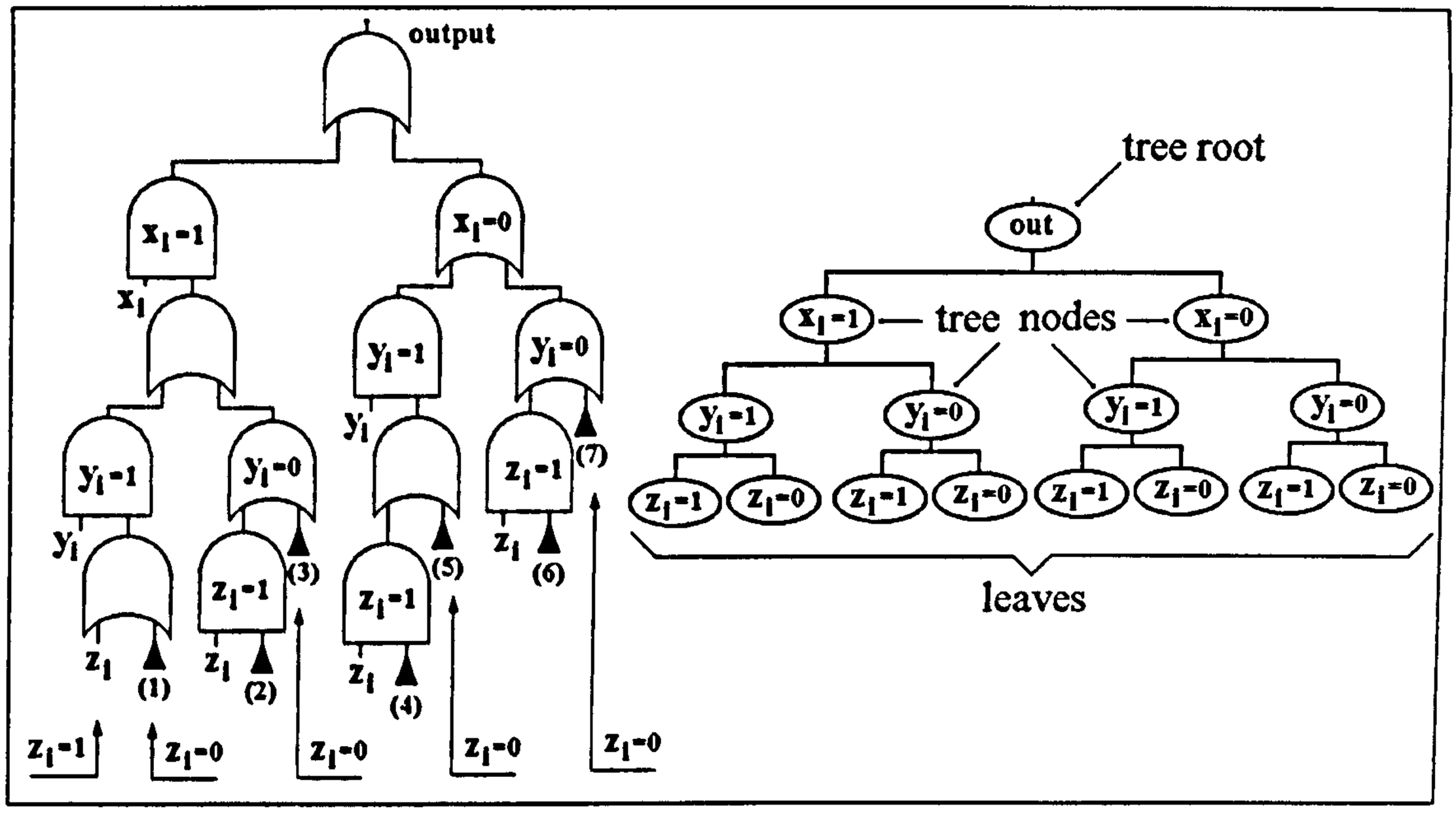

Fig. 5-23 –A three-dimensional subneurone implementation and the corresponding binary tree

As illustrated in Fig. 5-23, each node of the binary tree generates one input in the logic gate structure if the related input bit (x_i, y_i, z_i) is '1' on that node. Otherwise, no input is necessary. Only half of nodes correspond to input bits '1'. Thus, the total number of inputs in the logic gate structure is the sum between the number of subneurones and the number of nodes divided by two (5-51).

$$N_{inputs}(n_a) = 2^{n_a} - 1 + \frac{2^{n_a+1} - 2}{2} = 2^{n_a+1} - 2 \qquad (5\text{-}51)$$

The maximal hardware complexity of one subneurone can be calculated as a function of n_a, according to (5-52).

$$N_{GT2\text{-}max}(n_a) = N_{inputs}(n_a) - 1 = 2^{n_a+1} - 3 \qquad (5\text{-}52)$$

The total pyramidal structure of logic gates corresponding to one neurone has a complicated structure due to the variable number of higher-order subneurones feeding each lower-order subneurone. However, it is possible to determine the total number of subneurones of any order without analysing the detailed interconnections between them and the subneurones of other orders. The number of subneurones equals the number of subquadrants of corresponding size that are crossed by the hyperplane in the input data space. In a two-dimensional case, the subquadrants are square-shaped and there are 2^{i-1} subquadrants on each side of the square input data space. The maximal number of crossed subquadrants is $2 \cdot 2^{i-1}$-1. In a n_a-dimensional data space the subquadrants are cubes (n_a=3) or hypercubes (n_a>3) and the previous result can be generalised to:

$$N_{subn}(i) = 2 \cdot \left[2^{(i-1)}\right]^{(n_a-1)} - 1 = 2^{(i-1)\cdot(n_a-1)+1} - 1 < 2^{(i-1)\cdot(n_a-1)+1} \quad \textbf{(5-53)}$$

The generalisation is based on the fact that the hyperplane can be projected on a base with n_a-1 dimensions upon which lie a number of maximum $2^{(i-1)\cdot(na-1)}$ quadrants. Each of these quadrants is the bottom of a n_a-dimensional prism containing at most two subquadrants that are crossed by the hyperplane. Fig. 5-24 illustrates the two-dimensional and the three-dimensional situations.

Therefore, the total number of subneurones is given by equation (5-54).

$$N_{total-subn} = \sum_{i=1}^{n_b} 2 \cdot 2^{(n_a-1)\cdot(i-1)} - 1 < 2 \cdot \frac{2^{(n_a-1)\cdot n_b} - 1}{2^{n_a-1} - 1} \quad \textbf{(5-54)}$$

Based on the previous results, an absolute upper limit of the total implementation complexity of all the analogue neurones can be calculated as in (5-55). This calculation does not take into account the fact that order-n_b subneurones have a smaller hardware complexity.

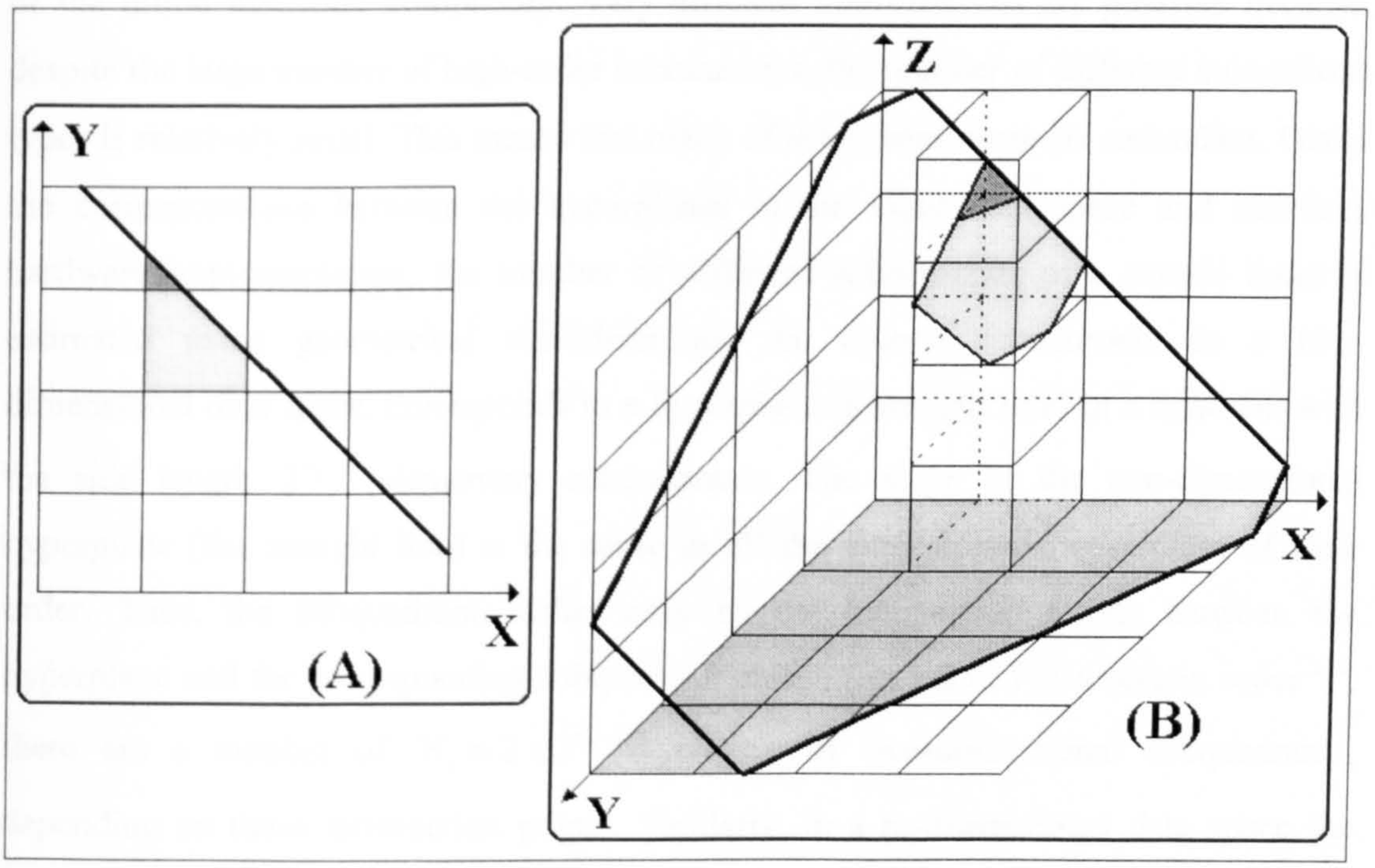

Fig. 5-24 – The intersection between the hyperplane and the subquadrants of second order (i=2) in the two-dimensional case (A), and three-dimensional case (B).

$$N_{GT2-total} = N_{neur} \cdot N_{total-subn} \cdot N_{GT2-max} < N_{neur}\left(2 \cdot \frac{2^{n_a n_b - n_b} - 1}{2^{n_a-1} - 1}\right) \cdot \left(2^{n_a+1} - 3\right) \quad \textbf{(5-55)}$$

The expression (5-55) can be replaced with the higher maximal limit that has a simpler expression, as in (5-56).

$$N_{GT2-total} < N_{neur}\left(2\cdot\frac{2^{n_a n_b - n_b}-1}{2^{n_a-2}}\right)\cdot\left(2^{n_a+1}-3\right) < 2^4\cdot N_{neur}\cdot\left(2^{n_a n_b - n_b}-1\right) \quad (5\text{-}56)$$

The order of complexity generated by the adopted implementation method is given by (5-57).

$$O\left(N_{neur}\cdot 2^{n_a n_b - n_b}\right) \quad (5\text{-}57)$$

This result is superior to the one presented in (5-49) which corresponds to the method presented in [30] because the order of complexity (5-57) is smaller than the order of complexity (5-49). Therefore, the implementation of large neural networks designed with Voronoi diagrams is most efficient using the implementation strategy presented in section 5.1.3 even if no hardware optimisation is carried out.

5.3.2.2 Optimised Implementations

The optimisation process presented in section 5.2 decreases even further the level of the initial hardware complexity. Very efficient optimisations are possible because, despite the large number of high-order subneurones, the number of different subneurone types is relatively small. This means that many of the subneurones are redundant. Given the correspondence between the hyperplanes in the input data space and neurones hardware implementation, the number of different subneurones of a certain order is estimated using geometrical considerations. An order-i subneurone, in a two-dimensional data space, corresponds to a hyperplane (a straight line) in a quadrant with the side length 2^{n_b-i+1} elementary subquadrants. The slope of the two-dimensional hyperplane (the straight line) is the same in all the subquadrants, regardless of their order. Thus, the subquadrants differ only by the intersection points between the hyperplane and the corresponding subquadrant sides (Fig. 5-25). For a certain order 'i', there are a number of $N_c = 2 \times 2^{n_b-i+1}$ classes of two-dimensional subquadrants, depending on these intersection points. Similarly, in a n_a-dimensional data space the number of classes is $N_c = n_a \cdot 2^{n_b-i+1}$.

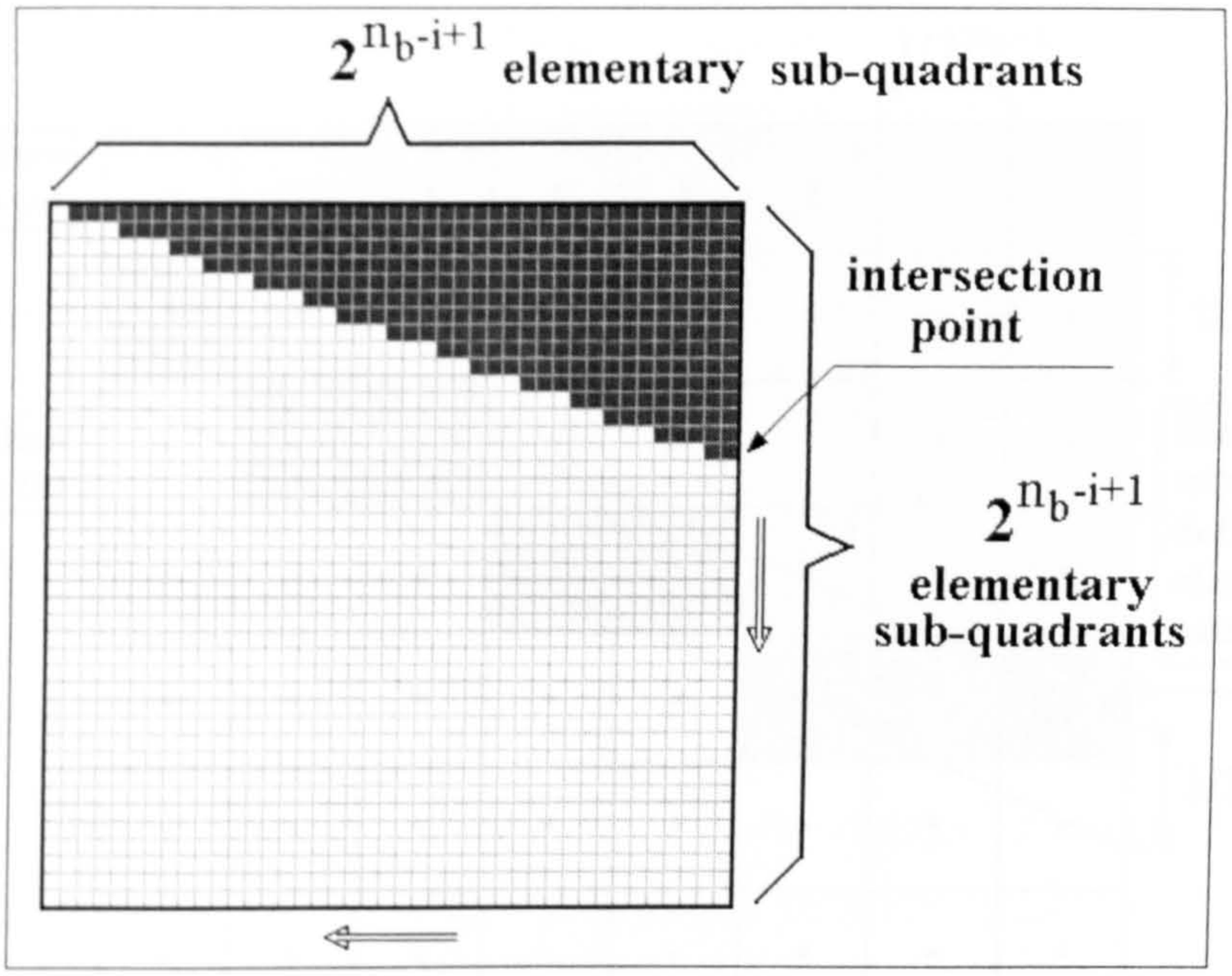

Fig. 5-25 – Example of order-i subquadrant

The subquadrants in the same class can differ by the exact shape of the boundary between the active region and the inactive region. The shape of this boundary depends on the slope and position of the hyperplane and it contains a precisely determined sequence of steps as in the example presented in Fig. 5-25. The maximal number of different step patterns for a given class of subquadrants can be calculated using algebraic and geometrical considerations. First, this upper limit is determined in a two-dimensional input data space and then the result is generalised for an n-dimensional situation.

The step pattern in each subquadrant of order 'i' depends on the exact position of the straight line in the subquadrant. An elementary subquadrant is included in the active region of the two-dimensional input data space if more than half of its surface is situated on the active side of the hyperplane defining the neurone. In a two-dimensional case, the hyperplane is reduced to a straight line. The inclusion or the exclusion of each elementary quadrant can be determined by analysing the position of its centre. If the centre lies on the active side of the hyperplane then it has to be included in the active region of the input data space. Otherwise, it is excluded form the active region of the input data space. All the step patterns included in one of the N_c classes contain a common elementary quadrant on one side of the current subquadrant. This elementary subquadrant is determined by the intersection point between the hyperplane and the side of the subquadrant, as shown in Fig. 5-26.

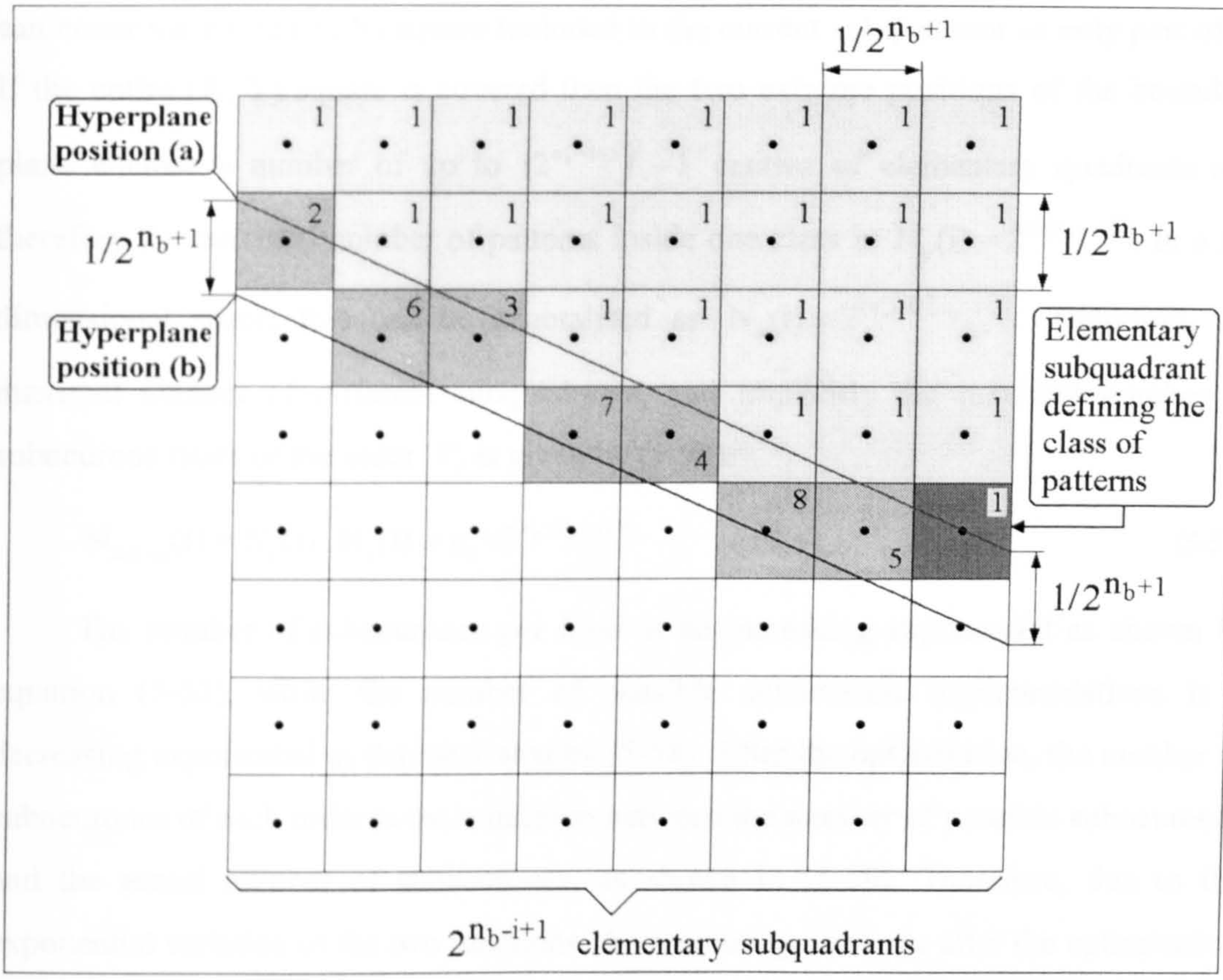

Fig. 5-26 – The step patterns included in one class of order 'i' subquadrants

The exact position of the intersection point can vary with as much as $1/2^{nb+1}$ for one class of patterns, and it determines which other elementary subquadrants are included in the active region. This corresponds to two extreme hyperplane positions (a) and (b), presented in Fig. 5-26. When the hyperplane is in position (a) then only the elementary subquadrants marked by number '1' in Fig. 5-26 are included in the active region. If the hyperplane position changes continuously from position (a) to position (b) then new elementary quadrants are included in the active region in the order determined by the distance between the corresponding centres and the initial hyperplane (a). This order is indicated for the example in Fig. 5-26 by the numbers 2 through 7. The inclusion of each elementary quadrant generates a new pattern included in the current class. In the end, when the position (b) is attained, a maximal number of $2^{n_b-i+1}-1$ new elementary subquadrants have been added. This means that in a two-dimensional input data space the maximum number of patterns in a class is $N_p(i) = 2^{n_b-i+1}$.

In a three-dimensional situation, the hyperplane is immersed in a three-dimensional data space as initially presented in Fig. 5-24 (B). The elementary subquadrants are in this case cubes grouped together into prisms with square bases included in the (X, Y) plane. Depending on its slope, the projection of the hyperplane

can cover the entire (X, Y) square included in the current subquadrant or only part of it. If the entire (X, Y) square is covered then the two extreme positions of the boundary plane enclose a number of up to $\left(2^{n_b-i+1}\right)^2-1$ centres of elementary quadrants and therefore the maximal number of patterns inside one class is $N_p(i)=2^{2\cdot(n_b-i+1)}$. In a n_a-dimensional space, this can be generalised as $N_p(i)=2^{(n_a-1)\cdot(n_b-i+1)}$. Therefore, the maximal number of different subquadrants, and implicitly the maximal number of subneurone types of the order 'i', is given by (5-58).

$$N_{sub-q}(i)=N_c(i)\cdot N_p(i)=n_a\cdot 2^{n_a\cdot(n_b-i+1)} \tag{5-58}$$

The number of subneurones per layer is an increasing exponential as shown by equation (5-53), while the number of possible subneurone implementations is a decreasing exponential as demonstrated by (5-58). After the optimisation, the number of subneurones of each order is the minimum between the number of possible subneurones and the actual number of subneurones, as shown in (5-59). Therefore, due to the exponential variation of the two functions, the remaining neurones after the optimisation are very few as compared to the initial number of neurones. This situation is illustrated by the example in Fig. 5-27.

$$N_{subn-opt}(i)=\min\left\{N_{subn}(i);N_{sub-q}(i)\right\}=\min\left\{2^{(n_a-1)(i-1)+1};\ n_a\cdot 2^{n_a(n_b-i+1)}\right\} \tag{5-59}$$

$$N_{subn-opt}(i)=\min\left\{N_{subn}(i);N_{sub-q}(i)\right\}=\min\left\{2^{(n_a-1)(i-1)+1};\ 2^{n_a(n_b-i+1)+\log_2 n_a}\right\} \tag{5-60}$$

The intersection point between the two graphs illustrated by Fig. 5-27 is placed at the location corresponding to equal exponents in (5-60). This is given by the solution of the equation (5-61).

$$(n_a-1)\cdot(i-1)+1=n_a(n_b-i+1)+\log_2 n_a \tag{5-61}$$

To compare the present algorithm with the algorithm presented in [30], the limit situation, with n_a and n_b having very large values, is analysed in (5-62).

$$\lim_{n_a;n_b\to\infty}\{i\}=\lim_{n_a;n_b\to\infty}\left\{\frac{n_a\cdot n_b+2n_a+\log_2 n_a-2}{2n_a-1}\right\}=\frac{n_b}{2} \tag{5-62}$$

Therefore, the number of optimised subneurones can be calculated as in equations (5-63), (5-64) and (5-65).

$$N_{subn-opt} < \sum_{i=1}^{n_b/2} 2 \cdot 2^{(n_a-1)(i-1)} + \sum_{i=n_b/2+1}^{n_b} n_a \cdot 2^{n_a(n_b-i+1)} \quad \textbf{(5-63)}$$

$$N_{subn-opt} = 2 \cdot \frac{2^{(n_a-1)\cdot\frac{n_b}{2}} - 1}{2^{n_a-1} - 1} + n_a \cdot \left(\frac{2^{n_a \cdot \left(\frac{n_b}{2}+1\right)} - 1}{2^{n_a} - 1} - 1 \right) \quad \textbf{(5-64)}$$

$$N_{subn-opt} \approx 2^{(n_a-1)\cdot\frac{n_b}{2}-n_a+2} + n_a \cdot 2^{n_a\frac{n_b}{2}} - n_a \quad \textbf{(5-65)}$$

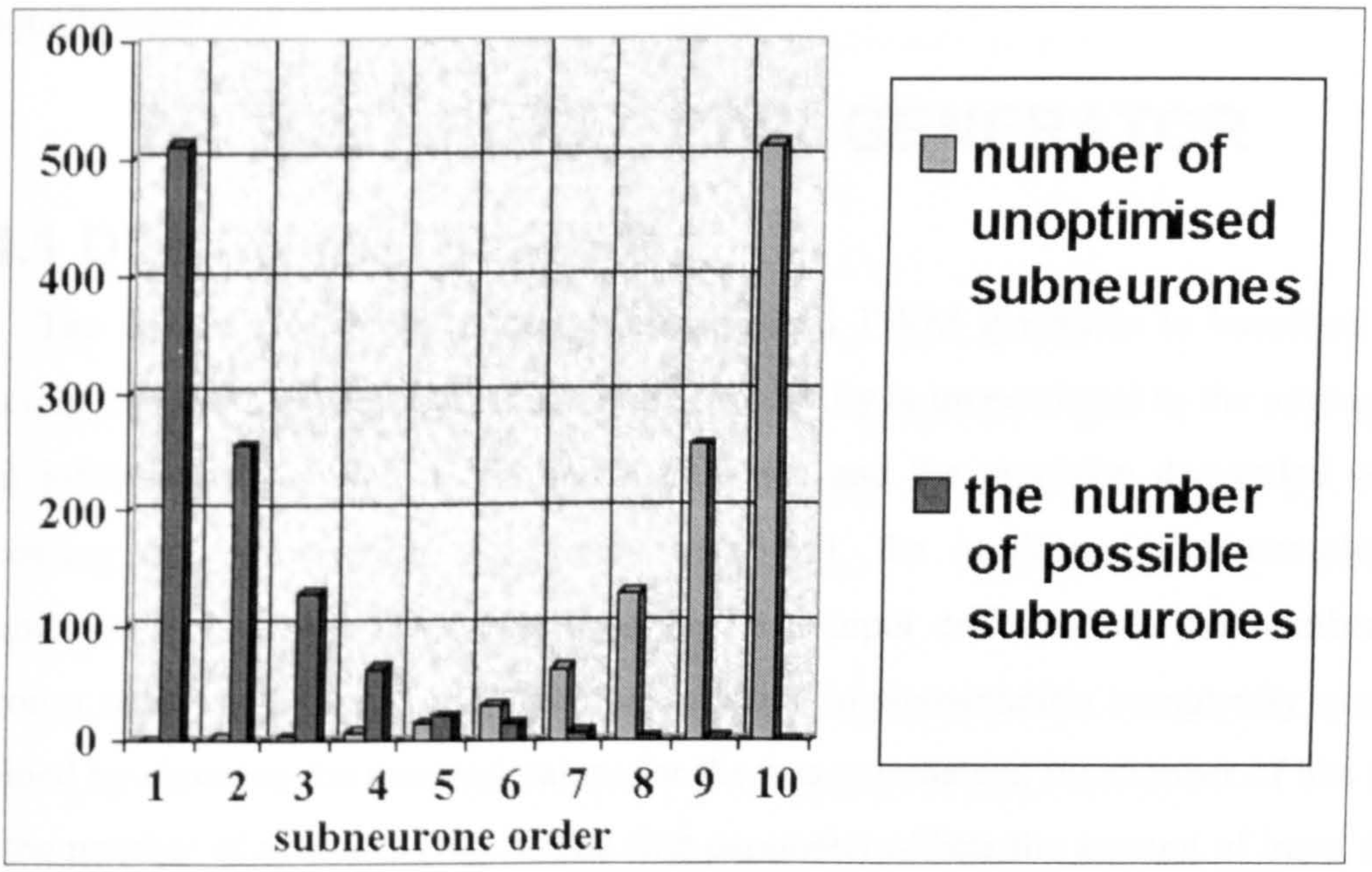

Fig. 5-27 – Graphical representation of the optimisation process (na=2, nb=10)

The resulting total complexity is given by (5-66).

$$N_{LG2-total} = N_{neur} \cdot N_{subn-opt} \cdot \left(2^{n_a+1} - 3\right) \approx N_{neur} \cdot \left(2^{\frac{n_a n_b}{2} - \frac{n_b}{2} + 3} + n_a \cdot 2^{n_a\frac{n_b}{2} + n_a + 1} \right) \quad \textbf{(5-66)}$$

Therefore, the resulting order of complexity is (5-67).

$$\boldsymbol{O}\left(N_{neur} \cdot n_a \cdot 2^{\frac{n_a n_b}{2} + n_a} \right) \quad \textbf{(5-67)}$$

This result did not take into account the cross-neurone optimisations, which can bring further hardware reduction. Still, it shows that the increase of the network implementation with parameters n_a and n_b is much slower in the case of the new implementation method, as compared with the standard method presented in [30]. The result in (5-67) is slightly larger than the square root of the initial result (5-49). This

implies a **substantial** gain in terms of hardware complexity reduction for large neural networks.

The method in [30] considers a particular case of Voronoi algorithm where all the hyperplanes are parallel to the axes in the input data space for each the generated order of complexity is reasonable. If a general Voronoi approach using oblique hyperplanes is necessary then the method [30] generates much larger implementations than the one presented in this chapter. In conclusion, the present implementation method is the most adequate for ANN network implementation designed using Voronoi diagrams with oblique hyperplanes.

5.4 THE NEURAL PWM GENERATOR

5.4.1 DESIGN GUIDELINES

The design philosophy underlying the neural PWM generator is based on the principle that the hardware implementation complexity is proportional to the amount of input information supplied to the neural network, and the precision demanded when processing this information. As shown in (5-66), the implementation complexity depends on the number n_b of bits used for each input code and on the number of neurones used by the network. Therefore, a good implementation complexity can be obtained by choosing the minimal values for the two parameters: the number of bits (n_b) and the number of neurones (N_{neur}). The first parameter affects the amount of input data while the other controls the calculation accuracy.

The use of a limited number of bits to code the analogue input signals causes digitisation errors that alter the position of the corresponding point in the input data space (the input-data point). If the input-data point is situated close to a boundary of a Voronoi cell, then the errors caused by digitisation become significant and they can cause the point in the data space to cross the boundary of the correct Voronoi cell. The result is that the network will generate incorrect output signals.

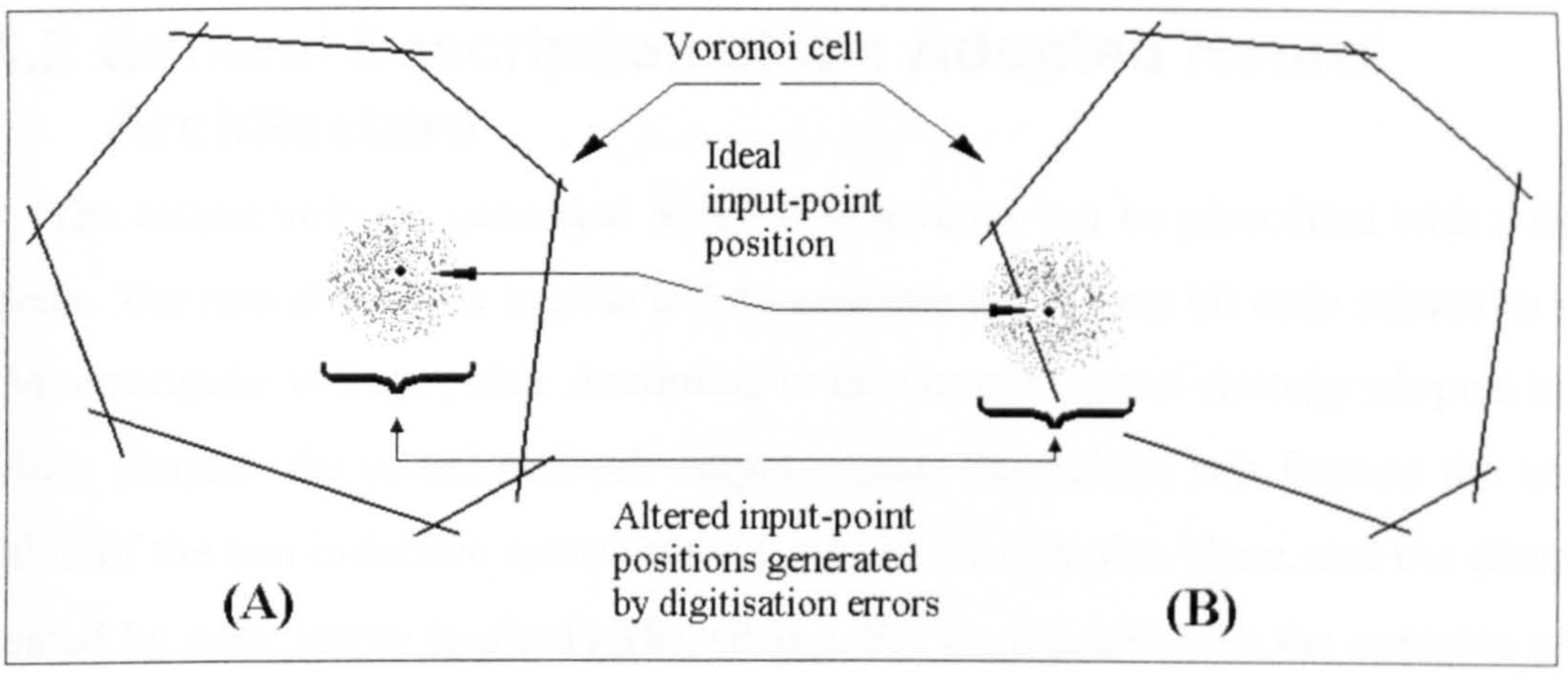

Fig. 5-28 – The effect of digitisation errors onto the neural network behaviour: (A) the errors are irrelevant; (B) the errors can cause incorrect output signals

Most practical applications require that the cell boundaries are curved hypersurfaces in the input data space, while the neurones model only hyperplanes. Therefore, the required boundaries are approximated by successions of hyperplanes. This generates differences between the real cell boundaries and the ideal ones. Therefore, the limited calculation precision associated with the limited number of neurones cause imperfect Voronoi cells. The smaller the number of neurones, the larger the calculation errors.

However, the behaviour of the drive system is influenced by the global properties of the PWM signal (the harmonic content). The individual voltage pulses in the PWM signal have a small influence on its harmonic content. This allows the use of imperfect control strategies, as long as the probability for errors is sufficiently low to have a negligible effect on the statistical properties of the output signals. Based on the previous considerations, the achieved neural control can be assessed by the probability of generating an erroneous output. A low probability is associated with a high control quality, while a high probability is associated with low control quality. The current control quality can be adjusted by varying the number of neurones in the network and the number of input bits for each neurone.

Therefore, the neural approach allows the adaptation of the price-performance ratio to the requirements of a wide range of applications. High quality control can be obtained with complex neural architectures containing a large number of neurones. On the other hand, inexpensive controllers can be produced by using lower precision neural networks that contain a limited number of neurones.

5.4.2 General Description of the Adopted Neural Architecture

The output voltages generated by a PWM inverter can be associated with a three-bit code. The neural network has the task to generate the correct bit code related to each PWM rectangular voltage pulse. According to the current control strategy adopted in the previous chapter, the neural network output signals depend on two factors: the vertex position of the non-inductive space vector $\underline{V}_{ni}(k)$ in the complex plane, and the direction indicated by error vector $[\underline{i}_{ref}(k+1)-\underline{i}(k)]=R\Delta\underline{i}_{ref}$. For each position in the complex plane and for each direction, one specific set of control signals is generated to the inverter.

The actual current $\underline{i}(k)$, the reference current $\underline{i}_{ref}(k+1)$ and the non-inductive voltage $\underline{V}_{ni}(k)$ are complex quantities treated as pairs of real values. This implies the construction of a four-dimensional Voronoi diagram: two dimensions correspond to current error vector, while the other two are the components of vector $\underline{V}_{ni}(k)$. To simplify the design process, the network has been decomposed into functional modules. Each module was then designed separately by means of two-dimensional Voronoi diagrams. The novel architecture is defined by three interconnected subnetworks (Fig. 5-29). The first neural component determines the position of the non-inductive voltage space vector $\underline{V}_{ni}(k)$ in the complex plane, while the second determines the direction of the current error vector $\Delta\underline{i}_{ref}$. The third subnetwork merges the two results and generates a three-bit code associated with one of the output voltages of the PWM inverter.

The first two subnetworks are designed by means of Voronoi diagrams. Therefore, the initial four-dimensional Voronoi diagram has been replaced by a pair of two-dimensional diagrams. The two-dimensional diagrams are the projections of the four-dimensional one in two perpendicular planes. Each combination of four real inputs corresponds to two input-data points in two different diagrams. They are the projections of the unique input-data point in the four-dimensional input data space.

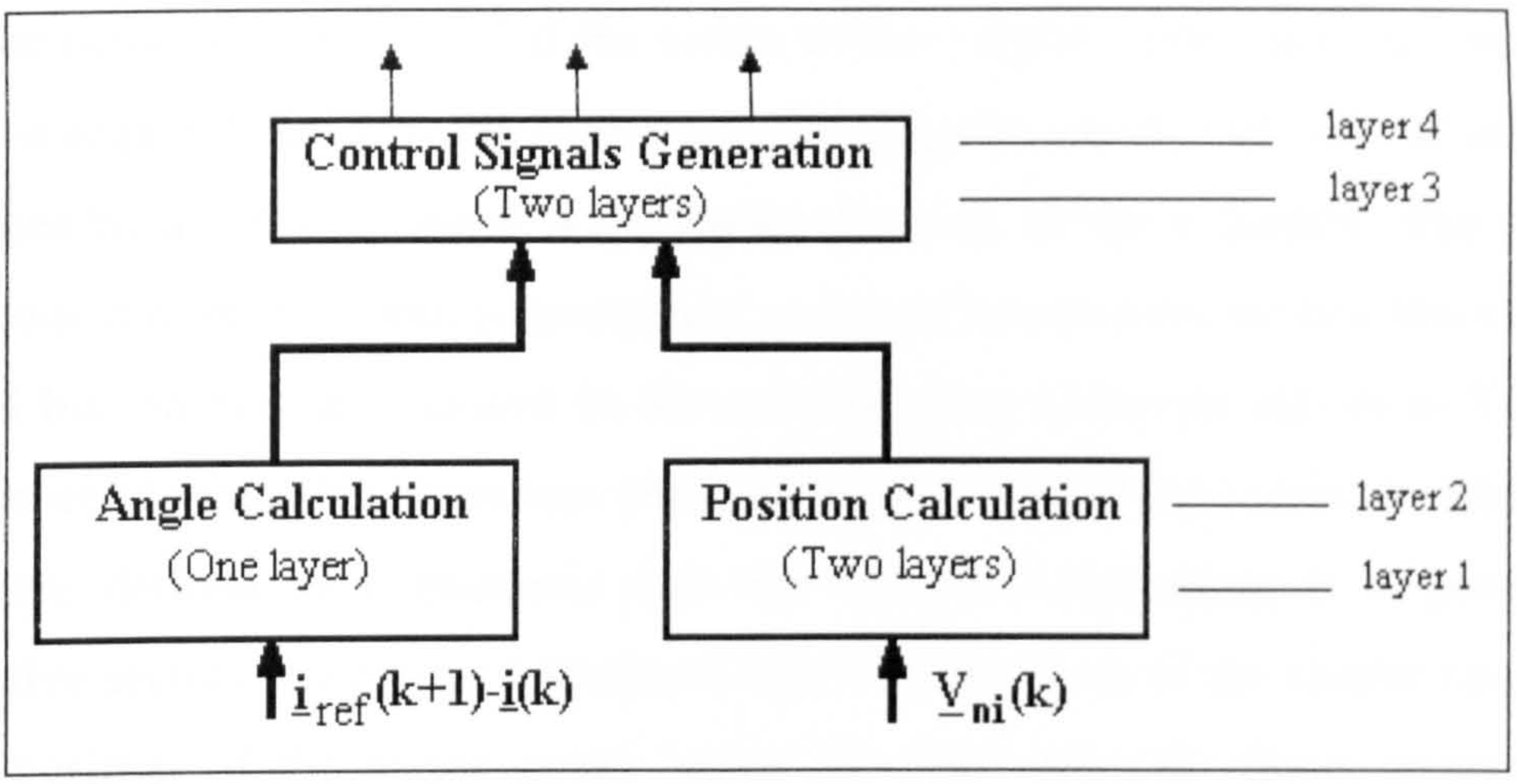

Fig. 5-29 - The architecture of the neural network

The adopted implementation solution is as follows: the angle subnetwork determines the argument of the error vector $\Delta\underline{i}_{ref}$ with a precision of $\pm\alpha^0$, while the position subnetwork divides the complex plane into 'm' polygonal cells and determines the cell which includes the vertex of $\underline{V}_{ni}(k)$. The generation of the control signals to be supplied to the PWM inverter considers that the vertex of $\underline{V}_{ni}(k)$ is located in the centre of the corresponding cell, and the calculation is performed accordingly.

5.4.3 The Angle Subnetwork

This subnetwork uses a number of 'n' neurones placed within a single layer to divide the complex plane into '2n' sectors (Fig. 5-30). The calculation error '$\Delta\varepsilon$' is related to the number of neurones 'n' according to equation (5-68):

$$\Delta\varepsilon = \frac{360^o}{2 \cdot n} \tag{5-68}$$

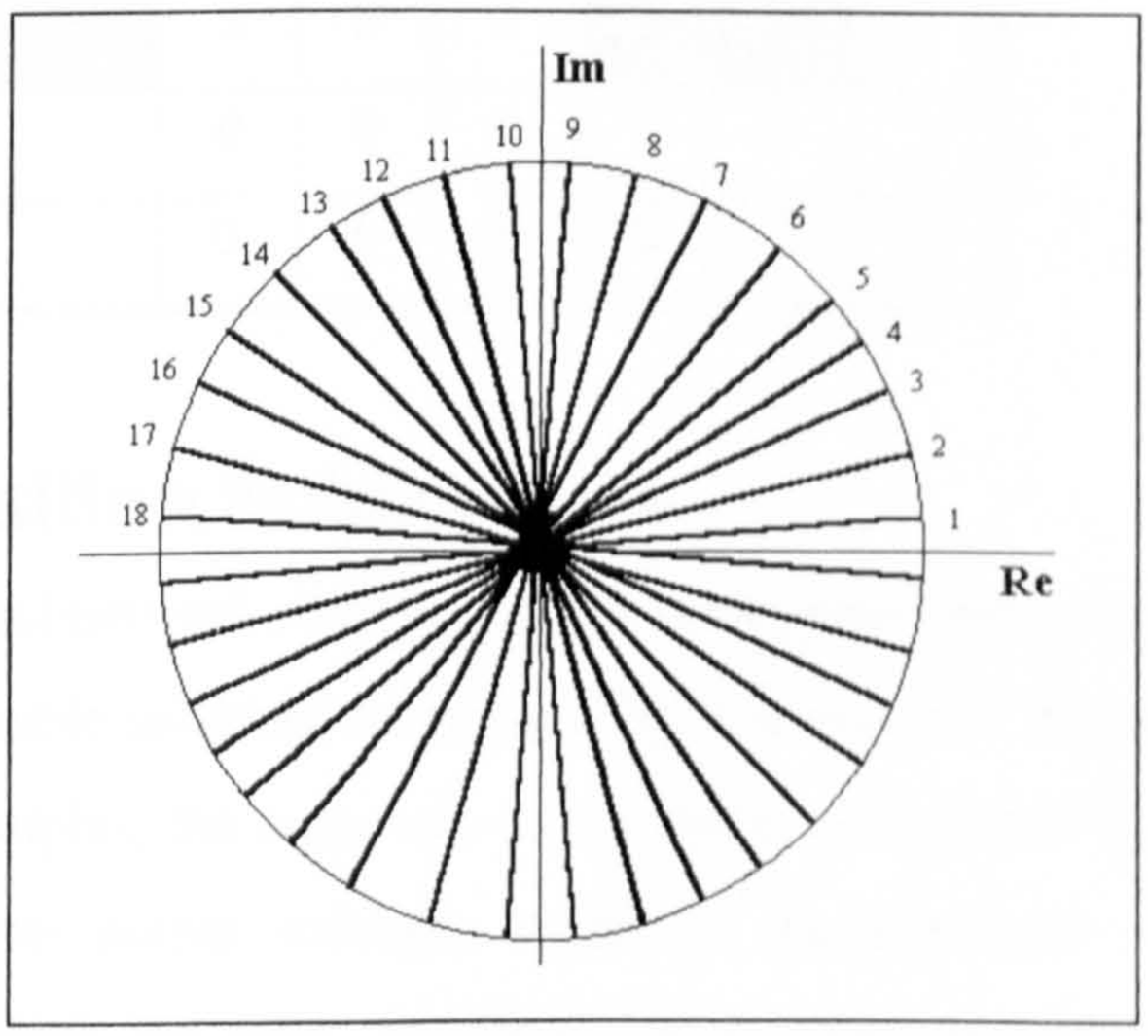

Fig. 5-30 - The division of the complex plane into sectors (angular Voronoi cells)

The neurone output is '1' if the vertex of the current error vector is located into the active region defined by the neurone in the complex plane, and it is '0' otherwise. Consequently, a different result is generated for each of the n sectors. The obtained binary code has an important property: any group of consecutive sectors shares several identical bits on certain positions in the corresponding codes, as shown in Table 5-2. The number of shared bits decreases when the width of the group increases. Thus, if the sectors are defined by n neurones then the codes corresponding to a group of m consecutive sectors share n-m+1 identical bits. The positions of the shared bits depend on the position of the sector group inside the 360^0 interval. These properties are exploited by the control signal subnetwork described in section 5.4.5, and by the implementation of the on-line inductance estimator.

Table 5-2 - The codes generated by an angle subnetwork with n=6 neurones

Angle Interval	Code					
$[-15^0; 15^0)$	0	0	0	0	0	0
$[15^0; 45^0)$	1	0	0	0	0	0
$[45^0; 75^0)$	1	1	0	0	0	0
$[75^0; 105^0)$	1	1	1	0	0	0
$[105^0; 135^0)$	1	1	1	1	0	0
$[135^0; 165^0)$	1	1	1	1	1	0
$[165^0; 195^0)$	1	1	1	1	1	1
$[195^0; 225^0)$	0	1	1	1	1	1
$[225^0; 255^0)$	0	0	1	1	1	1
$[255^0; 285^0)$	0	0	0	1	1	1
$[285^0; 315^0)$	0	0	0	0	1	1
$[315^0; 345^0)$	0	0	0	0	0	1

5.4.4 The Position Subnetwork

The position subnetwork divides the complex plane into polygonal cells. There is a large range of possible solutions of performing this division. According to the adopted current control principles, the argument of the difference vector $\underline{V}_{ni}$-u_{PWM}, where $\underline{u}_{PWM}$ is one of the inverter output voltages, equals to the argument of the corresponding current variation across the load. To ensure good operation accuracy, the arguments corresponding to the same voltage $\underline{u}_{PWM}$ but to different vectors $\underline{V}_{ni}$ inside the same

Voronoi cell, should have almost equal values. Using geometrical considerations, three conclusions can be drawn:

1. Increasing the number of cells increases the operation accuracy of the neural network.
2. The point in the complex plane corresponding to an inverter output voltages $\underline{u}_{PWM}$ cannot be included in any Voronoi cell. If such a point were included in a cell then $\arg\{\underline{V}_{ni}-\underline{u}_{PWM}\}$ varies between 0° and 360° for different vectors $\underline{V}_{ni}$ in the same cell. Therefore, these point need to be part of the cell boundaries.
3. As a consequence of the previous point, the vectors $\underline{V}_{ni}$ situated close to a voltage vector $\underline{u}_{PWM}$ should be separated into a large number of cells, the criterion of separation being the value of $\arg\{\underline{V}_{ni}-\underline{u}_{PWM}\}$. This means that the complex plane division into cell should have a radial structure around each point corresponding to an inverter output voltage. Around such a point, the Voronoi diagram has to be similar to Fig. 5-30.

The adopted solution simultaneously takes into account the previous three points and the need to minimise the number of neurones. Therefore, the division into cells shown in Fig. 5-31 has been chosen. It has the advantage that one neurone can be involved in the radial configuration of two or three different inverter voltages thereby optimising the ratio between the current control quality and the hardware implementation complexity.

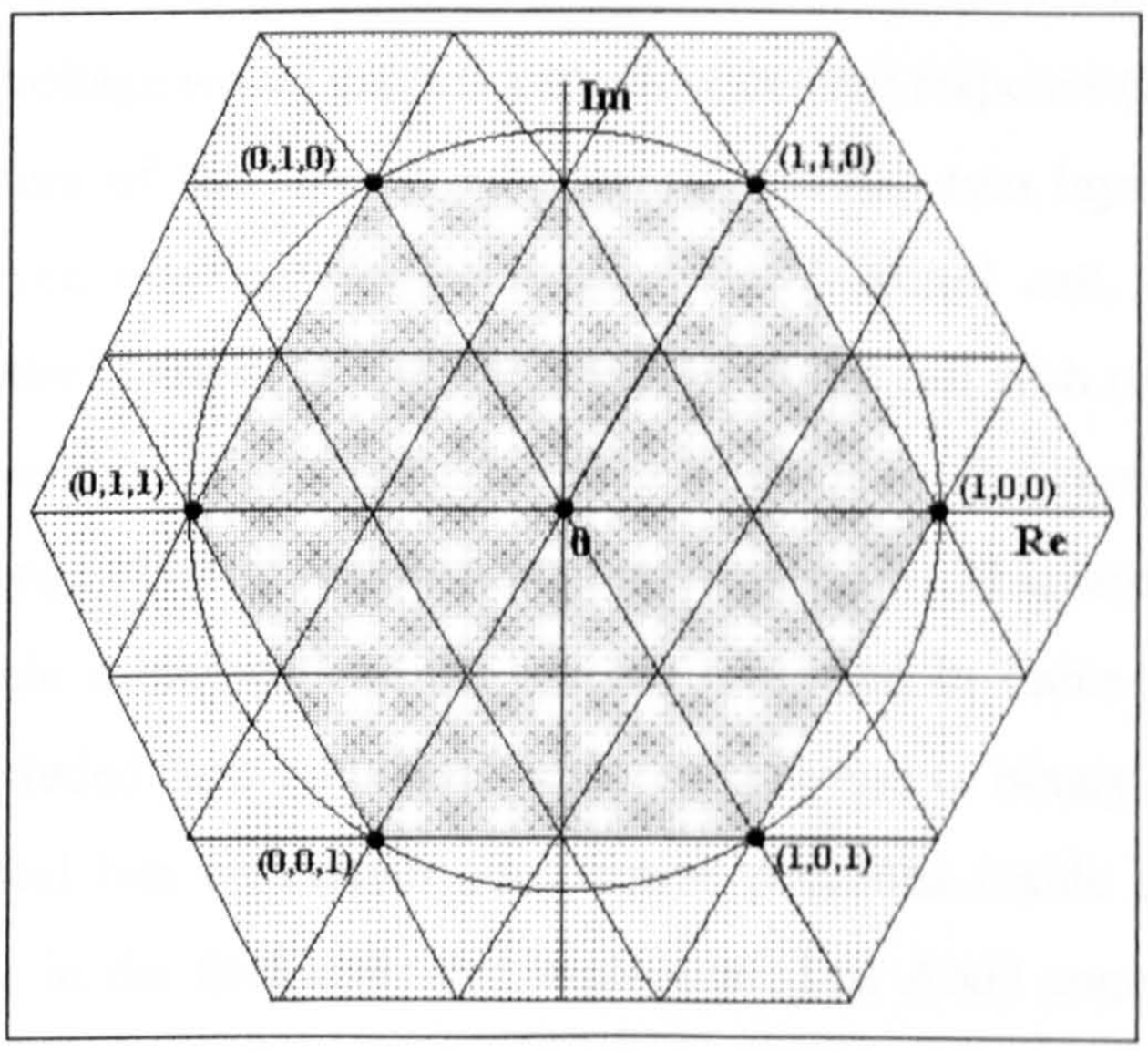

Fig. 5-31 - The partition of the interest area into Voronoi cells

The position subnetwork contains two layers. The first layer models the boundaries of the triangular Voronoi cells. The second layer contains a number of

neurones equal to the number of Voronoi cells. Each neurone is activated if the input-data point is situated inside the associated cell. The output data generated by this subnetwork is therefore a string of N_V bits that contains a single bit '1' and N_V-1 bits '0', where N_V is the total number of Voronoi cells. The neurones in the second layer can be implemented as a combination of NOT gates and 3-input AND gates that are driven by the neurones in the first layer.

5.4.5 The Control Signal Subnetwork

The control signal subnetwork has the task of generating the three-bit output code related to the inverter voltage, using the information generated by the other two subnetworks. For each triangular Voronoi cell, the argument of the current error vector can have values between 0^0 and 360^0. The interval $[0^0; 360^0]$ corresponding to each cell is divided into sectors related to different inverter output voltages. The division into sectors is carried out considering that the vector $\underline{V}_{ni}$ is always situated in the centre of the corresponding triangular Voronoi cell. According to the analysis in the previous chapter, there are two alternative control strategies: the zero output voltage generated by the inverter is either included or excluded from the calculations. Therefore, the 360^0 interval is divided either into six or into seven intervals, depending whether the zero voltage is used or not. The zero voltage is never used by the Voronoi cells in the immediate neighbourhood of the complex plane origin because in this case $|\underline{V}_{ni}|$ is small, and using the zero voltage would cause a very slow current response (Fig. 5-32).

The architecture of the control subnetwork contains two layers. The first layer includes six or seven neurones for each triangular Voronoi cell, depending on the adopted current control version and on the position of the cell with respect to the origin of the complex plane. Each neurone identifies a sector in the complex plane that is associated with a range of error vectors arguments $\arg\{\Delta\underline{i}_{ref}\}$. This argument information is coded by the angle subnetwork in the manner presented in Table 5-2. Therefore, all the angle values included inside a certain sector correspond to binary codes that share a given set of identical bits on certain consecutive positions inside these codes. As a result, the neurones in the first layer are implemented as AND connected to a certain number of NOT gates depending on the numbers of bits '0' and '1' to be tested. An additional input of the AND gate is connected to the output of one neurone in the position subnetwork. The output of this neurone is activated when the vector $\underline{V}_{ni}$ is situated in the correct triangular Voronoi cell.

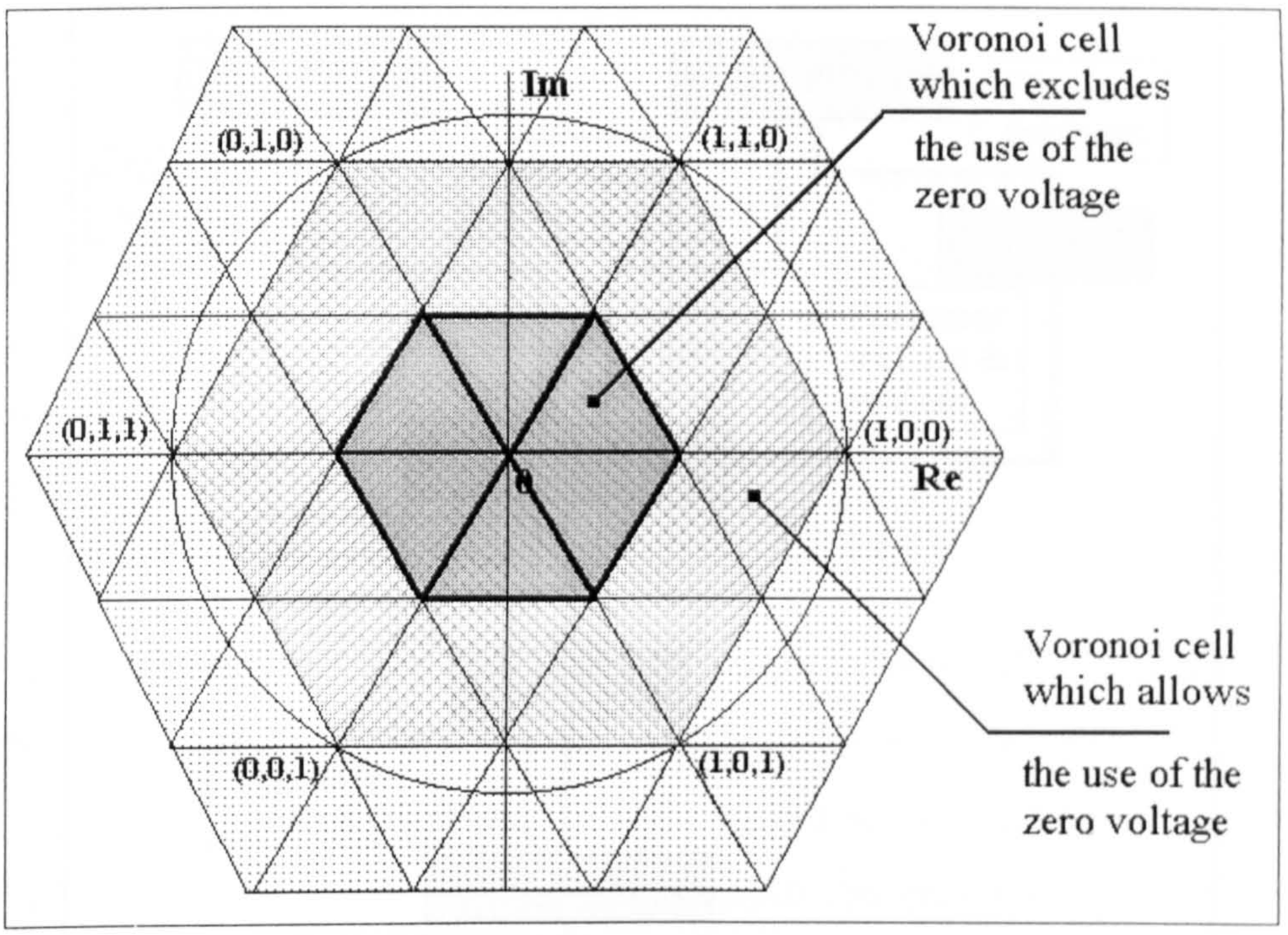

Fig. 5-32 - The triangular cell classification based on their position with respect to the origin

The second layer consists of three neurones generating the three general output bits of the neural network. These neurones multiplex the information supplied by the first layer neurones, and they are implemented as OR logic gates.

5.4.6 The Automated Design Process

A set of three programs has been developed in order to generate the adequate matrix description for the three neural subnetworks. These programs are used in conjunction with the three universal programs: CONV_NET, OPTIM and VHDL_TR (presented in section 5.2) to obtain a complete automation of the neural network design and implementation. The conversion process is monitored by a master program (PWM_GEN) that controls the user interface and calls all the six specialised programs in the correct order. The logical connections between these programs are illustrated by Fig. 5-33. The master program allows the user to control the main parameters of the neural networks to be generated:

- The number of triangular Voronoi cells.
- The number 'n' of sectors used to divide the 360 degrees interval.
- The number of bits used to code the analogue inputs of the angle subnetwork
- The number of bits used to code the analogue inputs of the position subnetwork
- The maximum number of inputs allowed for one gate
- Whether 6 or 7 PWM output voltages are used

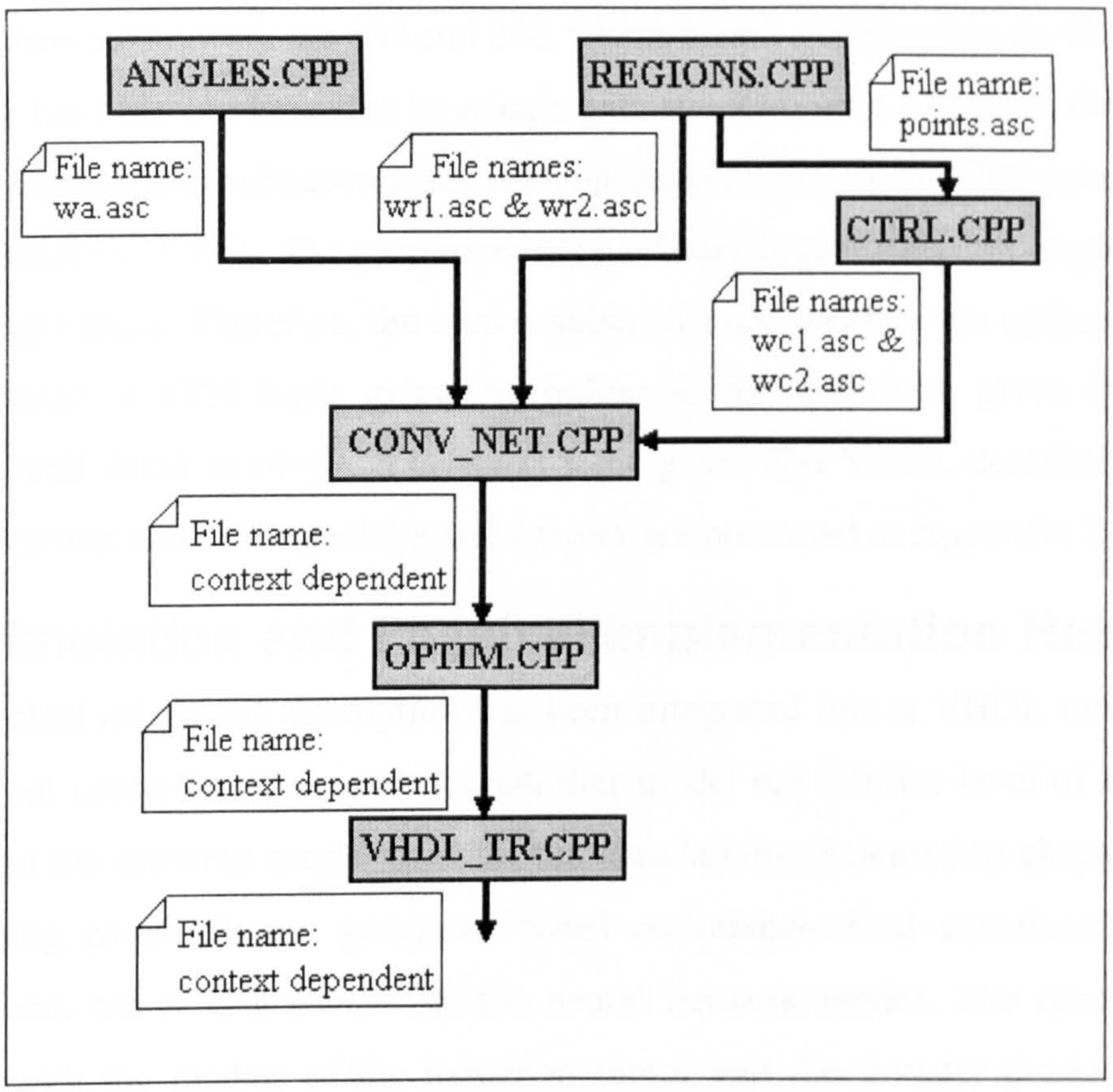

Fig. 5-33 - The neural PWM generator design programs and their interconnections

Note: If the number of triangular Voronoi cells is only 6, the present current control method becomes similar to the control algorithm presented in [106] where the complex plane is divided into 6 regions. The control algorithm in [106] uses very limited information on vector $\underline{e}$ because only the region in the complex plane that includes the vertex of $\underline{e}$ can be calculated. The information on $\underline{V}_{ni}$ used by the present control algorithm is more accurate because the inductance L is estimated on-line in the manner described in the previous chapter.

Alternative architectures defined by different numerical parameters have been tested by means of computer simulation. The solution generating an optimal performance-complexity ratio has been adopted. The implementation solution uses n_b=5 input bits to code each analogue input for both the angle subnetwork and the position subnetwork. The 360^0 interval is divided into 36 sectors while the complex plane is divided into 54 triangular Voronoi cells. The zero voltage is not used (the parameter $|\underline{V}_{ni}|_{crt}$ was considered infinite).

The initial netlist description of the angle subnetwork contained 660 logic gates arranged on 11 gate layers. The netlist was eventually optimised to 378 gates (representing 57.27% of the initial gate count). The initial and the final number of gates

for the position subnetwork are 567 and 242, which means compression to 42.68%. This subnetwork has been implemented by a logic gate structure with 14 layers. On the other hand, the control signal subnetwork has been optimised from 3026 to 709 gates resulting in compression to 23.43%. The corresponding hardware implementation contains only 6 layers of logic gates. Therefore, the total number of logic gates in the optimised neural implementation is 1329 logic gates. According to the definition given in 5.3, the obtained circuit depth is 14+6=20 layers of logic gates. The VHDL descriptions of the angle subnetwork and of the position subnetwork are presented in Appendix B.

5.4.7 Simulation and Physical Implementation Results

The obtained neural description has been integrated into a VHDL model of the motor current controller. This current controller model has a lower level of abstraction compared to the previous model used for the simulations presented in chapter 4. Here the switching pattern is not generated based on mathematical equations as in the previous case, but it is produced by the neural network model. The controller was combined with the models of the induction motor and the inverter presented in the previous chapter, and simulations have been performed to demonstrate the correct operation of the neural network. Fig. 5-34 illustrates the simulation results obtained with the 11.1 kW induction motor parameters. The results prove the correct operation of the neural network, and shows good performance of the novel neural control strategy. The simulation parameters are:

- The switching frequency: 20kHz
- The sampling frequency: 300 kHz
- The reference current frequency was 50Hz
- The reference phase current amplitude is 6.66A, corresponding to a space vector amplitude of 10A.
- $|\underline{V}_{ni}|_{crt}=\infty$ (only six inverter voltages are used, the zero voltage being eliminated)

The operation speed of the neural network has been tested by means of timing simulations using Xilinx Foundation 1.4 software and by practical implementation in a Xilinx XC4010XL FPGA chip which is included on a XS40 test board. To carry out the two tests, a VHDL testbench supplying the neural network with variable input signals has been developed.

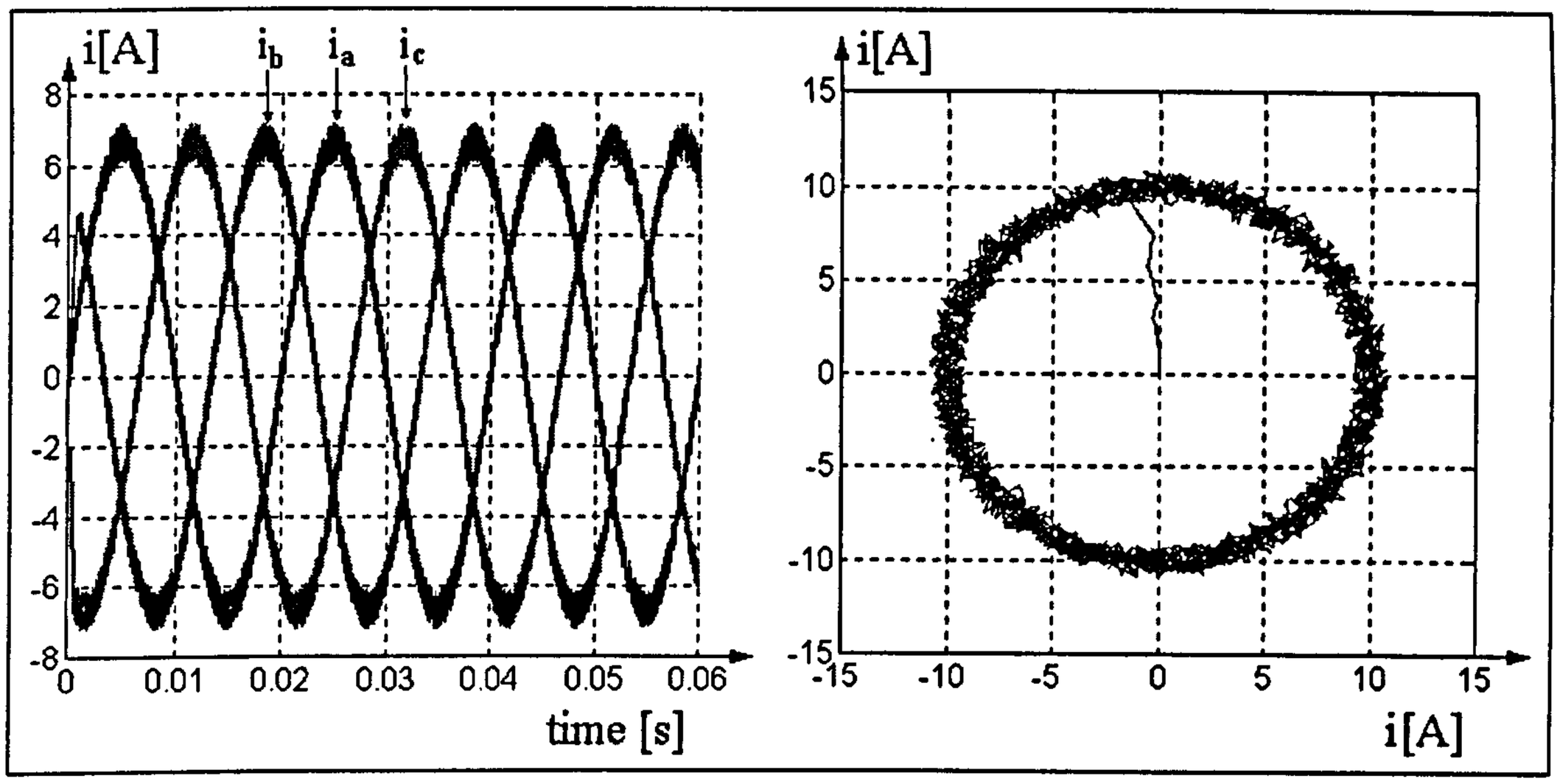

Fig. 5-34 – VHDL simulation results: the three phase currents and the corresponding space vector

The neural network in the adopted configuration requires a total of 20 input bits. The required series of 20-bit input patterns is generated by a 20-bit counter. To obtain a pseudo-random sequence of input patterns, a supplementary block has been inserted in the testbench between the counter and the neural network (Fig. 5-35). This block simply rearranges the 20 bits inside the neural network input pattern. The XS40 test board contains a 12MHz clock oscillator. A supplementary 8-times clock divider has been added to the testbench structure to decrease the initial 12MHz to 1.5MHz as shown in Fig. 5-35.

The least significant output bit of the 20-bit counter changes every time a new input pattern is generated. This signal (named 'CADENCE' in Fig. 5-35) indicates the rhythm of the pattern generation process and it can be used to measure the propagation delays through the neural network. Thus, the propagation delay is the measured time between the edge of signal CADENCE and the moment when signals BX, BY, BZ are stable on the output pins.

The VHDL testbench has been synthesised and downloaded into a Xilinx XC4010XL FPGA chip. The synthesis tool provided by Xilinx Foundation analyses the abstract VHDL model and generates an optimised implementation file (a bitstream file) in accordance with the application requirements and the structural details of the target chip. The software allows two types of optimisation: for speed and for implementation complexity (or chip area). Each of them can be carried out with high or with low computational effort.

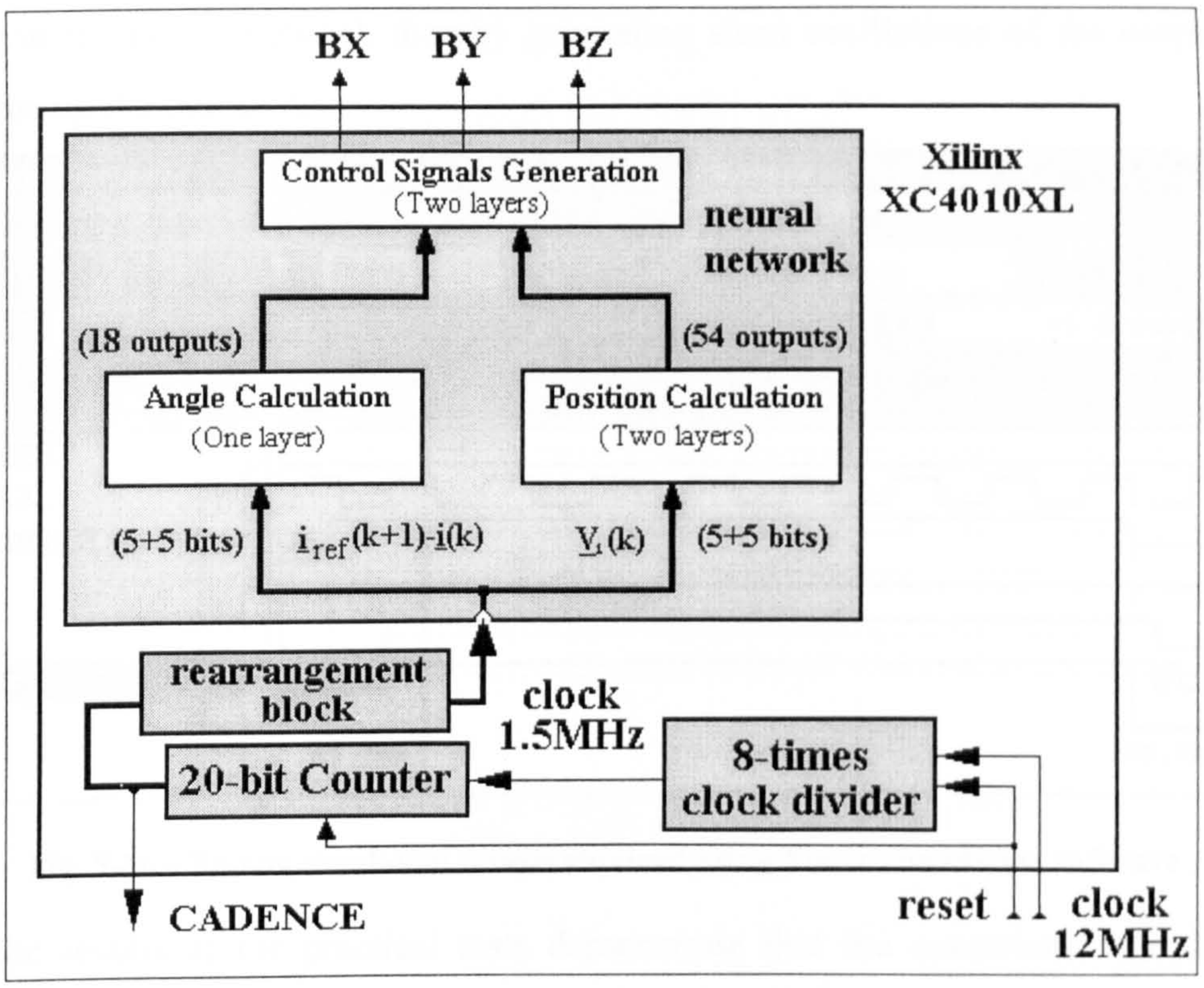

Fig. 5-35 – Testbench for testing the neural network operation speed

The FPGA chip used for this implementation contains 400 Complex Logic Blocks (CLBs). Each CLB includes two D-type flip-flops and three logic function generators. Two of the function generators have 4 inputs each while the third one has only 3 inputs. All the function generators are implemented as look-up tables and therefore the corresponding propagation delays are independent of the Boolean functions.

The synthesis of the complete VHDL testbench was performed using the options of chip area optimisation with high computational effort. The resulting implementation used 192 CLBs, which means 48% of the hardware resources available in the chip. On the other hand, the separate implementation of the neural network consumed only 179 CLBs representing 45% of the chip resources.

After synthesis, timing simulations have been performed to analyse the propagation delays in the FPGA chip. The maximal propagation delay found during the simulation is 110ns, the minimal one is 35ns, but the majority of the delay times are about 50ns. In other words, the propagation time is less than 1.5 clock cycles, which proves the very high operation speed of the neural network. No other digital circuit could perform the same calculations during less than two clock cycles.

Fig. 5-36 presents a fragment of a timing simulation result. It demonstrates that the delay times related to the three output bits are not equal. Furthermore, the three propagation delays vary from one transition of inputs to another. Hazard effects are also

present in the neural network thereby generating short oscillations of the output logic states during the transient.

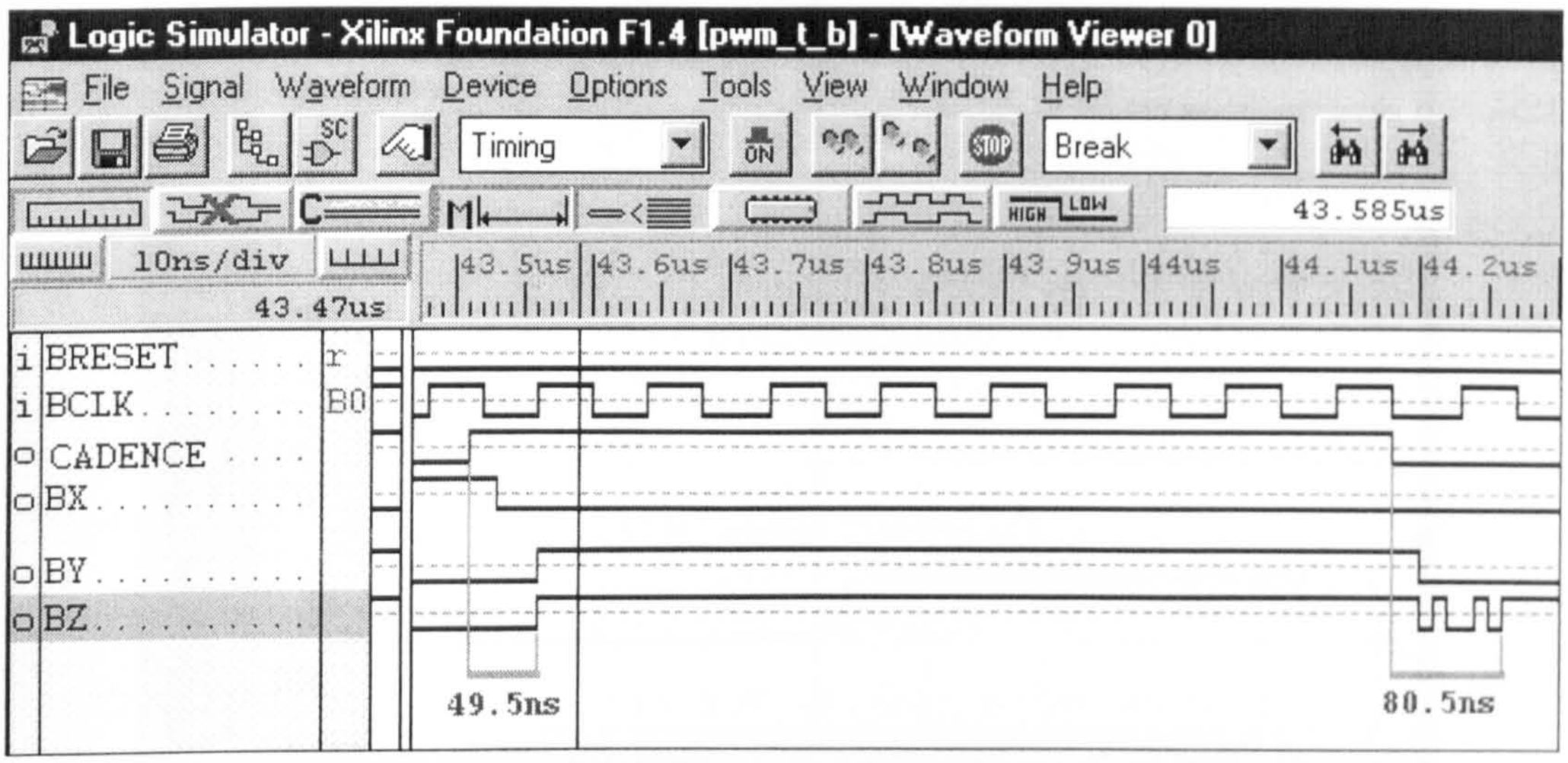

Fig. 5-36 – Timing simulation results obtained using Xilinx Foundation software

The results of the practical tests demonstrate that the operational speed of the neural network is even higher than estimated using timing simulations. Fig. 5-37 shows the waveforms captured with a Hewlett Packard digital oscilloscope using a time scale of 200ns per division. Fig. 5-38 presents the same waveforms at a smaller time scale (50ns per division) so that the propagation delays can be easily determined. All the practical measurements have led to the conclusion that the propagation delay is less than 80ns which is equivalent to less than a single clock cycle.

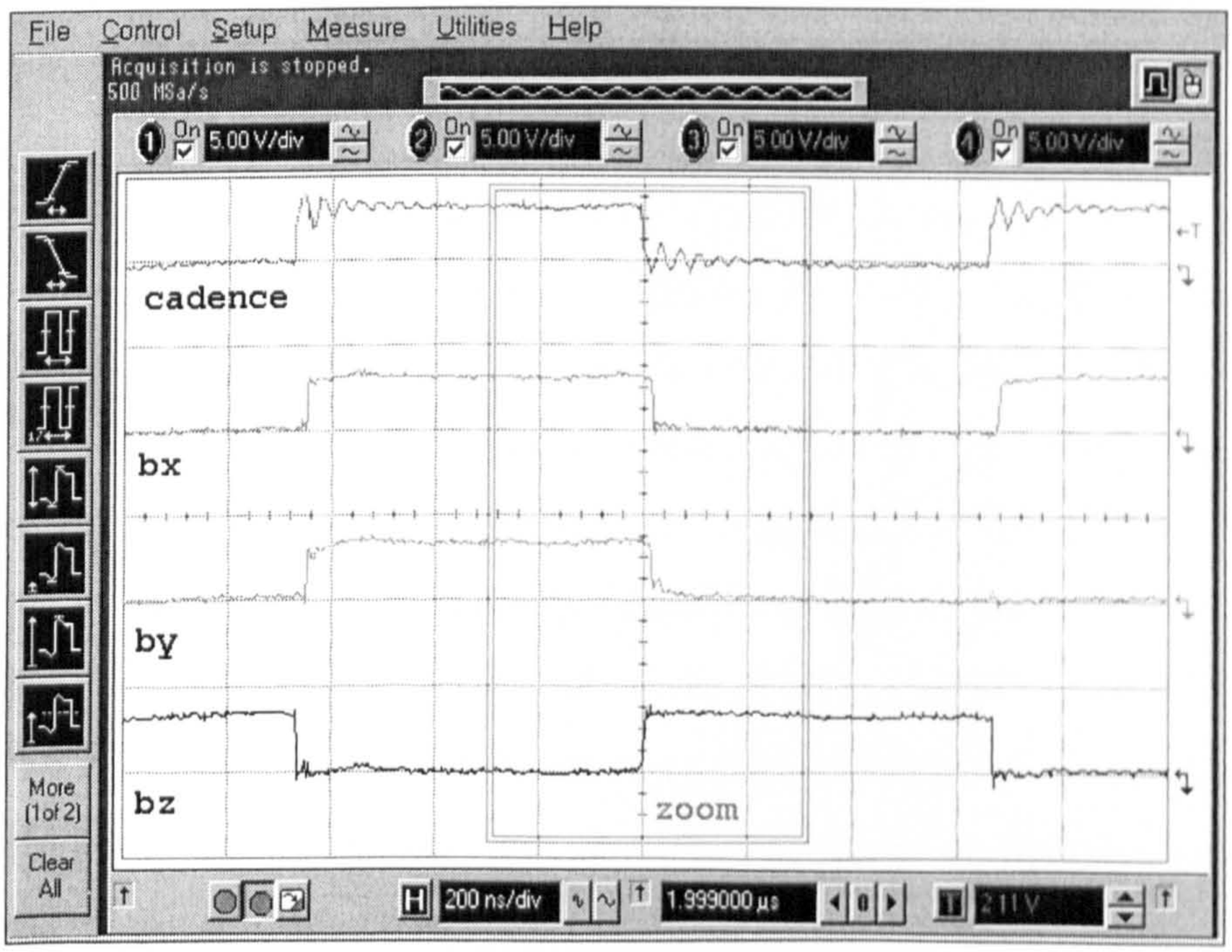

Fig. 5-37 – Test results using the hardware implemented VHDL testbench

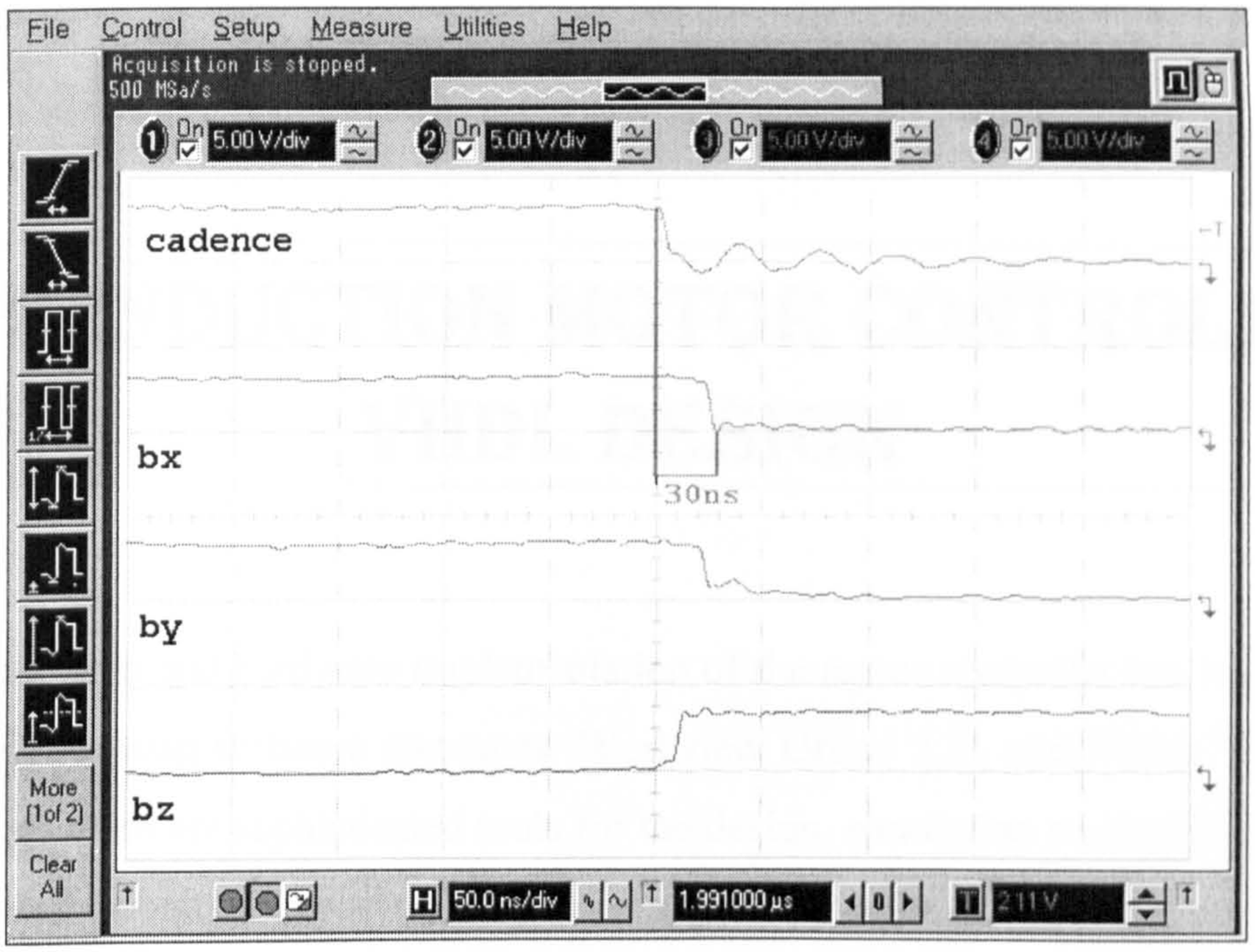

Fig. 5-38 – Propagation delay measurements (detail of Fig. 5-37)

This chapter presented a new approach to ANN hardware implementation, and applied the novel principles to derive a neural network capable of controlling the stator current of an induction motor. The next chapter presents the complete FPGA implementation of the induction motor controller, which includes the neural architecture presented in this chapter.

6. THE INDUCTION MOTOR CONTROLLER VHDL DESIGN

The design and hardware implementation of the motor controller has been carried out using two main software resources: Workview Office 7.31 and Xilinx Foundation 1.4. Both of them are sophisticated tools for the design, simulation and testing of FPGA implemented circuits. Workview Office is a general software package produced by Viewlogic [2], [3] and it can be used in conjunction with FPGAs manufactured by a large variety of producers. It includes a flexible VHDL simulator that supports all the features described by the IEEE 1076-1993 standard definition of the language [9]. On the other hand, Xilinx Foundation is a software package specialised in developing applications using the FPGA families manufactured by Xilinx [5]. It is capable of optimising the hardware implementation according to speed and chip area requirements of particular applications being more versatile than Workview Office from this point of view. However, Xilinx Foundation supports only a subset of the standard VHDL language, namely those statements and functions that can be synthesised and directly implemented into hardware. Furthermore, this software lacks a VHDL simulator. It performs simulations using the netlist files obtained after the synthesis stage, which limits its capabilities and slows down the design and test cycle. Consequently, the VHDL design and simulation have been performed using Workview Office while the implementation and timing verification have been carried out using Xilinx Foundation. The combined use of the two software packages improved the overall efficiency of the simulation, troubleshooting, and synthesis stages in the design cycle.

The VHDL description of the complete motor controller includes the two algorithms described in chapter 4: the current control strategy and the sensorless speed control algorithm. The model has been developed using a hierarchical approach and contains four tiers that consist of several specialised logic blocks (Fig. 6-1). The VHDL code related to these blocks utilises generic parameters to define the size of the registers, adders, subtracters, busses and other elements involved. This allows rescaling of the

controller hardware structure according to the calculation precision imposed by the available types of FPGA circuits. The present version of the motor controller has been implemented into a Xilinx XC4010XL FPGA included on a XS40 test board, which contains a 12 MHz clock circuit.

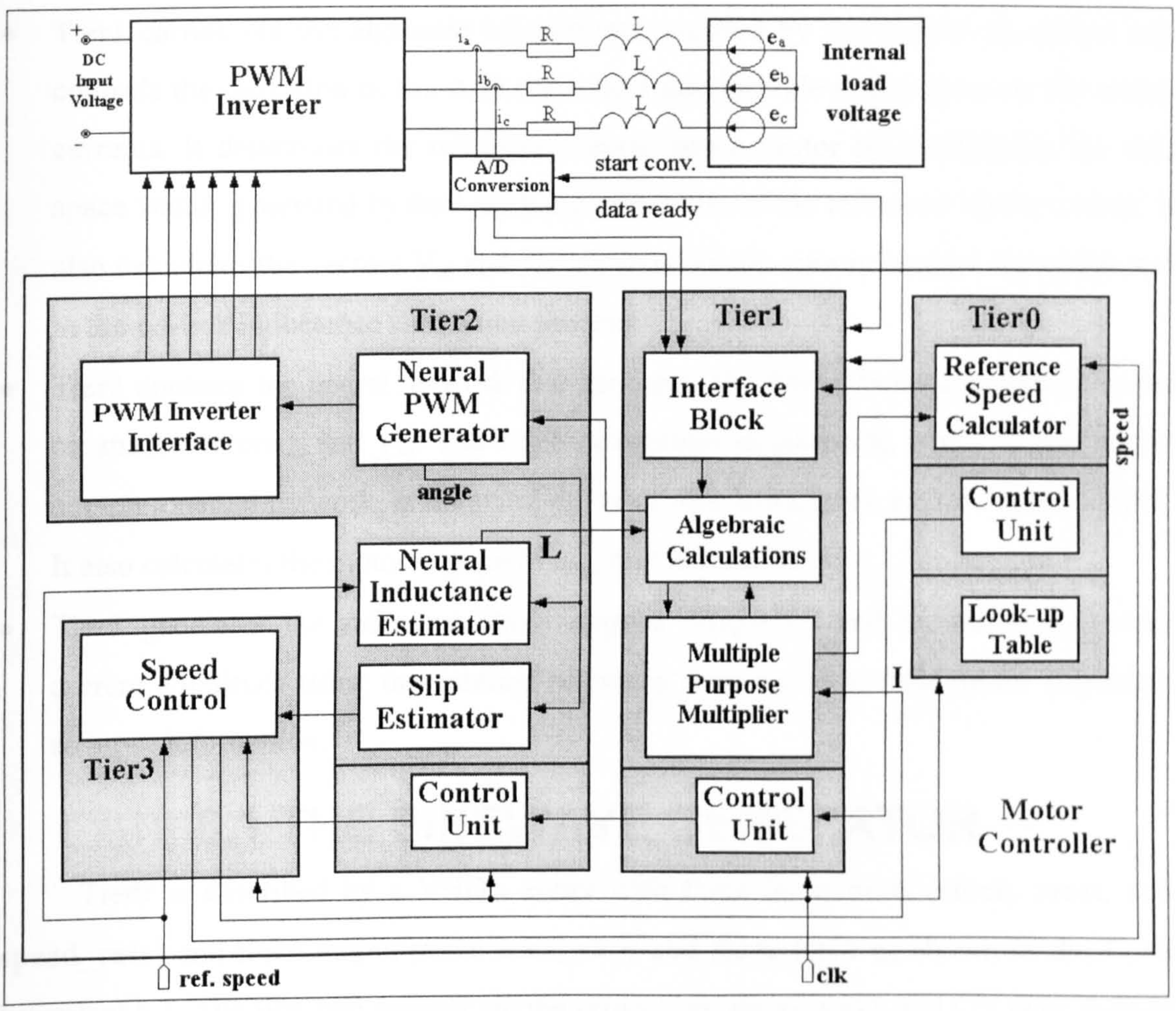

Fig. 6-1 – The most important functional blocks of the FPGA motor controller

The control units included in the structure of tiers 0, 1 and 2 synchronise the operation of all the other logic blocks and control the information transfer between different tiers. They have been designed as clock-synchronised finite state machines (FSM) using the specialised State Editor program included in the Xilinx Foundation package. Using this program, a FSM is graphically described as a state diagram, which can be automatically converted into a VHDL model. The other blocks in each tier have been described directly in VHDL using the register-transfer logic (RTL) manner. Therefore, they consist of registers interleaved with combinational logic structures: adders, subtracters, comparators, etc.

Each of the four tiers performs specific tasks:

- Tier0 generates a space vector of constant amplitude and variable angular speed. The angular speed corresponds to the stator current frequency. The vector is defined by its real and imaginary parts in the stator reference frame. Therefore, tier0 is a sinewave generator. It produces two variable frequency sinewaves shifted with 90°.
- Tier1 carries out the algebraic calculations required by the control algorithm and controls the operation of the A/D converters that provide information on the motor currents. It determines the reference current space vector by multiplying the unit space vector generated by tier1 with the amplitude of the reference stator current. It also calculates the vectors $\underline{V}_{ni}$ and V_{Δ} involved in the current control algorithm and in the on-line inductance estimation process.
- Tier2 contains the neural network that generates the PWM switching pattern based on space vectors $\underline{i}_s$ and $\underline{V}_{ni}$. The angle calculation subnetwork, which is part of the complex neural network, is involved in the on-line inductance estimation algorithm. It also calculates the motor slip angle α_{eqv} that is used by tier3.
- Tier3 generates the reference stator angular frequency and the reference stator current amplitude using the external reference rotor speed and the motor slip angle as input information.

6.1 THE SINEWAVE GENERATOR

Tier0 is described by a VHDL entity with three input ports (**clock, reset,** and **speed_rate**) and three output ports: **cosx, cosy** and **start_tier1** as shown in the Code Fragment 6.1. The first two outputs are the projections on axes OX and OY (Fig. 6-2) of the unit vector rotating around the origin of the two-dimensional plane with the angular speed indicated by the input port **speed_rate.** The output port **start_tier1** informs tier1 when the calculations performed by tier0 have been completed and the data on the output ports **cosx** and **cosy** is ready.

```
-- Code Fragment 6.1
entity tier0 is
    port (
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        speed_rate: in STD_LOGIC_VECTOR(16 downto 0);
        cosx: out STD_LOGIC_VECTOR (8 downto 0);
        cosy: out STD_LOGIC_VECTOR (8 downto 0);
        start_tier1: out STD_LOGIC
    );
end tier0;
```

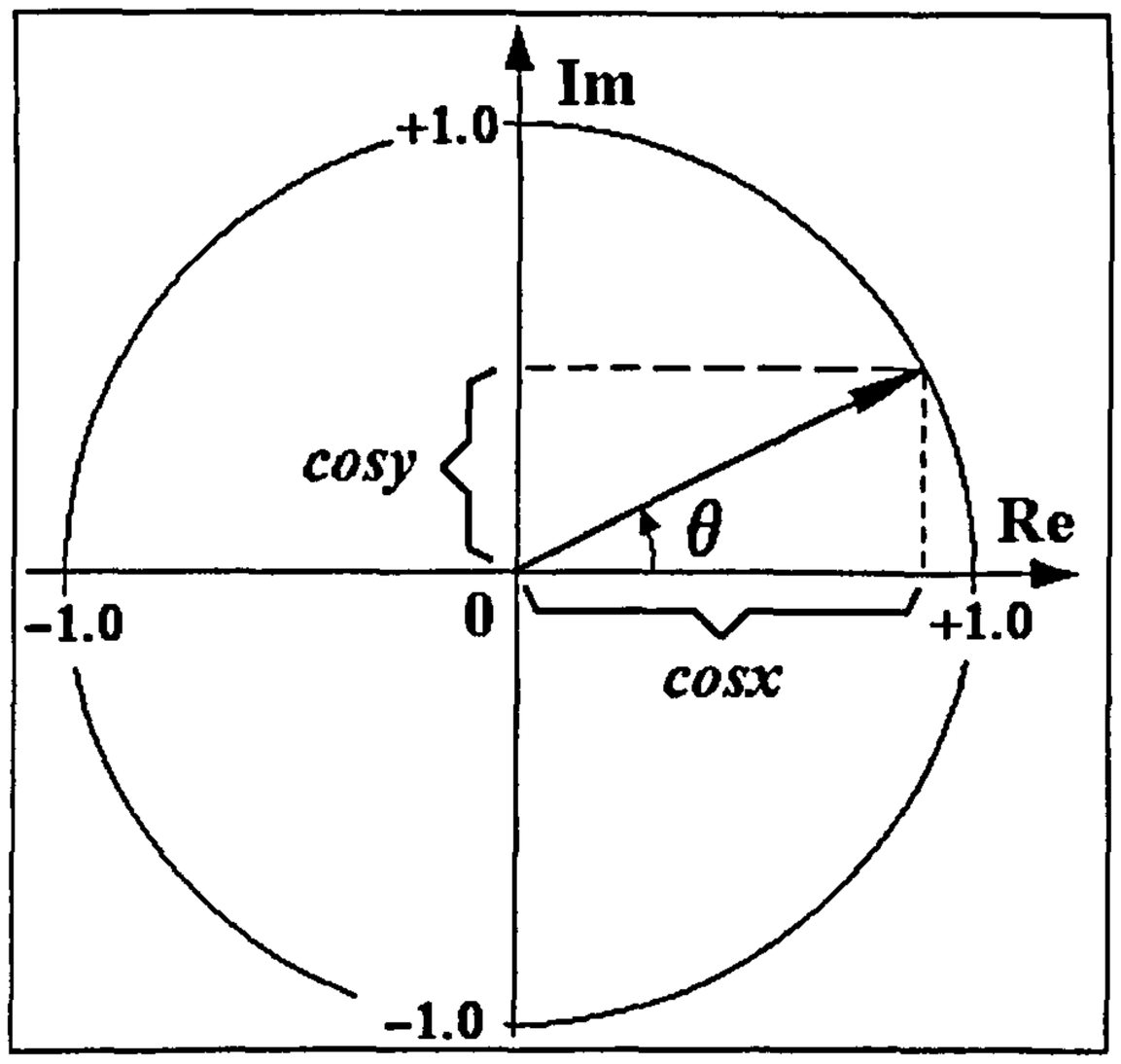

Fig. 6-2 – The real and imaginary components of the unit vector

The architecture of tier0 contains three components: a data processing unit (**data_tier0**), a control unit (**ctrl_tier0**), and a look-up table (**sin_rom**). The look-up table stores information about the waveform of the two sinewaves that need to be generated. Traditionally, the look-up table contains the samples of the sinewave to be generated and the data processing unit reads the table in sequence and at the required speed. To reduce the memory size, only the information referring to a quarter of a sinewave period is stored in the memory. However, such a look-up table is still too large to be efficiently implemented in the same FPGA with the rest of the controller. To minimise the memory implementation size, the differential modulation technique has been used. This was made possible by the fact that the angle θ in Fig. 6-2 has a predictable variation in time due to its relation to the stator current angular frequency: $d\theta/dt=\omega_{es}$. It increases or decreases depending on the sign of ω_{es}, but is not subject to sudden variations. Therefore, the values of **cosx** and **cosy** can be determined by adding or subtracting small increments to the values calculated during the previous calculation cycle. The increments take up fewer bits than the corresponding sinewave samples because they are small quantities. These small increments can be stored in a compact look-up table that requires much less hardware resources than the classical look-up table.

The VHDL code describing the look-up table has been automatically generated by the specialised C++ program Sin_Rom.CPP presented in Appendix C. The program has two parameters defining the amplitude of the sinewave (**ampl**) and the number of entries in the look-up table (**n_steps**). Therefore, several versions of the look-up table can be

generated by altering these two parameters. The optimal version depends on the required precision and on the available hardware resources. Different alternatives have been tested by simulation and physical implementation, and the solution given by **ampl**=127 and **n_steps**=64 has eventually been adopted. These two parameters define a sinewave with 265 samples per period and values between –127 and +127. The difference between two samples varies between 0 and 7, so 3 bits are sufficient for each memory location. In conclusion, the differential modulation technique reduces the size of the look-up table to 33% because the initial 9 bit samples can be replaced by 3-bit sample differences.

The VHDL model automatically generated by the C++ program is an entity with one input port (the address bus) and one output port (the data bus). The associated architecture contains a single process that produces the data corresponding to the address using the constant array of 64 std_logic_vector elements shown in Code Fragment 6.2

```
-- Code Fragment 6.2
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY sin_rom IS
  PORT(
    A: IN std_logic_vector(5 DOWNTO 0);
    DO: OUT std_logic_vector(2 DOWNTO 0));
END sin_rom;

ARCHITECTURE sin_rom_arch OF sin_rom IS
  TYPE mem_data IS ARRAY (0 TO 63) OF std_logic_vector(2 downto 0);
  constant VD: mem_data :=
  (  ('0','0','0'),
  ('0','0','0'),
  ('0','0','1'),
  ('0','0','0'),
-- ..................
  ('1','1','1'),
  ('1','1','0'),
  ('1','1','0'),
  ('1','1','0'),
  ('1','1','1'),
  ('1','1','0'));
BEGIN
  PROCESS(A)
  begin
      DO<=VD(conv_integer(A));
  END PROCESS;
END sin_rom_arch;
```

The data processing unit inside the sinewave generator has a cyclical operation. During each cycle, it reads the look-up table in sequence and adds or subtracts the memory values to the current outputs in order to generate the required sinewaves. The operation cycles are initiated by a clock divider modelled by the VHDL process **start_generator** inside the architecture of **data_tier0**. The corresponding VHDL code is:

```
-- Code Fragment 6.3
start_generator: process(reset,clk)
    variable counter_val: integer range 0 to 511;
  begin
    if reset='1' then
      counter_val:=2;
      start<='0';
    elsif clk='1' and clk'event then
      if counter_val=0 then
        counter_val:=UpperCount_tierf;
        start<='1';
      else
        counter_val:=counter_val-1;
        start<='0';
      end if;
    end if;
  end process;
end data_tierf_arch;
```

The clock division ratio specified by the constant **UpperCount_tierf** has been set to 79. Given the 12 MHz frequency of the clock signal, a number of 150,000 operation cycles are initiated every second. A complete sinewave period contains 256 samples, so each sample is repeatedly generated for a number of times that depends on the required sinewave frequency. To achieve this, a new value read from the ROM memory is added to the previous result only when the memory address changes. Thus, the speed of changing the memory address is proportional to the required frequency. The multiplication with the corresponding proportionality constant is performed by tier1, which transmits the result to tier0 on the input port **speed_rate.**

The value of **speed_rate** is added to the signal **adr_cosy** and the result is stored in the register **next_adr_cosy.** Based on the information stored in **adr_cosy** and **next_adr_cosy,** the correct memory address is generated, after which the value of **cosy** is updated. At the end of each operation cycle, **next_adr_cosy** is copied to **adr_cosy** so that a new value for **next_adr_cosy** can be calculated at the beginning of the next cycle. The addition is performed using the '+' operator defined in **std_logic_signed** package from IEEE library. The operators in this VHDL package have the advantage that the sign bit of the shorter operand is always extended on the empty positions as shown in

Fig. 6-3. This simplifies the design process for complementary code adders and subtracters as they can be directly modelled by the corresponding algebraic equations.

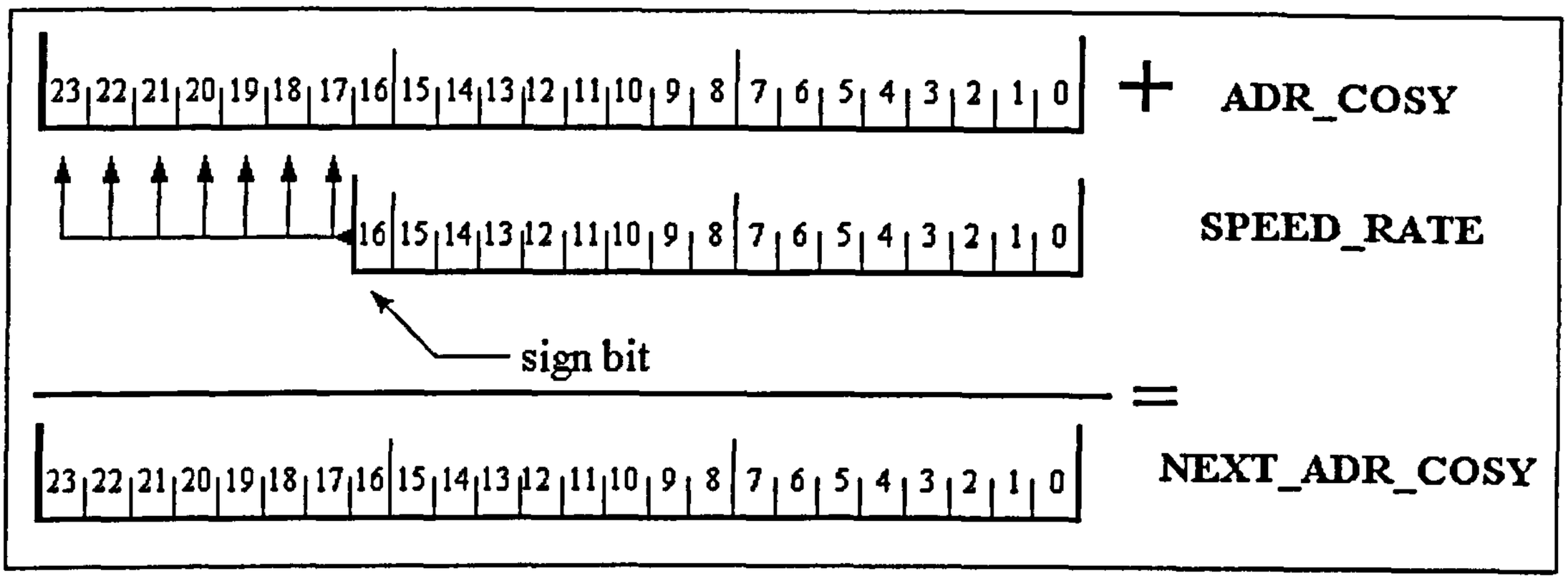

Fig. 6-3 – NEXT_ADR_COSY calculation

Two signals generated by the control unit, **add_speed_rate** and **inc_adr**, indicate the moments when the two address values **adr_cosy** and **next_adr_cosy** are updated. They correspond to independent VHDL processes because the updating operations are carried out at separate moments in time:

```
-- Code Fragment 6.4
process(reset,add_speed_rate)
  begin
    if reset='1' then
      next_adr_cosy<=(others=>'0');
      speed_sign<='0';
    elsif add_speed_rate='1' and add_speed_rate'event then
      next_adr_cosy<=adr_cosy+speed_rate;
      speed_sign<=speed_rate(16);
    end if;
  end process;

process(inc_adr,reset) --It is the last process to be performed
  begin
    if reset='1' then
      adr_cosy<=(others=>'0');
    elsif inc_adr='1' and inc_adr'event then
      adr_cosy<=next_adr_cosy;
    end if;
  end process;
```

The signals **adr_cosy** and **next_adr_cosy** are 24-bit vectors whose most significant 8 bits correspond to the sample index relative to the beginning of the sinewave period (a number between 0 and 255). During each operation cycle, **next_adr_cosy** is compared to **adr_cosy**. If the most significant 8 bits in the two vectors are different, it means that the current sample index has changed after **next_adr_cosy** has been modified by adding **speed_rate**. Consequently, a new value is to be read from the memory and it has to be either added to or subtracted from **cosy**,

depending on the slope of the sinewave around the current sample. Otherwise no operation is performed.

The look-up table has only 64 entries corresponding to a quarter of the complete sinewave period. As a result, the address required by the look-up table is made up of only 6 bits and varies between 0 and 63. The address needs to be calculated according to a certain algorithm, which locates the correct look-up table entry depending on the current sample index (between 0 and 255) and the sinewave frequency sign. The details of the address calculation algorithm are first presented for positive speeds and then it is extended to both positive and negative speeds. As the sample index increases from 0 to 255, the memory address varies according to Fig. 6-4. Thus, when the sample index is inside the interval [0; 63] (the first sinewave quarter), the bits 16 to 21 of **adr_cosy** are used as memory address:

$$\text{mem\_adr} = \text{adr\_cosy}(21 \text{ downto } 16) \tag{6-1}$$

For sample indices in the interval [64; 127] (the second sinewave quarter), the memory values need to be extracted in the reversed order. The address is in this case calculated using the formula:

$$\text{mem\_adr} = 127 - \text{adr\_cosy}(22 \text{ downto } 16) \tag{6-2}$$

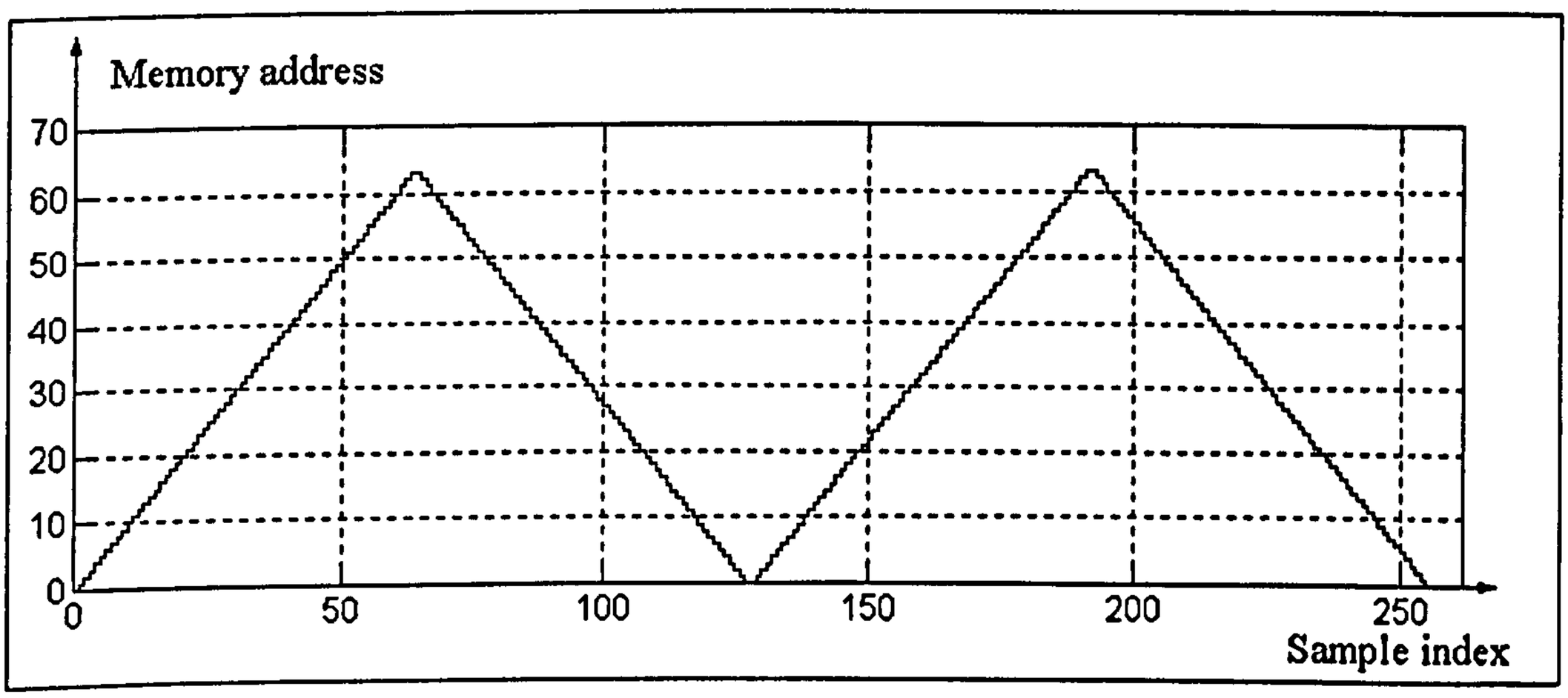

Fig. 6-4 – The correspondence between sample index inside the complete sinewave and the memory address

The same addressing sequence is used for the second sinewave half because the two halves differ only by the most significant bit of **adr_cosy**, which is '0' during the first half and '1' during the second half. The bits 16 to 22 of **adr_cosy** undergo the same sequence of changes during the two sinewave halves (Fig. 6-5) and therefore the same

calculations can be used to generate the entire waveform. Thus, formula (6-1) is applied for the third sinewave quarter, while formula (6-2) is used for the fourth quarter.

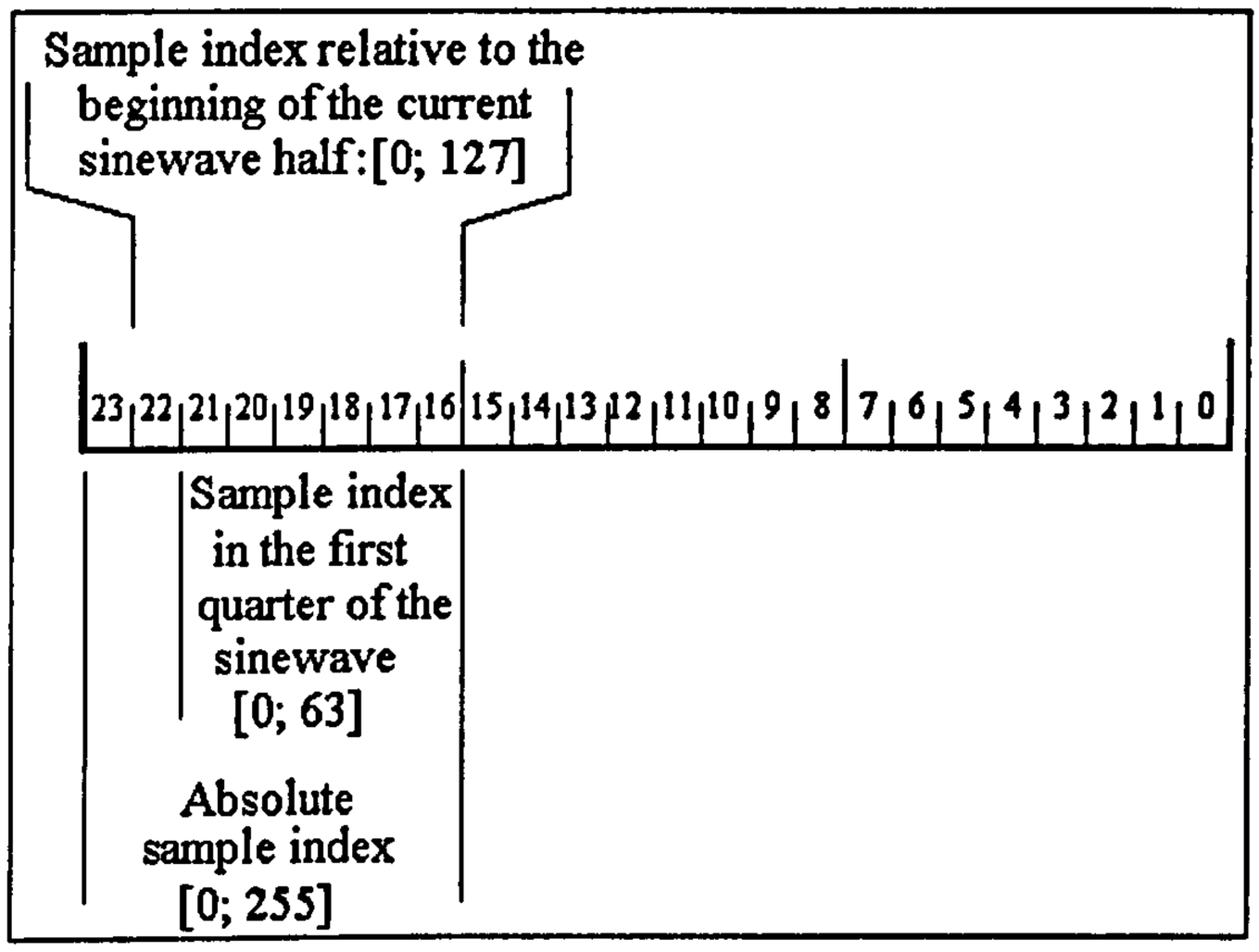

Fig. 6-5 – The bits of signal adr_cosy

The most significant bit of **adr_cosy** (the bit 23) is used to decide whether the new memory value has to be added to or subtracted from the current **cosy** value. These values are added during the first half and subtracted during the second as shown in Fig. 6-6.

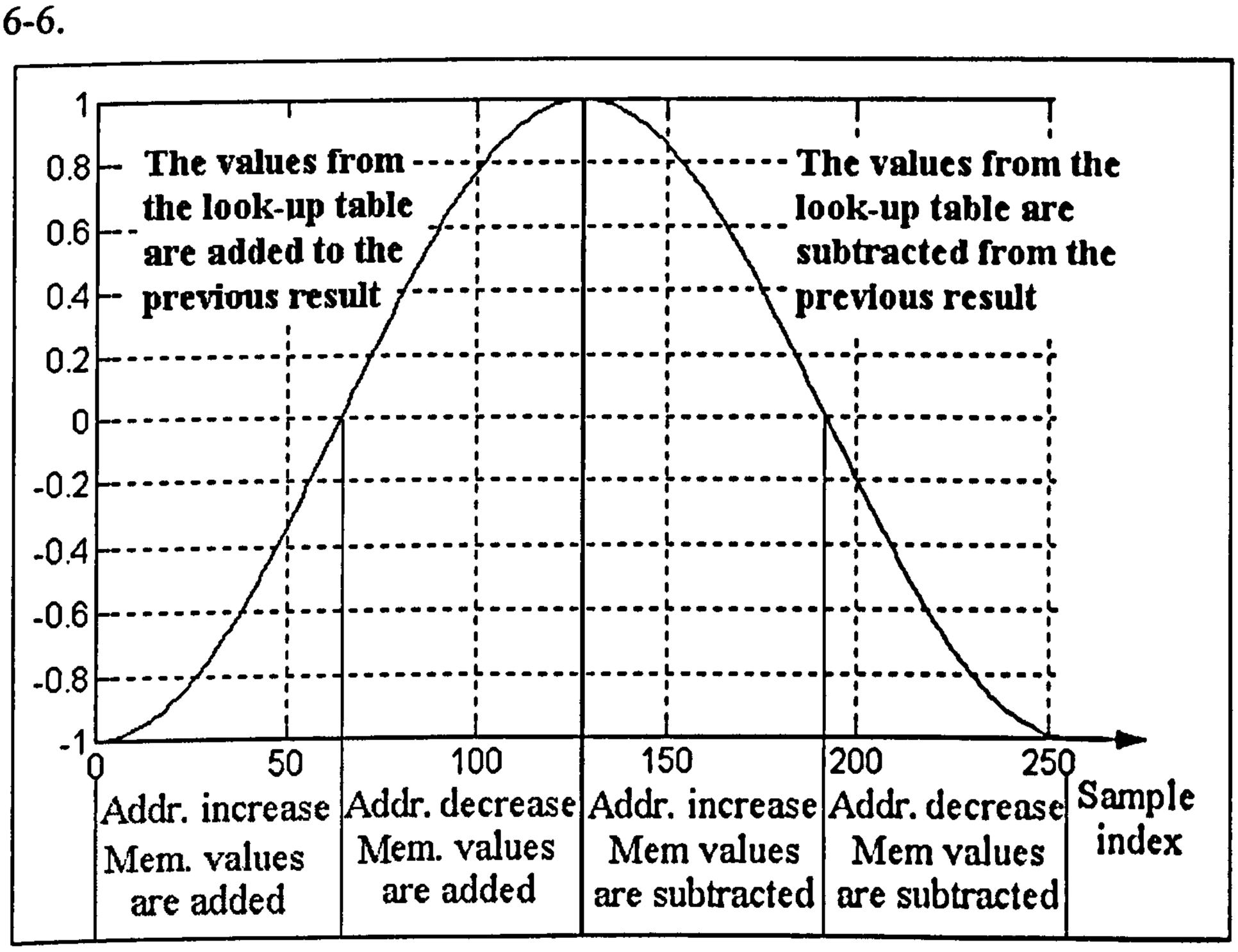

Fig. 6-6 – Sinewave generation algorithm (positive speeds)

The algorithm can be now extended for both positive and negative speeds. To do so, it must be noted that each location in the look-up table stores the difference between

the next sinewave sample and the current sinewave sample for positive sinewave slope and positive speed:

$$TABLE[mem\_adr] = |NextSample - CurrentSample| \quad (6\text{-}3)$$

At negative speeds the sequence of samples is reversed so that the previous sample is calculated instead of the next sample:

$$TABLE[mem\_adr - 1] = |CurrentSample - PreviousSample| \quad (6\text{-}4)$$

The vector **next_adr_cosy** is larger then **adr_cosy** at positive speed because **speed_rate** is a positive value. When the speed is negative, **speed_rate** is negative as well, and **next_adr_cosy** becomes smaller then **adr_cosy**. Thus, **adr_cosy** is used to generate **mem_adr** when the speed is positive, while **next_adr_cosy** is used to calculate **mem_adr-1** when the speed is negative.

The memory address used to update **cosx** is derived from signals **adr_cosx** and **next_adr_cosx** which are 8-bit long vectors. Their values are related to the values of **adr_cosy** and **next_adr_cosy** because the two sinewaves are 90^0 shifted, which is translated into a sample index difference of 64. Consequently, **adr_cosx** is obtained adding 64 to the most significant 8 bits of **adr_cosy**, which can be reduced to adding "01" to the bits 22 and 23 of **adr_cosy** (Fig. 6-7). The vector **next_adr_cosx** is used only to generate the memory addresses at negative speeds because the transitions between two sinewave samples is already determined by the difference between **adr_cosy** and **next_adr_cosy**. Consequently, **next_adr_cosx** is calculated according to the simple equation

$$next\_adr\_cosx = adr\_cosx - 1 \quad (6\text{-}5)$$

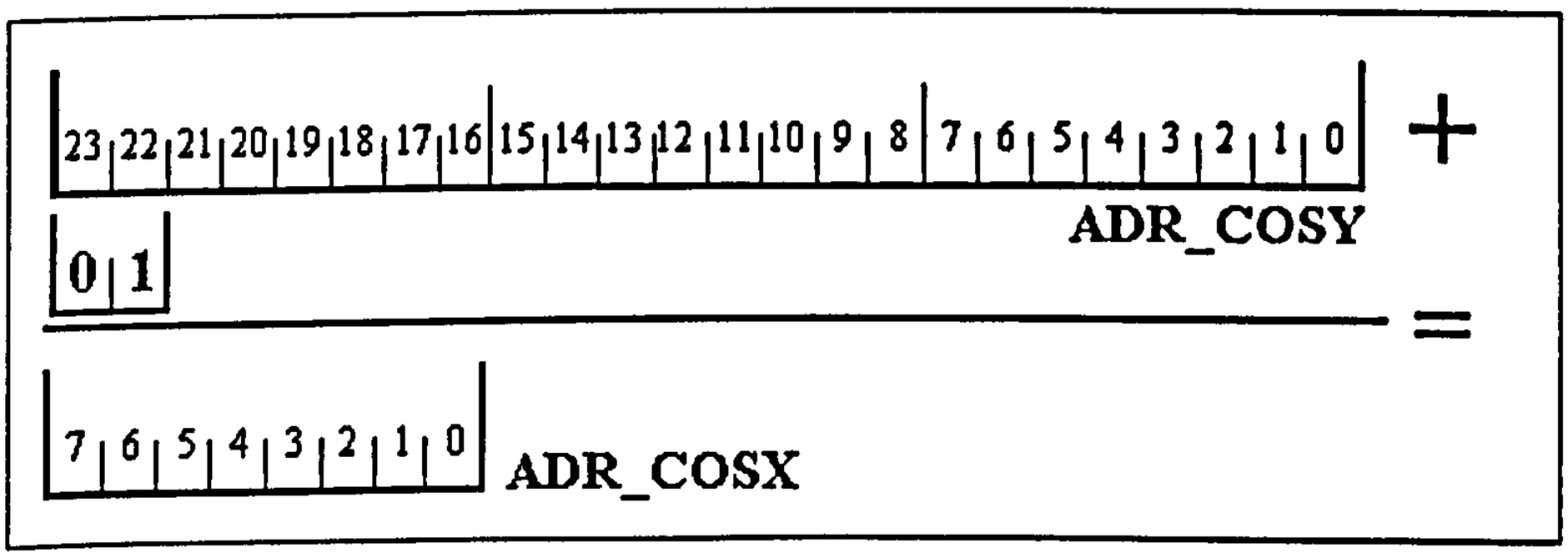

Fig. 6-7 - ADR_COSX calculation manner

The sinewave generator calculates the address in two stages implemented as two VHDL processes. First, an internal memory address (**int_mem_adr**) is calculated based

on the 7 bits which give the relative sample index to the beginning of the current sinewave half (Fig. 6-5). The internal memory address is compared to 63 during the second stage, and in case it surpasses this limit then equation (6-2) is used to calculate the equivalent address, which is confined between 0 and 63. When this upper limit is not surpassed, no calculation is performed. The final result **mem_adr** consists of the least significant 6 bits of the vector **x** generated at stage two.

Due to the large number of operation cycles performed every second compared to the number of sinewave samples, there are numerous cycles when the memory is not addressed because tier0 outputs do not need to be updated. During the cycles when the memory needs to be addressed, the operation is carried out twice: first time to update **cosy** and second time to update **cosx**. The signal controlling which of the two memory addresses is to be calculated at a certain moment **(adr_mux)** is generated by the control unit. This signal is '0' when the address corresponding to **cosy** is calculated, and it is '1' otherwise. Therefore, the calculation of **int_mem_adr** in the first VHDL process depends both on **adr_mux** and on the speed sign stored by the signal **speed_sign** as shown in the following code fragment.

```
-- Code Fragment 6.5
process(adr_cosy,adr_cosx,next_adr_cosy,next_adr_cosx,
        adr_mux,speed_sign)
  begin
    if speed_sign='0' and adr_mux='0' then
      int_mem_adr<=adr_cosy(22 downto 16);
    elsif speed_sign='0' and adr_mux='1' then
      int_mem_adr<=adr_cosx(6 downto 0);
    elsif speed_sign='1' and adr_mux='0' then
      int_mem_adr<=next_adr_cosy(22 downto 16);
    else
      int_mem_adr<=next_adr_cosx(6 downto 0);
    end if;
  end process;

  process(int_mem_adr)
    variable x: std_logic_vector(6 downto 0);
  begin
    x:=int_mem_adr;
    if x(6)='1' then
      x:="0111111"-('0' & x(5 downto 0));
    end if;
    mem_adr<=x(5 downto 0);
  end process;

  adr_cosx<=(adr_cosy(23 downto 22)+"01") & adr_cosy(21 downto 16);
  next_adr_cosx<=adr_cosx-"01";
```

Signals **cosy** and **cosx** are updated inside two VHDL processes, which are activated by the signals **add_cosy** and **add_cosx** generated by the control unit. These

processes decide whether to add or subtract the value read from the look-up table based on the speed sign and the most significant bit of **adr_cosy** and **adr_cosx**, respectively. This most significant bit indicates if the current sample is situated in the first or in the second half of the sinewave period. This information is correlated with the sign of the sinewave slope. If the slope is positive the new value has to be added to the output signal, otherwise it has to be subtracted. The **reset** signal is part of the sensitivity list of the two processes so it initialises the outputs at the beginning of the circuit operation. The two outputs **cosx** and **cosy** are also periodically reinitialised to the correct values whenever **adr_cosy** is zero (Code Fragment 6.6). This mechanism improves the system robustness by avoiding the accumulation of errors caused by possible electromagnetic interference generated by the power transistors in the PWM inverter.

```
-- Code Fragment 6.6
process(adr_cosy)
  --This reset ensures that errors are periodically eliminated
  begin
    if adr_cosy(23 downto 16)="00000000" then
      periodical_reset<='1';
    else
      periodical_reset<='0';
    end if;
  end process;

  process(add_cosy,reset,adr_cosy,periodical_reset)
  begin
    if (reset='1') or (periodical_reset='1') then
      int_cosy<=(8=>'1',0=>'1',others=>'0');
    elsif add_cosy='1' and add_cosy'event then
      if (adr_cosy(23) xor speed_sign)='0' then
        int_cosy<=int_cosy+('0' & data);
      else
        int_cosy<=int_cosy-('0' & data);
      end if;
    end if;
  end process;

--The value of 'cosx' is reset whenever the memory address is 0 and
--mux_adr=0. When mux_adr=1 it means cosx will be increased.
--Therefore it mustn't be reset any longer.

  process(add_cosx,reset,adr_cosy,adr_mux,periodical_reset)
  begin
    if reset='1' or (periodical_reset='1' and adr_mux='0') then
      int_cosx<=(others=>'0');
    elsif add_cosx='1' and add_cosx'event then
      if (adr_cosx(7) xor speed_sign)='0' then
        int_cosx<=int_cosx+('0' & data);
      else
        int_cosx<=int_cosx-('0' & data);
      end if;
    end if;
  end process;
  cosy<=int_cosy;
```

```
cosx<=int_cosx;
```

As previously mentioned, the control unit has been designed as a finite state machine using the State Editor included in Xilinx Foundation software package. Thus, the operation of ctrl_tier0 was initially described by a state diagram, which has subsequently been converted into a VHDL model. The elements of a typical diagram are states, transitions, transition conditions, actions, the reset signal, the clock signal, input ports and output ports.

- The transitions between states are triggered by the clock signal. The state machine can be defined as either active on the rising clock edge or active on the falling clock edge.
- A condition assigned to a transition inhibits the state change until the condition is fulfilled. All conditions need to comply with the VHDL syntax because they are included as they were written in the automatically generated VHDL model of the FSM.
- The actions performed by the state machine are changes of the output ports. There are three different types of actions: entry actions, state actions, and exit actions. The changes occur at different moments in time: at the transition from the previous state to the current state (entry action), during the current state (state action) or at the transition between current state and next state (exit actions).
- The reset signal is a special input port that brings the state machine in its original state. It can be defined as synchronous or asynchronous. The synchronous reset brings the FSM in the initial state only when the correct clock edge occurs, while the effect of an asynchronous reset signal is instantaneous.
- There are two types of output ports: registered and combinational. The registered outputs are modelled as registers and therefore the effect of any action is valid until the next action modifies the port. The combinational outputs have no memory. The effect of any action lasts as long as the FSM is in the state linked to the respective action, after which the output returns value specified for the original state (the state forced by the **reset** signal).

The model of ctrl_tier0 has been defined as a state machine with six states that is active on the falling clock edge and uses an asynchronous reset signal. The control unit has five output ports: **adr_mux, add_cosx, add_cosy, add_speed_rate** and **inc_adr.** The port **adr_mux** is registered while the others are combinational (Fig. 6-8). Each

operation cycle of ctrl_tier0 begins when the **start** signal is activated by the process in Code Fragment 6.3 contained in the data processing unit. The first action carried out during the control unit operation cycle is to trigger the calculation of **next_adr_cosy** by adding **speed_rate** to its previous value. The vectors **adr_cosy** and **next_adr_cosy** are compared by the data processing unit and the signal **equal** is set accordingly. This requires the inclusion of a comparator in the structure of the data processing unit. The corresponding VHDL model is described by Code Fragment 6.7.

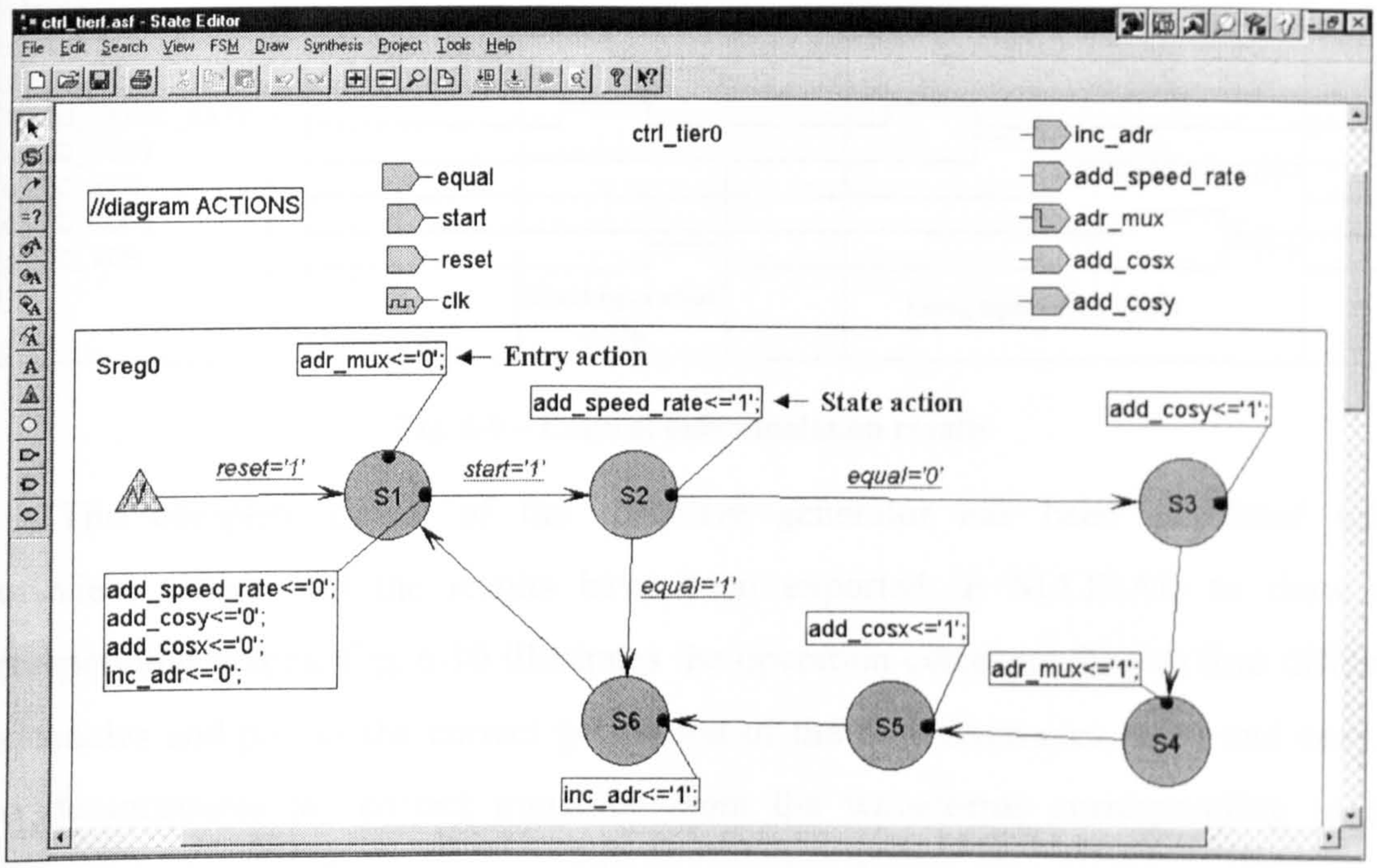

Fig. 6-8– The state diagram of ctrl_tier0

```
-- Code Fragment 6.7
process(adr_cosy,next_adr_cosy)
  begin
    if adr_cosy(22 downto 16) /= next_adr_cosy(22 downto 16) then
      equal<='0';
    else
      equal<='1';
    end if;
  end process;
```

If the most significant 8 bits of **next_adr_cosy** and **adr_cosy** are different, then the values of **cosx** and **cosy** need to be updated. During states S1, S2 and S3 the variable **adr_mux** is set to '0' so that **cosy** can be updated when **add_cosy** is activated. During states S4, S5 and S6 **adr_mux** is set to '1' to calculate the memory address corresponding to **cosx**. The output vector **cosx** is updated during the state S5. During state S6, the signal **inc_adr** is activated and, as shown by Code Fragment 6.4, the vector **adr_cosy** is updated. If the vectors **adr_cosy** and **next_adr_cosy** are equal then the operation cycle comprises only the final action linked to the state S6. Consequently,

there are short operation cycles and long operation cycles depending on the value of the signal **equal** generated by the data processing unit. These two cycle types are illustrated by the simulation results in Fig. 6-9.

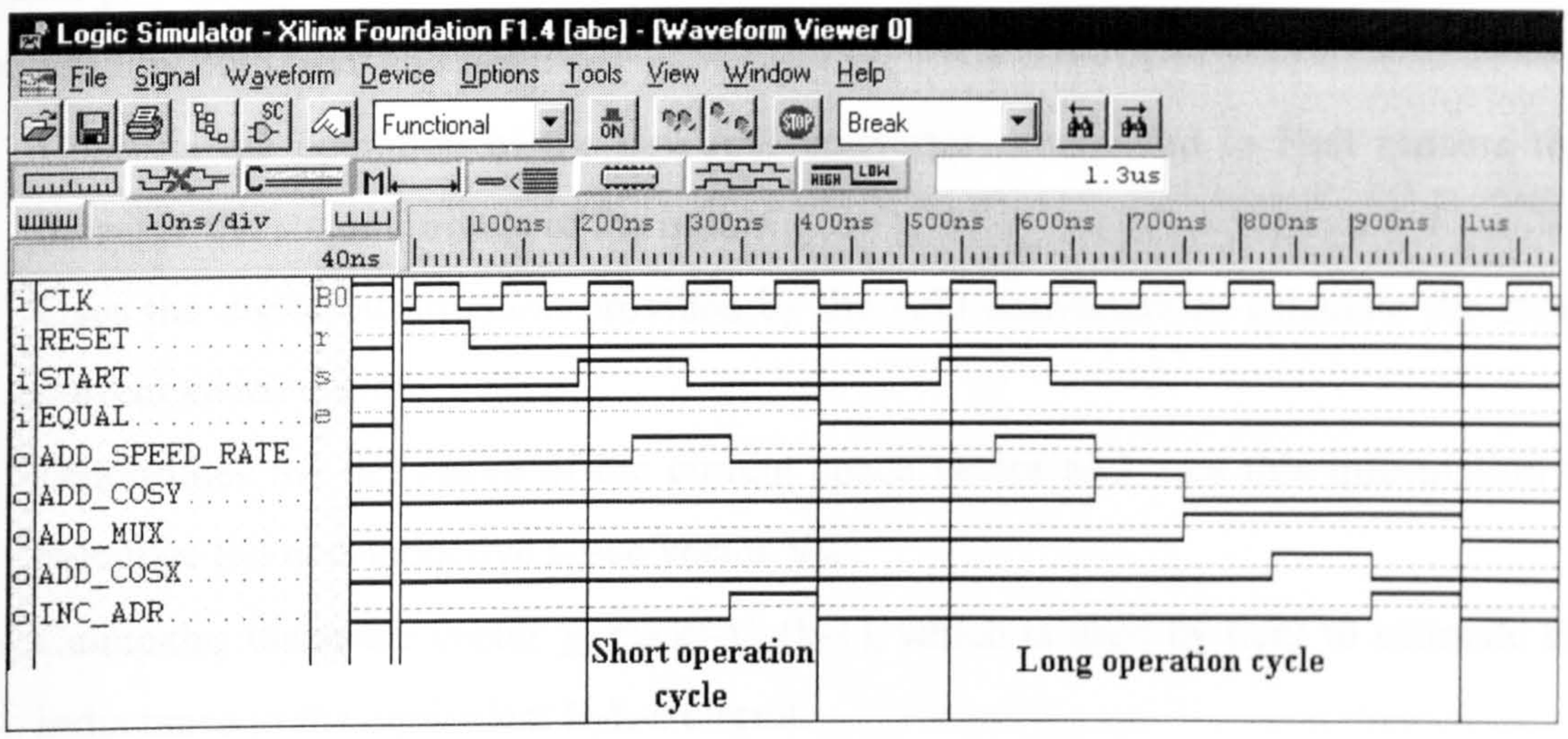

Fig. 6-9 – Control unit simulation results

The complete model of the sinewave generator has been simulated using Workview Office and the results have been exported in MATLAB to draw the corresponding graphs. Fig. 6-10 illustrates the operation corresponding to four different frequencies and proves the correct generation of the two sinewaves, **cosx** and **cosy**. It also demonstrates the correct transition from the waveforms corresponding to one frequency to the waveforms corresponding to another frequency.

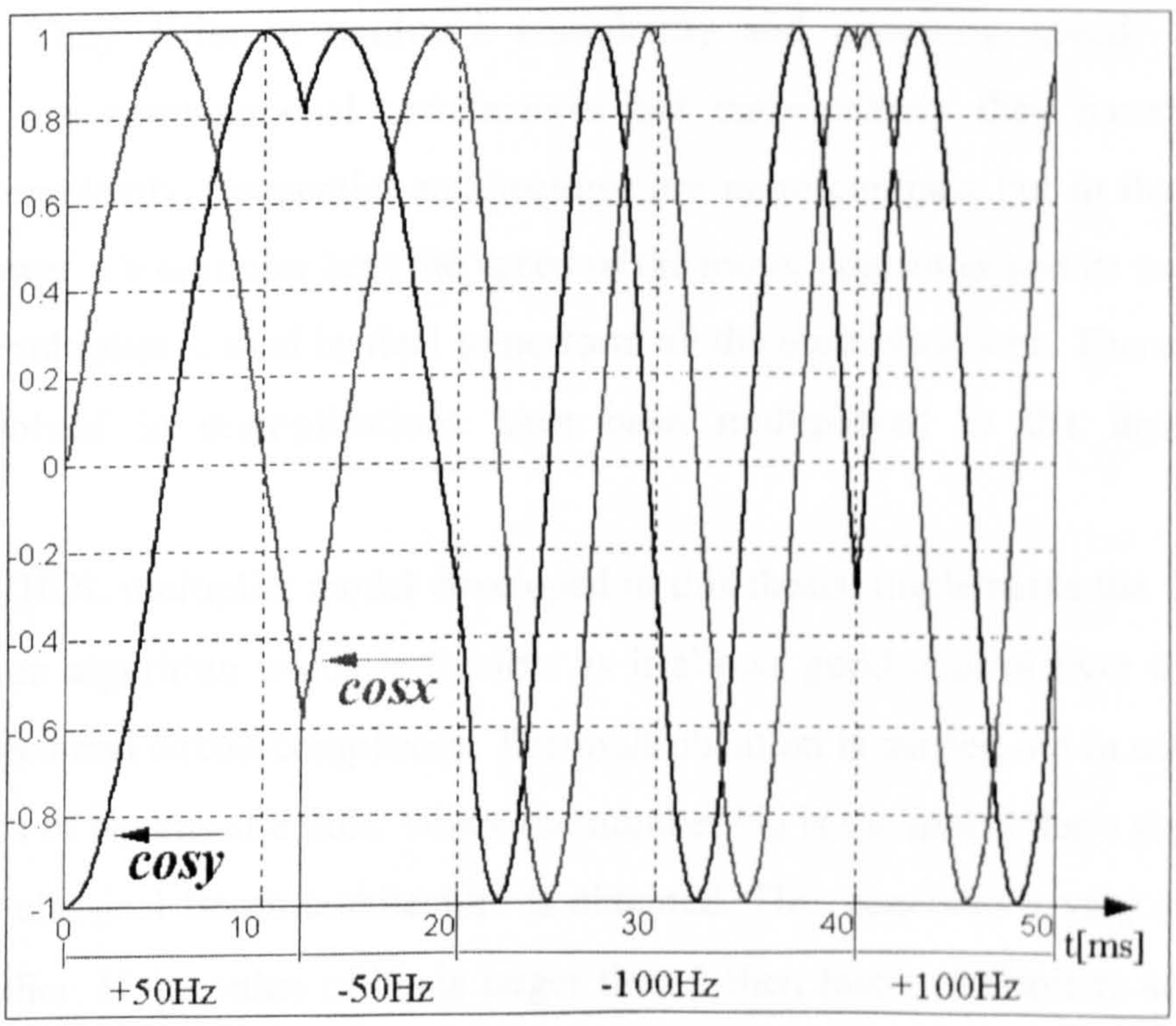

Fig. 6-10 – The operation of tier0

6.2 THE STRUCTURE OF TIER1

Tier1 is a complex processing module composed of a control unit (ctrl_tier1) and a data processing unit (data_tier1) that performs several calculations and control tasks required by both tier0 and tier2:

- Controls the operation of the two A/D converters connected to Hall sensors that measure the stator currents of the motor.
- Uses the digital information provided by the A/D converters to calculate the stator current space vector.
- Determines the derivative of the current space vector and uses this information to calculate the non-inductive space vector $\underline{V}_{ni}$
- Calculates the space vector $\underline{V}_{\Delta}=\underline{u}(k)-\underline{V}_{ni}(k-1)$, which is used by tier2 to estimate the inductance in the equivalent R-L-e circuit.
- Calculates the vector **speed_rate** used by tier0.
- Calculates the rectangular co-ordinates of the reference stator current by multiplying the unit vector produced by tier0 with the amplitude calculated by tier3.
- Performs adjustments of the numerical values supplied to the neural network located in tier2, so that the network operation speed is maximised.

To achieve all these tasks, tier1 needs to perform several multiplications. There are numerous multiplier hardware architectures reported in the literature [137], [138], [48], [86]. They differ in hardware complexity and operating speed. The fastest multipliers use combinational architectures but unfortunately they have very large hardware complexity. Sequential architectures are more compact but in the same time they are slower. To optimise both the speed of the motor controller and its complexity, a single fast multiplier is used by tier1 to perform all the multiplications. Therefore, all the signals involved in multiplications have been multiplexed to the inputs of this multiplier.

The VHDL multiplier model developed in this thesis, implements the $2^{N_{sl}}$ – radix multiplication algorithm which is flexible as it allows good control over the trade-off between speed and circuit complexity. The multiplication is carried out in several stages using groups of N_{sl} bits at a time, where the number N_{sl} is the multiplier's step length. If N_{sl} is 1 the classical Booth architecture is obtained. This generates a very compact but slow multiplier. If the value of N_{sl} is larger then 1 then faster multipliers are obtained, but they occupy more space on the chip. The fastest architecture is obtained when N_{sl}

equals the length of the second operand. In this case, the corresponding hardware multiplier has a purely combinational structure, which makes it, space inefficient but very fast.

The VHDL multiplier model uses three generic parameters that define the length of the two operands and the step length, as shown in Code Fragment 6.8. These parameters allow the adaptation of the multiplier to any application requirements referring to speed, operand size and hardware complexity.

```
-- Code Fragment 6.8
entity smultiplier is
    generic(n,m,step_length: integer);
    port (
        a: in STD_LOGIC_VECTOR (n-1 downto 0);
-- Can be only positive
        b: in STD_LOGIC_VECTOR (m-1 downto 0);
-- Can be both positive and negative
        prod: out STD_LOGIC_VECTOR (n+m-1 downto 0);
        clk,start: in STD_LOGIC;
        ready: out STD_LOGIC
    );
end smultiplier;
```

The fact that all the multiplications required by the motor control algorithm involve a signed operand and an unsigned operand has been exploited to optimise the structure of the controller. Thus, a specialised multiplier has been created, which has an unsigned input bus (**b**) and a signed one (**a**). The multiplication process is composed of a series of simple operation cycles. Each cycle consists in multiplying the operand **a** with the least significant N_{sl} bits of **b** and adding the result into a shift register. Both the result and the operand **b** are then shifted with N_{sl} positions to the right. The architecture comprises two shift registers, a control unit and a reduced size combinational multiplier with input buses of width equal to the size of **a** and N_{sl}, as illustrated in Fig. 6-11.

The multiplication process is triggered by the **start** input signal. When this signal is active (is '1') both the control unit and the multiplication result register are reset. When **start** returns to '0', the control unit initiates the multiplier operation by activating the signal **first_step** which causes the operand **b** to be loaded into the corresponding shift register. After **b** has been loaded, the series of calculation cycles commences. During each cycle, the **load_step** signal is activated first and then **shift_step** is activated. After the shift, the most significant N_{sl} bits of the two registers in Fig. 6-11 are padded with zeroes.

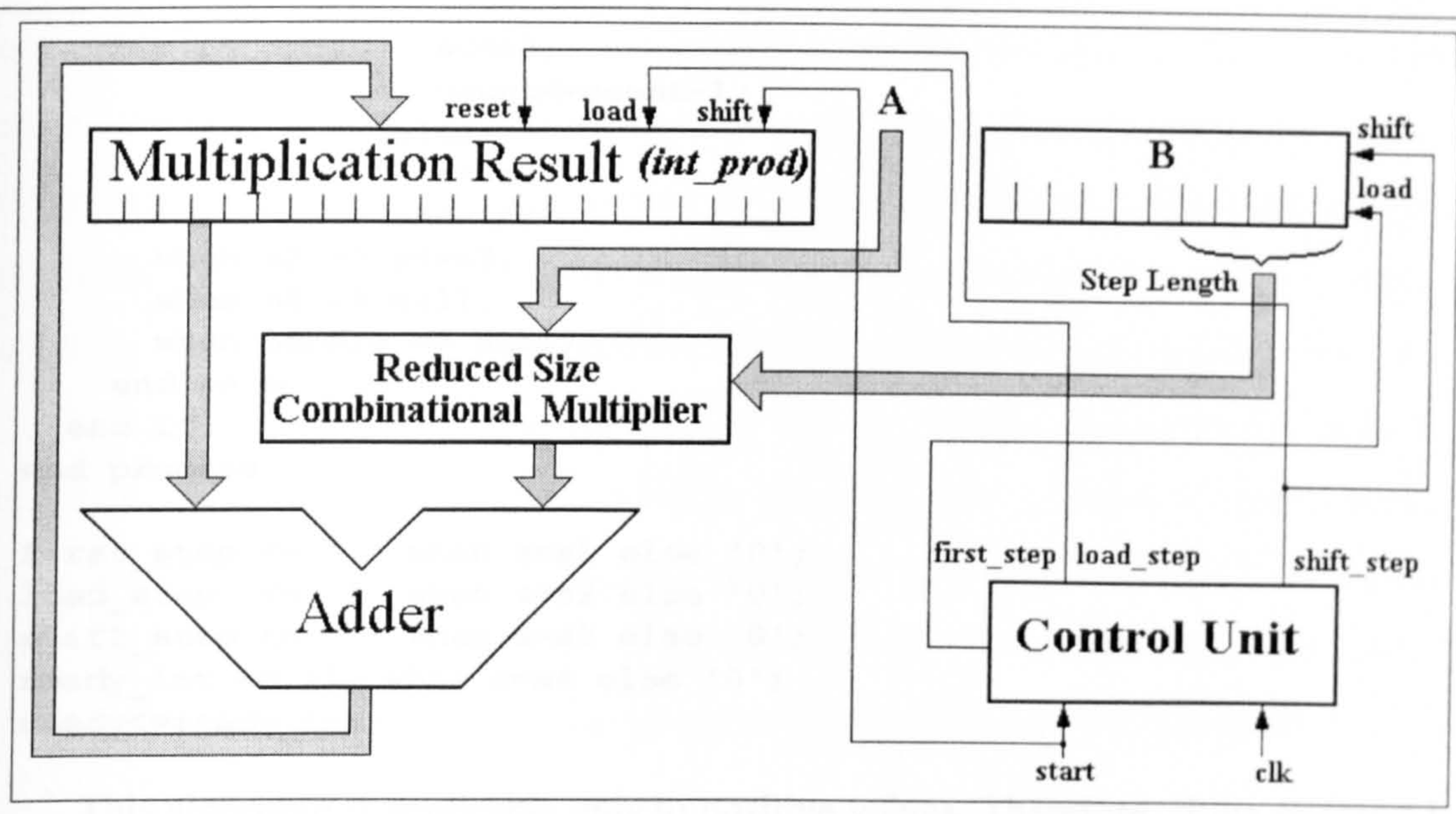

Fig. 6-11 - Flexible multiplier structure

The number of necessary calculation cycles is given by the VHDL constant **nsteps** that depends on the length of **b** (parameter **m**) and on the step length N_{sl}, according to equation

$$\text{nsteps} = \text{ceiling}\left(m / N_{sl}\right) \tag{6-6}$$

where the function 'ceiling' generates the smallest integer that is larger than its argument.

Signal **count** is loaded with value **nsteps** when the input signal **start** is activated, and it is decreased at each calculation cycle simultaneously with adding the partial multiplication result to the result register. If **count** is larger than 1 then the two registers are shifted and a new cycle is initiated. Otherwise the calculations are stopped and the **ready** signal is activated. The VHDL model of the control unit, shown in Code Fragment 6.9, operates with five different states (s0, s1, s2, s3, s4), each of them activating one of the control signals previously discussed.

```
-- Code Fragment 6.9
type state is (s0,s1,s2,s3,s4);
  signal s: state;
constant nsteps: integer := (m+step_length-1)/step_length;
process(clk,start)
  begin
    if start='1' then
      s<=s0;
      count<=nsteps;
    elsif clk='1' and clk'event then
      case s is
        when s0 => s<=s1;
        when s1 => s<=s2;
        when s2 => if count>1 then
```

```
                    s<=s3;
                    count<=count-1;
                  else
                    s<=s4;
                  end if;
        when s3 => s<=s2;
        when s4 => null;
        when others => null;
      end case;
    end if;
end process;

first_step <= '1' when s=s1 else '0';
load_step  <= '1' when s=s2 else '0';
shift_step <= '1' when s=s3 else '0';
ready_int <= '1' when s=s4 else '0';
ready<=ready_int;
```

This algorithm is applicable only to positive values. Therefore, if the operand **b** is positive then the multiplication result is the value stored in the result register after the calculation sequence has finished. Otherwise, this result has to be transformed into a valid two's complement codification of the negative multiplication result. This transformation is based on the next considerations:

- If **b** is a negative number then '2^m-**b**' which is its two's complement is a positive number.
- If **b** is replaced by '2^m-**b**' in the multiplication process, the result is '$(2^m\text{-}\mathbf{b})\cdot\mathbf{a}$' that has the same module as the correct result but it has the opposite sign.
- The correct multiplication result is obtained by reversing the sign of the previous expression. Therefore, the calculation formula is: $\mathbf{a}\cdot\mathbf{b}-2^m\cdot\mathbf{a}$.

The VHDL implementation of this principle is described by the process in Code Fragment 6.10, where the information is transferred from the internal register **int_prod** to the output port **prod** in two manners, depending on the most significant bit of **b**. If **b(m-1)**='0' then the operand b is positive and the internal information is copied to the output port, while if **b(m-1)**='1' then the previous calculation formula is used.

```
-- Code Fragment 6.10
process(a,b,int_prod)
  begin
    if b(m-1)='0' then
      prod<=int_prod(n+m-1 downto 0);
    else
      prod<=int_prod(n+m-1 downto 0) - (a & zeroes(m));
    end if;
  end process;
```

Fig. 6-12 presents the pipelined architecture of data_tier1 that includes the multiplier previously discussed. The first operation performed is the calculation of the

rectangular components i_x and i_y of the stator current space vector. The calculation is carried out using a modified form of the classical conversion equations. Thus, the two components are replaced by equivalent values that are rescaled to limit their maximal values and to limit the number of bits necessary to be allocated for each of them. The rescaling technique allows a good control over the number of bits used by each internal signal, but on the other hand decreases the calculation precision of data_tier1. Furthermore, the rescaling factors need to be taken into account by the subsequent calculations that involve the equivalent quantities. The simulation and the synthesis results showed that rescaling with 0.5 offers the best trade-off between precision and hardware complexity. Therefore, the calculations are performed according to:

$$\begin{cases} i_x = \frac{3}{2}\cdot i_a & \Rightarrow \quad i_x^{eqv} = \frac{i_x}{2} = \frac{3}{4}\cdot i_a \\ i_y = \frac{\sqrt{3}}{2}\cdot(2i_b + i_a) & \Rightarrow \quad i_y^{eqv} = \frac{i_y}{2} = \frac{\sqrt{3}}{2}\cdot\left(i_b + \frac{i_a}{2}\right) = 0.866\cdot\left(i_b + \frac{i_a}{2}\right) \end{cases} \tag{6-7}$$

The multiplication with 0.866 required in (6-7) is performed by the multiplier integrated in data_tier1. Once the two rectangular components have been determined, the current error vector $\Delta\underline{i}_{ref}(k)=\underline{i}_{ref}(k+1)-\underline{i}(k)$ and the current variation $\Delta\underline{i}(k)=\underline{i}(k)-\underline{i}(k-1)$ are calculated by simple subtracters. The components of $\Delta\underline{i}(k)$ are multiplied with the estimated inductance L and subtracted from the corresponding voltage components to determine the vector $\underline{V}_{ni}$, which is used by both the neural network generating the PWM signal and by data_tier1 to calculate the vector $\underline{V}_{\Delta}$.

The adapter blocks included in Fig. 6-12 enhance the operation precision of the angle subnetwork inside the neural network contained by tier2. The angle subnetwork calculates the argument of a space vector based on its rectangular co-ordinates. The number of input bits of the angle subnetwork is smaller than the total number of bits of the two co-ordinates. Therefore, it uses only the most significant **n** bits of these co-ordinates. If the two values are small numbers, the most significant bits are all '0' or all '1' (depending on the sign of the numbers) and an incorrect result is generated. The adapter simultaneously shifts the two co-ordinate values to the left until their leftmost n positions contain significant bits. Shifting a binary number to the left is equivalent to a multiplication by a power of two. As the two co-ordinates are simultaneously multiplied with the same power of two, the adapter actually amplifies the module of the vector but leaves the vector argument unchanged.

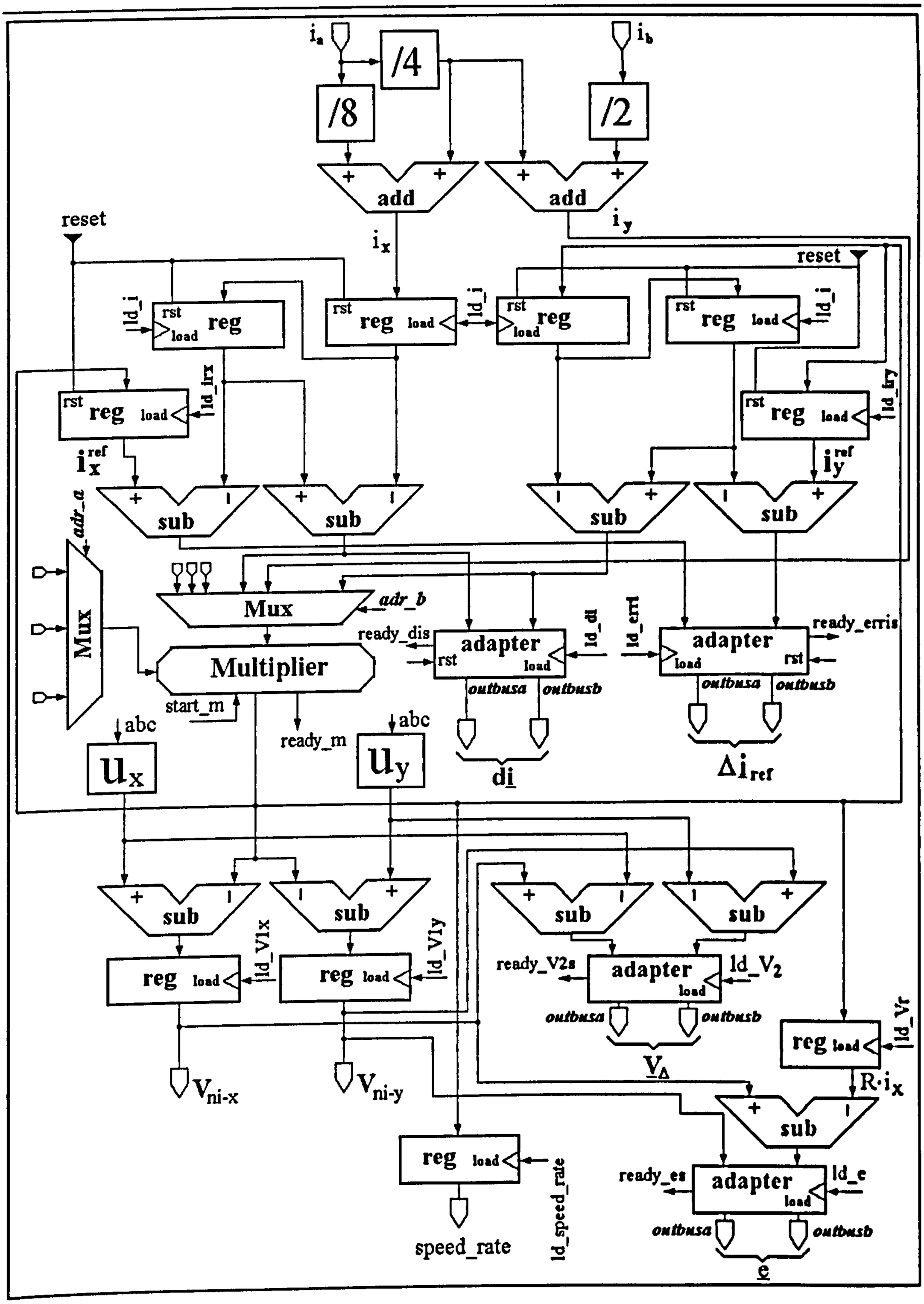

Fig. 6-12 - The structure of Data_Tier1

The VHDL model of the adapter has two main input ports **inbusa** and **inbusb** (the initial two co-ordinates) and two main outputs **outbusa** and **outbusb**, as shown in Code Fragment 6.11. The generic parameter 'n' defines the width of the input and the output busses. It has to be set in accordance with the width of the VHDL signals to which it is

connected. All the signals in this tier have correlated widths that are primarily determined by a generic parameter 'ni' which determines the width of the busses i_a and i_b presented in Fig. 6-12.

```
-- Code Fragment 6.11
entity adapter is
    generic(n: integer);
    port (
        inbusa: in STD_LOGIC_VECTOR (n-1 downto 0);
        inbusb: in STD_LOGIC_VECTOR (n-1 downto 0);
        outbusa: out STD_LOGIC_VECTOR (n-1 downto 0);
        outbusb: out STD_LOGIC_VECTOR (n-1 downto 0);
        clk: in STD_LOGIC;
        ld: in STD_LOGIC;
        ready: out STD_LOGIC
    );
end adapter;
```

The additional input **ld** triggers the shifting process while **ready** signals the moment when the process is finished. Each step of the process is synchronised by the clock signal **clk**. The method to determine the end of the process is to test the most significant two bits in each of the two partial results. If any of the two pairs contains different bits then the process must stop to avoid an overflow that would change the sign of at least one of the co-ordinates. The process must also be stopped if all the bits are zero in the same time. This happens when both input co-ordinates are simultaneously zero, which would cause an infinite shifting process. The architecture of the adapter contains two VHDL processes: the first shifts the two input values in a synchronised manner, while the second verifies the existence of non-zero bits and communicates the result through the internal signal **not_all_zero**:

```
-- Code Fragment 6.12
architecture adapter_arch of adapter is
  signal int_busa,int_busb: STD_LOGIC_VECTOR(n-1 downto 0);
  signal not_all_zero: std_logic;
begin
  process(ld,clk,inbusa,inbusb)
  begin
    if clk='1' and clk'event then
      if ld='1' then
        int_busa<=inbusa;
        int_busb<=inbusb;
        ready<='0';
      elsif (int_busa(n-1)=int_busa(n-2)) and
            (int_busb(n-1)=int_busb(n-2))
              and (not_all_zero='1') then
        int_busa<=int_busa(n-1) & int_busa(n-3 downto 0) & '0';
        int_busb<=int_busb(n-1) & int_busb(n-3 downto 0) & '0';
      else
        ready<='1';
      end if;
    end if;
```

```
  end process;
process(int_busa,int_busb)
    variable x: std_logic;
  begin
    x:='0';
    for i in 0 to n-1 loop
      x:=x or int_busa(i);
      x:=x or int_busb(i);
    end loop;
    not_all_zero<=x;
  end process;
  outbusa<=int_busa;
  outbusb<=int_busb;
end adapter_arch;
```

Fig 6-13 presents the state diagram that describes the operating sequence of tier1, which is controlled by ctrl_tier1 modelled as a pair of correlated state machines. The first dictates the sequence of mathematical operations performed by tier1 while the other controls the interface with the A/D converters. The A/D circuits TLC1550 produced by Texas Instruments [1] have been used, as they offer the advantage of integrating the sample-and-hold circuit, an internal clock oscillator and the digital converter itself in the same chip. Moreover, this type of chip can be easily interfaced with other digital circuits due to the 3-state parallel port, and it can be addressed as an external memory device. Thus, it has a **RD** input pin that triggers the conversion and an active low **EOC** output pin that signals the end of the conversion. The interface state machine activates **RD** in state S28 and then waits for the conversion to finish. After each conversion, it loads the information in the specialised registers, as shown in Fig 6-13 correlated with Fig. 6-12. The operation of the first state machine in Fig 6-13 is described by a linear sequence of states which controls the multiplier, the associated multiplexers and loads each partial result in the specially allocated register. The activity of the two state machines is correlated by means of the internal signal **RdNow** that is used to indicate the moments when the analogue to digital conversion is finished. Each operation cycle of ctrl_tier1 is triggered by the **start_tier1** signal, which has a frequency of 150 kHz and it is generated by ctrl_tier0. As a result, the A/D converters are activated with the same frequency and therefore the sampling frequency of the motor controller is 150 kHz as well.

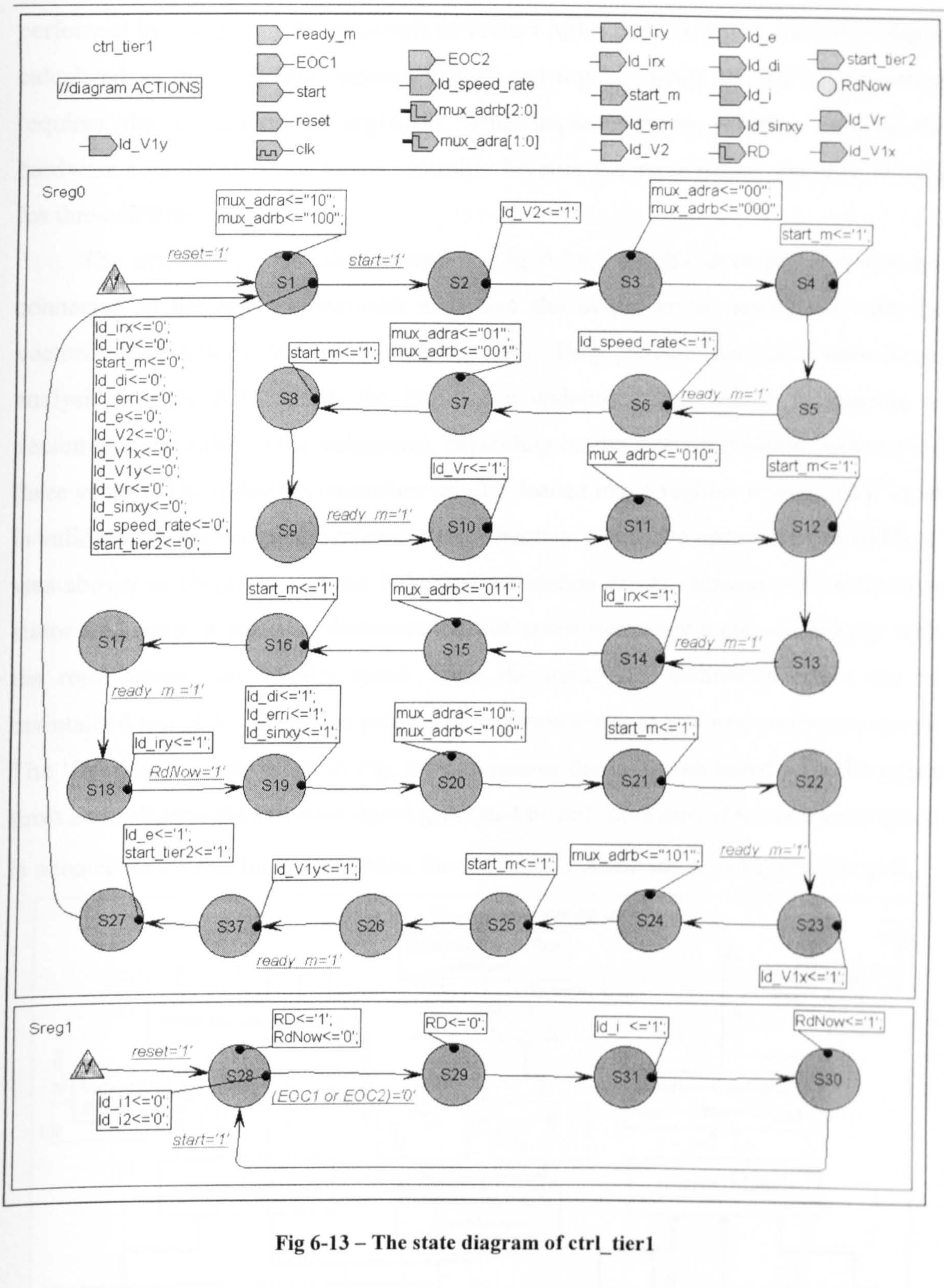

Fig 6-13 – The state diagram of ctrl_tier1

6.3 THE PWM GENERATION AND THE ON-LINE INDUCTANCE ESTIMATION

The data processing unit of tier2 named data_tier2 (Fig. 6-14) contains the PWM signal generator, which includes the neural network presented in chapter 5, the on-line inductance estimator and the motor slip calculator. All these three digital structures use information provided by the angle subnetwork. Thus, the inductance estimation is

performed by comparing the arguments of vectors $\Delta\underline{i}(k)$, $\Delta\underline{i}(k-1)$, $\underline{V}_\Delta(k)$, the motor slip is calculated as the difference between $\arg\{\underline{e}\}$ and $\arg\{\underline{i}_s\}$, while the PWM generation requires the calculation of $\arg\{\Delta\underline{i}_{ref}\}$. This implementation solution reduces the hardware complexity of the motor controller because the same neural structure is used for three different purposes.

The structure of data_tier2, shown in Fig. 6-14, includes three registers that are connected to the angle subnetwork and store the output codes associated with the vectors $\Delta\underline{i}(k)$, $\Delta\underline{i}(k-1)$, $\underline{V}_\Delta(k)$ calculated by tier1. They provide this information to an analysis module that controls the inductance updating block, which increments or decrements the value of the inductance, depending on the relative position between the three vectors. The induction estimation result is loaded into a register whose 'load' input is validated by comparing the reference rotor frequency with the upper limit of 10 Hz. It was shown in chapter 4 that the induction estimation errors increase with increasing stator frequency. It was also demonstrated that stator frequency increases linearly with the rotor steady-state angular speed. Thus, the inductance estimation errors can be maintained low if the estimation process is performed only at low rotor reference speed. The 'frequency check' block in Fig. 6-14 compares the reference speed with the upper limit and validates the **estimate** signal generated by ctrl_tier2 only if the reference speed is situated below this limit. Otherwise, the estimated inductance $\hat{L}$ is kept unchanged.

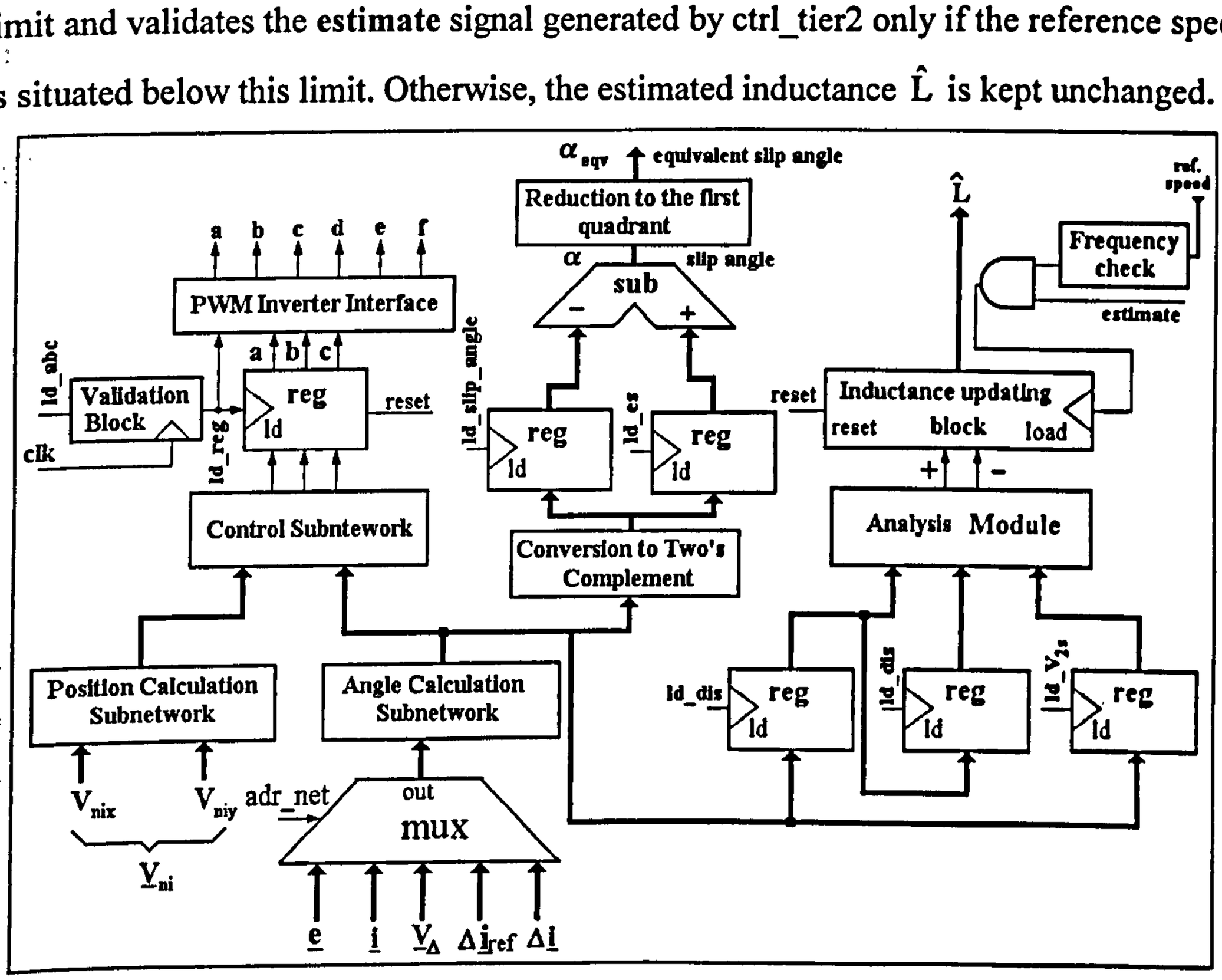

Fig. 6-14 - The RTL description of data_tier2

According to the induction estimation algorithm discussed in chapter 4, the analysis module has to determine which vector ($\Delta\underline{i}(k)$, $\Delta\underline{i}(k-1)$ or $\underline{V}_{\Delta}(k)$) is situated between the other two. As demonstrated in chapter 5, the angle subnetwork divides the 360^{o} interval into a number of equal sectors. All the output codes associated with space vectors that belong to a given group of consecutive sectors correspond to binary codes that share a certain number of identical bits. One space vector is situated between the other two if it is included in the group of sectors delimited by the two vectors. Therefore, the relative position of the space vectors can be determined by analysing the codes associated by the angle subnetwork. This method is illustrated by an example in Table 6-1 that involves three vectors $\underline{v}_2$, $\underline{v}_4$, $\underline{v}_6$ situated in the sectors 2, 4 and 6. It can be calculated that $\underline{v}_4$ lies between $\underline{v}_2$ and $\underline{v}_6$ because the code associated with $\underline{v}_4$ shares the bits b_5 and b_0 with the codes corresponding to $\underline{v}_2$ and $\underline{v}_6$. On the other hand, the vectors $\underline{v}_2$ and $\underline{v}_4$ share the bits b_5, b_2, b_1, b_0. Therefore, $\underline{v}_6$ is not situated between $\underline{v}_2$ and $\underline{v}_4$ because the code of $\underline{v}_6$ does not share the bits b_2 and b_1 with the other two codes.

Table 6-1 - The output codes of an angle subnetwork with n=6 neurones

Sector Index	Angle Interval	Code					
		b_5	b_4	b_3	b_2	b_1	b_0
1	$[-15^0; 15^0)$	0	0	0	0	0	0
2	$[15^0; 45^0)$	1	0	0	***0***	***0***	0
3	$[45^0; 75^0)$	1	1	0	0	0	0
4	$[75^0; 105^0)$	1	1	1	***0***	***0***	0
5	$[105^0; 135^0)$	1	1	1	1	0	0
6	$[135^0; 165^0)$	1	1	1	***1***	***1***	0
7	$[165^0; 195^0)$	1	1	1	1	1	1
8	$[195^0; 225^0)$	0	1	1	1	1	1
9	$[225^0; 255^0)$	0	0	1	1	1	1
10	$[255^0; 285^0)$	0	0	0	1	1	1
11	$[285^0; 315^0)$	0	0	0	0	1	1
12	$[315^0; 345^0)$	0	0	0	0	0	1

The PWM inverter interface transforms the three bits 'abc' defining the desired inverter output voltage into a vector with six control signals 'abcdef' that are transmitted to the power transistors. The edges of the signals controlling the transistors in the same

inverter leg (a-d, b-e, and c-f) do not occur simultaneously so that short-circuits are avoided. Thus, a transistor is turned on at 2.5 μs after its counterpart in the inverter leg has been turned off. This is achieved by using the hardware structure in Fig. 6-15 where the control signals are generated by AND gates whose outputs depend both on the current bits and on the previous bits generated by the neural network. The previous bits are stored into a 6-bit register that is loaded at 2.5 μs after the neural network new output has been transmitted to the interface block. If one of the bits was previously '1' while the current value is '0', then the corresponding AND gate output changes from '1' to '0' immediately. If the previous value was '0' and the current value is '1' the AND gate output transition cannot occur immediately because the gate inputs are different for a period of 2.5 μs. The 2.5 μs delay is generated by a down-counter that is reset by the same signal **ld_reg** which loads the 'abc' register (Fig. 6-14).

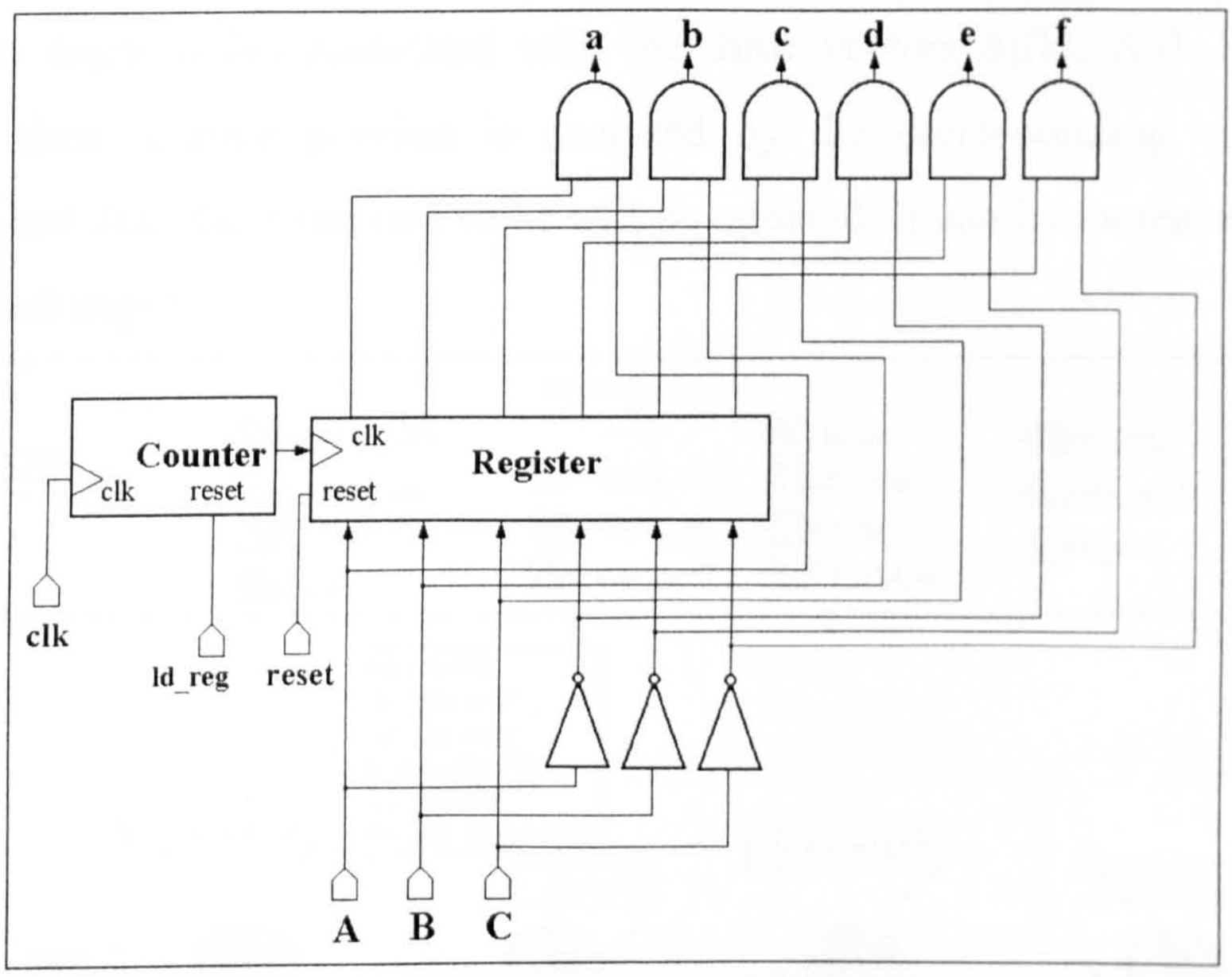

Fig. 6-15 - The PWM inverter interface

The signal **ld_reg** is generated by the validation block, which is a modulo 9 counter. Therefore, the frequency of **ld_reg** is 10 times smaller than the frequency of **ld_abc**. The signal **ld_abc** is activated once during each operation cycle of the motor controller. The operation cycles are initiated by tier0 with a frequency of 150 kHz, which entails that the signal **ld_reg** has a frequency of 15 kHz. Therefore, in this configuration the frequency of the PWM signal generated by the motor controller is 15 kHz.

The control unit of **tier2** (Fig. 6-16) synchronises the operation of the multiplexer connected to the angle subnetwork in Fig. 6-14 with the activity of the registers and the

operation of the inductance updating block. The operation cycle of the control unit is described by Fig. 6-16 and it consists of the following steps:

1. The stator current vector is supplied to the angle subnetwork and after the calculations the three-bit vector abc is loaded in the corresponding register.
2. The shifted components of vector $\underline{e}$ are analysed and the corresponding value is stored in the specialised register by activating signal **ld_es**.
3. The angle of the current space vector $\underline{i}_s$ is determined and the quantity proportional with the slip angular frequency is determined.
4. The code corresponding to the argument of the vector $\underline{V}_\Delta$ is calculated and stored in a register.
5. The argument of $\Delta\underline{i}(k)$ is calculated and it replaces the value of $\Delta\underline{i}(k-1)$ which is transferred into a second register (Fig. 6-14).
6. Once the angle codes associated with the three vectors $\Delta\underline{i}(k)$, $\Delta\underline{i}(k-1)$, $\underline{V}_\Delta(k)$ are known, their relative position is analysed by the corresponding combinational module and then the estimated value of L is updated. It can be increased, decreased or left unchanged.

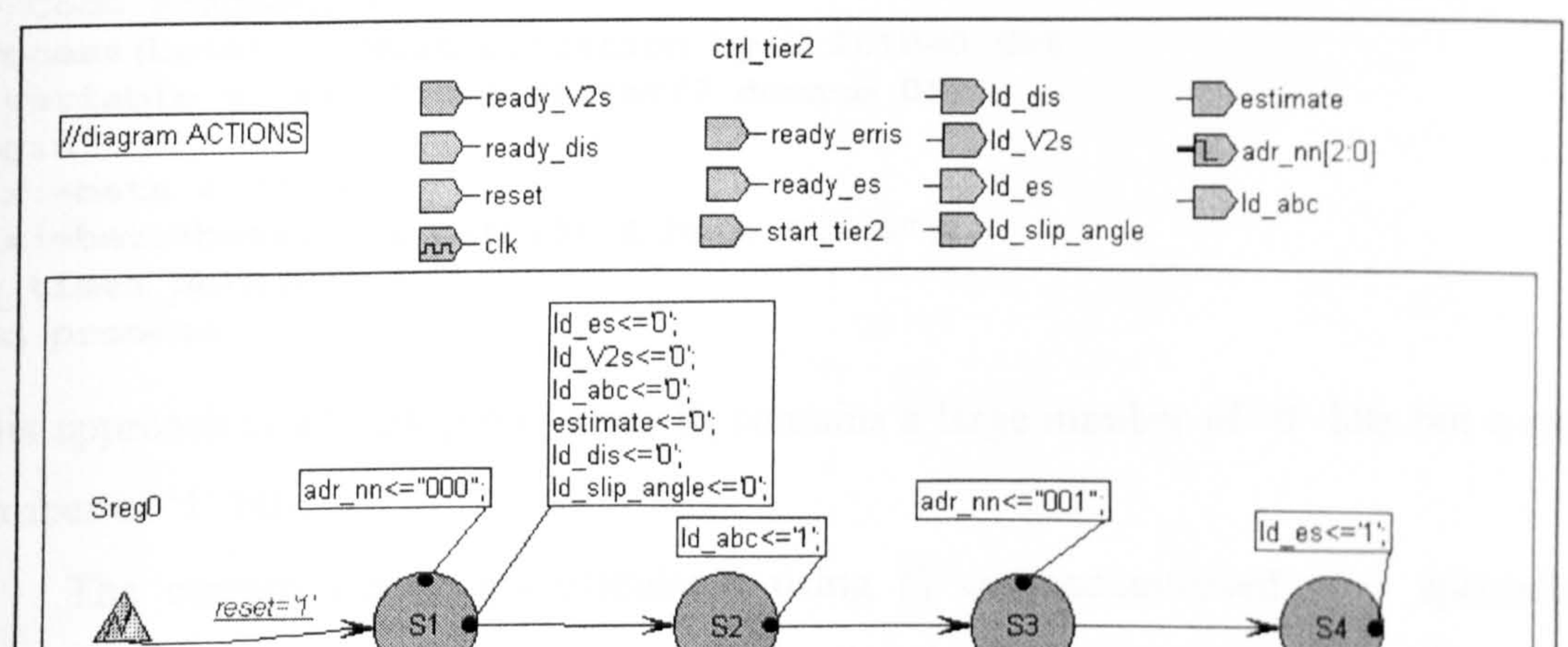

Fig. 6-16 – The state diagram of ctrl_tier2

Each operation cycle starts when the necessary input information, calculated by tier1, is available on the input ports of tier2. The appropriate moment to start the operation of tier2 is indicated by the signal **start_tier2** generated by tier1. On the other hand, ctrl_tier2 waits for the four adapters included inside tier1 to finish their shifting tasks. As shown in Fig. 6-16, the signals **ready_erris, ready_es, ready_V2s** and **ready_dis** generated by the adapters in Fig. 6-12 are used to test the validity of the input information before it is processed by the angle subnetwork.

6.4 THE IMPLEMENTATION OF THE SPEED CONTROL STRATEGY

Tier3 calculates the stator frequency and the stator current amplitude using the slip angle and the rotor reference speed. The calculations are performed according to the control principles discussed in chapter 4. The simplified function F_I using a single proportionality constant K_I has been used to minimise the hardware structure of this tier. To simplify the VHDL model even further, the multiplication with K_I has been modelled as a set of shifts and additions as follows:

```
-- Code Fragment 6.13
process(beta) -- Multiplication by 0.101B=0.625
  variable x: std_logic_vector(7 downto 0);
begin
  x:=beta & "0000";
  x:=beta+beta(3) & beta(3) & beta & "00";
Ki_times_beta<=x;
end process;
```

This approach is advantageous when K_I contains a large number of '0' bits but a small number of '1' bits.

The current increments calculated using F_I are accumulated in a specialised register. The accumulation process can be inhibited if I_s decreases under the limit $I_{s\text{-}min}$ or increases over $I_{s\text{-}max}$, thereby maintaining the current amplitude within the acceptable range of values (Fig. 6-17). On the other hand, the speed derivative function F_ω has been implemented as a look-up table. The angle neural network divides the 360° interval into 36 sectors, which means that only 9 sectors are allocated to each quadrant. Therefore, the look-up table is small and can be modelled by a **case** VHDL statement:

```
-- Code Fragment 6.14
process(alpha)
begin
  case alpha is
    when "0000" => F_omega<="00011";
    when "0001" => F_omega<="00011";
    when "0010" => F_omega<="00011";
```

```
    when "0011" => F_omega<="00100";
    when "0100" => F_omega<="00110";
    when "0101" => F_omega<="01000";
    when "0110" => F_omega<="01010";
    when "0111" => F_omega<="01100";
    when "1000" => F_omega<="01110";
    when others => F_omega<="01111"
  end case;
end process;
```

This approach has the advantage that non-linear versions of F_ω can be implemented with the same hardware resources as the piecewise linear versions. The values generated by F_ω are always positive. They need to be added or subtracted depending on the difference between the reference speed and the calculated speed of the rotor. Furthermore, the constant quantity Ω_{slp} needs to be added to or subtracted from the previously obtained result depending on the sign of the reference speed. All these situations are analysed by the simple interconnection of adders, subtracters and multiplexers shown in Fig. 6-17.

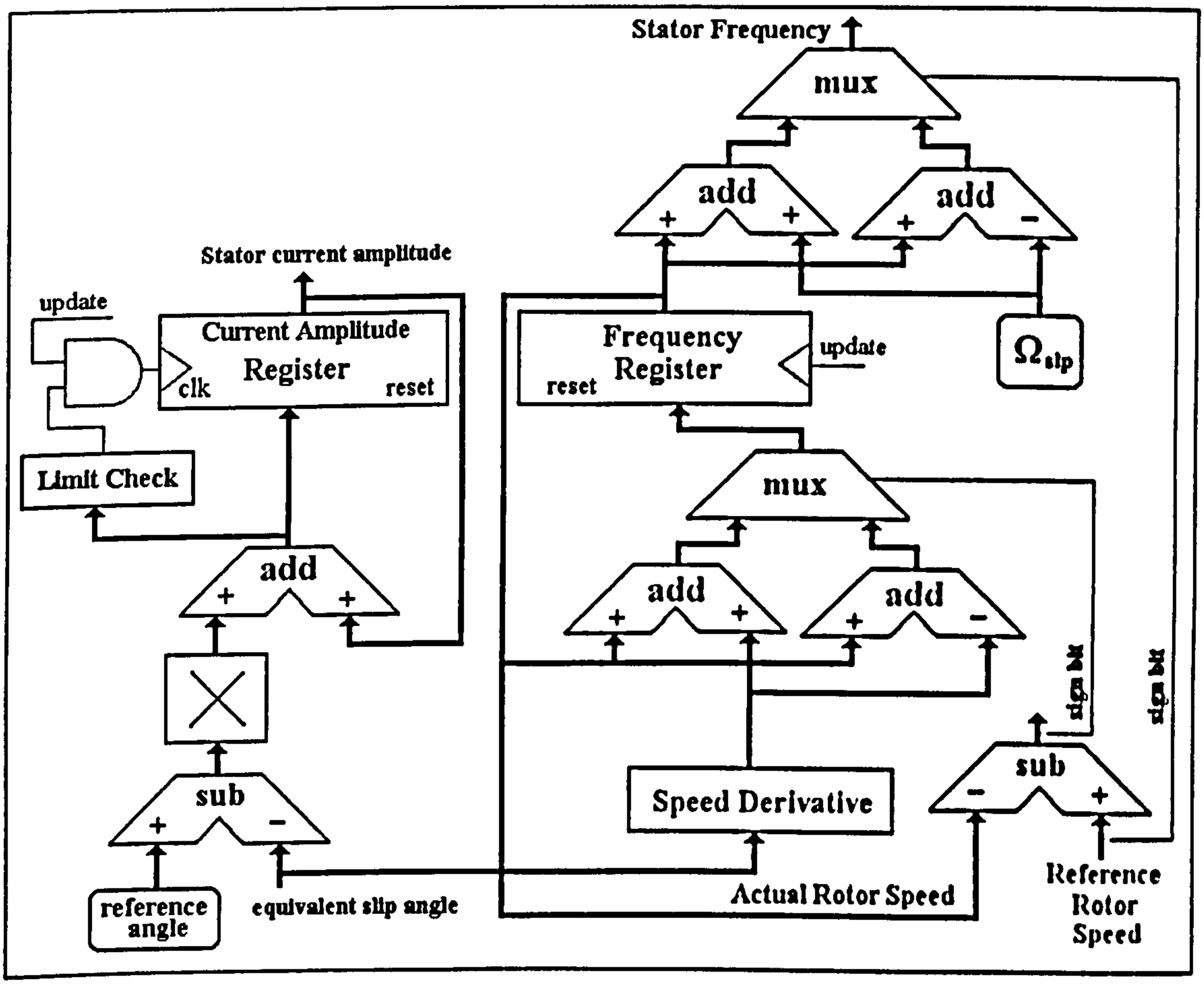

Fig. 6-17 – The structure of Tier3

The operation of tier3 does not require a control unit. Al the results are updated when the input signal **update** is activated by tier2. This input signal is connected to the

signal **estimate** generated by ctrl_tier2. The signal **estimate** is activated at the end of the operation cycle of tier2 after the motor slip angle has been calculated. Therefore, it indicates an appropriate moment for tier3 to perform its calculations as the slip angle α_{eqv} is one of the two input data used by this tier.

6.5 THE COMPLETE MOTOR CONTROLLER SIMULATIONS

A VHDL testbench has been developed by combining the model of the complete controller with the VHDL model of the three-phase induction motor presented in chapter 4. Several simulations have been performed using Workview Office software package with different values of the generic parameters involved in the controller description, in order to test its correct operation. The simulations demonstrate the controller capability to generate correct PWM signals (Fig. 6-18), to accurately control the stator current and to determine the motor equivalent inductance.

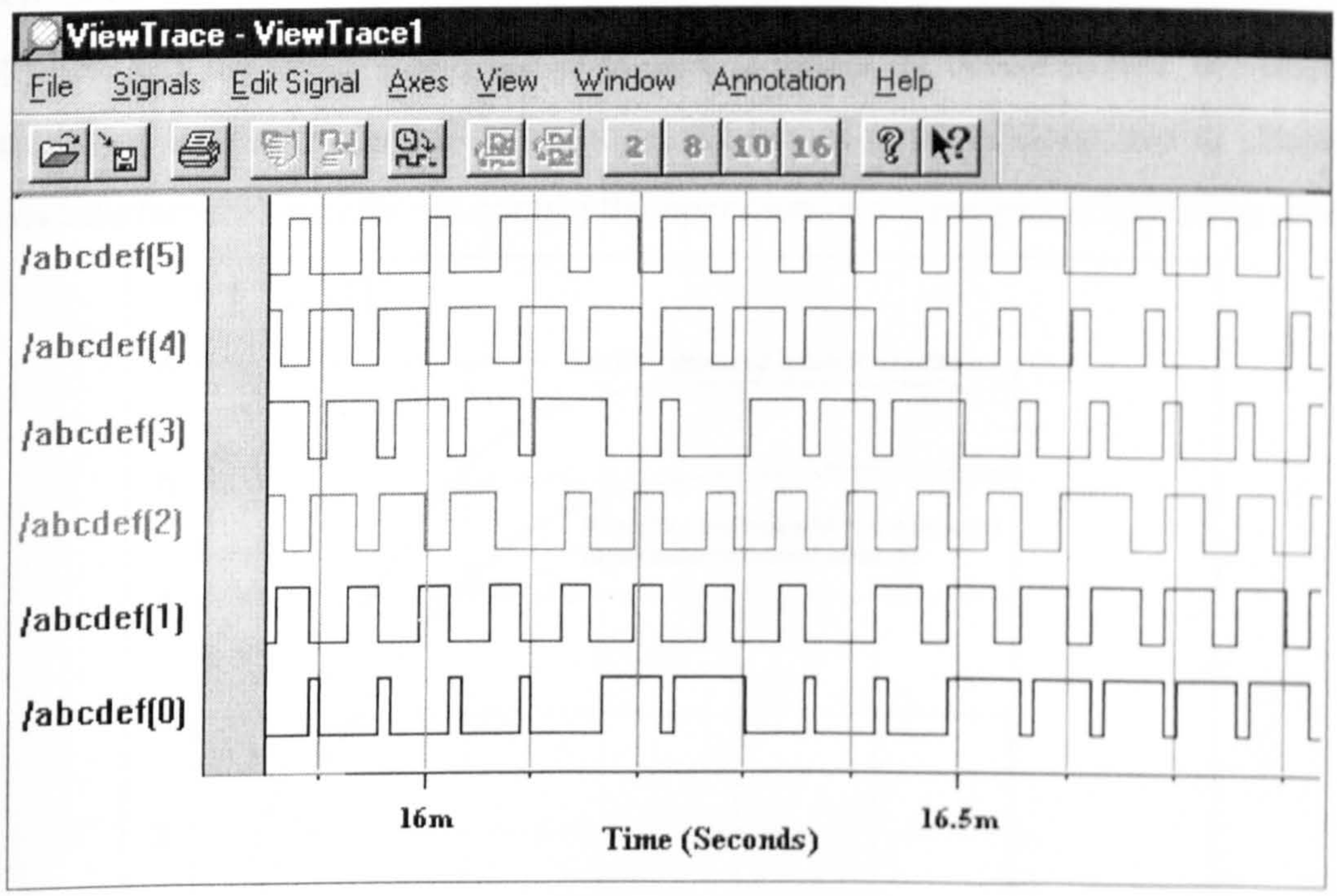

Fig. 6-18 – Motor controller simulation results

As discussed in chapter 5, the precision of the neural network generating the PWM signal is restricted by the limited number of Voronoi cells. However, given an adequately high number of cells, the operation imprecision is sufficiently low to have a negligible effect on the overall operation of the drive system. The parameter values given in chapter 5, ensure sufficiently accurate control of the stator current vector. On the other hand, the on-line induction estimation process is affected by the limited

precision of the angle subnetwork involved (±5° error), and as a result, the final estimated inductance is smaller than the correct value. Fig. 6-19 presents a comparison between the initial simulation results in chapter 4, performed in ideal conditions (perfectly accurate angle calculations), and the simulation results obtained with the controller model that performs angle calculations using the hardware implemented angle subnetwork. The inductance estimation inaccuracy causes errors in the calculation of vectors $\underline{V}_{ni}$ and $\underline{e}$ but this does not affect their average value over several operation cycles of the controller. Thus, the effect of the inductance estimation error over the motor speed control is minimal due to the high inertia of the rotor that filters out the ripples of the control signals, originated by the induction estimation errors. The accuracy of the inductance estimation can be increased by increasing the number of neurones included in the angle subnetwork. However, the simulations performed proved that such an approach brings a minimal improvement in the global behaviour of the drive system, which does not justify the increase of the total hardware complexity of the motor controller. On the other hand, the computer simulations demonstrated the adequate operation of the motor controller including the neural network described in chapter 5, and its satisfactory capability to control the operation of a three-phase induction motor.

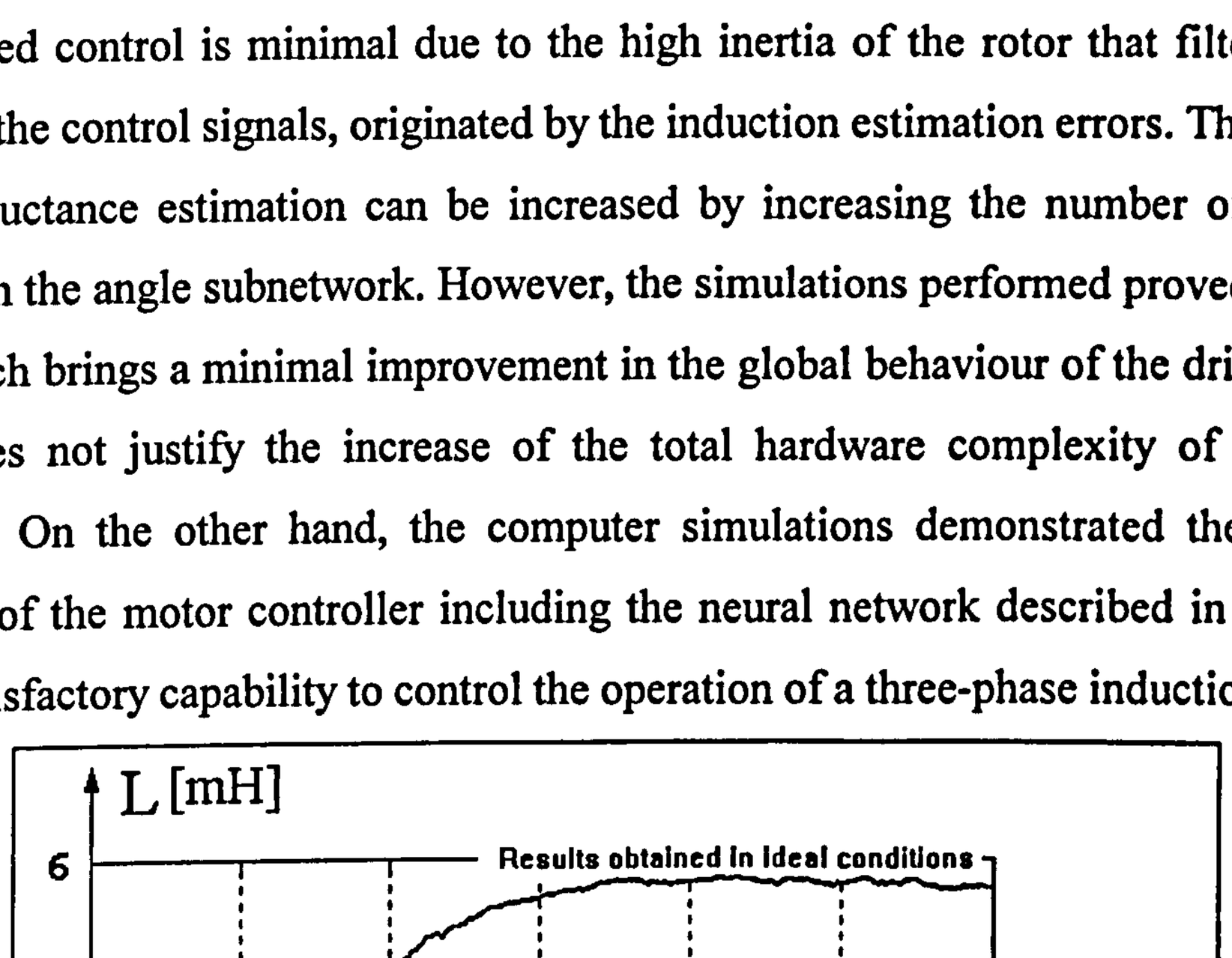

Fig. 6-19 – Comparison of inductance estimation simulation results

After the successful implementation of the motor controller into a Xilinx XC4010XL FPGA, it has been tested in conjunction with a small three-phase induction motor (less than 0.5 kW). The next chapter presents the experimental results obtained.

7. EXPERIMENTAL RESULTS

This chapter presents experimental results relating to the performance of a complete three-phase induction motor drive system controlled by the new neural FPGA controller.

7.1 THE DRIVE SYSTEM

The practical tests have been performed using the FH2 MkIV testbench produced by TecQuipment [4]. The testbench offers the facility to mount up to two electrical machines, DC and AC on the same shaft (Fig. 7-1) and includes speed and torque sensors that allow testing the motor operation.

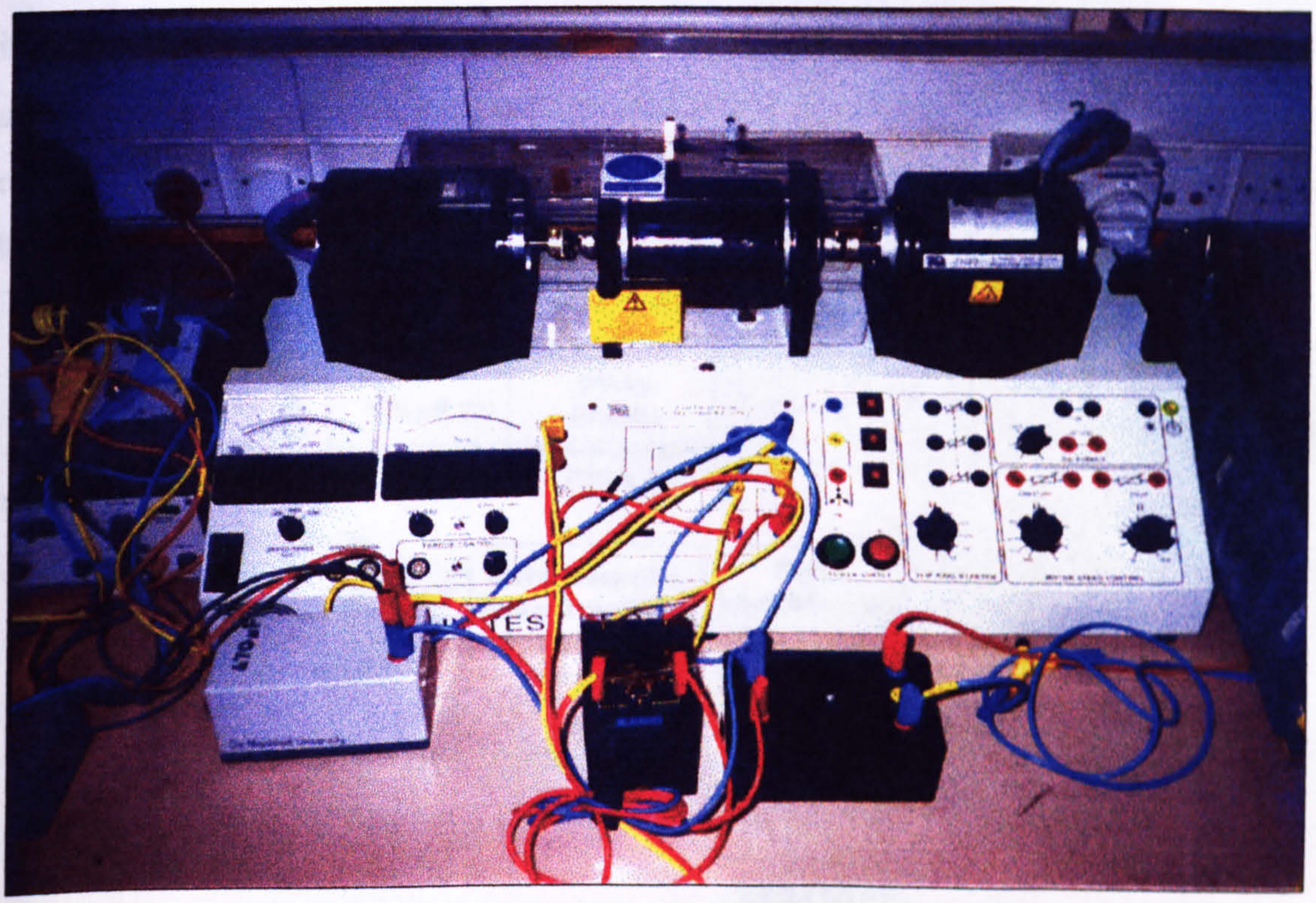

Fig. 7-1 –The FH2 MkIV testbench

The laboratory testbench configuration available includes the FH90 four-pole three-phase cage induction motor and the FH50 DC motor (not used in the experiments). The stator windings are reconfigurable, both Δ-connection and Y-connection being possible. Nevertheless, TecQuipment recommends that the Δ-connection is used. The

rated line voltage is 220V in this configuration, while the line currents have values of up to 1 A, depending on the load torque. The practical experiments proved that the speed control principles discussed in chapter 4 are valid for both Y-connection and Δ-connection. This experimental conclusion is supported by the theoretical possibility of transforming any Δ-connected load supplied by a sinusoidal voltage system into an equivalent Y-connected load. The high frequency harmonics contained by the PWM voltage are filtered by the motor inductances and therefore the corresponding current harmonics are negligible. As a result, only the fundamental harmonics of the voltage and current need to be taken into account and the Y-connected equivalent R-L-e circuit is applicable to Δ-connected motors as well.

The experimental setup that includes the FH2 MkIV testbench and the FH90 induction motor is presented as a block diagram in Fig. 7-2 and is illustrated by the photograph in Fig. 7-3. The motor is supplied by a PWM inverter bridge SKM40GD132D produced by Semikron [6] which contains 1200 V IGBT transistors. The bridge is supplied with DC voltage by a diode rectifier via a low-pass filter. The input voltage of the rectifier can be adjusted using an autotransformer, which allows the smooth control of the DC-link voltage. The IGBT transistors in the PWM inverter are controlled by the XC4010XL FPGA controller on the XS40 test board. This FPGA is a low voltage device that associates the voltage level of 3.3 V with logic '1' [7].

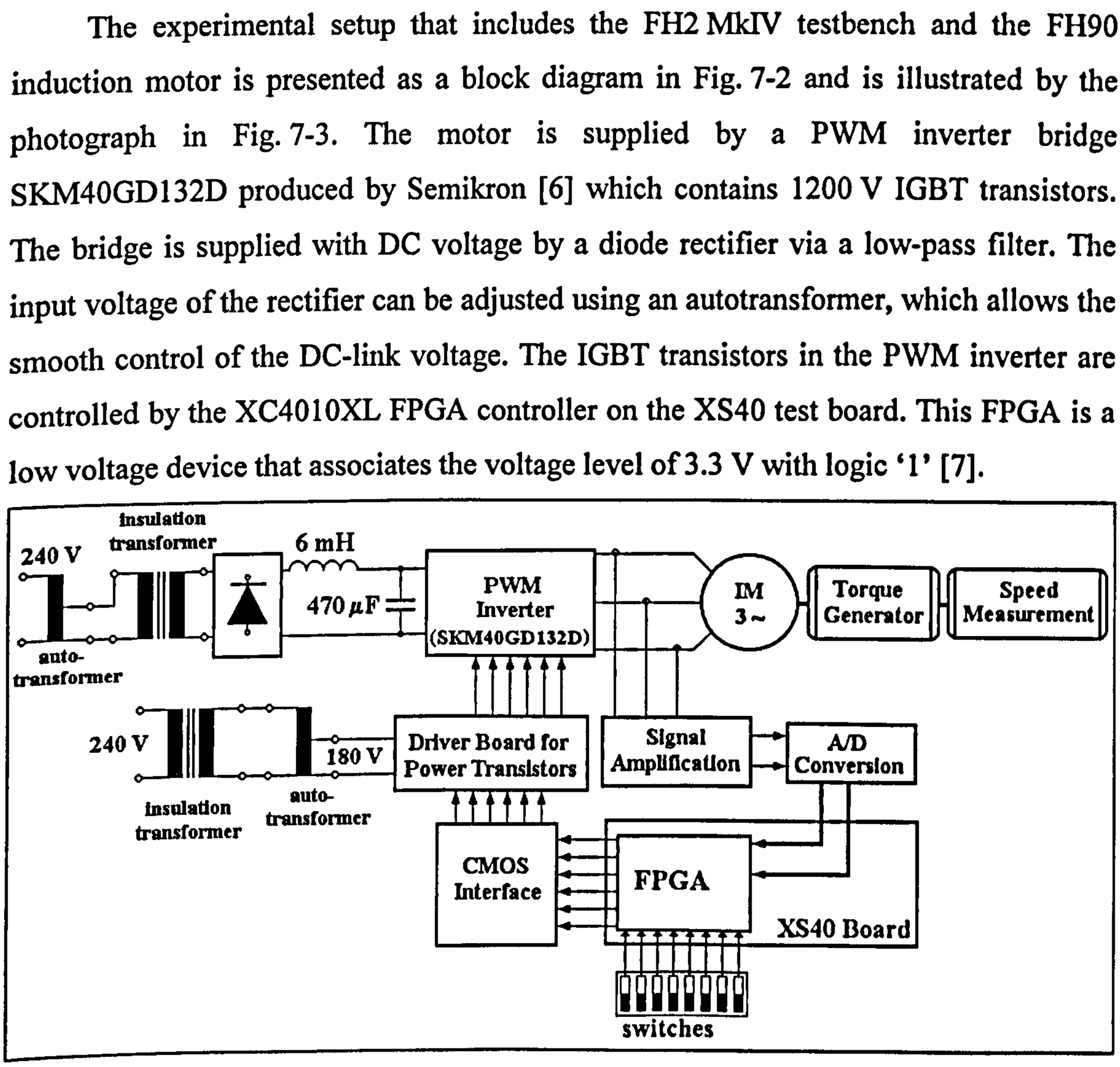

Fig. 7-2 – The schematic of the experimental setup

The voltage level of the control signals is adapted in two stages to the electrical characteristics of the power transistors. First, the CMOS interface in Fig. 7-2 amplifies the output signals of the FPGA to 5V and supplies them to the transistor driver board. In

the same time, the CMOS interface protects the control circuits against the damaging effects of any failure that may occur in the power circuits. In the second stage, the driver board amplifies the control signals to 15 V, which is the control voltage level, recommended by the IGBT manufacturer.

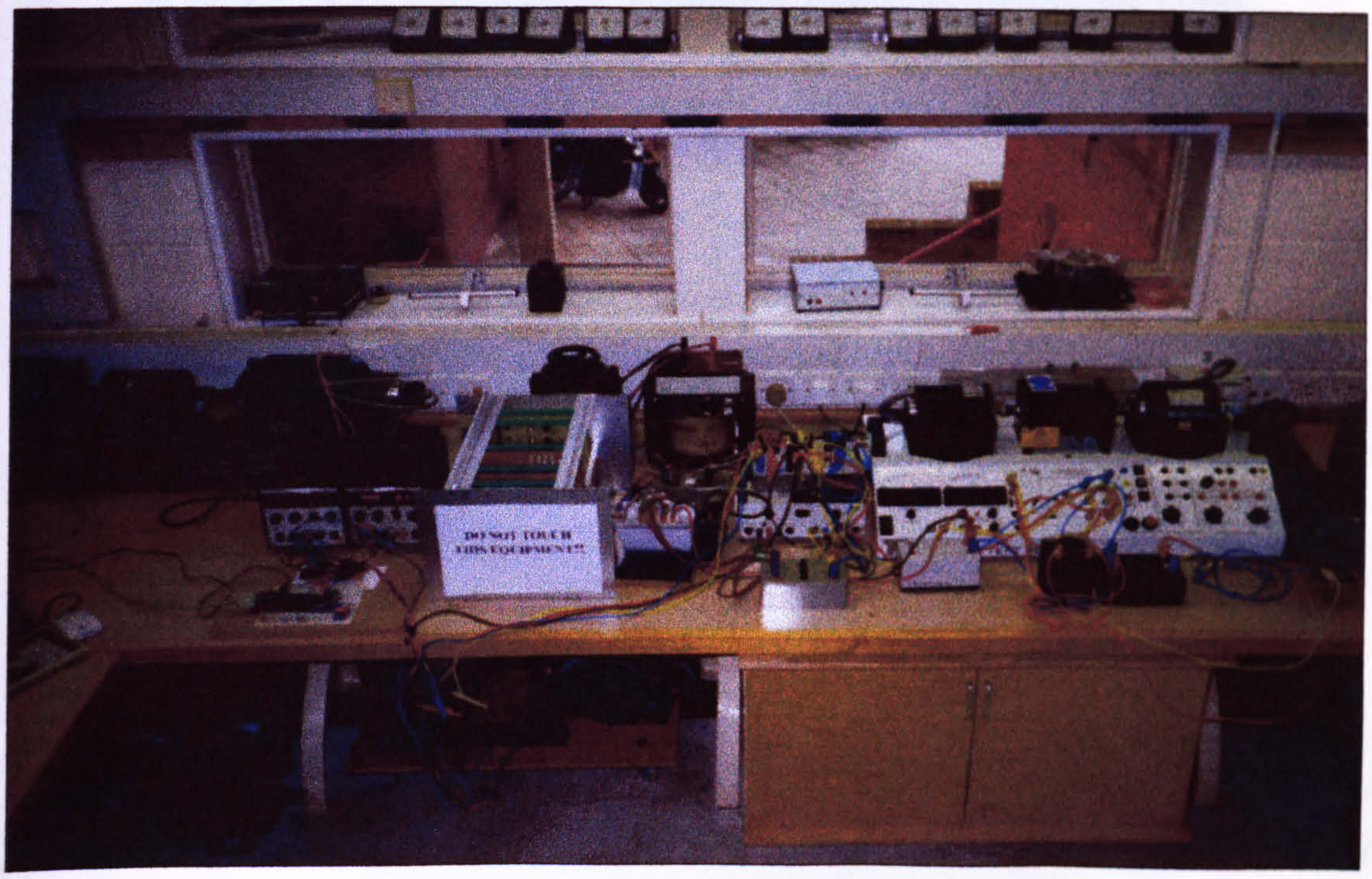

Fig. 7-3 – The experimental set-up

Two of the motor currents are measured using Hall effect transducers that generate a voltage proportional to the measured current, which are then amplified using simple operational amplifiers and transmitted to the TLC1550 10-bit A/D converters produced by Texas Instruments [1]. The binary codes produced by the A/D converters are transmitted to the FPGA controller. As mentioned in chapter 6, the VHDL controller model contains a series of generic parameters defining the size of several internal modules. Many of these parameters are correlated and depend on the width 'ni' of the two current input busses 'ia' and 'ib'. To implement the entire motor controller in a single XC4010XL FPGA, the generic parameter 'ni' was limited to 8. Thus, only the 8 most significant bits are used by the FPGA controller in this configuration. The reference speed of the motor is set in two's complement code using a set of 8 switches. Consequently, the rotor electrical angular frequency ω_{er} can be set at values between -128 and +127 Hz, corresponding to mechanical speeds between –3840 rev/s and +3810 rev/s. The CMOS interface and the operational amplifiers have been implemented on a single interface board, illustrated in Fig. 7-4 together with the XS40 board.

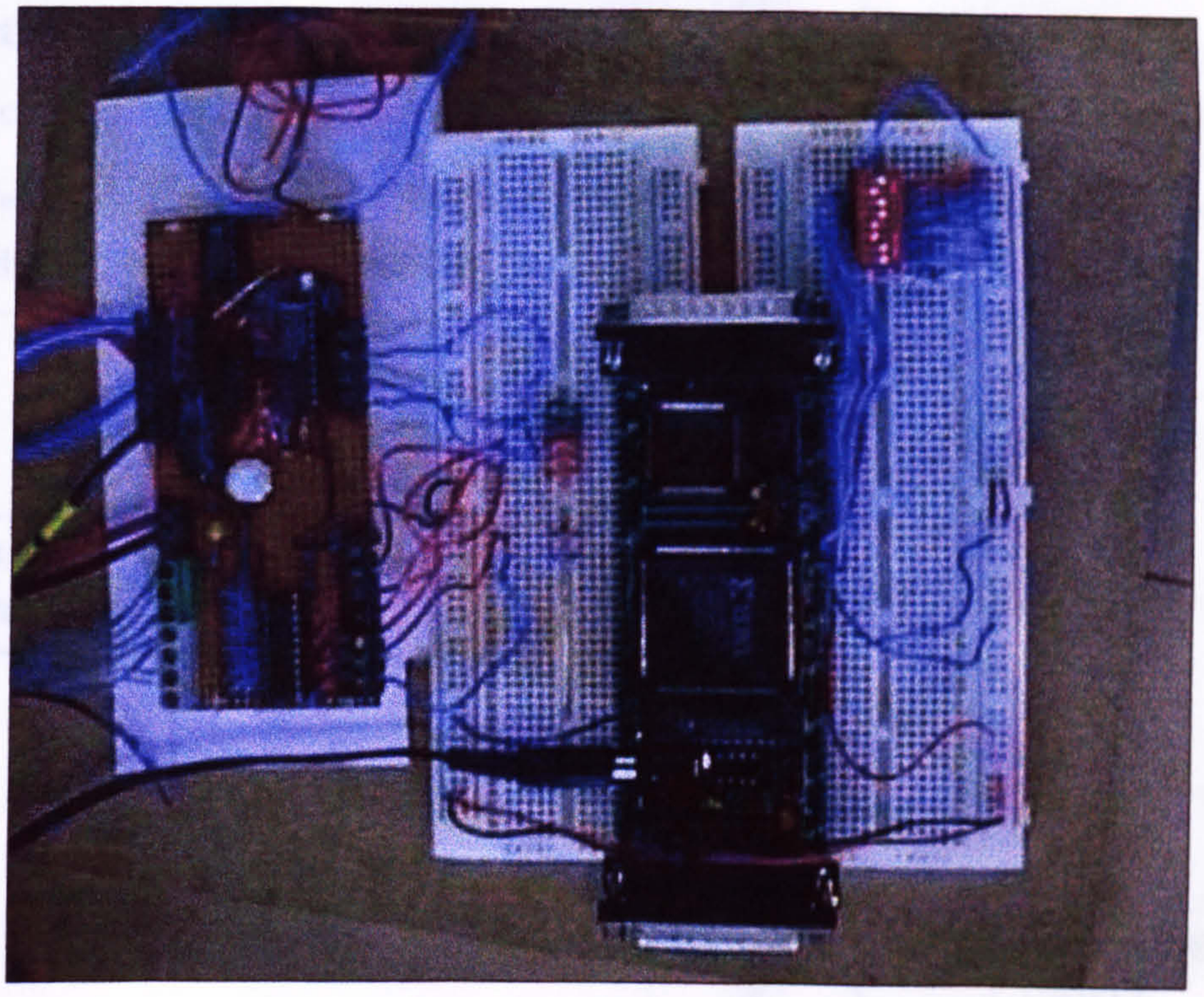

Fig. 7-4 – The XS40 board and the interface board

The parameters of the speed control algorithm implemented by tier3 (see chapter 6) have been determined based on practical experiments with the motor. The equivalent parameters of the FH90 motor were initially determined. The stator resistance was measured directly, the result being 95 Ω, which is a large value for a three-phase induction motor. The equivalent inductance L has been approximated by measuring the voltages across the motor and currents at 50 Hz stator frequency, while the rotor speed was forced to zero. In these conditions, the internal voltage e has a small value and the total impedance of the motor is mostly given by the resistance R and by the equivalent inductance. The measurements and the calculations showed that L is about 220 mH. The critical slip angular frequency was determined while keeping the stator current constant. This was achieved by connecting the rotor windings to variable resistors as shown in Fig. 7-5. The obtained result was $\omega_{slp}^{k} = 95$ rad/s corresponding to a speed of 1050 rev/s (the rated speed is 1500 rev/s). The equivalent slip angle α_{eqv}^{ref} has been arbitrarily set at 65° (according to the considerations in chapter 4, it can have any value between 45° and 90°). The equation

$$\tan^{-1}\left(\alpha_{eqv}^{ref}\right) = \frac{\Omega_{slp}}{\omega_{slp}^{k}} \qquad (7\text{-}1)$$

demonstrated in chapter 4, yields the value Ω_{slp}=44.3 rad/s, corresponding to a slip frequency of 7 Hz. The other parameters that define the speed control algorithm ($K_{\omega 1}$, $K_{\omega 2}$, β_{max}, K_I) have been determined by trial and error in practical experiments that testing the performance of the drive system.

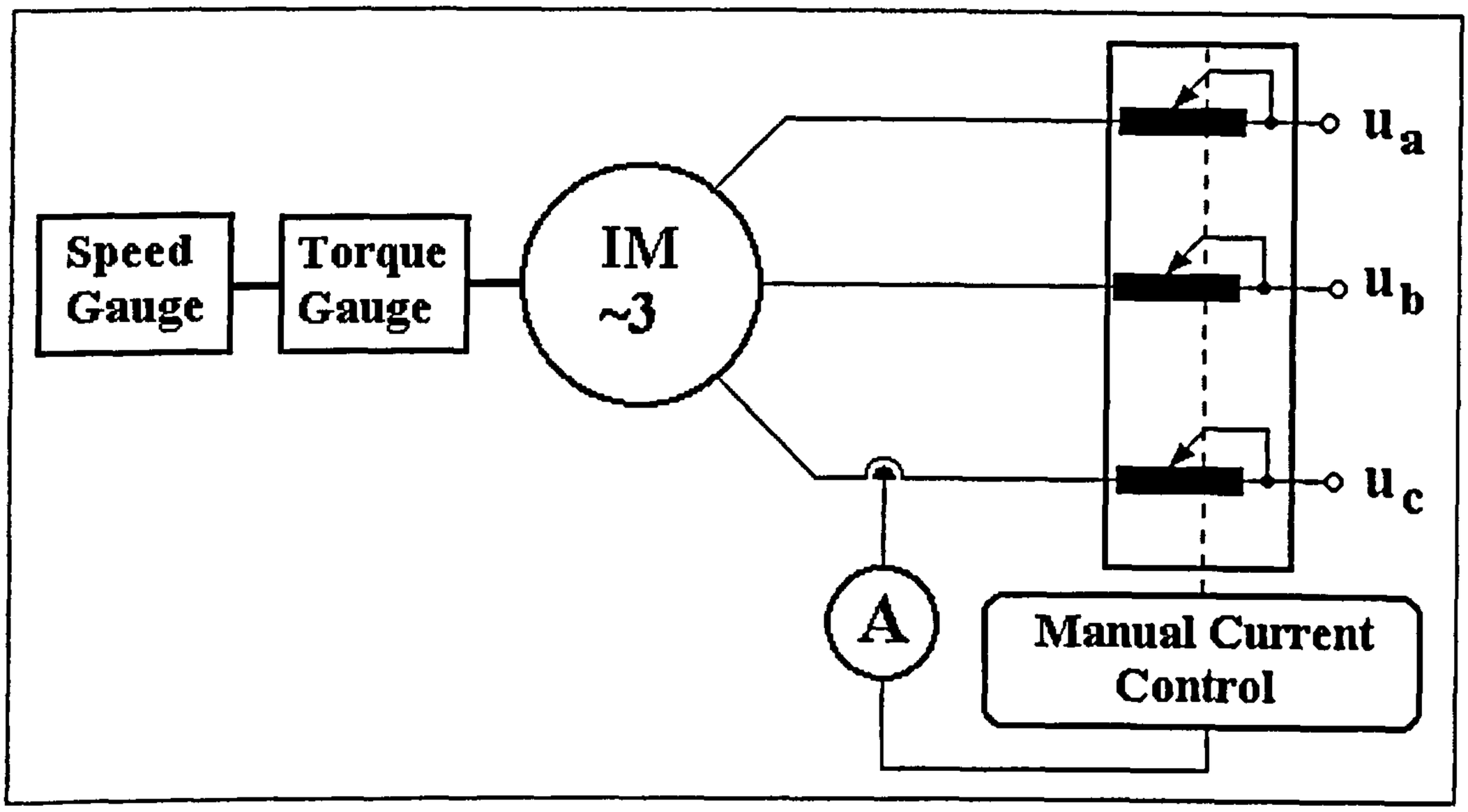

Fig. 7-5 – The measurement of the critical slip at constant current

As a conclusion of the considerations presented in this chapter and in the previous ones, the main characteristics of the adopted VHDL motor controller are:

- The sampling frequency (tier1) f_s=150 kHz
- The PWM frequency (tier2) f_{PWM}=15 kHz
- The maximal rotor speed allowing inductance estimation 600 rev/min (10 Hz)
- The width of the input busses i_a and i_b (tier1) ni=8
- The step length of the multiplier N_{sl}=4
- The stator resistance (defined by a constant in the model of tier1): R_s=95Ω
- The equivalent slip angle α_{eqv}=65°
- The reference slip angular frequency Ω_{slp}=44.3 rad/s
- The parameters defining the speed control dynamics: $K_{\omega 1}$=650 s^{-1}, $K_{\omega 2}$=200 s^{-1}, β_{max}=0.7 rad, K_I=1 A/s
- The switching delay between the two transistors in the same inverter leg: 2.5μs
- The input clock frequency f_{clk}=12 MHz
- The number of samples used to generate the reference sinewave (tier0): 256
- The number of bits for each entry of the look-up table (tier0): 3
- Number of bits used for the reference motor speed: 8

- The number of inverter output voltages used: 6 out of 7 (the null voltage is not used)
- The number of triangular Voronoi cells of the position neural subnetwork : 54
- The number of sectors of the angle subnetwork: 36
- The number of bits used to code the input signals of the position subnetwork: 5
- The number of bits used to code the input signals of the angle subnetwork: 5

In this configuration, the implementation of the controller took up 98% of the hardware resources available on the XC4010XL FPGA. The values of all the parameters can be easily modified by altering a series of constants and generic parameters in the VHDL code describing the model of the motor controller. Consequently, given the appropriate FPGA, the controller can be adapted in terms of hardware complexity and operation accuracy to the requirements of a large range of particular applications.

Two sets of experiments have been carried out. The first set verified the PWM voltage generation and the current control accuracy, while the second set reefer to the speed control performance of the drive system.

7.2 CURRENT AND VOLTAGE CONTROL TESTS

A special version of the controller VHDL model has been created for the first set of experiments. This version lacks tier3 so the frequency of the stator current is identical to the frequency specified by means of the 8 switches and the stator current amplitude is constant. This approach simplifies the testing procedure because it checks the operation of tiers 0,1 and 2 without the feedback signals calculated by tier3 and therefore any operational error can be located much easier.

Fig. 7-6 presents four of the PWM control signals generated by the FPGA motor controller. They have been monitored using a four-channel Hewlett Packard digital oscilloscope. The figure demonstrates the correlation between two of the signals that control the transistor on the same inverter leg (**ctrl(5)** and **ctrl(2)**). Thus, the two signals have complementary values: when one of them is '0' the other is '1'. The 2.5 μs delay generated by the interface block contained in tier2 is not visible in Fig. 7-6 due to the inappropriate time scale (50 μs/div), but it can be observed in Fig. 7-7 where the time scale is 15 μs/div. The overall PWM voltage across the motor can be seen in Fig. 7-8.

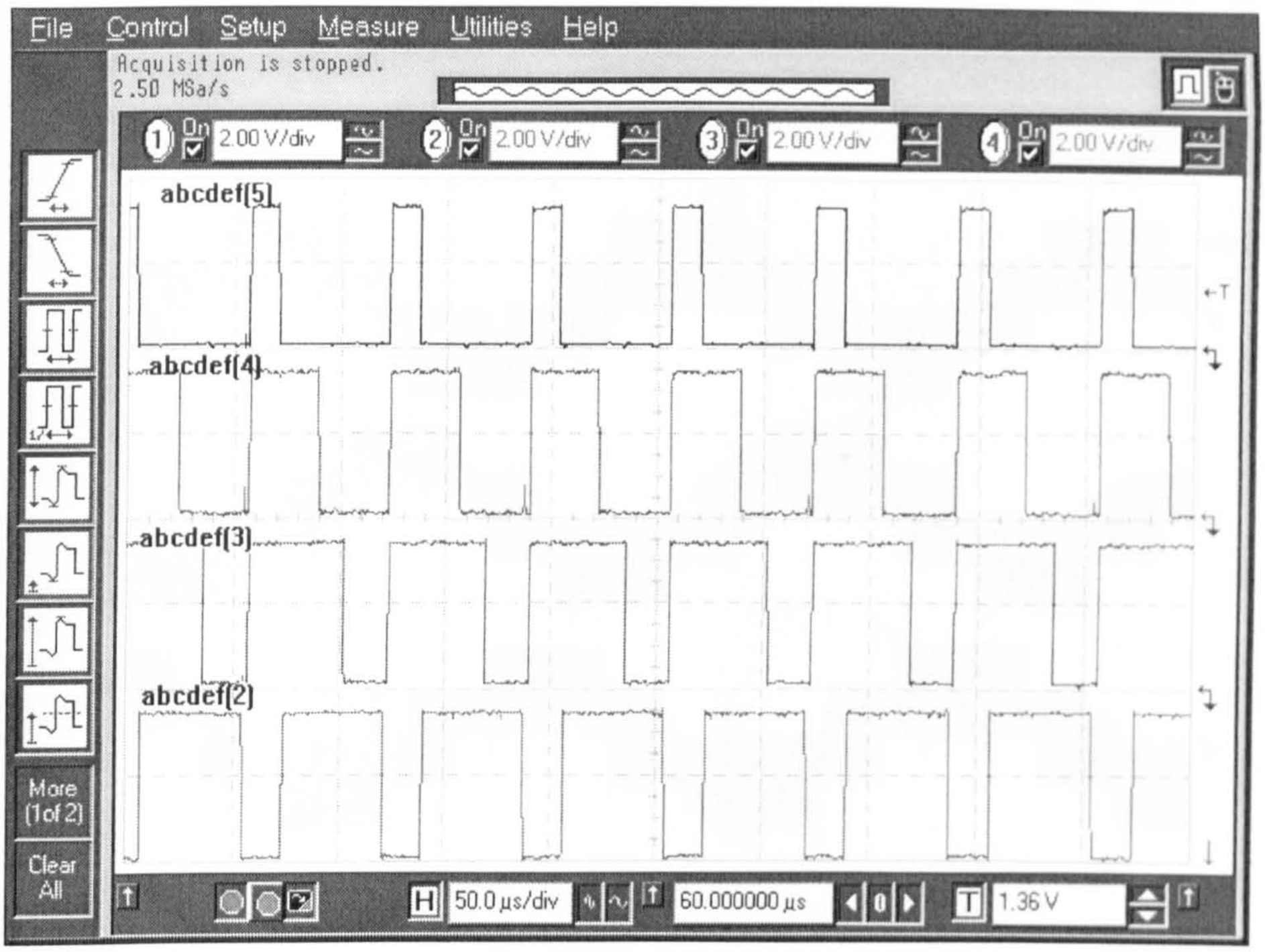

Fig. 7-6 – Four of the FPGA output signals controlling the transistors in the PWM inverter

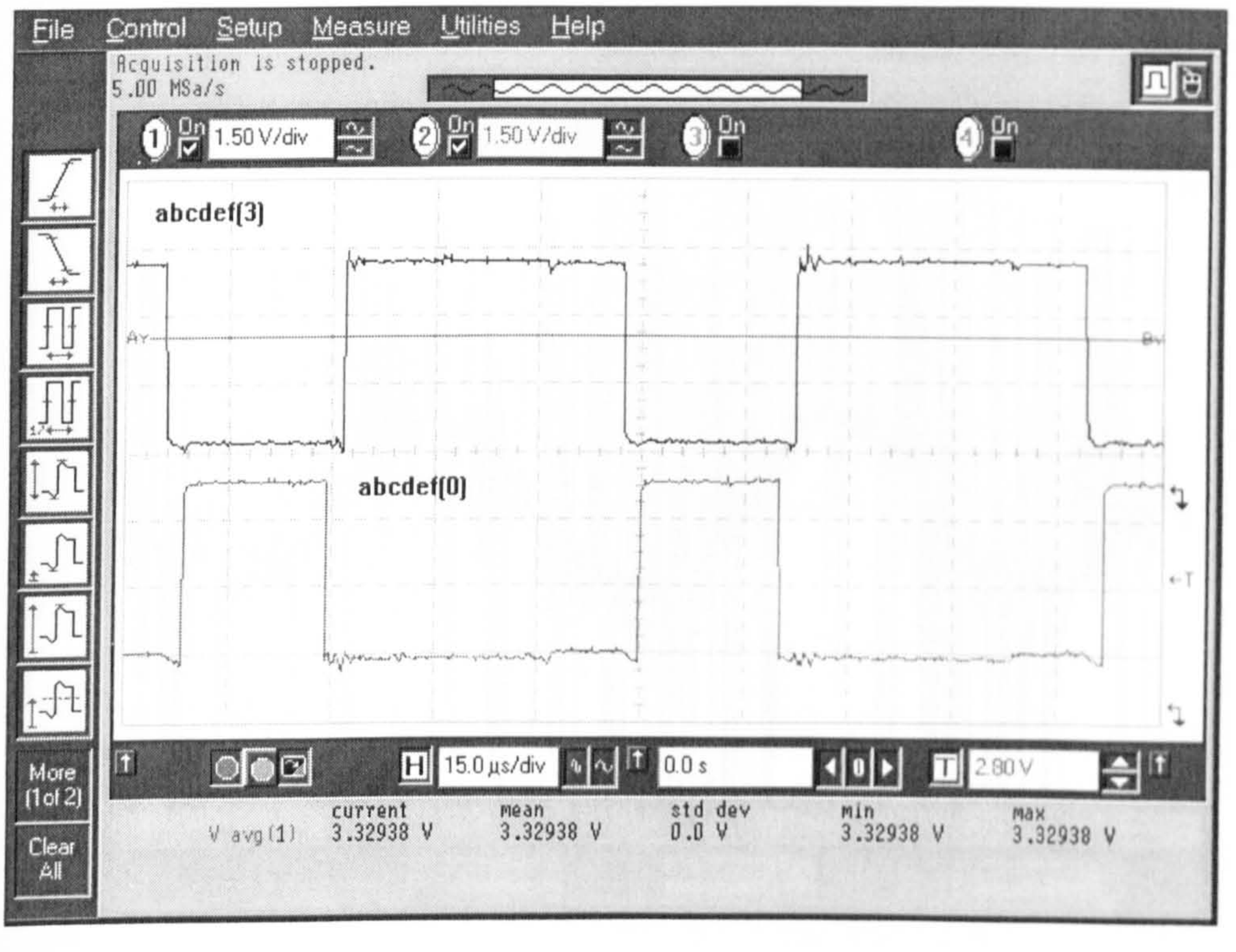

Fig. 7-7 – The switching delays produced by tier2 to avoid the short-circuits in the PWM inverter

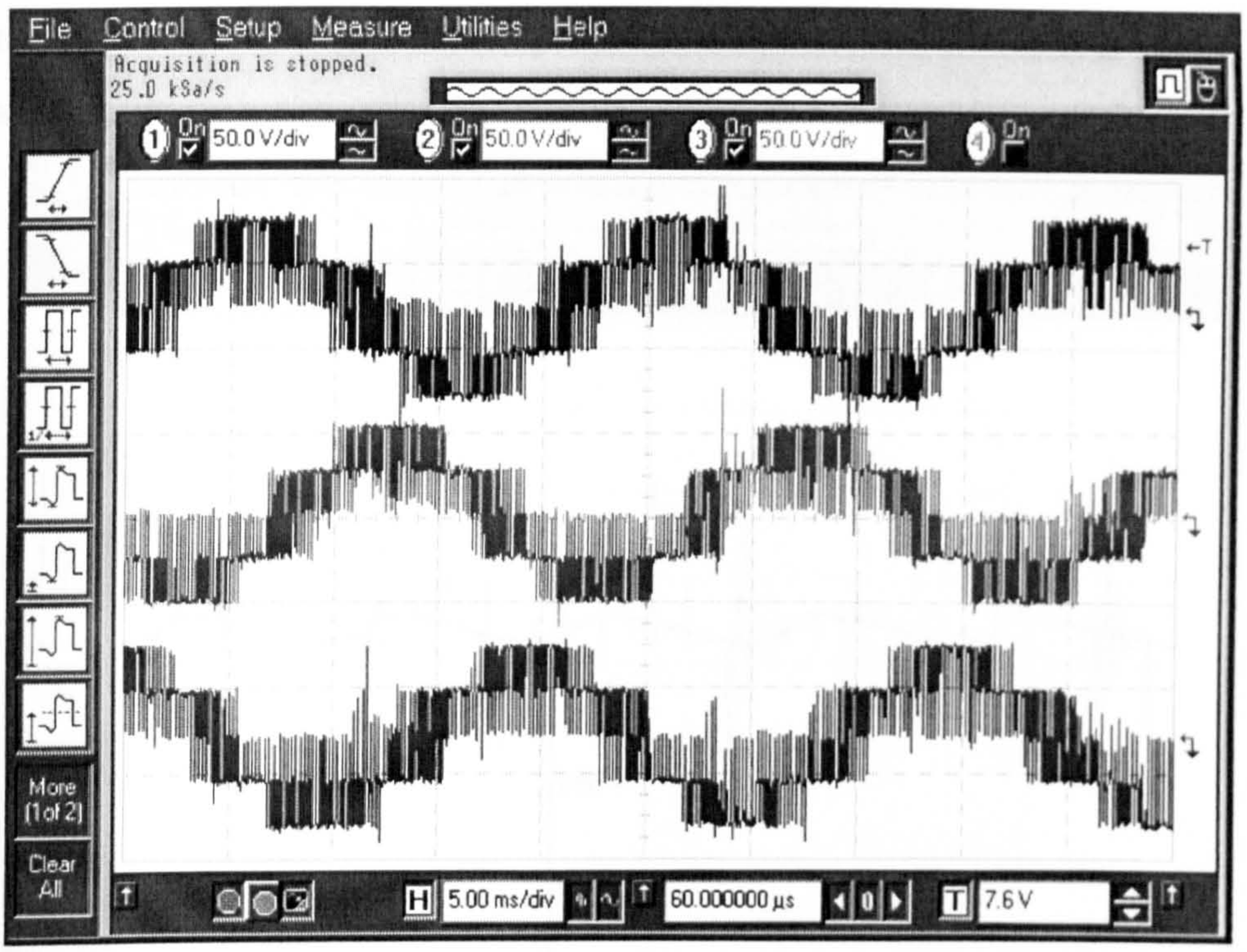

Fig. 7-8 – The PWM voltage generated by the inverter for a reference stator frequency of 50 Hz

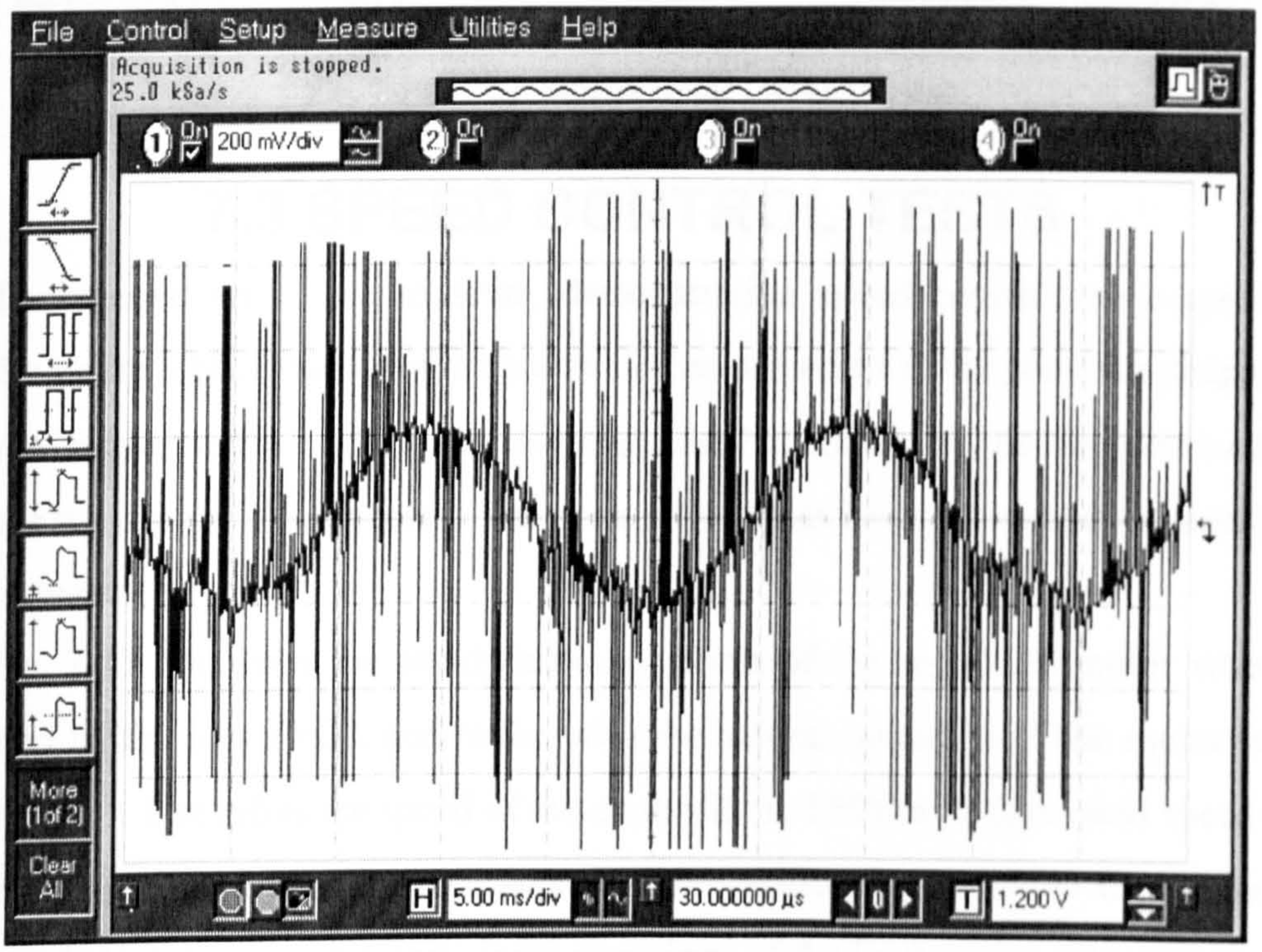

Fig. 7-9 – One of the stator currents (the stator is Y-connected) measured using a Hall transducer

Fig. 7-9 illustrates one of the motor currents corresponding to the operation mode generated by the voltages in Fig. 7-8. The voltage signal in Fig. 7-9 is acquired from one of the Hall transducers and its amplitude of 200 mV corresponds to current amplitude of 0.4 A. Fig. 7-10 presents the DC-link voltage in the same operation conditions and demonstrates that the gamma filter composed of the 6 mH inductor and 470 µF

capacitor illustrated in Fig. 7-2 is capable to maintain the DC voltage level within acceptable limits.

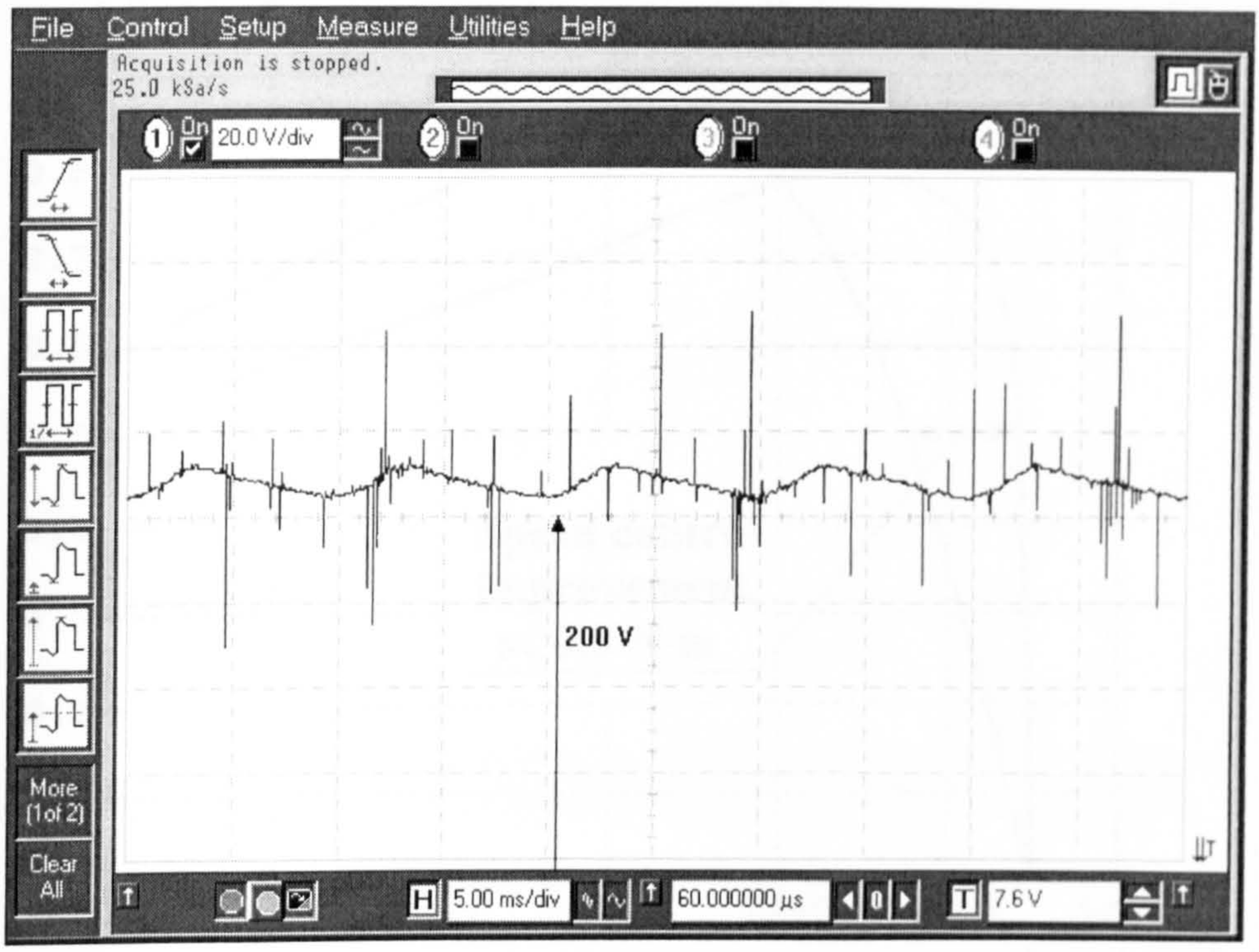

Fig. 7-10 – The voltage ripple on the DC link

7.3 SPEED CONTROL TESTS

The second set of experiments, demonstrating speed control, were carried out using the entire VHDL model of the controller, as presented in the previous chapter. The tests were performed with the stator Δ-connected, but similar results are obtained when the stator winding is Y-connected. The drive system has been tested both in steady-state and in transient operation.

Fig. 7-11 compares the steady-state operation of the induction motor when it is controlled by the new FPGA controller, with the natural operation of the motor without any controller. The reference speed of the controller is 1500 rev/s (the rated speed of the FH90 motor). This figure demonstrates that the controller is capable to maintain the rotor speed almost constant despite large variations of the load torque. The improvement brought by the use of the new controller is given by the higher rigidity of speed-torque characteristic. This improvement can be quantified as the speed increase produced by the FPGA controller for each value of the load torque (Fig. 7-11).

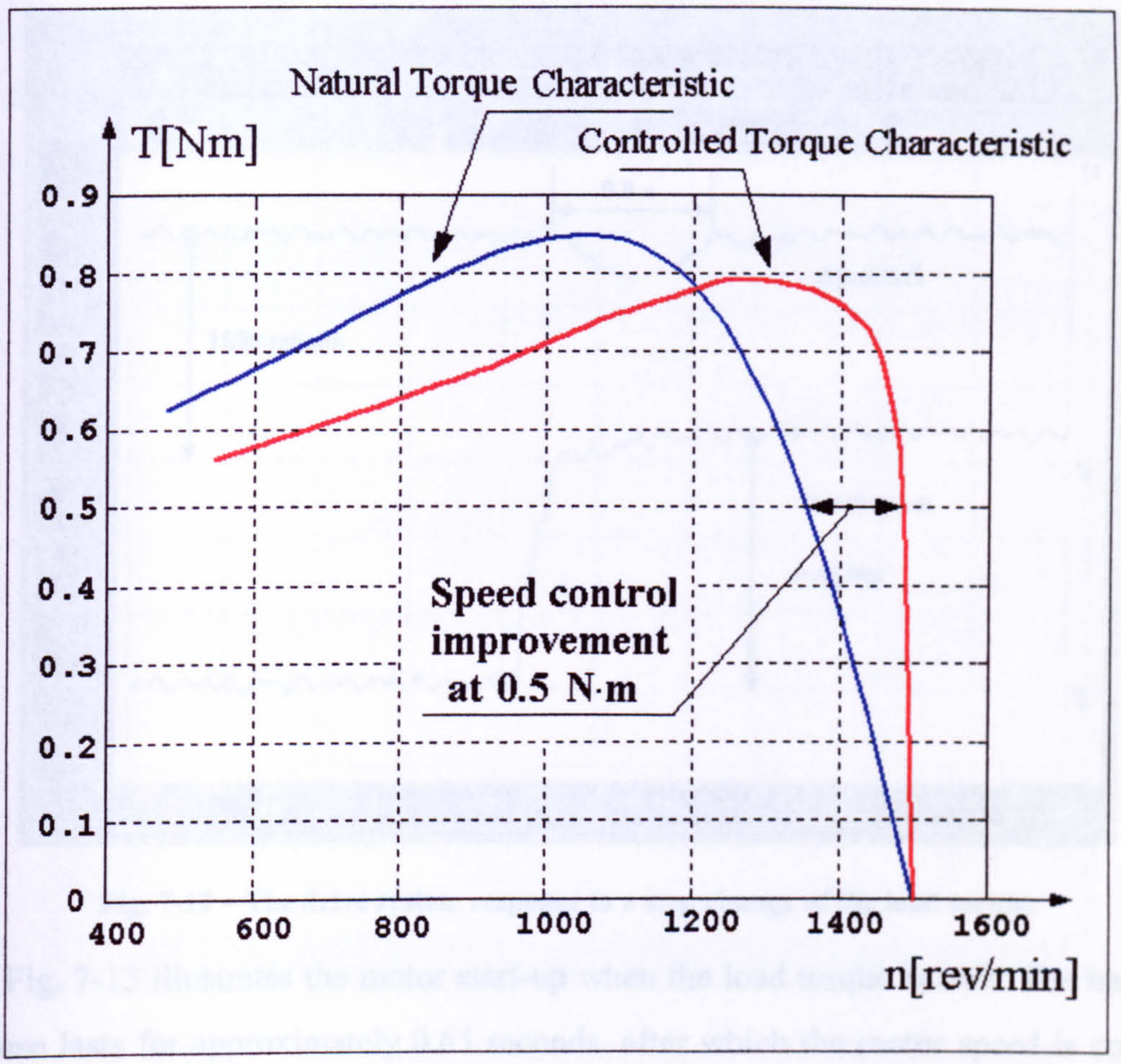

Fig. 7-11 – The static mechanical characteristic of the motor with and without digital controller

Fig. 7-12 presents the motor transient response to a fast variation of the torque. A perfect step variation of the load torque could not be achieved due to the limitations of the testbench. However, Fig. 7-12 shows the capability of the FPGA controller to maintain the speed almost constant while the load torque undergoes significant variations relative to the motor rated power. Thus, the increase of the torque causes a slight slow down of the rotor and this leads to an increase of the motor slip above the imposed value. Therefore, the slip compensation mechanism included in the controller increases the motor current and boosts the active torque reducing the slip frequency to the initial value. The transient process takes approximately 0.8 s.

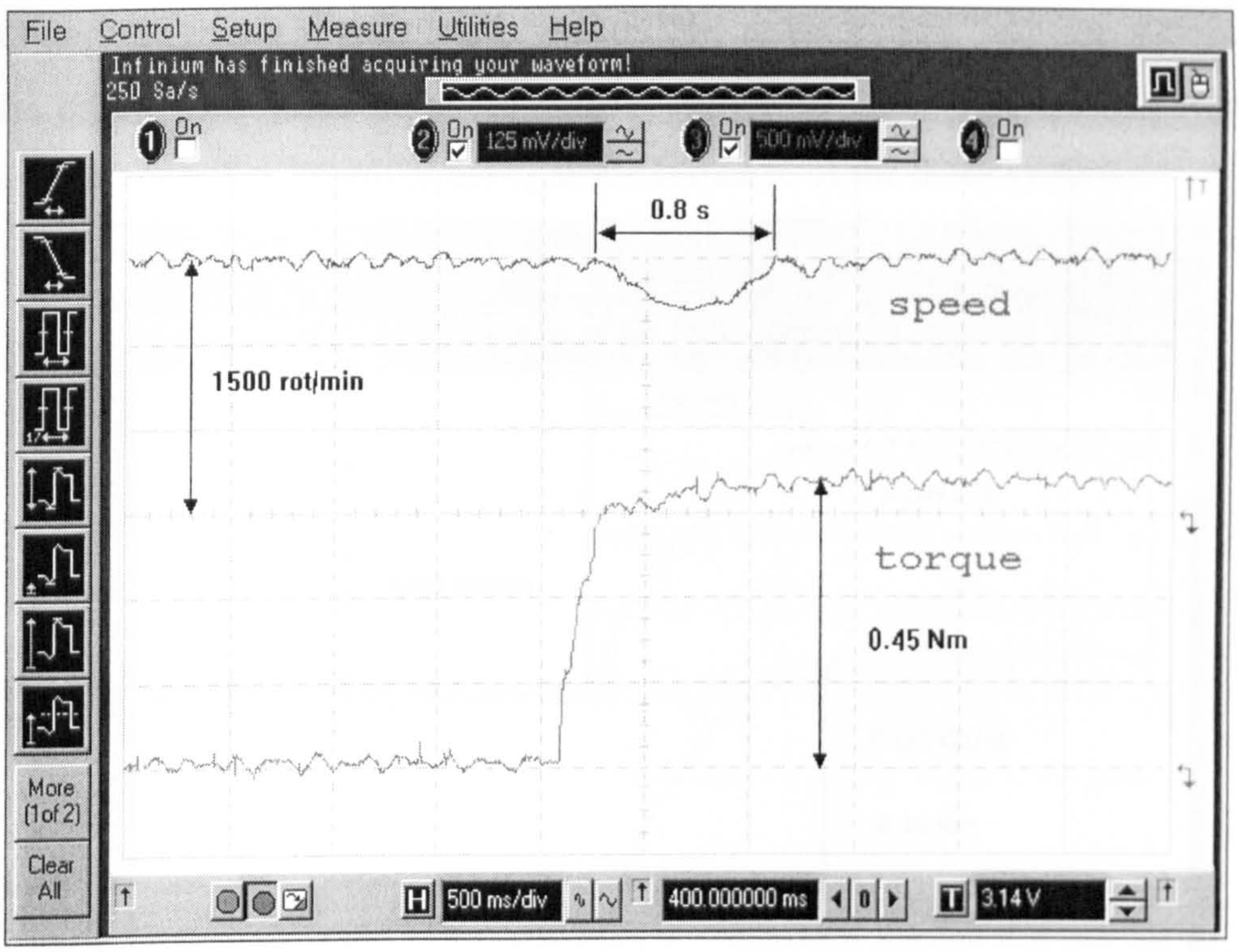

Fig. 7-12 – The drive system response to a step change of the load torque

Fig. 7-13 illustrates the motor start-up when the load torque is null. The transient response lasts for approximately 0.65 seconds, after which the motor speed is constant at 1500 rev/min.

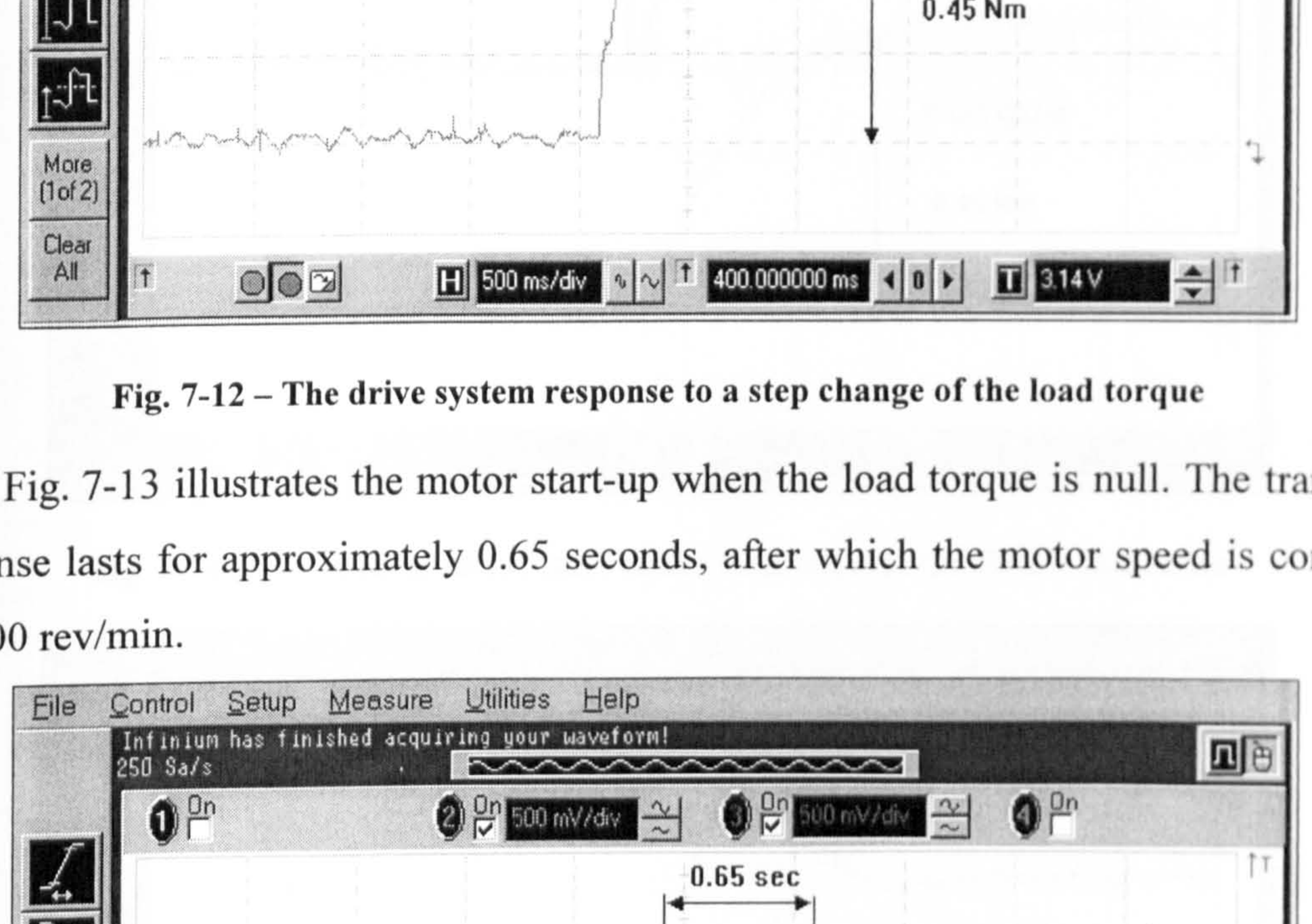

Fig. 7-13 –The motor start without external load torque

The load torque during the transient operation illustrated in Fig. 7-14 and Fig. 7-15 is proportional to the rotor speed. As a result, the motor acceleration is more

difficult and the transient response is slower. The result in Fig. 7-15 can be compared with the motor start when no speed control system is used (Fig. 7-14). Clearly, the FPGA speed controller improves the dynamic response of the drive system.

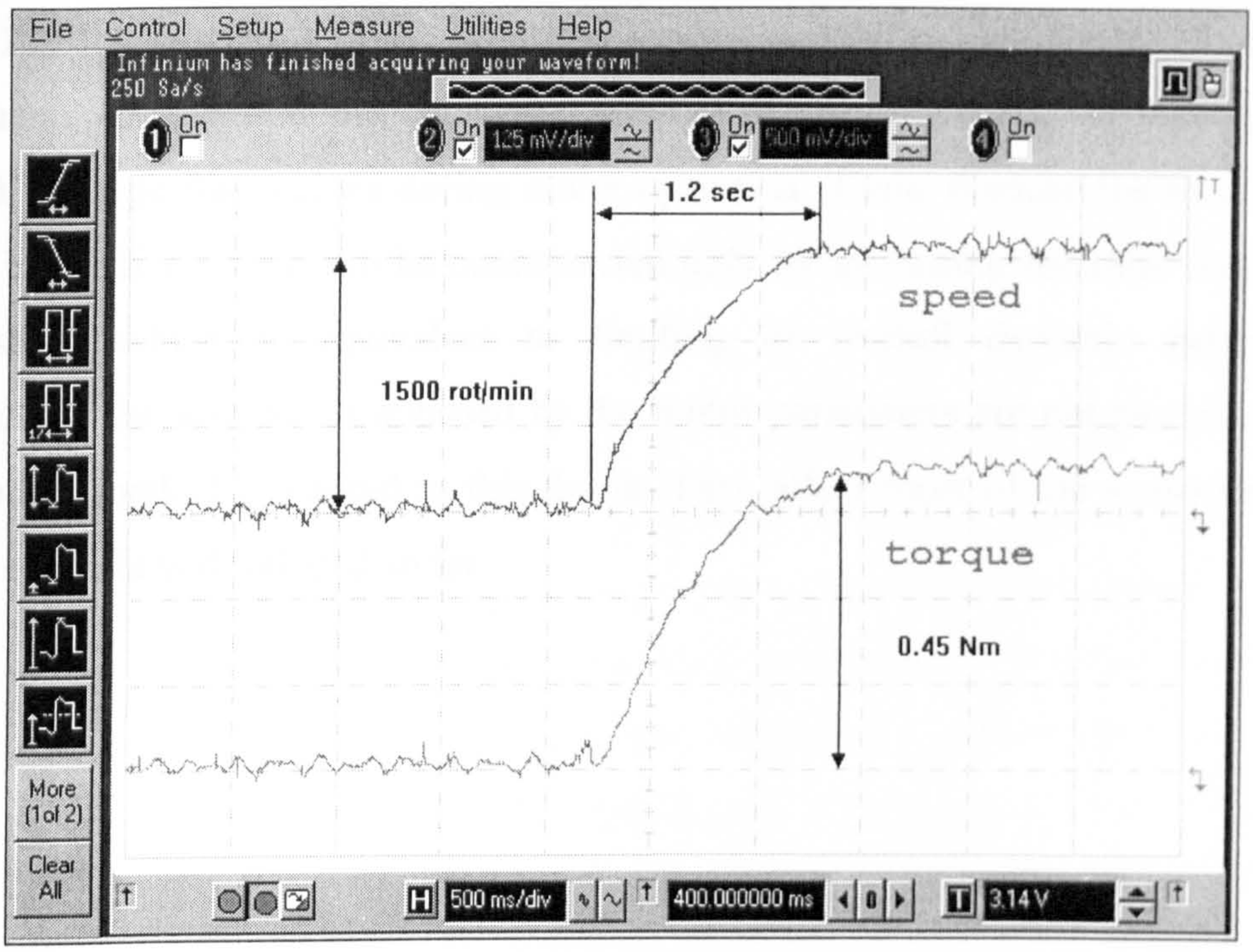

Fig. 7-14 - The natural motor start under load without the FPGA controller

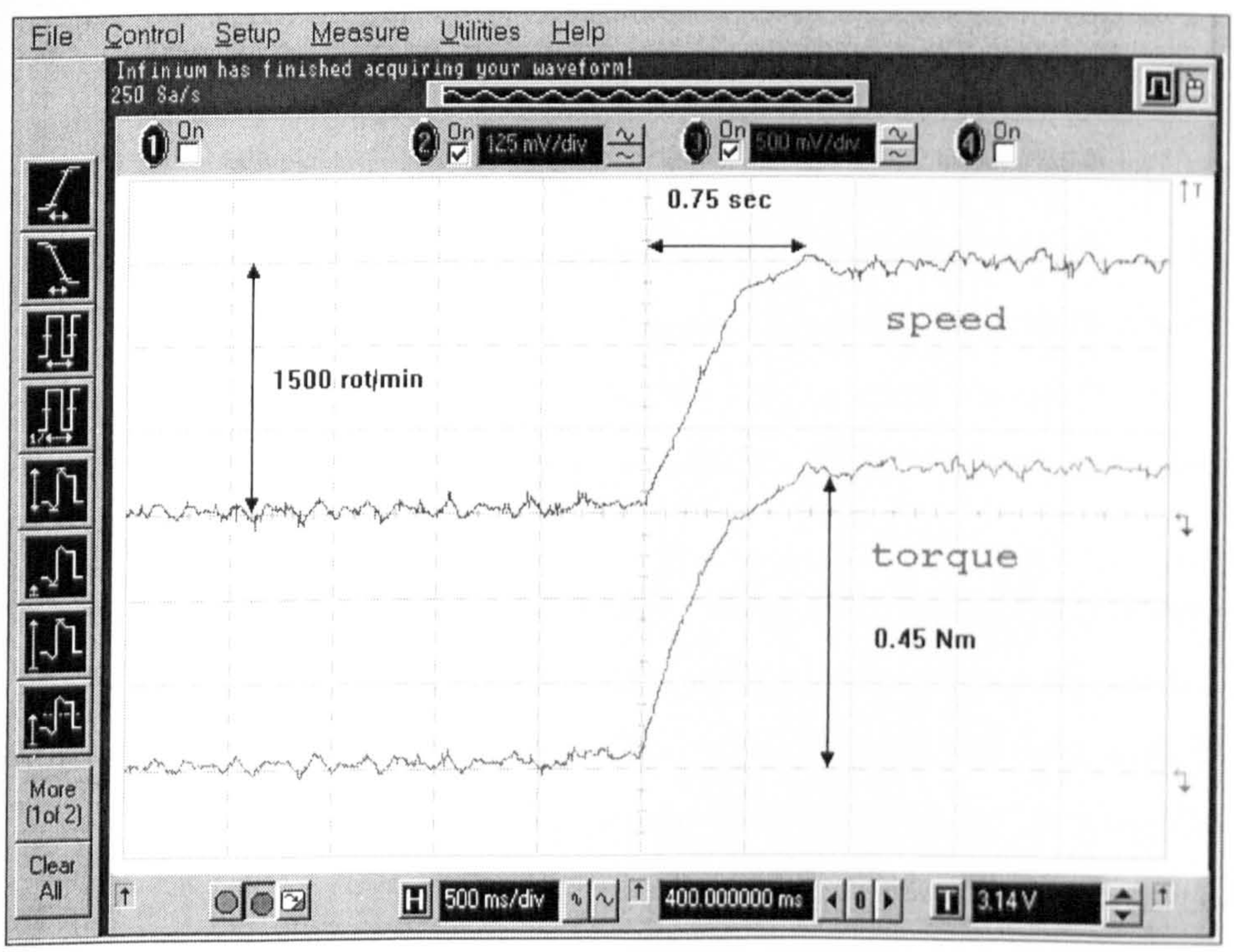

Fig. 7-15 - The motor start under load when controlled by the FPGA device

The improvement is limited by two factors:

- The motor parameters
- The available hardware resources

The parameters of the motor FH90 are not suitable for sensorless control with high response speed because the stator resistance is large (95 Ω) and therefore the resistive voltage component is large. Furthermore, the motor power is small (approximately 0.5 kW) so the amplitude of the internal voltage $\underline{e}$ is smaller than the resistive voltage component. As a result, the calculated equivalent slip angle α_{eqv} is inaccurate and affected by large fluctuations during motor operation. These fluctuations tend to cause system instability, which can be counteracted only by increasing the time constants of the system, which is equivalent to limiting its overall dynamic performance. Nonetheless, the limitations imposed by the motor parameters are not particular to the new control method presented in this thesis. They affect most of the sensorless speed control algorithms developed so far.

8. CONCLUSIONS AND FURTHER WORK

8.1 DISCUSSION AND CONCLUSIONS

An original sensorless speed control strategy for induction motors has successfully been developed, implemented and tested by simulations and practical experiments. The theoretical underpinning is based on an equivalent R-L-e circuit of the induction motor, and it includes a universal current control algorithm applicable to any three-phase power system, in addition to a specific speed control algorithm designed for induction motors. The two algorithms have been devised in a manner that minimises the number of algebraic calculations and transfers part of the control tasks to hardware implemented neural networks. This approach generates a more compact hardware structure than a controller using classical current control methods and field oriented control strategies.

The current control algorithm is an improved predictive control method enhanced by the inclusion of an on-line induction estimation technique, which calculates the inductance in the equivalent R-L-e circuit. The equivalent inductance value is used to determine the optimal switching sequence of the PWM inverter in order to obtain fast current response while limiting the current ripples. The current control algorithm does not necessitate high calculation precision and it is most naturally expressed in geometrical terms, which makes the implementation, using simple neural networks, a feasible solution.

The speed sensorless control method devised uses space vectors expressed in polar co-ordinates instead of the normal rectangular co-ordinates, which simplifies the co-ordinate transformations between the fixed stator reference frame and the mobile reference frames. Moreover, the control algorithm is based on input quantities that are invariant to such transformations: phase-shifts between space vectors (most efficiently calculated by neural networks), vector amplitudes and vector angular speeds. A stator current oriented reference frame is used, which eliminates the need to calculate the rotor flux vector Ψ_r. However, a quasi-field oriented control method is obtained by taking Ψ_r into account in an indirect manner that is, its position in the complex plane is estimated

based on the position of the internal voltage $\underline{e}$, and the relation between the frequency and the amplitude of the stator current is defined in a manner that approximates the behaviour of a classical space vector controller. Therefore, good control quality is obtained with a reduced computational effort.

The work demonstrates the multiple advantages of using hardware implemented neural networks in the motor control process. Thus, the neural approach reduces the complexity of the control system by eliminating the need for large external EPROM-implemented look-up tables, increases the calculation speed of the controller, and enables the rescaling of the controller structure with good control over the performance-complexity ratio. A new implementation strategy has been developed for feed-forward artificial neural networks containing neurones with step activation functions. This strategy transforms the mathematical model of the network into an optimised VHDL description which contains only AND, OR and NOT logic gates. Given the high operation speed of the obtained neural networks and the limited hardware resources, the optimisation aimed to minimise the implementation gate count. The entire process was successfully automated by means of C++ programs.

The XC4010XL FPGA containing the equivalent of only 10,000 gates was used experimentally to prove the validity of the new sensorless control principles. Such a small device is sufficient for simple applications, which do not require high dynamic performance. The industrial applications in this category include drive systems for fans and pumps where fast speed changes are not only unnecessary but they are also pernicious due to the shock waves (stresses) which can be created in the system. The cost of applying control to these applications is decreased by the elimination of the speed sensor, which is made possible by the introduction of the new FPGA controller using hardware implemented neural networks.

High dynamic performance can be obtained with more complex controllers implemented in larger devices from the new FPGA families Spartan and Virtex, which offer more than 100,000 equivalent gates per chip. These devices can integrate complex neural networks alongside high precision classical digital structures that operate with quantities represented by a large number of bits. In this case, the control performance is of the same level of precision as the accurate sensorless vector control algorithms that include Extended Kalman Filters, but without the need for extremely high-speed DSP devices. The optimal size of the FPGA for a given application depends on the required dynamic performance combined with the price limit imposed by the market.

The theoretical investigations, the computer simulations and the experiments proved the validity of the induction motor speed control principles but at the same time highlighted some practical limitations. Thus, as in the case of other sensorless speed control strategies, the controller needs to be tuned to the parameter values of each particular motor. Furthermore, the speed control is more accurate in the case of medium and high power induction motors, as they have small stator resistance and large internal voltage amplitude during normal operation. An advantage is that the current control method does not require any previous information on the load parameters and generates similar results regardless of the motor type and its rated power.

8.2 FURTHER WORK

The research results described open a series of new development directions for the improvement of the control quality of the present motor controller and for the extension of its applicability area.

One of the most important improvements can be the elimination of the possible steady-state speed errors caused by the variation of the rotor resistance as an effect of the temperature rise during the motor operation. This can be achieved by implementing a simple thermal model of the induction motor in order to calculate the rotor temperature based on the initial temperature before the motor start-up and on the variation of the stator currents in time. Heating inside the motor is generated mainly by the stator current $\underline{i}_s$ and by the rotor current $\underline{i}_r$. If the motor slip angular frequency ω_{slp} is maintained constant at the reference value Ω_{slp} then, based on the mathematical analysis presented in chapter 4, the rotor current space-vector can be calculated as:

$$\underline{i}_r^{syn} = \frac{-j\omega_{slp}L_m I_s}{R_r + j\omega_{slp}L_r} = \frac{-j\Omega_{slp}L_m I_s}{R_r + j\Omega_{slp}L_r} \qquad (8\text{-}1)$$

and therefore the amplitude of the rotor current is proportional to the amplitude of the stator current. The heat generated by the motor currents is eliminated through the outer layers of the stator and by means of the air circulated by the motor fan. Therefore, the rotor temperature is the solution of the differential equation

$$T_Q \cdot \frac{d\tau}{dt} + \tau = K_{QI} \cdot I_s^2(t) - F_Q(\omega_r(t)) \qquad (8\text{-}2)$$

where T_Q is the motor thermal time constant, I_s is the stator current amplitude, F_Q is a function of rotor speed and τ is the motor temperature rise compared with the starting moment. Identifying F_Q is a prerequisite for solving the equation. Given the physics

underlying the operation of a typical fan, the function F_Q is depends on the rotor speed and on the rotor speed squared. Consequently, equation (8-2) can be approximated as:

$$T_Q \cdot \frac{d\tau}{dt} + \tau = K_{Q1} \cdot I_s^2(t) - K_{Q2} \cdot \omega_r(t) - K_{Q3} \cdot \omega_r^2(t) \qquad (8\text{-}3)$$

Thus, to increase the steady-state speed control accuracy it is required to identify the parameters T_Q, K_{Q1}, K_{Q2} and K_{Q3} and to implement the equation (8-3) into hardware. This solution increases the hardware implementation of the controller but it is much less complex than an extended Kalman filter so it is the preferable alternative for the practical applications that do not necessitate extremely high speed control accuracy.

A second improvement can be generated by the refinement of the motor control strategy summarised in (4-172). Thus, in a very general case both F_I and F_ω may depend on all the four parameters involved ω_{es}^{ref}, ω_{es}, I_s and β_{eqv}. A thorough mathematical analysis combined with computer simulations can determine mathematical expressions for F_I and F_ω that yield superior dynamic performance compared to the results obtained so far. If F_I and F_ω turn out to have a complicated non-linear dependency on the four input parameters then the neural approach can be extended to the implementation of the two functions. Initially a stepwise approximation needs to be generated and then this approximation can be transformed into a series of Voronoi cells that are readily transformed into a neural network implementable into hardware. Such an implementation strategy needs to be compared to the classical implementation method that uses digital structures, which explicitly perform all the calculations involved. The neural solution is advantageous if either high calculation speed is necessary or if the neural hardware complexity level is lower than the level corresponding to the classical implementation solution.

Another solution to limit the number of calculations involved in the induction motor sensorless speed control process is to adopt a fuzzy control strategy. The control method developed can be readily transformed into a fuzzy strategy by the fuzzification of the two fundamental functions F_I and F_ω. This approach can be particularly fruitful if the exact mathematical expressions of the functions are strongly non-linear and depend on all the four input parameters, thereby making direct mathematical calculations difficult to implement into hardware.

The overall speed control quality can be improved by implementing a control strategy with multiple reference angles α_{eqv}^{ref} between the space vectors $\underline{e}$ and $\underline{i}$ that correspond to multiple reference slip values Ω_{slp}. Large values for Ω_{slp} allow the

generation of larger torque at the same stator current amplitude but in the same time they worsen the speed control capability of the system. Thus, small slip values are preferable when the torque is small, but higher slips need to be adopted if the load torque increases. An improved control system may use a set of three to five different reference slip angles, the decision on which angle to be used at a certain moment being taken based on the stator current amplitude I_S. Thus, whenever the stator current amplitude surpasses an upper limit, a smaller α_{eqv}^{ref} is adopted, which corresponds to a larger Ω_{slp} and has the effect of reducing the necessary I_s. Conversely, when the stator current decreases below a lower limit, a larger value for α_{eqv}^{ref} is adopted and the speed control performance is improved.

The applicability of the current control strategy included in the general control approach is not limited to induction motors. It can be used to control the current in any power system that may include other types of three-phase motors such as synchronous or DC-brushless motors. Specific speed control strategies can be devised for these motors based on the same general R-L-e equivalent circuit as in the case of induction motors. Any of the new speed control strategies can be then used in conjunction with the current control strategy presented in this research work.

The extension of the present control principles to a large range of power systems and the investigation of the advantages generated by the hardware implemented neural approach in each particular case offers a large area of possibilities for future fertile research.

REFERENCES

(In alphabetical order)

[1] *** "Analogue and Mixed Signal Catalogue", Texas Instruments, copyright 1999.

[2] *** "Designing with Powerview/WorkviewPLUS", Viewlogic Systems, Inc., 293 Boston Post Road West, Marlboro Massachusetts 01752-4615, copyright 1993.

[3] *** "Digital Analysis with Powerview/WorkviewPLUS", Viewlogic Systems, Inc., 293 Boston Post Road West, Marlboro Massachusetts 01752-4615, copyright 1993.

[4] *** "FH2/3 MkIV Electrical Machines Teaching System", TecQuipment, Bonsall Street, Long Eaton, Nottingham, NG10 2AN, UK, 1997

[5] *** "Foundation Series Quick Start Guide 1.4", Xilinx Inc, 1997.

[6] *** "Semikron Innovation + Service", Semikron International, Sigmundstr. 200, D-90431, Nürnberg, Germany, copyright 1998.

[7] *** "The Programmable Logic Data Book", Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124, USA, copyright 1999.

[8] *** "TMS320C5x – User's Guide", Texas Instruments Inc. 1992

[9] ***, "IEEE Standard VHDL Language Reference Manual", IEEE standard 1076-1993, 1994.

[10] ***: "Energy Savings with Electric Motors and Drives" Guide provided by Energy Efficiency Enquiries Bureau, Crown copyright 1998

[11] ***: "Intel 80170NX ETANN Data Sheets", February 1991

[12] ***: "Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP", Literature Number: BPRA057, Texas Instruments Europe, July 1997, (www.ti.com).

[13] Alspector, J.; Allen, R.B.; Jayakumar, A.; Zeppenfeld, T.; Meir, R.: "Relaxation Networks for Large Supervised Learning Problems", in "Advances in Neural Information Processing Systems 3", (Lippmann, R.P.; Moody, J.E.; Touretzky, D.S. eds), Morgan Kaufmann 1991, San Mateo, CA, pp. 1015-1021.

[14] Al-Tayie, J.K.; Acarnley, P.P.: "Estimation of Speed, Stator Temperature and

Rotor Temperature in Cage Induction Motor drive Using the Extended Kalman Filter Algorithm" IEE Proceedings on Electric Power Applications vol. 144, no. 5, September 1997, pp. 301-309.

[15] Andreou, A.G.; Boahen, K.A.; Pouliqueen, P.O.; Jenkins, R.E.; Strohbehn, K: "Current-mode subthreshold MOS circuits for analogue VLSI neural systems", IEEE Transactions on Neural Networks, vol. 2, 1991, pp. 205-213.

[16] Armstrong W.W.; Gecsei, J.: "Adaption Algorithms for Binary Tree Networks", IEEE Transactions on System, Man, Cybernetics, no. 9, pp. 276-285, 1979.

[17] Atkinson, D.J.; Hopfensperger, B.; Lakin, R.A.: "Field Oriented Control of a Doubly-Fed Induction Machine using Coupled Microcontrollers", EPE'99 Lausanne (CD).

[18] Aubebart, F.; Girerd, C.; Chapuis, Y.A.; Poure, P.; Braun, F.: "ASIC Implementation of Direct Torque Control for Induction Machine : Functional Validation by Power and Control Simulation" PCIM'98, Power Converter and Intelligent Motion Conference, Nuremberg, May 25-28 1998, pp 251-260.

[19] Ayestaran, H.E.; Prager, R.W.: "The Logical Gates Growing Network", Technical Report CUED/F-INFENG/TR 137, Engineering Dept., Cambridge University, 1993.

[20] Bartos, F. J.: "Sensorless Vector Drives Strive for Recognition", Control Engineering, September 1996.

[21] Bashagha, A.E.; Ibrahim, M.K.: "A New Digit-Serial Divider Architecture" International Journal on Electronics, vol. 75, no. 1, pp. 133-140.

[22] Bashagha, A.E.; Ibrahim, M.K.: "A New High Radix Non-Restoring Divider Architecture" International Journal on Electronics, vol. 79, no. 4, pp. 455-470, 1995.

[23] Bashagha, A.E.; Ibrahim, M.K.: "Design of a Square-Root Architecture: Digit Serial Approach" International Journal on electronics, vol. 76, no. 1, pp. 15-25, 1994

[24] Bashagha, A.E.; Ibrahim, M.K.: "Digit-Serial Squaring Architecture" Journal of Circuits, Systems and Computers, vol. 4, no. 1, pp. 99-108, 1994.

[25] Bashagha, A.E.; Ibrahim, M.K.: "Nonrestoring Radix-2^k Square Rooting Algorithm" Journal of Circuits, Systems and Computers, vol. 6, no. 3 pp. 267-285, 1996

[26] Baum, E. B.: "On the Capabilities of Multilayer Perceptrons", J. Compl., no. 4, 1988, pp. 193-215.

[27] Beierke, S.; Konigbauer, R.; Krause, B.; Altrock, C. V.: "Fuzzy Logic Enhanced Control of AC Motor Using DSP"; Embedded Systems Conference California; 1995, pp 101-106

[28] Beiu, V.: "Entropy, Constructive Neural Learning and VLSI Efficiency" in Proceedings of NEUROTOP'97: Neural Priorities on Data Transmission and EDA, Brasov Romania, 22-30 May 1997, pp. 38-74.

[29] Beiu, V.: "VLSI Complexity of Discrete Neural Networks", Gordon and Breach & Harwood Academics Publishing, 1998.

[30] Beiu, V.; Taylor, J.G.: "Optimal Mapping of Neural Networks onto FPGAs", 'Natural to Artificial Neural Computation', Lecture Notes in CS930, Springer-Verlag, Berlin, June 1995.

[31] Ben-Brahim, L.: "Motor Speed Identification Via Neural Networks" IEEE Industry Applications Magazine, vol. 1, no. 1 January/February 1995, pp. 28-32

[32] Boldea, I.; Nasar, S.A.: "Vector Control of A.C. Drives", CRC Press, Bota Raton, 1992.

[33] Bose, K.B.: "High Performance Control of Induction Motor Drives", IEEE Industrial Electronics Society Newsletter, September 1999, pp. 7-11

[34] Bose, N.K.; Garga, A.K.: "Neural Network Design Using Voronoi Diagrams" IEEE Transactions on Neural Networks vol. 4, no. 5, September 1993 pp. 778-787.

[35] Bowes, S. R.; Mount, M.J.: "Microprocessor Control of PWM Inverters", IEEE Trans. Ind. Applicat., vol. 128, no. 6, pp. 293-305, 1981

[36] Brod, D.M.; Novotny, D.W.: "Current Control of VSI-PWM Inverters", in IEEE-IAS Conf. Rec. 1984, pp. 418-425.

[37] Butcher, J. C.: "The Numerical Analysis of Ordinary Differential Equations - Runge-Kutta and General Linear Methods", John Wiley & Sons, 1987

[38] Cecati, C; Rotondale, N.: "Torque and Speed Regulation of Induction Motors Using the Passivity Theory Approach", IEEE Transactions on Industrial Applications, vol. 46, no.1, February 1999, pp. 119-127.

[39] Chapuis, Y.A.; Poure, P.; Braun, F. "Torque Dynamic Correction of Direct Torque Control for Induction machine Using a DSP", PCIM'98, Power

Converter and Intelligent Motion Conference, Nuremberg, May 25-28 1998, pp. 241-250.

[40] Chapuis, Y.A.; Roye, D.: "Optimization of Square Wawe Transition for Direct Torque Control of Induction Machine" Proceedings of the Intelligent Motion Conference, Nürnberg, 10-12 June 1997.

[41] Churcher, D.; Baxter, D.J.; Hamilton, A.; Murray, A.F.; Reekie, H.M: "Generic Analog Neural Computation – The EPSILON chip", in "Advances in Neural Information Processing Systems", (Hanson, S.J.; Cowan, D.J.; Giles, C.L. eds.), Morgan Kaufmann 1993, San Mateo, CA: pp. 773-780.

[42] Conradi, P.: "Reuse in Electronic Design: From Information Modelling to Intellectual Properties", John Wiley & Son Ltd, 1999.

[43] Cybenko, G.: "Aproximations by Superposition of a Sigmoidal Function", Mathematics of Control, Signal and Systems, vol. 2, pp. 303-314, 1989

[44] DeDoncker, R; Novotny, D.W.: "The Universal Field Oriented Controller", IEEE-IAS Trans., vol. 30, no. 1 January/February 1994, pp. 92-100.

[45] Driankov, D.; Hellendoorn, H.; Reinfrank, M.: "An Introduction to Fuzzy Control" Springer-Verlag, Berlin Heidelberg 1993

[46] Dwyer, R.A.: "High-dimensional Voronoi Diagrams in Linear Expected Time", Discr. Comput. Geom., vol. 6, 1991, pp. 343-367

[47] Eichmann, G.; Caulfield, H.J.: "Optical learning (inference) machine", Applied Optics, no. 24, 1985, pp. 378.

[48] Eldridge, S.E.; Walter, C.D.: "Hardware Implementation of Montgomery's Modular Multiplication Algorithm", IEEE Transactions on Computers vol. 42, pp. 693-9, 1993.

[49] EL-Sharkawi, M.; Neibur, D. (Eds.): "Artificial neural networks applied to power systems", IEEE Power Engineering Society tutorial course, IEEE catalogue number 96 TP 112-0, 1996

[50] Farhat, N.H.; Psaltis, D., Prata, A.; Paek, E.: "Optical Implementation of the Hopfield Model", Applied Optics, no. 24, 1985, pp. 339.

[51] Fodor, D.; Vas, J.; Katona, Z.: "Fuzzy Logic Based Vector Control of AC Motor Using Embedded DSP Controller Board", Proc. of PCIM'98 - Intelligent Motion, pp 235-240

[52] Foussier, P.; Calmon, F.; Carrabina, J.; Fathallah, M.; Grennerat, V.; Jorda, X.; Gontrand, C.; Retif, M.J.; Chante, J.-P.: "Practical Example of Algorithm

Integration for Electrical Drives", EPE'99 Lausanne (CD).

[53] Goslin, G. R.: "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices", Proceedings of the 1995 DSP Technical Program, pp. 595-604.

[54] Goslin, G.; Newgard, B.: "16-Tap, 8-Bit FIR Filter Application Guide", Xilinx Inc., November, 1994.

[55] Gottlieb, I.M.: "Power Supplies, Switching Regulators, Inverters and Converters", 1st ed., Blue Ridge Summit, PA., 1984

[56] Gottlieb, I.M.: "Practical Power-Control Techniques", 1st ed., Indianapolis, IN, USA : H.W. Sams, 1987

[57] Gottlieb, I.M.: "Regulated Power Supplies", 4th Edition, Blue Ridge Summit, PA, 1992

[58] Grzesiak L.; Beliczynski B.: "Simple Neural Cascade Architecture for Estimating of Stator and Rotor Flux", EPE'99 Lausanne (CD)

[59] Habetler, T.G.; "A space vector-based regulator for AC/DC/AC converters", IEEE Trans. on Power Electronics, vol 8, no. 1, pp. 30-36, 1993.

[60] Hammerstrom, D.: "The Conectivity Analysis of Simple Association - or - How Many Connections Do You Need", Proceedings. NIPS'87 (Denver USA), Amer. Inst. Phys., 1987, pp. 338-347.

[61] Harrer, H.; Nossek, J.A.; Stelzl, R.: "An Analog Implementation of Discrete-Time Cellular Neural Networks", IEEE Transactions on Neural Networks, vol 3, no 3, May 1992, pp. 466-476.

[62] Haykin, S. "Neural Networks A Comprehensive Foundation" Macmillan College Publishing Company, Inc. 1994

[63] Heht-Nielsen, R.: "Neurocomputing. Reading", Addison-Wesley Publishing Co. 1990.

[64] Holtz, J.: "Pulsewidth modulation - A Survey", IEEE Trans. Ind. Electron., vol. 39, no. 5, pp. 410-420, 1992.

[65] Holtz, J.: "Speed Estimation and Sensorless Control of AC Drives" in proceedings of IECON'93, IEEE-Industrial Electronics Society 1993, pp. 649-654.

[66] Holtz, J.; Stadtfeld, S.: "A Predictive Controller for the Stator Current Vector of AC Machines Fed from Switched Voltage Source", JIEE IPEC-Tokyo Conf. Rec. 1983, pp. 1665-1675.

[67] Hopcroft, J.E.; Mattson, R.L.: "Synthesis of Minimal Threshold Logic Networks", IEEE Trans. on Electr. Comp., EC-6, pp. 552-560, 1965.

[68] Hornik, K.; Stinchcombe, M.; White, H.: "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, No. 2, 1989, pp. 359-366.

[69] Hu, W.Y.; Zhong, L.; Rahman, M.F; Lim, K.W.: "A Fuzzy Observer for Stator Resistance for Application in Direct Torque Control of Induction Motor Drives", Proc of the Second International Conference on Power Electronics and Drives (PEDS'97), 26-29 May, 1997, Singapore, Vol. 1, pp. 91-96.

[70] Hu, Y.W.; Rahman, M.F., et al: "Direct Torque Control of Induction Motor Using Fuzzy Logic", Canadian Conference on Electrical and Computer Engineering, St. John's, Newfoundland, Canada, May 25-28, 1997, Vol. 2, pp. 767-772.

[71] Hwang, B.; Saif, M.; Jamshidi, M.: "A Neural Network Based Fault Detection and Identification (FDI) for a Pressurized Water Reactor", Proceedings of the 12th IFAC World Congress on Automatic Control, Sydney, Australia, 1993

[72] Hwang, B.C.; Saif, M.; Jamshidi, M.: "Fault Detection and Diagnosis of a Nuclear Power Plant Using Artificial Neural Networks", Journal of Intelligent and Fuzzy Systems, Vol. 3, No. 3, pp. 197-213, 1995

[73] Ibrahim, M.K.; Bashagha, A.E.: "Area-Time Efficient Two's Complement Square Rooting" International Journal on electronics, vol. 86, no. 2, pp.127-140, 1999.

[74] Irwin, G. W.; Warwick, K.; Hunt K. J.: "Neural Network Applications in Control" Institution of Electrical Engineers, London 1995

[75] Javurek, J: "Possibilities of Improving the Method of Direct Control of Asynchronous Machine Torque, Automatizace Journal (Chech Republic) No. 12, 1997, pp. 789-794

[76] Jeong, S.G.; Myung-Ho, W.: "DSP Based Active Power Filter with Predictive Current Control", IEEE Trans. on Industrial Electronics vol. 44, No. 3, June 1997

[77] Jönsson, R: "Natural Field Orientation (NFO) Provides Sensorless Control of AC Induction Servo Motors" PCIM Magazine June 1995, pp. 10-17

[78] Jönsson, R; Leonhard, W.: "Control of an Induction Motor without a Mechanical Sensor, based on the Principle of 'Natural Field Orientation'

(NFO)" International Power Electronics Conference IPEC-Yokohama 1995, pp 101-106.

[79] Kalman, R.E.:"A New Approach to Linear Filtering and Prediction Problems", Trans. ASME (J. Basic Engineering) vol. 82D, no. 1, March 1960, pp. 35-45.

[80] Kalman, R.E.; Bucy, R.S.: "New Results in Linear Filtering and Prediction Theory", Trans. ASME (J. Basic Engineering), vol. 83D, no. 1, March 1961, pp. 95-108.

[81] Kanmachi, T.; Takahashi, I: "Sensor-Less Speed Control of an Induction Motor", IEEE Industry Applications Magazine, vol. 1, no. 1, January/February 1995, pp. 22-27.

[82] Kawamura, A.; Hoft, R.G.: "Instantaneous Feedback Controlled PWM Inverters with Adaptive Hysteresis", IEEE Trans. Ind. Appl. vol. IA-20, pp. 769-775, 1984.

[83] Kazmierkowski, M. P.; Dzieniakowski, M. A.: "Review of Current Regulation Techniques for Three-Phase PWM Inverters", IEEE IECON Conf. Rec., pp. 567-575, 1994

[84] Kelemen A.; Panã, T.: "Simultaneous Speed and Rotor Resistance Estimation for Sensorless Vector-Controlled Induction Motor Drives" Proceedings of the Twenty-Seventh International Inteligent Motion Conference, June 20-22, 1995 Nürnberg Germany, pp.523-530.

[85] Knapp, S. K., Xilinx Corporate Applications Manager: "Using Programmable Logic to Accelerate DSP Functions", Xilinx, Inc. December 1996.

[86] Koc, C.K. Johnson S.: "Multiplication of signed-digit numbers", Electronics Letters, May 1994 pp. 840-841.

[87] Kolar, J.K.; Ertl, H.; Zach F.C.: "Analysis of on- and off-line optimised predictive current controllers for PWM converter systems", IEEE Trans. on Power Electronics, vol. 6, pp. 451-462, July 1991.

[88] Krzeminski, Z.: "An Observer System for Induction Motor without Speed Sensor" Proceedings of the Twenty-Seventh International Inteligent Motion Conference, June 20-22, 1995 Nürnberg Germany, pp. 143-153.

[89] Krzeminski, Z.; Guzinski, J.: "DSP Based Sensorless Control System of the Induction Motor", Proc. of PCIM'98 - Intelligent Motion, pp 137-146.

[90] Kung, S.Y.: "Digital Neural Networks", Prentice Hall 1993.

[91] Lanser, J.A.; Lehmann, T.: "An Analog CMOS Chip Set for Neural Networks with Arbitrary Topologies", IEEE Transactions on Neural Networks, vol. 4, no 3, May 1993, pp. 441-444.

[92] Le-Huy, H.; Dessiant L.A: "An addaptive current control scheme for PWM synchronous motor drives: analysis and simulation"" IEEE Trans. on Power Electronics, vol. 4, pp. 486-495 Oct. 1989.

[93] Leonhard, W.: "Control of electrical drives", 2nd edition Springer, Berlin 1996.

[94] Lin, F.-J.; Shyu, K.-K.; Wai, R.-J.: "DSP-Based Minmax Speed Sensorless Induction Motor Drive With Sliding Mode Model-Following Speed Controller", IEE Proceedings - Electric Power Applications, November 1999, Vol. 146, Issue 6, pp.471

[95] Lindsey, C.S.; Lindblad, Th.: "Review of hardware neural networks: A user's perspective", Proceedings of the 3rd Workshop on Neural Networks: From Biology to High Energy Pyhsics, Isola d'Elba, Italy, September 26-30, 1994.

[96] Lüdtke, I.; Jayne, M.G.: "A new direct torque control strategy", IEE Colloquium on Advances in Control Systems for Electric Drives, 24 May 1995, London, UK, Digest No: 1995 pp. 114-120.

[97] Macq, D.; Verleysen, M.; Jespers, P.; Legat, J.D.: "Analog Implementation of a Kohonen Map with On-Chip Learning", IEEE Transactions on Neural Networks, vol 4, no 3, May 1993, pp. 456-461.

[98] Manes, C.; Parasiliti, F.; Tursini, M.: "Comparative Study of Rotor Flux Estimation in Induction Motors with a Nonlinear Observer and the Extended Kalman Filter", Proceedings of the 20th International Conference on Industrial Electronics Control and Instrumentation IECON - Bologna, vol.3, pp. 2149, 1994.

[99] Manes, C.; Parasiliti, F.; Tursini, M.: "DSP Based Fiels-Oriented Control of Induction Motor with a Non-Linear State Observer" 27th Annual IEEE Power Electronics Specialists Conference, vol 2, pp, 1254, 1996

[100] Massengill, L.W.; Mundie, D.B.: "An Analog Neural Hardware Implementation Using Charge-Injection Multipliers and Neuron-Specific Gain Control", IEEE Transactions on Neural Networks, vol. 3, no. 3, May 1992, pp. 354-362.

[101] Matsuo, T.; Lipo, T.A.: "A Rotor Parameter Identification Scheme for Vector

Controlled Induction Motor Drives", Rec. IEEE-IAS Annual Meeting, pp. 538-545

[102] Mead, C.A.; Ismail, M.: "Analogue VLSI Implementation of Neural Systems", MA: Kluwer, Boston, 1989

[103] Mirchandini, G; Cao, W.: "On Hidden Nodes in Neural Nets", IEEE Transactions on Circuits and Systems No. 36, May 1989, pp. 661-664

[104] Mortara, A.; Vittoz, E.A.: "A Communication Architecture Tailored for Analog VLSI Artificial Neural Networks: Intrinsic Performance and Limitations", IEEE Transactions on Neural Networks, vol. 5, Number 3, May 1994, pp. 459-466.

[105] Murray, A.F.; Del Corso, D.; Tarassenko, L.: "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques", IEEE Transactions on Neural Networks, vol. 2, no. 2, March 1991, pp. 193-204.

[106] Nabae, S.; Ogasawara M.; Akagi, H.: "A New Control Scheme for Current Controlled PWM Inverters", IEEE Trans. Ind. Appl., vol IA-22, no. 4, pp. 697-701, July/August 1986.

[107] Navabi, Z. "VHDL - Analysis and Modeling of Digital Systems", Electrical and Computer Engineering Series, McGraw-Hill International Editors, 1993.

[108] New, B. "A distributed arithmetic approach to designing scaleable DSP chips", EDN, August 17, 1995, pp.107-114.

[109] Novotny, D.W.; Lipo, T.A.: "Vector Control and Dynamics of AC Drives", Oxford Science Publications, Clarendon Press – Oxford 1996.

[110] Perry D.L.: "VHDL" - Second edition, McGraw-Hill Series on Computer Engineering, McGraw-Hill Inc 1994

[111] Profumo, F.; Griva, G.; Tenconi, A.; Abrate, M.; Ferraris, L.: "Stability Analysis of Luenberger Observers for Speed Sensorless High Performance Spindle Drives", EPE'99 Lausanne (CD).

[112] Rajashekara, K.; Kawamura, A.; Matsuse, K.: "Senorless Control of AC Drives", IEEE Press, 1996.

[113] Ramacher, U.; Wesseling, M.: "A Geometrical Approach to Neural Network Design", in Proceedings. IJCNN'89 (Washington USA), IEEE Press, vol 2, pp. 147-153, January 1989.

[114] Säkinger, E.; Boser, B.E.; Jackel, L.D.: "A neurocomputer board based on the ANNA neural network chip", in "Advances in Neural Information Processing

Systems 4", (Moody, J.E.; Hanson, S.J.; Lippmann, R.P. eds.), Morgan Kaufmann 1992, San Mateo, CA, pp. 773-780.

[115] Satieo, S.; Torrey, D.A.: "Fuzzy Logic Control of a Space-Vector PWM Current Regulator for Three-Phase Power Converters", IEEE Transactions on Power Electronics, vol. 13 no. 3, pp. 419-426, 1998

[116] Schwartz, M.; Shaw, L: "Signal Processing - Discrete Spectral Analysis, Detection and Estimation" McGraw-Hill Book Company 1975.

[117] Seepold, R.; Kunzmann, A. (eds.): "Reuse Techniques for Vlsi Design", Kluwer Academic Publishers, 1999.

[118] Smieja, F.J.: "Neural Network Constructive Algorithm: Trading Generalisation for Learning Efficiency?", Circuits, Systems, Signal Processing, vol. 12, no. 2, pp. 331-374, 1993.

[119] Summer, M.; Campbell, J.; Curtis, M.: "A Stator Resistance Estimator for Sensorless Vector Controlled Drives using Artificial Neural Networks", EPE'99 Lausanne (CD)

[120] Tajima, H.; Hori, Y.: "Speed Sensorless Field Orientation Control of the Induction Machine", IEEE Trans. Ind. Appl. vol. 29, 1993, pp. 175-180.

[121] Tan, S.; Vandewalle, J.: "Efficient Algorithm for the Design of Multilayer Feed-Forward Neural Networks", Proceedings IJCNN'92 (Baltimore USA), IEEE Press, vol. 2, 1992, pp. 190-195.

[122] Teske, N.; Asher, G.M.; Bradley, K.J.; Summer, M.: "Sensorless Position Control of Induction Machines using Rotor Saliencies under Load Conditions", EPE'99 Lausanne (CD).

[123] Trynadlowski, A. M.; Legowski, S.: "Minimum-loss vector PWM strategy for three-phase inverters", IEEE Trans. Power Electron., vol. 9 no. 1, pp. 26-34, 1994

[124] Tzou, Y.-Y.; Tsai, M.-F.; Lin Y. F.; Wu, H., "Dual-DSP Fully Digital Control of an Induction Motor", IEEE ISIE Conf. Rec., Warsaw, Poland, pp. 673-678, June 17-20, 1996

[125] Tzou, Y-Y.; Hsu, H-J.: "FPGA Realization of Space-Vector PWM Control IC for Three-Phase PWM Inverters" IEEE Transactions on Power Electronics, vol. 12, No. 6, November 1997, pp. 953-963.

[126] Tzou; Y-Y, Yeh, S-T.; Wu, H.: "DSP-Based Rotor Time Constant Identification and Slip Gain Auto-Tuning for Indirect Vector-Controlled

Induction Drives", IECON Proc. Vol. II, pp. 1228, Taiwan 1996

[127] Vadivel, S.; Bhuvaneswari, G.; Rao, G.S.: "A Unified Approach to Real Time Implementation of DSP Based PWM Waveforms", IEEE Trans. Power Electron. vol. 6, no. 4, pp. 565-575, 1991

[128] Van der Broeck, H. W.; Skudelny, H C.; Stanke, G. V.: "Analysis and Realisation of a Pulsewidth Modulator Based on Voltage Space Vectors" IEEE Transactions on on Industry Applications, vol. 24, No. 1 January/February 1988 pp. 142-150.

[129] Vanlandingham, H.F.: "Introduction to Digital Control Systems", MacMillan Press, NewYork, 1992.

[130] Vas P.; Li J.; Stronach, A.F.: "Artificial Neural Network-Based Control of Electromechanical Systems". in Proceedings of 4th European Conference on Control IEE. Coventry. 1994.

[131] Vas, P.: "Artificial-Intelligence-Based Electrical Machines and Drives. Application of Fuzzy, Neural, Fuzzy-Neural and Genetic Algorithms", Monographs in Electrical and Electronic Engineering, Oxford University Press, 1999.

[132] Vas, P.: "Electrical Machines and Drives, A space-vector theory approach", Monographs in Electrical and Electronic Engineering, Oxford University Press, 1992.

[133] Vas, P.: "Sensorless Vector and Direct Torque Control", Monographs in Electrical and Electronic Engineering, Oxford University Press, 1998.

[134] Vas, P.: "Vector Control of AC Machines", Monographs in Electrical and Electronic Engineering, Oxford University Press, 1990.

[135] Vas, P.; Stronach, A. F.; Neuroth, M.: "DSP-Based Speed-Sensorless High-Performance Torque Controlled Induction Motor Drives", Proc. of PCIM'98 – Intelligent Motion, pp 225-234.

[136] Vas, P.; Stronach, A.F.; Neuroth, M.; Du, T.: "A fuzzy controlled speed-sensorless induction motor drive with flux estimators", IEE EMD, Durham, 1995, pp. 315-319.

[137] Walter, C.D.: "Fast Modular Multiplication using 2-Power Radix", International Journal of Computer Mathematics No. 3, pp. 21-28, 1991.

[138] Walter, C.D.: "Systolic Modular Multiplication, IEEE Transactions on Computers" vol. 42, pp. 376-378, 1993.

[139] White, D. A.; Sofge D. A.: "Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches" Multiscience Press, Inc. 1992.

[140] Zurada, J.M: "Introduction to Artificial Neural Systems", West Publishing Company, 1992.

LIST OF PUBLICATIONS

I. Dinu A.; Cirstea M. N.; McCormick M; Ometto A.; Rotondale N.: "New Approach for PWM Inverter Using a State Space Observer", in The Proceedings of Symposium on Power Electronics, Electrical Drives, Advanced Machines, Power Quality (SPEEDAM' 98), **Sorrento** June 3-5 1998 Italy pp. A2-31 - A2-36.

II. Dinu A.; Cirstea M.N.; McCormick M: "Virtual Prototyping of a Digital Neural Current Controller" in Ninth International Workshop on Rapid System Prototyping June 3-5 1998, **Leuven** Belgium pp. 176-180.

III. Dinu A.; Cirstea M.N.; McCormick M: "An Adaptive Control Strategy for Electric Drives" in the proceedings of (PCIM'98) **Nürnberg**, May 28-28 1998, pp. 101-107.

IV. Dinu A.; Cirstea M.N.; McCormick M; Ometto A.; Rotondalle N. "Neural ASIC Controller for PWM Power Systems" in Eleventh Annual IEEE International ASIC Conference "ULSI - Making It Real" - **Rochester** September 13-16 1998", pp. 29-33.

V. Dinu. A.; Cirstea M. N.; McCormick M.: "A Novel Neural PWM Controller", in proceedings of the IEE International Conference on Simulation (SIMULATION'98), York, UK, September 30 - October 2 1998, pp 375-379.

VI. Dinu A.; Cirstea M.N.; McCormick M; Ometto Antonio; Rotondalle Nicola: "Load Independent Current Control Strategy for PWM Inverters" in the proceedings of UKACC International Conference on Control (CONTROL'98) - September 1-4 1998 **Swansea** UK pp. 1118-1122.

VII. Dinu A.; Cirstea M.N.; McCormick M; Ometto A.; Rotondalle N.: "Neural Network for Control of PWM Inverters" in the proceedings of Power Electronics and Motion Control (PEMC'98), **Prague**, September 8 1998, CDROM.

VIII. Dinu, A.; Cirstea, M.N.; McCormick, M.; Ometto, A.; Rotondale, N.: "Sensorless Induction Motor Control Strategy Optimised for FPGA Hardware Implementation", Proc. of Int. Conf. on Optimization of Electric & Electronic Equipment (OPTIM-IEE), Brasov, Romania, May 2000, PP. 625-630.

IX. Dinu, A.; Cirstea, M.N.; McCormick, M.; Haydock, L.; Al-Khayat, N.: "Neural Current Controller for Induction Motor Applications", Proc. of Int. Conf. on

Optimization of Electric & Electronic Equipment (OPTIM-IEE, IEEE), Brasov, Romania, May 2000, PP. 665-670.

Papers Accepted for Publication:

- Aounis A.; Cirstea M.N.; McCormick M.; Dinu, A.: "Vector Controlled Induction Motor Drive Modelling Using VHDL", Proc. of IEE Int. Conf. on Computer Aided Control Systems Design (CACSD 2000), Salford, UK, 11-14 Sept. 2000.
- Cirstea M.N.; Aounis A.; Dinu, A.; McCormick M.: " VHDL Approach to Induction Motor Modelling", Proc. of the IEE CONTROL 2000, Cambridge, UK, 4-7 Sept. 2000.

Appendix A
UNIVERSAL C++ PROGRAMS FOR NEURAL NETWORK HARDWARE IMPLEMENTATION

Appendix A.1 - CONV_NET.CPP

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <memmanag.h>

#define MaximumDepth 100
#define LengthInputTab 2000
#define AND 1
#define ANY 0

ofstream output_file;
int *out_layer, *index, *node, *inverter, *used;
//Index specifies the initial position of the weights in the matrix
row
//before they were rearranged according to their descending values.
//Node is a vector which stores the node numbers corresponding to the
//weights. The node number is not generally speaking corelated with
the
//input numbers because there are inversor gates and because some
neurones
//are located in other layers than the first.
double *w; //The weights vector for one neurone
int no_w; //The number of weights per neurone
int current_node,output_node,no_gates=0,max_depth,depth=0;
int no_max_inputs;

void add_gate(int ind_init, double threshold, char and_gate);

void arrange(void)
{
  int i,i_max,i_aux,j;
  double max,max_aux;
  for(i=0;i<no_w;i++)
  {
    max=w[i];
    i_max=i;
  for(j=i+1;j<no_w;j++)
    if(max<w[j])
    {
    max=w[j];
    i_max=j;
    }
if(i_max != i)
{
  max_aux=w[i_max];
  w[i_max]=w[i];
  w[i]=max_aux;
  i_aux=index[i_max];
  index[i_max]=index[i];
  index[i]=i_aux;
```

```
    }
  }
}

int add_inverter(int i)
{
  no_gates++;
  ++current_node;
  output_file<<".NOT ";
  output_file<<node[index[i]]<<" "<<current_node<<"\n";
  if(!output_file.good())
  {
    cout<<"\n\aError on file writing";
    exit(1);
  }
  return current_node;
}

double*   measure_matrix(int&   no_neu,int&   no_values,   istream&
input_file)
{
  double temp;
  char buffer[2];
  no_neu=no_values=0;
  while(input_file.read(buffer,1),!input_file.eof())
  {
    if(buffer[0]=='\n')
    no_neu++;
  }
  input_file.clear();
  input_file.seekg(0,ios::beg);
  do
  {
    input_file>>temp;
    if(!input_file.eof())
      no_values++;
  }
  while(!input_file.eof());
  if(no_neu==0)
    no_neu++;
  if((no_values/no_neu)*no_neu == no_values)
    no_values=no_values/no_neu;
  else
    if((no_values/(no_neu+1)) == no_values)
    {
      no_values=no_values/(no_neu+1);
      no_neu++;
    }
    else
    {
      cout<<"\n\aError in input file!";
      exit(1);
    }
  input_file.clear();
  input_file.seekg(0,ios::beg);
  w=alloc_double(no_values);
  return w;
}

void read_line_matrix(istream& input_file)
```

```
{
  int i;
  for(i=0;i<no_w+1;i++)
    input_file>>w[i];
  if(!input_file.good())
  {
    cout<<"\n\aError when reading the input file";
    exit(1);
  }
}

int convert_neuron(void)
{
  int i;
  double threshold;
  inverter=alloc_int(no_w); //Shows if the corresponding weight was
negative
                            //thereby requiring an inversor gate
  used=alloc_int(no_w);     //Shows if the node has been already used
in the
                            //past so that the inverter has already been put
  index=alloc_int(no_w+1);

  for(i=0;i<no_w;i++)
  {
    index[i]=i;
    inverter[i]=0;
    used[i]=0;
  }
  for(i=0;i<no_w;i++)
  {
    if(w[i]<0)
    {
      w[i]=-w[i];
      inverter[index[i]]=1;
    }
    w[no_w]-=w[i];
    w[i]=2*w[i];
  }
  threshold=-w[no_w];
  if(threshold<=0)
  {
    output_node=-1; //Output value is constantly 1
    cout<<"\n\aWarning: The output of a neurone is constantly 1";
  }
  else
  {
    arrange();
    add_gate(0,threshold,ANY);
  }
  delete index;
  delete used;
  delete inverter;
  return output_node;
}

int port_number(int i)
{
  int rez;
  if (inverter[index[i]]==0)
```

```
{
  rez=node[index[i]];
  if(depth>max_depth)
    max_depth=depth;
}
else
{
  if(depth+1>max_depth)
    max_depth=depth+1;
  if (used[index[i]])
    rez=used[index[i]];
  else
  {
    used[index[i]]=add_inverter(i);
    rez=used[index[i]];
  }
}
return rez;
}

int det_num_internal_gate_layers(int no_inp)
{
  int no_inp_top,no_layers=1;
  if(no_inp<=no_max_inputs)
    return 1;
  else
  while(no_inp>no_max_inputs)
  {
    no_inp_top=no_inp/no_max_inputs;
    if(no_inp>no_inp_top*no_max_inputs)
      no_inp_top++;
    no_inp=no_inp_top;
    no_layers++;
  }
  return no_layers;
}


int cursor=0;
int input[LengthInputTab];

void write_gate(char *name,int no_inputs, int local_cursor)
{
  int i,no_inp_top_gate,no_last_inputs;
  if(no_inputs<=no_max_inputs)
  {
    output_file<<name<<no_inputs;
  for(i=0;i<no_inputs;i++)
    output_file<<" "<<input[local_cursor+i];
  current_node++;
  output_file<<" "<<current_node<<"\n";
  output_node=current_node;
  no_gates++;
  if(!output_file.good())
  {
    cout<<"\n\aError on file writing";
    exit(1);
  }
}
else
```

```
{
  no_inp_top_gate=no_inputs/no_max_inputs;
  if(no_inputs>no_inp_top_gate*no_max_inputs)
    no_inp_top_gate++;
  if(cursor+no_inputs+no_inp_top_gate>=LengthInputTab)
  {
    cout<<"\n\aError: Input table is full";
    exit(1);
  }
  for(i=0;i<no_inp_top_gate-1;i++)
  {
    local_cursor=cursor+i*no_max_inputs;
    write_gate(name,no_max_inputs,local_cursor);
    input[cursor+no_inputs+i]=output_node;
  }
  local_cursor=cursor+(no_inp_top_gate-1)*no_max_inputs;
  no_last_inputs=no_inputs-(no_inp_top_gate-1)*no_max_inputs;
  if(no_last_inputs>1)    //It is possible to have only one remaining
input
  {
    write_gate(name,no_last_inputs,local_cursor);
    input[cursor+no_inputs+no_inp_top_gate-1]=output_node;
  }
  else
    input[cursor+no_inputs+no_inp_top_gate-
1]=input[cursor+no_inputs-1];
  for(i=0;i<no_inp_top_gate;i++)
    input[cursor+i]=input[cursor+no_inputs+i];
  write_gate(name,no_inp_top_gate,cursor);
 }
}

double sum;
int j;

void add_gate(int ind_init, double threshold, char and_gate)
{
  int i,no_inputs,no_big_weights,ind_for_AND;
  //These are local variables because they need to be preserved during
the
  //recursive calls of the function
  if(threshold<=0)
  {
    cout<<"\n\aError: The output of a subneurone is constantly 1";
    exit(1);
  }
  if(!and_gate)
{
  sum=0;
  no_inputs=0;
  no_big_weights=0;
  ind_for_AND=ind_init;
  for(i=ind_init; i<no_w;i++)
    if(w[i]>=threshold)
    {
    no_inputs++;
    no_big_weights++;
    ind_for_AND=i+1;
    }
  for(i=ind_for_AND;i<no_w;i++)
```

```
    {
      sum=0;
      for(j=i;j<no_w;j++)
      sum+=w[j];                  //The sum will be the result of several
cumulated
      if(sum>=threshold)     //inputs anyway because these are not big
weights.
      no_inputs++;
    }
  }
  if((no_inputs>1) && (!and_gate))
  {
    depth=depth+det_num_internal_gate_layers(no_inputs);          //'-1'
because there is
                                                        //one gate anyway
    if(depth>MaximumDepth)
    {
       cout<<"\n\aError: Too many recursive calls!";
       exit(1);
    }
    for(i=0;i<no_inputs;i++)
      if(i>=no_big_weights)
      {
      cursor+=no_inputs;
      if(cursor>=LengthInputTab)
      {
         cout<<"\n\aError:  Input  table  is  full.  Enlarge  the  input
table";
        exit(1);
      }
      add_gate(ind_init+i,threshold,AND);
      cursor-=no_inputs;                          //necessarily be an AND
gate
      input[cursor+i]=output_node;
      }
      else
      input[cursor+i]=port_number(ind_init+i);
      write_gate(".OR",no_inputs,cursor); //Here the gate is actually
written
      depth=depth-det_num_internal_gate_layers(no_inputs);
  }
  else if((no_inputs==1) && (no_big_weights==1) && (!and_gate))
    output_node=port_number(ind_init);                  //It  is  just  a
straightforward
                                        //input-output conection
  else if(((no_inputs==1) && (no_big_weights==0)) || and_gate)
                                          //An AND gate will be used
  {
    no_inputs=1;    //When is just a simple AND gate, it coresponds to
a
                //single combination of inputs. Variable 'no_inputs' is
                 //used for economy of space in the stack. The first
input
                //is compulsory to be used which is why no_inputs=1.
    sum=0;
    for(i=ind_init;i<no_w;i++)
      sum+=w[i];
    for(i=ind_init+1;i<no_w;i++)
      if(sum-w[i]<threshold)
      no_inputs++;
```

```
      else
      break;
    sum=0;
    for(i=ind_init;i<ind_init+no_inputs;i++)
      sum+=w[i];
    if(threshold-sum>0)
      no_inputs++;  //A further subneurone is required
    if(no_inputs<2)
    {
      cout<<"\n\aError in algorithm! An AND gate has less than 2
inputs!";
      exit(1);
    }
    depth+=det_num_internal_gate_layers(no_inputs);   //'-1'   because
there is
                                                      //one gate anyway
    if(depth>MaximumDepth)
    {
       cout<<"\n\aError: Too many recursive calls!";
       exit(1);
    }
    for(i=0;i<no_inputs;i++)
    {
      if((i<no_inputs-1) || (threshold-sum<=0))
      input[cursor+i]=port_number(i+ind_init);
      if((i==no_inputs-1) && (threshold-sum>0))
      {
      cursor+=no_inputs;                    //The suplementary subneurone is
added
      if(cursor>=LengthInputTab)
        {
          cout<<"\n\aError:  Input  table  is  full.  Enlarge  the  input
table";
          exit(1);
        }
      add_gate(ind_init+no_inputs-1,threshold-sum,ANY);
      cursor-=no_inputs;
      input[cursor+i]=output_node;
      }
    }
    write_gate(".AND",no_inputs,cursor);
    depth-=det_num_internal_gate_layers(no_inputs);
  }
  else if(no_inputs==0)
  {
    output_node=0;  //Output value is constantly 0
    cout<<"\n\aWarning: The output of a neurone is constantly 0";
  }
  else
  {
    cout<<"\n\aError in conversion algorithm";
    exit(1);
  }
}

void main(int no_par, char** par)
{
  ifstream input_file;
  int no_neu=0, no_values_per_line, previous_no_neu,gate_layers=0;
  int i,no_file;
```

```
if(no_par<4)
{
  cout<<"\nToo few parameters";
  exit(1);
}
no_max_inputs=atoi(par[no_par-1]);
output_file.open(par[no_par-2],ios::out);
if(!output_file.good())
{
  cout<<"\n\aError: The output file cannot be opened!";
  exit(1);
}
cout<<"\n----- Start conversion -----";
for(no_file=1;no_file<no_par-2;no_file++)
{
  input_file.open(par[no_file],ios::in);
  if(!input_file.good())
  {
    cout<<"\n\aError: The input file cannot be opened!";
    exit(1);
  }
  cout<<"\nProcessing file "<<no_file;
  previous_no_neu=no_neu;
  w=measure_matrix(no_neu,no_values_per_line, input_file);
  no_w=no_values_per_line-1;
  if((no_file>1)&&(previous_no_neu != no_values_per_line-1))
  {
    cout<<"\n\aError: Wrong number of neurones in layer"<<no_file;
    exit(1);
  }
  out_layer=alloc_int(no_neu);
  if(no_file==1)
  {
    node=alloc_int(no_w);
    for(i=0;i<no_w;i++)
    {
    node[i]=i+1;
    output_file<<".INPUT "<<node[i]<<"\n";
    if(!output_file.good())
    {
      cout<<"\n\aError on file writing";
      exit(1);
    }
    }
    current_node=no_w;
}
max_depth=0;
    for(i=0;i<no_neu;i++)
    {
      read_line_matrix(input_file);
      out_layer[i]=convert_neuron();
    }
    gate_layers+=max_depth;
    if(no_file==no_par-3)       //It was the last input file so the
output
                          //ports must be written
      for(i=0;i<no_neu;i++)
      {
      output_file<<".OUTPUT "<<out_layer[i]<<"\n";
      if(!output_file.good())
```

```
      {
        cout<<"\n\aError on file writing";
        exit(1);
      }
      }
    delete node;
    if(no_file<no_par-3) //File no 'no_par-2' is the output file so
this is
    {                        //not the last input file
      node=alloc_int(no_neu);
      for(i=0;i<no_neu;i++)
      {
      node[i]=out_layer[i];  //The outputs of the previous layer are
                             //the inputs for the next one
        if(node[i]<=0)
        {
          cout<<"\n\aWarning: The output of the hidden neuron "<<(i+1);
          cout<<" in layer "<<no_file<<" is constant!";
        }
        }
    }
    input_file.close();
    delete w;
    delete out_layer;
  }
  input_file.close();
  cout<<"\nThe output file contains "<<no_gates<<" logic gates on ";
  cout<<gate_layers<<" gate layers\n";
}
```

Appendix A.2 - OPTIM.CPP

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <memmanag.h>

#define NOT 3
#define AND 1
#define OR  4
#define INPUT 2
#define OUTPUT 5
#define NO_WORDS 5
#define CANCELLED -2
//'0' is already defined in CONV_NET!.CPP as 'ground' and '-1' as
'Vcc'.

char* words[5]={".AND",".INPUT",".NOT",".OR",".OUTPUT"};
int length[5]={4,6,4,3,7};

struct gate_string
{
  char name[8];
  int no_gates;
  int cursor;
  int* gate_nodes;
};

gate_string *and_s, *or_s;
int *input_s, *output_s, *not_s;
```

```
int max_input=0,max_output=0,max_and=0,max_or=0,max_not=0;
int cursor_input=0,cursor_output=0,cursor_not=0;
int no_init_gates,no_fin_gates;

gate_string* alloc_gate_string(int no_gate_strings)
{
  gate_string *pointer;
  if(no_gate_strings>0)
  {
    if(!(pointer=new gate_string[no_gate_strings]))
    {
      cout<<alloc_err;
      exit(1);
    }
    return pointer;
  }
  else
    return NULL;
}

void init_structures(void)
{
  int i;
  for(i=0;i<max_and-1;i++)
  {
    and_s[i].no_gates=0;
    and_s[i].cursor=0;
    and_s[i].name[0]=0;
  }
  for(i=0;i<max_or-1;i++)
  {
    or_s[i].no_gates=0;
    or_s[i].cursor=0;
    or_s[i].name[0]=0;
  }
}

int check_word(char* buffer)
{
  int i,j,found;
  for(i=0;i<NO_WORDS;i++)
  {
    found=1;
    for(j=0;j<length[i];j++)
      if(buffer[j] != words[i][j])
      {
      found=0;
      break;
      }
    if(found==1)
      return i+1;
  }
  return 0;
}

int det_inputs(char *buffer)
{
  int i=0;
  while(((buffer[i]>='A') && (buffer[i]<='Z')) || (buffer[i]=='.'))
   i++;
```

```
  return atoi(buffer+i);
}

void first_scan(ifstream& in_file)
{
  char buffer[10];
  int i, node,ind_word,no_inputs;
  cout<<"First scan\n";
  while(!in_file.eof())
  {
    in_file>>buffer;
    ind_word=check_word(buffer);
    switch (ind_word)
{
  case INPUT: in_file>>node;
            max_input++;
          break;
  case OUTPUT:in_file>>node;
            max_output++;
          break;
  case AND: no_inputs=det_inputs(buffer);
        if(no_inputs>max_and)
          max_and=no_inputs;
        for(i=0;i<=no_inputs;i++)
          in_file>>node;
        break;
  case OR: no_inputs=det_inputs(buffer);
        if(no_inputs>max_or)
          max_or=no_inputs;
         for(i=0;i<=no_inputs;i++)
          in_file>>node;
          break;
     case NOT: in_file>>node;
           in_file>>node;
           max_not++;
           break;
     default: if(buffer[0]==0)
           break;
           else
           {
           cout<<"\n\aSyntax error in input file";
           exit(1);
           }
  }
}
and_s=alloc_gate_string(max_and-1);
  or_s=alloc_gate_string(max_or-1);
  input_s=alloc_int(max_input);
  output_s=alloc_int(max_output);
  not_s=alloc_int(2*max_not);
  init_structures();
  no_init_gates=max_not;
}

void second_scan(ifstream& in_file)
{
  char buffer[10];
  int i, node,ind_word,no_inputs;
  cout<<"Second scan\n";
  in_file.seekg(0,ios::beg);
```

```
in_file.clear();
while(!in_file.eof())
{
  in_file>>buffer;
  ind_word=check_word(buffer);
  switch (ind_word)
  {
    case INPUT: in_file>>node;
           input_s[cursor_input]=node;
           cursor_input++;
           break;
    case OUTPUT:in_file>>node;
           output_s[cursor_output]=node;
           cursor_output++;
           break;
case AND: no_inputs=det_inputs(buffer);
      and_s[no_inputs-2].no_gates++;
      strcpy(and_s[no_inputs-2].name,buffer);
      for(i=0;i<=no_inputs;i++)
        in_file>>node;
      break;
case OR: no_inputs=det_inputs(buffer);
       or_s[no_inputs-2].no_gates++;
       strcpy(or_s[no_inputs-2].name,buffer);
       for(i=0;i<=no_inputs;i++)
       in_file>>node;
       break;
case NOT: in_file>>node;
      not_s[cursor_not++]=node;
      in_file>>node;
           not_s[cursor_not++]=node;
           break;
    default: if(buffer[0]==0)
           break;
           else
           {
           cout<<"\n\aSyntax error in input file";
           exit(1);
           }
  }
}
for(i=0;i<max_and-1;i++)
{
  and_s[i].gate_nodes=alloc_int((and_s[i].no_gates)*(i+3));
  no_init_gates+=and_s[i].no_gates;
}
  for(i=0;i<max_or-1;i++)
  {
    or_s[i].gate_nodes=alloc_int((or_s[i].no_gates)*(i+3));
    no_init_gates+=or_s[i].no_gates;
  }
}

void third_scan(ifstream& in_file)
{
  char buffer[10];
  int i,node,ind_word,no_inputs,curs;
  cout<<"Third scan\n";
  in_file.seekg(0,ios::beg);
  in_file.clear();
```

```
while(!in_file.eof())
{
  in_file>>buffer;
  ind_word=check_word(buffer);
  switch (ind_word)
  {
    case AND: no_inputs=det_inputs(buffer);
          curs=and_s[no_inputs-2].cursor;
          for(i=0;i<=no_inputs;i++)
            in_file>>and_s[no_inputs-2].gate_nodes[curs+i];
          and_s[no_inputs-2].cursor+=(no_inputs+1);
          break;
    case OR: no_inputs=det_inputs(buffer);
          curs=or_s[no_inputs-2].cursor;
          for(i=0;i<=no_inputs;i++)
            in_file>>or_s[no_inputs-2].gate_nodes[curs+i];
          or_s[no_inputs-2].cursor+=(no_inputs+1);
          break;
      case NOT:    in_file>>node;           //If it is a inverter gate
there are
      case INPUT:                          //two nodes to be read. If it
is just
      case OUTPUT: in_file>>node;          //a port, there is only one
node to
             break;                  //be read;
      default: if(buffer[0]==0)
            break;
            else
            {
            cout<<"\n\aSyntax error in input file";
            exit(1);
            }
    }
  }
}

void write_file(ofstream& out_file)
{
  int i,j,k;
  for(i=0;i<max_input;i++)
    out_file<<".INPUT "<<input_s[i]<<"\n";
  for(i=0;i<max_not;i++)
    if(not_s[2*i] != CANCELLED)
      out_file<<".NOT "<<not_s[2*i]<<" "<<not_s[2*i+1]<<"\n";
  for(i=0;i<max_and-1;i++)
    if(and_s[i].name[0])
      for(j=0;j<and_s[i].no_gates;j++)
      if(and_s[i].gate_nodes[j*(i+3)]!=CANCELLED)
      {
        out_file<<and_s[i].name;
        for(k=0;k<i+3;k++)
            out_file<<" "<<and_s[i].gate_nodes[j*(i+3)+k];
        out_file<<"\n";
      }
for(i=0;i<max_or-1;i++)
  if(or_s[i].name[0])
    for(j=0;j<or_s[i].no_gates;j++)
    if(or_s[i].gate_nodes[j*(i+3)]!=CANCELLED)
    {
      out_file<<or_s[i].name;
```

```
        for(k=0;k<i+3;k++)
          out_file<<" "<<or_s[i].gate_nodes[j*(i+3)+k];
        out_file<<"\n";
      }
  for(i=0;i<max_output;i++)
    out_file<<".OUTPUT "<<output_s[i]<<"\n";
}

void arrange_inputs(int *begin, int length)
{
  int i,j,i_min,min,aux;
  for(i=0;i<length-2;i++)       //The last is the output node and the
  {                                    //second-last doesn't need to be
exchanged
    min=begin[i];                 //with itself
    i_min=i;
    for(j=i+1;j<length-1;j++)    //The last is the output node which is
not
      if(begin[j]<min)            //to be modified
      {
      min=begin[j];
      i_min=j;
      }
    if(i != i_min)
    {
      aux=begin[i];
      begin[i]=begin[i_min];
      begin[i_min]=aux;
    }
  }
}

void arrange_all_inputs(void)
{
  int i,j;
  for(i=0;i<max_and-1;i++)
    for(j=0;j<and_s[i].no_gates;j++)
      arrange_inputs(and_s[i].gate_nodes+j*(i+3),i+3);
  for(i=0;i<max_or-1;i++)
    for(j=0;j<or_s[i].no_gates;j++)
      arrange_inputs(or_s[i].gate_nodes+j*(i+3),i+3);
}

int check_inputs(int *begin1, int *begin2, int length)
{
  int i, rez=1;
  if((*begin1 == CANCELLED) || (*begin2 == CANCELLED))
    return 0;
  else
  {
    for(i=0;i<length-1;i++)
      if(begin1[i] != begin2[i])
      {
      rez=0;
      break;
      }
    return rez;
  }
}
```

```
void replace_all(int dest, int source)
{
  int i,j;
  for(i=0;i<max_not;i++)
    if(not_s[2*i]==dest)
      not_s[2*i]=source;
  for(i=0;i<max_and-1;i++)
    for(j=0;j<(and_s[i].no_gates)*(i+3);j++)
      if(and_s[i].gate_nodes[j]==dest)
      and_s[i].gate_nodes[j]=source;
  for(i=0;i<max_or-1;i++)
    for(j=0;j<(or_s[i].no_gates)*(i+3);j++)
      if(or_s[i].gate_nodes[j]==dest)
      or_s[i].gate_nodes[j]=source;
  for(i=0;i<max_output;i++)
    if(output_s[i]==dest)
      output_s[i]=source;
}

void optimise_structure(void)
{
  int i,j,k,replacement;
  do
  {
    replacement=0;
    arrange_all_inputs();
    for(i=0;i<max_not-1;i++)
      for(j=i+1;j<max_not;j++)
      if((not_s[2*i]==not_s[2*j]) && (not_s[2*i] != CANCELLED))
      {
      not_s[2*j]=CANCELLED;
      replace_all(not_s[2*j+1],not_s[2*i+1]);
      replacement=1;
      no_fin_gates--;
    }
for(i=0;i<max_and-1;i++)
  for(j=0;j<and_s[i].no_gates-1;j++)
  for(k=j+1;k<and_s[i].no_gates;k++)
    if(check_inputs(and_s[i].gate_nodes+j*(i+3),
                and_s[i].gate_nodes+k*(i+3),i+3))
    {
      and_s[i].gate_nodes[k*(i+3)]=CANCELLED;
      replace_all(and_s[i].gate_nodes[k*(i+3)+i+2],
              and_s[i].gate_nodes[j*(i+3)+i+2]);
      replacement=1;
      no_fin_gates--;
      }
  for(i=0;i<max_or-1;i++)
    for(j=0;j<or_s[i].no_gates-1;j++)
    for(k=j+1;k<or_s[i].no_gates;k++)
      if(check_inputs(or_s[i].gate_nodes+j*(i+3),
                  or_s[i].gate_nodes+k*(i+3),i+3))
      {
        or_s[i].gate_nodes[k*(i+3)]=CANCELLED;
        replace_all(or_s[i].gate_nodes[k*(i+3)+i+2],
                or_s[i].gate_nodes[j*(i+3)+i+2]);
        replacement=1;
        no_fin_gates--;
      }
}
```

```
  while(replacement);
}

void dealloc_everything(void)
{
  int i;
  for(i=0;i<max_and-1;i++)
    delete and_s[i].gate_nodes;
  for(i=0;i<max_or-1;i++)
    delete or_s[i].gate_nodes;
  delete and_s;
  delete or_s;
  delete not_s;
  delete input_s;
  delete output_s;
}

void main(int no_par, char** par)
{
  ifstream in_file;
  ofstream out_file;
  if(no_par<3)
  {
    cout<<"\nToo few parameters";
    exit(1);
  }
  in_file.open(par[1],ios::in);
  if(!in_file.good())
  {
    cout<<"\n\aError: The input file cannot be opened!";
    exit(1);
  }
  cout<<"\n---------- Start optimisation ----------\n";
  first_scan(in_file);
  second_scan(in_file);
  third_scan(in_file);
  cout<<no_init_gates<<" gates in the input file\n";
  no_fin_gates=no_init_gates;
  optimise_structure();
  out_file.open(par[2],ios::out);
  if(!out_file.good())
  {
    cout<<"\n\aError: The output file cannot be opened!";
    exit(1);
  }
  write_file(out_file);
  dealloc_everything();
  in_file.close();
  out_file.close();
  cout<<no_fin_gates<<" gates in the output file\n";
  if(no_fin_gates<no_init_gates)
  {
    cout<<"The structure has been compressed at ";
    cout<<((100.0*no_fin_gates)/no_init_gates)<<"%  from  the  initial
size\n";
  }
  else
    cout<<"The structure could not be compressed\n";
}
```

Appendix A.3 - VHDL_TR.CPP

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <memmanag.h>

#define NO_WORDS 5
#define BUFFER_SIZE 30
#define NO_INPUTS_MAX 25
#define NO_PORTS 300
#define NO_MAX_FILES 5

#define _and 1
#define _or 2
#define _inv 3
#define _input 4
#define _output 5

#define INPUT_TYPE 1
#define NODE_TYPE 2
#define OUTPUT_TYPE 3

typedef char standard_list[2][NO_INPUTS_MAX];

char *dict[NO_WORDS]={"AND","OR","NOT","INPUT","OUTPUT"};
char buffer[BUFFER_SIZE];
char* node_list;
int input_list[NO_PORTS],output_list[NO_PORTS];
int max_node_number,internal_nodes,input_cursor,output_cursor;
int gate_count[NO_MAX_FILES];
int no_first_net;
ofstream output_file;

void init_list_gate_count(void)
{
  int i;
  for(i=0;i<NO_MAX_FILES;i++)
    gate_count[i]=0;
}

int search_word(char* name, int length)
{
  int i,answer=0;
  char* temp=alloc_char(length+1);
  for(i=0;i<length;i++)
    temp[i]=name[i];
  temp[length]=0;
  for(i=0;i<NO_WORDS;i++)
    if(!strcmp(dict[i],temp))
    {
      answer=i+1;
      break;
    }
  delete temp;
  return answer;
}
```

```
int word_limit(char* buffer, int& w_beg, int& w_end)
{
  int i;
  w_beg=w_end=-1;
  for(i=0;i<BUFFER_SIZE && buffer[i]!=0;i++)
  {
    if(buffer[i]>='A' && buffer[i]<='Z' && w_beg==-1)
      w_beg=i;
    if(buffer[i]<'A' || buffer[i]>'Z' && w_beg!=-1)
      w_end=i-1;
    if(w_beg!=-1  && w_end!=-1)
      return 1;
  }
  if(buffer[i]==0 && w_beg>-1)
  {
    w_end=i-1;
    return 1;
  }
  return 0;
}

void count_ports_and_nodes(int no_file, char** par)
{
  ifstream input_file;
  int w_beg,w_end,index;
  int input_number=0,output_number=0,current_node_number;
  input_cursor=0;
  output_cursor=0;
  input_number=0;
  output_number=0;
  cout<<"\n Processing file "<<no_file;
  input_file.open(par[no_file],ios::in);
  if(!input_file.good())
  {
    cout<<"\n\aError: The input file cannot be opened";
    exit(1);
  }
  input_file>>buffer;
  while(!input_file.eof() || buffer[0])
  {
    if(word_limit(buffer,w_beg,w_end))
      if(index=search_word(&buffer[w_beg],w_end-w_beg+1))
      {
      if(index==_input)
        input_number=1;
      if(index==_output)
        output_number=1;
      }
      else
      {
      cout<<"\n\aError: Sytax error in input file";
      exit(1);
      }
  else
  {
    current_node_number=atoi(buffer);
    if(max_node_number<current_node_number)
    max_node_number=current_node_number;
    if(input_number==1)
    {
```

```
      input_list[input_cursor++]=current_node_number;
      input_number=0;
      }
      if(output_number==1)
      {
      output_list[output_cursor++]=current_node_number;
      output_number=0;
      }
    }
    input_file>>buffer;
}
input_file.close();
if(input_cursor>=NO_PORTS)
    {
      cout<<"\n\aError: Too many input ports";
      exit(1);
    }
    if(output_cursor>=NO_PORTS)
    {
      cout<<"\n\aError: Too many output ports";
      exit(1);
    }
}

int find_node_in_vector(int* vector, int node_num, int total_num)
{
  int i;
  for(i=0; i<NO_PORTS;i++)
  {
    if(vector[i]==node_num)
        return total_num-i-1;
  }
  cout<<"\n\aSerious internal error in the algorithm";
  exit(1);
  return 0;
}

void write_network_entity(int no_file)
{
  output_file<<"LIBRARY ieee;\nUSE ieee.std_logic_1164.all;\n\n";
  output_file<<"ENTITY network"<<(no_file+no_first_net-1)<<" IS\n";
  output_file<<"  PORT(d_in : IN std_logic_vector("<<(input_cursor-1);
  output_file<<" DOWNTO 0);\n        d_out: OUT std_logic_vector(";
  output_file<<(output_cursor-1)<<" DOWNTO 0));\nEND network";
  output_file<<(no_file+no_first_net-1)<<";\n\n";
}

void write_logic_exp(ifstream& input_file,int no_file)
{
  #define NO_GATE_INP 30
  char gate_name[5];
  int i,no_inputs,w_beg,w_end,index;
  int gate_nodes[NO_GATE_INP];
  input_file>>buffer;
  while(!input_file.eof())
  {
    if(word_limit(buffer,w_beg,w_end))
    {
      index=search_word(&buffer[w_beg],w_end-w_beg+1);
      if(index<=3)
```

```
{
for(i=0;*(dict[index-1]+i)!=0;i++)
  gate_name[i]=(*(dict[index-1]+i));
gate_name[i]=0;
gate_count[no_file-1]++;
if(index<3)
  no_inputs=atoi(&buffer[w_end+1]);
else
  no_inputs=1;
if(no_inputs>NO_GATE_INP)
{
  cout<<"\n\aError: One of the gates has too many inputs";
  exit(1);
}
for(i=0;i<no_inputs+1;i++)
  input_file>>gate_nodes[i];
if(node_list[gate_nodes[no_inputs]-1]==NODE_TYPE)
  output_file<<"  n"<<gate_nodes[no_inputs];
else
{
  output_file<<"  d_out("<<find_node_in_vector(output_list,
                  gate_nodes[no_inputs],output_cursor);
  output_file<<")";
}
output_file<<"<=";
if (no_inputs>1)
{
  if(node_list[gate_nodes[0]-1]==NODE_TYPE)
    output_file<<" n"<<gate_nodes[0];
  else
  {
   output_file<<" d_in("<<find_node_in_vector(input_list,
                  gate_nodes[0],input_cursor);
   output_file<<")";
  }
for(i=1;i<no_inputs;i++)
{
  output_file<<" "<<gate_name;
  if(node_list[gate_nodes[i]-1]==NODE_TYPE)
    output_file<<" n"<<gate_nodes[i];
  else
  {
    output_file<<" d_in("<<find_node_in_vector(input_list,
                  gate_nodes[i],input_cursor);
    output_file<<")";
  }
}
  output_file<<";\n";
}
else
{
  output_file<<" "<<gate_name;
  if(node_list[gate_nodes[0]-1]==NODE_TYPE)
    output_file<<" n"<<gate_nodes[0];
  else
  {
    output_file<<" d_in("<<find_node_in_vector(input_list,
                  gate_nodes[0],input_cursor);
    output_file<<")";
  }
```

```
        output_file<<";\n";
      }
      }
    }
    input_file>>buffer;
  }
}

void write_network_architecture(char* file_name,int no_file)
{
  ifstream input_file;
  int
i,it_is_input,it_is_output,w_beg,w_end,index,current_node_number;
int no_signals;
node_list=alloc_char(max_node_number+1); //+1 is for safety
for(i=0;i<max_node_number;i++)
  node_list[i]=0;
cout<<"\n Reprocessing file "<<no_file;
output_file<<"ARCHITECTURE arch_network"<<(no_file+no_first_net-1);
output_file<<" OF network"<<(no_file+no_first_net-1)<<" IS\n";
input_file.open(file_name,ios::in);
if(!input_file.good())
{
  cout<<"\n\aError: The input file cannot be opened";
  exit(1);
}
input_file>>buffer;
while(!input_file.eof()|| buffer[0])
{
  if(word_limit(buffer,w_beg,w_end))
  {
    index=search_word(&buffer[w_beg],w_end-w_beg+1);
    if(index==_input)
    it_is_input=1;
    else
    it_is_input=0;
    if(index==_output)
    it_is_output=1;
    else
    it_is_output=0;
  }
  else
  {
    current_node_number=atoi(buffer);
    if(it_is_input)
      node_list[current_node_number-1]=INPUT_TYPE;
    else if(it_is_output)
      node_list[current_node_number-1]=OUTPUT_TYPE;
    else if(node_list[current_node_number-1]!=INPUT_TYPE &&
          node_list[current_node_number-1]!=OUTPUT_TYPE)
    {
      node_list[current_node_number-1]=NODE_TYPE;
      internal_nodes=1;
    }
  }
  input_file>>buffer;
}
no_signals=1;
if (internal_nodes)
{
```

```
output_file<<"    SIGNAL n";
for(i=0;i<max_node_number;i++)
  if(node_list[i]==NODE_TYPE)
  {
  output_file<<(i+1);
  break;
  }
for(i++;i<max_node_number;i++)
{
  if(node_list[i]==NODE_TYPE)
  {
    no_signals++;
  if(no_signals%10==0)
      output_file<<",\n            n";
    else
      output_file<<",n";
    output_file<<(i+1);
    }
  }
  output_file<<": std_logic;\n";
 }
 output_file<<"BEGIN\n";
 input_file.seekg(0,ios::beg);
 input_file.clear();
 write_logic_exp(input_file, no_file);
 output_file<<"END arch_network"<<(no_file+no_first_net-1)<<";\n\n";
 input_file.close();
 delete node_list;
}

void write_network_configuration(int no_file)
{
  output_file<<"CONFIGURATION conf_network"<<(no_file+no_first_net-1);
  output_file<<" OF network";
  output_file<<(no_file+no_first_net-1)<<" IS\n  FOR arch_network";
  output_file<<(no_file+no_first_net-1);
  output_file<<"\n                              END                    FOR;\nEND
conf_network"<<(no_file+no_first_net-1);
  output_file<<";\n\n";
}

void write_networks(int no_file, char** par)
{
  write_network_entity(no_file);
  write_network_architecture(par[no_file],no_file);
  write_network_configuration(no_file);
}

void main(int no_par, char** par)
{
  ifstream input_file;
  int i,j,total_gate_count=0;
  init_list_gate_count();
  if(no_par<4)
  {
    cout<<"\n\aError: Too few parameters!";
    exit(1);
  }
  if(no_par>NO_MAX_FILES+3) //3 is for prog.name+out file+ no. first
net.
```

```
{
  cout<<"\n\aError: Too many files!";
  exit(1);
}
no_first_net=atoi(par[no_par-1]);
output_file.open(par[no_par-2],ios::out);
if(!output_file.good())
{
  cout<<"\n\aError: Output file cannot be opened!";
  exit(1);
}
for(i=1;i<no_par-2;i++)
{
    for(j=0;j<NO_PORTS;j++)
    {
      input_list[j]=0;
      output_list[j]=0;
    }
    max_node_number=0;
    internal_nodes=0;
    count_ports_and_nodes(i, par);
    write_networks(i,par);
    cout<<"\nArchitecture no. "<<i<<" contains "<<gate_count[i-1]<<"
gates";
    total_gate_count+=gate_count[i-1];
  }
  cout<<"\nTotal gate count: "<<total_gate_count;
  output_file.close();
}
```

Appendix A.4 - MEMMANAG.H

```
//This is a header file used by the three universal programs
#if !defined( __STDLIB_H )
#include <stdlib.h>
#endif

char* alloc_err="\n\aError: Insufficient RAM memory for dinamic
allocation!";

float* alloc_float(int mem_length)
{
  float *pointer;
  if(mem_length>0)
  {
    if(!(pointer=new float[mem_length]))
    {
      cout<<alloc_err;
      exit(1);
    }
    return pointer;
  }
  else
    return NULL;
}

double* alloc_double(int mem_length)
{
  double *pointer;
  if(mem_length>0)
  {
```

```
    if(!(pointer=new double[mem_length]))
    {
      cout<<alloc_err;
      exit(1);
    }
    return pointer;
  }
  else
    return NULL;
}

int* alloc_int(int mem_length)
{
  int *pointer;
  if(mem_length>0)
  {
    if(!(pointer=new int[mem_length]))
    {
      cout<<alloc_err;
      exit(1);
    }
    return pointer;
  }
  else
    return NULL;
}

char* alloc_char(int mem_length)
{
  char *pointer;
  if(mem_length>0)
  {
    if(!(pointer=new char[mem_length]))
    {
      cout<<alloc_err;
      exit(1);
    }
    return pointer;
  }
  else
    return NULL;
}
```

Appendix A.5 - MATRIX.H

```
//This is a header file used by the three universal programs
#include <iostream.h>
#include <process.h>

char*  alloc_error="\n\aError:  Not  enough  memory  for  dinamic
allocation!";

class vector
{
  private:
    int length;
    double* no;
  public:
    vector(void);
    vector(int);
```

```
    ~vector(void);
    double& operator[](int);
    void resize(int);
};

class matrix
{
  private:
    int rows,columns;
    vector* val;
  public:
    matrix(int,int);
    ~matrix();
    vector& operator[](int);
    int no_rows(void);
    int no_columns(void);
};

vector::vector(void)
{
  length=0;
  no=NULL;
}

vector::vector(int nlength)
{
  length=nlength;
  if(length>0)
  {
    if(!(no=new double[length]))
    {
      cout<<alloc_error;
      exit(1);
    }
  }
  else
    no=NULL;
}

vector::~vector(void)
{
  if(no!=NULL)
    delete no;
}

double& vector::operator[](int index)
{
  if(index<0 || index>=length)
  {
    cout<<"\n\aError: Index value is outside limits";
    exit(1);
  }
  return no[index];
}

void vector::resize(int nlength)
{
  if (no!=NULL)
    delete no;
  length=nlength;
```

```
  if(length>0)
  {
    if(!(no=new double[length]))
    {
      cout<<alloc_error;
      exit(1);
    }
  }
  else
    no=NULL;
}

matrix::matrix(int length1, int length2)
{
  int i;
  if(length1<=0 || length2<=0)
  {
    cout<<"\n\aError: The matrix dimensions must be positive!";
    exit(1);
  }
  rows=length1;
  columns=length2;
  if(!(val=new vector[rows]))
  {
    cout<<alloc_error;
    exit(1);
  }
  for(i=0;i<rows;i++)
    val[i].resize(columns);
}

matrix::~matrix(void)
{
  delete [] val;
}

vector& matrix::operator[](int index)
{
  if(index<0 || index>=rows)
  {
    cout<<"\n\aError: Index value is outside limits";
    exit(1);
  }
  return val[index];
}

int matrix::no_rows(void)
{
  return rows;
}

int matrix::no_columns(void)
{
  return columns;
}
```

Appendix B

THE VHDL MODELS OF THE ANGLE SUBNETWORK AND OF POSITION SUBNETWORK

Appendix B.1 - THE POSITION SUBNETWORK

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY network1 IS
  PORT(d_in : IN std_logic_vector(9 DOWNTO 0);
       d_out: OUT std_logic_vector(53 DOWNTO 0));
END network1;

ARCHITECTURE arch_network1 OF network1 IS
    SIGNAL n11,n12,n13,n14,n16,n17,n18,n21,n22,
           n24,n25,n26,n28,n29,n30,n31,n32,n33,n34,
           n35,n36,n37,n38,n39,n40,n41,n42,n43,n44,
           n45,n46,n47,n48,n49,n50,n51,n52,n53,n54,
           n55,n57,n58,n59,n60,n61,n62,n63,n64,n65,
           n66,n67,n68,n74,n75,n80,n81,n86,n87,n93,
           n94,n95,n96,n97,n98,n99,n100,n108,n109,n110,
           n116,n117,n133,n134,n149,n150,n151,n152,n153,n154,
           n163,n164,n165,n166,n173,n174,n183,n184,n185,n197,
           n198,n199,n217,n218,n220,n221,n239,n240,n241,n242,
           n243,n245,n246,n247,n248,n249,n250,n251,n253,n254,
           n258,n259,n260,n261,n264,n265,n266,n267,n268,n269,
           n270,n271,n272,n273,n274,n275,n276,n277,n278,n279,
           n280,n281,n282,n283,n284,n285,n288,n289,n293,n294,
           n296,n297,n298,n304,n305,n314,n315,n316,n317,n324,
           n325,n326,n329,n330,n334,n335,n348,n349,n367,n368,
           n369,n372,n373,n379,n380,n390,n391,n399,n400,n401,
           n411,n412,n450,n451,n452,n453,n455,n457,n460,n462,
           n465,n469,n470,n487,n491,n492,n514,n521,n547: std_logic;
BEGIN
  n11<= NOT d_in(4);
  n28<= NOT d_in(8);
  n29<= NOT d_in(7);
  n30<= NOT d_in(6);
  n31<= NOT d_in(5);
  n243<= NOT d_in(9);
  n452<= NOT n154;
  n453<= NOT n285;
  n455<= NOT n14;
  n457<= NOT n199;
  n460<= NOT n18;
  n462<= NOT n242;
  n465<= NOT n11;
  n469<= NOT n110;
  n470<= NOT n326;
  n487<= NOT n22;
  n491<= NOT n68;
  n492<= NOT n369;
  n514<= NOT n26;
```

```
n521<= NOT n412;
n547<= NOT n451;
n13<= d_in(3) AND n12;
n22<= n11 AND n21;
n26<= n11 AND n25;
n34<= d_in(9) AND n33;
n36<= d_in(2) AND n35;
n38<= n30 AND n37;
n41<= d_in(3) AND n40;
n43<= d_in(2) AND n42;
n45<= n30 AND n44;
n47<= n29 AND n46;
n50<= n11 AND n49;
n52<= d_in(1) AND n51;
n53<= d_in(0) AND n31;
n55<= d_in(2) AND n54;
n59<= d_in(3) AND n58;
n60<= d_in(1) AND n31;
n62<= d_in(2) AND n61;
n65<= n29 AND n64;
n67<= n28 AND n66;
n75<= n29 AND n74;
n81<= d_in(9) AND n80;
n87<= n28 AND n86;
n94<= n29 AND n93;
n97<= d_in(3) AND n96;
n100<= n11 AND n99;
n117<= n28 AND n116;
n134<= d_in(9) AND n133;
n150<= n28 AND n149;
n153<= n11 AND n152;
n164<= n30 AND n163;
n166<= n11 AND n165;
n174<= n28 AND n173;
n185<= d_in(9) AND n184;
n218<= n29 AND n217;
n221<= n11 AND n220;
n240<= d_in(9) AND n239;
n246<= d_in(1) AND n245;
n247<= d_in(5) AND d_in(0);
n249<= n243 AND n248;
n251<= d_in(8) AND n250;
n254<= d_in(2) AND n253;
n259<= d_in(6) AND n258;
n261<= d_in(3) AND n260;
n265<= d_in(2) AND n264;
n268<= d_in(7) AND n267;
n270<= n11 AND n269;
n272<= d_in(7) AND n271;
n274<= d_in(6) AND n273;
n277<= d_in(2) AND n276;
n280<= d_in(3) AND n279;
n284<= d_in(8) AND n283;
n289<= d_in(3) AND n288;
n294<= d_in(7) AND n293;
n298<= n243 AND n297;
n305<= d_in(8) AND n304;
n317<= n11 AND n316;
n330<= n11 AND n329;
n335<= d_in(8) AND n334;
```

```
n349<= n243 AND n348;
n373<= d_in(7) AND n372;
n380<= n11 AND n379;
n391<= d_in(8) AND n390;
n401<= n243 AND n400;
n451<= n243 AND n450;
d_out(53)<= n452 AND n453;
d_out(52)<= n154 AND n455;
d_out(50)<= n199 AND n460;
d_out(48)<= n242 AND n465;
d_out(47)<= n11 AND n453;
d_out(46)<= n469 AND n470;
d_out(38)<= n22 AND n470;
d_out(37)<= n491 AND n492;
d_out(27)<= n26 AND n492;
d_out(26)<= n369 AND n455;
d_out(16)<= n242 AND n369;
d_out(15)<= n412 AND n460;
d_out(7)<= n199 AND n412;
d_out(6)<= n451 AND n465;
d_out(5)<= n11 AND n491;
d_out(3)<= n22 AND n469;
d_out(1)<= n26 AND n452;
d_out(0)<= n154 AND n451;
n17<= d_in(3) AND d_in(2) AND n16;
n24<= d_in(2) AND d_in(1) AND d_in(0);
n39<= d_in(1) AND d_in(0) AND n31;
n57<= d_in(1) AND n30 AND n51;
n183<= d_in(3) AND n29 AND n108;
n198<= n11 AND n28 AND n197;
n266<= d_in(6) AND d_in(5) AND d_in(1);
n275<= d_in(5) AND d_in(1) AND d_in(0);
n296<= d_in(6) AND d_in(2) AND n281;
n315<= d_in(7) AND d_in(3) AND n314;
n368<= d_in(8) AND n11 AND n367;
n399<= d_in(7) AND d_in(3) AND n324;
d_out(51)<= n14 AND n453 AND n457;
d_out(49)<= n18 AND n453 AND n462;
d_out(45)<= n110 AND n285 AND n455;
d_out(44)<= n14 AND n452 AND n470;
d_out(43)<= n154 AND n285 AND n460;
d_out(42)<= n18 AND n457 AND n470;
d_out(41)<= n199 AND n285 AND n465;
d_out(40)<= n11 AND n462 AND n470;
d_out(39)<= n242 AND n285 AND n487;
d_out(36)<= n68 AND n326 AND n455;
d_out(35)<= n14 AND n469 AND n492;
d_out(34)<= n110 AND n326 AND n460;
d_out(33)<= n18 AND n452 AND n492;
d_out(32)<= n154 AND n326 AND n465;
d_out(31)<= n11 AND n457 AND n492;
d_out(30)<= n199 AND n326 AND n487;
d_out(29)<= n22 AND n462 AND n492;
d_out(28)<= n242 AND n326 AND n514;
d_out(25)<= n14 AND n491 AND n521;
d_out(24)<= n68 AND n369 AND n460;
d_out(23)<= n18 AND n469 AND n521;
d_out(22)<= n110 AND n369 AND n465;
d_out(21)<= n11 AND n452 AND n521;
d_out(20)<= n154 AND n369 AND n487;
```

```
d_out(19)<= n22 AND n457 AND n521;
d_out(18)<= n199 AND n369 AND n514;
d_out(17)<= n26 AND n462 AND n521;
d_out(14)<= n18 AND n491 AND n547;
d_out(13)<= n68 AND n412 AND n465;
d_out(12)<= n11 AND n469 AND n547;
d_out(11)<= n110 AND n412 AND n487;
d_out(10)<= n22 AND n452 AND n547;
d_out(9)<= n154 AND n412 AND n514;
d_out(8)<= n26 AND n457 AND n547;
d_out(4)<= n68 AND n451 AND n487;
d_out(2)<= n110 AND n451 AND n514;
n63<= d_in(1) AND d_in(0) AND n30 AND n31;
  n95<= d_in(2) AND d_in(1) AND n30 AND n31;
  n109<= d_in(3) AND n28 AND n29 AND n108;
  n278<= d_in(6) AND d_in(5) AND d_in(1) AND d_in(0);
  n282<= d_in(7) AND d_in(6) AND d_in(2) AND n281;
  n325<= d_in(8) AND d_in(7) AND d_in(3) AND n324;
  n48<= d_in(2) AND d_in(1) AND d_in(0) AND n30 AND n31;
  n411<= d_in(8) AND d_in(7) AND d_in(3) AND n11 AND n314;
  n98<= d_in(2) AND d_in(1) AND d_in(0) AND n29 AND n30 AND n31;
  n151<= d_in(3) AND d_in(2) AND d_in(1) AND n29 AND n30 AND n31;
  n241<= d_in(3) AND d_in(2) AND d_in(1) AND n11 AND n28 AND n29 AND
n30 AND n31;
  n12<= d_in(2) OR d_in(1);
  n14<= n11 OR n13;
  n16<= d_in(1) OR d_in(0);
  n18<= n11 OR n17;
  n21<= d_in(3) OR d_in(2);
  n25<= d_in(3) OR n24;
  n33<= d_in(0) OR n32;
  n44<= d_in(1) OR n31;
  n46<= n43 OR n45;
  n51<= d_in(0) OR n31;
  n61<= n30 OR n60;
  n64<= n62 OR n63;
  n66<= n59 OR n65;
  n96<= n94 OR n95;
  n108<= n55 OR n57;
  n152<= n150 OR n151;
  n154<= n134 OR n153;
  n163<= d_in(1) OR n53;
  n197<= n97 OR n98;
  n199<= n185 OR n198;
  n217<= n43 OR n164;
  n239<= n67 OR n221;
  n242<= n240 OR n241;
  n245<= d_in(5) OR d_in(0);
  n258<= n246 OR n247;
  n264<= d_in(6) OR n246;
  n267<= n265 OR n266;
  n276<= n274 OR n275;
  n281<= d_in(5) OR d_in(1);
  n283<= n280 OR n282;
  n314<= n254 OR n259;
  n316<= n305 OR n315;
  n324<= n277 OR n278;
  n367<= n261 OR n268;
  n369<= n349 OR n368;
  n412<= n401 OR n411;
```

```
n450<= n270 OR n284;
n37<= d_in(1) OR d_in(0) OR n31;
n42<= d_in(1) OR n30 OR n31;
n54<= n30 OR n52 OR n53;
n58<= n29 OR n55 OR n57;
n68<= n34 OR n50 OR n67;
n93<= n36 OR n38 OR n39;
n99<= n87 OR n97 OR n98;
n110<= n81 OR n100 OR n109;
n149<= n41 OR n47 OR n48;
n184<= n166 OR n174 OR n183;
n253<= d_in(6) OR d_in(1) OR n247;
n260<= d_in(7) OR n254 OR n259;
n269<= n251 OR n261 OR n268;
n273<= d_in(5) OR d_in(1) OR d_in(0);
n279<= n272 OR n277 OR n278;
n285<= n249 OR n270 OR n284;
n326<= n298 OR n317 OR n325;
n390<= n289 OR n294 OR n296;
n400<= n380 OR n391 OR n399;
n35<= d_in(1) OR d_in(0) OR n30 OR n31;
n40<= n29 OR n36 OR n38 OR n39;
n49<= n28 OR n41 OR n47 OR n48;
n86<= d_in(3) OR n29 OR n43 OR n45;
n133<= n11 OR n59 OR n65 OR n117;
n173<= d_in(3) OR n62 OR n63 OR n75;
n220<= n28 OR n41 OR n48 OR n218;
n271<= d_in(6) OR d_in(5) OR d_in(2) OR d_in(1);
n293<= d_in(6) OR d_in(2) OR n246 OR n247;
n304<= d_in(7) OR d_in(3) OR n265 OR n266;
  n348<= n280 OR n282 OR n330 OR n335;
  n74<= d_in(2) OR d_in(1) OR d_in(0) OR n30 OR n31;
  n165<= d_in(3) OR n28 OR n29 OR n43 OR n164;
  n288<= d_in(7) OR d_in(6) OR d_in(5) OR d_in(2) OR d_in(1);
  n297<= d_in(8) OR n11 OR n289 OR n294 OR n296;
  n372<= d_in(6) OR d_in(5) OR d_in(2) OR d_in(1) OR d_in(0);
  n379<= d_in(8) OR d_in(3) OR n265 OR n266 OR n373;
  n80<= d_in(3) OR n11 OR n28 OR n62 OR n63 OR n75;
  n334<= d_in(7) OR d_in(6) OR d_in(3) OR d_in(2) OR n246 OR n247;
  n116<= d_in(3) OR d_in(2) OR d_in(1) OR d_in(0) OR n29 OR n30 OR
n31;
  n250<= d_in(7) OR d_in(6) OR d_in(5) OR d_in(3) OR d_in(2) OR
d_in(1) OR d_in(0);
  n32<= d_in(3) OR d_in(2) OR d_in(1) OR n11 OR n28 OR n29 OR n30 OR
n31;
  n248<= d_in(8) OR d_in(7) OR d_in(6) OR d_in(3) OR d_in(2) OR n11 OR
n246 OR n247;
  n329<= d_in(8) OR d_in(7) OR d_in(6) OR d_in(5) OR d_in(3) OR
d_in(2) OR d_in(1) OR d_in(0);
END arch_network1;

CONFIGURATION conf_network1 OF network1 IS
  FOR arch_network1
  END FOR;
END conf_network1;
```

Appendix B.2 - THE ANGLE SUBNETWORK

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY network2 IS
  PORT(d_in : IN std_logic_vector(9 DOWNTO 0);
       d_out: OUT std_logic_vector(17 DOWNTO 0));
END network2;

ARCHITECTURE arch_network2 OF network2 IS
    SIGNAL n11,n12,n13,n14,n15,n16,n17,n18,n19,
           n20,n21,n22,n29,n30,n31,n32,n33,n34,n35,
           n36,n37,n38,n39,n40,n41,n43,n44,n46,n47,
           n48,n49,n57,n58,n59,n60,n62,n63,n64,n65,
           n69,n70,n76,n77,n78,n79,n80,n81,n82,n83,
           n84,n85,n89,n90,n91,n92,n93,n100,n101,n103,
           n104,n105,n106,n110,n111,n112,n113,n114,n115,n116,
           n117,n118,n120,n121,n122,n123,n124,n125,n127,n128,
           n129,n130,n131,n132,n133,n135,n137,n138,n139,n140,
           n147,n148,n150,n151,n152,n157,n158,n159,n160,n173,
           n174,n175,n176,n205,n206,n215,n216,n217,n218,n219,
           n222,n223,n225,n226,n230,n231,n232,n233,n235,n236,
           n238,n239,n240,n241,n242,n243,n244,n245,n246,n250,
           n251,n252,n253,n260,n261,n263,n264,n265,n266,n267,
           n270,n271,n272,n273,n274,n275,n276,n277,n278,n280,
           n281,n282,n283,n287,n288,n290,n291,n292,n293,n294,
           n302,n305,n306,n310,n311,n312,n313,n314,n315,n316,
           n317,n318,n319,n320,n321,n322,n330,n331,n332,n333,
           n334,n335,n336,n337,n343,n344,n345,n346,n347,n348,
           n349,n350,n351,n352,n353,n361,n362,n365,n366,n367,
           n368,n369,n370,n372,n373,n375,n376,n377,n378,n379,
           n380,n381,n382,n383,n384,n385,n395,n396,n397,n398,
           n399,n400,n401,n405,n406,n407,n408,n409,n410,n411,
           n412,n413,n414,n415,n419,n420,n421,n422,n423,n424,
           n426,n427,n437,n438,n439,n440,n441,n443,n444,n446,
           n447,n448,n450,n452,n453,n454,n455,n456,n457,n459,
           n460,n461,n462,n466,n467,n468,n469,n472,n473,n474,
           n475,n476,n477,n478,n479,n490,n491,n496,n497,n505,
           n506,n507,n508,n509,n529,n530,n542,n544,n545,n548,
           n551,n552,n553,n554,n555,n556,n557,n558,n559,n561,
           n562,n563,n564,n565,n566,n570,n571,n572,n573,n576,
           n577,n579,n580,n581,n582,n593,n595,n596,n597,n598,
           n602,n603,n604,n605,n606,n607,n608,n610,n611,n612,
           n616,n617,n618,n619,n620,n621,n622,n623,n624,n632,
           n633,n637,n638,n639,n640,n641,n643,n644,n646,n647,
           n648,n649,n650,n651,n652,n653,n661,n662,n667,n668,
           n669: std_logic;
BEGIN
  n11<= NOT d_in(4);
  n12<= NOT d_in(8);
  n13<= NOT d_in(7);
  n18<= NOT d_in(6);
  n19<= NOT d_in(5);
  n343<= NOT d_in(3);
  n344<= NOT d_in(2);
  n347<= NOT d_in(1);
  n348<= NOT d_in(0);
  n15<= d_in(0) AND n14;
  n17<= n11 AND n16;
  n20<= n18 AND n19;
  n30<= d_in(2) AND n29;
  n31<= d_in(0) AND n13;
  n33<= n12 AND n32;
  n36<= n11 AND n35;
```

```
n38<= n13 AND n37;
n41<= n12 AND n40;
n44<= n13 AND n43;
n47<= d_in(1) AND n46;
n58<= d_in(1) AND n57;
n60<= d_in(0) AND n59;
n63<= n13 AND n62;
n65<= d_in(2) AND n64;
n70<= d_in(1) AND n69;
n77<= n12 AND n76;
n79<= n11 AND n78;
n80<= d_in(0) AND n18;
n82<= d_in(1) AND n81;
n85<= d_in(2) AND n84;
n91<= n12 AND n90;
n101<= d_in(1) AND n100;
n104<= d_in(3) AND n103;
n106<= d_in(2) AND n105;
n111<= n13 AND n110;
n114<= n12 AND n113;
n118<= n11 AND n117;
n120<= n18 AND n115;
n121<= d_in(0) AND n19;
n123<= n12 AND n122;
n125<= n13 AND n124;
n128<= d_in(1) AND n127;
n130<= d_in(2) AND n129;
n133<= d_in(3) AND n132;
n135<= d_in(1) AND n37;
n140<= d_in(9) AND n139;
n148<= n18 AND n147;
n150<= d_in(1) AND n115;
n152<= n13 AND n151;
n158<= d_in(2) AND n157;
n160<= n12 AND n159;
n174<= d_in(3) AND n173;
n176<= d_in(9) AND n175;
n206<= n11 AND n205;
n215<= d_in(1) AND n19;
n217<= n12 AND n216;
n219<= n13 AND n218;
n223<= d_in(2) AND n222;
n226<= d_in(3) AND n225;
n233<= d_in(9) AND n232;
n236<= d_in(3) AND n235;
n238<= d_in(2) AND n124;
n240<= n18 AND n239;
n243<= n13 AND n242;
n246<= n12 AND n245;
n253<= n11 AND n252;
n261<= n18 AND n260;
n263<= d_in(2) AND n239;
n265<= n13 AND n264;
n267<= n18 AND n266;
n271<= d_in(3) AND n270;
n273<= d_in(9) AND n272;
n274<= d_in(1) AND d_in(0);
n276<= n18 AND n275;
n278<= n19 AND n277;
n281<= d_in(2) AND n280;
```

```
n283<= n13 AND n282;
n288<= n18 AND n287;
n292<= d_in(3) AND n291;
n302<= d_in(2) AND n147;
n306<= d_in(3) AND n305;
n311<= n18 AND n310;
n313<= d_in(9) AND n312;
n315<= d_in(3) AND n314;
n317<= n18 AND n316;
n319<= n13 AND n318;
n330<= d_in(2) AND n277;
n332<= d_in(9) AND n331;
n334<= n19 AND n333;
n335<= d_in(3) AND d_in(2);
n346<= d_in(9) AND n345;
n349<= n347 AND n348;
n351<= n343 AND n350;
n362<= n343 AND n361;
n366<= n19 AND n365;
n368<= n18 AND n367;
n370<= d_in(9) AND n369;
n373<= n343 AND n372;
n376<= n18 AND n375;
n379<= n13 AND n378;
n382<= d_in(4) AND n381;
n385<= n12 AND n384;
n396<= n344 AND n395;
n399<= n13 AND n398;
n401<= n18 AND n400;
n406<= n343 AND n405;
n409<= d_in(9) AND n408;
n411<= n18 AND n410;
n413<= n344 AND n412;
n415<= n13 AND n414;
n420<= n344 AND n419;
n422<= n343 AND n421;
n424<= d_in(4) AND n423;
n427<= n12 AND n426;
n438<= n347 AND n437;
n439<= n19 AND n348;
n441<= n12 AND n440;
n444<= n344 AND n443;
n448<= n343 AND n447;
n450<= n18 AND n412;
n455<= d_in(9) AND n454;
n457<= n343 AND n456;
n459<= n18 AND n395;
n460<= n19 AND n347;
n462<= n13 AND n461;
n469<= n12 AND n468;
n473<= n344 AND n472;
n478<= d_in(4) AND n477;
n491<= n13 AND n490;
n497<= n12 AND n496;
n506<= n343 AND n505;
n509<= d_in(9) AND n508;
n530<= d_in(4) AND n529;
n542<= n18 AND n437;
n545<= n343 AND n544;
n548<= n13 AND n443;
```

```
n552<= n12 AND n551;
n554<= n347 AND n553;
n559<= d_in(4) AND n558;
n561<= n12 AND n456;
n563<= n347 AND n562;
n564<= n18 AND n348;
n566<= n344 AND n565;
n573<= n343 AND n572;
n577<= n13 AND n576;
n582<= d_in(9) AND n581;
n593<= n13 AND n562;
n596<= n344 AND n595;
n598<= n347 AND n597;
n603<= n12 AND n602;
n606<= d_in(4) AND n605;
n608<= n347 AND n607;
n610<= n13 AND n553;
n612<= n344 AND n611;
n617<= n13 AND n616;
n619<= n12 AND n618;
n621<= d_in(9) AND n620;
n624<= n343 AND n623;
n633<= n12 AND n632;
n637<= n59 AND n348;
n639<= n347 AND n638;
n641<= d_in(4) AND n640;
n644<= n12 AND n643;
n647<= n347 AND n646;
n650<= n344 AND n649;
n662<= d_in(4) AND n661;
n667<= n12 AND n21;
n34<= d_in(1) AND d_in(0) AND n13;
n39<= d_in(0) AND n18 AND n19;
n83<= d_in(0) AND n13 AND n18;
n93<= d_in(9) AND d_in(3) AND n92;
n241<= d_in(1) AND d_in(0) AND n19;
n290<= d_in(2) AND n19 AND n277;
n294<= n11 AND n12 AND n293;
n322<= n11 AND n12 AND n321;
n397<= n19 AND n347 AND n348;
n446<= n18 AND n347 AND n437;
n453<= n13 AND n344 AND n452;
n467<= n18 AND n344 AND n466;
n476<= n13 AND n343 AND n475;
n555<= n18 AND n19 AND n348;
n557<= n13 AND n344 AND n556;
n571<= n13 AND n347 AND n570;
n580<= n12 AND n344 AND n579;
n653<= d_in(9) AND n343 AND n652;
n49<= d_in(9) AND d_in(3) AND d_in(2) AND n48;
n89<= d_in(0) AND n13 AND n18 AND n19;
n112<= d_in(1) AND d_in(0) AND n18 AND n19;
n138<= d_in(2) AND n12 AND n13 AND n137;
n231<= d_in(2) AND n13 AND n18 AND n230;
n244<= d_in(2) AND d_in(1) AND n18 AND n19;
n251<= d_in(3) AND d_in(2) AND n13 AND n250;
n377<= n19 AND n344 AND n347 AND n348;
n474<= n18 AND n19 AND n347 AND n348;
n648<= n13 AND n18 AND n19 AND n348;
n116<= d_in(2) AND d_in(1) AND n13 AND n18 AND n115;
```

```
  n131<= d_in(1) AND d_in(0) AND n13 AND n18 AND n19;
  n337<= n11 AND n12 AND n13 AND n18 AND n336;
  n353<= d_in(4) AND n12 AND n13 AND n18 AND n352;
  n380<= n18 AND n19 AND n343 AND n344 AND n347;
  n407<= n18 AND n19 AND n344 AND n347 AND n348;
  n604<= n13 AND n18 AND n19 AND n347 AND n348;
  n651<= n12 AND n13 AND n18 AND n347 AND n348;
  n669<= d_in(9) AND n343 AND n344 AND n347 AND n668;
  n320<= d_in(3) AND d_in(2) AND d_in(1) AND d_in(0) AND n18 AND n19;
  n507<= n13 AND n18 AND n19 AND n344 AND n347 AND n348;
  n22<= d_in(9) AND d_in(3) AND d_in(2) AND d_in(1) AND d_in(0) AND
n12 AND n21;
  n383<= n13 AND n18 AND n19 AND n343 AND n344 AND n347 AND n348;
  n622<= n12 AND n13 AND n18 AND n19 AND n344 AND n347 AND n348;
  n479<= n12 AND n13 AND n18 AND n19 AND n343 AND n344 AND n347 AND
n348;
  n14<= n12 OR n13;
  n21<= n13 OR n20;
  d_out(17)<= n17 OR n22;
  n32<= d_in(1) OR n31;
  n37<= d_in(0) OR n18;
  n43<= d_in(0) OR n20;
  n46<= n39 OR n44;
  n48<= n41 OR n47;
  d_out(16)<= n36 OR n49;
  n59<= n18 OR n19;
  n62<= n20 OR n60;
  n76<= n63 OR n70;
  n81<= n13 OR n80;
  n90<= n82 OR n89;
  n92<= n85 OR n91;
  d_out(15)<= n79 OR n93;
  n110<= n80 OR n101;
  n115<= d_in(0) OR n19;
  n127<= n18 OR n121;
  n129<= n125 OR n128;
  n137<= n39 OR n135;
  n139<= n133 OR n138;
  d_out(14)<= n118 OR n140;
  n157<= n148 OR n150;
  n173<= n152 OR n158;
  n205<= n160 OR n174;
  d_out(13)<= n176 OR n206;
  n222<= n18 OR n150;
  n230<= n121 OR n150;
  n239<= d_in(1) OR n19;
  n250<= n148 OR n241;
  n252<= n246 OR n251;
  d_out(12)<= n233 OR n253;
  n270<= n263 OR n267;
  n277<= d_in(1) OR d_in(0);
  n280<= n274 OR n278;
  n291<= n288 OR n290;
  n293<= n283 OR n292;
  d_out(11)<= n273 OR n294;
  n310<= n215 OR n302;
  n316<= d_in(2) OR n19;
  n318<= n315 OR n317;
  n321<= n319 OR n320;
  d_out(10)<= n313 OR n322;
```

```
n336<= n334 OR n335;
d_out(9)<= n332 OR n337;
n350<= n344 OR n349;
n352<= n19 OR n351;
d_out(8)<= n346 OR n353;
n365<= n347 OR n348;
n367<= n344 OR n366;
n381<= n379 OR n380;
n384<= n382 OR n383;
d_out(7)<= n370 OR n385;
n412<= n19 OR n347;
n419<= n19 OR n349;
n421<= n411 OR n420;
n423<= n415 OR n422;
n426<= n383 OR n424;
d_out(6)<= n409 OR n427;
n437<= n19 OR n348;
n452<= n397 OR n450;
n466<= n438 OR n439;
n472<= n18 OR n438;
n475<= n473 OR n474;
n477<= n469 OR n476;
n553<= n18 OR n348;
n556<= n554 OR n555;
n570<= n439 OR n542;
n576<= n347 OR n542;
n579<= n474 OR n577;
n581<= n573 OR n580;
d_out(3)<= n559 OR n582;
n616<= n20 OR n348;
n618<= n608 OR n617;
n620<= n612 OR n619;
n623<= n621 OR n622;
d_out(2)<= n606 OR n624;
n638<= n13 OR n637;
n652<= n650 OR n651;
d_out(1)<= n641 OR n653;
n668<= n348 OR n667;
d_out(0)<= n662 OR n669;
n40<= d_in(1) OR n38 OR n39;
n57<= d_in(0) OR n13 OR n20;
n64<= n12 OR n58 OR n63;
n69<= n13 OR n20 OR n60;
n84<= n12 OR n82 OR n83;
n100<= d_in(0) OR n18 OR n19;
n113<= n106 OR n111 OR n112;
n132<= n123 OR n130 OR n131;
n147<= d_in(1) OR d_in(0) OR n19;
n151<= d_in(2) OR n148 OR n150;
n159<= d_in(3) OR n152 OR n158;
n175<= n11 OR n160 OR n174;
n225<= n112 OR n219 OR n223;
n242<= n238 OR n240 OR n241;
n245<= n236 OR n243 OR n244;
n264<= d_in(3) OR n261 OR n263;
n266<= d_in(2) OR d_in(1) OR n19;
n275<= d_in(2) OR n19 OR n274;
n282<= d_in(3) OR n276 OR n281;
n287<= d_in(2) OR n274 OR n278;
n305<= n18 OR n278 OR n302;
```

```
n314<= d_in(2) OR n18 OR n19;
n361<= n18 OR n19 OR n344;
n375<= n19 OR n344 OR n349;
n378<= n373 OR n376 OR n377;
n395<= n19 OR n347 OR n348;
n405<= n396 OR n397 OR n401;
n410<= n19 OR n344 OR n347;
n414<= n343 OR n411 OR n413;
n443<= n18 OR n347 OR n439;
n447<= n13 OR n444 OR n446;
n461<= n344 OR n459 OR n460;
n468<= n457 OR n462 OR n467;
d_out(5)<= n455 OR n478 OR n479;
n505<= n444 OR n474 OR n491;
n529<= n497 OR n506 OR n507;
d_out(4)<= n479 OR n509 OR n530;
n551<= n344 OR n446 OR n548;
n562<= n18 OR n19 OR n348;
n565<= n13 OR n563 OR n564;
n572<= n561 OR n566 OR n571;
n602<= n555 OR n593 OR n598;
n607<= n13 OR n18 OR n348;
n611<= n12 OR n608 OR n610;
n632<= n13 OR n347 OR n348;
n646<= n13 OR n20 OR n348;
n649<= n644 OR n647 OR n648;
n78<= d_in(9) OR d_in(3) OR n65 OR n77;
n105<= d_in(1) OR d_in(0) OR n13 OR n18;
n117<= d_in(9) OR n104 OR n114 OR n116;
n124<= d_in(1) OR d_in(0) OR n18 OR n19;
n232<= n11 OR n217 OR n226 OR n231;
n260<= d_in(2) OR d_in(1) OR d_in(0) OR n19;
n272<= n11 OR n12 OR n265 OR n271;
n333<= d_in(3) OR d_in(2) OR d_in(1) OR d_in(0);
n372<= n18 OR n19 OR n344 OR n349;
n398<= n18 OR n343 OR n396 OR n397;
n400<= n19 OR n344 OR n347 OR n348;
n454<= d_in(4) OR n441 OR n448 OR n453;
n490<= n18 OR n344 OR n347 OR n439;
n496<= n343 OR n444 OR n474 OR n491;
n508<= d_in(4) OR n497 OR n506 OR n507;
n558<= d_in(9) OR n545 OR n552 OR n557;
n595<= n12 OR n347 OR n555 OR n593;
n597<= n13 OR n18 OR n19 OR n348;
n643<= n13 OR n20 OR n347 OR n348;
n16<= d_in(9) OR d_in(3) OR d_in(2) OR d_in(1) OR n15;
n35<= d_in(9) OR d_in(3) OR n30 OR n33 OR n34;
n103<= d_in(2) OR n12 OR n13 OR n39 OR n101;
n122<= d_in(2) OR d_in(1) OR n13 OR n120 OR n121;
n216<= d_in(3) OR d_in(2) OR n13 OR n148 OR n215;
n218<= d_in(2) OR d_in(1) OR d_in(0) OR n18 OR n19;
n235<= d_in(2) OR d_in(1) OR n13 OR n18 OR n121;
n312<= n11 OR n12 OR n13 OR n306 OR n311;
n369<= d_in(4) OR n12 OR n13 OR n362 OR n368;
n408<= d_in(4) OR n12 OR n399 OR n406 OR n407;
n605<= d_in(9) OR n343 OR n596 OR n603 OR n604;
n640<= d_in(9) OR n343 OR n344 OR n633 OR n639;
n29<= d_in(1) OR d_in(0) OR n12 OR n13 OR n18 OR n19;
n440<= n13 OR n18 OR n343 OR n344 OR n438 OR n439;
n456<= n13 OR n18 OR n19 OR n344 OR n347 OR n348;
```

```
  n544<= n12 OR n13 OR n344 OR n347 OR n439 OR n542;
  n331<= d_in(3) OR n11 OR n12 OR n13 OR n18 OR n19 OR n330;
  n345<= d_in(4) OR n12 OR n13 OR n18 OR n19 OR n343 OR n344;
  n661<= d_in(9) OR n12 OR n13 OR n343 OR n344 OR n347 OR n348;
END arch_network2;

CONFIGURATION conf_network2 OF network2 IS
  FOR arch_network2
  END FOR;
END conf_network2;
```

Appendix C
SIN_ROM.CPP

```
/* This program generates the VHDL model of the internal look-up table
used by tier1.
*/
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <process.h>
#include <string.h>
#include <conio.h>

#define AMPL 255
#define N_STEPS 64
#define FileName "c:\\andrei\\sin_rom.vhd"
const int upper_index=((int)floor(log(N_STEPS-1)/log(2)));

void write_header(ofstream& f)
{
  f<<"LIBRARY IEEE;"<<endl;
  f<<"USE IEEE.std_logic_1164.ALL;"<<endl;
  f<<"USE IEEE.std_logic_unsigned.ALL;"<<endl<<endl;
  f<<"ENTITY sin_rom IS"<<endl;
  f<<"  PORT("<<endl;
  f<<"    A: IN std_logic_vector("<<upper_index;
  f<<" DOWNTO 0);"<<endl;
  f<<"    DO: OUT std_logic_vector(2 DOWNTO 0));"<<endl;
  f<<"END sin_rom;"<<endl<<endl;
  f<<"ARCHITECTURE sin_rom_arch OF sin_rom IS"<<endl;
  f<<"  TYPE mem_data IS ARRAY (0 TO "<<(pow(2,upper_index+1)-1);
  f<<") OF std_logic_vector(2 downto 0);"<<endl;
  f<<"  constant VD: mem_data :="<<endl<<"  (";
}

void write_end(ofstream& f)
{
  f<<"BEGIN"<<endl;
  f<<"  PROCESS(A)"<<endl;
  f<<"  begin"<<endl;
  f<<"      DO<=VD(conv_integer(A));"<<endl;
  f<<"  END PROCESS;"<<endl;
  f<<"END sin_rom_arch;";
}

void main(void)
{
  clrscr();
  ofstream f;
  int sample;
  double step=M_PI/2.0/N_STEPS;
  int sum=-AMPL,max=0;
  f.open(FileName,ios::out);
  if(f.fail())
  {
    cout<<"Error:The file could not be opened"<<endl;
    exit(1);
  }
```

```
write_header(f);
for(int i=0;i<N_STEPS;i++)
{
  sample=floor(AMPL*sin(-M_PI_2+(i+1)*step)-sum+0.5);
  if(max<sample)
    max=sample;
  sum+=sample;
  cout<<sample<<endl;
  switch(sample)
  {
    case 0: f<<"  ('0','0','0')";
          if (i<N_STEPS-1)
          f<<","<<endl;
          break;
case 1: f<<"  ('0','0','1')";
          if (i<N_STEPS-1)
          f<<","<<endl;
          break;
case 2: f<<"  ('0','1','0')";
          if (i<N_STEPS-1)
          f<<","<<endl;
          break;
case 3: f<<"  ('0','1','1')";
          if (i<N_STEPS-1)
          f<<","<<endl;
          break;
case 4: f<<"  ('1','0','0')";
          if (i<N_STEPS-1)
          f<<","<<endl;
          break;
        case 5: f<<"  ('1','0','1')";
              if (i<N_STEPS-1)
              f<<","<<endl;
              break;
        case 6: f<<"  ('1','1','0')";
              if (i<N_STEPS-1)
              f<<","<<endl;
              break;
        default: f<<"  ('1','1','1')";
              if (i<N_STEPS-1)
              f<<","<<endl;
    }
  }
  f<<");"<<endl;
  write_end(f);
  f.close();
}
```