An Asynchronous Soft-Output Viterbi Algorithm Decoder

CHAN Wing-kin

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Electronic Engineering

Abstract of this thesis entitled:

An Asynchronous

Soft-Output Viterbi Algorithm Decoder

Submitted by CHAN Wing-kin

for the degree of Master of Philosophy in Electronic Engineering

at The Chinese University of Hong Kong in July 2003

This thesis presents a new asynchronous VLSI design of Soft-Output Viterbi Algorithm decoder. The Viterbi algorithm is already widely used in digital communication systems and deep space communications. A new class of concatenated convolutional codes is developed in the past fifteen years, which can give bit-error rate close to the Shannon Limit. The Soft-output Viterbi Algorithm (SOVA) is a modification of the classical Viterbi algorithm that allows it to be used as a component decoder for decoding concentrated convolutional codes.

There is no global clock in asynchronous circuits; instead local handshakes and timing are employed to transfer data. Asynchronous also give out lower electromagnetic emission than synchronous counterpart that is preferable in many applications such as mixed-signal systems for reliability and lower interference.

This work adopts an asynchronous approach and static CMOS technology is used. New asynchronous add-compare-select unit and novel asynchronous traceback algorithm are developed in this project. Through asynchronous circuit style and voltage scaling, it also gives opportunity for low-power applications like mobile handset for telecommunication systems.

摘要

本論文介紹了一基於異步集成電路設計方法的軟式輸出維特比演算法解碼器。維特比演算法於數位通訊系統及太空通訊中的使用十分廣泛。近十五年來發展出來的迴旋鏈結編碼更可使其錯誤更正能力十分接近理論上的雪農極限。軟式輸出維特比演算法爲演變自經典維特比演算法，它可以用作迴旋鏈結編碼解碼器中的組成解碼器。

異步電路中沒有全晶片的時脈，而數據傳送是靠聯絡電路控制。異步電路發出的電磁波干擾比同步電路爲低，這於混合訊號系統可靠性及減少干擾十分有利。

本設計使用異步電路設計方法及靜態 CMOS 技術。同時開發新的異步「加-比較-選擇」單元及 Traceback 單元。以異步電路技術及電壓調整，低功率應用如移動電話等可以實現。

Acknowledgements

Thanks are due to my supervisor Prof. C. S. Choy. It was his patience and invaluable guidance that this work was accomplished. I would also like to express my appreciation for Prof. C. F. Chan and Prof. K. P. Pun for their guidance on VLSI design. I would like to thank the lab technician Mr. W. Y. Yeung who prepares and maintains the CAD tools needed in the project. I would also like to express my deepest gratitude to my colleagues in ASIC lab and various persons who give assistance in this project. Finally, I am indebted to my Lord Jesus Christ who has been my source of inspiration and strength.

Chan Wing Kin

July 2003

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Overview of Communication Systems

In a communication system, the original signal sources are first converted to a bit-stream in a source encoder (e.g. audio/video encoder). The channel encoder then encodes the information bits before sending to the channel by the modulator. Channel encoding introduces redundancy to the information bit-stream to combat noise and imperfection by forward error correction (FEC). The information is modulated with a chosen scheme and carrier frequency.

The received signal is first demodulated and then decoded according to the coding scheme. Then the bit-stream is decoded to original by source decoder. The convolutional encoder and the Viterbi Decoder is usually used in the channel coding and decoding accordingly in modern communication systems such as in IS-95 CDMA and GSM.

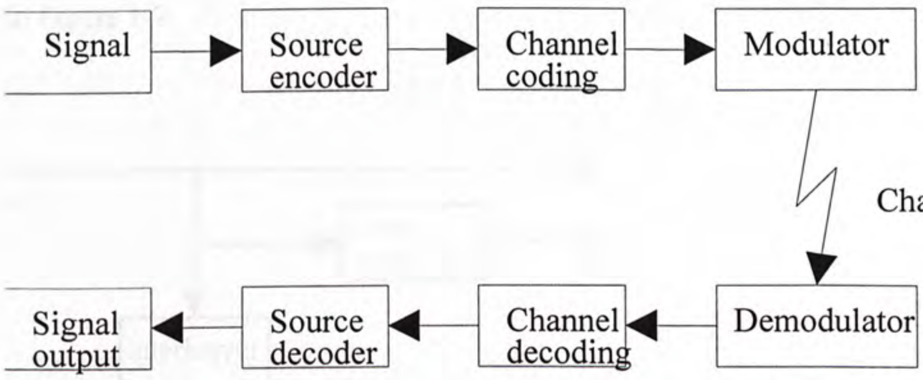


Figure 1-1 – A typical digital communication system

1.2 Soft-output Viterbi Decoder and Turbo Code

Turbo Code is an essential part for next-generation wireless communication systems such as Third-generation (3G) mobile communication systems[1]. It is an error-correcting code introduced in 1993, which gives error recovery to almost at the Shannon's channel limit [2]. There are several popular algorithms for decoding Turbo Codes such as Maximum Aposteriory Algorithm (MAP), Max-log-MAP and Soft-output Viterbi Algorithm (SOVA). SOVA is the least complex decoder among the algorithms mentioned that gives opportunity for low-power applications like mobile handset for 3G systems.

The Soft-Output Viterbi Algorithm [4] is a modification of the classical Viterbi Algorithm [7] that provides a soft output along the decoded bit. The soft output is then used in iterative decoder for decoding turbo codes.

Turbo code is a kind of Parallel Concatenated Convolutional Code (PCCC). The turbo encoder consists of convolutional encoder and a random interleaver as shown in Figure 1-2.

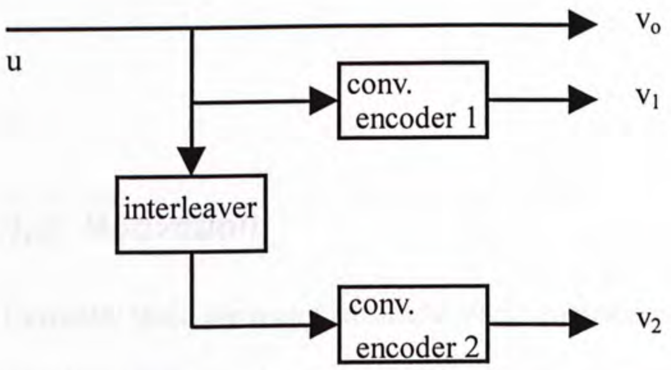


Figure 1-2 – A rate 1/3 Turbo Encoder

1.3 Iterative Decoding

It is possible to use SOVA algorithm to decode Turbo Code. Figure 1-3 shows a block diagram of an iterative Turbo Decoder using SOVA as component decoder [8]. The first decoder produces soft output that is interleaved and used by the second decoder. The second decoder also produces soft output that is needed by the first decoder for next decoding process.

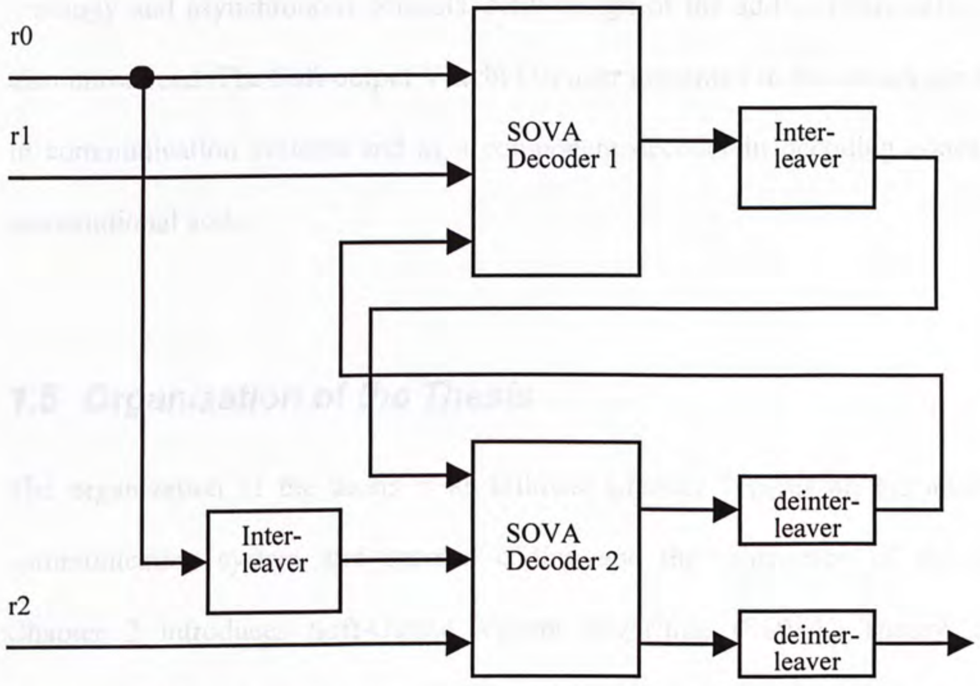


Figure 1-3 – A Turbo Code decoder based on SOVA algorithm

1.4 Motivation

Currently there are many advanced VLSI implementations on the Viterbi Algorithm [21] [22] [23] in which both speed and power consumption is considered. Recently, there are also some implementations on SOVA Decoder [15] [24], in which speed is emphasized. There is an implementation [25] using special layout RAM for traceback memory to achieve low power. Particularly, there is also a VLSI implementation of Viterbi decoder with asynchronous circuit methodology [26], however the power consumption is very high in this implementation

This thesis introduces a new asynchronous traceback memory to reduce power consumption by reducing data movement in the memory with a memory of ring topology and asynchronous controls. New design of the add-compare-select unit is also introduced. The Soft-output Viterbi Decoder presented in this thesis can be used in communication systems and as a component decoder in decoding concatenated convolutional code.

1.5 Organization of the Thesis

The organization of the thesis is as follows: Chapter 1 gives an introduction on communication system and channel coding and the motivation of this project. Chapter 2 introduces Soft-Output Viterbi Algorithm (SOVA) Theory and the decoding algorithm. Chapter 3 describes the asynchronous circuit design methodology adopted in this thesis. Chapter 4 describes the detailed design and implementation of the Asynchronous Soft-output Viterbi Decoder. Chapter 5 presents the Experiment results, comparison and discussion based on the design of the SOVA decoder. The conclusion of the thesis is given in the last chapter.

Chapter 2 Self-timed Circuit Design Methodology

Asynchronous circuit design differs from synchronous counterparts in eliminating all the clocks. Generally, in asynchronous circuits, data are passed through each processing unit controlled by handshakes rather than relying on a global clock. Based on handshaking protocols, asynchronous circuits can be divided into Bundled-data and completion-detection protocol.

2.1 Properties of Self-Timed Design

Self-timed circuits have been appeared since the vacuum tube era [3] and such circuits operate at its natural maximum speed, where performance depends on actual operating condition rather than worse case delay. Self-timed logic passes data from one stage to another by handshakes, which can be completion-detection signals. Unlike its synchronous counterpart, which a global clock controls all the data propagation, self-timed circuits pass information on data change locally, eliminating the clock skew problem and power-dissipation of the clock signal. However, self-timed circuits are more difficult to design due to lack of dedicated design automation tools, and sometimes there is also an area overhead in such circuits.

Self-timed circuit are usually implemented with dynamic circuits such as Differential Cascode Voltage Switch Logic (DCVSL) in which dual-rail encoding is used to detecting completion. However, DCVSL is inherently very power consuming due to the precharge-evaluate phase of every logic gate. Moreover, dynamic logics are not compatible with standard cells in most CMOS processes. Full static CMOS is preferred as it can be implemented by standard methodologies and easier to test than dynamic logics.

Self-timed or asynchronous circuits also give out lower electromagnetic emission than synchronous counterpart [5] that is preferable in many applications such as mixed-signal systems for reliability and lower interference. Another advantage of self-timed systems is that the supply voltage can vary to adopt different data rate easily. This implies we can lower the supply voltage when the data rate is low to gives extra saving of energy. This scenario can arise in mobile communication systems where the bit rate of wireless channels may vary with the handset. Figure 2-1 shows an example of voltage scaling in self-timed circuit with synchronous environment which is given in [6]. The system detects if the input FIFO is nearly empty, the speed of the self-timed circuit is higher than enough and the supply voltage can be reduced. On the other hand, the supply voltage has to be increased when the FIFO is nearly full to accommodate higher data rate.

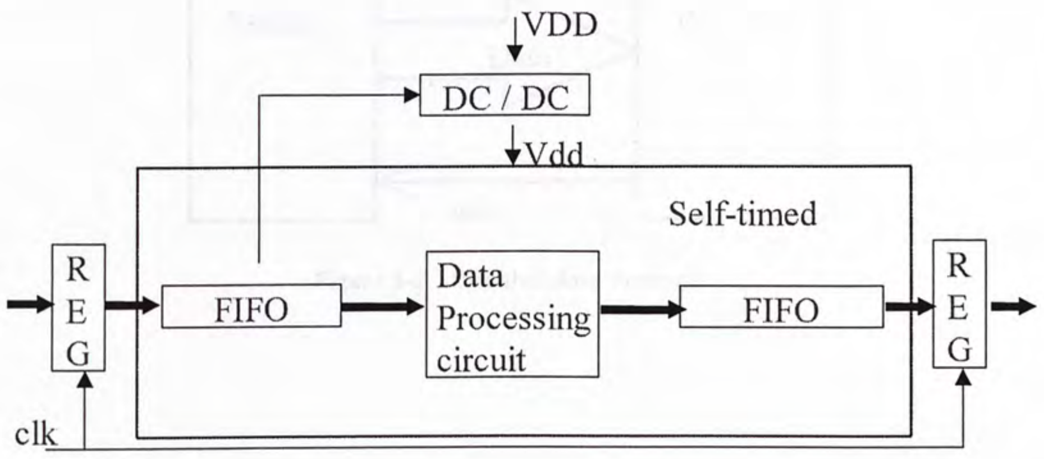


Figure 2-1 – An example of voltage scaling in self-timed circuit with synchronous environment

For self-timed design, there are mainly two design methodologies, namely bundled-data and completion-detection.

2.2 Bundled-data Protocol

In bundled-data protocols, the data are represented by normal Boolean values accompanied by separate *request* (req) and *acknowledge* (ack) signals. The sender puts the valid data onto the bus and then releases a *req* signal to indicate the data are ready. Then the receiver accepts the data and sends back an *ack* signal to tell the sender that it has received the data. When the sender receives the *ack* signal from the receiver, the sender is then ready to start another communication cycle.

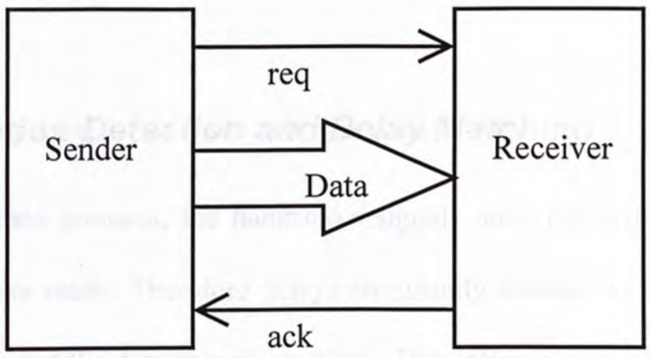


Figure 2-2 – Bundled-data Protocol

2.3 Two-phase verses Four-phase Handshaking

The protocol used in handshaking can be 4-phase or 2-phase. The 4-phase protocol uses logic level to indicate the data validity and the 2-phase protocol uses logic transition to indicate the validity. Figure 2-3 shows the timing diagram of the two protocols. The dotted arrow indicates the linkage between data and handshakes. The 4-phase protocol needs a return-to-zero phase that is similar to the clock in synchronous circuits.

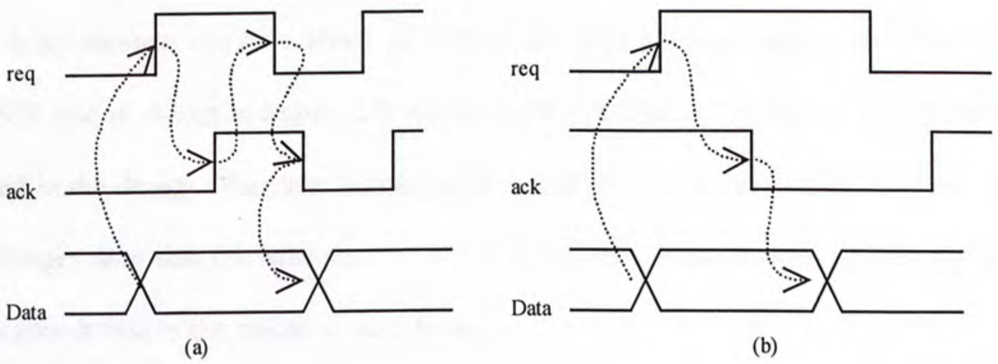


Figure 2-3 – Timing Diagram of (a) 4-phase handshake and (b) 2-phase handshake

2.4 Completion-Detection and Delay Matching

In the bundled-data protocol, the handshake signals must not arrive to next block before the data are ready. Therefore delays are usually inserted to match the timing of the handshake and the data processing logic. The delays are usually fixed, but it is also possible to insert delays controlled by external means [13].

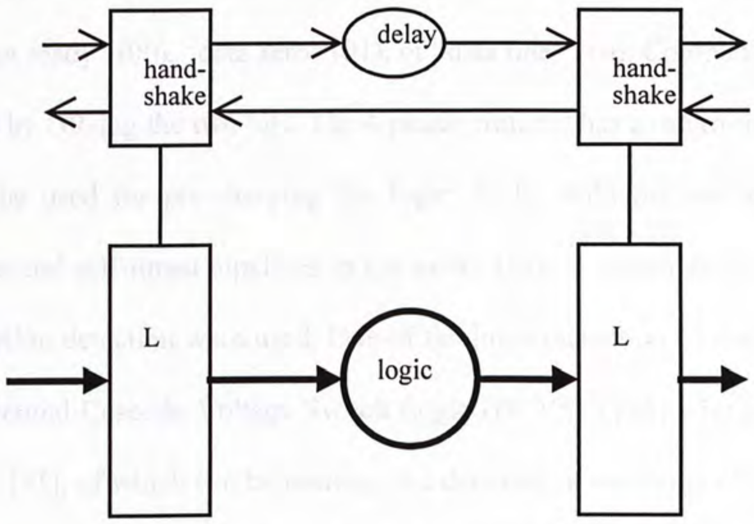


Figure 2-4 – Bundled-delay model

A delay element can be a chain of inverter or AND gates. A delay block built with AND gate is shown in Figure 2-5 which is adopted from [40]. This delay element is used in the design. The time for propagating the change "0 to "1 from A to Q (t_{01}) is longer than that the time for "1 to "0 (t_{10}). This is suitable in 4-phase protocol because it makes the return-to-zero faster.

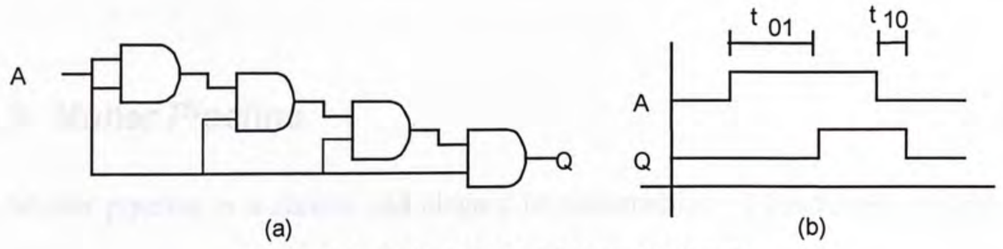


Figure 2-5 – Delay block built from AND gates

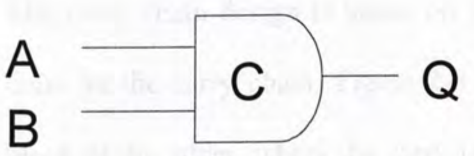
In 4-phase dual-rail protocol, the *request* signal is encoded into the data bus. As shown in Table 2-1, a bit of data is represented by two-bit wire (dual rail), indicating the bit is "not ready" (00), "data zero" (01), or "data one" (10). Completion detection can be done by OR-ing the two bits. The 4-phase protocol has a return-to-zero phase, which can be used for pre-charging the logic. T. E. Williams has analyzed and improved several self-timed pipelines in his works [12], in which dual-rail pipelines with completion detection were used. One of the implementation of this encoding is using Differential Cascode Voltage Switch Logic (DCVSL) [28] with novel control circuits [29] [31], of which can be running at a data rate in the range of GHz [30]. In order to use dynamic logic, a pre-charge period between successive evaluation periods is required. This encoding is robust and fast, but causing 100% hardware overhead and large power consumption.

	Rail 0	Rail 1
Not ready	0	0
Data '0	0	1
Data '1	1	0
Unused	1	1

Table 2-1 – Dual-rail Encoding of a bit

2.5 Muller Pipeline

A Muller pipeline is a simple and elegant implementation of handshake control. It mainly consists of C-elements and inverters. A C-element is a state-holding device that changes its state (output) only when all the inputs are at the same logic level. It holds its previous state if the inputs differ. There are many possible implementation of the C-element and are investigated in [19].



A	B	Q_n
0	0	0
0	1	Q_{n-1}
1	0	Q_{n-1}
1	1	1

Figure 2-6 – C-element and its logical behaviour

The Muller pipeline can be viewed as both 2-phase or 4-phase handshake protocol. Also, the circuit can operates in reverse direction by swapping the *ack* and *req* signals. Lastly, the Muller pipeline is delay-insensitive. This means the circuit works correctly regardless the delays in gates and wires.

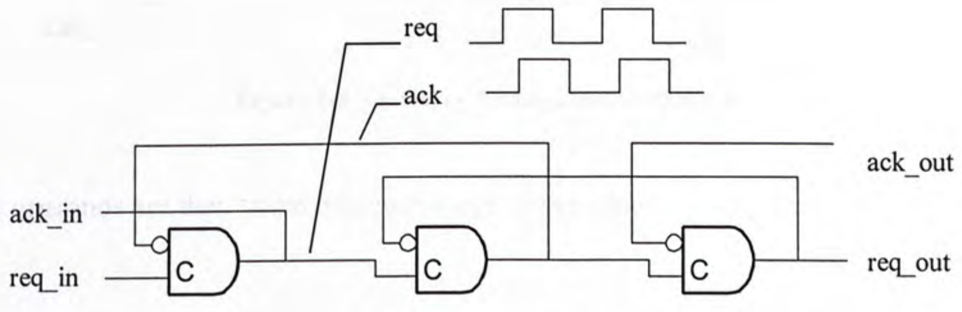


Figure 2-7 – Muller Pipeline

2.6 Design of the Adder

2.6.1 Basic Structure

The basic idea of the adder is to calculate the sum and carry independently. The carry chain is the critical path of an adder; the sum can be calculated in parallel while the carry has to propagate from one stage to another.

2.6.2 Carry Chain and Completion Detection

The carry chain design is based on [3] and [35], in which completion detection is done on the carry chain. Figure 2-8 shows the structure of the Carry Propagation block of the adder, where the carry is dual-rail encoded for completion-detection of each adder stage. The carry chain across the adder is represented by dual-rail signals as in Table 2-2. In practice, the CPs are implemented with complex gates.

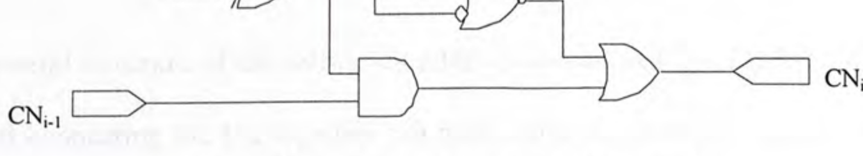


Figure 2-8 - A Carry Propagation (CP) Block

The operands are then taken into each stage of the adder.

Co	CoN	Status
0	0	Not Ready
0	1	Carry '0'
1	0	Carry '1'
1	1	Not used

Table 2-2 - Dual-rail Encoding of Carry Chain

The carry and its complement of each stage are computed independently along the carry chain as in (1) and (2).

$$C_i = AB + C_{i-1}(A \oplus B) \qquad (1)$$

$$CN_i = \sim(A+B) + CN_{i-1}(A \oplus B) \qquad (2)$$

One can observe that, the carry out of an adder stage can be computed independently from previous stage if A equals B. Otherwise, the carry out has to be determined by the carry of previous stage. This dual-rail implementation can give completion-detection of each stage by taking OR of C_i and CN_i. When one of the operands is always '0', which may happen when adding two data with unequal width, the carry chain can be simplified as $C_i = AC_{i-1}$ and $CN_i = (\sim A) + CN_{i-1}$ which is only a one-gate implementation.

The general structure of the self-timed adder is shown in Figure 2-9. Cascading the CP and connecting the FA together can make adder of arbitrary length. The FA in each adder stage is a 3-input exclusive gate, which adds the two one-bit operands and the carry from previous stage. Its delay should be less than the generation of the completion-detection signals of the whole adder that is about 3-gate delay.

i

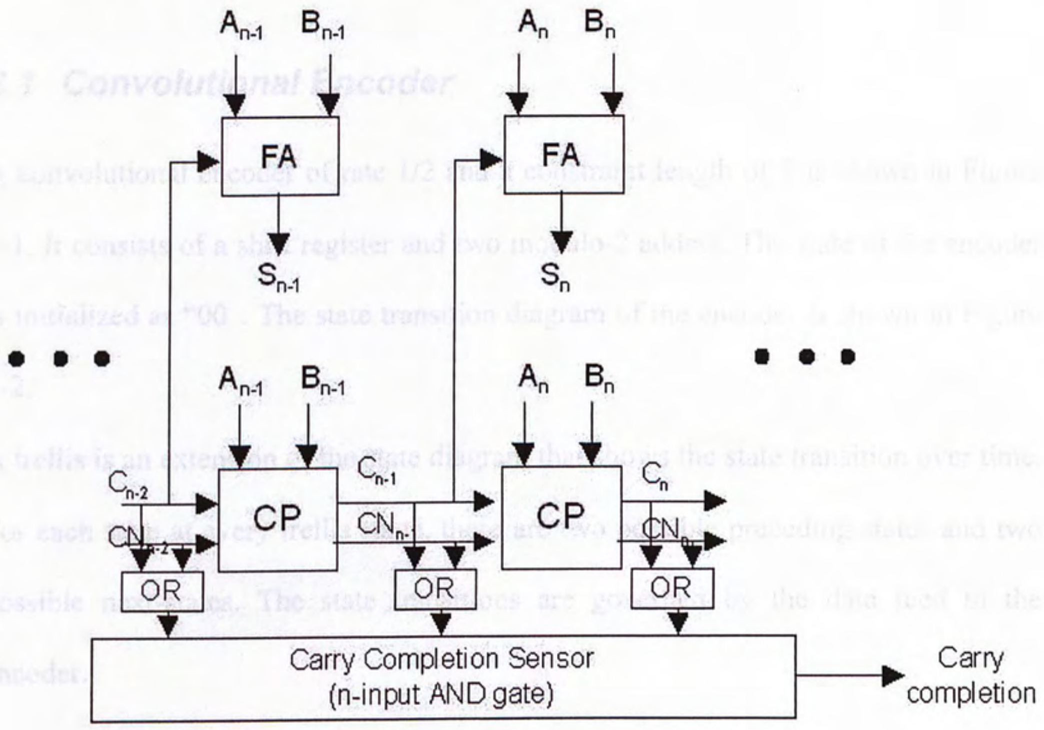


Figure 2-9 - Architecture of the Self-Timed Adder

Chapter 3 SOVA Theory

3.1 Convolutional Encoder

A convolutional encoder of rate 1/2 and a constraint length of 3 is shown in Figure 3-1. It consists of a shift register and two modulo-2 adders. The state of the encoder is initialized as "00 . The state transition diagram of the encoder is shown in Figure 3-2.

A trellis is an extension of the state diagram that shows the state transition over time. For each state at every trellis stage, there are two possible preceding states and two possible next-states. The state transitions are governed by the data feed to the encoder.

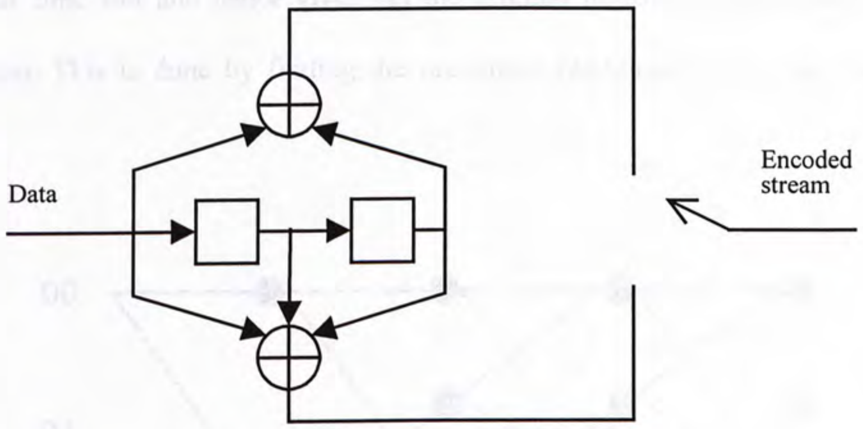


Figure 3-1 – A rate 1/2 convolutional encoder

Data are shifted into the encoder and it generates two bits for every information bits. Hence the rate of the encoder is 1/2. The constraint length of an encoder K equals v+1 bits when v is the memory of the encoder. Therefore the constraint length of the encoder in Figure 3-1 is 3.

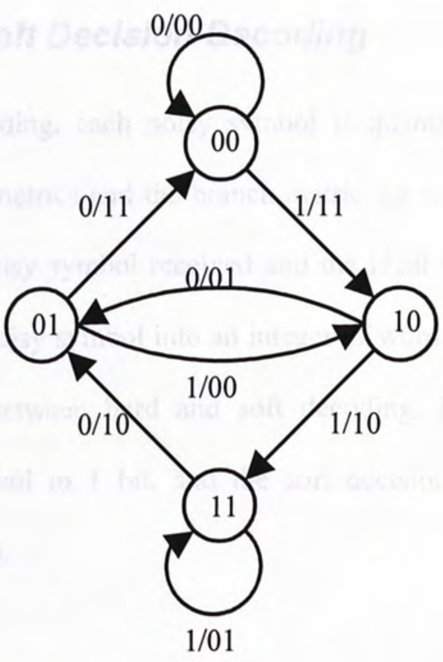


Figure 3-2 – State Diagram of the encoder in Figure 3-1.

At the receiver side, the decoder find the most possible state of the encoder at a particular time slot and hence gives out the original information bits based on the estimation. This is done by finding the maximum likelihood (ML) path along the trellis.

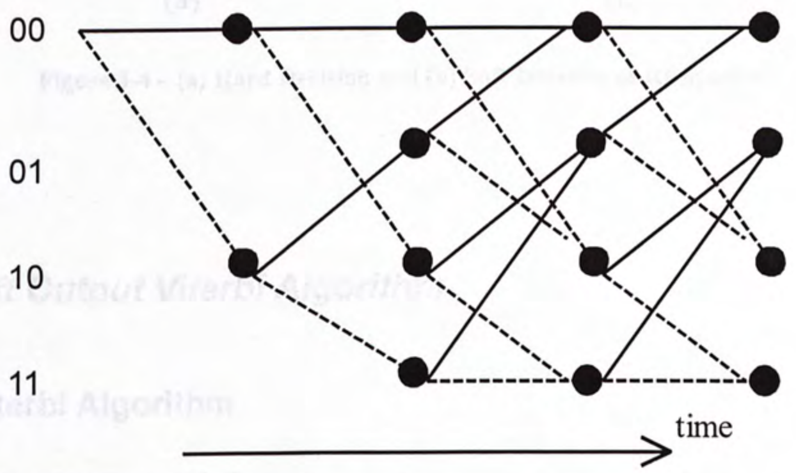


Figure 3-3 – Trellis Diagram for the encoder in Figure 3-1

3.2 Hard verse Soft Decision Decoding

For hard decision decoding, each noisy symbol is quantized into one bit before calculating the Branch metrics and the branch metric for each state is the hamming distance between the noisy symbol received and the ideal symbol. In contrast, soft decision quantizes the noisy symbol into an integer of wordlength w bits. Figure 3-4 shows the difference between hard and soft decoding. Hard decision decoding digitalizes the RF symbol to 1 bit, and the soft decision decoding converts the symbol into a 3-bit value.

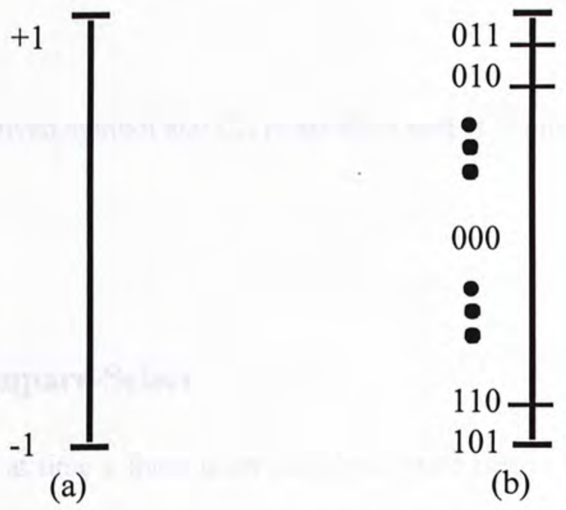


Figure 3-4 – (a) Hard Decision and (b) Soft Decision of RF symbol

3.3 Soft Output Viterbi Algorithm

3.3.1 Viterbi Algorithm

The Viterbi Algorithm [7] finds the most possible sequence of binary codes of a convolutional encoder sent over a noisy channel. The algorithm recursively finds the survivor path over the trellis which is the closest path to the received sequence of symbols over the channel. The received symbols may be noisy and may not give a valid path over the trellis. The decoding algorithm can be divided into three steps.

3.3.1.1 Branch Metric Computation

The Branch Metric is the distance between the received symbol and all the possible transition among the states in the trellis. It is the squared distance between the noisy symbol and the ideal symbol of the transitions. Mathematically,

$$BM_{i,j,n} = (y_n - C_{i,j})^2 \qquad \textbf{Equation 3-1}$$

where y_n is the received symbol and $C_{i,j}$ is the ideal output symbol of transition from state i to state j.

3.3.1.2 Add-Compare-Select

For each state $S_n^{(k)}$ at time n there is an associated state metric $M(S_n^{(k)})$. The two branch metrics of the incoming path is added to the state metric individually and the results are compared. The one with smaller metric is then selected as surviving branch and the other one is discarded.

$$M(S_n^{(k)}) = \min\{S_{n-1}^{(k)} + BM_{i1,k,n} , S_{n-1}^{(k)} + BM_{i2,k,n} \} \qquad \textbf{Equation 3-2}$$

For each state in the trellis this calculation is performed and the results are used in the traceback

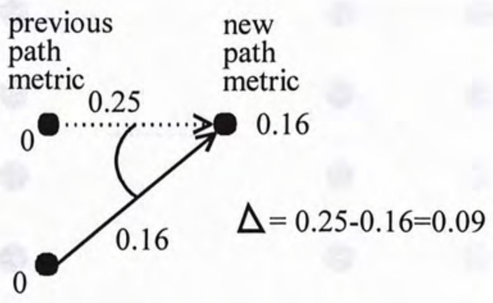


Figure 3-5 – Add-Compare-Select Example

The difference between the path metrics of the surviving and discarded path, Δ, is also saved for the soft output decoding as discussed later.

3.3.1.3 Decoding Symbols

Figure 3-6 shows a simplified four state trellis with the decisions calculated as in the previous section. There are no know start and end states in the trellis, as in most digital communication applications. At time k, there are four paths that are all probable correct path on the trellis for the received symbols. These paths are traced along the trellis with the decisions calculated and they will merge into a single state after a certain steps of traceback [9]. The length for the paths to merge, D, is called survivor depth. With this property, it is sufficient to perform traceback from only one state at time k, and the survivor path will gives the decoded bit at time k-D.

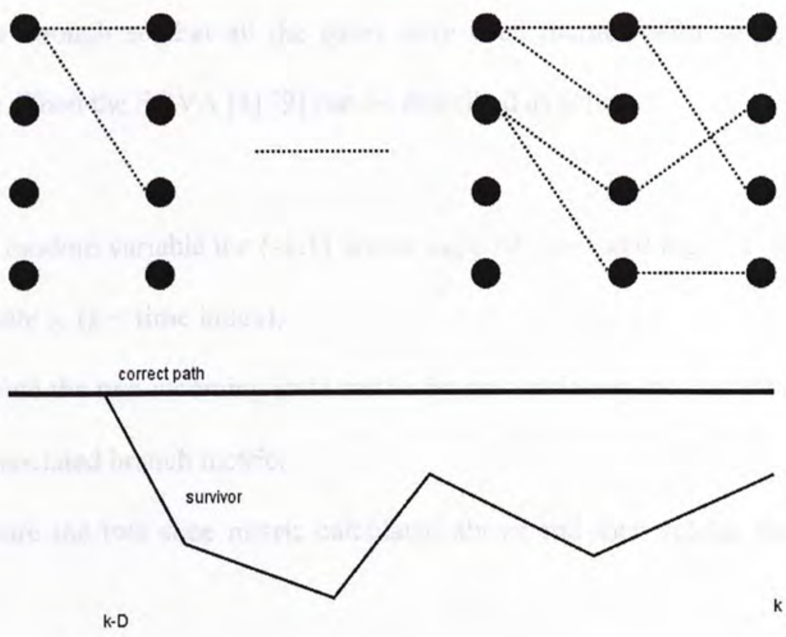


Figure 3-6 – Finding the survivor path

The survivor path may not be the correct path as shown in the figure. However, the property that all paths will merge after D traceback steps makes the decoding of an information bitstream possible in a fixed latency D.

3.3.2 Soft Output Algorithm

The Soft Output Viterbi Algorithm (SOVA) described in this section was proposed by Joachim Hagenauer and Peter Hoeher [4]. It is a modification of the Viterbi algorithm where it provides the reliability measure (soft output) together with a decoded bit. The soft output is needed for component decoders for decoding concatenated convolutional codes such as Turbo Code [2].

For simplicity we assume a rate of 1/N convolutional code below. The number of states S is 2^v, where v is the code memory. Further we assume that the traceback length δ is enough so that all the paths have been merged with sufficiently high probability. Then the SOVA [4] [9] can be described as follows.

For binary random variable u є {-1,1} where logic '0 is –1 and logic '1 is 1.
For each state s_k (k = time index),

- Compute the two incoming state metric by accumulating the current metric with the associated branch metric.
- Compare the two state metric calculated above and then selects the minimum one.
- Store the minimum metric found from previous step.
- Store Δ equals the absolute difference between the two incoming state metric for soft-deciding update in following stage.

This reliability measure (soft-output) is given out in a log-likelihood ratio (LLR) $L_c(u_k)$ by

$$L(u_k) = \log_e\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \qquad \textbf{Equation 3-3}$$

where $P(u_k = +1)$ is the probability that $u_k = +1$ and $P(u_k = -1)$ is the probability that $u_k = -1$

For two metric $M(s_k^s)$ and $M(\hat{s}_k^s)$ of the two path entering states s_k^s and $\hat{s}_k^s$ at time *k*, the metric difference is defined as

$$\Delta_k^s = M(s_k^s) - M(\hat{s}_k^s) \geq 0$$ **Equation 3-4**

The probability that the decision is correct when selecting path s_k^s and discarding path $\hat{s}_k^s$ is then [14]

$$P(S_k = s) = \frac{e^{\Delta_k^s}}{1 + e^{\Delta_k^s}}$$ **Equation 3-5**

and the LLR is simply Δ_k^s. To determine the LLR of the decoded bit, the Soft Output Viterbi algorithm takes account on all Δ_k^s along the maximum likelihood path (ML path) for all the traceback stage. This LLR can be approximated [9] by:

$$L(u_k) \approx u_k \min_{\substack{i=k\ldots k+\delta \\ u_k \neq u_k^i}} \Delta_i^{s_i}$$ **Equation 3-6**

L(u$_k$) is the LLR of decoded bit u$_k$ at time k. u_k^i is the value of the decoded bit of the discarded path at trellis stage *i* if it had been selected as survivor.

The minimization in Equation 3-6 is carried out for the path merging to the maximum likelihood path at u$_k$ but gives out different decision as the ML path does. The same hard decisions as the classical Viterbi Algorithm is obtained and the reliability of the decisions is obtained by taking the minimum of the metric differences along the maximum likelihood path at where the decision of the maximum likelihood path and the merging path differs.

Here is an example:

Figure 3-7 shows a simplified trellis of four states and a traceback length 5. The path difference between the surviving path and the discarded path (not shown) is also calculated and stored. In the figure, the solid line represents the final survivor and its competing path. The decisions along the paths are also shown in the figure.

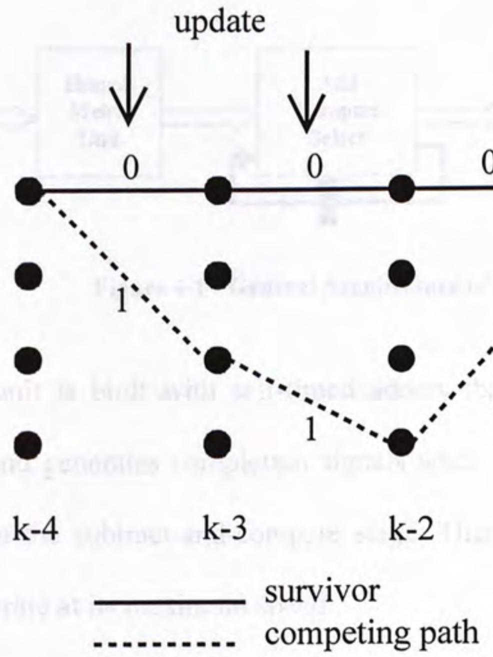


Figure 3-7 – Soft update example

The soft update process begins when the algorithm has identified the final survivor. It can be seen from the figure that the two paths gives different decisions at t = k-2 and k-3. Only the path difference along the final survivor in which the decision is different from the competing path is updated.

Chapter 4 Proposed SOVA Decoder Design

4.1 Overview

The general architecture of a SOVA decoder is shown in Figure 4-1. The bottleneck that limits the speed of the whole decoder is the Add-Compare-Select (ACS) unit because it is a nonlinear operation. The ACS block needs to add pairs of path and state metrics, compare the results and select the path with smaller metric. It also has to output the differences between the two incoming state metric for soft-output in path metric unit.

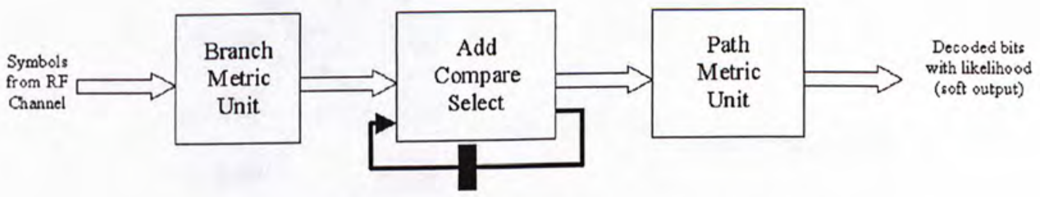


Figure 4-1 - General Architecture of a SOVA decoder

The ACS unit is built with self-timed adders that compute the carry only when necessary and generates completion signals when calculation is done. The adder is also used in the subtract-and-compare stage. Therefore the ACS unit is self-timed and can operate at its maximum speed.

4.2 SOVA Decoder Architecture

As discussed in §3.3, the operation of the SOVA decoder is based on finding the maximum likelihood path along the trellis over time. Ideally the time for finding the maximum likelihood path should be infinity long that is not possible in real implementation. Previous findings [34] show that tracing back length of 5 times the constraint length is enough for most applications.

The implementation of the SOVA decoder is based on a trellis of 8-state as in Figure 4-2, corresponding to an encoder with generating polynomials G_0= $1+D^2+D^3$, G_1= $1+D+D^3$, G_2= $1+D^1+D^2+D^3$,

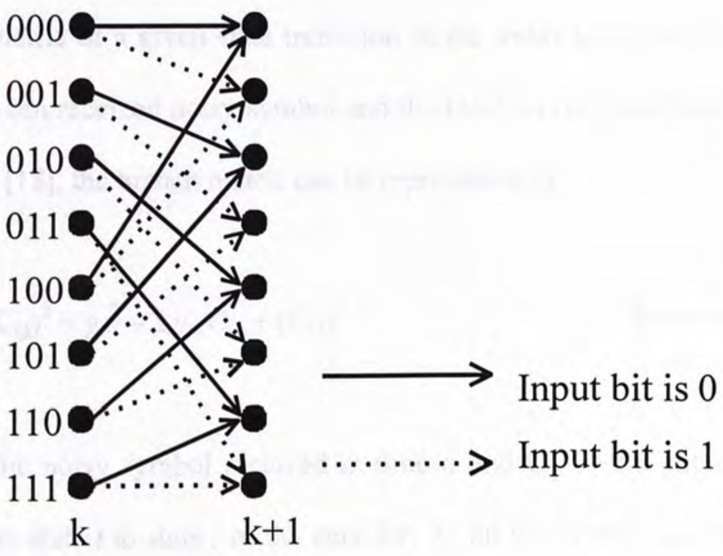


Figure 4-2 – The Trellis that the SOVA decoder based on.

The Branch Metric Unit generates the metrics of the incoming symbol against all the states of the trellis at each time slot. The smaller the branch metric associated to a state, the higher the probability the symbol is corresponding to that particular state.

The ACS unit calculates the accumulative state metric of all the state in the trellis at a time slot by adding the branch metric and the previous state metric for the two incoming paths of a state. The path with smaller metric (the survivor) is saved for next iteration. The difference between the survivor and the discarded path is fed to the Traceback unit for soft update.

4.3 Branch Metric Unit

4.3.1 Branch Metric Generation

The branch metric of a given state transition in the trellis is defined as the squared distance between received noisy symbol and the ideal output symbol of the encoder. According to [18], the branch metric can be represented by

$$BM_{i,j,n} = (y_n - C_{i,j})^2 = y_n^2 - 2\, y_n\, C_{i,j} + (C_{i,j})^2 \qquad \textbf{Equation 4-1}$$

where y_n is the noisy symbol received at time *n* and $C_{i,j}$ is the output symbol for transition from state *i* to state *j* of the encoder. As all the branch metrics contain the y_n^2 term, it can be eliminated without altering the difference among the branch metrics. Therefore

$$BM_{i,j,n} = -2\, y_n\, C_{i,j} + (C_{i,j})^2 \qquad \textbf{Equation 4-2}$$

For output binary symbol a and –a, $C_{i,j}$ is in the set {-a, a}. Then

$$BM_{i,j,n} - BM_{k,j,n} = \begin{cases} 0 & \text{if } C_{i,j} = C_{k,j} \\ -2y_n C_{i,j} + 2y_n C_{k,j} & \text{if } C_{i,j} \neq C_{k,j} \end{cases} \qquad \textbf{Equation 4-3}$$

Since dividing all branch metric by a constant does not alter comparison results, the branch metric can be further simplified to:

$$BM_{i,j,n} = -y_n C_{i,j} = \begin{cases} -y_n & \text{if } C_{i,j} = a \\ y_n & \text{if } C_{i,j} = -a \end{cases}$$ **Equation 4-4**

For a rate 1/3 decoder with soft decision input y_0, y_1 and y_2, the branch metric for the state transition producing output (1,1,0) is $(-y_0) - y_1 + y_2$.

4.3.2 Implementation

The structure of the branch metric is shown in Figure 4-3. The minus sign indicates negation is taken before addition. For 3-bit soft decision, 5-bit is needed for the branch metrics. The adders are the self-timed adders discussed in §2.6.

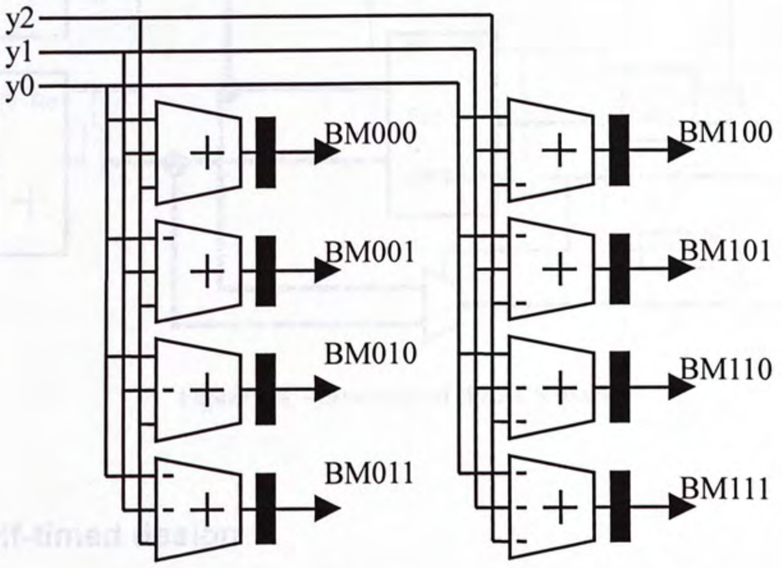


Figure 4-3 – Architecture of the Branch Metric Unit (BMU)

4.4 Add-Compare-Select Unit

4.4.1 Basics

The basic structure of the add-compare-select block is shown in Figure 4-4. The adders and subtractors are built with the self-timed adder described in previous section. Light lines represent handshake controls while heavy lines stand for data flow. The C-element in between waits for the two adders to complete calculations and then propagate the request to the subtractor-comparator. Registers are placed in-between adders and subtractor-comparator to reduce glitch-propagation. Transistor level simulation shows a 7% power saving with the register added. Matched-delays are used to ensure the timing of handshake signals behaves correctly.

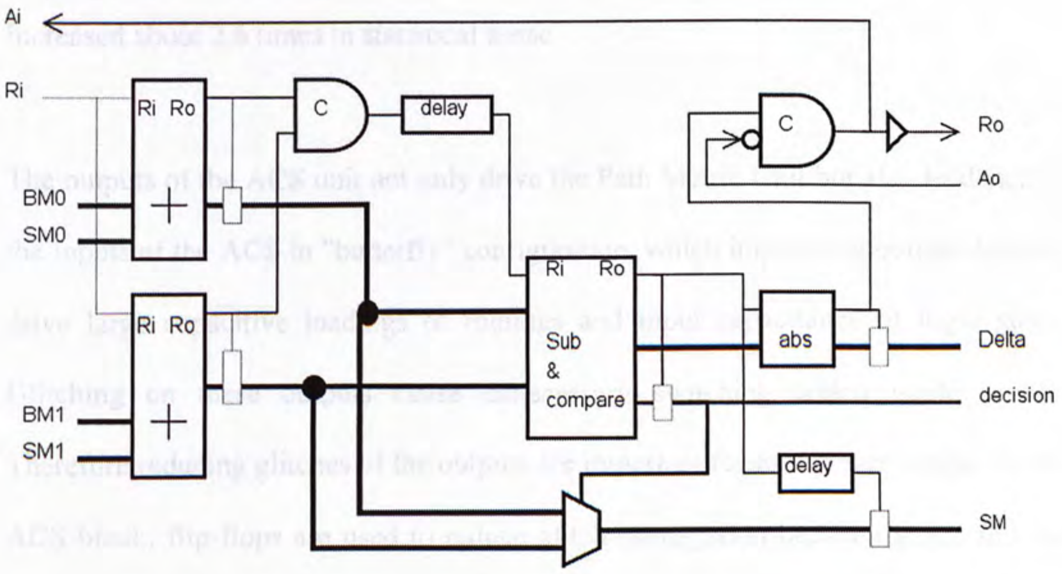


Figure 4-4 - Structure of the ACS block

4.4.2 Self-timed design

As mentioned in section 3.3, the output of an ACS unit have to feed back to its input for calculations of next state-metric. Therefore its speed limits the performance of whole decoder. The critical path of the adder is the carry chain. For wide operand, algorithms like carry-lookahead is suitable to reduce the length of carry chain. However, it is not efficient for small operand (8-bit here) as the area overhead is relatively large.

The expected value of the length of the carry chain of an adder is $\log_2 N$ [32]. The self-timed adder takes the advantages of propagating the carry only when necessary and therefore its average length of carry chain is also $\log_2 N$. For the synchronous counterpart, ripple carry adder is usually used and the designer has to tolerate the longest carry chain which is N bit long. For an 8-bit adder, the speed can be increased about 2.6 times in statistical sense.

The outputs of the ACS unit not only drive the Path Metric Unit but also feedback to the inputs of the ACS in "butterfly" configuration, which implies the outputs have to drive large capacitive loadings of routings and input capacitance of logic gates. Glitching on these outputs cause unnecessary switching which waste power. Therefore reducing glitches of the outputs are important for low-power design. In the ACS block, flip-flops are used to reduce glitch propagation between adder and the subtract-compare block.

4.4.3 Metric Normalization

Metric normalization schemes are adopted for the ACS unit to ensure the path metric registers not to overflow. There are mainly two methods for normalization, namely *subtracting the minimum state metric* and *modulo normalization.*

For subtracting minimum state metrics, it first determines the minimum state metric of all the states. Then all the ACS processing elements will subtract the minimum metric from its path metric. This method needs global communication between all the ACS processing elements, which is a relatively large computation overhead. Therefore it is suitable for low throughput application or software implementation.

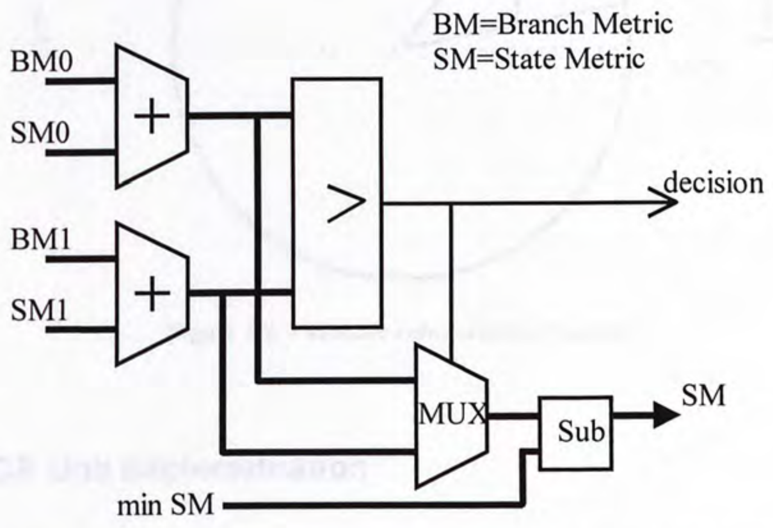


Figure 4-5 – Metric normalization by subtracting minimum state metric

To eliminate the extra comparison and subtraction, we can add one more bit in the path metric registers so that *Modulo Normalization* [17] can be used.

If the maximum possible difference between path metric is D_{max}, then the path metric register is of length $2D_{max}$. Two's complement representation is used for branch and state metric calculation in this case.

This situation can be viewed as two points running on a circle as shown in Figure 4-6. For $| m_1 - m_2 | < D_{max}$, we have $m_1 < m_2$ if and only if $\alpha < \pi$.

Therefore the comparison is based on the 2 s complement subtraction of the two metrics.

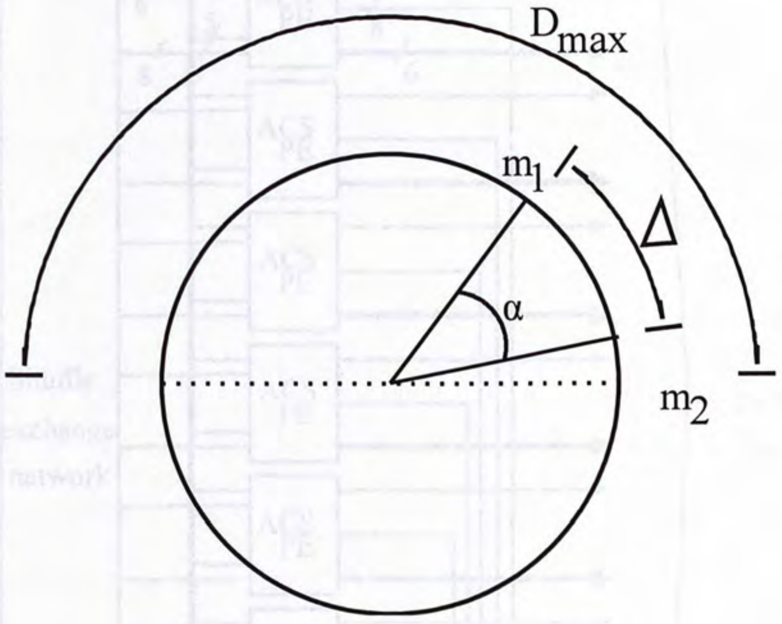


Figure 4-6 – Modulo comparison of metrics

4.4.4 ACS Unit Implementation

The ACS unit adopts state-parallel design, in which one ACS processing element described in section 4.4.2 is mapped to a state of the trellis. The state-parallel design simplified the data flow of the ACS unit and allows adequate processing speed in low-voltage operation in the cost of area. The ACS unit consists of eight ACS processing element which map the eight states of the trellis. Each ACS processing element calculates the state metric of the two incoming path of a state and select the smaller metric as survivor. It also gives out the path metric difference of the two incoming path for each state which is needed to calculate the "soft-output" in the traceback unit.

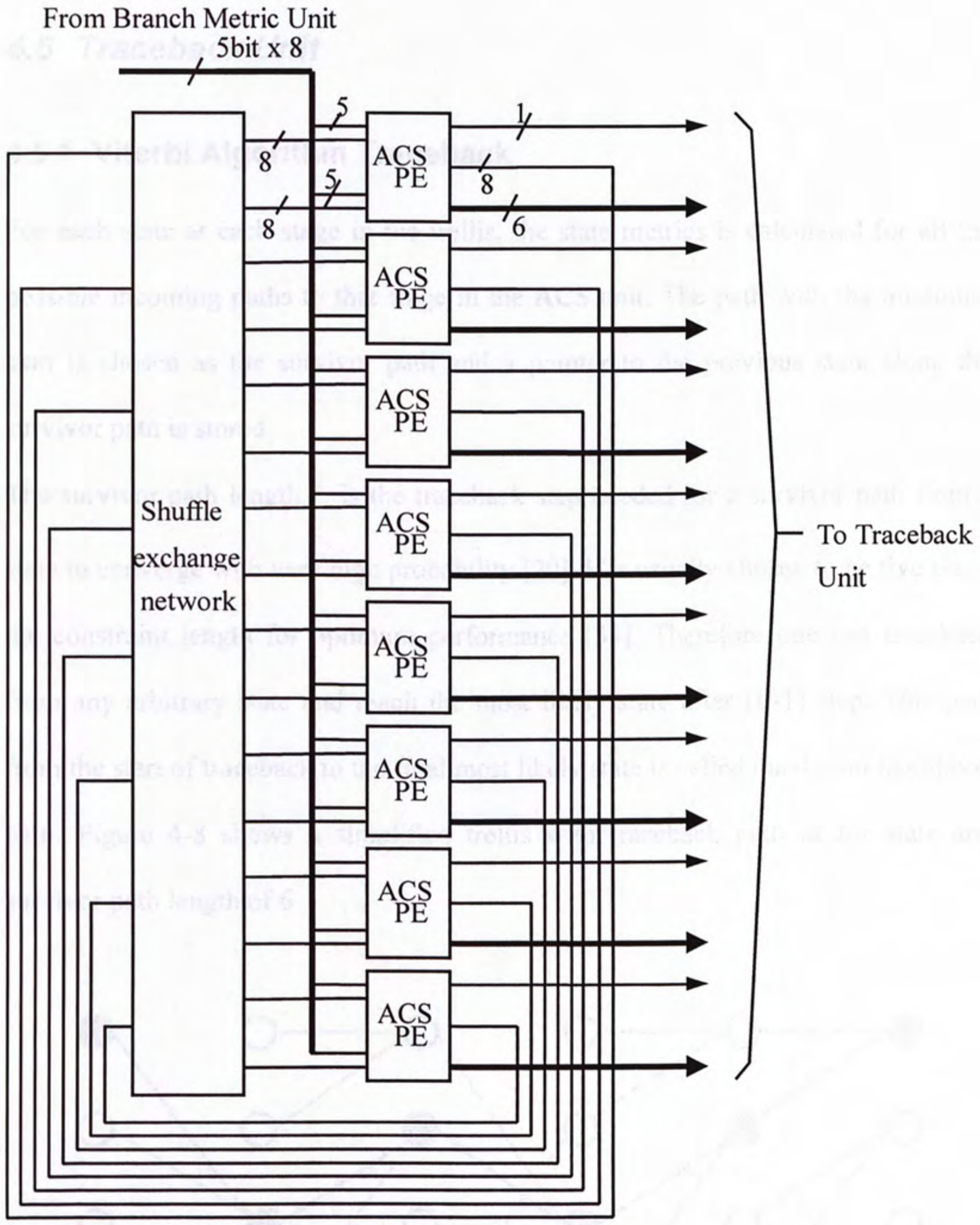


Figure 4-7 – The ACS unit implementation

The routing of the shuffle exchange network in Figure 4-7 corresponds to the topology of the 8-state trellis as in Figure 4-2.

4.5 Traceback Unit

4.5.1 Viterbi Algorithm Traceback

For each state at each stage in the trellis, the state metrics is calculated for all the possible incoming paths to that stage in the ACS unit. The path with the minimum sum is chosen as the survivor path and a pointer to the previous state along the survivor path is stored.

The survivor path length L is the traceback step needed for a survivor path from a state to converge with very high probability [20]. L is usually chosen to be five times the constraint length for optimum performance [34]. Therefore one can traceback from any arbitrary state and reach the most likely state after (L-1) step. This path from the start of traceback to the final most likely state is called maximum likelihood path. Figure 4-8 shows a simplified trellis with traceback path of for state and survivor path length of 6.

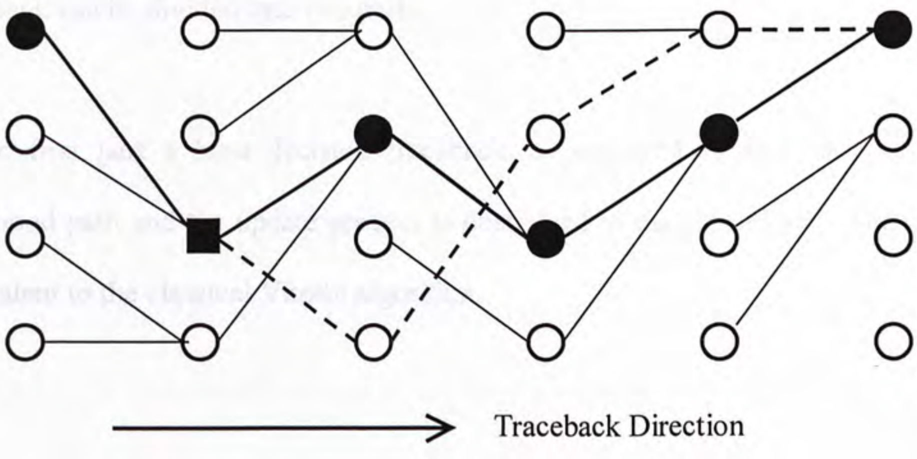


Figure 4-8 – Traceback with error

For an error occurred when receiving the bitstream from the channel (as the square in Figure 4-8). The traceback follows the wrong path (the dotted line). However, the error is corrected by the decision of each individual state in the trellis and reaches the same state as the original path eventually. The decoded bit is hence the same as if all the symbols received are not corrupted.

4.5.2 Two Step SOVA

In the original SOVA algorithm, it is required to construct all the competing paths of all the states and the update procedure have to be carried out along the paths. However, only the final survivor and the associated reliability are outputted.

Therefore, to discard the unnecessary calculation, the two-step SOVA traceback [37] [38] is used to reduce the calculation complexity. In this method, regular Viterbi Algorithm is used to find the maximum likelihood path. The soft update is then carried out over the maximum likelihood path and its competitor. Therefore the traceback can be divided into two parts.

In the first part a hard decision traceback is employed to find the maximum likelihood path and the update process is postponed to the second part. This part is equivalent to the classical Viterbi algorithm.

In the second part, only the competing path against the maximum likelihood path is constructed and the reliability of the final decision is updated with respect to it. Now only a single path comparison is required instead of 2^v path in the original algorithm because the final survivor is known.

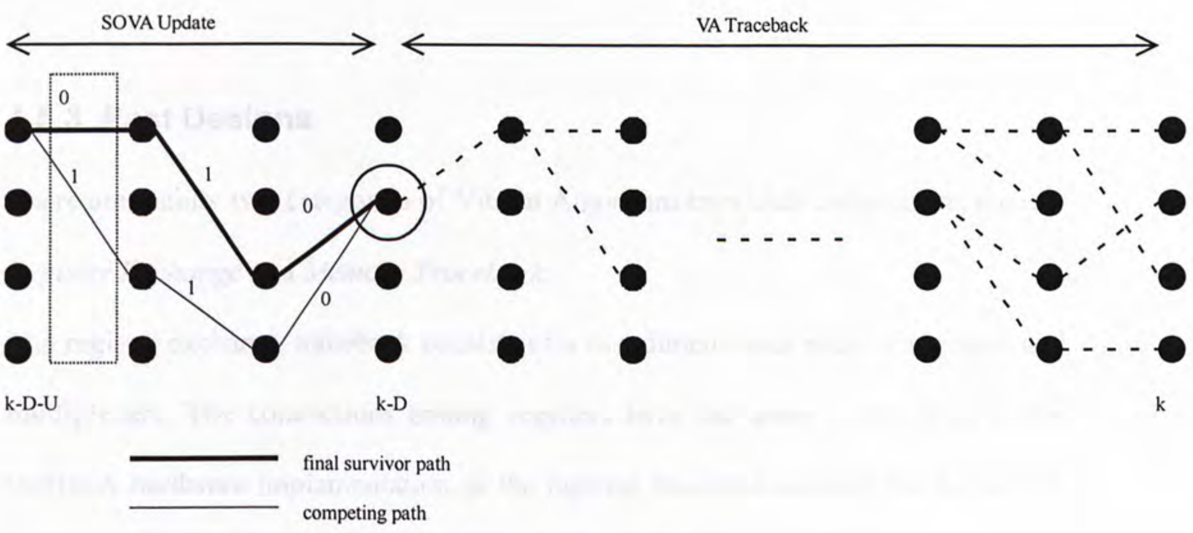


Figure 4-9 – Two Step SOVA Example

In Figure 4-9, the right hand side shown the regular Viterbi Algorithm where the maximum likelihood path is determined. The left hand side shows the comparison between the survivor and the competing path at each step. The reliability is initiated as infinity at step k-D. Then the bit decisions of the two paths at each step are compared. If the decisions differ (i.e. one is '0 while the other is '1), the minimum between the path metric difference of the survivor path at that step and the reliability of previous step is taken as the updated reliability. If the decision equals, the reliability measure is not updated and the value of previous step is used. Therefore, in the example, the reliability update is only performed at the step indicated by the dotted rectangle.

The survivor path traceback length is D while the SOVA update length is U. The length of D is usually five times the constraint length and the length of U can be twice of the constraint length should be optimum [39].

4.5.3 Past Designs

There are mainly two categories of Viterbi Algorithm traceback architecture, namely *Register Exchange* and *Memory Traceback.*

The register exchange traceback consists of a two-dimensional array of registers and multiplexers. The connections among registers have the same topology as in the trellis. A hardware implementation of the register traceback network for a rate 1/2, constraint length of 3 and 4-state decoder is shown in Figure 4-10. The multiplexers are controlled by the corresponding decision bits from ACS unit and the registers are controlled by the global clock.

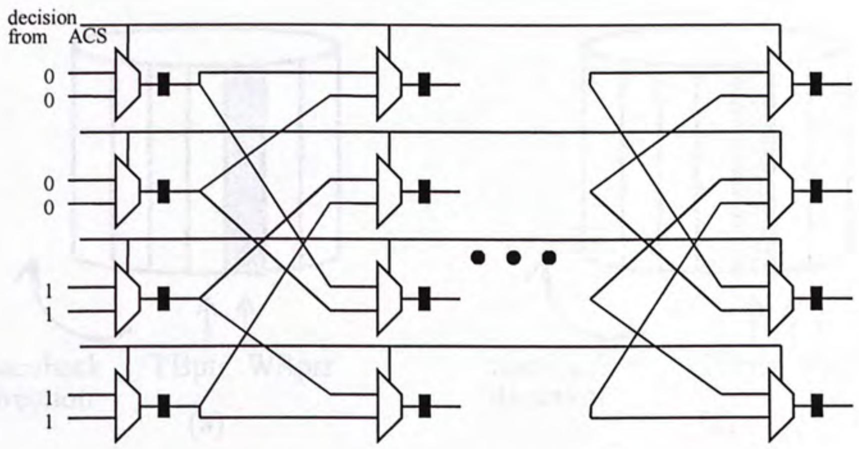


Figure 4-10 – An Example of Register Exchange Traceback

The throughput can be very high in this architecture, in the cost of high power consumption due to high memory bandwidth needed.

For SOVA implementation, the register exchange method is used to find the most likelihood path and then the delayed symbol from ACS unit and the path differences Δ^{s}_{k} are then used to calculate the soft output at the second step. This architecture can achieve high decoding rate as 500 M Symbol/s in a recent implementation [15] using advanced CMOS process. However, it needs much memory for storing the path information and the data are moving along the path, costing high power consumption.

Another traceback architecture is called *Memory Traceback*. In this method, a vector of path decision is written to the survivor memory (RAM) for every iteration of the ACS unit. In other words, each cell in memory array stores a pointer to the previous state. The data are then read back in the opposite direction. One can imagine the survivor memory as a circular ring shown in Figure 4-11. The WRptr is the address to write the vector from ACS. The TBptr is the memory location to start traceback.

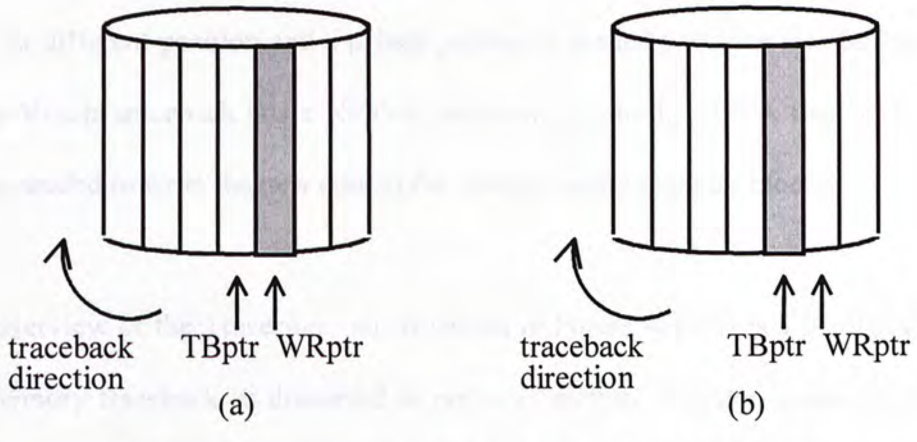


Figure 4-11 – Traceback memory (a) at time k, (b) at time k+1

After one traceback iteration, both the TBptr and WRptr move one memory location opposing the traceback direction, new data are written to the memory pointed by WRptr and new traceback iteration starts over again. Therefore the traceback always starts from the most recently stored data to the oldest data in the memory. This method is slower than the register exchange method, but gives more efficient use of memory and hence lowers power consumption in general. Garrett and Sten use an orthogonal memory (write by rows read by columns) in SOVA traceback [25] to reduce read operations to the memory.

4.5.4 New Traceback Architecture

4.5.4.1 Overview

An architecture where all of the data are stored in memory and do not propagate through a pipeline is considered. The advantage is the majority of the data is static in the memory and does not move around as in register exchange method. For every iteration the oldest data is overwritten by the newest data. However, the traceback starts at different position and a pHead pointer is needed to locate the starting point of the Viterbi traceback and a pSOVA pointer to locate the SOVA traceback, and a bus is needed to write the new data to the corresponding memory block.

The overview of the Traceback unit is shown in Figure 4-12. It is a modification of the memory traceback as discussed in previous section. Register is used instead of RAM as the timing of RAM is undetermined in low-voltage operation for which this asynchronous circuit implementation is not suitable.

The Traceback Unit is composed of identical Traceback cells (TBcell), which are connected as a ring. The handshake topology of the Traceback Unit is borrowed from [26], but the operation is very different. The pHead pointer, generated by a ring counter, will activate one of the TBcell as the first cell of the ring and the cell just before it in the ring becomes the last cell of the ring. Data from the ACS unit are written to the first cell where information for current time slot is saved. The memory used are registers instead of RAM, which eliminates address decoding and undetermined behaviour of RAM at low voltage.

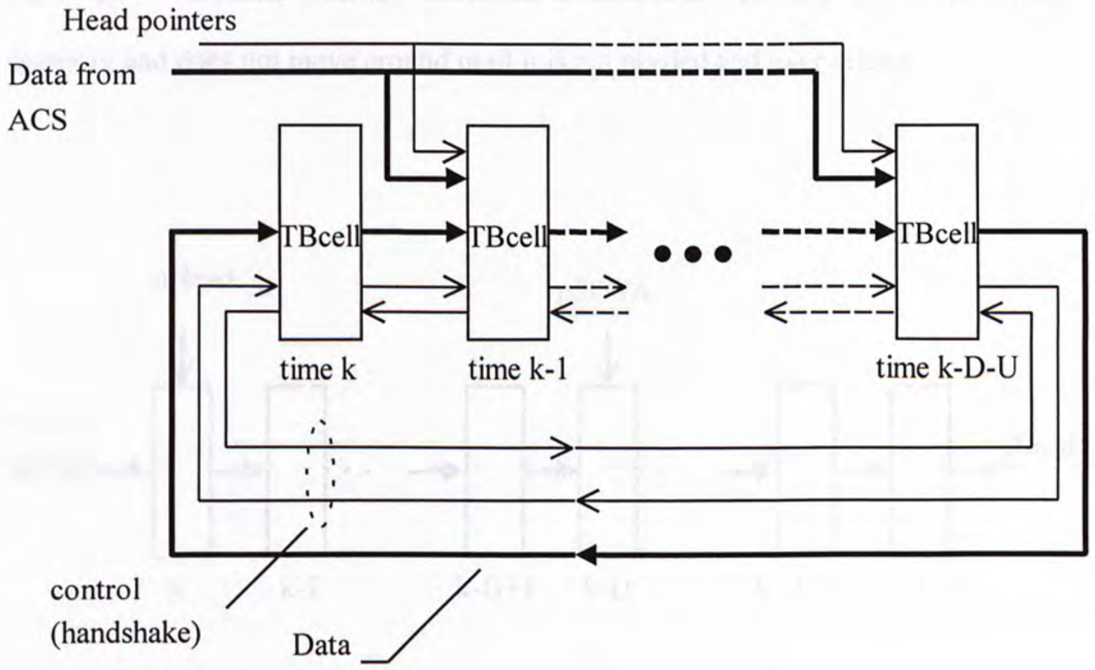


Figure 4-12 – Overview of the Traceback unit

The traceback will go through each TBcell by each TBcell. Finally the traceback will reach the first cell and then the traceback is completed. Then the Traceback Unit will generate a "Ro" signal indicating that the decoded bit and the reliability information are ready. It will hold the decoded information until it receives the acknowledge signal "Ao" from outside the decoder.

4.5.5 Traceback operation

In Figure 4-13 and Figure 4-14, the rectangles represent the storage for the decision and path metric differences from ACS unit. These storages are connected in a ring topology to resemble memory traceback architecture. The data are stored in the memory and does not move around until it is not needed and overwritten.

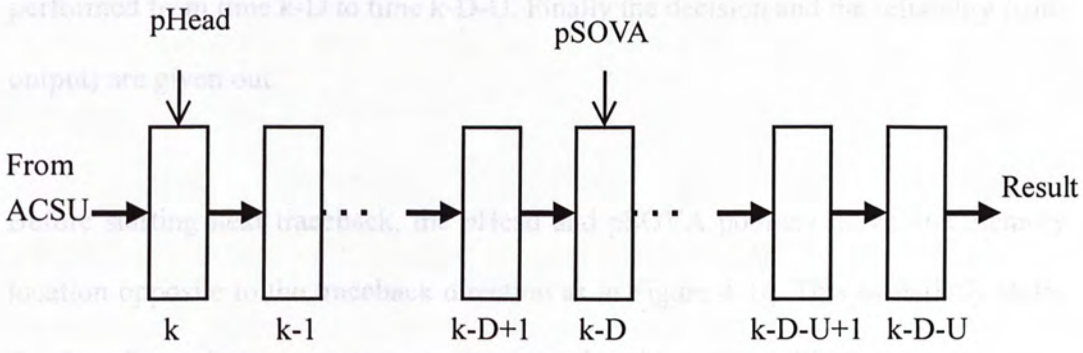


Figure 4-13 – Traceback at time j

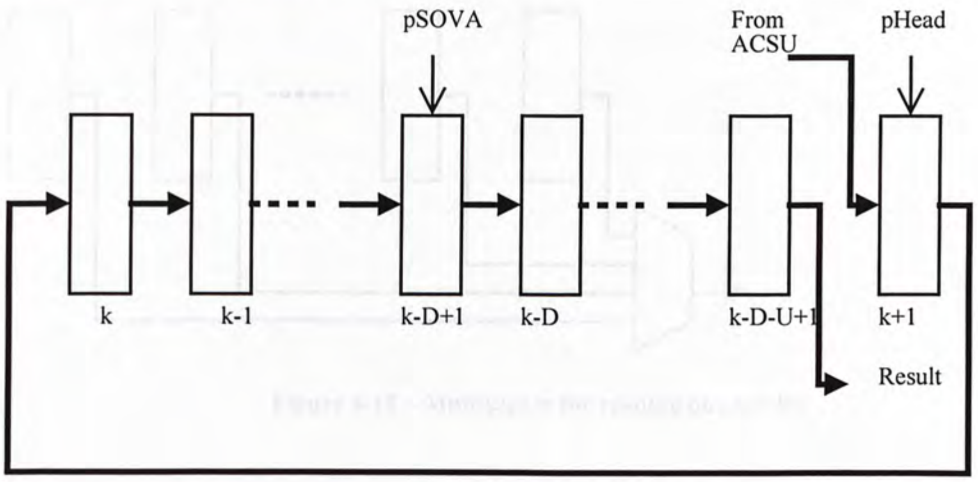


Figure 4-14 – Traceback at time j+1

The traceback is based on the two-step SOVA approach. In Figure 4-13, new data are written to the memory pointed by pHead. Then the VA traceback is started from time k to time k-D as indicated by pSOVA. This will find out the maximum likelihood path as in the first step of two-step SOVA. After that the SOVA updated is performed from time k-D to time k-D-U. Finally the decision and the reliability (soft-output) are given out.

Before starting next traceback, the pHead and pSOVA pointers move one memory location opposite to the traceback direction as in Figure 4-14. This essentially slides the decoding window one step, overwritten the oldest data with the new one from ACS unit. The traceback then starts again from pHead as before.

In conclusion, the data are hold in the memory and not moved. Instead the starting point of traceback and the point for collecting results changes at every traceback. Therefore a multiplexer is needed to select the right output from the memory.

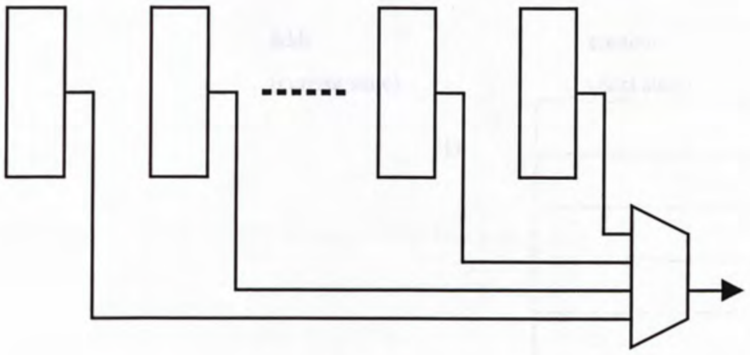


Figure 4-15 – Multiplexer for reading out results

4.5.6 Traceback Implementation

4.5.6.1 State Memory and Path Metric Difference Memory

The state memory is an array of pointers. Both the width of address and the memory content are 3 bits, which represents the eight states of the encoder. The address represents the current state and the content at that address is a pointer to the estimated next state. This information is written from the ACS unit when a TBcell is selected by pHEAD. Therefore one of the content of the memory will be selected as the current state for next stage according to the state of previous stage in the traceback unit. Figure 4-16 illustrates the structure of the state memory.

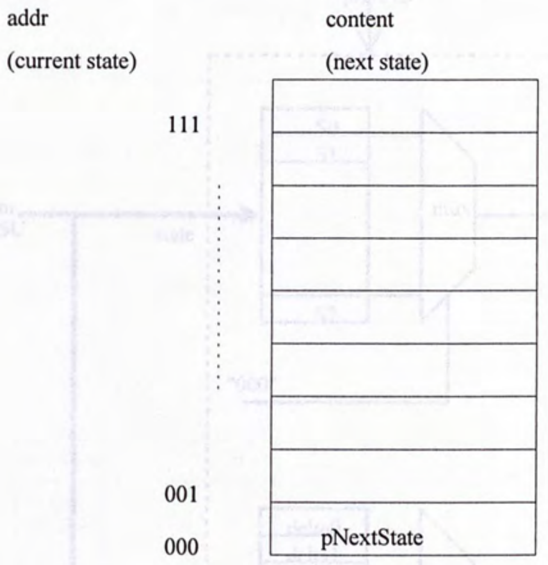


Figure 4-16 – Structure of State Memory

The path difference memory is similar to the State Memory, with the 3-bit address is the state and the 6-bit data is the path metric difference calculated from the ACS unit.

4.5.6.2 The Start of the Traceback

At the beginning of the traceback (pointed by pHead), the decisions and the path metric differences of each state are written to the memory respectively.

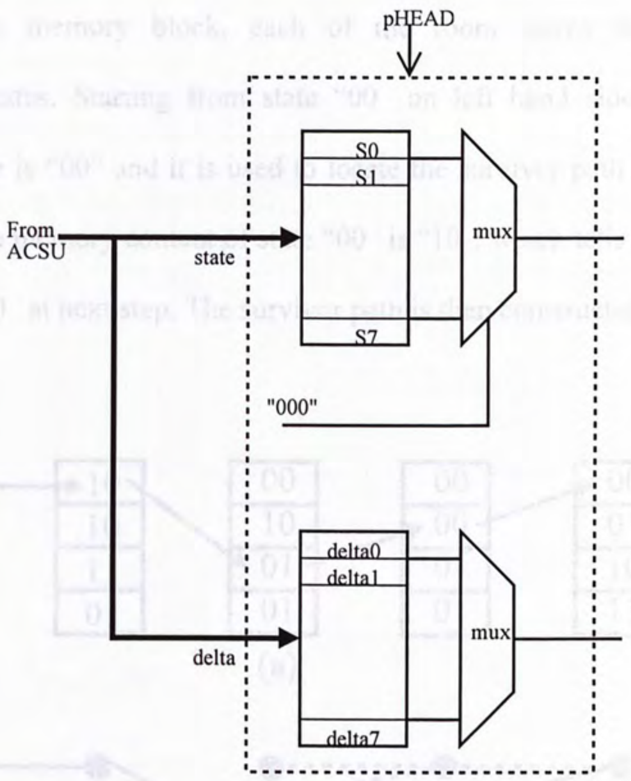


Figure 4-17 – Beginning of the traceback

As in Figure 4-17, the memory for the state and the path difference (delta) are written to the beginning of the traceback memory. Through the data bus is connected to all the blocks of the traceback memory, only the block pHead points to can be overwritten.

4.5.6.3 VA Traceback

For VA traceback, the process begins at the pHead location. It then follows the decision of the current survivor state and the pointer to the previous survivor state in time so that to construct the survivor path till the location pointed by pSOVA.

Figure 4-18a shows an example of memory traceback of four states and traceback length of 5. Figure 4-18b illustrates the corresponding trellis. There are four rooms in a survivor state memory block, each of the room stores the paths of the corresponding states. Starting from state “00 on left hand side, the content of memory for state is “00” and it is used to locate the survivor path to next stage. At the next stage the memory content of state “00 is “10 , which tells the survivor path travel to state “10 at next step. The survivor path is then constructed in this way.

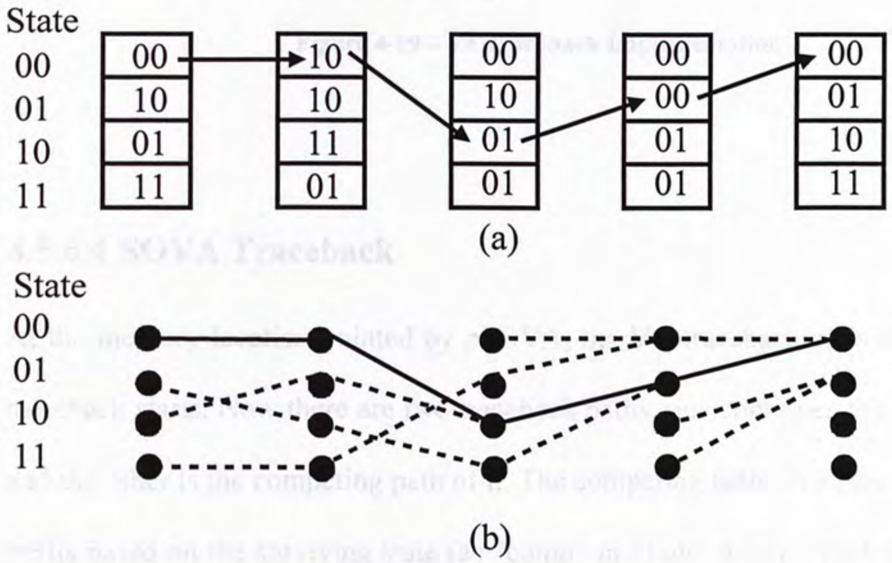


Figure 4-18 – VA Traceback example: (a) Survivor Path Memory (b) Survivor Path in the Trellis

In the implementation, the memory is built with registers and the memory content is selected by a multiplexer as shown in Figure 4-19. Within a traceback stage, the surviving state of previous stage is used to select the current surviving state; the corresponding memory content is used by the next stage to determine the surviving state.

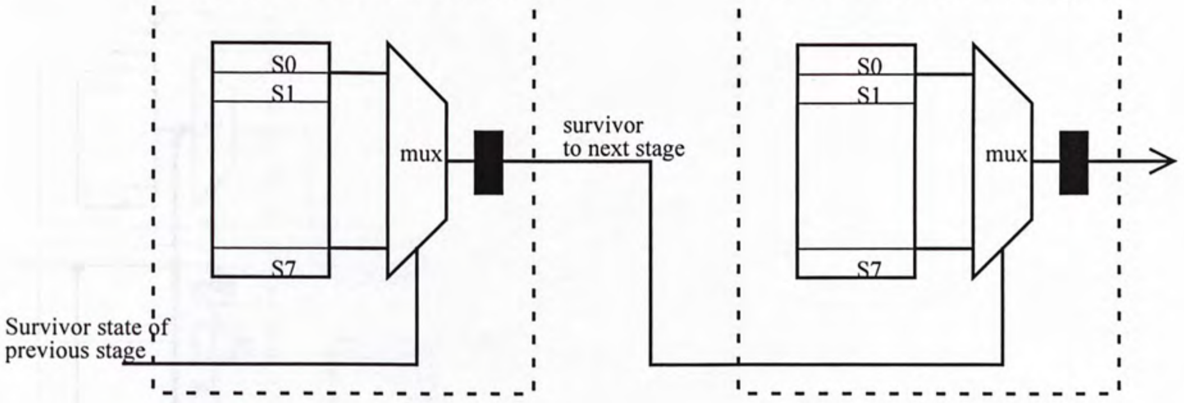


Figure 4-19 – VA Traceback Implementation

4.5.6.4 SOVA Traceback

At the memory location pointed by pSOVA, the VA traceback ends and the SOVA traceback starts. Now there are two traceback paths, one continues the VA traceback and the other is the competing path of it. The competing path is created according the trellis based on the surviving state (as "comp" in Figure 4-20). The S (survivor) and S' (competitor) are selected with the multiplexer controlled by the previous surviving state. Both S and S' are forwarded to next stage traceback for constructing the two paths. The path difference is initiated as infinity ("111111) for the starting of SOVA traceback.

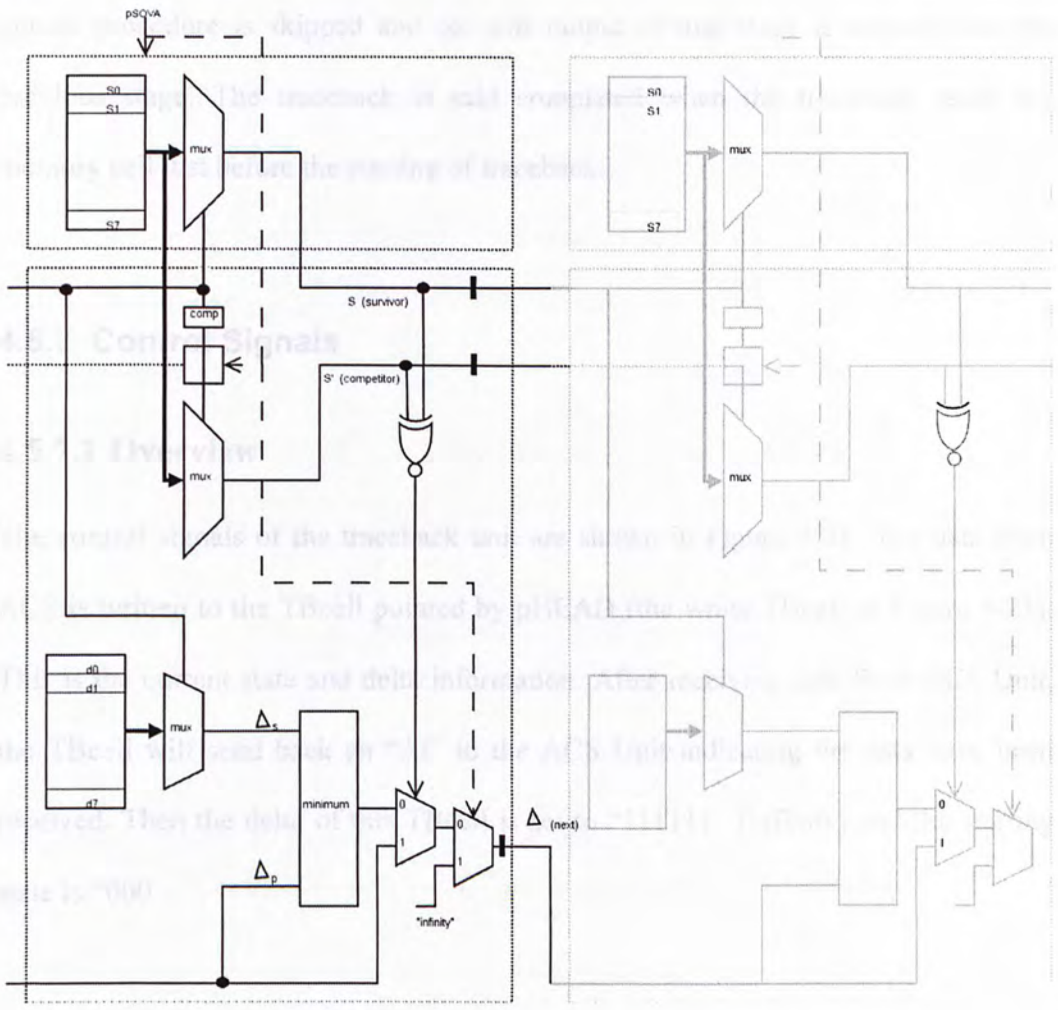


Figure 4-20 – SOVA Traceback Implementation

The state of surviving path, competing path and the path differences are then advanced to next stage. The pSOVA pointer is not pointing to it so that the competing path multiplexer takes the previous S' as input. The multiplexers then select the next S and S' as output. Now the two path traceback independently, however, sharing the same memory content in the stage.

The path differences Δ_s and Δ_p of the survivor and traceback paths are selected by multiplexers and compared. The minimum one is then propagate to next traceback stage. If the decision of S and S' are the same (indicated by the XNOR gate), the soft update procedure is skipped and the soft output of that stage is copied from the previous stage. The traceback is said completed when the traceback reach the memory cell just before the starting of traceback.

4.5.7 Control Signals

4.5.7.1 Overview

The control signals of the traceback unit are shown in Figure 4-21. The data from ACS is written to the TBcell pointed by pHEAD (the white TBcell in Figure 4-21). This is the current state and delta information. After receiving data from ACS Unit, the TBcell will send back an "Ai" to the ACS Unit indicating the data have been received. Then the delta of this TBcell is set to "111111 (infinity) and the starting state is "000 .

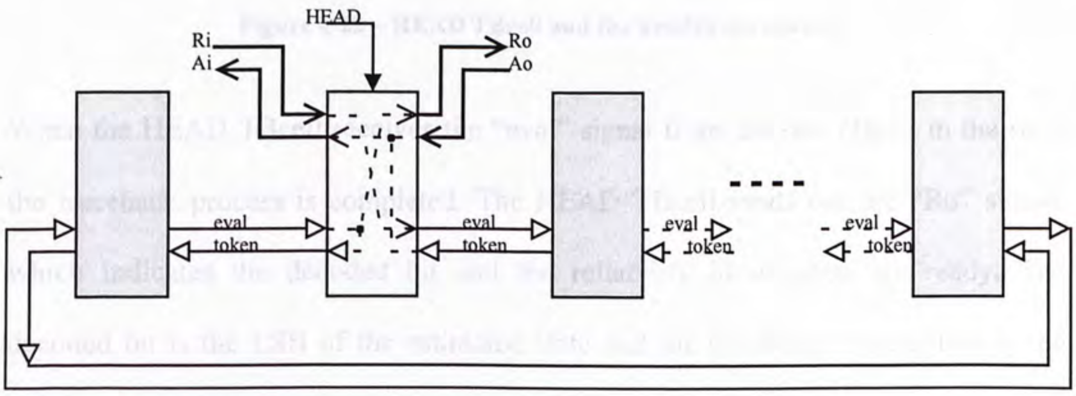


Figure 4-21 – Control signals in Traceback Unit

The pHead pointer and the request signal from ACS unit to the Traceback Unit initiate the traceback at the first TBcell of the traceback ring. This TBcell finds the estimated next state stored at address "000 and the reliability. This information is passed to next TBcell together with the "eval" signal. The "eval" signal initializes traceback in the next TBcell. The next TBcell then finds the estimated next state and calculate the soft-update if needed as discussed in section 4.5.6.1. The next time-slot will issue back a "token" signal when it completes receiving the data from previous slot. This process is repeated one by one through the ring until the HEAD TBcell receives the "eval" signal.

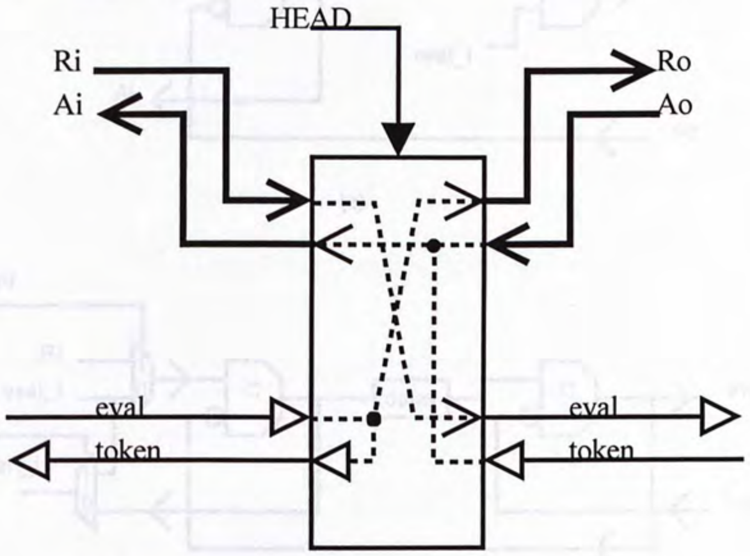


Figure 4-22 – HEAD TBcell and the handshake control

When the HEAD TBcell receives the "eval" signal from the last TBcell in the ring, the traceback process is completed. The HEAD TBcell sends out the "Ro" signal, which indicates the decoded bit and the reliability information are ready. The decoded bit is the LSB of the estimated state and the reliability information is the delta from last TBcell.

4.5.7.2 Implementation of Control Signals

There are two groups of control signals: one is for communication between the whole traceback unit and the other units in the decoder; the other is the handshake for communication between traceback memory cells. Figure 4-23 shows the control signals: (a) is the handshake for whole traceback unit; (b) is the handshake between memory cells. The four-phase handshake protocol is employed and C-elements are used as handshake controls.

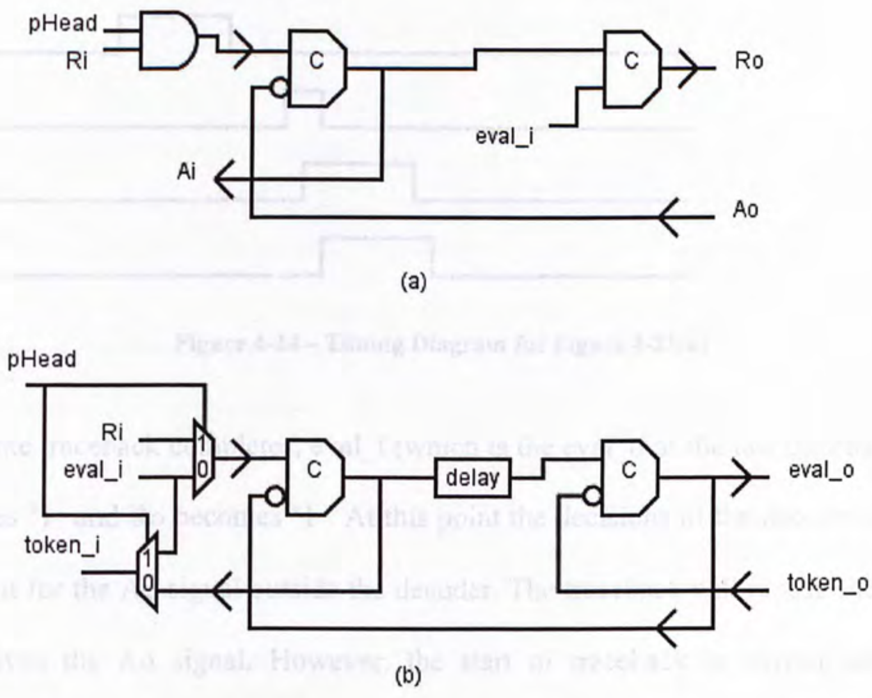


Figure 4-23 – Handshake Control circuit of the traceback memory cell

The handshake circuit for the Traceback unit (in Figure 4-23a) is built into every memory cell but only one of them operates. This is controlled by the AND gate and pHead. The C-element is disabled unless pHead is '1 . The pHead then defines and activates the start of traceback and the associated handshake signals in that particular memory cell.

Consider when pHead equals '1 , that is when the memory cell is the start of traceback. When Ri goes from '0 to '1', the output of the first C-element becomes '1 and it triggers the memory cell to save the data from ACS unit. It also generates

Ai signal which is the acknowledge signal from the Traceback Unit to the ACS unit. The Ri will also trigger the traceback inside the Traceback Unit as discussed later.

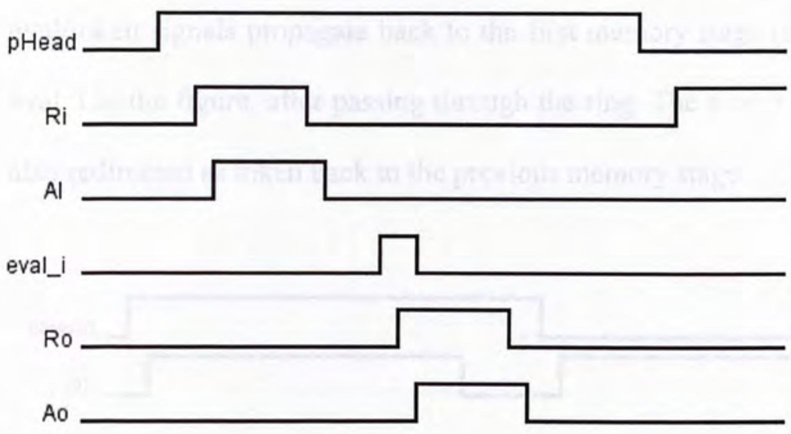


Figure 4-24 – Timing Diagram for Figure 4-23(a)

When the traceback completes, eval_i (which is the eval_o of the last traceback cell) becomes '1 and Ro becomes '1 . At this point the decisions of the decoder is ready and wait for the Ao signal outside the decoder. The traceback will repeat once when it receives the Ao signal. However, the start of traceback is moved one stage backward as controlled by pHead. Therefore the handshakes in Figure 4-23a are kept idle when pHead is '0 .

Figure 4-23b shows the handshake control for communication between the traceback memory cells. Since the eval signals are connected in a ring as the memory cells, there should be a point on the ring that acts as the start and end of the traceback and initiate the traceback handshake. The pHead is used to indicate the start/end position.

When pHead is '1 , the memory cell becomes the start of the traceback and Ri initiates the eval/token handshake. The delay element in Figure 4-23b is used to match the delay of the logic for looking up survivor state and soft update. The eval signal is then propagates to next memory stage controlled by the next C-element. The next stage will send back a token signal indicating the stage has taken the data. The eval/token signals propagate back to the first memory stage (where pHead = '1) as eval_i in the figure, after passing through the ring. The eval_i signal triggers Ro and also redirected as token back to the previous memory stage.

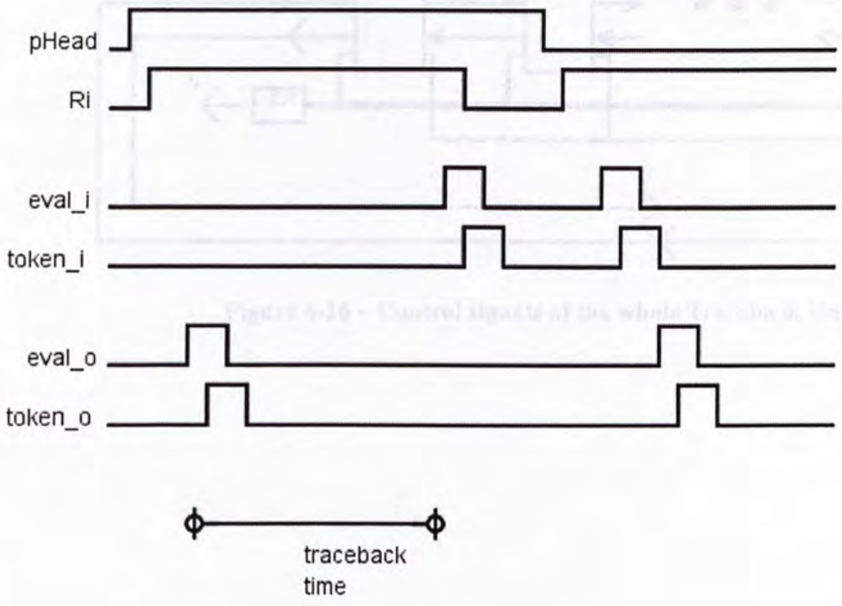


Figure 4-25 –Timing Diagram for Figure 4-23(b)

When pHead returns '0 , Ri is not involved in the handshaking of the memory stage. It then becomes an ordinary stage in the traceback.

Figure 4-26 shows the connection of the control signals of the whole Traceback Unit. pHead and pSOVA are generated by ring counters to avoid glitching in these signals. The Ri and Ao signals are connected to all the cells. The Ro and Ai signals are generated by OR-ing the Ro and Ai of individual cells as only one of them operates at a time. The *eval* and *token* signals control the traceback flow inside the Traceback

Unit. Ri triggers both the counters and the traceback. Therefore a delay element is necessary to keep the start of traceback well after the output of the counter.

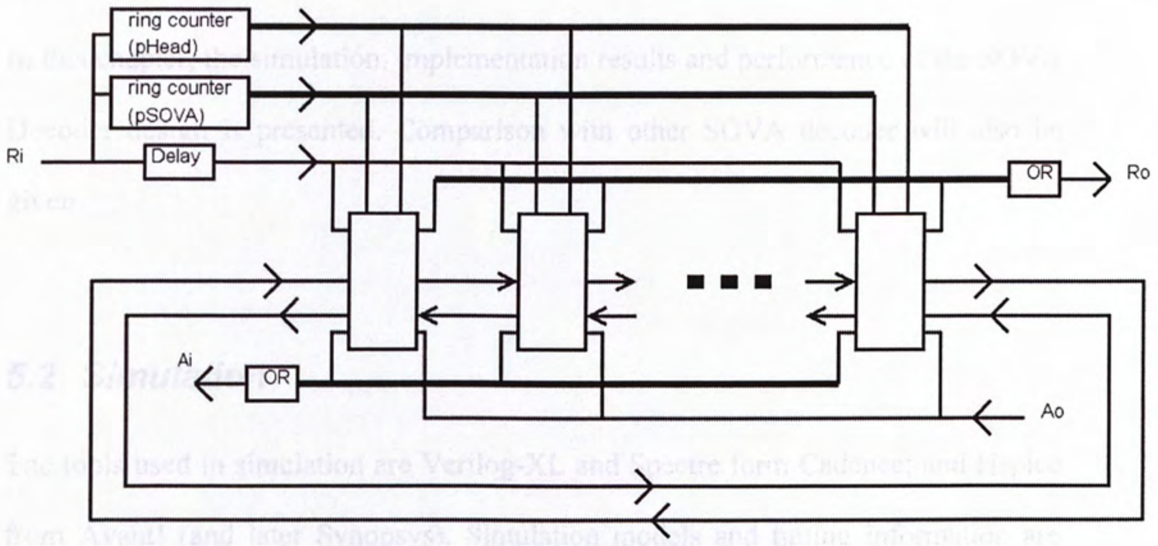


Figure 4-26 – Control signals of the whole Traceback Unit

Chapter 5 Implementation Result and Discussion

5.1 Overview

In this chapter, the simulation, implementation results and performance of the SOVA Decoder design is presented. Comparison with other SOVA decoder will also be given.

5.2 Simulation

The tools used in simulation are Verilog-XL and Spectre form Cadence; and Hspice from Avant! (and later Synopsys). Simulation models and timing information are from the foundry AustriaMicroSystems, HitKit v3.40.

5.2.1 Branch Metric Unit

The Branch Metric Unit as described in section 4.3, is built with self-timed adders as in section 2.6.

In Figure 5-1, *H* is an active-high signal derived from request-in signal indicating the inputs Go, G1 and G2 are ready. When the input data are ready, the internal adders will start calculate the Branch Metrics and indicate calculation complete by a cc signal. When all the cc signals are ‘1 , the calculation is said completed and a request-out is generated by the BMU. The next calculation starts when the next stage (ACS unit) generates a acknowledge signal to BMU and next symbol is ready (indicated by request in of the decoder).

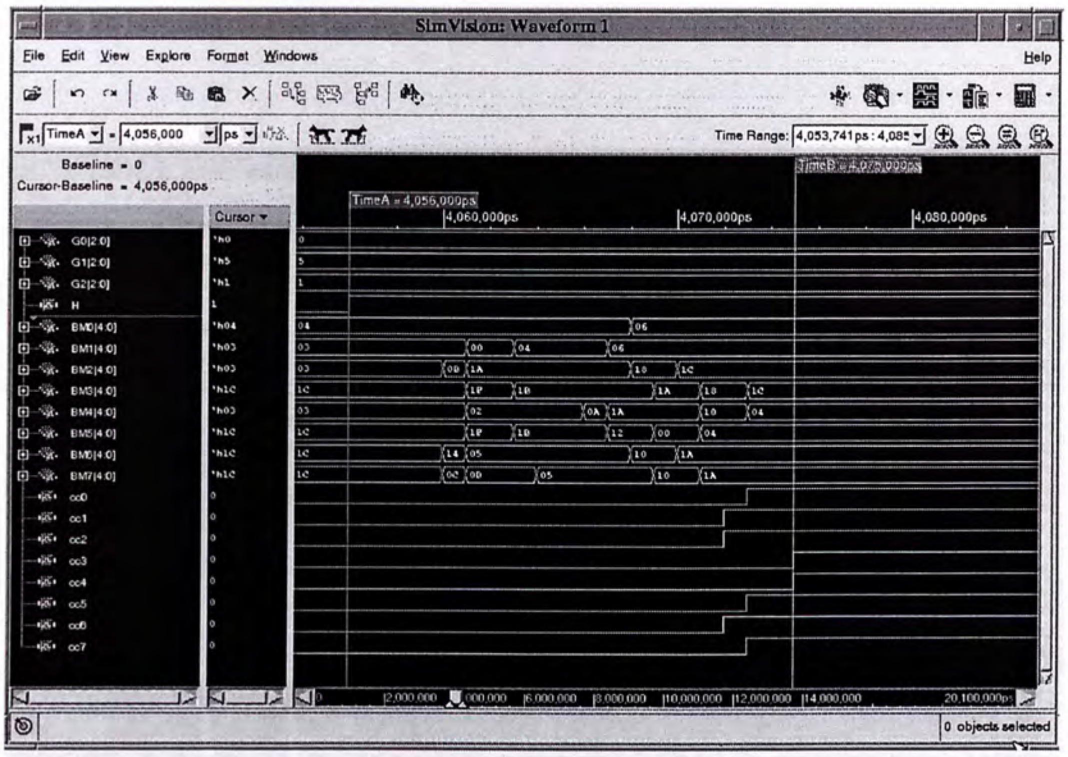


Figure 5-1 – Simulation Waveform of the Branch Metric Unit

5.2.2 ACS Unit

Figure 5-2 shows a simulation of the Add-Compare-Select Unit as described in section 4.4. In the figure, Ri and Ai are the request and acknowledge signal to and fro the Branch Metric Unit; Ro and Ao are the request and acknowledge signal to the Traceback Unit respectively. When Ri changes from '0 to '1 , it indicates the output from BMU is ready. The ACS unit then calculates the state metrics and the delta of all the states. The Ro, which is an AND function of the Ro's of all individual ACS blocks, changes from '0 to '1 to show the calculation is completed. However, the Ro of the ACS unit will be delayed if it has not received the acknowledge signal Ao from the next block, Traceback Unit. The Ro of the ACS unit will be issued only when its last output data has been acknowledged.

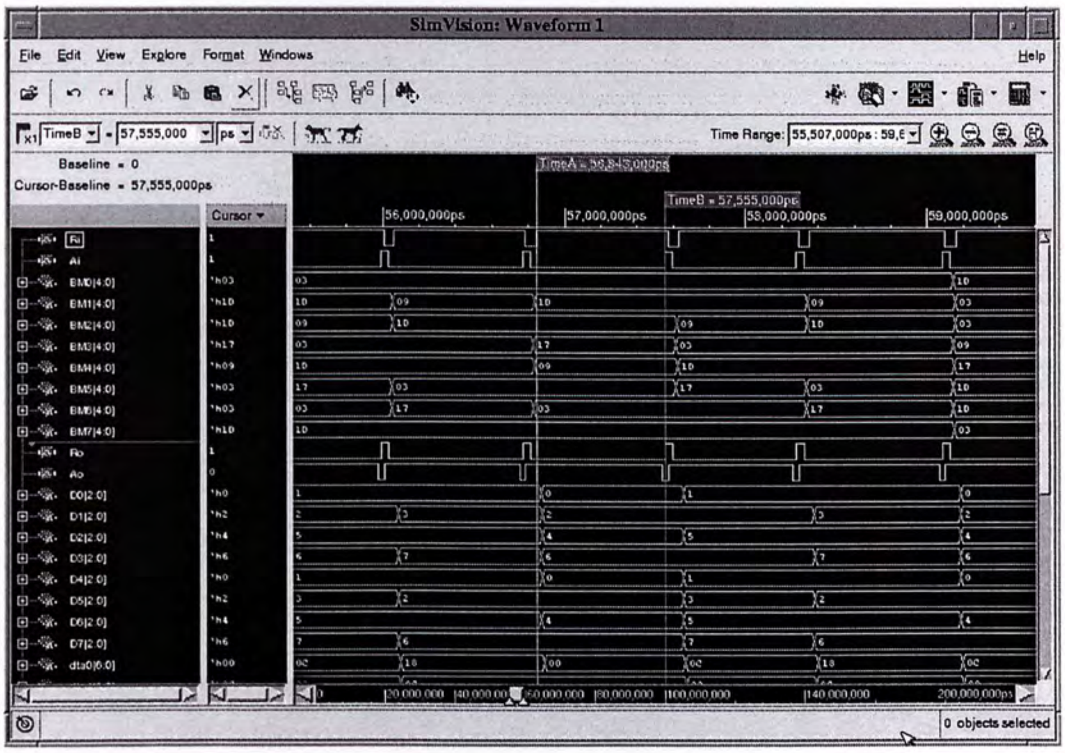


Figure 5-2 – Simulation Waveform of the Add-Compare-Select Unit

5.2.3 Traceback Unit

Figure 5-3 shows the major handshake signals of the Traceback Unit. As discussed in section 4.5.7, the handshakes are divided into two parts: one is controlling the data in and out the traceback unit; the other is the traceback path inside the traceback unit. First the state metrics are fed into the Traceback unit, with the request signal Ri indicates its readiness. The Traceback unit will then acknowledge the receive data via Ai, and internally start traceback controlled by eval (as shown in Figure 5-3) and token (not shown in the figure).

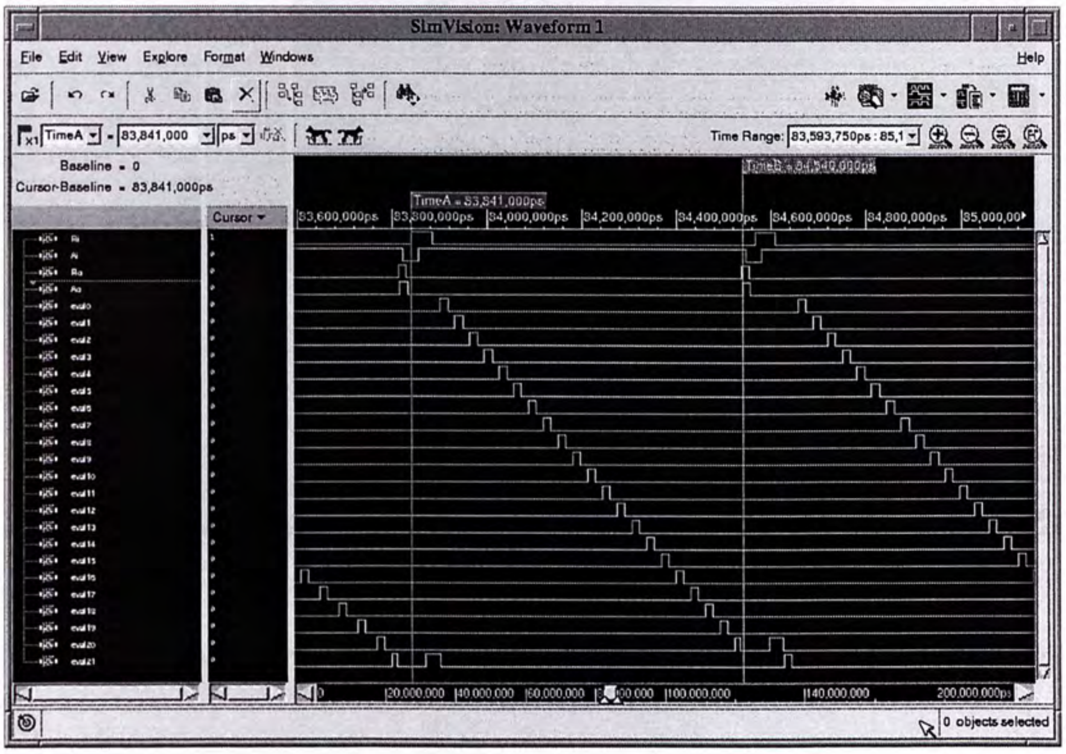


Figure 5-3 - Simulation Waveform of the handshake signals of Traceback Unit

Figure 5-4 shows the handshake signals of a traceback cell in the Traceback Unit. The handshake signals of data to and fro traceback unit is activated only when it becomes the head of traceback (head == '1). For the other times, the traceback is not started from a particular cell if it is not the head even the request signal (Ri) comes in for every traceback in the Traceback unit.

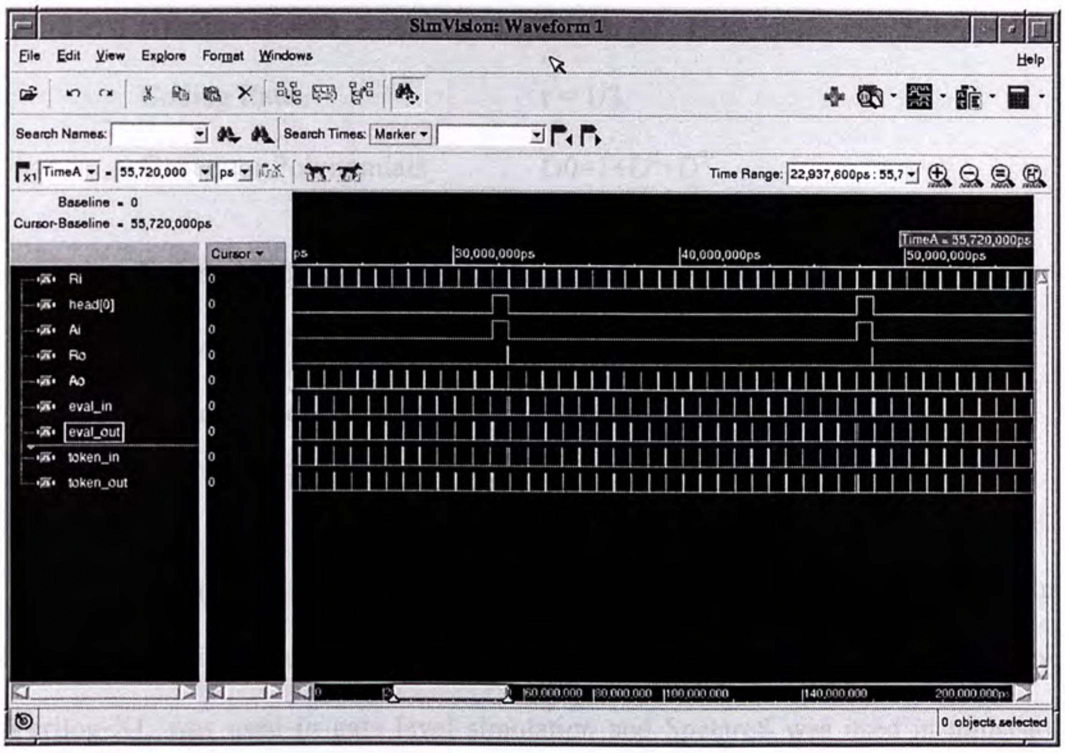


Figure 5-4 – Simulation Waveform of the handshake signals of a traceback cell in the Traceback Unit

5.3 Chip Fabrication

The test chip consists of the SOVA decoder, a convolutional encoder and a Linear Feedback Shift Register (LFSR). The LFSR generates a pseudo-random bitstream that is encoded by the convolutional encoder. The SOVA decoder is configured so that it can decode the bitstream from the internal encoder or outside the chip.

The SOVA decoder accepts three 3-bit soft input symbols and gives out 6-bit soft output along the decoded bits. Table 5-1 summarizes the specification of the SOVA Decoder.

Number of state	8 state
Coding Rate	r = 1/3
Generator Polynomials	$G0=1+D^2+D^3$ $G1=1+D+D^3$ $G2=1+D^1+D^2+D^3$
External Input Symbol	3-bit soft-input symbol
Path-Metric	8-bit Modulo Arithmetic
Soft output decision level	6-bit

Table 5-1 – Specification of the SOVA Decoder

The design schematics were captured in Cadence Virtuoso Schematic Editor. The standard cell library used was AMS CSI 0.35um HRDLIB standard cell library. Verilog-XL was used in gate level simulation and SpectreS was used in transistor level simulation. Cadence Silicon Ensemble was used in placement and routing of the chip. The timing of all the individual building blocks are simulated in SpectreS for correct timing at 3.3V and 1.2V in typical conditions.

The chip was fabricated using AMS CSI 0.35um 3-layer metal CMOS technology through CMP France and was packed in a 40-pin dual-in-line package. The size of the core is 2.5 mm^2. One functional flaw was found in the chip causing the soft update performed on the whole traceback path instead of the original form. The decoded bit is not affected while the soft output tends to be underestimated; yet the power of the chip still serves as a good power estimation for the decoder design as the calculation complexity is about the same.

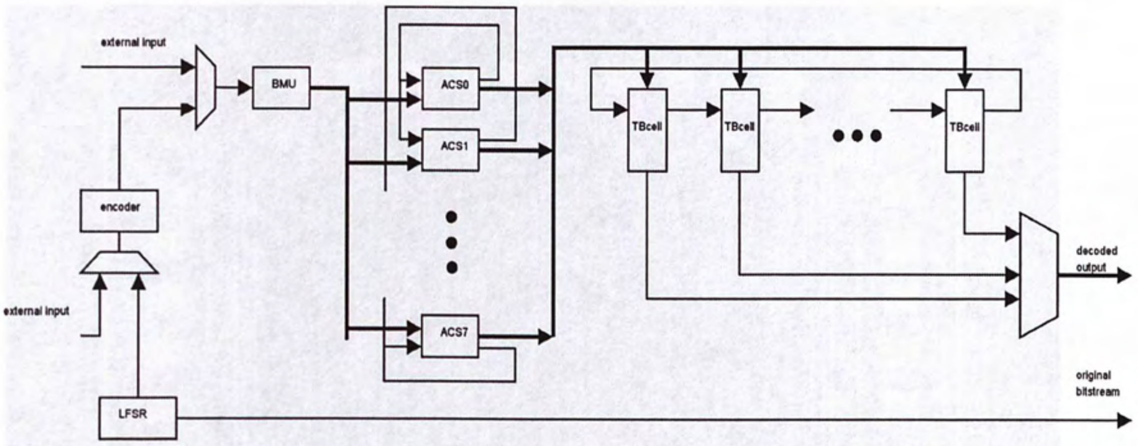


Figure 5-5 – The whole SOVA Decoder test chip architecture.

Technology	AMS 0.35 µm 3-layer metal CMOS
Chip Size	2.5 x 2.5 mm^2
Core Size	1.6 x 1.6 mm^2
Gate count	28k
Power Dissipation (excluding Pad)	6.23mW (2 M Symbol/s, 3.3V),

Table 5-2 – Chip Summary

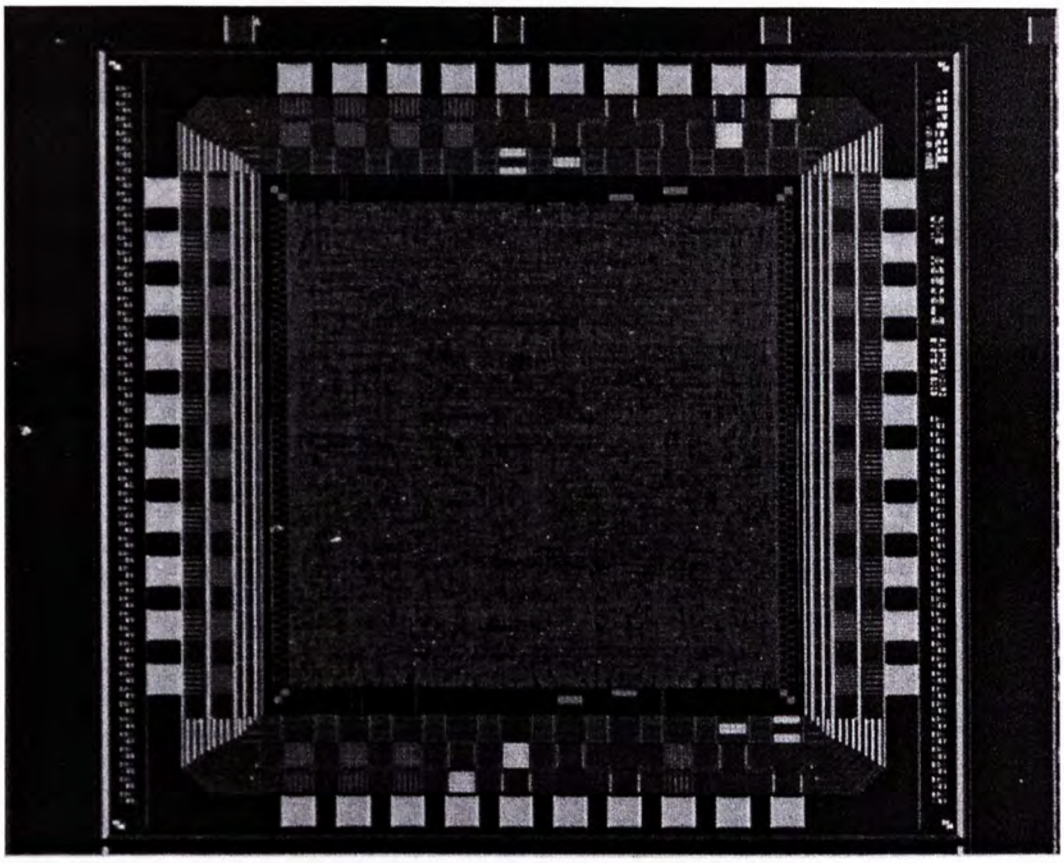


Figure 5-6 – Micrograph of the SOVA test chip

The layout of the test chip is core-limited, as the I/O of the test chip are mainly in serial form. Core utilization of the test chip is about 60%. Cadence Silicon Ensemble DSM is used in place-and-route, and Cadence Diva is used in DRC and LVS of the test chip before the chip was taped-out.

5.4 Measurements

5.4.1 Measurement Setup

Figure 5-7 shows the measurement setup for testing. The chip was tested with HP16702B logic analyzer mainframe. Test pattern is generated from pattern generator and encoded with the on chip built-in encoder. The encoded bitstream is then decoded with the SOVA decoder and the output waveforms are captured by the logic analyzer. Delay element is used to provide the time-delayed *request_out* signal as *acknowledge* signal at decoder output. The delay element is built with several logic buffers from 74HC4050. The delay time is about half of the period of *Ro* signal. Power measurement is done at 3.3 V Vcc with RMS current measurement by Fluke 189 multimeter.

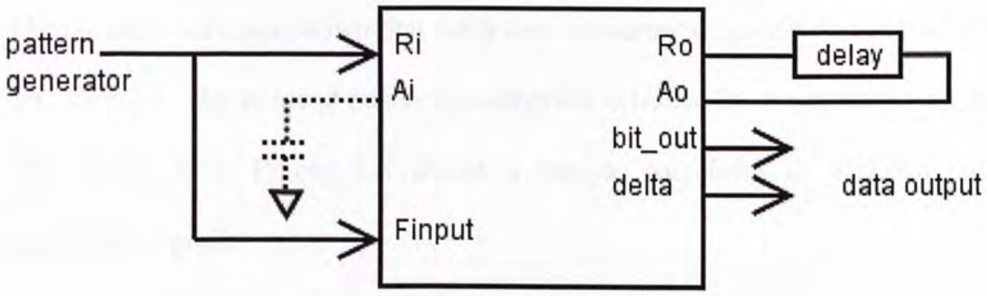


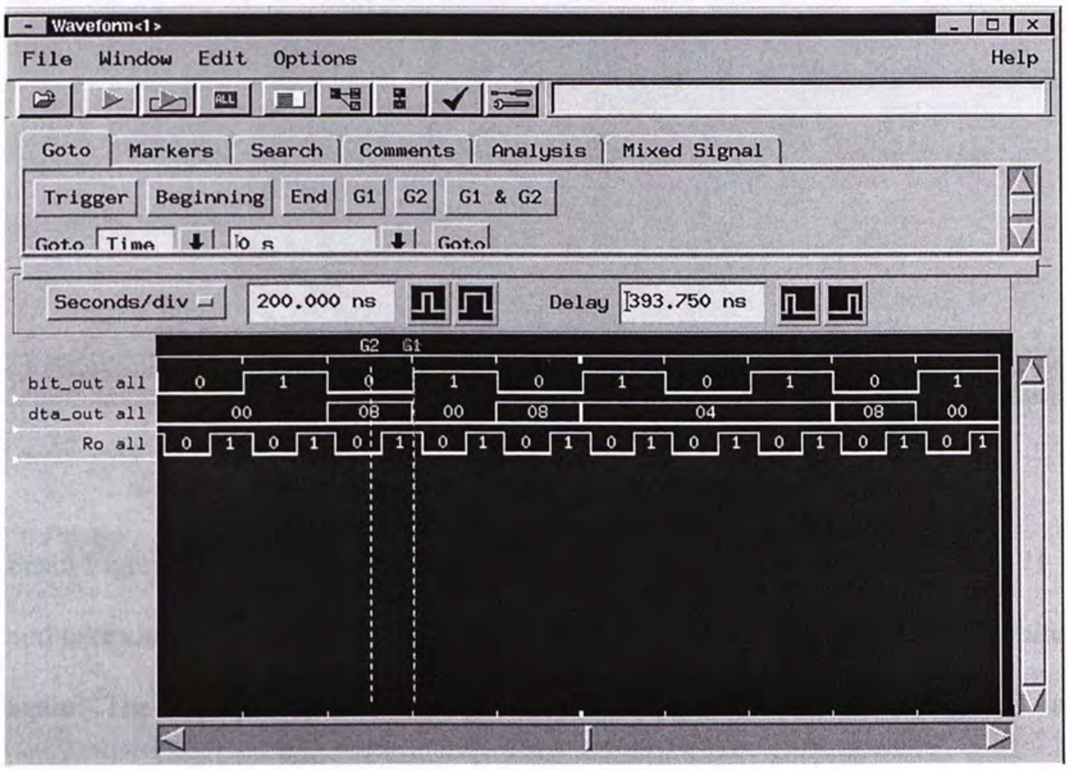
Figure 5-7 – Measurement Setup

The input signal Ao, which is a delay version of Ro, act as an acknowledgement signal to the decoder indicating the data is taken and the decoder can change the output data for next decoded symbol. The Ao signal will trigger Ro to be ‘0 and next decoded symbol is prepared when Ro is ‘0 internally at output stage of the decoder. The Ro indicates the output symbol is ready by changeing to ‘1 .

5.4.2 Performance

The dataflow of the decoder has been shown in Chapter 4. The decoder requires a circular traceback which limits the highest speed of the decoder. This circular traceback also introduce a relative long latency before data can be given out. On the other hand, the traceback ring does not move data around the traceback ring and therefore much power can be saved. However, the area of the traceback is larger than ordinary memory traceback decoders as register is required to store the intermediate traceback data instead of using RAM. This also affect the place-and-route of the implementation on silicon for the more complicated datapath routing, which is reflected by a core utilization of 60% for this 3-metal layer area-routing implementation.

The measurement has shown that the power consumption of the decoder is only 6.23 mW at 3.3V. The reduced power consumption is traded by the speed of the decoder and silicon area. Figure 5-8 shows a sample waveform of decoded data and handshake signals.

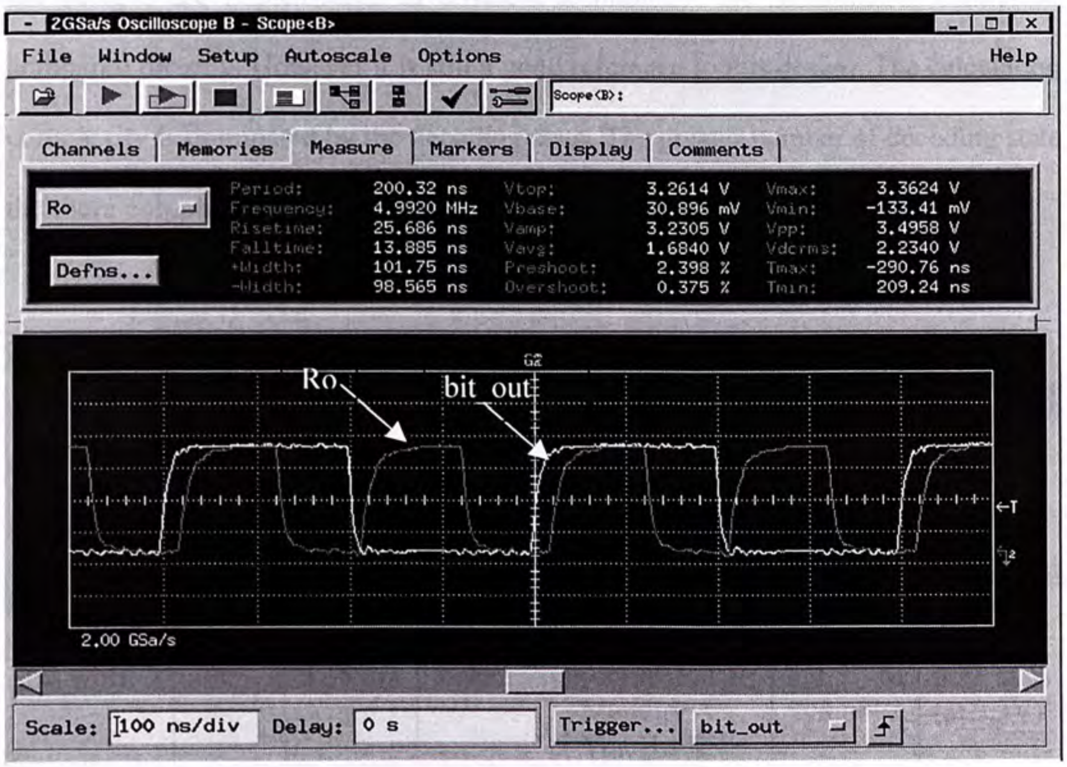

Waveform<1>
File Window Edit Options Help
Goto Markers Search Comments Analysis Mixed Signal
Trigger Beginning End G1 G2 G1 & G2
Goto Time 0 s Goto
Seconds/div
Delay
G2 G1
bit_out all
dta_out all
Ro all

Seconds/div 200.000 ns Delay 393.750 ns

```
bit_out all  0 1 0 1 0 1 0 1 0 1
dta_out all  00 08 00 08 04 08 00
Ro all       0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

Figure 5-8 – Decoded output with data stream "1010 "

The speed of the design is deliberately only high enough for mobile application to tradeoff between speed and power, in which power is the major concern in this design. The maximum decoding rate of the decoder is 5 M Symbol/s. This decoding rate is sufficient for channel decoding in current 3GPP specification [1]. Figure 5-9 shows the analog waveform of the output of the decoder at 5 M Symbol/s, which is relatively low for a 0.35 um CMOS technology implementation.



```
Scope<B>:

Period:      200.32 ns    Vtop:         3.2614 V    Vmax:      3.3624 V
Frequency:   4.9920 MHz   Vbase:        30.896 mV   Vmin:     -133.41 mV
Risetime:    25.686 ns    Vamp:         3.2305 V    Vpp:       3.4958 V
Falltime:    13.885 ns    Vavg:         1.6840 V    Vdcrms:    2.2340 V
+Width:      101.75 ns    Preshoot:      2.398 %    Tmax:     -290.76 ns
-Width:      98.565 ns    Overshoot:     0.375 %    Tmin:      209.24 ns

Scale: 100 ns/div   Delay: 0 s
```

Figure 5-9 – Analog waveform of decoded bit and *Ro*

From Figure 5-9, it can be observed that the data is hold valid through the Ro is '1 , and take changes when the Ro is '0 . The Ro goes to '1 again when the data is valid again. The four-phase handshaking used in this design can be through as in synchronous digital designs, in which the Ro acts like clock and the data is ready at the rising edge of Ro. Hence normal flip-flops, rather than special two-phase latches, are used in this implementation.

5.4.3 Comparison with other Design

Table 5-3 shows performance comparison of different implementations of SOVA decoder. One functional flaw was found in the fabricated device by Garrett and Stan [25] and the power consumption is their estimation with another similar chip. Since the designs are implemented in different technology, the performance cannot be compared directly. However it is still a good reference to this design. The calculation complexity is dominated by the decoding state. The greater number of decoding state, the more complex of the calculations.

Design	**Year**	**Tech.**	**Decoding State**	**Traceback Type**	**Data rate**	**Power Consumption**
Joeressen and Meyr [24]	1995	1 μm	16-state	Register Exchange	40 M Symbol/s	1600 mW
Garrett and Stan [25]	2001	1.2 μm	4-state	Memory Traceback	2.5 M Symbol/s	23 mW (estimated)
E. Yeo, et. al. [15]	2002	0.18 μm	8-state	Register Exchange	400 M Symbol/s	400 mW
This work	2003	0.35 μm	8-state	Asynchronous Memory Traceback	2.0 M Symbol/s	6.23 mW (at 3.3V)

Table 5-3 – Performance comparison of different SOVA implementations

It can be seen that the register exchange type traceback can reach a high decoding rate while lower power consumption is archived by memory traceback. High decoding rate is necessary in applications such as magnetic storage devices while the low power approach is suitable in portable device such as cell phones.

Chapter 6 Conclusion

The Viterbi Algorithm is used to find the optimum state transition in a trellis of binary convolutional codes and has been employed widely in digital communication systems. Convolutional coding and Viterbi decoders are used in various situations for their forward error correction capability over a noisy channel. The Soft output Viterbi algorithm can produce the reliability of the decision along the decoded bit and it can be used as component decoder of concatenated convolutional codes, which is used in next-generation digital communication systems such as third generation (3G) mobile phones.

Several Asynchronous circuit techniques have also been discussed, and a new asynchronous design of the add-compare-select unit and traceback algorithm is also presented. Based on the asynchronous traceback architecture, a soft output Viterbi algorithm (SOVA) decoder is designed and implemented in AMS CSI 0.35 μm technology.

The design of the SOVA decoder adopts an asynchronous logic style where local handshakes are used instead of a global clock to control data flow. An asynchronous memory traceback unit is also developed. The [illegible] [illegible] are stored in the traceback memory which reduce the switching and hence the power consumption of the decoder. Voltage scaling can further reduce the power consumption of the decoder.

Chapter 6 Conclusion

The Viterbi Algorithm is used to find the optimum state transition in a trellis of binary convolutional codes and has been employed widely in digital communication systems. Convolutional coding and Viterbi decoders are used in various situations for their forward error correction capability over a noisy channel. The Soft output Viterbi algorithm can produce the reliability of the decision along the decoded bit and it can be used as component decoder of concatenated convolutional codes, which is used in next-generation digital communication systems such as third generation (3G) mobile phones.

Several Asynchronous circuit techniques have also been discussed, and a new asynchronous design of the add-compare-select unit and traceback algorithm is also presented. Based on the asynchronous traceback architecture, a soft output Viterbi algorithm (SOVA) decoder is designed and implemented in AMS CSI 0.35 μm technology.

The design of the SOVA decoder adopts an asynchronous logic style where local handshakes are used instead of a global clock to control data flow. An asynchronous memory traceback unit is also developed. The majority data are static in the traceback memory which reduce the switching and hence the power consumption of the decoder. Voltage scaling can further reduce the power consumption of the decoder.

The drawback of this traceback design is only low decode rate can be archived which is common among other memory traceback design. Therefore this design is not suitable in high throughput application such as magnetic storage device. However, it is appropriate for portable devices where the data rate is relatively lower and low power consumption is essential.

References

[1] The 3rd Generation Partnership Project, http://www.3gpp.org

[2] C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes (1)," *in Proc., IEEE Int. Conf. on Commun.*, (Geneva, Switzerland), pp. 1064--1070, May 1993.

[3] Bruce Gilchrist, J.H. Pomerene and S. Y. Wong "Fast Carry Logic for Digital Computers," *IRE Transactions on Electronic Computers*, December, pp. 133-136, 1955.

[4] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its application," *IEEE Globecom*, pp.1680-1686, 1989.

[5] Jon K. Lexau, John E. Will and Ian W. Jones, "EM Emissions of an Asynchronous Test Chip," *Proceedings of the 14th International Zurich Symposium & Technical Exhibition of Electromagnetic Compatibility*, Zurich Switzerland, pp. 579-584, 20-22 Feb. 2001. Also available on http://research.sun.com/async/Publications/KeyPapersPD1.html

[6] Lar S. Nielsen, Cees Niessen, Jens Sparsø and Kees van Berkel, "Low-Power Operation Using Self-Timed Circuits and Adaptive Scaling of the Supply Voltage," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp 391-397, Dec. 1994.

[7] G.D. Forney, "The Viterbi Algorithm," *Proc. IEEE*, vol. 61, pp. 268-278, Mar. 1973.

[8] Branka Vucetic and Jinhong Yuan, "Turbo Codes Principles and Applications," Kluwer Academic Publishers, 2000. pp 171-177.

[9] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, New York: Mc Graw-Hill, 1979.

[10] J. Hagenauer, "Source-Controlled Channel Decoding," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449-2457, Sept. 1995.

[11] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720-738, Jun 1989.

[12] T. E. Williams, "Self-timed Rings and their Application to Division," Ph.D. Thesis, Standford University, May 1991.

[13] D. Kearney and N. W. Bergmann, "Bundled Data Asynchronous Multipliers with Data Dependent Computation Times," *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, pp. 186-197,Apr 1997.

[14] Jason P. Woodard and Lajos Hanzo, "Comparative Study of Turbo Decoding Techniques: An Overview," *IEEE Transactions on Vehicular Technology*, vol. 49, no. 6, pp. 2208-2233, Nov. 2000.

[15] E. Yeo, S. Augsburger, Wm. R Davis and D. Nikolic, "Implementation of High Throughput Soft Output Viterbi Decoders," *Proc. IEEE Workshop on Signal Processing Systems, 2002. (SIPS '02)*, pp. 146-151, Oct. 16-18, 2002.

[16] Bernard Sklar, "A Primer on Turbo Code Concepts," *IEEE Communications Magazine*, pp. 94-102, Dec. 1997.

[17] C. B. Shung, P. H. Siegel, G. Ungerboeck and H. K. Thaper, "VLSI Architectures for Metric Normalization in the Viterbi Algorithm," *Conference Record, IEEE International Conference on Communications*, vol. 4, pp. 1723-1728, Apr. 1990.

[18] H. L. Lou, "Implementing the Viterbi Algorithm," *IEEE Signal Processing Magazine*, Sept 1995, pp. 42-52.

[19] M. Shams, J. C. Ebergen and M. I. Elmasry, "Modeling and Comparing CMOS Implementations of the C-Element," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, Dec 1998.

[20] R. J. McEliece and I. M. Onyszchuk, "Truncation Effects in Viterbi Decoding," *Proceedings of the IEEE Conference on Military Communications*, pp.29.3.1-29.3.3, Oct 1989.

[21] I. Kang and A. N. Willson, "Low-Power Viterbi Decoder for CDMA Mobile Terminals," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, Mar 1998.

[22] Y. N. Chang, H. Suzuki and K. K. Parhi, "A 2-Mb/s 256-State 10-mW Rate-1/3 Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 6, Jun 2000.

[23] G. Fettweis and H. Meyr, "High-Rate Viterbi Processor: A Systolic Array Solution," *IEEE Journal on Selected Areas in Communications*, vol. 8 no. 8, Oct 1990.

[24] O. J. Joeressen and H. Meyr, "A 40 Mb/s Soft-output Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 7, Jul 1995.

[25] D. Garrett and M. Stan, "A 2.5 Mb/s, 23 mW SOVA Traceback Chip for Turbo Decoding Application," *Proc. IEEE ISCAS 2001*, vol. 4, pp.61-64.

[26] L. E. M. Brackenbury, "An Asynchronous Viterbi Decoder," *Principles of Asynchronous Circuit Design*, pp. 249-272, Kluwer Academic Publisher, 2001.

[27] H. Dawid, O. J. Joeressen and H. Meyr, "Viterbi Decoders: High Performance Algorithms and Architectures," *Digital Signal Processing for Multimedia Systems*, pp. 417-459, Marcel Dekker Inc. 1999.

[28] J. Butas, C. S. Choy, J. Povazanec and C. F. Chan, "Asynchronous Cross-Pipelined Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 8, Aug 2001.

[29] C. S. Choy, J. Butas, J. Povazanec and C. F. Chan, "A New Control Circuit for Asynchronous Micropipelines," *IEEE Transactions on Computers*, vol. 50, no. 9, Sept 2001.

[30] T. -Y. Sin, E. M. C. Wong and I. C. C. Jong, "A 1.6-GHz 16x16b Asynchronous Pipelined Multiplier," *Proc. IEEE 2001 Midwest Symposium on Circuit and Systems*, vol. 1, pp. 336-339.

[31] J. –L. Yang, C. S. Choy, C. F. Chan, "A Self-Timed Divider Using a New Fast and Robust Pipeline Scheme," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 6, Jun 2001.

[32] A. W. Burks, H. H. Goldstine and J. v. Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," *Inst. For Advanced Study Report (1946)*. Appears on Computer Structures: Reading and Examples, McGraw-Hill Inc., 1971.

[33] M. Bossert, "Convolutional Codes," *Channel Coding for Telecommunications*, John Wiley & Sons, 1998, pp. 201-286.

[34] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding.* McGraw-Hill Book Company, New York, NY, 1979.

[35] B. Parhami, *Computer Arithmetic Algorithms and Hardware Design*, Oxford University Press, New York, 2000.

[36] J. Hagenauer, E. Offer and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, Mar 1996.

[37] O. Joeressen, M. Vaupel and H. Meyr, "High-Speed VLSI Architectures for Soft-Output Viterbi Decoding," *Proc. IEEE Conference on Application Specific Array Processors*, pp.373-384, 1992.

[38] C. Berrou, P. Adde, E. Angui and S. Faudeil, "A Low Complexity Soft-Output Viterbi Decoder Architecture," in *Proc. IEEE International Conference of Communications*, vol. 2, pp. 737-740, 1993.

[39] O. J. Joeressen, M. Vaupel and H. Meyr, "Soft-Output Viterbi Decoding: VLSI Implementation Issues," *in Proc. IEEE Vehicular Technology Conference 1993*, pp. 941-944.

[40] K. T. Christensen, P. Jensen, P. Korger and J. Sparso, "*The design of an asynchronous TinyRISCTM TR4101 microprocessor core*," in Proc. Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 108 –119.

Appendix

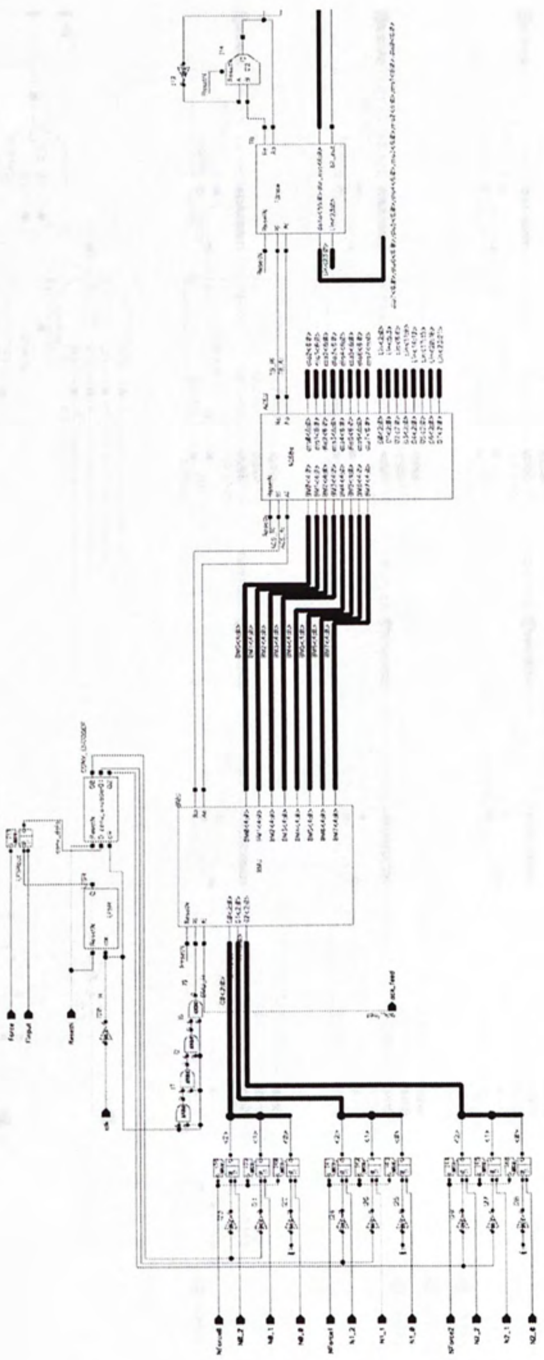
Pin Assignment of the SOVA test chip

VDD3RP	1		40	GND3RP
ResetN	2		39	VDD3IP
NForce0	3		38	GND3IP
N0 2	4		37	VDDOP
N0 1	5		36	GNDOP
N0 0	6		35	dta out6
Finput	7		34	dta out5
Force	8	SOVA	33	dta out4
req	9		32	bit out
ack feed	10	DIL 40	31	Ao
NForce1	11		30	Ro
N1 2	12		29	dta out3
N1 1	13		28	dta out2
N1 0	14		27	dta out1
NForce2	15		26	dta out0
N2 2	16		25	VDD3RP
N2 1	17		24	GND3RP
N2 0	18		23	LFSRout
GND3OP	19		22	VDD3IP
VDD3OP	20		21	GND3IP

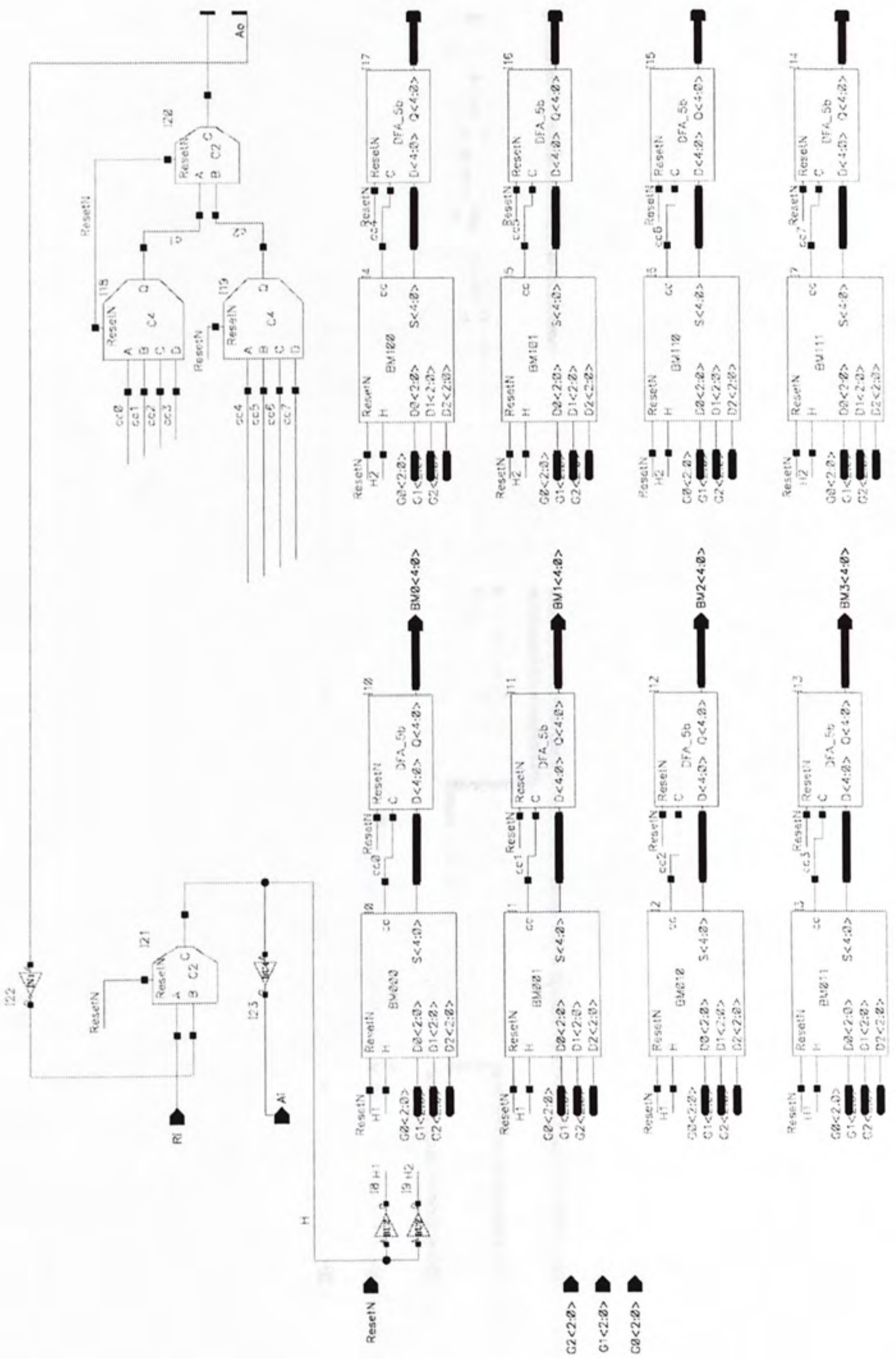
Pin Number	Pin Name	Direction	Description
1	VDD3RP	INOUT	VDD for PAD
2	ResetN	IN	Global Reset
3	Nforce0	IN	0: The decoder accept 1st symbol from internal encoder 1: The decoder accept 1st symbol from N0
4	N0_2	IN	Bit 2 of N0
5	N0_1	IN	Bit 1 of N0
6	N0_0	IN	Bit 0 of N0
7	Finput	IN	Bitstream to internal encoder
8	Force	IN	0: The internal encoder encodes the bit stream from LFSR 1: The internal encoder encodes bit stream from Finput
9	req	IN	Request in
10	ack	OUT	Acknowledge in
11	Nforce1	IN	0: The decoder accept 2nd symbol from internal encoder 1: The decoder accept 2nd symbol from N1
12	N1_2	IN	Bit 2 of N1
13	N1_1	IN	Bit 1 of N1
14	N1_0	IN	Bit 0 of N1
15	Nforce2	IN	0: The decoder accept 3rd symbol from internal encoder 1: The decoder accept 3rd symbol from N2
16	N2_2	IN	Bit 2 of N2
17	N2_1	IN	Bit 1 of N2
18	N2_0	IN	Bit 0 of N2
19	GND3OP	INOUT	GND for PAD
20	VDD3OP	INOUT	VDD for PAD
21	GND3IP	INOUT	GND for core
22	VDD3IP	INOUT	VDD for core
23	LFSRout	OUT	Output from the internal LFSR
24	GND3RP	INOUT	GND for PAD
25	VDD3RP	INOUT	VDD for PAD
26	dta_out0	OUT	bit 0 of soft output
27	dta_out1	OUT	bit 1 of soft output
28	dta_out2	OUT	bit 2 of soft output
29	dta_out3	OUT	bit 3 of soft output
30	Ro	IN	Request out
31	Ao	OUT	Acknowledge out
32	bit_out	OUT	Decoded bit

33	dta_out4	OUT	bit 4 of soft output
34	dta_out5	OUT	bit 5 of soft output
35	dta_out6	OUT	bit 6 of soft output
36	GNDOP	INOUT	GND for PAD
37	VDDOP	INOUT	VDD for PAD
38	GND3IP	INOUT	GND for core
39	VDD3IP	INOUT	VDD for core
40	GND3RP	INOUT	GND for PAD

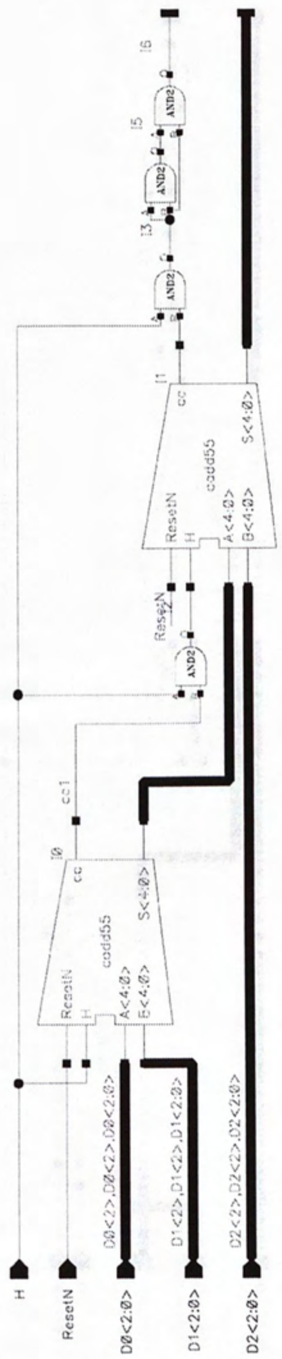
Schematic of the whole SOVA Decoder



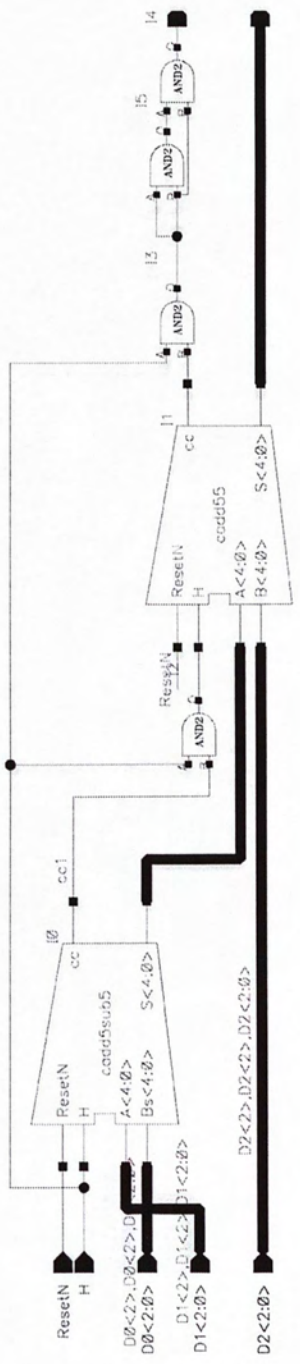
Schematic of module BMU



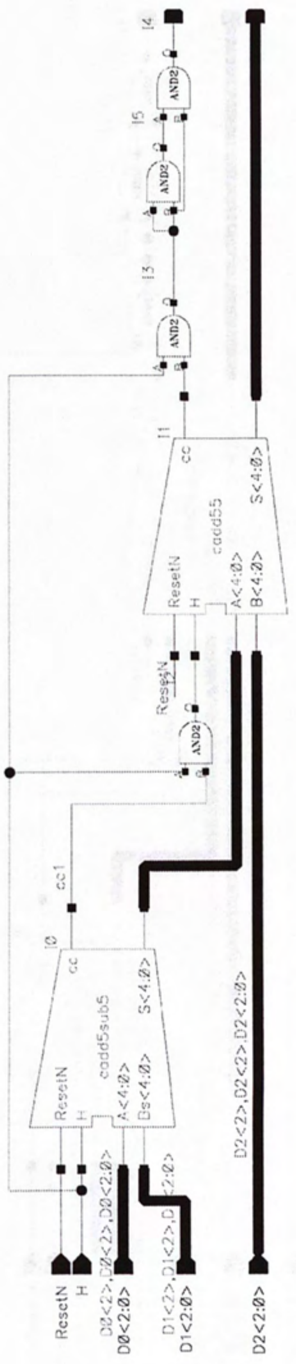
Schematic of Module BM000



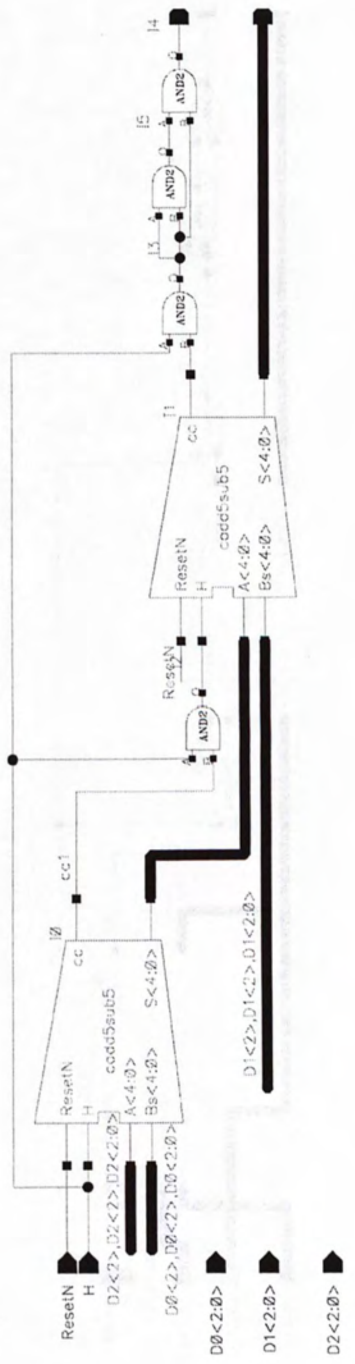
Schematic of module BM001



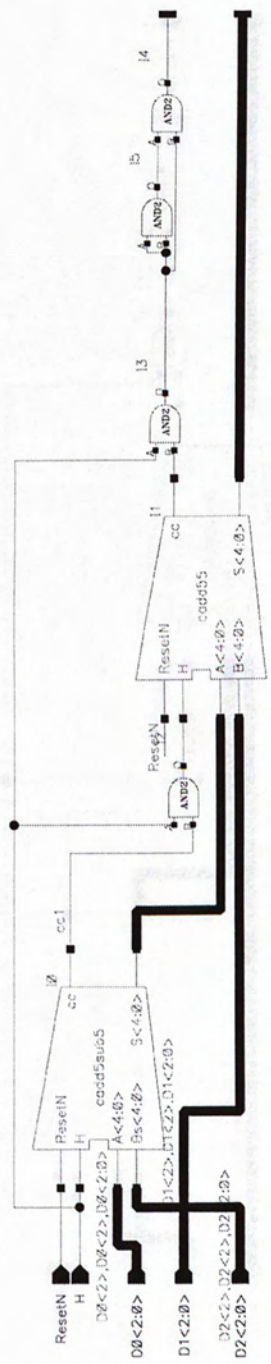
Schematic of module BM010



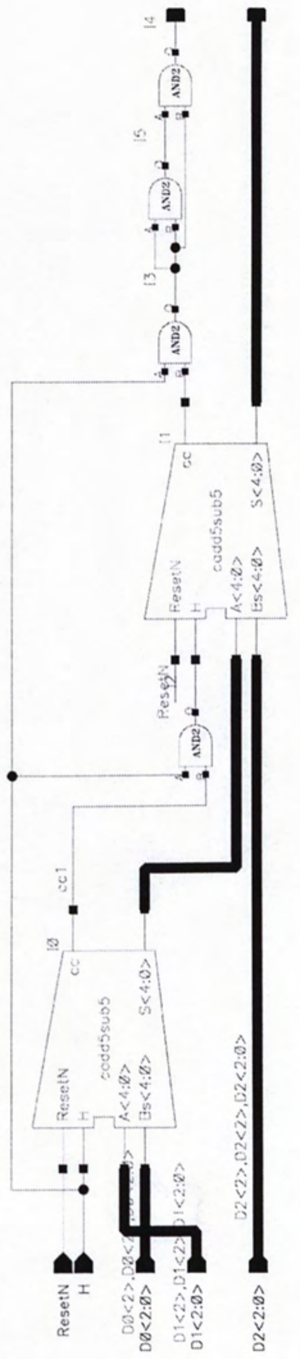
Schematic of module BM011



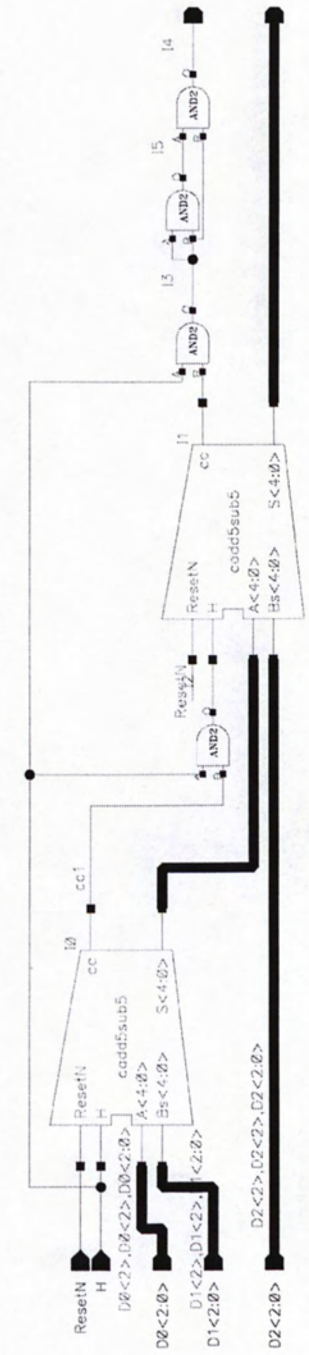
Schematic of module BM100



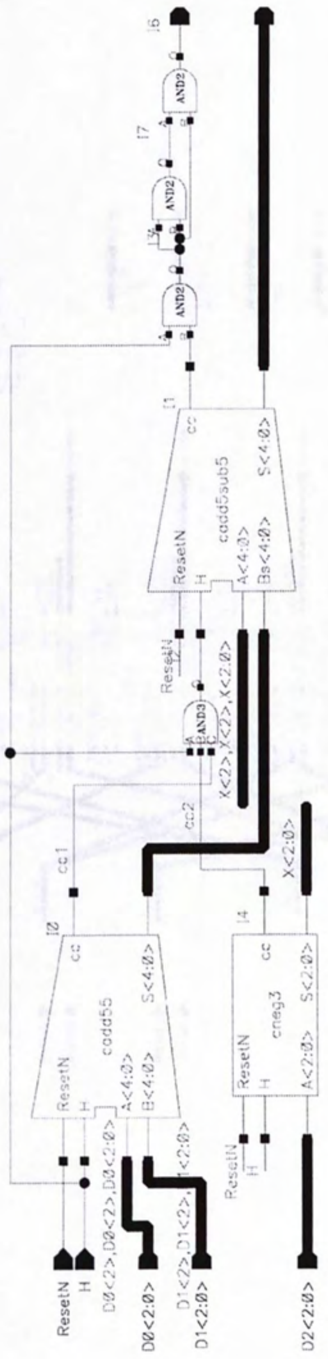
Schematic of module BM101



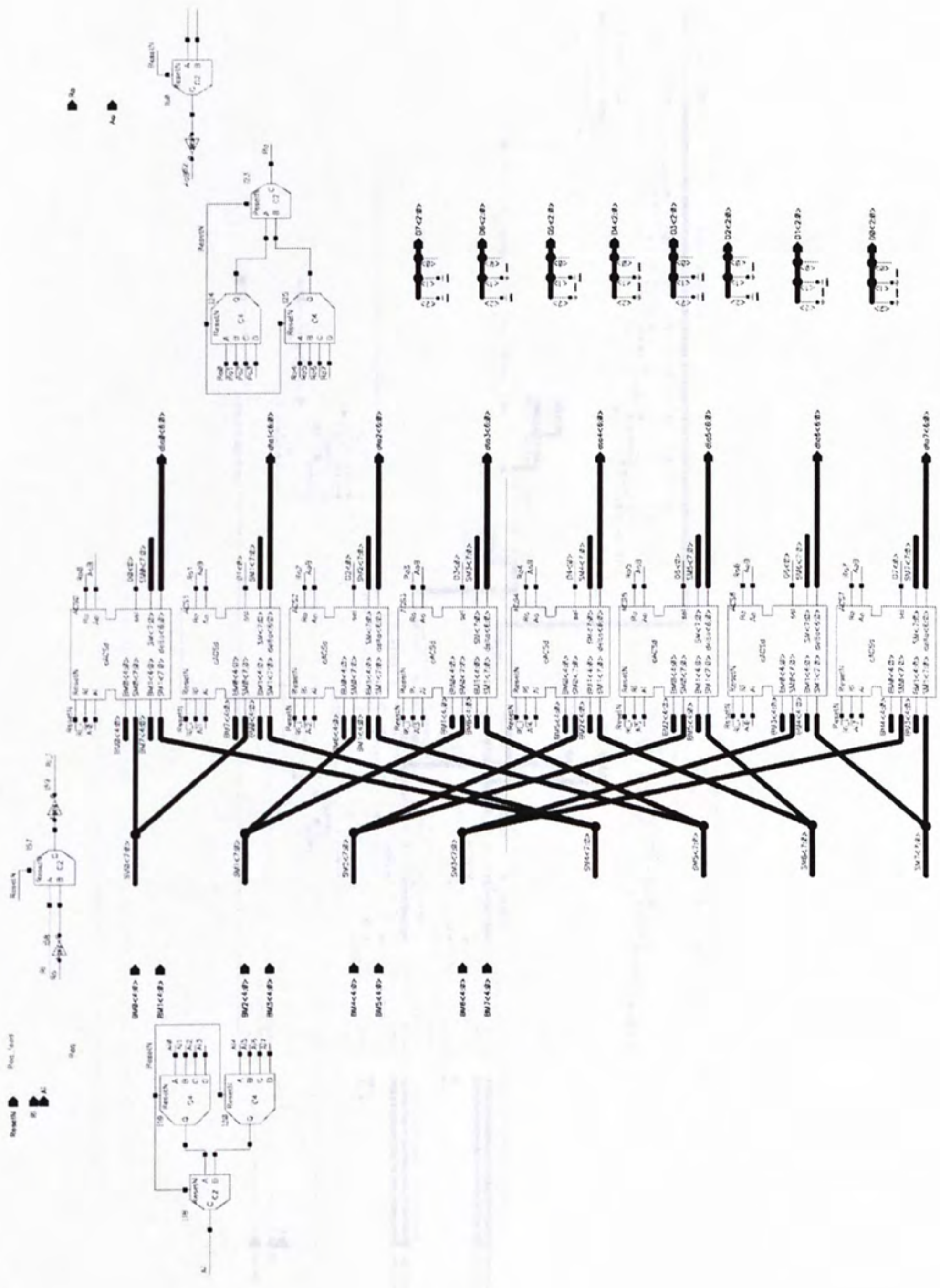
Schematic of module BM110



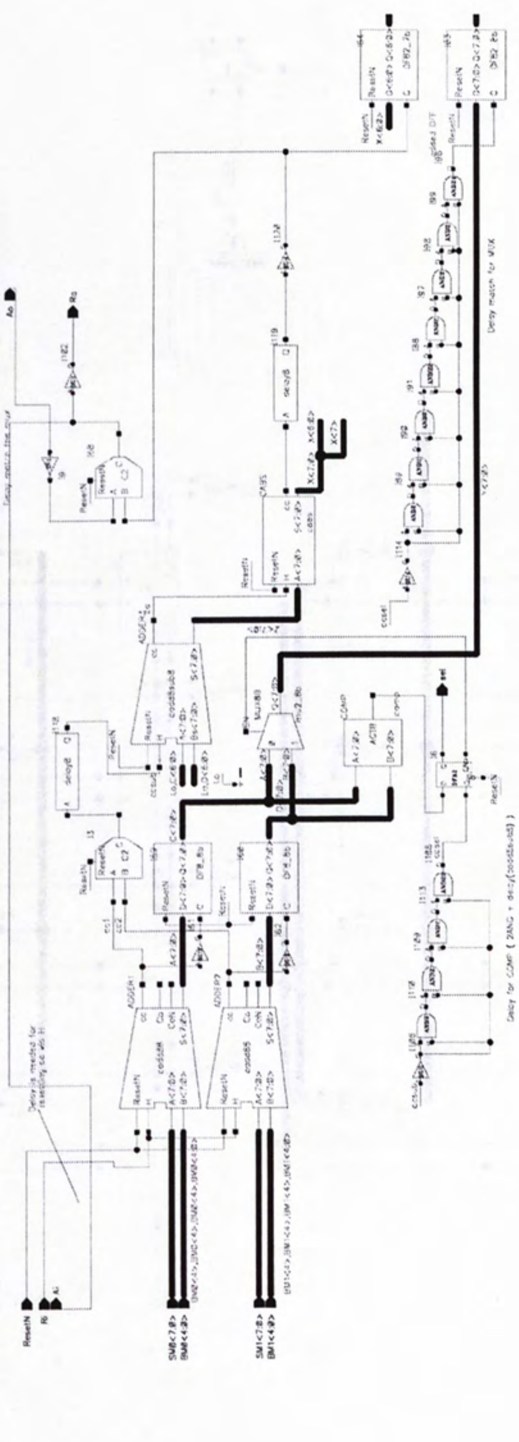
Schematic of module BM111



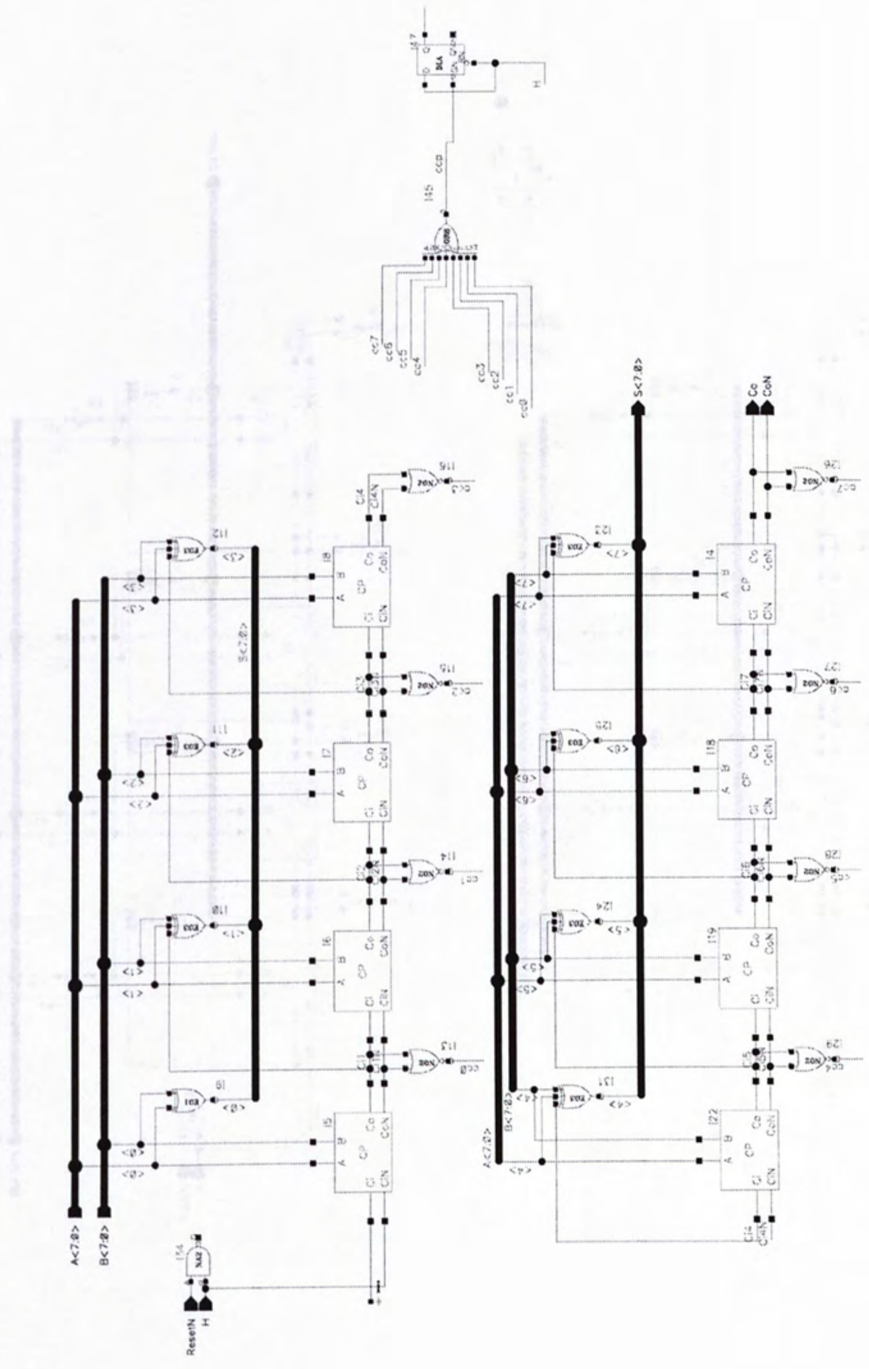
Schematic of ACSU



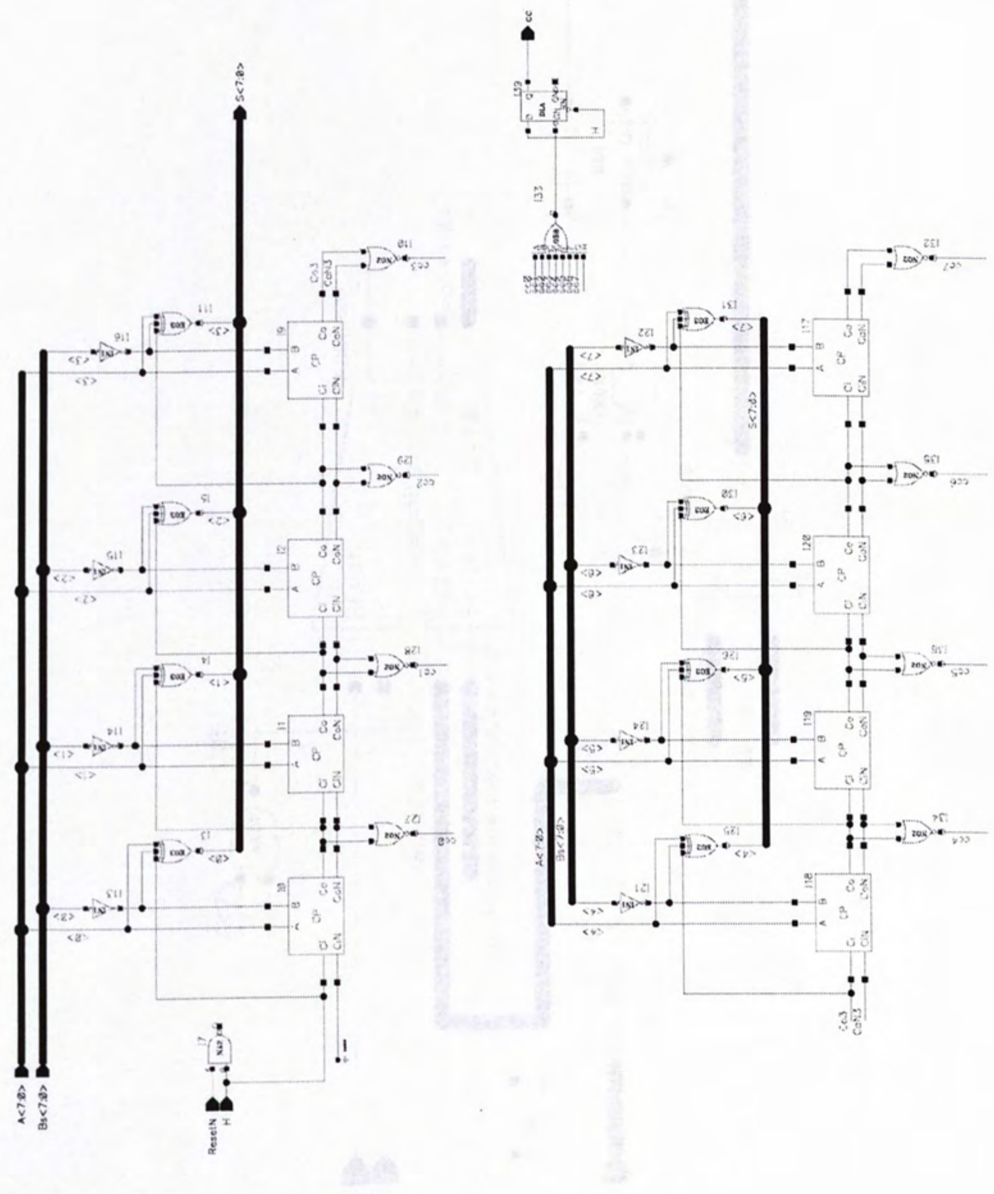
Schematic of cACSd



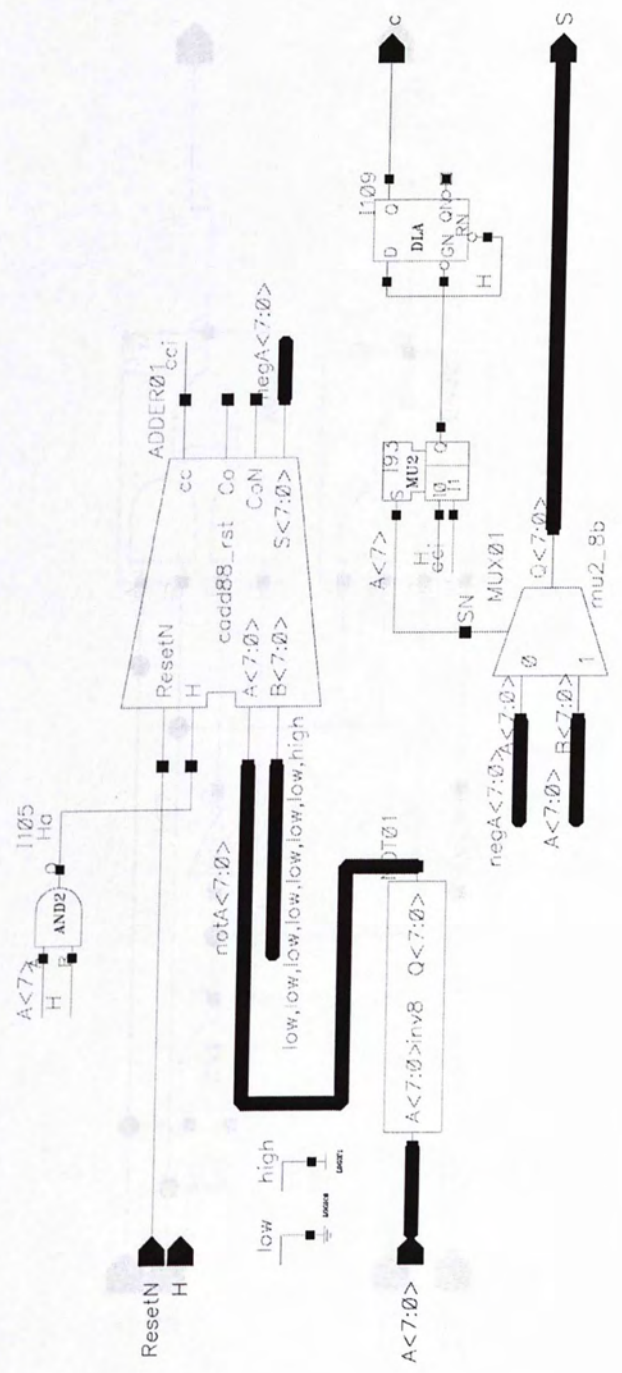
Schematic of cadd88



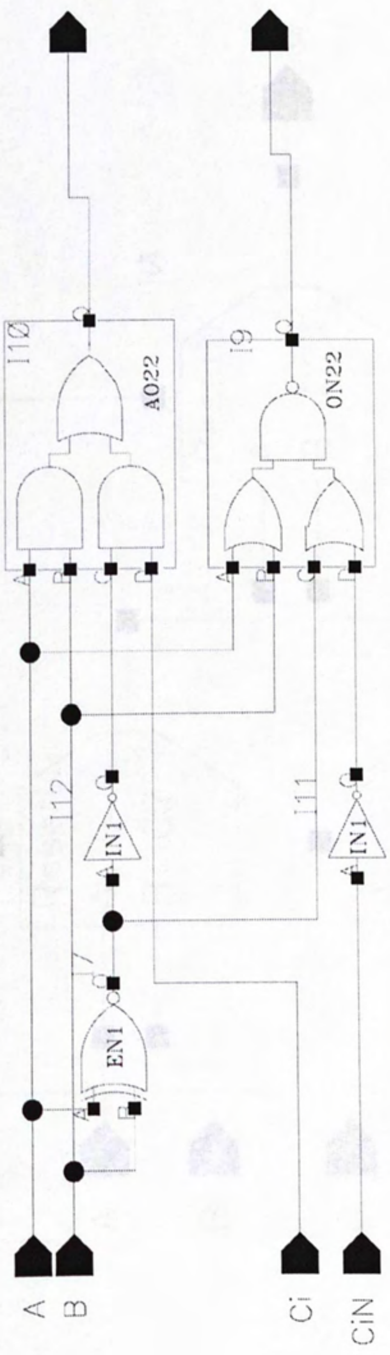
Schematic of cadd8sub8



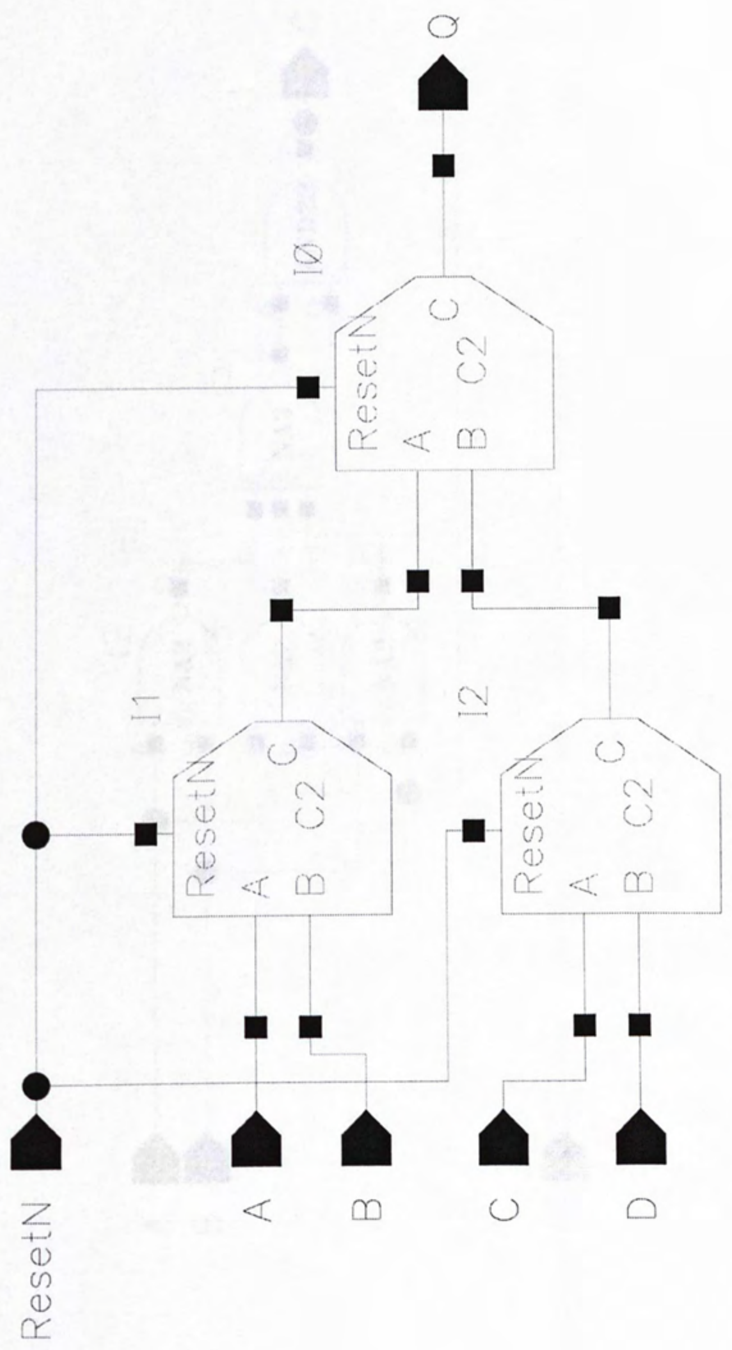
Schematic of cabs



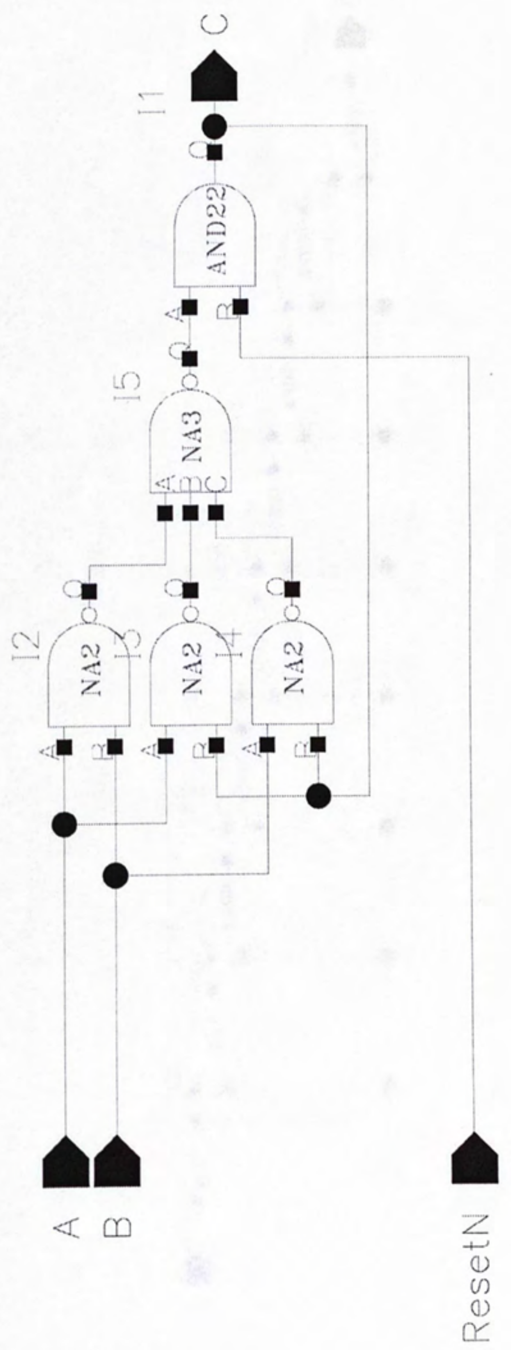
Schematic of CP



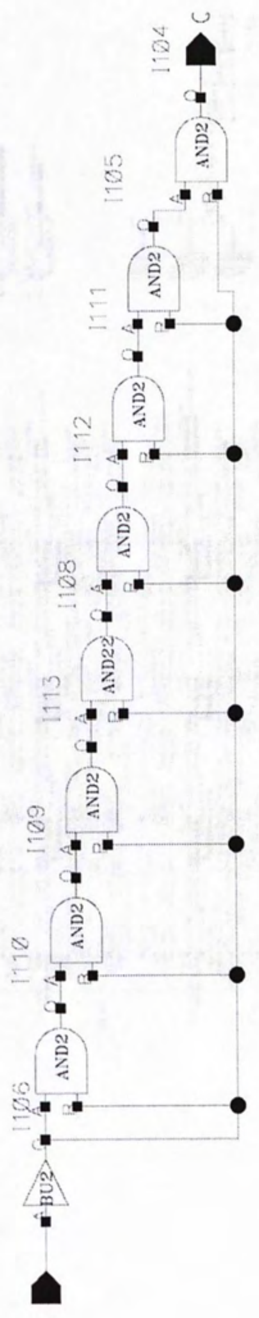
Schematic of C4



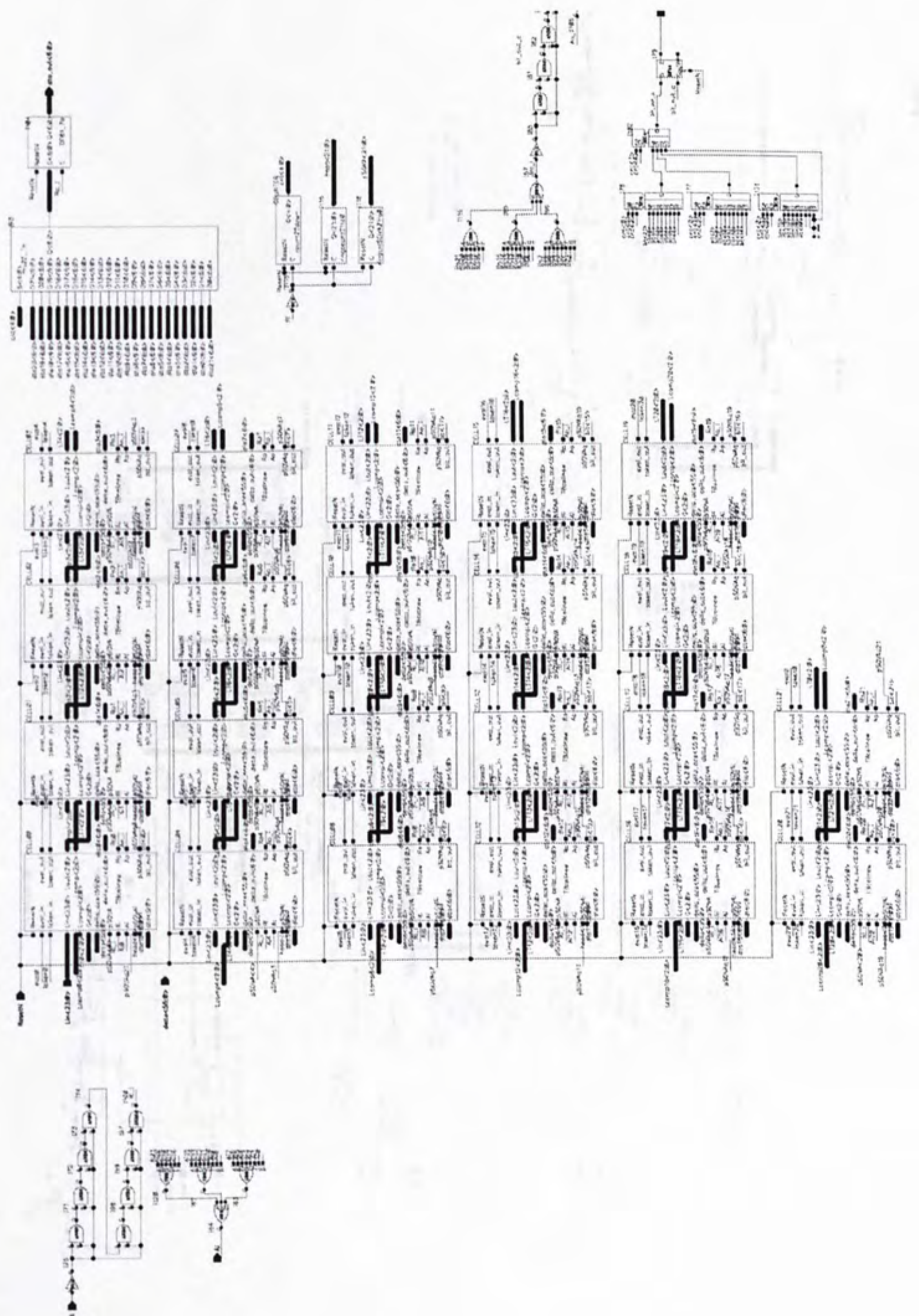
Schematic of C2



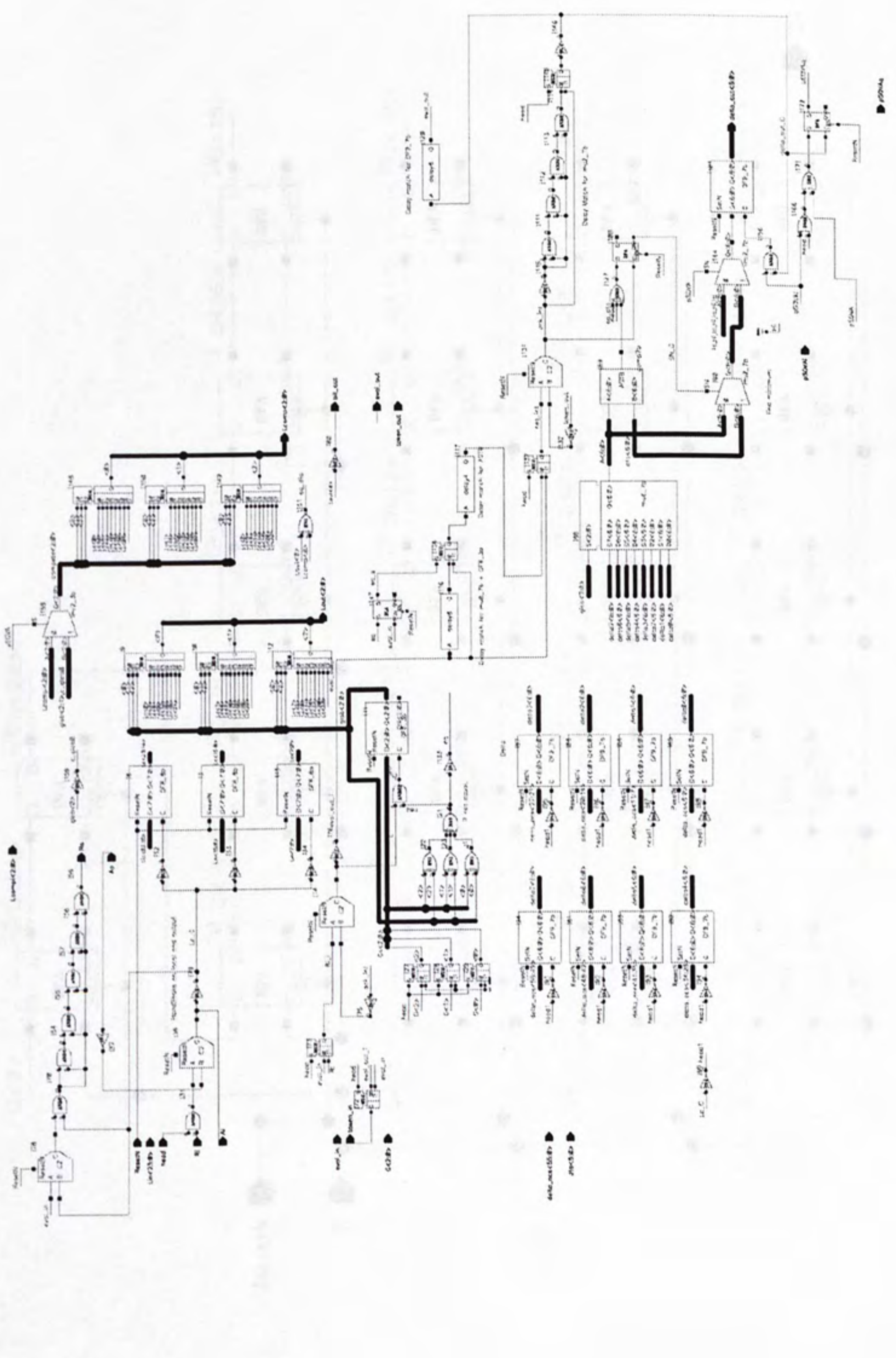
Schematic of delay8



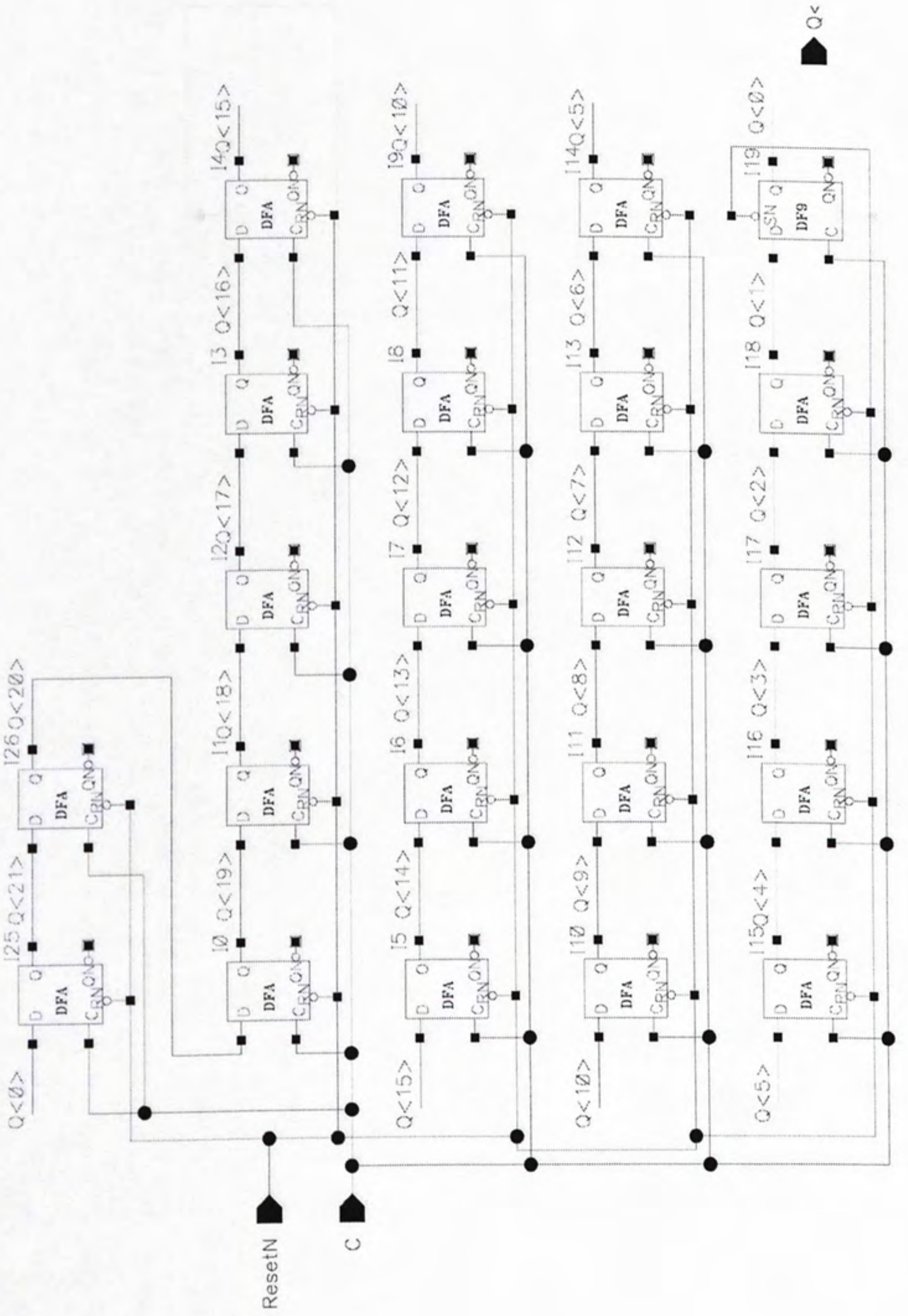
Schematic of TBnew



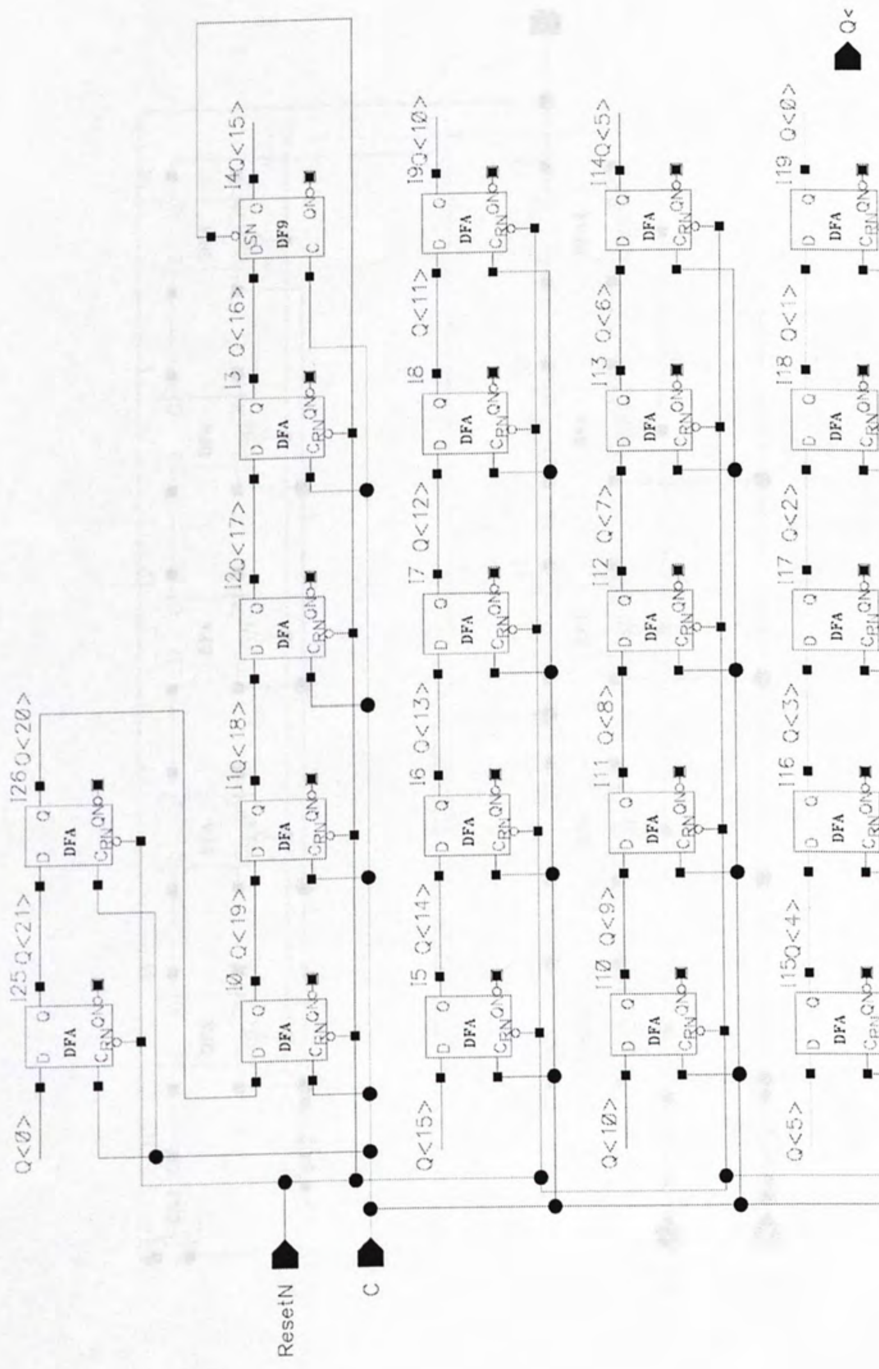
Schematic of TBcell



Schematic of ringcount21to0

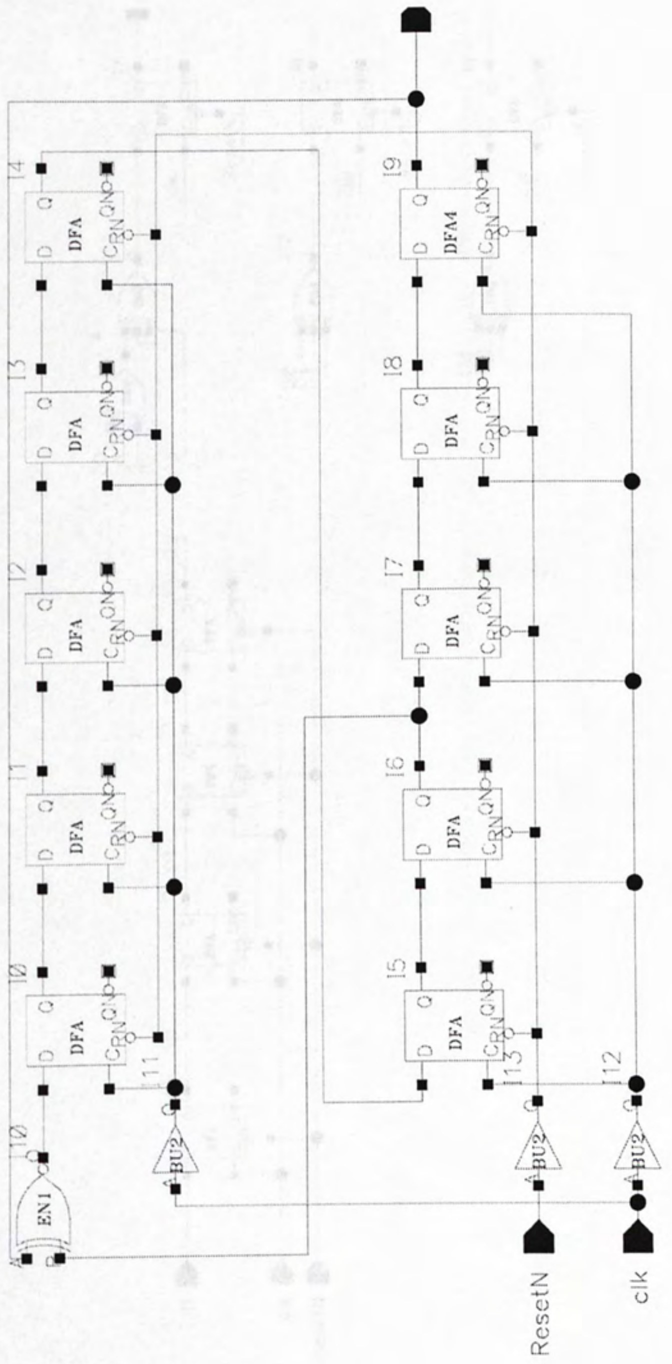


Schematic of rigncountSOVA21to0

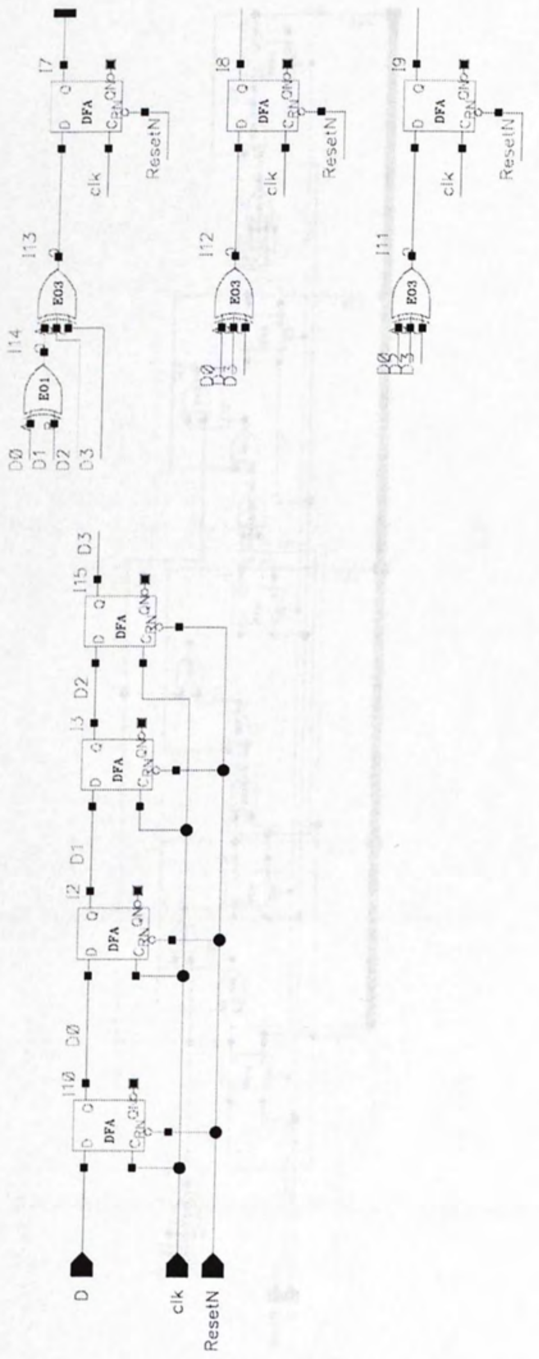


)

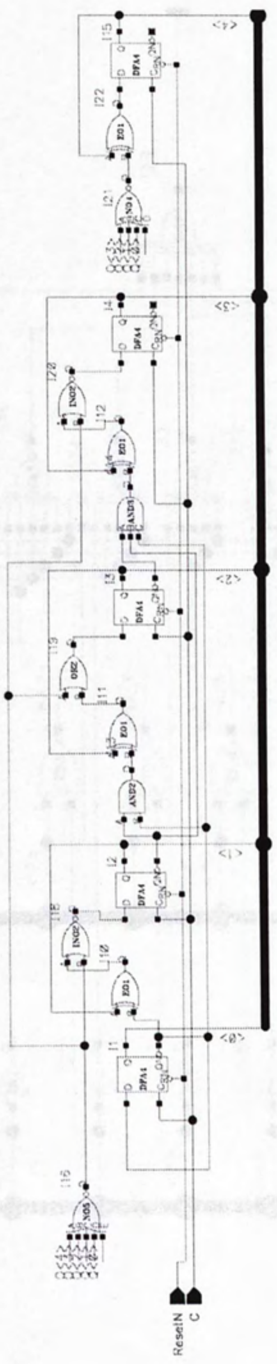
Schematic of LFSR



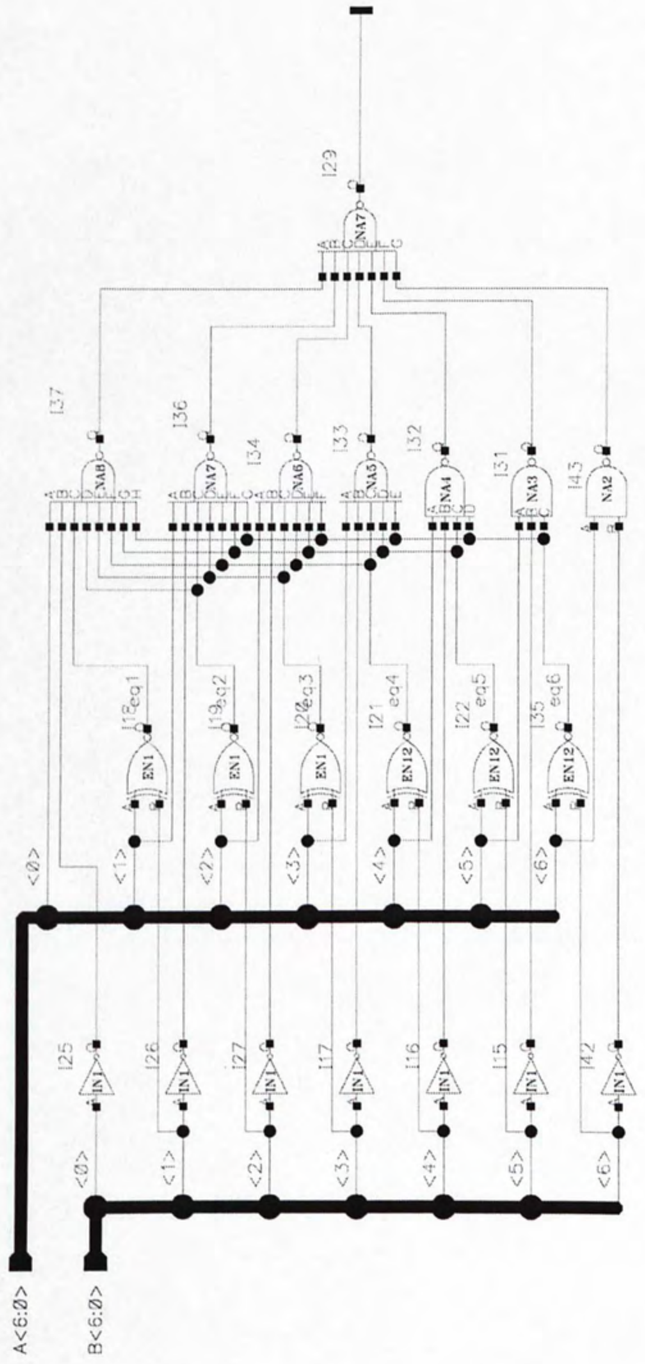
Schematic of conv_encoder



Schematic of count22down



Schematic of comp7b



A<6:0>
B<6:0>
I29
NA7
I37
NA8
I36
NA7
I34
NA6
I33
NA5
I32
NA4
I31
NA3
I43
NA2
I18 eq1
I19 eq2
I20 eq3
I21 eq4
I22 eq5
I35 eq6
EN1
EN12
I25
I26
I27
I17
I16
I15
I42
IN1
<0>
<1>
<2>
<3>
<4>
<5>
<6>