Decomposition and encoding of finite state machines for FPGA implementation

Decomposition and Encoding of Finite State Machines for FPGA Implementation

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. R.A. van Santen,
voor een commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
woensdag 15 december 2004 om 14.00 uur

door

Aleksander Stanisław Ślusarczyk

geboren te Warschau, Polen

Dit proefschrift is goedgekeurd door de promotoren:

prof.ir. M.P.J. Stevens
en
prof.dr.ir. R.H.J.M. Otten

Copromotor:
dr.ir. L. Jóźwiak

Druk: Universiteitsdrukkerij Eindhoven

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Slusarczyk, Aleksander S.

Decomposition and encoding of finite state machines for FPGA implementation /
by Aleksander Stanislaw Slusarczyk. – Eindhoven : Technische Universiteit Eindhoven,
2004.
Proefschrift. – ISBN 90-386-1663-5
NUR 959
Trefw.: sequentiele machines / decompositiemethoden / programmeerbare logische
schakelingen / logische schakelingen ; automatentheorie / logische schakelingen ; CAD.
Subject headings: state assignment / finite state machines / field programmable gate
arrays / sequential circuits / electronic design automation.

Summary

One of the most important steps in the synthesis of digital systems is the state encoding of Finite State Machines (FSMs). This step performs translation of the symbolic functional description of the system's architecture module into its binary representation. Therefore, it has profound influence on all quality aspects of the final binary logic implementation. Unfortunately, despite many research efforts, the problem of effective state encoding has not been so far satisfactorily solved for most implementation technologies.

This is in particular the case for the modern programmable implementation platforms, such as Field Programmable Gate Arrays (FPGAs), reconfigurable systems-on-chip (SoC) platforms and embedded FPGAs. These platforms possess unique characteristics, such as the availability of universal function blocks only limited by their number of inputs and outputs and the critical importance of interconnections, that have not been adequately addressed by the existing state encoding methods.

This thesis proposes and discusses a new approach to the problem of state encoding of FSMs targeting FPGA implementation and new method and electronic design automation tool that implements this approach. The proposed approach is based on FSM decomposition. Decomposition is a natural method of handling complex problems. It consists in splitting a complex problem into smaller, interrelated sub-problems and solving the network of sub-problems. Applied to FSMs, decomposition consists in realizing the behavior of a complex FSM as a network of smaller, collaborating FSMs.

To form sound theoretical base for the research on decomposition and encoding, this thesis presents extension of the existing General Decomposition Theorem to the case of multi-state realization of incompletely specified FSMs. This theorem covers as special cases all other known decomposition structures, such as serial and parallel decomposition, and the decomposition of discrete functions and relations. The theorem explicitly treats state and input information flows within the decomposed machine or function and thus supports the novel, information-driven approach to decomposition.

In this thesis, the problem of state encoding is also considered as a special case of general decomposition. This way, the extended decomposition theorem forms the basis for the formulation of flexible conditions for valid state encoding that enable implicit minimizations of the machine in the process of encoding and, in consequence, its efficient implementation. Based on the encoding validity conditions, a generic, implementation-platform-independent state assignment method is proposed.

This generic method is further augmented with the heuristics targeting efficient FPGA implementation. The heuristics exploit the information modeling apparatus of set systems used in formulation of the General Decomposition Theorem and analysis apparatus of information relationships and measures to perform analysis and optimiza-

tion of the information flows within the encoded machine during the encoding process. The heuristics were implemented in the software tool SECODE, which is discussed in detail.

The effectiveness of the decompositional, information-based approach to state encoding is analyzed by testing the developed encoding method and tool on a set of standard benchmarks and a number of generated FSMs exhibiting characteristics typical to the circuits encountered in various industrial applications. The presented experimental results indicate that the new approach proposed and researched in this thesis compares favorably with the existing state encoding methods.

Samenvatting

Een van de meest belangrijke stappen in de synthese van digitale systemen is toestandcodering van Finite State Machines (FSM's). Deze stap realiseert de vertaling van de symbolische functionele beschrijving van de systeemarchitectuurmodule naar de binaire realisatie daarvan. Daarom heeft het een diepgaande invloed op alle kwaliteitsaspecten van de definitieve implementatie van binaire logica. Helaas is ondanks vele onderzoekspogingen het probleem van de effectieve toestandcodering voor de meeste implementatie-technologieën tot nu toe nog niet naar tevredenheid opgelost.

Dit is met name het geval voor de moderne programmeerbare implementatie platforms, zoals Field Programmable Gate Arrays (FPGA's), aangepaste systems-on-chip (SoC) platforms en ingebedde FPGA's. Deze platforms bezitten unieke eigenschappen zoals de beschikbaarheid van universele functieblokken die slechts beperkt zijn door het aantal ingangen en uitgangen en het kritische belang van onderlinge verbindingen die door de bestaande toestandcoderingsmethodes nog niet goed genoeg geadresseerd zijn.

Dit proefschrift introduceert en behandelt een nieuwe aanpak van het probleem van toestandscodering van FSM's zich richtend op FPGA implementatie evenals een nieuwe methode en tool voor elektronische design automatizering die deze aanpak implementeren. De voorgestelde benadering is gebaseerd op FSM decompositie. Decompositie is een natuurlijke methode om met complexe problemen om te gaan. Het is gebaseerd op het delen van een complex probleem in kleinere, met elkaar in verband staande subproblemen en het oplossen van het netwerk van subproblemen. Toegepast op FSM's is decompositie gebaseerd op het realiseren van het gedrag van een complex FSM als een netwerk van kleinere samenwerkende FSM's.

Om een grondige theoretische basis voor het onderzoek naar decompositie en codering te leggen, introduceert dit proefschrift een uitbreiding van het bestaande General Decompositie Theorema voor wat betreft de multi-toestandrealisatie van incompleet gespecificeerde FSM's. Dit theorema dekt als bijzondere gevallen alle andere bekende decompositiestructuren zoals seriële en parallelle decompositie en decompositie van discrete functies en relaties. Dit theorema behandelt toestands- en ingangsinformatiestromen binnen de gedesintegreerde machine of functie en ondersteunt op deze manier de nieuwe informatiegedreven aanpak voor decompositie.

In dit proefschrift wordt het probleem van toestandcodering ook als een bijzonder geval van algemene decompositie beschouwd. Op deze manier vormt het uitgebreide decompositie theorema de basis voor de formulering van flexibele condities voor geldige toestandcodering die geïmpliceerde minimalizaties van de machine in het proces van codering en, als gevolg daarvan, de efficiënte implementatie daarvan mogelijk maken. Gebaseerd op de coderingsvaliditeitsvoorwaarden wordt een generieke implementatie-

platform-onafhankelijke toestandstoewijzingmethode voorgesteld.

Deze generieke methode wordt verder uitgewerkt door het ontwikkelen van de heuristische beslissingsmechanismen die zich richten op de efficiënte FPGA-implementatie. Deze heuristische beslissingsmechanismen maken gebruik van informatiemodelleringsgereedschap van set systemen dat in de formulering van het General Decomposition Theorema gebruikt wordt en tevens van analysegereedschap van informatierelaties en maten om analyse en optimalisering van de informatiestromen binnen de gecodeerde machine gedurende het coderingsproces ten uitvoer te brengen. De heuristische beslissingsmechanismen werden in de software tool SECODE geïmplementeerd; deze wordt in detail besproken.

De doeltreffendheid van de decompositionele, informatiegebaseerde aanpak van toestandcodering wordt geanalyseerd door het testen van de ontwikkelde coderingsmethode en coderingstool op een verzameling standaard benchmarks en een aantal gegenereerde FSM's die eigenschappen vertonen die typisch zijn voor de schakelingen waarmee men in diverse industriële toepassingen geconfronteerd werd. De nieuwe, in dit proefschrift voorgestelde en onderzochte aanpak levert betere resultaten op dan de reeds bestaande toestandcoderingsmethodes.

Acknowledgments

I would like to thank everyone who contributed to the work presented in this thesis.

I would like to thank prof. Mario Stevens, who sadly is no longer amongst us, for giving me the opportunity to carry out this research. I will remember the years in his CND group as an extremely stimulating and interesting period of my life. I also wish to thank prof. Ralph Otten for his assistance in the later stages of my work in ICS group and on this thesis.

My warmest thanks go to the co-promotor of this thesis, dr. Lech Jóźwiak, for his continuous support and for sharing his insight and enthusiasm during our countless discussions.

A special thanks is also directed to my fellow researchers – Artur Chojnacki, Szymon Biegański and Dominik Gawłowski – for the time we spent together at work and at leisure. I would also like to thank all members of CND and ES group, and especially the "coffee room club", for all the exchanges of views we had on all topics imaginable.

Rian van Gaalen and Marja de Mol-Regels deserve especially warm thank you for all the support they provide to the whole ICS group and especially to new foreign members, often struggling in the new environment.

Najserdeczniejsze podziękowania kieruję do Rodziców i do Uli. Jesteście dla mnie niewyczerpanym źródłem ciepła, radości i siły.

Contents

List of definitions

Chapter 1

Introduction

1.1 Digital Systems

The past several decades have witnessed explosive growth of integrated circuit technology and its applications in our everyday life. Faithful to the famous prediction of Moore's law, the capacity of integrated circuits (or chips) doubles every 18 months, delivering the devices that currently fit tens of millions of transistors on a few square centimeters of silicon and capable of performing billions of operations per second. The rapid technology revolution and enthusiastic acceptance of these devices have resulted in a significant price decrease that made it feasible to introduce digital systems into virtually every aspect of life.

The most prominent example of this trend has been the personal computer (PC), which evolved from an experimental, expensive curiosity to an ubiquitous and affordable work tool indispensable in a growing number of professions, as well as an entertainment and information source present in most of today's homes. But this best known example is just a tip of an iceberg of billions of chips produced every year and installed in modern devices, machines and facilities. The integrated circuits in production and measurement equipment help in work, making it easier and increasing our productivity. Consumer electronics devices such as television sets, video recorders and digital audio and video players have leveraged the advances in integrated circuit technology to deliver higher quality video and sound at ever decreasing price. Chips help to drive cars, fly planes, connect telephone calls and save lives in medical equipment. The latest trends embodied in terms such as "ambient intelligence" or "ubiquitous computing" predict expansion of computer chips even further, to equip with "intelligence" domestic appliances, lighting, even furniture and clothes.

All these developments have led consumers to expect advanced functionality features in new consumer devices and prompted the device vendors to deliver ever more sophisticated products. Also in the market of professional applications, the demands on the functionality and, often more importantly, cost, efficiency and power consumption of integrated circuits are growing. To remain competitive, companies must develop and enhance products with unprecedented constraints on cost, functionality, power consumption and, crucially, time-to-market.

To satisfy the demand, semiconductor industry has responded with a number of tech-

nology options to implement the digital circuits embedded in the newest products. These options represent different cost, performance and flexibility characteristics and enable a designer to make a choice of the most suitable technology and make different trade-offs based on the particular application.

Unfortunately, the explosive development of technology has not been matched by an equal progress in the field of designer productivity and software tools supporting the design of digital systems. The insufficient support offered by the Electronic Design Automation (EDA) tools is responsible for increased design times and suboptimal circuit realizations. This phenomenon known as "design gap" makes it more and more difficult for designers to take full advantage of the opportunities created by the technology. This is the reason for the continuous interest that EDA field has been receiving from the research community. The work presented in this thesis is also a result of this interest.

The subject of this thesis is circuit synthesis of finite state machines (FSMs) — one of the basic forms of description for digital system behavior. An FSM describes behavior of the system in terms of distinct states that the system can be in and transitions between the states that are triggered by particular input signals. The combination of the current state of the FSM and the input signals determine the output produced by the FSM and its next state.

The particular problem addressed by this thesis is state machine encoding, i.e. translation from symbolic, conceptual form to binary functions implementing the behavior described by the FSM. When encoding the machine, we are particularly targeting the implementation of the resulting logic in Field Programmable Gate Array (FPGA).

In this chapter we will present a short introduction to the available implementation options for sequential machines and, particularly, digital controllers, the design methodology used to implement the machines and look closer at the problem of state assignment.

1.2 Implementation technologies

Modern microelectronics industry offers to a digital system designer a number of implementation options for the system. Currently the most prominent of these options are:

- Standard general purpose processors, such as microprocessors, microcontrollers, DSPs, multimedia processors, etc. (software solution).
- Application specific integrated circuits — ASICs (hardware or hardware/software solution).
- Field programmable logic (hardware or hardware/software solution).

Since the subject of this thesis is related to FSM's hardware implementation, we will focus on the two hardware options — ASICs and programmable logic.

1.2.1 ASICs

Due to the complexity and the cost of the chip production technology, application specific integrated circuits are usually not manufactured directly by the system designer, but by a specialized *silicon foundry* — a firm that receives from the designer the design of the

chip and manufactures the required volume of chips based on this design. Depending on the level of detail the design requires, several options are available. They range from full custom design, where designer specifies all features of single transistors on the chip; through standard cell approach, where a number of functional cells is used in the design; to gate array, where only connections between prefabricated functional blocks can be determined.

The **full custom** option involves design of the integrated circuit at the lowest level of single transistors, or even particular features of a single transistor. On silicon transistors are manufactured in multiple layers, each layer corresponding to a part of the transistor (e.g. well, source and drain, gate, contacts) and a number of layers where interconnections are realized. Full custom design requires separate definition of each layer, which usually involves drawing rectangles corresponding to particular elements of the transistor. Hence the popular term "polygon pushing" used to describe the process. This approach allows designer to determine position and size of the transistor and therefore gives ultimate control over speed, area and power dissipation of the circuit. The price for the level of control achieved in full custom designs is the huge effort required to design any but the simplest circuits. Moreover, this effort is usually required for each separate design, as the results of the previous designs cannot be readily reused. The large non-recurring engineering (NRE) costs induced by this labour-intensive process make this approach only practical for the most performance-, density- or power-constrained devices, or for very high volume parts. It can also be applied for large regular structures such as memories, where the design of a single cell is multiplied millions of times and thus any saving gained by hand-crafting the cell offers significant reduction of the whole structure.

To avoid the effort associated with designing each separate transistor by hand, the **standard cell** approach has been introduced. In this methodology, the circuit is constructed from a number of pre-designed standard cells. The cell library, usually delivered by the silicon foundry and optimized for the foundry's process, includes basic logic building blocks, such as logic gates, flip-flops, registers, and often more complex functions, such as multiplexers, ALUs, multipliers and memories. The library often includes several implementations of a particular function, each implementation optimized for a different parameter, such as area, delay, power, drive strength, etc. While the regular shape of the cells gives rise to regular, possibly suboptimal layout and wiring, the highly optimized implementation of particular cells may make up for that effect and often the circuits of quality close to full custom design are produced. This is achieved at a fraction of the design costs, as much of the work of mapping the design to cells, placement and routing are automated and may only require some designer intervention for particularly difficult or critical elements.

While standard cell approach significantly reduces the design cost of a digital system, it does not address another crucial cost ingredient — the fabrication. As mentioned before, transistors are realized in silicon in layers. Each layer is created in a photolithography process by exposing the silicon covered with photoresist to ultraviolet light via a mask. The exposed regions of the photoresist change their chemical properties and can be washed away with a developer solution, leaving the unexposed regions untouched. Since there are multiple layers, multiple masks are necessary to fabricate a working chip. Unfortunately, masks are high-precision, expensive devices, with the cost for a complete set of masks reaching $1 million and more. The chilling cost effect is compounded by

the fact that the masks in general cannot be corrected or modified. Therefore, if a flaw is detected in the design after the masks have been fabricated, the mask set needs to be replaced. To avoid this very costly operation, extensive tests, verification and simulations need to be performed in the design stage, increasing the development time and costs even further.

The problem of high mask costs motivated the development of **Mask Programmable Gate Arrays (MPGA)**. In this technology, a chip containing an array of unconnected gates is prefabricated by the foundry. The customer's design is mapped onto the gate fabric and the interconnection structure necessary to realize the desired functionality is determined. In this process chip-specific masks are only necessary for the contact and interconnection layers, thereby reducing the unique mask set to 2-5 masks. Naturally, there is very little room for optimizations in this methodology. Also, for some designs significant amount of prefabricated logic may never be used. Still, this option is attractive for less demanding applications. Except for the reduced mask cost, the chip cost is reduced and the reliability is improved due to the fact that the foundry produces large numbers of identical prefabricated wafers which can be used by multiple customers.

Recently, the MPGA idea has been revitalized by the so called structured ASIC approach. It is similar to the original MPGA idea as it also offers prefabricated logic with the programming taking place at the interconnect level. However, structured ASICs contain more coarse-grain logic. The prefabricated logic includes complex programmable functions, flip-flops, memories, etc.

1.2.2 Programmable logic

The reduced mask cost offered by the MPGA or structured ASIC methodology still does not solve most of the fundamental problems of the fixed logic implementations related to design and fabrication cost, flexibility, time-to-market etc. Still, any flaws in the original design are severely penalized if discovered in the manufacturing stages. This is also the case for other updates to the design, involving additional features or modified functionality due to changing requirements, standards, etc. Also, the typical turn around time of a foundry is measured in weeks, which hurts the increasingly important time-to-market. At the same time, foundries will set a minimum limit on the number of parts produced. In the times of fierce competition, when not all products turn out to be immediate success, this may leave the vendor with a large and expensive stock of unsold chips. Moreover, the design tools and expertise required to design an ASIC are expensive resources in general unavailable to small enterprises.

An answer to these problems is offered by programmable logic. As the name implies, logic functions of the devices in this methodology can be programmed rather than prefabricated. Moreover, the whole programming process takes place at the designer's site, or even in the actual working system where the chip is used, rather than in the foundry. After the circuit has been synthesized and an appropriate programming file generated, the programming of the chip can be performed on-site and takes just seconds. This way, the production turn around time is dramatically reduced. Many of the programmable devices offer re-programmability as well, what eliminates problems with bugfixes and functionality updates to the chip. If a different logic behavior is required, a new programming file can be downloaded to a chip and the new functionality is available in seconds. Nowadays, these updates can be performed remotely in a working system,

via Internet, wireless communication links or other connection. Another important factor in the success of programmable logic is the availability and relative ease of use of the corresponding software design tools. There are many software tools available in the market that help design programmable circuits even by the designers with minimum experience. Moreover, these tools are very cheap when compared to any ASIC-flow tools. In fact, some device vendors offer free design packages with their products.

The price paid for low NRE costs, flexibility and short time-to-market is a lower logic density, lower speed and higher power dissipation of programmable logic as compared to ASICs. The simple and cheap push-of-a-button design tools do not produce the circuits approaching the quality or performance of the designs hand-crafted by the expert designer. Also, a single programmable chip is more expensive than a single ASIC in a long production series. While this is unacceptable for high-end manufacturers offering high volume products, for smaller vendors planning production of several hundred or thousand units the performance may be just enough and a small number of relatively expensive programmable devices may still cost less than NRE cost associated with ASIC production.

The two dominant programmable logic technologies are Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs).

CPLD

The popular CPLDs evolved as an extension of simple Programmable Logic Array (PLA). These arrays are optimized for realization of logic as a two-level Boolean sum-of-products expression. PLA device (see Fig. 1.1) is composed of the programmable AND-plane, where input signals available in direct and negated polarity are combined to form product

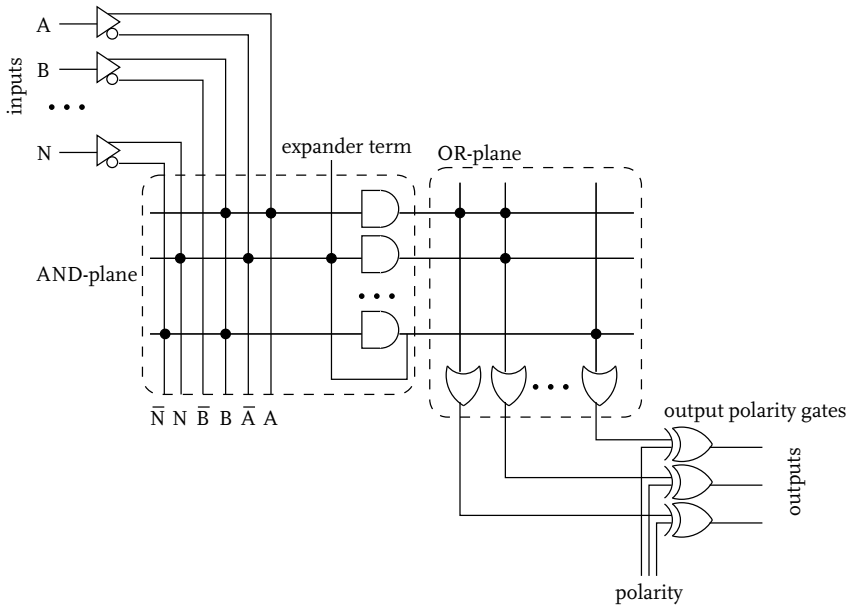


Figure 1.1. PLA structure

terms, and programmable OR-plane which performs logical-or of the product terms from AND-plane. Often, a set of output XOR-gates is available for programmable output signal polarity. The prevalent programming technology for PLA is Electrically Erasable Programmable Read-Only Memory (EEPROM). In this technology floating gate transistors are used to program connections. A transistor can be permanently switched-off by applying appropriate voltage, thus breaking the connection. This process is fully reversible by applying the reverse voltage, making the circuit re-programmable. While PLA-based devices consist of AND-plane and OR-plane, in Programmable Array Logic (PAL, see Fig. 1.2) devices, the OR-plane is replaced by fixed OR-gates, with programmable input and output interconnections. The CPLD devices typically include a number of PLA-like or PAL-like structures called macrocells, together with programmable interconnections between the cells.

This type of architecture is represented by the popular MAX devices from Altera Cor-

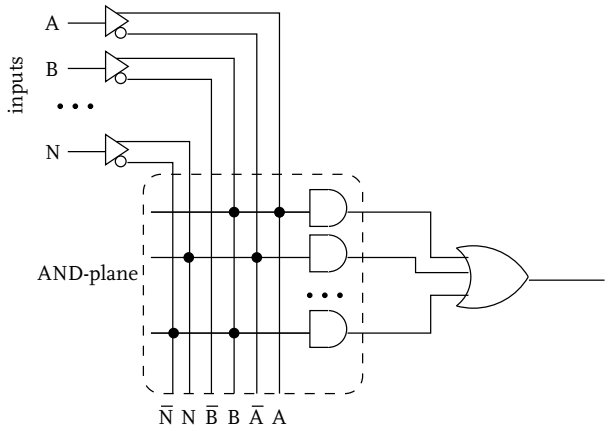


Figure 1.2. PAL structure

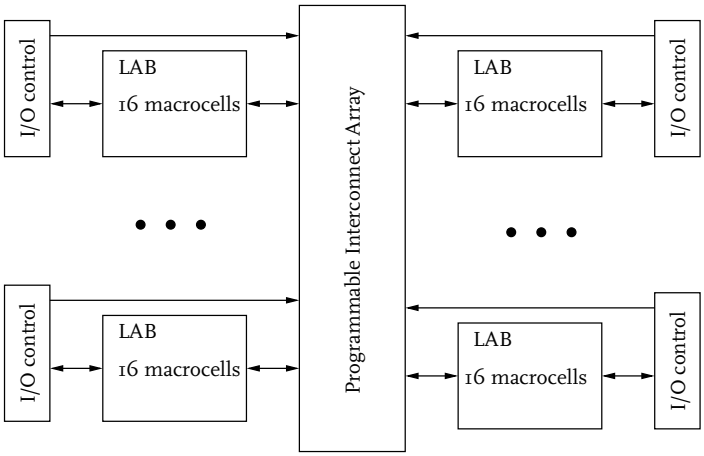


Figure 1.3. Altera MAX CPLD

poration. In the MAX 7000 devices (depicted in Fig. 1.3), the macrocells are organized in the Logic Array Blocks (LABs), 16 macrocells per LAB. The LABs are interconnected by means of Programmable Interconnect Array (PIA) that provides full connectivity between LABs. Each macrocell within a LAB contains a PAL element capable of realizing sum of 5 product terms of 36 external signals and 16 so-called shared expander signals. The external signals can come from the PIA or the I/O control blocks that interface with the device's pins. Shared expanders can be used to feed output of one macrocell within a LAB to an input of another macrocell, allowing for realization of complex logic expressions. Additionally, a macrocell output signal may registered in the configurable flip-flop available in the macrocell. The devices of 7000 family include 2 to 32 LABs.

CPLD devices offer relatively small logic density and capacity, with the largest devices having logic capacity equivalent to 10,000 gates. However, the simplicity of structure results in very fast circuits and predictable timing characteristics, making CPLD suitable for synthesis of timing-critical control applications. Naturally, the small capacity is also reflected in the low price of the devices, which makes CPLDs attractive for tight-budget applications.

FPGA

A more prominent place in the programmable logic market is taken by the Field Programmable Logic Array (FPGA) technology. In these devices, a large number of configurable logic blocks capable of realizing complex functions of a small number of inputs is connected by means of programmable interconnection fabric. The typical FPGA architecture is represented by the devices from Virtex family manufactured by Xilinx. A Virtex device comprises of logic blocks laid out in regular matrix, interconnection resources routed between the matrix rows and columns and I/O blocks at the perimeter of the chip. All these elements are programmable to allow realization of complex logical functions

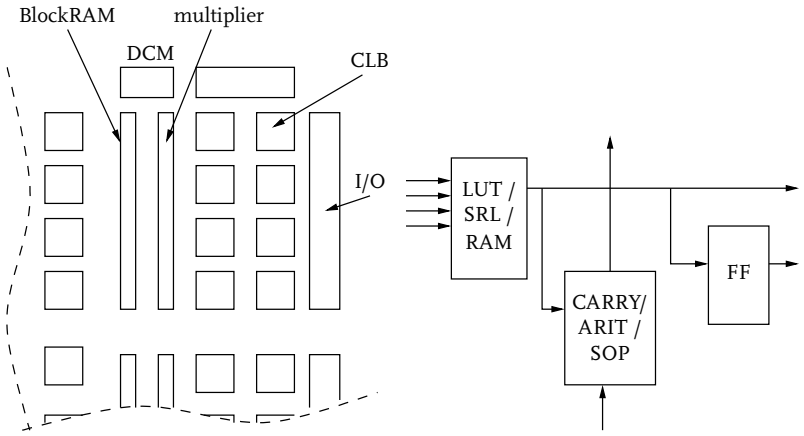


Figure 1.4. Virtex II FPGA

with flexible interconnections and communication with the outside world using multitude of I/O standards. The programming of the device is achieved by filling Static RAM (SRAM) memory cells controlling the behavior of different resources.

The primary logic resource of Virtex FPGA is a slice. It includes two 4-input function generators which can be configured to be used as a 4-input Look Up Table (LUT), 16-bit shift register, or 16-bit RAM element. A half of a slice is schematically depicted in Fig. 1.4(b). In the LUT configuration, the function generator can be programmed to realize any logical function of 4 inputs, by storing output value for each of the 16 input combinations. Groups of four slices form Configurable Logic Blocks (CLBs). Slices within CLB have fast local interconnections and can be combined by means of dedicated multiplexers to realize any function of up to eight inputs, with some functions of a larger number of inputs also possible. In addition to LUTs, a slice contains two configurable flip-flops and a number of dedicated logic gates: XOR gates for implementation of adders, AND gates for simplified multiplier designer and OR gates for realization of large sum-of-product expressions. The fast local interconnections between the slices in a CLB include two carry chains with associated multiplexors used to implement addition and subtraction, or to cascade LUTs to implement wide logic functions.

In addition to CLBs for implementation of random logic, Virtex FPGAs contain a number of other dedicated, programmable resources. These include:

- Block SelectRAM — 18 Kbit memories with configurable word and address widths and dual-port capabilities
- embedded 18-bit multipliers
- Digital Clock Management circuitry for programmable clock signal division and multiplication, delay compensation and phase shifting
- (Virtex II Pro family) up to four general-purpose PowerPC processor cores integrated within the FPGA fabric

Top-of-the-range Virtex II Pro FPGA currently contains over 14,000 CLBs with logic capacity of about 8 million logic gates, 10 Mbits of Block RAM, 556 multipliers, and 4 PowerPC processor cores.

The hierarchical programmable interconnection structure connects logic elements together. In particular, in addition to fast inter-slice connections, each CLBs has direct connections to all its immediate neighbors and a limited number of connections to every second, third and sixth block away from the CLB in all four directions. For long-range communications, a number of global signals routed along the entire width and height of the chip is available. Other global communication resources include eight dedicated low-skew clock networks and three-state buses. For off-chip communications I/O blocks are available that are configurable for multiple voltage and impedance standards. Dedicated high-speed transceivers allow inter-chip serial communication at 3.125 Gbps.

1.2.3 Summary

In this section, we outlined characteristics of two basic hardware implementation strategies for digital systems: with fixed and programmable logic.

The fixed logic methodology, due to its inherently high design and manufacturing costs is especially suitable for high-performance, high-volume products. Still, the high logic density and performance levels of ASICs continue to make it the dominant technology for digital systems implementation in parallel to standard general purpose programmable processors.

However, in the recent years this picture has begun to change. Programmable logic solutions, in particular FPGAs, have reached maturity and begin to challenge ASICs. Latest families of FPGA devices reach performance and logic density levels that is sufficient for many applications. Elimination of manufacturing costs and dramatical reduction of turn-around times make FPGAs particularly attractive for short-series, high-end industrial applications. In the markets such as telecommunication, networking, broadcasting, or medical instrumentation, several thousands of dollars paid for an advanced FPGA device is more than justified by the saved design and manufacturing time and the opportunity for product updates throughout its lifetime.

Interestingly, important market segment for FPGAs is created by the ASIC designers. Programmable logic has become a standard way of emulating and testing of ASIC designs before manufacturing. Instead of performing the time-consuming software simulation of an ASIC, it can be implemented in FPGA and tested in circuit, and at speed, increasing simulation speed and reliability.

In addition to time and cost constraints, other economical and technical factors are playing in favor of FPGA adoption. FPGAs are mass-produced, highly reliable devices manufactured in the cutting edge processes, otherwise unavailable to most designs. The ability to buy and program in-house arbitrary number of FPGA devices frees designers from close ties with the foundries, allowing flexible stock control and protecting their intellectual property.

Except for the above mentioned reasons, which essentially promote FPGAs as short-series replacement for ASICs, programmable logic with its in-system, even on-the-fly re-programmability introduces a new design paradigm, unavailable for ASIC implementations. FPGA-based hardware has a potential to change its programming and, in result, behavior adopting this way to new operating conditions. Experiments in evolutionary hardware [20], where a part of FPGA is programmed to re-program other part in reaction to changing conditions, are showing just one of the possible development directions.

During the last decade the reconfigurable systems evolved from a cost-effective replacement of ASICs to the mainstream implementation option of application specific (embedded) systems in parallel to programmable general purpose processors and ASICs. Also, emerging FPGA-based system-on-chip structures are ideally suited for implementation of modern system-on-chip designs. Devices such as Virtex II Pro include in one chip general purpose processors for control and non-critical computations, programmable logic for hardware acceleration of timing- and power-critical tasks, reasonably large memories for storage, multipliers for arithmetical and DSP applications, and high speed transceivers for communication. These characteristics allow implementation of the entire system on just one chip, increasing its speed and reliability, decreasing power dissipation and eliminating costs and inefficiencies associated with multi-chip or general purpose processor implementation.

All of the above reasons indicate large potential and quickly growing importance of the FPGA technology and motivate intensive research in the areas associated with FPGA-based system design. In particular, much of the research effort is invested to deliver high-

quality, easy to use design tools capable of producing efficient FPGA implementation.

1.3 FPGA design flow

In the previous section, we pointed out that the availability of cheap and easy to use design tools is one of the key factors to FPGA success. These tools enable description of the system in a high level language or in a graphical form and automatically perform synthesis of the circuit with the described behavior. The typical synthesis flow is presented in Fig. 1.5.

The input to a typical FPGA synthesis tool is a specification of the system using one of *hardware (or system) description languages*. The most popular method of description is by means of a Hardware Description Language (HDL), such as Verilog [26] or VHDL [27]. In these languages the system can be described by a mixed behavioral and structural model, with algorithmic description of operation sequences, control decisions, I/O operations etc.

Another form of behavior specification is a *finite state machine* (FSM), which describes system in terms of states that the system can be in, transitions between the states triggered by received input signals and the output values produced by the system based on current state and input values. An FSM can be described in a tabular form by a state transition table (STT), in a graphical form by a state transition graph (STG), or by an appropriate HDL construct.

A lower-level system specification can also be provided by means of Register Transfer Level (RTL) description. In RTL, designer specifies the system in terms of registers and operations on data flowing between the registers. These operations can be specified as abstract data operations, such as addition or multiplication, or the actual networks of gates realizing the data transformation. The RTL description can be provided in HDL or in graphical form as a circuit diagram.

Less popular specification options include *formal description languages* represented by generalizations of FSMs, such as statecharts [21], Estelle [28], SDL [18] or Promela [24]. For these languages, separate tools may be provided that generate equivalent HDL or netlist descriptions. The generated circuit description is then synthesized in the regular flow.

The system specification expressed in one of the abovementioned forms is processed by an appropriate compiler and converted to a network of combinational logic blocks and finite state machines realizing the required behavior. To implement the FSM behavior, the structure depicted in Fig. 1.6 is used. The current state of the machine is stored in the register. Based on the current state and the inputs, the combinational logic block computes output values and the next state of the machine, which is fed back to the register. However, the states of an FSM are present in the original design as symbolic state names. To be able to store the state in a binary register, binary vector (code) corresponding to each of the machine's states needs to be determined. This task is performed by *state assignment* (or state *encoding*).

After state assignment, the entire system is described by an RTL-level netlist of registers and Boolean functions. (The functions that can be realized using dedicated logic resources, such as multipliers or carry chains, are usually directly instantiated in the original system description or in the RTL netlist by the design compiler and are not the

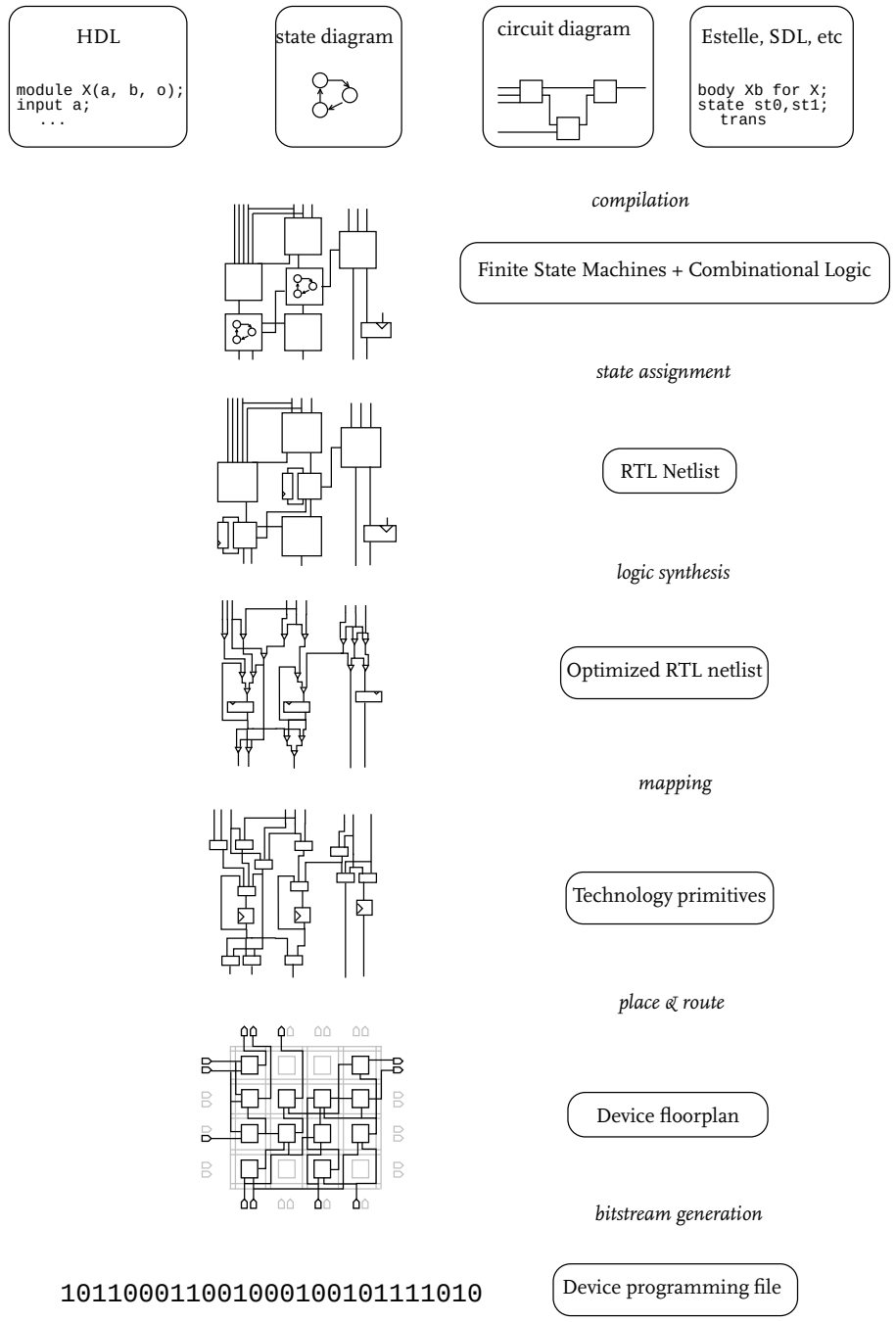


Figure 1.5. FPGA synthesis flow

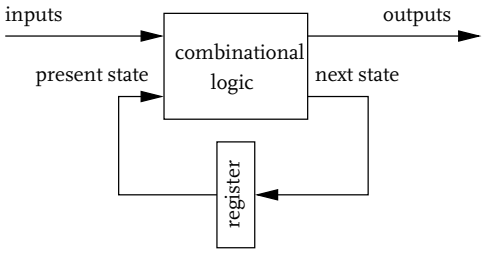


Figure 1.6. FSM implementation structure

subject of further optimization.) In the next step, *logic synthesis* processes the Boolean functions to find their effective and efficient realization. Usually, logic synthesis will only perform partial optimizations of the functions that are expected to result in efficient implementation. The task of actual realization of the functions is in this approach deferred to the *technology mapping* phase, where the simplified functions are realized by logic blocks available in the target implementation platform (so called technology primitives). In particular, in LUT-based FPGA, such as Virtex, the logic functions are mapped to 4-input LUTs, multiplexors and gates available in a slice.

The result of technology mapping is a network of technology primitives. These primitives are then assigned to particular slices within the target FPGA device and the interconnection structure between the slices is determined. These processes are referred to as *placement and routing*. They ultimately determine how particular logic resources and interconnections within device need to be configured and result in so-called *bitfile* or programming file that contains complete information necessary to program the target device.

1.3.1 State assignment

The main subject of this thesis is the state assignment of finite state machines. Our interest in the subject stems from the fact that the influence of state assignment on the circuit realization goes far beyond simple association of state names and binary codes.

As depicted in Fig. 1.6, bits of the binary vector representing current state are used both as inputs and outputs of the combinational logic component. Consequently, they have crucial influence on the form of binary functions necessary to compute outputs and next states. As a result, the complexity of the combinational component for different encodings can differ dramatically. This is the reason why the topic has attracted attention of researchers for the past forty years.

However, even though much effort has been invested, the problem of the optimal state assignment is still far from being solved. The main reason is the complex dependency between the choice of the state codes and the resulting Boolean functions. Moreover, even after the encoding is chosen and the combinational component is determined, the main attributes of its circuit realization, such as area, speed, or power consumption cannot be readily obtained. As illustrated by Fig. 1.5, the encoded machine is further subjected to the combinational logic synthesis, technology mapping and placement and routing, each of these steps applying complex heuristics to find optimum implementation. The state assignment method has no direct or easy way of predicting the result of

these operation and therefore the task of determining codes resulting in effective and efficient implementation is extremely difficult.

The problem is further complicated by the feedback loop introduced by the state register. The fact that state-code-bits are both inputs and outputs of the combinational component implies that the consequences of choosing a particular state assignment need to be considered both from the input and output point of view. The requirements resulting from these two types of analysis may turn out to be conflicting and require intelligent resolving.

Finally, the desired form of the combinational component strongly depends on the target technology, as it is the implementation technology that ultimately determines which functions can be efficiently realized. This fact limits applicability of the state assignment methods developed for previous technologies to the new ones.

Related research

The first attempts at algorithms of state assignment for computer program implementation date back to 1960s, when the implementation technology were diodes, transistors and simple gates. Following the methods of human designers, formulated by Humphrey [25] as the "rules for *state code adjacency*", the algorithms proposed in [3][2][15] are based on grouping together 1's in the Karnaugh tables of the resulting binary output and next-state functions, and in result on maximizing the size and minimizing the number of product terms in the sum-of-product expressions describing the combinational part of the FSM realization.

Even though the technology changed, the observations formulated by Humphrey became foundation for the next generations of algorithms.

This was also the case with the family of algorithms based on *symbolic minimization*. This novel idea implemented by de Micheli in KISS [13] in 1985 consists in performing the logic minimization phase *before* the state encoding. The minimized functions could be realized in PLA technology at the cost (in terms of circuit area) determined prior to encoding, assuming that the constructed code will satisfy all encoding constraints resulting from the symbolic minimization phase. The state encoding problem is in this way reduced to satisfaction of the so called *face* or *input* constraints expressing certain relations between the codes of some groups of states.

The method implemented in KISS has however several shortcomings. First of all, in the symbolic minimization phase the binary next-state functions are assumed to have disjoint on-sets. Thus, the next-state function minimization is not taken into account. This aspect is especially apparent for counter FSMs, where minimized symbolic cover for $2p$-state counter has 2^p product terms, while the optimum is $O(p^2)$. Moreover, the FSM's feedback is not properly taken into account. Also the constraint satisfaction algorithm is just a simple-minded, greedy search.

The idea corresponding to symbolic minimization (minimization of generalized implicants) was even earlier pursued in [6]. However, the computational complexity of the presented (admittedly, much more sophisticated) method made it impractical for FSMs larger that 8 states.

In 1988, Jóźwiak published the method of maximal adjacencies, MAXAD, targeting two-level logic implementations [30][31][33]. Although the method uses the "state code adjacency" concept, it differs considerably from the previous methods based on this idea.

Jóźwiak considers many sorts of adjacencies, performs a sophisticated adjacency analysis and uses the results of the analysis in a sophisticated code construction performed in the framework of an effective and efficient double-beam search algorithm. This resulted in a state assignment tool that efficiently produced much better results than any other method published at that time. Compared to KISS, the machines encoded by MAXAD have realizations with on average 13% smaller PLA area and 28% smaller feedback area.

Some of the shortcomings of KISS are addressed in its successor — NOVA [74]. NOVA takes more efficient and flexible approach to constraints satisfaction, representing it as a graph embedding problem and solving in several, heuristic strategies producing superior results and offering quality/runtime trade-offs. Also the output encoding problem is considered here, however in a marginal manner, subordinate to the KISS-like input encoding. The best strategy of NOVA – *iohybrid* – produced results of comparable quality to the results of the maximal adjacency method.

The output encoding problem is addressed in two *dichotomy-based* methods, DUET [11] and DISA [61]. They represent constraints as dichotomies, thereby introducing uniform representation of encoding and constraints. The problem of the encoding constraint satisfaction is transformed in this way to the problem of the compatible dichotomies merging. The resulting merged dichotomies define unambiguously certain encoding, which satisfies all the constraints represented by the component dichotomies.

All of the above symbolic minimization-based methods assume two-level PLA implementation of the functions resulting from the state encoding. This assumption drives the symbolic minimization, which aims at generation of minimum-cardinality symbolic cover – a reasonably good approximation of the optimal FSM realization, especially for input-dominated FSMs. They differ from each other in the level of detail in constraint consideration and in constraint satisfaction method.

Unfortunately, no approximation as good as term number for two-level implementation has been discovered for any sort of multi-level logic (except for multiplexor networks). The possibility of realizing multiple trade-offs and, in particular, the characteristics of the new multi-level implementation platforms greatly complicate construction of good cost functions, which hinders quality estimation of the constructed code. For these reasons, symbolic minimization is difficult to realize in the multi-level domain (one of the few attempts was taken in MIS-MV [55]). Instead, most encoding methods for the multi-level implementations settle for crude predictions of the operations of subsequent logic synthesis steps, and attempt to create such an encoding, which produces output functions "easy" for a certain multi-level combinational logic minimizer targeting a particular implementation technology.

One of the earliest multi-level state assignment methods, MUSTANG [14], falls into this category. Designed to work with MIS logic synthesis system, it attempts to maximize the number and size of the common (sub-)cubes in expressions describing the output functions. These common (sub-)cubes will make it easier for MIS to realize its objective of minimizing the number of literals. The common cube maximization is realized in the process of adjacent code assignment to some selected pairs of states. The selection of the pairs, interestingly, is guided by rules similar to those of Humphrey. Two other well know methods, JEDI [57] and MUSE [16], closely follow the concepts implemented in MUSTANG, with additional improvements introduced by elements of simulated annealing and consideration of Boolean (as opposed to algebraic) operations in common-cube extraction. Since codes produced by MAXAD also result from cumulation of adjacencies,

it belongs to the same class of methods and produces comparable results for multi-level circuits.

A different approach is taken in MIS-MV [55]. It follows in the footsteps of KISS and applies symbolic minimization to the multi-level realizations. MIS-MV is actually not a state encoding method, but a multi-level logic minimizer able to handle multivalued variables. It is therefore able to minimize the combinational component of FSM before the actual state encoding, when the state variable is still in its symbolic, multivalued form. Symbolically minimized function is *then* encoded with a simple algorithm based on simulated annealing, guided by the number of literals as its cost function.

Some of the interesting alternative approaches to state encoding include genetic algorithms [1][10]. These methods, however, suffer from the fact that basic genetic algorithms are known to poorly handle the problems with complicated and time consuming quality function [46]. It is apparent in the run-times of the tools, which exceed in some, even small, cases those of the classical algorithms by a factor of 100. An interesting direction relevant to some implementation technologies pointed in [10] is, however, the simultaneous encoding and selection of the types of flip-flops used to store state variables (D or J/K type). In some cases, the choice of J/K flip-flops (which are known to require less complicated excitation functions) reduced the combinational component of the FSM realization by as much as 80%.

The challenges introduced by the multi-level implementations were further deepened with the advent of FPGA devices. The characteristics of FPGAs invalidate estimations of the implementation cost of the Boolean functions by the number of terms or literals. Till now, little has been done in the field of sequential synthesis targeting FPGA implementations. Notable exceptions are the programs LAX [69] – a BDD-based tool for multiplexor-based FPGA architectures and MINISUP [56] – a tool for reducing input support of the binary functions implemented in LUT-FPGAs.

A totally different view at the state encoding problem is presented in the algebraic structure theory of sequential machines due to Hartmanis and Stearns [22]. The theory utilizes the concepts of partitions and set systems to model the FSM's information structure, and trace dependencies between the information about the FSM's states, inputs and outputs. Observation of these dependencies gives some hints related to the direction, in which the encoding of the states should follow to reduce the dependecies of the next-state and output functions from the state and input variables. This approach seems to be especially interesting in the case of LUT-FPGAs, where reducing dependencies satisfies two important goals of reducing the functions' input supports and interconnections.

However, the hints of Hartmanis and Stearns were limited to only some special implementation structures of sequential machines (parallel and serial decomposition), required extensive computations (computation of SP-partitions and partition pairs) and were not formulated into any method or algorithm. Moreover, although Hartmanis and Stearns understood that partitions and set systems model information, they did not discover any method of expressing what particular information is modeled, and therefore they were unable to characterize and measure the modeled information and the relationships between the modeled information stream.

Therefore, Jóźwiak formulated the theory and methodology of general decomposition [35] that give the most general known generator of correct sequential and combinational circuit structures, and explained how to use the generator for the (near-)optimal circuit construction. To enable effective and efficient circuit construction, Jóźwiak pro-

posed the notion of elementary information [36] that enables precise characterization of information as modeled, for instance, by set systems and partitions, and using this notion formulated the theory of information relationships and relationship measures [36]. The apparatus of information relationships and relationship measures enables analysis and measurement of the modeled information and relationships between information streams in a given discrete function, relation or sequential machine, as well as in the circuit under construction. The information delivered by the apparatus of information relationships and measures can be used to control the circuit generator of general decomposition in order to construct only the most promising circuit structures and arrive at some (near-)optimal circuits. In this way, information relationships and measures make operational the theory of partitions and set systems of Hartmanis and Stearns and its extensions. In particular, it enables construction of effective and efficient decomposition and encoding algorithms based on this theory and its extensions.

Some applications of the algebraic structure theory (without its extensions) to FSM decomposition or encoding were also considered by some other authors (see a.o. [4, 63, 72, 76]).

1.4 General decomposition

Our approach to state assignment is based on a special case of general decomposition of finite state machines. Decomposition is a natural method of handling complex problems. It consists in splitting a complex problem into smaller, interrelated subproblems and solving each of the subproblems separately. General decomposition of finite state machines is a theory developed by Jóźwiak that deals with realization of the behavior of large FSMs by networks of interconnected, smaller FSMs operating in parallel (so called partial machines). In [35] Jóźwiak published theorem fundamental to this theory that established conditions for valid decomposition of completely specified, deterministic finite state machines. The theorem describes decomposition by partitions – mathematical constructs with well-defined operations. This enables efficient manipulation of the decomposition structures in the domain of partitions, while preserving the behavior of the decomposed finite state machine.

The theorem considers general decomposition structure presented in Fig. 1.7, where the partial machines are fed input information processed by some primary input decoder, their interconnections are realized by means of connection modules and their output is combined by the output encoder block to produce the output of the network. The entire network, when considered from the point of view of primary inputs and outputs behaves identically as the FSM being realized.

The general decomposition theory of completely specified FSMs was later extended by Jóźwiak and the author of this dissertation to the case of incompletely specified FSMs and their multi-state realizations [48]. This extension accounts for incompletely specified output function (output don't-cares) as well as for the incompletely specified state transition function (non-determinism) for both original machine and the partial machines. Multi-state realization (see also Definition 3.15) allows the modeling of decomposition networks having multiple states corresponding to a single state of the original machine. This way, the extended theory covers *all* finite state machines and a much wider range of their decompositions that include not only identical, but also *compatible* behavior realiza-

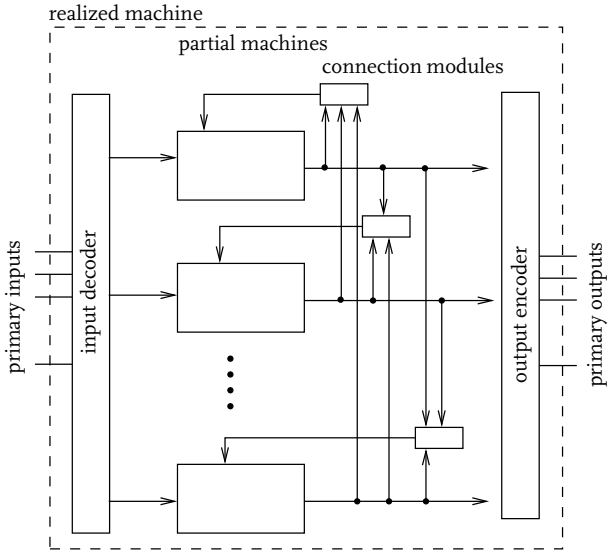


Figure 1.7. General decomposition

tions. The extended theory is presented and used in this thesis.

In a separate work [36], Jóźwiak showed correspondence between the partitions used to describe the decomposition structure and information flowing within the structure. That work introduces the apparatus of Information Relationships and Measures (IRM) that enables analysis of information in a system and in particular:

- analysis of the information flows (where and how particular information is produced, and where and how it is consumed)
- analysis of the relationships (similarity, difference) between various information channels
- quantitative analysis of the information flows and their relationships (quantity, importance of information)

General decomposition theorem and IRM together form the basis for construction of effective and efficient decomposition methods and tools, with decomposition theory providing engine for creating correct-by-construction realization structures and IRM providing measures to guide the construction process towards the most efficient realizations.

One particular special case of general decomposition is functional decomposition. It deals with state machines having a single state and trivial state behavior, i.e. with combinational functions. In this approach, a large combinational function is realized by a network of smaller subfunctions. Functional decomposition has been successfully used among others for logic synthesis for FPGA platforms [41].

However, the applications of general decomposition are not limited to circuit synthesis. It can be used in analysis and decomposition of any discrete, binary, multi-value or symbolic system in many fields of modern engineering and science. Known applications

include pattern analysis, knowledge discovery, machine learning, neural network training, decision systems, databases, encryption, compression, encoding etc. The particular suitability of decomposition-based methods to all these fields stems from the fact that, as the name suggests, general decomposition does not make any prior assumptions about the form of the network or the partial machines (or functions). Therefore, it allows the discovery of any "natural" structure that best realizes the required behavior. This is in contrast to many current methods popular in particular in logic synthesis. Most of these methods are devoted to some very special cases of possible implementation structures involving some minimal functionally complete systems of logic gates (e.g. AND+OR+NOT, NAND, AND+EXOR, etc.). This is particularly unsuitable for implementation technologies such as FPGAs, where the constraints on the network do not involve the kind of the logic functions (LUT in FPGA is capable of realizing any function of a limited number of inputs), but rather their structural characteristics, such as number of inputs and outputs (LUT is a 4-input, 1-output universal gate), or number and length of interconnections.

In this thesis, we present extension of the General Decomposition Theorem to the case of incompletely specified, non-deterministic finite state machines with multi-state behavior realization. This extends the field of realization structures covered by GDT to include *all* valid implementations, not only strictly equivalent to the original specification, but all *compatible* with the original description. The extension allows to handle not only simple machines and functions, but also incompletely specified, multi-valued machines and relations, widening even further applicability of the theory to other fields of science and engineering.

In particular, we formulate the state assignment problem as a special case of general decomposition. Based on this formulation, we use set systems to describe state assignment of an FSM and, based on General Decomposition Theorem, we derive conditions for the encoding to be valid. These conditions give rise to a general state assignment method that is capable of not only identifying valid encodings, but also performing implicit machine simplification (e.g. implicit minimization of the number of states) in the encoding process, using freedom and redundancy present in specification. However, while the general method describes mechanics of the encoding construction process, it does not prescribe the decisions necessary to steer the construction process towards efficient implementation. These decisions require heuristics reflecting the particular encoding objectives. We implemented such heuristics for the case of FSM targeting implementation in FPGA. These heuristics basically consist in confronting the information streams in a particular FSM with the constraints imposed by the FPGA technology and the optimization objectives. The analysis of the information streams and their relationships is facilitated by the apparatus of information relationships and measures. The software tool implementing these heuristics and experimental results obtained with this tool are also discussed in this thesis.

1.5 Research aims and thesis overview

1.5.1 Motivation

Programmable logic fills the flexibility, performance, power dissipation, and development and fabrication cost gap between the application specific integrated circuits and standard (general purpose) programmable micro-processors. During the last decade it

became the mainstream implementation technology for custom computation and embedded system products in such fields as telecommunication, networking, image processing, video processing, multimedia, DSP, cryptography, embedded control etc. To efficiently develop, implement and use the systems based on programmable logic, adequate computer-aided support tools are necessary. Since most such systems are implemented using the look-up table (LUT) FPGAs, the circuit synthesis tools targeting this technology are of primary importance for their effective and efficient implementation.

However, the new set of characteristics and constraints imposed by the modern FPGA technology is not properly addressed by the available synthesis tools. These characteristics include the availability of multi-level networks of arbitrary functions limited by their dimensions (maximum number of inputs and outputs) rather than specific functions limited by their form (number of terms or literals), and the growing influence of interconnections on all important circuit characteristics (speed, area, power dissipation, etc.). To address these issues, a new family of synthesis methods based on the general or functional decomposition has been recently proposed. The decompositional approach, and in particular the general decomposition approach proposed by Jóźwiak [35], has the potential of directly building such complex multi-level networks of functions with constrained number of inputs and/or outputs, and directly controlling the number and length of interconnections. The combinational synthesis methods based on the general decomposition significantly outperform other methods used in academic and commercial tools, demonstrating the effectiveness of this approach [41]. While the problems of combinational synthesis for reconfigurable platforms have received significant attention of researchers, there is very little done in the field of sequential synthesis targeting FPGA implementation.

Meanwhile, the sequential synthesis has a profound impact on the final implementation cost. In sequential synthesis, the symbolic description of the FSM functionality is translated to a set of Boolean functions implementing this functionality. The form of these functions determines all crucial characteristics of the system's implementation, such as area, delay and power dissipation. The importance of sequential synthesis is paired with great difficulty of the problems involved. Decomposition and encoding of FSMs are both well-known to be NP-hard problems, which makes the search for the high-quality solutions only feasible through heuristic methods. These heuristic methods are hindered by the complex evaluation criteria for particular solutions. The complexity stems from the fact that the actual implementation cost of a given solution is not known until the combinational synthesis and technology mapping were performed on the particular Boolean network produced for the given solution. Significance of the FSM decomposition and assignment problems and the lack of answers to them, especially in relation to the FPGA technology, motivated us to perform the research in the field of sequential synthesis of FSMs targeting FPGA implementation.

Recently, Jóźwiak proposed a new information-driven approach to circuit synthesis and formulated two theories that support this approach:

- theory of general decomposition of discrete functions and sequential machines [35], and
- theory of information relationships and information relationship measures [36, 37].

This way, a new theoretical and methodological framework has been established for analysis and synthesis of sequential and combinational logic networks. The framework con-

sists of the information-driven approach to circuit synthesis, theory of general decomposition, and the information modeling and analysis apparatus based on information relationships and measures. The framework was successfully applied by Jóźwiak and his collaborators to a variety of problems, including combinational synthesis [38, 41, 43, 44, 51, 52, 64, 65].

Encouraged by the success of the decompositional combinational synthesis methods and the level of control provided by general decomposition over the vital characteristics of the synthesized system, we decided to pursue the information-driven decompositional approach to sequential synthesis.

1.5.2 Research overview

The main **aims** of this research can be sub-divided into:

1. analysis aims, including the aims related to:
   - research field and problem analysis, and
   - research result analysis, including experimental research
2. synthesis aims, including:
   - theoretical
   - methodological, and
   - application-oriented

The **main theoretical aim** was to further analyze and adequately supplement the theory related to the subject field, to obtain a more complete theoretical base for this and further research in FSM decomposition and encoding.

This aim has been successfully realized. The **main result** is related to the theory of general decomposition of sequential machines. The central point of this theory is a constructive theorem on the existence of a general decomposition. This theorem provides a generator of correct by construction circuit structures. Originally it was formulated by Jóźwiak for the case of deterministic (completely specified) sequential machines with single-state behavior realization [35]. At the start of this research, Jóźwiak proposed the outline of the extended version of the theorem to account for the sequential machines with incompletely specified output function and transition function (nondeterministic) and their multi-state realizations and outlined the proof of the extended theorem. The final version of the extended general decomposition theorem and its complete proof have been constructed together by the author and Jóźwiak as a part of the research reported in this thesis.

The **main methodological aim**, being the **central aim of this research**, was to develop an effective and efficient FSM state assignment method for the decompositional state assignment, based on the information-driven approach to circuit synthesis, theory of general decomposition, and theory of information relationships and measures. The associated **primary application-oriented aim** was the implementation of the method in the form of an EDA software tool and their testing.

These aims have been fully realized. Based on the outline of the FSM state assignment method proposed by Jóźwiak, I developed and implemented in a prototype software

tool a complete effective and efficient FSM state assignment method, including its particular data structures and algorithms.

The **primary analytical aim** of the research was to analyze the characteristics of the modern FPGA devices and how these characteristics are addressed by the available state assignment methods. The methods used in commercial FPGA synthesis tools were of particular interest.

To this end, I reviewed the available FPGA product families and their features. I also analyzed the results of state assignment with various academic tools and the methods used in commercial synthesis software. In particular, the applicability of one-hot encoding prevalent in commercial tools was a subject of investigation. While I found one-hot encoding in most cases inadequate for FPGA implementation, some of the cases where it does produce efficient realizations led to the development of a method reported in this thesis that automatically identifies good candidate FSMs for one-hot encoding.

Another **analytical aim** was to support the development of the information-driven decompositional state assignment method with experimental research and analysis of its results. Within this scope, we were particularly interested in interactions between our state assignment method and the combinational synthesis methods that were used to synthesize the encoded FSMs.

This aim was realized by evaluating the synthesis results of a large number of FSMs encoded with the proposed state assignment method. I analyzed the synthesis results both after combinational synthesis of the encoded machines and after placement and routing of the synthesized circuit in the actual FPGA device. To further support this aim, we developed together with L. Jóźwiak and D. Gawlowski sequential benchmark generator BENGEN [42] that enabled efficient generation of various sorts of well-characterized FSM benchmarks and this way facilitated the analysis of effectiveness and robustness of the proposed method on a wide spectrum of FSMs. The results of this analysis were used to improve the effectiveness of the state assignment method, and to finally characterize and evaluate the method.

1.5.3 Thesis overview

In the remainder of this thesis we will discuss the results of the research. We will present the extensions we introduced to the existing general decomposition theory of completely specified FSMs to account for non-determinism, incompletely specified output functions and multi-state behavior realization. The extensions are introduced in the new formulation of General Decomposition Theorem, which we present along with the proof. The extended General Decomposition Theorem constitutes the most general known generator of correct decompositional circuit structures. It covers all other specific decomposition structures, such as parallel or serial decomposition, as well as the decomposition of discrete functions and relations, as special cases [35]. The theorem explicitly treats information flows within the decomposed machine or function and thus supports the novel, information-driven approach to decomposition.

We will then show the correspondence between the state assignment and decomposition, and how we used this correspondence to derive the conditions for valid state assignment from the General Decomposition Theorem. These conditions are more general and flexible than conditions used in other state assignment methods, and thus allow the encoding method to search in a larger solution space. They form the basis of a gen-

eral, implementation-platform-independent encoding method that constructs encoding by merging the variables of the specific initial encoding. Thanks to the flexible formulation of the validity conditions, the method enables implicit optimizations of the encoded machine during the encoding process in a manner unexplored by other methods.

Further, we will analyze the FPGA characteristics that influence the implementation efficiency of sequential circuits and, based on this analysis, we will propose novel heuristics for state assignment. These heuristics exploit the information modeling apparatus used in formulation of the General Decomposition Theorem and analysis apparatus of information relationships and measures to perform analysis and optimization of the information flows within the encoded machine during the encoding. We implemented these heuristics in the software tool SECODE, which we will discuss in some detail.

Finally, we will present and discuss the results of the experimental research we performed using the newly developed state assignment tool and some of the other popular state assignment methods. We will show that the FSM implementations resulting from the encoding process performed with SECODE are smaller and faster than those resulting from other encodings.

In the presented research, we considered the important and difficult problems of decomposition and state assignment of FSMs for the emerging, FPGA-based reconfigurable system implementation platforms. We extended the underlying theory and applied it to develop the novel state assignment method, which outperforms the currently used methods. The experimental research of the developed theory, method and prototype software tool demonstrates the effectiveness of the information-driven approach to circuit synthesis, and shows that they form a solid base for further theoretical and experimental research in this field.

1.5.4 Thesis outline

The presentation of the research is organized in the following manner.

In Chapter 2, we introduce the necessary theoretical background related to Boolean functions, finite state machines and information modeling and analysis. Based on this background, we will present in Chapter 3 the theory of general decomposition. To better introduce the complex notions of decomposition, we will first present and illustrate with an example the General Decomposition Theorem proven in [35] for the case of completely specified FSMs. Then, we will discuss modifications to the theorem necessary to account for incompletely specified output and next-state functions and multi-state behavior realization. Finally, we will prove and illustrate with another example the extended General Decomposition Theorem for incompletely specified FSMs with multi-state behavior realization.

In Chapter 4, we will show the correspondence between state assignment and decomposition of an FSM. This correspondence will allow us to derive from the General Decomposition Theorem the conditions for a valid state assignment that form the basis of a general, implementation-platform-independent encoding method. While the general encoding method presented in Chapter 4 provides mechanics of constructing a valid state assignment, it does not introduce any implementation-platform-dependent knowledge necessary to construct an encoding that will result in efficient implementation of the FSM on the target platform. In Chapter 5, we will present heuristics guiding the general state assignment method towards encodings that result in efficient FPGA imple-

mentation of the FSM, and the software tool SECODE implementing these heuristics.

The results of the experiments with SECODE will be discussed in Chapter 6. We will conclude with the summary of the research results and recommendations for future work in Chapter 7.

Chapter 2

Preliminaries

Digital circuits can be classified, with respect to their behavior, into two main classes: combinational and sequential. In combinational circuits, the output of the circuit depends only on the current value of the inputs. The functionality of a combinational circuit can be described by Boolean functions of its outputs. In the sequential circuits, the output depends not only on the current inputs but also on the history of execution, represented by the current *state* of the circuit. The current state is stored in some memory elements and influences the output of the circuit as well as the next state. The functionality of a sequential circuit can be described by a Finite State Machine (FSM).

Both Boolean functions and Finite State Machines can be modeled as information processing systems that require some input information, possibly use some internal state information (FSMs) and from combination of these two sources produce the output information. The analysis of the information and information flows in FSMs is the basis of the state assignment method presented in this thesis. To model the information, we use constructs called covers and to analyze the relationships and characteristics of the information modeled by covers, we use information relationships and measures.

In this chapter we will introduce basic notions related to Boolean functions, Finite State Machines and information modeling and analysis with covers and information relationships and measures.

2.1 Boolean functions

Definition 2.1 (Completely specified Boolean function). *Completely specified Boolean function of n variables is a mapping $B^n \rightarrow B$, where $B = \{0, 1\}$.*

Definition 2.2 (Incompletely specified Boolean function). *Incompletely specified Boolean function of n variables is a mapping $B^n \rightarrow B^*$, where $B^* = \{0, 1, -\}$ and "$-$" stands for "don't care" and indicates any of the values 0 or 1.*

Boolean variables will be denoted in the following by lowercase letters (e.g. a). A negated Boolean variable is denoted by $\overline{a}$ and represents completely specified (c.s.) Boolean function $\neg a$. A *literal* is a variable or its negation (a or $\overline{a}$). A *cube* is a set of literals that represents c.s. Boolean function being the product of the literals (e.g. cube $\{a, b, \overline{c}\}$ represents function $a \wedge b \wedge \neg c$, also denoted $ab\overline{c}$). A *sum-of-products (SOP)* expression is

a set of cubes representing c.s. function corresponding to the disjunction of the cubes' functions (e.g. $\{\{a, b, \overline{c}\}, \{\overline{d}\}\}$ is a SOP corresponding to function $a \wedge b \wedge \neg c \vee \neg d$, or $ab\overline{c}+\overline{d}$). A *truth table* is a table listing values of a Boolean function for some input combinations. For the input combinations not explicitly listed in the truth table, the don't-care ("$-$") value is assumed.

The above notions can be extended to discrete multiple-valued (symbolic) functions and relations. This is partially done in Chapter 3, when considering the general decomposition of FSMs, as discrete functions are a special case of FSMs.

2.2 Finite State Machines

Definition 2.3 (Completely specified finite state machine (FSM) – Mealy type).
Completely specified Mealy-type finite state machine $\mathbf{M}$ *is an algebraic system defined by:*
$\mathbf{M} = \{I, S, O, \delta, \lambda\}$

Where:
I *– a finite set of inputs*
S *– a finite non-empty set of internal states*
O *– a finite set of outputs*
δ *– the next-state function* $\delta : S \times I \rightarrow S$
λ *– the output function* $\lambda : S \times I \rightarrow O$

Definition 2.4 (Completely specified finite state machine (FSM) – Moore type).
Completely specified Moore-type finite state machine $\mathbf{M}$ *is an algebraic system defined by:*
$\mathbf{M} = \{I, S, O, \delta, \lambda\}$

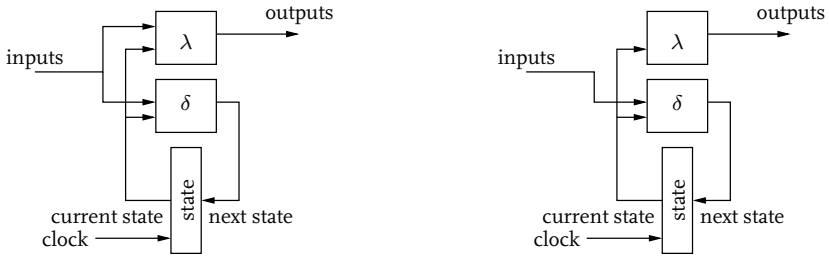
Where:
I *– a finite set of inputs*
S *– a finite non-empty set of internal states*
O *– a finite set of outputs*
δ *– the next-state function* $\delta : S \times I \rightarrow S$
λ *– the output function* $\lambda : S \rightarrow O$

The two types of Finite State Machines differ in the method of producing output. In the Moore FSM, output is produced "in the state", i.e. the output depends exclusively on the current state of the machine. A Mealy FSM, on the other hand, produces outputs "on the transition", i.e. the output depends not only on the current state, but also on the current input and is produced concurrently with computation of the next state. The schematic structures of both types of FSMs are presented in Fig.2.1.

For any Moore machine there exists a Mealy machine with equivalent output behavior and vice versa, which makes both types equally expressive in terms of functionality. Also, when abstracting from the above described difference in the output timing and the related physical implementation consequences, Moore machine can be considered as a special case of Mealy machine. Therefore, theory and methods developed for Mealy machines will be also applicable to Moore machines (in some cases after small modifications related to the difference in output computation timing). For this reason in the following we will focus on the Mealy-type machines, and by FSM we will mean Mealy-type FSM, unless explicitly stated otherwise.

One of the forms of the specification of a finite state machine is a State Transition Table (STT). The table consists of rows (transitions), each transition in the form $(x, s, \delta(s, x), \lambda(s, x))$, i.e. it consists of the input value, current state of the machine, the next state of the machine for this input/current state combination and the output of the machine for this input/current state combination. This specification can be easily transformed into corresponding State Transition Graph (STG) representation. In STG, nodes represent states of the machine and directed edges represent transitions. A transition edge connects a state with its next state and is labeled with the input combination that triggers this transition and the output value produced by the FSM in these input/current state conditions.

Ex. 2.2.1. *Consider for example the State Transition Table in Fig. 2.2(a). It describes a (Mealy) Finite State Machine with one binary input, three states a, b and c, and two binary outputs. The first row of the STT specifies that when the FSM is in state a and the input is 0, the machine moves to state c and produces output 00. The corresponding STG is shown in Fig. 2.2(b). The first transition is reflected by the edge from state a to c labeled with input combination 0 and the produced output value 00.* ■

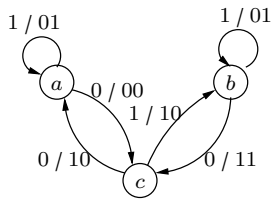


(a) Mealy (b) Moore

Figure 2.1. Finite State Machine types

in	ps	ns	out
0	a	c	00
1	a	a	01
0	b	c	11
1	b	b	01
0	c	a	10
1	c	b	10

(a) State Transition Table



(b) State Transition Graph

Figure 2.2. Completely specified sequential machine

Definition 2.5 (Incompletely specified sequential machine). *Incompletely specified sequential machine* $\mathbf{M}$ *is an algebraic system defined by:* $\mathbf{M} = \{I, S, O, \delta, \lambda\}$

Where:
I *– a finite set of inputs*
S *– a finite non-empty set of internal states*
O *– a finite set of outputs*
δ *– the next-state function* $\delta : S \times I \rightarrow 2^S$
λ *– the output function* $\lambda : S \times I \rightarrow 2^O$

In incompletely specified sequential machines next-state and output functions incorporate *generalized don't-cares* – for each input/state combination they return a subset (a choice) of values from state (output) alphabet. "Traditional" don't-cares ("–") are a special case of the generalized don't-cares, because "–" represents a whole set of states (for next-state don't-care) or output values (for output don't-care). Thus, Def. 2.5 covers the traditional definition of an incompletely specified sequential machine as its special case. This definition covers also nondeterministic FSMs, which are defined as FSMs having transition function that for a given input and current state combination may return multiple next states.

in	ps	ns	out
0	a	b, c	$00, 11$
1	a	a	01
0	b	a, c	11
1	b	b	01
0	c	a	$10, 01$
1	c	b	$10, 01, 11$

Figure 2.3. Incompletely specified sequential machine

Ex. 2.2.2. *Consider the STT in Fig 2.3. Transition 1 specifies a move from state* a *under input of* 0 *to* either b or c*, producing output* 00 *or* 11. ∎

When performing any operations or modifications of an FSM, it is usually necessary to ascertain that the behavior of the modified FSM does not change. For this purpose a concept of *behavior realization* is introduced that indicates that the modified FSM still has the same behavior as the original machine. There exist two notions of behavior realization for FSMs: output behavior realization and state behavior realization.

Output behavior realization means that for any sequence of inputs values, the sequences of output values are identical. The following definition states that property formally and relaxes the requirement for identical or isomorphic sequences to allow for *equivalent* sequences, which introduces mapping functions between the input/output alphabets of the original machine and its equivalent realization.

Since a single state of the realization machine $\mathbf{M}'$ is used here to represent a certain state of the specification machine $\mathbf{M}$, such realizations are called *single-state realizations.*

Definition 2.6 (Single-state output behavior realization of completely specified FSM). *Completely specified machine* $\mathbf{M}'(I', S', O', \delta', \lambda')$ *is an output behavior realization of c.s.*

machine $\mathbf{M}(I, S, O, \delta, \lambda)$ *if and only if functions:* $\Psi : I \to I', \Phi : S \to S', \Theta : O' \to O$ *exist, such that*

$$\forall s \in S \, \forall x \in I \quad \delta'\big(\Phi(s), \Psi(x)\big) = \Phi\big(\delta(s, x)\big) \wedge \Theta\Big(\lambda'\big(\Phi(s), \Psi(x)\big)\Big) = \lambda(s, x)$$

For many applications, only output behavior of the FSM is relevant, and the internal state behavior is never apparent to, or used by, the environment. However, in some cases the current state of the machine may be used outside it. Then, the concept of the state behavior realization is necessary that requires that the current state of the modified machine unambiguously identifies the state of the original machine (possibly, after renaming).

Definition 2.7 (Single-state output and state realization of completely specified FSM). *Completely specified machine* $\mathbf{M}'(I', S', O', \delta', \lambda')$ *is an output and state behavior realization of c.s. machine* $\mathbf{M}(I, S, O, \delta, \lambda)$ *if and only if functions:* $\Psi : I \to I', \Phi' : S' \to S, \Theta : O' \to O$ *exist, such that*

$$\forall s' \in S' \, \forall x \in I \quad \Phi'\Big(\delta'\big(s', \Psi(x)\big)\Big) = \delta\big(\Phi'(s'), x\big) \wedge \Theta\Big(\lambda'\big(s', \Psi(x)\big)\Big) = \lambda(\Phi'(s'), x)$$

For the case of incompletely specified FSMs, the above requirements for behavior realization can be further relaxed to allow for the realization machine to use some of the implementation freedom allowed for by the original FSM's don't-cares. In particular, in given state/input conditions the next state and output of the realization machine does not have to be exactly equivalent to the next state and output of the specification machine. It is sufficient if they are *compatible,* i.e. belong to a subset of the next-states and output of the specification machine. Also, a single state of a specification machine $\mathbf{M}$ can be in general represented by multiple states of a corresponding implementation machine. As a result of the above considerations, the conditions of the behavior realization can be relaxed to check for compatibility (i.e. inclusion) of next-states and outputs instead of equivalence (i.e. equality).

Definition 2.8 (Output behavior realization of incompletely specified FSM). *Incompletely specified machine* $\mathbf{M}_r(I_r, S_r, O_r, \delta_r, \lambda_r)$ *is an output behavior realization of i.s. machine* $\mathbf{M}(I, S, O, \delta, \lambda)$ *if and only if functions:* $\Psi : I \to I_r, \Phi : 2^S \to 2^{S_r}, \Theta : 2^{O_r} \to 2^O$ *exist, such that*

$$\forall s \in S \, \forall x \in I \quad \delta_r\big(\Phi(s), \Psi(x)\big) \subseteq \Phi\big(\delta(s, x)\big) \wedge \Theta\Big(\lambda_r\big(\Phi(s), \Psi(x)\big)\Big) \subseteq \lambda(s, x)$$

If $\mathbf{M}_r$ is an output behavior realization of $\mathbf{M}$ then, for all possible input sequences, the output sequences produced by imitation $\mathbf{M}_r$ after renaming them are subsets of the output sequences of $\mathbf{M}$.

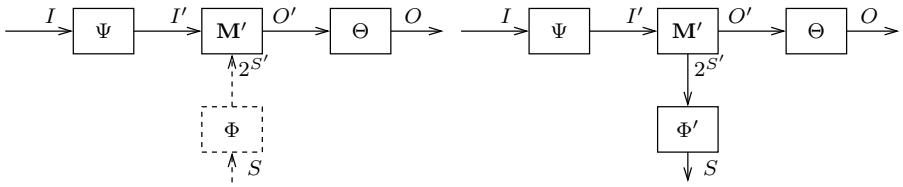
Definition 2.9 (Output and state realization of incompletely specified FSM). *Incompletely specified machine* $\mathbf{M}_r(I_r, S_r, O_r, \delta_r, \lambda_r)$ *is an output and state behavior realization of i.s. machine* $\mathbf{M}(I, S, O, \delta, \lambda)$ *if and only if functions:* $\Psi : I \to I_r, \Phi_r : 2^{S_r} \to 2^S, \Theta : 2^{O_r} \to 2^O$ *exist, such that*

$$\forall s_r \in S_r \, \forall x \in I \quad \Phi_r\Big(\delta_r\big(s_r, \Psi(x)\big)\Big) \subseteq \delta\big(\Phi_r(s_r), x\big) \wedge \Theta\Big(\lambda_r\big(s_r, \Psi(x)\big)\Big) \subseteq \lambda(\Phi_r(s_r), x)$$

Given a specification machine $\mathbf{M}$ and a machine $\mathbf{M}'$ realizing its output and/or state behavior, the machine compatible with the original can be built by adding input / output / state mapping functions to the realization machine. Such structure is referred to as *realization structure of* $\mathbf{M}$.

Definition 2.10 (Realization structure). *The sequential machine composed as a structure consisting of* Ψ, $\mathbf{M}'$ *and* Θ *(and* Φ' *for state behavior realization) is referred to as the* realization structure *for* $\mathbf{M}$ *defined by* $\mathbf{M}'$ *and denoted as* $\mathbf{str}(\mathbf{M}')$

Schematic representation of the realization structure for output and state and output behavior realization is presented in Fig. 2.4.



(a) Output realization

(b) Output and state realization

Figure 2.4. FSM behavior realization structures

2.3 Covers

The concepts of covers, set systems and partitions are central to the information analysis apparatus used extensively in this thesis. They are used as representation of information in the modeled circuits. These concepts were used under various names by various researchers, e.g. in [7, 8, 23]. Below, we present the nomenclature introduced in [48].

Definition 2.11 (Cover). *Cover* ϕ_S *on* S *is defined as*

$$\phi_S = \left\{ B_i | B_i \subseteq S \ \wedge \bigcup_i B_i = S \right\}$$

Cover is any set of subsets of S covering S.

Ex. 2.3.1. *For set* $S = \{1 \ldots 5\}$ $\phi_S^1 = \{\overline{1}; \overline{12}; \overline{23}; \overline{45}; \overline{45}\}$ *is a cover, but* $\phi_S^2 = \{\overline{1}; \overline{12}; \overline{45}; \overline{45}\}$ *is not.* ∎

Definition 2.12 (Unique-block cover). *Unique-block cover* ϕ_S *on* S *is defined as*

$$\phi_S = \left\{ B_i | B_i \subseteq S \ \wedge \bigcup_i B_i = S \ \wedge \ i \neq j \Rightarrow B_i \neq B_j \right\}$$

Unique-block cover is a cover with no identical blocks.

Ex. 2.3.2. ϕ_S^1 *from Example 2.3.1 is not a unique-block cover, but* $\phi_S^3 = \{\overline{1}; \overline{12}; \overline{23}; \overline{45}\}$ *is.* ∎

Definition 2.13 (Set system). *Set system ϕ_S on S is defined as*

$$\phi_S = \left\{ B_i | B_i \subseteq S \ \wedge \ \bigcup_i B_i = S \ \wedge \ i \neq j \Rightarrow B_i \nsubseteq B_j \right\}$$

Set system is a unique-block cover with no blocks entirely included in others.

Ex. 2.3.3. *ϕ_S^3 from Example 2.3.2 is not a set system, but $\phi_S^4 = \{\overline{12}; \overline{23}; \overline{45}\}$ is.* ∎

Definition 2.14 (Partition). *Partition π_S on S is defined as*

$$\pi_S = \left\{ B_i | B_i \subseteq S \ \wedge \ \bigcup_i B_i = S \ \wedge \ i \neq j \Rightarrow B_i \cap B_j = \emptyset \right\}$$

Partition is a set system with non-overlapping blocks.

Ex. 2.3.4. *ϕ_S^4 from Example 2.3.3 is not a partition, but $\phi_S^5 = \{\overline{12}; \overline{3}; \overline{45}\}$ is.* ∎

For a given $s \in S$ the set of blocks of cover containing s is denoted as $[s]\phi$. Similarly, $[\mathcal{S}]\phi$ for $\mathcal{S} \subseteq S$ is the set of blocks containing all of the elements $s \in \mathcal{S}$ ($[\mathcal{S}]\phi = \bigcap_{s \in \mathcal{S}} [s]\phi$).

Definition 2.15 (Partition product). *A product of partitions π_S^1 and π_S^2 is a partition π_S such that:*

$$\pi_S = \{ B_i \mid \exists B^1 \in \pi_S^1 \exists B^2 \in \pi_S^2 \ : \ B_i = B^1 \cap B^2 \quad \wedge \quad i \neq j \Rightarrow B_i \cap B_j = \emptyset \}$$

Ex. 2.3.5. $\{\overline{1,2}; \overline{3,4,5}\} \cdot \{\overline{1,5}; \overline{2}; \overline{3,4}\} = \{\overline{1}; \overline{2}; \overline{3,4}; \overline{5}\}$ ∎

Definition 2.16 (Unique-block cover product). *A product of unique-block covers ϕ_S^1 and ϕ_S^2 is a unique-block cover ϕ_S such that:*

$$\phi_S = \{ B_i \mid \exists B^1 \in \phi_S^1 \exists B^2 \in \phi_S^2 \ : \ B_i = B^1 \cap B^2 \quad \wedge \quad i \neq j \Rightarrow B_i \neq B_j \}$$

Ex. 2.3.6. $\{\overline{1,2}; \overline{1,2,3}; \overline{4,5}\} \cdot \{\overline{1,3}; \overline{2,3,4,5}\} = \{\overline{1}; \overline{2}; \overline{1,3}; \overline{2,3}; \overline{4,5}\}$ ∎

Covers are multiplied by intersecting their blocks and removing blocks included in other blocks (for set systems) or identical blocks (for unique-block covers).

Definition 2.17 (Cover sum). *A sum of covers ϕ_S^1 and ϕ_S^2 is a* partition *π_S such that $[s]\pi_S = [t]\pi_S$ if and only if a sequence $s_0 = s, s_1, \ldots, s_n = t$, $s_i \in S$ exists where either $[s_i]\phi_S^1 \cap [s_{i+1}]\phi_S^1 \neq \emptyset$ or $[s_i]\phi_S^2 \cap [s_{i+1}]\phi_S^2 \neq \emptyset$.*

Ex. 2.3.7. $\{\overline{1,2}; \overline{2,3}; \overline{4}; \overline{5}\} + \{\overline{1}; \overline{2}; \overline{3}; \overline{4,5}\} = \{\overline{1,2,3}; \overline{4,5}\}$ ∎

A sum of covers is formed by merging blocks that have any element in common.

Definition 2.18 (Cover relation $\leq$). $\phi_1 \leq \phi_2 \iff \forall B^1 \in \phi^1 \exists B^2 \in \phi^2 : B^1 \subseteq B^2$

Ex. 2.3.8. *$\{\overline{1,2,3}; \overline{1,3}; \overline{4}; \overline{5}\} \leq \{\overline{1,2,3}; \overline{1,2}; \overline{4,5}\}$ but for $\phi^1 = \{\overline{1,2,3}; \overline{1,3}; \overline{4}; \overline{5}\}$ and $\phi^2 = \{\overline{1,2}; \overline{3}; \overline{4,5}\}$ neither $\phi^1 \leq \phi^2$ nor $\phi^2 \leq \phi^1$* ∎

Definition 2.19 (Zero-cover). *A zero-cover is a partition $\pi_S(0)$ such that*

$$\pi_S(0) = \{B_i \mid |B_i| = 1 \quad \wedge \quad \bigcup_i B_i = S\}$$

A zero-cover puts each element of S in a separate block.

Definition 2.20 (One-cover). *A one-cover is a partition $\pi_S(1) = \{S\}$.*

A one-cover is a cover with all elements in one block.

Note that for any ϕ_S on any S the following equalities hold:

$$\begin{aligned} \phi_S + \phi_S(0) &= \phi_S \\ \phi_S + \phi_S(1) &= \phi_S(1) \\ \phi_S * \phi_S(0) &= \phi_S(0) \\ \phi_S * \phi_S(1) &= \phi_S \end{aligned}$$

Definition 2.21 (Induced cover). $\phi_{A\times B} = ind^A_{A\times B}(\phi_A) = \{X \times B | X \in \phi_A\}$

$\phi_{A\times B}$ is a cover on $A \times B$ induced by ϕ_A if it keeps all pairs of $A \times B$ involving the elements of A placed in the same block of ϕ_A in one block.

Ex. 2.3.9. *For $A = \{a, b, c, d\}$, $B = \{0, 1\}$ and $\phi_A = \{\overline{a, b}; \overline{b}; \overline{c, d}\}$, $ind^A_{A\times B}(\phi_A) = \{\overline{(a,0)(a,1)(b,0)(b,1)}; \overline{(b,0)(b,1)}; \overline{(c,0)(c,1)(d,0)(d,1)}\}$* ∎

2.3.1 Dichotomies

For the special case of two-block set system, a useful shorthand notation can be introduced in the form of *dichotomy*.

Definition 2.22 (Unordered dichotomy). *An unordered dichotomy on S is a set of two disjoint subsets of S.*

Note that the union of blocks of a dichotomy does not necessarily cover S, what is required for a cover. However, a dichotomy can be transformed into a valid set system in a straightforward manner by placing the missing symbols in *both* blocks. Thus, a dichotomy can be treated as a compact notation of a two-block set system. As we will show in the following, two-block set systems play major role in modeling binary information and the introduction of dichotomies and their associated operations simplifies analysis and manipulation of the information without any loss of generality.

In some applications, such as state assignment, it is necessary to establish an ordering of the dichotomy's blocks, and associate a number (0 or 1) with each of the blocks.

Definition 2.23 (Ordered dichotomy). *An ordered dichotomy on S is an ordered pair of two disjoint subsets of S. The first subset is referred to as left- or zero-block, while the second is a right- or one-block.*

In the following, we will mostly deal with *unordered* dichotomies and refer to them in short as dichotomies. For the compactness of notation and where appropriate, we will denote an unordered dichotomy $d = \{P; Q\}$ as P/Q and an ordered dichotomy $e = (P, Q)$ as $P//Q$.

Ex. 2.3.10. *Consider set $S = \{1, \ldots, 5\}$ and a set system $\phi_S = \{\overline{1,2,3}; \overline{3,4,5}\}$ on S. An unordered dichotomy corresponding to ϕ_S can be created by removing the repeated symbols (3) from both blocks. The resulting dichotomy is $d = \{\{1,2\};\{4,5\}\} = 12/45$. The dichotomy d has two ordered versions: $e = (\{1,2\},\{4,5\}) = 12//45$ and $f = (\{4,5\},\{1,2\}) = 45//12$.* ∎

Definition 2.24 (Unordered dichotomy compatibility). *Two unordered dichotomies $d_1 = \{P_1; Q_1\}$ and $d_2 = \{P_2, Q_2\}$ are compatible (denoted as $d_1 \sim d_2$) iff $P_1 \cap Q_2 = \emptyset \ \wedge \ Q_1 \cap P_2 = \emptyset$ (direct compatibility, denoted $\sim_+$) or $P_1 \cap P_2 = \emptyset \ \wedge \ Q_1 \cap Q_2 = \emptyset$ (inverse compatibility, denoted $\sim_-$)*

Definition 2.25 (Ordered dichotomy compatibility). *Two ordered dichotomies $d_1 = (P_1, Q_1)$ and $d_2 = (P_2, Q_2)$ are compatible (denoted as $d_1 \sim d_2$) iff $P_1 \cap Q_2 = \emptyset \ \wedge \ Q_1 \cap P_2 = \emptyset$*

Note that while unordered dichotomies can be compatible either in direct or inverse form, ordered dichotomies can only be compatible in direct form.

For two compatible ordered or unordered dichotomies we define the operation of *merging* with as result a dichotomy with the blocks being the union of the corresponding blocks of the component dichotomies.

Definition 2.26 (Dichotomy merging operator $*$).

$$d_1 = \{P_1; Q_1\} \ \wedge \ d_2 = \{P_2; Q_2\} \ \wedge \ d_1 \sim_+ d_2 \ \Rightarrow \ d_1 *_+ d_2 = \{P_1 \cup P_2; Q_1 \cup Q_2\}$$

(direct merging of unordered dichotomies)

$$d_1 = \{P_1; Q_1\} \ \wedge \ d_2 = \{P_2; Q_2\} \ \wedge \ d_1 \sim_- d_2 \ \Rightarrow \ d_1 *_- d_2 = \{P_1 \cup Q_2; Q_1 \cup P_2\}$$

(inverse merging of unordered dichotomies)

$$d_1 = (P_1; Q_1) \ \wedge \ d_2 = (P_2; Q_2) \ \wedge \ d_1 \sim d_2 \ \Rightarrow \ d_1 * d_2 = (P_1 \cup P_2, Q_1 \cup Q_2)$$

(merging of ordered dichotomies)

The covering relation indicates the fact that the blocks of one dichotomy are subsets of the blocks of the other dichotomy

Definition 2.27 (Dichotomy covering relation $\subseteq$).

$$\text{For } d_1 = \{P_1; Q_1\}, d_2 = \{P_2; Q_2\} \qquad \big(d_1 \subseteq d_2 \iff P_1 \subseteq P_2 \wedge Q_1 \subseteq Q_2 \vee \ \vee \ P_1 \subseteq Q_2 \wedge Q_1 \subseteq P_2\big)$$

$$\text{For } d_1 = (P_1, Q_1), d_2 = (P_2, Q_2) \qquad \big(d_1 \subseteq d_2 \iff P_1 \subseteq P_2 \wedge Q_1 \subseteq Q_2\big)$$

Ex. 2.3.11. *Consider $S = \{1 \ldots 6\}$ and unordered dichotomies $d_1 = 12/45$, $d_2 = 3/4$, $d_3 = 3/6$. d_1 and d_2 are directly compatible, so they can be directly merged and form $d_{12} = d_1 *_+ d_2 = 123/45$. d_1 and d_2 are not inversely compatible, since the second block of d_1 and the second block of d_2 share an element (4). d_1 and d_3 are both directly and inversely compatible, so they can be merged and form $d_{13} = d_1 *_+ d_3 = 123/456$ or $d_{31} = d_1 *_- d_3 = 126/345$. Also, ordered dichotomies $e_1 = 12//45$ and $e_2 = 3//6$ are compatible and can be merged to form $e_{12} = e_1 * e_2 = 123//456$.* ∎

2.4 Information analysis

In the recent years, a number of promising approaches to problems in diverse fields have been proposed that are based on the analysis and manipulation of information. These approaches use set systems (also named *blankets* or *rough sets* by some authors) to model information in applications in many fields of modern engineering and science, including logic and architecture synthesis for VLSI systems [32, 34–37, 39–41, 47, 49, 50, 66, 67] pattern analysis, knowledge discovery, machine learning, neural network training, decision systems, data bases, encryption, compaction, encoding etc. [9, 19, 58–60, 68, 70, 73].

The work in information modeling with partitions and set systems was initiated by Hartmanis and Stearns who used partitions and set systems for the analysis of algebraic structure of finite state machines [23], and applied the analysis results to reason about special decompositions (i.e. parallel and serial) and state assignment of FSMs. Although the work of Hartmanis and many authors following him was based on intuitive understanding that set systems model some information present in the considered system, it lacked a precise apparatus for expressing *what* information is actually modeled and for analysis of the modeled information streams and their relationships. This problem was alleviated with the introduction of Information Relationships and Measures (IRM) by Jóźwiak [36].

The IRM framework introduces a concept of *elementary information item* or *atom*. Using this concept, it is possible to analyze any information flowing or used within the modeled system as a collection of atomic information items from a certain set of elementary information items. The representation of information as a set enables the application of the set-theoretical operators and methods to reason about relations and attributes of different information flows. In particular, IRM apparatus facilitates the following aspects of system analysis:

- analysis of the information flows — where and how a particular information is produced, and where and how it is consumed,
- analysis of the relationships (similarity, difference) between various information flows,
- introduction of the quantitative flavor (quantity, importance, weight) to characterize the analyzed information flows and their relationships.

The IRM apparatus forms the basis for much of the work presented in this thesis. In the remainder of this section we will therefore introduce basic notions of the information modeling and analysis. We illustrate these notions with the examples from the domain of this thesis, i.e. the synthesis of digital circuits. Note, however, that the apparatus is very general and can be used for analysis of any systems involving discrete functions, relations or finite state machines.

2.4.1 Information model

Elementary information

As discussed in the introduction, the behavior and structure of digital circuits are commonly specified using high-level (hardware) description languages. Despite different

form and syntax, various high-level language specifications of digital circuits and their compiled versions represent some networks of discrete functions, relations or sequential machines. Discrete functions, relations and sequential machines define some mappings between elements of some finite discrete sets (e.g. inputs, states, outputs). Let us use the term *symbol* to designate an element of such a set.

For instance, in the case of a finite state machine, the current conditions of the machine are determined by the values of the primary inputs and the present state of the machine. Therefore, we can interpret an FSM as an information processing system that combines partial information about the conditions delivered by its primary inputs with the information delivered by the current state variable to obtain the full information about the conditions. Then, FSM uses that combined information to determine the next state and the primary output. In this interpretation, introduced by Jóźwiak in [35], FSM processes information about "condition-symbols", where each symbol is associated with a separate input/current-state condition. Consider, for example, the FSM in Fig. 2.5. It has two binary primary inputs, three states and a single binary primary output. In the figure, we associated with each of the seven input/current-state combinations a symbol from a set $S = \{0 \ldots 6\}$.

To model the information about the symbols delivered by different sources (primary inputs and current state), we use the definition of information formulated in the Information Relationships and Measures apparatus. In the IRM interpretation, **information about symbols pertains to the ability to distinguish certain symbols from other symbols**. From this formulation follows the definition of an **elementary information item** as a **simplest possible distinction between two particular symbols**.

Definition 2.28 (Elementary information). *An elementary information describes the ability to distinguish a certain single symbol s_i from another single symbol s_j ($s_i, s_j \in S$ and $s_i \neq s_j$).*

Consider, for instance, information about symbols from S delivered by the first primary input x_1 of the FSM in Fig. 2.5. The fact that the variable has different values for the conditions 0 and 1 means that the variable is able to distinguish condition 0 from 1 and, therefore, the information delivered by this variable will contain the elementary distinction $0|1$. This fact means that, knowing that the current condition is *either* 0 or 1, the variable x_1 is capable of indicating which of the two it is. Note that this is different from saying that the variable is capable of indicating whether the current condition is 0. This would require distinguishing condition 0 from *all other* conditions, not just 1, and would be described by a set of elementary information items $\{0|1; 0|2; 0|3; 0|4; 0|5; 0|6\}$.

S	x_1x_2	p	n	y_1
0	$0-$	s_0	s_1	0
1	$1-$	s_0	s_2	1
2	00	s_1	s_0	0
3	$1-$	s_1	s_2	1
4	-1	s_1	s_2	1
5	-0	s_2	s_1	0
6	-1	s_2	s_2	1

Figure 2.5. Example finite state machine

Analyzing the values of x_1 for the remaining conditions (symbols of S), we conclude that the two values of x_1 allow to distinguish between the following symbols: $0|1$, $0|3$, $1|2$ and $2|3$. Since the value of x_1 for condition 4 is don't-care, which indicates 0 and 1, the variable is incapable of distinguishing 4 from any other condition. The set of atomic distinctions realized by variable x_1 is called an *information set of* x_1 (denoted $IS(x_1) = \{0|1, 0|3, 1|2, 2|3\}$) and it models the entire information about the symbols delivered by this variable.

Definition 2.29 (Information set). *A given (partial) discrete information about a set of symbols S can be represented by an information set IS defined on $S \times S$ as follows: $IS = \{\{s_i, s_j\} | s_i$ is distinguished from s_j by the given information $\}$.*

In an analogous way, the concept of *abstraction* was introduced in [36] that expresses the *inability* of the given information to distinguish between symbols. The elementary item of abstraction is defined as follows.

Definition 2.30 (Elementary abstraction). *An elementary abstraction describes the inability to distinguish a certain single symbol s_i from another single symbol s_j ($s_i, s_j \in S$ and $s_i \neq s_j$).*

The total abstraction associated with the given information stream can be described by an abstraction set.

Definition 2.31 (Abstraction set). *An abstraction on a set of symbols S corresponding to a given (partial) information can be represented by an abstraction set AS defined on $S \times S$ as follows: $AS = \{\{s_i, s_j\} | s_i$ is not distinguished from s_j by the given information $\}$.*

Note that for a given information stream, the information and abstraction sets are complementary and their union is $S \times S$. Therefore, information set unambiguously identifies the abstraction set and vice versa.

Set systems

Another alternative of expressing abstraction and, by extension, the corresponding information is to identify blocks of symbols that are not distinguished by the modeled information. In our example, the variable x_1 through its two values 0 and 1 induces two compatibility classes on the symbols: $B^0 = \{0, 2, 4, 5, 6\}$ and $B^1 = \{1, 3, 4, 5, 6\}$. x_1 has value $0(1)$ for each symbol in class $B^0(B^1)$ (don't-care "$-$" means: 0 and 1). Thus, variable x_1 is not able to distinguish between symbols 0, 2, 4, 5 and 6, because they belong to the same compatibility class. The set of compatibility classes associated with all values of a variable forms a *set system* that models the information delivered by the variable. In this case, the set system is $\phi_S(x_1) = \{\overline{0, 2, 4, 5, 6}; \overline{1, 3, 4, 5, 6}\}$.

There exist operations of conversion between a set system representing given information and the corresponding information set, and vice versa. Since the sets of distinguishable symbols and non-distinguishable symbols are complementary, to convert a set system to an information set, it is sufficient to list all pairs of symbols that are *not* placed together in any block of the set system. Conversely, to find a set system corresponding to an information set, one needs to start with a single block containing all symbols and, for all distinctions in IS, split all blocks containing both symbols of the considered distinction into two copies, each of the copies with one of the symbols removed. In this process, some of the blocks may become a subsets of other blocks, and they need to

be removed. For instance, if we list all 21 pairs of 7 symbols in S, we can verify that only the four pairs of symbols present in $IS(x_1)$ are not placed together in any of the two blocks of $\phi_S(x_1)$. To perform the reverse mapping, we start with a set system containing single block $\phi_S^1 = \{\overline{0,1,2,3,4,5,6}\}$. Then, we take the first distinction from $IS(x_1)$, $0|1$ and split the single block of ϕ_S^1 into two copies — one of the without 0, the other without 1. Thus, the set system becomes $\phi_S^2 = \{\overline{0,2,3,4,5,6}; \overline{1,2,3,4,5,6}\}$. Further, we split the first block containing both symbols of the next distinction, $0|3$, and obtain $\phi_S^3 = \{\overline{0,2,4,5,6}; \overline{2,3,4,5,6}; \overline{1,2,3,4,5,6}\}$. Since the second block of ϕ_S^3 is entirely included in the third block, it is removed and the set system becomes $\phi_S^3 = \{\overline{0,2,4,5,6}; \overline{1,2,3,4,5,6}\}$ For the distinction $1|2$, the second block is split into $\overline{1,3,4,5,6}$ and $\overline{2,3,4,5,6}$ resulting in $\phi_S^4 = \{\overline{0,2,4,5,6}; \overline{2,3,4,5,6}; \overline{1,3,4,5,6}\}$. Finally, the distinction $2|3$ splits the second block into $\overline{2,4,5,6}$, which is covered by the first block, and $\overline{3,4,5,6}$, which is covered by the third block, and thus the set system $\phi_S = \{\overline{0,2,4,5,6}; \overline{1,3,4,5,6}\}$ corresponding to $IS(x_1)$ is constructed.

While the procedure described above generates a unique set system for any information set, in general there are multiple set system representations for the same information set. Consider, for instance two set systems on $S = \{1 \dots 5\}$: $\phi_S^1 = \{\overline{123}, \overline{45}\}$ and $\phi_S^2 = \{\overline{12}, \overline{23}, \overline{13}, \overline{45}\}$. For both set systems, the corresponding information set is $IS = \{1|4, 1|5, 2|4, 2|5, 3|4, 3|5\}$. The issue of multiple set system representation was discussed in depth in [75]. It was shown there that for any information set there exists a *canonical* set system representation that is unique. The blocks of canonical set system have to be the *maximal* compatibility classes of symbols. In the above example, ϕ_S^1 is the canonical representation of IS, while ϕ_S^2 is not, because the three first blocks of ϕ_S^2 can be merged to a larger compatibility class $\overline{123}$. Note, however, that the conversion procedure described above will always create a canonical set system, as it only separates the symbols that *cannot* be in the same block, and therefore it implicitly preserves maximal compatibility classes of symbols.

At this point we note the two special set systems, zero-cover and one-cover (see Def. 2.19 and 2.20). We recall that zero-cover is in fact a partition with all symbols in separate blocks. Its corresponding information set contains therefore *all* distinctions of its symbols, and in this way models *full* information about the symbols. This situation corresponds to the modeled variable having separate value for each of the symbols. On the other end of the spectrum is one-cover, which contains all symbols in one block and, therefore, corresponds to an empty information set. It represents the variable with don't-cares (or the same, single value) for all symbols and therefore, it does not deliver any information. Note that the initial cover in the construction of a cover corresponding to a given information set is a one-cover, so there are no distinctions represented. In the following construction steps, the covers represent information sets with one additional distinction, until all required distinctions are represented.

The above notions, illustrated with the examples of binary variables, also hold for the multi-valued variables. In any case, each value of a variable defines a single block of a set system representing the information associated with this variable. In this block, all symbols associated with this value of the variable are placed. For instance, the present-state variable p of the FSM in Fig. 2.5, induced through its three values three blocks of symbols: $\overline{0,1}$ (for value s_0), $\overline{2,3,4}(s_1)$ and $\overline{5,6}(s_2)$. Thus, the set system associated with p is $\phi_S(p) = \{\overline{0,1}; \overline{2,3,4}; \overline{5,6}\}$. As illustrated above, the information set corresponding to $\phi_S(p)$ can be determined by listing all symbol pairs not placed together in a block

of $\phi_S(p)$. This results in $IS(p) = \{0|2, 0|3, 0|4, 0|5, 0|6, 1|2, 1|3, 1|4, 1|5, 1|6, 2|5, 2|6, 3|5, 3|6, 4|5, 4|6\}$.

While the distinctions relating to the input variables are interpreted as information delivered by the variables, the distinctions relating to the output variables can be interpreted as the information *required* to compute the given output. Take, for instance, the output y_1 of the FSM in Fig. 2.5. Its set system is $\phi_{y_1} = \{\overline{0,2,5}; \overline{1,3,4,6}\}$, and the information set is $IS(\phi_{y_1}) = \{0|1, 0|3, 0|4, 0|6, 1|2, 1|5, 2|3, 2|4, 2|6, 3|5, 4|5, 5|6\}$. Clearly, to be able to decide whether the output is 0 or 1, the FSM needs to know whether the current condition corresponds to one of the symbols $\{0, 2, 5\}$ or $\{1, 3, 4, 6\}$. This requires the information items contained in $IS(\phi_{y_1})$. On the other hand, the output variable can be viewed as an input to the "user" of the FSM, and thus considered to deliver its information to the environment.

To summarize, the corresponding set systems and information sets for all (primary and state) inputs and outputs of the finite state machine shown in Fig. 2.5 are as follows:

$$
\begin{aligned}
&\phi_{x_1} = \{\overline{0,2,4,5,6}; \overline{1,3,4,5,6}\}, \phi_{x_2} = \{\overline{0,1,2,3,5}; \overline{0,1,3,4,6}\},\\
&\phi_p = \{\overline{0,1}; \overline{2,3,4}; \overline{5,6}\}, \phi_n = \{\overline{2}; \overline{0,5}; \overline{1,3,4,6}\}, \phi_{y_1} = \{\overline{0,2,5}; \overline{1,3,4,6}\},\\
&IS(\phi_{x_1}) = \{0|1, 0|3, 1|2, 2|3\}, IS(\phi_{x_2}) = \{2|4, 2|6, 4|5, 5|6\},\\
&IS(\phi_p) = \{0|2, 0|3, 0|4, 0|5, 0|6, 1|2, 1|3, 1|4, 1|5, 1|6, 2|5, 2|6, 3|5, 3|6, 4|5, 4|6\},\\
&IS(\phi_n) = \{0|1, 0|2, 0|3, 0|4, 0|6, 1|2, 1|5, 2|3, 2|4, 2|5, 2|6, 3|5, 4|5, 5|6\},\\
&IS(\phi_{y_1}) = \{0|1, 0|3, 0|4, 0|6, 1|2, 1|5, 2|3, 2|4, 2|6, 3|5, 4|5, 5|6\}
\end{aligned}
$$

Information-related set system operations

From the point of view of information manipulation, there are two important set system operations: multiplication of set systems (Def. 2.16) and smaller-or-equal relation (Def. 2.18).

The *multiplication* of set systems corresponds in the information set domain to the sum of information modeled by the set systems. In this way, it is a very useful operation that allows combining information delivered by multiple information streams. For instance, the total information delivered by the two set systems $\phi_S^1 = \{\overline{12}, \overline{234}\}$ and $\phi_S^2 = \{\overline{134}, \overline{23}\}$ is described by the product set system $\phi_S = \phi_S^1 \cdot \phi_S^2 = \{\overline{1}; \overline{23}; \overline{34}\}$. This is reflected by the fact that $IS(\phi_S) = IS(\phi_S^1) \cup IS(\phi_S^2)$. An important result proven in [75] shows that multiplication preserves canonicity of set systems. Therefore, information streams can be combined with the multiplication operation preserving unambiguity of information representation.

The *smaller-or-equal relationship* between two set systems implies that the information set corresponding to the smaller set system is a superset of the information set of the larger set system. In other words, the smaller set system provides *more* information. This is a direct consequence of the fact that a smaller set system has smaller blocks, and therefore provides less abstraction and more information. Consider, for instance, two set systems on $S = \{1 \dots 4\}$: $\phi_S^1 = \{\overline{12}, \overline{34}\}$ and $\phi_S^2 = \{\overline{1}, \overline{2}, \overline{34}\}$ with information sets $IS(\phi_S^1) = \{1|3, 1|4, 2|3, 2|4\}$ and $IS(\phi_S^2) = \{1|2, 1|3, 1|4, 2|3, 2|4\}$ Clearly, $\phi_S^2 \leq \phi_S^1$ and

$IS(\phi^2_S) \supseteq IS(\phi^1_S)$. This is because the two blocks $\overline{1}$ and $\overline{2}$ of ϕ^2_S that are included in the block $\overline{12}$ of ϕ^1_S allow to distinguish symbol 1 from 2 that are indistinguishable by ϕ^1_S.

Note that smaller-or-equal relationship *implies* information set inclusion, but is not equivalent to it. I.e. it is a sufficient but not necessary condition. This is caused by the canonicity issues of the set system representation that were discussed in the previous section. In particular, it may happen that two non-canonical set systems ϕ^1_S and ϕ^2_S have identical information sets, while neither $\phi^1_S \leq \phi^2_S$ nor $\phi^2_S \leq \phi^1_S$. This is, for instance, the case for the following two set systems: $\phi^1_S = \{\overline{123}, \overline{45}, \overline{56}, \overline{46}\}$ and $\phi^2_S = \{\overline{12}, \overline{23}, \overline{13}, \overline{456}\}$. However, as shown in [75], if the set systems are canonical, the smaller-or-equal relationship is both sufficient and necessary condition for the information set inclusion.

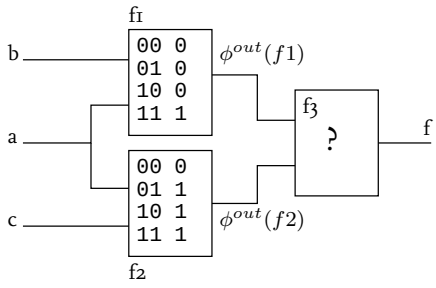
The concept of information set inclusion, and therefore of the smaller-or-equal set system relation, is crucial to the analysis of information flows. In particular, given two set systems ϕ^i_S and ϕ^o_S, if the information set of ϕ^i_S is a superset of the information set of ϕ^o_S, it is possible to build a relation (or function) that has as input the information stream ϕ^i_S and produces output information stream ϕ^o_S. If both set systems are canonical, this requirement is equivalent to $\phi^i_S \leq \phi^o_S$.

Let us illustrate the usage of multiplication and $\leq$ operator to the analysis of information flows in a Boolean function network. Consider, for instance, the incompletely specified Boolean function in Fig. 2.6(a). It has three inputs: a, b and c and output f. In the figure, all the possible input combinations of the function were associated with symbols from set $S = \{0 \dots 7\}$. The information about input combination required to compute the output f is given by set system $\phi(f) = \{\overline{02367}; \overline{1345}\}$. At the same time, each of the inputs delivers a partial information about input combination given by the following set systems: $\phi(a) = \{\overline{0123}; \overline{4567}\}$, $\phi(b) = \{\overline{0145}; \overline{2367}\}$ and $\phi(c) = \{\overline{0246}; \overline{1357}\}$. The realization network for the function f is presented in Fig. 2.6(b). The network is partially built with two sub-functions $f1$ and $f2$ already built and the third sub-function $f3$ being constructed. The analysis of information flows will allow us to analyze how the input information was filtered by $f1$ and $f2$; if the filtered information is sufficient to construct the missing sub-function; and what function has to be realized by $f3$.

The input information available to $f1$ is a combination of information delivered by inputs a and b. It is expressed by the input set system $\phi^{in}(f1) = \phi(a) \cdot \phi(b) = \{\overline{01}(00); \overline{23}(01); \overline{45}(10); \overline{67}(11)\}$. The values in parenthesis indicate what combination of the variables a and b is associated with a particular block of the input set system. The

abc	f	S
000	0	0
001	1	1
010	0	2
011	–	3
100	1	4
101	1	5
110	0	6
111	0	7

(a) Example function



(b) Realization network

Figure 2.6. Example of information flow analysis

function $f1$ assigns one output value (0) to its input combinations $00, 01$ and 10. For the output set systems of function $f1$, $\phi^{out}(f1)$, it means that the function abstracts from any distinction between these three situations, and therefore in the output set system the three blocks of the input set system will be merged together. The fourth block of $\phi^{in}(f1)$ is assigned a separate value 1 by $f1$, and therefore will be preserved in the output set system. Thus, the output set system becomes $\phi^{out}(f1) = \{\overline{012345}(0); \overline{67}(1)\}$, with the output value associated with each block given in parenthesis.

In an analogous manner, the input information of $f2$ is $\phi^{in}(f2) = \phi(a) \cdot \phi(c) = \{\overline{02}(00); \overline{13}(01); \overline{46}(10); \overline{57}(11)\}$. The function $f2$ abstracts from input combinations $01, 10$ and 11 and therefore the output set system of $f2$ is $\phi^{out}(f2) = \{\overline{02}(0); \overline{134567}(1)\}$.

We can see that in both cases the input set systems are smaller than the output set systems. It is a natural consequence of the fact that a function can only abstract from the input information, i.e. assign same value to different input combinations, and therefore the output set system is always a result of some merging of the blocks of the input set system.

The input of $f3$ is formed by two information streams: first from $f1$, described by $\phi^{out}(f1)$, and second from $f2$, described by $\phi^{out}(f2)$. The input information of $f3$ is therefore $\phi^{in}(f3) = \phi^{out}(f1) \cdot \phi^{out}(f2) = \{\overline{02}(00); \overline{1345}(01); \overline{67}(11)\}$. To check whether it is possible to build a function that will produce output f using this input information, we just need to check the condition $\phi^{in}(f3) \leq \phi(f)$. In this case, we can see that the condition is fulfilled. By merging the first and third block of $\phi^{in}(f3)$, a two-block set system $\phi'(f) = \{\overline{0267}; \overline{1345}\}$ is built that is smaller than $\phi(f)$, and therefore represents output of function f with some don't-cares filled. In this case, for input condition 3, which was a don't-care condition in f the value 1 was assigned, making f' a completely specified realization of the incompletely specified function f. At the same time, the merging of blocks of $\phi^{in}(f3)$ defines the function of the block $f3$. Since the blocks corresponding to input combinations 00 and 11 were merged together to form the output set systems $\phi'(f)$, the function $f3$ will have the same output value 0 for these two input combinations. The second block of the input set system associated with input combination 01 was not modified in the output set system, and therefore the value of $f3$ for input 01 is 1. We can see that the input combination 10 does not appear at all in the input set system, what indicates that this is a don't-care condition of $f3$, which can be used for minimization of $f3$.

Dichotomies

As discussed in Section 2.3.1, dichotomies can be used as a short-hand notation for two-block set systems and therefore can be used to model information delivered by binary variables. Since a dichotomy is exactly equivalent to the corresponding canonical set system, we can unambiguously associate the set system's information set with the dichotomy.

Consider, for instance, the set system $\phi = \{\overline{123}; \overline{2345}\}$. The corresponding dichotomy d is created by removing repeating symbols from both blocks. Thus, $d = 1/45$. The information set of d is equal to the information set of ϕ and is $IS = \{1|4, 1|5\}$. In the case of dichotomies the information set can be derived even more easily by simply enumerating all pairs that the symbols in the left block can form with the symbols in the right block.

Since binary variables are prevalent in today's implementations of digital systems, the dichotomies as the information modeling tool for these variables are very useful. In particular, in this thesis we will show how dichotomies can be used to represent information about states of a finite state machine delivered by binary encoding variables.

2.4.2 Information relationships and measures

Representation of information as an information set means that information relationships between variables or set systems representing various information streams can be analyzed by considering relationships between their corresponding information sets. For instance, a question about common information delivered by two variables can be answered by identifying the elementary distinctions present in the information sets of both variables. The fact that information is composed of discrete atoms makes it also possible to measure the *amount* of information as the number of atoms, or to associate importance, or weight, to a particular information item.

In [36][37], an appropriate analysis apparatus is proposed that exploits these characteristics: the theory of information relationships and measures. In particular, a number of relationships between information streams is defined in [36][37] :

- *common information* CI (i.e. information that is present in both ϕ_1 and ϕ_2): $CI(\phi_1, \phi_2) = IS(\phi_1) \cap IS(\phi_2)$
- *total (combined) information* TI (i.e. information that is present in either ϕ_1 or ϕ_2): $TI(\phi_1, \phi_2) = IS(\phi_1) \cup IS(\phi_2)$
- *missing information* MI (i.e. information that is present in ϕ_1 but missing in ϕ_2): $MI(\phi_1, \phi_2) = IS(\phi_1) \setminus IS(\phi_2)$
- *extra information* EI (i.e. information that is missing in ϕ_1 but present in ϕ_2): $EI(\phi_1, \phi_2) = IS(\phi_2) \setminus IS(\phi_1)$
- *different information* DI $DI(\phi_1, \phi_2) = MI(\phi_1, \phi_2) \cup EI(\phi_1, \phi_2)$

Also for abstraction, a number of relationships is defined in [36] that describes common, total, missing, extra and different abstraction.

Consider, for instance, the set systems on p. 38 modeling information flows in the FSM in Fig. 2.5. The information delivered by the present state variable p is described by the set system $\phi_p = \{\overline{0,1}; \overline{2,3,4}; \overline{5,6}\}$ with the corresponding information set $IS(\phi_p) = \{0|2, 0|3, 0|4, 0|5, 0|6, 1|2, 1|3, 1|4, 1|5, 1|6, 2|5, 2|6, 3|5, 3|6, 4|5, 4|6\}$. The information required to calculate the next state variable n is modeled by the set system $\phi_n = \{\overline{2}; \overline{0,5}; \overline{1,3,4,6}\}$ and the information set $IS(\phi_n) = \{0|1, 0|2, 0|3, 0|4, 0|6, 1|2, 1|5, 2|3, 2|4, 2|5, 2|6, 3|5, 4|5, 5|6\}$. The common information $CI(\phi_n, \phi_p)$, i.e. the information used by n and delivered by p is determined by $IS(\phi_n) \cap IS(\phi_p) = \{0|2, 0|3, 0|4, 0|6, 1|2, 1|5, 2|5, 2|6, 3|5, 4|5\}$. The information used by n but missing in p is described by $MI(\phi_n, \phi_p) = IS(\phi_n) \setminus IS(\phi_p) = \{0|1, 2|3, 2|4, 5|6\}$. The extra information present on p but not used by n is $EI(\phi_n, \phi_p) = IS(\phi_p) \setminus IS(\phi_n) = \{0|5, 1|3, 1|4, 3|6, 4|6\}$.

The strength of information or abstraction relationships can be measured in the simplest manner by measuring the amount of common, total, etc. information (or abstraction), expressed as the number of information (or abstraction) items. This way, the following measures can be defined:

- *information similarity* $ISIM(\phi_1, \phi_2) = |CI(\phi_1, \phi_2)|$
- *information difference* $IDIS(\phi_1, \phi_2) = |DI(\phi_1, \phi_2)|$
- *information decrease (loss)* $IDEC(\phi_1, \phi_2) = |MI(\phi_1, \phi_2)|$
- *information increase (growth)* $IINC(\phi_1, \phi_2) = |EI(\phi_1, \phi_2)|$
- *total information quantity* $TIQ(\phi_1, \phi_2) = |TI(\phi_1, \phi_2)|$

For example, the information similarity of the above mentioned variables p and n can be measured as $|CI(\phi_n, \phi_p)| = 10$, the information loss: $|MI(\phi_n, \phi_p)| = 4$ and the information growth: $|EI(\phi_n, \phi_p)| = 5$.

In [36][37] some normalized and weighted measures are also defined, by associating an appropriate importance weight $w(s_i|s_j)$ with each elementary information. The importance of information may be, for instance, related to its availability, i.e. the number of variables at which this information is present. Let o be a certain output variable, X be a set of some input variables of o and $ISS(X)$ be the set of information sets of variables from X.

Definition 2.32 (Occurance multiplicity of elementary information). *Occurrence multiplicity m of an elementary information $s_i|s_j$ from $IS(o)$ in $ISS(X)$ is defined as follows:*

$$m(s_i|s_j)\Big|_{ISS(X)}^{IS(o)} = \sum_{x \in X} occ(s_i|s_j)\Big|_{IS(x)}^{IS(o)}$$

where:

$$occ(s_i|s_j)\Big|_{IS(x)}^{IS(o)} = \begin{cases} 1 : \text{if}(s_i|s_j) \in IS(o) \cap IS(x) \\ 0 : \text{otherwise} \end{cases}$$

If $m(s_i|s_j)\Big|_{ISS(X)}^{IS(o)} = 1$, $s_i|s_j$ required by o is provided by only a single variable from X, then $s_i|s_j$ is called a *unique information* with respect to X. Unique information is of primary importance. For instance, in the example FSM in Fig. 2.5, the distinction $2|4$ is necessary to compute the next-state variable n ($2|4 \in IS(n)$). We can see that this item of information is only delivered by a single input variable x_2 ($2|4 \in IS(x_2)$). Therefore,

$$m(2|4)\Big|_{ISS(x_1,x_2,p)}^{IS(n)} = 1$$

and $2|4$ is an item of unique information.

The introduction of weights gives rise to the weighted information relationships measures, such as weighted information similarity measure:

$$WISIM(\phi_1, \phi_2) = \sum_{s_i|s_j \in CI(\phi_1,\phi_2)} w(s_i|s_j)$$

These measures reflect not only how much information is common, or different in the two information streams, but also how important the information is.

Analysis of information and information relationships enabled by IRM apparatus is of primary importance for analysis and synthesis of digital information systems. In particular, knowing the required and delivered information, amount and importance of information, relationships between the information in different information streams, the strength of these relationships, and combining this knowledge with the synthesis objectives and constraints, one can effectively and efficiently make design decisions, i.e. determine the structure of the required system in order to satisfy the given constraints and optimize the given objectives. In this thesis we will demonstrate how IRM can be used to analyze information flows in the encoded finite state machine and how the result of this analysis can be used to guide the encoding process towards efficient implementation of the machine.

Chapter 3

General Decomposition Theorem

The term *composition* means the act or state of arrangement of parts into a proper relation in order to form a whole (an aggregate). The term *decomposition* means the act or state of disintegration or breakdown of a whole into a system of constituent parts (elements or simpler aggregates). These two notions are very general and can be used in a lot of various contexts. In particular, decomposition is one of the basic analysis and synthesis concepts of complex systems. It consists in breaking down a complex system into a network of less complex and relatively independent collaborating sub-systems that are easier to analyze, comprehend, synthesize or implement, in such a way, that the original system's behavior is preserved.

In this chapter we will discuss the decomposition of finite state machines. Since a discrete relation (function) can be considered as a special case of an FSM with single state and trivial state-transition function, the results presented in this chapter also extend to the decomposition of discrete functions and relations. In the context of sequential machine analysis and synthesis, decomposition consists in representation of a given sequential machine as a composition of a number of collaborating partial machines that together realize behavior of the given machine. As discussed in Section 1.4, the behavior of the decomposed machine is realized by a network of partial machines with some interconnections between them, input encoder that distributes and encodes primary inputs of the original FSM to the partial machines and the output decoder that decodes the outputs of the partial machines to produce the primary output of the original FSM.

The central point of this chapter is a constructive theorem on the existence of general decomposition of an incompletely specified FSM. This theorem describes the decomposition in terms of covers on the input, output and state alphabets of the decomposed FSM. These covers model partial information about input, output and state of the decomposed machine that is available to the partial machines in the decomposition network. The theorem describes conditions under which the covers describe a valid decomposition. If these conditions are met, it is possible to derive from the covers state transition tables of the partial machines as well as the rules to connect them and the input encoder and output decoder functions. Thus, the theorem defines a generator of correct circuit structures. For a given function or sequential machine, correct circuits that realize the function/machine can be constructed by repetitive use of the generator or its special cases.

The theorem presented in this chapter is an extension of the General Decomposition Theorem for completely specified FSMs published by Jóźwiak in [35]. The extended theorem was formulated and proven by the author together with Jóźwiak and was originally published and discussed in [48]. The extension allows modeling of FSMs with incompletely specified output and next-state functions (i.e. non-determinism) and multi-state behavior realization (i.e. multiple states of the realization network corresponding to a single state of the realized machine). In this way it is the most general theorem describing decompositions of any finite state machine and, as a special case, of any discrete function or relation. To introduce the complex notions involved in formulation of the general case, we will start with the completely specified case and illustrate it with an example. Building on that, we will show what modifications to the formulation of the theorem are necessary to account for incompletely specified output and next-state functions and the multi-state behavior realization. This will lead us to the final formulation of the General Decomposition Theorem. Finally, we will present the proof of the extended theorem and an example to illustrate its working.

3.1 Decomposition of completely specified FSM

3.1.1 General Decomposition Theorem

In the view of the General Decomposition Theorem, the realization network is a system of information-processing sub-systems (partial machines). Each of these sub-systems requires some input information and produces some output information. The conditions stated in the General Decomposition Theorem in their essence require that the produced output information can be legally derived from the input and/or state information. These conditions are expressed in terms of partitions, set systems or covers modeling the input, state and output information streams of the realized machine and of the partial machines. In the case of completely specified FSMs, it is possible to use partitions instead of more general set systems or covers to model this information, as in the completely specified machine the distinctions between symbols are "crisp", i.e. the compatibility between the symbols that is introduced by don't-cares in specification is replaced with a strictly defined equivalence. Later, we will see that introduction of the output and state don't-cares requires using covers rather than partitions for information modeling.

For the input and output information streams described by the partitions on the same sets of symbols, the fact that the output information can be legally derived from the input is described by the $\leq$ operator. As discussed in Section 2.4, the $\leq$ operator indicates that the smaller partition has smaller blocks than the larger partition, and therefore delivers *more* information. Hence, it is always possible to construct a function that, based on the input information, will unambiguously produce the output information. Therefore, the requirement of non-increasing output information can be expressed as $\pi_X^{in} \leq \pi_X^{out}$.

The situation is more complicated when the information about one set symbols is used to compute information about another set of symbols. Then, the $\leq$ operator is not defined and a way to map between the sets of symbols is necessary. In the case of FSM, the total input information to a machine is an information about the current state and the primary input. Therefore, the input information can be described by a partition on the Cartesian product of the state and input alphabets — $S \times I$. The output information is the information about the next state (modeled by a partition on S) and information about the

primary output (modeled by a partition on O). To be able to determine whether the given input information about $S \times I$ is sufficient to compute the given output information about S or O, the concept of *partition pair* is introduced. A given partition $\pi_{S\times I}$ on $S \times I$ forms a pair with a given partition π_S on S or π_O on O if the input/state combinations placed in one block on $\pi_{S\times I}$ are mapped by δ and λ, respectively, to the a single block of π_S and π_O. Therefore, the existence of partition pair means that knowing the input/state conditions with the precision to a block of $\pi_{S\times I}$ makes it possible to calculate the next state or output information of a machine with the precision to a block of π_S or π_O, respectively. In other words, it is possible to build a function that will map the blocks of $\pi_{S\times I}$ to the blocks of π_S or π_O, and thus the output information can be legally derived from the input information.

Definition 3.1 (Block transition function).

$$\overline{\delta} : 2^{S\times I} \rightarrow 2^S \quad \textit{and} \quad \overline{\delta}(D) = \{s' \mid \delta(s,x) = s' \wedge (s,x) \in D \wedge D \subseteq S \times I\}$$

Definition 3.2 (Block output function).

$$\overline{\lambda} : 2^{S\times I} \rightarrow 2^O \quad \textit{and} \quad \overline{\lambda}(D) = \{o \mid \lambda(s,x) = o \wedge (s,x) \in D \wedge D \subseteq S \times I\}$$

Definition 3.3 ($S \times I - S$ partition pair). *$(\pi_{S\times I}, \pi_S)$ is a $S \times I - S$ partition pair iff*

$$\forall D \in \pi_{S\times I} \exists B \in \pi_S : \overline{\delta}(D) \subseteq B$$

Definition 3.4 ($S \times I - O$ partition pair). *$(\pi_{S\times I}, \pi_O)$ is a $S \times I - O$ partition pair iff*

$$\forall D \in \pi_{S\times I} \exists C \in \pi_O : \overline{\lambda}(D) \subseteq C$$

Ex. 3.1.1. *Consider the state transition table in Figure 3.1. For $D_1 = \{(a,0)(b,0)\}$, $\overline{\delta}(D_1) = \{\delta(a,0), \delta(b,0)\} = \{c\}$ and $\overline{\lambda}(D_1) = \{\lambda(a,0), \lambda(b,0)\} = \{00, 11\}$.*
For $D_2 = \{(a,1)(b,1)(c,0)(c,1)\}$, $\overline{\delta}(D_2) = \{a,b\}$ and $\overline{\lambda}(D_2) = \{01, 10\}$. Therefore, $\pi_{S\times I} = \{D_1, D_2\}$ forms a $S \times I - S$ pair with $\pi_S = \{\overline{a,b}; \overline{c}\}$ and $S \times I - O$ pair with $\pi_O = \{\overline{00, 11}; \overline{01, 10}\}$. ∎

in	ps	ns	out
0	a	c	00
1	a	a	01
0	b	c	11
1	b	b	01
0	c	a	10
1	c	b	10

Figure 3.1. Completely specified sequential machine

General decomposition consists of representation of a given sequential machine as a composition of a number of collaborating partial machines that together realize behavior of the given machine. The concept of *general composition* describes the system of partial machines together with their interconnections.

Definition 3.5 (General composition of sequential machines). *A general composition of n sequential machines* $\mathbf{M}_i$*,* $GC = (\{\mathbf{M}_i\}, \{Con_i\})$ *consists of the following objects:*

1. $\{\mathbf{M}_i = (I_i^*, S_i, O_i, \delta_i, \lambda_i),\ I_i^* = I_i \times I_i',\ 1 \leq i \leq n\}$*, a set of sequential machines referred to as* **component (partial) machines**.
2. $\{Con_i : \times O_j \rightarrow I_i', 1 \leq i, j \leq n\}$*, a set of surjective functions referred to as connecting rules of the component machines*

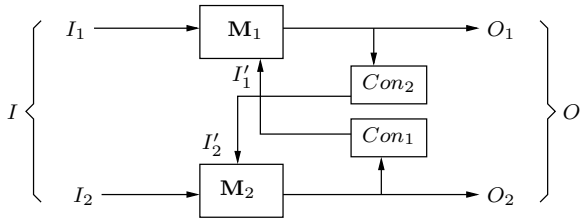


Figure 3.2. General composition of two component machines

Definition 3.6 (General full-decomposition). *The machine* $\mathbf{str}(\{\mathbf{M}_i\}, \{Con_i\}))$ *is a* general full-decomposition *of the machine* $\mathbf{M}$ *if and only if a general composition of* $\mathbf{M}_i$ *realizes* $\mathbf{M}$

Let π_I^i, π_S^i and $\pi_{S\times I}^i$ be partitions on $\mathbf{M} = (I, S, O, \delta, \lambda)$ on I, S and $S \times I$, respectively. Let $\pi_{S\times I}^{ij}$ be a partition on $S \times I$ such that $\pi_{S\times I}^i \leq \pi_{S\times I}^{ij}$ and ${\pi'_{S\times I}}^j = \prod_{i=1\ldots n} \pi_{S\times I}^{ij}$. Let $\pi_{S\times I} = \prod_{i=1\ldots n} \pi_{S\times I}^i$. Let ${\pi_{S\times I}^I}^i$ and ${\pi_{S\times I}^S}^i$ be partitions induced on $S \times I$ by π_I^i and π_S^i, respectively. $\pi_{S\times I}^I = \prod_{i=1\ldots n} {\pi_{S\times I}^I}^i$ and $\pi_{S\times I}^S = \prod_{i=1\ldots n} {\pi_{S\times I}^S}^i$.

Theorem 3.1 (General Decomposition Theorem for completely specified FSMs [35]). *A sequential machine* $\mathbf{M} = (I, S, O, \delta, \lambda)$ *has a general full decomposition with the output behavior realization with* n *component machines if and only if* n *trinities of partitions* $(\pi_I^i, \pi_S^i, \pi_{S\times I}^i)$ *exist, such that:*

1. $({\pi_{S\times I}^S}^i \cdot {\pi_{S\times I}^I}^i \cdot {\pi'_{S\times I}}^i, \pi_S^i)$ *is an* $S \times I - S$ *partition pair*
2. ${\pi_{S\times I}^S}^i \cdot {\pi_{S\times I}^I}^i \cdot {\pi'_{S\times I}}^i \leq \pi_{S\times I}^i$
3. $\pi_{S\times I}^S \cdot \pi_{S\times I}^I \leq {\pi'_{S\times I}}^i$
4. $(\pi_{S\times I}, \pi_O(0))$ *is an* $S \times I - O$ *partition pair*

   Additionally, if

5. $\prod_i \pi_S^i = \pi_S(0)$

   is satisfied, then the state behavior will be realized too.

A corresponding scheme of a general decomposition of $\mathbf{M}$ into n partial machines $\mathbf{M}_i$ is presented in Figure 3.3. It involves n partial machines $\mathbf{M}_1$ through $\mathbf{M}_n$. The state, input and output alphabets of each of the machines are defined by their corresponding partitions π_S^i, $\pi_I^i \times {\pi'_{S\times I}}^i$, and $\pi_{S\times I}^i$, respectively. Each partial machine $\mathbf{M}_i$ using its own state information (π_S^i), its primary input information (π_I^i) and information imported from some other machines (${\pi'_{S\times I}}^i$), after combining all this information computes its own next-state (π_S^i) and output ($\pi_{S\times I}^i$) information, being a partial state and output information of the original specification machine $\mathbf{M}$. The connections between machines are realized by the connection blocks Con_i. Each block Con_i delivers to its corresponding partial machine $\mathbf{M}_i$ the imported information extracted from the combined output information of some other partial machines and represented by the partition ${\pi'_{S\times I}}^i$. The input information of each partial machine $\mathbf{M}_i$ (π_I^i) is extracted from the primary input of the original specification machine $\mathbf{M}$ and appropriately encoded by the input encoder block Ψ. The output decoder block Θ combines the output information of the partial machines (product of $\pi_{S\times I}^i$) and translates it to the output of the specification machine $\mathbf{M}$.

In this context, the conditions of the General Decomposition Theorem can be interpreted as follows. The condition (1) demands that for any partial state machine, its next state has to be unambiguously defined using only partial state/input information available to the machine. The condition (2) requires that this information is also sufficient to compute the output of the partial machine. To guarantee that the imported information can be produced in some partial machine, condition (3) demands that any imported

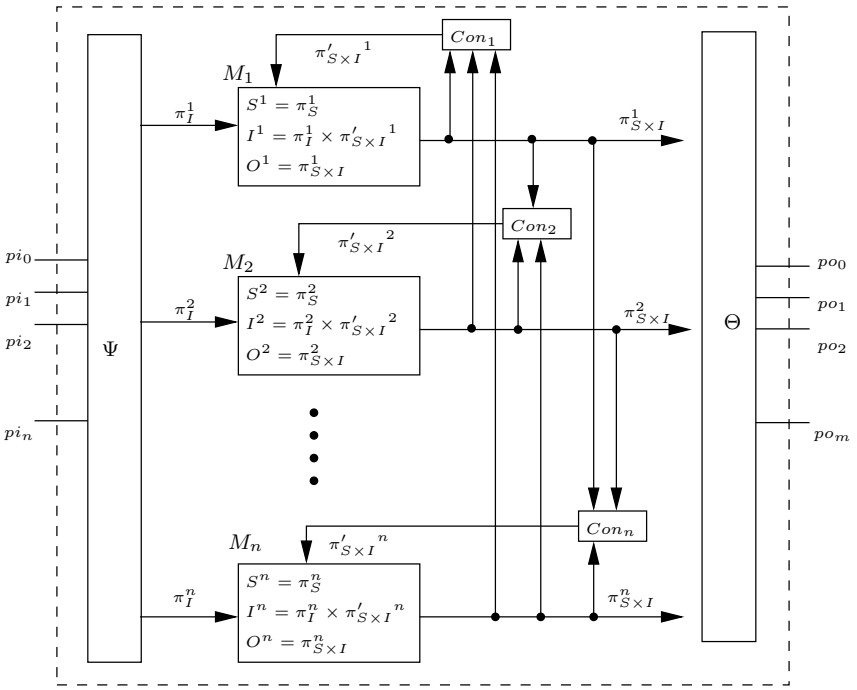


Figure 3.3. Scheme of a general decomposition of $\mathbf{M}$ into n partial machines $\mathbf{M}_i$

$PI_1\ PI_2$	PS	NS	$PO_1\ PO_2$
00	s_1	s_1	00
01	s_1	s_4	11
10	s_1	s_4	11
11	s_1	s_3	11
00	s_2	s_1	11
01	s_2	s_4	01
10	s_2	s_4	01
11	s_2	s_3	11
00	s_3	s_2	00
01	s_3	s_4	11
10	s_3	s_4	11
11	s_3	s_3	11
00	s_4	s_1	00
01	s_4	s_4	01
10	s_4	s_4	01
11	s_4	s_3	11
00	s_5	s_5	00
01	s_5	s_4	11
10	s_5	s_4	11
11	s_5	s_3	11
00	s_6	s_5	11
01	s_6	s_4	01
10	s_6	s_4	01
11	s_6	s_3	11

Figure 3.4. STT of the specification machine $\mathbf{M}$

information has to be less than the whole state/input information available in the network. It eliminates the possibility of combinational loops and the usage of information not present in the realization network. The realization of the correct output function is guaranteed by the condition (4) that demands that the combined output of the partial machines has to provide sufficient information to compute the primary output of the realized machine. Finally, condition (5) requires that, for state behavior realization, the combined current state of the partial machines unambiguously identifies the current state of the realized machine.

For more detailed explanation and example of the general decomposition structure see the following example.

3.1.2 Example

Let us consider a completely specified machine $\mathbf{M} = (I, S, O, \delta, \lambda)$ given in Fig. 3.4. $\mathbf{M}$ has input alphabet $I = \{00, 01, 10, 11\}$, state alphabet $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ and output alphabet $O = \{00, 01, 10, 11\}$.

To explain Theorem 3.1 and its use for the construction of decompositional realization structures of sequential machines, let us consider the decomposition of machine $\mathbf{M}$ defined by two trinities of partitions: $(\pi_I^1, \pi_S^1, \pi_{S\times I}^1)$ and $(\pi_I^2, \pi_S^2, \pi_{S\times I}^2)$, where:

$$
\begin{aligned}
\pi_I^1 &= \{\overline{00};\overline{01,10,11}\} = \{i_1, i_2\} \\
\pi_S^1 &= \{\overline{s_1, s_2};\overline{s_3, s_4};\overline{s_5, s_6}\} = \{a, b, c\} \\
\pi_{S\times I}^1 &= \{\overline{(s_1,00)(s_2,00)(s_5,00)(s_6,00)};\overline{(s_3,00)(s_4,00)}; \\
&\quad \overline{(s_1,01)(s_1,10)(s_1,11)(s_2,01)(s_2,10)(s_2,11)(s_3,01)(s_3,10)(s_3,11)}\ldots \\
&\quad \ldots\overline{(s_4,01)(s_4,10)(s_4,11)(s_5,01)(s_5,10)(s_5,11)(s_6,01)(s_6,10)(s_6,11)}\} = \\
&= \{o_1, o_2, o_3\} \\
\pi_I^2 &= \{\overline{00,11};\overline{01,10}\} = \{j_1, j_2\} \\
\pi_S^2 &= \{\overline{s_1, s_3, s_5};\overline{s_2, s_4, s_6}\} = \{x, y\} \\
\pi_{S\times I}^2 &= \{\overline{(s_1,00)(s_1,01)(s_1,10)(s_1,11)(s_2,11)(s_3,00)(s_3,01)(s_3,10)(s_3,11)}\ldots \\
&\quad \ldots\overline{(s_4,00)(s_4,11)(s_5,00)(s_5,01)(s_5,10)(s_5,11)(s_6,11)};\\
&\quad \overline{(s_2,00)(s_2,01)(s_2,10)(s_4,01)(s_4,10)(s_6,00)(s_6,01)(s_6,10)}\} \\
&= \{p_1, p_2\}.
\end{aligned}
$$

First, we will show that the two trinities $(\pi_I^1, \pi_S^1, \pi_{S\times I}^1)$ and $(\pi_I^2, \pi_S^2, \pi_{S\times I}^2)$ satisfy the conditions of Theorem 3.1 and therefore they define a decomposition of $\mathbf{M}$ with two partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$, each based on a corresponding trinity. We will also demonstrate how to construct the decompositional realization structure (see Fig. 3.3) for $\mathbf{M}$ based on these two trinities. The structure is constructed in four steps. Firstly, we build the input encoder block Ψ, which transforms information delivered by primary inputs into a form suitable for partial machines. Secondly, state transition tables (STT) of partial machines $\mathbf{M}_i$ are constructed. Then, the network of connections between the partial machines is determined (Con_i functions). Finally, we construct the output decoder block Θ, which transforms the output information of the partial machines into the primary output of the specification machine.

Machine $\mathbf{M}$ computes its next state and output from information about its present state S and input I. Thus, it computes two functions $\delta : S \times I \to S$ and $\lambda : S \times I \to O$. For its computations $\mathbf{M}$ uses complete information about the $S \times I$ space. Partial machines, on the other hand, use a partial information on the points in $S \times I$ space to compute their own next state and output. The abovementioned partition describe this partial information.

As discussed in Section 2.4, a partition models information stream that delivers information allowing to distinguish symbols placed in different blocks of the partition. The partition describing information delivered by a given variable can be derived by putting in one block all the symbols for which the value of the variable is the same. Thus, the partition has as many blocks as there are values of the variable. Applying this method in reverse, given a partition modeling given information stream, one can construct a variable delivering the information modeled by the partition by creating a variable with as many values as there are blocks in the partition and associating a single value of the variable with all the symbols placed in a single block of the partition. For example, the partial information about states modeled by π_S^1 can be delivered by a three-valued variable, which assumes first value (e.g. a) for states s_1 and s_2, second value (b) for states s_3 and s_4 and the third value (c) for states s_5 and s_6.

Exploring the correspondence between partitions and variables, the input, state and

output variables of a partial machine $\mathbf{M}_i$ can be constructed using partitions π_I^i, π_S^i and $\pi_{S\times I}^i$, correspondingly.

Construction of the input encoder Ψ

The construction of the input encoder is determined by the primary input information required by the partial machines. This information is defined by the input partitions π_I^1 and π_I^2. For example, π_I^1 implies that the first partial machine $\mathbf{M}_1$ only needs information about the primary input that allows one to distinguish input combination 00 from all the other combinations. To supply this information, the input encoder Ψ needs to compute values of a two-valued variable associated with π_I^1, which will assume its first value (e.g. i_1) when the input is 00, and the second value (i_2) for all the other input combinations. The complete function table of the input encoder Ψ is given in Figure 3.5.

PI_1	PI_2	π_I^1	π_I^2
0	0	i_1	j_1
0	1	i_2	j_2
1	0	i_2	j_2
1	1	i_2	j_1

Figure 3.5. Function table of input encoder Ψ

Construction of partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$

Information available to partial machines. The partial information about the state and input ($S\times I$) of the specification machine $\mathbf{M}$ available to a partial machine $\mathbf{M}_i$ is delivered by a combination of three information sources: information about $S \times I$ derived from its own state information (given by partition ${\pi_{S\times I}^S}^i$ on $S \times I$ induced by π_S^i), information about $S \times I$ derived from its own primary input information (given by partition ${\pi_{S\times I}^I}^i$ on $S \times I$ induced by π_I^i) and information imported from other partial machines (${\pi'_{S\times I}}^i$).

The induced partitions can easily be determined according to Def. 2.21 from input and state partitions π_I^i and π_S^i. For example, the block of ${\pi_{S\times I}^S}^1$ induced by the first block $\overline{(s_1, s_2)}$of π_S^1 will include all state/input combinations involving states s_1 and s_2, i.e. $\overline{(s_1,00)(s_1,01)(s_1,10)(s_1,11)(s_2,00)(s_2,01)(s_2,10)(s_2,11)}$. This fact has a simple interpretation: knowing *only* that the specification machine $\mathbf{M}$ is in either state s_1 or s_2, we are not able to distinguish whether the current state/input situation is $(s_1, 00)$, $(s_1, 01)\ldots$ or $(s_2, 11)$. The induced partitions therefore are the following (the symbols in parenthesis describe the values inducing a particular block) :

$$
\begin{aligned}
{\pi_{S\times I}^S}^1 \quad = \quad & \{\overline{(s_1,00)(s_1,01)(s_1,10)(s_1,11)(s_2,00)(s_2,01)(s_2,10)(s_2,11)}(s_1,s_2);\\
& \overline{(s_3,00)(s_3,01)(s_3,10)(s_3,11)(s_4,00)(s_4,01)(s_4,10)(s_4,11)}(s_3,s_4);\\
& \overline{(s_5,00)(s_5,01)(s_5,10)(s_5,11)(s_6,00)(s_6,01)(s_6,10)(s_6,11)}(s_5,s_6)\}
\end{aligned}
$$

$$
\begin{aligned}
{\pi^S_{S\times I}}^2 \;=\; \{&\overline{(s_1,00)(s_1,01)(s_1,10)(s_1,11)(s_3,00)(s_3,01)(s_3,10)(s_3,11)}\ldots\\
&\ldots\overline{(s_5,00)(s_5,01)(s_5,10)(s_5,11)}(s_1,s_3,s_5);\\
&\overline{(s_2,00)(s_2,01)(s_2,10)(s_2,11)(s_4,00)(s_4,01)(s_4,10)(s_4,11)}\ldots\\
&\ldots\overline{(s_6,00)(s_6,01)(s_6,10)(s_6,11)}(s_2,s_4,s_6)\}
\end{aligned}
$$

$$
\begin{aligned}
{\pi^I_{S\times I}}^1 \;=\; \{&\overline{(s_1,00)(s_2,00)(s_3,00)(s_4,00)(s_5,00)(s_6,00)}(00);\\
&\overline{(s_1,01)(s_1,10)(s_1,11)(s_2,01)(s_2,10)(s_2,11)(s_3,01)(s_3,10)(s_3,11)}\ldots\\
&\ldots\overline{(s_4,01)(s_4,10)(s_4,11)(s_5,01)(s_5,10)(s_5,11)(s_6,01)(s_6,10)(s_6,11)}\\
&(01,10,11)\}
\end{aligned}
$$

$$
\begin{aligned}
{\pi^I_{S\times I}}^2 \;=\; \{&\overline{(s_1,00)(s_1,11)(s_2,00)(s_2,11)(s_3,00)(s_3,11)(s_4,00)(s_4,11)}\ldots\\
&\ldots\overline{(s_5,00)(s_5,11)(s_6,00)(s_6,11)}(00,11);\\
&\overline{(s_1,01)(s_1,10)(s_2,01)(s_2,10)(s_3,01)(s_3,10)(s_4,01)(s_4,10)}\ldots\\
&\ldots\overline{(s_5,01)(s_5,10)(s_6,01)(s_6,10)}(01,10)\}.
\end{aligned}
$$

The theorem does not prescribe the method to derive the partitions describing the information imported by particular partial machines from some other partial machines (${\pi'_{S\times I}}^i$). It only puts limits on these partitions. The upper bound of the information that can be imported by a particular machine $\mathbf{M}_i$ is all the information available for importing, i.e. all information produced by the other machines (${\pi'_{S\times I}}^i = \prod_j \pi^{ji}_{S\times I}$, where $\pi^{ji}_{S\times I}$ describes the fraction of the output information of $\mathbf{M}_j$ that is imported by $\mathbf{M}_i$, and $\pi^{j}_{S\times I} \leq \pi^{ji}_{S\times I}$). The lower bound is imposed by conditions (1) and (2) of the theorem. The imported information has to be sufficiently large for these conditions to be satisfied. Between these two limits any imported partition will allow a construction of a valid decomposition. It is left as a design choice how much information will actually be imported and in which form (the choice of ${\pi'_{S\times I}}^i$) and from which machines this information will originate (the choice of $\pi^{ji}_{S\times I}$).

The only additional limit on the imported information is given by condition (3). It demands that no information can be imported by a partial machine that cannot be originally derived from the total primary input or state information (described by $\pi^I_{S\times I}$ and $\pi^S_{S\times I}$, respectively). This condition prevents the creation of combinational loops due to interconnections between the partial machines.

In this example, it can be verified that the lower bound on the information imported by $\mathbf{M}_1$ is zero information, while the lower bound and upper bound on the information imported by $\mathbf{M}_2$ is the entire output information of $\mathbf{M}_1$. Therefore,

$$
\begin{aligned}
{\pi'_{S\times I}}^1 &= \pi^{21}_{S\times I} = \pi_{S\times I}(1)\\
{\pi'_{S\times I}}^2 &= \pi^{12}_{S\times I} = \pi^1_{S\times I}
\end{aligned}
$$

As discussed in Section 2.4, we use one-partition $\pi_{S\times I}(1)$ to indicate the lack of information. Both partitions ${\pi'_{S\times I}}^i$ satisfy condition (3), with the partition $\pi^I_{S\times I} * \pi^S_{S\times I}$ describing

the global state and input information:

$$\begin{aligned}\pi^I_{S\times I} * \pi^S_{S\times I} &= \{\overline{(s_1,00)};\overline{(s_2,00)};\overline{(s_3,00)};\overline{(s_4,00)};\overline{(s_5,00)};\overline{(s_6,00)};\overline{(s_1,11)};\\ &\quad \overline{(s_2,11)};\overline{(s_1,01)(s_1,10)};\overline{(s_2,01)(s_2,10)};\overline{(s_3,11)};\overline{(s_4,11)};\\ &\quad \overline{(s_3,01)(s_3,10)};\overline{(s_4,01)(s_4,10)};\overline{(s_5,11)};\overline{(s_6,11)};\overline{(s_5,01)(s_5,10)};\\ &\quad \overline{(s_6,01)(s_6,10)}\}\end{aligned}$$

Using the above results, we can derive the partitions ${\pi^{in}_{S\times I}}^1$ describing the combined information about the state and input of the specification machine available to a partial machine $\mathbf{M}_i$.

$$\begin{aligned}{\pi^{in}_{S\times I}}^1 &= {\pi^S_{S\times I}}^1 \cdot {\pi^I_{S\times I}}^1 \cdot {\pi'_{S\times I}}^1 =\\ &= \{\overline{(s_1,00)(s_2,00)};\overline{(s_3,00)(s_4,00)};\overline{(s_5,00)(s_6,00)};\\ &\quad \overline{(s_1,01)(s_1,10)(s_1,11)(s_2,01)(s_2,10)(s_2,11)};\\ &\quad \overline{(s_3,01)(s_3,10)(s_3,11)(s_4,01)(s_4,10)(s_4,11)};\\ &\quad \overline{(s_5,01)(s_5,10)(s_5,11)(s_6,01)(s_6,10)(s_6,11)}\}\\ {\pi^{in}_{S\times I}}^2 &= {\pi^S_{S\times I}}^2 \cdot {\pi^I_{S\times I}}^2 \cdot {\pi'_{S\times I}}^2 =\\ &= \{\overline{(s_1,00)(s_5,00)};\overline{(s_3,00)};\overline{(s_1,11)(s_3,11)(s_5,11)};\overline{(s_2,00)(s_6,00)};\\ &\quad \overline{(s_4,00)};\overline{(s_2,11)(s_4,11)(s_6,11)};\\ &\quad \overline{(s_1,01)(s_1,10)(s_3,01)(s_3,10)(s_5,01)(s_5,10)};\\ &\quad \overline{(s_2,01)(s_2,10)(s_4,01)(s_4,10)(s_6,01)(s_6,10)}\}\end{aligned}$$

Construction of $\mathbf{M}_1$. In the same fashion as the partition π^1_I defines the primary input alphabet of $\mathbf{M}_1$ to be $\{i_1, i_2\}$ (see the input encoder construction), partitions π^1_S and $\pi^1_{S\times I}$ determine the state and output alphabet of $\mathbf{M}_1$, respectively. Thus, $\mathbf{M}_1$ has three states defined by the three blocks of the state partition π^1_S. $\mathbf{M}_1$ is in the state a corresponding to the first block of the state partition π^1_S whenever the specification machine is in the state s_1 or s_2. Similarly, state b of $\mathbf{M}_1$ is associated with the states s_3 and s_4 of the specification machine $\mathbf{M}$ and c with s_5 and s_6. Partition $\pi^1_{S\times I}$ defines a ternary output variable, which for example assumes value o_2 defined by block $\overline{(s_3,00)(s_4,00)}$ when the specification machine $\mathbf{M}$ sees 00 on its input and is in either state s_3 or s_4.

The conditions of the General Decomposition Theorem guarantee that for the partial machine $\mathbf{M}_1$ defined in this way it is possible to build STT consistent with the specification machine. We will show how these conditions are verified and use the results of the verification to construct the STT.

The total information available to a partial machine $\mathbf{M}_i$ is described by ${\pi^{in}_{S\times I}}^i$. Each block of this partition therefore defines a single indivisible input situation recognized by the partial machine. It therefore corresponds to a single transition of $\mathbf{M}_i$. For example, block $\overline{(s_3,00)(s_4,00)}$ of ${\pi^{in}_{S\times I}}^1$ corresponds to the situation in which the specification machine is in state s_3 or s_4 and the input is 00. In these conditions the state variable of $\mathbf{M}_1$ assumes the value associated by π^1_S with states s_3 and s_4 (b), while the primary input variable assumes the value associated by π^1_I with the input 00 (i_1). It means that $\mathbf{M}_1$ is in state b and its input is i_1. In this situation, the behavior of $\mathbf{M}_1$ is determined by the requirement of consistency with the specification machine $\mathbf{M}$. For $\mathbf{M}_1$ to behave in a man-

ner consistent with the specification machine, it has to transit to the state corresponding to the subset of the next states of the specification machine in these conditions. The subset of the next states of the specification machine is determined by $\overline{\delta}(\overline{(s_3,00)(s_4,00)})$ = $\{\delta(s_3,00);\delta(s_4,00)\}$ = $\{s_1,s_2\}$. This means that in these input/state conditions the specification machine transits to either s_1 or s_2. However, since $\mathbf{M}_1$ does not recognize s_1 from s_2 anyway (s_1 and s_2 are placed in the same block of π_S^1), $\mathbf{M}_1$ unambiguously performs the transition to its state associated with the block $\overline{s_1,s_2}$ (a) of π_S^1. Note that in this way we have shown that for block $\overline{(s_3,00)(s_4,00)}$ of ${\pi_{S\times I}^{in}}^1$ there is a block of π_S^1 including $\overline{\delta}(\overline{(s_3,00)(s_4,00)})$, and thus verified condition (1) of the theorem for the block $\overline{(s_3,00)(s_4,00)}$. At the same time, we have determined that the next state of $\mathbf{M}_1$ in state b under input i_1 is a. To determine the output value of $\mathbf{M}_1$ we need to find a block of $\pi_{S\times I}^1$ unambiguously indicated by the current input/state conditions ($(s_3,00)(s_4,00)$). In this case it is the second block ($\overline{(s_3,00)(s_4,00)}$) associated with the output value o_2. As a result, we have determined the block of $\pi_{S\times I}^1$ including the considered block of ${\pi_{S\times I}^{in}}^1$, which verifies the condition (2) for ($(s_3,00)(s_4,00)$) and determines the output of $\mathbf{M}_1$ to be o_2.

Thus, we have constructed one transition of $\mathbf{M}_1$ (i_1,b,a,o_2), i.e. transition 3 in Fig. 3.6. Following the same procedure, conditions (1) and (2) can be verified for the remaining blocks of ${\pi_{S\times I}^{in}}^1$ and the other transitions of $\mathbf{M}_1$ can be determined. The complete STT of $\mathbf{M}_1$ is given in Fig. 3.6.

Construction of $\mathbf{M}_2$. In the case of $\mathbf{M}_1$ we were able to neglect the information imported from the partial machines, since $\mathbf{M}_1$ does not import any information (${\pi'_{S\times I}}^1 = \pi_{S\times I}(1)$). $\mathbf{M}_2$ on the other hand imports the information from $\mathbf{M}_1$ carried by a three-valued output variable of $\mathbf{M}_1$ described by ${\pi'_{S\times I}}^2$. The input alphabet of $\mathbf{M}_2$ therefore becomes a combination of symbols delivered by the input encoding block Ψ and imported from machine $\mathbf{M}_1$. It is described by the Cartesian product $\{j_1,j_2\}\times\{j'_1,j'_2,j'_3\}$, where j'_1,j'_2,j'_3 are the values of the imported variable. Since $\mathbf{M}_2$ directly imports the output of $\mathbf{M}_1$ without any encoding, $\{j'_1,j'_2,j'_3\}=\{o_1,o_2,o_3\}$ and the input alphabet of $\mathbf{M}_2$ becomes $\{j_1,j_2\}\times\{o_1,o_2,o_3\}$.

The state alphabet of $\mathbf{M}_2$ is determined by the blocks of partition π_S^2. Therefore, $\mathbf{M}_2$ has two states: x corresponding to the states s_1, s_3 and s_5 of the specification machine;

in (π_I^1)	ps (π_S^1)	ns (π_S^1)	out ($\pi_{S\times I}^1$)
i_1	a	a	o_1
i_2	a	b	o_3
i_1	b	a	o_2
i_2	b	b	o_3
i_1	c	c	o_1
i_2	c	b	o_3

Figure 3.6. STT of $\mathbf{M}_1$

in (π_I^2)	imp (${\pi'_{S\times I}}^2$)	ps (π_S^2)	ns (π_S^2)	out ($\pi_{S\times I}^2$)
j_1	o_1	x	x	p_1
j_1	o_2	x	y	p_1
j_1	o_3	x	x	p_1
j_2	$-$	x	y	p_1
j_1	o_1	y	x	p_2
j_1	o_2	y	x	p_1
j_1	o_3	y	x	p_1
j_2	$-$	y	y	p_2

Figure 3.7. STT of $\mathbf{M}_2$

and y corresponding to the states s_2, s_4 and s_6.

To explain the transition determination for $\mathbf{M}_2$, let us consider the block $\overline{(s_1,00)(s_5,00)}$ of ${\pi^{in}_{S\times I}}^2$. It corresponds to the state s_1 or s_5 of the specification machine with the primary input 00. In this situation, $\mathbf{M}_2$ is in state x (s_1 and s_5 are included in block $\overline{s_1,s_3,s_5}$ of π^2_S associated with state x), the primary-input-component of its input is j_1 (00 is included in the block $\overline{00,11}$ of π^2_I associated with the value j_1) and the imported component of the input is o_1 ($\overline{(s_1,00)(s_5,00)}$ is a subset of the first block of ${\pi'_{S\times I}}^2$ associated with the value o_1).

The next states of the specification machine in $(s_1,00)$ or $(s_5,00)$ are s_5 or s_1, which are included in the first block ($\overline{s_1,s_3,s_5}$) of π^2_S, which corresponds to the state x of $\mathbf{M}_2$. The output of $\mathbf{M}_2$ is determined by the block of $\pi^2_{S\times I}$ including $\overline{(s_1,00)(s_5,00)}$, i.e. the first block of $\pi^2_{S\times I}$ associated with the value p_1. Thus, we have verified conditions (1) and (2) for the block $\overline{(s_1,00)(s_5,00)}$ and constructed transition $((j_1,o_1),x,x,p_1)$, i.e. the first transition in Fig. 3.7. The verification of the conditions for the remaining blocks of ${\pi^{in}_{S\times I}}^2$ results in STT of $\mathbf{M}_2$, presented in Fig. 3.7.

Interconnection network.

The interconnection network is responsible for the exchange of information between the partial machines. The functions implemented within the interconnection network (Con_i) provide the mapping between the combined output of the partial machines and the imported input of a particular partial machine $\mathbf{M}_i$. In this case, $\mathbf{M}_1$ does not import any information, so there's no connection from $\mathbf{M}_2$ to $\mathbf{M}_1$, while $\mathbf{M}_2$ imports the entire output information of $\mathbf{M}_1$, so function Con_2 is an identity function (a multi-valued "wire" from $\mathbf{M}_1$ output to $\mathbf{M}_2$ input).

Construction of the output decoder Θ.

The task of the output decoder is to determine the primary output value of the specification machine $\mathbf{M}$ based on the partial information about state/input delivered by the output variables of the partial machines $\mathbf{M}_i$. This partial information is a combination of the output information of the partial machines $\mathbf{M}_i$ and it is described by the partition $\pi_{S\times I}$.

$$\begin{aligned}\pi_{S\times I} \;=\; & \pi^1_{S\times I}\cdot\pi^2_{S\times I} = \{\overline{(s_1,00)(s_5,00)}(o_1,p_1);\overline{(s_2,00)(s_6,00)}(o_1,p_2);\\ & \overline{(s_3,00)(s_4,00)}(o_2,p_1);\\ & \overline{(s_1,01)(s_1,10)(s_1,11)(s_2,11)(s_3,01)(s_3,10)(s_3,11)(s_4,11)}\ldots\\ & \ldots\overline{(s_5,01)(s_5,10)(s_5,11)(s_6,11)}(o_3,p_1);\\ & \overline{(s_2,01)(s_2,10)(s_4,01)(s_4,10)(s_6,01)(s_6,10)}(o_3,p_2)\}\end{aligned}$$

The interpretation of this partition is that when the machine $\mathbf{M}_1$ produces output o_1 and the machine $\mathbf{M}_2$ produces output p_1, the decoder "knows" that the specification machine $\mathbf{M}$ is either in state s_1 or s_5 and the input of the network is 00. However, since the decoder cannot distinguish between these two situations, the output of the specification machine for $(s_1,00)$ and $(s_5,00)$ has to be the same. This is the sense of

condition (4). Consulting the STT of the specification machine $\mathbf{M}$, we determine that the output for both $(s_1, 00)$ and $(s_5, 00)$ is indeed the same, i.e. 00, and therefore condition (4) is satisfied for the first block of $\pi_{S\times I}$. At the same time we determine that the output of the output decoder is 00 for the output o_1 of $\mathbf{M}_1$ and p_1 of $\mathbf{M}_2$, and thus also the primary output of $\mathbf{M}$ is 00. The consideration of the remaining blocks of $\pi_{S\times I}$ leads to the output decoder function presented in Fig. 3.8.

$\pi^1_{S\times I}$	$\pi^2_{S\times I}$	PO_1	PO_2
o_1	p_1	0	0
o_1	p_2	1	1
o_2	p_1	0	0
o_3	p_1	1	1
o_3	p_2	0	1

Figure 3.8. Output decoder function

State behavior.

Finally, the product of the state partitions $\pi^1_S \cdot \pi^2_S$ of the partial machines is $\pi_S(0)$, which means that the combination of the states of the partial machines always indicates only a single state of the specification machine or is empty. Thus, in accordance with condition (5), the state behavior of the specification machine $\mathbf{M}$ is also realized.

Resulting network.

All of the above information is sufficient to construct the decomposition network composed of the partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$, input encoder Ψ, output decoder Θ and interconnection Con_2, which realizes the state and output behavior of the original machine $\mathbf{M}$. This network is depicted in Fig. 3.9.

The combined behavior of the network of partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$ is described by a composition machine in Fig. 3.10. Let us consider, for example, the situation when $\mathbf{M}_1$ is in state a, $\mathbf{M}_2$ is in state x and the output of the input encoder Ψ is (i_1, j_1). This corresponds to the situation when the specification machine is in state s_1 (the intersection of the block $\overline{s_1, s_2}$ corresponding to the state a of $\mathbf{M}_1$ and block $\overline{s_1, s_3, s_5}$ corresponding to the state x of $\mathbf{M}_2$) with the input 00 (see the input encoder construction). This situation triggers the transition (i_1, a, a, o_1) in $\mathbf{M}_1$. Since the output of $\mathbf{M}_1$ is o_1 in $\mathbf{M}_2$ the transition $((j_1, o_1), x, x, p_1)$ is triggered. The resulting next state of the network therefore is (a, x) (which translates to the state s_1 of the specification machine) and the output is (o_1, p_1)(which is decoded to 00 by the output decoder block). Comparing this with Fig. 3.4 we can verify that this is consistent with the specification machine $\mathbf{M}$.

Output-only behavior realization.

It can be verified that all conditions except (5) of Theorem 3.1 are still satisfied when π^1_S in the trinity $(\pi^1_I, \pi^1_S, \pi^1_{S\times I})$ is replaced by $\{\overline{s_1, s_2, s_5, s_6}(a'); \overline{s_3, s_4}(b)\}$, without any changes to the other partitions in the two trinities. It means that with the modified π^1_S it is possible to realize the output behavior of the specification machine without realizing

its state behavior. Note that this change is interpreted as merging the states of $a(\overline{s_1, s_2})$ and $c(\overline{s_5, s_6})$ of the original $\mathbf{M}_1$ into a new state a', so it is equivalent to implicit state minimization. Indeed, upon inspection of STT for $\mathbf{M}_1$ it can be observed that the states a and c are equivalent.

The partitions affected by the change of ϕ_S^1 are

$$\begin{aligned}\pi_{S\times I}^{S\ 1} \;=\; & \{\overline{(s_1,00)(s_1,01)(s_1,10)(s_1,11)(s_2,00)(s_2,01)(s_2,10)(s_2,11)}\ldots\\ & \ldots\overline{(s_5,00)(s_5,01)(s_5,10)(s_5,11)(s_6,00)(s_6,01)(s_6,10)(s_6,11)}\\ & (s_1,s_2,s_5,s_6);\\ & \overline{(s_3,00)(s_3,01)(s_3,10)(s_3,11)(s_4,00)(s_4,01)(s_4,10)(s_4,11)}(s_3,s_4)\}\end{aligned}$$

$$\begin{aligned}\pi_{S\times I}^{in\ 1} \;=\; & \{\overline{(s_1,00)(s_2,00)(s_5,00)(s_6,00)};\\ & \overline{(s_1,01)(s_1,10)(s_1,11)(s_2,01)(s_2,10)(s_2,11)(s_5,01)(s_5,10)(s_5,11)}\ldots\\ & \ldots\overline{(s_6,01)(s_6,10)(s_6,11)};\\ & \overline{(s_3,00)(s_4,00)};\overline{(s_3,01)(s_3,10)(s_3,11)(s_4,01)(s_4,10)(s_4,11)}\}\end{aligned}$$

$$\begin{aligned}\pi_{S\times I}^{S} * \pi_{S\times I}^{I} \;=\; & \{\overline{(s_1,00)(s_5,00)};\overline{(s_1,11)(s_5,11)};\\ & \overline{(s_1,01)(s_1,10)(s_5,01)(s_5,10)};\overline{(s_3,00)};\overline{(s_3,11)};\overline{(s_3,01)(s_3,10)};\\ & \overline{(s_2,00)(s_6,00)};\overline{(s_2,11)(s_6,11)};\overline{(s_2,01)(s_2,10)(s_6,01)(s_6,10)};\\ & \overline{(s_4,00)};\overline{(s_4,11)};\overline{(s_4,01)(s_4,10)}\}\end{aligned}$$

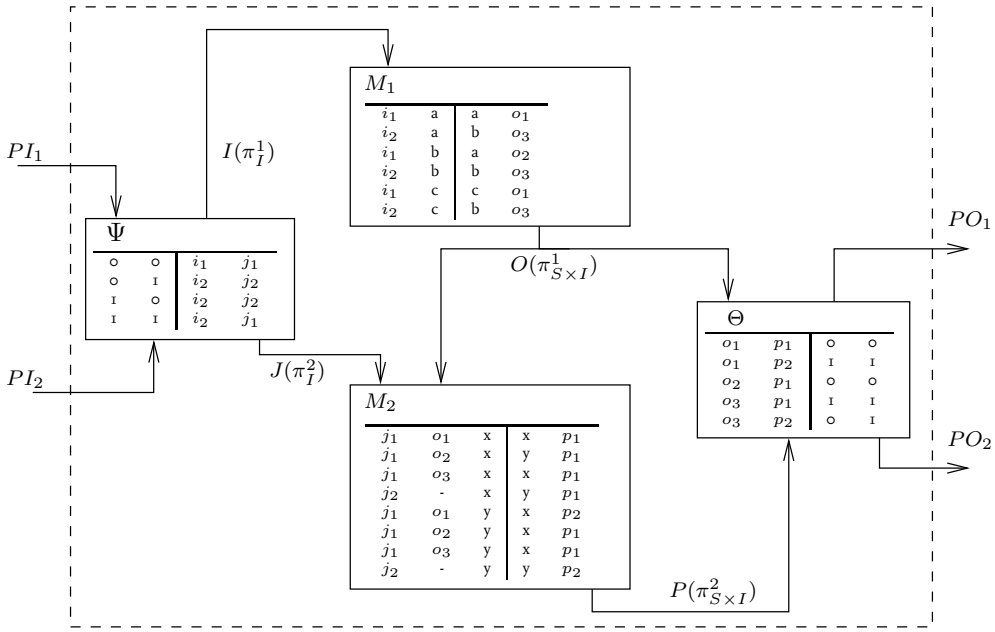


Figure 3.9. Scheme of the example decomposition

in	ps	ns	out
(00) i_1j_1	ax (s_1)	ax (s_1)	o_1p_1 (00)
(01) i_2j_2	ax (s_1)	by (s_4)	o_3p_1 (11)
(10) i_2j_2	ax (s_1)	by (s_4)	o_3p_1 (11)
(11) i_2j_1	ax (s_1)	bx (s_3)	o_3p_1 (11)
(00) i_1j_1	ay (s_2)	ax (s_1)	o_1p_2 (11)
(01) i_2j_2	ay (s_2)	by (s_4)	o_3p_2 (01)
(10) i_2j_2	ay (s_2)	by (s_4)	o_3p_2 (01)
(11) i_2j_1	ay (s_2)	bx (s_3)	o_3p_1 (11)
(00) i_1j_1	bx (s_3)	ay (s_2)	o_2p_1 (00)
(01) i_2j_2	bx (s_3)	by (s_4)	o_3p_1 (11)
(10) i_2j_2	bx (s_3)	by (s_4)	o_3p_1 (11)
(11) i_2j_1	bx (s_3)	bx (s_3)	o_3p_1 (11)
(00) i_1j_1	by (s_4)	ax (s_1)	o_2p_1 (00)
(01) i_2j_2	by (s_4)	by (s_4)	o_3p_2 (01)
(10) i_2j_2	by (s_4)	by (s_4)	o_3p_2 (01)
(11) i_2j_1	by (s_4)	bx (s_3)	o_3p_1 (11)
(00) i_1j_1	cx (s_5)	cx (s_5)	o_1p_1 (00)
(01) i_2j_2	cx (s_5)	by (s_4)	o_3p_1 (11)
(10) i_2j_2	cx (s_5)	by (s_4)	o_3p_1 (11)
(11) i_2j_1	cx (s_5)	bx (s_3)	o_3p_1 (11)
(00) i_1j_1	cy (s_6)	cx (s_5)	o_1p_2 (11)
(01) i_2j_2	cy (s_6)	by (s_4)	o_3p_2 (01)
(10) i_2j_2	cy (s_6)	by (s_4)	o_3p_2 (01)
(11) i_2j_1	cy (s_6)	bx (s_3)	o_3p_1 (11)

Figure 3.10. STT of the composition machine

Considering the modified input partition ${\pi^{in}_{S\times I}}^1$ of $\mathbf{M}_1$, we see that in the first block of ${\pi^{in}_{S\times I}}^1$ being $\overline{(s_1,00)(s_2,00)(s_5,00)(s_6,00)}$ the next state for symbols $\overline{(s_1,00)}$ and $\overline{(s_2,00)}$ is s_1 , while for $\overline{(s_5,00)}$ and $\overline{(s_6,00)}$ it is s_5 . However, π^1_S has states s_1 and s_5 in one block (a'), so the block $\overline{(s_1,00)(s_2,00)(s_5,00)(s_6,00)}$ still unambiguously identifies the next state of $\mathbf{M}_1$ to be a' . The remaining blocks of ${\pi^{in}_{S\times I}}^1$ lead to the states b, a' and b , respectively. Thus, condition (1) remains satisfied for ${\pi^{in}_{S\times I}}^1$. Furthermore, ${\pi^{in}_{S\times I}}^1$ is still smaller-or-equal-to $\pi^i_{S\times I}$, so condition (2) is also satisfied. Since ${\pi'_{S\times I}}^1$ = $\pi_{S\times I}(1)$, and $\pi_{S\times I}$ remains unchanged, conditions (3) and (4) are not affected by the state minimization. Finally, the product state partition $\pi_S = \{\overline{s_1,s_5};\overline{s_2,s_6};\overline{s_3};\overline{s_4}\}$, is not a zero-partition, so the state behavior of $\mathbf{M}$ is not realized, as the combination of the partial machines does not recognize state s_1 from s_5 and s_2 from s_6.

3.2 Decomposition of incompletely specified FSM

3.2.1 Extensions of GDT

What is central to the General Decomposition Theorem (GDT) for completely specified FSMs is the concept of a partition pair, which reflects the possibility to compute some partial information from another partial information, i.e. compute information modeled by one partition using information modeled by some other partition. Let us consider how this concept changes with the introduction of nondeterminism, output don't-cares and multi-state realizations.

Partitions are no longer sufficient to express don't-care conditions for state and output. For this reason, they need to be replaced with more general unique-block covers (simply called *covers* in the following).

Nondeterminism

The definition of $S \times I - S$ partition pair uses block transition function $\overline{\delta}(D)$, which calculates a set of next states (combined next state) for a subset D of $S \times I$. In the case of a nondeterministic machine each element from $S \times I$ leads not to a single next state, but rather to a *choice* of next states. Conceptually, an element of $S \times I$ can lead to any subset of its next-states-set. To establish the possible sets of next states for D, we need to consider a combination of all subsets of all the next-state-sets for all $S \times I$ elements in D. Such a combination is performed by the operator $\overline{\bigcup}$.

Definition 3.7 (Operator $\overline{\bigcup}$). $\overline{\bigcup} : 2^{S^n} \rightarrow 2^{2^S}$ *and*

$$\overline{\bigcup}([D_i]_{i=1\ldots n}) = \{B | \exists C^1 \subseteq D_1, C^1 \neq \emptyset \ldots \exists C^n \subseteq D_n, C^n \neq \emptyset \ : \ B = \bigcup_{i=1\ldots n} C^n\}$$

Operator $\overline{\bigcup}$ accepts a vector of subsets of S. It returns all the possible combinations of sums of the non-empty subsets of S, provided that each subset is taken from another vector position.

Ex. 3.2.1.

$$\overline{\bigcup}\left(\left[\begin{array}{c}\{1,2,3\}\\ \{1,4\}\end{array}\right]\right) = \{\{1\},\{1,2\},\{1,3\},\{1,4\},\{2,4\},\{3,4\},\{1,2,3\},\{1,2,4\},\{2,3,4\},\{1,2,3,4\}\}$$

For the example of the application of operator $\overline{\bigcup}$ to the block transition function see also Example 3.2.3. ∎

Definition 3.8 (Operator $\overline{\bigcap}$). $\overline{\bigcap} : 2^{2^{S^n}} \rightarrow 2^{2^S}$ *and*

$$\overline{\bigcap}([D_i]_{i=1\ldots n}) = \{B | \exists C^1 \in D_1 \ldots \exists C^n \in D_n \ : \ B = \bigcap_{i=1\ldots n} C^n\}$$

Operator $\overline{\bigcap}$ accepts a vector of sets of subsets of S. It returns all the possible combinations of intersections of the subsets of S, provided that each intersected subset is taken from another vector position.

Ex. 3.2.2. *For* $S = \{1 \dots 5\}$

$$\overline{\bigcap}\left(\begin{bmatrix} \{\{1\}\{1,2\}\{2,3\}\{4\}\} \\ \{\{1,2,3\}\{4,5\}\} \\ \{\{1,2,4,5\}\} \end{bmatrix}\right) = \{\{1\},\{1,2\},\{2\},\{4\}\}$$

■

Block transition function $\overline{\delta}$ now becomes

$$\overline{\delta} : 2^{S\times I} \to 2^S \quad \text{and} \quad \overline{\delta}(D) = \overline{\bigcup_{d\in D}} \delta(d)$$

To adjust the partition pair concept to this situation, we need to use covers and ensure that at least one of the possible next-state-sets for D is included in a block of ϕ_S^i cover, i.e. there is a choice of the next states for each state/input combination in D such that the resulting next-state-sets lead to a block of ϕ_S^i.

For this purpose we introduce a "cover transition function", which returns those next state blocks of $\overline{\delta}$ that are included in a block of ϕ_S .

Definition 3.9 (Cover transition function).

$$\Delta^{\phi_S} : 2^{S\times I} \to 2^{\phi_S} \quad \textit{and} \quad \Delta^{\phi_S}(D) = \{B' \in \phi_S | \exists B'' \in \overline{\delta}(D) : B'' \subseteq B'\}$$

The $S \times I - S$ cover pair condition can then be formulated as

$$\forall D \in \phi_{S\times I} : \Delta^{\phi_S}(D) \neq \emptyset$$

Note that if the machines are deterministic, this condition is exactly equivalent to the partition pair condition.

in	ps	ns	out
0	a	b, c	$00, 11$
1	a	a	01
0	b	a, c	11
1	b	b	01
0	c	a	$10, 01$
1	c	b	$10, 01, 11$

Figure 3.11. Incompletely specified sequential machine

Ex. 3.2.3. *For machine in Fig. 3.11 and* $\phi_{S\times I} = \{\overline{(a,0)(b,0)}; \overline{(a,1)(b,1)(c,0)(c,1)}\} = \{D_1; D_2\}$

$$\overline{\delta}(D_1) = \overline{\bigcup}\begin{bmatrix} \delta(a,0) \\ \delta(b,0) \end{bmatrix} = \overline{\bigcup}\begin{bmatrix} \{b,c\} \\ \{a,c\} \end{bmatrix} = \{\{a,b\};\{a,c\};\{b,c\};\{c\};\{a,b,c\}\}$$

$$\overline{\delta}(D_2) = \overline{\bigcup}\begin{bmatrix} \delta(a,1) \\ \delta(b,1) \\ \delta(c,0) \\ \delta(c,1) \end{bmatrix} = \overline{\bigcup}\begin{bmatrix} \{a\} \\ \{b\} \\ \{a\} \\ \{b\} \end{bmatrix} = \{\{a,b\}\}$$

Therefore, for $\phi_S = \{\overline{a,b,c}; \overline{b,c}\}$, $\Delta^{\phi_S}(D_1) = \{\overline{a,b,c}; \overline{b,c}\} \neq \emptyset$ *and* $\Delta^{\phi_S}(D_2) = \{\overline{a,b,c}\} \neq \emptyset$*, which means that* $\phi_{S\times I}$ *and* ϕ_S *form a* $S \times I - S$ *pair.* ■

Furthermore, due to the fact that there is a *choice* of next states for the composition machine, it is not sufficient for a partial machine to move to a state consistent with *any* of the possible next states of $\mathbf{M}$. *All* partial machines need to move to their respective states consistent with *the same* subsets of the $\mathbf{M}$'s next states. Otherwise, the product of the next states of partial machines is empty and the state of $\mathbf{M}$ is unspecified.

This leads to the concept of a synchronized set of cover pairs. A set of cover pairs is synchronized if and only if each of the pairs is a cover pair, and each of the blocks of the first covers leads to such blocks of the second covers that the intersection of the blocks of the first covers leads to the intersection of the blocks of the second covers. Note that synchronized sets of cover pairs is a stronger requirement than the cover pair of the product of the first covers with the product of the second covers.

Definition 3.10 (Synchronized set of $S \times I - S$ cover pairs). *$([\phi^i_{S\times I}]_i, [\phi^i_S]_i)$ is a synchronized $S \times I - S$ cover pair if and only if*

1 . $\forall i \forall D_i \in \phi^i_{S\times I} \Delta^{\phi^i_S}(D_i) \neq \emptyset$ *(cover pair condition)*

2 . $\forall i \forall D^i \in \phi^i_{S\times I} \exists B'_{D^i} \in \Delta^{\phi^i_S}(D^i) : \forall [D^i]_i \in \times_i \phi^i_{S\times I} : \bigcap_i D^i \neq \emptyset \Rightarrow$

$\Rightarrow \forall B'_D \in \overline{\bigcap}_i B'_{D^i} \exists B' \in \Delta^{\prod_i \phi^i_{S_l}}(\bigcap_i D^i) : B' = B'_D$ *(synchronization condition)*

Taking all this into account, the condition (1) of GDT can be rewritten as

$\left([\phi^{in}_{S\times I}{}^i]_i, [\phi^i_S]_i\right)$ is a synchronized set of $S \times I - S$ cover pairs

For an example of synchronized set of cover pairs see Section 3.4.1.

Output don't-cares

Similarly, the consideration of output don't cares requires the adjustment of a block output function $\overline{\lambda}$ and introduction of a cover output function Λ^{ϕ_O} .

Definition 3.11 (Block output function).

$$\overline{\lambda} : 2^{S\times I} \to 2^O \quad \textit{and} \quad \overline{\lambda}(D) = \overline{\bigcup_{d\in D} \lambda(d)}$$

$\overline{\lambda}$ computes a set of all possible combinations of choices between values of output for elements of D.

Definition 3.12 (Cover output function).

$$\Lambda^{\phi_O} : 2^{S\times I} \to 2^{\phi_O} \quad \textit{and} \quad \Lambda^{\phi_O}(D) = \left\{C \in \phi_O \mid \exists C' \in \overline{\lambda}(D) : C' \subseteq C\right\}$$

Λ^{ϕ_O} returns those blocks of ϕ_O that can be calculated as outputs for input combinations in D, allowing for don't-care assignment.

Definition 3.13 ($S \times I - O$ cover pair). *$(\phi_{S\times I}, \phi_O)$ is a $S \times I - O$ cover pair if and only if*

$$\forall D \in \phi_{S\times I} : \Lambda^{\phi_O}(D) \neq \emptyset$$

Ex. 3.2.4. *For machine and $\phi_{S\times I}$ from Example 3.2.3*

$$\begin{aligned}
\overline{\lambda}(D_1) &= \overline{\bigcup}\begin{bmatrix}\lambda(a,0)\\ \lambda(b,0)\end{bmatrix} = \overline{\bigcup}\begin{bmatrix}\{00,11\}\\ \{11\}\end{bmatrix} = \{\{00,11\};\{11\}\}\\
\overline{\lambda}(D_2) &= \overline{\bigcup}\begin{bmatrix}\lambda(a,1)\\ \lambda(b,1)\\ \lambda(c,0)\\ \lambda(c,1)\end{bmatrix} = \overline{\bigcup}\begin{bmatrix}\{01\}\\ \{01\}\\ \{10,01\}\\ \{10,01,11\}\end{bmatrix} =\\
&= \{\{01\};\{10,01\};\{01,11\};\{10,01,11\}\}
\end{aligned}$$

Therefore, for $\phi_O = \{\overline{01};\overline{11};\overline{00,10}\}$, $\Lambda^{\phi_O}(D_1) = \{\overline{11}\} \neq \emptyset$ and $\Lambda^{\phi_O}(D_2) = \{\overline{01}\} \neq \emptyset$, which means that $\phi_{S\times I}$ and ϕ_O form a $S \times I - O$ pair. ■

Since output don't cares can prevent some output symbols from being produced, the cover specialization concept needs to be introduced to allow for the reduction of the output alphabet.

Definition 3.14 (Cover specialization). *Specialization of cover ϕ_S is a cover $\phi_{S'}$ such that*

$$\begin{aligned}
& S' \subseteq S\\
\wedge\ & \forall A \in \phi_S\ \exists B \in \phi_{S'} : B \subseteq A\\
\wedge\ & \forall B \in \phi_{S'}\ \exists A \in \phi_S : A \supseteq B
\end{aligned}$$

Cover is specialized by removing some elements from its blocks.

With this in mind, condition (4) of the GDT becomes:

ϕ_O has specialization $\phi_{O'}$ and $\left(\phi_{S\times I}, \phi_{O'}\right)$ is a $S \times I - O'$ cover pair .

Multi-state realization

Definition 3.15 (Multi-state realization). *Incompletely specified machine $\mathbf{M}_l(I_l, S_l, O_l, \delta_l, \lambda_l)$ is a multi-state realization of i.s. machine $\mathbf{M}(I, S, O, \delta, \lambda)$ if and only if*

1. $I_l = I$
2. $O_l = O$
3. *there is an injective* copy-assignment function $L : S \to 2^{S_l}$*, which establishes correspondence between states in $\mathbf{M}$ and their copies in $\mathbf{M}_l$*
4. *δ_l is a* multi-state realization *of δ, i.e.*
   $\forall s \in S\ \forall x \in I\ \forall s_l \in L(s) \quad \delta(s,x) \supseteq L'(\delta_l(s_l,x))$
5. *λ_l is a* multi-state realization *of λ, i.e.*
   $\forall s \in S\ \forall x \in I\ \forall s_l \in L(s) \quad \lambda(s,x) \supseteq \lambda_l(s_l,x)$

where $L' : 2^{S_l} \to 2^S$ is inverse of L defined as $L'(A_l) = \{s \in S | L(s) \cap A_l \neq \emptyset\}$

in	ps	ns	out
0	$a1$	b,$c1$,$c2$	00
1	$a1$	$a2$	01
0	$a2$	b,$c2$	00,11
1	$a2$	$a1$	01
0	b	$a1$,c	11
1	b	b	01
0	$c1$	$a1$,$a2$	10,01
1	$c1$	b	10,01,11
0	$c2$	$a2$	10
1	$c2$	b	10,11

Figure 3.12. Multi-state realization of machine in Fig. 3.11

Ex. 3.2.5. *Consider the state transition table in Fig. 3.12. It was created by splitting states a and c of the machine in Fig. 3.11 into its two copies: $a1, a2$ and $c1, c2$, respectively. It can be observed that outputs of the copies of the states are subsets of the outputs of their originals (under the same input) and that the next-states of the copies of the states are some copies of the next-states of the originals.* ∎

The possibility of multi-state realization means that there are multiple states of the network of partial machines (i.e. of the composition machine) corresponding to a single state in the specification machine. We treat these multiple states as the "labeled copies" of a particular state of the specification machine. Note that while we introduce this concept for incompletely specified FSMs, it is also valid for completely specified machines as a special case — their decomposition may also have multiple states corresponding to a single state of the original machine.

The most immediate consequence for the conditions of GDT is the expansion of the machine's state set from S to S_l, and input space from $S \times I$ to $S_l \times I$.

This introduces additional freedom to the composition machine. For the behavior of the network to be consistent with the behavior of the specification machine, it is sufficient for copies of a certain state to transit to *some* copies of its next state, and produce a compatible output under the same input. This freedom can be exploited to optimize the FSM implementation or to resolve some implementation problems (e.g. race-free assignment for asynchronous level-mode circuits using multi-state assignment called "multiple-row assignment"). We recall that the partition (or cover) pair concept corresponds to the ability of partial machines to perform transitions under the given input conditions (a block of input cover $\phi^{in}_{S\times I}{}^{i}$) to their next states (blocks of state cover ϕ^i_S) in a manner consistent with the behavior of the specification machine. For this consistency to be preserved in the case of labeled covers $\phi^i_{S_l\times I}$ and $\phi^i_{S_l}$ it is sufficient if *unlabeled* blocks of $\phi^i_{S_l\times I}$ transit to *unlabeled* blocks of $\phi^i_{S_l}$. This can be expressed by an adequate reformulation of the cover transition and cover output functions to correctly operate on covers on labeled copies of the states.

Definition 3.16 (Labeled cover transition function).

$$\Delta^{\phi_{S_l}} : 2^{S_l\times I} \to 2^{\phi_{S_l}} \quad \textit{and} \quad \Delta^{\phi_{S_l}}(D) = \{B'_l \in \phi_{S_l} | \exists B' \in \overline{\delta}(L'(D)) : B' \subseteq L'(B'_l)\}$$

Definition 3.17 (Labeled cover output function).

$$\Lambda^{\phi_O} : 2^{S_l \times I} \rightarrow 2^{\phi_O} \quad \textit{and} \quad \Lambda^{\phi_O}(D) = \left\{C | C \in \phi_O \ \wedge \ \exists C' \in \overline{\lambda}(L'(D)) : C' \subseteq C\right\}$$

As a consequence, cover pairs for the multi-state realizations become the following:

Definition 3.18 ($S_l \times I - S_l$ cover pair). *$(\phi_{S_l \times I}, \phi_{S_l})$ is a $S_l \times I - S_l$ cover pair if and only if*

$$\forall D \in \phi_{S_l \times I} : \Delta^{\phi_{S_l}}(D) \neq \emptyset$$

Definition 3.19 ($S_l \times I - O$ cover pair). *$(\phi_{S_l \times I}, \phi_O)$ is a $S_l \times I - O$ cover pair if and only if*

$$\forall D \in \phi_{S_l \times I} : \Lambda^{\phi_O}(D) \neq \emptyset$$

Ex. 3.2.6. *Consider machine* $\mathbf{M}$ *in Fig. 3.11 with state a split (labeled) into two copies $a1$ and $a2$ and covers $\phi_{S_l \times I} = \{\overline{(a1,1)(c,0)}; \overline{(a2,1)(c,0)(b,0)}; \overline{(a1,0)(a2,0)(b,1)(c,1)}\} = \{D_1; D_2; D_3\}$ and $\phi_{S_l} = \{\overline{a1,c}; \overline{a2}; \overline{b,c}\}$. The unlabeled version of D_1 is $L'(D_1) = \overline{(a,1)(c,0)}$. For these two input combinations M moves to*

$$\overline{\delta}(L'(D_1)) = \overline{\bigcup} \begin{bmatrix} \delta(a,1) \\ \delta(c,0) \end{bmatrix} = \overline{\bigcup} \begin{bmatrix} \{a\} \\ \{a\} \end{bmatrix} = \{\{a\}\}$$

After labeling, block $\{a\}$ can fit into either $\overline{a1,c}$ or $\overline{a2}$ of ϕ_{S_l}, so $\Delta^{\phi_{S_l}}(D_1) = \{\overline{a1,c}; \overline{a2}\}$. For D_2 and D_3: $L'(D_2) = \overline{(a,1)(b,0)(c,0)}$ and $L'(D_3) = \overline{(a,0)(b,1)(c,1)}$.

$$\begin{aligned} \overline{\delta}(L'(D_2)) &= \overline{\bigcup} \begin{bmatrix} \delta(a,1) \\ \delta(b,0) \\ \delta(c,0) \end{bmatrix} = \overline{\bigcup} \begin{bmatrix} \{a\} \\ \{a,c\} \\ \{a\} \end{bmatrix} = \{\{a\}; \{a,c\}\} \quad \text{and} \\ \Delta^{\phi_{S_l}}(D_2) &= \{\overline{a1,c}; \overline{a2}\}. \end{aligned}$$

$$\begin{aligned} \overline{\delta}(L'(D_3)) &= \overline{\bigcup} \begin{bmatrix} \delta(a,0) \\ \delta(b,1) \\ \delta(c,1) \end{bmatrix} = \overline{\bigcup} \begin{bmatrix} \{b,c\} \\ \{b\} \\ \{b\} \end{bmatrix} = \{\{b\}; \{b,c\}; \} \quad \text{and} \\ \Delta^{\phi_{S_l}}(D_3) &= \{\overline{b,c}\} \end{aligned}$$

Therefore, $\phi_{S_l \times I}$ and ϕ_{S_l} form a $S_l \times I - S_l$ pair. ∎

Finally, the fact that there are multiple instances of the same state necessitates the introduction of the $S_l - S$ cover pair. It expresses the fact that blocks of ϕ_{S_l} only include the copies of the states that are included in the same blocks of ϕ_S.

Definition 3.20 ($S_l - S$ cover pair). *(ϕ_{S_l}, ϕ_S) is a $S_l - S$ cover pair if and only if*

$$\forall D \in \phi_{S_l} \ \exists B \in \phi_S \quad L'(D) \subseteq B$$

3.2.2 General Decomposition Theorem

Using the above discussed extensions necessary to account for nondeterminism (next-state don't-cares), incompletely specified output function (output don't-cares) and multi-state realizations, the General Decomposition Theorem for the multi-state realizations of incompletely specified nondeterministic FSMs can be formulated as follows:

Let ϕ_O and ϕ_S be covers on $\mathbf{M} = (I, S, O, \delta, \lambda)$ on O and S, respectively. Let $\phi^i_I, \phi^i_{S_l}, \phi^i_{S_l \times I}$ be covers on multi-state specialization $\mathbf{M}_l = (I, S_l, O, \delta_l, \lambda_l)$ of machine $\mathbf{M}$, on I,S_l and $S_l \times I$, respectively. Let cover ϕ_O be a partition defined by the following expression:

$$[o]\phi_O = [p]\phi_O \quad \Longleftrightarrow \quad \forall s \in S \forall x \in I : o \in \lambda(s,x) \iff p \in \lambda(s,x) \tag{3.2.1}$$

(ϕ_O groups in one block the output symbols, which are not distinguished by *any* input combination).

Let $\phi^{ij}_{S_l \times I}$ be covers of $S_l \times I$ such that $\phi^{ij}_{S_l \times I} \geq \phi^i_{S_l \times I}$ and ${\phi'_{S_l \times I}}^{j} = \prod_{i=1\ldots n} \phi^{ij}_{S_l \times I}$. Let $\phi_{S_l \times I} = \prod_{i=1\ldots n} \phi^i_{S_l \times I}$. Let ${\phi^I_{S_l \times I}}^{i}$ and ${\phi^{S_l}_{S_l \times I}}^{i}$ be covers induced on $S_l \times I$ by ϕ^i_I and $\phi^i_{S_l}$, respectively. Let $\phi^I_{S_l \times I} = \prod_{i=1\ldots n} {\phi^I_{S_l \times I}}^{i}$ and $\phi^{S_l}_{S_l \times I} = \prod_{i=1\ldots n} {\phi^{S_l}_{S_l \times I}}^{i}$. Let ${\phi^{in}_{S_l \times I}}^{i} = {\phi^I_{S_l \times I}}^{i} \cdot {\phi^{S_l}_{S_l \times I}}^{i} \cdot {\phi'_{S_l \times I}}^{i}$.

Theorem 3.2 (General full-decomposition of incompletely specified FSM). *A sequential machine $\mathbf{M} = (I, S, O, \delta, \lambda)$ has a general full-decomposition with the output behavior realization with n component machines if and only if n input covers ϕ^i_I, n state covers ϕ^i_S, and n label assignments $\mathcal{L}^i : \phi^i_S \to \phi^i_{S_l}$ exist such that n trinities of covers $(\phi^i_I, \phi^i_{S_l}, {\phi_{S_l \times I}}^{i})$ exist that satisfy the following conditions:*

(1) $\left([{\phi^{in}_{S_l \times I}}^{i}]_i, [\phi^i_{S_l}]_i\right)$ *is an $S_l \times I - S_l$ synchronized cover set pair*

(2) ${\phi^{in}_{S_l \times I}}^{i} \leq {\phi_{S_l \times I}}^{i}$

(3) $\phi^I_{S_l \times I} \cdot \phi^{S_l}_{S_l \times I} \leq {\phi'_{S_l \times I}}^{i}$

(4) ϕ_O *has specialization* $\phi_{O'}$ *and* $\left(\phi_{S_l \times I}, \phi_{O'}\right)$ *is a $S_l \times I - O'$ cover pair*

Additionally, if

(5) $\left(\prod_i \phi^i_{S_l}, \phi_S(0)\right)$ *is an $S_l - S$ cover pair*

then the state behavior of $\mathbf{M}$ will be realized too.

3.3 Proof of the General Decomposition Theorem

3.3.1 Forward proof

For more clarity, we will conduct the forward part of the proof in two steps. First, we will introduce a machine $\mathbf{M}^*$ that is built based on the covers mentioned in the Theorem and represents the combined behavior of the partial machines. We will show that, if the conditions of the theorem are fulfilled, $\mathbf{M}^*$ is a realization of the specification machine

$\mathbf{M}$. Then, we will show that a network of partial machines built based on the covers in the theorem is indeed equivalent to $\mathbf{M}^*$, and hence is a realization of $\mathbf{M}$.

In each of the steps, we will propose appropriate transition and output functions (δ and λ) for the discussed machines and show that under the conditions of the theorem they are well defined. Then, we will show that thus defined machines are realizations of the specification machine in the light of the definitions of the behavior realization (Defs. 2.8 and 2.9).

In the proof, we will use the following lemmas.

Lemma 3.3.1. $\forall S,T : \phi^1_S \leq \phi^2_S \quad \Rightarrow \quad ind^S_{S\times T}(\phi^1_S) \leq ind^S_{S\times T}(\phi^2_S)$

Proof.

$\phi^1_S \leq \phi^2_S \quad \Longleftrightarrow \quad \forall A^1 \in \phi^1_S \exists A^2 \in \phi^2_S : A^1 \subseteq A^2$
additionally, $A^1 \subseteq A^2 \Rightarrow A^1 \times T \subseteq A^2 \times T$ hence,
$\forall B^1 \in \{A^1 \times T | A^1 \in \phi^1_S\} \exists B^2 \in \{A^2 \times T | A^2 \in \phi^2_S\} : B^1 \subseteq B^2$
by definition of induced cover (Def. 2.21)
$\forall B^1 \in ind^S_{S\times T}(\phi^1_S) \exists B^2 \in ind^S_{S\times T}(\phi^2_S) : B^1 \subseteq B^2$
which means that $ind^S_{S\times T}(\phi^1_S) \leq ind^S_{S\times T}(\phi^2_S)$

Lemma 3.3.2.

$$\forall \phi_{S_l}\ \forall D_l \subseteq S_l \times I\ \forall (s_l,x) \in D_l \quad : \quad \Delta^{\phi_{S_l}}(D_l) \neq \emptyset \ \Rightarrow$$
$$\Rightarrow \quad \forall B'_l \in \Delta^{\phi_{S_l}}(D_l) \exists s'_l \in B'_l : s'_l \in L(\delta(L'(s_l),x))$$

(for each element (s_l,x) of D_l, each block of $\Delta^{\phi_{S_l}}(D_l)$ contains a copy of the specification machine's next state for the unlabeled version of (s_l,x))

Proof. By definition of $\Delta^{\phi_{S_l}}$ it only returns those blocks of ϕ_{S_l} which, after unlabeling, entirely include a block of $\overline{\delta}(L'(D_l))$ (i.e. contain at least copies of all elements of a block of $\overline{\delta}(L'(D_l))$). By definition of $\overline{\delta}$, each block of $\overline{\delta}(D)$ contains a next state for each element $d \in D$. Therefore, each block of $\overline{\delta}(L'(D_l))$ contains a next state for the unlabeled version of each element $d_l \in D_l$. Therefore, for each block returned by $\Delta^{\phi_{S_l}}$, it contains a copy of the next state for the unlabeled version of each element of D_l.

Lemma 3.3.3. *For the specialization $\phi_{O'}$ of ϕ_O defined by (3.2.1)*

$$\forall D \in S_l \times I\ \forall (s_l,x) \in D\ :\ \Lambda^{\phi_{O'}}(D)) \neq \emptyset\ \Rightarrow\ \forall Y \in \Lambda^{\phi_{O'}}(D) : Y \subseteq \lambda(L'(s_l),x)$$

(for each element of D, each block of $\Lambda^{\phi_{O'}}(D)$ is a subset of the output values for the unlabeled version of this element in the specification machine)

Proof. Each block of $\Lambda^{\phi_{O'}}(D)$ includes some output for any element of D. The blocks of ϕ_O, and hence ϕ'_O only contain output values not distinguished by *any* input combination (by definition of ϕ_O and specialization). Hence, any returned block of ϕ'_O only contains outputs that are common to all combinations in D.

Lemma 3.3.4.

$$\forall D^1 \subseteq S \times I\ \forall D^2 \subseteq S \times I : D^1 \subseteq D^2 \Rightarrow \Delta^{\phi_S}(D^1) \supseteq \Delta^{\phi_S}(D^2)$$

Proof.

by definition of $\overline{\delta}: D^1 \subseteq D^2 \Rightarrow \forall B^2 \in \overline{\delta}(D^2)\ \exists B^1 \in \overline{\delta}(D^1) : B^1 \subseteq B^2$
hence, $\forall B^2 \in \overline{\delta}(D^2)\ \forall B' \in \phi_S : (B^2 \subseteq B' \Rightarrow \exists B^1 \in \overline{\delta}(D^1) : B^1 \subseteq B')$
by the definition of $\Delta^{\phi_S}(D) \forall B' \in \Delta^{\phi_S}(D^2) : B' \in \Delta^{\phi_S}(D^1)$
and $\Delta^{\phi_S}(D^1) \supseteq \Delta^{\phi_S}(D^2)$

Composition machine

First, we will show that machine $\mathbf{M}^*$, defined by the trinity of covers $(\prod_i \phi_I^i, \prod_i \phi_{S_l}^i, \prod_i \phi_{S_l \times I}^i)$ that satisfy conditions of Theorem 3.2, is a realization of $\mathbf{M}$.

Let functions Φ^*, Ψ^* and Θ^* be defined as follows:

$$\Phi^* : 2^S \to 2^{\phi_{S_l}} \quad \text{and} \quad \Phi^*(\mathcal{S}) = \bigcup_{s_l \in L(\mathcal{S})} [s_l]\phi_{S_l} \tag{3.3.1}$$

Φ^* returns all the blocks of ϕ_{S_l} including copies of elements of $\mathcal{S}$.

$$\Psi^* : I \to 2^{\phi_I} \quad \text{and} \quad \Psi^*(x) = [x]\phi_I \tag{3.3.2}$$

Ψ^* returns all the blocks of ϕ_I including x.

$$\Theta^* : 2^{\phi_{S_l \times I}} \to 2^O \quad \text{and} \quad \Theta^*(\mathcal{D}) = \bigcup_{D \in \mathcal{D}} \Lambda^{\phi_{O'}}(D) \tag{3.3.3}$$

Let $\mathbf{M}^* = (\phi_I, \phi_{S_l}, \phi_{S_l \times I}, \delta^*, \lambda^*)$ be the machine describing the combined behavior of the composition of n component machines. Let:

$$\phi_I = \prod_i \phi_I^i \tag{3.3.4}$$

$$\phi_{S_l} = \prod_i \phi_{S_l}^i \tag{3.3.5}$$

$$\phi_{S_l \times I} = \prod_i \phi_{S_l \times I}^i \tag{3.3.6}$$

$$\delta^* : \phi_{S_l} \times \phi_I \to \phi_{S_l} \quad \text{and} \quad \delta^*(B, A) = \Delta^{\phi_{S_l}}(B \times A) \tag{3.3.7}$$

$$\lambda^* : \phi_{S_l} \times \phi_I \to \phi_{S_l \times I} \quad \text{and} \quad \lambda^*(B, A) = [B \times A]\phi_{S_l \times I} \tag{3.3.8}$$

Since $\phi_I = \prod_i \phi_I^i \leq \phi_I^i$, by Lemma 3.3.1, $ind^I_{S_l \times I}(\phi_I) \leq ind^I_{S_l \times I}(\phi_I^i)$ and $\forall A \in \phi_I \exists A^i \in {\phi^I_{S_l \times I}}^i : S_l \times A \subseteq A^i$. Similarly, $\phi_{S_l} = \prod_i \phi^i_{S_l} \leq \phi^i_{S_l}$, $ind^{S_l}_{S_l \times I}(\phi_{S_l}) \leq ind^{S_l}_{S_l \times I}(\phi^i_{S_l})$ and $\forall B \in \phi_{S_l} \exists B^i \in {\phi^{S_l}_{S_l \times I}}^i : B \times I \subseteq B^i$. Finally, $ind^{S_l}_{S_l \times I}(\phi_{S_l}) \cdot ind^I_{S_l \times I}(\phi_I) \leq \phi'_{S_l \times I}$ (by Condition (3)), so $\forall A \in \phi_I \forall B \in \phi_{S_l} \exists C^i \in \phi'_{S_l \times I} : B \times A \subseteq C^i$

From all of the above:

$$\begin{aligned}
&\forall A \in \phi_I \forall B \in \phi_{S_l}\ \forall i \\
&\exists D^i \in {\phi^{in}_{S_l \times I}}^i\ \exists A^i \in {\phi^I_{S_l \times I}}^i \exists B^i \in {\phi^{S_l}_{S_l \times I}}^i \exists C^i \in {\phi'_{S_l \times I}}^i : \\
&(A^i \supseteq S_l \times A) \wedge (B^i \supseteq B \times I) \wedge (C^i \supseteq B \times A) \wedge \\
&\wedge (D^i = A^i \cap B^i \cap C^i \supseteq B \times A)
\end{aligned} \tag{3.3.9}$$

Hence,

$$\forall A \in \phi_I \forall B \in \phi_{S_l} \exists [D^i]_i \in \times_i {\phi^{in}_{S_l \times I}}^i : B \times A \subseteq \bigcap_i D^i$$

Furthermore, condition (1 p.2) guarantees that

$$\forall [D^i]_i \in \times_i {\phi^{in}_{S_l \times I}}^i : \bigcap_i D^i \neq \emptyset \Rightarrow \Delta^{\prod_i \phi^i_{S_l}}(\bigcap_i D^i) \neq \emptyset \tag{3.3.10}$$

From Lemma 3.3.4,

$$B \times A \subseteq \bigcap_i D^i \Rightarrow \Delta^{\phi_{S_l}}(B \times A) \supseteq \Delta^{\prod_i \phi^i_{S_l}}(\bigcap_i D^i) \neq \emptyset$$

Hence,

$$\forall A \in \phi_I \forall B \in \phi_{S_l} \ : \ \Delta^{\phi_{S_l}}(B \times A) \neq \emptyset \Rightarrow \delta^*(B, A) \neq \emptyset$$

Similarly, from (3.3.9), condition (2), and the fact that $\phi_{S_l \times I} = \prod_i \phi^i_{S_l \times I}$, we conclude that $B \times A$ is included in some block of $\phi_{S_l \times I}$, so $\lambda^*(B, A) \neq \emptyset$.

We will now show that $\mathbf{M}^*$ is a realization of $\mathbf{M}$.

Let $\delta^*(\Phi^*(s), \Psi^*(x)) = \bigcup_{B \in \Phi^*(s), A \in \Psi^*(x)} \delta^*(B, A)$.

by definition of Φ^* : $\quad \forall B \in \Phi^*(s) \exists s_l \in L(s) : s_l \in B$

by definition of Ψ^* : $\quad \forall A \in \Psi^*(x) : x \in A$

hence, $\forall B \in \Phi^*(s) \forall A \in \Psi^*(x) \exists s_l \in L(s) : (s_l, x) \in B \times A$

by Lemma 3.3.2 $\quad \forall B'_l \in \Delta^{\phi_{S_l}}(B \times A) \exists s'_l \in B'_l : s'_l \in L(\delta(L'(s_l), x)) = L(\delta(s, x))$

from that and the definition of δ^* $\quad \forall B'_l \in \delta^*(B, A) \exists s'_l \in L(\delta(s, x)) : s'_l \in B'_l$

hence, $\delta^*(B, A) \subseteq \bigcup_{s'_l \in L(\delta(s,x))} [s'_l]\phi_{S_l} = \Phi^*(\delta(s, x))$

Finally,

$$\forall s \in S \forall x \in I : \delta^*(\Phi^*(s), \Psi^*(x)) \subseteq \Phi^*(\delta(s, x)) \tag{3.3.11}$$

Further,

$$\begin{aligned} \forall B \in \Phi^*(s) \, \forall A \in \Psi^*(x) \ : \ \Theta^*(\lambda^*(B, A)) &= \Theta^*\Big([B \times A]\phi_{S_l \times I}\Big) = \\ &= \bigcup_{D \in [B \times A]\phi_{S_l \times I}} \big(\Lambda^{\phi_{O'}}(D)\big) \end{aligned}$$

By Lemma 3.3.3:

$$\forall (s_l, x) \in D \, \forall Y \in \Lambda^{\phi_{O'}}(D) \ : \ Y \subseteq \lambda(L'(s_l), x) \tag{3.3.12}$$

By definition of Φ^*, $\exists s_l \in L(s) : (s_l, x) \in B \times A$,

$$\forall D \in [B \times A]\phi_{S_l \times I} \ : \ (s_l, x) \in D \tag{3.3.13}$$

From (3.3.12) and (3.3.13) it follows that

$$\forall D \in [B \times A]\phi_{S_l \times I}\ \forall Y \in \Lambda^{\phi_{O'}}(D)\ :\ Y \subseteq \lambda(L'(s_l), x) = \lambda(s, x)$$

Hence

$$\bigcup_{D \in [B \times A]\phi_{S_l \times I}} \Lambda^{\phi_{O'}}(D) \subseteq \lambda(s, x)$$

and

$$\Theta^*\Big(\lambda^*\big(\Phi^*(s_l), \Psi^*(x)\big)\Big) \subseteq \lambda(s, x) \tag{3.3.14}$$

From (3.3.11), (3.3.14) and definition 2.8 it follows that $\mathbf{M}^*$ is an output behavior realization of $\mathbf{M}$.

State behavior realization

We will show that under the additional condition (5) of the Theorem, the composition machine $\mathbf{M}^*$ is a state behavior realization (s.b.r.) of $\mathbf{M}$.

Mapping function:

$$\Phi'^* : 2^{\phi_{S_l}} \to 2^S \quad \text{and} \quad \Phi'^*(\mathcal{B}) = \bigcup_{B \in \mathcal{B}} L'(B)$$

Then

$$\forall B \in \phi_{S_l}\ \forall x \in I\ :\ \Phi'^*(\delta^*(B, \Psi^*(x))) = \Phi'^*(\delta^*(B, [x]\phi_I))$$

$$\forall A \in [x]\phi_I\ :\ \Phi'^*(\delta^*(B, A)) = \Phi'^*\Big(\Delta^{\phi_{S_l}}(B \times A)\Big)$$

From (5) it follows that

$$\forall B \in \phi_{S_l}\ \dot{\exists} s_B \in S\ :\ L'(B) = \{s_B\} \tag{3.3.15}$$

so,

$$\forall B \in \phi_{S_l}\ \forall s_l \in B \dot{\exists} s_B \in S\ :\ L'(B) = L'(s_l) = s_B \tag{3.3.16}$$

Hence,

$$\forall (s_l, x) \in B \times A : L'(s_l) = s_B$$

then, by Lemma 3.3.2

$$\forall B' \in \Delta^{\phi_{S_l}}(B \times A) \exists s'_l \in B' : s'_l \in L(\delta(L'(s_l), x)) = L(\delta(s_B, x))$$

hence, since $B' \in \phi_{S_l}$ and (3.3.16)

$$\forall B' \in \Delta^{\phi_{S_l}}(B \times A) \forall s'_l \in B' : s'_l \in L(\delta(s_B, x))$$

Finally,

$$\forall B' \in \Delta^{\phi_{S_l}}(B \times A) \ : \ L'(B') \subseteq \delta(s_B, x) = \delta(\Phi^{'*}(B), x)$$
$$\forall B \in \phi_{S_l} \forall x \in I \ : \ \bigcup_{A \in \Psi^*(x)} \ \bigcup_{B' \in \Delta^{\phi_{S_l}}(B \times A)} L'(B') \subseteq \delta(\Phi^{'*}(B), x)$$

and hence

$$\Phi^{'*}(\delta^*(B, \Psi^*(x))) \subseteq \delta(\Phi^{'*}(B), x) \tag{3.3.17}$$

Additionally, by Lemma 3.3.3

$$\begin{aligned} &\forall B \in \phi_{S_l} \ \forall x \in I \ \forall A \in \Psi^*(x) \ \forall D \in \lambda^*(B, A) \\ &\forall (s_l, x) \in D \ \forall Y \in \Lambda^{\phi_{O'}}(D) \ : \\ &Y \subseteq \lambda(L'(s_l), x) \overset{(3.3.16)}{=} \lambda(L'(B), x) \end{aligned}$$

hence,

$$\forall B \in \phi_{S_l} \forall x \in I \ : \ \bigcup_{D \in \lambda^*(B, \Psi^*(x))} \Lambda^{\phi_{O'}}(D) \subseteq \lambda(L'(B), x) = \lambda(\Phi^{'*}(B), x)$$

and

$$\Theta^*(\lambda^*(B, \Psi^*(x))) \subseteq \lambda(\Phi^{'*}(B), x) \tag{3.3.18}$$

By (3.3.17),(3.3.18) and definition 2.9, machine $\mathbf{M}^*$ therefore is a state behavior realization of $\mathbf{M}$. Since, as shown in the following section, the network of partial machines $\mathbf{M}_i$ *always* is a state realization of $\mathbf{M}^*$, we conclude that if condition (5) is met, the network of partial machines $\mathbf{M}_i$ is a state and output behavior realization of $\mathbf{M}$.

Component machines

Let $\mathbf{M}_i = \left(\phi_I^i \times {\phi'_{S_l \times I}}^i, \phi^i_{S_l}, \phi^i_{S_l \times I}, \delta^i, \lambda^i\right)$ be a component sequential machine for which the following conditions are satisfied:

1. $\left(\phi_I^i, \phi^i_{S_l}, \phi^i_{S_l \times I}\right)$ satisfy the conditions of Theorem 3.2
2. $$\begin{aligned} &\forall B \in \phi_{S_l} \ \forall A \in \phi_I \ \forall B^i \in [B]\phi^i_{S_l} \ \forall A^i \in [A]\phi_I^i \ \forall C^i \in [B \times A]{\phi'_{S_l \times I}}^i \ : \\ &\delta^i\left(B^i, (A^i, C^i)\right) = \Big\{ B'_i \mid B'_i \in \Delta^{\phi^i_{S_l}}\left(D^i\right) \ \wedge \\ &\wedge D^i \in [(B^i \times A^i) \cap C^i]{\phi^{in}_{S_l \times I}}^i \ \wedge \ \forall [D^j]_j \in \times_j {\phi^{in}_{S_l \times I}}^j \ : \\ &\bigcap_j D^j \neq \emptyset \Rightarrow \forall B' \in \overline{\bigcap}_j \delta^j(D^j) \exists B^* \in \delta^*(B, A) : B^* \supseteq B' \Big\} \end{aligned} \tag{3.3.19}$$
3. $$\lambda^i\left(B^i, (A^i, C^i)\right) = \left[\left(B^i \times A^i\right) \cap C^i\right] \phi^i_{S_l \times I} \tag{3.3.20}$$

The two conditions for the next state of component machine (B_i') may interpreted as follows:

- The first ($B_i' \in \Delta^{\phi^i_{S_l}}(\dots)$) is a *local* condition and ensures that the component machine may only perform a transition to its legal state (a block of $\phi^i_{S_l}$).
- The second is a *global* synchronization condition – it ensures that all the component machines make the choice between their next-states in such way that the resulting state of the network is a legal state (a block of ϕ_{S_l}).

Since $(B^i \times A^i) \cap C^i$ is included in a block of ${\phi^{in}_{S_l \times I}}^i$ (by definition of ${\phi^{in}_{S_l \times I}}^i$), $[(B^i \times A^i) \cap C^i]{\phi^{in}_{S_l \times I}}^i \neq \emptyset$. From condition (1.1) it follows that there is a block of $\phi^i_{S_l}$ that includes next-states of $D^i \in [(B^i \times A^i) \cap C^i]{\phi^{in}_{S_l \times I}}^i$. Hence, $\Delta^{\phi^i_{S_l}}(D^i) \neq \emptyset$. Also, there is a block of $\phi^i_{S_l \times I}$ containing $(B^i \times A^i) \cap C^i$ (from condition (2)).

Since $\delta^*(B, A) = \Delta^{\prod_i \phi^i_{S_l}}(B \times A)$, condition (1.2) guarantees that

$$\exists B_i' \in \Delta^{\phi^i_{S_l}}(D^i) \quad : \quad \forall [D^j]_j \in \times_j [B \times A]{\phi^{in}_{S_l \times I}}^j : \bigcap_j D^j \neq \emptyset \Rightarrow \forall B' \in \overline{\bigcap}_j \delta^j(D^j) \exists B^* \in \delta^*(B, A) : B^* = B'$$

hence,

$$\forall B \in \phi_{S_l}\ \forall A \in \phi_I\ \forall B^i \in [B]\phi^i_{S_l}\ \forall A^i \in [A]\phi^i_I\ \forall C^i \in [B \times A]{\phi'_{S_l \times I}}^i : \delta^i(B^i, (A^i, C^i)) \neq \emptyset \wedge \lambda^i(B^i, (A^i, C^i)) \neq \emptyset$$

The behavior of the net of component machines is described by the following functions:

$\delta_{GC} : \times_i \phi^i_{S_l} \times \times_i \phi^i_I \to \times_i \phi^i_{S_l}$ and

$$\delta_{GC}([B^i]_i, [A^i]_i) = \Big[\bigcup_{C^i \in [B \times A]{\phi'_{S_l \times I}}^i} \delta^i(B^i, (A^i, C^i)) \Big]_i$$

$\lambda_{GC} : \times_i \phi^i_{S_l} \times \times_i \phi^i_I \to \times_i \phi^i_{S_l \times I}$ and

$$\lambda_{GC}([B^i]_i, [A^i]_i) = \Big[\bigcup_{C^i \in [B \times A]{\phi'_{S_l \times I}}^i} \lambda^i(B^i, (A^i, C^i)) \Big]_i$$

Let the mapping functions be:

$$\Phi \quad : \quad \phi_{S_l} \to \times_i 2^{\phi^i_{S_l}} \quad \text{and} \quad \Phi(B) = \big[[B]\phi^i_{S_l}\big]_i \tag{3.3.21}$$

$$\Psi \quad : \quad \phi_I \to \times_i 2^{\phi^i_I} \quad \text{and} \quad \Psi(A) = \big[[A]\phi^i_I\big]_i \tag{3.3.22}$$

$$\Theta \quad : \quad \times_i 2^{\phi^i_{S_l \times I}} \to 2^{\phi_{S_l \times I}} \quad \text{and} \quad \Theta\Big(\big[\mathcal{D}^i\big]_i\Big) = \bigcup_{[D^i]_i \in \times_i \mathcal{D}^i} \big[\bigcap_i D^i\big]\phi_{S_l \times I} \tag{3.3.23}$$

$$\Phi' \quad : \quad \times_i \phi^i_{S_l} \to 2^{\phi_{S_l}} \quad \text{and} \quad \Phi'([B^i]_i) = \big[\bigcap_i B^i\big]\phi_{S_l} \tag{3.3.24}$$

From the above it follows that

$$\forall B \in \phi_{S_l}\ \forall [B^i]_i \in \times_i [B]\phi^i_{S_l}\ \forall A \in \phi_I \quad : \quad \Phi'\Big(\delta_{GC}\big([B^i]_i, \Psi(A)\big)\Big) = \\ = \Phi'\Big(\delta_{GC}\big([B^i]_i, [A]\phi^i_I\big)\Big)$$

and

$$\forall [A^i]_i \in \times_i [A]\phi^i_I : \delta_{GC}\big([B^i]_i, [A^i]_i\big) = \Big[\bigcup_{C^i \in [B\times A]\phi'^{\,i}_{S_l \times I}} \delta^i\big(B^i, (A^i, C^i)\big)\Big]_i$$

By definition of δ^i

$$\forall B' \in \overline{\bigcap}_i \delta^i(B^i, (A^i, C^i)) \exists B^* \in \delta^*(B, A) : B^* = B'$$

hence,

$$\forall B' \in \overline{\bigcap}_i [\bigcup_{C^i \in [B\times A]\phi'^{\,i}_{S_l \times I}} \delta^i(B^i, (A^i, C^i))] \exists B^* \in \delta^*(B, A) : B^* = B'$$

Since $\forall B^* \in \delta^* : B^* \in \phi_{S_l}$:

$$\forall B' \in \overline{\bigcap}_i [\bigcup_{C^i \in [B\times A]\phi'^{\,i}_{S_l \times I}} \delta^i(B^i, (A^i, C^i))] \exists B^* \in \delta^*(B, A) : [B']\phi_{S_l} = [B^*]\phi_{S_l} = B^*$$

Therefore,

$$\Phi'\Big(\delta_{GC}\big([B^i]_i, [A^i]_i\big)\Big) \subseteq \delta^*(B, A) = \delta^*(\Phi'([B^i]_i), A)$$

and

$$\forall B \in \phi_{S_l} \forall [B^i]_i \in \times_i [B]\phi^i_{S_l} \forall A \in \phi_I \ : \ \Phi'\Big(\delta_{GC}\big([B^i]_i, \Psi(A)\big)\Big) \subseteq \delta^*(\Phi'([B^i]_i), A) \tag{3.3.25}$$

Further,

$$\forall B \in \phi_{S_l}\ \forall [B^i]_i \in \times_i [B]\phi^i_{S_l}\ \forall A \in \phi_I \quad : \quad \Theta\Big(\lambda_{GC}\big([B^i]_i, \Psi(A)\big)\Big) \\ = \Theta\Big(\lambda_{GC}\big([B^i]_i, [A]\phi^i_I\big)\Big)$$

and

$$\forall [A^i]_i \in \times_i [A]\phi^i_I : \lambda_{GC}\big([B^i]_i, [A^i]_i\big) = \Big[\bigcup_{C^i \in [B\times A]\phi'^{\,i}_{S_l \times I}} \lambda^i\big(B^i, (A^i, C^i)\big)\Big]_i$$

Since $(B^i \times A^i) \cap C^i \supseteq B \times A$

$\forall [Y^i]_i \in \lambda_{GC}\big([B^i]_i, [A^i]_i\big) Y^i \supseteq B \times A$
therefore,
$\forall [Y^i]_i \in \lambda_{GC}\big([B^i]_i, [A^i]_i\big) \bigcap_i Y^i \supseteq B \times A$

and
$[\bigcap_i Y^i]\phi_{S_l \times I} \subseteq [B \times A]\phi_{S_l \times I} = \lambda^*(B, A)$

This leads to

$$\forall B \in \phi_{S_l} \forall [B^i]_i \in \times_i [B]\phi^i_{S_l} \forall A \in \phi_I \ : \ \Theta\Big(\lambda_{GC}\big([B^i]_i, \Psi(A)\big)\Big) \subseteq \lambda^*(\Phi'([B^i]_i), A) \tag{3.3.26}$$

From (3.3.25) and (3.3.26) it follows that the net of component machines is an unconditional state and output behavior realization of the composition machine $\mathbf{M}^*$. Therefore, it is an output behavior realization of $\mathbf{M}$ whenever $\mathbf{M}^*$ is (i.e. under conditions (1)-(4)) and state and output behavior realization, whenever $\mathbf{M}^*$ is (i.e. under conditions (1)-(5))

3.3.2 Reverse proof

Let the output behavior of $\mathbf{M}$ be realized by a general composition $\mathbf{M}^*$ of n machines $\mathbf{M}_i = (I_i^*, S_i, O_i, \delta^i, \lambda^i)$, where:

$$\begin{aligned} & I_i^* = I_i' \times I_i \\ & Con_i : \times 2^{O_j} \to I_i' \quad \text{is a surjective function} \end{aligned}$$

Let:

$$\begin{aligned} \Psi &: I \to \times I_i \quad \text{be a function} \\ \Phi &: S \to 2^{\times S_i} \quad \text{be a function} \\ \Theta &: \times 2^{O_i} \to 2^O \quad \text{be a surjective partial function} \end{aligned}$$

In any case, there is a multi-state specialization $\mathbf{M}_l$ of $\mathbf{M}$ such that $\mathbf{M}^*$ is an output behavior realization of $\mathbf{M}_l$ (in a trivial case, $\mathbf{M}_l \equiv \mathbf{M}$). Hence, it is possible to identify two functions : $\Phi_l : S \to 2^{S_l}$ and $\Phi^* : S_l \to \times S_i$ such that $\Phi(s) = \Phi^*(\Phi_l(s))$. Intuitively, the process of mapping of states of $\mathbf{M}$ to (sets of) states of $\mathbf{M}^*$ is divided in two parts: first, a state of $\mathbf{M}$ is mapped to a set of states of its multi-state specialization $\mathbf{M}_l$, and then a state of $\mathbf{M}_l$ is mapped to a state of composition machine $\mathbf{M}^*$. Again, in a trivial case, where $\mathbf{M}_l \equiv \mathbf{M}$, $S_l = S$, Φ_l is the identity function and $\Phi^* = \Phi$.

Then, the mappings Ψ and Φ^* introduce the following covers on I and S_l, respectively:

$$\begin{aligned} \phi^i_I &: [x]\phi^i_I \cap [z]\phi^i_I \neq \emptyset \iff x_i = z_i \\ &\quad \text{where } (x_1, \ldots, x_i, \ldots, x_n) = \Psi(x) \text{ and } (z_1, \ldots, z_i, \ldots, z_n) = \Psi(z) \\ \phi^i_{S_l} &: [s]\phi^i_{S_l} \cap [t]\phi^i_{S_l} \neq \emptyset \iff s_i = t_i \\ &\quad \text{where } (s_1, \ldots, s_i, \ldots, s_n) = \Phi^*(s) \text{ and } (t_1, \ldots, t_i, \ldots, t_n) = \Phi^*(t) \end{aligned}$$

If the composition of $\mathbf{M}_i$ is legal, then the output function λ^i of each component machine computes its values from (a part of) the original primary input and state information present in the composition machine (directly or imported). Therefore, λ^i can be considered as a function $\lambda^i : S_l \times I \to 2^{O_i}$. Thus, it introduces a cover $\phi^i_{S_l \times I}$ on $S_l \times I$ such, that

$$[(s_l, x)]\phi^i_{S_l \times I} \cap [(t_l, z)]\phi^i_{S_l \times I} \neq \emptyset \iff \lambda^i(s_l, x) \cap \lambda^i(t_l, z) \neq \emptyset$$

The values of λ^i (the elements of O_i) can be considered as the blocks of $\phi^i_{S_l \times I}$, or the names of the blocks of $\phi^i_{S_l \times I}$.

In a similar way, the connection functions Con_i introduce the covers ${\phi'_{S_l \times I}}^i =$ $= \prod_j \phi^{ji}_{S_l \times I}$.

Since $\lambda^i : S_i \times I_i \times I'_i \to 2^{O_i}$ is a function, values of λ^i (i.e. blocks of $\phi^i_{S_l \times I}$) must be computed from $\phi^i_{S_l}({\phi^{S_l}_{S_l \times I}}^i), \phi^i_I({\phi^I_{S_l \times I}}^i)$, and ${\phi'_{S_l \times I}}^i$. This is equivalent to condition (2). Similarly, since $\delta^i : S_i \times I_i \times I'_i \to 2^{S_i}$ is a function, the blocks of $\phi^i_{S_l}$ must be computed from $\phi^i_{S_l}({\phi^{S_l}_{S_l \times I}}^i), \phi^i_I({\phi^I_{S_l \times I}}^i)$, and ${\phi'_{S_l \times I}}^i$. This requirement is stated in condition (1 p.1).

The condition (1 p.1), however, only guarantees that for any state/input combination for each partial machine $\mathbf{M}_i$ there *is* a valid next-state that the partial machine can transit to. The fact that the partial machines transit to their valid states does not guarantee that the composition machine does. If the intersection of the current states of partial machines (i.e. intersection of blocks of $\phi^i_{S_l}$) is empty, then the present state of the composition machine (i.e. a block of ϕ_{S_l}) is undefined. To avoid such a situation, it must be guaranteed that for any state/input combination there is a choice of next-states in the partial machines such that the intersection of the chosen next-states results in a valid state of the composition machine. This synchronization requirement is fulfilled by the condition (1 p.2).

If the composition does not contain combinational loops, and thus is legal, then values of each connection function, and so the values of $\phi'_{S_l \times I}$ must only be computed (directly or indirectly) from the original primary input and state information present in the composition. Since the total primary input information in the composition is defined by $\phi^I_{S_l \times I}$ and the total state information by $\phi^{S_l}_{S_l \times I}$, condition (3) must be satisfied.

The output information produced by all the component machines and described by $\phi_{S_l \times I}$ enables the computation of the output information of the original machine $\mathbf{M}$. Let us consider the original output function $\lambda : S \times I \to 2^O$ of $\mathbf{M}$. λ introduces the output cover (in fact, it always is a partition) ϕ_O such that

$$[o]\phi_O = [p]\phi_O \quad \iff \quad \forall s \in S \forall x \in I : o \in \lambda(s, x) \iff p \in \lambda(s, x)$$

The blocks of ϕ_O thus are the blocks of values of λ not distinguished from each other by any input/state combination (in most cases, $\phi_O = \phi_O(0)$). To calculate the output we need to be able to calculate either a block of ϕ_O or a block of a cover smaller than ϕ_O (with smaller blocks - removing output value from a block of ϕ_O is equivalent to filling in the don't-care). This is guaranteed by condition (4).

If additionally the state behavior of $\mathbf{M}$ is realized, then the state information of all the component machines enables the unambiguous computation of the state for the specification machine $\mathbf{M}$, i.e. a surjective partial function $\Phi' : \times S_i \to S$ exists. Such a function

introduces the following n covers $\phi^1_{S_l}$ on S:

$$\phi^i_{S_l} \quad : \quad [s]\phi^i_{S_l} \cap [t]\phi^i_{S_l} \neq \emptyset \iff s_i = t_i$$
$$\text{where } s = \Phi'(s_1, \ldots, s_i, \ldots, s_n), t = \Phi'(t_1, \ldots, t_i, \ldots, t_n)$$

Since Φ' is a surjective partial function, each element from S must be unambiguously defined by n-tuples of elements from $\times\phi^i_{S_l}$, i.e. elements of $\pi_S(0)$ must be calculated from the elements of $\prod_i \phi^i_{S_l}$. This is equivalent to stating condition (5).

Summarizing, if a sequential machine $\mathbf{M}$ has a general full-decomposition then n trinities of partitions $(\phi^i_I, \phi^i_{S_l}, \phi^i_{S_l \times I})$ exist and they satisfy conditions (1)–(5) of Theorem 3.2. This ends the proof.

3.4 General decomposition example

3.4.1 Output and state behavior realization

$i_0 i_1$	ps	ns	$o_0 o_1$
00	1	1	00
01	1	1	11,01
10	1	1	11,01
11	1	3	11
00	2	1,2	11
01	2	1	01
10	2	1	01
11	2	3	11
00	3	2	00
01	3	1	11,01
10	3	1	11,01
11	3	3	11
00	4	4	00
01	4	1	11,01
10	4	1	11,01
11	4	3	11
00	5	4,5	11
01	5	1	01
10	5	1	01
11	5	3	11

Figure 3.13. Example FSM

	00	01	10	11
$1a$	0	1	2	3
$1b$	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23

Figure 3.14. $S_l \times I$ space mapping

Let us now illustrate with an example the General decomposition theorem for the multi-state realizations of incompletely specified nondeterministic FSMs and its application for the construction of the decompositional FSM structures.

Let us consider an incompletely specified finite state machine $\mathbf{M}$ in Fig. 3.13 and its

decomposition into two partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$ defined by the following covers:

$$\phi_S^1 = \{\overline{1,2};\overline{1,3};\overline{4,5}\} \text{ and } \phi_I^1 = \{\overline{00};\overline{01,10,11}\} \text{ for } \mathbf{M}_1$$
$$\text{and } \phi_S^2 = \{\overline{1,3,4};\overline{1,2,5}\} \text{ and } \phi_I^2 = \{\overline{00,11};\overline{01,10}\} \text{ for } \mathbf{M}_2$$

After the labeling of states this introduces two copies of state 1: $1a$ and $1b$ and leaves other states unchanged. Two trinities of covers are given: $(\phi_I^1, \phi_{S_l}^1, \phi_{S_l \times I}^1)$ and $(\phi_I^2, \phi_{S_l}^2, \phi_{S_l \times I}^2)$, where:

$$\begin{aligned}
\phi_I^1 &= \{\overline{00};\overline{01,10,11}\} = \{i_1; i_2\} \quad \phi_{S_l}^1 = \{\overline{1a,2};\overline{1b,3};\overline{4,5}\} = \{a;b;c\} \\
\phi_{S_l \times I}^1 &= \{\overline{0,8,16,20};\overline{4,12}; \\
&\quad \overline{1,2,3,5,6,7,9,10,11,13,14,15,17,18,19,21,22,23}\} \\
&= \{o_1; o_2; o_3\} \\
\phi_I^2 &= \{\overline{00,11};\overline{01,10}\} = \{j_1; j_2\} \\
\phi_{S_l}^2 &= \{\overline{1a,3,4};\overline{1b,2,5}\} = \{x; y\} \\
\phi_{S_l \times I}^2 &= \{\overline{0,1,2,3,4,7,11,12,13,14,15,16,17,18,19,23}; \\
&\quad \overline{1,2,5,6,8,9,10,13,14,17,18,20,21,22}\} = \{p_1; p_2\}
\end{aligned}$$

In the above, for the conciseness of notation, the elements of the $S_l \times I$ space were replaced with the numbers assigned to them by the mapping introduced in Fig. 3.14.

The interpretation of this situation is as follows. Machine $\mathbf{M}$ has a multi-state realization $\mathbf{M}_L$, which in turn is composed of two partial machines: $\mathbf{M}_1$ and $\mathbf{M}_2$. States of the partial machines are described by covers $\phi_{S_l}^i$ $(i = 1, 2)$. Each cover $\phi_{S_l}^i$ groups in one block the states of the machine $\mathbf{M}_L$, which correspond to a single state of a partial machine $\mathbf{M}_i$ defined by this block. For example, $\phi_{S_l}^2$ defines a two-state machine $\mathbf{M}_2$, which is in its first state $\overline{1a,3,4}(x)$, whenever the machine $\mathbf{M}_L$ is in state $1a, 3$, or 4, and it is in its second state $\overline{1b,2,5}(y)$ whenever the machine $\mathbf{M}_L$ is in one of the states $1b, 2$ or 5.

Similarly, covers ϕ_I^i $(i = 1, 2)$ define (multi-valued) input variables to the partial machines $\mathbf{M}_i$. Values of the original inputs of $\mathbf{M}$ grouped in a single block of ϕ_I^i correspond to the value of i_i associated with this block. Note that except for primary input information delivered to partial machines and modeled by ϕ_I^i, the input space of partial machines is expanded by information imported from other partial machines, and modeled by ${\phi'_{S_l \times I}}^i$.

Multi-valued output variables of the partial machines $\mathbf{M}_i$ defined by the blocks of covers $\phi_{S_l \times I}^i$ $(i = 1, 2)$ convey partial information about the current state and input of the machine $\mathbf{M}_L$, which is computed from the total input and state information of $\mathbf{M}_i$ and transferred to $\mathbf{M}_i$'s output. This information is used by function Θ to determine the values of the primary outputs of $\mathbf{M}_L$ and by other partial machines to acquire the state/input information necessary for their computations and not produced locally. The scheme of this decomposition is presented in Fig. 3.15. In the following we will show how to determine the encoder/decoder functions and the STTs of the partial machines.

First, let us check whether the trinities define a valid decomposition. The derived covers mentioned in Theorem 2 are the following:

$$\phi_O = \{\overline{00};\overline{11};\overline{01}\}$$

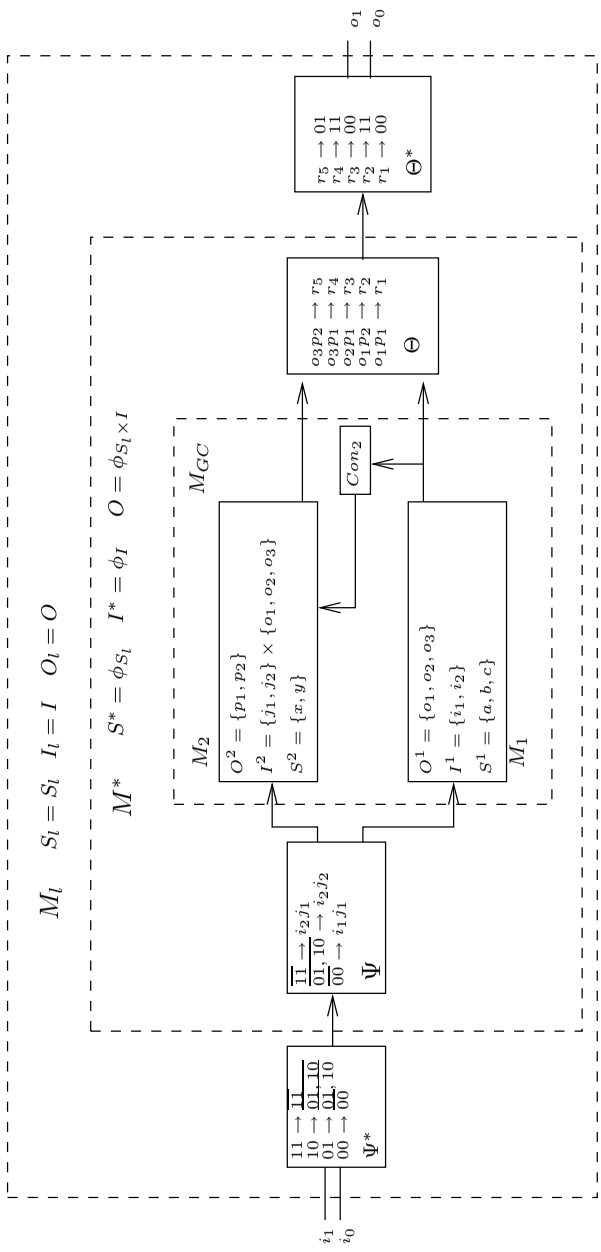


Figure 3.15. Scheme of the example decomposition

ϕ_O groups in one block these output symbols, which are not distinguished by *any* input combination; in this case there are no such symbols and ϕ_O is a zero-partition;

$$\begin{aligned}{\phi^I_{S_l\times I}}^1 &= \{\overline{0,4,8,12,16,20}(00);\\ &\quad \overline{1,2,3,5,6,7,9,10,11,13,14,15,17,18,19,21,22,23}(01,10,11)\}\end{aligned}$$

is the state/input information derived from the primary input information available to machine $\mathbf{M}_1$;

$$\begin{aligned}{\phi^I_{S_l\times I}}^2 &= \{\overline{0,3,4,7,8,11,12,15,16,19,20,23}(00,11);\\ &\quad \overline{1,2,5,6,9,10,13,14,17,18,21,22}(01,10)\}\end{aligned}$$

is the state/input information derived from the primary input information available to machine $\mathbf{M}_2$;

$$\begin{aligned}{\phi^{S_l}_{S_l\times I}}^1 &= \{\overline{0,1,2,3,8,9,10,11}(a);\overline{4,5,6,7,12,13,14,15}(b);\\ &\quad \overline{16,17,18,19,20,21,22,23}(c)\}\end{aligned}$$

is the state/input information derived from the state information available to machine $\mathbf{M}_1$;

$$\begin{aligned}{\phi^{S_l}_{S_l\times I}}^2 &= \{\overline{0,1,2,3,12,13,14,15,16,17,18,19}(x);\\ &\quad \overline{4,5,6,7,8,9,10,11,20,21,22,23}(y)\}\end{aligned}$$

is the state/input information derived from the state information available to machine $\mathbf{M}_2$;

$$\begin{aligned}
\phi^I_{S_l\times I} &= {\phi^I_{S_l\times I}}^1 * {\phi^I_{S_l\times I}}^2 = \{\overline{0,4,8,12,16,20}(00);\overline{3,7,11,15,19,23}(11);\\
&\quad \overline{1,2,5,6,9,10,13,14,17,18,21,22}(01,10)\}\\
\phi^{S_l}_{S_l\times I} &= {\phi^{S_l}_{S_l\times I}}^1 * {\phi^{S_l}_{S_l\times I}}^2 = \{\overline{0,1,2,3}(1a);\overline{4,5,6,7}(1b);\overline{8,9,10,11}(2);\\
&\quad \overline{12,13,14,15}(3);\overline{16,17,18,19}(4);\overline{20,21,22,23}(5)\}\\
\phi^{21}_{S_l\times I} &= {\phi'_{S_l\times I}}^1 = \phi_{S_l\times I}(1)\\
\phi^{12}_{S_l\times I} &= {\phi'_{S_l\times I}}^2 = \phi^1_{S_l\times I}\\
\phi_{S_l} &= \phi^1_{S_l}\cdot\phi^2_{S_l} = \{\overline{1a}(ax);\overline{2}(ay);\overline{3}(bx);\overline{1b}(by);\overline{4}(cx);\overline{5}(cy)\}\\
\phi_{S_l\times I} &= \phi^1_{S_l\times I}\cdot\phi^2_{S_l\times I} = \{\overline{0,16}(o1,p1);\overline{8,20}(o1,p2);\overline{4,12}(o2,p1);\\
&\quad \overline{1,2,3,7,11,13,14,15,17,18,19,23}(o3,p1);\\
&\quad \overline{1,2,5,6,9,10,13,14,17,18,21,22}(o3,p2)\}\\
&= \{r_1,r_2,r_3,r_4,r_5\}
\end{aligned}$$

Condition (1 p.1) requires that the entire information available to a partial machine $\mathbf{M}_i$ (i.e. ${\phi^I_{S_l\times I}}^i\cdot{\phi^{S_l}_{S_l\times I}}^i\cdot{\phi'_{S_l\times I}}^i$) is sufficient to unambiguously calculate the next state of

$\mathbf{M}_i$. Similarly, condition (2) requires that this information is sufficient to unambiguously calculate the output of $\mathbf{M}_i$. The total information available to the partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$ is as follows:

$$\begin{aligned}
{\phi^{in}_{S_l\times I}}^1 &= {\phi^{I}_{S_l\times I}}^1 \cdot {\phi^{S_l}_{S_l\times I}}^1 \cdot {\phi'_{S_l\times I}}^1 = \\
&\{\overline{0,8};\overline{4,12};\overline{16,20};\overline{1,2,3,9,10,11};\overline{5,6,7,13,14,15};\\
&\overline{17,18,19,21,22,23}\} - \text{for } \mathbf{M}_1\\
{\phi^{in}_{S_l\times I}}^2 &= {\phi^{I}_{S_l\times I}}^2 \cdot {\phi^{S_l}_{S_l\times I}}^2 \cdot {\phi'_{S_l\times I}}^2 = \\
&\{\overline{0,16};\overline{12};\overline{3,15,19};\overline{8,20};\overline{4};\overline{7,11,23};\overline{1,2,13,14,17,18};\\
&\overline{5,6,9,10,21,22}\} - \text{for } \mathbf{M}_2
\end{aligned}$$

It is easy to verify that the condition (2) is met, i.e. ${\phi^{in}_{S_l\times I}}^1 \leq {\phi_{S_l\times I}}^1$ and ${\phi^{in}_{S_l\times I}}^2 \leq {\phi_{S_l\times I}}^2$.

To show how to verify condition (1), let us consider the first block of ${\phi^{in}_{S_l\times I}}^1$: $\overline{0,8}$. First, we check whether this block leads to a block of $\phi^1_{S_l}$. For this purpose, we calculate the value of $\Delta^{\phi^1_{S_l}}(\overline{0,8})$ (Def. 3.16). Recall that $\Delta^{\phi^i_{S_l}}(D)$ returns a block of $\phi^i_{S_l}$, which includes possible next states for all elements of D. Since $D = \overline{0,8} = \overline{(1a,00)(2,00)}$, its unlabeled version is $L'(D) = \overline{(1,00)(2,00)}$. In the specification machine $\mathbf{M}$ this block can lead to the blocks described by

$$\overline{\delta}(\overline{(1,00)(2,00)}) = \overline{\bigcup}\begin{bmatrix}\delta(1,00)\\ \delta(2,00)\end{bmatrix} = \overline{\bigcup}\begin{bmatrix}\{1\}\\ \{1,2\}\end{bmatrix} = \{\{1\},\{1,2\}\}.$$

For the composition machine to be consistent with the specification machine, the next state of $\mathbf{M}_1$ ($B'_l \in \phi^1_{S_l}$) must include copies of the states in $\overline{\delta}(\overline{0,8})$, i.e. $\{1\} \subseteq L'(B'_l)$ or $\{1,2\} \subseteq L'(B'_l)$. The blocks of $\phi^1_{S_l}$ satisfying this condition are $\overline{1a,2}$ and $\overline{1b,3}$. Thus, $\Delta^{\phi^1_{S_l}}(\overline{0,8}) = \{\overline{1a,2};\overline{1b,3}\} \neq \emptyset$ and the first part of condition (1) (cover pair condition) for the block $\overline{0,8}$ is satisfied.

Similarly, for the first block of ${\phi^{in}_{S_l\times I}}^2 = \overline{0,16} = \overline{(1a,00)(4,00)}$, $L'(\overline{(1a,00)(4,00)}) = \overline{(1,00)(4,00)}$.

$$\overline{\delta}(\overline{(1,00)(4,00)}) = \overline{\bigcup}\begin{bmatrix}\delta(1,00)\\ \delta(4,00)\end{bmatrix} = \overline{\bigcup}\begin{bmatrix}\{1\}\\ \{4\}\end{bmatrix} = \{\{1,4\}\}.$$

The only block of $\phi^2_{S_l}$ that is consistent with $\overline{1,4}$ is $\overline{1a,3,4}$, and therefore $\Delta^{\phi^2_{S_l}}(\overline{0,16}) = \{\overline{1a,3,4}\} \neq \emptyset$.

The synchronization criterion of the synchronized sets of cover pairs additionally requires that the intersection of the blocks of the input covers of the partial machines ($\bigcap_i D^i \in {\phi^{in}_{S_l\times I}}^i$) leads to a block being an intersection of the partial machine's next states. In the case of the previously discussed blocks, for block $\overline{0,8}$ of ${\phi^{in}_{S_l\times I}}^1$ there are two blocks of ${\phi^{in}_{S_l\times I}}^2$ with nonempty intersections: $\overline{0,16}$ and $\overline{8,20}$. For the first combination ($D^1 = \overline{0,8}$, $D^2 = \overline{0,16}$), $\Delta^{\phi^1_{S_l}}(D^1) = \{\overline{1a,2},\overline{1b,3}\}$, $\Delta^{\phi^2_{S_l}}(D^2) = \{\overline{1a,3,4}\}$, i.e. in these input conditions, $\mathbf{M}_1$ can perform the transition to state $\overline{1a,2}$ or $\overline{1b,3}$, and $\mathbf{M}_2$

transits to $\overline{1a,3,4}$. The possible products of the states are given by

$$\overline{\bigcap}_i \Delta^{\phi^i_{S_l}}(D^i) = \overline{\bigcap}\left[\begin{array}{c}\{\overline{1a,2},\overline{1b,3}\}\\ \{\overline{1a,3,4}\}\end{array}\right] = \{\overline{1a};\overline{3}\}.$$

This means that the combination of the two partial machines can move to either $\overline{1a}$ or $\overline{3}$ in these conditions . Meanwhile, the product of the input conditions ($\bigcap_i D_i = \overline{0}$) should lead to the state $\Delta^{\prod_i \phi^i_{S_l}}(\overline{0})$ in the composition machine. $\overline{\delta}(L'(\overline{0})) = \overline{\delta}(1,00) = 1$, so $\Delta^{\prod_i \phi^i_{S_l}}(\overline{0}) = \{\overline{1a};\overline{1b}\}$. This means that for the composition machine to be consistent with the specification machine, it has to move to either $\overline{1a}$ or $\overline{1b}$. The confrontation of these results shows that it is possible for the combination of partial machines to move to a state inconsistent with the specification machine ($\overline{3}$). In terms of the definition of the synchronized set of cover pairs it means that for $B'_{D^1} = \overline{1b,3} \in \Delta^{\phi^1_{S_l}}(D^1)$ and $B'_{D^2} = \overline{1a,3,4} \in \Delta^{\phi^2_{S_l}}(D^2)$ their intersection ($\bigcap_i B'_{D^i} = \overline{3}$) is not included in any block of $\Delta^{\prod_i \phi^i_{S_l}}(\bigcap_i D^i) = \{\overline{1a};\overline{1b}\}$.

However, if we limit the freedom of $\mathbf{M}_1$ by removing the state $\overline{1b,3}$ from the set of its possible next states, the synchronization condition for the remaining state of $\mathbf{M}_1$ ($\overline{1a,2}$) is satisfied, because for $B'_{D^1} = \overline{1a,2} \in \Delta^{\phi^1_{S_l}}(D^1)$ and $B'_{D^2} = \overline{1a,3,4} \in \Delta^{\phi^2_{S_l}}(D^2)$ their intersection ($\bigcap_i B'_{D^i} = \overline{1a}$) is included in a block of $\Delta^{\prod_i \phi^i_{S_l}}(\bigcap_i D^i)$.

Following this procedure we can verify that the next state blocks for the blocks of ${\phi^{in}_{S_l\times I}}^1$ are as follows: $\{\overline{1a,2}(a),\overline{1b,3}(b)\}$ for $\overline{0,8}$ (and $\overline{1b,3}(b)$ is then removed by the synchronization criterion); a for $\overline{4,12}$; c for $\overline{16,20}$; b for $\overline{1,2,3,9,10,11}$; b for $\overline{5,6,7,13,14,15}$; and b for $\overline{17,18,19,21,22,23}$. For the blocks of ${\phi^{in}_{S_l\times I}}^2$ the next state blocks are as follows: x for $\overline{0,16}$; y for $\overline{12}$; x for $\overline{3,15,19}$; $\{x;y\}$ for $\overline{8,20}$; $\{x;y\}$ for $\overline{4}$ (and y is removed by the synchronization criterion); x for $\overline{7,11,23}$; $\{x;y\}$ for $\overline{1,2,13,14,17,18}$ (and x is removed by the synchronization criterion); $\{x;y\}$ for $\overline{5,6,9,10,21,22}$ (and x is removed by the synchronization criterion).

Condition (3) requires that the information imported by any partial machine does not exceed the total information available to all partial machines. It is satisfied, because

$$\begin{array}{rcl}\phi^I_{S_l\times I}\cdot\phi^{S_l}_{S_l\times I} & = & \{\overline{0};\overline{1,2};\overline{3};\overline{4};\overline{5,6};\overline{7};\overline{8};\overline{9,10};\overline{11};\overline{12};\overline{13,14};\overline{15};\overline{16};\\ & & \overline{17,18};\overline{19};\overline{20};\overline{21,22};\overline{23}\}\end{array}$$

is smaller than both ${\phi'_{S_l\times I}}^i$.

The condition (4) is met when all elements in a block of $\phi_{S_l\times I}$ (after unlabeling) have a common subset of output values in the specification machine $\mathbf{M}$ that is included in a block of some specialization $\phi_{O'}$ of ϕ_O. Recall that $\phi_{S_l\times I} = \{\overline{0,16};\overline{8,20};\overline{4,12};$ $\overline{1,2,3,7,11,13,14,15,17,18,19,23};\overline{1,2,5,6,9,10,13,14,17,18,21,22}\}$ and $\phi_O =$ $= \{\overline{00};\overline{11};\overline{01}\}$. Let $\phi_{O'} = \phi_O$. To check condition (4) we need to compute values of $\Lambda^{\phi_{O'}}(D)$ for each $D \in \phi_{S_l\times I}$.
For example, for block $D = \overline{1,2,3,7,11,13,14,15,17,18,19,23}$, the value of $\overline{\lambda}(L'(D))$ is the combination of all possible output values produced in the specification machine by elements of D. For the unlabeled elements $3,7,11,15,19,23$ the output value of the specification machine is 11. The unlabeled versions of elements $1,2,13,14,17,18$ produce output 01 or 11 in the specification machine. Therefore, $\overline{\lambda}(L'(D)) = \{\overline{11};\overline{11,01}\}$.

Since $\overline{11}$ fits in the second block of $\phi_{O'}$, so $\Lambda^{\phi_{O'}}(D) = \overline{11} \neq \emptyset$. For the other blocks of $\phi_{S_l \times I}$, the values of $\Lambda^{\phi_{O'}}(D)$ are: 00 for $\overline{0, 16}$, 11 for $\overline{8, 20}$, 00 for $\overline{4, 12}$ and 01 for $\overline{1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22}$.

Condition (5) must only be met for the state behavior realization. It states that each block of the product state cover of the partial machines needs to contain exclusively copies of the same state of $\mathbf{M}$, so that the state of the specification machine can be determined unambiguously. In this case, $\phi^1_{S_l} \cdot \phi^2_{S_l} = \phi_{S_l}(0)$ so the condition is met and the decomposition is both state and output behavior realization.

Thus we have demonstrated that the above given covers indeed define a valid decomposition of $\mathbf{M}$.

3.4.2 Output behavior realization

Similarly as in the example in Section 3.1.2, the state space of machine $\mathbf{M}^*$ can be minimized without affecting its output behavior. If the state space of $\mathbf{M}_1$ is defined by the cover $\phi^1_{S_l} = \{\overline{1a, 2, 4, 5}(a); \overline{1b, 3}(b)\}$, the product state cover of both partial machines becomes $\phi_{S_l} = \{\overline{1a, 4}(ax); \overline{2, 5}(ay); \overline{3}(bx); \overline{1b}(by); \}$, which does not form a $S_l - S$ pair with $\phi_S(0)$, since the first block contains copies of two different states 1 and 4 for example. This is a violation of (5), and so the state behavior of $\mathbf{M}$ is not realized.

If the input and output covers remain unchanged, the only covers affected by the change of $\phi^1_{S_l}$ are the following:

$$\begin{aligned}
{\phi^{S_l}_{S_l \times I}}^1 = \ & \{ \ \overline{0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, 19, 20, 21, 22, 23}(a); \\
& \overline{4, 5, 6, 7, 12, 13, 14, 15}(b)\} \\
{\phi^{in}_{S_l \times I}}^1 = \ & \{ \ \overline{0, 8, 16, 20}; \overline{1, 2, 3, 9, 10, 11, 17, 18, 19, 21, 22, 23}; \\
& \overline{4, 12}; \overline{5, 6, 7, 13, 14, 15}\} \\
\phi^{S_l}_{S_l \times I} * \phi^{I}_{S_l \times I} = \ & \{ \ \overline{0, 16}; \overline{1, 2, 17, 18}; \overline{3, 19}; \overline{4}; \overline{5, 6}; \overline{7}; \overline{8, 20}; \overline{9, 10, 21, 22}; \\
& \overline{11, 23}; \overline{12}; \overline{13, 14}; \overline{15}\}
\end{aligned}$$

The verification of condition (1) proceeds along the lines outlined in Section 3.4.1. For example, let us consider block $D = \overline{0, 8, 16, 20}$, $\Delta^{\phi^1_{S_l}}(D) = \{B'_l \in \phi_{S_l} | \exists B' \in \overline{\delta}(L'(D)) : B' \subseteq L'(B'_l)\}$. The possible next state of the specification machine for this block is given by $\overline{\delta}(L'(D))$. Since $L'(D) = L'(\overline{(1a, 00)(2, 00)(4, 00)(5, 00)}) = \overline{(1, 00)(2, 00)(4, 00)(5, 00)}$,

$$\begin{aligned}
\overline{\delta}(L'(D)) &= \overline{\bigcup} \begin{bmatrix} \delta(1, 00) \\ \delta(2, 00) \\ \delta(4, 00) \\ \delta(5, 00) \end{bmatrix} = \overline{\bigcup} \begin{bmatrix} \{1\} \\ \{1, 2\} \\ \{4\} \\ \{4, 5\} \end{bmatrix} = \\
&= \{\{1, 4\}; \{1, 2, 4\}; \{1, 2, 5\}; \{1, 4, 5\}; \{1, 2, 4, 5\}\}.
\end{aligned}$$

Since any of these blocks fits in the unlabeled version of the first block of $\phi^1_{S_l}$ ($L'(\overline{1a, 2, 4, 5}) = \overline{1, 2, 4, 5}$), the next state of $\mathbf{M}_1$ for the input situation described by D will be a. For the other blocks of ${\phi^{in}_{S_l \times I}}^1$, the next states are b, a and b, respectively.

As the cover ${\phi^{in}_{S_l \times I}}^2$ did not change, the condition of $S_l \times I - S_l$ pair for ${\phi^{in}_{S_l \times I}}^2$ and $\phi^2_{S_l}$ is still satisfied. However, the change of ${\phi^{in}_{S_l \times I}}^1$ can influence the synchronization

criterion of the synchronized set of cover pairs. In this case, the synchronization criterion is still satisfied, so condition (1) is still met for both partial machines. Also, ${\phi^{in}_{S_l \times I}}^1$ is still smaller-or-equal-to $\phi^1_{S_l \times I}$, so condition (2) is met. Since ${\phi'_{S_l \times I}}^1 = \phi_{S_l \times I}(1)$, condition (3) is still satisfied. Finally, the output cover of $\mathbf{M}_1$ did not change, so condition (4) remains satisfied.

Thus, if $\phi^1_{S_l} = \{\overline{1a, 2, 4, 5}(a); \overline{1b, 3}(b)\}$ the covers satisfy all conditions of General Decomposition Theorem except for (5) and the network of partial machines described by these covers realizes output behavior of $\mathbf{M}$ without realizing its state behavior.

3.4.3 Construction of the decomposition structure

Composition machine $\mathbf{M}^*$

We will show, how the machines $\mathbf{M}^*$ and $\mathbf{M}_i$ considered in the proof of the General Decomposition Theorem can actually be constructed from their cover descriptions, and that they indeed constitute a realization of machine $\mathbf{M}$.

Machine $\mathbf{M}^*$ is defined by a quintuple $(\phi_I, \phi_{S_l}, \phi_{S_l \times I}, \delta^*, \lambda^*)$. Its input alphabet is described by the cover $\phi_I = \phi^1_I * \phi^2_I = \{\overline{00}; \overline{11}; \overline{01, 10}\}$. It therefore has three distinct input values. Similarly, its state cover $\phi_{S_l} = \phi^1_{S_l} * \phi^2_{S_l} = \{\overline{1a}; \overline{1b}; \overline{2}; \overline{3}; \overline{4}; \overline{5}\}$, describes a 6-valued state variable. Finally, output of the machine is described by the output cover

$$\phi_{S_l \times I} = \phi^1_{S_l \times I} \cdot \phi^2_{S_l \times I} = \quad \{ \quad \overline{0, 16}(r_1); \overline{8, 20}(r_2); \overline{4, 12}(r_3); \\ \overline{1, 2, 3, 7, 11, 13, 14, 15, 17, 18, 19, 23}(r_4); \\ \overline{1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22}(r_5)\}$$

that defines a 5-valued output variable.

To build the state transition table (STT) for machine $\mathbf{M}^*$, we need to calculate values of its next-state and output functions for all combinations of input (blocks of ϕ_I) and state (blocks of ϕ_{S_l}). Recall that according to definitions (3.3.7) and (3.3.8), the functions are defined as follows:

$$\forall B \in \phi_{S_l}\ \forall A \in \phi_I\ :\ \delta^*(B, A) = \Delta^{\phi_{S_l}}(B \times A) \text{ and } \lambda^*(B, A) = [B \times A]\phi_{S_l \times I}$$

Let us consider the behavior of $\mathbf{M}^*$ in the state $B = \overline{1a}$ under the input $A = \overline{01, 10}$. The total state/input information available in these conditions is $B \times A = \overline{(1a, 01)(1a, 10)} = \overline{1, 2}$. Next state of the machine is calculated by calculating the next state for this block as $\Delta^{\phi_{S_l}}$. In this case, the unlabeled version of this block is $\overline{(1, 01)(1, 10)}$, which leads to state 1 in the specification machine. In $\mathbf{M}^*$ state 1 can be labeled to fit either in the block $\overline{1a}$ or $\overline{1b}$ of $\phi^1_{S_l}$. Therefore, $\delta^*(B, A) = \{\overline{1a}; \overline{1b}\}$. The output of $\mathbf{M}^*$ in this situation is established as a set of the blocks of the output cover including $B \times A$. In this case, these are blocks 4 and 5, so $\lambda^*(B, A) = r_4, r_5$.

Following this procedure, the STT in Fig. 3.16 is built. Since a check was done in the verification process of the condition (4) to see whether for any block of $\phi_{S_l \times I}$ an output value of the specification machine can be computed, the corresponding output values are given in parenthesis. It can be easily verified that machine in Fig. 3.16 is a multi-state state and output behavior realization of machine in Fig. 3.13, because for any copy of a state of $\mathbf{M}$, $\mathbf{M}^*$ moves under the same input conditions to some copy (or copies) of the next states of $\mathbf{M}$ and produces a subset of output values.

in	ps	ns	out
$\overline{00}$	$1a$	$1a, 1b$	$r_1(00)$
$\overline{01, 10}$	$1a$	$1a, 1b$	$r_4(11), r_5(01)$
$\overline{11}$	$1a$	3	$r_4(11)$
$\overline{00}$	$1b$	$1a, 1b$	$r_3(00)$
$\overline{01, 10}$	$1b$	$1a, 1b$	$r_5(01)$
$\overline{11}$	$1b$	3	$r_4(11)$
$\overline{00}$	2	$1a, 1b, 2$	$r_2(11)$
$\overline{01, 10}$	2	$1a, 1b$	$r_5(01)$
$\overline{11}$	2	3	$r_4(11)$
$\overline{00}$	3	2	$r_3(00)$
$\overline{01, 10}$	3	$1a, 1b$	$r_4(11), r_5(01)$
$\overline{11}$	3	3	$r_4(11)$
$\overline{00}$	4	4	$r_1(00)$
$\overline{01, 10}$	4	$1a, 1b$	$r_4(11), r_5(01)$
$\overline{11}$	4	3	$r_4(11)$
$\overline{00}$	5	$4, 5$	$r_2(11)$
$\overline{01, 10}$	5	$1a, 1b$	$r_5(01)$
$\overline{11}$	5	3	$r_4(11)$

Figure 3.16. Transition table of machine $\mathbf{M}^*$

Partial machines $\mathbf{M}_i$

Remember that the partial machine $\mathbf{M}_1$ has three states associated with the states of the original machine in the way defined by cover $\phi^1_{S_l}$. We will refer to those states as a, b and c. The input alphabet of $\mathbf{M}_1$ is determined by combination of ϕ^1_I and ${\phi'_{S_l \times I}}^1$. In this case, since ${\phi'_{S_l \times I}}^1 = \phi_{S_l \times I}(0)$ imported cover does not introduce new information, and may be omitted in consideration. The possible input values for $\mathbf{M}_1$ therefore are i_1 and i_2. Let us establish functions δ^1 and λ^1, as defined by expressions (3.3.19) and (3.3.20).

For example, let us consider behavior of $\mathbf{M}_1$ in state a under input i_1. The current state and input of the specification machine $\mathbf{M}$ available to the partial machine $\mathbf{M}_1$ is given by the expression $(a \times i_1) \cap (S_l \times I)$. The first component of the expression is the information available locally in the machine (combination of local state information and local input information), while the second component expresses a refinement of the information available through imported information. In this case, since no information is imported, the lack of refinement is indicated by an intersection with the entire space $(S_l \times I)$. The information available to the machine therefore is $(a \times i_1) = (\overline{1a, 2} \times \overline{00}) = \overline{(1a, 00); (2, 00)}$ or, in short, $\overline{0, 8}$. To establish the next state for these input conditions, we follow the procedure used for verification of condition (1) in Section 3.4.1. Since each distinct input situation of a partial machine corresponds to a block of ${\phi^{in}_{S_l \times I}}^i$, condition (1) guarantees that a next state can be found for this input condition. In this case, in Section 3.4.1 we have shown that the next state in input conditions $\overline{0, 8}$ is a or b, but the synchronization criterion excludes b. Thus, $\delta^1(a, i_1) = a$. Condition (2) guarantees that $\overline{0, 8}$ is included in some block of $\phi^1_{S_l \times I}$ (in this case, the first), and the output of $\mathbf{M}_1$

therefore is $\lambda^1(a, i_1) = o_1$. Following the same procedure we obtain the complete state transition table of machine $\mathbf{M}_1$ given in Fig. 3.17.

Similarly, machine $\mathbf{M}_2$ has two states x and y defined by blocks of $\phi^2_{S_l}$. However, since $\mathbf{M}_2$ imports information from $\mathbf{M}_1$, its input alphabet is a combination of symbols of ϕ^2_I and ${\phi'_{S_l \times I}}^2 : \{j_1, j_2\} \times \{J'_1, J'_2, J'_3\}$. Since ${\phi'_{S_l \times I}}^2 = \phi^1_{S_l \times I}$, $J'_1 = o_1$,$J'_2 = o_2$ and $J'_3 = o_3$, and hence $I^2 = \{(j_1, o_1); (j_1, o_2); (j_1, o_3); (j_2, o_1); (j_2, o_2); (j_2, o_3)\}$. Let us consider, for example, the transition of $\mathbf{M}_2$ from state x under input (j_1, o_1). The input/state of the specification machine is given by $(x \times j_1) \cap o_1 = (\overline{1a, 3, 4} \times \overline{00, 11}) \cap \overline{0, 8, 16, 20} = \overline{0, 3, 12, 15, 16, 19} \cap \overline{0, 8, 16, 20} = \overline{0, 16}$. As discussed earlier, this combination leads to the state associated with the first block of $\phi^2_{S_l}$. The next state for this combination therefore is $\delta^2(x, (j_1, o_1)) = x$. The output of the partial machine $\mathbf{M}_2$ for this combination is the block of $\phi^2_{S_l \times I}$ containing $\overline{0, 16}$, i.e. p_1. In the same fashion, the remaining transitions of $\mathbf{M}_2$ can be determined. The resulting STT of $\mathbf{M}_2$ is given in Fig. 3.18. Note that four of the transitions do not have next-state/output values specified. These transitions correspond to the input combinations of j_2 and o_1 or o_2, which actually cannot occur. This stems from the fact that j_2 corresponds to primary input values 01 or 10, while both o_1 and o_2 may only be produced by $\mathbf{M}_1$ under input i_1, which corresponds to primary input value 00. These two input conditions cannot occur at the same time, and therefore the four transitions may be treated as don't-cares and used for minimization of $\mathbf{M}_2$.

The combination of the above-constructed partial machines $\mathbf{M}_1$ and $\mathbf{M}_2$ (the machine $\mathbf{M}_{GC}$) has a transition table defined by functions δ_{GC} and λ_{GC}. To illustrate determination of these functions, let us consider a situation, in which machine $\mathbf{M}_1$ is in state a, $\mathbf{M}_2$ in state x, and the value of function Ψ is $i_1 j_1$.

in	ps	ns	out
i_1	a	a	o_1
i_2	a	b	o_3
i_1	b	a	o_2
i_2	b	b	o_3
i_1	c	c	o_1
i_2	c	b	o_3

Figure 3.17. STT of $\mathbf{M}_1$

in	imp	ps	ns	out
j_1	o_1	x	x	p_1
j_1	o_2	x	y	p_1
j_1	o_3	x	x	p_1
j_2	o_1	x	*	*
j_2	o_2	x	*	*
j_2	o_3	x	y	p_1,p_2
j_1	o_1	y	x, y	p_2
j_1	o_2	y	x	p_1
j_1	o_3	y	x	p_1
j_2	o_1	y	*	*
j_2	o_2	y	*	*
j_2	o_3	y	y	p_2

Figure 3.18. STT of $\mathbf{M}_2$

$$\delta_{GC}\left(\left[\begin{array}{c} a \\ x \end{array}\right],\left[\begin{array}{c} i_1 \\ j_1 \end{array}\right]\right) = \left[\begin{array}{c} \bigcup_{C^1\in[(a\cap x)\times(i_1\cap j_1)]\phi'_{S_l\times I}{}^1}\delta^1(a,(i_1,C^1)) \\ \bigcup_{C^2\in[(a\cap x)\times(i_1\cap j_1)]\phi'_{S_l\times I}{}^2}\delta^2(x,(j_1,C^2)) \end{array}\right]$$
$$= \left[\begin{array}{c} \delta^1(a,(i_1,S_l\times I)) \\ \delta^2(x,(j_1,o_1)) \end{array}\right] = \left[\begin{array}{c} \{a\} \\ \{x\} \end{array}\right]$$
$$\lambda_{GC}\left(\left[\begin{array}{c} a \\ x \end{array}\right],\left[\begin{array}{c} i_1 \\ j_1 \end{array}\right]\right) = \left[\begin{array}{c} \bigcup_{C^1\in[(a\cap x)\times(i_1\cap j_1)]\phi'_{S_l\times I}{}^1}\lambda^1(a,(i_1,C^1)) \\ \bigcup_{C^2\in[(a\cap x)\times(i_1\cap j_1)]\phi'_{S_l\times I}{}^2}\lambda^2(x,(j_1,C^2)) \end{array}\right]$$
$$= \left[\begin{array}{c} \lambda^1(a,(i_1,S_l\times I)) \\ \lambda^2(x,(j_1,o_1)) \end{array}\right] = \left[\begin{array}{c} \{o_1\} \\ \{p_1\} \end{array}\right]$$

This situation triggers transition (i_1, a, a, o_1) in $\mathbf{M}_1$. Since the output of $\mathbf{M}_1$ is o_1, the transition (j_1, o_1, x, x, p_1) is triggered in $\mathbf{M}_2$. Combined, this results in transition $([i_1, j_1], [a, x], [a, x], [o_1, p_1])$. The entire STT of $\mathbf{M}_{GC}$is given in Fig. 3.19. Taking into consideration the mapping functions Ψ, Φ and Θ, we obtain corresponding values of input, state and output of the machine $\mathbf{M}^*$ given in parenthesis. Again, it can be verified that the composition of partial machines $\mathbf{M}_i$ is a state and output behavior realization of $\mathbf{M}^*$ and of $\mathbf{M}$.

In this way, we have demonstrated how the trinities of covers from Theorem 3.2 determine the multi-state decompositional realization structures of incompletely specified sequential machines.

in	ps	ns	out
$[i_1, j_1](00)$	$[a, x](1a)$	$[a, x](1a)$	$[o_1,p_1](r_1)$
$[i_2, j_2](01, 10)$	$[a, x](1a)$	$[b, y](1b)$	$[o_3,p_1](r_4)$,$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[a, x](1a)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_1](00)$	$[a, y](2)$	$[a, x](1a), [a, y](2)$	$[o_1,p_2](r_2)$
$[i_2, j_2](01, 10)$	$[a, y](2)$	$[b, y](1b)$	$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[a, y](2)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_1](00)$	$[b, x](3)$	$[a, y](2)$	$[o_2,p_1](r_3)$
$[i_2, j_2](01, 10)$	$[b, x](3)$	$[b, y](1b)$	$[o_3,p_1](r_4)$,$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[b, x](3)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_1](00)$	$[b, y](1b)$	$[a, x](1a)$	$[o_2,p_1](r_3)$
$[i_2, j_2](01, 10)$	$[b, y](1b)$	$[b, y](1b)$	$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[b, y](1b)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_1](00)$	$[c, x](4)$	$[c, x](4)$	$[o_1,p_1](r_1)$
$[i_2, j_2](01, 10)$	$[c, x](4)$	$[b, y](1b)$	$[o_3,p_1](r_4)$,$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[c, x](4)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_1](00)$	$[c, y](5)$	$[c, x](4), [c, y](5)$	$[o_1,p_2](r_2)$
$[i_2, j_2](01, 10)$	$[c, y](5)$	$[b, y](1b)$	$[o_3,p_2](r_5)$
$[i_2, j_1](11)$	$[c, y](5)$	$[b, x](3)$	$[o_3,p_1](r_4)$
$[i_1, j_2](\emptyset)$	*	*	*

Figure 3.19. STT of $\mathbf{M}_{GC}$

3.5 Conclusions

In this chapter, we have presented the theory of decomposition of finite state machines based on information modeling with covers. We have presented the theorem about existence of the general decomposition of an incompletely specified FSM into a network of partial FSMs interconnected in an arbitrary fashion, with input encoder and output decoder and with multi-state realization of the FSM. The theorem views all elements of the decomposition (i.e. partial machines and coders) as information processing sub-systems, where the input and output information streams are modeled by covers. The theorem formulates conditions that these input/output covers need to fulfill in order to define a valid decomposition. Given a set of covers fulfilling the conditions of the theorem, we are guaranteed to be able to build an actual network of partial FSMs that will realize the behavior of the original specification machine. We demonstrated with an example, how the covers describing the decomposition relate to the particular partial machines and coder blocks in the decomposition network.

The decomposition scheme described by the theorem is the most general known decomposition scheme for any finite state machine. It includes multi-state behavior realizations and generalized don't-cares in both output and next-state functions. Thus, it covers also nondeterministic FSMs. Since Boolean functions are a special case of FSMs with a single state and trivial next-state function, the theorem also covers decompositions of Boolean functions. In [35], Jóźwiak showed that other well known decomposition structures, such as serial and parallel decomposition, are some special cases of the structures described by the General Decomposition Theorem.

In the following chapters we will focus on one specific special case of the General Decomposition Theorem that is related to FSM state assignment, and we will use the conditions of the theorem to formulate conditions for a valid assignment. We will also mention other practical application of the theorem to functional decomposition of Boolean functions. However, the applicability of the theorem goes far beyond these two applications. It can be used in any field dealing with discrete systems involving finite state machines, functions or relations, and their decompositions.

Chapter 4

General Decomposition in circuit synthesis

4.1 Sequential synthesis

The goal of FSM state encoding is to assign binary codes to the symbolic states of a sequential machine in such a way that the resulting binary next-state and output functions can be effectively and efficiently realized in the target implementation technology. Regardless of the method used to assign the codes, a sequential machine encoded with a valid encoding is functionally equivalent or compatible with the original, symbolic machine. If no state minimization is performed by the state encoding, the encoded machine is a state and output behavior realization of the symbolic machine. Otherwise it is an output behavior realization. We can interpret the encoded machine structure as a special case of the general decomposition of the original, symbolic machine into a network of two-state (binary) partial machines.

In the following, we will show how a particular state assignment induces the corresponding decomposition of the symbolic machine. By applying conditions of the General Decomposition Theorem to this specific decomposition, we derive conditions for a valid state assignment. These conditions, which are more general and flexible than conditions used in current state assignment methods, form the basis of a general, implementation-platform-independent encoding method. As a result of the underlying flexible conditions, the method is able to exploit some implicit optimizations of the encoded machine, making its implementation more efficient.

4.1.1 State-encoding-induced decomposition

Let us consider how state encoding induces decomposition of a symbolic finite state machine. First, we will discuss encoding of completely specified machines and then extend the results to the incompletely specified case.

To illustrate the process, we will consider the completely specified FSM given in Fig. 4.1(a). The example state assignment for this machine, given in Fig. 4.1(b), minimized the state space of the machine by assigning the same code to states b and e. The gate implementation of the example machine is shown in Fig. 4.2. We superimposed on

the circuit the outline of the decomposition structure introduced by the encoding. Let us discuss how this structure was derived.

Decomposition structure

The current state of the symbolic machine is determined by the value of a single symbolic (multi-valued) state variable. In the case of encoded machine, the current state is a composition of the states of the binary encoding variables. For instance, in the encoded machine in Fig. 4.1 the current state a is composition of the state '0' of the first encoding bit and state '1' of the second encoding bit, while state b is indicated by state '1' of the first encoding bit and '1' of the second. In this formulation, we can view each binary encoding variable as a state variable of a partial, two-state machine. The composition of the current states of these partial machines gives the current state of the realized original symbolic machine.

In addition to the current state computation performed by the partial state machines, the encoded machine has to produce the primary output of the realized machine. This task is performed by the output decoder block Θ. This combinational block uses outputs of the partial machines (binary encoding bits), which deliver the state information, and the primary inputs to compute the primary outputs. (In general, the output decoder uses some sub-sets of the state variables and input variables to compute a particular primary output). Naturally, as in the case of any combinational function, the output encoder block can be viewed as a special case of sequential machine with trivial state behavior. This way, it is subject to further decomposition according to General Decomposition Theorem. In particular, it can be decomposed into a set of binary functions (combinational partial machines), each producing a single binary primary output of the original machine.

The last element of general decomposition is the input encoder block Ψ. Since state assignment does not introduce any input encoding, the role of the input encoder is limited to distribution of the binary inputs to the particular partial machines and the output decoder sub-circuits. The connection between the input encoder and the output decoder blocks is realized by a trivial combinational partial machine that simply transfers inputs received from the input encoder block to the output decoder block (identity function). Note that for a Moore machine, where the output only depends on the current state, this trivial machine is not needed, as the output decoder does not need any primary input information.

The resulting decomposition structure is shown in Fig. 4.3. The partial machine M1 is the sequential machine associated with the first encoding bit. Therefore, the machine has two states: '0' for the original machine's states a and d and '1' for states b, c and e (see encoding table in Fig. 4.1). Thus, its state space is described by the set system $\pi_S^1 = \{\overline{a,d}; \overline{b,c,e}\}$. The second state-bit-machine, M2, has the state space defined by the second encoding bit — it is in state '0' for the states c and d and in state '1' for the states a, b and e. Therefore, $\pi_S^2 = \{\overline{c,d}; \overline{a,b,e}\}$.

Since in general state assignment does not introduce any encoding of the inputs or state information flowing between the machines, the role of both the input encoder block Ψ and the inter-machine connection blocks $Con_{i,j}$ is reduced to distribution of the binary input and state variables. The selection of particular binary signals to distribute to partial machines is determined by the input supports of the combinational functions associated with each of the machines. These input supports can be derived from the encoded form

$pi_1 pi_2$	ps	ns	$po_1 po_2$
00	a	d	01
01	a	a	00
10	a	c	11
11	a	b	10
00	b	a	11
01	b	d	10
10	b	a	11
11	b	d	10
00	c	d	11
01	c	a	11
10	c	d	11
11	c	a	11
00	d	a	01
01	d	d	01
10	d	e	11
11	d	c	11
00	e	a	11
01	e	d	10
10	e	a	11
11	e	d	10

(a) Transition table

state	code
a	01
b	11
c	10
d	00
e	11

(b) Encoding

pi_1	pi_2	sb_1	sb_2	sb_1	sb_2	po_1	po_2
0	0	0	1	0	0	0	1
0	1	0	1	0	1	0	0
1	0	0	1	1	0	1	1
1	1	0	1	1	1	1	0
0	0	1	1	0	1	1	1
0	1	1	1	0	0	1	0
1	0	1	1	0	1	1	1
1	1	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	1	1
1	0	1	0	0	0	1	1
1	1	1	0	0	1	1	1
0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	1
1	0	0	0	1	1	1	1
1	1	0	0	1	0	1	1
0	0	1	1	0	1	1	1
0	1	1	1	0	0	1	0
1	0	1	1	0	1	1	1
1	1	1	1	0	0	1	0

(c) Encoded machine

Figure 4.1. Example finite state machine

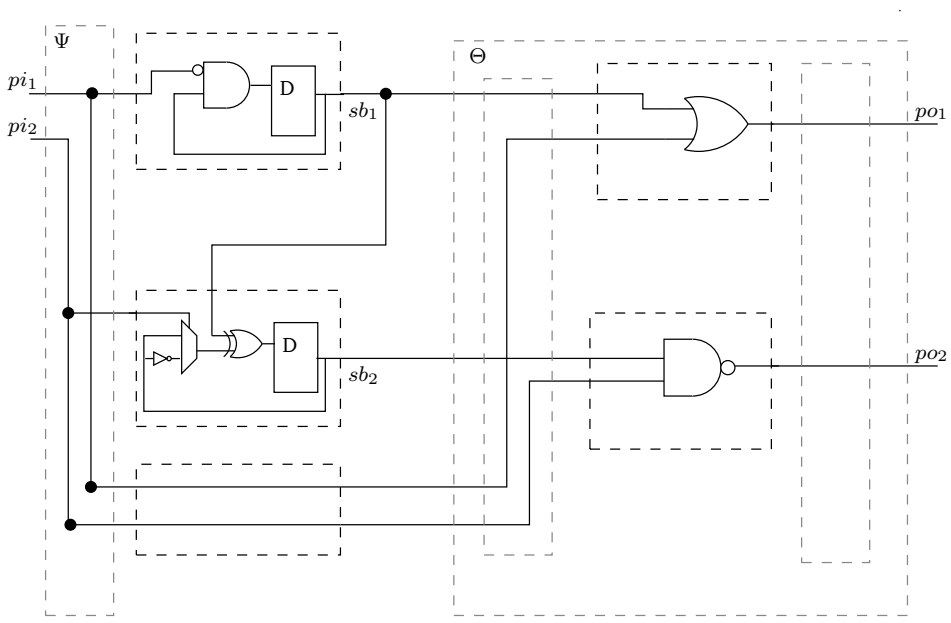


Figure 4.2. Circuit implementing the encoded machine

of the finite state machine, such as the one presented in Fig. 4.1(c). Using any input support minimization method we can determine that the supports of the binary outputs are as follows:

$$sb_1 : \{pi_1, sb_1\}$$
$$sb_2 : \{pi_2, sb_1, sb_2\}$$
$$po_1 : \{pi_1, sb_1\}$$
$$po_2 : \{pi_2, sb_2\}$$

We may further differentiate between the primary inputs and state inputs in the input support :

$$\begin{aligned} sb_1 &: \{IS_1, SS_1\} = \{\{pi_1\}, \{sb_1\}\} \\ sb_2 &: \{IS_2, SS_2\} = \{\{pi_2\}, \{sb_1, sb_2\}\} \\ po_1 &: \{IS_3, SS_3\} = \{\{pi_1\}, \{sb_1\}\} \\ po_2 &: \{IS_4, SS_4\} = \{\{pi_2\}, \{sb_2\}\} \end{aligned} \tag{4.1.1}$$

This way we determine that the primary input information delivered to M1 by the Ψ block is equal to the set system induced on the input symbols by the first input bit, i.e. $\pi_I^1 = \{\overline{00,01}; \overline{10,11}\}$. In general, the primary input information available to a partial

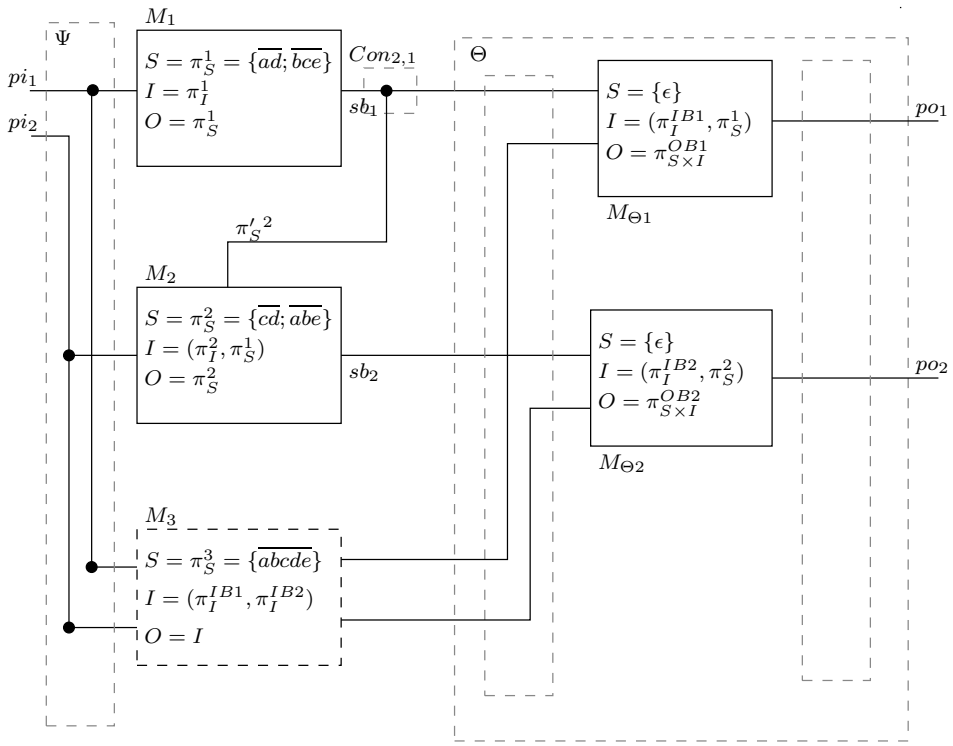


Figure 4.3. Encoding-induced decomposition

machine can be described by the product of set systems induced by the primary inputs in the input support of the state bit associated with this machine, i.e.

$$\pi_I^i = \prod_{pi_j \in IS_i} \pi_I^{IBj} \tag{4.1.2}$$

where:
– IS_i are the primary inputs in the support of the state bit i (primary support)
– π_I^{IBx} is the set system induced on the input symbols by the primary input bit x.

Similarly, the state information imported by a particular partial machine is a product of the state set systems associated with the state bits in the state input support of the partial machine, i.e.

$$\pi_S'^{\,i} = \prod_{sb_j \in SS_i \setminus \{sb_i\}} \pi_S^j \tag{4.1.3}$$

where SS_i are the state inputs in the support of the state bit i (state support)

In the example above, the state support of M2 contains sb_1, and therefore the state information imported by M2 is $\pi_S'^{\,2} = \pi_S^1 = \{\overline{ad}, \overline{bce}\}$.

This example brings us to another specific feature of the encoding-induced decomposition. Since the output of a partial sequential machine is in fact an encoding bit, it only conveys state information, without any partial primary input information. Therefore, instead of describing the output of a partial machine by $\pi_{S\times I}^i$, as in General Decomposition Theorem, we can use simplified representation by the state set system of the partial machine π_S^i.

Let us consider now the combinational machines $M_{\Theta 1}$ and $M_{\Theta 2}$ producing the primary outputs. As already mentioned, they are the result of decomposition of the combinational output decoder block Θ. As for any combinational machine, the state space of Θ is a single state that does not change, and therefore the state behavior of the output machines is trivial and can be omitted. The input of Θ is composed of state bits (outputs of sequential partial machines M1 and M2) and the primary inputs forwarded by the trivial machine M3. These inputs are distributed without any encoding to the output machines according to the input supports of the primary output functions. Therefore, as in the case of sequential machines, the input information of an output machine has two components: the primary input component being the combination of primary input information delivered by the primary input bits in the input support of the binary output function computed by the machine, and the state component being the combination of state information delivered by the state bits in the input support of the binary output function. Since the output of each output machine is a primary binary output, the output information of the machine is described by the set system on the state/input space induced by the corresponding binary output variable ($\pi_{S\times I}^{OBi}$).

Impact on General Decomposition Theorem

The structure described in the previous section is a special case of the general decomposition of the symbolic machine. Therefore, as for any other decomposition, the conditions imposed by the General Decomposition Theorem on the set systems describing the

information flows within the decomposition structure have to hold. Now, we can formulate the conditions imposed by the General Decomposition Theorem on the set systems defining the encoding. Let us analyze how the above mentioned specific features of the encoding-induced decomposition influence the specific formulation of General Decomposition Theorem.

Partial machines In our analysis we have to distinguish between the two types of machine present in the encoded network: the sequential partial machines (state-bit machines) implementing the next-state function, and the combinational partial machines (output machines) implementing the output decoder, i.e. the output function. We will omit the trivial machine that forwards the input information to the Θ block (M3 in Fig.4.3), as it does not perform any information processing, and therefore does not introduce any new set systems.

The most significant impact on the formulation of General Decomposition Theorem is associated with the fact that the trinity of partitions $(\pi_I^i, \pi_S^i, \pi_{S\times I}^i)$ is no longer necessary to define a partial machine. In the case of state-bit machines, their input information is unambiguously defined by their input supports, and the state information is defined by the state partition. The output is equal to the state partition, and so $\pi_{S\times I}^i$ is reduced to π_S^i. A state-bit machine can be therefore described by a pair $\{(IS_i, SS_i), \pi_S^i\}$. For each output machine, its input information is also defined by its input support, and its output information is induced by the corresponding primary output variable. Thus, an output machine can be described by a pair $\{(IS_o, SS_o), \pi_{S\times I}^{OBo}\}$.

Input information of a partial machine This influences the expression for the total input and state information available to a partial machine ($\pi_{S\times I}^{in\ i} = \pi_{S\times I}^{S\ i} \cdot \pi_{S\times I}^{I\ i} \cdot \pi'_{S\times I}{}^{i}$). For state-bit machines, the last factor in this expression (the imported information) becomes now the input/state information induced by the imported state information, i.e.

$$\pi'_{S\times I}{}^{i} = ind_{S\times I}^{S}\left(\pi'_{S}{}^{i}\right) = ind_{S\times I}^{S}\left(\prod_{sb_j \in SS_i} \pi_S^j\right) \tag{4.1.4}$$

This expression contains the information about the machine's own state if it is used by the machine (π_S^i if $sb_i \in SS_i$), so the first factor of the input information $\pi_{S\times I}^{S\ i} = ind_{S\times I}^{S}(\pi_S^i)$ is covered by the third and can be removed.

The output machines do not import information from each other, but to introduce a uniform notation we may treat the state-bit inputs in a support of an output machine as an imported information. Then, the imported part of the input information is described with the same expression as for state-bit machines. The output machines do not have any own state information, so the first factor of input information can also be removed.

The second factor, the state/input information delivered by primary inputs can be described for both state-bit and output machines by

$$\pi_{S\times I}^{I\ i} = ind_{S\times I}^{I}(\pi_I^i) = ind_{S\times I}^{I}\left(\prod_{pi_j \in IS_i} \pi_I^{IBj}\right) \tag{4.1.5}$$

In summary, the total state/input information available to either state-bit or output partial machine is

$$\pi_{S\times I}^{in}{}^{i} = ind^{I}_{S\times I}\left(\prod_{pi_j\in IS_i}\pi_I^{IBj}\right)\cdot ind^{S}_{S\times I}\left(\prod_{sb_j\in SS_i}\pi_S^{j}\right) \tag{4.1.6}$$

This expression can be used to formulate the specific modified conditions (1) and (2) of the General Decomposition Theorem.

In the example we follow, the state information available to M1 is determined by its state input support ($\{sb_1\}$, see Eq.4.1.1) and is equal to π_S^1. The primary input information is determined by the primary input support ($\{i_1\}$) of the state bit associated with the machine and is equal to π_I^{IB1}. Therefore, the total input information available to M1 is

$$\begin{aligned}
\pi_{S\times I}^{in}{}^{1} &= ind^{I}_{S\times I}\left(\pi_I^{IB1}\right)\cdot ind^{S}_{S\times I}\left(\pi_S^1\right) = \\
&= ind^{I}_{S\times I}\left(\{\overline{00,01};\overline{10,11}\}\right)\cdot ind^{S}_{S\times I}\left(\{\overline{ad};\overline{bce}\}\right) \\
&= \{\overline{(a,00)(a,01)(b,00)(b,01)(c,00)(c,01)(d,00)(d,01)(e,00)(e,01)};\\
&\quad \overline{(a,10)(a,11)(b,10)(b,11)(c,10)(c,11)(d,10)(d,11)(e,10)(e,11)}\}\\
&\quad \cdot\{\overline{(a,00)(a,01)(a,10)(a,11)(d,00)(d,01)(d,10)(d,11)};\\
&\quad \overline{(b,00)(b,01)(b,10)(b,11)(c,00)(c,01)(c,10)(c,11)(e,00)(e,01)(e,10)(e,11)}\}\\
&= \{\overline{(a,00)(a,01)(d,00)(d,01)};\overline{(a,10)(a,11)(d,10)(d,11)};\\
&\quad \overline{(b,00)(b,01)(c,00)(c,01)(e,00)(e,01)};\\
&\quad \overline{(b,10)(b,11)(c,10)(c,11)(e,10)(e,11)}\}
\end{aligned} \tag{4.1.7}$$

Machine M2 uses primary input $\{i_2\}$ and both state bits, and so its input information is

$$\begin{aligned}
\pi_{S\times I}^{in}{}^{2} &= ind^{I}_{S\times I}\left(\pi_I^{IB2}\right)\cdot ind^{S}_{S\times I}\left(\pi_S^1\cdot\pi_S^2\right) = \\
&= \{\overline{(a,00)(a,10)};\overline{(a,01)(a,11)};\overline{(c,00)(c,10)};\overline{(c,01)(c,11)};\\
&\quad \overline{(b,00)(b,10)(e,00)(e,10)};\overline{(b,01)(b,11)(e,01)(e,11)};\overline{(d,00)(d,10)};\\
&\quad \overline{(d,01)(d,11)};\}
\end{aligned} \tag{4.1.8}$$

For machines $M_{\Theta 1}$ and $M_{\Theta 2}$ input information is

$$\begin{aligned}
{\pi^{in}_{S\times I}}^{3} &= ind^{I}_{S\times I}\left(\pi^{IB1}_{I}\right)\cdot ind^{S}_{S\times I}\left(\pi^{1}_{S}\right) = \\
&= ind^{I}_{S\times I}\left(\{\overline{00,01};\overline{10,11}\}\right)\cdot ind^{S}_{S\times I}\left(\{\overline{ad};\overline{bce}\}\right) \\
&= \{\overline{(a,00)(a,01)(d,00)(d,01)};\overline{(a,10)(a,11)(d,10)(d,11)}; \\
&\quad \overline{(b,00)(b,01)(c,00)(c,01)(e,00)(e,01)}; \\
&\quad \overline{(b,10)(b,11)(c,10)(c,11)(e,10)(e,11)}\} \qquad (4.1.9) \\
{\pi^{in}_{S\times I}}^{4} &= ind^{I}_{S\times I}\left(\pi^{IB2}_{I}\right)\cdot ind^{S}_{S\times I}\left(\pi^{2}_{S}\right) = \\
&= \{\overline{(a,00)(a,10)(b,00)(b,10)(e,00)(e,10)}; \\
&\quad \overline{(a,01)(a,11)(b,01)(b,11)(e,01)(e,11)}; \\
&\quad \overline{(c,00)(c,10)(d,00)(d,10)};\overline{(c,01)(c,11)(d,01)(d,11)}\}
\end{aligned}$$

Output of the partial machines The condition (2) of the General Decomposition Theorem requires in general case that the output information of a machine can be computed from its input information. For state-bit machines, the output of a machine is its state, so a machine is capable of producing the output as long as it is capable of producing the state. Thus, condition (2) is covered by condition (1) and can be left out. For output-bit machines, the input information has to be sufficient to compute the binary primary output of the machine that is associated with the particular partial machine. This can be described as

$${\pi^{in}_{S\times I}}^{n+o} \leq \pi^{OBo}_{S\times I} \qquad (4.1.10)$$

The $n+o$ index reflects the numbering of the partial machines, such that the machines $1,\ldots,n$ (where n is the code length) are the state-bit partial machines, while $n+1,\ldots,n+l$ (where l is the number of primary output bits) are the output-bit partial machines.

Remaining conditions The condition (3), which prevents forming of combinational loops through imported information, can be removed as only state information is transferred between the machines and the feasibility of producing this information is already ascertained by condition (1).

The condition (4) of General Decomposition Theorem describes the conditions for the network to produce valid primary output of the realized machine. This is, however, already guaranteed by the validity of the output machines, i.e. condition (2). Therefore, condition (4) can be omitted.

The condition (5) of General Decomposition Theorem remains unaffected by the specific nature of the decomposition. For the network to realize the state behavior of the symbolic machine, the combination of state information delivered by particular machines still has to allow for unambiguous identification of the current state of the symbolic machine.

State assignment theorem As discussed above, the specific structure of the encoding-induced decomposition gives rise to specific reformulation of the conditions of the General Decomposition Theorem.

The particular form of partial machines and the structure of their interconnections allows to express the input information of the partial machines as a function of the supports of the binary functions realized by these machines. This influences the formulation of the conditions expressing the relationship of input and output information in the partial machine. Moreover, the distinction between the state-bit and output-bit partial machines allows to omit some of the conditions concerning state or output behavior. In particular, the output behavior of state-bit machines is the direct consequence of their state behavior, so the output-behavior condition (2) for these machines can be omitted. On the other hand, the output-bit machines are purely combinational, so the state-behavior condition (1) can be omitted for them.

Also, the conditions (3) and (4) turn out to be superseded by the other conditions, and thus can be removed. The only condition that remains essentially unaffected by the encoding-specific structure is the condition (5).

Taking into account all of the above, we can derive from the General Decomposition Theorem the conditions for a valid state assignment described by a set of two-block set systems.

Let $\mathbf{M} = \{I, S, O, \delta, \lambda\}$ be a completely specified finite state machine with a set of k binary inputs $PI_M = \{pi_1, \ldots, pi_k\}$ and a set of l binary outputs $PO_M = \{po_1, \ldots, po_l\}$. Let π_I^{IBi} for $i = 1, \ldots, k$ be a partition induced on the input alphabet symbols by the binary primary bit input number i and π_I^{OBo} for $o = 1, \ldots, l$ be a partition induced on the output alphabet symbols by the binary primary output bit number o. Let encoding $E_M = \{\pi_S^1, \ldots, \pi_S^n\}$ be a set of n two-block partitions on the set of states S of $\mathbf{M}$. Let $SB_M(E_M) = \{sb_1, \ldots, sb_n\}$ be a set of binary state variables, such that a variable sb_i assumes value 0 for states that are present in the first block of π_S^i and value 1 for states that are present in the second block of π_S^i.

Definition 4.1 (Support pair set). *A support pair set for a k-input, l-output machine $\mathbf{M}$ encoded with n-bit encoding E_M is a set of pairs (IS_i, SS_i) for $i = 1, \ldots, n, \ldots, n+l$, such that $IS_i \subseteq PI_M$, $SS_i \subseteq SB_M(E_M)$ and $IS_i \cup SS_i$ is an input support of the i^{th} binary next-state $(i = 1, \ldots, n)$ or output function $(i = n+1, \ldots, n+l)$ of the encoded $\mathbf{M}$.*

For a given encoding E_M and some support pair set $\{(IS_i, SS_i) \mid i = 1, \ldots, n+l\}$ for $\mathbf{M}$ encoded with E_M let

$$\pi_{S\times I}^{in\ i} = ind_{S\times I}^{I}\left(\prod_{pi_j \in IS_i} \pi_I^{IBj}\right) \cdot ind_{S\times I}^{S}\left(\prod_{sb_j \in SS_i} \pi_S^{j}\right) \tag{4.1.11}$$

denote the input information of a partial state-bit machine (for $i = 1, \ldots, n$) or a partial output-bit machine (for $i = n+1, \ldots, n+l$).

Theorem 4.1 (Encoding of completely specified FSM). *Let $\mathbf{M} = \{I, S, O, \delta, \lambda\}$ be a completely specified finite state machine with k binary inputs and l binary outputs. A set of two-block partitions $E = \{\pi_S^1, \ldots, \pi_S^n\}$ defines a valid n-bit encoding with output behavior realization of $\mathbf{M}$ if and only if a support pair set $\{(IS_i, SS_i) \mid i = 1, \ldots, n+l\}$ exists for $\mathbf{M}$ encoded with E such that*

(1) $\forall i \in \{1 \ldots n\}$ $(\pi_{S\times I}^{in\ i}, \pi_S^i)$ *is a* $(S \times I - S)$ *partition pair*

(2) $\forall o \in \{1 \ldots l\} \quad \pi_{S\times I}^{in\ o+n} \leq \pi_{S\times I}^{OBo}$

Additionally, if

(3) $\prod_{i\in\{1\ldots n\}} \pi_S^i = \pi_S(0)$

then the state behavior of M *will be realized too.*

To verify the conditions of the above theorem for the example machine, we first analyze the next states of the blocks of ${\pi_{S\times I}^{in}}^1$ (Eq.4.1.7). For elements of the first block $\overline{(a,00)(a,01)(d,00)(d,01)}$ the next states of the realized machine are d, a, a, d, respectively. Therefore, $\overline{\delta}(\overline{(a,00)(a,01)(d,00)(d,01)}) = \{a,d\}$ is a subset of the first block of π_S^1. Similarly, for the remaining blocks of ${\pi_{S\times I}^{in}}^1$ the next state sets are $\{b,c,e\}, \{a,d\}$, and $\{a,d\}$, respectively. Each of these next state sets is included in a block of π_S^1, so $({\pi_{S\times I}^{in}}^1, \pi_S^1)$ is a $(S\times I - S)$ pair. In the same manner it can be verified that $({\pi_{S\times I}^{in}}^2, \pi_S^2)$ is a $(S\times I - S)$ pair, and therefore the condition (1) is fulfilled.

Condition (2) requires that the input partitions of the output machines (${\pi_{S\times I}^{in}}^3$ and ${\pi_{S\times I}^{in}}^4$, see Eq. 4.1.9) are smaller or equal to the partitions induced by the values of the primary output bits. We can derive the induced partitions directly from the STT in Fig. 4.1

$$\begin{aligned}
\pi_{S\times I}^{OB1} = \{&\overline{(a,00)(a,01)(d,00)(d,01)};\\
&\overline{(a,10)(a,11)(b,00)(b,01)(b,10)(b,11)(c,00)(c,01)(c,10)(c,11)(d,10)(d,11)}\cdots\\
&\cdots\overline{(e,00)(e,01)(e,10)(e,11)}\}\\
\pi_{S\times I}^{OB2} = \{&\overline{(a,01)(a,11)(b,01)(b,11)(e,01)(e,11)};\\
&\overline{(a,00)(a,10)(b,00)(b,10)(c,00)(c,01)(c,10)(c,11)(d,00)(d,01)(d,10)(d,11)}\cdots\\
&\cdots\overline{(e,00)(e,10)}\}
\end{aligned} \tag{4.1.12}$$

and verify that indeed every block of ${\pi_{S\times I}^{in}}^3$ fits into a block of $\pi_{S\times I}^{OB1}$ and every block of ${\pi_{S\times I}^{in}}^4$ fits into a block of $\pi_{S\times I}^{OB2}$.

Finally, the product partition of the state partitions $\pi_S^1 \cdot \pi_S^2 = \{\overline{a}; \overline{be}; \overline{c}; \overline{d}\}$ is not a zero-partition, so the encoded machine only realizes the output behavior of the symbolic machine, without realizing its state behavior.

Multi-state incompletely specified encoding We can also apply the above reasoning to the General Decomposition Theorem for the incompletely specified, non-deterministic machines. In this case, the theorem reads as follows.

Let $\mathbf{M} = \{I, S, O, \delta, \lambda\}$ be an incompletely specified finite state machine with a set of k binary inputs $PI_M = \{pi_1, \ldots, pi_k\}$ and a set of l binary outputs $PO_M = \{po_1, \ldots, po_l\}$. Let ϕ_I^{IBi} for $i = 1, \ldots, k$ be a set system induced on the input alphabet symbols by the binary primary bit input number i and ϕ_I^{OBo} for $o = 1, \ldots, l$ be a set system induced on the output alphabet symbols by the binary primary output bit number o. Let encoding $E_M = \{\phi_S^1, \ldots, \phi_S^n\}$ be a set of n two-block set systems on the set of states S of $\mathbf{M}$. Let $SB_M(E_M) = \{sb_1, \ldots, sb_n\}$ be a set of binary state variables, such that a variable sb_i assumes value 0 for states that are present in the first block of ϕ_S^i, value 1 for states that are present in the second block of ϕ_S^i and any of the values 0 or 1 (denoted "–") for states that are present in both blocks of ϕ_S^i. Let $\mathcal{L}^i$ for $i = 1, \ldots, n$ be

a set of mappings of blocks of ϕ_S^i containing states of $\mathbf{M}$ to the blocks containing some copies of the same states in a multi-state realization $\mathbf{M}_l = \{I, S_l, O, \delta_l, \lambda_l\}$ of $\mathbf{M}$.

For a given encoding E_M and some support pair set $\{(IS_i, SS_i) \mid i = 1, \ldots, n+l\}$ for $\mathbf{M}$ encoded with E_M let

$$\phi_{S_l \times I}^{in\ i} = ind_{S_l \times I}^{I} \left(\prod_{pi_j \in IS_i} \phi_I^{IBj} \right) \cdot ind_{S_l \times I}^{S_l} \left(\prod_{sb_j \in SS_i} \mathcal{L}^j(\phi_S^j) \right) \tag{4.1.13}$$

denote the input information of a partial state-bit machine (for $i = 1, \ldots, n$) or a partial output-bit machine (for $i = n+1, \ldots, n+l$).

Theorem 4.2 (Encoding of incompletely specified FSM). *Let $M = \{I, S, O, \delta, \lambda\}$ be an incompletely specified non-deterministic finite state machine with k binary inputs and l binary outputs. A set of two-block set systems $E = \{\phi_S^1, ..., \phi_S^n\}$ defines a valid n-bit encoding with output behavior realization of M if and only if $n + l$ support pair sets (IS_i, SS_i) and n label assignments $\mathcal{L}^i : \phi_S^i \rightarrow \phi_{S_l}^i$ exist such that*

(1) $\forall i \in \{1 \ldots n\} \quad \left(\left[\phi_{S_l \times I}^{in\ i}\right]_i, \left[\phi_{S_l}^i\right]_i\right)$ *is a synchronized $(S_l \times I - S_l)$ cover pair*

(2) $\forall o \in \{1 \ldots l\} \quad L^{-1}(\phi_{S_l \times I}^{in\ o+n}) \leq \phi_{S \times I}^{OBo}$

Additionally, if

(3) $\left(\prod_{i \in \{1 \ldots n\}} \phi_{S_l}^i, \phi_S(0)\right)$ *is a $S_l - S$ cover pair*

then the state behavior of M will be realized too.

4.1.2 Mechanics of state assignment

The example in the previous section illustrated the specific structure of the encoding-induced decomposition. In this structure, a single partial machine is present for each of the binary next-state and output functions induced by the encoding. The partial machines are interconnected with binary wires, without any interconnection encoders.

Therefore, the problem of state encoding can be formulated as the problem of finding the specific decomposition of a symbolic sequential machine to partial machines with at most two states (two states for sequential machines and one state for combinational). Additionally, the output set systems of the combinational partial machines are constrained to the set systems induced by the primary output variables (output of a combinational machine is a binary primary output), and the output set systems of the sequential machines are constrained to the state set systems π_S^i (output of a sequential machine is a binary state bit). The set systems are subject to conditions formulated in Theorem 4.2.

This formulation is a generalization of the formulation used by the dichotomy-based state assignment methods, which search for a set of dichotomies defining particular encoding bits (equivalent to 2-block set systems defining the state spaces of the sequential partial machines). However, most current dichotomy-based methods require that the product of all dichotomies is equal to zero-partition. In other words, they require a unique code for each of the states. This requirement is equivalent to the condition (3) of Theorem 4.2, i.e. the state behavior realization. Meanwhile, usually it is not necessary to realize state behavior of a sequential machine as long as the output behavior is

realized. The condition (3) can be therefore dropped. Such relaxation of the conditions means that the encoded machine is only required to realize the output behavior of the symbolic machine, and therefore allows for implicit state minimization during state assignment. Under this formulation, very flexible state assignment is possible, allowing for state minimization and incompletely specified codes (state set systems with overlapping blocks).

In the following, we will describe mechanics of the construction of a valid state encoding of a finite state machine, which exploit the freedom afforded by the Theorem 4.2. This mechanics are independent of the target implementation platform, as they only consider validity of the encoding, without attempt at optimizing the implementation effectiveness or cost. On top of this mechanics, platform-specific heuristics have to be used to guide the encoding construction process towards efficient implementations.

State assignment construction

The goal of a state encoding method is the construction of a set of two-block set systems fulfilling the conditions of Theorem 4.2. In this process we propose to exploit the correspondence between a set system and its information set.

A symbolic FSM state variable has a single value to represent each state of the machine, e.g. the state variable of the example FSM in Fig. 4.1 has a separate value a, b, c, d or e for each of the machine's states. Therefore, if we denote the set of states $\{a, b, c, d, e\}$ as T, the symbolic state variable delivers the full information about T, described as a zero partition $\pi_T(0) = \{\overline{a}; \overline{b}; \overline{c}, \overline{d}, \overline{e}\}$ or an information set of all distinctions $IS(\pi_T(0)) = \{a/b, a/c, a/d, a/e, b/c, b/d, b/e, c/d, c/e, d/e\}$. After encoding, this information is delivered by a set of binary state variables, each of which can be represented by their corresponding two-block set system or information set. Therefore, the construction of the two-block set systems that define the encoding can be described as a problem of finding a set of two-block set systems that in product give the zero partition describing the information delivered by the symbolic encoding variable. Alternatively, it is equivalent to the problem of distribution of the distinctions from $IS(\pi_T(0))$ between the information sets of the encoding set systems.

The above formulation obeys the condition (3) of Theorem 4.2, which is required for state behavior realization. If we drop this requirement and only require output behavior realization of the encoded machine, the product of the encoding set systems does not need to be a zero partition. In the information set formulation it means that the sum of information sets of the encoding variables does not need to include all distinctions from $IS(\pi_T(0))$. Therefore, a valid encoding can be constructed by distribution of *some* of the state distinctions realized by the symbolic encoding variable between the information sets of the encoding variables in such way that the set systems corresponding to these information sets fulfill the conditions of Theorem 4.2.

For instance, to construct the encoding in Fig. 4.1(b), the distinctions from $IS(\pi_T(0))$ can be distributed over the two information sets $IS_1 = \{a/b, a/c, a/e, b/d, c/e, d/e\}$ and $IS_2 = \{a/c, a/d, b/c, b/d, c/e, d/e\}$. Their corresponding set systems are $\phi_S^1 = \{\overline{a, d}; \overline{b, c, e}\}$ and $\phi_S^2 = \{\overline{c, d}; \overline{a, b, e}\}$, which we recognize as the encoding set systems of the example machine. Note that the information sets need not be disjoint and that not all distinctions from $IS(\pi_T(0))$ are present in the union of IS_1 and IS_2. In particular, the distinction b/e is missing from the union, which manifests itself in the encoding by the

fact that the states b and e have the same code.

The construction procedure of the encoding set systems performs simultaneous selection of the distinctions for the information set and construction of the corresponding encoding set system. For this purpose, dichotomies (see Def. 2.22) are used. We recall that dichotomy is an equivalent shorthand notation for a two-block set system and therefore is suitable for encoding representation. The procedure starts with a set of atomic dichotomies. Atomic dichotomy represents elementary information and is equivalent to set systems delivering single distinction between two particular states. A set system delivering a single distinction is a set system that has the information set containing exactly one elementary information (distinction).

Take for instance the FSM in Fig. 4.4. It has four states a, b, c and d, and therefore all state distinctions are $\{a/b, a/c, a/d, b/c, b/d, c/d\}$. The set system delivering a single distinction a/b is $\phi_{a/b} = \{\overline{a, c, d}; \overline{b, c, d}\}$ and it has an equivalent atomic dichotomy a/b.

At this point the set of atomic dichotomies already defines a valid encoding (so called *atomic encoding*, see Fig. 4.5), as the union of their information sets gives the information set containing all state distinctions, and therefore the encoded machine is state and output behavior realization of the symbolic machine. However, the length of the encoding ($s \cdot (s-1)/2$, where s is the number of states) is likely to introduce prohibitive cost of the realization. Therefore, a more compact encoding is constructed by combining some atomic dichotomies what corresponds to combining information sets of the combined atomic dichotomies, i.e. by construction of encoding set systems with information sets containing more elementary information items (more state distinctions). Naturally, the more state distinctions a single encoding set system realizes, the less set systems are necessary to cover all required distinctions, and therefore the less the length of the encoding. The process of combining the information sets of particular dichotomies corresponds to merging of the dichotomies involved (see Def. 2.26). For example, joining informa-

i	ps	ns	o
0	a	c	1
1	a	b	0
0	b	c	1
1	b	a	0
0	c	a	0
1	c	d	1
0	d	c	1
1	d	b	1

Figure 4.4. Example FSM

dich	a/b	a/c	a/d	b/c	b/d	c/d
ϕ	$\{\overline{acd}; \overline{bcd}\}$	$\{\overline{abd}; \overline{bcd}\}$	$\{\overline{abc}; \overline{bcd}\}$	$\{\overline{abd}; \overline{acd}\}$	$\{\overline{abc}; \overline{acd}\}$	$\{\overline{abc}; \overline{abd}\}$
a	0	0	0	–	–	–
b	1	–	–	0	0	–
c	–	1	–	1	–	0
d	–	–	1	–	1	1

Figure 4.5. Atomic encoding of the example machine

tion sets of the atomic dichotomies a/c and c/d corresponds to merging these two dichotomies to a dichotomy $\{\overline{a,d};\overline{c}\}$ (or ad/c) with information set $IS(ad/c) = \{a/c, c/d\}$.

Additionally, assuming that only output behavior realization is required, the encoding can be compacted by removing some of the atomic dichotomies from the encoding, as long as the conditions (1) and (2) of the Theorem 4.2 are fulfilled by the remaining set of dichotomies.

For instance, the dichotomy a/b can be removed from the atomic encoding without violating the first two conditions of the Theorem 4.2. The remaining dichotomies are then ad/c, a/d, b/c and b/d, and the corresponding encoding set systems: $\phi_S^1 = \{\overline{abd};\overline{bc}\}$, $\phi_S^2 = \{\overline{abc};\overline{bcd}\}$, $\phi_S^3 = \{\overline{abd};\overline{acd}\}$ and $\phi_S^4 = \{\overline{abc};\overline{acd}\}$ define four partial state machines associated with four state bits $\{sb_1, sb_2, sb_3, sb_4\}$. After encoding the FSM with the encoding defined by these set systems (see Fig. 4.6), input supports of the state bits and output are as follows

$$
\begin{aligned}
sb_1 &: \{IS_1, SS_1\} = \{\{i\}, \{sb_1, sb_3\}\} \\
sb_2 &: \{IS_2, SS_2\} = \{\{i\}, \{sb_3\}\} \\
sb_3 &: \{IS_3, SS_3\} = \{\{i\}, \{\}\} \\
sb_4 &: \{IS_4, SS_4\} = \{\{\}, \{sb_1\}\} \\
o &: \{IS_5, SS_5\} = \{\{i\}, \{sb_1, sb_2, sb_3, sb_4\}\}
\end{aligned}
$$

Let us assume that encoding does not introduce any multi-state realization for any of the states of the symbolic FSM. Then, all mappings $\mathcal{L}^i$ are identity functions and all input and output set systems in Theorem 4.2 are on $S \times I$ instead of $S_l \times I$. The input

dich	$sb_1 : ad/c$	$sb_2 : a/d$	$sb_3 : b/c$	$sb_4 : b/d$
ϕ	$\{\overline{abd};\overline{bc}\}$	$\{\overline{abc};\overline{bcd}\}$	$\{\overline{abd};\overline{acd}\}$	$\{\overline{abc};\overline{acd}\}$
a	0	0	–	–
b	–	–	0	0
c	1	–	1	–
d	0	1	–	1

Figure 4.6. Modified encoding of the example machine

set systems of the four state bit machines and one output machine are then

$$\begin{aligned}
{\phi^{in}_{S\times I}}^1 &= ind^I_{S\times I}\left(\phi^{IB1}_I\right)\cdot ind^S_{S\times I}\left(\phi^1_S\cdot\phi^3_S\right)\\
&= ind^I_{S\times I}\left(\{\overline{0};\overline{1}\}\right)\cdot ind^S_{S\times I}\left(\{\overline{abd};\overline{bc}\}\cdot\{\overline{abd};\overline{acd}\}\right)\\
&= ind^I_{S\times I}\left(\{\overline{0};\overline{1}\}\right)\cdot ind^S_{S\times I}\left(\{\overline{abd};\overline{c}\}\right)\\
&= \{\overline{(a,0)(b,0)(c,0)(d,0)};\overline{(a,1)(b,1)(c,1)(d,1)}\}\cdot\\
&\quad\cdot\{\overline{(a,0)(a,1)(b,0)(b,1)(d,0)(d,1)};\overline{(c,0)(c,1)}\}\\
&= \{\overline{(a,0)(b,0)(d,0)};\overline{(c,0)};\overline{(a,1)(b,1)(d,1)};\overline{(c,1)}\}\\
{\phi^{in}_{S\times I}}^2 &= \{\overline{(a,0)(b,0)(d,0)};\overline{(a,0)(c,0)(d,0)};\overline{(a,1)(b,1)(d,1)};\overline{(a,1)(c,1)(d,1)}\}\\
{\phi^{in}_{S\times I}}^3 &= \{\overline{(a,0)(b,0)(c,0)(d,0)};\overline{(a,1)(b,1)(c,1)(d,1)}\}\\
{\phi^{in}_{S\times I}}^4 &= \{\overline{(a,0)(a,1)(b,0)(b,1)(d,0)(d,1)};\overline{(b,0)(b,1)(c,0)(c,1)}\}\\
{\phi^{in}_{S\times I}}^5 &= \{\overline{(a,0)(b,0)};\overline{(c,0)};\overline{(d,0)};\overline{(a,1)(b,1)};\overline{(c,1)};\overline{(d,1)}\}
\end{aligned} \tag{4.1.14}$$

It can be verified that the next state sets of the blocks of ${\phi^{in}_{S\times I}}^1$ are $\{c\}$, $\{a\}$, $\{a,b\}$ and $\{d\}$, respectively, which fit into blocks 2,1,1, and 2 of ϕ^1_S, respectively. ${\phi^{in}_{S\times I}}^1$ and ϕ^1_S are therefore a $S\times I - S$ cover pair. Since there is no non-determinism present in the FSM, this is sufficient to verify the condition (1) of Theorem 4.2 for $i=1$. The verification for $i=2,3,4$ is left as an exercise. To verify condition (2), we have to show that $L^{-1}({\phi^{in}_{S\times I}}^5)\leq\phi^{OB1}_{S\times I}$. This is true, since

$$\phi^{OB1}_{S\times I}=\{\overline{(a,1)(b,1)(c,0)};\overline{(a,0)(b,0)(c,1)(d,0)(d,1)}\} \tag{4.1.15}$$

and L^{-1} is the identity function. Thus, we have verified that removing atomic dichotomy a/b from the set of encoding dichotomies, and therefore assigning overlapping codes to states a and b, still preserves the output behavior realization of the original machine.

Further, dichotomy ad/c can be merged with b/c and a/d with b/d, and thus the final encoding defined by two dichotomies abd/c and ab/d is constructed (Fig. 4.7).

Additional considerations

The method of combining information sets of the encoding dichotomies, or merging the dichotomies, offers a significant flexibility in the construction of the encoding. In fact, using this method, any valid non-redundant encoding of the FSM can be constructed.

dich	$sb_1: abd/c$	$sb_2: ab/d$
ϕ	$\{\overline{abd};\overline{c}\}$	$\{\overline{abc};\overline{cd}\}$
a	0	0
b	0	0
c	1	–
d	0	1

Figure 4.7. Final encoding of the example machine

However, the merging of information sets is not entirely arbitrary. There are two important aspects that have to be considered: compatibility and implied information (distinctions).

The **compatibility** consideration has to do with the fact that the merged information set has to correspond to a two-block set system to have a valid dichotomy representation and define a valid encoding bit. However, for some sets of information (distinctions) it is impossible to build a corresponding two-block set system. Consider, for example, the information set $IS_{acd} = \{a/c, a/d, c/d\}$. The corresponding set system with the minimum of blocks $\{\overline{ab}; \overline{bc}; \overline{bd}\}$ has three blocks rather than two, and therefore it does not define a single valid binary encoding variable. In the case of code construction by pairwise merging of the encoding dichotomies this problem is alleviated by considering dichotomy compatibility. If two dichotomies are compatible (Def. 2.24) they can be merged to a single dichotomy and therefore their combined information set is guaranteed to have a corresponding two-block set system. If we consider a pair of dichotomies a/cd and c/d, with corresponding information sets $\{a/c, a/d\}$ and $\{c/d\}$, the compatibility test reveals that both blocks of dichotomy c/d intersect the right-block of dichotomy a/cd, and therefore the two dichotomies are incompatible and their combined information set $\{a/c, a/d, c/d\}$ does not have a corresponding two-block set system. If, on the other hand, the second dichotomy was a/b, the pair of dichotomies a/cd and a/b would be compatible and the combined information set $\{a/b, a/c, a/d\}$ would have a corresponding two-block set system $\{\overline{a}; \overline{bcd}\}$. The compatibility test fails however for the sets of dichotomies larger than two. For example, the three atomic dichotomies defined by the distinctions in IS_{acd} cannot be merged together, even though they are pairwise-compatible. The problem of compatibility of an arbitrary set of dichotomies is in general equivalent to graph coloring, and therefore it is NP-hard.

The second consideration when merging dichotomies and their information sets is the **implied information** (implied distinctions). The implied information consists of all elementary information items in the information set of a dichotomy that were not present in the information sets of the dichotomies that were merged to obtain this dichotomy. Consider, for example, two dichotomies a/b and c/d. When merged, they result in dichotomy ac/bd (or ad/bc), which has information set $\{a/b, a/d, b/c, c/d\}$. Since the only elementary distinctions present in the original information sets were a/b and c/d, the remaining two distinctions a/d and b/c are implied. The implied information is the result of the requirement that the information set of a dichotomy has to have a corresponding two-block set system, the same requirement that introduced the compatibility problem. In the above example, naturally it is possible to just merge the information sets of the dichotomies, but the set system corresponding to the resulting information set $\{a/b, c/d\}$ is $\{\overline{ac}; \overline{ad}; \overline{bc}; \overline{bd}\}$ – a four-block set system, which does not have a dichotomy representation and does not define a single valid binary encoding bit.

As a result of the implied information, the encoding bits defined by the merged dichotomies may deliver elementary distinctions that they were not intended to deliver. In some cases this may be beneficial to the encoding process. In particular, this means that there is more information compressed on a single binary state variable, which may minimize both interconnections and input supports of the functions using this variable. This is the case when the implied information is relevant, i.e. it can be used by the functions using this variable. However, if the implied information is redundant, it may actually complicate implementation of the functions using this variable. Then, the implementa-

tion networks of the functions have to "sort out" the redundant information, which may result in more complex network structures. Also, the fact that a state variable delivers some extra information may have influence on the information that is necessary to compute this variable. In particular, the fact that a variable delivers more information may mean that more information is needed to compute this variable. In this way the implementation of the function that computes this variable may be more complicated and less efficient.

Summary

In this section, we have outlined a general method that constructs a state assignment of an FSM. The method uses dichotomies to represent the encoding. It starts with the dichotomies representing the encoding in which each state variable delivers a single elementary distinction between the states. In the code construction process, the encoding dichotomies are merged together to compress the state information delivered by the merged variables on a single variable. To ascertain validity of the constructed encoding, the method uses correspondence between dichotomies and two-block set systems. This correspondence allows to check the validity of the encoding defined by the dichotomies (and their corresponding set systems) against the conditions of Theorem 4.2.

The fact that Theorem 4.2 covers all valid state assignments, including state assignments reducing (for output-only behavior realization) and expanding (for multi-state realizations) the state space of machine, allows the method to explore the solution space unavailable to most other state assignment methods. Starting from the atomic encoding and using dichotomy merging to modify it makes it possible to reach any point of this solution space and provides fine-grained control over the form of constructed encoding. As a result, the method is extremely flexible and provides any control algorithm guiding it towards high-quality solution with a great deal of information about the structure of the FSM and information flows within it. In this thesis, we will show how the flexibility of the method and the information provided by it can be utilized as the basis for effective and efficient target-specific state assignment method.

4.2 Combinational synthesis

In the previous sections we described how the General Decomposition Theorem defines conditions that must be satisfied by a valid state assignment of a finite state machine. In this section, we will briefly discuss another special case of general decomposition and its application to combinational logic synthesis. The method, authored by Artur Chojnacki and Lech Jóźwiak and implemented in the software tool IRMA2FPGA is described in detail in [41].

A combinational function can be interpreted as a special case of finite state machine with a single state and trivial state transition function. In this interpretation, the General Decomposition Theorem can be used to describe a decomposition of the function into a network of partial combinational functions. The synthesis method is based on the recursive application of the serial decomposition scheme, in which at each step the considered function is decomposed into two sub-functions (partial combinational machines): *predecessor sub-function* g and *successor sub-function* h (see Fig. 4.8). The selection of function g is driven by the choice of the input and output set systems of the function. At the

same time, the objectives and constraints of the target implementation architecture are taken into account and g functions are built that fulfill these constraints and optimize the objectives. Therefore, the decomposed network is directly mappable, as it is composed of valid and optimized technology primitives. This approach of combination of global information processing optimization with platform-specific local optimization decisions gives superior synthesis results, as reported in [41].

In the combinational network built by the method no state information is present. Therefore, the set systems describing input and output information of the partial machines (functions) are reduced from the set $S \times I$ to I, i.e. only information about the primary input symbols is present and processed by the network. Therefore, the input information available to a partial machine is composed of the primary input information ϕ_I^i augmented by the information imported from other machines (${\phi'_{S\times I}}^i$), which in a combinational network become ${\phi'_I}^i$. Since the network is built bottom-up (from primary inputs towards the outputs) and no loops are allowed in the circuit, the signals available as inputs to a particular g function being built are the primary inputs and the outputs of the blocks built earlier (at the lower levels of the circuit). All these signals can be treated uniformly as delivering information about inputs described by set systems, and thus the input set system of the function ϕ_I^{ini} can be determined as the product of the set systems associated with each of the variables in the bound set. The output of a partial machine is described in General Decomposition Theorem by the output set system $\phi_{S\times I}^i$. Again, due to lack of state information, the symbol set of this set system is reduced from $S \times I$ to I, resulting in the output set system ϕ_I^{outi}. The structure of an example combinational decomposition is shown in Fig. 4.9.

Note that a truth-table (or PLA) description of a Boolean function specifies output values of the function for different input combinations. Each of such (not necessarily disjoint) input combinations (referred to as *term* or *cube*) can be treated as a separate symbol from the input alphabet I of the function and, therefore, the above mentioned set systems used to analyze the information flows in the function can be defined on the set of terms of the function. This approach is illustrated by the example of functional decomposition of a Boolean function presented in Section 2.4.1.

In [41], the conditions for existence of serial decomposition are formulated as a special case of the General Decomposition Theorem. Let f be a k-input, n-output incompletely specified Boolean function. Let $X = \{x_i \mid i = 1 \ldots k\}$ be the set of binary input variables

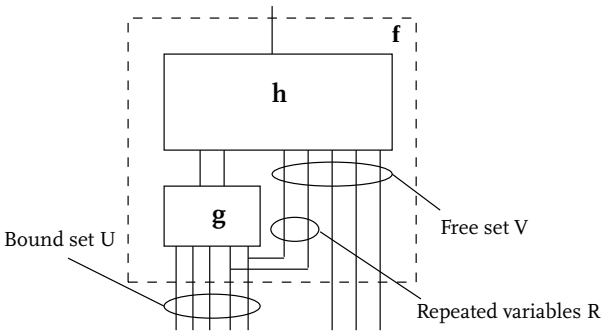


Figure 4.8. Serial decomposition

of f and ϕ_{x_i} ($i = 1 \ldots k$) be the set systems induced by the particular input variables x_i on the set of the terms (cubes) of function f. Let $Y = \{y_i \mid i = 1 \ldots n\}$ be the set of binary output variables of f and ϕ_{y_i} ($i = 1 \ldots n$) be the set systems induced by the particular output variables y_i on the set of the terms of f. Let $\phi_Y = \bullet_{i=1 \ldots n} \phi_{y_i}$ be the product set system induced by all binary output variables on the set of the terms of f. Let U and V be the subsets X, referred to as bound set and free set, respectively, and the set systems $\phi_U = \bullet_{x_i \in U} \phi_{x_i}$ and $\phi_V = \bullet_{x_i \in V} \phi_{x_i}$ be the set systems on the terms of f induced by the input variables in U and V, respectively.

Theorem 4.3 (Existence of serial decomposition [41]). *If there exists a set system ϕ_g on the set of terms of f, such that $\phi_U \leq \phi_g$ and $\phi_g \cdot \phi_V \leq \phi_Y$, then the function f has a serial functional decomposition with respect to (U, V) in the form $f = h(g(U), V)$.*

This theorem is recursively applied to the function f and then sub-functions g and h until the entire network is composed of the functions directly implementable in the target architecture.

The construction of function g can be completed in the following three steps: selection of an adequate input support of g (bound-set selection); construction of multi-valued function g with this support (construction of the output set system ϕ_I^{outi}, or ϕ_g in the Theorem 4.3); and implementation of the multi-valued function as a set of binary function(s) (encoding). Based on information relationship analysis, a limited set of the promising bound sets U is selected. For each bound set, the input set system ϕ_I^{ini} is calculated as a product of the set systems associated with the variables in the bound set. Then, the output set system is built. The condition (2) of General Decomposition Theorem requires that the output set system of a partial machine is larger than the input set system (i.e. provides less information). Therefore, any legal output set system can be constructed by

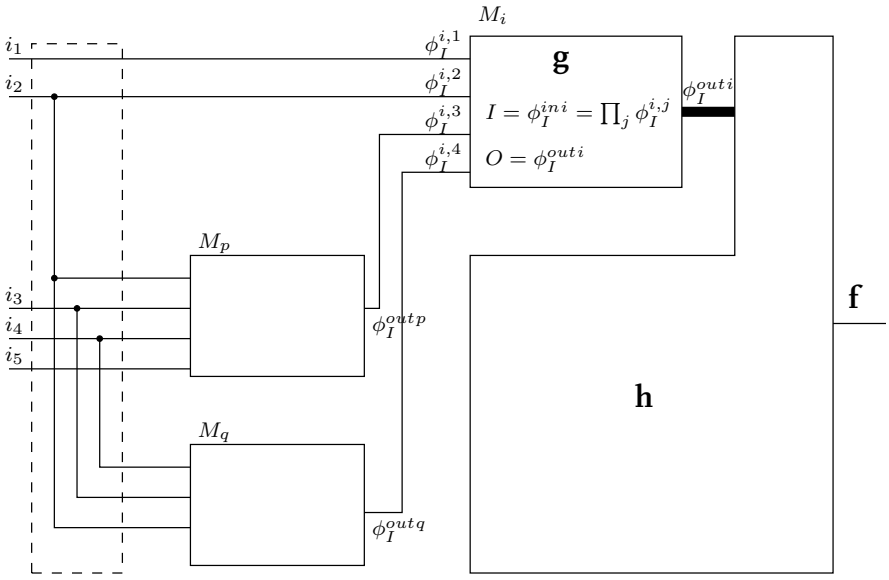


Figure 4.9. Combinational decomposition

merging the blocks of the input set system. In this method, the merging is performed in such a way as to minimize the number of blocks in the output set system (i.e. the number of output values of the multi-valued function g) without losing any unique distinctions (the distinctions only present on the inputs to the g block). The selection of the best bound set favors the bound sets, which result in the output set systems with the smallest number of blocks, as this number determines the number of binary outputs of the block ($\lceil log_2(|\phi_I^{out\,i}|)\rceil$), and therefore the size of implementation and the convergence of the block. In addition to convergence criterion, a number of information structuring criteria is used to break the ties.

The choice of the input support and the output set system defines a multi-valued function g with binary inputs. To implement this function in hardware, it has to be transformed into a set of binary function, i.e. the output values of the function have to be assigned with binary codes. As in the case of state assignment, the problem of encoding can be expressed as the problem of finding a set of two-block set systems that in product give the multi-block set system to encode (${\phi_I}_i^{out}$). In this process the sets of two-block set systems are favored that repeat the least available ("unique") information from the bound set on as many binary output variables as possible. In this way, the further decomposition of function h is facilitated by eliminating the constraints associated with unique information transmission.

4.3 Conclusions

In this chapter, we have discussed some special cases of the General Decomposition Theorem that were used in the research reported in this thesis. In particular, using the conditions of the theorem, we derived conditions for a valid state assignment of FSM and based on that proposed a general, target-independent method of building an encoding. The method starts with the encoding defined by the elementary state information items (atomic dichotomies) and constructs a more compact encoding by merging compatible information items (dichotomies). The conditions formulated in this chapter for a valid encoding are more general than those used in other state assignment methods and therefore they allow to consider a larger number of valid encodings. In particular, the method allows considering compatible machines with reduced and expanded state sets (state minimization and multi-state realization, respectively). This potentially enables identification of efficient assignments unattainable for other methods.

The general encoding method provides mechanics for construction of a valid encoding. To ascertain that the valid encoding also results in an efficient implementation of the encoded machine on the target platform, platform-specific heuristics have to be used on top of this mechanics to guide the encoding construction process. In the next chapter, we will discuss such heuristics leading to the encodings resulting in efficient implementation in FPGA. We will present a complete FPGA-targeted state assignment method that is the result of implementing the heuristics in the framework of the general encoding method.

In this chapter we also showed that the functional decomposition of Boolean functions (combinational machines) is a special case of the general decomposition of sequential machines and outlined a combinational circuit synthesis method based on the functional decomposition. The combinational synthesis method is compatible with the

FPGA-targeting state assignment method we will propose in the next chapter, as they are both based on the same information-driven circuit synthesis paradigm and on the general decomposition theory. This makes this method particularly suitable for combinational synthesis of the Boolean functions resulting from the state assignment of FSM performed with the proposed state assignment method. Therefore, in our experiments with state assignment we will use the software tool implementing the combinational synthesis method, IRMA2FPGA, to implement the encoded FSMs.

Chapter 5

Effective and efficient state assignment for LUT-FPGAs

In Chapter 4 we outlined the general, target-independent mechanics of a state assignment procedure based on a bitwise code construction using atomic state information and dichotomies. We pointed out that this procedure does not include any target-specific criteria to arrive at an encoding resulting in an efficient FSM implementation. It only assures that the resulting encoding is a valid one. The target-specific criteria need to be applied by particular encoding methods using this general scheme.

In this chapter, we will describe the criteria and heuristics guiding the general code construction procedure to achieve encoding resulting in an efficient implementation of the FSM in look-up-table-based FPGA. We will show how these heuristics are implemented in a state assignment software tool – SECODE. We will present the outline and an example of the operations of the tool and then focus on some specific data structures and algorithms crucial to the effectiveness and efficiency of the tool. We will also propose a dedicated analysis procedure that will allow us to address a specific encoding strategy – one-hot encoding.

5.1 Introduction

The choice of particular encoding ultimately determines the number of flip-flops and the binary functions (combinational components of the partial machines), and hence the form and cost of the circuit's implementation. Therefore, the primary concern of any state assignment method is the creation of the encoding that will minimize the implementation cost (in terms of area, delay, power dissipation, etc.). To achieve this goal, state assignment needs to consider three important factors of the implementation process: the target implementation platform, the optimization objectives, and the combinational synthesis method that follows the encoding.

Clearly, the target implementation platform has decisive influence on the cost of the implementation. Depending on the platform different features of binary functions determine their implementation cost. For instance, for PLA structures the circuit area depends on the number and size of the product terms in the minimized sum-of-products

description of the binary function. PLA-targeting state assignment aims therefore on the encoding, which induces binary next-state and output functions described by small sum-of-products expressions. On the other hand, in LUT-based FPGAs the number or the size of terms do not play any role — a building block of this architecture – LUT or CLB – is capable of realizing *any* function with a fixed number of inputs (typically, 4 to 6). However, a totally different set of objectives and cost considerations may be applicable if the optimization objective changes, and in place of area, e.g. the delay or power is to be optimized.

In addition to platform characteristics, state assignment has to account for the method used to implement the combinational component on the target architecture. With combinational synthesis methods applying elaborate heuristics to find a near-optimal realization, state assignment cannot easily predict, how a particular binary function will be realized, and therefore its cost. Traditionally, state assignment methods deal with this uncertainty by taking into account main heuristics of a particular combinational synthesis method and searching for an encoding that results in binary functions *easy to implement* for the combinational synthesis method. JEDI is an example of this approach. It assumes that the combinational synthesis is based on extraction of common cubes in sum-of-products expression, and therefore creates encoding that maximizes potential for common cube extraction. Therefore, it is particularly suitable for use with division-based logic synthesis systems, such as SIS [71].

In our experience, however, SIS handles FPGA synthesis inadequately. Therefore, we are targeting logic synthesis with IRMA2FPGA — the logic synthesis method based on functional decomposition and outlined in Section 4.2. This tool significantly outperforms state-of-the-art academic and commercial tools, synthesizing circuits on average about 2 times smaller and 50% faster, and reaching up to 5 times area and 3 times delay reduction for some specific function classes (such as symmetric functions and functions with many don't-cares) [41].

5.2 State assignment heuristics

5.2.1 Heuristic outline

In the previous chapter, we explained that to find a valid state encoding for a symbolic finite state machine it is sufficient to find a set of n (for n-bit encoding) 2-block set systems (or dichotomies), and we formulated conditions on these set systems in Theorem 4.2. When building a set system describing an encoding bit we have to have in mind two important considerations: the production and consumption of the information associated with this bit. We recall from Section 4.1.1 that each state bit is realized by a partial two-state sequential machine. Each such state-bit-machine is implemented as a flip-flop driven by the output of a combinational circuit implementing certain Boolean function of some primary inputs and state bits. From the viewpoint of the production of the information for the particular encoding bit it is therefore vital that the related combinational logic can be efficiently implemented in the target architecture. On the other hand, from the viewpoint of the consumption of the information from a particular encoding bit, the bit is used as an input to some other partial machines. Therefore, the information delivered by this bit has influence on the implementation of other partial machines.

As mentioned earlier, the character of this influence heavily depends on the target

implementation platform. In the case of LUT-FPGAs there are two important factors in the implementation of a circuit: size of the input supports of the functions and the number and length of interconnections. Since LUT is capable of realizing any function of no more than a certain maximal number of variables, the size of the input support of a function becomes a much better indicator of the realization complexity than number of terms or literals. Additionally, the interconnections (especially long), which are relatively scarce and slow on the FPGA platform, are often the determining factor in the circuit's delay and area. Thus, to facilitate efficient implementation of a circuit, (long) interconnections and input supports of particular functions should be minimized.

Therefore, the heuristic of *optimized information flows* for the selection of encoding set systems was proposed. In this context, optimization of information flows means distribution of state information between the state variables aiming at optimization of production and consumption of this information. From the viewpoint of information production, it is advantageous to group together the elementary information items (distinctions) which require similar information for their production. Then, the state bit being the composition of these elementary information items will require relatively little and relatively similar input information. This results in a compact input support and little interconnections required by partial machine realizing this state bit. From the point of view of information consumption, the information (distinctions) used together, especially by many functions, should be delivered together on the same state bit. Then, the functions using this information require fewer input support bits and fewer connections.

To properly analyze production and consumption of particular atoms of state information, we begin with FSM encoded with the atomic encoding and calculate input supports of primary and state output functions in this encoding. The input supports of the functions supply us with detailed information about how a particular atomic information is produced (which primary inputs and atomic state variable are in the support of the output function corresponding to the atomic state bit associated with this information) and how it is used (which primary and state output functions use the input bit corresponding to the atomic state variable associated with this information). This analysis is a basis for the creation of clusters of atomic dichotomies that should be conveyed together by the same state variable for the benefit of state information production, distribution and consumption. Once the clusters of atomic dichotomies are identified, their relative importance or quality is estimated and the actual encoding is constructed by considering clusters of dichotomies in descending order of their quality and merging the dichotomies in the considered cluster together on the same encoding dichotomy, or, if this turns out to be impossible, on a smallest possible set of dichotomies.

5.2.2 Information flows example

To illustrate the ideas behind information flow optimization with an example, let us consider again the FSM in Fig. 4.1(a). Fig. 5.1 shows the graph of information flows between the particular elementary state distinctions and primary inputs and outputs. As discussed above, the graph was determined by encoding the FSM with atomic encoding and computing input supports of all atomic encoding variables and primary output variables. Analysis of the graph reveals the structure of the information flows. We see very strongly connected subgraphs of the graph marked A and B. Within and between these subgraphs there is a lot of state information exchanged. Moreover, both subgraphs use

the same primary input pi_1 and are used together to compute the primary output po_1. From this we conclude that if the atomic dichotomies in subgraphs A and B are merged together, the resulting code bit(s) will be very independent (they will only depend on one primary input and themselves) and they will provide compressed information to compute the primary output po_1. Clearly, this makes the dichotomies very good candidates for merging.

The fact that the coherent system of six dichotomies was split into two subgraphs A and B is the consequence of another information flow consideration. We can see from the graph that the three dichotomies in the subgraph A deliver state information to the other subgraph of the information flow graph — C. The subgraph C forms another coherent and relatively independent component of the graph. The dichotomies in the subgraph C depend on the same primary input pi_2 and on each other. They are also all used to compute the primary output po_2. Therefore, they are good candidates to form a separate subsystem. As a consequence, all external information delivered the potential subsystem (i.e. information produced in subgraph A) should be compressed to optimize its usage and minimize interconnections. Another motivation to consider subgraphs A and B

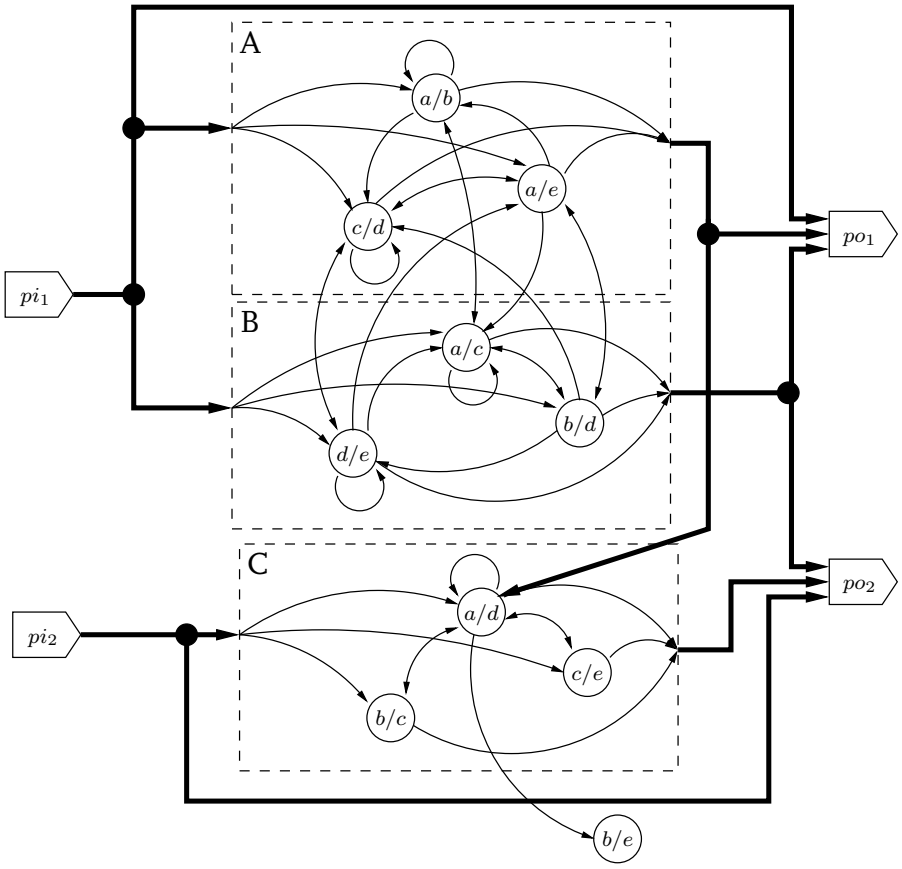


Figure 5.1. Information flows in the example FSM

separately is the fact that dichotomies in B are used to compute both primary outputs, while the dichotomies in A are only used by output po_1.

Finally, the isolated dichotomy b/e, which is not used by any other node in the graph, indicates that this particular state distinction is not necessary to compute any other state or output information, and thus suggests possible state space minimization.

Given the above analysis, we can reason about the most beneficial merging of the atomic dichotomies. The prime candidate for merging is the subgraph A. There are two possible ways of merging the dichotomies in A on one dichotomy – ac/bde or ad/bce. We will evaluate these two options with the information flows in mind. For that purpose, let us analyze the *complete* information delivered by the encoding dichotomies, i.e. the implied dichotomies (see Section 4.1.2) as well as the merged atomic dichotomies. For the encoding dichotomy ac/bde, the information set is $\{a|b, a|d, a|e, b|c, c|d, c|e\}$, while for ad/bce it is $\{a|b, a|c, a|e, b|d, c|d, d|e\}$. In the first case, we can see that the implied dichotomies ($a|d, b|c, c|e$) are not closely related (as far as information flows are concerned) to the dichotomies in the subgraph A. They use different input information and are used by different outputs. In the case of the second merging option, on the other hand, the implied dichotomies ($a|c, b|d, d|e$) are all contained in the subgraph B that is very closely related to A. Clearly, this is the preferred merging option.

When merging the dichotomies in subgraph C, we are also confronted with two merging choices: abe/cd or ac/bde. In this case both options introduce implied distinctions that are not closely related to the distinctions in C. In the first case, these are the distinctions in subgraph B, in the second case – the distinctions in subgraph A. At this point, we may consider the consumption of the resulting information by primary outputs. We may observe that the primary output po_2 that uses information produced in C also uses information from B. This indicates that if C is merged in such way that it implies B, the resulting state bit will deliver complete state information consumed by the primary output po_2.

5.2.3 Additional issues

In the process of identification and merging of dichotomy groups, we have to consider additional issues discussed in Section 4.1.2 – the compatibility problem and the implied dichotomies. The problem of implied dichotomies was already discussed and illustrated in the above example. The compatibility issue, on the other hand, means that we cannot assume that any set of distinctions (atomic dichotomies) can be grouped (merged) together and represented by a valid encoding dichotomy with two blocks. To circumvent this issue, set systems with more than two blocks can be initially allowed to define the sequential partial machines. In this case, the encoding is not a binary encoding – an encoding variable has as many values as there are blocks in the encoding set system. Returning to the analogy between encoding and decomposition, the partial state-bit-machines have more than two states (they have as many states as there are blocks in the set system) and the encoding becomes in this way a specific decomposition, which can be transformed to a valid encoding structure by encoding the partial non-binary sequential machines. This concession gives additional flexibility to the process of collecting related distinctions and therefore allows determination of a more natural, non-constrained structure of the finite state machine. The resulting larger partial machines form then "subsystems" processing and producing related information. This can be beneficial from the point of view of

minimizing long interconnections (much of the information is processed locally by the partial machines) as well as the potential for common sub-functions. The latter stems from the fact that such subsystems have several binary outputs requiring similar information, which provides room for common sub-functions.

Since the ultimate target remains the binary encoding, each of the non-binary partial machines has to be further encoded. However, if the initial decomposition does not depart too far from the target binary decomposition, i.e. the sequential partial machines are not too large, the encoding problem for the partial machines is much smaller and easier then the original encoding problem. This problem can be again formulated as the distribution of the distinctions present in the multi-block set system defining a particular partial machine among several two-block set systems. In this case, the distribution can be determined by the local analysis of the relations of the information (e.g. some distinctions may be "more related" than others).

5.3 The method

5.3.1 Outline

The goal of the information-driven state encoding that we propose is to assign binary codes to the symbolic states of a machine in such a manner that the resulting network of the binary output- and next-state functions is composed of small, coherent, and relatively independent parts, with particular binary next-state or output functions depending on a small number of inputs. To achive this goal, the state information is assigned to particular binary state variables in such a way that the information using particular inputs, as well as the information used by particular outputs is grouped on as few as possible state variables, thus reducing the cost of information production and distribution.

The binary state variables are represented using dichotomies. The state information delivered by a particular binary variable is given by the atomic dichotomies covered by the encoding dichotomy representing this variable. We construct the adequate encoding dichotomies for particular state variables delivering a certain desired information by merging together the atomic dichotomies corresponding to this information. The choice of the atoms of information to be merged together is guided by their affinity in relation to a combination of their common source (common inputs used to produce particular atoms of information) and common destination (common outputs using the atoms).

The state encoding process is divided in two phases (see Fig. 5.2):

- analysis phase – computation of the affinities for the atomic state information
- code construction phase – using the affinity information for code construction

Analysis phase

The analysis phase is composed of four steps:

1. Initial atomic encoding of the FSM.
2. Input support computation of the next-state and output variables of the initially encoded FSM.

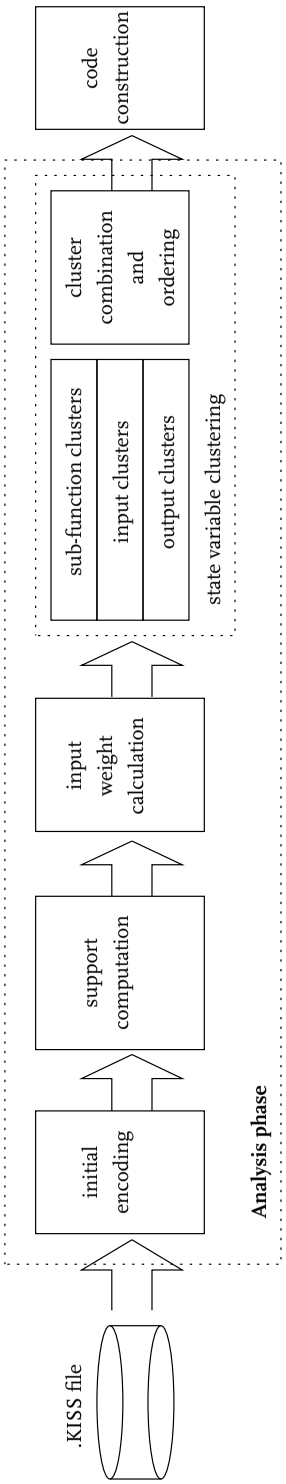


Figure 5.2. Outline of the state assignment method

3. Input weight calculation.
4. Clustering of the state variables from the initial encoding based on the results of steps 2 and 3.

The clustering results are then used to determine which state variables of the initial encoding will be merged together in the code construction phase.

The analysis of the source and destination of a particular atomic information is performed by encoding the machine with the **atomic encoding**, i.e. the encoding in which each state variable is defined by an atomic dichotomy, thus delivering atomic state information (i.e. a single distinction between two particular states). Analysis of the **input supports** in such encoded machine supplies us with the detailed information about how a particular atomic information is produced (which primary inputs and atomic state variables are in its support) and how it is used (in which supports it appears). This analysis is augmented by considering not only which inputs are present in particular input supports, but also their relative importance. The importance of an input is expressed as its **input weight**. The weight reflects the influence of the input on the quality of the solution.

Consider, for example, the FSM in Fig. 5.3. The initial atomic encoding for the five states of the machine is given in Fig. 5.4. Encoding the machine with the atomic encoding results in the PLA description (truth table) of the combinational logic necessary to realize the next state and output behavior of the machine. The PLA is presented in Fig. 5.5. Given the binary next state and output function, we calculate minimum input supports for the functions, shown in Fig. 5.6. These supports are the basis for information flow analysis. The analysis informs us, for example, that to produce an atom of information distinguishing states a and b we require only the distinction d/e. The distinction a/b is then used to calculate distinction c/d and the primary output o_1. For clarity, we will not consider input weights in this example. The issue of input weights is discussed in detail in Section 5.3.3.

Using the results of the input support analysis, we identify **clusters of atomic dichotomies** that need to be merged together to optimize information flow. For the sake of optimized information distribution and consumption, dichotomies that are used by the outputs that may form independent subsystems should be conveyed together. Then,

i	ps	ns	o
0	a	e	00
0	b	e	00
0	c	e	00
0	d	e	00
0	e	b	01
1	a	d	11
1	b	c	01
1	c	d	11
1	d	a	10
1	e	c	01

Figure 5.3. Example FSM

	a/b	a/c	a/d	a/e	b/c	b/d	b/e	c/d	c/e	d/e
a	0	0	0	0	–	–	–	–	–	–
b	1	–	–	–	0	0	0	–	–	–
c	–	1	–	–	1	–	–	0	0	–
d	–	–	1	–	–	1	–	1	–	0
e	–	–	–	1	–	–	1	–	1	1

Figure 5.4. Initial atomic encoding

the subsystem formed by the related outputs is fed with compressed, relevant information. To identify such related outputs, we analyze information consumed by the outputs and cluster together outputs with high similarity of consumed information, or with high potential for extracting common subfunctions.

In the example we are considering, we observe that the outputs o_2, a/b, a/c, a/d and b/e use similar information. They have therefore the potential to form a separate, coherent sub-network. From the viewpoint of the state information transmission and consumption it would be therefore advantageous to combine the information consumed by these outputs and deliver it in as compressed form as possible. The information used by these outputs is a/d, a/e, b/d, b/e, c/d, c/e and d/e. These atomic distinction form therefore a dichotomy cluster that is a candidate for merging together.

On the other hand, from the viewpoint of the state information production, we would like the dichotomies (state information) produced using the same information to be

i	$\frac{a}{b}$	$\frac{a}{c}$	$\frac{a}{d}$	$\frac{a}{e}$	$\frac{b}{c}$	$\frac{b}{d}$	$\frac{b}{e}$	$\frac{c}{d}$	$\frac{c}{e}$	$\frac{d}{e}$	$\frac{a}{b}$	$\frac{a}{c}$	$\frac{a}{d}$	$\frac{a}{e}$	$\frac{b}{c}$	$\frac{b}{d}$	$\frac{b}{e}$	$\frac{c}{d}$	$\frac{c}{e}$	$\frac{d}{e}$	o_1	o_2
0	0	0	0	0	–	–	–	–	–	–	–	–	–	1	–	–	1	–	1	1	0	0
0	1	–	–	–	0	0	0	–	–	–	–	–	–	1	–	–	1	–	1	1	0	0
0	–	1	–	–	1	–	–	0	0	–	–	–	–	1	–	–	1	–	1	1	0	0
0	–	–	1	–	–	1	–	1	–	0	–	–	–	1	–	–	1	–	1	1	0	0
0	–	–	–	1	–	–	1	–	1	1	1	–	–	–	0	0	0	–	–	–	0	1
1	0	0	0	0	–	–	–	–	–	–	–	–	1	–	–	1	–	1	–	0	1	1
1	1	–	–	–	0	0	0	–	–	–	–	1	–	–	1	–	–	0	0	–	0	1
1	–	1	–	–	1	–	–	0	0	–	–	–	1	–	–	1	–	1	–	0	1	1
1	–	–	1	–	–	1	–	1	–	0	0	0	0	0	–	–	–	–	–	–	1	0
1	–	–	–	1	–	–	1	–	1	1	–	1	–	–	1	–	–	0	0	–	0	1

Figure 5.5. PLA of the FSM with atomic encoding

$a/b : \{d/e\}$
$a/c : \{b/d, d/e\}$
$a/d : \{a/d, c/d\}$
$a/e : \{i\}$
$b/c : \{i\}$
$b/d : \{i\}$
$b/e : \{a/e, b/e, c/e, d/e\}$
$c/d : \{a/b, a/e, b/c, c/e\}$
$c/e : \{i\}$
$d/e : \{i\}$
$o_1 : \{i, a/b, a/e, b/c, b/d, c/e, d/e\}$
$o_2 : \{i, a/d, a/e, b/d, b/e, c/d, c/e, d/e\}$

Figure 5.6. Initial input supports

	acd/be	abc/de
a	0	0
b	1	0
c	0	0
d	0	1
e	1	1

Figure 5.7. Final encoding

merged, so that the distribution and usage of the input information is optimized. Therefore, we construct **clusters of state outputs** using similar input information. The dichotomies represented by these outputs are candidates for merging. In our example, the obvious candidates for merging are dichotomies a/e, b/c, b/d, c/e and d/e, which all use the same, single primary input. If they are merged together, the resulting code bit can potentially only depend on one input, thus making its implementation very efficient.

Once the clusters are determined, they are combined and ordered according to a **cluster quality measure**. The measure estimates the saved interconnections and circuitry if the dichotomies in the cluster are merged together. For instance, the dichotomy cluster $\{a/e, b/c, b/d, c/e, d/e\}$ has a very high quality, as it results in a very small implementation of the code bit. This is the consequence of the fact that, when merged together, the dichotomies form an encoding bit that can be computed using only a single primary input.

Code construction

In the code construction phase, the atomic dichotomy clusters sorted in the descending order of their quality are merged in order to realize the gains predicted by the cluster quality. The encoding dichotomy(-ies) resulting from the merging of particular clusters define consecutive state variables of the final encoding. Since some atomic dichotomy clusters cannot be merged to one dichotomy due to compatibility constraints, the distribution of the clusters' atomic dichotomies over the two or more encoding dichotomies has to be established. This process is driven by the atomic-dichotomy-affinity measure, which for a given pair of atomic dichotomies reflects, how important it is for the two dichotomies to be merged together. This measure is calculated based on the number and quality of the atomic dichotomy clusters, in which the pair of dichotomies appears together.

The merging process stops when all the atomic dichotomies that appear in any input support are realized by some of the current encoding dichotomies. Note, that this is not necessarily equivalent to realizing all the atomic dichotomies, since some state distinctions may not be needed (as is the case in the not minimized sequential machines). This relaxation of the constraint to realize all the atomic dichotomies is a consequence of Theorem 4.2 and means that our method is capable of implicit state minimization of the encoded machine by assigning overlapping, incompletely specified codes to equivalent states.

For instance, the dichotomies in the cluster $\{a/e, b/c, b/d, c/e, d/e\}$ can be merged together to a single encoding dichotomy acd/be. However, the dichotomies in the second cluster $\{a/d, a/e, b/d, b/e, c/d, c/e, d/e\}$ cannot be merged to one bit, due the fact that three dichotomies a/d, a/e and d/e can never be realized on a single binary state variable. The maximum compression of this information is achieved when the atomic dichotomies are merged to two dichotomies: abc/de and d/e. In this way, we have obtained three encoding dichotomies : abc/de, d/e and acd/be. Since the second dichotomy d/e is covered by the third, we may remove it. Finally, we have obtained two encoding dichotomies, which define the encoding showed in Fig. 5.7. We may observe that the states a and c were assigned with the same code. This is because the distinction a/c is not realized by any of the encoding dichotomies. However, if we analyze the input supports in Fig. 5.4, we can see that the distinction a/c is not used for calculation of any output,

so it needs not to be computed. In this way, an implicit state minimization is performed: states a and c are both represented by a single state `00` in the final encoding.

The realization structure of the resulting binary machine is shown in Fig. 5.8. We can observe that the encoding introduced a serial decomposition of the realization structure, with two distinct submachines. First submachine, associated with the encoding bit defined by dichotomy acd/be, only uses primary input and local state information to produce the encoding bit and the first primary output. The second submachine, associated with encoding bit abc/de, imports state information from the first submachine and produces the second encoding bit and the second primary output.

In contrast, in Fig. 5.9, the realization structure for the same machine encoded with the natural binary encoding is shown. Not only does it require more memory elements, as it does not benefit from the implicit state minimization performed by our method, but also the supports of the functions are larger and the interconnection structure is much more complicated. This results in a much less efficient implementation.

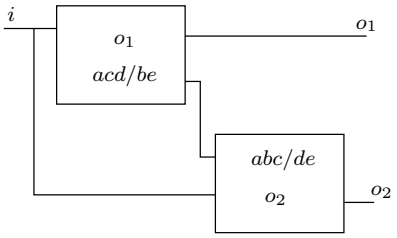


Figure 5.8. Realization structure of the encoded machine

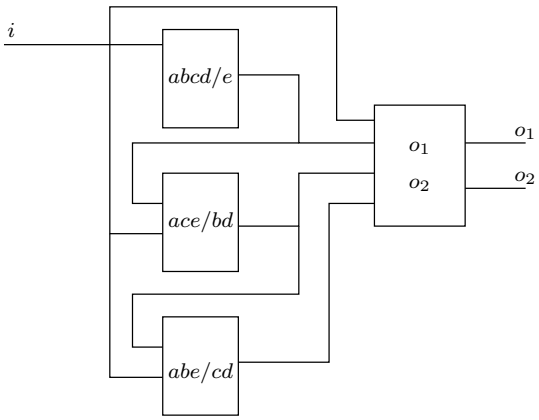


Figure 5.9. Realization structure of the binary-encoded machine

Summary

In this section, we have outlined the state assignment method based on information flow analysis and optimization. The method encodes the FSM with the initial encoding based on atomic dichotomies. It then computes the input supports of the binary functions in thus encoded machine to analyze the origin and destination of particular state information items. It also determines the relative importance of the information flowing within the network by assigning weights to signals carrying this information. Based on that analysis, the method determines clusters of related logic that may form coherent subsystems and the clusters of atomic dichotomies that need to be delivered together to optimize production and consumption of information in the subsystems. The atomic dichotomies in the clusters are merged together to compress the information represented by them on as few encoding variables as possible. In the following, we will describe in more detail the abovementioned steps.

5.3.2 Initial encoding

The input of the method is a Mealy-type finite state machine in KISS format. In this format the transitions of the machine are listed in the form

$$< input\ combination > \quad < present\ state > \quad < next\ state > \quad < output >$$

As described in Section 4.1.2, the machine is initially encoded with the atomic encoding. This encoding is defined by $s(s-1)/2$ atomic dichotomies (where s is the number of states). An atomic dichotomy is constructed for each pair (a, b) of states by placing the first state in the left block of the dichotomy, and the second in the right block. An encoding bit defined by an atomic dichotomy (a/b) has 0 for state a, 1 for state b, and don't-care for the rest of states. Thus, a symbolic FSM is transformed to a table of the binary output and next state functions by replacing the state symbols (names) in the state transition table with binary vectors (codes of the states). The result is an incompletely specified Boolean function with $i + sb$ inputs (i is number of the FSM inputs, sb is the number of state bits in initial encoding) and $sb + o$ outputs (o is number of outputs of the FSM).

5.3.3 Input supports calculation

To analyze production and consumption of the atomic information about states, the minimum input supports of the binary next state and primary output functions resulting from the atomic encoding are calculated.

Minimum input support selection is formulated as row covering problem. For each binary output function a set of distinction between the STT rows necessary to calculate this output is found. A distinction between rows r_i and r_j is necessary to calculate an output o if and only if the values of the output o are distinct (i.e. both are specified – not don't-care – and different) for rows r_i and r_j. For each row distinction a set of inputs capable of making this distinction is found. An input i is capable of making a distinction between rows r_i and r_j if and only if the values of the input i are distinct for rows r_i and r_j. (Minimum) input support of output o is a (minimum cardinality) set of inputs such that each of the row distinctions necessary for the output function is realized by at least one of the inputs. The choice of the input set is made by constructing a cover matrix $\mathbf{C}$ with columns labeled by the inputs and rows labeled by the distinctions required for the output. Matrix $\mathbf{C}$ has 1 in position (i, j) iff input j is capable of realizing distinction i. Therefore the input support problem may be reformulated as finding (minimum) number of columns (inputs) such that at least one 1 is present in each of the rows (each distinction is realized by at least one input). In our state assignment tool, the task of column selection is realized by QuickScan [45] algorithm.

Inputs in each of the input supports may be divided in two groups: primary inputs and state inputs. The fact that each of the state bits is defined by an atomic dichotomy, and therefore may be interpreted as delivering an elementary information about states (single distinction between two states) allows us to interpret the state input support as the state information required for the "production" of the function. Therefore, the production of a particular state distinction is determined by the state distinctions present in the input support of the output state bit defined by the distinction. Its consumption is then determined by the input supports in which the input state bit defined by the distinction appears.

This type of analysis provides us with very detailed information about the information flows and dependencies in the analyzed machine.

Input weights The information about production and consumption of the state information also allows us to make informed decision about the relative importance of the state distinctions and primary inputs. The importance is related to the influence of a particular input on the cost of the solution. We recall that our main stated objective is the reduction of the input supports of the particular binary functions and interconnections, as this has positive influence on the implementation cost of the circuit. Therefore, the importance of an input (or input weight) is related to the predicted input support saving associated with the input.

Intuitively, the weight reflects how important a particular input is from the viewpoint of the input support reduction, and hence how important it is to merge two dichotomies sharing this input. The weights depend on the frequency of occurance of a particular input in the input supports. The rationale behind this approach is that the inputs that are used by most of the outputs will have to appear in all or almost all of the input supports anyway, so no significant gain can be realized. Also, the inputs which appear only in one or two supports offer no gain, since they will be used by one output only anyway the merging proceeds. For these inputs the potential gains are low, hence their low weights. On the other hand, high gains may be realized in the case of inputs that appear in about 30% of input supports. If the dichotomies that use them are merged together, those inputs may potentially be used by very few outputs. However, if the dichotomies that use the outputs are not merged together, the inputs may be used everywhere in the circuit. It is therefore very important to group outputs having common high-weight input bits in supports, because this is where the savings lay.

To determine the weights, we calculate for each input bit a number of input supports in which the input is expected to appear. This figure has two components: input supports of primary outputs (these are the certain part, and will not change), and input supports of state bits (which are still to be merged, so the final input supports are not certain). The first component is calculated by simple counting the primary outputs, which have the input bit in the support. For estimation of the second component, we take the percentage of the expected final number of the state bits that is proportional to the percentage of all atomic state bits using the particular input.

Once the expected number of the outputs using given input is calculated, the input weight is determined by the weight mapping function that for a given percentage of

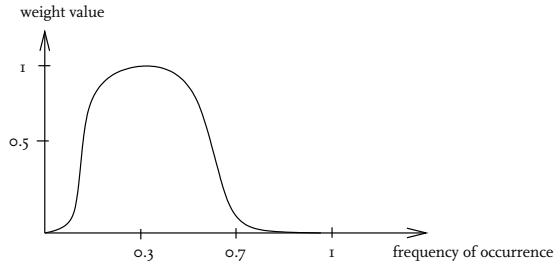


Figure 5.10. Weight mapping function

output number returns the input weight. The function is built based on the reasoning outlined above: the inputs with very low or very high utilization are assigned low weights. The maximum weight is assigned to the inputs with about 30% occurrence rate. In our experience, good results are achieved for the weight mapping functions of the shape similar to the function in Fig. 5.10.

5.3.4 Dichotomy clustering

The main problem in the general encoding scheme presented in Section 4.1.2 is the selection of the groups of atomic dichotomies to be merged together on the encoding dichotomies. In our method this selection is performed by clustering the atomic dichotomies in groups based on several information-flow-based criteria. In this procedure we identify several types of clusters of output functions, each reflecting different aspect of possible information flow optimization:

- **output clusters (input approach)** The output logic is analyzed to identify groups of outputs with large common subsets of input supports. Such relationship can indicate significant similarity between the outputs and possible realization as a separate, coherent and relatively independent subsystem. During implementation, the circuitry for such group of outputs is likely to be placed together in the layout. To optimize information delivery to such prospective subsystem, the information consumed by the subsystem's functions should be compressed. Therefore, the atomic dichotomies in the supports of the clustered outputs are candidates to be merged together.

- **output clusters (sub-function approach)** The output logic is analyzed to identify potential groups of outputs with large common sub-functions. Such groups are very good candidates for subsystems within the circuit. Following the reasoning of compressed information for subsystems, the atomic dichotomies in the supports of such related outputs are good candidates to be merged together.

- **next-state-bit clusters (output approach)** To optimize information production, it is advisable to compute together state information that consumes similar information. Then, the resulting next state functions will have compact input supports and will be relatively independent of the other next state functions, thus reducing interconnections. This is reflected by the dichotomy clusters that group together atomic dichotomies with similar input supports.

The clusters are created by two general clustering approaches: hierarchical and seed clustering. The hierarchical clustering creates non-disjoint clusters of elements by iterative merging of the current set of clusters. The seed clustering, on the other hand, initially identifies a number of "seeds" that initialize the set of clusters and than assigns the remaining elements to one of the clusters. Using different affinity functions and cluster selection criteria, the two general approaches are used to create the output and dichotomy clusters. For more detailed discussion of clustering techniques see Section 5.4.2.

The clustering procedure produces three different types of clusters, with different quality criteria and different impact on the implementation cost. Also, some of the clusters identified may not be useful, e.g. because they are too large or too small and thus impose too many or too few constraints on the encoding to be meaningful. To build

a framework for unified analysis of the clusters and their relative importance, a single cluster quality measure is introduced. It is related to the estimated size of saved circuitry and connections in the final circuit if the atomic dichotomies associated with the cluster are merged to one encoding dichotomy in comparison to their distribution over more encoding dichotomies. Based on their qualities, the different clusters are ordered.

In the following we will discuss the steps of the clustering and cluster ordering process: creation of the three types of clusters, ordering of the different types of clusters based on a common quality measure, transformation of the different clusters to dichotomy clusters, and the final ordering of the dichotomy clusters.

Input-oriented clustering

In the input-oriented clustering we identify groups of outputs that use similar state and input information and therefore have a chance of becoming coherent subsystems in the circuit. The similarity of the consumed information is estimated based on the input supports of the output functions. The clustering is performed in two steps. First, hierarchical clustering is applied to the set of singleton clusters. The affinity measure (`TotalDifferenceAffinity`) used in this step favors clustering outputs with similar supports (small difference). This phase, however, fails to identify relationships between outputs with large and small supports, even if small support is completely included in the large one. Therefore, the resulting clusters of the first phase are subjected to hierarchical clustering with inclusion-favoring measure (`InclusionAffinity`).

The reason for this two-phase approach is the tendency for the primary outputs to dominate the clustering process. We recall that at this point the state outputs are defined by atomic dichotomies. This means that they require relatively little information to be computed and therefore have small input supports, when compared to the primary outputs. Such small supports tend to be easily dominated by the large supports of the primary outputs. The result are large clusters built around particular primary outputs. To avoid this effect, in the first phase clusters of outputs with similar size of supports are favored.

Phase I. `TotalDifferenceAffinity` calculates similarity of two clusters as a difference between the weight of the inputs in the common input support and the weight of the remaining inputs in the supports. This value is given by $2 \cdot common\_weight - summary\_weight$

Both common and summary weight are calculated as a sum of two components: weight of primary inputs in the common or summary support and estimated weight of state inputs in the common or summary support. The state input weight is estimated by multiplying average weight of state input in the state supports by the expected number of state bits, on which the state bits will be merged. The expected number of merged bits is directly proportional to the number of atomic dichotomies in the supports. In this phase criteria allow clustering of the clusters with positive affinity, i.e. only outputs that have more common inputs bits that different are clustered together.

Phase II. `InclusionAffinity` detects inclusion of small supports in larger ones. It reflects the percentage of the small support that is included in the large support. The

value of the affinity is calculated as

$$1 - \frac{smaller\_tail\_size}{smaller\_IS\_size}$$

Where: $smaller\_IS\_size$ is the number of inputs in the smaller of the two supports.

$smaller\_tail\_size$ is the number of inputs present in smaller of the input supports and not present in the larger one

Since in this step we are interested in clustering outputs with fully or almost fully covered input supports, the affinity criteria allow for clustering of outputs with affinity above 0.8.

Sub-function clustering

This procedure identifies clusters of outputs with potentially large common subfunctions. The clusters are created in hierarchical fashion (see 5.4.2), with the initial cluster set consisting of singleton clusters. The affinity measure guiding the clustering algorithm is based on the number of common row distinctions between the output functions. Since the combinational synthesis method implemented in IRMA2FPGA is based on analysis of information expressed as term (row) distinctions, common row distinctions are consistent with the notion of common information central to the synthesis. Therefore, it is likely that the combinational synthesis detects and extracts common logic from two functions with high affinity measure.

The affinity measure is calculated as the ratio of common distinctions to the sum of distinctions, i.e. for two outputs o_i and o_j the affinity measure is

$$aff(o_i, o_j) = \frac{|IS(\phi_{S\times I}(o_i)) \cap IS(\phi_{S\times I}(o_j))|}{|IS(\phi_{S\times I}(o_i)) \cup IS(\phi_{S\times I}(o_j))|}$$

As described in Section 5.4.2, the hierarchical clustering procedure uses affinity criteria to determine which affinities between the clusters are high enough to merge the clusters together. In the subfunction-oriented clustering, the affinity criteria allow for merging of the clusters with the minimum amount of 70% of common distinctions.

Next-state-bit clustering

In the next-state-bit clustering step the next state outputs consuming similar information are clustered. The functions that compute these outputs have the potential for being compact, independent subsystems, and therefore the dichotomies that define them will be merged together. When clustering the state bits we are interested in a partition of dichotomies into non-overlapping blocks, and, preferably, each block corresponding to a state bit. For this reason, seed clustering (see Section 5.4.2) is performed rather than the hierarchical clustering.

The seed clustering starts by selecting a number of "seed" state bits around which the clusters will form. Since the target number of clusters is close to the number of expected encoding bits, the number of initial seeds is equal to the minimum encoding length of

the FSM. The remaining state bits are then assigned to one of the existing clusters or, if they do not fit into any cluster, a new cluster is created.

This process is guided by the affinity measure reflecting similarity of input supports. This measure is the weighted average of the affinities of the primary and state input supports. The weight is related to the ratio of primary to state inputs in the encoded machine. The support similarity is evaluated as the ratio of common inputs in the two supports to the sum of all inputs in the two supports. Thus, the affinity measure is

$$\left(p \cdot \frac{common\_primary\_weight}{summary\_primary\_weight} + (1-p) \cdot \frac{common\_state\_weight}{summary\_state\_weight}\right) \cdot \cdot(1 - size\_penalty)$$

where p is the percent of primary inputs in the final number of inputs given by

$$\frac{primary\_inputs}{primary\_inputs + \lceil log_2 \#states \rceil}$$

The additional factor $size\_penalty$ can be introduced to prevent too large clusters from forming. The penalty is related to the size of the cluster size. In our experiments we used linear penalty ranging from 0 to 1 for clusters of the sizes between m and M, where M is the number of all dichotomies and m is the minimum penalized size, e.g. 50% above the cluster size that would result from the balanced partition to equally sized clusters.

Output cluster ordering

The output groups identified by the clustering procedures described in the previous sections represent three different approaches to the optimization of the encoded machine. To reconcile the three views and compare their relative influence on the cost of implementation, a combined quality measure is necessary. The measure reflects savings in the realization cost if a given cluster is taken into account in the code creation. Of course, since each of the cluster types was created with different optimizations in mind, the quality estimation procedures will have to consider different saving sorts for different cluster types.

The procedures described below analyze possible savings in the realization logic if the dichotomies associated with a particular cluster are merged together. These savings involve both primary output and next state logic. To estimate the savings, the procedures analyze reductions of the input supports. These reductions involve primary input bits and present state bits. However, at this stage of the analysis, the state bits are defined by atomic dichotomies. This means that their number is much larger than it is going to be in the final encoding, but also that the state bits will change significantly, so any savings associated with them have to be considered as uncertain. Therefore, the identified savings have two distinct components: the primary savings – associated with primary inputs and outputs – and the state savings – associated with the state bits.

Output clusters The output clusters (both subfunction-based and input-based) are created to optimize information delivery to possible subsystems in the realization structure. The aim is to merge the dichotomies used by the clustered outputs to make the output's

supports compact and delivering compressed, relevant information. Therefore, the quality measure for output clusters reflects savings in the input supports of the clustered outputs if the used dichotomies are merged together.

These savings are evaluated as a difference between worst-case and best-case sizes of state bit support for the outputs in the cluster. The worst-case is when all the dichotomies in the support in the final realization are on separate state bits (limited by the expected number of state bits). Best-case is determined by the coloring of (no preference) twin graph of dichotomies. This procedure returns (close to) minimum number of dichotomies on which the given dichotomies may be merged (for details see 5.4.1).

Next-state-bit clusters We recall that the next-state-bit clusters are created to group together the state bits that share common input information and therefore make good candidates to be realized together. The savings associated with merging of such a group of state bits are twofold. On the one hand, the shared information is used by a single state bit (or a small number of bits) created by the merging of the dichotomies, thus making its distribution easier and promoting decomposition of the circuit into independent subsystems. On the other hand, we have to consider the influence of the created state bit on the supports of other functions, as it will be used by the functions instead of the individual dichotomies in the cluster.

Therefore, the quality measure for the state bit clusters has two components: input gain and output gain. Input gain expresses gain in terms of number of inputs saved if the state bits in the cluster are merged, and therefore calculated as one state bit, instead of being realized separately, on different state bits. Output gain expresses gain in terms of the number of connections saved if the state bits are sent to the functions that use them on one bit, instead of being sent on many bits. The above values are estimated as follows:

- **input gain**

  is calculated as a difference between estimated worst-case and best-case scenario. The worst-case input support for the dichotomies in the cluster is calculated by randomly placing the dichotomies on all (predicted) state bits, and assuming the input support of the cluster to be the sum of the input supports of thus created state bits. A state bit support is the sum of the supports of component dichotomies. Best-case is placing the dichotomies on as few bits as possible (no preference twin-graph coloring) and calculating the resulting input support as above.

- **output gain** For all outputs including in their supports some of the dichotomies in the cluster, the gain is estimated as the difference between the case, when all the used dichotomies are on maximum versus minimum possible number of bits.

Cluster transformation

At this point, the three types of output clusters exist and have cluster qualities associated with them. We recall, however, that for different clusters different sets of dichotomies need to be merged to realize the savings associated with the cluster. In this step, the groups of dichotomies associated with each cluster are created and organized as dichotomy clusters.

The transformation of clusters resulting from sub-function clustering and input-oriented clustering is performed in the following manner. Dichotomies (state input bits) in the input support of the outputs in the cluster are extracted. Intersections of the dichotomy supports form a partition into the dichotomy groups. Each group has a "depth" associated with the number of supports intersected, i.e. the number of outputs in the cluster using the dichotomies in the group. The graphical illustration of dichotomy groups is presented in Fig. 5.11. The sum of the dichotomy groups forms the dichotomy cluster.

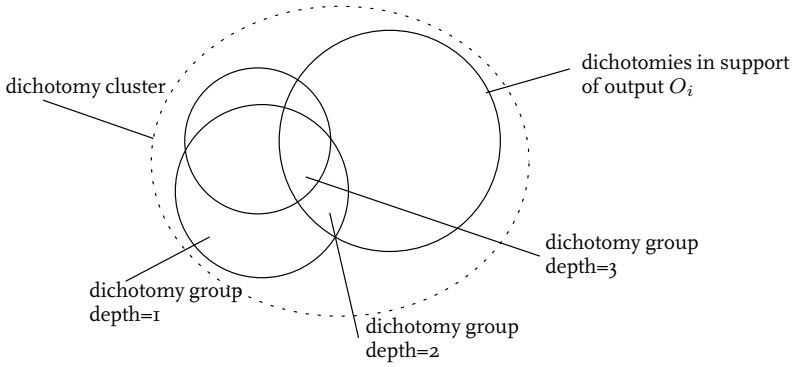


Figure 5.11. Dichotomy groups

The clusters resulting from output-oriented clustering already include the dichotomies we wish to merge together. Therefore, the transformation of these clusters only involves replacing the atomic state bits in the clusters with the atomic dichotomies corresponding to the bits and placing the atomic dichotomies in a single dichotomy group that makes up the dichotomy cluster.

Dichotomy cluster ordering

Finally, the identified dichotomy clusters are ordered according to their quality. We recall that the estimated quality of clusters has two components: primary savings, involving primary inputs and outputs and therefore relatively easy to predict; and state savings, which involve atomic state bits and can therefore significantly change in the dichotomy merging process. The ordering criterion is therefore the "reliable" primary gain on the cluster. The uncertain state gain is used as tie-breaker. It can only cause a change in the order introduced by certain gain if the difference in uncertain gain is very large (two times), and the difference in certain gain – small (10 percent).

Clustering process summary

In the above we have described the steps of the clustering of atomic dichotomies that should be conveyed together on the same encoding variables to optimize the implementation of the encoded FSM.

To find the dichotomy clusters, we identified three types of output function clusters that correspond to different potential optimizations. The input-oriented clusters focus

on the output functions that share common inputs and therefore may form a coherent sub-system. The sub-function-clusters reflect the term distinctions common to different output functions, and thus the potential for common subfunctions for these functions. Finally, the next-state-bit clusters identify atomic dichotomies that share input information and therefore should be realized together.

The identified clusters are ordered by introducing the common cluster quality measure related to estimated savings if the cluster is considered in the code construction. The ordered clusters are then transformed to corresponding clusters of dichotomies that should be merged, and the dichotomy clusters are finally ordered to prepare them for the code construction process.

5.3.5 Code construction

In this phase the created clusters of dichotomies are considered in the descending order of quality and the dichotomies in the cluster under consideration are merged on a (close to) minimum set of bits. Thus, an attempt is made to realize the gains estimated by quality of the clusters. The final result is a set of n encoding dichotomies, which defines the n-bit encoding.

In the beginning of the procedure, the preferences for each pair of the dichotomies to be realized on one bit are gathered over all the dichotomy clusters. This is done by considering number and quality of the clusters, in which the dichotomies appear together. The result is the dichotomy-affinity-matrix, containing in position (i, j) the level of preference for dichotomies i and j to be realized on one bit.

Then, the code is created. In each step a set of already created encoding dichotomies (partial code) is available. The highest quality, yet not merged cluster is selected, and the dichotomies in it are merged. The merging procedure has as the objective to fit the merged dichotomies on a minimum number of bits (encoding dichotomies). If the merged dichotomies do not fit on one bit (due to incompatibility), the distribution of dichotomies on the set of bits is guided by the dichotomy-affinity matrix, in such way that the distribution chosen (heuristically) maximizes summary preference over all pairs over all created bits. The merging procedure is stopped, when the current set of encoding dichotomies realizes all the atomic dichotomies.

Dichotomy-affinity matrix The dichotomy-affinity matrix reflects number and quality of the clusters that the particular pair of dichotomies is placed in. Therefore, it reflects preference for these two dichotomies to be realized on one code bit.

The matrix is initialized with zero values, and then updated by analysis of the dichotomy cluster set. For each dichotomy group in each dichotomy cluster, all the pairs of dichotomies formed within the group are enumerated and each the affinity of each of the pairs is increased with the value of cluster quality (predicted gain) multiplied by group depth. Additionally, for all the pairs in the cluster (even if they are not within one group) the affinity is increased by cluster quality value.

Cluster merging In this phase the code is constructed in such way that the dichotomies placed in one cluster are (preferably) put on a minimum number of code bits. To achieve this the clusters are considered in descending order of quality. For each cluster the dichotomies in the cluster are merged on a minimum number of encoding dichotomies.

If the number of resulting dichotomies is larger than one, the distribution of the atomic dichotomies over the encoding dichotomies is related to the pairwise affinity of the dichotomies expressed in the affinity matrix. This task is performed by the twin graph coloring procedure (see Section 5.4.1) with the preferences between dichotomies determined by the dichotomy-affinity-matrix.

After merging some encoding dichotomies may include not all state symbols. This is especially the case if the merged cluster was relatively small or if it did not fit on one encoding dichotomy and a small number of atomic dichotomies had to be merged on a separate encoding dichotomy. Such "low-density" encoding dichotomies realize only a small number of state distinctions, so they contribute to the growing of the number of state bits in the final encoding. In general, we want to avoid this effect, unless the low-density dichotomy is of such high quality that the code length increase is justified.

To increase the density of the encoding dichotomy, we select a limited number of not yet merged dichotomy clusters and attempt to add the not merged atomic dichotomies to the encoding dichotomy. This solution is only considered if the dichotomies can be merged on the same number of encoding dichotomies, as the existing cluster, i.e. the compactness of representation for the previously merged atomic dichotomies is not affected. This is especially important when we consider that the previously merged clusters have higher quality, and hence are more important to realize compactly. If the equally compact representation is possible, the quality of the two competing solutions is compared. The quality measure involves the increase of the density of the encoding dichotomy, weighted against the increased input support of the dichotomy.

Stepwise merging

As an alternative to the regular cluster-based merging procedure so called *stepwise merging* was developed. The method is similar to the approach proposed in [56], where at each step a pair of state bits occurring together in a large number of input supports is merged. In this way, each of the supports where the pair occurs is reduced by one bit.

Our method selects at each step a pair of compatible state bits (a pair of compatible current encoding dichotomies) with the highest affinity and merges them together. The affinity measure considers similar aspects as the clustering methods. It takes into account the aspects important for efficient realization of the resulting binary functions: the production and consumption of the state information.

The efficient consumption of the created encoding bit is taken into account in two ways. Firstly, the candidates for merging are selected from the pairs of encoding bits which occur together in a large number of input supports. These bits are often used together and therefore it is advantageous to merge them together. The rationale behind it is that the two merged state bits will be replaced by one bit in all the supports where the pair occurs and therefore the supports will be reduced by one bit. Furthermore, the number of interconnections routed to each of the functions will decrease, further simplifying implementation. When considering the input support reduction as a result of merging, one needs to take into account not only the two merged dichotomies, but also all the dichotomies implied by the merged dichotomy. For instance, merging encoding bits defined by dichotomies a/b and c/d will produce an encoding dichotomy ac/bd which will replace in the input supports not only a/b and c/d, but also a/d and b/c. Therefore,

the total input support saving introduced by merging two dichotomies d_1 and d_2 is

$$iss(d_1, d_2) = \sum_{\{is \in IS \mid d_1 \in is \wedge d_2 \in is\}} \left| \{d \mid d \in is \ \wedge d \subseteq \ d_1 * d_2\} \right| - 1$$

The second aspect of information consumption considered when merging state bits is the number of term distinctions realized by the new bit that are required by output functions. The larger the number, the more information required by the output is delivered by the input bit and therefore the fewer other inputs may be required by the output.

From the point of view of information production, the two merged bits (and all the implied dichotomies) should share the common input information, so that the resulting next-state function has an efficient implementation. To measure this aspect of state bit's affinity, the input support affinity measure is used (see 5.3.4). Its value, calculated for all pairs of dichotomies covered by the merged one, reflects how much the supports of the component dichotomies have in common.

The merging procedure selects from the current set of encoding dichotomies the dichotomy pairs that appear together in the largest number of supports. Then it analyzes the other encoding dichotomies covered by the merged pair and the supports they are used in. During this analysis, the pairwise input support affinities of the covered dichotomies are calculated. Finally, the state input bit resulting from merging the dichotomies is constructed and the term distinction realized by it and used by outputs are identified. The above measures are normalized to the range of $[0.0; 1.0]$ and multiplied to arrive at the composite affinity measure of the two considered dichotomies. The pair of encoding dichotomies with the highest composite affinity measure is merged and the data structures are updated to include the new bit in the input supports.

The update procedure is crucial to the efficiency of the method. In general, to determine input supports of the output functions at each step of the merging procedure, the machine should be encoded with the current encoding and time consuming input support procedure should be called. However, considering that the merging process is composed of relatively small changes to the encoding, most of the input support information in a given step can be obtained by appropriate modification of the input supports from the previous step. During the affinity analysis we already established the current encoding bits covered by the newly created bit. These bits can all be removed from the current encoding and they can be replaced in the input supports of the remaining outputs by the new bit. The input support of the new bit can be either established as a sum of the supports of the covered dichotomies or, for more precision, its input support may be calculated by the regular input support routine. The recalculation of just one input support at each merging step does not introduce severe performance penalty.

5.4 Structures and algorithms

In this section, we will discuss in more detail some of the fundamental data structures and algorithms used in SECODE.

5.4.1 Twin Graph

Twin graph is a structure first introduced in [12]. It is an undirected and self-loop-free graph with two types of edges: regular and "twin edges". Each node of the graph has

exactly one "twin node" that is connected to it with a twin edge. Additionally, every node can be connected with arbitrary other nodes by undirected regular edges. In this section we will present the formal definitions of twin graph and their application to dichotomy merging and the heuristics we developed to efficiently merge dichotomies with given preferences.

Definition 5.1 (Matching). *Let $G = (V, E)$ be an undirected, self-loop-free graph. A* matching *on a set of vertices $X \subseteq V$ is a set of edges $T \subseteq E$ such that*

$$\forall x \in X \quad \exists! \{v, v'\} \in T \ : \ x = v \oplus x = v'$$

$$\forall \{v, v'\} \in T \quad \forall \{w, w'\} \in T \setminus \{v, v'\} \ : \ \{v, v'\} \cap \{w, w'\} = \emptyset$$

In other words, a matching is a set of edges that group the vertices from X into unique, disjoint pairs.

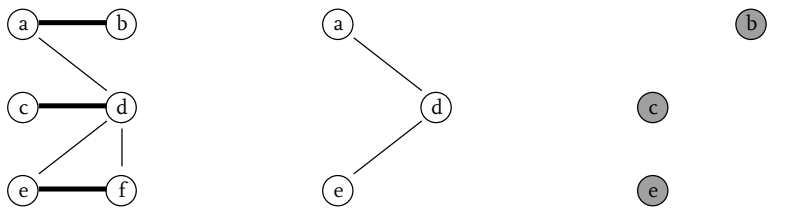
Definition 5.2 (Twin graph). *A* twin graph *is a pair (G, T), where $G = (V, E)$ is an undirected, self-loop-free graph and T is a matching on V. Each pair $\{v, v'\} \in T$ is called a* twin couple *and v is called a* twin *of v'.*

The graphical representation of a twin graph (G, T), with $G = (V, E)$, $V = \{a, b, c, d, e\}$, $E = \{\{a, b\}, \{a, d\}, \{c, d\}, \{d, e\}, \{d, f\}, \{e, f\}\}$, $T = \{\{a, b\}, \{c, d\}, \{e, f\}\}$ is shown in Fig 5.12(a). The twin edges are represented with thick lines.

An *instance graph* of a twin graph (G, T) is a graph $G \setminus V'$, where V' is a set of twin vertices that does not contain any twin couple and $|V'| = |T|$. In other words, an instance graph is a graph with exactly one vertex from each twin couple removed (see Fig. 5.12(b)). The remaining twin vertex is called *twin representative*. A *coloring* of a twin graph is a coloring of one of its instance graphs and *minimum coloring* is the minimum cardinality coloring that can be obtained for a twin graph over all its instance graphs (Fig. 5.12(c)).

Dichotomy merging with twin graphs

Twin graphs are particularly suitable for merging of sets of unordered dichotomies. For any set of unordered dichotomies, a twin graph can be built with a couple of twin nodes for each of the dichotomies. Each of the twins corresponds to one of the two possible orderings of the unordered dichotomy. For instance, the unordered dichotomy ab/cd generates a pair of twin nodes with associated ordered dichotomies $ab//cd$ and $cd//ab$. Then,



(a) Twin graph (b) Instance graph (c) Minimum coloring

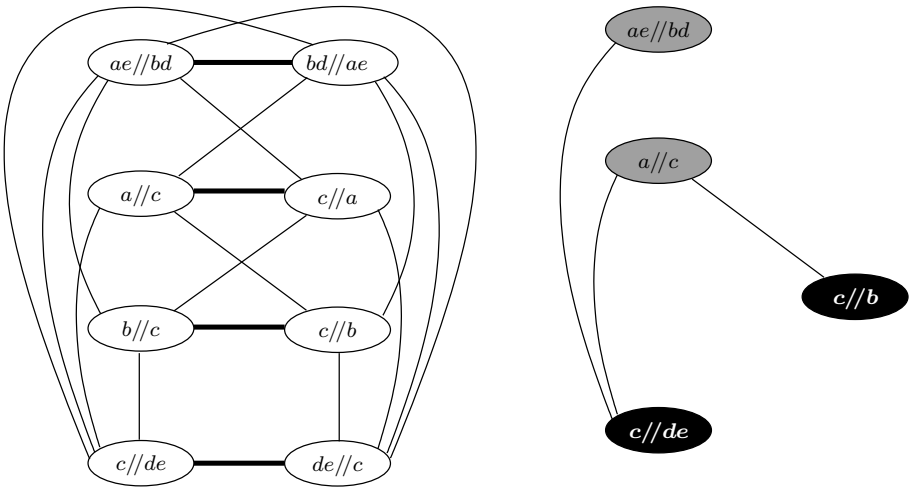
Figure 5.12. Twin graph example

a regular edge between the nodes indicates incompatibility of the ordered dichotomies associated with the nodes connected by the edge. The minimum coloring of the twin graph constructed in this way gives the minimum number of dichotomies that the initial dichotomies can be merged to. The resulting merged dichotomies are obtained by merging together all ordered dichotomies associated with the nodes colored to the same color.

Consider, for example, the set of unordered dichotomies $\{ae/bd, a/c, b/c, c/de\}$. The twin graph for this set is shown in Fig. 5.13(a). The colored instance graph is given in Fig. 5.13(b). The dichotomies ae/bd and a/c were selected in the positive polarity (i.e. $ae//bd$ and $a//c$) and given the same color. They are therefore merged to form the dichotomy $ae//bcd$. Note that due to the symmetry of the dichotomy merging problem, any instance graph $G \setminus V'$ has the same cardinality coloring as the instance graph V'. In other words, given a colored instance graph, same cardinality coloring can be obtained for the instance graph created by replacing each node by its removed twin. In this example, in such inverse graph, the dichotomies ae/bd and a/c would be both present in negative polarity ($bd//ae$ and $c//a$), colored to the same color and merged to dichotomy $bcd//ae$. Since merging in both polarities is possible, the result is an unordered dichotomy ae/bcd. The same reasoning applies to the dichotomies b/c and c/de, which can be merged in negative-positive polarity ($c//b$, $c//de$) or positive-negative ($b//c$, $de//c$), so the result is the dichotomy c/bde.

Heuristics for twin graph coloring

The problem of graph coloring is known to be NP-complete. Therefore, efficient heuristics are necessary to solve it in reasonable time. In the case of twin graph coloring, the



(a) Twin graph representation

(b) Coloring

Figure 5.13. Dichotomy merging with twin graph

process is additionally complicated by the necessity to select one vertex out of each twin couple.

The basic heuristic is derived from the well-known DSATUR [5] graph coloring heuristic. It is based on sequential selection of the "most difficult to color" vertices and coloring them to the first available color. This way, the most constrained vertices are dealt with in the early stages of coloring, when the freedom of color's choice is not yet limited by the previous choices. In this context, the most difficult vertices are the ones that already have many colored neighbors (they have high *saturation*), so the number of colors allowed for these vertices is small. As a tie breaker, the number of edges incident to the vertex is used. This follows the rationale that vertices that impose many constraints should be colored early on, so that the imposed constraints can be addressed in the early stages of the coloring procedure.

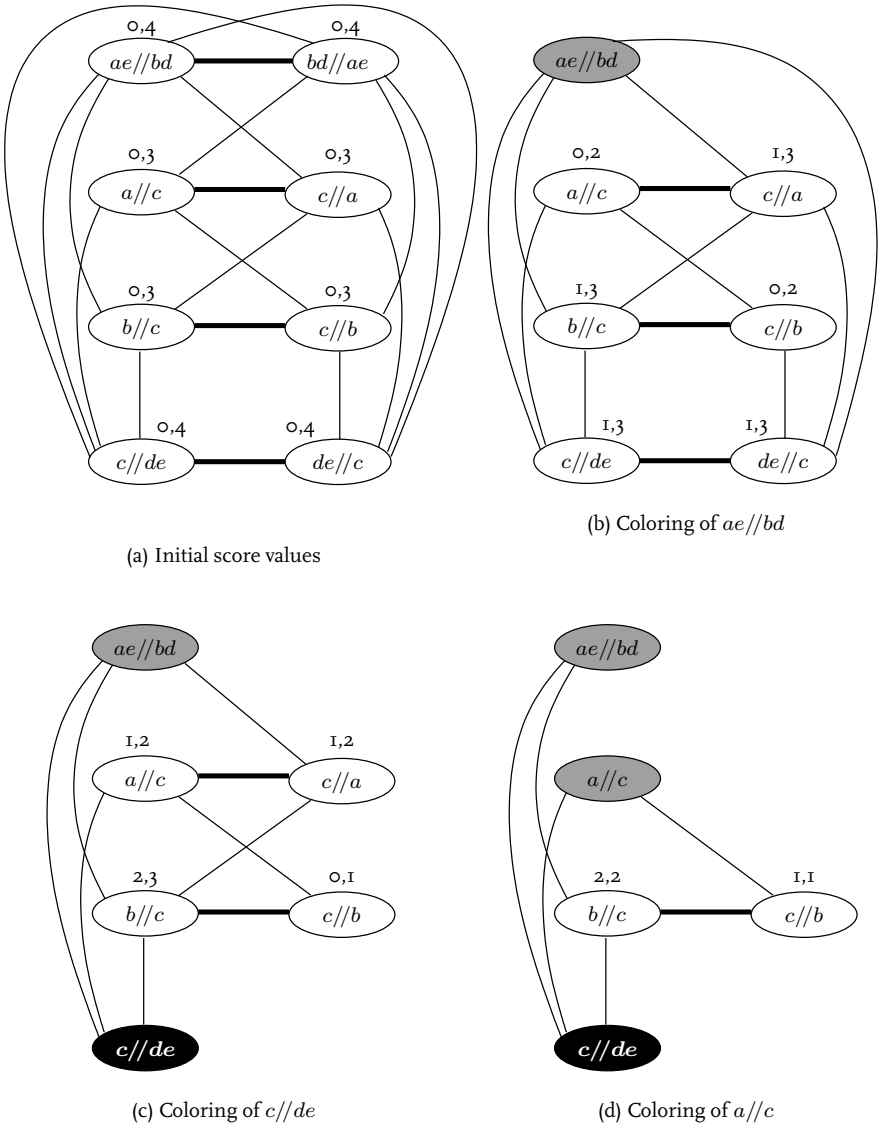
On top of that, the heuristic for twin representative selection needs to introduced. In this case, the obvious choice is the selection of the less constrained of the two twins (i.e. the twin with a lower DSATUR-score).

These two heuristics are intertwined in the framework of the coloring method. The strategy that proves to work well is the selection at each step of the most constrained twin couple (i.e. the couple for which the representative is more constrained than representatives for other couples), removing the more constrained twin, and coloring of the remaining representative.

Consider the initial score values for the twin graph in Fig. 5.13(a) (scores for particular nodes are given in Fig. 5.14(a) in form *saturation, number of neighbors*). The most constrained twin pairs are the pairs associated with dichotomies ae/bd and c/de. Both twin vertices in each of these pairs can be considered a representative with no colored neighbors (saturation = 0) and four incident regular edges. Let us select the pair ae/bd and the vertex $ae//bd$ to be the representative of the pair. The twin $bd//ae$ is then removed and $ae//bd$ is colored to the first color (gray). The resulting saturations are shown in Fig. 5.14(b). Now, both pairs a/c and b/c have a representative with score (0,2). In the pair c/de, on the other hand, both vertices have score (1,3), so this pair is selected as the most constrained, the vertex $c//de$ is arbitrarily selected as the representative and colored to the second color (black). Further, the pair a/c has two representatives with score (1,2) and the pair b/c a representative with score (0,1). $a//c$ is selected as the representative of the more constrained pair and colored to the first color. Finally, $c//b$ is the representative of the remaining pair, and is colored to the second color, resulting in the coloring in Fig. 5.13(b).

Heuristics specific to dichotomy merging

As illustrated by the above example, even with multiple constraints, the choice of equally constrained vertices to color can be quite large. Furthermore, in some cases a choice of colors can be available for a selected vertex. For instance, when coloring the pair a/c in the above example, both twin vertices were equally constrained with the score of (1,2), but the vertex $a//c$ could only be colored to the first color if selected as representative, while the vertex $c//a$ could only be colored to the second color. Also, if there were more than two colors currently used in the graph, both of the vertices would have additional color to choose from. In situations like that, additional criteria associated with particular problem can be used to break the ties.



(a) Initial score values

(b) Coloring of $ae//bd$

(c) Coloring of $c//de$

(d) Coloring of $a//c$

Figure 5.14. Twin graph coloring process

In the case of dichotomy merging, one of such criteria can be the preference of particular dichotomies to be merged together. This preference is given as an affinity matrix, where the pairwise preferences for all pairs of dichotomies are recorded. For instance, the choice of coloring $a//c$ to the first color over coloring $c//a$ to the second color could be dictated by the fact that the dichotomy a/c has more preferences towards merging with ae/bd rather than with c/de. Even more detail is introduced by analyzing atomic dichotomies resulting from coloring a particular vertex (i.e. merging it with other dichotomies already colored to the same color). Returning to the previous example, it might be the case that the atomic dichotomy a/c has more affinity towards the atomic dichotomies a/b, a/d, b/e, and d/e represented by ae/bd, than towards c/d and c/e represented by c/de.

Another criterion for the selection of colors can be the atomic dichotomies implied and forbidden by the choice of particular coloring. In general, the goal of dichotomy merging is to determine as few as possible dichotomies that cover all the initial dichotomies. In some cases, it may additionally be desirable that the merged dichotomies cover as few as possible atomic dichotomies *outside* of the initial set, provided that the minimum cardinality of the coloring is still preserved. An example of such case is the encoding construction, where a number of atomic dichotomies is selected to be merged together on as few as possible encoding dichotomies, and the preference is that *only these* atomic dichotomies are present on the resulting encoding dichotomies. Consider, for example, the set of atomic dichotomies a/b, a/c, b/c and c/d, and assume that $a//b$ and $a//c$ where both colored to color 1 and $b//c$ to color 2. This means that $d//c$ has a choice of being colored to any of these colors. However, if colored to color 1, the resulting merged dichotomy ad/bc would include additional implied atomic dichotomy b/d and would forbid the dichotomy a/d from being realized on the same bit. On the other hand, if $d//c$ is colored to color 2, the resulting merged dichotomy bd/c does not introduce any implied or forbidden dichotomies.

Twin graph implementation

The abovementioned heuristics were implemented in `TwinGraph` class. The class constructor accepts a list of unordered dichotomies and optionally the affinity matrix for the dichotomies. The affinity matrix can include positive affinities as well as negative (indicating that the dichotomies should not merged if possible). The negative affinity value `INT_MIN` is reserved to indicate that the two dichotomies should never be merged together. For each unordered dichotomy, a couple of twin nodes is constructed and connected with a twin edge. Then, the incompatibilities between the ordered dichotomies represented by nodes are determined and represented as regular edges in the graph. The coloring procedure is described in Alg. 5.1.

The procedure at each step selects the best candidate for coloring and colors it to the best color. The candidates for coloring are selected from the group of twin couple representatives with the highest DSATUR score (line 3). All the candidates are required to have the same saturation value, but the number of neighbors can differ within a given margin of the highest neighbors number in the group. Then, the candidate list is sorted in the descending order of preference level (line 6). The preference level is a sum of absolute values of positive and negative affinities that the candidate vertex has to other vertices in the graph. A large value of preference level indicates that the vertex has strong

Algorithm 5.1 Twin graph coloring

```
1: while( uncolored vertices present )
   {
3:    candidates = list of twin representatives within a margin
                   of the highest DSATUR score;

6:    if( preferences present ) sort candidates by preference level;
      else sort candidates by DSATUR score;

9:    candidates = a number of candidates from the front of the list;

      for each v in candidates {
12:      determine colors available for v;
13:      determine preferences of v for each of the available colors;
         colors = select a number of colors with highest preference;
         for each c in colors {
16:         analyze dichotomies implied and forbidden by coloring
            vertex v to color c;
18:         if( better that current best ) bestv, bestc = v, c
         }
      }

22:   remove twin of bestv;
      color bestv to bestc;
   }
```

preferences in the selection of a color for it, so it should be considered first. Depending on the size of the problem, some of the less promising candidates can be discarded to save runtime (line 9). For each of the remaining candidates, available colors and preferences to the vertices already colored to the available colors are determined (lines 12 and 13). The most preferred colors are then analyzed to determine, what dichotomies would be implied and forbidden if the vertex was colored to a particular color (line 16). The best vertex and its best color with high preference and low implied/forbidden score is selected (line 18), the vertex's twin is removed and the vertex is colored (line 22).

5.4.2 Clustering

A major part of the analysis phase of SECODE is finding groups of outputs and atomic dichotomies that are related as far as consumed and produced information is concerned. To find these groups, we employ clustering techniques. Clustering of data is a well known and much studied problem used in diverse fields, such as statistics, social sciences, marketing, geography, biology and many others. An extensive overview of clustering classification and techniques can be found in [29] and some practical implementations of clustering are discussed in [53].

In our problem, we applied two different clustering strategies. First of them is a variation of hierarchical clustering, modified to handle cases of non-disjoint clusters. The other one is a partitional (disjoint) clustering method based on growing clusters around

initialy selected "seed" elements. Both algorithms were designed in a flexible manner to be able to apply various affinity measures, and selection criteria.

Hierarchical clustering

The hierarchical clustering algorithm accepts as input initial clusters (usually, one-element clusters) and merges them into larger ones. The resulting clusters constitute the input to the next level of clustering. The clustering stops when no merging is possible. The criteria for merging are based on similarity measure between clusters. At each level, similarities between all pairs of clusters are calculated. The similarity-measuring-function is a parameter to the clustering algorithm. The resulting similarities are presented as a graph (clusters are nodes, similarities are the weights of the edges). Then, the criterion-function, which also is a parameter to the algorithm, is applied to eliminate some of the edges. This way, a sort of threshold graph is constructed. However, the application of criterion-function rather than a fixed threshold makes the selection much more flexible and capable of dynamic adjustment of thresholds, depending on the characteristics of the problem or on how advanced the solution is. In the resulting graph the cliques (complete sub-graphs) are identified, and the clusters in the cliques are merged together.

Seed clustering

The seed clustering algorithm identifies (based on similarity matrix) seeds of the clusters (single objects, mutually distant and with strong connections to other objects). The seeds form initial (one-element) clusters. Then, each of the objects left (one at a time) is assigned to one of the existing clusters. If for an object no cluster exists to which the object could be added, a new cluster is created, with the object as an element. The seed selection procedure starts with identification of the object with strongest relations (largest average affinity to its neighbors). Then, one-by-one, the remaining seeds are selected as the objects with lowest summary affinity to the current seed set (with 20% tolerance) and (tie-breaker) strongest relations. In the object assignment phase, an object with the strongest relations is selected. Its affinity to current clusters is determined as average affinity to the cluster's elements. If the affinity to a cluster is larger than affinity to remaining neighbors of the object, the object is placed in the cluster. Otherwise, a new cluster is created, with the object as a seed.

5.4.3 Code improvement with simulated annealing

In many problems with complicated characteristics, after a solution has been found, it is often a good idea to perform a local search around the solution to determine if no better solutions similar to the original one are present. Since the concerned neighborhood is much smaller than the entire solution space, more detailed and more time-consuming criteria can be applied to search for an improved solution, without a prohibitive performance penalty.

Simulated annealing

Simulated annealing is an optimization technique based on the analogy to the physical process of metal recrystallization [54]. In this process, a metal is heated to high tempera-

ture, at which it enters a highly disordered, high energy state. At any given temperature, the melt is allowed to stabilize, i.e. to achieve the lowest possible energy state at this temperature. Due to the nature of the process, the route to the stable low energy state may lead via some higher energy states. Once the equilibrium is achieved, the temperature is lowered and the melt is allowed to reach equilibrium at the new temperature. As cooling proceeds, the system becomes more ordered and approaches a "frozen", cristalline minimum energy state.

Optimization methods based on simulated annealing draw the analogy between the state of the thermodynamic system and the solution of the optimization problem at hand. The energy of the system is reflected by the cost of the solution. Following the analogy, at each temperature the solution is perturbed to form a new candidate solution. If the candidate solution is better (has a lower energy, i.e. lower cost), it is accepted. However, even if it is worse, it can still be accepted with the probability given by the Boltzman factor $exp(-\Delta E/T)$, where ΔE denotes the energy (cost) increase and T is current temperature. Note that, faithful to the physical analogy, the procedure will allow higher cost solutions to be accepted at higher temperatures, but with cooling the probability of such an event decreases. The outline of a typical simulated annealing algorithm is presented in Alg. 5.2.

Algorithm 5.2 Simulated annealing

```
determine initial solution S;
T = T_0;
repeat {
  while ( not an equilibrium )
  {
    perturb S to get a new solution S';
    deltaE = E(S') - E(S);
    if (deltaE < 0)
      replace S with S';
    else
      replace S with S' with probability exp(-deltaE/T);
  }
  T = T * alpha;                          // 0 < alpha < 1
} until (freeze);
```

A number of variations on this general scheme is possible. Some approaches use a fixed number of perturbations at each temperature, without the requirement for equilibrium. This saves significant amount of runtime, especially in the early stages of the algorithm, when the system is unstable due to high temperature. This savings come naturally at the expense of good statistical properties of the solution, but these may have small effect on the quality, especially if the initial solution was already a good one, so no large perturbations and no extensive search of the search space is required. It is also possible to dynamically adjust the number of perturbations at a given temperature, depending on the current conditions. In the early stages, it is also possible to use less precise cost functions, as the high temperature and the resulting high acceptance rate introduce an element of error that may offset small errors in the cost function [17].

One of the most important aspects of the algorithm is so called “annealing schedule”,

that is the choice of initial temperature and the method to decrease it in the subsequent steps of the algorithm. The simple-minded approach is the linear decrease of temperature by a factor $\alpha \in (0, 1)$ (as in Alg.5.2). More elaborate schedules are also possible. The most popular is the schedule that saves the runtime of the algorithm by quick cooling at the initial temperatures (e.g. by a factor of α^2) and slower cooling in the finishing stages (e.g. by α).

Code improvement

The code improvement procedure was implemented as simulated annealing in the scheme presented in Alg. 5.2. The solution is a given encoding, with the initial solution being a result of the code construction stage of SECODE. The initial temperature is selected in such way that a solution with the quality 10% worse that the initial solution is accepted with probability of 50%. This value is given by equation

$$T_0 = initial\_cost \cdot \frac{0.1}{ln(2)}$$

A current encoding is perturbed in a random manner by adding or removing a symbol from a block of one of the encoding dichotomies. If a symbol is removed, the atomic dichotomies lost by removing this symbol and not realized by other encoding dichotomies are recovered by adding appropriate symbols to the other encoding dichotomies. At each temperature, a limited number of perturbations is generated. This number increases logarithmically from the number of perturbations equal to the length of the encoding at the highest temperature (i.e. statistically, each bit is perturbed once) to 10 times the length of the encoding at the temperature 1. The number of perturbations at a given temperature is therefore given by

$$num\_perturbations(T) = encoding\_length \cdot \left(10 - 9 \cdot \frac{ln(T)}{ln(T_0)}\right)$$

The new solution is then evaluated by a cost function. The cost function is a parameter to the annealing procedure. We implemented two basic cost functions: the dichotomy-affinity-cost-function and the input-support-cost-function. The dichotomy-affinity-cost-function evaluates a solution based on a dichotomy affinity matrix, which reflects pairwise affinities of atomic dichotomies to be merged together. A solution is evaluated by enumeration of atomic dichotomies realized by the encoding dichotomies and summing up pairwise affinities of the atomic dichotomies realized on the same encoding dichotomy. The input-support-cost-function performs more precise evaluation of the solution by determining input supports of the output and next-state functions resulting from encoding the FSM with the encoding represented by the evaluated solution. The cost is then determined as the sum of sizes of input support of the functions. This procedure in general can be time consuming. However, taking into account that subsequent solutions are produced by small perturbations of the previous solutions, caching is extensively used to only recalculate the input supports actually affected by the perturbation, and to cache the unchanged rest.

5.5 Special encodings

When designing any computer optimization method it is usually desirable to make the method as general as possible, with few assumptions about the character of the sought solution. Any assumptions inevitably limit the explored solution space and may therefore preclude some very good, but "unexpected" solutions. To avoid this effect, a method should strive to search the solutions that optimize objective quality criteria, rather than the ones that follow a presumed pattern. On the other hand, such general, unconstrained methods usually have trouble finding some very specific, characteristic solutions that may be good in the particular domain. Naturally, when exploring a huge and irregular, discrete solution space, chance of finding one very specific, isolated point in it is small, however sophisticated the method. Therefore, practical methods often implement a general search algorithm coupled with a separate analysis of some few specific, well known solutions.

This is unfortunately not the case in state encoding methods implemented in most of the current commercial FPGA synthesis tools. They essential limit themselves to the analysis of only some specific encodings, namely: one-hot, sequential (natural binary), and Gray code, or even leave the choice entirely to the user. No general search for a good encoding is performed. Undoubtedly, this choice was dictated by the difficulty of the encoding problem and the perception that finding a good code is too great a burden on the runtime of the tool. However, a quick but naive choice of state assignment may result in a circuit which is not only much larger and slower, but also, due to its complexity, much harder to synthesize. The additional effort required from the combinational synthesis method may in this case exceed the effort required to find a more suitable encoding.

However limited this simplistic approach to encoding is, it stems from a valid observation that a certain limited number of practical, industrial benchmarks have good implementations when encoded with one of these specific encodings. This is particularly true for one-hot encoding, which is known to result in combinational logic described by very simple sum-of-product (SOP) expressions. While usually sum-of-product form is not the best indicator of the complexity of FPGA implementation, the specific form of the one-hot-induced functions makes them more often good candidates for an efficient implementation. In particular, the SOPs resulting from one-hot encoding often depend on less than all state variables and take form of a sum of very small (sometimes single-literal) product terms, with little overlap between the terms. This may result in the Boolean functions that are unate in some of their input variables and possess good disjoint decompositions. Both of these traits simplify logic synthesis of the Boolean functions and may result in effective circuit implementations. On the other hand, in many cases the sheer number of functions and memory elements required to implement a one-hot encoded machine is so large that the resulting realization is very large, however simple the particular functions are. Also, even the simplest functions with very wide supports still require a large number of LUTs to realize, as the LUTs are limited by the maximum number of inputs. As a result, one-hot encoding can only be applied to a small subset of all finite state machines and is not a universal method.

In this section we will describe a method of determining good candidates for one-hot encoding before the actual encoding. With this method we are able to efficiently analyze if a machine has a potential for good one-hot realization and if so, encode it without the relatively costly general encoding procedure, yielding good results in a fraction of the

runtime.

5.5.1 One-hot encoding

Our method is based on the prediction of the number of look-up-tables required for realization of the machine encoded with one-hot encoding and with some minimum-length encoding. Since our general encoding method results in close-to-minimum encoding length, it is a reasonable approximation of the possible state assignment result. The estimation of the number of LUTs is based on the number of inputs required to compute each function.

One-hot encoding cost

In the case of one-hot encoding, the inputs required to calculate each particular output and state bit are easy to estimate by analyzing the state transition table. Since each state will be represented in the one-hot encoded machine by a separate bit, the state bit support for a primary output bit consists of state bits corresponding to the states, in which the particular output is active. Also, for a next-state-bit, the previous states of the state corresponding to this particular bit will form it its state-bit-support. In the cases where thus assessed state bit support is very large, it should be reduced to account for the possibility of expressing the state conditions in a complementary fashion. For instance, for a 5-state machine the output that is active for the states 1 through 4 (and therefore depends on the first four state bits) can be expressed as an output active when the state is *not* 5, and therefore depends solely on the fifth state bit.

To estimate the primary inputs involved in the computation of a particular output, the primary input cubes for which the output is active can be analyzed to identify the active inputs. Of course, due to possible redundancy of representation in state transition table some of the seemingly active inputs may be not actually used by the output. For instance, an output active for input cubes `111` and `110` really only depends on the first two input bits, as the value of the third bit does not influence the value of the output. To identify such situation, a quick distance-1 merge can be performed among the primary input cubes. For more precision, actual input supports of the output functions can also be computed using a simple minimum input support algorithm (e.g. best first search).

Once the number of inputs used by an output is determined, the expected realization size is estimated by the expression

$$lut\_count\_hot(in) = \begin{cases} 0 & in \leq 1 \\ 1 & 2 \leq in \leq 5 \\ 2 + \frac{in-6}{2} & in \geq 6 \end{cases}$$

Note that the expression introduces linear dependency between the support size and the size of the realization. While in general it is far from true, the specific form of the Boolean functions resulting from one-hot encoding is such that their realization is very simple and requires little logic. In particular, one-hot encoded Boolean functions have very often very good disjoint decompositions, which simplifies functional decomposition of the circuit performed in the combinational synthesis and results in compact realization.

Minimum length encoding cost

The estimation of realization size for the machine encoded with a minimum length encoding is much more difficult. Naturally, the realizations for different encodings of the same length differ very considerably, which is exactly the reason that fuels state assignment research. Therefore, any estimations dealing with the precise size of the realization have to be wildly inaccurate.

However, in this case we are not interested in the exact size of the realization, but rather with the relation between the size of one-hot and some min-length realization. We are looking for a verdict that will indicate whether one of the realizations is clearly better, or the difference is too small to determine by such an inaccurate method. This approach calls for much smaller precision and, as we found out in our experiments, the rough estimation based on input support size is in most cases sufficient.

As in the case of one-hot realization, the min-length realization size is estimated based on predicted input supports of the binary next state and output functions. When determining the supports, we assume that a primary output function will use the same primary inputs it used in one-hot realization plus all encoded state bits. The next state functions have all the same support consisting of all primary inputs used by the state logic in one-hot encoding plus all the encoded state bits.

Once the predicted input supports of the output functions are determined, the size of realization is estimated by the sum of realizations of particular primary output and next state functions. The realization cost of a single output function is a function of its input support size and is given by the expression

$$lut\_count\_minlen(in) = \begin{cases} 0 & in \leq 1 \\ 1 & 2 \leq in \leq 5 \\ (2 + \frac{in-6}{2}) \cdot \frac{in}{6} & in \geq 6 \end{cases}$$

As can be observed, this expression has an additional factor of $\frac{in}{6}$ when compared with the expression used for evaluation of the size of one-hot realization. This factor introduces non-linear dependency of realization size on the number of inputs. This is necessary for functions resulting from min-length encoding, as they are much more complicated to realize than the one-hot functions. This is due to the fact that the information about states is encoded on the state bits in a more complicated fashion than in one-hot encoding, so additional circuitry is necessary to decode the information necessary to calculate the output functions.

5.6 Conclusions

In this chapter, we have discussed the state assignment method implemented in software tool SECODE. The method builds upon the general set-system-based encoding procedure proposed in Chapter 4. It uses the information relationships and measures apparatus to analyze relationships between information flows within the encoded machine and constructs the final encoding in such manner, as to optimize the information flows. To this end, it considers production and consumption of state and primary input information by the next-state and primary output functions. It then identifies groups of items of elementary state information that should be conveyed by the same binary encoding bits because

of the common source or destination of the elementary state information. Finally, it creates an encoding by explicitly constructing binary encoding bits conveying the desired elementary state information.

The information flow optimization implemented in SECODE realizes a general goal of finding a "natural" decomposition structure of the realization circuit, with separate, relatively independent, coherent sub-circuits. The optimized information flows deliver to these sub-circuits compressed, relevant information that can be efficiently processed. This approach is beneficial to any logic synthesis method or target implementation platform, but it addresses particularly the characteristics of LUT-based FPGAs and IRMA2FPGA as a logic synthesis method. In LUT-FPGAs, limits are placed on the number of inputs to a logic block and on the interconnection structure. Compression of relevant information results in Boolean functions with few inputs and the separation of independent sub-systems reduces the inter-system communication and, thus, the non-local interconnections. This approach is also compatible with the logic synthesis method implemented in IRMA2FPGA. As discussed in Section 4.2, this functional decomposition method is based on analysis of information delivered by particular inputs of a Boolean function and required by the output of the function. The method builds a network that removes the redundant input information and reorganizes it to form the desired output information. If the inputs deliver mostly relevant information, the process of removing the redundant information is greatly simplified, and smaller realization circuits can be found.

In this way, SECODE and IRMA2FPGA form a coherent synthesis chain for FPGA implementations of sequential circuits. In the next chapter, we will present experimental results testifying to the effectiveness of this approach.

Chapter 6

Experiments

In the previous chapters, we described the state assignment method based on set system (or dichotomy) representation of encoding and the heuristics that are used in the framework of this method to find encodings resulting in effective state assignment. To test the effectiveness of the proposed approach, we implemented this encoding method in the experimental EDA software tool SECODE. In this chapter, we will present and discuss the results of encoding experiments that we performed with the tool using a large set of benchmarks.

6.1 Comparison of one-hot and min-length encodings

In Section 5.5.1, we indicated that while generality is a very desired quality of a synthesis method, it comes at an expense of having to deal with a large solution space. In this huge space, a general synthesis method being guided only by some general heuristics may have a difficulty to reach some very specific solution points, even if they are known to often result in high quality circuits. Therefore, we proposed to accompany our general encoding method by a dedicated procedure to test in relation to a given FSM for high quality of some specific encodings, and in particular of the one-hot encoding. We proposed the method of evaluating the size of implementation of a one-hot-encoded FSM as compared to the FSM encoded with a minimum length encoding. This method enables us to predict with a reasonably high accuracy, which sort of encoding will be better for a given FSM: the specific one-hot encoding or the general encoding.

In Tab. 6.1 we present experimental results of the proposed method on the standard set of IWLS benchmarks [62]. In these experiments, we confronted the predictions of the method with the actual synthesis results of machines encoded both with the one-hot and min-length encodings. For min-length encoding we used JEDI, which is known to produce high-quality minimum length encodings. The encoded FSMs were synthesized with IRMA2FPGA, the synthesis method targeting LUT-FPGAs and based on functional decomposition [41]. IRMA2FPGA produces on average much better circuits than any other available academic or commercial combinational synthesis method [41]. In our experiments with IWLS benchmarks, it outperformed the popular academic logic synthesis tool, SIS [71], on average by 25% in terms of area of the synthesized circuits and by 30% in terms of delay measured by the number of levels.

fsm	jedi	hot	ind	real	fsm	jedi	hot	ind	real
a02	22	32	–	45%	opus2	26	21	–	−19%
a03	15	16	?	7%	patgen	29	36	–	24%
a04	43	53	+	23%	patrec	49	46	–	−6%
a05	11	16	+	45%	percent	15	20	+	33%
bbara	15	20	+	33%	planet	119	88	–	−26%
bbsse	29	31	?	7%	pma	43	47	+	9%
bbtas	5	8	+	60%	s1	33	94	?	185%
beecnt	9	13	+	44%	s1488	164	119	–	−27%
coffee	23	20	–	−13%	s1494	165	120	–	−27%
cse	53	56	?	6%	s208	20	25	+	25%
dk14	21	29	+	38%	s27	4	15	+	275%
dk15	7	15	+	114%	s386	41	29	–	−29%
dk17	6	16	+	167%	s420	19	26	+	37%
dk27	5	8	+	60%	s510	64	67	–	5%
dk512	7	19	+	171%	s8	5	8	+	60%
ex1	84	71	?	−15%	s820	114	65	–	−43%
ex4	15	20	?	33%	s832	107	71	–	−34%
ex5	11	15	+	36%	sand	191	174	?	−9%
ex6	24	25	+	4%	shiftreg	4	9	+	125%
example	9	11	?	22%	sse	29	31	?	7%
keyb	50	67	+	34%	tav	7	4	–	−43%
lion	3	5	+	67%	tbk	58	207	+	257%
lion9	5	10	+	100%	tma	21	32	+	52%
mark1	22	25	–	14%	train11	6	13	+	117%
mc	7	7	?	0%	vtiidec	53	56	–	6%
opus	23	18	–	−22%	vtiuar	71	66	–	−7%

Table 6.1. Realization cost of one-hot vs min-length encoding

The parameters of the synthesized circuits are given in Tab. 6.1. The columns *jedi* and *hot* give size (in 5-input LUTs) of the implementation of FSMs encoded with JEDI and 1-hot, respectively. The column *ind* shows the results of our cost estimation analysis. '+' stands for predicted increase of size in one-hot implementation, '–' for decrease in size and '?' for uncertain. The uncertain status was assigned to cases where the predicted difference in implementation size was under 15%. The *real* column shows real increase in the size of the realization when one-hot encoding is used instead of min-length.

We can observe that in over 80% of the cases the relationship between the size of one-hot and min-length realization is evaluated correctly. Especially for larger machines, the estimations agree with reality (e.g. *planet*, *s1488* or *s820*).

In most cases where the estimation is wrong, it overestimates the size of min-length realization and thus incorrectly indicates that one-hot encoding is better. In these cases, JEDI is able to find some optimization very specific to the given machine, which is impossible to predict just analyzing the input supports. This is particularly glaring in the case of the $s1$ FSM. The procedure estimated the sizes of both realizations as comparable, while in reality the one-hot realization is 185% larger than min-length. In this particular example, however, the large area reduction is due to very large number of common sub-

functions in the realization of min-length encoding. This type of optimization is naturally very difficult to accurately predict with our simple, but fast, estimation method.

Our assignment cost prediction method is very fast and accurate enough to be used in practical settings. Moreover, in the uncertain cases, we can always check both assignment methods.

6.2 SECODE

In this section, we will discuss the results of experiments with our information-driven state assignment method described in Chapter 5 and implemented in the software tool SECODE. First, we will present the results for the set of standard IWLS benchmarks, followed by the discussion of results for a large set of FSMs generated with benchmark generation tool BENGEN that we developed. BENGEN enables us to generate diverse FSMs having characteristics typical to circuits encountered in various industrial applications.

6.2.1 Standard benchmarks

For comparison of the encoding effectiveness, we used JEDI state assignment tool, as in our experiments it consistently produced better results than other available tools, such as NOVA or MUSTANG. Again, as in the case of one-hot experiments described in the previous section, we used IRMA2FPGA for combinational synthesis. The results of synthesis of the IWLS FSMs encoded with SECODE and JEDI are presented in Tab. 6.2. For each method, we give the number of 5-LUTs in the realization (column *a*) and the depth of the circuit (column *d*). Additionally, the percentage of difference with JEDI in area and depth is given in columns *a*% and *d*%, respectively.

fsm	JEDI		SECODE				SECODE'			
	a	d	a	d	a%	d%	a	d	a%	d%
a02	22	3	20	2	-9	-33	32	3	45	0
a03	15	2	14	2	-7	0	14	2	-7	0
a04	43	3	44	3	2	0	44	3	2	0
a05	11	2	8	2	-27	0	8	2	-27	0
bbara	15	2	12	2	-20	0	12	2	-20	0
bbsse	29	3	26	2	-10	-33	26	2	-10	-33
bbtas	5	1	5	1	0	0	5	1	0	0
beecnt	9	2	6	1	-33	-50	6	1	-33	-50
coffee	23	2	18	2	-22	0	20	2	-13	0
cse	53	3	49	3	-8	0	49	3	-8	0
dk14	21	2	18	2	-14	0	18	2	-14	0
dk15	7	1	7	1	0	0	7	1	0	0
dk17	6	1	6	1	0	0	6	1	0	0
dk27	5	1	5	1	0	0	5	1	0	0
dk512	7	1	7	1	0	0	7	1	0	0
ex1	84	3	50	2	-40	-33	50	2	-40	-33
ex4	15	2	14	2	-7	0	14	2	-7	0
ex5	11	2	7	2	-36	0	7	2	-36	0
ex6	24	2	21	2	-13	0	21	2	-13	0

example	9	2	7	1	-22	-50	7	1	-22	-50
keyb	50	4	41	5	-18	25	41	5	-18	25
lion	3	1	3	1	0	0	3	1	0	0
lion9	5	1	3	1	-40	0	3	1	-40	0
mark1	22	2	23	2	5	0	25	2	14	0
mc	7	1	7	1	0	0	7	1	0	0
opus	23	2	14	2	-39	0	18	2	-22	0
opus2	26	3	19	2	-27	-33	21	2	-19	-33
patgen	29	3	33	3	14	0	36	2	24	-33
patrec	49	3	51	3	4	0	46	2	-6	-33
percent	15	2	7	2	-53	0	7	2	-53	0
planet	119	4	155	6	30	50	88	3	-26	-25
pma	43	3	37	3	-14	0	37	3	-14	0
s1	33	3	26	3	-21	0	26	3	-21	0
s1488	164	5	176	5	7	0	119	3	-27	-40
s1494	165	4	176	6	7	50	120	4	-27	0
s208	20	3	16	2	-20	-33	16	2	-20	-33
s27	4	2	4	2	0	0	4	2	0	0
s386	41	3	24	3	-41	0	29	2	-29	-33
s420	19	3	17	3	-11	0	17	3	-11	0
s510	64	5	83	4	30	-20	67	3	5	-40
s8	5	2	1	1	-80	-50	1	1	-80	-50
s820	114	5	75	3	-34	-40	65	3	-43	-40
s832	107	6	71	3	-34	-50	71	3	-34	-50
sand	191	8	201	8	5	0	201	8	5	0
shiftreg	4	1	4	1	0	0	4	1	0	0
sse	29	3	26	2	-10	-33	26	2	-10	-33
tav	7	2	6	1	-14	-50	4	1	-43	-50
tbk	58	4	55	4	-5	0	55	4	-5	0
tma	21	2	20	2	-5	0	20	2	-5	0
train11	6	2	3	1	-50	-50	3	1	-50	-50
vtiidec	53	3	51	3	-4	0	56	3	6	0
vtiuar	71	4	44	2	-38	-50	66	3	-7	-25
Σ	1981	139	1816	125			1660	116		
$\Delta\%$			-8	-10	-14	-9	-16	-17	-15	-14

Table 6.2. Comparison JEDI with SECODE

As we can see in the table, in 65% of cases, SECODE manages to find encodings that result in smaller circuits than JEDI and for further 20% the results are the same. The identical results occur in most cases for the FSMs so small that no further improvement is possible. In some non-trivial examples, such as *ex1*, *s386*, *s820*, *s832* or *vtiuar*, the improvement of area reaches and exceeds one-third. Crucially, the area improvement does not come at a cost of delay increase. In fact, in the case of *s832* and *vtiuar* the 35% area improvement is accompanied by 50% delay improvement. The global improvement (summary over all machines) delivered by SECODE is equal to 8% for area and 10% for delay. On average (average percentage improvement per machine), the area of the implementations of FSMs encoded with SECODE is 14% smaller and the delay 9% smaller than of those encoded with JEDI.

Careful consideration of the results indicates that the global improvement is adversely

affected by just a few relatively large FSMs (such as *planet*, *s1488* or *s1494*), for which SECODE encoding is worse than JEDI. Further analysis of the results reveals that these are the machines that have a good one-hot encoding. It turns out that SECODE performs encoding that heads towards one-hot, however, in the process it is discouraged by the rapidly growing encoding length and applies more aggressive merging strategy that limits the further growth of the number of state bits. The result is the encoding that offers neither the simplicity of functions in one-hot encoding, nor the small number of functions of near-minimum-length encoding.

While we could change the heuristic to account for this effect, this was not necessary, as the functions with good one-hot realizations can be efficiently detected by the procedure described in the previous section. In the experiments reported in the third column of Tab. 6.2 (*secode'*), we integrated the one-hot analysis with the SECODE tool. If the analysis indicated strong preference for one-hot realization ('–' in Tab. 6.1), one-hot encoding was performed. Otherwise, the FSM was encoded with the general SECODE procedure. As discussed in Section 6.1, in some cases the one-hot analysis method incorrectly identifies good one-hot candidates. For instance, for *a02*, it indicated good one-hot realization, which is fact is 45% worse in terms of area. However, on the majority of benchmarks, the

fsm	JEDI		bin				Gray			
	a	d	a	d	a%	d%	a	d	a%	d%
a02	22	3	22	3	0	0	26	3	18	0
a03	15	2	15	2	0	0	15	2	0	0
a04	43	3	50	3	16	0	48	3	12	0
a05	11	2	13	2	18	0	12	2	9	0
bbara	15	2	15	2	0	0	11	2	-27	0
bbsse	29	3	39	3	34	0	41	3	41	0
bbtas	5	1	5	1	0	0	5	1	0	0
beecnt	9	2	10	2	11	0	8	2	-11	0
coffee	23	2	21	3	-9	50	22	3	-4	50
cse	53	3	48	4	-9	33	53	3	0	0
dk14	21	2	26	2	24	0	27	2	29	0
dk15	7	1	7	1	0	0	7	1	0	0
dk17	6	1	6	1	0	0	6	1	0	0
dk27	5	1	5	1	0	0	5	1	0	0
dk512	7	1	7	1	0	0	7	1	0	0
ex1	84	3	76	4	-10	33	72	3	-14	0
ex4	15	2	16	2	7	0	18	2	20	0
ex5	11	2	10	2	-9	0	8	2	-27	0
ex6	24	2	29	2	21	0	29	2	21	0
example	9	2	10	2	11	0	9	2	0	0
keyb	50	4	51	5	2	25	59	4	18	0
lion	3	1	3	1	0	0	3	1	0	0
lion9	5	1	4	1	-20	0	4	1	-20	0
mark1	22	2	24	2	9	0	23	2	5	0
mc	7	1	7	1	0	0	7	1	0	0
opus	23	2	30	2	30	0	23	3	0	50
opus2	26	3	25	3	-4	0	21	2	-19	-33
patgen	29	3	50	6	72	100	33	3	14	0
patrec	49	3	65	4	33	33	53	4	8	33

percent	15	2	13	2	-13	0	13	2	-13	0
planet	119	4	148	6	24	50	134	5	13	25
pma	43	3	47	4	9	33	42	3	-2	0
s1	33	3	154	7	367	133	172	8	421	167
s1488	164	5	168	6	2	20	174	6	6	20
s1494	165	4	164	5	-1	25	169	6	2	50
s208	20	3	12	2	-40	-33	18	3	-10	0
s27	4	2	7	2	75	0	5	2	25	0
s386	41	3	41	3	0	0	37	3	-10	0
s420	19	3	13	3	-32	0	18	3	-5	0
s510	64	5	83	6	30	20	53	5	-17	0
s8	5	2	5	2	0	0	6	2	20	0
s820	114	5	122	8	7	60	130	8	14	60
s832	107	6	134	8	25	33	129	7	21	17
sand	191	8	206	8	8	0	215	8	13	0
shiftreg	4	1	4	1	0	0	4	1	0	0
sse	29	3	39	3	34	0	41	3	41	0
tav	7	2	7	2	0	0	6	2	-14	0
tbk	58	4	107	6	84	50	142	7	145	75
tma	21	2	24	2	14	0	20	2	-5	0
train11	6	2	7	2	17	0	7	2	17	0
vtiidec	53	3	59	3	11	0	55	3	4	0
vtiuar	71	4	76	4	7	0	65	4	-8	0
Σ	1981	139	2329	163			2310	157		
$\Delta\%$			18	17	17	13	17	13	14	10

Table 6.3. Comparison JEDI with binary encoding

indications of the one-hot analysis method are correct. Therefore, it complements very well the general method and the combined procedures achieve global 16% area reduction and 17% delay reduction on the IWLS benchmarks.

In the next series of experiments reported in Tab. 6.3 we evaluated two encoding strategies prevalent in commercial synthesis tools: sequential binary encoding that assigns to states successive minimum length binary vectors representing natural numbers, and Gray encoding – another min-length encoding that assigns to subsequent states codes that differ on exactly one bit. Together with the earlier discussed one-hot encoding, these two approaches give the comparison of our method to the state assignment methods currently used in industrial synthesis tools.

As we can see from the results, sequential and Gray encodings are clearly inferior to all other presented methods. On average, sequential encoding results in circuits that are 17% larger and 13% slower that those encoded with JEDI, and over 40% larger and slower than encoded with our method. For Gray, the respective numbers are: 14% area increase w.r.t. JEDI and 46% w.r.t. SECODE, and, respectively, 10% and 35% delay increase. This result is not surprising, as both encodings, although having some specific patterns, are in fact random encodings for a particular FSM and can hardly be expected to produce consistently good results. Such encoding strategies are clearly not able to identify any optimizations of the encoded machine that the constructive methods such as JEDI and SECODE realize. Being minimum-length encodings they also do not benefit from the simplified output function form that in some cases favors one-hot encoding.

6.2.2 Interconnections

The results presented in the previous section evaluate the quality of implementation based on the number of blocks and the depth of the realization network for the encoded machine. While the number of blocks is very good indicator of the area of the circuit (there is a direct correspondence with FPGA slices), the depth of the circuit does not necessarily reflect the actual delay of the circuit once it is implemented in FPGA. This is due to the fact that in many circuits implemented in the modern FPGA devices interconnections become the dominant contributor to the overall delay of the circuit, instead of the logic. This is caused, among others, by the delay introduced by switches that the longer, programmable interconnections have to pass through. This switch delay is comparable to the LUT delay.

For this reason we indicated in Section 5.2.1 that one of the targets of our encoding heuristic is minimization of interconnections in the circuit, and particularly – long interconnections. In this section we discuss the results of the comparison of interconnection complexity encountered in circuits encoded with SECODE and JEDI.

Table 6.4 shows the results of analysis of interconnection structure in the circuits

```
sweep
xl_part_coll -m -g 2
xl_coll_ck
xl_partition -m
simplify
xl_imp
xl_partition -t
xl_cover -e 30 -u 200
xl_coll_ck -k
xl_merge -u 8 -o doc -l
```

Figure 6.1. SIS script

fsm	jedi+irma		secode+irma		jedi+sis	
	c	lc	c	lc	c	lc
a02	99	7	146	2	290	7
a03	69	0	66	0	172	4
a04	200	3	208	4	301	52
a05	52	0	41	0	94	6
bbara	73	0	57	0	96	4
bbsse	139	0	126	0	183	4
bbtas	30	0	26	0	28	0
beecnt	42	0	36	0	71	0
coffee	110	0	100	0	132	5
cse	252	4	231	3	262	14
dk14	94	0	86	0	103	0
dk15	42	0	42	0	42	0
dk17	36	0	36	0	35	0
dk27	25	0	25	0	24	0
dk512	42	0	42	0	41	0

ex1	382	10	234	0	526	21
ex4	78	0	76	0	104	1
ex5	52	0	34	0	66	0
ex6	117	0	95	0	146	11
example	39	0	31	0	80	0
keyb	214	16	180	15	274	19
lion	15	0	15	0	14	0
lion9	20	0	18	0	49	3
mark1	93	0	117	0	113	0
mc	42	0	42	0	24	0
opus	105	0	94	0	118	0
opus2	104	8	99	0	159	5
patgen	139	1	166	0	168	7
patrec	236	6	229	0	306	15
percent	73	0	37	0	96	4
planet	533	29	414	6	843	60
pma	204	5	170	1	702	63
s1	138	14	123	4	193	9
s1488	734	42	511	10	909	89
s1494	731	46	512	14	820	62
s208	86	1	72	0	118	4
s27	19	0	19	0	29	0
s386	175	7	133	0	184	7
s420	89	0	75	2	140	7
s510	273	18	271	2	345	23
s8	21	0	5	0	45	13
s820	531	23	297	2	463	16
s832	482	35	320	2	483	20
sand	752	137	820	110	692	49
shiftreg	20	0	20	0	10	0
sse	139	0	126	0	183	4
tav	29	0	32	0	43	0
tbk	271	17	246	14	305	18
tma	108	0	107	0	314	14
train11	25	0	15	0	52	0
vtiidec	250	10	268	5	358	3
vtiuar	347	8	320	8	442	13
Σ	8971	447	7611	204	11790	656
j+iΔ%			-15	-54	31	47
s+iΔ%					55	222

Table 6.4. Interconnection complexity for different encoding methods

encoded and synthesized with different encoding and synthesis methods. For each encoding/synthesis combination we give the number of resulting short and long interconnections between LUTs in the network (columns *c* and *lc*, respectively). For the purposes of this comparison, we define a connection as a single path from an output of a LUT to an input of another LUT. To evaluate the length of connections, we adopted a simplifying assumption that a short interconnection is a connection between two LUTs on neighboring levels of the network (produced output is consumed directly on the next level), while

a long interconnection has to travel over more than one level. In the comparison we used IRMA2FPGA and the popular division-based combinational synthesis tool SIS [71]. The SIS script used for FPGA synthesis was the "good" script proposed in [71], and repeated in Figure 6.1.

The results indicate that our encoding method indeed reduces number of interconnections, in particular the long ones. As we can see in the table, number of short interconnections is 15% smaller in SECODE-encoded circuits than in circuits encoded by JEDI, when both circuits are synthesized with IRMA2FPGA. For long interconnections, this reduction increases to 54%.

At the same time, we can see that the overall number of long interconnections resulting from synthesis with IRMA2FPGA is very small. This is an expected result, as interconnection reduction is one of the fundamental targets of the functional decomposition method implemented in IRMA2FPGA. Comparing the results of synthesis of the same JEDI encoding with IRMA2FPGA versus SIS, we can see that the synthesis method implemented in SIS introduced 31% more short interconnections, and 47% more long interconnections.

The combination of interconnection reduction introduced by the whole decomposition-based synthesis chain (SECODE + IRMA2FPGA) is clearly visible in comparison with the results of the division-based synthesis chain (JEDI + SIS). The achieved reduction reaches 55% percent for short and as much as 222% for long interconnections. This result is the testimony to the effectiveness, with which the information-driven, decomposition-based sequential synthesis chain addresses interconnections - the crucial aspect of modern digital circuits.

6.2.3 Layout results

In the previous sections we showed that SECODE generates encodings that result in circuits with smaller number of LUTs, smaller depth and less interconnections (particularly, long interconnections) in comparison with JEDI. However, to conclusively evaluate the quality of the synthesized circuits, we need to consider the area and the delay of the circuits *after placement and routing* on the target FPGA device. Only then are the final placement of the logic blocks and the resulting interconnections between them fully known.

To perform this comparison, we generated layouts for the finite state machines from the IWLS benchmark, encoded with JEDI and SECODE and synthesized with IRMA2FPGA.

fsm	JEDI		SECODE				SECODE'			
	a	d	a	d	a%	d%	a	d	a%	d%
a02	18	4, 4	16	3, 8	-11	-13	25	4, 3	39	-2
a03	10	3, 1	10	2, 8	0	-9	10	2, 8	0	-9
a04	34	5, 8	39	5, 7	15	-3	39	5, 7	15	-3
a05	8	3, 0	7	3, 5	-13	17	7	3, 5	-13	17
bbara	12	4, 3	9	3, 6	-25	-17	9	3, 6	-25	-17
bbsse	23	5, 1	22	4, 0	-4	-22	22	4, 0	-4	-22
bbtas	5	3, 2	4	3, 2	-20	-0	4	3, 2	-20	-0
beecnt	6	2, 8	6	2, 6	0	-7	6	2, 6	0	-7
coffee	18	5, 4	14	4, 4	-22	-19	13	4, 1	-28	-24
cse	46	6, 4	42	6, 3	-9	-2	42	6, 3	-9	-2

dk14	13	3, 2	12	3, 4	-8	7	12	3, 4	-8	7
dk15	7	3, 3	7	2, 6	0	-19	7	2, 6	0	-19
dk17	6	3, 2	6	3, 2	0	0	6	3, 2	0	0
dk27	3	2, 0	3	2, 0	0	0	3	2, 0	0	0
dk512	7	3, 2	7	3, 2	0	-0	7	3, 2	0	-0
ex1	64	6, 9	37	6, 3	-42	-8	37	6, 3	-42	-8
ex4	11	4, 3	12	4, 4	9	1	12	4, 4	9	1
ex5	8	3, 1	5	3, 0	-38	-4	5	3, 0	-38	-4
ex6	20	4, 3	16	5, 4	-20	25	16	5, 4	-20	25
example	6	4, 0	5	2, 6	-17	-34	5	2, 6	-17	-34
keyb	41	8, 2	35	9, 3	-15	14	35	9, 3	-15	14
lion	2	1, 9	2	1, 9	0	0	2	1, 9	0	0
lion9	3	2, 6	3	3, 2	0	23	3	3, 2	0	23
mark1	11	3, 2	16	5, 2	45	62	14	3, 5	27	10
mc	7	3, 2	7	2, 7	0	-17	7	2, 7	0	-17
opus	16	4, 2	10	4, 4	-38	3	12	3, 4	-25	-20
opus2	18	4, 8	15	5, 0	-17	5	14	3, 9	-22	-18
patgen	22	5, 0	29	9, 4	32	89	23	4, 6	5	-8
patrec	42	8, 6	38	6, 6	-10	-23	32	4, 4	-24	-49
percent	12	4, 3	6	3, 2	-50	-25	6	3, 2	-50	-25
planet	97	9, 1	124	12, 3	28	36	62	4, 1	-36	-55
pma	35	9, 8	30	6, 5	-14	-34	30	6, 5	-14	-34
s1	25	6, 1	20	6, 3	-20	3	20	6, 3	-20	3
s1488	137	13, 9	141	12, 2	3	-12	78	5, 0	-43	-64
s1494	138	8, 3	135	14, 0	-2	70	79	4, 3	-43	-48
s208	15	4, 2	11	3, 2	-27	-23	11	3, 2	-27	-23
s27	2	2, 7	3	2, 6	50	-3	3	2, 6	50	-3
s386	29	6, 1	20	3, 7	-31	-39	19	3, 9	-34	-36
s420	14	4, 2	12	2, 3	-14	-44	12	2, 3	-14	-44
s510	49	12, 1	66	9, 2	35	-24	41	3, 2	-16	-73
s8	3	3, 2	1	0, 0	-67	-100	1	0, 0	-67	-100
s820	99	11, 3	56	6, 1	-43	-46	44	4, 8	-56	-57
s832	93	11, 6	50	7, 1	-46	-38	49	5, 7	-47	-51
sand	164	16, 5	163	13, 0	-1	-21	163	13, 0	-1	-21
shiftreg	2	1, 9	1	1, 9	-50	-0	1	1, 9	-50	-0
sse	23	5, 1	22	4, 0	-4	-22	22	4, 0	-4	-22
tav	5	1, 9	6	2, 2	20	14	6	2, 2	20	14
tbk	54	7, 1	47	6, 5	-13	-9	47	6, 5	-13	-9
tma	18	5, 0	18	4, 1	0	-18	18	4, 1	0	-18
train11	3	3, 8	2	1, 9	-33	-48	2	1, 9	-33	-48
vtiidec	40	4, 1	40	4, 8	0	17	41	4, 1	2	-1
vtiuar	57	8, 8	33	4, 9	-42	-44	46	5, 8	-19	-34
Σ	1601	283, 7	1441	256, 0			1230	211, 8		
Δ%			-10	-10	-10	-7	-23	-25	-14	-18

Table 6.5. Results of experiments after layout

The output networks of 5-LUTs synthesized by IRMA2FPGA were converted to the networks of FPGA primitives in EDIF format and fed to Xilinx ISE 5.2 synthesis system for placement and routing on the VirtexII FPGA (device 2V1000FG256-4). The resulting

area of the circuit (in slices) and the clock period (in nanoseconds) are given in Tab. 6.5.

As we can see, the layout results confirm the improvements achieved using SECODE encoding instead of JEDI. The average improvements of 15% and 14% for area and delay remained after layout essentially the same at 14% and 18%. Of course, calculation of improvement as an average of per-machine improvements increases the influence of smaller machines, which form the majority of IWLS benchmark. In the average, reduction of 1 block in a 10-block machine will weigh the same as reduction of 10 blocks in a 100-block machine (both represent 10% improvement). Meanwhile, we can see that for many of the small machines no further reduction is possible. On the other hand, SECODE manages to achieve significant improvements primarily for larger machines, where the optimization potential is the greatest. These improvements are more clearly visible in the summary numbers over all machines, where each machines weighs according to its size. In terms of summary area and delay, the improvement introduced by SECODE reached 23% and 25%, respectively.

We can also observe that after layout the summary improvements have increased both for area and for delay when compared to the results after synthesis only. The additional area improvements stem from the difference between a 5-input LUT and a slice. While a slice (see Section 1.2.2) is usually treated as a single 5-input LUT, in fact it is composed of two 4-input LUTs, which can be configured to use separately. As we can see in Tab. 6.5, using number of slices as an area measure, SECODE outperforms JEDI by 23%, compared to 16% improvement in terms of 5-LUTs (Tab. 6.2). This indicates that the function blocks in synthesized JEDI circuits often use all 5 inputs, so they use a whole slice to implement. Meanwhile, blocks in SECODE circuits are not only fewer, but also smaller and use more often functions of 4 and less inputs. This enables implementation of more than one function in a single slice, and constitutes an additional advantage of SECODE.

We can also see that the summary delay improvement of 17% that SECODE achieves versus JEDI after logic synthesis, improves to 25% after layout. This can be contributed to the fact that the SECODE-encoded circuits have not only less levels, but also less interconnections.

6.2.4 Comparison of synthesis chains

In the previous sections we focused on the comparison of various state assignment techniques — the main subject of this thesis. To obtain a full picture of the synthesis, we performed additional evaluation of the complete synthesis chains — from FSM transition table to mapped LUT network.

As we indicated in Section 5.1, good cooperation between state assignment and combinational synthesis is essential to obtaining good quality results. Both steps of the synthesis trajectory need to have similar objectives and follow compatible heuristics. Otherwise, state assignment may produce functions difficult for synthesis and the combinational synthesis may not fully realize optimization potential introduced by the state assignment. Therefore, in this section we present the results of the experiments with complete synthesis chains, i.e. state assignment and combinational synthesis methods implemented in the same packages or following the same heuristics.

In Table 6.6 we present the results of comparison of four synthesis chains: information-driven, decomposition-based flow represented by SECODE and IRMA2FPGA; division-based flow represented by JEDI and SIS; and two popular commercial FPGA synthesis

tools, denoted A and B. In all cases, the input to the flow was transition table of the FSM expressed in KISS format or equivalent Verilog description (for commercial tools). All four synthesis chains were then used to perform state assignment (commercial tools were setup to automatically determine the best state assignment) and synthesize and map the network. Synthesis setup was as follows: for SIS we used script described in Figure 6.1; since the main target objective of IRMA2FPGA is the delay reduction, for commercial tools we used the highest available effort settings for speed-driven optimization and the target clock speed of 1 GHz. The resulting networks were placed and routed on VirtexII-1000-fg256-4 device with Xilinx ISE 5.2. The results are reported as the number of slices (column *sl*) and clock period (column *clk*) in the resulting circuit.

As we can see, the uniform, information-driven sequential synthesis chain significantly outperforms all other synthesis flows. In particular, division-based approach produced circuits in total 60% larger and slower than our methods. The comparison is more favorable for the evaluated commercial methods. Tool A produced circuits 15% larger and 7% slower than the decompositional methods. Tool B managed to produce circuits fraction faster than our methods, but at the expense of 50% area increase.

We should note, however, that the actual performance of state assignment and logic synthesis alone are difficult to evaluate in case of commercial tools. Their closed architecture does not allow to precisely determine what steps are performed to produce the final network. These tools utilize additional optimization steps at the network level, such as retiming, and the intimate knowledge of the target architecture, to produce best circuits. In particular, they are able to produce networks composed not only of LUTs, but also of specific dedicated resources available at the target FPGA device (e.g. additional logic gates, carry chains, special fast local interconnections, etc.). In comparison, IRMA2FPGA, or SIS produce very simple networks composed of basic building blocks alone. It is very likely that if our synthesis methods were better integrated in the mapping process and thus were able to additionally benefit from device-dependent optimizations performed by commercial tools, the results would significantly improve.

	secode+irma		jedi+sis		A		B	
	sl	clk	sl	clk	sl	clk	sl	clk
a02	25	4,274	37	6,914	42	5,414	45	5,394
a03	10	2,818	23	8,255	12	3,138	32	4,997
a04	39	5,668	69	11,076	56	7,791	42	4,248
a05	7	3,525	13	4,932	21	3,549	19	3,129
bbara	9	3,569	14	5,798	8	3,285	13	4,136
bbsse	22	3,971	30	7,06	25	4,362	19	4,07
bbtas	4	3,201	4	3,207	3	1,93	5	2,146
beecnt	6	2,647	7	3,957	10	3,227	12	3,397
coffee	13	4,111	19	6,543	12	4,285	11	3,653
cse	42	6,305	47	7,96	41	5,815	34	4,21
dk14	12	3,361	19	3,67	24	3,856	25	3,961
dk15	7	2,633	7	3,254	7	2,389	17	3,801
dk17	6	3,215	6	2,674	10	3,333	12	3,399
dk27	3	1,967	3	2,423	3	2,184	4	2,815
dk512	7	3,215	6	3,207	7	2,593	9	2,852

ex1	37	6,323	82	8,099	52	5,294	44	5,126
ex4	12	4,362	11	3,715	16	4,46	14	2,474
ex5	5	3,031	8	4,304	10	3,624	11	4,113
ex6	16	5,4	28	6,554	25	4,766	20	4,171
example	5	2,609	6	3,358	9	3,667	2	1,864
keyb	35	9,338	56	9,934	37	7,493	56	7,204
lion	2	1,947	2	2,41	2	1,947	4	2,951
lion9	3	3,205	3	3,211	10	3,073	8	2,485
mark1	14	3,515	16	4,67	12	4,406	18	3,182
mc	7	2,664	3	1,951	4	2,732	7	2,048
opus	12	3,39	19	5,715	15	3,834	15	2,869
opus2	14	3,922	23	5,989	16	4,172	20	4,613
patgen	23	4,582	28	7,489	21	2,942	27	2,868
patrec	32	4,381	48	9,873	33	6,323	33	5,23
percent	6	3,198	14	5,651	8	3,285	13	4,136
planet	62	4,077	155	11,068	64	3,697	79	3,819
pma	30	6,453	42	7,404	42	6,429	40	4,498
s1	20	6,291	76	12,857	56	4,341	56	5,331
s1488	78	5,039	164	10,894	89	6,252	159	6,001
s1494	79	4,254	160	10,796	90	6,04	197	5,039
s208	11	3,209	15	5,024	26	3,399	33	5,641
s27	3	2,605	4	3,073	10	4,936	11	3,059
s386	19	3,933	30	6,06	26	3,889	22	4,015
s420	12	2,315	17	5,171	10	3,723	18	5,102
s510	41	3,209	69	12,643	34	11,653	47	3,208
s8	1	0	8	4,352	3	3,012	9	2,848
s820	44	4,845	83	14,031	51	7,144	67	6,172
s832	49	5,682	89	13,325	49	5,38	66	6,8
sand	163	12,962	142	10,766	92	6,028	113	5,952
shiftreg	1	1,948	2	2,388	1	1,913	5	1,973
sse	22	3,971	30	7,06	24	4,454	19	5,131
tav	6	2,211	6	2,3	5	2,232	8	2,217
tbk	47	6,468	62	10,574	73	7,354	196	8,886
tma	18	4,111	29	5,913	31	5,336	30	3,528
train11	2	1,947	4	0	8	3,214	8	2,472
vtiidec	41	4,089	56	6,078	44	3,78	37	2,83
vtiuar	46	5,848	87	11,945	30	3,778	39	4,904
Σ	1230	211,814	1981	337,575	1409	227,153	1850	210,968
$\Delta\%$			61,1	59,4	14,6	7,2	50,4	−0,4

Table 6.6. Comparison of synthesis chains

6.3 Generated benchmarks

In the previous sections, we performed various comparisons of sequential and combinational synthesis methods on the set of standard IWLS benchmarks. To further evaluate

the effectiveness of our encoding method on a larger set of benchmarks with various characteristics, we developed together with Lech Jóźwiak and Dominik Gawlowski the benchmark generation software BENGEN. With this tool, we generated 350 FSMs exhibiting characteristics typical to the circuits encountered in various industrial applications, and compared the different encoding results for these machines.

In the following, we will shortly introduce the benchmark generator BENGEN. The more detailed discussion of the tool can be found in [42]. Further, we will discuss the characteristics of the generated benchmarks and analyze the results of encoding of the machines with various encoding methods.

6.3.1 BENGEN benchmark generator

The benchmark generator was developed in response to the shortage of typical industrial benchmarks. Such benchmarks are necessary to evaluate the effectiveness of the state assignment method on relevant machines commonly encountered in real-life systems. Furthermore, to assess robustness of a method, it is necessary to have access to a large number of benchmarks. Unfortunately for the EDA research community, such benchmarks are guarded by the designers of the systems, as well as by the vendors of EDA tools that usually have access to the circuit libraries of their clients.

To address this issue, we developed the benchmark generator BENGEN that enables us to generate large sets of FSMs with various characteristics. These include:

- FSMs with different number of states and various transition patterns between the states (e.g. chains of states with forward and/or backward transitions, loops, conditional "case" structures etc. and their combinations);
- FSMs with different numbers of inputs and outputs, and different proportions between the next-state and output logic (state-dominated, balanced or output-dominated), as well as, between the primary-input and state-input (input-dominated, balanced, state-dominated), and their mixtures;
- FSMs with various dependence of particular transitions and output variables on the number of inputs and input conditions;
- completely, incompletely and weakly specified FSMs.

This also includes FSMs representative to various typical industrial application areas, as for instance, having typical structure of controllers from various application areas, or representing various sequential data-path circuits (e.g. counters).

BENGEN enables us to efficiently construct FSMs, but also to modify the constructed or industrial FSMs, and to very precisely "fine-tune" the benchmarks. This last feature is extremely useful in sensitivity analysis of state assignment to the changes in the input data, i.e. small changes in the FSM characteristics.

The generation process is based on guided random generation of *branches* of an FSM. A branch is a series of states following one another, with possible backward transitions from next states back to the previous, and state self-loops indicating that FSM stays in the same state. By choosing the first and the last state of different loops, their length and patterns of backward- and self-transitions, arbitrary FSM state transition structures

can be constructed. The primary input values that trigger particular transitions are also generated according to user preferences.

BENGEN has two work modes: *batch* and *interactive* mode. In the *batch mode*, the parameters of the FSM to generate are supplied in a script file. These parameters include: the number of inputs and outputs of the FSM; the number and length of the FSM's branches; the characteristics of a branch, such as the number of backward transitions or loops; the number of inputs and outputs active for a given branch, etc. Most of these parameters can be specified in the form of a probability distribution to randomize their values in the specific instances of the generated FSMs. As a result, in the batch mode BenGen can be used to easily generate large sets of FSM benchmarks with certain characteristics using the same script file. The script file can be easily modified to generate a next batch of somewhat different FSMs.

The *interactive mode* of BenGen provides more control over the generation process. The user can interactively enter any of the parameters available in the batch mode. In addition to that, operations allowing modifications of single branches, or transitions are provided. This makes BENGEN in interactive mode an ideal tool for fine-tuning of the generated or industrial FSMs for the specific characteristics, required for instance to check the behavior or sensitivity of a method or tool in relation to a certain aspect.

In both modes, BENGEN takes away the burden of tedious specification of single transitions, or checking the consistency of the constructed FSM. Instead, the user can focus on specifying high-level characteristics of the machine. Given the global machine's characteristics, BENGEN generates the required number of state chains, with the requested number of states, and appropriate backward transitions between and self-loops in the states, if needed. Given the state-transition behaviour defined in abstract terms, BENGEN generates the input conditions for particular transitions, etc. In this process not only does it consider user's requirements concerning the active inputs for a given branch, but also ascertains that the machine is consistent, i.e. distinct transitions have disjoint input conditions and all possible input conditions are specified (for completely specified FSMs). The generator also determines the output values for the transitions, taking into account the outputs active for a given branch.

With the above characteristics, BENGEN is a very useful tool enabling efficient generation of an arbitrary number of various sorts of well-characterized FSM benchmarks, with the minimum effort of the user. On the other hand, it also enables fine-grained control over the generation process and editing of the generated or industrial FSMs. Appropriately used, BENGEN enables generation of FSMs representative to many practical applications. Examples of such machines are discussed in the following section.

6.3.2 Experimental results for generated benchmarks

To evaluate the effectiveness of the proposed state assignment method, we used BENGEN to generate 350 FSMs that exhibit characteristics typical to FSMs encountered in various real-life industrial applications. We identified a number of typical schemes of sequential behavior and for each scheme generated a set of benchmarks of different sizes and with differing proportions of input, state and output logic. For instance, the sequential behavior schemes included: a single loop of states (typical for a counter or simple sequencer); a number of loops starting from a common initial state (typical for a controller realizing a few different control programs for different operation modes); a single loop with

sub-loops attached to states along the main loop (a main control "program" with "subroutines" - the subroutine loops may have their own sub-subroutine loops); and more complicated cases of sequential behavior. Within each scheme, we varied proportions of backward transitions and state self-loops within loops and branches generated.

The generated machines were encoded with four encoding methods: JEDI, SECODE, one-hot and sequential binary encoding. The results of logic synthesis of the encoded machines with IRMA2FPGA are presented in detail in Appendix A. For clarity of presentation in this section, we categorized the generated machines according to three criteria: the size, the proportion of the number of primary input bits to the number of state bits, and the proportion the number of primary output bits to the number of state bits. The size criterion divides FSMs into *small* (max. 8 states), *medium* (9 to 32 states) and *large* (more than 32 states). The proportion of the number of primary input/output bits to state bits categorizes the FSMs as *input/output dominated* if the number of input/output bits is 50% larger than the number of state bits, *state dominated* if the number of state bits is larger than the number of input/output bits and *balanced* otherwise. In the generated benchmark, we made sure that each category is well represented, with some discrepancies between the number of machines in each category consistent with the occurrence frequency in real-life applications. Numbers of machines in each category are summarized in Table 6.7, in the row marked with $\#$.

The overall picture of efficiency of SECODE encoding is consistent with the results achieved for standard IWLS benchmarks. Out of 350 generated benchmarks, SECODE produced encodings resulting in smaller realization circuits than JEDI in 278 cases (80%), 153 of these with the improvement of over 10%. In 44 cases, the results for both encoding methods were equal, and only for 28 machines (8%), JEDI outperformed SECODE. The circuits encoded with SECODE achieved area results between the reduction of 40% and 15% area increase. Globally, the circuits encoded with SECODE were 7% smaller and 4% faster than those encoded with JEDI.

size	small		medium		large	
#	87		219		44	
Δ	-10.3%	-4.9%	-9.2%	-3.3%	-4.4%	-5.2%
input	input dom.		balanced		state dom.	
#	105		156		89	
Δ	-9.3%	-6.1%	-5.9%	-1.6%	-5.3%	-4.2%
output	output dom.		balanced		state dom.	
#	125		149		76	
Δ	-5.4%	-5.0%	-9.4%	-3.8%	-8.2%	-2.4%

Table 6.7. Summary results for generated benchmarks

Table 6.7 summarizes the comparison between the results achieved with SECODE and JEDI encodings in each of the identified FSM categories — with respect to size, input and output characteristics. For each category, in the row marked with Δ, two numbers indicate the difference within the category between the size and the depth of the circuits for FSMs encoded with JEDI and SECODE. The analysis of the results confirms that SECODE is a stable method achieving good results regardless of the characteristics of the encoded FSM. The overall area improvement of 7% fluctuated between the categories from 4.4%

to 10.3%. However, SECODE still produced better results that JEDI in all categories. Also the delay improvement remained in all categories in the region of overall improvement of 4%, ranging from 2.6% to 6.1%.

In another comparison, we augmented our encoding approach with the method for identifying efficient one-hot encodings described in Section 5.5.1. The FSMs identified by the method as having potentially efficient one-hot encoding are marked in the table in Appendix A with "+" in the last column ($h+$). As we can see in the table, the method has correctly predicted efficiency of one-hot encoding in 301 out of 350 of the cases (86%). When the FSMs identified as having good one-hot encoding are encoded with one-hot instead of SECODE, the improvement of the area over JEDI grows from 7% for SECODE alone to 8% for the combined SECODE/one-hot. Unfortunately, this 1% of area reduction is accompanied by 1% delay increase — from 4% reduction to 3% reduction.

The relatively small improvement achieved by introduction of one-hot results is the consequence of the overall poor performance of one-hot encoding. On the generated benchmarks, only for 38 machines one-hot produced better circuit than SECODE. In total, one-hot encoding increased the area by over 9% when compared with JEDI and over 17% when compared with SECODE. The delay was also only marginally (under 1%) better when compared with JEDI, but clearly worse (4%) when compared with SECODE. These results confirm our observation that one-hot encoding is in general not effective for a wide variety of FSM types. As discussed earlier and confirmed by these results, only a small proportion of specific FSMs can benefit from simplified function form achieved in one-hot encoding sufficiently to offset the increased number of output functions and inputs to the functions introduced by this encoding.

Not surprisingly, even worse results were recorded for sequential binary encoding. Consistently with the results for IWLS benchmarks, binary encoding produced circuits 21% larger and 15% slower than JEDI and 30% larger and 20% slower than SECODE. An aspect that deserves a comment here is the difference between improvement achieved by our method with respect to essentially random binary encoding in the case of industrial IWLS benchmarks and the generated benchmarks (40% versus 30%). This discrepancy is the result of special characteristics and additional optimization opportunities encountered in real-life FSMs that are very difficult to emulate in partially random machines. However, the fact that the overall trend remains the same for IWLS and generated benchmarks supports our claim that the machines generated with BENGEN reflect well the characteristics of real-life industrial circuits.

Even assuming that for each machine full synthesis is performed for the one-hot and binary encodings and the better result is chosen (even though that would double the synthesis time), the combined methods are still inferior to SECODE. The combined results of the one-hot and binary encoding are on average 10% larger and 6% slower than the circuits resulting from SECODE encoding. In our experiments, next to the two above methods, we also considered Gray encoding — the minimum-length encoding, where the consecutive state codes differ on exactly one code bit. The results for this method (not reported here) show similar picture to the sequential binary encoding. This is consistent with our expectations, as Gray encoding is in fact also a random encoding method that cannot be expected to consistently deliver effective results with respect to area and speed of the synthesized circuits. It is, on the other hand, known to reduce the power consumed by the circuits, provided that the states are assigned consecutive Gray codes in correct order.

Summing up the above analysis, we conclude that our encoding method implemented in the prototype tool SECODE is more effective with respect to the area and speed of the resulting circuits than the most popular current academic (represented by JEDI) and industrial (represented by one-hot, binary and Gray) encoding methods.

Chapter 7

Conclusions and future work

In this thesis we have discussed the problems of decomposition and state encoding of Finite State Machines. In the introduction, we pointed out that FSMs are universally accepted as a functional description of sequential digital circuits. In the face of dynamic growth of the number and importance of digital circuit applications, the efficient synthesis of FSMs in hardware became a problem of primary practical importance. Within this domain, we are particularly interested in hardware realizations in the emerging FPGA-based reconfigurable platforms, such as reconfigurable System-on-Chip (SoC) platforms, and SoCs with embedded FPGAs. These platforms have recently gained much popularity due to their low costs for short and medium production series and inherent flexibility allowing virtually unlimited modifications to the already produced circuit or system. With the logic densities and speed of new generations of reconfigurable platforms growing rapidly, we expect that their challenge to hardwired ASICs will continue and gain momentum.

The **main contributions** of this work are as follows:

- Together with Jóźwiak, I formulated the final version and developed the precise proof of the General Decomposition Theorem of incompletely specified, non-deterministic FSMs with multi-state behavior realization, based on the initial version of this theorem and the outline of its proof proposed by Jóźwiak as an extension of his General Decomposition Theorem for completely specified FSMs [35]. This theorem states conditions for a legal decomposition of an FSM into a network of cooperating partial FSMs and this way defines the most general known generator of correct circuit structures (decompositions) for FSMs. It covers as special cases all other known decomposition structures, such as serial or parallel FSM decomposition, and the decomposition of discrete functions and relations. This way, a sound theoretical base was created for research in decomposition of FSMs and discrete relations.
- Based on the extended GDT, we expressed the state assignment of an FSM as a special case of the general decomposition and formulated the problem of state assignment in terms of finding and appropriately implementing of some specific collections of two-block set systems, or dichotomies, to define the corresponding partial machines and their interconnections. We have shown that the new formulation allows for more flexibility in state assignment than in the previously proposed

methods, permitting among others explicit consideration of information flows and their relationships and allowing for incompletely specified, overlapping codes. The formulation enables the explicit consideration of the interconnection structure, input supports of particular partial machines and sub-functions, as well as implicit behavior-preserving optimizations, such as state minimization during the state assignment process.

- Based on the new formulation of the conditions for a valid state assignment, we proposed a generic state assignment method. The method constructs a valid encoding by forming the encoding dichotomies as a combination of smaller "atomic" dichotomies. In this approach, very detailed analysis of information dependencies is possible, using simple and efficient means of input support minimization. Also, by merging atomic dichotomies any encoding can be constructed, including encodings with incompletely specified, overlapping state codes. This features allows exploitation of the encoding freedom afforded by the flexible formulation of the conditions for valid state assignment discussed above.

- On top of the generic state assignment method, we have proposed a system of novel heuristics that guide the encoding construction towards encoded FSMs that have efficient implementation in FPGAs. The heuristics are based on the analysis and optimization of the information flows within the encoded machine. This analysis is supported by the application of the apparatus of Information Relationships and Measures. The generic state assignment method equipped with the heuristics constitutes a complete effective and efficient FPGA-targeted FSM state encoding method.

- I implemented the FPGA-targeted FSM state assignment method in the form of the prototype EDA software tool – SECODE – that significantly outperforms other popular academic and industrial state assignment approaches and tools by producing encodings resulting in smaller and faster FPGA realizations of the FSMs.

- As a result of the above developments and experimental research, we demonstrated that the information-driven approach based on the general decomposition and the apparatus of information relationships and measures is an effective and efficient approach to sequential circuit synthesis.

While in this thesis we applied the theories of general decomposition and information relationships and measures to state assignment of FSMs, the practical usefulness of these theories extends far beyond that problem. In particular, the theories have been successfully applied in a separate work to circuit synthesis of Boolean functions [41]. Furthermore, general decomposition can be applied in any field of modern engineering and science that deals with finite state machines, discrete functions and relations.

Other contributions of the work include:

- We have evaluated and debunked the common conviction that any particular encoding method, be it one-hot, sequential binary or any other particular sort of semi-random encoding can universally produce a high-quality encoding, especially in relation to the FSM implementations in FPGAs, with their complex characteristics. Various encodings are good for various FSMs, different implementation technologies and various optimization objectives. Even for a particular implementation

technology and specific objectives (as in the case of FPGA implementations considered in this thesis), a specific encoding (e.g. one-hot or Gray) is only good for a certain specific subset of FSMs.

- The above argument notwithstanding, we recognize that for a certain class of FSMs, one-hot encoding can lead to efficient FPGA implementation in terms of area and speed. We have analyzed this problem and proposed a reasonably effective heuristic to identify good candidates for one-hot encoding prior to performing the actual encoding. Thus, we are able to find efficient encodings for a group of FSMs at a fraction of the effort required to perform the full encoding.

- To address the issue of low availability of industrial FSM benchmarks, we developed together with L. Jóźwiak and D. Gawlowski the benchmark generator software BENGEN. This software enables efficient generation of an arbitrary number of various sorts of well-characterized FSM benchmarks, with the minimum effort of the user, while retaining fine-grained control over the generation process and editing of the generated or industrial FSMs.

- The implementation of the prototype encoding tool SECODE resulted in the development of a software library that efficiently implements various operations frequently encountered in logic synthesis, and in particular in decomposition and state assignment. The contents of the library ranges from basic data structures, such as graphs or efficient vectors of Boolean and logic variables with their related operations, to generic and extensible implementations of algorithms for problems from graph theory, clustering, simulated annealing, etc. Within the library, a number of generic algorithms was extended with heuristics relevant to the subject field. For instance, the twin graph coloring approach was extended with the heuristics for efficient merging of dichotomies, while considering their mutual affinities. Also, an incremental input support minimization routine was developed that saves significant computing effort by evaluating input supports of a modified output function as a derivative of the original function, instead of fully recomputing the input support.

  The library can be useful for further research in this field, providing basis for rapid development of new algorithms and enabling quick experimental evaluation of new research ideas. Also, the SECODE software itself is a good framework for further analysis and experiments with encoding and decomposition of FSMs.

While the theory of general decomposition, which has been developed during the past 15 years, is quite mature, the newly developed state assignment method and software is in the prototype stage and would benefit from **further development**.

In particular, an important role in the analysis process is played by the input support computation. Quite often, especially in optimized machines, there is a unique input support for a given output function. In other cases, however, there is a choice of several minimum or close-to-minimum input supports for the given function. In these cases, the selection of one of the supports may have influence on further analysis and hence on the final encoding. It would be interesting to thoroughly investigate and exploit the influence of these choices on the results. A particularly interesting choice is between the primary and state variables in the support. It may occur that, for instance, two primary

input variables in the minimum input support may be replaced with several atomic state variables. In terms of decomposition, it means that the partial machine imports some state/input information from other partial machines rather than deriving it from the primary input information. The input support with state variables is then no longer minimum. However, after the atomic state variables are merged, it may turn out that the atomic state variables in the input support are merged to a single encoding variable and, therefore, in the final input support the two primary inputs are replaced with a single merged state variable. In this context, it is even possible to store multiple input supports for an output function and determine its affinity to other functions based on the support that fits best with the other function.

It would be also beneficial to continue research into predicting good specific encodings, such as one-hot or Gray encoding. More sophisticated cost estimation for one-hot encoding are certainly possible. Also, for Gray encoding, in which the consecutive codes have Hamming distance-1, one could analyze the best order of states for a given machine.

An important issue is compatibility of the state assignment method with the combinational synthesis method used to synthesize the encoded FSMs. As discussed in Chapter 5, the state assignment method has to be aware of the combinational synthesis to be able to produce binary functions that will have efficient implementations when using a particular combinational synthesis method targeting a particular implementation technology. Therefore, the encoding method would certainly benefit from further research into the desired characteristics of Boolean functions that yield efficient FPGA realizations. In particular, such research would be beneficial for the case of our combinational method of choice – the functional decomposition implemented in software tool IRMA2FPGA. Knowing more precisely characteristics of functions that are "simple" for IRMA2FPGA to synthesize would make it possible to favor such functions when encoding the FSM.

Finally, there is a number of possible optimization objectives that can drive the state assignment process. In our method we addressed the objectives that are important for most applications – the area and the delay of the circuit implementation. As discussed in Chapter 5, the area is addressed by limiting the interconnection and compressing the information relevant to particular functions on as few input variables as possible, and in this way reducing the input supports. Delivering compressed, relevant information to functions also tends to simplify the processing of the information, thus it may reduce the depth and hence the delay of the circuit. Delay is also reduced by limiting interconnections and especially avoiding long interconnections. In the further research on the topic of state assignment, some other objectives could be considered, such as power dissipation or testability of the circuit, even though they are perhaps more relevant to other target implementation platforms, such as ASICs involving CMOS gate networks. The other objectives can be reasonably easy to account for in our approach by adding some extra constraints and heuristics to the existing FSM assignment method and tool, and should not require any changes to the underlying theories or generic assignment method.

Summing up, the research presented in this thesis resulted in a sound theoretical base for research in FSM decomposition and encoding, generic information-driven FSM state assignment method, complete heuristic FPGA-targeted state assignment method and the prototype EDA software tool that implements the method. Using our encoding tool and a large set of benchmark FSMs, we performed an extensive experimental

research. The results of the experimental research clearly demonstrate that our tool significantly outperforms other popular academic and industrial FSM state assignment approaches and tools. Consequently, the research presented in this thesis demonstrates that the information-driven approach to circuit synthesis based on the general decomposition and information relationships and measures is an effective and efficient approach to sequential circuit synthesis. This way, the aims of the research reported in this thesis have been fully realized.

Bibliography

[1] A. E. A. Almaini, J. F. Miller, P. Thomson, and S. Billina. State assignment of finite state machines using a genetic algorithm. *IEE Proc. on Computers and Digital Techniques*, pages 279–286, July 1995.

[2] D. B. Armstrong. On the efficient assignment of internal codes to sequential machines. *IRE Trans. on Electronic Computers*, pages 611–622, October 1962.

[3] D. B. Armstrong. A programmed algorithm for assigning internal codes to sequential machines. *IRE Trans. on Electronic Computers*, pages 466–472, August 1962.

[4] P. Ashar, S. Devadas, and A. R. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, 1992.

[5] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1997.

[6] E. Bruce Lee and M. A. Perkowski. Concurrent minimization and state assignment of finite state machines. *Proc. of Int. Conf. on Systems, Man and Cybernetics*, pages 248–260, October 1984.

[7] J. A. Brzozowski and J. J. Lou. *To check*, chapter Blanket algebra for multiple-valued function decomposition, pages 262–276. To check, 2000?

[8] J.A. Brzozowski and T. Luba. *Decomposition of Boolean Functions Specified by Cubes*. University of Waterloo Research Report, CS-97-01, Waterloo, Canada, January (revised October 1998), 1997.

[9] M. Burns, M. Perkowski, and L. Jóźwiak. An efficient approach to decomposition of multi-output boolean functions with large sets of bound variables. In *Proc. EUROMICRO-98 Conference, Vasteras, Sweden*, pages 16–23, 1998.

[10] S. Chattopadhyay and P. Pal Chaudhuri. Genetic algorithm based approach for integrated state assignment and flipflop selection in finite state machine synthesis. *Proc. of Int. Conf. on VLSI Design*, pages 522–527, 1997.

[11] M. J. Ciesielski and J. Shen. A unified approach to input-output encoding for FSM state assignment. *Proc. of 28th Design Automation Conf.*, pages 176–181, 1991.

[12] O. Coudert. A new paradigm for dichotomy-based constrained encoding. In *Proc. Design, Automation and Test in Europe*, pages 830–834, 1998.

[13] G. de Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Trans. on CAD*, pages 269–284, 1985.

[14] S. Devadas, H. TonyMa, A. R. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: state assignment of finite state machines for optimal multi-level logic implementation. *Proc. of Int. Conf. on CAD*, pages 16–19, 1987.

[15] T. A. Dolotta and E. J. McCluskey. The coding of internal sates of sequential circuits. *IEEE Trans. on Electronic Computers*, pages 549–562, October 1964.

[16] X. Du, G. Hachtel, B. Lin, and A. R. Newton. MUSE: a multilevel symbolic encoding algorithm for state assignment. *IEEE Trans. on CAD*, pages 28–38, January 1991.

[17] M. D. Durand and S.R. White. Trading accuracy for speed in parallel simulated annealing with simultaneous moves. *Parallel Computing*, 26(1):135–150, 2000.

[18] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL – Formal Object-oriented Language for Communicating Systems*. Prentice-Hall, 1997.

[19] G. Files and M. Perkowski. Multi-valued functional decomposition as machine learning method. In *Proc. ISMVL'98, Fukuoka, Japan.*, pages 173–178, 1998.

[20] T.G.W. Gordon and P.J. Bentley. On evolvable hardware. In S. Ovaska and L. Sztandera, editors, *Soft Computing in Industrial Electronics*, pages 279–324. Springer Verlag, 2002.

[21] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[22] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, 1966.

[23] J. Hartmanis and R.E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Englewood Cliffs, N.J.: Prentice-Hall., 1966.

[24] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[25] W. S. Humphrey. *Switching Circuits with Computer Applications*. McGraw-Hill, New York, 1958.

[26] IEEE. *IEEE Standard for Verilog Hardware Description Language*. IEEE, 2001.

[27] IEEE. *IEEE Standard VHDL Language Reference Manual*. IEEE, 2002.

[28] ISO/TC97/SC21. *Information processing systems – Open systems interconnection – Estelle – a formal description technique based on an extended state transition model*. IS 9074, International Organization for Standardization, 1997.

[29] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[30] L. Jóźwiak. *Minimal realization of sequential machines: The method of maximal adjacencies, EUT-Report 88-E-209*. Eindhoven Univ. of Tech., the Netherlands, 1988. ISBN 90-6144-209-5.

[31] L. Jóźwiak. Efficient suboptimal state assignment of large sequential machines. *Proc. of EDAC*, pages 536–541, 1990.

[32] L. Jóźwiak. Simultaneous decomposition of sequential machines. *Microprocessing and Microprogramming*, 30:305–312, 1990.

[33] L. Jóźwiak. An efficient heuristic method for state assignment of large sequential machines. *Journal of Circuits, Systems and Computers*, 2(1):1–26, 1992.

[34] L. Jóźwiak. Decompositional logic synthesis: Correctness aspects. In *Proc. APCHDLSA - 93: Asian Pacific Conference on Hardware Description Languages, Standarts and Applications, Brisbane, Australia*, 1993.

[35] L. Jóźwiak. General decomposition and its use in digital circuit synthesis. *VLSI Design*, 3(3):225–248, 1995.

[36] L. Jóźwiak. Information relationships and measures - an analysis apparatus for efficient information system synthesis. In *Proc. of the 23rd EUROMICRO Conference, Budapest, Hungary*, pages 13–23, 1997.

[37] L. Jóźwiak. Information relationship measures in application to logic design. In *Proc. IEEE International Symposium on Multiple-Valued Logic, Freiburg Im Breisgan, Germany*, 1998.

[38] L. Jóźwiak and S. Biegański. Information-driven library-based circuit synthesis. In *Proc. Euromicro Symposium on Digital Systems Design*, pages 148–155, 2003.

[39] L. Jóźwiak and A. Chojnacki. Effective and efficient FPGA synthesis through functional decomposition based on information relationship measures. In *Proc. DSD'- Euromicro Symposium on Digital System Design*, pages 30–37, 2001.

[40] L. Jóźwiak and A. Chojnacki. High-quality sub-function construction in functional decomposition based on information relationship measures. In *Proc. DATE'- Design, Automation, and Test in Europe Conference, Munich, Germany*, pages 383–390, 2001.

[41] L. Jóźwiak and A. Chojnacki. Effective and efficient combinational circuit synthesis for the FPGA-based reconfigurable systems. *Special Issue of Journal of Systems Architecture on Reconfigurable Computing*, 49(4-6):247–265, 2003.

[42] L. Jóźwiak, D. Gawlowski, and A. Ślusarczyk. An effective solution of benchmarking problem fsm benchmark generator and its application to analysis of state assignment methods. In *accepted for Euromicro Symposium on Digital System Design*, 2004.

[43] L. Jóźwiak and J.C. Kolsteren. An efficient method for the sequential decomposition of sequential machines. *Microprocessing and Microprogramming*, 32:657–664, 1991.

[44] L. Jóźwiak and P. Konieczny. Input support minimization for efficient PLD and FPGA synthesis. In *Proc. International Workshop on Logic and Architecture Synthesis*, pages 30–32, 1996.

[45] L. Jóźwiak and P. Konieczny. Input support minimization for efficient PLD and FPGA synthesis. In *Proc. IWLAS*, pages 30–37, 1996.

[46] L. Jóźwiak and A. Postula. Genetic engineering versus natural evolution: Genetic algorithms with deterministic operators. *Proc. of Int. Conf. on Artificial Inteligence IGAI'99*, pages 58–64, 1999.

[47] L. Jóźwiak and A. Ślusarczyk. A new state assignment method targeting FPGA implementations. In *Proc. EUROMICRO Symposium on Digital System Design DSD, Maastricht, the Netherlands*, pages 50–59, 2000.

[48] L. Jóźwiak and A. Ślusarczyk. General decomposition of incompletely specified sequential machines with multi-state behavior realization. *to be published in Journal of Systems Architecture on Reconfigurable Computing*, 2004.

[49] L. Jóźwiak, A. Ślusarczyk, and A. Chojnacki. Fast and compact sequential circuits through the information-driven circuit synthesis. In *Proc. DSD'01- Euromicro Symposium on Digital System Design, Warsaw, Poland*, pages 46–53, 2001.

[50] L. Jóźwiak, A. Ślusarczyk, and A. Chojnacki. Fast and compact sequential circuits for the FPGA-based reconfigurable systems. *Special Issue of Journal of Systems Architecture on Reconfigurable Computing*, 49(4-6):227–246, 2003.

[51] L. Jóźwiak and F. Vankan. Bit full decomposition of sequential machines: Algorithms and results. In *Proc. of the Canadian Conference on Electrical and Computer Engineering, Montreal*, 1989.

[52] L. Jóźwiak and F. Volf. An efficient method for decomposition of multiple output boolean functions and assigned sequential machines. In *Proc. EDAC - The European Conference on Design Automation, Brussels, Belgium*, pages 114–122, 1992.

[53] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data*. Wiley, 1990.

[54] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[55] L. Lavagno, S. Malik, R. K. Brayton, and A. Sangiovanni-Vincentelli. MIS-MV: optimization of multi-level logic with multiple-valued inputs. *Proc. of Int. Conf. on CAD ICCAD'92*, pages 560–563, 1992.

[56] I. Lemberski. Modified approach to automata state encoding for LUT-FPGA implementation. *Proc. of of 24th Euromicro Conf.*, pages 196–199, 1998.

[57] B. Lin and A. R. Newton. Synthesis of multiple level logic from symbolic high-level description languages. *Proc. of IFIP Int. Conf. on VLSI*, pages 187–196, 1989.

[58] T.Y. Lin and N. Cercone(Eds.). *Rough Sets and Data Mining - Analysis of Impresise Data*. Kluwer, 1997.

[59] T. Luba. Decomposition of multiple-valued functions. In *Proc. IEEE ISMVL'95, Bloomington, Indiana, USA*, 1995.

[60] T. Luba and J. Rybnik. *Rough Sets and Some Aspects of Logic Synthesis*. in R. Slowinski (Ed.): Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory, Kluwer., 1993.

[61] M. Martinez, M. J. Avedillo, J. M. Quintana, and J. L. Huertas. A dynamic model for the state assignment problem. *Proc. of DATE Conf.*, pages 835–839, 1998.

[62] K. McElvain. *IWLS'93 Benchmark Set: Version 4.0.* Distributed as a part of IWLS'93 benchmark set, 1993.

[63] J. Monteiro, J. Kukula, S. Devadas, and H. Neto. Bitwise encoding of finite state machines. *Proc. of 7th Conf. on VLSI Design*, pages 379–382, 1994.

[64] M. Perkowski, M. Marek-Sadowska, L. Jóźwiak, T. Luba, S. Grygiel, M. Nowicka., R. Malvi., Z. Wang, and S. Zhang. Decomposition of multiple-valued relations. In *Proc. International Symposium on Multiple-Valued Logic*, pages 13–18, 1997.

[65] M. Rawski, L. Jóźwiak, and T. Luba. The influence of the number of values in sub-functions on the effectiveness and efficiency of the functional decomposition. In *Proceedings of the 25th EUROMICRO Conference*, September 8-10 1999. Milan, Italy.

[66] M. Rawski, L. Jóźwiak, and T. Luba. Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures. *Journal of Systems Architecture*, 2(47):137–155, 2001.

[67] M. Rawski, L. Jóźwiak, M. Nowicka, and T. Luba. Non-disjoint decomposition of boolean functions and its application in FPGA-oriented technology mapping. In *Proc. of the EUROMICRO'97 Conference, Budapest, Hungary*, pages 24–30, 1997.

[68] T. Ross, M. Noviskey, T. Taylor, and D. Gadd. *Pattern Theory: An Engineering Paradigm for Algorithm Design.* Final Technical Report, Wright Laboratories, WL/AART/WPAFB, 1991.

[69] C. Sarwary, E. Prado Lopes, L. Burgun, and A. Greiner. FSM synthesis on FPGA architectures. *Proc. of 7th IEEE ASIC Conf.*, pages 178–181, 1994.

[70] H. Selvaraj, H. Niewiadomski, M. Pleban, and P. Sapiecha. Decomposition of digital circuits and neural networks. In *Proc. Special Sessions on Modern Digital System Synthesis at The Fifth Multi-Conference on Systemics, Cybernetics and Informatics - SCI'2001, Orlando, USA*, pages 302–307, 2001.

[71] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Synthesis, Memorandum No. UCB/ERL M92/41.* University of California, Berkeley, 1992.

[72] J. Shen, Z. Hasan, and M. J. Ciesielski. State assignment for general FSM networks. *Proc. of EDAC*, pages 245–249, 1992.

[73] M. Venkatesan, H. Selvaraj, and R. Bignall. Character recognition using functional decomposition. In *Proc. International Conference on Computational Intelligence and Multimedia Applications*, pages 721–726, 1998.

[74] T. Villa and A. Sangiovanni-Vincentelli. NOVA: state assignment of finite state machines for optimal two-level logic implementation. *IEEE Trans. on CAD*, pages 905–924, 1990.

[75] F.A.M Volf. *A bottom-up approach to multiple-level logic synthesis for look-up table based FPGAs.* Technische Universiteit Eindhoven, 1997. Ph.D. thesis.

[76] M. K. Yajnik and M. J. Ciesielski. Finite state machine decomposition using multiway partitioning. *Proc. of Int. Conf. on Computer Design: VLSI in Computers and Processors, ICCD'92,* pages 320–323, 1992.

Appendix A

Encoding results for generated FSMs

The following table summarizes the synthesis results for generated benchmark FSMs encoded with four encoding methods: popular multi-level encoding method JEDI, the method presented in this thesis – SECODE, one-hot and sequential binary encoding. All encoded FSMs were synthesized with IRMA2FPGA [41] logic synthesis method as a network of 5-input LUTs.

For each FSM, the parameters of the machines are listed — number of states (column s), number of primary input (column i) and number of primary outputs (column o). Further, for each encoding methods the results of logic synthesis of the encoded machine are reported as the number of LUTs and number of levels of the LUT-network. Finally, the last column ($h+$) marks with "+" the machines, for which the algorithm discussed in Section 5.5.1 indicated efficient one-hot realization.

name	s	i	o	jedi		secode		hot		bin		h+
				lut#	lvls	lut#	lvls	lut#	lvls	lut#	lvls	
fsm000	19	3	2	24	3	21	2	34	2	31	3	
fsm001	13	4	12	35	2	34	2	41	2	42	3	
fsm002	11	4	4	16	2	16	2	17	2	19	2	
fsm003	6	8	4	20	3	17	2	25	3	25	3	+
fsm004	7	1	1	4	1	4	1	4	1	4	1	
fsm005	8	4	4	16	3	18	3	35	3	21	3	
fsm006	20	2	8	40	2	40	2	45	2	39	2	
fsm007	7	4	4	10	2	7	1	12	2	9	2	
fsm008	12	4	4	37	3	33	3	53	3	48	3	
fsm009	6	4	4	22	2	23	2	33	2	23	2	
fsm010	10	12	16	36	3	34	3	32	3	50	5	+
fsm011	9	4	4	30	3	27	3	43	3	31	3	
fsm012	8	4	4	17	2	17	2	30	2	23	2	
fsm013	15	2	2	14	2	14	2	19	2	16	2	
fsm014	10	4	4	43	3	38	3	65	3	51	3	
fsm015	15	8	4	65	5	46	4	49	3	61	5	

fsm016	20	2	8	23	2	22	2	34	2	24	2	
fsm017	14	2	2	13	2	10	2	16	2	14	2	
fsm018	8	4	4	11	2	10	2	17	2	13	2	
fsm019	6	4	12	20	2	15	2	19	2	19	2	
fsm020	20	2	8	35	2	33	2	37	2	34	2	
fsm021	10	12	16	37	3	27	2	24	2	32	3	+
fsm022	31	2	2	22	2	20	2	36	2	20	2	
fsm023	38	4	10	168	5	132	4	154	5	156	6	
fsm024	12	4	4	25	2	20	2	25	2	26	3	
fsm025	20	2	8	35	2	30	2	42	2	34	2	
fsm026	10	4	4	19	2	16	2	22	2	21	2	
fsm027	17	8	4	96	4	81	4	87	4	126	6	
fsm028	25	2	2	20	2	20	2	32	2	22	3	
fsm029	10	4	4	26	2	21	2	23	2	25	3	
fsm030	8	4	4	17	2	17	2	30	3	19	2	
fsm031	12	8	4	155	7	133	7	171	7	168	7	
fsm032	14	4	4	18	2	18	2	29	2	22	2	
fsm033	30	2	2	22	2	22	2	39	2	24	3	
fsm034	17	8	8	115	4	95	4	104	5	112	4	+
fsm035	13	12	4	46	4	40	3	39	2	57	4	
fsm036	9	4	4	25	3	23	3	35	3	30	3	
fsm037	9	4	4	12	2	11	2	15	2	15	2	
fsm038	15	2	2	17	2	14	2	18	2	19	2	
fsm039	6	8	4	31	4	27	3	42	4	44	4	+
fsm040	12	4	4	44	3	42	3	60	3	43	3	
fsm041	14	4	4	22	2	23	2	29	2	27	3	
fsm042	12	4	4	23	2	17	2	22	2	21	2	
fsm043	12	8	12	94	3	81	4	70	3	96	4	
fsm044	28	3	2	43	3	41	3	42	3	45	3	
fsm045	8	4	12	34	2	32	2	46	2	36	2	
fsm046	8	4	4	12	2	10	2	13	2	13	2	
fsm047	11	4	4	26	2	23	2	26	2	28	2	
fsm048	64	4	4	272	6	269	6	337	8	291	6	
fsm049	37	12	12	226	6	204	4	144	3	315	7	+
fsm050	7	4	4	14	2	13	2	20	2	14	2	
fsm051	8	4	4	25	2	23	2	47	2	26	3	
fsm052	16	4	4	25	2	24	2	31	3	34	3	
fsm053	19	2	2	16	2	17	2	22	2	16	2	
fsm054	12	8	4	38	4	35	3	39	3	51	5	
fsm055	13	8	16	95	3	83	4	85	3	113	4	+
fsm056	40	8	12	106	3	105	4	102	3	118	5	
fsm057	7	1	1	4	1	4	1	6	2	4	1	
fsm058	9	4	12	27	2	24	2	28	2	27	2	
fsm059	36	12	8	95	3	89	4	81	3	113	5	+
fsm060	10	8	2	15	2	11	2	14	2	16	3	
fsm061	39	12	12	481	8	481	9	406	10	1012	9	+

fsm062	7	4	4	10	2	8	2	14	2	10	2	
fsm063	18	4	12	51	3	53	2	62	3	56	3	
fsm064	12	4	4	35	3	35	2	45	3	47	3	
fsm065	14	4	4	25	3	26	3	47	3	35	4	
fsm066	12	8	12	123	4	103	5	136	5	114	4	+
fsm067	13	8	16	52	3	45	3	53	3	57	4	
fsm068	11	4	4	40	3	37	3	58	4	45	3	
fsm069	28	8	4	140	5	132	6	144	8	148	7	
fsm070	10	8	2	15	2	13	2	14	2	18	4	
fsm071	62	4	12	257	5	250	5	216	4	263	5	
fsm072	20	2	8	39	2	38	2	50	3	43	3	
fsm073	7	4	4	10	2	10	2	17	2	15	2	
fsm074	13	4	12	45	2	44	2	46	2	44	2	
fsm075	12	4	4	35	3	32	4	53	4	44	4	
fsm076	5	4	12	20	2	16	2	20	2	17	2	
fsm077	13	4	12	24	2	23	2	31	2	25	2	
fsm078	12	8	12	47	3	39	3	46	3	51	4	
fsm079	19	2	2	16	2	15	2	19	2	20	2	
fsm080	7	4	4	14	2	14	2	24	2	17	2	
fsm081	29	8	4	142	7	133	6	156	7	157	6	
fsm082	7	8	12	31	3	29	2	42	3	39	3	
fsm083	13	8	12	117	5	96	4	101	4	123	4	+
fsm084	21	2	2	24	2	22	2	27	2	23	2	
fsm085	10	4	12	49	2	39	2	45	2	46	3	
fsm086	12	8	12	100	4	79	4	95	5	105	4	
fsm087	5	4	4	19	2	17	2	27	2	20	2	
fsm088	7	8	12	20	2	20	2	27	3	20	2	+
fsm089	8	4	4	15	2	13	2	14	2	14	2	
fsm090	7	4	4	15	2	16	2	31	3	19	3	
fsm091	11	4	12	27	2	26	2	29	2	29	2	
fsm092	12	8	12	108	4	98	4	127	5	107	4	+
fsm093	8	4	4	14	2	14	2	24	2	15	2	
fsm094	14	4	4	49	3	47	3	80	4	54	3	
fsm095	14	8	2	47	4	41	3	46	3	60	4	
fsm096	26	2	2	20	2	19	2	32	2	21	3	
fsm097	7	4	4	21	2	21	2	38	2	22	2	
fsm098	20	8	8	109	4	102	4	102	5	124	7	+
fsm099	7	8	2	15	2	12	2	19	2	20	4	
fsm100	11	4	4	18	2	15	2	22	2	23	3	
fsm101	7	8	8	17	2	13	2	15	2	17	2	+
fsm102	9	4	12	24	2	23	2	28	2	28	2	
fsm103	5	4	4	13	2	12	2	22	2	15	2	
fsm104	56	4	4	262	6	244	6	334	8	310	6	
fsm105	12	4	4	27	2	26	2	33	3	28	3	
fsm106	8	4	12	34	2	26	2	37	2	36	2	
fsm107	7	8	12	34	3	31	3	36	3	36	3	

fsm108	15	8	4	48	3	50	3	48	3	59	4	
fsm109	7	8	2	14	2	15	2	22	2	24	4	
fsm110	15	2	2	16	2	15	2	20	2	19	2	
fsm111	14	2	2	15	2	13	2	20	2	19	2	
fsm112	39	12	8	87	4	82	3	68	3	114	5	+
fsm113	13	4	12	25	2	23	2	34	2	31	3	
fsm114	12	4	4	52	3	44	3	68	4	62	4	
fsm115	20	2	8	22	2	20	2	36	2	22	2	
fsm116	8	4	4	16	2	18	3	38	2	18	3	
fsm117	10	4	4	42	3	39	3	59	3	46	3	
fsm118	39	4	16	265	5	232	4	289	5	266	6	
fsm119	6	8	4	19	2	15	2	20	2	18	3	+
fsm120	10	12	16	38	4	28	2	27	2	38	4	+
fsm121	12	4	12	54	2	53	2	57	2	60	3	
fsm122	11	4	4	31	3	30	3	46	3	42	3	
fsm123	71	4	16	500	7	486	7	457	6	469	7	
fsm124	34	4	12	94	3	84	3	86	3	85	5	
fsm125	6	4	4	13	2	13	2	25	2	15	2	
fsm126	25	2	2	25	2	26	2	32	2	27	3	
fsm127	10	8	2	13	2	11	2	14	2	17	4	
fsm128	39	8	12	103	3	102	4	99	3	118	5	
fsm129	10	4	4	45	3	40	3	60	3	52	3	
fsm130	18	12	16	98	3	99	3	81	3	124	5	+
fsm131	11	2	2	15	2	14	2	14	2	17	2	
fsm132	92	4	16	673	7	680	7	614	8	663	7	
fsm133	7	1	1	4	1	4	1	5	1	4	1	
fsm134	14	8	2	49	4	38	3	46	3	65	5	
fsm135	17	8	4	113	4	98	6	89	4	143	6	
fsm136	13	4	4	48	4	41	3	58	3	48	4	
fsm137	11	4	4	18	2	18	2	21	2	23	2	
fsm138	12	8	4	58	3	53	4	53	3	75	5	
fsm139	7	4	12	16	2	17	2	23	2	20	2	
fsm140	7	4	4	11	2	9	2	14	2	11	2	
fsm141	14	2	2	15	2	12	2	16	2	16	2	
fsm142	8	8	8	38	3	34	3	50	3	46	4	
fsm143	5	4	12	14	2	10	1	17	2	14	2	+
fsm144	20	2	8	34	2	35	2	40	2	38	3	
fsm145	8	4	4	13	2	12	2	22	2	15	2	
fsm146	7	1	1	4	1	1	1	4	1	4	1	
fsm147	41	12	8	135	4	128	5	98	4	137	5	+
fsm148	20	2	8	35	2	33	2	44	2	35	2	
fsm149	13	4	12	37	2	38	3	45	2	45	3	
fsm150	14	8	4	195	7	185	7	212	7	193	7	
fsm151	23	2	2	21	2	21	2	28	2	19	3	
fsm152	13	4	12	41	2	37	2	40	2	41	3	
fsm153	19	2	2	18	2	18	2	20	2	18	2	

fsm154	13	4	12	27	2	25	2	33	2	33	3	
fsm155	10	4	4	16	2	13	2	21	3	16	2	
fsm156	39	4	12	214	6	190	5	206	5	231	5	
fsm157	8	4	12	21	2	21	2	28	2	23	2	
fsm158	26	2	2	27	2	25	2	36	2	26	3	
fsm159	7	8	12	25	3	22	3	41	4	29	3	+
fsm160	13	4	12	34	2	33	2	39	2	36	2	
fsm161	13	4	4	57	4	55	3	82	3	78	4	
fsm162	14	4	12	43	2	44	2	47	2	50	3	
fsm163	13	8	4	52	4	46	3	50	2	60	5	
fsm164	14	4	4	56	4	54	4	80	4	78	4	
fsm165	25	2	2	21	2	20	2	33	2	17	2	
fsm166	13	8	4	51	4	45	3	49	3	66	4	
fsm167	10	8	16	32	3	26	2	23	2	32	3	+
fsm168	23	2	2	26	2	25	2	37	3	24	3	
fsm169	11	4	4	33	3	33	3	47	3	36	3	
fsm170	19	2	2	22	2	19	2	25	2	21	2	
fsm171	10	8	2	19	3	19	2	24	3	28	4	
fsm172	12	12	4	145	6	124	6	142	7	200	9	
fsm173	38	4	10	189	5	176	5	204	5	235	6	
fsm174	9	4	4	16	2	11	2	16	2	17	2	
fsm175	17	2	2	18	2	14	2	20	2	15	2	
fsm176	13	8	4	36	4	35	3	38	2	44	4	
fsm177	26	2	2	25	2	25	2	31	2	26	3	
fsm178	13	8	16	118	4	110	4	131	5	139	5	
fsm179	29	2	2	27	3	25	2	36	2	29	3	
fsm180	7	4	4	16	2	14	2	28	2	16	2	
fsm181	5	8	12	29	2	27	2	35	3	35	3	
fsm182	10	4	4	17	2	16	2	22	2	19	2	
fsm183	21	2	2	20	2	20	2	27	2	19	2	
fsm184	12	4	4	25	3	24	2	42	3	33	3	
fsm185	13	4	4	36	3	31	2	45	3	37	3	
fsm186	6	8	4	28	4	24	3	39	4	38	4	+
fsm187	13	8	4	36	4	33	3	43	4	49	4	
fsm188	8	4	4	15	2	11	2	17	2	14	2	
fsm189	12	4	4	46	3	47	3	74	3	56	4	
fsm190	7	8	12	26	2	23	2	27	2	21	2	+
fsm191	13	4	4	37	3	36	3	47	3	44	3	
fsm192	16	8	2	168	7	156	8	217	8	212	7	
fsm193	16	4	4	20	2	18	2	29	3	28	3	
fsm194	7	4	4	7	1	7	1	11	2	9	2	
fsm195	8	4	4	23	2	19	3	34	2	24	2	
fsm196	8	4	4	24	3	21	2	41	3	28	3	
fsm197	59	4	4	209	6	217	6	269	7	256	6	
fsm198	37	12	12	255	11	240	9	218	5	589	9	
fsm199	13	4	4	49	3	40	3	59	3	44	3	

fsm200	7	4	4	9	2	8	2	12	2	10	2	
fsm201	12	8	4	76	4	67	4	73	4	91	4	
fsm202	13	4	4	19	2	18	3	25	3	21	2	
fsm203	7	8	12	24	3	22	2	32	3	28	3	
fsm204	19	10	12	36	2	31	2	26	1	44	4	+
fsm205	12	8	4	57	4	61	4	73	4	64	4	
fsm206	14	4	12	27	2	24	2	38	2	29	2	
fsm207	36	4	10	109	3	101	3	92	3	134	5	
fsm208	11	4	4	18	2	17	2	22	2	21	2	
fsm209	29	2	2	23	3	20	2	37	2	25	3	
fsm210	7	4	4	8	2	7	2	11	2	10	2	
fsm211	7	4	4	17	2	15	2	22	2	16	2	
fsm212	45	8	16	982	9	980	9	1567	12	1895	8	
fsm213	9	8	4	70	6	59	6	92	5	92	7	
fsm214	6	8	8	19	2	16	2	25	2	28	4	+
fsm215	13	4	4	45	3	39	3	59	3	52	4	
fsm216	36	12	8	166	5	159	4	112	4	214	7	+
fsm217	10	4	4	15	2	14	2	19	2	20	3	
fsm218	36	8	12	174	5	150	5	156	5	209	5	+
fsm219	36	4	10	78	4	64	3	72	2	77	5	
fsm220	7	8	2	12	2	12	2	19	2	20	4	
fsm221	7	8	4	12	2	10	2	12	2	13	2	+
fsm222	13	4	4	18	2	16	2	21	2	21	2	
fsm223	9	4	12	26	2	23	2	30	2	25	2	
fsm224	30	2	2	26	2	25	2	36	2	28	3	
fsm225	31	2	2	27	2	24	2	39	2	25	2	
fsm226	9	8	16	40	4	33	2	34	2	45	4	+
fsm227	20	4	4	75	3	69	3	99	4	95	5	
fsm228	12	8	4	57	4	55	4	58	4	66	5	
fsm229	11	4	12	24	2	23	2	24	2	25	2	
fsm230	9	4	12	27	2	24	2	30	2	32	2	
fsm231	13	4	12	26	2	24	2	33	2	28	3	
fsm232	15	2	2	17	2	14	2	19	2	20	2	
fsm233	14	2	2	16	2	16	2	23	2	22	2	
fsm234	37	8	12	80	4	81	3	81	3	97	5	
fsm235	13	4	4	43	3	40	3	62	3	50	3	
fsm236	11	4	4	30	3	27	2	37	3	28	2	
fsm237	24	4	4	105	5	93	5	124	5	99	5	
fsm238	13	4	4	56	4	52	4	104	5	79	4	
fsm239	8	4	4	12	2	11	2	19	2	17	2	
fsm240	39	4	10	94	3	99	3	91	3	114	4	
fsm241	10	8	4	32	3	26	2	31	2	45	4	
fsm242	7	8	8	15	2	13	2	16	2	16	2	
fsm243	41	12	8	387	8	337	8	310	9	622	8	+
fsm244	20	2	2	19	2	15	2	25	2	17	2	
fsm245	36	12	12	85	3	79	3	80	3	85	4	

fsm246	17	4	4	50	3	38	3	43	3	42	3	
fsm247	37	12	12	471	9	463	9	373	9	492	10	+
fsm248	19	3	2	29	2	26	3	36	2	32	3	
fsm249	12	8	4	36	3	34	3	42	3	60	4	
fsm250	12	8	4	48	3	46	4	56	3	74	5	
fsm251	21	3	2	36	3	32	3	37	3	33	3	
fsm252	7	4	4	7	1	7	1	10	1	9	2	
fsm253	13	4	4	16	2	16	2	21	2	20	2	
fsm254	18	4	4	70	4	71	3	104	4	110	5	
fsm255	13	8	4	49	4	36	3	46	3	64	5	
fsm256	15	8	4	59	3	51	3	47	3	61	4	
fsm257	10	12	16	29	2	26	2	23	2	30	3	+
fsm258	19	2	2	17	2	18	2	24	2	18	2	
fsm259	60	4	4	246	6	236	6	274	8	231	6	
fsm260	20	2	8	24	2	22	2	37	2	23	2	
fsm261	11	4	4	34	3	31	2	42	2	41	3	
fsm262	14	4	4	56	3	50	3	76	3	56	3	
fsm263	7	8	12	24	3	24	3	32	3	31	3	+
fsm264	14	8	4	78	4	71	4	79	4	97	7	
fsm265	33	2	2	43	3	30	3	44	2	40	3	
fsm266	12	8	4	162	7	155	7	207	8	183	7	
fsm267	25	4	4	85	4	70	4	100	5	94	5	
fsm268	15	4	4	44	3	46	3	66	3	52	4	
fsm269	8	8	8	59	3	50	4	64	3	58	4	
fsm270	7	4	4	21	2	17	2	34	2	20	2	
fsm271	7	8	12	33	2	32	2	47	3	45	3	
fsm272	26	2	2	21	2	21	2	33	2	21	3	
fsm273	10	4	12	42	2	41	3	51	2	47	3	
fsm274	9	4	4	36	3	27	2	43	3	39	3	
fsm275	7	8	12	36	3	30	2	36	3	40	3	
fsm276	7	4	4	15	2	11	2	18	2	17	2	
fsm277	8	4	4	12	2	11	2	16	2	13	2	
fsm278	38	4	12	193	5	175	4	189	5	202	5	
fsm279	15	4	4	40	3	40	3	51	3	47	4	
fsm280	28	3	3	42	3	34	3	43	3	40	3	
fsm281	33	2	2	41	3	26	3	41	2	35	3	
fsm282	14	4	4	17	2	17	2	24	2	21	3	
fsm283	20	2	2	24	2	22	2	28	2	21	2	
fsm284	18	4	12	60	3	56	3	64	3	67	3	
fsm285	5	8	12	18	2	18	2	27	3	24	3	+
fsm286	6	4	12	13	2	13	2	16	2	17	2	+
fsm287	40	4	10	245	6	243	6	280	7	298	5	
fsm288	37	8	12	421	8	403	7	401	10	632	7	
fsm289	20	4	4	66	3	58	3	69	3	85	5	
fsm290	13	4	4	41	3	39	3	55	3	42	3	
fsm291	17	4	4	34	3	25	2	38	2	26	3	

fsm292	9	4	4	38	3	36	3	60	4	47	3	
fsm293	13	8	12	43	5	38	3	47	2	53	4	
fsm294	13	4	12	48	2	41	2	48	2	52	3	
fsm295	9	8	8	30	3	30	3	35	3	37	4	+
fsm296	6	8	4	20	2	19	2	21	2	22	2	+
fsm297	35	10	8	86	4	74	3	67	3	114	5	+
fsm298	14	4	4	39	4	36	5	63	4	62	4	
fsm299	39	4	12	187	4	192	4	202	4	225	4	
fsm300	13	12	4	32	4	27	3	38	3	49	4	
fsm301	20	2	8	34	2	33	2	43	2	33	2	
fsm302	6	8	8	25	2	23	2	31	2	33	4	+
fsm303	6	8	4	42	4	32	3	46	4	52	4	+
fsm304	13	4	12	26	2	26	2	33	2	32	3	
fsm305	14	4	4	40	3	40	3	60	3	51	4	
fsm306	12	4	4	20	2	17	2	22	2	19	2	
fsm307	36	8	12	250	6	245	7	233	6	383	7	
fsm308	8	4	4	18	2	16	2	28	2	19	3	
fsm309	12	8	12	48	4	45	4	55	5	52	4	
fsm310	7	1	1	4	1	3	1	4	1	4	1	
fsm311	6	8	4	18	2	17	2	25	3	22	3	+
fsm312	34	4	12	141	4	116	3	107	3	127	5	
fsm313	13	4	4	61	4	58	4	101	4	80	4	
fsm314	14	8	4	180	7	171	8	249	7	221	7	
fsm315	8	4	12	17	2	16	2	20	2	17	2	
fsm316	7	4	4	23	2	21	2	34	2	23	2	
fsm317	13	4	4	30	3	29	3	46	3	31	3	
fsm318	12	4	4	38	3	37	3	56	3	49	3	
fsm319	16	8	2	70	5	60	4	67	4	77	5	
fsm320	14	8	4	92	7	79	7	106	7	110	7	
fsm321	8	4	12	17	2	16	2	25	2	17	2	
fsm322	10	4	4	49	3	36	3	59	3	44	3	
fsm323	13	4	12	40	2	41	2	49	2	44	3	
fsm324	102	4	16	692	7	714	7	664	7	720	8	
fsm325	5	4	12	17	2	13	1	19	2	16	2	
fsm326	13	4	4	13	2	13	2	17	2	16	2	
fsm327	12	4	12	30	2	29	2	38	2	35	3	
fsm328	7	4	4	15	2	12	2	21	2	17	2	
fsm329	7	8	12	29	3	25	2	29	3	30	3	+
fsm330	8	4	4	12	2	11	2	18	2	12	2	
fsm331	38	10	8	92	4	90	3	84	3	135	6	
fsm332	13	4	4	29	3	26	2	36	2	36	4	
fsm333	38	8	10	111	4	109	3	91	3	124	5	
fsm334	13	4	12	42	2	42	2	44	2	47	3	
fsm335	17	2	2	15	2	13	2	20	2	13	2	
fsm336	33	12	8	212	5	173	5	146	5	318	9	+
fsm337	9	8	4	152	6	130	8	158	5	155	7	

fsm338	16	8	16	135	4	130	4	126	4	150	6	
fsm339	17	4	4	82	4	77	4	106	4	96	5	
fsm340	7	1	1	4	1	4	1	5	1	4	1	
fsm341	9	4	12	35	2	32	2	36	2	36	2	
fsm342	9	4	12	33	2	29	2	33	2	34	2	
fsm343	7	4	12	23	2	21	2	31	2	29	2	
fsm344	7	4	12	22	2	20	2	27	2	24	2	
fsm345	13	4	12	42	2	37	2	47	2	41	2	
fsm346	38	10	10	131	5	127	4	105	3	166	6	
fsm347	25	3	3	49	3	44	3	49	3	47	3	
fsm348	14	4	4	24	2	23	2	31	2	28	2	
fsm349	14	4	12	40	2	35	2	41	2	39	2	
Σ				21682	1051	20167	1009	23714	1044	26336	1204	
Δ%						-7,0	-4,0	9,4	-0,7	21,5	14,6	

Biography

Aleksander Ślusarczyk was born in 1974 in Warsaw, Poland.

After receiving primary and secondary education in Warsaw, he enrolled in 1993 at the Faculty of Electronics and Information Techniques at the Warsaw University of Technology. During academic year 1996/7 he studied at the University of Wales Swansea in Great Britain. In 1998, he graduated with honours from Warsaw University of Technology in the specialty Computer Construction and Programming.

In the same year, he joined the faculty of Electronics of Technische Universiteit Eindhoven in Eindhoven, the Netherlands as a Ph.D. candidate. He performed there the research on Finite State Machine decomposition and encoding for FPGA implementation under the supervision of prof. Mario Stevens and dr. Lech Jozwiak. He hopes to be able to defend the thesis on this subject on 15 December 2004.

In June 2004 Aleksander Ślusarczyk joined Topic Automatisering, where he is currently active in the area of system design of medical equipment.