The Design of a Branch Target Cache for an Asynchronous Microprocessor

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

October 1998

By

Suck-Heui Chung

Department of Computer Science

Table of Contents

1 Introduction 12

1.1 Why asynchronous design? 13
1.2 History of the AMULET processors 16
1.3 Overview of the thesis 17
1.4 Contribution of the thesis 20

2 Asynchronous design 23

2.1 Basic concepts 23
2.2 Signalling protocols 24
2.2.1 2-phase protocol 25
2.2.2 4-phase protocol 25
2.2.3 Comparison between 2-phase and 4-phase protocol 27
2.3 Asynchronous delay models 27
2.3.1 Asynchronous finite state machines 28
2.3.2 Delay-insensitive circuits 28
2.3.3 Quasi delay insensitive circuits 29
2.3.4 Speed-independent circuits 29
2.3.5 Comparison between asynchronous delay models 29
2.4 Data signalling 31
2.4.1 Bundled data 31
2.4.2 Encoded data 32
2.4.3 Sutherland’s micropipelines 33
2.4.4 Comparison between data signalling techniques 34
2.5 Control circuit synthesis 35
2.5.1 Compilation style 35
2.5.2 Asynchronous finite state machine style 35
2.5.3 Graph based style 36
2.5.4 Comparison between control circuit synthesis techniques 37

3 The AMULET3 microprocessor 39

3.1 AMULET3i 40
3.2 AMULET3 42

4 Instruction prefetch unit 48

4.1 Overview 48

4.2 Instruction prefetch unit 49

4.2.1 Configuration 49

4.2.2 Functions 57

5 Branch Prediction Mechanism 61

5.1 Basic concepts 63

5.1.1 Branch target prediction schemes 64

5.1.2 How to get the branch target address earlier 67

5.2 AMULET2e branch target cache 68

5.3 AMULET3 branch target cache 72

5.4 Summary 78

6 Implementation 80

6.1 Basic concepts 80

6.2 Front-end implementation 82

6.2.1 Data path circuit implementation 83

6.2.2 Control path circuit implementation 88

6.3 Back-end implementation 91

6.4 Evaluation 92

6.5 Summary 94

7 Conclusion 97

7.1 Contributions 97

7.2 Future work 99

Bibliography 101

A Schematics 109

B Layouts 124

List of Figures

2-1 Data-validity scheme for the 2-phase protocol 25
2-2 Data-validity schemes for the 4-phase protocol 26
2-3 Bundled data scheme 32
2-4 Encoded data scheme 32
2-5 Micropipelines 33
3-1 AMULET3i block diagram (Courtesy of Prof. Steve Furber) 40
3-2 AMULET3 block diagram (Courtesy of Prof. Steve Furber) 43
3-3 AMULET3 organization (Courtesy of Prof. Steve Furber) 47
4-1 First IPU organisation 50
4-2 Second IPU organisation 52
4-3 Simulation result comparing first and second IPU organisations 53
4-4 New parallel organisation 55
4-5 Simulation result comparing first and parallel IPU organisations 57
5-1 BTC structure 62
5-2 Address interface of AMULET2e (Courtesy of Jim Garside) 69
5-3 Write operation in the BTC 70
5-4 Look-up operation in the BTC 71
5-5 CAM comparison circuitry 73
5-6 Mode detection circuitry 75
5-7 Odd and even hit determination mechanism 76
5-8 Effect of the BTC size on prediction rates (Courtesy of Jim Garside) . 77
6-1 CAM cell 1 83
6-2 CAM cell arrays 84
6-3 CAM cell 2 85
6-4 Dummy bit cell for CAM cell 1 86
6-5 Dummy bit cell for CAM cell 2 86
6-6 RAM register cell 88
6-7 RAM cell arrays 89
6-8 CAM control circuit for write and hit detection 90
6-9 BTC layout diagram 91
6-10 Test circuit for simulation 94
6-11 CAM cell layout diagram for one word 96

List of Tables

2-1 Asynchronous Circuits Delay Models 28
4-1 Exception processing mode 59
6-1 HSPICE Simulation result 93

Abstract

A high performance, low power asynchronous branch target cache with several new features has been developed for the *AMULET3* microprocessor at a low hardware cost. A new design for the *THUMB* instruction set has been implemented, together with several circuit design techniques including dynamic comparison logic, resulting in a comparison time in 1.06ns with 0.35 µm three-level metal CMOS process technology.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

The Author

The author was awarded the degrees of Bachelor of Science in Electronic Engineering, at Yonsei University, Korea, in 1990. Significant experience was gained from involvement in the 16 bit Digital Signal Processing processor and ASIC chip design at LG Semicon and Samsung Electronics, Korea, from 1992 to 1996 and from 1991 to 1992, respectively. He became interested in asynchronous design when at Cogency Technology in UK, as a co-developing designer, in 1996. Currently, he is working on “Low Power Asynchronous VLSI Design” within the *AMULET* group at the University of Manchester.

Acknowledgements

My deepest thanks go to Professor Steve Furber for his insightful comments and invaluable guidance in the supervision of this work. I could not hope for a better supervisor.

Special thanks to Jim Garside and Jiangwei Liu for their efforts to make me feel easier in my research work. Thanks to Phil Endecott, Oleg Petlin, Steve Temple, and Siamak Mohammadi for their kind help. Thanks to William J Bainbridge and David Lloyd for their reading and commenting on the draft of this thesis. Thanks to Sun-Yen Tan for his helpful advice and interesting conversations. Thanks to everyone else of the *AMULET* group for their support and encouragement.

I would also like to acknowledge with gratitude the grant from the British Council in Korea and the European Commission.

Last but not least, I am grateful to Nigel Paver, who brought me into the asynchronous world when I stayed at Cogency Technology in UK as a co-engineer in 1996.

Dedication

To

My parents	—	Sang-Sun Chung and Young-Ja An
My sister	—	Hoon-Hee Chung
And one of my seniors	—	Dong-Woon Han

저의 사랑하는 부모님 (정 상선, 안 영자) 과
동생 (정 훈희),
그리고 존경하는 한 동운 선배님께
이 논문을 바칩니다.

Introduction 1

Following the advent of the semiconductor integrated circuit (IC) in the 1960s, many researchers have tried to find ways to improve the performance of ICs [1]. For the last 30 years, the main stream of very large scale integrated (VLSI) chip design has been the synchronous design style. It has been thought that a synchronous design style is easier to develop and more reliable in operation than an asynchronous one [2][3]. Recently, interest in asynchronous design styles has increased since the synchronous design style faces many difficulties [3][28].

This thesis presents the results of a design exercise, the objective of which is to develop a branch prediction mechanism for the AMULET3 asynchronous microprocessor. This chapter gives the background to this work. Section 1.1 describes the main advantages of using an asynchronous design style for making a VLSI chip. Section 1.2 presents the history of the AMULET processors, and an overview of this thesis and the contribution made by the author are contained in the last two sections, 1.3 and 1.4, respectively.

1.1 Why asynchronous design?

Asynchronous design has attracted renewed interest in recent years. People are talking about it as if it is a new idea. It is, however, not a new paradigm but rather a resurrection from the forgotten past [4][5]. The advantages of using an asynchronous design style [27] are as follows, where they are compared to the opposite style, synchronous design:

- *No clock skew problem.* In synchronous design, there is a global synchronous clock which is used to store and access the state of each storage element in the storage devices of a silicon chip. As the clock cycle time is reduced and the number of transistors connected to a global clock increases it is difficult for designers to keep exact synchrony due to delays in the clock distribution net. The differences in the clock delays to different nodes is called 'clock skew', and this can cause the unwanted malfunction of storage elements synchronised by the clock signal. This problem is getting more severe as transistors are shrinking with new process technology. This can be explained as follows: shrinking the design improves clock speed because parasitic capacitance reduces with the shrinkage. This clock-speed gain is hard to achieve, however, when building a bigger chip on the new process. In fact, designers can confront the problem of the same length of clock track with reduced metal width, which means that if the height of the clock metal line and the distance between this line and another metal line in an adjacent layer are reduced at the same rate as the width of the clock metal line, the capacitance (C) of the clock line will be held constant, but the resistance (R) of the clock line will increase. Therefore the RC product will make the delay of the clock line worse, and this is the main cause of the clock skew. It has been shown that this effect needs very careful design effort, for example in the DEC Alpha chip [6] and the AMD-K6 chip [7]. On the other hand, in asynchronous design, there is no clock signal. This means that an asynchronous design, as a result of its nature, has no clock skew problem.

- *Low electro-magnetic interference.* Since hand-held wireless electronic products like cellular phones became popular, electro-magnetic compatibility problems including excessive electro-magnetic emission and susceptibility to interference from electro-magnetic fields have been increasingly important issues since such products are required to meet rigorous electro-magnetic compatibility specifications. Asynchronous design may offer reduced noise emission since there is no interference generated by regular clocking and asynchronous signals spread their energy over a broader and lower frequency range and do not generate relatively high-energy levels at any one frequency. On the other hand, repetitive signals, such as clocks, are potentially troublesome, because periodic signals concentrate their energy in discrete harmonics and high-frequency clock signals are the primary source of electro-magnetic radiation from a system.

- *Low power consumption*. Complementary Metal Oxide Semiconductor (CMOS) technology has become the most effective fabrication process for the production of VLSI digital circuits basically because of low power dissipation. Even if a designer uses CMOS technology (which is used for the AMULET3 microprocessor and is the assumed technology throughout this thesis), the recent trend to portable systems creates a need for decreasing power consumption to increase the life of the battery [8][9][10]. In synchronous design, the clock keeps running even if a block is not activated. This can cause unnecessary power consumption. Asynchronous design does not suffer from this kind of problem since it has no clocking mechanism.

- *Potential for better performance.* In synchronous design, the critical path in every pipeline stage is constrained by the fixed clock period, and the clock period must be no shorter than the longest path in any stage. Even if a designer can make a circuit in a particular pipeline stage faster, this will have no benefit unless every other path is already faster than this one. This argument applies to the different pipeline stages also. Even if a designer can implement a faster pipeline stage, he must also speed up other slower stages before he can make the global clock faster. In summary, since the clock period is defined by one critical path, other faster paths must wait doing nothing. As a natural consequence, synchronous system design uses a worst-case approach. On the other hand, in asynchronous design, communications happen when individual blocks are ready, so an average-case performance can be achieved.

- *Easy to modify.* When a higher clock rate is needed or a new VLSI process technology is introduced, a synchronous designer needs to check every effect caused by the new clock rate and the new process and every circuit must obey the new timing requirements. An asynchronous designer only needs to see whether the communication protocol is observed or not. This latter procedure is a kind of functional test. Normally, a functional test is easier to verify than a timing test.

1.2 History of the AMULET processors

The AMULET (Asynchronous Microprocessor Using Low Energy Techniques) group was established late in 1990, led by Professor Steve Furber, to investigate the possibility of using asynchronous techniques for VLSI design and to see how electrical power consumption can be reduced with asynchronous design [11].

In April 1994 the group delivered the AMULET1 microprocessor [12], the world’s first implementation of a commercial microprocessor architecture (ARM) in asynchronous logic. The primary intent was to demonstrate that an asynchronous microprocessor can offer a reduction in electrical power consumption over a synchronous design in the same role. It demonstrated the feasibility of asynchronous design and opened a new era of asynchronous design in the respect that it showed comparable characteristics to its synchronous counterpart. The 2-phase micropipeline design method was used (see chapter 2 for an explanation of these terms). In spite of the success of AMULET1, it did not fully exploit the potential of the asynchronous design style to deliver improved performance and power consumption.

AMULET2e was produced in 1996 [13]. It is an embedded system chip incorporating an enhanced version of AMULET1. AMULET2e demonstrated competitive performance and power-efficiency, ease of design, and innovative features that exploit its asynchronous operation to advantage in power-sensitive applications. Since it turned out that the 4-phase signalling protocol is more efficient than the 2-phase one, 4-phase micropipelines were used in AMULET2e (again, these terms are explained in chapter 2 and an explanation of why the 4-phase protocol is more efficient than the 2-phase protocol is also presented there).

Currently AMULET3 is being developed as the first commercial embedded asynchronous 32 bit microprocessor in the world. It will be a significant milestone from the viewpoint of the commercial acceptance of asynchronous design by industry and is expected to lead to a commercially viable product as a result of its inherent low electromagnetic interference properties.

1.3 Overview of the thesis

This thesis covers a number of aspects of silicon design using asynchronous techniques. The main topic is the implementation of the branch target cache for the instruction prefetch unit in AMULET3. This thesis shows how to implement an asynchronous system from the standpoint of transistor level design and gives practical examples rather than a theoretical approach with the particular subject matter of the branch target cache of the instruction prefetch unit. However, detailed descriptions of the block implementations are not included in the main text. Instead several major schematics and layouts are attached as appendices at the end of the thesis.

It should be noted here that the design of the AMULET3 microprocessor is a major cooperative project involving many people. The author is responsible for designing the instruction prefetch unit in the AMULET3 microprocessor, and the branch target cache is a part of the instruction prefetch unit.

In chapter 2, basic and fundamental background knowledge is introduced. A brief description of the AMULET3 microprocessor follows in chapter 3 to give the reader an overview of the context of the branch target cache. To help the reader to understand the functions of the branch target cache in the instruction prefetch unit, chapter 4 explains what the instruction prefetch unit is, what kinds of sub-blocks exist, and describes the functions and configuration of the instruction prefetch unit. The remaining chapters are devoted to the design of the branch target cache for the instruction prefetch unit. An improved branch prediction mechanism is described in chapter 5, where it is compared to the prediction mechanism used in AMULET2e. The implementation is presented in chapter 6, starting from a schematic level and referring to the low level. Front-end design is carried out using static and dynamic circuit techniques, and then the back-end implementations of circuits for the data and control paths follow. Finally, in chapter 7 the work is evaluated and future work is proposed.

Before going any further, it is worthwhile mentioning particular features of the instruction prefetch unit in AMULET3 compared to those of AMULET2e. They are as follows:

- *Harvard architecture.* In AMULET2e the instruction prefetch unit was coupled with the data address interface, allowing the processor to be connected to a single memory for both instructions and data. This led to a complex architecture which was not efficient. For AMULET3, a Harvard architecture is introduced and the instruction address and data address interfaces are separated. Each is connected to its own memory port through its own bus. The first AMULET3 system incorporates a dual ported memory, so the instruction and data address interfaces have independent access to a unified instruction and data memory. This will boost the total performance of AMULET3 to support faster operation of the addressing blocks and to give the chip more concurrent behaviour.

- *Non-sequential instruction address stream handling.* An interrupt can be treated as a branch instruction from the point of view that it causes the processor to deviate from sequential instruction execution. Because of this, the AMULET3 interrupt handling block is included in the instruction prefetch unit. Normal instruction address sequences can be changed by three factors: system reset, an interrupt, or a branch. All the logic handling this issue is implemented in the instruction prefetch unit. Using an asynchronous arbiter, the asynchronous nature of an interrupt which could lead to a synchronisation problem in synchronous design can be implemented easily, and several innovative control mechanisms are used.

- *Improved branch prediction mechanism.* A branch prediction mechanism was introduced in AMULET2e to improve performance since a non-sequential instruction fetch takes some time to settle down into a sequential stream [14]. In AMULET2e, the branch target cache stores 20 predicted branches and this is expanded to 32 entries in AMULET3. The greater the number of entries, the more possibility there is to improve performance. Since the THUMB instruction set [15] is introduced in AMULET3, in which the length of an instruction is 16 bits, new special circuitry is required to support it.

- *Halt function.* Another advantage of using an asynchronous design style is the 'halt' function. In a synchronous design, when halting a chip, a designer should consider whether the clock is stopped or not and if it is stopped how it can be brought back when necessary. By contrast, in an asynchronous design, a chip can be stopped abruptly and revived instantly without any redundant circuitry. Except for minimal power consumption due to the inherent leakage currents in a CMOS transistor cell, the power consumption can be almost zero when 'halt' is asserted. This halt function is also implemented in the instruction prefetch unit.

Detailed explanations are presented in chapters 4, 5, and 6.

1.4 Contribution of the thesis

The title of this thesis implies that the thesis describes the design work carried out on the branch target cache in the AMULET3 microprocessor. Since the instruction prefetch unit of AMULET3 was designed by the author and the branch target cache is a part of the instruction prefetch unit, the design of the instruction prefetch unit is also described here (in chapter 4) to help the reader understand the environment surrounding the branch target cache. The high level specification of and interface to the instruction prefetch unit were defined by the AMULET3 design team, led by Dr. Jim Garside. The author was responsible for translating the specification into a detailed VLSI implementation. The work involved:

- understanding the high-level specification as defined by the AMULET3 ‘LARD’ model, which is written in an asynchronous hardware description language developed by Dr. Philip Endecott [99].
- using the LARD model to explore two alternative organisations for the instruction prefetch unit.
- developing low-level schematic and layout details to yield detailed performance estimates.
- on the basis of these numbers, rejecting both proposed organisations.
- devising and proposing a third organisation with increased concurrency, and proving its functionality in the LARD model.
- developing a detailed implementation of this third organisation to show that it will meet the performance targets.
- completing and documenting the final design of the instruction prefetch unit, including design-for-test features.

The last two steps in the work will only be fully completed when the AMULET3 design is finalised for fabrication.

The main contributions of the thesis, describing work carried out by the author are, as follows:

- In chapter 4, a high performance, low power asynchronous instruction prefetch unit is introduced. Unlike the case of AMULET2e, the instruction prefetch unit is detached from the data address interface unit - AMULET3 uses a Harvard architecture. All the specifications and descriptions and the new architecture of the instruction prefetch unit are presented.

- In chapter 5, an improved (compared to that of AMULET2e) branch prediction mechanism is described. It was designed for higher performance and supports more functions. The number of entries stored in the predictor is increased from 20 to 32, and new function blocks supporting the THUMB instruction set are included.

- In chapter 6, various techniques which could be used to implement the branch target cache of the instruction prefetch unit and the instruction prefetch unit itself are shown. Custom cell design techniques for the datapath and the control path design are proposed. This chapter can be read as a guide book to asynchronous system design, not just the branch target cache of the instruction prefetch unit, since it includes both data and control path design and together these are the components required to build any VLSI chip.

Asynchronous design 2

In order to define a system and its environment, different approaches and diverse models are used in accordance with different situations. So it is for asynchronous design. Some approaches emphasize the communications between blocks and others emphasize the behaviours of the blocks themselves. Some describe the communication as a sequential handshake and others see it as multiple changes of inputs and outputs. Different approaches to asynchronous design offer different prospective and employ different rules. Subsequent sections explain the models used in asynchronous design according to various different ways to interpret a system and the environment surrounding it.

2.1 Basic concepts

Synchronous design forces every circuit to follow one rule - obey the clock. This centralized system appears at first sight to be an easier method for implementing a silicon chip than asynchronous design. Because the designer's attention is confined to the periodic clocking, his concern is solely whether he can meet a timing constraint with his implemented circuit. When he succeeds in observing this rule, his circuit is safe and will work well.

Asynchronous design, however, has no clock guiding the way which you should follow. In some sense it sounds anarchic, but through well disciplined methods one component can detect whether other components are ready. These methods are called protocols, with which blocks and cells in a silicon chip can communicate with each other. This is a basic concept in asynchronous design. To make asynchronous design easier, many researchers have investigated and invented many types of models to describe asynchronous behaviours. The rest of this chapter is dedicated to show these asynchronous models.

2.2 Signalling protocols

Communication requires that something happens between two participants. The ‘handshake protocol’ can be explained as follows: there are two sides, a sender and a receiver. The sender transfers information to the receiver with a ‘request’ signal and the receiver accepts it. After the receiver has accepted the information, it sends an ‘acknowledge’ signal back to the sender. Then the sender is allowed to send further information to the receiver. However, the initiator that starts this communication can be either side depending on the specification. Depending on this, the protocol can be categorised as either a push transfer or a pull transfer [16]. In a push transfer, the initiator sends the data as in the case described above. In a pull transfer, the initiator requests the data. In this case the receiver sends a request signal to the sender and the sender can send data to the receiver. Even if a pull protocol is available, communicating data with handshakes in a pull protocol is not very common. In this thesis, only the push protocol is considered.

2.2.1 2-phase protocol

The 2-phase protocol uses transition signalling. Since there are only two transitions available in the digital domain, 0 to 1 and 1 to 0, a data transfer happens at each transition edge. When data are ready to be transferred, the sender sends a request transition to the receiver (in the case of the push protocol). The receiver receives data and returns an acknowledge transition to the sender. After the sender receives the acknowledge, it will be able to send more data. Figure 2-1 shows a diagram of the data-validity scheme for the 2-phase protocol.

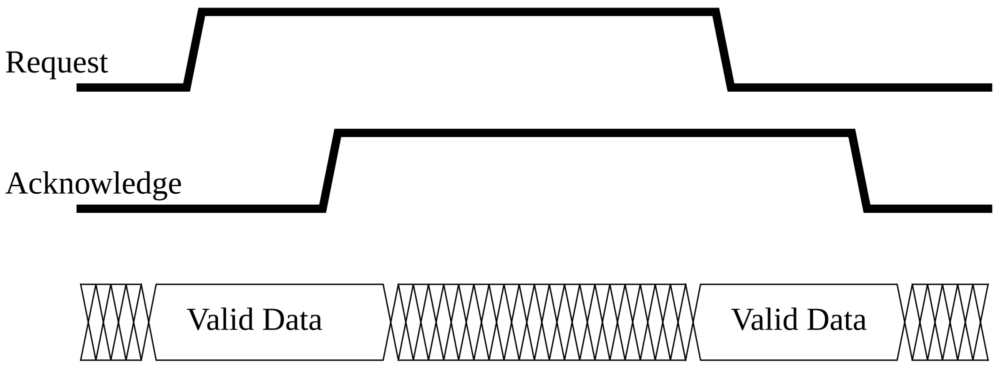


Figure 2-1: Data-validity scheme for the 2-phase protocol

2.2.2 4-phase protocol

The 4-phase protocol is called level signalling, since its actions follow a signal level rather than a transition. A redundant ‘return to zero’ signal change is required. Unlike the 2-phase protocol, the 4-phase protocol has three data-validity schemes - early, broad, and late. They are shown in figure 2-2.

Consider the early data-validity scheme. When data is ready for transferring from the sender to the receiver, the sender sends a request signal to the receiver and the receiver responds with an acknowledge signal (in the case of the push protocol). However, the transfer is not yet finished. The sender must still return the request signal back to the inactive level and then the receiver must also follow the same procedure with the acknowledge signal. The return to zero phase is a redundant function which does nothing but return the signals to the original state. This seems like a waste of time. But designers should take into account the fact that most of the data storage elements available in reality, such as latches and flip/flops, naturally operate with the 4-phase protocol [17].

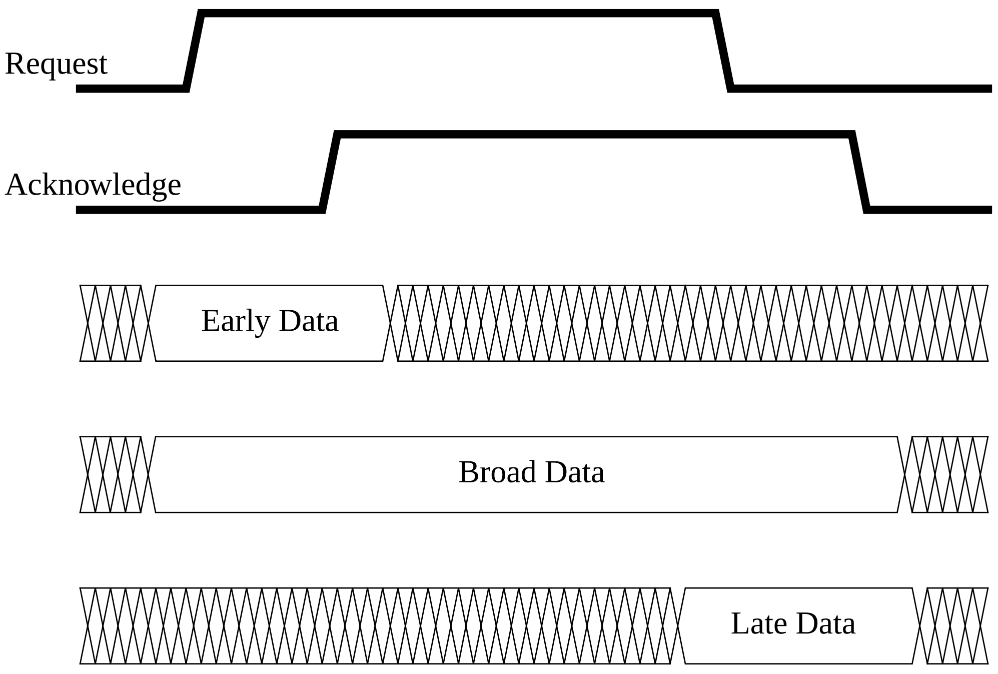


Figure 2-2: Data-validity schemes for the 4-phase protocol

2.2.3 Comparison between 2-phase and 4-phase protocol

The 2-phase protocol has the advantage compared to the 4-phase protocol in that it does not have any redundant signal transitions. It could make a faster control circuit to use the 2-phase protocol rather than the 4-phase protocol, since it could save time taken for resetting signals in the 4-phase protocol. In order to use the 2-phase protocol, however, designers must use a double edge-triggered flip/flop for data storage which requires approximately twice the area compared to a level-sensitive latch and consumes up to four times as much power [18]. Alternatively, if designers want to use the 2-phase protocol with a single edge-triggered flip/flop or a latch as in conventional designs, they must convert the protocol from 2-phase to 4-phase [19], which requires rather complex control circuitry. 2-phase control was used in the AMULET1 design, but for AMULET2e and AMULET3 4-phase control was chosen. This was because 4-phase control is easier to use with dynamic logic as shown in [94], and 2-phase control circuits are slow in practice since they make the exclusive use of XOR gates which, in CMOS, are expensive in terms of speed and area. For these reasons the 4-phase protocol seems to be used more in practice.

2.3 Asynchronous delay models

Asynchronous design requires some assumptions to be made about wire and/or gate delays. These delay assumptions are summarized in table 2-1 (Terms are explained in subsequent sections)

.

Table 2-1: Asynchronous Circuits Delay Models

Models	Gate assumption	Wire assumption
Asynchronous FSM	bounded	bounded
Delay Insensitive Circuits	unbounded	unbounded
Quasi Delay Insensitive Circuits	unbounded	unbounded + some isochronic fork
Speed Independent Circuits	unbounded	all isochronic fork

2.3.1 Asynchronous finite state machines

Asynchronous finite state machines (AFSMs) [4][5] are comprised of combinational logic and feedback delay paths from outputs to inputs. They appear to be similar to synchronous finite state machines (SFSMs) [34] except for the fact that the clocks in SFSMs are replaced by feedback delay elements. Just as SFSMs should observe the clocking period, AFSMs also should observe the limit of the delay element. The inputs of AFSMs cannot change before the feedback delay signals are stable. This is known as the limit of fundamental mode which assumes that only one input can change at once, and the next input change can enter the circuit only after the entire circuit has reached a stable state.

2.3.2 Delay-insensitive circuits

The delay-insensitive circuit [20][21][22] operates correctly regardless of gate and wire delay variations. This assumes that gates and wires have arbitrary finite delays. This is such an attractive approach in that all the data and control can be defined by signal transitions and then true asynchronous design can be achieved. For data transfer, dual rail encoding can be used; one line is used to transfer a 0 and another is used to transfer a 1. Consequently, a delay-insensitive circuit will work with any amount of delay. Delay-insensitive circuits must be designed so that delay variations on the wires do not cause a malfunction of the circuit.

2.3.3 Quasi delay insensitive circuits

A circuit is said to be quasi delay insensitive [25] if its correct operation is independent of the delays of gates and wires, except for certain wires that form isochronic forks. The term ‘fork’ means that there are two or more wire paths available from the output of a component to the inputs of other components. The term ‘isochronic fork’ means that the delays in wires from the same output to separate inputs are equal.

2.3.4 Speed-independent circuits

In a speed independent circuit [29][30][31], it is assumed that wires have zero delay and the global behaviour of the circuit is independent of the delays of all of the gates. That is, ordered input events produce ordered output events and all the forks are isochronic.

2.3.5 Comparison between asynchronous delay models

The limit of fundamental mode in AFSMs is a very weak point in terms of performance, and only by making many back-annotation simulations with timing factors extracted from the layout can the exact behaviour after fabrication be guaranteed. When AFSMs are used in datapath pipelines, the timing constraints even affect the next AFSMs connected in series. The first AFSM must not accept new input changes before its own timing constraints and the next one’s have been satisfied. This results in extremely poor throughput in designs with many pipelines.

The set of components connected with wires which can support the delay-insensitive delay model is very limited, since forks are allowed but it cannot be assumed that they are isochronic, and most VLSI components can fail due to inputs having a very slow edge-speed [23][24].

The quasi delay insensitive circuit has a strong possibility of failure if designers use an auto place and route layout approach. To meet the criteria of quasi delay insensitive operation, careful circuit design is needed [24]. Designers must avoid slow edges on control wires. If not, two different gates on an isochronic fork may see the same transition at very different times [26].

The speed independent assumption is viable when the wire within a chip has negligible delays compared to gate delays. Therefore all the wire routing must be localized so that the wire delay is small compared to the gate delay, and the skew between wire delays after a fork must be less than the gate delay. Even if these requirements are fulfilled, the behaviour of isochronic forks can break down where a wire is connected to a gate which has an early logic threshold voltage [24]. The output of this gate is triggered before the input reaches a discrete logic level, and then the output of the gate triggers the next gate when the input still has not reached a discrete logic level. This can break the isochronic fork assumption. Therefore it is important to keep the logic threshold voltage of gates as uniform as possible [24][26].

Nevertheless, the speed independent circuit model is powerful since multiple input changes can be allowed without timing constraints, thereby invoking more concurrent behaviour. This is why the AMULET3 design adopted the speed independent model for the control path. It is important for the speed independent circuit to make correct correlational specifications between ordered inputs and outputs, since outputs are followed by inputs and vice versa. Because designers cannot change the environment, they should know the behaviour of the environment after the outputs of the circuit are generated [32].

2.4 Data signalling

Most VLSI chips contain a datapath on which the data is transferred when a computation is running. A typical synchronous datapath is formed by pipelines that have registers in their input and output sides to store the data between combinational circuits. These registers are controlled by clocks.

In asynchronous design, two kinds of data transfer method are available: the bundled data method and the data-encoding method.

2.4.1 Bundled data

The bundled data method [16][33][34], as shown in figure 2-3, has a request signal, an acknowledge signal, and data lines. A block of combinational logic sends a request signal to the next block when data is available, and the next block sends an acknowledge signal to the previous block in return to indicate that the data has been received and it is available for the next data transfer.

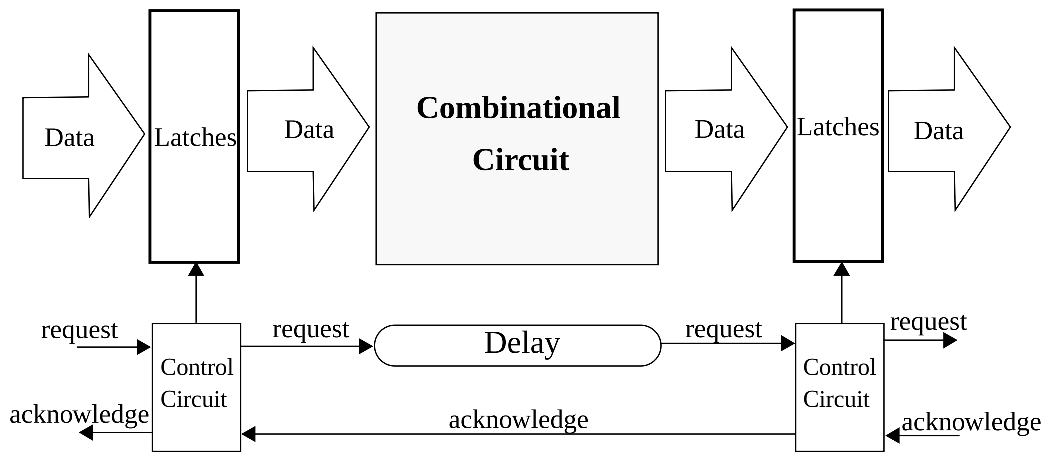


Figure 2-3: Bundled data scheme

2.4.2 Encoded data

The encoded-data method [35] generates the completion detection signal when the data transfer is finished depending on the latched data pattern as shown in figure 2-4. One of

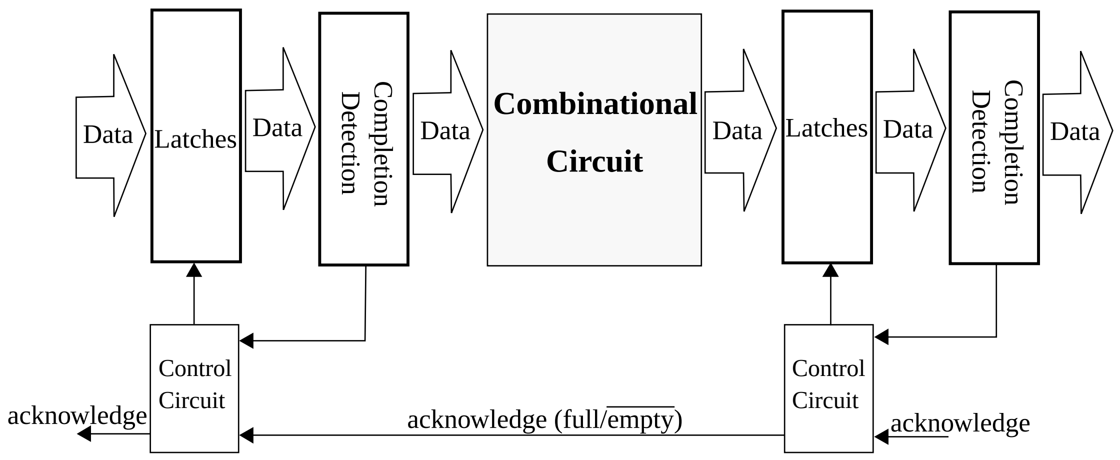


Figure 2-4: Encoded data scheme

the possible encoded data methods is the dual-rail style, which has 2 wires for every data bit. For example, suppose 01 is used to transfer a 0 bit and 10 is used to transfer to 1 bit; every data bit will be encoded as 01 or 10 after the computation is done. Therefore one of the 2 wires becomes 1. When one wire of every data bit changes to 1, the sender sends the completion detection signal to the receiver and the receiver returns the acknowledge signal back to the sender and then the sender will reset every data bit.

2.4.3 Sutherland's micropipelines

Micropipelines were introduced by Ivan Sutherland [36]. A micropipeline is similar to a synchronous pipeline without the clocking mechanism. There are registers between combinational logic blocks, which are controlled by circuits which use the request and acknowledge signal. Each stage of a micropipeline has a request signal to inform the next stage when the data is ready and the next stage returns an acknowledge signal when the data is received. In conclusion, micropipelines use the bundled data method for transferring data and an event-driven 2-phase signal protocol for the control circuit. This is shown in figure 2-5.

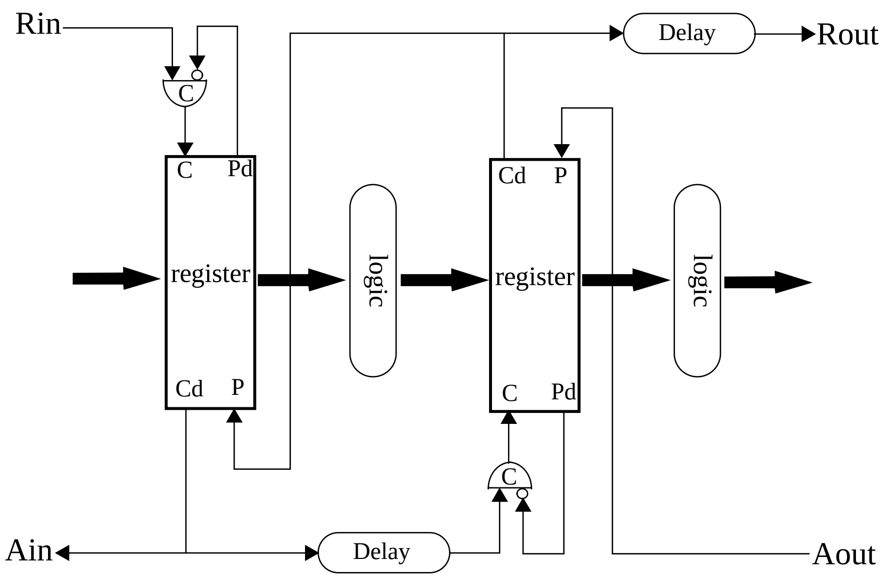


Figure 2-5: Micropipelines

2.4.4 Comparison between data signalling techniques

An advantage of the bundled data method is that normal standard datapath components can be used. This is why all AMULET processors adopted the bundled data method. A disadvantage of this method is that a matched dummy single-bit datapath is normally used to generate a completion signal, to use as a request signal to the next block, and this delay must be at least equal to the worst case datapath delay. This means that designers may have to sacrifice one of the advantages of asynchronous design - the ability to achieve average case performance.

The encoded-data method can be efficient in terms of speed compared to the bundled data method - it achieves average case performance - but the completion detection overhead due to using 2 wires for every data bit cannot be neglected in terms of power and silicon area. Even when designers implement a processing pipeline, no benefit will be obtained by completing processing early if the subsequent pipeline stage is not free to accept the data. This may mean that the average performance obtained does not justify the overhead of the dual-rail logic and completion detection circuits.

Sutherland’s micropipelines share the same problem manifested in other circuits which use the bundled data method as mentioned in section 2.4.1 - delay matching. It must be guaranteed that the data arrives at the receiver before the request signal from the sender. So carefully designed delay elements may be required in the sender’s request line. Furthermore in order to cope with the 2-phase protocol, a specially designed register must be used, which Sutherland also proposed in [36].

2.5 Control circuit synthesis

Some designers can make control circuits intuitively but not everybody can do it well. Testing these circuits is also empirical. The intuitive method can result in mistakes which designers do not recognize when they first make the circuits. There have been several design methods proposed for control circuits which formalize the design flow and assure the result if the specifications are correct.

2.5.1 Compilation style

This method makes circuits by compiling high-level languages which express concurrency [37][38][39][40][41][42]. The result of synthesis is usually a delay-insensitive or a speed-independent circuit. Fundamentally, this method maps language descriptions to hardware components.

2.5.2 Asynchronous finite state machine style

The asynchronous finite state machine (AFSM) [4][5] was the first asynchronous design methodology. It assumed that a single input change invokes the system and the next input cannot enter before the system is stable. This means that inputs that change serially should wait for some time to guarantee the system and the output is settled. This is called the fundamental delay mode. This may be a weak point to make a system in terms of concurrent behaviour.

To overcome this disadvantage, a new AFSM was proposed, named the burst-mode machine [43][44][45], which allows multiple input changes. When the specified set of input edges appear, the system generates a set of output changes and then new multiple input changes can be accepted. The specified input changes can happen in any time and a set of output changes can happen concurrently. Since burst-mode AFSMs use the same finite machine style as used in synchronous design, they appear familiar to designers. However, they suffer from the problem that input changes are not allowed to be concurrent with output changes.

Recently, the extended burst-mode machine [46][47][48] was introduced to reduce the problem of the burst-mode machine and to add more flexible input choice. Directed-don't-cares and conditionals were devised. Directed-don't-cares allow an input signal to change concurrently with output signals and conditionals allow control flow to depend on the input signal levels.

2.5.3 Graph based style

The graph based style means using a Petri net [49] or a similar graphical representation [50] of concurrency to specify the required functionality. The Petri net is a model describing a concurrent system. The signal transition graph (STG) [51][52][53][54][55][56][57][58][59][60] was introduced as an interpreted Petri net. It interprets value changes on input and output signals of the specified circuit as transitions of the STG. Positive transitions (labelled with a '+') represent a $0 \rightarrow 1$ change and negative transitions (labelled with a '-') represent a $1 \rightarrow 0$ change. This way, designers can specify changes of all the inputs and related outputs. Generally, the strong point of this method lies in its ability to describe concurrency.

Recently, a very powerful tool named Petrify [61][62] was introduced. It has basic functions which allow the manipulation of concurrent specifications. This tool surmounts the problem, which was thought to be a disadvantage of this method, of specifying input choices. Given an initial STG or Petri net, the tool checks the property of Complete State Coding [63]; whether different states of the system are encoded with the same binary code. If there is a violation of this property, the tool automatically inserts a new internal state variable. The tool can make a speed independent circuit which has no timing constraints, unlike the extend burst-mode machine.

2.5.4 Comparison between control circuit synthesis techniques

An advantage of the compilation style is that designers can write a concise and well ordered program and get a silicon result in much shorter time than using a traditional hand made design methodology. However, a drawback is that it is difficult to get a very optimized circuit, since a mapping function is used to do the translation and engineers cannot optimize further to the level below the basic library components. A method for simplifying these synthesized circuits by repeated provable refinement has been demonstrated which allows some of this complexity to be reduced [98].

When the extended burst mode machine is used to design a system, because it is based on the fundamental delay mode, it must be guaranteed that input changes cannot occur before the system has stabilized, and delay elements must be inserted in the feed back paths. However, this method could be attractive to designers since this is the same method as used in synchronous design except that in the synchronous machine the clock is used to control feedback paths using memory elements.

As was mentioned earlier, the graph based style has the advantage in that it can be used to describe highly concurrent systems without timing constraints. This is why this method has been adopted for AMULET3. In many cases, however, this method produces very complex and slow circuitry to implement the full concurrency. It is necessary for designers to be aware of the critical path in their STG definition and to reduce concurrency to lessen the complexity of the circuitry within the limits of the system specification requirement. This process needs very careful intuition and experience.

The AMULET3 microprocessor 3

Though several asynchronous microprocessors have been developed [64][65][66][67][68][69][70][71][72][12][13], most were designed for the purpose of demonstrating the feasibility of asynchronous design. AMULET1 and AMULET2e are in this category. Unlike AMULET1 and AMULET2e, AMULET3 is being developed for a commercial application as an embedded 32bit RISC microprocessor in a communications chip. Currently AMULET3i (the AMULET3 asynchronous island) is under development [73]. AMULET3i is an asynchronous embedded subsystem chip incorporating AMULET3 as a microprocessor.

AMULET1 showed the feasibility of implementing an asynchronous design with a highly concurrent behaviour. AMULET2e proved that asynchronous design could achieve competitive performance on an equal footing with synchronous design. AMULET3 is intended to be the first commercial application of the AMULET asynchronous technology. Through AMULET3’s use in the commercial domain, asynchronous design can win recognition as having a role in mainstream VLSI chip design. The rest of this chapter will describe the structures and functions of AMULET3 and AMULET3i in order to give an overview of the context for the work described in the rest of the thesis, which covers the design of the instruction prefetch unit for the AMULET3 microprocessor.

3.1 AMULET3i

AMULET3i (the AMULET3 asynchronous island) is an integrated asynchronous microprocessor subsystem based around AMULET3. Its block diagram is shown in Figure 3-1.

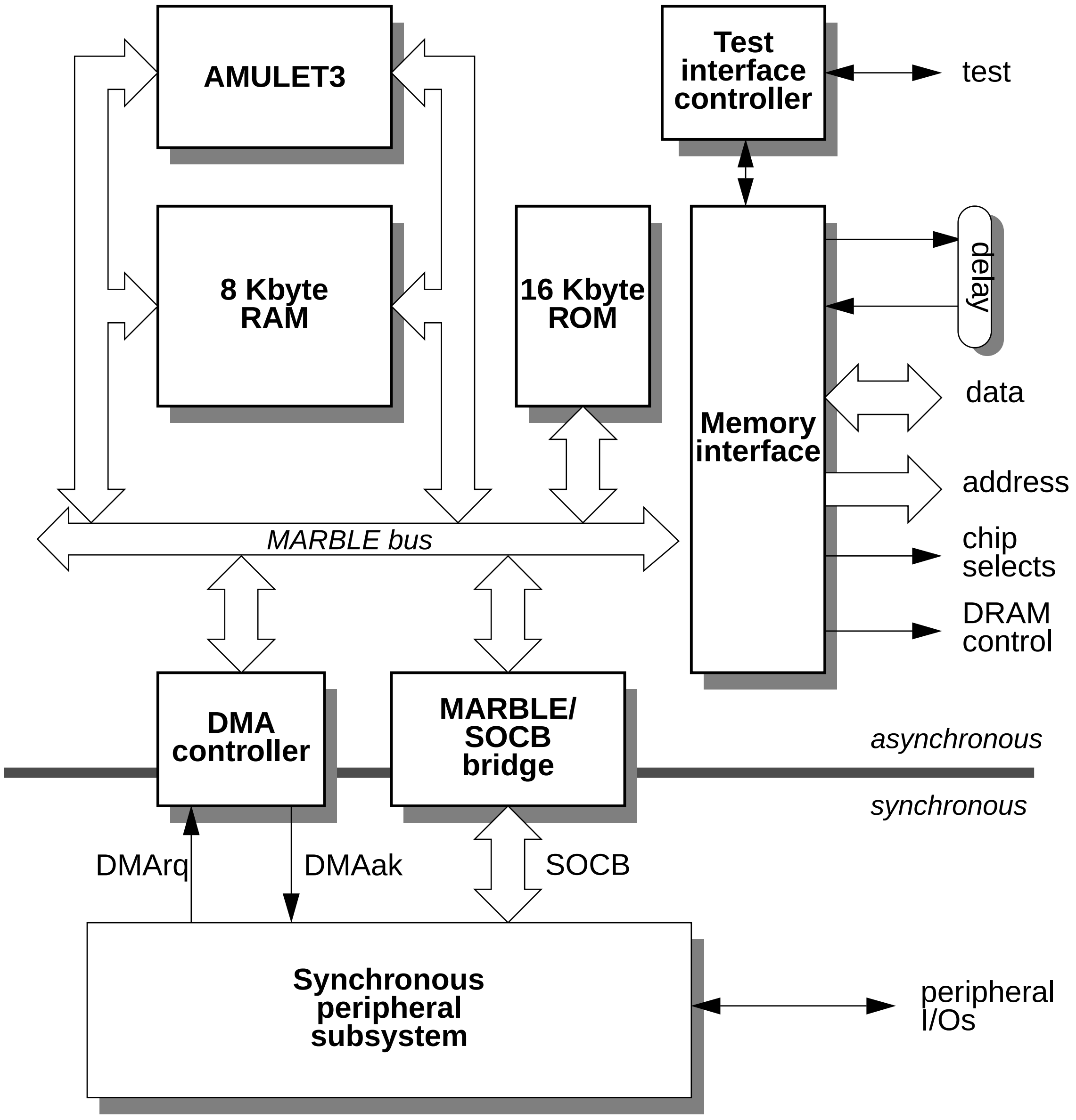


Figure 3-1: AMULET3i block diagram (Courtesy of Prof. Steve Furber)

In addition to the AMULET3 processor, AMULET3i comprises:

- *8 Kbyte RAM*. The 8 Kbyte internal static memory is divided into eight 1Kbyte blocks. Each block contains 64 lines of 4 words.
- *DMA controller*. The DMA controller has 32 independently programmable channels each of which can perform memory to memory, memory to peripheral, peripheral to memory or peripheral to peripheral transfers.
- *MARBLE bus*. The Manchester AsynchRonous Bus for Low Energy is a multi-master on-chip bus for connecting macrocells.
- *MARBLE/SOCB bridge*. The MARBLE to Synchronous On-Chip Bus bridge is a single MARBLE target device which handles the bus handshake and control signal retiming on behalf of the SOCB.
- *16 Kbyte ROM*. The 16 Kbyte ROM contains application code and also a number of routines to support the testing of AMULET3i components.
- *Test interface controller*. The test interface controller supports the direct access to individual on-chip macrocells via the external memory interface and MARBLE.
- *Memory interface*. The AMULET3 external memory interface supports the direct connection of external memory and peripheral devices.
- *Synchronous peripheral subsystem*. This contains telecommunication peripheral devices.

Static memory devices, such as SRAM, EPROM and peripheral chips, can be connected directly to the processor with no extra logic. In addition, DRAM is supported, again with no external support logic.

3.2 AMULET3

AMULET3 is the third generation asynchronous ARM microprocessor core and supports the ARM 32bit RISC architecture. It implements the ARM architecture version 4T [74] and supports Thumb instruction set compatibility [15][74]. The processor core can be divided into 5 sub blocks, as shown below in figure 3-2. The 'Prefetch' block is the instruction prefetch unit which is being developed by the author. This is explained further in chapter 4. The 'Decode & Register read' block is to decode instructions, to read the register values, and to produce control signals to relevant blocks. The 'Execute' block is comprised of the ALU and the multiplier to execute arithmetic/logical manipulations. The 'Data access' block is to access data memory. The 'Reorder & Writeback' block is to implement a reorder buffer and result forwarding mechanism. The detailed AMULET3 organization is shown in figure 3-3.

The processor core contains a number of novel features to enhance performance. This chapter explains in brief only the distinguishing features of AMULET3 compared with the previous AMULET1 and AMULET2e designs. They are:

- *A Dual Bus Interface (Harvard Architecture).* As shown in Figure 3-2, the instruction prefetch unit and the data interface unit are separated, unlike those of AMULET2e. The data interface is sidelined from the main instruction flow allowing the decoupling of data transfer operations (especially multiple register moves) from purely internal operations. Interestingly, although separate instruction and data buses are used, a unified memory (which is an internal Random Access Memory (RAM) in the case of AMULET3i) can still be used. This is possible because a new memory architecture has been developed using block-level arbitration between instruction and data requests. The memory area is not divided into instruction and data areas. When instruction and data requests access the same block in the memory, the request which arrives earlier will be granted access to the memory via an asynchronous arbiter. After this access finishes, the later request which was held at the input to the arbiter will be granted access to the same block. In short, this memory behaves like a dual port RAM but uses a number of transistor not much greater than in a single port RAM. Compared to a true dual port RAM, the new RAM can save a lot of silicon area.

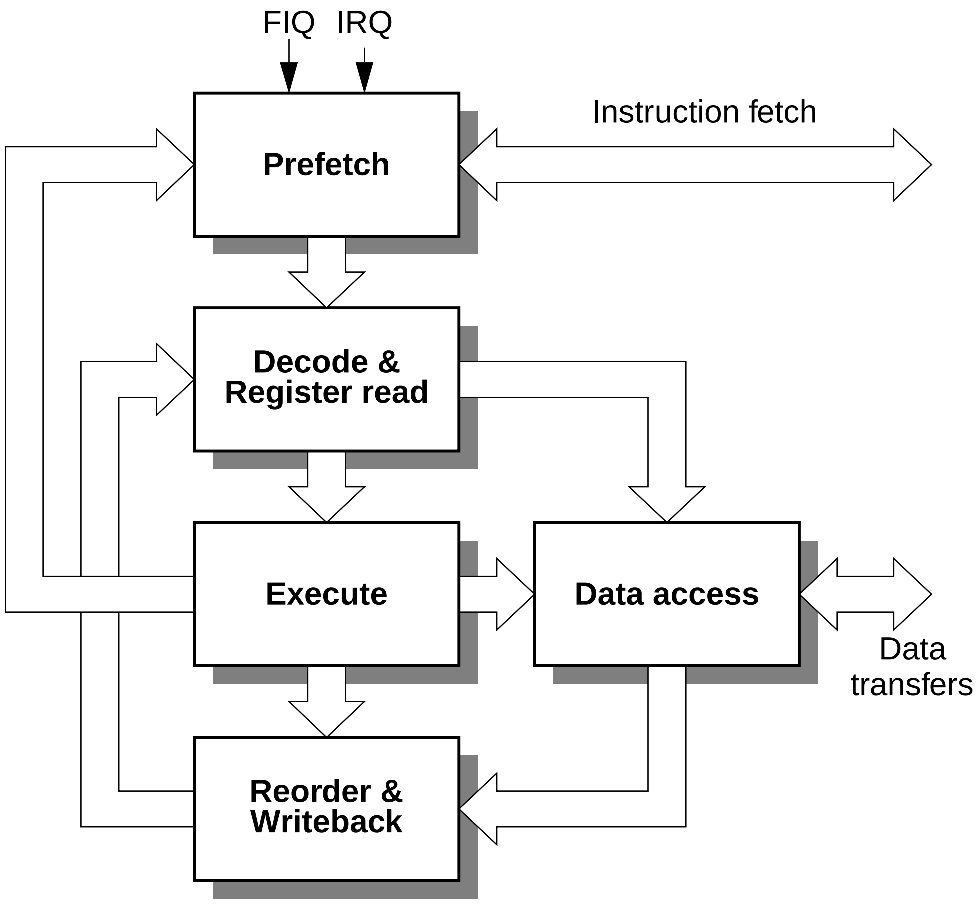


Figure 3-2: AMULET3 block diagram (Courtesy of Prof. Steve Furber)

- *Reorder Buffer and Result Forwarding Mechanism.* Instructions can complete out of order and execution results are stored in the reorder buffer to be written back to the general registers [76]. The reorder buffer supports a result forwarding mechanism, so if an operation undergoing processing requires a recent result which may not have been returned to the general register bank, it can be accessed from the reorder buffer. If the data fetch operation is aborted, results in the reorder buffer generated by subsequent instructions are discarded whereas results from previous instructions are written to the register bank. In short, the reorder buffer gives each instruction a slot in the buffer and the result produced by the execution of each instruction goes to its slot, possibly out of order. Writing out to the general register bank is deterministic and sequential. This mechanism provides a general solution to the forwarding problem while still allowing precise aborts.

- *Branch Prediction.* Branch prediction can be performed on a proportion of previously encountered branches. This increases performance and lowers *system* power consumption since it reduces the number of erroneous prefetch cycles. Until a branch instruction is decoded and recognized as a branch, subsequent instructions following the branch are prefetched from the instruction memory and sent to the decoder through the instruction pipeline. This will burn unnecessary power when the branch is taken. It has been estimated in a cached ARM that the processor core is typically responsible for only 30% of the total power dissipation. 70% of the power consumption happens in the cache/memory system. This means that even if the branch target cache uses power to check for a predicted branch every cycle, and thereby increases the processor's power consumption for a given instruction frequency, a predicted branch can save power overall by reducing the number of unwanted memory accesses. This is explained further in chapter 5.

- *Fast interrupt response.* Interrupts are not dealt with in the instruction decoder but in the instruction prefetch unit. This was made possible by separating the instruction and data interface. An interrupt is treated as a kind of unconditional predicted branch and the interrupt service code can be fetched as soon as the interrupt occurs.

- *Halt mechanism*. A halt mechanism can be implemented easily by intercepting and disabling request or acknowledge signals at some critical point in the processor. Furthermore, recovery from the halted state can be achieved instantly by releasing the intercepted signals, while a synchronous design would wait for stabilization of the clock, which could take some time and require careful consideration of dealing with clock generation blocks. In the instruction prefetch unit, this mechanism is implemented by disabling a request signal in the unit until an interrupt occurs.

As highlighted above, three of the five major features are related to the instruction prefetch unit. This implies the design of the instruction prefetch unit will be a crucial factor in determining the AMULET3 performance. As was mentioned in chapter 1, the interrupt and halt mechanism is explained in chapter 4 and the branch prediction mechanism is dealt with in detail in chapter 5.

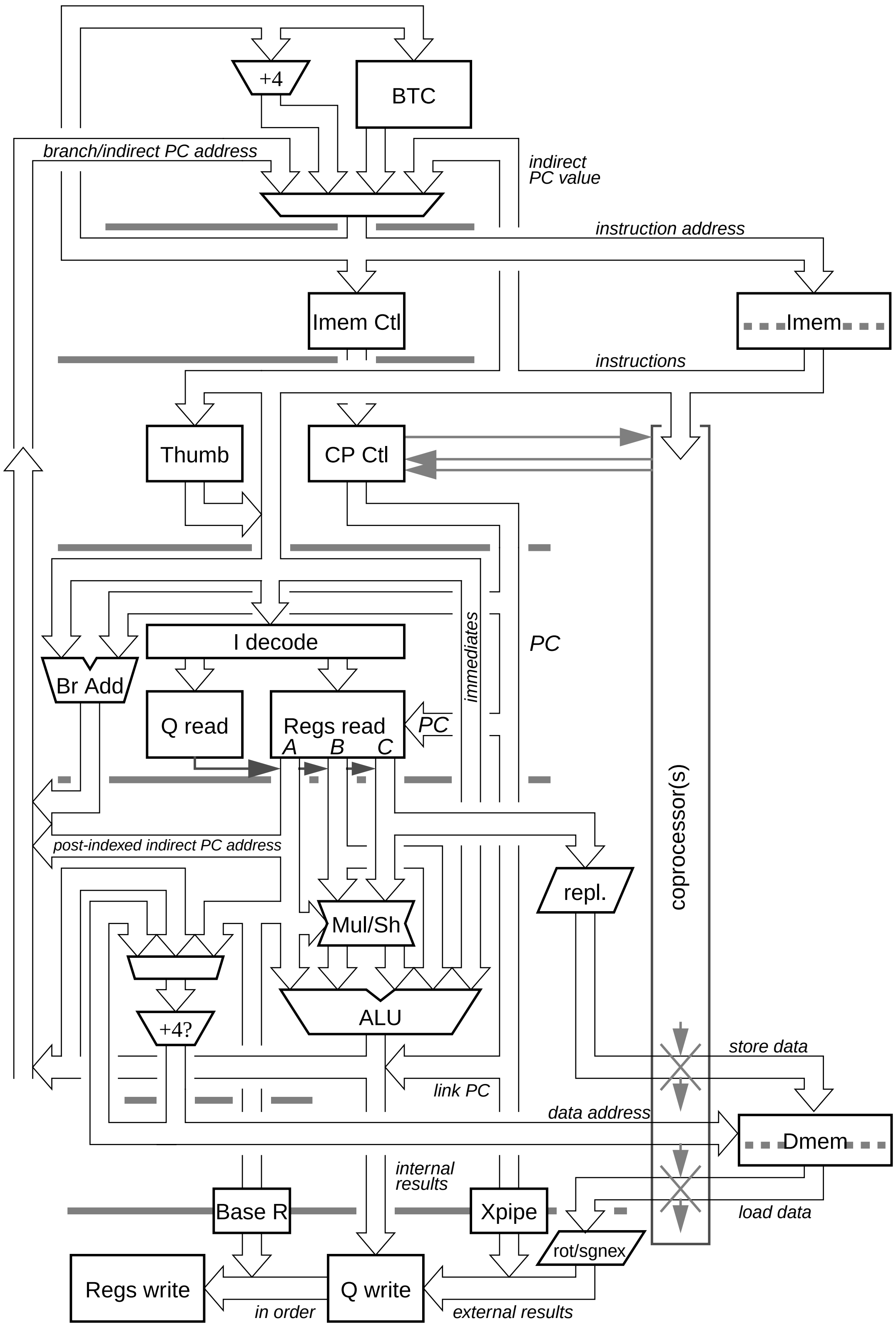


Figure 3-3: AMULET3 organization (Courtesy of Prof. Steve Furber)

Instruction prefetch unit 4

This chapter describes the configuration and functions of the instruction prefetch unit (IPU) in AMULET3 to help the reader to understand the function of the branch target cache (BTC).

4.1 Overview

Programs are stored in a memory and a processor fetches appropriate instructions and data and runs the instructions. The traditional approach to connecting the processor to the memory started with the simple idea called the von Neumann method. All instructions and data are stored together in a single memory. Instruction and data addresses are generated in a dedicated block in the processor, normally called the address generation unit. From the viewpoint of the memory, instructions and data are treated the same. This approach has a big disadvantage whenever a program accesses instruction and data addresses in turn. A large burden is imposed on the address generation unit in order to handle the instruction and data addresses together and as a consequence performance falls. A solution is to divide the address generation unit into two separate units, the instruction and the data address generation blocks, and to have separate instruction and data memories. This is normally called a Harvard architecture.

4.2 Instruction prefetch unit

In AMULET2e, the IPU was included in the address interface unit. The address interface unit generated both instruction and data addresses. For AMULET3, a Harvard architecture is introduced and the instruction address and data address interfaces are separated. Each is connected to its own memory port through its own bus. The first AMULET3 system incorporates a dual ported memory, so the instruction and data address interfaces have independent access to a unified instruction and data memory.

4.2.1 Configuration

First organisation

The high level specification of, and interface to, the IPU were defined by the AMULET3 design team, led by Dr. Jim Garside. The author was responsible for translating the specification into a detailed VLSI implementation. The original proposed organisation of the IPU is shown in figure 4-1.

This organisation has the forward path from the memory address register multiplexer (MARMUX) to the memory address register (MAR) and the program counter register (PC) via the exception unit (EU), the branch target cache (BTC) and the program counter multiplexer (PCMUX). There is also the backward path from the PC to the MARMUX.

The normal instruction address path starts from the MARMUX, which accepts the instruction address either from the PC or from the ALU, and produces the program counter address to the EU. The EU checks whether or not an exception has happened.

(The exceptions are explained in section 4.2.3.) The result from the EU is sent to the MAR, the INC and the BTC. The MAR waits for the result from the BTC.

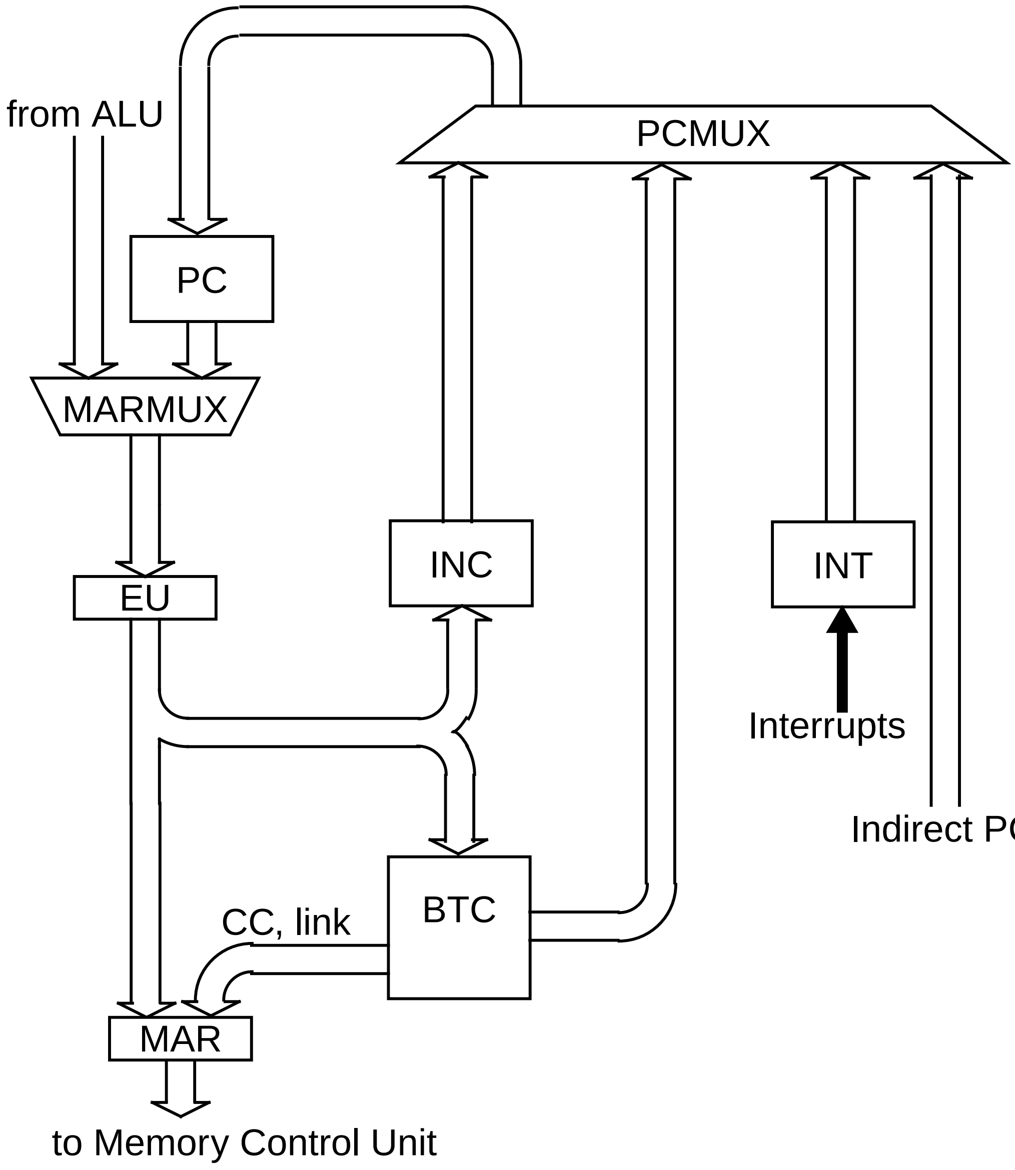


Figure 4-1: First IPU organisation

If a hit happens in the BTC, the condition code and the link bit of a predicted branch in the BTC go to the MAR, and the MAR sends the program counter address with these condition code and link bits to the memory control unit. The BTC also sends a branch target address to the PC.

If there is no hit in the BTC, the MAR sends the program counter address to the memory control unit, and the incremented result from the INC goes to the PC.

When an interrupt or indirect PC load happens, an interrupt vector or the indirect program counter address takes the path to the PC instead of the BTC or the INC.

In this organisation, the critical path is from the MARMUX to the PC and the MAR through the EU and the BTC. The forward path is long and the backward path is very short. To reduce the long processing time caused by this long serial forward path, an attractive organisation was proposed by the author as shown in figure 4-2.

Second organisation

As was mentioned above, the first organisation has the long serial forward path. In order to reduce this forward time the author proposed the second organisation which put more emphasis on the backward path as shown in figure 4-2. This organisation has the forward path from the MARMUX to the MAR via the EU. Unlike the first organisation, the MAR does not wait for the result from the BTC. The result from the EU goes to the BTC and the INC.

When there is no hit, the incremented result from the INC goes to the PC. When a hit happens, the condition code and the link bit go to the PC, and the PC accepts the program counter address from the INC. This is different from the first organisation. At this time, the target address is stored at the latch in the BTC and the flag indicating that a hit happened is set. In the next cycle, the PC gets this latched target address instead of the incremented address from the INC.

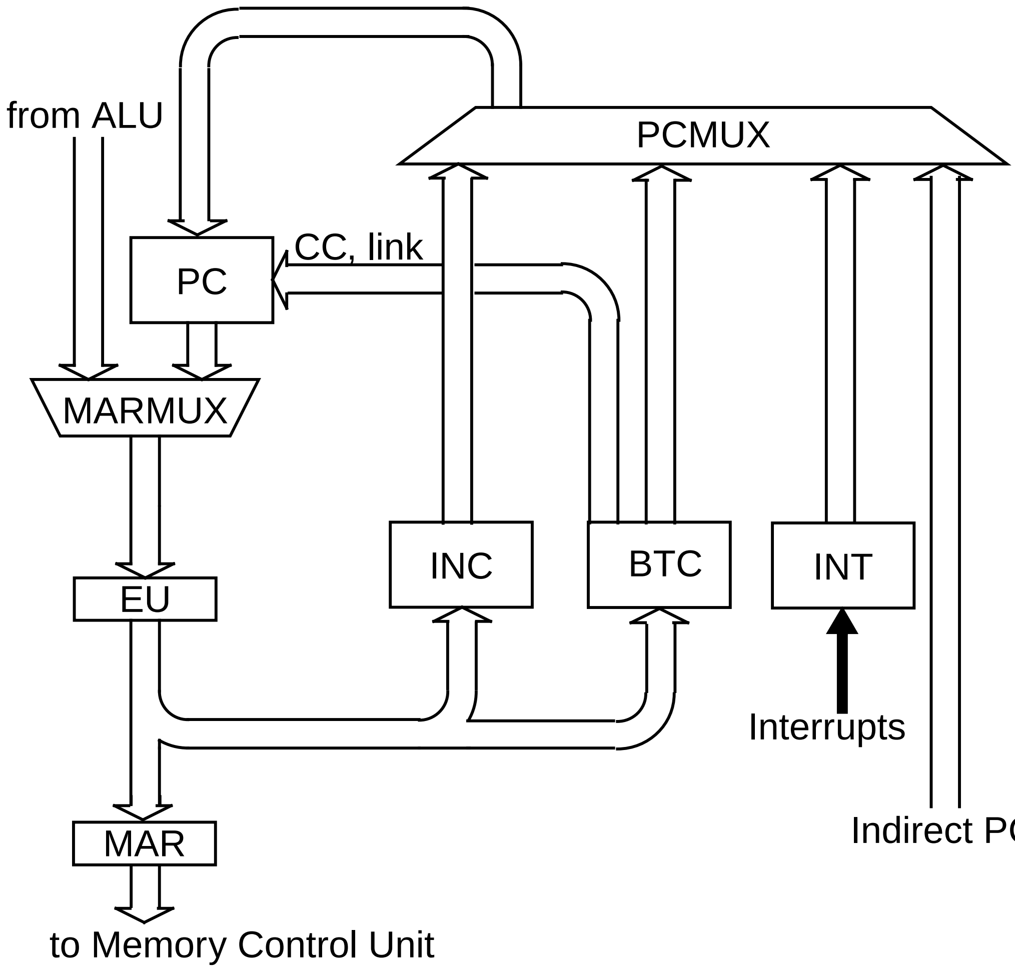


Figure 4-2: Second IPU organisation

Comparison between first and second organisations

The second organisation was based on the assumption that the total processing time of the AMULET3 microprocessor would be shorter if the forward path processing time in the IPU was faster even though the IPU cycle time was same. (The cycle time is the sum of the forward time and the backward time.) But, simulation using the LARD hardware description language with the dhrystone test program gave a different result as shown in figure 4-3. Each pipeline cycle time in the AMULET3 LARD model was set as 100 time units, which is a nominal value. Each block in the IPU was set as 20 time units. The figure shows how the BTC processing time affects the total simulation time to finish the dhrystone test program. Before the BTC processing time reaches 90 time units, the first organisation has better results. This means that there is little impact from the forward time in the IPU on the AMULET3 processing time. After the BTC processing time exceeds 90, the forward time burden in the IPU can be a major obstacle to the system simulation run time. But, in this case, the cycle time in the IPU is too long to meet the AMULET3 specification. Therefore, the first organisation was chosen for the IPU.

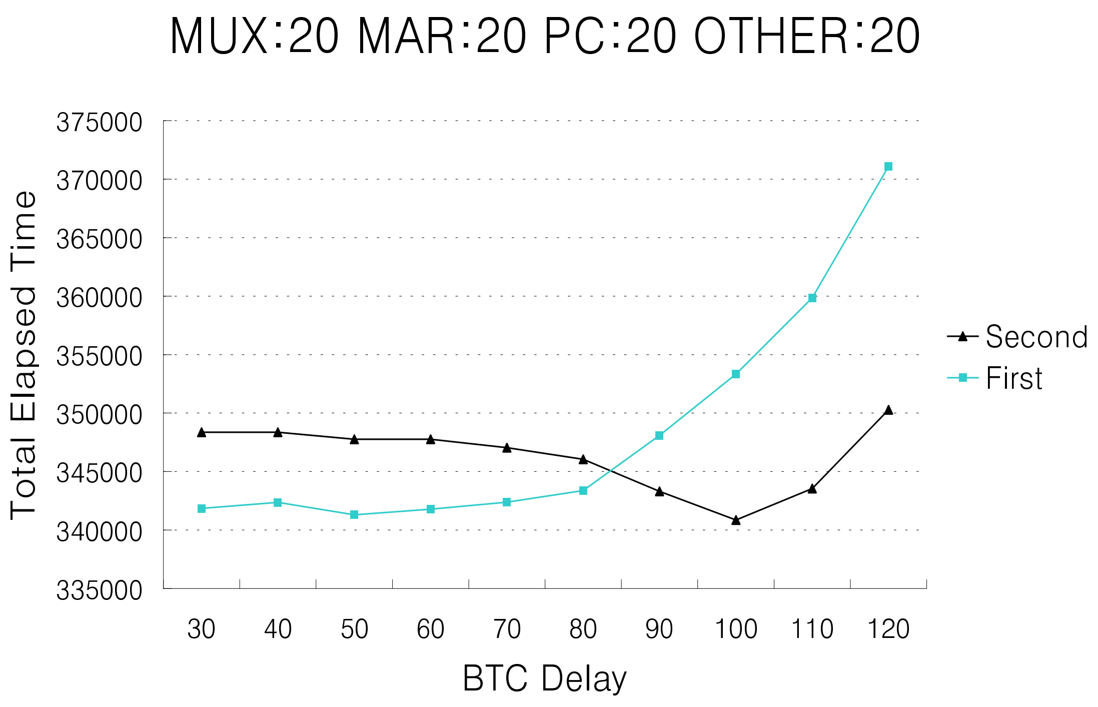


Figure 4-3: Simulation result comparing first and second IPU organisations

Need for another organisation

Although the first organisation was chosen for the IPU, an unexpected problem arose when the author tried to implement the IPU schematic. As was mentioned earlier, the first organisation has the long serial forward path from the MARMUX to the MAR and to the PC via the EU and the BTC. After the author implemented the schematic, it was evident that this organisation could not be used, because the forward path time of the schematic implementation is too slow to meet the AMULET3 specification. The AMULET3 specification requires that the IPU cycle time must be less than about 6.5ns, which is equivalent to about 150 MIPS in a synchronous ARM microprocessor. The schematic implementation of the first organisation took about 13ns cycle time. This result was measured by counting the number of request and acknowledge signal inversions. It was assumed that it takes about 0.2ns for one inversion of the signal, which is equivalent to an inverter delay time. So the author proposed a third parallel organisation as shown in figure 4-4.

New parallel organisation

The new parallel organisation was proposed by the author as shown in figure 4-4. The forward path of this organisation is from MARMUX to the MAR and to the PC via the EU, the INC, the BTC, or the IND. In the first organisation, the EU and the BTC were connected serially. In the new parallel organisation, every unit is located in parallel after the MARMUX. This can reduce the forward time significantly compared to the first organisation. Since the processing time in the BTC is the longest of the parallel blocks, we can assume the total cycle time is defined as follows: the time for the MARMUX +

the time for the BTC control circuits + the time for the BTC itself + the time for the PCMUX + the time from the PC to the MARMUX.

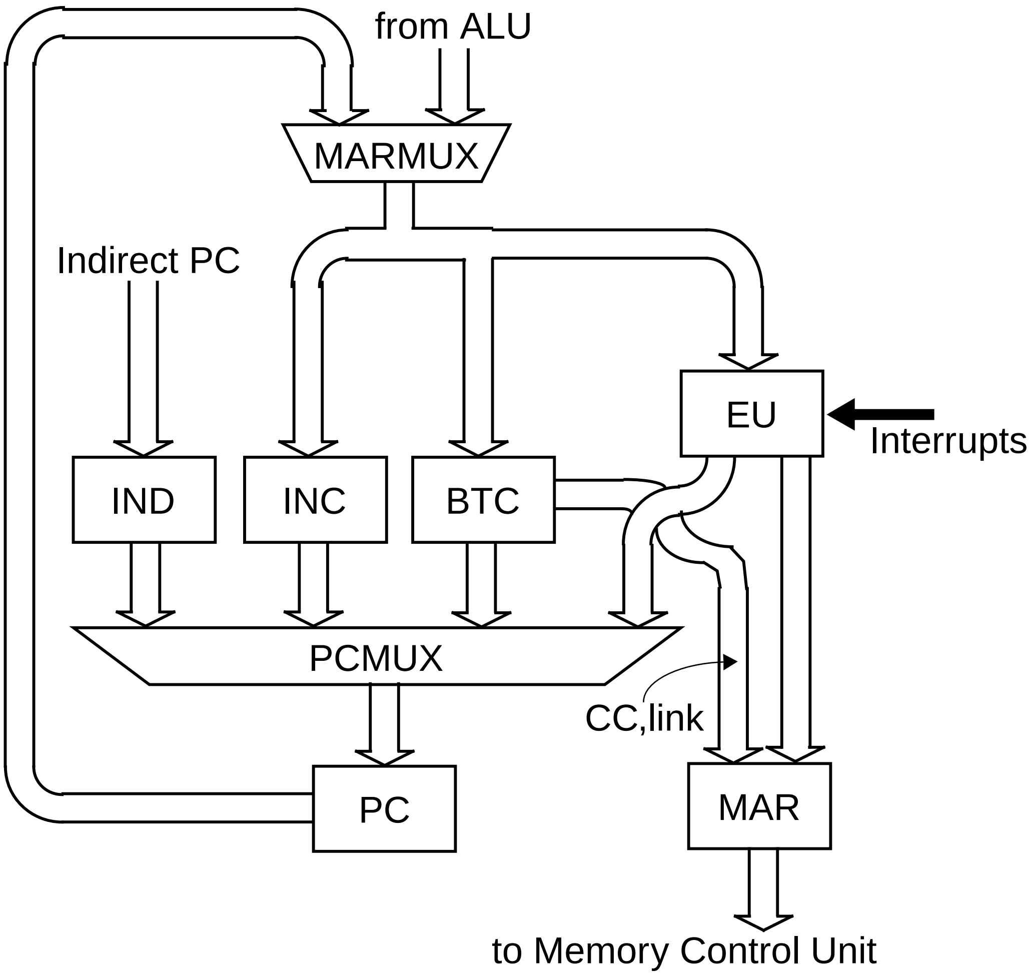


Figure 4-4: New parallel organisation

In normal instruction execution, the result of the MARMUX, which is either from the PC or from the ALU, goes to the MAR via the EU, and the incremented program counter address from the INC is stored into the PC.

When a hit happens in the BTC, the target address goes to the PC and the condition code and the link bit go to the MAR. The MAR gets the present PC from the EU and sends it together with the condition code and the link bit to the memory control unit.

When an exception happens, the EU detects it and sends an exception vector address to the PC and the MAR. During the next cycle, the IPU does nothing but increment the program counter address in the INC and store the incremented address into the PC, because the MAR already had an exception vector address in the previous cycle, and the next address in the MAR must be the exception vector address + 4. For example, assume the instruction address from the MARMUX is 100 and the EU detects the software interrupt. The EU will produce the address 8 to the PC and the MAR. The MAR sends 8 to the memory control unit. The PC sends 8 to the MARMUX. The MARMUX sends 8 to each of the parallel blocks. But, this time only the incremented value 12 will go to the PC and the MAR does nothing. Then the PC will send 12 to the MARMUX and normal operation will be continued. This seems a waste of an IPU cycle. But exceptions are very rare, and therefore this redundant cycle affects little the total performance. This was verified by LARD simulation using the dhrystone program and the results are shown in figure 4-5.

Comparison between first and new parallel organisations

A comparison between the first and the new parallel organisation is shown in figure 4-5. If the IPU cycle time is the same, the simulation time of the new parallel organisation increases very slightly since there are redundant cycles after exceptions. But, with the new parallel organisation, the IPU cycle time can meet the specification, which is about 6.5ns. Because the IPU schematic of the first organisation has about 13ns cycle time, it can be assumed that its cycle time is almost double that of the new parallel organisation. In this case we can compare the cycle time of 100 time units for the new parallel organisation with the cycle time of 200 time units for the first organisation in figure 4-5. It is evident that the new parallel organisation produces a faster simulation time.

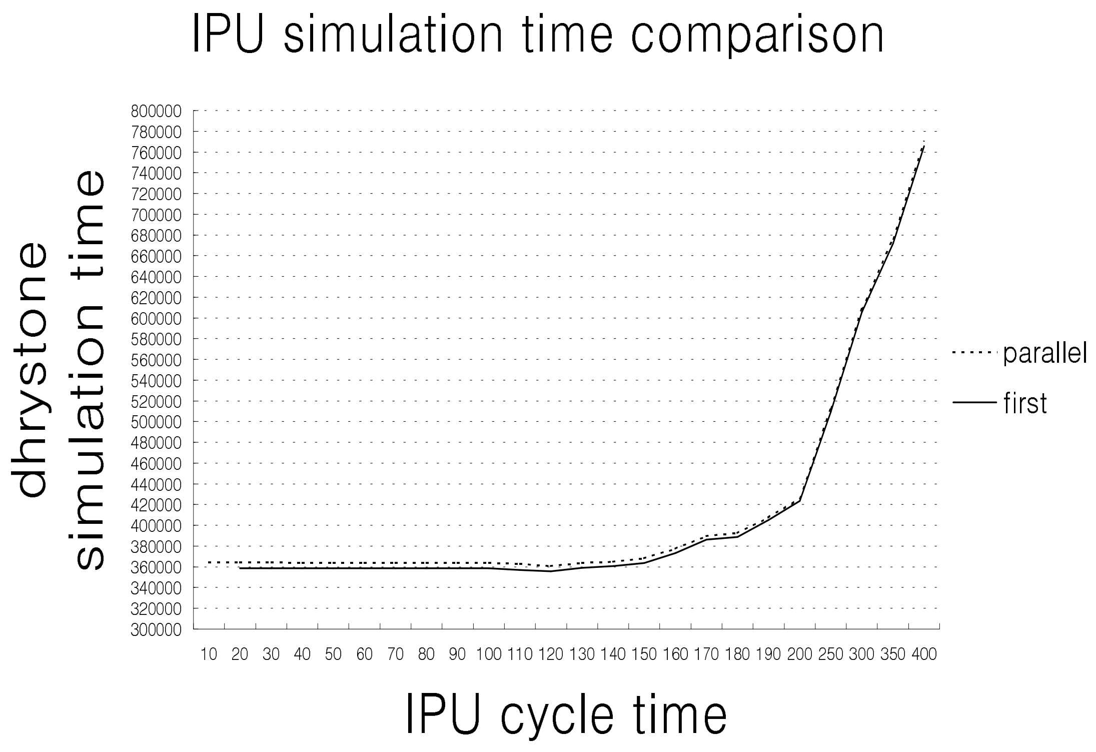


Figure 4-5: Simulation result comparing first and parallel IPU organisations

4.2.2 Functions

The new parallel IPU has five major functions as follows:

1. Program Counter Incrementing
2. Branch Address Management
3. Interrupt Handling
4. Indirect Program Counter Loading
5. Halt

Each function is explained as follows:

- *Program Counter Incrementing:* THUMB is a compressed representation of the ARM instruction set; ARM instructions have a 32 bit length whereas THUMB instructions have a 16 bit length. In the ARM processor the two instruction sets can be used alternately but not mixed. Thus two different instruction execution modes are available in one ARM processor. The first is called ARM mode where 32 bit instructions are used, and the second THUMB mode where 16 bit instructions are used. The current instruction address is incremented by 4 bytes to produce the next instruction address in ARM mode and, in THUMB mode, the present instruction address is incremented by 2 bytes under the normal instruction sequence. Exceptional cases causing a deviation from the normal instruction sequence are a branch, an interrupt, and an indirect PC. In these cases the present address changes depending on each situation. In AMULET3, the present instruction address always can be incremented by 4 bytes under normal sequential execution as THUMB instructions are fetched in pairs. This function is performed in the incrementer.

- *Branch Address Management:* One of the exceptions from the normal instruction sequence is a branch. If there is a branch in the program, the branch address is calculated in the ALU and inserted into the IPU. This address goes to MAR via the MARMUX and the EU. When a branch happens, the source address and the target address of the branch are stored in the BTC. The source address is placed in the Content Addressable Memory (CAM) in the BTC and the target address is stored in the associated RAM memory. At every fetch cycle the present instruction address is passed from the MARMUX to the BTC and is compared with addresses in the CAM to see whether a matched address exists. If there is a matched address in the CAM, the associated target address in the RAM goes to the PC, instead of the result of the incrementer, via the program counter multiplexer (PCMUX). Thus branch prediction is accomplished. This mechanism is explained in chapters 5 and 6.

- *Interrupt handling:* There are seven interrupts (These are called exceptions in the ARM processor manual [74]. In table 4-1, the term exception is used instead of interrupt.) in the ARM processor as shown in table 4-1. Depending on each interrupt, the EU produces the appropriate vector address. This address goes to the MAR and to the PC instead of the result of the incrementer via the PCMUX.

Table 4-1: Exception processing mode

Exception type	Mode	Vector address
Reset	SVC	0x00000000
Undefined Instructions	UNDEF	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch Abort (Instruction fetch memory abort)	ABORT	0x0000000c
Data Abort (Data access memory abort)	ABORT	0x00000010
IRQ (Interrupt)	IRQ	0x00000018
FIQ (Fast Interrupt)	FIQ	0x0000001c

- *Indirect Program Counter Loading:* The indirect program counter loading happens when a load multiple instruction (LDM) containing the PC in the register list or any single register load (LDR) to the PC is executed. This causes the PC to be loaded from the memory via the PCMUX.

- *Halt:* The ‘halt’ mechanism can be initiated when a program enters an idle loop, and is implemented using the ARM ‘B .’ instruction. This normally causes an instruction to loop back continuously to itself until an interrupt occurs, which is clearly wasting power and doing no useful work. Instead, as in AMULET2e, AMULET3 will halt the processor by blocking a local handshake until an interrupt occurs. Blocking one handshake causes a local stall which rapidly propagates through the system, reducing the system activity to zero. This mechanism can be implemented easily by intercepting and disabling request or acknowledge signals. In the IPU, this mechanism is implemented by intercepting a request signal in the unit until an interrupt occurs. For example, the request from the MARMUX is grabbed by the halt function unit and then the processor will stop operating since there is no more updating of the present instruction address in the MAR. After receiving an interrupt, this reserved request signal will be released and then the processor will operate again.

Branch Prediction Mechanism 5

To improve the performance of AMULET2e, a branch prediction unit was introduced [13]. Dynamic and static algorithms for branch prediction were fully investigated [14], then a branch target cache (or branch target buffer) was chosen because of the ease of integration into the AMULET2e implementation.

From the viewpoint of low power technology, the branch target cache of AMULET2e seems to have a problem. It attempts a branch prediction during every instruction prefetch cycle regardless of the frequency of branch instructions, checking whether the present instruction memory address corresponds to a previously executed branch instruction. At every instruction fetch, significant circuitry is activated to search for an address match, consuming significant power. Therefore in the sense of low power implementation, using a branch target cache as a branch prediction unit for AMULET2e has a drawback. However, research has shown that there can be a performance benefit from adopting branch prediction, especially where a pipeline structure is used [77], and total system power can be saved since the frequency of wasted memory accesses for instruction prefetch is reduced (as shown in [13]).

From the silicon test results on AMULET2e shown in [13], the power-efficiency of the core drops by 5% when the branch target cache is turned on, though the overall system power-efficiency rises by 4% when the code is being executed from the external memory due to the reduction in wasted instruction fetches. Thus, AMULET2e used a branch prediction unit using a branch target cache to achieve better performance and to save power. Furthermore, the comparison logic of the AMULET2e BTC is divided into two areas as shown in figure 5-1: high bits and low bits. Since most instruction fetches run

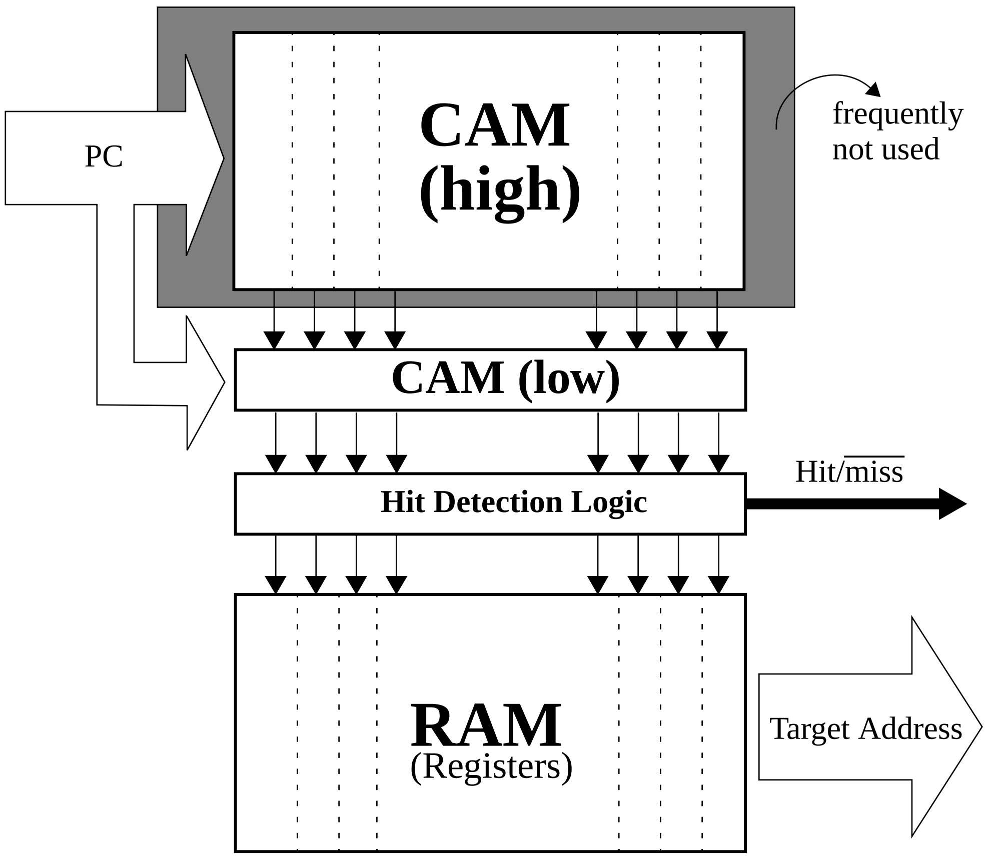


Figure 5-1: BTC structure

sequentially, the high bits change rarely and the high section of the Content Addressable Memory (CAM) need not be invoked. Therefore only the low bits are normally compared with the present program counter address. This is the power saving scheme of the BTC, which saves around 70% of the power consumption of the CAM and also reduces the average look-up time improving performance. From the test results on AMULET2e shown in [13], the power-efficiency of the core with this segmentation of the CAM is almost the same as without the BTC due to the reduction in wasted instruction fetches. This means that when the CAM segmentation is used in the BTC the core power-efficiency loss is eliminated showing that branch prediction can be power neutral with respect to the core when carefully designed, and can contribute significantly to overall system performance and power-efficiency.

A very similar branch prediction algorithm is used in AMULET3. (AMULET3 is an ongoing project, so the hardware implementation, proposed in this thesis, could change later.) However, new functions have been added and a previously existing block has been improved. Firstly, the THUMB instruction mode [15] has been added. Secondly, the condition code and the link bit of a branch address are stored in the Random Access Memory (RAM) of the branch prediction unit together with the target address in order to avoid fetching predicted branches from memory. Thirdly, the number of entries in the CAM and the RAM in the branch prediction unit has been increased from 20 in AMULET2e to 32. Detailed explanations of these three features are given in section 5.3.

5.1 Basic concepts

According to Hennessy and Patterson [77], about 20% of the 80x86 instructions in the five SPECint92 programs are categorised as branches, of which about 80% are conditional branch instructions. This suggests that microprocessors using pipeline techniques can have a branch penalty every six or seven instructions. In general, the deeper the pipeline, the worse the branch penalty. For example, architectures with very deep pipelines, such as the DEC Alpha [78] and MIPS R4000 [79], suffer a heavy pipeline penalty for mispredicting a branch (up to 10 cycles [78]). In order to address this loss of performance, we should focus on two issues as follows:

1. Detection of whether or not the branch is taken early in the pipeline: the earlier a processor knows whether the branch is taken or not, the fewer unwanted instructions following the branch instruction which enter the pipeline. Of course, when the branch is not taken, there is no difference. When the branch is taken, however, a processor need not fetch instructions following the branch instruction. Those instructions which have entered the pipeline will need to be discarded. This is why the detection of whether or not the branch is taken early in the pipeline is important.

2. Knowing the address of the branch target earlier in the pipeline: even if a processor can decide whether or not the branch is taken early in the pipeline, the branch instruction will stall in the pipeline if the branch target address cannot be calculated in time. If the target is specified indirectly, for example using the contents of a register or memory location, and the branch target is not in the data/instruction cache, this will cause the pipeline to stall until the target can be fetched from the external memory.

5.1.1 Branch target prediction schemes

The first issue is related to the branch prediction strategy. Many branch prediction strategies have been investigated in the quest to improve the performance of pipelined microprocessors.

Branch prediction strategies can be divided into two groups: static and dynamic predictors. Static predictors are so-named because the action taken does not depend on dynamic program behaviour. Dynamic prediction means that the prediction will change if the branch changes its behaviour while the program is running.

Static branch prediction schemes use information gathered before program execution, such as branch opcodes or profiles, to predict the branch direction. The simplest form of these predicts that all conditional branches are taken, as in MIPS-X [80], or are not taken as in the Motorola MC88000 [81]. Other static prediction schemes can be based on the opcode or on the direction of the branch as in “if the branch is backward, predict taken; if forward, predict not taken” [82]. This scheme is effective for loop intensive code, but does not work well for programs where the branch behaviour is irregular. Some processors [83] allow the compiler to pass prediction information to the hardware with additional hint bits. Run-time profile information from program execution is typically used to predict branches statically. This profile-based branch prediction is based on the results determined by profiling the program on a training input data set [84]. Unfortunately, branch behaviour for the sample data may be very different from the data that appears at run-time.

To get more precise results from branch prediction schemes, it is essential to use run-time information. Dynamic branch prediction algorithms use information gathered at run-time to predict branch direction. Smith [82] proposed a branch prediction scheme using a table of two-bit saturating up-down counters that is incremented when the branch is taken and decremented when it is not; the most-significant bit is used to predict the future direction so that the branch is predicted taken if this bit is set and not taken if reset.

Each branch is mapped via its address to a counter. The advantage of the two-bit method is that a single unusual iteration will not change the predicted direction.

For further improvement in prediction accuracy, Yeh and Patt [85] proposed the two-level branch predictor. Their algorithm is based on the fact that more history information can enable greater branch prediction accuracy. In order to achieve this, two levels of branch history information are used. The first level is the history of the previous *k* branches encountered. The second level is the branch behaviour for the last *s* occurrences of the specific pattern of these *k* branches. The two level branch predictor uses one or more k-bit shift registers, called Branch History Registers (BHR), to record the branch outcomes of the most recent *k* branches. It uses one or more arrays of 2-bit saturating up-down counters, called the Pattern History Table (PHT), to keep track of the more likely direction for branches. The lower bits of the branch address are used to select the proper PHT and the contents of the BHR are used to choose the appropriate 2-bit counter within that PHT.

Because the complete two level branch predictor requires a time-consuming pair of lookups, commercial processors generally use a simplified version in which a global history value is used to index into the history table. Pan, So, and Rahmeh proposed [86] a derivative of this algorithm, called Gselect [87], in which the counter table is indexed with a concatenation of the global history and some bits of the branch address. Since the same global history patterns can occur for different branches during program execution, the global history pattern can be less efficient at identifying the current branch than the branch address itself. To overcome this disadvantage, McFarling proposed Gshare [87], another derivative of the global history two-level predictor which XORs the global history with the branch address to index the PHT. This algorithm is used in several recently announced microprocessors and is likely to become standard practice as designers revise their processors over the next couple of years [88]. Hybrid branch predictors have recently been proposed in order to improve prediction accuracy further [87][89][90]. A hybrid branch predictor comprises two or more single-scheme predictors and a mechanism to select among these predictors.

In conclusion, Gshare proposed by McFarling is expected to become a standard in industry over the next couple of years as explained above. However, it is very hard to say which branch prediction scheme must be used for every case.

Remember that AMULET2e adopted a simple branch prediction scheme not because a sophisticated branch predictor cannot be developed for AMULET2e but because even a simple predictor can deliver a good result with a limited silicon resource. Since AMULET3 is an embedded microprocessor for a communication application, it cannot allow the branch target cache to take an excessive area to improve performance. If some of the schemes described above were used for AMULET3, the core performance might increase but at a significant cost in silicon area. Since an embedded microprocessor should put more emphasis on area than the general-purpose microprocessor, a similar scheme to that used in AMULET2e was adopted for AMULET3.

5.1.2 How to get the branch target address earlier

As for the second issue, we should consider the types of addressing methods in the instruction. When the target address is pointed to with direct or absolute addressing, there is no problem to get the target address. The target address can be produced directly from an instruction in the decoding stage in the pipeline, but specifying a full 32 bit address within the instruction is not practical. To get the target address in the decoding stage, PC-relative addressing requires an additional separate branch adder since the main ALU is busy dealing with an earlier instruction. If the target is specified with indirect addressing, for example using the contents of a register or memory location, getting the target address in the decoding stage is very difficult since a pipelined microprocessor cannot evaluate a register or memory location without recognizing whether an earlier instruction is about to alter its contents.

Other solutions are based on the cache structure. When a branch is taken, the target address of the branch is stored in a cache and this will be used the next time the same branch address is predicted taken. This results in efficient and fast target address submission to the instruction memory and we do not need to use an additional branch adder and logic to calculate the target address. A variety of branch prediction schemes can be coupled with this branch target address cache.

5.2 AMULET2e branch target cache

Various static and dynamic branch prediction schemes had been investigated prior to the implementation of the AMULET2e branch predictor. The choice was heavily influenced by the ease of integration into the AMULET2e implementation as shown in figure 5-2. In view of this, the dynamic Branch Target Cache (BTC) was chosen [14].

Figure 5-1 shows the structure of the BTC in AMULET2e. The CAM and the RAM can store 20 words and each word has 30 bits. The CAM stores words which can be compared against an input address word (bit31...bit2) as shown in figure 6-2. A match detection signal which is the hit/$\overline{\text{miss}}$ line in figure 6-2 is sent by the CAM to indicate whether or not a value stored in the CAM array matches with the input address word. A CMOS implementation for one of the CAM word lines is depicted in figure 5-5. It is readable and writable just like an ordinary RAM cell. During the precharge phase, the hit/$\overline{\text{miss}}$ line is precharged high by the active low $\overline{\text{Precharge}}$ signal, the Write signal is low, and the Bit lines are predischarged low. During the look-up operation, the $\overline{\text{Precharge}}$ signal is inactive. If the word in the CAM does not match, the hit/$\overline{\text{miss}}$ signal is discharged low. Otherwise the hit/$\overline{\text{miss}}$ signal will be preserved high and this means that there is a matched word in the CAM.

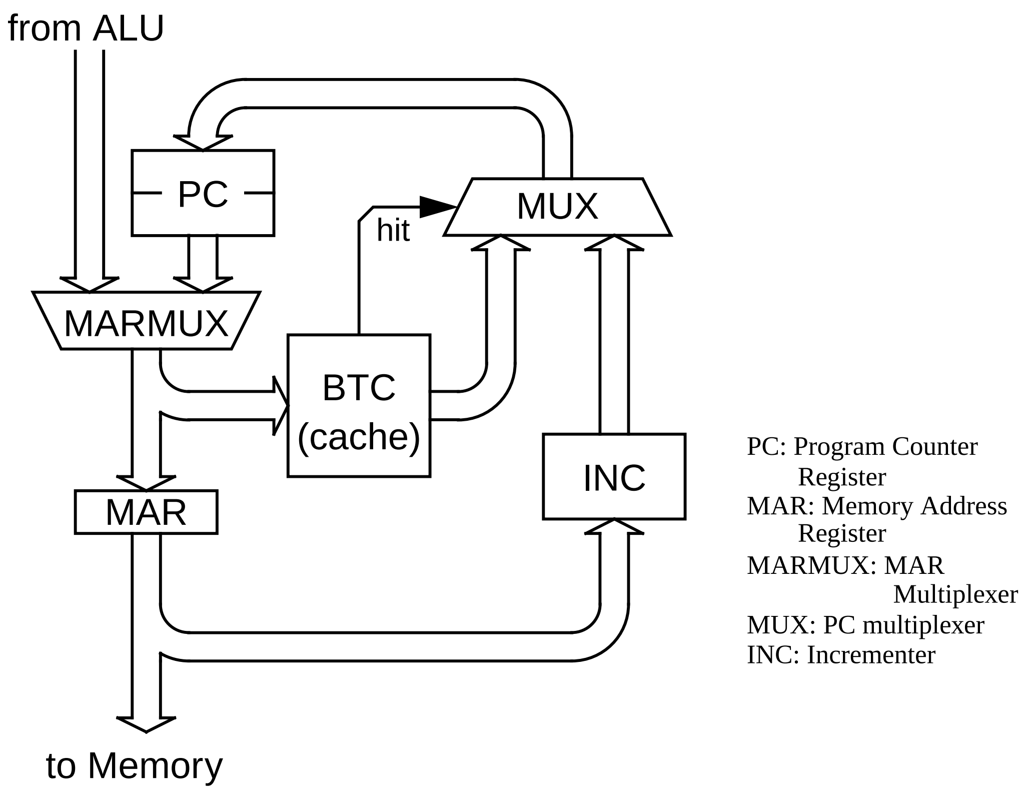


Figure 5-2: Address interface of AMULET2e (Courtesy of Jim Garside)

As for the write procedure, when a new branch target address, which is to be stored in the RAM, enters from the ALU into the main instruction address stream, the original branch instruction address is also inserted into the BTC and stored in the CAM as depicted in figure 5-3.

Consider the look-up operation in the BTC. The BTC accepts an address from either the ALU or the PC and compares it with the stored addresses in the CAM at every instruction fetch cycle. If the same address is found in the CAM, the BTC indicates this via the hit signal as explained above. The hit signal indicates that the same address was used previously for a branch instruction address and it is predicted to be taken. The hit signal indexes the appropriate target address out of the RAM, which was previously loaded when the branch instruction jumped to the target address before. This procedure is depicted in figure 5-4.

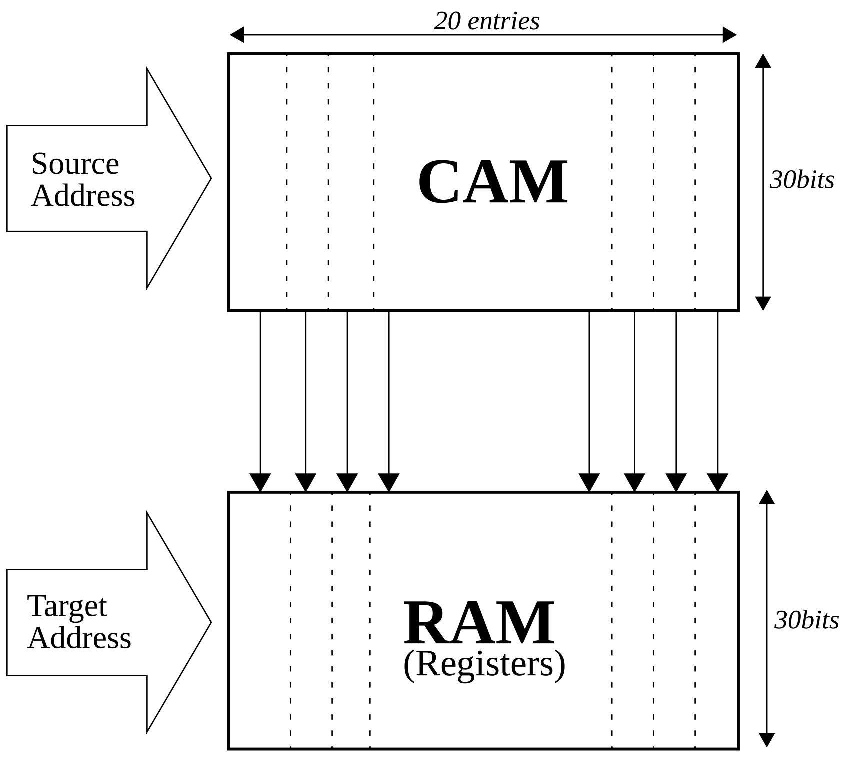


Figure 5-3: Write operation in the BTC

The look-up is a simple and efficient mechanism and, because of this, the BTC was chosen for the branch prediction function in AMULET2e.

Since the BTC takes a longer time to produce its result than the incrementer, and is located in parallel to the incrementer as seen in figure 5-2, it is part of the critical path for the performance of the address interface in AMULET2e. To reduce the layout area, the comparison logic is implemented using dynamic circuitry. This is configured with a precharged, wire-ORed miss line as shown in figure 5-5 [92].

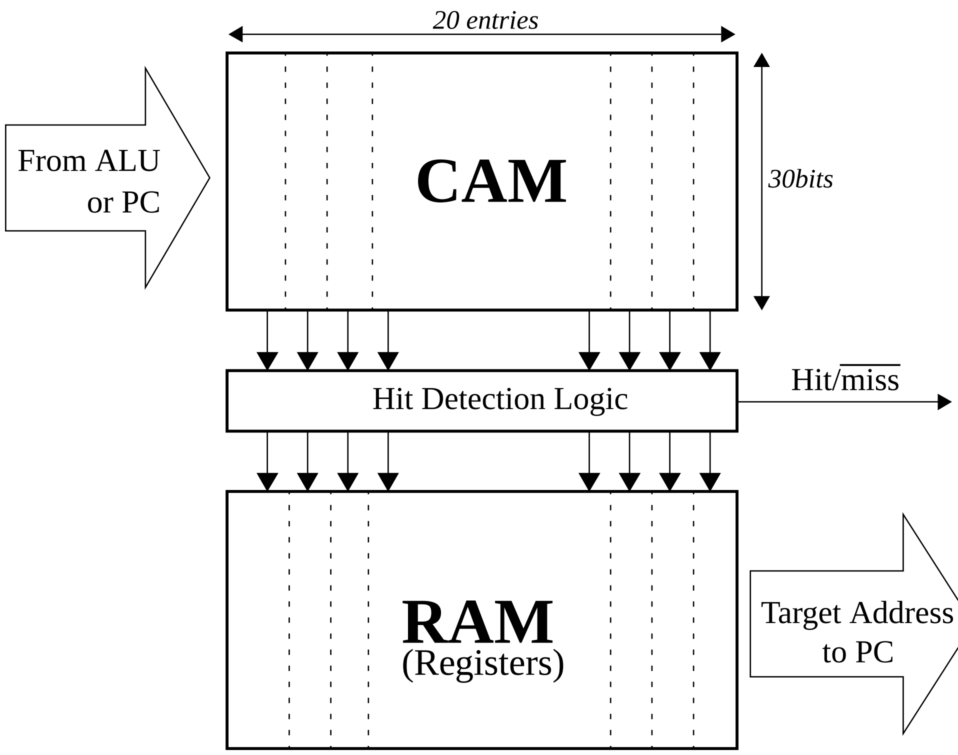


Figure 5-4: Look-up operation in the BTC

To reduce the power consumption in the BTC, each CAM entry is split into two parts as shown in figure 5-1. These parts may perform separate comparisons; only if both parts indicate a hit is the address recognised. The advantage of this mechanism is that the address bits in the upper section (26 bits in AMULET2e) rarely change, since most instruction fetches run sequentially (about 75% of instruction fetches [14]). Thus by comparing only the lower section (4 bits in AMULET2e) in most cycles, the power consumption of the split BTC can be reduced to about 30% of that of a more simplistic design by the test result of AMULET2e shown in [13]. There is an explanation in section 6.2.2 of how and when the upper and lower sections are activated.

5.3 AMULET3 branch target cache

Fundamentally, the BTC of AMULET3 is similar to that of AMULET2e. However, there are major improvements implemented in the BTC of AMULET3.

1. To support the THUMB instruction set, the functionality of the THUMB mode is added.

2. The condition code and the link bit are stored in the RAM so no instruction fetch is needed for a predicted branch.

3. To improve performance, the number of CAM entries is increased from 20 to 32.

The instruction prefetch unit (IPU) of AMULET3 is depicted in figure 4-4. The BTC receives its address either from the branch channel which contains the branch instruction address or from the PC channel which contains the next address. It sends the target address to the PC register in the case of a hit.

THUMB is a compressed representation of the ARM instruction set; ARM instructions are 32 bits long whereas THUMB instructions are 16 bits long but with a similar function

[15][74]. Two modes, the ARM and the THUMB mode, are available in a single processor. In the instruction memory of AMULET3, all the contents are handled as 32 bit quantities. There is no difference between the ARM and the THUMB mode from the viewpoint of the instruction memory. This means that two THUMB instructions are fetched simultaneously and then decoded separately in the instruction decoder.

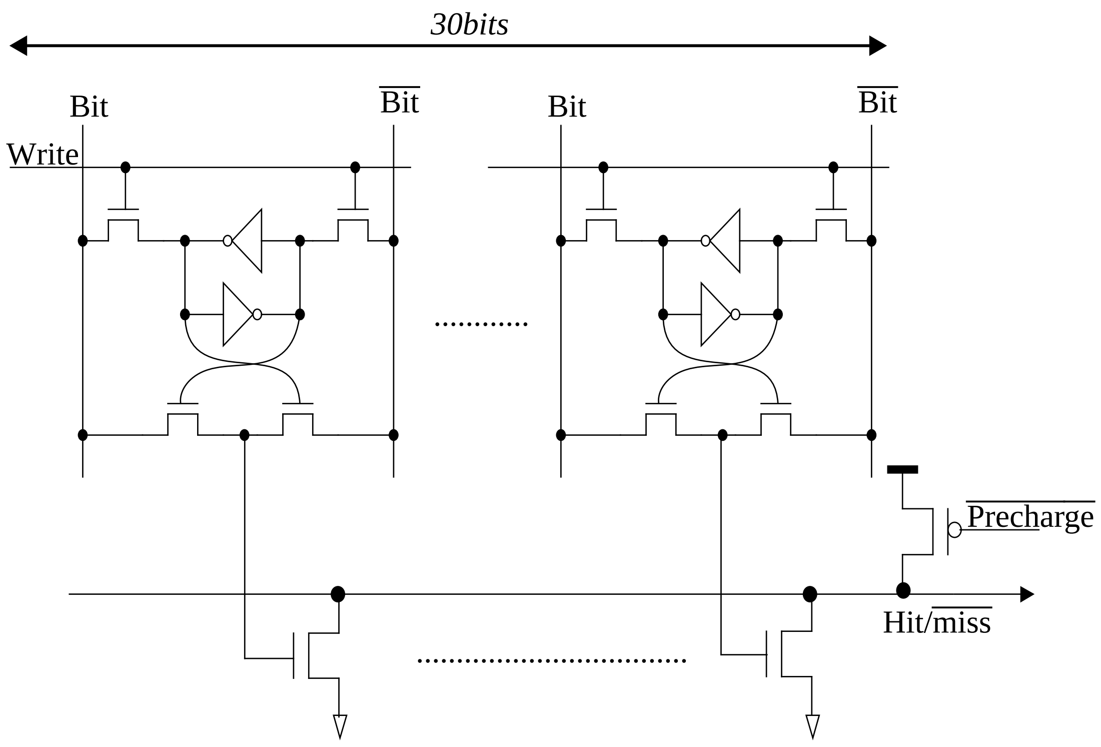


Figure 5-5: CAM comparison circuitry

In ARM mode, a branch destination can only be a multiple of 4 bytes, but in THUMB mode, after a branch, an address could be a multiple of 2 bytes and thus only one of the instructions in the 32 bit word is required. In ARM mode, bit<1> and bit<0> of the instruction address in the IPU are always ‘00’. On the other hand in THUMB mode, bit<0> of the instruction address in the IPU is always ‘0’ and bit<1> of the instruction address in the IPU could be either ‘0’ or ‘1’. Therefore bit<0> of the instruction address in the IPU is never used and all instruction addresses in the IPU have 31 bits. In THUMB mode, when there is no branch instruction, the instruction address also increases like in ARM mode. For example, instruction addresses increase through 0, 4, 8 and so on in THUMB mode. When the address 0 goes to the instruction memory, a 32 bit word comes out of the instruction memory and this will be divided into 2 parts; the higher 16 bits of the instruction correspond to the address 2, and the lower 16 bits of the instruction correspond to the address 0. Then address 4 goes to the instruction memory and this will yield two instructions with addresses 4 and 6. However, after a branch, the instruction address in THUMB mode could be an address which is an odd multiple of 2 bytes. For example, the instruction address could be 6 if this is a target address of a branch instruction. In this case, the contents of the instruction address 4 are fetched from the instruction memory and only the higher 16bits are decoded. This feature confuses the operation of the BTC CAM, since a word in the CAM has 30 bits. For example, the addresses 4 and 6 have no difference in the CAM. It is possible to make words in the CAM have 31 bits, but this will increase the layout area. In order to solve this matter the following novel scheme was proposed.

To support THUMB code, the CAM is divided into two equal sized sections, “odd” and “even”. In THUMB mode, a branch instruction address must be cached in the odd section if bit<1> of its address is high or in the even section if bit<1> of its address is low. For example, whereas the source address 4 is stored in the even section, the source address 6 is put in the odd section. In ARM mode, a branch instruction can be stored in either section. For example, the source address 4 could be stored either in the odd section or in the even section. This is implemented by attaching simple mode detection circuitry to the input lines of the CAM address decoder as shown in figure 5-6 and appendix A.4. The number of lines in the CAM address decoder is 5 bits to support 32 entries, and mode detection circuitry is attached to the most significant bit. In the RAM, a 31 bit target address is stored, unlike the 30 bits used in AMULET2e.

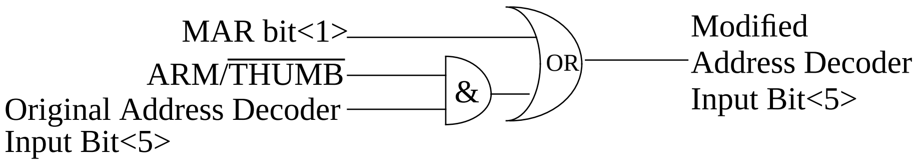

MAR bit<1>
ARM/$\overline{\text{THUMB}}$
Original Address Decoder Input Bit<5>
&
OR
Modified Address Decoder Input Bit<5>

Figure 5-6: Mode detection circuitry

The look-up operation in the BTC presents no problem in ARM mode, since only one hit may happen at any time either in the odd section or in the even section. In THUMB mode, only the odd section may have a hit when bit<1> of the MAR is high. If bit<1> of the MAR is low in THUMB mode, however, there are several possible cases: the BTC could miss, predict a hit in either section, or in both sections. For example, in THUMB mode, the source address 4 is stored in the even section and the source address 6 is put in the odd section. Since the CAM has 30 bits in a word, address 4 and 6 are stored as 4 in both cases. So when the instruction address 4 enters the CAM for the look-up operation, the CAM can have two hits, one in the odd section and one in the even section. In the case of two hits in THUMB mode with MAR bit<1> low, only the even section must be chosen. This mechanism is pictured in figure 5-7.

In the RAM, the condition code and the link bit are stored together with the target address. This makes it possible not to fetch the branch instruction itself. If the BTC predicts a hit, the target address goes to the PC and the condition code and the link bit are sent to the memory control block with the bypass signal. Thus there is no need to access the instruction memory to get the condition code and the link bit of the branch instruction. This is also expected to reduce system power since the frequency of memory access is decreased.

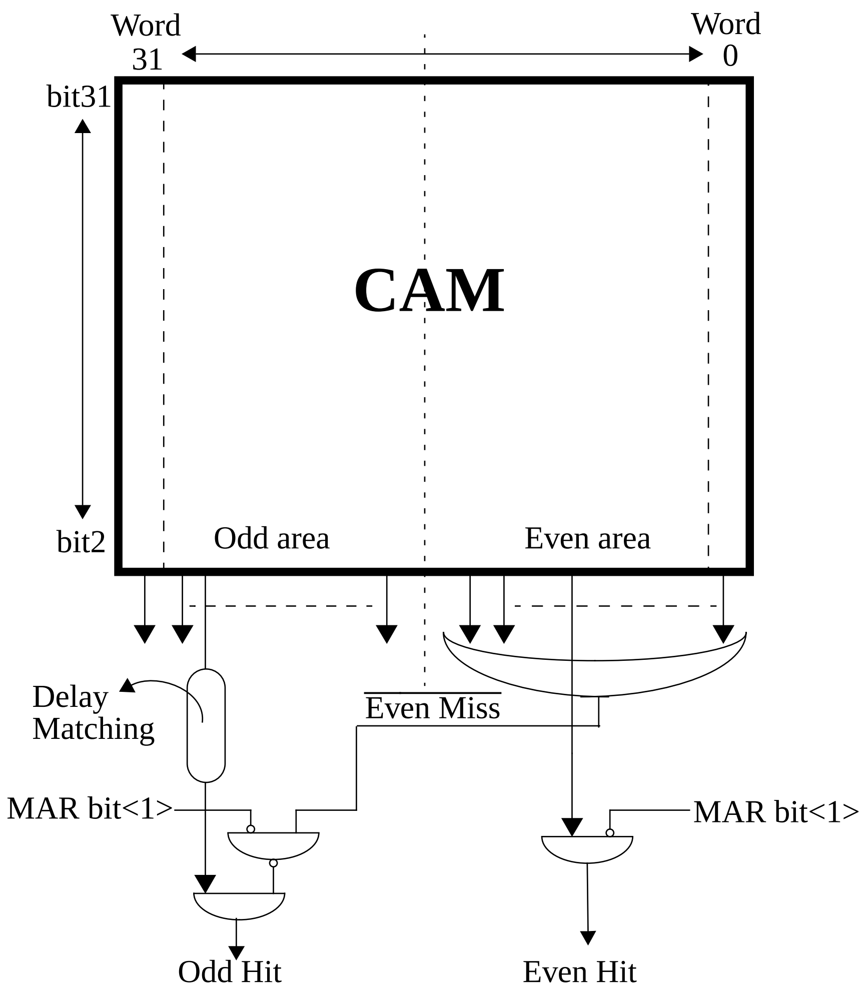


Figure 5-7: Odd and even hit determination mechanism

In AMULET3 the number of BTC entries is increased from 20 to 32. Figure 5-8 explains why the number of the entries are important for the performance of the BTC. For AMULET2e, two test programs, espresso and a C compiler, were used as the basis to determine the number of entries. Another program, dhrystone, was tested but not used since this program has an untypically long loop with many branches. In addition, the cost of implementing more entries cannot be disregarded in terms of layout area. However, if there is enough area to implement more entries, there is no reason not to implement more. We can get higher prediction accuracy in the case where a program has long loop behaviour such as dhrystone. As shown in figure 5-8, in the case of the dhrystone test, the

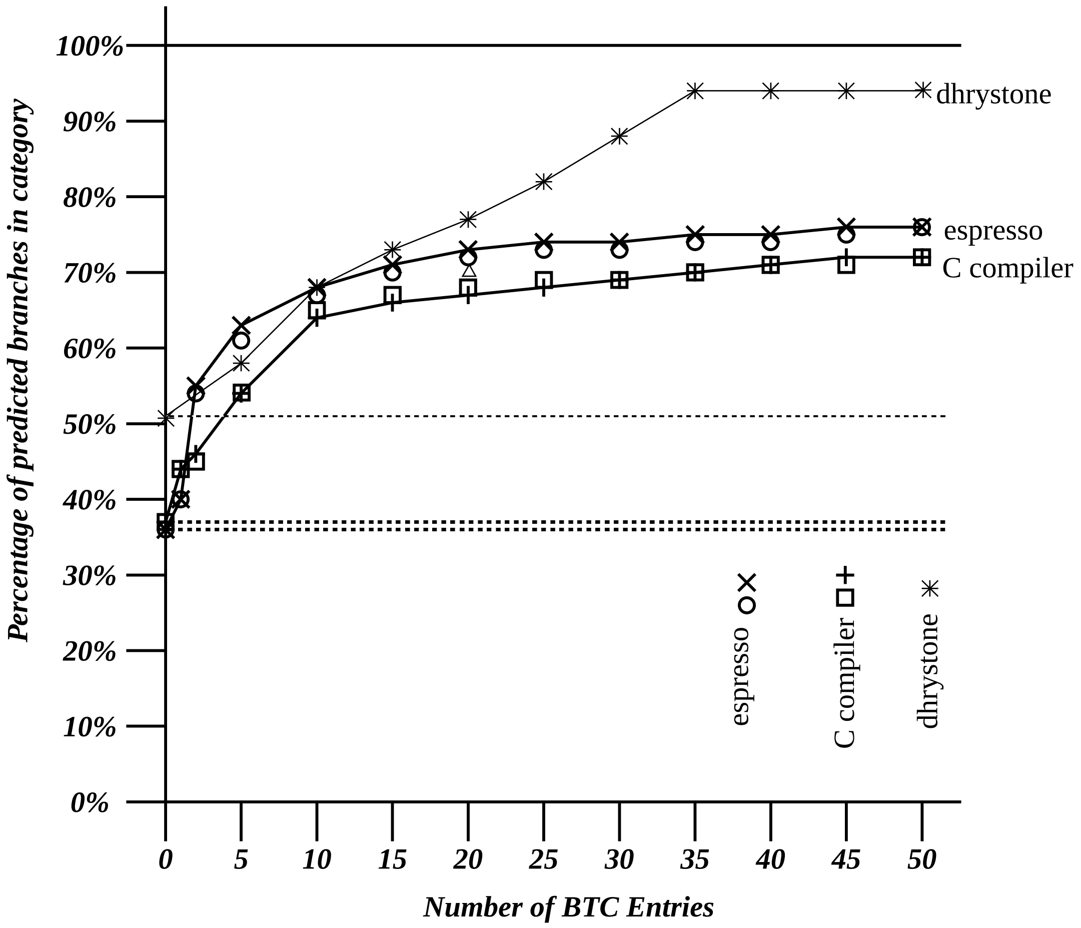


Figure 5-8: Effect of the BTC size on prediction rates (Courtesy of Jim Garside)

percentage of predicted branches increases by about 14%. (The increase are much smaller in the cases of espresso and the C compiler test.) Furthermore, five bits of address are already available for 20 entries, and implementing 32 entries increases no overhead in the control circuitry. Therefore the number of entries is increased to 32. This change is a result of an engineering trade-off - silicon area versus performance. The 32 entries may be reduced again later in the design process depending on the final AMULET3 layout size.

5.4 Summary

A similar branch prediction scheme is used for the BTCs of AMULET2e and AMULET3. The main improvements (compared to the AMULET2e BTC) of the AMULET3 BTC are as follows.

- In order to support THUMB mode, a novel BTC organisation was required. Two 16 bit THUMB instructions are fetched simultaneously from memory, and either or both of these could be branches that should be cached in the BTC. Handling double BTC hits and half-word predicted branch targets required significant changes to the AMULET2e BTC organisation.

- In order to avoid fetching predicted branches from memory, the condition code and link bit are stored in the RAM. This reduces power consumption and improves performance since the number of memory access is reduced.

- By increasing the number of entries, the percentage of correctly predicted branches is expected to be about 90% when a test program such as dhrystone is running. With a more accurate branch prediction scheme such as a hybrid predictor this figure could reach about 97% or 98% in typical programs [87][90] but only at the cost of substantial additional complexity. This is a field for future improvement.

As was mentioned earlier, depending on the requirements of the AMULET3 design, the detailed implementation of the BTC could be changed later. However, the branch prediction scheme will be maintained.

Implementation 6

6.1 Basic concepts

Several logic design techniques can be used in CMOS. However, all of them belong to one of the following logic disciplines [91][95][96]:

1. Static logic
2. Dynamic logic

Static logic is simple and straightforward. It is so-named since the information is permanently stored so long as the circuit is powered, and any gate output node is connected via a conducting transistor part to either Vdd or Vss.

Dynamic logic is based on the concept of precharging, which consists of pulling a gate output node up to Vdd or down to Vss either to charge or discharge the parasitic capacitance associated with that node. If the inputs of a gate generate the output value driven during precharge, no change in the output node occurs. Otherwise the node is strongly pulled down if precharge was high, or pulled up if precharge was low. During precharge, the precharge value is stored in parasitic capacitors, such as the gate capacitance, and disappears in a time of few hundreds of μs to a few ms after precharge, the actual time being a function of the temperature, the storage capacitance, and the leakage current, unless the node is recharged (or refreshed).

Static logic has a benefit in terms of power saving since there is no static power dissipation and no periodic recharge. It can be implemented easily since it is a ratioless logic.

Since the gate of each n-channel device is connected to the gate of the corresponding p-channel device, static logic has a bigger area and output capacitance than dynamic logic. Thus static logic suffers from low density and long gate delays. Furthermore, in submicron technology, area and speed are no longer independent variables. Larger areas lead to longer interconnections and therefore to lower speed.

Dynamic logic has the advantages of smaller area and faster speed over its static counterpart. Smaller area can be achieved since the logic uses the nMOS circuit of the static gate without the pMOS circuit, replacing the pMOS circuit with a single pMOS transistor for precharge. Faster speed can result from several factors when the precharge phase is not considered; firstly, the output of the dynamic gate drives a capacitance which is the sum of all the gate input capacitances of the n-channel (or p-channel) devices connected to it, whereas a static gate sees both p- and n-channel transistor capacitances. Secondly, the switching threshold of the gate depends on the switching threshold of the device itself (the device threshold voltage), rather than half of “Vdd-Vss” (the gate threshold voltage) for a balanced complementary static gate. Finally, the stray capacitances are larger in static logic design, since dynamic gates require less area than static gates do.

However, dynamic logic has some disadvantages also. Firstly, it has the problem of charge sharing or redistribution. Dynamic logic can work correctly only when the value of the sensing capacitor is much smaller than the value of the capacitor which stores information. Otherwise, it will fail to operate correctly. The charge will redistribute itself between the sensing capacitor and the output node capacitor. Secondly, the charge stored in the output node will leak away if there is no recharge operation after precharging the node. Thus when a power down mode is applied to dynamic logic, there must be a charge storage scheme on every output node of the dynamic logic. Finally, dynamic logic cannot be fully utilized. All dynamic logic uses precharging techniques that lower the availability of the circuit, since during precharge, the logic cannot be utilized.

6.2 Front-end implementation

Dynamic and static logic are used together in the branch target cache (BTC) of the instruction prefetch unit (IPU). Dynamic logic is used to give smaller layout and faster speed than static logic. However, dynamic logic needs careful design, since the designer must make sure that the circuit has the correct behaviour between the precharge and the evaluation period, and that there is no charge sharing problem.

The subsequent sections will show the reader how to implement the data and control path of the BTC in detail.

6.2.1 Data path circuit implementation

The datapath components in the BTC are divided into three sections: the Content Addressable Memory (CAM), the Random Access Memory (RAM) and the input latches as shown in appendix A.1.

The input latches are sets of transparent true single phase clock (TSPC) latches which store data when the enable signal EN is low as shown in appendix A.3. This transparent latch is open normally. This means a change on the data input is transferred to the output when the enable signal EN is high. After the enable signal EN goes low, a change on the data input cannot be transferred to the output and the output of the latch holds the value that was on the data input before the enable signal EN went low.

The CAM cell consists of a normal Static RAM cell with additional pass transistors P1 and P2 which form an XOR gate, and N1, which is a distributed NOR pull-down [92]. This is shown in figure 6-1 and in appendix A.9.

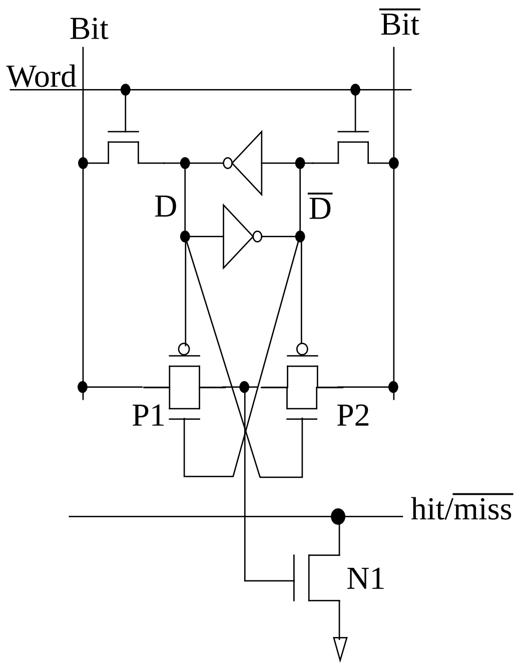


Figure 6-1: CAM cell 1

The write operation is simple. When the Word line is taken high, the information on the Bit line is stored in D via a pass transistor and the $\overline{\text{Bit}}$ line in $\overline{\text{D}}$. A look-up operation is performed to see whether the value stored in the RAM on D and $\overline{\text{D}}$ matches with the new input value on the Bit and $\overline{\text{Bit}}$ lines. During the precharge period the Bit and $\overline{\text{Bit}}$ lines are predischarged and the hit/$\overline{\text{miss}}$ line is precharged. Consider the case where D is high and the $\overline{\text{Bit}}$ line is also high during the evaluation period. In this case the $\overline{\text{Bit}}$ line pulls down the precharged hit/$\overline{\text{miss}}$ line to Vss by switching the N1 transistor on via P2. If the $\overline{\text{Bit}}$ line is low, the N1 transistor is kept off and no change happens on the precharged hit/$\overline{\text{miss}}$ line. The drains of the N1 transistors of all the cells in the same row are commoned as shown in figure 6-2. These form a distributed NOR gate using dynamic logic. Each

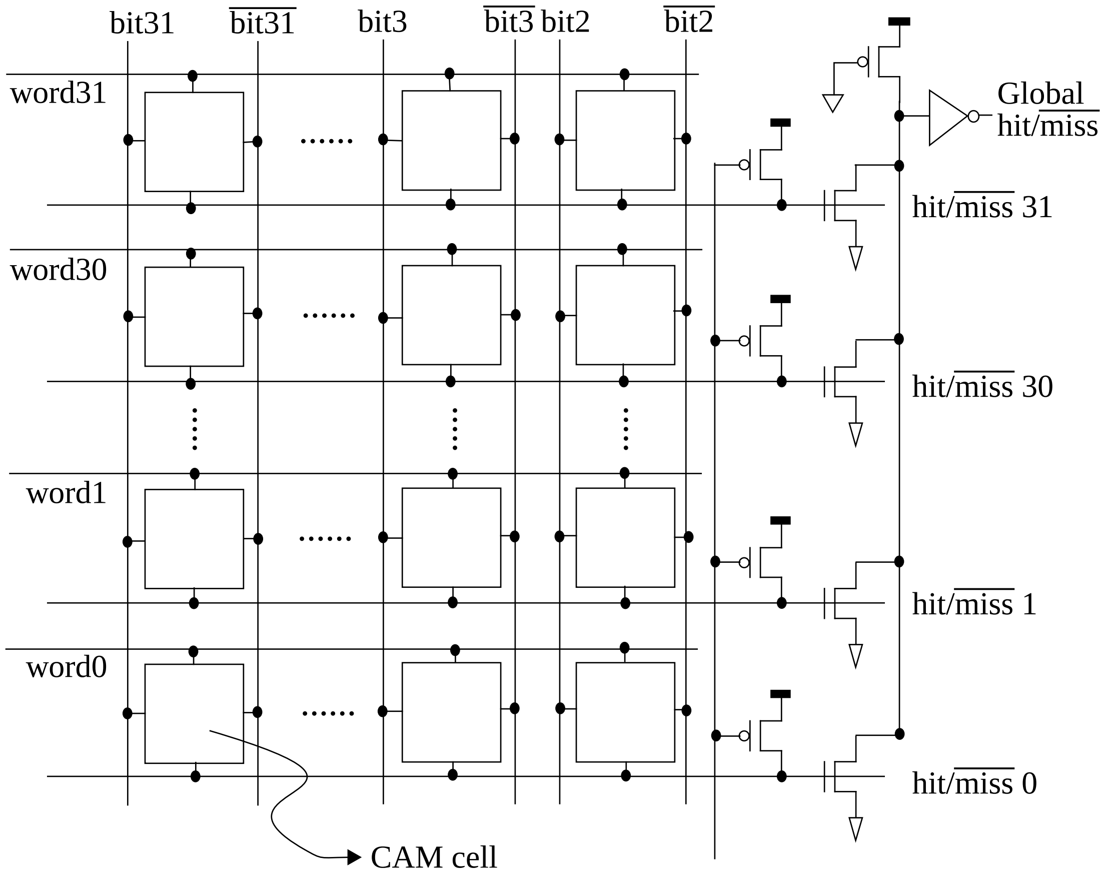

bit31
$\overline{\text{bit31}}$
bit3
$\overline{\text{bit3}}$
bit2
$\overline{\text{bit2}}$
word31
word30
word1
word0
Global hit/$\overline{\text{miss}}$
hit/$\overline{\text{miss}}$ 31
hit/$\overline{\text{miss}}$ 30
hit/$\overline{\text{miss}}$ 1
hit/$\overline{\text{miss}}$ 0
CAM cell

Figure 6-2: CAM cell arrays

hit/$\overline{\text{miss}}$ line remains high if all the cells in the same row have the same values of D and $\overline{\text{D}}$ as the values on each Bit and $\overline{\text{Bit}}$ line. This hit/$\overline{\text{miss}}$ line is used to select the RAM row to get the target address which goes to the PC. To indicate the overall hit/$\overline{\text{miss}}$ function, the hit/$\overline{\text{miss}}$ line of each row is used as an input signal in the global NOR gate as shown in figure 6-2. (This figure shows a simplified structure to illustrate the behaviour of the comparison function, not the exact implementation in the BTC.)

A different CAM cell structure as shown in figure 6-3 can be used [95]. This comprises a

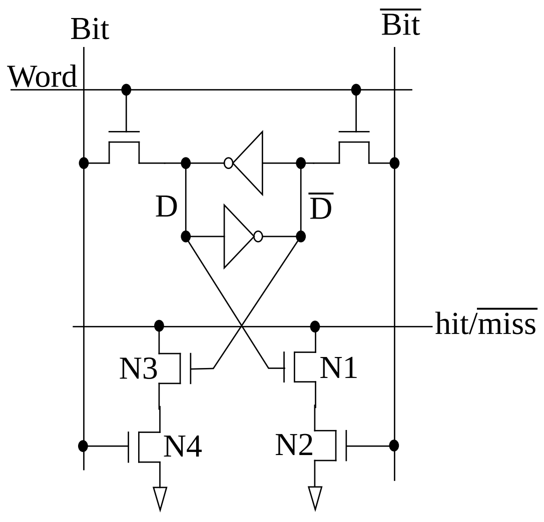


Figure 6-3: CAM cell 2

normal static RAM cell with additional transistor pairs: N1 + N2 and N3 + N4. This has an advantage over the previous CAM cell when a dummy bit cell is used for the self-timing completion detection function; when the hit/$\overline{\text{miss}}$ line remains high after comparison, there is the need to detect when the comparison has finished. In order to implement this function, a self-timed dummy bit cell is used as shown in figure 6-4. If

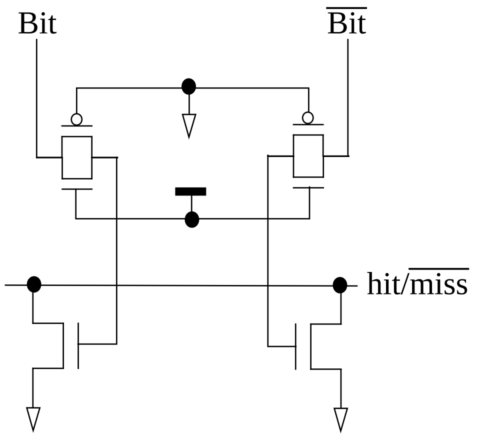


Figure 6-4: Dummy bit cell for CAM cell 1

the CAM cell in figure 6-3 is used, a more precise self-timed dummy bit cell can be implemented as shown in figure 6-5. This can be explained as follows. Bit and $\overline{\text{Bit}}$ lines

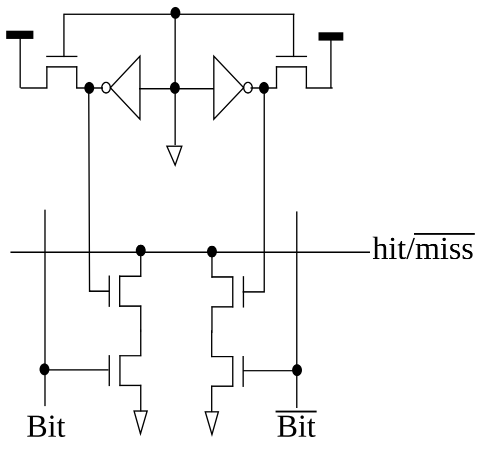


Figure 6-5: Dummy bit cell for CAM cell 2

in the first CAM cell are coupled through two pass transistors. Always, one of the two pass transistors is open and the other is closed. Thus the Bit or $\overline{\text{Bit}}$ line is connected to the pull down transistor and the closed pass transistor via the open pass transistor. But, in the self-timed dummy bit cell of figure 6-4 both pass transistors should be open together to mimic the hit operation in both the case when Bit is high and the case when $\overline{\text{Bit}}$ is high. So the Bit and $\overline{\text{Bit}}$ lines are not connected to the closed pass transistor as shown in figure 6-4. This completion detection circuit can produce a different result from the real circuit operation. If the second CAM cell is used, the self-timed dummy bit cell is made as shown in figure 6-5. This can mimic the real CAM cell timing more precisely. However, since there are serial nMOS transistors to pull down the hit/$\overline{\text{miss}}$ line, it might take a longer time to produce the result on that line. In order to escape from this disadvantage, wider nMOS transistor could be required and need more layout area.

The top schematic of the RAM is shown in appendix A.12 which has an address decoder and RAM register cells. The address decoder is simple and provides Write signals to the RAM register cells. The RAM cell is made by using a normal register cell as shown in figure 6-6 and appendix A.14. This register can be written to using one write enable line and one data bit line with minimum input capacitance.

The behaviour of the RAM cell is as follows. D is the input of the RAM cell and $\overline{\text{O}}$ is the output which is precharged high and stays high or is discharged low during the RAM read operation. When the Write signal goes high, N2 and N3 are open and the input D is stored in the internal node Q. When D is high, N1 is open and this helps Q go high quickly since $\overline{\text{Q}}$ is discharged via N3 and N1. During the read operation, the Read signal is high and $\overline{\text{O}}$ depends on the value of Q. When the value of Q is high, this turns on N4 and serially open N4 and N5 will discharge $\overline{\text{O}}$, which is already precharged during the precharge operation. When the value of Q is low, this switches off N4 and the precharged $\overline{\text{O}}$ is kept high. Since $\overline{\text{O}}$ shows the inverse value of Q during the read operation, it is written overlined. The precharge circuit for the outputs of the RAM register cells is depicted in appendix A.13. The top view of the RAM cell array is shown in figure 6-7. This comprises 32 words and each word has 31 bits.

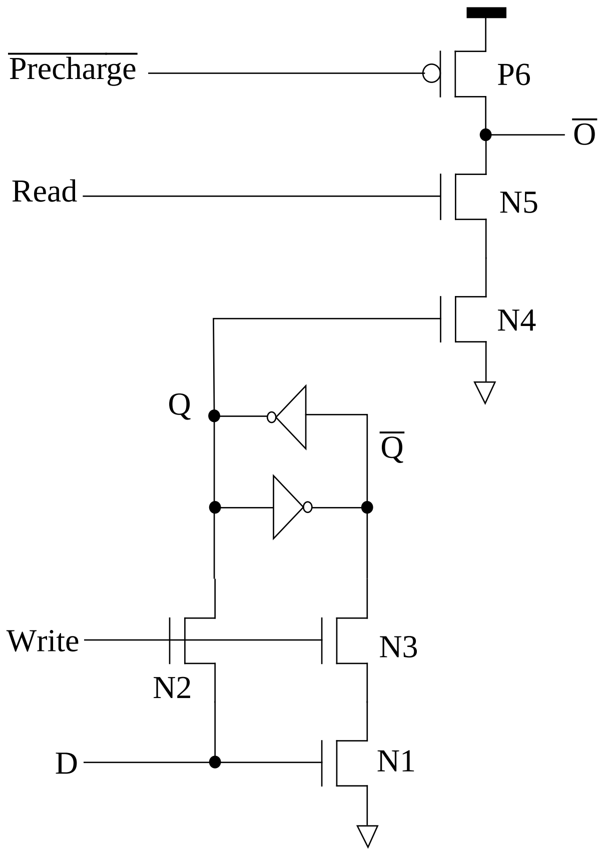


Figure 6-6: RAM register cell

6.2.2 Control path circuit implementation

The RAM control circuit consists of the decoder to issue write enable signals and dynamic logic to produce the output during a read operation. This is shown in figure 6-7. (The decoder is omitted since it is a simple address generator.)

The CAM, as was mentioned in chapter 5, is divided into two sections: odd and even. The hit detection logic for this feature is described in chapter 5.3 and is implemented with simple logic as shown in appendix A.10 and A.11. To reduce power consumption during the CAM comparison, the same row is divided into two sections: the high bits<31:6> and the low bits<5:2>. The behaviour of this function is explained in chapter 5. The circuit implementation of this function is illustrated in figure 6-8 and explained as follows.

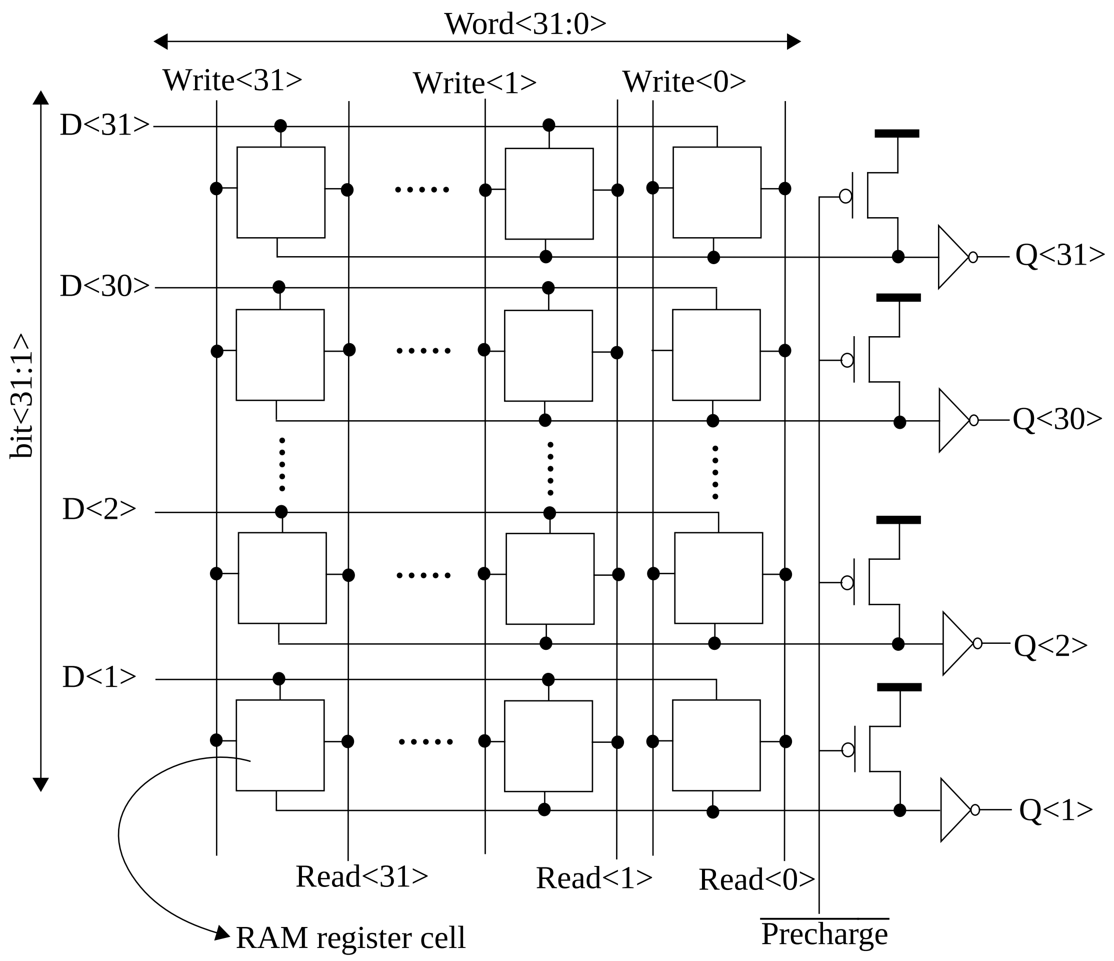


Figure 6-7: RAM cell arrays

The nWrite signal comes from the address decoder. If this signal is low, a write operation is activated and only when this is high can the read signal be invoked. During a write operation the hit/$\overline{\text{miss}}$ line is discharged. The hit/$\overline{\text{miss}}$ line is divided into two parts: HitH and HitL.

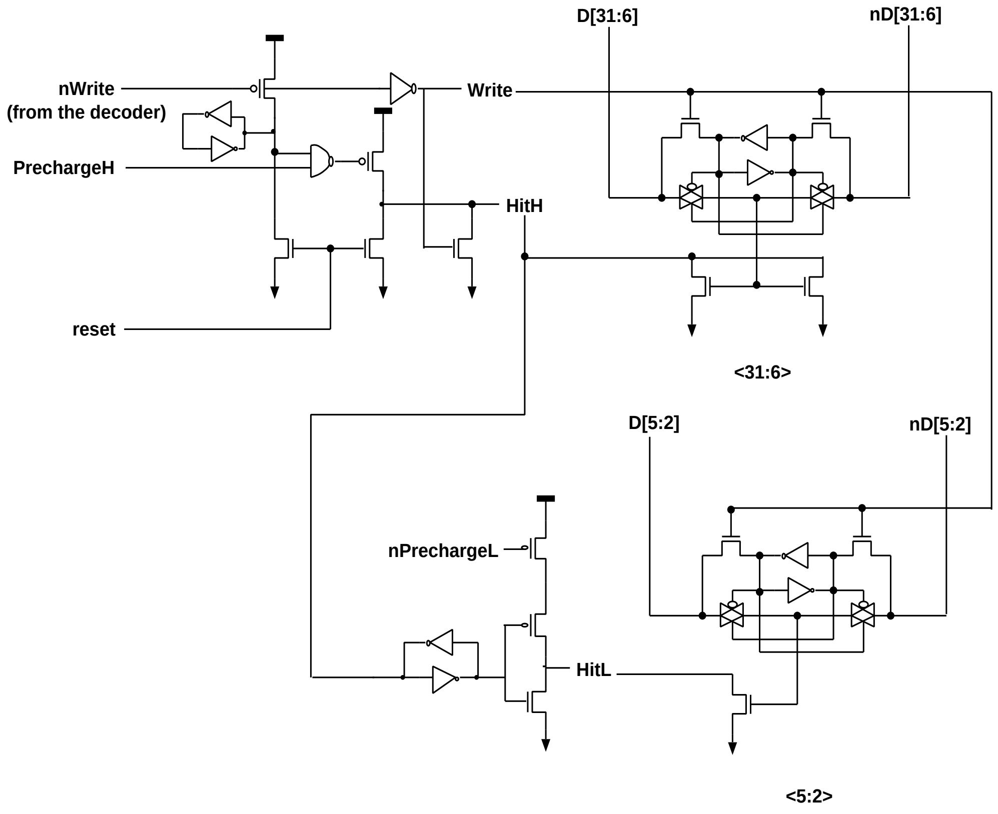


Figure 6-8: CAM control circuit for write and hit detection

Normally the HitL line is precharged every cycle for comparison. That is, the low bits are activated every cycle. When there is a matched address in the low bits, HitL stays high and this means that a hit has been detected. When the low bits do not match, HitL is discharged low.

There are four cases when the HitH line is precharged (when the PrechargeH signal is taken high and the CAM high bits are activated). Firstly, when the special HitH precharge signal is inserted from the outside, this line is precharged. This forces the high bits to be activated from the outside. Secondly, after new data is stored in the CAM, this line is precharged, since the sequential instruction address stream in the IPU is broken.

Thirdly, if there is a hit in the BTC, this line is precharged, since this also alters the sequential instruction address stream in the IPU. Finally, when the low bits<5:2> of the address are all 1s, this line is precharged, since the address is about to overflow into the higher bits. When there is no match in the high bits, the HitH signal is discharged low and this makes the HitL signal low also, whether or not there is a match in the low bits.

6.3 Back-end implementation

The BTC has been laid out using 0.35 micron triple metal CMOS technology.

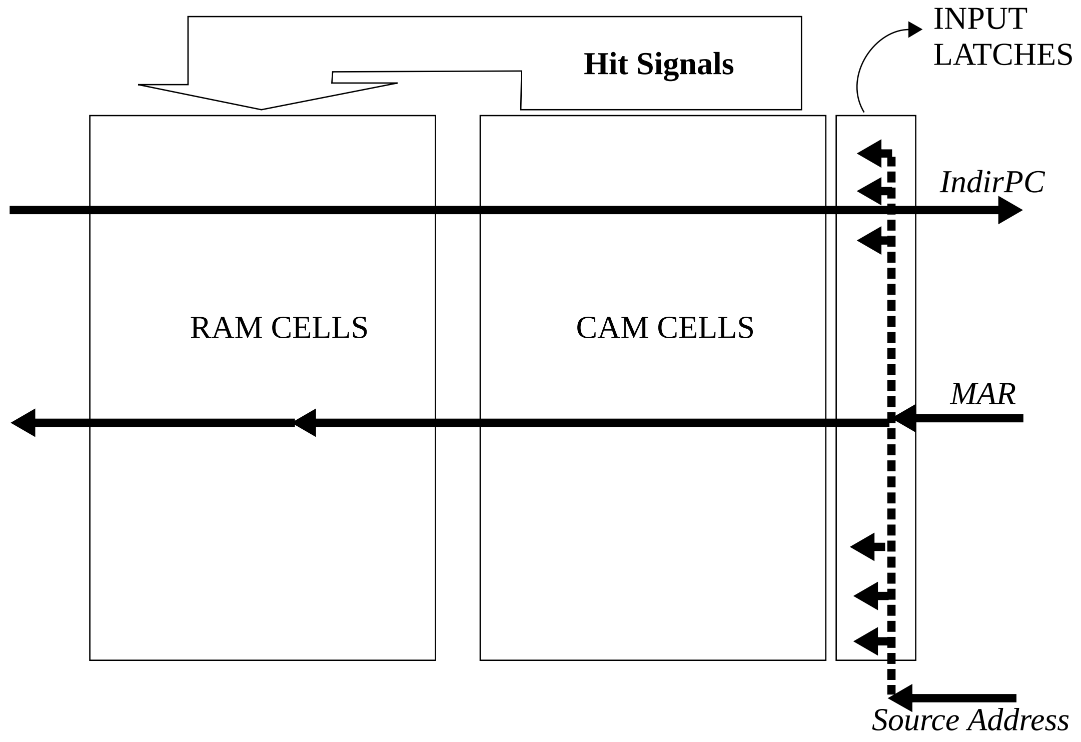


Figure 6-9: BTC layout diagram

As shown in figure 6-9, the RAM is located at the left side of the CAM. Three data buses called IndirPC, MAR, and Source Address (SA), pass across the BTC. The MAR and SA buses are directly connected to the CAM. The MAR bus is connected to the RAM also.

Each layout in appendix B matches a corresponding schematic in appendix A with the same name.

The RAM cell arrays are straightforward as shown in figure 6-7. The CAM cell layout diagram for one word is depicted in figure 6-11. A more detailed explanation is given in the next section.

6.4 Evaluation

The CAM and RAM have been simulated using HSPICE operating at typical-case conditions (Vdd = 3.3V, Vss = 0.1V, typical-typical process corner, at 100 ^{o}C). The simulation results are shown in table 6-1.

The critical path in the CAM for the look-up operation lies in the comparison circuit, from the Read signal to the hit/$\overline{\text{miss}}$ signal when only one mismatch happens in the 30 CAM cells, shown as bit<31> of word<31> in the upper drawing of figure 6-10. This is explained as follows. The RD signal activates the DRIVER cell and this invokes the look-up operation. The worst case happens at bit<31> of word<31>, since this cell is farthest from the DRIVER and from the hit/$\overline{\text{miss}}$ signal at the right side of the HITL cell. The data write time from the write signal to data loading in the CAM has been simulated also. The read operation of the RAM has been simulated from the read signal to the data out line which is shown in the lower drawing of figure 6-10.

The write time simulation of the CAM was performed as measuring the time from the WR signal to the D and DN changes of a CAM bit cell located far away from the driver, which is word<31> in the upper drawing of figure 6-10. In order to measure the look-up

Table 6-1: HSPICE Simulation result

Simulation Path	Result (ns)
WR ↑ -> D ↑	0.51
WR ↑ -> DN ↑	0.38
WR ↑ -> D ↓	0.43
WR ↑ -> DN ↓	0.43
RD ↑ -> HITH ↓	0.75
RD ↑ -> HITL ↓	1.06
RD ↑ -> Data out ↓	0.46

operation time, the most significant bit cell, which is bit<31> of word<31> in the upper drawing of figure 6-10, is given a different value from other bit cells which are from bit<30> to bit<2> in the same picture. Then the time from the RD signal rising to HITH falling and HITL falling are measured. (As was explained earlier, the hit/$\overline{\text{miss}}$ signal is at the right side of the HITL signal.) The different rise time for D and DN is due to different loading depending on whether the pass transistor in the CAM cell is open. The read simulation of the RAM used the same method as the CAM. After writing a different value from the rest of the bit cells, the time from the RD signal rising to Data out falling is measured. The test circuit is shown in figure 6-10.

The silicon layout diagram of a CAM cell array is drawn in figure 6-11. The Precharge Low cell matches with the HITH cell in the upper drawing of figure 6-10 and the Hit Detection cell is identical with the HITL cell in the upper drawing of figure 6-10. The arrow named Hit Signal to RAM means the Hit/$\overline{\text{miss}}$ signal in the upper drawing of figure 6-10. As depicted in figure 6-9, the RAM cell arrays are located at the left side of the CAM cell arrays in figure 6-11 (though this is not drawn in the figure). VDD and VSS stand for Power and Ground respectively.

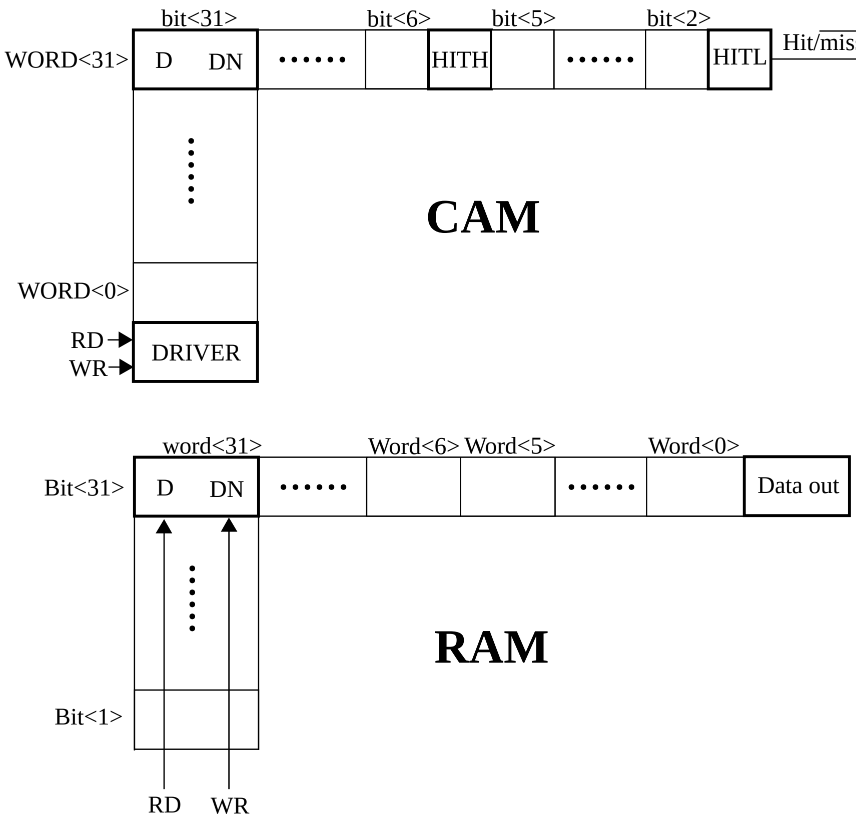


Figure 6-10: Test circuit for simulation

6.5 Summary

The design of the BTC for the AMULET3 processor has been carried out using a self-timed technique. The majority of the design is purely static, composed of complementary CMOS gates. In certain situations, wide NOR functions are required and these are implemented in dynamic logic. To indicate when the read operation of the CAM and the

RAM is finished, a self-timed technique is used. That is, an extra self-timing column of dummy bit cells with a dynamic bit line is implemented to mimic the timing of the data bit lines.

As was mentioned in section 5.3, the new THUMB function is added in AMULET3. In terms of functionality, this is the biggest change in the design of the AMULET3 BTC compared to that of AMULET2e. The layout for the storage for the condition code and the link bit was added and of course the layout geometry is totally renewed as shown in figures 6-9, 6-10, and 6-11.

Depending on the progress in designing the rest of the AMULET3 processor, the detailed implementation of the BTC may change. However, the major look-up and write operations of the CAM cell arrays and the read and write operation of the RAM cell arrays are unlikely to be changed.

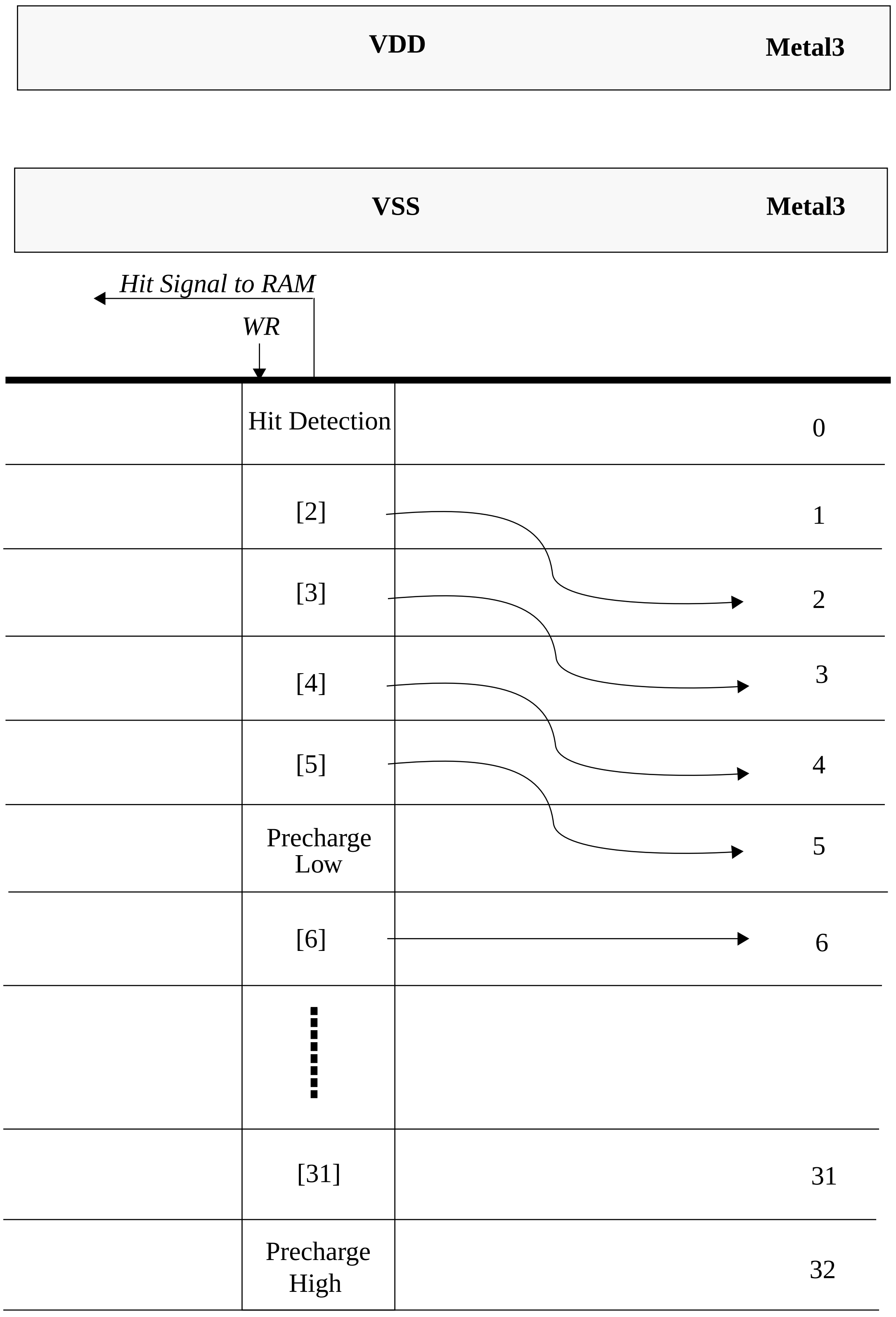


Figure 6-11: CAM cell layout diagram for one word

Conclusion 7

This thesis has presented engineering work on asynchronous design. The branch target cache (BTC) was designed and implemented for the instruction prefetch unit (IPU) in the AMULET3 processor. As was mentioned in chapter 1, the author is responsible for designing the IPU, and the BTC is a part of the IPU. The BTC consists of the content addressable memory (CAM) and the random access memory (RAM). These two components detect completion using dummy bit self-timed logic. The CAM comprises the CAM cell arrays and the hit/$\overline{\text{miss}}$ detection logic which is the critical path for the performance of the BTC. The core of the work is mainly engineering which focuses on implementing low level transistor circuitry. Many asynchronous design techniques were used in the course of the work.

7.1 Contributions

The design of the BTC in AMULET3 has shown that it is possible to achieve better performance and more functionality using a new configuration, although the design is similar to the one used before in AMULET2e. As shown in figure 5-8, in the case of the dhrystone test, the percentage of predicted branches increases by about 14% compared to that of the AMULET2e BTC. (The increases are much smaller in the cases of espresso and the C compiler test.) As shown in chapter 5.3, new functionality to support THUMB mode has been added, and the condition code and the link bit of a branch address are now stored in the BTC RAM together with the target address, in order to avoid fetching predicted branches from memory.

AMULET3 is an ongoing project, and so is the BTC design. Therefore the hardware implementation, proposed in this thesis, could be changed later. For the same reason, the exact numerical values of the power consumption, total speed, and total layout area of the BTC are not available since they depend on the rest of AMULET3. Nevertheless, since the CAM and RAM cell designs are finished, the speed of the CAM block, which is the critical path in the BTC and the major factor of the BTC access time, can be ascertained and is given in this thesis.

Methods for implementing the BTC using static and dynamic logic have been described in detail. Although a similar BTC was used in AMULET2e, there are three distinct improvements implemented in AMULET3.

- In order to support the THUMB instruction set, the functionality of the THUMB mode is added. Thus from the viewpoint of the BTC the THUMB and the ARM instruction sets are equally supported.

- The condition code and link bits in the branch instruction are stored in the RAM, so there is no need to fetch the instruction for a predicted branch. This saves power and increases performance as it saves a memory access.

- The number of the CAM entries is increased from 20 to 32. Thus the total performance is increased.

Post-layout simulation, in a 0.35 micron triple metal CMOS technology, shows that the comparison function of the CAM takes 1.06ns to produce the hit/$\overline{\text{miss}}$ signal when a full 30 bit comparison is performed. The higher bit<32:6> comparison takes 0.75ns. Taking account of the fact that usually only the lower bit<5:2> comparison is performed, just 0.51ns is taken for the CAM comparison in the most frequent case.

In addition, AMULET3 has a much shorter average cycle time than AMULET2e. Some of this is achieved through using a more advanced process technology, but the rest (about a further factor 2) has required a radical redesign of the IPU organisation in order to ensure that the IPU is not a major bottleneck in the design.

7.2 Future work

Historically, asynchronous design has been considered to have potential advantages in the implementation of designs with low power consumption. As described in chapter 5, asynchronous design has been shown to have potential for low power consumption, as evidenced by AMULET2e. It is natural to think AMULET3 will also have good low-power characteristics in the light of past experience.

There is another issue related to asynchronous design, electro-magnetic interference. This emerging issue is considered as one of the most important features in asynchronous design. In synchronous design, clock speeds have already reached 500 MHz, and gigahertz processors will probably be available within the next few years. At those clock rates, even short transmission lines will act as antennas, producing unwelcome amounts of electro-magnetic interference and cross-talk [97]. The fundamental property of the periodic operation defined by the clock worsens this problem. However, this effect does not appear in asynchronous design since all signal changes are aperiodic. To make matters worse, in synchronous design logic activity happens immediately following the clock edge, whereas in asynchronous design it is distributed over time. Thus in asynchronous design the noise spectrum is spread without the high amplitude peaks which are found in the spectrum of synchronous designs. This rigorous EMI compliance will be proven in AMULET3.

Bibliography

[1] G.D. Hutcheson and J. D. Hutcheson, “Technology and Economics in the Semiconductor Industry”, Scientific American The Solid-State Century, Special Issue Volume 8, November 1, 1997.

[2] W. I. Fletcher, “Why Asynchronous Circuits?”, Engineering Approach to Digital Design, chap 10-2, Prenctice-Hall, Inc., 1980.

[3] S. Hauck, “Asynchronous Design Methodologies: An Overview”, Proceedings of the IEEE, Vol. 83, No. 1, pp. 69-93, January 1995.

[4] D. A. Huffman, “The synthesis of sequential switching circuits”, J. Franklin Institute, March/April 1954.

[5] S. H. Unger, “Asynchronous Sequential Switching Circuits”, Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[6] D. Dobberpuhl et al, “A 200-MHz 64-b Dual-Issue CMOS Microprocessor”, IEEE Journal of Solid-State Circuits, vol. 27, no. 11, November 1992.

[7] D. Draper et al, “Circuit Techniques in a 266-MHz MMX-Enabled Processor”, IEEE Journal of Solid-State Circuits, vol. 32, no. 11, November 1997.

[8] D. Manners, “Portable Prompt Low-Power Chips”, Electronics Weekly, no. 1574, pp. 22, November 13, 1991.

[9] D. Maliniak, “Better Batteries for Low-Power Jobs”, Electronic Design, vol. 40, no. 15, pp. 18, July 23, 1992.

[10] J. Mello and P. Wayner, “Wireless Mobile Communications”, Byte, vol. 18, no. 2, pp. 146-153, February 1993.

[11] The AMULET Group, “About AMULET”, http://www.cs.man.ac.uk/amulet.

[12] S. B. Furber et al, “AMULET1: A Micropipelined ARM”, Proceedings of CompCon’94, pp. 476-489, IEEE Computer Society Press, CompCon’94, San Francisco, March 1994.

[13] S. B. Furber et al, “AMULET2e: An Asynchronous Embedded Controller”, Proceedings Async ‘97, pp. 290-299, IEEE Computer Society Press, April 1997.

[14] R. York, “Branch Prediction Strategies for Low Power Microprocessor Design”, MSc thesis, Department of Computer Science, The University of Manchester, 1994.

[15] J. L. Turley, “Thumb Squeezes ARM Code Size”, Microprocessor Report, vol. 9, no. 4, March 1995.

[16] Ad M. G. Peeters, “Single-Rail Handshake Circuits”, Proefschrift Technische Universiteit Eindhoven, The Netherlands, 1996.

[17] S. B. Furber and P. Day, “Four-Phase Micropipeline Latch Control Circuits”, IEEE Transactions on VLSI Systems, vol. 4, no. 2, pp. 247-253, June 1996.

[18] K. Y. Yun, P. A. Beerel, and J. Arceo, “High-Performance Two-Phase Micropipeline Building Blocks: Double Edge-Triggered Latches and Burst-Mode Select and Toggle Circuits”, IEE Proceedings-Circuits, Devices and Systems, pp. 282-288, vol. 143, no. 5, October 1996.

[19] N. C. Paver, “The Design and Implementation of an Asynchronous Microprocessor”, PhD Thesis, Dept. of Computer Science, University of Manchester, 1994.

[20] C. E. Molnar, T. -P. Fang, and F. U. Rosenberger, “Synthesis of Delay-insensitive Modules”, In Henry Fuchs, editor, 1985 Chapel Hill Conference on Very Large Scale Integration, pp. 67-86, Computer Science Press, Inc., 1985.

[21] J. Sparso and J. Staunstrup, “Delay-insensitive Multi-ring Structures”, Integration, the VLSI journal, vol. 15, pp. 313-340, 1993.

[22] J. Udding, “Classification and Composition of Delay-insensitive Circuits”, PhD thesis, Technische Universiteit Eindhoven, 1984.

[23] A. J. Martin, “The Limitation to Delay-insensitive in Asynchronous Circuits”, 6th MIT Conference on Advanced Research in VLSI, MIT Press, Cambridge, Massachusetts, 1990.

[24] K. van Berkel, “Beware the Isochronic Fork”, Integration, the VLSI journal, vol. 13, pp. 103-128, 1992.

[25] R. Manohar and A. J. Martin, “Quasi-delay-insensitive Circuits are Turing-complete”, Technical Report, Caltech-CS-TR-95-21, Department of Computer Science, California Institute of Technology, 1995.

[26] S. B. Furber, “Fundamental Concepts: part 2”, Lecture Notes, Summer School on Asynchronous Circuit Design, Technical University of Denmark, August 18-22, 1997.

[27] A. J. Martin, “Tomorrow’s Digital Hardware will be Asynchronous and Verified”, Technical Report, Caltech-CS-TR-93-26, Department of Computer Science, California Institute of Technology, 1993.

[28] S. B. Furber, “Breaking Step - the Return of Asynchronous Logic”, Keynote talk - IEE Colloquium on the Design and Test of Asynchronous System, Savoy Place, London, 28 February 1996, pp. 1/1 - 1/4.

[29] D. E. Muller and W. C. Bartky, “A Theory of Asynchronous Circuits”, Kluwer Academic Publishers, 1993.

[30] Kishinevsk et al, “Concurrent Hardware: the Theory and Practice of Self-timed Design”, John Wiley & Sons, Inc., New York, 1994.

[31] R. M. Keller, “Towards a Theory of Universal Speed-independent Modules”, IEEE Transactions on Computers, vol. C-23, no. 1, pp. 21-33, January 1974.

[32] S. B. Furber, “Experiences with Petrify”, In Proceedings of the Third UK Forum on Asynchronous System, Department of Computer Science, The University of Edinburgh, Edinburgh, Scotland, U.K., December 15-16, 1997.

[33] W. A. Clark, “Macromodular Computer Systems”, In AFIPS Conference Proceedings: 1967 Spring Joint Computer Conference, vol. 30, pp. 335-336, Atlantic City, NJ, Academic Press, 1967.

[34] C. L. Seitz, “System Timing”, In C. A. Mead and L. A. Conway, editors, Introduction to VLSI Systems, Addison-Wesley, 1980.

[35] T. Verhoff, “Delay-insensitive codes - an Overview”, Distributed Computing, vol. 3, pp. 1-8, 1988.

[36] I. E. Sutherland, “Micropipelines”, The 1998 Turing Award Lecture, Communications of the ACM, vol. 32, pp. 720-738, June 1988.

[37] A. J. Martin, “Synthesis of Asynchronous VLSI Circuits”, Course Notes, VLSI~91, Edinburgh, August 1991.

[38] S. M. Burns, “Performance Analysis and Optimization of Asynchronous Circuits”, PhD thesis, California Institute of Technology, December 1990.

[39] K. van Berkel, “Handshake Circuits: an Asynchronous Architecture for VLSI Programming, vol. 5 of International Series on Parallel Computation. Cambridge University Press, 1993.

[40] E. Bruvand, “Designing Self-timed System using Concurrent Programs”, Journal of VLSI Signal Processing, 7(1/2): 47-59, February, 1994.

[41] Jo C. Ebergen, J. Segers, and I. Benko, “Parallel Programs and Asynchronous Circuit Design”, In Graham Britwistle and Al Davis, editors, Asynchronous Digital Circuit Design, Workshops in Computing, pp. 51-103, Springer-Verlag, 1995.

[42] A. Bardsley, “Balsa: An Asynchronous Circuit Synthesis System”, MPhil thesis, Department of Computer Science, The University of Manchester, 1997.

[43] W. S. Coates, A. L. Davis, and K. S. Stevens, “The Post Office Experience: Designing a Large Asynchronous Chip”, Integration, the VLSI Journal, 15(4): 341-366, 1993.

[44] S. M. Nowick and D. L. Dill, “Synthesis of Asynchronous State Machines using a Local Clock”, In Proceedings of the 1991 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 192-197, IEEE Computer Society Press, October 1991.

[45] S. M. Nowick, “Automatic Synthesis of Burst-mode Asynchronous Controllers”, PhD thesis, Stanford University, 1993.

[46] K. Y. Yun, “Synthesis of Asynchronous Controllers for Heterogeneous Systems”, PhD thesis, Stanford University, August 1994.

[47] P. A. Beerel, K. Y. Yun, and W. C. Chou, “Optimizing Average-case Delay in Technology Mapping of Burst-mode Circuits”, In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, March 1996.

[48] S. Chakraborty et al, “Timing Analysis for Extended Burst-Mode Circuits”, Third International Symposium on Advanced research in Asynchronous Circuits and Systems, IEEE Computer Society Press, April 1997.

[49] J. L. Peterson, “Petri Net Theory and the Modelling of Systems”, Prentice-Hall, Inc., N. J., 1981.

[50] R. Y. Rosenblum and A. V. Yakovlev, “Signal Graphs: from Self-timed to Timed ones”, In Proceedings of International Workshop on Timed Petri Nets, IEEE Computer Society Press, pp. 199-207, Torino, Italy, July 1985.

[51] T. -A. Chu, “On the Models for Designing VLSI Asynchronous Digital Circuits”, Integration, the VLSI Journal, 4(2): 99-113, June 1986.

[52] T. -A. Chu, “Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications”, PhD thesis, MIT, June 1987.

[53] A. V. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, “A Unified Signal Transition Graph Model for Asynchronous Control

Circuit Synthesis”, Formal Methods in System Design, 9: 139-188, 1996.

[54] L. Lavagno, “Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs”, PhD thesis, Department of Electrical Engineering and Computer Science, The University of California at Berkeley, 1992.

[55] P. Vanbekbergen et al, “Optimized Synthesis of Asynchronous Control Circuits from Graph-theoretic Specifications”, IEEE Transactions on Computer-Aided Design, 11(11): 1426-1438, November 1992.

[56] L. Lavagno and A. Sangiovanni-Vincentelli, “Algorithms for Synthesis and Testing of Asynchronous Circuits”, Kluwer Academic Publishers, 1993.

[57] M. A. Kishinevsky, A. Y. Kondratyev, and A. R. Taubin, “Specification and Analysis of Self-timed Circuits”, Journal of VLSI Signal Processing, 7(1/2): 117-135, February 1994.

[58] J. Cortadella et al, “Deriving Petri Nets from Finite Transition Systems”, Technical Report UPC-DAC-1996-19, Department of Computer Architecture, Universitat Politecnica de Catalunya, June 1996.

[59] J. Cortadella et al, “Technology Mapping of Speed-independent Circuits Based on Combinational Decomposition and Resynthesis”, In Proc. European Design and Test Conference, 1997.

[60] J. Cortadella et al, “Tutorial: Synthesis of Control Circuits from STG Specifications”, Lecture Notes, Summer School on Asynchronous Circuit Design, Technical University of Denmark, August 18-22, 1997.

[61] J. Cortadella et al, “Petrify User’s Manual”, VLSI Design Group in the Department of Computer Science, The University of Newcastle upon Tyne, June 1997.

[62] J. Cortadella et al, “Petrify: a Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers”, IEICE Transactions on Information and Systems, E80-D(3): 315-325, 1997.

[63] J. Cortadella et al, “Complete State Encoding Based on the Theory of Regions”, In International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 36-47, March 1996.

[64] A. J. Martin et al, “The First Asynchronous Microprocessor: the Test Results”, Computer Architecture News, 17(4): 95-110, June 1989.

[65] A. J. Martin et al, “A 100-MIPS GaAs Asynchronous Microprocessor”, IEEE Design and Test of Computers, 11(2): 43-49, 1994.

[66] M. E. Dean, “STRiP: A Self-Timed RISC Processor Architecture”, PhD thesis, Stanford University, 1992.

[67] T. Nanya et al, “TITAC: Design of a Quasi-delay-insensitive Microprocessor”, IEEE Design & Test of Computers, 11(2): 50-63, 1994.

[68] T. Nanya et al, “TITAC-2: A 32-bit Scalable-Delay-Insensitive Microprocessor”, HOT Chips IX, Stanford, pp. 19-32, August 1997.

[69] E. Bruvand, “The NSR Processor”, In Proceedings of the 26th Annual Hawaii International Conference on System Science, pp. 428-435, Maui, Hawaii, 1993.

[70] W. F. Richardson and E. Brunvand, “Fred: An Architecture for a Self-Timed Decoupled Computer”, Technical Report UUCS-95-008, The University of Utah, 1995.

[71] R. F. Sproull, I. E. Sutherland, and C. E. Molnar, “Counterflow Pipeline Processor Architecture”, IEEE Design and Test of Computers, vol. 11, no. 3, 1994.

[72] S. V. Morton, S. S. Appleton, and M. J. Liebelt, “ECSTAC: A Fast Asynchronous Microprocessor”, In Proceedings of the Second Working Conference on Asynchronous Design Methodologies, pp. 180-189, London, U.K., 1995.

[73] The AMULET Group, “The AMULET3 Microprocessor”, http://www.cs.man.ac.uk/amulet/AMULET3_uP.html.

[74] D. Jagger, “ARM Architecture Reference Manual”, Prentice-Hall, July 1996.

[75] C. Svensson and D. Liu, “Low Power Circuit Techniques”, In J. M. Rabaey and M. Pedram, editors, Low Power Design Methodologies, Kluwer Academic Publishers, 1996.

[76] D. A. Gilbert, “Dependency and Exception Handling in an Asynchronous Microprocessor”, PhD thesis, Department of Computer Science, The University of Manchester, 1997.

[77] J. L. Henessy and D. A. Patterson, “Computer Architecture: A Quantitative Approach”, Morgan Kaufman Publishers, Inc., San Francisco, California, Second Edition, 1996.

[78] R. L. Sites, “Alpha Architecture Reference Manual”, Digital Press, Burlington, MA, 1992.

[79] G. Kane and J. Heinrich, “MIPS RISC Architecture”, Prentice Hall, 1992.

[80] P. Chow and M. Horowitz, “Architecture Tradeoffs in the Design of MIPS-X”, In Proceedings of the 14th Annual International Symposium on Computer Architecture, June 1987.

[81] C. Melear, “The Design of the 8800 RISC Family”, IEEE Micro, pp. 26-38, April 1989.

[82] J. E. Smith, “A Study of Branch Prediction Strategies”, In Proceedings of the 8th International Symposium on Computer Architecture, pp. 135-148, May 1981.

[83] Ed. C. May et al, “The PowerPC Architecture: A Specification for a New Family of RISC Processors”, Morgan Kaufman Publishers, Inc., San Francisco, CA, 1994.

[84] J. A. Ficher and S. M. Freudenberger, “Predicting Conditional Branch Predictions from Previous Runs of a Program”, 5th International Conference on Architectural Support for Programming Languages and Operating Systems, 1992.

[85] T. -Y. Yeh and Y. N. Patt, “Two-level Adaptive Branch Prediction”, 24th ACM/IEEE International Symposium on Microarchitecture, November 1991.

[86] S. T. Pan, K. So, and J. T. Rahmeh, “Improving the Accuracy of Dynamic Branch Prediction using Branch Correlation”, In Proceedings of ASPLOSV, pp. 76-84, Boston, MA, October 1992.

[87] S. McFarling, “Combining Branch Predictors”, WRL Technical Note TN-36, Digital Western Research Laboratory, Palo Alto, CA, June 1993.

[88] L. Gwennap, “Gshare, “Agrees” Aid Branch Prediction”, Microprocessor Report, November 17, 1997.

[89] P. Chang and U. Banerjee, “Profile-guided Multiheuristic Branch Prediction”, In Proceeding of the International Conference on Parallel Processing, July 1995.

[90] M. Evers, P. -Y. Chung, and Y. N. Patt, “Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches”, The 23rd Annual International Symposium on Computer Architecture, May 1996.

[91] J. Yuan and C. Svensson, “High-Speed CMOS circuit techniques”, IEEE Journal of Solid-State Circuits, vol. 24, pp. 723-726, February 1989.

[92] N. Weste and K. Eshrachian, “Principles of CMOS VLSI Design: A System Perspective”, Second Edition, Addison-Wesley Publishing Company, 1994.

[93] J. Yuan and C. Svensson, “New Single-Clock CMOS Latches and Flipflops with Improved Speed and Power Savings”, IEEE Journal of Solid-State Circuits, vol. 32, no. 1, pp. 62-69, January 1997.

[94] J. Liu, “Arithmetic and Control Components for an Asynchronous System”, PhD thesis, Department of Computer Science, The University of Manchester, 1998.

[95] A. Bellaouar and M. I. Elmsry, “Low-Power Digital VLSI Design: Circuits and Systems”, Kluwer Academic Publishers, 1995.

[96] M. Annaratone, “Digital CMOS Circuit Design”, Kluwer Academic Publishers, 1986

[97] M. S. Mirotznik and D. Prather, “How to choose EM software”, In IEEE Spectrum Magazine, December 1997

[98] T. Ono-Tesfaye, C. Kern, M. Greenstreet, “Verifying a Self-Timed Divider”, Proc. International Workshop Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 146-158, IEEE Computer Society Press.

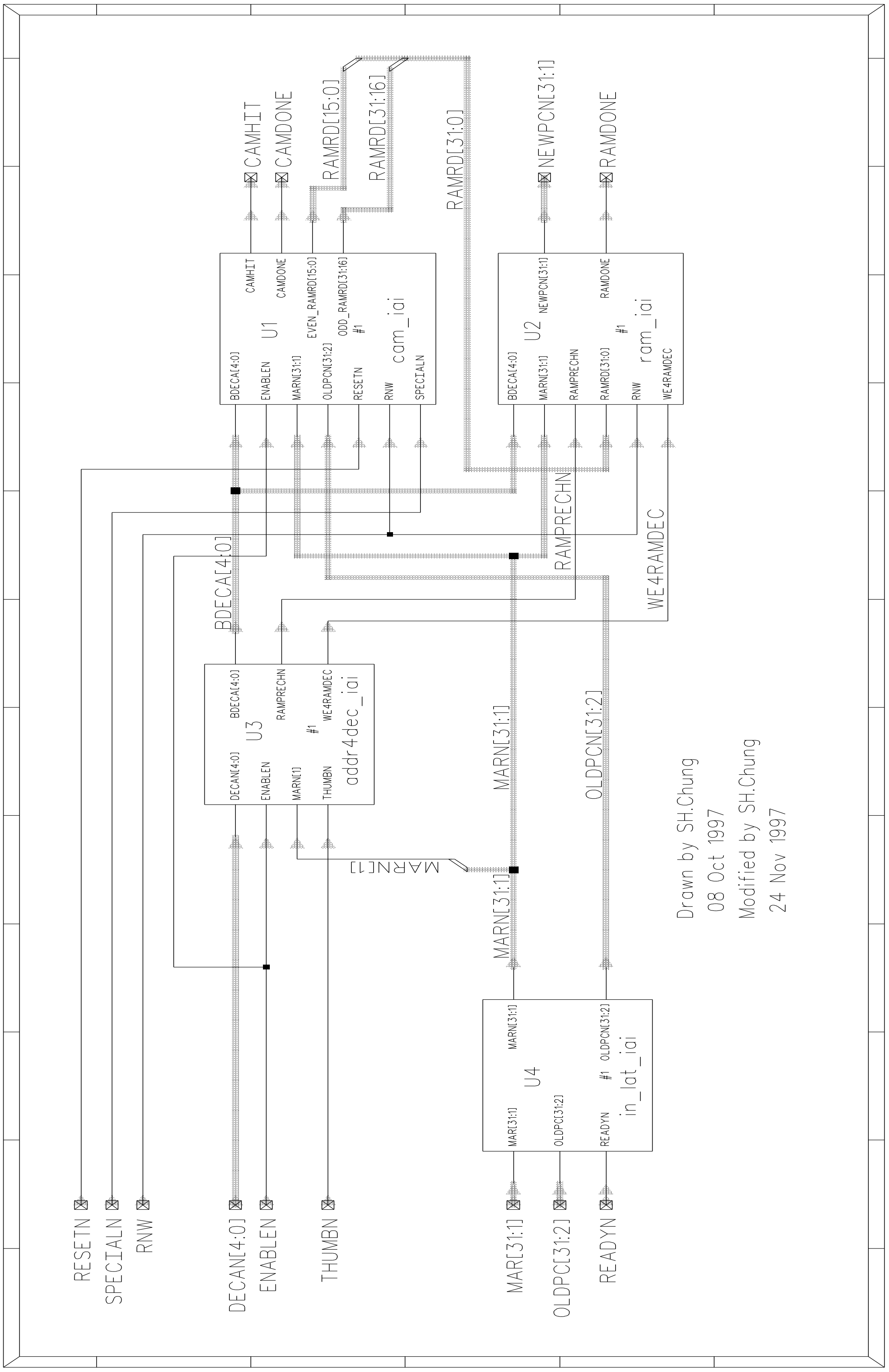
[99] http://www.cs.man.ac.uk/amulet/projects/lard

Schematics A

This appendix contains the schematics of some of the cell library for the *AMULET3* branch target cache. Below is a list of the following appendix sections:

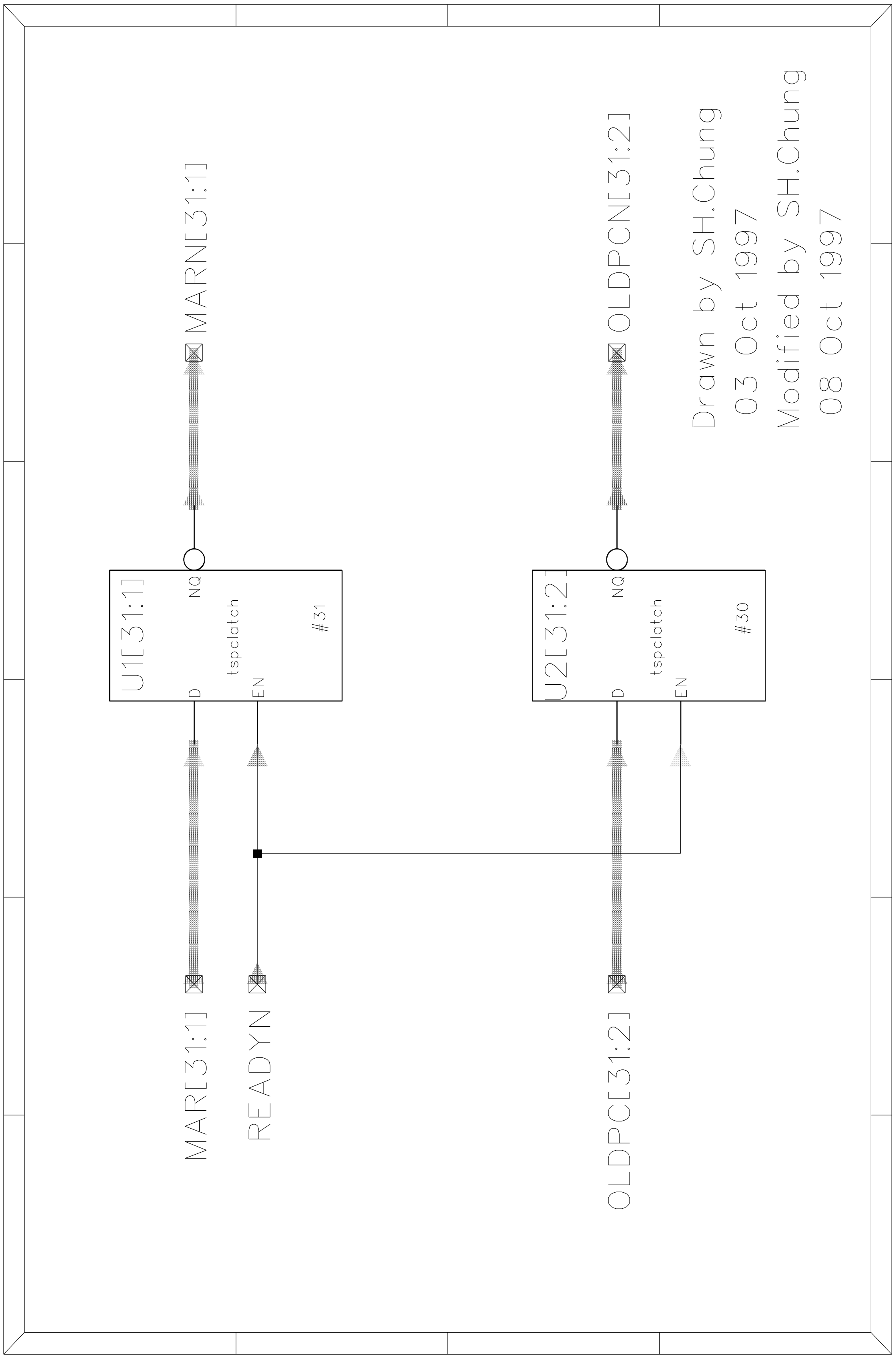
- ❏ **BTC Top**
- ❏ **Input Latch**
- ❏ **TSPC Latch**
- ❏ **Address Decoder**
- ❏ **CAM Top**
- ❏ **CAM Driver**
- ❏ **CAM High Precharge**
- ❏ **CAM Low Precharge**
- ❏ **CAM Cell**
- ❏ **CAM Odd Hit Check**
- ❏ **CAM Even Hit Check**
- ❏ **RAM Top**
- ❏ **RAM Precharge**
- ❏ **RAM Cell**

A.1 BTC Top

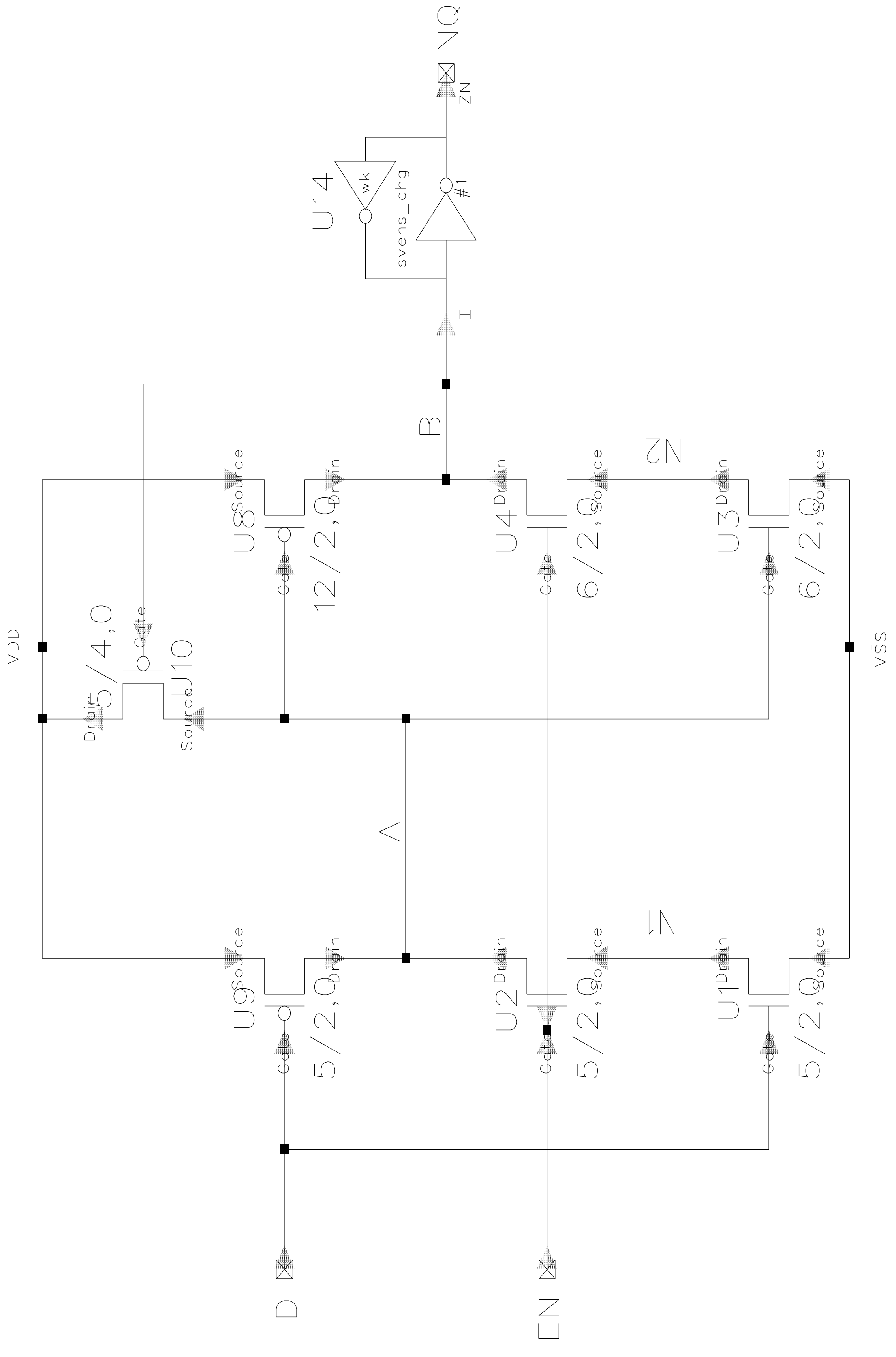


CAMHIT
CAMDONE
RAMRD[15:0]
RAMRD[31:16]
RAMRD[31:0]
NEWPCN[31:1]
RAMDONE
U1
cam_iai
#1
BDECA[4:0]
ENABLEN
MARN[31:1]
OLDPCN[31:2]
RESETN
RNW
SPECIALN
EVEN_RAMRD[15:0]
ODD_RAMRD[31:16]
U2
ram_iai
RAMPRECHN
RAMRD[31:0]
WE4RAMDEC
U3
addr4dec_iai
DECAN[4:0]
MARN[1]
THUMBN
U4
in_lat_iai
MAR[31:1]
OLDPC[31:2]
READYN
WE4RAMDEC
MARN[31:1]
OLDPCN[31:2]
DECAN[4:0]
Drawn by SH.Chung
08 Oct 1997
Modified by SH.Chung
24 Nov 1997

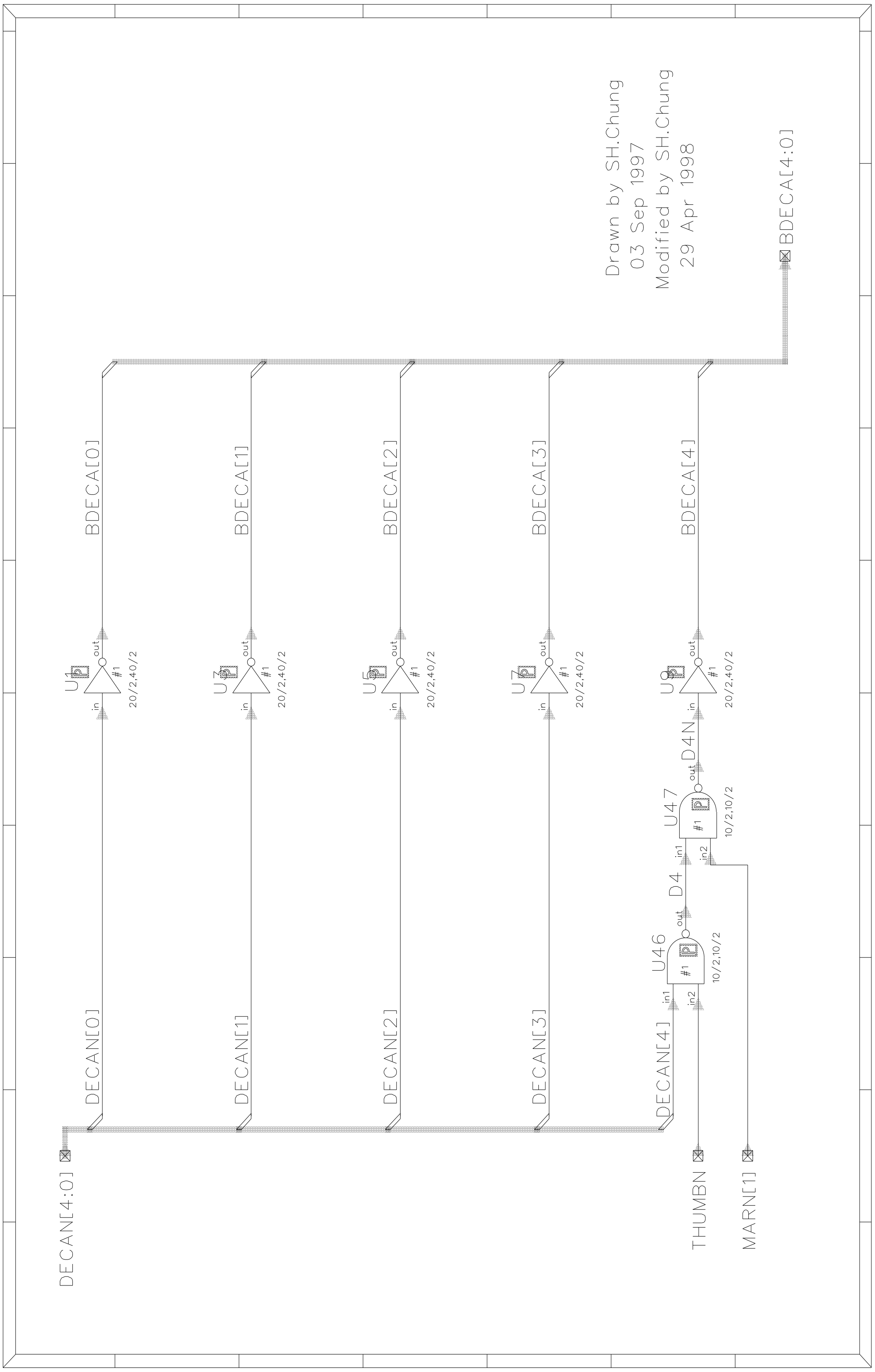
A.2 Input Latch



A.3 TSPC Latch

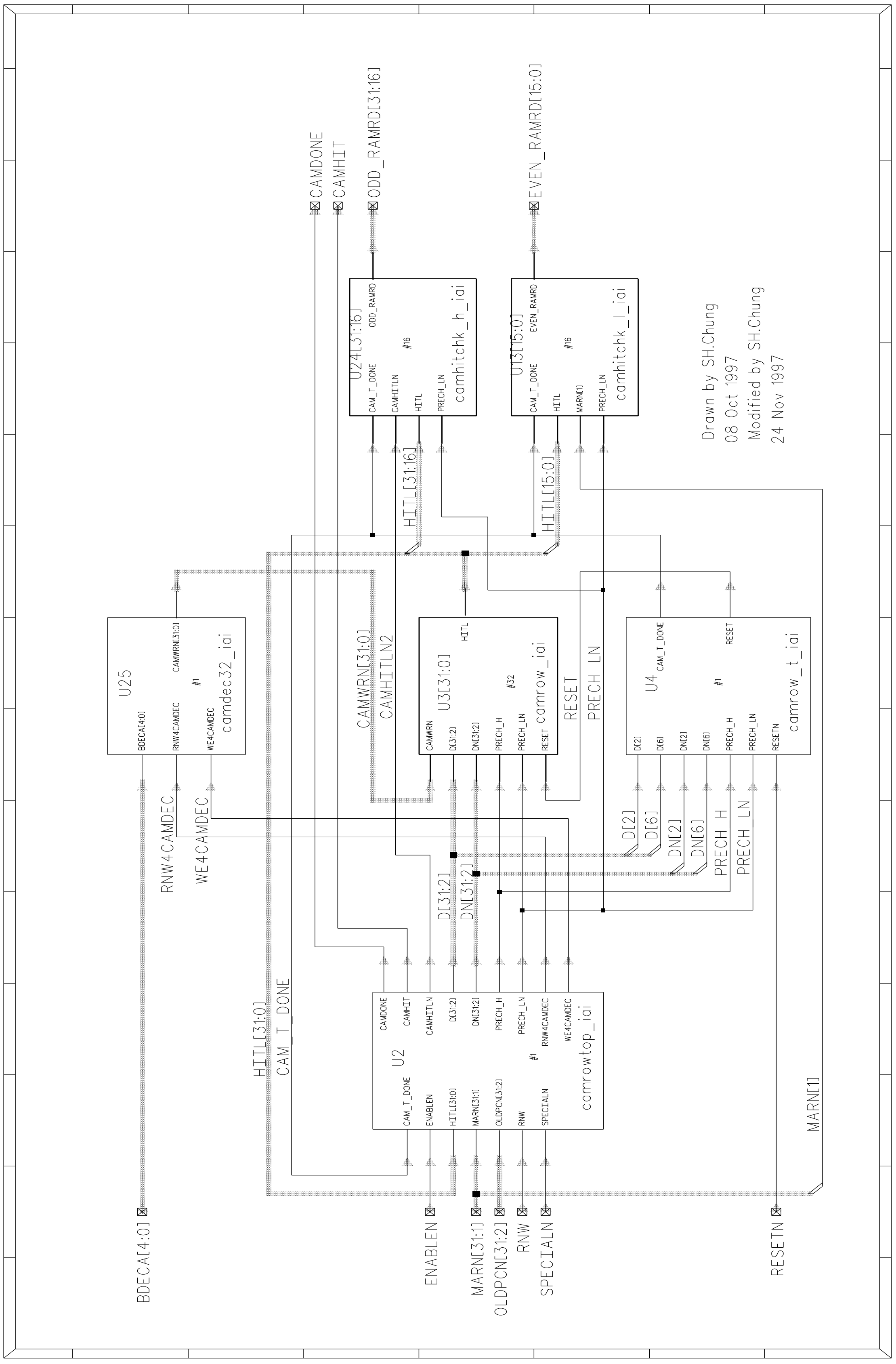


A.4 Address Decoder

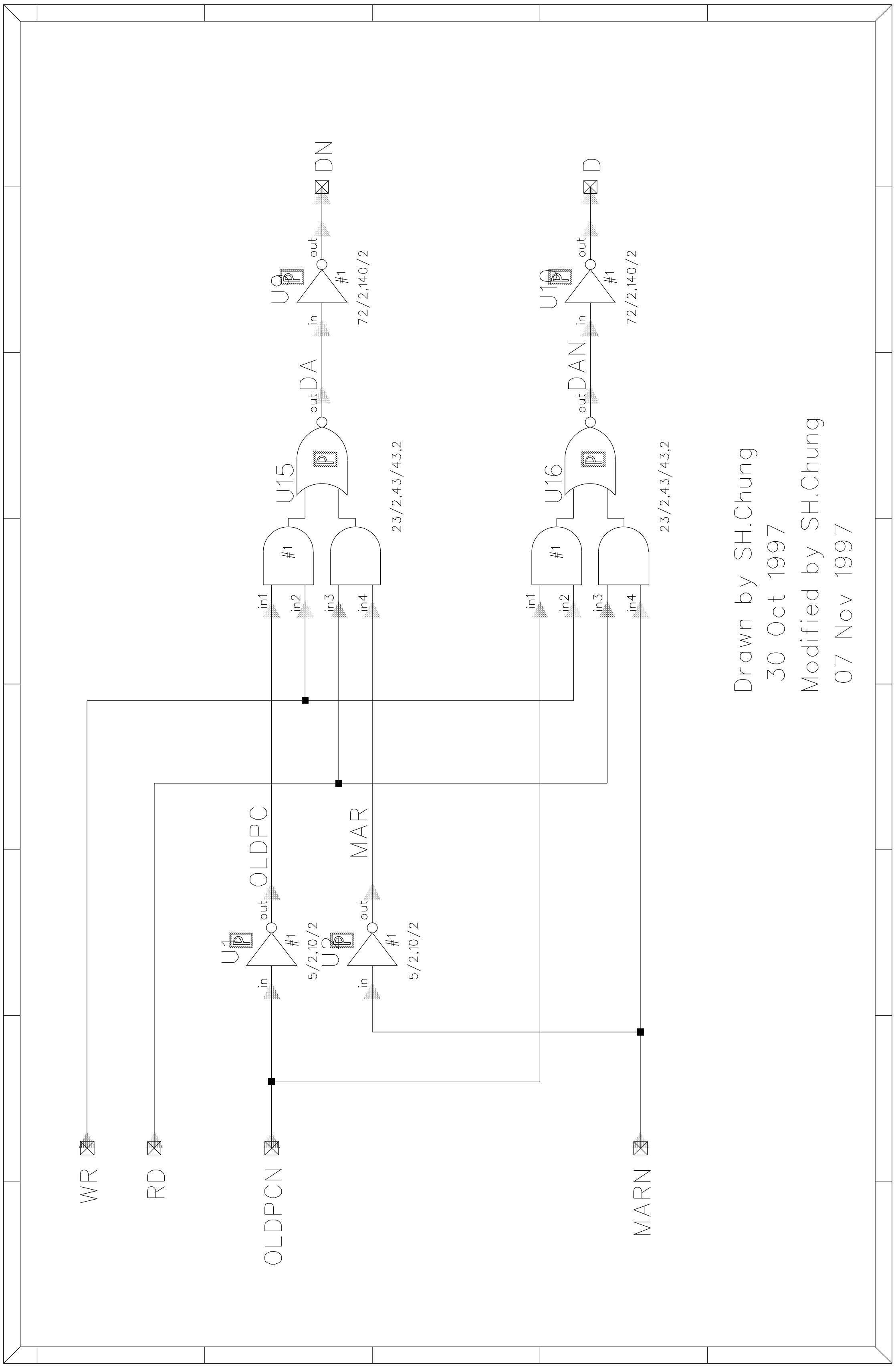


Compass Design Automation plot [la]addr4dec_iai by chungsh on 23-Sep-98 at 6:14 P.M.

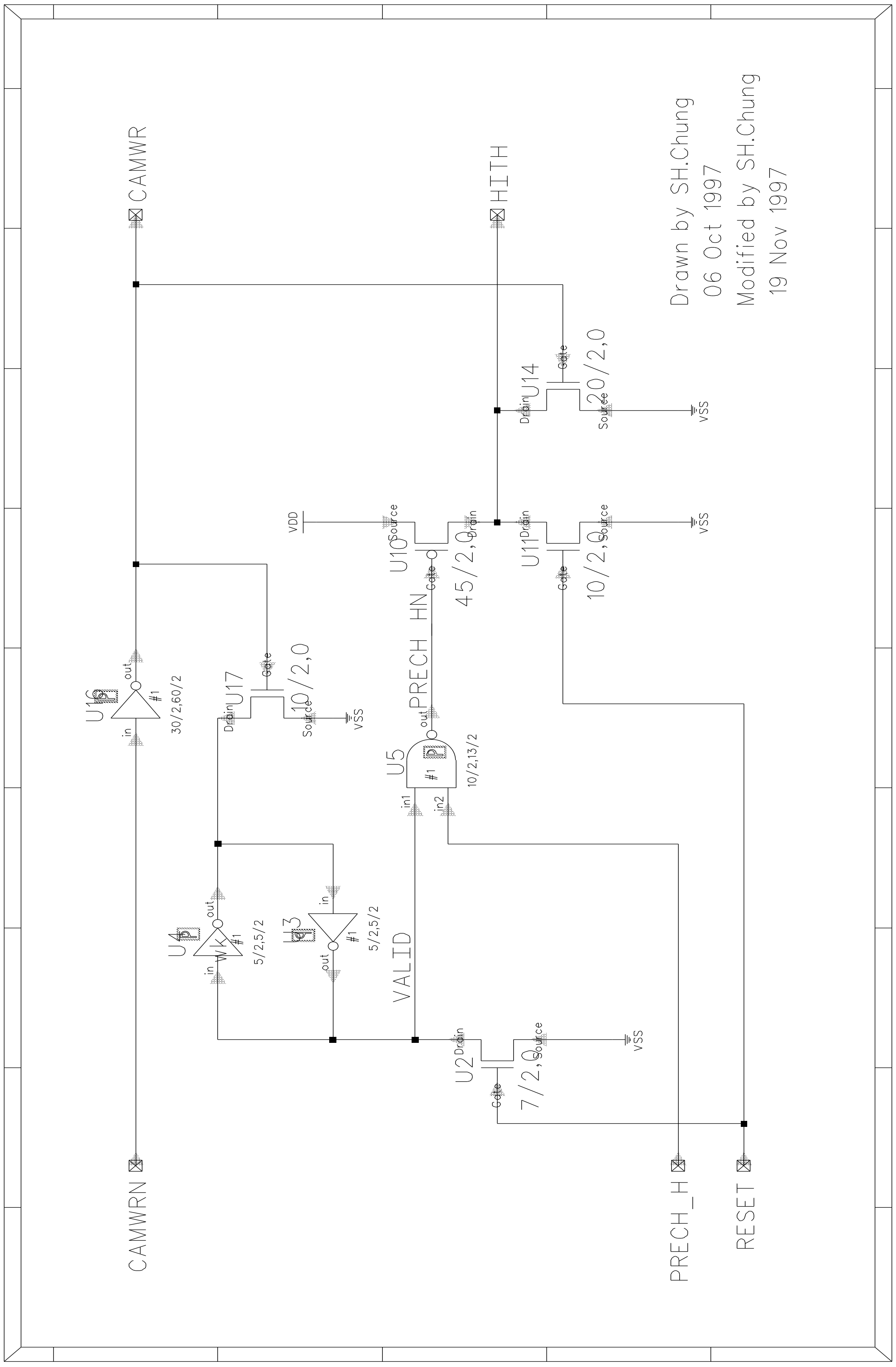
A.5 CAM Top


CAMDONE
CAMHIT
ODD_RAMRD[31:16]
EVEN_RAMRD[15:0]
U24[31:16]
CAM_T_DONE
CAMHITLN
HITL
PRECH_LN
ODD_RAMRD
#16
camhitchk_h_iai
U1[15:0]
CAM_T_DONE
HITL
MARN[1]
PRECH_LN
EVEN_RAMRD
#16
camhitchk_l_iai
Drawn by SH.Chung
08 Oct 1997
Modified by SH.Chung
24 Nov 1997
HITL[31:16]
HITL[15:0]
U25
BDECA[4:0]
RNW4CAMDEC
WE4CAMDEC
CAMWRN[31:0]
#1
camdec32_iai
CAMWRN[31:0]
CAMHITLN2
U3[31:0]
CAMWRN
D[31:2]
DN[31:2]
PRECH_H
PRECH_LN
RESET
HITL
#32
camrow_iai
RESET
PRECH_LN
U4
CAM_T_DONE
RESET
D[2]
D[6]
DN[2]
DN[6]
PRECH_H
PRECH_LN
RESETN
#1
camrow_t_iai
RNW4CAMDEC
WE4CAMDEC
D[2]
D[6]
DN[2]
DN[6]
PRECH_H
PRECH_LN
D[31:2]
DN[31:2]
HITL[31:0]
CAM_T_DONE
U2
CAMDONE
CAMHIT
CAMHITLN
D[31:2]
DN[31:2]
PRECH_H
PRECH_LN
RNW4CAMDEC
WE4CAMDEC
CAM_T_DONE
ENABLEN
HITL[31:0]
MARN[31:1]
OLDPCN[31:2]
RNW
SPECIALN
#1
camrowtop_iai
MARN[1]
BDECA[4:0]
ENABLEN
MARN[31:1]
OLDPCN[31:2]
RNW
SPECIALN
RESETN

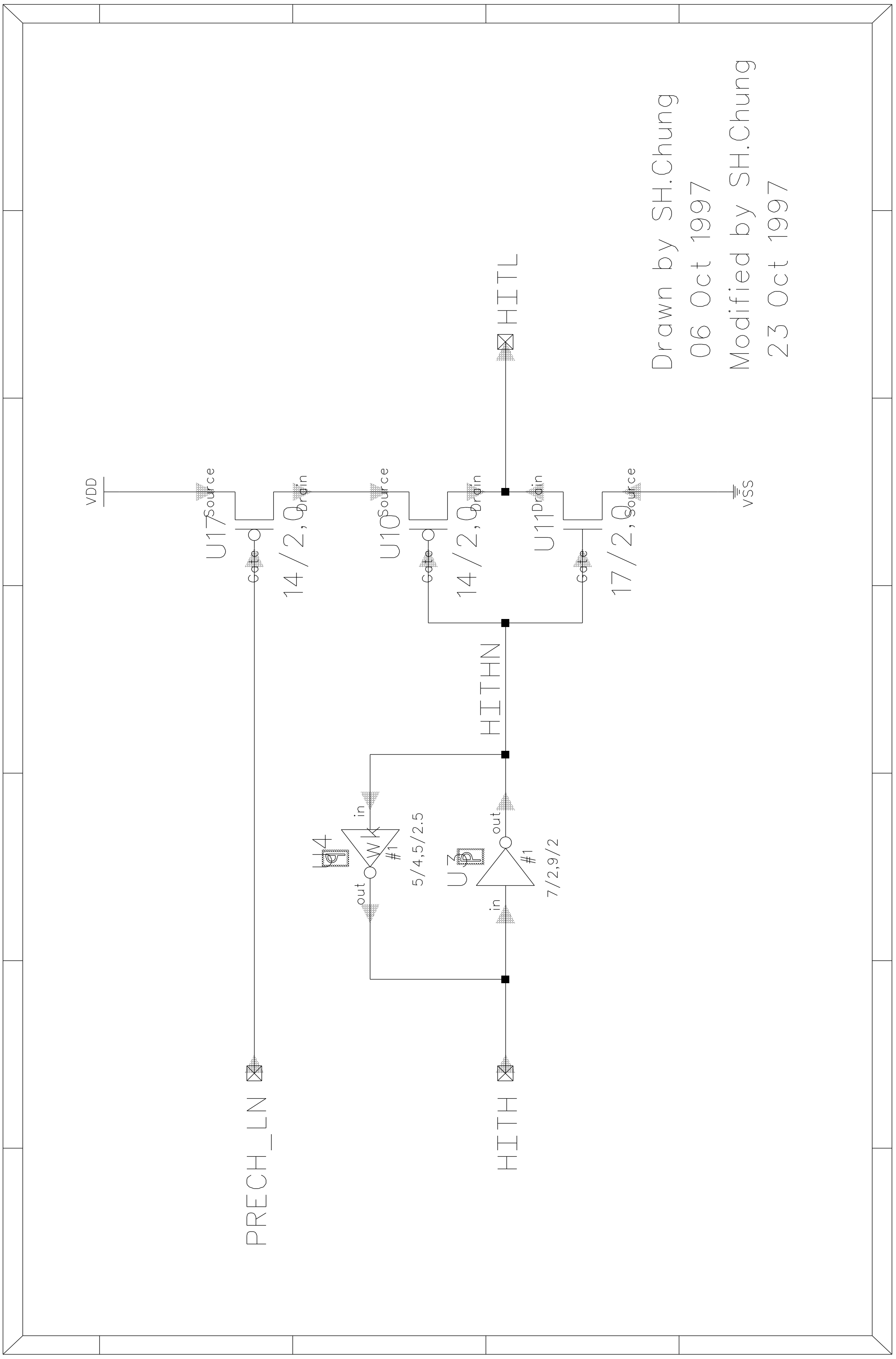
A.6 CAM Driver



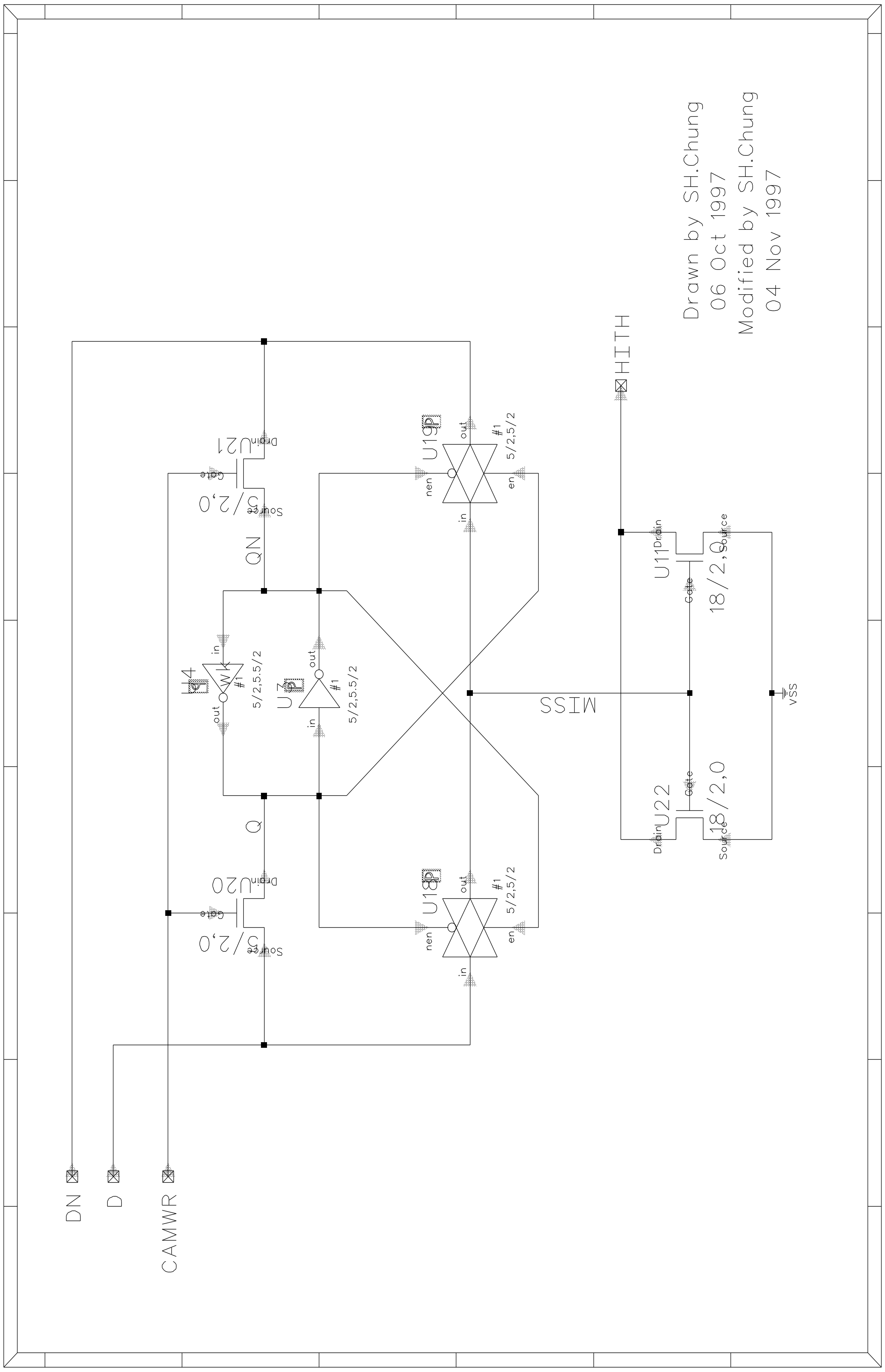
A.7 CAM High Precharge



A.8 CAM Low Precharge

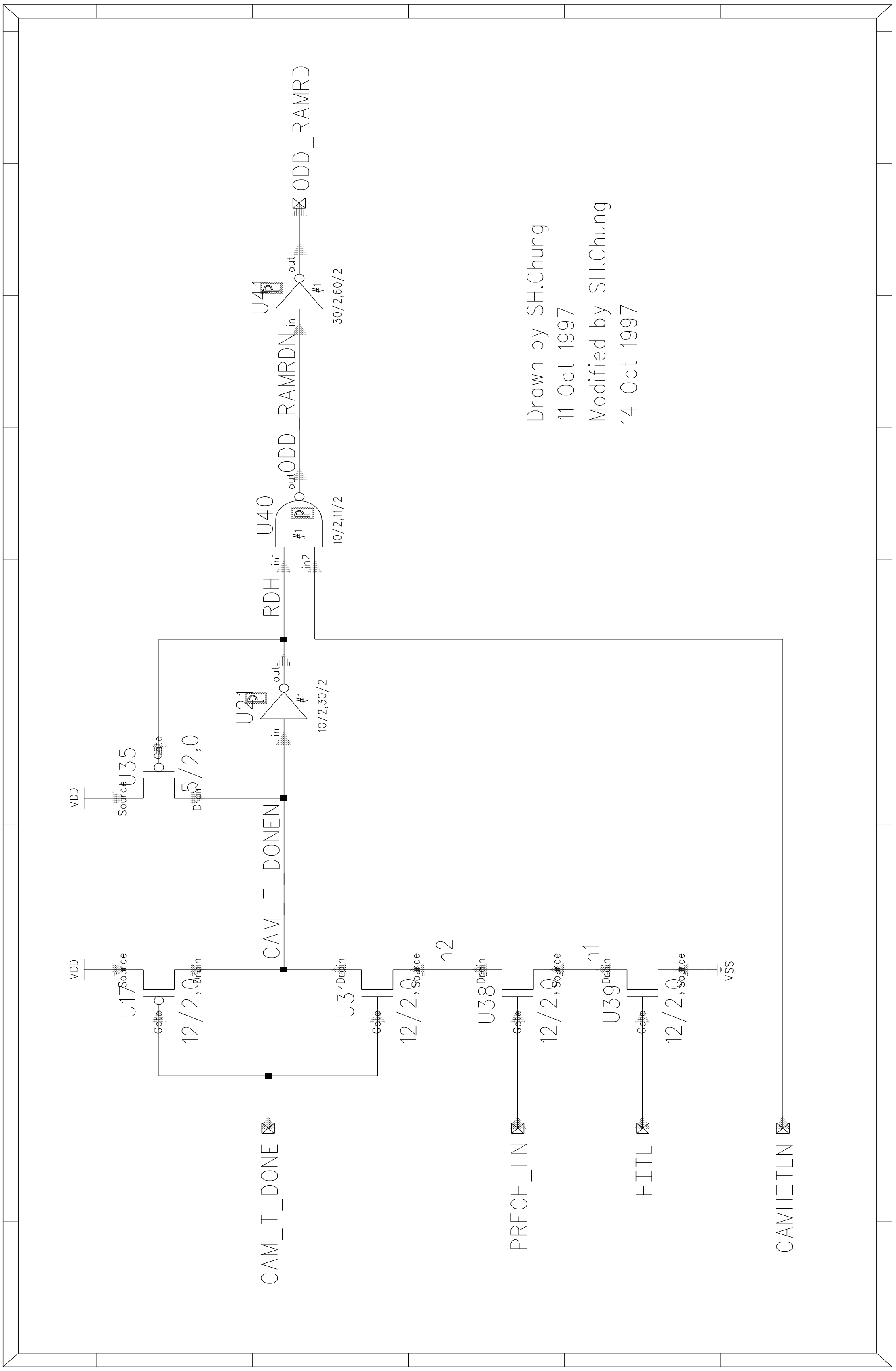


A.9 CAM Cell

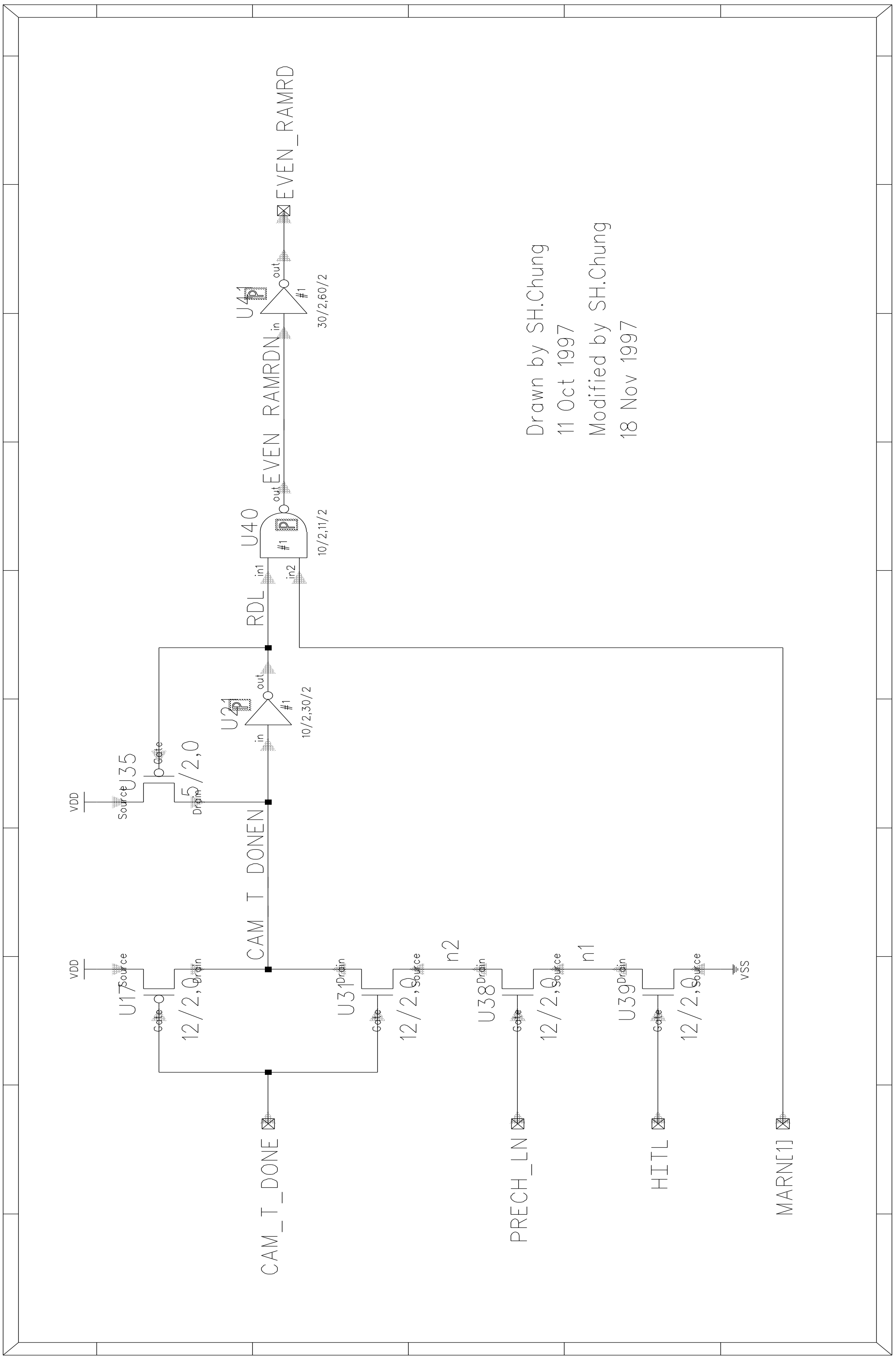


Drawn by SH.Chung
06 Oct 1997
Modified by SH.Chung
04 Nov 1997
HITH
MISS
VSS
U11
18/2,0
U22
18/2,0
U19
U18
5/2,5/2
U21
U20
5/2,0
QN
Q
U4
U3
5/2,5/2
DN
D
CAMWR

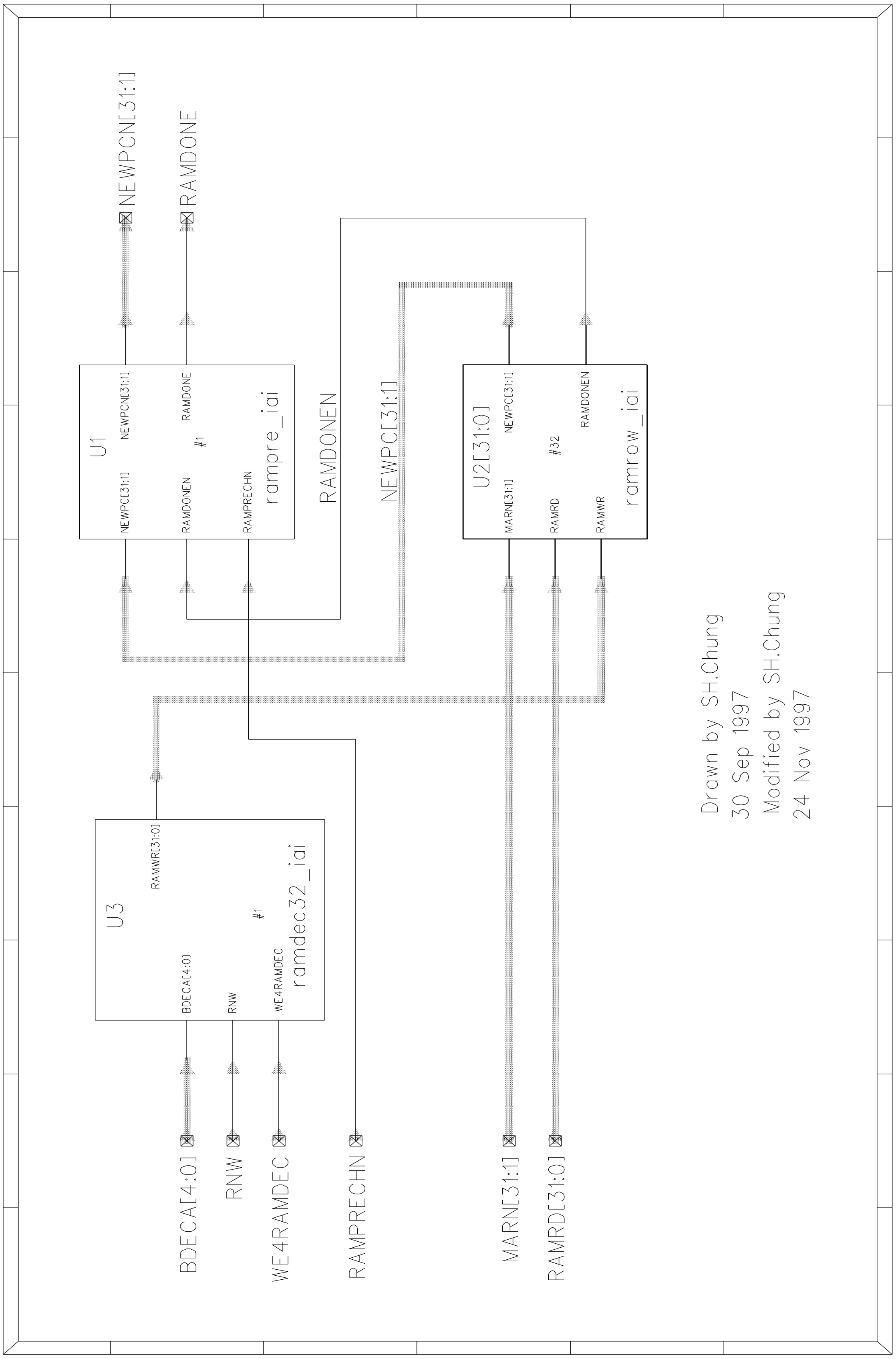
A.10 CAM Odd Hit Check



A.11 CAM Even Hit Check

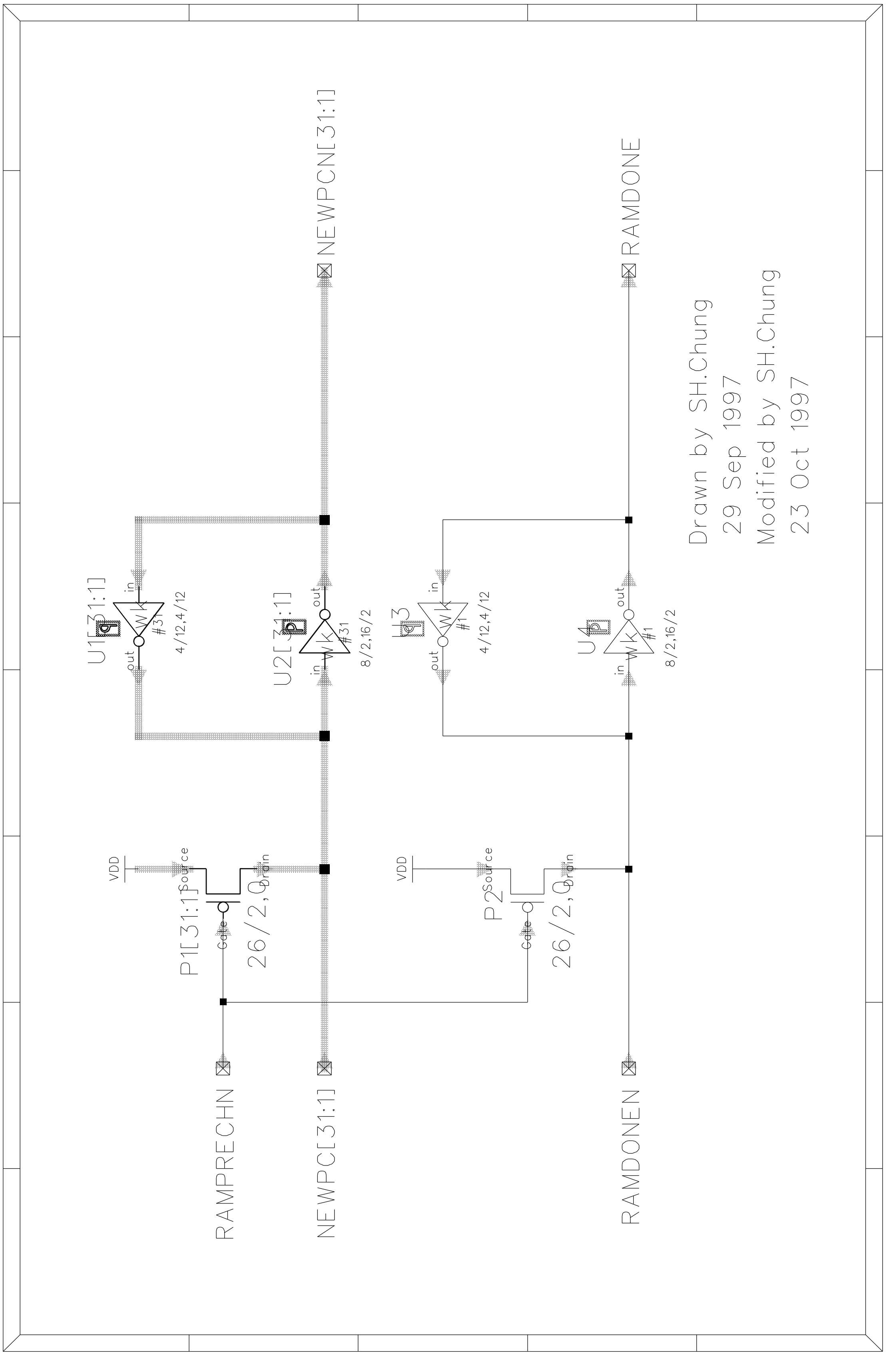


A.12 RAM Top

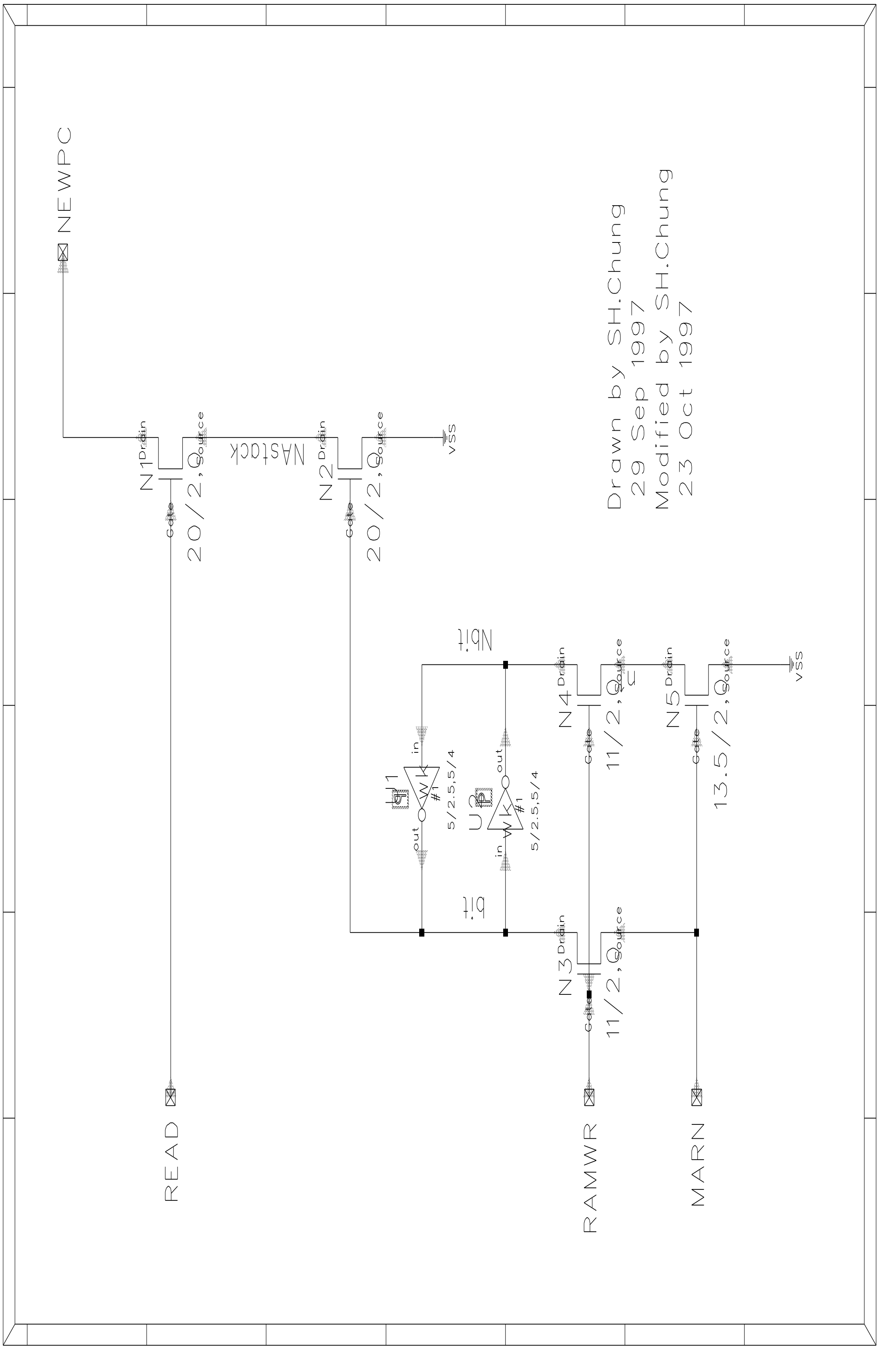


NEWPCN[31:1]
RAMDONE
U1
NEWPC[31:1]
NEWPCN[31:1]
RAMDONEN
RAMDONE
#1
RAMPRECHN
rampre_iai
RAMDONEN
NEWPC[31:1]
U2[31:0]
MARN[31:1]
NEWPC[31:1]
#32
RAMRD
RAMDONEN
RAMWR
ramrow_iai
U3
RAMWR[31:0]
BDECA[4:0]
RNW
WE4RAMDEC
#1
ramdec32_iai
Drawn by SH.Chung
30 Sep 1997
Modified by SH.Chung
24 Nov 1997
BDECA[4:0]
RNW
WE4RAMDEC
RAMPRECHN
MARN[31:1]
RAMRD[31:0]

A.13 RAM Precharge



A.14 RAM Cell

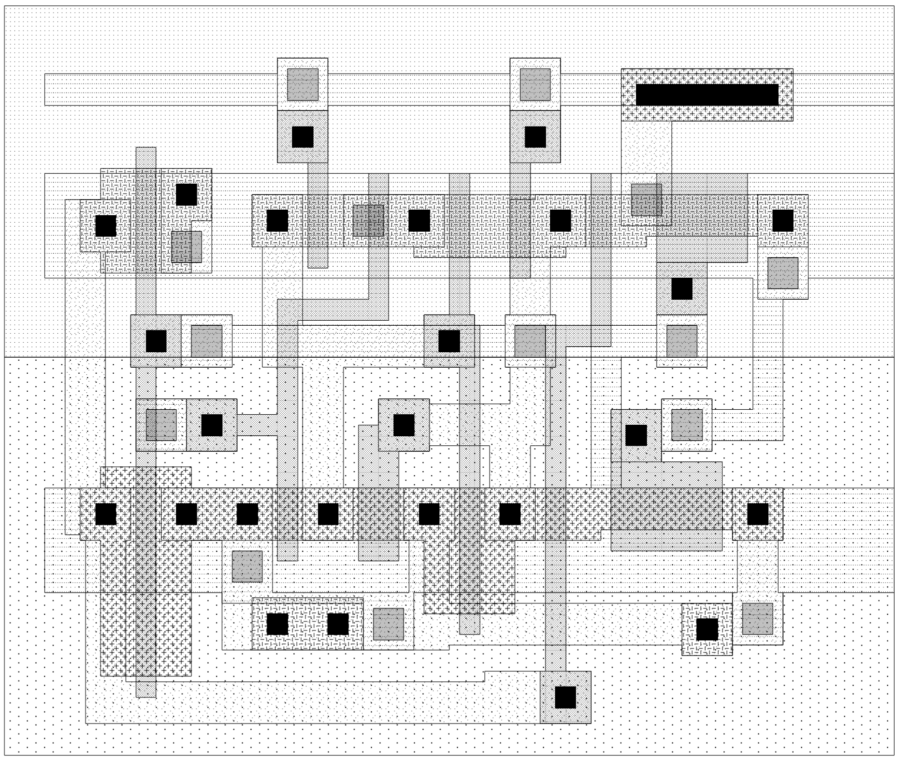

NEWPC
N1
20/2,0
N2
20/2,0
NAstack
VSS
Drain
Source
Gate
Drawn by SH.Chung
29 Sep 1997
Modified by SH.Chung
23 Oct 1997
Nbit
N4
11/2,0
N5
13.5/2,0
U1
U2
WK
#1
5/2.5,5/4
in
out
bit
N3
11/2,0
READ
RAMWR
MARN

Layouts B

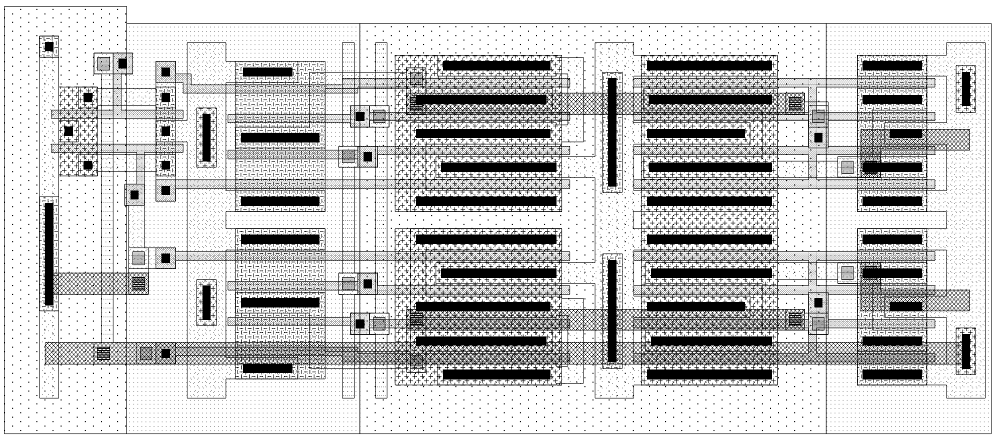
This appendix contains the layouts of some of the cell library for the *AMULET3* branch target cache. Below is a list of the following appendix sections:

- ❏ **TSPC Latch**
- ❏ **CAM Driver**
- ❏ **CAM High Precharge**
- ❏ **CAM Low Precharge**
- ❏ **CAM Cell**
- ❏ **CAM Odd Hit Check**
- ❏ **CAM Even Hit Check**
- ❏ **RAM Cell**

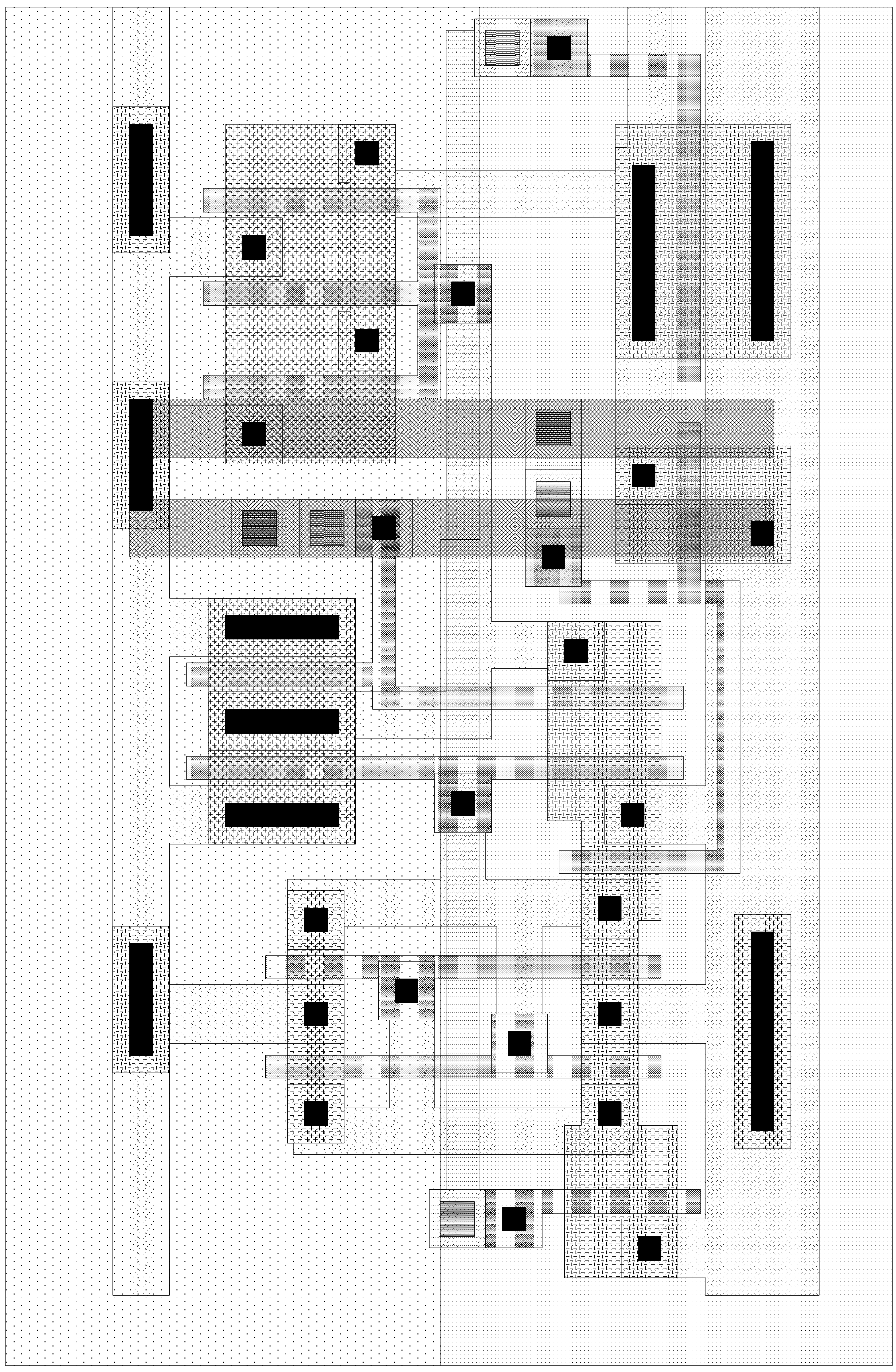
B.1 TSPC Latch



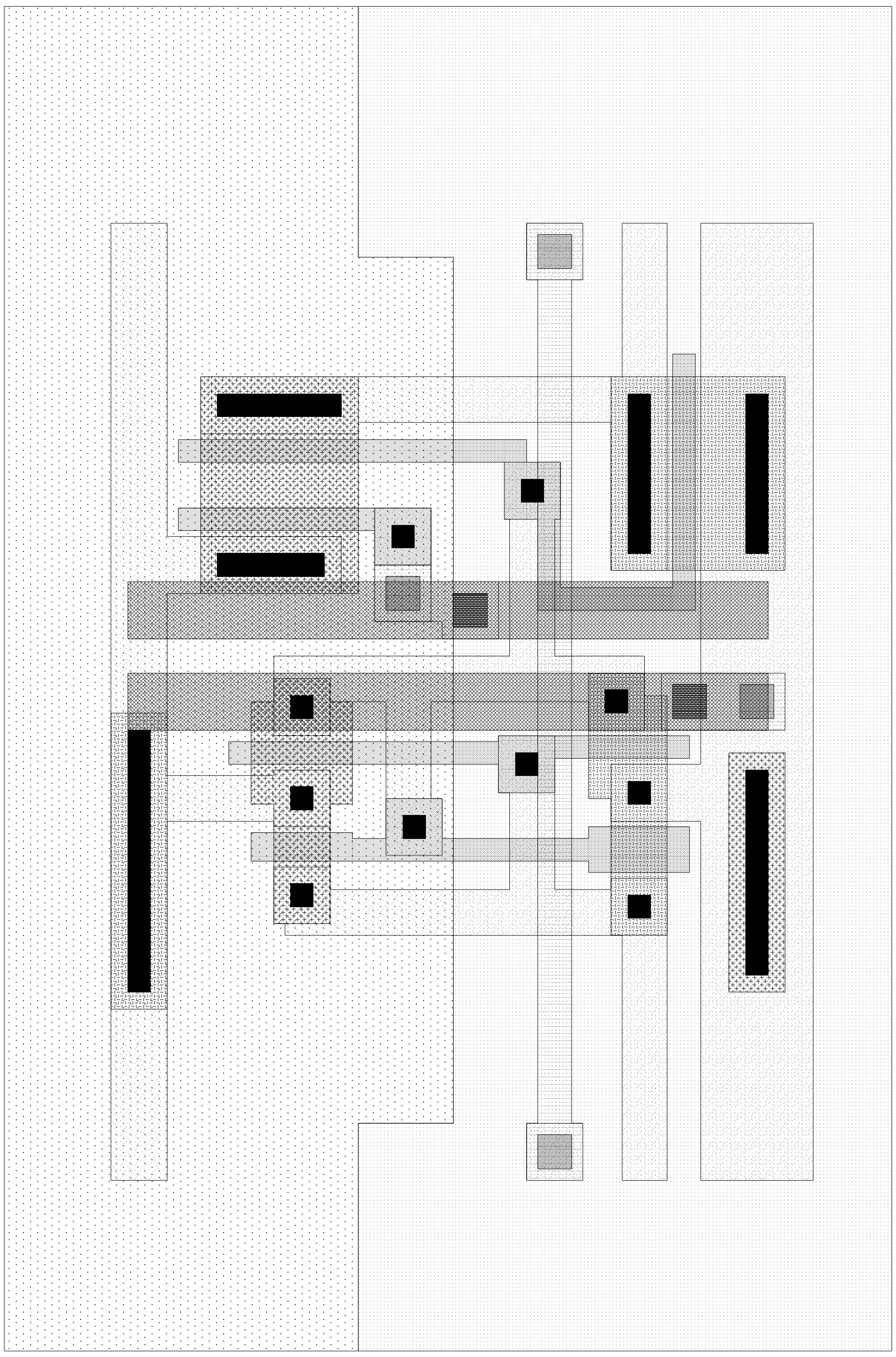
B.2 CAM Driver



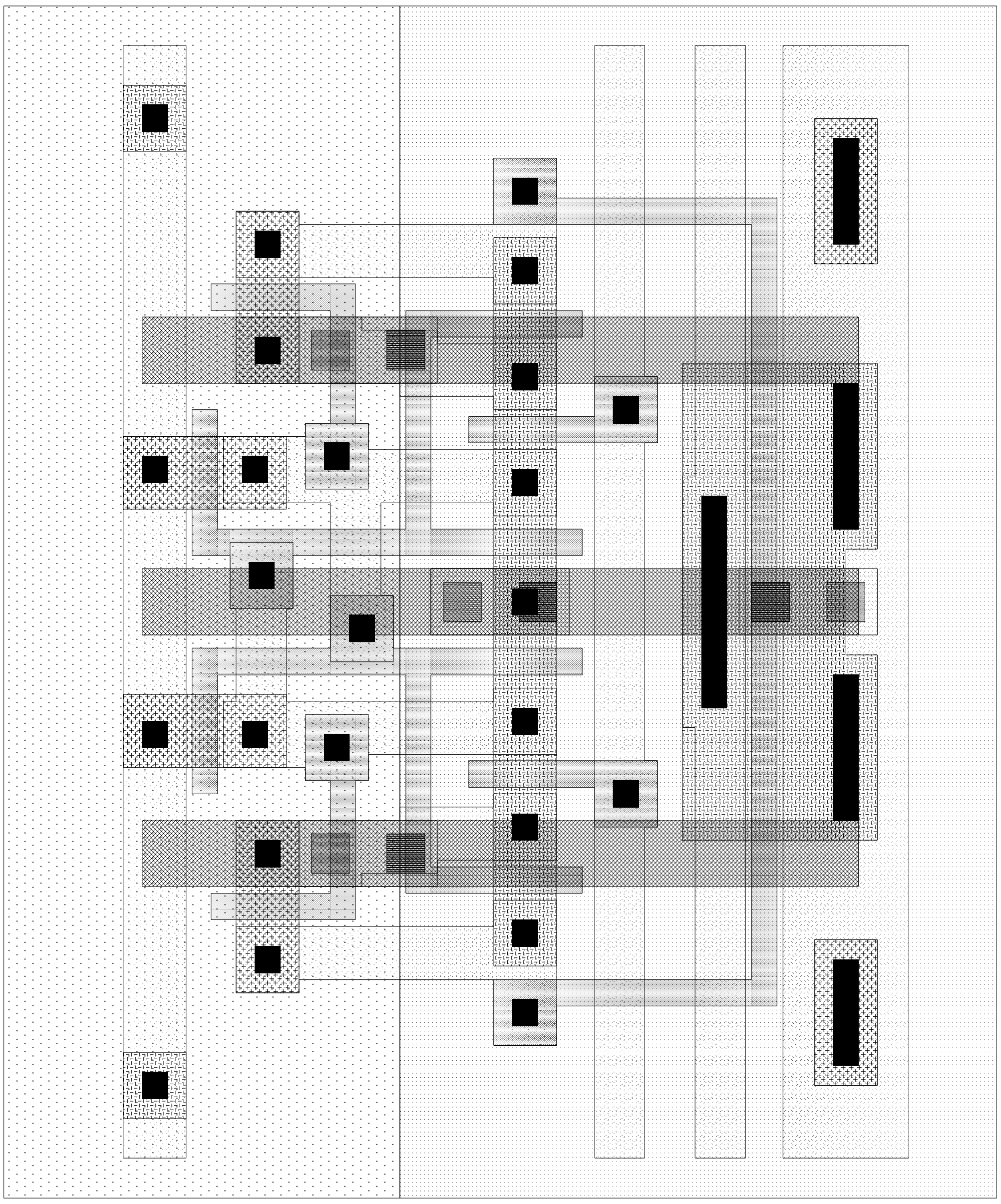
B.3 CAM High Precharge



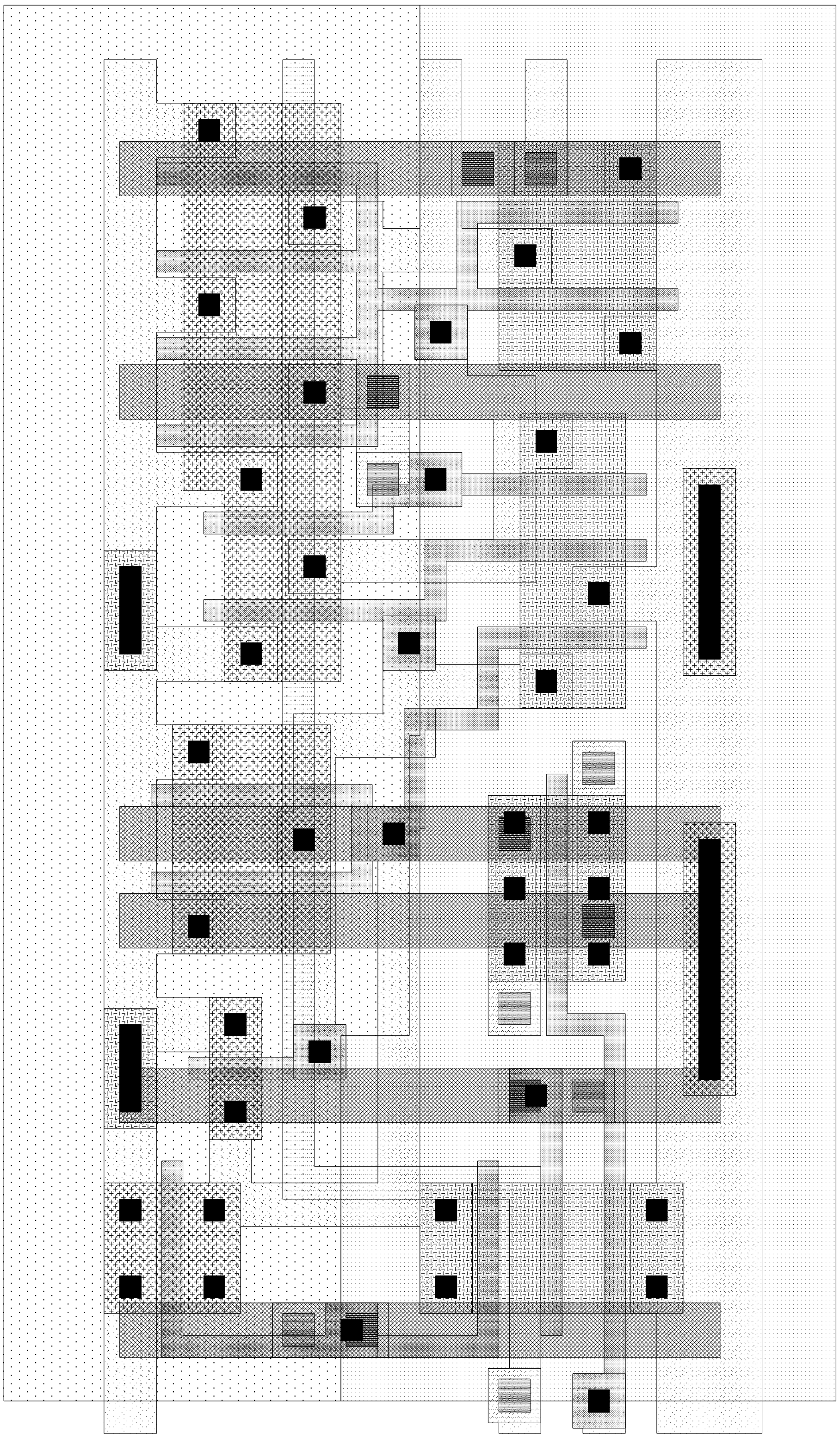
B.4 CAM Low Precharge



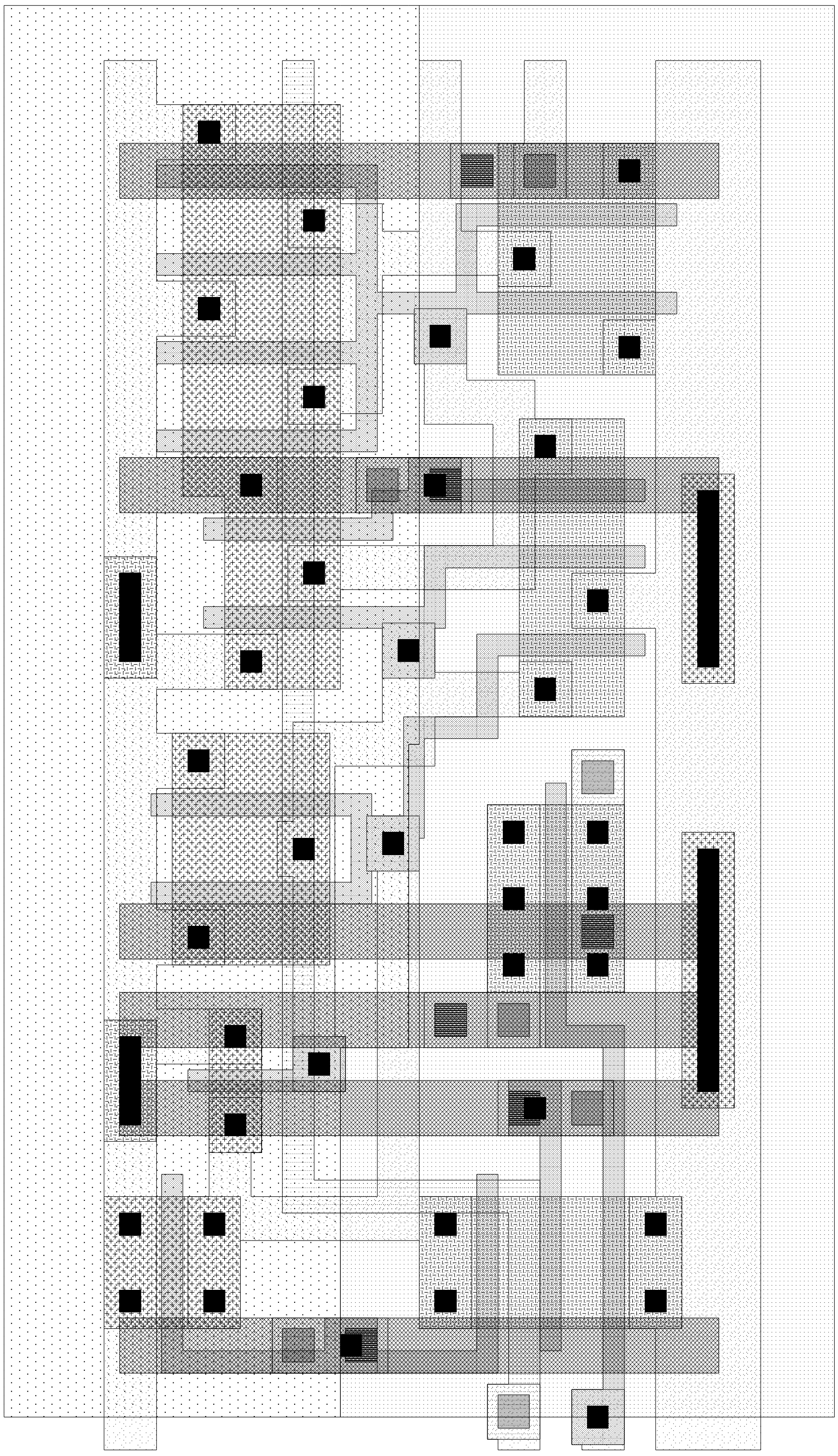
B.5 CAM Cell



B.6 CAM Odd Hit Check



B.7 CAM Even Hit Check



B.8 RAM Cell