The Design of an Asynchronous Multiplier

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

September 1995

By

Jianwei Liu

Department of Computer Science

Table of Contents

List of Figures

List of Tables

Abstract

A high performance, low power asynchronous 32×32-bit multiplier with several novel features has been developed for the *AMULET2* microprocessor at a low hardware cost. A new design for a 4-2 counter using differential pass-transistor logic (DTPL) has been used, together with several circuit design techniques including true single-phase clocking registers, resulting in a computation rate exceeding 200 MHz and about 58 mW power consumption with 0.6 μm three-level metal CMOS process technology. The resulting cell is compact with a datapath area of only $0.32 \times 0.71\ mm^2$.

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.

Copyright and the Ownership of Intellectual Property Rights

The Author

The author was awarded the degrees of Bachelor of Science and Master of Science, both in Electrical and Electronic Engineering, at Shenyang Polytechnic University and Harbin Institute of Technology, China, in 1984 and 1987, respectively. Significant experience was gained from involvement in the 16-bit and 32-bit microprocessor, FPGA and ASIC chip designs at the Northeast Microelectronics Institute, China, from 1987 to 1993. He became interested in asynchronous design, VHDL design and formal approaches to hardware design when at the Technical University of Denmark, as a visiting scholar, from 1993 to 1994. 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 Nigel Paver and Paul Day for their efforts to make me feel easier in my research work. Thanks to Steve Temple, Jim Garside, Craig Farnsworth, Oleg Petlin and Phil Endecott for their kind help. Thanks to Dave Gilbert for his reading and commenting on the draft of this thesis. Thanks to Sukendeep Samra 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 grants from ORS and URS.

Last but not least, I am grateful to Professor Jørgen Staunstrup, who brought me into the asynchronous world when I stayed at the Technical University of Denmark as a visiting scholar from 1993 to 1994.

Dedication

To

My parents — Pengqi Liu and Shuhua Zhang

My wife — Li He

And my son — Dan Liu

献给：

我的父母 -- 刘鹏起 和 张淑华

我的妻子 -- 何莉

我的儿子 -- 刘丹

Introduction 1

There is considerable interest in the application of asynchronous logic design techniques to the development of systems with low power consumption. Asynchronous logic design techniques have widened the choice available for low power design, but have also been questioned and some objections raised. The two main objections are the issue of the feasibility of developing asynchronous systems at the levels of complexity of current synchronous designs, and the fact that no demonstrable large-scale fabricated chips are available to endorse the asynchronous logic design methodology.

In April 1994 Professor Steve Furber's *AMULET* (Asynchronous Microprocessor Using Low Energy Techniques) group in the Department of Computer Science at the University of Manchester took delivery of the *AMULET1* microprocessor [1,2,3,4], the world's first implementation of a commercial microprocessor architecture (*ARM*) [5] in asynchronous logic. The *AMULET1* work answered the first question by demonstrating the feasibility.

The *AMULET2* microprocessor, a major rework of *AMULET1* incorporating several novel features, is currently under development. This research has the objective of realizing an asynchronous microprocessor with higher performance and lower power consumption than is currently available using synchronous techniques. The on-going *AMULET2* work promises to overcome the second objection by supplying a high performance and low power chip.

1.1 Why low power?

Low power design has emerged as a major technology driver due to two independent factors: market forces and packaging costs.

The market for portable computing and wireless communication networking are growing rapidly. Battery life is a very important specification point for those products that use batteries for power such as portable computers and personal digital assistants. Every bit of power saved in a low power circuit contributes directly to prolonged battery life.

Another problem relates to packaging costs for more power-hungry chips. Chips that consume 30 to 40 watts are not unusual today. Advanced packaging technologies need to be used to remove the heat from the chip. However, sophisticated packaging means an increase in cost, and associated cooling for high power also results in high noise. It is better to design low power circuits instead of accepting high power consumption.

1.2 Why asynchronous design?

Low power design spans a broad range from semiconductor devices, process technologies, circuit techniques, digital logic styles, and design methodologies all the way up to architecture, algorithms, and applications of complete VLSI systems. One active area of low power design covers the asynchronous world [6] that is supposed to be an ideal paradigm for low power applications due to the following reasons:

- Asynchronous circuits only dissipate power when and where they are active, whereas synchronous circuits always remain active irrespective of whether or not they are doing anything useful. Some advanced synchronous circuits may exploit power management mechanisms that can shut down the idle parts of the circuit with extra circuitry help. However, asynchronous circuits have this efficiency by nature.
- Asynchronous circuits abandon global synchrony, instead using local handshaking protocols. Global clocks in a synchronous circuit often account for a significant component of the total power consumption. Previous work has shown that the clock power is about twice the logic power for static logic and about three times the logic power for dynamic logic [7].
- The modularity of asynchronous circuits allows one to break up a system into subsystems that can be optimized for low power.
- Asynchronous circuits can be optimized for typical conditions. More power is wasted than is necessary under typical conditions in a synchronous circuit, because considerable circuitry is employed to make rare worst case operations fast enough to operate within a fixed clock period.
- Asynchronous circuits increase the latch-enable efficiency and avoid the need for using slave latches. On the other hand, synchronous circuits employ slave latches in order to provide high reliability by eliminating two-sided timing constraints [8], which, in general, doubles the number of latches required.
- The handshaking mechanism of asynchronous circuits gets rid of spurious transitions and each transition has its own role; this also contributes to power savings.

However, it is worth noting that extra transitions required for synchronization and the dual-rail implementation of asynchronous circuits work against power efficiency. So, there is the need to investigate the asynchronous logic design methodology to see to what extent the potential power savings of asynchronous design can be realized. This is the motivation behind the *AMULET2* project.

1.3 MSc work

A high performance, low power asynchronous 32 × 32-bit multiplier with several novel features has been developed for the *AMULET2* microprocessor at a low hardware cost. The MSc work described in this thesis involved the development of this multiplier, in which attention is focused on circuit design techniques.

1.4 Thesis structure

This thesis is organized into two main parts. The first part (Chapters 1 through 5) describes the purpose of the project and design methodologies for low power. This includes asynchronous logic styles and circuit design issues such as speed, power, noise margins, and clocking. The second part (Chapters 6 through 11) deals with multiplication algorithms, the *AMULET2* multiplier organization, the *AMULET2* multiplier circuit design and implementation, evaluation, and conclusions.

Chapter 1 gives a brief introduction.

Chapter 2 treats low power design methodologies.

Chapter 3 surveys asynchronous logic styles.

Chapter 4 introduces pass-transistor logic for low power design.

Chapter 5 presents the true single-phase clocking (TSPC) scheme.

Chapter 6 summarizes multiplication algorithms.

Chapter 7 describes the organization of the *AMULET2* multiplier.

Chapter 8 covers the circuit design of the *AMULET2* multiplier.

Chapter 9 deals with the layout design of the *AMULET2* multiplier.

Chapter 10 evaluates the *AMULET2* multiplier design.

Chapter 11 makes some concluding remarks about this work.

Appendix A provides schematics of the datapath of the *AMULET2* multiplier.

Appendix B contains layouts of the cell library for the *AMULET2* multiplier.

Low-power design 2

Dynamic switching power is the dominant component of overall power consumption in a "well-designed" CMOS circuit. It is proportional to the net loading capacitance, the transition frequency, and the square of the supply voltage:

Dynamic Switching Power = $\boldsymbol{C \bullet f \bullet V_{dd}^{2}}$

C ~ the net loading capacitance

f ~ the transition frequency

V_{dd} ~ the supply voltage

As the equation above illustrates, low power design can be achieved by reducing any of these three factors.

2.1 Supply voltage

Clearly, reducing the supply voltage is the first choice for power reduction since it has the greatest effect on the power due to its quadratic contribution. However, this comes at the cost of degraded performance and reduced noise margins unless some compensating changes are made. The engineering trade-offs [9] that must be made are becoming extremely important and difficult.

2.1.1 Scaling limit

Firstly, let's look at to what extent the supply voltage can be lowered, still using currently available CMOS design techniques. This can be studied simply using the symmetrical CMOS inverter whose transfer characteristic is shown in figure 2-1. The minimum supply voltage should cover three parts: the logic low noise margin NM_L, the logic high noise margin NM_H, and the transition width TW that provides a measure of the input sensitivity. Therefore, the minimum supply voltage is defined by these three factors.

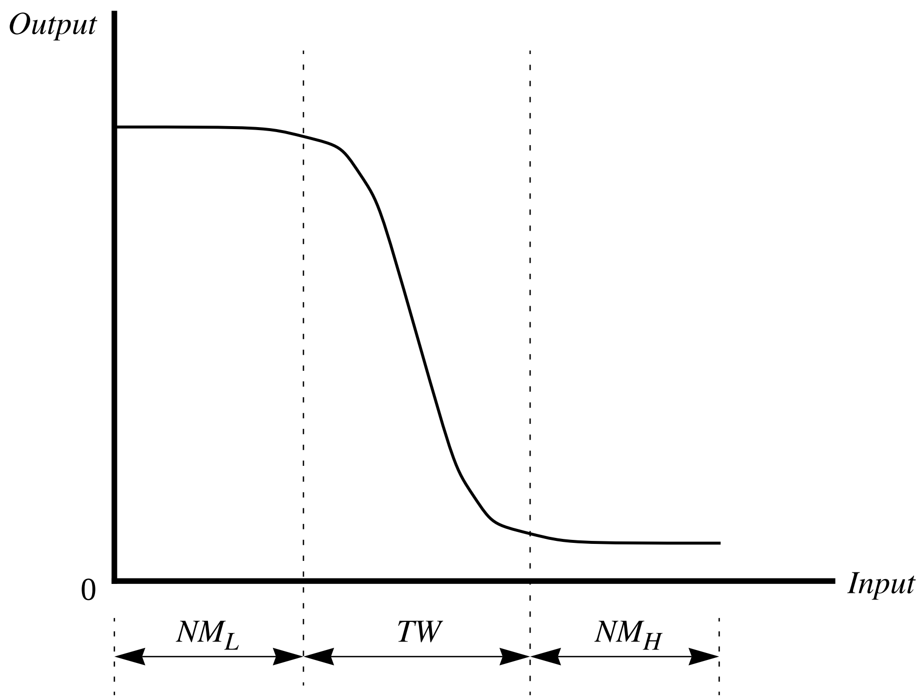


Figure 2-1: CMOS inverter transfer characteristic

Lowering the threshold voltage is a prerequisite for a circuit with a low supply voltage since it is the key compensating method to offset the degraded performance resulting from the low supply voltage. The question is how low can the threshold voltage go before reaching the limit set by the requirement for low subthreshold current. The threshold voltage is determined by both the subthreshold slope S_{th} and the on-off current ratio. The subthreshold slope S_{th}, defined as the amount of voltage required to reduce the subthreshold current by a factor of ten, ranges between 60 and 90mV/decade current in today's technologies [10]. The on-off current ratio usually lies between 10^2 and 10^5 for most VLSI applications. From the view of low power design, the higher the on-off current ratio is, the better the circuits are. Given the subthreshold slope S_{th} and the on-off current ratio, the threshold voltage V_t can be determined as shown in table 2-1.

Table 2-1: Optimal threshold voltage V_t

Threshold voltage V_t (volt)		On-off current ratio			
		10^2	10^3	10^4	10^5
Subthreshold slope S_{th} (mV / decade current)	60	0.12	0.18	0.24	0.30
	70	0.14	0.21	0.28	0.35
	80	0.16	0.24	0.32	0.40
	90	0.18	0.27	0.36	0.45

In what follows, the threshold voltage V_t is assumed to be 0.3 volt including variations due to process and temperature fluctuations (about 0.1 volt). The noise margins are assumed to be twice the threshold voltage V_t (this noise margin is reasonable because of the reduced switching current in a circuit with a low supply voltage). The first order estimate of the transition width TW is about $(V_{dd} - 2V_t)/4$ [11]. Therefore,

$$V_{dd} = NM_L + NM_H + TW = 2Vt + 2Vt + (V_{dd} - 2V_t)/4 = (V_{dd} + 14V_t)/4.$$

So, $V_{dd} = 14V_t/3 = 1.4$ (volt). This means that it is possible to design a low power circuit with a 1.4 volt supply voltage still using currently available CMOS circuit techniques through careful circuit design and good engineering compromises. However, for a circuit with a supply voltage below 1.4 volt, it is necessary and desirable to exploit novel circuit techniques. Multi-threshold voltage CMOS techniques (similar to multi-threshold voltage nMOS techniques, e.g., used in *Intel* nMOS 8086 16-bit microprocessor design) seem an optimal way to achieve the above goal.

2.1.2 Performance

While a change from a 5 volt supply voltage to a 1.5 supply voltage gives over 90% reduction in power consumption, some issues such as degraded performance and reduced noise margins arise.

There are two factors responsible for speed degradation. Firstly, reducing the supply voltage means reducing the charging and discharging drive capability, which is the main reason for speed degradation. Secondly, reducing the supply voltage forces a circuit to operate in the low power operating mode [12], which is often ignored by designers. A CMOS circuit operates in two distinct modes at high and at low supply voltage. At a high supply voltage, series transistors charge or discharge simultaneously. At a low supply voltage, charging or discharging of a series transistor chain occurs sequentially, and therefore the delays become significant larger.

A considerable speed compensation can be gained through the use of low-threshold transistors. However, there is a limit on how low the threshold voltage can be set (see “Scaling limit” on page 20). A speed enhancement can also be achieved using dedicated circuit design (and layout design) techniques such as pass-transistor logic design (see “Pass-transistor logic” on page 41). In addition, close attention must be paid to the low power operating mode and the arrival sequence of input signals in a circuit. It is worth noting that “power-down” schemes [13] may be useful for some applications.

2.2 Capacitance

The net loading capacitance consists of the gate capacitance, source/drain capacitance, and interconnect capacitance. A small loading capacitance resulting in speed enhancement also leads to power savings.

2.2.1 Transistor sizing

Whatever other low power design techniques are used, transistor size optimization will play an important role in reducing power consumption. Using smaller geometry transistors is a good way to minimize the loading capacitance if the interconnect capacitance is not a dominant factor. A chip area decreased by careful layout design also contributes to a reduction of the net capacitance.

2.2.2 True single-phase clocking (TSPC)

It is important to minimize the number of global control nets by using the true single-phase clocking (TSPC) methodology (see “TSPC methodology” on page 47). A TSPC register provides almost a 50% power reduction over a non-overlapping two-phase register [14].

2.2.3 Sub-micron process technology

As process technology has advanced, one important change should be noticed. While the source/drain capacitance was about 2 ~ 3 times the gate capacitance in a typical 2 μm process [15], the gate capacitance is about 3 ~ 5 times the source/drain capacitance in today's process technology (e.g., the gate capacitance is 3 times the source/drain capacitance in *VLSI Technology's CMN5* CMOS process). This change has significant implications. Firstly, the considerable advantage dynamic logic held over static logic due to the small self-load drain capacitance does not exist, which is one of the reasons why static logic is returning to a central position in today's CMOS design. Secondly, it paves the way for pass-transistor logic. This can easily be explained referring to figure 2-2. If the source/drain capacitance is larger than the gate capacitance, the path with a small capacitance through a transistor is that shown in figure 2-2 (a). On the other hand, if the source/drain capacitance is smaller than the gate capacitance, the path with a small capacitance through a transistor is that shown in figure 2-2 (b), which is in fact the pass-transistor logic style.

Figure 2-2: Two arrangements of a transistor

2.3 Frequency

Frequency is difficult to manage because the demand for higher computation rates naturally leads to higher frequencies. One way to reduce frequency while maintaining a high computation rate is to exploit parallelism at the system level. Another approach is to minimize wasted transitions at the logic level. Asynchronous design (see “Asynchronous logic” on page 27) has inherent properties which lead to achieving the above goal. At the system level, it is easier to exploit parallelism in the asynchronous conceptual framework. At the logic level, an asynchronous circuit only activates a particular part of the circuit for a required function and so does not dissipate any power in an idle part of the circuit.

2.4 Other sources of power consumption

In addition to dynamic switching power consumption, there are two other sources: static power consumption and short-circuit power consumption.

2.4.1 Static power consumption

There are three components that establish the amount of static power consumption:

- ❏ reverse junction leakage between diffusion regions and the substrate.
- ❏ subthreshold conduction leakage through the transistor channel.
- ❏ static bias current in some special circuits such as pseudo-nMOS gates.

The leakage current, including both reverse bias and subthreshold conduction factors, ranges between 0.1 nA and 0.5 nA per device at room temperature. Of course, a circuit with static bias current is not of interest in low power design.

2.4.2 Short-circuit power consumption

Short-circuit power consumption results from both overlapping current and spurious transitions. Overlapping current is a current pulse from the power supply to the ground during the transition since both nMOS and pMOS transistors are on for a short period of time. Spurious transitions is due to propagation delay differences between several logic paths in a circuit. Overlapping current can become significant in a “poorly-designed” CMOS circuit [16]. Overlapping current is dependent on the transistor gain factor and the input waveform rise and fall times. The slower the rise and fall time, the more the power consumption. Thus it is desirable to keep all edges fast if power consumption is a concern. Using buffers, where necessary, to keep rise and fall times low in order to reduce overlapping current is good design practice. It is customary that there should be no more than three (or at most four) transistors in series from the power supply or ground to any output node in order to ensure fast rise and fall times.

Asynchronous logic 3

Asynchronous logic, which is a dramatic departure from the conventional wisdom of synchronous designs, is an ideal paradigm for low power applications. Asynchronous designs focus on ordering a sequence of events to exploit the potential advantage of circuits and thereby achieve low power and high performance.

3.1 Delay models

The term asynchronous logic here refers to circuits that operate correctly without relying on global clocks for synchronization. Asynchronous circuits may be divided into four groups as shown in table 3-1 according to the assumptions made about gate and wire delays.

Table 3-1: Classification of asynchronous circuits

Models	Gate delay assumption	Wire delay assumption
Fundamental-mode model	*bounded delay*	*bounded delay*
Speed independent model	*unbounded delay*	*zero delay*
Delay insensitive model	*unbounded delay*	*unbounded delay*
Quasi delay insensitive model	*unbounded delay*	*isochronic fork*

3.1.1 Fundamental-mode model

The fundamental-mode model [17,18], also called the Huffman model, assumes that gate and wire delays are bounded. The Huffman model of an asynchronous circuit is shown in figure 3-1. This model decomposes the circuit into a combinational logic part and a feedback part. Let the maximum and minimum delays of the combinational logic be $max(D_c)$ and $min(D_c)$, and those of the feedback delays be $max(D_f)$ and $min(D_f)$. The shortest time and the longest time the circuit can take to change from the present state to a new state are $min(D_f+D_c)$ and $max(D_f+D_c)$, respectively.

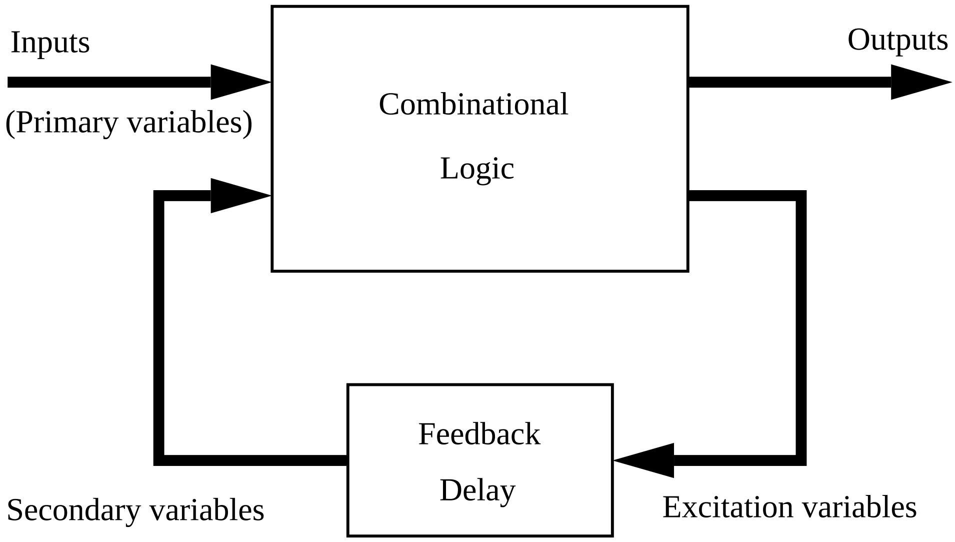


Figure 3-1: The Huffman model of an asynchronous circuit

So, two time intervals $min(D_f+D_c)$ and $max(D_f+D_c)$ can be defined [19,20]. If input changes are within the time interval $min(D_f+D_c)$, then they are considered as simultaneous, since the circuit perceives these changes in the present state. If input changes are beyond the time interval $max(D_f+D_c)$, then they are considered as distinct, since the circuit perceives these changes to take place in the new state. However, if input changes are between the time interval $min(D_f+D_c)$ and $max(D_f+D_c)$, the circuit behaviour is undefined. In other words, the inputs may steer the circuit into unexpected states that are not specified by the state graph.

With the assumption of bounded delays, the environment must be capable of controlling the timing of inputs such that input changes occur only when the circuit is in either the present state or the next state. This requirement means that the next input changes cannot occur until the entire circuit settles in to a new state.

3.1.2 Speed independent model

The speed independent mode [21,22], also called the Muller model, assumes that gate delays are unbounded but finite and wire delays are zero. The Muller model of an asynchronous circuit views a circuit and its environment as a set of gates $\{Z_1,\ldots,Z_n\}$, with each gate associated with a logic expression which describes its behaviour:

$$Z_i = \Phi_i\,(Z_1,\ldots,Z_n) \qquad i \in \{1,\ldots,n\}$$

A gate is combinational if its expression does not depend on the output signal, while it is sequential if its expression depends on the output signal. The variable Z_i is in a stable state if and only if $Z_i = \Phi_i\,(Z_1,\ldots, Z_n)$, and in an excited state otherwise. An excited state is a transition process from the present stable state to a new stable state. In this sense, the greater the number of excited variables, the greater the number of gates acting concurrently.

Speed independent circuits are circuits whose global behaviour is independent of the delays of all gates. A subset of speed independent circuits is comprised of the so-called “semi-modular” circuits whose global and local behaviours are both totally independent of the delays of every gate. A circuit is semi-modular if and only if every excited signal becomes stable by changing its value [22,23].

The assumption of unbounded gate delays deprives designers of the power to order the events by adjusting gate delays or even inserting wire delays. However, the events in a speed-independent circuit are well ordered by nature of the unique topological arrangement of the circuit.

Speed independent circuits are often viewed as working in a so-called “input-output” mode, similar to the wave pipeline concept in synchronous design. The input-output mode allows the inputs to change while the outputs are still being generated; this promises higher speed circuits than can be designed under fundamental mode.

3.1.3 Delay insensitive model

The delay insensitive model [23,24,25] assumes that both gate and wire delays are unbounded but finite. This model imposes the fewest restrictions on circuit delays. A delay insensitive circuit works correctly as long as all the gate and wire delays are finite. This property makes it particularly attractive for synthesis and testing.

However, it has been proved [26] that the class of delay insensitive circuits is very limited. In fact, it is very doubtful if such a circuit exists. Let’s take the simplest CMOS inverter circuit shown in figure 3-2 for example. It is obvious that a large delay difference from the input *In* to the pMOS and nMOS transistor will make the circuit fail, since both transistors may turn on at the same time. It must be assumed that the skew between the wire delays after the forking is negligible. In practice it is often assumed that the skew is less than one gate delay. This example shows that even the simplest CMOS inverter is delay sensitive.

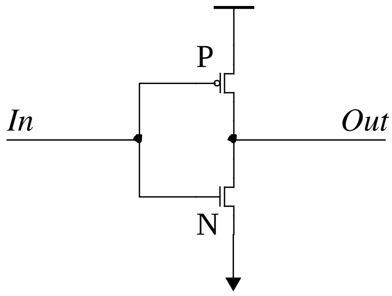


Figure 3-2: CMOS inverter

3.1.4 Quasi delay insensitive model

Various asynchronous models differentiate themselves in their choice of compromise to delay insensitivity. One reasonable compromise may be the introduction of the isochronic fork [27,28] to the delay insensitive model. An isochronic fork is a set of interconnecting wires where the delay difference between the branches is negligible compared to the gate delays. The quasi delay insensitive model adopts the delay insensitive assumptions that both gate and wire delays are unbounded, but augments this with isochronic forks. Based on this model, more practical circuits can be constructed.

It should be noted that the implementation of an isochronic fork is not just the local arrangement of the wires, because logic threshold differences between the gates may make the isochronic fork assumption unrealistic. This problem becomes serious when using conventional C-gates. Figure 3-3 shows the transistor implementation of a 2-input conventional C-gate. Consider the case of the output transition from low to high caused by the input *In1* transition from low to high (the input *In2* must be high). The high input *In2* closes the path from the power supply to the output. Therefore this C-gate loses the fighting mechanism between pull-up and pull-down during the transition to keep the logic threshold around *VDD/2,* and this results in a low logic threshold. The C-gate circuit shown in figure 3-4 does not have the above problem. It should also be noted that the isochronic fork assumption may require expensive delay fault testing [29].

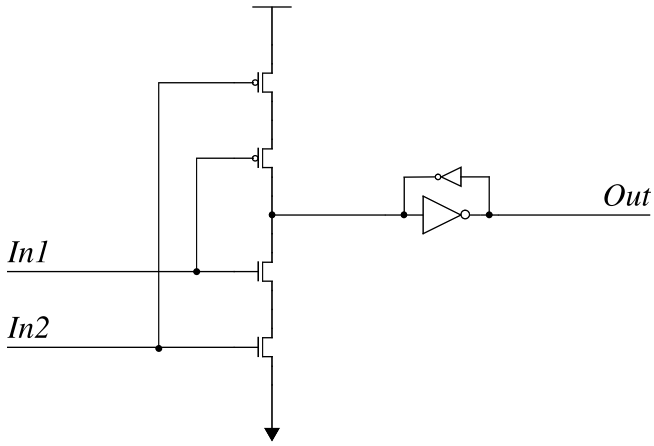


Figure 3-3: 2-input C-gate circuit

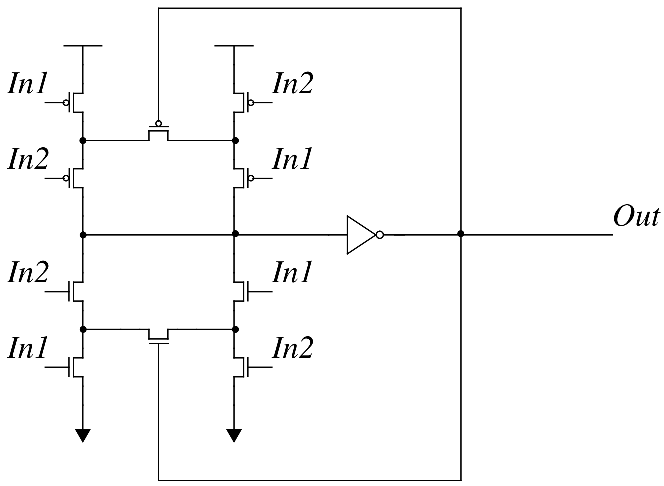


Figure 3-4: 2-input C-gate circuit with improved logic thresholds

3.2 Signalling protocols

With an unbounded model, there is no guarantee that the data from the sender will arrive at the receiver in any given time. The receiver should inform the sender whether or not it has received something by using some sort of handshaking mechanism. The sender that delivers a request must ensure that the request has been observed by the receiver for which it is intended. Two signalling protocols have been developed and are classified according to the number of transitions passing through handshaking signal wires.

3.2.1 Two-phase signalling protocol

The two-phase signalling protocol is illustrated in figure 3-5. The sender makes a transition or event by complementing the *Reques*t signal to indicate the availability of the data. Eventually the receiver senses the request, services the request and then issues a transition or event by complementing the *Acknowledge* signal to complete one data transfer. The sender may then transfer more data in the same way.

When communication time dominates computation time in a system, which is usually the case for communication between chips, the two-phase signalling protocol promises higher speed and lower power circuits because it allows data transmission without the redundancy of having an intermediate stage. Note that rising and falling edges are both significant and have the same meaning.

3.2.2 Four-phase signalling protocol

The four-phase signalling protocol is illustrated in figure 3-6. Initially, both *Request* and *Acknowledge* signals are at an inactive level, in this case a low level (one of several possibilities). The *Request* signal indicates the availability of the data by adopting its active level, in this case a high level. The *Acknowledge* signal indicates the acknowledgement of receiving the data by adopting its active level. Then the sender deactivates the *Request* signal, whereupon the receiver deactivates the *Acknowledge* signal. The latter two phases are inserted primarily to return the *Request* and *Acknowledge* signals to their initial level and are not used for timing information.

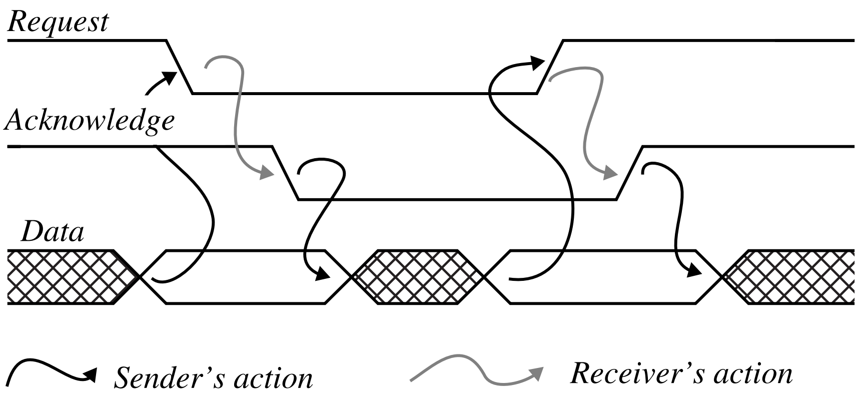


Figure 3-5: 2-phase signalling protocol

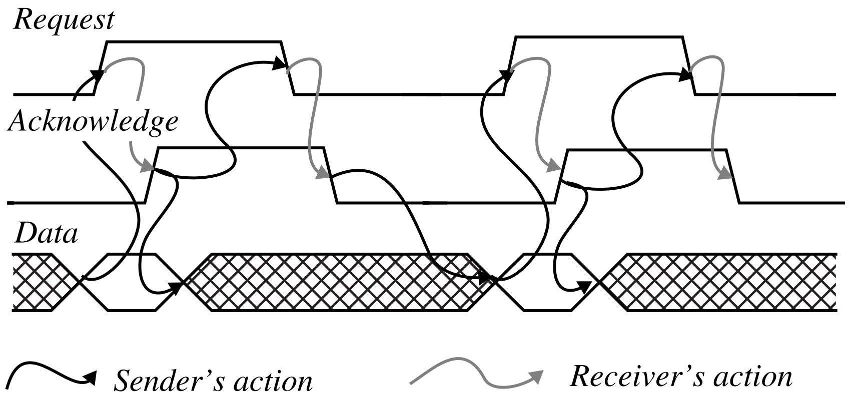


Figure 3-6: 4-phase signalling protocol

When computation time dominates communication time in a system, which is usually the case inside a chip, the four-phase signalling protocol appears to be more competitive because most practically efficient circuits are themselves inherently four-phase. The recovery process can often also be done concurrently with computations and thus "hidden" by using decoupling techniques [30,31].

3.3 Data representations

Due to there being no global clock signals to determine when a computation can start and when it is complete, an asynchronous circuit must be able to sense the arrival of data. In other words, data must be represented including some timing information. There are two approaches to representing data: bundled data and coded data.

3.3.1 Bundled data

Data bundling is a common technique for data transmission in asynchronous circuits. It uses a bundle of data wires associated with a single control wire that indicates the presence of valid data. The bundling constraint requires that the data wires and the control wire be constructed such that a transition on the control wire does not occur until the receiver sees the stable data.

The main advantage of bundled data is that standard datapath techniques or an existing cell library can be used. In general, any combinatorial function can be viewed and used as a bundled datapath element, together with corresponding matched control signals. The main disadvantage is that it is inherently delay bounded rather than speed independent. The bundling constraint must be analysed thoroughly and carefully to guarantee that it will be met in all the design corners. Bundled data can save a significant number of gates at the cost of having to meet the bundling constraints.

3.3.2 Coded data

Unlike Bundled data, coded data does not separate timing information from the data. There are many methods to encode data, such as one-hot code, Berger code and Knuth code [32]. A well known method is the dual-rail code that requires two wires to encode every bit of data value.

The main advantage of coded data is that it is possible for a circuit to be speed independent and make achieving average performance easily. The main disadvantage is a significant overhead in chip area and the number of wires. Also the dual-rail implementation of an asynchronous circuit is not power efficient.

3.4 Micropipelines

There are various asynchronous design styles based on combinations of delay models, signalling protocols, and data representations. One of the more practically efficient asynchronous design techniques, “Micropipelines”, was introduced in Ivan Sutherland’s Turing Award lecture of 1988 [33]. The micropipeline design technique views the circuit as being composed of a speed independent two-phase control circuit and a bounded delay datapath.

The basic micropipeline structure is shown in figure 3-7. It has combinational logic and event-controlled registers for the datapath, and C-gates for the control circuit. Initially event-controlled registers are all reset to the transparent state where data flows directly from the input to the output. A capture event at the *C* input of an event-controlled register, indicating the availability of new data, latches the data into the register and then causes a capture done event at the *Cd* output. A pass event at the *P* input of an event-controlled register, indicating the acknowledgement of receiving the data, switches the register back into the transparent mode and then causes a pass done event at the *Pd* output. In this way, every stage of the pipeline receives data from the preceding stage, performs the required functions and then transfers the result to the next stage. A request event terminates the sender's action, whereas an acknowledge event terminates the receiver's action. The delay introduced in the request event path must match the processing logic delay in order to preserve the data bundling convention.

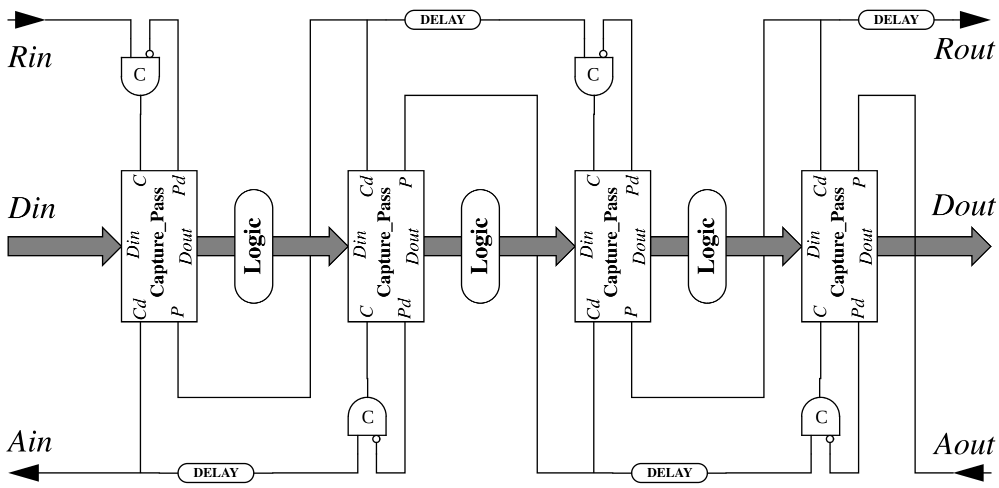


Figure 3-7: Basic micropipeline structure

3.5 Simplified C-gate

One main penalty of an asynchronous design is a considerable chip area overhead. Despite several dedicated optimizations, the resulting circuits are still quite large (e.g., the total area overhead of the asynchronous control is about 16% of the total core area of *AMULET1* [4]). One of the reasons is the fact that there are a high number of C-gates required in an asynchronous design (e.g., 98 C-gates required in *AMULET1* [4]). Since

C-gates are widely used as primitive logic elements in an asynchronous design, it raises the issue that there is considerable justification for designing efficient or simplified C-gate circuits for particular purposes.

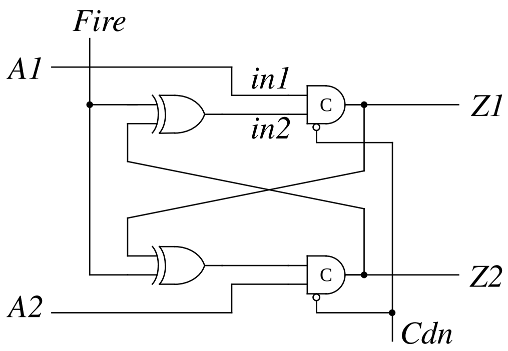


Figure 3-8: Decision-Wait element

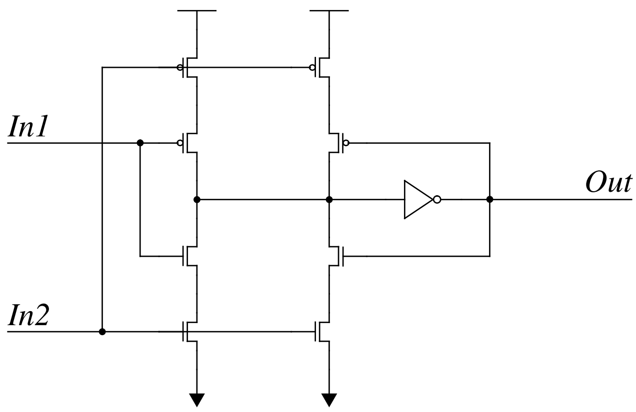


Figure 3-9: Simplified C-gate for the Decision-Wait element

Two implementations of a C-gate are presented in section 3.1.4 (see “Quasi delay insensitive model” on page 31). By taking into account the fact that some C-gates are just “rendezvous” places where designers know what it is going to happen (although do not know when), C-gates may be more or less optimized depending on their particular applications. Let’s take the Decision-Wait element shown in figure 3-8 for example. From the function of the Decision-Wait element, it is known that the transitions on the output of the C-gates always follow the transitions on the input *in2*. By exploiting this, a more efficient C-gate circuit can be derived as shown in figure 3-9. This C-gate has two main advantages. Firstly, it does not suffer from the low logic threshold problem, which is the case for conventional C-gates (see "Quasi delay insensitive model" on page 31). Secondly, it eliminates two "bridge" transistors in the circuit shown in figure 3-4 that make the layout difficult.

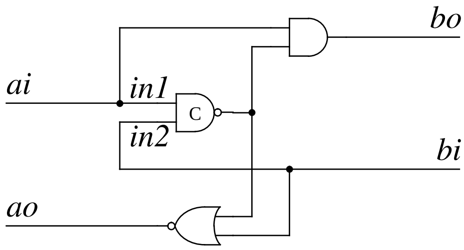


Figure 3-10: Q-element

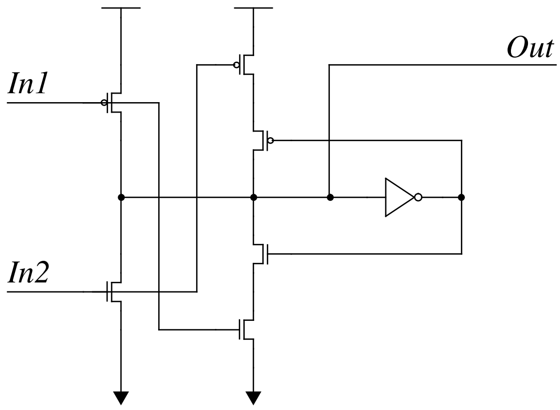


Figure 3-11: Simplified C-gate for the Q-element

Let's take another element, the Q-element (or sequencer in *TANGRAM* [34]) shown in figure 3-10, for example. From the function of the Q-element, it is known that the transition order on the inputs of the C-gate is *in1* (↑) ⇒*in2* (↑) ⇒ *in2* (↓) ⇒ *in1* (↓). By exploiting this, a more efficient C-gate can be implemented as shown in figure 3-11 [35]. Note that the output is taken from after the first inverting stage, which is based on the assumption of extended isochronic forks [36], in order to reduce one gate depth delay.

Pass-transistor logic

4

Techniques for low power design demand a low supply voltage and favour small transistor sizes. However, this naturally leads to degraded performance and reduced noise margins unless some compensating changes are made. How can an engineering compromise be made for achieving low power while maintaining performance and keeping noise margins acceptable? Conventionally, a CMOS circuit is designed not at the transistor level but at the gate level, in a way that usually creates redundancy. It may seem reasonable to design CMOS circuits at the transistor level rather than at the gate level of Boolean algebra. Several pass-transistor logic styles have thus been proposed to improve the performance of CMOS circuits due to their low input capacitance and high logic functionality. Furthermore, the analogue concept of a differential signal pair can be borrowed to fight off noise problems.

4.1 MOS transistor switches

MOS transistor switches are used as primitive logic elements in CMOS circuits. There are two types of MOS switches: nMOS and pMOS. It is well-known that an nMOS transistor is an almost perfect switch when passing a '*0*' signal, but an imperfect switch when passing a '*1*' signal. Firstly, passing a '*1*' signal is about 4 ~ 5 slower than passing a ‘*0*’ signal. Secondly, the threshold drop across the nMOS transistor when passing a ‘*1*’ signal results in reduced drive capability and hence slow operation. Finally, the degraded ‘*1*’ signal causes the pMOS transistor of the following stage not to turn fully off; this is important for low power design since direct-path static power consumption may not be negligible. On the other hand, a pMOS transistor is almost perfect for passing a ‘*1*’ signal but imperfect for passing a ‘*0*’ signal. It is worth noting that a degraded signal can be judiciously used for limiting voltage swings in some special circuits [37].

4.2 An example: XOR gate design

The XOR gate is the key unit for most arithmetic functions, such as comparators, adders, multipliers, and so on. Pass-transistor logics are attractive as few transistors are required to implement arithmetic functions due to their high logic functionality. By using the transmission gate approach [38,39], four different XOR gates can be implemented at the transistor level as shown in figure 4-1; all have simpler circuits than the conventional static gate versions.

4.2.1 CMOS transmission XOR gate

Figure 4-1 (a) shows an XOR gate implementation using CMOS transmission gates. This circuit requires four transistors, but the apparent drawbacks are the need for the true and complement input variables and unbalanced input capacitance.

4.2.2 Symmetric XOR gate

Figure 4-1 (b) shows a symmetric XOR gate implementation. Each input variable is connected to the gate of one transistor and the source/drain of another. This results in a balanced capacitance and equal delays for both inputs. However, the requirement for complementary signals still remains. There is difficulty in the layout design because of more interconnections among source, drain and gate of the transistors.

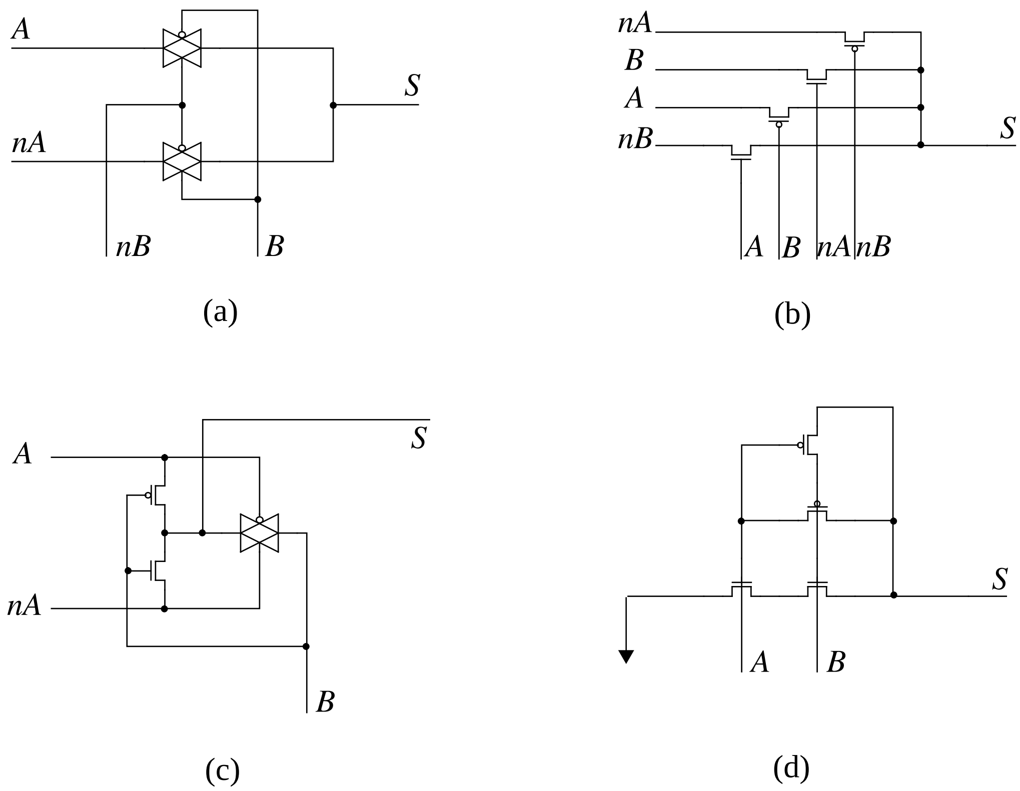


Figure 4-1: Four implementations of an XOR gate

4.2.3 Inverter-based XOR gate

Figure 4-1 (c) shows an XOR gate implementation based on the inverter configuration. The circuit needs only one complementary input but breaks the symmetry of the previous example. The operation of this XOR gate implementation is that when the *A* signal is high, the *A* and *nA* signal pair acts just as supply rails; when the *A* signal is low, the transmission gate is on, passing the *B* signal to the *S* output.

4.2.4 Tiny XOR gate

Figure 4-1 (d) shows an XOR gate implementation which rearranges the configuration of the pMOS and nMOS transistors. This circuit does not require complementary inputs and also retains a symmetric structure. However, a main drawback of the circuit is a poor '*0*' output value when *AB*='*00*'. The double path drive capability compensates somewhat for the speed degradation of the pMOS transistors due to both pMOS transistors being on in this case [40].

4.2.5 Comparisons

The simulated delays and power consumption of these four kinds of XOR gates under worst case conditions (3.3 volt supply voltage, slow-slow process corner, and 100˚C temperature) are given in table 4-1. The technology used here is *VLSI Technology's CMN5* CMOS process. The gate ratio in pMOS and nMOS is 2/1. The physical width/length size of pMOS and nMOS transistors are 3.0μm/0.6μm and 1.5μm/0.6μm, respectively.

Four options are now given in an XOR gate design. Where should one choose which choice? The circuits shown in figure 4-1 (a) and (b) seem the best options where complementary signals are available and the skew of complementary signals is negligible. However, complementary signals are troublesome under some circumstances. These complementary signals may be generated at some distance and heavily loaded with large capacitance, and therefore the skew of complementary signals may not be well under control. In these cases, the circuits shown in figure 4-1 (c) and (d) are of possible benefit. The point to be noted here is that there is not the skew problem of complementary signals in the circuit shown in figure 4-1 (c), although a complementary signal still exists. The circuit shown in figure 4-1 (d) should not be used for low supply voltage applications due to its degraded '*0*' signal. It is potentially of use in normal supply voltage applications or low supply voltage applications by pulling the degraded output signal back to full supply voltage.

Table 4-1: Simulated results of the four XOR gate implementations

(*30 fF load*)	input *AB* changes				Power @ 100 MHz
	00->10	*10->11*	*11->01*	*01->00*	
(a) with external complementary signals	0.29 ns	0.42 ns	0.26 ns	0.39 ns	127 µW
(a) with internally generated complementary signals	0.25 ns	0.72 ns	1.00 ns	1.13 ns	338 µW
(b) with external complementary signals	0.42 ns	0.47 ns	0.39 ns	0.44 ns	63 µW
(b) with internally generated complementary signals	0.46 ns	0.70 ns	0.73 ns	0.50 ns	168 µW
(c) with external complementary signal	0.57 ns	0.53 ns	0.19 ns	0.34 ns	90 µW
(c) with internally generated complementary signal	0.74 ns	0.64 ns	0.44 ns	0.37 ns	189 µW
(d)	0.28 ns	0.72 ns	0.42 ns	0.52 ns	48 µW

4.3 Differential pass-transistor logic

Reducing the supply voltage is often the first choice for power reduction due to its quadratic dependence (see "Supply voltage" on page 19). However, it is important to take into account the problem of reduced noise margins. A small signal swing resulting from a low supply voltage makes it difficult to achieve reasonable noise margins that can normally be achieved at a relative large signal swing as in the case of conventional CMOS designs.

In order to overcome the above problem, an approach [41,42] was proposed which uses differential signal pairs, borrowing from the conventional wisdom of how analogue designs handle their more serious noise problems. The key idea of differential pass-transistor logic (DPTL) is its immunity to common-mode noise (noise that is picked up by both nodes of a differential signal pair). Therefore acceptable noise margins can be achieved from the structural symmetry even at a low supply voltage.

TSPC methodology

5

Global clock generation and distribution is becoming responsible for a significant portion of the total power consumption in a synchronous CMOS circuit. In *DEC's Alpha* chip [43], for example, the clock power is about 40% of the total power consumption. Although, in an asynchronous circuit, global clocks are abandoned and replaced by local handshaking mechanisms, the issue of register enabling still remains. The difference in terms of register enabling between synchronous and asynchronous datapath circuits (using a bounded delay model) is just whether global or local enabling signals are used. The shift from a conventional non-overlapping multiphase clocking scheme to the true single-phase clocking methodology is a way to minimize the number of enabling nets as well as the total enabling transistor capacitance, both of these factors contributing to low power and also to high performance.

5.1 Non-overlapping two-phase clocking

The non-overlapping two-phase clocking scheme [8] shown in figure 5-1 is often applied to CMOS systems, since it is recognized as a safe way to avoid the races that can result from clock skew and/or clock slope problems. However, this safety comes with a price. Firstly, there are two dead periods α and β in figure 5-1 that act as buffers to accommodate clock skew and clock slope in order to avoid both latches being open at the same time. These two dead periods become unacceptable for a circuit with a very high frequency clock since they may be of the same order as the working periods *t1* or *t2*. Secondly, this two-phase scheme is in fact pseudo two-phase clocking for CMOS circuits because there is the need for another two complementary signals. That means much silicon area is needed to distribute these four clock signals.

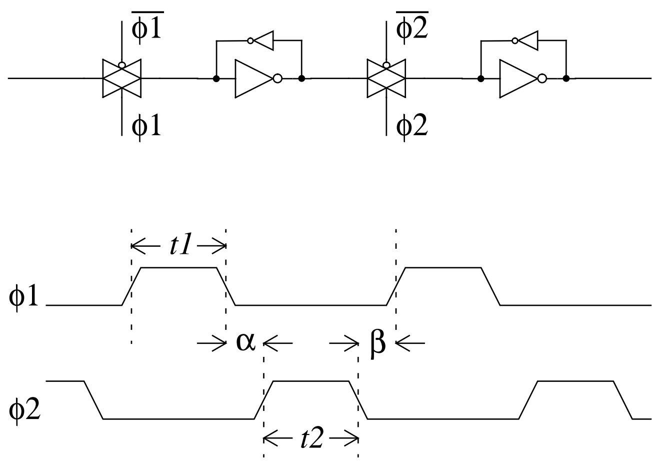


Figure 5-1: Non-overlapping two-phase clocking

5.2 Single-phase clocking

In order to avoid the penalty of dead periods, the single-phase clocking scheme (pseudo single-phase clocking for CMOS circuits) shown in figure 5-2 may be employed with extreme care. The minimum delay of combinational circuits such as an inverter could results in a data race problem. There are two engineering approaches to single-phase clocking designs. Firstly, the complementary signal is generated by using a local inverter and the skew of the complementary signal pair is thus controlled locally and can be precise as required. This is the normal way used in gate arrays and standard cell designs.

Secondly, clock signals can be generated and balanced using global clock buffers. Close attention must be paid to the skew of the complementary signal pair since these signals traverse the chip and may be heavily loaded. This is the normal approach used in custom designs that favour a small register configuration. However, it is still likely that the resulting circuit will end up with a “poor” design since the safety of the circuit is very sensitive to the clock slope [44].

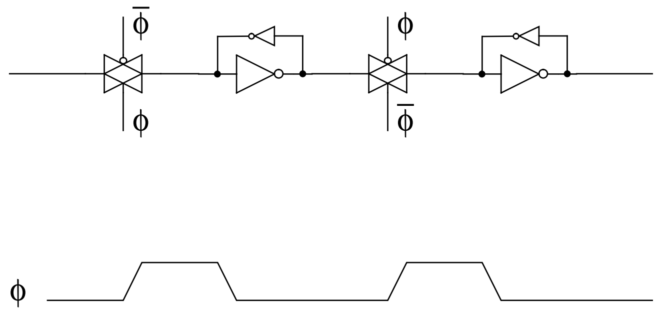


Figure 5-2: Single-phase clocking

5.3 True single-phase clocking

The skew problem of the complementary signal pair is removed if the true single-phase clocking (TSPC) methodology [45] is adopted, since the complementary signal is not necessary. Figure 5-3 shows the TSPC latch versions as proposed originally by Yuan and Svensson. The circuit shown in figure 5-3 (a) is low level sensitive, and the circuit shown in figure 5-3 (b) is high level sensitive. These circuits are safe and robust since the slope limit in terms of rise and fall time is about 20 minimum size inverter delays [46]. It is worth noting that the “skew” caused by the clock delay itself between different logic components will not be negligible in a large system. This problem can be minimized by distributing the clock in the opposite direction to the flow of data. By using the circuits shown in figure 5-3, only one global enabling net is needed and the total transistor load attached to the enabling net represents the minimum number of transistors and the minimum transistor size possible. This leads to a significant power saving over the non-overlapping multiphase clock scheme. Having only one enabling net to route also results in a silicon area reduction.

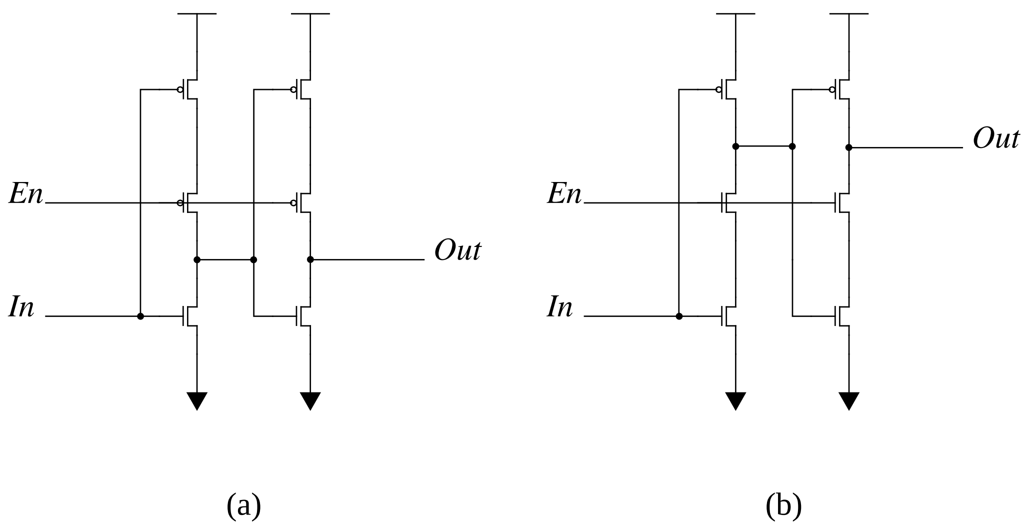


Figure 5-3: TSPC latches

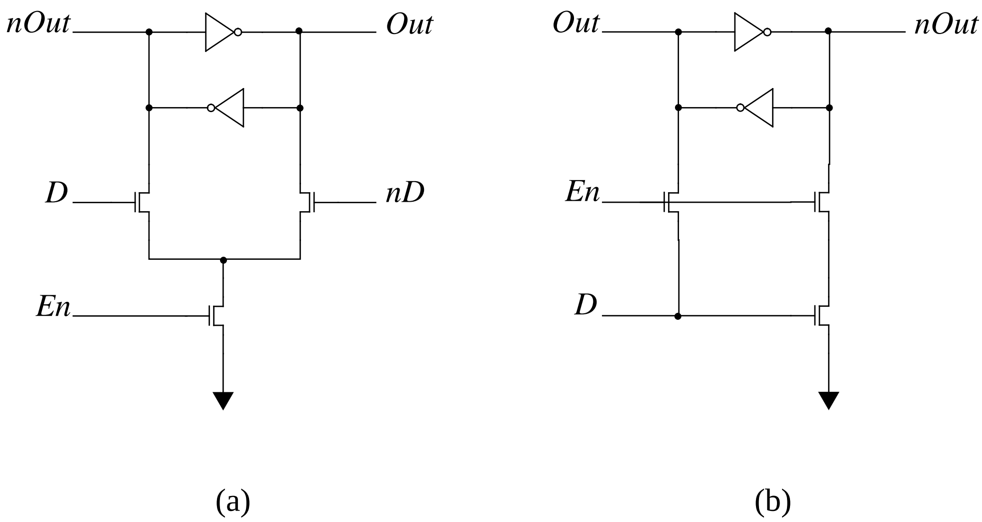


Figure 5-4: TSPC memory structures

Figure 5-4 shows TSPC memory structures that are often used in register bank designs. The circuit shown in figure 5-4 (a) is the complementary input signal version, and the circuit shown in figure 5-4 (b) is the single-ended input signal version.

By combining two level-sensitive latches (one low sensitive and one high sensitive), an edge-triggered register can be implemented. It is interesting that edge-triggered registers can be optimized if latches shown in figure 5-3 are to be used (see “Register design” on page 74).

5.4 Low limit of clock frequency

There is a minimum operating clock frequency for any dynamic memory elements. The low limit of operating clock frequency is usually determined by the leakage current (see “Static power consumption” on page 25). Assuming a supply voltage of 3.3 volt, a leakage current of 0.5 nA and a storage capacitance of 30 *f*F, then the low limit of operating clock frequency is $\frac{2 \cdot \Delta I}{C \cdot \Delta V} \approx 10$ KHz, which is suitable for most VLSI applications.

6 Multiplication algorithms

The general principle by which computers carry out multiplication is quite simple. The multiplication of two 1-bit binary numbers is even simpler than addition since there is no need for carry to propagate. Multiplication is just performed by sequences of additions and shifts. However, one of the challenging problems in multiplication is deciding how to deal with the task of summing together partial products with different weights. Another challenging task is determining how to handle the most significant bit (sign bit) in signed number multiplication.

6.1 Basic algorithms

Consider the multiplication of two binary numbers using the ordinary paper-and-pencil method. Figure 6-1 illustrates a convenient dot representation [47] of the multiplication of two 8-bit binary numbers. Each horizontal row of dots represents zero or a single copy of the multiplicand dependent on a particular bit of the multiplier that broadcasts through the whole row. Roughly speaking, the number of dots in the dot diagram reflects the amount of hardware (in parallel multipliers) or time (in serial multiplier) required to carry out multiplication. The height of the dot diagram relates to the latency of an implementation of the algorithm. In the paper-and-pencil method, all the partial products are generated and then added up simultaneously, proceeding column-wise from right to left.

Although conceptually simple, direct hardware implementations of such a method would lead to very inefficient designs because of the asymmetry between different columns. There exists some degree of symmetry between different rows in terms of the number of dots except their different weights. It is therefore desirable to proceed row-wise from top to bottom sequentially or simultaneously for VLSI implementations.

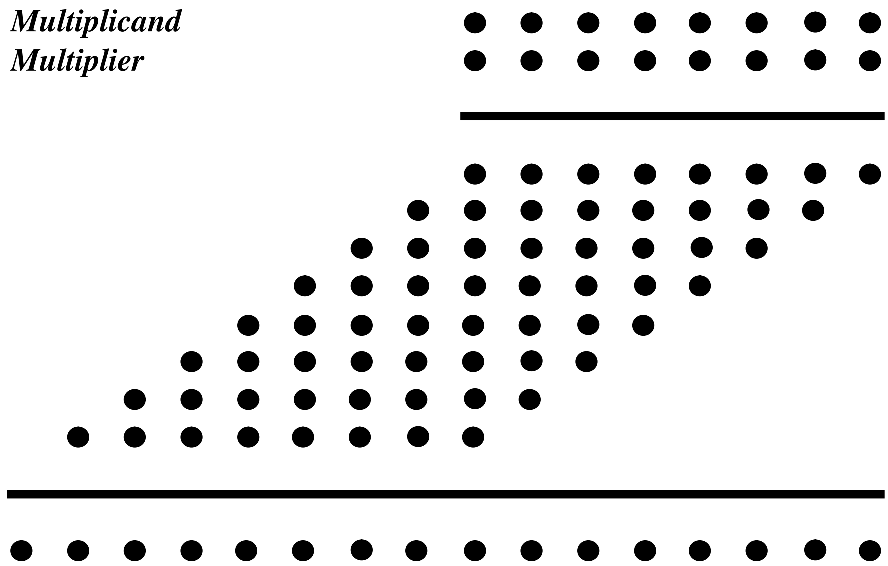


Figure 6-1: Dot representation of 8 × 8-bit multiplication

The multiplication algorithm resulting from a straightforward application of the paper-and-pencil method is essentially a process of repetitive addition by conditionally adding the multiplicand to a running partial product. Therefore there are two basic approaches to improving the speed of the multiplication operation: making each addition faster, and reducing the number of additions required. An additional trick is to use an early termination scheme.

6.2 Making each addition faster

6.2.1 Redundant binary addition

In conventional number representations such as an unsigned and a signed number, the addition of two binary numbers requires a computation time at least proportional to the logarithm of the size of the adder because of carry propagation. Limiting or eliminating carry propagation is the key to making addition faster. Redundant binary representations [48] can be used to limit carry propagation to one bit in order to achieve fast addition independent of the word length of the operands.

The radix-2 redundant binary representation has a digit set {$\bar{1}$, 0, 1}, where $\bar{1}$ denotes -1. An n-bit redundant binary number Y=[y_{n-1},…,y_0] has the value $\sum_{i=0}^{n-1} y_i \times 2^i$, where y_i belongs to {$\bar{1}$, 0, 1}. It is similar to an unsigned binary representation except that y_i can be $\bar{1}$.

Table 6-1: Example of redundant binary addition

Augend (105)	0 1 0 $\bar{1}$ 0 1 0 1 $\bar{1}$
Addend (110)	0 1 $\bar{1}$ 1 0 1 1 1 0
Intermediate sum	0 0 1 0 0 0 $\bar{1}$ 0 1
Intermediate carry	0 1 $\bar{1}$ 0 0 1 1 1 $\bar{1}$
Shift intermediate carry	1 $\bar{1}$ 0 0 1 1 1 $\bar{1}$ 0
Final result (215)	1 $\bar{1}$ 1 0 1 1 0 $\bar{1}$ 1

The main property of redundant binary representations is that a number can be represented in several ways. For example [0101], [011$\bar{1}$], [1$\bar{1}$01], [1$\bar{1}$1$\bar{1}$] and [10$\bar{1}\bar{1}$] all represent “5”. This redundant information offers the possibility of limiting or eliminating carry propagation that is required for efficient addition. As a result, the addition of two redundant binary numbers can be performed in a constant time independent of the word length of operands.

Redundant binary addition without carry propagation is carried out in two steps [49] as shown in table 6-1. Firstly, the intermediate sum S_i and the intermediate carry C_i are generated at each bit position, satisfying the equation $X_i + Y_i = S_i + 2C_i$, where X_i and Y_i are two operands. Secondly, the final result is obtained by adding the intermediate sum S_i and the intermediate carry C_{i-1} from the next lower order position. The key idea to avoid carry propagation is to set the intermediate sum to 1 when there is a negative carry from the next lower order position and to set the intermediate sum to $\overline{1}$ when there is a positive carry from the next lower order position. In other words, the intermediate sum S_i and the intermediate carry C_{i-1} cannot both be 1 or $\overline{1}$ at the same time. In this way, there is no need to know the lower order intermediate carry C_{i-1} to obtain the intermediate carry C_i. Thus it is possible to design an adder whose computation time is independent of the size of the adder.

6.2.2 Carry-save addition

As we know, multiplication is essentially a process of repetitive addition by conditionally adding the multiplicand to a running partial product. The time necessary for carry propagation imposes the ultimate limit on the speed of addition and thus multiplication. All the techniques for faster adders are also available to speed up multiplication. However, multiplication is a special case of repetitive addition in which the result of the intermediate partial products is not of interest. So it is not strictly necessary to propagate the carries during every addition cycle. Instead, the carries generated during the current addition cycle may be saved and used again on the next addition cycle with an appropriate shift [50]. In this way, a partial sum and a partial carry (saved carries) together presents the partial product. Thus each addition cycle will need only the time required for a 1-bit addition since all the carry bits are just simply considered as outputs instead of as internal intermediate signals. Evidently, carry-save addition can be used for all but the last addition cycle. Only on the last cycle need the carries be propagated to completion instead of being saved.

Table 6-2: Multiplication of 6 × 7 with carry-save addition

initial partial sum	0000
initial partial carry	0000
add 1st multiplicand	0110
1st partial sum	0110
1st partial carry	0000
shift partial sum right	00110
add 2nd multiplicand	0110
add 1st partial carry	0000
2nd partial sum	01010
2nd partial carry	0010
shift partial sum right	001010
add 3rd multiplicand	0110
add 2nd partial carry	0010
3rd partial sum	011010
3rd partial carry	0010
shift partial sum right	0011010
propagate final carries	0010
final result (42)	0101010

An example is shown in table 6-2. It is worth noting that since the partial product has now been replaced by a partial sum and a partial carry, carry-save addition in effect employs a redundant concept. The difference is that carry-save addition uses the digit set {0, 1, 2, 3} instead of {$\bar{1}$, 0, 1} since the combination of a partial sum and a partial carry results in four types of unsigned number representations.

6.3 Reducing the number of additions required

6.3.1 Multi-operand addition

The additions mentioned above are two or three operand operations. One obvious way to reduce the number of additions for speeding up multiplication is to use multi-operand additions (more than three operands). There are several ways to compose a multi-operand adder from half or full adders.

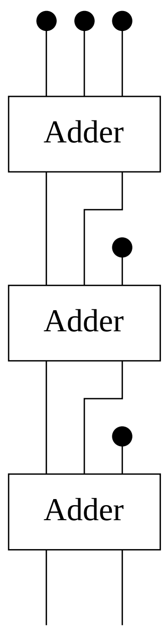


Figure 6-2: Linear array

Linear array

A linear array shown in figure 6-2 has a regular cellular array structure of unit cells and is favourable for VLSI implementations. However, its computation time is linearly proportional to the size of the multiplier.

Wallace tree

A Wallace tree [51] shown in figure 6-3 is well known for its optimal computation time. However, its implementation is often too expensive to justify the speed obtained.

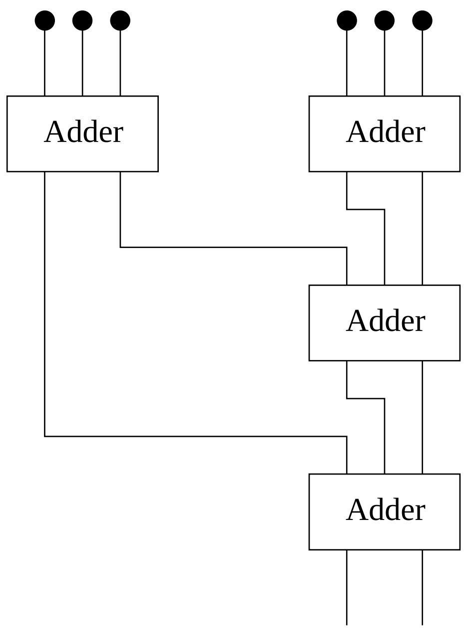
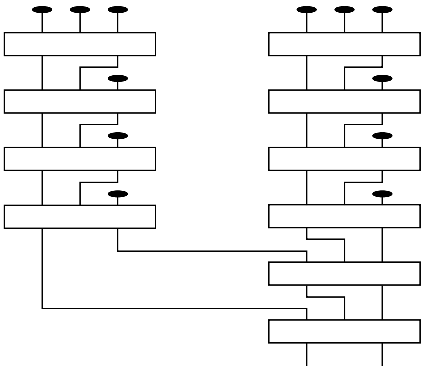


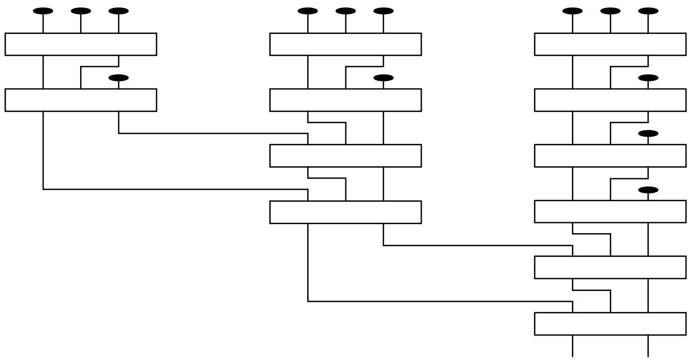
Figure 6-3: Wallace tree

Other structures

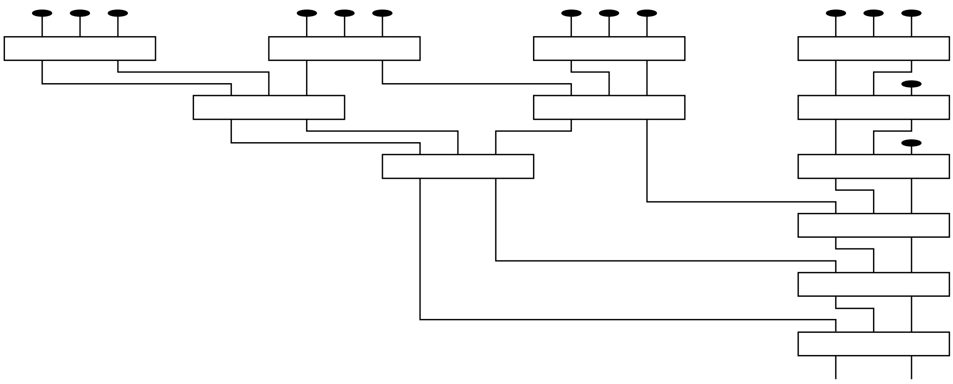
The following array or tree structures shown in figure 6-4 have been proposed by trading speed for regularity.



(a) Double linear array



(b) Zuras & McAllister tree



(c) Overturned Stairs tree

Figure 6-4: Other structures

- ❏ Double linear array [52].
- ❏ Zuras & McAllister tree [53].
- ❏ Overturned Stairs tree [54].

6.3.2 4-2 counter

As seen from the above, all the array or tree structures are composed from full adders (i.e., 3-2 counters). A 3-2 counter takes three inputs and produces two outputs. This 3:2 input versus output ratio makes it impossible to build a completely regular tree structure. It leads to look at a counter that has a 2:1 input versus output ratio, which is similar to a binary tree. A so-called 4-2 counter was thus proposed in [55]. Figure 6-5 shows a 4-2 counter that is constructed from two 3-2 counters. By considering the *Cin* and *Cout* signals as intermediate signals instead of as the input and output, this circuit takes four inputs and produces two output, which is completely symmetric.

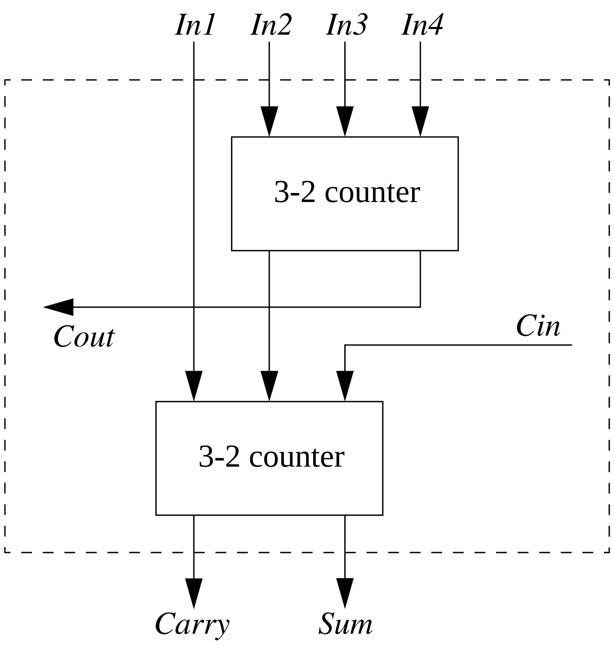


Figure 6-5: 4-2 counter structure

Since the *Cout* signal is independent on the *Cin* signal, there does not exist a propagation problem if several 4-2 counters are abutted into the same row, which is the key idea behind 4-2 counters. A 4-2 counter is similar to but different from a 5-3 counter. A 5-3 counter has three different weights for the outputs, while a 4-2 counter has two different weights for the outputs.

6.3.3 Encoding techniques

Booth's algorithm

The most obvious way to reduce the number of additions is to skip over any contiguous string of 1s and 0s in the multiplier, rather than form a partial product for each bit. The original Booth' algorithm [56] is based on this idea.

In two's complement form, an n-bit binary word $A = a_{n-1}\, a_{n-2} \ldots a_1\, a_0$ can be expressed as:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

The principle of the original Booth's algorithm is to rewrite this number as:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i = \sum_{i=0}^{n-1} 2^i (a_{i-1} - a_i) = \sum_{i=0}^{n-1} 2^i k_i$$

where a_{-1} is a dummy bit that is equal to zero, and k_i (= $a_{i-1} - a_i$) belongs to the digit set of {-1, 0, +1}. In this way, the original Booth's algorithm may be viewed as a conversion of the multiplier representation from a conventional binary code into a redundant binary code. The redundant binary code is {-1, 0, 1}, and the radix is two. Note that a redundant binary addition or carry-save addition scheme encodes the multiplicand to a redundant binary code, while the original Booth's algorithm encodes the multiplier to a redundant binary code.

Modified Booth's algorithm

A slightly different algorithm, called the modified Booth's algorithm [57], considers groups of bits of the multiplier, rather than skipping over arbitrarily long strings. The multiplier bits are divided into two-bit groups. Three bits are scanned at a time, two bits from the present group and the third bit being the higher-order bit of the lower-order group.

The principle of the modified Booth's algorithm is to rearrange a number as:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i = \sum_{i=0}^{\left|\frac{n}{2}\right|-1} 2^{2i}(a_{2i-1} + a_{2i} - 2a_{2i+1}) = \sum_{i=0}^{\left|\frac{n}{2}\right|-1} 2^{2i}k_i$$

where a_{-1} is a dummy bit that is equal to zero, and k_i $(= a_{2i-1} + a_{2i} - 2a_{2i+1})$ belongs to the digit set of {-2, -1, 0, +1, +2}.

Table 6-3: Modified Booth algorithm

Group	Action	Reasoning by string property
0 0 0	0	no string of 1s
0 0 1	+1	end of string
0 1 0	+1	isolated 1
0 1 1	+2	end of string
1 0 0	-2	beginning of string
1 0 1	-1	isolated 0
1 1 0	-1	beginning of string
1 1 1	0	centre of string

In this way, the modified Booth's algorithm may be viewed as a conversion of the multiplier representation from a conventional binary code into a redundant binary code. The redundant binary code is {-2, -1, 0, 1, 2}, and the radix is four. The modified Booth's algorithm is described in table 6-3. The modified Booth's algorithm halves the number of additions that have to be performed, therefore speeding up multiplication

6.3.4 Early termination

A trick that may be used to further reduce the number of additions is to check in each addition cycle if the shifted multiplier contains only 1s or 0s, or the shifted multiplicand contains only 0s, and if so, to terminate the multiplication procedure early [58]. An example is shown in table 6-4.

Table 6-4: Multiplication of 31 × 2 with early termination

multiplicand (31)		11111
multiplier (2)		00010
initial partial result		00000
shift multiplier right		00001(0)
add zero		00000
1st partial result		00000
shift partial result right		000000
shift multiplier right	➠	00000(1)
add multiplicand		11111
2nd partial result	terminate here ☞	111110
final result (62)		111110

Multiplier organization 7

The *AMULET2* microprocessor is an asynchronous implementation of the *ARM* microprocessor, and is object code compatible with the 32-bit *ARM6* microprocessor. Therefore, the decisions in the development of the *AMULET2* multiplier obviously depend on some constraints defined by the architecture of the *ARM* microprocessor.

7.1 The *AMULET2* processor

Figure 7-1 shows the internal organization of the *AMULET2* microprocessor. Four major function blocks operate concurrently and asynchronously to each other, and synchronize only when it is necessary to exchange information across their respective interfaces.

- ❏ **Address interface (A)** includes an autonomous non-deterministic instruction prefetch unit with support for exceptions and virtual memory [4].
- ❏ **Register bank (R)** has a novel register locking mechanism that can solve register dependencies without arbitration [59].
- ❏ **Data interface (D)** manages the flow of data between the *AMULET2* microprocessor and its memory system.
- ❏ **Execution unit (E)** comprises a multiplier, a barrel shifter and an *ALU* [60].

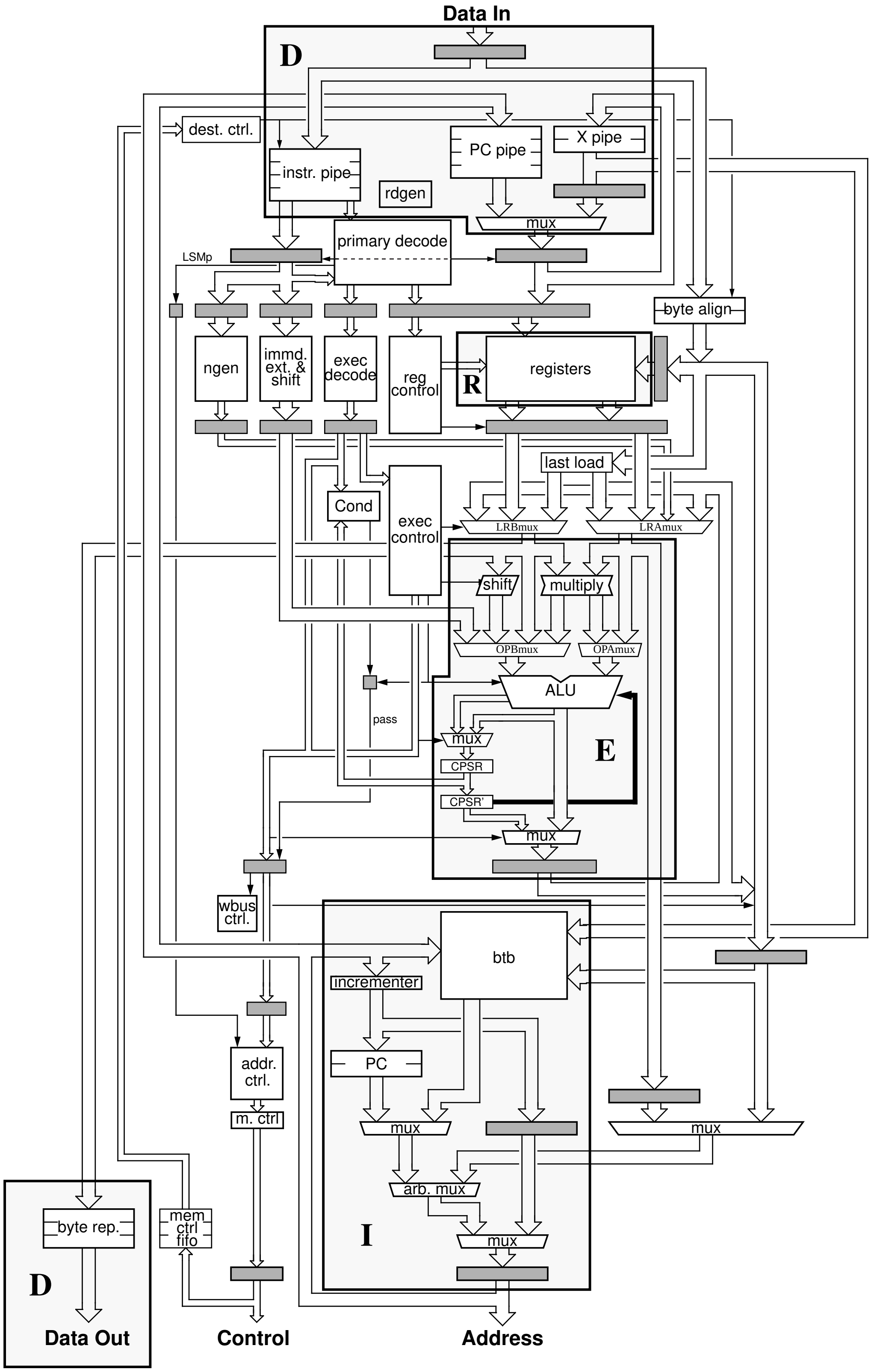


Figure 7-1: Processor organization

Since a small percentage of instructions take advantage of the multiply operation, the *AMULET2* multiplier contains an internal bypass mechanism for other more frequent instructions, and is thus not activated for most operations.

7.2 System issues

7.2.1 Multiplier

The goal of the multiplication support in the *AMULET2* microprocessor is to provide a reasonable amount of performance at a low power consumption with a low hardware cost. Figure 7-2 shows the block diagram of the *AMULET2* multiplier. With valid operands on buses *A* and *B*, a multiply request *Mle* signal will initialize the multiplication process, and produce two final partial results. With two final valid partial results on buses *C* and *D*, a request *Mld* signal to the following *ALU* will start the addition process, and produce the final result of multiplying the two operands.

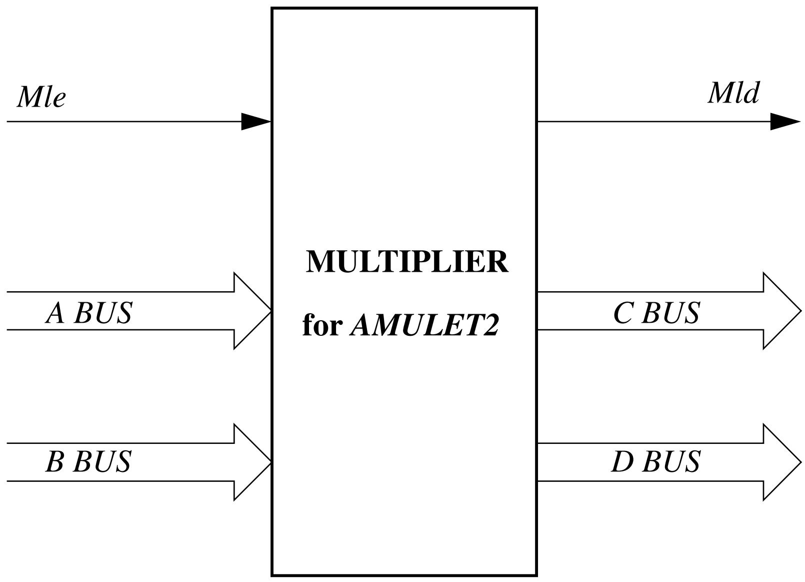


Figure 7-2: Block diagram of the *AMULET2* multiplier

7.2.2 Full 32-bit operation

In the *ARM* processor, the arguments and the result of a multiply operation are all 32 bit. The result of multiplying two 32-bit numbers is the least significant 32 bits of the 64-bit product [5]. The following arguments support this style:

- ❑ the least significant *32*-bit result is adequate for many *DSP* applications.
- ❑ it is sufficient for calculating the address of an element of a multi-dimensional array since all addresses are less than 32 bits [5].
- ❑ in high-level languages, the arguments and the result of an operation are usually the same type.

One benefit from this style is that both unsigned and signed (*2*'s complement) number multiplication have the same least significant 32-bit result.

7.2.3 Overflow

If the result does not fit in *32* bits, should multiplication set a bit or trap on overflow? The *AMULET2* microprocessor does nothing on overflow and leaves it to software, either to constrain the operands to ensure there is no overflow, or to perform explicit checks.

7.3 Implementation considerations

7.3.1 Iterative structure

Multipliers may be classified according to the format how operands are accessed, namely, serial, serial/parallel, and parallel forms. In general, the choice is based on factors such as latency, throughput, chip area, and numerical resolution. In the

AMULET2 multiplier design, latency and chip area are considered the most important parameters to be minimized, whereas throughput and numerical resolution are relatively less important. Although it is possible to implement a fast parallel multiplier in the *AMULET2* chip, a significant amount of hardware would be needed. Therefore, a serial/parallel multiplier (an iterative structure) is what the *AMULET2* microprocessor needs, since it is much faster than a serial multiplier but also less area-hungry than a parallel multiplier. The small amount of hardware required for a serial/parallel multiplier also means that more functions can be placed on the same chip or the cost is lower.

7.3.2 Bus-oriented datapath

The layout cells for the datapath of the *AMULET2* multiplier should be fixed in the constant bit slice (see “Full-custom format” on page 86) as shown in figure 7-3. The number of tracks available for buses is nine per bit slice in the *AMULET2* datapath. Only four tracks are for local routing in the multiplier region, since the other five tracks are used for through buses. This bus-oriented datapath structure forces some circuits for the *AMULET2* multiplier to be single-ended rather than complementary signals, and thus has a significant effect on circuit design (see “New proposed 4-2 counter” on page 79).

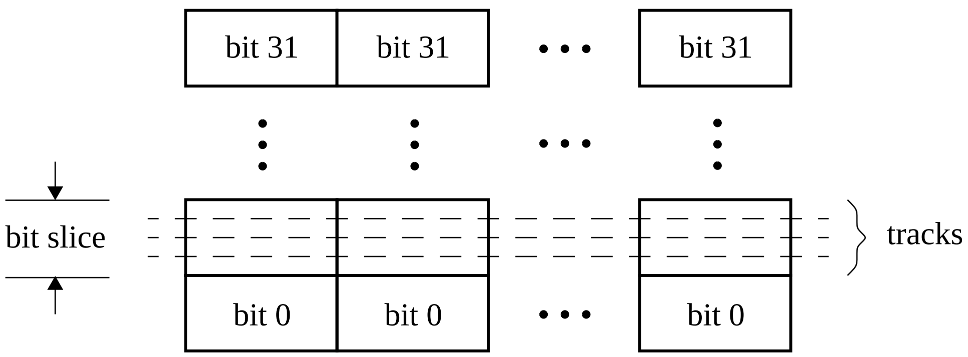


Figure 7-3: Bus-oriented datapath structure

7.4 Multiplier organization

The internal organization of the *AMULET2* multiplier is shown in figure 7-4. It is partitioned into two major blocks. The datapath is responsible for performing multiplication, and also contains the interface to the control logic and the data buses. The control circuit is responsible for generating control signals to the datapath and communicating with other parts of the *AMULET2* microprocessor. The execution of a multiply instruction consists of four primary steps:

1. with valid operands on the buses *A* and *B*, a multiply request *Mle* signal initializes the *AMULET2* multiplier by latching two operands into the *A* and *B* registers, and resetting the *P1*, *P2*, *S*, and C registers.
2 check if the shifted *A* register contains only ones or zeroes; if so, set a bit on termination, otherwise reset this termination bit.
3 if the termination bit is not set, perform a one cycle operation, and then repeat step 2.
4 if the termination bit is set, produce request *Mld* signal, indicating two final valid partial results available on the buses *C* and *D*.

7.5 Registers

The *AMULET2* multiplier provides two operand registers (*A* and *B*), two datapath pipeline registers (*P1* and *P2*), and two result register (*S* and *C*), all in 32 bits; and one 7-bit control pipeline register (*Q*). These registers are responsible for providing operands, pipelining operations, and retaining results. Note that the initialization of the *Q* register is dependent on the four least significant bits of the *A* register.

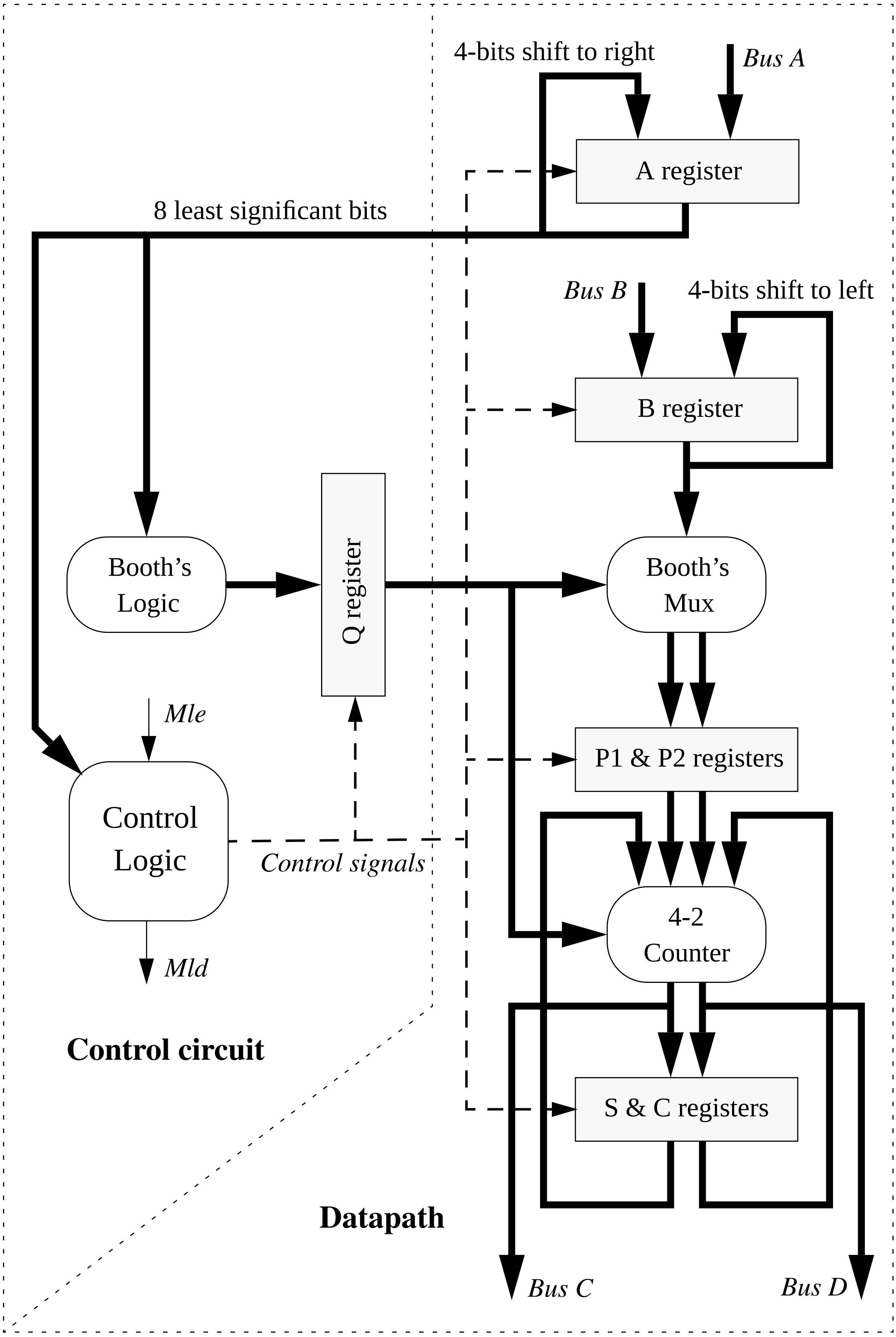


Figure 7-4: Multiplier organization

The *A* register shifts data four bits to the right at a time, i.e., towards the least significant bit, with sign extension. The *B* register shifts data four bits to the left, i.e., towards the most significant bit, with zeros filling the least significant bits. By shifting two operands in opposite directions, the result will be bit position aligned [58].

7.6 Booth's logic & Booth-Mux

Two modified Booth's schemes are shown in table 7-1. The decision that the second encoding method was chosen in the *AMULET2* multiplier design is discussed in section 8.2 (see "Booth-Mux cell design" on page 77).

Table 7-1: Two encoding methods

Bit position			Actions	Standard encoding					second encoding		
i+1	*i*	*i-1*		*YZ*	*YP1*	*YP2*	*YN1*	*YN2*	*N*	*S*	*Z*
0	0	0	0	1	0	0	0	0	0	0	1
0	0	1	+ M	0	1	0	0	0	0	0	0
0	1	0	+ M	0	1	0	0	0	0	0	0
0	1	1	+ 2M	0	0	1	0	0	0	1	0
1	0	0	- 2M	0	0	0	0	1	1	1	0
1	0	1	- M	0	0	0	1	0	1	0	0
1	1	0	- M	0	0	0	1	0	1	0	0
1	1	1	0	1	0	0	0	0	1	0	1

7.7 4-2 counter

4-2 counters are widely used in new multiplier designs in recent years [61,62,63,64] due to their regular and symmetric structures. A 4-2 counter can be constructed from two 3-2 counters as seen in section 6.3.2 (see "4-2 counter" on page 60). However, this circuit with four XOR gate delays is not very efficient. From the truth table for 4-2 counters as shown in table 7-2, a more efficient circuit with a small delay can be constructed (see “New proposed 4-2 counter” on page 79).

Table 7-2: Truth table for 4-2 counters

The number of inputs’s high	*Cin*	*Cout*	*Sum*	*Carry*
0	0	0	0	0
1	0	0	1	0
2	0	1/0(note)	0	0/1(note)
3	0	1	1	0
4	0	1	0	1
0	1	0	1	0
1	1	0	0	1
2	1	0/1(note)	1	1/0(note)
3	1	1	0	1
4	1	1	1	1
(note) — either *Cout* or *Carry* may be one or zero, but not both.				

7.8 Control logic

Figure 7-5 shows the self-timed control logic of the *AMULET2* multiplier. The *Q*-element is well known in asynchronous designs whose function is to execute four-phase handshakes along the channel *b*, and then along the channel *a* when activated along the channel *a*. The asymmetric C-gate is used to achieve very efficient four-phase circuits (see details in [30]). The functions of the signals shown in figure 7-5 are as following:

- the *Reset* signal is for resetting the *P1*, *P2*, *S*, and *C* registers.
- the *Lt* and *nLt* signals are for loading data into the *A* and *B* registers.

- the *En* signal is for controlling addition cycle.
- the *Terminate* signal is for representing early termination information.
- the *Mle* signal is for activating a multiply operation.
- the *Mld* signal is for indicating the completion of the multiplication process.

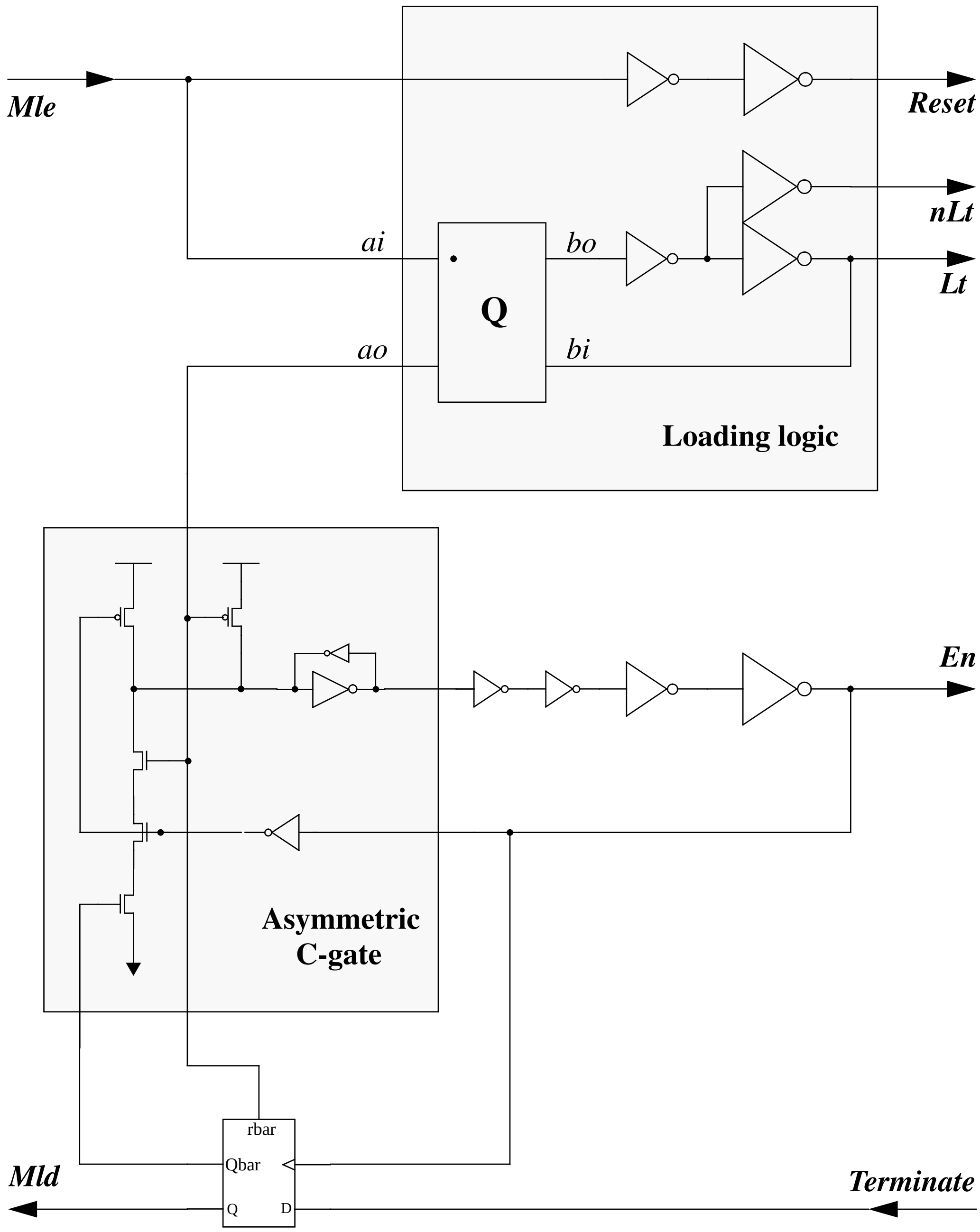


Figure 7-5: Self-timed control logic

Circuit design 8

A high performance, low power asynchronous 32 $\times$ 32-bit multiplier with a low hardware cost has been developed for the *AMULET2* microprocessor using the following circuit design techniques: true single-phase clocking (TSPC) registers, pass-transistor circuits, the new proposed 4-2 counter using differential pass-transistor logic (DPTL), a static early termination circuit and modified weak feedback memory elements.

8.1 Register design

The reasons for using the true single-phase clocking (TSPC) methodology in the register design are as follows:

- a dynamic form of the TSPC registers is well known to be fast.
- the TSPC registers avoid the clock skew problems of complementary phase or multiphase clocking schemes.
- only one enabling signal is required in the TSPC registers.
- the minimum number and size of transistors are needed in the TSPC registers.

The first two factors contribute to achieving high speed, whereas the last two factors lead to power savings.

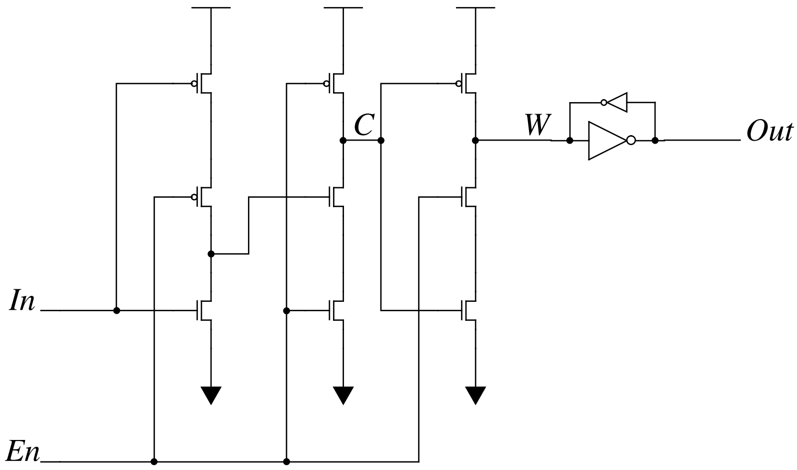


Figure 8-1: Positive-edge triggered TSPC register

A positive-edge triggered TSPC register consists of three clocked inverting stages and a weak feedback memory element as shown in figure 8-1. The first clocked inverting stage is transparent and the third clocked inverting stage is opaque when *En* is low. On the other hand, the first clocked inverting stage is opaque and the third clocked inverting stage is transparent when *En* is high. At the time that *En* is low, node *C* in the second clocked inverting stage is precharged high. When *En* rises, node *C* is either pulled low (*In* high) through two pull-down transistors or remains high (*In* low), and this level is then stored into the weak feedback memory element through the third clocked inverting stage.

Table 8-1: Simulated results of the TSPC register

(30 fF load)	*In -> En*	*En -> Out*	*Power*
typical process case	0.39 ns	0.83 ns	107 µW @ 100 MHz
			214 µW @ 200 MHz
slow-slow process corner	0.55 ns	1.19 ns	99 µW @ 100 MHz
			200 µW @ 200 MHz

The TSPC register has been analysed using HSPICE on extracted layout under the conditions of 3.3 volt supply voltage and 100˚C temperature. The simulated results are given in table 8-1. The estimation of power consumption of a circuit is a difficult problem since it is a strong function of the inputs and their history. With given input patterns, HSPICE can measure the power consumption of a circuit. For the sake of simplicity, the power consumption was measured under the assumption of 100% input activity.

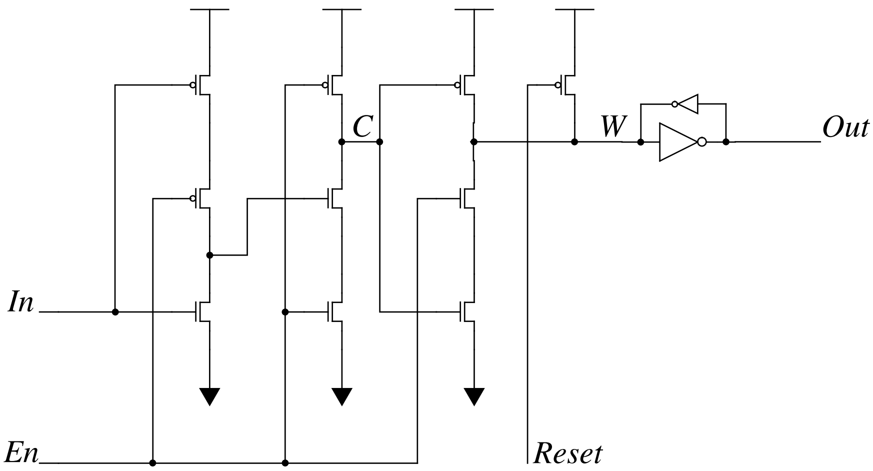


Figure 8-2: TSPC register with reset capability

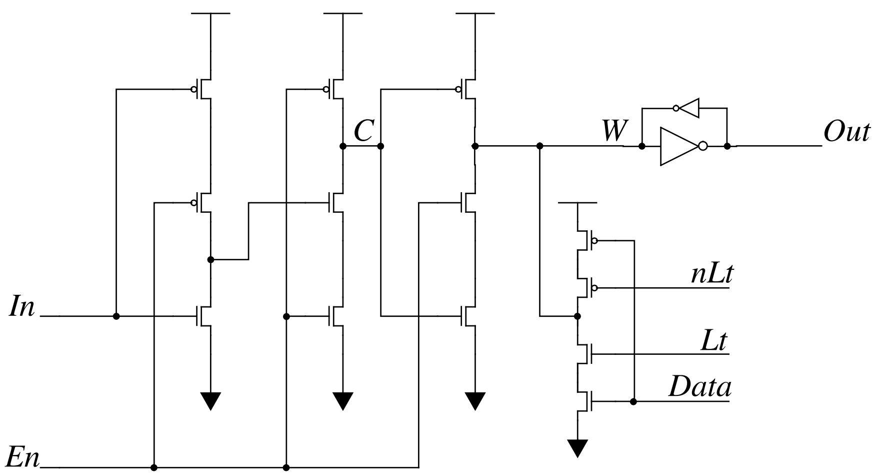


Figure 8-3: TSPC register with direct-load capability

The basic TSPC register shown in figure 8-1 can be modified to provide reset (see figure 8-2), and direct-load (see figure 8-3) capabilities, which is used in the *AMULET2* multiplier. Furthermore, logic may be integrated into the first clocked inverting stage in order to reduce overall delay. This technique is used in the *Pipe* register design shown in figure 8-4.

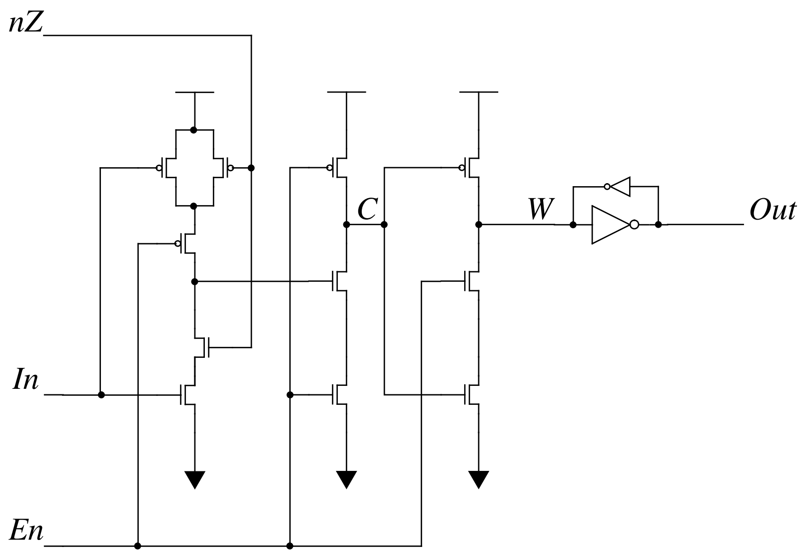


Figure 8-4: *Pipe* register in the *AMULET2* multiplier

8.2 Booth-Mux cell design

The modified Booth's algorithm examines three bits of the multiplier at a time to determine whether to add 0, +1, -1, +2 and -2 times the multiplicand. The function of the Booth-Mux cell is just for this purpose, and it steers one of them to the output. Note that 0 times the multiplicand is merged into the *Pipe* register in the *AMULET2* multiplier. The circuit shown in figure 8-5 is often used in multiplier designs. However, from the view of low power design, this is not a good circuit since there is no guarantee that only one path from the input to the output is on at any time.

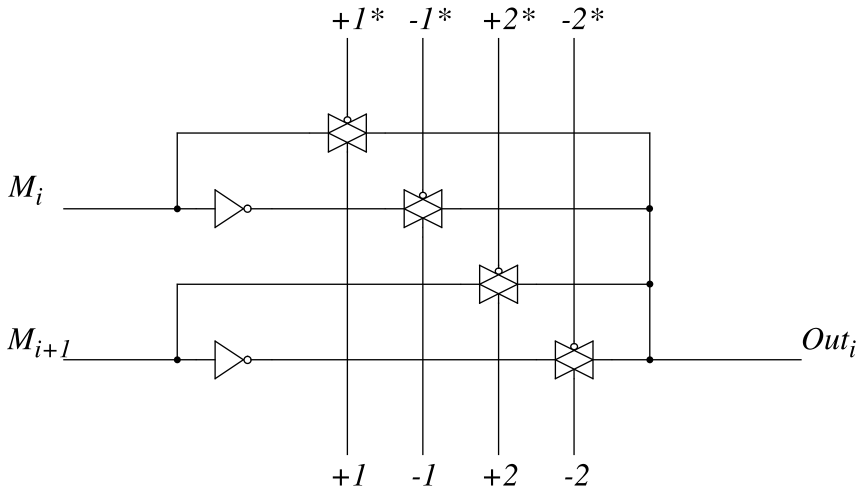


Figure 8-5: Standard encoding Booth-Mux cell

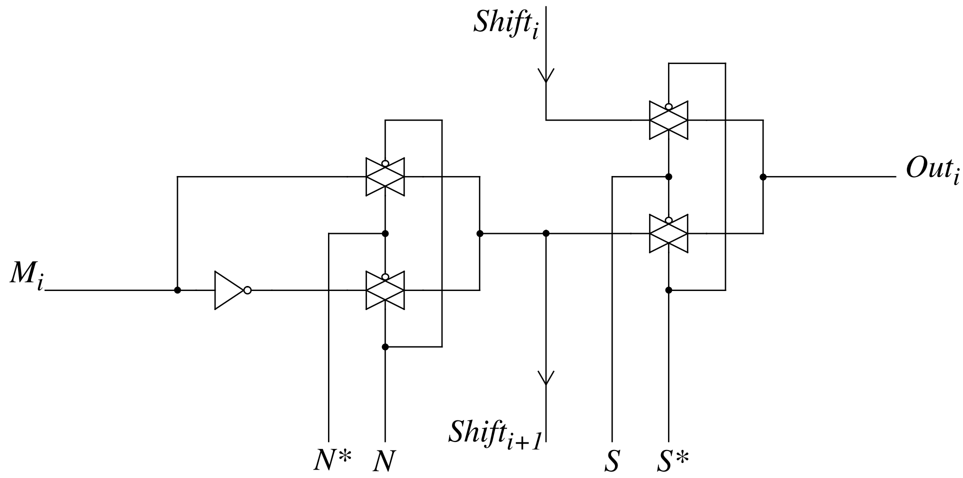


Figure 8-6: Second encoding Booth-Mux cell

Figure 8-6 shows an alternative version of the Booth-Mux cell that is used in the *AMULET2* multiplier. This Booth-Mux cell has been analysed using HSPICE on extracted layout under the conditions of 3.3 volt supply voltage and 100˚C temperature. The simulated results are given in table 8-2. For the sake of simplicity, the power consumption was measured under the assumption of 100% input activity. It is important to realize that the *N* and *S* signals have to travel across the 32-bit datapath together with their complementary *N** and *S** signals, and hence the capacitive load is high.

Table 8-2: Simulated results of the Booth-Mux cell

(30 fF load)	*Critical delay*	*Power*
typical process case	0.44 ns	98 µW @ 100 MHz
		198 µW @ 200 MHz
slow-slow process corner	0.54 ns	83 µW @ 100 MHz
		166 µW @ 200 MHz

8.3 New proposed 4-2 counter

4-2 counters are used to speed up the partial product compression process. The main advantage of 4-2 counters over the more familiar 3-2 counters (i.e., full adders) is that their structure is analogous to a binary tree, which leads to regular layout and improved

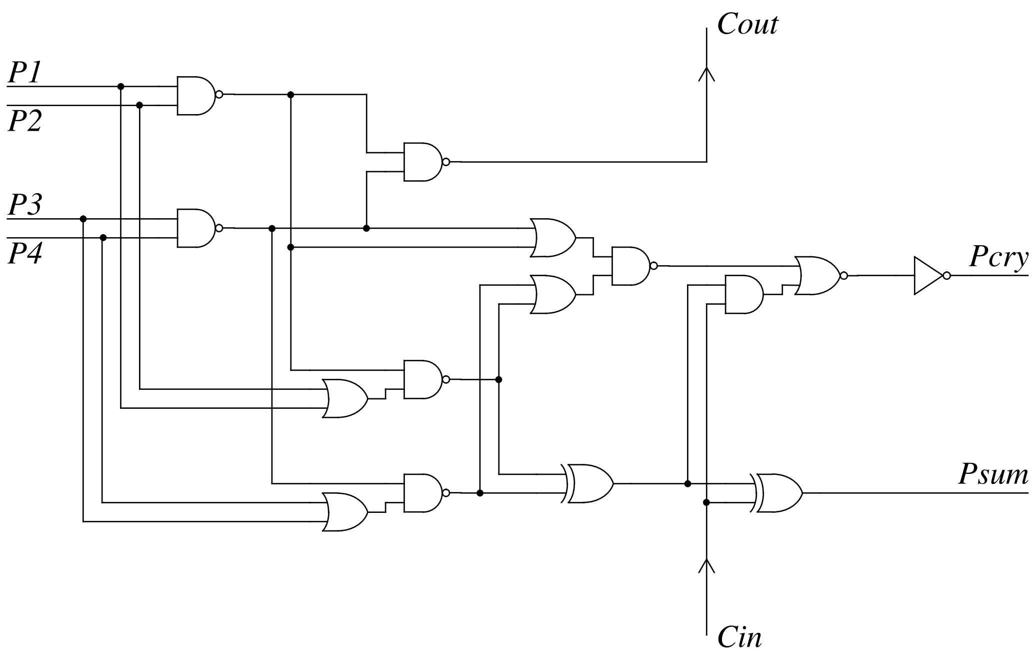


Figure 8-7: Static 4-2 counter

speed. Figure 8-7 shows a static 4-2 counter. The engineering estimate of delays usually treats an XOR or XNOR gate as 1.5 gate delays, a two-input NAND or NOR gate as 1 gate delay, and an inverter or a CMOS transmission gate as 0.5 gate delays. Therefore, the delay of the static 4-2 counter shown in figure 8-7 is about 4.5 gate delays.

Figure 8-8 shows the new proposed 4-2 counter using differential pass-transistor logic (DPTL). The critical path of the circuit only goes through two equivalent 3-input inverting stages. This circuit borrows a common practice from analog designs in that noise immunity is achieved by using differential signals. The use of both nMOS and pMOS transistors here instead of only nMOS transistors in DPTL ensures the full signal swing to reduce power consumption. Various 4-2 counters using the DPTL concept can be obtained by changing the different styles of XOR and XNOR gate implementations described in section 4.2 (see “An example: XOR gate design” on page 42). Since the number of tracks available for buses is limited in the *AMULET2* datapath design (see “Full-custom format” on page 86), the partial product signals have to be single-ended signals. The inverter-based XOR gates are thus used in the first stage of the proposed 4-2 counter. The CMOS transmission XOR gates are used in the following stage of the proposed 4-2 counter after the internal complementary signals are available.

Table 8-3: Simulated results of the new proposed 4-2 counter

<table>
<tr><th>(30 fF load)</th><th>Critical delay</th><th>Power</th></tr>
<tr><td rowspan="2">typical process case</td><td rowspan="2">1.36 ns</td><td>258 µW @ 100 MHz</td></tr>
<tr><td>543 µW @ 200 MHz</td></tr>
<tr><td rowspan="2">slow-slow process corner</td><td rowspan="2">1.66 ns</td><td>249 µW @ 100 MHz</td></tr>
<tr><td>517 µW @ 200 MHz</td></tr>
</table>

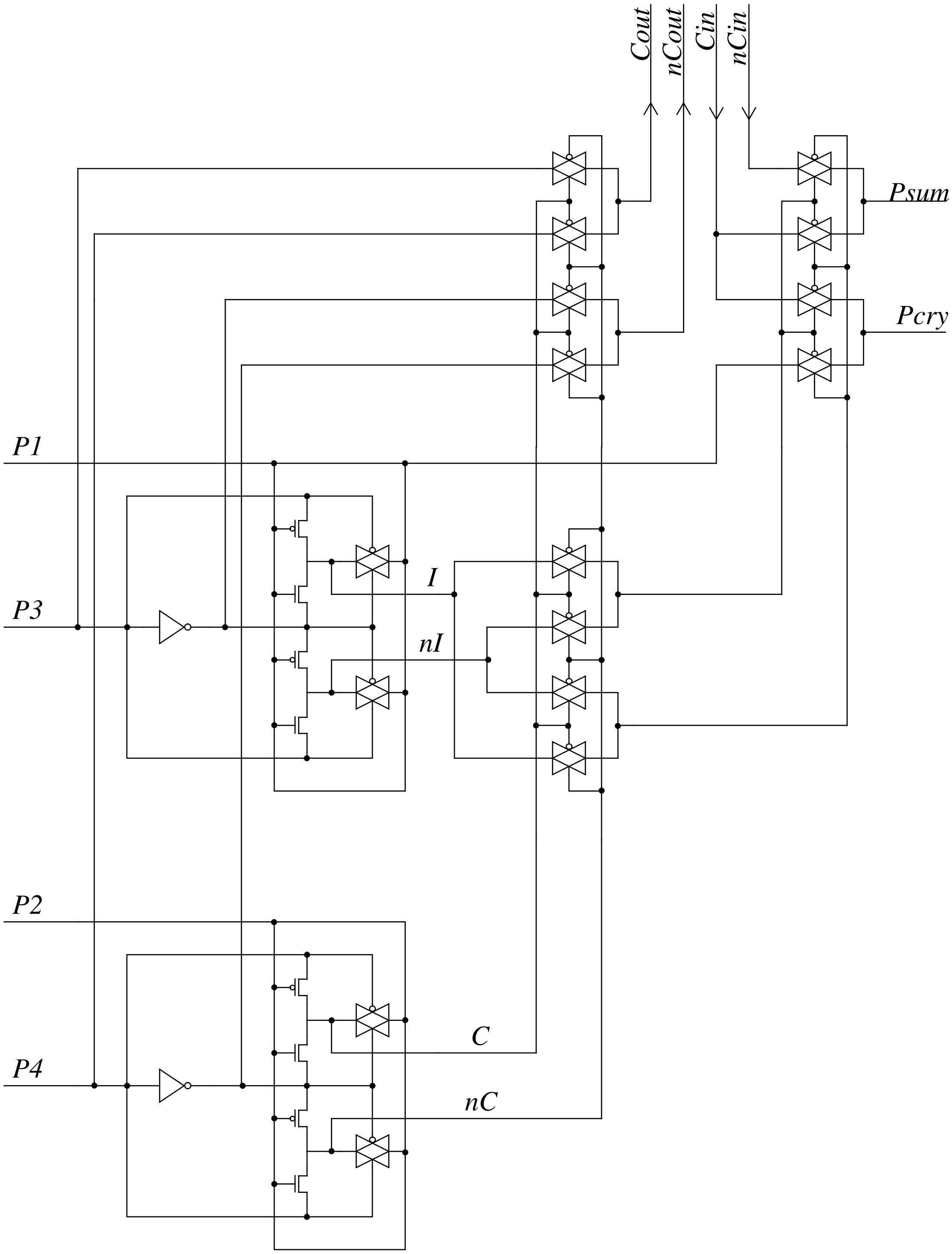


Figure 8-8: New proposed 4-2 counter

The new proposed 4-2 counter has been analysed using HSPICE on extracted layout under the conditions of 3.3 volt supply voltage and 100˚C temperature. The simulated results are given in table 8-3. For the sake of simplicity, the power consumption was measured under the assumption that only one input is active.

8.4 Early termination design

The early termination circuit is responsible for detecting all ones or all zeros in the multiplier register *A*. Although dynamic or pseudo-nMOS techniques are extensively used for zero/one detectors, a tree of static gates are employed here instead. Figure 8-9 shows the 8-bit static tree zero/one detector, which is used to implement the 32-bit zero/one detector in the *AMULET2* multiplier. The reasons for using static gates for the termination circuit are as follows:

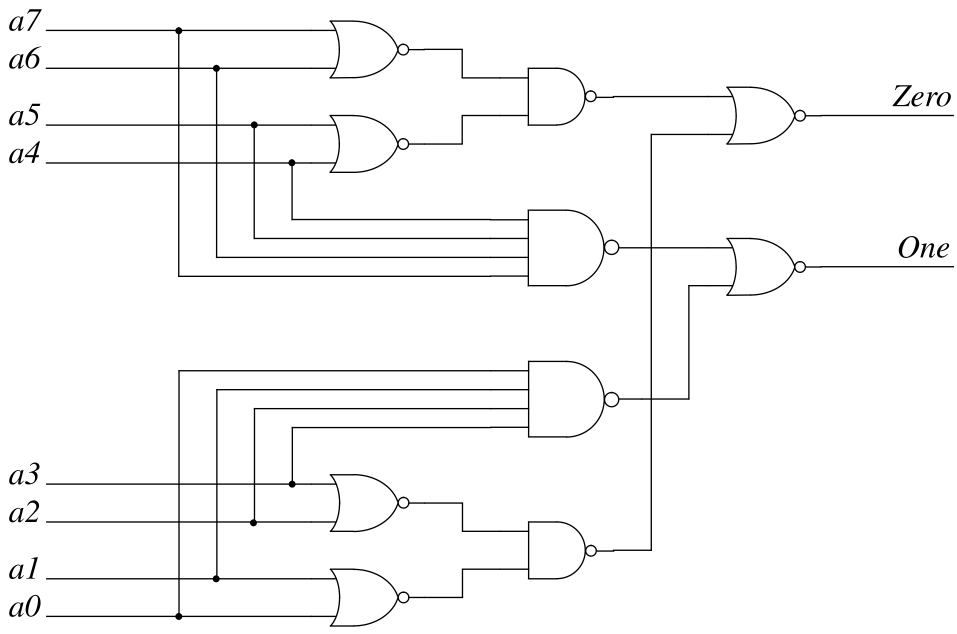


Figure 8-9: 8-bit static tree zero/one detector

- ❏ dynamic techniques require a specific clock to precharge a large capactive node, which dissipates much power.
- ❏ the precharging clock destroys the benefit from the TSPC technique, since the skew between the precharging clock and the enabling clock must be closely examined.
- ❏ pseudo-nMOS techniques are not suitable for low power design because the self-loading transistor is always on.
- ❏ the static technique in this particular case has led to a low power design, since the number of ones or zeros in the multiplier register *A* will increase after each shifting cycle, which means that an increasing part of the tree of static gates will be stable (and will not dissipate dynamic switching power) and a decreasing part remains active.

The 8-bit static tree zero/one detector has been analysed using HSPICE on extracted layout under the conditions of 3.3 volt supply voltage and 100˚C temperature. The simulated results are given in table 8-4. For the sake of simplicity, the power consumption was measured under the assumption that only 4 inputs (on average) are changed.

Table 8-4: Simulated results of the 8-bit static 0/1 detector

<table>
<tr><th>(30 fF load)</th><th>Critical delay</th><th>Power</th></tr>
<tr><td rowspan="2">typical process case</td><td rowspan="2">1.04 ns</td><td>181 µW @ 100 MHz</td></tr>
<tr><td>374 µW @ 200 MHz</td></tr>
<tr><td rowspan="2">slow-slow process corner</td><td rowspan="2">1.39 ns</td><td>173 µW @ 100 MHz</td></tr>
<tr><td>352 µW @ 200 MHz</td></tr>
</table>

8.5 Memory element design

The memory element previously used in *AMULET1* is the weak feedback structure shown in figure 8-10. The n/p transistor network must be capable of overdriving the weak feedback inverter which is usually implemented using transistors with a long channel length. This memory element is in fact a pseudo-static scheme. The purpose of the weak feedback inverter here is to prevent static power consumption and provide some noise margins. One disadvantage of this scheme is that the node *N* can be “floating” when the n/p transistor network logic is closed. In other words, no circuitry drives the node *N*, so it is very noise sensitive. Therefore, the node *N* cannot be loaded without proper buffering.

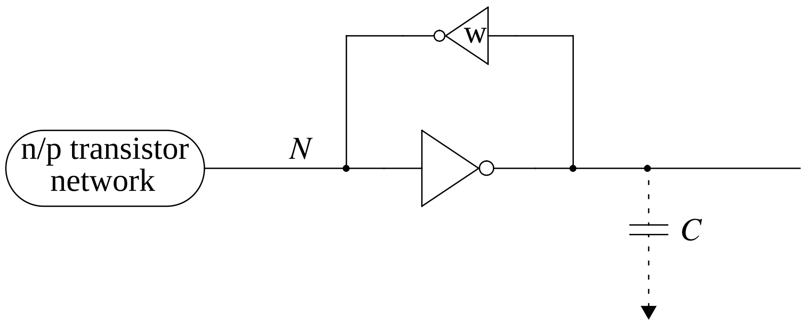


Figure 8-10: Weak feedback memory element

Another disadvantage is that the output node sees the large capacitance caused by using long length transistors. To solve this problem, the weak feedback inverter can be implemented using minimum size enabling transistors in series with hard-wired on transistor resistors [65] as shown in figure 8-11. The output capacitance of the modified weak feedback memory element is reduced, resulting in an increase in speed and a reduction in power consumption.

The cost of the modified weak feedback memory element is a 20% increase in silicon area since the power rail connections crossover the gate [65]. However, in standard cell designs, the height of the standard cells is usually determined by a buffer with a 2 ~ 3 fanout drive capability. For example, the overall height and width of the standard cell for *AMULET2* are 108λ and a multiple of 9λ, respectively. This means that there is still spare room in the cell for the modified weak feedback memory element, and it is thus possible to “hide” some of the silicon area cost.

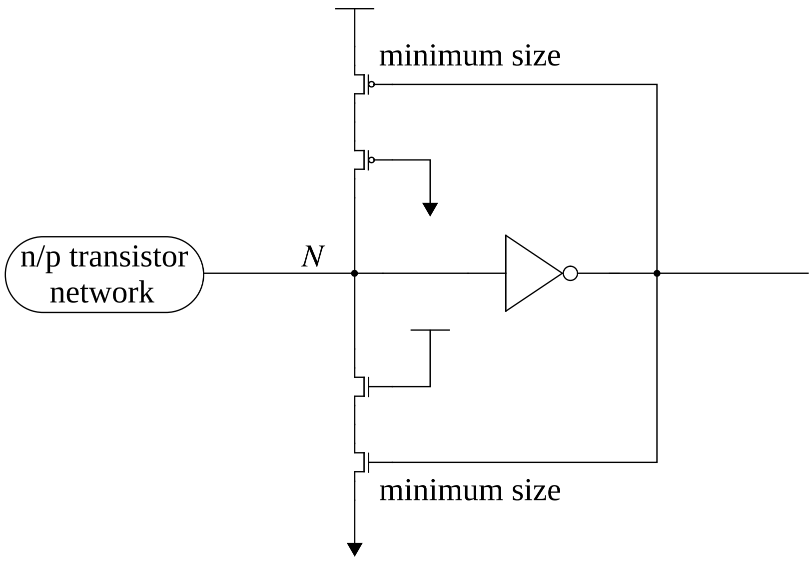


Figure 8-11: Modified weak feedback memory element

Layout design

9

The layout design of the *AMULET2* multiplier uses a full-custom style for the datapath, where the circuit and layout of almost every transistor is optimized, and a standard cell style for the control logic, where the layout is automatically placed and routed using Compass Design Automation Systems.

9.1 Full-custom format

The full-custom style is used in order to exploit the regularity of the datapath by designing only one “bit slice”. The height of the bit slice in the datapath design is 67λ for the *AMULET2* multiplier. The number of tracks available for buses is nine per bit slice. Only four tracks are for local routing, since the other five tracks are used for through buses. This forces the use of single-ended signals for the circuits, which narrows the choice for the circuit design (otherwise, RAM-style registers might be considered). Data flow is routed horizontally in metal3, while control information is relayed vertically in metal2. Both metal1 and metal2 are used for local interconnections in cells. The global power rails use metal1 and metal3, and the local power rails use metal2. Note that the regularity of the early termination circuit is achieved by considering eight bit as only “one” bit.

9.2 Standard-cell format

The overall height and width of the standard cells for *AMULET2* are 108λ and a multiple of 9λ, respectively. This means that the connectors of a cell must have a 9λ spacing and a 4.5λ horizontal margin to either side of a cell. By taking into account existing open vertical routing tracks inside the standard cells, the routing over cell algorithm helps to reduce the final chip size.

9.3 Layout design

The cells for the datapath usually comprise two stages. The first stage contains small transistors to minimize the required area and power, whereas the second stage uses ordinary transistors to ensure the capability to drive the capacitive load at the cell outputs. When the layout of a cell was complete, it was verified against the correspondent schematic (using Compass Design Automation Systems), and then simulated again using HSPICE. Figure 9-1 shows the datapath arrangement of the *AMULET2* multiplier. Figure 9-2 shows the physical layout of the datapath of the *AMULET2* multiplier.

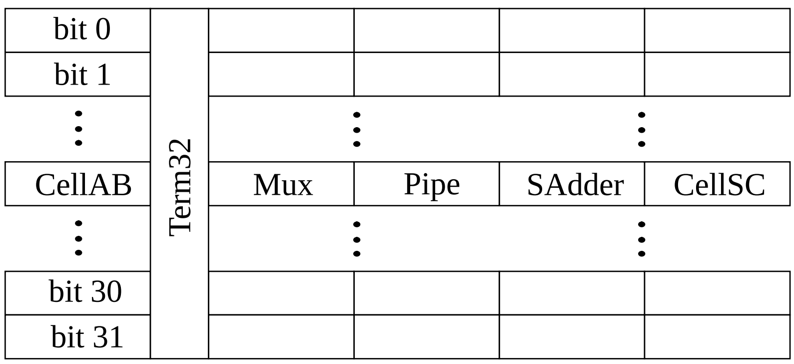


Figure 9-1: Datapath arrangement

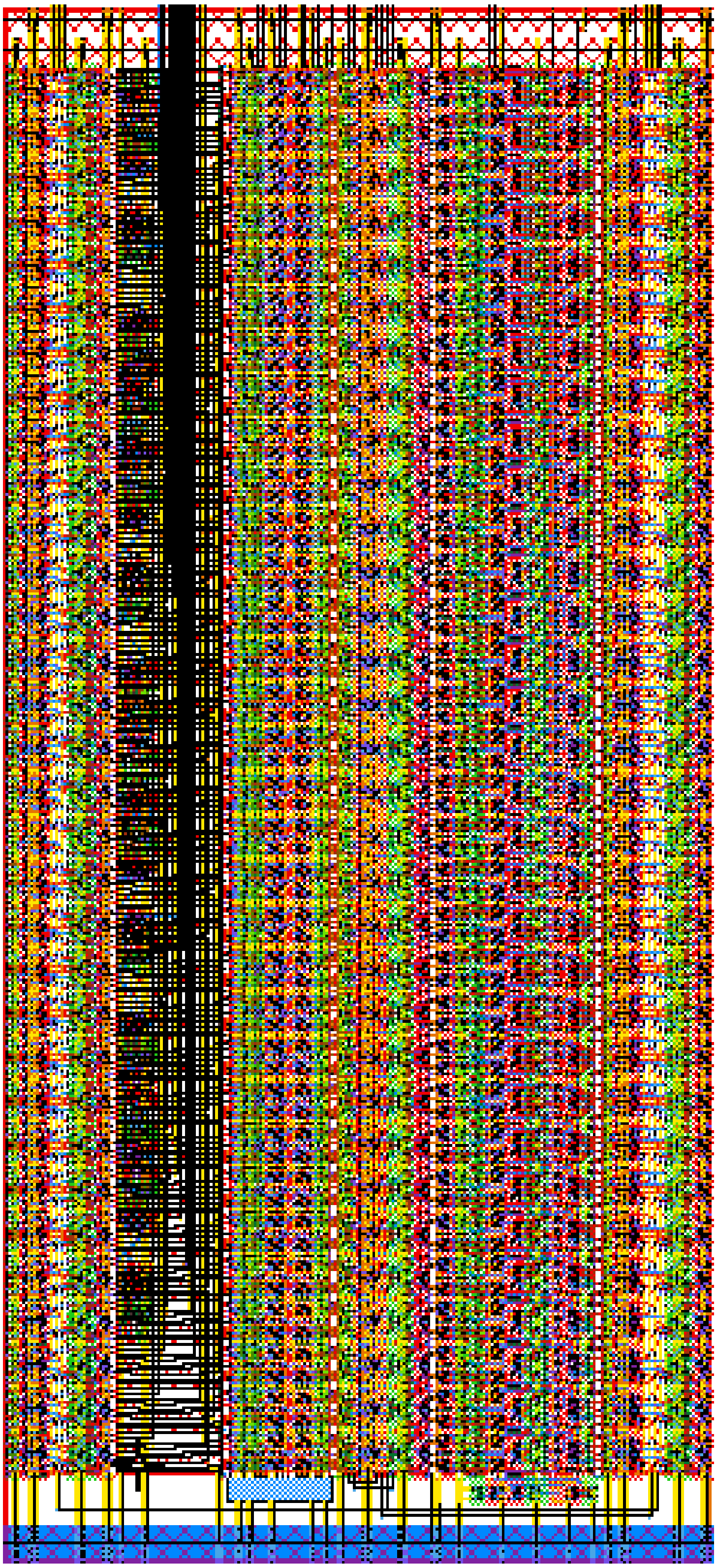


Figure 9-2: Physical layout of the datapath

Evaluation 10

An evaluation of the *AMULET2* multiplier in terms of performance, power consumption and silicon area is presented in this chapter. The estimation of power consumption is a difficult problem since it is a strong function of the inputs and their history. A rough estimate of power consumption is given based on some assumptions.

10.1 Performance

A summary of the simulated delays (see “Circuit design” on page 74) is shown in figure 10-1. The critical path delay is about 3.4 ns under the worst case, resulting in a 200 MHz computational speed with a 1.6 ns engineering margin.

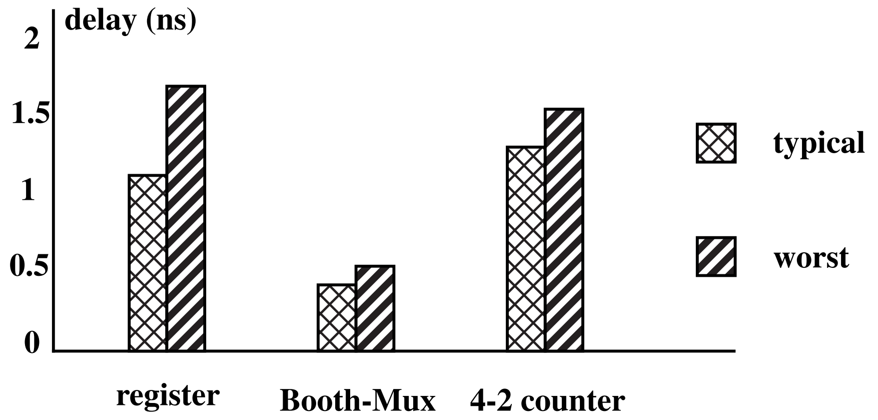


Figure 10-1: Simulated delays

10.2 Power consumption

Figure 10-2 shows a summary of the simulated power consumption (see "Circuit design" on page 74). It is highly unlikely that all data bits will change for every data value. Based on the assumptions that half the data bits on average will change and that the dynamic switching power is 90% of the total power, the power estimate of the datapath (typical case) at 100 and 200 (MHz) is about 29 and 58 (mW), respectively.

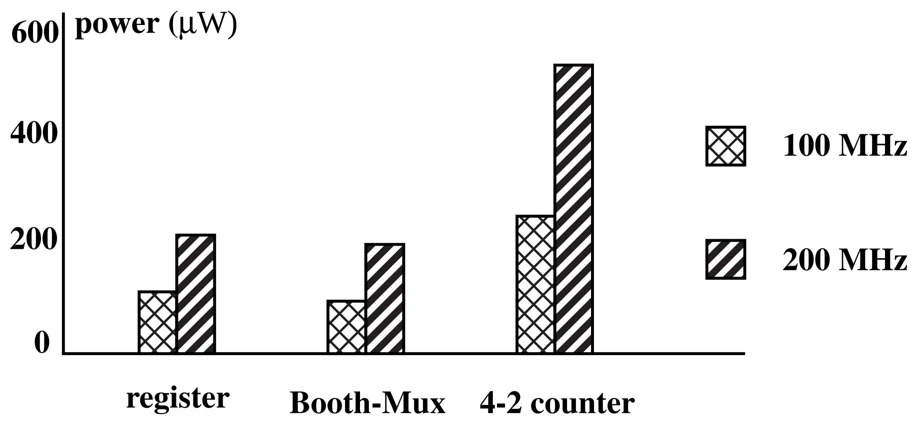


Figure 10-2: Simulated power consumption

10.3 Silicon area

The silicon area of the datapath is $1079\lambda \times 2371\lambda$ (0.32×0.71 mm^2). The silicon area required comprises four parts: *R* (for registers), *C* (for routing), *B* (for Booth' decoders), and *A* (for 4-2 counters). The area comparison of these four parts is shown in table 10-1. The area occupied by registers, 46% of the total datapath area, is a major part.

Table 10-1: Area comparison of four parts

	R	*C*	*B*	*A*
Area percentage	46 %	16 %	15 %	23 %

Conclusions 11

A high performance, low power asynchronous 32×32-bit multiplier with several novel features has been developed for the *AMULET2* microprocessor at a low hardware cost. A new design for a 4-2 counter using differential pass-transistor logic (DTPL) has been used, together with several circuit design techniques including true single-phase clocking registers, resulting in a computation rate exceeding 200 MHz and about 58 mW power consumption with 0.6 μm three-level metal CMOS process technology. The resulting cell is compact with a datapath area of only 0.32×0.71 mm^2.

11.1 Features

The resulting design exhibits the following features:

- **Asynchronous self-contained operation** — The multiplier communicates asynchronously with other parts of the *AMULET2* microprocessor.
- **Full 32-bit** — The arguments and the result of a multiply operation are all 32 bits.
- **Iterative structure** — The iterative structure provides the benefit of reducing the hardware cost by increasing hardware utilization.

- **Efficient Pipeline** — The efficient pipeline design results in balanced delays between pipeline stages.
- **Modified Booth's algorithm** — The modified Booth's algorithm reduces the number of partial products by a factor of two.
- **4-bit shift at a time** — 2 steps of the Booth's algorithm are performed each cycle by shifting 4 bits at a time.
- **Early termination** — Early termination means that the multiplication time is data dependent, and hence the circuit achieves typical speed and saves power.
- **Carry save scheme** — The carry save scheme avoids carry propagation during each partial product compression cycle.
- **TSPC register** — High speed, low power true single-phase clocking registers are used.
- **4-2 counter** — The proposed 4-2 counter using differential pass-transistor logic is symmetric with respect to its inputs and outputs.
- **Minimum size transistor** — The use of small size transistors is a design technique for low power.
- **Differential pass-transistor logic (DPTL)** — The symmetrical arrangement of DPTL improves the speed and the noise margins of a circuit.

11.2 Characteristics

The characteristics of the *AMULET2* multiplier are as follows:

- **Process technology** — *VLSI Technology's CMN5* CMOS process.
- **Design feature** — 0.6 μm design rule.

- ❑ **Transistor count** — 6895 transistors.
- ❑ **High performance** — 200 MHz computational speed.
- ❑ **Low power** — 58 mW (@ 200 MHz) under the typical case.
- ❑ **Small area** — 0.32×0.71 mm^2 of the datapath.

11.3 8-bit shift at a time

Evaluations of the *AMULET2* multiplier have shown that some improvements may possibly be made in several areas. One of them is to use an 8-bit shift at a time scheme.

11.3.1 Necessity

The register overhead in the *AMULET2* multiplier is high. Firstly, the register delay is about 1.74 ns, which is 51% of the critical delay and does not contribute to computation. Secondly, the chip area occupied by the registers is 46% of the total datapath area.

What is the advantages of an 8-bit shift at a time scheme? Firstly, this scheme halves the number of cycles, which means that a 50% reduction of the total register delay can be achieved although the single register delay remains the same. Secondly, it also halves the clock frequency, which contributes to considerable power savings.

11.3.2 Possibility

4-2 counters, state-of-the-art implementations of partial product trees, have small delays and hence offer the possibility of adding additional rows of 4-2 counters. Furthermore, 8-2 counters, based on a similar idea to the 4-2 counters, may be derived with even smaller delays.

11.4 Redundancy

From the design of the *AMULET2* multiplier, redundancy is the key concept, which is applied to both the multiplier and the multiplicand. Can further exploitation of the redundancy concept be made? As we all know, higher order versions of Booth's algorithm have rarely been used in multiplier designs, since it is difficult to generate the required multiples such as 3 times the multiplicand. By using the redundancy concept, this problem may possibly be avoided [47].

Bibliography

[1] Furber, S.B., Day, P., Garside, J.D., Paver, N.C., Temple, S. and Woods, J.V., “The design and evaluation of an asynchronous microprocessor”, Proceedings of ICCD 94, Boston, Massachusetts, October 1994.

[2] Furber, S.B., Day, P., Garside, J.D., Paver, N.C. and Woods, J.V., “A Micropipelined ARM”, Proceedings of the IFIP TC 10/WG 10.5 International Conference on Very Large Scale Integration (VLSI’93), Grenoble, France, September 1993, Ed. Yanagawa, T. and Ivey, P.A., Pub. North Holland.

[3] Furber, S.B., Day, P., Garside, J.D., Paver, N.C. and Woods, J.V., “AMULET1: A Micropipelined ARM”, Proceedings of the IEEE Computer Conference, San Francisco, March 1994.

[4] Paver, N.C., “The design and implementation of an asynchronous microprocessor”, PhD thesis, University of Manchester, June 1994.

[5] Furber, S.B., “VLSI RISC architecture and organization”, Marcel Dekker Inc., New York, 1989.

[6] Furber, S.B., “Computing without clocks”, Proceedings of the VII Banff Workshop: Asynchronous hardware design, Banff, Canada, 1993.

[7] Liu, D. and Svensson, C., “Power consumption estimation in CMOS VLSI chips”, IEEE Journal of Solid-State Circuits, vol. 29, June 1994, pp. 663-670.

[8] Mead, C. and Conway, L., “Introduction to VLSI systems”, Addison-Wesley, Reading, Massachusetts, 1980.

[9] Liu, D. and Svensson, C., “Trading speed for low power by choice of supply and threshold voltages”, IEEE Journal of Solid-State Circuits, vol. 28, January 1993, pp. 10-17.

[10] Chandrakasan, A.P., Sheng, S. and Brodersen, E.W., “Low-power CMOS digital design”, IEEE Journal of Solid-State Corcuits, vol. 27, April 1992, pp. 473-484.

[11] Uyemura, J.P., “Circuit design for CMOS VLSI”, Kluwer Academic Publishers, 1992.

[12] Shoji, M., “CMOS digital circuit technology”,
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988.

[13] Nielsen, L.S., Niessen, C., Sparsø, J. and Berkel, K.V.,
“Low-power operation using self-timed circuits and adaptive scaling of the supply voltage”, IEEE Transaction on VLSI Systems, vol. 2, December 1994, pp. 391-397.

[14] Burd, T., “Low-power CMOS library design methodology”,
MSc thesis, University of California, Berkeley, 1994.

[15] Weste, N.H.E. and Eshraghian, K.,
“Principle of CMOS VLSI design: a system perspective”,
Addison-Wesley, Massachusetts, 1988.

[16] Veendrick, H.J.M., “Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits”, IEEE Journal of Solid-State Circuit, vol. 19, August 1984, pp. 468-473.

[17] Huffman, D.A., “The synthesis of sequential switching circuits”,
J.Franklin Institute, vol. 257, March/April 1954,
pp. 161-190/275-303.

[18] Unger, S.H., “Asynchronous sequential switching circuits”,
Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[19] Chu, T.A., “On the models for designing VLSI asynchronous digital systems”, Intergration, the VLSI Journal, vol. 4, 1986, pp. 99-113.

[20] Lavagno, L. and Vincentelli, A.S.,
“Algorithms for synthesis and testing of asynchronous circuits”,
Kluwer Academic Publishers, 1993.

[21] Muller, D.E. and Bartky, W.C., “A theory of asynchronous circuits”,
Annals of Computing Laboratory of Harvard University,
1959, pp.204-243.

[22] Kishinevsk, M., Kondratyev, A., Taubin, A. and Varshavsky, V.,
“Concurrent hardware: the theory and practice of self-timed design”, John Wiley & Sons, Inc., New York, 1994.

[23] Staunstrup, J., “A formal approach to hardware design”,
Kluwer Academic Publishers, 1994.

[24] Sparsø, J. and Staunstrup, J.,
“Delay-insensitive multi-ring structures”,
Integration, the VLSI Journal, vol. 15, 1993, pp. 313-340.

[25] Ebergen, J.C., “Translating programs into delay-insensitive circuits”, PhD thesis, Eindhoven University of Technology, 1987.

[26] Martin, A.J., “The limitations to delay-insensitivity in asynchronous circuits”, Sixth MIT Conference on Advanced Research in VLSI, Cambridge, Massachusetts, 1990, Ed. Dally, W.J., Pub. MIT press.

[27] Martin, A.L., “Synthesis of asynchronous VLSI circuits”, Technical Report, Caltech-CS-TR-93-28, Department of Computer Science, California Institute of Technology, 1993.

[28] Berkel, K.V., “Beware the isochronic fork”, Integration, the VLSI Journal, vol. 13, 1992, pp. 103-128.

[29] Lin, C.J. and Reddy, S.M., “On delay fault testing in logic circuits”, IEEE Transactions on Computer Aided Design, vol. 6, September 1987, pp. 694-703.

[30] Furber, S.B. and Day, P., “Four-phase micropipeline latch control circuits”, submitted to IEEE Transaction on VLSI Systems, 1995.

[31] Day, P. and Woods, J.V., “Investigation into micropipeline latch design styles”, IEEE Transaction on VLSI Systems, vol. 3, June 1995, pp. 264-272.

[32] Verhoeff, T., “Delay-insensitive codes — an overview”, Distributed Computing, vol. 3, 1988, pp. 1-8.

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

[34] Berkel, K.V., “Handshake circuits: an asynchronous architecture for VLSI programming”, International Series on Parallel Computation, Cambridge University Press, 1993.

[35] Berkel, K.V., Burgess, R., Kessels, J.L.W., Peeters, A., Roncken, M. and Schalij, F., “A fully asynchronous low power error corrector for the DCC player”, IEEE Journal of Solid-State Circuits, vol. 29, December 1994, pp. 1429-1239.

[36] Berkel, K.V., Huberts, F. and Peeters, A., “Stretching quasi delay insensitivity by means of extended isochronic forks”, Proceeding of the Second Working Conference on Asynchronous Design Methodologies, London, England, May 1995.

[37] Chandrakasan, A.P., Burstein, A. and Broderson, R.W., “A low-power chipset for a portable multimedic I/O terminal”, IEEE Journal of Solid-State Circuits, vol. 29, December 1994, pp. 1415-1428.

[38] Wu, X. and Prosser, “Theory of transmission switches and its application to design of CMOS digital circuits”, Int. J. Circuit Theory Application, vol. 20, 1992.

[39] Zhuang, N., and Wu, H., “A new design of the CMOS full adder”, IEEE Journal Solid-State Circuit, vol. 27, May 1992, pp. 840-844.

[40] Wang, J.M., Fang, S.C. and Feng, W.S., “New efficient designs for XOR and XNOR functions on the transistor level”, IEEE Journal of Solid-State Circuits, vol. 29, July 1994, pp. 780-786.

[41] Pasternak, J.H. and Salama, C.A.T., “Design of submicrometer CMOS differential pass-transistor logic circuits”, IEEE Journal Solid-State Circuit, vol. 26, September 1991, pp. 1249-1258.

[42] Yano, K., Yamanaka, T., Nishida, T., Saito, M., Shimonigashi, K. and Shimizu, A., “A 3.8-ns CMOS 16×16-b multiplier using complementary pass-transistor logic”, IEEE Journal Solid-State Circuit, vol. 25, April 1990, pp. 388-395.

[43] Dobberpuhl, D.W. et al., “A 200- MHz 64-b dual-issue CMOS microprocessor”, IEEE Journal of Solid-State Circuit, vol. 27, November 1992, pp. 1555-1565.

[44] Unger, S.H. and Tan, C., “Clocking schemes for high-speed digital systems”, IEEE Transaction on Computer, vol. 35, 1986, pp. 880-895.

[45] Yuan J. and Svensson C., “High-speed CMOS circuit techniques,” IEEE Journal of Solid-State Circuits, vol. 24, February 1989, pp. 62-70.

[46] Larsson, P. and Svensson, C., “Impact of clock slope on true single phase clocked (TSPC) CMOS circuits”, IEEE Journal Solid-State Circuit, vol. 29, June 1994, pp. 723-726.

[47] Bewick, G. and Flynn, M.J., “Binary multiplication using partially redundant multiples”, Technical Report, CSL-TR-92-528, Computer Systems Laboratory, Standford University, June 1992.

[48] Avizienis, A., “Signed-digit number representations for fast parallel arithmetic”, IRE Transactions on Electronic Computers, vol. 10, September 1961, pp. 389-400.

[49] Harata, Y., Nakamura, Y., Nagase, H., Takigawa, M. and Takagi, N., “A high-speed multiplier using a redundant binary adder tree”, IEEE Journal of Solid-State Circuits, vol. 22, February 1987, pp. 28-34.

[50] Omondi, A.R., “Computer arithmetic systems – algorithms, architecture and Implementation”, Prentice Hall International (UK) Limited, Cambridge, 1994.

[51] Wallace, C.S., “A suggestion for parallel multipliers”, IEEE Transactions on Electronic Computer, vol. 13, 1964, pp. 14-17.

[52] Hennessy, J.L. and Patterson, D.A., “Computer architecture, a quantitative approach”, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[53] Zuras, D. and McAllister, W., “Balanced delay trees and combinatorial division in VLSI”, IEEE Journal of Solid-State Circuits, vol. 21, October 1986, pp. 814-819.

[54] Mou, Z. and Jutand, F., “A class of close to optimum adder trees allowing regular and compact layout”, IEEE Transactions on Computers, vol., 1990, pp. 251-254.

[55] Santoro, M., “SPIM: a pipelined 64×64-bit iterative multiplier”, IEEE Journal of Solid-State Circuits, vol. 24, April 1989, pp. 487-493.

[56] Booth, A.D., “A signed binary multiplication technique”, Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, June 1951, pp. 236-240.

[57] MacSorley, O.L., “High-speed arithmetic in binary computers”, Proceedings of the IRE, vol. 49, January 1961, pp. 67-91.

[58] Day, P., “A micropipelined multiplier”, ACiD-WG/EXACT Workshop on Asynchronous Processing, Veldhoven, Netherlands, December 1992.

[59] Paver, N.C., Day, P., Furber, S.B., Garside, J.D. and Woods, J.V., “Register locking in an asynchronous microprocessor”, Proceedings of ICCD 92, October 1992.

[60] Garside, J.D., “A CMOS VLSI implementation of an asynchronous ALU”, IFIP Working Conference on Asynchronous Design Methodologies, Manchester, England, April 1993.

[61] Nagamatsu, M., Tanaka, S., Mori, J., Hirano, K., Noguchi, T. and Hatanaka, K., “A 15-ns 32×32-b CMOS multiplier with an improved parallel structure”, IEEE Journal of Solid-State Circuits, vol. 25, April 1990, pp. 494-497.

[62] Mori, J. et al., “A 10-ns 54×54 parallel structured full array multiplier with 0.5-µm CMOS technology”, IEEE Journal of Solid-State Circuits, vol. 26, 1991, pp. 600-606.

[63] Goto, G, Sato, T., Nakajima, M. and Sukemura, T., “A 54×54 regularly structured tree multiplier”, IEEE Journal of Solid-State Circuits, vol. 27, September 1992, pp. 1229-1236.

[64] Gerosa, G, Gary, S. and Dietz, C., “A 2.2W 80MHz superscalar RISC microprocessor”, IEEE Journal of Solid-State Circuits, vol. 29, December 1994, pp. 1440-1454.

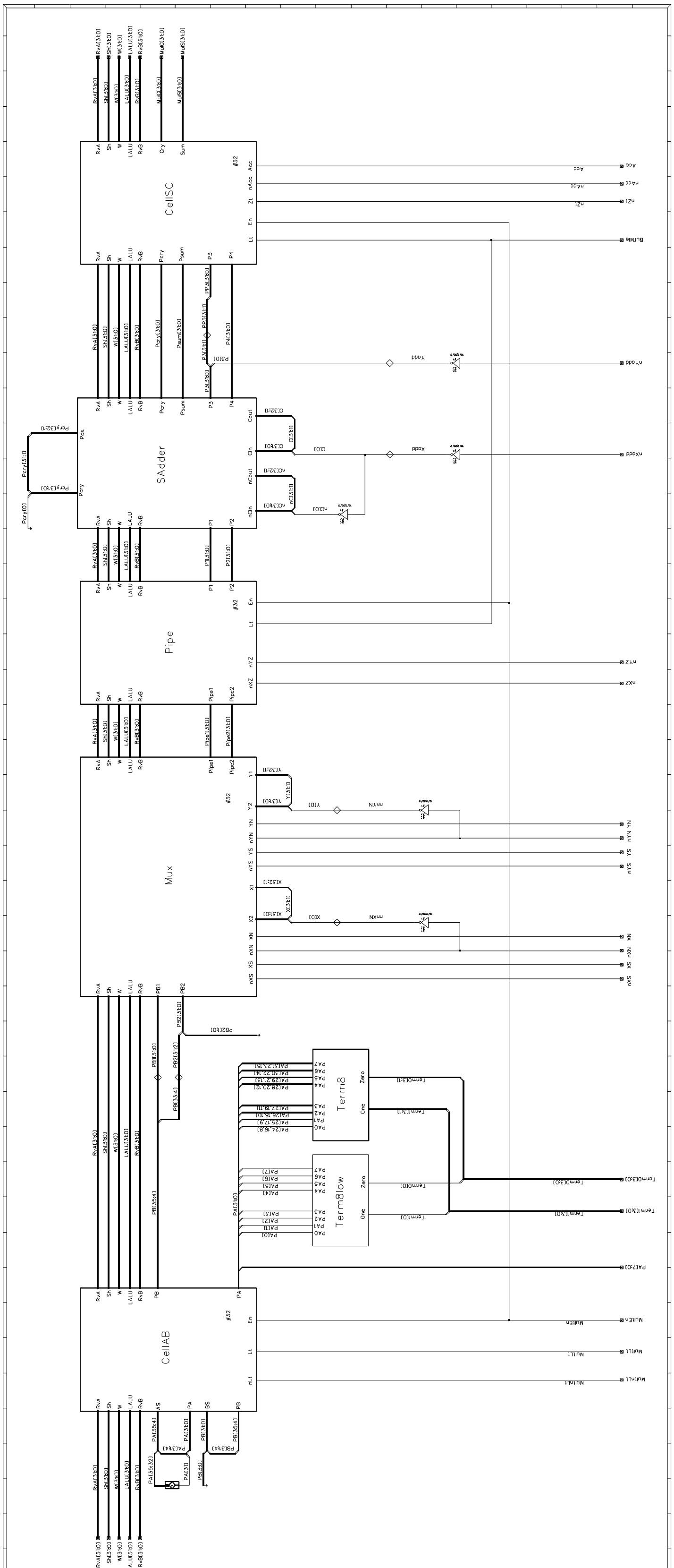
[65] Craig Farnsworth, Doug Edwards, Jianwei Liu and Shiv Sikand, “A hybrid Asynchronous System Design Environment”, Proceeding of the Second Working Conference on Asynchronous Design Methodologies, London, England, May 1995.

Schematics A

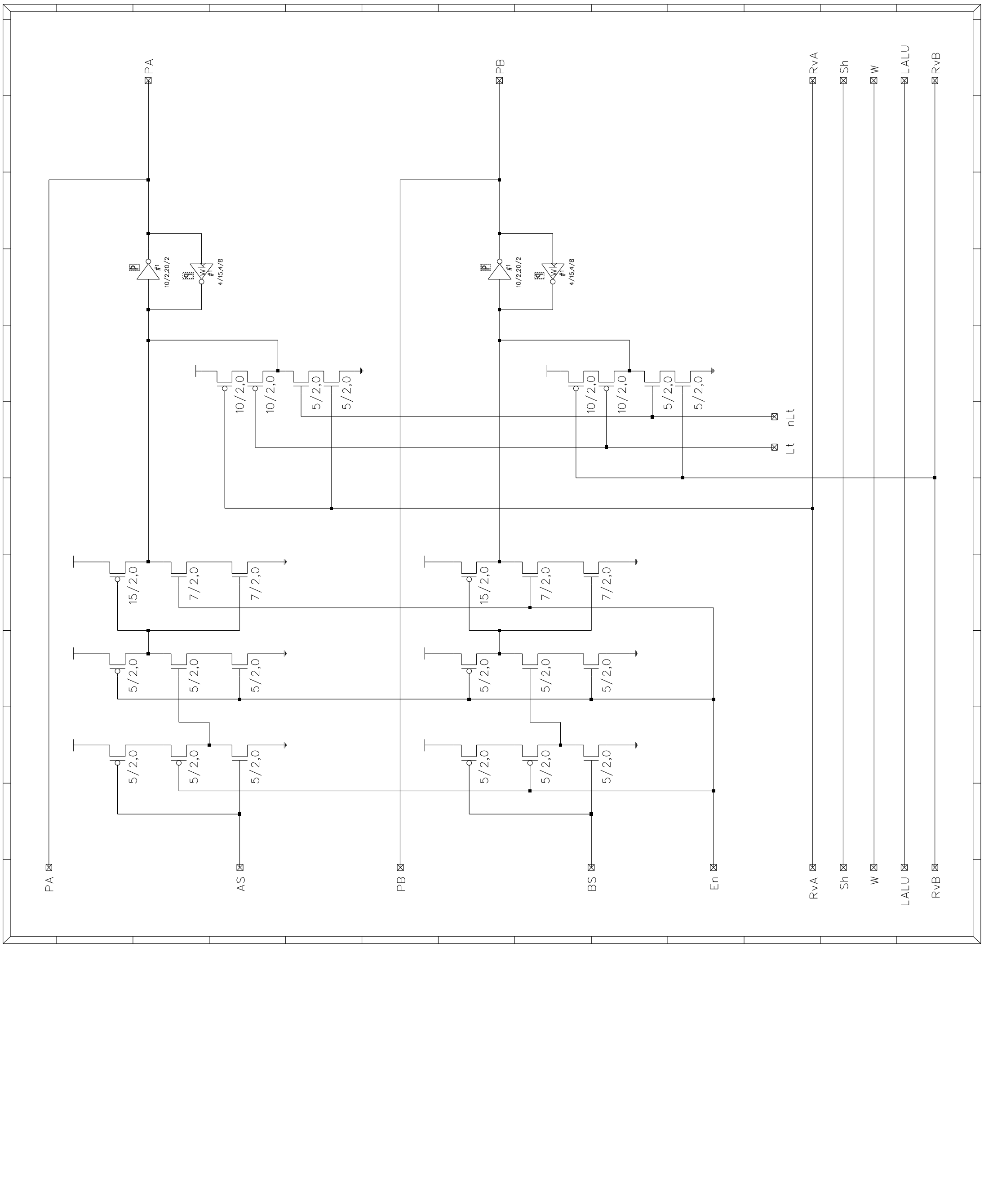
This appendix contains the schematics of some of the cell library for the *AMULET2* multiplier. Below is a list of the following appendix sections:

- **Datapath**
- **CellAB**
- **Term8**
- **Term8llow**
- **Mux**
- **Pipe**
- **SAdder**
- **CellSC**

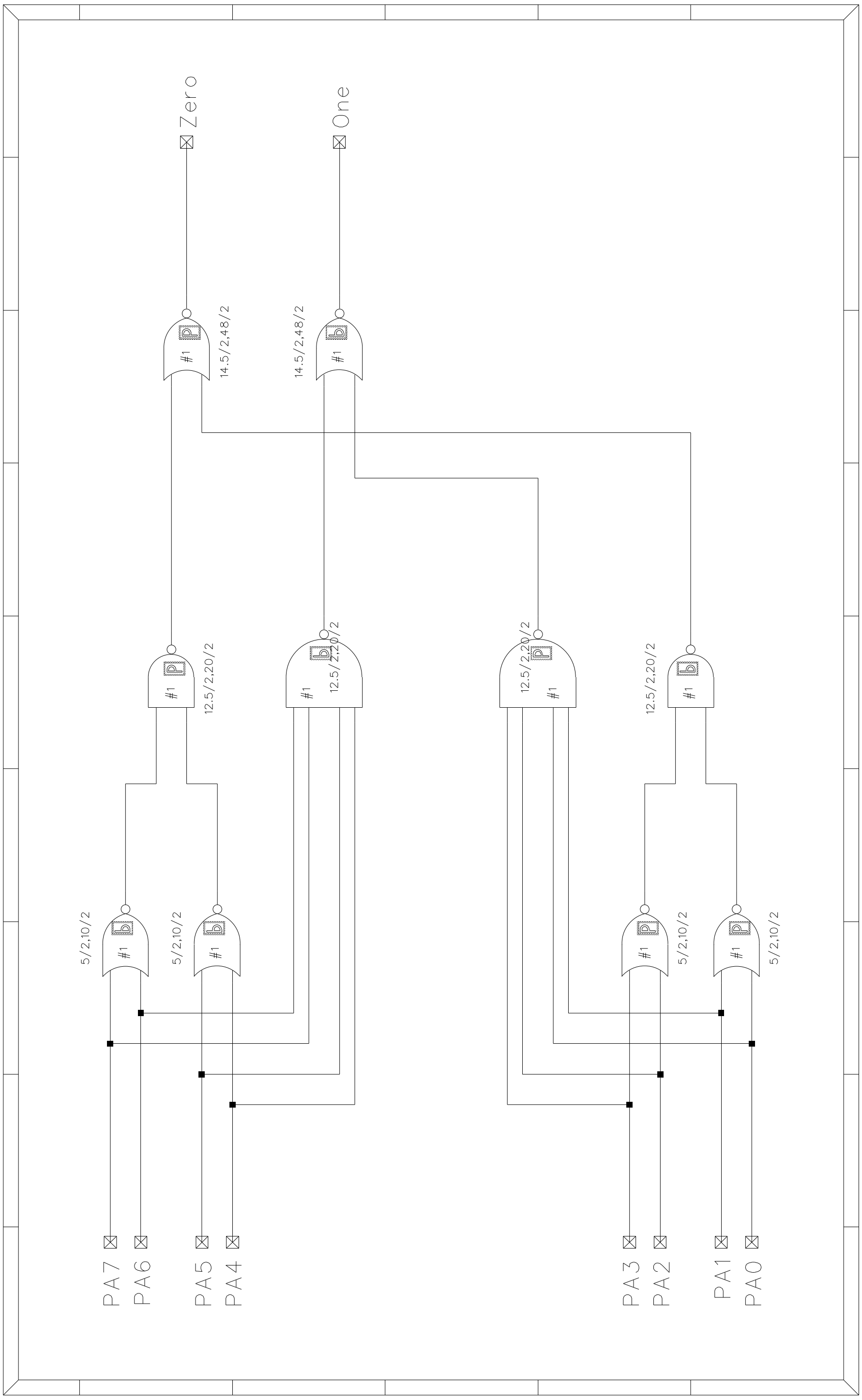
A.1 Datapath



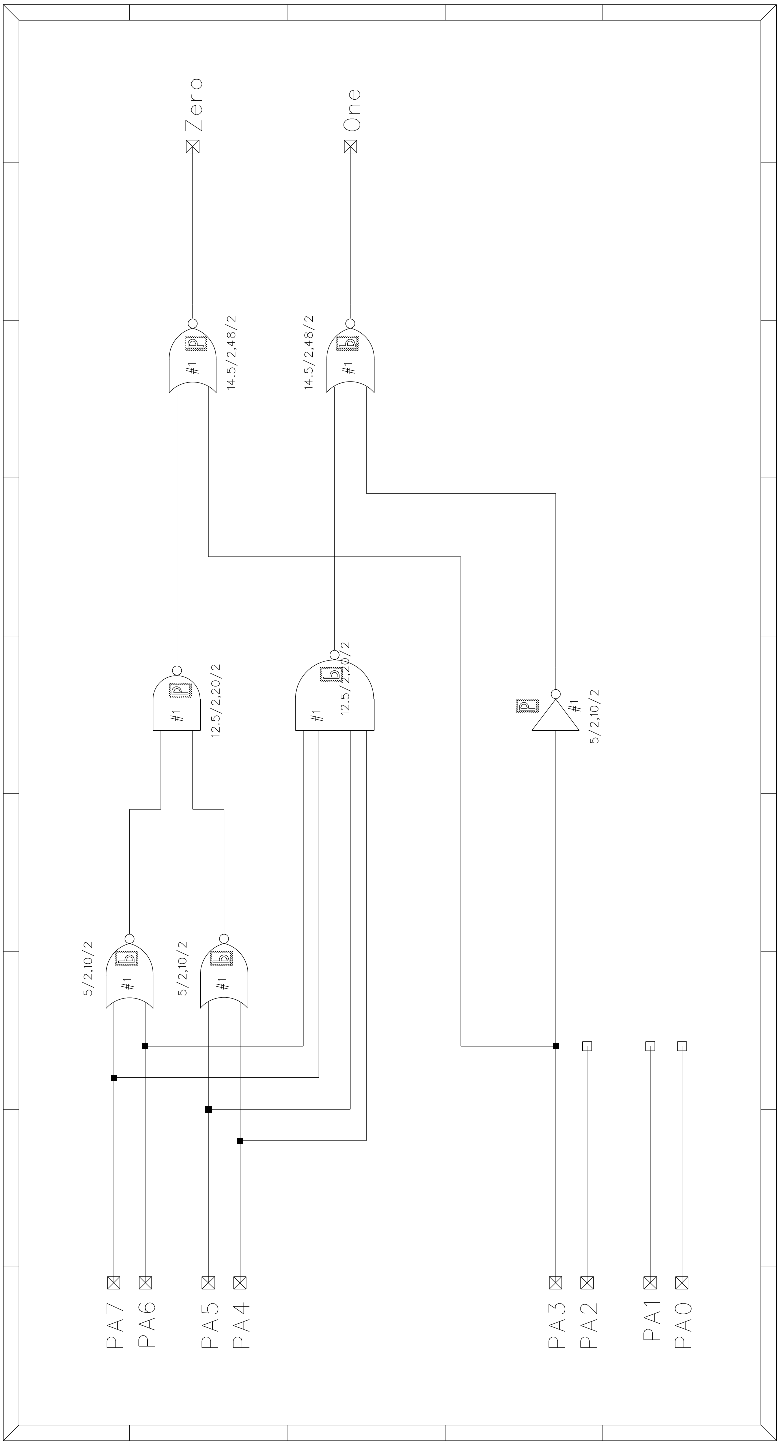
A.2 CellAB



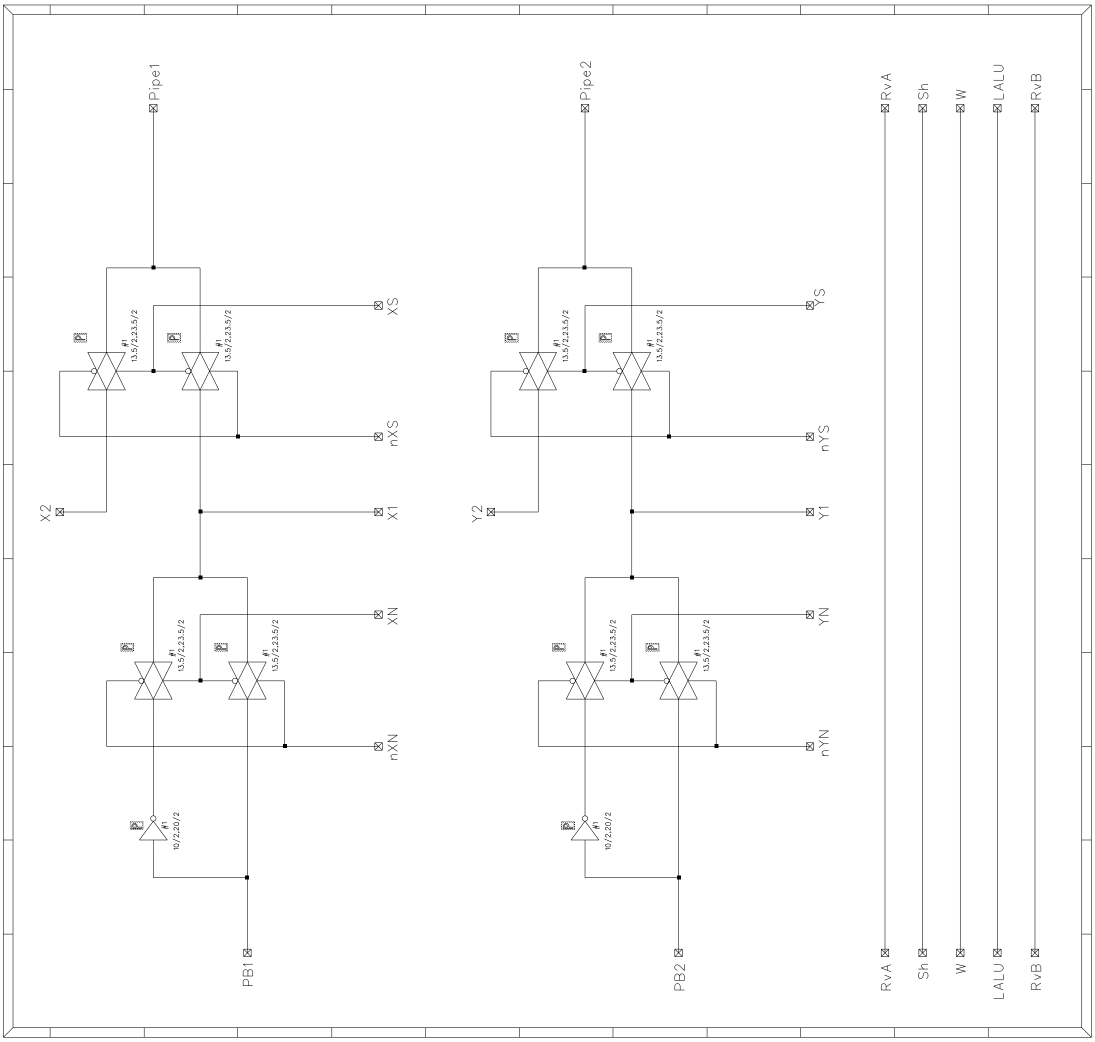
A.3 Term8


Zero
One
14.5/2.48/2
14.5/2.48/2
12.5/2.20/2
12.5/2.20/2
12.5/2.20/2
12.5/2.20/2
5/2.10/2
5/2.10/2
5/2.10/2
5/2.10/2
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0

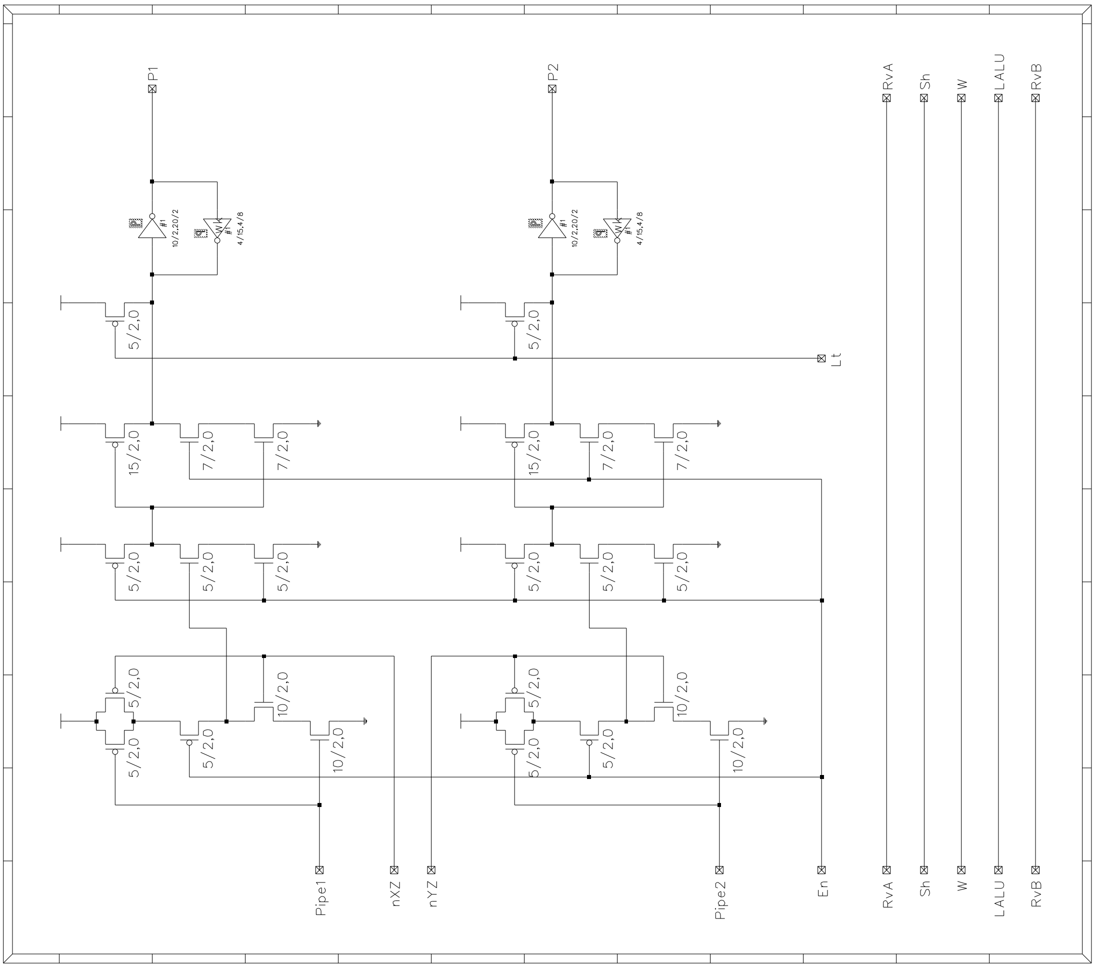
A.4 Term8low


Zero
One
14.5/2,48/2
14.5/2,48/2
12.5/2,20/2
12.5/2,20/2
5/2,10/2
5/2,10/2
5/2,10/2
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0

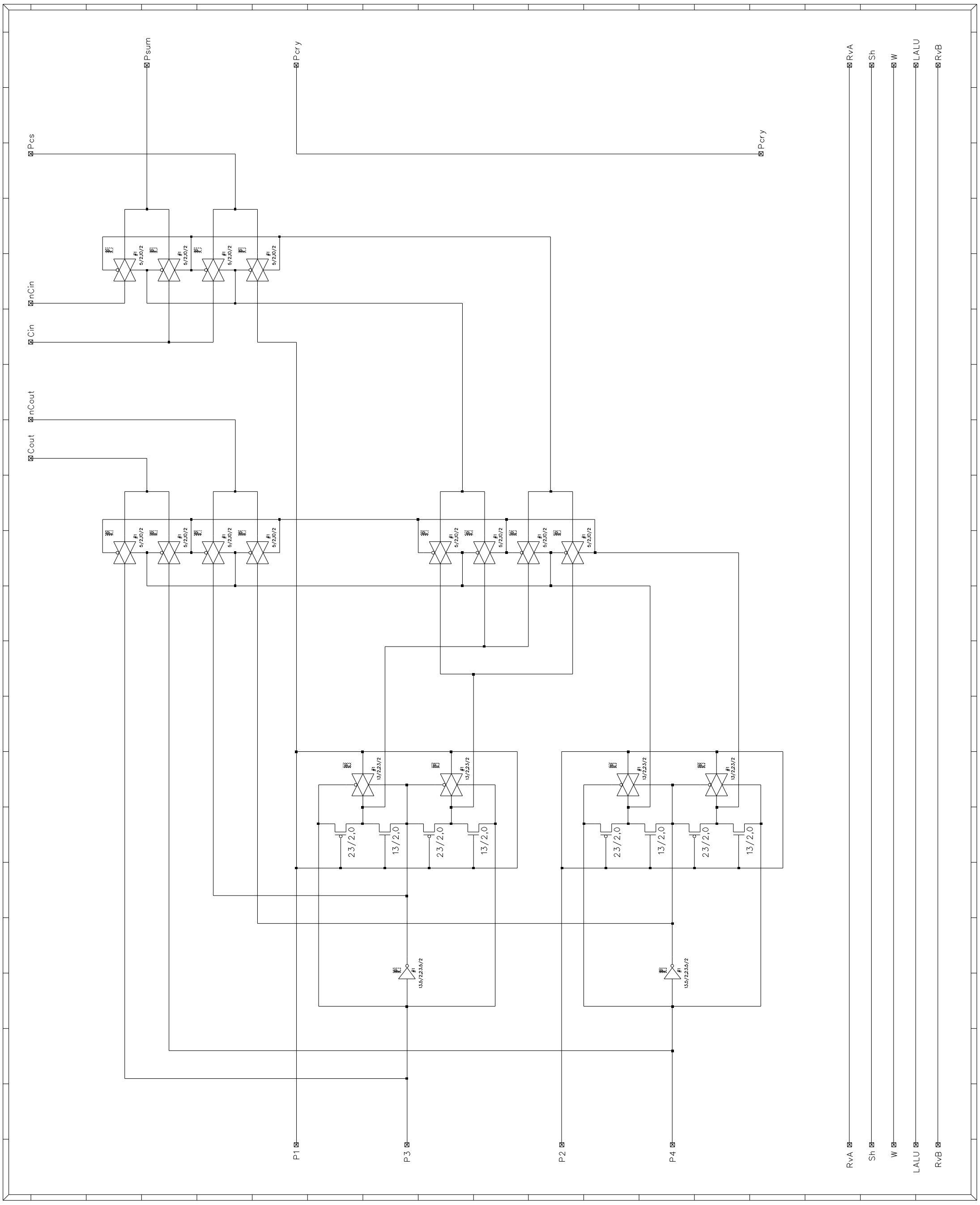
A.5 Mux


Pipe1
Pipe2
RvA
Sh
W
LALU
RvB
XS
nXS
X2
X1
XN
nXN
PB1
YS
nYS
Y2
Y1
YN
nYN
PB2
RvA
Sh
W
LALU
RvB

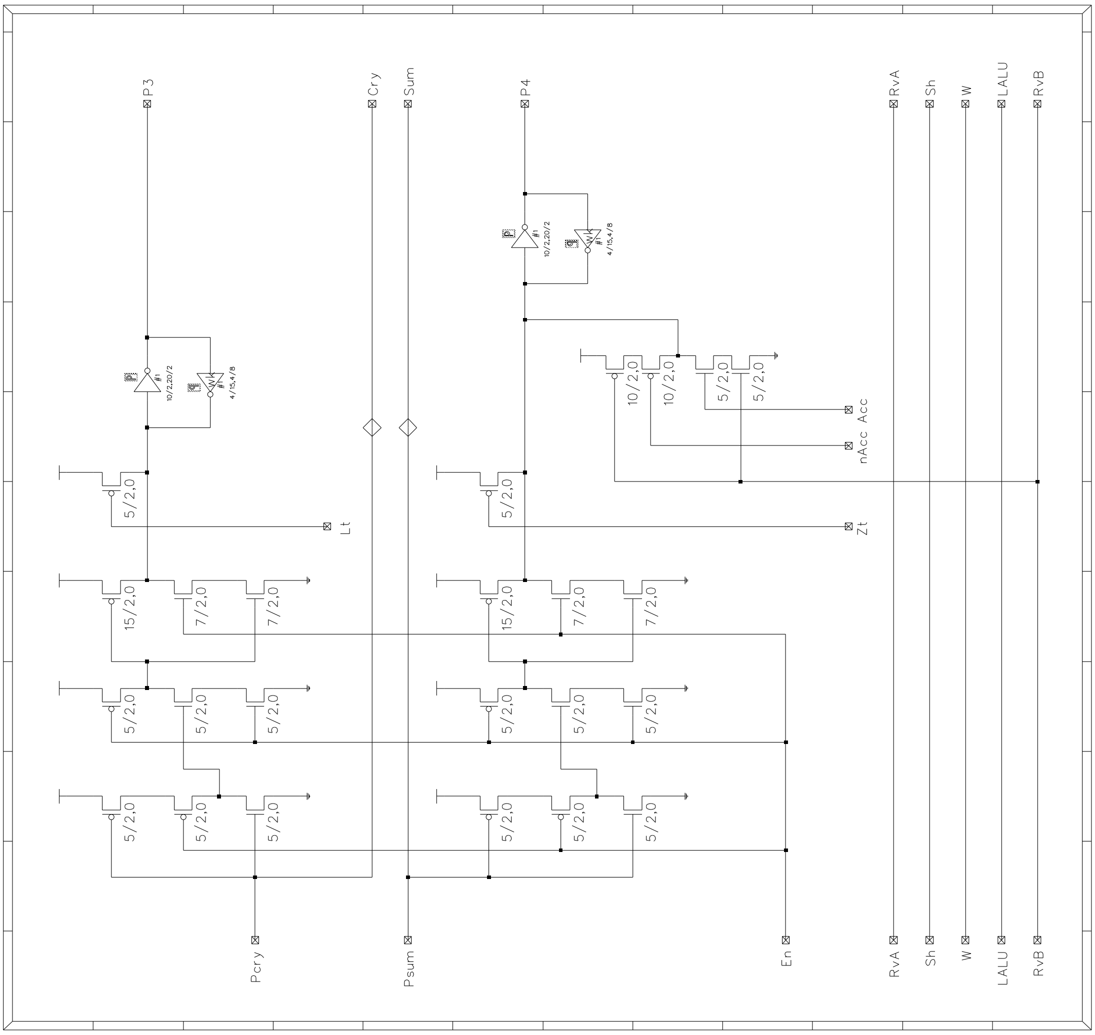
A.6 Pipe



A.7 SAdder


Psum
Pcry
Pcs
Pcry
nCin
Cin
nCout
Cout
RvA
Sh
W
LALU
RvB
23/2,0
13/2,0
P1
P3
P2
P4

A.8 CellSC


P3
Cry
Sum
P4
RvA
Sh
W
LALU
RvB
Acc
nAcc
Lt
Zt
Pcry
Psum
En
RvA
Sh
W
LALU
RvB

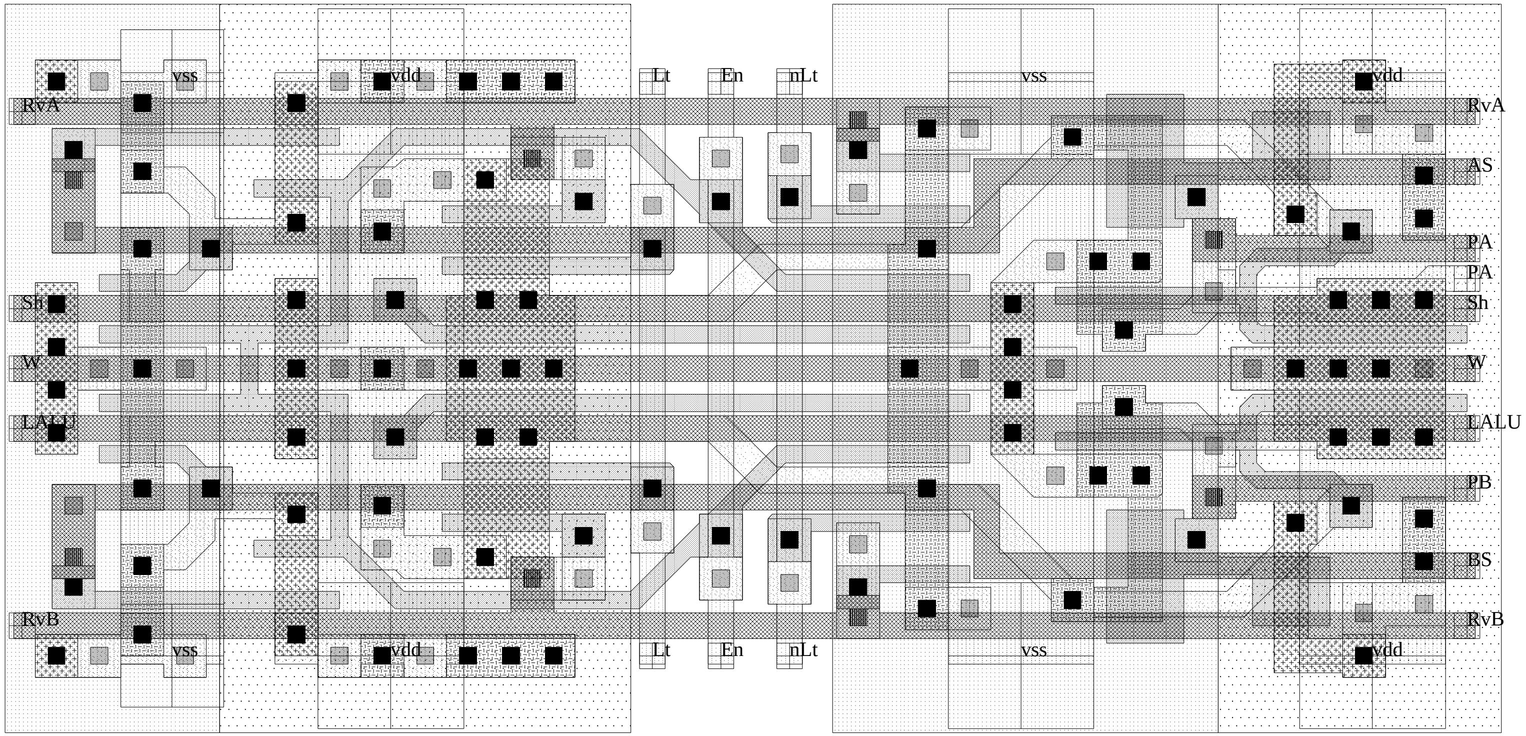
Layouts

B

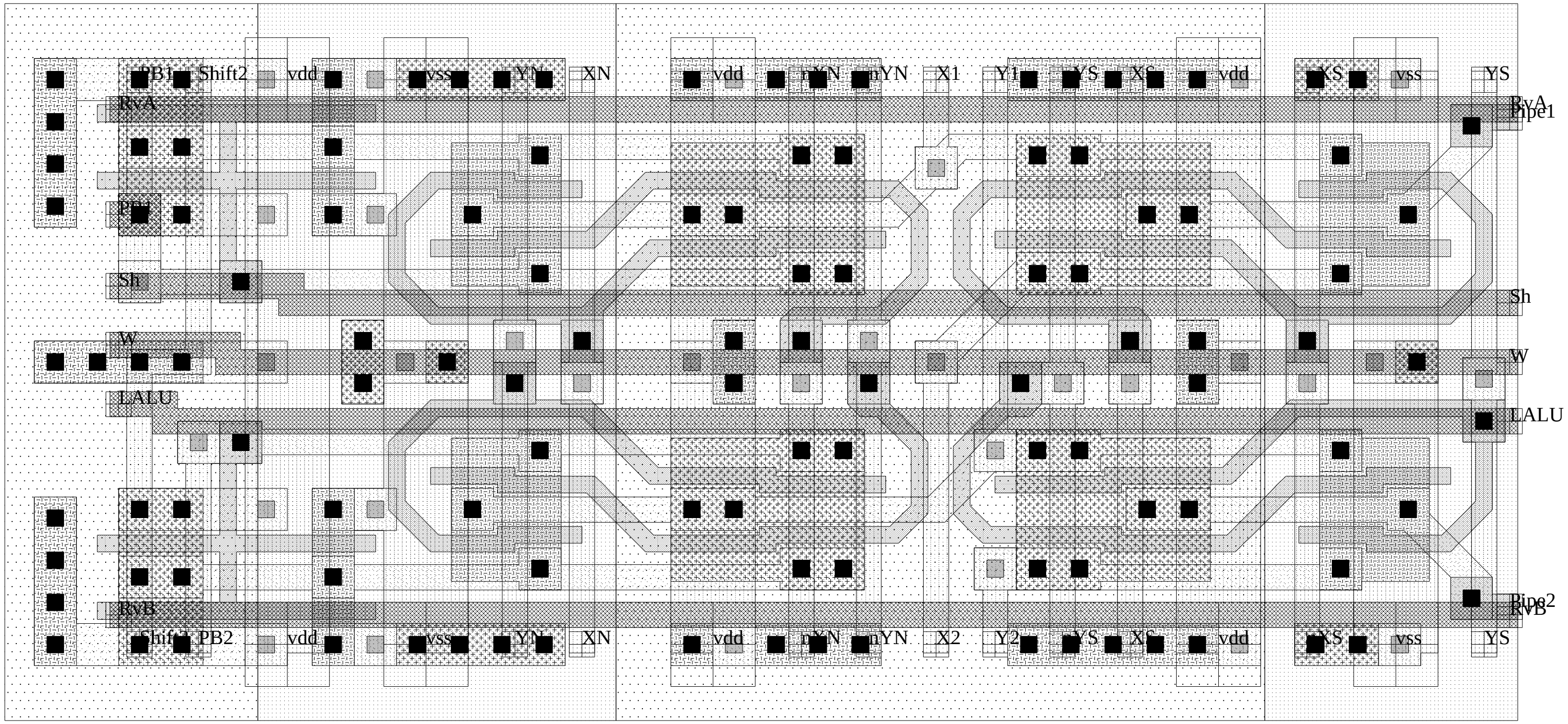
This appendix contains the layouts of some of the cell library for the *AMULET2* multiplier. Below is a list of the following appendix sections:

- **CellAB**
- **Mux**
- **Pipe**
- **SAdder**
- **CellSC**

Compass Design Automation plot [ly]CellAB, strip 1, sheet 1

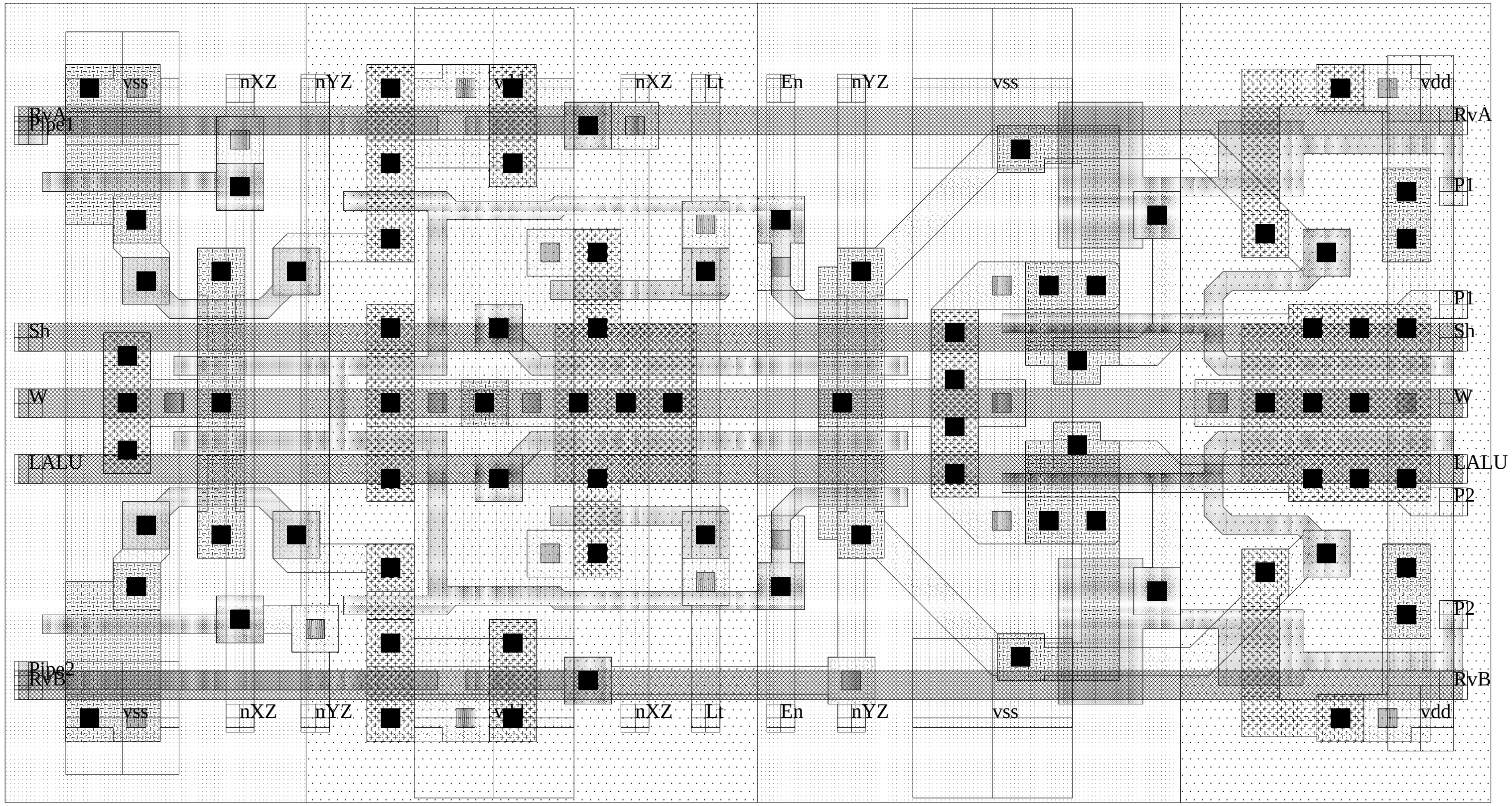


Compass Design Automation plot [ly]Mux, strip 1, sheet 1



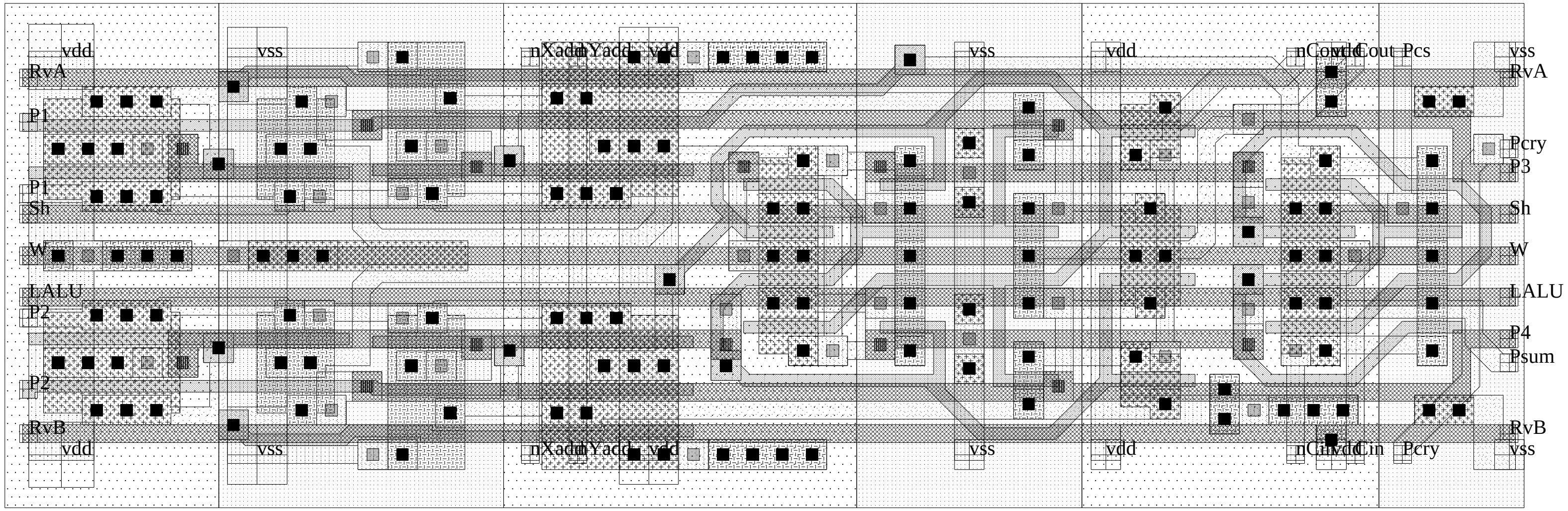
B.3 Pipe

Compass Design Automation plot [ly]Pipe, strip 1, sheet 1



B.4 SAdder

Compass Design Automation plot [ly]SAdder, strip 1, sheet 1



Compass Design Automation plot [ly]CellSC, strip 1, sheet 1