Modern ASIC Design

Dian Zhou (周电)
E. E. Department
The University of Texas at Dallas
USA

Chapter 4 Architecture Design

- Objectives
  - Datapath architecture
  - Parallel and sequential structure
  - Controller
  - Functional blocks
  - IPs
  - Interface
  - Time budget
  - A sample architecture of MSDAP

Introduction

- What is the architecture of an IC
  - Architecture determines the main functional blocks, their relationship with each other, and more importantly where and when high level operations are performed.
- Importance
  - A proper architecture is critical to ASIC chip performance and cost.
  - It is perhaps the most important criterion in distinguishing a good design from a poor one.
- Architecture design relies on the knowledge and experience of a designer,
  - It is still a matter of art that cannot be fully automated by EDA tools.
- Figure 4-1 indicates the phase of architecture design in the design flow and highlights the involved work.

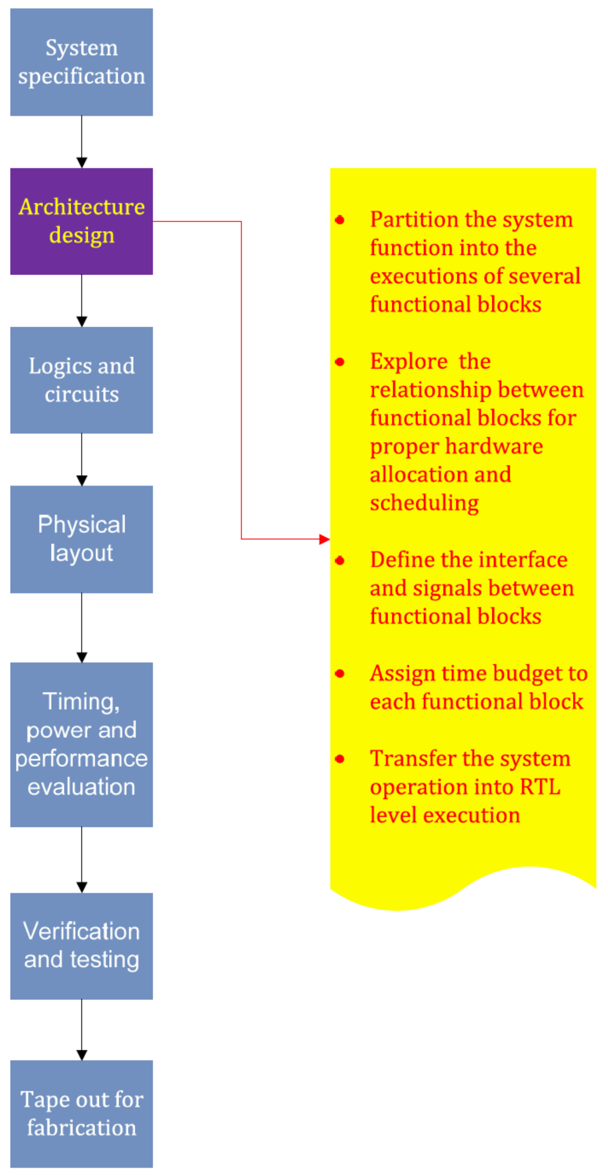

System specification
Architecture design
Logics and circuits
Physical layout
Timing, power and performance evaluation
Verification and testing
Tape out for fabrication
• Partition the system function into the executions of several functional blocks
• Explore the relationship between functional blocks for proper hardware allocation and scheduling
• Define the interface and signals between functional blocks
• Assign time budget to each functional block
• Transfer the system operation into RTL level execution

Figure 4-1 Architecture design phase in ASIC design flow and its task

- At architecture level, functional blocks or Intellectual Properties (IPs) are considered as basic components.
  - In this chapter we shall study how to model and connect these functional blocks.
- Operations of functional blocks and data flow will be coordinated by a Finite State Machine (FSM).
- The system architecture is finally described using VHDL (or Verilog) at the RTL level
  - It can be used for a clock cycle-by-cycle simulation, as well as for next phase synthesis and verification.
- At the end of this chapter, an example architecture of the MSDAP is presented in detail to illustrate architecture development.

Datapath Structure

- At the top level, a digital system can be partitioned into two parts: a datapath and a controller.
- The datapath is used to store, manipulate, and transfer data, and the controller provides the command for these operations.
  - Figure 4-2 illustrates a general datapath structure, where the controller is usually an FSM; ALU is an arithmetic logic unit consisting of an adder, multiplexer and shifter; input/output interface usually is made of shift registers. The memory and bus are used to store and transfer data, respectively.

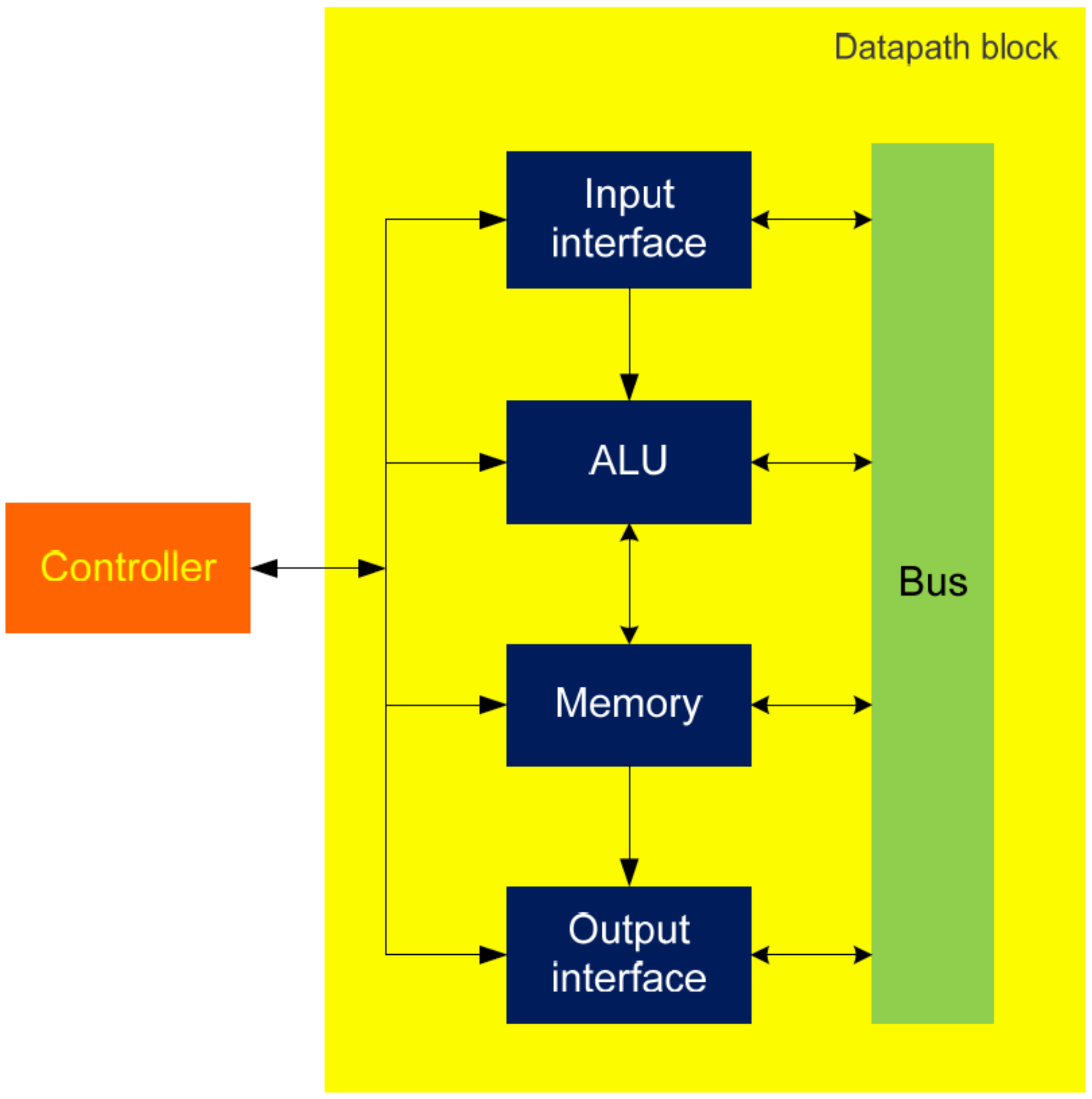


Figure 4-2 A general datapath structure

What to do in architecture design

- In an ASIC architecture each functional block must be defined precisely, including its functionality, input/output pins and signals, as well as the performance requirement in terms of clock cycles.
- An architecture designer partitions the system function into sub-functions, each of them realized by a functional block.
- The architecture designer also needs to specify the control signals of each block, which controls the corresponding operation and data flow.
- In many cases, one has to use the components from a company's existing cell library or IPs from other parties.
  - This adds additional constraints to the architecture design.

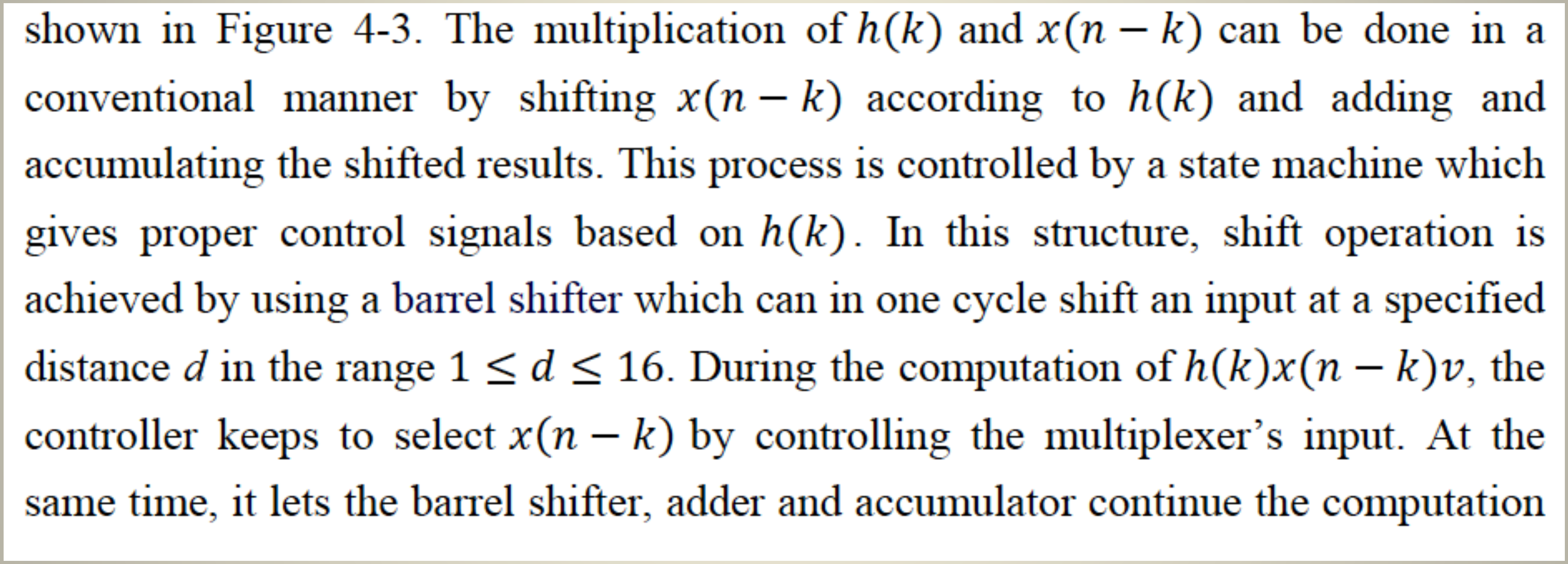
- The objectives of architecture design are twofold:
  - to minimize hardware resources which implement the specified function while satisfying the given performance requirement, and
  - to have a clean partition of the function and a clear relationship between the partitioned sub-functions which facilitates verification and testing.

Single Processor Sequential Structure

- In the single processor structure, the number of ALU units is minimal and other components are added, such as data and coefficient RAMs, and data path controllers.
- In the previous general datapath structure shown in Figure 4-2, functional blocks are represented at a very high level, and many details are omitted.
  - For instance, the ALU is a general arithmetic logic unit without implementation details.
  - Even for such a simple ALU there might be more than one implementation option.

- Two possible ALU architectures for the computation of linear convolution
  - A straight forward architecture without specific constraint
  - An architecture with the constraint proposed by the MSDAP

- A straight forward architecture


shown in Figure 4-3. The multiplication of $h(k)$ and $x(n-k)$ can be done in a conventional manner by shifting $x(n-k)$ according to $h(k)$ and adding and accumulating the shifted results. This process is controlled by a state machine which gives proper control signals based on $h(k)$. In this structure, shift operation is achieved by using a barrel shifter which can in one cycle shift an input at a specified distance d in the range $1 \leq d \leq 16$. During the computation of $h(k)x(n-k)v$, the controller keeps to select $x(n-k)$ by controlling the multiplexer's input. At the same time, it lets the barrel shifter, adder and accumulator continue the computation

of $h(k)x(n-k)v$. After completing the multiplication of $h(k)$ and $x(n-k)$, the multiplication of $h(k+1)x(n-k-1)$ is executed in a similar way. At this time the controller selects the next input $x(n-k-1)$ through the multiplexer, and continue to shift according to $h(k+1)$, and adds and accumulates the shifted $h(k+1)x(n-k-1)$ to the previous production result $h(k)x(n-k)$. All operations are controlled cycle-by-cycle by the state machine (an FSM) in the architecture. Connections between the controller and these components are not shown in the figure but can be easily added.

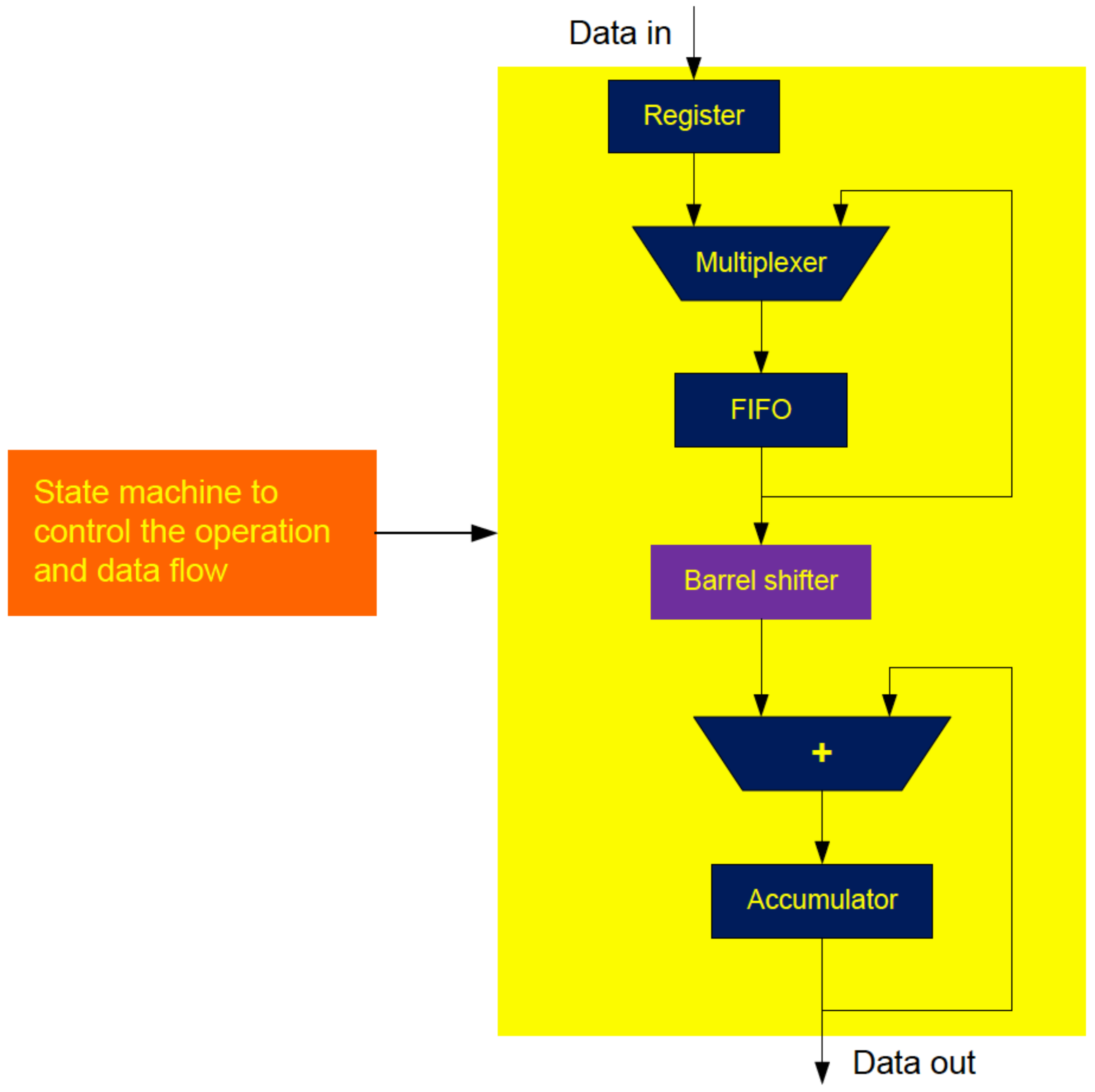
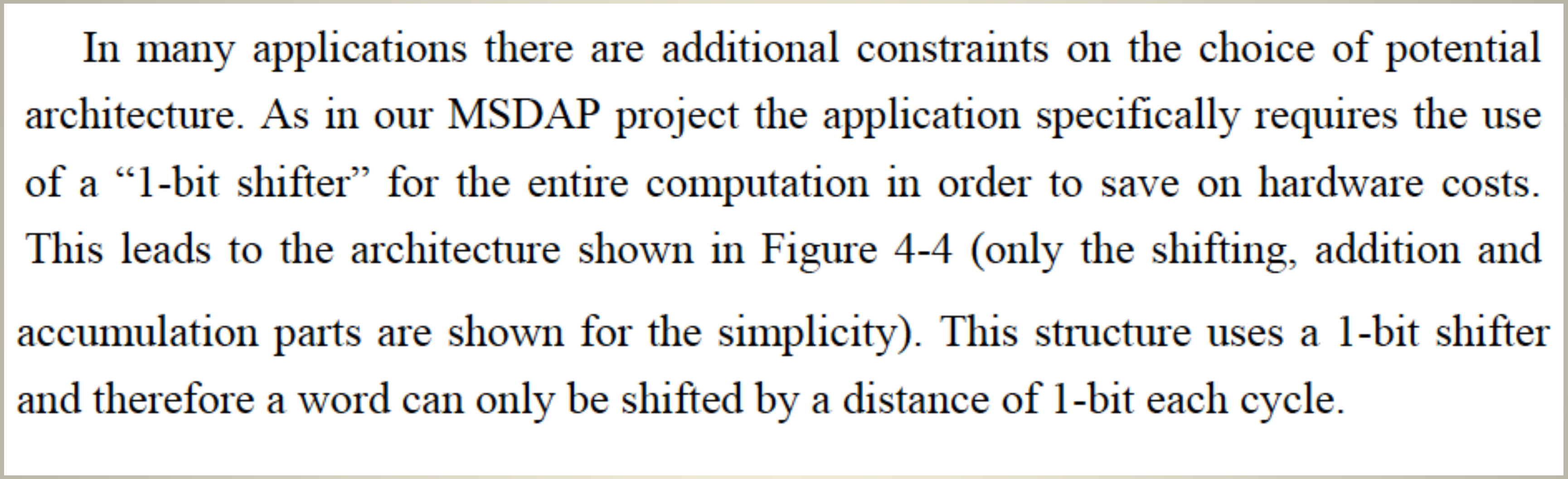


Figure 4-3 A single processor architecture using a barrel shifter for linear filter

- An architecture with the constraint



In many applications there are additional constraints on the choice of potential architecture. As in our MSDAP project the application specifically requires the use of a "1-bit shifter" for the entire computation in order to save on hardware costs. This leads to the architecture shown in Figure 4-4 (only the shifting, addition and accumulation parts are shown for the simplicity). This structure uses a 1-bit shifter and therefore a word can only be shifted by a distance of 1-bit each cycle.

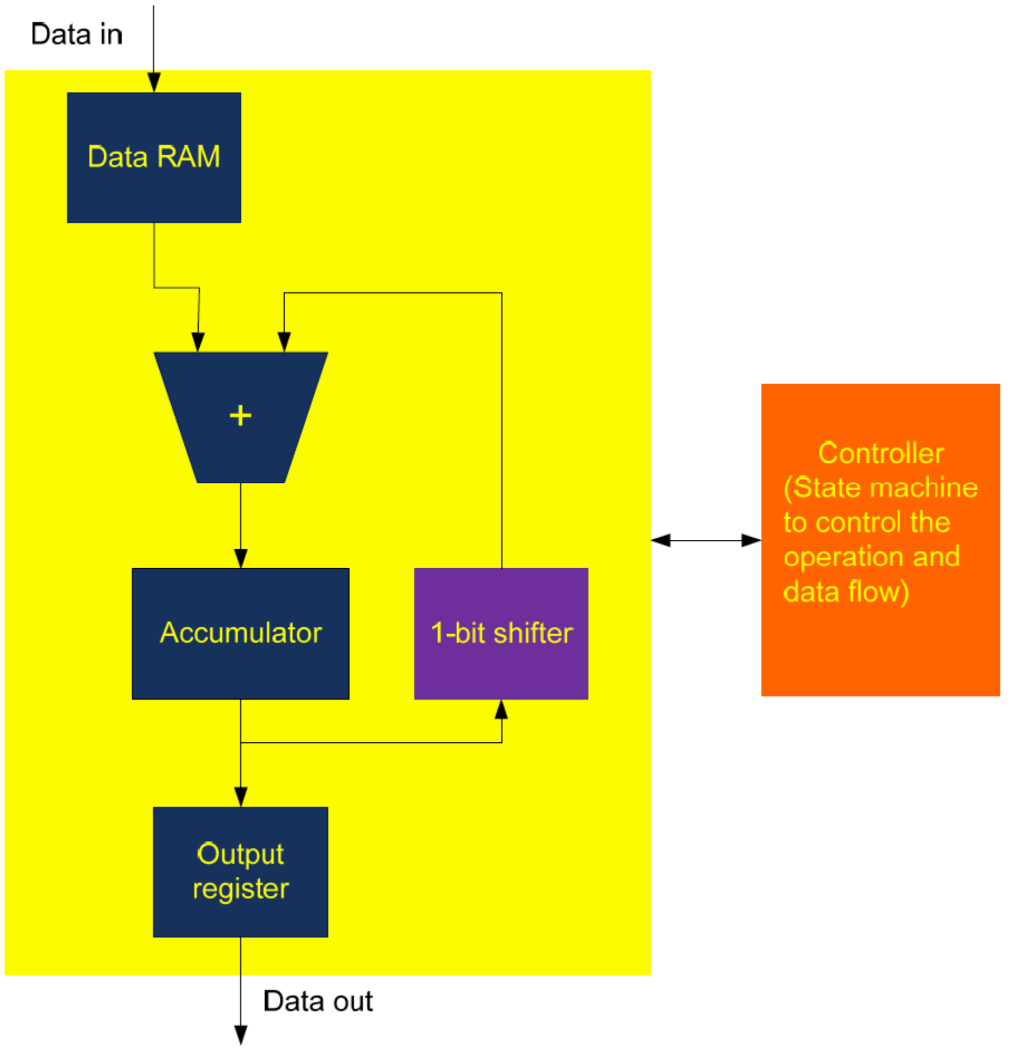
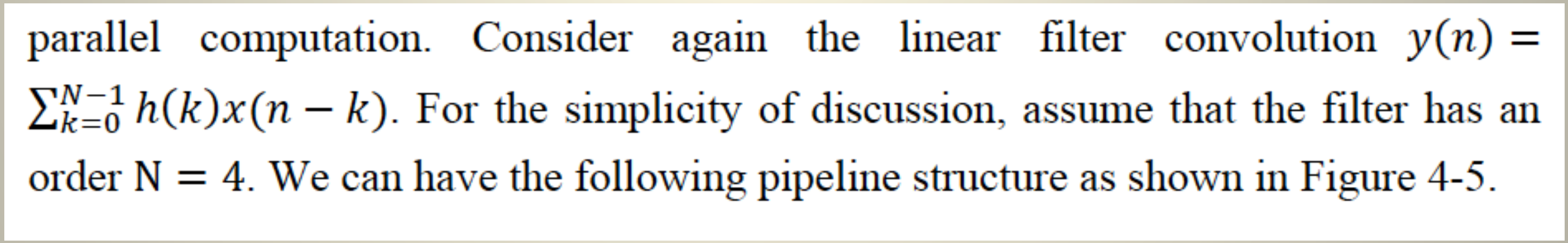

Data in
Data RAM
+
Accumulator
1-bit shifter
Output register
Data out
Controller (State machine to control the operation and data flow)

Figure 4-4 A single processor architecture using a 1-bit shifter for the MSDAP project

Multi-processor Parallel Structure

- Parallel properties are explored in the design when a single processor, for instance one ALU, doesn't satisfy the application computation speed requirement.
- We use a pipeline structure as an example to explore the property of the multiprocessor parallel computation.
  - An example



parallel computation. Consider again the linear filter convolution $y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$. For the simplicity of discussion, assume that the filter has an order N = 4. We can have the following pipeline structure as shown in Figure 4-5.

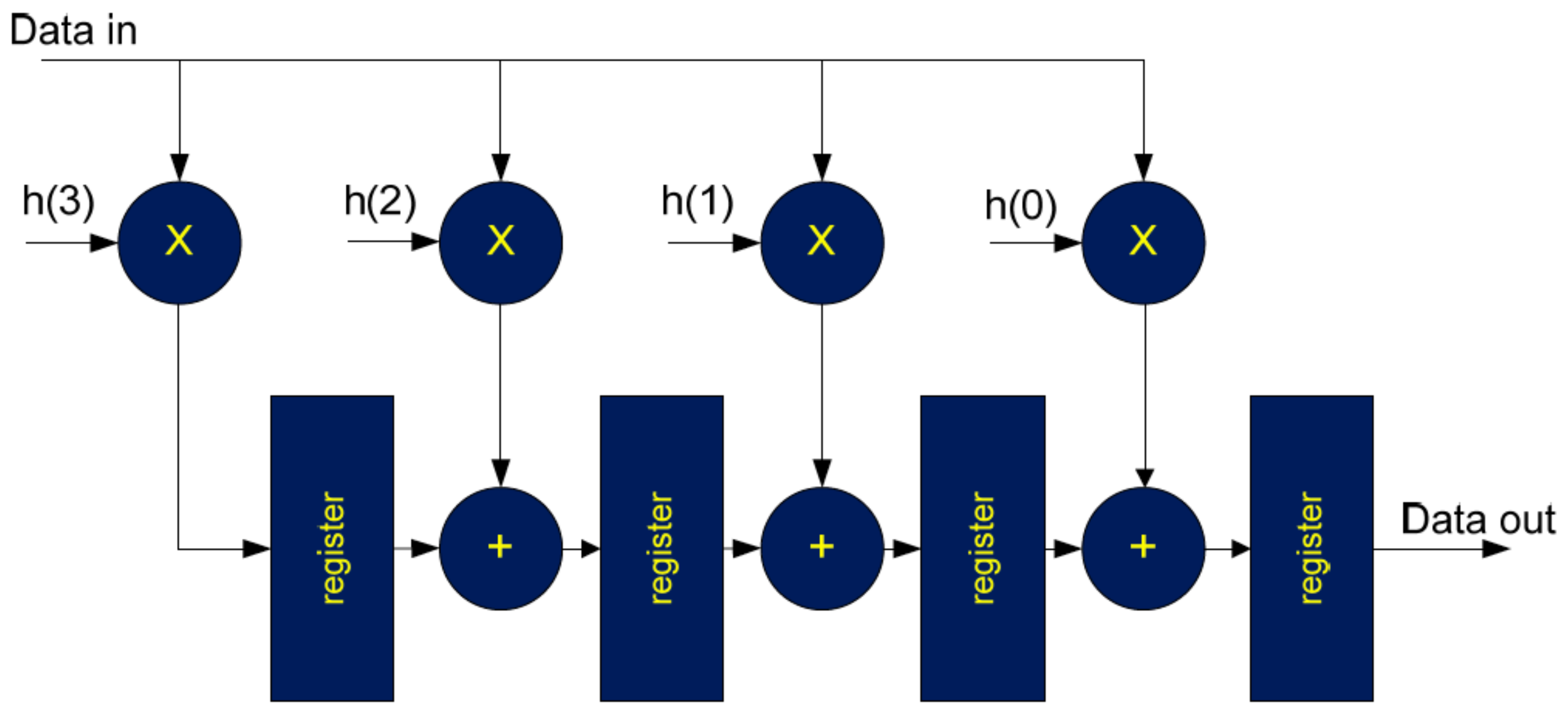
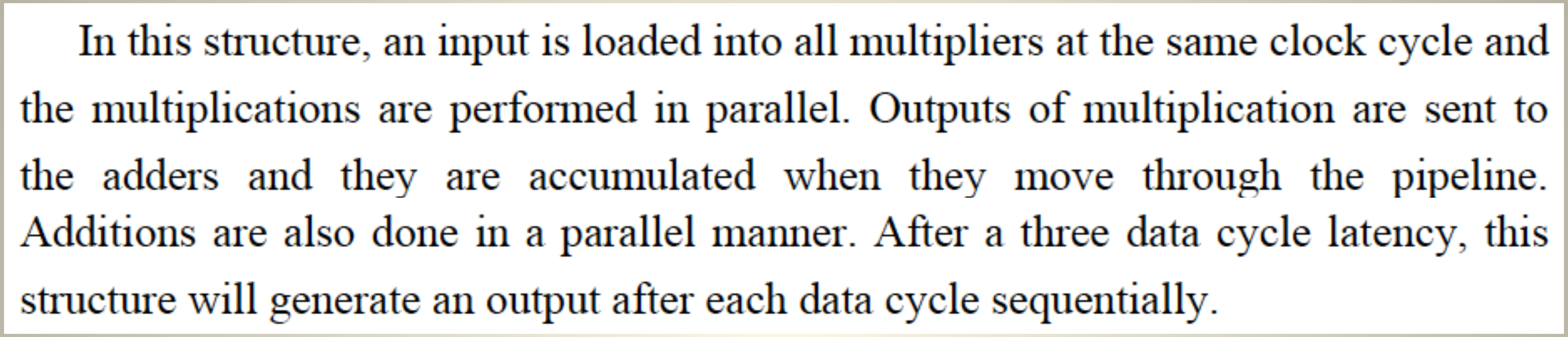


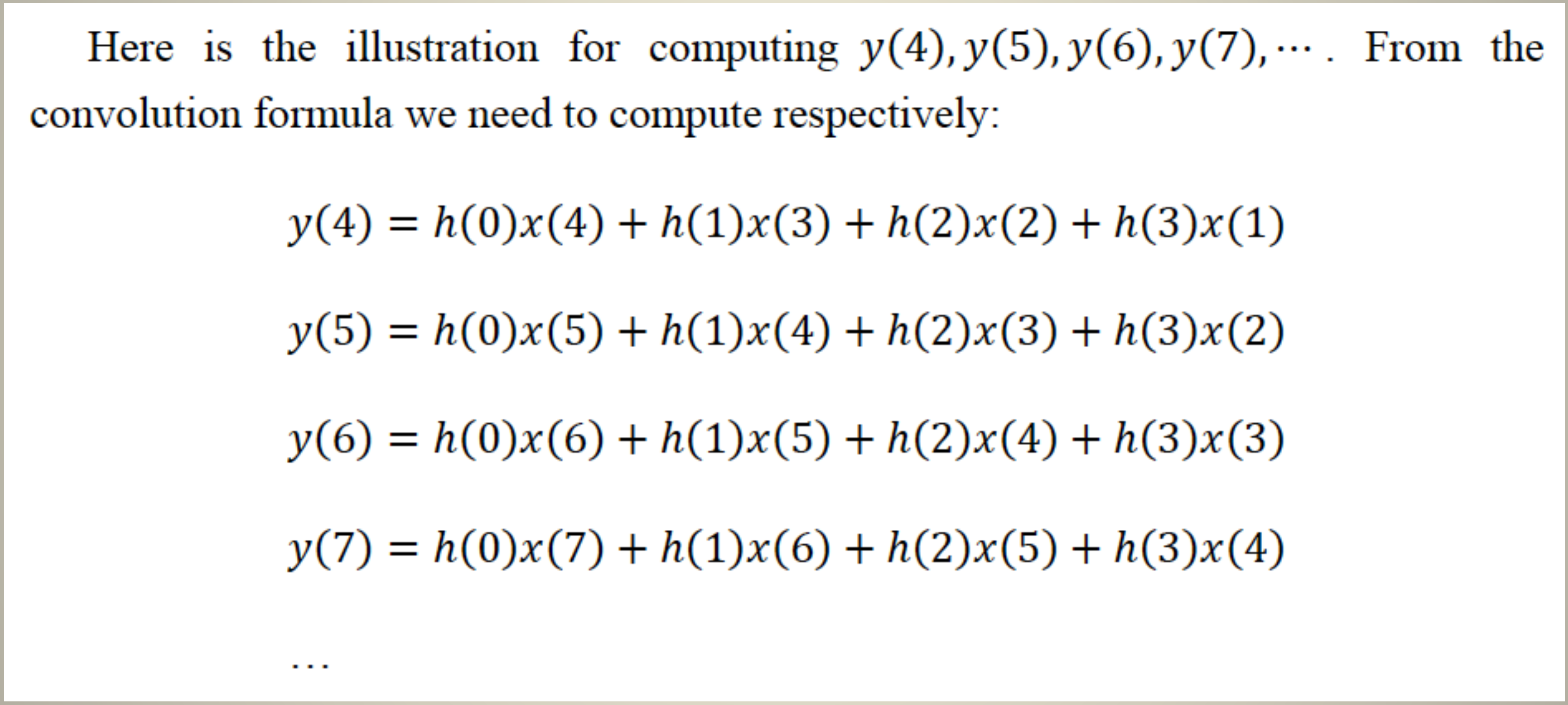
Figure 4-5 A multiprocessor pipeline structure for linear filter

- How it works



In this structure, an input is loaded into all multipliers at the same clock cycle and the multiplications are performed in parallel. Outputs of multiplication are sent to the adders and they are accumulated when they move through the pipeline. Additions are also done in a parallel manner. After a three data cycle latency, this structure will generate an output after each data cycle sequentially.

- Operation details



Here is the illustration for computing $y(4), y(5), y(6), y(7), \cdots$. From the convolution formula we need to compute respectively:

$$y(4) = h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1)$$

$$y(5) = h(0)x(5) + h(1)x(4) + h(2)x(3) + h(3)x(2)$$

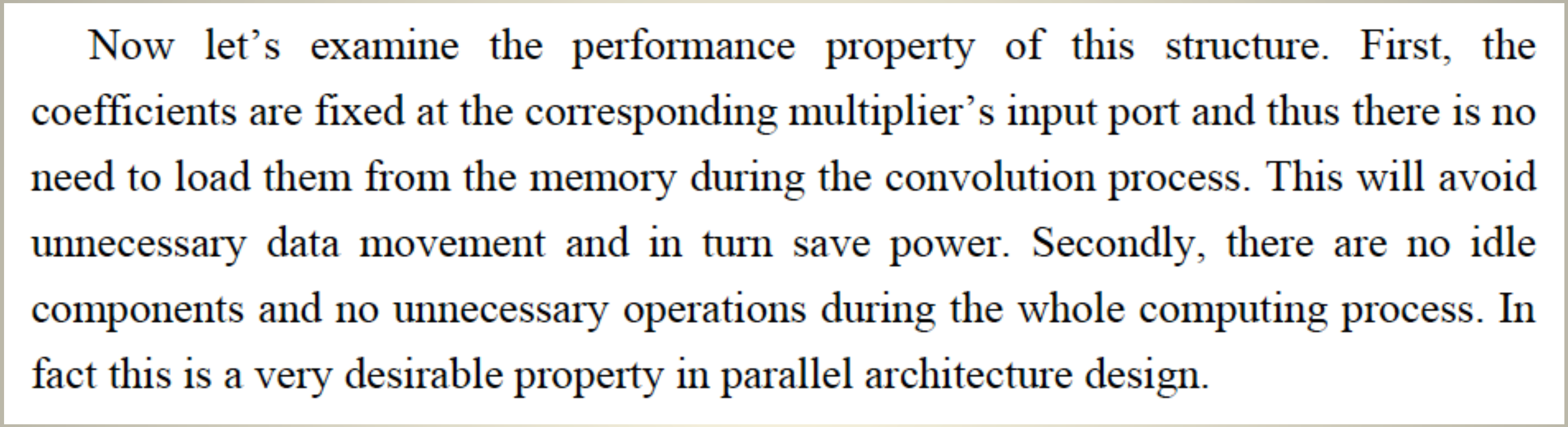
$$y(6) = h(0)x(6) + h(1)x(5) + h(2)x(4) + h(3)x(3)$$

$$y(7) = h(0)x(7) + h(1)x(6) + h(2)x(5) + h(3)x(4)$$

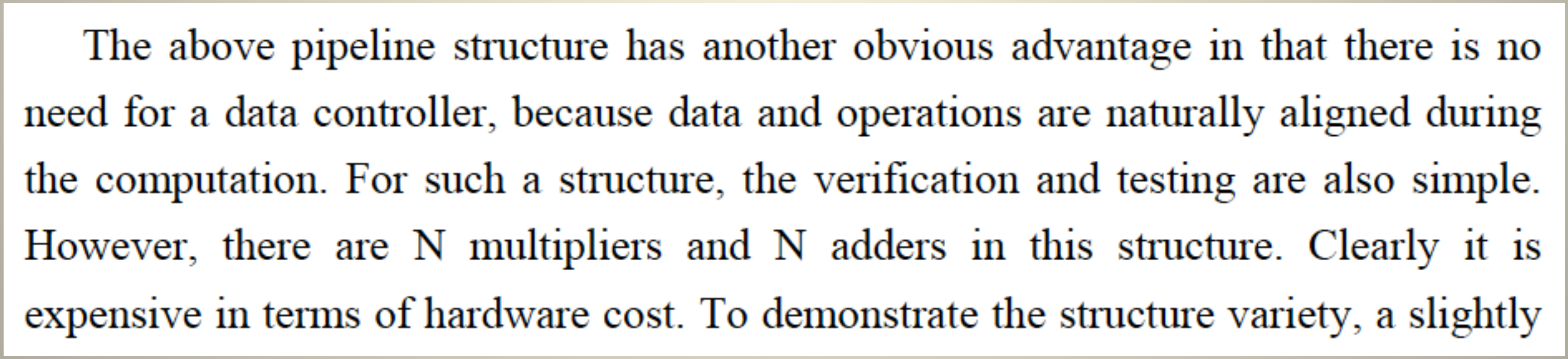
$$\dots$$

Prior to sending in data inputs, coefficients $h(0), h(1), h(2)$ and $h(3)$ are loaded into the multiplier's input ports as shown in the figure. After $x(4)$ is loaded into the data input line, $x(4)h(3)$, $x(4)h(2)$, $x(4)h(1)$ and $x(4)h(0)$ are performed in parallel by four multipliers. Note that the second register from the right had previously stored the result $h(1)x(3) + h(2)x(2) + h(3)x(1)$, which together with the newly calculated $h(0)x(4)$ will become the inputs to the rightmost adder. After the addition, $y(4)$ is obtained which will be sent into the rightmost register for output. The computation of $y(5), y(6), y(7), \cdots$, will be done in a similar fashion.

- Its properties and advantages



Now let's examine the performance property of this structure. First, the coefficients are fixed at the corresponding multiplier's input port and thus there is no need to load them from the memory during the convolution process. This will avoid unnecessary data movement and in turn save power. Secondly, there are no idle components and no unnecessary operations during the whole computing process. In fact this is a very desirable property in parallel architecture design.



The above pipeline structure has another obvious advantage in that there is no need for a data controller, because data and operations are naturally aligned during the computation. For such a structure, the verification and testing are also simple. However, there are N multipliers and N adders in this structure. Clearly it is expensive in terms of hardware cost. To demonstrate the structure variety, a slightly

- To demonstrate the structure variety, a slightly different version can be introduced as shown in Figure 4-6.

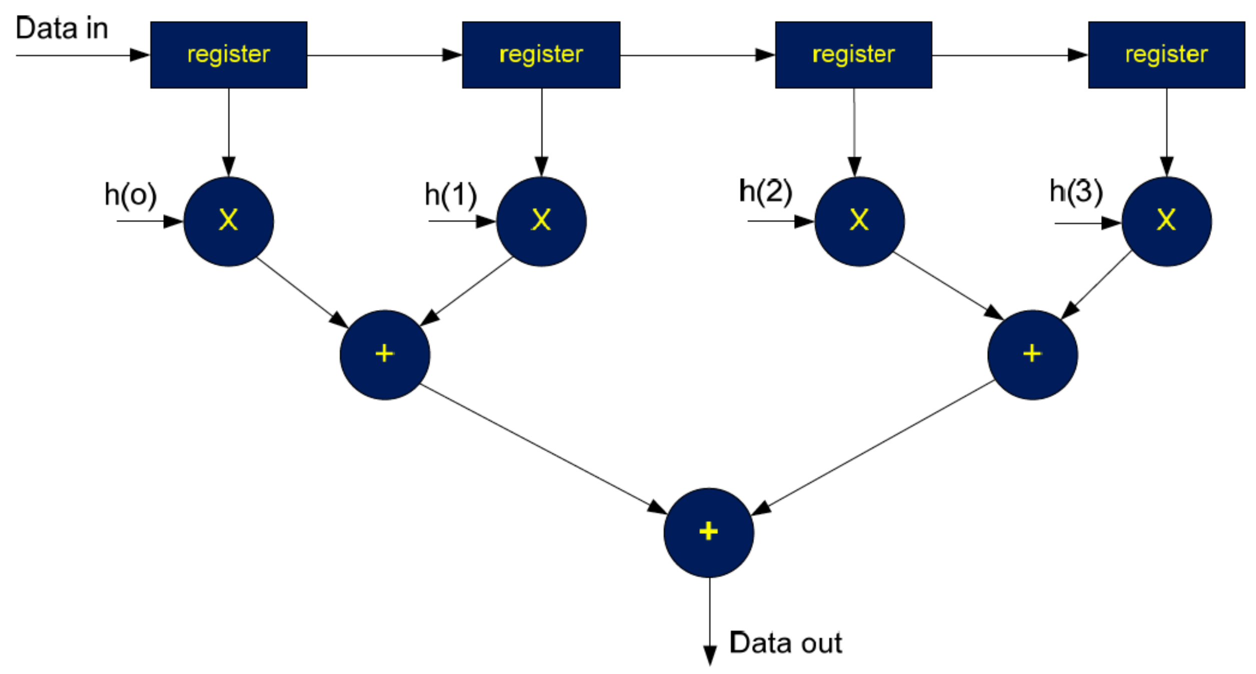
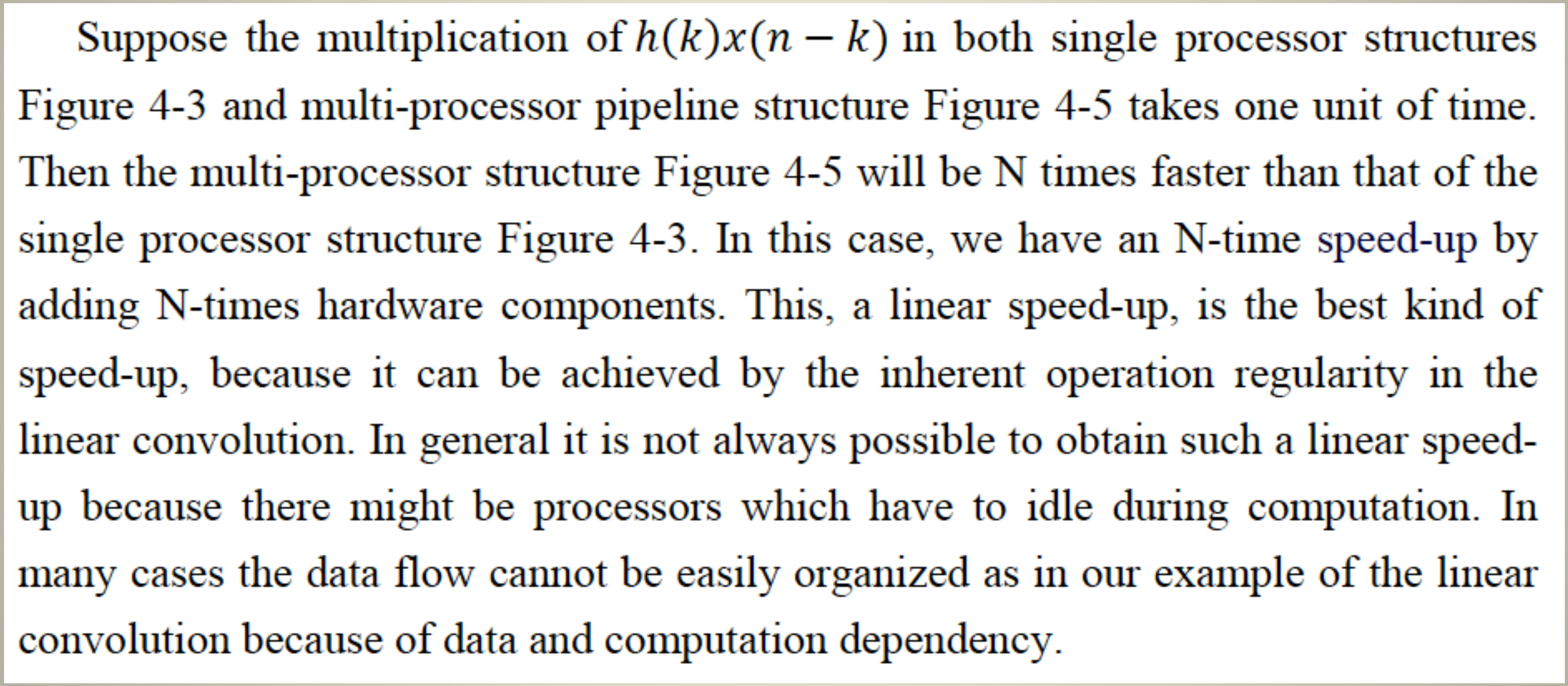


Figure 4-6 A different multiprocessor pipeline structure for the linear filter

- Speed-up obtained by parallel structure



Suppose the multiplication of $h(k)x(n-k)$ in both single processor structures Figure 4-3 and multi-processor pipeline structure Figure 4-5 takes one unit of time. Then the multi-processor structure Figure 4-5 will be N times faster than that of the single processor structure Figure 4-3. In this case, we have an N-time speed-up by adding N-times hardware components. This, a linear speed-up, is the best kind of speed-up, because it can be achieved by the inherent operation regularity in the linear convolution. In general it is not always possible to obtain such a linear speed-up because there might be processors which have to idle during computation. In many cases the data flow cannot be easily organized as in our example of the linear convolution because of data and computation dependency.

- The choice between single and multi-processor structure will mainly be determined by the cost, provided that both can accomplish the specified task.
  - For instance, the MSDAP project not only uses a single processor structure, but further uses a 1-bit shifter to reduce the cost, because today's IC speeds can easily satisfy such computation needs.

Functional Blocks and IPs

- We have seen that the datapath structure consisted of functional blocks as the basic circuit elements.
- In real design practice, the complexity of these functional blocks may vary from a simple adder to a powerful microprocessor, such as an ARM core in some ASIC designs.
- Regardless of whether it is a simple or a complex functional block, it needs to be completely defined in a similar way as we define a normal ASIC chip.

- The definition should include the following basic information:
  - Functionality
  - Input/output pins
  - Signals
  - Interface requirements
  - Layout area and shape specifications
  - Performance requirements
  - Other requirements depending on the application settings

- A general view of a functional block

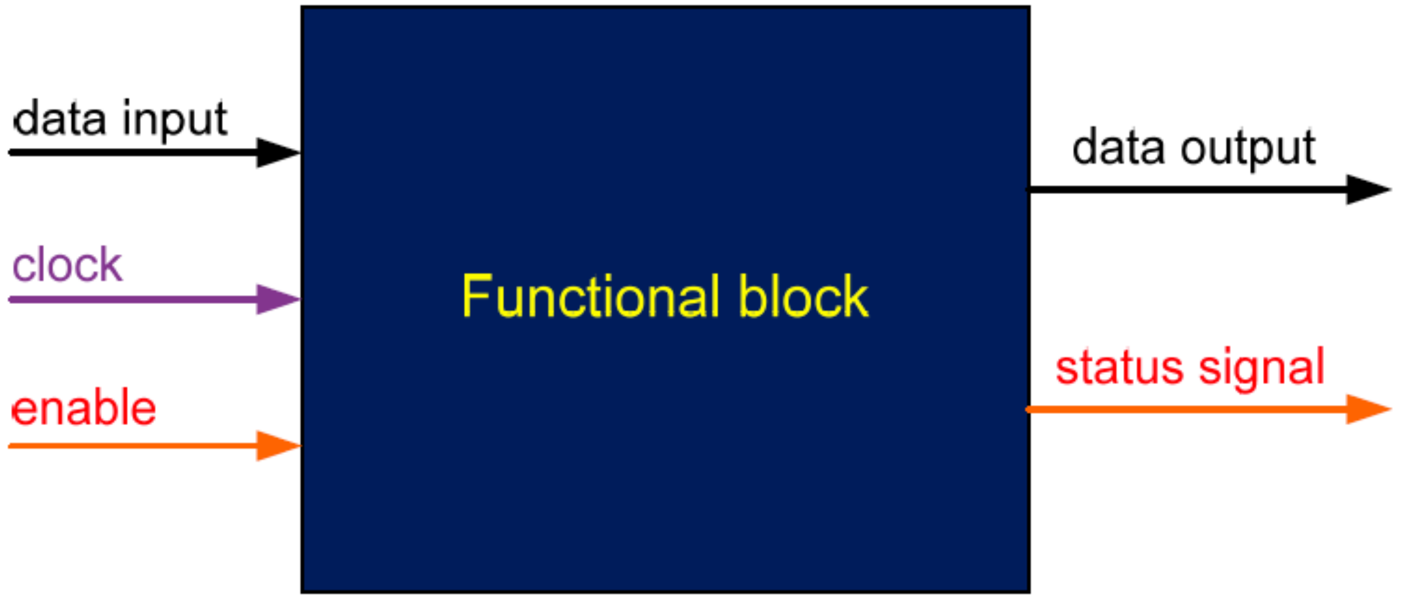


Figure 4-7 A general view of the functional block

IP Cores

- In electronic design one can use the existing functional block provided by the third party, called semiconductor intellectual property core, IP core, or IP block.
- This is a reusable unit of logic, cell, or chip layout design that is the intellectual property of another party.
- The term is derived from the licensing of the patent and source code copyright intellectual property rights that concern the design.
- IP cores are frequently used as building blocks within ASIC chip designs.

- IP cores in the electronic design industry have had a profound impact on the design of systems on a chip.
  - For instance, ARM cores have been widely used in many cell phones and multimedia applications.
- By licensing a design multiple times, IP core providers spread the cost of development among multiple chip makers.
  - IP cores are the re-usable functional blocks within the entire IC industry, instead of within a single IC company.

- IP cores for standard processors, interfaces, and internal functions enable chip makers to put more resources into developing the differentiating features of their chips.
  - As a result, chip makers are able to develop new innovations faster.

- The licensing and use of IP cores in chip design became common practice in the 1990s.
  - The microprocessor cores of ARM Holdings are recognized as some of the first widely licensed IP cores.
- It has been a popular choice for many digital applications because of its competitive time to market.
  - Using such IPs we can significantly improve design efficiency and product reliability since these IP cores have been optimized and tested.

Soft core

- IP cores are typically offered as synthesizable RTL netlists.
  - Synthesizable cores are delivered in a hardware description language such as Verilog or VHDL.
  - These are analogous to high level languages such as C in the field of computer programming.
- IP cores delivered to chip makers as RTL allow chip designers to modify designs if necessary, though many IP vendors offer no warranty or support for modified designs
  - Thus modifications require extreme caution and detailed verification.

- IP cores are also sometimes offered as generic gate-level netlists.
  - The netlist is a Boolean-algebra representation of the IP's logical function implemented as generic gates or process specific cells. A gate-level netlist is analogous to an assembly-code listing in the field of computer programming.
  - A netlist gives the IP core vendor reasonable protection against reverse engineering.
- Both netlist and synthesizable cores are called soft cores, because both allow a synthesis, placement and route (SPR) design flow.

- The advantage of an IP core implemented as a generic gate is that it is portable to other process technologies, if the technology is compatible.
- Because different processes might be used to implement the gates in the netlist, the timing of the logic circuit may change.
  - A careful timing analysis, both static and dynamic, should be followed to ensure the correct operation.

Hard core

- Analog and mixed-signal logics are generally defined as lower-level physical descriptions.
  - Hence, analog IP (PLLs, DAC, ADC, etc.) are provided to chip makers in transistor-layout format such as in GDSII (GDSII 2011).
- Digital IP cores are sometimes offered in a layout format as well. Such cores, whether analog or digital, are called hard cores (or hard macros), because the core's application function cannot be meaningfully modified by chip designers.
  - Transistor layouts must obey the target foundry's process design rules, and hence, hard cores delivered for one foundry's process cannot be easily ported to a different process or foundry.

- Merchant foundry operators offer a variety of hard-macro IP functions built for their own foundry process, helping to ensure customer lock-in.
- Hard cores, by the nature of their low-level representation, offer better predictability of chip performance in terms of timing performance and area.

Functional Blocks in the MSDAP Architecture

- With the presented specs and signal definitions in Chapter 3 for the MSDAP, we now show an example architecture from the project report (Patel and Hernandez-Garduno 2009).
- A high level abstraction is presented in Figure 4-8 to facilitate the discussion of defining the functionality of each block.
  - The timing information related to the detail clock is not complete since we still need more detailed structures of ALU to budget the number of operations.
- Each block is defined as required in Figure 4-7 so that its functionality, input/output, status and control signals are precisely stated.

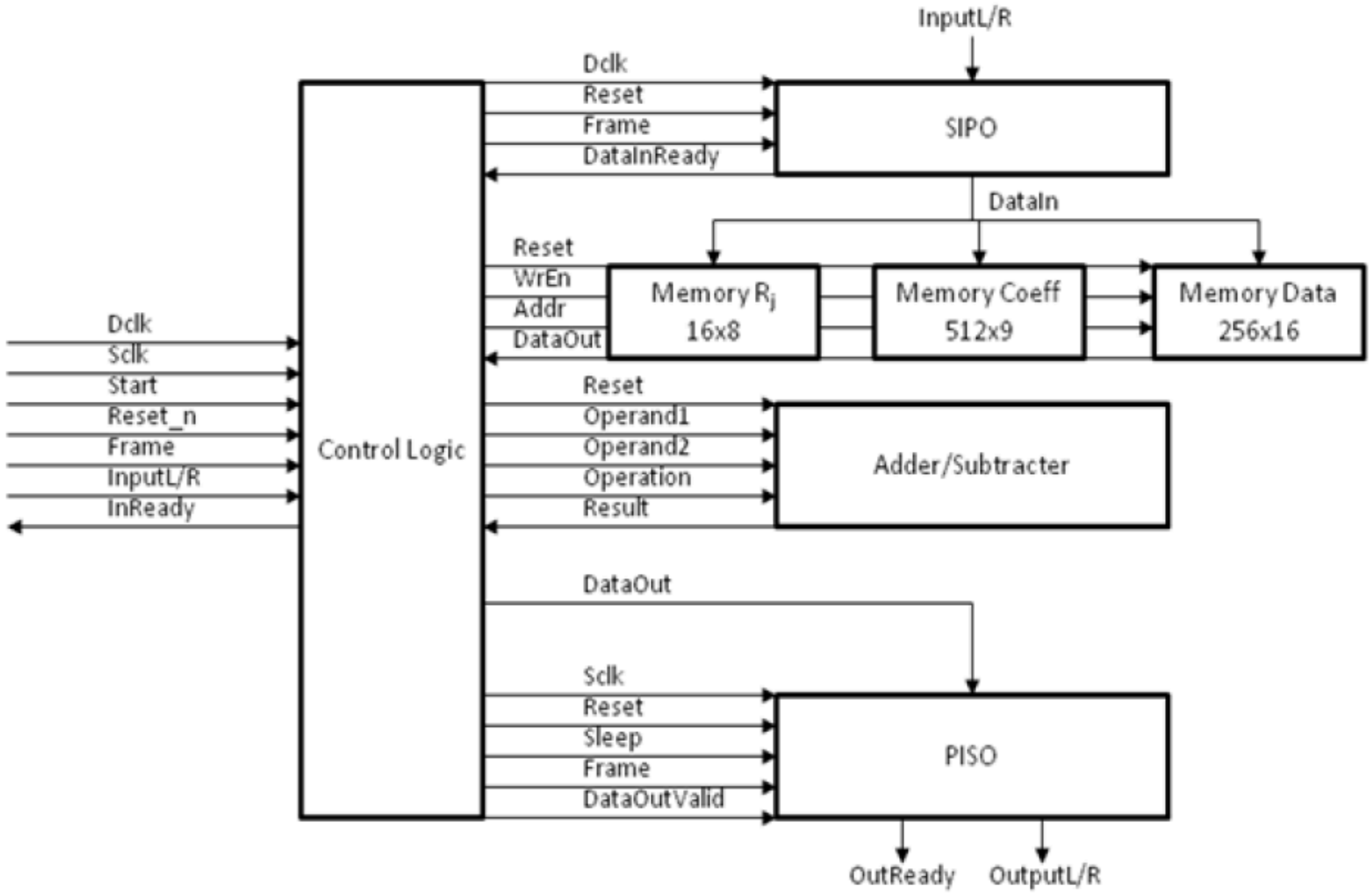
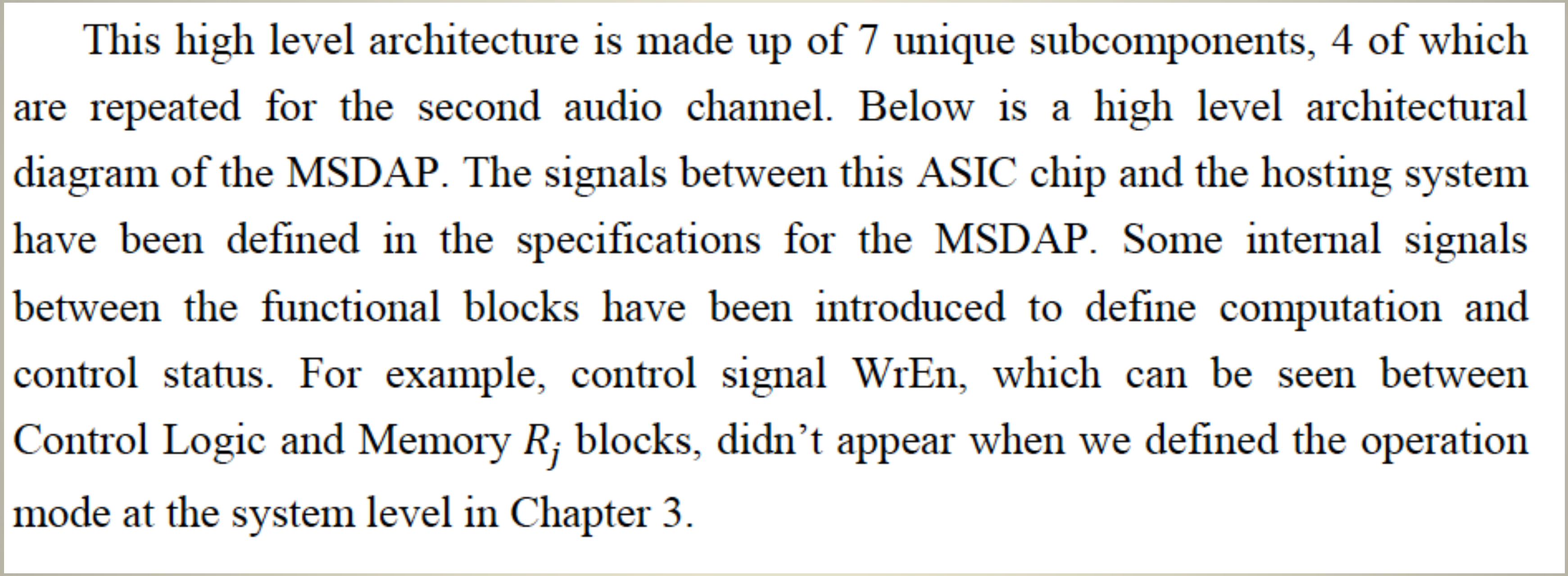


Figure 4-8 A high level abstraction of the MSDAP architecture which facilitates to define functional blocks

- Definition of the function blocks in the architecture

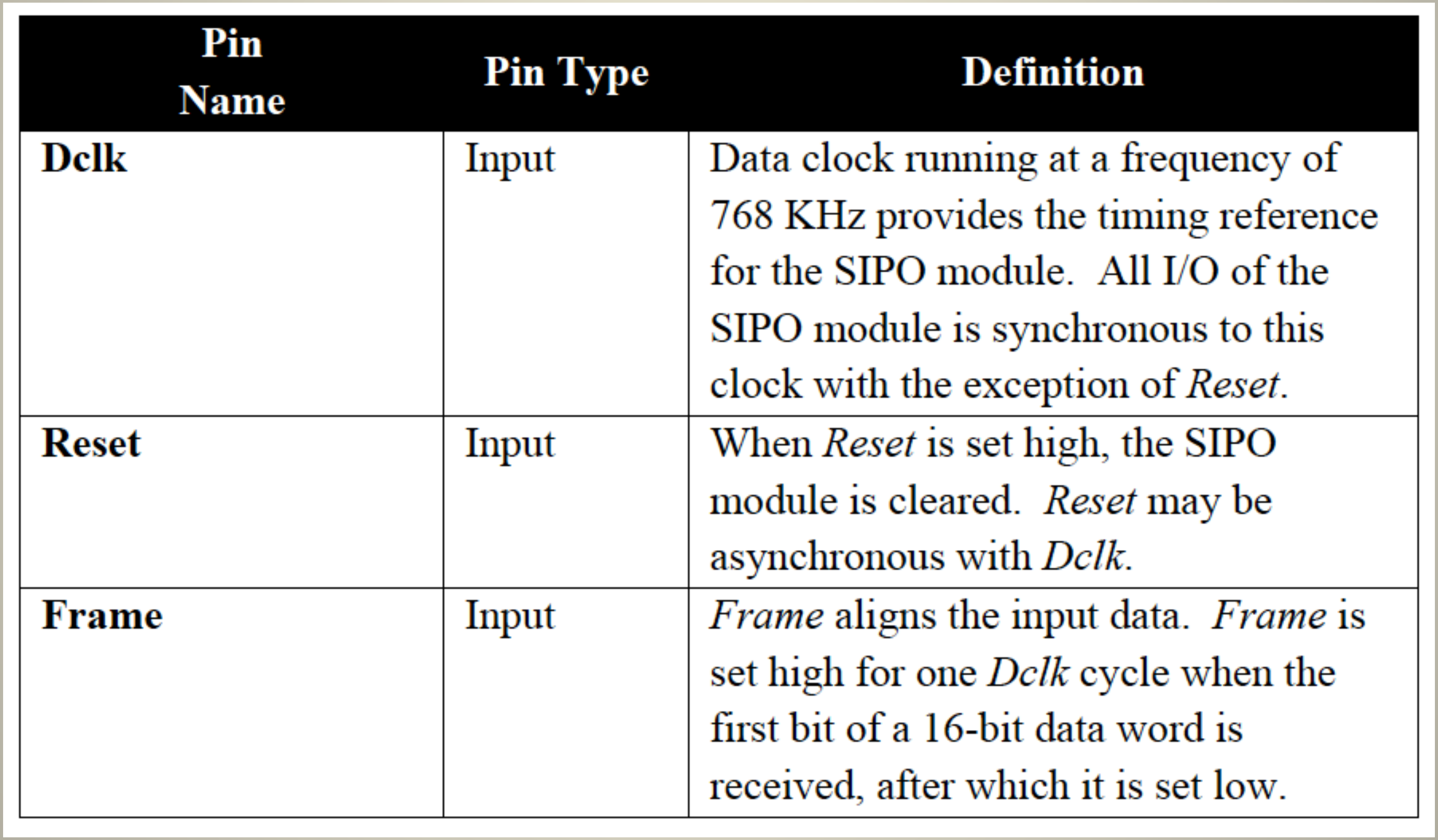


This high level architecture is made up of 7 unique subcomponents, 4 of which are repeated for the second audio channel. Below is a high level architectural diagram of the MSDAP. The signals between this ASIC chip and the hosting system have been defined in the specifications for the MSDAP. Some internal signals between the functional blocks have been introduced to define computation and control status. For example, control signal WrEn, which can be seen between Control Logic and Memory R_j blocks, didn't appear when we defined the operation mode at the system level in Chapter 3.

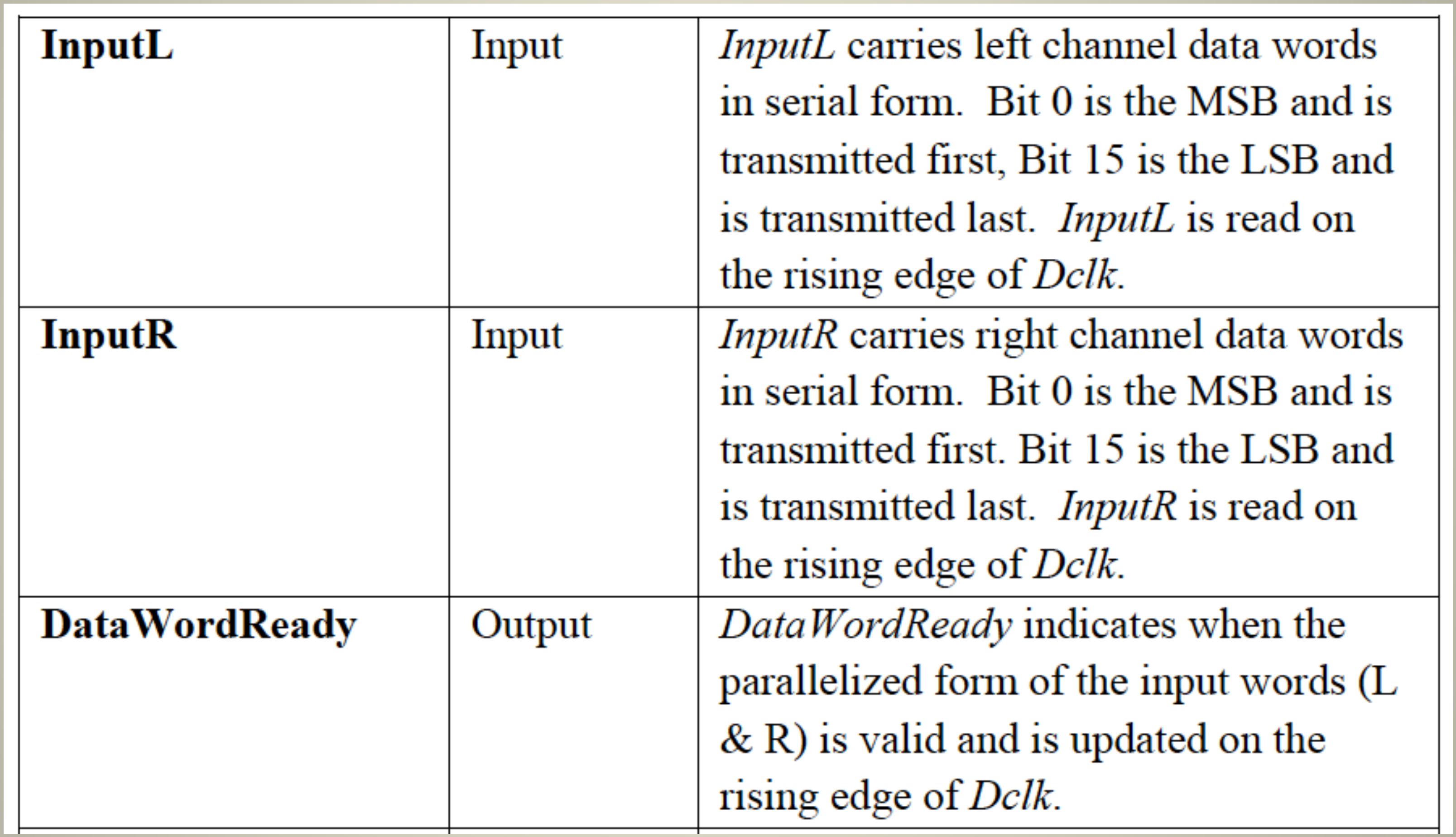
- Control logic definition
  - The main function of the Control Logic of the MSDAP is described in Chapter 3 when discussing the system mode and settings.
    - A sample VHDL code has been developed which handles the state transitions.
  - In addition, this Control Logic handles all data transfers and memory reads and writes. It also handles trivial operations such as sign extension and shift operations.
  - For this reason, other components in the MSDAP architecture are designed as asynchronous elements which are able to start their function within 1 Sclk cycle.
  - The Control Logic also completes the task of monitoring and updating top level control signals as specified by the MSDAP specification (the operation mode in Figure 3-7, and signal format and waveform at Section 3.5.2).

- SIPO definition
  - The SIPO module is the front-end data interface of the MSDAP that takes 1-bit serial data synchronous to a Data Clock and converts it to 16-bit parallel data.
  - New input words are signaled by an increase in the *Frame* signal on the first bit of the serial frame and output words are signaled ready by an increase in the *DataWordReady* signal for the clock cycle, so that the word is guaranteed valid.
  - The SIPO module is cleared by an active high *Reset* signal. Following are its I/Os and data and control signal definitions.

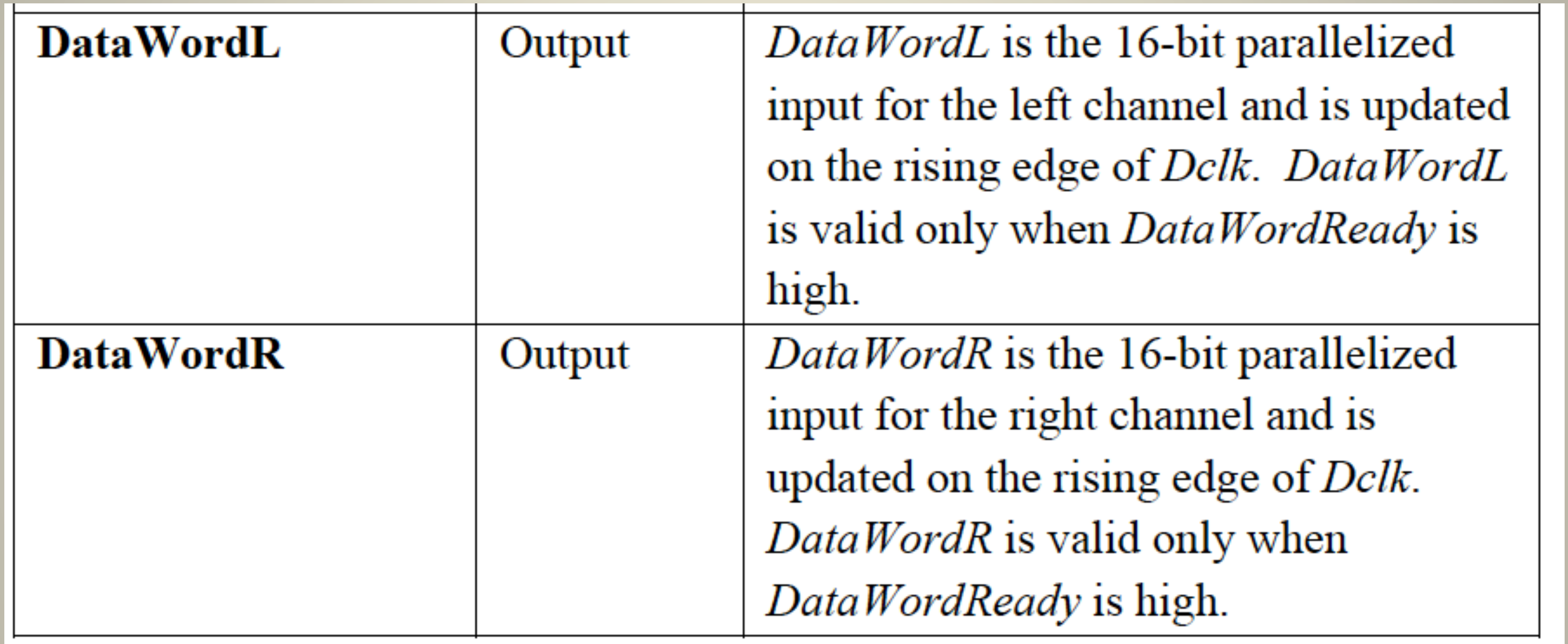
- SIPO definition (details)



Pin Name	Pin Type	Definition
Dclk	Input	Data clock running at a frequency of 768 KHz provides the timing reference for the SIPO module. All I/O of the SIPO module is synchronous to this clock with the exception of *Reset*.
Reset	Input	When *Reset* is set high, the SIPO module is cleared. *Reset* may be asynchronous with *Dclk*.
Frame	Input	*Frame* aligns the input data. *Frame* is set high for one *Dclk* cycle when the first bit of a 16-bit data word is received, after which it is set low.



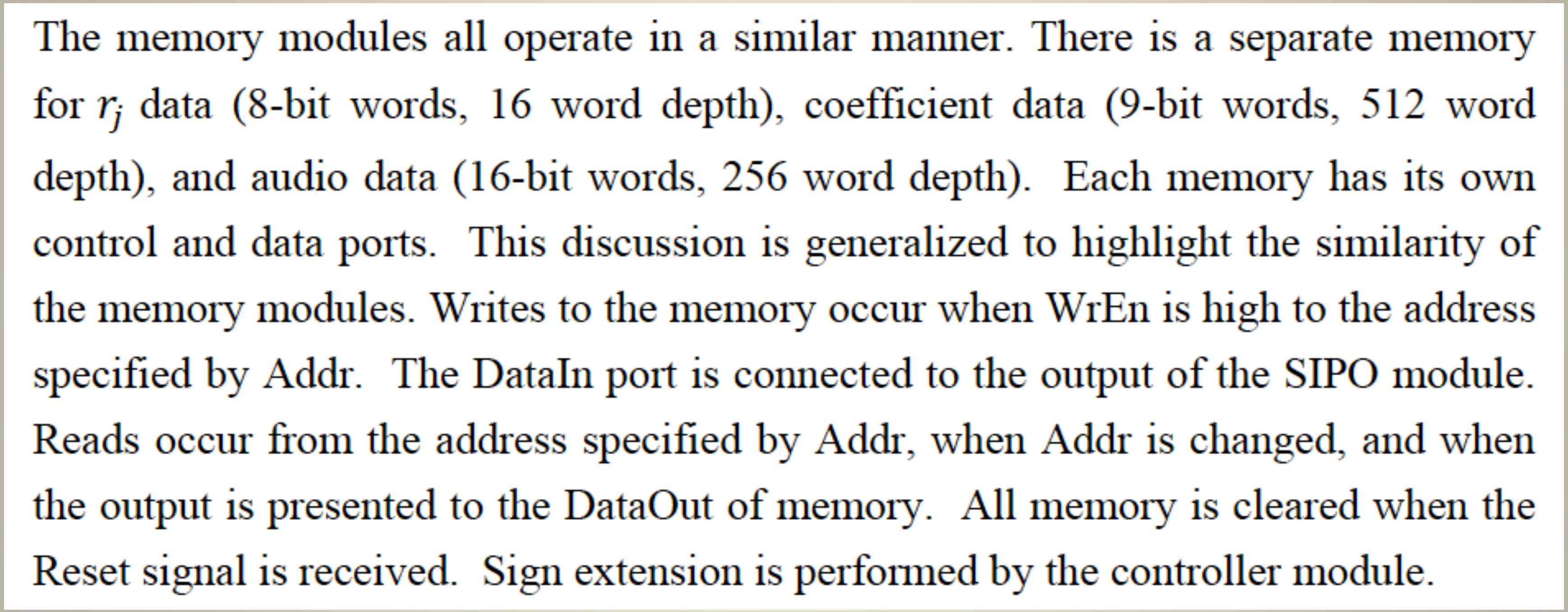
InputL	Input	*InputL* carries left channel data words in serial form. Bit 0 is the MSB and is transmitted first, Bit 15 is the LSB and is transmitted last. *InputL* is read on the rising edge of *Dclk*.
InputR	Input	*InputR* carries right channel data words in serial form. Bit 0 is the MSB and is transmitted first. Bit 15 is the LSB and is transmitted last. *InputR* is read on the rising edge of *Dclk*.
DataWordReady	Output	*DataWordReady* indicates when the parallelized form of the input words (L & R) is valid and is updated on the rising edge of *Dclk*.



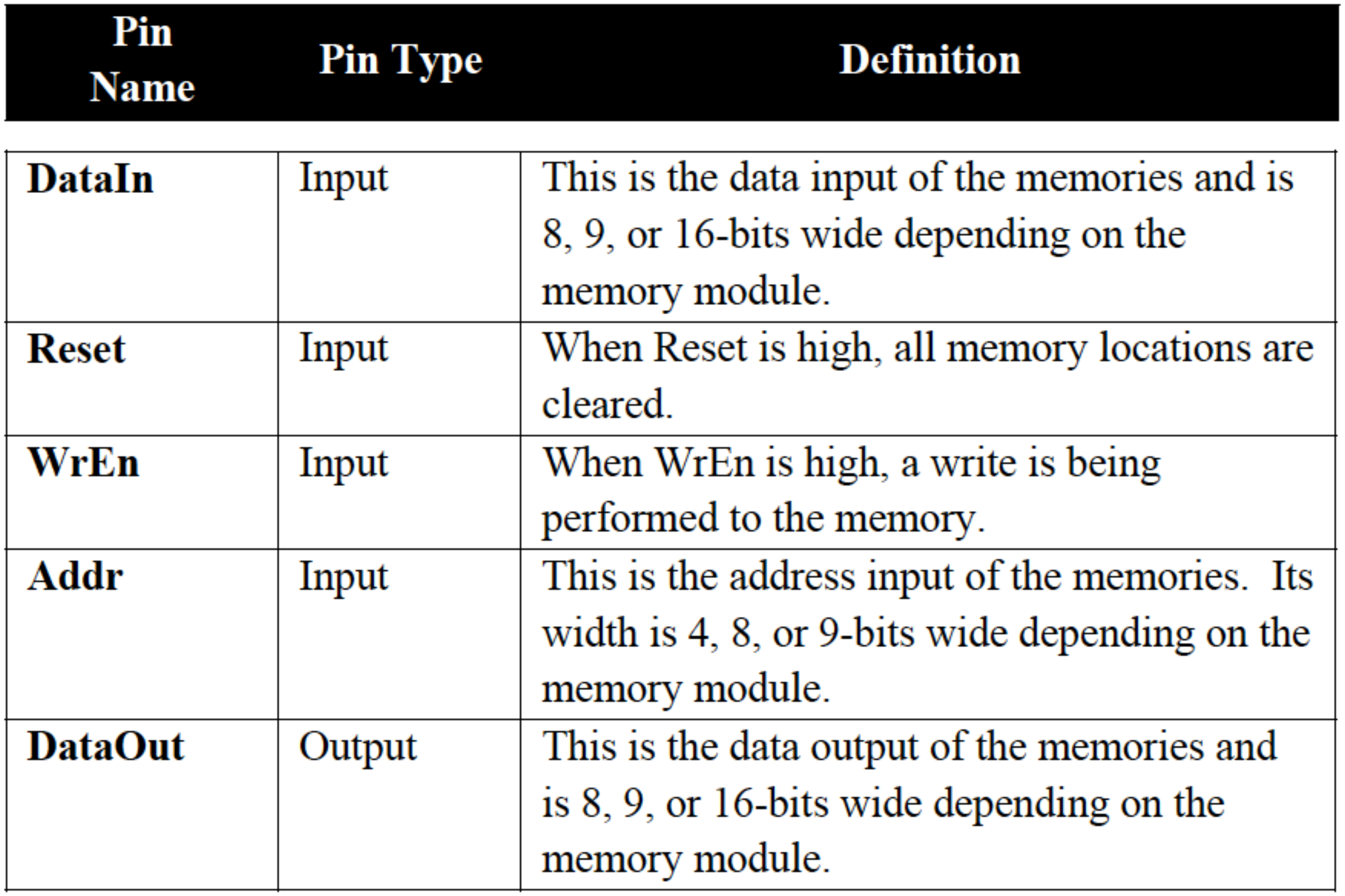
DataWordL	Output	*DataWordL* is the 16-bit parallelized input for the left channel and is updated on the rising edge of *Dclk*. *DataWordL* is valid only when *DataWordReady* is high.
DataWordR	Output	*DataWordR* is the 16-bit parallelized input for the right channel and is updated on the rising edge of *Dclk*. *DataWordR* is valid only when *DataWordReady* is high.

Table 4-1 Signal and pin definition of SIPO

- Memory definition



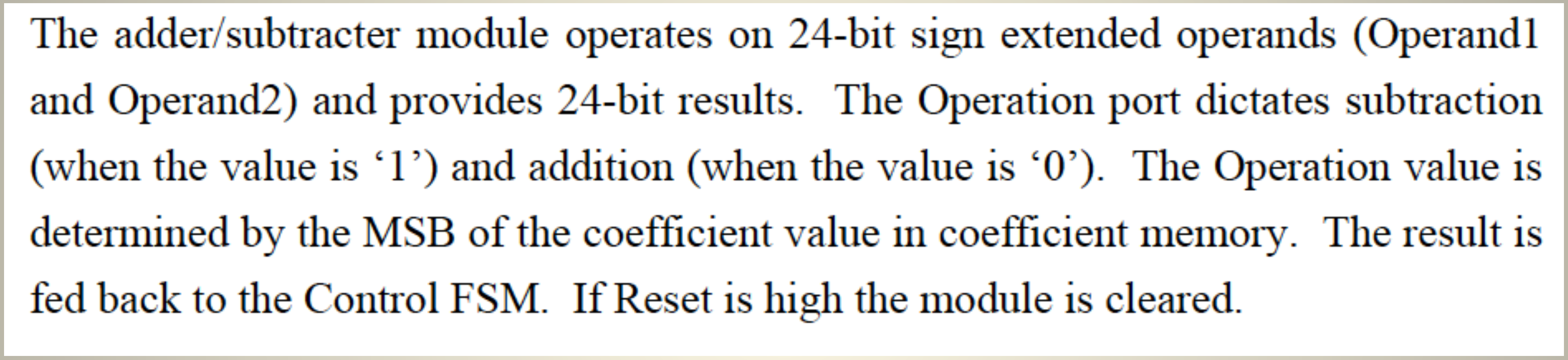
The memory modules all operate in a similar manner. There is a separate memory for r_j data (8-bit words, 16 word depth), coefficient data (9-bit words, 512 word depth), and audio data (16-bit words, 256 word depth). Each memory has its own control and data ports. This discussion is generalized to highlight the similarity of the memory modules. Writes to the memory occur when WrEn is high to the address specified by Addr. The DataIn port is connected to the output of the SIPO module. Reads occur from the address specified by Addr, when Addr is changed, and when the output is presented to the DataOut of memory. All memory is cleared when the Reset signal is received. Sign extension is performed by the controller module.



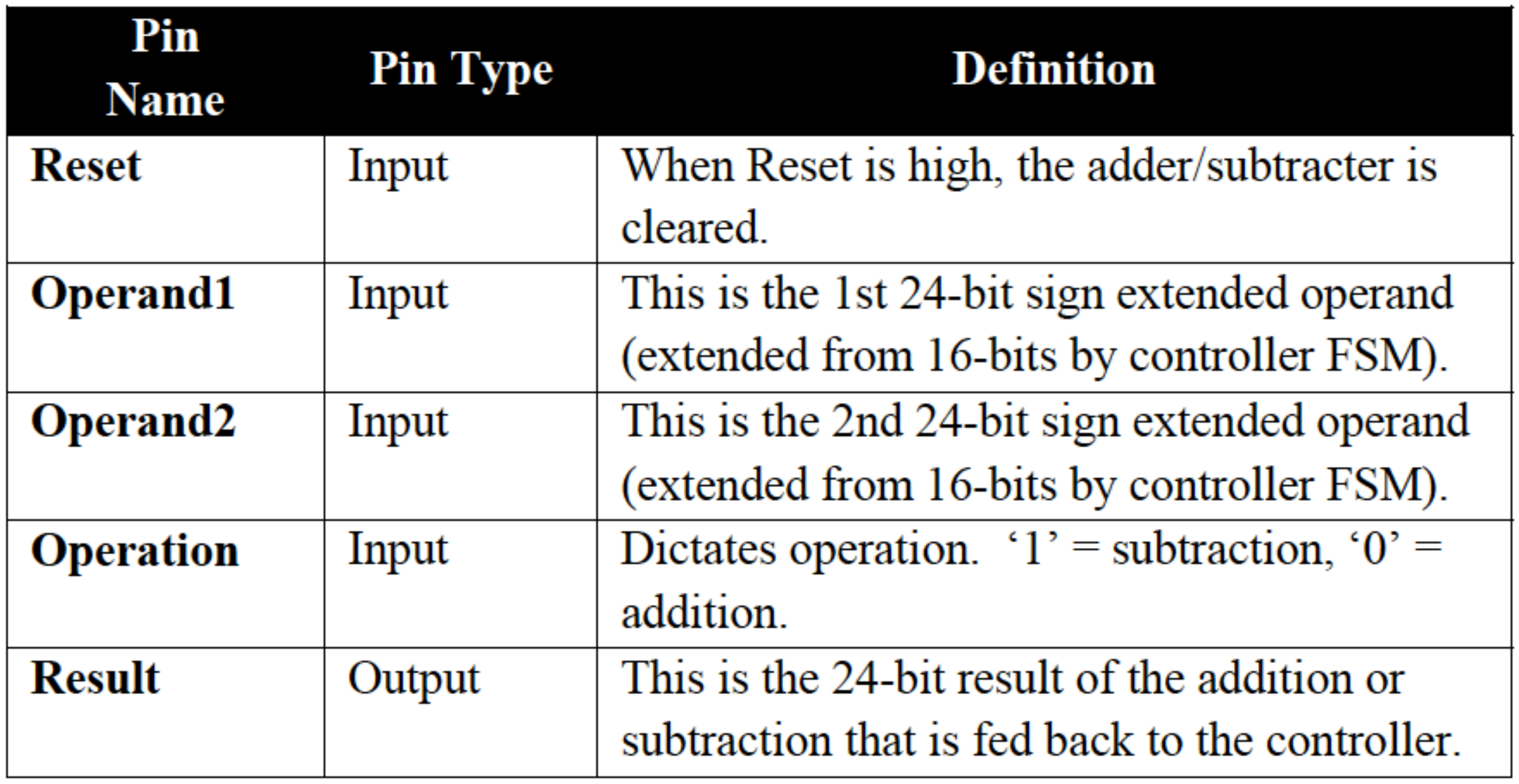
Pin Name	Pin Type	Definition
DataIn	Input	This is the data input of the memories and is 8, 9, or 16-bits wide depending on the memory module.
Reset	Input	When Reset is high, all memory locations are cleared.
WrEn	Input	When WrEn is high, a write is being performed to the memory.
Addr	Input	This is the address input of the memories. Its width is 4, 8, or 9-bits wide depending on the memory module.
DataOut	Output	This is the data output of the memories and is 8, 9, or 16-bits wide depending on the memory module.

Table 4-2 Signal and pin definition of memory block

- Adder/Subtracter definition



The adder/subtracter module operates on 24-bit sign extended operands (Operand1 and Operand2) and provides 24-bit results. The Operation port dictates subtraction (when the value is ‘1’) and addition (when the value is ‘0’). The Operation value is determined by the MSB of the coefficient value in coefficient memory. The result is fed back to the Control FSM. If Reset is high the module is cleared.

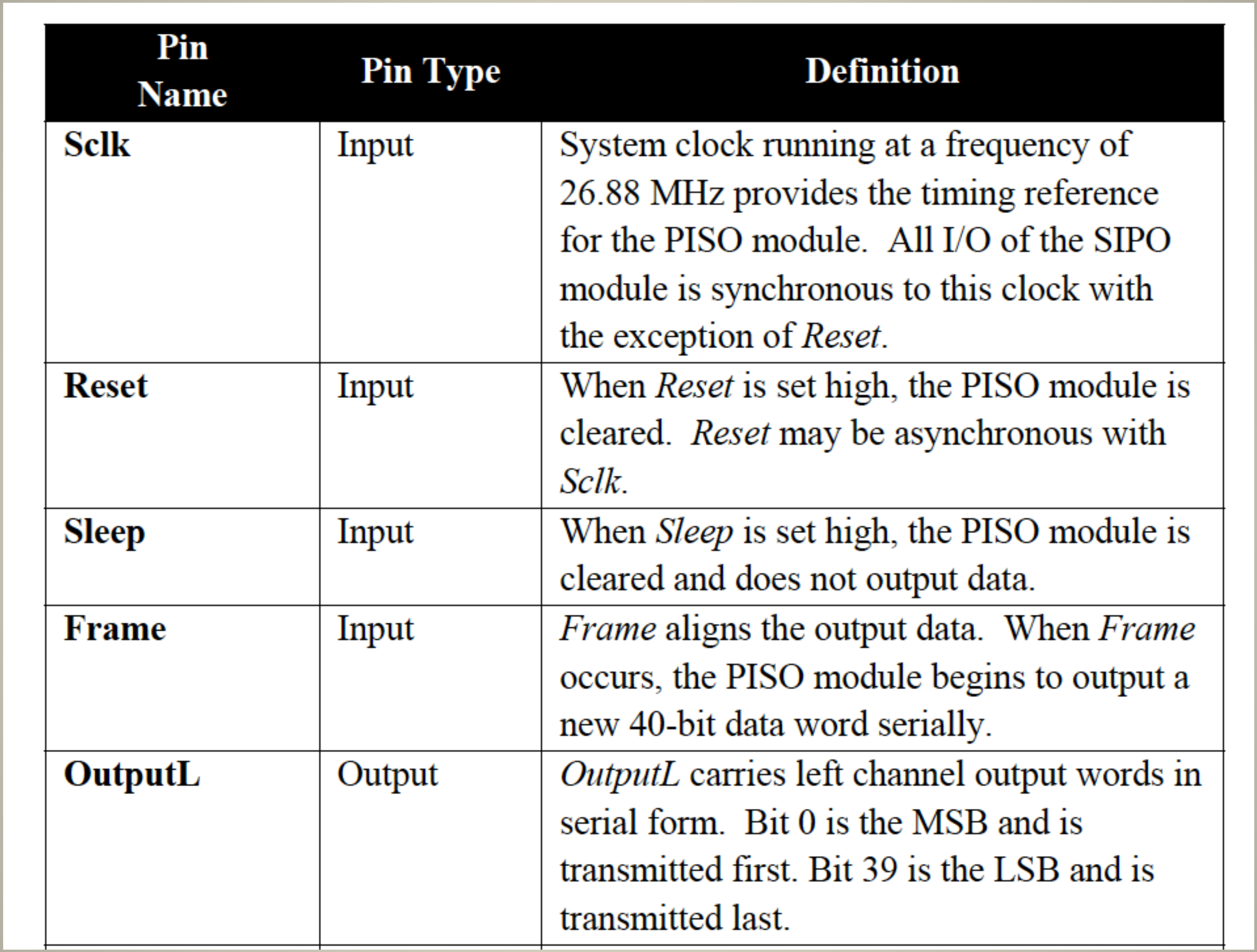


Pin Name	Pin Type	Definition
Reset	Input	When Reset is high, the adder/subtracter is cleared.
Operand1	Input	This is the 1st 24-bit sign extended operand (extended from 16-bits by controller FSM).
Operand2	Input	This is the 2nd 24-bit sign extended operand (extended from 16-bits by controller FSM).
Operation	Input	Dictates operation. ‘1’ = subtraction, ‘0’ = addition.
Result	Output	This is the 24-bit result of the addition or subtraction that is fed back to the controller.

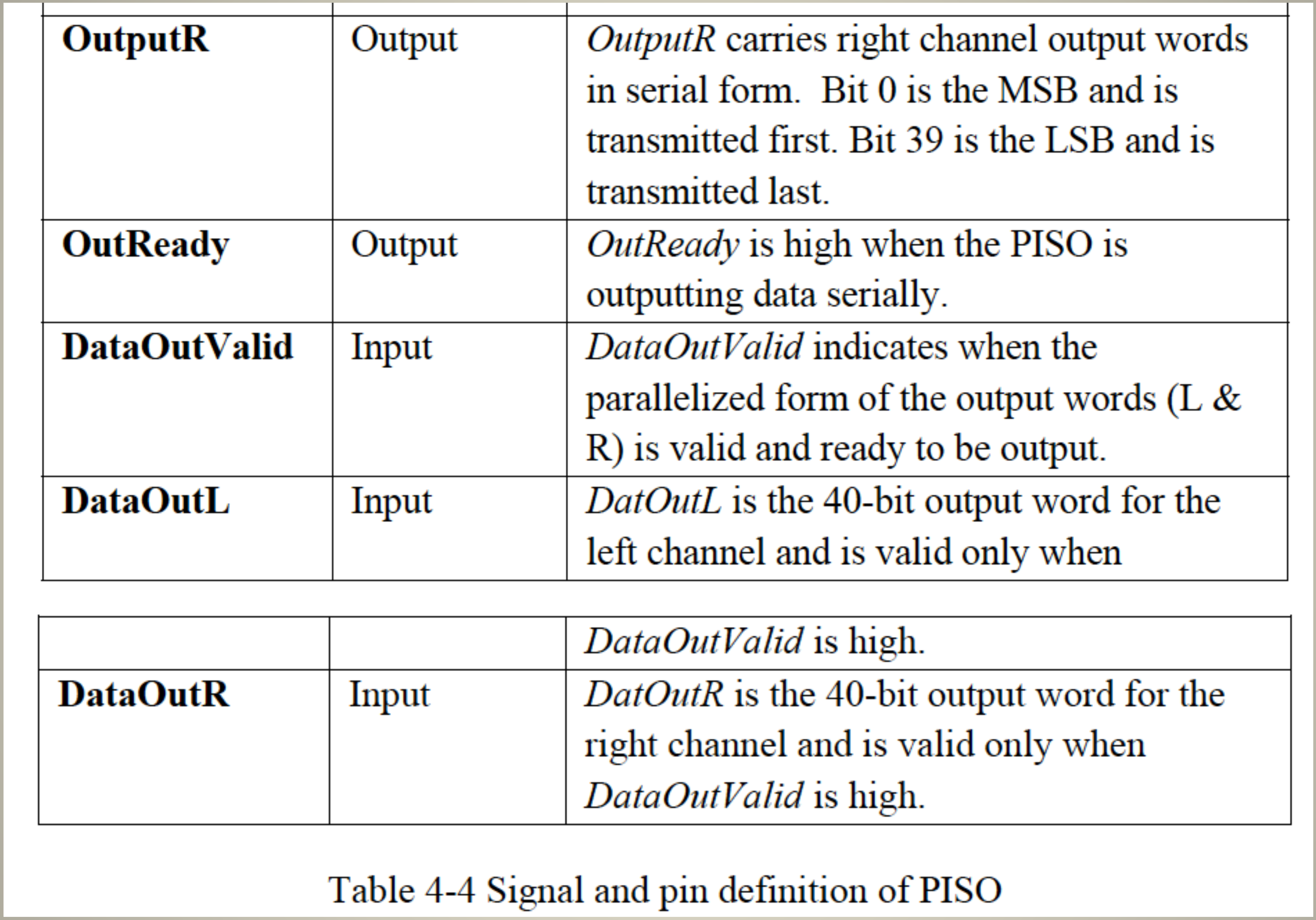
Table 4-3 Signal and pin definition of ALU

- Shift operation definition
  - The shifting operation of the MSDAP is performed by the Controller FSM by reassigning the bit positions of the output of the adder when accumulating the final result and thus is not specified as a separate module.
  - The DataOut signal is the final shifted result that is outputted by the PISO module.

- PISO definition
  - The PISO module is the data interface of the MSDAP that processes 40-bit parallel data outputs serially via a 1-bit port synchronous to Sclk upon the arrival of the next Frame signal.
  - Output words are signaled ready by an increase of the DataOutValid signal.
  - The OutReady signal is set high for the 40 bits that the PISO is serially outputting data.
  - The SIPO module is cleared by an active high Reset signal and does not output when it receives a high Sleep signal.



Pin Name	Pin Type	Definition
Sclk	Input	System clock running at a frequency of 26.88 MHz provides the timing reference for the PISO module. All I/O of the SIPO module is synchronous to this clock with the exception of *Reset*.
Reset	Input	When *Reset* is set high, the PISO module is cleared. *Reset* may be asynchronous with *Sclk*.
Sleep	Input	When *Sleep* is set high, the PISO module is cleared and does not output data.
Frame	Input	*Frame* aligns the output data. When *Frame* occurs, the PISO module begins to output a new 40-bit data word serially.
OutputL	Output	*OutputL* carries left channel output words in serial form. Bit 0 is the MSB and is transmitted first. Bit 39 is the LSB and is transmitted last.



OutputR	Output	*OutputR* carries right channel output words in serial form. Bit 0 is the MSB and is transmitted first. Bit 39 is the LSB and is transmitted last.
OutReady	Output	*OutReady* is high when the PISO is outputting data serially.
DataOutValid	Input	*DataOutValid* indicates when the parallelized form of the output words (L & R) is valid and ready to be output.
DataOutL	Input	*DatOutL* is the 40-bit output word for the left channel and is valid only when *DataOutValid* is high.
DataOutR	Input	*DatOutR* is the 40-bit output word for the right channel and is valid only when *DataOutValid* is high.

Table 4-4 Signal and pin definition of PISO

- From the above example, we see how functional blocks are defined.
- With a complete definition, blocks can be assigned to different design groups and be verified individually before integrating them into the system.
- The definition of a function block also includes control signals since a block is controlled by a top level FSM.
  - For instance, a signal Reset is a signal from the Controller FSM that instructs the memory, SIPO and PISO to clear their contents and set them to an initial condition.
  - A functional block also usually feeds the controller information about its status. *DataOutValid* in the PISO block is such an instance.

Time Budget and Scheduling

- In a parallel manner, during the architecture development we not only define the functional blocks and corresponding relations, but also assign a time budget to each functional block.
- Specifically,
  - Each functional block should be given adequate cycles to complete its assigned job;
  - The input to each functional block must be ready when it is needed.

- Time budget is usually obtained from a process called resource allocation and scheduling.
- Resource allocation assigns computation tasks to functional blocks, and scheduling arranges these tasks in such a way that they can be executed without confliction.
- To find the time window for a task to be executed we can use the schedule techniques like as soon as possible and as late as possible.
  - However, combining scheduling with optimal resource allocation is a difficult task. Most of the time, it is still a decision made based on experience.

- The time budget depends on the proposed architecture and its feasibility must be analyzed carefully in the architecture design stage.
- A structure is requisite to discussing this topic.
- In the next section we shall use the architecture of our MSDAP project to show how to allocate the time budget.

A Sample Architecture of the MSDAP project

- Figure 4-8 is a top level general datapath architecture, which we have used to assist the discussion of how to define functional blocks.
- However, architecture should be developed such that it can eventually be coded at an RTL level so that a cycle-by-cycle simulation can be carried out to verify the correctness of the digital system without involving detailed logic and circuit designs.
- The degree of detail depends on the designer, but it must be to a level where we can write VHDL code and verify the hardware design.

- In doing so, we need to do work similar to what we did in developing the specification of the MSDAP. That is, we need to define the signals and functions among the functional blocks.
  - For example, in referring to the general architecture in Figure 4-8, we do not know how to write the VHDL code since we haven't defined the interfaces between the functional blocks.
  - Therefore, one can't tell in what way ALU works with memory and output interface blocks.
  - In short, too many details are missing there.
  - In the following we look at how to develop such an architecture.

An Architecture Sample

- Based on the spec, a proposed architecture for the MSDAP is shown in Figure 4-9, which allows us to describe it in VHDL.
  – Only one channel is shown here for clarity. Notice that not all the I/O pins for each functional block have been included.
  – Also, the main controller is shared between the two identical datapath blocks, left and right.
  – (Appendix D presents a complete version of this architecture with all detailed notations and definitions.)

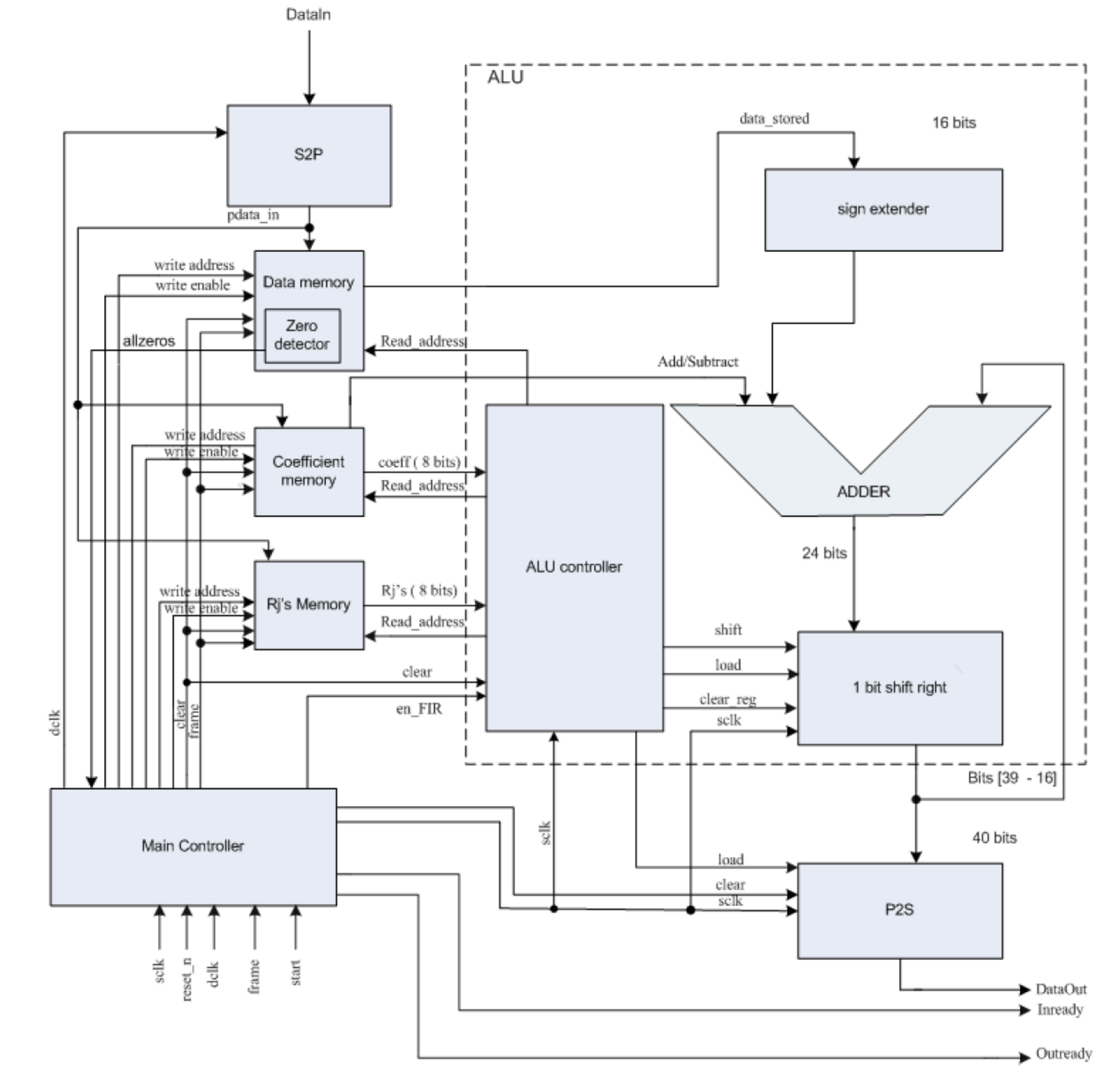
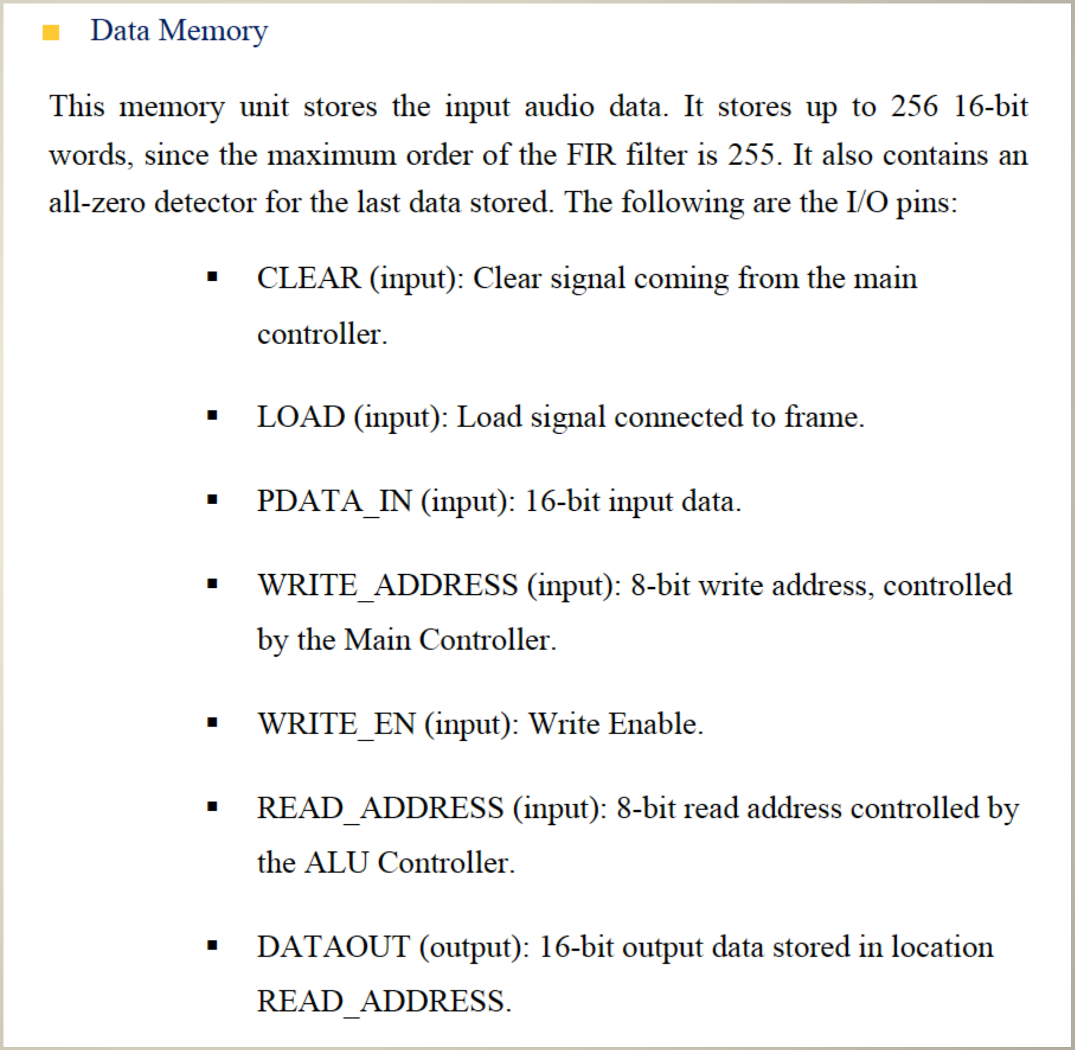


Figure 4-9 A sample architecture

- Definition of each functional block in this architecture are formally presented in the text book.

Here is the definition of Data Memory →



■ Data Memory

This memory unit stores the input audio data. It stores up to 256 16-bit words, since the maximum order of the FIR filter is 255. It also contains an all-zero detector for the last data stored. The following are the I/O pins:

- CLEAR (input): Clear signal coming from the main controller.
- LOAD (input): Load signal connected to frame.
- PDATA_IN (input): 16-bit input data.
- WRITE_ADDRESS (input): 8-bit write address, controlled by the Main Controller.
- WRITE_EN (input): Write Enable.
- READ_ADDRESS (input): 8-bit read address controlled by the ALU Controller.
- DATAOUT (output): 16-bit output data stored in location READ_ADDRESS.

Time Budget Justification of the Proposed Architecture

For the proposed architecture, we can analyze the time budget for each block. Readers may need to revisit Chapter 3 to connect this discussion with the specs of the MSDAP, especially the top level system settings. In the following we focus on the budget for the ALU unit. Considering that the frequency of the data clock is 768kHz and each frame has 16 DCLKs, we can calculate that each frame lasts $T_{FRAME} = \frac{16}{768kHz} = 20.83\mu s$. The system clock frequency is 26.88 MHz, and thus has a period of $T_{SCLK} = \frac{1}{26.88MHz} = 37.2ns$. Consequently, there are $\frac{T_{FRAME}}{T_{SCLK}} = 16\left(\frac{26.88MHz}{768kHz}\right) = 560$ SCLK cycles in one frame to perform all the computations. As mentioned previously, the rate of the system clock depends on the particular implementation and can be different for different chips. The system clock rate based on the above analysis is actually realized in the design shown in Appendix D. The time budget for each operation is shown in Figure 4-10 and the justification is detailed in the following.

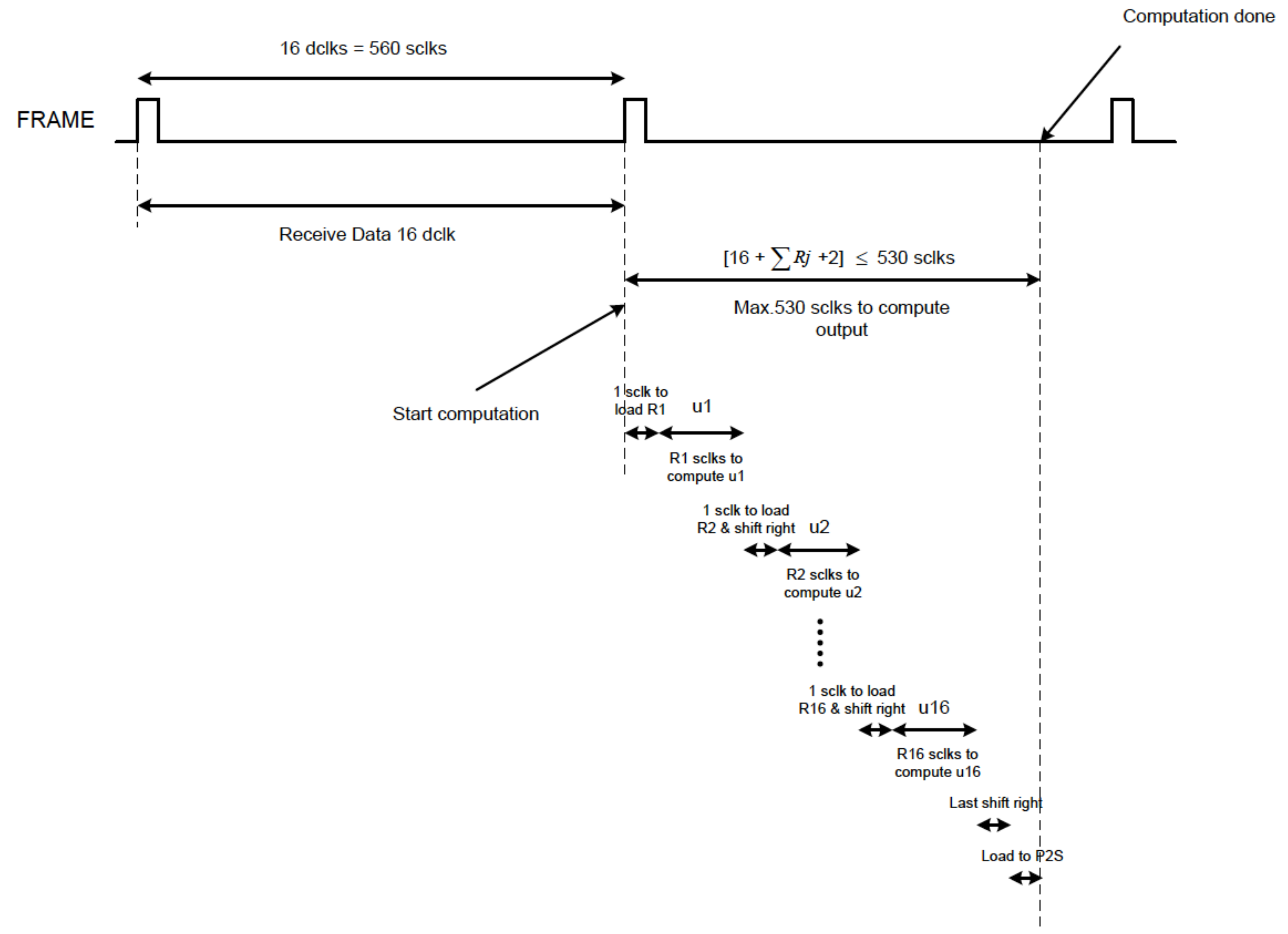
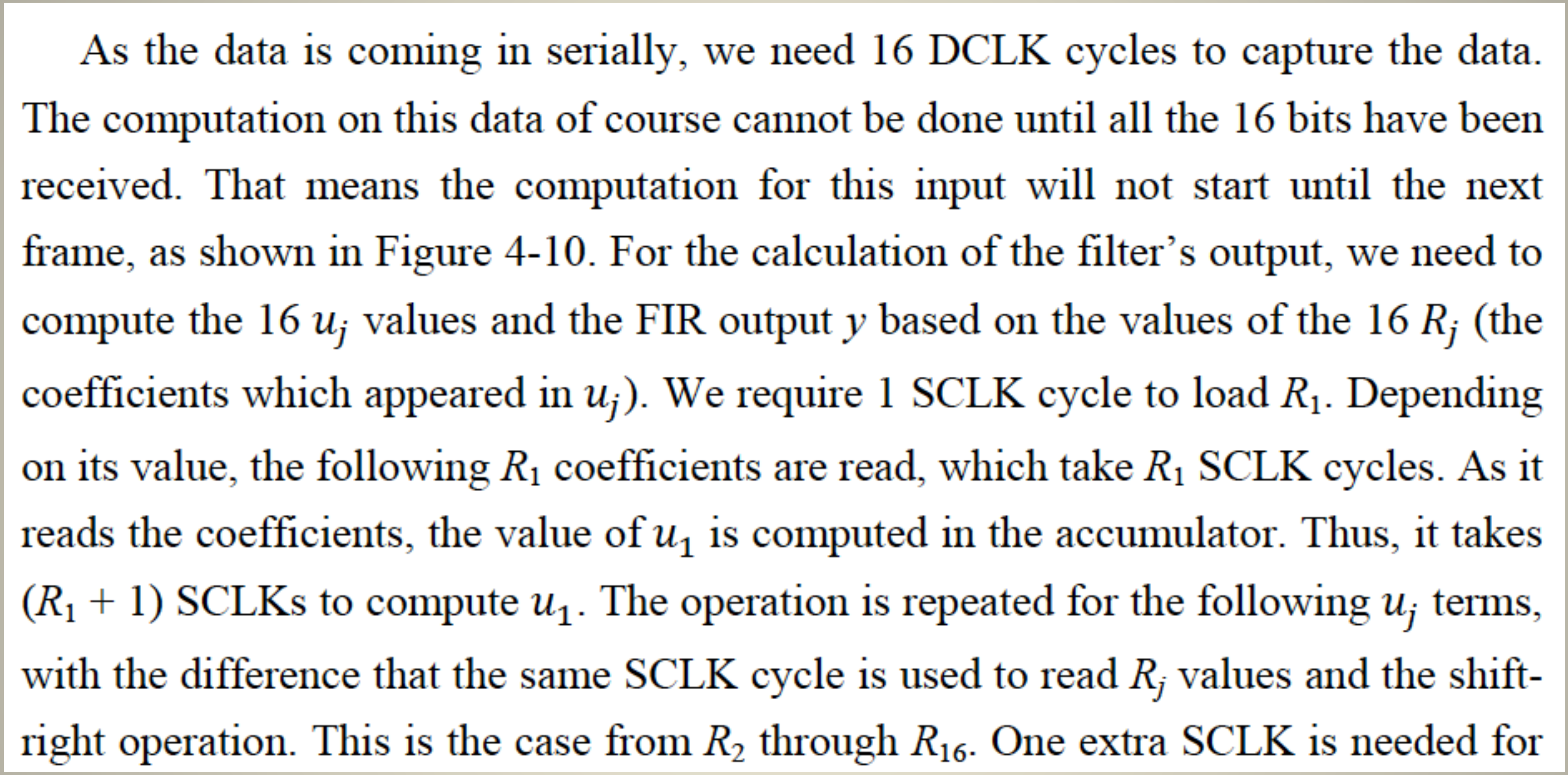
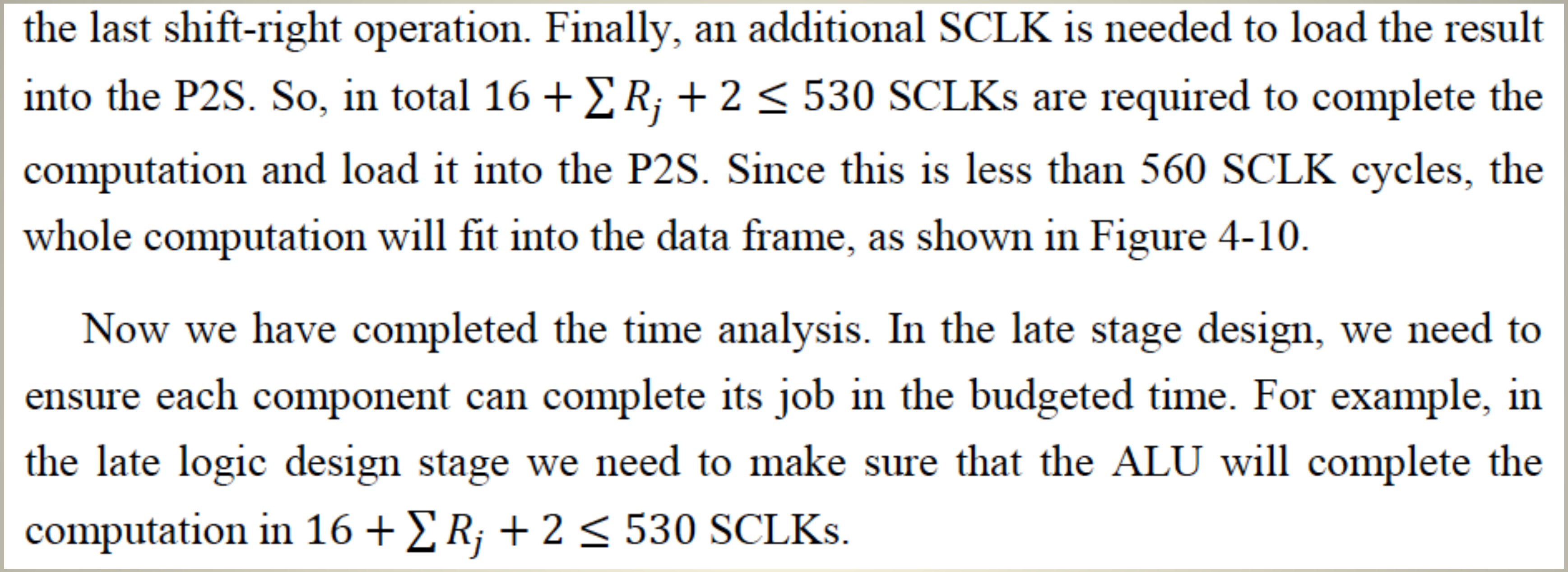


Figure 4-10 Time budget analysis for the ALU unit

Time budget feasibility analysis



As the data is coming in serially, we need 16 DCLK cycles to capture the data. The computation on this data of course cannot be done until all the 16 bits have been received. That means the computation for this input will not start until the next frame, as shown in Figure 4-10. For the calculation of the filter's output, we need to compute the 16 u_j values and the FIR output y based on the values of the 16 R_j (the coefficients which appeared in u_j). We require 1 SCLK cycle to load R_1. Depending on its value, the following R_1 coefficients are read, which take R_1 SCLK cycles. As it reads the coefficients, the value of u_1 is computed in the accumulator. Thus, it takes $(R_1 + 1)$ SCLKs to compute u_1. The operation is repeated for the following u_j terms, with the difference that the same SCLK cycle is used to read R_j values and the shift-right operation. This is the case from R_2 through R_{16}. One extra SCLK is needed for



the last shift-right operation. Finally, an additional SCLK is needed to load the result into the P2S. So, in total $16 + \sum R_j + 2 \leq 530$ SCLKs are required to complete the computation and load it into the P2S. Since this is less than 560 SCLK cycles, the whole computation will fit into the data frame, as shown in Figure 4-10.

Now we have completed the time analysis. In the late stage design, we need to ensure each component can complete its job in the budgeted time. For example, in the late logic design stage we need to make sure that the ALU will complete the computation in $16 + \sum R_j + 2 \leq 530$ SCLKs.

Summary

- In this chapter we discussed how to develop an architecture based on the given application.
- The most popular architecture is a datapath which consists of control logic, input/output, and an ALU unit.
- These blocks can be further partitioned into smaller functional blocks to facilitate a clear and accurate definition of timing and operation.
  - Such level of detail is necessary for an accurate analysis of timing budget.
- Final architecture should be described at the RTL level for synthesis and verification.

- A sample architecture for the MSDAP is given to illustrate the concepts mentioned in the discussion.
  - VHDL codes of the main blocks of the sample architecture are given to show how to accurately describe each functional block without any human language ambiguity, since such code can be simulated for verification.
- The whole design of the MSDAP, including its architecture, is given in Appendix D, from which the interested reader can have a complete picture of the design process.

- This chapter also discussed how to explore the parallelism by using multiple processors.
  - The key issue is to properly align the data and operations to obtain the maximum speed-up.
- The optimal design is if all processors are doing the necessary computations and not being idle.
- Optimal scheduling and resource allocation is the key for a good architecture.
  - This topic has been studied extensively in high-level synthesis. Readers who are interested in this topic can check the research literatures on this subject and find the state-of-the-art CAD tools to assist architecture design.

Homework

1. Explain how to define function blocks in architecture design. Try to formalize the method and give an example.
2. Give an example to show how data and computation dependency prevents us from achieving linear speed-up.
3. Summarize the method of architecture development in ASIC design.
4. Synthesizing a behavior model to get an RTL level netlist needs an "underlined architecture", for any non-trivial application. Discuss this issue using an example.
5. Discuss why architecture design automation is extremely difficult. Suggest the method to solve this problem.
6. Develop an architecture of the MSDAP.
7. Explain the issue of "resource allocation" and "scheduling" in the architecture design. Use the architecture if MSDAP as an example.