2006

A configurable decoder for pin-limited applications

Matthew Collin Jordan
Louisiana State University and Agricultural and Mechanical College

A CONFIGURABLE DECODER FOR PIN–LIMITED APPLICATIONS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Matthew Collin Jordan
B.S., Michigan Technological University, 2004
December 2006

Acknowledgments

First, I would like to thank Dr. Ramachandran Vaidyanathan for his patience, guidance, and advice throughout the course of the past two years. His assistance has not only developed the research in this thesis, but also the abilities I have as a student and engineer. I would also like to thank the contributions of the members of the committee, Dr. Jerry Trahan and Dr. Suresh Rai, which have greatly helped in the development of the thesis. Finally, I would like to thank my parents and my siblings. Without their support, none of this research would have been possible. This thesis is dedicated to them.

Table of Contents

List of Tables

List of Figures

Abstract

Pin limitation is the restriction imposed on an IC chip by the unavailability of a sufficient number of I/O pins. This impacts the design and performance of the chip, as the amount of information that can be passed through the boundary of the chip becomes limited. One area that would benefit from a reduction of the effect of pin limitation is reconfigurable architectures. In this work, we consider reconfigurable devices called Field Programmable Gate Arrays (FPGAs). Due to pin limitation, current FPGAs use a form of 1-hot decoder to select elements (one frame at a time) during partial reconfiguration. This results in a slow and coarse selection of elements for reconfiguration. We propose a module that performs a focused selection of only those elements that require reconfiguration. This reduces reconfiguration overheads and enables the speeds needed for dynamic reconfiguration.

The problem is that of selecting subsets of an n-element set in a fast, focused and inexpensive manner. This thesis proposes such a configurable decoder that bridges the gap between the inexpensive, but inflexible, fixed 1-hot decoder, and the expensive, but flexible, pure LUT-based decoder. Our configurable decoder uses a LUT with a narrow output and a low cost in tandem with a special fixed decoder called a mapping unit that expands the output of the LUT to a desired n-bit output. We demonstrate several implementations of the mapping unit, each with different capabilities and trade-offs. A key result of this work is that for any gate cost $G = O(n \log^k n)$ (where k is a constant), if a pure LUT-based solution produces λ independent subsets, then our method produces $\Omega\Big(\frac{\lambda \log n}{\log \log n}\Big)$ independent subsets for the same cost. Our decoder also produces many more dependent subsets (that depend on the choice of the $\Omega\Big(\frac{\lambda \log n}{\log \log n}\Big)$ independent subsets).

We provide simulation results for the configurable decoder and predict future trends from the simulation data; these confirm the theoretical advantages of the proposed decoder. We illustrate the implementation of important subset classes on our configurable decoder and make key observations on a generalized variant.

Chapter 1
Introduction

Over time, as processor speeds increased faster than the rate at which information could enter and exit a chip, in many cases, it was found that increasing processor speed while ignoring the effects of I/O produced little results [14] — essentially, if information cannot get into or out of the chip at a fast enough rate, then CPU speed diminishes in importance. This implies that modern chips benefit from a high rate of data transfer between the inside and outside of the chip. This data transfer can be improved by increasing the bit rate and/or the number of I/O pins. Since pins cannot be miniaturized to the same extent as transistors (pins must be physically strong enough to withstand contact), the rate at which the number of transistors on a chip has increased far outpaces the rate at which the number of pins on a chip has increased [26]. For example, in Intel microprocessors, the number of transistors has increased by a factor of 20,000 in the last 30 years, whereas the number of pins in these chips increased merely by a factor of 30 [15]. Therefore, the rate at which a chip can generate and process information is much larger than the available conduit to convey this information. The restriction imposed by the unavailability of a sufficient number of pins in a chip is called *pin limitation*.

This thesis proposes a method to alleviate the pin limitation constraint in IC chips. Past research in this area approached the problem by increasing the number of pins on the chip [25]. Others have proposed methods to change the application or chip functionality to reduce pin requirements [12, 21]. In modern ICs, however, there seems to be little room for increasing the number of pins that can be physically placed on a single chip without a substantial change in technology. We seek a way to make better use of the available pins without altering the functionality of the underlying chip.

One area that would benefit from a reduction of the effects of pin limitation is reconfigurable architectures, in particular, Field Programmable Gate Arrays (FPGAs) [5, 9]. An FPGA is an array of programmable logic elements, all of which must be configured to suit the

application at hand. Since FPGA elements are simple logic blocks, information to configure the chip must come from outside the chip; given that a modern FPGA can require over 70 million bits for a single configuration encompassing the entire chip [29], a dearth of pins to input this information can have severe time consequences.

FPGAs have evolved out of being simple electronic breadboards to being a competitor to Application Specific Integrated Circuits (ASICs) and even microprocessors in many applications [13]. A number of applications benefit greatly from a technique called *dynamic reconfiguration*, in which elements of the FPGA chip are reconfigured (to alter their interconnections and functionality) while the application is executing on the FPGA [22]. This form of reconfiguration holds the promise for better resource usage and faster execution of certain algorithms. However, it requires fast reconfiguration. Currently, FPGAs adopt a method called partial reconfiguration [2, 27, 29], where only a portion of the FPGA is reconfigured, while the remainder works on. This involves selecting the portion of the FPGA requiring reconfiguration and inputting the necessary configuration bits. Due to pin limitation, only a very coarse selection is available on FPGAs, resulting in a large number of elements being selected for reconfiguration. Unfortunately, this implies that elements that do not need to be reconfigured must be "configured" anyway to their existing states along with those that actually require reconfiguration. Moreover, this process could take multiple cycles of reconfiguration to set all desired elements to the desired configuration. Consequently, current FPGAs fail to fully exploit the power of dynamic reconfiguration demonstrated on theoretical models [22].

Selection of elements for reconfiguration is performed by *decoders*. Technically, an x to n decoder (where $x \ll n$) converts x input bits to n output bits. If these output bits are viewed as representing elements of an n-element set $\mathcal{Z}_n$, then the decoder simply selects the elements of a subset of $\mathcal{Z}_n$. Current FPGAs employ "fixed decoders" that fix the mapping between input and output bits. In fact, the fixed decoder that is normally employed is the 1-hot decoder [29] that accepts a $\log n$-bit input and generates a 1-element subset of $\mathcal{Z}_n$. That is, a 1-hot decoder can select only a single element at a time. This causes problems if, in an array of n elements, some arbitrary pattern of elements is needed. Here, selecting

an appropriate subset can take up to $O(n)$ rounds. Notwithstanding this inflexibility, 1-hot decoders are simple combinational circuits with a low $O(n \log n)$ gate cost and a low $O(\log n)$ propagation delay.

If flexibility is desired, then a "configurable decoder" is required. Currently, a configurable decoder amounts to a look-up table (LUT); we will call this existing decoder a *pure LUT-based configurable decoder*. A $2^x \times n$ LUT is simply a 2^x–location table of n-bit entries. It can produce 2^x independently chosen n-bit patterns that can be selected by an x-bit address. As such, this decoder is highly flexible as the n-bit patterns chosen for the LUT need no relationship to each other. Unfortunately, it is also costly; the gate cost of such a LUT is $\Theta(n2^x)$. For a gate cost of $\Theta(n \log n)$, this LUT only produces $\Theta(\log n)$ subsets; to produce the same number of subsets as a 1-hot decoder, the pure LUT-based configurable decoder has $\Theta(n^2)$ gate cost. Clearly, this does not scale well.

In this thesis we propose a configurable decoder that seeks to bridge the gap between the high flexibility, high cost LUT and the low flexibility, low cost fixed decoder by utilizing both in the manner shown in Figure 1.1. In the first stage a small LUT provides the flexibility

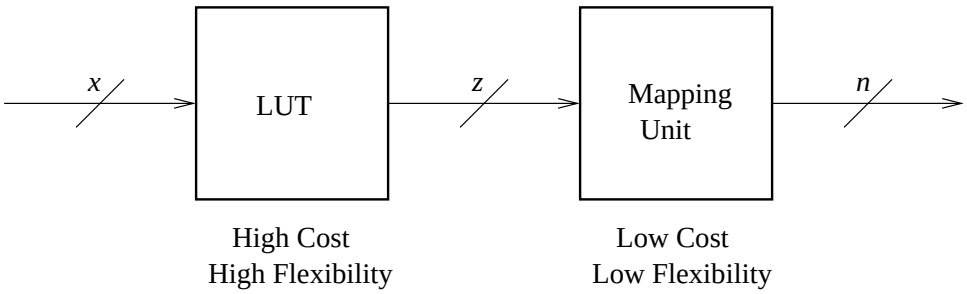


FIGURE 1.1: Proposed configurable decoder overview

with low cost as $z \ll n$. In the second stage an inexpensive decoder expands the output to the form required. This solution is somewhat similar to the well-known accelerated cascading technique for parallel algorithms [16], where a slow but efficient algorithm is used initially to reduce the problem size sufficiently for a subsequent fast and inefficient algorithm to complete the solution.

One key result that we derive is that for any gate cost G, such that $G \leq Cn \log^k n$, where C and k are constant, if a pure LUT-based configurable decoder can produce λ subsets, then

our method can produce $\Omega\left(\frac{\lambda \log n}{\log \log n}\right)$ subsets for the same cost. Our decoder also produces many more dependent subsets (that depend on the choice of the first $\Omega\left(\frac{\lambda \log n}{\log \log n}\right)$ subsets). A particular case of this result occurs when $G = \Theta(n \log n)$. The pure LUT-based solution produces $\Theta(\log n)$ subsets while we produce $\Omega\left(\frac{\log^2 n}{\log \log n}\right)$ subsets. That is, for the cost of a 1-hot decoder, our method exceeds the flexibility of a LUT-based decoder. Note that while the 1-hot decoder or any other fixed decoder can produce just that fixed set of (albeit n) subsets, configurable decoders can produce different subsets (selected arbitrarily by the user) at different times (by off-line reconfiguration). This flexibility is important, particularly in an environment such as an FPGA, whose reconfigurability has made it the platform of choice for many applications, even in preference to ASICs.

This thesis is organized as follows. In Chapter 2, the motivations for this thesis are expanded upon, including pin limitation and dynamically reconfigurable systems. We also explore current solutions to the pin limitation constraint and demonstrate an initial view of the capabilities of decoders.

In Chapter 3, we provide the notation and assumptions used throughout the remainder of this thesis. The basic building blocks upon which our configurable decoder is built are explained and analyzed. We conclude this chapter with an introduction to a configurable decoder and demonstrate that the pure LUT-based configurable decoder is not feasible as a solution to pin limitation.

In Chapter 4, we provide the theoretical basis for the mapping unit (Figure 1.1), a key component of the proposed decoder, which expands the z-bit output of a LUT to the n-bit decoder output. This chapter lays the groundwork for the capabilities of our configurable decoder. In Chapter 5, realizations of the mapping unit are presented. We demonstrate several possible implementations, each with different capabilities and trade-offs.

In Chapter 6, these mapping units are integrated with the preceding LUT to create configurable decoders. The performance of the proposed configurable decoder is analyzed for a given gate cost G and compared against the pure LUT-based approach. We show that in every case, our solution outperforms the LUT in some capacity.

In Chapter 7, we show how several examples of classes of algorithms and communications that have interesting corresponding subsets can be produced by our configurable decoder. Chapter 8 provides simulation results for different implementations of the configurable decoder. Nonlinear regressions performed on this data provide the constants hidden by the asymptotic notation. This provides insight into future cost trends for the proposed modules. In Chapter 9, some observations on a more generalized variant of our configurable decoder are made. Patterns of bits that are difficult for our configurable decoder to produce are more easily produced by this variant. Finally, in Chapter 10, we summarize our results and identify some future avenues of research.

Chapter 2
Pin Limitation

Input and output (I/O) pins allow communication between the interior and exterior of an Integrated Circuit (IC) chip. Typically, this communication manifests as signals from an external source to components within the chip or vice-versa. However, the number of pins available is limited. This "pin limitation" stems primarily from the extent to which pins can be miniaturized without compromising their structural integrity. Pin limitation impacts the design as well as the performance of a chip, as the amount of information that can be passed through the boundary of the chip is limited by the number of I/O pins.

While, according to Moore's Law, the number of transistors in a chip doubles roughly every two years [26], the number of I/O pins available on that chip does not. This is

TABLE 2.1: Intel microprocessor characteristics, 1971–2001 [15]

Processor	Year	Feature Size (μm)	Transistors	Frequency (MHz)	Package
4004	1971	10	2.3k	0.75	16-pin DIP
8008	1972	10	3.5k	0.5–0.8	18-pin DIP
8080	1974	6	6k	2	40-pin DIP
8086	1978	3	29k	5–10	40-pin DIP
80286	1982	1.5	134k	6–12	68-pin PGA
Intel386	1985	1.5–1.0	275k	16–25	100-pin PGA
Intel486	1989	1–0.6	1.2M	25–100	168-pin PGA
Pentium	1993	0.8–0.35	3.2 - 4.5M	60–300	296-pin PGA
Pentium Pro	1995	0.6–0.35	5.5M	166–200	387-pin MCM PGA
Pentium II	1997	0.35–0.25	7.5M	233–450	242-pin SECC
Pentium III	1999	0.25–0.18	9.5 - 28M	450–1000	330-pin SECC2
Pentium 4	2001	0.18–0.13	42 - 55M	1400–3200	478-pin PGA

demonstrated in Table 2.1, where, over the past three decades, the number of transistors available on an Intel microprocessor has increased by a factor greater than 20,000, while the number of pins on the microprocessor package has only increased by a factor of 30 (see also

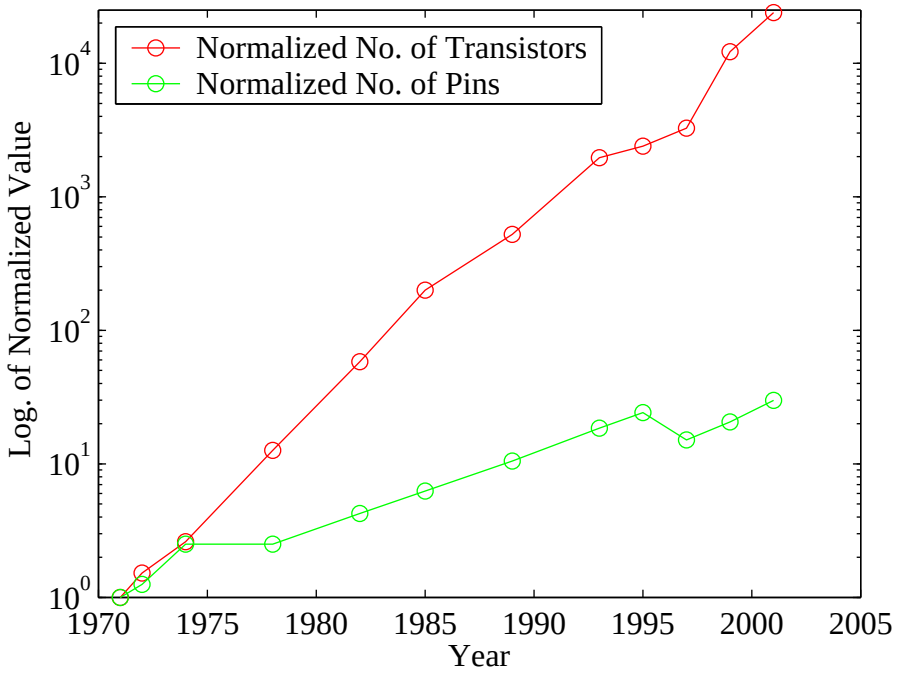


FIGURE 2.1: Normalized transistor and pin counts for 1971 – 2001

Figure 2.1). Thus, the potential amount of information needed inside a chip has increased significantly faster than the means to allow that information in and out of the chip. Under current technology, this trend does not appear likely to change.

While techniques to alleviate pin limitation are applicable in the design of any IC chip, certain applications benefit more significantly. A reconfigurable system is one such example that we elaborate upon next.

2.1 Pin Limitation in Reconfigurable Architectures

In a reconfigurable architecture, the functionalities of the components that make up the architecture and the way in which these components are interconnected can be altered to suit the demands of a particular application [4, 22]. Such architectures are generally composed of a mesh of configurable elements connected by a configurable interconnection network. If

the architecture can be reconfigured with little overhead, then it is said to be *dynamically reconfigurable* [22]. Dynamically reconfigurable architectures can particularly benefit from an increase in the number of input pins to the chip (as we will show later in this section).

Dynamic reconfiguration has two main benefits. First, a dynamically reconfigurable architecture can reconfigure between various stages of an application to use its resources optimally at each stage. That is, it reuses hardware resources more efficiently across different parts of an algorithm. For example, an algorithm using two multipliers in Stage 1 and eight adders in Stage 2 can run on dynamically reconfigurable hardware that configures as two multipliers for Stage 1 and as eight adders for Stage 2. Consequently, this algorithm will run on hardware that has two multipliers or eight adders, as opposed to a non-reconfigurable architecture that would need two multipliers and eight adders.

The second benefit of dynamic reconfiguration is a fine tuning of the architecture to exploit characteristics of a given instance of the problem. For example in matching a sequence to a given pattern, the internal "comparator" structure can be fine-tuned to the pattern. Further, this tuning to a problem instance can also produce faster solutions [22].

However, the benefits of dynamic reconfiguration come at a cost. Dynamic reconfiguration has its architectural and algorithmic overheads and can be difficult to realize [22]. Since the primary motivating factor in our work is reconfigurable computing, we will focus our discussion on this application area; however, other areas may benefit from this work.

In the next section we begin our discussion of the advantages of dynamic reconfiguration in the setting of a theoretical model. Then, in Section 2.1.2 we place this advantage in the context of a practical reconfigurable environment that shows the implications of pin-limitation in this area.

2.1.1 The R-Mesh: A Theoretical Reconfigurable Model

An R-Mesh is a two-dimensional array of processors connected by an underlying mesh network. Each processor has four ports named by the cardinal directions, North, South, East, and West, connecting it to its nearest mesh neighbors. These ports can be connected internally to create seamless buses through multiple processors. As shown in Figure 2.2, there are

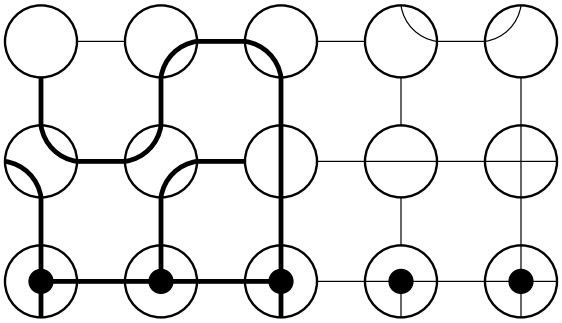


FIGURE 2.2: A 3×5 R-Mesh with all possible port connections; one bus is shown in bold.

a total of 15 possible port connections that allow a rich variety of buses [22]; for clarity, one of the buses is shown in bold. R-Meshes can also be based on meshes of different dimensions. For example, a one-dimensional R-Mesh would be a linear array of processors, each capable of connecting or disconnecting its East and West ports.

As an example of the capabilities of dynamic reconfiguration, consider an N-processor one-dimensional R-Mesh whose processors are partitioned into k groups, each of size $\frac{N}{k}$. A group is considered "flagged" if any processor in the group is flagged. Suppose we wish to determine the number of flagged groups. This problem is easily solved on the one-dimensional R-Mesh in $O(\log k)$ time. In contrast, most other models will require $\Omega(\log N)$ time. In particular, a one-dimensional (non-reconfigurable) mesh (or linear array) will require $\Theta(N)$ time for this problem.

The algorithm proceeds in three separate stages on the R-Mesh. In the first two stages, each of the k groups determines if any processor in the group is flagged. This is based on an algorithm for the finding the OR of N bits [22]. In Stage 1, the first processor of each group disconnects its ports and never uses its West port, thereby disconnecting its group from the previous group, if any (Figure 2.3). With each of the k groups now disconnected, Stage 2 boils down to finding the OR on k separate $\frac{N}{k}$-processor one-dimensional R-Meshes. If the first processor of a group is flagged, then it indicates to the rest of the processors in the group that nothing further need be done. This information can be broadcast on the unique bus that runs through each group. If the first processor of a group is not flagged then we proceed as follows. If a processor other than the first processor of the group is not flagged

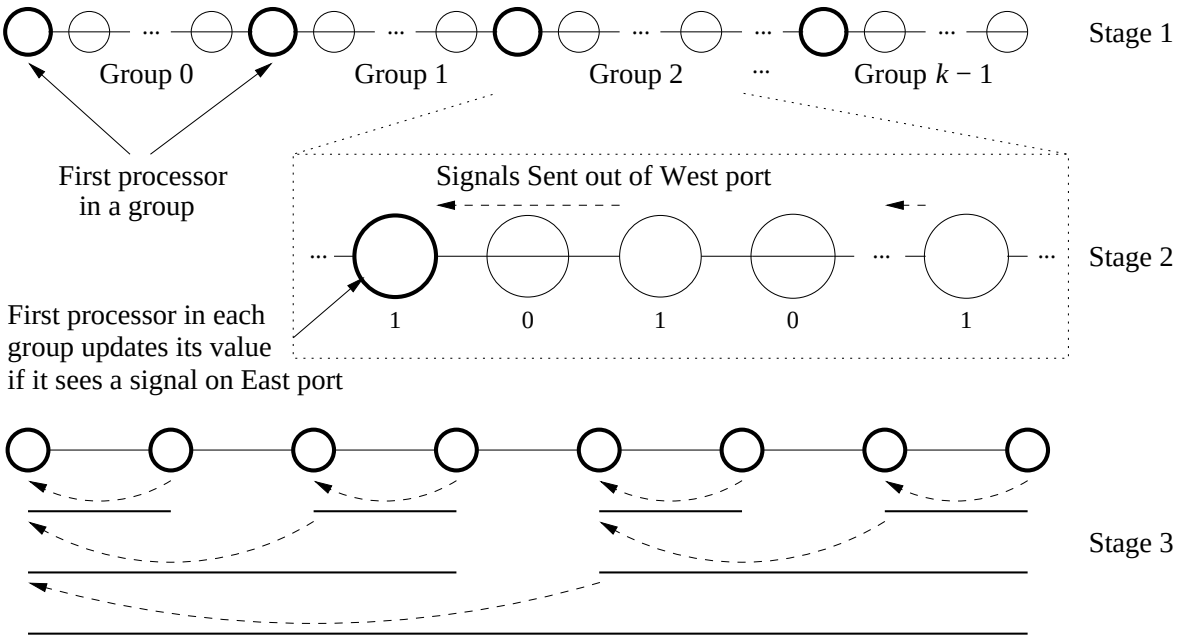


FIGURE 2.3: Finding the number of flagged groups on a one-dimensional R-Mesh

(indicated by a '0' in Figure 2.3), it connects its East and West ports; if a processor is flagged (indicated by a '1' in Figure 2.3), it disconnects its East and West ports (Figure 2.3, Stage 2). Each processor with a flag then sends a signal out its West port. The first processor in the group listens on its East port, and marks its group as flagged if and only if it detects a signal on its East port.

In Stage 3, the number of flags is tallied. Since the first processor in each group now contains all information for that group, this is accomplished using the well-known binary tree paradigm for reduction algorithms [11, 22] to add the values contained in the first processor in each group and store the final result in the first processor of the one-dimensional R-Mesh, i.e., Processor 0. Figure 2.3 illustrates the algorithm with $k = 8$. In general, since this stage of the algorithm is implemented as a balanced binary tree with k leaves (groups), it runs in $O(\log k)$ time. Since the first two stages of the algorithm run in a constant number of steps, the algorithm runs in $O(\log k)$ time. These time complexities hinge on the assumption of a fast, constant delay bus.

This example demonstrates the two main benefits of dynamic reconfiguration (Section 2.1). In Stage 1 and in the steps of Stage 3, the processing elements connect their ports in a manner determined *a priori* in order to optimally use resources for a given stage in the algorithm.

In Stage 2 however, the processing elements connect their ports in a manner determined by the value of their flag; this allows the algorithm to take advantage of the given instance of the problem.

This example also shows several key features of the R-Mesh model that allow it to be dynamically reconfigurable. The first is the coarse-granularity of the R-Mesh's processing elements, which allows them to execute basic instructions in a synchronous environment. In addition, the processing elements can each change their configurations independently. This *connection autonomy* is key to the power of the R-Mesh. The previous example uses a one-dimensional R-Mesh. The two-dimensional (or higher dimensional) R-Mesh is even more powerful. However, the arbitrarily shaped busses of the two-dimensional R-Mesh make it difficult to realize [22]. Thus, while the R-Mesh provides a powerful reconfigurable model, practical considerations lead us to the discussion in the next section of a currently realizable reconfigurable platform, the Field Programmable Gate Array (FPGA).

2.1.2 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a reconfigurable architecture that extends the functionality of a traditional Programmable Logic Device (PLD) [5]. While FPGAs were initially used for rapid prototyping, their ability to configure as any desired circuit allows them to compete favorably with Application Specific Integrated Circuits (ASICs), particularly in low to medium yield situations as the high manufacturing cost and slow design cycles of ASICs are a major disadvantage [5]. As illustrated in Section 2.1.1, dynamic reconfiguration holds tremendous benefits. While current FPGAs are capable of some limited dynamic reconfiguration, they are not as nimble as the R-Mesh in adapting to a given problem. In this section we show how a solution to the pin limitation problem can considerably increase the utility of FPGAs.

A typical FPGA structure consists of a two-dimensional mesh of configurable logic elements connected by a configurable interconnection network [9]. Figure 2.4 shows such a structure, where the Configurable Logic Blocks (CLBs) are the configurable functional elements, and the switches (S) are the configurable elements in the interconnection network.

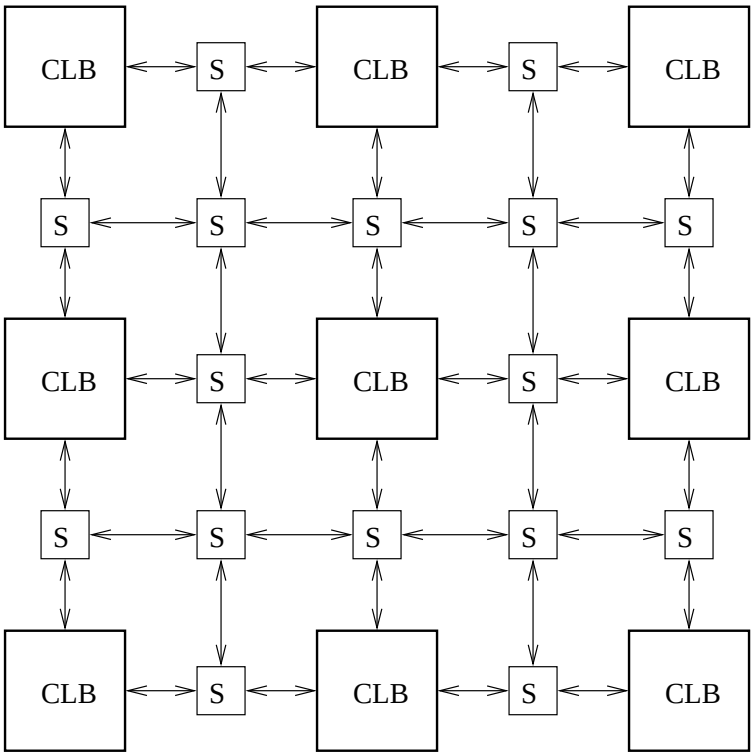


FIGURE 2.4: A typical FPGA structure

Each CLB in an FPGA is sometimes subdivided into smaller configurable logic elements. For example, the Xilinx Virtex-5 FPGA's CLBs each contain two elements known as slices (Figure 2.5). At the deepest level, the most basic functional element in an FPGA usually

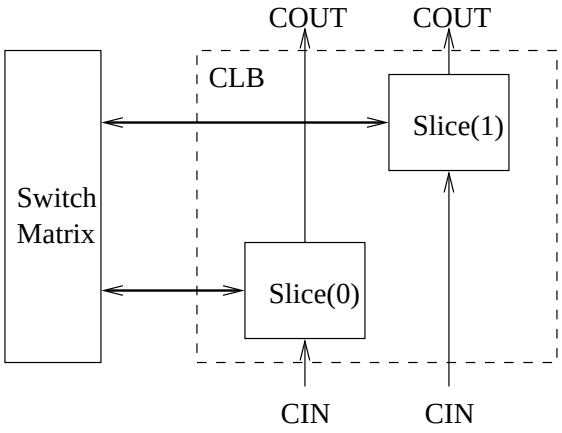


FIGURE 2.5: Xilinx Virtex-5 configurable logic block [28]

consists of some combination of one or more Look-Up Tables (LUTs), combinational logic gates, flip-flops, and other basic logic elements. In the Virtex-5, each slice contains four 64×1 LUTs, four flip-flops, an arithmetic and carry chain, and several multiplexers used to

combine the outputs of the LUTs [28]. Often the CLBs in an FPGA are also interspersed with other functional units, such as small memory blocks, other adder chains, and multipliers. Thus, a CLB can contain many configurable switches. Notwithstanding variations in FPGA terminology, we will use the term "CLB" to denote the basic unit represented in Figure 2.4.

The FPGA's interconnection network is typically a two-dimensional mesh of configurable switches. As in a CLB, each switch S represents a large bank of configurable elements. The state of all switches and elements within all CLBs is referred to as a "configuration" of the FPGA. Because there is a large number of configurable elements in an FPGA (LUTs, flip-flops, switches, etc.), a single configuration requires a large amount of information. For example, the Xilinx Virtex-5 FPGA with a 240×108 array of CLBs requires in the order of 79 million bits for a single configuration [28, 29]. Unlike the coarse-grained processing elements of the R-Mesh (Section 2.1.1), the FPGA's CLBs are fine-grained functional elements that are incapable of executing instructions or generating configuration bits internally. Thus, configuration information must come from outside the chip. A limited amount of configuration information can be stored in the chip as "contexts;" however, given the limited amount of memory available on an FPGA for such a purpose, an application may require more contexts than can be stored on the FPGA. Hence, in most cases, configuration information must still come from outside the chip.

When used as an electronic breadboard, an FPGA can be configured off-line with no regard to the amount of time needed for configuration. In this work we deal primarily with dynamic reconfiguration, for which an FPGA must reconfigure while an application is executing on it. As we noted earlier, since most configuration information must come from outside the chip and the number of bits needed for a configuration is large, reconfiguration is time consuming. Because of this, only selected parts of the FPGA are configured in order to avoid large overheads. This mode of reconfiguring is called *partial reconfiguration* [2, 22, 27, 29].

In partial reconfiguration, the information entering the chip can be classified into two categories: (a) selection and (b) configuration. The selection information contains the addresses

of the elements that require reconfiguration, while the configuration information contains the necessary bits to set the state of the targeted elements.

In order to facilitate partial reconfiguration, FPGAs are typically divided into sets of *frames*, where a frame is the smallest addressable unit for reconfiguration. In current FPGAs, a frame is typically one or more columns of CLBs. Currently, partial reconfiguration can only address and configure a single frame at a time. If we assume that each CLB receives the same number of configuration bits, say α, and the number of CLBs in each frame is the same, say C, then the number of configuration bits needed for each frame is $C\alpha$. If the number of bits needed for selecting a frame is b, then the total number of bits B needed to reconfigure a frame is:

$$B = b + C\alpha$$

Since the granularity of reconfiguration is at the frame level, every CLB in a frame would be reconfigured, regardless of whether or not the application required them to be reconfigured. This can result in a "poorly-focused" selection of elements for reconfiguration, as more elements than necessary are reconfigured in each iteration. This implies that a large number of bits and a large time overhead are spent on the reconfiguration of each individual frame. If the granularity of selection is increased, i.e., if fewer CLBs are in each frame, then the number of selection bits needed to address the frames increases while the number of configuration bits for each frame decreases. However, this also increases the total number of iterations necessary to reconfigure the same amount of area in the FPGA. As an illustration of this trade-off, consider the following two examples of reconfiguring an 8×12 FPGA.

Example 2.1 Consider an 8×12 FPGA divided into 4 frames, where each frame contains three columns of CLBs (Figure 2.6). Assume the number of configuration bits per CLB to be $\alpha = 6$. Since there are 4 frames, the number of selection bits $b = 2$. Since each frame contains 24 CLBs, the number of configuration bits for each frame is $C\alpha = 144$. If the shaded CLBs shown in Figure 2.6 require reconfiguration, then we require 4 iterations of 146 bits each, for a total of $4 \times (b + C\alpha) = 584$ bits. (Recall that only one frame can be selected at a time and that the entire frame must be reconfigured.) Since only 48 bits are necessary

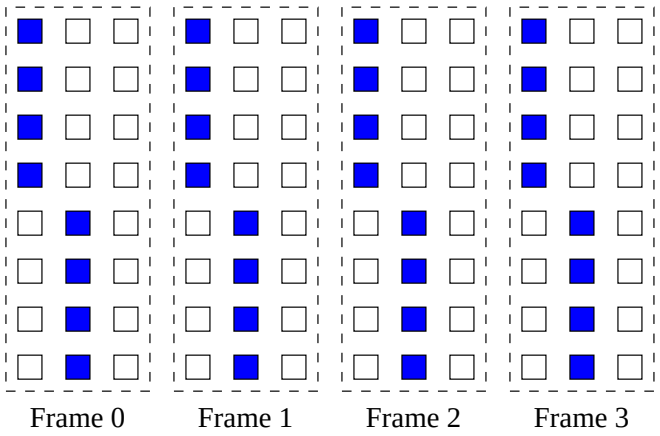


FIGURE 2.6: Reconfiguration of an 8×12 FPGA of Example 2.1

to reconfigure the desired CLBs in each frame, each iteration of reconfiguration requires 96 bits more than necessary.

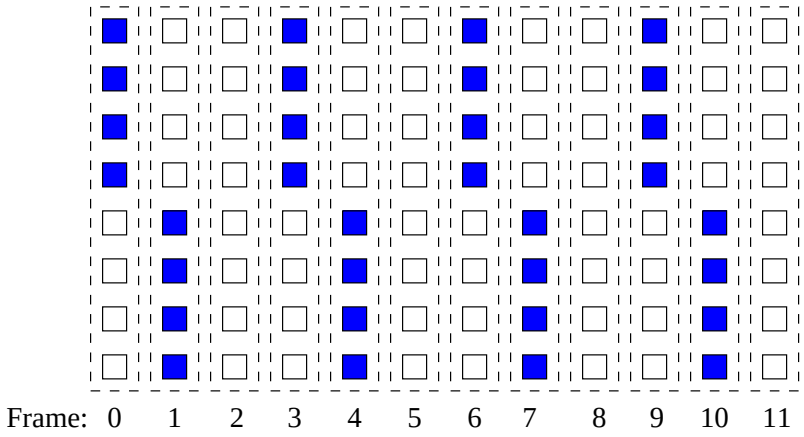


FIGURE 2.7: Reconfiguration of an 8×12 FPGA of Example 2.2

Example 2.2 As a second example, consider the same 8×12 FPGA divided this time into 12 frames, where each frame contains a single column of CLBs (Figure 2.7). Again, let $\alpha = 6$. Since there are 12 frames, the number of selection bits b is 4. Since each frame contains 8 CLBs, the number of configuration bits for each frame $C\alpha$ is 48. If the shaded CLBs in Figure 2.7 require reconfiguration, then we would need 8 iterations of 52 bits each, for a total of $8 \times (b + C\alpha) = 416$ bits. Since only 24 bits are necessary to reconfigure the desired CLBs in each frame, each iteration of reconfiguration requires 24 bits more than necessary.

Thus, targeting only those elements that require reconfiguration is desirable as it can decrease the number of configuration bits ($C\alpha$). While this increases the number of selection

bits b, this increase is generally small compared to $C\alpha$. In the extreme, if a frame consisted of a single CLB, then reconfiguration would require selection bits for both the rows and the columns of the FPGA, but would reconfigure only those CLBs that required reconfiguration based on the application. However, since a typical FPGA selects only one element at a time, this is not practical as the number of iterations would be prohibitive. If the ability to quickly select only those elements that require reconfiguration is not available, then a good design choice must weigh the benefits of better focus (smaller frames in Example 2.2) with the penalty of a larger number of iterations. Since different applications demand different patterns of reconfiguration, a simple "one-size-fits-all" solution is seldom efficient. Consequently, current FPGAs often do not fully exploit the benefits of dynamic reconfiguration that a problem holds.

Another advantage of flexibility in selecting entities within an FPGA is in the area of *configuration contexts*. A configuration context is a long stream of bits, one per configurable element of the FPGA. This context is typically distributed across several LUTs. For example, an FPGA with a 16-location context LUT in each CLB may hold 16 contexts, each distributed over all context LUTs. Loading context 7, for example, would load the contents of location 7 from each of these context LUTs. Thus, the entire FPGA can hold no more than 16 different contexts. If it is possible to select a location (say 7) from some of the LUTs, and a different location (say 6) from the rest, $16^2 = 256$ contexts are possible. In the extreme, where each LUT can be individually addressed, the flexibility approaches the connection autonomy of the R-Mesh. As in partial reconfiguration, a good mechanism for selecting context LUTs is advantageous here.

Pin limitation thus creates a severe restriction on the extent to which an FPGA can be dynamically reconfigured. Clearly more pins will allow parallel input of several configuration bits. We now explore possible solutions to the pin limitation constraint and introduce our approach in the next section.

2.2 Approaches for the Pin Limitation Constraint

There are a number of ways of alleviating the effects of the pin limitation problem. These include (1) multiplexing, (2) storing information within the design, and (3) decoding. The first two approaches are discussed briefly. Our solution is the decoding approach.

Multiplexing: The concept of multiplexing refers to combining a large number of channels into a single channel. This can be accomplished in a variety of ways depending on the technology. Each method assumes the availability of a very high speed, high bandwidth channel on which the multiplexing is performed. For example, in the optical domain, wavelength division multiplexing allows multiple signals of different wavelengths to travel simultaneously in a single waveguide. While some optoelectronic FPGAs have been proposed [23, 24], this is far from practice. Time division multiplexing requires the multiplexed signal to be much faster than the signals multiplexed. Used blindly, this is largely useless in the FPGA setting, as it amounts to setting an unreasonably high clocking rate for the FPGA. A more innovative use of multiplexing is described below.

Storing Information Within the Design: This attempts to alleviate the pin limitation problem by generating most information needed for execution of an application inside the chip itself (as opposed to importing it from outside the chip). This requires a more "intelligent" chip. In an FPGA setting it boils down to an array of coarse grained processing elements rather than simple functional blocks (CLBs). One example is the use of virtual wires [3], in which each physical wire corresponding to an I/O pin is multiplexed among multiple logical wires. The logical wires are then pipelined at the maximum clocking frequency of the FPGA, in order to utilize the I/O pin as often as possible. Another example of such a solution is the Self-Reconfigurable Gate Array (SRGA) architecture [12, 21]. However, this approach is a significant departure from current FPGA architectures.

Decoders: A decoder is typically a combinational circuit that takes in as input a relatively small number of bits, say x bits, and outputs a larger number of bits, say n bits, according

to some mapping; such a decoder is called an x to n decoder. If the x inputs are pins to the chip and the n outputs are expanded within the chip, a decoder provides the means to deliver a large number of bits to the interior of the chip (see Figure 2.8). An x to n decoder can clearly produce no more than 2^x output sequences.

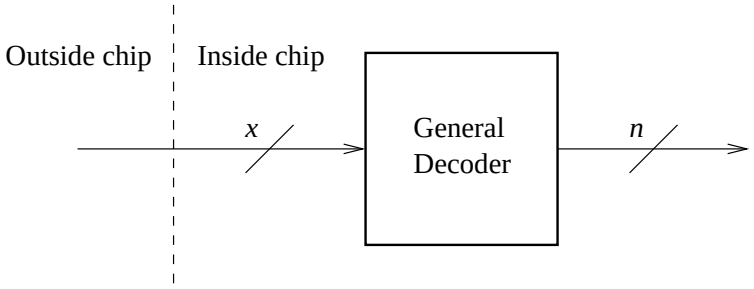


FIGURE 2.8: An x to n decoder in an IC chip

If $\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$, then each output sequence of the decoder can be interpreted a subset of $\mathcal{Z}_n$. Let $\mathcal{S}$ be a set of "desired" subsets of $\mathcal{Z}_n$ that need to be generated by the decoder. For example, let $n = 8$ and $\mathcal{Z}_n = \{0, 1, 2, \ldots, 7\}$. For this example, Table 2.2 shows

TABLE 2.2: An illustration of a decoder for four sets $\mathcal{S}_i$ of subsets of $\mathcal{Z}_8$

Decoder Inputs	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$
000	00000001	01010101	11111111	00001101
001	00000010	10101010	00001111	10010010
010	00000100	00110011	00000011	10100010
011	00001000	11001100	00000001	00111101
100	00010000	00001111	11110000	01001110
101	00100000	11110000	11000000	11010001
110	01000000	11111111	10000000	11100001
111	10000000	00000000	00111100	01111110

a decoder input sequence with four corresponding sets of subsets, $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, where:

$\mathcal{S}_0 = \{\{0\}, \{1\}, \{2\}, \ldots, \{7\}\}$, the subsets for a 1-hot decoder,

$\mathcal{S}_1 = \{\{0, 2, 4, 6\}, \{1, 3, 5, 7\}, \ldots, \emptyset\}$, the subsets representative of ASCEND/DESCEND communication patterns [1],

$\mathcal{S}_2 = \{\{0,1,2,3,4,5,6,7\},\{0,1,2,3\},\ldots,\{2,3,4,5\}\}$, the subsets representative of a type of reduction [11, 22] and

$\mathcal{S}_3 = \{\{0,2,3\},\{1,4,7\},\ldots,\{2,3,4,5,6\}\}$, an "arbitary" collection of subsets.

For example, if the decoder produces $\mathcal{S}_2$, then, for input '100', the output is '11110000'.

In an FPGA setting a subset $S_j \in \mathcal{S}$ typically represents a subset of CLBs that need to be reconfigured. To accomplish this reconfiguration in one iteration, the decoder must generate a superset $\widehat{S}_j$ of S_j, i.e., $\widehat{S}_j \supseteq S_j$. Then, reconfiguring all CLBs of $\widehat{S}_j$ and reloading existing states of all CLBs of $\widehat{S}_j - S_j$ achieves the desired effect.

We now identify some "performance properties" of a decoder operating in this setting.

- Cost: The hardware cost of the decoder, typically given as the number of gates.
- Speed: The amount of time needed to generate $\widehat{S}_j$ for any $\widehat{S}_j \in \mathcal{S}$. This could be some function of the delay of the decoder and the number of iterations over which the decoder generates $\widehat{S}_j$.
- Focus: This is $\max\{|\widehat{S}_j - S_j| : S_j \in \mathcal{S}\}$. This parameter measures how accurately the decoder can generate the required subset S_j.
- Flexibility: This is an indication of how easily $\mathcal{S}$ can be altered.

Current decoders in FPGAs are 1-hot decoders that expand $\log n$ input bits to n output bits with only one of the n output bits set to 1 (such as the set $\mathcal{S}_1$ in Table 2.2). These have a low $\Theta(n \log n)$ cost and high speed ($\Theta(\log n)$ delay per iteration). However, as noted in Section 2.1.2, this allows FPGAs to only reconfigure one frame at a time (requiring multiple iterations), and is also not flexible (as the decoder cannot produce any other set of subsets $\mathcal{S}_i$). Because a frame in current FPGAs contains many CLBs, the focus of current FPGA decoders can be very poor (as was illustrated in Examples 2.1 and 2.2). Thus, we look to design a decoder with low cost ($\Theta(n \log n)$) and high speed ($\Theta(\log n)$ delay per iteration) that is flexible and can provide a higher degree of focus for addressing interesting sets (such as $\mathcal{S}_3$ in Table 2.2). Achieving all this will also ensure a small number of iterations. We begin a discussion of our solution in the next chapter.

Chapter 3
Preliminaries

As we observed in Chapter 2, the approach we adopt is the use of decoders to alleviate the pin limitation constraint. Unlike conventional "fixed" decoders, we propose a "configurable" decoder that has the speed and low cost of fixed decoders but with considerably higher flexibility and focus in selection.

This chapter introduces the basic ideas needed for the remainder of this thesis. We begin in Section 3.1 by outlining the assumptions and notation used throughout our work. In Section 3.2, we define the building blocks that are used to construct the various versions of a configurable decoder. Finally, in Section 3.3 we introduce the structure of the configurable decoder itself before providing more details in subsequent chapters.

3.1 Assumptions and Notation

A configurable decoder is a combinational circuit (with the exception of the bit-slice units detailed in Section 5.3), that, in order to achieve a degree of flexibility, uses Look-Up Tables (LUTs). While LUTs could be implemented using sequential elements, for this work, LUTs are functionally identical to combinational memory such as ROMs. Some of the ideas we discuss here assume the configurable decoder to be a combinational circuit. The minor extensions needed for the sequential circuits of Sections 5.3 will be discussed later.

To allow us to work at a conveniently abstract level while accounting for realistic constraints, we make the following assumptions.

1. All gates are assumed to have a constant fan-in and fan-out of at least 2; that is, the maximum number of inputs to a gate and the maximum number of other gates driven by the output of a given gate are independent of the problem size. When the fan-out of a signal in a circuit exceeds the driving capacity of a gate, buffers are inserted into the design. These additional buffers increase the cost and delay of the circuit. Gates typically have a fixed number of inputs. Realizing gates with additional inputs boils

down to constructing a tree of gates. Assuming a nonconstant fan-in and fan-out would ignore the additional gate cost and delay imposed by these elements.

2. We assume that each instance of a gate has unit cost and delay. While the cost and delay of some logic gates (such as XOR) is certainly larger than the size and delay of smaller logic gates (such as NAND in some technologies), the overall number of gates in the circuit and the depth of the circuit provide a better measure of the circuit's costs, rather than factors arising from choices specific to a technology and implementation.

3.1.1 Performance Parameters

We divide the performance parameters into two categories: independent parameters and problem dependent parameters. Independent parameters are applicable to all circuits, while problem dependent parameters are specific to decoders. All parameters are expressed in terms of their asymptotic complexity to avoid minor variations due to technology and other implementation-specific details.

Independent Parameters:

Gate Cost G**:** the gate cost of a circuit is the number of gates (AND, OR, NOT) in it. Clearly, the use of other gates such as NAND, XOR, etc. will not alter the gate cost expressed in asymptotic notation.

Delay D**:** the delay or time cost of a combinational circuit is the length of the longest path from any input of the circuit to any output.

Problem Dependent Parameters: As we noted earlier in Chapter 2, the basic function of a decoder can be interpreted as that of selecting subsets of a set. Consider an x to n decoder. Functionally, this decoder accepts x input bits and produces n output bits, where $x \ll n$. Since the decoder is a combinational circuit, x input bits produce at most 2^x different outputs. Each n-bit output can be interpreted as a subset of an n-element set[1]

[1]We call a set of n elements an n-set and a subset of an n-set as an n-subset.

$\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$ using the standard characteristic function representation, that is, each bit position of the n-bit output corresponds to an element of $\mathcal{Z}_n$ and the bit value indicates membership status. As an example of this notation, consider $n = 8$ and $\mathcal{Z}_n = \{7, 6, \ldots, 0\}$. Then an 8-bit binary string '0001101' represents the subset $\{4, 3, 0\}$, where the leftmost position is for element 7 and the rightmost for element 0.

For any integer $n \geq 1$, let $\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$ be the set whose subsets the decoder is to produce. Let $\wp(\mathcal{Z}_n)$ denote the powerset of $\mathcal{Z}_n$, that is, the set of all subsets of $\mathcal{Z}_n$; clearly,[2] $|\wp(\mathcal{Z}_n)| = 2^n$. We can represent the elements of $\wp(\mathcal{Z}_n)$ by the 2^n values of an n-bit string. Let the decoder produce the set $\mathcal{S}' \subseteq \wp(\mathcal{Z}_n)$ of subsets of $\mathcal{Z}_n$ and let $|\mathcal{S}'| = \Lambda$.

In summary, the problem is to construct a decoder with x-bit inputs and n-bit outputs such that the set $\mathcal{S}$' of subsets it generates "is sufficient" for the application at hand. Different applications require different sets of subsets of $\mathcal{Z}_n$, and do so with different constraints on speed and cost.

For a configurable decoder, a portion of the hardware can be changed (off-line). This allows one to freely select a portion of the subsets produced by the decoder. Let $\mathcal{S} \subseteq \mathcal{S}'$ denote the portion of subsets that can be produced independently (by configuring the decoder in any manner of choice). Ideally, $\mathcal{S} = \mathcal{S}'$, but in some variants of the configurable decoder, this may make the cost prohibitive (see Section 3.3). In others, $\emptyset \subset \mathcal{S} \subset \mathcal{S}'$; that is, the decoder allows some of the subsets to be chosen arbitrarily while others are defined by this choice. In fixed (non-reconfigurable) decoders, $\mathcal{S} = \emptyset$. From this perspective, we define the following two parameters that are specific to decoders.

1. Number of independent subsets: $\lambda = |\mathcal{S}|$

2. Total number of subsets: $\Lambda = |\mathcal{S}'|$

Since flexibility is important for a reconfigurable platform, the number of independent subsets, rather than simply the total number of subsets, is emphasized in this thesis.

[2]For any set $\mathcal{S}$, we denote its cardinality by $|\mathcal{S}|$.

3.1.2 Other Notation and Concepts

The following notation is used throughout this thesis. In general we will denote inputs, outputs, and intermediate signals by uppercase letters such as A, B, Q, etc. Each of these signals could be several bits wide. The number of bits in a signal A is denoted $\#(A)$, and the bits themselves are denoted by $A(\#(A)-1), \ldots, A(1), A(0)$. The signal A can take up to $2^{\#(A)}$ values. We will denote the set of values that signal A can have also by the letter A; typically the context will make the distinction clear. Thus, if a is a $\#(A)$-bit value that signal A can have, then we will say that $a \in A$.

We now briefly discuss some well known concepts, as they will be used extensively in Chapter 4.

All logarithms are to base 2. For any $n > 0$, and any integer $k \geq 0$, $\log^k n = (\log_2 n)^k$, whereas $\log^{(k)} n = \underbrace{\log \log \ldots \log}_{k \text{ times}} n$. Note that $\log^{(k)} n \neq \log^k n \neq \log n^k = k \log n$.

Let S be an n-element set. A *partition* [17] of S is a division of the elements of the set into disjoint non-empty subsets, $S_0, S_1, \ldots, S_{k-1}$. More formally, a partition π of set S is a set of nonempty subsets $\{S_0, S_1, \ldots, S_k\}$ such that $\bigcup_{i=0}^{k-1} S_i = S$ and $S_i \cap S_j = \emptyset$, for $i \neq j$. A partition π with k blocks ($0 \leq k < n$) is called a *k-partition of S*. For example, a 3-partition of the set $S = \{8, 7, 6, 5, 4, 3, 2, 1, 0\}$ is $\pi = \{\{7, 6, 5, 4\}, \{3, 2\}, \{1, 0\}\}$.

A useful operation on partitions is the *product* of two partitions π_1 and π_2, which can be defined as follows. Let $\pi_1 = \{S_0, S_1, \ldots, S_k\}$ and $\pi_2 = \{P_0, P_1, \ldots, P_\ell\}$ be partitions of set S. Define the product $\pi_1\pi_2$ of π_1 and π_2 to be a partition $\{Q_0, Q_1, \ldots, Q_m\}$ such that for any block $Q_h \in \pi_1\pi_2$, elements $a, b \in Q_h$ if and only if there exists blocks $S_i \in \pi_1$ and $P_j \in \pi_2$ such that $a, b \in S_i \cap P_j$. As an example, consider the partitions $\pi_1 = \{\{7, 6, 5, 4\}, \{3, 2\}, \{1, 0\}\}$ and $\pi_2 = \{\{7, 6\}, \{5, 4, 3, 2\}, \{1, 0\}\}$. Then $\pi_1\pi_2 = \{\{7, 6\}, \{5, 4\}, \{3, 2\}, \{1, 0\}\}$.

3.2 Building Blocks

In this section we define and analyze some basic hardware structures that will serve as building blocks for subsequent designs. Specifically, we describe fan-in and fan-out circuits, 1-hot decoders, multiplexers, look-up tables (LUTs), shift registers, and modulo-α counters.

As these building blocks are used extensively in subsequent chapters, we summarize their asymptotic costs and delays in Table 3.1.

TABLE 3.1: Asymptotic gate cost and delay of building blocks

Building Block	Gate Cost	Delay
Fan-in and Fan-out	$O(fz)$	$O(\log f)$
1-hot Decoder	$O(z2^z)$	$O(z)$
Multiplexer	$O(z2^z)$	$O(z)$
LUT	$O(2^z(z+m))$	$O(z+\log m)$
Shift Register $\left(\alpha, \frac{z}{\alpha}\right)$	$O(z)$	$\Theta(1)$
Modulo-α counter	$O(\log^2 \alpha)$	$O(\log\log \alpha)$

z = number of input bits.

m = number of output bits.

f = fan-in or fan-out of a signal.

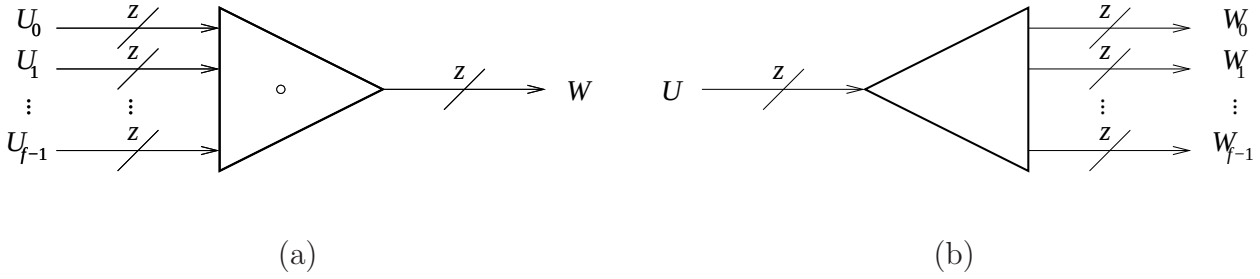
3.2.1 Fan-in and Fan-out

While we assume that the fan-in and fan-out of logic gates is a constant, signals may have to be fanned in from or fanned out to a non-constant number of places. In these cases, gates are inserted into the design to provide additional driving and fan-in capabilities. Since these additional elements increase the cost and delay of the circuit, we discuss a general method to fan-in a signal from more than a constant number of places and fan-out a signal to more than a constant number of places.

If the number of places the signal is fanned-in from or fanned-out to is f and the width of the signal is z-bits, we denote this as a fan-in and fan-out problem of degree f and width z, respectively. The fan-in and fan-out problems can be stated as follows.

For integers $f, z \geq 1$, let $U_0, U_1, \ldots, U_{f-1}$ be f signals each z-bits wide. A "fan-in circuit" of degree f and width z (Figure 3.1 (a)) produces a z-bit output W such that for any $0 \leq i < z$,

$$W(i) = U_0(i) \circ U_1(i) \circ \ldots \circ U_{f-1}(i),$$



(a) (b)

FIGURE 3.1: Fan-in problem (a) and fan-out problem (b) of degree f and width z

where $\circ$ is an associative Boolean binary operation.

For integers $f, z \geq 1$, let U be a signal z-bits wide. A "fan-out circuit" of degree f and width z (Figure 3.1 (b)) produces f z-bit outputs $W_0, W_1, \ldots, W_{f-1}$ such that for any $0 \leq i < z$ and $0 \leq j < f$,

$$W_j(i) = U(i).$$

Using a standard balanced tree construction, we have the following result.

Lemma 3.1 *Fan-in and fan-out circuits of degree f and width z can be constructed with a gate cost of $O(fz)$ and a delay of $O(\log f)$.*

Proof: A balanced binary tree with f leaves has $\Theta(f)$ nodes and $\Theta(\log f)$ depth. Having z such trees results in $\Theta(fz)$ nodes, each with a depth of $\Theta(\log f)$. ■

3.2.2 Fixed Decoders – 1-hot Decoders

As stated previously (Section 2.2), one method of rating the performance of a decoder is its flexibility, that is, how easily the set of outputs of the decoder (that is, the set $\mathcal{S}$) can be changed. This divides decoders into two broad classifications: fixed decoders, which are inflexible ($\mathcal{S}$ cannot be changed), and configurable decoders, where $\mathcal{S}$ can be changed in some manner (typically off-line). This section explores an important fixed decoder known as a 1-hot decoder; many ideas developed for 1-hot decoders are applicable to other fixed decoders as well. Section 3.3 will introduce implementations of configurable decoders.

In a z to n fixed decoder (Figure 3.2), the manner in which the z-bits of the input signal

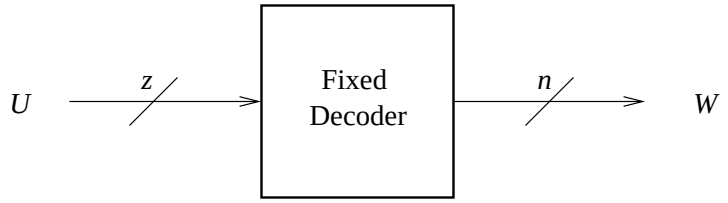


FIGURE 3.2: A fixed z to n decoder

U are expanded to the n-bit output signal W is fixed at manufacture. Since there are 2^z possible values for a z-bit signal, the above fixed decoder can have up to 2^z possible n-bit outputs.

TABLE 3.2: Two possible 1-hot bit patterns for $z = 3$, $n = 8$

z-bit Input	Active High Output	Active Low Output
000	00000001	11111110
001	00000010	11111101
010	00000100	11111011
011	00001000	11110111
100	00010000	11101111
101	00100000	11011111
110	01000000	10111111
111	10000000	01111111

A common selection pattern that is desirable for many applications is the 1-hot bit pattern, an example of which can be seen in Table 3.2. For a 1-hot decoder, $n = 2^z$ and each of the n-bit patterns has only one active bit (usually with a value of '1'), all other bits being inactive (usually '0'). This decoder selects one element at a time. Usually, such a decoder also has a select input that allows the output set to be $\emptyset$. The 1-hot decoder is used so often that the term "decoder" is usually taken to mean a 1-hot decoder. Currently, FPGAs also use 1-hot decoders to select frames during partial reconfiguration (Section 2.1.2).

A simple $\log n$ to n 1-hot decoder implementation (for example, [6, 26]) consists of n AND gates with true and complementary versions of the input bits. Since a bit of the output

signal must be '1' if and only if all other output bits are '0', the AND gate corresponding to a given bit in the output has a unique sequence of true and complementary versions of the input bits. Figure 3.3 illustrates this for a 4 to 16 1-hot decoder. The basic idea of this implementation is to identify the min-term that causes a bit to be active. Since each output is active on exactly one input combination, a simple gate (of sufficiently large fan-in) suffices. In general for a z to 2^z 1-hot decoder, each input fans-out to 2^z gates and each gate accepts z inputs. Thus, a general implementation has the form shown in Figure 3.4. We now have the following result. In Chapter 9 we outline a more sophisticated approach that implements a 1-hot decoder of $\Theta(n)$ cost.

Lemma 3.2 *For any $z \geq 1$, a z to 2^z 1-hot decoder can be implemented as a circuit with a cost of $O(z2^z)$ and a delay of $O(z)$.*

Proof: The fan-out circuit of Figure 3.4 has a delay of $\Theta(z)$ and cost $\Theta(z2^z)$ (see Lemma 3.1). Each of the 2^z fan-in circuits of Figure 3.4 has a delay of $\Theta(\log z)$ and a cost of $\Theta(z)$. So, the total delay is $\Theta(z + \log z) = \Theta(z)$ and the total cost is $\Theta(z2^z + z2^z) = \Theta(z2^z)$. ■

Remark: Often larger decoders are built using smaller decoders as building blocks. This amounts to using the construction of Figure 3.4.

In general, it is difficult to predict the exact cost of a fixed decoder. One class of fixed decoders where input bits are simply fanned out to form output bits that has a low cost is used in our result (see Chapters 4 and 5).

3.2.3 Multiplexers

A multiplexer (MUX) is a combinational circuit that selects information from many inputs and directs it to a single output line (for example, [6]). In general, a 2^z to 1 multiplexer takes 2^z data bits,[3] and using z control bits, selects one of the 2^z data inputs and directs it to a single output line (Figure 3.5).

[3]For the purpose of this work, we will assume that the multiplexer takes in as input 2^z 1-bit data signals. In general, these signals could be replaced with signals of any width w with no change to its delay but with an added $\Theta(w)$ factor to the gate cost.

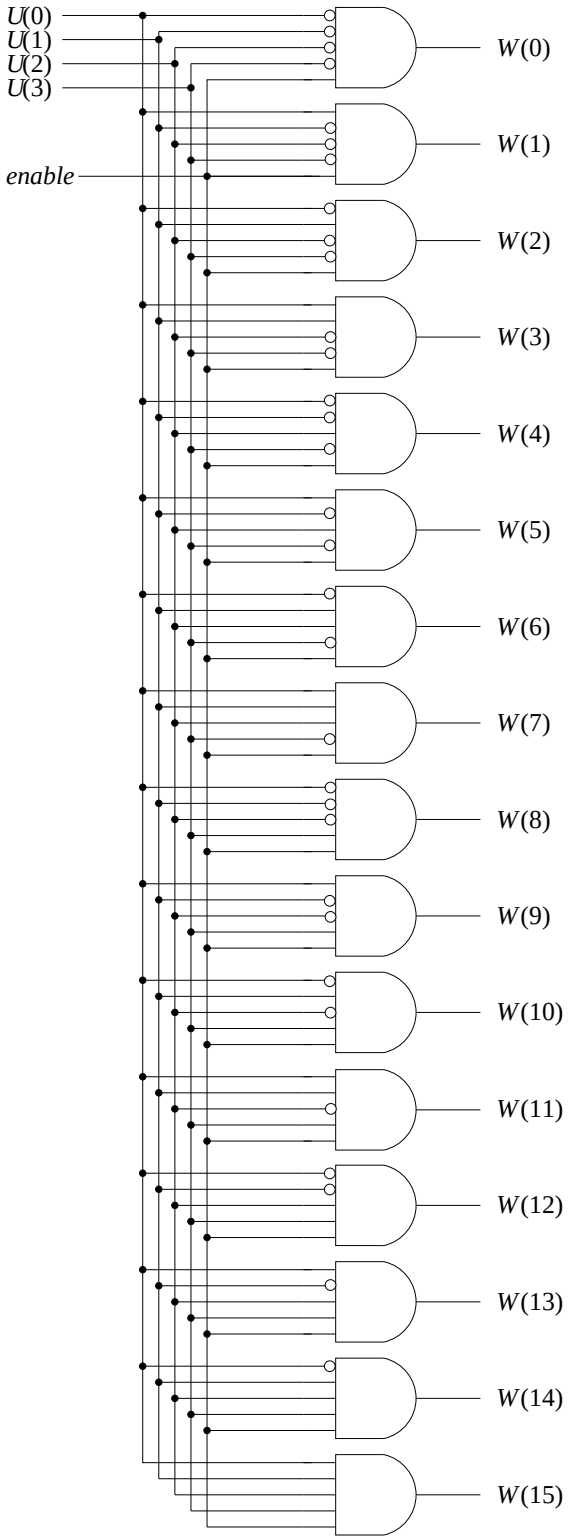


FIGURE 3.3: A logic circuit for a 4 to 16 1-hot decoder. Note the use of an enable signal to force the output of the decoder to $\emptyset$.

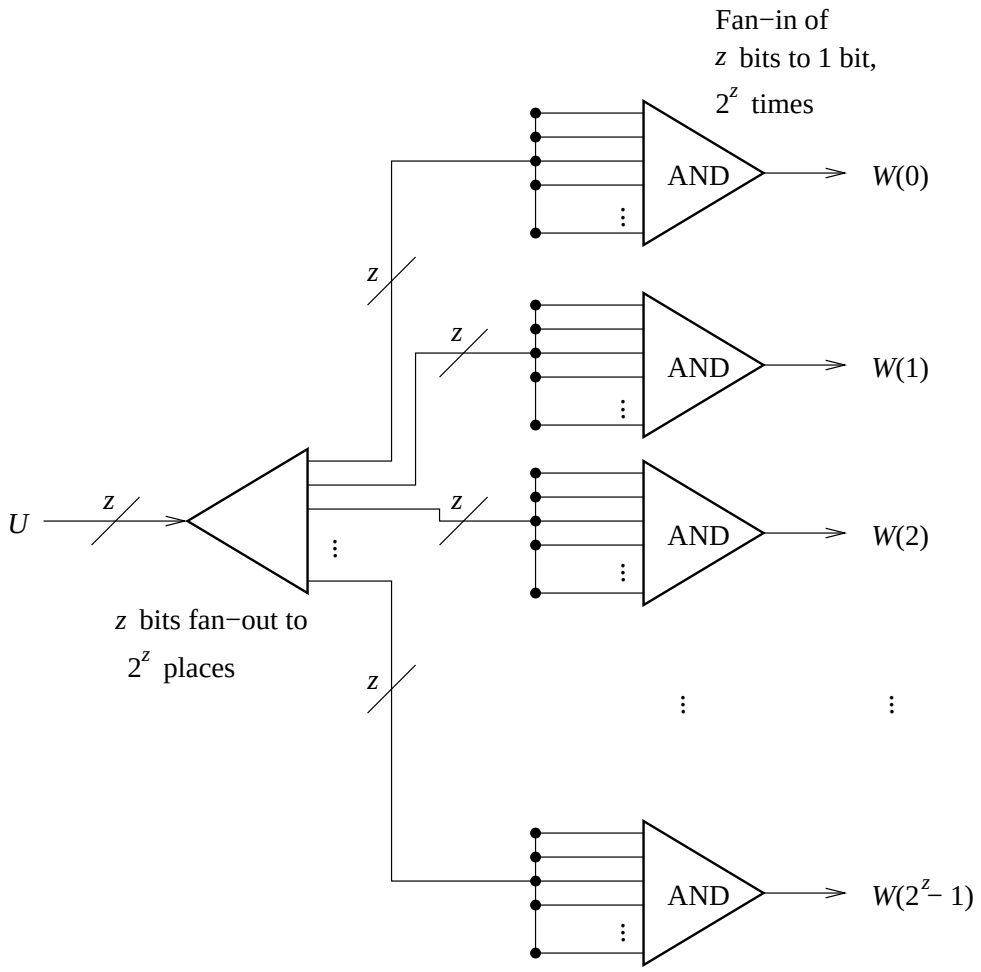


FIGURE 3.4: General implementation of a 1-hot decoder

A 2^z to 1 multiplexer can be constructed as a combinational circuit using 2^z AND gates, each with $(z+1)$ inputs, and a 2^z-input OR gate. An example of such a multiplexer with four inputs is shown in Figure 3.6. Each of the four data inputs, U_0, U_1, U_2, U_3, is selected via an AND gate and a combination of the two control bits $V(0)$ and $V(1)$, much like in the 1-hot decoder. The logic in Figure 3.6 generalizes to the following result.

Lemma 3.3 *A 2^z to 1 multiplexer can be implemented as a circuit with a gate cost of $O(z2^z)$ and a delay of $O(z)$.*

Proof: Each of the z selection bits are required to select one of the 2^z inputs to the multiplexer. This requires a fan-out of the z selection bits to 2^z places, requiring a gate cost of $O(z2^z)$ and a delay of $O(\log z)$ (Lemma 3.1). Each of the 2^z inputs are then combined with the z selection bits. This requires 2^z AND gates, each with a fan-in of $z+1$ bits, repeated

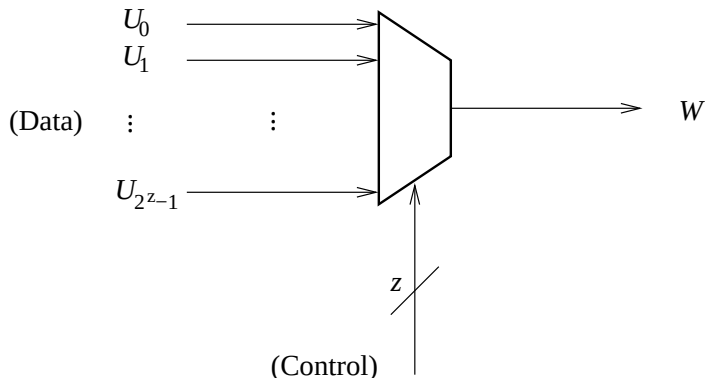


FIGURE 3.5: Multiplexer block diagram

2^z times. By Lemma 3.1, this implies a gate cost of $O(z2^z)$ and a delay of $O(\log z)$. Finally, each of the 2^z-bits resulting from the previous fan-in operations must be fanned-in to a single output using an OR gate. By Lemma 3.1, this has a gate cost of $O(2^z)$ and a delay of $O(z)$.

Overall, the multiplexer has a gate cost of $O(z2^z + z2^z + 2^z) = O(z2^z)$ and a delay of $O(\log z + \log z + z) = O(z)$. ■

3.2.4 Look-up Table

A $2^z \times m$ look-up table (LUT) is a storage device (Figure 3.7) with $m2^z$ storage cells organized as 2^z m-bit words. This LUT has as input z-bits to address the 2^z locations and outputs an m-bit word. While a LUT can act as a basic memory device, LUTs have a variety of other applications, such as implementing small logic functions. A $2^z \times m$ LUT can implement any m Boolean functions of z variables by storing its truth tables [6]. This is of particular use in FPGAs, where Static Random Access Memory (SRAM) based LUTs with four to six inputs are commonly used to implement Boolean functions [19].

While LUTs can be implemented in a variety of ways, all LUTs require the same two components: a memory array and a method of addressing a word in the memory array. One possible method of addressing the LUT is to use a z to 2^z 1-hot decoder. The output of the 1-hot decoder activates a wordline and enables the outputs of the memory storage cells. Each of the memory storage cell outputs are then fanned-in to form a m-bit output word (Figure 3.8).

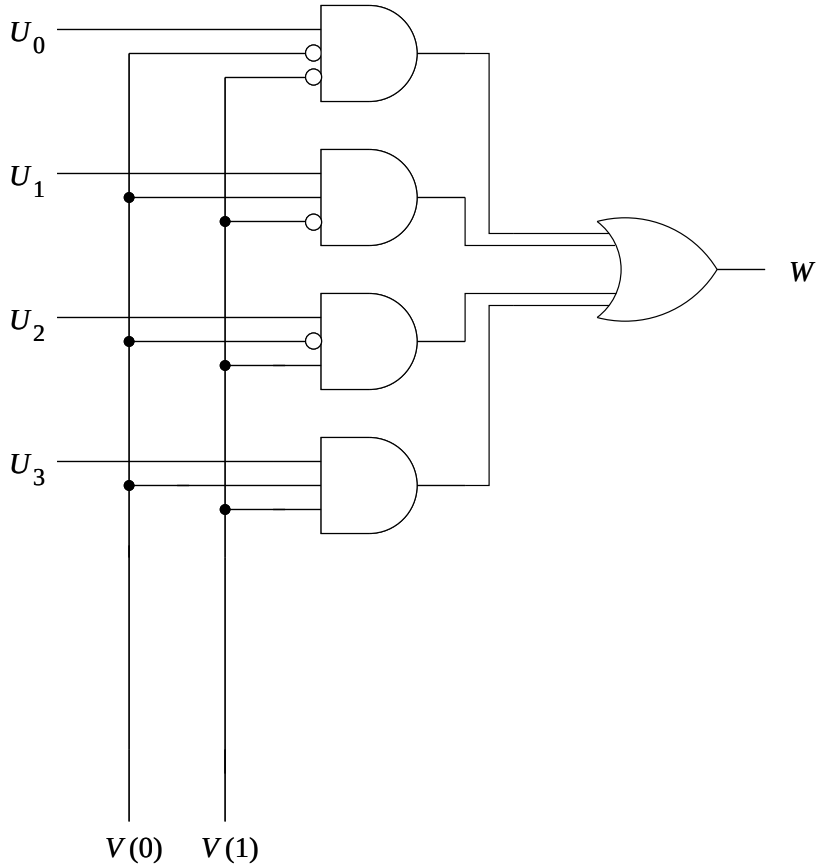


FIGURE 3.6: A 4 to 1 multiplexer circuit

This implementation is independent of the choice of memory storage elements. SRAM–based LUTs are perhaps the most common implementation; however, with minimal modifications this basic design can easily accomodate other memory cell types. Dynamic Random Access Memory (DRAM) based LUTs would require the addition of sense amplifiers and write line decoders; Read-Only Memory (ROM) such as Flash, Erasable Programmable ROMs (EPROM), or Electrically Erasable Programmable ROMs (EEPROM) would require an additional layer of polysilicon and some additional column logic [26]. LUTs composed of sequential elements are also possible, however this would require the use of a clock. This clock can be independent of any other clock in the system. Regardless of the implementation chosen, the asymptotic cost of the structure is unchanged; choices in memory technology only alter the size and access times of the LUT by a constant factor. Thus, we may consider the LUT to be a combinational element as stated in Section 3.1.

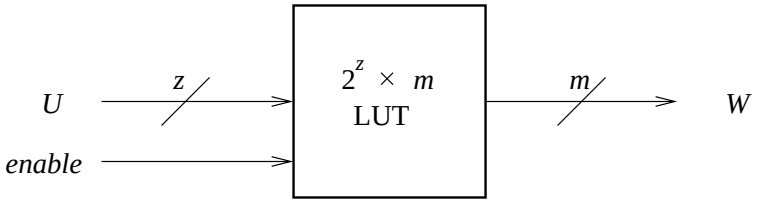


FIGURE 3.7: Look-up table block diagram

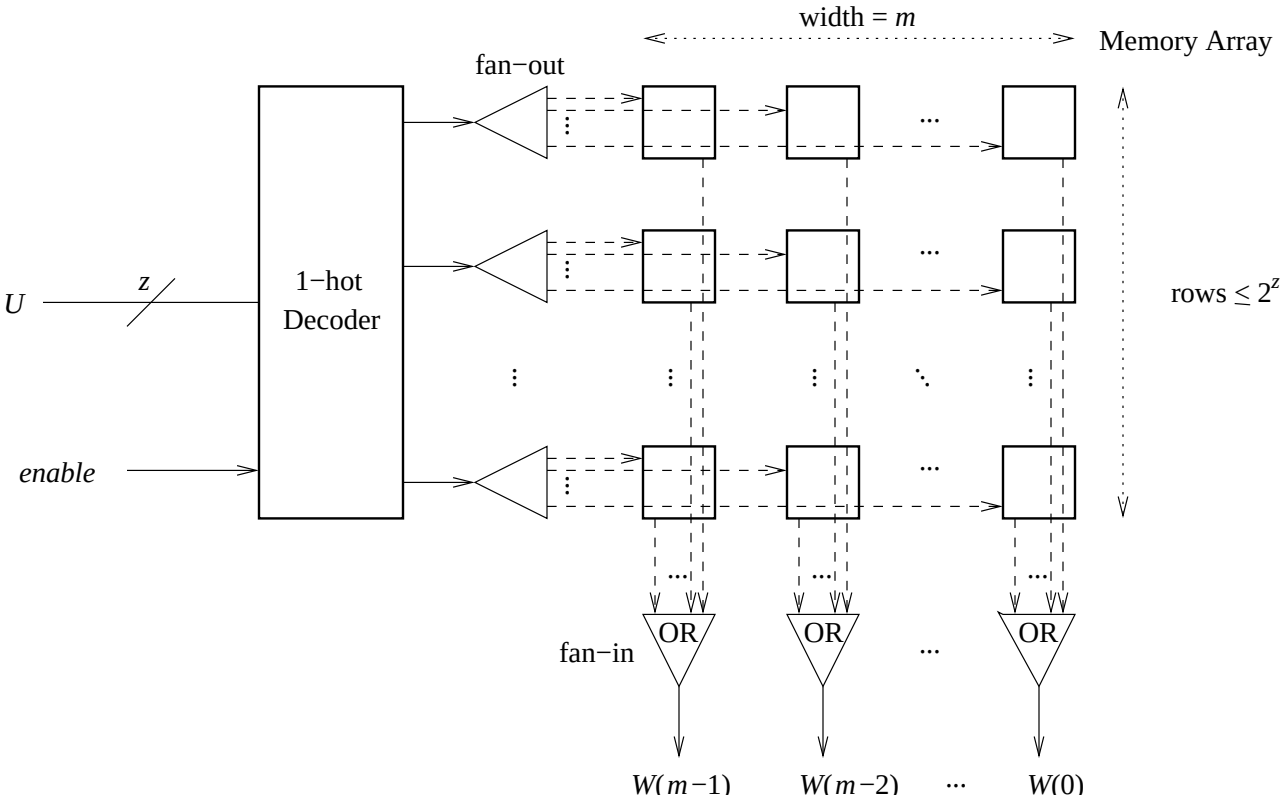


FIGURE 3.8: Look-up table implementation

Lemma 3.4 *A $2^z \times m$ look-up table can be implemented as a circuit with a gate cost of $O(2^z(z+m))$ and a delay of $O(z+\log m)$.*

Proof: Using the implementation described previously and shown in Figure 3.8, the LUT consists of two modules: the decoder and the memory array. By Lemma 3.2, a z to 2^z 1-hot decoder has a gate cost of $O(z2^z)$ and a delay of $O(z)$. Each of the outputs of the decoder selects a single row of the memory array and drives a word-line. The selection of all elements in a row of the memory array requires a fan-out of degree m that occurs at most 2^z times, which by Lemma 3.1 results in a gate cost of $O(m2^z)$ and a delay of $O(\log m)$.

As the LUT has 2^z rows each with m storage elements, the minimum gate cost for the memory array is $O(m2^z)$. When a row in the memory array is selected, each of the m-bits must be fanned-in to the output from the 2^z rows. This results in a fan-in of degree 2^z that occurs m times, resulting in a gate cost of $O(m2^z)$ and a delay of $O(z)$ (Lemma 3.1).

The overall gate cost is thus $O(z2^z + m2^z + m2^z + m2^z) = O(2^z(z+m))$ while the overall delay is $O(z + \log m + z) = O(z + \log m)$. ■

3.2.5 Shift Register

An α-position shift register of width $\frac{z}{\alpha}$ (Figure 3.9), denoted by $SR(z, \frac{z}{\alpha})$, accepts as input a

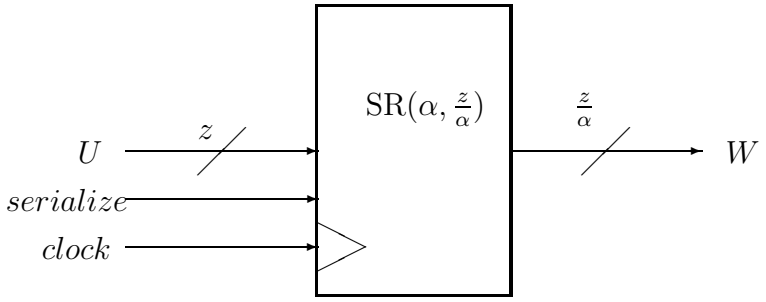


FIGURE 3.9: An α-position shift register of width $\frac{z}{\alpha}$

z-bit signal and, every clock cycle, outputs $\frac{z}{\alpha}$-bit slices of the input signal, for α clock cycles. Figure 3.10 illustrates an implementation of $SR(z, \frac{z}{\alpha})$. At each clock cycle, if the value of the input signal *serialize* is '0', the $\frac{z}{\alpha}$ register either shifts its contents to the $\frac{z}{\alpha}$ register to its left or, if it is the last register in the chain, outputs its contents (via signal $W_{\alpha-1}$). When *serialize* is asserted, a new value is stored in the α $\frac{z}{\alpha}$-bit registers. The shift register serializes the z-bit signal U based on the value of α; clearly, if $\alpha = z$, the shift register outputs each bit of the input signal sequentially. Where $serialize = 0$, a signal can be serially shifted in, $\frac{z}{\alpha}$ bits at a time and output in parallel through lines $W_{\alpha-1}, W_{\alpha-2}, \ldots, W_0$ after α cycles. From this construction, we have the following result.

Lemma 3.5 *An α-position shift register of width $\frac{z}{\alpha}$, $SR(\alpha, \frac{z}{\alpha})$, can be realized as a circuit with a gate cost of $O(z)$ and a constant delay between clock cycles.*

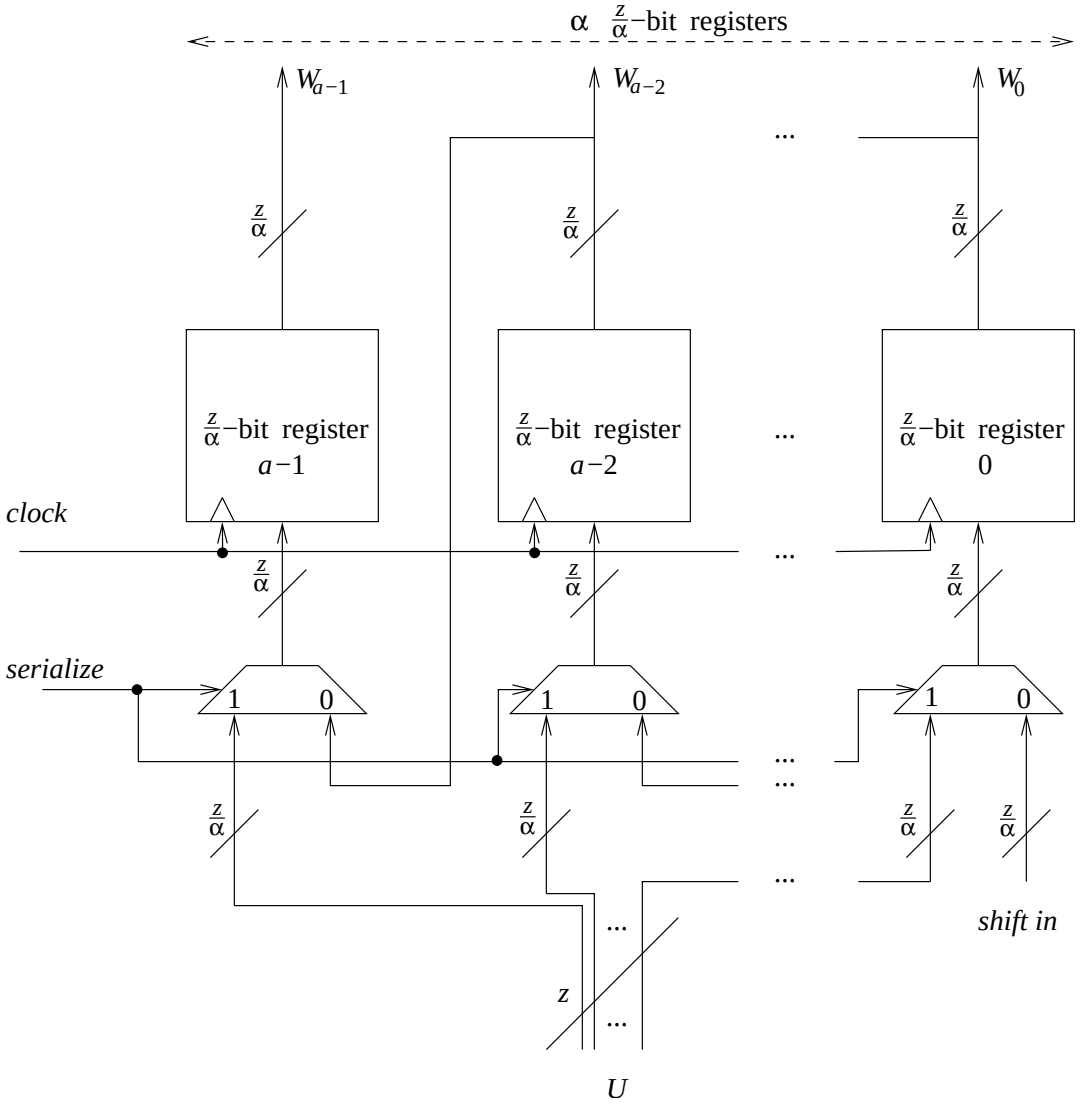


FIGURE 3.10: An implementation of a SR($\alpha, \frac{z}{\alpha}$)

Proof: The shift register consists of α banks of $\frac{z}{\alpha}$ registers, each of which is constructed from a constant number of gates and flip-flops, implying a gate cost of $O(\alpha(\frac{z}{\alpha})) = O(z)$ and a constant delay. By Lemma 3.3, each of the 2 to 1 MUXs of $\frac{z}{\alpha}$ width have a $O(\frac{z}{\alpha})$ gate cost and a constant delay. During a change of state, that is, a shifting of the contents of the registers or an input of a new signal, all propagation paths have a constant fan-out, implying a constant delay. Thus, the overall gate cost is $O(z + \frac{\alpha z}{\alpha}) = O(z)$ and the overall delay is constant between clock cycles. ■

3.2.6 Modulo-α Counter

For any $\alpha \geq 1$, a *modulo-α* (or *mod-α*) counter [6] (Figure 3.11) increments its output by

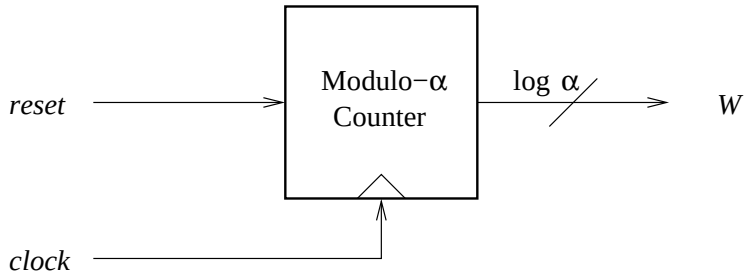
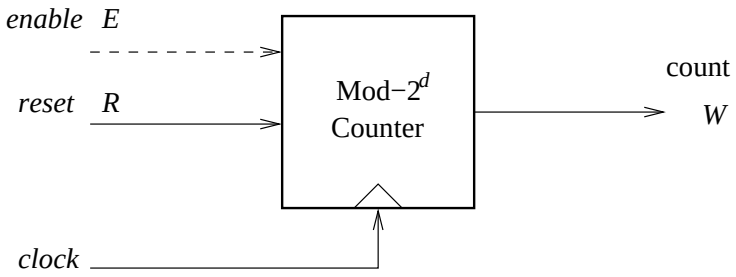


FIGURE 3.11: A modulo-α counter block diagram

'1' every clock cycle, returning to '0' after a count of $\alpha - 1$. Let $2^{d-1} < \alpha \leq 2^d$. We first construct a mod-2^d counter with synchronous reset (see Figure 3.12). Then, we use this to construct a mod-2 counter. Let $W = W(d-1)W(d-2)\ldots W(1)W(0)$ be a d-bit signal. Let k (where $0 \leq k < d$) be the smallest index such that $W(k) = 0$; $k = d-1$ implies that W has no 0s. Incrementing W amounts to complementing bits $W(k), W(k-1), \ldots, W(0)$. That is, $(W+1) \pmod{2^d} = W(d-1)W(d-2)\ldots W(i+1)\overline{W(i)}\ \overline{W(i-1)}\ \ldots \overline{W(1)}\ \overline{W(0)}$.



E	R	old W	new W
1	0	$2^d - 1$	0
1	0	$W < 2^d - 1$	$W + 1$
1	1	-	0
0	-	-	W

FIGURE 3.12: A mod-2^d counter with truth table

Let $V(i) = 1$ if and only if $i \leq k$, then the new value of W is $W \oplus V$, where W is the old value and $\oplus$ denotes a bitwise Exclusive OR.

Observe that $V(0) = 1$ and for all $0 < i < d$, $V(i) = V(i-1)$ AND $W(i-1)$. Solving this recurrence we have $V(i) = \bigwedge_{j=0}^{i-1} W(j)$ for $0 < i < d$. Factoring in the reset and enable

lines we now have

$$W = \begin{cases} W, & \text{if } E = 0 \\ 0, & \text{if } R = 1,\ E = d \\ W \oplus V, & \text{if } R = 0,\ E = 1. \end{cases}$$

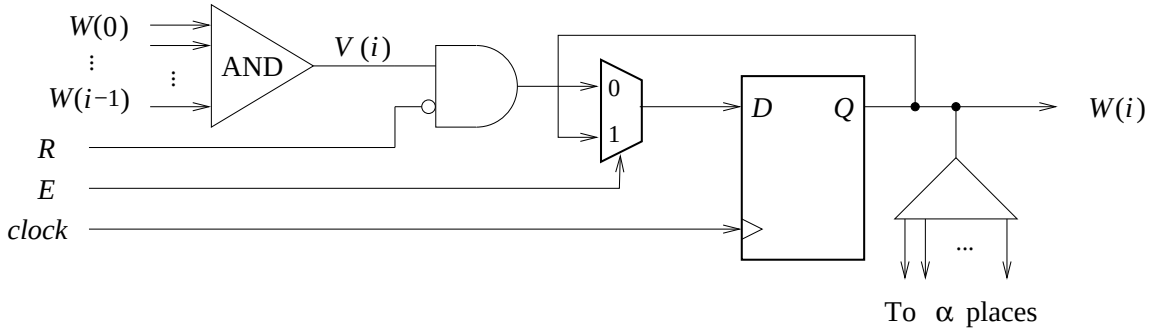


FIGURE 3.13: Circuit for bit i of a synchronous counter

Figure 3.13 shows the logic needed to compute the new value of $W(i)$. The combinational delay (between each clock tick) of a 2^d-bit counter is

$$\max_{i=0}^{d-1} \left(\underbrace{O(\log i)}_{\texttt{fan-in}} + \underbrace{O(\log d)}_{\texttt{fan-out}} \right) = O(\log d).$$

The delay to fan-out E and R to all d flip-flops is factored into the fan-out. The gate cost is $O\left(\left(\sum_{i=0}^{d-1} i\right) d\right) = O(d^2)$. This subsumes the $O(d)$ cost of fanning out E and R to all d flip-flops. Therefore we have the following result.

Lemma 3.6 *A mod-2^d counter can be realized as a circuit with a gate cost of $O(d^2)$ and a delay of $O(\log d)$.* ■

To construct a mod-α counter from such a structure requires resetting the counter when the value of W is $\alpha - 1$. This is accomplished by adding an $\alpha - 1$ detection unit that determines if the output of the counter is $\alpha - 1$ and, if so, asserts the reset input in time for the next clock tick (Figure 3.14). This computation can be performed by an AND gate with true and complementary inputs corresponding to the value of $\alpha - 1$. For example, if

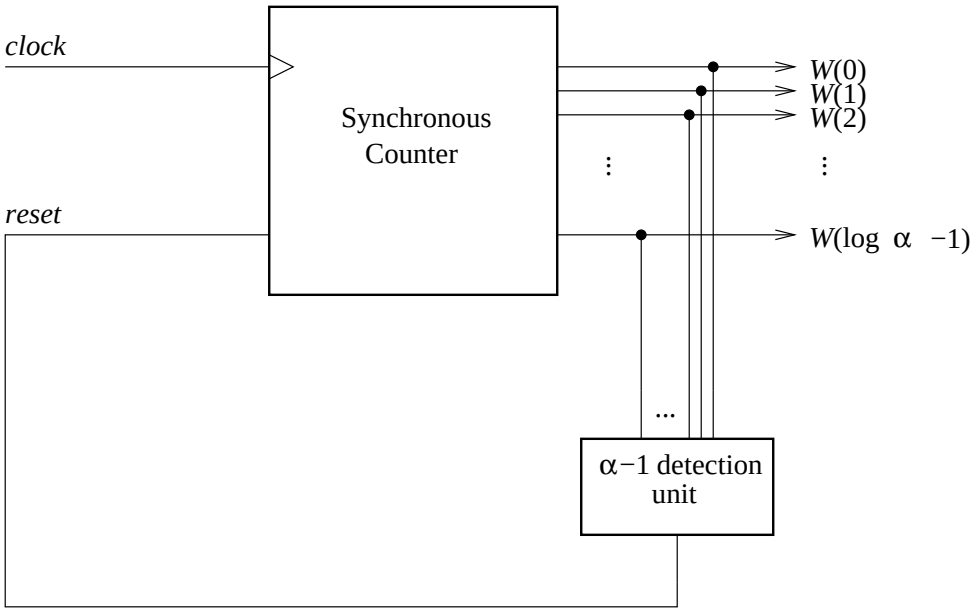


FIGURE 3.14: A modulo-α counter implementation using a synchronous counter and a mask computation

$\alpha - 1 = 5 = 101$ (a mod-6 counter), then the AND gate would complement the second least-significant input bit coming from the output of the counter. The output of the AND gate would only be a '1' if and only if the input to the AND gate was '101'. As shown in Figure 3.14, this would assert the reset input to the counter and set the counter to '0' after the clock tick. From Lemma 3.6 we have the following result.

Lemma 3.7 *A mod-α counter can be implemented as a circuit with gate cost of $O(\log^2 \alpha)$ and a delay of $O(\log \log \alpha)$.* ■

3.3 Configurable Decoders

A configurable decoder has the same basic functionality as the general decoder described in Section 3.1.1. An x to n configurable decoder accepts an x-bit input and outputs up to 2^x n-bit outputs. As mentioned in Section 3.1.1, unlike fixed decoders the output of a configurable decoder (the set $\mathcal{S}'$) is not fixed at manufacture. With reconfiguration, the n-bit outputs can be changed to a different pattern of bits, thus supplying a degree of flexibility not present in fixed decoders.

The simplest implementation of an x to n configurable decoder is a $2^x \times n$ LUT. As noted in Section 3.2.4, a $2^x \times n$ LUT takes in an x-bit input and outputs up to 2^x n-bit words, where the n-bit words are determined by the contents of its memory array. Unfortunately, this "pure LUT-based" configurable decoder is expensive. By Lemma 3.4, the gate cost of this LUT is $O(2^x(x+m))$. If this decoder was implemented on the same scale as a $\log n$ to n 1-hot decoder, then $x = \log n$. This results in a decoder that, while able to produce any n of the 2^n subsets of $\mathcal{Z}_n$, has a gate cost of $\Theta(n^2)$. On the other hand, if the pure LUT-based configurable decoder were restricted to the same asymptotic gate cost as the 1-hot decoder (that is, $\Theta(n \log n)$), it would only be able to produce $\Theta(\log n)$ subsets of $\mathcal{Z}_n$ (being at most a $\log n \times n$ LUT). Although the flexibility of the pure LUT-based configurable decoder is desirable, its cost does not scale well and an alternative is needed.

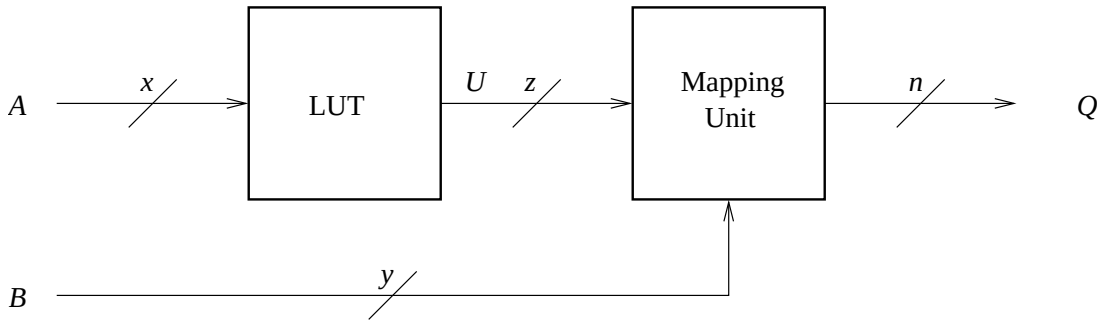


FIGURE 3.15: A configurable decoder block diagram

Our solution, which will be explained in depth in subsequent chapters, is a configurable decoder that uses a LUT with a smaller order of cost, combined with a special type of decoder called a 'Mapping Unit' (Figure 3.15). The mapping units we consider have the same order of cost as the LUT. This allows the LUT cost to be kept as small as a fixed decoder while allowing a large number of n-bit subsets to be produced within the same order of gate cost as fixed decoders. Chapters 4 and 5 will further explain the capabilities of the mapping units, while Chapter 6 will explore the capabilities of our configurable decoder.

Chapter 4

The Mapping Unit: Theory

Recall that the main problem we address is that of producing subsets of a n-set $\mathcal{Z}_n$. As we showed in Section 3.3, a pure LUT-based configurable decoder with $\log n$ input bits is capable of producing up to n of the 2^n different subsets of $\mathcal{Z}_n$, but its $\Theta(n^2)$ cost does not scale well. Thus, we seek to create a configurable decoder (as shown in Figure 3.15) that, while still using a LUT to achieve a degree of flexibility, does so with a smaller cost. We introduce in this chapter a module called the mapping unit (Figure 4.1) that serves to convert the output

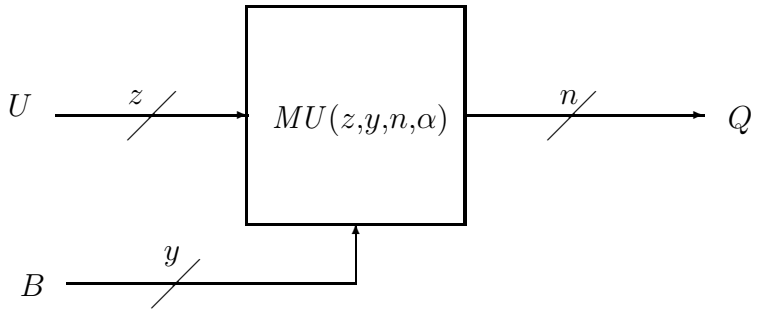


FIGURE 4.1: A mapping unit decoder block diagram

of an inexpensive LUT to the form representative of a subset of a n-set.

This chapter introduces the functionality of the mapping unit and derives some bounds on its capabilities. In Section 4.1 we provide a general view of the mapping unit, including a functional description of its operation (Section 4.1.1) and an explanation of the mapping of the z-bit inputs to the n-bit outputs (Section 4.1.2). In Section 4.2, we explore the bounds on the capabilities of the mapping unit, namely, the number of independent subsets producible by the mapping unit (Section 4.2.1) and the total number of subsets it can produce (Section 4.2.2). Later in Chapter 5 we will describe realizations of the mapping unit.

4.1 A General View of the Mapping Unit

As previously noted, in the larger context of the configurable decoder the mapping unit serves to convert the output of a $2^x \times z$ LUT to an n-bit output representing a subset of an n-set. The

mapping unit can be viewed as a type of decoder, as it takes in a relatively small number of bits (z-bits) and expands them to a larger number of bits (n-bits), where $z < n$. The mapping unit accomplishes this expansion by "multicasting" the z-bits to n places. As an example,

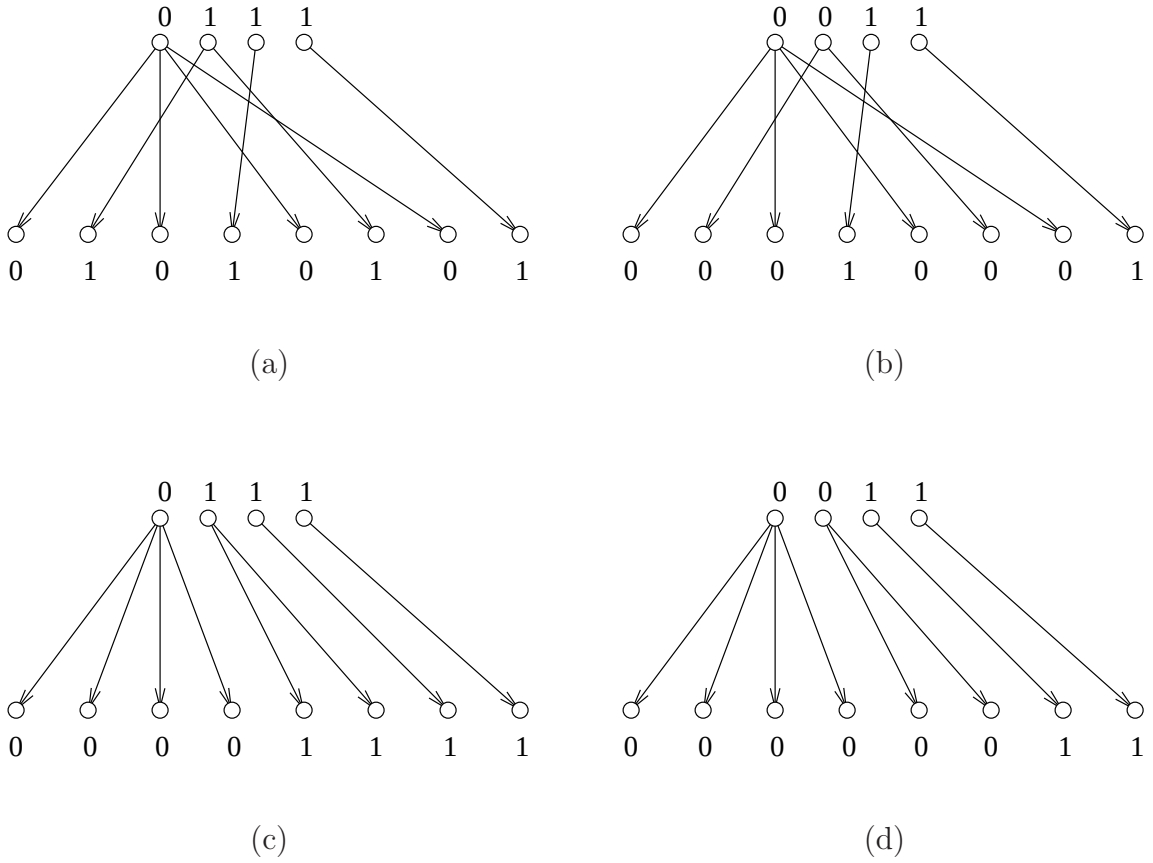


FIGURE 4.2: Multicasts of 4-bits to 8-bits, for two different multicast schemes each with two different values.

consider a multicast of four bits $a(3)a(2)a(1)a(0)$ to 8 bits $b(7)b(6)b(5)b(4)b(3)b(2)b(1)b(0)$, such that $b(0) = a(0)$, $b(1) = b(3) = b(5) = b(7) = a(3)$, $b(2) = b(6) = a(2)$ and $b(4) = a(1)$. If $a = 0111$, then $b = 01010101$ (Figure 4.2(a)). On the other hand, if $a = 0011$, then $b = 00010001$ (Figure 4.2(b)). If we change the mapping of a to b, then again, different outputs can be obtained. For example, if $b(0) = a(0)$, $b(1) = a(1)$, $b(2) = b(3) = a(2)$ and $b(4) = b(5) = b(6) = b(7) = a(3)$ then for $a = 0111$ (resp., 0011), $b = 00001111$, (resp., 00000011) (see Figures 4.2(c) and (d)).

We now characterize the multicasts described above in terms of "ordered partitions." Recall from Section 3.1.2 that a k-partition π, for any $1 \leq k \leq n$, of a n-set S is a division of S into k disjoint nonempty subsets, $S_0, S_1, \ldots, S_{k-1}$. For any $1 \leq k \leq n$, an *ordered*

k-partition $\vec{\pi}$ of an n-set S is a k-partition $\{S_0, S_1, \ldots, S_{k-1}\}$ of S with an order (from 0 to $k-1$) imposed on the blocks. We denote this ordered partition by $\vec{\pi} = \langle S_0, S_1, \ldots, S_{k-1} \rangle$. In this notation, $\langle S_0, S_1 \rangle \neq \langle S_1, S_0 \rangle$.

Consider a multicast of bits $a(z-1), a(z-2), \ldots, a(1), a(0)$ to bits $b(n-1), b(n-2), \ldots, b(1), b(0)$. An ordered z-partition $\langle S_0, S_1, \ldots, S_{z-1} \rangle$ of $\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$ represents this multicast if and only if for each $0 \leq i < z$, for all bit positions $j \in S_i$, bit $b(j)$ gets its value from $a(i)$.

For example, the multicasts of Figure 4.2(a),(b) and (c),(d) correspond to the ordered 4-partitions $\vec{\pi}_1 = \langle \{7,5,3,1\}, \{6,2\}, \{4\}, \{0\} \rangle$ and $\vec{\pi}_2 = \langle \{7,6,5,4\}, \{3,2\}, \{1\}, \{0\} \rangle$, respectively.

4.1.1 Functional Description of the Mapping Unit

Consider a mapping unit that expands a z-bit signal U to the n-bit signal Q (see Figure 4.1); note that the input B is explained later, while the parameter α is dependent on the implementation of the mapping unit and is explained in Chapter 5. As noted earlier, a multicast can be represented as an ordered partition $\vec{\pi}$ of $\mathcal{Z}_n = \{0, 1, \ldots, n-1\}$. Therefore, $\vec{\pi}$ and an instance $u \in U$ of the z-bit input to the mapping unit uniquely specify an n-bit output $q \in Q$.

For example, the ordered partition $\vec{\pi}_1 = \langle \{7,5,3,1\}, \{3,2\}, \{1\}, \{0\} \rangle$ and $u_1 = 0111$ of Figure 4.2(a) produces output $q_{1,1} = 01010101$. If u_1 is replaced by $u_2 = 0011$, then the output is $q_{1,2} = 00010001$ (see Figure 4.2(b)). Similarily, if the ordered partition is $\vec{\pi}_2 = \langle \{7,6,5,4\}, \{3,2,\}, \{1\}, \{0\} \rangle$, then the outputs corresponding to u_1 and u_2 are $q_{2,1} = 00001111$ (Figure 4.2(c)) and $q_{2,2} = 00000011$ (Figure 4.2(d)).

In general, the mapping unit uses several ordered partitions $\vec{\pi} \in \mathcal{Y}$. The y-bit input B of Figure 4.1 selects one of these ordered partitions; clearly $\mathcal{Y} \leq 2^y$. Since input (set) B of y-bit strings may be thought to be in one-to-one correspondence with $\mathcal{Y}$, we can describe the mapping unit $MU(z,y,n,\alpha)$, shown in Figure 4.1, by the following function μ.

$$\mu : \mathcal{Z}_{2^z} \times \mathcal{Z}_{2^y} \rightarrow \mathcal{Z}_{2^n}$$

Since "sets" U, B, Q are sets of z-bit, y-bit, and n-bit strings, respectively, we can also write

$$\mu : U \times B \rightarrow Q.$$

In summary, $MU(z,y,n,\alpha)$ accepts as input a z-bit string (the *source string*) and an ordered partition $\vec{\pi}$ (one among 2^y). It produces as output an n-bit string (subset of $\mathcal{Z}_n$). The source string could assume any value from $\{0,1\}^z$. The set of 2^y ordered partitions are generally fixed (usually hardwired in the mapping unit or configured into a LUT internal to the mapping unit).

4.1.2 Constructing Ordered Partitions for a Mapping Unit

Let $\mathcal{S}$ be a set of subsets of $\mathcal{Z}_n$ that we wish a mapping unit to generate. This section details a procedure for constructing a set of partitions that (along with a set of source string values) generates all elements of $\mathcal{S}$. (In the process, we may generate a set $\mathcal{S}' \supseteq \mathcal{S}$ of subsets.) Before we proceed a few definitions are needed.

A subset $S \subseteq \mathcal{Z}_n$ induces a 1- or 2-partition $\pi_S = \{S, \mathcal{Z}_n - S\}$. If $S = \emptyset$ or $S = \mathcal{Z}_n$, then π_S is the 1-partition $\{\mathcal{Z}_n\}$; otherwise, π_S is a 2-partition. Clearly, the induced partition is not unique for a given S, as $\pi_S = \pi_{\mathcal{Z}_n - S} = \{S, \mathcal{Z}_n - S\}$. When S is represented by its n-bit characteristic string, the induced partition π_S places bit positions with the same value in the same block of π_S.

Let $\mathcal{S} = \{S_0, S_1, \ldots, S_{k-1}\}$ be a set of subsets of $\mathcal{Z}_n$. For $0 \leq i < k$, let subset S_i induce partition π_i. Define the partition induced by $\mathcal{S}$ to be $\pi_{\mathcal{S}} = \pi_0 \pi_1 \ldots \pi_{k-1}$; the product of partitions is defined in Section 3.1.2.

We now illustrate these ideas with an example.

Example 4.1 Consider the sets of subsets $\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{S}_2$ of $\mathcal{Z}_8$ shown in Table 4.1, where $\mathcal{S}_i = \{S_j^i : 0 \leq j < 4\}$, for $0 \leq i < 3$. Sets $\mathcal{S}_0$ and $\mathcal{S}_1$ represent two types of reduction, and $\mathcal{S}_2$ is a set of "arbitrary" subsets of $\mathcal{Z}_8$.

For $0 \leq i < 3$ and $0 \leq j < 4$, let $\pi_{i,j}$ be the partition induced by subset S_j^i. Table 4.2 shows $\pi_{i,j}$. Let set $\mathcal{S}_i$ induce partition $\pi_i = \pi_{i,0}\pi_{i,1}\pi_{i,2}\pi_{i,3}$. Then, we have

TABLE 4.1: Sets of subsets of $\mathcal{Z}_8$ for Example 4.1.

S_j^i	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$
S_0^i	11111111	11111111	10100010
S_1^i	01010101	00001111	11111101
S_2^i	00010001	00000011	01011010
S_3^i	00000001	00000001	00000111

TABLE 4.2: Partition $\pi_{i,j}$ for subsets S_j^i of Table 4.1

S_j^i	$\pi_{0,j}$	$\pi_{1,j}$	$\pi_{2,j}$
S_0^i	{{7,6,5,4,3,2,1,0}}	{{7,6,5,4,3,2,1,0}}	{{6,4,3,2,0},{7,5,1}}
S_1^i	{{7,5,3,1},{6,4,2,0}}	{{7,6,5,4},{3,2,1,0}}	{{1},{7,6,5,4,3,2,0}}
S_2^i	{{7,6,5,3,2,1},{4,0}}	{{7,6,5,4,3,2},{1,0}}	{{7,5,2,0},{6,4,3,1}}
S_3^i	{{7,6,5,4,3,2,1},{0}}	{{7,6,5,4,3,2,1}{0}}	{{7,6,5,4,3},{2,1,0}}

$\pi_0 = \{\{7,5,3,1\},\{6,2\},\{4\},\{0\}\}$

$\pi_1 = \{\{7,6,5,4\},\{3,2\},\{1\},\{0\}\}$

$\pi_2 = \{\{7,5\},\{6,4,3\},\{1\},\{2,0\}\}$.

We now come back to a procedure that uses the given set $\mathcal{S}$ to generate a set of ordered partitions and a source string value for a mapping unit to generate $\mathcal{S}$.

1. Number the elements of $\mathcal{S}$ in some order so that $\mathcal{S} = \{S_0, S_1, \ldots, S_{k-1}\}$.

2. For each $S_i \in \mathcal{S}$, compute its induced partition π_{S_i}.

3. Starting from π_0, pick the largest integer ℓ such that $\pi_{S_0}\pi_{S_1}\ldots\pi_{S_{\ell-1}}$ has $\leq z$ blocks. Let $\pi_0 = \pi_{S_0}\pi_{S_1}\ldots\pi_{S_{\ell-1}}$.

4. Starting from π_ℓ, pick the largest integer m such that $\pi_{S_\ell}\pi_{S_{\ell+1}}\ldots\pi_{S_{\ell+m-1}}$ has $\leq z$ blocks. Let $\pi_1 = \pi_{S_\ell}\pi_{S_{\ell+1}}\ldots\pi_{S_{\ell+m-1}}$.

5. Repeat this procedure till all induced partitions π_{S_i} have been included in some π_j.

6. Convert each π_j to an ordered partition $\vec{\pi}_j$ using some arbitrary ordering of its blocks. The ordered partitions $\vec{\pi}_0, \vec{\pi}_1, \ldots$ are the ones needed in the mapping unit.

We illustrate this procedure with the following example.

Example 4.2 Let $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \mathcal{S}_2$ of Example 4.1, and let $z = 4$. Then, $\mathcal{S} = \{S_0^0, S_1^0, S_2^0, S_3^0, S_1^1, S_2^1, S_0^2, S_1^2, S_2^2, S_3^2\}$. The induced partitions corresponding to each S_j^i are in Table 4.2. Let the order enumerated above be the order in which we consider the partitions. Then using the above procedure, the partitions π_0, π_1, and π_2 are constructed as shown below.

$$\begin{aligned}
\pi_{0,0}\pi_{0,1} &= \{\{7,5,3,1\},\{6,4,2,0\}\} \\
\pi_{0,0}\pi_{0,1}\pi_{0,2} &= \{\{7,5,3,1\},\{6,2\},\{4,0\}\} \\
\pi_{0,0}\pi_{0,1}\pi_{0,2}\pi_{0,3} &= \{\{7,5,3,1\},\{6,2\},\{4\},\{0\}\} = \pi_0 \\
\\
\pi_{1,1}\pi_{1,2} &= \{\{7,6,5,4\},\{3,2\},\{1,0\}\} = \pi_1 \\
\\
\pi_{2,0}\pi_{2,1} &= \{\{7,5\},\{1\},\{6,4,3,2,0\}\} \\
\pi_{2,0}\pi_{2,1}\pi_{2,2} &= \{\{7,5\},\{2,0\},\{6,4,3\},\{1\}\} \\
\pi_{2,0}\pi_{2,1}\pi_{2,2}\pi_{2,3} &= \{\{7,5\},\{2,0\},\{6,4,3\},\{1\}\} = \pi_2
\end{aligned}$$

Order the partitions as $\vec{\pi}_0 = \langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$, $\vec{\pi}_1 = \langle\{7,6,5,4\},\ \{3,2\},\ \{1,0\}\rangle$, $\vec{\pi}_2 = \langle\{7,5\},\{2,0\},\{6,4,3\},\{1\}\rangle$. The mapping unit uses these ordered partitions with the values of the source strings shown in Table 4.3 to generate each subset in $\mathcal{S}$.

We note some interesting points from Examples 4.1 and 4.2. Note that, in general, we will call the arbitrarily chosen set $\mathcal{S}$ as the set of independent subsets and denote $|\mathcal{S}|$ by λ. The set of all subsets generatable by the ordered partitions and the source strings is the set of all subsets denoted by $\mathcal{S}'$ with $|\mathcal{S}'| = \Lambda$.

- A subset can be generated in a variety of ways, as the same z-bit source string applied to different ordered partitions can result in the same value. For example, the subset $S_0^0 = 11111111$ could be produced from any partition $\vec{\pi}_k$ with the source string 1111.

TABLE 4.3: Mapping unit values used to produce the sets in Example 4.1

S_j^i	$u \in U$	$\vec{\pi}_k$	$q \in Q$
S_0^0	1111	$\langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$	11111111
S_1^0	0111		01010101
S_2^0	0011		00010001
S_3^0	0001		00000001
S_0^1	1111		11111111
S_1^1	d011	$\langle\{7,6,5,4\},\{3,2\},\{1,0\}\rangle$	00001111
S_2^1	d001		00000011
S_3^1	0001	$\langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$	00000001
S_0^2	1010	$\langle\{7,5\},\{2,0\},\{6,4,3\},\{1\}\rangle$	10100010
S_1^2	1101		11111101
S_2^2	0011		01011010
S_3^2	0101		00000111

d indicates a don't care value.

In addition, two different source strings applied to two differently ordered partitions can result in the same value. For example, consider two orderings of partition $\vec{\pi}_0$, $\vec{\pi}_0^1 = \langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$, while $\vec{\pi}_0^2 = \langle\{0\},\{4\},\{7,5,3,1\},\{6,2\}\rangle$. Then the source string 0111 with $\vec{\pi}_0^1$ and the source string 1101 with $\vec{\pi}_0^2$ will both produce the same subset, 01010101.

- A subset not in $\mathcal{S}$ can be produced. For example, using the z-string 1010 with the ordered partition $\vec{\pi}_0$ produces the subset 10111010.

- Subsets and their induced partitions may be repeated. For example, subsets S_3^0 and S_3^1 of Example 4.1 are equal. While the procedure ignores repeated subsets and their induced partitions in generating ordered partitions, partitions corresponding to classes of algorithms or specific applications may benefit from repeating subsets.

- A partition with fewer than z blocks, such as $\vec{\pi}_1$, results in "don't care" values (d) for the bits not corresponding to any block in the partition. Thus, the subset S_1^1 with source string $d011$ may be produced from the z-string 0011 or 1011.

- In the procedure, a different sequence of considering the induced partitions $\pi_{i,j}$ can produce a different set or number of ordered partitions. For example, if the induced partitions were considered in reverse order, that is, starting with $\pi_{2,3}$, then $\pi_{2,2}$, etc., such that $\pi_0 = \pi_{2,3}\pi_{2,2}\pi_{2,1}\ldots$, the set of partitions would result in $\vec{\pi}_0 = \langle\{7,5\},\{2,0\},\{6,4,3\},\{1\}\rangle$, $\vec{\pi}_1 = \langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$, and $\vec{\pi}_2 = \langle\{7,5,3,1\},\{6,2\},\{4,0\}\rangle$.

- The conversion of an unordered partition to an ordered partition can be done in as many $z!$ ways. Some of these may be more advantageous than others. An ordering that results in common source strings used to produce the subsets of $\mathcal{S}_i$ and $\mathcal{S}_k$ (corresponding to different ordered partitions) can be useful when the mapping unit is used as part of a larger design. This is because the same z-bit source strings can be used to produce both $\mathcal{S}_i$ and $\mathcal{S}_k$. Table 4.4 demonstrates two ordered partitions for $\mathcal{S}_0$ and $\mathcal{S}_1$, resulting in two sets of source strings for each set. Note that two of the sets of source strings, one for $\mathcal{S}_0$ and one for $\mathcal{S}_1$, are the same.

4.2 Number of Subsets Produceable by $MU(z,y,n,\alpha)$

In the procedure of Section 4.1.2, it is not clear how many ordered partitions are produced, except that it is at most $|\mathcal{S}|$. In this section we answer some natural questions that arise in this context. For this discussion, assume a mapping unit $MU(z,y,n,\alpha)$ and an independent set $\mathcal{S}'$ of subsets of $\mathcal{Z}_n$.

Question 1: If the 2^y ordered partitions of $MU(z,y,n,\alpha)$ have not been fixed, how large can the independent set $\mathcal{S}$ be?

Question 2: If all 2^y ordered partitions of $MU(z,y,n,\alpha)$ have been fixed, how large can the independent set $\mathcal{S}$ be?

TABLE 4.4: Two different orderings for the partitions of sets $\mathcal{S}_0$ and $\mathcal{S}_1$ in Example 4.1 resulting in different sets of source strings used to produce the subsets in each set.

S_j^i	$\vec{\pi}$	z-bit value needed	Q
S_0^0	$\langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$	1111	11111111
S_1^0		0111	01010101
S_2^0		0011	00010001
S_3^0		0001	00000001
S_0^0	$\langle\{4\},\{6,2\},\{7,5,3,1\},\{0\}\rangle$	1111	11111111
S_1^0		1101	01010101
S_2^0		1001	00010001
S_3^0		0001	00000001
S_0^1	$\langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$	1111	11111111
S_1^1		0111	00001111
S_2^1		0011	00000011
S_3^1		0001	00000001
S_0^1	$\langle\{3,2\},\{0\},\{7,6,5,4\},\{1\}\rangle$	1111	11111111
S_1^1		1101	00001111
S_2^1		0101	00000011
S_3^1		0100	00000001

Question 3: If the 2^y ordered partitions of $MU(z,y,n,\alpha)$ have not been fixed, how large can the total set $\mathcal{S}'$ be?

We now address these questions in this section.

4.2.1 Number of Independent Subsets

We first consider the case where the ordered partitions have not been fixed.

Lemma 4.1 *For any $k \geq 1$, let $\{S_0, S_1, \ldots, S_{k-1}\}$ be a set of subsets of $\mathcal{Z}_n$. For each $0 \leq i < k$, let S_i induce a partition π_i. Then $\pi_0\pi_1 \ldots \pi_{k-1}$ has at most 2^k blocks.*

Proof: Each π_i has at most 2 blocks. Each product divides an existing block into at most two "sub-blocks". Therefore, over $k-1$ products we have at most $2 \cdot 2^{k-1} = 2^k$ blocks. ■

Remark: A more formal proof can be constructed by induction on k.

TABLE 4.5: A set of $\log z$ subsets of $\mathcal{Z}_{16}$, where the number of blocks induced by the product of the partitions of the subsets has $z = 8$ blocks.

S_i	π_i
0101010101010101	$\{\{15,13,11,9,7,5,3,1\},\{14,12,10,8,6,4,2,0\}\}$
0011001100110011	$\{\{15,14,11,10,7,6,3,2\},\{13,12,9,8,5,4,1,0\}\}$
0000111100001111	$\{\{15,14,13,12,7,6,5,4\},\{11,10,9,8,3,2,1,0\}\}$

Table 4.5 illustrates a set $\mathcal{S}$ of subsets, $|\mathcal{S}| = 3$, whose ordered partition meets the upper bound on the number of blocks given by Lemma 4.1. As shown below, the ordered partition $\vec{\pi}$ resulting from $\pi_0\pi_1\pi_2$ has $z = 8$ blocks.

$$\left.\begin{aligned} \pi_0 &= \{\{15,13,11,9,7,5,3,1\},\{14,12,10,8,6,4,2,0\}\} \\ \pi_0\pi_1 &= \{\{15,11,7,3\},\{14,10,6,2\},\{13,9,5,1\},\{12,8,4,0\}\} \\ \pi_0\pi_1\pi_2 &= \{\{15,7\},\{14,6\},\{13,5\},\{12,4\},\{11,3\},\{10,2\},\{9,1\},\{8,0\}\} \end{aligned}\right\} \vec{\pi}$$

Theorem 4.1 *Let $\mathcal{S}$ be an independent set of MU(z,y,n,α). Let $2^y\lfloor \log z \rfloor \leq 2^z$. If the partitions of MU(z,y,n,α) have not been fixed, then $|\mathcal{S}| = \lambda \geq 2^y\lfloor \log z \rfloor$.*

Proof: By Lemma 4.1, a collection of $\lfloor \log z \rfloor$ subsets induces a partition with at most $2^{\lfloor \log z \rfloor} \leq z$ blocks. Thus, as many as $2^y\lfloor \log z \rfloor$ arbitrarily selected subsets can be included in $\mathcal{S}$, using 2^y partitions, each with $\leq z$ blocks. Also, since $2^y\lfloor \log z \rfloor \leq 2^z$, there is no constraint on whether an appropriate source string is available for generation of a given subset. ■

Now we address Question 2, namely, the number of independent subsets that can be generated if the partitions are fixed.

Theorem 4.2 *Let $\mathcal{S}'$ be an independent set of subsets of $\mathcal{Z}_n$. For any z, y, n such that $z + y \leq n$, and for a mapping unit MU(z,y,n,α) with fixed partitions, $|\mathcal{S}| = \lambda = 0$.*

Proof: Since $z + y \leq n$, $2^z 2^y < 2^n$. Therefore there is at least one subset belonging to $\wp(\mathcal{Z}_n)$ that cannot be generated from the 2^z possible source strings and 2^y partitions that are inputs to the mapping unit. ■

Note that the number of independent subsets of $\mathcal{Z}_n$ produced by a mapping unit does not include what is possible under reconfiguration; however, the above theorem establishes the usefulness of mapping units with configurable partitions, explored in Chapter 5. This leads us to the following definition.

Definition 4.1 *A mapping unit MU(z,y,n,α) is universal if and only if it can, under reconfiguration, produce any set of* $2^y \log z$ *arbitrarily selected subset of* $\mathcal{Z}_n$.

4.2.2 Total Number of Subsets

We now address the question of how many subsets (not necessarily independent) can be generated by $MU(z,y,n,\alpha)$ (using partitions with $\leq z$ blocks) for any source string u and any ordered partition $\vec{\pi}_i$. In general, one could construct a mapping unit that produces the same subset, regardless of the input. So instead of addressing the question of the minimum number of sets that $MU(z,y,n,\alpha)$ can produce, we derive a lower bound on the maximum number of distinct subsets $MU(z,y,n,\alpha)$ can produce.

Recall from Section 4.1.1 that the output of $MU(z,y,n,\alpha)$ is given by the function $\mu(u, \vec{\pi})$. The following Lemma describes the output of the mapping unit for any two source strings and a single ordered partition.

Lemma 4.2 *For any ordered z-partition* $\vec{\pi}$ *and any pair of distinct source strings* u_1, u_2, *the outputs* $\mu(u_1, \vec{\pi}) \neq \mu(u_2, \vec{\pi})$.

Proof: Since $u_1 \neq u_2$, there exists an i $(0 \leq i < z)$ such that bit $u_1(i) \neq u_2(i)$. Since $\vec{\pi}$ is a z-partition, every bit of the source string is used by $\vec{\pi}$. If $\vec{\pi} = \{B_0, B_1, \ldots, B_{z-1}\}$ then all bits of B_i are assigned to $u_1(i)$ in $\mu(u_1, \vec{\pi})$, which differs from the value(s) assigned in $\mu(u_2, \vec{\pi})$. ■

We now extend this idea to a set of Y ordered partitions, where $Y \leq 2^{\left\lceil \frac{n}{z-1} \right\rceil - 1}$. Since the quantity $\left\lceil \frac{n}{z-1} \right\rceil - 1$ will be used extensively in this section, we let $\chi = \left\lceil \frac{n}{z-1} \right\rceil - 1$.

Divide the n-bits of $MU(z,y,n,\alpha)$ into $\chi+1$ buckets of at most $(z-1)$ contiguous bits. If any bucket has fewer than $z-1$ bits, then make that the rightmost bucket. Specifically, for $1 \leq i \leq \chi$, bucket B_i contains indices α_i to β_i, where

$$\alpha_i = n - (\chi - i + 1)(z-1) \text{ and } \beta_i = n - (\chi - i)(z-1) - 1.$$

Bucket B_0 (the rightmost bucket) ranges from bit $\alpha_0 = 0$ to bit $\beta_0 = n - \chi(z-1) - 1$.

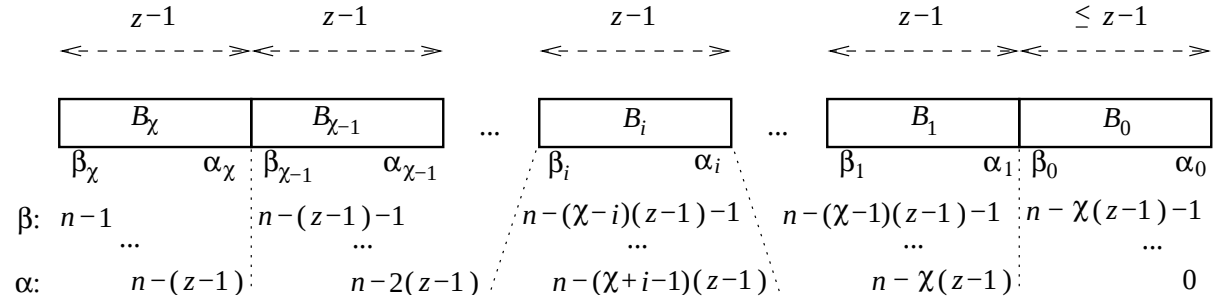


FIGURE 4.3: Division of an n-bit quantity into $\chi+1$ buckets of at most $(z-1)$ contiguous bits

Figure 4.3 illustrates this. Thus, for $1 \leq i \leq \chi$, B_i has $\beta_i - \alpha_i + 1 = z-1$ indices and B_0 has $\beta_0 - \alpha_0 + 1 = \beta_0 + 1 = n - \chi(z-1) = n - \left(\left\lceil \frac{n}{z-1} \right\rceil - 1\right)(z-1) = n + (z-1) - \left\lceil \frac{n}{z-1} \right\rceil (z-1)$.

We now specify $\vec{\pi}$ by assigning each bit of each bucket B_i to a bit of a source string u. Let $m = m(\chi-1)m(\chi-2)\ldots m(1)m(0)$ be a χ-bit number. Writing m as a $\chi+1 = \left(\left\lceil \frac{n}{z-1} \right\rceil\right)$-bit number we have

$$m = m(\chi)m(\chi-1)\ldots m(1)m(0),$$

where $m(\chi) = 0$. The above binary representation of m induces an ordered partition $\vec{\pi}$ as follows.

Recall that for any $0 \leq i \leq \chi$, the bits of bucket B_i are $\beta_i, \beta_i - 1, \ldots, \alpha_i + 1, \alpha_i$. If $m(i) = 0$, then multicast $u(z-1)$ (the most significant bit of the source string u) to all bits $\beta_i, \beta_i - 1, \ldots, \alpha_i + 1, \alpha_i$ of B_i. If $m(i) = 1$, then assign $u(z-2)$ to β_i, $u(z-3)$ to $\beta_i - 1$, $u(z-4)$ to $\beta_i - 2$ and so on; if $i = 0$ and B_0 has fewer than $z-1$ bits, the last few bits of u are not used. Figure 4.4 shows the manner in which source string bits are assigned to bucket indices. It should be clear that two χ-bit numbers m, m' will induce two different

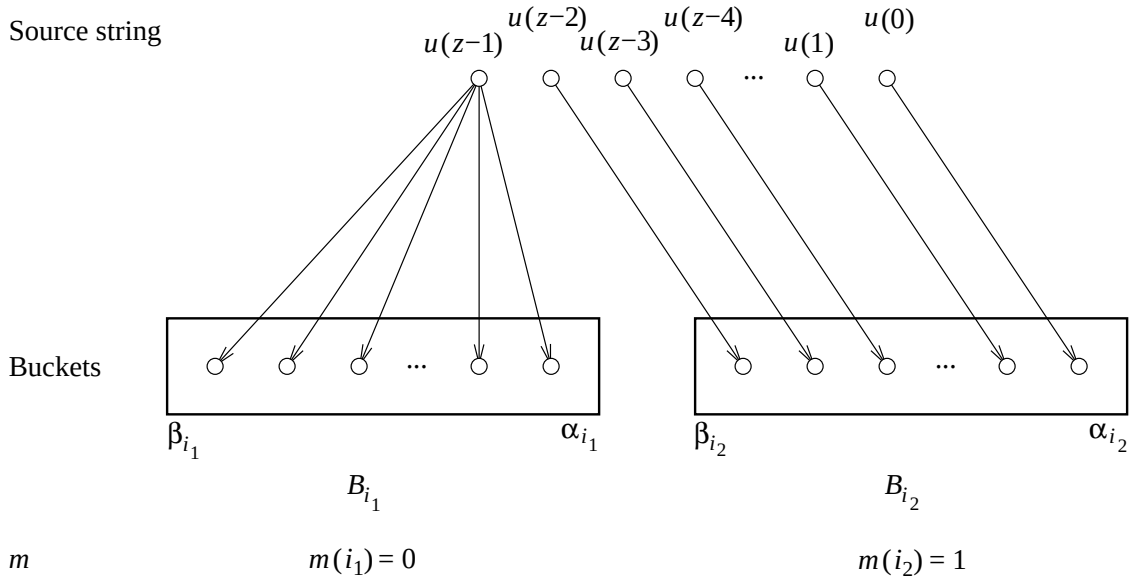


FIGURE 4.4: Assignment of source string bits to bucket indices

ordered partitions $\vec{\pi}$, $\vec{\pi}'$ if and only if $m \neq m'$ (as the bit where m and m' differs will cause a different multicast in the two cases). Since we have 2^χ distinct values for m, 2^χ distinct ordered partitions may be created as described above.

Lemma 4.3 *Let $\vec{\pi}_1$, $\vec{\pi}_2$ be any two ordered partitions created from χ-bit integers m_1, m_2 (as described above). Then for any (not necessarily distinct) source strings u_1, u_2, where $0 < u_1, u_2 < 2^z - 1$, we have $\mu(u_1, \vec{\pi}_1) \neq \mu(u_2, \vec{\pi}_2)$.*

Proof: Since $\vec{\pi}_1 \neq \vec{\pi}_2$, we have $m_1 \neq m_2$. Let $m_1(i) \neq m_2(i)$ for some $0 \leq i < \chi$. Without loss of generality, let $m_1(i) = 0$ and $m_2(i) = 1$. Figure 4.5 shows how u is mapped to bucket B_i of ordered partitions $\vec{\pi}_1$ and $\vec{\pi}_2$. We now consider two cases.

Case 1: There is some $u_2(\ell)$ (where $0 \leq \ell < z-1$) that is different from $u_1(z-1)$. Without loss of generality, let $u_1(z-1) = 0$ and $u_2(\ell) = 1$. Then bucket B_i of $\mu(u_1, \vec{\pi}_1)$ has all 0's whereas the bucket of $\mu(u_2, \vec{\pi}_2)$ has a 1 in the position corresponding to $u_2(\ell)$.

Case 2: $u_1(z-1) = u_2(z-2) = u_2(z-3) = \ldots = u_2(1) = u_2(0) \neq u_2(z-1)$. Without loss of generality, let $u_1(z-1) = 0$ and $u_2(z-1) = 1$. Consider bucket B_χ. Since $m_1(\chi) = m_2(\chi) = 0$, bucket B_χ of $\mu(u_1, \vec{\pi}_1)$ has all 0's whereas the same bucket for $\mu(u_2, \vec{\pi}_2)$ has all 1's.

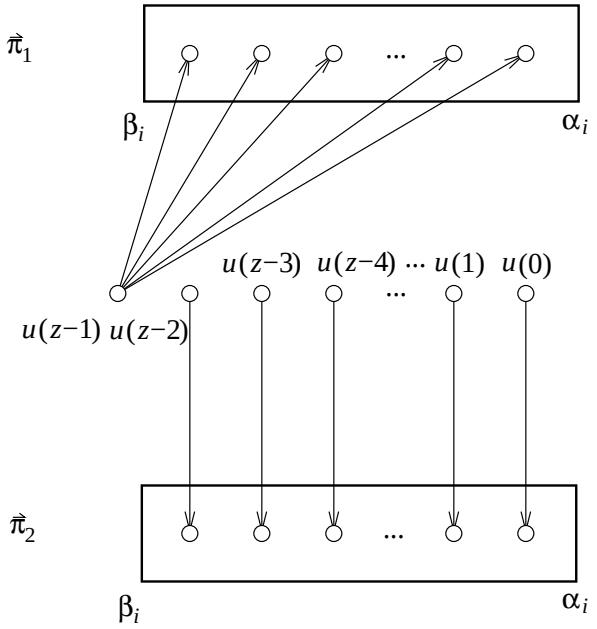


FIGURE 4.5: Mapping of a source string to bucket B_i under two different ordered partitions $\vec{\pi}_1$, $\vec{\pi}_2$

In either case, $\mu(u_1, \vec{\pi}_2) \neq \mu(u_2, \vec{\pi}_2)$. ■

We now put Lemmas 4.2 and 4.3 together to derive the main result for Question (3) (from page 46).

Theorem 4.3 *For integers $n, z \geq 2$ and $\chi = \left\lceil \frac{n}{z-1} \right\rceil - 1$, there exists a mapping unit that accepts C values from the set $\{u : 0 < u < 2^z - 1\}$ as source strings and one of $Y \leq 2^\chi$ ordered partitions that produces CY distinct subsets.*

Proof: Construct Y partitions as shown earlier from a set of χ-bit numbers. Consider source string(s) u_1, u_2 and ordered partitions $\vec{\pi}_1$, $\vec{\pi}_2$. If $u_1 = u_2$ and $\vec{\pi}_1 = \vec{\pi}_2$ then clearly $\mu(u_1, \vec{\pi}_1) = \mu(u_2, \vec{\pi}_2)$.

If $u_1 \neq u_2$ then by Lemmas 4.2 and 4.3 $\mu(u_1, \vec{\pi}_1) \neq \mu(u_2, \vec{\pi}_2)$. If $u_1 = u_2$, but $\vec{\pi}_1 \neq \vec{\pi}_2$, then again by Lemma 4.3 $\mu(u_1, \vec{\pi}_1) \neq \mu(u_2, \vec{\pi}_2)$.

Thus, under the conditions laid out in the theorem, if the ordered pairs $\langle u_1, \vec{\pi}_1 \rangle$ and $\langle u_2, \vec{\pi}_2 \rangle$ (or inputs to the mapping unit) are distinct, then so are the outputs of the mapping unit. So the number of distinct outputs equals the number of distinct inputs, which is CY. ■

Remark: In general, C can be as large as $2^z - 2$ and Y can be as large as 2^y provided $y < \left\lceil \frac{n}{z-1} \right\rceil$. So in this case, $2^y(2^z - 2)$ subsets can be produced.

The above theorem shows the existence of a set of Y ordered partitions for which a large number of subsets can be produced. Actually, this is a "class" of sets of Y ordered partitions. Clearly we need not set $m(z-1)$ to 0; any bit of m can be fixed at 0 or 1. Additionally, the buckets need not be as stated, as any fixed permutation of the n bits into buckets of "equal size" would be equivalent. The fixing of one bucket (B_χ in our construction) was needed to avoid partitions based on integers m and m', where the binary representations of m and m' are complements of each other. Including both m, m' will make the proof of Lemma 4.3 incomplete. However, the same effect of avoiding "complementary" partitions can be obtained by restricting source strings to be non-complementary. Other more fine-tuned observations can be made for specific cases. Thus, while the set of CY subsets that can be created as described is somewhat more restricted than those in Theorem 4.1 (where the subsets are independent), the restriction is not nearly as severe as Theorem 4.3 seems to imply.

In the next chapter, realizations of the mapping unit are presented.

Chapter 5

The Mapping Unit: Realizations

In the previous chapter, a mapping unit $MU(z,y,n,\alpha)$ (Figure 5.1) was described as a decoder that accepts as input a source string of z-bits (given by a $u \in U$) and an ordered partition $\vec{\pi}$ of an n-set with at most z blocks (selected by a $b \in B$). Using the operation μ (described in Section 4.1.1) the mapping unit $MU(z,y,n,\alpha)$ produces an n-bit string. In this chapter, we present several realizations of the mapping unit and detail their operation.

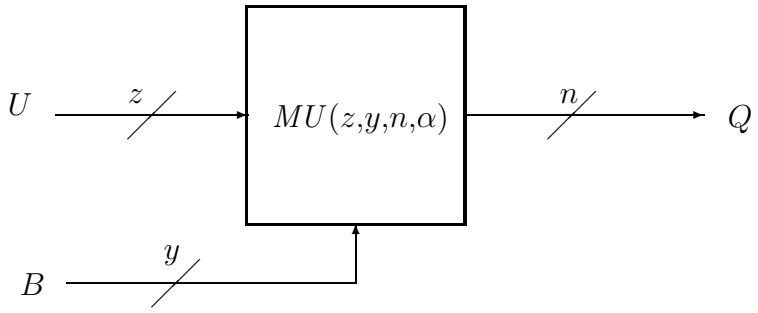


FIGURE 5.1: Block diagram of a mapping unit $MU(z,y,n,\alpha)$

We first provide a classification of the mapping units in this chapter (see Figure 5.2). A

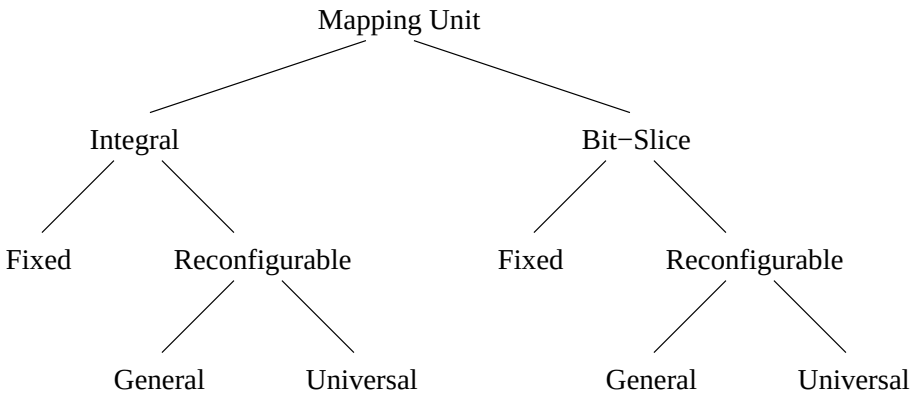


FIGURE 5.2: Classification of mapping unit realizations

mapping unit $MU(z,y,n,\alpha)$ can be integral (by default) or bit-slice. An integral mapping unit generates all n output bits simultaneously and (for reasons explained below) has $\alpha = 1$. A bit-slice mapping unit, on the other hand, generates the n output bits in α rounds; i.e.,

$\frac{n}{\alpha}$-bits at a time. One could view the integral mapping unit as a bit-slice mapping unit with $\alpha = 1$. The default for a mapping unit is the integral attribute. Another way to categorize mapping units (both integral and bit-slice) is in terms of whether they are fixed or reconfigurable (that is, based on whether they can be configured off-line to alter their behavior). Reconfigurable mapping units can be general (default) or universal. In informal terms, a universal mapping unit can produce any subset. It was established in Theorem 4.2 that fixed mapping units cannot be universal. Later in this chapter we show that there exists a universal reconfigurable mapping unit. However, it is not known whether or not all reconfigurable mapping units are universal. The "general" attribute should be interpreted as "not known if universal."

We begin our discussion of mapping unit realizations with the simplest class, fixed mapping units, explored in Section 5.1. We then describe a more flexible class, reconfigurable mapping units, in Section 5.2. Finally we conclude this chapter with bit-slice mapping units that use some of the results of Sections 5.1 and 5.2. The various mapping unit implementations will be used in Chapter 6 in the construction of a configurable decoder.

5.1 Fixed Mapping Units

The basic strategy of the fixed mapping unit (FMU) is to hardwire connections according to each ordered partition (multicast), superimposing these connections through a set of multiplexers, and using the y-bit signal b to select the multiplexer output. For example, let $z = 4$, $y = 1$, and $n = 8$. Then there are $2^y = 2$ ordered partitions mapping the 4 source string bits to the 8 output bits. Let the mappings be as shown in Figure 4.2(a),(b) and (c),(d) (page 40), which produce the sets of subsets $\mathcal{S}_0$ and $\mathcal{S}_1$ from Table 4.1 (see Example 4.1, page 43). The resulting FMU is shown in Figure 5.3. Notice that if input signal $B = 0$, then $U(0)$ is connected to $Q(0)$, $U(1)$ is connected to $Q(4)$, $U(2)$ to $Q(2)$ and $Q(6)$, and $U(3)$ to $Q(1)$, $Q(3)$, $Q(5)$, and $Q(7)$. This matches the connections shown in Figure 4.2(a) and (b). Similarily, verify that where $B = 1$, the resulting connections match those of Figure 4.2(c) and (d).

The general structure of an FMU is shown in Figure 5.4. How signal U is fanned out

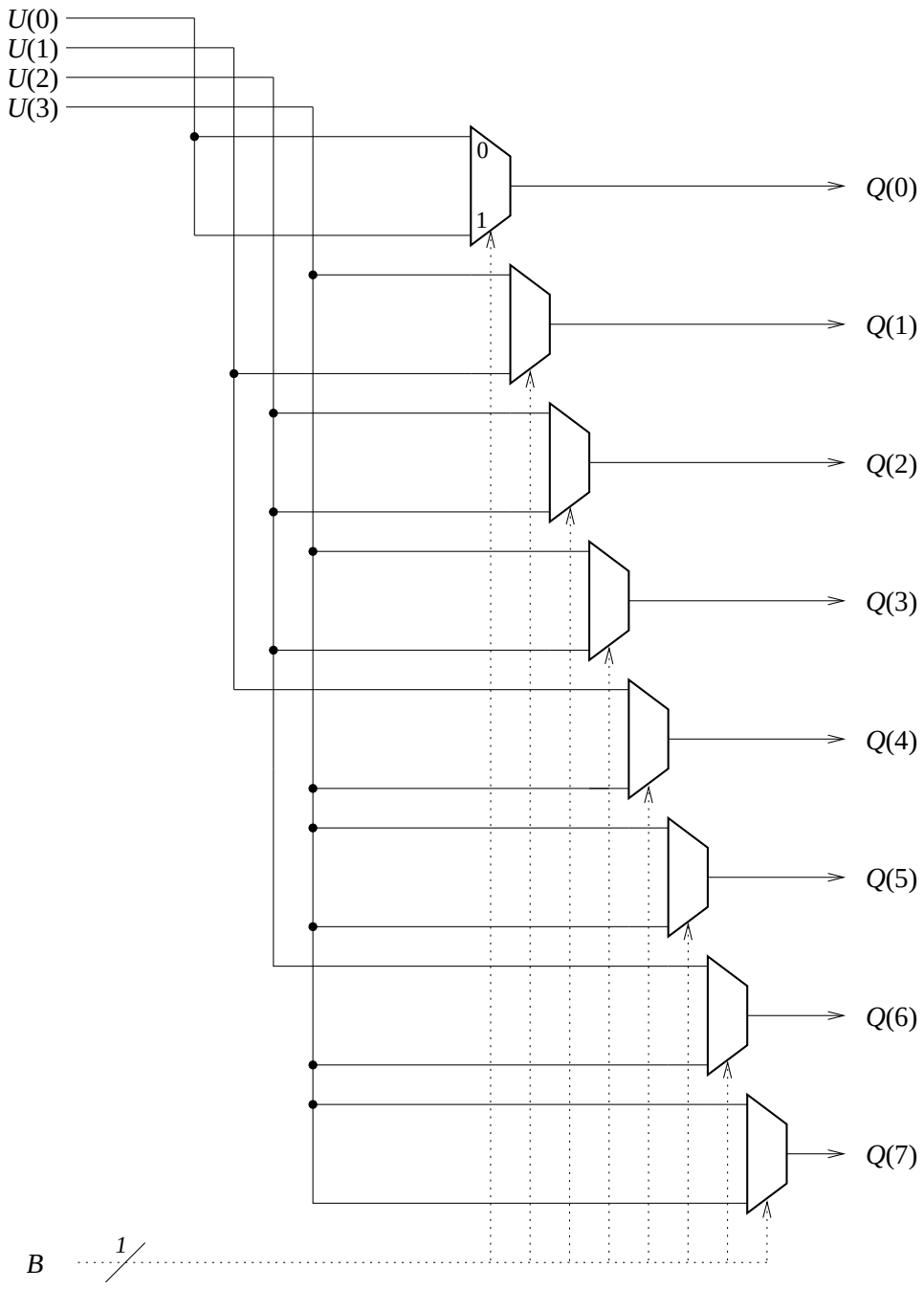


FIGURE 5.3: A fixed mapping unit $MU(4,2,8,1)$ that produces $\mathcal{S}_0$ and $\mathcal{S}_1$ in Table 4.1

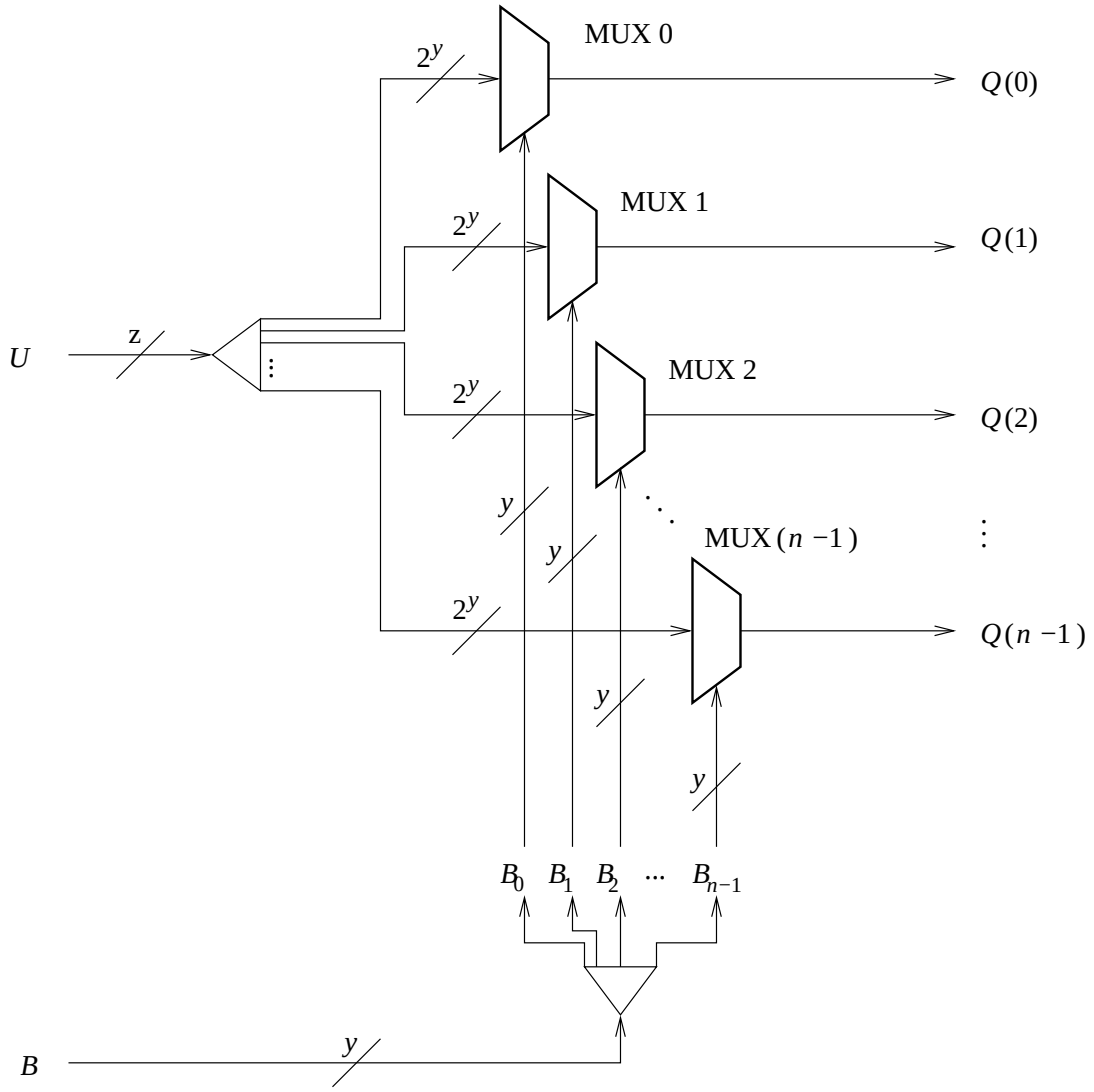


FIGURE 5.4: General structure of a fixed mapping unit; signals $B_0, B_1, \ldots, B_{n-1}$ are discussed later

to the various multiplexers depends on the 2^y ordered partitions used. In general, each multiplexer receives 2^y bits, so the z bits of U are collectively fanned out to $n2^y$ places. We begin the construction of the cost of a fixed mapping unit with the following theorem.

Theorem 5.1 *A fixed mapping unit $MU(z,y,n,\alpha)$ can be realized as a circuit with a gate cost of $O(ny2^y)$ and a delay of $O(y + \log n)$.*

Proof: The cost of the FMU is the summation of the costs of its internal building blocks. From Figure 5.4, the building blocks consist of n multiplexers and the fan-out of the signals U and B. By Lemma 3.3, the n multiplexers, each with 2^y inputs, can be realized as circuits with an overall gate cost of $O(ny2^y)$ and a delay of $O(y)$.

The fan-out of signal B has degree n and width y. By Lemma 3.1, it has a gate cost of $O(ny)$ and a delay of $O(\log n)$ (Lemma 3.1). As observed earlier, the z-bit signal U is fanned out to $n2^y$ multiplexer inputs. If bit i $(0 \leq i < z)$ of U is fanned out to n_i places, then its delay is $\log n_i = O(\log n)$ and its cost is $O(n_i)$. The total delay is $O(y + \log n)$ and the total cost is $O\left(\sum_{i=0}^{z-1} n_i\right) = O(n2^y)$.

The overall delay and cost of the FMU is thus $O(y + \log n + y + \log n) = O(y + \log n)$ and $O(ny2^y + ny + n2^y) = O(ny2^y)$. ■

In general, there is no relationships between the values z and y. Figure 5.3 illustrates a case where $z > 2^y$. Figure 5.5 illustrates a case where $z = 2^y$. Note that if $z = 2^y$, then the number of inputs of each MUX is z (as shown in Figure 5.5), implying a gate cost of $O(z \log z)$ for each of the n multiplexers.

As an example of these fixed mapping units, consider the sets shown in Table 5.1, where sets $\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{S}_3$ are the sets of subsets from Example 4.1, while set $\mathcal{S}_2$ is a set of subsets whose ordered partitions satisfy the constraints imposed by the construction for Theorem 4.3. We have used an intelligent ordering (see Table 4.4, page 47) of the partitions of $\mathcal{S}_0$ and $\mathcal{S}_1$, as a result of which the z-bit source strings of U required to produce the subsets of $\mathcal{S}_0$ and $\mathcal{S}_1$ are the same. This reduces the number of rows needed to store the values in a LUT preceding the mapping unit (see Figure 3.15 and Chapter 6) in the configurable decoder. Note that since $\mathcal{S}_3$ contains three blocks in its partition, the most significant bit of the z-bit source strings that produce the subsets of $\mathcal{S}_3$ have a "don't care" value d.

Figure 5.3 illustrates an implementation of the FMU that can produce the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ of Table 5.1, as $2^y = 2$. The FMU of Figure 5.5 can produce all subsets in Table 5.1, as $z = 2^y = 4$. Note that in each implementation, the first input to a MUX corresponds to the ordered partition to produce $\mathcal{S}_0$, the second input to a MUX corresponds to the ordered partition to produce $\mathcal{S}_1$, etc. Thus, to produce S_2^3 in the FMU, input signal U would have a value of 0101 and input signal B would have a value of 11.

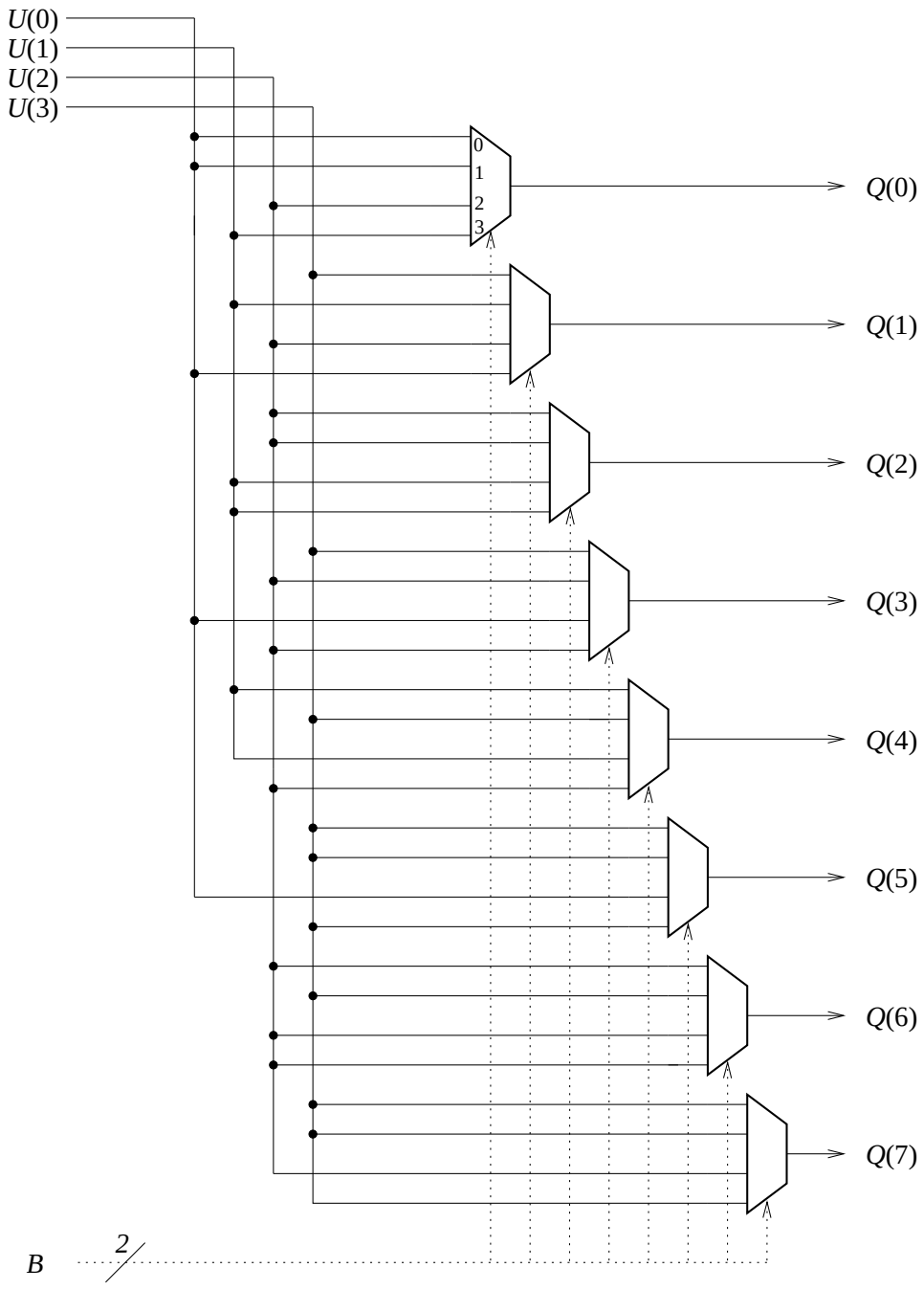


FIGURE 5.5: A fixed mapping unit $MU(4,4,8,1)$ that produces all subsets in Table 5.1

TABLE 5.1: Sets of n-subsets ($n = 8$, $z = 4$) used for fixed mapping units in Figures 5.3 and 5.5

S_j^i	$q \in Q$	$\vec{\pi}_i \in \mathcal{Y}$	$u \in U$
S_0^0	11111111	$\langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$	1111
S_1^0	01010101		0111
S_2^0	00010001		0011
S_3^0	00000001		0001
S_0^1	11111111	$\langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$	1111
S_1^1	00001111		0111
S_2^1	00000011		0011
S_3^1	00000001		0001
S_0^2	00 01 01 00	$\langle\{7,6,1,0\},\{4,2\},\{5,3\}\rangle$	d010
S_1^2	00 10 10 00		d001
S_2^2	00 11 11 00		d011
S_3^2	11 00 00 11		d100
S_4^2	11 01 01 11		d110
S_5^2	11 10 10 11		d101
S_0^3	10100010	$\langle\{7,5\},\{6,4,3\},\{2,0\},\{1\}\rangle$	1001
S_1^3	11111101		0110
S_2^3	01011010		0101
S_3^3	01011101		1110

d indicates a don't care value.

5.2 Reconfigurable Mapping Units

By Theorem 4.2 (page 48), when the ordered partitions of a mapping unit are fixed, certain subsets cannot be produced. Here, we seek to provide a means to change the ordered partitions off-line in a "reconfigurable mapping unit."

A reconfigurable mapping unit (RMU) (Figure 5.6) allows the set $\mathcal{Y}$ of ordered partitions

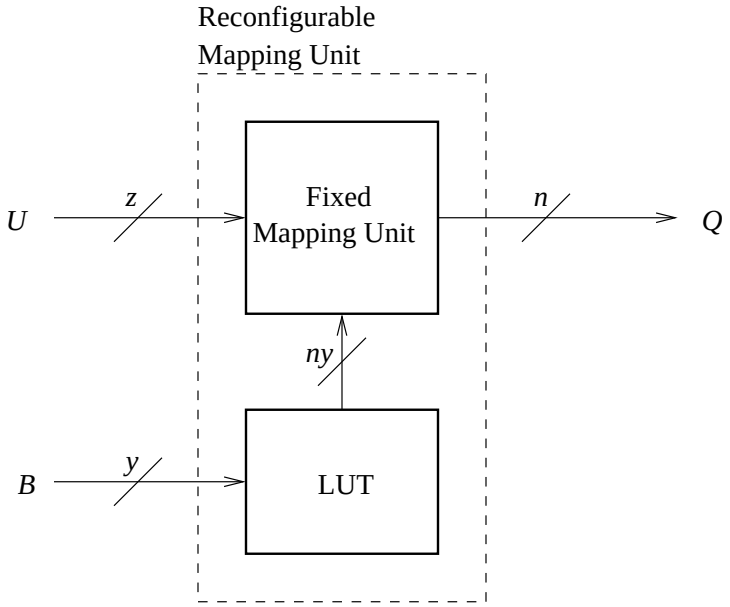


FIGURE 5.6: A reconfigurable mapping unit $MU(z,y,n,\alpha)$

to be changed off-line. While $\mathcal{Y}$ may not be totally arbitrary, a degree of flexibility is allowed that is not seen in the fixed mapping units of Section 5.1.

The flexibility of the RMU comes from a $2^y \times ny$ LUT (that is, a LUT with 2^y rows and a word size of ny) called a "configuration LUT." The output of the configuration LUT generates the FMU signal shown as $B_0, B_1, \ldots, B_{n-1}$ in Figure 5.4. The main advantage of the RMU is that it can control the signals $B_0, B_1, \ldots, B_{n-1}$ at will. In contrast, $B_i = B$, for each $0 \leq i < n$ in the FMU. We first derive the cost and delay of an RMU.

Theorem 5.2 *A reconfigurable mapping unit $MU(z,y,n,\alpha)$ can be realized as a circuit with a gate cost of $O(ny2^y)$ and a delay of $O(y + \log n)$.*

Proof: By Theorem 5.1, the FMU has a gate cost of $O(ny2^y)$ and a delay of $O(y + \log n)$. This gate cost would be unchanged even if the fan-out of B is ignored. By Lemma 3.4, a

$2^y \times ny$ LUT has a gate cost of $O(2^y(y+ny)) = O(ny2^y)$ and a delay of $O(y + \log(ny)) = O(y + \log n)$. The overall gate cost of the reconfigurable mapping unit is thus $O(ny2^y)$ while the overall delay is $O(y + \log n)$. ■

As an example of the functionality of an RMU, consider the FMU with $z = 2^y$ of Figure 5.3, which implemented all four sets of subsets in Table 5.1. If an RMU was used to implement the same set of subsets using the same wiring of the signal U to the n multiplexers, then Table 5.2 shows the contents of the configuration LUT of this RMU. Note that the LUT

TABLE 5.2: Configuration LUT words to produce the subsets from Table 5.1

Address $b \in B$	$n \log z$-bit word in LUT	Set $\mathcal{S}_i$
00	00 00 00 00 00 00 00 00	$\mathcal{S}_0$
01	01 01 01 01 01 01 01 01	$\mathcal{S}_1$
10	10 10 10 10 10 10 10 10	$\mathcal{S}_2$
11	11 11 11 11 11 11 11 11	$\mathcal{S}_3$

contents specify an ordered partition corresponding to a set of subsets, and not the subset itself. For example, when $b = 00$ the LUT word is 00 00 00 00 00 00 00 00 corresponding to the ordered partition $\vec{\pi}_0$ for set $\mathcal{S}_0$ (see Tables 5.1 and 5.2). Then with $u = 0111$, we have $\mu(u, \vec{\pi}_0) = 01010101$. Similarily, with $u = 0011$, we have $\mu(u, \vec{\pi}_0) = 00010001$. Thus, in this illustration $b = 00$ corresponds only to the ordered partition $\vec{\pi}_0$ for $\mathcal{S}_0$.

There are two important properties of the reconfigurable mapping unit that can be seen from this example. The first is that from a perspective outside of the mapping unit, nothing changes between a fixed mapping unit and a reconfigurable mapping unit; that is, to produce a desired subset S^i_j, the same values are neeeded for signals U and B in a reconfigurable mapping unit as they are in a fixed mapping unit. The second is that each "grouping" of the $\log z$-bits (each corresponding to a particular MUX) in the $n \log z$-bit words has the same value in an FMU; this does not have to be the case in an RMU. For example, a word in the LUT illustrated in Table 5.2 could have the value 00 01 10 11 00 01 10 11; this would imply

that bit 7 of the 8-bit output would be derived from $\vec{\pi}_0$, bit 6 would be derived from $\vec{\pi}_1$, etc. Using the ordered partitions presented in Table 5.1, a word in the LUT with the value 00 01 10 11 00 01 10 11 would result in the partition $\vec{\pi} = \langle\{7,6,3,1\},\{4,2\},\{0\},\{5\}\rangle$. Not all sets of subsets can be generated by the RMU however, as fixing the multicasts of the bits of U to the n MUXs may preclude certain subset considerations.

A Universal Reconfigurable Mapping Unit: One particular case of the RMU requires further elaboration. When $z = 2^y$, we may broadcast U to all multiplexers; that is, with suitable reconfiguration of the configuration LUT, each of the n-bits of the output signal Q can be mapped to any of the bits of the source string signal U. This RMU is a universal mapping unit (see Definition 4.1 on page 49).

Theorem 5.3 *A universal reconfigurable mapping unit* $MU(2^y,y,n,\alpha)$ *can be realized as a circuit with gate cost* $O(ny2^y)$*, a delay of* $O(y + \log n)$*, and with suitable reconfiguration of its configuration LUT, can produce any set* $\mathcal{S} \in \wp(\mathcal{Z}_n)$ *of* $\lambda = y2^y$ *independent subsets of* $\mathcal{Z}_n$*.*

Proof: The cost of the mapping unit is given by Theorem 5.2. Since all source string bits $U(i)$ are hardwired to all output places (that is, the n MUXs corresponding to the n-bit output Q), every output $Q(j)$ can be set to any input bit $U(i)$ by a y-bit grouping in the ny-bit word of the configuration LUT. This implies that every output bit can be placed in any block in an ordered partition $\vec{\pi}_k$. Since up to 2^y arbitrary partitions can be represented in this way, by Theorem 4.1, a total of $\lambda = y2^y = 2^y \log z$ independant subsets can be produced by a single set of values in the configuration LUT. ■

Remark: As noted in Theorem 4.2, a fixed mapping unit that has its partitions hardwired cannot produce any independent subsets. However, since any partition can be realized in the universal reconfigurable mapping unit through reconfiguration of its LUT, then any set of independent subsets can be realized in a single instance of the values in its LUT. As noted in Theorem 4.1, $2^y \lfloor \log z \rfloor$ is the best possible number of independent subsets.

Reconfiguration of a Reconfigurable Mapping Unit: While it is clear that the universal reconfigurable mapping unit can represent any set of partitions through reconfiguration, it is not clear if this is true for any reconfigurable mapping unit. We now address the question of which sets of subsets an RMU can generate. As we have not been able to construct all aspects of a proof, we present some of our observations as a conjecture. Before we proceed, we pin down some terms.

Recall that an RMU hardwires bits of a source string u to the MUX inputs. For $0 \leq i < 2^y$, let the i^{th} *ordered partition pattern* be the ordered partition resulting from setting all MUX controls to i. Denote the i^{th} ordered partition pattern by $\vec{\sigma_i}$. An RMU has 2^y fixed ordered partition patterns (as does the FMU). Unlike the FMU, however, the RMU can address each MUX individually, thereby using parts of different ordered partition patterns simultaneously (as demonstrated previously). Nevertheless, the existence of these hardwired patterns imposes certain restrictions on the type of sets of subsets that can be produced by the RMU.

As an example, consider the partitions hardwired in the mapping unit according to Figure 5.3. If the contents of the configuration LUT are as specified by Table 5.2, then the partition patterns for the RMU are given in Table 5.3 (which are the same as if the mapping

TABLE 5.3: Ordered partition patterns for an RMU resulting from the configuration LUT words of Table 5.2 and the hardwiring shown in Figure 5.3.

Address $b \in B = i$	$n \log z$-bit word in LUT	Ordered partition pattern $\vec{\sigma_i}$
00	00 00 00 00 00 00 00 00	$\langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$
01	01 01 01 01 01 01 01 01	$\langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$
10	10 10 10 10 10 10 10 10	$\langle\{7,6,1,0\},\{4,2\},\{5,3\}\rangle$
11	11 11 11 11 11 11 11 11	$\langle\{7,5\},\{6,4,3\},\{2,0\},\{1\}\rangle$

unit was a fixed mapping unit). However, as noted earlier, a word in the LUT with the value 00 01 10 11 00 01 10 11 would result in the partition $\vec{\pi} = \langle\{7,6,3,1\},\{4,2\},\{0\},\{5\}\rangle$

Let $\{\vec{\sigma_i} : 0 \leq i < 2^y\}$ be the set of ordered partition patterns of RMU $MU(z,y,n,\alpha)$. For each $0 \leq i < 2^y$, let $\vec{\sigma_i} = \langle T_j^i : 0 \leq j < z \rangle$. For example, consider the configuration LUT word 01 01 01 01 01 01 01 01 corresponding to $i = 1$. Then the resulting ordered partition pattern $\vec{\sigma_1}$ has blocks $T_0^1 = \{0\}$, $T_1^1 = \{1\}$, $T_2^1 = \{3, 2\}$, and $T_3^1 = \{7, 6, 5, 4\}$. Likewise, if the configuration LUT word corresponding to $i = 2$ were 00 01 10 11 01 10 11, then the ordered partition pattern $\vec{\sigma_2}$ has blocks $T_0^1 = \{5\}$, $T_1^1 = \{0\}$, $T_2^1 = \{4, 2\}$, and $T_3^1 = \{7, 6, 3, 1\}$. Note that if some bit $u(k)$ of the source string U is not used in $\vec{\sigma_i}$, then $T_k^i = \emptyset$ (for example, denoted by d in Table 5.1).

For each $0 \leq j < z$, define set

$$M_j = \bigcup_{i=0}^{2^y-1} T_j^i. \tag{5.1}$$

Set M_j is the set of all bit positions of the output to which source string bit $u(j)$ can contribute.

As an illustration of the construction of the sets M_j, consider the ordered partition patterns in Table 5.3. Then,

$$M_0 = \bigcup_{i=0}^{2^y-1} T_j^0 = \{0\} \cup \{0\} \cup \{5, 3\} \cup \{1\} = \{5, 3, 1, 0\}$$

$$M_1 = \bigcup_{i=0}^{2^y-1} T_j^1 = \{4\} \cup \{1\} \cup \{4, 2\} \cup \{2, 0\} = \{4, 2, 1, 0\}$$

$$M_2 = \bigcup_{i=0}^{2^y-1} T_j^2 = \{6, 2\} \cup \{3, 2\} \cup \{7, 6, 1, 0\} \cup \{6, 4, 3\} = \{7, 6, 4, 3, 2, 1, 0\}$$

$$M_3 = \bigcup_{i=0}^{2^y-1} T_j^3 = \{7, 5, 3, 1\} \cup \{7, 6, 5, 4\} \cup \emptyset \cup \{7, 5\} = \{7, 6, 5, 4, 3, 1\}.$$

Recall that the configuration LUT has ny bit words, each consisting of n, y-bit controls, one per MUX. We now correlate LUT values with M_j. Let $k \in M_j$. This implies that source string bit $u(j)$ goes to MUX k. Let this bit go to input $\alpha(\ell, k)$ of MUX k (it may go to multiple inputs of MUX k). Then if a LUT word has the y-bit control value $\alpha(\ell, k)$ corresponding to MUX k, then using this word guarantees that output bit $q(k)$ gets its value from source string bit $u(j)$ according to the hardwired partition $\vec{\pi}_\ell$.

Let $\mathcal{S}$ be a set of subsets of $\mathcal{Z}_n$ and let $\mathcal{S}$ induce the partition $\pi_{\mathcal{S}} = \{B_0, B_1, \ldots, B_{z-1}\}$; see Section 4.1.2 for the definition of an induced partition. We assume that $\mathcal{S}$ has $z' \leq z$ blocks.

Theorem 5.4 *Consider a mapping unit MU(z,y,n,α) with set* $\{\vec{\sigma_i} : 0 \leq i < 2^y\}$ *of ordered partition patterns and sets* M_j *defined by Equation 5.1. A set* $\mathcal{S}$ *with unordered z-partition* $\pi_{\mathcal{S}}$ *of subsets of* $\mathcal{Z}_n$ *can be realized on the mapping unit if there exists an injection* $f : \{0, 1, \ldots, z-1\} \rightarrow \{M_j : 0 \leq j < z\}$ *such that each* $B_i \in \pi_{\mathcal{S}}$ *satisfies* $B_i \subseteq M_{f(i)}$.

Proof: For each $B_i \in \pi_{\mathcal{S}}$, all elements of B_i are either present, or all absent in any subset in $\mathcal{S}$. Therefore all bit positions represented as elements belonging to B_i must have the same value, and must come from a single source string bit in U (as the partition $\pi_{\mathcal{S}}$ already has the maximum number of allowed blocks z). Assume that there exists an M_j such that $B_i \subseteq M_j$. Recall that for each element $a \in M_j$, there exists a partition that specifies a connection from $U(j)$ to $Q(a)$, that is, MUX a. Thus, if $B_i \subseteq M_j$, there exists connections (specified by ordered partitions hardwired in the mapping unit) that connect j to all elements $k \in B_i$, that is, the outputs $Q(k)$. This implies that if for all blocks B_i, there exists an order such that each $B_i \subseteq M_{f(i)}$, that is, each B_i is a subset of a different set $M_{f(i)}$, then there is a hardwired connection in the mapping unit from input bit $U(f(i))$ to all elements in B_i, for all i. ■

Conjecture 5.1 *We also conjecture that the converse is true. If there exists a block* B_i *of z-partition* $\pi_{\mathcal{S}}$ *that is not a subset of any* M_j*, then all elements belonging to* B_i *must be derived from the same source string bit of U. However, the fact that there is no set* M_j *containing all elements of* B_i *implies that there is no hardwired connection in the mapping unit (for all ordered partitions hardwired in the mapping unit) that maps a single input bit to all elements of* B_i*. Thus, the set of subsets* $\mathcal{S}$ *with the z-partition* $\pi_{\mathcal{S}}$ *cannot be produced.*

Remark: Note that if the partition does not have z blocks, the conjecture assuredly does not hold true; for example, a single subset has a 1- or 2-partition (which is not necessarily a subset of any set M_j) but is producible under many partitions. Essentially, this is because

although all bits in a partition with less than z blocks must be the same, more than one source string bit can map the values of a single block in such a case. The above formulation depends on the assumption that all elements in a block B_i must come from a single source bit. Because of this, it is difficult to characterize any set of subsets with a partition consisting of fewer than z blocks with the above formulation.

5.3 Bit-Slice Mapping Units

In this section, we consider a bit-slice mapping unit $MU(z,y,n,\alpha)$, that is, a mapping unit with $\alpha > 1$ but with α polylogarithmically bounded in n. A bit-slice mapping unit generates just part of the output subset (represented by an n-bit string) at a time. It constructs a subset over α iterations, generating $\frac{n}{\alpha}$ bits in each iteration. This allows the mapping unit to exploit repeated patterns, such as these demonstrated in Table 5.4, representing two forms

TABLE 5.4: Subsets with repeated patterns for $n = 16$, $\alpha = 4$

Subset S	Repeated Patterns
1111111111111111	1111
0001000100010001	0001
0000000100000001	0000, 0001
0000000000000001	0000, 0001
0000000011111111	0000, 1111
0000000000001111	0000, 1111
0000000000000011	0000, 0011
0000000000000001	0000, 0001

of reduction. Notice that to generate 8 strings, each 16-bits, only 6 strings, each 4-bits, need to be generated. For example, the subset $S = 0001000100010001$ can be constructed over 4 iterations using the bit pattern 0001. Overall, this allows the bit-slice mapping unit to decrease the required gate cost of its internal components in situations where an increased delay is tolerable.

A possible implementation of $MU(z,y,n,\alpha)$ is shown in Figure 5.7. A shift register acts as

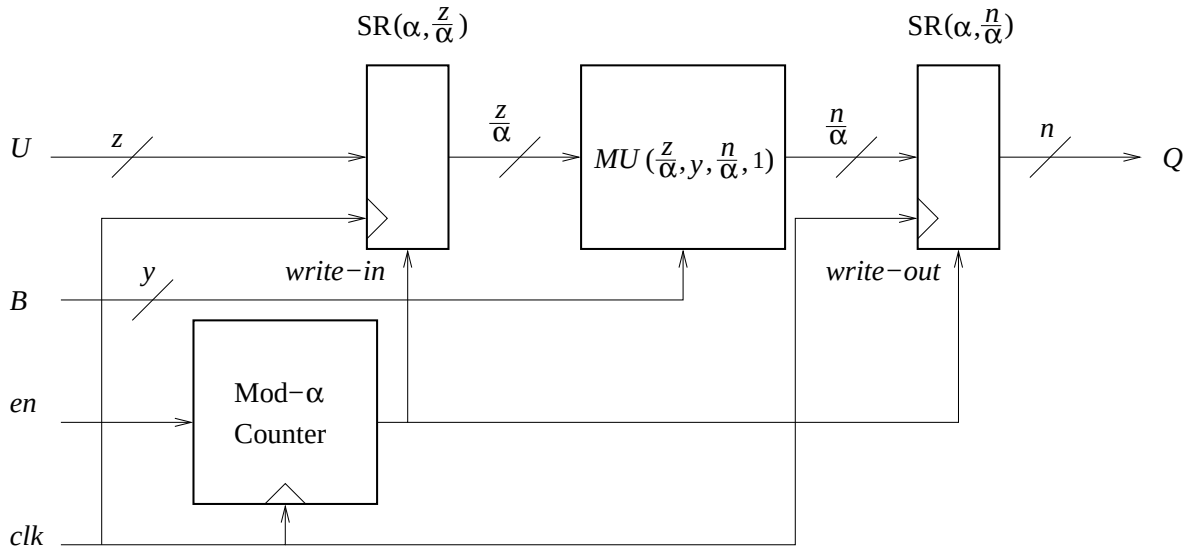


FIGURE 5.7: Bit-slice mapping unit implementation

a parallel to serial converter and stores the z-bit source strings and outputs $\frac{z}{\alpha}$-bits every cycle to the internal mapping unit $MU(\frac{z}{\alpha},y,\frac{n}{\alpha},1)$. The $\frac{n}{\alpha}$-bit output of the mapping unit is stored in another shift register which parallelizes the α, $\frac{n}{\alpha}$-bit strings into one n-bit string. A mod-α counter orchestrates this parallel to serial conversion by triggering a write-in operation on the input shift register and a write-out on the output shift register every α cycles. This allows a new source string to be input into the bit-slice mapping unit and an n-bit output q written out every α cycles.

Because the bit-slice mapping unit is a sequential circuit, we modify the definition of delay from Section 3.1.1. For sequential circuits, we assume that the clock delay of the circuit to be the longer of (a) the longest path between any flip-flop output and any flip-flop input and (b) the longest path between any circuit input and output. Using this notion of delay, we have the following result.

Theorem 5.5 *A bit-slice mapping unit $MU(z,y,n,\alpha)$, where $z \neq 2^y$, can be realized as a circuit with a gate cost of $O\Big(\log^2\alpha + n\left(1+\frac{y2^y}{\alpha}\right)\Big)$ and a delay of $O(\alpha(\log\log\alpha+\log n+y))$.*

Proof: The input and output shift registers have a gate cost of $O(z)$ and $O(n)$, respectively, and constant delays (Lemma 3.5). The mod-α counter has a gate cost of $O(\log^2\alpha)$ and a

delay of $O(\log\log\alpha)$ (Lemma 3.7). The output of the mod-α counter is tested for value α (with $O(\log\log\alpha)$ delay and $O(\log\alpha)$ cost) to generate the bits that trigger the shift registers. Because this output is fanned-out to all bits in both shift registers, it has a fan-out of $O(z+n)=O(n)$ and a delay of $O(\log z+\log n)=O(\log n)$ (Lemma 3.1).

Adding the cost and delay of the internal mapping unit, the total gate cost and delay are $O\Big(\log^2\alpha+\log\alpha+n+\frac{ny2^y}{\alpha}\Big)=O\Big(\log^2\alpha+n\left(1+\frac{y2^y}{\alpha}\right)\Big)$ and a delay of $O(\alpha(\log\log\alpha+\log n+y+\log\frac{n}{\alpha}))=O(\alpha(\log\log\alpha+\log n+y))$. ■

Remark: Note that for the number of subsets produced by a bit-slice mapping unit, the allowed cost of the mapping unit is decreased by a factor of α, and the number of source string bits is decreased by roughly a factor of α. Hence, the number of independent subsets produced by the bit-slice mapping unit is $\Theta(\frac{2^y}{\alpha}\log\frac{z}{\alpha})$.

A factor that needs attention is the matter of how partitions play out in the bit-slice mapping unit. For example, the subsets of Table 5.4 produced by a fixed mapping unit $MU(z,y,n,\alpha)$ with $z=5$, $2^y=2$ require two ordered partitions ($\vec{\pi}_1=\langle\{15,14,13,11,10,9,7,6,5,3,2,1\},\{12,4\},\{8\},\{0\}\rangle$ and $\vec{\pi}_2=\langle\{15,14,13,12,11,10,9,8\},\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$) and four, 5-bit source strings (11111, 00111, 00011, 00001) to produce the $n=16$-bit outputs. In a bit-slice mapping unit, with $\left\lceil\frac{z}{\alpha}\right\rceil=2$ and $\left\lceil\frac{n}{\alpha}\right\rceil=4$, only two ordered partitions ($\vec{\pi}'_1=\langle\{3,2\}\{1,0\}\rangle$, $\vec{\pi}'_2=\langle\{3,2,1\}\{0\}\rangle$) and three, 2-bit source strings (00, 01, and 11) are needed to produce the $\frac{n}{\alpha}$-bit repeated patterns 0011, 0001, 0000, and 1111. For these particular subsets of $\mathcal{Z}_n$, the bit-slice mapping unit shows good savings.

Consider the same example, but with the additional subset 0101010101010101. For $z=5$, $2^y=2$, two ordered partitions are needed, $\vec{\pi}_=\langle\{15,13,11,9,7,5,3,1\},\{14,6,2\},\{12,4\},\{8\},\{0\}\rangle$ and $\vec{\pi}_2=\langle\{15,14,13,12,11,10,9,8\},\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$, along with four 5-bit source strings (11111, 01111, 00111, 00001) to produce the 16-bit outputs. However, the bit-slice mapping unit of this implementation now has to produce the 4-bit pattern 0101 in addition to those previously required (in order to produce the subset 0101010101010101). Hence, a third partition $\vec{\pi}'_3=\langle\{3,1\},\{2,0\}\rangle$ is needed to produce all the 4-bit patterns. This implies that the number of inputs needed at each multiplexer in the bit-slice mapping unit is three.

Since 2^y doesn't change between the mapping unit implementation and the bit-slice mapping unit implementation, this results in a gate cost decrease of a factor slightly less then α. Thus, in determining whether or not a bit-slice mapping unit is suitable to a design, a variety of considerations must be taken into account.

Overall, the following theorem captures the performance of $MU(z,y,n,\alpha)$.

Theorem 5.6 *For any $\alpha \geq 1$, a mapping unit $MU(z,y,n,\alpha)$ has the following performance parameters:*

a) *delay of* $O(\alpha(\log y + \log n)$,

b) *gate cost of* $O\Big(n\left(1 + \frac{y2^y}{\alpha}\right)\Big)$,

c) *number of independent subsets* $\lambda = \frac{2^y}{\alpha}\left\lfloor \log \frac{z}{\alpha} \right\rfloor$, *and*

d) *total number of subsets produceable* $\Lambda = 2^y(2^z - 2)$, *provided* $y < \left\lceil \frac{n}{z-1} \right\rceil$.

■

This chapter has provided a general view of the mapping unit decoder, in terms of its cost and capabilities, and illustrated several means of realizing its operation. The next chapter incorporates this structure as part of a larger design in the configurable decoder.

Chapter 6
A Configurable Decoder

In general, a decoder is a module that maps elements of $\{0,1\}^x$ to $\{0,1\}^n$, where $x \ll n$. In a configurable decoder, this mapping can be altered. In this thesis we consider two types of configurable decoders: (1) pure LUT-based configurable decoders (described in Section 3.3) and (2) mapping unit-based configurable decoders (to which this chapter is devoted). As noted in Figure 5.2, a mapping unit comes in different forms. Likewise, a mapping unit-based configurable decoder can be integral or bit-slice, fixed or reconfigurable and general or universal.

As noted in Section 3.3, the simplest configurable decoder is a LUT; however, it is expensive and does not scale well. The main idea underlining our solutions is to use a LUT with a "narrow" output (that provides a significant amount of flexibility, considering its low cost) and a mapping unit (Chapters 4 and 5) that expands this narrow output into a wide n-bit output representing a subset of $\mathcal{Z}_n$. Figure 6.1 shows a block diagram of the configurable decoder. To put the figure in perspective, generally, $x \ll z \ll n$. So, unlike the pure LUT-based solution, our solution expands the x-bit input in stages to construct the n-bit output.

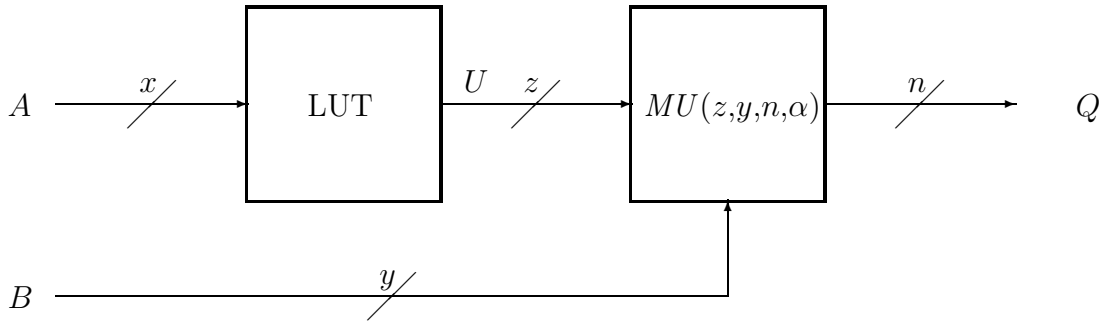


FIGURE 6.1: Block diagram of a configurable decoder $CD(x,z,y,n,\alpha)$

As discussed in Chapters 4 and 5, the mapping unit $MU(z,y,n,\alpha)$ accepts as input a z-bit string $u \in U$ and an ordered z-partition $\vec{\pi}$ (selected by a y-bit signal B). The $MU(z,y,n,\alpha)$ then uses the operation $\mu(u,\vec{\pi})$ to produce an n-bit string representative of

a subset of $\mathcal{Z}_n$. In this chapter we integrate $MU(z,y,n,\alpha)$ with a $2^x \times z$ LUT to create the configurable decoder, $CD(x,z,y,n,\alpha)$ (shown in Figure 6.1).

At this point, a fair question to ask is "what does the LUT contribute?" As noted in the previous chapters, the flexibility of the configurable decoder hinges on the LUT and the value of z (number of independent subsets). While z larger than a polynomial in n does not yield significant benefits, a small z (such as $z = \log n$) severely limits the subsets that can be generated by the mapping unit. Without the LUT, z has to be this small to address the pin limitation problem. Thus the role of the LUT is to start from a small number of input bits and expand it to z-bits, trading the value of z off with the number of locations in the LUT. This provides ample room for constructing the configurable decoder to particular specifications.

This chapter explores the properties and costs of our configurable decoder. In Section 6.1, we illustrate the mapping units of Chapter 4 in the context of our configurable decoder. In Section 6.2, we derive the basic parameters applicable to all our configurable decoders. Finally, in Section 6.3, we cast these parameters in the context of a fixed gate cost and compare the configurable decoder's theoretical performance with that of a pure LUT-based configurable decoder.

6.1 Illustrative Examples

Recall from Chapters 4 and 5 that the mapping unit uses a set $\mathcal{Y}$ of ordered partitions of the set of $\mathcal{Z}_n$ to expand the z-bit source strings to an n-bit subset of $\mathcal{Z}_n$. We begin by providing an example of this operation in the context of a configurable decoder. The first two examples demonstrate a configurable decoder with an integral mapping unit, where we consider $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$, where the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are from Table 4.1, page 43. Table 6.1 shows these sets with their unordered partitions for $n = 8$, $2^y = 2$, and $z = 4$.

Example 6.1 *:* Using sets $\mathcal{S}_0$ and $\mathcal{S}_1$ from Table 6.1, order the partitions such that $\vec{\pi}_0 = \langle\{0\},\{7,5,3,1\},\{6,2\},\{4\}\rangle$ and $\vec{\pi}_1 = \langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$. Then, for set $\mathcal{S}_0$, the source strings 1111, 1011, 1001, and 1000 would be needed to produce the subsets S_0^0,

TABLE 6.1: Sets $\mathcal{S}_0$ and $\mathcal{S}_1$ with corresponding partitions

S_j^i	$q \in Q$	$\pi_{\mathcal{S}_i}$
S_0^0	11111111	$\{\{0\},\{7,5,3,1\},\{6,2\},\{4\}\}$
S_1^0	01010101	
S_2^0	00010001	
S_3^0	00000001	
S_0^1	11111111	$\{\{7,6,5,4\},\{3,2\},\{1\},\{0\}\}$
S_1^1	00001111	
S_2^1	00000011	
S_3^1	00000001	

S_1^0, S_2^0, and S_3^0, respectively. Likewise, for set $\mathcal{S}_1$, the source strings 1111, 0111, 0011, 0001 would be needed to produce the subsets S_0^1, S_1^1, S_1^2, and S_1^3, respectively. This implies that a LUT with a size of at least 7×4 would be needed to contain all the source strings. Assume that we use a LUT of size 8×4 to store the source strings as we want to produce the subsets of $\mathcal{S}_0 \cup \mathcal{S}_1$ in the order shown in Table 6.1 for the purpose of our algorithm (in this case, two types of reduction). Assign the source string 1111 to the first row in the LUT, the source string 1011 to the second row in the LUT, and so on. Table 6.2 shows the values needed for the inputs of the configurable decoder to produce the desired subsets of $\mathcal{S}_0$ and $\mathcal{S}_1$. Here, a total of 4 bits are needed to produce all subsets. Note that these aren't the only subsets producible by the decoder. If the source strings for set $\mathcal{S}_0$ were used with the partition $\vec{\pi}_1$ and vice versa, different subsets are possible (see Table 6.3).

In the previous example, since all rows in the LUT were used to produce the subsets of $\mathcal{S}_0$ and $\mathcal{S}_1$, the "extra" subsets generated by the configurable decoder were fixed. The next example will explore a "better" ordering of the partitions $\mathcal{S}_0$ and $\mathcal{S}_1$ that provide additional options.

Example 6.2 *:* Again using sets $\mathcal{S}_0$ and $\mathcal{S}_1$ from Table 6.1, order the partitions such that $\vec{\pi}_0 = \langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$ and $\vec{\pi}_1 = \langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$. Then, for both

TABLE 6.2: Input values needed for the configurable decoder to produce the subsets of $\mathcal{S}_0$ and $\mathcal{S}_1$ in Table 6.1

$a \in A$	Source string	$b \in B$	$\vec{\pi}_i$	$q \in Q$
000	1111	0	$\langle\{0\},\{7,5,3,1\},\{6,2\},\{4\}\rangle$	11111111
001	1011			01010101
010	1001			00010001
011	1000			00000001
100	1111	1	$\langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$	11111111
101	0111			00001111
110	0011			00000011
111	0001			00000001

sets $\mathcal{S}_0$ and $\mathcal{S}_1$, the source strings 1111, 0111, 0011, 0001 produce the subsets, where 1111 produces S_0^0 and S_0^1, 0111 produces S_1^0 and S_1^1, 0011 produces S_2^0 and S_2^1, and 0001 produces S_3^0 and S_3^1. This implies that a LUT with a size of 4×4 suffices to produce all subsets.

There are two cases to consider, each with their own advantages. If a 4×4 LUT is used to hold the four needed source strings, than a savings in gate cost results over the configurable decoder in Example 6.1 (as the LUT is reduced from a 8×4 LUT to a 4×4 LUT). No "extra" subsets can be generated, however, as all combinations of source strings in the LUT and ordered partitions in the mapping unit are needed to produce the subsets of $\mathcal{S}_0$ and $\mathcal{S}_1$. In the second case, the size of the LUT remains 8×4; however, only four rows are needed to hold the source strings for $\mathcal{S}_0$ and $\mathcal{S}_1$. Thus, four additional rows exist in the LUT which could be used to produce any four of the subset pairs from Table 6.4. Note that selecting source string 1010, for example, means that both subsets 10111010 (from $\mu(1010, \vec{\pi}_0)$) and 11110010 (from $\mu(1010, \vec{\pi}_1)$) could be generated. The implications of this are that the ordering of the partitions can determine not only the size of the LUT in the configurable decoder (and thus also the values of parameters), but also the subsets that can be produced.

TABLE 6.3: Subsets produced by combining source strings of $\mathcal{S}_0$ (resp., $\mathcal{S}_1$) with partition of $\vec{\pi}_1$ (resp., $\vec{\pi}_0$)

$\mathcal{S}_i$	Source String (u)	$\vec{\pi}_i$	$\mu(u, \vec{\pi}_i)$
$\mathcal{S}_0$	1111	$\langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$	11111111
	1011		11110011
	1001		11110001
	1000		11110000
$\mathcal{S}_1$	1111	$\langle\{0\},\{7,5,3,1\},\{6,2\},\{4\}\rangle$	11111111
	0111		11111110
	0011		01010100
	0001		00010000

The next example illustrates a configurable decoder with a bit-slice mapping unit.

Example 6.3 Consider the sets $\mathcal{S} = \mathcal{S}_0$ and $\mathcal{S}_1$ shown in Table 6.5, where $z = 5$ and $2^y = 2$. Note that the ordered partitions for sets $\mathcal{S}_0$, $\mathcal{S}_1$ are $\vec{\pi}_0 = \langle\{15,13,11,9,7,5,3,1\},\{14,10,6,2\}, \{12,4\},\ \{8\},\{0\}\rangle$ and $\vec{\pi}_1 = \langle\{15,14,13,12,11,10,9,8\},\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$, respectively. Then a $CD(x,z,y,n,\alpha)$ with a fixed mapping unit would require 16 multiplexers with 2 inputs each and a 5×5 LUT to hold the values of the source strings (note that this is due to the intelligent ordering; in general the LUT could be as much as 10×5). Assume that $\alpha = \log n = 4$. Then in each iteration of a $CD(x,z,y,n,\alpha)$, the decoder must produce the $\frac{n}{\alpha}$-bit strings from the $\left\lceil \frac{z}{\alpha} \right\rceil$-bit strings shown in Table 6.6.

For these $\frac{n}{\alpha}$-bit strings, three partitions are needed, $\vec{\pi}_0^{bs} = \langle\{3,2\},\{1,0\}\rangle$, $\vec{\pi}_1^{bs} = \langle\{3,1\},\{2,0\}\rangle$, and $\vec{\pi}_2^{bs} = \langle\{3,2,1\},\{0\}\rangle$. Since the original fixed mapping unit had values of $z = 5$ and $2^y = 2$, the number of inputs to each multiplexer in the internal mapping unit of the bit-slice mapping unit would increase by one (from 2 to 3). However, the number of multiplexers would decrease from $n = 16$ to $\frac{n}{\alpha} = 4$. This would imply a reduction in cost by a factor of $\frac{16 \times 2}{4 \times 3} \approx 2.67$.

TABLE 6.4: Possible subsets produceable from $\mu(u_j, \vec{\pi}_0)$ and $\mu(u_j, \vec{\pi}_1)$; $\vec{\pi}_0 = \langle\{7,5,3,1\},\{6,2\},\{4\},\{0\}\rangle$, $\vec{\pi}_1 = \langle\{7,6,5,4\},\{3,2\},\{1\},\{0\}\rangle$

$u_j \in U$	$\mu(u_j, \vec{\pi}_{\mathcal{S}_0})$	$\mu(u_j, \vec{\pi}_{\mathcal{S}_1})$
0000	00000000	00000000
0001	00000001	00000001
0010	00010000	00000010
0011	00010001	00000011
0100	01000100	00001100
0101	01000101	00001101
0110	01010100	00001110
0111	01010101	00001111
1000	10101010	11110000
1001	10101011	11110001
1010	10111010	11110010
1011	10111011	11110011
1100	11101110	11111100
1101	11101111	11111101
1110	11111110	11111110
1111	11111111	11111111

Regardless, the LUT must still supply a z-bit word to the bit-slice mapping unit (which in this case may increase to a 6-bit word based on the rounding of $\left\lceil \frac{z}{\alpha} \right\rceil$). Thus, the implementation depends on the allowable costs, the number of z-bit source strings and the corresponding size of the LUT, and the subsets that must be produced.

With these examples providing the proper context of the mapping unit with regards to the preceding LUT, we now proceed to the performance of a configurable decoder $CD(x,z,y,n,\alpha)$.

TABLE 6.5: Sets $\mathcal{S}_0$ and $\mathcal{S}_1$ of $\mathcal{Z}_{16}$ for Example 6.3

S_j^i	$q \in Q$	$z \in U$
S_0^0	1111111111111111	11111
S_1^0	0101010101010101	01111
S_2^0	0001000100010001	00111
S_3^0	0000000100000001	00011
S_4^0	0000000000000001	00001
S_0^1	1111111111111111	11111
S_1^1	0000000011111111	01111
S_2^1	0000000000001111	00111
S_3^1	0000000000000011	00011
S_4^1	0000000000000001	00001

6.2 Performance of $CD(x,z,y,n,\alpha)$

In this section we develop general expressions for the delay, gate cost, and subsets that can be produced by a configurable decoder $CD(x,z,y,n,\alpha)$.

Delay: The delay of $CD(x,z,y,n,\alpha)$ is clearly the sum of the delays due to a $2^x \times z$ LUT and a $MU(z,y,n,\alpha)$. Therefore we have the following result.

Theorem 6.1 *For any $\alpha \geq 1$, a configurable decoder $CD(x,z,y,n,\alpha)$ has a delay of $O(x + \log z + \alpha(y + \log n))$.*

Proof: By Lemma 3.4, a $2^x \times z$ LUT has a delay of $O(x + \log z)$. By Theorem 5.6, the delay of a mapping unit $MU(z,y,n,\alpha)$ is $O(\alpha(y + \log n))$. Overall, this results in a delay of $O(x + \log z) + O(\alpha(y + \log n)) = O(x + \log z + \alpha(y + \log n))$. ■

Remark: In general, $y = O(\log n)$, $x = O(\log n)$, and z is polynomial in n. Therefore, the delay is usually $O(\alpha \log n)$.

TABLE 6.6: $\frac{n}{\alpha}$-bit strings produced from $\left\lceil \frac{z}{\alpha} \right\rceil$-bit input strings in $CD(x,z,y,n,\alpha)$

S_j^i	$\left\lceil \frac{z}{\alpha} \right\rceil$-bit input string	$\frac{n}{\alpha}$-bit string produced
S_0^0	11	1111
S_1^0	01	0101
S_2^0	01	0001
S_3^0	00, 01	0000, 0001
S_4^0	00, 01	0000, 0001
S_0^1	11	1111
S_1^1	00, 11	0000, 1111
S_2^1	00, 11	0000, 1111
S_3^1	00, 01	0000, 0011
S_4^1	00, 01	0000, 0001

Gate Cost: As in delay, the gate cost of $CD(x,z,y,n,\alpha)$ is the summation of the gate costs of a $2^x \times z$ LUT and a $MU(z,y,n,\alpha)$. We now have the following result.

Theorem 6.2 *For any $\alpha \geq 1$, a configurable decoder $CD(x,z,y,n,\alpha)$ has a gate cost of $O\Big(2^x(x+z) + n\left(1 + \frac{y2^y}{\alpha}\right)\Big)$.*

Proof: By Lemma 3.4, a $2^x \times z$ LUT has a gate of $O(2^x(x+z))$. By Theorem 5.6, the gate cost of a mapping unit $MU(z,y,n,\alpha)$ is $O\Big(n\left(1 + \frac{y2^y}{\alpha}\right)\Big)$. Overall, this results in a gate cost of $O(2^x(x+z)) + O\Big(n\left(1 + \frac{y2^y}{\alpha}\right)\Big) = O\Big(2^x(x+z) + n\left(1 + \frac{y2^y}{\alpha}\right)\Big)$. ■

Producible Subsets: Recall from Chapter 4 that the subsets produced by a decoder can be broadly divided into two classifications: independent subsets (that is, the set $\mathcal{S}'$) and subsets produced by the decoder that are a result of choices made in the configuration of the decoder (that is, the set $\mathcal{S}'$). We extend the results of Chapter 4 here, beginning with the set $\mathcal{S}$ of independent subsets.

Theorem 6.3 *A configurable decoder CD(x,z,y,n,α) can produce at least* $\lambda = \min\left\{2^x, \frac{2^y}{\alpha}\left\lfloor\frac{\log z}{\alpha}\right\rfloor\right\}$ *independent subsets.*

Proof: By Theorem 5.6, a mapping unit $MU(z,y,n,\alpha)$ can produce $\frac{2^y}{\alpha}\left\lfloor\frac{\log z}{\alpha}\right\rfloor$ independent subsets of $\mathcal{Z}_n$. Since each source string can be unique, each of the source strings uses one of the 2^x rows in the LUT preceding the mapping unit. Thus, the number of independent subsets produced by $CD(x,z,y,n,\alpha)$ is at least $\lambda = \min\left\{2^x, \frac{2^y}{\alpha}\left\lfloor\frac{\log z}{\alpha}\right\rfloor\right\}$. ■

We now extend the results for the maximum number of subsets producible by a configurable decoder $CD(x,z,y,n,\alpha)$.

Theorem 6.4 *For* $2^x \leq 2^z - 2$ *and* $y \leq \left\lceil\frac{n}{z-1}\right\rceil - 1$, *a configurable decoder CD(x,z,y,n,α) exists that can produce* $\Lambda = 2^{x+y}$ *distinct subsets of* $\mathcal{Z}_n$.

Proof: By Theorem 4.3, a $MU(z,y,n,\alpha)$ using the Lemmas 4.2 and 4.3 can produce CY subsets, where $Y = 2^y \leq 2^{\lceil\frac{n}{z-1}\rceil - 1}$, and C is a subset of the $2^z - 2$ values of U that can result in distinct subsets of $\mathcal{Z}_n$. As the LUT can produce a subset of the 2^z values of U, then if $2^x = C \leq 2^z - 2$, a configurable decoder consisting of a $2^x \times z$ LUT and the same $MU(z,y,n,\alpha)$ can produce $CY = 2^{x+y}$ distinct subsets of $\mathcal{Z}_n$. ■

These results are now used to establish that our configurable decoder asymptotically outperforms a pure LUT-based configurable decoder in every conceivable situation.

6.3 Gate-Cost Constrained Configurable Decoders

In this section we consider a configurable decoder $CD(x,z,y,n,\alpha)$ whose gate cost is $G \geq n$. We constrain the delay to be $O(\alpha \log n)$ and G to be polynomial in n. Also, recall from Chapter 5 that $z \ll n$ and α is polylogarithmically bounded in n, that is, $\alpha = O(\log^k n)$ for constant $k > 0$. We first derive conditions on x, z, and y needed to preserve the gate cost of G. Before we proceed, we note the maximum number of independent subsets for a pure LUT-based configurable decoder.

Lemma 6.1 *A pure LUT-based configurable decoder with gate cost G can produce at most $\Theta\left(\frac{G}{n}\right)$ independent subsets.*

Proof: By Lemma 3.4, a $2^{z_L} \times m$ LUT has a gate cost of $O(2^{z_L}(z_L + m))$, where z_L is the number of input bits, 2^{z_L} is the number of rows in the LUT, and m is the number of output bits (that is, the length of the word in the LUT). Since $m = n$, each independent subset requires one row in the LUT. This results in $O(2^{z_L}(z_L + n))$ cost. For a cost of G, we have $2^{z_L} = \Theta\left(\frac{G}{n}\right)$. This implies that the maximum number of rows (and thus, independent subsets) in the LUT is $\Theta\left(\frac{G}{n}\right)$. ■

Remark: For a pure LUT-based configurable decoder, the number of independent subsets λ is also the total number of subsets producible, Λ.

From Theorem 6.1 a delay of $O(\alpha \log n)$ implies $x + \log z + \alpha(y + \log n) = O(\alpha \log n)$, or

$$\frac{x}{\alpha} + \frac{\log z}{\alpha} + y = O(\log n). \tag{6.1}$$

Since $z \leq n$, $\log z = O(\log n)$ is guaranteed. The constraints that $\frac{x}{\alpha} + y = O(\log n)$ implies that $x + y$ is polylog in n (as α is polylog in n). This is consistent with the fact that the number of pins entering the configurable decoder must be small.

From Theorem 6.2, a gate cost of G implies that

$$2^x(x + z) = O(G) \tag{6.2}$$

and

$$n\left(1 + \frac{y2^y}{\alpha}\right) = O(G). \tag{6.3}$$

From Equation 6.2 we have

$$2^x = O\left(\frac{G}{\log G}\right) = O\left(\frac{G}{\log n}\right) \tag{6.4}$$

and

$$z = O\Big(\frac{G}{2^x}\Big). \tag{6.5}$$

From Equation 6.3, we have

$$2^y = O\left(\frac{\frac{G\alpha}{n}}{\log \frac{G\alpha}{n}}\right). \tag{6.6}$$

Since the number of independent subsets is $\Theta(2^y \log z)$ and the cost of the LUT increases with z, we need a large value of z (to get a larger number of independent subsets), but not so large that the LUT becomes too expensive. Select

$$z = \Theta(n^\epsilon) \tag{6.7}$$

for some small constant $\epsilon > 0$, so that $\log z$ is still $\Theta(\log n)$ but the contribution of z to the LUT cost is $\Theta(n^\epsilon)$. Since $x = O(\log^k n)$, for constant k, we have $x + z = \Theta(z)$. So, from Equation 6.5 select

$$2^x = \Theta\Big(\frac{G}{z}\Big) = \Theta\Big(\frac{G}{n^\epsilon}\Big). \tag{6.8}$$

Clearly this will result in $\Theta(\log n)$ delay and $\Theta(G)$ gate cost.

The number of independent subsets produced is $\Theta\Big(\min\Big\{2^x, \frac{2^y}{\alpha}\log\frac{z}{\alpha}\Big\}\Big)$ (see Theorem 6.3) which is $\Theta\Big(\min\Big\{\frac{G}{n^\epsilon}, \frac{G\log\frac{n^\epsilon}{\alpha}}{n\log\big(\frac{G\alpha}{n}\big)}\Big\}\Big)$.

Observe that $\alpha = o(n^\delta)$ for every constant $\delta > 0$. Therefore, $\Theta\Big(\min\Big(\frac{G}{n^\epsilon}, \frac{G\log\big(\frac{n^\epsilon}{\alpha}\big)}{n\log\big(\frac{G\alpha}{n}\big)}\Big)\Big) = \Theta\Big(\min\Big(\frac{G}{n^\epsilon}, \frac{G\log n}{n\log\big(\frac{G}{n}\big)}\Big)\Big)$.

Note that for asymptotically large n, $\log\Big(\frac{n^\epsilon}{\alpha}\Big) = \Theta(\log n)$ and $\log\Big(\frac{G\alpha}{n}\Big) = \Omega(1)$. So, $\frac{G\log\big(\frac{n^\epsilon}{\alpha}\big)}{n\log\big(\frac{G\alpha}{n}\big)} = O\left(\frac{G\log n}{n}\right) = O\Big(\frac{G}{n^\epsilon}\Big)$. So for asymptotically large n, the number of independent subsets is $\Theta\left(\frac{G\log n}{n\log\Big(\frac{G}{n}\Big)}\right) = \Theta\left(\frac{G\log n}{n\log\Big(\frac{G\alpha}{n}\Big)}\right)$.

If $\frac{G}{n} = \Theta(\log^\sigma n)$ for constant $\sigma > 0$ then $\log\Big(\frac{G}{n}\Big) = \Theta(\log\log n)$. Here, the number of independent subsets is $\Theta\Big(\frac{G\log n}{n\log\log n}\Big)$ (as α is polylog in n), while the maximum number of dependent subsets can be as large as $\Theta(2^{x+y}) = \Theta\Big(\frac{G}{n}\frac{G\alpha}{n^\epsilon}\frac{1}{\log\log n}\Big) = \Theta\Big(\frac{n^{1-\epsilon}\log^{2\sigma} n}{\log\log n}\Big)$. On the other hand, if $G = \Theta(n^\sigma)$ for any $\sigma > 0$, then the number of independent subsets is

$\Theta\left(\frac{G}{n}\right)$, while the number of dependent subsets can be as large as $\Theta(2^{x+y}) = \Theta\left(\frac{n^{\delta}}{\log^{\psi} n}\right)$, for constants $\delta, \psi > 0$.

From the above discussion and Lemma 6.1, we have the following result that establishes the advantages of our configurable decoder compared to the pure LUT-based solution.

Lemma 6.2 *A configurable decoder CD(x,z,y,n,α) with polylogarithmically bounded α and polynomially bounded gate cost $G \geq n$ produces at least λ independent subsets, where*

$$\lambda = \begin{cases} \frac{G \log n}{n \log \log n}, & \text{if } \frac{G}{n} \text{ is polylogarithmically bounded in } n \\ \frac{G}{n}, & \text{otherwise,} \end{cases}$$

and it is capable of producing a total number of Λ subsets, where

$$\Lambda = \begin{cases} \frac{G}{n}\left(\frac{n^{\epsilon} \log^{\sigma} n}{\log \log n}\right), & \text{if } \frac{G}{n} \text{ is polylogarithmically bounded in } n \\ \frac{G}{n}\left(\frac{n^{\epsilon}}{\log^{\sigma} n}\right), & \text{otherwise,} \end{cases}$$

where $\epsilon, \sigma > 0$ are constants. ■

Remark: Since the total number of dependent subsets depends on the value of 2^x, a different choice in the values of z may allow 2^x to be slightly larger, thereby also increasing the number of total subsets producible by a configurable decoder. However, this would also decrease the number of independent subsets; therefore, we do not consider it here.

From Lemma 6.2, we have the following.

Theorem 6.5 *Let P be a pure LUT-based configurable decoder and let C be the proposed configurable decoder, each producing subsets of $\mathcal{Z}_n$. If both decoders have a gate cost of $\Theta(G) \geq n$, then*

a) *if $G = \Theta(n \log^{\sigma} n)$, then C produces a factor of $\Theta\left(\frac{\log n}{\log \log n}\right)$ more independent subsets than P and is capable of producing a factor of $\Theta\left(\frac{n^{\epsilon} \log^{\sigma} n}{\log \log n}\right)$ more dependent subsets for any constant $0 \leq \epsilon < 1$.*

b) *if* $G = n^{1+\sigma}$*, then* C *would produce the same order of independent subsets as* P *and is capable of producing up to* $\Theta\Big(\frac{G}{n}\Big(\frac{n^{\epsilon}}{\log^{\sigma} n}\Big)\Big)$ *dependent subsets, for any constants* $0 \leq \epsilon < 1$*.*

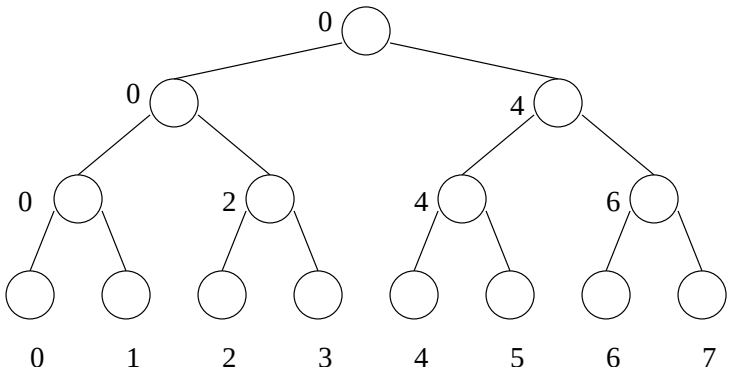
■

This chapter has shown that the proposed configurable decoder has substantial advantages over both fixed and pure LUT-based configurable decoders.

Chapter 7

Implementations of Useful Subsets

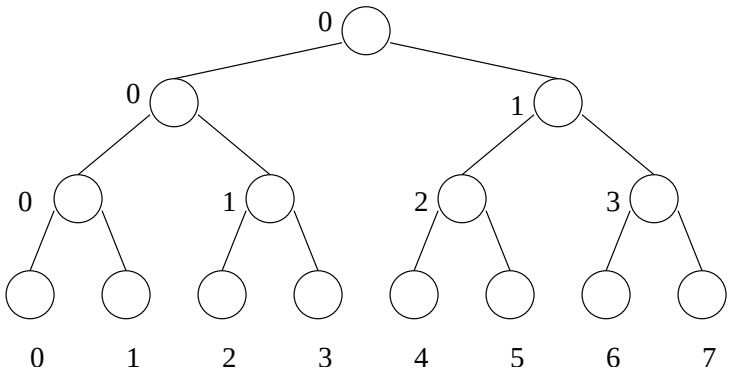
Many applications and algorithms display standard patterns of resource use. For example, consider a binary tree reduction, shown in Figure 7.1 [11]. In each reduction, the number



Corresponding n-bit patterns

00000001
00010001
01010101
11111111

(a)



Corresponding n-bit patterns

00000001
00000011
00001111
11111111

(b)

FIGURE 7.1: Two binary tree reductions of $n = 8$ elements

of resources is reduced by a factor of two in each level of the tree; Figure 7.1(a) and (b) illustrate this for two particular reductions. The bit patterns representing these reductions are also shown, where a bit has a value of '1' if it survives the reduction at a particular level in the tree and a value of '0' if it does not.

Communication patterns can also induce subsets. For example, if a node can either send or receive in a given communication, but not both simultaneously, then for an AS-

CEND/DESCEND pattern of communications [1] we have the send/receive pairs shown in Figure 7.2. The subsets represent a set of processors that may be sending (or receiving)

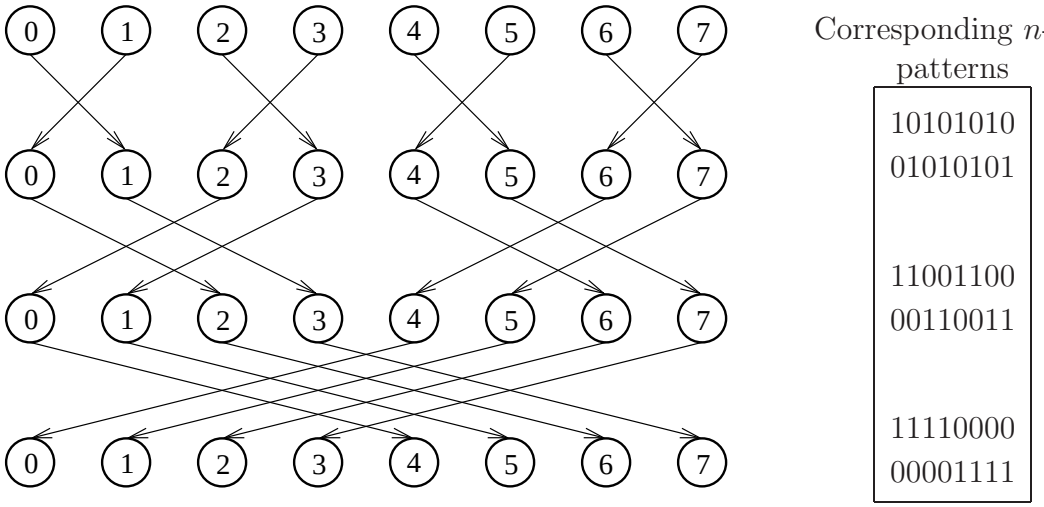


FIGURE 7.2: ASCEND/DESCEND communication pairs for $n = 8$

simultaneously.

In this chapter we examine three useful classes of subsets, namely (1) Binary Reduction (Section 7.1), (2) ASCEND/DESCEND (Section 7.2), and (3) 1-hot (Section 7.3). We examine ways of implementing these classes of subsets in mapping units as an indication of where the mapping unit can successfully take advantage of patterns in communication and where certain patterns pose challenges.

7.1 Binary Reduction

As illustrated previously, the class of binary tree based reduction algorithms reduces the number of resources by a factor of two in each level of the algorithm. This reduction can occur in a variety of ways; regardless, all binary tree based reductions have the following properties.

1. For any set $\mathcal{S}$ of subsets with n-bit patterns characterizing a binary tree based reduction, the number of subsets in the set is $\log n + 1$ (the additional subset comes from including the root of the binary tree).

2. Assume that if $S_i(j) = 1$, then resource j participates in the reduction. Then, if the subsets in $\mathcal{S}$ are ordered such that $S_0 \in \mathcal{S}$ corresponds to the state in the reduction where only 1 resource exists, S_1 corresponds to the state in the reduction where $2^1 = 2$ resources participate, and so on, then the number of bits with a value of '1' in subset S_i is 2^i. Also for each $i \leq \log n$, $S_i \subset S_{i+1}$.

As an example of this, consider the two binary tree based reductions illustrated previously, and shown again here in Table 7.1. Consider the set S_0^2 and the set S_1^2. Here, $i = 2$; thus,

TABLE 7.1: Two binary tree based reduction patterns

S_0^i	n-bit pattern		S_1^i	n-bit pattern
S_0^0	00000001		S_1^0	00000001
S_0^1	00010001		S_1^1	00000011
S_0^2	01010101		S_1^2	00001111
S_0^3	11111111		S_1^3	11111111

the number of bits with a value of '1' in the n-bit pattern is $2^2 = 4$, which is what is shown.

From this, we can conclude that a mapping unit with a single $(\log n + 1)$-block partition π can produce all $\log n + 1$ subsets with $\log n + 1$ source strings. Note that the product of any two partitions induced by the subsets of $\mathcal{S}$ result in exactly one new block, as exactly 2^{i-1} bits are different between S_k^{i-1} and S_k^i, and all 2^{i-1} bits that are different have the same value. Hence, $\pi_{S_k^{i-1}}\pi_{S_k^i}$ results in one new block to account for these 2^{i-1} bits.

For example, consider the set $\mathcal{S}_0$. Then, the induced partitions for the subsets are $\pi_{0,0} = \{\{7,6,5,4,3,2,1\},\{0\}\}$, $\pi_{0,1} = \{\{7,6,5,3,2,1\},\{4,0\}\}$, $\pi_{0,2} = \{\{7,5,3,1\},\{6,4,2,0\}\}$, and $\pi_{0,3} = \mathcal{Z}_n$. Note that the product of $\pi_{0,0}\pi_{0,1}$ results in $2^0 = 1$ bit in a new block (bit 4); the product of $\pi_{0,0}\pi_{0,1}\pi_{0,2}$ results in $2^1 = 2$ bits in a new block (bits 6 and 2), and so on.

From this illustration, we can also note that if a single configurable decoder is to produce two or more such binary tree based reductions, then the $(\log n + 1)$-partitions can be ordered such that the same $\log n + 1$ source strings produce any of the sets, as source string i contains the same number of 1's and 0's corresponding to the blocks in the partition regardless of the layout of resource allocation in a binary tree based reduction.

7.2 ASCEND/DESCEND

The subsets of the ASCEND/DESCEND class of communications (See Figure 7.2) are more difficult than those of the binary tree based reduction for a mapping unit to produce. This is because the product of all induced partitions of the $2 \log n$ subsets of the ASCEND/DESCEND class of communications results in an n-partition of $\mathcal{Z}_n$; as $z \ll n$, this cannot be represented by a single partition.

One method of generating these subsets is to use $\frac{\log n}{\log z}$ z-partitions, each with $2 \log z$ source strings (where z is a power of 2, say $z = 2^k$). Note that for a given level of the ASCEND/DESCEND communications, the send/receive pairs are complements; since all bit positions have different values between the two subsets for a given level, a single 2-partition can represent both subsets with 2 source strings. For example, the partition for the first level of communications is $\pi_1 = \{\{7, 5, 3, 1\}, \{6, 4, 2, 0\}\}$. Taken for $\log z$ such levels, this results in a single z-partition that with $2 \log z$ source strings can produce $2 \log z$ of the different $2 \log n$ subsets. For example, consider $z = 4$. Then, $\log z = 2$, which implies that two levels can be represented by a single partition. If a partition represents levels one and two, then this results in the partition $\pi = \{\{7, 3\}, \{6, 2\}, \{5, 1\}, \{4, 0\}\}$.

Taken for all $2 \log n$ subsets, this results in a total of $\frac{\log n}{\log z}$ such partitions, and a total of $2 \log z$ source strings. Table 7.2 illustrates a possible ordering of the partitions and source strings for the ASCEND/DESCEND bit patterns shown in Figure 7.2.

7.3 1-Hot

Recall from Section 3.2.2 that a set of 1-hot subsets is a set of n-bit subsets of $\mathcal{Z}_n$, where each of the n-bit patterns has only one active bit (usually with a value of '1'), all other bits being inactive (usually '0'). Table 7.3 illustrates this for $n = 16$.

Even though the 1-hot sets are easy to produce in a conventional fixed decoder, they present one of the more difficult classes for our configurable decoder. Note that each subset of the 1-hot set has an induced partition with 2 blocks, where one block contains the bit position of the bit with a value of '1' and the other block contains all other bit positions.

TABLE 7.2: Partitions and source-strings generated for ASCEND/DESCEND bit patterns; for $n = 8$ and $z = 4$

S_i	$\vec{\pi}$	Source strings	Bit-pattern
S_0	$\langle\{7,3\},\{6,2\},\{5,1\},\{4,0\}\rangle$	1010	10101010
S_1		0101	01010101
S_2		1100	11001100
S_3		0011	00110011
S_4	$\langle\{7,6,5,4\},\{3,2,1,0\}\rangle$	$dd10$	11110000
S_5		$dd01$	00001111

d denotes a don't care value

Without loss of generality, assume that block B_0 is the single element block in each induced partition. Since each subset has a different bit position with a value of '1', then each induced partition has a different bit position in block B_0. Using the method from Section 4.1.2 (page 43), each product of $\pi_i\pi_j$ would result in a partition with an additional block. Taken for all n partitions, this would result in an n-partition; clearly, this is difficult for a mapping unit to produce as each partition used by it has at most $z \ll n$ blocks.

As noted in Section 7.2, the ASCEND/DESCEND class of subsets also induces an n-partition; however, unlike that class, we have a simpler solution here. One method of producing the 1-hot subsets in a configurable decoder is to use a LUT with $2^x = n$ rows (or $x = \log n$). By Lemma 3.4, a LUT contains a 1-hot address decoder. Since a configurable decoder $CD(x,z,y,n,\alpha)$ contains a $2^x \times z$ LUT, with $n = 2^x$, a simple switch allowing the output of the LUT's address decoder to be the output of the configurable decoder automatically allows the configurable decoder to produce the 1-hot subset. We develop a slightly different solution in Section 9.1.

TABLE 7.3: A set of 1-hot subsets of $\mathcal{Z}_{16}$

S_i	n-bit value															
S_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
S_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
S_2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
S_3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
S_4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
S_5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
S_6	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
S_7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
S_8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
S_9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
S_{10}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
S_{11}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
S_{12}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
S_{13}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
S_{14}	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_{15}	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Chapter 8
Simulation Results

This chapter presents simulation results for the configurable decoders of Chapter 6. While the previous chapters analytically established the validity of our approach, the simulations in this chapter allow the constants hidden by the asymptotic notation in the cost equations to be analyzed and trends in the data to be extrapolated. The aim of the simulations is twofold, (1) to compare our solutions to existing solutions and (2) to derive reasonable predictors of the constants (cost factors independent of problem size n) across technologies. While these results are specific to our implementation, they nevertheless are good predictors of trends that may be expected from the state of the art technology.

Section 8.1 outlines our simulation methodology, including the parameters for all simulations and an explanation as to why they were chosen, details regarding the CAD tools used, and the analysis methods. Section 8.2 provides the simulation results for both integral decoders (Section 8.2.1) and bit-slice decoders (Section 8.2.2). Finally in Section 8.3, regression functions and expected trends are illustrated.

8.1 Methodology

In this section, we detail the rationale for our choice of simulation parameters, including details of problem size, CAD tools, and analysis methods.

Choice of Problem Parameters: As noted above, one of the aims of the simulations is to compare existing decoders to the proposed solutions. The 1-hot decoder is simple and one of the most widely used. Its $\Theta(\log n)$ delay and $\Theta(n \log n)$ gate cost are used as baselines against which other delays and gate costs are compared. However, the 1-hot decoder has no flexibility ($\lambda = 0$, see Section 3.1.1) in terms of the types of subsets it can produce. Among the configurable decoders, we expect the configurable decoder with a fixed mapping unit to have the best performance (measured as the number of independent subsets for a given gate cost). This is because all the configurable decoders can be wired or reconfigured to produce

any set of $2^y \log z$ independent subsets; however, a configurable decoder based on a fixed mapping unit has the lowest constant terms for its gate cost due to its complexities.

In addition to the 1-hot decoder and the configurable decoder with a fixed mapping unit, the modules we study in this chapter include the pure LUT-based configurable decoder (see Section 3.3) and the configurable decoder with a reconfigurable mapping unit (Section 5.2). We also separately consider a universal configurable decoder with reconfigurable mapping unit and a configurable decoder with fixed mapping unit with similar design parameters. We do not expressly simulate the bit-slice configurable decoder, but outline an approach to derive results for it. For each decoder, we let $n = 2^k$ for $k \geq 2$. For most simulations, system restrictions allow values up to $n = 256$.

Consider the decoders shown in Figure 8.1. All decoder have n-bit outputs. Of these the value of n completely specifies only the 1-hot decoder. For the configurable decoders, we use the number of independent subsets generated as a common thread.

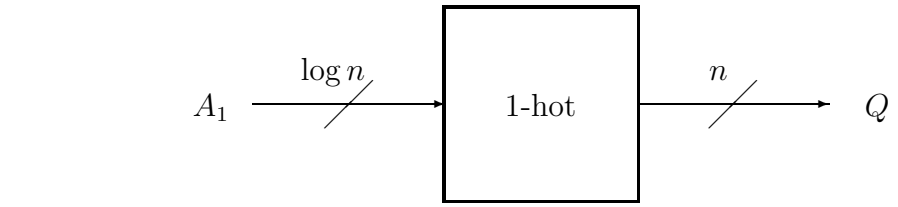
Consider the configurable decoder with fixed mapping unit of Figure 8.1(c). Let its gate cost be $G = n \log n$. Then, from Section 6.3, we have $2^{y_f} = \dfrac{\frac{G}{n}}{\log\left(\frac{G}{n}\right)} \approx \dfrac{\log n}{\log\log n}$. Also let $z_f = n^\epsilon$ (Section 6.3). Clearly, ϵ needs to be as large as possible. The value of x_f is bounded by the cost of the LUT in the configurable decoder, so we set $2^{x_f} = \dfrac{G}{z_f} = \dfrac{G}{n^\epsilon}$. However, the number of independent subsets produced $\lambda_f = 2^{y_f} \log z_f$ must be no more than $2^{x_f} = \dfrac{G}{n^\epsilon}$. Putting these together we have $2^{x_f} = \dfrac{G}{n^\epsilon} \geq 2^{y_f} \log z_f \approx \dfrac{\epsilon \log^2 n}{\log\log n}$.

Thus, for each value of n, we need to find the largest value of ϵ such that

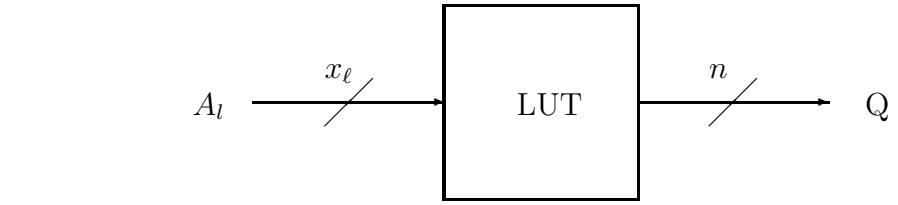
$$\frac{n^{1-\epsilon}}{\epsilon} \geq \frac{\log n}{\log\log n}.$$

Table 8.1 shows the values of n, the corresponding ϵ, and value of n^ϵ, $\dfrac{\epsilon \log^2 n}{\log\log n}$, and $\dfrac{\log n}{\log\log n}$ (the last three being indicative of the values of z_f, $\lambda_f \approx 2^{x_f}$, and 2^y, respectively).

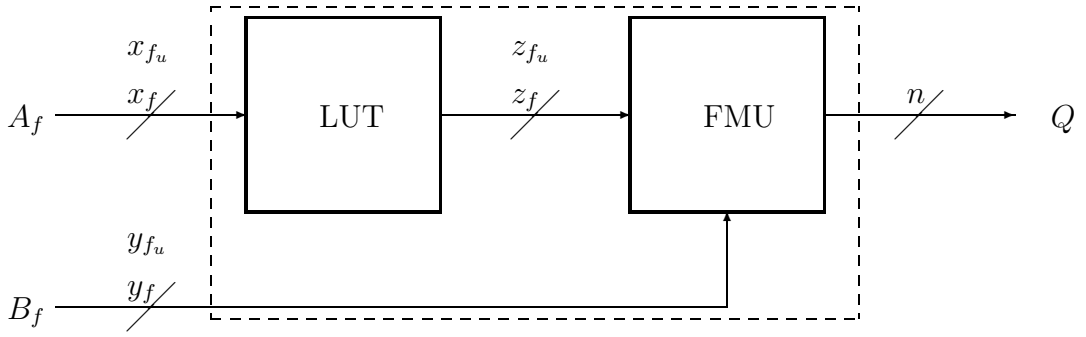
With this table as the guildeline, we now derive values of z_f, x_f, and y_f that produce λ_f independent subsets for a configurable decoder with fixed mapping unit. Table 8.2 shows these values. In deriving these values, we made simple approximations (using $\lceil\ \rceil$ and $\lfloor\ \rfloor$) to



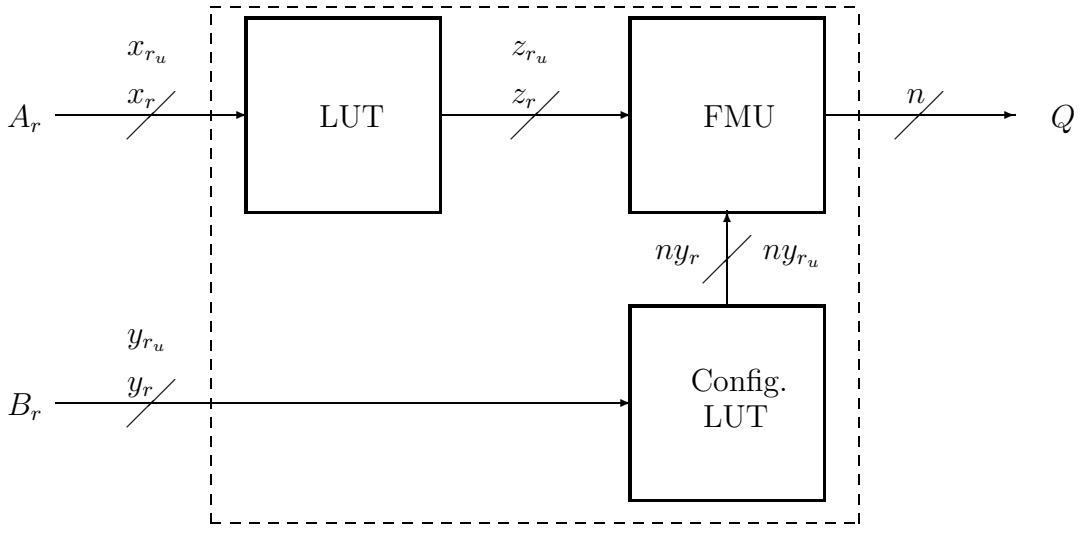
(a)



(b)



(c)



(d)

FIGURE 8.1: Block diagrams of all decoders simulated, (a) 1-hot, (b) pure LUT-based, (c) configurable decoder with FMU, and (d) configurable decoder with RMU

TABLE 8.1: Parameter values for a configurable decoder with FMU, for $G = n \log n$

n	ϵ	n^{ϵ}	$\frac{\epsilon \log^2 n}{\log \log n}$	$\frac{\log n}{\log \log n}$
4	0.7284997	2.754537	2.914	2
8	0.8002940	5.28126	4.54437	1.89279
16	0.8210948	9.74309	6.56876	2
32	0.8318129	17.865	8.95606	2.15338
64	0.8395750	32.8415	11.6925	2.32112
128	0.8461295	60.6698	14.7685	2.49345
256	0.8520038	112.676	18.1761	2.6667

ensure that parameters (signal sizes, number of subsets, etc.) are integers. We now use λ_f for each value of n as the basis to derive parameters for other configurable decoders. In fact,

TABLE 8.2: Parameter values for a configurable decoder with fixed mapping unit (CDF)

n	x_f	y_f	z_f	λ_f
4	2	1	3	3
8	3	1	6	5
16	3	1	10	7
32	4	2	18	9
64	4	2	33	12
128	4	2	61	15
256	5	2	113	19

we adjust the parameters to make the number of independent subsets λ the same across all configurable decoders.

For a pure LUT-based configurable decoder (see Figure 8.1(b)) the number of independent subsets is $\lambda_\ell = 2^{x_\ell}$. If we set $\lambda_\ell = \lambda_f$, then

$$x_\ell = \log \lambda_\ell = \log \left(\frac{\epsilon \log^2 n}{\log \log n} \right) = \log \epsilon + 2 \log \log n - \log^{(3)} n.$$

Table 8.3 shows the values for x_ℓ and λ_ℓ substituted for the LUT. For example, with $n = 4$, we have $x_\ell = 2$ and $\lambda_\ell = 3$. Though a 2 address LUT can have 4 locations, our simulation ensured that only $\lambda_\ell = 3$ locations were used in the LUT.

TABLE 8.3: Parameter values for a $\lambda_\ell \times n$ LUT

n	x_ℓ	λ_ℓ
4	2	3
8	3	5
16	3	7
32	4	9
64	4	12
128	4	15
256	5	19

For the configurable decoder with reconfigurable mapping unit (see Figure 8.1(e)), we have $x_r = x_f$, $y_r = y_f$, and $z_r = z_f$ (which is the same as in Table 8.2). However, since the configurable decoder with reconfigurable mapping unit uses a $2^{y_r} \times ny_r$ LUT (Table 8.4) also shows ny_r.

TABLE 8.4: Parameter values for a configurable decoder with reconfigurable mapping unit

n	x_r	y_r	z_r	ny_r	λ_r
4	2	1	3	4	3
8	3	1	6	8	5
16	3	1	10	16	7
32	4	2	18	96	9
64	4	2	33	128	12
128	4	2	61	256	15

The universal version of the configurable decoder with reconfigurable mapping unit sets $2^{y_r} = z_r$ (see Section 5.2). We chose the value of y_r such that $\lambda_r = 2^{y_r} \log z_r = y_r 2^{y_r} = \lambda_f$. Table 8.5 shows the values for this configurable decoder.

TABLE 8.5: Parameter values for a universal configurable decoder with reconfigurable mapping unit

n	x_{r_u}	y_{r_u}	z_{r_u}	ny_{r_u}	λ_{r_u}
4	2	2	3	8	3
8	3	2	4	16	5
16	3	2	4	32	7
32	4	3	6	96	9
64	4	3	6	192	12
128	4	3	8	384	15

We also tested a configurable decoder with fixed mapping unit using the same parameters for y and z as in the universal configurable decoder with reconfigurable mapping unit.

Regression Analysis: The simulations described above were used for the nonlinear regression analysis. For most cases, the data obtained was sufficient to produce steady state results.

Simulation Environment: Figure 8.2 shows the basic structure of the simulation process. We briefly describe its components.

Source Code Development: All decoders were defined in *Verilog Hardware Description Language*, or *Verilog HDL* (for example, see [10])

Functional Testing: A functional verification of the hardware description files took place using the Cadence NC-Verilog tool [8]. This provides a verified template for $n = 16$ and selections of other parameter values. This template was modified as described below.

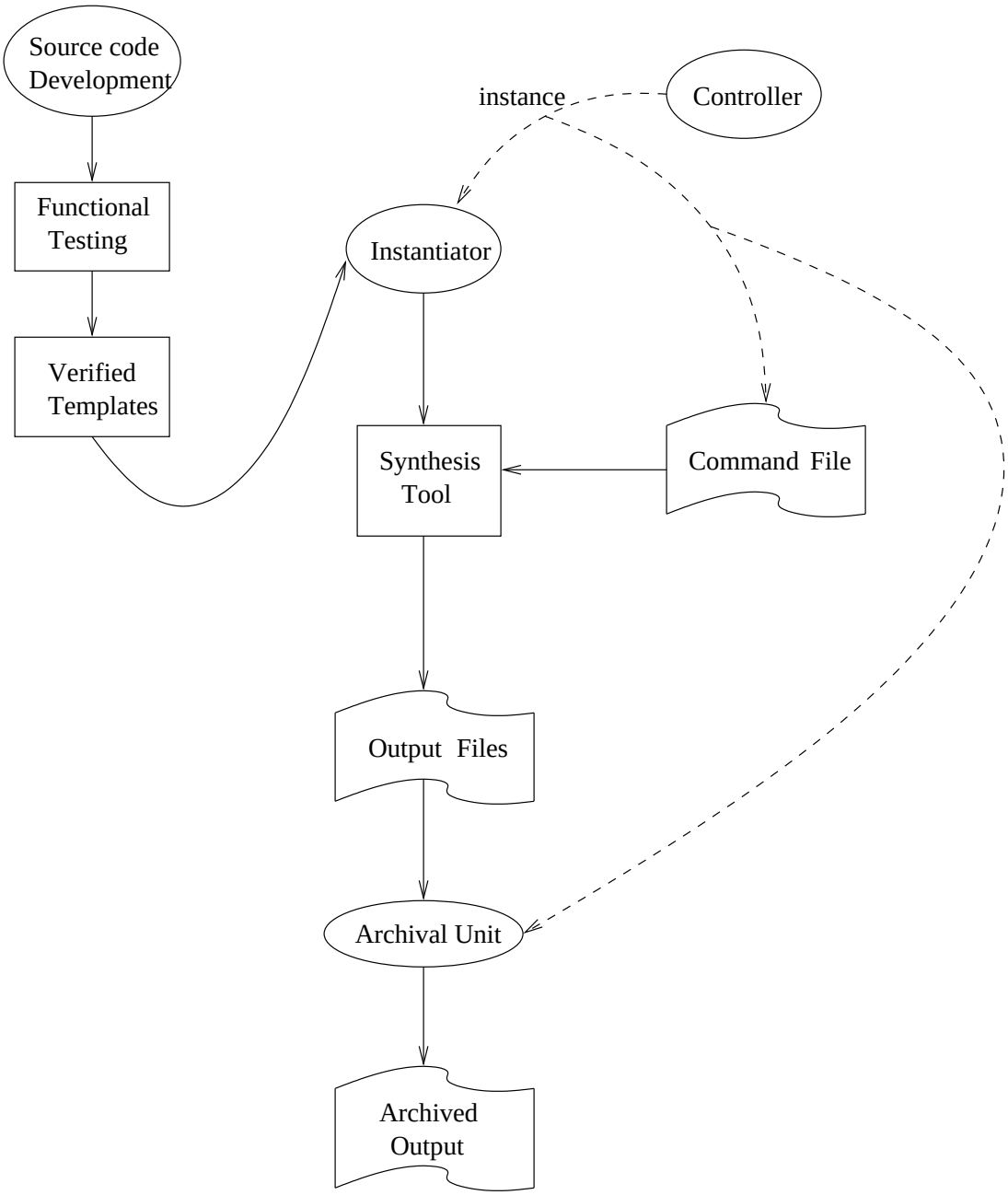


FIGURE 8.2: Simulation process

Instantiator: This UNIX shell script creates a new set of files from a given verified template for a given set of parameters. For example, if a verified template with $n = 16$ uses 16 multiplexers called *MUX_0* ... *MUX_15*, when $n = 32$, we have *MUX_0* ... *MUX_31*, all of which must be separately defined. This script automates this file conversion.

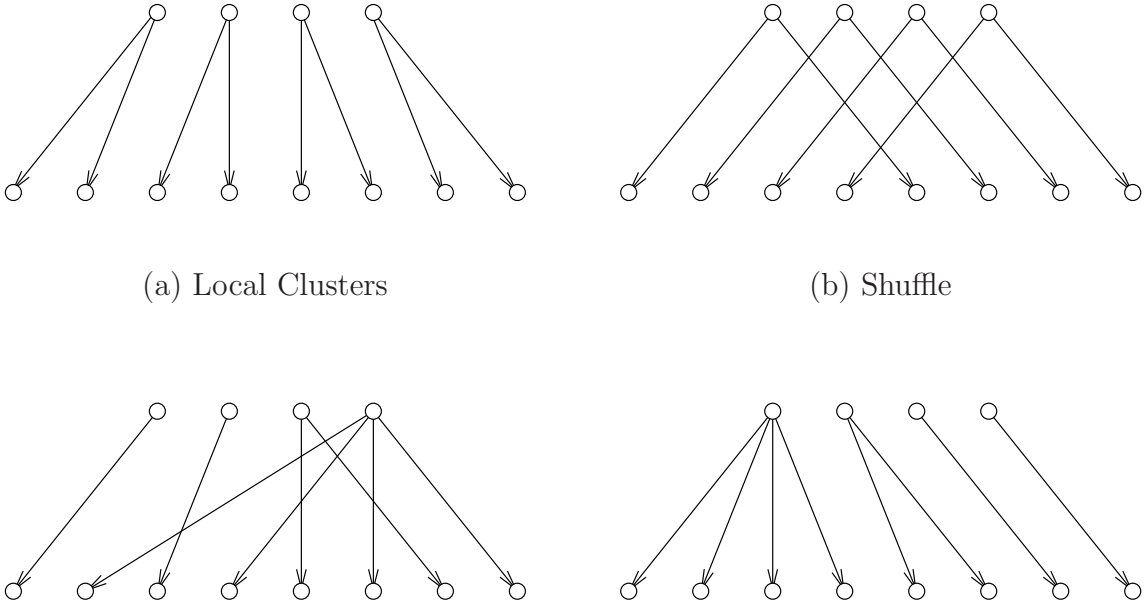
Controller: The set of input parameters is provided to the instantiator by a controller script, which systematically plods through feasible parameters values (described in Tables 8.2–8.5) for the different configurable decoder implementations. These parameter constructions are also used to customize the commands to the synthesis tool and to save the outputs systematically.

Synthesis Tool: The synthesis of the hardware was performed using the Cadence Physically Knowledgeable Synthesis (PKS) tools [7]. Cadence PKS performs a physical mapping of a hardware description to a given process and technology, and using this mapping, derives the overall area, delay, and power consumption of the design in terms of square microns, nanoseconds, and milliwatts, respectively. The area and number of gates would differ primarily in situations where the interconnects dominate the area. In all our designs, the interconnects (wires) occupied an insignificant part of the area. Consequently, the area data is also indicative of the number of gates. In the synthesis of our designs, we used a 0.25 μm process technology library developed by Artisan Components, Inc.[1] For each hardware design, we performed one synthesis optimizing for area and another optimizing for delay. It was found that for our designs, there were no significant differences between the different optimizations (typically, a small number of gates and one or two hundredths of a nanosecond were the differences between the two cases); hence, the results presented here, for all simulations, were optimized for area.

Archival Unit: This script uses the current parameters instance (provided by the controller) to save the simulation output in an appropriately named file.

[1]The technology library used, "demo25," copyright 2000 Artisan Components, Inc.

As the specific internal connections of all mapping units (except the universal mapping unit) are not fixed and determine the functionality of the mapping unit, a variety of connection choices were made that distributed the wirings differently across multiplexers. Figure 8.3 shows some basic connection patterns between source string bits and MUXs. We



(a) Local Clusters

(b) Shuffle

(c) ASCEND

(d) Reduction

FIGURE 8.3: Wire distributions in simulated mapping units.

used these basic schemes and their combinations. While a different distribution might reduce the resulting area slightly; overall, it was found to not make a significant difference.

We note the following assumptions and limitations that occurred during the synthesis of the designs.

1. As we did not have access to a memory generator, all synthesized memory elements (including LUTs) resulted in arrays of sequential elements (flip-flops). Additionally, the synthesis was not able to create a memory element with single port read and write capabilities; thus all memory cells were dual-ported. This results in a substantial increase in the size of the memory generated over what would be expected of traditional implementations (such as SRAM). In Section 8.2 we provide an interpretation of the data that, to an extent, alleviates this concern.

2. The implementation of the fan-in and fan-out of signals was left up to the synthesis tool. For some particular designs and for some particular values of n, the designs could be optimized effectively; in other cases, it was apparent that the fan-out of the signals resulted in drivers with a large delay.

3. The system executing the simulations was unable to synthesize certain designs for $n = 256$ and all designs for $n > 256$. Hence, any trends are derived from data points that extend from $2 \leq \log n \leq (7, 8)$.

However, despite these limitations, this chapter still demonstrates (a) a comparison of the performance of the configurable decoder to the current state of the art on a relatively level playing field and (b) an observation of the trends in that performance, and with some extrapolation, a prediction of future trends with newer technology files.

8.2 Simulations

In this section we present simulation data for the delay, area (raw and adjusted for memory implementation), and power consumption for different modules. The data is categorized by module name and the value of n. Other parameters needed to determine the module (such as x and y for configurable decoders) have been specified in Tables 8.2–8.5. The first set of data is for the following integral decoders.

1. 1-hot decoder (1-Hot)

2. Pure LUT-based decoder (LUT)

3. Configurable decoder with a fixed mapping unit (CDF)

4. Configurable decoder with a reconfigurable mapping unit (CDR)

5. Universal configurable decoder (Univ.)

6. Counterpart of universal decoder with a fixed mapping unit (F-Univ.)

The second set of data is for the bit-slice configurable decoders. As there are two independent variables used in the construction of bit-slice decoders (n and α), the data presented

is (a) the LUT configurations used in the bit-slice decoders (3–6 above), (b) mapping units for a range of n, from which mapping units for a range of $\frac{n}{\alpha}$ are used in the regression analysis, (c) the cost of the shift registers and mod-α counters for a range of n and α, and (d) the extrapolated cost of bit-slice configurable decoders for a range of n and α and for the mapping units presented. Note that (1) we do not compare the bit-slice configurable decoders with a 1-Hot or a LUT, due to the difference in their capabilities and the situations in which a bit-slice configurable decoder would be employed, and (2) due to the complexity of a regression analysis required for the bit-slice configurable decoder, we only present the derived area.

8.2.1 Integral Decoders

TABLE 8.6: Integral decoder delays [ns]

n	1-Hot	LUT	CDF	CDR	Univ.	F-Univ.
4	0.16	0.86	1.23	1.17	0.85	0.79
8	0.26	1.59	1.67	1.64	1.78	1.65
16	0.34	3.51	2.95	2.91	3.67	3.18
32	0.79	3.52	4.14	4.22	4.76	4.01
64	1.16	2.39	3.18	2.97	5.71	4.47
128	1.44	5.2	5.02	5.57	9.08	6.67
256	1.85	12.74	8.23	-	-	4.16

Table 8.6 and Figure 8.4 illustrate the delay of the configurable decoders as compared to the 1-hot decoder and the LUT. Note that the implementations with larger LUTs (the configurable decoders with RMUs and the LUT) have larger delays; this is most likely an effect of the implementation of the LUT as a sequential circuit. The discontinuities, primarily at $n = 64$, are likely due to a technology dependent factor such as fan-in/fan-out.

Table 8.7 and Figure 8.5 illustrate the results compared against a $\lambda \times n$ LUT and the $\log n$ to n 1-hot decoder. As demonstrated, the configurable decoders with fixed mapping units perform very well against the pure-LUT based implementation and, out of all the

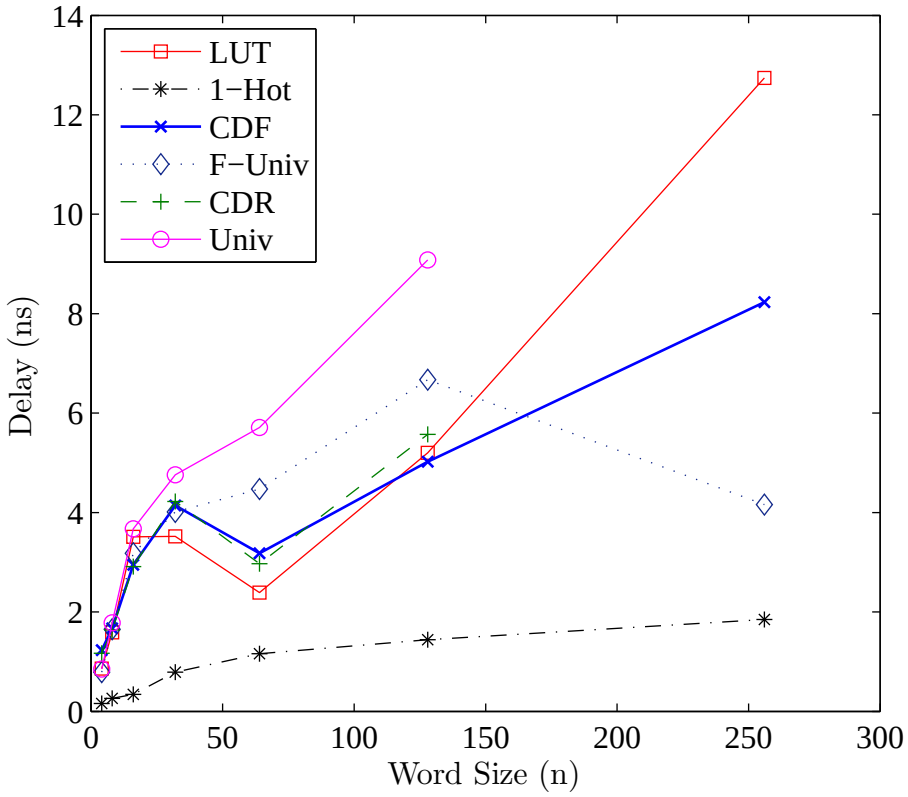


FIGURE 8.4: Integral decoder delays [ns]

configurable decoders, come closest to the area of the 1-hot decoder. Interestingly, the F-Univ. has a lower area than the CDF; this arises from the value of z being $O(n^{\epsilon})$ in the CDF, as the corresponding size of the LUT is large. Additionally, we can note that the CDR begins to outperform the LUT for $n = 128$; however, the Univ. performs worse than the LUT. This is because the universal reconfigurable decoder is only marginally asymptotically better than the LUT, and the constants are quite large. Hence, with the range of data available, the point at which the Univ. becomes better than the LUT is not visible in Figure 8.5.

As stated in Section 8.1, a limitation of this simulation was the lack of a memory generator. We made the following assumption regarding the required area of an SRAM cell in order to predict the trends for a more realistic memory element. In the technology library files used, the flip-flops used for memory cells had a cell area of 27.00 μm^2, while a standard CMOS inverter had a cell area of 2.00 μm^2. Knowing that a standard SRAM cell is com-

TABLE 8.7: Integral decoder areas [μm^2]

n	1-Hot	LUT	CDF	F-Univ.	CDR	Univ.
4	15	387	328	583	1237	486
8	43	1261	1021	1524	2842	801
16	83	3557	2371	3366	5238	1213
32	180	9030	5624	11503	20985	3390
64	319	23400	13126	24833	40678	5565
128	596	60794	30547	54085	107023	11204
256	1142	152427	71551	-	-	21083

posed of six transistors, and that a standard CMOS inverter is two transistors, we divided the area of the sequential elements in all memory blocks in all designs by a factor of 10, that is, we assumed that all sequential elements took up 2.70 μm^2. Figure 8.6 illustrates this recalculated area for all designs tested. As this figure demonstrates, even with a reduction in the cost of sequential elements to a fairly low area, the configurable decoders (with the exception of the Univ.) outperform the LUT. In addition, this brings the area required by the configurable decoders closer to that of the 1-hot decoder.

TABLE 8.8: Integral decoder power consumptions [mW]

n	1-Hot	LUT	CDF	F-Univ.	CDR	Univ.
4	2.178×10^{-5}	1.11×10^{-3}	9.340×10^{-4}	1.320×10^{-3}	1.686×10^{-3}	3.645×10^{-3}
8	6.478×10^{-5}	3.61×10^{-3}	2.867×10^{-3}	2.267×10^{-3}	4.414×10^{-3}	8.915×10^{-3}
16	1.262×10^{-4}	1.167×10^{-2}	6.840×10^{-3}	4.086×10^{-3}	1.850×10^{-2}	3.380×10^{-2}
32	2.177×10^{-4}	7.14×10^{-2}	3.800×10^{-2}	8.689×10^{-3}	0.1407	0.1771
64	3.634×10^{-4}	0.1872	7.660×10^{-2}	1.740×10^{-2}	0.3232	1.3642
128	5.502×10^{-4}	4.2476	0.5013	6.830×10^{-2}	4.3532	16.5308
256	1.091×10^{-3}	55.1795	7.7395	6.190×10^{-2}	-	-

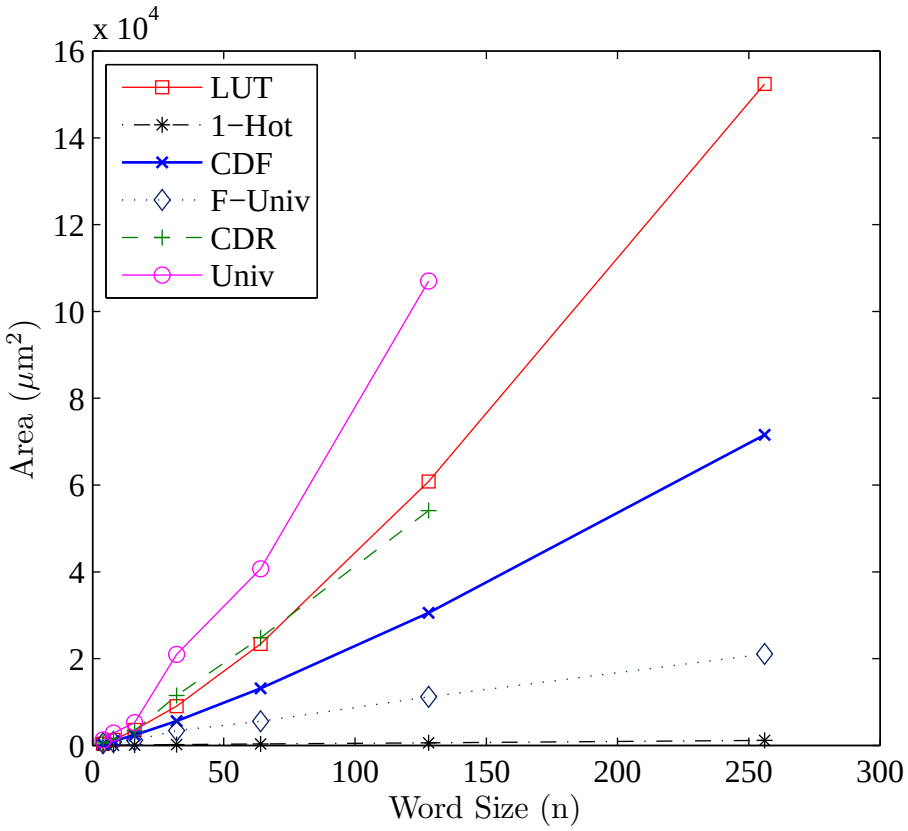


FIGURE 8.5: Integral decoder areas [μm^2]

Finally, Table 8.8 and Figure 8.7 shows the power consumption of the configurable decoders as compared to a LUT and a 1-hot decoder. The simulation provided an estimate of the internal cell power, leakage power, and net power of each design; the data illustrated in Table 8.8 and Figure 8.7 is the sum of those values. Note that a LUT consumes significantly more power for large values of n when compared to our configurable decoders; however, the Univ. appeared to have a higher rate of power consumption as compared to the LUT for $n = 128$.

8.2.2 Bit-slice Decoders

As the size of the LUT in a configurable decoder is not affected by the value of α in a bit-slice implementation, that is, for the λ's shown in Table 8.2, the values of x and z are as shown in Table 8.3. The costs of the LUT for the configurable decoders are given in Table 8.9.

The mapping units used in the bit-slice configurable decoder include an FMU, an FMU with universal decoder parameters (F-Univ.), an RMU, and a universal RMU (Univ.). Ta-

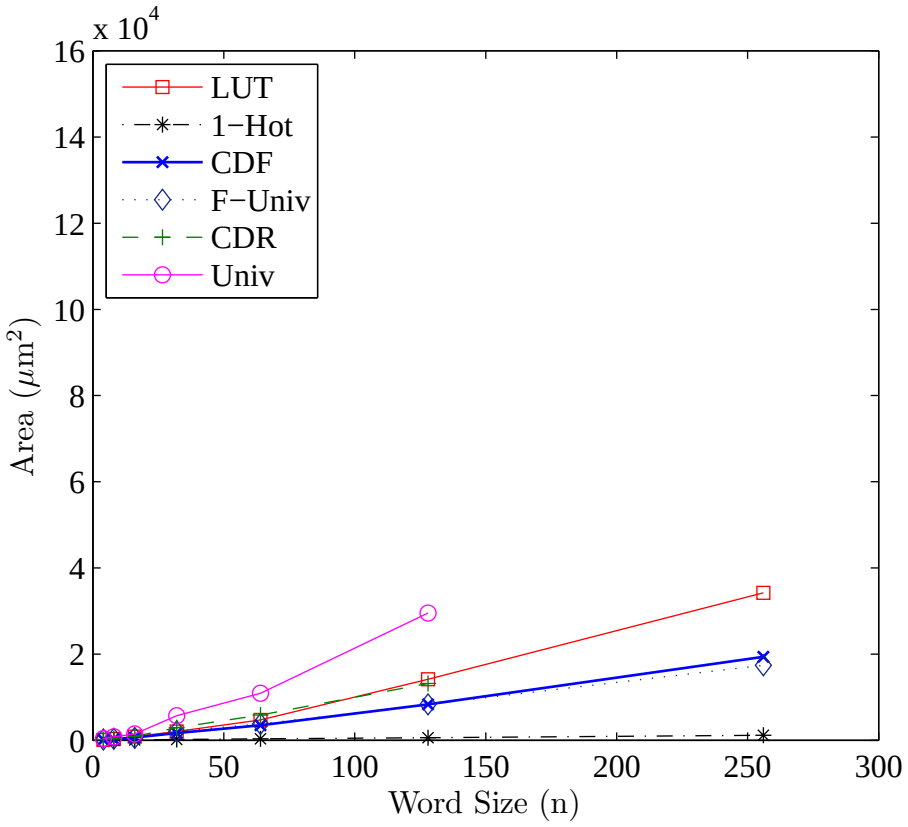


FIGURE 8.6: Integral decoder recalculated areas [μm^2]

ble 8.10 and Figure 8.8 present the results for the area of the different mapping units. Note that the reconfigurable mapping units include the cost of their configuration LUTs. From this data, functions were extrapolated that allowed a prediction of the size of a mapping unit given a value of $\frac{n}{\alpha}$ (the regression analysis follows the method explained in Section 8.3).

Table 8.11 illustrates the area for the mod-α counter and shift registers for the bit-slice configurable decoders for a range of n and α. Note that (a) the large cost for these elements primarily comes from the output shift register (with n registers) and the flip-flops used by the technology library for memory elements and (b) the lack of data for certain points is typically a result of values of $\frac{z}{\alpha} < 1$.

Tables 8.12–8.15 and Figures 8.9–8.12 illustrate the results of the simulation for the bit-slice configurable decoders; that is, the combined area of the LUTs from Table 8.9, the mod-α counter and shift registers from Table 8.11, and the mapping unit area derived from the data of Table 8.10. If we compare the area of a bit-slice CDF with an integral CDF,

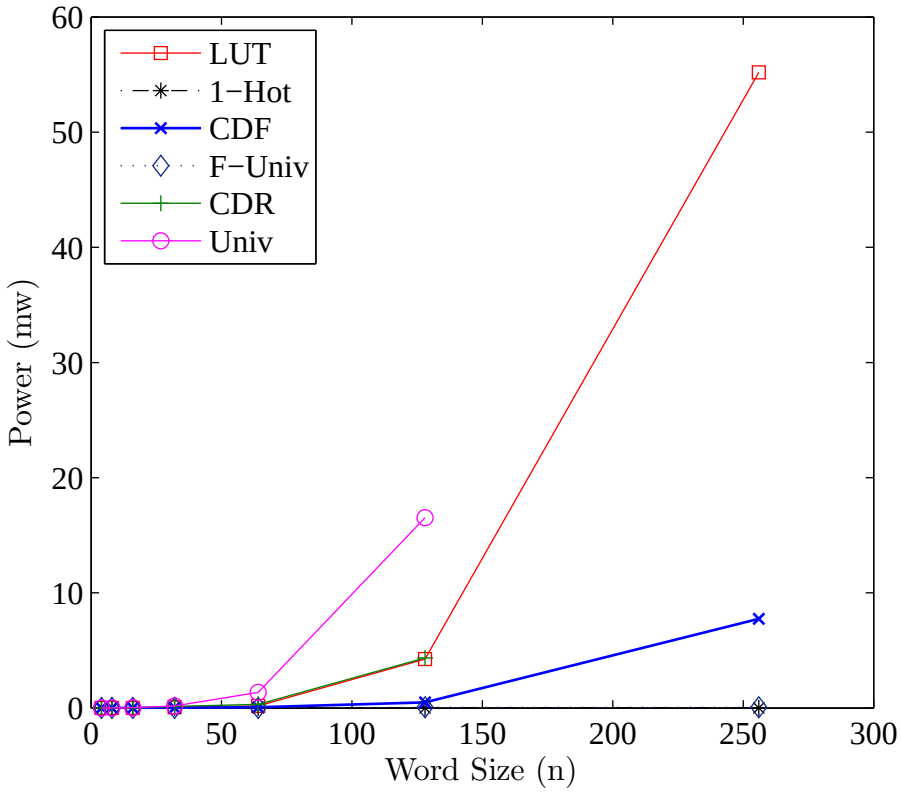


FIGURE 8.7: Integral decoder power consumption [mW]

we can note that for $n = 256$, $\alpha = 32$, a bit-slice CDF has an area of 78393 μm^2, while an integral CDF has an area of 71551 μm^2. This is because the LUT is especially large for the CDF (19×113, from Table 8.2), and the bit-slice CDF imposes an additional 256-bit register. Combined with the flip-flop area penalty of our technology, this results in a construction that is actually more costly than the integral CDF. However, the bit-slice universal decoder, for $n = 128$, $\alpha = 32$, has an area of 21015 μm^2 while an integral universal decoder for the same value of n has an area of 54085 μm^2. This is again because of the size of the preceding LUT; here the size of the LUT is only 19×8 (see Table 8.5). However, we expect the bit-slice configurable decoder to be advantageous where source strings are reduced substantially for a given application. For these cases, the size of the LUT and mapping units becomes smaller.

TABLE 8.9: LUT areas [μm^2] in a bit-slice configurable decoder

n	CDF	CDR	Univ.	F-Univ.
4	296	296	402	402
8	957	957	657	657
16	2243	2231	925	925
32	5144	5152	1758	1758
64	12166	12171	2297	2305
128	28627	28752	3960	4161
256	67711	-	4907	-

TABLE 8.10: Mapping unit areas [μm^2]

n	FMU	RMU	Univ. RMU	FMU (F-Univ.)
4	32	287	835	84
8	64	567	2185	144
16	128	1135	4333	288
32	480	6351	19223	1632
64	960	12662	38373	3268
128	1920	25333	108636	7244
256	3840	-	-	16176

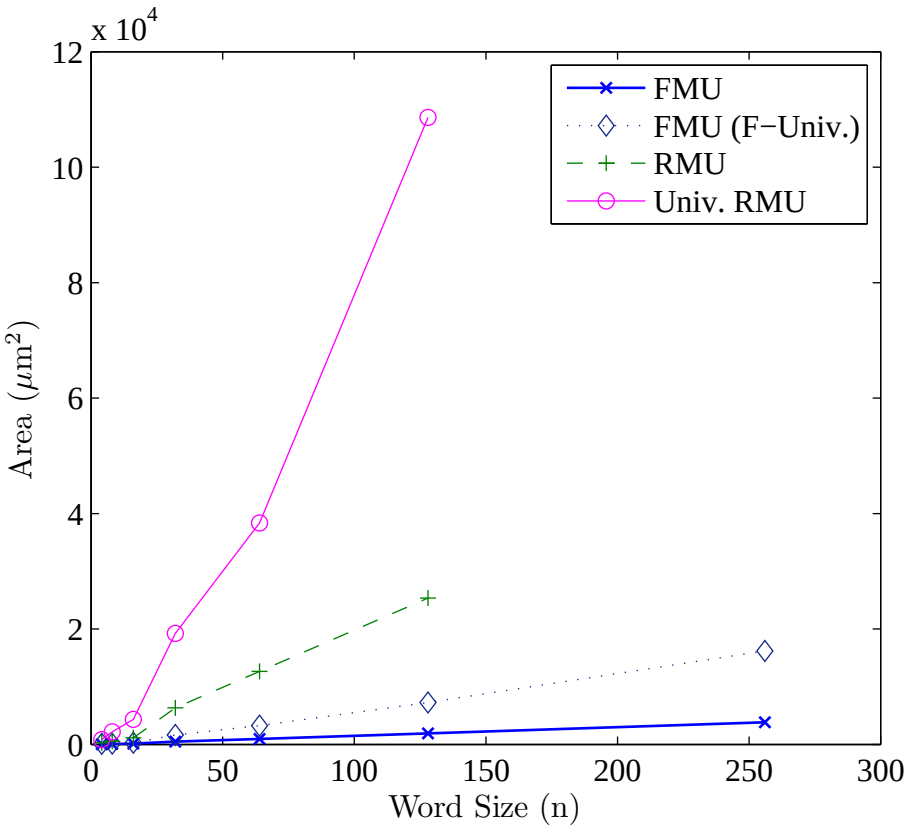


FIGURE 8.8: Mapping unit area (μm^2)

TABLE 8.11: Area (μm^2) for mod-α counter and shift registers, $2 \leq \log n < 256$, $1 \leq \log \alpha < 6$

n / α	2	4	8	16	32
4	424	-	-	-	-
8	817	756	-	-	-
16	1427	1342	1424	-	-
32	2639	2508	2566	-	-
64	4980	4844	4856	5022	-
128	9654	9364	9145	8989	8812
256	18833	18234	18023	18065	17846

TABLE 8.12: Bit-slice CDF area (μm^2)

n / α	2	4	8	16	32
4	737	-	-	-	-
8	1815	1754	-	-	-
16	3762	3677	3759	-	-
32	7983	7852	7910	-	-
64	17572	17436	17448	17614	-
128	39177	38887	38668	38512	34939
256	88414	87815	83188	87646	78393

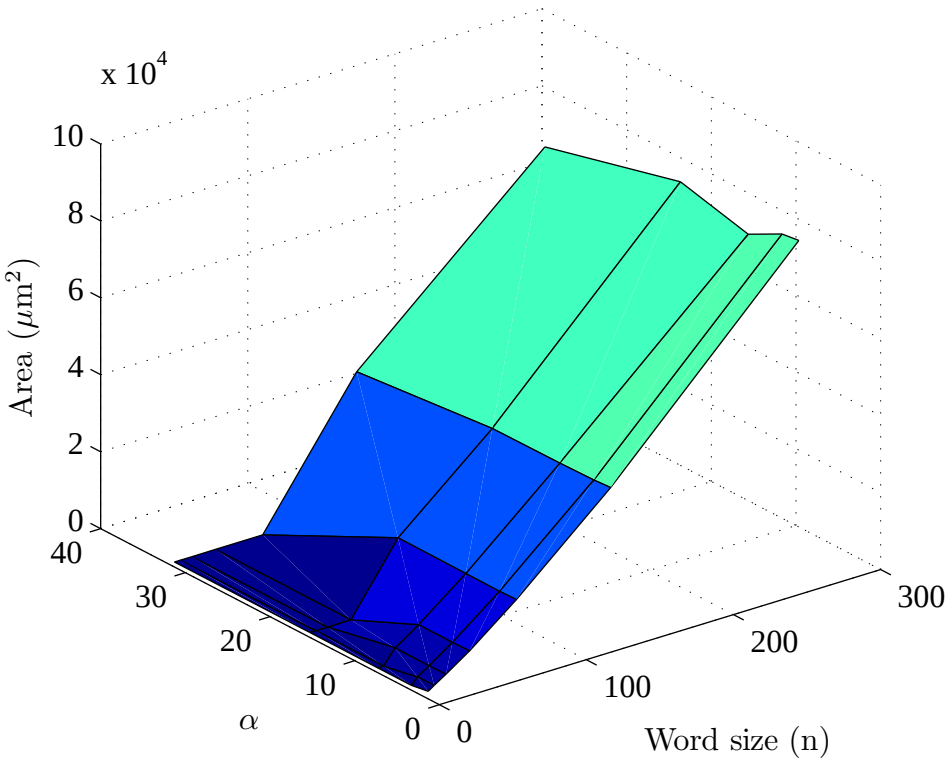


FIGURE 8.9: Bit-slice CDF area [μm^2]

TABLE 8.13: Bit-slice CDR area (μm^2)

n / α	2	4	8	16	32
4	438	-	-	-	-
8	1728	1667	-	-	-
16	4343	4258	4340	-	-
32	10076	9945	10003	-	-
64	22729	22593	22605	22771	-
128	50639	50349	50130	49974	46401

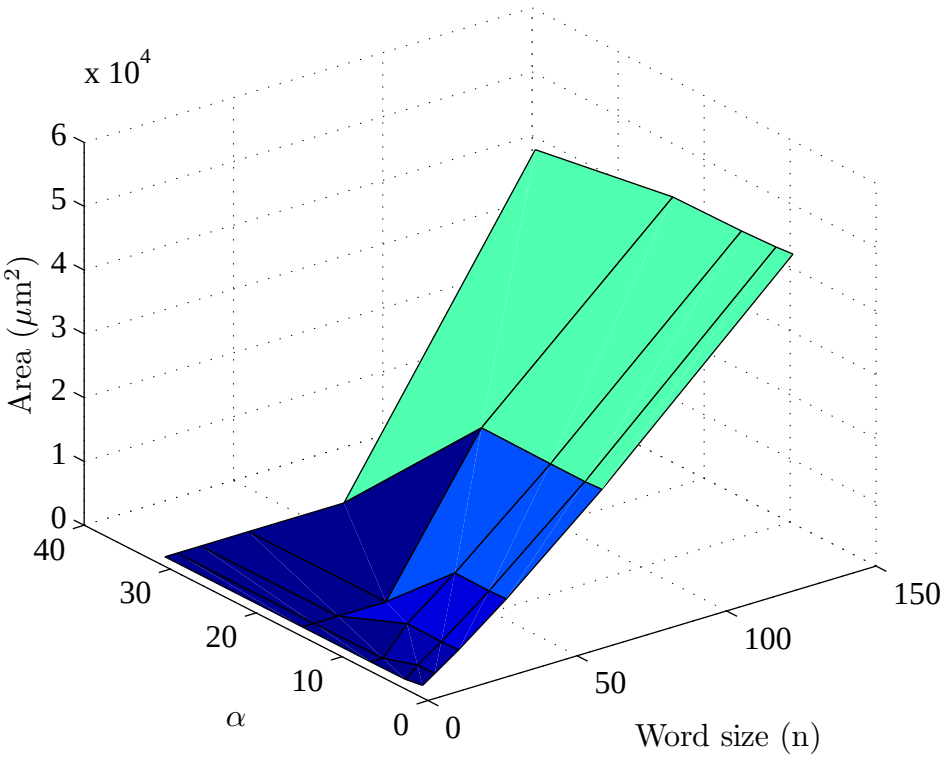


FIGURE 8.10: Bit-slice CDR area [μm^2]

TABLE 8.14: Bit-slice Univ. area (μm^2)

n / α	2	4	8	16	32
4	-	-	-	-	-
8	2415	2354	-	-	-
16	4791	4706	4788	-	-
32	10133	10002	10060	-	-
64	22537	22401	22413	22579	-
128	55044	54754	54535	54379	50806

The lack of data for $n = 4$ arose from a discontinuity in the regression function for the MU

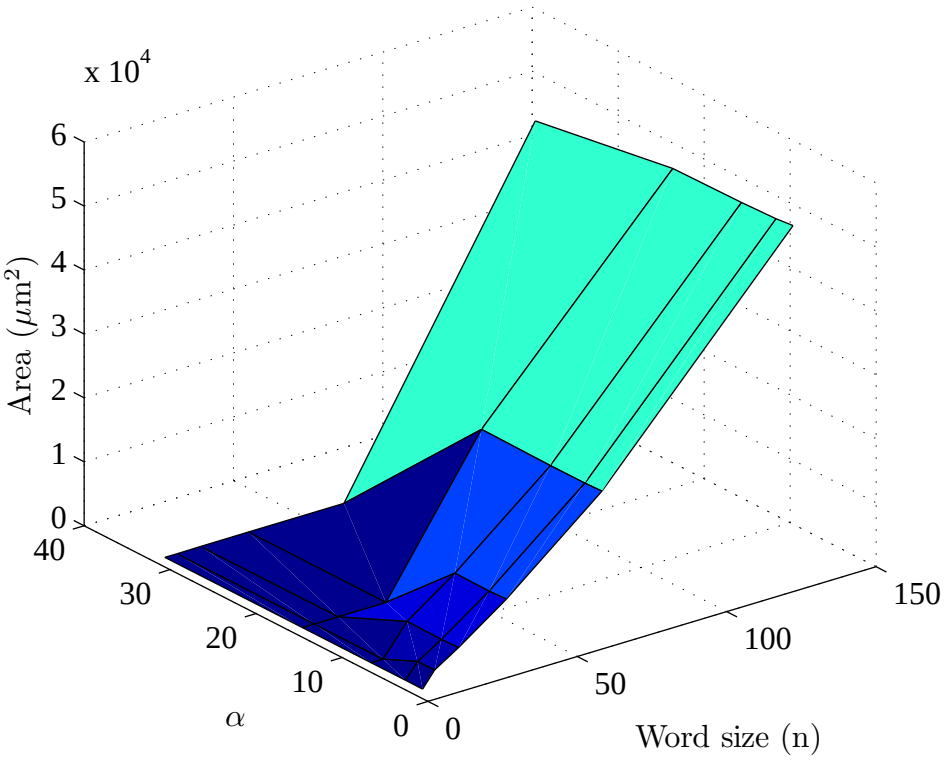


FIGURE 8.11: Bit-slice Univ. area [μm^2]

TABLE 8.15: Bit-slice F-Univ. area (μm^2)

n / α	2	4	8	16	32
4	-	-	-	-	-
8	1497	1436	-	-	-
16	2472	2387	-	-	-
32	4926	4795	4853	-	-
64	8694	8558	8570	-	-
128	16897	16607	16388	16232	-
256	31036	30437	28810	30268	21015

The lack of data for $n = 4$ arose from a discontinuity in the regression function for the MU

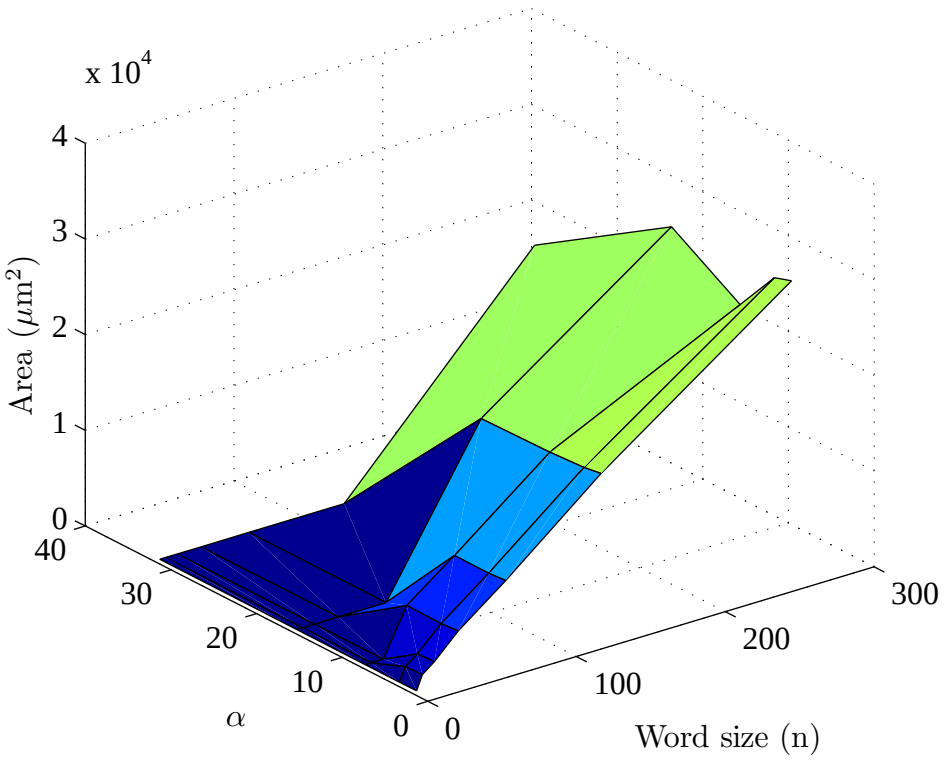


FIGURE 8.12: Bit-slice F-Univ. area [μm^2]

8.3 Regression Analysis Results

In order to determine the values of the constants hidden by the asymptotic notation of the gate cost of the configurable decoders, we performed a nonlinear least squares regression analysis using the Trust Algorithm in Matlab [20]. Let n be a value used for the simulation (for example 4, 8, 16, ...). Let $D(n)$ be a data corresponding to n; for example, if we are considering the area of a CDF, then from Table 8.7, $D(4) = 328$. The aim is to use the data points available to generate a function $f(n)$ that fits the data. For this purpose, the regression has to be supplied a set of functions $f_1(n), f_2(n), \ldots, f_k(n)$ such that $f(n) = \sum_{i=1}^{k} a_i f_i(n)$ would be a likely representation of the function we seek. Moreover, the value of k should be somewhat smaller than the number of data points to get a reasonably good fit. In order to determine whether the function is a good fit, the regression tool minimizes the quantity $\sum_n (D(n) - f(n))^2$. For all our regression analysis, we used $k \leq (4, 5)$ as the number of data points was around 8.

The various modules constructed in this thesis have complex cost function representations. For example, the number of gates in a $2^z \times m$ LUT (see Section 3.2.4) has the form $a_1 m 2^z + a_2 z 2^z + a_3 m + a_4 2^z + a_5 z + a_6$, where $a_1 \ldots a_6$ are constants. Translated to a $\frac{\epsilon \log^2 n}{\log \log n} \times n$ LUT used in the pure LUT-based solution (see Section 8.1) this results in many different functions of n. Our analysis in Section 3.2.4 simply accounts for the fastest growing term and ascertains that the gate cost of this LUT is $\Theta\left(\frac{n \log^2 n}{\log \log n}\right)$. However, other terms may be significant. For this LUT, we use the functions

$$f_1(n) = \frac{n \log^2 n}{\log \log n}, f_2(n) = \log^2 n, f_3(n) = 1, f_4(n) = n, f_5(n) = \frac{\log^2 n}{\log \log n}.$$

Our choice of these 5 functions from among the 8 that make up an analytical formula for the cost is based on what we believe would be the most significant terms. We always include the fastest growing term and the constant function. We recognize that very slow growing functions such as $\log \log n$ are nearly constant over the range of values of n considered. Therefore, we select only one among a set of functions such as n, $n \log \log n$, $\frac{n}{\log \log n}$, as we

do not expect a significantly different nonconstant contribution from these. Note that for a 1-hot decoder, as n becomes large, many of the AND gates become redundant and are eliminated. This technique (known as predecoding, see [18, 26]) reduces the number of gates by a constant factor, resulting in a small coefficient for the asymptotic gate cost function $n \log n$.

TABLE 8.16: Functions used in regression analysis for each module

Module	$f_1(n)$	$f_2(n)$	$f_3(n)$	$f_4(n)$	$f_5(n)$
1-Hot	$n \log n$	1	-	-	-
LUT	$\frac{n \log^2 n}{\log \log n}$	$\log^2 n$	1	n	$\frac{\log^2 n}{\log \log n}$
CDF	$n \log n$	$n^{1-\epsilon} \log^2 n$	$\log n$	n^{ϵ}	1
CDR	$n \log n$	$n^{1-\epsilon} \log^2 n$	$\log n$	n^{ϵ}	1
Univ.	$\frac{n \log^2 n}{\log \log n}$	$\frac{\log^4 n}{(\log \log n)^3}$	$\log \log n$	1	-
F-Univ.	$\frac{n \log^2 n}{\log \log n}$	$\frac{\log^4 n}{(\log \log n)^3}$	$\log \log n$	1	$\log^2 n$

ϵ was approximated to 0.85 for all simulations

Table 8.16 shows the values of the functions $f_1 \ldots f_{4,5}$ used for the different modules and Table 8.17 shows the constants obtained from the regression analysis. This table also shows the "relative error" (which equals the average value of the residual error $\frac{|D(n) - f(n)|}{f(n)}$ over all n).

TABLE 8.17: Constants found from regression analysis for each module

Module	a_1	a_2	a_3	a_4	a_5	Error
1-Hot	0.543	61.8	-	-	-	0.263
LUT	31.1	-26.1	291	-67.5	59.6	0.077
CDF	51.8	55.9	279	-170	13.86	0.0098
CDR	27.9	180	-1919	161	2471	0.086
Univ.	46.04	3.02	325	170	-	0.097
F-Univ	2.23	78.61	-100	-391	-6.67	0.0817

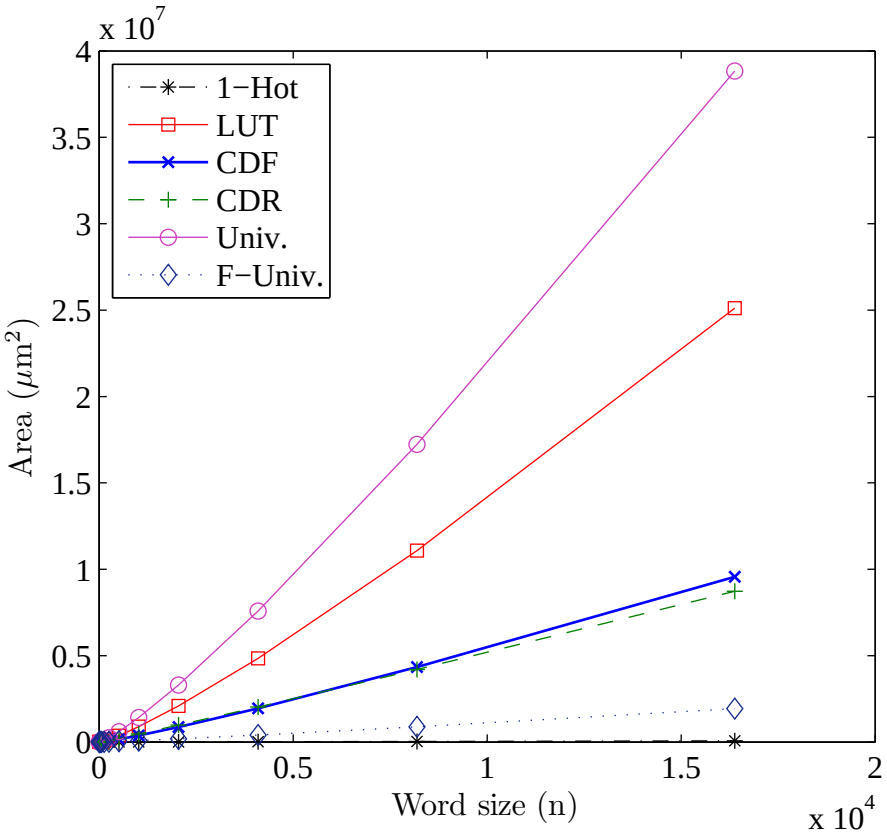


FIGURE 8.13: Integral decoder expected area (μm^2) under regression analysis

The functions outlined above are illustrated in Figure 8.13. While most of the trends are as expected, there are some interesting cases to note. First, at around $n = 8192$, the CDR begins to outperform the CDF. This clearly should not be the case, as the CDR contains all elements of the CDF as well as an additional configuration LUT. Additionally, the functions derived fail to demonstrate the asymptotic cost of the Univ. decoder, as for very large values of n the Univ. decoder outperforms the LUT. Regardless of these inconsistencies, the functions derived provide an indication as to the general trend of the configurable decoders; as expected, our configurable decoders (with the exception of the Univ. decoder, as noted) consistently outperform the pure LUT-based configurable decoder.

Chapter 9
Parallel Configurable Decoder

In this chapter we introduce a variant on the configurable decoder, a parallel configurable decoder ($CD(x,z,y,n,\alpha,P)$), that utilizes a merge operation (such as an associative Boolean operation) to combine the outputs of two or more configurable decoders. The parameter P denotes the number of configurable decoders connected in parallel in $CD(x,z,y,n,\alpha,P)$. This parallel configurable decoder is an interesting case that can produce sets of subsets of $\mathcal{Z}_n$ not easily produced by the configurable decoders previously presented.

9.1 An Illustrative Example

We begin our discussion of the parallel configurable decoder through the set of 1-hot subsets, which is not easily produced by the configurable decoders of Chapter 6 but can be produced rather easily using a parallel variant. We first consider two sets $\mathcal{S}_0$, $\mathcal{S}_1$ of subsets of $\mathcal{Z}_n$. Assume an integer m that divides n so that $n = km$ for some integer $k \geq 1$. Then $\mathcal{Z}_n = \{0, 1, \ldots, m-1, m, \ldots, 2m-1, \ldots, im, \ldots, (i+1)m-1, \ldots, (k-1)m, \ldots, km-1\}$. For $0 \leq i < m$ and $0 \leq j < \frac{n}{m}$, let

$$q_{i,0} = \{i + \ell : 0 \leq \ell < k\}$$

and let

$$q_{j,1} = \{jm + \ell : 0 \leq \ell < m\}.$$

Clearly, $q_{i,0}$ and $q_{i,1}$ are subsets of $\mathcal{Z}_n$. Table 9.1 illustrates the subsets for $n = 20$ and $m = 4$.

Let $\mathcal{S}_0 = \{q_{i,0} : 0 \leq i < m\}$ and $\mathcal{S}_1 = \{q_{j,1} : 0 \leq j < \frac{n}{m}\}$. It is easy to verify that $\mathcal{S}_0$ and $\mathcal{S}_1$ induce partitions $\pi_0 = \{q_{i,0} : 0 \leq i < m\}$ and $\pi_1 = \{q_{j,1} : 0 \leq j < \frac{n}{m}\}$. So, for $z = m = \frac{n}{m}$, two z-partitions of n can generate these subsets in a configurable decoder of the form shown

TABLE 9.1: Subsets $q_{i,0}$ and $q_{i,1}$ for $n = 20$ and $m = 4$

$q_{i,0}$	n-bit string																			
$q_{0,0}$	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
$q_{1,0}$	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
$q_{2,0}$	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
$q_{3,0}$	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
$q_{j,1}$	n-bit string																			
$q_{0,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$q_{1,1}$	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
$q_{2,1}$	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
$q_{3,1}$	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
$q_{4,1}$	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

in Chapter 6 (see Theorem 6.3, page 79). Put differently, each subset of $\mathcal{S}_0$ and $\mathcal{S}_1$ can be independently generated by different configurable decoders using just one partition each.

Lemma 9.1 *For all* $0 \leq i, j < m$ *and* $0 \leq j < \frac{n}{m}$,

$$q_{i,0} \cap q_{j,1} = \{jm + i\}.$$

Proof: Consider $0 \leq x < n$. If $x \in q_{i,0} \cap q_{j,1}$, then there exists integers $0 \leq \ell < \frac{n}{m}$ and $0 \leq \ell' < m$ such that $x = i + \ell m = jm + \ell'$. This implies that $(\ell - 1)m + (i - \ell') = 0$. Without loss of generality, let $\ell \geq j$. Clearly, $i - \ell > -m$. Then, for $(\ell - j)m + (i - \ell')$ to be 0, $\ell = j$ and $i = \ell'$. So, $x = i + jm$. Also, $i = x \bmod m$ and $j = x \bmod \frac{n}{m}$ implies that i and j are unique for a given x. Thus, $q_{i,0} \cap q_{j,1} = \{i + jm\}$. ■

Corollary 9.1 *For each* $x \in \mathcal{Z}_n$, *there exists unique values* $0 \leq i < m$ *and* $0 \leq j < \frac{n}{m}$ *such that* $x \in q_{i,0} \cap q_{j,1}$.

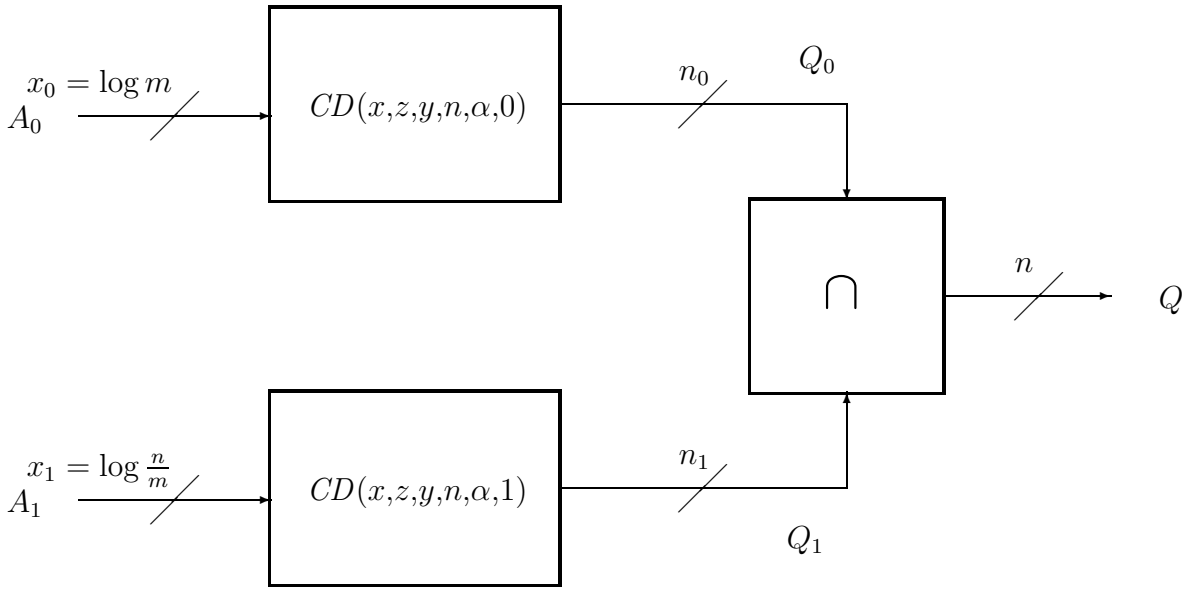


FIGURE 9.1: A parallel configurable decoder that generates the 1-hot subset of $\mathcal{Z}_n$

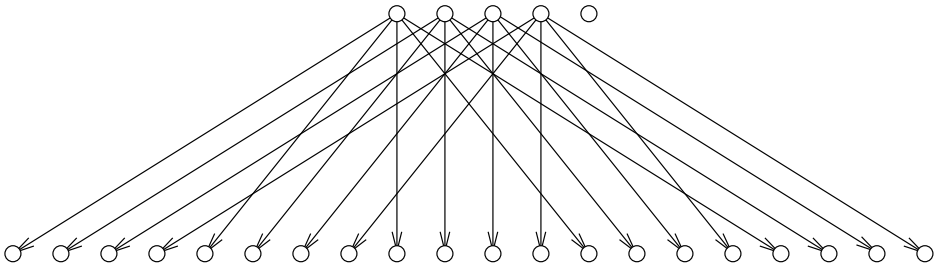
Proof: Since $i = x \bmod m$ and $j = x \bmod \frac{n}{m}$ are unique for a given x by Lemma 9.1, $x \in q_{i,0} \cap q_{j,1}$. ■

As a direct consequence of Lemma 9.1 and Corollary 9.1, we have the following result.

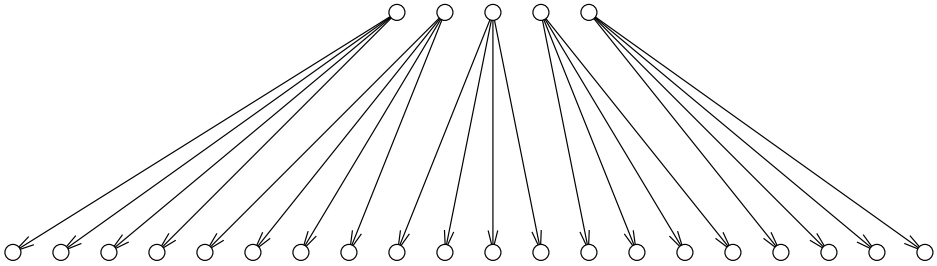
Theorem 9.1 $\mathcal{S} = \{q_{i,0} \cap q_{j,1} : 0 \leq i < m \text{ and } 0 \leq j < \frac{n}{m}\}$ *is the set of 1-hot subsets.* ■

A simple method to generate the 1-hot subsets is illustrated in Figure 9.1. If $m = \sqrt{n}$, then both m and $\frac{n}{m}$ form feasible values for the input for a mapping unit; that is, $z = m = \frac{n}{m}$. We do not need a y input as only 1 partition is used (a y input would allow additional subsets to be generated from additional partitions however). Thus, for the configurable decoders, $x_0 = \log m = \log \sqrt{n} = \log \frac{n}{m} = x_1$ and $z_0 = m = \sqrt{n} = \frac{n}{m} = z_1$ and $y_0 = y_1 = 0$. Clearly, $n_0 = n_1 = n$. Both configurable decoders use a single partition, hardwired into their respective mapping units (see Figure 9.2).

The cost of each configurable decoder is the cost of a $\sqrt{n} \times \sqrt{n}$ LUT with a $CD(\frac{1}{2}\log n, \sqrt{n}, 0, n, 0, 1)$ which is $\Theta(n)$. Clearly, increasing y_0 and y_1 to any constant will increase the number of subsets produced without altering the $\Theta(n)$ gate cost. It is easy to verify that two smaller $\log \sqrt{n}$ to $\sqrt{n}$ 1-hot decoders arranged as shown in this example will also produce a larger $\log n$ to



Hardwired partition for $\mathcal{S}_0$



Hardwired partition for $\mathcal{S}_1$

FIGURE 9.2: Hardwired partitions in the parallel configurable decoder generating the 1-hot subset of $\mathcal{Z}_n$

n 1-hot decoder with $O(n)$ cost. However, our approach offers room for additional partitions and hence additional subsets (within the same cost) and considerably higher flexibility.

9.2 General Observations

In general, a P-element $CD(x,z,y,n,\alpha,P)$ (see Figure 9.3) uses P configurable decoders, CD_0, CD_1, ..., CD_{P-1} in parallel where CD_i is a $CD(x_i,z_i,y_i,n_i,\alpha_i,i)$. Two CDs, say CD_i and CD_j, may use the same input bit for their LUT; that is, the set of x_i bits to CD_i and the set of x_j bits to CD_j could have common bits. Therefore, $\sum_{i=0}^{P-1} x_i \geq x$, as each input bit is assumed to be used at least once. We also have $x_i \leq x$. Similarly, $y_i \leq y$, $\sum_{i=0}^{P-1} y_i \geq y$, $n_i = n$ and $\sum_{i=0}^{P-1} n_i \geq n$.

The merge unit could perform functions ranging from set operations (where $n_i = n$, for all i) to simply rearranging bits (when $\sum_{i=0}^{P-1} n_i = n$). The (optional) control allows it to select from a range of options.

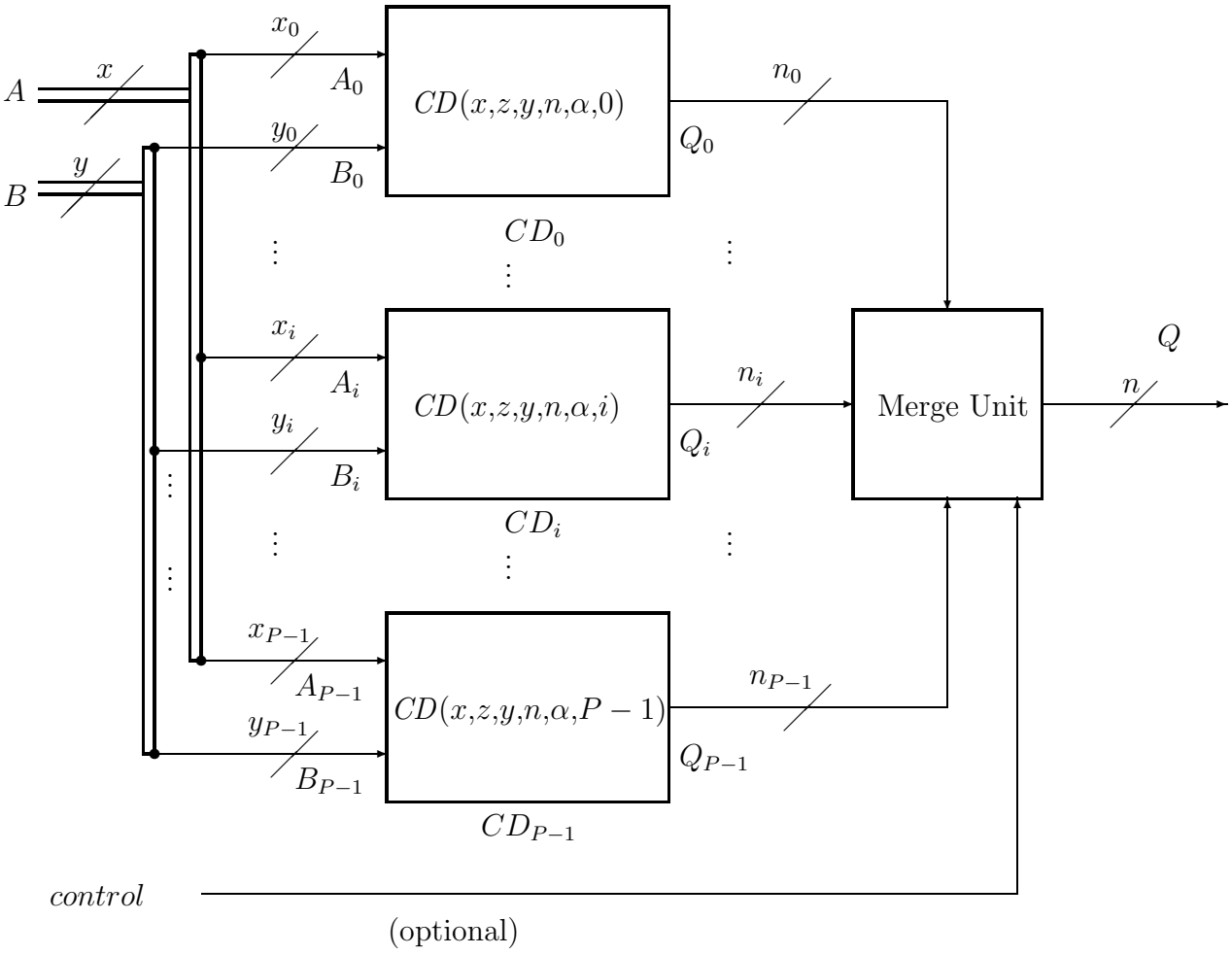


FIGURE 9.3: A parallel configurable decoder $CD(x,z,y,n,\alpha,P)$

Let CD_i have a delay of D_i and a gate cost of G_i. If D_M and G_M are the delay and gate costs of the merge unit, then the delay D and gate cost G of the parallel configurable decoder $CD(x,z,y,n,\alpha,P)$ is

$$D = \max(D_i) + D_M + O(\log P)$$

$$G = \sum_{i=0}^{P-1}(G_i) + G_M + O(P(x+y)).$$

If the merge unit uses simple associative set operations (such as Union, Intersection, Ex-OR) that correspond to bit-wise logical operations, than $D_M = O(\log P)$ and $G_M = O(nP)$.

Since $x + y \leq n$, the overall cost and delay for this structure is

$$D = \max(D_i) + O(\log P)$$

$$G = \sum_{i=0}^{P-1}(G_i) + nP.$$

Clearly, each CD_i can produce its own independent set of n_i-bit outputs. The manner in which these outputs combine depends on the merge unit. For example, let each CD_i produce an n-bit output (that is, a subset of $\mathcal{Z}_n$) and let $\mathcal{S}_i$ be the independent set of subsets produced by CD_i. Let the merge operations be $\circ$, an associative set operation with identity S_0 (that is, for any set S, $S \circ S_0 = S_0 \circ S = S$; Intersection, Union, and Ex-OR represent such an operation with $\mathcal{Z}_n$, $\emptyset$, and $\emptyset$, respectively as the identities). If each CD_i also produces S_0, then the whole configurable decoder $CD(x,z,y,n,\alpha,P)$ produces an independent set that includes $\bigcup_{i=0}^{P-1} \mathcal{S}_i$.

For example, an element $S \in \mathcal{S}_0$ can be produced as $S \circ \underbrace{S_0 \circ S_0 \circ \ldots \circ S_0}_{P-1 \texttt{ times}}$. Clearly, the $CD(x,z,y,n,\alpha,P)$ produces many more dependent subsets.

Chapter 10
Conclusions

In this thesis, we have addressed the pin limitation constraint in IC chips (particularly FPGAs) by providing a fast, flexible, and scalable configurable decoder that bridges the gap between the inexpensive, but inflexible, fixed decoders and the flexible, but expensive, pure LUT-based configurable decoders. As demonstrated in Chapter 6, for a fixed gate cost of G and when $\frac{G}{n}$ is polylogarithmically bounded in n, we outperform the LUT by producing an $\Omega\left(\frac{\log n}{\log\log n}\right)$ factor more independent subsets than the pure LUT-based configurable decoder and significantly more dependent subsets. If $\frac{G}{n}$ is not polylogarithmically bounded in n, we still produce the same order of independent subsets as the pure LUT-based configurable decoder, but continue to provide significantly more dependent subsets not producible by the LUT solution. The contributions of this work can be summarized as follows.

We demonstrated an interesting fixed decoder (called the mapping unit) that uses multicasts as a way of expanding information from z-bits to n-bits. We formally represented these multicasts as ordered partitions of an n-set. Bounds on its capabilities were derived, including the minimum number of independent subsets producible from a mapping unit decoder. We presented a method to produce the maximum possible number of dependent subsets.

Several realizations of the mapping unit were presented (fixed, reconfigurable, bit-slice) that offered various trade-offs between speed, cost, and number of independent subsets. The various mapping unit realizations are melded with the flexibility afforded by a LUT to generate a range of configurable decoders. The functionality of the mapping units allows the cost of the LUTs to be lowered, allowing a solution that has a low gate cost, low delay, but high degree of flexibility.

We applied our results to subsets generated by some well-known classes of communication patterns (binary tree based reduction, ASCEND/DESCEND communications, and 1-hot subsets). We presented extensive simulation results for our designs. The simulation data was used to predict the constants hidden by the asymptotic notation and the future cost trends

for large values of n. These trends suggest that our method will continue to outperform the pure LUT-based solution. We also introduced a generalization of the configurable decoder, a parallel configurable decoder, and made some observations for it. We show its utility by demonstrating how certain sets of subsets that are difficult for our original design can be effectively produced on the generalization.

We now highlight some other ideas and variants that were explored in the course of this research. While these designs did not result in cost-effective solutions, we present them here as a means to guide future research in this area.

10.1 Other Configurable Decoder Variants

The other variants explored during the course of this research were (1) a serial configurable decoder and (2) a recursive bit-slice configurable decoder. These variants were discarded as they did not provide any benefit over the designs included in Chapter 6. We provide some observations about their limitations here.

A Serial Configurable Decoder: In a serial configurable decoder, shown in Figure 10.1,

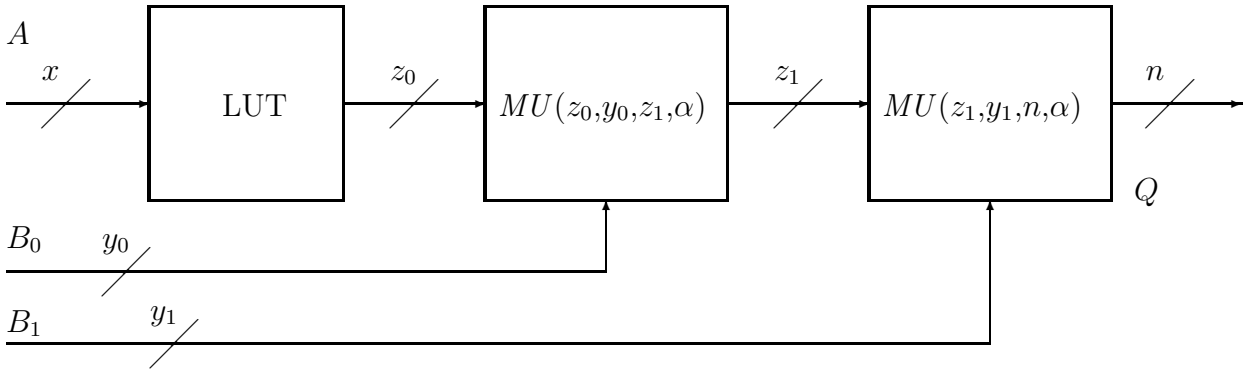


FIGURE 10.1: A serial configurable decoder variant

two or more mapping units are cascaded to construct the subsets of $\mathcal{Z}_n$ (here we will restrict ourselves to examining only two mapping units; extrapolating these results to more than two mapping units would not be difficult). By the definition of a mapping unit decoder, $x \ll z_0 \ll z_1 \ll n$. Note that the independent subsets produced by the second mapping

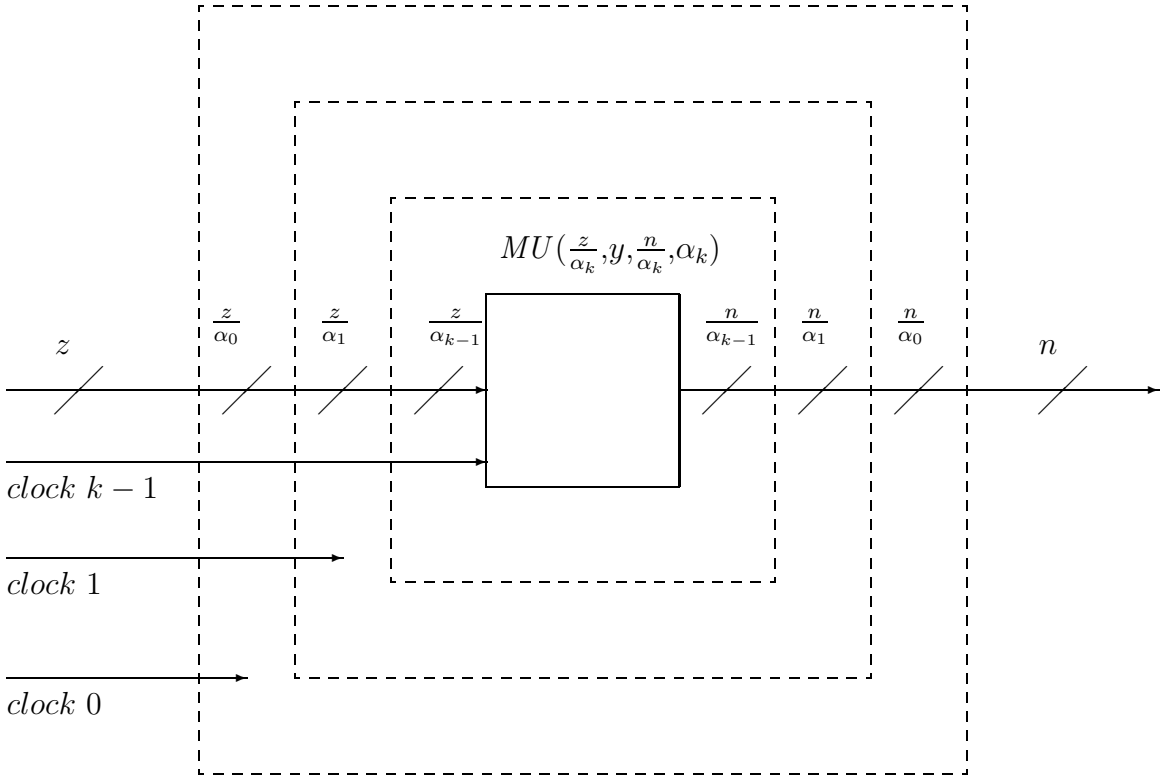


FIGURE 10.2: A conceptual view of a recursive bit-slice configurable decoder. Note that $\alpha_i = \alpha_0 \alpha_1 \ldots \alpha_{i-1}$.

unit are dependent on what is provided to it, that is, the range of values of z_1, which is in turn dependent on the number of independent subsets produced by the first mapping unit. Thus, since the first mapping unit in Figure 10.1 can produce $2^{y_0} \lfloor \log z_0 \rfloor$ independent subsets, where z_0 is a relatively small value, a single LUT can usually subsume both the LUT and the first mapping unit in the serial variant, and be within the gate cost of the second mapping unit and provide more independent subsets.

A Recursive Bit-Slice Configurable Decoder: In a recursive bit-slice configurable decoder, illustrated in Figure 10.2, two or more bit-slice mapping units are nested within each other, such that an input to the first bit-slice configurable decoder is broken down by a factor of α_0, then broken down by a factor of α_1, and so on, until it reaches the lowest level mapping unit. It is then reconstructed to an n-bit output. However, this is not a worthwhile construction, as the large number of shift registers and multiple clocks result in a complex

construction, and the linear (with α) reduction of cost does not provide any benefit that a single bit-slice decoder does not.

10.2 Future Directions

While this thesis has demonstrated a measurable performance gain over pure LUT-based configurable decoders, there is a rich variety of future directions that can be explored in this area.

Mapping Units: The mapping units presented in this thesis are one manner of expanding the output of a smaller LUT to the n-bit output. In fact, any inexpensive z to n decoder will do. Are there other approaches to constructing a decoder that acts as a mapping unit? In addition to this, our realizations of the mapping unit represent several ways of constructing a multicasting module; are there other ways of realizing this operation?

Parallel Configurable Decoders: The initial investigation of the parallel configurable decoder (Chapter 9) has shown promise. Future directions in this area include a deeper exploration of the number of and the types of subsets produced by any merge unit, by a merge unit implementing simple set operations, and a range of different types of merge units for different operations.

Applications: The configurable decoder, while presented for a reconfigurable system, is a more general technique for alleviating the pin limitation problem. What other applications could benefit from this work? We identify two such applications below.

Sensor Networks: A configurable decoder (and a reverse encoder) can serve to reduce the number of bits transmitted between sensor nodes without requiring a drastic redesign of the sensor nodes.

External Power Controllers: The configurable decoder works to select a subset. This can be used by a smart agent (perhaps a chip) that observes data from a collection

of chips and issues commands to selectively power-down portions of these chips. A sharp focused selection (such as that afforded by the configurable decoder) could be useful here.

Bibliography

[1] A. Ali and R. Vaidyanathan, "Exact Bounds on Running ASCEND/DESCEND and FAN-IN Algorithms on Synchronous Multiple Bus Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 783–790, August 1996.

[2] Atmel Corp., "AT6000 Series Configuration," Configuration Guide, 1997.

[3] J. Babb, R. Tessier, and A. Agarwal, "Virtual wires: overcoming pin limitations in FPGA-based logic emulators," *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, April 1993, pp. 142–151.

[4] K. Bondalapati and V. K. Prasanna, "Reconfigurable Computing Systems," *Proc. of the IEEE*, vol. 90, no. 7, July 2002, pp. 1201–1217.

[5] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial," *IEEE Design and Test of Computers*, vol. 13, 1996, pp. 42–57.

[6] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*, McGraw-Hill Companies, Inc., Boston, Massachusetts, 2000.

[7] PKS User Guide, Product Version 5.0, May 2002.

[8] Cadence NC-Verilog Simulator Help, Product Version 5.4, November 2004.

[9] P. Chow, S. Ong Seo, J. Rose, K. Chung, G. Paez-Monzon, and I. Rahardja, "The Design of an SRAM-Based Field-Prorammable Gate Array - Part I: Architecture," *IEEE Transactions on VLSI Systems*, Vol. 7, No. 2, June 1999, pp. 191-197.

[10] M. D. Ciletti, *Modeling, Synthesis, and Rapid Prototyping with the Verilog HDL*, Prentice-Hall, New Jersey, 1999.

[11] H. P. Dharmasena and R. Vaidyanathan, "Lower Bounds on the Loading of Multiple Bus Networks for Binary Tree Algorithms," *IEEE Transactions on Computers*, Vol. 53, No. 12, December 2004, pp. 1535–1546.

[12] H. M. E. El-Boghdadi, "On Implementing Dynamically Reconfigurable Architecture", Ph.D. dissertation, Dept. of Electrical and Computer Eng., Louisiana State University, 2003.

[13] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. J. Wirthlin, "Dynamic Reconfiguration for Management of Radiation-Induced Faults in FPGAs," *Proc. Reconfigurable Architectures Workshop, Int. Parallel and Distributed Processing Symp.*, 2004.

[14] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd. Ed., Morgan Kauffman, San Francisco, CA, 2003.

[15] Intel Corporation, *Microprocessor Quick Reference Guide*, http://www.intel.com/pressroom/kits/quickreffam.htm, 2003.

[16] J. JáJá, *An Introduction to Parallel Algorithms*, Edison Wesley, Reading, MA, 1992.

[17] C. L. Liu, *Elements of Discrete Mathematics*, 2nd. Edition, McGraw-Hill, Inc., New York, 1985.

[18] R. Lyon and R. Schediwy, "CMOS static memory with a new four-transistor memory cell," *Proc. Advanced Research in VLSI*, March 1987, pp. 111–132.

[19] P. Mal, J. F. Cantin, and F. R. Beyette, "The Circuit Designs of an SRAM Based Look-Up Table for High Performance FPGA Architecture," *The 2002 45th Midwest Symposium on Circuits and Systems*, vol. 3, August 2002, pp. 227–230.

[20] The MathWorks, "Curve Fitting Toolbox User's Guide," Version 1.0,
available at: http://www.mathworks.com/access/helpdesk/help/pdf_doc/curvefit/curvefit.pdf

[21] R. Sidhu, S. Wadhwa, A. Mei, and V.K. Prasanna, "A Self-Reconfigurable Gate Array Architecture," *Intl. Conf. on Field Programmable Logic and Applications*, 2000, Springer Verlag Lecture Notes in Comupter Sc., vol. 1896, pp. 106–120.

[22] R. Vaidyanathan and J. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*, New York: Kluwer Academic / Plenum Publishers, 2003.

[23] J. Van Campenhout, H. Van Marck, J. Depreitere, J. Dampre, "Optoelectronic FPGAs," *IEEE Journal of Selected Topics in Quantum Electronics*, Vol. 5, No. 2, March - April 1999, pp. 306–315.

[24] J. Van Campenhout, "Solving the Interconnect Bottleneck. Optoelectronic FPGAs," *Broadband Optical Networks and Technologies: An Emerging Reality/Optical MEMS/Smart Pixels/Organic Optics and Optoelectronics, 1998 IEEE/LEOS Summer Topical Meetings*, July 1998.

[25] H. Van Marck, J. Depreitere, D. Stroobandt, and J. Van Campenhout, "A Quantitative Study of the Benefits of Area-I/O in FPGAS," *Proc. of the 8th Great Lakes Symposium on VLSI*, Febraury 1998, pp. 392–399.

[26] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective, Third Ed.*, Boston: Person Education, Inc., 2005.

[27] M. J. Wirthlin and B. L. Hutchings, "DISC: The Dynamic Instruction Set Computer," *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, J. Schewel, ed., *Proceedings of SPIE*, vol. 2607, 1995, pp. 92–103.

[28] Xilinx Inc., "Virtex-5 User Guide,"
available at: http://direct.xilinx.com/bvdocs/userguides/ug190.pdf.

[29] Xilinx Inc., "Virtex-5 Configuration User Guide,"
available at: http://direct.xilinx.com/bvdocs/userguides/ug191.pdf.

Vita

Matthew Collin Jordan was born on November 25 1981, in Lansing, Michigan. In May 2004 he graduated *cum laude* from Michigan Technological University with a Bachelor of Science in Computer Engineering. Subsequently he joined the graduate program in the Department of Electrical and Computer Engineering at Louisiana State University. He is expected to receive his Master of Science in Electrical Engineering in August of 2006.