A HIGH-PERFORMANCE, HYBRID WAVE-PIPELINED LINEAR FEEDBACK SHIFT REGISTER WITH SKEW TOLERANT CLOCKS

By

JEFFREY LOWE

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

Washington State University
School of Electrical Engineering and Computer Science

August 2004

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JEFFREY LOWE find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

A HIGH-PERFORMANCE, HYBRID WAVE-PIPELINED LINEAR FEEDBACK SHIFT REGISTER WITH SKEW TOLERANT CLOCKS

Abstract

By Jeffrey Lowe, M.S.
Washington State University
August 2004

Chair: Jabulani Nyathi

Clock skew and clock distribution are increasingly becoming a major design concern in high performance, high density synchronous systems. Large clock networks are required for efficient clock distribution and they contribute significantly to the power dissipated by the system, while clock skew takes up a considerable percentage of the clock period.

Design effort for clock networks is currently estimated to take up 20% of system design time, while power dissipation due to the clock network is reported to be 30% of the total dissipation. It is therefore necessary to investigate the possibilities of other schemes that could results in cost reduction by avoiding complicated architectures while facilitating fast logic.

We explore the possibility of managing clock skew and reducing clock loading by applying the hybrid wave-pipelining scheme to a linear feedback shift register. The hybrid wave-pipelining scheme takes advantage of interconnects and data path delays to optimize clock skew and allows the clock to "travel" with its associated data. The system's clock in conjunction with stage delays is used to generate wave-pipelined clocks that have short cycle times and are skew tolerant. The hybrid wave-pipelined clock is designed to mimic the data path elements of the LFSR stage, thus reducing the uncontrolled clock skew, as well as clock loading. Thus, the

resulting skew is a result of the data path circuitry. A LFSR would provide a good means of measuring clock skew since the common edge of the clock triggers data transfer.

Linear feedback shift registers also have numerous common uses including pseudorandom number generator, random pattern generator and analyzer, encryption/decryption and direct sequence spread spectrum for digital signal processing. Their study with different clocking schemes is beneficial as LFSRs are easy to analyze and are found in many applications as mentioned.

In this thesis, it is shown that the use of hybrid wave-pipelining provides significant clock skew improvements (six times) compared with a buffered clock design, and also offers improved clock cycle time. There is also potential for a reduction in power dissipation associated with the clock trees, since the scheme reduces the need for complicated clock distribution networks.

Table of Contents

List of Figures

Page

List of Tables

Chapter 1

Introduction

In today's market, two of the important issues of system design are power dissipation and throughput. As the limits of silicon technology are being reached, the need for different methods to increase the throughput of a system is of more concern. The most common method of increasing the throughput of a system is by pipelining the operation. This allows for several sets of data to be processed in parallel with each other in assembly line fashion [1]. However, with conventional pipelining, the intermediate latches between stages create overhead in both path delay and system clock load, which reduces the maximum clock frequency achievable [2].

One way of overcoming the overhead created in conventional pipelining is using wave-pipelining techniques. These techniques can reduce clock network loading and distribution problems. In a fully wave-pipelined system the intermediate latches between stages are removed. The data can now be processed in waves as closely spaced as possible so long as data corruption from mixing unrelated data waves together does not occur [2]. This technique allows for less overhead caused by the intermediate latches in conventional pipelining. There is also a speed increase seen in wave-pipelining both from the latches being removed, and the fact that the data waves can be spaced in such a way to recover idle time lost by stages operating faster than the system clock in conventional pipelining. The major drawbacks of wave-pipelining are the design time needed to balance the delay paths so that the probability of data waves overlapping is

minimized, the internal nodes are not easily accessible, and delay grows as pipeline depth increases.

Hybrid wave-pipelining is a technique which draws from both of the previously mentioned methods to balance the gains of each method. It uses wave-pipelining methods within the individual stages and the conventional pipelining method of intermediate latches between stages. This allows for the data waves present in the system to be compressed further, since the system is now dependant upon the difference in path delays per stage instead of in the whole pipeline [7].

The advantages of hybrid wave-pipelining will be explored using a Linear Feedback Shift Register (LFSR) which enables the study of performance improvements as well as the constraints due to logic in the feedback path. The two major configurations for the feedback calculations in LFSRs are Fibonacci and Galois [8]. LFSRs have many common uses including pseudorandom number generators and cryptography [10]. Another common use of LFSRs is built in self testing (BIST), which generates test vectors to the system and an output ROM to compare the results to expected values [11].

LFSR systems are typically designed using either FPGAs or DSPs. While this leads to a working system that is flexible, the achievable speed is limited by the fact that FPGAs and DSPs are general purpose designs. By using VLSI techniques to design an LFSR, the throughput can be increased and the LFSR is easily integrated into a system design since the area needed is minimal.

One complicated issue with using wave-pipelining techniques on the LFSR is the feedback path. In a fully wave-pipelined system the timing of when the feedback arrives back to the input is difficult to control and relatively unexplored. This leads to the use of hybrid wave-

pipelining techniques to combine the advantages of both conventional and wave-pipelining with possibly less design time needed.

The details of pipelining including conventional pipelining, wave-pipelining and hybrid wave-pipelining are covered in Chapter 2. In Chapter 3, LFSR configurations and applications are discussed. Chapters 4 and 5 present the physical implementation and results of the hybrid wave-pipelined LFSR, respectively. Chapter 6 provides some concluding remarks and future research possibilities from this work. Finally, the software prediction code is included in Appendix A, and selected layouts are included in Appendix B.

Chapter 2

Pipelining Schemes

Pipelining involves dividing a process up into smaller tasks so that many different operations can be done simultaneously in assembly line fashion. By breaking the process up into many smaller tasks, the overall throughput of a system can be increased. However, the total time needed for a single operation is longer due to the intermediate latches needed between each pipelined stage for synchronization and control.

There are three pipelining techniques that will be covered in this chapter. The first technique, which is the most common, is conventional pipelining. The second technique is wave-pipelining, which seeks to improve the throughput of a system over that achieved through conventional pipelining. The last technique is hybrid wave-pipelining, which is a combination of conventional and wave-pipelining techniques. This technique attempts to achieve and surpass the gains of wave-pipelining without the added design complexity.

2.1 Conventional Pipelining

In conventional pipelining, the pipelining process contains control latches between each pipelined stage.

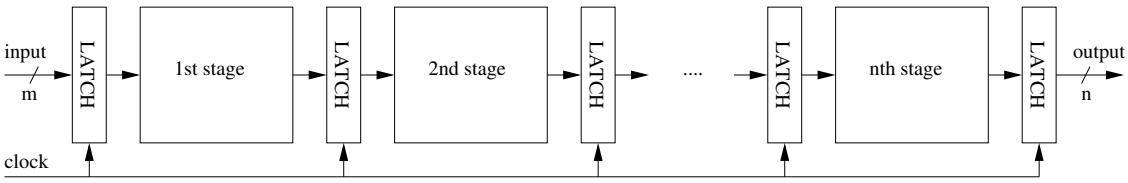

Figure 2-1: Conventional Pipelining System Architecture

Since all of the latches must switch at the same time to shift the data between stages, the system clock needs to be set to the longest stage delay. Thus, if all of the stages do not operate at exactly the same speed, any stage that runs faster than the slowest stage will be idling until the next clock edge arrives. The equation for the clock period of a conventional pipelining system is shown in Equation 2-1.

$$T_{clk} = D_{\max} + T_{setup} + T_{hold} + T_{skew} \tag{2-1}$$

A simple example of conventional pipelining in computer architecture is a five stage pipeline. These five stages are Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory (MEM), and Write-Back (WB) [1]. The clock speed of an unpipelined system is determined by the amount of time needed for an instruction to be completely processed by all five stages. The following figure shows the flow of the unpipelined system.

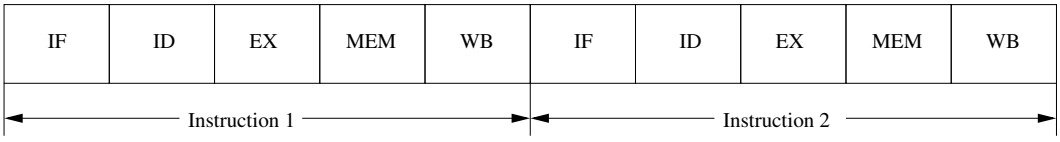

Figure 2-2: Instruction Execution in a 5-stage Unpipelined System

Once the pipe has been filled with instructions, every stage is actively working on a different instruction simultaneously in assembly line fashion. While the time-per-instruction increases slightly due to pipeline overhead, the throughput of the system increases greatly. This is due to the fact that an instruction completes execution every clock cycle, which is set to the

longest pipeline stage delay instead of the total amount of time an instruction needs to execute. The following figure shows five instructions in execution in the five stage conventional pipelining scheme example [1].

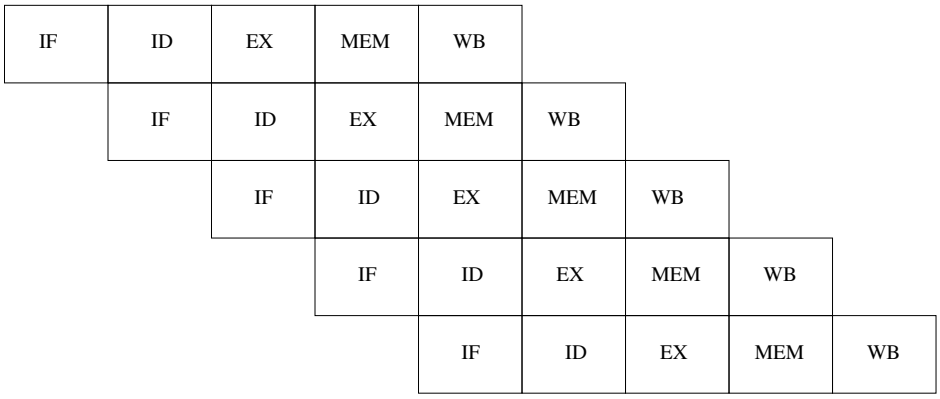

Figure 2-3: Instruction Execution in a 5-stage Conventional Pipelining System

Figure 2-4 shows a possible timing diagram of a conventional pipelined system where the 2nd stage has the longest delay path associated with it. The idle time is shown by the striped boxes, which can vary significantly between stages and accounts for large percentages of the clock period. Breaking the operation up into more balanced stages can reduce the idle time of the system.

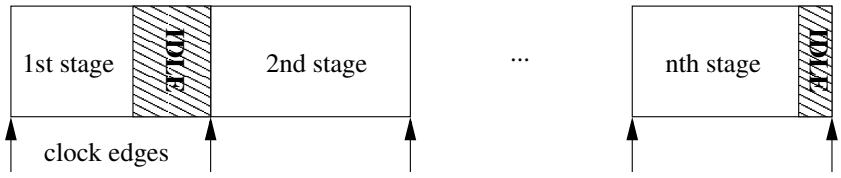

Figure 2-4: Conventional Pipelining Idle Time

The advantage of conventional pipelining is a system which can work on numerous different operations at the same time which maximizes the hardware usage and increases the throughput of a system. The design process is also simplified since each stage of the operation can be optimized and designed on a smaller scale. The time spent idling in the faster stages can

be recovered using different techniques, which is what wave-pipelining techniques are intended to do.

2.2 Wave-Pipelining

In wave-pipelining the intermediate latches between pipeline stages are removed. Since there is no longer synchronization of data transfer, the path delays in the pipeline stages need to be matched as closely as possible to increase the performance of the system using wave-pipelining techniques.

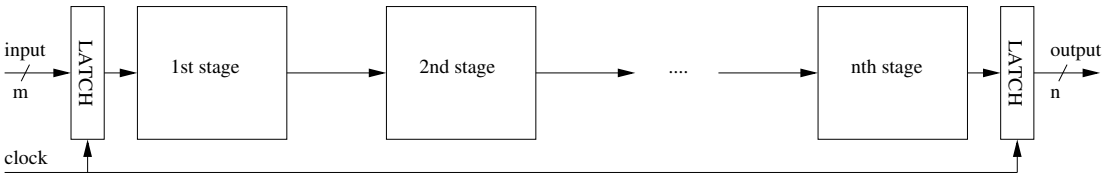

Figure 2-5: Wave-Pipelining System Architecture

Once the path delays are balanced, the output of a pipeline stage is sent to the next stage at approximately the same time regardless of the path taken. By knowing how long the path delay is through a stage, it can be mathematically determined how often data can be processed at each stage without prior data being overrun by new data. The result is closely spaced data waves that can be individually processed by the system without corruption.

The first restriction on the system clock period is determined by the difference between the minimum and maximum data path delay [2]. This difference determines the timing uncertainty in data arrival at the output latch. Another component of the clock period is the output latch which requires specific setup and hold times for data to be valid. Finally, any skewing in the clock will vary the clocking of the data into and out of the latch. These conditions lead to the following equation for the minimum clock period.

$$T_{clk(w)} \geq (D_{\max} - D_{\min}) + T_{setup} + T_{hold} + T_{skew} \quad (2\text{-}2)$$

There are two ways to reduce the path delay difference part of Equation 2-2. One way is to spend more time designing the logic of the circuit and try to balance the switching delays, which can be time consuming depending on the complexity of the circuit. The other way is to determine the worst-case data path delay and add delay elements to the other paths to optimize the difference and thus reduce the uncertainty of data arrival [3].

Table 2-1: Summary of variable names used in temporal/spatial figures

$D_{\min}$	the minimum propagation delay through the logic
$D_{\max}$	the maximum propagation delay through the logic
T_{clk}	the clock period between input waves
T_L	period data can be sampled at the output register
T_{hold}	setup and hold time period of the output register
$D_{\min\_hold}$	overall minimum delay including register hold times

The data waves in the system take N clock cycles to be processed. There is also an uncertainty due to uncontrolled clock skew. Equation 2-3 defines the clocking period on the output registers [4].

$$T_L = N \cdot T_{clk(w)} + \Delta \quad (2\text{-}3)$$

Figure 2-6 shows a temporal/spatial representation of the data waves in a wave-pipelining system [7]. The data path delays, setup and hold time of the output latch, and clock skew affect

the minimum spacing possible in the waves. This spacing needs to assure that there is no data loss due to overlapping waves. It can also be seen that at any given point of time, multiple waves can be present in the system. In Figure 2-6, there are two waves present in the system at any time instance as indicated by the dotted line [6].

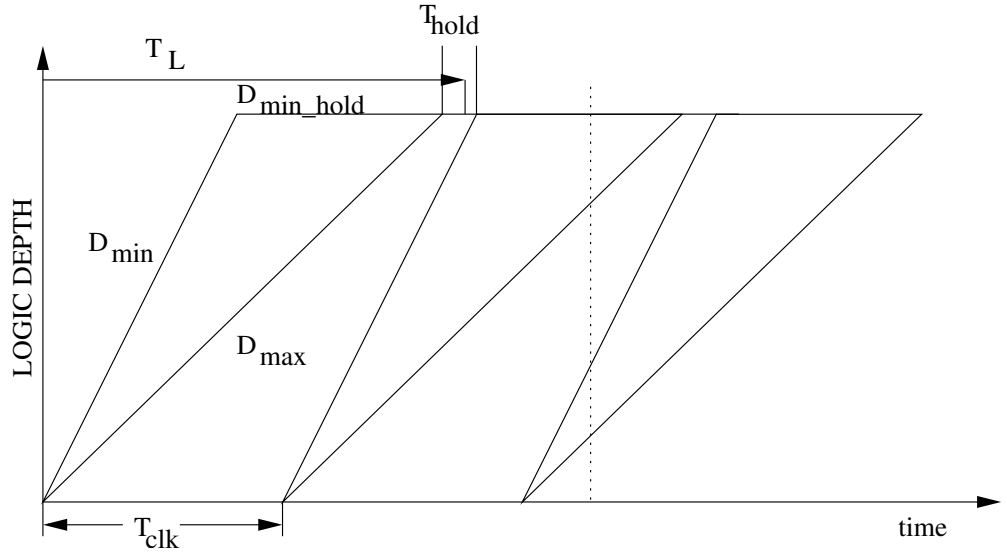

Figure 2-6: Temporal/Spatial Diagram of a Wave-Pipelined System

Wave-pipelining techniques increase the design complexity of a system. Since the path delays need to be carefully matched to achieve the best results, this technique should only be used at the module level. Using latches to connect modules together breaks the wave-pipelining design process down to smaller circuits that can be more easily optimized to maximum the number of data waves in the logic simultaneously.

2.3 Hybrid Wave-Pipelining

In hybrid wave-pipelining the techniques of both conventional pipelining and wave-pipelining are combined to take advantage of the positive aspects of each technique. When using hybrid wave-pipelining techniques the architecture looks similar to that of conventional

pipelining. The difference is that instead of optimizing the delay path to balance the delays between stages, the delay path is optimized with wave-pipelining techniques to optimize the delay difference within each stage. This results in the delay difference of a stage with the largest variation determining the constraints on the clock instead of the delay difference through the entire pipeline. These results also improve over conventional pipelining since the clock does not depend on the longest delay through a single stage.

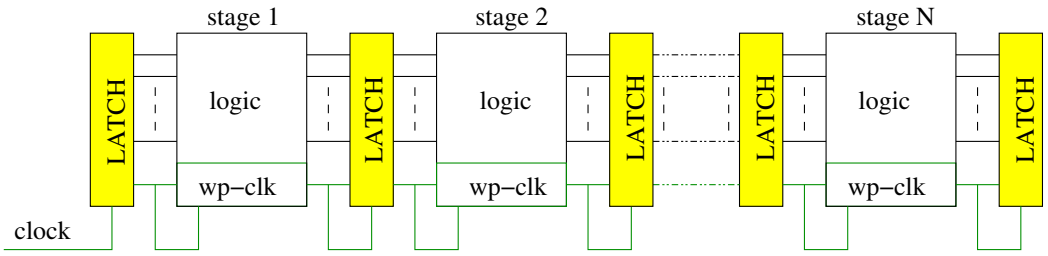

Figure 2-7: Hybrid Wave-Pipelining System Architecture

Since the clock constraints are determined by the largest stage delay difference instead of the entire pipeline delay difference, the clock period can be reduced. In the temporal/spatial chart this allows some overlapping of the delay cones that would be seen in a wave-pipelined system. Figure 2-8 shows the temporal/spatial diagram of a hybrid wave-pipelined system [7].

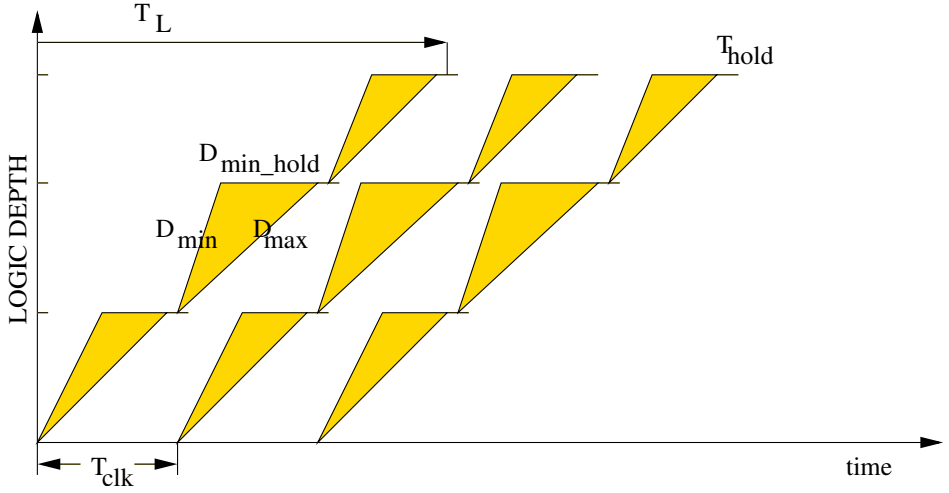

Figure 2-8: Temporal/Spatial Diagram of a Hybrid Wave-Pipelined System

The equations that describe the hybrid wave-pipelined system of Figure 2-8 can be seen in Equation 2-4. The clock period for a hybrid wave-pipelined system is shown in Equation 2-4. The output latch period of the hybrid wave-pipelined system is similar to that of the wave-pipelined system except that the clock period involves $D_{\min\_hold}$ instead of $D_{\min}$.

$$T_{clk(h)} \geq (D_{\max} - D_{\min\_hold}) + T_{setup} + T_{hold} + T_{skew}$$
$$D_{\min\_hold} \geq D_{\min} \qquad (2\text{-}4)$$

Since the hold time for the minimum path must be greater than or equal to the minimum path, the difference between the max and min path is reduced compared to wave-pipelining [4]. This reduction in the difference term results in a reduced clock period for the hybrid wave-pipelined system.

The benefit of hybrid wave-pipelining is a further reduced clock period and more control over the data in the pipeline. These systems have a stage structure more similar to that of conventional pipelining, but use different design techniques on the clock to recover the idle time lost in conventional pipelining. The ability to include feedback in a hybrid wave-pipelined system is also easier, compared to wave-pipelining, due to the fact that this system includes intermediate latches. In Figure 2-9, the hybrid wave-pipelined system with feedback can be seen.

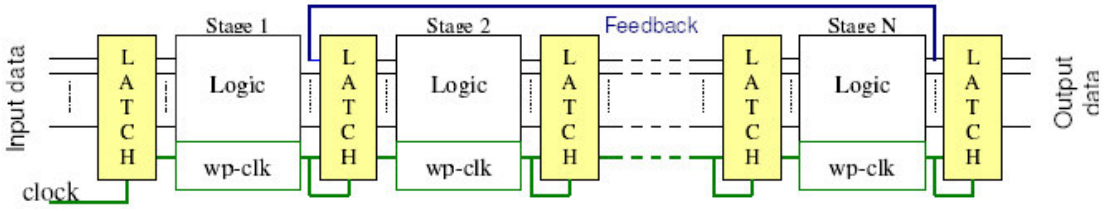

Figure 2-9: Hybrid Wave-Pipelining System Architecture with Feedback

In the architecture shown in Figure 2-9, the feedback is simply an interconnect wire between two stages. This feedback is shown in the temporal diagram shown in Figure 2-10.

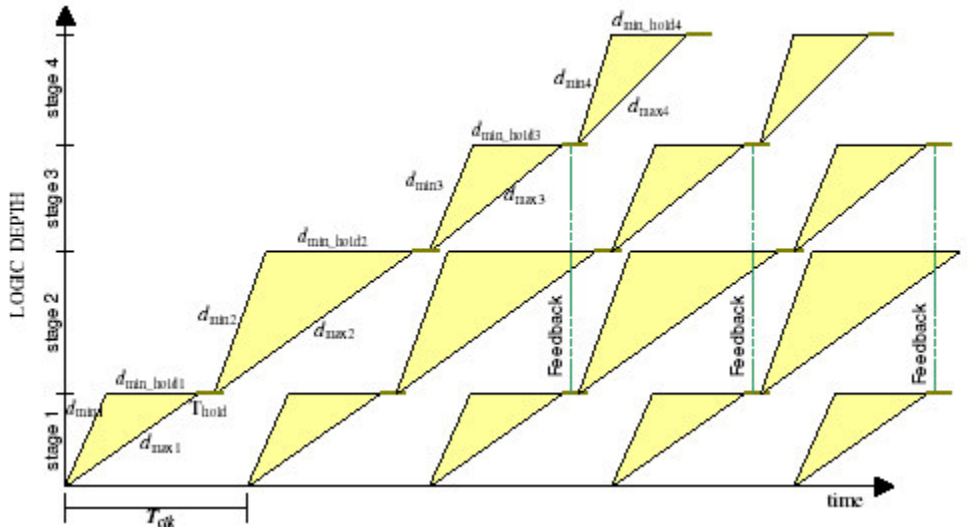

Figure 2-10: Temporal/Spatial Diagram of a Hybrid Wave-Pipelined System with Feedback

The clock period needs to be set such that the feedback path is synchronized with the stages in the forward path. The equation for the clock period when the feedback is taken from the output of stage *k* and sent to the input of stage *i* is shown in Equation 2-5. The value for *N* represents the number of stages contained in the feedback loop. A conditional statement determines the clock period since the clock period could be also be determined by the largest delay difference of a stage within the feedback loop.

$$\begin{aligned} T_{clk} &\geq T_{hold} + T_{setup} + T_{skew} + d_{\min\_hold(j)} && \text{if } d_{\min\_hold(j)} > \frac{1}{N}\sum_{m=i}^{k} d_{\max(m)} \\ T_{clk} &\geq T_{hold} + T_{setup} + T_{skew} + \frac{1}{N}\sum_{m=i}^{k} d_{\max(m)} && \text{otherwise} \end{aligned} \tag{2-5}$$

A comparison between the clock period equations of the pipelining techniques discussed in this chapter is shown in Table 2-2.

Table 2-2: Comparison of Clock Period Constraints between Pipelining Techniques

Pipelining Technique	**Cycle Time (T_{clk})**
Conventional Pipelining	$T_{clk} = D_{\max} + T_{setup} + T_{hold} + T_{skew}$
Wave-Pipelining	$T_{clk(w)} \geq (D_{\max} - D_{\min}) + T_{setup} + T_{hold} + T_{skew}$
Hybrid Wave-Pipelining	$T_{clk(h)} \geq (D_{\max} - D_{\min\_hold}) + T_{setup} + T_{hold} + T_{skew}$

In conventional pipelining, the clock period was determined by the maximum delay through a stage. The clock period for the wave-pipelining techniques was determined by the delay difference between the maximum and minimum path through the system. Since $D_{\min\_hold} \geq D_{\min}$, the clock period in the hybrid wave-pipelined system is reduced compared to the wave-pipelined system.

Chapter 3

Linear Feedback Shift Registers

A Linear Feedback Shift Register (LFSR) usually has feedback to enable it to perform more advanced operations that might require a specific function to model the state behavior. The nature of the feedback circuit, plus which register location within the shift register it is connected to determines the function of the LFSR. This leads to a flexible circuit, especially if a reconfigurable design is used, that can be used in many applications depending on which function it is needed to implement. These applications include pseudorandom number generator, random pattern generator and analyzer, encryption/decryption and direct sequence spread spectrum for digital signal processing [16]. The nature of the function is particularly important in encryption key generation as it directly relates to the strength of the encryption.

LFSRs in the simplest definition are used as pseudorandom number generators. When properly configured for maximum length sequences, each state will be reached only once until every state has been reached. Once every state has been reached, the sequence will be repeated. This chapter will discuss the two main implementations of the feedback network, how to configure the LFSR for maximum length sequences, and introduce some applications of LFSRs.

Linear feedback shift registers in general are constructed with D-type flip-flops in the forward path and linear XOR or XNOR logic in the feedback path. The initial value of the shift register, shift register taps and feedback logic determine the output sequence [15]. If XOR gates are used in the feedback path, the LFSR will not change states if the current state is all 0's.

Likewise, if XNOR gates are used, the LFSR will not change states if the current state is all 1's [9].

3.1 Feedback Configurations

As was previously stated, the nature of the feedback path determines the function of the LFSR. There are two major approaches that can be used in the design of the feedback network, namely the Fibonacci and Galois implementations. The Fibonacci implementation has logic in the feedback path, whereas the Galois implementation has an output that is fed back to selected points in the feed forward path. The feed forward path in the Galois implementation contains the feedback logic.

3.1.1 Galois Linear Feedback Shift Register

The major advantage of the Galois implementation is the fact that the output of the series is fed back to each of the selected points at the same time instance, versus the Fibonacci implementation where minor delays between feedback signals are possible. Since the feedback is in series with the operation of the shift register, the extra time needed for feedback calculations in the Fibonacci case can be reduced. This implies that the length of the series can be increased with very minimal increase in computational delays. This configuration involves weighted modulo-2 summations in the feed forward path [8].

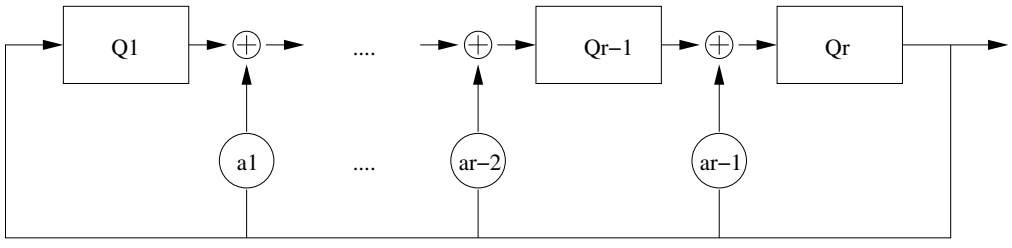

Figure 3-1: Galois Linear Feedback Shift Register

The taps a_1 through a_{r-1} are determined based on the polynomial that will be used to describe the behavior of the LFSR. These can be weighted values to achieve different characteristics, or simply '0' and '1' for not connected or connected respectively. The following pair of equations can be used to model the behavior of the Galois linear feedback shift register [8].

$$\begin{aligned} Q_i' &= Q_{i-1} + a_{i-1}Q_r, \quad for\ 2 \le i \le r \\ Q_1' &= Q_r \end{aligned} \tag{3-1}$$

In equation 3-1, r represents the number of stages in the LFSR, Q_i represents the current value of the i^{th} stage and Q_i' represents the new value for that stage.

3.1.2 Fibonacci Linear Feedback Shift Register

The Fibonacci implementation extracts the modulo-2 summation blocks from the feed forward path and performs the computations in the feed back path to the first element of the shift register. While this implementation is not as efficient as the Galois method, it allows for the study of feedback elements using hybrid wave-pipelining techniques.

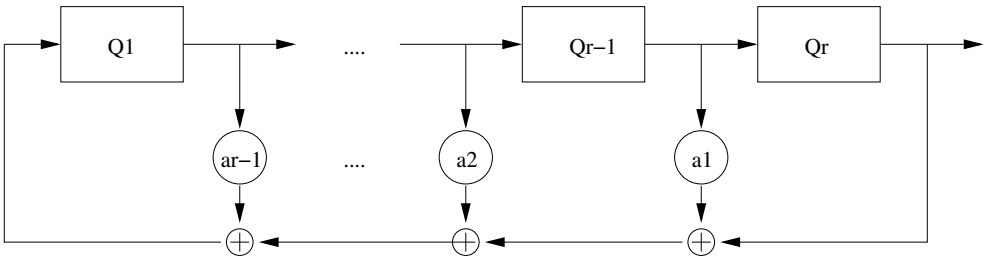

Figure 3-2: Fibonacci Linear Feedback Shift Register

A similar set of equations to those previously listed in equation 3-1 can be used to describe the Fibonacci implementation. The difference between the two implementations is that the taps are in reverse order [10].

The Galois implementation has an advantage in current technologies since an increase in entries does not necessarily translate to an increase in delay. However, in the nanometer range, the Fibonacci implementation might contribute to performance improvements since the transistors in the feedback path are expected to switch faster than interconnects. This is due to the long wire connecting the output to the selected taps contributing considerable and dominant delays in nanometer range technology. Since the Galois implementation does not have logic in the feedback path, the feedback wire could be of considerable length in a large stage LFSR.

3.2 Maximum Length Sequences

The maximum length sequence of an LFSR is the longest number of cycles in the LFSR until the generated pattern repeats itself. One major characteristic of the maximum length sequence is that every state will only be entered once between repetitions. Thus, for an N-bit LFSR with appropriate feedback taps selected, the sequence will take $2^N - 1$ clock cycles to repeat. The missing cycle is due to the lockup state of the feedback path.

Figure 3-3 shows the block diagram of a three-stage Fibonacci implementation. Since only two taps are required to configure the LFSR for maximum sequence length, a single XOR or XNOR gate can be used in the feedback path to perform the modulo-2 summation.

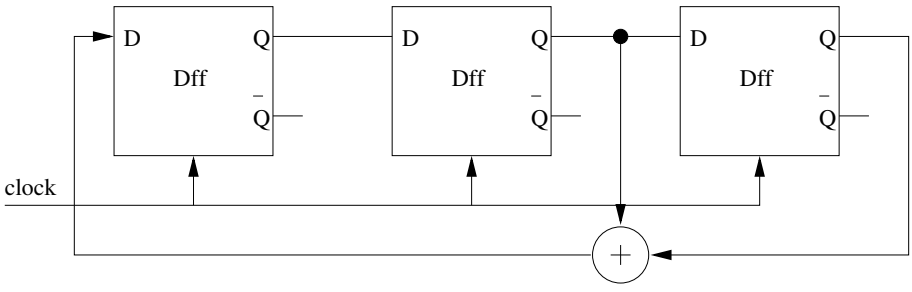

Figure 3-3: A Block Diagram of a 3-bit LFSR

In the 3-bit LFSR shown, there are $2^3 - 1$ or 7 states in the sequence. If an XOR gate were to be used in the feedback path, which does not allow the all 0 state, the unique values of Table 3-1 show the resulting sequence with a starting seed of 0,0,1 [9].

Table 3-1: Unique values for a 3-bit LFSR

Q3	Q2	Q1	
0	0	1	Seed
0	1	0	
1	0	1	
0	1	1	
1	1	1	
1	1	0	
1	0	0	
0	0	1	Repeat

As the number of stages in the LFSR increases, so do the number of possible configurations of the feedback network to achieve a maximum length sequence. Each of these different configurations results in a different sequence generated. If more feedback taps are used, then the equation to model the sequence becomes more complicated and thus the resulting maximum length sequence will be more complicated. The following table gives a select sample of the possible feedback taps to produce maximum length sequences [10]. The number of possible ways to configure the feedback as well as the number of taps used varies especially in larger LFSRs.

Table 3-2: Select N-bit LFSR maximum length sequence taps

# stages	2 taps	4 taps	6 taps
3	[3,2]	N/A	N/A
4	[4,3]	N/A	N/A
8	N/A	[8, 7, 6, 1]	[8, 7, 6, 5, 4, 2]
16	N/A	[16, 14, 13, 5]	[16, 15, 14, 13, 9, 3]
32	N/A	[32, 31, 30, 10]	[32, 31, 30, 29, 26, 4]

3.3 Applications

There are several applications involving the use of LFSRs. These applications include Built in Self Test (BIST) circuits, encryption key generation, and pseudorandom number generators. The following sections describe these applications in more detail.

3.3.1 Built in Self Test

There are two major components to a BIST scheme outside of the circuit under test. These are a test pattern generator and some sort of output response test circuit. The output test will verify the expected response of a circuit for the given test vector that was generated. This cannot cover every case however since more complex outputs could be based on a certain sequence of events which may have to be programmed into the vector generator or tested by hand. Since a large number of test vectors can be easily generated using an LFSR and little area overhead is required, their use in BIST circuits is advantageous.

The amount of time it takes for the self test circuit to run should be minimized so a large amount of time is not needed to run the built in test. A proposed method for this is a reseeding technique, where the seed in the LFSR is reprogrammed automatically in order to test for certain vectors that may occur in the normal operating mode [11].

3.3.2 Encryption Keys

The ability to generate a pseudorandom sequence of varying complexity lends itself to its use in encryption by generating complex keys. The use of LFSRs to generate these keys is widely used and can be found in places such as e-commerce, cell phones, and Bluetooth. The complexity of the encryption is determined by the number of stages in the LFSR as well as the

number of feedback taps used, which adds complexity to the polynomial used to describe the system.

The strength of the encryption key used determines how well protected the data being transmitted is. There are several criteria that can be used to measure the strength of the encryption key. These include the randomness properties, complexity, periodicity, correlation immunity of the key outputs, and nonlinear functions of the algorithm [14]. LFSRs can be used to generate long pseudorandom number sequences and the feedbacks can be easily changed to vary the complexity and algorithm of the system.

One such case is the Bluetooth system, which requires sixteen different polynomials to be implemented. The length of the encryption key in the Bluetooth system can vary between 8 and 128 bits. Since sixteen different polynomials need to be implemented to perform the encryption of data, there are sixteen different configurations of the LFSR needed. However with a 128-bit reconfigurable model the LFSR could be reconfigured for each polynomial as needed, which would reduce the amount of chip area needed to perform the encryption [13].

Chapter 4

System Design

The entire system has several modules that include the LFSR stage, feedback network, clocking schemes, and an I/O MUX. The single LFSR stage is duplicated to make up the N-bit LFSR module. The feedback network configures the LFSR to model a particular function. The clocking schemes include hybrid wave-pipelining and wave-pipelining. The I/O MUX connects the modules to the pad frame for the integrated circuit fabrication, and is also used in simulations to isolate the signals from each other so the module can load the initial seed.

This chapter covers the design of the individual modules used for the LFSR. The first section covers the LFSR stage, and possible circuits that could be used in its design. The next section covers the feedback network of the fixed model, as well as detail of how the configurable model works. In the third section, the different wave-pipelined clocking schemes are explained in detail. The I/O MUX, that is used to connect the system together, will be covered in the last section.

4.1 16-bit Linear Feedback Shift Register

A LFSR stage consists of a D Flip-Flop as was stated in Chapter 3. There are numerous possible configurations in the design of a single stage using either a D Flip-Flop or D Latch depending on if memory is required within each stage. Figure 4-1 shows several configurations that were explored.

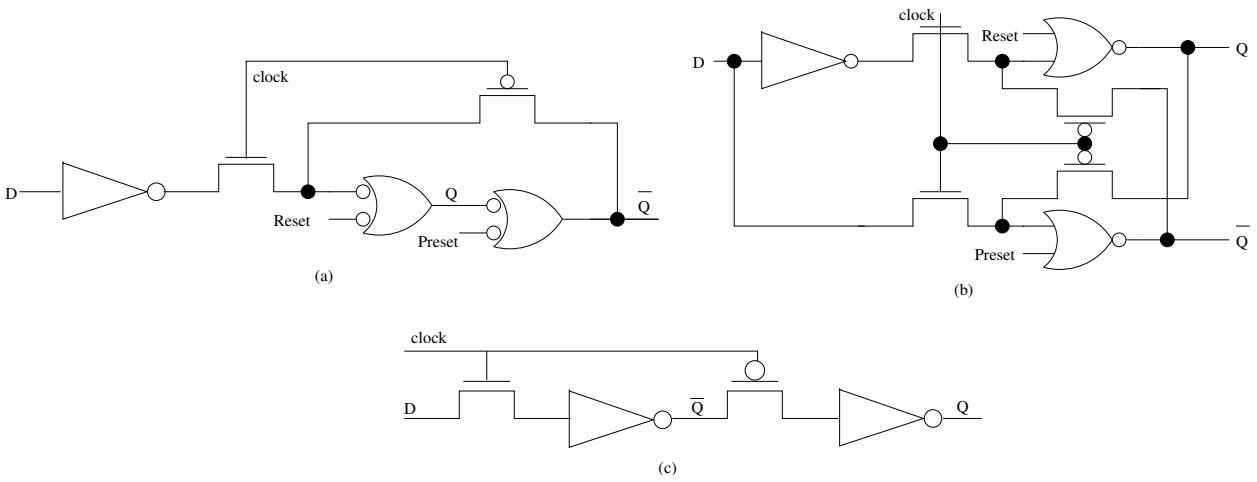

Figure 4-1: LFSR Stage Circuit Configurations

Since the goals of this thesis were to explore wave-pipelining techniques in systems with feedback, the LFSR stage configuration in Figure 4-1(c) was chosen. This configuration has a smaller circuit size and less external signals to control it than the alternate configurations shown in Figure 4-1(a) and Figure 4-1(b). Since it has no external signals, the number of pins required in the fabricated design as well as the control logic to put the LFSR into operation is reduced. The configuration shown in Figure 4-1(c) is also much simpler to apply and analyze wave-pipelining techniques since there are no feedback paths within the LFSR stage and only in the feedback network. As this configuration is a latch and not a flip-flop, if memory were to be required to enable the system to stall itself for a period of time, it could be latched into a RAM cache and reloaded to come out of the stall state.

This configuration can be broken into two operations that can be optimized using wave-pipelining techniques on the clock. The first operation of the LFSR stage is calculating the $\overline{Q_i}$'s. The second operation calculates the Q_i's which are used in the feedback network. By combining the feedback calculations with the Q_i calculations the setup and hold times are minimized. This

leads to a clock design goal that calculates all of the $\overline{Q_i}$'s in the least amount of time possible, thus maximizing the amount of time for feedback logic. The wave-pipelined clock design approach will be covered in section 3 of this chapter.

4.2 Feedback

The LFSR feedback network performs modulo-2 summation as was discussed in Chapter 3. These summations can be performed with either XOR or XNOR gates in the logic. The design uses the Fibonacci approach to enable an investigation of the effect hybrid wave-pipelining has on reducing delays associated with logic in the feedback path. Figure 4-2 shows the feedback logic using XOR gates for an LFSR with 4 taps. The tap numbers in Figure 4-2 and Figure 4-3 are taken from Table 3-2 for a 16-stage LFSR.

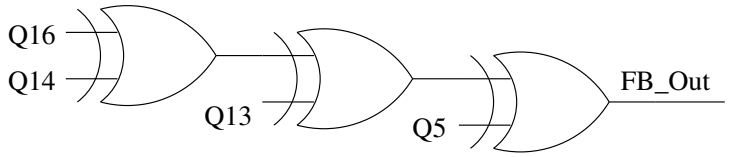

Figure 4-2: 4-tap Feedback Network using XOR Gates

Since the output of the feedback is determined by the number of logic '1's in the tapped stages of the LFSR, the feedback calculations can be performed in parallel. In a 4-tap system this reduces the number of layers of XOR gates in the feedback logic from three layers to two. An even greater increase in performance can be seen in an 8-tap system, where the number of XOR layers in the feedback path is reduced from seven to three.

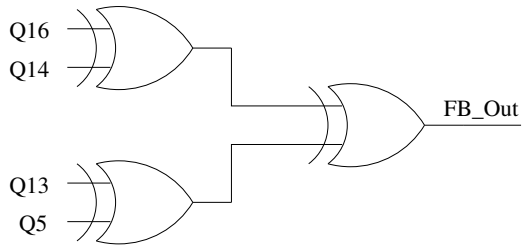

Figure 4-3: Delay Optimized 4-tap Feedback Network

Figure 4-3 shows the delay optimized network of the 4-tap feedback network. The third layer in an 8-tap feedback network would have four XOR gates in the first layer to connect all eight taps, and then the network of Figure 4-3 to determine the final output.

Each XOR gate was designed using the 6 transistor model shown in Figure 4-4 below. The advantage of this XOR gate is that the inputs *A* and *B* do not need to be inverted.

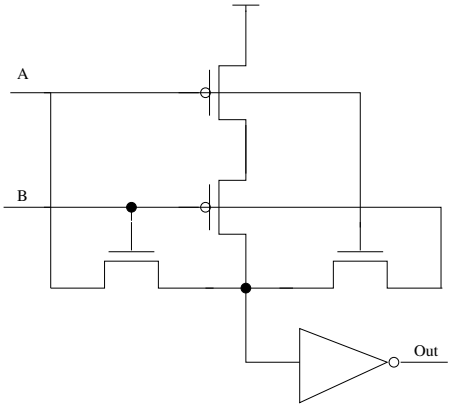

Figure 4-4: 6 Transistor XOR Gate

The configurable feedback model allows the user to select any tap, that the design allows, while the system is running by changing the EN(i) signals. When the enable line is selected, the related Q signal is passed to the input of the XOR gate. However, when the enable line is deselected, a logic '0' is passed to the input of the XOR gate instead of the related Q signal. Figure 4-5 shows a block diagram of a N-stage LFSR system with configurable feedback. The last stage in the network only needs the feedback control circuitry to reduce the XOR count of

the system by one. The feedback control connection from the last stage goes to the XOR in the next to the last stage. Every other stage mirrors the feedback control and XOR configuration as shown in stages 1 and 2.

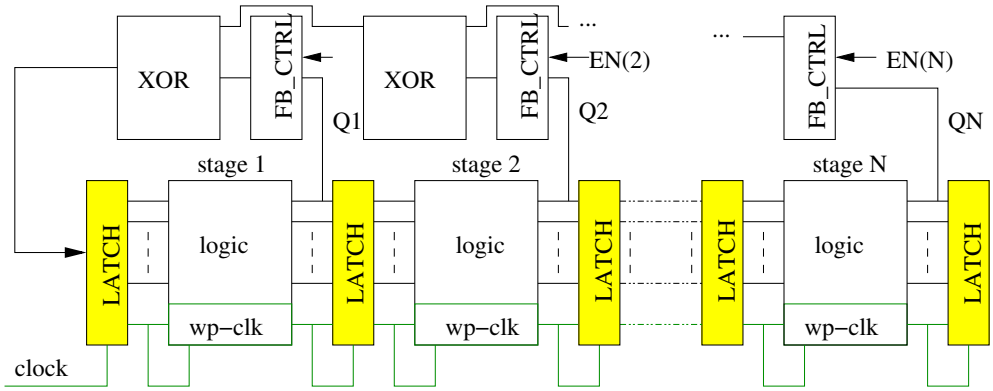

Figure 4-5: Configurable N-bit LFSR Block Diagram

The clock period for this system is based off the hybrid wave-pipelined equation without feedback. However, this equation needs to be modified since the feedback logic must finish calculations before the input can be clocked into the system. Equation 4-1 shows the modification to the hybrid wave-pipelined system equation for the clock period.

$$T_{clk(h\_fb)} \geq (D_{\max} - D_{\min\_hold}) + T_{setup} + T_{hold} + T_{skew} + T_{FB\_\max} \quad (4\text{-}1)$$

The expanded view of the feedback control block can be seen in figure 4-6.

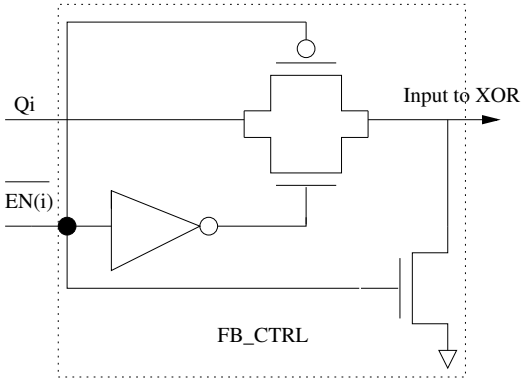

Figure 4-6: Configurable LFSR Feedback Enable

A single stage of the configurable LFSR feedback enable is shown in Figure 4-6. This circuit will allow any stage to be a feedback tap. When the feedback is disabled, a logic '0' is present at the input of the XOR. This means that the output of that particular XOR gate is equivalent to the input on the other terminal. The feedback network delay will increase as the number of LFSR stages increase, since the feedback signal must propagate through every XOR gate. The same technique that was shown in Figure 4-3 to optimize the 4-tap feedback network can be applied to the configurable LFSR feedback network. However, since the number of XOR gates is related to the number of stages in the LFSR, the feedback network can grow large as the number of stages increases. This can be limited by only allowing a certain number of stages to be included in the possible feedback network configuration.

4.3 Clock

The design of the hybrid wave-pipelined clock was mirrored as closely as possible to contain the same logic elements as the LFSR circuit to match path delays. Since the clock and data path both contain the same circuit elements, the process variations would affect the delay path in similar fashions and therefore these effects are minimized.

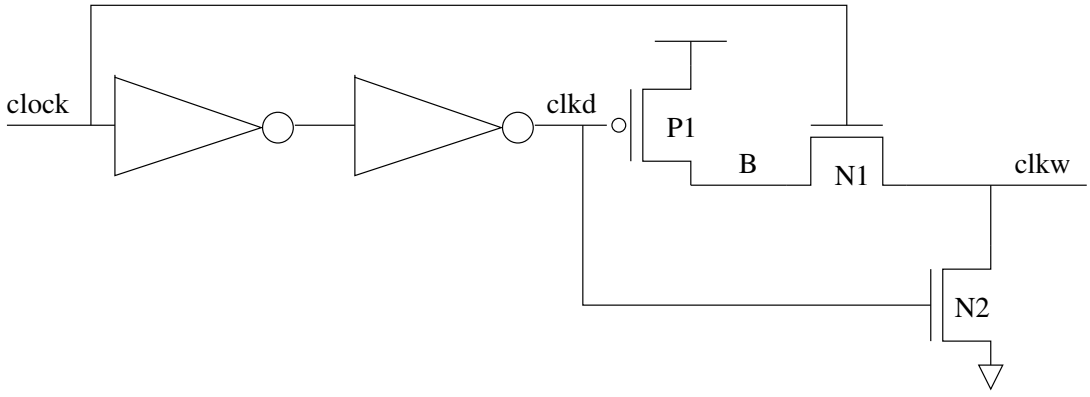

Figure 4-7: Hybrid Wave-Pipelined Clock Generator

The delayed system clock labeled *clkd* in Figure 4-7 provides a path to ground whenever it is high and places the *clkw* node at logic '1' whenever it is low through transistors N2 and P1 respectively. The system clock (clock) is input to transistor N1 preventing the *clkw* signal from being the exact inverse of the delayed clock and serving to charge up node B just before evaluation occurs [9]. A parallel circuit involving similar logic is used to generate the inverse *clkw* signal instead of just inversion.

The hybrid wave-pipelined clock seen in Figure 4-7 has a brief window in which a floating node occurs, which can cause problems at slower clock speeds. The peak voltage levels of the clock signals also do not reach full swing, which does not pose a functionality problem but can be fixed. The revised circuitry in Figure 4-8 fixes both the voltage swing and the floating node problem.

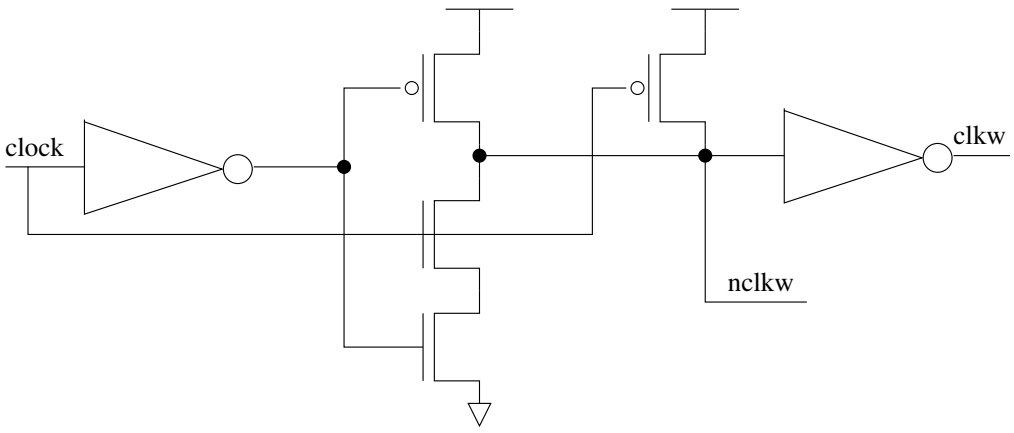

Figure 4-8: Revised Hybrid Wave-Pipelined Clock Generator

However, since both clock signals are generated in one circuit path, an output buffer needs to be used to isolate the generation circuitry from the LFSR network. With a buffered clock generator the signals can be cleaner and attain full swing, however the delay between the generated clock signals and the true system clock increases. This extra delay between the signals

is acceptable only if the setup/hold times on the output buffer takes this into account as the signals must be present when the system clock is available to clock the data between modules.

Another method to generate the clock signal using Hybrid wave-pipelining techniques is shown in Figure 4-9. With a properly designed system, it is possible for the clock signal to be completely pipelined with the associated data signals. With a clock signal that is pipelined with the data, the system is self-clocking when data is available. The clock generator shown in figure 4-9 attempts to accomplish this goal.

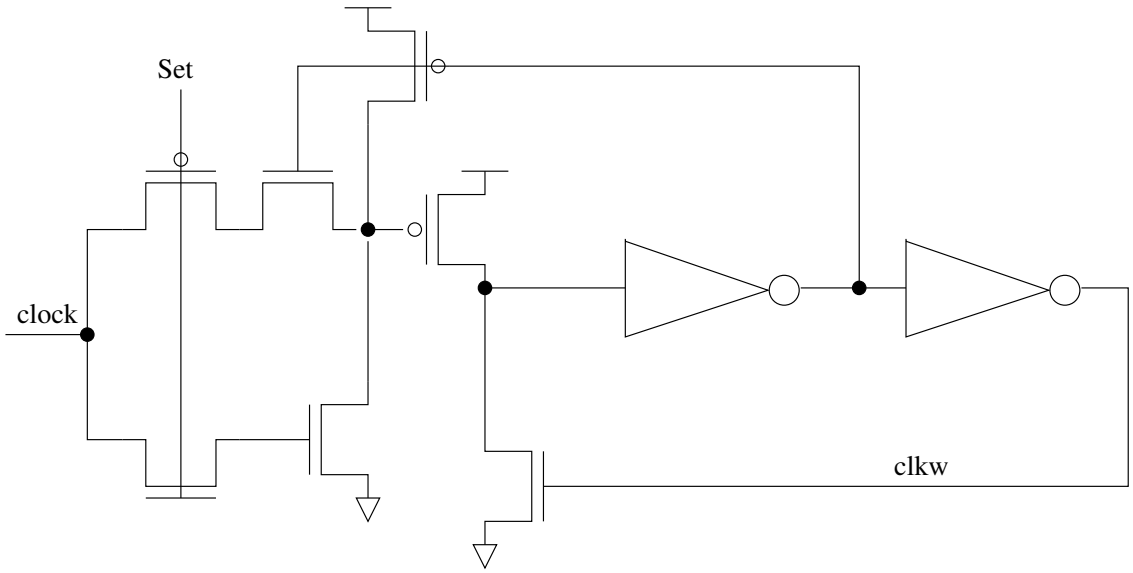

Figure 4-9: New Hybrid Wave-Pipelined Clock Generator

The difficulty with a hybrid wave-pipelined clock that is pipelined with the data is the feedback path. The clock signal needs to be able to pass through this different path delay in a similar fashion so it can clock the new data back into the system. The other critical point of this system is the output latch configuration to interface this module design with other modules that need the generated data.

4.4 I/O MUX

The I/O MUX is used to connect the outputs of both the fixed and configurable LFSR system to the same 16 pins of the integrated circuit package. Since the I/O MUX is connected to both an input and output pin of each LFSR circuit configuration, it needs to be a 16x4 MUX (or 16x2x2). The I/O MUX also is involved in switching the connections to the LFSR stages, since these 16 pins are also used as inputs when loading the initialization seed into the LFSR. The following figure shows a block diagram of the two LFSR system and the I/O MUX connections.

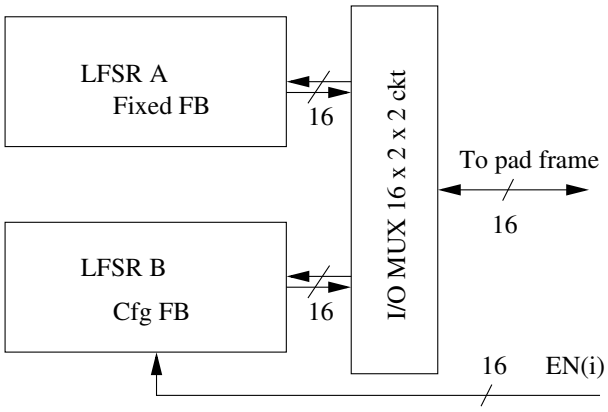

Figure 4-10: System Block Diagram with I/O MUX Interface

In Figure 4-11, a detailed view of one of the I/O MUX channels can be seen. This figure shows the input and output wires to both the fixed LFSR (A) and the configurable LFSR (B). Thus it takes sixteen instances of Figure 4-11 to form the complete I/O block seen in Figure 4-10.

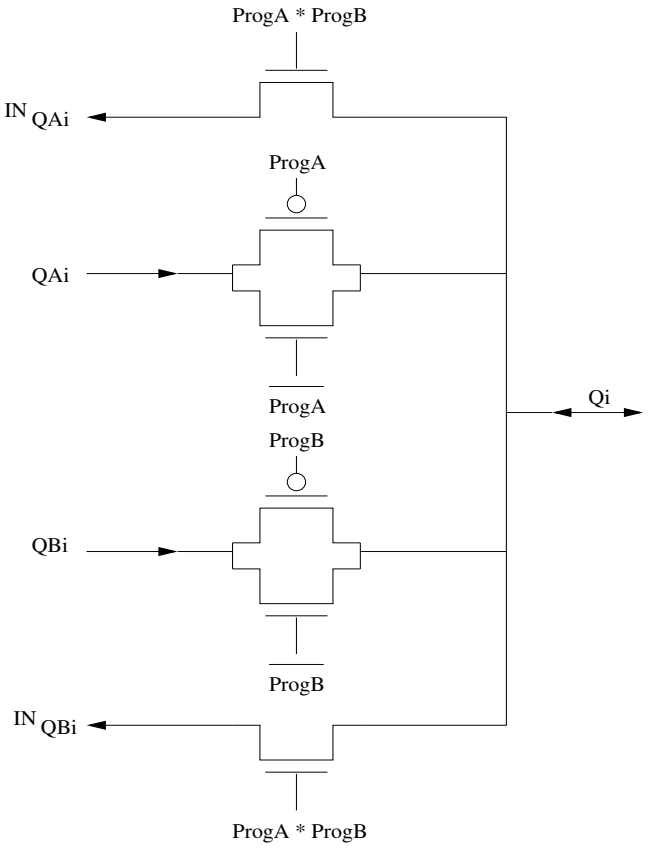

Figure 4-11: 2 Circuit I/O MUX

If the only input to the LFSR were at stage 1, then it would take N clock cycles to load the seed into all N stages. If there were input into every stage of the LFSR, the LFSR could be initialized to the seed value in 1 clock cycle. The single cycle initialization can be accomplished by disabling the clock and attaching the inputs inside of each LFSR stage as seen in Figure 4-12.

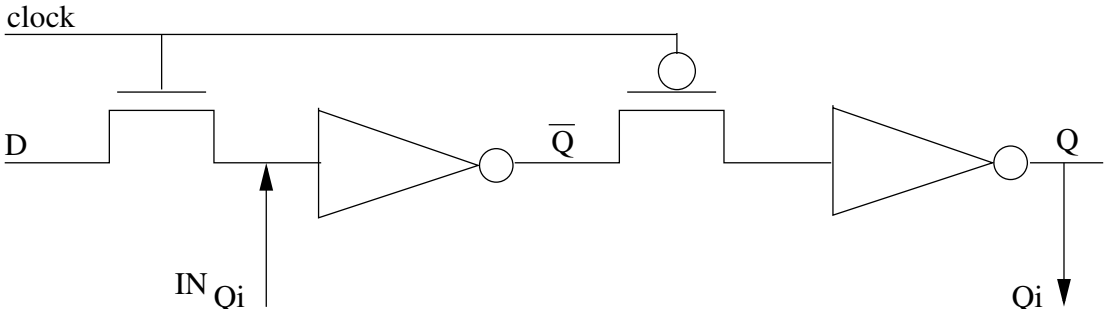

Figure 4-12: I/O Connections to LFSR Stage

The clock disable circuitry for single cycle seed initialization is also used in the fabricated design to choose whether the fixed or configurable system will be in operation. Since all of the Q values of each LFSR are connected to the same output pins on the other side of the I/O MUX, only one of the LFSR systems is usable at a time. By disabling the other system, potential signal conflicts can be avoided as well as overall power savings.

The hybrid wave-pipelined clock system simulations and fabricated chip results are shown in the following chapter. The other clock designs were not fabricated, but the simulations for these designs can be found in the following chapter as well.

Chapter 5

Results

In this chapter the benefits of wave-pipelining techniques will be shown. The functionality of the design using a hybrid wave-pipelined clock scheme is seen both in simulation and in practice. The design was fabricated using 0.5μm technology. This chapter will also present computer methods for verifying the output sequence of the LFSR, which could be adapted for automatic testing in larger systems.

The simulation results using Cadence are discussed in depth in the first section of this chapter. The results of the fabricated chip are reported in the second section. In the last section of this chapter, the software prediction model that was developed to verify the sequence in both the simulation and the fabricated chip is covered.

5.1 Simulated Results

This section includes the simulation results for the different clocking schemes experimented with, the clock skew measurements, and the functionality of the LFSR. All simulated results discussed are generated using Cadence software, TSMC 0.25μm technology, and a 3.3 volt supply level.

The wave-pipelined signal generated by the circuit of Figure 4-5 appears in Figure 5-1. The inverse of the wave-pipelined clock is also shown. The wave-pipelined clock has 2.37 volts as its highest value denoting logic ‘1’ and its pulse width is much shorter than that of the

reference clock. Permitting the logic of the system to determine the signal pulse widths and amplitudes results in a clock pulse width reduction of 55 percent. This implies that the clock cycle time can be improved considerably.

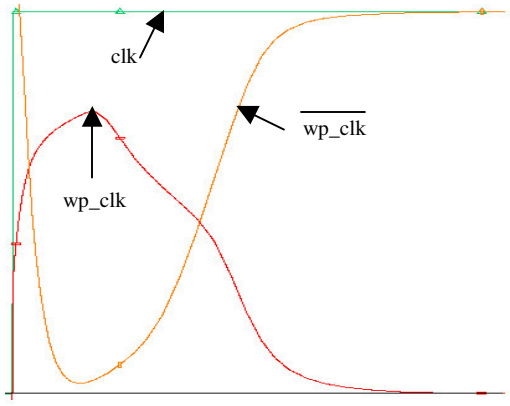

Figure 5-1: Reference and Hybrid Wave-Pipelined Clocks

Having the logic determine pulse widths and amplitudes of the clock permits the feedback logic to receive inputs early, thus reducing the overall system delays associated with the feedback path. This is a direct result of the hybrid wave-pipelining approach, where the delay differences are reduced per stage and the intermediate latches are used to balance the delay paths. Wave-pipelining the clock allows the system to maintain high clock speeds even with added logic in the feedback path. An increase of logic stages in the feedback path can occur when the number of taps within the shift register is increased and similar approaches can be used to minimize the effect of this logic delay.

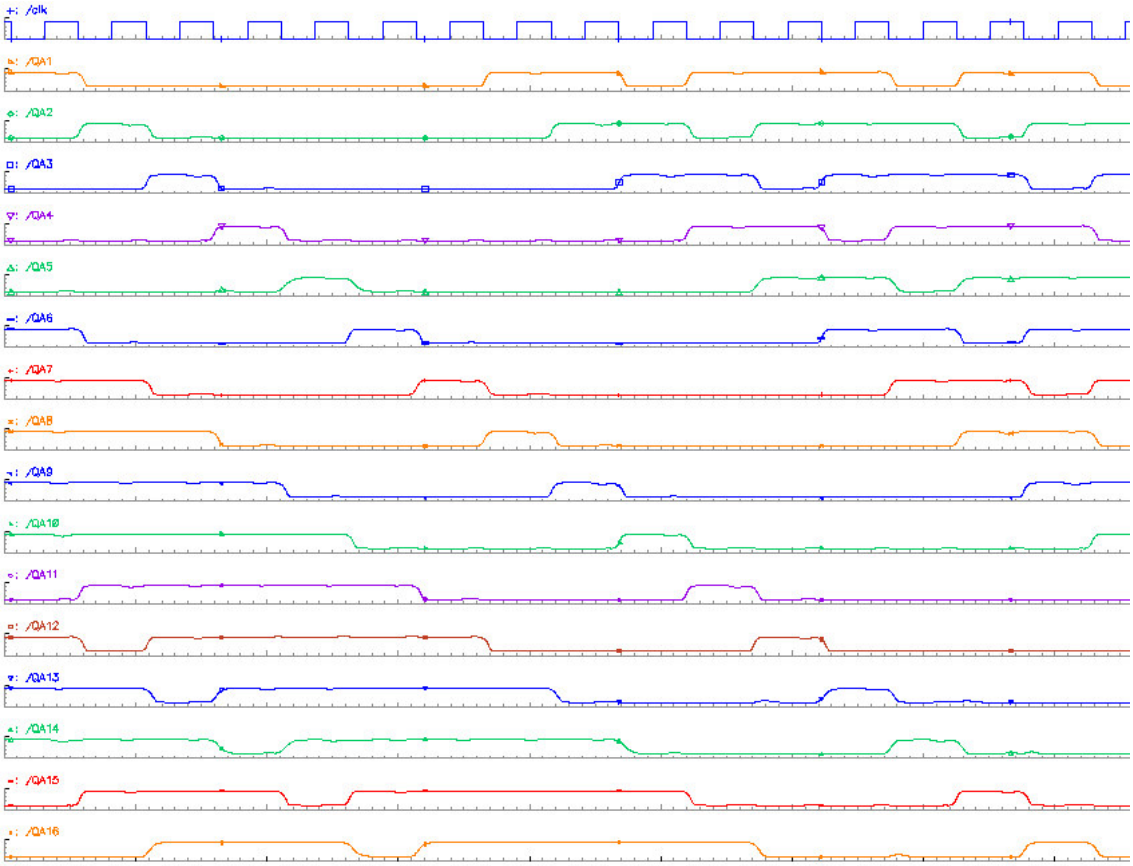

Figure 5-2: Logic '1' Propagation for the 16-bit LFSR

Simulation results of a 16-stage hybrid wave-pipelined LFSR showing the clock and the propagation of a logic '1' through the stages are shown in Figure 5-2.

The clock skew measurements for the hybrid wave-pipelined and buffered clocks can be seen in Figure 5-3. These measurements are taken with respect to outputs Q1 and Q8. The capacitance load between Q1 and Q8 are equal since neither have a feedback tap, so both drive the following stage only. These points are also the closest and furthest away from the clock generator, so the skew between clock edges can be measured.

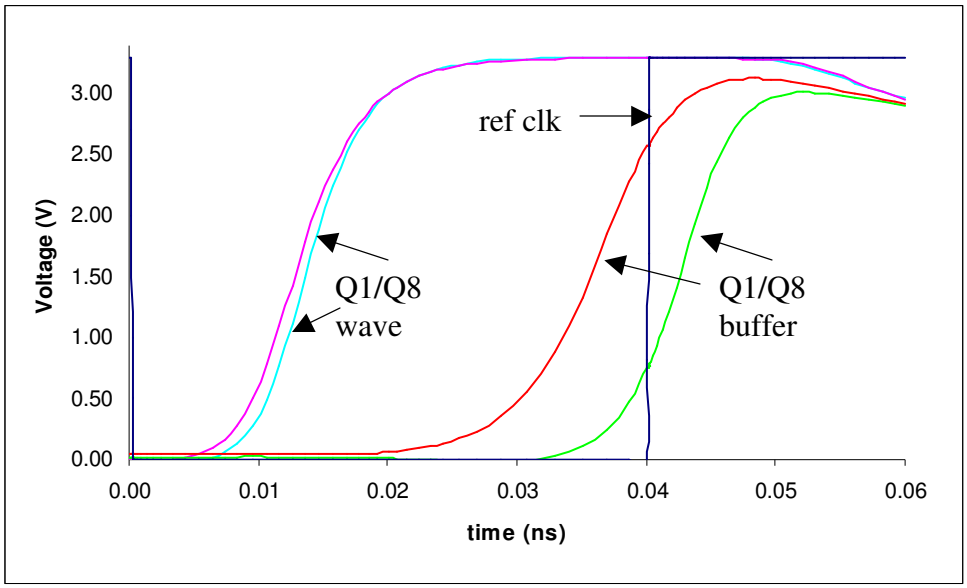

Figure 5-3: Clock Skew Comparisons between Hybrid and Buffered Clocks

In Figure 5-3, the signals using the hybrid wave-pipelined techniques have 10ps of measured skew between their rising edges. This is considerable improvement compared with the 68ps measured for the buffered clock approach.

The revised hybrid wave-pipelined clock signals are shown in Figure 5-4. This circuitry fixed the floating node problem in the original hybrid wave-pipelined clock circuit. The revised method also raised the clock signals closer to full voltage swing.

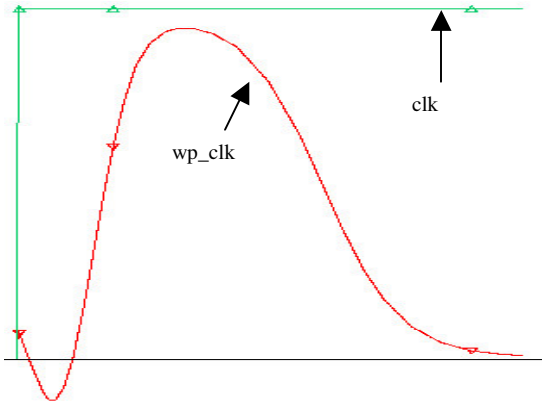

Figure 5-4: Reference and Revised Hybrid Wave-Pipelined Clocks

The new hybrid wave-pipelined clock signals are shown in Figure 5-5. This scheme is currently being investigated for its advantages compared to the hybrid wave-pipelined clock system presented earlier. While these results are preliminary, the clock signals seen in Figure 5-5 show promise.

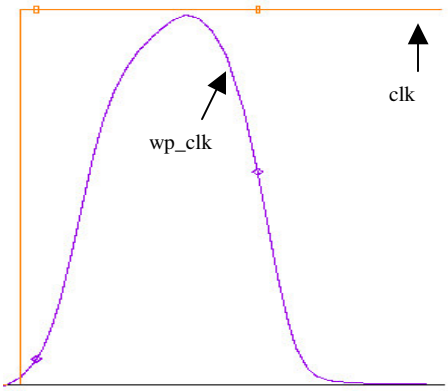

Figure 5-5: Reference and New Hybrid Wave-Pipelined Clocks

It should be noted that the new hybrid wave-pipelined clock circuitry is also designed to trigger on both edges of the reference clock, thus doubling the frequency. However, with the current design, the delay between the reference and generated clocks varies slightly. This results in a clock that does not have an equal duty cycle.

One of the comparisons between the clocking methods is the amount of delay between the system reference clock and the rising edge of the generated clock. The measurements were taken from the system reference clock edge to the point where the generated clocks reached 50% of the power supply on their rising edge. The following table includes measurements of these delays for the hybrid, revised and new hybrid wave-pipelined clock schemes.

Table 5-1: Delay between reference clock and generated clock

Hybrid wave-pipelined clock	8.55ps
Revised hybrid wave-pipelined clock	63.14ps
New hybrid wave-pipelined clock	51.94ps

The original hybrid wave-pipelined clock outperformed the other two methods in terms of skew between the reference clock and the generated clock. However, this method has an issue with the floating node in the generator. The revised hybrid wave-pipelined clock fixed the floating node issue as well as reaching a higher peak voltage level at a cost of larger skew between the reference and generated clocks. The new hybrid wave-pipelined clock has a similar peak voltage level and improves the delay between the reference and generated clocks over the revised method.

Since one of the advantages of wave-pipelining techniques is decreased clock loading, one would expect the power measurements of a wave-pipelined system to be less than that of a standard buffered block system. Preliminary results show that these clocking methods do in fact decrease the average current draw of the clock system as expected.

5.2 Chip Results

The circuit was fabricated with TSMC 0.5μm technology. The test equipment used was the TestosterICs Brain Box developed by One Hot Logic, and a Tektronix TLA714 Logic Analyzer. The Brain Box allowed for test vector generation and power and ground signals that were sent to the IC. The test vectors were generated through a simple file that included the values, pin numbers and timing requirements. Once loaded the software would transfer these signals with a clock to the integrated circuit via the test board.

Figure 5-6 can be used in the verification process of the fabricated design, and shows correct propagation of a logic '1' through all sixteen stages of the LFSR. The clock signal is on the first row of the figure. The following sixteen rows show the values of Q1 through Q16 with Q1 on the second row and Q16 on the bottom. The propagation of the '1' can be seen by following the diagonal from the top left to the bottom right of the figure. A different sequence pattern from the fabricated chip design will be shown in the following section.

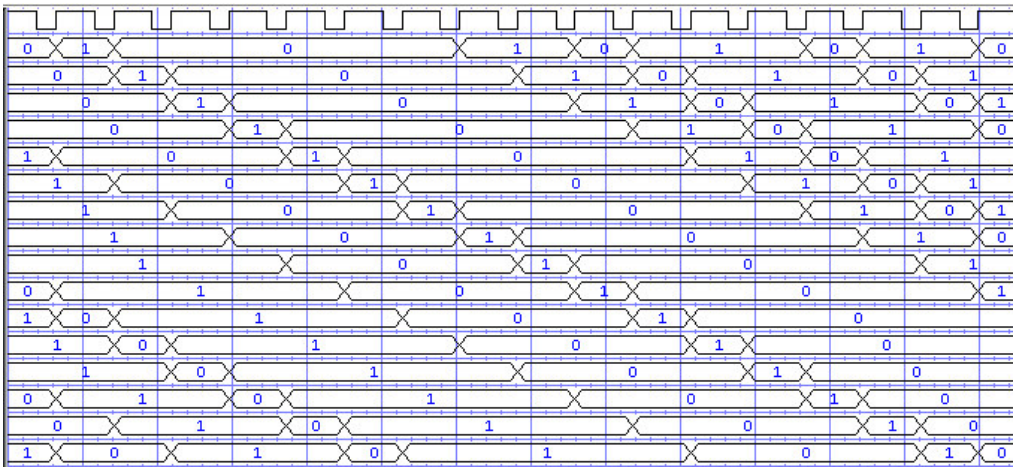

Figure 5-6: Propagation of a '1' through the LFSR using a Logic Analyzer

Due to limitations in the test equipment, the clock signal levels and frequency could not be analyzed. However, the fabricated design was shown to be successful in functionality both in the previously shown figures and in the next section that covers the software prediction code.

5.3 Validation using Software Prediction

Since the feedback configuration of the LFSR is easily mapped into a characteristic polynomial, a C program could be designed using this polynomial to model the behavior of the LFSR. Such a program was developed to either predict the following outcome sequence based

on any given input, or print out the entire sequence to a file thus enabling the user to verify the length of the sequence.

This program was first used to verify that the feedback taps [10] used did indeed yield maximum length sequences since this is cumbersome to show in hardware or simulation. The next step in the verification process was to simulate the LFSR with various starting seeds and then compare the results with the same starting seed entered into the program and lining up the predicted values with the simulation.

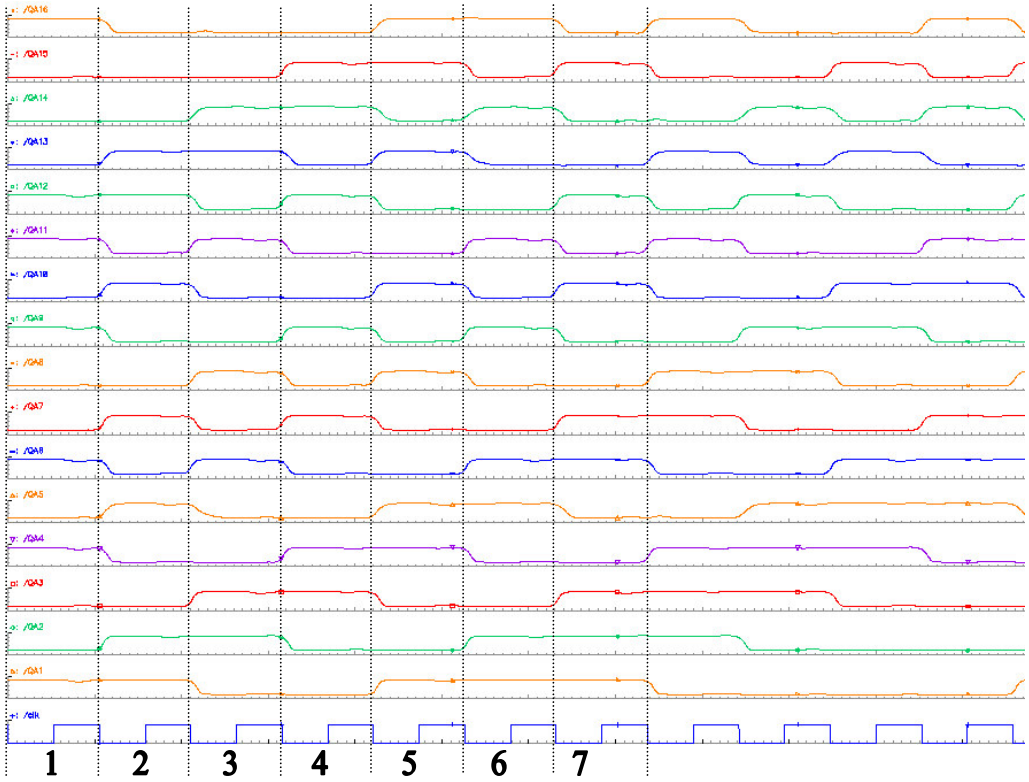

Figure 5-7: 16-bit LFSR Simulated Sequence Segment

The taps for the 16-bit LFSR used in design were 16, 14, 13, and 5. These values can be entered into the software prediction code along with the starting seed of 8D29 to verify the results of figure 5-7. The following figure is a screen capture of the software prediction code

execution in the 'predict following outcome' mode. The generated output of the program lists the current state (Cur), the resulting feedback value (In) and the next state (Out).

```
Enter number of stages: 16
Enter number of taps: 4
Enter tap 1: 16
Enter tap 2: 14
Enter tap 3: 13
Enter tap 4: 5
(m)seq length or (p)redict following outcome:
Input (Hex): 8D29
values are printed as HEX(DECIMAL)
(enter) to advance, (e)nter new input, (q) to exit
Cur 8d29(36137) : In 1 : Out 1a53(6739)
:
Cur 1a53(6739) : In 0 : Out 34a6(13478)
:
Cur 34a6(13478) : In 0 : Out 694c(26956)
:
Cur 694c(26956) : In 1 : Out d299(53913)
:
Cur d299(53913) : In 1 : Out a533(42291)
:
Cur a533(42291) : In 1 : Out 4a67(19047)
:
Cur 4a67(19047) : In 0 : Out 94ce(38094)
:
Cur 94ce(38094) : In 0 : Out 299c(10652)
:
Cur 299c(10652) : In 0 : Out 5338(21304)
:
Cur 5338(21304) : In 0 : Out a670(42608)
:
```

Figure 5-8: 16-bit LFSR Software Prediction Comparison Sequence

Figure 5-7 shows a simulation of the 16-bit fixed LFSR with a given starting seed. The verification using the software prediction for this seed can be seen in figure 5-8. By using random starting seeds in the simulation and verifying the resulting sequence for several cycles, the accuracy of the LFSR can be determined. This is more practical than verifying the entire maximum length sequence, since that would require a significant amount of time to simulate. It would also be difficult to verify the data unless an automatic test was developed.

Table 5-2: 16-bit LFSR truncated sequence

Cycle	State
1	8D29
2	1A53
3	34A6
4	694C
5	D299
6	A533
7	4A67

A similar comparison of sequences can be done using a logic analyzer and the same software prediction code to verify the functionality of the fabricated design. Figure 5-9 shows the same sequence as shown in the software prediction in Figure 5-8, and simulation in Figure 5-7, which are summarized in Table 5-1. The fabricated chip is verified in the same way as the simulation verification was done, which was accomplished by taking the starting seed and matching the sequence with the software prediction.

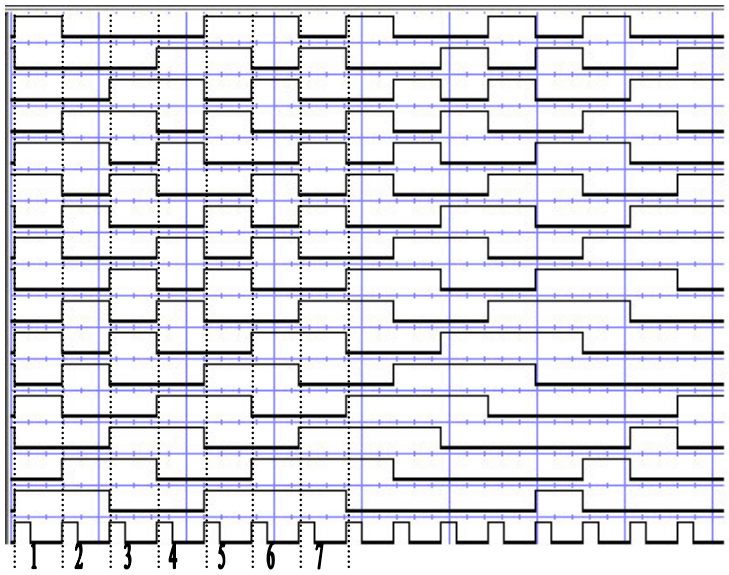

Figure 5-9: Fabricated 16-bit LFSR Sequence using a Logic Analyzer

The above comparisons were done for several starting seeds and given the pseudorandom behavior of the sequence yields a high probability that the entire maximum length sequence was generated correctly. The software prediction code was also very useful in the reconfigurable LFSR model where the feedback taps could be chosen on the fly. For further detail about the software prediction code see the C source code listed in Appendix A.

Chapter 6

Concluding Remarks

Several pipelining techniques that include conventional pipelining, wave-pipelining and hybrid wave-pipelining were covered in this thesis. Hybrid wave-pipelining was shown to have great potential for addressing clock skew and clock loading problems. It also enhances performance by reducing clock cycle time as well as logic idle time.

6.1 Summary

It was shown that in conventional pipelining the faster stages spend a significant portion of their time idling. In wave-pipelining the temporal/spatial cone for $D_{\min}$ and $D_{\max}$ spreads as the number of stages increases thus resulting in more design effort to reduce the difference between the two. The hybrid wave-pipelining technique combines the previously mentioned techniques, using wave-pipelining within a stage to reduce the delay differences and employing intermediate latches as in conventional pipelining to allow data to be issued to the next stage at the same time.

To demonstrate how hybrid wave-pipelining combines some attributes of conventional pipelining and wave-pipelining for superior performance, an LFSR module was designed, fabricated and tested. The basics of the LFSR and its applications were covered in Chapter 3. There are two major implementations to design the LFSR, which are the Galois and Fibonacci. Both of these methods use the same equations to generate the LFSR function. However, the

numbering of the feedback taps in the Fibonacci implementation occurs in the opposite direction of the Galois implementation. The Galois implementation was shown to include the feedback computations in the feed-forward path, whereas the Fibonacci implementation performs the feedback computations in the feedback path.

In current technologies, the Galois implementation outperforms the Fibonacci implementation due to the fact that no logic is present in the feedback path. However, in future technologies approaching the nanometer range, the long interconnect wire in the feedback path will most likely present the dominant delay path. In this case, the Fibonacci method is expected to perform better as the logic in the feedback path breaks the interconnect wire down into shorter segments, and the logic is expected to have faster switching transitions than the interconnect wires.

The design of the system was presented in Chapter 4 where all of the major cell designs such as the clock, LFSR stage, feedback path, and I/O MUX were included. The system that was fabricated on 0.5μm technology included two different LFSR modules. Both modules had 16-bit LFSR stages, but one had a fixed 4-tap feedback and the other was configurable on the fly. The configurable module allows the user to select the number of taps and which stages were tapped at any given time by programming the enable signals to the chip. With a different set of taps, the LFSR will model a different function in the output sequence generated. The fixed method obviously performs faster as the feedback path is compressed into three XOR gates in two levels. The reconfigurable LFSR would be good in encryption/decryption systems so that multiple keys could be generated with a single LFSR module and encryption functions could be changed on the fly without redesigning the module.

The I/O MUX presented in Chapter 4 was designed not only to connect the input and output lines of each LFSR stage to the integrated circuit (IC) pins, but also to switch these same IC pins between the fixed and the reconfigurable LFSR. Since all of the stages were connected to the same pins, only one of the LFSR modules could be running at a time. The clock signal is only passed to the circuit of choice, thus enabling the system to use less power than it would if both LFSRs ran regardless of which module was connected to the IC pins.

The results of the LFSR using the hybrid wave-pipelining scheme, as well as selected performance comparisons were shown in Chapter 5. The original hybrid wave-pipelined clock was shown to have a very fast response time in terms of delay between reference clock and generated clock. However, the original design for this method experienced a floating node in the generation system that would cause problems at slower speeds. A revised design for the hybrid wave-pipelined clock improved upon this floating node and increased the peak amplitude of the generated clock but at a cost of slower response time between reference and generated clocks. An improved circuit over the first two presented a new hybrid wave-pipelined clock design that triggered on both edges of the reference clock. This method also improved the response time between reference and generated clocks over the revised method.

Also presented in Chapter 5 was a software technique to generate the LFSR sequence automatically. This software prediction scheme reduces the likelihood of human error in calculating the expected sequence by hand. The verification process is very simple and a sample output window using the software was provided. An output file can be setup using the software interface to log the sequence. This file could be used with another program that extracts the simulation output and does an automatic compare for quicker testing on larger LFSR systems that would be difficult to test by hand.

6.2 Contributions

The contributions of the work contained within this thesis are as follows:

- Shown that hybrid wave-pipelining can be mapped to systems with feedback using an LFSR as a testbed.
- Hybrid wave-pipelining can alleviate clock skew.
- Wave-pipelining the clock would result in improved control of clock skew since the clock travels with data and thus experiences the same delays.
- Combining the best of Conventional and Wave-pipelining techniques results in improved performance as seen in the reduced skew (6 times).

6.3 Future Work

Future work based on the results of this thesis could include:

- Address the shortfalls of the wave-pipelined clock presented in Chapters 4 and 5, such as the inability to have a uniform duty cycle.
- Design an LFSR for a general or specific application with quantitative power analysis compared to existing designs.
- Explore the potential for the scheme's clock power reduction.
- Explore hybrid wave-pipelining techniques using combination logic systems with feedback.

Bibliography

[1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd edition, Morgan Kaufmann Publishers, 2003.

[2] C. Thomas Gray, Wentai Liu and Ralph K. Cavin, III, *Wave Pipelining: Theory and CMOS Implementation*, Kluwer Academic Publisher, 1994.

[3] B. Ekroot. *Optimization of Pipelined Processsors by Insertion of Combinational Logic Delay.* PhD dissertation, Stanford University, 1987.

[4] J. Nyathi. *A Flexible High-Performance Network Router with Hybrid Wave-Pipelining.* PhD dissertation, Binghamton University, 2000.

[5] W. Burleson, L.W. Cotten, F. Klass and M Ciesielski, "Wave-pipelining: Is it Practical?," *1994 IEEE International Symposium on Circuits and Systems*, Vol. 4, pp. 163-166, June 1994.

[6] W. Burleson, F. Klass and W. Liu, "Wave-pipelining: A tutorial and survey of recent research," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 6, Issue 3, pp. 464-474, Sept. 1998.

[7] J. Nyathi and J.G. Delgado-Frias, "A Hybrid Wave-Pipelining Network Router," *IEEE Trans. on Circuits and Systems*, Vol. 49, Issue 12, pp. 1764-1772, Dec. 2002.

[8] M. Goresky and A. M. Klapper, "Fibonacci and Galois Representations of Feedback-With-Carry Shift Registers," *IEEE Trans. on Information Theory*, Vol. 48, No. 11, pp. 2826-2836, Nov. 2002.

[9] J. Nyathi, J.G. Delgado-Frias and J. Lowe, "A High-Performance, Hybrid Wave-Pipelined Linear Feedback Shift Register with Skew Tolerant Clocks," *46th IEEE Midwest Symposium on Circuits and Systems*, Cairo, Egypt, In Press, Dec. 2003.

[10] New Wave Instruments. (2002, June 21). Linear Feedback Shift Registers: Implementation, M-Sequence Properties, Feedback Tables. Available: http://www.newwaveinstruments.com /resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm

[11] Y. Shi and Z. Zhang, "Multiple Test Set Generation Method for LFSR-Based BIST," *Proceedings of the ASP-DAC 2003*, pp. 863-868, Jan. 2003.

[12] N. Lai and S. Wang, "A Reseeding Technique for LFSR-Based BIST Applications," *Proceedings of the 11th Asian Test Symposium*, pp. 200-205, Nov. 2002.

[13] P. Kitsos, N. Sklavos, N. Zervas and O. Koufopavlou, "A Reconfigurable Linear Feedback Shift Register (LFSR) for the Bluetooth System," *8th IEEE International Conference on Circuits and Systems*, Vol. 2, pp. 991-994, Sept. 2001.

[14] A. Mostafa and A. Omar, "Complexity Measure of Encryption Keys Used for Securing Computer Networks," *Proceedings of the 14th Annual Computer Security Applications Conference*, pp. 250-255, Dec. 1998.

[15] M. George and P. Alfke, "Linear Feedback Shift Registers in Virtex Devices (application note)" http://www.xilinx.com/bvdocs/appnotes/xapp210.pdf

[16] http://www.sss-mag.com/pdf/lfsr.pdf "Linear Feedback Shift Register Megafunction," Version 1, Dec. 1996.

[17] A. Iyer and D. Marculescu, "Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors," *Proceedings, 29th Annual International Symposium on Computer Architecture*, pp. 158-168, May 2002.

[18] V. Mehrotra and D. Boning, "Technology Scaling Impact of Variation on Clock Skew and Interconnect Delay," *Proceedings of the IEEE 2001 Interconnect Technology Conference*, pp. 122-124, June 2001.

[19] R. Y. Chen, N. Vijaykrishnan and M. J. Irwin, "Clock Power Issues in System-on-a-Chip Design," *Proceedings IEEE Computer Society Workshop on VLSI*, pp. 48-53, Apr. 1999.

Appendix A

Software Prediction Code

```
/*************************************************
* Jeff Lowe
*   Graduate Student, Washington State University
*
* Program to calculate the length of the LSFR series
* or to predict the next outcome given any HEX input.
* Output prediction is based of XOR logic in the
* feedback.
*
* Created: 4/24/2003
*************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>

/*************************************************
* Following are explanations for use of this file
*
* stages:
*     The # of stages included in the feedback.
*     (ie. last feedback tap = last stage)
*
* ntaps:
*     The # of taps used in the equations.
*
* bit[ntaps]:
*     For each feedback tap, define the stage
*     number that the tap occurs at.  Order does
*     not matter.
*************************************************/

//values for 16 stages
//#define stages 16
//#define ntaps 4
//int taps[] = {16, 14, 13, 5};

#define maxtaps 32
```

```
void main(void){
   int i, count, Fsave = 0;
   FILE *file;
   unsigned int input, output;
   char cont = 'a', whattodo = 'a';
   char filename[64];
   unsigned int length = 0, start;
   unsigned int stagemask = 0, bit[maxtaps];
   int stages = 0, ntaps;
   int taps[maxtaps];

   while (stages <= 0){
      printf("Enter number of stages: ");
      scanf("%d", &stages);
   }
   printf("Enter number of taps: ");
   scanf("%d", &ntaps);
   for (i = 0; i < ntaps; i++){
      printf("Enter tap %d: ", i+1);
      scanf("%d", &taps[i]);
   }

   for (i = 0; i < stages; i++){    // build bitmask for # of stages
      stagemask <<= 1;             // shift left by 1 position
      stagemask++;                 // add a bit (stage)
   }

   for (i = 0; i < ntaps; i++){     // build bitmasks for each tap
      bit[i] = 1;                  // start at first stage
      bit[i] <<= taps[i] - 1;      // shift by (n-1) stages
   }

   while ((tolower(whattodo) != 'm') && (tolower(whattodo) != 'p')){
      printf("(m)seq length or (p)redict following outcome: ");
      whattodo = getch();
      printf("\n");
      fflush(stdin);
   }
   if (tolower(whattodo) == 'm'){
      printf("Save outputs to file (y/N)? ");
      cont = getch();
      printf("\n");
      fflush(stdin);
      if (tolower(cont) == 'y'){
           printf("file name: ");
           scanf("%s", filename);
           file = fopen(filename, "w");   // open file for write
           if (file == NULL){
              printf("Error opening file");
              exit(1);
           }else{
              printf("File %s opened successfully\n", filename);
              Fsave = 1;
           }
      }
  }
```

```
   printf("Input (Hex): ");
   scanf("%x", &input);
   input &= stagemask;
   start = input;

   fflush(stdin);
   if (tolower(whattodo) == 'p'){
      printf("values are printed as HEX(DECIMAL)\n");
      printf("<enter> to advance, <e>nter new input, <q> to exit\n");
   }

   while (tolower(cont) != 'q'){
      count = 0;
      for (i = 0; i < ntaps; i++){
           if (input & bit[i])
              count++;
      }
      if (count % 2){    // if count is odd, remainder from modulus exists
         count = 1;              // if odd => XOR feedback = 1
      }else{
         count = 0;
      }
      if (Fsave == 1){
           fprintf(file, "%d, %d\n", input);
      }
      output = input << 1;     // shift input 1 bit left
      output &= stagemask;     // mask out any extra bits
      output += count;         // add feedback bit

      if (tolower(whattodo) == 'p'){
           printf("Cur %x(%d) : In %d : Out %x(%d)\n", input, input, count, output, output);
           printf(":");
           cont = getch();
           printf("\n");
           if (tolower(cont) == 'e'){
              printf("Input (HEX): ");
              scanf("%x", &output);
              output &= stagemask;
           }
           fflush(stdin);
      }else{
           if (start == output){
              printf("length of mseq = %d\n", length+1);
               if (Fsave == 1){
                  fprintf(file, "%d, %d\n", output);
               }
              cont = 'q';  // set flag to exit
           }else
              length++;
      }
      input = output;
   }
   if (Fsave == 1)
      fclose(file);
}
```

Appendix B

Layouts

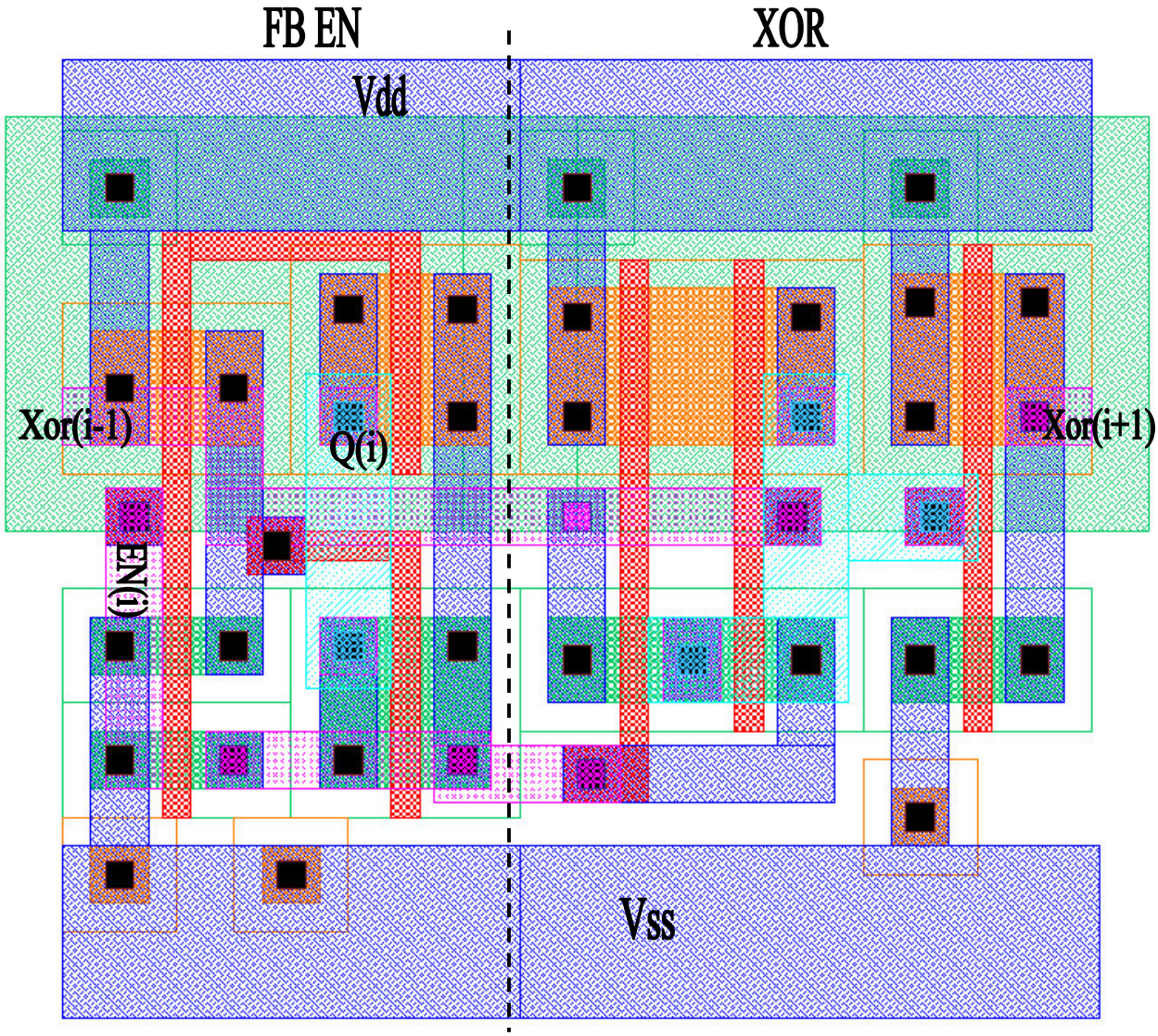

Figure B-1: Six Transistor XOR with Feedback Enable Logic

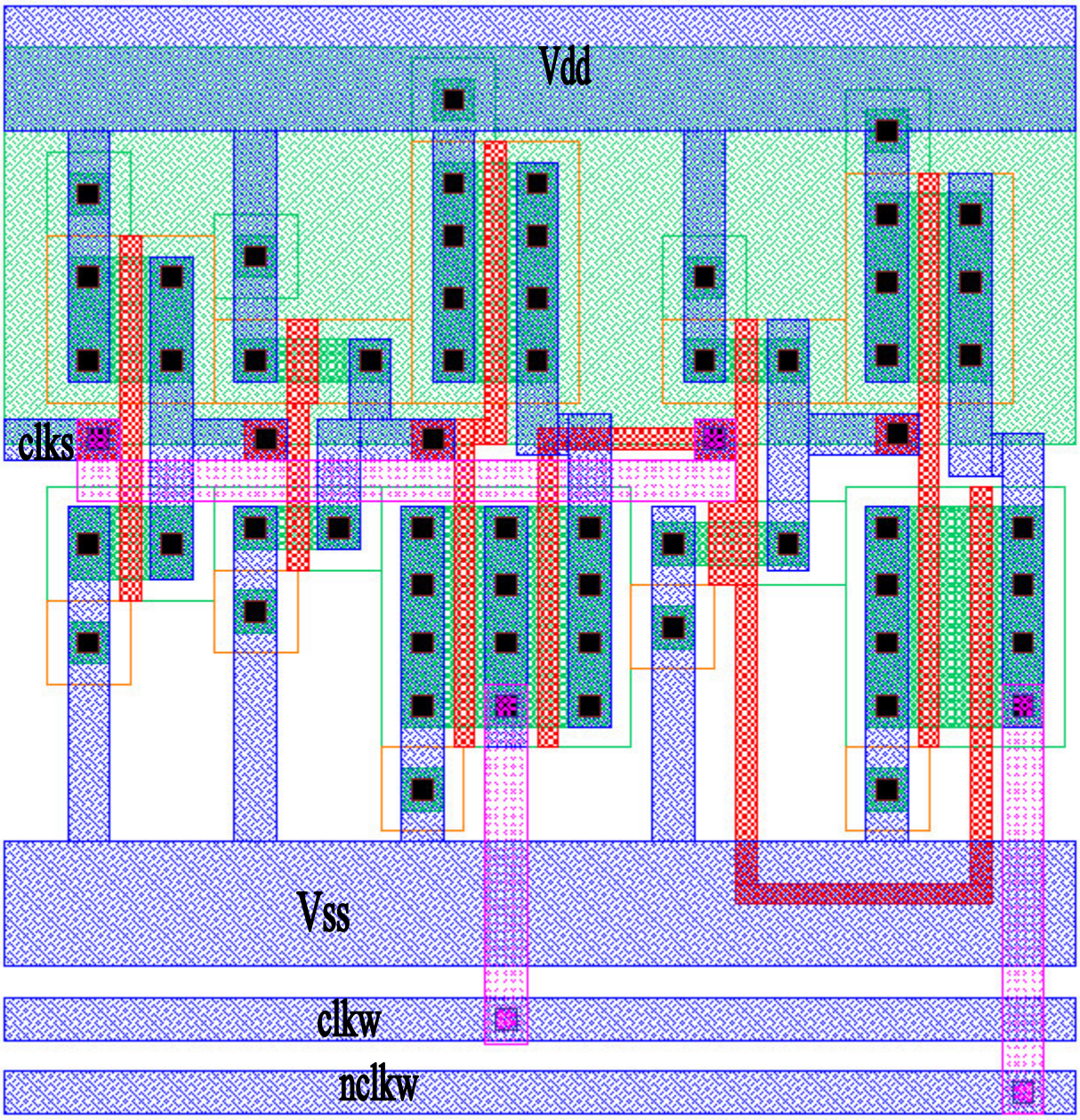

Figure B-2: Hybrid Wave-Pipelined Clock Generator

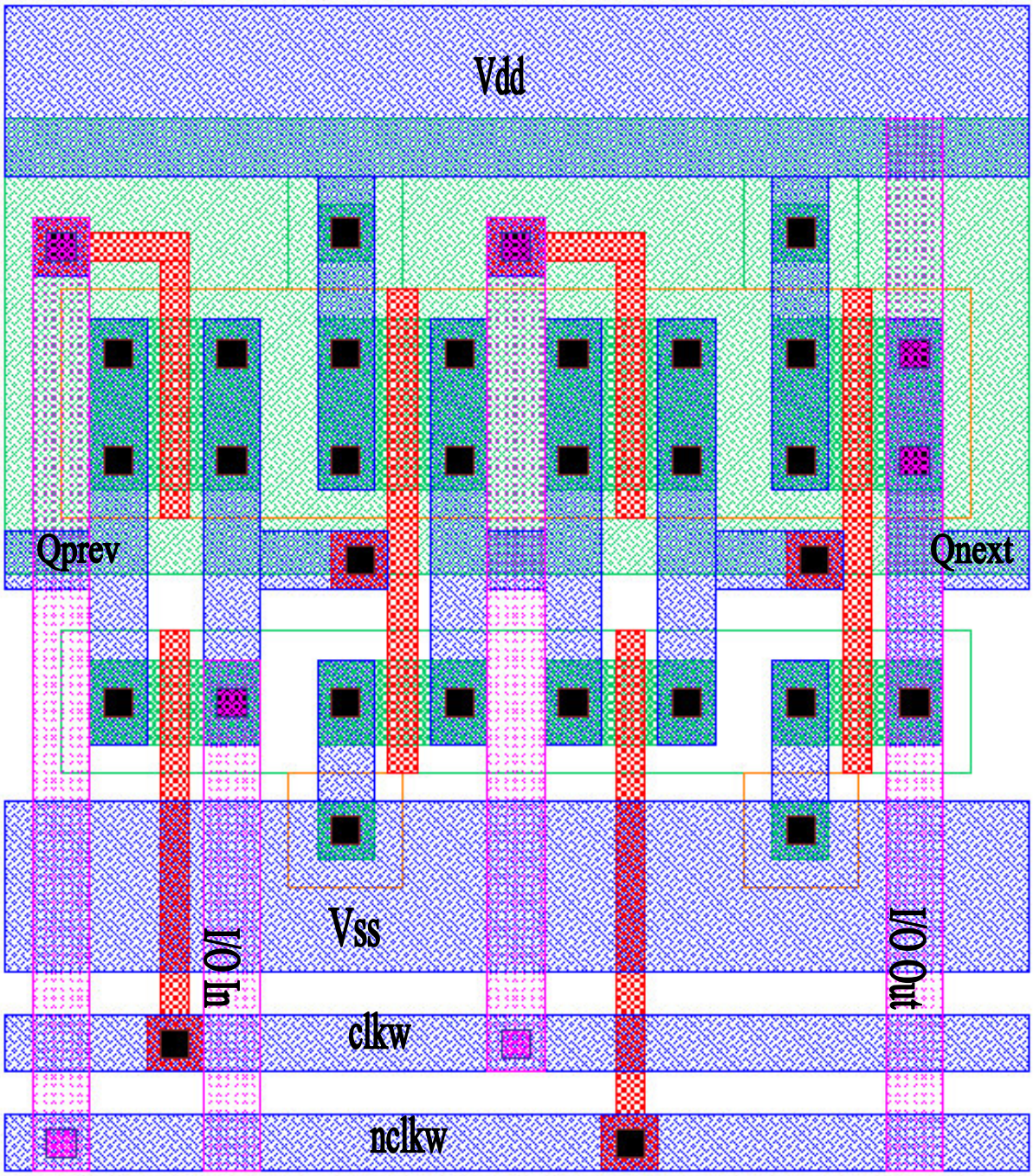

Figure B-3: Single Linear Feedback Shift Register Stage

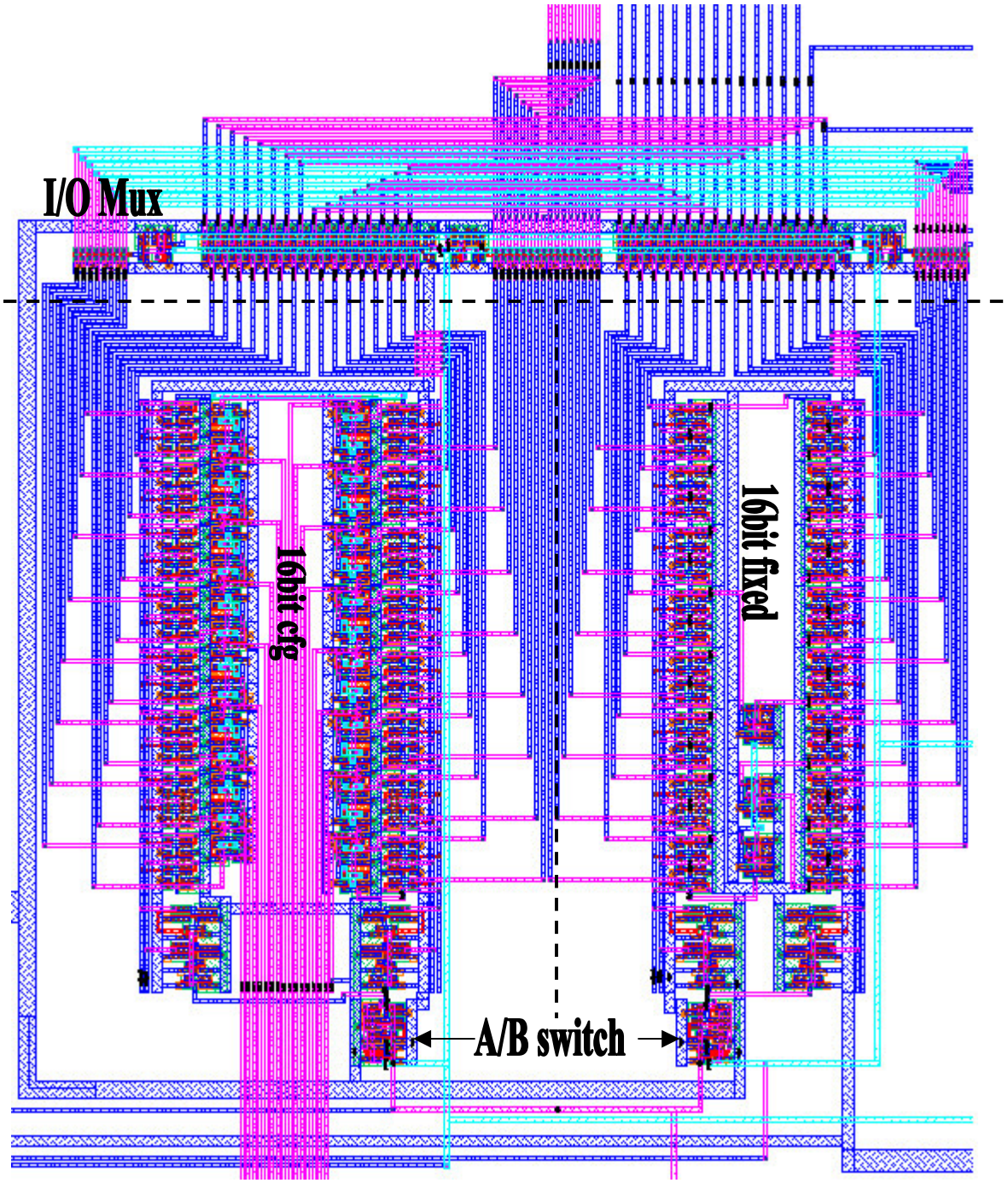

Figure B-4: Fixed/Configurable Feedback Linear Feedback Shift Register Module