A 45nm CMOS, Low Jitter, All-Digital Delay Locked Loop with a Circuit to Dynamically Vary Phase to Achieve Fast Lock

A Thesis Presented
by
Soumya Shivakumar Begur
to
The Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

October 2011
Thesis Title: A 45nm CMOS, Low Jitter, All-Digital Delay Locked Loop with a Circuit to Dynamically Vary Phase to Achieve Fast Lock.

Author: Soumya Shivakumar Begur.

Department: Electrical and Computer Engineering.
Abstract

The objective of the thesis is to address the problem of clock skew between two different clock domains in modern day microprocessors due to the process, voltage and temperature (PVT) variations. In order to mitigate the misalignment of the clocks in the different clock domains, a delay line is added in all but the reference clock domain. These delay lines add or subtract the delay (as necessary) to keep the clocks continuously aligned to a common reference clock delay. This ensures error free data transfer between any two clock domains.

A novel Digital DLL design is proposed to achieve short locking time by having a separate circuitry for fast lock in the DLL. The fast lock mechanism can be switched off and the power dissipation can be returned to normal levels after the DLL has locked. A separate fine-delay block makes the proposed DLL have ultra-low jitter after lock. The results show that the proposed DLL, implemented in 45nm CMOS technology, needs only 24 cycles to correct (i.e. to lock) for 500ps clock skew when compared to more than 38 lock cycles for a conventional DLL without the fast lock mechanism. A Monte-Carlo simulation yielded a RMS jitter and peak-to-peak jitter values after lock of 5.26ps and 10.57ps respectively. Average power consumption before lock is <425µW with this number falling to <335µW after lock. The frequency of operation of the proposed DLL is 280MHz - 1.63GHz and can be used for a variety of applications which require precise time intervals.
Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Yong-Bin Kim for his guidance and encouragement. I am extremely grateful for his help and support throughout my graduate school experience. Without his valuable suggestions and assistance, this thesis would not have been possible.

I would like to thank my brother-in-law, Mr. Sompur Shivakumar for his constant encouragement and evincing keen interest in my thesis. I am grateful for his valuable advice, motivation and support throughout my graduate career.

I would also like to thank my friends for their suggestions and support.

Finally, I would like to express my profound thanks to my beloved parents, Dr. B. S. Shivakumar and Kalpana Shivakumar, and my sister, Dr. Shilpa Shivakumar, who encouraged me continuously and stood by me throughout my studies. I would not have made it through the toughest times without their support.

Soumya S. Begur
October 2011
Contents

1 Introduction 1
   1.1 Delay Locked Loop Overview ........................................ 3
       1.1.1 Analog DLL .................................................. 4
       1.1.2 Digital DLL .................................................. 5
       1.1.3 Comparison between Analog DLL and Digital DLL .......... 5
   1.2 Conclusion .......................................................... 6

2 Design of a Digital DLL 7
   2.1 Phase Detector ..................................................... 8
   2.2 Control Mechanism .................................................. 10
   2.3 Digital Controlled Delay Line ...................................... 11
       2.3.1 Coarse Delay Line ............................................. 11
       2.3.2 Fine Delay Line ............................................... 13
   2.4 Conclusion .......................................................... 15

3 Design of Proposed Fast Lock Digital DLL 16
   3.1 Design Flow of the DLL controller ................................. 17
   3.2 Phase Detector ..................................................... 19
   3.3 Controller ........................................................... 21
       3.3.1 Binary Up-Down Counter ...................................... 21
           3.3.1.1 5-Bit Up-Down Counter ................................ 21
           3.3.1.2 4-Bit Up-Down Counter ................................ 26
       3.3.2 Fast Lock Mechanism .......................................... 30
           3.3.2.1 Fast Lock Block ......................................... 31
           3.3.2.2 Encoder .................................................. 37
           3.3.2.3 Encoder Flip-Flop ...................................... 39
       3.3.3 Adder ........................................................... 39
       3.3.4 Lock Detector .................................................. 40
       3.3.5 Adder Flip-Flop block ........................................ 42
       3.3.6 Select .......................................................... 42
       3.3.7 Decoder ........................................................ 43
   3.4 Digital Controlled Delay Line ..................................... 46
       3.4.1 Coarse Delay Line ............................................. 46
       3.4.2 Fine Delay Line ............................................... 47
           3.4.2.1 Fine Delay Block ....................................... 48
           3.4.2.2 16-bit Multiplexer ................................... 51
3.5 Conclusion ................................................................. 52

4 Results and Performance Analysis 53

5 Conclusion and Future Works 61
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Clock distribution in a Microprocessor</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>DLL implementation</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Conventional Analog DLL</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Conventional Digital DLL</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>D-flip flop based phase detector [1]</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Dual output phase detector [2]</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Dynamic phase detector [3]</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Control mechanism in Digital DLLs [4]</td>
<td>10</td>
</tr>
<tr>
<td>2.5</td>
<td>Finite state machine based control block in Digital DLLs [4]</td>
<td>11</td>
</tr>
<tr>
<td>2.6</td>
<td>Inverter based digital controlled coarse delay line [4]</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>NAND based digital controlled coarse delay line [4]</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>MUX based digital controlled coarse delay line [5]</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>Typical digital controlled fine delay line [4]</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Thermometric/logarithmic digital controlled fine delay line [4]</td>
<td>14</td>
</tr>
<tr>
<td>2.11</td>
<td>Phase blending circuit as the fine delay line and its output waveforms [6]</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Counter-controlled Digital DLL</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Proposed Digital DLL</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Proposed Digital DLL</td>
<td>18</td>
</tr>
<tr>
<td>3.4</td>
<td>Block diagram of the Phase detector</td>
<td>19</td>
</tr>
<tr>
<td>3.5</td>
<td>Schematic of the Phase detector</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Input and output waveforms of the Phase detector</td>
<td>20</td>
</tr>
<tr>
<td>3.7</td>
<td>Block diagram of 5-bit counter</td>
<td>22</td>
</tr>
<tr>
<td>3.8</td>
<td>K-maps for the flip-flop inputs, a) $DA_0$, b) $DA_1$, c) $DA_2$, d) $DA_3$ and e) $DA_4$</td>
<td>23</td>
</tr>
<tr>
<td>3.9</td>
<td>Schematic of 5-bit up-down counter of the DLL</td>
<td>25</td>
</tr>
<tr>
<td>3.10</td>
<td>Output of 5-bit up-down counter</td>
<td>26</td>
</tr>
<tr>
<td>3.11</td>
<td>Block diagram of 4-bit counter</td>
<td>27</td>
</tr>
<tr>
<td>3.12</td>
<td>K-maps for the flip-flop inputs $DA_0$, $DA_1$, $DA_2$ and $DA_3$</td>
<td>28</td>
</tr>
<tr>
<td>3.13</td>
<td>Schematic of 4-bit up-down counter of the DLL</td>
<td>29</td>
</tr>
<tr>
<td>3.14</td>
<td>Output of 4-bit up-down counter</td>
<td>30</td>
</tr>
<tr>
<td>3.15</td>
<td>Block diagram of Fast Lock Mechanism</td>
<td>31</td>
</tr>
<tr>
<td>3.16</td>
<td>Fast Lock Block</td>
<td>32</td>
</tr>
<tr>
<td>3.17</td>
<td>Fast Lock Block</td>
<td>32</td>
</tr>
<tr>
<td>3.18</td>
<td>Fine Delay Unit</td>
<td>33</td>
</tr>
<tr>
<td>3.20</td>
<td>Output of FLBM for a phase difference of 200ps</td>
<td>34</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 State table for the 5-bit up-down counter using DFF . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
3.2 State table for the 4-bit up-down counter using DFF . . . . . . . . . . . . . . . . . . . . . . . . . . . 27  
3.3 Truth table for 8-bit priority encoder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38  
3.4 Control inputs T[2:0] and the corresponding delay of CDU . . . . . . . . . . . . . . . . . . . . . . . . . . . 47  
4.1 PD and corresponding lock cycle values of the proposed DLL . . . . . . . . . . . . . . . . . . . . . . . . . . 58  
4.2 Performance comparison of the proposed DLL with other Analog DLLs . . . . . . . . . . . . . . . . . . . . . 59  
4.3 Performance comparison of the proposed DLL with other Digital DLLs . . . . . . . . . . . . . . . . . . . . . 60
Chapter 1

Introduction

Modern day microprocessors have multiple clock domains with each clock domain being restricted to a block. At any instance in time, some of them are powered down (to minimize power dissipation) while some of them are up and running depending on the scenario of operation. They are always supplied by the same Phase locked loop (PLL) [8]. Figure 1.1 shows the clock distribution in a microprocessor.

![Figure 1.1: Clock distribution in a Microprocessor.](image)

When the data is transferred between block-2 and block-n by means of flip flops, it is
important that the clocks clocking the flip flops be synchronized, so that block-n receives the data sent from block-2 and vice versa reliably. One of the main reasons why the data transfer may fail is if the clocks supplying those flip-flops are misaligned due to clock skew brought on by Process, voltage, temperature variations or due to the differing delays of the clock distribution networks in either of these blocks. While the later can be corrected before taping-out a chip, the former is harder to correct. To alleviate this, Delay locked loops (DLL) are used to synchronize the clocks continuously in domain-2 and domain-n. In a System on Chip (SoC) environment, the block-1, block-2, etc. may be CPU instances.

![Diagram of DLL implementation](image)

**Figure 1.2: DLL implementation.**

In this thesis we are trying to equalize the delay @1 to @2 and @1 to @3 as in the Figure 1.2. A novel Digital DLL design is proposed to achieve short locking time by having a separate fast lock mechanism in the DLL controller. The fast lock mechanism can be switched off and
the normal power dissipation can be achieved after the DLL has locked. The proposed DLL also has less jitter after lock and less residual, uncorrectable offset. The thesis is organized as following: In Chapter 1 the need for the DLLs, their working principle and the types of DLLs is given. In Chapter 2 detailed schematic analysis of a Digital DLL is given. The modified DLL architecture of digital DLL is described in Chapter 3. Finally, the results are discussed in detail and compared with the results of other DLLs in 4.

1.1 Delay Locked Loop Overview

The reduction of clock skew is one of the important problems in the VLSI design. DLLs are used in VLSI circuits in order to decrease clock skew in the clock networks. DLL is a feedback system that aligns the feedback clock to the reference clock. This is done by delaying the input feedback clock after passing it through a delay line and controlling the delay using the control mechanism. Once the input feedback clock is delayed, a phase detector (PD) compares the phases of the two inputs. Based on PD output value, the delay is adjusted (increased or decreased) until the two phases are aligned. A DLL is widely used as a timing circuit in many systems for the purpose of clock generation [9], [10], signal synchronization [11], and others [12]. For example, a DLL is able to provide multiple clock signals which are separated from each other by a well-controlled phase shift (delay). This application of DLL is widely used in DDR memory circuits to align the DQS strobe in the middle of the data eye DQ. When appropriate logic, such as edge combining is used, a new clock signal which is of a different frequency can be generated by the DLL. Such an application of a DLL has been reported in [13] for personal communication services (PCS). Another application of a DLL is for the purpose of clock deskewing in synchronous data transfer among communication chips. Reducing clock skew has become increasingly important with larger die size and higher clock frequency. A DLL for this application requires fast lock time and good phase alignment between the reference signal and the corrected output signal. A DLL serving as a clock deskwk buffer was reported in [14] and is also discussed in this thesis. In addition, a DLL can also be used for signal synchronization, for example, between a CPU and a co-processor so that they can share the same data bus. This application was reported in [15].
Many of the above applications share similar requirements for the DLL, such as a short locking time, low jitter, and a wide locking range [2]. DLLs can be either primarily analog [14], [16] or primarily digital [17], [18] depending on the design components. Both of these are discussed in the next sections.

1.1.1 Analog DLL

Analog DLLs were first used in clock distribution applications [9], [16]. The components of a conventional Analog DLL are, phase detector, charge pump, low pass filter and voltage controlled delay line as shown in Figure 1.3.

A phase detector compares two input signals and generates "UP" and "DN" output pulses that represent the direction and magnitude of the input phase error. There are many types of phase detectors; they differ in how they sense the input signals, what input phase difference would result to detect no phase error, and how the phase error is represented in the output pulses. Considering simple type of phase detectors such as the phase-frequency detectors are those which are only rising and falling edge sensitive. They produce a single pulse at the UP or DN output for each pair of input reference and feedback edges depending on which edge arrives first. A charge pump, connected to the phase detector, sources or sinks current for the duration of the UP and DN pulses from the phase detector. The net output charge is proportional to the difference between the pulse widths of the UP and Dn outputs. The charge pump drives the loop filter which integrates and filters the charge current to produce the control voltage. The control Voltage drives a voltage-controlled delay line (VCDL) in a DLL which generates a delay proportional to the control voltage [19].
1.1.2 Digital DLL

Digital DLLs are characterized by their use of digitally controlled delay line. They are typically made of simple digital circuit elements. Figure 1.4 shows a conventional Digital DLL. The simplicity of the digital circuit elements helps to design portable digital DLL which can be adapted to different technologies. The conventional digital DLLs provide only moderate phase resolution, lock time and jitter performance [17], [18], [20], [21], [6], [22], [23], [24].

![Digital DLL Diagram](image)

Figure 1.4: Conventional Digital DLL.

The control logic controlling the delay line has been implemented in different ways. An example of the Digital DLL with one of the designs of control logics will be discussed in Chapter 2.

1.1.3 Comparison between Analog DLL and Digital DLL

An analog DLL is a complex analog circuit requiring process-specific implementation. Scaling of analog integrated circuits directly affects output resistance and intrinsic gain of the circuit. As we cross below the 100nm mark, the design of these analog circuits becomes quite exigent, especially for low supply voltages around 1 V. This results in higher power consumption and enhanced design complexity. [25]. So, it is not practical to reuse the same design for different technology, making analog DLL a non-portable architecture. Analog DLL generally provides better jitter performance at the expense of greater complexity [24]. Analog DLLs are more susceptible to process variations and less immune from power-supply noise because of smaller noise margins. In an analog DLL, if the low pass filter has less capacitor value then this will result in more jitter and thus providing more variation in Voltage controlled delay line. If the
capacitor value is increased to achieve less jitter then the area of the DLL increases. While the analog circuits may be adversely affected by scaling, digital circuits become increasingly efficient by operating at higher speeds, consuming less power and occupying smaller area [25]. Although the Digital DLL has more Jitter than Analog DLL, its greater simplicity, portability, lower power consumption, its ability to operate at lower voltages, smaller area, more noise immunity, short lock time and synthesizable circuit makes it very attractive for clock alignment applications [26], [27], [28].

1.2 Conclusion

In this chapter general information about DLL and the types of DLL is described. The advantages and disadvantages of the Analog and Digital DLL are also discussed. The common requirements for the design of a DLL are short locking time, low jitter performance, less power consumption, small area and less phase error after lock or minimum time of resolution.

In the next chapter, a detailed discussion of the design of a Digital DLL is given.
Chapter 2

Design of a Digital DLL

The Digital DLLs are characterized by the controller that is used to control the delay line. Digital DLLs are roughly divided into four categories. The first one is the Register-Controlled DLL. The locked time and the number of delay cells increase exponentially as the number of control bits increases. The second category is the Counter-Controlled DLL [3], [17]. The counter-based controller [3], [17] replaces the register-based one to reduce the hardware of the controller. However, the locked time and the required delay cells of the counter-controlled DLL are similar to those of the register-controlled DLL. The third category is the DLL using the Successive Approximation Register-Controlled (SAR) scheme [29]. It can reduce the locked time by using the binary search algorithm. The last category is the DLL using the Time-to-Digital Converter (TDC) [30]. Large chip area and power are required especially when it operates in the wide frequency range. Both TDC and SAR schemes are open-loop. The open-loop characteristic is hard to track the process, voltage, temperature and loading (PVT) variations. Besides, the wide range DLLs usually needs an external reset signal when the frequency of input clock changes significantly [28].

In this thesis, a Counter-Controlled DLL has been considered and the fast lock mechanism has been added to the control logic to reduce the lock time of the DLL. Each component of the conventional Counter-controlled digital DLL is discussed in detail in the following sections.

Digital DLL is composed of the following main component blocks.

• Phase detector (PD)
• Control mechanism (CM)
• Digital controlled delay line (DCDL)

2.1 Phase Detector

In a digital DLL the output of the phase detector is a single signal (up/down) which drives the subsequent counter or FSM depending upon the application of DLL. Generally a PD must satisfy the following design requirements:
• To avoid false detection the PD should spend minimum time in metastable state.
• The setup and hold times must be reasonably small.
• The setup and hold times should be comparable so as the resulting phase information is not biased.
• The clock to Q delay should be reasonably small but is usually not a stringent requirement.

Based on the different applications, different PDs have been proposed and implemented [31], [2], [3]. Few types of PD designs are discussed here. The simplest of the phase detectors can be a single D flip flop (DFF) as shown in Figure 2.1. The DFF is clocked by the input reference clock and detects the difference between the clocks by selecting the Q output of the DFF. If the Q output is low it means that the output clock (clock signal form the end of digitally controlled delay line) is leading the input reference clock and thus the delay of the delay line must be increased so as to compensate for the difference in phase until the lock condition is achieved [1].

Figure 2.1: D-flip flop based phase detector [1].
Though this design is simple and effective, it has shortcomings. If the setup and hold times are not symmetric this may introduce a regular phase error as lagging and leading decisions based on the hold and setup times, respectively.

Figure 2.2: Dual output phase detector [2].

Figure 2.3: Dynamic phase detector [3].

Figure 2.2 shows another implementation of PD which doesn’t have the flaws of PD in Figure 2.1. This PD consists of two identical blocks generating UP or DOWN signal. Within each block there are two stages with a pre-charge PMOS in each stage. Output of the first stage
controls the second stage pre-charging.

Figure 2.3 shows another PD design. To align signal phases with precision, it is important to reduce dead zones in the PDs. The dynamic PD in Figure 2.3 improves the phase sensitivity of Latch 1 because sharper waveforms of the reference clock and the feedback clock are obtained by using precharged dynamic circuits. Latch 2 is connected to Latch 1 to hold the signal states.

2.2 Control Mechanism

The controlling mechanism of the digital DLL also varies based on the application of the DLL. Counter-Controlled DLLs could be implemented in two ways. Figure 2.4 and Figure 2.5 shows these two implementations.

A straightforward control mechanism is shown in Figure 2.4. In this design, the PD controls an up/down counter that is used to set the delay of the DCDL so as to compensate for the phase difference between the PD inputs. When the output of the PD is high, the counter is incremented, increasing the delay of DCDL and when PD output is low, the counter is decremented, to decrease the delay of DCDL, thus attaining the lock condition [1].

![Control mechanism in Digital DLLs](image)

The Figure 2.5 shows an advanced control mechanism in which a finite state machine (FSM) is placed between the PD and the up/down counter. Here, the FSM sends a high or low signal to the input of the counter not only based on the PD output but also on the internal state of the FSM, which is set based on the initial conditions of the system. The design of such an FSM
CHAPTER 2. DESIGN OF A DIGITAL DLL

is very complex and may directly affect the dynamic range and stability of DLL [1].

Figure 2.5: Finite state machine based control block in Digital DLLs [4].

2.3 Digital Controlled Delay Line

The delay line is the most important component of the DLL and has a profound effect on the overall performance of the DLL. Commonly, digital DLLs are composed of two delay lines: a coarse delay line and a fine delay line. The coarse delay line is also referred to as Gate-delay DCDLs while the fine delay line is also referred to as Subgate-delay DCDL. There are various number of delay line structures that are employed or that have been proposed for different applications. The following section briefly discusses few of the commonly used DCDL structures.

2.3.1 Coarse Delay Line

The coarse delay lines are composed of CMOS logic gates and are cascaded to form a delay line. The simplest of the delay lines for digital DLL is a chain of cascaded inverters with each stage consisting of a pair of inverters and the required output tap is selected by a multiplexer. The minimum delay of each stage is $2T_D$ where $T_D$ is the average CMOS gate delay and requires $\log_2 N$ storage elements where N is the number of stages [1]. A coarse delay line design discussed in [4] is shown in the Figure 2.6.

This implementation has four delay stages between the input and output. If the number of delay stages is increased as per the design requirement, this may load the input clock. To
overcome the loading, separate buffers must be introduced thus increasing the delay of each cell to more than $2T_D$. This increase in delay would depend on the amount of added buffering [4], [1].

Figure 2.6: Inverter based digital controlled coarse delay line [4].

Figure 2.7: Nand based digital controlled coarse delay line [4].
A work-around to this problem is shown in Figure 2.7. In this structure, the loading on the input clock is prevented by varying the delay line in a telescopic fashion. Adapting this structure eliminates the need to introduce internal buffering of the clock signal thus maintaining the delay of each stage to $2T_D$. A shift register controls the delay of each cell of these cells. For example, when $Q[0] = 1$ and $Q[3:1] = 0$ the output clock signal would have a two nand gate delay i.e. the delay of A and B nand gates in cell 0. When $Q[1] = 1$ and the rest of the values are low the output path would be nand gate C in cell 0, nand gate A in cell 1, nand gate B in cell 1, nand gate B in cell 0. The wrap around in the delay cell 3 toggles the delay line between various modes [4], [1].

Figure 2.8 shows another implementation of coarse delay line. Here, the delay line consists of multiple delay cells connected as shown in the Figure 2.8. Each delay cell consists of a multiplexer (MUX) and a buffer. The input1 of the MUX of first delay cell is connected to vdd and for all other delay cells the input1 is connected to the output of the previous stage cell. The input2 of all the MUX are connected to clock. The delay line is controlled through the select lines which are the output of the control logic.

![Figure 2.8: MUX based digital controlled coarse delay line [5].](image)

In this thesis, we are using similar MUX based digital controlled delay line.

### 2.3.2 Fine Delay Line

Two designs of the fine delay line, which generate delay by using the RC delay characteristics are shown in Figure 2.9 and Figure 2.10. The structure in Figure 2.9 relies on varying cell resistance. The branches are controlled by using digital bits $Q[2:0]$ which switch in a fixed specified pattern. The bits can only be thermometrically encoded for this circuit due to poor
linearity behavior of the circuit. For increased dynamic range several stages can be cascaded. The greater the number of devices that are ON at a particular time smaller will be the RC delay.

![Typical digital controlled fine delay line](image1)

Figure 2.9: Typical digital controlled fine delay line [4].

In the second design of the fine delay shown in Figure 2.10, the RC delay increases with the number of ON devices. The digital control bits can be encoded using both thermometric and logarithmic codes due to linear characteristics of the circuit. As in former structure the dynamic range increases by cascading several stages [1]. These RC based delay lines are discussed in more detail in [4].

![Thermometric/logarithmic digital controlled fine delay line](image2)

Figure 2.10: Thermometric/logarithmic digital controlled fine delay line [4].

The design proposed in [6] also helps to have very fine phase resolution which can be used as the fine delay line. This design uses the inverter phase blending mechanism as shown in
the Figure 2.11. This circuit receives two phase adjacent input signals, $\Phi_A$ and $\Phi_B$, which are separated in phase by one inverter delay. The phase blender directly passes these two signals with a simple delay to produce output signals $\phi_A$, $\phi_B$ and it also uses a pair of phase-blending inverters to interpolate between these two input signals to produce a third output signal, $\phi_{AB}$, having a phase between that of $\phi_A$ and $\phi_B$ and this effectively doubles the available phase resolution. The graph in Figure 2.11 shows the output phase blended waves. Phase blending mechanism is discussed in detail in [6].

![Diagram of phase blending circuit]

Figure 2.11: Phase blending circuit as the fine delay line and its output waveforms [6].

2.4 Conclusion

In this chapter detailed information about the Digital DLL and previous Digital DLL component designs is provided. These designs forms the basis for the design of the DLL in this thesis.

In the next chapter detailed design of the proposed Digital DLL is given.
Chapter 3

Design of Proposed Fast Lock Digital DLL

The proposed DLL is an improved design of a Counter-Controlled Digital DLL as mentioned before. This chapter covers the block diagram of the proposed DLL along with the detailed explanation of each module in the block diagram. The logic design is also described thoroughly. The design goal is to reduce the lock time of the DLL without changing the power consumption of the circuit after lock and also to reduce the jitter and the residual, uncorrectible offset DLL with the DLL operating frequency of 280MHz - 1.63GHz.

Figure 3.1: Counter-controlled Digital DLL.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

The process used for the design and simulation of the proposed DLL is 45nm CMOS technology. The tools used for the design and simulation are Cadence and HSPICE respectively. The Figure 3.1 and Figure 3.2 shows the block diagram of a complete Digital DLL without the implementation of the Fast lock mechanism and with the implementation of Fast lock mechanism respectively.

Figure 3.2: Proposed Digital DLL.

3.1 Design Flow of the DLL controller

The complete design flow is given in this section. Figure 3.3 shows a flowchart of the design flow of the proposed DLL. Once the start signal is asserted, the counters and the flip flops in the DLL are reset. If the Up_Dn output signal of the PD is high(1), then the 5-bit counter increments. If the Up_Dn signal is low(0), then the 5-bit counter decrements. Depending on the amount of clock skew, the fast lock block in the fast lock mechanism provides the 8 bit control word output to the encoder. The encoder converts the 8 bit control word to give 3-bit control output to a set of flip flops. The 5-bit counter output and the clocked output from the encoder flip-flop block are fed to the adder input. The adder adds these two inputs to give 5-bit sum output. The buffered adder output is fed to a set of flops which helps to fix the adder output after lock and also to select line and lock detect block.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

The output of the lock detector, Lock acts as the select line of the select block. If Lock is 1, the select line selects the output of the flip flop block which will be fixed throughout the DLL operation after lock. If Lock is 0, the select block selects the buffered adder output to feed the input of the 5-to-32 bit decoder. The decoder generates the controller output to control the
coarse delay line. Based on the controller output corresponding delay is added to the input
clock. Also, when Lock is 1, the 4 bit counter controlling the fine delay line is enabled and
when the Lock is 0, the 4 bit counter is disabled and the counter value is b’0000. This complete
process is repeated continuously in a loop as shown in the Figure 3.3 in order to align the
feedback clock (Fbk_clk) to the reference clock (Ref_clk).

The proposed digital DLL also consists mainly of 3 parts namely,
• Phase detector (PD)
• Controller
• Digitally controlled delay line (DCDL)

The design of each block is discussed in detail in the following sections.

3.2 Phase Detector

The use of a PD and its different designs were discussed in chapter 2. The PD used in the
proposed DLL design is a dynamic PD. The block diagram showing the inputs and outputs of
the PD is shown in the Figure 3.4. Ref_clk and Fbk_clk are the inputs to the PD and Up_dn is
the output signal. The PD consists of three SR latches and few basic gates connected as shown
in the Figure 3.5. The NOR gate is used to produce the reset signal for the PD. The first two
latches produce the PD output based on the clock skew between the Ref_clk and the Fbk_CLK.
The third latch is used to latch the PD output to the negative edge of the Ref_clk. The dummy
is used for load balancing between the upper and lower part of the PD.

![Figure 3.4: Block diagram of the Phase detector.](image-url)
The input and output waveforms of the PD are shown in the Figure 3.6. A, B and C in Figure 3.6 shows the output of the PD for three different cases of the Fbk_clk. A corresponds to the output of PD when the Ref_clk lags Fbk_clk, B corresponds to the output when Ref_clk leads Fbk_clk, and C refers to the output when the Ref_clk and Fbk_clk are exactly aligned.
The PD output is high when the Ref$_{clk}$ lags Fbk$_{clk}$. When the Ref$_{clk}$ leads Fbk$_{clk}$, the PD output goes low at the falling edge of the clock which appears first (here it is the Ref$_{clk}$). When the inputs are exactly aligned, the output of the PD goes high at the falling edge of the clock appearing first. The PD is very sensitive and it can detect even 1ps of lead or lag condition of the input clocks.

### 3.3 Controller

The controller block consists of different components namely,

- Binary up-down counters
- Fast lock mechanism
- Adder
- Lock detector
- Adder flip-flop block
- Select
- Decoder

Each of these components are discussed in detail in the following sections.

#### 3.3.1 Binary Up-Down Counter

A binary Up-down counter is a counter which counts up when the input of the counter is high and counts down when the input is low [32]. Two counters are used in the proposed DLL. They are,

- 5-bit up-down counter
- 4-bit up-down counter

#### 3.3.1.1 5-Bit Up-Down Counter

A basic block diagram of the 5-bit up down counter is shown in the Figure 3.7. Up$_{dn}$ signal from the PD is the data input(In) to the counter. The counter is clocked by the Ref$_{clk}$%2 (f2)
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

Figure 3.7: Block diagram of 5-bit counter.

Table 3.1: State table for the 5-bit up-down counter using DFF.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State / D inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DA4, DA3, DA2, DA1, DA0</td>
</tr>
<tr>
<td>A4 A3 A2 A1 A0</td>
<td>U</td>
</tr>
<tr>
<td>00000</td>
<td>0001</td>
</tr>
<tr>
<td>00001</td>
<td>0001</td>
</tr>
<tr>
<td>00010</td>
<td>0011</td>
</tr>
<tr>
<td>00011</td>
<td>0010</td>
</tr>
<tr>
<td>00100</td>
<td>0101</td>
</tr>
<tr>
<td>00101</td>
<td>0110</td>
</tr>
<tr>
<td>00110</td>
<td>0111</td>
</tr>
<tr>
<td>00111</td>
<td>0100</td>
</tr>
<tr>
<td>01000</td>
<td>0101</td>
</tr>
<tr>
<td>01001</td>
<td>0110</td>
</tr>
<tr>
<td>01010</td>
<td>0111</td>
</tr>
<tr>
<td>01011</td>
<td>0100</td>
</tr>
<tr>
<td>01100</td>
<td>0110</td>
</tr>
<tr>
<td>01101</td>
<td>0111</td>
</tr>
<tr>
<td>01110</td>
<td>1000</td>
</tr>
<tr>
<td>01111</td>
<td>1001</td>
</tr>
<tr>
<td>10000</td>
<td>1000</td>
</tr>
<tr>
<td>10001</td>
<td>1001</td>
</tr>
<tr>
<td>10010</td>
<td>1011</td>
</tr>
<tr>
<td>10011</td>
<td>1010</td>
</tr>
<tr>
<td>10100</td>
<td>1011</td>
</tr>
<tr>
<td>10101</td>
<td>1100</td>
</tr>
<tr>
<td>10110</td>
<td>1101</td>
</tr>
<tr>
<td>10111</td>
<td>1110</td>
</tr>
<tr>
<td>11000</td>
<td>1111</td>
</tr>
<tr>
<td>11001</td>
<td>1110</td>
</tr>
<tr>
<td>11010</td>
<td>1111</td>
</tr>
<tr>
<td>11011</td>
<td>1100</td>
</tr>
<tr>
<td>11100</td>
<td>1101</td>
</tr>
<tr>
<td>11101</td>
<td>1110</td>
</tr>
<tr>
<td>11110</td>
<td>1111</td>
</tr>
<tr>
<td>11111</td>
<td>0000</td>
</tr>
</tbody>
</table>
Figure 3.8: K-maps for the flip-flop inputs, a) $DA_0$, b) $DA_1$, c) $DA_2$, d) $DA_3$, and e) $DA_4$.

\[ DA_0 = \overline{A_0} = A_0 \oplus (U + \overline{U}) \]
\[ DA_1 = \overline{A_1} \cdot A_0 + A_1 \cdot \overline{A_0} \\
= A_1 \oplus (U \cdot A_0 + \overline{U} \cdot \overline{A_0}) \]
\[ DA_2 = A_2 \cdot \overline{A_1} + A_2 \cdot \overline{A_0} + \overline{A_1} \cdot A_2 \cdot A_0 \\
= A_2 \oplus (U \cdot A_1 \cdot A_0 + \overline{U} \cdot \overline{A_1} \cdot \overline{A_0}) \]
\[ DA_3 = A_3 \cdot \overline{A_1} + A_3 \cdot \overline{A_2} + A_3 \cdot \overline{A_0} + \overline{A_3} \cdot A_2 \cdot A_1 \cdot A_0 \\
= A_3 \oplus (U \cdot A_0 \cdot A_1 \cdot A_2 + \overline{U} \cdot \overline{A_0} \cdot \overline{A_1} \cdot \overline{A_2}) \]
\[ DA_4 = A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot \overline{A_4} + (\overline{A_1} + \overline{A_3} + \overline{A_2} + \overline{A_0}) \cdot A_4 \\
= A_4 \oplus (U \cdot A_0 \cdot A_1 \cdot A_2 \cdot A_3 + \overline{U} \cdot \overline{A_0} \cdot \overline{A_1} \cdot \overline{A_2} \cdot \overline{A_3}) \]
clock(Clk). The counter is enabled when \( \text{enb} = 0 \). The counter acts as an Up counter when \( \text{Up}\_\text{dn} = 1 \) and it acts as a Down counter when \( \text{Up}\_\text{dn} = 0 \). The outputs of the counter are denoted as \( A[4:0] \). \( A4 \) is the MSB and \( A0 \) is the LSB in the output of the counter.

In order to design such a 5-bit counter, a state change table is created as shown in Table 3.1. Here U represents UP and D represents Down. In our design if \( \text{Up} = 1 \) then \( D = 0 \) and if \( \text{Up} = 0 \) then \( D = 1 \). Therefore \( D \) can be represented as \( \overline{U} \).

K-maps are usually used to derive the flip-flop input equations. By using the Table 3.1, K-map is drawn as shown in the Figure 3.8. The flip-flop equations in 3.1 are derived for \( DA_0 \), \( DA_1 \), \( DA_2 \), \( DA_3 \) and \( DA_4 \) from the K-map. The schematic for the 5-bit counter is based on the above mentioned equations obtained from the K-map. The equations in 3.1 shows that the counter design requires up to 5 input AND gates. As the basic gates with more than 3 inputs are less efficient, the counter is designed with a maximum of 3 input AND gates and the equation requiring more than 3 inputs gates have be simplified in the design. The Figure 3.9 shows the schematic of the 5-bit-up-down counter. The Figure 3.10 shows the output of the counter.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

Figure 3.9: Schematic of 5-bit up-down counter of the DLL.
3.3.1.2 4-Bit Up-Down Counter

The basic block diagram of the 4-bit up-down counter is shown in the Figure 3.11. As in the 5-bit counter, Up\_dn signal from the PD is the data input(In) to the counter. The counter is clocked by the f2 clock (Clk) and it is enabled when en = 1. Even here, the counter acts as an Up counter when Up\_dn = 1 and it acts as a Down counter when Up\_dn = 0. The outputs of the counter are represented as A[3:0]. A3 is the MSB and A0 is the LSB in the output of the counter. In order to design such a 4-bit counter, a state change table is created as shown in Table 3.2. In the table, U represents UP and D represents Down. In our design if Up = 1 then \(D = 0\) and if Up=0 then D=1. Therefore D can be represented as \(\overline{U}\).
K-maps are used to derive the flip-flop input equations as before. By using the Table 3.1, K-map is drawn as shown in the Figure 3.12. From the K-maps, the equations 3.2 are derived for $DA_0$, $DA_1$, $DA_2$ and $DA_3$. The schematic for the 4-bit counter is based on the above mentioned equations obtained from the K-map. The equations in 3.2 shows that the counter design requires up to 4 input AND gates. As mentioned in the 5-bit counter design, the counter is designed with a maximum of 3 input AND gates and the equation requiring more than 3 inputs gates have be simplified in the design. The Figure 3.13 shows the schematic of the 4-bit up-down counter. The Figure 3.14 shows the output of the counter.
Figure 3.12: K-maps for the flip-flop inputs $DA_0$, $DA_1$, $DA_2$ and $DA_3$

\[ DA_0 = \overline{A_0} = A_0 \oplus (U + \overline{U}) \]
\[ DA_1 = \overline{A_1} \cdot A_0 + A_1 \cdot \overline{A_0} \]
\[ = A_1 \oplus (U \cdot A_0 + \overline{U} \cdot \overline{A_0}) \]
\[ DA_2 = A_2 \cdot \overline{A_1} + A_2 \cdot \overline{A_0} + \overline{A_1} \cdot A_2 \cdot A_0 \]
\[ = A_2 \oplus (U \cdot A_1 \cdot A_0 + \overline{U} \cdot \overline{A_1} \cdot A_0) \]
\[ DA_3 = A_3 \cdot \overline{A_1} + A_3 \cdot \overline{A_2} + A_3 \cdot \overline{A_0} + \overline{A_3} \cdot A_2 \cdot A_1 \cdot A_0 \]
\[ = A_3 \oplus (U \cdot A_0 \cdot A_1 \cdot A_2 + \overline{U} \cdot \overline{A_0} \oplus A_1 \cdot \overline{A_2}) \]
Figure 3.13: Schematic of 4-bit up-down counter of the DLL.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

3.3.2 Fast Lock Mechanism

The addition of the fast lock mechanism to the conventional digital DLL design is the highlight of this thesis. Figure 3.15 show its block diagram and it consists of three components namely,

- Fast lock block (FLB)
- Encoder
- Encoder flip-flops (EFF)

Figure 3.14: Output of 4-bit up-down counter.

This 4-bit counter is used to control the Fine delay line, which will be discussed in Delay Line section.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

3.3.2.1 Fast Lock Block

The fast lock block is the main unit of the FLM. It consists of phase width detector (PWD), 8 MUXs denoted as fast lock block MUX (FLBM), DFF, buffers, transmission gate based 2-bit MUXs and basic OR gate cascaded as shown in the Figure 3.16. The input to the FLB are the Ref_clk, Fbk_clk, control inputs t[2:0], enb and reset. The outputs of FLBMs, first set of DFFs and second set of DFFs (output of FLB) are denoted by CW[1:8], D[1:8] and C[1:8] respectively. The FLB is enabled using enb as the select line of transmission gate MUXs which have the inputs as Ref_clk and Fbk_clk. When enb = 1, the FLB is disabled as the Ref_clk and Fbk_clk are not allowed to pass through the transmission gate MUXs and instead low signals are passed through the circuit. When enb = 0 the FLB is enabled and the PWD detects the amount of phase difference between the two clock signals. The Figure 3.17 shows the schematic of the PWD. The buffered output of PWD acts as the select line of FLBMs. Each FLBM consists of a MUX and few basic gates and NMOS transistors connected as shown in the Figure 3.18. The 3 transmission gates connected to 3 NMOS transistor help in providing different delay based on the control inputs, t[2:0]. So, t[2:0] can be adjusted accordingly for different input clock frequencies.
The output of one FLBM is fed to the input of the next FLBM and also to the input of the DFF. The width of the pulse at the output of FLBM gradually decreases from first FLBM to last FLBM and depending on the phase difference between the input clocks, the pulse gradually disappears. The amount of decrease in the width of the pulse at successive FLBM is equal
to the delay of FLBM. The first DFF is used to detect the pulse at the output of FLBMs. As the pulse widths could be very small, a DFF which has very less set-up and hold-time is required. Hence the DFF design discussed in [7] is used. The schematic of the DFF is as shown in the Figure 3.19. The set-up time and hold-time of the DFF is measured as 1ps and -14ps respectively. The clock to Q delay of the DFF is measured as 75ps.

The first DFF is clocked by the buffered output of the FLBM and is reset using delayed Ref clk so as to ensure successful transmission of the data input to the output of the DFF. The second DFF is used to latch the output of first DFF to the rising edge of the Ref clk. For
example, when the phase difference between the Ref clk and the Fbk clk is 200ps, the outputs at the FLBM, first set of DFFs and second set of DFFs (output of FLB) are as shown in Figure 3.20, Figure 3.21 and Figure 3.22 respectively. The 4 output lines of the FLB, C[1:4] are high for a phase difference of 200ps. The Figure 3.23 shows different phase differences and their corresponding active FLBM.

![Figure 3.20: Output of FLBM for a phase difference of 200ps.](image-url)
Figure 3.21: Output of first set of DFFs for a phase difference of 200ps.
Figure 3.22: Output of second set of DFFs for a phase difference of 200ps.
3.3.2.2 Encoder

An encoder is a digital circuit which has $2^n$ input lines and $n$ output lines. An encoder that generates a code based on the highest priority input is called a priority encoder [32].

In this DLL, a 8-bit priority encoder is used as shown in the Figure 3.24. This circuit basically converts a one-hot encoding into a binary representation. If input $n$ is active, all lower inputs ($n-1$ to $0$) are ignored. Table 3.3 shows the truth table of the 8-bit priority encoder. The encoder is designed based on the Boolean equations (3.3) obtained from the truth table in Table 3.3. The $x$ term in the truth denotes don’t care terms which are ignored. In this DLL, the value of $x$ is always 1.

$$Y_2 = E_7 + E_6 + E_5 + E_4$$
$$Y_1 = E_7 + E_6 + E_5 \cdot E_4 \cdot E_3 + E_5 \cdot E_4 \cdot E_2$$
$$Y_0 = E_7 + E_6 \cdot E_5 + E_6 \cdot E_4 \cdot E_3 + E_6 \cdot E_4 \cdot \overline{E_2} \cdot E_1$$

(3.3)
Table 3.3: Truth table for 8-bit priority encoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E7 E6 E5 E4 E3 E2 E1 E0</td>
<td>Y2 Y1 Y0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 x</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 1 x x x</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 1 x x x</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 0 1 x x x</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 1 x x x x</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 x x x x x</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Figure 3.24: 8-bit Priority Encoder.

The 8-bit priority encoder is used to convert the 8 bit input from the FLB to a 3-bit output.
3.3.2.3 Encoder Flip-Flop

The encoder flip-flop (EFF) block has a set of flip flops to clock the 3-bit inputs to the rising edge of the Ref clk. Figure 3.15 shows the block diagram of EFF as well. The input to the EFF block are the outputs from the 8-bit encoder, Ref clk, Reset and en. The en signal is used to fix the value of the encoder output after one Ref_clk cycle. Thus, the EFF block output doesn’t change in the successive clock cycles.

3.3.3 Adder

An adder is a digital circuit that performs addition of numbers. The most common adders operate on binary numbers. A 5-bit Ripple carry adder is used in the DLL to add the 5-bit counter outputs and the flopped 8-bit encoder outputs. Figure 3.25 shows the block diagram of the adder.

![Block diagram of Adder](image)

Figure 3.25: Block diagram of Adder.

Ripple carry adder is designed using multiple full adders to add N-bit numbers. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of adder is called a ripple carry adder, since each carry bit "ripples" to the next full adder [32]. A full adder (FA) is a logical circuit that accepts two operand bits, say A and B, and an incoming carry bit, denoted by Cin to give output Sum and an outgoing carry bit denoted by Cout. The FA is a combinational digital circuit implementing the binary addition of three bits through the Boolean equations in 3.4. Ripple carry adder is preferred in our design due to its less complexity and area efficiency.
\[
\begin{align*}
Sum &= A \oplus B \oplus Cin \\
Cout &= A \cdot B + (A + B) \cdot Cin
\end{align*}
\tag{3.4}
\]

However, the delay time for worst case is more when compared to other adders. Figure 3.26 shows a 5-bit ripple carry adder and a full adder. Here, inputs \(A[4:0]\) are the 5-bit counter outputs and \(B[2:0]\) are the flopped encoder outputs. \(B3, B4, Cin\) and \(Cout\) are connected to ground. \(S[4:0]\) represents the adder sum outputs \(SUM[4:0]\).

3.3.4 Lock Detector

The lock detector (LD) plays a very important role in the functioning of the DLL. LD is used to check the dithering condition of the adder output as shown in Figure 3.27. The schematic representation of the LD is shown in Figure 3.28. It consists of a pair of DFFs connected in series for each bit of the adder output. The inputs of the LD are the buffered outputs of the
adde
denoted by $S[4:0]$, $f2$ and reset signal and the output is the lock signal. Each bit of
the buffered adder output is compared with the corresponding second DFF output to check if
they are same by using the basic XNOR gate. When all the output bits of the adder are same
for 2 consecutive cycles, then the output of all the XNOR gates will be high and at this point
the lock signal goes high and the DLL is considered to be locked. Once the DLL is locked, the lock signal is tied high for all the successive cycles of the DLL using the DFF at the output of the AND gate as shown in the Figure 3.28. The lock signal is used to control different blocks of the DLL namely, adder flip-flop block, select block, 4-bit counter and also the fast lock block. The role of the lock signal in these blocks is discussed in the corresponding sections.

### 3.3.5 Adder Flip-Flop block

The set of flip-flops used to flop the buffered adder outputs is the Adder flip-flop (AFF) block. Figure 3.29 shows the block diagram of AFF. The inputs to the AFF are S[4:0] from the adder, lock, reset and Ref clk (clk). The output of AFF is denoted by SF[4:0]. The lock signal from the LD is used to fix the output of adder to a particular value. When lock = 0, the output of AFF is same as the output of adder and when lock = 1, the output of AFF will be fixed to the previous adder output and does not change in the successive clock cycles.

![Figure 3.29: Block diagram of Adder Flip-Flop block.](image)

### 3.3.6 Select

The select block consists of a set of transmission gate based MUXs to choose between the buffered adder output and the flopped adder output. Figure 3.30 shows the block diagram of the select block. The inputs S[4:0] and SF[4:0] denote the adder output and the AFF output respectively and SM[4:0] denotes the select block output. The output of LD acts as the select line of the MUXs in the select block. When lock = 1, the AFF output is selected and when
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

lock = 0, the buffered adder output is selected by the select block.

![Block diagram of Select](image)

Figure 3.30: Block diagram of Select.

### 3.3.7 Decoder

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of $2^n$ unique output lines [33]. A 5-to-32 bit decoder is used in the DLL. This decoder is used to convert the select block output to the control inputs of the coarse delay line. In this thesis, the 5-to-32 decoder is obtained by using one 2-to-4 decoder and four 3-to-8 decoder as shown in the Figure 3.31.

A 2-to-4 decoder has 2 input lines and $2^2 = 4$ output lines as shown in Figure 3.32. The 2-to-4 decoder is implemented through Boolean equations in 3.5. This decoder controls the enable input of the four 3-to-8 decoders in the 5-to-32 decoder as in Figure 3.31.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

Figure 3.31: 5-to-32 Decoder.

Figure 3.32: 2-to-4 Decoder.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

\[ D_0 = \overline{I_1} \cdot \overline{I_0} \cdot En \]
\[ D_1 = \overline{I_1} \cdot I_0 \cdot En \]
\[ D_2 = I_1 \cdot \overline{I_0} \cdot En \]
\[ D_3 = I_1 \cdot I_0 \cdot En \]  

(3.5)

Similarly, a 3-to-8 decoder has 3 input lines and \(2^3 = 8\) output lines as shown in Figure 3.33 and it is implemented through the Boolean equations in 3.6.

\[ D_0 = \overline{I_2} \cdot \overline{I_1} \cdot \overline{I_0} \cdot En \]
\[ D_1 = \overline{I_2} \cdot I_1 \cdot \overline{I_0} \cdot En \]
\[ D_2 = \overline{I_2} \cdot I_1 \cdot I_0 \cdot En \]
\[ D_3 = I_2 \cdot \overline{I_1} \cdot \overline{I_0} \cdot En \]
\[ D_4 = I_2 \cdot I_1 \cdot \overline{I_0} \cdot En \]
\[ D_5 = I_2 \cdot I_1 \cdot I_0 \cdot En \]
\[ D_6 = I_2 \cdot I_1 \cdot \overline{I_0} \cdot En \]
\[ D_7 = I_2 \cdot I_1 \cdot I_0 \cdot En \]  

(3.6)
3.4 Digital Controlled Delay Line

The delay line consists of two components namely,

- Coarse delay line
- Fine delay line

3.4.1 Coarse Delay Line

A MUX based coarse delay line (CDL) is used in this DLL. The CDL consists of 32 coarse delay units (CDU) as shown in the Figure 3.34. The design of CDU is same as that of the FLBM as shown in Figure 3.35. Each CDU has a 2-to-1 MUX and few basic gates as shown in the Figure 3.35. The output from the MUX in the previous stage is fed to the input 1 of the MUX and the clock is fed to the input 2 of all the MUXs. The 32 bit control signal from the 5-to-32 bit decoder serves as the select lines of the MUXs in the CDL. The 3 transmission gates connected to 3 NMOS help in providing different delay based on the control inputs, t[2:0]. Each CDU provide a delay between 32ps - 46ps depending on the control inputs. This helps to have DLL operate for a wide range of frequencies between 280MHz to 1.62GHz. Table 3.4 gives the different control inputs and the corresponding delay of each CDU. The output of the CDL is the delayed clock and the amount of delay is based on the controller outputs.

Figure 3.34: Coarse Delay Line
3.4.2 Fine Delay Line

The fine delay line (FDL) mainly consists of two components namely,

- Fine delay block
- 16-bit Multiplexer

The output of the CDL is the input to the FDL. Figure 3.36 shows the block diagram of the FDL.
3.4.2.1 Fine Delay Block

The design of the fine delay block (FDB) is mainly based on the phase blending technique discussed in [6]. The phase blending was explained in chapter 2 as well. FDB consists of 3 fine delay units (FDU), multiple phase blenders and dummy blocks as shown in the Figure 3.37. The design of FDU is same as that of the FLBM shown in Figure 3.38. FDUs are used to generate clock signals of three different phases from a single clock input. The FDUs are cascaded in such a way that the output of first FDU is fed to a input of the second FDU and the output of second FDU is in turn fed to the input of the third FDU. The output of the FDUs, clk1, clk2 and clk3 are blended in three stages so as to get 16 signals, c1 to c16 which are of different phases. Figure 3.39 shows the circuit of a phase blender unit. The minimum phase difference between any two consecutive outputs of the FDB is 4ps. So, the minimum time resolution or the residual, uncorrectible offset of the DLL is 4ps. The outputs of the three FDUs and the 16 phase blended outputs of the FDB are shown in Figure 3.40. As explained in the CDL section, the control bits t[2:0] helps in adjusting the delay of the FDU according to the frequency of operation of the DLL. The advantage of having FDU based FDL is, as the delay of the FDU changes, the phase difference between the consecutive outputs of the FLB also changes proportionately. This helps in having the FDL work in accordance with the CDL.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

Figure 3.37: Fine Delay Block.

Figure 3.38: Fine Delay Unit.
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

Figure 3.39: Phase Blender Unit.

Figure 3.40: Waveforms of a) Input, Output of FDUs and b) Output of FDB.
### 3.4.2.2 16-bit Multiplexer

A n-bit multiplexer takes n input signals, and shifts one of them to the single output under the control of \( \log_2 n \) bit select signal [33]. A 16-bit MUX has 16 input signals with 4 select lines and a single output as shown in Figure 3.41. This 16-bit MUX is designed using two 8-bit MUXs and one 2-bit MUX. The output of the 4-bit up-down counter discussed in the controller section acts as the 4-bit select line of the 16-bit MUX.

![Figure 3.41: 16-bit Multiplexer.](image)

Each 8-bit MUX consists of multiple 2-bit MUXs connected as shown in the Figure 3.42. The 2-bit mux used in this design are simple transmission gate MUXs with very less delay. The Figure 3.42 also shows the path from input to output for the select line input, \( S[2:0] = \text{b’100} \).
CHAPTER 3. DESIGN OF PROPOSED FAST LOCK DIGITAL DLL

3.5 Conclusion

In this chapter, detailed information about the design of each block of the proposed Digital DLL and their corresponding simulation results are given. The advantage and the purpose of using a particular design for each block is also explained.

In the next chapter, the results of the complete DLL are discussed and also the results of the proposed DLL are compared with the results of other analog and digital DLLs.
Chapter 4

Results and Performance Analysis

The total number of NMOS and PMOS transistors used in the proposed DLL design are 2011 and 2403 respectively. The number of lock cycles, jitter after lock, power dissipation, and phase error are the most important parameters considered for the performance analysis of the DLL. The Monte-carlo simulation is carried out for a Monte Value of 30 with the following variations in order to obtain the jitter values.

- Voltage variation from 0.9V to 1.1V with a step of 0.02
- Temperature variation from 25F to 125F with a step of 25F
- Gaussian distribution for channel length with Mean= 50nm, Absolute variation = ±2 and \( \sigma = 3 \)

The detailed performance analysis for two cases of phase differences between the Ref_clk and Fbk_clk is given.

- Case1: Phase Difference = 200ps

In order to align the Fbk_clk to Ref_clk with different phase differences, the number of lock cycles utilized by the DLL with and without fast locking mechanism is plotted in the Figure 4.2 and Figure 4.1 respectively. The Figure 4.2 and Figure 4.1 shows that less number of cycles are required to align the Fbk_clk to ref_clk when compared to that of the DLL without FLM. The RMS jitter and peak to peak jitter values are measured as 1.084ps and 2.169ps respectively as shown in Figure 4.3

53
Figure 4.1: PD vs. Coarse delay, No. of delay elements for 250ps PD without FLM.

Figure 4.2: PD vs. Coarse delay, No. of delay elements for 250ps PD with FLM.
CASE 2: PHASE DIFFERENCE = 500ps

In order to align the Fbk_clk to Ref_clk with different phase differences, the number of lock cycles utilized by the DLL with and without fast locking mechanism is plotted in the Figure 4.5 and Figure 4.4 respectively. The Figure 4.2 and Figure 4.1 also shows that less number of cycles are required to align the fbk_clk to ref_clk when compared to that of the DLL without FLM. The RMS jitter and peak to peak jitter values are measured as 5.26ps and 10.57ps respectively as shown in Figure 4.6.
Figure 4.4: PD vs. Coarse delay, No. of delay elements for 500ps PD without FLM.

Figure 4.5: PD vs. Coarse delay, No. of delay elements for 500ps PD with FLM.
The Figure 4.7 shows the graph of PD vs. lock and it is plotted using the values in Table 4.1. The graph shows that the lock time is less for the DLL with fast lock mechanism when compared to that of a conventional DLL. Figure 4.8 shows the graph of PD vs. power consumption before and after DLL locks. The graph shows that the power consumption of the DLL after lock is almost same as that of the power consumption of the conventional DLL as the fast lock mechanism is turned off after DLL locks.
Figure 4.7: PD vs. Lock cycles for proposed DLL.

Table 4.1: PD and corresponding lock cycle values of the proposed DLL.

<table>
<thead>
<tr>
<th>PD [ps]</th>
<th>With FLB</th>
<th>Without FLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>150</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>200</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>250</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>300</td>
<td>14</td>
<td>26</td>
</tr>
<tr>
<td>350</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>400</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>450</td>
<td>22</td>
<td>36</td>
</tr>
<tr>
<td>500</td>
<td>24</td>
<td>38</td>
</tr>
<tr>
<td>550</td>
<td>26</td>
<td>44</td>
</tr>
<tr>
<td>600</td>
<td>28</td>
<td>46</td>
</tr>
<tr>
<td>650</td>
<td>30</td>
<td>48</td>
</tr>
<tr>
<td>700</td>
<td>32</td>
<td>50</td>
</tr>
<tr>
<td>750</td>
<td>34</td>
<td>52</td>
</tr>
<tr>
<td>800</td>
<td>36</td>
<td>54</td>
</tr>
<tr>
<td>850</td>
<td>38</td>
<td>56</td>
</tr>
<tr>
<td>900</td>
<td>40</td>
<td>58</td>
</tr>
<tr>
<td>950</td>
<td>42</td>
<td>60</td>
</tr>
</tbody>
</table>
Figure 4.8: PD vs. Power consumption before and after lock for proposed DLL.

The Table 4.2 shows the performance comparison of the proposed DLL with other analog DLLs in [10], [34] and [35]. From this we can infer that the jitter values of the proposed DLL are on par with that of the analog DLLs.

Table 4.2: Performance comparison of the proposed DLL with other Analog DLLs.

<table>
<thead>
<tr>
<th></th>
<th>[3]</th>
<th>[34]</th>
<th>[35]</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Analog</td>
<td>Analog</td>
<td>Analog</td>
<td>Digital</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>0.18 µm</td>
<td>0.35 µm</td>
<td>0.35 µm</td>
<td>45 nm</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>2 GHz</td>
<td>1.1 GHz</td>
<td>1.8 GHz</td>
<td>500 MHz</td>
</tr>
<tr>
<td><strong>Supply Voltage</strong></td>
<td>1.8 V</td>
<td>3.3 V</td>
<td>3.3 V</td>
<td>1 V</td>
</tr>
<tr>
<td><strong>RMS Jitter</strong></td>
<td>1.6 ps</td>
<td>2 ps</td>
<td>1.8 ps</td>
<td>5.26 ps</td>
</tr>
<tr>
<td><strong>Peak-to-peak Jitter</strong></td>
<td>13.1 ps</td>
<td>14.6 ps</td>
<td>12.2 ps</td>
<td>10.57 ps</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>12 mW</td>
<td>42.9 mW</td>
<td>86.6 mW</td>
<td>287.24 µW</td>
</tr>
</tbody>
</table>

The Table 4.2 shows the performance comparison of the proposed DLL with other digital DLLs in [36], [37], [38], [39], [40] and [41]. This shows that the performance of the proposed DLL is better than that of the DLLs discussed in [36], [37], [38], [39], [40] and [41].
Table 4.3: Performance comparison of the proposed DLL with other Digital DLLs.

<table>
<thead>
<tr>
<th></th>
<th>[36]</th>
<th>[37]</th>
<th>[38]</th>
<th>[39]</th>
<th>[40]</th>
<th>[41]</th>
<th>This Work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Digital</td>
<td>Digital</td>
<td>Digital</td>
<td>Digital</td>
<td>Digital</td>
<td>Digital</td>
<td>Digital</td>
</tr>
<tr>
<td><strong>Timing Resolution</strong></td>
<td>X</td>
<td>X</td>
<td>&lt; 28 ps</td>
<td>70.9 ps</td>
<td>&lt; 30 ps</td>
<td>X</td>
<td>4 ps</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>0.35 μm</td>
<td>0.13 μm</td>
<td>0.18 μm</td>
<td>0.25 μm</td>
<td>0.35 μm</td>
<td>0.18 μm</td>
<td>45 nm</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>130 MHz</td>
<td>500 MHz</td>
<td>700 MHz</td>
<td>100 MHz</td>
<td>250 MHz</td>
<td>800 MHz</td>
<td>500 MHz</td>
</tr>
<tr>
<td><strong>Supply Voltage</strong></td>
<td>3.3 V</td>
<td>&lt; 1.8 V</td>
<td>1.8 V</td>
<td>1 V</td>
<td>3.3 V</td>
<td>1.8 V</td>
<td>1 V</td>
</tr>
<tr>
<td><strong>Lock time</strong></td>
<td>~1130 cycles</td>
<td>&lt; 150 cycles</td>
<td>32 cycles</td>
<td>8 cycles</td>
<td>10 cycles</td>
<td>&lt;80 cycles</td>
<td>&gt;50 cycles</td>
</tr>
<tr>
<td><strong>RMS Jitter</strong></td>
<td>3.297 ps</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>6.2 ps</td>
<td>5.26 ps</td>
</tr>
<tr>
<td><strong>Peak-to-peak Jitter</strong></td>
<td>24.3 ps</td>
<td>&lt; 25 ps</td>
<td>17.6 ps</td>
<td>30 ps</td>
<td>20.4 ps</td>
<td>20.4 ps</td>
<td>10.57 ps</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>132 mW</td>
<td>24 mW</td>
<td>23 mW</td>
<td>2.43 mW</td>
<td>9.9 mW</td>
<td>12 mW</td>
<td>287.24 μW</td>
</tr>
</tbody>
</table>

X: Not mentioned
Chapter 5

Conclusion and Future Works

DLL plays a vital role in the successful error free data transfer between different clock domains by aligning clocks of different domains to the reference clock. The proposed DLL has improved performance in terms of lock cycles, jitter and less residual, uncorrectable offset by bringing back the power consumption of the DLL to normal values after lock. The FLB of the proposed DLL can detect a phase difference up to 400ps in a single cycle. After one cycle the FLB is turned off. So, the delay line adjusts up to 400ps in a single cycle. For the phase difference above 400ps, the delay line provides one delay unit delay for the successive clock cycles after the first cycle. So, in order to further reduce the lock cycles and jitter, this design can be further improved to store the value of the encoder output in the first cycle and then add the encoder output values obtained in each cycle to the stored value successively.
Bibliography


