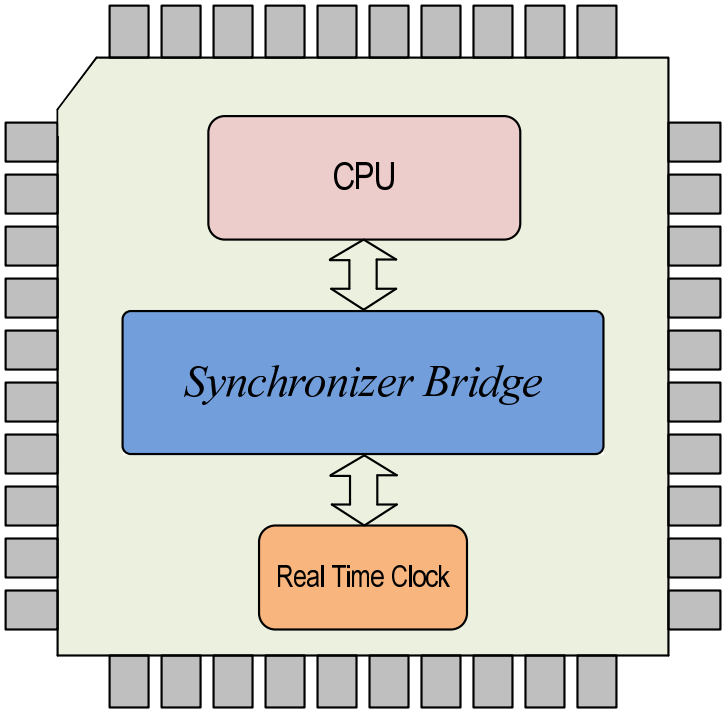
CHALMERS



Evaluation of synchronization methods in multi-clock domain systems

Master of Science thesis in Integrated Electronic System Design

Amit Kulkarni

CHALMERS UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Göteborg, Sweden, June 2012

Evaluation of synchronization methods in multi-clock domain systems

Amit Kulkarni

Examiner: Lars Svensson

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden

Abstract

Modern SoC employ multi clock domains on the same die, this is because each block of the system may require different clock voltage and frequencies which results to economical benefits. Systems with embedded processors generally requires high speed clocks, it becomes very difficult to maintain one global clock connected all sub blocks to meet the speed requirement of the system. One way to overcome this difficulty is to employ GALS clock system. In GALS clock system each block/island is locally synchronous and connected to other blocks by an asynchronous interconnect system.

Synchronization bridge is required in between multiple clock domains to avoid the risk of metastability to those signals which are prone to frequent transitions, and are propagated between two multiple clock domains. In this project, the bridges are applied to the Wishbone bus in which two clocks differs from each other, in frequency and phase. Each clock is separately applied for Wishbone Master and Wishbone Slave thus acts as two as different clock domain systems w.r.t signals that are propagated between them.

Bridges such as two flip flop and locally delayed latching are designed and implemented. The performance is evaluated and compared. Further the bridges are processed to standard VLSI place and route.

Acknowledgement

First and foremost I would like to thank my examiner and supervisor for their help and guidance throughout the thesis:

Examiner:	**Lars Svensson**
	Head of Division, Computer Engineering, Chalmers University of Technology, Göteborg, Sweden
Supervisor:	**Ronan Barzic**
	Senior Discipline Manager AVR32 Mid-end, Atmel Corporation, Trondheim, Norway

Further I would like to thank everyone at Atmel who has contributed to the ideas and helped me with insightful feedback. My thanks also goes to Per Larsson-Edefors, Sally McKee and the VLSI work group at Chalmers for their interest and feedback.

Special thanks to friends and family for their support.

Preface

This report is the result of master thesis work carried out at Atmel Corporation, Trondheim Norway. The thesis also fulfils the partial requirement for the Master Degree "Integrated Electronic System Design" at Chalmers University of Technology, Gothenburg, Sweden.

I would like to thank Akshay Vijayshekar, Hasan Ali and Kristoffer Koch for helping me during crucial situations of my thesis. Their timely and good advice made the completion of this thesis possible. Finally I would like to show my gratitude to program manager Frode Sundal and all the members at MaxTouch team for providing me a lively and motivating environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The demand for high speed in electronic devices forces the technology to use high speed clock. The requirement for high speed in systems becomes increasingly difficult to maintain a single clock for whole system. One way to reduce this difficulty is to use the GALS concept. In GALS system, the locally synchronous domains are connected to an asynchronous interconnect system [1].

Typically in an integrated circuit, the locally synchronous islands are interconnected with a high speed bus such as Wishbone or AMBA AHB/APB bus [2]. The interconnect computer bus is a synchronous system which is driven by the system clock; the bus ensures the correct data transmission between multiple locally synchronous islands.

1.2 Problem Description

In a multi-clock domain system, each domain has individual clock with different clock parameters such as frequency and voltage. The data transfer between the clock domains is asynchronous and is prone to metastability. Hence it is required to have a synchronizer between clock domains to make sure that data transferred are safe by avoiding metastability. The thesis was started to examine some different synchronizers drafted in literature by implementing them. The most popular synchronizers such as single/two flip-flop, clock stretching and LDL are being used in a typical multi-clock domain integrated circuit. It is required to evaluate the different synchronizers, implement them and perform standard VLSI place and route. Key parameters such as such as data latency, hardware area and power are compared.

We performed this task considering Wishbone bus, and evaluate two synchronizers:

- 4 phase two flip-flop synchronizer
- LDL synchronizer

1.3 Thesis Outline

The second chapter gives an overview of the Wishbone bus and its functions. The third chapter gives the theory of metastability and synchronizer design methodologies. In chapter four, standard four phase two flip-flop synchronizer bridge is introduced with discussion of its working concept. Chapter five begins with the application of four phase two flip-flop synchronizer bridge to Wishbone bus and some optimizations are discussed. Chapter six gives a brief introduction to GALS and discusses the standard LDL synchronizer. Chapter seven is dedicated to application of LDL synchronizer bridge to Wishbone bus and some optimizations are discussed. Chapter eight describes the complete verification of the synchronizer bridges. Chapter nine gives description of the hardware of synchronizers, and layouts of the two synchronizers are being depicted. Finally chapter ten is dedicated for the documentation of overall results and comparison of key parameters.

1.4 Notation

1.4.1 Abbreviations

ACK	Acknowledge
ACK+	Rising edge of Acknowledge
ACK-	Falling edge of Acknowledge
AMBA AHB	Advanced Microcontroller Bus Architecture Advanced High Performance Bus
AMBA APB	Advanced Microcontroller Bus Architecture Advanced Peripheral Bus
a.k.a	Also Known As
async.	Asynchronous
CLK	Clock
D flip-flop	Data flip-flop
DVE	Discovery Visualization Environment
E	Enable
EDA	Electronic Design Automation
etc.	et cetera
FIFO	First In First Out
FO4	Fan Out 4
GALS	Globally Asynchronous Locally Synchronous
GPIO	General Purpose Input Output
Hz	Hertz
HDL	Hardware Description Language
IP	Intellectual Property
kHz	kilo Hertz
L	Latched
LDL	Locally Delayed Latching

MCC	Master Clock Cycle
MHz	Mega Hertz
MTBF	Mean Time Between Failures
MUTEX	Mutually Exclusive
ns	nano second
ps	pico second
R-DATA	Received Data
REGD	Register Data
REGR	Register Receiver
REQ	Request
REQ+	Rising edge of Request
REQ-	Falling edge of Request
RXE	Receive Enable
RZ	Return-to-Zero
S-DATA	Send Data
S-R Latch	Set - Reset Latch
SCC	Slave Clock Cycle
SNT	Sent
SoC	System on Chip
STG	State Transition Graph
SYSCON	System Control
T	Time period
Tx-FSM	Transmitter - Finite State Machine
TXE	Transmitter Enable
VCS	Verilog Compiled code Simulator
VI	Valid Input
VLSI	Very Large Scale Integration
VO	Valid Output
w.r.t	with respect to
WACK	Write Acknowledge
WDATA	Write Data

1.4.2 Abbreviations of Wishbone bus signals

ACK_I	Acknowledge Input
ACK_O	Acknowledge Output
ADR_I( )	Address Input Bus
ADR_O( )	Address Output Bus
CLK_I	Clock Input
CLK_O	Clock Output
CYC_I	Bus cycle Input
CYC_O	Bus cycle Output
DAT_I( )	Data Input Bus

DAT_O( )	Data Output Bus
RST_I	Reset Input
RST_O	Reset Output
SEL_I( )	Select Input Bus
SEL_O( )	Select Output Bus
STB_I	Strobe Input
STB_O	Strobe Output
TAGN_I( )	Tag data Input
TAGN_O( )	Tag data Output
WE_I	Write Enable Input
WE_O	Write Enable Output

1.4.3 Symbols

τ	Metastability resolving time constant
C_L	Combinational logic
D_{CTRL}	Asynchronous Controller Delay
D_{CTRL1}	Asynchronous Controller Delay of LDL bridge connected from Wishbone master to slave
D_{CTRL2}	Asynchronous Controller Delay of LDL bridge connected from Wishbone slave to master
DFF_LDL_{Delay}	Delay of a flip-flop that holds the output of LDL bridge
DFF_LDL_{Delay1}	Delay of a flip-flop that holds the output of LDL bridge connected from Wishbone master to slave
DFF_LDL_{Delay2}	Delay of a flip-flop that holds the output of LDL bridge connected from Wishbone slave to master
DFF_{MS}	Delay in flip-flops present inside the two flip-flop bridge connected from Wishbone master to slave
DFF_{SM}	Delay in flip-flops present inside the two flip-flop bridge connected from Wishbone slave to master
D_L	Sequential Logic
F_c	Clock Frequency
F_d	Data Frequency
m	Number of Wishbone master clock cycles
N	Number of gates
n	Number of Wishbone slave clock cycles
$P(FFfailure)$	Probability of flip-flop failures
P_{fm}	Probability of flip-flop entering metastability
D	Data
T	Time period of clock
$T_{2FFcycle}$	Theoretical Time period for Wishbone single read/write cycle with two flip-flop synchronizer bridge
T_{2FFsim}	Simulation Time period for Wishbone single read/write cycle

	with two flip-flop synchronizer bridge
$T_{2FFsynchronizer}$	Time delay in two flip-flop synchronizer bridge
T_c	Time period of clock
T_δ	Difference in Time delay of synchronizer bridges
$T_{LDLcycle}$	Theoretical Time period for Wishbone single read/write cycle with LDL synchronizer bridge
T_{LDLsim}	Simulation Time period for Wishbone single read/write cycle with LDL synchronizer bridge
$T_{LDLsynchronizer}$	Time delay in LDL synchronizer bridge
$T_{m/s}^{MUTEX}$	Metastability Resolution time of MUTEX element
$T_{m/s}^{MUTEX1}$	Metastability Resolution time of MUTEX element present in the LDL input port connected from Wishbone master to slave
$T_{m/s}^{MUTEX2}$	Metastability Resolution time of MUTEX element present in the LDL input port connected from Wishbone slave to master
T_w	Window time period within which metastability can occur
$T_{Wb_{master\_clock}}$	Time period of Wishbone master clock
$T_{Wb_{slave\_clock}}$	Time period of Wishbone slave clock
Wb_master_{delay}	Time delay occurred in Wishbone master during active cycle
$Wb\_master1_{delay}$	Time taken by the Wishbone master to detect ACK_I and de-assert REQ_O
$Wb\_master2_{delay}$	Time delay in Wishbone master to detect ACK_O being de-asserted by slave
Wb_slave_{delay}	Time delay occurred in Wishbone slave during active cycle
Wb_slave1_{delay}	Time taken by the Wishbone slave to detect STB_I and assert ACK_O
Wb_slave2_{delay}	Time take by Wishbone slave to detect falling edge of STB_I and de-assert ACK_O
$Y1$	Locally delayed signal in LDL input port

Chapter 2

Wishbone Bus

This chapter introduces the basic concepts of Wishbone bus and gives a summary of its architecture and topologies.

Wishbone bus is an open source communication bus that is used to establish the communication between different blocks of an integrated circuit. The communication is established by synchronous handshake signals of a standard Wishbone bus. The communication is generic and can be applied between different kind of blocks of an integrated circuit.

2.1 Wishbone Bus Architecture

Figure 2.2 shows the architecture of the Wishbone bus. It consists of standard "Master" and "Slave" architecture which are connected to each other with a special interface called "SYSCON". In a typical integrated circuit, the Wishbone master and slave can be connected with different topologies which is discussed in section 2.7. Thus they act as a data exchange port between other blocks.

The master initiates the control signals to slave in order to start a bus cycle. The slave decodes the signals and acts according to standard data transfer cycles. Two modes of operation are supported supported :

- Classic Standard mode
- Classic Pipeline mode

Classic Pipeline mode is considered to be out of scope in this project and hence we mainly concentrate on Classic Standard mode of read and write cycles. In the following sections, the details of the Wishbone signals are described.

2.2 Wishbone Signals

In the following, we classify the Wishbone signals into four different categories:

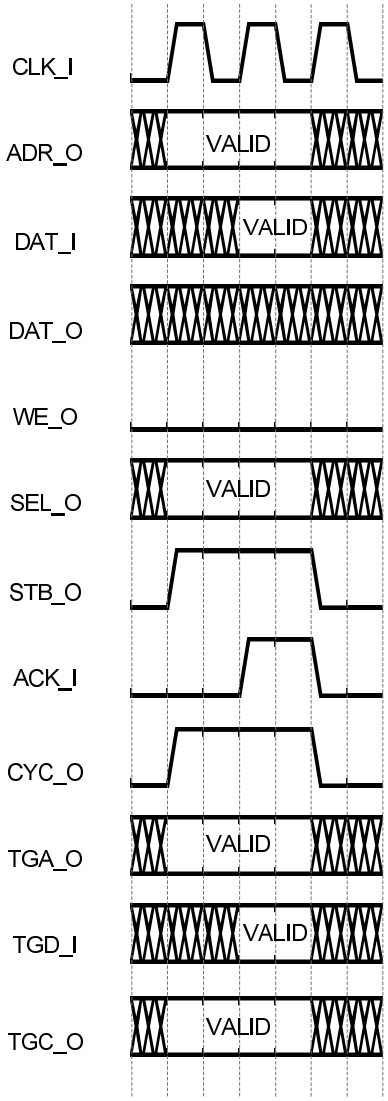


Figure 2.1: Classic single read cycle [3]

- SYSCON module Signals
- Signals common to Wishbone Master and Wishbone Slave
- Wishbone Master Signals
- Wishbone Slave Signals

2.2.1 SYSCON Module Signals

SYSCON mainly generates two signals, RST_O and CLK_O. The RST_O is used to instruct the system reset to master as well as slave. The CLK_O provides the required

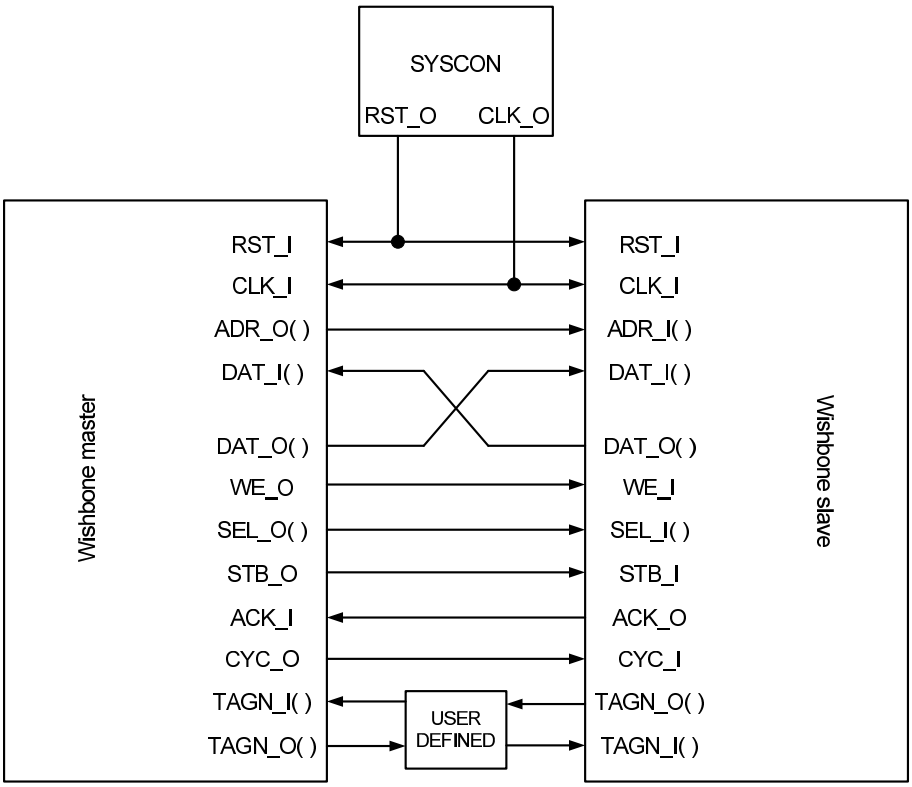


Figure 2.2: Wishbone architecture [3]

clock to both master and slave thus keeps them in synchronous operation.

2.2.2 Signals common to Wishbone Master and Wishbone Slave

The following signals are common to both Master and Slave:

- CLK_I: Clock input used to synchronize all the logical activities inside master and slave. This signal is connected to CLK_O of SYSCON module.
- RST_I: Reset input is used to trigger the system reset. This makes the Wishbone bus to return to known working state. This signal is connected to RST_O of SYSCON module.
- DAT_I( ): Data input bus is used to receive an array of data. The size of data is determined by the port size defined during design of Wishbone bus.
- DAT_O( ): Data output bus is used to transmit an array of data. The size of data is determined by the port size defined during design of Wishbone bus.
- TAGN_I( ): Tag data input bus is used to accept additional information regarding the data, select lines etc.

- TAGN_O( ): Tag data output bus is used to send additional information regarding the data, select lines etc.

2.2.3 Master Signals

The following signals are manipulated by master:

- ACK_I: Acknowledge input signal is used by master for normal termination of the current bus cycle.
- ADR_O( ): Address output is used to send the address in the form of binary data, it is generally used to point the address of the location where data has to be written or read. Its port size is defined during design of Wishbone bus.
- CYC_O: The Cycle output is used to indicate that valid bus cycle is in progress. The signal is asserted as long as the current operation is still in progress.
- SEL_O( ): The Select output bus holds the binary data indicating the place of valid data that is present in DAT_O and DAT_I.
- STB_O( ): The Strobe output signal is used to indicate the valid data transfer cycle, it is used for qualifying the presence of valid data on data and select bus.
- WE_O( ): The Write Enable output signal is used to flag the current bus cycle is of data write.
- TGA_O( ): The Tag Address output bus holds the additional information about the address bus ADR_O.
- TGC_O( ): The Tag Cycle output bus holds the additional information about the bus cycles.

2.2.4 Slave Signals

The following are the signals that are manipulated by slave:

- ACK_O: Acknowledge output signal is sent by slave for normal termination of the current bus cycle.
- ADR_I( ): Address input is used to accept the address in the form of binary data, it is generally used to point the address of the location where data has to be written or read by the master. Its port size is defined during design of Wishbone bus.
- CYC_I: The Cycle input is used to detect that valid bus cycle is in progress. The signal is asserted as long as the current operation is still in progress.
- SEL_I( ): The Select input bus holds the binary data indicating the place of valid data that is present in DAT_O and DAT_I.

- STB_I( ): The Strobe input signal is used to detect the valid data transfer cycle, it is used for qualifying the presence of valid data on data and select bus.
- WE_I( ): The Write Enable input signal is used to accept the current bus cycle is of data write.
- TGA_I( ): The Tag Address input bus holds the additional information about the address bus ADR_O.
- TGC_I( ): The Tag Cycle input bus holds the additional information about the bus cycles.

2.3 Classic Standard Single Read Cycle

In single read cycle, only one chunk of data is fetched by master from slave. The chunk size is predefined by designer and may be 8-bit, 16-bit, 32-bit, or 64-bit [3].

The bus cycle of classic single read is depicted in figure 2.3 and is self explaining. The STB_O and ACK_I plays important part of handshake actions and ensures the the required synchronization with slave. WE_O is always de-asserted throughout the bus cycle to maintain the read activity.

2.4 Classic Standard Single Write Cycle

In single write cycle, only one chunk of data is written by master to slave. The chunk size is predefined by designer and may be 8-bit, 16-bit, 32-bit, or 64-bit [3].

The bus cycle of classic single write is depicted in figure 2.4 and is self explaining. The only difference compared to single read cycle is, WE_O is asserted at required period and hence it ensures the write activity.

2.5 Classic Standard Block Read Cycle

In block read cycle, a complete block of data is read by master from slave. A group of single chunks of data forms one block.

The bus cycle of classic block read is depicted in figure 2.5 and is self explaining. The STB_O and ACK_I plays important part of handshake actions and ensures the the required synchronization with slave. WE_O is always de-asserted to ensure that complete bus cycle is of only read activity. The CYC_O is asserted till complete block of data is fetched.

2.6 Classic Standard Block Write Cycle

In block write cycle, only a complete block of data is written by master to slave. A group of single chunks of data forms one block.

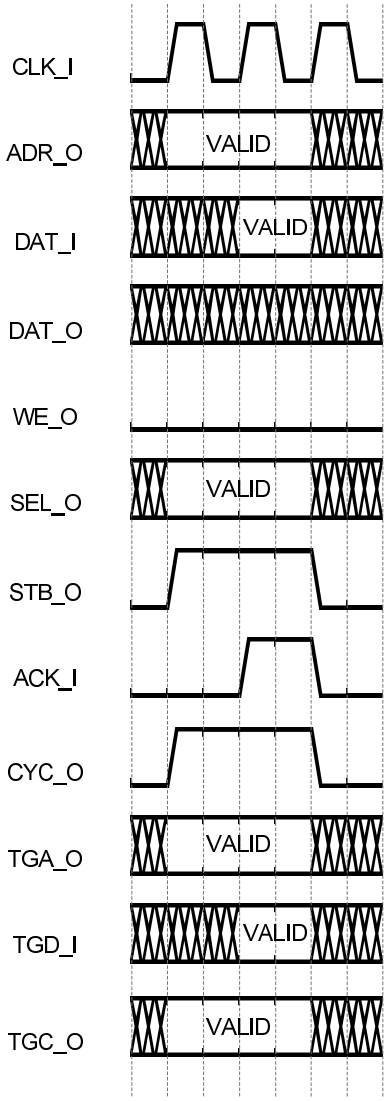


Figure 2.3: Classic single read cycle [3]

The bus cycle of classic block write is depicted in figure 2.6 and is self explaining. The only difference from block read cycle is, WE_O is always asserted at right period to ensure that complete bus cycle is of only write activity.

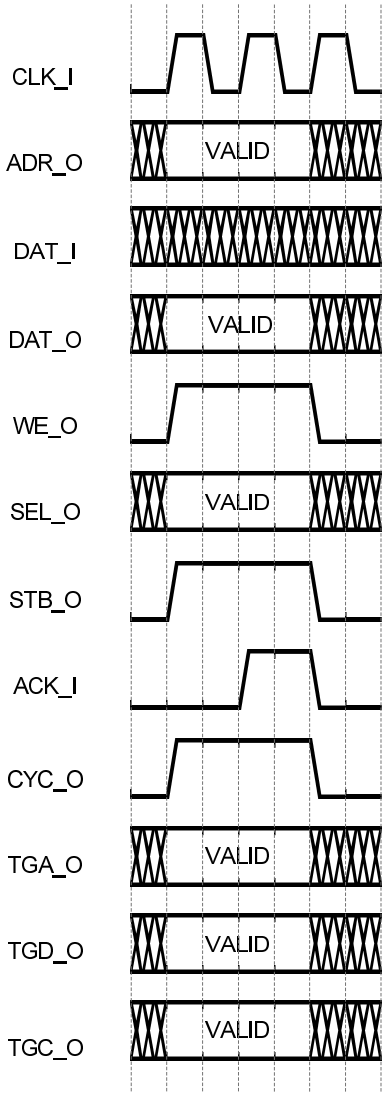


Figure 2.4: Classic single write cycle [3]

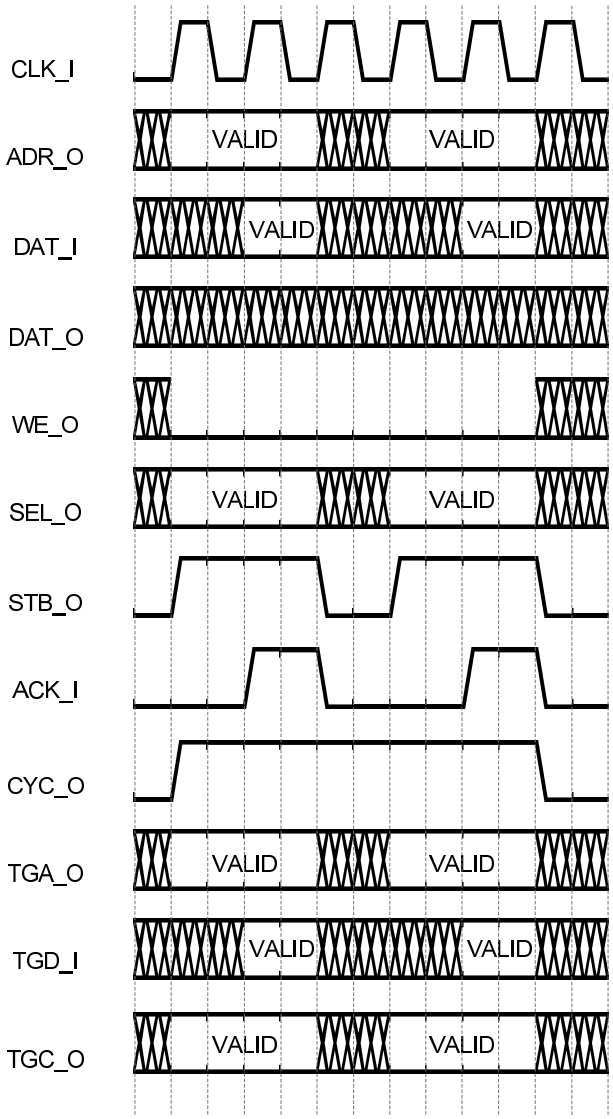


Figure 2.5: Classic block read cycle [3]

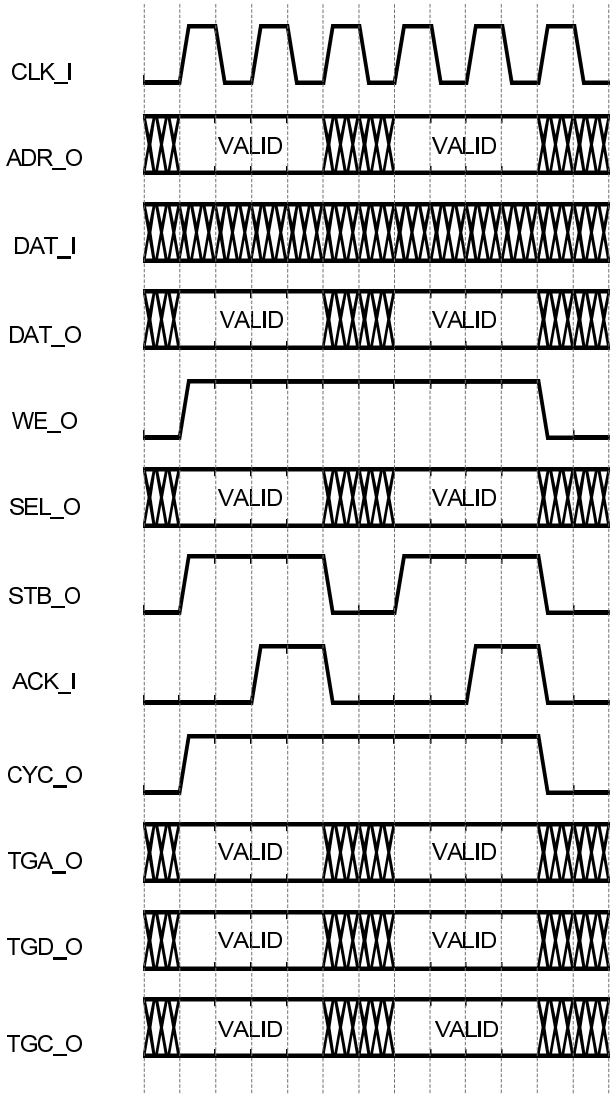


Figure 2.6: Classic block write cycle [3]

2.7 Wishbone Topologies

Four different topologies that Wishbone adapts for communication interconnection are discussed in the following sections:

2.7.1 Point to Point Interconnection

In point-to-point interconnection, single Wishbone master is connected to a single Wishbone slave directly and the data transfer takes place according to the handshake protocol [4]. This kind of interconnection is simplest and the data transfer follows the handshake protocol. The configuration is depicted in figure 2.7 and figure 2.2.

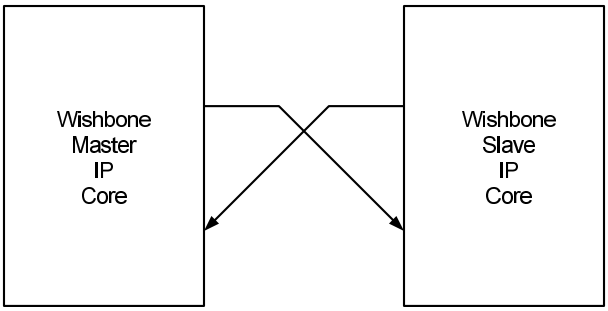


Figure 2.7: Wishbone Point to Point Interconnection [4]

2.7.2 Shared Bus Interconnection

In shared bus interconnection, there exists multiple Wishbone master and Wishbone slave. All of them share a common bus; the bus is accessible to single Wishbone master at a time, the other Wishbone master has to wait until it gets the bus access; the bus arbitration is done by the bus arbiter controller(not shown in figure). Once the Wishbone master gets a grant, it initiates the bus cycle by targeting a particular Wishbone slave [4]. The configuration is depicted in figure 2.8.

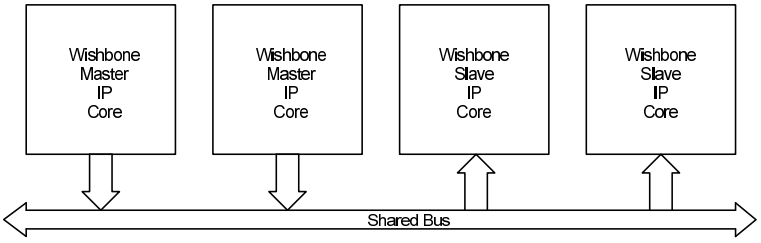


Figure 2.8: Shared Bus Interconnection [4]

2.7.3 Crossbar Switch Interconnection

In crossbar switch interconnection, there exist multiple Wishbone master and Wishbone slave. This topology is suitable for multicore SoC where a single master can access multiple slaves. The crossbar switch supports parallel data transfer and hence multiple master can access different slaves at a time. There exists a bus arbiter controller(not shown in figure) that decides which master can have access to which slave; the complete interconnect system can have concurrent interconnect and hence the data transfer is faster compared to previous topologies [4]. The interconnection is depicted in figure 2.9.

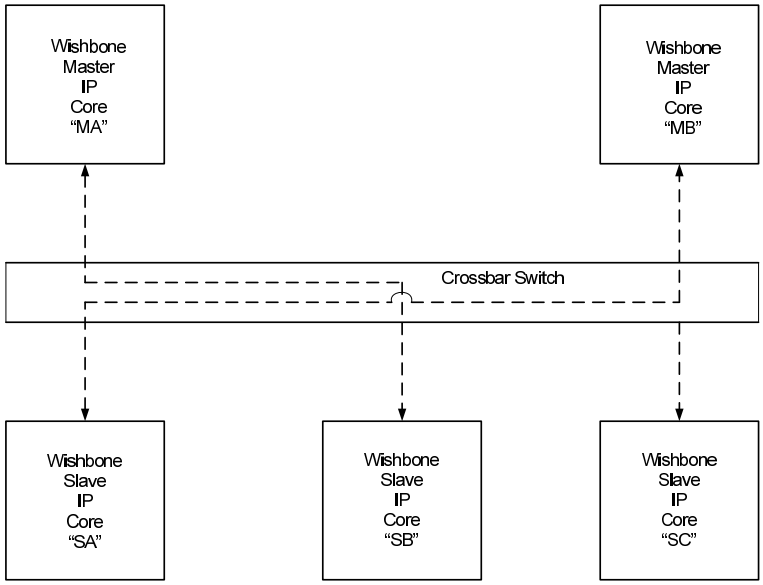


Figure 2.9: Crossbar Switch Interconnection [4]

2.7.4 Data Flow Interconnection

The data flow interconnection provides data access similar to the sequential data processing. Each IP core contains both Wishbone master and Wishbone slave; the configuration enables the interconnect to the next Wishbone master with Wishbone slave and hence forming a sequential chain. The data flows from one IP core to the next IP core; the data flow allows to support parallelism and hence data processing is fast [4]. The data traffic is controlled by the handshake signals. The topology is shown in figure 2.10.

Henceforth all the discussion about Wishbone bus will be w.r.t point-to-point interconnection Wishbone topology.

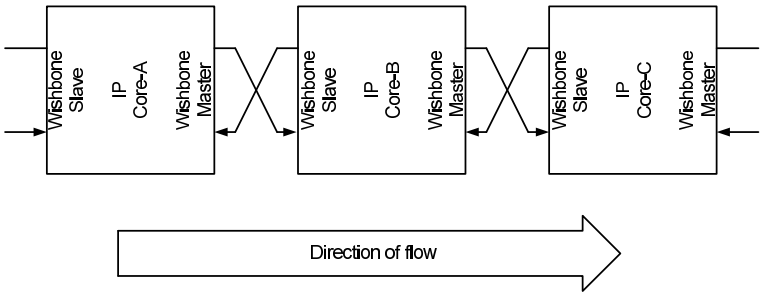


Figure 2.10: Data Flow Interconnection [4]

Chapter 3

Metastability and Synchronizers

This chapter mainly focuses on metastability and synchronizers. In this project it is important to know the basics of metastability since it acts as a main challenge to overcome during transmission of data between two different clock domains. Metastability problems are common in digital circuits and we use synchronizers to overcome its effect.

3.1 Metastability and its effect

Metastability is a state of logic in digital circuits which is indecision between logic-1 and logic-0. To explain more precisely, we present an analogy of golf trap as shown in figure 3.1.

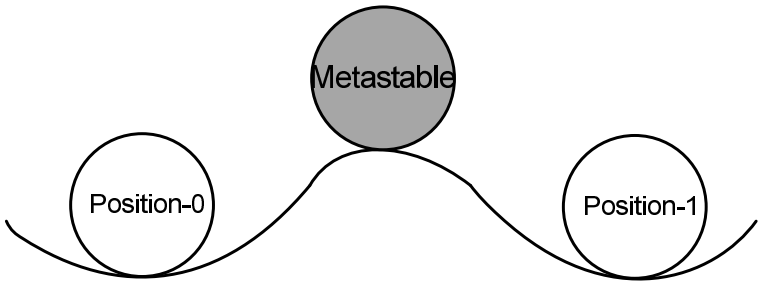


Figure 3.1: Mechanical metastability

If the driving force is very little then the golf ball settles at the position 0 and if the driving force is high then golf ball settles in position 1. However while transforming from one position to another it requires to cross a small hill and if the ball stops in this position then it cannot be predicted whether the ball falls into position 0 or position 1 and the ball is said to be in metastable state. Similarly if the output of a digital circuit is changing from one state to another state and if the sampling of the logic occurs at the time of transition then it result to an indecision state called a metastable state [5].

3.1.1 Metastability in flip flop

Metastability can occur in a flip flop because of violation of, setup or hold times. Consider a D-flip flop shown in figure 3.2.

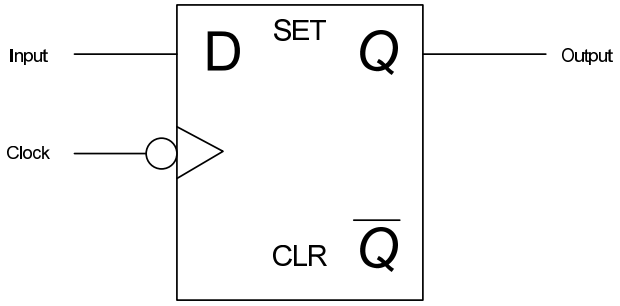


Figure 3.2: D-flip flop

The setup and hold times are shown in 3.3. If an asynchronous input data D_a is stable during setup and hold times, then the output Q_a of a flip flop is stable and valid as depicted in figure 3.4. If the same input data D_b changes during setup time then it is a setup time violation and if the same input data D_c changes during hold time then it is a hold time violation.

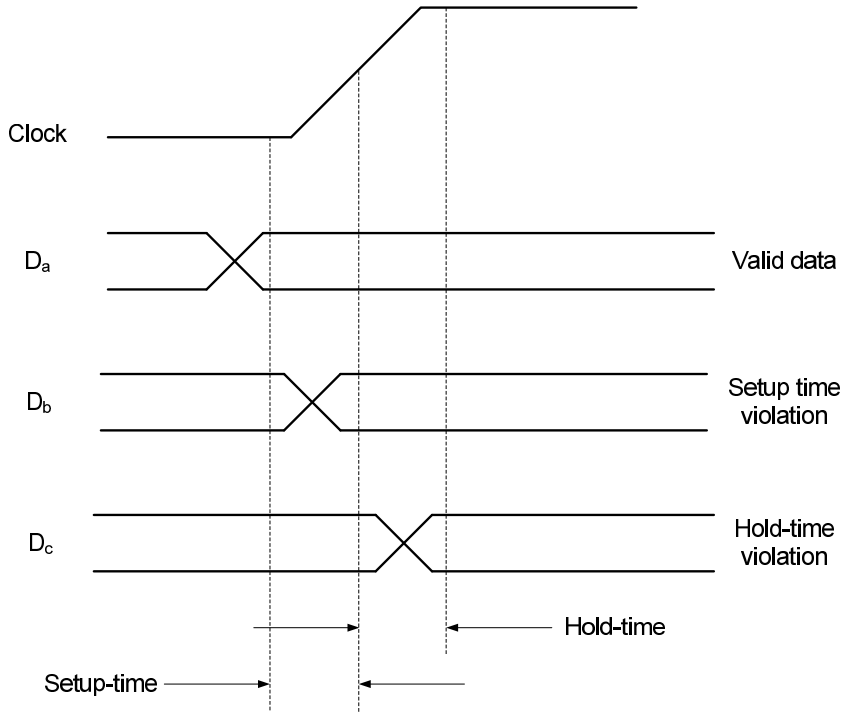


Figure 3.3: Inputs to D-flip flop with Setup and Hold time violation [6]

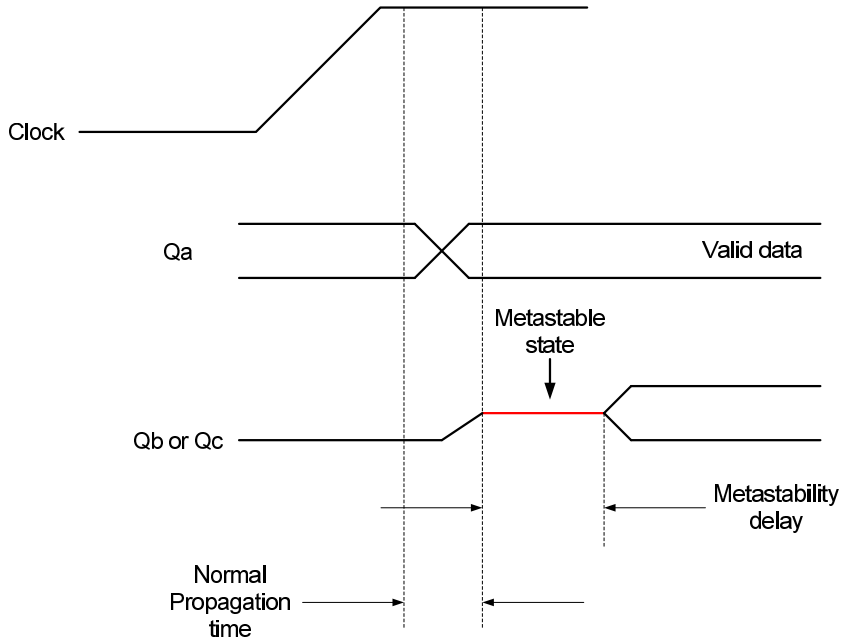


Figure 3.4: Output of D-flip flop with metastability [6]

If any one of the violation occurs then there are four possibilities that can occur to the output Q:

- The output Q_b or Q_c may oscillate between valid states for a long time and results to metastability as shown in figure 3.4.
- The output Q may take a state, because of missing the valid state of input data.
- An increase in output propagation delay may occur and hence the next connected element can see wrong value.
- The output Q may move to the correct state with small or no propagation delay.

It is seen that the probability of metastability occurrence is more as compared to other possibilities. Complete explanation of metastability at circuit level of flip flop is given in [5].

3.1.2 Effects of Metastability

Metastability may lead to malfunction in digital electronics circuits. It may lead to failure of a logic and eventually causes malfunction of the whole system. It is not possible to determine the active duration of metastability, hence it must be considered to be random during simulation. It is a common situation in high complexity circuits with multiple clock domains.

3.1.3 Minimizing the problems of metastability

There exist multiple approaches to overcome the metastability problem such as: design of system by paying special attention to setup and hold times, use of low frequency clock and use of synchronizers etc. The most feasible option to minimize the effects of metastability is to use synchronizers. Synchronizers increases the probability of supply of a stable data for sampling and gives sufficient time to resolve the metastability state of a logic signal. However the synchronizers may fail to perform its intended function and this leads to reliability calculation of a synchronizer.

3.2 Two flip flop synchronizer unit

Two flip flops connected back to back as shown in figure 3.5 can form a synchronization unit, however it still does not form a complete synchronizer.

Output of the flip flop "F1" i.e Q1 may become metastable but it is sampled one cycle later by flip flop "F2". Assuming that one clock cycle is enough to resolve the metastable state of Q1 we use this unit as synchronizer. The reliability of this unit can be derived by suitable mathematical treatment as discussed in the following section.

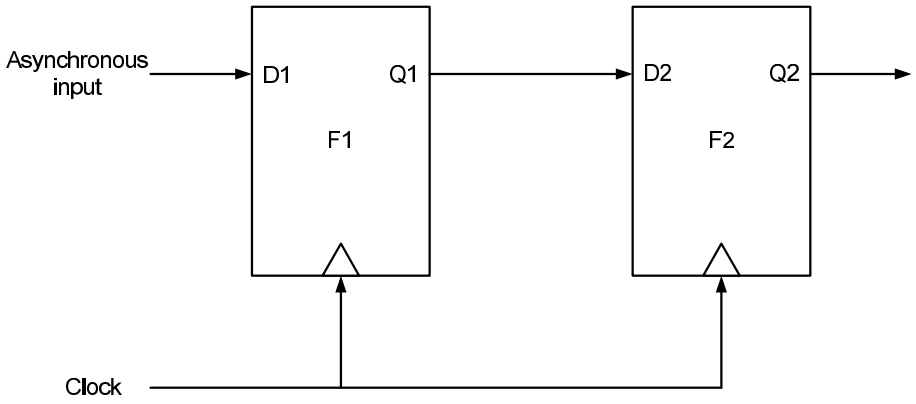


Figure 3.5: Two flip flop synchronization unit [5]

3.2.1 Reliability of a synchronizer

First we proceed to calculate the failure rate of a synchronizer. We use probability treatment to calculate how likely a flip flop is in the situation to enter into metastable state at the time of chance of switching of clock, and data are unknown. For a simple model, we make an assumption that an asynchronous input to a flip flop is likely to change at any point of time with an uniform distribution [5].

Let us assume "T_w" is a window period defined around the sampling edge of the clock such that if the data toggles within "T_w" period of time then the flip flop may encounter metastability. If "T_c" defines period of one clock cycle and the data is assumed to be

changed in uniform distribution over "T_c" then the probability of flip flop "P_{fm}" entering metastability [5] is:

$$P_{fm} = \frac{T_w}{T_c} = T_w.F_c \tag{3.1}$$

Equation 3.1 assumes that input data toggles at every clock cycle, which is not realistic. Suppose data toggles with the toggle rate "F_d", then rate of metastability changes:

$$P_{fm} = T_w.F_c.F_d \tag{3.2}$$

Let us assume "S" is the synchronization period, i.e the time period for resolving the metastable state of the output of the flip flop. The actual failure of single flip flop synchronizer occurs when:

- The flip flop has encountered metastability after sampling edge of the clock.
- The metastable output has failed to resolve after "S" period of time.

Above two events are independent of each other and hence we can multiply their probability to achieve the probability of failure of flip flop failure:

$$\begin{aligned} P(FFfailure) &= P(\text{enter metastable state}) \times P(\text{time to exit metastability} > S) \\ P(FFfailure) &= T_w.F_c \times e^{\frac{-S}{\tau}} \end{aligned} \tag{3.3}$$

where "τ" is CMOS technology node dependant constant and is estimated to have a value of 10ps for a 28nm high performance CMOS technology [5].

The rate of failures with toggle rate of data as "F_d" is given by:

$$P(failure) = T_w.F_c.F_d \times e^{\frac{-S}{\tau}} \tag{3.4}$$

And the mean time between failure (MTBF) is given by:

$$MTBF = \frac{e^{\frac{S}{\tau}}}{T_w.F_c.F_d} \tag{3.5}$$

Consider figure 3.5, suppose if Q1 goes metastable then it is sampled by "F2" one clock later hence, $S = T_c$. A failure in two flip flop synchronizer is defined as Q2 being metastable but if input D1 switches from logic-0 to logic-1 and switching is very closer to the rising edge of the clock then there are six different outcomes [5] that may happen to Q2:

- Valid data is being switched at first clock cycle and Q2 copies the same data in second clock cycle.

- If the valid data after switching is missed by Q1 then, Q1 copies the valid data after one clock cycle; and later Q2 copies the same value delayed by another clock cycle. Thus there exist 2 clock delays to obtain valid Q2.
- "F1" enters metastability and its output Q1 remains low, after "S" period the metastability is resolved so that Q1 goes high. Q2 copies the resolved logic, rising to high during second clock cycle.
- "F1" enters metastability and its output Q1 remains low. After "S" period the metastability is resolved so that Q1 goes low. Now the situation is same as case - 2 later Q1 is forced to high in second clock cycle and Q2 copies the same, rising to high at third clock cycle.
- "F1" enters metastability and its output Q1 goes high, after "S" period the metastability is resolved so that Q1 goes low hence a glitch appears on Q1 between first and second clock cycle. However Q1 goes high at the start of second clock cycle and finally Q2 copies the resolved logic, rising to high during third clock cycle.
- "F1" enters metastability and its output Q1 goes high. After "S" period the metastability is resolved so that Q1 remains high so that Q2 copies the same and rises to high at second clock cycle

The main important point to note is that Q2 never(may be once in every MTBF) become metastable in any of the cases discussed above. Q2 switches from logic-0 to logic-1 in one or two clock cycle delay thus flip-flops "F1" and "F2" acts as a synchronizer unit. However once in every MTBF, Q1 may switch very near to the sampling edge of flip flop "F2" and thus causing metastability.

Connecting two flip flops does not form a complete synchronizer; this is because the sender has to know how long D1 has to be kept high and did "F1" sampled it or not? Synchronization fails if D1 is sampled when it holds a invalid value. Thus it is required to use handshake signals such as Request a.k.a REQ and Acknowledge ACK. The details of complete synchronizer built with two flip flop synchronizer unit is discussed in chapter 4.

3.2.2 Probability of failure distribution

Using equation 3.4, the probability of distribution can be analysed. Table 3.1 shows the values that are w.r.t 28nm CMOS technology [5] and are used to get a negative exponential plot which demonstrates $P(failure)$ vs S.

Figure 3.6 shows the negative exponential curve and infers that, increase in metastability resolution time will decrease the probability of failure. The reciprocal of probability of failure is MTBF.

Designers would decide the required MTBF of the synchronizer and then the maximum metastability resolution time can be derived. However the tail of the curve would not meet zero reference line as shown in figure 3.7, this is because metastability resolution time requires infinite length to completely resolve the metastability and is not practical.

Parameter	Value
T_w	20 ps
F_c	1 GHz
F_d	100 MHz
τ	100 ps

Table 3.1: Values used for plotting $P(failure)$ vs S

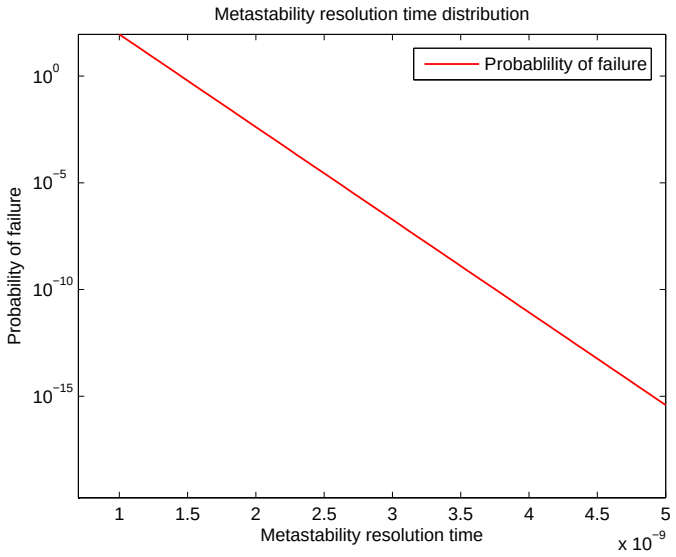


Figure 3.6: Probability of failure distribution

3.3 MUTEX element as a synchronizer unit

MUTEX is a two port element as depicted in figure 3.8. It is a mutually exclusive element whose main function is to resolve the contention between two inputs from independent sources R1 and R2. Once contention is solved then the inputs are passed to corresponding outputs G1 and G2 respectively such that only one output is active at any point of given time. If only one request arrives first then the other request is blocked until the first request is de-active [7].

If signals of both input arrives at the same time then, the device goes into metastability. To avoid metastability at the output, a metastability filter is employed [7]. The complete circuit of MUTEX element is shown in figure 3.9.

The metastability filter composes of two inverters driven by the output of bistable circuit. In figure 3.9, the top inverter of metastability filter connect to output G2. Before the MUTEX flips, the output of bistable circuit x1 and x2 are held high. This drives G1

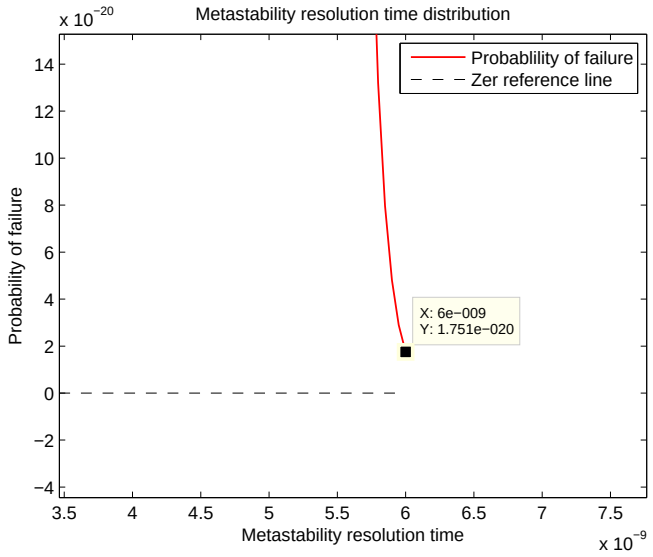


Figure 3.7: Tail of probability of failure distribution

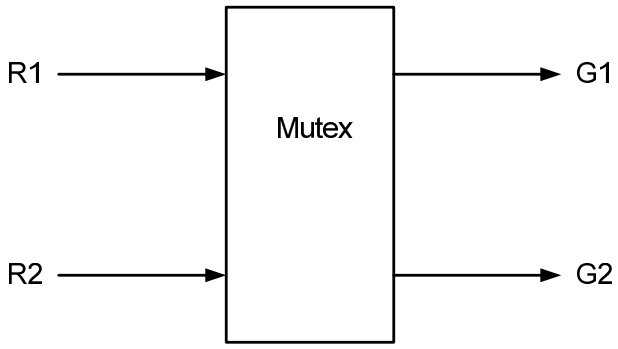


Figure 3.8: MUTEX element [7]

and G2 output to low.

Once the MUTEX flips, x1 or x2 will go low; suppose x1 goes low and x2 is still high, then top inverter is in active hence G2 remains low while G1 goes high. This case is depicted in Case-1 of figure 3.10. Similarly if x2 goes low and x1 is still high, then bottom inverter is in active hence G1 remains low while G2 goes high, this case is depicted in Case-2 of figure 3.10.

Suppose both inputs R1 and R2 goes high almost at the same time then x1 and x2 becomes metastable momentarily but the output G1 and G2 remains low (same as the case of before MUTEX flips). The time taken to resolve the metastability is random and once the MUTEX flips one of the output will be pulled high as shown in Case-3 and Case-4 of figure 3.10.

The bottom line is G1 and G2 never become metastable and the contention is solved

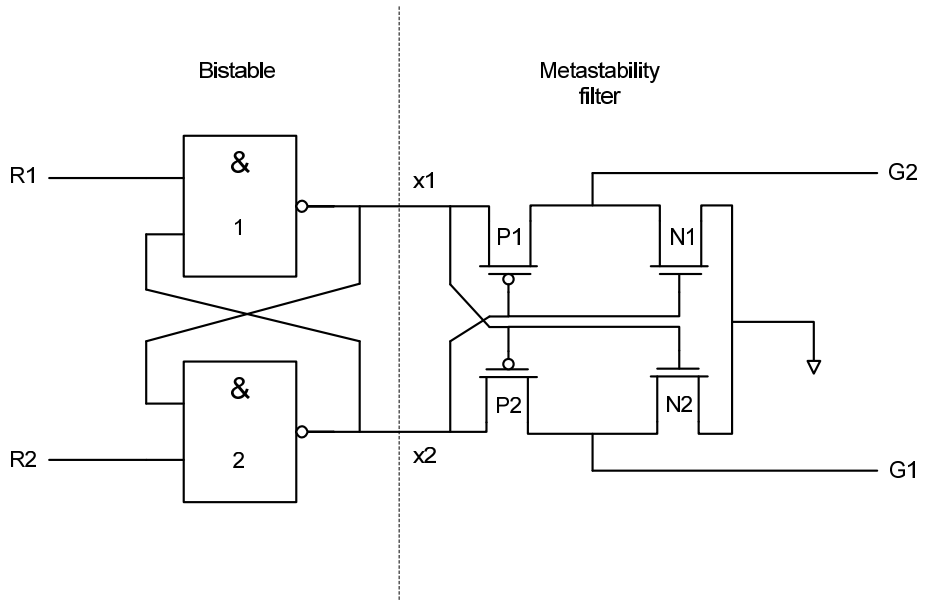


Figure 3.9: MUTEX circuit [7]

between inputs R1 and R2. Assuming any one of the inputs to the MUTEX is clock and another input is a data then the chances of metastability of the signals(inputs) is zero. Hence MUTEX element can also be used as synchronizer unit. However use of MUTEX element alone does not form a complete synchronizer but it can be used as a main element of the synchronizer bridge. The synchronizer bridge built with MUTEX element is described in chapter 6.

Two different synchronizer bridges - a 4 phase two flip flop synchronizer bridge [8] and Locally delayed latching bridge [9], are investigated and discussed in next chapters.

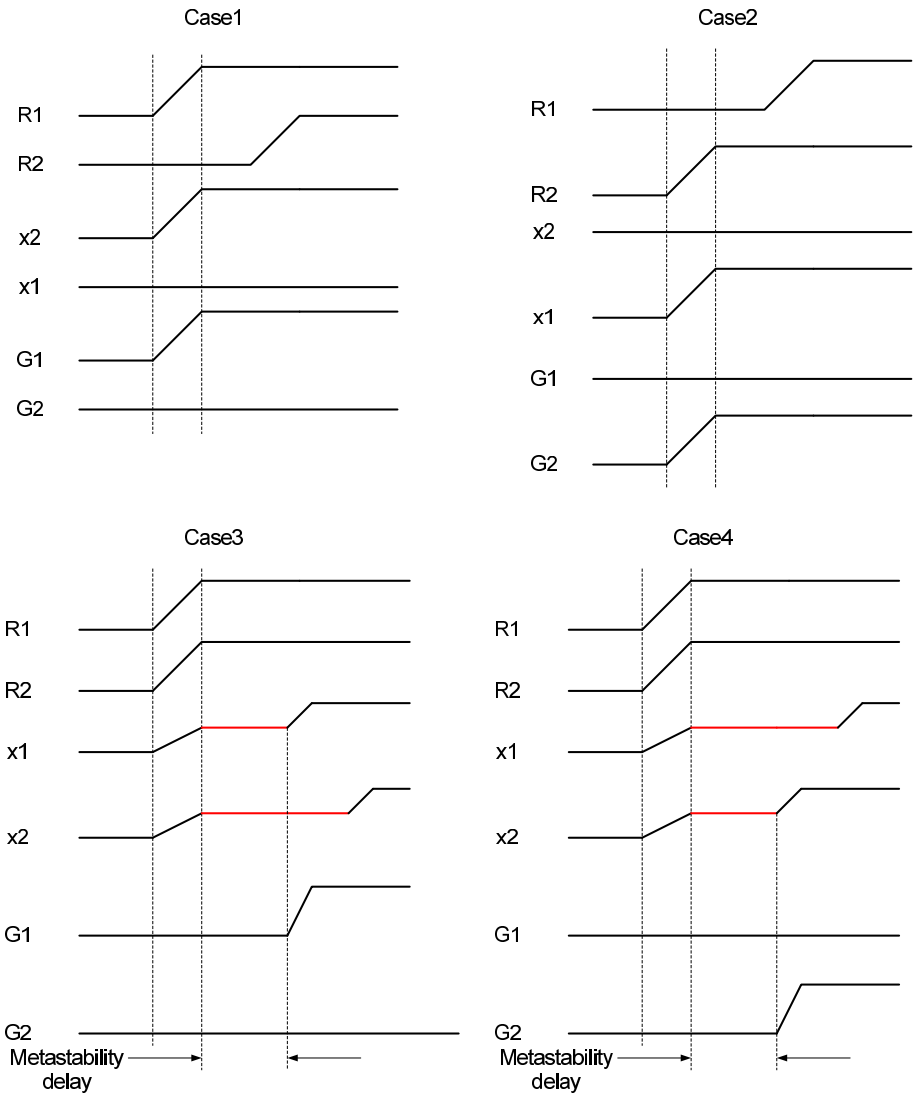


Figure 3.10: Input and output waveforms of MUTEX

Chapter 4

Four-phase two flip flop synchronizer bridge

Synchronizer built with two flip flop synchronizer unit is depicted in figure 4.1. Flip flops "F1", "F2", "F4", and "F5" are the main synchronization unit while other components are used to maintain 4-phase bundled-data transition sequence. Before understanding the synchronizer, we first discuss the concept of 4-phase bundled-data handshake protocol.

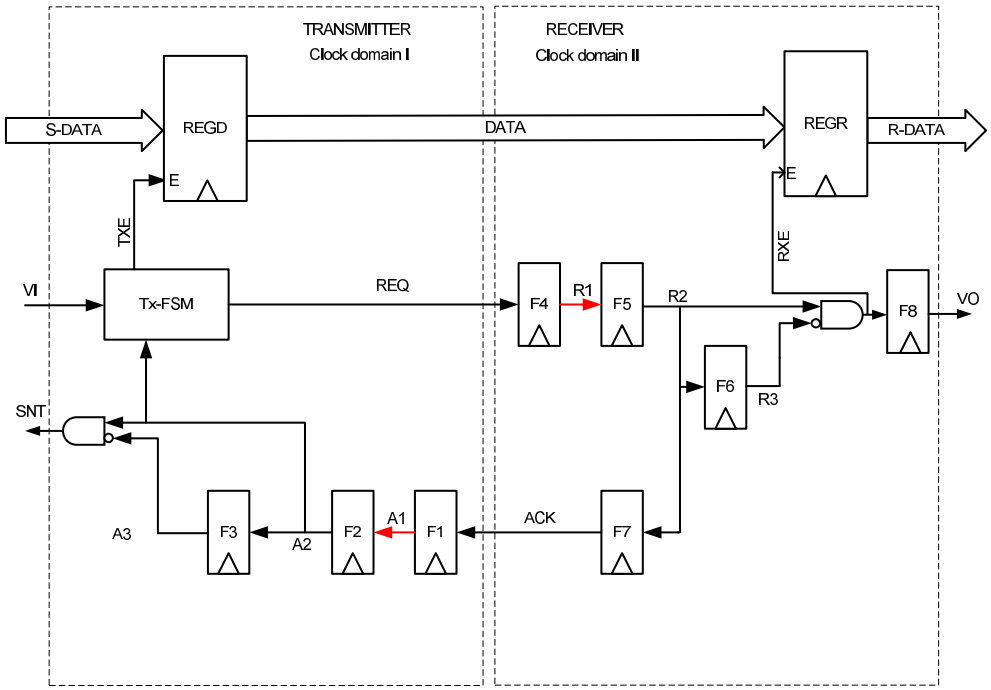


Figure 4.1: Two flip flop 4 phase synchronizer bridge [5]

4.1 4-phase bundled-data protocol

The term "bundled" refers to control set of data signals(bundle) by the handshake signals. Here the data signals use the normal boolean encoding to represent the data. Basically these signals follow RZ level signalling [7]. Figure 4.2 describes the 4-phase bundled data transition. As discussed in chapter 3, a complete synchronizer consists of handshake signals such as REQ and ACK.

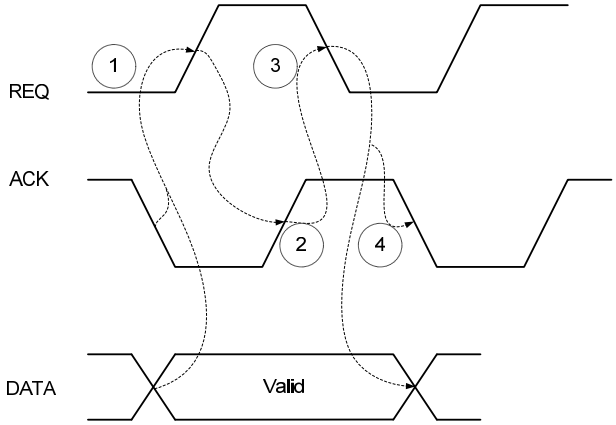


Figure 4.2: 4-phase transition sequence [7]

- The handshake signal REQ makes a transition from low to high if and only if ACK is low and the data to be transmitted between two different clock domain is stable. This phase is first transition.
- Once the REQ is detected, the ACK makes a transition from low to high and this forms another phase called second phase of transition.
- Next the REQ takes a transition from high to low after detecting ACK is at signal level high. This is a third phase of transition.
- Finally the ACK transits from high to low after detecting REQ has made a transition from high to low and the data can now change its value.

There exist other different kind of handshake protocols but they are considered as out of scope in this project. More details on those protocols can be found in [7].

4.2 4-phase two flip-flop synchronizer

Figure 4.1 depicts complete architecture of 4-phase two flip-flop synchronizer. It consists of Transmitter and Receiver; each is driven by separate clock. The synchronizer uses - two flip flop synchronization unit.

The handshake signal crossing one clock domain to another has a high probability of becoming metastable, hence two flip flops are provided to sample them and give enough time of one clock cycle to resolve the metastability. The red connecting line in figure 4.1 should not be involved in any kind of logic. If required metastability resolution time is more than one clock cycle, then additional flip flop can be added in between the two flip flops.

Figure 4.1 shows the finite state machine used in transmitter and overall state transition graph of the synchronizer [8] [9].

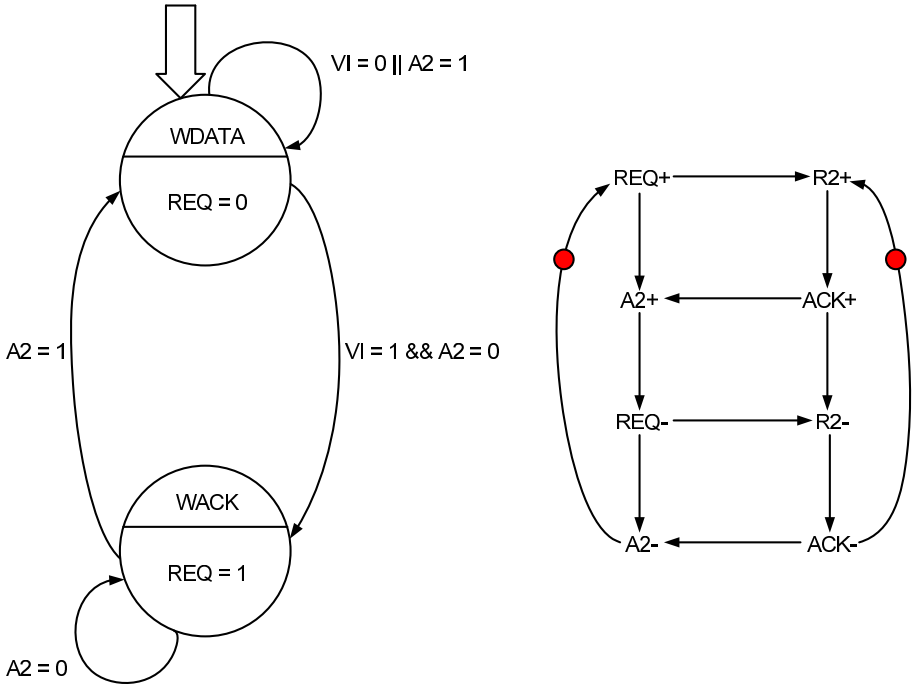


Figure 4.3: Tx-FSM and STG used in 4-phase two flip flop synchronizer [5]

The REQ is asserted by transmitter if and only if there exist valid data on the bus and VI is high. This occurs if the transmitter has completed its previous cycle i.e. A2 = 0. VO is pulsed at the receiver side for every new data reception cycle. The SNT is pulsed at the transmitter once the data is successfully transmitted i.e. A2 is received and thus synchronized. These actions of the handshake signals resembles the 4-phase bundled-data protocol and hence the name two flip flop 4-phase bundled-data synchronizer.

Thus the failure of two flip-flop synchronizer would occur once in every MTBF. This failure causes the synchronizer to let the metastable state signals to pass into the systems. This potential failure may lead to malfunction in the system once in every MTBF.

In the next chapter 5 we see how the two flip flop 4-phase synchronizer can be used for the Wishbone bus whose master and slave are driven by two different clock domains to synchronize the data and signal of Wishbone bus.

Chapter 5

Four-phase two flip flop synchronizer bridge applied to Wishbone bus

This chapter gives an overview and explains the application of two flip flop synchronizer bridge to Wishbone bus.

Here, Wishbone master and Wishbone slave are driven by different clock domains respectively so as to emulate multi-clock domain architecture. The synchronizer bridge is placed in between Wishbone master and Wishbone slave so as to provide required synchronization and make sure that Wishbone signals are synchronized and data is transferred safely between multi-clock domain as shown in figure 5.1.

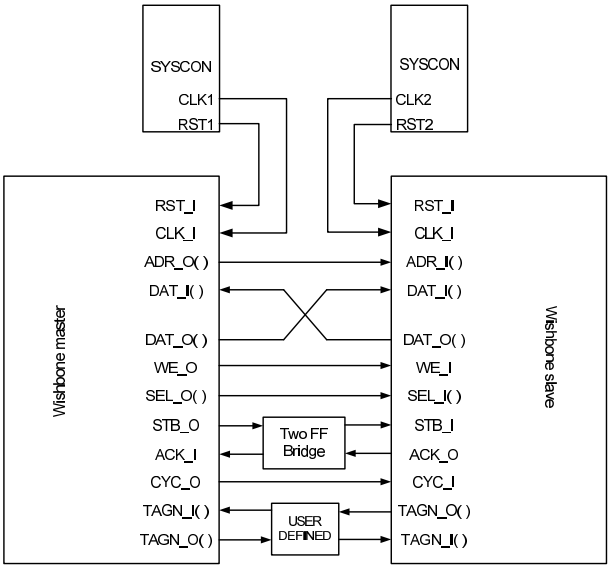


Figure 5.1: Two flip flop synchronizer applied to Wishbone bus

5.1 Wishbone signals STB, ACK and 4-phase bundled-data protocol

Wishbone signals STB and ACK acts as handshake signals between Wishbone master and Wishbone slave resembling the 4-phase bundled-data protocol. After careful observation of Wishbone read/write (single/block) data cycles with the 4-phase bundled-data protocol (figure 4.2). it can be interpreted that Wishbone STB and ACK follows the 4-phase bundled-data protocol and hence most of the components of a standard two flip flop 4-phase synchronizer are being already embedded in Wishbone architecture. It is just the two flip flops to be added between multi-clock domain signal transition to avoid metastability. Hence the synchronizer reduces to just two flip flop connected back to back as shown in figure 5.2.

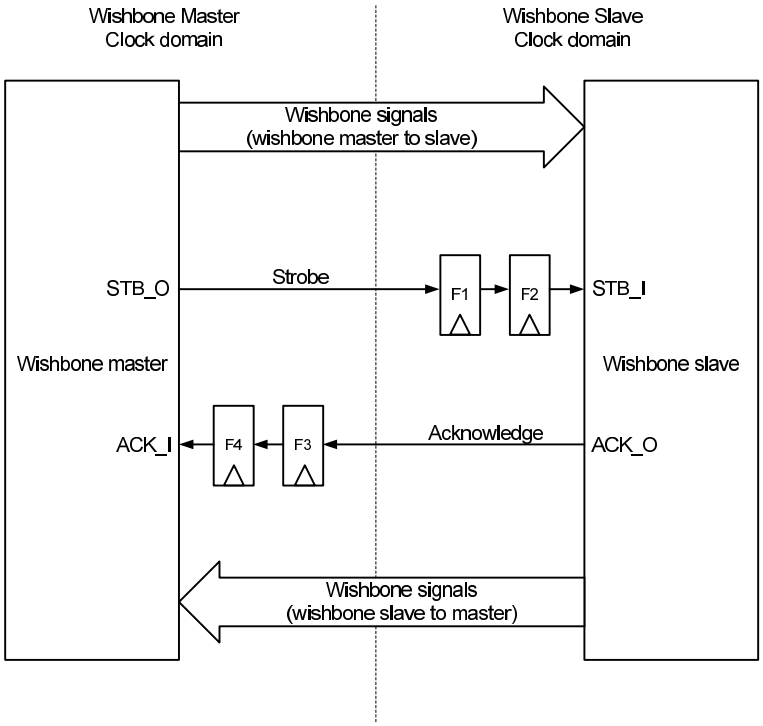


Figure 5.2: Two flip flop synchronizer applied to Wishbone bus

The Wishbone STB and ACK signals are initiated only if the other signals of the Wishbone bus are stable. Thus all other signals are safely transited between multi-clock domains before the handshaking between Wishbone master and slave begins. Only the Wishbone ACK and STB signals are prone to metastability and hence the two flip flops are being added only to these paths as shown figure 5.2.

The MTBF calculation for the two flip flop synchronizer bridge will remain same as discussed in chapter 3.

The complete block two flip flop 4-phase synchronizer used for Wishbone bus is shown

in figure 5.3.

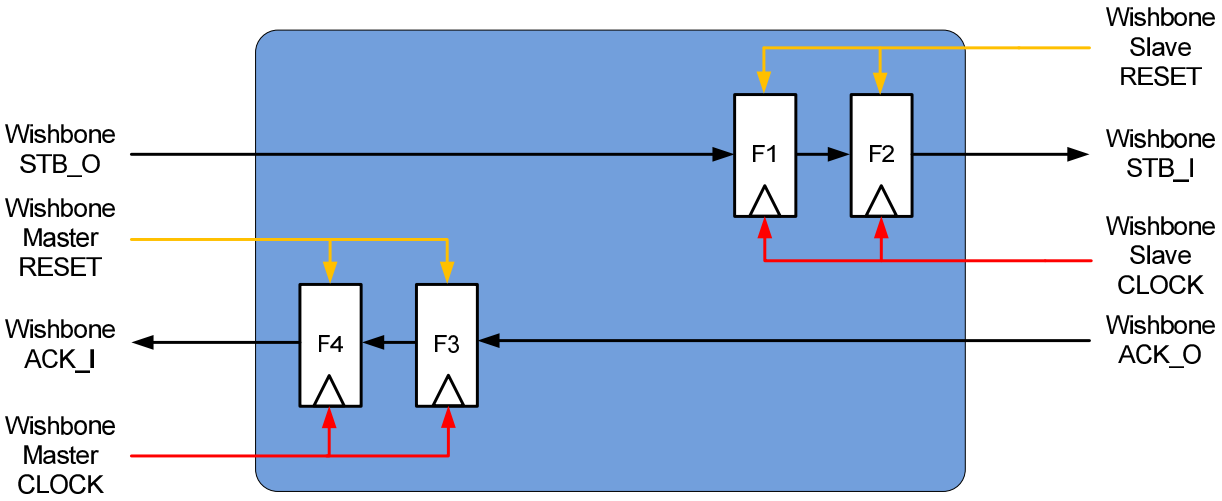


Figure 5.3: Two flip flop 4-phase synchronizer applied to Wishbone bus

Chapter 6

Introduction to GALS and LDL Synchronizer

This chapter gives a brief introduction to GALS concept. GALS implementation is another view on synchronization problem. It gives a model to SoCs for safe transition of data between multiple clock domains.

6.1 GALS wrapper

GALS defines a self timed wrapper around local synchronous island in multi-core-SoCs which are locally synchronous. Figure 6.1 shows the self timed wrapper. Locally synchronous island is wrapped with input and output port driven with local clock generator. The clock drives only those elements present in the wrapper. The input and output ports has the port controllers responsible to generate handshake signals used for asynchronous communication between other locally synchronous island core which will be wrapper with GALS self timed wrapper.

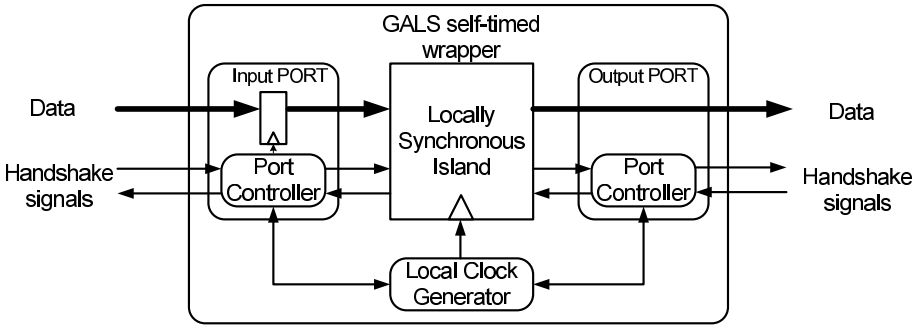


Figure 6.1: GALS wrapper [1]

The input port latches the data at proper time when data is stable. The latching is controlled by the input port controller with appropriate generation of handshake signals.

The output port is responsible to send the data to other local island core. It takes care of queuing and stalling the data when other local island core is slow or not ready to accept the data.

There are different kinds of synchronizers designed w.r.t GALS definition, all of them differ in the design of port controllers [1]:

- Stoppable clock generation
- LDL Synchronization
- Two clock FIFO synchronizer

This project is devoted to explore only LDL synchronizer. In the next section the LDL synchronizer is explained.

6.2 LDL Synchronization

Locally delayed latching synchronizer is a zero latency synchronizer [9] which means the time from the synchronizer to write a data word at output register of transmitter to writing the same data into first register of the receiver is zero. The latency induced by the multi-clock domains is unavoidable.

Two clock FIFO synchronizer would induce the data latency if the queue is empty and affects directly to the system speed. Stoppable clock generation also costs the latency delay induced by the clock tree distribution. However the LDL synchronizer is considered as a zero latency synchronizer, and would provide a chance to sample the correct data as soon as earliest possible sampling edge of the clock [9].

The LDL synchronizer has a GALS wrapper with input and output ports. In the following section, the input and output ports are discussed.

6.2.1 LDL input port

Figure 6.2 shows the block diagram of LDL input port. It has a port controller and is responsible for the generation of handshake signals that are necessary for the synchronization. In this synchronizer the system clock is not affected by the synchronization methods used, hence system clock is not delayed and is independent of the dynamic scaling of system clock [10] [11] [12].

The asynchronous controller controls the latch L and input data latch Y1 of the locally synchronous island. It also issues the Valid signal at every new data latch to the port and thus prevents data hazards such as write-after-read [9].

Suppose if there is a conflict between REQ and clock Y. Instead of modifying REQ or Y, controller delays the latching of data in "R1" by delaying the Y1+ and Y1- is unaffected. However the data is latched at the time of data arrival in "LATCH" with help of signal L during acceptance of REQ at low phase of Y. The conflict between the REQ+ and Y+ are resolved with help of MUTEX element present inside the controller. The details of controller and synchronization procedure is explained in the section 6.2.2. The timing assumptions for this synchronizer is explained in the next section.

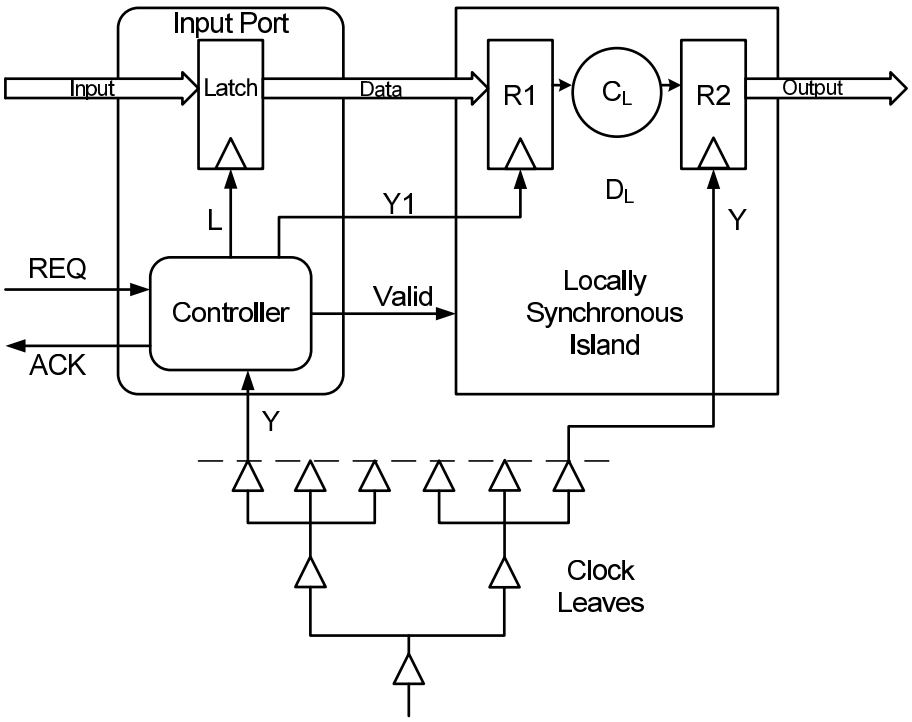


Figure 6.2: LDL input port with locally synchronous island [9]

6.2.2 LDL input port circuit

Figure 6.3 shows the complete details of LDL input port. The main part of this port is MUTEX element which acts as a synchronizer unit. Its main job is to resolve the metastability that would occur when there is conflict between REQ+ and Y+. The Muller-C element is responsible to provide the input ASK' to the MUTEX at every new request ASK arrival from the asynchronous controller [9].

The output of the MUTEX element is being latched by asynchronous S-R Latch. If the new REQ+ wins the contention then, L is set and the data is latched, thus providing a Valid signal to the locally synchronous island. If Y+ wins the contention then the S-R latch is reset and Valid is de-asserted meanwhile, Y1+ is delayed to latch the stable data at "R1".

Figure 6.4 shows the STG of asynchronous controller used in this input port. The circuit of asynchronous controller and corresponding waveforms are depicted in the same figure.

6.2.3 Timing assumptions for LDL input port

Figure 6.5 shows the timing budget assumption for the LDL synchronizer.Time allocated to resolve the metastability is based on the MUTEX element. It is assumed that sufficient time is allocated to resolve the metastability when the conflict between Y+ and REQ+

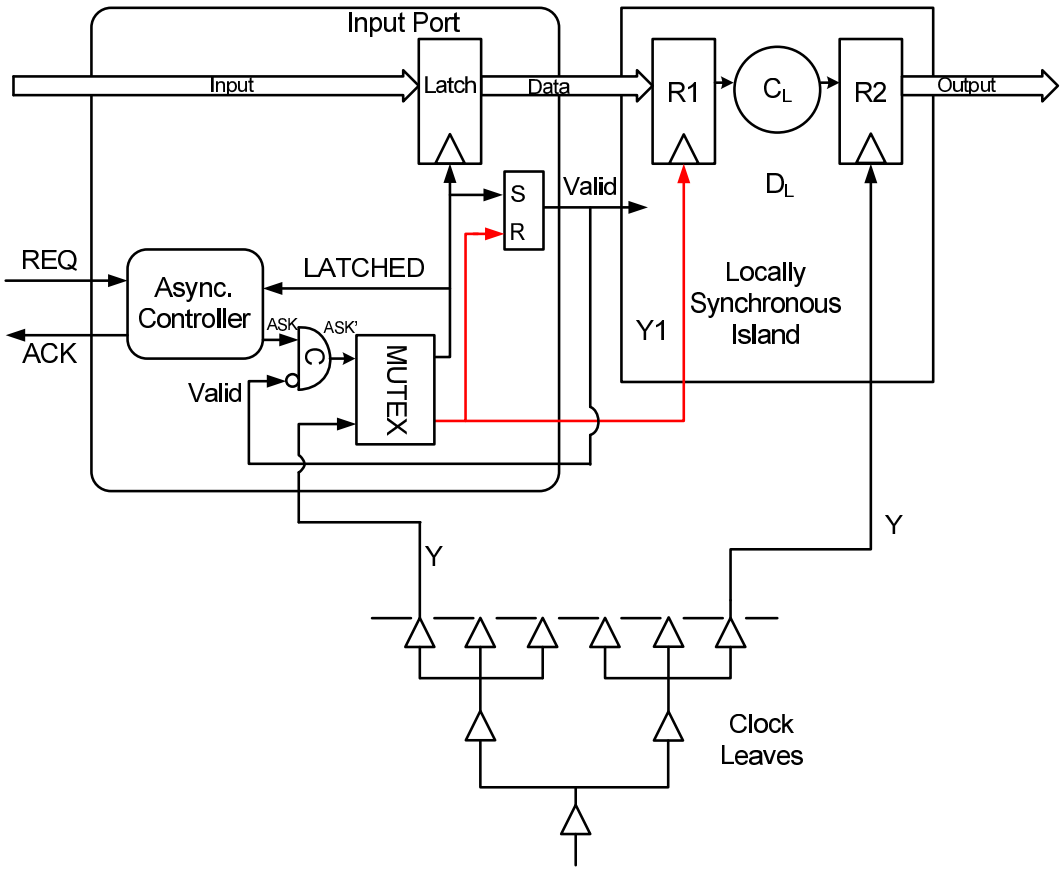


Figure 6.3: LDL input port circuit [9]

is imminent. The other time budgets are dedicated for time delay "D_{CTRL}" that occurs for data latching in the port and it depends on the architecture of the asynchronous controller and minimum width of high phase "HP" of clock which is enough to sample the data at "R1".

Y1+ is delayed by the controller until the time allocated for metastability expires. Once the metastability is resolved, the data is stable and hence it is being latched by "R1".

6.2.4 LDL output port circuit

Figure 6.6 shows the output port of the LDL synchronizer. It requires an additional circuit to be embedded within the locally synchronous island. The main function of the output port is to provide the synchronization with incoming ACK signal from the LDL input port of another locally synchronous island.

The required STG to be produced by the async.controller is shown in the figure 6.7. The D_L block at would provide the required digital logic for the pulse generation.

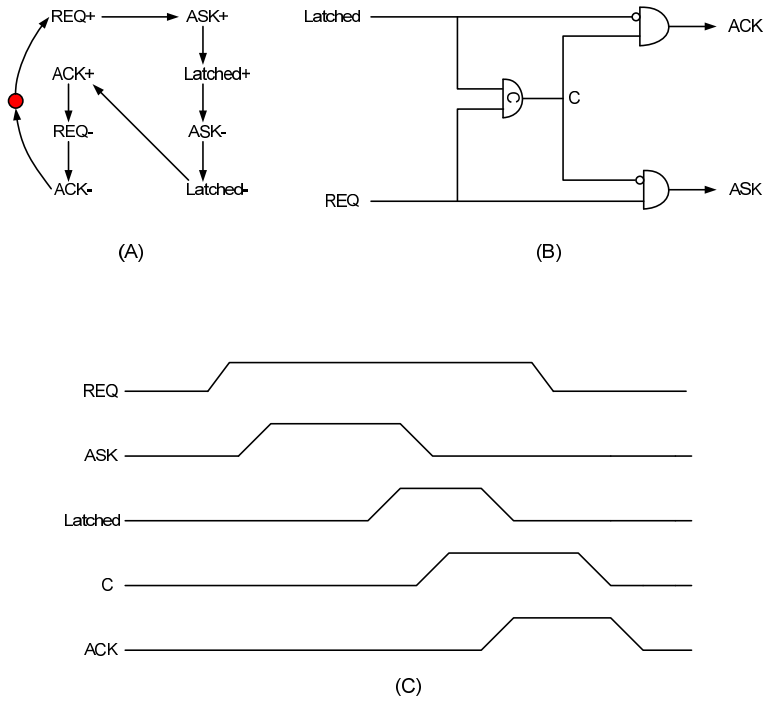


Figure 6.4: LDL input port STG, async. controller and handshake signals [9]

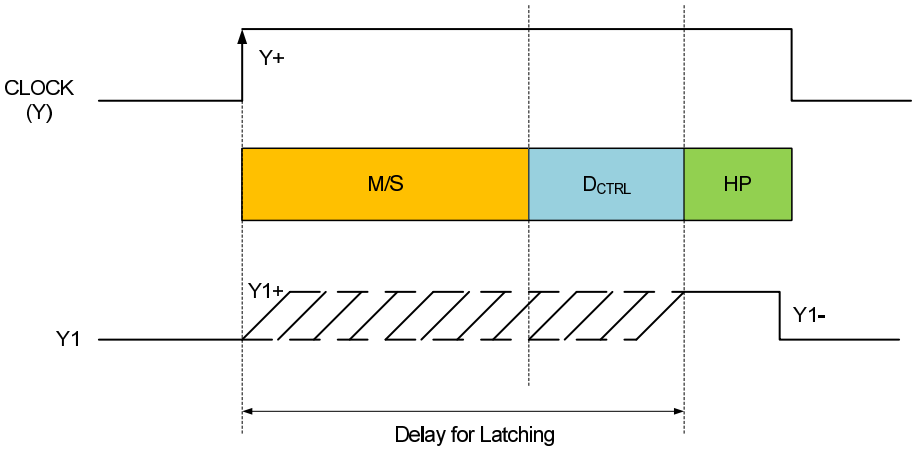


Figure 6.5: LDL timing budget [9]

6.2.5 LDL operating modes

The main concern that LDL handles is the conflict between REQ and system clock Y resulting in metastability. Figure 6.8 shows overall operation and operating modes of a standard LDL bridge. Apparently four different cases must be discussed.

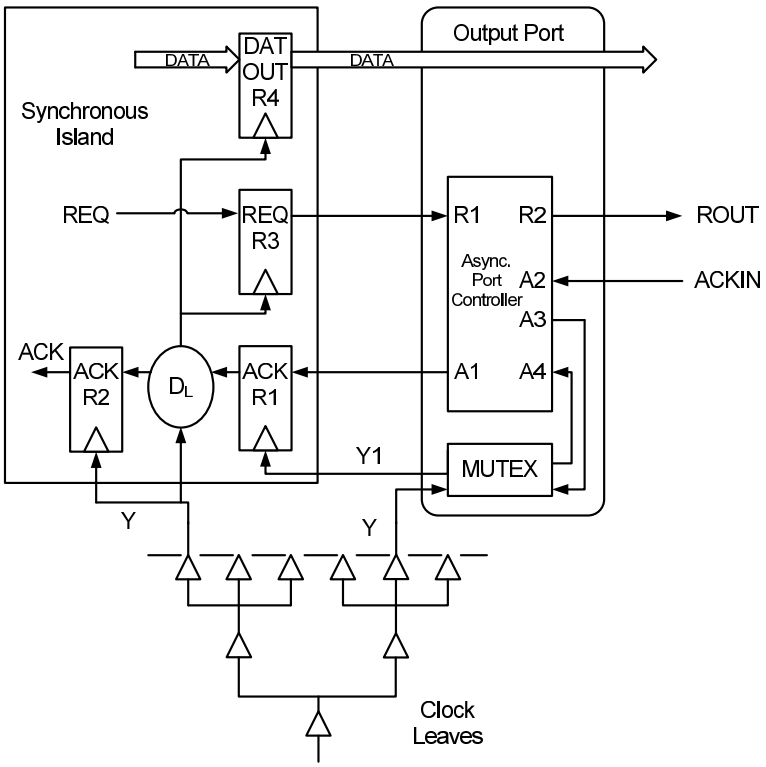


Figure 6.6: LDL output port [1]

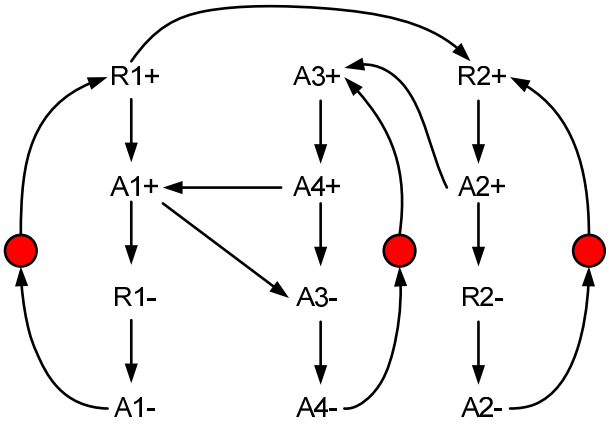


Figure 6.7: LDL output port STG [1]

- No conflict: In case of no conflict between REQ+ and system clock Y+ then there is no metastability. Also the signals such as REQ and ACK follows the 4 phase transition as shown in figure 6.8 under "Conflict-1" column. The minimum time period maintained by the REQ signal is same as the minimum time period required to latch the data by the flip flop "R1" in figure 6.3.

- Conflict between REQ+ and Y+, REQ+ wins : In column "Conflict-1" of figure 6.8, there exist the conflict between REQ+ and system clock Y+ (marked with red dots). The REQ has a normal time period. In this case, the MUTEX element enters into metastability because of two input signals REQ+ and Y+ rising simultaneously. The MUTEX delays the signal Y1+ so that the data latched by "R1" is not metastable and the delayed width of Y1+ is enough to resolve metastability after which MUTEX grants Valid [13] to locally synchronous island as shown in figure 6.3. Thus the locally synchronous island receives the data which is not metastable. This makes LDL synchronizer avoid metastable data entering into locally synchronous island. This conflict is also the worst case(in terms of latency) scenario of all the cases that would occur in LDL synchronization.

- Conflict between REQ+ and Y+, Y+ wins: In column "Conflict-2" of figure 6.8, there exist the conflict between REQ+ and system clock Y+ (marked with red dots). The MUTEX enters into metastable state. Suppose the MUTEX grants the access to Y then the pulse width of Y1+ is not altered however, the latching of the data in the input port by the latch is delayed. This causes the delay in stable data appearing to locally synchronous island.

- No conflict with reduced cycle : Suppose REQ+ pulse appears much advance than clock Y and at the proper time to sample REQ by Ythen latching of data in input port is much quicker and also the latching of stable data in locally synchronous island is quick since the delay of Y1+ is minimum compared to all above cases. Also the local clock to the synchronous island in all above cases remain uninterrupted.

6.2.6 LDL Performance and reliability

In this section we discuss about the best case and the worst case of LDL synchronizer. Later we also analyse the reliability of LDL.

- Best Case: The reduced cycle case is the best case that would occur in LDL synchronizer. Here there is no conflict between REQ+ and Y+, thus the time assumed for metastability resolution is zero and the controller delay is minimum, however the required pulse width of the high phase should be maintained in the implementation so that the high phase is enough to sample the data at "R1". The REQ+ is being sampled at the ideal instant of time and the difference between the positive edges of Y and Y1 is minimum. The over all details are discussed in "No conflict with reduced cycle" in section 6.2.5.

- Worst Case: Suppose there is a conflict between REQ+ and Y+, the REQ+ wins the contention then there exist a delay in Y1+. This delay includes the time required for metastability resolution introduced by MUTEX element; if this time period is too long then it contributes to the overall latency of data transfer. Also the time given to combinational logic next immediate of "R1" is less compared to all cases.

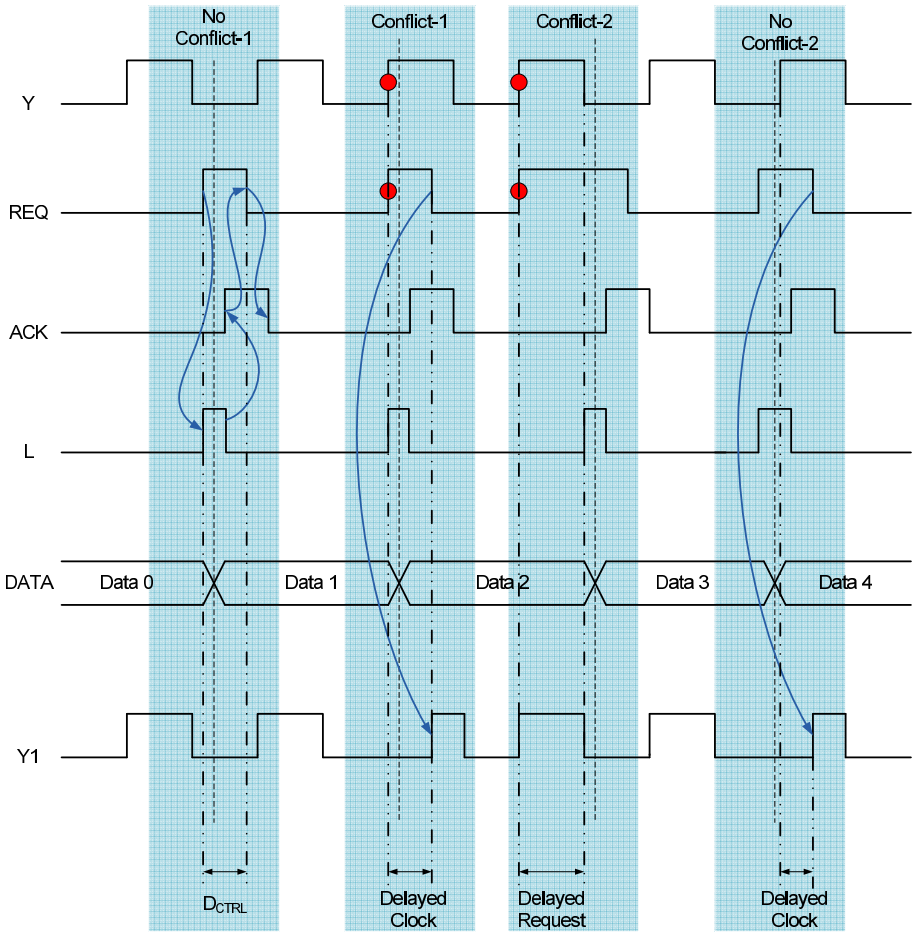


Figure 6.8: LDL operating modes [1] [13]

The overall details are discussed in "Conflict between REQ+ and Y+, REQ+ wins" in section 6.2.5.

Reliability Analysis

We define the following terms that are useful in reliability analysis:

- *Resolution Latency*: The metastable MUTEX element is said to be resolved if the value stored in bi-stable circuit assigns to logic-0 or logic-1 and the output of MUTEX is evaluated. The time spent on this work is called as resolution latency. The resolution latency is unbounded and indeterminate.
- *Failure*: Failure is said to occur if the metastable MUTEX is not resolved within predefined maximum time period "$T_{m/s}^{MUTEX}$".

- *Safety*: A circuit is said to be MTBF-Safe if the time for failure between two successive failures exceeds the desired MTBF.
- *Minimum High Phase Clock*: "T_{HP}^{min}" is defined as minimum high phase width of clock required for flip flop to latch a data. The "T_{HP}^{min}" is about three FO4 delay of a inverter gate delay w.r.t its CMOS technology node.

Suppose we design a SoC with required MTBF of at least 100 years and which has K = 100 synchronizers. To achieve this we need to have the MTBF of each synchronizer to be at least K times the MTBF of SoC [14] i.e. the each synchronizer must have MTBF of at least 100 × 100 = 10,000 years.

The shortest clock cycle for a digital IC built with standard EDA tools is about 100-160 FO4 inverter inverter gate delays [15]. Thus the high phase of a fastest clock cycle is 50 FO4 inverter gate delays long.

The MTBF of the MUTEX element is given by [1]:

$$MTBF = \frac{e^{\frac{T_{m/s}^{MUTEX}}{\tau}}}{T_w.F_c.F_d} \tag{6.1}$$

To calculate the worst-case MTBF, we assume that the values, τ = 1 FO4 inverter gate delays and T_w = 2 inverter gate delays [16]. Also for worst-case MTBF analysis, let $F_c = F_d$ and the clock cycle time period be T = 100τ. This simplifies to,

$$F_c = F_d = \frac{1}{100 \times \tau} \tag{6.2}$$

Hence MTBF in terms of number of gate delays "N" is given by,

$$MTBF = \frac{e^N}{2 \times \frac{1}{100} \times \frac{1}{100}} \times \tau \tag{6.3}$$

For MTBF $\geq$ 10000 years, and $10^{-11} < \tau < 10^{-10}$s, we calculate "$T_{m/s}^{MUTEX}$" in terms of "$N$" as,

$$\frac{e^N}{2 \times \frac{1}{100} \times \frac{1}{100}} \times \tau \geq 10^4 \tag{6.4}$$

$$41 < N < 43 \tag{6.5}$$

We see that for a system clock of T = 100τ, at least one half of the clock period is needed to resolve the metastability to achieve above mentioned MTBF. And for the slower metric clock T = 160τ it requires one fourth of the clock period to resolve metastability to achieve the same MTBF. This assumption is suitable for the SoCs that have a system clock in terms of MHz.

The minimal high phase to be maintained in LDL synchronizer is given by [1] [13],

$$\frac{T}{2} - D_{CTRL} - T_{m/s}^{MUTEX} > T_{HP}^{min} \tag{6.6}$$

Also the controller delay must satisfy,

$$D_{CTRL} < \frac{T}{2} - T_{HP}^{min} - T_{m/s}^{MUTEX} \tag{6.7}$$

Failure in LDL synchronizer can happen once in MTBF. However the failure can lead to unknown delay during which the time taken to resolve the metastability is too high. This failure cannot lead to induction of metastable signals into the system but if the delay is too high then it may lead to system level malfunction.

Chapter 7

LDL synchronizer bridge applied to Wishbone bus

In this chapter we discuss how the LDL synchronizer bridge can be applied to Wishbone bus. Optimization in standard LDL synchronizer is required in order to make the LDL bridge suitable to Wishbone bus. According to standard Wishbone bus, the Wishbone master signals such as STB_O and ACK_I must follow 4-phase bundled data protocol. The transition of these signals occurs only when other Wishbone signals are ready on their respective bus.

The main observation we have to make here is the Wishbone signals such as strobe and acknowledge signals undergo transition very frequently and hence they are prone to metastability when they cross between multi-clock domains. Thus those signals are fed into LDL synchronizer bridge as shown in figure 7.1.

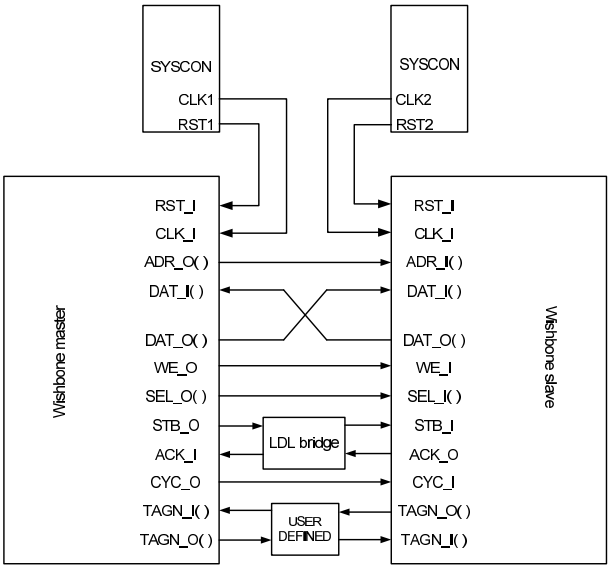


Figure 7.1: LDL synchronizer bridge applied to Wishbone bus

7.1 Optimizations in standard LDL synchronizer before applying to Wishbone bus

Before applying LDL synchronizer to Wishbone bus, it is required to make optimizations in it in order to make LDL synchronizer suitable to Wishbone bus. Below are the optimizations made:

- Removal of ACK signal in LDL synchronizer bridge: The Wishbone bus has 4-phase bundled data protocol. Hence it is not required to have additional ACK signal of a standard LDL synchronizer.
- Removal of output port in LDL synchronizer bridge: Since there is no dependency on LDL input port ACK signal, it is not required to used output port in our application to Wishbone bus. Hence only input port is used in LDL synchronizer used for applying Wishbone bus.
- Removal of data register in LDL synchronizer bridge: The Wishbone bridge already has the register to latch the data. Hence there is no need to have extra registers which are present in standard LDL input port.

Considering all above optimizations, the LDL synchronizer input port reduces to a simple synchronizer input port as shown in figure 7.2.

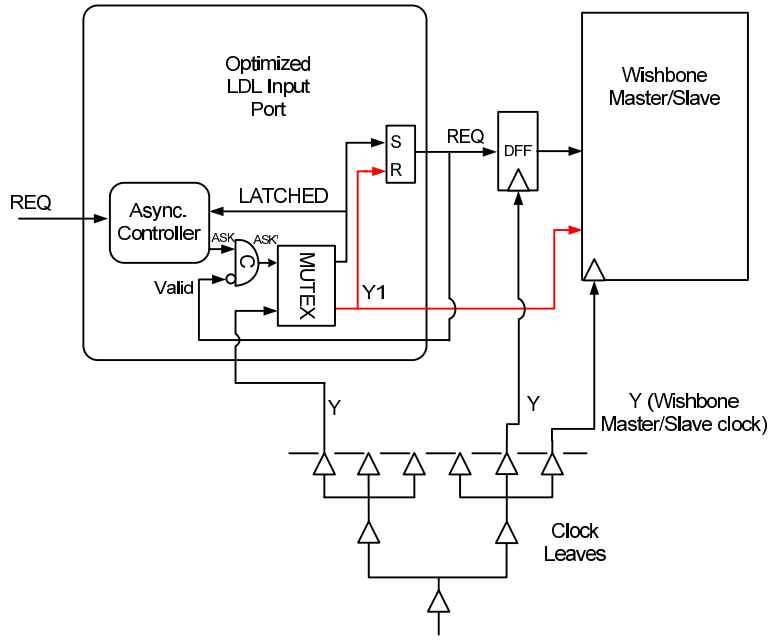


Figure 7.2: Optimized LDL input port used for Wishbone bridge

The signal Y1 in figure 7.2 is still provided so that in future scope of this work, if Wishbone interrupt, Wishbone error signals and other Wishbone extra signals are

suppose added then, if they are more prone to switching while crossing multi-clock domain then, they have to be latched with a register separately with help of Y1.

The application of this optimized LDL bridge would make the synchronizer very simple and would fit suitably for Wishbone bus. The overview of this application is shown in figure 7.3.

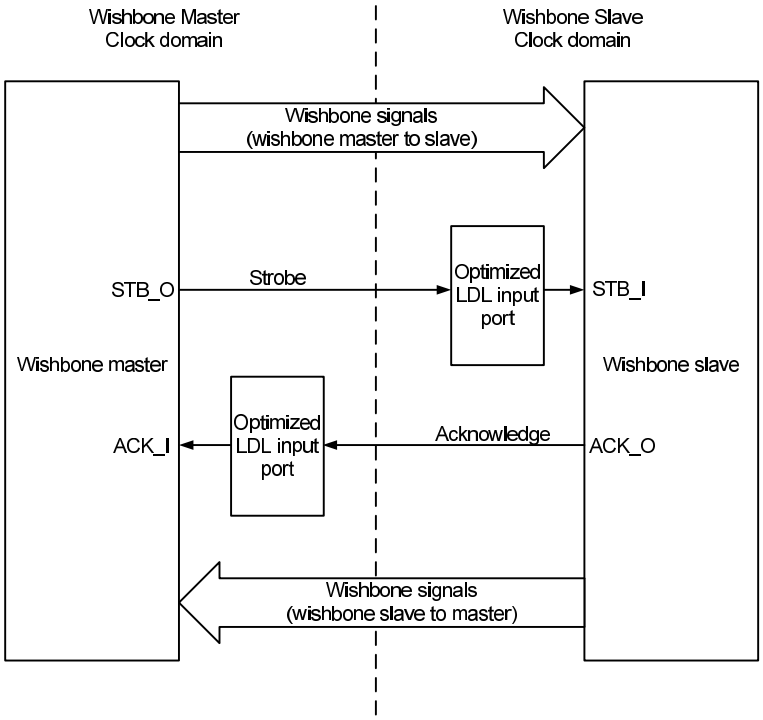


Figure 7.3: Application of optimized LDL bridge to Wishbone bus

Figure 7.4 gives a complete Optimized LDL bridge that is used for synchronization in multi-clock domain for Wishbone bus. In the next chapter we will give an overview of verification process that we carried out in this project.

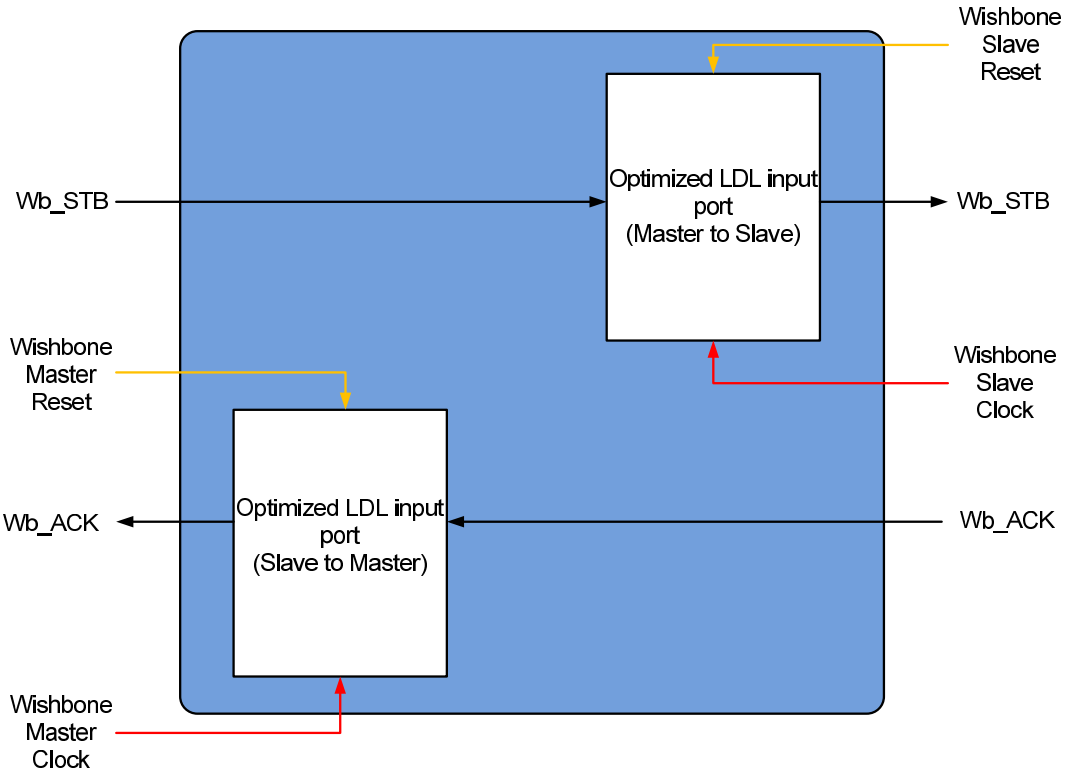


Figure 7.4: Optimized LDL synchronizer bridge

Chapter 8

Verification of synchronizer bridge using GPIO IP core

This chapter introduces to the verification process that we carried out in this project. We used GPIO IP core made available by OpenCores to verify our designed synchronizer bridges. The GPIO is a general purpose Input Output user programmable core which is made up of bunch of registers. The block diagram of GPIO IP core is shown in figure 8.1.

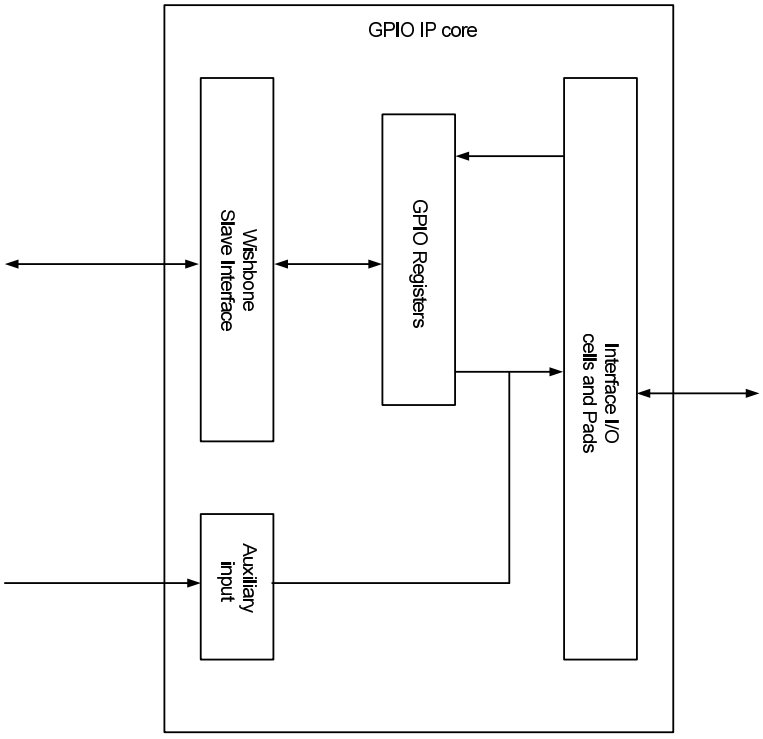


Figure 8.1: GPIO IP core architecture [17]

There exists a Wishbone slave which is a part of GPIO IP core [17] and we make use of it to verify our synchronizer bridges. The Wishbone master is also provided in the GPIO IP core as a part of verification accessory.

We used the following tools for simulation and debugging:

- HDL Simulator: Synopsys VCS [18]
- Debugging and Visualization: Synopsys DVE [18]

8.1 Verification procedure:

The Auxiliary input is not required for us hence we remove it and connect the Wishbone master along with our bridges. The whole architecture of synchronizer bridge verification is shown in figure 8.2.

The Wishbone master and GPIO core are driven by different clocks and thus emulates multi-clock domain environment. The synchronizer bridge which we designed are connected between Wishbone master and Wishbone slave(part of a GPIO core).

The test bench is placed at the top level which sets up the test environment as shown in figure 8.2. It is required to verify standard Wishbone single read/write and standard Wishbone block read/write cycles after insertion of synchronizer bridge, for different variation of clock frequencies. This makes our verification complete.

8.1.1 Verification of standard Wishbone single read cycle:

The test bench generates a random data of suitable size (test vector) and writes it into a register of GPIO core externally i.e. using "I/O interface" of GPIO core. Later the test bench triggers the Wishbone single read cycle providing the address of the register in which data was set externally before. Once the data is fetched by the test bench through Wishbone master, it is given to comparator. The comparator compares the random generated data and the fetched data from Wishbone master. If both data are equal then Wishbone single read cycle works fine, hence the comparator gives the output "OK"; if both data are unequal then the comparator gives the output "Not OK". The whole procedure is repeated with different test vectors. This makes the Wishbone single read cycle verification complete.

8.1.2 Verification of standard Wishbone single write cycle:

The test strategy remains same for the single write cycle; the only difference is that the single chunk of data is written into GPIO register via Wishbone single write cycle and verified by reading the GPIO register using external "I/O interface" of GPIO core.

8.1.3 Verification of standard Wishbone block read cycle:

It is required to use more than one register of GPIO core for verification. The test bench generates two different random data (test vectors) and writes them into two different

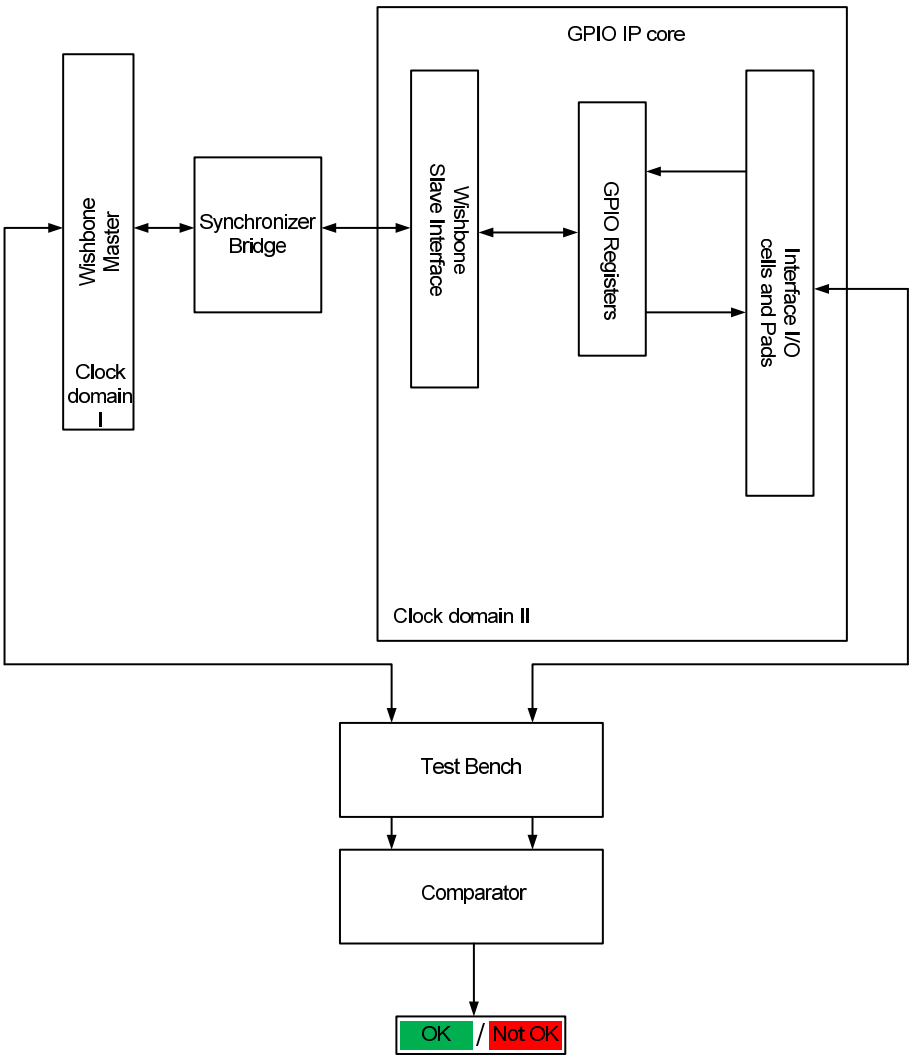


Figure 8.2: Verification architecture

registers of GPIO core which has different address, the writing of data is done externally using "I/O interface" of GPIO core. The test bench initiates the Wishbone block read cycle by providing two address of the register within one active cycle. The Wishbone master fetches the data one by one using different address under a single cycle, thus two chunks of data forming one block. The fetched data and the generated data are applied to comparator, if the data blocks are equal then Wishbone block read cycle works fine hence comparator gives the output "OK", if data block are unequal then comparator gives the output "Not OK". The whole procedure is repeated with different test vectors. This makes the Wishbone block read cycle verification complete.

8.1.4 Verification of standard Wishbone block write cycle:

The test strategy remains same except that multiple block of data are written into GPIO registers using Wishbone block write cycle and verified by reading the GPIO registers data via external "I/O interface" of GPIO core.

All above test cases are repeated for both two flip flop synchronizer and LDL synchronizer with multiple and different clock frequencies of Wishbone master and GPIO core. If all the test cases are passed then the designed synchronizer is adaptive to the Wishbone bus and hence proves the design is good enough to synchronize the data for multi-clock domain.

Master Clock	Slave Clock	Test Count	Result (OK/Not OK)
250 MHz	10 MHz	20	OK
200 MHz	25 MHz	20	OK
100 MHz	75 MHz	20	OK
100 MHz	100 MHz	20	OK
75 MHz	125 MHz	20	OK
50 MHz	150 MHz	20	OK
25 MHz	200 MHz	20	OK
10 MHz	250 MHz	20	OK

Table 8.1: Test cases for different master/slave clock frequencies

Table 8.1 shows the regression test that was carried out individually for two flip-flop and LDL synchronizer bridges to verify the behaviour level of the implementation. The regression was carried out for both single and block read/write cycle of the Wishbone bus with synchronizers.

The regression was also carried out for the verification of the netlist thus forming netlist simulations with other clock frequencies that are suitable for 180nm CMOS technology library. The netlist simulation details are given in chapter 9.

Chapter 9

Hardware

This chapter gives all the required details of the hardware that was synthesized after the design of two flip-flop synchronizer and LDL synchronizer.

The RTL synthesis was carried out using Synopsis Design compiler [19] by performing technology mapping with 180nm technology library. The synthesis includes the synchronizer bridges included within the OpenCores GPIO IP core, hence the GPIO IP core was also synthesized along with the synchronizer bridge. There exist one important synthesis constraint for the LDL synchronizer:

- Synthesis of MUTEX and Muller-C element: The MUTEX and Muller-C element are not synthesized since they are not part of a standard cell library. However the MUTEX and Muller-C element is made as a black box and the functionality is reflected from the behavioural design. Thus the design compiler treats both of them as a simple black box during synthesis.

9.1 Netlist simulation

Once the netlist obtained after performing RTL synthesis of both synchronizers, they were both subjected to simulation with the test bench as described in chapter 8. Table 9.1 gives the different configuration of clocks that were used to set up the netlist simulation.

The netlist simulations were carried out considering different process parameter variations called process corners [20]. The process corners used here are named as “min” and “max”. The characteristics of these process corners are tabulated in table 9.2.

9.2 Area and Power estimation

Both synchronizer bridges were synthesized along with the GPIO IP core. It is obvious that the area of the synchronizer bridges were too small compared to GPIO IP core since the core has Wishbone slave and set of registers. However, the presence of hardware constraints such as treating MUTEX and Muller-C element as a black box would cause a constraint in estimating the area and power. The estimation will be genuine if and

Master Clock	Slave Clock	Test Count	Result (OK/Not OK)
100 MHz	25 MHz	20	OK
50 MHz	100 kHz	20	OK
10 MHz	30 kHz	20	OK
500 kHz	400 kHz	20	OK
400 kHz	500 kHz	20	OK
30 kHz	10 MHz	20	OK
100 MHz	50 MHz	20	OK
25 MHz	100 MHz	20	OK
30 kHz	30 kHz	20	OK
10 MHz	10 MHz	20	OK

Table 9.1: Netlist simulation data

Process corner	Voltage (v)	Temperature (degree Celsius)	Process type
max	1.6	100	slow
min	1.95	-40	fast

Table 9.2: Process parameters

only if there exist no hardware constraints. Comparison of area and power estimation is also omitted because of the same reason.

9.3 Place and route

A standard place and route was carried on and resulting layouts are depicted in the following figures.

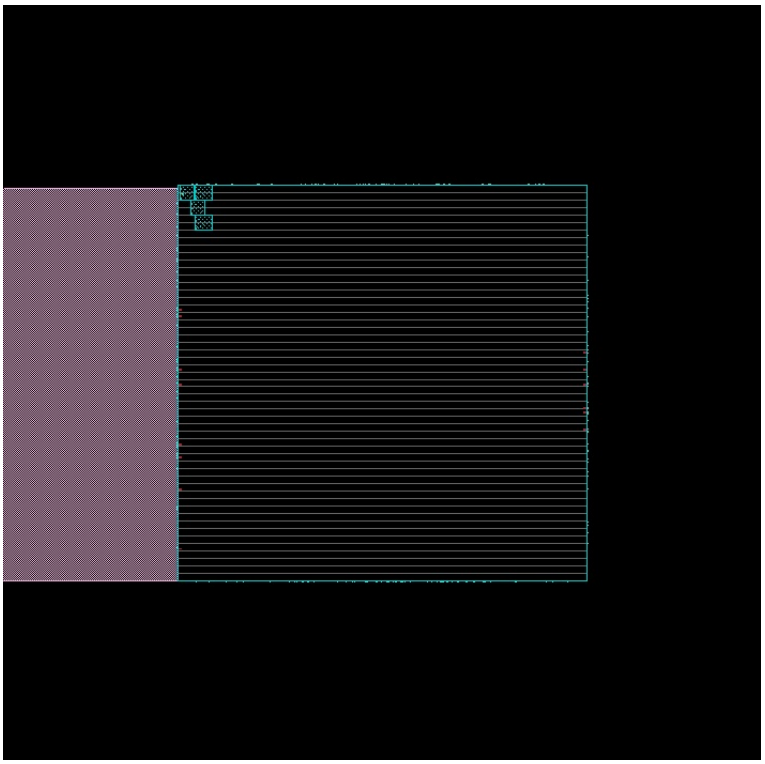


Figure 9.1: Floor plan for two flip-flop bridge with GPIO IP core

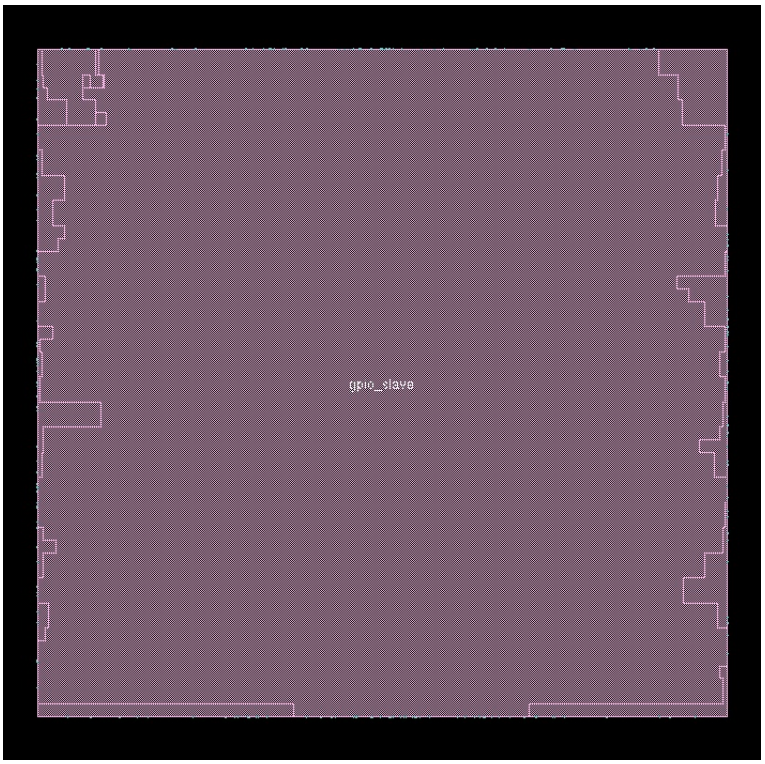


Figure 9.2: Ameobioic view classifies two flip-flop bridge and GPIO IP core

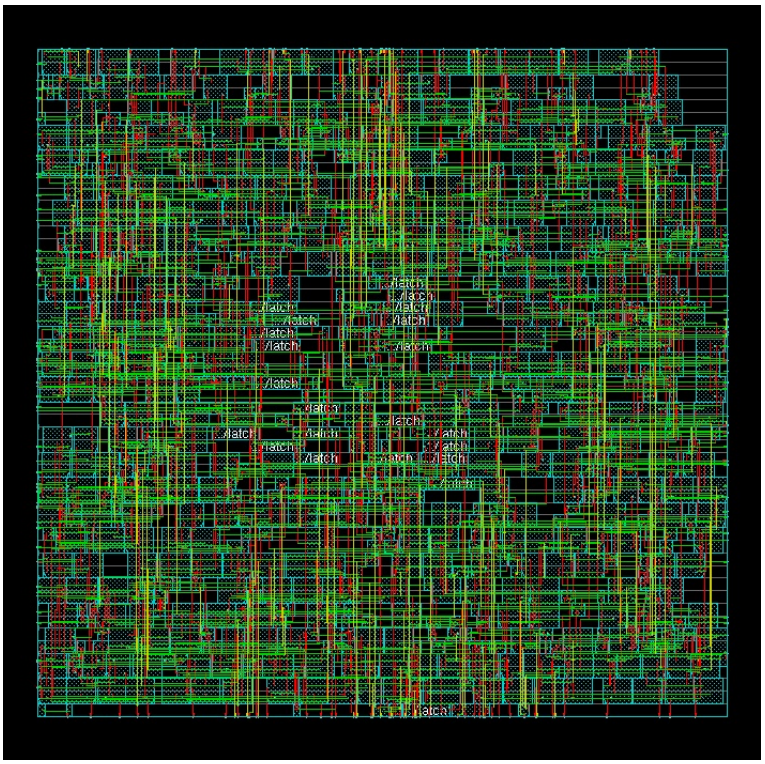


Figure 9.3: Layout of two flip-flop bridge with GPIO IP core

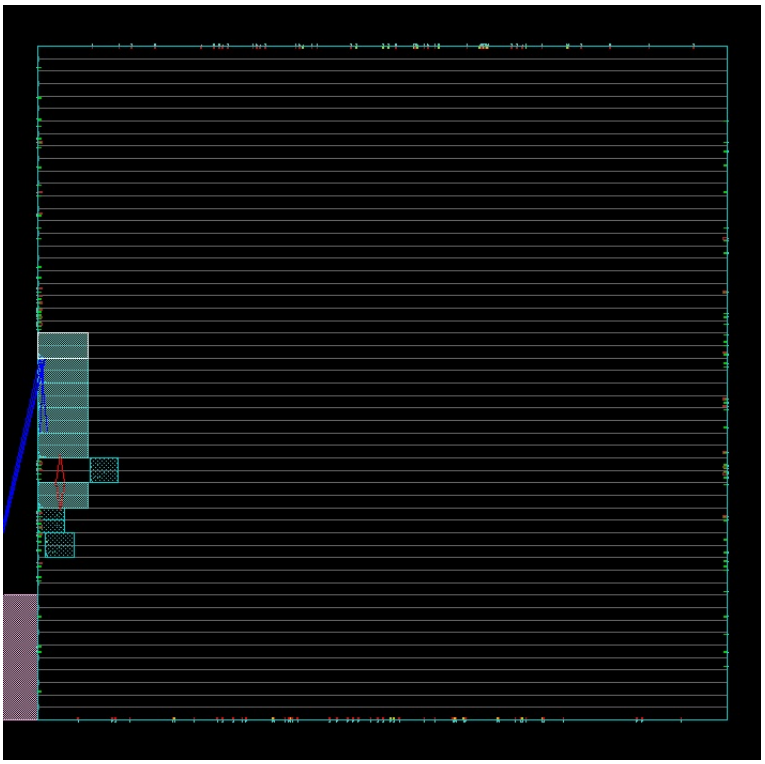


Figure 9.4: Floor plan for LDL bridge with GPIO IP core

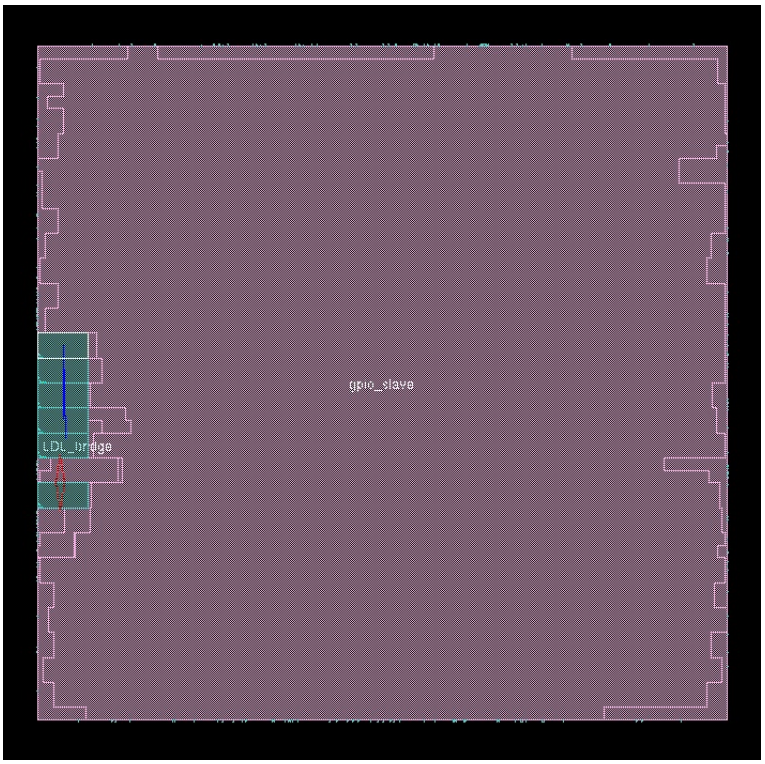


Figure 9.5: Ameobioic view classifies LDL bridge and GPIO IP core

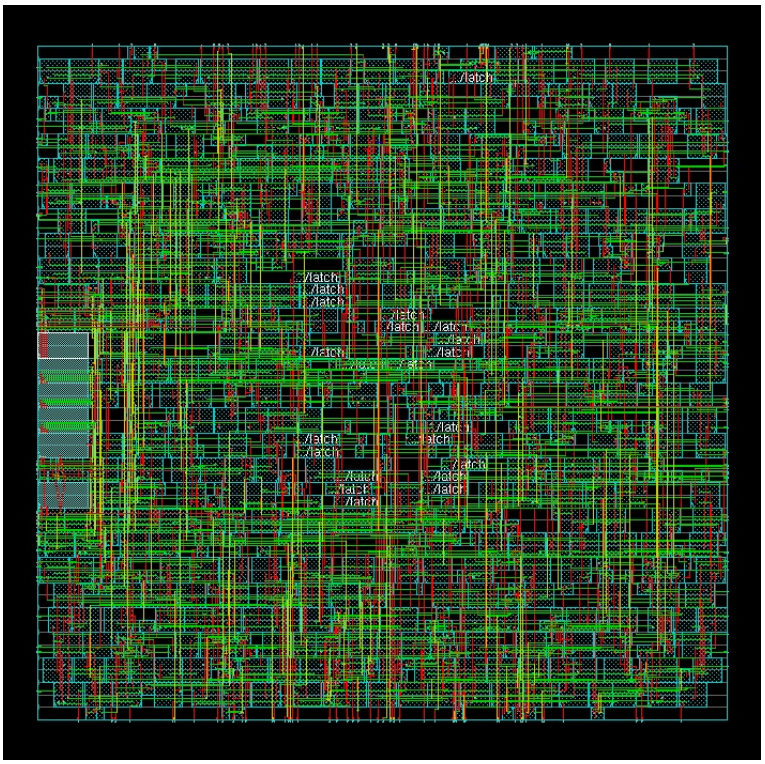


Figure 9.6: Layout of LDL bridge with GPIO IP core

Chapter 10

Results

This chapter is dedicated to describe and interpret overall results obtained after implementation of the synchronizers. Equations are formulated according to the structure of the implementation to calculate the delay expenditure that would occur during transfer of data between multi-clock domain systems.

We estimate a formula for the delay expense that would occur during transfer of data between multi-clock domain systems for each of the synchronizer. Next, the overall time taken for Wishbone single read/write cycle is estimated. Later the complete theoretical formulas are compared with the simulation results and are described in the following sections.

10.1 Two flip-flop synchronizer

According to figure 10.1, single read/write cycle would experience a delay that has affected the transition of STB_O and ACK_I as a point of Wishbone master perception.

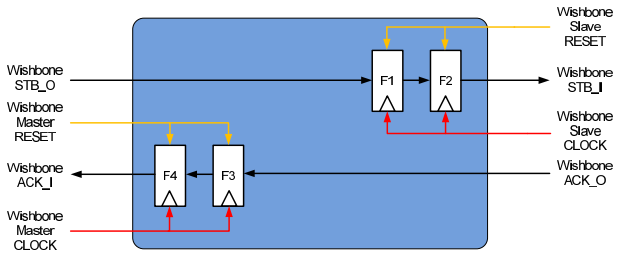


Figure 10.1: Two flip flop 4-phase synchronizer applied to Wishbone bus

- Delay introduced by two flip-flops while transferring the signal STB_O from master to slave. Typically the expense would be two clock cycles of the Wishbone slave. We denote this delay by "DFF_{MS}"
- Delay that would incur in Wishbone slave. This delay is the time taken to detect the Wishbone strobe, process the request and finally acknowledge by asserting

ACK_O. We denote this delay by "Wb_slave1_{delay}".

- Delay introduced by two flip-flops while transferring the signal ACK_I from slave to master. Typically the expense would be two clock cycles of the Wishbone master. We denote this delay by "DFF_{SM}".
- Delay introduced by the Wishbone master is the time taken to detect the ACK_I signal and de-assert the STB_O . We denote this delay by "$Wb\_master1_{delay}$".
- Delay introduced by the Wishbone slave is the time taken to detect the STB_O signal(de-asserted) and de-assert the ACK_O . We denote this delay by "Wb_slave2_{delay}".
- Delay introduced by the Wishbone master is the time taken to detect the ACK_I signal(de-asserted) after de-assertion of ACK_O by slave. We denote this delay by "$Wb\_master2_{delay}$".

Considering all of the above factors, the estimation of the time to complete Wishbone read/write cycle would be,

$$T_{2FFcycle} = DFF_{MS} + Wb\_slave1_{delay} + DFF_{SM} + Wb\_master1_{delay} + Wb\_slave2_{delay} + Wb\_master2_{delay} \tag{10.1}$$

Expressing equation 10.1 in terms of number of clock expense would result to,

$$T_{2FFcycle} = n \times T_{Wb_{slave\_clock}} + n \times T_{Wb_{slave\_clock}} + m \times T_{Wb_{master\_clock}} + m \times T_{Wb_{master\_clock}} + n \times T_{Wb_{slave\_clock}} + m \times T_{Wb_{master\_clock}} \tag{10.2}$$

The delay introduced by the two flip-flop synchronizer bridge is given by,

$$T_{2FFsynchronizer} = DFF_{MS} + DFF_{SM} \tag{10.3}$$

Expressing equation 10.3 in terms of number of clock expense would result to,

$$T_{2FFsynchronizer} = n \times T_{Wb_{slave\_clock}} + m \times T_{Wb_{master\_clock}} \tag{10.4}$$

10.1.1 Case study

In this section we verify the formula 10.1 by calculating theoretically and the value obtained from the simulation result. The clock configurations used are depicted in table 10.1.

Clock Domain	Input Clock	Time period (ns)
Wishbone master	10 MHz	100
Wishbone slave	32 kHz	31250

Table 10.1: Clock configuration

This clock configuration matches with the real use case where the fast CPU (of clock 10 MHz) will try to access a real time clock peripheral (of clock 32 kHz) that keeps track of time and date.

According to simulation, the total time taken to complete single read/write cycle is T_{2FFsim} = 375500 ns.

Table 10.2 gives the data estimation of the delay expenditure that would occur for single read/write cycle. The delay in Wishbone master and slave cannot be estimated and hence those delays are fetched from simulation.

Parameter	Equivalent clock expense	Clock expense(ns)	Estimated values(ns)	Clock cycles
DFF_{MS}	n $\times T_{Wb_{slave\_clock}}$	2 ×31250	62500	2 SCC
Wb_slave1_{delay}	n $\times T_{Wb_{slave\_clock}}$	5 ×31250	156250	5 SCC
DFF_{SM}	m $\times T_{Wb_{master\_clock}}$	2 ×100	200	2 MCC
$Wb\_master1_{delay}$	m $\times T_{Wb_{master\_clock}}$	1 ×100	100	1 MCC
Wb_slave2_{delay}	n $\times T_{Wb_{slave\_clock}}$	5 ×31250	156250	5 SCC
$Wb\_master2_{delay}$	m $\times T_{Wb_{master\_clock}}$	2 ×100	200	2 MCC
Total	$\sum$	375500	375500	5 MCC + 12 SCC

Table 10.2: Delay estimate

All the values in table 10.2 are plugged in the equation 10.1. The total delay obtained is $T_{2FFcycle}$ = 375500 ns which is same as T_{2FFsim} as tabulated in table 10.3. Therefore the estimated equation 10.1 is valid.

Parameter	Time for single read/write cycle (ns)
$T_{2FFcycle}$	375500
T_{2FFsim}	375500

Table 10.3: Comparison between theoretical and simulation data

The delay introduced by the two flip-flop synchronizer bridge is calculated using equation 10.3 and the value obtained after calculation is tabulated in table 10.4,

We observe that the total delay induced by the two flip-flop synchronizer bridge is approximately equal to 2 clock cycles of the slowest clock i.e. Wishbone slave clock since $T_{Wb_{slave\_clock}} \gg T_{Wb_{slave\_clock}}$. Thus for any clock configuration, the delay that would occur is almost equivalent to 2 clock cycles of the slowest clock in the multi-clock domain. This delay is unavoidable as per the synchronizer architecture hence it is one of the major drawback of the two flip-flop synchronizer.

Parameter	Delay (ns)	Clock cycles
DFF_{MS}	62500	2 SCC
DFF_{MS}	200	2 MCC
$T_{2FFsynchronizer}$	62700	$\approx$ 2 SCC

Table 10.4: Two flip-flop synchronizer bridge delay

10.2 LDL synchronizer

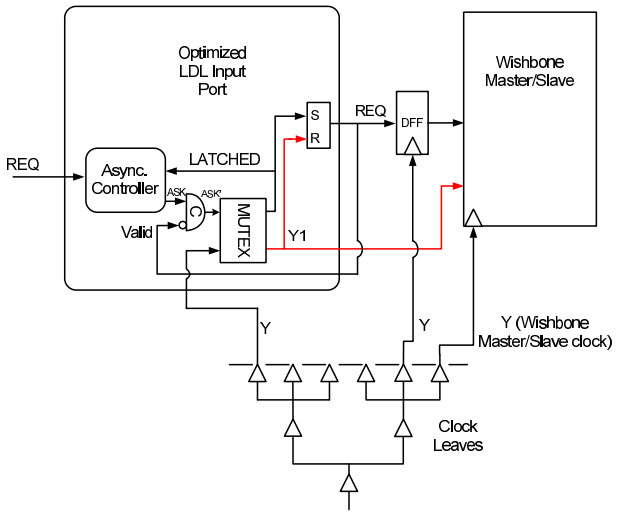


Figure 10.2: Optimized LDL input port used for Wishbone bridge

The delay estimation of optimized LDL input port shown in figure 10.2 needs the consideration of the following factors:

- The asynchronous controller delay introduces "D_{CTRL}" during processing of the input signal REQ . This delay includes all the logic present in the controller except the metastability resolution time of MUTEX refer figure 6.2.
- The metastability resolution time "$T_{m/s}^{MUTEX}$" is also introduced by the MUTEX during the conflict of REQ+ and Y+ . This delay is random and has no fixed amount of time, thus the MUTEX element grants the output only after a random time "$T_{m/s}^{MUTEX}$" during metastability.
- The delay introduced by a flip-flop "DFF" to capture the output of the S-R latch as shown in figure 7.2. We denote this delay by "DFF_LDL_{delay}".

Above two factors are to be considered for only one input port but for the complete LDL bridge shown in figure 7.3 and 7.4. It requires the same input port bridge to process

Wishbone ACK_I. Thus the delay caused by above factors has to considered for processing Wishbone STB_O and ACK_I. It is also required to consider the delay introduced by the Wishbone bus as given below,

- Delay that would occur in Wishbone slave. This delay is the time taken to detect the Wishbone strobe, process the request and finally acknowledge by asserting ACK_O. We denote this delay by "Wb_slave1_{delay}".
- Delay introduced by the Wishbone master is the time taken to detect the ACK_I signal and de-assert the STB_O. We denote this delay by "$Wb\_master1_{delay}$".

Considering all above factors the total time taken to complete Wishbone single read/write cycle would be:

$$T_{LDLcycle} = D_{CTRL1} + T_{m/s}^{MUTEX1} + DFF\_LDL_{delay1} + Wb\_slave1_{delay} + D_{CTRL2} + T_{m/s}^{MUTEX2} + DFF\_LDL_{delay2} + Wb\_master1_{delay} \tag{10.5}$$

Expressing in terms of clock expense would result in:

$$T_{LDLcycle} = D_{CTRL1} + T_{m/s}^{MUTEX1} + n \times T_{Wb_{slave\_clock}} + n \times T_{Wb_{slave\_clock}} + D_{CTRL2} + T_{m/s}^{MUTEX2} + m \times T_{Wb_{master\_clock}} + m \times T_{Wb_{master\_clock}} \tag{10.6}$$

The total delay induced by the LDL synchronizer bridge is given by equation 10.7,

$$T_{LDLsynchronizer} = D_{CTRL1} + T_{m/s}^{MUTEX1} + DFF\_LDL_{delay1} + D_{CTRL2} + T_{m/s}^{MUTEX2} + DFF\_LDL_{delay2} \tag{10.7}$$

Expressing in terms of clock expense would result in:

$$T_{LDLsynchronizer} = D_{CTRL1} + T_{m/s}^{MUTEX1} + n \times T_{Wb_{slave\_clock}} + D_{CTRL2} + T_{m/s}^{MUTEX2} + m \times T_{Wb_{master\_clock}} \tag{10.8}$$

10.2.1 Case study

In this section we note the various delay expense from the simulation and use the formula 10.5 to calculate the total delay expense for Wishbone single read/write cycle. The clock configurations used are tabulated in table 10.1. Most of the parameters used in equation 10.5 are unpredictable because of random behaviour of MUTEX element present in LDL synchronizer bridge. Also the chances of conflict occurrence at the input of MUTEX is also random and is very less. However the minimum time delay introduced by the MUTEX is the delay occurred only when there is no conflict and no metastability can occur and is estimated to be equivalent to SR-latch delay. The maximum delay

introduced by the MUTEX is random and depends upon the probability of metastability occurrence.

The occurrence of conflict between STB_O and Wishbone slave clock, ACK_O and Wishbone master clock is unpredictable. For the chosen clock configuration, it was possible to simulate the conflict between STB_O and Wishbone slave clock only and is considered to be typical but not generic. For this typical case, the following data are measured.

According to simulation, the total time taken to complete single read/write cycle is T_{LDLsim} = 187703.79 ns.

We use the unpredictable values from the simulation data and the predictable values such as delay in flip-flops are theoretically assumed.

Table 10.5 gives the data of the delay expenditure that would occur for single read/write cycle.

Parameter	Equivalent clock expense	Clock expense(ns)	Simulated values(ns)	Clock cycles
D_{CTRL1}	-	-	0.37	-
$T_{m/s}^{MUTEX1}$	-	-	2	-
DFF_LDL_{delay1}	n $\times T_{Wb_{slave\_clock}}$	1 $\times$31250	31250	1 SCC
Wb_slave1_{delay}	n $\times T_{Wb_{slave\_clock}}$	5 $\times$31250	156250	5 SCC
D_{CTRL2}	-	-	0.42	-
$T_{m/s}^{MUTEX2}$	-	-	1	-
DFF_LDL_{delay2}	m $\times T_{Wb_{master\_clock}}$	1 $\times$100	100	1 MCC
$Wb\_master1_{delay}$	m $\times T_{Wb_{master\_clock}}$	1 $\times$100	100	1 MCC
Total	$\sum$	187703.79	187703.79	$\approx$ 2 MCC + 6 SCC

Table 10.5: Simulation data with LDL synchronizer bridge

Since there was no conflict between Wishbone ACK_O and the master clock, the delay in MUTEX element of the LDL input port receiving ACK_O is minimum. Considering all the data tabulated in table 10.5, the total time consumed to complete one read/write cycle is $T_{LDLcycle}$ = 187703.79 ns. Again the estimated theoretical value and the simulation value are same. Both values are tabulated in table 10.6.

The total delay induced by the LDL synchronizer bridge is tabulated in table 10.7.

Parameter	Time for single read/write cycle (ns)
T_{LDLsim}	187703.79
$T_{LDLcycle}$	187703.79

Table 10.6: Comparison between theoretical and simulation data

Parameter	Delay (ns)	Clock cycles
D_{CTRL1}	0.37	-
$T_{m/s}^{MUTEX1}$	2	-
DFF_LDL_{delay1}	31250	1 SCC
D_{CTRL2}	0.42	-
$T_{m/s}^{MUTEX2}$	2	-
DFF_LDL_{delay2}	100	1 MCC
$T_{LDLsynchronizer}$	31354.79	$\approx$ 1 SCC

Table 10.7: LDL synchronizer bridge delay

10.3 Performance comparison between two flip-flop and LDL synchronizer bridges

In case of two flip-flop synchronizer, two clock cycles of each clock domain are reserved for metastability resolution. Hence the total time expense for single read/write cycle would always experience the delay of minimum two clock cycles from each clock domain. This would ensure that proper handling of metastable signals and it would never allow the metastable signals to escape from bridge to the locally synchronous islands such as Wishbone master/slave except once in MTBF. However there is no assurance that the conflict can happen every time during transfer of STB_O and ACK_O; giving a scope to avoid the unnecessary delay when the conflict is not imminent.

In case of LDL synchronizer, the data transferred from one clock domain to another clock domain would experience a delay induced by MUTEX element, if and only if the conflict between the data and the system clock is imminent. Hence the bottom line is basically to avoid unnecessary delay every time when there is no conflict. Also the conflict occurrence is comparatively less for the clock configuration (table 10.1) used. Suppose if the conflict occurred, maximum delay expense would be one clock cycle followed by the metastability resolution time caused by MUTEX. In the case study of LDL synchronizer, the total delay induced by the LDL bridge is less than the delay induced by the two flip-flop synchronizer bridge.

Finally comparing the synchronizer bridge delays for a typical case study the values and the difference is shown in table 10.8.

Synchronizer bridge type	Parameter	Delay (ns)
Two flip-flop synchronizer	$T_{2FFsynchronizer}$	62700
LDL synchronizer	$T_{LDLsynchronizer}$	31354.79
Difference	T_{δ}	31345.21

Table 10.8: Bridge delay comparison for a typical case

It can be noted that LDL synchronizer bridge is almost twice as fast as two flip-flop bridge and this comparison applies to only a typical case given in case studies of both synchronizers.

Chapter 11

Conclusion and Future work

The following conclusions have been drawn from the thesis work.

- Signals which are closely switching at the switching levels of the clock are prone to metastability. The common scenario where the metastability can occur is data transfer between multiple clock domains. They need proper synchronization methods to avoid the sampling of a signal with metastable state otherwise this may lead to malfunction of electronic systems.

- We investigated the metastability occurrence that may happen within Wishbone bus with different clock domain of Wishbone master and slave. We observe that Wishbone signals such as STB_O and ACK_O follows the 4-phase bundled data protocol. This create a scope to implement an asynchronous bridge that will provide synchronization to transfer the data between multi-clock domain.

- Two flip-flop synchronizer and LDL synchronizer were designed and they were implemented with exploiting the scope of 4-phase bundled protocol behaviour in Wishbone bus. The inherent property of Wishbone such as, all the signals(including data) has to be stable before read/write cycle is initiated, will give an opportunity to optimize the synchronizer architectures greatly that will contribute to the hardware area and power consumption.

- These synchronizers can be used as a bridge between Wishbone bus where the Wishbone master and slave are driven by different clock-domains. Frequently switching signals such as STB_O and ACK_O which are more prone to metastability are synchronized and are treated properly to resolve the metastability that will avoid the propagation of metastable values into Wishbone master/slave. The verification of implemented bridges is performed using GPIO IP core.

- Two flip-flop synchronizer bridge will always induce a delay of two clock cycles of each clock-domains irrespective to the metastable conditions. This is gives rise to a MTBF-safe architecture at the cost of two clock cycles and hence the total latency caused by two flip-flop synchronizer bridge is higher than LDL synchronizer bridge.

- The two flip-flop synchronizer can fail once in MTBF and has high potential to inject metastable value into the system once in MTBF. This may cause system malfunction.

- The LDL synchronizer bridge uses asynchronous circuit elements such as Muller-C element and MUTEX element to resolve the metastability. This synchronizer will not induce delay until the metastability is imminent. Therefore the total latency induced by the LDL synchronizer bridge is on average lesser than two flip-flop bridge.

- The LDL synchronizer can fail due to induction of random delay of MUTEX element. However the failure mode is different than that of two flip-flop synchronizer, the LDL bridge would never inject a metastable value into the system but it may take too long time to resolve the metastability during the conflict. Thus in real time systems the use of this kind of synchronizer can lead to malfunction of the system.

- The metastability can also occur in the systems which uses asynchronous reset. A reset synchronizer is discussed in Appendix A.

Future work

- **Design of MUTEX circuit**: We used a behavioural model of the MUTEX element in this project. This was a constraint in deciding the exact area and power consumption of LDL synchronizer bridge. It is required to develop a typical MUTEX element in the analog environment where the required transistor sizing has to be done for the metastability filter.

  Once the design of MUTEX circuit is done, it is required to analyse the feasible maximum metastability resolution time it may take during the occurrence of conflict between its two input signals. After which the parameter "τ" can be evaluated which will help to calculate accurate MTBF of the LDL synchronizer. Also the exact latency induced by the same synchronizer can be evaluated for different cases of conflict.

- **Design of Muller-C element**: The Muller-C element used in this project is also a behavioural model. Hence it causes constraints in implementation area and power consumption evaluation. It is required to design a typical Muller-C element in analog environment and can be analysed with its various parameters.
- **Removal of a flip-flop in LDL synchronizer bridge**: There exist one flip flop inside the in the locally synchronous island such as Wishbone master/slave to hold Wishbone STB_O and ACK_O after metastability resolution. Extra logic can be added in the LDL synchronizer to create a required delay in STB_O and ACK_O signals that would suffice to sample the same by Wishbone master/slave at the cost of extra implementation area.
- **Measurement of metastability in two flip-flop and LDL synchronizers**: A deep measurement for the metastability in digital circuits for 180nm technology can be done by implementing suitable metastability measurement circuits. Those circuits are discussed in [21]. Also the performance effect of the synchronizers with metastability can be evaluated by special circuits discussed in [22].

Appendix A

Reset Synchronizer

In this section we discuss the application and importance of reset synchronizer used in this project. As per Wishbone specification, the system reset used should be active high. However use of synchronous or asynchronous reset is upto designer's decision. We used asynchronous reset for in the synchronizer bridges.

The system reset is to be driven by the reset synchronizer to avoid metastability that may occur due to change of levels in system reset close to the switching levels of the clock. It is common that reset is prone to metastability during its transition from high to low asynchronously and clock is about to take its rising edge at the same instance of time. To avoid the reset metastability problem we use reset synchronizer.

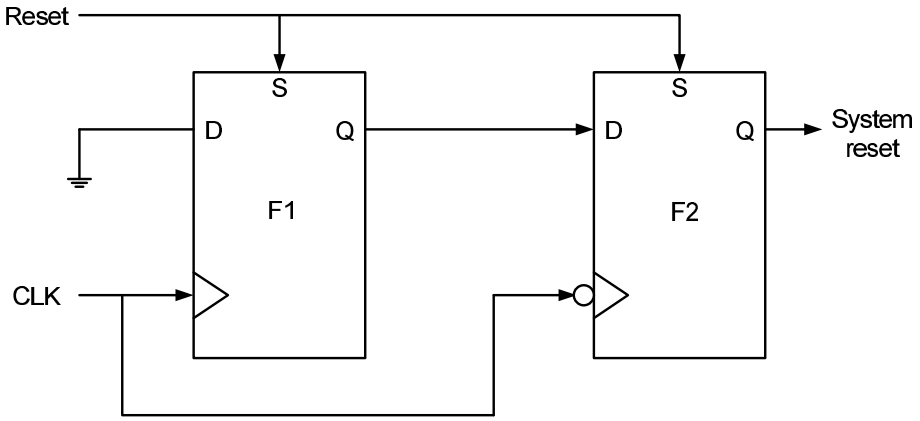


Figure A.1: Reset synchronizer circuit [23]

The reset synchronizer circuit is depicted in figure A.1, it consists of two data flip-flops with asynchronous set, connected back to back. They both have same clock but the flip-flop "F2" has a clock inverted w.r.t "F1". The asynchronous reset is driven by the asynchronous reset signal.

The flip-flop "F1" input is always connected to the ground and hence "F1" always gives the output as logic-0 for each clock cycle until asynchronous reset is asserted,

the same situation occurs in “F2” hence the system reset signal is always zero except asynchronous reset is asserted.

Once asynchronous reset signal is asserted, logic-1 is propagated to the input of “F2” at the rising edge of clock of “F1”. The system reset signal is asserted at the next falling edge of the clock by “F2”. However at the de-assertion of the reset signal, the synchronizer takes care of switching the system reset signal from high to low only at the falling edge of the clock. This is illustrated in figure A.2

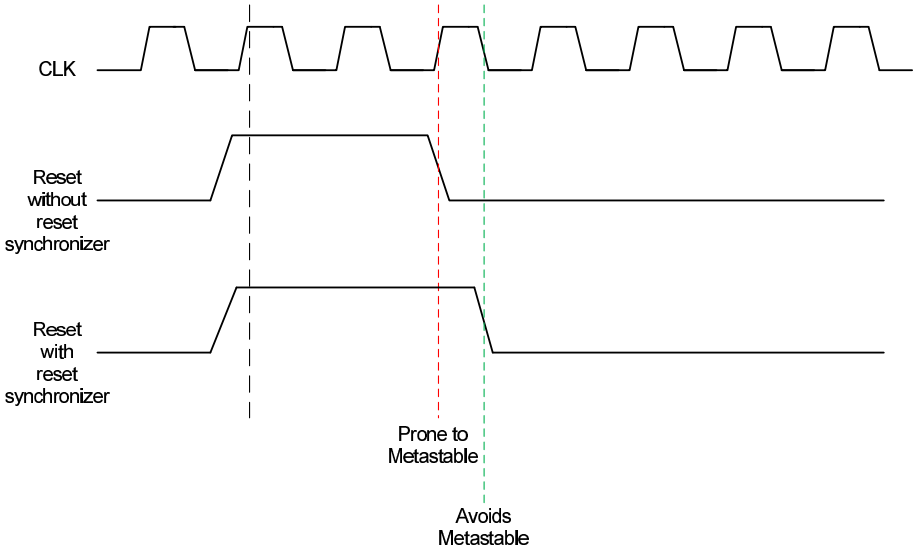


Figure A.2: Reset signals with and without reset synchronizer

Thus the switching action from high to low of reset signal is restricted to the falling edge of the clock only. This makes the reset signal to escape from metastable condition.

Bibliography

[1] R. Dobkin, R. Ginosar, and C. Sotiriou, "High rate data synchronization in GALS SoCs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 10, pp. 1063 –1074, oct. 2006.

[2] A. Swain and K. Mahapatra, "Design and verification of Wishbone bus interface for system-on-chip integration," in *India Conference (INDICON), 2010 Annual IEEE*, dec. 2010, pp. 1 –4.

[3] *"WISHBONE System-on-Chip Interconnection Architecture for Portable IP Cores"*. OpenCores, 2010.

[4] D. K. Mohandeep Sharma, "Wishbone bus architecture: A survey and comparison," *International Journal of VLSI design Communication Systems (VLSICS) Vol.3, No.2*, vol. 3, no. 2, pp. 110 –111, april. 2012.

[5] R. Ginosar, "Metastability and synchronizers: A tutorial," *Design Test of Computers, IEEE*, vol. 28, no. 5, pp. 23 –35, sept.-oct. 2011.

[6] L. Davis. (2012, May) "Digital Logic Metastability". [Online]. Available: http://www.interfacebus.com/Design_MetaStable.html

[7] J. Sparso and S. F. (eds), *"Principles of asynchronous circuit design - A systems perspective"*. Kluwer Academic Publishers, 2001.

[8] R. R. Dobkin and R. Ginosar, "Integrated circuit and system design. power and timing modeling, optimization and simulation," pp. 199–208, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-95948-9_20

[9] R. G. Rostislav Dobkin, *"Zero latency synchronizers using four and two phase protocol"*, 2007.

[10] G. Semeraro, D. Albonesi, S. Dropsho, G. Magklis, S. Dwarkadas, and M. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, 2002, pp. 356 – 367.

[11] W. Daasch, C. Lim, and G. Cai, "Design of vlsi cmos circuits under thermal constraint," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 49, no. 8, pp. 589 – 593, aug 2002.

[12] L. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 391 –397, dec. 1994.

[13] R. Dobkin, R. Ginosar, and C. Sotiriou, "Data synchronization issues in GALS SoCs," in *Asynchronous Circuits and Systems, 2004. Proceedings. 10th International Symposium on*, april 2004, pp. 170 – 179.

[14] R. Ginosar. (2012, May) "MTBF of a multi-synchronizer system on chip". [Online]. Available: http://www.ee.technion.ac.il/~ran/papers/MTBFmultiSyncSoc.pdf

[15] (2012, May) "International Technology Roadmap for Semiconductors (ITRS)". [Online]. Available: www.itrs.net

[16] C. Dike and E. Burton, "Miller and noise effects in a synchronizing flip-flop," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 6, pp. 849 –855, jun 1999.

[17] G. D. Damjan Lampret, *"GPIO IP core specification"*. OpenCores, 2003.

[18] (2008, March) "VCS/VCSi user guide". [Online]. Available: http://users.ece.utexas.edu/~patt/10s.382N/handouts/vcs.pdf

[19] *"Design Compiler® User Guide"*. Synopsys®, June 2010.

[20] N. H. E. Weste and D. Harris, *"CMOS VLSI design : a circuits and systems perspective"*, 3rd ed. Boston: Pearson/Addison-Wesley, 2005.

[21] J. Zhou, D. Kinniment, C. Dike, G. Russell, and A. Yakovlev, "On-chip measurement of deep metastability in synchronizers," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 2, pp. 550 –557, feb. 2008.

[22] D. Kinniment, C. Dike, K. Heron, G. Russell, and A. Yakovlev, "Measuring deep metastability and its effect on synchronizer performance," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1028 –1039, sept. 2007.

[23] D. M. Clifford E. Cummings. (2002) "synchronous resets? asynchronous resets? i am so confused! how will i ever know which to use?". [Online]. Available: http://www.engsoc.org/~jvd/docs/hdl/CummingsSNUG2002SJ_Resets.pdf