T.C.
ISTANBUL UNIVERSITY-CERRAHPASA
INSTITUTE OF GRADUATE STUDIES

M.Sc. THESIS

A HIGH-SPEED MIXED SIGNAL EMBEDDED SYSTEM DESIGN FOR SAW BASED BIOSENSORS

Selma UÇAR AKDENİZ

SUPERVISOR
Assist. Prof. Dr. Koray GÜRKAN

Department of Electrical and Electronic Engineering

Electrical and Electronic Engineering Programme

ISTANBUL- June, 2019

This study was accepted on 24/6/2019 as a M. Sc. thesis in Department of Electrical and Electronic Engineering, Electrical and Electronic Engineering Programme by the following Committee.

Examining Committee Members

[illegible]

Assist. Prof. Dr. Koray GÜRKAN(Supervisor)
İstanbul University-Cerrahpaşa
Faculty of Engineering

[illegible]

Prof. Dr. Fırat Kaçar
İstanbul University-Cerrahpaşa
Faculty of Engineering

[illegible]

Assist. Prof. Dr. Sinan AKŞİMŞEK
İstanbul Kültür University
Faculty of Engineering

FOREWORD

This thesis is dedicated to all the people who have helped me throughout the research. I would like to thank İstanbul University-Cerrahpaşa for facilities provided and opportunities.

I express sincere appreciation to my supervisor Assist. Prof. Dr. Koray GÜRKAN for his guidance throughout the research project. His invaluable feedback and support helped me to structure and improve this thesis.

I would also like to thank to TÜBİTAK BIDEB 2210-A National Scholarship Program for MsC Students for their financial support during my master study.

I am thankful to my family Aydın, Raziye, Hasan Ali and Tarık UÇAR for their encouragement and patience. Special thanks go to my husband Mustafa AKDENİZ for his motivation, understanding and support.

I would like to express my gratitude towards Erhan SARIOĞLU for his contribution. Lastly, I am thankful to all my friends who made my stay at the university a memorable and valuable experience.

June 2019 Selma UÇAR AKDENİZ

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol	Explanation
Tclk	: clock period
Fclk	: clock frequency
Fout	: output frequency
Fin	: input frequency
Msps	: mega sample per second
Mbps	: mega bit per second
MHz	: mega hertz
Tcq	: clock to q time
Tnext	: change time of next register
Tsetup	: setup time of the flip-flop
Tskew	: skew time
Tr	: resolution time
Fr	: resolution frequency
FF	: Flip-Flop
Tcomb	: propagation delay of the combinatorial circuit
Rmeta	: the average rate of the flip-flop goes into metastable state
P(Tr)	: the probability of flip-flop cannot solve metastability in given time
AF	: average number of synchronization failure
&	: the phase increment value
$	: the decay time constant
B	: the number of bits employed in the phase accumulator
V	: Volt
bps	: bit per second
kB	: kilo Byte
w	: the period
td	: critical time window length

Abbreviation	**Explanation**
ADC	: Analog to Digital Converter
ARM	: Advanced RISC Machine
ASIC	: Application-Specific Integrated Circuit
CPU	: Central Processing Unit
CRC	: Cyclic Redundancy Code
DCM	: Digital Clock Management
DDS	: Direct Digital Synthesizer
FIFO	: First-in First-out Buffer
FPGA	: Field Programmable Gate Array
FSM	: Finite State Machine
GALS	: Globally Asynchronous Locally Synchronous
IDT	: Interdigital Transducers
IP	: Intellectual Property
JTAG	: Joint Test Action Group
LIFO	: Last-in First-out Buffer
LSB	: Least Significant Bit
MEMS	: Microelectromechanical Systems
MSB	: Most Significant Bit
MTBF	: Mean Time Between Failure
PC	: Personel Computer
RAM	: Random Access Memory
SAW	: Surface Acoustic Wave
SRAM	: Static Random Access Memory
UART	: Universal Asynchronous Receiver Transmitter
USB	: Universal Serial Bus
VLSI	: Very Large-Scale Integration

ÖZET

SAW BİYOSENSÖRLER İÇİN YÜKSEK HIZLI ÇOKLU SİNYAL GÖMÜLÜ SİSTEM TASARIMI

YÜKSEK LİSANS TEZİ

Selma UÇAR AKDENİZ

İstanbul Üniversitesi-Cerrahpasa

Lisansüstü Eğitim Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman : Dr. Öğr. Üyesi Koray GÜRKAN

Teknolojinin gelişmesiyle elektronik sistemlerde tek bir merkezi birim bulunması yerine birçok bağımsız birimden oluşmaktadır ve bu birimlerden bazıları daha yüksek saat hızlarına (clock speed) ihtiyaç duymaktadır. Birçok sistemde, yüksek hızlı birimlerle düşük hızlı birimler arasında veri aktarımı gerekmektedir. Bu veri aktarımını doğru şekilde gerçekleştirebilmenin en iyi yolu, farklı hızdaki birimler arasına çift saatli arabellek (dual-clock buffer) kurmaktır. En yaygın ve kullanışlı arabellek türü FIFO (first in – first out, ilk giren ilk çıkar) arabellektir. Bu tezin amacı FPGA (Field programmable gate array, Sahada programlanabilir kapı dizileri) üzerinde farklı saat hızlarında çalışan birimlerin haberleşmesi için bir çift saatli FIFO arabellek kurmak ve bu arabelleğin kontrolü için bir paket protokolü tasarlamaktır. Öncelikle FIFO mimarileri araştırılıp, FPGA üzerinde FIFO arabellek kuruldu. Farklı saat hızlarından dolayı senkronizasyon hataları oluşabileceğinden, FIFO arabelleğin giriş sinyalleri iki D flip-flop senkronizör tarafından senkronize edildi. Sistemin kapsamlı kontrolünü sağlamak için güçlü bir haberleşme paket protokolü oluşturuldu. Sistemin giriş çıkış sinyalleri incelenerek yapılan çalışma doğrulandı.

Haziran 2019, 81 sayfa.

Anahtar kelimeler: FPGA, FIFO, UART, Metastabilite, Saat Domeni Geçişi

SUMMARY

A HIGH-SPEED MIXED SIGNAL EMBEDDED SYSTEM DESIGN FOR SAW BASED BIOSENSORS

M.Sc. THESIS

Selma UÇAR AKDENİZ

Istanbul University-Cerrahpasa

Institute of Graduate Studies

Department of Electrical and Electronic Engineering

Supervisor : Assist. Prof. Dr. Koray GÜRKAN

With the development of technology, electronic systems consist of many independent units instead of single central unit and some of these units need higher clock speeds. In many systems, data transfer between high-speed units and low-speed units is required. The best way to implement this data transfer correctly is to construct a dual-clock buffer between units at different speeds. The most common and useful buffer type is FIFO buffer (first in first out). The purpose of this thesis is to build FPGA (Field programmable gate array) based a dual-clock FIFO buffer for communicating units operating at different clock speeds and to design a packet protocol for controlling this buffer. First of all, the FIFO architectures were searched, and a FIFO buffer was constructed on the FPGA. Since synchronization errors can occur due to different clock speeds, input signals of the FIFO buffer were synchronized by the two D flip-flop synchronizers. A strong communication packet protocol was designed to provide comprehensive control of the system. The input and output signals of the system were analyzed, and the study was verified.

June 2019, 81 pages.

Keywords: FPGA, FIFO, UART, Metastability, Clock Domain Crossing

1. INTRODUCTION

Surface acoustic wave sensors have demonstrated useful in several fields as mostly mass sensitive devices capable of replying to small environmental disturbances. Acoustic wave devices have praiseworthy qualities that make them valuable in sensoring systems including for medical diagnostics. These sensors can be configured for a wide range of applications and are highly mobile with high sensitivity.

Surface acoustic wave sensors are a type of microelectromechanical systems (MEMS) which depend on the modulation of surface acoustic waves to sense ultra-low concentrations of certain biochemical. The sensor converts an input electrical signal into a mechanical signal which, dissimilar to an electrical signal, can be easily affected by physical phenomena. This interaction can be in the form of delay or phase shift, attenuation, multiple reflections, or more complex nonlinear effects. A SAW biosensor also converts this transduced mechanical signal back to an electrical signal, which can be digitized and processed.

The main SAW device as shown in Figure 1.1 consists of a delay path with the piezoelectric substrate for the acoustic wave to spread, and two IDTs for input and output transduction. This region is named the delay line because of the signal, which is a mechanical signal at this point, propagates much slower than its electromagnetic form, thus causing a detectable effect amplitude and phase.

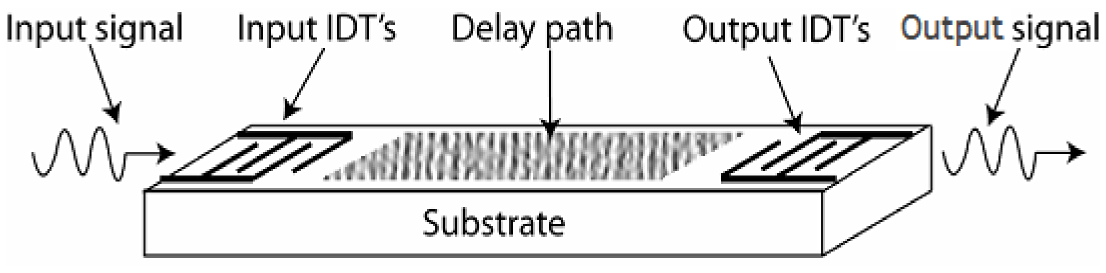


Figure 1.1: SAW sensor design including input/output signals and IDTs.

Embedded systems are designed for a specific purpose and perform this purpose as fast as possible with low power consumption. Designing and testing an embedded system has become much easier with the result of inventing FPGAs. It was invented by Ross Freeman who is one of the founders of the Xilinx Company in 1984. Because of flexible and rapid prototyping capability of FPGA, to work on it has many advantages as an integrated circuit (IC) platform for verification and prototyping [1].

The Field Programmable Gate Array consists of three main parts, configurable logic blocks in the matrix structure, input-output blocks surrounding this block array and interconnections between these block arrays. The configuration of the programmable logic blocks and the communication between these blocks takes place through the interconnections. Interconnections of logic cells take place with matrix-shaped data paths and programmable switches (according to the program loaded into the FPGA). The FPGA design determines the function that each logic cell will perform and the state of the programmable switches (on / off). The input and output blocks perform the connection between the interconnections and the pins of the integrated circuit. FPGAs can be easily programmed for any application, the same FPGA reprogrammed for another application. The architecture of an FPGA is theoretically given in Figure 1.2.

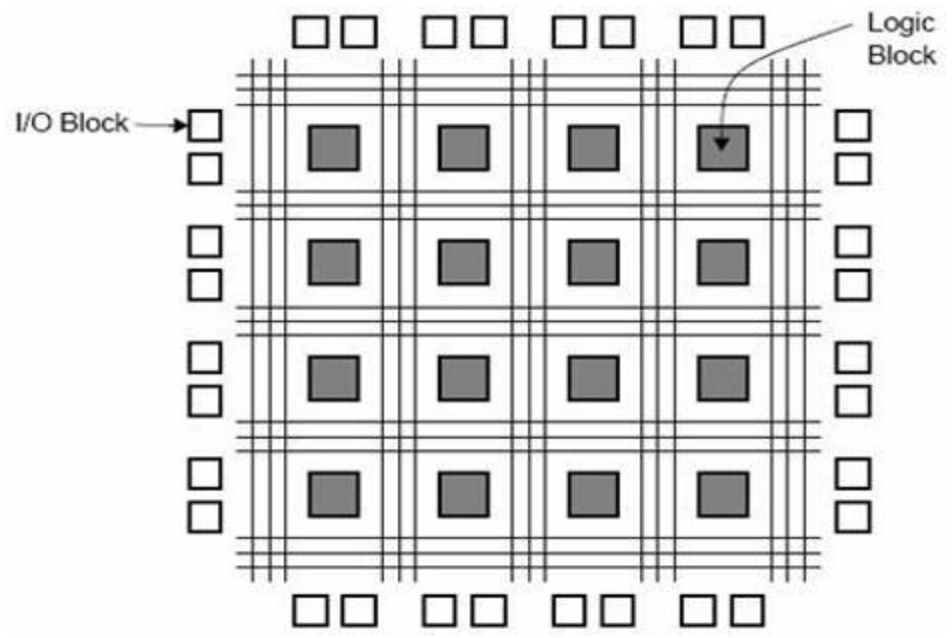


Figure 1.2: The architecture of an FPGA.

Today, as the processing capability expectation of embedded systems is increasing, many systems are passing to FPGAs, becoming more important in the designing of a digital circuit, instead of embedded CPUs which process functions conventionally [2]. To use FPGA makes system flexible and reliable and reduces power consumption. FPGAs can be utilized not only to implement simple circuit but also to build complex state machines to satisfy different system needs [3].

FPGAs have become very popular with the industry thanks to their flexible architecture and reprogramming features and are now preferred in many application areas that especially demand flexibility with high performance such as automotive, telecommunication, defense industry, and space applications [4]. Xilinx and Altera are the two largest companies in the world. These companies produce a variety of FPGA families for different needs.

Because of increasing both size and complexity of systems, timing requirements of VLSI systems are becoming complex. In order to satisfy these strict timing requirements, traditional digital circuit design methods must be improved. The best way to cope with this stringent timing requirements is to design an asynchronous circuit [5]. However, the risk and cost of moving away from the design approach that has been successful in the past, prevents a major transition to this advanced design approach. For this reason, even present-day synchronous design approach is predominant method of implementation in digital electronics design [6].

The global clock must supply to all areas of the circuit at almost the same time to be able to construct accurate system. Clock frequencies have been increasing and transistor sizes have been shrinking. Because of these improvements new systems are working at high speed and are becoming more complex. The most difficult problem of digital system design is global clock signal distribution circuit which needs a lot of time and effort [7]. As integrated circuit technologies are improving, the circuit scale is increasing rapidly and the system clock is being needed more often. The system may behave like an asynchronous system because the size increase of circuit might lead to clock skew [8]. Due to the clock skew of a large circuit, the best approach to constitute a reliable system is to divide the system to subsystems. However, sometimes to determine the relationship between the clock signals of subsystems is not possible. These subsystems are considered as independent clock domains.

Additionally, multiple clocks may be essential in some cases. For instance, digital systems often need to interact with external systems like peripheral devices. These external systems may be working at different clock frequencies. Since the interaction between the system and the external system includes two different clocks, this system works asynchronously [9].

To create systems with synchronous and asynchronous design method together is an alternative technique. This technique is called as a Globally Asynchronous Locally Synchronous (GALS) system. In this approach, local blocks are based on synchronous design techniques, but these synchronous blocks do not share global clock signal and are asynchronous to each other. Advance timing and synchronization techniques improve the result of traditional design techniques [10].

The difficulty of creating GALS system is data transfer between two clock domains. Due to the fact that one domain of the system does not know about the clock information of other domain, the signal may be changed at the sampling edge of the clock of other domain. Then setup and hold time violation occur. This violation causes flip-flops to go to metastable state. Then flip-flops behave unpredictably [11]. If the system has an asynchronous element, metastability is unavoidable. For this reason, we must reduce the probability of occurring metastability in design process [3]. The advantage of synchronous design is to be able to avoid the system going to timing violation. However, since the timing violation is unavoidable in domain crossing, it is important what we will do after the violation occurs [9]. In fact, The FF will finally turn into a stable state. The solution is to let the system resolve this problem, providing enough time. The best way of providing time is to accommodate two flip-flops to make the signal synchronous [12].

It is an important task to perform a GALS system without data loss or corruption and to dissipate energy to this system without a lot of power consumption. Phase control method and double jump technology are some of the solutions of this asynchronous problem. But these solutions are not efficient [8]. The best well-suited approach of this task is to construct first in first out (FIFO) memory buffer. As indicated by its name, the first data accommodating in the buffer will be the first data that retrieves from the buffer. In this way, a FIFO that perform with two independent clock inputs can be built. The data which is written with one clock edge can be read another clocks sampling edge. So data can be safely passed from one clock domain to another clock domain [13].

FIFOs are frequently used to data caching and storing especially different clock frequency or phase. FIFOs provide quickly, safely and more flexible data transfer between clock domains and is used in the standard interface [3]. Additionally, High performance and high complexity digital systems frequently require data transfer between different even unrelated clock domains. Thus, FIFOs are often used data transfer between processing blocks [14]. Asynchronous FIFOs have two clock domains. The data stored in one clock domain is retrieved by another clock domain [15]. Since clocks are completely independent, the probability of data loss cannot be zero.

FIFO is a circular array which consists of identical cells. It includes full and empty flags and control logic for write and read operation. Flags observe FIFO state and decide whether FIFO full or empty. Control logic controls the write and read counters. A written data does not move as long as the reading request comes. As FIFOs have low latency, the data can be read immediately as soon as it is written [16]. Asynchronous FIFO clocks are running at a different time because of clocking with two different domains [17]. The most difficult part of designing an asynchronous FIFO is to design empty and full flags. Because the write and read counters are equal both empty and full status of FIFO. The status must be distinguished on another. Adding an extra bit to counters is one solution. Although other bits are equal in empty and full status, these extra bits are equal in empty status, unequal in full status [8]. In addition, it is hard to synchronize binary count value from one clock domain to another. Because many bits may change at the same time, metastability occurrence ratio increases. For instance, when four-bit counter move from 7 (0111) to 8 (1000) all bits change [17]. It is a good way to use Gray code instead of binary for counters. Every next value differs only one bit from the previous [3]. It is a common approach to construct to write and read counters with Gray code. Due to this technique, counters can be compared and status flags can be set asynchronously [18].

Asynchronous FIFO behaved as a data bridge between high-speed Analog to Digital Converter (ADC) and ARM processor. It provides for flowing and storing data. If high-speed data is sent to ARM processor directly, ARM processor has difficulty processing these data. FPGA can cope with this problem to creating asynchronous FIFO [19]. Due to the features of FPGAs such as excellent control logic, low power consumption, high reliability, reconfigurability, and low development cost, it is very advantageous to constitute a high-speed asynchronous FIFO on FPGA [8].

This thesis aim is to calculate the delay time between the input and the output signals of SAW biosensors. For this purpose, we need to sample two high frequencies signal (input, output). To understand the time delay between two signals, we will implement some signal processing algorithm tools. In this project, the data will be processed using an ARM processor and the signals will be sampled using a Dual-ADC; 12-bit, 80 Msps, 1.8 V.

There will be data loss while transferring high-speed signals from ADC to ARM due to their difference of speed in data processing. The best solution, proposed by the commercial product, is to place a FIFO buffer between the ADC and the ARM. So, we will design multi-clock FIFO buffer on FPGA. This FIFO buffer will save the 50 Msps data from ADC and send it at low speed to the ARM processor.

ARM processor will control the ADC and FPGA card will communicate to ARM processor via UART. Our main objective is to make an FSM design containing FIFO and UART modules. To take this thesis to the next step, we will implement all the ARM processing tools into FPGA; then the final data will be sent to PC. The block diagram of this system is given in Figure 1.3.

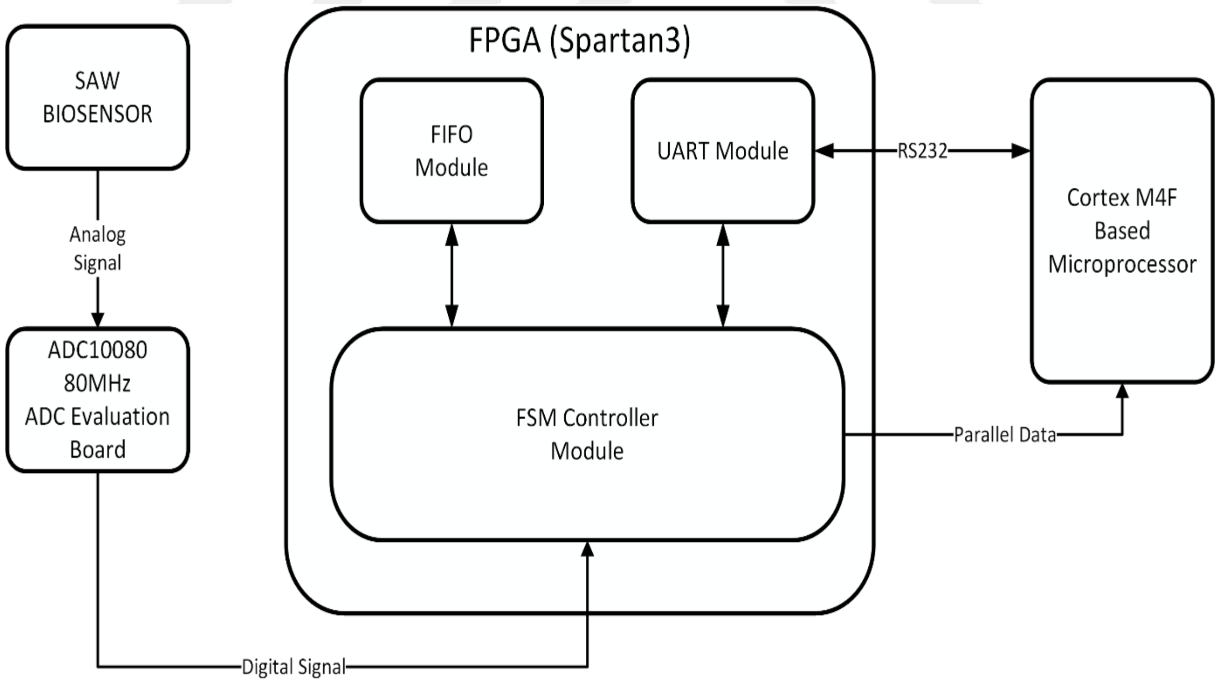


Figure 1.3: The block diagram of the target system.

There are three main objectives for this project. The first is to research synchronization strategies and asynchronous design. The second is to design a FIFO and to transport data between two unrelated clock domains. The last one is to develop a communication packet protocol to control this FIFO efficiently and easily.

This thesis is organized as follows. In chapter 1 some brief information about SAW biosensors and FPGAs are summarized. This chapter also includes the purpose of the thesis, some knowledge about related work and target system. Purpose of the thesis is to design dual-clock FIFO then the necessary background information for understanding the problems are explained. Chapter 1.1 discusses metastability and synchronization background and focuses on the transfer of data between completely unrelated clock domains. Chapter 1.2 introduces structure and parameters for all styles of FIFO buffers. To be able to understand the dual-clock FIFO structure firstly single-clock FIFO structure is explained. Afterwards, problems of dual-clock FIFOs are described.

In chapter 1.3 some brief information about UART communication protocol which is used to communicate between FPGA and external device in this project are mentioned briefly. In chapter 2.1 the packet protocol which is created by us for this thesis is described in detailed. In chapter 2.2, some tools used in this thesis to make experiment easily are explained. Chapter 2.3 describes the hardware implementation of the dual-clock FIFO and our packet protocol. This chapter also explains IP cores and our modules. Chapter 3 includes result of the experiments by graph. In chapter 4 is discussion part. Finally, chapter 5 summarizes the work presented and proposes future related work. Furthermore, the Verilog codes and datasheet of our packet protocol are added to the appendix.

1.1. METASTABILITY

1.1.1. Clock and Synchronization

In an ideal situation, the entire digital system is driven by the same clock signal. The sampling edge of this clock signal arrives all registers of the system at the same time. However, it is not possible in reality. Thus, the non-ideal clock signal must be considered when a digital system is designed. Clock skews must be considered, insomuch as, if it needs, large system must divide to subsystems.

1.1.1.1. Clock Distribution Network

Clock distribution network drives all flip-flop of the system. This distribution network is designed at transistor level because it is not a logic function. Also, the reset signal connects to all flip-flop of the system. So, the construction of reset signal is similar to the construction of clock signal. However, implementation of the reset signal is easier and less critical than clock signal because it does not have strict timing constraints.

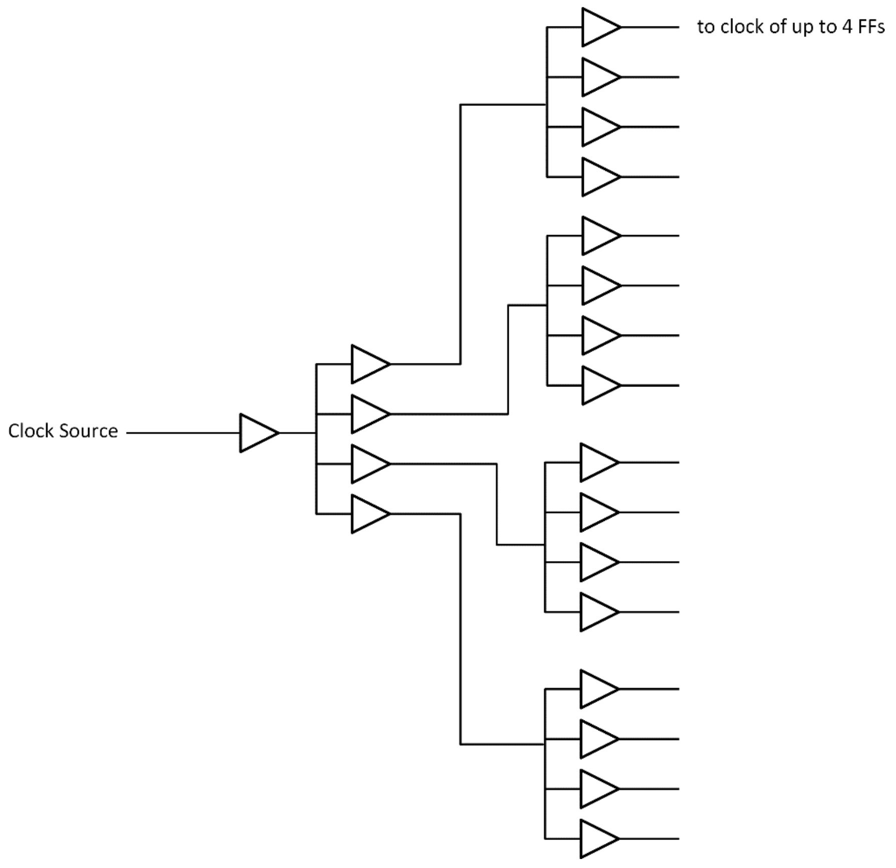


Figure 1.4: Conceptual clock distribution network.

In a conceptual clock distribution network given in Figure 1.4, each buffer drives four parts. So, the propagation delays of each flip-flop are different. In a recursive H-shaped network given in Figure 1.5, the wire lengths from the clock source to the input port of flip-flops are nearly equal. Thus, the propagation delays from the clock source to the input port of flip-flops are nearly identical. So, the sampling edge of the clock signal arrives to the input port of the flip-flop at almost same time.

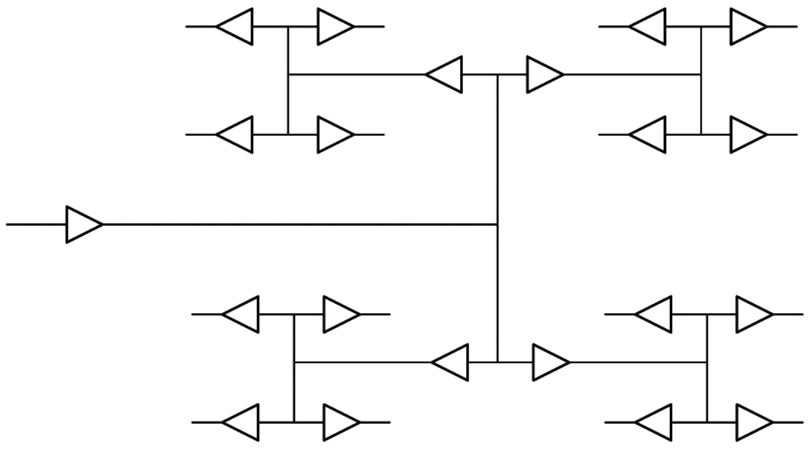


Figure 1.5: Idealized routing of a clock distribution network.

Propagation delay may be variable because of different buffering and routing. Clock skew is the difference between the arrival time of sampling edge. According to the worst-case scenario, if the system has a great number of the register, clock skew is considered as the difference between the latest and earliest arrival times.

As the number of flip-flop increase in the circuit, the clock distribution network becomes large and more complex. So, it causes to increase arrival times, that is clock skew. As a result, as the size of circuit increases, clock skew increases. The clock distribution network is modified according to the size of circuit and clock rate. In a small circuit, the sampling edge of the clock arrives to all flip-flops nearly same time because the propagation delay is small and almost identical. Clock skew can be ignored in these systems. Moderately- sized systems can be thought such as ideal synchronous system because their clock skew is a few percent of the clock skew. However, this small skew causes to decrease the system performance. Today’s technology provides for ignoring this skew up to 50000 gates. But, in high-speed and large-scale systems, clock skew might even come up to the clock period. Thus, this skew cannot be ignored any more in these systems. One of the solutions of this problem is to divide this system to several subsystems. Each subsystem is driven by the independent clock, the interface between subsystems is asynchronous. Asynchronous interface causes timing violation. Special schemes and protocols are necessary to be sure that data and control signals transfer correctly between subsystems.

1.1.1.2. Timing Analysis with Clock Skew

Clock skew is the difference between arrival times of clock signal's sampling edge to the input port of flip-flops in the system. Clock skew effects especially sequential circuit a lot. Also, it gets the system performance reduced and causes tighter hold time constraint.

Clock to setup: a clock to setup path given in Figure 1.6, starts at the clock input of a flip-flop and ends at register input of the next flip-flop. It passes the q output of flip-flop firstly, then any number of levels of combinatorial logic. This delay is sum of the clock to q delay of the flip-flop, the path delay from that flip-flop to next flip-flop and, the setup delay of next flip-flop. If first and next flip-flops are driven by different clock signals, the clock period of first flip-flop must be greater than path delay.

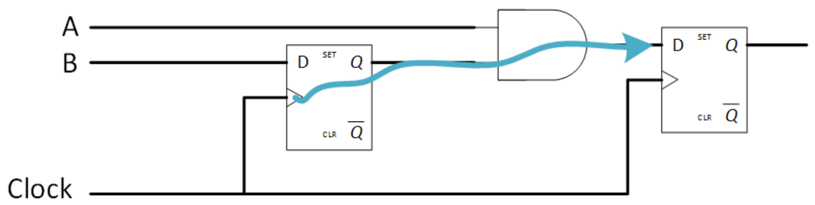


Figure 1.6: Clock to setup path.

Clock to pad: a clock to pad delay given in Figure 1.7, starts at the clock input of flip-flop and ends at output pad. It passes the q output of flip-flop firstly, then any number of levels of combinatorial logic. This delay is sum of the clock to q delay of the flip-flop, the path delay from that flip-flop to the chip output and, output delay.

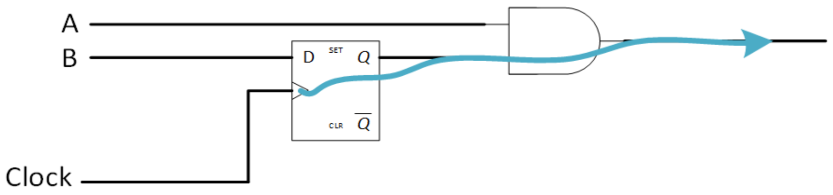


Figure 1.7: Clock to pad path.

Pad to pad: a pad to pad path starts at clock input pad, passes any number of levels of combinatorial logic and, ends at output pad.

Pad to setup: a pad to setup path starts at clock input pad of the chip and ends at d input of the flip-flop.

Paths ending at clock pin of the flip-flop: a path ending at clock pin of flip-flop delay starts at chip input and ends at the clock input of the first flip-flop. It passes the chip input firstly, then any number of levels of combinatorial logic. This delay is sum of the input delay and, the path delay from input to clock flip-flop.

Setup to clock at the pad: a setup to clock at the pad delay starts at input pad and ends at d input of the flip-flop. It passes the input pad firstly, then any number of levels of combinatorial logic.

Clock pad to output pad: a clock pad to output pad delay starts at clock input pad, passes clock input of flip-flop and, ends at output pad.

1.1.1.3. Negative Clock Skew

If the sampling edge of the clock signal arrives destination before the source, negative clock skew occurs as seen in Figure 1.8. In this situation, the clock period must be greater than the sum of the path delay and the clock skew between flip-flop.

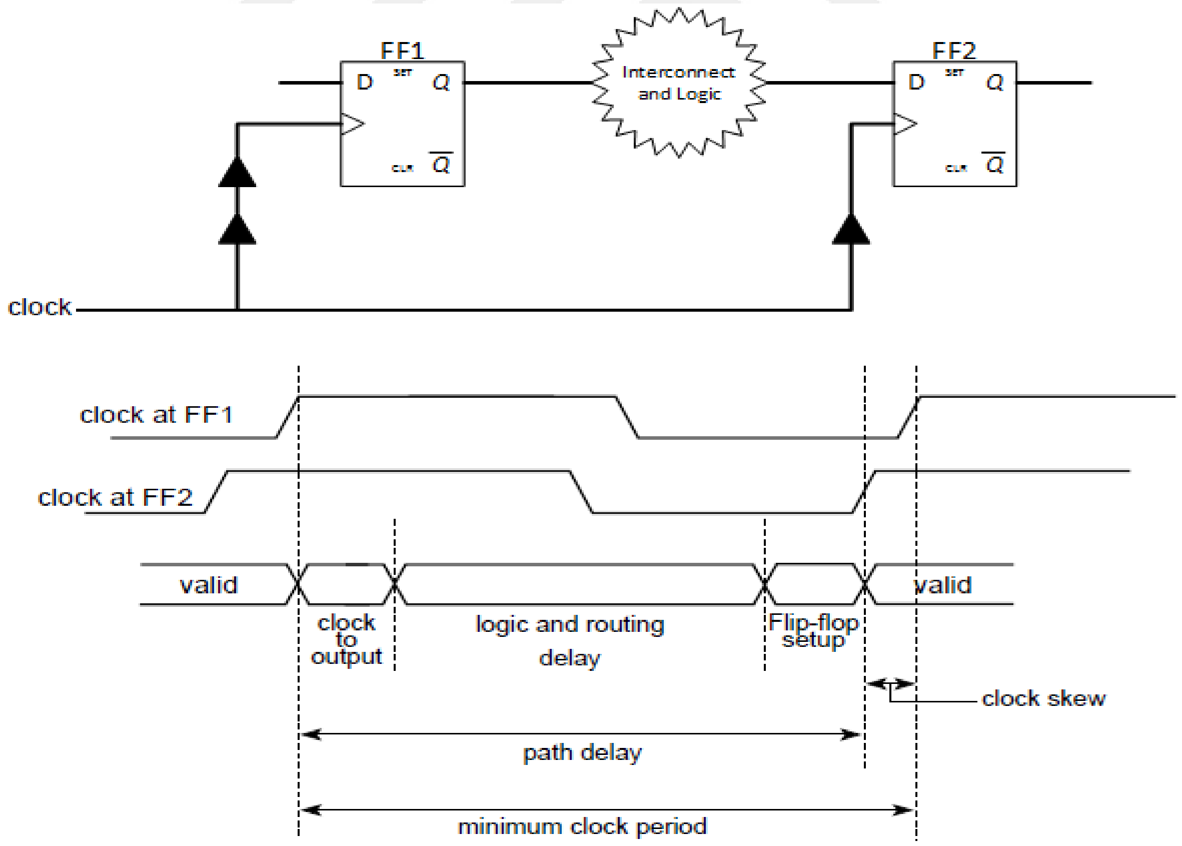


Figure 1.8: Negative clock skew.

1.1.1.4. Positive Clock Skew

If the sampling edge of the clock signal arrives source before the destination, positive clock skew occurs as seen in Figure 1.9. In this situation, the clock period must be minimum the difference between path delay and the clock skew.

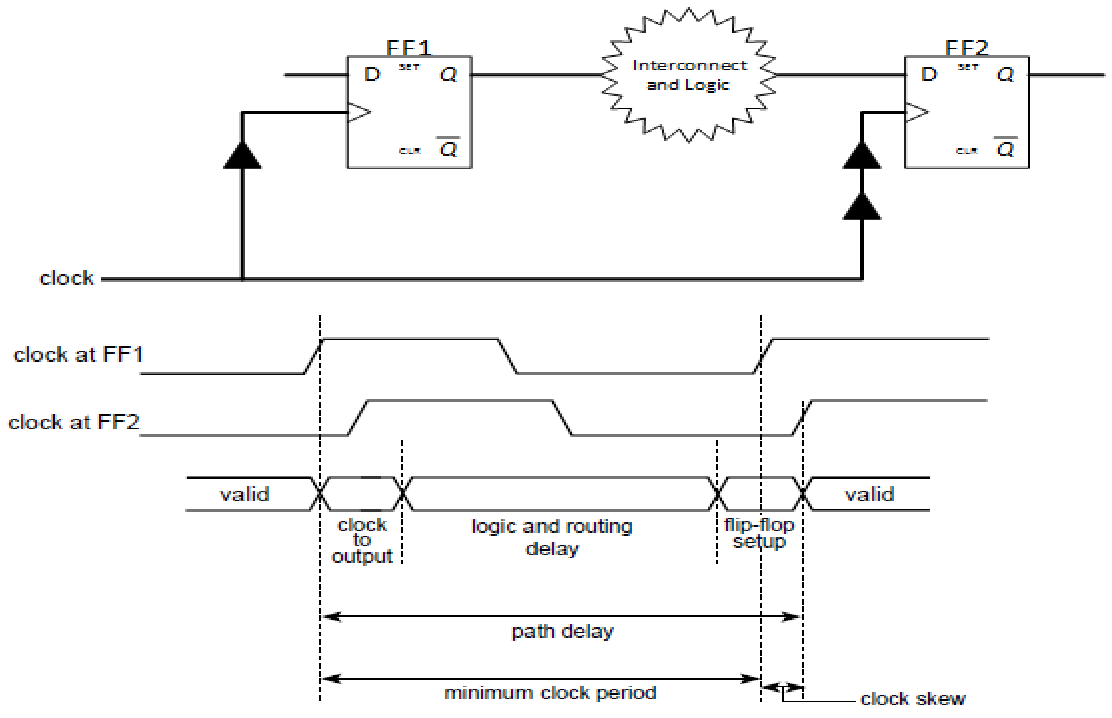


Figure 1.9: Positive clock skew.

As negative clock skew increases clock period, positive clock skew decreases. However, it is difficult to take advantage of this positive clock skew in reality. Because there are many feedback paths in large systems. A system must be designed according to the worst-case scenario. Thus, clock period must be at least;

$$\mathrm{Tcq} + \mathrm{Tnext} + \mathrm{Tsetup} + \mathrm{Tskew} \qquad (1.1)$$

1.1.2. Multiple Clock System

In synchronous design methodology, all flip-flops are controlled by a single global clock. It is difficult or even impossible to implement this methodology as the system becomes large and complex. In some situations, multiple clock becomes obligatory.

- The most digital system must work with external systems. These external systems may not share the same clock.

- As the circuit size becomes large, clock skew increases. Thus, it is necessary to divide the system to subsystems, and these subsystems are driven by different clocks.

- If an external system works at 100MHz even system works at low-speed, it is needed to use 100MHz clock to synchronize this system. So, other externals and the core are worked at 100MHz. This causes to increase the complexity of the system. Dividing this system to subsystems makes the design simple.

- Like the above item, driving a system at 100MHz increases power consumption. Dividing the system to subsystems saves power.

In multiple clock systems, subsystems are synchronous in themselves. A particular interface is designed subsystems to communicate. Subsystems are driven by a derived clock signal or independent clock signal in multiple clock systems. One of the most difficult parts in multiple clock system design is to decide the number of clocks and to route clocks [20].

1.1.2.1. System with Derived Clock Signal

Different frequency and phase clock signals are derived from original clock signal owing to a particular circuit. Moreover, these clock signals route to subsystems. The subsystems route this clock signal to register with its distribution network. Even doing this is easy in theory, clock skew occurs between original and derived clock because of the clock to q delay in reality. Besides, the skew between derived clocks cannot be calculated due to unknown wiring delay and variant clock to q delay.

1.1.2.2. GALS System

Subsystems of a large system are designed synchronous, the interface between these subsystems is designed asynchronous. This approach is called GALS, Globally Asynchronous Locally Synchronous. If convenient asynchronous interface develops, subsystems are thought as synchronous and designed easily. The difficulty of GALS system is how to transfer data between two clock domain. A subsystem may change the signal at sampling edge of the other clock because the subsystem does not know about the other clock. So, this causes setup-hold time violation. It is inevitable at domain crossing. Thus, what to do after the violation is more important.

1.1.3 Metastability Occurrence

The synchronizing problem of two systems which work at different frequency is common. In GALS systems, for instance, data may change at any time even at decision window. This causes time violation. The most basic time constraint of a circuit is a setup-hold time of flip-flops. This means that an input of flip-flop must not change at decision window of sampling edge of the clock. An input must be stable shortly before (setup time) and shortly after (hold time) the sampling edge of the clock to be sure that a flip-flop performs correctly as seen in Figure 1.10.

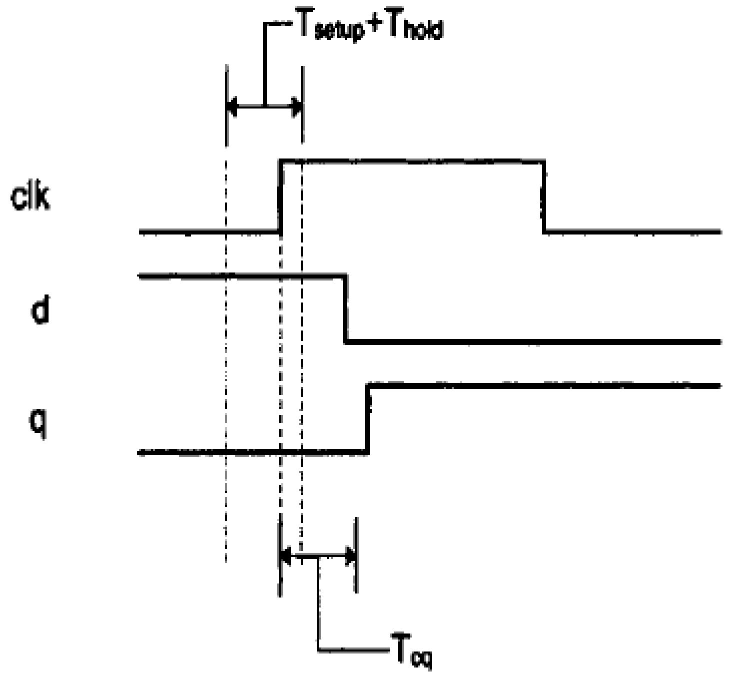


Figure 1.10: Timing diagrams of a D flip-flop.

Synchronization problem between the local clock and the external signal is solved by a flip-flop as seen in Figure 1.11. If setup and hold time of device at data sheet is violated, flip-flop goes into metastable state. It is possible for flip-flop to maintain these operating conditions with the synchronous circuit. Nevertheless, violation is inevitable. The external asynchronous signal must be performed by an internal synchronous signal in two systems which work asynchronously to one another.

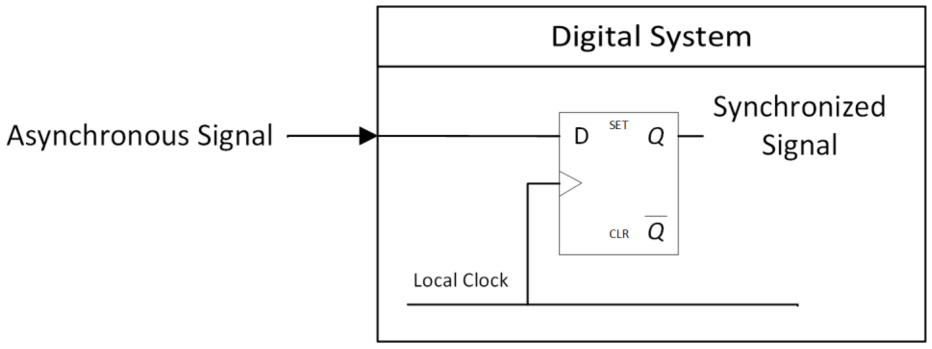


Figure 1.11: Synchronization of the external signal.

Setup and hold time must be kept in D type flip-flop. Reset and set input must not pass from active to inactive at the same time in RS type flip-flop. Otherwise, flip-flop goes into metastable state. In both, flip-flop adopts an undefined and unstable. Q output is unknown. Flip-flop goes into one of the two stable states soon. However, it is impossible to estimate which state be performed.

If input changes at a convenient time according to timing constraint, this data propagates after clock to q delay. If input changes at decision window, three different situations may occur; 0,1 and, metastable (in between voltage). The first one of odds is correct one; any trouble does not occur. The second one means that flip-flop holds the old value. It returns to normal value at the next sampling edge. So, it is not a huge problem. But, the third one is a huge problem. If a flip-flop goes into metastable state, the output voltage is between high and low, and it cannot be interpreted as 1 or 0. If the output of this flip-flop effects continuation of the circuit, as downstream all logic cells pass to unknown state.

In fact, tiny force can make the flip-flop stable again as seen in Figure 1.12. Namely, the flip-flop can reach to the stable state itself. Researches show that metastability is unavoidable in bistable systems. If enough time is provided to the device, it reaches to stable state. This enough time is called resolution time (Tr). It cannot be calculated, but it can be characterized with the aid of probability distribution function. Decay time ($) constant depends an electrical characteristic of flip-flop.

$$\mathrm{P(Tr)} = \mathrm{e} - (\mathrm{Tr}/\ \$) \tag{1.2}$$

Because decay time is a fraction of a nanosecond in today’s technology, flip-flop maintains metastable state Tr time after the clock edge.

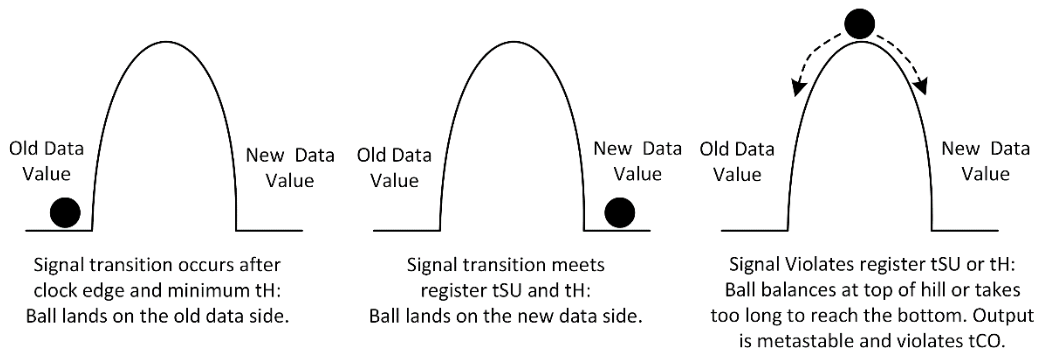

Old Data Value
New Data Value
Signal transition occurs after clock edge and minimum tH: Ball lands on the old data side.
Old Data Value
New Data Value
Signal transition meets register tSU and tH: Ball lands on the new data side.
Old Data Value
New Data Value
Signal Violates register tSU or tH: Ball balances at top of hill or takes too long to reach the bottom. Output is metastable and violates tCO.

Figure 1.12: Metastability represent.

1.1.3.1. MTBF

Setup and hold time cannot be 0 because of the physical structure of flip-flop. Thus, there is not faultlessly working synchronizing circuit. It is essential to prevent in-between value formed at the output of the flip-flop to spread because timing violation occurs at the asynchronous circuit. If the flip-flop cannot solve metastability in given time, this is called synchronization failure. MTBF is used to explain the quality of synchronization circuit and the reliability of the design. Mean time between failure is the average time between two synchronization failure. This is calculated with the frequency of the signal (fin), the clock frequency of synchronization circuit (fclk) and, critical time window length (td).

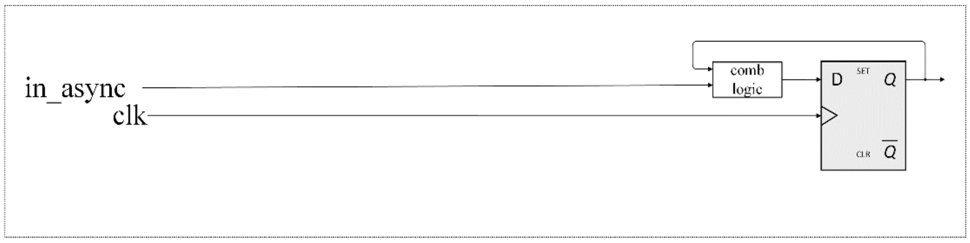
$$\mathrm{Rmeta} = w * \mathrm{fclk} * \mathrm{fin} \tag{1.3}$$

$$\mathrm{AF(Tr)} = \mathrm{Rmeta} * \mathrm{P(Tr)} = w * \mathrm{fclk} * \mathrm{fin} * e(-\mathrm{Tr}/\$) \tag{1.4}$$

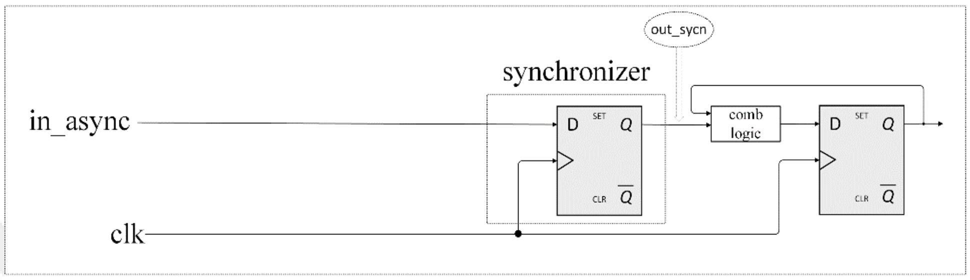
$$\mathrm{MTBF\,(Tr)} = 1 \,/\, \mathrm{AF(Tr)} = e(\mathrm{Tr}/\$) \,/\, (w * \mathrm{fclk} * \mathrm{fin}) \tag{1.5}$$

1.1.4. Synchronizer

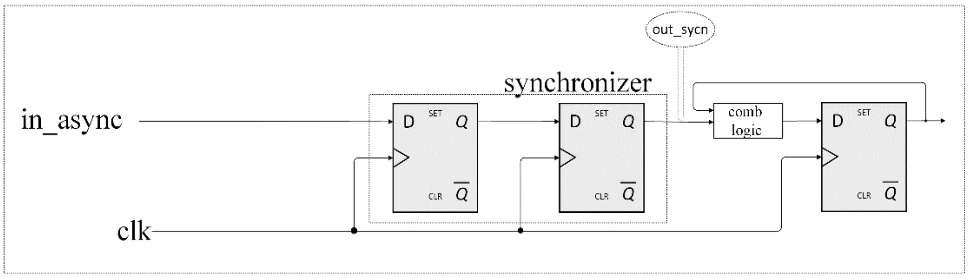
If an asynchronous input causes time violation, flip-flop becomes metastable, and in-between voltage is observed in the output of the flip-flop. This in-between voltage spreads to all circuit if it does not prevent. As the name implies, synchronizer makes asynchronous input synchronous with the system clock. Because there is any chance to prevent metastability in the bistable device, synchronizers can only prevent in-between value to spread all circuit. Synchronizer provides enough time to flip-flop to fix metastability itself. Some type of solution for asynchronous signal is explained below and given in Figure 1.13.



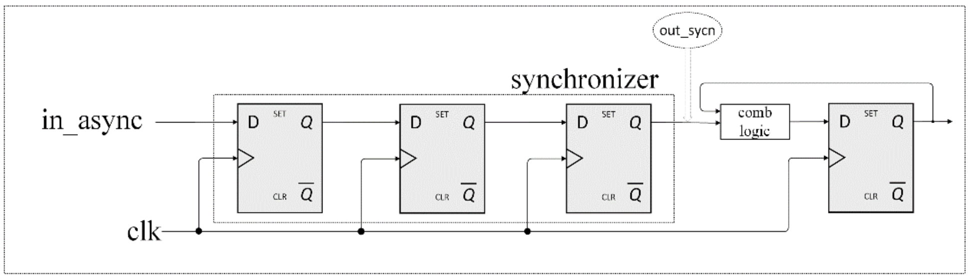
(a) No synchronizer



(b) One-FF synchronizer



(c) Two-FF synchronizer



(d) Three-FF synchronizer

Figure 1.13: Synchronizer types.

1. No synchronizer: if asynchronous signal causes timing violation, register goes into metastable state. If enough time is not provided to fix itself, in-between value spreads to all system.
2. One flip-flop synchronizer: according to the path from synchronizer's D flip-flop to system's D flip-flop, synchronizer provides one clock period to flip-flop to solve metastability.

$$\mathrm{Tr} = \mathrm{Tclk} - (\mathrm{Tsetup} + \mathrm{Tcomb}) \tag{1.6}$$

As we are known, tiny change of Tr extremely increases MTBF. Reducing slight Tcomb increases MTBF a lot because Tr is depended on Tcomb. Tcomb can be decreased by modification of combinatorial circuit.

3. Two flip-flop synchronizer: as we are known, Tcomb must be 0 to obtain maximum Tr. If two flip-flop is used, Tcomb before system flip-flop is 0. Then Tr = Tc – Tsetup. So, MTBF extremely increases. If these two flip-flops accommodate away from each other, wiring delay causes problem. Thus, these two flip-flops must be placed as close as possible. This approach is more common than other due to their robustness and applying easily.

   Many ASIC technology libraries have specialized D flip-flop cells. They are designed to reduce w, $ and, Tsetup then MTBF increases. Because these cells are several times larger than normal D flip-flop, they are not often used.

4. Three flip-flop synchronizer: the probability of reaching stable state increases when one cascade D flip-flop adds to two D flip-flop synchronizer. Resolution time becomes 2*(Tclk - Tsetup) from (Tclk - Tsetup) in two D flip-flop synchronizer. This gets MTBF increased a lot. Despite this advantage, it postpones the input signal one clock period more (from two periods to three periods). Because two D flip-flop synchronizers get MTBF increased sufficiently, three D flip-flop synchronizers are not needed.

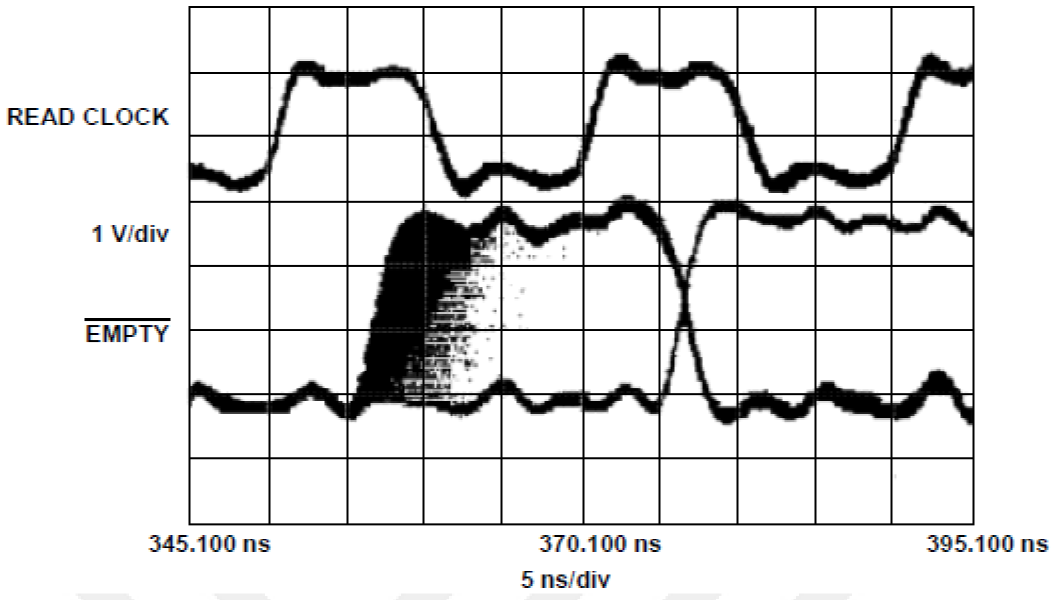


Figure 1.14: Signals of FIFO with one flip-flop synchronizer.

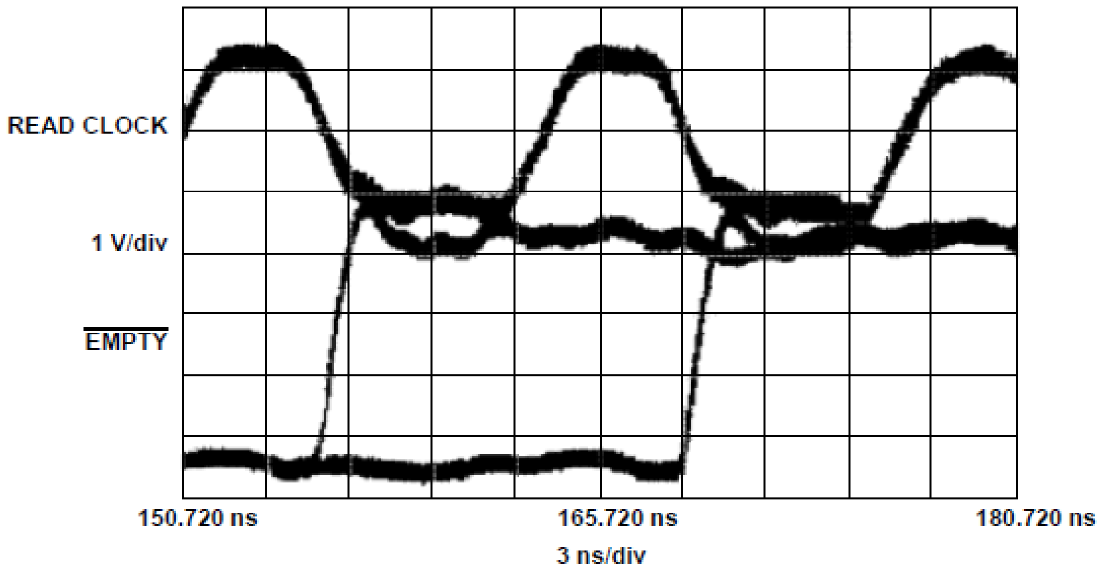


Figure 1.15: Signals of FIFO with three flip-flop synchronizer.

The figures shown above, represent the result of one and three flip-flop synchronizer circuit at metastable state. In one flip-flop synchronizer as seen in Figure 1.14, flip-flop sometimes decides 0, sometimes 1 at first clock edge. It takes much more time than normal. At the second clock edge, output reaches stable again in both situations. In three flip-flop synchronizer as seen in Figure 1.15, sometimes empty signal delays one clock cycle. However, metastability is not observed.

1.1.5. Crossing Clock Domain

In GALS systems, clock domains use an independent clock signal. Subsystem might have to work with much faster or much slower subsystem. Synchronizers only prevent the system to go into the metastable state. Also, the control circuit is needed to transfer data between two clock domains.

1.1.5.1 Single Enable Signal

Edge detection scheme: most digital systems include enable signal to start an operation. This signal must be sampled only one time not much. If enable signal comes from the slower domain, it is seen during several clock cycle at a current domain like a very wide pulse. Sampling this enable signal correctly is possible with edge detection. If enable signal comes from the faster domain, the sampling edge of the synchronizer D flip-flop might miss this narrow pulse.

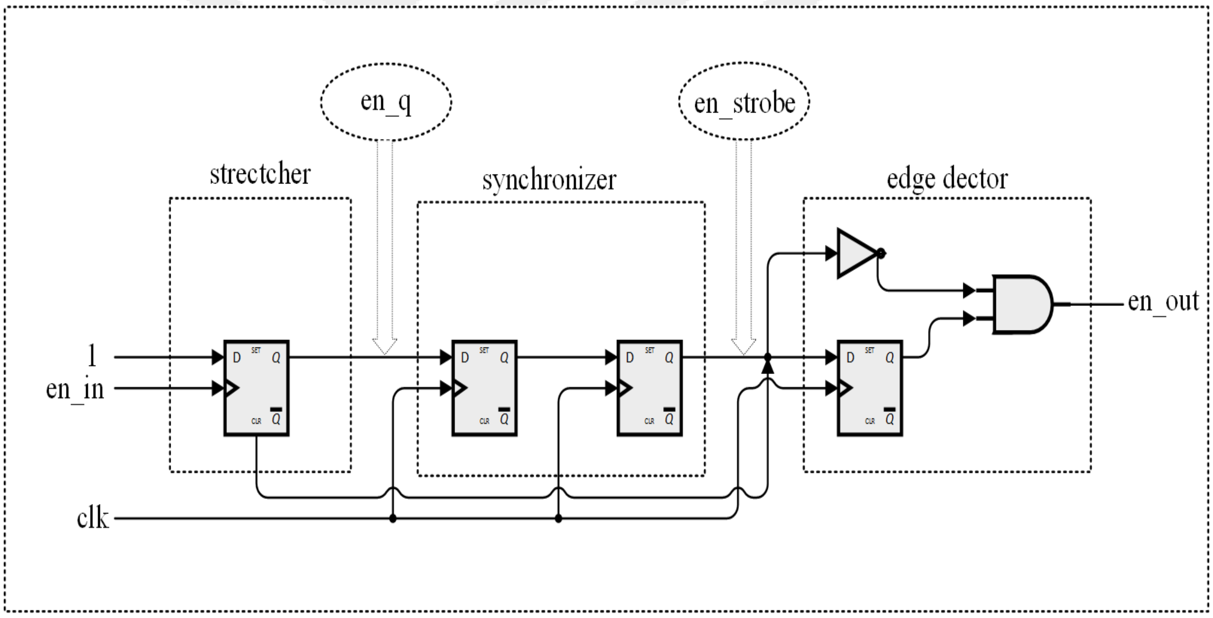


Figure 1.16: Regeneration of a narrow enable signal.

As seen in Figure 1.16, the stretch circuit is needed. The enable signal which comes to the stretcher’s D flip-flop behaves like clock signal and sends 1 to output. When the output of synchronizer becomes 1, stretcher D flip-flop is reset. This enable signal is seen during one or two clock period due to the synchronizer. So, edge detection is needed again.

1.1.5.2. Handshaking Protocol

If the subsystem from which enable signal comes too faster than current subsystem, missing data is possible. Two systems must communicate to each other to avoid this data missing and construct more robust systems. In this protocol, receiving subsystem sends feedback to another subsystem, and this subsystem regulates the enable signal.

Four-Phase Technique: the most common technique to communicate two clock domain. These two subsystems never pay attention to the clock rate of each other. Thus, this approach can be used in many implementations. Two subsystems are talker and listener. Talker sends req signal to the listener, listener sends ack signal to the talker. These req and ack signals need synchronizers because they are generated from different clock domain. These four phases as seen in Figure 1.17:

- Talker makes req signal as 1.
- Listener makes ack signal as 1 when it recognizes req signal is 1.
- Talker makes req signal as 0 when it recognizes ack signal is 1.
- Listener makes ack signal as 0 when it recognizes req signal is 0, and they return to initial state.

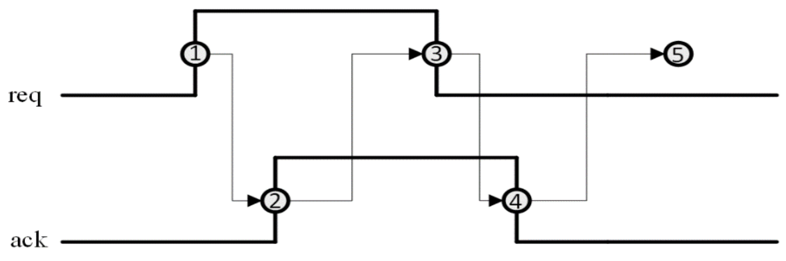


Figure 1.17: Timing diagrams of the four-phase handshaking protocol.

Two-Phase Technique: in four-phase technique, signals are activated at first term then they are deactivated at second term to be sure to return to initial. Two-phase can be used to improve efficiency. These req and ack signals need synchronizers again. Phases:

- Talker makes req signal as 1.
- Listener makes ack signal as 1 when it recognizes req signal is 1.
- Handshaking is over when talker recognizes ack signal is 1. At the next enable, respectively req and ack signals are deactivated and return to initial state.

1.1.6. Data Transfer

In synchronous systems, data transfer consists of only data passing, and it takes only one clock cycle. The interface between domains includes commands signals, data lines and, address lines in many implementations. The best approach is to bundle data and to coordinate with enable signal. Thus, it is enough to focus the synchronization of enable signal instead of all signals. However, it is not enough to solve the synchronization problems to transfer data reliably. Preventing data losses or duplicates is necessary.

1.1.6.1. Four-Phase Data Transfer

Highest overhead but most robust. At writing operation, talker makes req signal 1 and puts the data on the data line. When the listener recognizes req signal is 1, it retrieves the data from the data line and makes ack signal 1. When the talker recognizes ack signal is 1, it takes the data from the bus and makes req signal 0. Then, listener makes ack signal 0. At reading operation, talker makes req signal 1. When the listener recognizes req signal is 1, it puts the data on the data line and makes ack signal 1. When the talker recognizes ack signal is 1, it retrieves the data from the bus and makes req signal 0. Then, listener makes ack signal 0. In this technique, the synchronizer is needed. If two D flip-flop synchronizers are used, only one writing operation takes 7Tct + 6Tcl, too much.

1.1.7. Synthesis of A Multiple-Clock System

Designing multiple-clock system takes advantage of synchronous methodology. The most important point is to satisfy the timing constraints in synchronous methodology. The purpose is to prevent timing violation. An interface is needed when the multiple-clock system is divided into synchronous subsystems. Additionally, we can think as a regular synchronous system during design part because subsystems use the same clock within themselves. Crossing domain interface must include synchronization circuit and data transfer protocol. Designing this interface is more difficult than synchronous system design. The best method is to think as if synchronization circuit and data transfer interface are different modules.

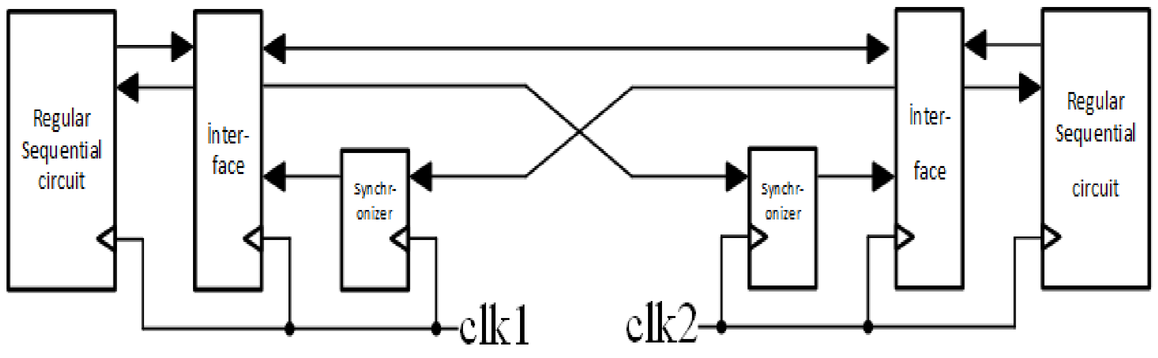


Figure 1.18: System with two clock domains.

Handshaking method is not efficient especially in large systems although it is reliable. Two different subsystems which write and read data via buffer are needed instead of direct transfer. Asynchronous memory or shared memory can be used for this purpose. These methods cannot prevent metastability, but it can reduce this. In this thesis, asynchronous FIFO buffer is used to provide data exchange between two different clock domains. In the next part, FIFO buffer architecture is covered primarily; then asynchronous FIFO buffer is covered. Two flip-flop synchronizers mentioned in this part, are used to prevent metastability in these asynchronous FIFO buffer as seen in Figure 1.18. Asynchronous FIFO buffer is used instead of crossing domain interface.

1.2. FIFO BUFFER

As we know, data transfer is often used in most of the today's digital systems. If data transfer at a higher speed than the speed of our system some problems occur such as data loss. Additionally, system processes data irregularly or batch data transfer causes some similar problems. In this situation, it is necessary to construct buffer and storage. Similar approaches are observed in our daily life. For instance, bank counters work slowly and continuously. The number of the incoming customers is irregular and variable. The customer who comes to the bank take row number. Thus the customer who comes first is served first.

This type of buffers is also advisable for the interface between systems which run at different speeds. In this system, the slowest component determines the all system speed including data transfer. For example, in a compact-disk player, the speed of the disk's rotation determines the data rate. The data rate of the ADC is controlled by a quartz crystal to take advantage of the speed independent of repeated audio fluctuations. The different data rates are compensated by buffering. Thus, the sound fluctuations are highly independent of speed of the disks' rotation [21].

It is a common way to use FIFO to transfer multi-bit data from one clock domain to another safely. Also, it is used to control the data flow between two systems working in the same clock domain [15]. Using this type of buffers became common because they are versatile. Thus FIFOs progressed from most basic logic functions to high-speed buffers such as SRAM.

FIFO is a special buffer type. FIFO means first in first out so the data which is written to the buffer first come out from the buffer first given in Figure 1.19. Another buffer type is shared or stack memory in other words LIFO. It means last in first out so the data which is written to the buffer last come out from buffer first. Which type of buffer we must choose is depends on the application.

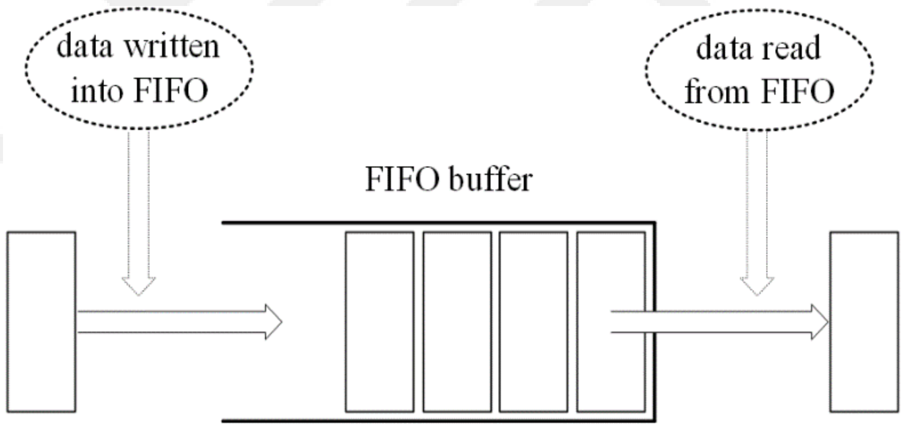


Figure 1.19: Conceptual diagram of a FIFO buffer.

It is possible to construct software or hardware FIFO. According to application and desired features hardware or software construct of FIFO is chosen. Because of flexible structure of software FIFO to modify FIFO is rather easy. It is enough for this to revise the code. However, in hardware construct, it is necessary to make new board layout. The advantage of hardware FIFO is to be able to run at higher speed than software FIFO.

1.2.1. FIFO Types

FIFO has had three different types throughout their development though they are not used today.

1. Shift Register: Most basic FIFO type. They can store an invariable number of data. When one data is written to this type of FIFO, one data must be read. Thus, write and read operations must be synchronous.
2. Exclusive Read / Write: Slightly more advanced type of FIFO. Although they can store variable number of data, write and read operations must be synchronous due to their internal structure.
3. Concurrent Read / Write: The most advanced type of FIFO. They can store a variable number of data. Also, read and write operations can be asynchronous.

Shift registers are not used to store data because their structure is first in first out by nature. Instead of this type, other type architectures, which can store an invariable number of data, are preferred. If write and read operations have a specific time condition, it is necessary to be synchronism between two systems such as exclusive read / write FIFO. If there is not any time constraint, it means write and read operations can be out of synchronism, concurrent read / write FIFO is preferred.

Writing data cannot be independent of reading data in the exclusive read / write FIFO. There is time relation between write clock and read clock. It is needed to add external synchronization circuit to use this FIFO type for two systems which work asynchronously to one another. However, these circuits considerably reduce the data rate. Writing data and reading data are independent of each other in concurrent read / write FIFO. It is possible to be overlapping or sequential reading/writing. So read and write systems can work at different frequencies. Taking care of synchronization between two systems is necessary. The control of this type of FIFO is performed by writing and reading clock signals. This FIFO fall into two types as synchronous and asynchronous. Exclusive read / write FIFO came out on the market first because it is rather easy to perform them. Today, concurrent FIFO is used in most implementations. Concurrent FIFO can be used in synchronous systems without any difficulty. Also, concurrent FIFO can exchange data between systems at different frequencies. It is enough to construct an internal synchronization circuit to achieve this.

1.2.2. FIFO Architecture

There are many different hardware architecture types of FIFO. Traditional FIFO architecture improved continually. They have fall-through architecture at first. Nowadays, they work such as SRAM which can store a great number of the word at high-speed. FIFO can be constructed both hardware and software.

Fall-through FIFOs are first generation architecture. It can be thought of like people waiting in line. So when the first person goes, everybody proceeds one unit. The time duration to shift data from input to output is called fall-through time. This time increases according to FIFO length. For instance, in 16x5 fall-through FIFO, the time duration from real clock to full signal can be 400 ns. This type of FIFO is not used for large length FIFO today because the time duration of shifting all data is really high. Besides, fall-through architecture has high latency, low power efficiency and, low memory density. Thus, this can be used only for very small FIFO.

1.2.2.1 Circular FIFO

The problems of fall-through architecture are solved by building circular FIFO and using two pointers as write and read as seen in Figure 1.20. This method is more efficient than fall-through. Nowadays, this architecture is always used. Data can be written or read from the array arbitrarily. Circular FIFO provides low latency, high energy efficiencies by the nature of random access. One of the most important features is that this design does not get difficult according to length. In circular FIFO, fall-through time is totally independent of length, unlike fall-through FIFO. So, it is possible to build fast and long FIFO with them. It is enough to revise the control logic circuit and to expand write and read pointers to increase the length. Also, this FIFO must keep track three things. These are empty and full flags and data occupancy between empty and full status.

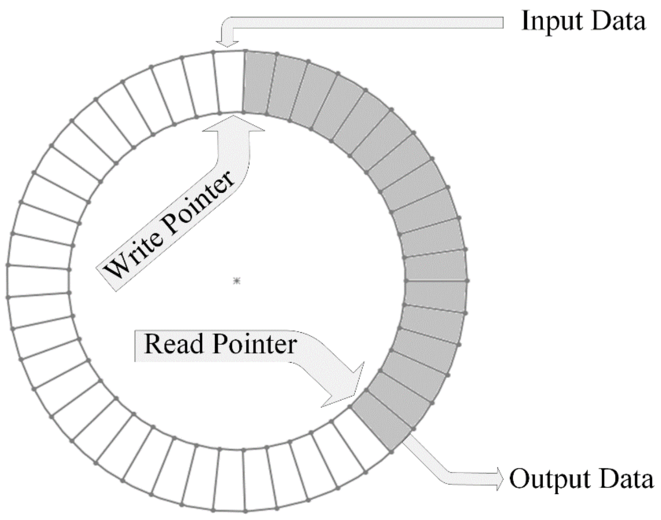


Figure 1.20: Circular FIFO with two pointers.

Hardware implementation of circular FIFO needs dual-port SRAM to store data. Write and read pointers keep memory addresses of SRAM as a binary counter. The number of the memory location of SRAM is 2^n, for instance, it is enough to use n-bit read / write pointers. It is a great advantage to be low bit number pointers even long SRAM given in Figure 1.21.

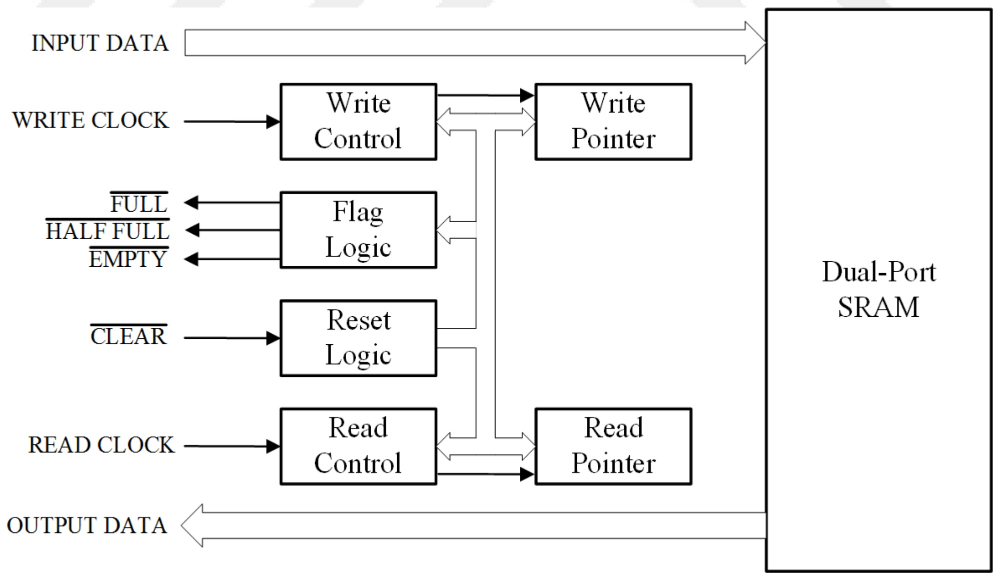


Figure 1.21: Block diagram of FIFO with static memory.

In a circular FIFO, also known as parallel FIFO, write pointer keeps incoming data addresses, read pointer keeps addresses of data written to FIFO. So, write pointer keeps location which will be written next, read pointer keeps location which will be read currently. If the buffer is not full and write signal is asserted in the rising edge of the clock, input data is stored and write pointer indicates the next memory location. Similarly, if the buffer is not empty and read signal is asserted in the rising edge of the clock, read pointer indicates the next memory location and release the read slot to future write. After incrementing and routing delay time, the read data appears on the output port. Read signal works as "remove signal". Both pointers indicate same memory location when the reset signal is asserted. Read pointer follows write pointer consistently. If read pointer catches up with write pointer, FIFO is empty. Similarly, if write pointer catches up with read pointer, FIFO is full.

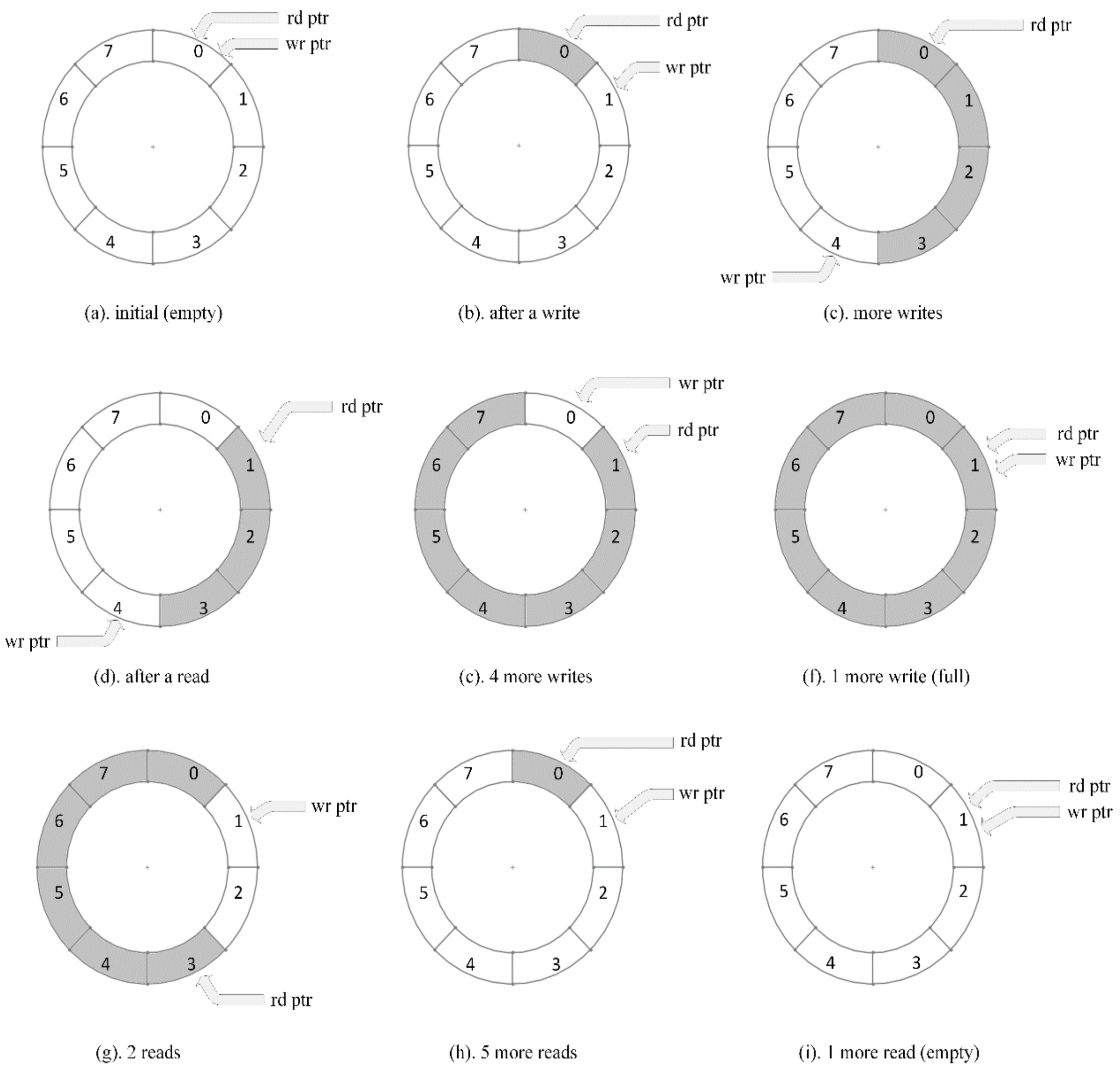


Figure 1.22: Circular FIFO buffer.

As seen in Figure 1.22, both write and read pointers indicate 0th address at the beginning. At that moment, reading cannot be operated because FIFO is empty. When one writing operates, incoming data is placed to 0th address and pointer starts to indicate the next location. When one reading operates, the data in the 0th address is sent to output and read pointer start to indicate the next location. If many writings operate without any reading, FIFO goes to full status. Any more writing cannot be operated. If many readings operate without any writing, FIFO goes to empty status again. Any more reading cannot be operated.

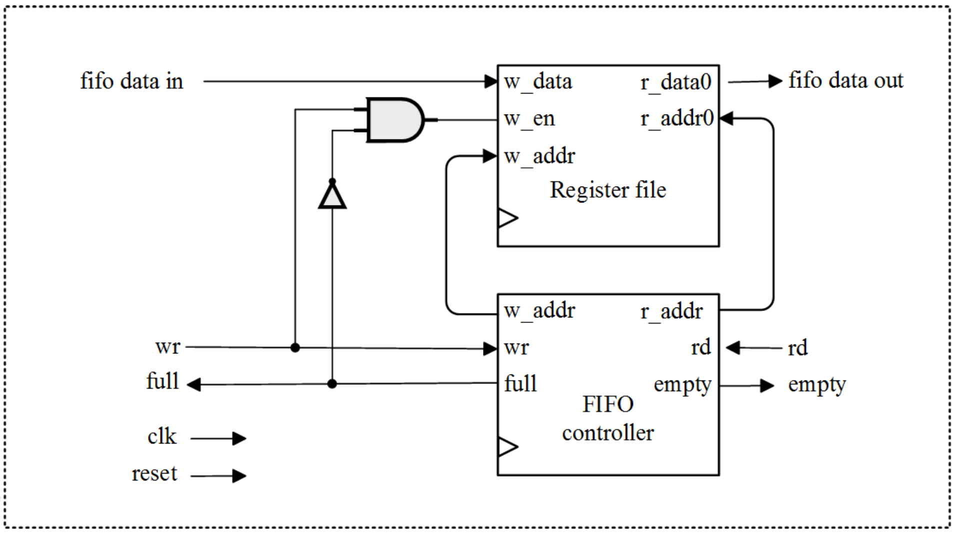


Figure 1.23: Block diagram of a register-based FIFO buffer.

Read address is generated by read pointer, write address is generated by the write pointer. Full and empty flags are generated by different flag logics. In addition to, some FIFOs have half full, almost full and, almost empty flags. Pointers and status logics of static memory FIFOs must be initialized as seen in Figure 1.23. If external system attempt to write data when FIFO is full, overflow occurs. Similarly, if external system attempt to read data when FIFO is empty, underflow occurs.

FIFO buffer must include empty and full status signal to be sure that the system works correctly. In the correctly designed system, external system must control the status signal before attempting to access to FIFO. It is necessary to have some security measure to be sure that data cannot be written to full FIFO and read from empty FIFO. Also in these erroneous attempting to access situations, full and empty status signals and read and write pointers must not change, must keep the previous value.

The difficult part of the control circuit is to differentiate two particular cases of FIFO buffer as empty and full status. If read and write pointer are equal, it means empty status in ordinary FIFO. When circular FIFO is empty, write and read pointers are equal. However, when circular FIFO is full, write and read pointers are equal again. Thus, empty or full status of FIFO cannot be determined only according to write and read pointers. It can be distinguished by which pointer causes the equality. If reset or read pointer cause the equality, the buffer is empty. If write pointer causes it, the buffer is full. There are many status signals generate schemes which include additional circuit and flip-flops. Two of them is explained.

FIFO control circuit with augmented binary counters: In this approach, write and read pointers are 1-bit augmented binary counters shown in Figure 1.24. Full and empty conditions are determined by comparing the MSB of these counters. We think that there is a FIFO which can be stored 8 words. Write and read pointers are 4-bit counters. 3-bit LSB of counters keep addresses. These 3-bits are equal in empty and full conditions. MSB of these counters is used to differentiate empty and full status. These two MSBs are equal in empty status. After 8 writing operations, write MSB is opposite to read MSB. It means FIFO is full. After 8 reading operations, write and read MSBs are equal again. It means FIFO is empty again.

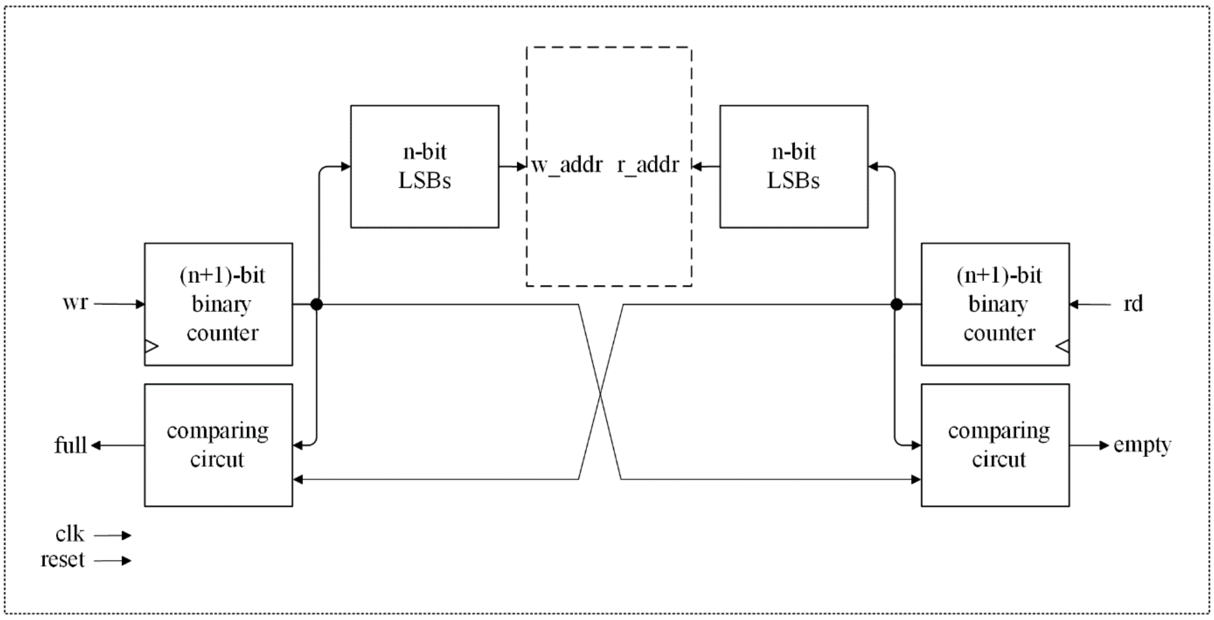


Figure 1.24: Block diagram of an augmented-binary-counter FIFO control circuit.

FIFO control circuit with status Flip-flops: Another approach to design the control circuit of FIFO’s empty and full state is to store empty and full status signal with a flip-flop. New conditions can be determined by read, write signals and these stored status data. This scheme does not need augmented counters, but it needs two extra flip-flops to record empty and full status. The full status flip-flop is set as 0; empty status flip-flop is set as 1 at the beginning. After that, on every rising edge of the clock, write and read signals are analyzed, and pointers and status flip-flops are modified.

- If write and read signals are 0, there is not any operation, and flip-flop remains.
- If write and read signals are 1, write and read operates same time. Then pointers are increased 1 but flip-flops remain because buffer size does not change.
- If write signal is 1 and read signal is 0, only write performs. Firstly, buffer status is controlled. If the buffer is not full, write pointer is increased 1 and the empty flip-flop is deasserted. If the buffer is full after increasing write pointer, full flip-flop set as 1.
- If write signal is 0 and read signal is 1, only read performs. Firstly, buffer status is controlled. If the buffer is not empty, read pointer is increased 1 and the full flip-flop is deasserted. If the buffer is empty after increasing read pointer, empty flip-flop set as 1.

In the first approach, counters enlarge 1-bit to determine FIFO status. This approach cannot work with non-binary counters. In the second approach, status signals rely on the successive value of the counters. It is often preferred because it can be performed with all counter types such as gray code counter.

It becomes increasingly important to use FIFO in high-speed systems. It is common way to use FIFO especially in data exchange between asynchronous clock domains. Asynchronous systems also called as unsynchronous system, don’t have match clock phase and frequency. This mismatch occurs intentionally or unintentionally. For example, in SoC with multiple clock domain systems, the mismatch is intentional. However, the mismatch causes of high clock skews in unintentionally in large clock distribution network.

One of the most common techniques of transferring data between clock domains is to design a FIFO [12]. FIFO can be used for transferring data in parallel or serial communication [3]. Storing data into the buffer with one clock signal and retrieving data from the buffer with another clock signal seems like an ideal and easy way to transferring data between different clock domains. However, designing the control circuit of empty and full status can be challenging [12]. ASIC designer has difficulty in constructing dual-clock FIFO [22].

Asynchronous FIFO is also known as dual-clock FIFO or mixed clock FIFO. Asynchronous meaning is thought as if lack of clock, in fact, it is term of FIFO with two clocks. It is good to understand synchronous FIFO before asynchronous FIFO. Because many concepts of both architectures are same. The most difficult part of designing dual-clock FIFO comparing to single clock FIFO is which information must be transferred between clock domains and how they are transferred. Otherwise, counter parts of both architectures are very similar.

1.2.3. Synchronous FIFO

All digital processors work synchronously with the system-wide clock. There is always a system clock even if anything is executed. Enable signal, also called as a chip-select signal, is activated by write and read operations.

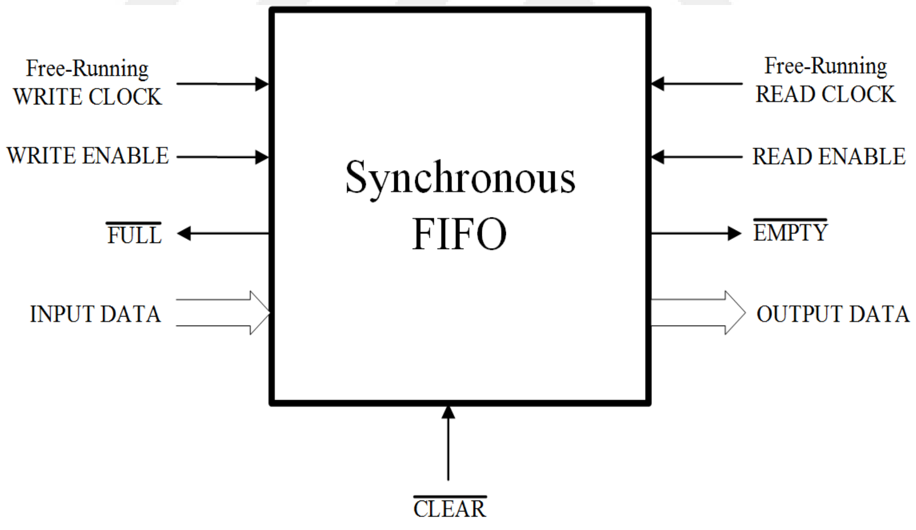


Figure 1.25: Connections of a synchronous FIFO.

The clock free-running from writing and reading systems is needed. The writing system is controlled by a write-enable signal synchronized with the write clock. The full status signal is synchronized with write clock by free-running clock. In an analogous manner, read enable signal is synchronized with read clock signal. The empty status signal is synchronized with read clock by free-running clock. Synchronous FIFO as seen in Figure 1.25, can be integrated easily into many processor architectures because full and empty signals are fully synchronized with the free-running clock.

1.2.4. Asynchronous FIFO

Synchronous FIFO design techniques help to design asynchronous FIFO shown in Figure 1.26. There are two difficult problems which cannot be ignored while asynchronous FIFO is being designed. One of them is to determine full and empty status according to read and write pointer. Another one is to synchronize the asynchronous domains without any metastability.

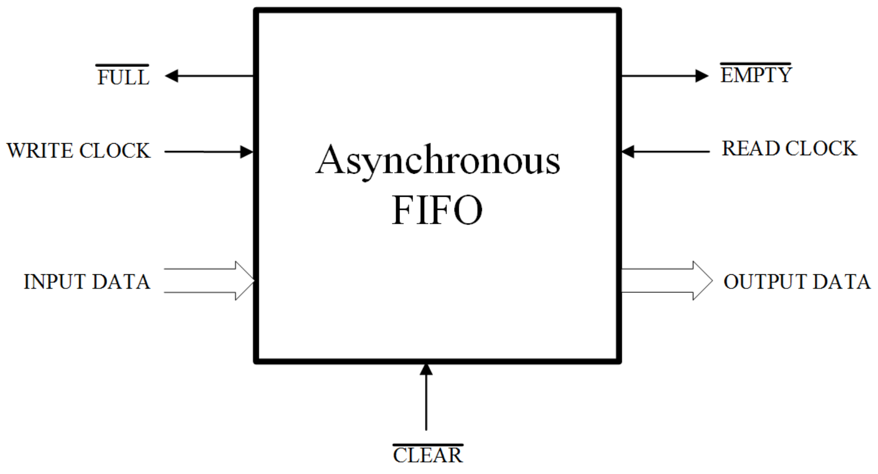


Figure 1.26: Connections of an asynchronous FIFO.

It is good to examine empty and full status at first. There is only one counter in synchronous design. This counter is increased one in every write operations and decreased one in every read operations. If both write and read operates in same clock edge, these counter holds. So, if the counter is 0, FIFO is empty, or if the counter is maximum value, FIFO is full. It is impossible to use only one counter in asynchronous FIFO because the counter is controlled by two different and asynchronous clock signal. So, it is necessary to use two counters for write and read operations. These two counters must be compared to determine full and empty status. The difficulty of circular FIFO is that pointers are equal both empty and full conditions. Gray code counters are useful for FIFO counters because Gray code only allows one bit to change for each clock edge [17].

1.2.4.1. FIFO Control Circuit with a Non-binary Counter

Write and read pointers can be binary in synchronous FIFO. Instead of n-bit counters (n+1)-bit counters can be used to determine empty and full status. But, multiple bits may change at the same clock edge in these counters. For example, when 3-bit binary counter wraps around from ‘111’ to ‘000’, 3 bits change at the same clock edge. This situation, which does not cause any problem. In synchronous FIFO, is huge problem in asynchronous FIFO. In multiple-clock systems, if multi-bits of the data, which is transferred between domains, change, errors occur. The best way to solve this problem to use (n+1)-bit Gray code counter as seen in table 1.1 instead of (n+1)-bit binary counter. N LSBs of the (n+1)-bit indicate write and read address, (n+1)th bit provides to distinguish empty and full status.

Table 1.1: Circulation pattern of 4-bit and 3-bit Gray counters.

4-Bit Gray counter	3 LSBs of 4-Bit Gray counter	3-Bit Gray counter
0000	000	000
0001	001	001
0011	011	011
0010	010	010
0110	110	110
0111	111	111
0101	101	101
0100	100	100
1100	100	000
1101	101	001
1111	111	011
1110	110	010
1010	010	110
1011	011	111
1001	001	101
1000	000	100

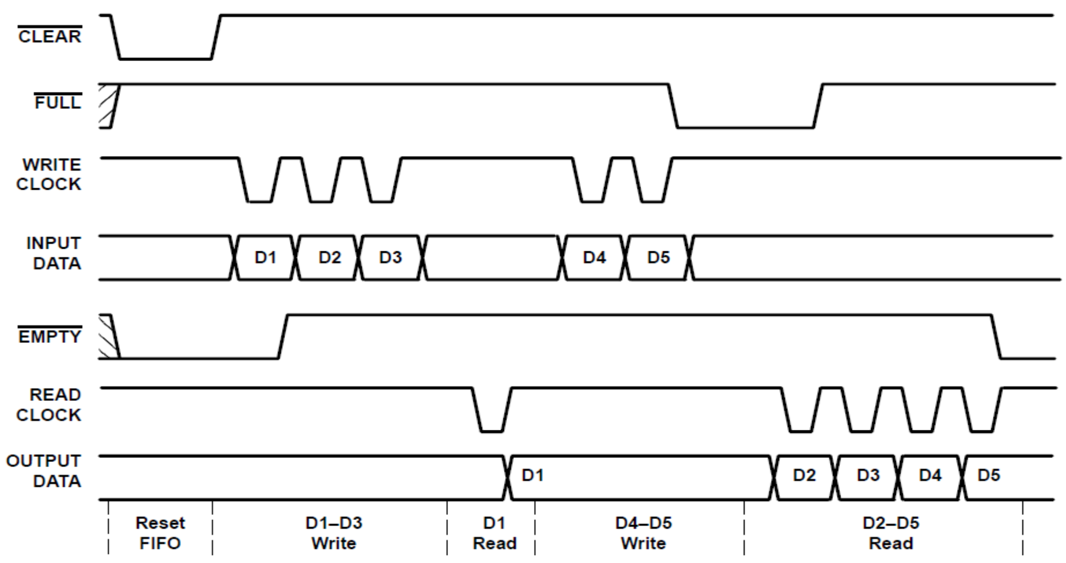


Figure 1.27: Timing diagram for asynchronous FIFO of length 4.

The full status signal is controlled before attempting to write to asynchronous FIFO. If FIFO has some free space, input data is stored at the sampling edge of the write clock. Analogously, the empty status signal is controlled before attempting to read from asynchronous FIFO. If FIFO has data, the data is retrieved at the sampling edge of the read clock. A timing diagram of an asynchronous FIFO with 4 slots is in Figure 1.27. As it is seen, empty and full signals are reset at first. Empty signal changes after one data is written. Full signal changes after one data is read and 4 data are written. Full signal returns to normal after one data is read. Also, the empty signal returns to normal after three more data are read.

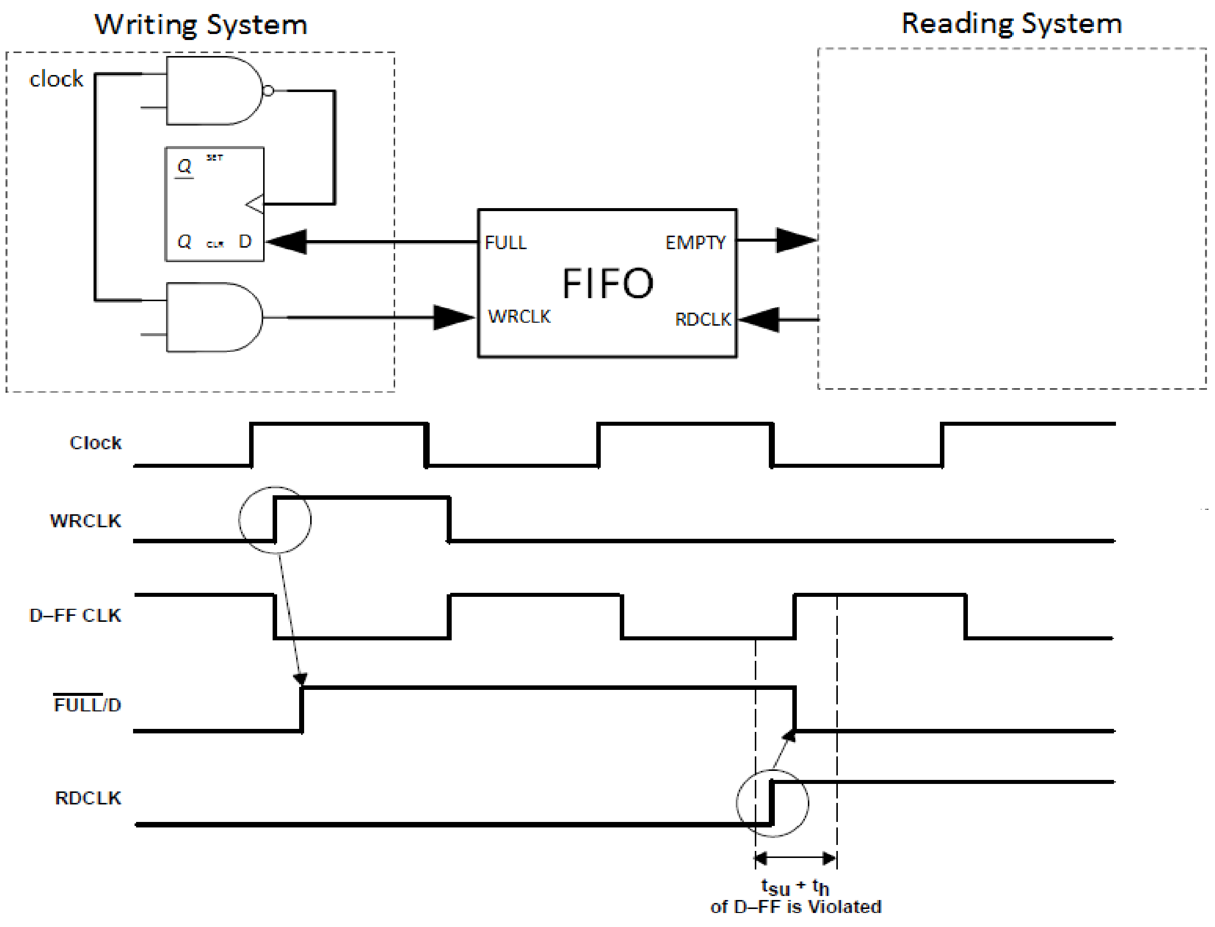


Figure 1.28: Asynchronism when resetting full signal.

The Figure 1.28 represents how time violation occurs in asynchronous FIFO. If FIFO has only one empty slot, full status is set at sampling edge of the write clock. Read clock resets full signal when a data retrieves. Full signal is connected to input port of a D flip-flop which is driven by another clock signal. Thus, the full signal can be reset at the setup-hold time of the D flip-flop. So, timing violation occurs and flip-flop goes into a metastable state. The same is true for empty status. The empty signal must be synchronized with the reading system and reset by writing system. Full status must be the opposite. Synchronization of these signals must be done externally.

As a result, the best solution is to use synchronizer for an asynchronous FIFO not to go into metastable state as it was mentioned in metastability part. Again, as it was mentioned in synchronizer section of metastability part, synchronizer can include 1,2 or, 3 flip-flops. One flip-flop synchronizer is not enough for advanced systems. Three flip-flop synchronizer is not preferred because it decreases the speed of systems. Two flip-flop synchronizer technique is the most common approach as seen in Figure 1.29.

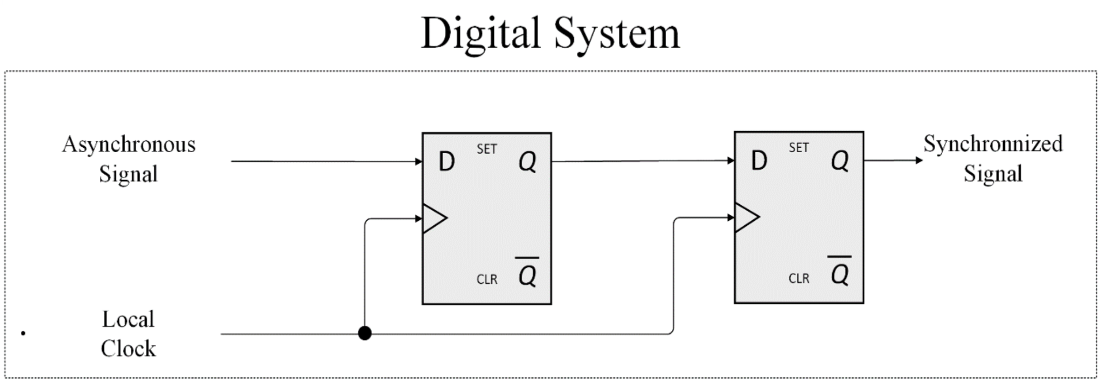


Figure 1.29: Block diagram of two-level synchronization.

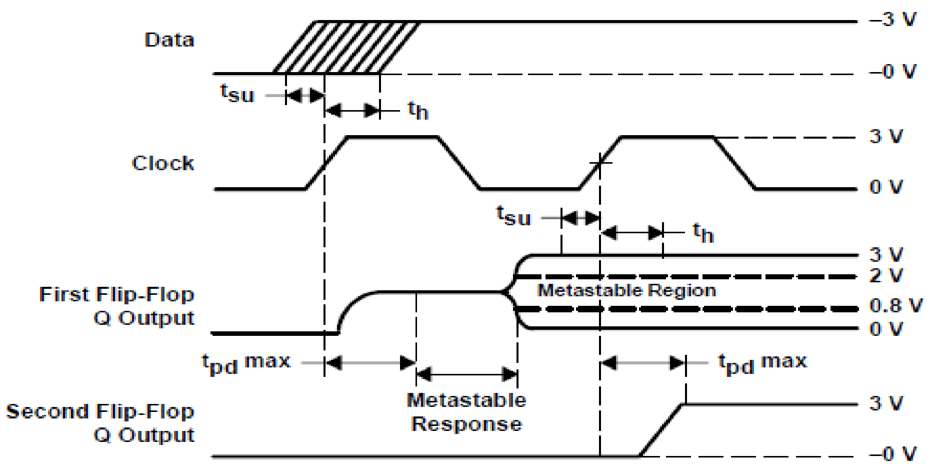


Figure 1.30: Timing diagram for two-level synchronization.

The Figure 1.29 represents a circuit of two flip-flop synchronizer, and the Figure 1.30 represents timing diagram of two flip-flop synchronizer. As seen above, the second flip-flop may go into metastable state, if the first flip-flop is in the metastable state already for a while. Moreover, if the signal period of the clock is greater than sum of the delay of the first flip-flop and setup time of the second flip-flop, the second flip-flop will never go into the metastable state. However, especially in high-speed systems, it is not possible. So, the second flip-flop cannot prevent metastability failure but can decrease appreciably.

1.3. UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

UART (Universal Asynchronous Receiver Transmitter) is a physical interface that sends a bit of parallel data in or out at each time, which is used to send over a serial communication network. Throughout the history of personal computers, data transfer between devices and computers has often been achieved via serial ports. The mouse, keyboard, and other peripherals are also connected to the computer in this way. Interfaces such as Ethernet, FireWire, and USB also send the data as a serial stream.

The serial communication protocol term is generally used for the RS232 standard. In RS232 standard, -12V is interpreted as 0 (low), +12V is interpreted as 1 (high). The RS232 TxD is in the idle position at -12V. However, the devices that we send data via RS232 do not usually work in this voltage range. Therefore, we need to add a converter between the external device and RS232 port. The advantages of serial data transfer compared to parallel data transfer are:

- Serial cables can be longer than parallel cables. In serial port, '1' is represented from -3 to -25 V and '0' is represented from + 3 to +25 V. In parallel port, '0' is represented as 0 V, and '1' is represented as 5 V. For this reason, the voltage drop in the cable at the serial port is not as much of a problem at the parallel port.
- Cables as many as parallel transmission are not needed. Using 3 wires is cheaper than using 19 or 25 wires as in parallel transmission if the device needs to be installed at a distance from the computer.
- The use of microcontrollers has also increased. Many of these have Serial Communication Interface units. Serial communication reduces the number of pins in these microcontrollers. Although 8 pins are used for 8-bit parallel transmission, only two pins are used for serial transmission.

1.3.1. Communication of Devices With RS-232 Port

RS232 communication is asynchronous. It means that clock signal is not sent with the data. Therefore, it is necessary to synchronize the received data with the sent data. Start bit, stop bit, parity, number of data bits and baud rate are the important items for RS232 communication and the receiver device must know them. The receiver device must know the baud rate. It will sample the signal according to baud rate information.

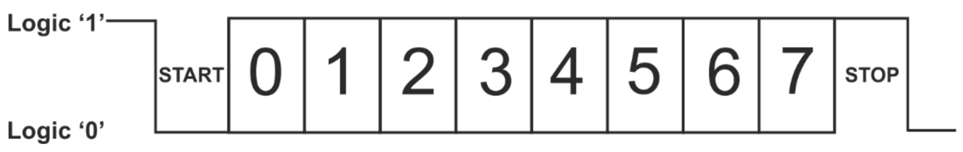


Figure 1.31: Serial transmission waveform.

The Figure 1.31 shows the expected waveform from the UART when the 8N1 form is used. 8N1 infers 8 data bits, No Parity, and 1 Stop bit. Logic '1' means that RS232 line is idle. Transmission starts with a start bit as 0. The LSB (Least Significant Bit) bit is sent first. Finally, the transmission is completed by sending a stop bit as 1. The data sent in this way is called as "framed data". That is, the data is framed between the start bit and the stop bit.

1.3.2. UART Settings

1.3.2.1. Baud Rate

Baud rate is measured in units of bits per second (bps) that determine how long a bit is to be sent and received. The baud rate parameter can be 300, 1200, 2400, 4800, 9600, 19200, 115200. The baud rate of the sender and receiver must be same so that the send data can be received correctly on the receiver side. Some devices can automatically determine the baud rate. Although the RS232 standard is officially limited to 20,000 bps, the serial ports used in PCs can be set to 115,000 bps. Since the speed also includes the frame bits, the effective data rate is lower than the bit rate. For example, in 8N1 form, only 80% of the bits can be used for data.

1.3.2.2. Data Bit

The number of data bits can be 5, 6, 7, 8(most common) or 9. In newer implementations, 8 data bits are used almost universally. 5 or 7 bits usually work with relatively older devices. Most serial communication designs send bits in bytes, with the LSB (least significant bit) being the first to send. This standard is also called "little endian". There is also a "big endian" standard that is rarely used, in other words, MSB (most significant bit) is the first to send.

1.3.2.3. Parity

Parity is a method of detecting errors that occur during transmission. An extra bit is sent along with the data bits. This bit is set so that the number of 1 bits in each character is always odd or always even, including the parity bit as seen in table 1.2. If the parity is false, it means that this byte is corrupted. If the parity is true, there is no error, or there is an even-numbered error. It is obtained by xor (exclusive or) each bit. Odd parity if the result is 1, even parity if the result is 0. The even parity is calculated by not operating after the Xor operation of all the bits of the data. Single parity, all bits of the data Xor operating. The odd parity is more frequent than the even parity, since at least one state transition occurs, which makes it more reliable.

Table 1.2: Calculating the parity bit.

7 bits of Data	Count of 1-bits	8 bits including parity	
		Even	Odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

1.3.2.4. Stop Bit

At the end of each transmitted byte, stop bits are sent so that the receiver can be synchronized again. Electronic devices usually use a single stop bit. Rarely, when slow devices are used one and a half or two stop bits are required.

2. MATERIALS AND METHODS

2.1. COMMUNICATION PACKET PROTOCOL

As known, we purpose to transfer the acquired samples from the high-speed ADC to the low-speed external device in this thesis. To achieve this purpose, a FIFO buffer which can work with the dual clock was built on FPGA. Additionally, this system can transfer the specific rate data to the external device without writing to FIFO. It was needed to give some settings and commands to this system to improve usability. External device must send some message to set the system either to write FIFO or to transfer directly. These settings:

- sample_num: the number of samples needed, max: 224-1.
- div_num: the rate of the sample, max: 224-1.
- bit_num: the bit number of samples, max: 8. Although we aimed to use 10-bit ADC in this project, we had to choose 8-bit FIFO word length on Spartan-3. For this reason, the maximum value of this setting is 8. If setting value greater than 8 is sent, an error message is received, and the setting value must be sent again.

Besides when the user wants to change the settings they can be cleared. Above all, any operation cannot happen before all settings are done. The system sends success message when settings are done successfully. Then the data can be written to FIFO, can be read from FIFO and be sent directly to the external device. The user can stop all these operations at any moment. Data can be written to FIFO and be read from FIFO at the same time, but data cannot be sent directly when the FIFO is busy. We consider necessarily having an option which gets to know about FPGA status to provide convenience to users when working with these constraints. We decided that it is useful for the system to send success or error message to the user during these operations. It is needed to have a systematic and strong communication packet protocol for this useful messaging. For this reason, we decided to construct own protocol. We analyzed some professional device’s data sheets. We decided what was needed. As we mentioned earlier, we prefer UART, which is one of the most common communication protocols, to communicate between FPGA and external device. So, users can use this system with many devices which support UART communication. The UART settings of the serial communication interface are selected as seen in table 2.1;

Table 2.1: Port Settings.

Parameter	Value
Baud rate	9600
Data bits	8
Parity	Even
Start bits	1
Stop bits	1
Flow control	None
Bit order	Least Significant First (after start bit)

The serial communication has two channel, master-slave interface between an external device and the FPGA. The external device is “master, ” and it initiates all communications. The FPGA is “slave” and generates a reply to each received message. For this document “received” messages mention to those from the master device to the FPGA, and “reply” messages refer to those from the FPGA to the master device. Additionally, FPGA can send “error” and “success messages without master’s permission.

2.1.1. Packet Protocol

First of all, we added process code to the first byte and stop code to last byte. The second byte is status code. External must send this byte as 0x00. FPGA edit this byte and send back in case of any packet error. Third-byte mention command. The fourth byte indicates the number of arguments. Next argument bytes are sent. Later CRC byte is sent. Finally, the packet, shown in table 2.2, is finished with a stop code.

Table 2.2: Packet protocol bytes.

Byte #	Upper Byte	Comments
1	Process Code	Set to 0x02 on all valid incoming and reply messages
2	Status Byte	See 3.2.1
3	Function Byte	See 3.2.2
4	Byte Count Byte	See 3.2.3
N	Argument Bytes	See 3.2.4
N+1	CRC Byte	See 3.2.5
N+2	Stop Code	Set to 0x03 on all valid incoming and reply messages

2.1.1.1. Status Byte

External must send status byte as 0x00. If the packet is received correctly, the status byte of the reply message from FPGA is 0x00 (same as original packet). Otherwise, the packet has an error.

FPGA edits each bit of status byte according to these errors as seen in table 2.3. If status byte is received different from 0x00 then 0th bit, if command byte is invalid then 1st bit, if the argument count byte is incompatible with command then 2nd bit, if the CRC (checksum) byte is false then 3rd bit, if stop byte is different from 0x03 then 4th bit, and if timeout occurs (stop code is not received 20 ms after process code is received) then 5th bit is edited. This obtained status byte is sent back to the external device.

Table 2.3: Status byte.

Bit Number	Error Code
Status[0]	Check the incoming status byte is zero
Status[1]	Check the function byte is valid
Status[2]	Check the byte count is valid
Status[3]	Check the CRC is matching
Status[4]	Check the stop code is 0x03
Status[5]	Timeout occurred (20ms)

2.1.1.2. Function Byte

The function-code byte specifies the function of incoming message. For all reply messages, the FPGA will echo back the function-code byte. A full list of function bytes shown in table 2.4.

Table 2.4: Function bytes.

Function Code	Command
0x11	Set_Sample_Num
0x22	Set_Division_Num
0x33	Set_Bit_Num
0x66	Send_Data_Direct
0x99	Stop
0xCC	Clear_Settings
0xFF	FIFO_Write
0xFD	FIFO_Read
0xAA	Get_FPGA_Status
0xBB	Success_Message
0xEE	Error_Message

2.1.1.3. Byte Count

The byte-count byte is used to specify the number of argument bytes in the message (not total number of bytes in the message). See Function Code Table for the expected byte count associated with each function-code byte.

2.1.1.4. Argument Bytes

The argument bytes encode the argument of message packet. See Function Code Table for argument definition for each message. Little-endian ordering is employed: Byte 2, Byte 1, Byte 0.

2.1.1.5. CRC Byte

On all incoming and outgoing messages, a CRC byte is calculated by bitwise XOR operation. CRC byte is calculated for first N byte except for process code (0x02).

2.1.2. Command List

0x11: the number of samples needed, have 3-byte argument so maximum value can be 2^24-1.

0x22: the rate of samples, have 3-byte argument then the maximum value can be 2^24-1. This value must be 1 when all samples are wanted.

0x33: the bit number of samples, have a 3-byte argument, the maximum value can be 8.

0x66: direct send, to transfer data to output without writing FIFO, have no argument, when transfer the requested number of data finish success message is sent.

0x99: stop command, have a 1-byte argument. If the argument is 0x55 then sending directly, if the argument is 0xEE then writing FIFO, if the argument is 0xDD then reading from FIFO operation, if the argument is 0xAA then what operation is working is stopped. If FPGA has no operation, then an error message is sent to external. Furthermore, the operation is stopped by stop command cannot send success message because requested number of data are not sent yet.

0xCC: this command clears the settings. Have no argument. If any operation is working, settings cannot be cleared.

0xFF: to write the requested number of data to FIFO. Have no argument. If FIFO is full, data is overwritten. It sends success message when the writing operation is finished.

0xFD: to read the requested number of data from FIFO and transfer them to output. Have no argument. If FIFO is empty, an error message is sent. It sends success message when the reading operation is finished.

0xAA: to get the status of FPGA, have 1-byte argument this query can be sent at any moment, the user must send this argument as 0x00. FPGA edits this byte according to status in the reply message. If FPGA is idle, the status byte will be 0x00. If settings are not finished yet, the status byte will be 0x01. If FPGA is sending directly, the status byte will be 0x02. If FPGA is writing to FIFO, the status byte will be 0x03. If FPGA is reading from FIFO, the status byte will be 0x04. If FPGA is both writing to FIFO and reading from FIFO, the status byte will be 0x05.

0xBB: unidirectional message, to inform the external about some success situation, have a 1-byte argument. This argument means which operation is completed successfully. If all settings are completed, this byte will be 0xAA. If direct send is completed this byte will be 0x55. If writing to FIFO operation is completed this byte will be 0xEE. If reading from FIFO operation is completed this byte will be 0xDD.

0xEE: unidirectional message, to inform the external about some error situation, have a 1-byte argument. This argument means what error occurred. If bit number is greater than 8, this byte will be 0xBB. If the reading request is received when FIFO is empty, this byte will be 0x11. If any operation request is received when writing or reading operate, this byte will be 0x22. If any operation request is received when sending direct operate, this byte will be 0x33. If any operation request is received before all settings are completed, this byte will be 0x44. If a stop request is received when FPGA is idle, this byte will be 0x66.

Table 2.5: Command List.

Function Code	Command	Byte Count	Argument	Note
0x11	Set_Sample_Num	3	1…(2^24-1)	Sets the sample number
0x22	Set_Division_Num	3	1…(2^24-1)	Sets the division number
0x33	Set_Bit_Num	3	1…8	Sets the output bit number
0x66	Send_Data_Direct	0	-	Commands FPGA to send input data directly to output pins
0x99	Stop	1	Cmd: 0x55 STOP_SEND_DIRECT 0xEE STOP_FIFO_WRITE 0xDD STOP_FIFO_READ 0xAA STOP_ALL	Stop all operations
0xCC	Clear_Settings	0	-	Clear current settings
0xFF	FIFO_Write	0	-	Write input data to FIFO buffer
0xFD	FIFO_Read	0	-	Write output data from FIFO buffer
0xAA	Get_FPGA_Status	1	Cmd:0x00 Reply:	Get current working status. Another use of this command

			0x00 IDLE 0x01 SETTINGS 0x02 SEND_DIRECT 0x03 FIFO_WRITE 0x04 FIFO_READ 0x05 FIFO_WRITE&READ	is to verify proper communication
0xBB	Success_Message	1	Reply: 0xAA SETTINGS_FINISHED 0x55 SEND_DIRECT_FINISHED 0xEE FIFO_WRITE_FINISHED 0xDD FIFO_READ_FINISHED	FPGA sends this message after successfully completed the operation
0xEE	Error_Message	1	Reply: 0xBB WRONG_BIT_NUMBER 0x11 FIFO_EMPTY 0x22 PROCESS_BUSY_FIFO 0x33 PROCESS_BUSY_DIRECT 0x44 SETTINGS_NOT_FINISHED 0x66 UNEXPECTED_STOP	FPGA sends this message after if any error occurred

Only error and success command are unidirectional from FPGA to external as seen in table 2.5. Other commands are sent from external to FPGA and FPGA send a reply message to external absolutely. If FPGA receives packet correctly, the reply message is same as the incoming message. Otherwise, FPGA edits the status byte.

2.2. HARDWARE AND VERIFICATION TOOLS

In this part, all devices and methods used for verification and the obtained results are explained. As it is known, this thesis aims to determine the delay between input and output signals of the biosensor. This delay is minimal. Thus, these signals must be sampled at high frequency. It is not possible to transfer data of high-speed ADC to PC or an ordinary microprocessor. It is needed to add a buffer between ADC and PC to make this possible. The purpose of this thesis is to construct this buffer and to create the communication between buffer and PC.

The best way to implement a high-speed buffer is to utilize FPGA. A dual-clock FIFO is constructed in this thesis. Optional direct transfer without FIFO is also provided. A communication packet protocol which includes some options and commands which provides easy usage during these operations is developed. So, this system can be used not only biosensors but also projects which need to transfer high-speed data.

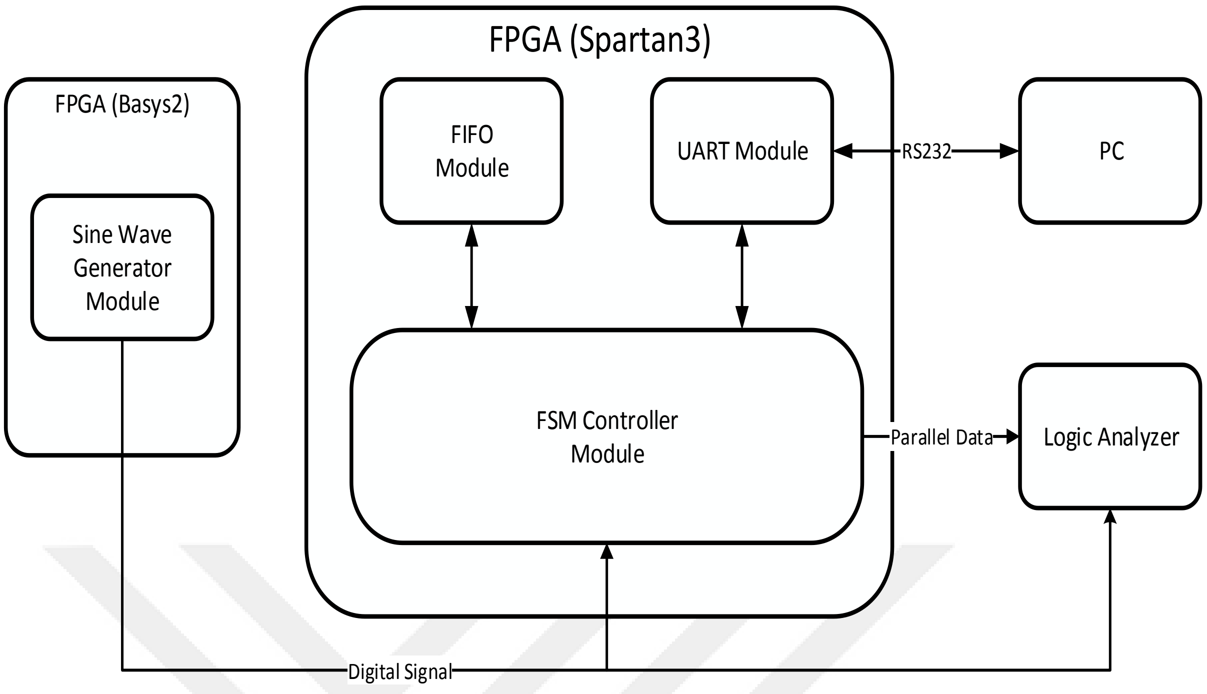


Figure 2.1: Block diagram of the test bed.

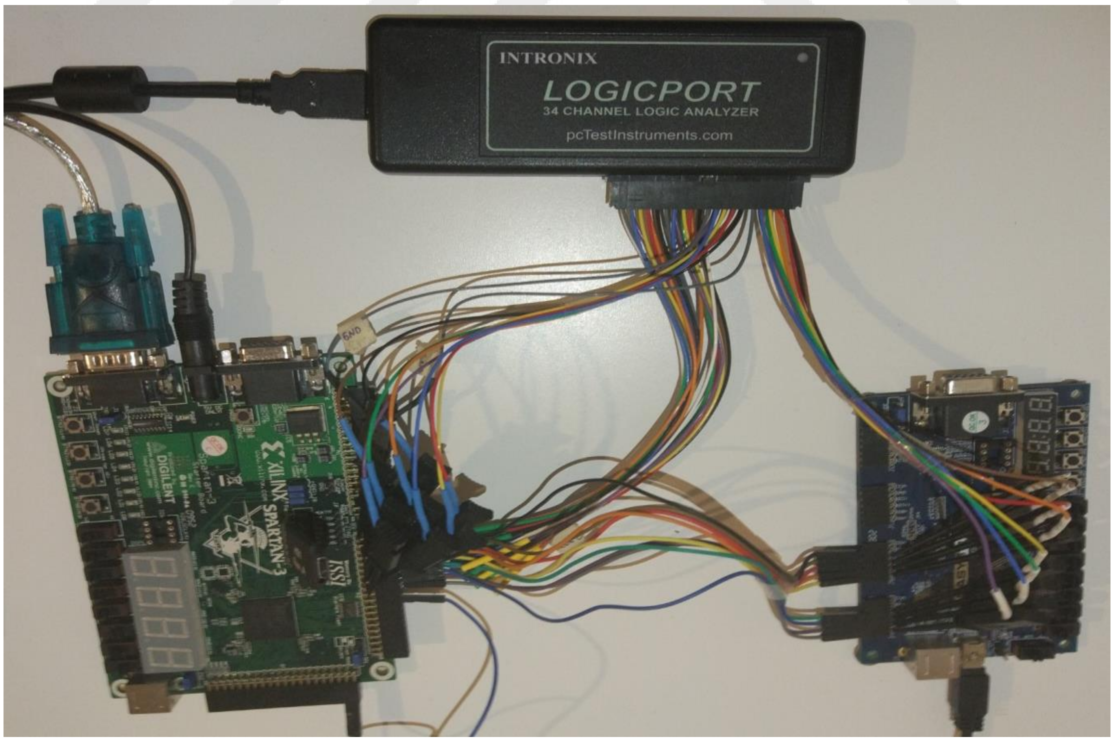


Figure 2.2: Connections of the test bed.

The block diagram and connections of test bed used during experiments is given Figure 2.1 and 2.2 respectively. 1 MHz sine wave signal generated by Basys-2 board is transferred continuously to Spartan-3 board via the 8-bit parallel port. Spartan-3 board communicates to PC via UART. According to incoming commands, Spartan-3 board write or read sine signal to FIFO or send directly at a specific rate. Many settings about these operations are sent from PC. The sine wave signal from Basys-2 board and the output signal of the Spartan-3 board are transferred to 500 MHz logic analyzer. Analyzing signals with this logic analyzer is possible. Data can be exported from the logic port program at any time. Also, data can be transferred to PC via another UART port. This part is explained at the continuation of this section.

2.2.1. Spartan-3 FPGA Board

During this thesis, Digilent’s Spartan-3 Starter Kit which includes Xilinx’s XC3S200FT256 chip is used as seen in Figure 2.3. This kit provides easy-to-use and low-cost development environment [23]. The board is preferred owing to clock rate and containing RS232 port. Additionally, it allows to construct IP FIFO. Key features of this board especially used in this project is given below:

- 200,000-gate and 4,320 logic cell equivalents
- Twelve 18K-bit block RAMs (216K bits)
- 50 MHz crystal oscillator clock source
- Four Digital Clock Managers (DCMs)
- RS-232 transceiver and 9 pin RS232 port
- Three 40-pin expansion connectors
- JTAG port for download cable

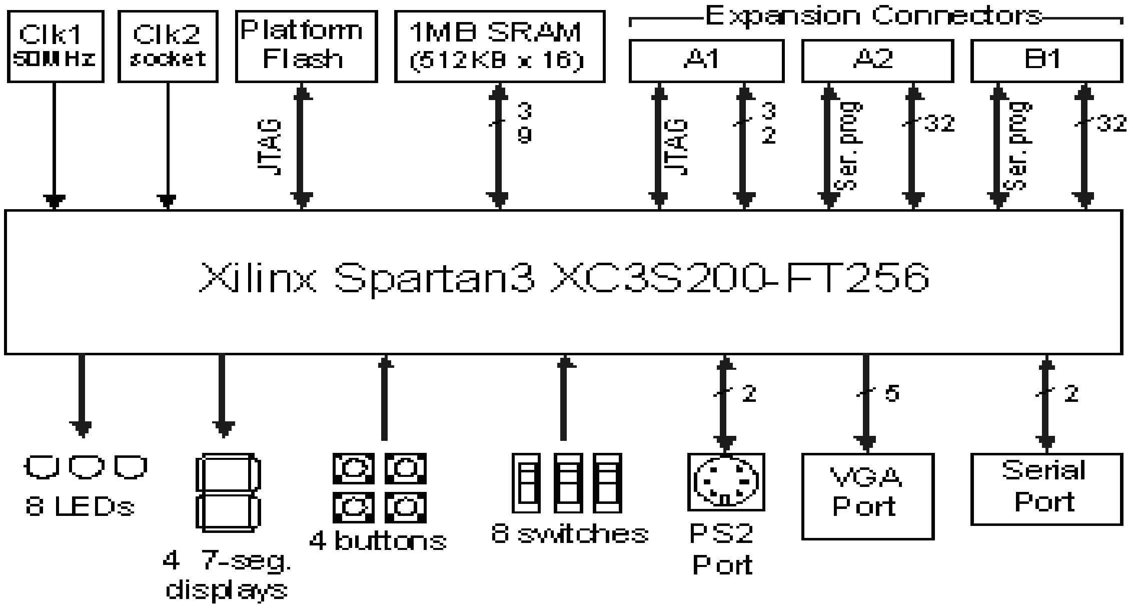
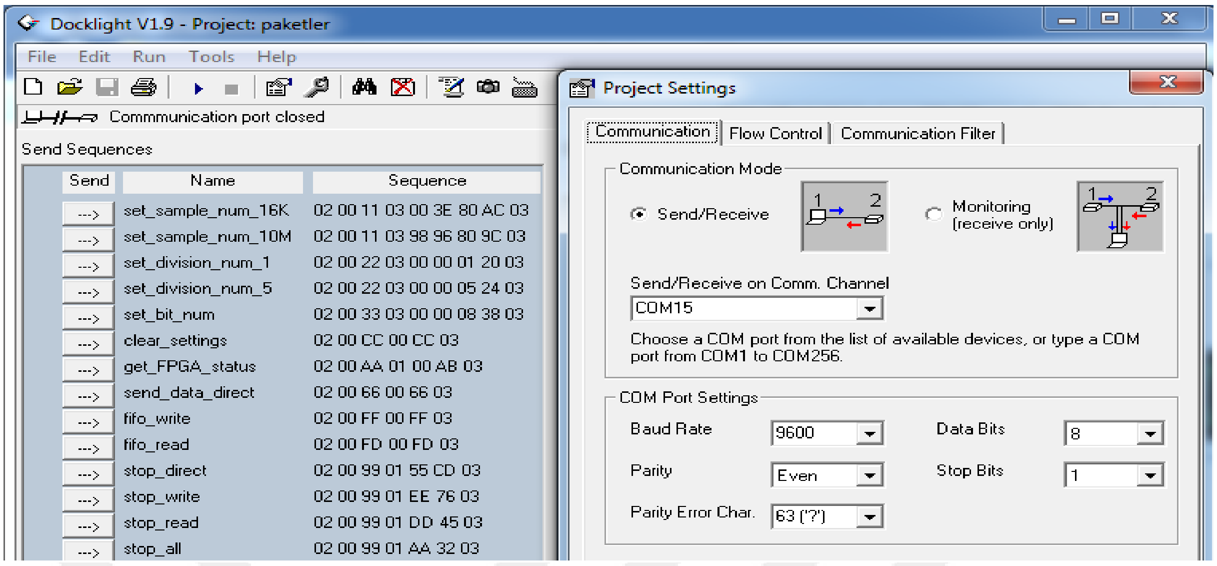


Figure 2.3: Block diagram of Spartan-3 Starter Kit Board.

2.2.2. Docklight RS232 Terminal

Docklight is the tool used during thesis for testing and simulation of RS232 serial communication protocol [24]. This interface makes testing easy throughout the project. Docklight can send sequences according to the defined packet protocol and it can respond to incoming sequences. This feature allows us to simulate the behavior of the system. It is useful especially to construct own communication packet protocol thanks to sending sequence feature. This tool also provides an option for logging all serial communication data. It can keep a huge amount of data which can be copied to a text file and can be used for plotting a graph. RS232 settings can be chosen as seen in Figure 2.4.



Commmunication port closed

Send Sequences

Send	Name	Sequence
--->	set_sample_num_16K	02 00 11 03 00 3E 80 AC 03
--->	set_sample_num_10M	02 00 11 03 98 96 80 9C 03
--->	set_division_num_1	02 00 22 03 00 00 01 20 03
--->	set_division_num_5	02 00 22 03 00 00 05 24 03
--->	set_bit_num	02 00 33 03 00 00 08 38 03
--->	clear_settings	02 00 CC 00 CC 03
--->	get_FPGA_status	02 00 AA 01 00 AB 03
--->	send_data_direct	02 00 66 00 66 03
--->	fifo_write	02 00 FF 00 FF 03
--->	fifo_read	02 00 FD 00 FD 03
--->	stop_direct	02 00 99 01 55 CD 03
--->	stop_write	02 00 99 01 EE 76 03
--->	stop_read	02 00 99 01 DD 45 03
--->	stop_all	02 00 99 01 AA 32 03

Communication Mode

[✓ Send/Receive] Monitoring (receive only)

Send/Receive on Comm. Channel: COM15

Choose a COM port from the list of available devices, or type a COM port from COM1 to COM256.

COM Port Settings

Baud Rate	9600	Data Bits	8
Parity	Even	Stop Bits	1
Parity Error Char.	63 ('?')		

Figure 2.4: Docklight RS232 Terminal.

2.2.3. Logic Analyzer Interface

The LogicPort logic analyzer, which is used during the experiment, has 34 channels and works up to 500 MHz [25]. So, it provides us to examine both 1 MHz sine wave signal generated by Basys-2 and the output signal of the Spartan-3 board. We can easily examine the data because of exporting data feature. Data can be plotted by the aid of MATLAB. The logic analyzer's hardware is powered and controlled via USB port. Some of the key features are selectable setup/hold time window and adjustable threshold.

2.3. DIGITAL DESIGN

2.3.1. DDS – Sine Wave

In many digital systems, Direct digital synthesizers (DDS) are one of the key components. The DDS IP core used in many applications to create sinusoidal waveforms [28]. The IP core contains two parts, which are available combined or individually, a SIN/COS Lookup Table and a Phase Generator. A typical technique for generating complex sinusoid is the lookup table scheme. The lookup table keeps samples of a sinusoid.

A sine wave is used for testing throughout development. This sine wave signal is constructed in Basys-2 FPGA board. Xilinx has an IP core to construct sine wave. 1 MHz sine wave is generated with using this IP core as seen in Figure 2.5 and is transferred to Spartan-3 board at 50 Msps thanks to 50 MHz clock of Basys-2 board.

The output frequency of the DDS waveform is a function of the phase increment value &, the phase width (number of bits) in the phase accumulator and the system clock frequency. The phase increment values in the range 0 to 2N-1 describes the range [0,360)°. The output frequency can be determined using:

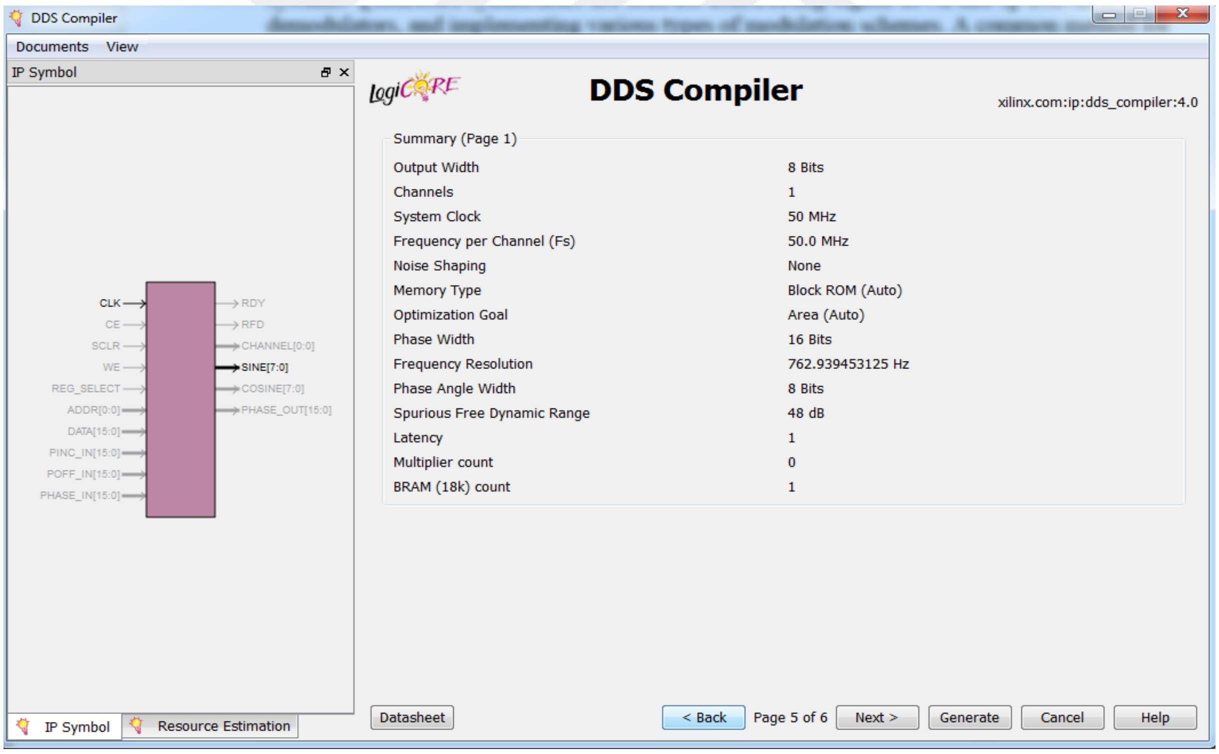
$$\text{fout} = (\text{fclk} * \&) / 2\hat{}B \tag{2.1}$$

For example, to obtain 1 MHz sine wave:

$$\text{fout} = (50\text{MHz} * 1310) / 2\hat{}16 \sim = 1\text{MHz} \tag{2.2}$$

The frequency resolution of the synthesizer is a function of the number of bits B and the clock frequency. The frequency resolution is defined by:

$$\text{fr} = \text{fclk} / 2\hat{}B = 50\text{MHz} / 2\hat{}16 = 763 \tag{2.3}$$



DDS Compiler

xilinx.com:ip:dds_compiler:4.0

Summary (Page 1)

Output Width	8 Bits
Channels	1
System Clock	50 MHz
Frequency per Channel (Fs)	50.0 MHz
Noise Shaping	None
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	16 Bits
Frequency Resolution	762.939453125 Hz
Phase Angle Width	8 Bits
Spurious Free Dynamic Range	48 dB
Latency	1
Multiplier count	0
BRAM (18k) count	1

Figure 2.5: DDS Compiler of Xilinx.

2.3.2. UART Communication Protocol

UART communication protocol is preferred for this thesis because it is common and reliable protocol. UART transmit and receive modules are designed at 9600 baud rate, 8 bits data, 1 stop bit, even parity [29]. Packet protocol, which is detailed exploration at protocol section, is developed. This protocol provides ease and flexibility to the user of the system. Number of samples, sample rate and, bit number of samples are chosen with this protocol. These choices can be changed. Data can be sent directly or write/read FIFO. The user can stop all operations or request the status at any time.

Also, some message is received during these operations. Each packet of this protocol contains start code, error, command, length (N), arguments (N-byte), checksum and, stop code. An FSM module read these bytes respectively with receiver module. This module also controls the correctness and timeout. When reading packet complete, another FSM module sends answer packet. If there is no error, the same packet is sent. If there is an error, error byte of the packet is edited according to error type and is sent. Answering packet is always same as packet except for status request packet. Answer of the status request packet is filled the empty byte with status information.

The answer is sent for all packet. This FSM module transmits bytes respectively with transmitter module as seen in Figure 2.6. This module also sends some message about successful or unsuccessful situations. In the situations of completing settings, finishing direct send or write/read FIFO, this module sends success message. In some situations such as trying to read when FIFO is empty, trying to operate before settings are not done, sending meaningless stop command and sending another request during direct send or write/read FIFO it sends unsuccess message. If there is any error the module, which gets packets, transfers the command and settings bytes to top FSM module to perform the operation requested.

Receiver and transmitter FSM modules receive and transmit start bit, 8 bits data, parity and stop bit. As receiver module transfers incoming 8 data bits to the top module, transmitter module transmits 8 data bits which it gets from top module one by one. Besides, a mod counter module is constructed to regulate the baud rate of this receiver and transmitter modules.

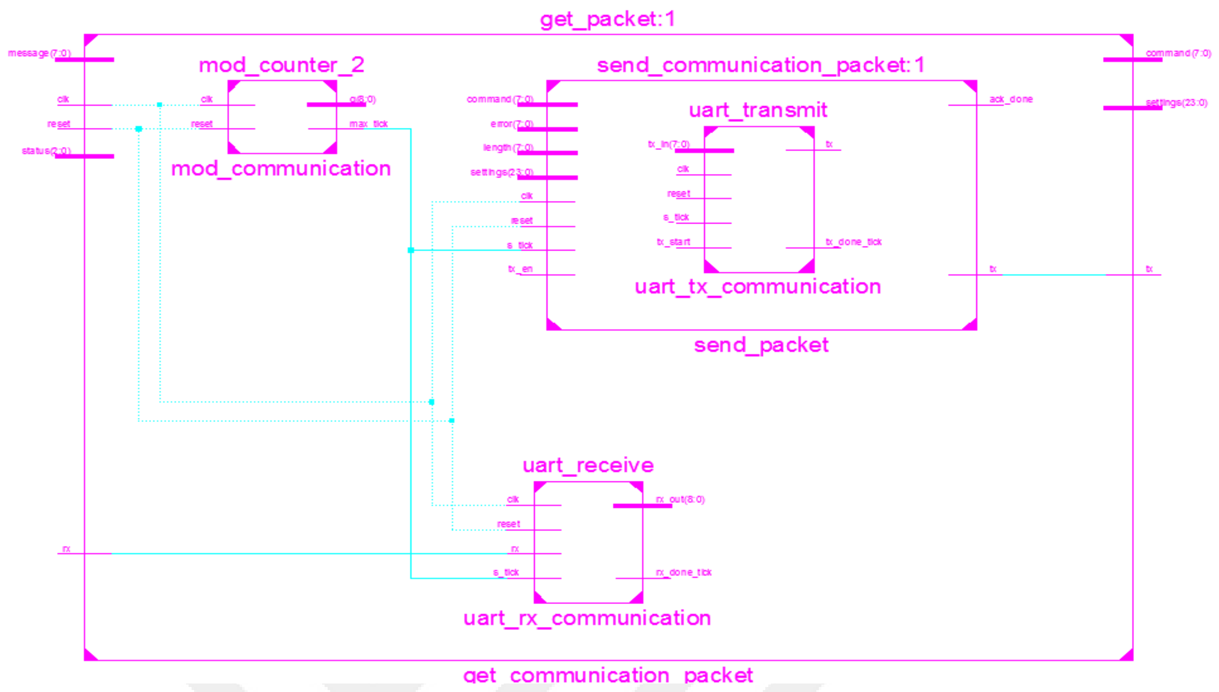


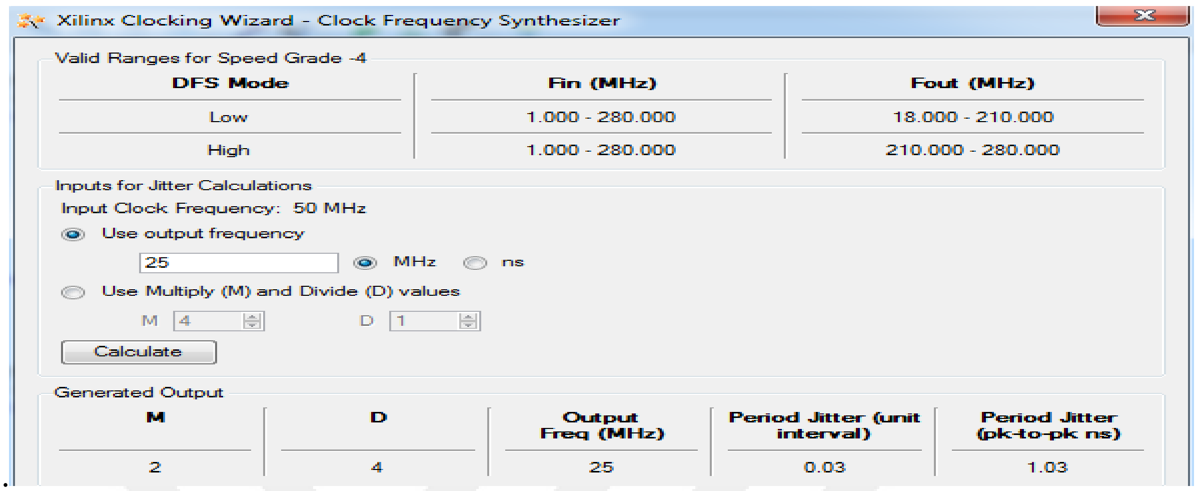
Figure 2.6: Block diagram of UART module.

2.3.3. DCM Core Generator

FPGA has a global clock distribution network to decrease the clock skew. DCMs are integrated directly into this clocking network [26]. As a result of that, DCMs solve many typical clocking problems, especially in high-frequency implementations:

- Decrease the clock skew and clock distribution delays
- Multiply or divide an input clock source
- Output clock with a 50% duty cycle
- Create a new frequency with a combined division and multiplication

Our system works at the default clock of Spartan-3 board, that is 50 MHz. Digital Clock Management IP core generator of Xilinx is used to reduce the clock rate of reading port of FIFO buffer. Read clock reduced to 25 MHz. The minimum clock rate of this DCM IP core generator is 18 MHz, as seen in Figure 2.7.



Xilinx Clocking Wizard - Clock Frequency Synthesizer

Valid Ranges for Speed Grade -4

DFS Mode	Fin (MHz)	Fout (MHz)
Low	1.000 - 280.000	18.000 - 210.000
High	1.000 - 280.000	210.000 - 280.000

Inputs for Jitter Calculations

Input Clock Frequency: 50 MHz

[✓ Use output frequency]

25 [✓ MHz] ns

Use Multiply (M) and Divide (D) values

M 4 D 1

Calculate

Generated Output

M	D	Output Freq (MHz)	Period Jitter (unit interval)	Period Jitter (pk-to-pk ns)
2	4	25	0.03	1.03

Figure 2.7: DCM core generator of Xilinx.

2.3.4. FIFO Buffer

The best way to transfer high-speed data to low-speed devices is to construct buffer. The main purpose of this dissertation is to implement a buffer on FPGA. Today's, the most common and advanced buffer type is FIFO buffer. For this purpose, we took advantage of dual-clock FIFO IP core generator of Xilinx. Furthermore, modules which control write and read port of this IP core FIFO are designed. Two D flip-flop synchronizers are placed to synchronize the input signal of this FIFO buffer. Also, DCM IP core generator of Xilinx is used for reading clock of FIFO buffer. In continuation of this section, these modules are explained in detail.

2.3.4.1. FIFO Buffer Core Generator

FIFO buffer core generator provided by Xilinx is used to obtain a more optimized FIFO buffer. At first, brief information about this core generator explained. The core supports maximum performance (up to 500 MHz) and also provides minimum resource usage [27].

The Native interface FIFO can be modified to build high-performance, area-optimized FPGA designs. Native FIFO key features:

- FIFO data widths from 1 to 1024 bits
- Selectable reset signal as asynchronous or synchronous
- Three options for memory type (block RAM, distributed RAM or built-in FIFO)

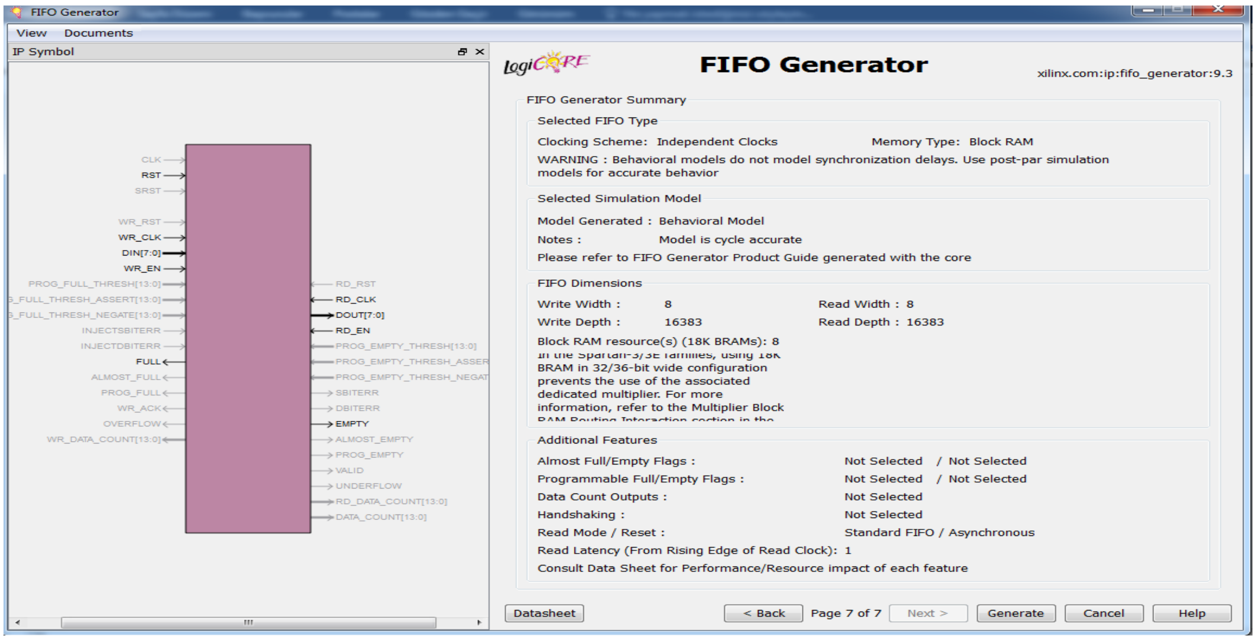
- Full, Empty, Almost Full and Almost Empty flags are available
- Programmable Full and Empty flags can be configured as constants

Independent Clocks feature lets us to choose block RAM or distributed RAM and provides independent clocks for write and read domains. Read data operations are synchronous to the read clock domain and write data operations are synchronous to the write clock domain.

2.3.4.2. FIFO Usage and Control

Write Operation: If write enable signal is set, data is appended to the FIFO from the input port. But write operations are only available if the FIFO is not full. If there is a new write operation request while the FIFO is full, the request is discarded and the overflow flag is asserted. The full flag shows that no more writes can be operated until read operation performed. The full status flag is synchronous with the write clock domain.

Read Operation: If read enable signal is set, data is transferred to the output port from the FIFO. But read operations are only available if the FIFO is not empty. If there is a new read operation request while the FIFO is empty, the request is discarded and the underflow flag is asserted. The empty flag shows that no more reads can be operated until write operation performed. The empty status flag is synchronous with the read clock domain.



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

FIFO Generator Summary

Selected FIFO Type

Clocking Scheme: Independent Clocks   Memory Type: Block RAM

WARNING : Behavioral models do not model synchronization delays. Use post-par simulation models for accurate behavior

Selected Simulation Model

Model Generated : Behavioral Model

Notes : Model is cycle accurate

Please refer to FIFO Generator Product Guide generated with the core

FIFO Dimensions

Write Width :	8	Read Width :	8
Write Depth :	16383	Read Depth :	16383

Block RAM resource(s) (18K BRAMs): 8
In the Spartan-3/3E families, using 18K
BRAM in 32/36-bit wide configuration
prevents the use of the associated
dedicated multiplier. For more
information, refer to the Multiplier Block
RAM Routing Interaction section in the

Additional Features

Almost Full/Empty Flags :	Not Selected / Not Selected
Programmable Full/Empty Flags :	Not Selected / Not Selected
Data Count Outputs :	Not Selected
Handshaking :	Not Selected
Read Mode / Reset :	Standard FIFO / Asynchronous
Read Latency (From Rising Edge of Read Clock):	1

Consult Data Sheet for Performance/Resource impact of each feature

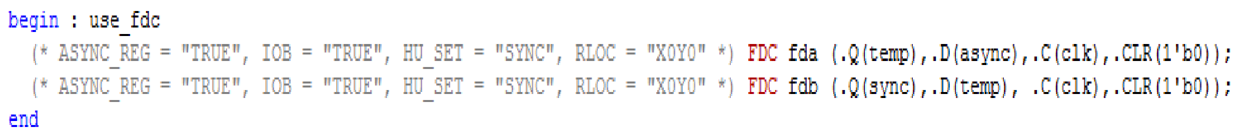
Figure 2.8: FIFO buffer core generator of Xilinx.

In FIFO part, FIFO types, architecture construct problems and solutions are expressed in detail. For this project, a standard dual-clock FIFO which has 8 bit write/read word width and 16384 depth (Spartan-3 board could support maximum this depth) is constructed shown in Figure 2.8. Write and read clock rate of FIFO buffer are different from each other. Furthermore, empty and full flags are enabled to achieve more reliable system.

2.3.5 Synchronizer

In metastability section, constructing synchronizer is explained exhaustively. The most efficient way is the signal to pass to D flip-flop. Thus, MTBF of the system considerably increases. In this project, a module gets each bit of asynchronous signal passed through two D flip-flops to synchronize the signal. The most important point mentioned before is to accommodate this two flip-flop next to each other on the FPGA chip. Otherwise, clock skew may occur between these two flip-flops. There are some mapping and placement constraints as seen in Figure 2.9 to accommodate these two flip-flops next to each other. Some information about these constraints is below:

- ASYNC_REG: Provides timing constraint improvements in simulation for asynchronous data. It deactivates 'X' propagation throughout timing simulation. When timing violation occurs, the previous value is preserved on the output instead of going unknown state.
- IOB: It is a primary synthesis and mapping constraint. It specifies which latches and flip-flops accommodate into the IOB. This constraint tells the mapper to move the instance into an IOB type component if applicable. Where; TRUE indicates the latch or flip-flop to be moved into an IOB and FALSE indicates vice versa.
- HU_SET: This constraint, used for advanced mapping, is described by the design hierarchy. On the other hand, it allows us to define a set name, so we can define different HU_SET sets by defining set names.
- RLOC: Relative location is a primary placement, synthesis and mapping constraint. RLOC constraints group elements and allow us to specify the position of an element within the other element in the same set, regardless of final placement in the total design.



```
begin : use_fdc
  (* ASYNC_REG = "TRUE", IOB = "TRUE", HU_SET = "SYNC", RLOC = "X0Y0" *) FDC fda (.Q(temp),.D(async),.C(clk),.CLR(1'b0));
  (* ASYNC_REG = "TRUE", IOB = "TRUE", HU_SET = "SYNC", RLOC = "X0Y0" *) FDC fdb (.Q(sync),.D(temp), .C(clk),.CLR(1'b0));
end
```

Figure 2.9: Mapping and placement constraints.

In controlling FIFO module as seen in Figure 2.10, firstly incoming asynchronous data is synchronized. This synchronous data is written to FIFO buffer when the write signal is enabled. Analogously, when the read signal is enabled, data is read and transferred to the parallel output port. Two modules are designed to control these write and read enable signals. Write control module performs the intended number of data to write to FIFO. When writing operations complete, this module asserts write success message signal. Similarly, read control module performs the intended number of data to read from FIFO and when reading operations complete asserts read success message signal. If FIFO is empty, it asserts read unsuccess message signal.

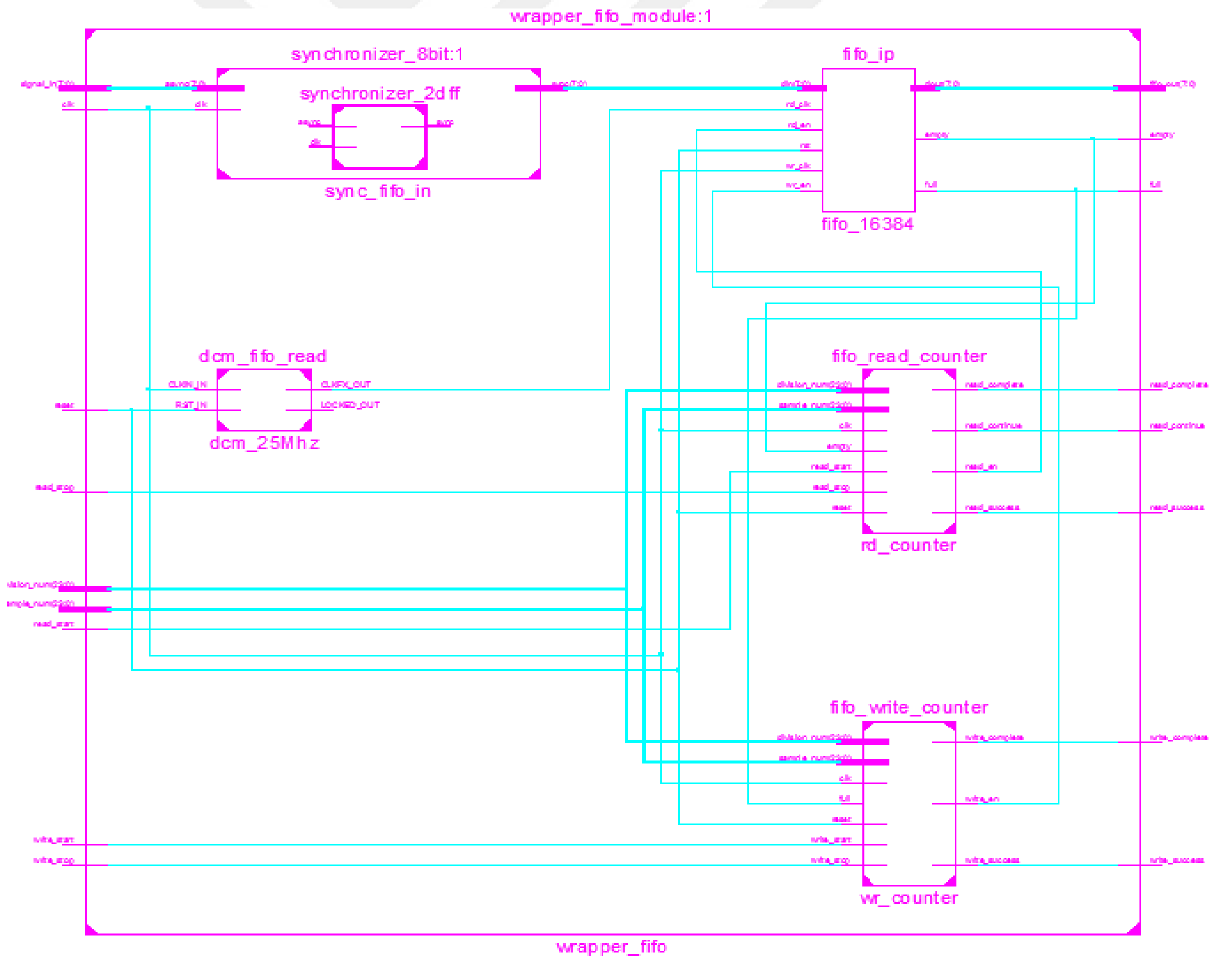


Figure 2.10: Block diagram of FIFO module.

Submodules of this project given in Figure 2.12 are explained in detail by now. The top module shown in Figure 2.11, performs the operations according to command and argument bytes received from an external device via UART. Settings are inferred at first, then according to intended operation sending direct, writing or reading FIFO is performed. Additionally, these operations can be stopped at any time. The data which are transferred to the logic analyzer via a parallel output port, are also transferred to PC via another UART port.

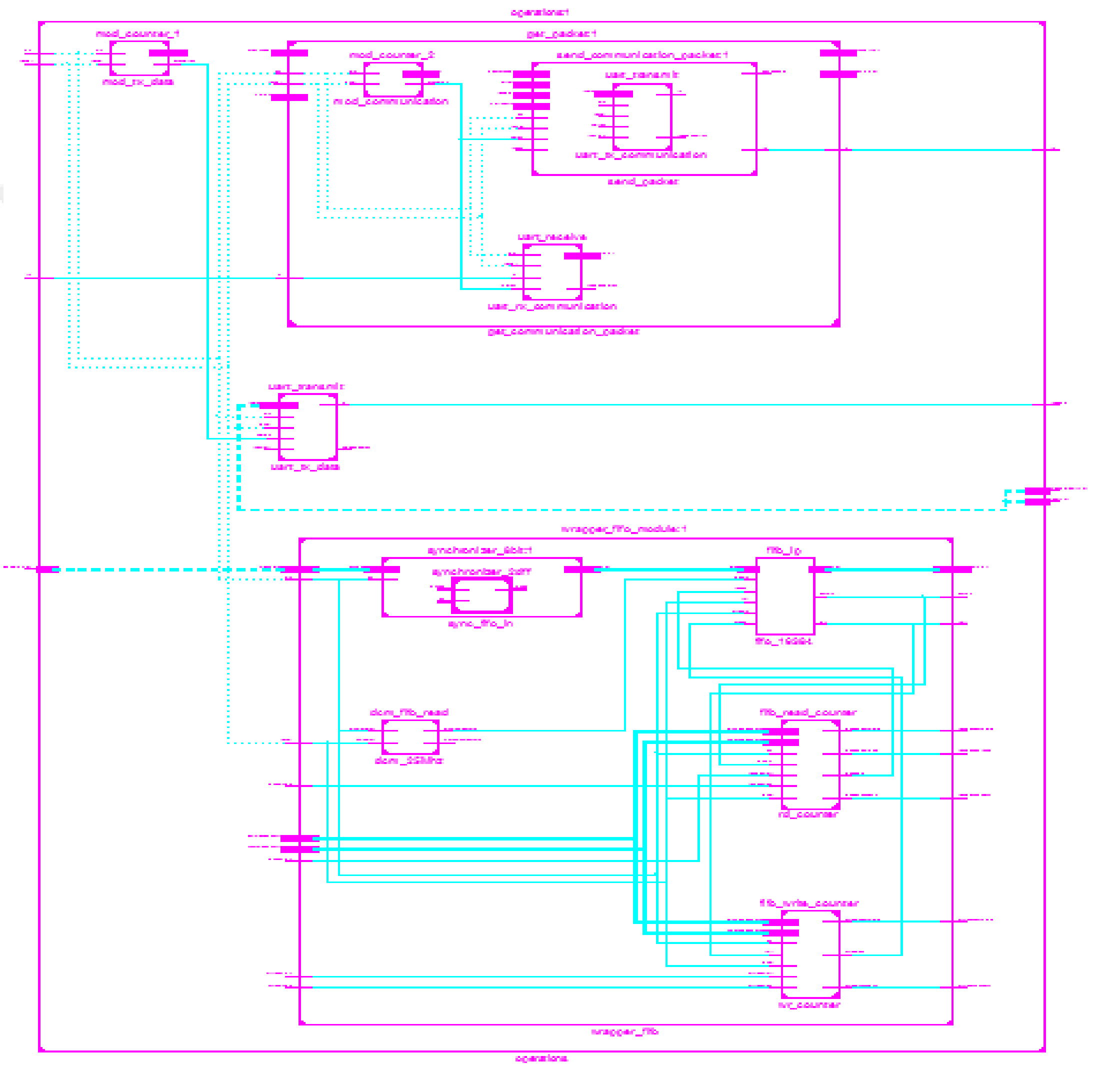
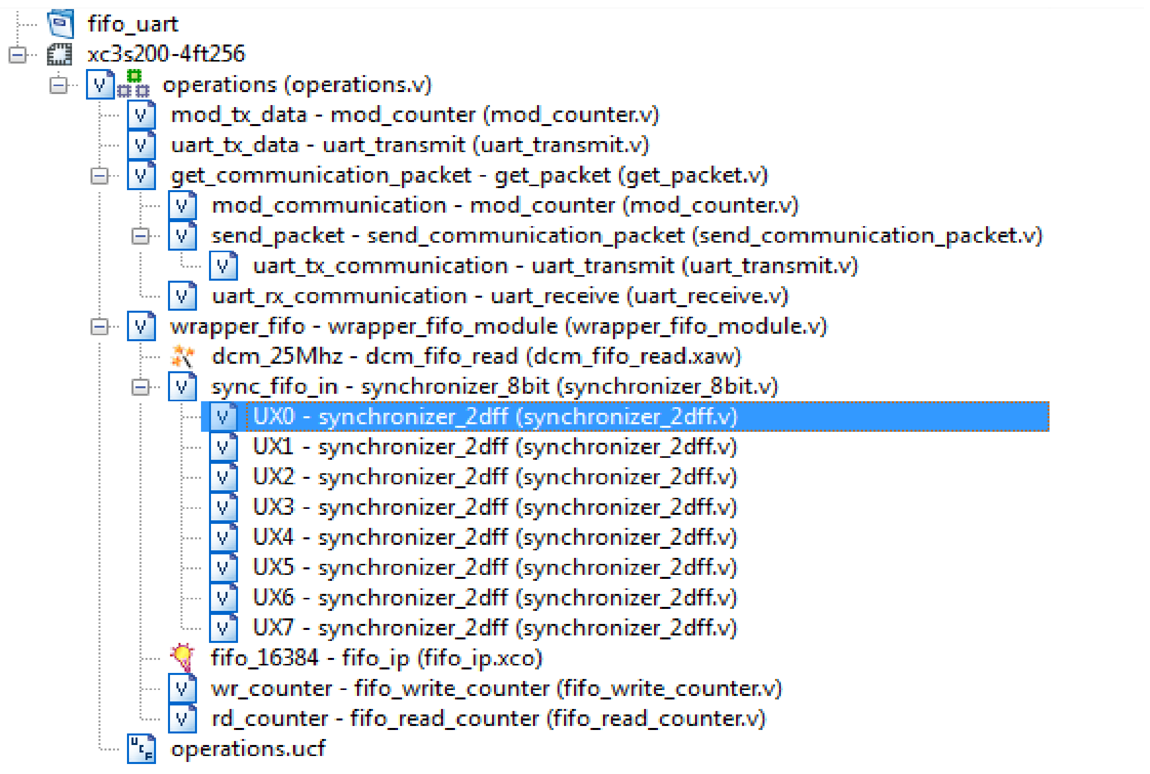


Figure 2.11: Block diagram of the top module.



```
fifo_uart
xc3s200-4ft256
    operations (operations.v)
        mod_tx_data - mod_counter (mod_counter.v)
        uart_tx_data - uart_transmit (uart_transmit.v)
        get_communication_packet - get_packet (get_packet.v)
            mod_communication - mod_counter (mod_counter.v)
            send_packet - send_communication_packet (send_communication_packet.v)
                uart_tx_communication - uart_transmit (uart_transmit.v)
            uart_rx_communication - uart_receive (uart_receive.v)
        wrapper_fifo - wrapper_fifo_module (wrapper_fifo_module.v)
            dcm_25Mhz - dcm_fifo_read (dcm_fifo_read.xaw)
            sync_fifo_in - synchronizer_8bit (synchronizer_8bit.v)
                UX0 - synchronizer_2dff (synchronizer_2dff.v)
                UX1 - synchronizer_2dff (synchronizer_2dff.v)
                UX2 - synchronizer_2dff (synchronizer_2dff.v)
                UX3 - synchronizer_2dff (synchronizer_2dff.v)
                UX4 - synchronizer_2dff (synchronizer_2dff.v)
                UX5 - synchronizer_2dff (synchronizer_2dff.v)
                UX6 - synchronizer_2dff (synchronizer_2dff.v)
                UX7 - synchronizer_2dff (synchronizer_2dff.v)
            fifo_16384 - fifo_ip (fifo_ip.xco)
            wr_counter - fifo_write_counter (fifo_write_counter.v)
            rd_counter - fifo_read_counter (fifo_read_counter.v)
        operations.ucf
```

Figure 2.12: List of all modules.

3. RESULTS

As mentioned earlier, sine wave signal generated by Basys-2 FPGA board has been used as a high-speed input signal. The graph of this sine wave sampled by the high-speed logic analyzer is given in Figure 3.1.

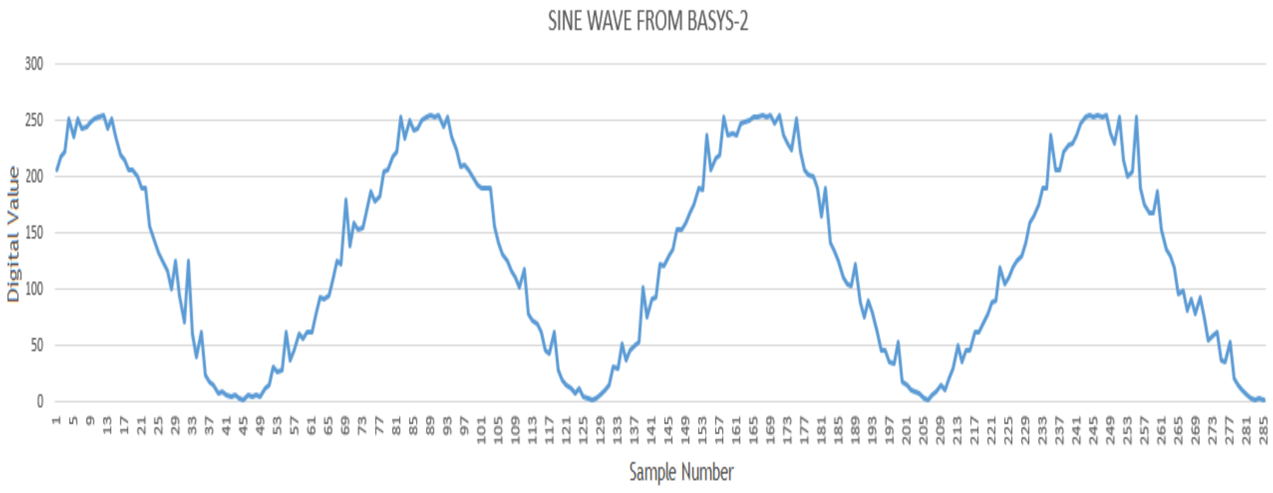


Figure 3.1: The graph of sine wave signal generated by Basys-2 FPGA board.

Sine wave signal generated by Basys-2 FPGA board is written to FIFO and read from FIFO. The graph of this output data of FIFO buffer sampled by the logic analyzer is given in Figure 3.2. This graph demonstrates the accuracy of our system.

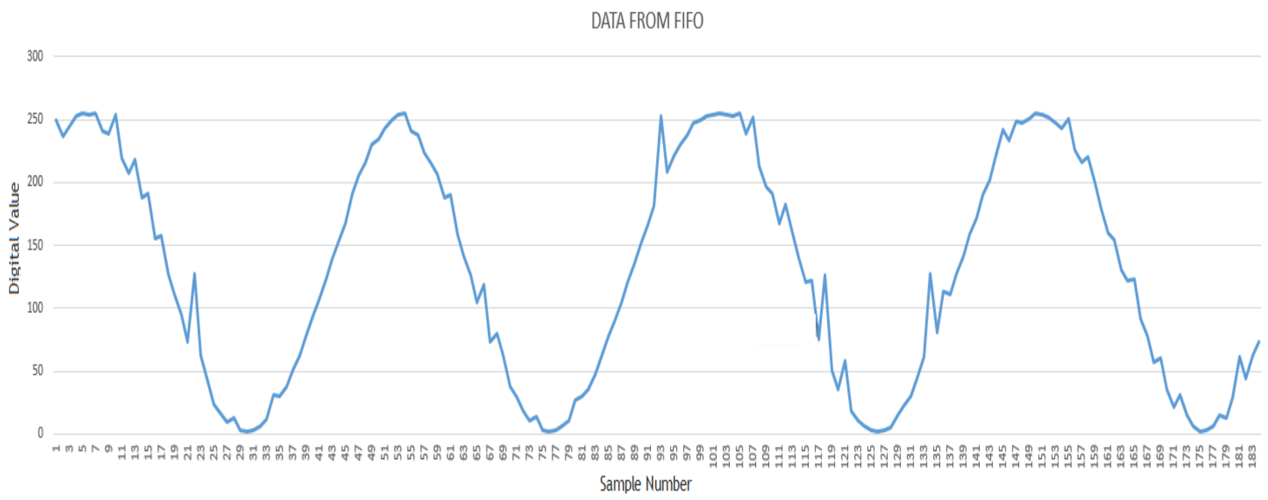


Figure 3.2: The graph of data read from FIFO buffer.

Sine wave signal generated by Basys-2 FPGA board can be sent directly at different division value to an external device via parallel port. The graph of this output data at the division as 1 sampled by the logic analyzer is given in Figure 3.3. This graph also demonstrates the accuracy of our system.

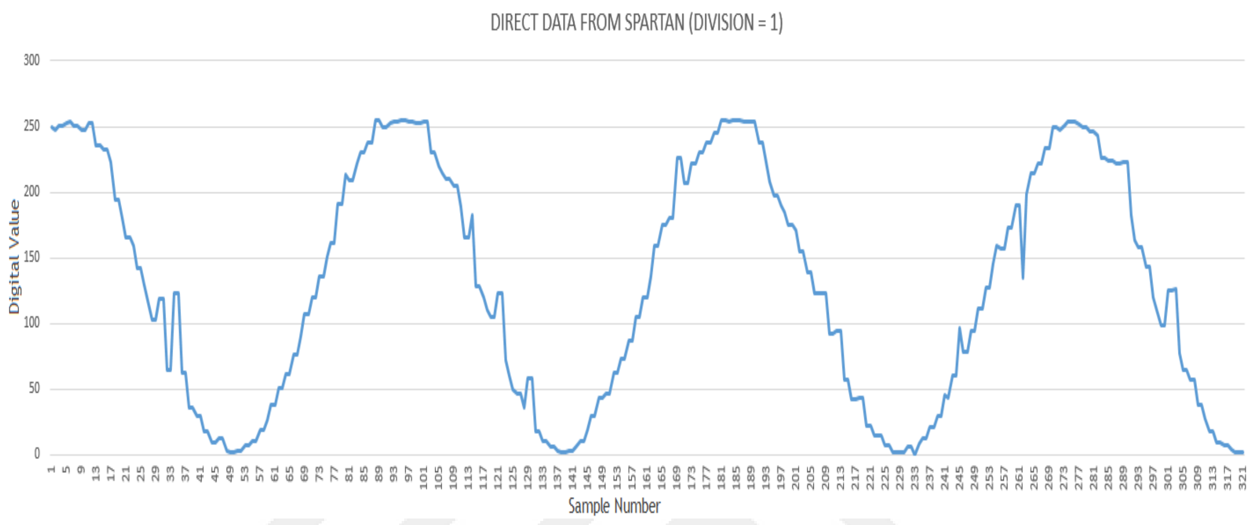


Figure 3.3: The graph of data sent directly at division =1.

The graph of another output data at the division as 10 sampled by the logic analyzer is given in Figure 3.4.

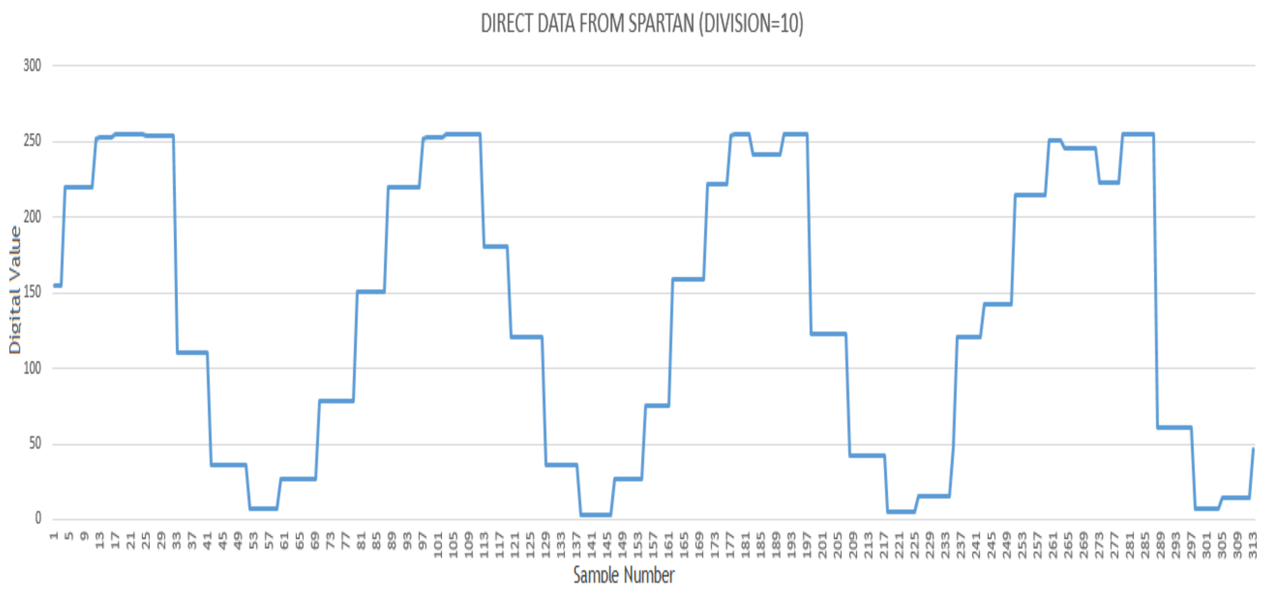


Figure 3.4: The graph of data sent directly at division =10

4. DISCUSSION

This system transfers data from the high-speed device to low-speed device. As it was mentioned in introduction chapter, this system is required for a project which purposes to calculate the delay time between input and output signals of SAW biosensors. These signals are sampled by high-speed ADC. There will be data loss while transferring data from high-speed ADC to low-speed devices like PC, ARM processor because of their differences of clock speed.

We have analyzed many products which transfer data from high-speed to low-speed, on the marketplace. The common solution we have observed in this research is to construct FIFO buffer. Since one side of this dual-clock FIFO buffer must work at high-speed, it is a good way to construct this on FPGA. So, we designed multi-clock FIFO buffer on FPGA. This FIFO buffer saves data at 50 MSPS from an external device and sends it at low speed to PC. Although the system is designed for this project, it also can be used for any project which needs to transfer data from high-speed to low-speed.

System Features:

- 16 kB FIFO depth with an 8-bit word width
- Works with single channel ADC
- Parallel input at 50 Msps
- Simple UART interface
- Based on Spartan-3 FPGA

On the market, there are comparatively better products whose FIFO depth that reaches to MBs or parallel input speed that reaches at GSPSs. Using Spartan-3 board during experiments has restricted the development of our project. Since Spartan-3 board has 50 MHz clock source, parallel input of our system works at 50 MSPS.

Similarly, because this board has 200K system gates, FIFO depth of our system is 16 kB with 8-bit width. For instance, if we had used Spartan-6 board, we would have had a system with parallel input at 100 MSPS or higher FIFO depth owing to 1M system gates of this board. These features could have been considerably improved by especially Virtex-7 FPGA families which have 200 MHz system clock and 2M system gates.

If our system had worked with multi-channel ADCs, but FIFO depth would not have reached even to 16kB. Therefore, we decided that single-channel was sufficient. Besides, UART was chosen for communication between FPGA and PC. This communication became more reliable thanks to packet protocol created by us. A comprehensive interface was built. Thus, the user can make settings (sample number, division rate or bit number), send command (send data, write to FIFO, read from FIFO, stop operations, clear settings) or request status. Owing to the interface, using this system became easy and efficient.

5. CONCLUSION AND RECOMMENDATIONS

Metastability and synchronization problems are explained in detail at first in this dissertation. Then a comprehensive exploration of first-in first-out buffer and dual-clock FIFO buffer approaches are covered. Afterwards, Universal Asynchronous Receiver Transmitter serial communication protocol, which provides to communicate between FPGA and PC, and the packet protocol, which is created by us – used during this communication are briefly mentioned. In hardware and verification section, all techniques, environments, and devices used throughout this thesis are expressed.

Purpose of this thesis is to transfer the input and output signal of a biosensor sampled by high-speed analog to digital converter to PC at low-speed. However, this system can be used as an interface to transmission between different clock domains. In this interface, the users can choose number of samples, sample rate and bit number of samples.

In this dissertation, the issue of data transmission from high-speed domain to low-speed domain is solved by a dual-clock FIFO buffer. However, data transmission via this FIFO buffer or sending directly without FIFO buffer are optional. Some features such as requesting status, stopping operation are provided to make the system more reliable. For obtaining more effective results, IP core generators of Xilinx are used for constructing FIFO and digital clock management modules.

The communication protocol is chosen as Universal Asynchronous Receiver Transmitter to provide convenience to users. Additionally, a strong packet protocol is created to understand system requests correctly, to make sure that data transfer is reliable and to infer errors easily with error and status flags.

High-speed sine wave signal generated by another FPGA board has been used as high-speed data throughout experiments. This sine wave signal as input signal and the output signal of our system are analyzed with a logic analyzer. Then these recorded data of input and output signals are plotted graphics with MATLAB. So, the correctness of our digital design system is demonstrated obviously.

In the future, the work in this dissertation can be extended in some aspects;

- If FPGA boards with sufficient features are provided, the system can be used for transferring data at much higher speeds.
- According to the purpose of projects, data can be processed on FPGA board instead of transferring to the external device. So, only FPGA board can realize all issue.
- As a microprocessor is constructed on FPGA board, data can be processed in this microprocessor instead of PC. Since processing data in FPGA is difficult, the microprocessor makes it easier.

REFERENCES

[1] Shaochun G., 2011, Realization of Replicated Streaming Applications on Multi-Clocked FPGAs, Thesis (M.Sc.), School of Information and Communication Technology, Royal Institute of Technology.

[2] Roth F., 2011, Using Low Cost FPGAs for Realtime Video Processing, Thesis (M.Sc.), Masaryk Unıversity, Faculty of Informatics.

[3] Yu S., Yi L., Chen W., Wen Z., 2007, Implementation of a Multi-channel UART Controller Based on FIFO Technique and FPGA, Second IEEE Conference on Industrial Electronics and Applications, 2007, China.

[4] Yağlıkçı A. G., 2014, FPGA Tabanlı Sayısal Sinyal İşleme Algoritmalarına Özelleştirilmiş Yardımcı İşlemci Tasarımı, Thesis (M.Sc.), Institute of Science, TOBB University.

[5] Semeraro G., Magklis G., Balasubramonian R., Albonesi D. H., Dwarkadas S., Scott M. L., 2002, Energy-efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling, International Symposium on High-Performance Computer Architecture, 2002 New York, USA.

[6] Rabaey J. M., Chandrakasan A., Nikolic B., 2003, Digital Integrated Circuits a Design Perspective, Semiconductor Memories.

[7] Ho R., Mai K., Horowitz M., 2001, The Future of Wires, Proceedings of the IEEE.

[8] Zhang Y., Yi C., Wang J., 2011, Asynchronous FIFO implementation using FPGA, International Conference on Electronics and Optoelectronics (ICEOE), 2011 China.

[9] Chu P. P., 2008, FPGA Prototyping by Verilog Examples, John Wiley & Sons, Inc., New Jersey, USA.

[10] Taskin B., 2005, Advanced Timing and Synchronization Methodologies for Digital VLSI Integrated Curciuts, Thesis (PhD), School of Engıneering, University of Pittsburgh.

[11] Kulmala A., Hämäläinen T. D., Hännikäinen M., 2006, Reliable GALS Implementation of MPEG-4 Encoder with Mixed Clock FIFO on Standard FPGA, Tampere University of Technology, Institute of Digital and Computer Systems, Tampere, Finland.

[12] Cummings C. E., 2005, Synthesis and Scripting Techniques for Designing Multi Asynchronous Clock Designs, SNUG-2001 San Jose.

[13] Apperson R. W., Yu Z., Meeuwsen M. J., Mohsenin T., Baas B. M., 2007, A Scalable Dual-Clock FIFO for Data Transfers Between Arbitrary and Haltable Clock Domains, IEEE Transactions on VLSI Systems.

[14] Apperson R. W., 2002, A Dual-Clock FIFO for the Reliable Transfer of High-Throughput Data Between Unrelated Clock Domains, Thesis (M.Sc.), University of Washington.

[15] Haritha, Raju D. P., 2015, Implementation of multichannel UART with FIFO, International Journal of Advanced Research in Science and Technology.

[16] Chelcea T., Nowick S. M., 2000, A Low–Latency FIFO for Mixed–Clock Systems, Department of Computer Science Columbia University.

[17] Cummings C. E., 2005, Simulation and Synthesis Techniques for Asynchronous FIFO Design, SNUG-2002 San Jose.

[18] Cummings C. E., Alfke P., 2005, Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons, SNUG-2002 San Jose.

[19] Xiao L., Chen X., Lin B., 2013, Design and Realization of Strain Measurement System Based on FPGA and ARM, Fourth International Conference on Intelligent Control and Information Processing (ICICIP), 2013 China.

[20] Budzin G., 2011, Programmable Logic Design, Wroclaw University of Technology.

[21] Forstner P., 1999, FIFO Architecture, Functions, and Applications, Texas Instruments.

[22] Nebhrajani V. A., 2009, Asynchronous FIFO Architectures.

[23] Xilinx Inc., 2013, Spartan-3 FPGA Family Data Sheet, https://www.xilinx.com/support/documentation/data_sheets/ds099.pdf, [3 May 2019].

[24]. Flachmann und Heggelbacher GbR, 2016, Docklight V2.2 User Manual, https://docklight.de/pdf/docklight_manual.pdf , [3 May 2019].

[25]. Intronix Test Instruments Inc., 2015 http://www.pctestinstruments.com/downloads/la1034_brochure_en.pdf, [3 May 2019].

[26]. Xilinx Inc., 2009, Digital Clock Manager (DCM) Module, https://www.xilinx.com/support/documentation/ip_documentation/dcm_module.pdf, [3 May 2019].

[27]. Xilinx Inc., 2011, LogiCORE IP FIFO Generator v8.1 User Guide, https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator_ug175.pdf, [3 May 2019].

[28]. Xilinx Inc., 2011, LogiCORE IP DDS Compiler v4.0, https://www.xilinx.com/support/documentation/ip_documentation/dds_ds558.pdf, [3 May 2019].

[29]. https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter, [3 May 2019].

CURRICULUM VITAE

Personal Information	
Name Surname	Selma UÇAR AKDENİZ
Place of Birth	Adana
Date of Birth	02.01 1991
Nationality	☑ T.C. ❒ Other:
Phone Number	0532 697 9426
Email	selma@ucarakdeniz.com
Web Page	

Educational Information	
B. Sc.	
University	Fatih University
Faculty	Faculty of Engineering
Department	Department of Electrical and Electronic Engineering
Graduation Year	18.06.2013

M. Sc.	
University	Istanbul University-Cerrahpasa
Institute	Institute of Graduate Studies
Department	Department of Electrical and Electronic Engineering
Programme	Electrical and Electronic Engineering Programme