ARL-TN-1239 ● FEB 2025

A Simple Time–Distance Measurement System Using a Field-Programmable Gate Array

by John Schill, Patrick Sykes, Naghmeh Karimi, Hasin I. Reefat, Toufiq H. Anik, and Hossein Pourmehrani

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-1239 ● FEB 2025

A Simple Time–Distance Measurement System Using a Field-Programmable Gate Array

John Schill and Patrick Sykes
DEVCOM Army Research Laboratory

Naghmeh Karimi, Hasin I. Reefat, Toufiq H. Anik, and Hossein Pourmehrani
University of Maryland, Baltimore County

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
February 2025	Technical Note	06/01/2023	12/31/2024

4. TITLE AND SUBTITLE
A Simple Time–Distance Measurement System Using a Field-Programmable Gate Array

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
John Schill, Patrick Sykes, Naghmeh Karimi, Hasin I. Reefat, Toufiq H. Anik, and Hossein Pourmehrani

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLA-PC
Adelphi, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-1239

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
We describe a time and distance measurement system that is based on a Xilinx KCU105 field-programmable gate array (FPGA) evaluation board using a MicroBlaze processor on the FPGA's fabric. All input/output is handled by the evaluation board. External to the evaluation board, we used a 1550-nm distributed-feedback laser with a Mach–Zehnder modulator and a seven-fiber bundle used for transmitting and receiving the modulated light. The bundle was designed in-house and is described in this report. After a discussion of the basic block diagram of the system, we describe each component in more detail. We have two different phase measurement schemes that are analyzed and discussed. Finally, we present some of the initial testing and results we have achieved.

15. SUBJECT TERMS
distance/time measurement; accurate positioning; field-programmable gate array; FPGA; optical fiber bundle; phase management; Photonics, Electronics, and Quantum Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	33

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
John Schill	(301) 394-1995

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. RF System Block Diagram and Fiber Bundle

A block diagram of the overall system is shown in Figure 1. The Xilinx KCU105 field-programmable gate array (FPGA) evaluation board will be described in more detail in Section 2. There are two different versions of the system, one at U.S. Army Combat Capabilities Development Command Army Research Laboratory and the other at University of Maryland, Baltimore County (UMBC). These systems are logically the same in that they basically have the same modules and result. The UMBC system will be described in more detail in Section 3.

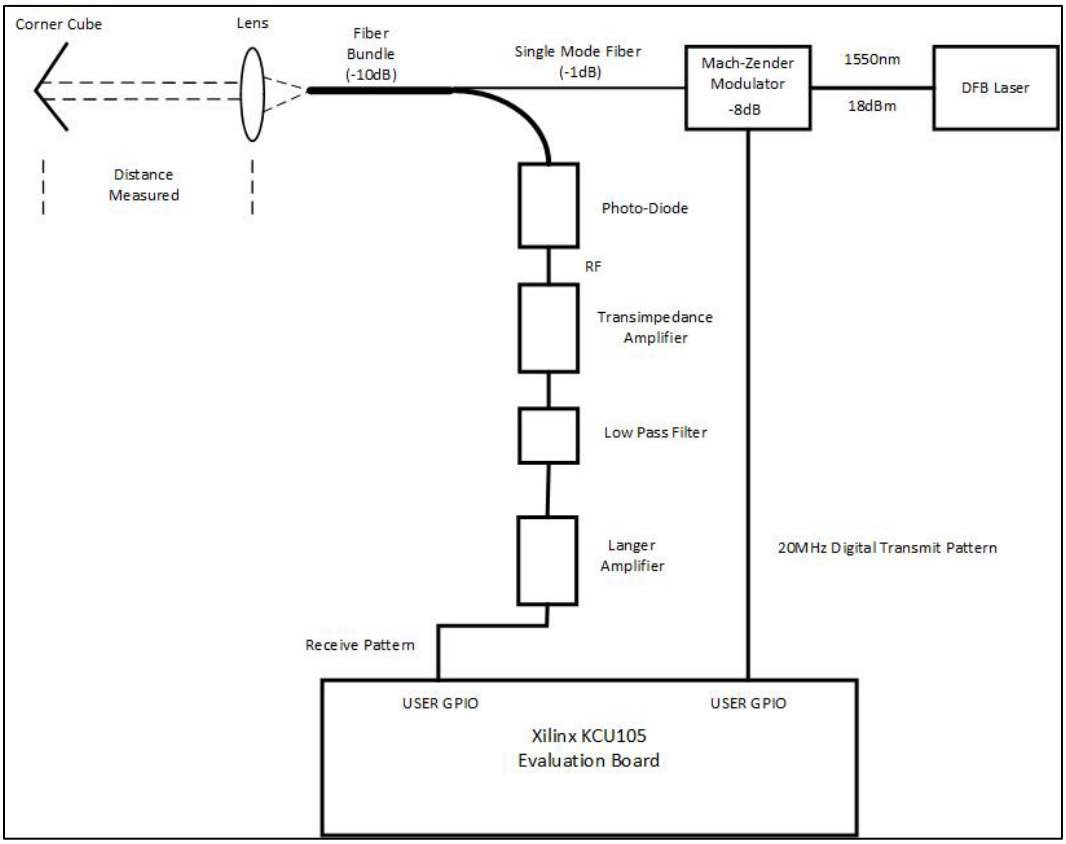

Figure 1. Block diagram of transmit/receive system.

A digital pattern is created within the evaluation board. This pattern is presented at a user general purpose input/output (GPIO) port. A distributed-feedback laser operating in continuous wave (CW) mode at 1550 nm (actually channel 33 of the International Telecommunication Union grid: 1550.92 nm) is on the right of the Figure 1 diagram. Its output level is approximately 18 dBm. The CW light from this laser is fed into a Mach–Zehnder modulator that amplitude modulates the digital signal onto the light. The insertion loss is approximately 8 dB. This modulator is one of several we use, so the make and model are unimportant. This signal is then sent via a single mode fiber (part of the fiber bundle) to the output

collimating lens. This bundle will be described in more detail in the following section. The lens does a reasonably good job of collimating the beam. We measured this double angle divergence to be 0.554 mRad. After the light travels to a corner cube and returns to the lens it is collected by the other fibers of the bundle and sent to a photodiode (Thorlabs SM05PD4A 900–1700-nm response) that demodulates the light back to a low-amplitude digital signal. This signal is amplified by a transimpedances amplifier (Thorlabs AMP145) with 100-MHz bandwidth (BW) and 2.5-kV/A gain. The signal then passes through a 44-MHz low-pass filter that further reduces the noise. Next, this signal is sent to an amplifier (Langer EMV-Technik PA 306 SubMiniature version A [SMA]) with 6-GHz BW and 30-dB gain. This amplifer acts somewhat like a comparator that goes rail-to-rail with the incoming signal. The output is similar to a digital square wave with rise time of <1 ns.

The signal level required by the FPGA to detect accurate ones and zeros at the receiving user GPIO is between 0.5 and 1.1 V. In other words, the FPGA needs less than 0.5 V to recognize a zero (0) and above 1 V to recognize a one (1). These values are somewhat arbitrary, but with testing we determined that there is some uncertainty near 0.7 V; thus by staying below 0.5 V and above 1.1 V, we are certain of the signals received. Although not shown, we achieve this signal level with bulk attenuators and a bias-T placed in the system to adjust the signal levels as needed. This is something that we plan to automate in the future by adding automatic gain and level controls.

The fiber bundle is a group of seven optical fibers. A diagram is shown in Figure 2. Six multimode fibers surround one single-mode fiber. The multimode fibers have a core diameter of 127 μm while the single-mode fiber has a core diameter of 9 μm. These fibers are placed inside a tube of heat-shrink material, filled with epoxy, and then heated to pull the structure together. After the epoxy is cured, the ends are polished to optical quality. As you can see in Figure 1, the single-mode fiber is pulled out from the bundle on the transmit side so it can be connectorized and attached to the Mach–Zehnder modulator.

To reiterate, both multifiber sides of the bundle are wrapped in heat-shrink and epoxy and polished. On the receiving side, the bundle is placed in a fixture with adjustment screws. This fixture allows the bundle to be positioned in front of the photodiode for maximum received signal strength.

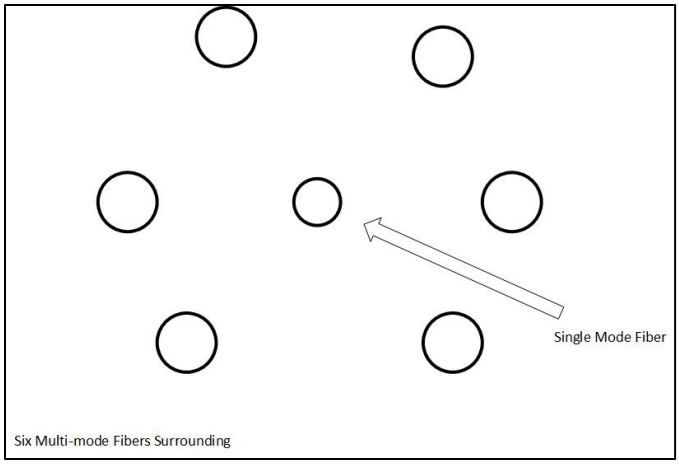

Figure 2. Conceptual diagram of the fiber bundle.

2. KCU105 Evaluation Board Block Diagram

The KCU105 evaluation board block diagram is shown in Figure 3. The MicroBlaze is a reduced instruction set computer soft processor that is implemented on the FPGA fabric. This computer is controlled via a C program that will be described in more detail in Section 2.6. There are several other pieces of Xilinx intellectual property used in our system. The clock wizard is the clock generation system on the FPGA. We use three clocks in our block diagram system at DEVCOM Army Research Laboratory. In Section 3.3.6, we describe a phase measurement system that might use up to 32 additional phase-shifted clocks. The ability of the KCU105 to handle this many clocks was one of the reasons we chose this board. The other three modules not discussed in their own sections are Advanced eXtensible Interface (AXI) GPIO modules. These modules are used to interface signals to and from the microprocessor. The reason for these modules is that Xilinx automatically connects all clocks, interrupts, and interfaces between the CPU and the external modules. We have an AXI universal asynchronous receiver/transmitter (UART) that facilitates output to the monitor (GUI) allowing us to see the result. The other two modules are AXI GPIO that permit a control signal to be sent to the pattern generation system to start the acquisition, and a 16-bit date word that will indicate the gross distance measurement to be sent to the CPU. The majority of the modules are clocked by a 40-MHz clock. This clock allows for a digital pattern of 20 Gbps to be sent. The distance counter is clocked by a 200-MHz clock. This allows for more refined distance measurements. Although the manufacturer claims that clocks up to 800 MHz can be used, we found that the maximum clock rate to avoid timing faults during compilation was 400 MHz. The 200-MHz clock rate was chosen because of the future plan to use multiple phase-shifted clocks in one of our phase measurement schemes.

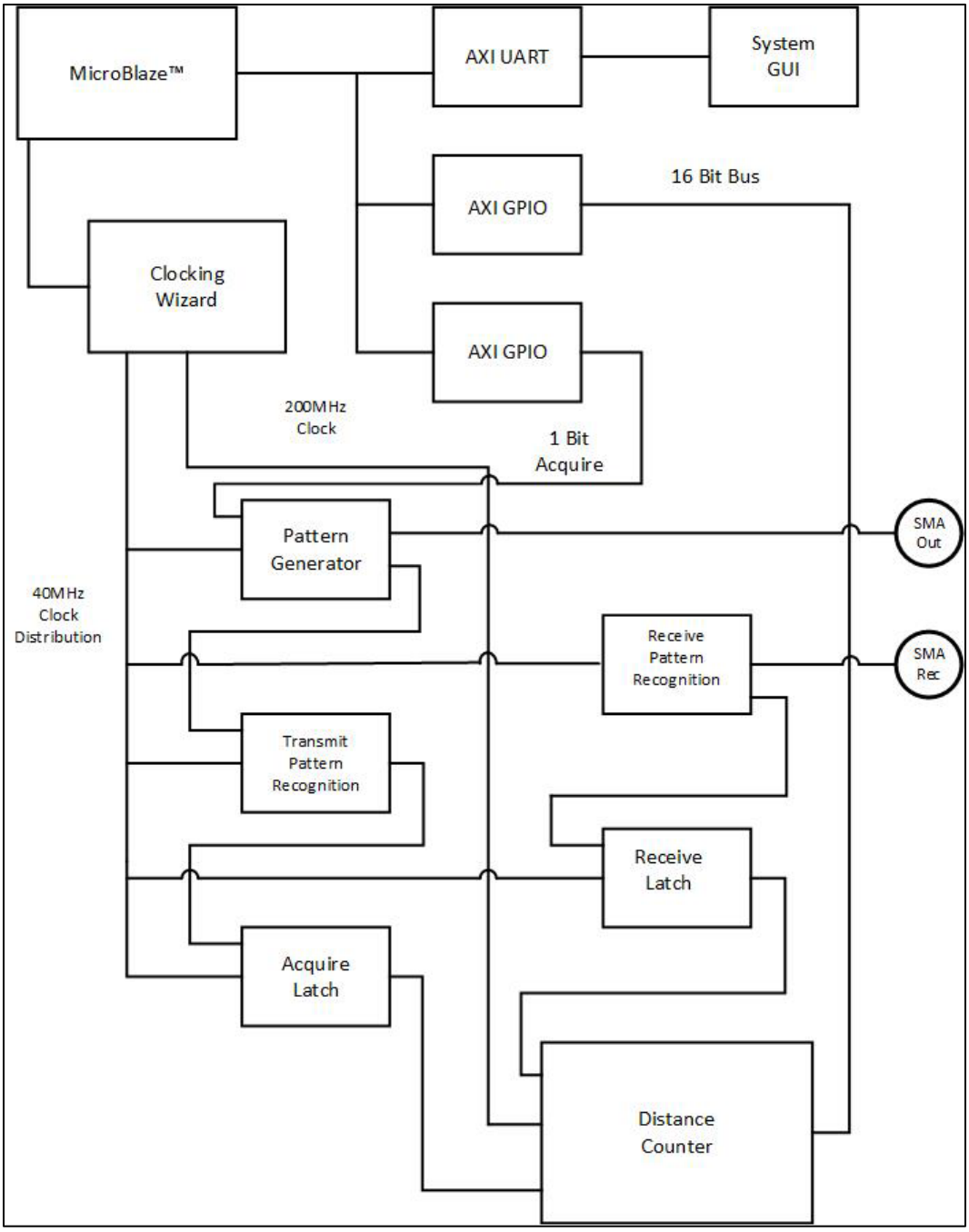

Figure 3. Block diagram of KCU105 evaluation board.

2.1 Pattern Generation

The pattern generation signal is quite simple in the ARL system. An array is created by writing a hexadecimal word that is 324 characters long. This translates to 1296 bits. The first hexadecimal character is a C (in bits, this is 1100). Thus, the first four bits are the pattern that is being detected by the pattern-recognition module. After the C, there are 323 A's. In bits, an A is 1010. This simply looks like

a clock. After the sequence of 1296 bits is complete, it repeats itself indefinitely. This does create an issue of missing the pattern on the first try and receiving the second or third. These differences are predictable. The UMBC system discussed in Section 3 has a better method of ensuring the correct result. The bit stream is sent to the user GPIO transmit SMA connector and the transmit pattern-recognition module.

2.2 Pattern Recognition

The pattern-recognition module is a Moore-type finite state machine. It samples each bit as it appears and changes state as it clocks through the pattern. If it receives a bit that is not in the desired sequence, it goes back to the first state and starts over again. The state diagram is shown in Figure 4.

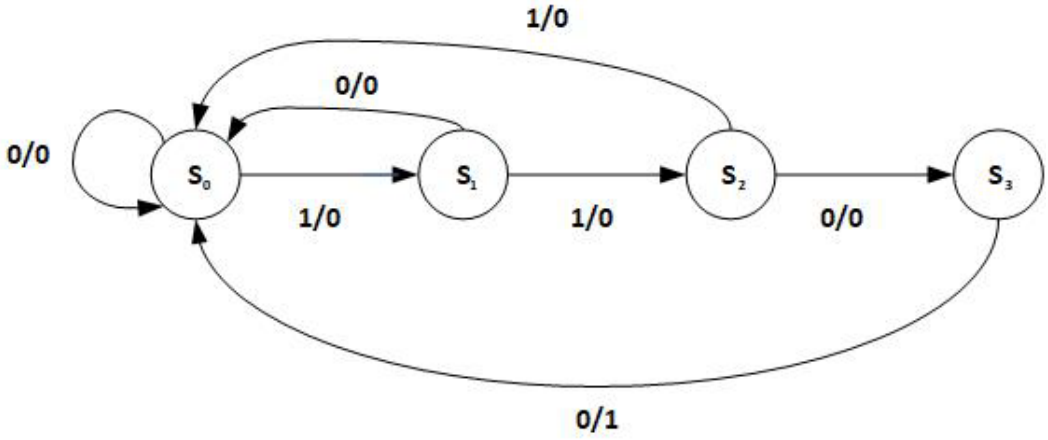

Figure 4. State diagram of the pattern-recognition system.

When the pattern is recognized, the module sets its output to one. The next module is there because this one would go back to zero on the next clock edge if it was not captured.

2.3 Latch

The latch is added because we want to keep the pattern detection signal as true while the program is running. This is better explained in the distance counter discussion. In other words, the states of the variables change at every leading edge of the clock in our implementation. To control the distance counter, we want the transmit pattern detection signal to remain true as long as the detection is going on. The latch module is added so that when the transmit pattern detection goes true, it remains true until the receive pattern is detected. The latch works in the same way on both transmitting and receiving sides.

2.4 Distance Counter

At the beginning of the program, the distance count is initiated to zero. On the leading clock edge, the module looks for the transmit pattern detection signal to be true. On the edge, if the transmit pattern detection signal is true and the receive pattern detection signal is false, then it adds one to the counter. As long as this state remains the same, it adds one count to the counter on each clock edge. When the transmit pattern detector is true and the receive pattern detector is true, the count stops, and this number is presented to the AXI GPIO so the MicroBlaze can use it in the distance calculation.

2.5 Receiving Side

The pattern-recognition module and the receive latch are logically the same on the receiving side as the transmitting side. Thus, it sets the receive latch when the receive pattern detector recognizes the pattern. This signal appears at the distance counter and stops the count. As described in the previous section, that count appears at the processor for analysis and display.

2.6 C Program Controlling the System

We have included the C code in Figure 5 to aid in discussing the logic of the program. The Xilinx Vitas tool allows the use of a C program to control the MicroBlaze processor. Xilinx has provided several header files that contain the subroutines needed to do this task. The xil_printf.h is not really required, but Xilinx claims that it does a more efficient job of printing; however, it does take up space in block memory. The most important file is xgpio.h. This contains the subroutines used in the program. The program starts out by initializing the platform and the variables used in the program. In lines 17–20, it initializes the AXI GPIO ports with direction, transmit, or receive. The names given to these ports in the configuration file are AXI_GPIO_0 and AXI_GPIO_1. It is important to use these names exactly. In lines 23 and 24, it creates masks that allow for 1 bit on the output and 16 bits on the input. Next, the discrete write to the output port is the initialization signal to the transmit pattern generator. This signal starts the transmission of the pattern.

Lines 26–31 are a do-while loop that is waiting for the data at the AXI GPIO input to be something other than zero. Note the reason that the variable “number” was initialized at zero. When a number is received, it prints the number on the screen. This number represents the number of clock counts it took for the pattern to leave the KCU105, go to the corner cube, and return. This is twice the distance to the corner cube. Later (when we also have a phase measurement), these two measurements will be used to determine the actual distance to the corner cube.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "xparameters.h"

int main()
{
    init_platform();

    int number = 0;
    int acq = 1;

    XGpio input,output;
    // initialize AXI GPIO
    XGpio_Initialize(&input, XPAR_AXI_GPIO_0_DEVICE_ID);
    XGpio_Initialize(&output, XPAR_AXI_GPIO_1_DEVICE_ID);

    // set data direction
    XGpio_SetDataDirection(&input, 1, 0xFFFF);
    XGpio_SetDataDirection(&output, 1, 0);

    XGpio_DiscreteWrite(&output, 1, acq);

    do {
    number = XGpio_DiscreteRead(&input, 1);
    }
    while (number < 1);

    xil_printf("The number of clock ticks to the target
is:%d\r\n",number);

    cleanup_platform();
    return 0;
}
```

Figure 5. A copy of the source code in C.

3. Phase Measurement

3.1 Introduction

A phase detection system is a crucial technology used in various fields of electronics and communications and is designed to measure the phase difference between two signals, typically oscillating waveforms like sinusoidal or digital signals. The phase refers to the relative alignment or timing between the peaks and troughs of these wave forms. When two signals have a phase difference, it indicates that one signal is either leading or lagging behind the other in time.[1]

A phase detector is the core component of the system. It compares two input signals, determines their phase difference, and outputs a signal representing this

difference. This output is often used as feedback to correct the phase relationship between signals. In phase-locked loops (PLLs), the phase detector is part of a closed-loop system where it locks the output signal to the phase of a reference signal by adjusting its frequency, thus maintaining synchronization.[2]

The phase detection system is widely used in various fields including telecommunications, where it is employed for signal synchronization in modulation and demodulation processes. In radar and sonar systems, phase detection helps determine the distance and velocity of objects by measuring the phase shift between transmitted and received signals. In motor control systems, phase detection enables precise control of motor positioning and speed. It is also crucial for clock and data recovery circuits in high-speed digital communication, where it helps recover timing information. Lastly, test and measurement equipment such as oscilloscopes and spectrum analyzers rely on phase detection for accurate signal analysis.[1,2]

In this report, we propose a method to calculate the delay time between sending and receiving a packet in a time transfer system, which can help estimate the distance traveled by the packet. This method combines both packet transfer and a phase detection system. The system consists of two points: one is the transmitter, and the other is the receiver. These two points may be located at the same place in a loop-back configuration.

In this system, a data packet containing information such as time is generated and sent using optical technology. After a delay, the packet reaches the receiver, which retrieves the data. Our goal was to calculate the delay time in the data transfer; however, the received signal is not synchronized with the receiver, resulting in a phase shift between the transmitted and received signals. Therefore, the purpose of this project is not only to calculate the delay time in the data transfer but also measure the phase shift within the system. By combining these two values, we can achieve a more accurate resolution of the transmission delay.

Sections 3.2 and 3.3 explain the overall structure of the design for estimating the time delay, followed by a detailed description of each component, highlighting their advantages and limitations.

3.2 The Concept of Clock Sweeping and Its Accuracy

The goal of this project is to accurately calculate the delay time between transmitting and receiving signals, which is essential for estimating distance. To achieve this, a phase detection system is required. Figure 6 illustrates the rationale behind designing such a system.

In Figure 6, the transmitted signal is generated and sent out at the first edge of the clock (represented by the dotted red line). Due to line delay, the received signal arrives later, but with a phase difference relative to the transmitted signal. Our goal is to calculate this phase difference accurately. If we rely solely on the system clock to detect the signal, we will detect the received signal at the second edge, which does not represent the correct phase difference.

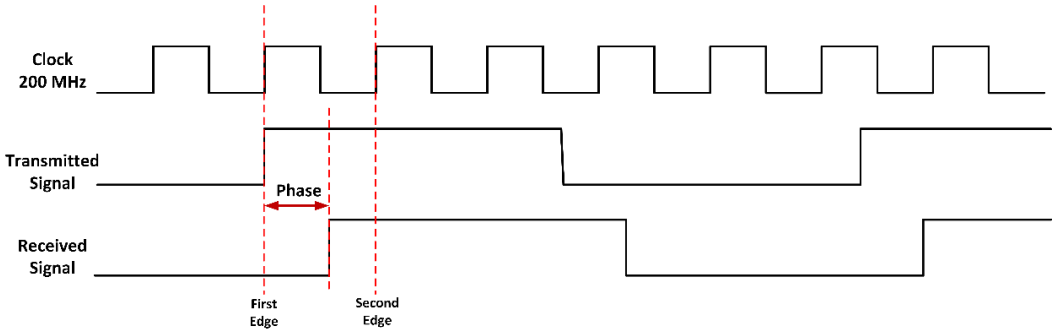

Figure 6. Phase difference between transmitting and receiving signal.

Therefore, we need a method to calculate this phase difference with greater precision. There are different methods for phase measurement, which can be categorized into digital and analog approaches. Our aim is to leverage the capabilities of the FPGA to calculate this phase difference without using additional hardware. As shown in Figure 5, one way to address this problem is by triggering the signal with multiple clocks that are close to the edge of the received signal. This allows for a more accurate phase difference estimation. Figure 7 demonstrates this concept using four clocks with the same frequency but different phases.

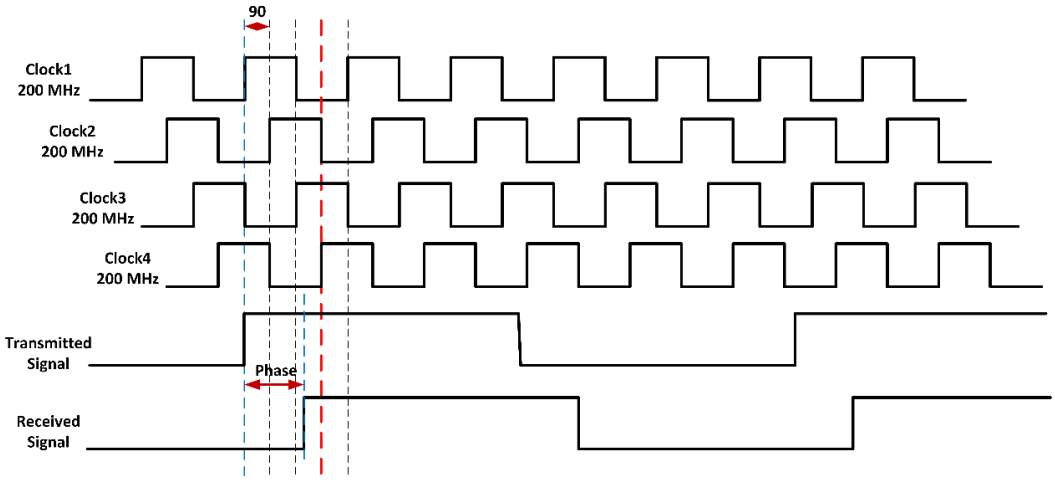

Figure 7. Estimating phase by four clocks with 90° phase shift.

We used four different clocks to detect the received signal, each with a 90° phase shift (Figure 7). Clock number 4, represented by the dotted red line, successfully detects the received signal in this setup. This clock is much closer to the edge of the received signal, meaning our new phase calculation is more accurate compared with the previous one shown in Figure 4. By measuring the phase using different clocks inside the FPGA, a question arises: is it possible to generate clocks with specific phase differences within the FPGA? The answer is yes. The FPGA's PLL includes a block called the Mixed-Mode Clock Manager (MMCM) that can generate clocks with a resolution as fine as 11.25° phase shifts, allowing for up to 32 different clocks.

Before expanding the system to use more clocks, let us consider the accuracy of using only four clocks, as shown in Figure 7. As previously mentioned, our clock frequency is 200 MHz, which corresponds to a period of 5 ns. Using 90° phase shifts divides the clock into four equal phases: 0°, 90°, 180°, and 270°. This means the 5-ns period is divided into four equal parts, each 1.25 ns. Therefore, our phase-measurement accuracy is 1.25 ns. Based on this time resolution and the speed of light, we can calculate the potential error in distance estimation.

We report accuracy in terms of time rather than degrees because the phase accuracy depends on the clock frequency. For example, with a 100-MHz clock, a 90° phase shift corresponds to a 2.5-ns time step, which is twice as long as the 1.25-ns step at 200 MHz. Thus, time resolution is a more consistent metric for reporting accuracy across different frequencies.

As shown in Figure 8, increasing the number of clocks allows us to improve the accuracy of phase measurement. For instance, the maximum number of clocks that can be generated using the MMCM is 32, with an 11.25° phase shift between each consecutive clock. For a 200-MHz clock, which has a period of 5 ns, dividing this period by 32 results in a time step of 156.25 ps. Consequently, the phase measurement accuracy of the system is 156.25 ps, which can be converted into distance for highly precise distance measurement.

However, generating 32 clocks inside the FPGA using MMCM comes with certain challenges. Achieving a phase resolution of 156.25 ps requires careful manual placement of components within the FPGA to ensure optimal clock distribution. In practice, several limitations—such as phase errors, signal jitter, and high rise and fall times—introduce difficulties in achieving this level of accuracy. These limitations will be discussed further in Section 3.3.6.

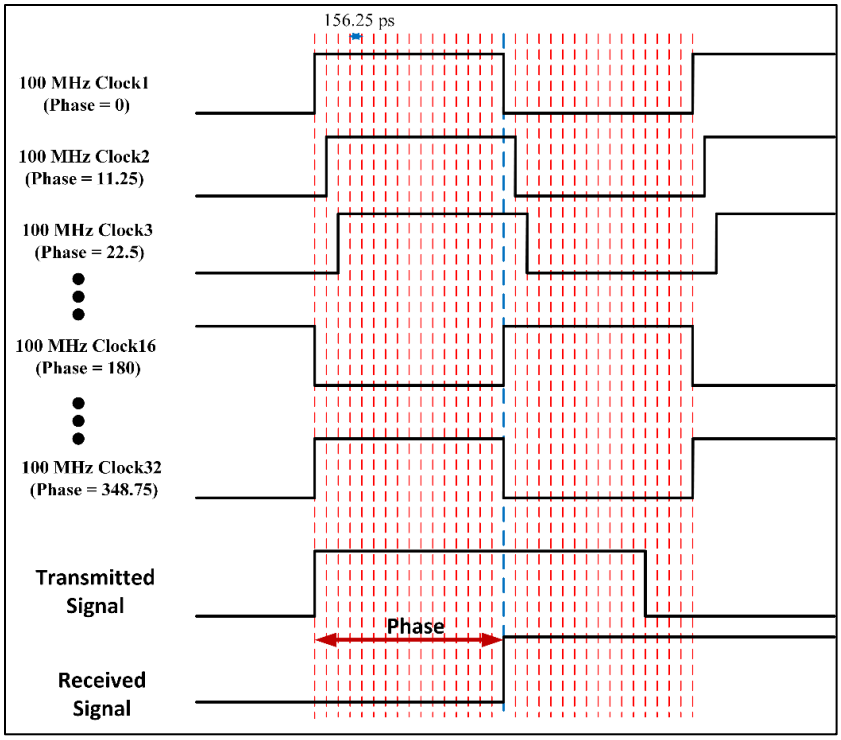

Figure 8. Phase measurement with a higher number of clocks to have better accuracy.

3.3 Method Implementation

As mentioned earlier, our goal is to calculate the time transfer of a signal through an optical system. This calculation involves both coarse and fine measurements. For instance, if the delay time is 12.5 ns, the 12 ns represents the whole part, and the 0.5 ns represents the fractional part. The whole part can be determined using straightforward methods that were discussed in Section 2.4. However, a phase detection system is required for the fractional part, as it falls below the resolution of typical measurement systems.

There are various methods for phase detection, including both analog and digital approaches. This project's objective is to leverage the existing capabilities of our hardware for phase detection without the need for additional components or equipment. This approach allows us to perform precise phase measurements within the system's current limitations.

Figure 9 illustrates the entire clock sweeping system. This model involves several components, including an MMCM to generate clocks; a MicroBlaze system as a processor to generate the content of the packets and monitor the final results; a packet transmitter to convert packets into a series of digital bits and send them out; a filter block to remove glitches from the input signal; a packet receiver to receive the signal and extract data from it; a priority encoder to detect which receiver

detects the packet first; and a result reporter to save and send the delay counter and phase number—as the whole and fractional parts of the estimation—to the MicroBlaze for display. These parts are explained in detail in Sections 3.3.1–3.3.5.

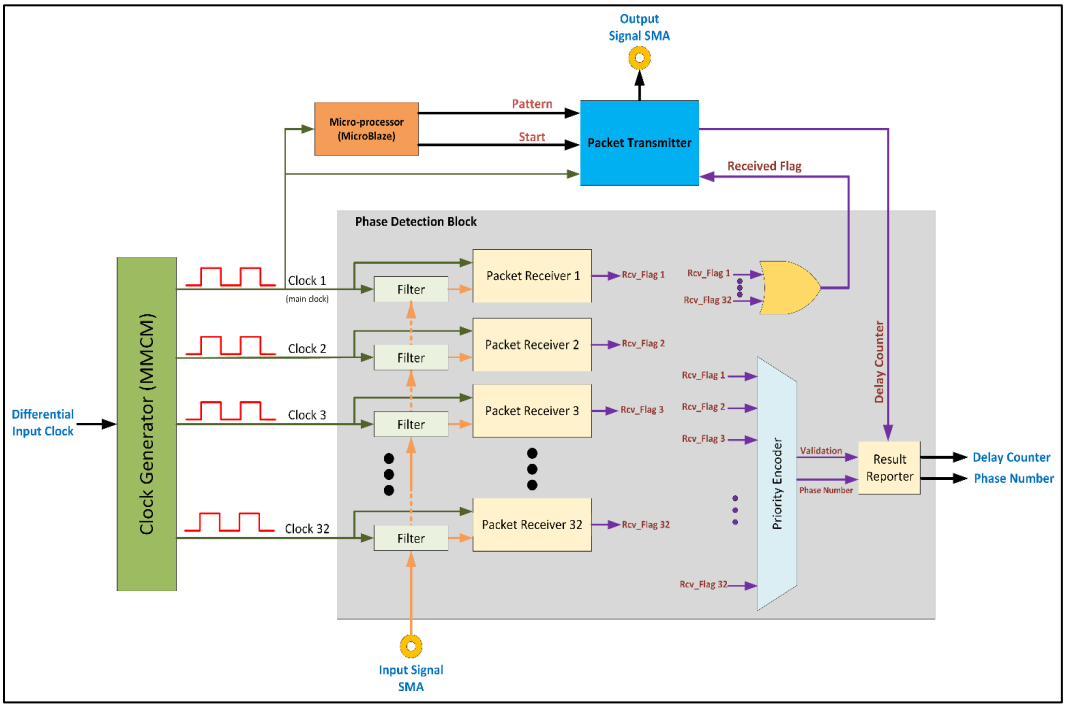

Figure 9. The structure of the whole clock sweeping system.

3.3.1 Packet Transmitter

The packet transmitter is responsible for sending packets. It is directly connected to the MicroBlaze processor to receive commands. The two key commands received by this block are the packet and start signal. In the MicroBlaze, a packet containing 16-bit randomly generated data is sent to the packet transmitter, followed by a 1-bit start signal. When the packet transmitter detects the positive edge of the start signal, it begins transmitting the packet.

The system is designed to continuously output a clock signal, meaning the output line cannot remain high (1) or low (0) for extended periods. When no packet is available for transmission, a clock signal must be sent to maintain system stability. For this purpose, a 20-MHz clock is chosen as the default output signal during idle periods. Figure 9 illustrates the signal structure on the output line. Inside the FPGA, the system operates at a 200-MHz frequency to manage all processes. The packet transmitter generates a 20-MHz clock based on an internal counter and sends it out continuously. When a start signal arrives from the MicroBlaze, the packet transmitter switches from sending the 20-MHz clock to transmitting the packet.

Suppose you want to send the data 0xb121 to the output line. As shown in Figure 10, the 20-MHz clock is on the output line by default. When the packet transmitter detects the positive edge of the start signal, it recognizes that the data is ready for transmission. To facilitate easier detection on the receiver side, data transmission begins with a start bit, which consists of a low signal (zero level) held for 20 clock cycles. All counters mentioned in this article are based on the main clock frequency of 200 MHz. The data is transmitted after the start bit, with each data bit held on the line for 10 cycles of the main clock. For example, if we transmit 0xb121 (binary: 1011 0001 0010 0001), we first send the most significant bit (1), followed by zero, and then two consecutive ones for the value 0xb. This process continues for all 16 bits of data. The system returns to idle mode after all bits have been transmitted, with the 20-MHz clock restored on the output line.

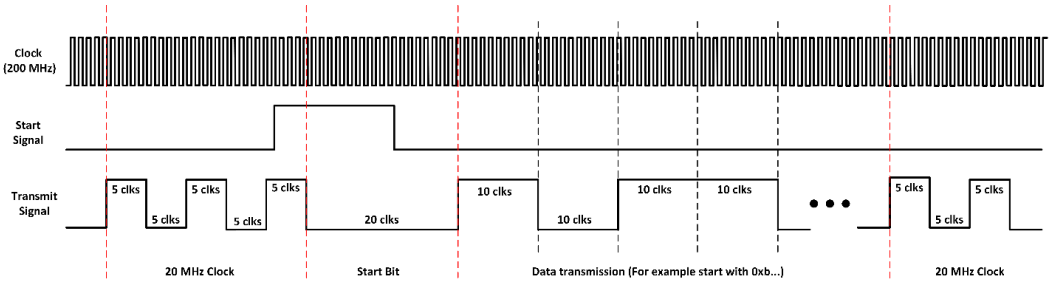

Figure 10. Signal structure from packet transmitter on the output line during packet sending.

The packet transmitter block also has the additional function of measuring delay time. When the transmitter begins sending data, it starts counting to track the delay between transmission and reception. The counter stops upon detecting the received flag (an input signal to this block), and the delay value is sent to the phase detection block for final calculations.

3.3.2 Filter Block

One of the most important blocks in this system is the filter block, which is designed to remove any unexpected glitches from the input signal. As shown in Figure 8, each receiver block has its own filter block. The purpose of this block is to improve the quality of the received input signal by eliminating noise. Figure 11 illustrates the functionality of this block and demonstrates how it effectively removes unwanted signals from the input.

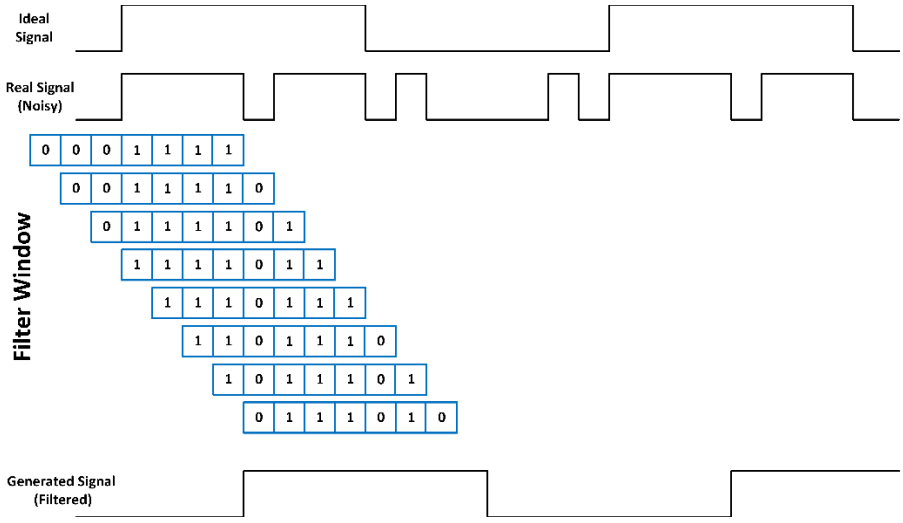

Figure 11. Operation of the filter block.

Based on Figure 11, our goal is to work with an ideal signal that is clean and free from noise. However, the received signal often contains glitches. Since the system relies on detecting positive or negative edges of the signal for processing, a noisy signal can lead to incorrect decisions. Therefore, we need a block to remove these glitches and produce a cleaner signal, such as the filtered signal shown in Figure 10, for further processing.

To achieve this, we designed a simple and compact filter that checks the number of ones and zeros on the line. It determines the real value of the output based on the majority of zeros and ones. In practice, the distribution of these glitches is typically random enough to be effectively filtered out by this approach.

This filter block has adjustable parameters, such as the window length and the decision margin for identifying ones and zeros. These parameters must be set appropriately based on the operating environment and the frequency of the signal being filtered. The output of this block is then sent to the packet receiver block, which is detailed in the following section.

3.3.3 Packet Receiver

There are multiple packet receivers in this design, each operating with a different phase-shifted clock. These receivers play a crucial role in the phase detection system. As mentioned earlier, the received signal is not synchronized with the internal FPGA clock. To accurately determine the phase shift between the transmitted and received signals, different clocks with the same frequency but varying phase shifts are used. Each clock is connected to a separate receiver block to detect the incoming packet. All receiver blocks function similarly, monitoring the input line for the start bit (Figure 9). When a receiver detects the start bit, it

begins receiving and storing each bit of the 16-bit packet. After all bits have been received, the received data is compared with the stored transmitted data and the detection flag is activated if they match. The purpose of having multiple receivers is to determine which one detects the packet first, as the clock associated with that receiver indicates the phase number to be used in the final time or distance estimation.

The output of each receiver block is a signal called the received flag, which is fed into the priority encoder block (Figure 9). There are multiple receiver blocks based on the number of phase-shifted clocks in the system, each with its own received flag output. Since the activation times of these flags differ, an OR gate logic is used to combine these signals and send the detection information to the final received flag for the packet transmitter, allowing it to record the delay time in this block.

3.3.4 Priority Encoder

Another important block in this design is the priority encoder. Since we have multiple receiver blocks, each associated with a phase-shifted clock, many (or even all) of them may detect the input packet. However, the key difference is the time when each receiver detects the packet. One receiver will detect it sooner than the others. The priority encoder's role is to identify which receiver detects the packet first and report the corresponding clock number. This is critical for accurate phase calculation.

The priority encoder is a simple logic block that monitors all the input received flags from the receiver blocks. When any of these flags are activated, the priority encoder sets a validation signal, reports the clock number, and holds this output until the next packet is received. The output of this block is then used by the result reporter block.

3.3.5 Result Reporter

The structure of this block is straightforward. It receives the validation signal, phase number, and delay counter, then saves the correct phase value and delay counter. This information is subsequently sent to the MicroBlaze for reporting the results to the user.

3.3.6 Clock Generation (MMCM)

This section provides a detailed explanation of the clock generation process and its limitations. This block plays a core role in the phase detection system. Our approach leverages the FPGA's built-in capabilities to generate multiple clocks for phase

detection. This method allows us to measure phase accurately without the need for additional hardware or external circuits.

In most FPGA architectures, an MMCM is used to generate various clocks. In devices based on the UltraScale architecture, the Clock Management Tile (CMT) contains an MMCM and two PLLs. While the primary function of the PLL is to generate clocking for the inputs/outputs (I/Os), it also includes a limited subset of MMCM functions for use within the FPGA fabric.

The clock input connectivity in these devices allows multiple sources to provide reference clocks to the MMCM. The MMCM has eight output counters (dividers), some of which are capable of generating an inverted clock signal with a 180° phase shift. Additionally, MMCMs offer fine phase shift capabilities in both directions, which can be used in dynamic phase shift mode. The resolution of this fine phase shift depends on the frequency of the voltage-controlled oscillator (VCO).[3]

Devices based on the UltraScale architecture include one CMT per I/O bank, with the MMCMs serving as key components for frequency synthesis across a wide range of frequencies, jitter filtering for both external and internal clocks, and clock deskewing. MMCMs are designed to reduce the jitter inherent in a reference clock.[3]

The MMCM can also be used as a stand-alone function specifically for jitter filtering, where it cleans up an external clock signal before passing it on to other blocks. In this mode, the MMCM essentially acts as a buffer, regenerating the input clock frequency on its output, ensuring a cleaner, more stable signal.

The MMCM has some restrictions that must be adhered to. In general, the major limitations are VCO operation range, input frequency, duty cycle programmability, and phase shift. In many cases, there needs to be a phase shift between clocks. The MMCM has multiple options to implement phase shifting. Static phase shifting can be achieved by selecting one of the eight VCO output phases, with additional fine phase shifting available in the CLKOUT output counters depending on the CLKOUT divide value. There is also an interpolated phase-shifting capability in either fixed or dynamic mode. The MMCM phase-shifting capabilities are very powerful, which can lead to complex scenarios. By using the Clocking Wizard, the allowable phase-shift values are determined based on the MMCM configuration settings.[3]

Each MMCM can provide up to seven output clocks with different configurations. Figure 12 shows the configuration of one MMCM with the minimum phase difference, which is 11.25°. As illustrated, the first clock has no phase shift, while each subsequent clock has an exact 11.25° phase shift relative to the previous one.

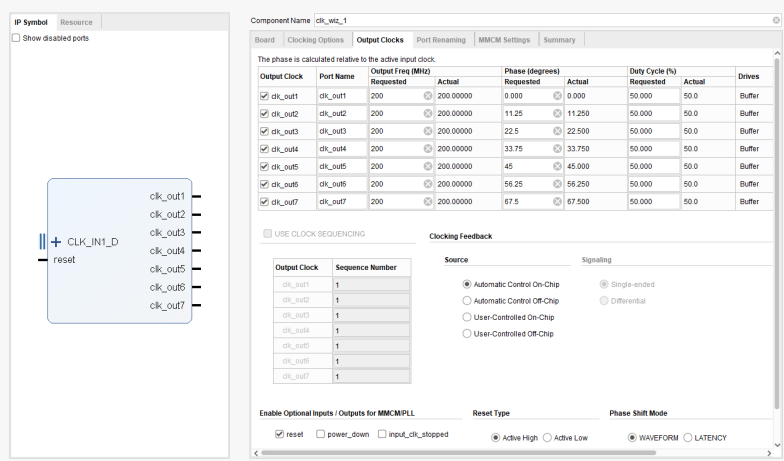

Figure 12. Configuration of one MMCM block to generate seven clocks with 11.25° phase shift.

However, there is a fixed phase error in this system, and as we increase the number of clocks with high-resolution phase shifts, such as 11.25°, the impact of these errors becomes more significant. Figure 13 presents the final report for generating these clocks. As shown, there is a phase error of 114.212 ps. This means that when using 32 clocks with an 11.25° phase shift for a 200-MHz clock, each clock covers 156.25 ps. Consequently, this error indicates that, at times, a clock may appear earlier or later than expected, affecting the accuracy of phase detection.

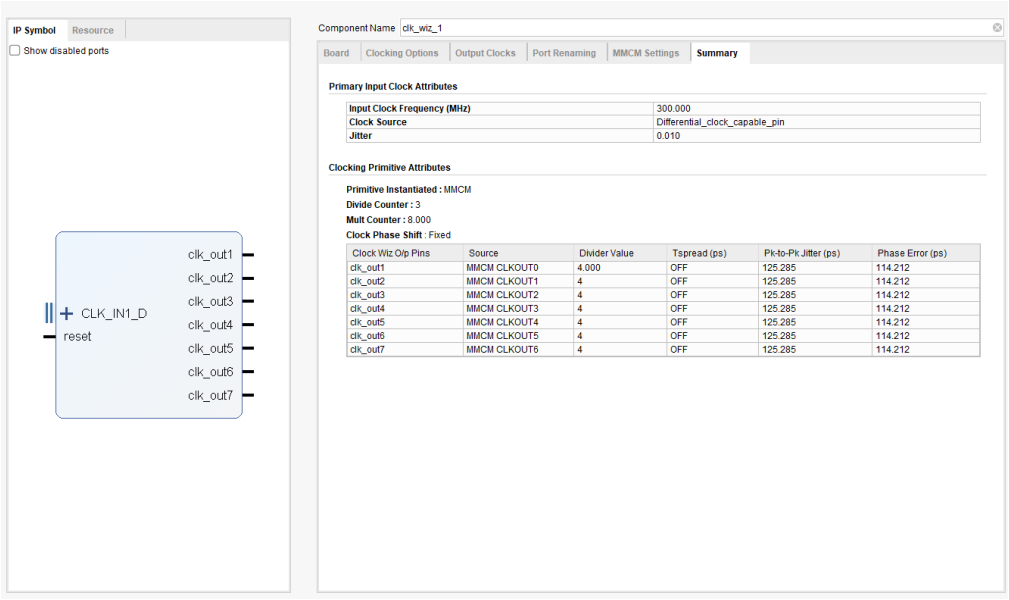

Figure 13. Final report of MMCM after generating seven clocks.

It is challenging to verify the accuracy of generating these clocks in real-world conditions, so we opted to simulate the generation of 32 clocks after the implementation process, taking all timing considerations into account. This

simulation step occurs just before generating the bitstream, making it a close approximation to actual hardware testing. Figure 14 illustrates the structure of these generated clocks in the simulation.

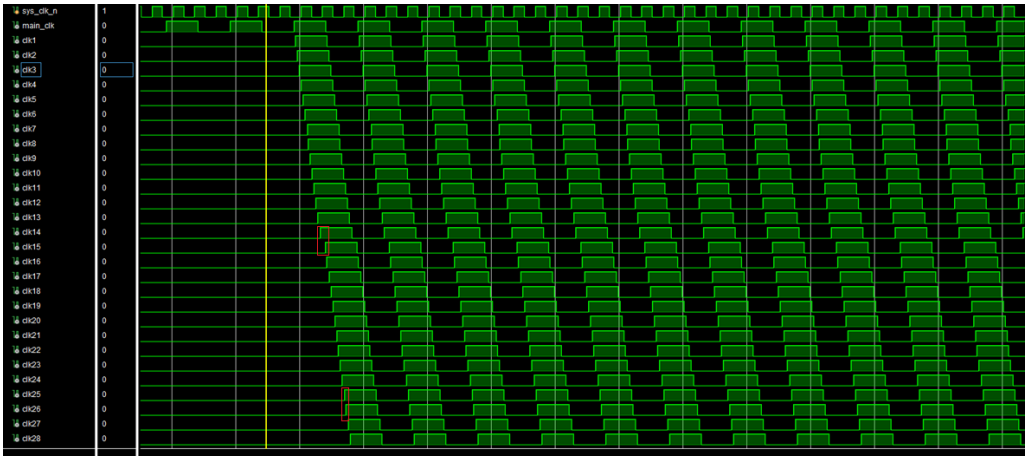

Figure 14. Timing simulation for generated clocks in implementation step.

As shown in Figure 14, some clocks exhibit slightly more or less phase shift than expected. Overall, this unit is responsible for generating multiple clocks with programmed phase shifts, which are then used in the phase-shift detection block to accurately detect the received pattern.

4. Pros and Cons of Clock Sweeping

The clock sweeping method for measuring phase is a straightforward approach that leverages the inherent capabilities of FPGAs to generate multiple clocks, allowing for accurate phase calculation. Like any system, this design has its advantages, disadvantages, and limitations that we discuss in this section.

Using an FPGA in an optical time transfer system presents a unique opportunity to utilize its features for phase measurement. It is important to note that employing the MMCM as the core of this method is an FPGA-specific solution. One key advantage of this approach is its simplicity, as it can be implemented without additional circuitry. The only consideration is that, to achieve better resolution, manual placement on the FPGA may be required after implementation to optimize clock skew, which improves delay and accuracy.

One limitation of this method is the number of output clocks that can be generated, which is constrained by the number of available MMCMs on the FPGA. However, this is generally not a significant issue, as most FPGAs have enough MMCMs to generate a sufficient number of clocks. A more critical limitation of the MMCM is that the phase-shift values cannot be set to arbitrary angles. For example, phase

shifts like 10°, 31°, or 16° cannot be directly configured. The phase shift values are determined by the divider counter, VCO settings, and other parameters, with the minimum phase shift typically being 11.25°. Values below this are not possible due to the structure of the MMCM.

Lastly, factors such as jitter and high rise and fall times can impact the performance of this phase detection system. These factors may limit the number of clocks that can be generated beyond a certain point. However, this issue can be mitigated by employing methods such as averaging between two consecutive clocks to compensate for these limitations.

5. Phase Detection Using Modified J-K Flip-Flop

Apart from using the clock sweeping method, another effective approach for phase detection involves utilizing a modified J-K flip-flop. This method operates by transmitting a predefined signal pattern, as described in Section 3.1. However, instead of using multiple receivers for phase comparison, a single receiver is employed to capture the transmitted signal. Once the signal is received, the system analyzes the pattern and measures the time interval between the transmission and reception of the signal. This measured time provides an initial distance count. However, a phase detection mechanism is necessary to achieve more precise distance measurements.

As shown in Figure 15, the phase detection circuit consists of two D flip-flops configured in a specific arrangement that functions as a modified J-K flip-flop (see Table 1). In this setup, the clock inputs of the D flip-flops are connected to the received signal and the system's internal reference clock. The output of this modified J-K flip-flop represents the exact phase difference between the internal clock and the received signal (Figure 16). The key principle is that the width of the output pulse generated by the flip-flop is directly proportional to the phase difference. By accurately measuring this pulse width, it becomes possible to determine the phase shift with high precision. One way to measure this pulse width is by sampling the pulse at a very high frequency. However, using a high-frequency sampling clock is not cost-effective, and if such a high-frequency clock were available, it would be more efficient to use it for direct distance sampling, eliminating the need for the phase detection circuit.

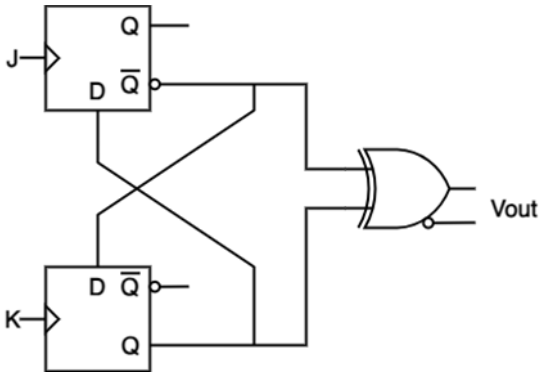

Figure 15. Modified J-K flip-flop.

Table 1. Truth table.

J	K	Q_{n+1}
0	0	Q_n
0	⎍	0
⎍	0	1
1	1	Q_n
1	⎍	not Q_n

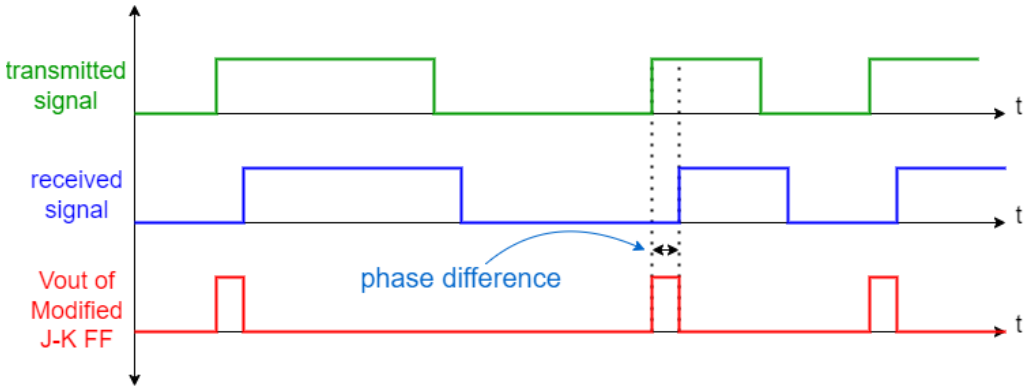

Figure 16. Timing diagram of phase measurement using a modified J-K flip-flop.

An alternative method involves feeding the generated pulse to a GPIO pin and then measuring the voltage at that pin. The GPIO pin outputs a signal whose root mean square voltage corresponds to the pulse width and, hence, the phase difference. To map this voltage to an accurate phase difference value, an analog-to-digital converter (ADC) is required. The challenge is that the onboard ADCs available in the FPGA are often limited in their voltage range and cannot measure voltages higher than 1 V. To overcome this limitation, we utilize an external Arduino Mega 2560 board with a more capable ADC to measure the voltage output from the GPIO pin.

To obtain the final result, the measured phase value from the Arduino needs to be integrated with the distance count from the FPGA. This is done by transmitting the distance counter value from the FPGA and the phase value from the Arduino to a

computer via UART communication. The data is then processed to adjust the distance count using the phase correction, resulting in a more accurate distance measurement. The whole high level block diagram is shown in Figure 17.

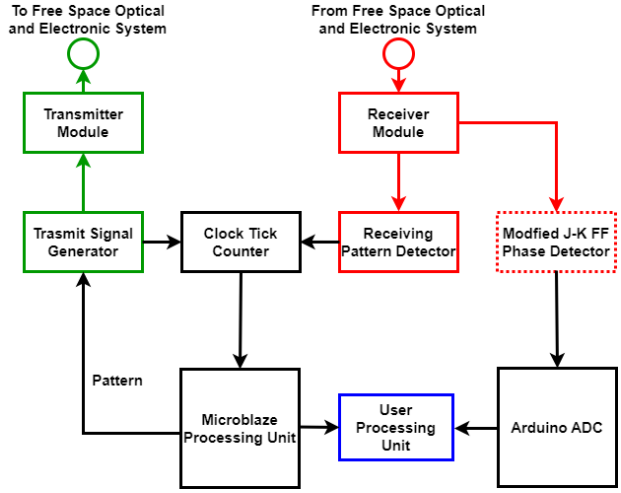

Figure 17. High-level block diagram of distance measurement system using a modified J-K flip-flop.

6. Pros and Cons of the Modified J-K Flip-Flop Method

The design of a modified J-K flip-flop using D flip-flops is relatively straightforward and easier to implement in hardware compared to more complex approaches, such as PLLs or delay-locked loops. Additionally, using a single receiver reduces resource overhead compared to systems employing multiple receivers for phase comparison, making this method particularly suitable for FPGA implementations with limited hardware resources. Moreover, integrating an external ADC (such as the one on an Arduino) helps overcome the limitations of the onboard FPGA ADCs, enabling a wider measurement range and improved accuracy.

However, the use of separate components and the need for UART communication between the FPGA and Arduino can introduce synchronization delays between the phase and distance data. This requires careful handling during the development phase to ensure accurate data integration. Furthermore, the accuracy of phase detection is highly dependent on the resolution of the ADC. If a low-resolution ADC is used, the precision of the measured phase difference may suffer, making careful selection of the ADC a critical design consideration. Additionally, to produce a complete and accurate result, the measured phase difference must be calibrated against the main distance counter value, which adds a layer of software complexity and requires precise tuning.

In conclusion, the modified J-K flip-flop-based phase detection system is well-suited for applications that require precise phase measurements with moderate resource usage. However, its implementation demands careful attention to hardware integration and management of latency to achieve optimal performance.

7. Test Results

A test of the system was conducted in an auditorium at UMBC. The results are presented in Table 2.

Table 2. Data collected at UMBC.

Distance (m)	Clock counts	Actual clock counts	Measured distance (m)
0	23	0	0.0
4	26	3	4.5
8	28	5	7.5
12	31	8	12.0
16	34	11	16.5
20	36	13	19.5

The actual clock counts include 23 clock cycles that can be accounted for by various lengths of cable and internal delays to the system. The main thing to understand is that this delay is the same in all measurement. Since the clock period is 5 ns (200 MHz), the per-period travel distance is approximately 1.5 m. As Table 2 shows, the distance measurements are all within this tolerance.

We are not planning to test the phase measurement system in the lab. Up to the time of this writing, we have had some issues with the rise time of the signal because it seems to be dependent on the level of the signal that reaches the comparator. Once these issues are solved, we will continue the testing.

We made a phase measurement using the J-K flip-flop system described in Section 5. The result is tabulated in Table 3. The team at UMBC used a standard tape measure with inch ruling; thus the data is in inches. The average error in the measurement is 1.726 inches, or a time equivalent of approximately 150 ps. For an initial measurement, we are very happy with this result. In the future, we plan to refine these measurements and move out of the laboratory to longer distances and varied weather conditions.

Table 3. Phase data collected at UMBC.

Phase	JK flip-flop phase measurement distance (inches)	Real distance (inches)	Error (inches)
1	48.14	48	0.14
2	65.36	66.5	–1.14
3	102.27	106	–3.73
4	92.42	93.5	–1.08
5	62.9	65	–2.1
6	75.21	74	1.21
7	72.75	70	2.75
8	45.68	44	1.68
9	40.76	37.5	3.26
10	35.83	36	–0.17
		Average:	1.726
		Standard deviation:	1.163
		Median:	1.445

Note: The average, standard deviation, and median are based on absolute value of the error.

8. References

1. Du B, Tan L. High-accuracy phase frequency detection technology based on bds time and frequency signals. Sensors. 2024;24:4606.
2. Maiti M, Saw SK, Nath V, Majumder A. A power efficient PFD-CP architecture for high speed clock and data recovery application. Microsyst Technol. 2019;25:4615–4624.
3. UltraScale architecture clocking resources user guide. Xilinx; 2013.

List of Symbols, Abbreviations, and Acronyms

ADC	analog-to-digital converter
ARL	Army Research Laboratory
AXI	Advanced eXtensible Interface
BW	bandwidth
CMT	Clock Management Tile
CPU	central processing unit
CW	continuous wave
DEVCOM	U.S. Army Combat Capabilities Development Command
FPGA	field-programmable gate array
GPIO	general purpose input/output
GUI	graphical user interface
I/O	input/output
MMCM	Mixed-Mode Clock Manager
PLL	phase-locked loop
RF	radio frequency
SMA	SubMiniature version A
UART	universal asynchronous receiver/transmitter
UMBC	University of Maryland, Baltimore County
VCO	voltage-controlled oscillator

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB