A High Resolution Time-to-Digital Converter on FPGA for Time-Correlated Single Photon Counting

by Qiuchen Yuan

B.S. July 2010, Beijing University of Technology

A Thesis submitted to

The Faculty of
School of Engineering and Applied Sciences
of The George Washington University
in partial fulfillment of the requirements
for the degree of Master of Science

August 31st, 2012

Thesis directed by

Mona E. Zaghloul

Chair, Department of Electrical and Computer Engineering,
Director of the Institute for MEMS/NEMS and VLSI Technologies

Dedicated to my Family and friends

Acknowledgments

First, I would like to express my gratitude to my advisor Dr. Mona Zaghloul and my co-advisor Dr. Jerry Wu for their support and guidance throughout my graduate school career at GWU. It was a nice and a rewarding experience to work with them.

I am extremely grateful to Zhenyu Li, and Bowei Zhang for the engaging technical discussions and advice.

Last but not least I would like to thank my family for providing me with constant encouragement and support.

Abstract of Thesis

A high resolution Time-to-Digital Converter on FPGA for time-correlated single photon counting

Time-Correlated Single Photon Counting (TCSPC) can provide not only the time information of a photon, but also the photon density information. Based on the conclusion of usual time interval measuring methods, this thesis presents the scheme of Time-to-Digital Converter (TDC) based on delay line structure, meeting the TCSPC system's requirement for high timing resolution. The TDC design uses Field-Programmable Gate Arrays (FPGA) devices as the platform. The FPGA platform is chosen because the design process of ASIC device not only can be expensive, especially if produced in small quantities, but also the design process is complex due to the long turn-around time and layout phase. This TDC device contains one main coarse counter and two fine counters, which use multiple parallel tapped delay lines to make the find counter more accurate. In this thesis, we use Altera Cyclone III EP3C25 chip to implement the TDC. The simulation of the Verilog code is using ModelSim SE 6.1f; and implement the TDC in Quartus 2 9.0. The histogram of the TCSPC in PC can get from FPGA board; and the communication between the FPGA board and PC by using RS-232. The realized TDC using Verilog, and it was tested by using software simulation and on-board test, and the implemented resulted in TDC system time resolution below 200 ps.

Table of Contents

Dedication ii

Acknowledgments iii

Abstract of Thesis iv

List of Figures vi

List of Tables viii

Chapter 1: Introduction 1

Chapter 2: Introduction of Time Difference Measurement 8

Chapter 3: The TDC Architecture 16

Chapter 4: Optimize the TDC Architecture 23

Chapter 5: Simulation 26

Chapter 6: Measurement 29

Chapter 7: Conclusion 44

Reference 46

Appendix 50

List of Figures

Figure 1.1……3

Figure 1.2……4

Figure 1.3……5

Figure 1.4……6

Figure 2.1……8

Figure 2.2……9

Figure 2.3……10

Figure 2.4……13

Figure 3.1……16

Figure 3.2……17

Figure 3.3……18

Figure 3.4……19

Figure 3.5……21

Figure 3.6……22

Figure 4.1……24

Figure 4.2……25

Figure 5.1……27

Figure 5.2……28

Figure 6.1……29

Figure 6.2……31

Figure 6.3……32

Figure 6.4……..33

Figure 6.5……..34

Figure 6.6……..35

Figure 6.7……..36

Figure 6.8……..37

Figure 6.9……..37

Figure 6.10…….……..38

Figure 6.11……..….39

Figure 6.12……..….43

List of Tables

Table 1…….15

Table 2…….28

Table 3…….43

Chapter 1

Introduction

1. Motivation

With the development of biology medicine, space environment detection, and nuclear detection technology, single photon counting technology was applied widely in biology fluorescent and spectroscopic measurement. The technology can provide not only the time information of a photon, but also the photon density information. Using this technology, time-resolved fluorescence spectroscopy is very powerful in fundamental physics. Using short light or flash, typically a laser pulse, the time-resolved fluorescence spectroscopy can be implemented in the time domain, which requires recording the time dependent intensity profile of the emitted light upon excitation. In principle, one could attempt to record the time decay profile of the signal from a single excitation-emission cycle, the practical problems is very challenging. First of all, the decay to be recorded is very fast. Typical fluorescence from important organic fluorophores lasts only some hundred picoseconds to some tens of nanoseconds. In order to recover not only fluorescence lifetimes but also the decay shape, one must be able to resolve the recorded signal at least to such an extent, that the decay is represented by some tens of samples. This work is very hard to get result with ordinary electronic transient recorders. It's too weak to sample analog time decay for the light. Indeed the signal may consist of just a few photons per excitation for one emission cycle. Then the discrete nature of the signal itself prohibits analog sampling. Due to

collection optic losses, spectral limits of detector sensitivity or photo-bleaching at higher excitation power, the limits is still there, even if one has some reserve to increase the excitation power to obtain more fluorescence light [1].

The solution for both problems is Time-Correlated Single Photon Counting (TCSPC). With periodic excitation, from a laser; it is possible to extend the data collection over multiple cycles and one can reconstruct the single cycle decay profile from single photon events collected over many cycles. The method is based on the repetitive precisely timed registration of single photons of a fluorescence signal. The reference for the timing is the corresponding excitation pulse. Provided that the probability of registering more than one photon per cycle is low, the histogram of photon arrivals per time bin represents the time decay one would have obtained from a "single shot" time-resolved analog recording. The precondition of single photon probability can (and must) be met by simply attenuating the light level at the sample if necessary [2].

The diagram illustrates how the histogram is formed over multiple cycles, shown in Fig. 1.1. In the example, fluorescence is excited by laser pulses. The time difference between excitation and emission is measured by electronics that act like a stop watch. If the single photon probability condition is met, there will actually be no photons for several cycles. This situation is shown after the third laser pulse. It should be noted that the occurrence of a photon or an empty cycle is entirely random and can only be described in terms of probabilities. The histogram is collected in a memory on FPGA board, where one memory cell holds the photon counts for one corresponding time bin, which is often called time channels.

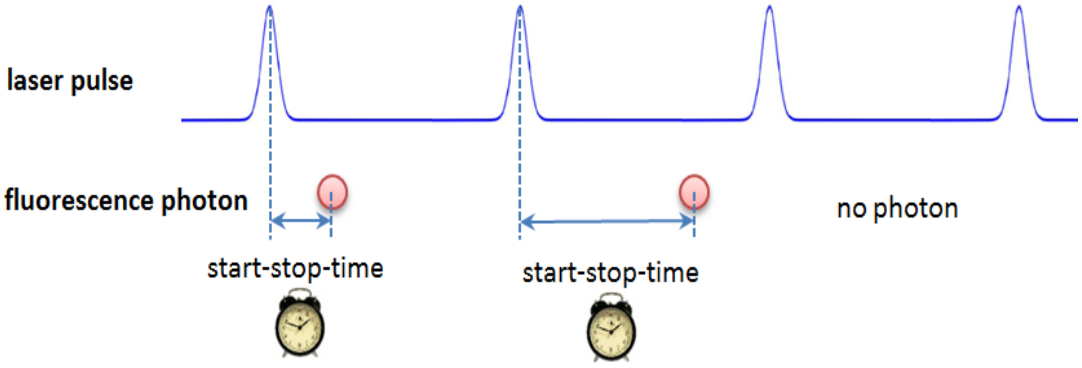


Fig. 1.1 Start-stop-time between laser pulses and photon

We need to record one photon by the single photon detector (SPAD) [3]. The time difference between the photon event and the corresponding laser pulse must be measured, like fluorescence photon.

The timing acquisition process consists of three phases shown in Fig. 1.2. The clock is running all the time, when laser pulse is high, the start signal is high. When photon is gone, the stop signal is high. First, the time interval t_2 between the rising edges of the START signal and the subsequent reference clock edge is measured. Then, a coarse counter measures the time interval t_1 between the two rising edges of the reference clock immediately following the START and the STOP signals. The same procedure is exploited to measure the time interval t_3 between the rising edges of the STOP signal and the subsequent reference clock. The time interval between the START and STOP signals, $T=t_1+ t_2- t_3$. The fine conversion dynamic ranges t_2 and t_3 are limited to only one reference clock cycle [4]. The START and STOP signal come from the single photon detector (SPAD).

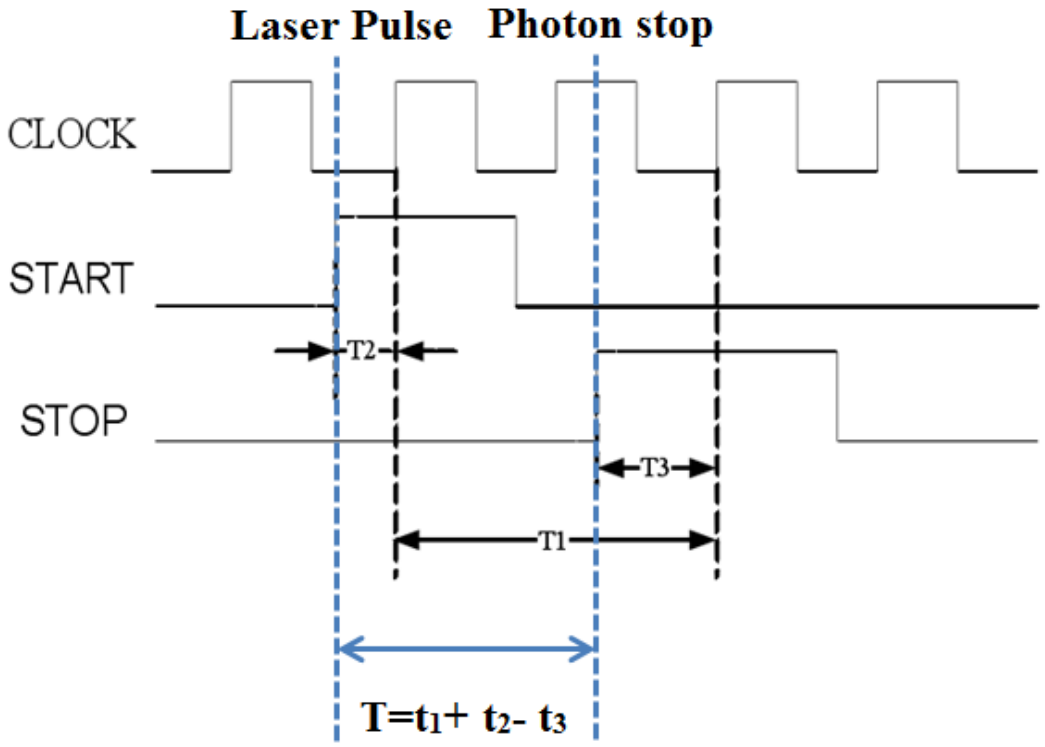


Fig. 1.2 The timing acquisition process consists of three phases

The actual time difference measurement is done by using fast digital timing result electronics. It is then used to lead to the memory in FPGA, so that each timing value corresponds to one memory (histogram bin), shown in Fig. 1.3. All steps are carried out by fast electronics so that the processing time required for each photon event can be as short as possible [5]. The histogram memory can be read out when time of the user set is up.

To get the histogram, X-axis is the result from FPGA counter time, and Y-axis is the number of the photon with the corresponding counter time. For example, with 1 second period time, the histogram will display a result for the corresponding number of the counter time.

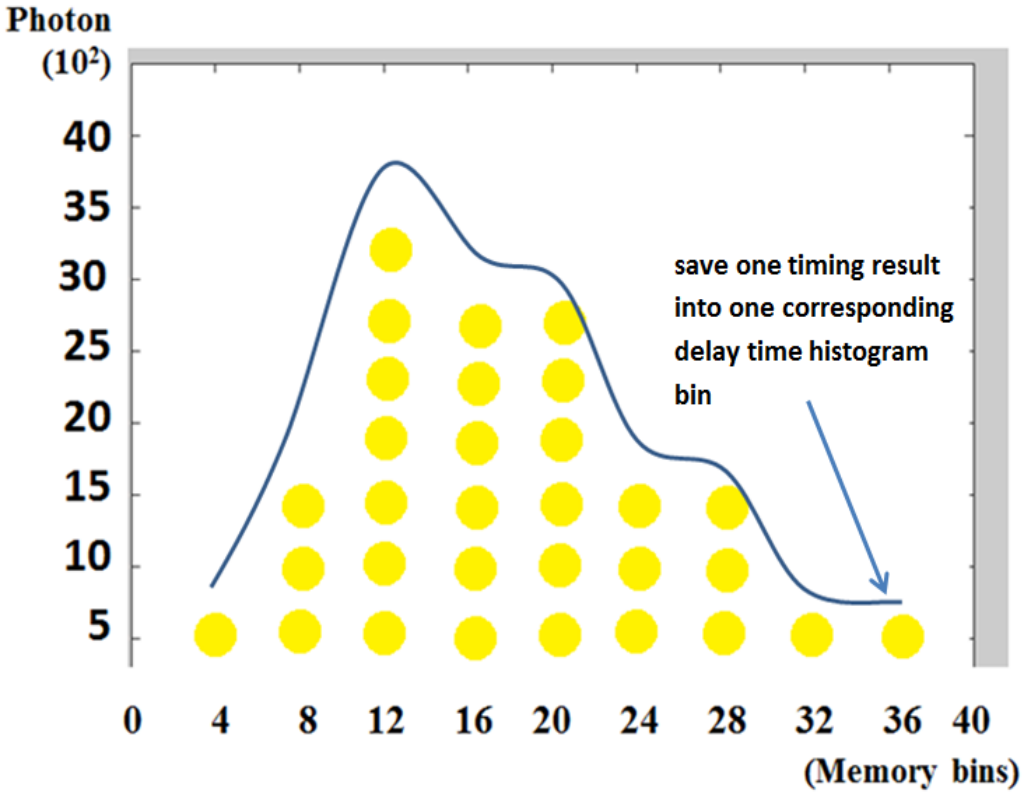


Fig. 1.3 The 40 memory bins histogram in FPGA

2. Accurate Time Difference Measurement

A technic called Time to Digital Converter (TDC) is the solution to get the accurate time difference measurement. For many years the main methods used to achieve the hundreds of picoseconds resolution have been based on time-stretching, Vernier and tapped delay lines. These techniques were designed both in ASIC and Field-Programmable Gate Arrays (FPGA) devices. However, the design process of an ASIC device not only can be expensive, especially if produced in small quantities, but also the design process is complex due to the long turn-around time and layout phase. On the other hand, low cost, fast development cycle and commercial availability are several driving motivations for using general purpose FPGA to implement the TDC without using any external circuits [6].

In this thesis, the TCSPC systems block diagram is shown in Fig. 1.4. We will introduce the TDC design and the method to get the histogram of TCSPC in the following section.

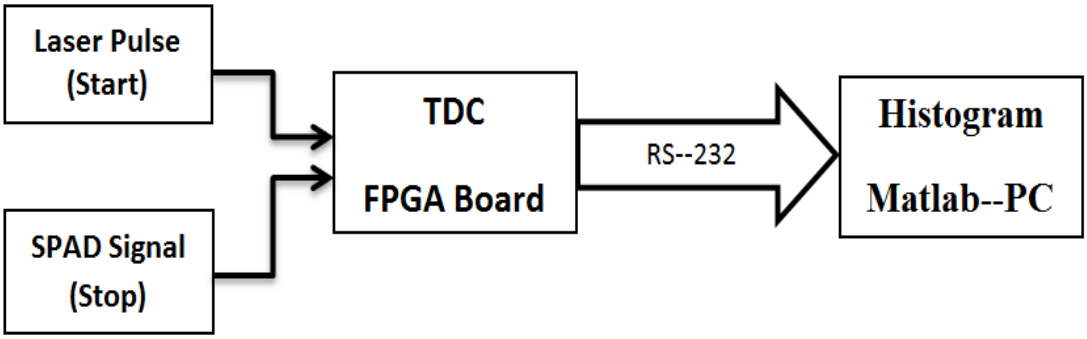


Fig. 1.4 The TCSPC systems

3. Organization of the thesis

There are seven chapters in this thesis. Chapter 1 gives an introduction to the research, including a short summary of a few receiver architectures and significant advantages of TDC converter on FPGA for time correlated single photon counting.

Chapter 2 introduces the theoretical basis of the time to digital measurement. After the introduction to the measurement method, the FPGA structure and function will be introduced. The historical development of the TDC, which will give the development history of TDC based on FPGA, will be presented.

Chapter 3 introduces the design of the TDC stricter. The TDC include delay line fine time counter module, coarse time counter module, encode module, and transfer data module.

Chapter 4 analyzes the optimization of the TDC based on FPGA module. Using three different delay lines for the fine time counter, and get the average of the timing result to get an accurate result for the TDC time correlated single photon counting.

Chapter 5 focuses on simulation of the TDC structure. Using Quartus II 9.0 and ModelSim SE 6.1f to get the simulation result.

In chapter 6, the measurement of the TDC structure will show up. The thesis will use FPGA to get the board level measurement result and analysis the result.

In chapter 7, conclusion will be presented.

Chapter 2

Introduction of Time Difference Measurement

1. The Methods of Time-to-Digital Converter

Because a large number of demand for the measurement of time, there are many time-to-digital conversion circuits come to our research. Depending on the application, each of the time-to-digital converter circuit has its own characteristics and applications [7].

1.1 Counter Method

In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

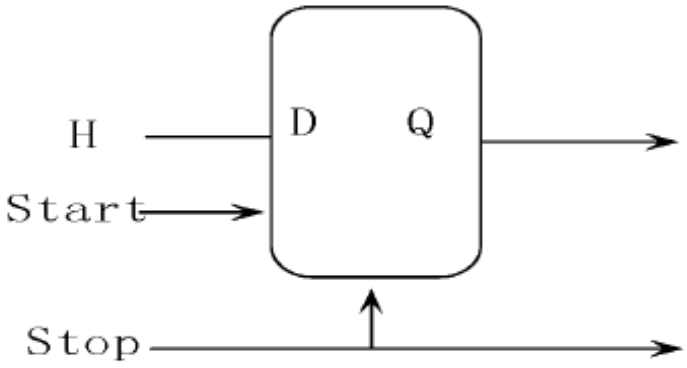


Fig. 2.1 The Counter Method basic structure

Counter technology to achieve the principle of time-to-digital conversion circuit is very simple; the core is a clock counter in the system. The principle of the counter is shown in

Fig. 2.1, so the start signal will get the DFF signal to an adder, and then the counter will work.

1.2 Current Integration Method

The current integration method used earlier in the time interval measurement, Fig. 2.2 shows the basic structure of current integration method.

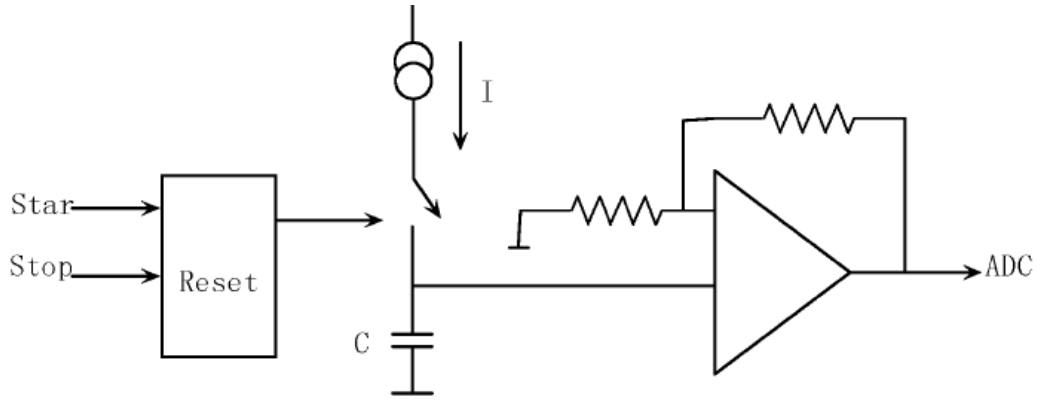


Fig. 2.2 The Current Integration basic structure

The voltage after the capacitor charged is: $V_{cap}=I(t_2-t_1)/C$ [8], $t_2 - t_1$ is the different time for the measurement.

The time to digital converter resolution is very high, mainly due to the constant current source of stability, linearity of the capacitor and leakage current, resolution and accuracy of the ADC chip. Another important point is that the current integration point is highly sensitive to noise.

The problem with the conversion circuit during the measurements is a lot of dead time between the two measurements. There is another negative point for the current integration technology is the dynamic range is not big enough.

1.3 Time Magnify Method

The analog time amplification technology is an extension of the current integration technology. In this scenario, the time interval to be measured is magnified to K times. K is defined as the amplification factor, and its size depends on the parameters of the circuit, shown in Fig. 2.3.

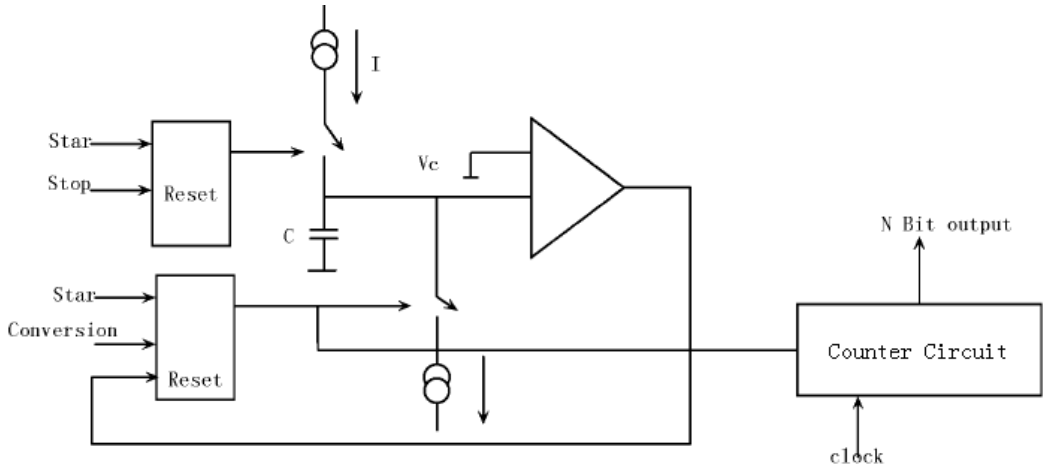


Fig. 2.3 The Time Magnify basic structure

1.4 Vernier Calipers Method

The basic principle of the vernier caliper is to use two clocks of small cycle difference, the difference of two clock cycles as the time unit of measurement. The basic idea comes from the French mathematician Pierre Vernier, who invented vernier caliper. The vernier caliper can also be seen as a time amplification technology. But the Vernier Calipers method is not fit for this project, because this method is not accurate enough.

2. The History of the development of TDC

Much work has been done in this field [9]. In 1995, an FPGA-based approach was proposed by Kalisz, et al. [10]. They made use of the difference between a latch delay and a buffer delay of QuickLogic's FPGA and achieved a time resolution of 100 ps [11]. Already in 1997, Kalisz et al. achieved a resolution of 200 ps on a QuickLogic FPGA [12]. In 2002, Andaloussi, et al. reported a novel TDC architecture based on a two-dimensional time delay matrix [13]. The architecture was implemented in a XCV3000 Virtex FPGA from Xilinx. The resulting TDC allowed for time measurements over a 21 ms range with a 150 ps resolution. In the same year, Fries, et al. realized a TDC in an FPGA using a 192 MHz quadrature clock [14]. This implementation achieved a resolution of better than 1.4 ns. In 2003, a TDC was implemented in an ACEX 1 K FPGA from Altera by Wu, et al. [15]. This TDC used cascade chains of the FPGA and offered a time resolution of about 400 ps. In the same year, Zielinski, et al. implemented a high-resolution time interval measuring system in a single FPGA device [16]. The bin size of the system was 500 ps. In 2004, a TDC with 75 ps single shot resolution was implemented in an FPGA from Xilinx by Xie, et al. [17]. The TDC was based on a counter and a two steps cascading delay line. In the same year, Szymanowski, et al. implemented a high-resolution TDC with two stage interpolators in a QL12X16B from QuickLogic [18]. The TDC had 200 ps resolution and standard measurement uncertainty below 140 ps. Xie, et al. extended in 2005 the previous approaches and achieved a resolution of 75 ps [19].

Wu, et al. implemented the TDL method in an Altera Cyclone II FPGA and proposed two methods to improve the resolution of the TDC beyond the intrinsic cell delay of the chain. The resolution was improved from 40 ps to 25 ps RMS with the first method and

down to 10 ps RMS with the second method but on cost of a larger resource effort and an increased dead time [20]. A TDC with 40 ps resolution based on the Vernier method was implemented by Junnarkar et al. in 2008 [21]. In 2009 a “pure” TDL version was implemented by Favi et al. on a Virtex-5 FPGA that is fabricated in the 65 nm process. On this chip a resolution of 17 ps (standard deviation) was achieved in some placement configurations [22].

3. The Basic Principle and Structure of FPGA [23]

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip.[5] Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which

carries analog values on its internal programmable interconnect fabric. In this thesis, the FPGA we used is Altera DE2-70, shown in Fig. 2.4.

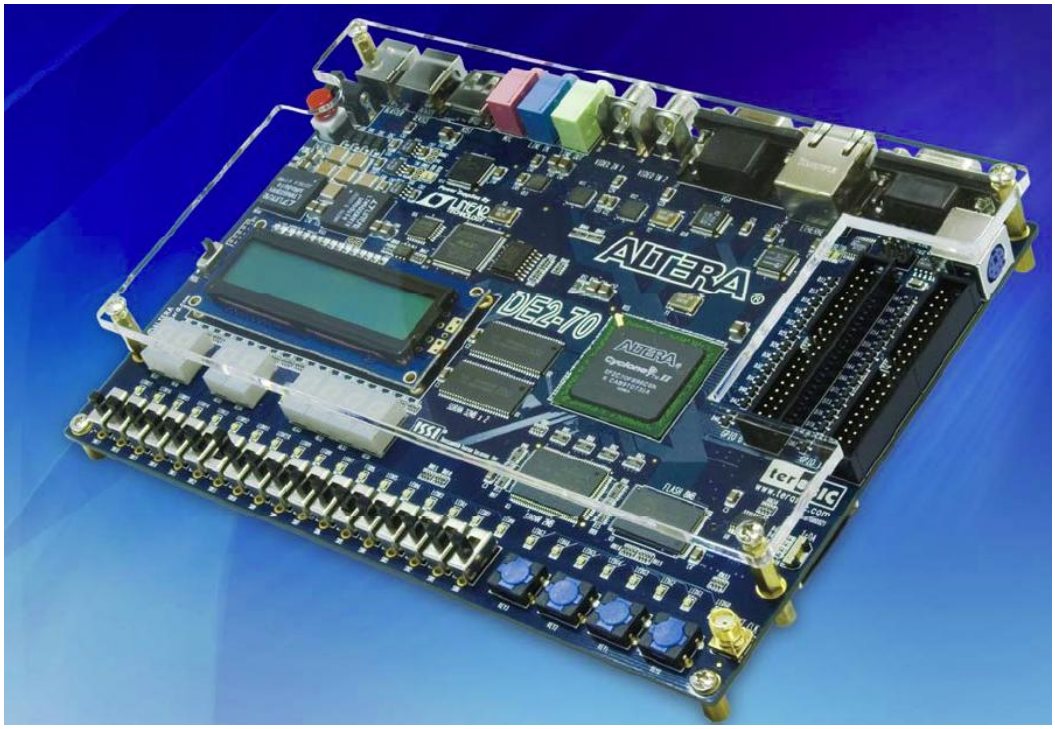


Fig. 2.4 Altera DE2-70

4. Use FPGA to Impalement TDC

With the rapid development of microelectronic technology, FPGA (Field Programmable Gate Array) devices are widely used in communication, intelligent control, information processing and suchlike fields. Embedded real-time image processing technology, as a security monitoring and industrial implementation of the smart key components, is becoming one of these days' hot issues on research because of its wide range of application prospects.

The general embedded system can be divided as follows types: general processors, ASIC, digital signal processor (DSP) and Field Programmable Gate Array (FPGA). General processors is used commonly in general areas of simple image processing tasks, or

applied to system with the custom image processing chip. ASIC which designed for specific Application integrated circuits, can be integrated in one or more processor cores, as well as a special image processing module (such as lens correction, smooth filtering, compression coding, etc.). It can achieve to a higher degree of parallel processing and the highest efficiency. However, long development cycles and high development costs of ASIC make it cannot suitable for the TDC implementation. Reconfigurable system is characterized by the use of reprogrammable devices; it can change the circuit structure on line, which makes the system both flexible and simple. Computing systems Based on this technology has a wide application prospect in TDC implementation.

Table 1 lists the performance comparison of ASIC, reconfigurable computer and microprocessor for implementing algorithms.

Up until recently, TDC circuits have almost exclusively been implemented as application-specific integrated circuits (ASICs), but by the last decade, many implementations were produced on configurable circuits such as field-programmable gate arrays (FPGAs). This is mostly attributable to the dazzling progress of that technology, considerably reducing the domain of circuits that can only be implemented as ASICs. One of the main incentives for FPGA implementation is probably the lower cost and lower design time required. However, since more and more digital systems target programmable fabrics, an FPGA implementation of a TDC can also provide a straightforward way of realizing high precision on-chip measurement [23].

In the project, we use DE2-70 board as the FPGA hardware. Because this FPGA board has many I/O port, which means it can help me to get more information. And the speed of

Altera Cyclone® II 2C70 FPGA device is fit for this project. If I can get a faster FPGA chip, I think the revolution for the TDC can better than this one.

TABLE I. PERFORMANCE COMPARISON

Implementation	PC	Reconfiguration	ASIC
Speed	Slow	Fast	Very fast
Parallelism	Instruction Level	Most	Any
Cost	Small	Small	Big
Scalability	Easy	Normal	Hard

Chapter 3

The TDC Architecture

In the TDC architecture, we use multiple tapped delay lines to make the fine counter more accurate. Using the fine time counter, fine time encoder and coarse time counter to get the timing result, and then we can save it in the RAM on FPGA.

The simplified circuit diagram of the TDC architecture is shown in Fig. 3.1. There are two fine time counters, because the TDC resolution is limited by the ultra-large delay cells, corresponding to the carry chain crossing at the boundaries of the logic array blocks. The apparent widths of these ultra-large delay cells can be several times bigger than the average cell delay width. At the last stage of the TDC, all time measurements are averaged to provide the final measure. And it also has one coarse time counter, encoder, and RAM in the FPGA.

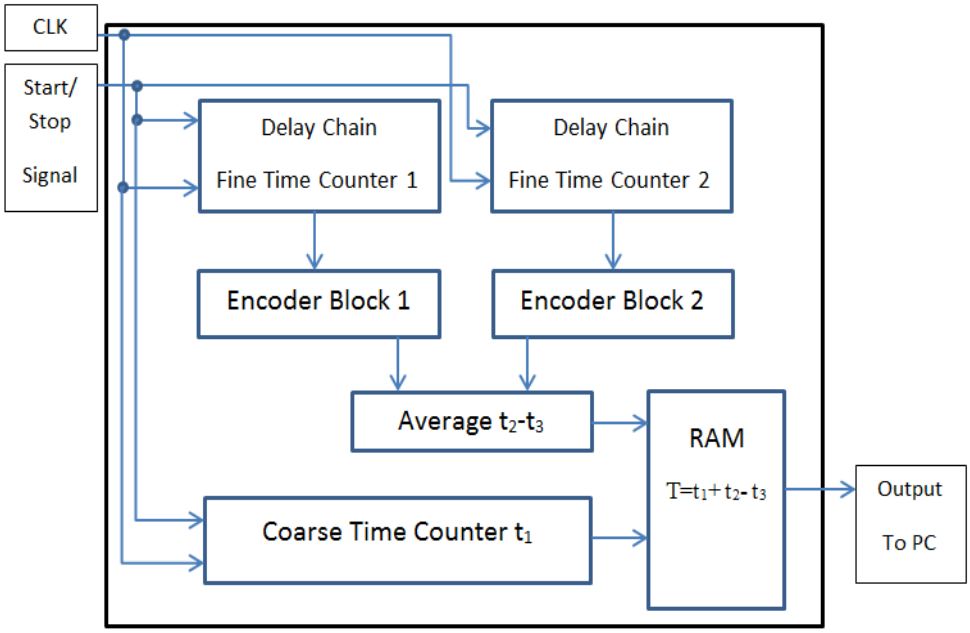


Fig. 3.1 Block diagram of the TDC architecture

1. Multiple Tapped Delay Lines

The tapped delay line method is probably the most intuitive approach to achieve fine time interval measurement. The idea is to tap a delay line at N points, effectively producing delayed copies of the input signal (Fig. 3.2). Ideally, these copies are regularly delayed so that the sampling resolution is increased by a factor N.

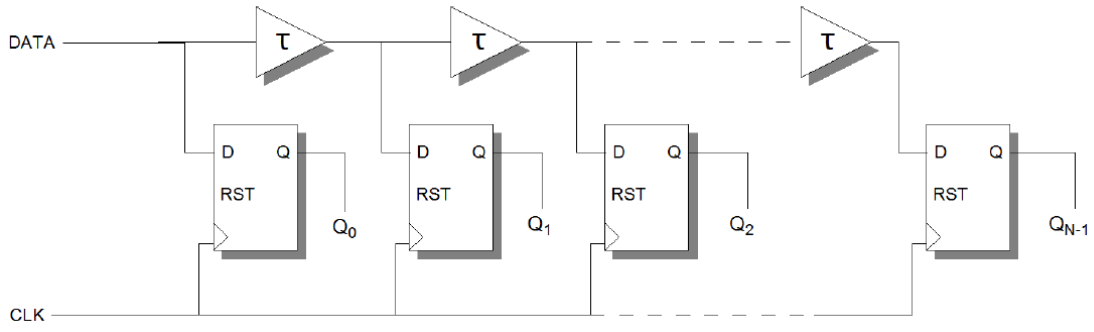


Fig. 3.2 Single tapped delay line architecture

In order to obtain the smallest delays t possible within an FPGA, a widespread approach consists in implementing the delay line with a fast dedicated routing structure such as the carry chain provided for arithmetic circuits. Fig. 3.3 illustrates a possible implementation using an N-bit adder. Initially, the data input is low and each output Sum_n is high. Upon reception of an impulse on the data input, a low level signal starts to propagate along the summation outputs, from LSB to MSB. Then on the next rising edge of the clock signal, the Sum output is latched and the highest index n for which Sum_n is low is used to determine the time interval between the rising edges of data and clk inputs. This index n is typically obtained by connecting the latched Sum output Q to a priority encoder [24].

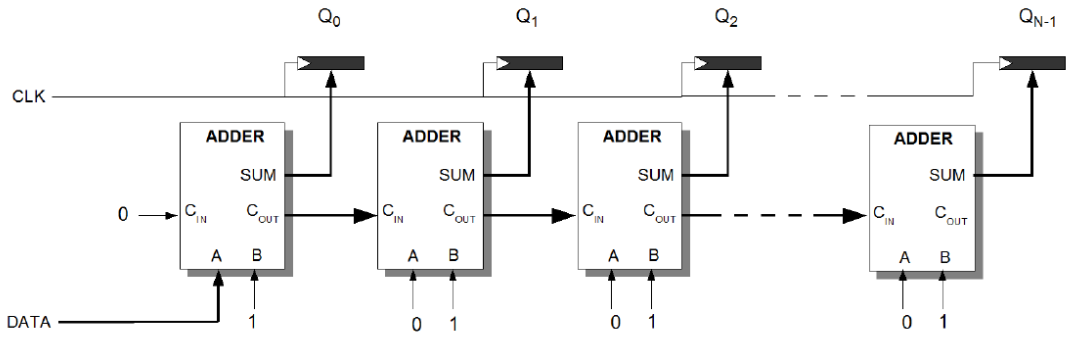


Fig. 3.3 Implementation using the carry chain of an adder circuit

Because of fabrication process variations and irregularities in FPGA architectures, delays t_n are generally not uniform. Moreover, the so small t required by high resolution makes clock skew far from negligible. Therefore, a calibration process is required to take into account this non-linearity and reduce measurement errors. It is possible to estimate the relative size of each division (differential non-linearity) by means of a code density test, in which the TDC is stimulated with K pseudorandom test pulses. The relative size r_n of each division's n against the ideal value of 1LSB can then be estimated by Equation 1. Given that the ideal value of 1LSB is obtained by the ratio T_s/n, where T_s corresponds to the sampling clock period, it is possible to express the time size t_n of each division with the following Equations.

$$r_n = \frac{N \times nbHits(n)}{K}$$

$$t_n = r_n \times \frac{T_s}{N} = \frac{T_s \times nbHits(n)}{K}$$

Once the time sizes t_n of each divisions are estimated, it is possible to provide a more accurate relation between TDC code (from priority encoder) and time interval measurement, C_n to T_n, where T_n is simply given by this Equation:

$$T_n = \sum_{k=0}^{n} t_k$$

As a result, it is rather straightforward to compensate the non-linearity of the tapped delay line by inserting a lookup table, implementing the relation C_n to T_n, at the output of the priority encoder [25].

2. TDC Fine Time Counter Architectures

In the FPGAs available today, there are high-speed chain structures that vendors designed for general-purpose applications. A few well-known examples are carry chains, cascade chains, sum-of-products chains. These chain structures provide short predefined routes between identical logic elements. They are ideal for TDC delay chain implementation. The delay line based TDC measures time difference between the HIT signal and the timing reference clock CLK signal as shown in Fig. 3.4.

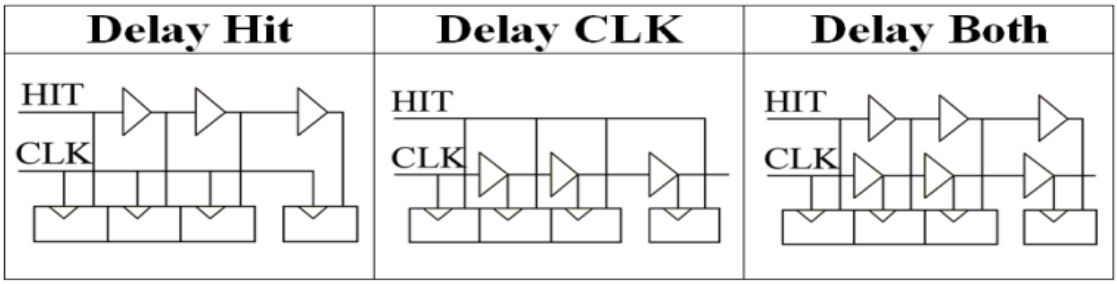


Fig. 3.4 TDC Delay Line architectures

The third delay line uses in this TDC architecture. In the first two delay line, the delay cell is determined only by one transistor, like invertor. While the third delay line is determined by the subtraction of the two delay cells, which means the delay time can be more accurate. The time of the HIT delay cell must shorter than the time of CLK delay cell. In this configuration the STOP signal is the 50 MHz system clock. The START signal after

each delay unit is sampled by the corresponding flip-flop on the rising edge of the STOP signal. In this configuration the delay line is created by the logic gate structure.

The delay line based TDC measures the time difference between the HIT signal and the timing reference clock CLK signal. The TDC can be classified by the signals being delayed and the signals used to clock the register array, as shown in Fig. 3.4.

The only TDC architecture that requires dual counters or Gray code counters is when the HIT signal is used as the clock for the register array. When the HIT signal arrives to record the coarse time, the coarse time counter driven by the CLK may be in an unstable condition and an incorrect time may be recorded.

In this architecture, two counters driven by both edges of CLK or Gray counters are utilized. With dual counters, at least one of them is stable and is selected based on the most significant bit of the fine time. Using a Gray counter, at most one bit is flipping at each clock edge, so the error of the unstable edge is confined in a single bit and can be corrected later.

3. Coarse Time Counter

The coarse counter has a 16 bits data width and is used in free-running mode. When the START signal transition occur the current state of the counter is sampled by the START register, and the same operation occurs also when the STOP signal is delivered to the TDC. The difference between the STOP and START register is the coarse measurement of the time interval, as shown in Fig. 3.5.

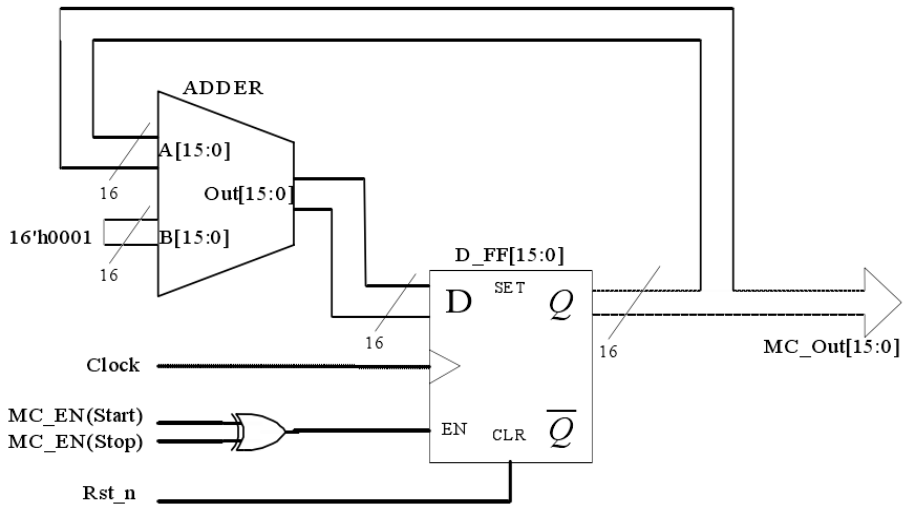


Fig. 3.5 Coarse time counter

FPGA-based TDCs use the architecture in which the HIT is delayed and CLK is used as the register array clock. In this case, the coarse time counter can be a plain binary counter and is implemented as shown in Fig. 3.5.

The input hit is recorded in the register array and the location is encoded as a fine time bin number. Note that the uncertainty of the relative timing between the hit and the CLK is confined in the register array which is the value to be measured by the TDC. All other signals are derived from the output of the register array and their timing is well-defined by the CLK.

While the fine time is being encoded, a data valid signal is generated by the hit detect logic. The simplest hit detect logic senses the logic level difference between both ends of the register array so that a data valid signal is generated for the clock cycle when the wave union is inside the array. This data valid signal is used to enable writing both the fine time and the coarse time into the temporary storage RAM. Note that only at the zero-suppression stage, i.e., when writing the valid hits into the temporary storage, the coarse time is

concatenated with the fine time. There may be other pipeline stages to process the fine time before saving it into the temporary storage and it is not necessary to introduce the coarse time in those pipeline stages. The setup and hold times are guaranteed since both the coarse time counter and the RAM block are driven by the same clock signal CLK.

4. Fine Time Encoder Block

The data from fine time counter, as discussed above, is not the final result; it needs an encoder to convert the original data to a readable binary data. The encoder will calculate the number of the DFF, which has changed from 0 to 1. And then output the data corresponding to the timing data of the signal. The fine_encoder is the name of the fine time encoder block, which has one input and one output, shown in Fig. 3.6. The input signal nDFF is the number of the DFF from one clock period, while the output nTIME is the time information for the Matlab use. Each DFF means 200ps in the delay line, so the time result output is based on this number.

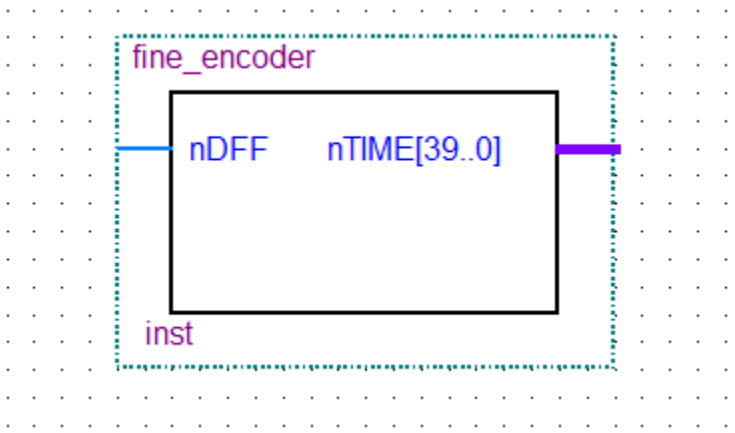


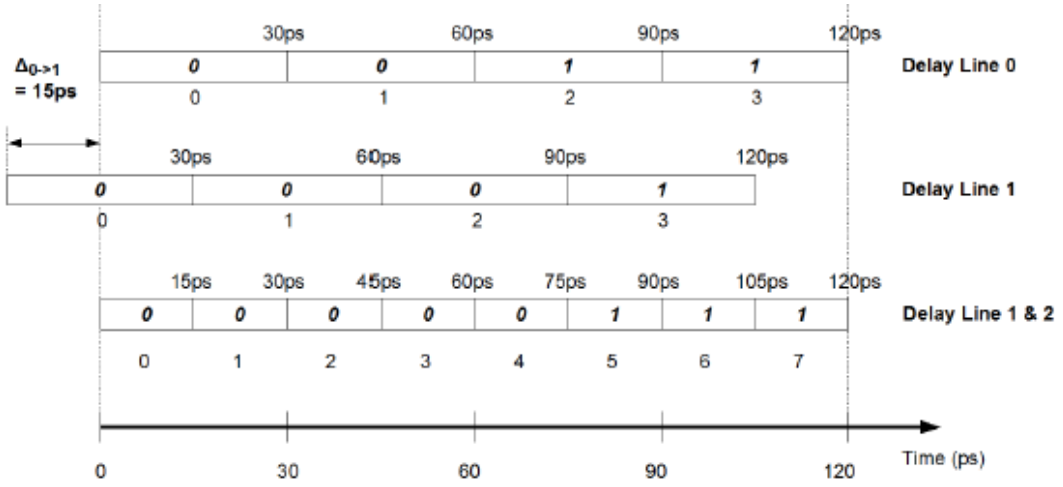
Fig. 3.6 The fine_encoder in the Quartus 2 9.0

Chapter 4

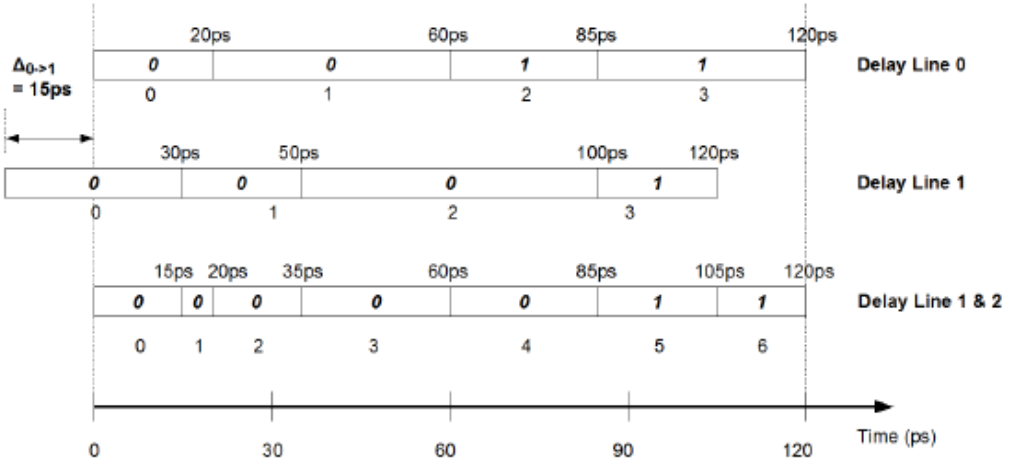
Optimize the TDC Architecture

In this section, we present our proposed TDC architecture using multiple parallel tapped delay lines.

The motivation for using multiple parallel tapped delay lines can easily be understood from a simple example limited to two tapped delay lines. In Fig. 4.1, the time divisions of each tapped delay line are represented by a ruler. Fig. 4.1(a) illustrates the case with ideal tapped-delay lines and input offset value time 1 to 0. Along each ruler, the time T_n of each division is shown and each rectangular division contains the registered Sum_n bit value, Q_n. For each line, we define the division $Time_n$ by the sum of T_n associated to Q_n, in addition to the delay line offset value time 1 to 0. Selecting the highest division $Time_n$ as a final measure, it can be observed that the resolution can be improved by a factor 2. Fig. 4.1(b) illustrates the more realistic case of non-uniform delays with random delay line input offsets. In that example, some resulting divisions can be very small or even lost, resulting in less than one resolution improvement factors. Using more tapped delay lines leads obviously to even better resolutions. In this work, we propose to select the final measure based on the pattern defined by the combination of each division $Time_n$. At this time, the final measure associated to a given pattern is produced by averaging all division $Time_n$ outputs.



(a)



(b)

Fig. 4.1 Example illustrating the use of multiple tapped delay lines

(a) Ideal situation with uniform delays and optimized offset value

(b) Real case featuring tapped delay lines with non-uniform delays

The architecture of the proposed TDC implementing the idea previously illustrated is presented in Fig. 4.2. It features M tapped delay lines implemented as N-bit adders, each followed by a priority encoder to convert the adder's output thermometer code into the largest time division index, n, for which Q_n is at a logical low level. The LUTs following the encoders implement the relation C_n to division $Time_n$, which is obtained by calibration. At the last stage of the TDC, all time measurements are averaged to provide the final measure [25].

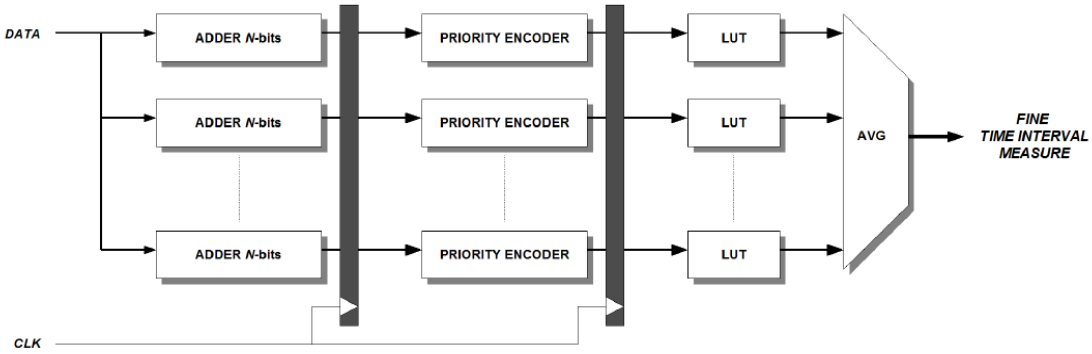


Fig. 4.2 Architecture of the proposed TDC using multiple parallel tapped delay lines.

Chapter 5

Simulation

In this work, we use Altera Cyclone III EP3C25 chip to implement the TDC, simulate the Verilog code by ModelSim SE 6.1f and implement the TDC in Quartus 2 9.0.

1. Introduction to ModelSim

ModelSim is a powerful simulator that can be used to simulate the behavior and performance of logic circuits. The simulator allows the user to apply inputs to the designed circuit, usually referred to as test vectors, and to observe the outputs generated in response. The user can use the Waveform Editor to represent the input signals as waveforms.

ModelSim is a powerful HDL simulation tool that allows you to stimulate the inputs of your modules and view both outputs and internal signals. It allows we to do both behavioral and timing simulation; however, this document will focus on behavioral simulation.

In behavioral simulation, we can write models that will not necessarily synthesize. This code cannot be synthesized, but it is intended to give a true reflection of the behavior of the memory chip so that you can test whether your memory controller is functioning properly.

The ModelSim SE 6.1f is shown in Fig. 5.1.

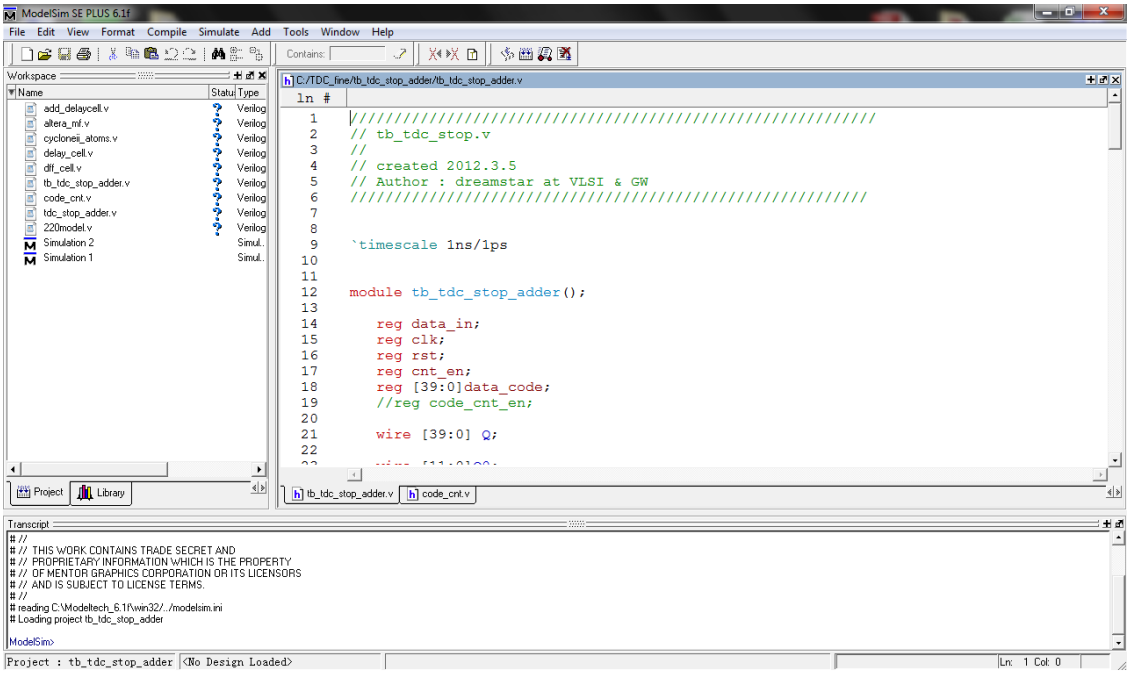


Fig. 5.1 The ModelSim SE 6.1f in the design

We need to design the test code for the original Verilog code, and run the simulation to ge the wave form.

2. The Result of the Simulation

After the simulation, we get the result for both coarse counter and fine counter, shown in Fig. 5.2. In this result: start_input is the start signal, stop_input is the stop signal, q_output is the result for the counter. After the start signal becomes to '1', the counter begins to count, and this process will stop until the stop signal appear.

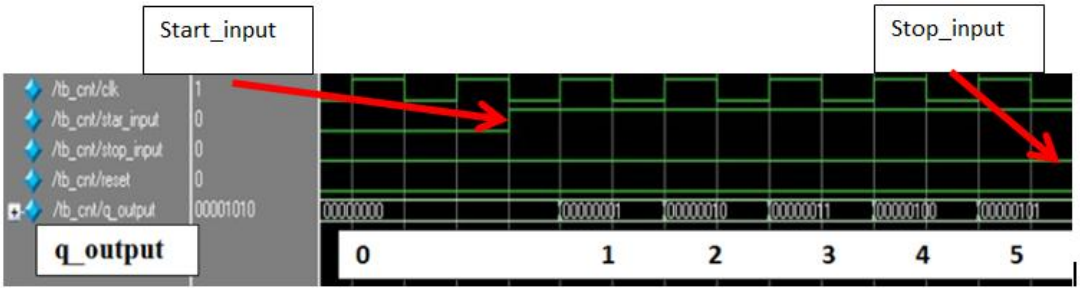


Fig. 5.2 Coarse counter simulation result

The fine counter timing simulation result is shown in TABLE II. Using the "Delay Both" structure in Fig. 3.4, we can get the delay cell data from ModelSim, and the time gap is about 133 ps, based on the simulation result.

TABLE II. DELAY LINE CELL SIMULATION

No.	**Delay Cell T_1 (ps)**	**Delay Cell T_2 (ps)**	**T_1-T_2 (ps)**
1	3810	3674	136
2	3805	3674	131
3	3805	3674	131

Chapter 6

Measurement

In this work, the TCSPC setup is shown in Fig. 6.1. We used signal generator and the PerkinElmer SPCM-AQR SPAD, as the FPGA board input for the START and STOP signal. And then connect the FPGA board to a PC by RS--232 port. Finally, the photon histogram results stored in the on-board RAM are loaded to the PC and analyzed by using Matlab.

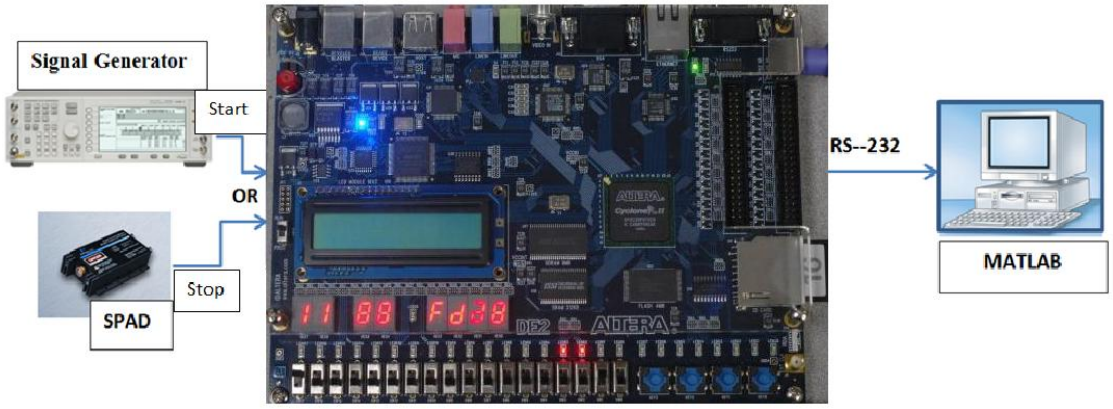


Fig. 6.1(a) TCSPC Measurement architecture

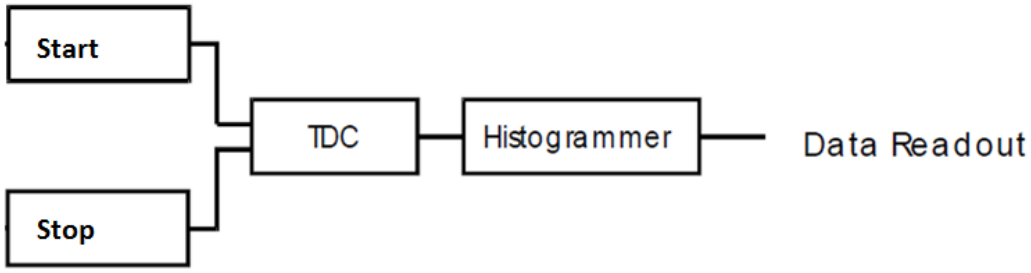


Fig. 6.1(b) The block diagram of TCSPC Measurement

Matlab will analysis the data, which is obtained from the RAM in FPGA. Then it will plot every bin's count in a frame of axis based on the total time we set for test period. Also, it will calculate the total photon appear in this test period [26].

1. Quartuas II 9.0

Quartus II by Altera is a PLD Design Software which is suitable for high-density Field Programmable Gate Array (FPGA) designs, low-cost FPGA designs, and Complex Programmable Logic Devices CPLD designs.

The Quartus II system includes full support for all of the popular methods of entering a description of the desired circuit into a CAD system.

Computer Aided Design (CAD) software makes it easy to implement a desired logic circuit by using a programmable logic device [27], such as a field-programmable gate array (FPGA) chip. A typical FPGA CAD flow is illustrated in Figure 6.2.

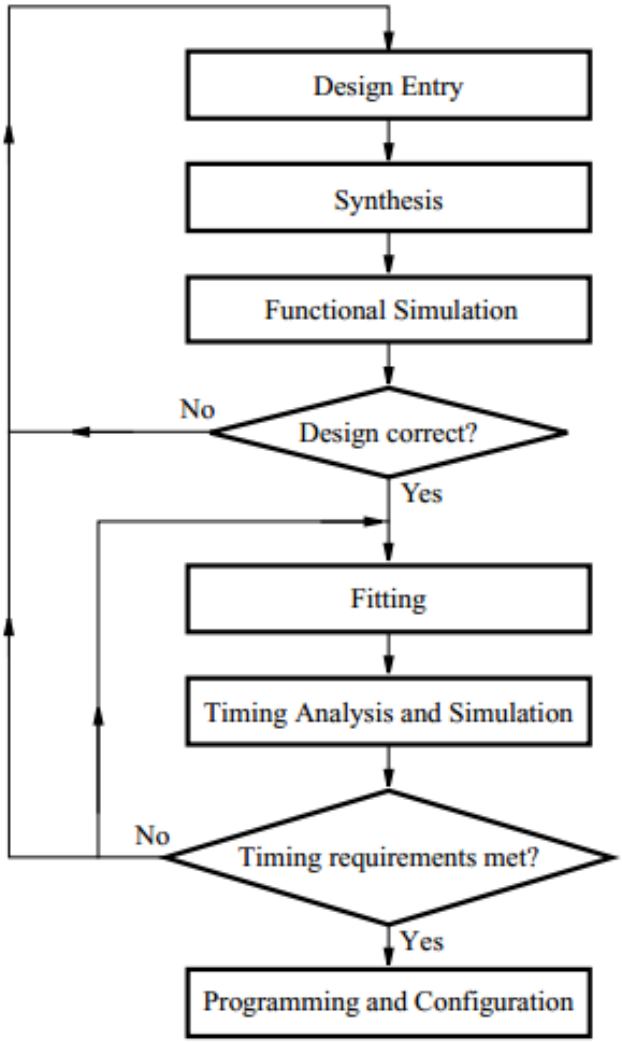


Fig. 6.2 Typical CAD flow

The CAD flow involves the following steps:

• Design Entry – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL

• Synthesis – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip

• Functional Simulation – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues

• Fitting – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs

• Timing Analysis – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit

• Timing Simulation – the fitted circuit is tested to verify both its functional correctness and timing

• Programming and Configuration – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections

2. The Operation of the TDC

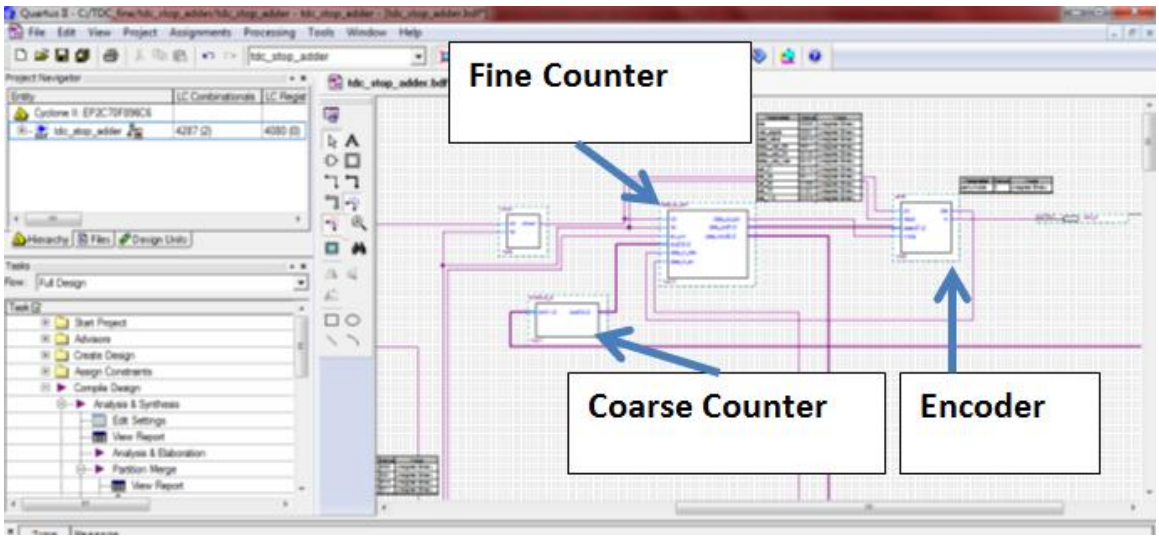


Fig. 6.3 The block diagram in Quartus 2 software

The TDC has three important modules, fine counter, coarse counter, and encoder, shown in Fig. 6.3. Each of them is described in the chapter 3, and I need to put them together to become a system.

When the start signal comes into the FPGA, the fine counter and coarse counter will begin to count. After the stop signal, the counter will output the information to encoder. And then FPGA will save the time information into the RAM on FPGA board. This is one process cycle of the TDC work. After the counter finish one cycle, it will reset to '0', and wait for the next signal input.

For the TCSPC system, I need to count the photon number whit in a fix time period, like 10s. So after the system finish 10s counting work, it will sent the data to PC by RS-232 port. PC will read the data and use Matlab to build a histogram for TCSPC.

In the Matlab environment, I use RS-232 port to get the FPGA data, and then draw a histogram, shown in Fig. 6.4.

```
for j=1:test_time
  for i=1:(histgram_period/time_resolution)
  fprintf(s, '*IDN?');
  Count(i)= fscanf(s,'%u');
  end
  plot((1:(histgram_period/time_resolution))*5,Count,'--rs');
  Count_sum(j) =sum(Count);
  %plot(j,Count_sum(j),'*');
  %hold on;
  pause(accumulation_time);
end
```

Fig. 6.4 One code in Matlab to draw histogram

The code is one part in Matlab to draw the histogram. After PC get the data from RS-232 port, it will become a histogram in the next step.

3. The Test Environment

3.1 The test environment

Fig. 6.1 is the test environment in this project. Because I need to know the resolution for the TDC, so I need to measure the delay cell in the fine counter. In one clock cycle, the signal will go through some delay cells, which is the max delay cell number. In this situation, I will record the max delay cell number to get the resolution for the delay line.

3.2 The process of the test environment

First, I connected each component in this project. The input for FPGA board is signal generator and SPAD.

Second, I need to do the compilation; the Quartus II can get the summary about the project, shown in Fig. 6.5. This step is to synthesize the Verilog code to a real circuit, and then route, place into the FPGA. After compilation, I can get a .sof file, which is the FPGA download file in Quartus II.

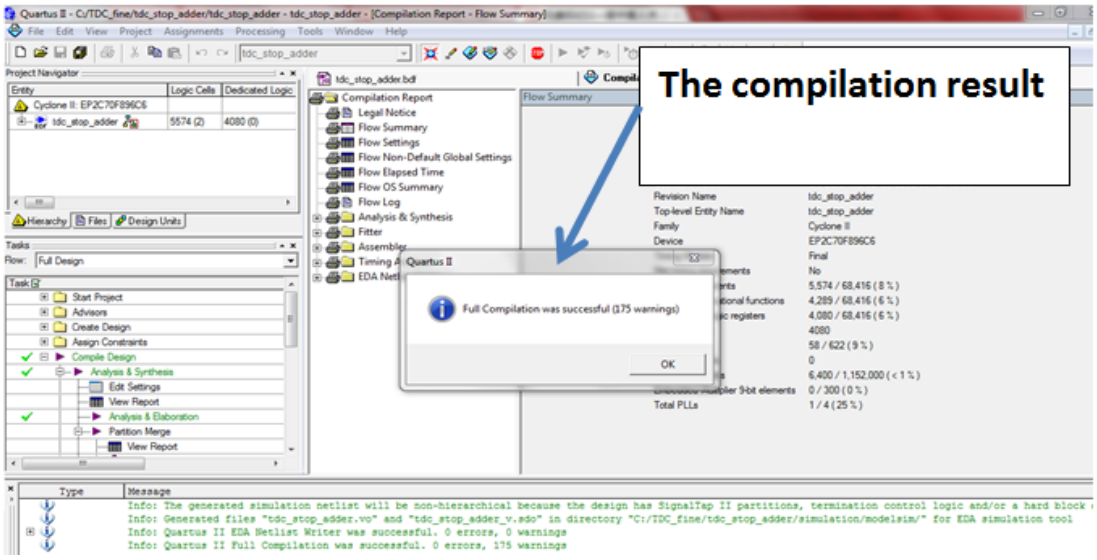


Fig. 6.5 The compilation result in Quartus II

The next step is to get the down load file from the Quartus II. The type of this file is .sof, so I need to down it to the FPGA chip by Quartus II, shown in Fig. 6.6.

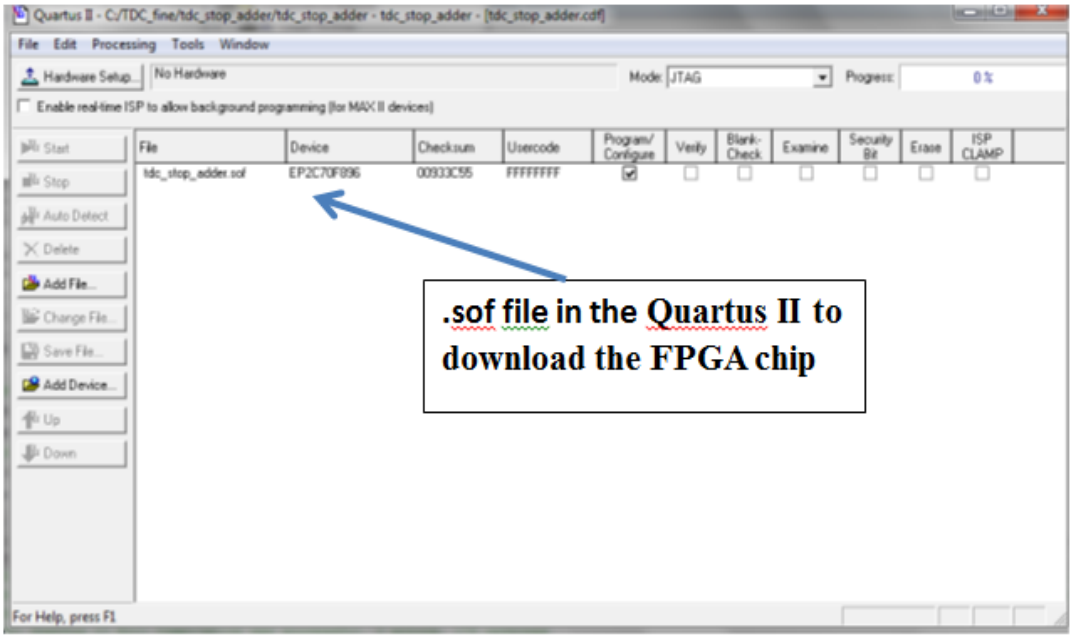


Fig. 6.6 The down load process in Quartus II

After down load the .sof file to the FPGA, the FPGA board can work under the Verilog code environment, shown in Fig. 6.7.

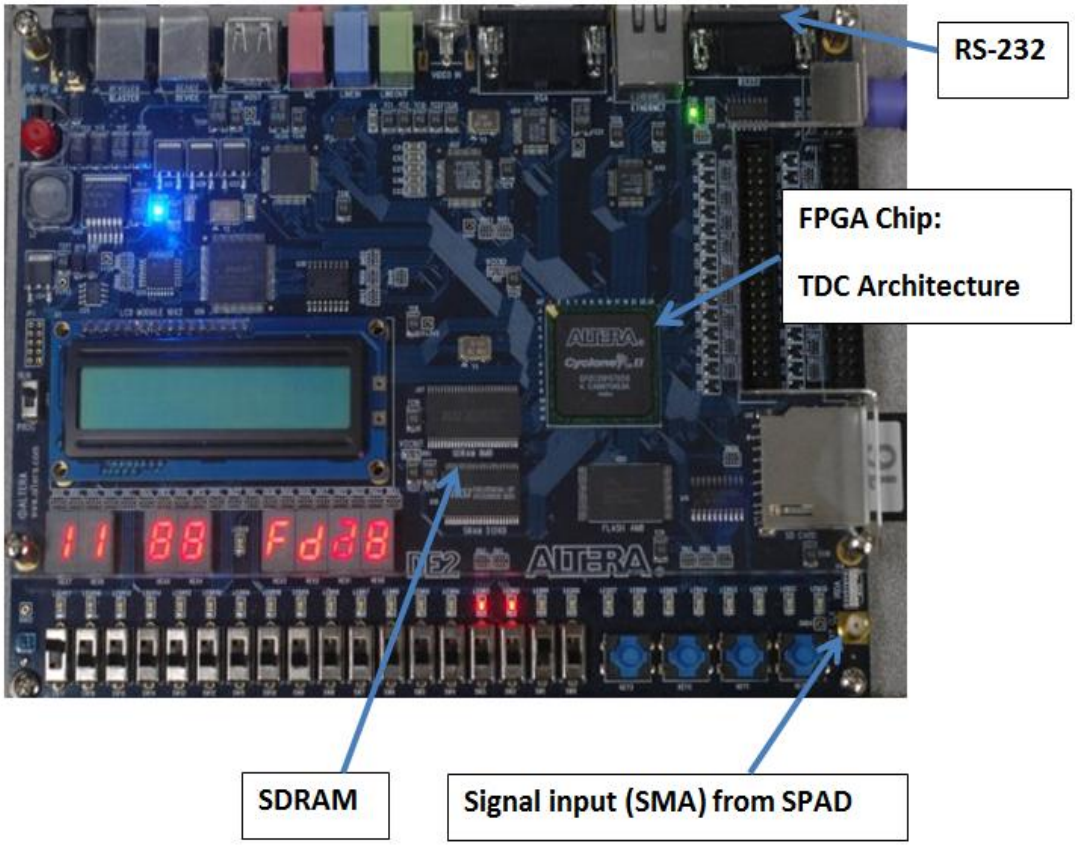


Fig. 6.7 The FPGA board after the Verilog code download

After I set up every component, I start to use signal generator to give the start signal; and I use SPAD to give the stop signal. After that, the histogram will show up in my PC.

4. Test Result

The SignalTap Logic Analyzer Editor can debug the design in real-time and at high-speed while performing an analysis in the Quartus II software. We used this technic as the oscillograph in the testing environment. After opening the SignalTap Logic Analyzer, we need to assign the pin to get waveform, shown in Fig. 6.8.

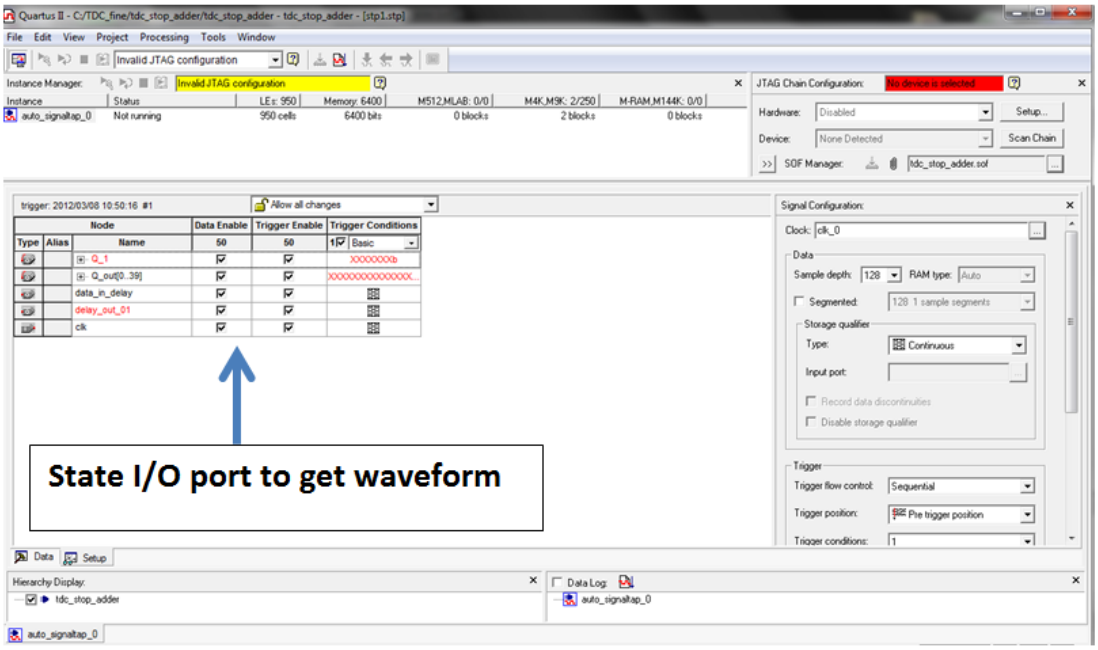


Fig. 6.8 State I/O port to get waveform in The SignalTap Logic Analyzer

We tested the delay cell by one clock signal period using signal generator, shown in Fig. 6.9 (a), which is got from SignalTap Analyzer in Quartus 2. By using the SignalTap Logic Analyzer, I will record the max delay cell number to get the resolution for the delay line.

The clock signal is 125MHz after use PLL in FPGA, and it uses 22 delay cells for one clock period, so we can calculate the delay resolution for each delay cell. In this case, the resolution in Fig. 6.9 (a) is about 200 ps.

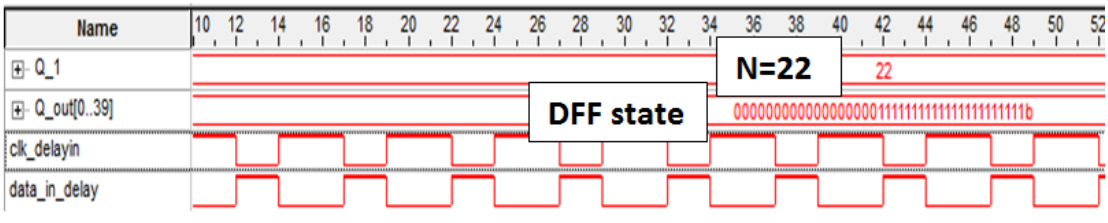


Fig. 6.9 (a) The delay cell result from SignalTap Analyzer in Quartus II

Because the ‘1’ in time period for 125MHz is 4ns, and there are 22 delay cells in the period, so each of them must has 181.8ps to delay the signal:

$$Resolution\ Time = \frac{1}{2f * N}$$

To compare the two designs in Chapter 4, one is before the optimization, and another is after the optimization. The TDC system timing result before the optimization is shown in Fig. 6.9 (b).

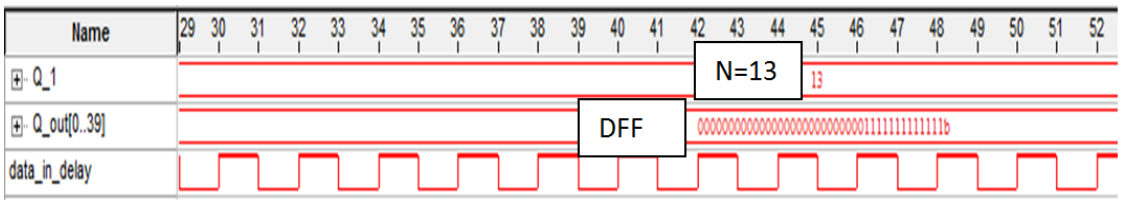
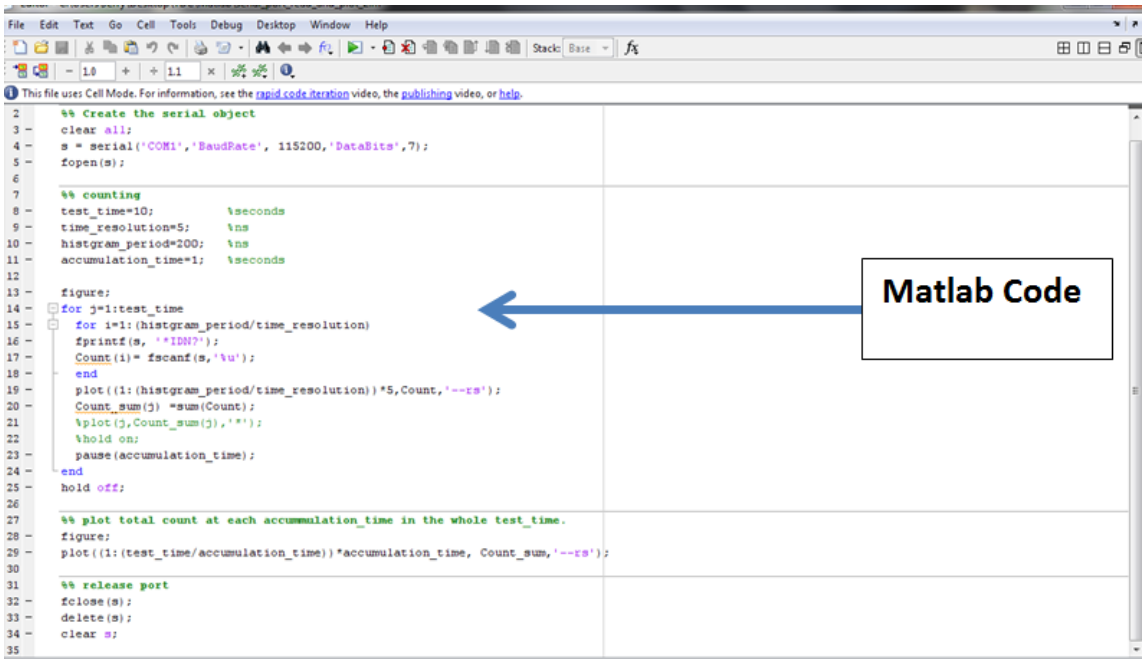


Fig. 9 (a) The TDC system timing result before the optimization

After that, I test a random signal with SPAD, and we get the histogram for about 1s each stop in the Matlab. By using Matlab, I can get this histogram. In the Matlab environment, I use RS-232 port to get the FPGA data, and then draw a histogram, shown in Fig. 6.10.




```
%% Create the serial object
clear all;
s = serial('COM1','BaudRate', 115200,'DataBits',7);
fopen(s);

%% counting
test_time=10;           %seconds
time_resolution=5;      %ns
histgram_period=200;    %ns
accumulation_time=1;    %seconds

figure;
for j=1:test_time
  for i=1:(histgram_period/time_resolution)
  fprintf(s, '*IDN?');
  Count(i)= fscanf(s,'%u');
  end
  plot((1:(histgram_period/time_resolution))*5,Count,'--rs');
  Count_sum(j) =sum(Count);
  %plot(j,Count_sum(j),'*');
  %hold on;
  pause(accumulation_time);
end
hold off;

%% plot total count at each accummulation_time in the whole test_time.
figure;
plot((1:(test_time/accumulation_time))*accumulation_time, Count_sum,'--rs');

%% release port
fclose(s);
delete(s);
clear s;
```

Fig. 6.10 The code in Matlab to draw histogram

The actual time difference measurement is done by using fast digital timing result electronics. It is then used to lead to the memory in FPGA, so that each timing value corresponds to one memory (histogram bin). The histogram memory can be read out when time of the user set is up. By using this technic, the histogram of photon arrivals per time bin represents the time decay one would have obtained from a “single shot” time-resolved analog recording. The precondition of single photon probability can (and must) be met by simply attenuating the light level at the sample if necessary.

The counting number for 1s comes from the RAM in FPGA, shown in Fig. 6.11(a). Each number in X-axis is the one memory bin, which is the delay cell number. The Y-axis is the total counter number for each bin with 1s.

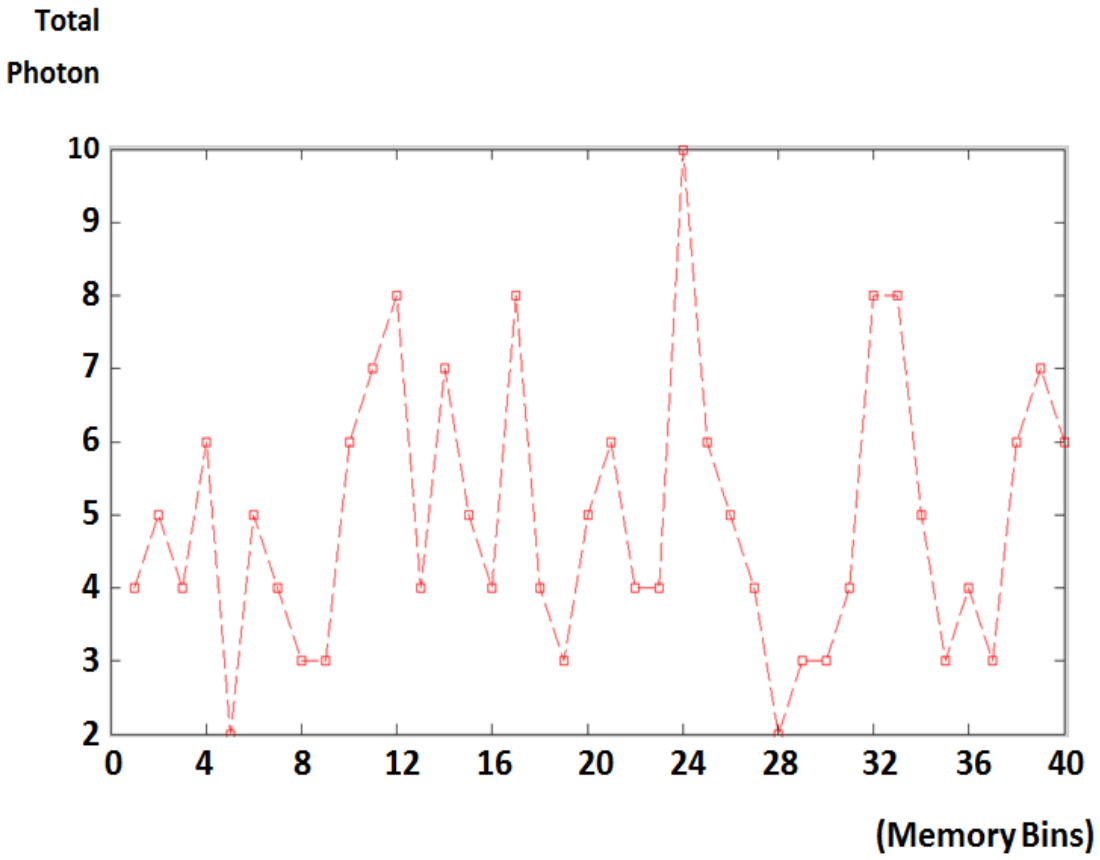


Fig. 6.11(a) The counting number for 1s from SAPD

FPGA sent the data to PC once per second, so we can get the total photon number for a customer set time, such as 10s period as showed in Fig. 6.11(b). Each X-axis number is the time from 1 to 10 second. This histogram is a 10 second record for the photon counting. Because the max frequency of dark count is 250Hz [28], so we get the right number.

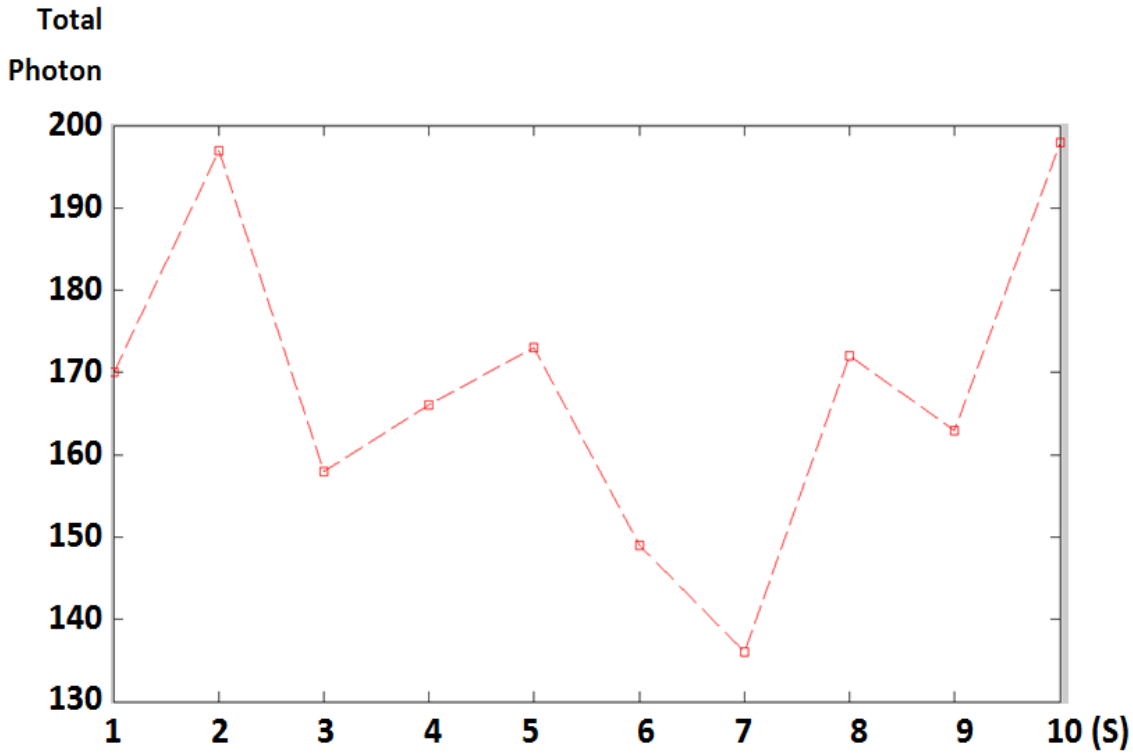


Fig. 6.11(b) The histogram about total photon number from SAPD

5. Frequency Influence

Because there are two elements, F and N, in the equation:

$$Resolution\ Time = \frac{1}{2f * N}$$

So I need to know the clock frequency influence. And the number of the delay cells will also change based on the different clock frequency.

Here is the result table about the change of frequency, shown in table 3.

TABLE III, CLOCK FREQUENCY, NUMBER OF DELAY CELLS, AND THE TIME RESOLUTION

Frequency (MHz)	N (delay cells)	Time resolution (PS)
75	35	190.5
100	28	178.6
125	22	181.8
150	18	185.2
175	15	190.5
200	14	178.6
225	12	185.2
250	10	200.0
275	9	202.0
300	9	185.2

And here is the plot for this result, shown in Fig. 6.12.

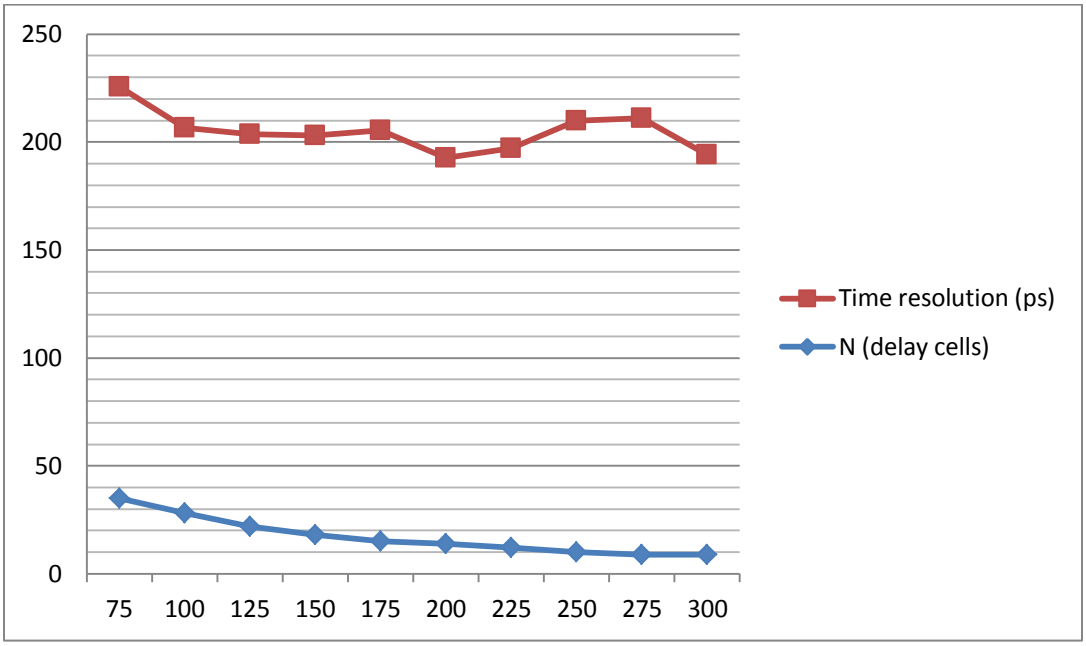


Fig. 6.12 The result for the clock frequency, number of delay cells, and time resolution

The result shows that the resolution time does not change a lot for the different clock frequency. Because the number of the delay cells is based on the different clock. And the real reason for the resolution time is the single delay time in each delay cell.

Chapter 7

Conclusion

The speed of semiconductor devices are becoming faster and faster. This allows us to have high resolution digital counter and short delay elements. Therefore, it is possible to develop a low cost and high resolution TDC exploiting FPGA based techniques. After compare four different methods for TDC structure, we propose to use the counter method as our structure, which has two fine time counters, one coarse time counter, encoder, and RAM. The TDC device uses one coarse counter and two fine counters. The fine counter is a multiple parallel tapped delay line structure, which can make the counter more accurate. The motivation for using multiple parallel tapped delay lines can easily be understood from a simple example limited to two tapped delay lines. Because the parallel tapped delay line structure can get the all-time measurements, and then averaged everyone to provide the final measure.

Time-Correlated Single Photon Counting (TCSPC) is using TDC as the core component. With periodic excitation, from a laser; it is possible to extend the data collection over multiple cycles and one can reconstruct the single cycle decay profile from single photon events collected over many cycles. The method is based on the repetitive precisely timed registration of single photons of a fluorescence signal. The reference for the timing is the corresponding excitation pulse. Provided that the probability of registering more than one photon per cycle is low, the histogram of photon arrivals per time bin represents the time decay one would have obtained from a “single shot” time-resolved analog recording. The

precondition of single photon probability can (and must) be met by simply attenuating the light level at the sample if necessary.

In this thesis, we use Altera Cyclone III EP3C25 chip to implement the TDC, the communication between the FPGA board and PC by using RS-232, which can get the histogram of the TCSPC system. The TDC system is modeled and simulated by the Verilog code under ModelSim SE 6.1f simulation environment and implemented the TDC system in Quartus 2 9.0. In one clock cycle, the signal will go through some delay cells, which is the max delay cell number. In this situation, I will record the max delay cell number to get the resolution for the delay line. In the Matlab environment, it is using RS-232 port to get the FPGA data, and then draw a histogram. In the framework of the TDC implement by using Verilog, we verified and compared the architecture of the delay element by using software simulation and on-board test. In this case, the TDC resolution is below 200ps. This Time-Correlated Single Photon Counting (TCSPC) is an accurate and low cost device to get the fluorescence lifetime and photon density.

Reference

[1] Markovic, B.; Tosi, A.; Zappa, F.; Tisa, S., "Smart-pixel with SPAD detector and time-to-digital converter for time-correlated single photon counting," 23rd Annual Meeting of the IEEE Photonics Society,, pp. 181 – 182, Nov. 2010

[2] Schwartz, D.E.; Charbon, E.; Shepard, K.L., "A Single-Photon Avalanche Diode Array for Fluorescence Lifetime Imaging Microscopy", IEEE Journal of Solid-State Circuits, vol. 43, issue: 11, pp. 2546 – 2557, 2008

[3] Bowei Zhang, Qiuchen Yuan, Zhenyu Li, Mona E. Zaghloul, "Single Photon Avalanche Diode in standard CMOS 0.5um technology," International Semiconductor Device Research Symposium (ISDRS), pp.1-2, Dec. 2011

[4] Haiting Tian ; Fernando, S. ; Hock Wei Soon ; Yajun Ha ; Nanguang Chen, "Design of a high speed pseudo-random bit sequence based time resolved single photon counter on FPGA," International Conference on Field Programmable Logic and Applications, pp. 583 – 586, Sept. 2008

[5] Wahl, Michael; Rahn, Hans-Jurgen; Rohlicke, Tino; Kell, Gerald; Nettels, Daniel; Hillger, Frank; Schuler, Ben; Erdmann, Rainer, "Scalable time-correlated photon counting system with multiple independent input channels," Review of Scientific Instruments, vol 79 , Issue:12 ,pp. 123113 - 123113-8, 2008

[6] J. Wu, Z. Shi, and I. Y. Wang, "Firmware-only implementation of time-to-digital converter in field programmable gate array," IEEE Nuclear Science Symposium Conference Record, vol. 1, 2003, pp. 177-181

[7] B. K. Swann, B. J. Blalock, L. G. Clonts, D. M. Binkley, J. M. Rochelle, E. Breeding, and K. M. Baldwin, “A 100 ps time-resolution CMOS time-to-digital converter for positron emission tomography imaging applications,” IEEE J. Solid-State Circuits, vol. 39, no. 11, pp. 1839–1852, Nov. 2004.

[8] K. Karadamoglou, N. P. Paschalidis, E. Sarris, N. Stamatopoulos, G. Kottaras, and V. Paschalidis, “An 11-bit high-resolution and adjustablerange CMOS time-to-digital converter for space science instruments,” IEEE J. Solid-State Circuits, vol. 39, no. 1, pp. 214–222, Jan. 2004.

[9] K. Maatta and J. Kostamovaara, “A high-precision time-to-digital converter for pulsed time-of-flight laser radar applications,” IEEE Trans. Instrum. Meas., vol. 47, no. 2, pp. 521–536, Apr. 1998.

[10] H. Matsumoto, O. Sasaki, K. Anraku, and M. Nozaki, “Low power high resolution TDC with fast data conversion for balloon-borne experiments,” IEEE Trans. Nucl. Sci., vol. 43, pp. 2195–2198, Aug. 1996.

[11] F. Bigongiari, R. Roncella, R. Saletti, and P. Terreni, “A 250-ps time resolution CMOS multihit time-to-digital converter for nuclear physics experiments,” IEEE Trans. Nucl. Sci., vol. 46, no. 4, pp. 73–77, Apr. 1999.

[12] A. Mantyniemi, T. Rahkonen, and J. Kostamovaara, “A nonlinearitycorrected CMOS time digitizer IC with 20 ps single-shot precision,” in Proc. IEEE Int. Symp. Circuits yst., 2002, pp. I-513–I-516.

[13] S. Minutoli and E. Robutti, “A 96-channel, 500 ps resolution TDC board for the BaBar experiment at SLAC,” IEEE Trans. Nucl. Sci., vol. 47, no. 2, pp. 147–150, Apr. 2000.

[14] R. Arcidiacono et al., “A new drift chamber TDC readout for the high intensity program of the NA48 experiment,” Nucl. Instum. Methods, vol. A518, pp. 493–494, Feb. 2004.

[15] “Preliminary design report of BEPCII—BESIII detector,” in Institute of High Energy Physics of Chinese Academy of Sciences: BESIII International Collaboration, 2004 , pp. 99–145.

[16] J. Kalisz, “Review of methods for time interval measurements with picosecond resolution,” Metrologis, vol. 41, pp. 17–32, 2004.

[17] J. Doernberg, H.-S. Lee, and D. A. Hodges, “Full-speed testing of A/D converters,” EEE J. Solid-State Circuits, vol. SSC-19, no. 6, pp. 820–827, Dec. 1984.

[18] “Dept. Elect. Eng. Comput.,” Ph.D. dissertation, Tech. Univ. Lisbon, Lisbon, Portugal, Oct. 2000.

[19] J. Wu, Z. Shi, and I. Y. Wang, “Firmware-only implementation of time-to-digital converter in field programmable gate array,” in Proc. IEEE Conf. Rec. NSS., vol. 1, 2003, pp. 177–181.

[20] M. Zielinski, D. Chaberski, M. Kowalski, R. Frankowski, and S. Grzelak, “High-resolution time-interval measuring system implemented in single FPGA device,” Measurement, vol. 35, pp. 311–317, 2004.

[21] D. K. Xie, Q. C. Zhang, G. S. Qi, and D. Y. Xu, “Cascading delay line time-to-digital converter with 75 ps resolution and a reduced number of delay cells,” Rev. Sci. Instrum., vol. 76, 2005. 014 701.

[22] R. Szymanowski and J. Kalisz, “Field programmable gate array time counter with two-stage interpolation,” Rev. Sci. Instrum., vol. 76, 2005.

[23] Altera Corporation, “Cyclone II Device Handbook”, (2007) available via: {http://www.altera.com/}

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 2nd ed. New York: McGraw-Hill, 2001, pt. 12.

[25] Marc-Andre Daigneault and Jean Pierre David, “A High-Resolution Time-to-Digital Converter on FPGA Using Dynamic Reconfiguration,” IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 60, NO. 6, JUNE 2011

[26] C. Ljuslin, J. Christiansen, A. Marchioro, and O. Klingsheim, “An integrated 16-channel CMOS time to digital converter,” IEEE Trans. Nucl. Sci., vol. 41, no. 4, pp. 1104–1108, Aug. 1994.

[27] J. Kalisz, M. Pawlowski, and R. Pelka, “A simple, precise, and low jitter delay/gate generator,” Rev. Sci. Instrum., vol. 74, pp. 3507–3509, 2003.

[28] Excelitas Technologies Corporation, “Photon Counting Module Datasheet”, available via: {http:// http://www.excelitas.com/}

Appendix

1. Verilog Code

1.1 Top-level

```
module tdc_stop_adder(
    clk,
    rst,
    cnt_en,
    data_in_start,
    data_in_sttop,
    uart_tx,
    clk_0,
    clk_180,
    clk_delayin,
    data_in_delay,
    delay_out_stop,
    Q_stop
);


input                                   clk;
input                                   rst;
input                                   cnt_en;
```

```
input                 data_in_start;
input                 data_in_sttop;
output                uart_tx;
output                clk_0;
output                clk_180;
output                clk_delayin;
output                data_in_delay;
output                delay_out_stop;
output                [6:0] Q_stop;

wire                  SYNTHESIZED_WIRE_183;
wire                  SYNTHESIZED_WIRE_184;
wire                  SYNTHESIZED_WIRE_2;
wire                  SYNTHESIZED_WIRE_3;
wire                  [15:0] SYNTHESIZED_WIRE_4;
wire                  [11:0] SYNTHESIZED_WIRE_5;
wire                  SYNTHESIZED_WIRE_6;
wire                  SYNTHESIZED_WIRE_8;
wire                  SYNTHESIZED_WIRE_9;
wire                  SYNTHESIZED_WIRE_10;
wire                  SYNTHESIZED_WIRE_11;
wire                  SYNTHESIZED_WIRE_12;
wire                  SYNTHESIZED_WIRE_13;
```

```
wire SYNTHESIZED_WIRE_14;
wire SYNTHESIZED_WIRE_15;
wire SYNTHESIZED_WIRE_16;
wire SYNTHESIZED_WIRE_17;
wire SYNTHESIZED_WIRE_18;
wire SYNTHESIZED_WIRE_19;
wire SYNTHESIZED_WIRE_20;
wire SYNTHESIZED_WIRE_21;
wire SYNTHESIZED_WIRE_22;
wire SYNTHESIZED_WIRE_23;
wire SYNTHESIZED_WIRE_24;
wire SYNTHESIZED_WIRE_25;
wire SYNTHESIZED_WIRE_26;
wire SYNTHESIZED_WIRE_27;
wire SYNTHESIZED_WIRE_28;
wire SYNTHESIZED_WIRE_29;
wire SYNTHESIZED_WIRE_30;
wire SYNTHESIZED_WIRE_31;
wire SYNTHESIZED_WIRE_32;
wire SYNTHESIZED_WIRE_33;
wire SYNTHESIZED_WIRE_34;
wire SYNTHESIZED_WIRE_35;
wire SYNTHESIZED_WIRE_36;
```

```
wire                SYNTHESIZED_WIRE_37;
wire                SYNTHESIZED_WIRE_38;
wire                SYNTHESIZED_WIRE_39;
wire                SYNTHESIZED_WIRE_40;
wire                SYNTHESIZED_WIRE_41;
wire                SYNTHESIZED_WIRE_42;
wire                SYNTHESIZED_WIRE_43;
wire                SYNTHESIZED_WIRE_44;
wire                SYNTHESIZED_WIRE_45;
wire                SYNTHESIZED_WIRE_46;
wire                SYNTHESIZED_WIRE_47;
wire                SYNTHESIZED_WIRE_48;
wire                SYNTHESIZED_WIRE_49;
wire                SYNTHESIZED_WIRE_51;
wire        [6:0]   SYNTHESIZED_WIRE_52;
wire        [7:0]   SYNTHESIZED_WIRE_53;
wire        [11:0]  SYNTHESIZED_WIRE_54;
wire        [11:0]  SYNTHESIZED_WIRE_55;
wire        [11:0]  SYNTHESIZED_WIRE_56;
wire        [11:0]  SYNTHESIZED_WIRE_57;
wire        [11:0]  SYNTHESIZED_WIRE_58;
wire        [11:0]  SYNTHESIZED_WIRE_59;
wire        [11:0]  SYNTHESIZED_WIRE_60;
```

```
wire [11:0] SYNTHESIZED_WIRE_61;
wire [11:0] SYNTHESIZED_WIRE_62;
wire [11:0] SYNTHESIZED_WIRE_63;
wire [11:0] SYNTHESIZED_WIRE_64;
wire [11:0] SYNTHESIZED_WIRE_65;
wire [11:0] SYNTHESIZED_WIRE_66;
wire [11:0] SYNTHESIZED_WIRE_67;
wire [11:0] SYNTHESIZED_WIRE_68;
wire [11:0] SYNTHESIZED_WIRE_69;
wire [11:0] SYNTHESIZED_WIRE_70;
wire [11:0] SYNTHESIZED_WIRE_71;
wire [11:0] SYNTHESIZED_WIRE_72;
wire [11:0] SYNTHESIZED_WIRE_73;
wire [11:0] SYNTHESIZED_WIRE_74;
wire [11:0] SYNTHESIZED_WIRE_75;
wire [11:0] SYNTHESIZED_WIRE_76;
wire [11:0] SYNTHESIZED_WIRE_77;
wire [11:0] SYNTHESIZED_WIRE_78;
wire [11:0] SYNTHESIZED_WIRE_79;
wire [11:0] SYNTHESIZED_WIRE_80;
wire [11:0] SYNTHESIZED_WIRE_81;
wire [11:0] SYNTHESIZED_WIRE_82;
wire [11:0] SYNTHESIZED_WIRE_83;
```

```
wire [11:0] SYNTHESIZED_WIRE_84;
wire [11:0] SYNTHESIZED_WIRE_85;
wire [11:0] SYNTHESIZED_WIRE_86;
wire [11:0] SYNTHESIZED_WIRE_87;
wire [11:0] SYNTHESIZED_WIRE_88;
wire [11:0] SYNTHESIZED_WIRE_89;
wire [11:0] SYNTHESIZED_WIRE_90;
wire [11:0] SYNTHESIZED_WIRE_91;
wire [11:0] SYNTHESIZED_WIRE_92;
wire [11:0] SYNTHESIZED_WIRE_93;
wire [11:0] SYNTHESIZED_WIRE_94;
wire [11:0] SYNTHESIZED_WIRE_95;
wire [11:0] SYNTHESIZED_WIRE_96;
wire [11:0] SYNTHESIZED_WIRE_97;
wire [11:0] SYNTHESIZED_WIRE_98;
wire [11:0] SYNTHESIZED_WIRE_99;
wire [11:0] SYNTHESIZED_WIRE_100;
wire [11:0] SYNTHESIZED_WIRE_101;
wire [11:0] SYNTHESIZED_WIRE_102;
wire [11:0] SYNTHESIZED_WIRE_103;
wire [11:0] SYNTHESIZED_WIRE_104;
wire [11:0] SYNTHESIZED_WIRE_105;
wire [11:0] SYNTHESIZED_WIRE_106;
```

```
wire [11:0] SYNTHESIZED_WIRE_107;
wire [11:0] SYNTHESIZED_WIRE_108;
wire [11:0] SYNTHESIZED_WIRE_109;
wire [11:0] SYNTHESIZED_WIRE_110;
wire [11:0] SYNTHESIZED_WIRE_111;
wire [11:0] SYNTHESIZED_WIRE_112;
wire [11:0] SYNTHESIZED_WIRE_113;
wire [11:0] SYNTHESIZED_WIRE_114;
wire [11:0] SYNTHESIZED_WIRE_115;
wire [11:0] SYNTHESIZED_WIRE_116;
wire [11:0] SYNTHESIZED_WIRE_117;
wire [11:0] SYNTHESIZED_WIRE_118;
wire [11:0] SYNTHESIZED_WIRE_119;
wire [11:0] SYNTHESIZED_WIRE_120;
wire [11:0] SYNTHESIZED_WIRE_121;
wire [11:0] SYNTHESIZED_WIRE_122;
wire [11:0] SYNTHESIZED_WIRE_123;
wire [11:0] SYNTHESIZED_WIRE_124;
wire [11:0] SYNTHESIZED_WIRE_125;
wire [11:0] SYNTHESIZED_WIRE_126;
wire [11:0] SYNTHESIZED_WIRE_127;
wire [11:0] SYNTHESIZED_WIRE_128;
wire [11:0] SYNTHESIZED_WIRE_129;
```

```
wire [11:0] SYNTHESIZED_WIRE_130;
wire [11:0] SYNTHESIZED_WIRE_131;
wire [11:0] SYNTHESIZED_WIRE_132;
wire [11:0] SYNTHESIZED_WIRE_133;
wire [11:0] SYNTHESIZED_WIRE_134;
wire [11:0] SYNTHESIZED_WIRE_135;
wire [11:0] SYNTHESIZED_WIRE_136;
wire [11:0] SYNTHESIZED_WIRE_137;
wire [11:0] SYNTHESIZED_WIRE_138;
wire [11:0] SYNTHESIZED_WIRE_139;
wire [11:0] SYNTHESIZED_WIRE_140;
wire [11:0] SYNTHESIZED_WIRE_141;
wire [11:0] SYNTHESIZED_WIRE_142;
wire [11:0] SYNTHESIZED_WIRE_143;
wire [11:0] SYNTHESIZED_WIRE_144;
wire [11:0] SYNTHESIZED_WIRE_145;
wire [11:0] SYNTHESIZED_WIRE_146;
wire [11:0] SYNTHESIZED_WIRE_147;
wire [11:0] SYNTHESIZED_WIRE_148;
wire [11:0] SYNTHESIZED_WIRE_149;
wire [11:0] SYNTHESIZED_WIRE_150;
wire [11:0] SYNTHESIZED_WIRE_151;
wire [11:0] SYNTHESIZED_WIRE_152;
```

```
wire [11:0] SYNTHESIZED_WIRE_153;
wire [11:0] SYNTHESIZED_WIRE_154;
wire [11:0] SYNTHESIZED_WIRE_155;
wire [11:0] SYNTHESIZED_WIRE_156;
wire [11:0] SYNTHESIZED_WIRE_157;
wire [11:0] SYNTHESIZED_WIRE_158;
wire [11:0] SYNTHESIZED_WIRE_159;
wire [11:0] SYNTHESIZED_WIRE_160;
wire [11:0] SYNTHESIZED_WIRE_161;
wire [11:0] SYNTHESIZED_WIRE_162;
wire [11:0] SYNTHESIZED_WIRE_163;
wire [11:0] SYNTHESIZED_WIRE_164;
wire [11:0] SYNTHESIZED_WIRE_165;
wire [11:0] SYNTHESIZED_WIRE_166;
wire [11:0] SYNTHESIZED_WIRE_167;
wire [11:0] SYNTHESIZED_WIRE_168;
wire [11:0] SYNTHESIZED_WIRE_169;
wire [11:0] SYNTHESIZED_WIRE_170;
wire [11:0] SYNTHESIZED_WIRE_171;
wire [11:0] SYNTHESIZED_WIRE_172;
wire [11:0] SYNTHESIZED_WIRE_173;
wire [11:0] SYNTHESIZED_WIRE_174;
wire [11:0] SYNTHESIZED_WIRE_175;
```

```
wire [11:0] SYNTHESIZED_WIRE_176;
wire [11:0] SYNTHESIZED_WIRE_177;
wire [11:0] SYNTHESIZED_WIRE_178;
wire [6:0] SYNTHESIZED_WIRE_179;
wire SYNTHESIZED_WIRE_181;
wire [7:0] SYNTHESIZED_WIRE_182;

assign clk_0 = SYNTHESIZED_WIRE_183;
assign clk_delayin = SYNTHESIZED_WIRE_49;
assign Q_stop = SYNTHESIZED_WIRE_52;

counter_start b2v_inst(
    .clk(SYNTHESIZED_WIRE_183),
    .rst(rst),
    .en_cnt(cnt_en),
    .start(data_in_start),
    .stop(data_in_sttop),
    .code_en(SYNTHESIZED_WIRE_51),
    .cnt_out(SYNTHESIZED_WIRE_53));
    defparam b2v_inst.idle = 4'b0000;
    defparam b2v_inst.start_rise = 4'b0010;
    defparam b2v_inst.stop_rise = 4'b0011;
    defparam b2v_inst.wait_signal = 4'b0001;
```

```
data_to_uart                                                    b2v_inst10(
    .clk(SYNTHESIZED_WIRE_184),
    .rst(rst),
    .en_cnt(cnt_en),
    .data_in_idle(SYNTHESIZED_WIRE_2),
    .data_in_en(SYNTHESIZED_WIRE_3),
    .bcd(SYNTHESIZED_WIRE_4),
    .data_out_en(SYNTHESIZED_WIRE_181),
    .data_mux(SYNTHESIZED_WIRE_179),
    .data_out(SYNTHESIZED_WIRE_182));
    defparam                                    b2v_inst10.cell_01 = 5'b00110;
    defparam                                    b2v_inst10.cell_02 = 5'b00111;
    defparam                                   b2v_inst10.cell_110 = 5'b01010;
    defparam                                    b2v_inst10.cell_30 = 5'b01000;
    defparam                                    b2v_inst10.cell_70 = 5'b01001;
    defparam                                 b2v_inst10.delay_cell_0A = 5'b00100;
    defparam                                 b2v_inst10.delay_cell_0D = 5'b00011;
    defparam                               b2v_inst10.delay_cell_wait = 5'b00101;
    defparam                                       b2v_inst10.idle = 5'b00000;
    defparam                                  b2v_inst10.read_data = 5'b00010;
    defparam                                b2v_inst10.wait_signal = 5'b00001;
```

```
bintobcd_8                                                    b2v_inst11(
    .bin(SYNTHESIZED_WIRE_5),
    .bcd(SYNTHESIZED_WIRE_4));


pass_enable                                                   b2v_inst13(
    .data_in(SYNTHESIZED_WIRE_6),
    .clk(SYNTHESIZED_WIRE_183),
    .rst(rst),
    .data_out(SYNTHESIZED_WIRE_3));
    defparam                                          b2v_inst13.idle = 4'b0000;
    defparam                                  b2v_inst13.wait_signal = 4'b0001;


pll_cell                                                      b2v_inst14(
    .inclk0(clk),
    .areset(SYNTHESIZED_WIRE_8),
    .c0(SYNTHESIZED_WIRE_183),
    .c1(clk_180),
    .c2(SYNTHESIZED_WIRE_49));
```

```
inv_cell                                                    b2v_inst15(
    .data(rst),
    .result(SYNTHESIZED_WIRE_8));



add01                                                       b2v_inst17(
    .data0x(SYNTHESIZED_WIRE_9),
    .data10x(SYNTHESIZED_WIRE_10),
    .data11x(SYNTHESIZED_WIRE_11),
    .data12x(SYNTHESIZED_WIRE_12),
    .data13x(SYNTHESIZED_WIRE_13),
    .data14x(SYNTHESIZED_WIRE_14),
    .data15x(SYNTHESIZED_WIRE_15),
    .data16x(SYNTHESIZED_WIRE_16),
    .data17x(SYNTHESIZED_WIRE_17),
    .data18x(SYNTHESIZED_WIRE_18),
    .data19x(SYNTHESIZED_WIRE_19),
    .data1x(SYNTHESIZED_WIRE_20),
    .data20x(SYNTHESIZED_WIRE_21),
    .data21x(SYNTHESIZED_WIRE_22),
    .data22x(SYNTHESIZED_WIRE_23),
    .data23x(SYNTHESIZED_WIRE_24),
```

```
.data24x(SYNTHESIZED_WIRE_25),
.data25x(SYNTHESIZED_WIRE_26),
.data26x(SYNTHESIZED_WIRE_27),
.data27x(SYNTHESIZED_WIRE_28),
.data28x(SYNTHESIZED_WIRE_29),
.data29x(SYNTHESIZED_WIRE_30),
.data2x(SYNTHESIZED_WIRE_31),
.data30x(SYNTHESIZED_WIRE_32),
.data31x(SYNTHESIZED_WIRE_33),
.data32x(SYNTHESIZED_WIRE_34),
.data33x(SYNTHESIZED_WIRE_35),
.data34x(SYNTHESIZED_WIRE_36),
.data35x(SYNTHESIZED_WIRE_37),
.data36x(SYNTHESIZED_WIRE_38),
.data37x(SYNTHESIZED_WIRE_39),
.data38x(SYNTHESIZED_WIRE_40),
.data39x(SYNTHESIZED_WIRE_41),
.data3x(SYNTHESIZED_WIRE_42),
.data4x(SYNTHESIZED_WIRE_43),
.data5x(SYNTHESIZED_WIRE_44),
.data6x(SYNTHESIZED_WIRE_45),
.data7x(SYNTHESIZED_WIRE_46),
.data8x(SYNTHESIZED_WIRE_47),
```

```
    .data9x(SYNTHESIZED_WIRE_48),
    .result(SYNTHESIZED_WIRE_52));


inv_cell                                                    b2v_inst18(
    .data(SYNTHESIZED_WIRE_49),
    .result(data_in_delay));


delay_cell_2                                                b2v_inst19(
    .data_in(data_in_sttop),
    .clk(clk),
    .delay_out(delay_out_stop),
    .Q0(SYNTHESIZED_WIRE_41),
    .Q1(SYNTHESIZED_WIRE_40),
    .Q2(SYNTHESIZED_WIRE_39),
    .Q3(SYNTHESIZED_WIRE_38),
    .Q4(SYNTHESIZED_WIRE_37),
    .Q5(SYNTHESIZED_WIRE_36),
    .Q6(SYNTHESIZED_WIRE_35),
    .Q32(SYNTHESIZED_WIRE_34),
    .Q33(SYNTHESIZED_WIRE_33),
    .Q34(SYNTHESIZED_WIRE_32),
```

```
.Q35(SYNTHESIZED_WIRE_30),
.Q36(SYNTHESIZED_WIRE_29),
.Q37(SYNTHESIZED_WIRE_28),
.Q38(SYNTHESIZED_WIRE_27),
.Q39(SYNTHESIZED_WIRE_26),
.Q16(SYNTHESIZED_WIRE_25),
.Q17(SYNTHESIZED_WIRE_24),
.Q18(SYNTHESIZED_WIRE_23),
.Q19(SYNTHESIZED_WIRE_22),
.Q20(SYNTHESIZED_WIRE_21),
.Q21(SYNTHESIZED_WIRE_19),
.Q22(SYNTHESIZED_WIRE_18),
.Q23(SYNTHESIZED_WIRE_17),
.Q24(SYNTHESIZED_WIRE_16),
.Q25(SYNTHESIZED_WIRE_15),
.Q26(SYNTHESIZED_WIRE_14),
.Q27(SYNTHESIZED_WIRE_13),
.Q28(SYNTHESIZED_WIRE_12),
.Q29(SYNTHESIZED_WIRE_11),
.Q30(SYNTHESIZED_WIRE_10),
.Q31(SYNTHESIZED_WIRE_48),
.Q7(SYNTHESIZED_WIRE_47),
.Q8(SYNTHESIZED_WIRE_46),
```

```
    .Q9(SYNTHESIZED_WIRE_45),
    .Q10(SYNTHESIZED_WIRE_44),
    .Q11(SYNTHESIZED_WIRE_43),
    .Q12(SYNTHESIZED_WIRE_42),
    .Q13(SYNTHESIZED_WIRE_31),
    .Q14(SYNTHESIZED_WIRE_20),
    .Q15(SYNTHESIZED_WIRE_9));


code_cnt                                                    b2v_inst4(
    .clk(SYNTHESIZED_WIRE_183),
    .rst(rst),
    .cnt_en(cnt_en),
    .code_cnt_en(SYNTHESIZED_WIRE_51),
    .data_in(SYNTHESIZED_WIRE_52),
    .data_in_coarse(SYNTHESIZED_WIRE_53),
    .data_out_en(SYNTHESIZED_WIRE_6),
    .cnt_100_mem(SYNTHESIZED_WIRE_96),
    .cnt_101_mem(SYNTHESIZED_WIRE_95),
    .cnt_102_mem(SYNTHESIZED_WIRE_94),
    .cnt_103_mem(SYNTHESIZED_WIRE_93),
    .cnt_104_mem(SYNTHESIZED_WIRE_92),
    .cnt_105_mem(SYNTHESIZED_WIRE_91),
```

```
.cnt_106_mem(SYNTHESIZED_WIRE_89),
.cnt_107_mem(SYNTHESIZED_WIRE_88),
.cnt_108_mem(SYNTHESIZED_WIRE_87),
.cnt_109_mem(SYNTHESIZED_WIRE_86),
.cnt_10_mem(SYNTHESIZED_WIRE_71),
.cnt_110_mem(SYNTHESIZED_WIRE_85),
.cnt_111_mem(SYNTHESIZED_WIRE_84),
.cnt_112_mem(SYNTHESIZED_WIRE_83),
.cnt_113_mem(SYNTHESIZED_WIRE_82),
.cnt_114_mem(SYNTHESIZED_WIRE_76),
.cnt_115_mem(SYNTHESIZED_WIRE_65),
.cnt_116_mem(SYNTHESIZED_WIRE_178),
.cnt_117_mem(SYNTHESIZED_WIRE_167),
.cnt_118_mem(SYNTHESIZED_WIRE_156),
.cnt_119_mem(SYNTHESIZED_WIRE_145),
.cnt_11_mem(SYNTHESIZED_WIRE_70),
.cnt_120_mem(SYNTHESIZED_WIRE_134),
.cnt_121_mem(SYNTHESIZED_WIRE_123),
.cnt_122_mem(SYNTHESIZED_WIRE_112),
.cnt_123_mem(SYNTHESIZED_WIRE_101),
.cnt_124_mem(SYNTHESIZED_WIRE_90),
.cnt_125_mem(SYNTHESIZED_WIRE_54),
.cnt_12_mem(SYNTHESIZED_WIRE_69),
```

.cnt_13_mem(SYNTHESIZED_WIRE_68),

.cnt_14_mem(SYNTHESIZED_WIRE_67),

.cnt_15_mem(SYNTHESIZED_WIRE_66),

.cnt_16_mem(SYNTHESIZED_WIRE_64),

.cnt_17_mem(SYNTHESIZED_WIRE_63),

.cnt_18_mem(SYNTHESIZED_WIRE_62),

.cnt_19_mem(SYNTHESIZED_WIRE_61),

.cnt_1_mem(SYNTHESIZED_WIRE_81),

.cnt_20_mem(SYNTHESIZED_WIRE_60),

.cnt_21_mem(SYNTHESIZED_WIRE_59),

.cnt_22_mem(SYNTHESIZED_WIRE_58),

.cnt_23_mem(SYNTHESIZED_WIRE_57),

.cnt_24_mem(SYNTHESIZED_WIRE_56),

.cnt_25_mem(SYNTHESIZED_WIRE_55),

.cnt_26_mem(SYNTHESIZED_WIRE_177),

.cnt_27_mem(SYNTHESIZED_WIRE_176),

.cnt_28_mem(SYNTHESIZED_WIRE_175),

.cnt_29_mem(SYNTHESIZED_WIRE_174),

.cnt_2_mem(SYNTHESIZED_WIRE_80),

.cnt_30_mem(SYNTHESIZED_WIRE_173),

.cnt_31_mem(SYNTHESIZED_WIRE_172),

.cnt_32_mem(SYNTHESIZED_WIRE_171),

.cnt_33_mem(SYNTHESIZED_WIRE_170),

.cnt_34_mem(SYNTHESIZED_WIRE_169),

.cnt_35_mem(SYNTHESIZED_WIRE_168),

.cnt_36_mem(SYNTHESIZED_WIRE_166),

.cnt_37_mem(SYNTHESIZED_WIRE_165),

.cnt_38_mem(SYNTHESIZED_WIRE_164),

.cnt_39_mem(SYNTHESIZED_WIRE_163),

.cnt_3_mem(SYNTHESIZED_WIRE_79),

.cnt_40_mem(SYNTHESIZED_WIRE_162),

.cnt_41_mem(SYNTHESIZED_WIRE_161),

.cnt_42_mem(SYNTHESIZED_WIRE_160),

.cnt_43_mem(SYNTHESIZED_WIRE_159),

.cnt_44_mem(SYNTHESIZED_WIRE_158),

.cnt_45_mem(SYNTHESIZED_WIRE_157),

.cnt_46_mem(SYNTHESIZED_WIRE_155),

.cnt_47_mem(SYNTHESIZED_WIRE_154),

.cnt_48_mem(SYNTHESIZED_WIRE_153),

.cnt_49_mem(SYNTHESIZED_WIRE_152),

.cnt_4_mem(SYNTHESIZED_WIRE_78),

.cnt_50_mem(SYNTHESIZED_WIRE_151),

.cnt_51_mem(SYNTHESIZED_WIRE_150),

.cnt_52_mem(SYNTHESIZED_WIRE_149),

.cnt_53_mem(SYNTHESIZED_WIRE_148),

.cnt_54_mem(SYNTHESIZED_WIRE_147),

.cnt_55_mem(SYNTHESIZED_WIRE_146),

.cnt_56_mem(SYNTHESIZED_WIRE_144),

.cnt_57_mem(SYNTHESIZED_WIRE_143),

.cnt_58_mem(SYNTHESIZED_WIRE_142),

.cnt_59_mem(SYNTHESIZED_WIRE_141),

.cnt_5_mem(SYNTHESIZED_WIRE_77),

.cnt_60_mem(SYNTHESIZED_WIRE_140),

.cnt_61_mem(SYNTHESIZED_WIRE_139),

.cnt_62_mem(SYNTHESIZED_WIRE_138),

.cnt_63_mem(SYNTHESIZED_WIRE_137),

.cnt_64_mem(SYNTHESIZED_WIRE_136),

.cnt_65_mem(SYNTHESIZED_WIRE_135),

.cnt_66_mem(SYNTHESIZED_WIRE_133),

.cnt_67_mem(SYNTHESIZED_WIRE_132),

.cnt_68_mem(SYNTHESIZED_WIRE_131),

.cnt_69_mem(SYNTHESIZED_WIRE_130),

.cnt_6_mem(SYNTHESIZED_WIRE_75),

.cnt_70_mem(SYNTHESIZED_WIRE_129),

.cnt_71_mem(SYNTHESIZED_WIRE_128),

.cnt_72_mem(SYNTHESIZED_WIRE_127),

.cnt_73_mem(SYNTHESIZED_WIRE_126),

.cnt_74_mem(SYNTHESIZED_WIRE_125),

.cnt_75_mem(SYNTHESIZED_WIRE_124),

.cnt_76_mem(SYNTHESIZED_WIRE_122),

.cnt_77_mem(SYNTHESIZED_WIRE_121),

.cnt_78_mem(SYNTHESIZED_WIRE_120),

.cnt_79_mem(SYNTHESIZED_WIRE_119),

.cnt_7_mem(SYNTHESIZED_WIRE_74),

.cnt_80_mem(SYNTHESIZED_WIRE_118),

.cnt_81_mem(SYNTHESIZED_WIRE_117),

.cnt_82_mem(SYNTHESIZED_WIRE_116),

.cnt_83_mem(SYNTHESIZED_WIRE_115),

.cnt_84_mem(SYNTHESIZED_WIRE_114),

.cnt_85_mem(SYNTHESIZED_WIRE_113),

.cnt_86_mem(SYNTHESIZED_WIRE_111),

.cnt_87_mem(SYNTHESIZED_WIRE_110),

.cnt_88_mem(SYNTHESIZED_WIRE_109),

.cnt_89_mem(SYNTHESIZED_WIRE_108),

.cnt_8_mem(SYNTHESIZED_WIRE_73),

.cnt_90_mem(SYNTHESIZED_WIRE_107),

.cnt_91_mem(SYNTHESIZED_WIRE_106),

.cnt_92_mem(SYNTHESIZED_WIRE_105),

.cnt_93_mem(SYNTHESIZED_WIRE_104),

.cnt_94_mem(SYNTHESIZED_WIRE_103),

.cnt_95_mem(SYNTHESIZED_WIRE_102),

.cnt_96_mem(SYNTHESIZED_WIRE_100),

```
    .cnt_97_mem(SYNTHESIZED_WIRE_99),
    .cnt_98_mem(SYNTHESIZED_WIRE_98),
    .cnt_99_mem(SYNTHESIZED_WIRE_97),
    .cnt_9_mem(SYNTHESIZED_WIRE_72));


mux_cell                                                    b2v_inst5(
    .data0x(SYNTHESIZED_WIRE_54),
    .data100x(SYNTHESIZED_WIRE_55),
    .data101x(SYNTHESIZED_WIRE_56),
    .data102x(SYNTHESIZED_WIRE_57),
    .data103x(SYNTHESIZED_WIRE_58),
    .data104x(SYNTHESIZED_WIRE_59),
    .data105x(SYNTHESIZED_WIRE_60),
    .data106x(SYNTHESIZED_WIRE_61),
    .data107x(SYNTHESIZED_WIRE_62),
    .data108x(SYNTHESIZED_WIRE_63),
    .data109x(SYNTHESIZED_WIRE_64),
    .data10x(SYNTHESIZED_WIRE_65),
    .data110x(SYNTHESIZED_WIRE_66),
    .data111x(SYNTHESIZED_WIRE_67),
    .data112x(SYNTHESIZED_WIRE_68),
    .data113x(SYNTHESIZED_WIRE_69),
```

.data114x(SYNTHESIZED_WIRE_70),

.data115x(SYNTHESIZED_WIRE_71),

.data116x(SYNTHESIZED_WIRE_72),

.data117x(SYNTHESIZED_WIRE_73),

.data118x(SYNTHESIZED_WIRE_74),

.data119x(SYNTHESIZED_WIRE_75),

.data11x(SYNTHESIZED_WIRE_76),

.data120x(SYNTHESIZED_WIRE_77),

.data121x(SYNTHESIZED_WIRE_78),

.data122x(SYNTHESIZED_WIRE_79),

.data123x(SYNTHESIZED_WIRE_80),

.data124x(SYNTHESIZED_WIRE_81),

.data12x(SYNTHESIZED_WIRE_82),

.data13x(SYNTHESIZED_WIRE_83),

.data14x(SYNTHESIZED_WIRE_84),

.data15x(SYNTHESIZED_WIRE_85),

.data16x(SYNTHESIZED_WIRE_86),

.data17x(SYNTHESIZED_WIRE_87),

.data18x(SYNTHESIZED_WIRE_88),

.data19x(SYNTHESIZED_WIRE_89),

.data1x(SYNTHESIZED_WIRE_90),

.data20x(SYNTHESIZED_WIRE_91),

.data21x(SYNTHESIZED_WIRE_92),

.data22x(SYNTHESIZED_WIRE_93),

.data23x(SYNTHESIZED_WIRE_94),

.data24x(SYNTHESIZED_WIRE_95),

.data25x(SYNTHESIZED_WIRE_96),

.data26x(SYNTHESIZED_WIRE_97),

.data27x(SYNTHESIZED_WIRE_98),

.data28x(SYNTHESIZED_WIRE_99),

.data29x(SYNTHESIZED_WIRE_100),

.data2x(SYNTHESIZED_WIRE_101),

.data30x(SYNTHESIZED_WIRE_102),

.data31x(SYNTHESIZED_WIRE_103),

.data32x(SYNTHESIZED_WIRE_104),

.data33x(SYNTHESIZED_WIRE_105),

.data34x(SYNTHESIZED_WIRE_106),

.data35x(SYNTHESIZED_WIRE_107),

.data36x(SYNTHESIZED_WIRE_108),

.data37x(SYNTHESIZED_WIRE_109),

.data38x(SYNTHESIZED_WIRE_110),

.data39x(SYNTHESIZED_WIRE_111),

.data3x(SYNTHESIZED_WIRE_112),

.data40x(SYNTHESIZED_WIRE_113),

.data41x(SYNTHESIZED_WIRE_114),

.data42x(SYNTHESIZED_WIRE_115),

.data43x(SYNTHESIZED_WIRE_116),

.data44x(SYNTHESIZED_WIRE_117),

.data45x(SYNTHESIZED_WIRE_118),

.data46x(SYNTHESIZED_WIRE_119),

.data47x(SYNTHESIZED_WIRE_120),

.data48x(SYNTHESIZED_WIRE_121),

.data49x(SYNTHESIZED_WIRE_122),

.data4x(SYNTHESIZED_WIRE_123),

.data50x(SYNTHESIZED_WIRE_124),

.data51x(SYNTHESIZED_WIRE_125),

.data52x(SYNTHESIZED_WIRE_126),

.data53x(SYNTHESIZED_WIRE_127),

.data54x(SYNTHESIZED_WIRE_128),

.data55x(SYNTHESIZED_WIRE_129),

.data56x(SYNTHESIZED_WIRE_130),

.data57x(SYNTHESIZED_WIRE_131),

.data58x(SYNTHESIZED_WIRE_132),

.data59x(SYNTHESIZED_WIRE_133),

.data5x(SYNTHESIZED_WIRE_134),

.data60x(SYNTHESIZED_WIRE_135),

.data61x(SYNTHESIZED_WIRE_136),

.data62x(SYNTHESIZED_WIRE_137),

.data63x(SYNTHESIZED_WIRE_138),

```
.data64x(SYNTHESIZED_WIRE_139),
.data65x(SYNTHESIZED_WIRE_140),
.data66x(SYNTHESIZED_WIRE_141),
.data67x(SYNTHESIZED_WIRE_142),
.data68x(SYNTHESIZED_WIRE_143),
.data69x(SYNTHESIZED_WIRE_144),
.data6x(SYNTHESIZED_WIRE_145),
.data70x(SYNTHESIZED_WIRE_146),
.data71x(SYNTHESIZED_WIRE_147),
.data72x(SYNTHESIZED_WIRE_148),
.data73x(SYNTHESIZED_WIRE_149),
.data74x(SYNTHESIZED_WIRE_150),
.data75x(SYNTHESIZED_WIRE_151),
.data76x(SYNTHESIZED_WIRE_152),
.data77x(SYNTHESIZED_WIRE_153),
.data78x(SYNTHESIZED_WIRE_154),
.data79x(SYNTHESIZED_WIRE_155),
.data7x(SYNTHESIZED_WIRE_156),
.data80x(SYNTHESIZED_WIRE_157),
.data81x(SYNTHESIZED_WIRE_158),
.data82x(SYNTHESIZED_WIRE_159),
.data83x(SYNTHESIZED_WIRE_160),
.data84x(SYNTHESIZED_WIRE_161),
```

```
    .data85x(SYNTHESIZED_WIRE_162),
    .data86x(SYNTHESIZED_WIRE_163),
    .data87x(SYNTHESIZED_WIRE_164),
    .data88x(SYNTHESIZED_WIRE_165),
    .data89x(SYNTHESIZED_WIRE_166),
    .data8x(SYNTHESIZED_WIRE_167),
    .data90x(SYNTHESIZED_WIRE_168),
    .data91x(SYNTHESIZED_WIRE_169),
    .data92x(SYNTHESIZED_WIRE_170),
    .data93x(SYNTHESIZED_WIRE_171),
    .data94x(SYNTHESIZED_WIRE_172),
    .data95x(SYNTHESIZED_WIRE_173),
    .data96x(SYNTHESIZED_WIRE_174),
    .data97x(SYNTHESIZED_WIRE_175),
    .data98x(SYNTHESIZED_WIRE_176),
    .data99x(SYNTHESIZED_WIRE_177),
    .data9x(SYNTHESIZED_WIRE_178),
    .sel(SYNTHESIZED_WIRE_179),
    .result(SYNTHESIZED_WIRE_5));


clkdiv                                                    b2v_inst8(
    .clk(clk),
```

```
    .rst(rst),
    .clkout(SYNTHESIZED_WIRE_184));


uarttx                                                    b2v_inst9(
    .clk(SYNTHESIZED_WIRE_184),
    .reset(rst),
    .wrsig(SYNTHESIZED_WIRE_181),
    .datain(SYNTHESIZED_WIRE_182),
    .idle(SYNTHESIZED_WIRE_2),
    .tx(uart_tx));
    defparam                                    b2v_inst9.paritymode = 1'b0;

    endmodule
```

1.2 Coarse counter

```
module counter_start(
clk,
rst,
en_cnt,
start,
stop,
cnt_out,
```

```
code_en


);
input clk, rst,  en_cnt;
input start, stop;


output [7:0] cnt_out;
reg [7:0] cnt_out;
output code_en;
reg code_en;
reg [1:0] cnt;
/// FSM


reg [3:0] state;


parameter idle                                          = 4'd0;
parameter wait_signal = 4'd1;
parameter start_rise                                    = 4'd2;
parameter stop_rise                                     = 4'd3;
//
// detect the rise start
reg q1_pp,q2_pp;
wire en_apd_rise_pp;
```

```
always@(posedge clk)
q1_pp<=start;

always@(posedge clk)
q2_pp<=q1_pp;
assign   q3_pp=!q2_pp;
assign   en_apd_rise_pp=q1_pp&&q3_pp;
// detect the rise stop
reg q1_p,q2_p;
wire en_apd_rise_p;

always@(posedge clk)
q1_p<=stop;

always@(posedge clk)
q2_p<=q1_p;
assign   q3_p=!q2_p;
assign   en_apd_rise_p=q1_p&&q3_p;

always@(posedge clk or negedge rst)
begin
if(!rst)
```

```
begin
cnt_out<=0;
code_en<=0;
cnt<=0;
state <= idle;
end
else
begin
case(state)
idle://0  //
begin
cnt_out <=0;
cnt<=0;
code_en<=0;
if(en_cnt)
state <= wait_signal;
else
state <= idle;
end

wait_signal://1 //
begin
```

```
if(en_apd_rise_pp && en_cnt)
begin
state <= start_rise;
cnt_out <= cnt_out+1;
code_en<=0;
end
else
begin
state <= wait_signal;
cnt_out <= cnt_out;
code_en<=0;
end
end

start_rise: //2 //
begin

if(en_apd_rise_p)
begin
cnt_out <= cnt_out;
state <= stop_rise;
code_en<=1;
```

```
end
else if(en_apd_rise_pp)
begin

state<=start_rise;
cnt_out <= 0;
code_en<=0;

end

else
begin
cnt_out <= cnt_out+1;
state <= start_rise;
code_en<=0;
end
end
stop_rise: //3 //
begin

if(cnt==1)
begin
cnt<=0;
```

```
state<=wait_signal;
cnt_out <= 0;
code_en<=0;
end
else
begin
cnt<=cnt+1;
cnt_out <= cnt_out;
code_en<=1;
state <= stop_rise;
end
end
default:
begin
cnt_out <= 0;
cnt<=0;
code_en<=0;
state <= idle;
end
endcase
end
end
endmodule
```

1.3 Encoder

```
module code_cnt (
clk,
rst,
data_in,
data_in_coarse,
cnt_en,
code_cnt_en,
////////////////////    output fot the 232 output 125
cnt_1_mem,
cnt_2_mem,
cnt_3_mem,
cnt_4_mem,
cnt_5_mem,
cnt_6_mem,
cnt_7_mem,
cnt_8_mem,
cnt_9_mem,
cnt_10_mem,
cnt_11_mem,
cnt_12_mem,
cnt_13_mem,
cnt_14_mem,
```

cnt_15_mem,

cnt_16_mem,

cnt_17_mem,

cnt_18_mem,

cnt_19_mem,

cnt_20_mem,

cnt_21_mem,

cnt_22_mem,

cnt_23_mem,

cnt_24_mem,

cnt_25_mem,

cnt_26_mem,

cnt_27_mem,

cnt_28_mem,

cnt_29_mem,

cnt_30_mem,

cnt_31_mem,

cnt_32_mem,

cnt_33_mem,

cnt_34_mem,

cnt_35_mem,

cnt_36_mem,

cnt_37_mem,

cnt_38_mem,

cnt_39_mem,

cnt_40_mem,

cnt_41_mem,

cnt_42_mem,

cnt_43_mem,

cnt_44_mem,

cnt_45_mem,

cnt_46_mem,

cnt_47_mem,

cnt_48_mem,

cnt_49_mem,

cnt_50_mem,

cnt_51_mem,

cnt_52_mem,

cnt_53_mem,

cnt_54_mem,

cnt_55_mem,

cnt_56_mem,

cnt_57_mem,

cnt_58_mem,

cnt_59_mem,

cnt_60_mem,

cnt_61_mem,

cnt_62_mem,

cnt_63_mem,

cnt_64_mem,

cnt_65_mem,

cnt_66_mem,

cnt_67_mem,

cnt_68_mem,

cnt_69_mem,

cnt_70_mem,

cnt_71_mem,

cnt_72_mem,

cnt_73_mem,

cnt_74_mem,

cnt_75_mem,

cnt_76_mem,

cnt_77_mem,

cnt_78_mem,

cnt_79_mem,

cnt_80_mem,

cnt_81_mem,

cnt_82_mem,

cnt_83_mem,

cnt_84_mem,

cnt_85_mem,

cnt_86_mem,

cnt_87_mem,

cnt_88_mem,

cnt_89_mem,

cnt_90_mem,

cnt_91_mem,

cnt_92_mem,

cnt_93_mem,

cnt_94_mem,

cnt_95_mem,

cnt_96_mem,

cnt_97_mem,

cnt_98_mem,

cnt_99_mem,

cnt_100_mem,

cnt_101_mem,

cnt_102_mem,

cnt_103_mem,

cnt_104_mem,

cnt_105_mem,

cnt_106_mem,

```
cnt_107_mem,
cnt_108_mem,
cnt_109_mem,
cnt_110_mem,
cnt_111_mem,
cnt_112_mem,
cnt_113_mem,
cnt_114_mem,
cnt_115_mem,
cnt_116_mem,
cnt_117_mem,
cnt_118_mem,
cnt_119_mem,
cnt_120_mem,
cnt_121_mem,
cnt_122_mem,
cnt_123_mem,
cnt_124_mem,
cnt_125_mem,
data_out_en
);

input clk, rst, code_cnt_en, cnt_en;
```

```
input   [6:0] data_in;
input   [7:0] data_in_coarse;

//output                                                [11:0] data_out;
output data_out_en;

//reg                                                   [11:0] data_out;
reg     data_out_en;

reg     [39:0] cnt;

//// reg gor the 125 cnt///


reg [11:0] cnt_1;
reg [11:0] cnt_2;
reg [11:0] cnt_3;
reg [11:0] cnt_4;
reg [11:0] cnt_5;
reg [11:0] cnt_6;
reg [11:0] cnt_7;
reg [11:0] cnt_8;
reg [11:0] cnt_9;
```

```
reg [11:0] cnt_10;
reg [11:0] cnt_11;
reg [11:0] cnt_12;
reg [11:0] cnt_13;
reg [11:0] cnt_14;
reg [11:0] cnt_15;
reg [11:0] cnt_16;
reg [11:0] cnt_17;
reg [11:0] cnt_18;
reg [11:0] cnt_19;
reg [11:0] cnt_20;
reg [11:0] cnt_21;
reg [11:0] cnt_22;
reg [11:0] cnt_23;
reg [11:0] cnt_24;
reg [11:0] cnt_25;
reg [11:0] cnt_26;
reg [11:0] cnt_27;
reg [11:0] cnt_28;
reg [11:0] cnt_29;
reg [11:0] cnt_30;
reg [11:0] cnt_31;
reg [11:0] cnt_32;
```

```
reg [11:0] cnt_33;

reg [11:0] cnt_34;

reg [11:0] cnt_35;

reg [11:0] cnt_36;

reg [11:0] cnt_37;

reg [11:0] cnt_38;

reg [11:0] cnt_39;

reg [11:0] cnt_40;

reg [11:0] cnt_41;

reg [11:0] cnt_42;

reg [11:0] cnt_43;

reg [11:0] cnt_44;

reg [11:0] cnt_45;

reg [11:0] cnt_46;

reg [11:0] cnt_47;

reg [11:0] cnt_48;

reg [11:0] cnt_49;

reg [11:0] cnt_50;

reg [11:0] cnt_51;

reg [11:0] cnt_52;

reg [11:0] cnt_53;

reg [11:0] cnt_54;

reg [11:0] cnt_55;
```

```
reg [11:0] cnt_56;
reg [11:0] cnt_57;
reg [11:0] cnt_58;
reg [11:0] cnt_59;
reg [11:0] cnt_60;
reg [11:0] cnt_61;
reg [11:0] cnt_62;
reg [11:0] cnt_63;
reg [11:0] cnt_64;
reg [11:0] cnt_65;
reg [11:0] cnt_66;
reg [11:0] cnt_67;
reg [11:0] cnt_68;
reg [11:0] cnt_69;
reg [11:0] cnt_70;
reg [11:0] cnt_71;
reg [11:0] cnt_72;
reg [11:0] cnt_73;
reg [11:0] cnt_74;
reg [11:0] cnt_75;
reg [11:0] cnt_76;
reg [11:0] cnt_77;
reg [11:0] cnt_78;
```

```
reg [11:0] cnt_79;

reg [11:0] cnt_80;

reg [11:0] cnt_81;

reg [11:0] cnt_82;

reg [11:0] cnt_83;

reg [11:0] cnt_84;

reg [11:0] cnt_85;

reg [11:0] cnt_86;

reg [11:0] cnt_87;

reg [11:0] cnt_88;

reg [11:0] cnt_89;

reg [11:0] cnt_90;

reg [11:0] cnt_91;

reg [11:0] cnt_92;

reg [11:0] cnt_93;

reg [11:0] cnt_94;

reg [11:0] cnt_95;

reg [11:0] cnt_96;

reg [11:0] cnt_97;

reg [11:0] cnt_98;

reg [11:0] cnt_99;

reg [11:0] cnt_100;

reg [11:0] cnt_101;
```

```
reg [11:0] cnt_102;

reg [11:0] cnt_103;

reg [11:0] cnt_104;

reg [11:0] cnt_105;

reg [11:0] cnt_106;

reg [11:0] cnt_107;

reg [11:0] cnt_108;

reg [11:0] cnt_109;

reg [11:0] cnt_110;

reg [11:0] cnt_111;

reg [11:0] cnt_112;

reg [11:0] cnt_113;

reg [11:0] cnt_114;

reg [11:0] cnt_115;

reg [11:0] cnt_116;

reg [11:0] cnt_117;

reg [11:0] cnt_118;

reg [11:0] cnt_119;

reg [11:0] cnt_120;

reg [11:0] cnt_121;

reg [11:0] cnt_122;

reg [11:0] cnt_123;

reg [11:0] cnt_124;
```

```
reg [11:0] cnt_125;

////

//// reg gor the 125 cnt to out put///

output reg [11:0] cnt_1_mem;
output reg [11:0] cnt_2_mem;
output reg [11:0] cnt_3_mem;
output reg [11:0] cnt_4_mem;
output reg [11:0] cnt_5_mem;
output reg [11:0] cnt_6_mem;
output reg [11:0] cnt_7_mem;
output reg [11:0] cnt_8_mem;
output reg [11:0] cnt_9_mem;
output reg [11:0] cnt_10_mem;
output reg [11:0] cnt_11_mem;
output reg [11:0] cnt_12_mem;
output reg [11:0] cnt_13_mem;
output reg [11:0] cnt_14_mem;
output reg [11:0] cnt_15_mem;
output reg [11:0] cnt_16_mem;
output reg [11:0] cnt_17_mem;
```

```
output reg [11:0] cnt_18_mem;
output reg [11:0] cnt_19_mem;
output reg [11:0] cnt_20_mem;
output reg [11:0] cnt_21_mem;
output reg [11:0] cnt_22_mem;
output reg [11:0] cnt_23_mem;
output reg [11:0] cnt_24_mem;
output reg [11:0] cnt_25_mem;
output reg [11:0] cnt_26_mem;
output reg [11:0] cnt_27_mem;
output reg [11:0] cnt_28_mem;
output reg [11:0] cnt_29_mem;
output reg [11:0] cnt_30_mem;
output reg [11:0] cnt_31_mem;
output reg [11:0] cnt_32_mem;
output reg [11:0] cnt_33_mem;
output reg [11:0] cnt_34_mem;
output reg [11:0] cnt_35_mem;
output reg [11:0] cnt_36_mem;
output reg [11:0] cnt_37_mem;
output reg [11:0] cnt_38_mem;
output reg [11:0] cnt_39_mem;
output reg [11:0] cnt_40_mem;
```

```
output reg [11:0] cnt_41_mem;
output reg [11:0] cnt_42_mem;
output reg [11:0] cnt_43_mem;
output reg [11:0] cnt_44_mem;
output reg [11:0] cnt_45_mem;
output reg [11:0] cnt_46_mem;
output reg [11:0] cnt_47_mem;
output reg [11:0] cnt_48_mem;
output reg [11:0] cnt_49_mem;
output reg [11:0] cnt_50_mem;
output reg [11:0] cnt_51_mem;
output reg [11:0] cnt_52_mem;
output reg [11:0] cnt_53_mem;
output reg [11:0] cnt_54_mem;
output reg [11:0] cnt_55_mem;
output reg [11:0] cnt_56_mem;
output reg [11:0] cnt_57_mem;
output reg [11:0] cnt_58_mem;
output reg [11:0] cnt_59_mem;
output reg [11:0] cnt_60_mem;
output reg [11:0] cnt_61_mem;
output reg [11:0] cnt_62_mem;
output reg [11:0] cnt_63_mem;
```

```
output reg [11:0] cnt_64_mem;
output reg [11:0] cnt_65_mem;
output reg [11:0] cnt_66_mem;
output reg [11:0] cnt_67_mem;
output reg [11:0] cnt_68_mem;
output reg [11:0] cnt_69_mem;
output reg [11:0] cnt_70_mem;
output reg [11:0] cnt_71_mem;
output reg [11:0] cnt_72_mem;
output reg [11:0] cnt_73_mem;
output reg [11:0] cnt_74_mem;
output reg [11:0] cnt_75_mem;
output reg [11:0] cnt_76_mem;
output reg [11:0] cnt_77_mem;
output reg [11:0] cnt_78_mem;
output reg [11:0] cnt_79_mem;
output reg [11:0] cnt_80_mem;
output reg [11:0] cnt_81_mem;
output reg [11:0] cnt_82_mem;
output reg [11:0] cnt_83_mem;
output reg [11:0] cnt_84_mem;
output reg [11:0] cnt_85_mem;
output reg [11:0] cnt_86_mem;
```

```
output reg [11:0] cnt_87_mem;
output reg [11:0] cnt_88_mem;
output reg [11:0] cnt_89_mem;
output reg [11:0] cnt_90_mem;
output reg [11:0] cnt_91_mem;
output reg [11:0] cnt_92_mem;
output reg [11:0] cnt_93_mem;
output reg [11:0] cnt_94_mem;
output reg [11:0] cnt_95_mem;
output reg [11:0] cnt_96_mem;
output reg [11:0] cnt_97_mem;
output reg [11:0] cnt_98_mem;
output reg [11:0] cnt_99_mem;
output reg [11:0] cnt_100_mem;
output reg [11:0] cnt_101_mem;
output reg [11:0] cnt_102_mem;
output reg [11:0] cnt_103_mem;
output reg [11:0] cnt_104_mem;
output reg [11:0] cnt_105_mem;
output reg [11:0] cnt_106_mem;
output reg [11:0] cnt_107_mem;
output reg [11:0] cnt_108_mem;
output reg [11:0] cnt_109_mem;
```

```
output reg [11:0] cnt_110_mem;

output reg [11:0] cnt_111_mem;

output reg [11:0] cnt_112_mem;

output reg [11:0] cnt_113_mem;

output reg [11:0] cnt_114_mem;

output reg [11:0] cnt_115_mem;

output reg [11:0] cnt_116_mem;

output reg [11:0] cnt_117_mem;

output reg [11:0] cnt_118_mem;

output reg [11:0] cnt_119_mem;

output reg [11:0] cnt_120_mem;

output reg [11:0] cnt_121_mem;

output reg [11:0] cnt_122_mem;

output reg [11:0] cnt_123_mem;

output reg [11:0] cnt_124_mem;

output reg [11:0] cnt_125_mem;


/// calulation

reg [9:0] number_counter;


//assign number_counter=data_in_coarse*25-data_in;


// the rise fignal for the cnt_en
```

```
reg q1,q2;
wire en_apd_rise;

always@(posedge clk)
     q1<=code_cnt_en;

always@(posedge clk)
     q2<=q1;
   assign   q3=!q2;
   assign   en_apd_rise=q1&&q3;

//



/*    / FSM for the 40 cnt counter
reg [3:0] state;

parameter idle  = 4'd0;
parameter wait_signal    = 4'd1;
parameter received_rise  = 4'd2;
parameter cnt_next= 4'd3;

*/   ////////////
```

```
always@(posedge clk or negedge rst)
begin

if(!rst)
    begin
    cnt<=0;
    data_out_en<=0;
    number_counter <= 0;

    cnt_1_mem<=0;
    cnt_2_mem<=0;
    cnt_3_mem<=0;
    cnt_4_mem<=0;
    cnt_5_mem<=0;
    cnt_6_mem<=0;
    cnt_7_mem<=0;
    cnt_8_mem<=0;
    cnt_9_mem<=0;
    cnt_10_mem<=0;
    cnt_11_mem<=0;
    cnt_12_mem<=0;
```

```
cnt_13_mem<=0;
cnt_14_mem<=0;
cnt_15_mem<=0;
cnt_16_mem<=0;
cnt_17_mem<=0;
cnt_18_mem<=0;
cnt_19_mem<=0;
cnt_20_mem<=0;
cnt_21_mem<=0;
cnt_22_mem<=0;
cnt_23_mem<=0;
cnt_24_mem<=0;
cnt_25_mem<=0;
cnt_26_mem<=0;
cnt_27_mem<=0;
cnt_28_mem<=0;
cnt_29_mem<=0;
cnt_30_mem<=0;
cnt_31_mem<=0;
cnt_32_mem<=0;
cnt_33_mem<=0;
cnt_34_mem<=0;
cnt_35_mem<=0;
```

```
cnt_36_mem<=0;
cnt_37_mem<=0;
cnt_38_mem<=0;
cnt_39_mem<=0;
cnt_40_mem<=0;
cnt_41_mem<=0;
cnt_42_mem<=0;
cnt_43_mem<=0;
cnt_44_mem<=0;
cnt_45_mem<=0;
cnt_46_mem<=0;
cnt_47_mem<=0;
cnt_48_mem<=0;
cnt_49_mem<=0;
cnt_50_mem<=0;
cnt_51_mem<=0;
cnt_52_mem<=0;
cnt_53_mem<=0;
cnt_54_mem<=0;
cnt_55_mem<=0;
cnt_56_mem<=0;
cnt_57_mem<=0;
cnt_58_mem<=0;
```

```
cnt_59_mem<=0;
cnt_60_mem<=0;
cnt_61_mem<=0;
cnt_62_mem<=0;
cnt_63_mem<=0;
cnt_64_mem<=0;
cnt_65_mem<=0;
cnt_66_mem<=0;
cnt_67_mem<=0;
cnt_68_mem<=0;
cnt_69_mem<=0;
cnt_70_mem<=0;
cnt_71_mem<=0;
cnt_72_mem<=0;
cnt_73_mem<=0;
cnt_74_mem<=0;
cnt_75_mem<=0;
cnt_76_mem<=0;
cnt_77_mem<=0;
cnt_78_mem<=0;
cnt_79_mem<=0;
cnt_80_mem<=0;
cnt_81_mem<=0;
```

```
cnt_82_mem<=0;
cnt_83_mem<=0;
cnt_84_mem<=0;
cnt_85_mem<=0;
cnt_86_mem<=0;
cnt_87_mem<=0;
cnt_88_mem<=0;
cnt_89_mem<=0;
cnt_90_mem<=0;
cnt_91_mem<=0;
cnt_92_mem<=0;
cnt_93_mem<=0;
cnt_94_mem<=0;
cnt_95_mem<=0;
cnt_96_mem<=0;
cnt_97_mem<=0;
cnt_98_mem<=0;
cnt_99_mem<=0;
cnt_100_mem<=0;
cnt_101_mem<=0;
cnt_102_mem<=0;
cnt_103_mem<=0;
cnt_104_mem<=0;
```

```
cnt_105_mem<=0;
cnt_106_mem<=0;
cnt_107_mem<=0;
cnt_108_mem<=0;
cnt_109_mem<=0;
cnt_110_mem<=0;
cnt_111_mem<=0;
cnt_112_mem<=0;
cnt_113_mem<=0;
cnt_114_mem<=0;
cnt_115_mem<=0;
cnt_116_mem<=0;
cnt_117_mem<=0;
cnt_118_mem<=0;
cnt_119_mem<=0;
cnt_120_mem<=0;
cnt_121_mem<=0;
cnt_122_mem<=0;
cnt_123_mem<=0;
cnt_124_mem<=0;
cnt_125_mem<=0;

cnt_1<=0;
```

```
cnt_2<=0;
cnt_3<=0;
cnt_4<=0;
cnt_5<=0;
cnt_6<=0;
cnt_7<=0;
cnt_8<=0;
cnt_9<=0;
cnt_10<=0;
cnt_11<=0;
cnt_12<=0;
cnt_13<=0;
cnt_14<=0;
cnt_15<=0;
cnt_16<=0;
cnt_17<=0;
cnt_18<=0;
cnt_19<=0;
cnt_20<=0;
cnt_21<=0;
cnt_22<=0;
cnt_23<=0;
cnt_24<=0;
```

```
cnt_25<=0;
cnt_26<=0;
cnt_27<=0;
cnt_28<=0;
cnt_29<=0;
cnt_30<=0;
cnt_31<=0;
cnt_32<=0;
cnt_33<=0;
cnt_34<=0;
cnt_35<=0;
cnt_36<=0;
cnt_37<=0;
cnt_38<=0;
cnt_39<=0;
cnt_40<=0;
cnt_41<=0;
cnt_42<=0;
cnt_43<=0;
cnt_44<=0;
cnt_45<=0;
cnt_46<=0;
cnt_47<=0;
```

```
cnt_48<=0;
cnt_49<=0;
cnt_50<=0;
cnt_51<=0;
cnt_52<=0;
cnt_53<=0;
cnt_54<=0;
cnt_55<=0;
cnt_56<=0;
cnt_57<=0;
cnt_58<=0;
cnt_59<=0;
cnt_60<=0;
cnt_61<=0;
cnt_62<=0;
cnt_63<=0;
cnt_64<=0;
cnt_65<=0;
cnt_66<=0;
cnt_67<=0;
cnt_68<=0;
cnt_69<=0;
cnt_70<=0;
```

```
cnt_71<=0;
cnt_72<=0;
cnt_73<=0;
cnt_74<=0;
cnt_75<=0;
cnt_76<=0;
cnt_77<=0;
cnt_78<=0;
cnt_79<=0;
cnt_80<=0;
cnt_81<=0;
cnt_82<=0;
cnt_83<=0;
cnt_84<=0;
cnt_85<=0;
cnt_86<=0;
cnt_87<=0;
cnt_88<=0;
cnt_89<=0;
cnt_90<=0;
cnt_91<=0;
cnt_92<=0;
cnt_93<=0;
```

```
cnt_94<=0;
cnt_95<=0;
cnt_96<=0;
cnt_97<=0;
cnt_98<=0;
cnt_99<=0;
cnt_100<=0;
cnt_101<=0;
cnt_102<=0;
cnt_103<=0;
cnt_104<=0;
cnt_105<=0;
cnt_106<=0;
cnt_107<=0;
cnt_108<=0;
cnt_109<=0;
cnt_110<=0;
cnt_111<=0;
cnt_112<=0;
cnt_113<=0;
cnt_114<=0;
cnt_115<=0;
cnt_116<=0;
```

```
        cnt_117<=0;
        cnt_118<=0;
        cnt_119<=0;
        cnt_120<=0;
        cnt_121<=0;
        cnt_122<=0;
        cnt_123<=0;
        cnt_124<=0;
        cnt_125<=0;

        end
    else if (cnt==200000000)  /////   1s
        begin

        cnt<=0;
        data_out_en<=1;
        number_counter <= 0;

        cnt_1_mem<=cnt_1;
        cnt_2_mem<=cnt_2;
        cnt_3_mem<=cnt_3;
        cnt_4_mem<=cnt_4;
        cnt_5_mem<=cnt_5;
```

```
cnt_6_mem<=cnt_6;
cnt_7_mem<=cnt_7;
cnt_8_mem<=cnt_8;
cnt_9_mem<=cnt_9;
cnt_10_mem<=cnt_10;
cnt_11_mem<=cnt_11;
cnt_12_mem<=cnt_12;
cnt_13_mem<=cnt_13;
cnt_14_mem<=cnt_14;
cnt_15_mem<=cnt_15;
cnt_16_mem<=cnt_16;
cnt_17_mem<=cnt_17;
cnt_18_mem<=cnt_18;
cnt_19_mem<=cnt_19;
cnt_20_mem<=cnt_20;
cnt_21_mem<=cnt_21;
cnt_22_mem<=cnt_22;
cnt_23_mem<=cnt_23;
cnt_24_mem<=cnt_24;
cnt_25_mem<=cnt_25;
cnt_26_mem<=cnt_26;
cnt_27_mem<=cnt_27;
cnt_28_mem<=cnt_28;
```

```
cnt_29_mem<=cnt_29;
cnt_30_mem<=cnt_30;
cnt_31_mem<=cnt_31;
cnt_32_mem<=cnt_32;
cnt_33_mem<=cnt_33;
cnt_34_mem<=cnt_34;
cnt_35_mem<=cnt_35;
cnt_36_mem<=cnt_36;
cnt_37_mem<=cnt_37;
cnt_38_mem<=cnt_38;
cnt_39_mem<=cnt_39;
cnt_40_mem<=cnt_40;
cnt_41_mem<=cnt_41;
cnt_42_mem<=cnt_42;
cnt_43_mem<=cnt_43;
cnt_44_mem<=cnt_44;
cnt_45_mem<=cnt_45;
cnt_46_mem<=cnt_46;
cnt_47_mem<=cnt_47;
cnt_48_mem<=cnt_48;
cnt_49_mem<=cnt_49;
cnt_50_mem<=cnt_50;
cnt_51_mem<=cnt_51;
```

```
cnt_52_mem<=cnt_52;
cnt_53_mem<=cnt_53;
cnt_54_mem<=cnt_54;
cnt_55_mem<=cnt_55;
cnt_56_mem<=cnt_56;
cnt_57_mem<=cnt_57;
cnt_58_mem<=cnt_58;
cnt_59_mem<=cnt_59;
cnt_60_mem<=cnt_60;
cnt_61_mem<=cnt_61;
cnt_62_mem<=cnt_62;
cnt_63_mem<=cnt_63;
cnt_64_mem<=cnt_64;
cnt_65_mem<=cnt_65;
cnt_66_mem<=cnt_66;
cnt_67_mem<=cnt_67;
cnt_68_mem<=cnt_68;
cnt_69_mem<=cnt_69;
cnt_70_mem<=cnt_70;
cnt_71_mem<=cnt_71;
cnt_72_mem<=cnt_72;
cnt_73_mem<=cnt_73;
cnt_74_mem<=cnt_74;
```

```
cnt_75_mem<=cnt_75;
cnt_76_mem<=cnt_76;
cnt_77_mem<=cnt_77;
cnt_78_mem<=cnt_78;
cnt_79_mem<=cnt_79;
cnt_80_mem<=cnt_80;
cnt_81_mem<=cnt_81;
cnt_82_mem<=cnt_82;
cnt_83_mem<=cnt_83;
cnt_84_mem<=cnt_84;
cnt_85_mem<=cnt_85;
cnt_86_mem<=cnt_86;
cnt_87_mem<=cnt_87;
cnt_88_mem<=cnt_88;
cnt_89_mem<=cnt_89;
cnt_90_mem<=cnt_90;
cnt_91_mem<=cnt_91;
cnt_92_mem<=cnt_92;
cnt_93_mem<=cnt_93;
cnt_94_mem<=cnt_94;
cnt_95_mem<=cnt_95;
cnt_96_mem<=cnt_96;
cnt_97_mem<=cnt_97;
```

```
cnt_98_mem<=cnt_98;
cnt_99_mem<=cnt_99;
cnt_100_mem<=cnt_100;
cnt_101_mem<=cnt_101;
cnt_102_mem<=cnt_102;
cnt_103_mem<=cnt_103;
cnt_104_mem<=cnt_104;
cnt_105_mem<=cnt_105;
cnt_106_mem<=cnt_106;
cnt_107_mem<=cnt_107;
cnt_108_mem<=cnt_108;
cnt_109_mem<=cnt_109;
cnt_110_mem<=cnt_110;
cnt_111_mem<=cnt_111;
cnt_112_mem<=cnt_112;
cnt_113_mem<=cnt_113;
cnt_114_mem<=cnt_114;
cnt_115_mem<=cnt_115;
cnt_116_mem<=cnt_116;
cnt_117_mem<=cnt_117;
cnt_118_mem<=cnt_118;
cnt_119_mem<=cnt_119;
cnt_120_mem<=cnt_120;
```

```
cnt_121_mem<=cnt_121;
cnt_122_mem<=cnt_122;
cnt_123_mem<=cnt_123;
cnt_124_mem<=cnt_124;
cnt_125_mem<=cnt_125;

cnt_1<=0;
cnt_2<=0;
cnt_3<=0;
cnt_4<=0;
cnt_5<=0;
cnt_6<=0;
cnt_7<=0;
cnt_8<=0;
cnt_9<=0;
cnt_10<=0;
cnt_11<=0;
cnt_12<=0;
cnt_13<=0;
cnt_14<=0;
cnt_15<=0;
cnt_16<=0;
cnt_17<=0;
```

```
cnt_18<=0;
cnt_19<=0;
cnt_20<=0;
cnt_21<=0;
cnt_22<=0;
cnt_23<=0;
cnt_24<=0;
cnt_25<=0;
cnt_26<=0;
cnt_27<=0;
cnt_28<=0;
cnt_29<=0;
cnt_30<=0;
cnt_31<=0;
cnt_32<=0;
cnt_33<=0;
cnt_34<=0;
cnt_35<=0;
cnt_36<=0;
cnt_37<=0;
cnt_38<=0;
cnt_39<=0;
cnt_40<=0;
```

```
cnt_41<=0;
cnt_42<=0;
cnt_43<=0;
cnt_44<=0;
cnt_45<=0;
cnt_46<=0;
cnt_47<=0;
cnt_48<=0;
cnt_49<=0;
cnt_50<=0;
cnt_51<=0;
cnt_52<=0;
cnt_53<=0;
cnt_54<=0;
cnt_55<=0;
cnt_56<=0;
cnt_57<=0;
cnt_58<=0;
cnt_59<=0;
cnt_60<=0;
cnt_61<=0;
cnt_62<=0;
cnt_63<=0;
```

```
cnt_64<=0;
cnt_65<=0;
cnt_66<=0;
cnt_67<=0;
cnt_68<=0;
cnt_69<=0;
cnt_70<=0;
cnt_71<=0;
cnt_72<=0;
cnt_73<=0;
cnt_74<=0;
cnt_75<=0;
cnt_76<=0;
cnt_77<=0;
cnt_78<=0;
cnt_79<=0;
cnt_80<=0;
cnt_81<=0;
cnt_82<=0;
cnt_83<=0;
cnt_84<=0;
cnt_85<=0;
cnt_86<=0;
```

```
cnt_87<=0;
cnt_88<=0;
cnt_89<=0;
cnt_90<=0;
cnt_91<=0;
cnt_92<=0;
cnt_93<=0;
cnt_94<=0;
cnt_95<=0;
cnt_96<=0;
cnt_97<=0;
cnt_98<=0;
cnt_99<=0;
cnt_100<=0;
cnt_101<=0;
cnt_102<=0;
cnt_103<=0;
cnt_104<=0;
cnt_105<=0;
cnt_106<=0;
cnt_107<=0;
cnt_108<=0;
cnt_109<=0;
```

```
        cnt_110<=0;
        cnt_111<=0;
        cnt_112<=0;
        cnt_113<=0;
        cnt_114<=0;
        cnt_115<=0;
        cnt_116<=0;
        cnt_117<=0;
        cnt_118<=0;
        cnt_119<=0;
        cnt_120<=0;
        cnt_121<=0;
        cnt_122<=0;
        cnt_123<=0;
        cnt_124<=0;
        cnt_125<=0;

        end
    else if(en_apd_rise && cnt_en)
        begin
        cnt<=cnt+1; ///  cnt fot the inside counter  1s
        data_out_en<=0;
        number_counter <= data_in_coarse*25-data_in;
```

```
case(number_counter)

//////////0ns~~20ns, 40 steps

1:                                                          //1
cnt_1<=cnt_1+1;
2:                                                          //2
cnt_2<=cnt_2+1;
3:                                                          //3
cnt_3<=cnt_3+1;
4:                                                          //4
cnt_4<=cnt_4+1;
5:                                                          //5
cnt_5<=cnt_5+1;
6:                                                          //6
cnt_6<=cnt_6+1;
7:                                                          //7
cnt_7<=cnt_7+1;
8:                                                          //8
cnt_8<=cnt_8+1;
9:                                                          //9
cnt_9<=cnt_9+1;
10:                                                         //10
```

```
cnt_10<=cnt_10+1;


11:                                                    //12
cnt_11<=cnt_11+1;
12:                                                    //13
cnt_12<=cnt_12+1;
13:                                                    //14
cnt_13<=cnt_13+1;
14:                                                    //15
cnt_14<=cnt_14+1;
15:                                                    //16
cnt_15<=cnt_15+1;
16:                                                    //17
cnt_16<=cnt_16+1;
17:                                                    //18
cnt_17<=cnt_17+1;
18:                                                    //19
cnt_18<=cnt_18+1;
19:                                                    //20
cnt_19<=cnt_19+1;
20:                                                    //21
cnt_20<=cnt_20+1;
21:                                                    //22
```

```
cnt_21<=cnt_21+1;
22:                                                    //23
cnt_22<=cnt_22+1;
23:                                                    //24
cnt_23<=cnt_23+1;
24:                                                    //25
cnt_24<=cnt_24+1;
25:                                                    //26
cnt_25<=cnt_25+1;
26:                                                    //27
cnt_26<=cnt_26+1;
27:                                                    //28
cnt_27<=cnt_27+1;
28:                                                    //29
cnt_28<=cnt_28+1;
29:                                                    //30
cnt_29<=cnt_29+1;
30:                                                    //31
cnt_30<=cnt_30+1;
31:                                                    //32
cnt_31<=cnt_31+1;
32:                                                    //33
cnt_32<=cnt_32+1;
```

```
33:                                                                    //34
cnt_33<=cnt_33+1;
34:                                                                    //35
cnt_34<=cnt_34+1;
35:                                                                    //36
cnt_35<=cnt_35+1;
36:                                                                    //37
cnt_36<=cnt_36+1;
37:                                                                    //38
cnt_37<=cnt_37+1;
38:                                                                    //39
cnt_38<=cnt_38+1;
39:                                                                    //40
cnt_39<=cnt_39+1;
40:                                                                    //40
cnt_40<=cnt_40+1;
41:                                                                     //1
cnt_41<=cnt_41+1;
42:                                                                     //2
cnt_42<=cnt_42+1;
43:                                                                     //3
cnt_43<=cnt_43+1;
44:                                                                     //4
```

```
cnt_44<=cnt_44+1;
45:                                                    //5
cnt_45<=cnt_45+1;
46:                                                    //6
cnt_46<=cnt_46+1;
47:                                                    //7
cnt_47<=cnt_47+1;
48:                                                    //8
cnt_48<=cnt_48+1;
49:                                                    //9
cnt_49<=cnt_49+1;
50:                                                    //0
cnt_50<=cnt_50+1;
51:                                                    //1
cnt_51<=cnt_51+1;
52:                                                    //2
cnt_52<=cnt_52+1;
53:                                                    //3
cnt_53<=cnt_53+1;
54:                                                    //4
cnt_54<=cnt_54+1;
55:                                                    //5
cnt_55<=cnt_55+1;
```

```
56:                                                                    //6
cnt_56<=cnt_56+1;
57:                                                                    //7
cnt_57<=cnt_57+1;
58:                                                                    //8
cnt_58<=cnt_58+1;
59:                                                                    //9
cnt_59<=cnt_59+1;
60:                                                                    //0
cnt_60<=cnt_60+1;
61:                                                                    //1
cnt_61<=cnt_61+1;
62:                                                                    //2
cnt_62<=cnt_62+1;
63:                                                                    //3
cnt_63<=cnt_63+1;
64:                                                                    //4
cnt_64<=cnt_64+1;
65:                                                                    //5
cnt_65<=cnt_65+1;
66:                                                                    //6
cnt_66<=cnt_66+1;
67:                                                                    //7
```

```
cnt_67<=cnt_67+1;
68:                                                    //8
cnt_68<=cnt_68+1;
69:                                                    //9
cnt_69<=cnt_69+1;
70:                                                    //0
cnt_70<=cnt_70+1;
71:                                                    //1
cnt_71<=cnt_71+1;
72:                                                    //2
cnt_72<=cnt_72+1;
73:                                                    //3
cnt_73<=cnt_73+1;
74:                                                    //4
cnt_74<=cnt_74+1;
75:                                                    //5
cnt_75<=cnt_75+1;
76:                                                    //6
cnt_76<=cnt_76+1;
77:                                                    //7
cnt_77<=cnt_77+1;
78:                                                    //8
cnt_78<=cnt_78+1;
```

```
79:                                                                    //9
cnt_79<=cnt_79+1;
80:                                                                    //0
cnt_80<=cnt_80+1;
81:                                                                    //1
cnt_81<=cnt_81+1;
82:                                                                    //2
cnt_82<=cnt_82+1;
83:                                                                    //3
cnt_83<=cnt_83+1;
84:                                                                    //4
cnt_84<=cnt_84+1;
85:                                                                    //5
cnt_85<=cnt_85+1;
86:                                                                    //6
cnt_86<=cnt_86+1;
87:                                                                    //7
cnt_87<=cnt_87+1;
88:                                                                    //8
cnt_88<=cnt_88+1;
89:                                                                    //9
cnt_89<=cnt_89+1;
90:                                                                    //0
```

```
cnt_90<=cnt_90+1;
91:                                                    //1
cnt_91<=cnt_91+1;
92:                                                    //2
cnt_92<=cnt_92+1;
93:                                                    //3
cnt_93<=cnt_93+1;
94:                                                    //4
cnt_94<=cnt_94+1;
95:                                                    //5
cnt_95<=cnt_95+1;
96:                                                    //6
cnt_96<=cnt_96+1;
97:                                                    //7
cnt_97<=cnt_97+1;
98:                                                    //8
cnt_98<=cnt_98+1;
99:                                                    //9
cnt_99<=cnt_99+1;
100:                                                   //0
cnt_100<=cnt_100+1;
101:                                                   //1
cnt_101<=cnt_101+1;
```

```
102:                                                    //2
cnt_102<=cnt_102+1;
103:                                                    //3
cnt_103<=cnt_103+1;
104:                                                    //4
cnt_104<=cnt_104+1;
105:                                                    //5
cnt_105<=cnt_105+1;
106:                                                    //6
cnt_106<=cnt_106+1;
107:                                                    //7
cnt_107<=cnt_107+1;
108:                                                    //8
cnt_108<=cnt_108+1;
109:                                                    //9
cnt_109<=cnt_109+1;
110:                                                    //0
cnt_110<=cnt_110+1;
111:                                                    //1
cnt_111<=cnt_111+1;
112:                                                    //2
cnt_112<=cnt_112+1;
113:                                                    //3
```

```
cnt_113<=cnt_113+1;
114:                                                    //4
cnt_114<=cnt_114+1;
115:                                                    //5
cnt_115<=cnt_115+1;
116:                                                    //6
cnt_116<=cnt_116+1;
117:                                                    //7
cnt_117<=cnt_117+1;
118:                                                    //8
cnt_118<=cnt_118+1;
119:                                                    //9
cnt_119<=cnt_119+1;
120:                                                    //0
cnt_120<=cnt_120+1;
121:                                                    //1
cnt_121<=cnt_121+1;
122:                                                    //2
cnt_122<=cnt_122+1;
123:                                                    //3
cnt_123<=cnt_123+1;
124:                                                    //4
cnt_124<=cnt_124+1;
```

```
        125:                                                    //5
        cnt_125<=cnt_125+1;
        endcase
        end
        end
end
endmodule
```

1.4 Fine counter

```
module delay_cell_2(
        data_in,
        clk,
        Q0,
        Q1,
        Q2,
        Q3,
        Q4,
        Q5,
        Q6,
        Q7,
        Q8,
        Q9,
        Q10,
```

Q11,

Q12,

Q13,

Q14,

Q15,

Q16,

Q17,

Q18,

Q19,

Q20,

Q21,

Q22,

Q23,

Q24,

Q25,

Q26,

Q27,

Q28,

Q29,

Q30,

Q31,

Q32,

Q33,

```
    Q34,
    Q35,
    Q36,
    Q37,
    Q38,
    Q39
);


input                                   data_in;
input                                   clk;
output                                  Q0;
output                                  Q1;
output                                  Q2;
output                                  Q3;
output                                  Q4;
output                                  Q5;
output                                  Q6;
output                                  Q7;
output                                  Q8;
output                                  Q9;
output                                  Q10;
output                                  Q11;
```

output Q12;

output Q13;

output Q14;

output Q15;

output Q16;

output Q17;

output Q18;

output Q19;

output Q20;

output Q21;

output Q22;

output Q23;

output Q24;

output Q25;

output Q26;

output Q27;

output Q28;

output Q29;

output Q30;

output Q31;

output Q32;

output Q33;

output Q34;

```
output Q35;
output Q36;
output Q37;
output Q38;
output Q39;

wire SYNTHESIZED_WIRE_79;
wire SYNTHESIZED_WIRE_80;
wire SYNTHESIZED_WIRE_81;
wire SYNTHESIZED_WIRE_82;
wire SYNTHESIZED_WIRE_83;
wire SYNTHESIZED_WIRE_84;
wire SYNTHESIZED_WIRE_85;
wire SYNTHESIZED_WIRE_7;
wire SYNTHESIZED_WIRE_86;
wire SYNTHESIZED_WIRE_87;
wire SYNTHESIZED_WIRE_88;
wire SYNTHESIZED_WIRE_89;
wire SYNTHESIZED_WIRE_90;
wire SYNTHESIZED_WIRE_91;
wire SYNTHESIZED_WIRE_92;
wire SYNTHESIZED_WIRE_93;
wire SYNTHESIZED_WIRE_94;
```

```
wire SYNTHESIZED_WIRE_95;
wire SYNTHESIZED_WIRE_96;
wire SYNTHESIZED_WIRE_97;
wire SYNTHESIZED_WIRE_98;
wire SYNTHESIZED_WIRE_99;
wire SYNTHESIZED_WIRE_100;
wire SYNTHESIZED_WIRE_101;
wire SYNTHESIZED_WIRE_102;
wire SYNTHESIZED_WIRE_103;
wire SYNTHESIZED_WIRE_104;
wire SYNTHESIZED_WIRE_105;
wire SYNTHESIZED_WIRE_106;
wire SYNTHESIZED_WIRE_107;
wire SYNTHESIZED_WIRE_108;
wire SYNTHESIZED_WIRE_109;
wire SYNTHESIZED_WIRE_110;
wire SYNTHESIZED_WIRE_111;
wire SYNTHESIZED_WIRE_112;
wire SYNTHESIZED_WIRE_113;
wire SYNTHESIZED_WIRE_114;
wire SYNTHESIZED_WIRE_115;
wire SYNTHESIZED_WIRE_116;
wire SYNTHESIZED_WIRE_117;
```

```
delay_t1                                                    b2v_inst(
    .data_in(data_in),
    .data_out(SYNTHESIZED_WIRE_83));


delay_t1                                                    b2v_inst10(
    .data_in(SYNTHESIZED_WIRE_79),
    .data_out(SYNTHESIZED_WIRE_93));


dff_cell                                                    b2v_inst101(
    .data(SYNTHESIZED_WIRE_80),
    .clock(clk),
    .q(Q34));


dff_cell                                                    b2v_inst104(
    .data(SYNTHESIZED_WIRE_81),
```

```
    .clock(clk),
    .q(Q35));


dff_cell                                                    b2v_inst107(
    .data(SYNTHESIZED_WIRE_82),
    .clock(clk),
    .q(Q36));


dff_cell                                                     b2v_inst11(
    .data(SYNTHESIZED_WIRE_83),
    .clock(clk),
    .q(Q0));


dff_cell                                                    b2v_inst110(
    .data(SYNTHESIZED_WIRE_84),
    .clock(clk),
    .q(Q37));


dff_cell                                                    b2v_inst113(
```

```
    .data(SYNTHESIZED_WIRE_85),
    .clock(clk),
    .q(Q38));


dff_cell                                                    b2v_inst116(
    .data(SYNTHESIZED_WIRE_7),
    .clock(clk),
    .q(Q39));


dff_cell                                                    b2v_inst12(
    .data(SYNTHESIZED_WIRE_86),
    .clock(clk),
    .q(Q1));


dff_cell                                                    b2v_inst13(
    .data(SYNTHESIZED_WIRE_87),
    .clock(clk),
    .q(Q2));
```

```
dff_cell                                                    b2v_inst14(
    .data(SYNTHESIZED_WIRE_88),
    .clock(clk),
    .q(Q3));


dff_cell                                                    b2v_inst15(
    .data(SYNTHESIZED_WIRE_89),
    .clock(clk),
    .q(Q4));


dff_cell                                                    b2v_inst16(
    .data(SYNTHESIZED_WIRE_90),
    .clock(clk),
    .q(Q5));


dff_cell                                                    b2v_inst17(
    .data(SYNTHESIZED_WIRE_91),
    .clock(clk),
    .q(Q6));
```

```
dff_cell                                                    b2v_inst18(
    .data(SYNTHESIZED_WIRE_92),
    .clock(clk),
    .q(Q7));


delay_t1                                                    b2v_inst19(
    .data_in(SYNTHESIZED_WIRE_93),
    .data_out(SYNTHESIZED_WIRE_94));


delay_t1                                                     b2v_inst2(
    .data_in(SYNTHESIZED_WIRE_83),
    .data_out(SYNTHESIZED_WIRE_86));


delay_t1                                                    b2v_inst20(
    .data_in(SYNTHESIZED_WIRE_94),
    .data_out(SYNTHESIZED_WIRE_95));


delay_t1                                                    b2v_inst21(
```

```
    .data_in(SYNTHESIZED_WIRE_95),
    .data_out(SYNTHESIZED_WIRE_96));


delay_t1                                                    b2v_inst22(
    .data_in(SYNTHESIZED_WIRE_96),
    .data_out(SYNTHESIZED_WIRE_97));


delay_t1                                                    b2v_inst23(
    .data_in(SYNTHESIZED_WIRE_97),
    .data_out(SYNTHESIZED_WIRE_98));


delay_t1                                                    b2v_inst24(
    .data_in(SYNTHESIZED_WIRE_98),
    .data_out(SYNTHESIZED_WIRE_99));


dff_cell                                                    b2v_inst25(
    .data(SYNTHESIZED_WIRE_79),
    .clock(clk),
    .q(Q8));
```

```
delay_t1                                                    b2v_inst26(
    .data_in(SYNTHESIZED_WIRE_99),
    .data_out(SYNTHESIZED_WIRE_100));


delay_t1                                                    b2v_inst27(
    .data_in(SYNTHESIZED_WIRE_100),
    .data_out(SYNTHESIZED_WIRE_101));


dff_cell                                                    b2v_inst28(
    .data(SYNTHESIZED_WIRE_93),
    .clock(clk),
    .q(Q9));


delay_t1                                                    b2v_inst29(
    .data_in(SYNTHESIZED_WIRE_101),
    .data_out(SYNTHESIZED_WIRE_102));
```

```
delay_t1                                                    b2v_inst3(
    .data_in(SYNTHESIZED_WIRE_86),
    .data_out(SYNTHESIZED_WIRE_87));


delay_t1                                                    b2v_inst30(
    .data_in(SYNTHESIZED_WIRE_102),
    .data_out(SYNTHESIZED_WIRE_103));


dff_cell                                                    b2v_inst31(
    .data(SYNTHESIZED_WIRE_94),
    .clock(clk),
    .q(Q10));


delay_t1                                                    b2v_inst33(
    .data_in(SYNTHESIZED_WIRE_103),
    .data_out(SYNTHESIZED_WIRE_104));


dff_cell                                                    b2v_inst34(
    .data(SYNTHESIZED_WIRE_95),
```

```
    .clock(clk),
    .q(Q11));


delay_t1                                                    b2v_inst35(
    .data_in(SYNTHESIZED_WIRE_104),
    .data_out(SYNTHESIZED_WIRE_105));


delay_t1                                                    b2v_inst36(
    .data_in(SYNTHESIZED_WIRE_105),
    .data_out(SYNTHESIZED_WIRE_106));


dff_cell                                                    b2v_inst37(
    .data(SYNTHESIZED_WIRE_96),
    .clock(clk),
    .q(Q12));


delay_t1                                                    b2v_inst38(
    .data_in(SYNTHESIZED_WIRE_106),
    .data_out(SYNTHESIZED_WIRE_107));
```

```
delay_t1                                                    b2v_inst39(
    .data_in(SYNTHESIZED_WIRE_107),
    .data_out(SYNTHESIZED_WIRE_108));


delay_t1                                                     b2v_inst4(
    .data_in(SYNTHESIZED_WIRE_87),
    .data_out(SYNTHESIZED_WIRE_88));


dff_cell                                                    b2v_inst40(
    .data(SYNTHESIZED_WIRE_97),
    .clock(clk),
    .q(Q13));


delay_t1                                                    b2v_inst41(
    .data_in(SYNTHESIZED_WIRE_108),
    .data_out(SYNTHESIZED_WIRE_109));
```

```
delay_t1                                                    b2v_inst42(
    .data_in(SYNTHESIZED_WIRE_109),
    .data_out(SYNTHESIZED_WIRE_110));


dff_cell                                                    b2v_inst43(
    .data(SYNTHESIZED_WIRE_98),
    .clock(clk),
    .q(Q14));


delay_t1                                                    b2v_inst44(
    .data_in(SYNTHESIZED_WIRE_110),
    .data_out(SYNTHESIZED_WIRE_111));


delay_t1                                                    b2v_inst45(
    .data_in(SYNTHESIZED_WIRE_111),
    .data_out(SYNTHESIZED_WIRE_112));


dff_cell                                                    b2v_inst46(
    .data(SYNTHESIZED_WIRE_99),
```

```
    .clock(clk),
    .q(Q15));


delay_t1                                                    b2v_inst47(
    .data_in(SYNTHESIZED_WIRE_112),
    .data_out(SYNTHESIZED_WIRE_113));


dff_cell                                                    b2v_inst48(
    .data(SYNTHESIZED_WIRE_100),
    .clock(clk),
    .q(Q16));


delay_t1                                                    b2v_inst49(
    .data_in(SYNTHESIZED_WIRE_113),
    .data_out(SYNTHESIZED_WIRE_114));


delay_t1                                                     b2v_inst5(
    .data_in(SYNTHESIZED_WIRE_88),
    .data_out(SYNTHESIZED_WIRE_89));
```

```
dff_cell                                                    b2v_inst50(
    .data(SYNTHESIZED_WIRE_101),
    .clock(clk),
    .q(Q17));


delay_t1                                                    b2v_inst51(
    .data_in(SYNTHESIZED_WIRE_114),
    .data_out(SYNTHESIZED_WIRE_115));


delay_t1                                                    b2v_inst52(
    .data_in(SYNTHESIZED_WIRE_115),
    .data_out(SYNTHESIZED_WIRE_116));


dff_cell                                                    b2v_inst53(
    .data(SYNTHESIZED_WIRE_102),
    .clock(clk),
    .q(Q18));
```

```
delay_t1                                                    b2v_inst54(
    .data_in(SYNTHESIZED_WIRE_116),
    .data_out(SYNTHESIZED_WIRE_117));


delay_t1                                                    b2v_inst55(
    .data_in(SYNTHESIZED_WIRE_117),
    .data_out(SYNTHESIZED_WIRE_80));


dff_cell                                                    b2v_inst56(
    .data(SYNTHESIZED_WIRE_103),
    .clock(clk),
    .q(Q19));


delay_t1                                                    b2v_inst57(
    .data_in(SYNTHESIZED_WIRE_80),
    .data_out(SYNTHESIZED_WIRE_81));


delay_t1                                                    b2v_inst58(
```

```
    .data_in(SYNTHESIZED_WIRE_81),
    .data_out(SYNTHESIZED_WIRE_82));


dff_cell                                                    b2v_inst59(
    .data(SYNTHESIZED_WIRE_104),
    .clock(clk),
    .q(Q20));


delay_t1                                                    b2v_inst6(
    .data_in(SYNTHESIZED_WIRE_89),
    .data_out(SYNTHESIZED_WIRE_90));


delay_t1                                                    b2v_inst60(
    .data_in(SYNTHESIZED_WIRE_82),
    .data_out(SYNTHESIZED_WIRE_84));


delay_t1                                                    b2v_inst61(
    .data_in(SYNTHESIZED_WIRE_84),
    .data_out(SYNTHESIZED_WIRE_85));
```

```
dff_cell                                                    b2v_inst62(
    .data(SYNTHESIZED_WIRE_105),
    .clock(clk),
    .q(Q21));


delay_t1                                                    b2v_inst63(
    .data_in(SYNTHESIZED_WIRE_85),
    .data_out(SYNTHESIZED_WIRE_7));


dff_cell                                                    b2v_inst65(
    .data(SYNTHESIZED_WIRE_106),
    .clock(clk),
    .q(Q22));


dff_cell                                                    b2v_inst68(
    .data(SYNTHESIZED_WIRE_107),
    .clock(clk),
    .q(Q23));
```

```
delay_t1                                                    b2v_inst7(
    .data_in(SYNTHESIZED_WIRE_90),
    .data_out(SYNTHESIZED_WIRE_91));


dff_cell                                                    b2v_inst70(
    .data(SYNTHESIZED_WIRE_108),
    .clock(clk),
    .q(Q24));


dff_cell                                                    b2v_inst72(
    .data(SYNTHESIZED_WIRE_109),
    .clock(clk),
    .q(Q25));


dff_cell                                                    b2v_inst75(
    .data(SYNTHESIZED_WIRE_110),
    .clock(clk),
    .q(Q26));
```

```
dff_cell                                                    b2v_inst78(
    .data(SYNTHESIZED_WIRE_111),
    .clock(clk),
    .q(Q27));


delay_t1                                                    b2v_inst8(
    .data_in(SYNTHESIZED_WIRE_91),
    .data_out(SYNTHESIZED_WIRE_92));


dff_cell                                                    b2v_inst81(
    .data(SYNTHESIZED_WIRE_112),
    .clock(clk),
    .q(Q28));


dff_cell                                                    b2v_inst84(
    .data(SYNTHESIZED_WIRE_113),
    .clock(clk),
    .q(Q29));
```

```
dff_cell                                                    b2v_inst87(
    .data(SYNTHESIZED_WIRE_114),
    .clock(clk),
    .q(Q30));


delay_t1                                                    b2v_inst9(
    .data_in(SYNTHESIZED_WIRE_92),
    .data_out(SYNTHESIZED_WIRE_79));


dff_cell                                                    b2v_inst90(
    .data(SYNTHESIZED_WIRE_115),
    .clock(clk),
    .q(Q31));


dff_cell                                                    b2v_inst96(
    .data(SYNTHESIZED_WIRE_116),
    .clock(clk),
    .q(Q32));
```

```
dff_cell                                                    b2v_inst98(
    .data(SYNTHESIZED_WIRE_117),
    .clock(clk),
    .q(Q33));


    endmodule
```

2. Matlab Code

```
%% Real time data collection from FPGA through uart
%% Create the serial object
clear all;
s = serial('COM1','BaudRate', 115200,'DataBits',7);
fopen(s);

%% counting
test_time=10;          %seconds
time_resolution=5;     %ns
histgram_period=200;   %ns
accumulation_time=1;   %seconds

figure;
for j=1:test_time
  for i=1:(histgram_period/time_resolution)
  fprintf(s, '*IDN?');
  Count(i)= fscanf(s,'%u');
  end
  plot((1:(histgram_period/time_resolution))*5,Count,'--rs');
  Count_sum(j) =sum(Count);
  %plot(j,Count_sum(j),'*');
  %hold on;
```

```
  pause(accumulation_time);
end
hold off;

%% plot total count at each accummulation_time in the whole test_time.
figure;
plot((1:(test_time/accumulation_time))*accumulation_time, Count_sum,'--rs');

%% release port
fclose(s);
delete(s);
clear s;
```