**Transceiver ADPLL Design**



**Submitted for the 2019 Digilent Design Contest Europe**

**11-12 May 2019**

**Advisor: Phd Orhan Gazi**

**Çankaya University**

**Ankara / Turkey**

****

**Seda ESEN\* Berkay ERGÜN\* Hasan TOSKAR\***

**sedaa\_esen@hotmail.com ergnberkay@gmail.com toskarhasan@gmail.com**

\*Equally contribution

**Contents**

1. Introduction ……………………………………………………………………………………………………….. 3

2. Design & Commentary ………………………………………………………………………………………….. 4

2.1. Transmitter Design & Commentary ………………………………………………………………………… 5

2.2. Square Root Raised Cosine(SRRC) Filter Design & Commentary ……………………………………... 6

2.3. Receiver Design & Commentary …………………………………………………………………………... 8

# 2.4. Digital Phase Lock Loop Design & Commentary …………………………………………………………. 9

# 2.5. Digital Phase Lock Loop Theory …………………………………………………………………………... 10

# 2.6. Ladder Circuit Design & Commentary ……………………………………………………………………. 13

# 2.7. Microprocessor Converter Circuit Design & Commentary ……………………………………………… 14

3. Discussions ……………………………………………………………………………………………………… 15

4. Engineering resources used …………………………………………………………………………………... 15

5. Marketability ……………………………………………………………………………………………………… 15

6. Community Feedback …………………………………………………………………………………………... 16

7. Reference ………………………………………………………………………………………………………… 16

8. SRRC Implementation Testing Code & Result on Matlab ………………………………………………….. 17

9. Transmitter Implementation Testing Code & Result on Matlab ……………………………………………. 18

10. Receiver Implementation Testing Code & Result on Matlab ……………………………………………… 21

11. SRRC Implementation Testing Code & Result on ISE and Oscilloscope ……………………………….. 23

12. Transmiter Implementation Testing Code & Result on ISE and Oscilloscope ………………………….. 24

13. PLL Implementation Code & Result on Vivado …………………………………………………………….. 27

14. Receiver Implementation Code & Result on Vivado ………………………………………………………. 31

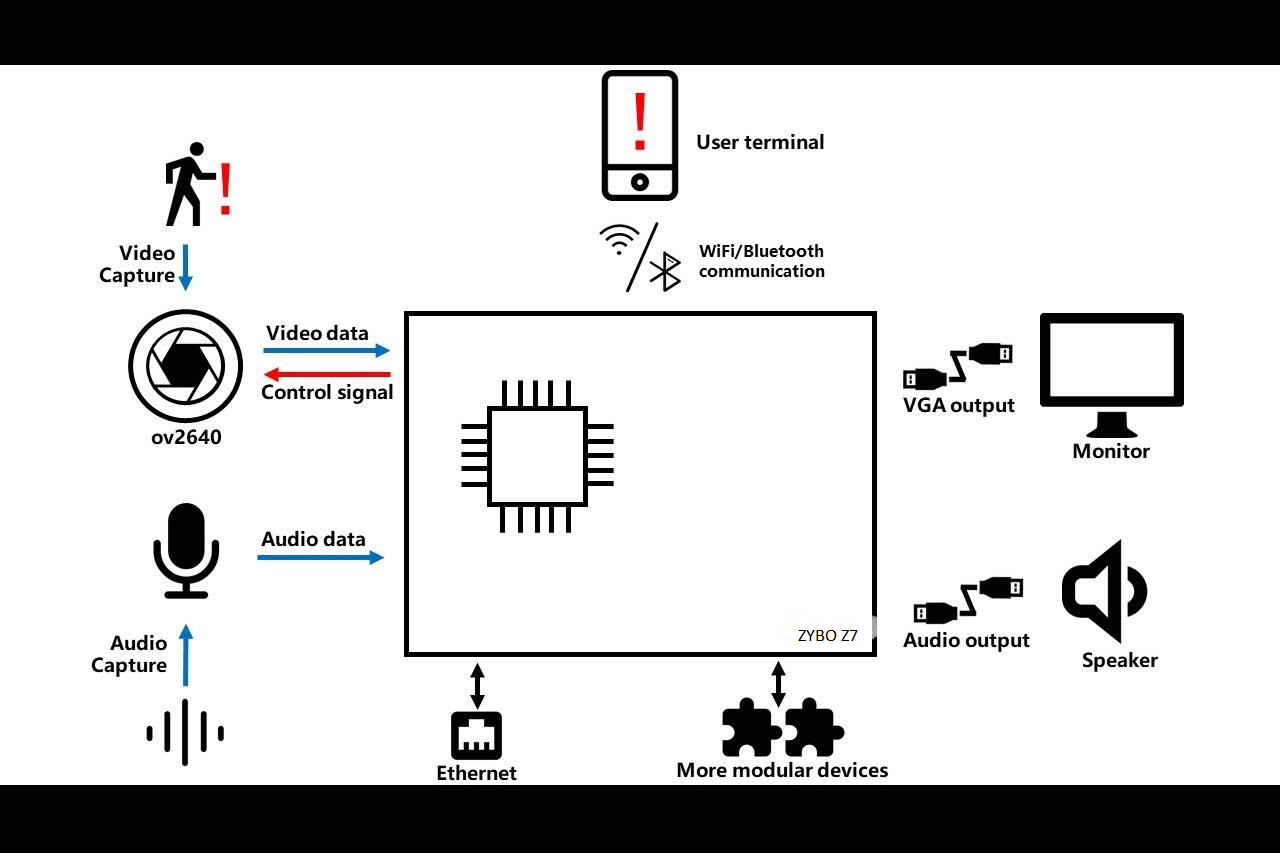
# Introduction

This project aims to create transmission line between two links using FPGA cards and solve the synchronization problems with ADPLL. Analog communication and synchronization systems are always been popular but not quite effective for any communication system. This project uses digital data bits to send any messages through any communication link with minimum phase errors.The circuit was designed on a Windows 10 PC ,Matlab 2016a, Vivado 217 Design Suite,Ladder circuit and Microcontroller PIC 16F877A Analog Digital Converter.

Required tools:

* Digilent Zybo board.
* Basys 3 board
* Ladder Circuit
* Microcontroller PIC 16F877A Analog Digital Converter
* PC running OS Windows 10 (or greater)
* 2 antennas

The project offered can be used in machine to machine communication. In this case, it can make the embedded system. It is used for fast and simultaneous analysis of incoming data from outside(other vehicle data) or inside(sensors). Some wireless communication systems may offer opportunities to communicate with other vehicles. Also another application of these systems is smart home systems. FPGA centered smart home system can beneficial to control all the sensors, lightbulbs and camera systems.



# Design & Commentary

Block diagram of our project is given in general. The block diagrams given below are mainly divided into two main groups as receiver and transmitter. In this section, the main sections of the receiver and transmitter sections, SRRC and PLL in block diagrams are examined. D / A converter is used to convert the data in the transmitter to analog signal. We used the ladder circuit instead of the D/A conversion. A total of two RF circuits were used in the receiver and transmitter for wireless communication of the system. On the receiving side, A/D converter was used to convert analog signal to digital signal. We used PIC 16F877A instead of A/ D converter.

|  |
| --- |
|  |
| *Figure 1.General Block Diagram* |
| 2.1. Transmitter Design & Commentary Transmitter block diagram is shown in figure 2. The mapping circuit decomposes input bit stream into In-phase (I) and Quadrature (Q) components according to the constellation assignment. Up-sampling produces the sequence that would have been obtained by sampling the signal at a higher rate. Square Root Raise Cosine (SRRC) filter is generally pulse-shaping filter in modulator which reduces Inter Symbol Interference (ISI). After multiplied by carrier signal, both in-phase and quadrature signals are combined at adder circuit to transmit over channel   |  |  |  | | --- | --- | --- | | I/O TABLE | | | | Pin Name | Pin Direction | Description | | CLK | Input | Global clock line, from board oscillator. Frequency: 50MHz | | RST | Input | Reset input from button 0 of the board. | | Out | Output | input signal to see the final version (8-bit number) | | Clk\_out\_original | Signal | Clock\_Divider | | Count | Signal | slowed down fpga clock to see signal. Frequency is 1000Hz | | Count2 | Signal | 256 to check the incoming signals individually. Frequency is 2000Hz | | Count3 | Signal | To observe good sampling of data from incoming signal | | Y | Signal | Data from matlab | | Z | Signal | entered data (8-bit number) | | CON | Signal | Control mechanism (00,01,10,11) | |
|  |
| *Figure 2.Transmitter Block Diagram* |
| 2.2 Square Root Raised Cosine Filter (SRRC) Design & Commentary   We used square root raise cosine (SRRC) on both the receiver and the transmitter side of the project. In signal processing, a root elevated-cosine filter (RRC), sometimes known as a square root elevated-cosine filter (SRRC), is often used as a transmit and receive filter in a digital communication system to perform a matching filtering. This helps minimize inter-symbol interference (ISI).   |  |  |  | | --- | --- | --- | | I/O TABLE | | | | Pin Name | Pin Direction | Description | | CLK | Input | Global clock line, from board oscillator. Frequency: 100MHz | | RST | Input | Reset input from button 0 of the board. | | SquareRoot\_OUT | Output | Driven data matrix of square root raised cosine filter which is slowed by Clk\_out\_divided. | | Clk\_out\_divided | Signal | Signal that is 50\_000 times slower than Fpga’s initial clock |   Low-level blocks are schematic files, such as most modules that appear on pages. Low-level modules VHDL are written as mounting and generously interpreted. Zybo Z7 is applied to the main control module card, and the Basys 3 is applied to the second control module bays. |
| C:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\srrc-sematik.png |
| *Figure 3.Square Root Rasied Cosine Filter RTL Design* |
|  |
| *Figure 4.Transmitter RTL Design* |
| The figure 4 shows the Transmitter RTL Design. The first box above shows only one of the 256 bits coming in. Multiplication (MUX) is used here because of the decision-making mechanism mentioned in the following pages. A selection is made to determine which entry will be given to the output. MUX output is an 8-bit number.  In the third box RTL is used. RLT is means that registration transfer language (RTL). It is used to define the flow of data at the register-transfer level of an architect.  The second box shows the MUX and RLT connections. |
| 2.3. Receiver Design & Commentary  |  |  |  | | --- | --- | --- | | I/O TABLE | | | | Pin Name | Pin Direction | Description | | Clk\_in | Input | Global clock line, from board oscillator. Frequency: 100MHz | | Data | Input | Digital data that comes from analog digital converter to JA port of FPGA | | Clk\_out\_divided | Signal | Signal that is slower than clk\_in by 50000 times. | | clk\_256 | Signal | Receive serial data | | bit\_rec | Output | Shows every 8 bit data that come from continuous digital data matrix. |     QAM demodulator block diagram is shown in figure 5. A demodulation is an inverse process of modulation; it is used to recover the original information content from the modulated carrier wave. In demodulator matched filter is used. It is used to detect the transmitted pulses in the noisy receiver signal. Unlike the interleaving zeros between input samples, the matching filter cannot combine with the decimation factor because the Phase Locked Loop (PLL) requires all of its output for synchronization purpose. Phase Locked Loop is a key component in carrier and carrier and timing recovery. Phase detector, Loop filter and Numerically Controlled Oscillator (NCO) are elements of phase locked loop. |
|  |
| *Figure 5.Receiver Block Diagram* |
| 2.4. Digital Phase Lock Loop Design & Commentary In receiver part we present our main objective as ADPLL(All Digital Phase Locked Loop) design. In this part we use the schematic that shown in Figure 6 to create and code the design of it. Firstly external input that comes from transmitter fed to the circuits exor detector to detect a phase error. Then this exor output send to DN/UP pins of K-Counter part that fed by KClock which has period of M\*fo(fo is initial clock of fpga). After that K-Counter decides whether carry or borrow is active . If up-counter of K-counter is K/2 carry is active otherwise borrow is active. In the next part ID counter gives some delayed TFF in carry’s rising\_edge or gives some advanced TFF signal. In the end TFF signal and ID-Clock(which has same period with K-Clock) are put in to nor gate circuit and gives the IDout signal. IDout connected to N-Counter which is generated a signal frequency that divided N times of IDout signal frequency. N times divided IDout signal used as new input of exor detector and the system is looping after that.   |  |  |  | | --- | --- | --- | | I/O TABLE OF PLL | | | | Pin Name | Pin Direction | Description | | CLK | Input | Global clock line, from board oscillator. Frequency: 100MHz | | RST | Input | Reset input from button 0 of the board. | | EXOR\_out | Input | Exor operation of input signal v1 and v2 signal | | V1 | Input | Input Signal coming from Transmitter | | KClock | Input | Signal that K times slower than fpga clock | | IDClock | Input | Signal that 2\*N times slower than fpga clock | | V2 | Inout | Exor output | | IDout | Output | Nor gated version of ID Clock and TFF signals | | Kup | Signal | Counts only Exor\_out=0 condition | | Kdn | Signal | Counts only Exor\_out=1 condition | | Borrow | Signal | If Kdn=K/2, Borrow=1. If Kdn=0 , Borrow=0. | | Carry | Signal | K/2, Carry=1. If Kup=0 , Carry=0 | | TFF | Signal | :Is a signal that is slowest as half as the ID clock. Rising\_edge carry or borrow gives 0 or 1 values for 2 cycle | |
|  |
| *Figure 6.Phase Lock Loop (PLL) Block Diagram* |
| 2.5. Digital Phase Lock Loop TheoryK-counter Both counters count upwards.  Carry = 1 when contents of the  UP counter ≥ K/2.  Borrow = 1 when contents of the  DN counter ≥ K/2    • If the DN / UP input from the XOR gate process is ‘1’ as the bit, run the K-Counter Down Counter (DN). During this step, the Up Counter (UP) portion remains at its default value until the end of the Down Counter operation.  • If the DN / UP input from the XOR gate process is ‘0’ as the bit, run the K-Counter Up Counter (UP) part. During this step, the Down Counter (DN) portion remains at its default value until the end of the Down Counter operation.  As a result, K-Counter provides two values.  • CARRY: If the Up Counter part of the K-Counter is activated, the counter will count between 0 and K-1 (K = the entered counter value). In this case, if the counter reaches K / 2, the Carry output is active and forwarded to the ID Counter for the next step.  • BORROW: If the Down Counter part of the K-Counter is activated, the counter will count between 0 and K-1 (K = the entered counter value). In this case, if the counter reaches K / 2, Borrow output is active and forwarded to ID Counter for the next step. ID-counter   The opening of ID is “increment correction”. ID counter 74xx297 circuit DCO (digital controlled oscillator) functions. The ID counter processes the carrier and borrow signals from the K counter.  ID out= (not(id clock) and (not toggle FF))  *Figure 7.ID Counter Block Diagram Figure 8.Signal Set*  For figure b, if there is no CARRY and BORROW signals on our circuit, it works like the figure 2 on the side. The ID clock must always be equal to the 2Nfo frequency. The N number comes from the N counter.  If there is a carrier signal in the circuit, the ID out output should be as in Figure 9. First, id clock should be on the rising edge. If ID clock is on the rising edge, then the first rising edge of the carrier signal is toggle FF 1 cycle, which means that it moves in the form of “high-low-low-high-high”. Since Toggle FF has changed, the output of ID output is as in Figure.  *Figure 9.Advance Signal* *Figure 10.Delayed Sİgnal*  If we have a borrow signal in our circuit, the ID out output should be as in Figure 10. First, id clock should be on the rising edge. If ID clock is on the rising edge, then the first rising edge of the borrow signal is toggle FF 1 cycle, which means that it moves in the form of “high-low-high-low-high”. Since Toggle FF has changed, the output of ID output is as in Figure.    **N-Dividers**      *Figure 11.N-Counter Block Diagram*    Dividers form the main function in PLL circuits. A PLL circuit must cover a wide variety of continuous divisions for Crystal reference(V1) and VCO (ID counter). The main task is to split the signal received from VCO at the specified rate and send it to the phase detection section. In this way, two types of dividers are used according to frequency, high speed, low speed.  **High Frequency Dividers:** In high frequency dividers, the output of the VCO (id counter) is between 200-2500 MHz. For this purpose, the N must be very high, because the channel range is narrow, the N value is high for the prescaller is used.  **Low Frequency Dividers** The VCO (ID Counter) output in low frequency splitters has a value range of <100 Mhz and 100 Mhz. For this, binary modules can be used (two Dividers).    There are two types of N divider: Integer N Divider and Fractional N Divider.  **Integer N Divider:** The split rate of the frequency from VCO (ID Counter) is N and constant. In each cycle period, the VCO is divided by N.  **Fractional N Divider:** The split ratio of the frequency from the VCO (ID Counter) is N, but the fractionation ratio in the fractional N divider is not constant and varies according to the rounding of the N + (K (variable) / F (reference signal)) ratio. The calculated value is rounded off because there is no circuit that can work in the analog circuits. |
| *Figure 12.Phase Lock Loop (PLL) RTL Schematic*  Phase Lock Loop (PLL) RLT schematic is given as above. PLL's definition and Vivado's code are described later in the report. |
|  |
| 2.6. Ladder Circuit Design & Commentary   A resistor ladder is an electrical circuit made from repeating units of resistors. An R–2R ladder is a simple and inexpensive way to perform digital-to-analog conversion, using repetitive arrangements of precise resistor networks in a ladder-like configuration. A basic R–2R resistor ladder network is shown in Figure 13. Bit-7 (most significant bit, MSB) through Bit-6 (least significant bit, LSB) are driven from digital logic gates. Ideally, the bit inputs are switched between V = 0 (logic 0) and V = Vref (logic 1). The R–2R network causes these digital bits to be weighted in their contribution to the output voltage Vout. Depending on which bits are set to 1 and which to 0, the output voltage (Vout) will have a corresponding stepped value between 0 and Vref minus the value of the minimal step, corresponding to bit 0. The actual value of Vref (and the voltage of logic 0) will depend on the type of technology used to generate the digital signals. We used 9 10k 7 5k resistors and connected the 2R resistor jam cables to the serial. We connected the other end of the cable to the JA output pin of Zybo z7 (analog output). In this way, we have obtained a digital analog converter.     |  |  | | --- | --- | | Component Name | Description | | Resistors | Using for setting reference voltage | |
|  |
| *Figure 13.Block Diagram of Ladder Circuit* |
| 2.7. Microprocessor Converter Circuit Design & Commentary In this project PIC16F877A microprocessor unit used as analog-digital converter. In order to achieve this system we used the port-B as digital output and port-E as an analog input. Circuit is fed by outer analog signal with voltage value between 0-3.3 Volts. These voltage pulses decide the digital data numbers. Digital data that has been send is 8 bit and divided one by one in port-B‘s output so the analog-digital conversion is completed with that operation.  **Hardware Features**   * Operation under small voltage is possible. * PICs can work with batteries that are called dry batteries in the market. Because the operating voltages are between 2 and 6 Volts. * High current can be taken from the output pin. * The price is quite cheap. |
|  |
| *Figure 14.Microprocesser Main Circuit* |

1. **Discussion**

There were many problems along the way in this project. For the first time, MATLAB started the project, but a few problems were encountered in order to make the logic better fit. When SRRC code was executed, there were difficulties in the time plane. The SRRC chart was properly observed by trying this problem and giving values to the time interval each time. Second, when the SRRC graph is looked at, since a few values are infinite, the convolution process was performed and infinite values were observed each time. The solution was solved by using another kind of SRRC formula. In the code written for the recipient later '00’,'10’, '01’,'11' sinus and cosine to meet the requirements of the report should be collected with the AA section was mentioned, but cosine and sinus collected with the data from sinus-related operations, especially when we observed that this problem was solved by editing the time intervals of sinus and cosine again. In the second phase of the project, the code written in MATLAB had to be converted to VHDL. But in doing so, VHDL also observed errors every time because there was no impact. The solution is to set the data in MATLAB from 256 to 256, then VHDL code has been redesigned. The design of the project is not finished because of a lot of changes in the design and continues to be.

1. **Engineering resources Used**

This project was designed by Çankaya University students as ECE 494 course final project. The project lasted for approximately 8 months (250 Hours) and is continuing. A two-month (50 Hours) preliminary research and Article review was carried out for the selection of design, materials and methods in the project. It is necessary to know the high level Vhdl language for the software part of the project and to be able to use ISE and Vivado tools effectively. The hardware part of the project consists of 2 main parts except FPGA cards. These are Analog to digital converter and Digital to analog converter. For these circuits, the use of ladder and microprocessor is dominant. Therefore, it is necessary to have intermediate C language. The project is examined in 4 chapters, including two main headings (Transmitter and Receiver) and 2 headlines (Square Root Raised Cosine and Phase Look Loop). High-level Matlab was used for the test part of the project. The project was first translated into Vhdl language by checking the design via Matlab. An oscilloscope and DC source were used to observe Matlab and Vivado outputs.

1. **Marketability**

This project is designed to provide communication as simple. Based on this design, it is aimed to develop receiver-transmitter systems which can be used in 5G communication systems by developing communication from machine to machine. It is intended to be used as master in smart houses and to read data from sensors quickly and in parallel. We aim to market the PLL part of the project as IP core. Turkcell and Turk Telekom companies have been identified as the main recipients of the project. The reason for the identification of these companies is that they are the companies that prepare the 5G infrastructure and have large base station networks in Turkey. The cost of the project depends on the software used, but varies hardware. Although the cost of the project is uncertain, it was determined as $ 500.The project, first called in the Turkish market, aims to spread to Europe from there to the world.

1. **Communicty Feedback**

The whole project was shared on the Digilent Project Forum and was displayed by 152 people on 3 May and added to the favorites by 4 people. About the phase lock loop part of the project, the Reddit issue was opened and two positive feedbacks was obtained. A lot of suggestions and ideas were obtained on the subject of the project's unresolved wireless communication in Reddit. The project was presented to Çankaya university teaching staff and contributed to the further development by noting the deficiencies. The process of disseminating the results of the project as an article continues

1. **Reference**
2. [**https://ijarcce.com/wp-content/uploads/2015/01/IJARCCE3J-a-sri-RaghavaKumari-Implementation-of-Embedded-1.pdf**](https://ijarcce.com/wp-content/uploads/2015/01/IJARCCE3J-a-sri-RaghavaKumari-Implementation-of-Embedded-1.pdf)
3. [**https://www.intel.com/content/www/us/en/products/programmable/topics.html**](https://www.intel.com/content/www/us/en/products/programmable/topics.html)
4. [**https://www.rs-online.com/designspark/10-things-you-can-do-with-software-defined-radio**](https://www.rs-online.com/designspark/10-things-you-can-do-with-software-defined-radio)
5. [**https://www.edgefx.in/fpga-architecture-applications/**](https://www.edgefx.in/fpga-architecture-applications/)
6. [**https://www.researchgate.net/publication/269327594\_FPGAs\_in\_software\_defined\_radio**](https://www.researchgate.net/publication/269327594_FPGAs_in_software_defined_radio)
7. [**https://www.researchgate.net/publication/224321693\_GSM-based\_remote\_sensing\_and\_control\_system\_using\_FPGA**](https://www.researchgate.net/publication/224321693_GSM-based_remote_sensing_and_control_system_using_FPGA)
8. [**http://pallen.ece.gatech.edu/Academic/ECE\_6440/Summer\_2003/L070-DPLL(2UP).pdf**](http://pallen.ece.gatech.edu/Academic/ECE_6440/Summer_2003/L070-DPLL(2UP).pdf)
9. [**http://pallen.ece.gatech.edu/Academic/ECE\_6440/Summer\_2003/L080-ADPLL(2UP).pdf**](http://pallen.ece.gatech.edu/Academic/ECE_6440/Summer_2003/L080-ADPLL(2UP).pdf)

|  |
| --- |
| 8. SRRC Implementation Testing Code & Result on Matlab   The output of the Square Root Raised Cosine(SRRC) Matlab is given in the figure. The purpose of our observation of SRRC in MATLAB is to test. Thus, it was observed that the output of SRRC MATLAB is the same as the output of VHDL. SRRC MATLAB code is the time-dependent formula for SRRC. This helped minimize the inter-symbols interference. |
| clc;clear all;  Tc=5;  a=0.02;  t=-12.7\*1.5:0.15:12.8\*1.5;  % t=-38.22\*Tc:2.56\*4\*Tc:38.22\*Tc;  c1=cos(2\*pi\*t/Tc);  c2=sin(2\*pi\*(t)/Tc);  p=zeros(1,length(t));  for i=1:1:length(t)  if t(i)==0  p(i)= (1-a)+4\*a/pi;  else if t(i)==1/(4\*a) || t(i)==-1/(4\*a)  p(i)=a/sqrt(2)\*((1+2/pi)\*sin(pi/(4\*a))+(1-2/pi)\*cos(pi/(4\*a)));  else  p(i) = (sin(pi\*t(i)\*(1-a))+4\*a\*t(i).\*cos(pi\*t(i)\*(1+a)))./(pi\*t(i).\*(1-(4\*a\*t(i)).^2));  end  end  end  figure(1)  plot(t,p)  hold on  plot(t,p.\*c1)  title('Square-Root Raised Cosine Filter Filling with Cosine Signal')  ylabel('Amplitude')  xlabel('Time')  grid on |
| Figure 15. SRRC Filter and SRRCxCosine Multiplication |
| 9. Transmitter Implementation Testing Code & Result on Matlab    |  |  | | --- | --- | | I/O Transmitter of MATLAB | | | Pin Name | Description | | a | Roll-off factor of SRRC | | Tc | Period of input signal | | t | Time interval of input signal | | c1 | Cosine signal | | c2 | Sinus signal | | data | Incoming data | | y | Cosine and sinus multiplication with SRRC | | y1 | Quantized state of ‘00’ | | y2 | Quantized state of ‘01’ | | y3 | Quantized state of ‘10’ | | y4 | Quantized state of ‘11’ | | yn | Quantized data in vector form |     The MATLAB code in the transmitter is given in the following figure. If you look at the transmitter block diagram, you can see that incoming data must be multiplied by SRRC and cosine or sinus because this code was continued with SRRC MATLAB code. Detailed description of the data is explained below with pictures. |
| clear all; clc;  Tc=10;  t=-4.5\*Tc:1e-2\*Tc:4.5\*Tc;  b=0.22;  SRRC=(sin(pi\*(t/Tc)\*(1-b))+4\*b\*(t/Tc).\*cos(pi\*(t/Tc)\*(1+b)))./(pi\*(t/Tc).\*(1-((4\*b\*(t/Tc)).^2)));  c1=cos(2\*pi\*t/Tc); c2=sin(2\*pi\*(t)/Tc);data=[1 1 0 0 1 0 0 1]; vec=[];  if data(1:2)== [1 1]  y=SRRC.\*c1+SRRC.\*c2;  elseif data(1:2)== [0 1]  y=-SRRC.\*c1+SRRC.\*c2;  elseif data(1:2)== [1 0]  y=SRRC.\*c1-SRRC.\*c2;  else  y=-SRRC.\*c1-SRRC.\*c2;  end  vec=[vec y];  if data(3:4)== [1 1]  y=SRRC.\*c1+SRRC.\*c2;  elseif data(3:4)== [0 1]  y=-SRRC.\*c1+SRRC.\*c2;  elseif data(3:4)== [1 0]  y=SRRC.\*c1-SRRC.\*c2;  else  y=-SRRC.\*c1-SRRC.\*c2;  end  vec=[vec y];  if data(5:6)== [1 1]  y=SRRC.\*c1+SRRC.\*c2;  elseif data(5:6)== [0 1]  y=-SRRC.\*c1+SRRC.\*c2;  elseif data(5:6)== [1 0]  y=SRRC.\*c1-SRRC.\*c2;  else  y=-SRRC.\*c1-SRRC.\*c2;  end  vec=[vec y];  if data(7:8)== [1 1]  y=SRRC.\*c1+SRRC.\*c2;  elseif data(7:8)== [0 1]  y=-SRRC.\*c1+SRRC.\*c2;  elseif data(7:8)== [1 0]  y=SRRC.\*c1-SRRC.\*c2;  else  y=-SRRC.\*c1-SRRC.\*c2;  end  vec=[vec y];  y1=-SRRC.\*c1-SRRC.\*c2;  y2=-SRRC.\*c1+SRRC.\*c2;  y3=SRRC.\*c1-SRRC.\*c2;  y4=SRRC.\*c1+SRRC.\*c2;  figure(1)  plot(vec)  title('Transmitter Output When Data=[1 1 0 0 1 0 0 1]')  ylabel('Amplitude')  xlabel('Time')  grid on |
| |  |  | | --- | --- | | 00  figure 16 is also shown in Matlab simulation of 1st condition. If the two bits of incoming data were '00', SRRC was multiplied by the sum of negative cosine and negative sinuses. it was observed that the output of MATLAB was the same as the output of Vivado and thus tested.    Figure 16. Analog Input for "00" | 01  figure 17 is also shown in Matlab simulation of 1st condition. If the two bits of incoming data were '01', SRRC was multiplied by the sum of negative cosine and positive sinuses. it was observed that the output of MATLAB was the same as the output of Vivado and thus tested.    Figure 17. Analog Input for "01" | | 10  figure 18 is also shown in Matlab simulation of 1st condition. If the two bits of incoming data were '10', SRRC was multiplied by the sum of positive cosine and negative sinuses. it was observed that the output of MATLAB was the same as the output of Vivado and thus tested.  Figure 18. Analog Input for "10" | 11  figure 19 is also shown in Matlab simulation of 1st condition. If the two bits of incoming data were '11', SRRC was multiplied by the sum of positive cosine and positive sinuses. it was observed that the output of MATLAB was the same as the output of Vivado and thus tested.  Figure 19. Analog Input for "11" | |
| 10. Receiver Implementation Testing Code & Result on Matlab  |  |  | | --- | --- | | I/O Receiver of MATLAB | | | Pin Name | Description | | a | Roll-off factor of SRRC | | Tc | Period of input signal | | t | Time interval of input signal | | c1 | First carrier of cosine signal | | c2 | First carrier of sinus signal | | cr1 | Second carrier of cosine signal | | cr2 | Second carrier of cosine signal | | y1 | Quantized state of ‘00’ | | y2 | Quantized state of ‘01’ | | y3 | Quantized state of ‘10’ | | y4 | Quantized state of ‘11’ | | yr11 | Quantized data in vector form |   Figure 20 is also shown in Matlab receiver detection. In the MATLAB section of the project, the code of the receiving party has been written. In fact, it works with the opposite logic of the transmitter. First, data from the transmitter comes to the receiver with the help of the antenna. Then, the SRRC code previously obtained for SRRC is used in this code. After converting the signal from the transmitter to digital signal with A/D converter, it splits the data into 2 bits and goes to the decision stage. Then the data is multiplied again by cosine or sinus. The data SRRC filter, which is multiplied by cosine or sine, is used as a filter that matches the demodulator, because the signal is free of noise. Phase Lock Loop (PLL), which is in the receiver's block diagram, is not written MATLAB code. This is because it is easier to test Vivado because it works digital. The last process is the detection process. After finding which of the incoming data is equal, these data are combined in vector form and observed. Finally, it was tested whether the data sent from the receiver went to the transmitter. |
| clc;clear all;  Tc=1/25.5;  a=0.22;  t=-38.22\*Tc:2.56\*4\*Tc:38.22\*Tc;  p=zeros(1,length(t));  for i=1:1:length(t)  if t(i)==0  p(i)= (1-a)+4\*a/pi;  else if t(i)==1/(4\*a) || t(i)==-1/(4\*a)  p(i)=a/sqrt(2)\*((1+2/pi)\*sin(pi/(4\*a))+(1-2/pi)\*cos(pi/(4\*a)));  else  p(i) = (sin(pi\*t(i)\*(1-a))+4\*a\*t(i).\*cos(pi\*t(i)\*(1+a)))./(pi\*t(i).\*(1-(4\*a\*t(i)).^2));  end  end  end  %carriers1  c1=(cos(2\*pi\*t/Tc)); c2=(sin(2\*pi\*(t)/Tc));  % carriers2  cr1=(cos(2\*pi\*t/Tc));cr2=(sin(2\*pi\*(t)/Tc));  a=1;b=1;  y1=(a\*p.\*c1+b\*p.\*c2);y2=(-a\*p.\*c1-b\*p.\*c2);  y3=(a\*p.\*c1-b\*p.\*c2);y4=(-a\*p.\*c1+b\*p.\*c2);  yr11=(cr1.\*y1);yr12=(cr2.\*y1);  yr21=(cr1.\*y2);yr22=(cr2.\*y2);  yr31=(cr1.\*y3);yr32=(cr2.\*y3);  yr41=(cr1.\*y4);yr42=(cr2.\*y4);  k11=yr11.\*p; k12=yr12.\*p;k21=yr21.\*p;k22=yr22.\*p;k31=yr31.\*p;  k32=yr32.\*p;k41=yr41.\*p;k42=yr42.\*p;  z=0; y=0; f=0; s=0; u=0;l=0; q=0; x=0;  for w=1:1:length(t)  z=k11(w)+z; y=k12(w)+y;f=k21(w)+f;s=k22(w)+s;u=k31(w)+u;l=k32(w)+l;  q=k41(w)+q;x=k42(w)+x;  end  vecd=[z y f s u l q x] ;  rec=zeros(1,length(vecd));  for que=1:1:length(vecd)  if vecd(que)>=0  rec(que)=1;  elseif vecd(que)<=0  rec(que)=0;  end  end  subplot(2,1,1)  stem(vecd)  axis([-1 9 -2 2])  title('QAM Coefficients That Received')  ylabel('QAM Coefficients')  xlabel('Data Vectors Order')  subplot(2,1,2)  stem(rec)  ylabel('Data Has Been Sent')  xlabel('Sended Bit Vector')  axis([-1 9 -2 2])  title('Initial Data Vectors That Detected') |
| *Figure 20.Detection Results of Receiver* |
|  |
| 11. SRRC Implementation Code & Result on ISE and Oscilloscopelibrary ieee;use ieee.std\_logic\_1164.all;use ieee.std\_logic\_arith.all;entity bitirme isport( clk\_in: in std\_logic;clk\_out\_original, clk\_out\_divided : inout std\_logic;P:out std\_logic\_vector(7 downto 0));end entity;architecture behavioral of bitirme issignal num:integer range 0 to 255;signal indx:natural range 1 to 256;signal count: natural range 0 to 5000000;signal temp\_clk\_out: std\_logic:='0';type int\_vector is array(natural range<>)of integer range 0 to 255;signal data\_vector:int\_vector(1 to 256);begindata\_vector<=(120,120,121,121,121,122,122,122,121,121,121,120,120,120,120,121,121,122,122,122,122,122,121,120,120,120,120,120,121,121,122,122,123,122,122,121,120,120,119,119,120,120,121,122,123,123,123,122,121,120,119,119,119,119,120,121,123,122,123,124,123,121,120,119,118,118,118,120,121,123,124,125,125,124,122,120,118,117,116,117,119,121,123,125,126,126,125,123,120,117,115,114,115,117,121,124,127,129,130,127,124,119,115,111,110,110,114,119,126,132,136,137,135,129,119,109,100,94,94,100,113,133,158,185,211,232,246,251,246,232,211,185,158,133,113,100,92,94,100,109,119,129,135,137,136,132,126,119,114,110,111,111,115,120,124,128,130,129,127,124,121,117,115,114,115,117,120,123,125,126,127,125,123,121,119,117,116,117,118,120,122,124,125,125,124,123,121,120,118,118,118,119,120,121,123,124,124,123,123,121,120,119,119,119,119,120,121,122,123,123,123,122,121,120,120,119,119,120,120,121,122,122,123,122,122,121,121,120,120,120,120,120,121,122,122,122,122,122,121,121,120,120,120,120,121,121,121,122,122,122,121,121,121,120,120,120);clk\_out\_original<=clk\_in;process(clk\_in)beginif(rising\_edge(clk\_in))thencount<=count+1;if(count=5000000)thentemp\_clk\_out<=not temp\_clk\_out;count<=0;end if;end if;end process;clk\_out\_divided<=temp\_clk\_out;process(clk\_out\_divided)beginif(rising\_edge(clk\_out\_divided))thenindx<=indx+1;if(indx=256)thennum<=data\_vector(indx);P<=conv\_std\_logic\_vector(num,8);end if;end if;end process;end architecture; C:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\WhatsApp Image 2019-05-04 at 13.42.29.jpeg |
| *Figure 21.SRRC Signal in Oscilloscope* |
| 12. Transmitter Implementation Code & Result on ISE and Oscilloscopelibrary ieee;use ieee.std\_logic\_1164.all;use ieee.std\_logic\_arith.all;entity transmitter isport( clk\_in: in std\_logic;output:out std\_logic\_vector(7 downto 0) );end entity;architecture logic\_flow of transmitter istype matrix is array(0 to 255) of integer range 0 to 255;signal y: matrix;signal bit\_vec:std\_logic\_vector(7 downto 0):="01011010";signal count: natural range 1 to 50000;signal count2: natural range 0 to 255;signal temp\_clk\_out: std\_logic:='0';signal temp\_clk\_out\_in: std\_logic:='0';signal i:integer:=0;signal q:integer;signal z:std\_logic\_vector(7 downto 0);signal CON:std\_logic\_vector(1 downto 0);signal count3:natural range 1 to 256;signal clk\_out\_divided: std\_logic;signal clk\_256: std\_logic;signal clk\_out\_original : std\_logic;beginz<="01001011";process(z,clk\_256)variable c : integer range 0 to 8 :=0;beginif(rising\_edge(clk\_256))thenif (c < 7) thenCON<=(z(c)&z(c+1));c := c+2;elsec:=0;end if;if(CON="00")then--y1y<=(127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,126,126,126,127,127,128,128,128,127,127,127,126,126,126,127,127,127,127,127,127,127,127,127,127,127,127,126,126,125,125,126,128,129,131,131,130,128,124,121,119,119,121,126,130,135,137,136,133,129,125,122,122,125,128,129,125,115,97,73,46,23,6,0,8,28,57,89,119,143,157,160,155,145,133,122,116,114,114,120,124,128,130,130,129,128,127,127, 127,128,128,128,127,126,126,125,126,126,127,128,128,128,128,127,126,126,126,126,127,127,128,128,128,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,126,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127);--y1--y2elsif(CON="01")theny<=(126,126,127,127,126,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,127,126,126,126,126,126,126,127,127,127,127,127,126,126,125,125,126,126,127,128,128,128,127,126,125,125,125,126,127,127,128,127,127,127,127,128,129,130,129,127,124,119,115,113,115,122,133,145,155,160,156,143,119,89,56,28,8,0,6,22,46,72,96,114,125,129,127,124,122,122,124,129,133,136,137,134,130,125,121,119,119,121,124,127,130,131,130,129,127,126,125,125,125,126,127,127,127,127,127,126,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,127,127,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,127,127,127,126,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126);--y2--y3elsif(CON="10")theny<=(127,127,127,126,127,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,127,127,127,127,127,127,126,126,126,126,126,127,127,128,128,127,127,126,125,125,125,126,127,128,128,128,127,126,126,125,126,126,127,126,125,124,123,124,126,129,134,138,140,138,131,120,108,98,93,97,110,134,164,197,225,245,253,247,231,207,181,157,139,128,124,126,129,131,131,129,124,120,117,116,119,123,128,132,134,134,132,129,126,123,122,123,124,126,127,128,128,128,127,126,126,126,126,126,127,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,126,126,126,127,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127);--y4=elsey<=(127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,126,126,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,127,126,127,127,127,128,128,128,127,127,126,126,126,126,127,127,128,128,128,127,127,127,127,127,127,127,127,126,126,126,127,127,128,129,129,128,126,124,123,123,124,126,130,133,135,135,132,128,123,119,116,117,120,124,129,131,131,129,126,124,128,139,157,181,207,231,247,253,245,225,197,165,134,110,97,93,98,108,120,131,138,140,138,134,130,126,124,123,124,125,126,127,126,126,125,126,126,127,128,128,128,127,126,125,125,125,126,127,127,128,128,127,127,126,126,126,126,126,127,127,127,127,127,127,126,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,126,126,126,126,127,127,127,127,127,127,126,126,126,126,126,127,127,127,127,127,126,126,126,126,126,126,127,127,127,127,127,127,126,127,127,127,127);end if;i<=i+2;end if;end Process;clk\_out\_original <= clk\_in;process(clk\_in)beginif (rising\_edge(clk\_in)) thencount <= count + 1;if(count=50000) thentemp\_clk\_out<=not temp\_clk\_out;count<=1;end if;end if;end process;clk\_out\_divided <= temp\_clk\_out;process(clk\_out\_divided)beginif (rising\_edge(clk\_out\_divided)) thencount3 <= count3 + 1;if(count3=256) thentemp\_clk\_out\_in<=not temp\_clk\_out\_in;count3<=1;end if;end if;end process;clk\_256<=temp\_clk\_out\_in;process(clk\_out\_divided)beginif (rising\_edge(clk\_out\_divided)) thenoutput<= conv\_std\_logic\_vector( y(count2),8 );if(count2=255) thencount2<=0;end if;count2 <= count2 + 1;end if;end process;end architecture; |
| C:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\WhatsApp Image 2019-05-04 at 13.50.45.jpegC:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\WhatsApp Image 2019-05-04 at 13.49.54.jpeg |
| *Figure 22.Transmitter Signals in Oscilloscope* |
| 13. PLL Implementation Code & Result on Vivadolibrary IEEE;use IEEE.STD\_LOGIC\_1164.ALL;entity PLL\_Test isport(clk:in std\_logic:='0';Kupper,Kdown:out integer;carry,borrow:buffer std\_logic;C,B,Kclk,XOR\_OUT,IDO,IDO8,tff1:out std\_logic;cyc:out integer);end PLL\_Test;architecture Behavioral of PLL\_Test issignal DNUP8,v1,v2:std\_logic\_vector(0 to 7);signal Kclock,IDclock,Nclock:std\_logic:='0';--signal carry,borrow:std\_logic:='0';signal DNUP:std\_logic;signal M:integer:=16;signal K:integer:=4;signal N:integer:=8;signal count:natural range 1 to M;signal KUP,KDN:natural range 0 to K-1:=0;--Kup ve Kdown sayacı.signal cyclecount:integer:=2;--Clock'un 1 Xor outputunun olduğu sürede 4 kez çalışması için varsignal cyclecount1:integer:=2;--Clock'un 1 Xor outputunun olduğu sürede 4 kez çalışması için varsignal cyclecount2:integer:=0;--Clock'un 1 Xor outputunun olduğu sürede 4 kez çalışması için varsignal IDout:std\_logic;signal Temp\_sig:std\_logic:='1';signal TFF\_sig:std\_logic:='0';signal TFF\_sig1:std\_logic;signal cnt1,cnt2:natural range 0 to 2:=1;signal count2:natural range 0 to 2:=0;signal count3:natural range 1 to 8:=2;signal IDout8:std\_logic:='1';begin--INPUTS--v1<="11011011";--Inputv2<="00000000";--N counter çıkışı--XOR GATE--DNUP8<=v1 xor v2;--Xor\_Outprocess(clk,DNUP8,cyclecount)variable c:integer range 0 to 8:=0;beginif(rising\_edge(clk))thenif(cyclecount2=0)thenif(c<=7)thenDNUP<=DNUP8(c);c:=c+1;if(c>7)thenc:=0;end if;end if;end if;cyclecount2<=cyclecount2+1;if(cyclecount2=K-1)thencyclecount2<=0;end if;end if;end process;----------K COUNTER------------K\_COUNTER:process(clk,DNUP,KUP,KDN)beginif(falling\_edge(clk))then--Sayaç Kısmı--if(DNUP='0')thenif(KUP=K-1)thencnt1<=cnt1+1;if(cnt1=2)thenKUP<=0;cnt1<=2;carry<='0';end if;end if;end if;if(DNUP='1')thenif(KDN=K-1)thencnt2<=cnt2+1;if(cnt2=2)thenKDN<=0;cnt2<=2;borrow<='0';end if;end if;end if;--Carry Borrow aktivasyonu--if(KUP=2)thencarry<='1';end if;if(KDN=2)thenborrow<='1';end if;if(DNUP='0')thenif(KUP<K-1)thenKUP<=KUP+1;KDN<=KDN;end if;elsif(DNUP='1')thenif(KDN<K-1)thenKDN<=KDN+1;KUP<=KUP;end if;end if;end if;end process;----------ID COUNTER--------------For ID clock we use K=2N then IDclk=Kclk----- count2<=count2+1;-- if(count2=2)then-- count2<=2;-- Temp\_sig<=not Temp\_sig;-- end if;--T\_Flip\_Flop:process(clk,carry,borrow,cyclecount,cyclecount1)beginif(rising\_edge(carry))thencyclecount<=0;Temp\_sig<='0';end if;if(rising\_edge(borrow) and clk'event)thencyclecount1<=0;Temp\_sig<='1';end if;if(rising\_edge(clk))thenif(cyclecount>=0 and cyclecount<1)thenTemp\_sig<='0';elsif(cyclecount=1)thenTemp\_sig<='1';elseif(cyclecount1<=1)thenTemp\_sig<=Temp\_sig;elseTemp\_sig<=not Temp\_sig;end if;end if;cyclecount<=cyclecount+1;if(cyclecount1>=0 and cyclecount1<1)thenTemp\_sig<='1';elsif(cyclecount1=1)thenTemp\_sig<='0';elseif(cyclecount<=0)thenTemp\_sig<=Temp\_sig;elseTemp\_sig<=not Temp\_sig;end if;end if;cyclecount1<=cyclecount1+1;TFF\_sig<=Temp\_sig;end if;end process;IDout<=clk nor TFF\_sig;--N Divider--tff1<=TFF\_sig;IDO<=IDout;IDO8<=IDout8;C<=carry;B<=borrow;Kclk<=clk;Kupper<=KUP;Kdown<=KDN;XOR\_OUT<=DNUP;cyc<=cyclecount;end Behavioral;C:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PLL_Result.pngC:\Users\berka\AppData\Local\Microsoft\Windows\INetCache\Content.Word\WhatsApp Image 2019-05-04 at 14.11.32.jpeg*Figure 22.PLL Textbook Example* |
| *Figure 23.PLL Algorithm Testbench Result* |
| 14. Receiver Implementation Code & Result on Vivadolibrary IEEE;use IEEE.STD\_LOGIC\_1164.ALL;entity receiver isPort (bit\_rec:out signed(7 downto 0);Data:in signed(7 downto 0);clk\_in:in std\_logic;SW:in std\_logic);end receiver;architecture Behavioral of receiver issignal count: natural range 1 to 50000;signal temp\_clk\_out:std\_logic:='0';type states\_rec is(b0,b1,b2,b3,b4,b5,b6,b7,stop,idle);signal pstates,nstates:states\_rec;signal RX\_Done:std\_logic;signal RX:std\_logic;signal count2: natural range 0 to 255;signal clk\_out\_divided,clk\_256:std\_logic;signal temp\_clk\_out\_in:std\_logic:='0';signal count3:natural range 1 to 256;type matrix is array(0 to 255) of signed(7 downto 0);signal y: matrix;signal CON:std\_logic\_vector(1 downto 0);beginprocess(clk\_in)beginif (rising\_edge(clk\_in)) thencount <= count + 1;if(count=50000) thentemp\_clk\_out<=not temp\_clk\_out;count<=1;end if;end if;end process;clk\_out\_divided <= temp\_clk\_out;process(clk\_out\_divided)beginif (rising\_edge(clk\_out\_divided)) thencount2 <= count2 + 1;if(count2=256) thentemp\_clk\_out\_in<=not temp\_clk\_out\_in;count2<=1;end if;end if;end process;clk\_256<=temp\_clk\_out\_in;process(clk\_out\_divided)beginif(rising\_edge(clk\_out\_divided))theny(count3)<=(Data);if(count3=255)thencount3<=0;end if;bit\_rec<=y(count3);count3<=count3+1;end if;end process;end Behavioral; |