UNIVERSITY OF HERTFORDSHIRE

Faculty of Engineering & Information Sciences

MSc DATA COMMUNICATIONS AND NETWORKS

Project Report

QPSK MODULATION AND DEMODULATION ON FPGA

LE, YANG

SEPTEMBER 2006

Abstract

The QPSK modulation and demodulation techniques have been widely used in communication systems. This report will introduced the QPSK modulator and demodulator design methods and processes on FPGA. Lots of theories will be studied before the introduction of the project design by the researcher. The MATLAB will be introduced as well to help the researcher finished the VHDL part design.

The simulation results of the modulator and demodulator will be discussed. In addition, the MATLAB part results will compare to the VHDL results in order to check the design whether is correct or not.

Moreover, some advices will be mentioned for the future work of this project. Some noise reduction and error correction techniques will be studied in the further work.

Acknowledgment

The researcher has finished the project design and report during this summer. Many people have give help to the researcher. In here, the researcher would like to thanks all these very kind and helpful people.

Firstly, the researcher would like to thanks the supervisor of this project - Mr. Jun Xu. He guided the researcher to finish this project. Although, sometimes he is busy, he still gives lots of suggestions and ideas to the researcher about the project designing and report writing. This is the main reason that the researcher can finished the project in just 3 months.

Secondly, the researcher would like to thanks Mr. Baochun Hou. He is the tutor of the FPGA course. He also gives lots of theory of the modulation and demodulation techniques and hardware design methods to the researcher.

Finally, the researcher would like to thanks his family and friends that gave him support on study and live. The researcher can not successfully complete this project and report without them.

Declaration

I certify that the work submitted is my own and that any material derived or quoted from the published or unpublished work of other persons has been duly acknowledged

Student Full Name:……..

Student Registration Number:…….

Signed:……..

Date:……..

CONTENTS

List of figures

List of tables

Chapter 1 Introduction:

1.1 Data communications

Nowadays, wireless communications becomes a kind of important technology. For example, mobile phones, wireless networks, satellite communication and etc. However, how the data is being transmitted and received from sources to destinations? In this report, the researcher will undertake a research on a kind of technique that used for data transmitting and receiving. On top of that, the researcher will design a modulator and a demodulator on FPGA.

Generally, there are two types of signals, which are analog signal and digital signal. Data Communication is a process that a transmission of data from one point to another over communication channels [13]. However, the data cannot be transmitted directly. There is a technique called modulation can change the data to a transmittable signal. Then this modulated signal can be transmitted and received.

Digital communications transmit the digital data. Meaning, all data is consisting of '1' and'0'. This kind of transmitting method gives much more advantages than analog transmitting. For example, there are many continuous data like voice, music, pictures and video. If these kinds of data digitalized before transmitted, they will be high percentage of accuracy. In addition, the optical fiber can reduce the transmitting cost of high bit rates over long distance. It could be suggested that, modern communication will be improved very fast by digitally transmitting data before they are transmitted. Higher and higher bit rates data can be transmitted over the transmitting medium [15]. Figure 1.1 shows the elements of a digital communication system.

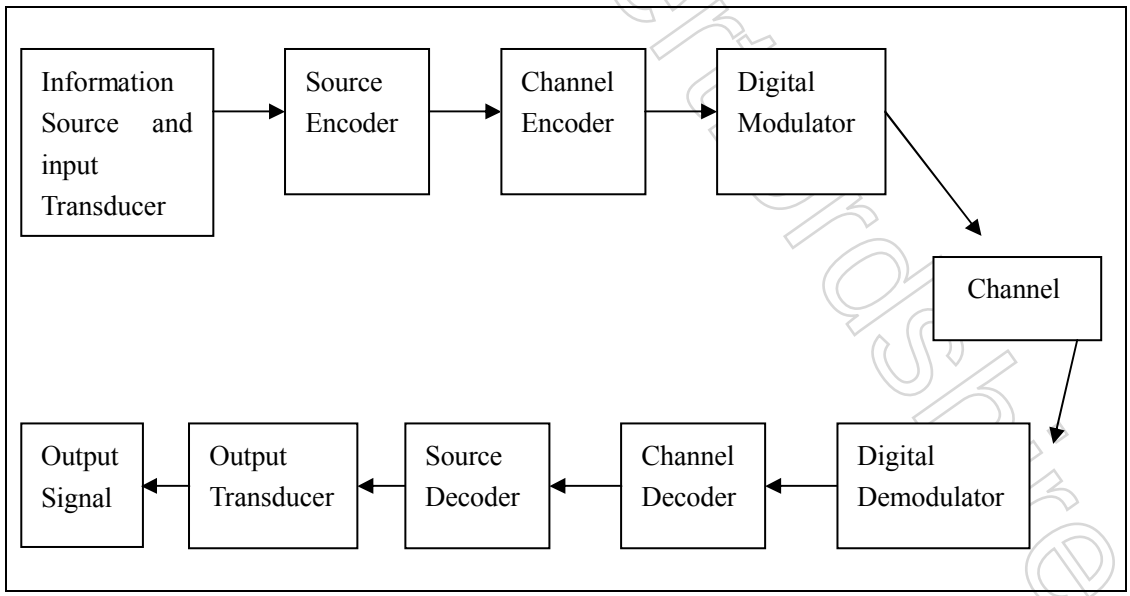

Figure 1.1: Basic elements of a digital communication system [15]

The source of information could be analog data; it can be converted into a sequence of binary data through the source encoding process. Then, the digital data can be modulated in digital modulator and be transmitted over the channel. In the receiver, the modulated signal can be demodulated in digital demodulator and converted back to original input data in source decoder. Moreover, the channel encoder and decoder are controlled manner; some binary information can be used to reduce the effects of noise and interference in the receiver. It can reduce the error data and some unwanted signal to improve the reliability of the received data in real communication systems [15].

1.2 Signal modulation

Analog modulation

Because of the signal can not be transmitted directly over the channel, the techniques which called signal modulation and demodulation need to be introduced. Practically, data signal needs a process that can translate it into a transmittable signal. This technique is called modulation. In general, modulation is a process to modify a signal's characteristics with another signal [4]. The signal which has been modified is called carrier signal. The signal doing the modification is called information signal [4]. Thus, the information signal need to be modulated with carrier signal first, and then it can be transmitted. Why the information signal requires modulation before transmitted? It is because of information signals need to be matched channels and it can provide some advantages for efficient communication (sharing the communication resources) and make antennas more efficient for radio communication systems [2]. Figure 1.2 shows the structure of information signal modulation.

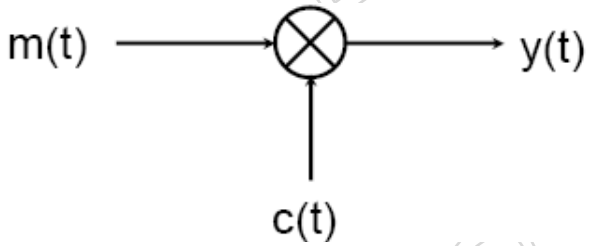

Figure 1.2: Structure of signal modulation [2].

Where,
M (t): information signal.
C (t): carrier signal.
Y (t): modulated signal.

In the figure, the information signal multiplied with the carrier signal. $y_{(t)} = m_{(t)} \times c_{(t)}$ (1.1).

In the previous years, the analog modulation has been widely used in data communication systems. There are many scheme of analog modulation. For example, amplitude modulation (AM), single-sideband modulation (SSB), double-sideband modulation (DSB), frequency modulation (FM), phase modulation (PM) and etc. Mathematically, the Y (t) can be represented like this: $M(t) = A_m \cos(2\pi f_m t)$, $c(t) = A_c \cos(2\pi f_c t)$, $y(t) = m(t) \times c(t)$ (1.2), calculate

the equation $y(t) = A_m A_c \cos(2\pi f_c t)\cos(2\pi f_m t)$ (1.3). Figure 1.3 shows the DSB in time domain.

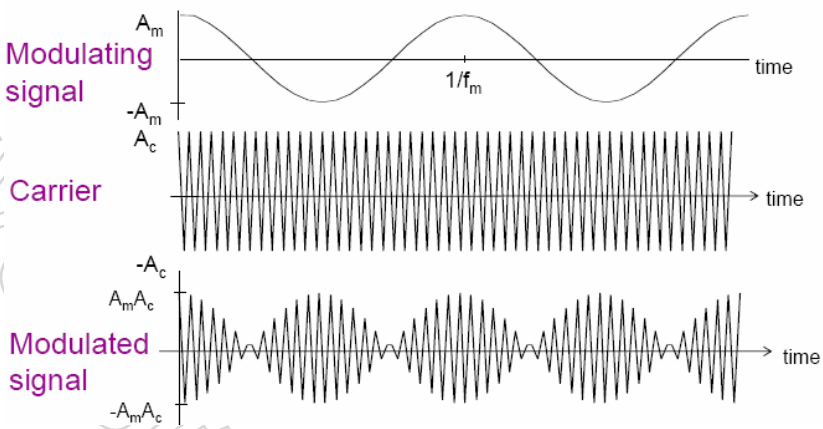

Figure 1.3: DSB in time domain [2].

Figure 1.4 shows the DSB in frequency domain

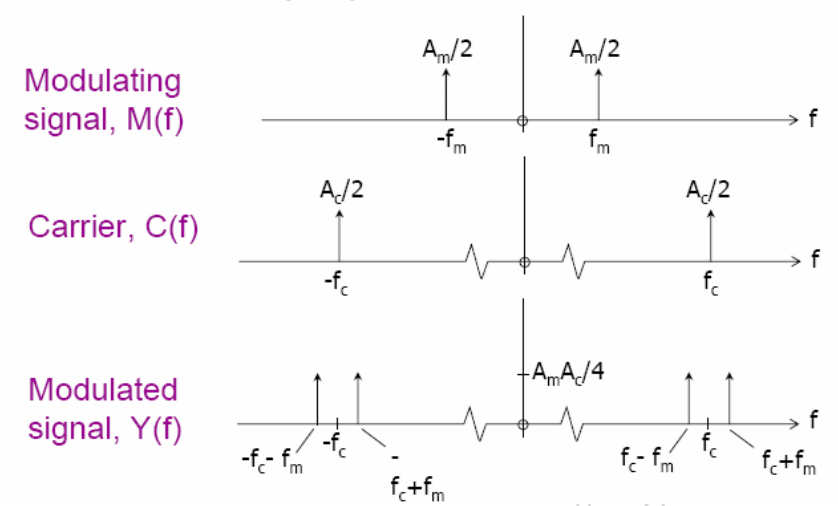

Figure 1.4: DSB in frequency domain [2].

In figure 1.3, we can observe that, the carrier signal has been changed according to the modulating signal during modulation process. However, the modulating and carrier signal are both analog signal. That is not an easy task to demodulate the transmitted signal because there is some noise and interference during the transmitting. Thus, the modulated signal could be very difficult to recognize in the demodulator. In conclusion, the analog modulation and demodulation need a very good noise reduction system for demodulating back the modulated signal. That will be much costly than the digital modulation and demodulation.

Digital modulation

The digital modulation is the process of a digital message to be modulated into a continuous signal that can be transmitted in communication channels. However, just modulation is not enough for the signal communication. It is mainly because of the destination needs a device to convert back from the carrier signal to the information data which is called demodulation. Digital demodulation is the process of converting a received signals that consisting of a signal component and a noise component into a digital information data [14].

In general, there are four types of digital modulations, which are Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), Phase Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM). They are changed amplitude, frequency and phase of carrier signal for ASK, FSK, PSK. The QAM is a combination of ASK and PSK. In this report, the researcher will concentrate on a kind of PSK that is called QPSK (Quadrature PSK). Figure 1.5 shows ASK, FSK and PSK in time domain.

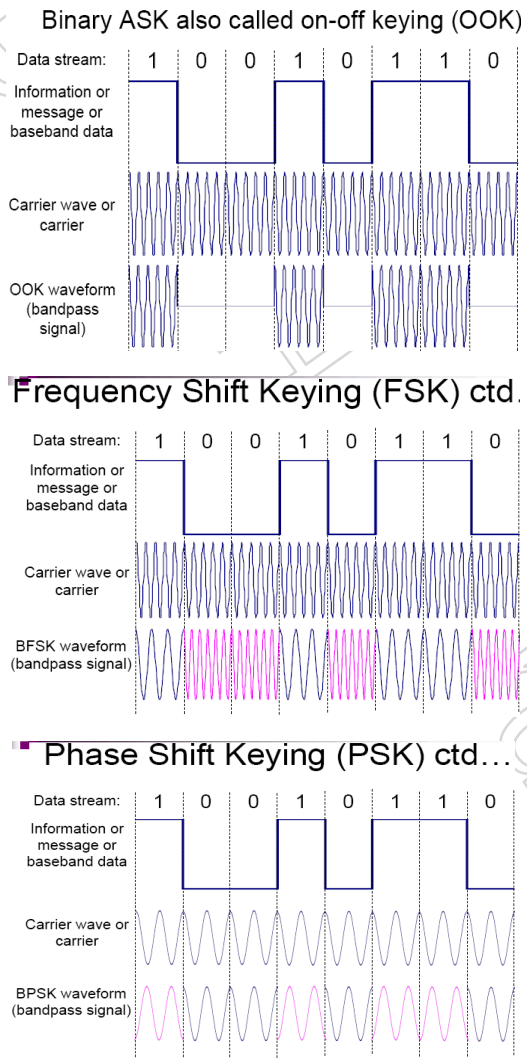

Figure 1.5: ASK, FSK, PSK in time domain [2].

1.3 Organization of report

This report will introduce the QPSK modulator and demodulator on FPGA. There are 8 chapters in this report.

Chapter 1: This chapter is the introductory chapter. This chapter has introduced the structure of this final project. Furthermore, the background of QPSK modulation and demodulation has been mentioned. For example, digital communications, signal modulation and digital modulation and demodulation methods. On top of that, the aims and objectives of this research will be introduced in order to give a clear picture to the reader what the researcher is going to undertake in this project. In addition, the research questions will be asked before the literature review and project design. Moreover, the project management and planning will be mentioned.

Chapter 2: This chapter is the review of the literature regarding QPSK modulation and demodulation. In this chapter, the researcher is mainly discussing the theory of QPSK and FPGA. How can the modulator and demodulator work? How to design the QPSK modulator and demodulator on FPGA?

Chapter 3: This chapter is all about the explanations regarding the research methods, which was undertaken by the researcher to produce project. On the other hand, all the details of the data collection and result analysis methods will be stated. This chapter will enable the readers to understand how the researcher collects the data and analyse the results of modulator and demodulator that the researcher designed. In addition, the explanation of the test method will be given, to check the design whether is correct or not.

Chapter 4: This chapter is about to describe the QPSK modulator and demodulator design in MATLAB part. Some design flow charts will be introduced and explained.

Chapter 5: This chapter is about to describe the QPSK modulator and demodulator design in VHDL part. Some design flow charts will be introduced and explained.

Chapter 6: This chapter is to collect the results of MATLAB and VHDL part. Compare these results and analyse them for the final check of the design. On top of that, many results which collected from the modulator and demodulator will be discussed and will proof the reliability of the design.

Chapter 7: This chapter is to conclude the project and advice will be given by researcher for the further works.

1.4 Aims and objectives:

The aims and objectives in this project are:

i. To introduce the background and theory on QPSK.
ii. To discuss about the process of modulation and demodulation.
iii. To design a QPSK modulator and demodulator on FPGA.
iv. To examine the results of designed modulator and demodulator.

The objective of this project is to design a QPSK modulator and demodulator on FPGA. QPSK modulation technique has widely been used on wireless digital communication systems. For example, mobile phones system, wireless networks and satellite communications [13]. The researcher will study VHDL to design this project because the FPGA is cheap and fast for logic design. At the end of this project, there are some binary inputs data will input into the QPSK modulator in order to obtain the outputs. Then, input these outputs into the QPSK demodulator, the outputs of demodulator will expect the same as the inputs of QPSK modulator. Meaning, the modulator and demodulator are worked correctly. In addition, a MATLAB function of modulator and demodulator will input same data with VHDL modulator and demodulator to compare the results whether they are same or different. Figure 1.6 shows the objective of results check.

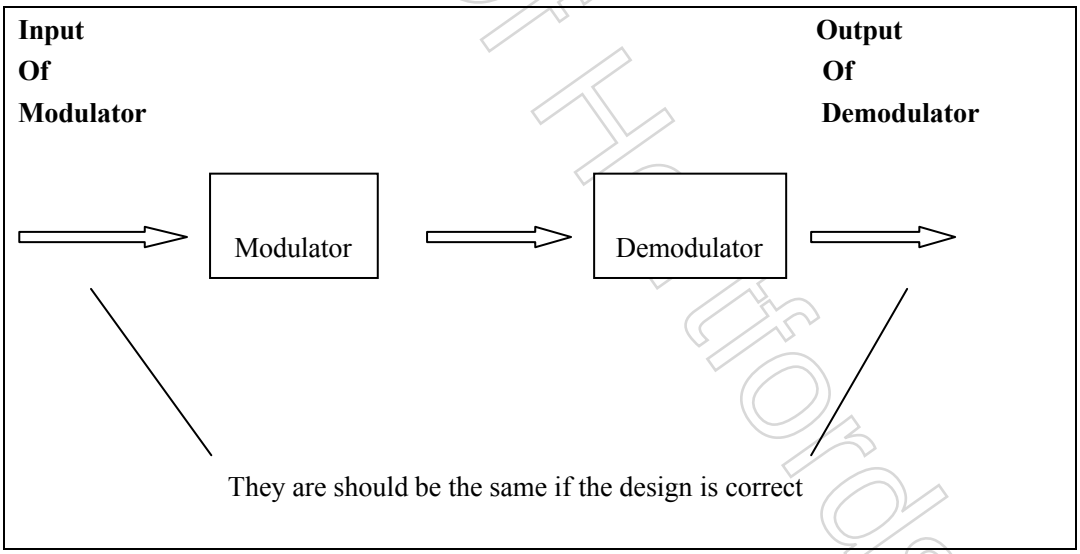

Figure 1.6: Results check diagram

1.5 Research questions

Q1: Why the QPSK widely used in wireless communications?

Q2: How the QPSK modulator and demodulator work?

Q3: Why the researcher uses VHDL to design the QPSK modulator and demodulator?

Q4: How to describe the QPSK modulator and demodulator in VHDL?

Q5: How to solve the QPSK demodulation with noise?

Due to the above research questions, the researcher has to undertake this project which the project title is 'QPSK modulator and demodulator on FPGA'. The following chapters will answer and solve the above questions.

1.6 Project management and planning

1.6.1 Feasibility

This project can be designed on the MATLAB and ISE (Integrated Software Environment). All designs will be accomplished by using the mentioned software. In the other words, this project has a very good feasibility for the researcher. The supervisor and researcher have decided to elide the synthesis process on hardware board. All designs can be done on a computer with Microsoft operating system and some software that required.

1.6.2 Time management

This project has started from June 2006 to September 2006 by the researcher. A Gantt chart (Appendix 8) was created before the project started. Although some steps were taking longer time than expected, however, this project has still finished on time (3rd September 2006).

1.6.3 Project's costing

This project is mainly time consuming. The researcher has done many researches and read lots of books and journals. All the software (MATLAB, ISE 8.1) that this project required was provided by supervisor. Discuss with supervisor is necessary for completing the project.

1.6.4 Logbook

The Logbook has been updated every time through online journal when meeting with supervisor. Logbook provides the researcher a clearer picture by refer the records regarding this project. Through the records and supervisor's recommendations and suggestions, researcher can improve this project.

Chapter 2 Literature Review:

2.1 Digital communication

In this project, the QPSK modulator and demodulator will be designed on FPGA. Due to this purpose, some necessary knowledge needs to be reviewed. In reality, many signals used in modern communication system is digital. It could be argued that, digital communication will becomes more convenient and more popular used as a communication technology. There are many advantages by using this kind of digital communication. First, the digital communication system can transmit binary data that used in computer programs. Due to this reason, the received data can be used in computer directly. Secondly, the digital communication can transmit digital signals and analog signals as well. It can be used to improve the quality of transmission and reduce the distortion and signal-to-noise ratio. In addition, some voice and video signals can be transmitted in wireless communication area with digital communication system. By looking at all the mentioned advantages, it could be suggested that, by using the digital communication, it will not only provide conveniences as the received data can be used in computer directly, and, it will improve the transmission quality and broader the transmission area as well [3].

In digital communication system, the information signal needs to be converted with a carrier signal, and then the modulated signal can be transmitted to channel. Why do we need this process before the signal be transmitted? It is because of the electromagnetic wave being through to the spaces for transmitting by the antennas. In addition, the antennas aperture dimensions need at least as large as the wavelength of the signal that be transmitted. For example, the wavelength is $\lambda = \frac{c}{f}$ where c is the speed of the light $3 \times 10^8 \, m/s$. If the sideband frequency *f*= 3000 Hz, $\lambda = 10^5 \, m$. If there is no modulation process, to transmitting the 3000 Hz signal require an 10^5 m antenna, that is very expensive and inconvenience. However, if the transmitting signal has been modulated with a carrier signal that is 30 GHz, the antenna diameter just need 0.5 inch. That is much smaller and cheaper when compare with antenna that with no modulation process. Due to these reasons, the modulation process should be done before the signal transmitting.

The receiver received the modulated signal and demodulates it to the information signal. Figure 2.1 shows the model of a digital communication system.

Figure 2.1: Model for a digital communication system [14].

In figure 7, the modulator input is binary information data and the output of modulator is the modulated signal. The modulated signal transmits through the channel to the demodulator. The demodulator demodulates back the information data from the modulated signal. Theoretically, this process is simple and straightforward. However, in practical, when the modulated signal transmitted, the noise may mix together with the modulated signal. Due to this reason, the demodulator will received some signal with noise. Thus, some noise reduction and error detection and correction techniques need to be added after the demodulator in order to let the destination user receives the correct information data.

2.2 PSK

There are many types of modulation techniques have been used in digital communication system. In this project, the researcher will concentrate on the PSK modulation and demodulation. The PSK modulation changes the phase of the carrier signal to represent data. Figure 2.2 shows phase shift of 180^0 as the "0" follows the "1" in time domain.

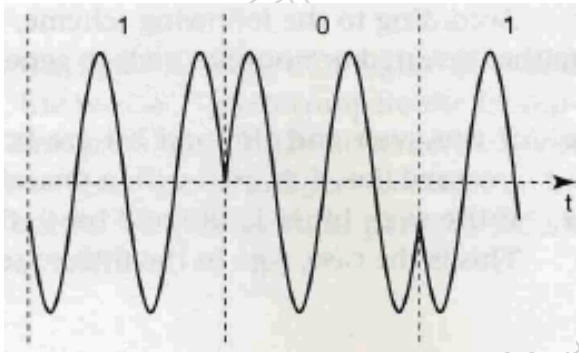

Figure 2.2: Phase shift keying (PSK) [16].

In this figure, the binary "0", "1" are represented the different cosine wave in different phase. This cosine wave is called carrier signal. It can be defined in equation: $g(t) = A_t \cos(2\pi f_t + \varphi_t)$ (2.1) (Where, A_t is the amplitude of carrier signal, f_t is the frequency of the carrier signal and φ_t is the phase of the carrier signal). In addition, figure 2.3 shows the PSK in phase domain.

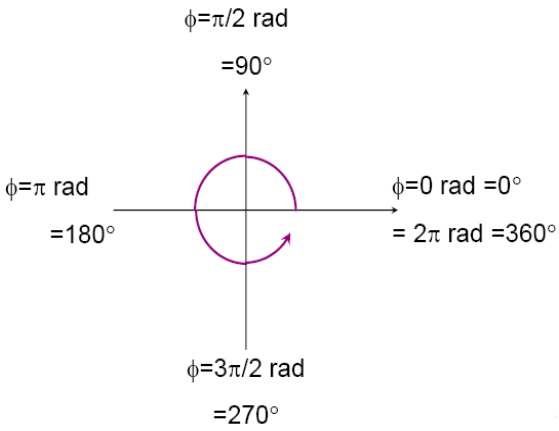

Figure 2.3: PSK in phase domain [2].

2.3 BPSK

There are few advances PSK. The BPSK (binary PSK) is one of them. The BPSK modulating information data shifts the phase of carrier signal to one of two states ("0" and "1"). In phase domain, the "0" of phase represent data "0" and the "π" represent data "1". Figure 2.4 shows the BPSK in phase domain.

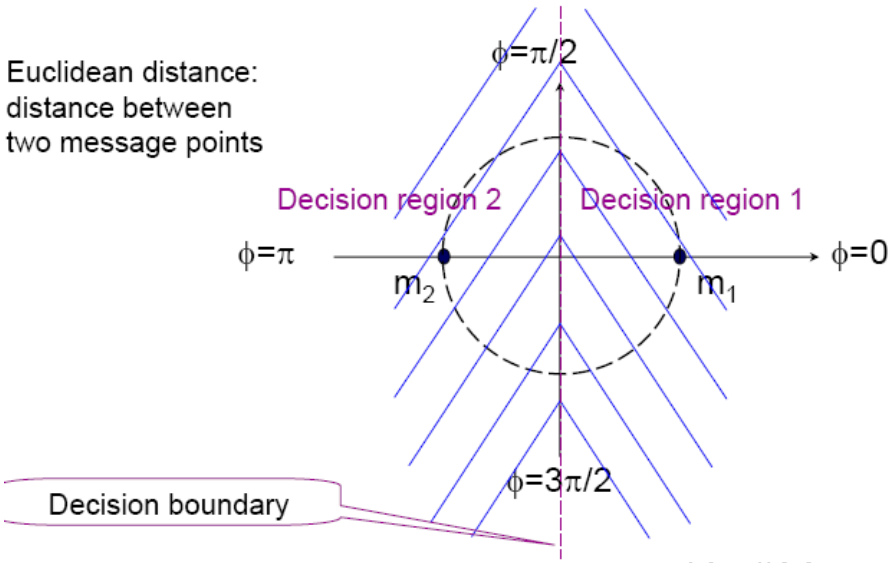

Figure 2.4: BPSK modulation in phase domain [2].

In this figure, all points in decision region 1 represent data m1 and all points in decision region 2 represent data m2. This principle is meant for bit error correction. In reality, during the process of signal transmission, the receiver may receive phases which are not at point m1 or m2, but the received data must be in the decision region 1 or decision region 2. It makes easier for signal demodulation.

2.4 QPSK

2.4.1 QPSK Modulation

Quadrature PSK is another modulation technology. It likes a combination of two BPSK. The QPSK provide advantages of delivering twice bits per symbol than BPSK. It is more efficiency [9]. This modulation can achieve for two bits data into one phase shift keying. Due to this reason, it could be argued that, QPSK reduces the bandwidth compare with the BPSK. The QPSK signal represents the data "11" at 45^0 or $\pi/4$, data "10" at 135^0 or $3\pi/4$, data "00" at 225^0 or $5\pi/4$ and data "01" at 315^0 or $7\pi/4$. Figure 2.5 shows the QPSK in phase domain.

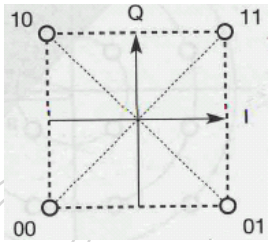

Figure 2.5: QPSK in phase domain [16].

In QPSK, all phase shifts will be related to the carrier signal. In the transmitter, the modulated signals as shown in figure 2.6 are concatenated together. To reconvert data, the receiver needs to compare the received signal with carrier signal and decide which data have been received [12] [16].

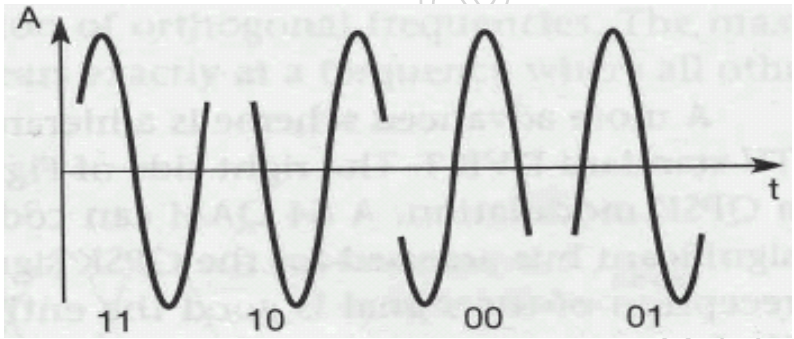

Figure 2.6: QPSK signal in time domain [16].

When the input data is "00" the output pulse is $g_{00}(t) = -\frac{A_c}{\sqrt{2}}\cos(2\pi f_c t) - \frac{A_c}{\sqrt{2}}\sin(2\pi f_c t) = A_c\cos(2\pi f_c t + 135^0)$ (2.2), the pulse of input "01" is $g_{01}(t) = -\frac{A_c}{\sqrt{2}}\cos(2\pi f_c t) + \frac{A_c}{\sqrt{2}}\sin(2\pi f_c t) = A_c\cos(2\pi f_c t + 225^0)$ (2.3), the pulse

of input "10" is $g_{10}(t) = \frac{A_c}{\sqrt{2}}\cos(2\pi f_c t) - \frac{A_c}{\sqrt{2}}\sin(2\pi f_c t) = A_c \cos(2\pi f_c t + 45^0)$ (2.4) and the pulse of input "11" is $g_{11}(t) = \frac{A_c}{\sqrt{2}}\cos(2\pi f_c t) + \frac{A_c}{\sqrt{2}}\sin(2\pi f_c t) = A_c \cos(2\pi f_c t + 315^0)$ (2.5). In the QPSK modulator, the input data needs to be converted from serial to parallel. It separate two serial bits $b_1 b_0$ to two modulators (b_1 available to the upper modulator and b_0 available to the lower modulator). Bits b_1 and b_0 are represented like bipolar voltages, value +1 volt for binary 1 and -1 volt for binary 0. Figure 2.7 shows the block diagram of a QPSK modulator.

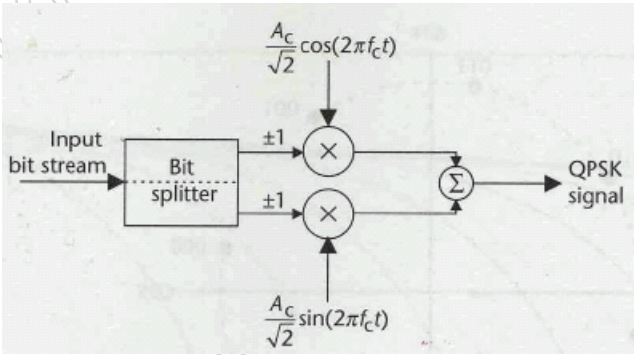

Figure 2.7: Block diagram of a QPSK modulator [13].

On the other hand, the QPSK can be defined two binary PSK (BPSK) combinations, one at a cosine carrier signal and one at orthogonal sine carrier signal [13].

2.4.2 QPSK Demodulation

In QPSK demodulator, there are two matched filters required. They can be used to detect the two bits $b_1 b_0$ transmitted in the channel. As figure 10 shown, the upper filter is matched to the cosine detector for detect data b_1 and the lower filter is matched to the sine detector for detect data. If the cosine component is positive, then b_1=1 and; if negative, b_1=0. Similarly, the sine component is positive for b_0=1 and negative for b_0=0. After the matched filter, there are two decision devices to decide the output b_1=1 when the input is positive and b_1=0 when the input is negative, output b_0=1 when the input is positive and b_0=0 when the input is negative. Before output the information data, the parallel-to-serial converter will be required. It can convert the parallel data to the serial and make the output data is same as the input data of the

QPSK modulator.

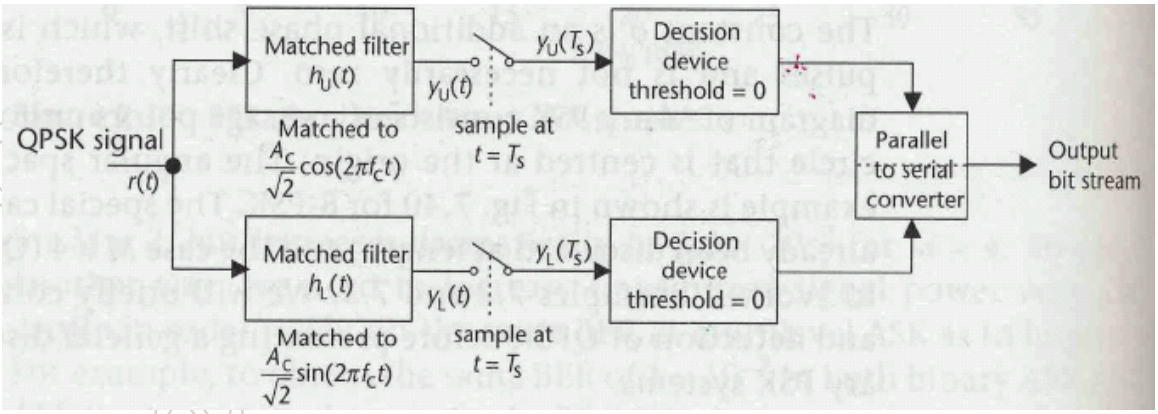

Figure 2.8: Block diagram of QPSK demodulator [13].

By, looking at the above mentioned, it could be noticed that, the demodulator is much more complex than modulator. Basically, in practical, received the signal will not completely the same as the transmitted signal because the bit error may be appeared when the signal transmitting. It would argue that, some error detection and correction devices will be required after the demodulator [13].

2.4.3 QPSK performance

The performance of QPSK system can be measured by SER (symbol error rate) and BER (bit error rate). The SER is the average probability of the received signal being in error, and the BER is the average probability of the received bits being in error [1]. The equation of the SER of QPSK with additive white Gaussian noise (AWGN) only and free transmitting inter symbol interference (ISI) is $P_E = erfc\left[\frac{1}{2}\left(\frac{C}{N}\right)\right]^{1/2}$ (2.6). Where, the $\frac{C}{N}$ is the carrier-noise-power ratio (CNR) and the *erfc* is the complementary error function: $erfc_{(z)} = 1 - erfc_{(z)} = \frac{2}{\sqrt{\pi}}\int_z^{\infty} e^{-x^2} dx$ (2.7) . On the other hand, the bit energy to noise power spectral density ($\frac{E_b}{N_o}$) is a very useful data for evaluate the performance of the modulation systems.

In Nyquist criterion, the relation between $\frac{C}{N}$ and $\frac{E_b}{N_o}$ for QPSK modulation is ($\frac{C}{N}$) $= \log_2 M\left(\frac{E_b}{N_o}\right) = \log_2 4\left(\frac{E_b}{N_o}\right) = 2\left(\frac{E_b}{N_o}\right)$ (2.8). So, the $P_E = erfc\left(\frac{E_b}{N_o}\right)^{1/2}$ (2.9). In these equations, if the M increase, the P_E will increase as well. That means, more bits per symbol will increase the symbol error rate. Although the M increase can make the PSK system more efficiency,

the reliability will decrease; this is a trade-off between reliability and efficiency in M-PSK system. It could be suggested that designer need to be considered this trade-off for different situation when design the PSK modulator and demodulator [1] [2].

2.5 MATLAB

This project has used MATLAB at high level to test the algorithms of design and result check. Due to this reason, the MATLAB need to be introduced first. The MATLAB is Matrix Laboratory. It is a kind of computing environment and programming language. The MATLAB is designed by THE MATHWORKS. It provided lots of functions on variety areas. For example, it has been widely used in numerical, algebra, computer science and engineering fields. For this project, the MATLAB is necessary; because of it provides feasibility in writing a program and fast to execute than VHDL language. In addition, the MATLAB code is easy to read and understand because it is a high level language [6]. It provides convenience to modify the code for designing the QPSK modulator and demodulator in high level. Moreover, the researcher can use the result of the MATLAB part to compare the result in VHDL part and make sure the design of VHDL is correct.

2.6 VHDL，FPGA and ISE

2.6.1 VHDL

VHDL is VHSIC Hardware Description Language. The VHSIC is Very High Speed Integrated Circuit. It is a kind of hardware description language for hardware design, simulation and synthesis. The VHDL is an industry standard of documentation and specification. In addition, the TESTBENCH has been provided for result test. Because of VHDL is hardware design language, it is not the same as C, and MATLAB. The VHDL just define input and output port in hardware. So the TESTBENCH can be used to input the data to the input port and get the result from the output port. That is very helpful for designer to check the accurately of hardware designs [5] [17] [18].

2.6.2 FPGA

FPGA is Field Programmable Gates Array. An FPGA is a logic cells or modules structure that is can be completely controlled by designers. There are many advantages of FPGA. For example, the FPGA is re-programmability, low cost, high performance and it is fast time-to market. In addition, the FPGA is a design platform. Designers can design the systems on chips. Moreover, the FPGA has been widely used in computers, wireless communications and video compression systems [5]. The Xilinx Company provides lots of FPGA series products. This project can be downloaded to the Xilinx FPGA as well. The ISE provide two hardware description languages. First is VHDL and the other is Verilog. These two languages are not very similar. They have a little bit different between grammars, structure and so on. Because of the researcher is more familiar with VHDL language, the VHDL will be used to describe the QPSK modulator and demodulator in ISE environment.

2.6.3 ISE

The ISE is Integrated Software Environment. It is a product of Xilinx Company. The ISE provides the programmable logic design solution for power management, optimal performance, productivity, and cost reduction and support for all Xilinx FPGAs and CPLDs (Complex Programmable Logic Device). In addition, the ISE is easy-to-use, has built-in tools and wizards make I/O assignment, make HDL simulation quickly and intuitively and provide power analysis, timing-driven design closure. It provides good performance, short design time and low cost to logic designers. [11]

Chapter 3 Methodology:

3.1 Code design method

In this project, researcher has designed a QPSK modulator and a demodulator on ISE software environment. Firstly, the researcher has done lots of researches about the digital communication, modulation, demodulation and QPSK modem (modulation and demodulation). The theory has been studied before design. Secondly, the MATLAB has been used. The program flow chart has been draw before writing the code. Then the researcher designed QPSK modulator and demodulator with MATLAB. After that, the MATLAB code has been checked and ensured that is correct, the VHDL code will then be written. The process is similar with the MATLAB part. According to the theory that has been studied, the VHDL code flow chart has been draw before the code writing. Because of the researcher is a VHDL starter, some VHDL program example must be studied first. The researcher has done a lot of exercises in VHDL code design and the supervisor has given the researcher many advices and helps during the project design.

For the flow chart draw, the Microsoft Visio 2003 has been used. This is a chart drawn tool. The MATLAB and VHDL program flow chart have been drawn by using this software.

3.2 Data collection method

The researcher collects the primary data and secondary data as well in this research. The secondary data will be collected for QPSK modulation, demodulation and FPGA in this research. The secondary data will obtain from some professional websites. Of top of that, other resources like books and journals obtained from the UH Learning Resource Centre (LRC) and the studynet's journals and databases as references. In the secondary data, the principles and algorithms of QPSK modulation and demodulation have been studied. In addition, the VHDL language study is required to design the modulator and demodulator. Due to this is a design project, the questionnaire for primary data collection is not necessary. The primary data will obtain from the results of MATLAB function and VHDL simulation. This is to check whether the design has been done correctly.

3.3 Result check method

3.3.1 MATLAB results self check

Because of this is a design project, the reliability of this project needs to be checked. For result check, the researcher generates a group binary data includes 100 binary numbers by using the MATLAB. These 100 numbers is the first group input data. These numbers will be saved into the "input.txt" file. Input these 100 numbers to the MATLAB modulator program, and then the modulator will get 50 numbers modulated signal data which are decimal number. This modulated signal data will then be converted to the binary data and save it into the "output_modulator_matlab.txt" file. Next, the modulated signal data will input to the

demodulator and get the original data from the output of demodulator. This data will be saved to "output_demodulator_matlab.txt" file. The data of input.txt and output_demodulator_matlab.txt will be compared. If they are same, that means that the modulator and demodulator in MATLAB design is correct. According to this method, the researcher has generated 9 more groups input data for check. Each group of input.txt and output_demodulator_matlab.txt should be same as well.

3.3.2 VHDL results self check

After the MATLAB results have been checked, the VHDL code will be written with the similar theories and ideas with MATLAB code on ISE software environment. Because of the ISE is a hardware design tool, the TESTBENCH is necessary for the simulation result check. The TESTBENCH will input the data that generated by MATLAB and saved into the "input.txt" file to the modulator input port and the simulation results of modulator will be written in the "output_modulator_VHDL.txt" file. The output_modulator_VHDL.txt saved the modulated signal data. Next, the modulated signal data will be input to the demodulator through the TESTBENCH as well. The results of the demodulator will be saved into "output_demodulator_VHDL.txt" file and then the output_modulator_VHDL.txt and output_demodulator_VHDL.txt will be compared. They are should be the same if the VHDL design is correct. Using the same method as the MATLAB check, input another 9 more group's data for more accurately result checking.

3.3.3 MATLAB and VHDL results compare check

Because of the 10 groups of input data of MATLAB and VHDL is the same, the modulated signal of MATLAB and VHDL should be the same and demodulated signal should be the same as well. For the purpose of double checking, the output_modulator_matlab.txt and the output_modulator_VHDL.txt should be the same and the output_demodulator_matlab.txt and the output_demodulator_VHDL.txt should be the same as well. If all above mentioned checks is correct, it could be suggested that this project design QPSK modulator and demodulator on FPGA has accomplished and done correctly.

Chapter 4 QPSK modulator and demodulator design on MATLAB:

4.1 QPSK modulator design on MATLAB

Due to design a QPSK modulator and demodulator on FPGA, the researcher needs to study the theory of modulation and demodulation. After that, the high level design tool as MATLAB will be required. The MATLAB code of modulator and demodulator can be designed with simple idea and the code can be read easily as well. The researcher used the theory that studied from books and journals which obtained from library and LRC databases to design the QPSK modulator and demodulator. Because of it is a high level design, it will be easier to change the code and get the result quicker than the low level design such as the FPGA design.

The QPSK modulator in MATLAB is designed with QPSK modulation theory, but not exactly the same. Because of this project is for digital communication, the input and output of the modulator are all in digital. This will make the design more simply and easy to complete. The researcher does not need to consider the analog wave in modulation. All analog waves that are introduced in literature review part will be represented in digital style. For example, when the input data is '00', the researcher does not need to consider what the carrier wave looks like. The modulator just changes the input data '00' into the modulated signal '0001' which is binary data. Meaning, the input data '00' can be represented by '0001' in the channel. When the input data is '01', the modulated signal is '0010'. When the input data is '10', the modulated signal is '0100' and when the input data is '11', the modulated signal is '1000'. Table 4.1 shows the relationship between input data and modulated signal.

Input data	00	01	10	11
Modulated signal	0001	0010	0100	1000

Table 4.1: Relationship between input data and modulated signal

In this table, we can observe that the modulated signal '0001', '0010', '0100', '1000' can be converted into decimal number '1', '2', '4', '8'. So the output of QPSK modulator in MATLAB is '1', '2', '4', '8'. This is easy to read and understand. For further result check with VHDL part, the output of QPSK modulator in MATLAB has been stored in binary number as well.

The QPSK modulator design theory is according to the figure 13. But the structure is not as complex as the figure 13 structure. Figure 4.1 shows the design flow chart of the QPSK modulator in MATLAB.

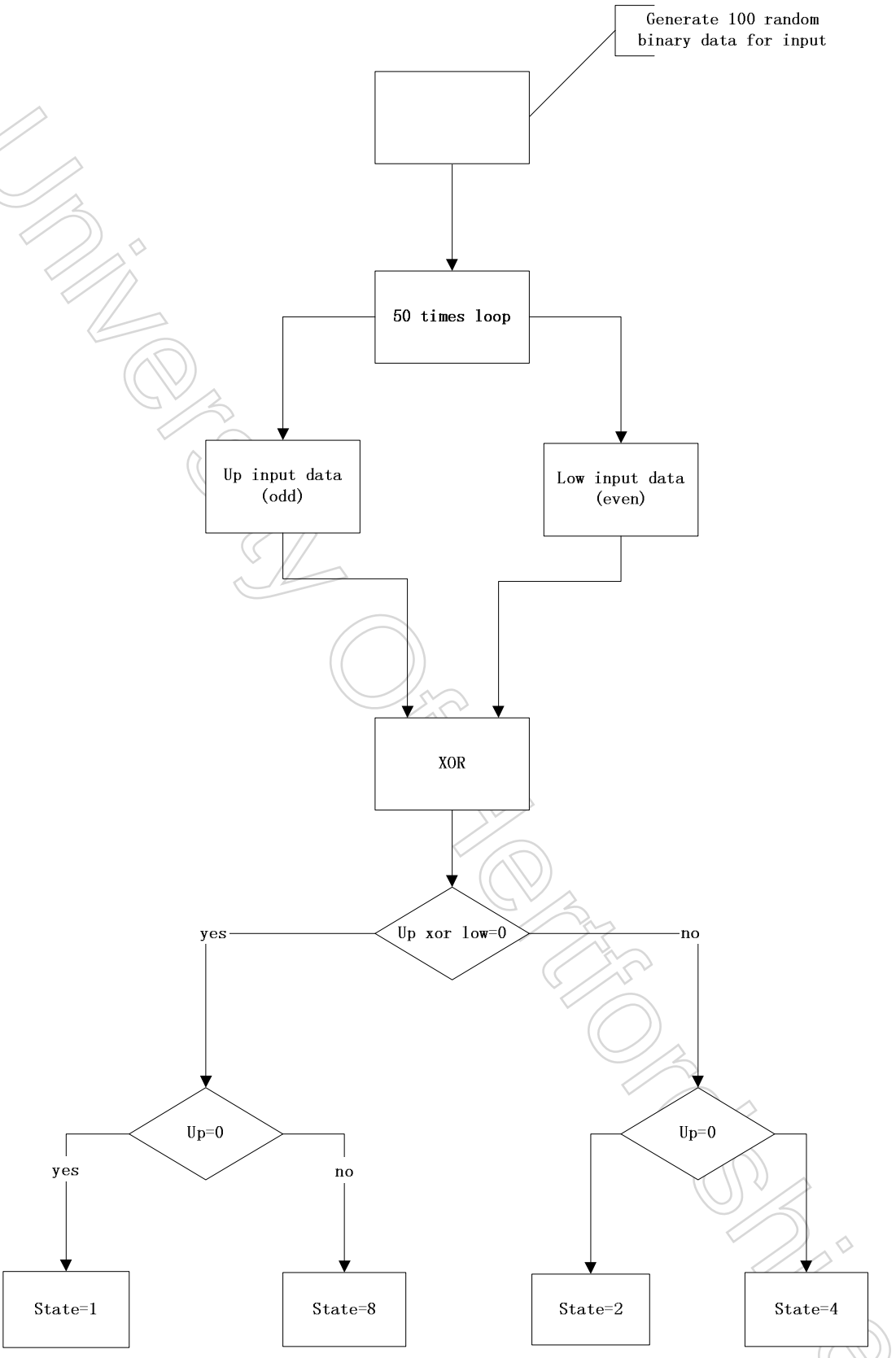

Figure 4.1: Flow chart of QPSK modulator in MATLAB

At the beginning of the program, a MATLAB function has been used to generate 100 random

binary numbers for input data. These numbers stored in an array named “input”. Next is a 50 times loop. In this loop, the input array will be separated from serial to parallel. All even position of the input array numbers will be stored in another array named “low” and all odd position of input array numbers will be stored in the array named “up”. After that, make out the XOR operation between the “up” and “low”. Here, the XOR needs to be introduced first. The XOR is eXclusive OR logic function. This function is like a binary addition but without any carry operations. The table 4.2 shows how the XOR work.

XOR	Input 1	
	0	1
Input 2 0	0	1
1	1	0

Table 4.2: XOR function truth table [7]

In the table 2, we can observe that, when the input 1 and input 2 are same, the output is false (‘0’); when the input 1 and input 2 are different, the output is true (‘1’). According to this truth table, if the “up” XOR “low” equals ‘0’, means they are same and go to the next step is if “up” equals ‘0’ means “low” same as “up” is ‘0’ as well. So, the input data is ‘00’ and get the output of ‘00’ is state equals ‘1’ (0001 in binary). If the “up” is not equals ‘0’ means it equals ‘1’ so the “low” same as “up” and equals ‘1’ as well. That means the input data is ‘11’ and output is state equals ‘8’ (1000 in binary).

If the “up” XOR “low” is not equals ‘0’, means they are different. In addition, if “up” equals ‘0’, means the “low” is different with it and “low” equals ‘1’. That means the input data is ‘01’ and the output will be the state equals ‘2’ (0010 in binary). If the “up” is not equals ‘0’, means it equals ‘1’ so the “low” is different with the “up”, means “low” equals ‘0’. So, the input data is ‘10’ and output is state equals ‘4’ (0100 in binary). For details of code of this part, please refer to appendix 1 (modulator in MATLAB part).

After the 50 times loop, the modulator will get the 50 numbers, which are the states of 50 pair of input data. Because of all states are decimal, it is easy to understand. However, the VHDL part is all binary data. For more convenience to compare the result, a function of MATLAB has been used to convert the state data from decimal to binary. This function called “dec2bin”. For details of this code, please refer to appendix 1 as well.

4.2 QPSK demodulator on MATLAB

The QPSK demodulator converts the modulated signal to the original input data. Practically, the demodulator needs the noise reduce equipment to increase the accuracy of the data that received from the receiver. It is because of the modulated signal transmits through the medium will mixed with some noise and unwanted signals. If these signals have without any noise reduction equipment, the demodulator can not be used to get the original input data. However, this project has designed in the computer and test the design environment has no noise. In addition, because of time constraint, the researcher has no enough time to add some noise reduction equipment in designing this project. However, the researcher will improve this project in the future works.

Figure 4.2 shows the flow chart of the QPSK demodulator in MATLAB.

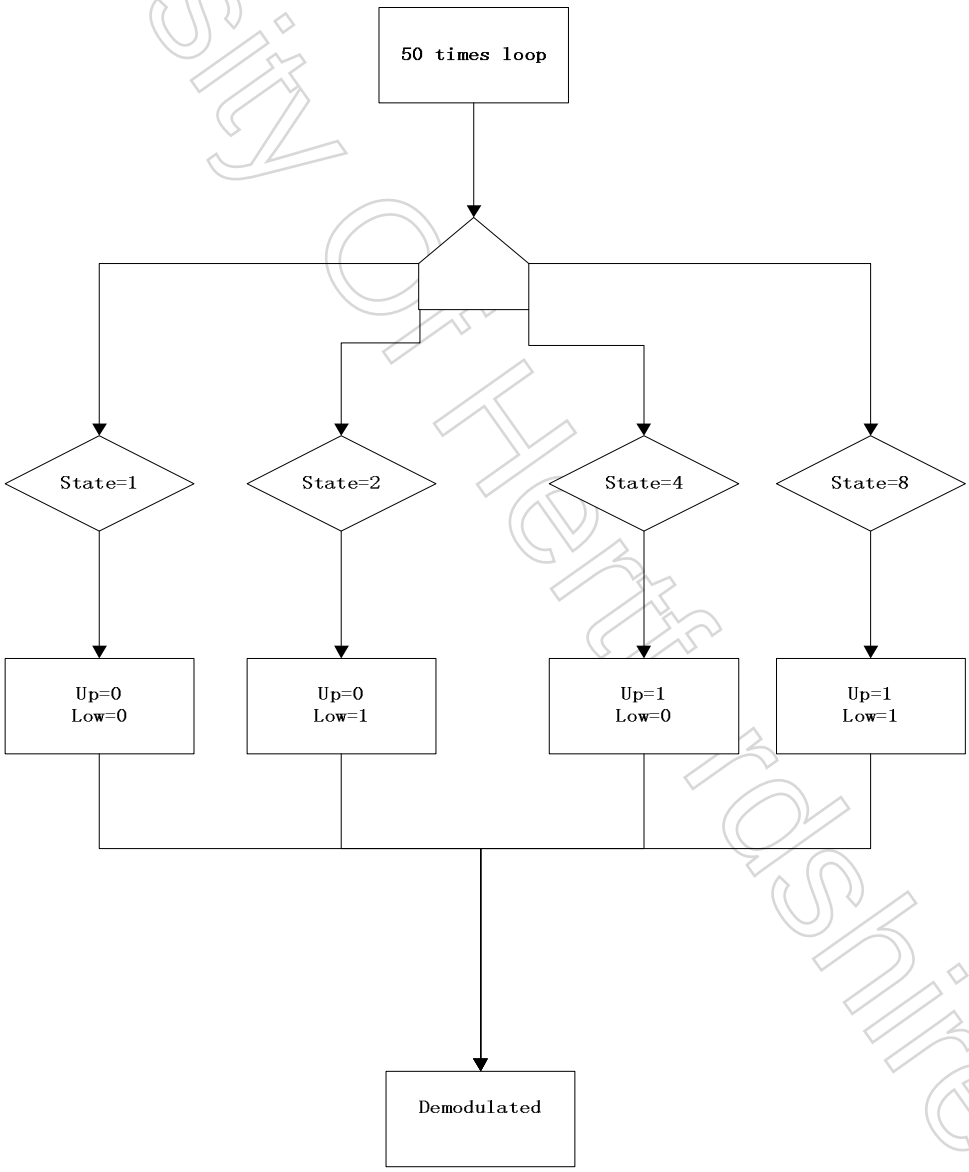

Figure 4.2: Flow chart of QPSK demodulator in MATLAB

This chart shows the 50 times loop. In this loop, the array of state that creates in the modulator has been used to estimate what the demodulated signal is. If the state = 1, that means the demodulated signal is "00", if the state = 2, means the demodulated signal is "01", if the state = 4, means the demodulated signal is "10" and if the state = 8, means the demodulated signal is "11". Table 4.3 shows the relationship between the state and the demodulated signal of the demodulator.

State	1 (0001 in binary)	2 (0010 in binary)	4 (0100 in binary)	8 (1000 in binary)
Demodulated signal	00	01	10	11

Table 4.3: relationship between state and demodulated signal

Because of there are 50 numbers of states, each of the demodulated signal is a pair of data and all demodulated signals are 100 binary numbers. In addition, these 100 binary numbers should be the same as the 100 numbers that stored in the input array which created in modulator part.

In the demodulator part design, the researcher used the parallel structure. That can be used to increase the speed of function. However, the parallel structure will increase the cost of the design as well, especially in the hardware design. About this issue, the researcher will discuss in the VHDL part. For all details of the QPSK demodulator in MATLAB, please refer to appendix 2 (demodulator in MATLAB part).

This demodulator needs the state data that have generated in the modulator. So, the demodulator can not be run without the output of the modulator. In addition, the output of the demodulator should be the same as the input of the modulator and it is the data that the user wants to be transmitted. Moreover, in this demodulator design, the researcher using the 'switch' functions to select the value of state. This is the parallel structure. In software design, the serial and parallel is not very different on program running. But they will be very different in hardware design. This will be discussed in next part.

Chapter 5 QPSK modulator and demodulator design on FPGA:

5.1 QPSK modulator design on FPGA

In this modulator, there are 4 ports on the hardware. 3 input port (reset, clock and input_data) and 1 output port (modulated_signal). There are 3 parts of the program. First part is the state registration process. Second is logic process and last one is modulated process. Figure 5.1 shows the relationship with these three parts.

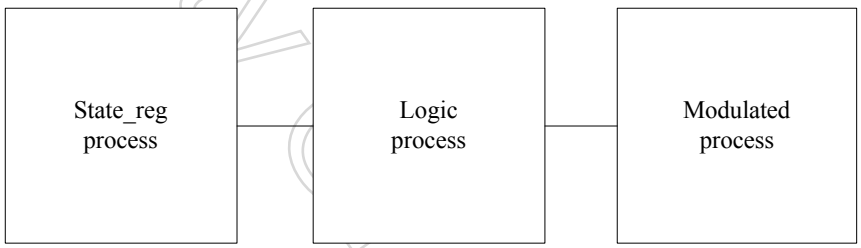

Figure 5.1: Modulator structure in VHDL

The function of State_reg process is to reset the modulator and get ready to modulating the input data to the modulated signal. In addition, the logic process estimate the value of the state signal and give the value to the temp signal and make decision for next state. In the modulated process, the modulated signal will be given the value according to the temp signal value.

Figure 5.2 shows the flow chart of the state_reg process

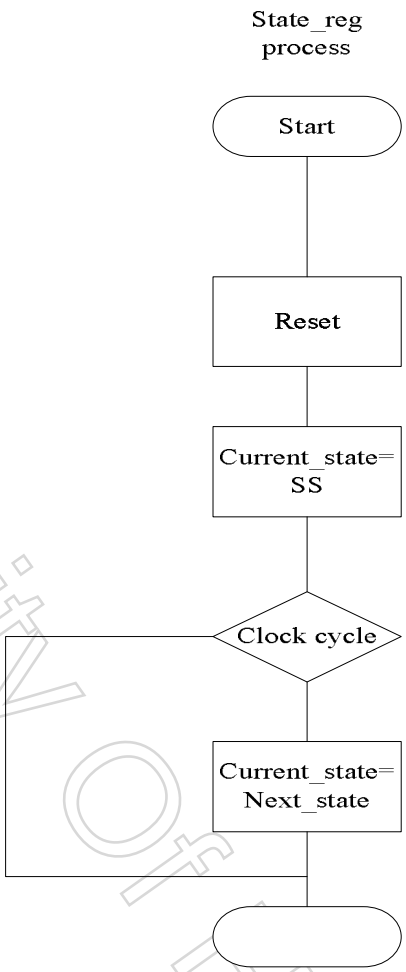

Figure 5.2: Flow chart of state_reg process

In this process, the modulator will be reset first. All signals will be given the starting value. The Current_state signal will be given the value 'SS'. Next, the each clock cycle, the current_state signal will be given the value of next_state.

Figure 5.3 shows the flow chart of the logic process

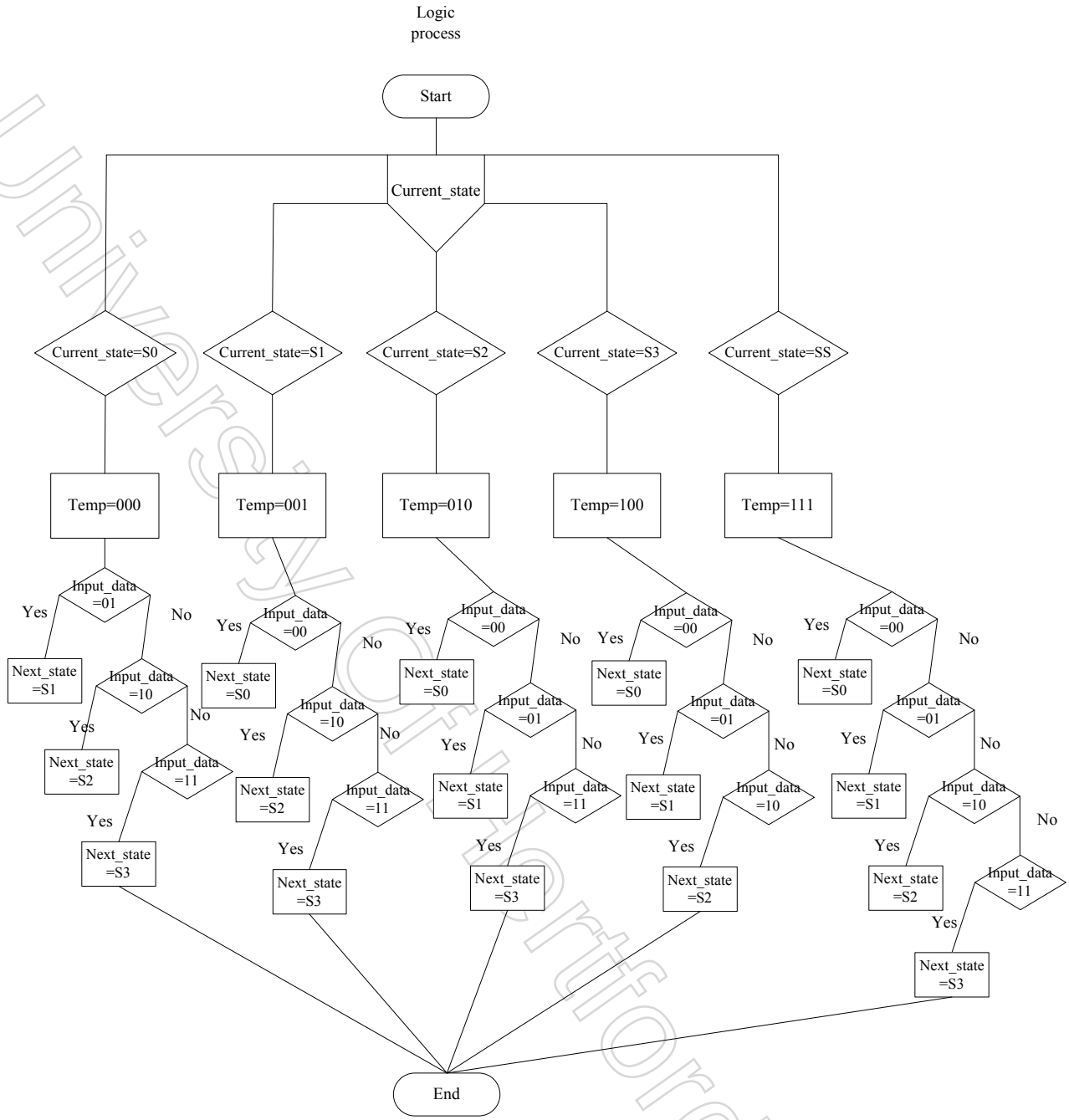

Figure 5.3: Flow chart of logic process

This process is the most important process in this project design. Firstly, the current_state will be given the 'SS' in the state_reg process. Next, there are 5 possibilities of the current_state (S0, S1, S2, S3 and SS). When the current_state equals 'S0', the temp signal will equal '000'. After that, there are 3 'if' functions for estimate the value of input_data and give the value S1, S2, S3 or SS to the next_state. Table 5.1 shows the relationship between input_data and state value.

Input_data	No input	00	01	10	11
State	SS	S0	S1	S2	S3

Table 5.1: Relationship between input_data and state

Moreover, the table 5.2 shows the relationship between current_state and temp signal

Current_state	SS	S1	S2	S3	S4
Temp signal	111	000	001	010	100

Table 5.2: Relationship between current_state and temp signal

Each clock cycle, the logic process will estimate the next_state and in the state_reg process, the current_state will be given the value of next_state that from the logic process. In other words, the logic process and state_reg process like two shift registers.

In the modulated process, the modulated signal will be given according to the temp signal. Figure 5.4 shows the flow chart of modulated process.

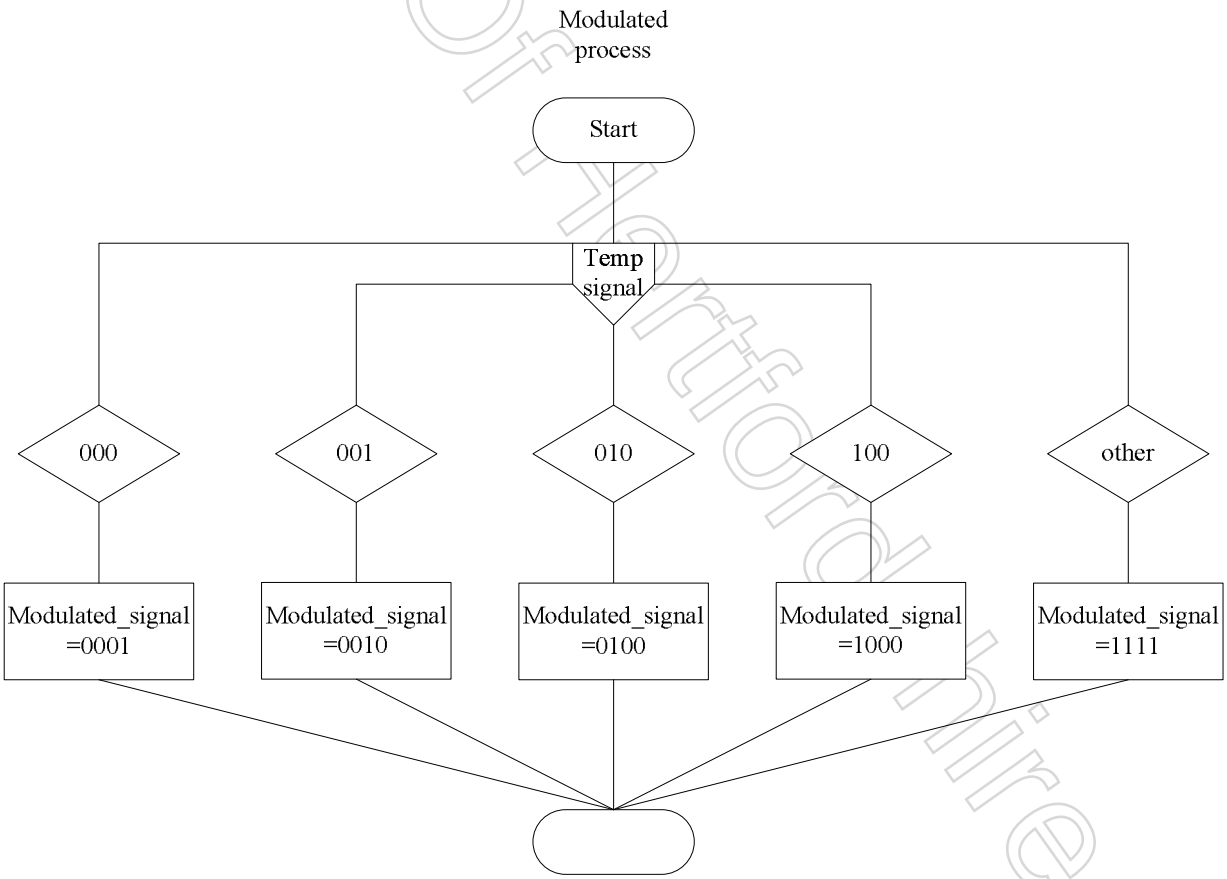

Figure 5.4: Flow chart of modulated process

Table 5.3 shows the relationship between temp signal and modulated_signal

Temp signal	000	001	010	100	other
Modulated_signal	0001 (1 in decimal)	0010 (2 in decimal)	0100 (4 in decimal)	1000 (8 in decimal)	1111

Table 5.3: Relationship between temp signal and modulated_signal

In the logic and modulated process, the researcher used the 'when' function. This function is parallel structure. In the hardware design, the parallel structure of program will increase the speed of simulation and synthesis, but it can increase the number and cost of the hardware gates. In the other hand, the serial will save the money in the hardware design, but the speed will much slower than the parallel structure. So the hardware design is a trade-off between the performance and the cost. In this project design, the researcher selected the parallel structure, because of this project is not very huge design, the cost will very low although the parallel structure has been used. In addition, the speed of simulation and synthesis will be faster than the serial structure. For all details of this part design code, please refer to appendix 3 (modulator in VHDL part).

5.2 QPSK demodulator design on FPGA

The structure of demodulator is very similar with the modulator. In the QPSK demodulator, there are four ports (3 input ports and 1 output port). 3 input ports are clock, reset and modulated_signal. The modulated_signal is the output of the modulator. The output port of the demodulator is original_data. Moreover, there are 3 blocks in the QPSK demodulator. In addition, the function of these 3 parts is similar with the modulator as well. Figure 5.5 shows the relationship between these 3 parts blocks

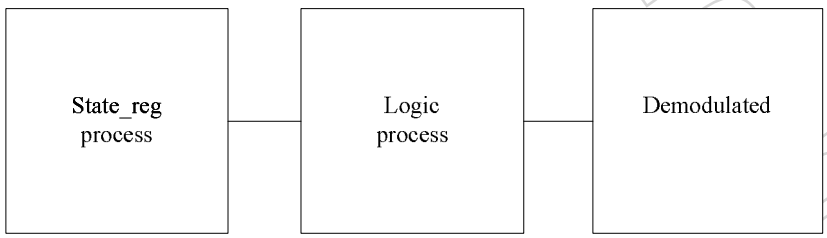

Figure 5.5: Relationship between state_reg, logic and demodulated process

Figure 5.6 shows the flow chart of state_reg process

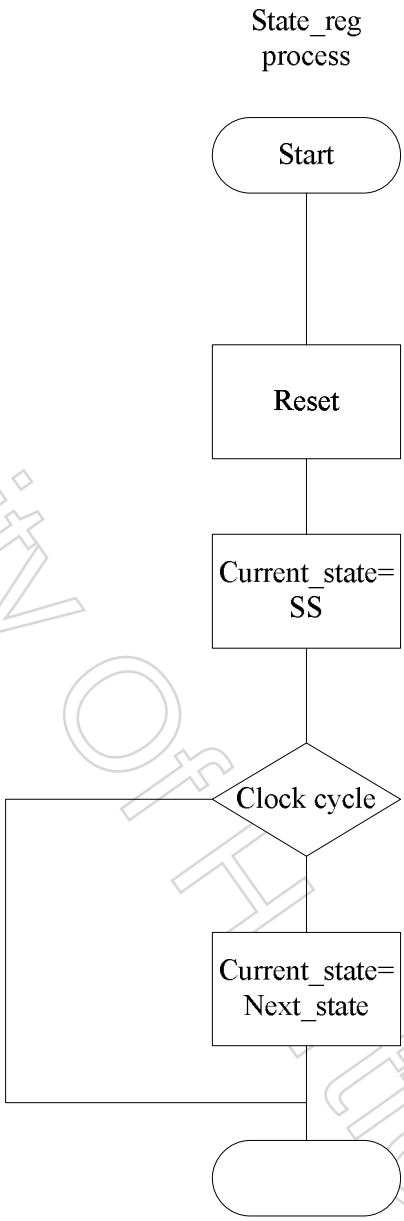

Figure 5.6: Flow chart of QPSK demodulator

This is exactly the same as the modulator. It is because of both of them need a state registration before the logic process. Figure 5.7 shows the flow chart of logic process.

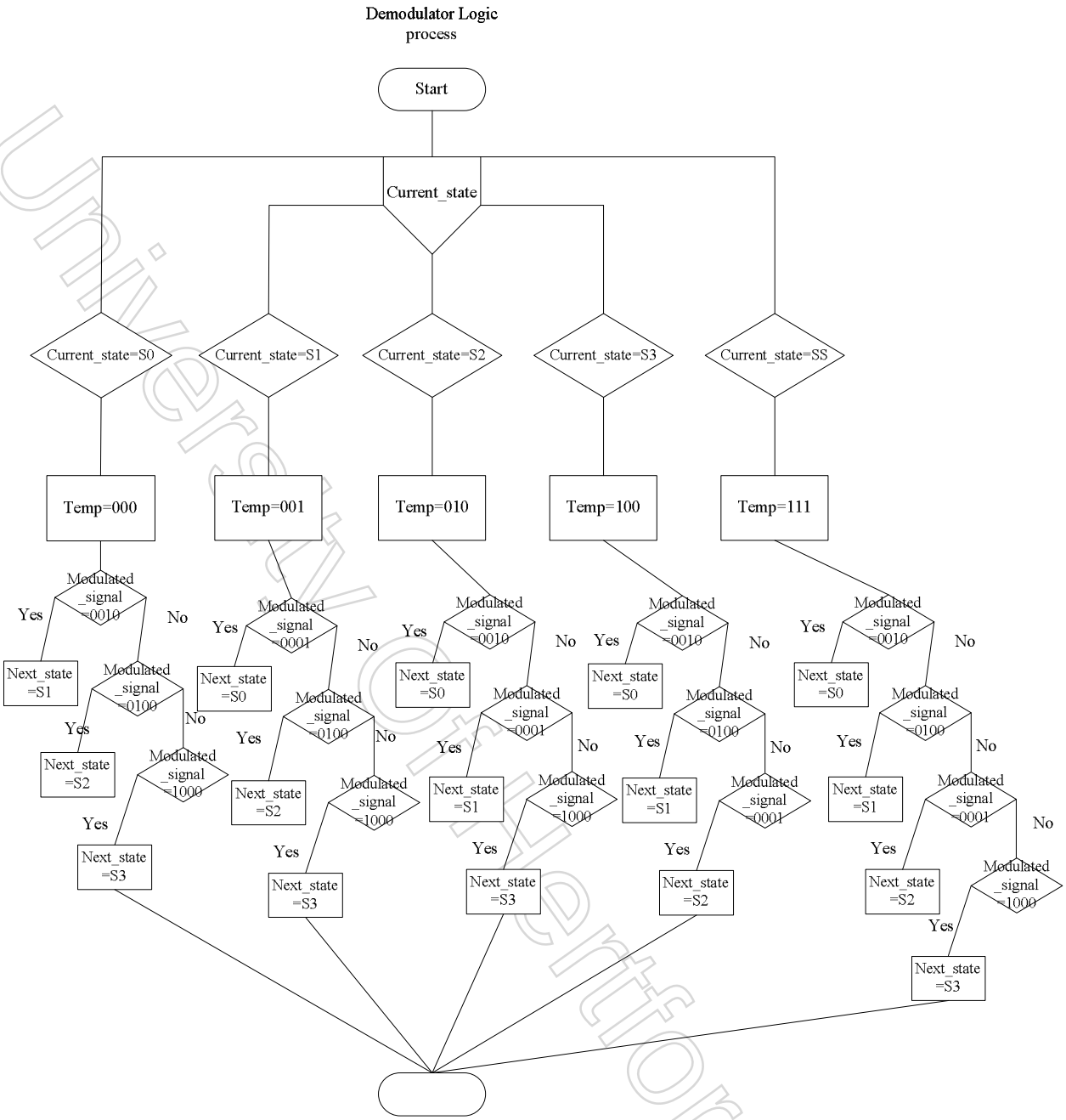

Figure 5.7: Flow chart of QPSK demodulator

This structure is the same as the modulator but the signals are different. The current_state decide the temp signal and the next_state will be decided according to the modulated_signal that is from the modulator. After the process, the temp signal will be stored in a shift register for next demodulated process. Figure 5.8 shows the flow chart of demodulated process.

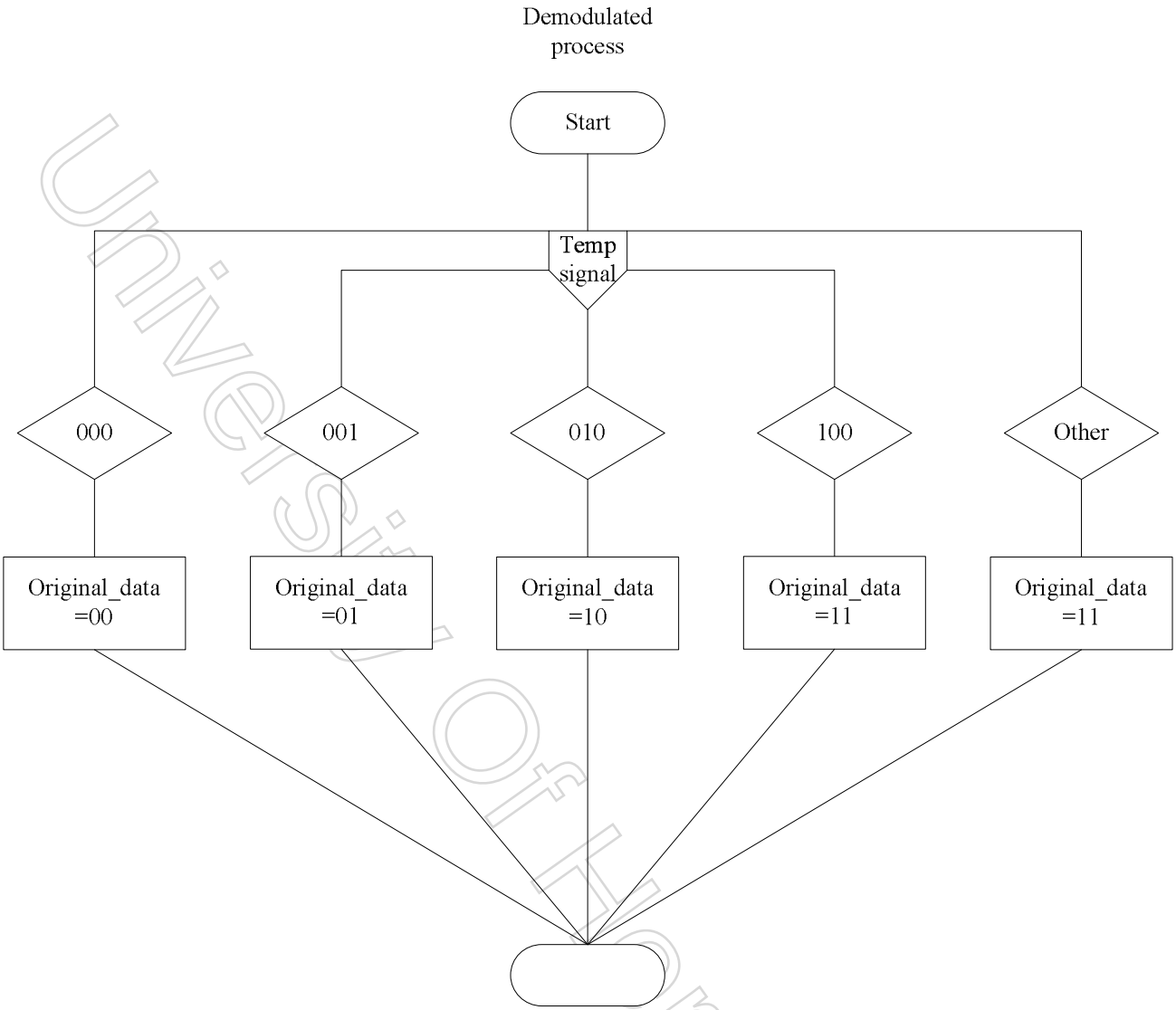

Figure 5.8: Flow chart of QPSK demodulated process

In this process, the value of original_data will be given according to the temp signal that generated from the logic process. The output of the demodulator is the original_data and this output should be the same as the input of the modulator.

For all details of this part design code, please refer to appendix 4 (demodulator in VHDL)

Chapter 6 Simulation and result analysis

6.1 Result of MATLAB

6.1.1 Modulator results on MATLAB

In the MATLAB design part, the input of modulator is an array generated by the 'rand' function that builds in the MATLAB. For the future check in VHDL part, the values of this input array will be stored in a text file called 'input.txt'. The researcher will generate 10 groups of the input data for more accuracy when checking the design. The result of this modulator is an array as well. The array stored the values of the state in decimal. In addition, the state in binary number will be stored in a text file called 'output_modulator_MATLAB.txt'. Table 6.1 shows the first group of the input data and output data.

Input	10101100101111001101001000010001010011011010111000010101011101111100011011011011111101001100100100001
Output (decimal)	4 4 8 1 4 8 8 1 8 2 1 4 1 2 1 2 2 1 8 2 4 4 8 4 1 2 2 2 2 8 2 8 8 1 2 4 8 2 4 8 8 8 2 1 8 1 4 2 1 2
Output (binary)	0100 0100 1000 0001 0100 1000 1000 0001 1000 0010 0001 0100 0001 0010 0001 0010 0010 0001 1000 0010 0100 0100 1000 0100 0001 0010 0010 0010 0010 1000 0010 1000 1000 0001 0010 0100 1000 0010 0100 1000 1000 1000 0010 0001 1000 0001 0100 0010 0001 0010

Table 6.1: Values of input and output of modulator in MATLAB

Please refer to appendix 5 (Results of input and output of modulator and demodulator in MATLAB and VHDL) for other groups of input and output.

In table 6.1, the inputs are 100 binary numbers. Every two numbers is a group. The QPSK can modulate input data with 2 numbers. Thus, the outputs of each group are 50 decimal numbers. The relationship between input and output has been introduced in table 4.1. The researcher make the four state in '1', '2', '4', '8', because of these four numbers can be easily converted to the binary and this will supply more convenience to the VHDL result compare.

6.1.2 Demodulator results on MATLAB

In the demodulator part, the input of the demodulator is the output of the modulator that is the state array, 50 decimal numbers for each group. The output of the demodulator is an array that included 100 binary numbers. In addition, the output of demodulator will be stored in a text file called 'output_demodulator_matlab.txt'. Table 6.2 shows the values of input and output of demodulator in MATLAB.

Input (decimal)	4 4 8 1 4 8 8 1 8 2 1 4 1 2 1 2 2 1 8 2 4 4 8 4 1 2 2 2 2 8 2 8 8 1 2 4 8 2 4 8 8 8 2 1 8 1 4 2 1 2
Output	10101100101111001101001000010001010011011010111 00 00101010111011111000110110110111111010011001001000 1

Table 6.2: Values of input and output of QPSK demodulator in MATLAB

According to the input of the demodulator, the output of the demodulator should be the same as the input of the QPSK modulator. Please refer to appendix 5 (Results of input and output of modulator and demodulator in MATLAB and VHDL) for other groups of input and output. Moreover, because of the MATLAB is a high level design environment, the speed of the program running is very fast. The researcher do not need to waste time on the program running. In the MATLAB design, the code has been written in simply way and the researcher has spent one week to finish this part of design. Basically, this part design is a check process. It gives the simple ideas and structure of QPSK modulator and demodulator to the researcher. In addition, the researcher can use these ideas and structures to design the VHDL part. Although, there are many differences between the software design and hardware design, the theory should be same. Moreover, the MATLAB part gives much convenience of the last result check in order to finish this project. In conclusion, the MATLAB is very necessary for hardware design.

6.1.3 MATLAB results analysis

In MATLAB part design, there are 10 groups of input have been generated by the modulator and 10 groups of output have been stored in 10 text file. The same process has been done in the demodulator. Through checking, each group of input of the modulator is same as the output of the demodulator. That means the transmitting data is same as the transmitted data. This proves that the modulator and demodulator design is correct. In addition, every group of input and output has been stored in text files for compare with the VHDL part design.

Moreover, this project assumes that there is no noise during the data transmission. So the input of the modulator and the output of the demodulator are exactly same. However, in the practical, the demodulator is much more complex than the demodulator that the researcher designed. Many functions need to be designed for reducing the noise. Because of the time constraint, the researcher can not finish the project with all these complex functions, but the modulator and demodulator will be improved in the future works.

6.2 Simulations on ISE

6.2.1 TESTBENCH of modulator and demodulator design

Because of the VHDL design is hardware design, the main program is just a description of input and output ports relationship. There is no input data for testing the accuracy of the design. The ISE provides a platform to the designers to check the design whether correct or not. This platform called TESTBENCH. Actually, the TESTBENCH is a part of VHDL code. It can transport the input data into the designed block in software environment and display the output of the designed block on the screen. That is very convenience to check the hardware design. It will save lots of money before the design synthesizes and loads onto the board. Designers can find out the mistakes through the TESTBENCH and save time to download the code onto the board.

In this project design, the TESTBENCH has been used in each modulator and demodulator results check. For all details of the TESTBENCH, please refer to appendix 6 (modulator and demodulator TESTBENCH codes). For test the VHDL part design quickly and accurately, the TEXTIO functions has been used. This function can read the input data from a text file and write the result to another text file as well. Thus, the researcher can write the TESTBENCH to read the input data that generated by MATLAB from the 'input.txt' file and write the result of the VHDL design to another files called 'output_modulator_VHDL.txt' and 'output_demodulator_VHDL.txt'. In addition, the TESTBENCH can display the simulation results in the waveforms on the computer screen. This is a very direct way to display the simulation result. Designers can read these results faster than the numbers.

6.2.2 Modulator results on FPGA

In this QPSK modulator design, the input data is obtained from the modulator in MATLAB part. Using the same input can make a double check for the design accuracy. The output of the QPSK modulator on FPGA is a group of 50 numbers. All numbers of the output are modulated signal. Figure 6.1 shows the simulation results of the modulator TESTBENCH.

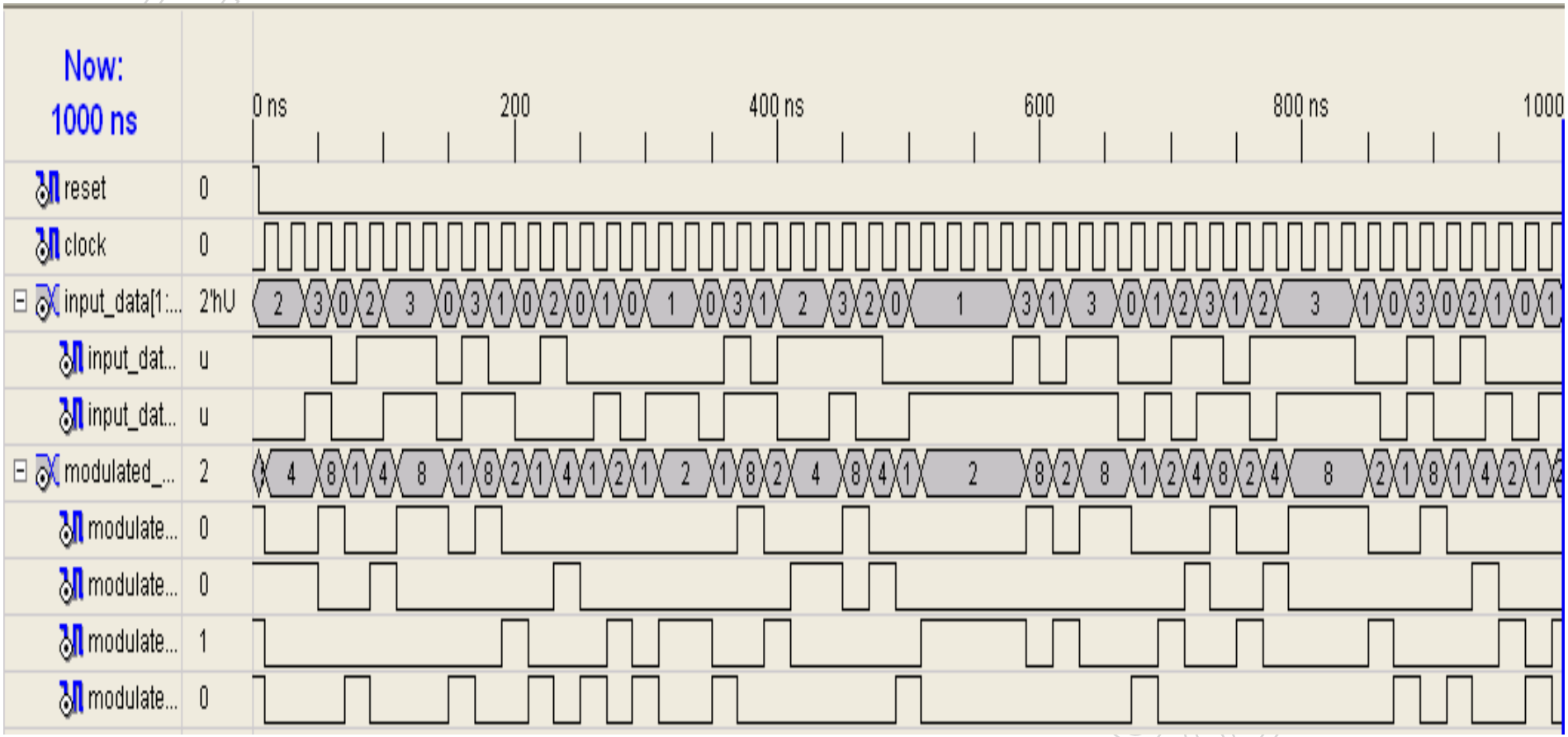

Figure 6.1: Simulation results of QPSK modulator TESTBENCH

In this figure, after the reset, the input data will be modulated to the modulated signal in each clock cycle. Because of the researcher using the TEXTIO function in TESHBENCH, the output data will be stored into a text file called 'output_modulator_VHDL.txt'. After the simulation, the output_modulator_VHDL.txt file will be created by the ISE. In this file, there are 50 lines numbers. Each line has 4 binary numbers which are the modulated signals that according to their input data. In addition, when the reset has been done, the modulator will save the 'UUUU' in the first line of the output_modulator_VHDL.txt file. It is because of after the reset, all signals in the modulator will be set to the original state or value. Table 6.3 shows the values of input and output of the QPSK modulator in VHDL part.

Input	10 10 11 00 10 11 11 00 11 01 00 10 00 01 00 01 01 00 11 01 10 10 11 10 00 01 01 01 01 11 01 11 11 00 01 10 11 01 10 11 11 11 01 00 11 00 10 01 00 01
Output	UUUU 0100 0100 1000 0001 0100 1000 1000 0001 1000 0010 0001 0100 0001 0010 0001 0010 0010 0001 1000 0010 0100 0100 1000 0100 0001 0010 0010 0010 0010 1000 0010 1000 1000 0001 0010 0100 1000 0010 0100 1000 1000 1000 0010 0001 1000 0001 0100 0010 0001 0010

Table 6.3: values of input and output of the QPSK modulator in VHDL part

The output of the QPSK modulator will be stored for the QPSK demodulator. It is because the input of the demodulator is the output of the modulator. Without the output of the modulator, the demodulator can not work. Please refer to appendix 5 (Results of input and output of modulator and demodulator in MATLAB and VHDL) for other groups of the QPSK modulator in VHDL part. Moreover, the input of the modulator is same as the MATLAB part. So the output of MATLAB and VHDL should be the same if the design is correct.

6.2.3 Demodulator results on FPGA

The output of the demodulator in the VHDL part is 50 lines binary numbers. Each line includes 2 binary numbers. In addition, the output of the demodulator should be the same as the input of the modulator, because this is no noise transmitting and demodulating. The figure 6.2 shows the simulation results of the QPSK demodulator TESTBENCH.

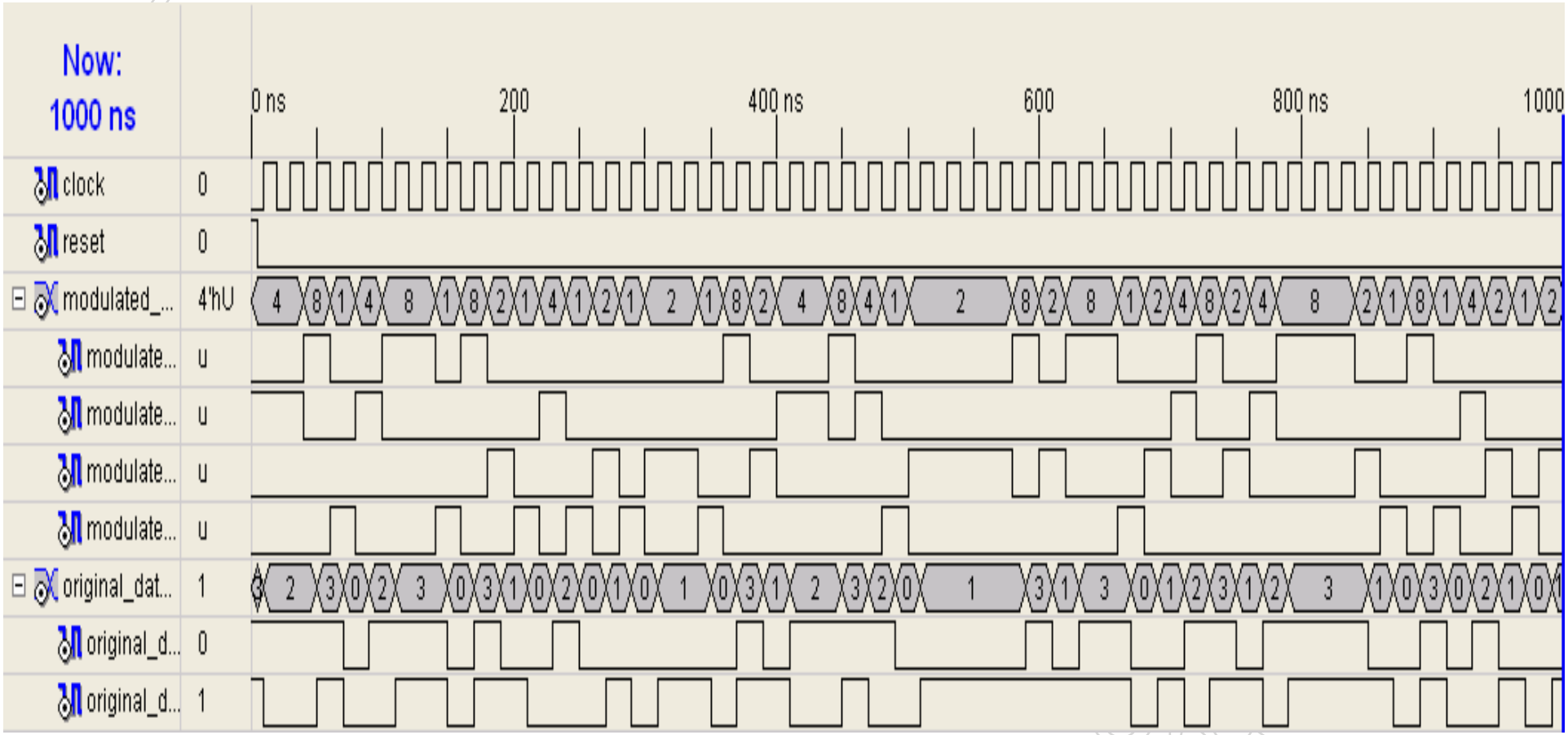

Figure 6.2: Simulation results of the QPSK demodulator TESTBENCH

After the reset, each modulated signal data will be demodulated back to the original data. In addition, the output of the demodulator is a 50 lines group numbers. Each line includes 2 binary numbers and it is same as the input of the modulator. That means the data which after transmitting is same as the data which before transmitting. In other words, the design of the modulator and demodulator is correct. Please refer to appendix 5 (Results of input and output of modulator and demodulator in MATLAB and VHDL) for other groups of the QPSK demodulator in VHDL part.

6.2.4 VHDL results analysis

In the VHDL design, the modulator and demodulator results have been discussed. The inputs of the modulator are the same as the MATLAB. In addition, the output of the modulator is the input of the demodulator and the output of the demodulator is same as the input of the modulator. For more accurate, there are another groups of input will be checked and get the outputs. Every group of the demodulator's outputs is the same as the inputs of the modulator as well. Because of there is no noise, the output of the demodulator is exactly the same as the input of the modulator.

Moreover, the first line of the modulator's output and demodulator's output is not numbers. It is because of the first reset has been done and all signals and values need to be reset to the start state. That is no affect to the final output of the demodulator. For the future works, the 'U' symbol will be removes after the demodulating.

6.2.5 VHDL synthesis results

Because of this project design is a hardware design, all program codes will be converted to the RTL (Resistor Transistor Logic). The ISE provides this function. It can create the RTL schematic according to the VHDL code design. Figure 6.3 shows the QPSK modulator RTL schematic.

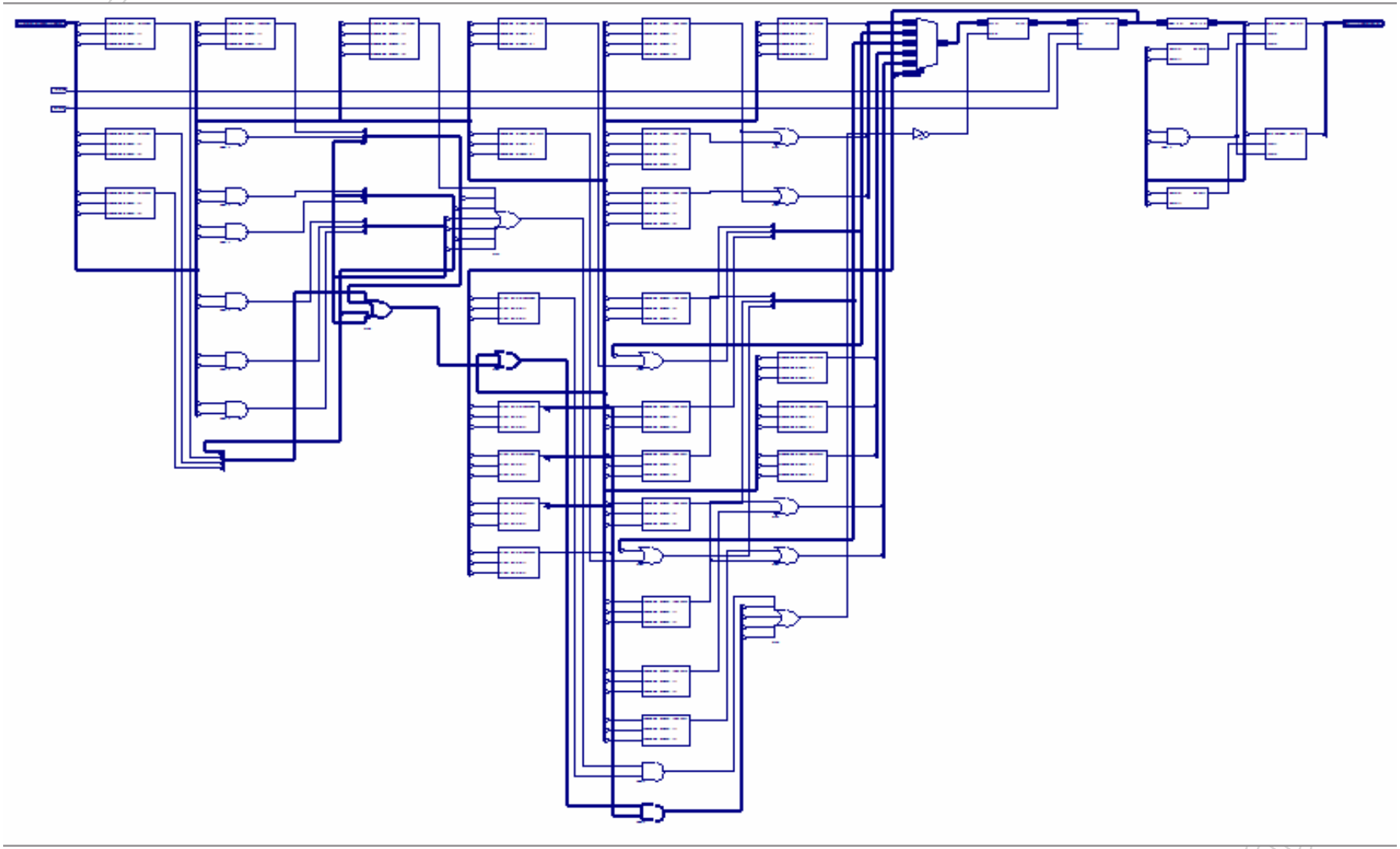

Figure 6.3: QPSK modulator RTL schematic

Figure 6.4 shows the demodulator RTL schematic

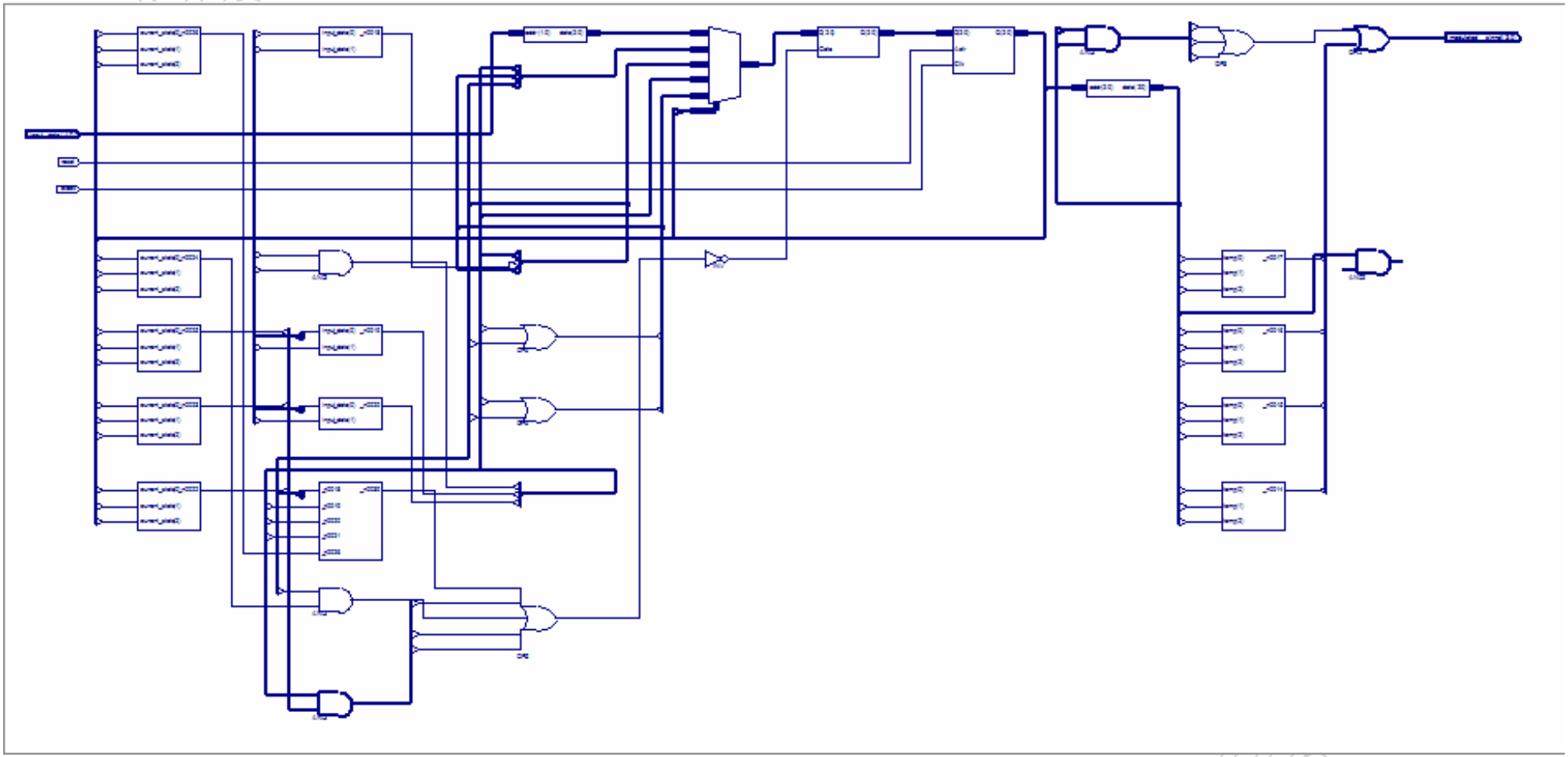

Figure 6.4: RTL schematic of demodulator

In addition, the ISE provides the report of the synthesis. This provides the help to the manufactures for built the gates on the board. For all details of the modulator and demodulator synthesis report, please refer to appendix 7 (synthesis report of modulator and demodulator).

6.3 Result compare between MATLAB and VHDL part

For double check of this project, the researcher will compare the output of the MATLAB part with the VHDL part. If they are same, it proves that the design is correct. According to the pervious chapters, the input of the modulator in MATLAB and VHDL is the same. Thus, the output of the modulator in MATLAB is same as the output of the modulator in VHDL. In addition, the output of the demodulator in MATLAB is same as the output of the demodulator in VHDL as well. Other groups of outputs are same as well. The researcher has done a lot of groups to check the design, because of the accuracy of this design is very important. If the modulator and demodulator are not accurate enough, the data will be lost or the receiver will receive the signal that is unwanted. This kind of design is useless.

Moreover, all inputs of the modulator are generated by the MATLAB function randomly. In other words, the input numbers are unorganized. The errors of the design should be found with these kinds of input data. In conclusion, it could be suggested that this project design 'QPSK modulator and demodulator on FPGA' is correct and completed.

Chapter 7 Conclusion and future works:

In conclusion, the theory of digital communication and QPSK modulator and demodulator has been introduced. In addition, the design methods of the project have been discussed. Through the MATLAB part guiding, the VHDL code has been designed. The flow chart of modulator and demodulator give much helps for programme code design in both MATLAB and VHDL part. Moreover, the charts give helps to other readers to understand this project design as well. After the design part, the results have been checked for the accuracy of this project design. 10 groups of input data have been checked and all outputs are correct. In addition, the RTL schematic chart of modulator and demodulator has been displayed. These give lots of information to the hardware builder. Because of this project's aim is to design the QPSK modulator and demodulator on software environment, the code and RTL schematic are not going to translate to the board.

This design modulator and demodulator are working in a perfect environment, that means there is no noise and other situation that can affect the data transmitting. Thus, the design can not be used in the real communication directly. There are lots of functions needs to improved and added. For example, noise reduction and error correction functions. Due to the time constraint, the researcher has no enough time to design a perfect MODEM during the three months. However, this project will be improved in the future time. In the future works, the QPSK modulator and demodulator could be combined together and some noise reduction and error correction functions will be added.

Finally, this project design has been completed. Although, there are lots of shortfalls in this design, the main functions of QPSK modulator and demodulator have been implemented.

References

[1] Aghvami. A. H., "Digital modulation techniques for mobile and personal communication systems", *Electronics & Communication Engineering Journa*l, Vol.5, Issue.3, pp. 125-132, June 1993

[2] Alinier. N., (2006), *radio and mobile communication systems notes*, University of Hertfordshire

[3] Blake.R., (1997), "comprehensive electronic communication", *Elm Street publishing services*, Inc., ISBN 0-314-20140-8

[4] Glover.I.A & Grant.P.M., (1998), "digital communications", *Prentice Hall Europe*., ISBN 0-13-565391-6

[5] Hou B., (2005), *DSP and ASIC architecture and Applications* University of Hertfordshire

[6] http://en.wikipedia.org/wiki/MATLAB

[7] http://www.accs.com/p_and_p/RAID/Definitions.html

[8] http://news.surfwax.com/telecom/files/QPSK.html

[9] http://www.rttonline.com/HotTopics/hottop_may02.htm

[10] http://www.synopsys.com/news/pr_kit/eda_glossary.html

[11] http://www.xilinx.com/ise/logic_design_prod/foundation.htm

[12] Moore. M. S. et al., (2002), "telecommunications a beginner's guide", *McGraw-Hill Companies*, ISBN 0-07-219356-5

[13] Otung. I., (2001), "Communication Engineering Principles", St. Martin's Press LLC Scholarly and Reference Division and Palgrave Publishers Ltd., ISBN 0-333-77522-8

[14] Pursley. M.B., (2005), "Introduction to Digital Communications", *Pearson Education*, Inc., ISBN 0-13-123392-0

[15]Proakis.J.G, (1995), "Digital communication", *McGraw-Hill Book Co.* ISBN 0-07-051726-6

[16] Schiller. J., (2003), "Mobile Communications", *Pearson Education Limited*, ISBN

0-321-12381-6

[17] Sjoholm. S & Lindh. L., (1997), "VHDL for Designers", *Prentice Hall Europe*, ISBN 0-13-473414-9

[18] Yalamanchili. S., (1998), "VHDL Starter's Guide", *Prentice- Hall*, Inc., ISBN 0-13-519802-X

Appendix 1: Modulator in MATLAB part

```
clear all
input=randint(1,100);%generate 100 random input data%
l=1;
lenth=100;
for s=1:lenth/2 %50 times loop to get the output of the modulator%
    up(l)=input(2*l-1);%change the serial numbers to parallel%
    low(l)=input(2*l);
    if xor(up(l),low(l))==0
        if up(l)==0
        state(l)=1;
        else
        state(l)=8;
    end
    else
        if up(l)==0
        state(l)=2;
        else
        state(l)=4;
        end
    end
l=l+1;
end
state
state_binary=dec2bin(state)%convert the decimal of modulator output to the binary number%
```

Appendix 2 :Demodulator in MATLAB part

```
l=1;
for s=1:50 %50 times loop%
switch state(l) %estimate the output of the modulator and give the value of the
demodulated_signal%
    case 1
        up(l)=0;
        low(l)=0;
    case 2
        up(l)=0;
        low(l)=1;
    case 4
        up(l)=1;
        low(l)=0;
    case 8
        up(l)=1;
        low(l)=1;
    otherwise
        disp('error')
end
demodulated_signal(1,2*l-1)=[up(l)];%convert the parallel numbers to the serial%
demodulated_signal(1,2*l)=[low(l)];
l=l+1;
end
demodulated_signal
```

Appendix 3: Modulator in VHDL part

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:      08:12:05 06/23/06
-- Design Name:       YANG LE
-- Module Name:      modulator - Behavioral
-- Project Name:     QPSK modulator and demodulator on FPGA

----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity modulator is
      Port ( reset : in std_logic;
                  clock : in std_logic;
                  input_data : in std_logic_vector (1 downto 0);
                  modulated_signal : out std_logic_vector (3 downto 0) );
end modulator;

architecture Behavioral of modulator is

      type state_type is (SS,S0,S1,S2,S3);
     signal current_state, next_state: state_type;
      signal temp: std_logic_vector(2 downto 0);
begin

   state_reg:process (reset, clock) is
   begin
      if (reset='1') then
                  current_state <= SS;
      elsif (clock'event and clock='1') then
            current_state <= next_state;
      end if;

     end process;
```

```
logic: process (current_state,input_data) is
 begin
 case current_state is
 when s0=> temp<="000";
       if input_data="01" then
            next_state<=s1;
       elsif input_data="10" then
            next_state<=s2;
       elsif input_data="11" then
            next_state<=s3;
       end if;


 when s1=> temp<="001";
       if input_data="00" then
            next_state<=s0;
       elsif input_data="10" then
            next_state<=s2;
       elsif input_data="11" then
            next_state<=s3;
       end if;

 when s2=> temp<="010";
       if input_data="01" then
            next_state<=s1;
       elsif input_data="00" then
            next_state<=s0;
       elsif input_data="11" then
            next_state<=s3;
       end if;

 when s3=> temp<="100";
       if input_data="01" then
            next_state<=s1;
       elsif input_data="10" then
            next_state<=s2;
       elsif input_data="00" then
            next_state<=s0;
       end if;

 when ss=> temp<="111";
       if input_data="01" then
            next_state<=s1;
       elsif input_data="10" then
```

```
            next_state<=s2;
        elsif input_data="00" then
            next_state<=s0;
        elsif input_data="11" then
            next_state<=s3;
        end if;
      end case;
    end process;

modulated: process (temp) is

begin
        case temp is
        when "000"=>modulated_signal<="0001";
        when "001"=>modulated_signal<="0010";
        when "010"=>modulated_signal<="0100";
        when "100"=>modulated_signal<="1000";
        when others =>modulated_signal<="1111";
        end case;
end process;


end behavioral;
```

Appendix 4: Demodulator in VHDL

```
----------------------------------------------------------------------------
-- Create Date:        12:00:47 07/22/06
-- Design Name:        YANG LE
-- Module Name:        demodulator - Behavioral
-- Project Name:      QPSK modulator and demodulator on FPGA
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity demodulator is
      Port ( modulated_signal : in std_logic_vector (3 downto 0);
                clock : in std_logic;
                reset : in std_logic;
                original_data : out std_logic_vector (1 downto 0));
end demodulator;

architecture Behavioral of demodulator is

type state_type is (SS,S0,S1,S2,S3);
    signal current_state, next_state: state_type;
     signal temp: bit_vector (2 downto 0);
begin

   state_reg:process (reset, clock) is
   begin
     if (reset='1') then
                current_state <= SS;
     elsif (clock'event and clock='1') then
           current_state <= next_state;
     end if;

    end process;

    logic: process (current_state,modulated_signal) is
     begin
     case current_state is
     when s0=> temp<="000";
```

```
        if modulated_signal="0010" then
            next_state<=s1;
        elsif modulated_signal="0100" then
            next_state<=s2;
        elsif modulated_signal="1000" then
            next_state<=s3;
        end if;

    when s1=> temp<="001";
        if modulated_signal="0001" then
            next_state<=s0;
        elsif modulated_signal="0100" then
            next_state<=s2;
        elsif modulated_signal="1000" then
            next_state<=s3;
        end if;

    when s2=> temp<="010";
        if modulated_signal="0010" then
            next_state<=s1;
        elsif modulated_signal="0001" then
            next_state<=s0;
        elsif modulated_signal="1000" then
            next_state<=s3;
        end if;

    when s3=> temp<="100";
        if modulated_signal="0010" then
            next_state<=s1;
        elsif modulated_signal="0100" then
            next_state<=s2;
        elsif modulated_signal="0001" then
            next_state<=s0;
        end if;

    when ss=> temp<="111";
        if modulated_signal="0010" then
            next_state<=s1;
        elsif modulated_signal="0100" then
            next_state<=s2;
        elsif modulated_signal="0001" then
            next_state<=s0;
        elsif modulated_signal="1000" then
```

```
            next_state<=s3;
        end if;
    end case;
    end process;

demodulted: process (temp) is
begin

    case temp is
    when "000"=>original_data<="00";
    when "001"=>original_data<="01";
    when "010"=>original_data<="10";
    when "100"=>original_data<="11";
    when others =>original_data<="11";
    end case;
    end process;


end behavioral;
```

Appendix 5: Results of input and output of modulator and demodulator in MATLAB and VHDL

Group 2:

Input_g2:
10101100101111001101001000010001010011011010111000010101011101111100011011011011111101001100100010001

Output_matlab_modulator_g2:
0100 0100 1000 0001 0100 1000 1000 0001 1000 0010 0001 0100 0001 0010 0001 0010 0010 0001 1000 0010 0100 0100 1000 0100 0001 0010 0010 0010 0010 1000 0010 1000 1000 0001 0010 0100 1000 0010 0100 1000 1000 1000 0010 0001 1000 0001 0100 0010 0001 0010

Output_matlab_demodulator_g2:
10101100101111001101001000010001010011011010111000010101011101111100011011011011111101001100100010001

Output_vhdl_modulator_g2:
UUUU 0100 0100 1000 0001 0100 1000 1000 0001 1000 0010 0001 0100 0001 0010 0001 0010 0010 0001 1000 0010 0100 0100 1000 0100 0001 0010 0010 0010 0010 1000 0010 1000 1000 0001 0010 0100 1000 0010 0100 1000 1000 1000 0010 0001 1000 0001 0100 0010 0001 0010

Output_vhdl_demodulator_g2:
UU10101100101111001101001000010001010011011010111000010101011101111100011011011011111101001100100010001

Group 3:

Input_g3:
10100011110011011010001011110001011010011011001101000000100110001001110000011101001100101100001010011

Output_matlab_modulator_g3:
0100 0100 0001 1000 1000 0001 1000 0010 0100 0100 0001 0100 1000 1000 0001 0010 0010 0100 0100 0010 0100 1000 0001 1000 0010 0001 0001 0001 0100 0010 0100 0001 0100 0010 1000 0001 0001 0010 1000 0010 0001 1000 0001 0100 1000 0001 0001 0100 0100 0010

Output_matlab_demodulator_g3:

10100011110011011010001011110001011010011011001101000000100110001001110000011101001100101100001010011

Output_vhdl_modulator_g3:
UUUU 0100 0100 0001 1000 1000 0001 1000 0010 0100 0100 0001 0100 1000 1000 0001 0010 0010 0100 0100 0010 0100 1000 0001 1000 0010 0001 0001 0001 0100 0010 0100 0001 0100 0010 1000 0001 0001 0010 1000 0010 0001 1000 0001 0100 1000 0001 0001 0100 0100 0010

Output_vhdl_demodulator_g3:
UU10100011110011011010001011110001011010011011001101000000100110001001110000011101001100101100001010011

Group 4:

Input_g4:
00000100011001011110010111000001010000100011110011000111101101110110101100110011111001111001001111110

Output_matlab_modulator_g4:
0001 0001 0010 0001 0010 0100 0010 0010 1000 0100 0010 0010 1000 0001 0001 0010 0010 0001 0001 0100 0001 1000 1000 0001 1000 0001 0010 1000 0100 1000 0010 1000 0010 0100 0100 1000 0001 1000 0001 1000 1000 0100 0010 1000 0100 0010 0001 1000 1000 0100

Output_matlab_demodulaor_g4:
00000100011001011110010111000001010000100011110011000111101101110110101100110011111001111001001111110

Output_vhdl_modulator_g4:
UUUU 0001 0001 0010 0001 0010 0100 0010 0010 1000 0100 0010 0010 1000 0001 0001 0010 0010 0001 0001 0100 0001 1000 1000 0001 1000 0001 0010 1000 0100 1000 0010 1000 0010 0100 0100 1000 0001 1000 0001 1000 1000 0100 0010 1000 0100 0010 0001 1000 1000 0100

Output_vhdl_demodulator_g4:
UU00000100011001011110010111000001010000100011110011000111101101110110101100110111001111001001111110

Group 5:

Input_g5:
01010111110110111011101010101000011011111001100000110010111011100001101011011100000010010101101100000

Output_matlab_modulator_g5:
0010 0010 0010 1000 1000 0010 0100 1000 0100 1000 0100 0100 0100 0100 0100 0001 0010 0100 1000 1000 0100 0010 0100 0001 0001 1000 0001 0100 1000 0100 1000 0100 0001 0010 0100 0100 1000 0010 1000 0001 0001 0001 0100 0010 0010 0010 0100 1000 0001 0001

Output_matlab_demodulator_g5:
010101111101101110111010101010000110111110011000001100101110111000011010110111000001001010110110000

Output_vhdl_modulator_g5:
UUUU 0010 0010 0010 1000 1000 0010 0100 1000 0100 1000 0100 0100 0100 0100 0100 0001 0010 0100 1000 1000 0100 0010 0100 0001 0001 1000 0001 0100 1000 0100 1000 0100 0001 0010 0100 0100 1000 0010 1000 0001 0001 0001 0100 0010 0010 0010 0100 1000 0001 0001

Output_vhdl_demodulator_g5:
UU0101011111011011101110101010100001101111100110000011001011101110000110101101110000100101011011000

Group 6:

Input_g6:
0011000001100111010110011100000011110111011100010011001011100011001000010101010110100010010001111100

Output_matlab_modualtor_g6:
0001 1000 0001 0001 0010 0100 0010 1000 0010 0010 0100 0010 1000 0001 0001 0001 1000 1000 0010 1000 0010 1000 0100 0001 0100 0010 0100 0010 0010 1000 0001 0010 0100 0010 0001 0001 0100 0100 0100 0100 1000 0010 0001 0010 0001 0100 0001 1000 1000 0001

Output_matalb_demodulator_g6:
0011000001100111010110011100000011110111011100010011001011100011001000010101010110100010010001111100

Output_vhdl_modulator_g6:
UUUU 0001 1000 0001 0001 0010 0100 0010 1000 0010 0010 0100 0010 1000 0001 0001 0001 1000 1000 0010 1000 0010 1000 0100 0001 0100 0010 0100 0010 0010 1000 0001 0010 0100 0010 0001 0001 0100 0100 0100 0100 1000 0010 0001 0010 0001 0100 0001 1000 1000 0001

Output_vhdl_demodulator_g6:

UU00110000011001110101100111000000111101110111100010011001011100011001000010
10101011010001001000111100

Group 7:

Input_g7:
00110111010110111010111110010110000101110000101111110110011111100000100101110
10101011100110101010101

Output_matlab_modulator_g7:
0001 1000 0010 1000 0010 0010 0100 1000 0100 0100 1000 1000 0100 0010 0010 0100
0001 0010 0010 1000 0001 0001 0100 1000 1000 1000 0010 0100 0010 1000 1000 0100
0001 0001 0100 0010 0010 1000 0010 0010 0010 0010 1000 0001 1000 0010 0010 0010
0010 0010

Output_matlab_demodualtor_g7:
00110111010110111010111110010110000101110000101111110110011111100000100101110
10101011100110101010101

Output_vhdl_modulator_g7:
UUUU 0001 1000 0010 1000 0010 0010 0100 1000 0100 0100 1000 1000 0100 0010 0010
0100 0001 0010 0010 1000 0001 0001 0100 1000 1000 1000 0010 0100 0010 1000 1000
0100 0001 0001 0100 0010 0010 1000 0010 0010 0010 0010 1000 0001 1000 0010 0010
0010 0010 0010

Output_vhdl_demodulator_g7:
UU00110111010110111010111110010110000101110000101111110110011111100000100101
11010101011100110101010101

Group 8:

Input_g8:
11000001100011011011000001001100011010101111101011100010011000000100001010010
11011100100001101011111

Output_matlab_modulator_g8:
1000 0001 0001 0010 0100 0001 1000 0010 0100 1000 0001 0001 0010 0001 1000 0001
0010 0100 0100 0100 1000 1000 0100 0100 1000 0100 0001 0100 0010 0100 0001 0001
0010 0001 0001 0100 0100 0010 0010 0100 1000 0100 0010 0001 0001 1000 0010 0010
1000 1000

Output_matlab_demodualtor_g8:
11000001100011011011000001001100011010101111101011100010011000000100001010010
11011100100001101011111

Output_vhdl_modulator_g8:
UUUU 1000 0001 0001 0010 0100 0001 1000 0010 0100 1000 0001 0001 0010 0001 1000 0001 0010 0100 0100 0100 1000 1000 0100 0100 1000 0100 0001 0100 0010 0100 0001 0001 0010 0001 0001 0100 0100 0010 0010 0100 1000 0100 0010 0001 0001 1000 0010 0010 1000 1000

Output_vhdl_demodulator_g8:
11000001100011011011000000100110001101010111110101110001001100000010000101001011011100100001101011111

Group 9:

Input_g9:
11111101001101010001111100100011011110011110100111111111110100000010010111000011010001010011011011111

Output_matlab_modulator_g9:
1000 1000 1000 0010 0001 1000 0010 0010 0001 0010 1000 1000 0001 0100 0001 1000 0010 1000 0100 0010 1000 0100 0100 0010 1000 1000 1000 1000 1000 0010 0001 0001 0001 0100 0010 0010 1000 0001 0001 1000 0010 0001 0010 0010 0001 1000 0010 0100 1000 1000

Output_matlab_demodulator_g9:
11111101001101010001111100100011011110011110100111111111110100000010010111000011010001010011011011111

Output_vhdl_modulator_g9:
UUUU 1000 1000 1000 0010 0001 1000 0010 0010 0001 0010 1000 1000 0001 0100 0001 1000 0010 1000 0100 0010 1000 0100 0100 0010 1000 1000 1000 1000 1000 0010 0001 0001 0001 0100 0010 0010 1000 0001 0001 1000 0010 0001 0010 0010 0001 1000 0010 0100 1000 1000

Output_vhdl_demodualtor_g9:
UU11111101001101010001111100100011011110011110100111111111110100000010010111000001101000101001101101111

Group 10:

Input_g10:
01101101110110100000110001111001111001000100100110000101001110111111010101100011011010000010101100000

Output_matlab_modulator_g10:

0010 0100 1000 0010 1000 0010 0100 0100 0001 0001 1000 0001 0010 1000 0100 0010 1000 0100 0010 0001 0010 0001 0100 0010 0100 0001 0010 0010 0001 1000 0100 1000 1000 0100 0100 0100 1000 0001 0010 0100 1000 0010 0001 0001 0010 0010 0010 0100 0001 0001

Output_matlab_demodulator_g10:
011011011101101000001100011110011110010001001001100001010011101111101010110001101101000001010110000

Output_vhdl_modulator_g10:
UUUU 0010 0100 1000 0010 1000 0010 0100 0100 0001 0001 1000 0001 0010 1000 0100 0010 1000 0100 0010 0001 0010 0001 0100 0010 0100 0001 0010 0010 0001 1000 0100 1000 1000 0100 0100 0100 1000 0001 0010 0100 1000 0010 0001 0001 0010 0010 0010 0100 0001 0001

Output_vhdl_demodulator_g10:
UU011011011101101000001100011110011110010001001001100001010011101111101010110001101101000001010110000

Appendix 6: Modulator and demodulator TESTBENCH codes

Modulator:

```
--------------------------------------------------------------------------------
-- Create Date:     17:56:23 07/18/2006
-- Design Name:     modulator
-- Module Name:     testbench.vhd
-- Project Name:    modulator

-- VHDL Test Bench Created by ISE for module: modulator

-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.    Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
library std;
use std.textio.all;
use ieee.std_logic_textio.all;
ENTITY testbench_vhd IS
END testbench_vhd;
ARCHITECTURE behavior OF testbench_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT modulator
    PORT(
        reset : IN std_logic;
        clock : IN std_logic;
        input_data : IN std_logic_vector(1 downto 0);
        modulated_signal : OUT std_logic_vector (3 downto 0)
        );
    END COMPONENT;
```

```
	--Inputs
	SIGNAL reset :	std_logic := '1';
	SIGNAL clock :	std_logic := '0';
	SIGNAL input_data :	std_logic_vector(1 downto 0) := (others=>'0');

	--Outputs
	SIGNAL modulated_signal :	std_logic_vector (3 downto 0);

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: modulator PORT MAP(
		reset => reset,
		clock => clock,
		input_data => input_data,
		modulated_signal => modulated_signal
	);
			clock<= not clock after 10 ns;
		reset<='0' after 5 ns;

	tb : PROCESS
	  variable text_line: line;
  variable i: std_logic_vector(1 downto 0);
  file my_file: TEXT open read_mode is "input.txt";
  file out_file:text open write_mode is "output.txt";

begin
while not endfile(my_file) loop
   readline (my_file,text_line);
   read(text_line,i);
   input_data<=i;

   write (text_line,modulated_signal);
   writeline (out_file,text_line);
   wait until clock='0';

end loop;
wait;
end process;
END;
```

Demodulator:

```
-- Create Date:      12:17:52 07/22/2006
-- Design Name:      demodulator
-- Module Name:      testbench.vhd
-- Project Name:    demodulator
-- VHDL Test Bench Created by ISE for module: demodulator

-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.    Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
library std;
use std.textio.all;
use ieee.std_logic_textio.all;
ENTITY testbench_vhd IS
END testbench_vhd;

ARCHITECTURE behavior OF testbench_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT demodulator
    PORT(
        modulated_signal : IN std_logic_vector(3 downto 0);
        clock : IN std_logic;
        reset : IN std_logic;
        original_data : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;

    --Inputs
    SIGNAL clock :    std_logic := '0';
    SIGNAL reset :    std_logic := '1';
    SIGNAL modulated_signal : std_logic_vector(3 downto 0) := (others=>'0');
```

```
	--Outputs
	SIGNAL original_data :	std_logic_vector(1 downto 0);

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: demodulator PORT MAP(
		modulated_signal => modulated_signal,
		clock => clock,
		reset => reset,
		original_data => original_data
	);

	  clock<= not clock after 10 ns;
		reset<='0' after 5 ns;
	tb : PROCESS
	  variable text_line: line;
 variable i: std_logic_vector(3 downto 0);
 file my_file: TEXT is in "input.txt";
 file out_file:TEXT is out "output.txt";

begin
while not endfile(my_file) loop
   readline (my_file,text_line);
   read(text_line,i);
   modulated_signal<=i;
   write (text_line,original_data);
   writeline (out_file,text_line);

   wait until clock='0';
 end loop;

wait;
	END PROCESS;

END;
```

Appendix 7: Synthesis report of modulator and demodulator

Modulator:

```
Release 8.1i - xst I.24
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.55 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.55 s | Elapsed : 0.00 / 1.00 s

--> Reading design: modulator.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) HDL Analysis
  4) HDL Synthesis
     4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
     5.1) Advanced HDL Synthesis Report
  6) Low Level Synthesis
  7) Final Report

=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "modulator.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "modulator"
Output Format                      : NGC
Target Device                      : Automotive CoolRunner2

---- Source Options
Top Module Name                    : modulator
Automatic FSM Extraction           : YES
```

FSM Encoding Algorithm : Auto
Mux Extraction : YES
Resource Sharing : YES

---- Target Options
Add IO Buffers : YES
MACRO Preserve : YES
XOR Preserve : YES
Equivalent register Removal : YES

---- General Options
Optimization Goal : Speed
Optimization Effort : 1
Keep Hierarchy : YES
RTL Output : Yes
Hierarchy Separator : /
Bus Delimiter : <>
Case Specifier : maintain

---- Other Options
lso : modulator.lso
verilog2001 : YES
safe_implementation : No
wysiwyg : NO

===

===
* HDL Compilation *
===
Compiling vhdl file "D:/modulator/modulator.vhd" in Library work.
Entity <modulator> compiled.
Entity <modulator> (Architecture <behavioral>) compiled.

===
* HDL Analysis *
===
Analyzing Entity <modulator> (Architecture <behavioral>).

```
Entity <modulator> analyzed. Unit <modulator> generated.

=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <modulator>.
    Related source file is "D:/modulator/modulator.vhd".
    Found 5x3-bit ROM for signal <temp>.
    Found 4x3-bit ROM for signal <$n0009> created at line 83.
WARNING:Xst:737 - Found 3-bit latch for signal <next_state>.
INFO:Xst:2371 - HDL ADVISOR - Logic functions respectively driving the data and gate
enable inputs of this latch share common terms. This situation will potentially lead to
setup/hold violations and, as a result, to simulation problems. This situation may come from
an incomplete case statement (all selector values are not covered). You should carefuly review
if it was in your intentions to describe such a latch.
    Found 3-bit 5-to-1 multiplexer for signal <$n0007>.
    Found 3-bit register for signal <current_state>.
    Summary:
    inferred     2 ROM(s).
Unit <modulator> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 2
  4x3-bit ROM                                          : 1
  5x3-bit ROM                                          : 1
# Registers                                            : 1
  3-bit register                                       : 1
# Latches                                              : 1
  3-bit latch                                          : 1
# Multiplexers                                         : 1
  3-bit 5-to-1 multiplexer                             : 1

=========================================================================
```

```
=========================================================================
=======
*                      Advanced HDL Synthesis                           *
=========================================================================
=======

=========================================================================
=======
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 2
  4x3-bit ROM                                          : 1
  5x3-bit ROM                                          : 1
# Registers                                            : 3
  Flip-Flops                                           : 3
# Latches                                              : 1
  3-bit latch                                          : 1
# Multiplexers                                         : 1
  3-bit 5-to-1 multiplexer                             : 1

=========================================================================
=======

=========================================================================
=======
*                      Low Level Synthesis                              *
=========================================================================
=======

Optimizing unit <modulator> ...
    implementation constraint: INIT=r   : current_state_2
    implementation constraint: INIT=r   : current_state_0
    implementation constraint: INIT=r   : current_state_1

=========================================================================
=======
*                      Final Report                                     *
=========================================================================
=======
Final Results
RTL Top Level Output File Name         : modulator.ngr
Top Level Output File Name             : modulator
```

```
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : YES
Target Technology                  : Automotive CoolRunner2
Macro Preserve                     : YES
XOR Preserve                       : YES
wysiwyg                            : NO

Design Statistics
# IOs                              : 8

Cell Usage :
# BELS                             : 73
#      AND2                        : 21
#      AND3                        : 5
#      INV                         : 34
#      OR2                         : 12
#      XOR2                        : 1
# FlipFlops/Latches                : 6
#      FDC                         : 3
#      LD                          : 3
# IO Buffers                       : 8
#      IBUF                        : 4
#      OBUF                        : 4
=========================================================================
CPU : 8.91 / 9.48 s | Elapsed : 9.00 / 10.00 s

-->

Total memory usage is 85488 kilobytes

Number of errors    :    0 (   0 filtered)
Number of warnings  :    1 (   0 filtered)
Number of infos     :    1 (   0 filtered)
```

Demodulator:

Release 8.1i - xst I.24

--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.18 s | Elapsed : 0.00 / 1.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.18 s | Elapsed : 0.00 / 1.00 s

--> Reading design: demodulator.prj

TABLE OF CONTENTS


```
=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "demodulator.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "demodulator"
Output Format                      : NGC
Target Device                      : Automotive CoolRunner2

---- Source Options
Top Module Name                    : demodulator
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Mux Extraction                     : YES
```

```
Resource Sharing                    : YES

---- Target Options
Add IO Buffers                      : YES
MACRO Preserve                      : YES
XOR Preserve                        : YES
Equivalent register Removal         : YES

---- General Options
Optimization Goal                   : Speed
Optimization Effort                 : 1
Keep Hierarchy                      : YES
RTL Output                          : Yes
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : maintain

---- Other Options
lso                                 : demodulator.lso
verilog2001                         : YES
safe_implementation                 : No
wysiwyg                             : NO

=========================================================================
=======


=========================================================================
=======
*                          HDL Compilation                              *
=========================================================================
=======
Compiling vhdl file "D:/demodulator/demodulator.vhd" in Library work.
Entity <demodulator> compiled.
Entity <demodulator> (Architecture <behavioral>) compiled.

=========================================================================
=======
*                          HDL Analysis                                 *
=========================================================================
=======
Analyzing Entity <demodulator> (Architecture <behavioral>).
Entity <demodulator> analyzed. Unit <demodulator> generated.
```

```
=========================================================================
=======
*                            HDL Synthesis                              *
=========================================================================
======

Synthesizing Unit <demodulator>.
    Related source file is "D:/demodulator/demodulator.vhd".
    Found 5x3-bit ROM for signal <temp>.
WARNING:Xst:737 - Found 3-bit latch for signal <next_state>.
INFO:Xst:2371 - HDL ADVISOR - Logic functions respectively driving the data and gate
enable inputs of this latch share common terms. This situation will potentially lead to
setup/hold violations and, as a result, to simulation problems. This situation may come from
an incomplete case statement (all selector values are not covered). You should carefuly review
if it was in your intentions to describe such a latch.
    Found 3-bit 5-to-1 multiplexer for signal <$n0008>.
    Found 3-bit register for signal <current_state>.
    Summary:
    inferred     1 ROM(s).
Unit <demodulator> synthesized.


=========================================================================
=======
HDL Synthesis Report

Macro Statistics
# ROMs                                          : 1
 5x3-bit ROM                                    : 1
# Registers                                     : 1
 3-bit register                                 : 1
# Latches                                       : 1
 3-bit latch                                    : 1
# Multiplexers                                  : 1
 3-bit 5-to-1 multiplexer                       : 1

=========================================================================
=======


=========================================================================
=======
*                         Advanced HDL Synthesis                        *
=========================================================================
```

```
=======

=========================================================================
=======
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 1
  5x3-bit ROM                                          : 1
# Registers                                            : 3
  Flip-Flops                                           : 3
# Latches                                              : 1
  3-bit latch                                          : 1
# Multiplexers                                         : 1
  3-bit 5-to-1 multiplexer                             : 1

=========================================================================
=======

=========================================================================
=======
*                      Low Level Synthesis                              *
=========================================================================
=======

Optimizing unit <demodulator> ...
    implementation constraint: INIT=r   : current_state_2
    implementation constraint: INIT=r   : current_state_1
    implementation constraint: INIT=r   : current_state_0

=========================================================================
=======
*                      Final Report                                     *
=========================================================================
=======
Final Results
RTL Top Level Output File Name     : demodulator.ngr
Top Level Output File Name         : demodulator
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : YES
Target Technology                  : Automotive CoolRunner2
Macro Preserve                     : YES
```

```
XOR Preserve                        : YES
wysiwyg                             : NO

Design Statistics
# IOs                               : 8

Cell Usage :
# BELS                              : 109
#      AND2                         : 18
#      AND3                         : 16
#      AND4                         : 1
#      INV                          : 51
#      OR2                          : 20
#      OR3                          : 1
#      XOR2                         : 2
# FlipFlops/Latches                 : 6
#      FDC                          : 3
#      LD                           : 3
# IO Buffers                        : 8
#      IBUF                         : 6
#      OBUF                         : 2
=========================================================================
=======
CPU : 8.08 / 8.28 s | Elapsed : 8.00 / 9.00 s

-->

Total memory usage is 85488 kilobytes

Number of errors    :    0 (   0 filtered)
Number of warnings  :    1 (   0 filtered)
Number of infos     :    1 (   0 filtered)
```

Appendix 8: Gantt chart

No.	Task name	Duration	Start	End	Status	Weeks
1	Meet with supervisor	1 day	03/03/2006	03/03/2006	Completed	
2	Study the background knowledge of the project	14 days	04/03/2006	17/03/2006	Completed	
3	Project proposal	21 days	01/04/2006	22/04/2006	Completed	
4	Preparation of proposal presentation	14 days	01/05/2006	15/05/2006	Completed	
5	QPSK modulator and demodulator design in Matlab code	21 days	01/06/2006	21/06/2006	Completed	
6	QPSK modulator design in VHDL code	28 days	22/06/2006	20/07/2006	Completed	
7	QPSK demodulator design in VHDL code	7 days	21/07/2006	28/07/2006	Completed	
8	Result check	4 days	28/07/2006	31/07/2006	Completed	
9	Project report	30 days	01/08/2006	29/08/2006	Completed	
10	Preparation of Poster	5 days	30/08/2006	3/09/2006	Completed	