



University of Tehran
College of Engineering
School of Electrical & Computer Engineering

Experiment 34
Sessions 8,9,10

Digital Modulation

Digital Logic Laboratory
ECE 045
Laboratory Manual

Fall 1403

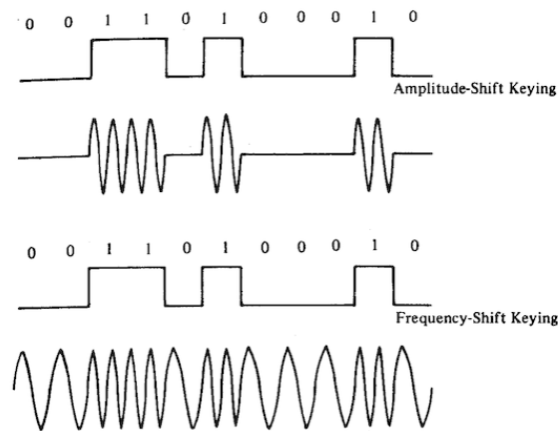
Spring 04



Contents

Contents	1
1 Introduction	2
2 Direct Digital Synthesis (DDS)	3
3 Frequency Divider	5
4 Message Process	6
5 Digital to Analog conversion using PWM	7
6 The Total design	9
Acknowledgment	10

Figure 1: FSK and ASK Modulation



1 Introduction

Modulation is a fundamental process in communication systems that enables the transmission of information or data over long distances. The original data (message signal) is often too weak to be transmitted directly through a channel. To address this, it is combined with a high-frequency, strong signal called the carrier signal, which acts as a medium to carry the message signal, such as sound, images, or data.

Modulation is essential in various applications, including radio broadcasting, television, mobile networks, and satellite communications. It is broadly classified into two types based on the nature of the message signal: analog modulation and digital modulation.

In digital modulation, there are four primary techniques:

1. Amplitude Shift Keying (ASK)
2. Frequency Shift Keying (FSK)
3. Phase Shift Keying (PSK)
4. Quadrature Amplitude Modulation (QAM)

In this context, we aim to design ASK and FSK modulation systems. figure 1 illustrates the modulated signals produced by ASK and FSK modulation for transmission through a channel.

As depicted, the message signal is binary, while the carrier signal is a high-frequency sinusoidal wave capable of supporting various frequencies. In this example, the message signal represents Morse code, which encodes text characters into sequences of dots (represented as binary 0) and dashes (represented as binary 1) for message transmission. figure 2 provides a visualization of Morse code sequences. In the modulation system a DAC is also used to convert the digital output to an analog signal, which can be observed and evaluated via an oscilloscope.

Figure 2: Morse code

A	• —	N	— •	1	• — — — —
B	— • • •	O	— — —	2	• • — — —
C	— • — •	P	• — — •	3	• • • — —
D	— • •	Q	— — • —	4	• • • • —
E	•	R	• — •	5	• • • • •
F	• • — •	S	• • •	6	— • • • •
G	— — •	T	—	7	— — • • •
H	• • • •	U	• • —	8	— — — • •
I	• •	V	• • • —	9	— — — — •
J	• — — —	W	• — —	0	— — — — —
K	— • —	X	— • • —		
L	• — • •	Y	— • — —		
M	— —	Z	— — • •		

Accordingly, the following topics are covered in detail in this experiment:

- Sine Wave Generator for the carrier signal
- Frequency Selector for supporting various frequencies in the sine wave
- Message Processing for the message signal
- Digital to Analog Converter (DAC)

By the end of this experiment, you should have learned:

- The fundamentals of digital modulation
- How to generate a sine waveform with various frequencies
- How to utilize ROM memory in your design
- Schematic design in Quartus II

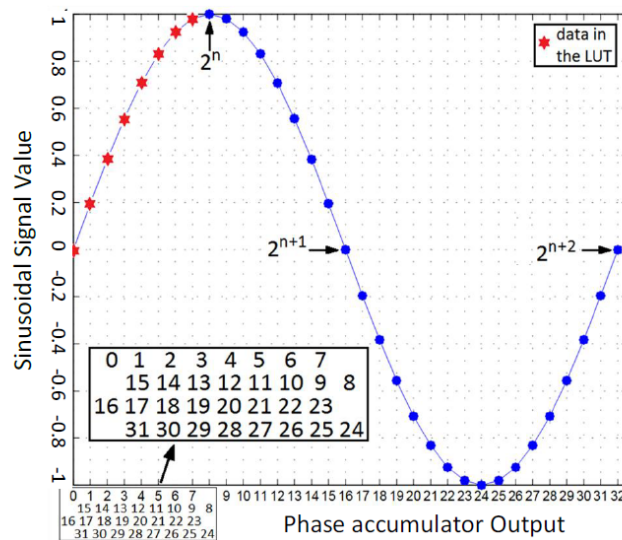
2 Direct Digital Synthesis (DDS)

Most modern function generators use Direct Digital Synthesis (DDS) for generating their output waveforms. DDS is used for generating arbitrary phase tunable output from a single fixed-frequency reference clock (like an oscillator). Here we use DDS to generate a sine wave for carrier signal.

The output of the DDS module is a quantized version of the output files (usually a sinusoid). Figure 4 shows the hardware design of the DDS module. To generate the DDS signal, you should use a 1-port ROM memory to store the value of a sine wave for several clock cycles. This ROM module has 64 entries and only contains one quadrant of the sine wave. You will receive a file named `sine.mif` that is used for ROM initialization.

A phase accumulator module, PA, can generate the address location of this memory. The PA will

Figure 3: Phase accumulator output generation



enumerate all the data point positions of the entire period of the waveform. Two bits of the PA output denoted as "*signbit*" and "*phasepos*", are used to indicate the quadrant information. The next 6 bits of the PA output, denoted as "*addr*", are fed to the ROM address input. Figure 3 shows how 2^{n+2} data points are generated from the 2^n entries stored in the ROM.

Signed and magnitude numbers are used to represent the sinusoidal signal waveform. The "*phasepos*" bit of the PA output is used to select either "*addr*" or its 2's complementary code as the ROM address. The former case represents the first and third quadrants of the waveform and the latter corresponds to the second and the fourth quadrants. When "*phasepos* = 1" and "*addr* = 0", the corresponding data points are the positive and negative peaks of the sinusoidal waveform, whose magnitude value is not stored in the ROM. The all-zero-detection circuit and the following AND-gate are used to detect such cases. The output of the AND gate guides the output multiplexer to select either the ROM output or the digital code "11..11" as the signal magnitude accordingly.

The clock signal for the DDS module must have a stable frequency. In this section, the 50 MHz clock frequency of the FPGA is used. However, in the complete design, since various frequencies are required, this clock will be replaced by the divided clock generated by a ring oscillator in the subsequent sections.

1. Write the Verilog code for the DDS (Direct Digital Synthesizer) module as shown in figure 4.
 - For the simulation step in ModelSim, use the \$readmemb command within the initial block to read the memory content from the sine.mem file.
 - For the implementation step in Quartus, utilize the memory blocks available in FPGA resources. Use the memory content stored in the sine.mif file. To achieve this, you can configure the FPGA to use either FPGA memory blocks or memory logic. figure 5 demonstrates how to use the romstyle keyword for this purpose.

Figure 4: DDS hardware design

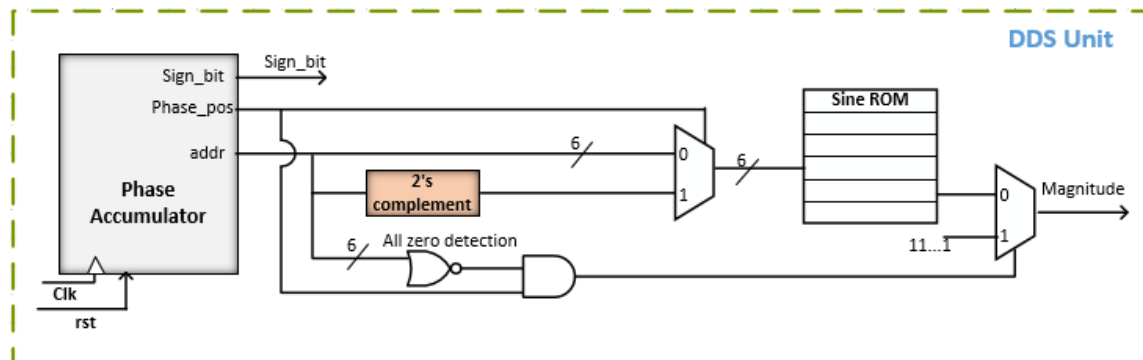


Figure 5: FPGA memory block specification

```
(* romstyle = "M9K" *) (* ram_init_file = "sine.mif" *) reg [7:0] LUT[0:63];
```

2. After completing the waveform generator block diagram in Quartus, synthesize the final design and include the synthesis summary in your report.
3. Before continuing, it is better to test your design in Modelsim. Therefore, write a simple testbench and verify the functionality of your design. For this part use the 50-MHz clock frequency.

3 Frequency Divider

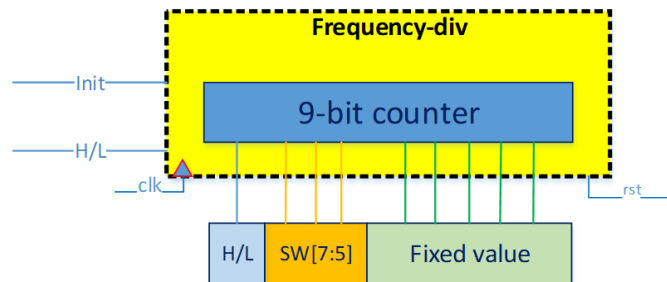
To generate various frequencies for the carrier signal, a frequency selector is required. The frequency selector uses a counter to divide a high-frequency input signal into the desired frequency. For this purpose, you can implement a simple 9-bit counter in Verilog, as shown in figure 6. To load the division value, you will need an external load signal. One of the FPGA push buttons can be used to perform this function.

It is important to note that a sampling frequency is required for capturing the message signal. Additionally, FSK modulation requires two distinct frequencies. To address this, two frequency dividers are instantiated:

- One frequency divider drives the clock of the message processing unit and provides the higher frequency for FSK modulation.
- The second frequency divider generates the lower frequency corresponding to the 0 value in the message signal for FSK modulation.

Inputs for selecting the appropriate frequency are applied to both units. The output of the frequency divider is used to drive the clock of the Direct Digital Synthesizer (DDS) based on the current message in FSK modulation. After adding the frequency divider to the design, synthesize your design.

Figure 6: Block diagram of frequency divider



1. Since two frequencies are needed for FSK, and one is half the other, the MSB (Most Significant Bit) of one frequency will be 0, while the MSB of the other will be 1. The next three MSBs are set using **SW[7:5]** to define the parallel load values for the frequency divider. Set the five least significant bits of the counter to a fixed value in your code. Use the push button (**KEY0**) to load the frequency divider.
2. Modify the testbench from Section 1 to verify the design for different desired frequencies. To achieve this, change the parallel load values (**SW[7:5]**) for at least three different frequencies and observe the results.
3. Include the waveforms of each desired frequency in your report while mentioning the achieved frequencies, the input parallel loads, and expected frequencies.

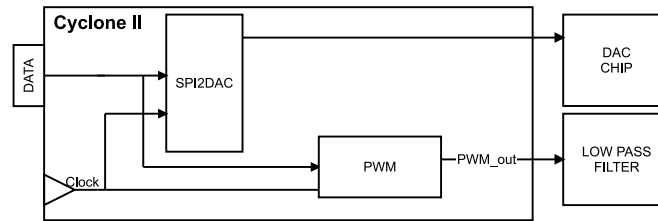
4 Message Process

To transmit a binary message signal, *Morse code* is used. Each character of the alphabet and each number is assigned a set of 0 and 1 values. The Morse code for each letter is transmitted through the channel, with the maximum size of the Morse code being 5 bits. Therefore, the message signal occupies 5 bits as the message input. A tag value, **0101**, is added at the beginning of each letter to distinguish them. This tag has no meaning in Morse code.

The "send" input indicates when a letter has been sent. When the "send" signal is active, the message is transmitted. Each bit of the message lasts for 4 periods of the sine wave (each sine wave requires $4 * 64$ time steps). Since there are 9 bits for each message (4 bits for the tag value + 5 bits for the data), two counters and a shift register are used to capture the message. The shift register helps identify the current bit of the message, which aids in selecting the correct modulation at the appropriate time, as determined by the counters.

1. Design the controller and the datapath of message process unit.
2. Use push button (**KEY1**) for send input and **SW[4:0]** for message input.

Figure 7: Block diagram of Digital to Analog Converter



5 Digital to Analog conversion using PWM

There are different methods for digital to analog conversion. You can either use an external chip like an add-on-board card that consists of both ADC and DAC or it can be implemented using Pulse Width Modulation (PWM). Figure 7 shows the DAC circuit. As can be seen for realizing an external DAC chip a serial to parallel interface is required between the FPGA board and the chip. In this experiment, you will use the second method to have a digital to analog conversion. The following is a brief description of how PWM works as a DAC.

A PWM signal is a sequence of periods in which the duration of the logic-high (or logic-low) voltage varies according to external conditions, and these variations can be used to transmit information. The PWM carrier frequency is constant, so the active state duration increases or decreases and vice versa. The duty cycle of the PWM signal is equal to:

$$duty\ cycle = \frac{T_{on}}{T_{on} + T_{off}}$$

The nominal DAC voltage observed at the output of the low-pass filter is determined by just two parameters, namely, the duty cycle and the PWM signal's logic-high voltage; in the figure 8, A denotes this logic-high voltage for “amplitude.” The relationship between duty cycle, amplitude, and nominal DAC voltage is fairly intuitive: In the frequency domain, a low-pass filter suppresses higher-frequency components of an input signal. The time-domain equivalent of this effect is smoothing, or averaging. Thus, by low-pass filtering a PWM signal, we are extracting its average value.

In this experiment, the period of PWM is fixed to 256 clocks and its pulse width is the value on the 8-bit input of the module. Figure 8 shows a sample PWM wave. For the first PW number of clock cycles, the output value is 1 and it is 0 for the rest of the cycles.

1. Write a Verilog code for the DAC module. The output of the PWM module will go to an external board with an RC low pass filter. The values of the resistor and capacitor are 1K ohm and 10 nF respectively.
2. You can use random data input from the switches to test the accuracy of your design.
3. Connect the PWM module to the design in the previous section, synthesize the design, and program the FPGA.
4. Observe the functions on the Oscilloscope and include them in your reports.

Figure 8: Pulse Width Modulation (PWM)

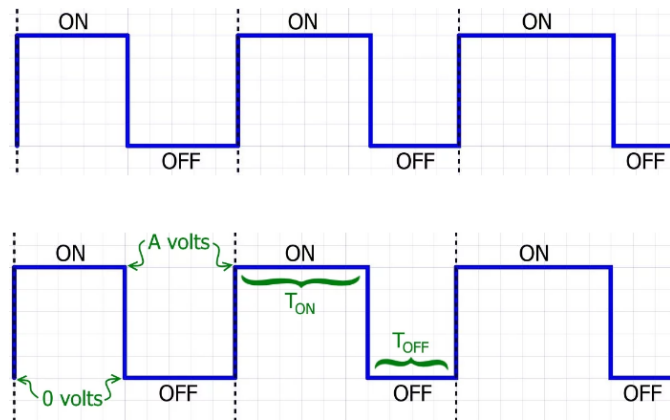


Figure 9: Total design

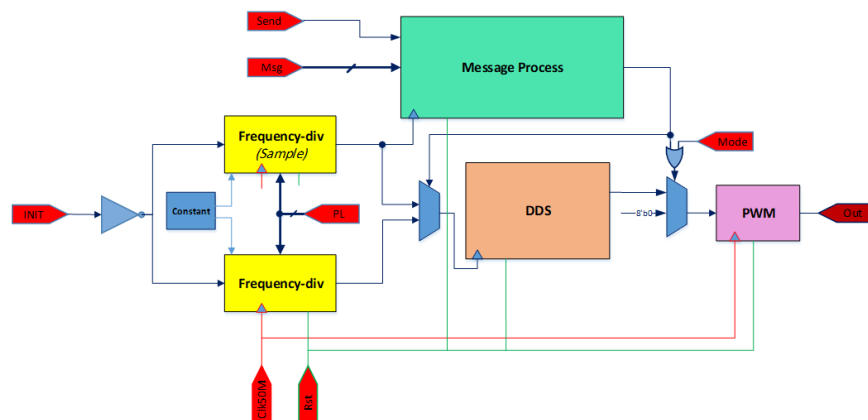
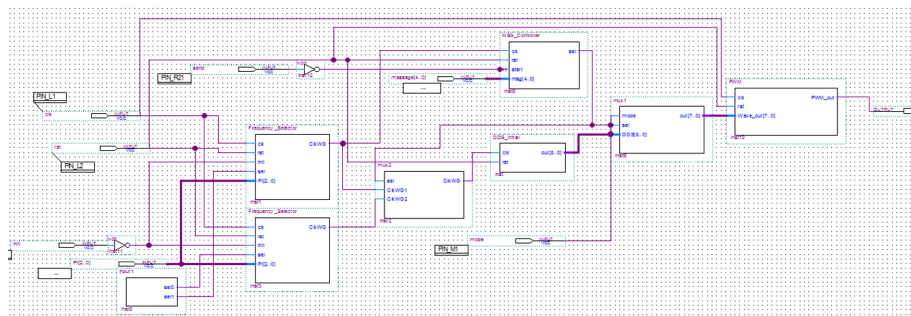


Figure 10: Final block diagram in Quartus II

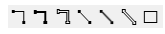


6 The Total design

Once all the components are designed and verified, as shown in figure 9, you can integrate them into a schematic design, as illustrated in figure 10. Note that the mode input is controlled by **SW[8]**, which selects the modulation type (FSK or ASK). Additionally, the design requires a reset input, assigned to **SW[9]**. Both the frequency divider and PWM modules need the 50 MHz clock from the FPGA.

Quartus II has this capability to generate symbols for any HDL code with the corresponding inputs and outputs.

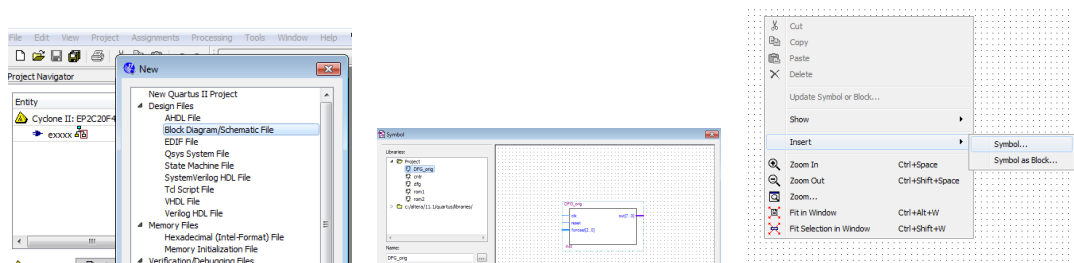
1. Click on *File* ▷ *New Project Wizard*.
2. Create a good directory for your project and complete the form as in the previous experiments.
3. In the *ADD/Remove Files in Project* page add all the Verilog code you have written.
4. By right-clicking on a Verilog code, you can create a symbol for it. If the symbol generation is done successfully, a file with the suffix **.bdf** will be generated.

You should use bus or line tool from the top toolbar menu  and make all the interconnects between components.

Double-click on the blank space in the Graphic Editor window, or click on the icon in the toolbar that looks like an AND gate. A pop-up box will appear.

Expand the hierarchy in the Libraries box. First, expand libraries, then expand the library primitives, followed by expanding the library logic comprising the logic gates. Use the gates if any is needed. When the schematic design is completed, compile it as before. Program the design on FPGA and record all the results.

Figure 11: Quartus II



Acknowledgment

This lab manual was prepared and developed by **Katayoon Basharkhah**, Ph.D. student of Digital Systems at the University of Tehran, under the supervision of Professor Zain Navabi.

This manual has been revised and edited by **Zahra Jahanpeima**, **Zahra Mahdavi** Ph.D. student of Digital Systems at the University of Tehran, **Mohammad Yahyapour**, and **Moeen Farhadi** undergraduate students of Electrical Engineering at the University of Tehran.