Lastname: ____________________

5Digit SS: ____________________

LABORATORY
Introduction to
MatLab M-File Scripts

LAB M-FILE: INTRODUCTION TO MATLAB M-FILE SCRIPTS (10 points)

In this lab you will be introduced to the MatLab M-file coding. MatLab is a powerful DSP interpreter, that allows you quickly and efficiently develop LUT and test bench data for your FPGA design. We will in this lab extend our function generator from Lab 1 with additional test functions and we will also write M-file scripts to test a complex multiplier design that needs 3 real multiplications and 5 add operations.

In the **pre-lab** you will compute with "pencil-and-paper" the results you later expect in your design implementation. In the **design part** you will design a function generator for 4 different functions and a complex multiplier.

Lab Objectives

After completing this lab you should be able to

- Write simple M-file scripts to define tables and to use predefined functions
- Use the MatLab help, demo and function library
- Design and simulate complex multiplier systems using Simulink

Pre-lab (3 points)

Note that MatLab can be accessed from any computer in the college using the X-Win32 program on dingo.

1. MatLab works like a very powerful pocket calculator and has a wide selection of predefined functions. You can easily define data, and process, manipulate, or plot these data. Using the MatLab prompt, determine the result of the following MatLab instructions:

MatLab instruction	Short description
a=zeros(3,5);	
Help fir1	
lookfor convolution	
Xpsound	
x=0:63;	
y=sin(2*pi*x/64);	
v=100*rand(1,64);	
w=floor(x/16);	
F=fft(y);	
plot(abs(F))	
whos	

2. A complex multiplication of the type $(x+jy)(c+js)=xc-sy+j(xs+yc)$=Re+jIm is a frequently used object in the DFT computation discussed in later labs. The complex multiplication in the direct form requires 4 real multiplications and 2 add operations. From an implementation standpoint, the 4 multipliers realized as array multipliers are the most resource intensive objects. Algorithms that

use 3 multipliers and 5 adders are usually more resource effective. Such a algorithm works as follows:

Compute first: $a_1=c+s$, $a_2=c-s$, and $a_3=x-y$
then compute $m_1=c*a_3$, $m_2=y*a_2$, and $m_3=x*a_1$,
and finally: $re=m_2+m_1$ and $im=m_3-m_1$

As can be seen above, the multiplier uses 3+2=5 add and 3 multiply operations. Develop a short MatLab script that computes the output for 8 bit data and coefficient quantization by completing the following M-Files:

```
%% m file to check the complex multiplier with 3*5+
x=60; y=40;
w=exp(j*pi/5);
c=round(real(w)*128);
s=round(imag(w)*128);
%%% add the equation for a1,a2,a3,m1,m2,m3, re and im here:
```

```
%%% display the results
check=(x+j*y)*w;
str= sprintf('result = %5d+j%5d => check: %5.3f+j%5.3f',...
   floor(re/128),floor(im/128),real(check),imag(check));
disp(str);
```

Save the file under the name `cmtest.m` on your **DSPwFPGAs** directory. To run the file, type its name (without the .m extension) in the MatLab prompt. Make sure to use dir *.m and cd instructions so that the MatLab prompt points to the directory where the M-file was placed. Now, use the script to complete the following test data table:

x+jy	w=exp(jφ)=(c+js)	(x+jy)*w	c*128	s*128	(c+s)*128	(c-s)128
60+j40	φ=π/5					
70+j50	φ=π/9					
60+j40	φ=-π/9					
70-j50	φ=-π/5					

Simulink Design-lab (8 points)

Follow the directions below to implement the 4 function generator and the complex multiplier.

A. Getting Started

If you are in room B114 or the digital logic lab:

1. On the desktop, double click on the **Engineering Folder**.
2. Double click on the MatLab icon to start MatLab. It will take a few seconds to load.
3. From the top icon list in the **MatLab** window click on the **Simulink** icon to start **Simulink.**
4. Use your **DSPwFPGAs** folder to save your designs. **Never** save your files to the local drive, use your **network drive** or a **USB drive** instead.

B. Design the Function generator systems

1. Download the file **funcgen.m** from the class webpage into your **DSPwFPGAs** folder.
2. For convenience, in **MatLab,** click on the "Current Directory" selection icon ... and select your **DSPwFPGAs** folder as the current directory.
3. The files in the **DSPwFPGAs** folder will be visible in the upper left **MatLab** window. Double click on the **funcgen.m** file to open it. After a moment, you should see the incomplete M-file:

```
%% m file to compute the 4 functions in the ML lab
x=0:2^6-1;                   %% all have length 64
y0= zeros(1,64);             %% first is the sine funtion
y1= zeros(1,64);             %% second is the multi step function
y2= zeros(1,64);             %% uniform random numbers
y3= zeros(1,64);             %% symmetric triangular


plot(x,y0,'k-',x,y1,'k--',x,y2,'k.',x,y3,'k-.');
legend('sine','multi step','random','triangular')
xlabel('Sample');ylabel('Amplitude');title('4 Functions generator')
axis([0 63 0 1.1*2^8]);
print -djpeg funcgen.jpg

LUT=[y0, y1, y2, y3];
```

4. Complete your M-file so that the required 4 functions, y0, y1, y2 and y4, are plotted as shown in Fig. 1. Note that the range is [0,2^8-1] unsigned.

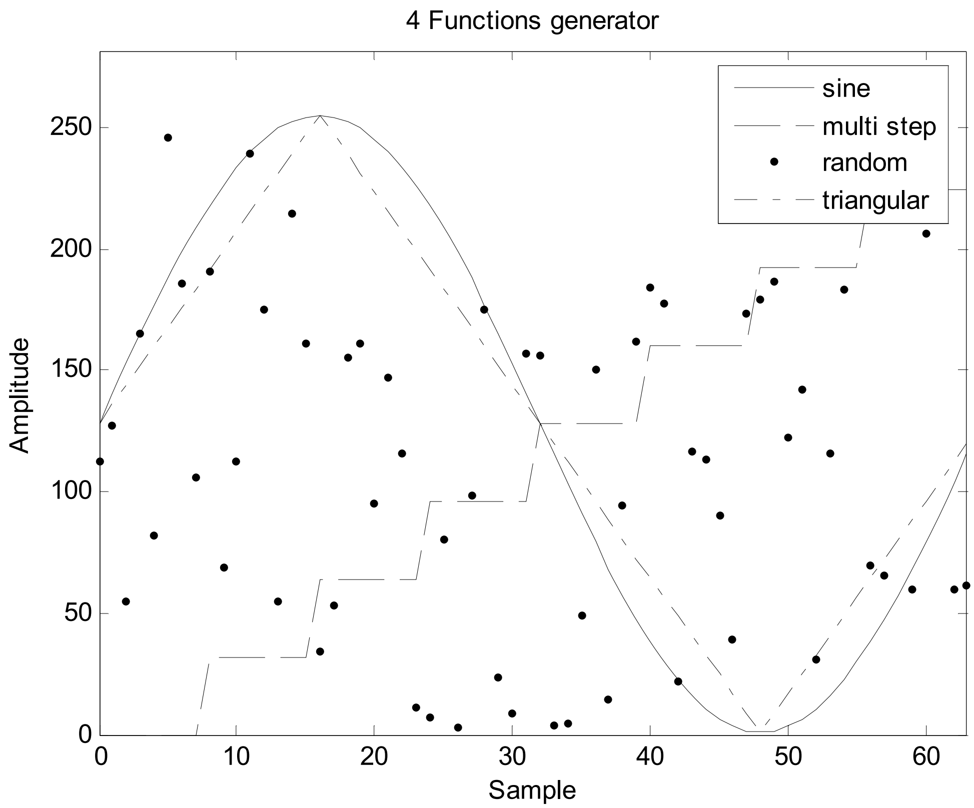

Fig.1: Plot of 4 different functions.

5. From the **Simulink Library Browser** window, open the design you created for lab 1 and save it under a different name, **lab4.mdl**. **Rename** the sine table block to **LUT table**.

6. **Rearrange** your design to match the changes show in **Fig.2**. Note that the **accumulator's input** is no longer connected to the switches but to a **constant value**. Also, the **LUT** is fed by the **concatenation** of the input from the **switches** and the **sliced accumulator output**.

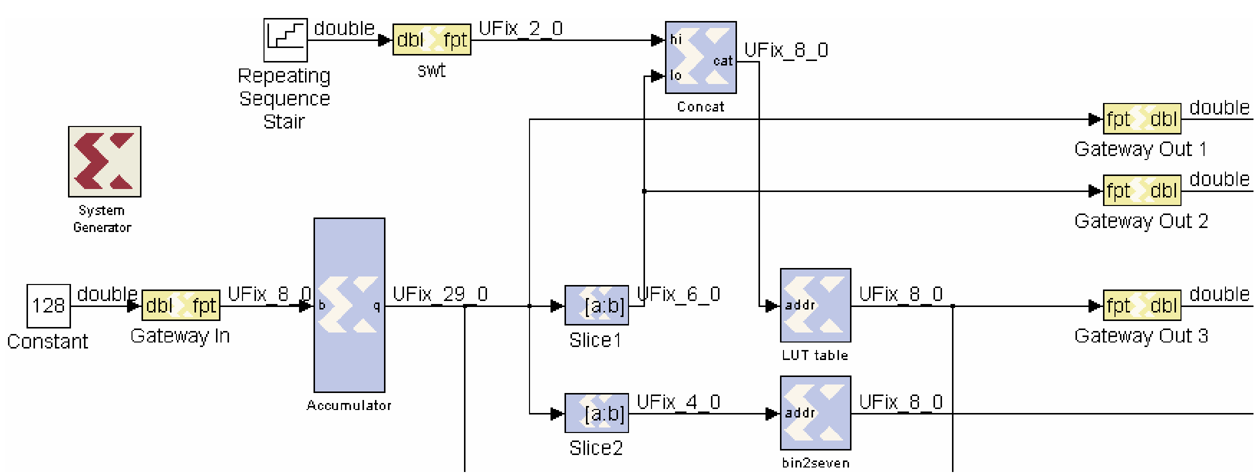

Fig.2: Changes made to lab1.mdl

7. If you haven't done so already, replace the **coeffs** Simulink block by a **Repeating Sequence Stair block**.

8. Double click on the **Gateway In** input block and change the **Output Data Type** to **unsigned**. Otherwise, the constant value 128 would be read as a negative value.

9. Double click on the **swt** input and change the **number of bits** to **2.** Since only four functions are present, only 2 switches (2 bits) are necessary.

10. Double click on the slice that feeds the **Concat** block, **Slice1**, and change the **width** of the slice to **6** bits. After concatenation, the width of the input to the look up table must still be 8 bits.

11. Double click the **LUT table** block and, under **Initial Value Vector**, enter your new variable **LUT**. This way, the **values** for the look up table are **generated** by the array LUT created by your **MatLab m file script.**

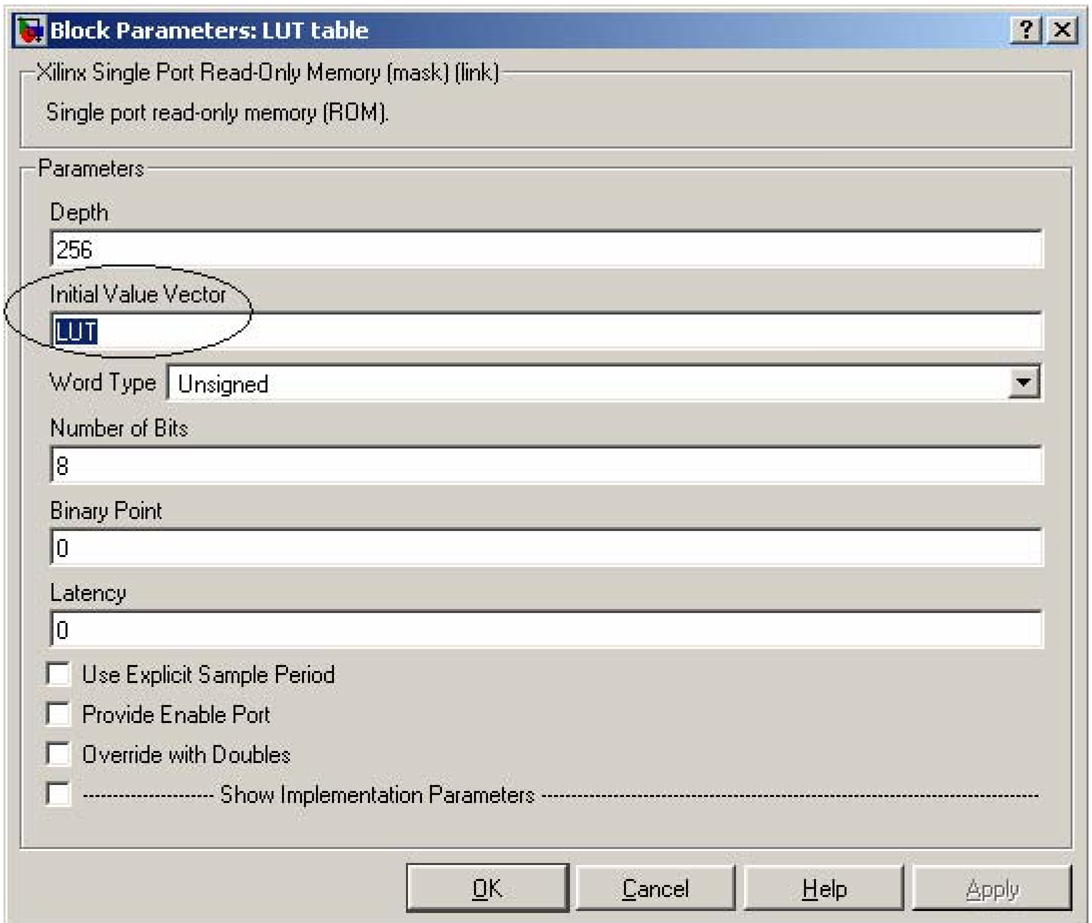

Fig.3: Block parameters for the LUT

C. Completion of the Function Generator design

1. Double click on the **Repeating Sequence Stair** block and enter **0 1 2 3** under **Vector of output values** and 512 under **Sample time**.

2. Change the Simulation **stop time** to **2048** so that only **one set of functions** is viewed. **Simulate** your design and **verify** that the **output** of the scope shows the **4 functions**. **Print** out the **simulation waveform**.

3. Next, double-click on the **SignalGenerator** block and create the design's project files.

4. Open **Xilinx Project Navigator** and generate the **Programming file** for your project.

5. Determine the number of required 4-input lookup tables (LUTs), Multipliers and the number of **block RAMs** from **View Design Summary**, and the maximum frequency from the **Post Place and Route Static Timing Report** under **Detailed Reports**.

4 input LUTs	=	____________________
Block RAMs	=	____________________
Multipliers 18x18s	=	____________________
Max. Freq.	=	____________________

D. Completion of the Complex multiplier Design

1. Create a new model and complete the design of the complex multiplier to match **Fig. 4**.

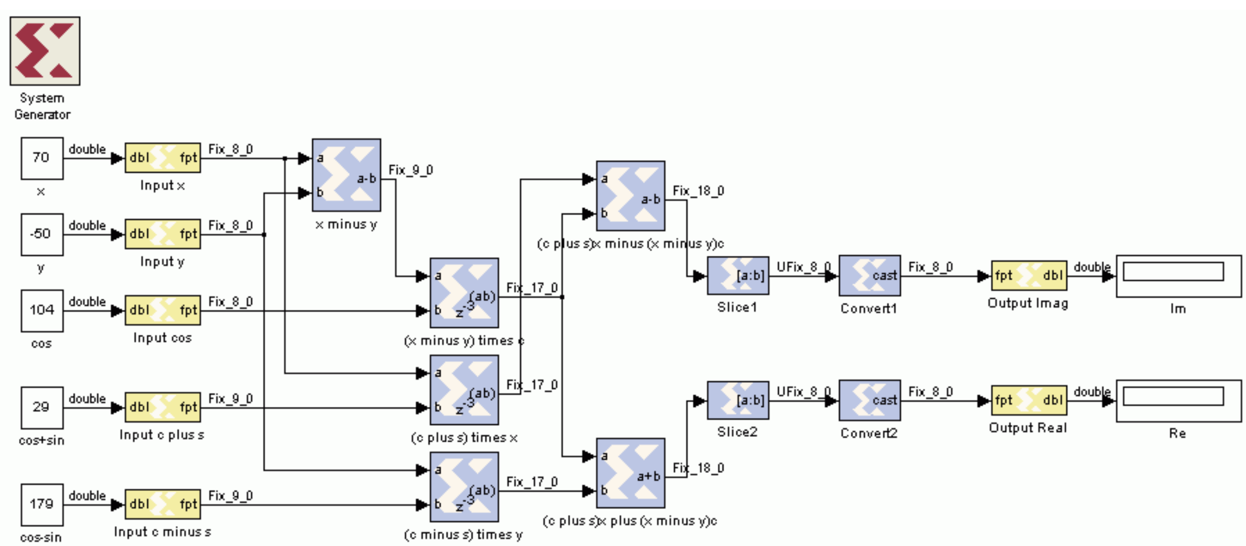

Fig. 4: Complex multiplier using three real multiplications.

2. **Configure** the **slice** blocks to “select” bits **7 through 14**. To do so, you will have to **specify the range** as **two bit locations**.

3. The convert blocks are only used to return the real and imaginary outputs to signed values. **Configure** the **convert** blocks to output **signed** values. Do not change the number of bits

4. **Simulate** your design and **verify** that the real and imaginary **outputs** match the 4 different **test data** computed in the pre lab. Use the complex multiplier you designed to complete the following table:

x+jy	w=exp(jφ)=(c+js)	Re	Im
60+j40	$\varphi=\pi/5$		
70+j50	$\varphi=\pi/9$		
60+j40	$\varphi=-\pi/9$		
70-j50	$\varphi=-\pi/5$		

5. Next, double-click on the **SignalGenerator** block and create the design’s project files.

6. Open **Xilinx Project Navigator** and generate the **Programming file** for your project.

7. Determine the number of required 4-input lookup tables (LUTs), Block RAMs, and Multipliers from **View Design Summary** and the maximum frequency from the **Post Place and Route Static Timing Report** under **Detailed Reports**.

4 input LUTs = ____________________

Block RAMs =____________________

Multipliers 18x18s =____________________

Max. Freq. = ____________________

F. Deliverables:

1. Solve the problems of the pre-lab. (3 points).

2. Print the MDF files and the Simulink simulation (7 points).

Make sure your name and SS is on all pages you turn in!