ELEC3342: Digital System Designs
First Semester, 2018–19

Homework 3 (r1.0)

Due:

- Part (A) -- 29 Nov, 2018, 11:59pm
- Part (B) -- 29 Nov, 2018, 11:59pm

Instruction: Submit your answers electronically through Moodle.

There are 2 major parts in this homework. Part A includes questions that aim to help you with understanding the lecture materials. They resemble the kind of questions you will encounter in quizzes and the final exam. Some of these questions may also ask you to implement and test small designs in VHDL. This part of homework must be completed individually.

Part B of this homework contains a mini-project that you should work in groups of 2. Your submitted work will be graded by an auto tester and therefore you should make sure your submitted files conform to the required format.

The following summarize the 2 parts.

Part	Type	Indv/Grp
A	Basic problem set	Individual
B	Mini-project	Group of 2

In all cases, you are encouraged to discuss the homework problems offline or online using Piazza. However, you should not ask for or give out solution directly as that defeat the idea of having homework exercise. Giving out answers or copying answers directly will likely constitute an act of plagiarism, which is a serious offense.

Part A: Problem Set

A.1 Registered Circuit

Figure A.1 shows a circuit that computes a registered four-input XOR function. Each two-input XOR gate has a propagation delay of 100 ps and a contamination delay of 55 ps. Each flip-flop has a setup time of 60 ps, a hold time of 20 ps, a clock-to-Q maximum delay of 70 ps, and a clock-to-Q minimum delay of 50 ps. Determine the following:

(a) If there is no clock skew, what is the maximum operating frequency of the circuit?

(b) How much clock skew can the circuit tolerate if it must operate at 2 GHz?

(c) How much clock skew can the circuit tolerate before it might experience a hold time violation?

(d) Your project partner points out that she can redesign the combinational logic between the registers to be faster and tolerate more clock skew. Her improved circuit also uses three two-input XORs, but they are arranged differently. What is her circuit? What is its maximum frequency if there is no clock skew? How much clock skew can the circuit tolerate before it might experience a hold time violation?

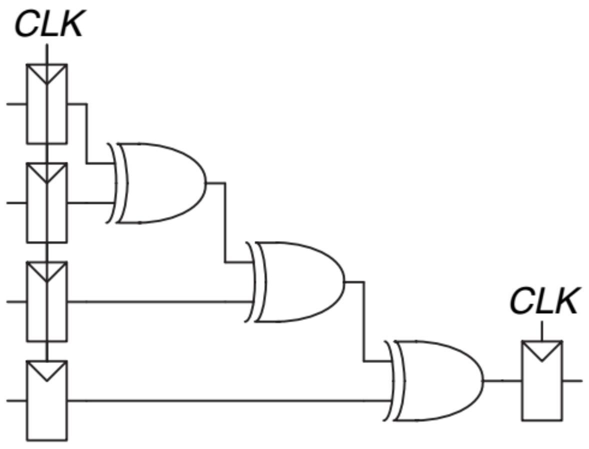

Figure A.1: A registered 4-input XOR gate

A.2 Circuit Pipelining

In theory, any feedforward circuits can be sped up by pipelining the datapath. For instance, Figure A.2 shows two circuits that perform $Y = A - (B + C)$. The only difference is that the circuit on the right has an additional pipeline stage and thus incurs an additional cycle of latency.

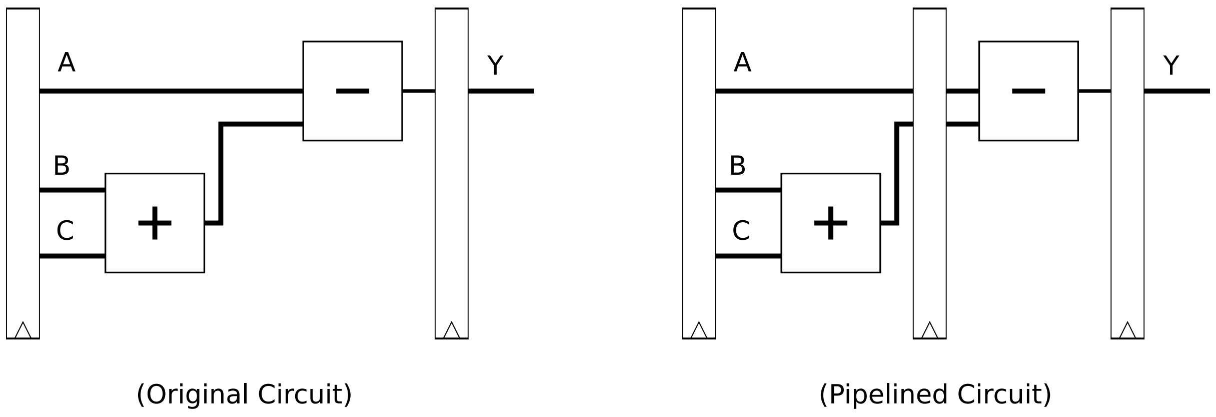

Figure A.2: Pipelining a feedforward circuit.

Let the setup and hold time of each register to be t_{setup} and t_{hold} respectively. The register has a clock-to-Q propagation delay t_{pcq} and a clock-to-Q contamination delay of t_{ccq}. Also, let the propagation and contamination delay of an adder/subtractor is $t_{pd,add}$ and $t_{cd,add}$ respectively.

A.2.1 Determine the maximum clock period that the circuit in Figure A.2 may run at without violating any of the setup and hold time constraints.

A.2.2 You are given the task to design a circuit that adds 1024 numbers. Assume the numbers, $a_0, a_1, \ldots, a_{1023}$, are all available and stored in their respective registers, $R_0, R_1, \ldots, R_{1024}$ in the beginning of computation. The result should be stored in an output register R_s.

Design a circuit that:

1. runs with the highest clock frequency f_1
2. completes adding all numbers with minimum number of clock cycles (in terms of $1/f_1$)

A.2.3 Assume instead that on every cycle, only 4 input numbers are available. That is:

$$\begin{aligned} \text{cycle0} &: a_0, a_1, a_2, a_3 \\ \text{cycle1} &: a_4, a_5, a_6, a_7 \\ &\vdots \end{aligned}$$

Given the above data arrival rate, design a circuit that:

1. runs with the highest clock frequency f_2
2. completes adding all numbers with minimum number of clock cycles (in terms of $1/f_2$)

Compare to your answer in Question A.2.2, how are the maximum clock frequencies (f_1 vs f_2) and the latency affected?

A.2.4 What are the data processing throughput of the above 2 circuits? You should express your answer in terms of number of data processed per second.

Part B: Mini-Project

Overview

In this homework, you will complete your design for the Morse Code decoder. Built upon the results from homework 1 and 2, there are 2 main additional tasks required in this homework for you to have a complete working decoder:

1. Implement and test your design from Homework 2 in actual FPGA hardware;
2. Connect the microphone module to the FPGA board and obtain audio signal as input to your decoder.

You are highly encouraged to tackle the 2 tasks as 2 milestones. Achieve the first milestone will give you a fully functional Morse code decoder. Achieving the second milestone will allow your Morse code decoder to receive input from actual real-world input.

B.1 Milestone 1: Implementation on FPGA

In this part of the project, your task is to implement your design from homework 2 on the actual FPGA board. For your reference, the board you will be using is the Digilent Basys 3 board. The main FPGA on your board is a Xilinx Artix-7 FPGA (XC7A35T- 1CPG236C). For details of the board, please refer to the Basys 3 FPGA Board Reference Manual. (Note: on Digilent website the link is labeled as Datasheet.)

A sample project, called `testdecoder` is provided to you with all of the necessary additional circuits and configurations correctly setup to test your designs from homeowkr 2. Figure **??** shows a top-level block diagram for the `testdecoder` project. Your task for this part is to simply test your design on the FPGA board.

When compared to your design from homework 2, note the following:

B.1.1 Simulation vs Synthesis In homework 2, you tested your design by simulation only. The benefit for simulation only testing is that you may change the testbench (or input to your circuit) easily in software. However, the drawback of a simulation-based testing methodology is that you cannot test any real-world circuit behaviors. For example, you need the actual FPGA to test the UART connection to your host computer.

One important benefit of an FPGA is that it is reprogrammable. Therefore, one way to test your final design is to include additional testing/testbench circuits in your design. Furthermore, you can add/remove these circuits as your design mature, and you may choose to test only part of your final design at a time.

In the `testdecoder` project, an additional testbench generator is created for exactly that purpose. It is a simple module with a large ROM that stores a long sequence of 1s and 0s (the d_{bin} signal) that mimics the signals that would have been generated from the audio processing circuits. *This additional testbench*

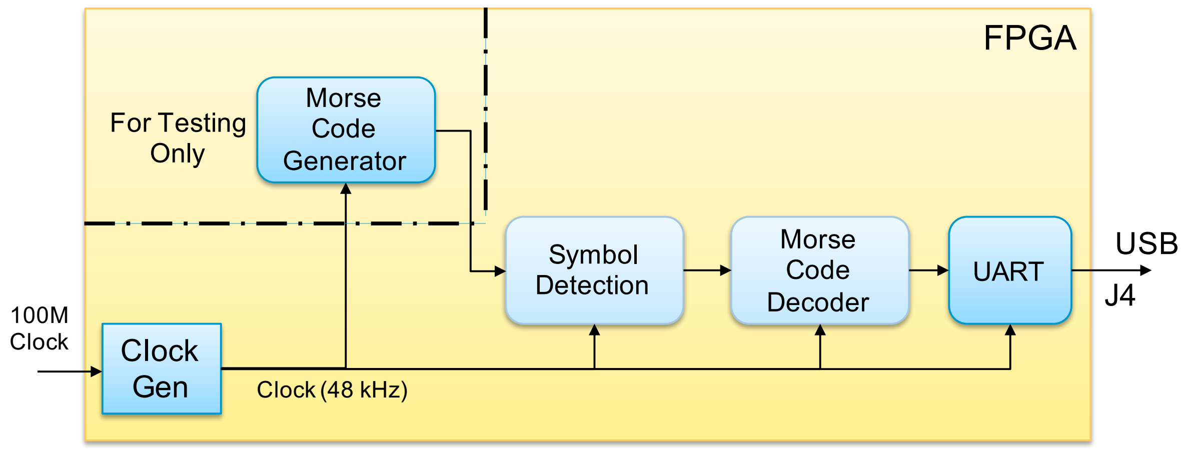

Figure B.1: A project to test Morse code decoder *without* the actual audio module.

modules thus allow you to test your Morse code decoder without, and independent of the audio module. You will replace this testbench modules with actual audio input in next part.

B.1.2 Clock An important part of a real project, as oppose to a simulation project, is that you need to run with an actual clock signal. On the Basys 3 board, a 100 MHz clock source is available and you will use that as the main clock course in your design.

In your design, you will need 2 clock signals to drive different parts of the circuits:

- `clk_6144`: A 6.144 MHz clock used in the audio ADC controller;
- `clk_48k`: A 48 kHz clock used for the main design.

For those of you who are curious, the above clock frequencies are chosen specifically for ease-of-implementation in your design: 48 kHz is a common audio processing sampling rate, and is also 5 times higher than the 9600 baud rate needed for UART communication. 6.144 MHz is 128 times higher than the main clock frequency, and can be precisely produced from the 100 MHz clock source with the FPGA's on-board multi-mode clock management (MMCM) circuit.

The circuits required to produced these clock signals are already provided to you. Feel free to explore how they work but DO NOT modify any of these circuits.

B.1.3 Implement Design Implement your design by replacing the 2 empty files (`symdet` and `mcdecoder`) with your implementation from homework 1 and 2. You can either copy-and-paste the files from within Vivado, or simply overwrite the vhd files in the project with your own version. Note, a *functional* UART has been provided to you. You may use your own UART if you want to.

Once you have all the files set up, clock the "Run Synthesis" button to start the synthesis process. If your synthesis runs correctly (check for Error and Warning messages), you should also run the "Generate Bitstream" step of the Vivado tool flow.

B.1.4 Configure FPGA To test your design on the Basys 3 FPGA board, do the following:

- Connect your Basys 3 board to your computer with a micro-USB cable. Connect your cable to connector **J4** (labeled as **PROG** on the board).
- Turn on the FPGA board. By default you will see numbers showing up on the 7-segment display and increases every second. It is the default designs shipped from the factory.

- To configure the FPGA, select "**Open Hardware Manager**" from Vivado. Your FPGA board should show up under the **Hardware** pane.
- Right click on the FPGA showed up (Artix-7) and select "**Program Device**...". Select the bitstream that you have just created (usually with the same name as your project with a `.bit` extension.)

B.1.5 Serial Terminal In order to receive data from your Basys 3 board, you need to run a serial terminal program on your host computer. Make sure you set up your serial terminal with 9600 baudrate and 8-N-1 configuration. Refer to homework 2 handout for more details.

The default configuration for the Basys 3 board will send a text string to the host computer when it first starts up through its UART. You should use this as a test of your serial connection before you test your own design.

If you set up is correct, when the Basys 3 starts up you should see a text string show on your computer screen.

B.1.6 Testing Morse Code Decoder The `testdecoder` project is setup such that it sends a Morse code sequence every time you press the middle button on the Basys 3 board. This sequence of the binary encoded Morse code is read from the testbench ROM and send to your `symdet` module as if it is received from the audio circuit.

Your `symdet` and `mcdecoder` modules will decode this sequence, and push the decoded alphabet to the UART module. The UART module will send this message to your screen.

Congratulation! If you can see the correctly decoded message on your computer screen, your Morse code decoder's decoding logic is completed!

B.2 Milestone 2a: Audio Input

Once you have a working Morse code decoder, your next task is to setup an input to feed this decoder. In this part, you will implement a controller to read from the externally connected audio module. The audio module you will used is the Digilent Pmod MIC3 modules, which contains a Knowles Acoustics SPA2410LR5H-B microphone and a Texas Instruments ADCS7476 analog-to-digital converter.

Your task is thus to implement a controller that controls and reads data from the ADC.

B.2.1 Conencting to the ADC The Pmod MIC3 reference manual from Digilent contains detailed information on how to communicate with the module. See the reference manual at `https://reference.digilentinc.com/reference/pmod/pmodmic3/reference-manual`.

In short, the ADCS7476 ADC uses the simple SPI protocol to communicate with the FPGA. In short, your FPGA design should provide (drive) the `SS` and `SCK` signals. The data from the ADC will then be sent back to the FPGA in the `MISO` signal. These SPI signals are connected to the ADC7476 with the following mapping:

SPI Signal	ADCS7476 Signal
`SS`	$\overline{\texttt{CS}}$
`SCK`	`SCLK`
`MISO`	`SDATA`

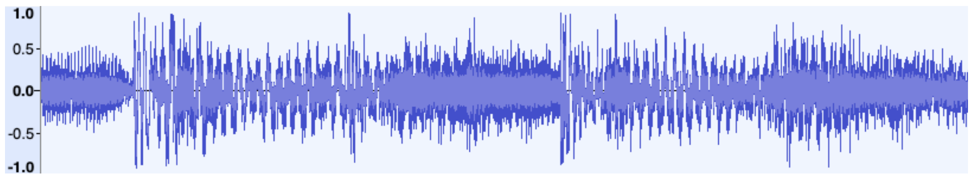

Figure B.2: A typical audio waveform.

B.2.2 Reading from ADC To read 1 data word (12 bit) from the ADC, you need to follow the timing diagram shown in the MIC3 reference manual, which is the same as Figure 2 from the ADCS7476 datasheet. In words the process involves roughly the following events:

1. Assert `SS` (= $\overline{\texttt{CS}}$). That is, set it to 0.
2. Wait a small delay t_2.
3. Pulsing `SCK` (= `SCLK`).
4. Each bit of data will appear in `MISO` (= `SDATA`) *after* each *falling* edge of `SCLK`.
5. Deassert $\overline{\texttt{CS}}$ after 16 `SCLK` cycles.

You *may* keep `SCLK` free running as long as you $\overline{\texttt{CS}}$ signal is correctly synchronized to `SCLK`, as illustrated by the dotted line.

From the ADCS7476 datasheet, the maximum `SCLK` clock rate is $f_{\text{SCLK}} = 20\,\text{MHz}$. Since your ADC controller is clocked with `clk_6144` running at 6.144 MHz, you are recommended to produce the `SCLK` signal yourself, setting $f_{\text{SCLK}} = 3.072\,\text{MHz}$. In other words, make a circuit that uses 2 cycles to read each data bit.

B.2.3 Audio Sampling The above step describes procedure to read 1 word (12-bit) of audio sample. For audio signal processing, you will want to sample the audio input at a fixed rate. For simplicity sake, you are recommended to sample at 48 kHz. In other word, initiate the above sequence every 1/48000 seconds. These audio data sample will then become the digital audio samples that can be processed in later steps.

B.3 Milestone 2b: Audio Signal Processing

Once you can correctly sample audio, you need to perform simple processing on the audio input in order to detect any Morse code transmission.

First a very brief introduction to audio processing. Audio signal travels in the form of mechanical wave over the air that causes change in air pressure. A microphone detects such subtle change in air pressure and convert that into an analog voltage similar to one shown in Figure B.2. The properties of this wave and its corresponding voltage signal thus determines the basic quality of audio:

Wave Property	**Audio Property**
frequency	pitch
amplitude	volume
shape	timbre

Therefore, for instances, a pure square wave will sound differently from a pure sine wave (timbre). A high frequency sine wave will results in a high pitch, and a loud voice will have wave with high amplitude.

In your project, the input Morse code will be transmitted as on-off pulses of audio tones. In the simplest form, these tones are simply pure sine wave of a particular frequency. Figure B.3 shows an ideal input Morse code waveform to your circuit.

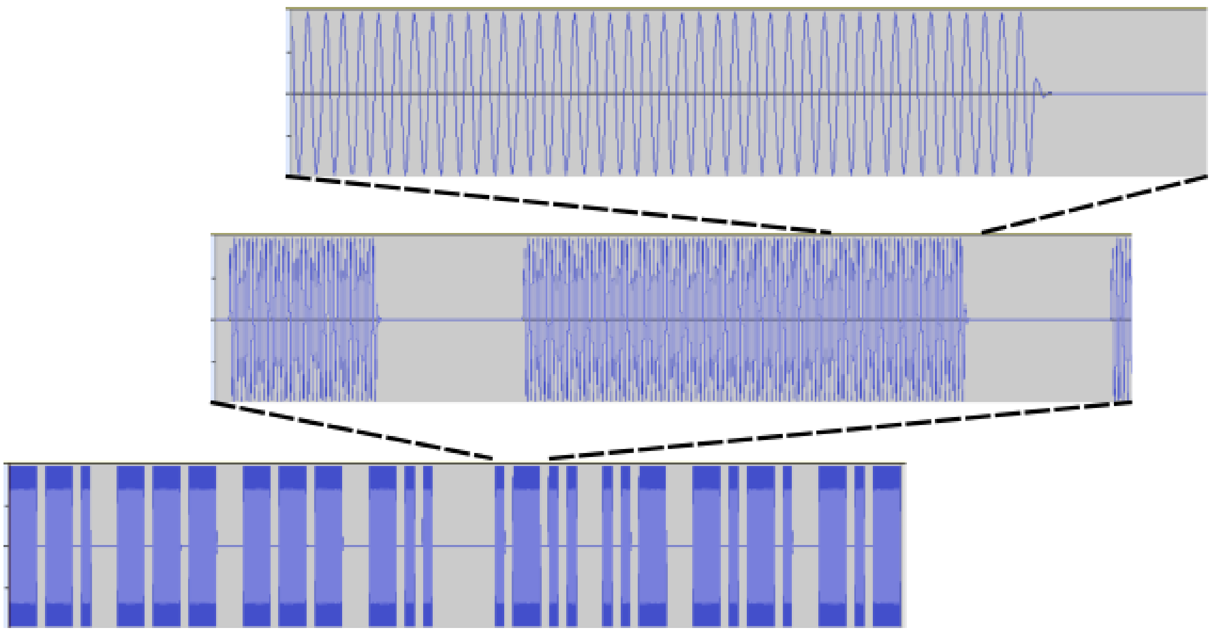

Figure B.3: Sample input waveform with Morse code transmitted. The input will be a tone around 50 Hz sine wave. The tone is pulsed on-off as Morse code.

B.3.1 Data Transmission & Encoding The sampled data from the ADC are encoded as simple binary numbers from `0000_0000_0000` to `1111_1111_1111`. A study of the MIC3 schematics show that the code `0000_0000_0000` represents the *most negative* air pressure while `1111_1111_1111` represents the *most positive* pressure. In other word, when there is no sound (0 air pressure, the reading is *around* `0111_1111_1111`. It is just an estimate as it depends on actual calibration of the circuit.

A subtle consequence of the above circuit is that the data values you are receiving are not represented in 2s complement. You will need to subtract `1000_0000_0000` from the input to obtain the true 2s complement of the data values.

B.3.2 Detecting Sound Source There are many ways you can process the audio samples to determine if there is a valid sound input. Here, a very simple thresholding scheme is shown. You are free to imagine your own scheme, or to apply sophisticated audio signal processing schemes here if you wish.

For the reommended simple scheme, the central idea is to notice that in the ideal Morse code wave form (similar to that shown in Figure B.3), all your circuit need to do is to detect if there is any sound. If there is **NO** sound, then it should output '0'. If there is **ANY** sound, it should output '1'.

Since the amplitude of a wave represents the level of sound, this simple scheme essentially performs a moving average of the input voltage amplitude over a small sampling window. If the average is above certain threshold, it regards there is audio input. If the average amplitude is below the threshold, it treats it as silence.

B.3.3 Average Amplitude Let $a[n]$ be the sample value (amplitude) at time step n. Then the moving average of the amplitude at step n over a window size of W is defined as:

$$\overline{a} = \frac{1}{W} \sum_{i=0}^{W-1} |a[n-i]| \tag{B.1}$$

For your project, set W to a suitable value that is power of 2 and you will then be able to compute (B.1) with simple add and shift.

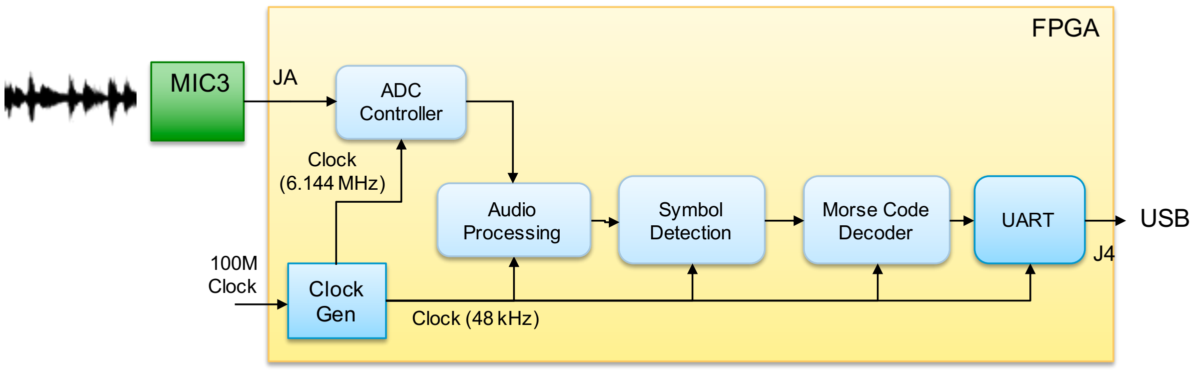

Figure B.4: Top-level block diagram of the entire design.

Note the absolute operation in (B.1). You may consider either implementing a true absolute function (which is not difficult), or you may additionally use an *approximation*:

$$|a| \approx \max(0, a) \tag{B.2}$$

You should explore different values of W or different ways to compute moving average to obtain better results.

B.3.4 Audio Processing Output The goal for the audio processing module is to produce the necessary `dbin` signal, which is the to the `symdet` module. As mentioned, you should process audio signals with the `clk_48k` clock, which results in a 1 or 0 at 48 kHz. As you will be sampling at 48 kHz, it means you simply need to determine the output every cycle, possibly based on the result of the moving average computation and a threshold that you set internally.

B.4 Combining All Parts

The final step is to combine the newly created audio processing module with the Morse code decoder that you have tested in B.1 to create a fully functional Morse code decoder. A top-level block diagram of the design is shown in Figure B.4.

To do that, simply replace the testbench module from B.1 to be the audio signal processing unit that you have created. You need to connect the MIC3 module to the FPGA and set the corresponding constraints in Vivado. Please ask on Piazza if you need help for that.

Once you combine all parts and reimplement the design, download toe board and enjoy your hard work!

B.5 Submission

For Part B of this homework, turn in, as a group, a very brief report describing your design. Include a top-level block diagram of your audio processing module. Explain the algorithm that you have implemented to perform audio detection.

Then during project demo, be prepared to show a working design. Before you attend the interview, setup the board and load the correct bitstream to the board for demo.