ELEC3342: Digital System Designs
First Semester, 2022–23

Homework 2 (r1.0)

Due:

- Part (A) -- 4 Nov, 2022, 11:59pm
- Part (B) -- 4 Nov, 2022, 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 summarizes the 2 parts.

Part	Type	Indv/Grp
A	Basic problem set	Individual
B	Mini-project	Group of 4

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 offence.

Part A: Problem Set

A.1 SR Latch

In this question, you will experiment with the design of latch circuits.

A.1.1 By means of "pushing bubbles" based off the SR latch shown in class, or otherwise, design an *active-low* SR-latch with exactly TWO NOR gates plus any number of NOT gates (bubble). When compared with the standard SR-latch shown in class, the active low SR-latch has input $\overline{S}$ and $\overline{R}$. When $\overline{S}$ is LOW, the output Q is HIGH and $\overline{Q}$ is LOW. When $\overline{R}$ is LOW, then $\overline{Q}$ is HIGH and Q is LOW.

A.1.2 To avoid having both $\overline{S}$ and $\overline{R}$ being low, one way is to introduce an additional circuit before the two input signals that avoid that condition. Design this circuit. In short, your circuit has 2 input ports $\overline{S}$ and $\overline{R}$ and produces 2 output ports $\overline{SP}$ and $\overline{RP}$. Normally, $\overline{SP} = \overline{S}$ and $\overline{RP} = \overline{R}$ except when both inputs are 0, then both output are 1.

A.2 Multimode Counter

Design a 4-bit multimode counter that supports 2 modes of operation:

(i) When `mode` is 0, the counter should increment by 2 every cycle.

(ii) When `mode` is 1, the counter should increment by 1 every cycle.

Your counter should also have a *synchronous* reset. At reset, the output should be the number 0 (`0000000000`). The counter wraps back to 0 if the value increments to larger than the maximum representable value.

Implement you multimode counter with VHDL. Submit your VHDL file as `multicounter.vhd`. Also submit a short answer to this question describing your circuit.

A.3 Variable Bit Rotation

Design a variable bit rotation circuit in this question. A rotation operation is similar to a shift operation except the bits that get rotated are shifted back to the register. Given an n-bit register with values `v(3 downto 0)`, and an input `k`, your rotator will rotate the four bits of `v` LEFT by k positions, where $0 \leq k \leq n$. Your rotator may take multiple cycles to complete the rotation and assert a `done` signal when it has finished rotating the bits. It also has a `d(3 downto 0)` signal for loading a value into `v`.

Name	Direction	Description
`d[3:0]`	input	value to be loaded into register
`k[1:0]`	input	number of positions to rotate
`ld`	input	1: load value `d`; 0: no effect
`lk`	input	1: load value `k` and start rotation; 0: no effect
`busy`	output	1: rotation in progress; 0: otherwise
`done`	output	1: asserted on the last cycle when the rotation is completed; 0: otherwise
`v[3:0]`	output	values stored in the register
`clk`	input	Clock input

Design your circuit using building blocks that we have described in class (e.g. DFF, muxes, gates, adders, counters, etc). Submit a short answer that include the top-level schematics of the circuit and describe how it works.

A.4 Look Up Table

A.4.1 Implement combinational logic functions using LUT Implement the following combinational logic functions using a 4-input LUT as shown in Figure A.1. In both cases, write down the configuration values of the LUTs.

- $Y = A\overline{B}C\overline{D} + ABCD$
- $Y = A + B\overline{D}$

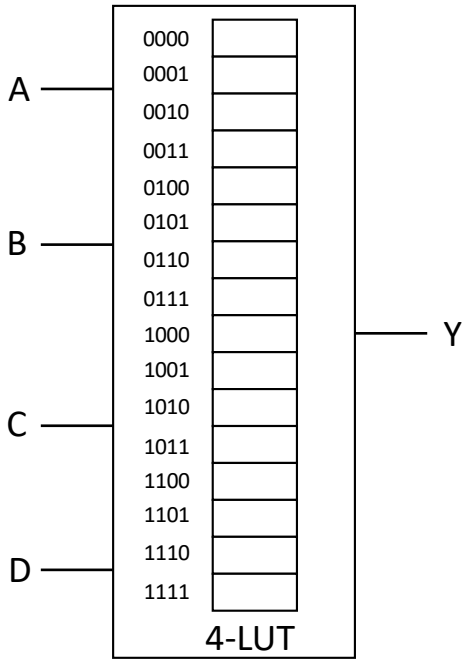


Figure A.1: Diagram of a 4-LUT

A.4.2 A 6 Input LUT Design a 6-input LUT using multiple 4-input LUTs and other combinational circuits. A 6-LUT has 6 inputs and 1 output, and can be configured to implement any 6-input combinational functions. Draw the diagram of your design and explain how your circuit implement the function of a 6-LUT.

Part B: Mini-Project

B.1 Music Code Revisit

In Homework 1, you have built a Music Code decoder that receives a symbol sequence as input and decodes the characters corresponding to the corresponding symbols (e.g. the code sequence `070732124646167070` is decoded as "LATTE"). In this homework, you will expand this design with two additional modules: one for detecting Music Code symbols and one for sending output to the computer.

Music Code: Review In Music Code, everything is being transmitted as musical note. In terms of a communication system, each of these notes is called a *symbol* and in the case of Music Code, eight symbols are defined:

Symbol	Music Note	Frequency (Hz)
0	C5	523.25
1	E5	659.25
2	G5	783.99
3	B5	987.77
4	D6	1174.66
5	F6	1396.91
6	A6	1760.00
7	C7	2093.00

In the base Music Code protocol, all notes are transmitted as *pure sine waves*. In other words, for example, the symbol 4 is transmitted as a sine wave with 1174.66 Hz over the air.

B.2 Module 1: Music Code Symbol Detection

The first module you need to design for this homework is a circuit to detect a symbol from an input wave form. The main function of this first module, called `symb_det`, can be decomposed as follows:

(i) Determine when to sample the input;

(ii) Determine the frequency of the input waveform;

(iii) Determine the symbol that is being transmitted;

(iv) Output the corresponding music codes as symbol sequences.

B.2.1 Step 1: When to Detect Symbols? The first question you need to answer when you design this symbol detector is WHEN to start detecting a symbol. Obviously, when there is no transmission, i.e., the environment is silent, then your detector should be idle too. However, when there are signals that start to arrive, then you need to determine when to make a decision to start the detection, and when to detect the following symbols.

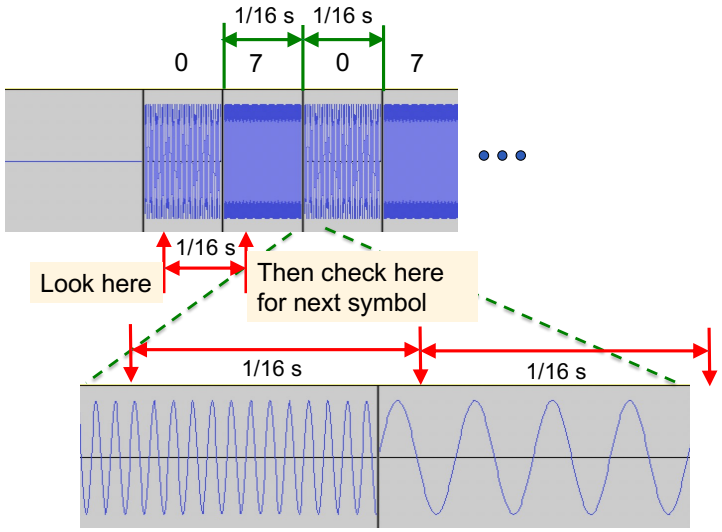


Figure B.1: An ideal waveform showing the period from idle to the first 4 symbols (0707).

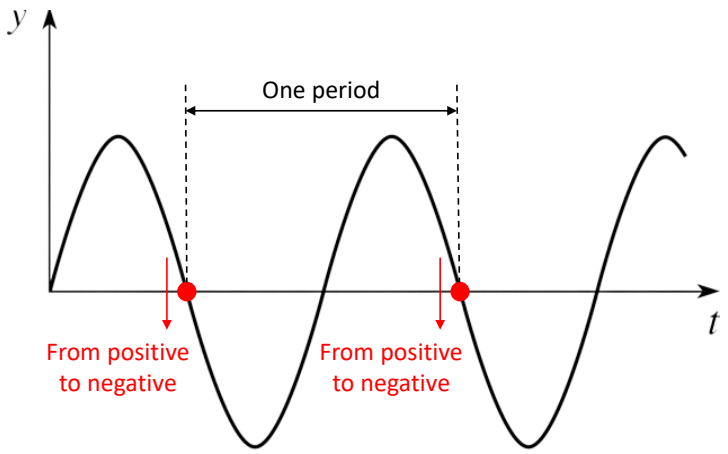


Figure B.2: Principle of zero crossing detection.

Recall that in the base Music Code protocol, the symbols are transmitted at exactly a rate of 16 symbols per second. Figure B.1 shows the waveform of the first 4 symbols. Starting with no sound, transmission of a Music Code streamlet begins with bursts of sine waves that last for 1/16 seconds each. As a result, once you have begun detecting the first symbol, you need to revisit the waveform at least every 1/16 seconds to look for the next symbol.

Hint: To increase the accuracy of your detector, you may consider delaying the detection point slightly as in Figure B.1 to avoid the period of time when the sine wave frequency changes. However, you do not want to delay too long such that you will miss the waves before the end of the 1/16 seconds period.

B.2.2 Step 2: Measuring Frequency: Zero Crossing Detection Once you are at the point to detect a symbol, your detector will then need to determine the frequency of the input sine wave at that symbol period. There are many methods to determine the frequency of an input signal. Here, we show you one of the easier method that you may consider using in your project: Since the input waveform is given as a pure sine wave, a simple way to determine its frequency is to use a method called *zero crossing detection (ZCD)*. Figure B.2 illustrates the ZCD principle.

With a pure sine wave input, we know that the signal value changes from positive to negative exactly once per period of the wave. Therefore, to determine the frequency of a sine wave, you can simply measure

the duration between two *consecutive zero crossings* with the *same direction*, i.e., negative-to-positive or positive-to-negative in the input signal.

In your system, the system clock frequency is set to be the same as the sample rate of the input data, i.e., a new sample of the input signal by the ADC is available to your circuit on each cycle. Therefore, you can measure the time between two zero-crossing points by counting the number of cycles between them. In this homework, the sampling rate and clock frequency are both set as

$$\text{sample rate} = f_{\text{clk}} = 96\,\text{kHz}.$$

For an input sine wave with frequency f_w, the counter value will therefore be:

$$\text{Cycle count} = \left\lceil \frac{f_{\text{clk}}}{f_w} \right\rceil$$

B.2.3 Step 3: Generate Symbols After obtaining the frequency (f_w) of input sine wave, your symbol detector should generate the corresponding symbol (music codes). Use the following table as a guide, determine the expected count number for each of the symbol:

Symbol	Frequency (Hz)	Counter Value
0	523.25	
1	659.25	
2	783.99	
3	987.77	
4	1174.66	
5	1396.91	
6	1760.00	
7	2093.00	

In real world, the count value you measure in hardware is unlikely to be exactly the value that you compute above. Many real world effects, such as imprecise transmission, noise or echo from the environment, measurement errors, can affect the count value you obtain above. As a result, it is possibly better if your design can accept a **range** of count values for a particular symbol to improve your design's robustness.

The detailed spec and ports of your decoder are listed in the following tables:

Port Name	Dir	Width	Description
`clk`	in	1	The main clock to your module, **96 kHz**
`clr`	in	1	When asserted 1, the entire decoder should reset to initial state regardless of the clk
`adc_data`	in	12	input sine waveform data. 12-bit will be your project ADC width. Note: signed data
`symbol_valid`	out	1	Indicate the current symbol is valid.
`symbol_out`	out	3	Generated Symbols 0-7

B.2.4 Example Run Based on our provided testbench, the waveform shown in Figure B.5 illustrates the expected behavior of your block when the output sequence is '0', '7', '0', '7', '1', '6', '3', '2', '1', '6', '1', '4', '7', '0', '7', '0'. You can also check your output sequence in the tcl console window, as shown in Figure B.3.

Note: we have provided the waveform file (`info_wave.txt`), you should use this as input when you are testing your design. Follow the steps do add the txt file into your project:

```
Note: Symbol index 0 is 0
Time: 960015 ns  Iteration: 1
Note: Symbol index 1 is 7
Time: 960015 ns  Iteration: 1
Note: Symbol index 2 is 0
Time: 960015 ns  Iteration: 1
Note: Symbol index 3 is 7
Time: 960015 ns  Iteration: 1
Note: Symbol index 4 is 1
Time: 960015 ns  Iteration: 1
Note: Symbol index 5 is 6
Time: 960015 ns  Iteration: 1
Note: Symbol index 6 is 3
```

Figure B.3: Sequence is shown in the Tcl console window

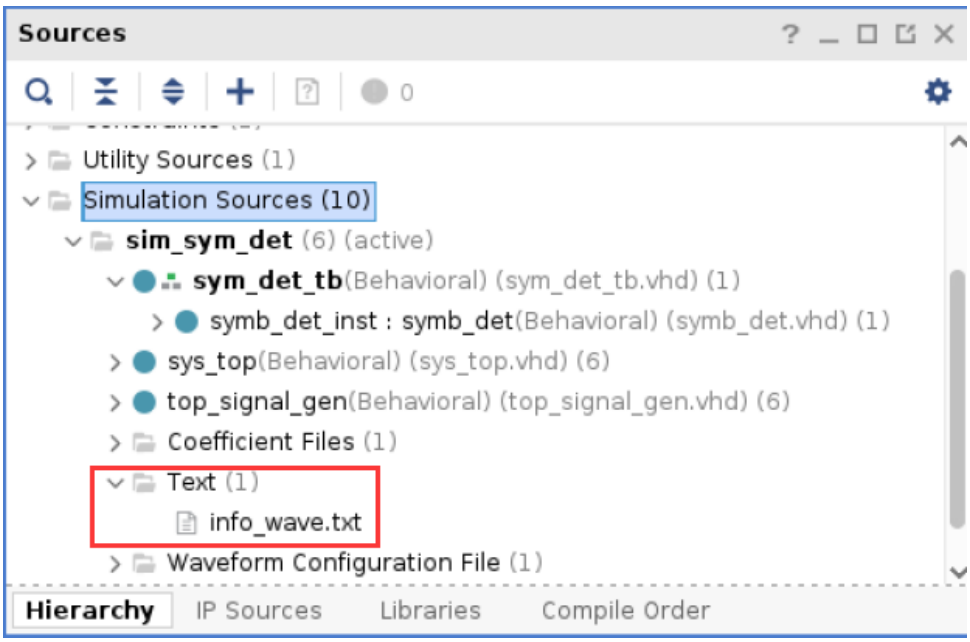


Figure B.4: Check if you have successfully added the waveform.

- Unzip the zip file of this homework, you will get `info_wave.txt`.
- Add sources → add simulation sources → add files → select `info_wave.txt`.
- After adding txt file, you can check if it exists in **Simulation Sources**, as Figure B.4 shows.
- Run simulation and check if adc_data is correct sine wave.

B.2.5 Submission For this part of homework, you should submit one file `symb_det.vhd` through Moodle. A template for that file is available on the course website, together with a sample testbench.

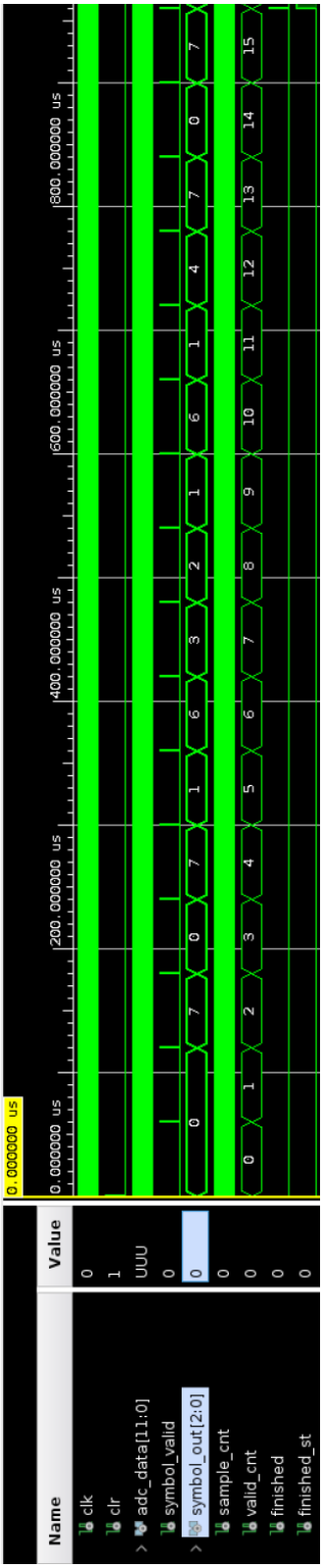


Figure B.5: Sample waveform based on the testbench. Note that each symbol_out should be only related to one symbol_valid

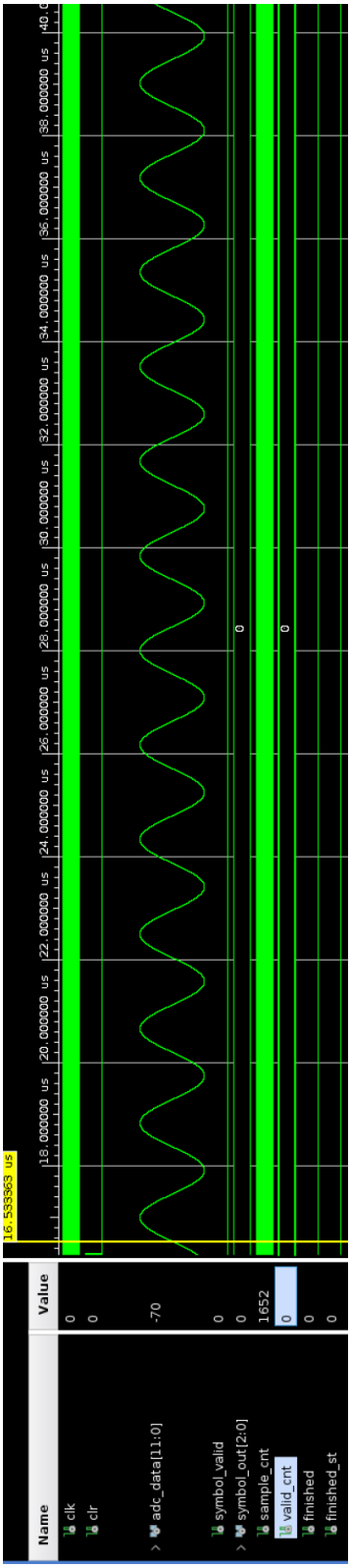


Figure B.6: When you change the radix of adc_data signal into unsigned decimal, and change its waveform style into analog, you can see some beautiful sine waves.

B.3 Module 2: UART

The second module your group needs to design is a UART for communication with the host computer. You will use this connection to display the alphabet or punctuation marks received. A basic introduction to serial communication UART can be found here: `https://learn.sparkfun.com/tutorials/serial-communication/all`

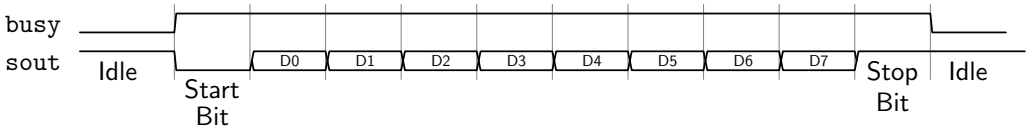
For this project, you only need the transmission (TX) part of a standard UART. Also, your UART does not need to be flexible, and should only communicate with the following configuration:

Baud Rate	9600
Data Bit	8
Parity	N
Stop Bit	1

Your UART has the following I/O ports:

Name	Direction	Description
`din[7:0]`	in	8-bit data input
`wen`	in	Asserted for 1 cycle to write data in `din` to transmit.
`sout`	out	Serial data to be transmitted.
`busy`	out	'1' when UART is busy transmitting and cannot accept new data, '0' otherwise.
`clr`	in	Asynchronous clear to initial state.
`clk`	in	Input clock at 48 kHz.

B.3.1 Principle of Operation With the above restrictions, the only task your UART needs to perform is to convert an input 8-bit data `din` into a data packet in the 8-N-1 format for serial communication output at the `sout` port. Specifically, given an 8-bit input `D[7:0]`, where `D0` is the lease significant bit (LSB) and `D7` is the most significant bit (MSB), your UART should produce the following waveform:



A few notes about the functions of UART:

- When idle, `sout` to be held at '1'
- Transmission begins by pulling `sout` to '0' for 1 baud.
- Transmission may begins after the stop bit.
- Since baud rate is 9600, the start bit, each data bit, and the stop but should be held for the entire duration of the baud, i.e, 1/9600 seconds.
- The `busy` signal should be asserted during transmission. Data pushed into `d` will be discarded when `busy` is asserted.
- Input clock is 96 kHz. Like in the symbol detector, you also need a counter to control the 9600 baud rate based on 96kHz clock (1/10).

B.3.2 Example Run Based on our provided testbench, the waveform shown in fig x illustrates the expected behavior of your block when the five input data are "01001011", "00011101", "11010110", "10101010", "00101111". You can check yourself by comparing the output serial `sout` and the input `din`, according to the waveform diagram in this sectiion. We also added a signal `baud_clk` in the testbench for you to align the bit changing in `sout`.

B.3.3 Submission For this part of homework, you should submit one file `myuart.vhd` through Moodle. A template for that file is available on the course website, together with a sample testbench.

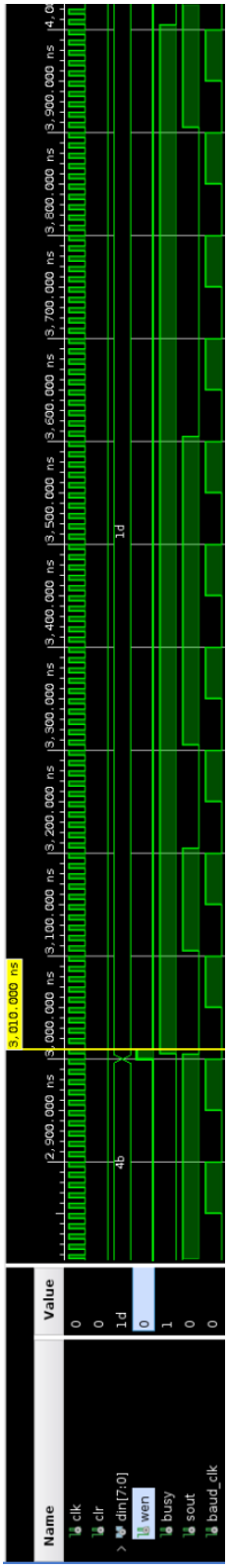


Figure B.7: Sample waveform based on the testbench (zoomed in).