Infrared Digital Timer

CS 272 Midterm Project – Fall 2019

Due: Wednesday, November 4, 2019 at 12:05pm

1 Introduction

Please read this entire document. It is lengthy and contains many details – but most of these details are meant to make this project easier. Save yourself some headache and read this first.

This is the midterm project for CS 272. You will be given *some* time in class to work on this project, but you are expected to mostly work on the project outside of the class time. The lab is available with swipe card access using your student ID after hours. This project must be done individually.

The purpose of the project is to use some of the circuit techniques we've worked on so far this semester, practice writing microcontroller programs, and integrating circuits with the microcontroller to make a (mostly pointless) gadget. The focus of this particular project is on the timing needed to communicate a value with another microcontroller, and to use a simple digital display to view the communicated value.

In this project you will develop a system that will act as a remote controlled countdown timer. There will be two parts to the system, the transmitter and the receiver. Your task will be to implement the receiver, and I will provide the transmitter. The remote control transmitter will allow you to pick a time between 0:01 and 9:59 minutes, then the time will be sent to the receiver at the press of a button. Your microcontroller will act as the receiver. When you receive a new time value, you should display the time (minute only) on a seven segment display, and then start counting time down, keeping the display up-to-date with the countdown. When the time reaches 0:00, then your microcontroller should display an "LED alarm".

You are responsible for making sure all voltage, current, and power dissipation values in your circuit are within the rated specifications of the various circuit elements. You can reference your previous work in lectures and labs for examples of each circuit required for this project, and for calculations needed to determine circuit component values.

It is highly advised to also split the microcontroller program into C++ classes. One class type could be used to represent the digit, another could be used to encapsulate the logic for handling the IR receiver, etc. It is up to you to divide the code into meaningful parts.

The due date for this project is **Monday, November 4th, 2019 at 12:05pm**. Full submission of the project consists of four pieces: 1) a demonstration of your system to me, 2) submitting the circuit diagrams **and** the calculations that you used to prove that circuit elements are within their rated capacities, 3) submitting the Teensy source code, including any C++ classes, and 4) leaving your breadboard assembled after demonstration so that I can ensure that each circuit is connected as intended (I'll give it back after grading). Late projects cannot be accepted. It is in your best interest to start this project as soon as possible. All of the material that you need to know to complete this project can be found in lecture topics 1 through 12 (up through BJTs).

2 Project Specification

This project requires a microcontroller program and circuit that is able to communicate with another microcontroller. Communications often involve accurately timing the sending and receiving of a series of high and low voltages on wires which vary over time. The number of signals, and the pattern of high/low pulses is collectively referred to as the signaling *protocol*. In this project, we will use an IR LED/BJT pair to communicate the high/low pulses and will use a simplified RS-232 (serial) signal pattern for the protocol.

Although this project involves communicating between two microcontrollers, you are only responsible for the microcontroller that *receives*. I will supply a microcontroller that will transmit. Furthermore, instead of communicating

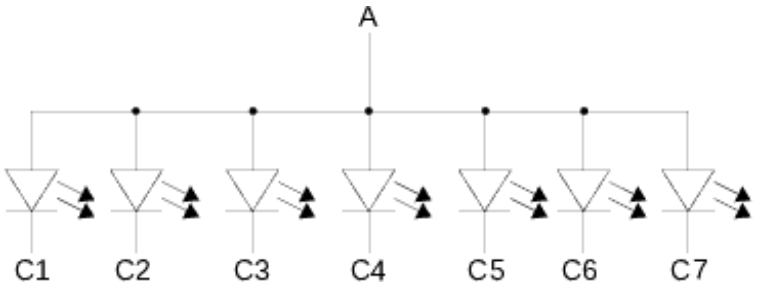

Figure 1: Internal circuit of a 7 segment display.

over wires, we will communicate wirelessly via infrared where high and low voltages will be represented by presence and absence of IR light. Since my microcontroller will send the IR, my microcontroller will have an IR LED. Your microcontroller will receive, and thus your microcontroller will use an IR transistor (BJT-based). The following sections describe the circuits needed for your microcontroller, as well as the protocol that you should use to receive updates.

2.1 IR Receiver

My microcontroller will emit infrared pulses in a specific pattern (outlined below). Your microcontroller must detect these pulses, and thus you will need to use an IR detector/receiver circuit. The IR receiver is a transistor that acts as a BJT. The the IR transistor only has connections for the emitter and collector terminals. The base terminal does not have a lead. Instead, the voltage of the base terminal is controlled by the amount of infrared light that is incident on the lens of device. With more infrared light, the higher the base voltage.

You should use an LTR-301 IR transistor for this project, arranged in a *switching* configuration (as opposed to an amplifier). Thus, the output of the IR transistor should either be a high or low voltage. The datasheet for the LTR-301 can be found at `http://optoelectronics.liteon.com/upload/download/DS-50-93-0013/LTR-301.pdf`. The maximum collector current I_C is 0.0002A (0.2mA). You must determine what resistors that you will use along with the LTR-301 IR transistor. You must submit the calculations that you used to determine the resistance(s), as well as proof that your selection limits the collector current to an acceptable maximum and that the resistor is within its maximum power dissipation.

The circuit arrangement for the IR transistor can be similar to the switching examples on slide 20 of topic12.pdf. The V_{out} can be connected directly to a digital input pin of your Teensy, and the on/off status determined with the `digitalRead()` function. You can supply the IR transistor with the fixed 3.3V supply pin of the Teensy. Be warned that ambient light (including florescent lights in the lab) may also emit infrared. If you suspect that the classroom or lab has stray IR that is making it difficult to turn the IR transistor off, you can cover your project or turn off the lights. To debug this, you can measure the V_{out} voltage of your IR circuit – if it does not fall to nearly 0V, then there may be too much light.

For testing, you can assume that there will be no obstructions between my IR transmitter, and your IR receiver. I will also make sure there is not too much ambient infrared in the room while testing. I expect that your project will function if the IR transmitter and receiver are at least a few inches apart.

Your project source code should implement a C++ class that handles receiving the IR pattern, and converts that pattern to an integer whose value corresponds to the time and that will be used to determine the "minutes" position to be displayed on the digital display. The IR pattern will be further discussed below in the "Signal Protocol" section.

2.2 Digital Display

For this project, you will need a *seven segment display*, which can display a single numeric digit from zero through nine. You can find seven segment displays in the cabinet (the door on the right). Seven segment displays are made of seven red LEDs. The anode terminals are all connected to a common pin of the display, and the cathode terminals are the remaining pins of the display. The schematic for the seven segment display is shown in Figure 1.

Each LED of the 7 segment display has a forward (barrier) voltage of 1.7V. You must determine how to connect these LEDs to the Teensy such that your Teensy can turn on/off each LED individually. Remember that the anode voltage must be greater than the cathode voltage by at least 1.7V to turn the LED on. You must use the `digitalWrite()`

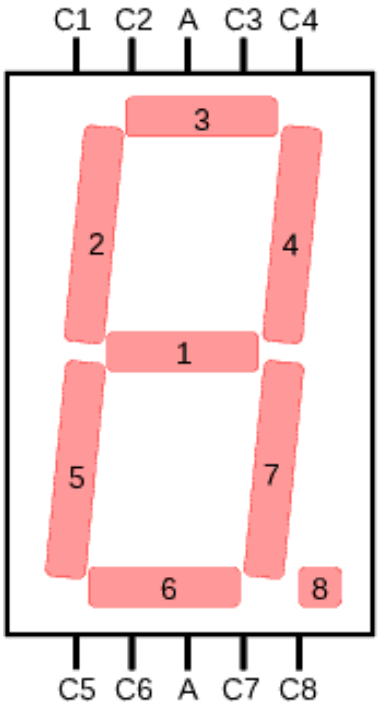

Figure 2: Mapping of LED segments to the pins of the 7 segment display.

function to turn the LEDs off and on. You must also have a current limiting resistor – the value of which you must prove to properly limit current to 16mA, and does not dissipate more than the 0.25W rating of the resistors in your kit.

It is your choice which Teensy pins to use for which segments. It is also up to you to determine how to display each decimal digit from zero to nine. The mapping in Figure 2 shows which pins of the 7 segment display are the cathode terminal of each LED. You do not need to light the decimal point LED of the display for this project. Assume that your time value is 0 at start-up, and you don't need to count down until you receive your first time value. More information about how the time should be displayed is discussed below in the section titled "Time Updates".

2.3 Signal Protocol

The pattern of high and low pulses of IR, used to communicate values, will loosely follow the RS-232 signaling protocol. This protocol is used to send a single, 8-bit value at a time.

Before a new value is transmitted, the IR light will be on, and thus, your IR transistor will be on as well. The start of a new value is indicated by turning the IR light off, and thus your IR transistor will turn off.

Once this on to off transition occurs, you must measure 10ms (10 milliseconds). After 10ms, you can read the status of the IR to determine the value of the most-significant bit of the 8-bit number (the 7th bit position). For this protocol, the presence of IR indicates a 1, and absence a 0. After another 10ms, you can read the status of the IR again to determine the next bit of the 8-bit number. Every 10ms, you can measure the IR to get the next lower bit position until you reach the least significant bit (the 0th bit position). Thus, it takes about 80 milliseconds to transmit a complete value.

After the last bit has been transmitted, the IR will be turned back on until the next 8-bit number should be transmitted.

Figure 3 shows an example timing diagram of the complete transmission of a single byte. Timing diagrams show the voltage on the y-axis (simply HIGH and LOW for digital binary values), and time on the x-axis. As can be seen, the IR is on initially. Eventually, the IR is turned off – labeled "Start" in the diagram. After 10ms, it is safe for your program to read the bit value for the most-significant (7th) bit, in this case, it happens to be 0. Ten milliseconds later, it is safe for your program to read the 6th bit, another 0 in this example. After 80ms total, your program should have read all of the 8 values. In this case, your program should have read 00011010_2, which is equivalent to 26_{10}. Once this sequence is complete, the IR is turned on in preparation for the next value transmission (not shown).

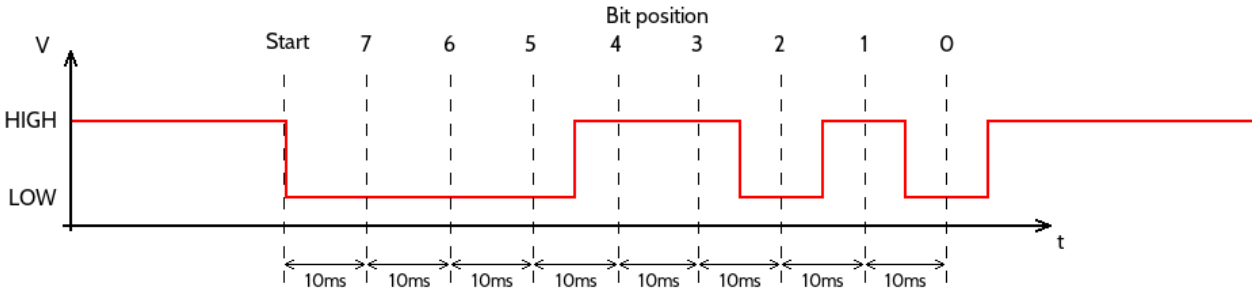

Figure 3: Example timing diagram showing the pattern used to transmit the value 26_{10} (0x1a).

Figure 4: Allowable digits for the time.

Since this protocol sends 8-bit values, the possible range for each transmission is 00000000_2 to 11111111_2, which is 0_{10} to 255_{10}. However, we would like to represent time from 0:01 (one second) up to 9:59 (almost ten minutes). The transmitted value will represent a number of *seconds*. So if each integer value represents a second, the span from one second to 9:59 will require values from 1_{10} to 599_{10} – which is not possible to represent in 8 bits. To get around this problem, the transmitter will need to send two 8 bit values, first the upper bits, then followed by the lower bits. This means that you will need to keep track of the first byte while waiting for a second transmission to get the second byte. You can read these into an integer, and use that integer to represent time. You are allowed to multiply the entire value by 1000_{10} in order to internally represent time in milliseconds instead of seconds if you wish.

This protocol will be used to send updated time values to your microcontroller. Each transmission of a pair of value using this protocol will signal to your microcontroller to update the countdown and the 7 segment display. A countdown may not have completed before a new pair of values is received – in that case, start your countdown over using the newly received value.

For this project, when I test I will only send a new value to your microcontroller at least 500 milliseconds after the previous value was sent. This 500 milliseconds is a lower-bound (this means that you do not need to handle new time values being sent sooner than every 500 milliseconds). There is no maximum time between new time values.

The two bytes that will represent the number of seconds your countdown timer should start with will be sent one after the other, using the protocol described above. The first byte transmitted will represent the upper bits of the time, and the second byte transmitted will represent the lower bits of the time. There will be at least 20 milliseconds but not more than 100 milliseconds between the first byte and the second byte.

2.4 Time Updates

As mentioned previously, the timer should display only the minutes remaining. Also the time displayed should initially be zero until the first time is received. Digits should appear as shown in Figure 4.

Once a time has been received (both bytes), then your microcontroller should automatically start counting down time. Your seven segment display should always stay up-to-date with the minute of the countdown. So, for example, when the time received is 1:05 (the value 65_{10} was transmitted), then your seven segment display should show the value 1 for five seconds, then display 0 for 60 seconds. If the time received is 3:12 (the value 192_{10} was transmitted), then your seven segment display should show 3 for 12 seconds, then 2 for 60 seconds, 1 for 60 seconds, and 0 for 60 seconds. If the time transmitted is less than a minute, then your display will show 0.

Valid times that you can expect from the transmitter will always be between 0:01 and 9:59.

Once your timer reaches zero, then you should "ring" an "LED alarm". The alarm in this case will simply be several LEDs from your lab kit. You should come up with an interesting, repeating, blinking pattern of your choosing. This blinking pattern will act as the alarm. You should add at least 4 LEDs (don't forget to use current-limiting

resistors for each). Your LED pattern should be something interesting, not just all on followed by all off. Get creative, have fun with it – feel free to arrange your LEDs in an interesting shape (appropriate LED shapes though, please, we don't want anything that would get you in trouble if it landed on the front page of the La Crosse Tribune). You are permitted to *additionally* use your seven segment display as part of the alarm pattern if you would like. The pattern should repeat until the next new time is received.

3 Software Architecture

Good software engineering practice means that you should think about splitting this program into classes. Spend some time thinking about what code functionality might be well contained within a class.

For example, you may want to move the logic that extracts a value from the IR transistor into its own class. This class could have a function that returns -1 (a time that is not possible) as long as no new value has been received. When a new value has been received, the function then returns the received value. The main `loop()` function can then just repeatedly call this function, do nothing when no new value has been received, and then update the displays when a new value is ready.

You could also split out the functionality needed for the 7 segment display. This class could have a function that is called when the countdown transitions to the next lower minute.

The use of `delay()` statements is allowed in this assignment. But be careful using `delay()` statements inside of your functions, since functions with `delay()` statements will not return until the delay is complete. You might want to consider using the `millis()` function instead of `delay()`. `millis()` immediately returns a `long` that represents the number of milliseconds that the Teensy has been powered on. Therefore, each call to `millis()` will return an ever increasing value. You could calculate the difference between two calls to `millis()` to establish an elapsed duration of time.

These are only possible ideas for the overall design of the microcontroller program. You are free to implement your program any way that you see fit.

4 Tester Microcontroller

To help you test your program, a microcontroller will be provided that you can use to transmit new time values via the IR LED. This microcontroller has a potentiometer and an LCD screen. You can turn the potentiometer to increase or decrease the time. The display will show you the currently selected time. I have found that the electrical noise in the room cause the measured potentiometer value shift around. I tried to mitigate the issue as much as possible without causing delays in updating time values.

The tester microcontroller also has a push button that, when fully pressed and released, will send whatever value is currently displayed. Thus, no values will be transmitted until you press and release the button.

This microcontroller is powered with a 9V wall charger, and does not need to be connected to a PC. There is an on/off toggle switch to turn the power on and off.

There will only be a single tester microcontroller! You must share the tester with other people. Please be courteous. Also, plan ahead... you may not have immediate access to the tester if you put this project off until the last minute!

5 Grading

As mentioned above, grading is based on four components. First, you must demonstrate your project before the deadline to me. The display must be updated to the correct values, also displaying the alarm pattern at the appropriate time. Be sure to test boundary values. For example, if I send the value 1:00 exactly, then I shouldn't see the display ever set to 1 – since once the countdown is started, the time is immediately 0:59, 0:58, etc.

Aside from the demonstration, there are several items that must be submitted. First, the circuit diagrams along with calculations that you used to determine resistor values, the voltage across and current through the various components, and power dissipation of the resistors should be submitted. You can either submit on paper, or on Canvas (pdf only, please). Be sure that it is clear which calculations are used for which circuits, and also which calculations are for

determining input or output voltages, which calculations are for power estimates, etc. I do not want a hodgepodge of equations that I need to piece together what they mean.

You should submit all of your source code (the .ino, .h and .cpp files) to the "Midterm Project" assignment on Canvas. Please do not zip these source code files, you can just submit the files individually.

Finally, you should leave your assembled breadboard(s) with me for grading. While this project is not a beauty contest for pretty wiring, it is expected that your breadboard(s) can be moved without being disassembled, and without wires coming unconnected, etc. You may lose points if your breadboard is so untidy that it does not work after being picked up and moved. I will test your circuit again after the demonstration. It is in your best interest to use wire that has been trimmed to proper lengths.