EEL 3744

Dr. Eric M. Schwartz

Revision **0**

Lab: Remote Controlled Synthesizer

OBJECTIVES

- To learn how to use the input timer system for a microcontroller, i.e., the input capture system on the XMEGA. This feature will be used to recognize signals from an infrared remote control.
- Combine XMEGA systems studied throughout the semester and use them simultaneously.

REQUIRED MATERIALS

- uPAD and Proto Base kit and tools
- Digilent Analog Discovery (DAD) kit
- Working keypad or serial interface
- Working LCD circuit
- Working Lab 7
- 1 – 40 kHz Remote Control
- 1 – 40 kHz IR receiver (with datasheet)
- 1 - 0.1 µF Capacitor
- XMEGA documents
  - doc8331 (Sections 6,14)
  - doc8045: *AVR1306: Using the XMEGA Timer/Counter*

PRELAB REQUIREMENTS

PART A – INTRODUCTION

In the previous lab, you learned how to use XMEGA's Timer/Counter (TC) system for output compare operations. In this lab, you will use TC for its input capture (IC) operation. Specifically, you will use IC function to create a multi-purpose remote control controlled receiver and appliance controller.

Before attempting this lab, make sure to review the TC system as described in Lab 7 document. In addition, read doc8331 section 14.7 and reread doc8045 (*AVR1306: Using the XMEGA Timer/Counter*). Unlike output compare (OC), IC system is used to timestamp events occurring on the input capture pin. This capture is accomplished when the processor latches the contents of the CNT register into a CCx register when a specific event (edge) occurs on an I/O pin.

In this lab, you will use the OUT pin of the IR detector (Figure 1) as the input capture pin. The infrared (IR) detector that you have is the GP1UE26XK0VF (from Sharp, DigiKey #425-2514-ND). This detector can transform an IR signal into digital pulses (as shown in Figure 2).

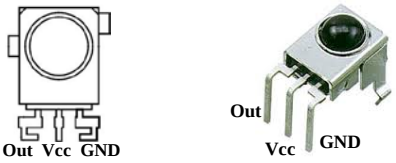


Figure 1: IR detector schematic and picture.

The left pin in the figures is the signal output, the middle pin is Vcc (3.3V), and the right pin is GND. An infrared *detector* takes the analog IR signal that it receives, demodulates it, and gives us a digital waveform on the output pin. Figure 1 shows an example of the output waveform from a remote control input.

Mount the IR detector to your uPAD Proto Base PCB. Connect the ground pin to the case with a small wire and solder. Add a bypass capacitor (0.1µF) between the Vcc (3.3V) and ground pins. Attach a pull-up resistor (with a value between 4.7kΩ and 10kΩ) between the output signal and Vcc and then attach the output signal to a CCx pin of your choice.

The IR detectors that were purchased this semester are extremely cheap and very susceptible to IR noise that can come from sunlight or even fluorescent lighting. (It may be that the dark plastic that is generally used in front of the IR detectors, and on the IR emitters on remote controls, does some filtering of these noise sources.) To test your IR detector circuit, connect the output to your DAD and use the LSA or oscilloscope function, triggered on a falling edge. Include a screen shot of this in your report. If necessary, cover your IR detector so that the IR noise is minimized during the remainder of your lab.

Find a remote control from a TV, DVR, DVD, Blu-ray, or VCR device. Aim the remote control at the IR detector, press a key (e.g., the "1" key on the remote) and again use your DAD's LSA or oscilloscope function to capture the waveform. Include a screen shot of this in your report. The waveform of will look similar to Figure 2. If you see anything very different than Figure 2, please send me the figure.

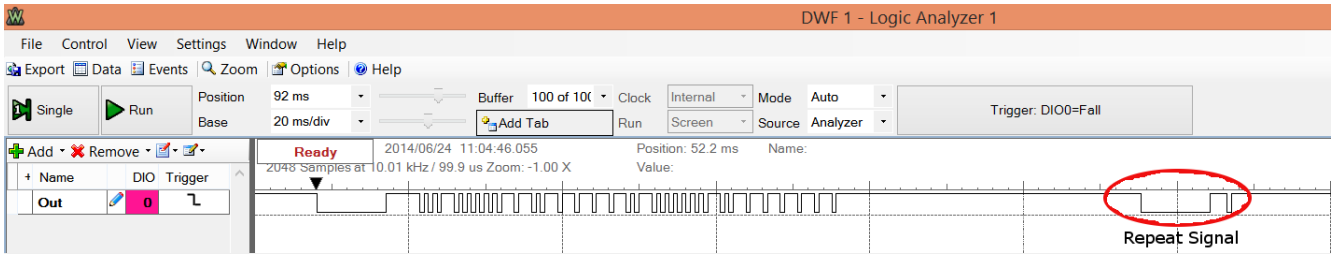


Figure 2: One example of the demodulated signal from a remote control's "volume up" button.

Lab: Remote Controlled Synthesizer

PART B - REMOTE CONTROL

Imagine … You have decided it is time to enhance your apartment so that you can spend more time in bed, on the couch, or in your favorite chair. You would like to operate various appliances remotely; however, you cannot afford a fancy wireless setup. Since you already have a TV/DVD/DVR/Blu-ray remote control, you decide to use this to control your appliances. You have decided that the first step is to design a receiver that can recognize your remote control signals. Because you are a microprocessor whiz, you are sure you can use the XMEGA to recognize keys from a remote control. In order to test your receiver, you recognize keys and display the name of the corresponding appliance being selected on an LCD to test your receiver.

Button	**Message** (// = new line)
1	1. MP3: Hotline // Bling
2	2. BR: Avengers // Ultron
3	3. DVR: Mr. // Robot
4	4. MP3: Downtown // Macklemore
5	5. TV: Agent // Carter
6	6. DVR: Game of // Thrones
7	7. RR: // www.is.gd/WVZvnI
8	8. BR: Furious 7
9	9. NF: House of // Cards
0	0. TV: Wheel of // Fortune

The receiver module should be able to record and recognize the ten channel keys (0 – 9) of a standard remote control. The XMEGA should then display the button number AND the Message on the LCD screen. If your LCD is not wide enough to display an entire entry, print the maximum number of characters you can for that entry. With the buttons 0-9 recognized, the notes from lab 7 should be played according to the lab 7 document for at least 1 second.

Each remote control unit comes with a unique controller. Some manufacturers have started building generic controllers that can record and play back the signals from several different controllers. This allows the user to replace all of his/her controllers with the one generic one. In this lab, we will design our own programmable infrared (IR) remote control receiver. You may use your own remote for this lab, but we may ask you to show that it works with our remotes when you get to lab. We will have remotes available in lab for in-lab use only.

TV/DVD/DVR/Blu-ray remotes work by sending out an infrared serial data stream. The sent data can be thought of as a series of 1's and 0's, although it is in fact a series of square waves at a single frequency that is either on or off; the result is an envelope of a modulated higher frequency square waves. (A waveform like that shown in Figure 1 can be ANDed with a higher frequency square wave to produce the signal sent out of the remote control's IR emitter.) The receiving device (i.e., the IR detector) can detect the infrared radiation, and provide the envelope of the modulated IR signal. Since there is a lot of infrared radiation in the environment, the infrared beam is modulated with a ~40 kHz waveform. When the signal is high '1', an IR signal pulsing at ~40 kHz is sent out. When the signal is low '0', no IR is sent. In TVs, DVD players, etc., a sensor module containing an IR sensitive transistor and a ~40 kHz filter detects and demodulates this ~40 kHz. (Other frequency detectors are also available.)

Figure 2 has an example of the data recorded from the IR detector when the "volume up" button on a remote was pressed. The longer period can be interpreted as a digital '1' and the smaller period can be interpreted as a digital '0'.

PART C - PROGRAM REQUIREMENTS

You must write all programs for this lab in C code. You may write the lab in assembly for partial credit if you cannot get your C code to work.

When you start your program, it should display a menu on the LCD that offers two choices to select with your keypad/keyboard:

Keypad Options:
1) Record a key
2) Play Something

The *Record* key option should allow you to choose the digit (using your keypad/keyboard) on your remote control that you want to record (0 through 9), prompt you to hit the appropriate key on the remote, record the IR signal, then tell you that the key has been recorded. The *Play* key option should allow you to hit a number on the remote and play the associated tone(s) until 1 second has elapsed (or until a song is completed) and output the appropriate message to the LCD. The IR signal should be ignored while playing the music or writing to the LCD.

So how do we recognize keys? As shown in Figure 2, each key sends a series of pulses of various widths. Your program should measure the time between each edge in the input signal (both rising edges and falling edges), and store these times in a table. (See Figure 14-11 in doc8331 for the suggested mode for capturing the pulse width durations.) Note that the waveform shown in has about 49 level changes, giving us about 100 pulse-width times. Your program should measure each **of up to 100** pulse widths (which may be needed for some remotes), and store these values in a table. You **cannot** count on each signal having exactly 100 pulses. You should write your code so that it can handle at most **100** pulses. **You should also write you code so that if it receives more than 100 pulses it does not overwrite the data for the next key.**

How do we detect the beginning and ending of each key? Notice that before and after each signal, there is a long

section (at least 10 ms) of silence. You should use that silence to trigger the start and end of a character. The key will start when it receives the first edge after 10 ms of silence (no transitions), and end when it receives the 100th pulse width (edge) or 10 ms of silence, whichever comes first.

We will describe a key as a table of pulse width times. In the example, there are 100 pulse widths that we can measure and store. The pulse width table for a key should include the number of pulse widths in that table, and then the list of pulse width measurements. We will need a table that can hold at least ten of these pulse width tables.

To detect that two keys are "equal," first the number of pulse width measurements must match. Secondly, ***each of the corresponding pulse width time measurements in the two tables we are comparing must be within 25% of each other***. These times will not be exactly equal, because in the real world, signals are not always precise. Your code must handle some imprecision in the input data, or you will not be able to recognize anything. Also, if the pulse width is extremely small, consider it a match to any other pulse of small length. (What is small? Look at your recorded data or the waveform captured with your DAD to determine "small.")

PROGRAMMING SUGGESTIONS

You are free to write your code any way you would like, as long as it accomplishes the program requirements. The following is offered **merely as a suggestion** of one possible way to break your program into modules.

MAIN:
Your main routine should contain a menu that allows the user to either record a new key or "**play something**" (TV/DVD/DVR/Blu-ray/MP-3). [Of course playing something can ultimately be replaced by controlling some electronics to do some other functions.] The **record** section should allow the user to specify which key is being recorded (0-9); then request that the user hit that key on the remote; and then let the user know that the key has been successfully recorded.

READ KEY:
The READ_KEY routine should setup the interrupts to record the pulse-width table for a key; then wait for that key to be completed. Note that you can setup the input capture to detect the first edge, and use that as the start of the character. There are a couple of ways you could detect the end of the character or you could record a set amount of pulses. A common solution is to use a timer overflow interrupt and set the period to be exactly 10ms. Then each time an Input Capture edge occurs, reset the timer and turn on the interrupt. When the overflow interrupt finally hits, you know that you have had 10ms of silence. The READ_KEY routine, if done correctly, could be used both to record new keys, and to read keys to be recognized.

ISRs:
You will need to write several interrupt service routines. These will measure the pulse-widths, and build the tables that describe a key. At least one OC will also be playing the note(s). You'll probably want two OCs for playing the note(s); one for the note frequency and the other for the note duration.

RECORD KEY:
The RECORD_KEY routine will allow the user to specify which key is being recorded and then will prompt the user to press that key on the remote. You can use the keypad to tell your program which remote key should be pressed, or your program can prompt you to press a given remote key. Then this routine confirms that the key has been successfully recorded. You should have a table with room for 10 different key patterns. The RECORD_KEY routine should make sure that the pulse-width table gets put in the proper slot.

IDENTIFY:
The IDENTIFY routine should read a key, then compare that key to each of the pre-recorded keys, and if it finds a match, should play the proper tone(s) via OC. The IDENTIFY routine will need a way to compare two pulse-width tables to see if they match (COMPARE_KEY).

COMPARE_KEY:
The COMPARE_KEY routine should compare two pulse-width tables. Note that the pulse-widths will not match exactly. Any two pulse widths that are within roughly **25%** of each other are a match. (Also, if the measured pulse width is less than a "small" length, consider it a match to any other pulse width of a "small" length.) ***Two keys match only if all of the pulse width times in their respective pulse width tables match***.

These above functions will be needed in your program, but you may break up the routines into functional blocks however you wish. The only requirement is that the program performs the function of recording and recognizing keys.

HINTS:
You are encouraged to "be the tortoise, not the hare," i.e., to build this design in pieces (as you did [or should have done] in each of the previous labs), testing each the intermediate programs in case you need to backtrack. Make your menus easy to use. Allow the user to re-record individual keys at will. Sometimes the hardware generates glitches that cause keys to be read incorrectly. You will want the capability to re-record individual keys.

Lab: Remote Controlled Synthesizer

Tip: Test your program in pieces, i.e., be the tortoise, not the hare!

Test the input capture portion of the programming by examining the contents of memory to verify you are getting valid data. You can also use breakpoints inside the ISR's to test if you initialized them correctly.

DEBUGGING HINTS:

- Make sure the IR controller you are using works and has **fresh** batteries.
- The IR sensor can trigger the input capture interrupt at random times due to ambient noise. Make sure you take this into account. (If necessary, you can try to protect the remote and detector circuit from light by putting a box over it.)
- It is very easy to make a programming mistake that will shift the pulse width values in the tables. Look for this in the memory view while debugging.
- The LED's that you added early in the semester are very useful for receiving feedback from your interrupt routines.
- Be sure to reset the timer at appropriate times since it will not reset automatically in the input capture mode. The timer will count up to whatever is in the period register before resetting to zero.
- Use the DAD's LSA (or oscilloscope) feature to measure the output of the IR receiver (as seen in Figure 2).
- Make sure you're not recording items such as repeat signals, unless your code is designed to deal with them.
- **Keep your interrupt routines as short as possible! You may miss edges if you don't.**

QUESTIONS

1. How could you modify your program to generate the patterns you recorded (turn it into a universal remote control)? What timing system would you use to do this?
2. What is the minimum pulse width that the XMEGA can measure?

PRELAB REQUIREMENTS SUMMARY

1. Add the IR detector circuit to your uPAD Base Board.
2. Perform some tests to verify that your IR Detector works properly, both by recording (using your DAD) an IR noise waveform and a remote control waveform. (Add both screenshots to your report.)
3. Write a complete program to play and record various notes/songs and LCD messages using an IR remote.

IN-LAB REQUIREMENTS

1. Run the program and verify that you can record and play keys using the IR remote, while also displaying an output on the LCD. You will use a lab remote control, not your own.
2. If you cannot get the complete program form above working, demo each part that does work.