OBJECTIVES

- To introduce the input timing feature of a microcontroller (called the input capture). This feature will be used to recognize signals from an infrared remote control.
- Combine the systems that have been used throughout the semester and use them simultaneously.

REQUIRED MATERIALS

- 1 – Keypad circuit (from previous lab)
- 1 - LCD circuit (from previous lab)
- 1 – 40 kHz Remote Control
- 1 – 4.7k-10k Resistor
- 1 - .1 µF Capacitor
- All the parts from lab 7
- You WILL need the following documentation:
  - Enhanced Pulse Width Modulator (ePWM) Module (OC) Reference Guide (SPRUG04A)
  - Enhanced Capture (eCAP) Module (IC) Reference Guide (SPRUG04A)
  - IR detector spec sheet

INPUT CAPTURE INTRODUCTION

The input capture function is an important architectural feature of the timer system of the DSP. Physical time is kept in the 32-bit free running counter register TSCTR. The timer can cleared to zero at configured resets and must be enabled to operate. Input capture functions are used to record the time at which events on the input capture pin, external to the processor, occur. This is accomplished (automatically) by latching the contents of the free running counter and storing it in an input-capture timer register (CAP1:CAP4). There are six input capture (eCAP) modules available to you, each with their own set of control and CAP registers; only one module is needed for this lab. The four CAP registers can be individually configured to latch the counter value at different times (rising or falling edges), and only ONE CAP register will be loaded at a time. The two-bit Mod4 counter determines which CAP register is next in line to be loaded with the count value. The free running (TSCTR) counter can be configured to reset after these CAP registers are loaded, or to continue running. See the documentation for more information.

The registers associated with each eCAP module are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSCTR</td>
<td>Time-Stamp Counter</td>
</tr>
<tr>
<td>CTRPHS</td>
<td>Counter Phase Offset</td>
</tr>
<tr>
<td>CAP1</td>
<td>Capture 1 Register</td>
</tr>
<tr>
<td>CAP2</td>
<td>Capture 2 Register</td>
</tr>
<tr>
<td>CAP3</td>
<td>Capture 3 Register</td>
</tr>
<tr>
<td>CAP4</td>
<td>Capture 4 Register</td>
</tr>
<tr>
<td>ECCTL1</td>
<td>Capture Control Register 1</td>
</tr>
<tr>
<td>ECCTL2</td>
<td>Capture Control Register 2</td>
</tr>
<tr>
<td>ECEINT</td>
<td>Capture Interrupt Enable Reg</td>
</tr>
<tr>
<td>ECFLG</td>
<td>Capture Interrupt Flag Reg</td>
</tr>
<tr>
<td>ECCLR</td>
<td>Capture Interrupt Clear Reg</td>
</tr>
</tbody>
</table>

PROGRAM INTRODUCTION

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/MP-3 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 TV/DVD/DVR/Blu-ray/MP-3 remote control signals (since you can’t even afford a new remote). Since you are a microprocessor whiz, you are sure you can use the DSP 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.

<table>
<thead>
<tr>
<th>Button</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MP-3: We are the Champions</td>
</tr>
<tr>
<td>2</td>
<td>DVR: The Daily Show</td>
</tr>
<tr>
<td>3</td>
<td>Audio Book: The Girl With the Dragon Tattoo</td>
</tr>
<tr>
<td>4</td>
<td>DVR: Biggest Loser</td>
</tr>
<tr>
<td>5</td>
<td>Blu-ray: The Sound of Music</td>
</tr>
<tr>
<td>6</td>
<td>TV: Dancing with the Stars</td>
</tr>
<tr>
<td>7</td>
<td>DVD: Wall-E</td>
</tr>
<tr>
<td>8</td>
<td>DVR: Colbert Report</td>
</tr>
<tr>
<td>9</td>
<td>MP-3: Bohemian Rhapsody</td>
</tr>
<tr>
<td>0</td>
<td>MP-3: Born This Way</td>
</tr>
</tbody>
</table>

The receiver module should be able to record and recognize the ten channel keys (0 – 9) of a standard remote control. The DSP 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 one 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 (using a cheap TV/DVD/DVR/Blu-ray/MP-3 remote). 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/MP-3 remotes work by sending out an infrared serial data stream. The receiving device can detect the infrared radiation. Since there is a lot of
infrared radiation in the environment, the infrared beam is modulated with a 40 kHz square wave. When the signal is high, an IR signal pulsing at 40 kHz is sent out. When the signal is low, no IR is sent. Figure 1 shows an example of the remote control waveform detected by the IR Detector Module.

In TVs, DVD players, DVR players, VCRs, and CD players, a sensor module containing an IR sensitive transistor and a 40 kHz filter detects and demodulates this 40 kHz IR signal.

The IR detector that you have is probably the GP1UE26XK0VF (from Sharp, DigiKey #425-2514-ND) shown here. The bottom right pin is the output, the middle pin is +3.3V and the top pin is GND. An infrared detector takes the IR signal that it receives, demodulates it, and gives us a nice, neat on/off voltage waveform on the signal output pin. Figure 1 shows an example of what this output looks like. Note that this output does NOT contain the 40 kHz modulation signal. You will connect this signal to one of your input capture pins.

Connect the ground pin to the case with a small wire and solder. Also, add a bypass capacitor (0.1μF) between +3.3V and ground. Attach a 4.7k to 10k pull-up resistor to the output signal and then attach the output signal to a eCAP GPIO pin (GPIO 24 for instance). Other resistor and capacitor values than those described previous may give better results. If you change your resistor/capacitor values, let me know of the values that maximize the range of your detector.

PROGRAM REQUIREMENTS

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

<table>
<thead>
<tr>
<th>Keypad Options:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Record a key</td>
</tr>
<tr>
<td>2) Play Something</td>
</tr>
</tbody>
</table>

The Record key option should allow you to choose the digit (using your keypad) 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 the 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 1, 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. Note that the waveform shown in has about 48 level changes, giving us 47 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 47 pulses. You should write your code so that it can handle at least 100 pulses. You should also write your 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 10 ms of silence.

We will describe a key as a table of pulse width times. In the example, there are 47 pulse widths that we can measure and store. The pulse width table for a key should

Figure 1: One example of the demodulated signal.
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 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. It is recommended that you use two CAP
registers to record all the pulse widths. You can set one up
to latch on a rising edge, and the other to latch on a falling
edge, and one interrupt to fire after they both latch their
count values. There are a couple of ways you could
detect the end of the character. A common solution is to
use one of the Output Compares. Each time an Input
Capture edge occurs, reset the Output Compare time to
10ms past the input capture time. When the Output
Compare 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 OC for playing the
note(s); one for the note(s) frequency or frequencies and
the other for the note(s) duration(s).

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.

Tip: Test your program in pieces.

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.
See the input capture examples in the documentation if you need a starting point. The examples start on page 28 in the eCAP Module reference. Examples 3 and 4 are the most relevant to this lab.

DEBUGGING HINTS:
- Make sure the IR controller you are using works and has fresh batteries.
- Notice that the registers in the eCAP documentation do not have their addresses listed, only an offset value.
- 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.)
- Make sure to clear the necessary interrupt flags at the ends of your ISR routines so that they can fire again.
- It is very easy to code a bug 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.
- **Keep your interrupt routines as short as possible! You may miss edges if you don’t.**

QUESTIONS
1. What is the minimum pulse width that the DSP can measure?
2. 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?

PRELAB REQUIREMENTS
1. Read the ENTIRE lab handout.
2. The circuit diagrams used in this lab. Wire up these simple circuits BEFORE lab.
3. Prepare detailed flowcharts or pseudo-code (BEFORE WRITING ANY CODE) for the RECORD_KEY and the COMPARE_KEY subroutines. You MUST have complete & detailed flowcharts in order to enter the lab.
4. The ASM files or error-free LST files.
5. Answer to the pre-lab question.

IN-LAB REQUIREMENTS
1. Run the program and verify you can record and play keys from the IR remote with display the output on the LCD.