Lab 16b Simulated IR Remote Control Interface

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

Goals
- Hardware/software development using the TExaS simulator,
- Assembly language development,
- Use the input capture to measure pulse width,
- Write and visualize interrupting software.

Review
- "How to program in Assembly" document in the **`assmbly`** folder on the CD
- Valvano Chapter 6 about input capture and output compare,
- The **`Readme.exe`** on the CD that accompanies the book,
- MC68HC812A4 Technical Reference Manual, Chapter 13, about input capture.

Starter files
- IR.* files in the MC6812 folder of the TExaS application

Background

Infrared remote controls are widely used for controlling TV's and VCR's. This experiment will illustrate how to use the free-running timer of the 6812 for measuring time periods. In particular, you will use the input capture and output compare interrupts to decode time-dependent signals from a simulated IR remote. The last part of the experiment performs real time profiling of the interrupt processes.

Most of the modern remote control devices are based on infrared emission. In this experiment, you are going to use the ITZA remote control. To detect the infrared signal generated by the remote control, you will use a simulated infrared detector module similar to the one manufactured by Radio Shack (Cat. No. 276-137A) or Sharp (GP1U57X). The simulated circuit is the following:

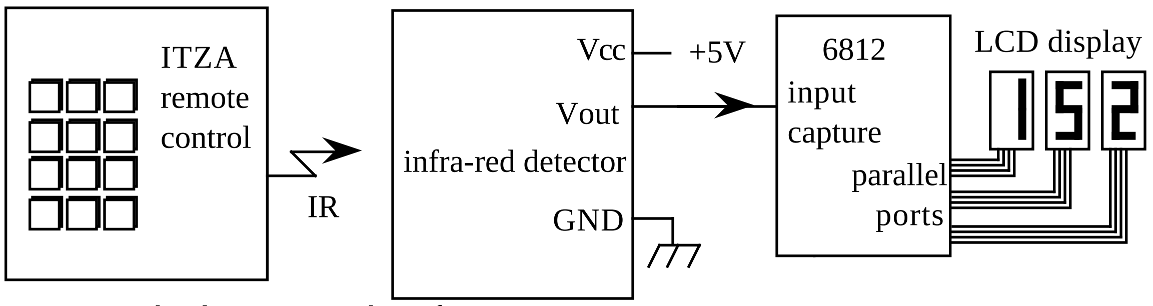

Figure 16.1. Simulated remote control interface.

The operator will type 0-9 digits on the simulated IR remote. The time-encoded waveforms will arrive on an input capture channel. Using both input capture and output compare, your software will decode the digits and display the results on a 3 digit LCD display. Except for the ritual, your software will run in the background. Your software should ignore waveforms that do not match the 0 to 9 decimal digital patterns. As each new number is typed, the display is left shifted and the new number is added on the right. For example

operator types	display shows
initially	000
1	001
5	015
2	152
7	527

There are different options for reading the sensor. The 6812 has a 16 bit free running up counter, TCNT, which is incremented every **n** E clocks. There is a prescaler (TMSK2 bits PR2, PR1, PR0) that selects the factor, **n** which can vary from 1 to 32. The prescaler affects the pulse width measurement resolution and the maximum pulse width. The input capture can be configured to activate on the rise, the fall, or both the rise and fall of input capture pin. Look at the explanation of the TCTL3 and TCTL4 registers. The 6812 will automatically save a copy of the TCNT register in the input capture register (e.g., TC0) at the time of the active edge of the input capture pin. If armed, an interrupt will be requested during input capture activation. A series of pulses are issued when you hit a key on the simulated remote, and you have to find a way to figure out what is the beginning and what is the end of the message. You can use the fact that there is a fairly long interval between the bursts of pulses.

Preparation (do this before your lab period)

1: Experimental measurements:
Run the existing assembly language example, IR.*, and observe the waveforms on the simulated logic analyzer. In particular, experimentally determine the following parameters:

• the number of pulses in a scan (ignore the first waveform after a reset),
• the three sizes of pulse widths: start, big and little,
• the total length of a waveform.

2: Hardware development:
Configure your IO document to include 3 LCD digits and an IR remote. Attach the LCD display to parallel ports, and connect the IR remote to one or two input captures. Modify two of the buttons (other than 0-9), so that one has two many pulses and the other has too few pulse. Your software should ignore the waveforms created by these two extra buttons.

3: Software:
Write the assembly language program to decode the data from the infrared sensor and output to the LCD display. The main program should initialize the interrupts and perform a do-nothing idle loop. The input capture interrupt handler(s) will measure the pulse widths in the waveform. There are many approaches to decoding the IR signal, but your software system must

• be written in assembly,
• use both input capture and output compare interrupts,
• include no foreground functions, other than initialization,
• be able to reject patterns that have a different number of pulses.

The following algorithm is included for consideration purposes. In other words, you are free and encouraged to develop you own algorithm. In this solution, the IR signal is connected to two input capture pins. It also uses one output compare channel. There are three globals in RAM: `mode count` and `result`. There will be a fixed constant table in EEPROM containing the `result` values for the digits 0 to 9. The initialization routine sets the `mode` flag to 0, meaning the system is searching for the first pulse. One input capture is defined as a falling edge input without interrupts, and the other input capture a rising edge input with interrupts. The output compare channel is initially disarmed. The initialization routine also specifies the 12 bits connected to the LCD are output, and initializes the LCD display to `000`. On the falling edge input capture, no interrupt is requested. Nevertheless, the time of the falling edge is captured into input capture latch. On the rising edge, an input capture interrupt is requested. The ISR will measure the pulse width, which will be the difference between the two latches, and acknowledge the interrupt. Each pulse width will either be big or little. If the `mode` flag is zero, then

• arm the output compare interrupt to occur after the entire scan should be finished,
• initialize the `result` to 0 or 1 depending one whether this pulse is big or little,
• set the `mode` flag to 1, meaning you are currently scanning for pulses,
• set the pulse `counter` to 1, meaning one pulse has been seen.

If the `mode` flag is one, then

• logical shift left the `result` making room in bit 0 for this measurement,
• add to `result` a 0 or 1 depending one whether this pulse is big or little,
• increment the pulse `counter` to 1, meaning one pulse more has been seen.

On the output compare interrupt, the entire scan will be processed. If the incorrect number of pulses has occurred, then the scan is ignored. Otherwise, you will look up the measured `result` value in the table to see if it is valid. If found, the index into the table will be the corresponding digit 0 to 9. Again, if the `result` value is invalid (not in the table), then the scan is ignored. If the number of pulses and the `result` are valid then the LCD display is shifted (middle digit moved into the most significant digit, and the least significant digit moved into the middle digit) and the new digit is added in the least significant position. In all cases (valid or invalid), the output compare interrupt is disarmed, and the `mode` is set back to 0.

A "syntax-error-free" hardcopy listing for the software is required as preparation. The TA will check off your listing at the beginning of the lab period. You are required to do your editing before lab. It is good practice to include testing procedures in the software design. Rather than turning it on and seeing if it works, add testing functions that can be run during the debugging phase. The debugging will be done during lab. Document clearly the operation of the routines.

Procedure (do this during your lab period)

Test the software in small stages. When testing one component, the other software modules could be "patched out". In particular, test

the LCD display software,
the pulse width measurement software,
the software that creates the `result` value,
the software the determines when the waveform should be over,
the table lookup software.

Checkout (show this to the TA)

Run the simulator at full speed (no FollowPC, no ShowCycles, close the Scope window) to verify proper operation of your software. The TA will program one key to have too many pulses and another to have too few pulses. Your system must be able to reject these bad waveforms, and be ready to accept valid waveforms.

Next, put a breakpoint in the input capture ISR. Run your program with FollowPC active and dumping instructions/cycles into the log window. Be prepared to explain to the TA exactly what happens when an input capture interrupt occurs. In particular, show the context switch, where the registers are pushed on the stack and the PC changes. Single step your program until a couple of instructions past the rti instruction. Again explain context switch, where the registers are pulled and the program returns to the foreground.

Hints/clarifications

1) You can write the software that measures pulse width, then experimentally determine the threshold separating the big and little pulses.

2) You can write the software that shifts the data into the result variable, and experimentally determine the values of result for each of the digits 0,1,2,3,4,5,6,7,8,9.

3) Since the interval of the pulses generated by the remote control is short, your interrupt software should be fast. Make sure your interrupt handler is fast enough for proper operation (the handler should finish its task before the next interrupt).

4) With TExaS simulator version 1.05 and before, you must acknowledge the output compare interrupt (clear the flag) even if you disarm the interrupt.

5) In a real system, pulses will occur from sources other than the remote. Your software must be able to reject these extra signals and recover so that actual signals can be subsequently received. In the simulation, you can hit the remote key again before the previous wave is complete to generate illegal patterns. You will also program some of the other keys to produce illegal patterns.

6) The first scan, after a reset, will be incorrect because the IR simulation is initialized low (0) rather than high (1). If your software is correct, then it should ignore the first waveform.