Group7
Adam Lengerich

Software Design Considerations, Narrative, and Documentation
(Homework 10)

Software Design Considerations:

The PIC 16F877 is an 8-bit processor, but uses 14-bit instructions. These instructions are stored in the PIC's 8K of flash program memory. Data is stored in the 368 bytes of the PIC's data memory. The I/O ports on the PIC can be accessed through the 256 bytes of the EEPROM data memory. Detailed mapping of these ports can be found in section four of [1].

The C compiler that we used created all the startup code needed to get our chip up and running. To do this, it uses information that we give it such as which pins are used for input/output, which clock to use, etc.

Our embedded application code was a polled program-driven. The only modules that are time sensitive are hit detection and sound generation. Fortunately, both do not ever need to be running at the same time. When a sound is playing, hit detection can be put on hold. The process used for hit detection is described in detail later in the software design narrative, but basically, each photocell must be scanned every 100^{th} of a second. After each round of scans, the microcontroller simply waits for about 10 ms. On any round of scanning, if a hit is detected, then a routine is called that updates the score, clears the status of all photocells, and then plays a sound file for a second or so. After that, scanning resumes as normal.

Software Design Narrative:

The software running on the microcontroller located in the target unit performs the following general tasks: detect hits on the photocells, sound signal generation, score keeping, and updating the output LED's. The program starts out letting the user select the number of players and the max score. After this, the main program loop is entered

into. It uses a hit detection module to check for hits and also checks for the time up condition. Upon a hit being detected or a change of turn, the sound generation module is called. When either the timer or player score changes the update LEDs module is called.

For hit detection, we used just over 20 photocells. Voltages were read off each of these photocells using the microcontroller's analog-to-digital converters. In order to feed this many voltages into the microcontroller, we used an analog multiplexer. Therefore, the microcontroller had to communicate to the multiplexer to select cells as they were to be checked. We used the differences in measured voltages over time to detect rises and falls that occur when the laser pulse hits a photocell. Checking for a rise followed shortly be a fall eliminated erroneous hit detects due to abnormalities such as lights being turned on. When a rise is followed by a fall in less than or equal time to the length of the laser pulse, a hit is detected. We check each photocell roughly once every 10 ms

The built-in PWM proved to be inadequate for sound generation. It couldn't be scaled down enough. Therefore, we created our own software PWM. It took the form of a function with both frequency and time parameters. The module uses equations to determine half the needed period and the needed number of periods to create the desired sound. A loop with delays in it then used these values output the wanted signal.

The score-keeping module uses a lookup table to determine the points awarded for each hit. Each ring has a value from 5 to 1 starting in the center. After a hit is detected, this function updates the current players score and if it is a two-player game, changes the current player.

The 7-segment displays are arranged in 3 pairs. When a score, the timer, or anything else that is displayed on them changes, a special function is called to update them. The first of two parameters to this function is then new integer to be displayed on the pair. The second is the pair number (0, 1, or 2). Each 7-segment display has its own decoder/latch. The code module first sets the correct address and number values, and then strobes the PLDs. This is then repeated for the second of the two displays in the pair.

Software Documentation:

List of References

[1] PIC16LF877A
http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf

[2] PIC16F87A
http://ww1.microchip.com/downloads/en/DeviceDoc/39025f.pdf

[3] C Compiler Reference Manual
http://www.ccsinfo.com/PIC-Sept2004.pdf