HEARm Design Report

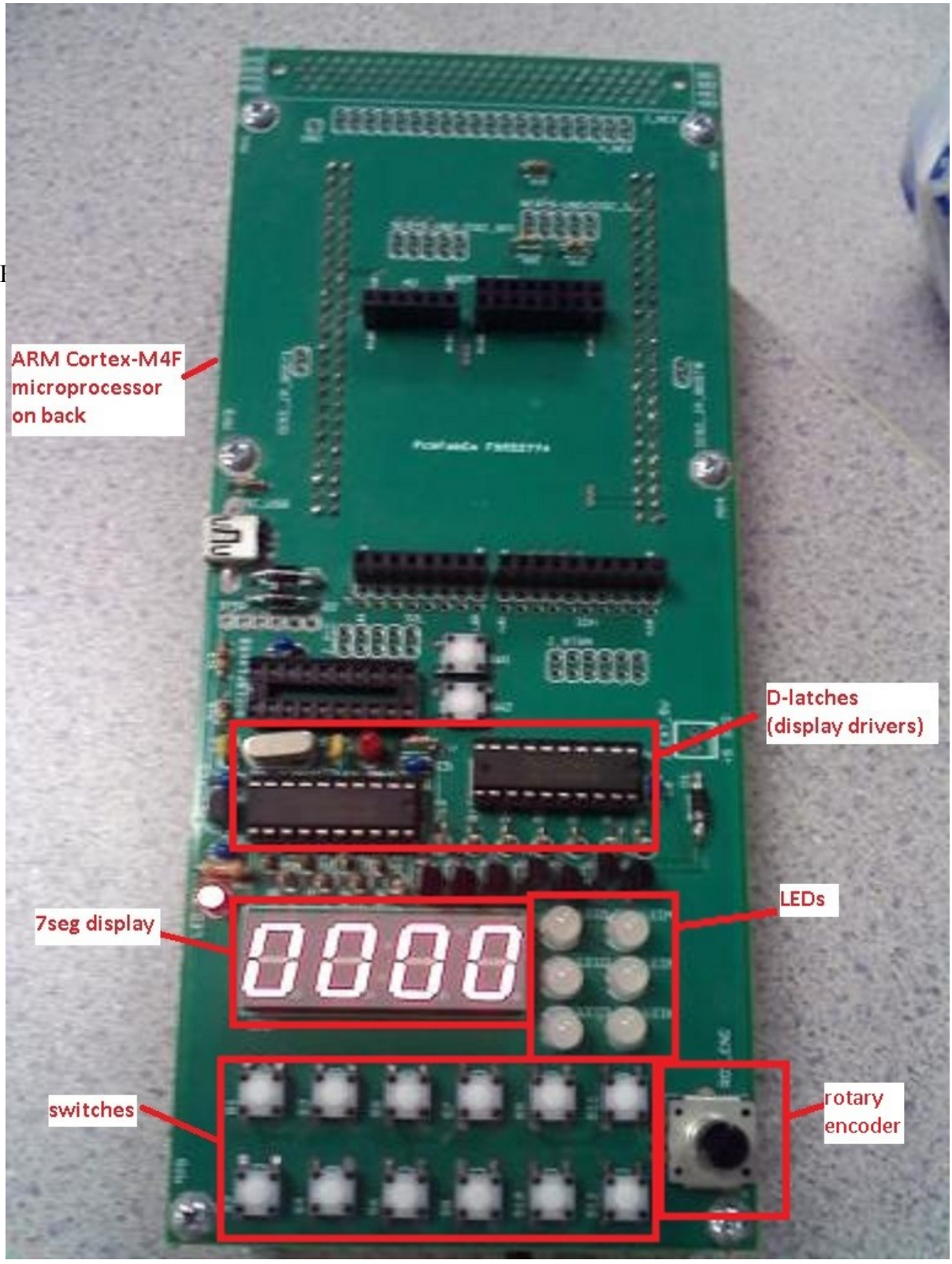
Author: Michael W. Schultz
ENEE440 Spring 2013
Dr. Hawkins

Table of Contents

The HEARm Circuit

I. Introduction

AR



The HEARm is an audio synthesizer (that does not yet actually generate sound). It is based on the Cortex-M4F microprocessor, which is mounted on a P24 I/O board which Dr. Hawkins designed and I soldered together. There are twelve switches and one rotary encoder for input, and a four digit seven-segment LED display and six normal LEDs for output.

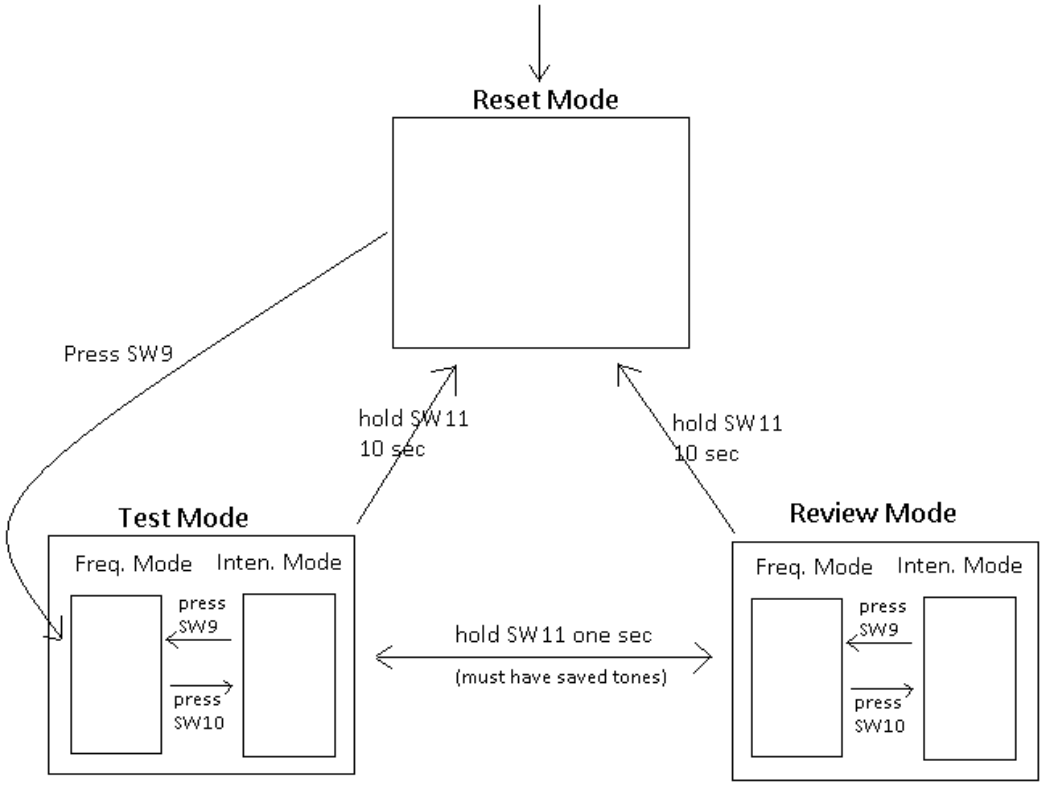
The basic organization of the software is some initial setup code (like port initialization), and then an infinite loop which samples input hardware periodically and updates output hardware accordingly.

All the low-level code that interfaces with the microprocessor's hardware registers was done in assembly. This includes port initialization, input reading, output writing, and interrupt initialization and operation.

II. Modes & Their Operation

The device can be in several operational modes, which are nested hierarchically as modes (Reset, Test, and Review) and submodes (Frequency and Intensity). Here is a diagram portraying the possible mode transitions of the device:

Modes of Operation:



There is a unique functionality associated with each mode; inputs and outputs behave differently.

A. Reset Mode

The device turns on in Reset Mode. The display here is all zeros, and does not change while in Reset Mode. The only input that works is switch 9, the frequency switch. Pressing this causes the device to enter Test Frequency Mode.

B. Test Mode

In Test Mode, one is able to change the frequency and intensity of the test tone, as well as save the test tone. Holding switch 11 for five seconds switches to Review Intensity mode, if tones have been saved. Holding switch 11 for five seconds causes LED5 to start blinking at an increasing rate, until at ten seconds of holding the device flashes zeros a few times, illuminates the top right LED (LED4) to indicate it is resetting, and then enters Reset Mode.

1. *Test Frequency Mode*

In this mode, switches 1-8 are used to alter the digits of the test frequency value, which is displayed on the seven segment display bar. The initial value displayed is 0125. Pressing the bottom-most switch below a digit causes it to decrement by one; pressing the switch above that causes it to increment by one. Pressing switch 12 saves the current test tone (which is test frequency plus test intensity) to the list of saved tones. Pressing switch 11 deletes the most recently saved tone. Pressing switch 10 goes to Test Intensity Mode. Switch 9 does nothing. The test frequency range is 0125-8000 (Hertz). Values are zero-padded on the left. The top left LED (LED1) is illuminated. The rotary encoder is disabled.

2. *Test Intensity Mode*

In this mode, the test intensity value is displayed (not zero-padded). The intensity range is -10-110 (dB). The default test intensity value (if you have just started or if the test frequency was changed) is -10. The lower left LED (LED3) is illuminated. Switches 1-8 are disabled. Switch 9 goes back to frequency mode. Switch 10 does nothing. Switch 12 saves the current test tone. Switch 11 deletes the most recently saved test tone.

The rotary encoder changes the test intensity. Two consecutive clockwise detents increments the intensity by five, and two consecutive counter-clockwise detents decrement the intensity by five.

C. Review Mode

The point of this mode is to view the frequency and intensity values of previously saved tones. LED3 will be illuminated while in Review Mode. One can always go to Reset Mode by holding switch 11 for ten seconds, just like in Test Mode. Switch 9 and 10 go between the Frequency and Intensity submodes, similar to Test Mode. However, one cannot change the frequency/intensity values displayed; they are just for review. So switches 1-8 and the rotary encoder are disabled. Pressing switch 12 repeatedly will step the display through all the saved tones (starting at the most recently saved tone), which repeat when you get to the end of the list.

III. I/O Port Initialization

All the input controls and output displays are connected to ports on the ARM microprocessor. These ports are initialized in software in the following way:

1) Enable their clocks in the Reset and Clock Control (RCC) registers. This is necessary since the clock signal for a port is normally disabled in order to save power.
2) Set output ports to "output" type in the MODER register of the GPIO register set.
3) Set OTYPER (output type) to push-pull.
4) Set OSPEEDR (output speed) to fast - 50MHz.
5) Set PUPDR (pull-up/pull-down) to pull-up.

There are four GPIO (general purpose input/output) registers in use on the ARM chip: registers A, B, C, and D. These include fifteen output ports: A0, A1, A4, A5, A6, A7, B9, C0, C1, C2, C3, C4, C5, C10, C11, and six input ports: B0, B1, B5, B6, C12, D2.

IV. Switches

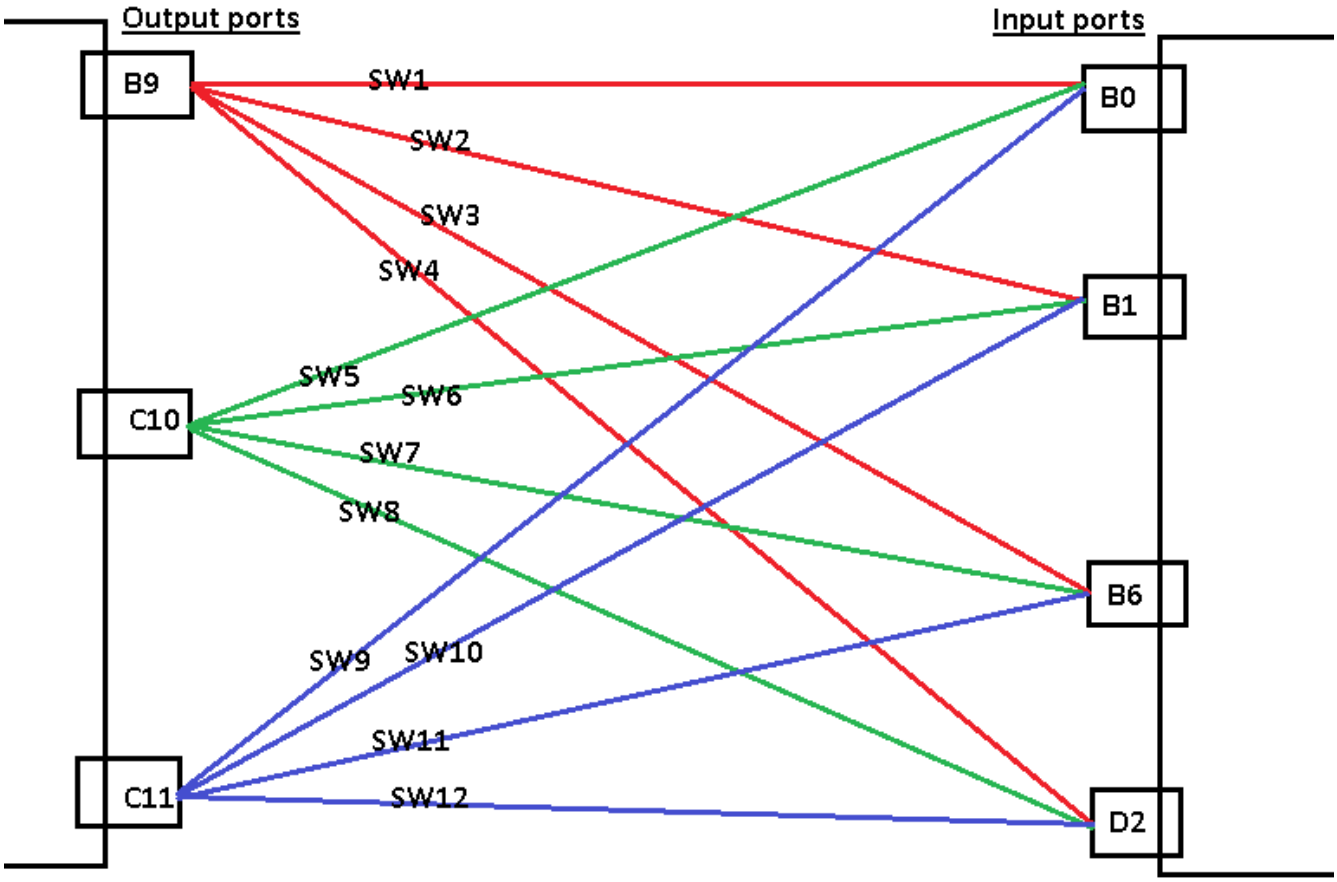
A. Hardware

There are twelves switches on the P24 board. While the switch is pressed, the connection between two of the switch's terminals is closed; while the switch is not pressed, that connection is open (i.e. no connection).

Each switch is connected directly to two ports on the ARM processor; an input port, and an output port. Each switch is periodically sampled. This is done by sending a signal from the output port and seeing if it registers on the input port. The input port is in a pull-up configuration, so when open it reads a high. So when we're sampling, we send a logic-0 from the output port, wait for accumulated charge to equalize across resistance in the circuit, and then check to see if the input port has received the logic-0 (meaning the connection is closed and thus the switch is being pressed).

One might imagine using 24 ports, with a unique port at both ends of each switch. But we can actually use far less ports by sharing some ports between switches. All we really need is a unique *pair* of ports, so one switch can use B9 to B0, another can use B9 to D2, etc. So we have twelve switches, which is three groups of four. So we use three output ports, each of which connects to the same four input ports. In this way, we use only seven ports to create twelve unique connections. I.e., each connection, composed of a pair of ports, crosses a unique switch. Below is a more precise diagram of this setup.

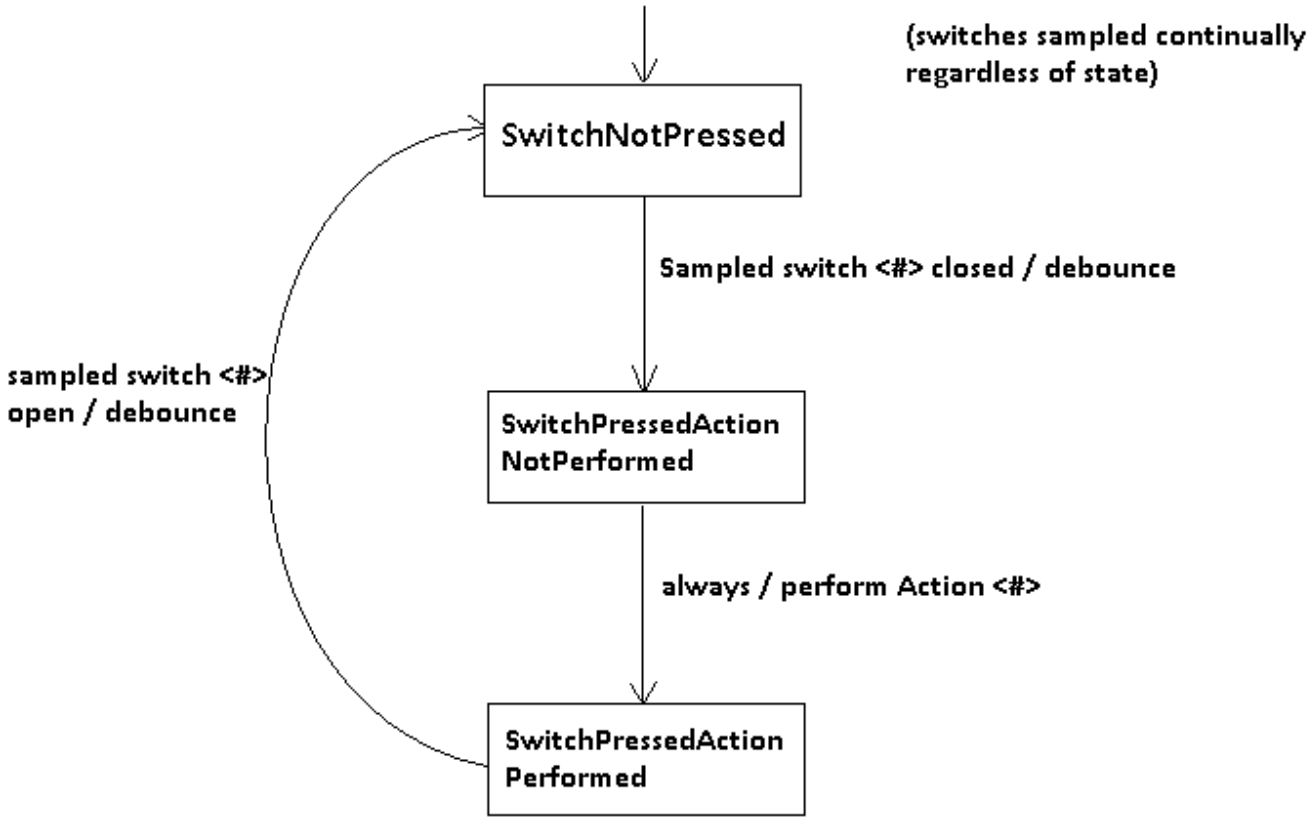
Port Configuration of Switches



B. Software

While repeating the infinite while loop, the program maintains a state regarding its switch status. The program beings in the switchNotPressed state. Each switch is sampled to see if it is being pressed. If it is, the program changes its switch status to the switchPressedActionNotPerformed state. Then the associated Action is performed once and we enter the SwitchPressedActionPerformed state. No other switches are honored or Actions performed while in this state, so the loop precedes looping and sampling as usual. When the previously pressed switch is first sampled and found open, another debounce-delay occurs and we go back to the original switchNotPressed state. Here is a diagram of the switch status state machine:

Switch Status State Diagram



Note that the process described only allows for one switch to be pressed at a time. Once we sample a switch to be closed and digress from the initial state, we will not sample switches for closed status again until we have finished the state machine circuit. This means the switch previously opened will have to be closed before we return to the initial state (SwitchNotPressed) and start sampling switches for closed status again.

For switch 11, which performs different actions depending on how long it is held, the program samples the time when the switch is pressed, waits for the switch to be released, and then samples the time again (the time is periodically updated by a timer interrupt). Then, the action to be performed is decided not only by the switch pressed but by the time for which it was held. In any mode besides Reset Mode, if switch 11 is held for 1sec it toggles Review/Test Mode, if it is held longer than 1sec it starts flashing the reset warning, and if it is held for ten seconds it will reset and go back to Reset Mode.

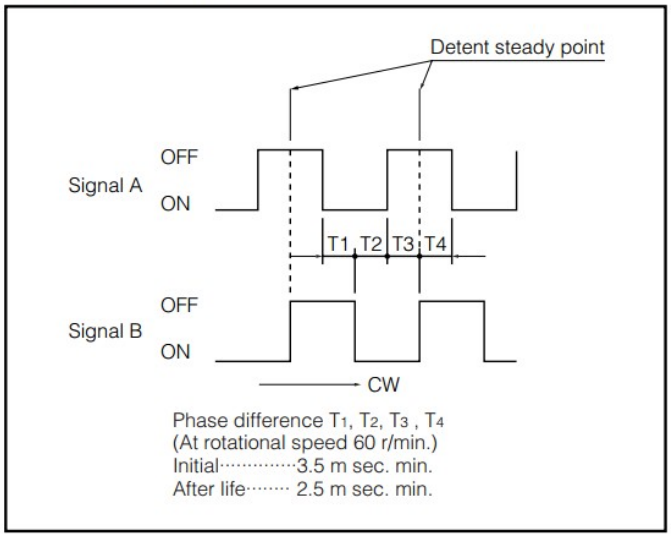
Which switches do what is described in more detail in the Modes section.

V. Rotary Encoder

A. Hardware

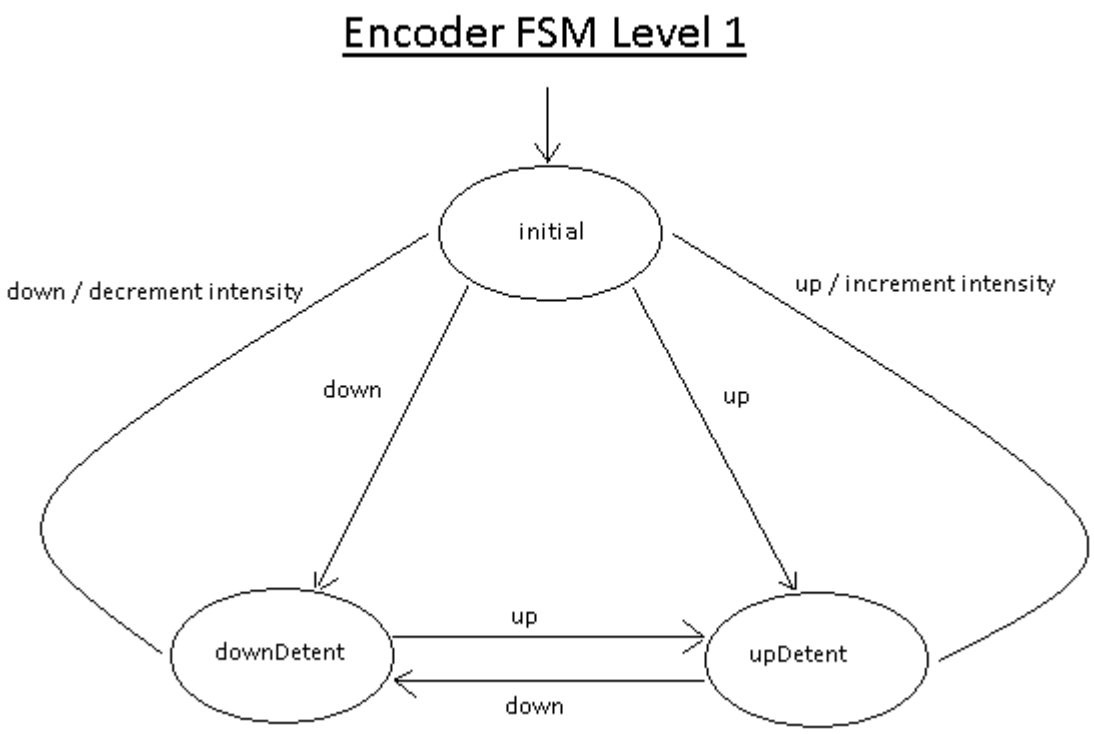
The rotary encoder has two terminals - which will be called signal A and B - connected to input ports B5 and C12. At rest, both signals are high. When turned clockwise, the combined signal proceeds through the following stages in a single detent: 01, 00, and then back to 11. The order is reversed for rotating counter-clockwise, which is how the software distinguishes the difference.

■ Phase Difference

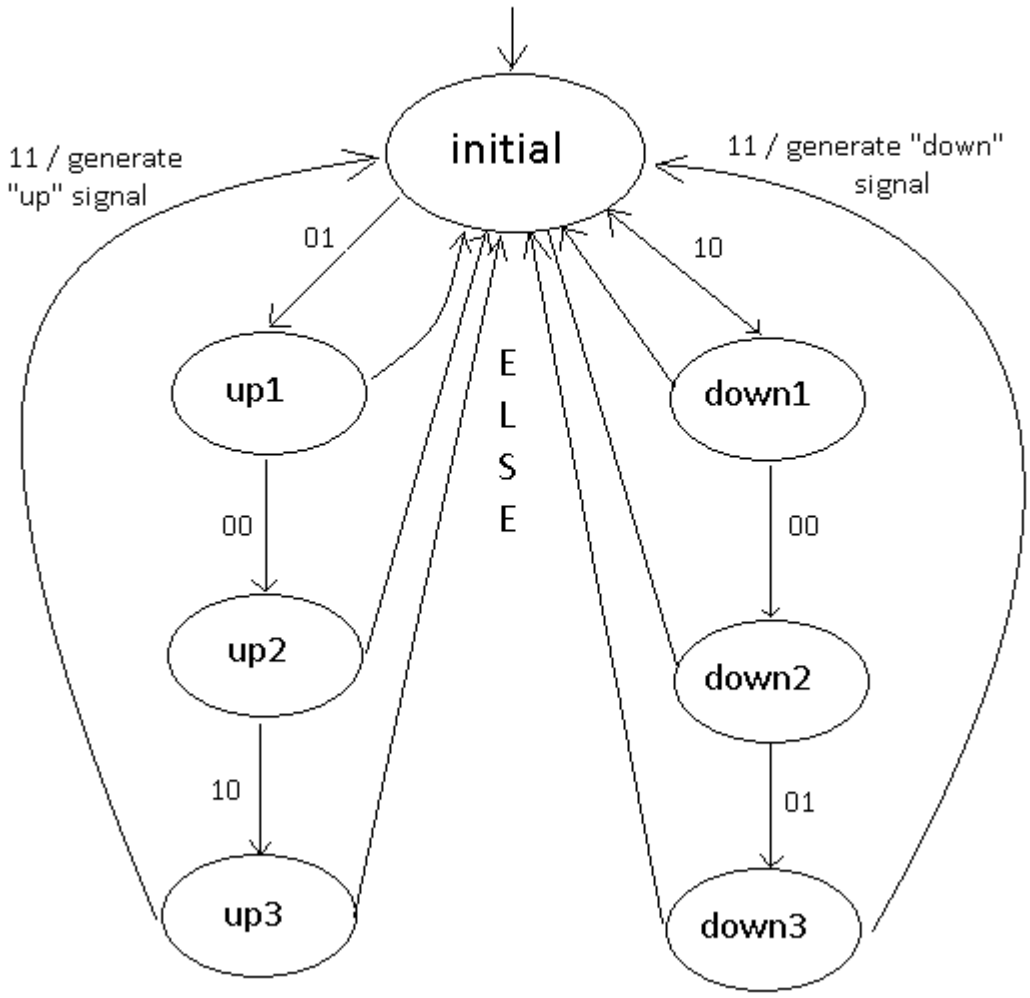


B. Software

A hierarchically nested state machine is used to track the state of the rotary encoder, based on the combined inputs signal A and B. Here is the state diagram for the first, highest, or coarsest level:



In each state at this top level, we assess the state transitions of the second level. If the second level generates the "up" signal (which signifies that the encoder has been turned by a single clockwise detent), then we precede to upDetent, etc. Basically, this diagram shows that two of these "up" signals in a row causes us to increment the intensity, and similarly for down. Now for the second level:



This state machine tracks the phase changes within one detent. A single detent is created by the combined signal sequence 10 - 00 - 01 - 11 (down-detent), or 01 - 00 - 10 - 11 (up-detent). If at any point a valid sequence is broken with an invalid signal, we return to the initial state. If we complete a valid sequence in entirety, we return to the initial state AND the appropriate "detent just occured" signal (either "up" or "down"). This is the signal sent to the upper level FSM described previously.

VI. Display

A. Hardware

The seven-segment display actually consists of four numbers, each one composed of seven LEDs (thus "seven segment"). These LEDs are in common anode configuration. This means that when you want a certain combination of segments to light up, you put logic-low at the cathodes of the individual segments you want to light up, logic-high at the individual segments you don't want to light up, and then logic-high at the common anode. That is all to display a number on a single digit seven-segment display (generally when I say display I'll be referring to all four digits, though). The six individual LEDs actually use the same common anode configuration, and are lit up in the same way.

The ARM output ports connect to the display via the D-latch "display driver". The driver "latches" the values put out by the ARM output ports (or rather I tell it do that in software), leaving the ports free to do other things. So the driver ends up powering the display, not the ARM chip.

B. Software

Various functions (i.e. user input at the buttons and encoder) change the frequency and intensity values to be displayed. These changes are written out to the display drivers from the ARM output ports via a timed interrupt, which calls a "display update" interrupt service routine every so often. Routine writes out the latest value *for a single digit* (where the six single LEDs are treated like another single digit). Every time the display update routine is called, it illuminates the next digit with the most current value, and then the next, etc. So instead of illuminating all the digits simultaneously, we iterate through them one at a time. This saves on the number of ports used, as well as power. Some brightness is sacrificed, also. There are five digits, so each one gets 20% of the duty cycle, and is therefore 20% as bright as it could be if it was just on the whole time.

The display update routine looks something like this:

1) disable drivers (there's actually two, one for anodes and one for cathodes)
2) write new values to drivers
3) latch and turn on drivers
4) ports are now free again!

Note that there will be two new values: the number to be displayed (cathode value), and the digit position (anode value).

VII. Interrupt Initialization & Operation

There are two timer interrupts fundamental to the operation of the HEARm. In order to get the interrupts working, a few things must be done. First, the address of the associated IRQ handler has to be added to the correct line in the startup code; it has to be on a certain line, because really you're adding it to a certain row in the NVIC table in memory, which contains all the addresses different interrupts go to. That is, this is the lookup table for different interrupt service routines, which is referenced by the interrupt itself when triggered. Also needed is the interrupt configuration (a routine in assembly), which can be broken down like so:

1) Enable TIM# from RCC_AP1ENR (I used interrupts tim2 and 3)
2) Set TIM# prescaler register
3) Enable update interrupt in TIM#_DIER register
4) Pre-load countdown value.
5) Configure control register.
6) Trigger update event to reload timer registers (TIM#_EGR register)
7) Enable the TIM# interrupt in the NVIC Table (IMPORTANT!)

And finally, the interrupt service routines are of course also created (in assembly).

A. TIM2 ISR (Display Update Interrupt)

This interrupt routine disables the display driver, and it writes out the new cathode and anode value from the ARM output ports to the display driver. It then writes out the necessary control bits to latch these values in the driver and enables it again. The cathode bitpattern written out is the number to be displayed, and the anode bitpattern is the digit to be illuminated (remembering only one at a time is displayed in succession).

B. TIM3 ISR (Wall Clock Update Interrupt)

This is another timer interrupt. Instead of periodically calling the display update routine, though, it calls the update_wall_clock routine. The point of this routine is pretty obvious: it just updates a global variable (wall clock) that keeps track of the current time. I use this clock variable to tell how long a button has been pressed down; this enables multi-purpose buttons which do different things depending on how long they are held. I'm, of course, talking about switch 11. When pressed for less than one second, it deletes a tone from the list of saved tones. When held down for 1 second, it toggles Test/Review Mode. When held down for five seconds, it flashes a reset warning. When held down for ten seconds it resets (enters Reset Mode).

This time sensitivity is achieved by continually referencing the time when after switch 11 has been discovered to be pressed. When the time difference between the current time and the time at which the button was originally pressed exceeds a certain amount, then we perform the relevant function. When the button is sampled as released, we reset the "time held" variable.