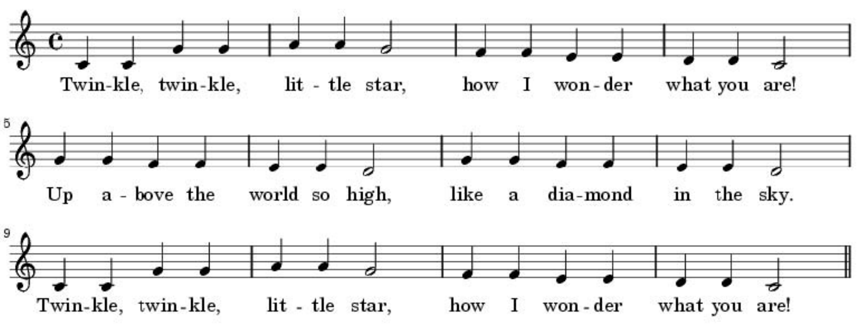
Spring 2013 - CSE 325: Embedded Microprocessor Systems

Project 4: Let's Play Some Music (and flash some lights)

Points: 10, Due Date: Mar 4, 2013


Twin-kle, twin-kle, lit - tle star, how I won - der what you are!
5
Up a - bove the world so high, like a dia-mond in the sky.
9
Twin-kle, twin-kle, lit - tle star, how I won - der what you are!

Introduction:

As far as I know there are several different musical scales where a scale consists of a sequence of notes in ascending or descending order. The notes are ordered by pitch where the pitch of a note correlates somewhat with the frequency of sound waves hitting your eardrums. According to these are the frequencies (in Hz) of the notes comprising the equal-tempered scale. We are interested in the notes B 3, C4, D4, E4, F4, G4, A4, B4, C5 and in particular, their frequencies (C4 is the well-known middle C on the piano keyboard).

PWM

If you configure the Pulse Width Modulation (PWM) module on the ColdFire microcontroller to generate an output signal at a frequency of approximately 246.94 Hz with 20% duty *and* you connect that signal to the piezoelectric speaker on the FST project board, then you should hear a tone, which corresponds to the note B3. Increasing or decreasing the duty cycle will alter the volume of the note, but I've determined that 20% sounds good to my ear.

That's what we need in order to play a musical melody. For example, above are the notes for one of my all-time favorites, Twinkle Twinkle Little Star, the notes for the first row (staff) are C4, C4, G4, G4, A4, A4, G4, F4, F4, E4, E4, D4, D4, and C4. The last two highlighted notes are a quarter note and a half note. The duration of a half note is held for twice as long as a quarter note. There are also whole, eighth, sixteenth, thirty-secondth, and sixty-fourth notes. Notes may be dotted. **You can find more information about the PWM module in Chapter 31 of the Reference Manual. You can also find information about the MIDI frequency needed for the notes by doing a Google Search.**

PIT

The Programmable Interrupt Timer (PIT) module in the MCF52259 is a timer , which can be programmed to generate, interrupts at a specific frequency. It actually consists of two timers PIT0 and PIT1. We will be configuring PIT0 to generate periodic interrupts, for the song's notes.

Out of all of the ColdFire peripherals we will work with in this course, it is one of the simplest to understand and use. There are only three registers, for this project we will be using PIT0 in "set-and-forget mode", see IMRM Section 24.3.1. For this discussion, I will just focus on PIT0. In set-and-forget mode, we configure the timer counter to start at some value (the starting value is written into PMR0[PM]) and to count down to 0. The frequency of the counter is controlled by a clock which is derived from *fbus* by dividing it by a precaler value in PCSR0[PRE]. When the counter reaches 0 and if interrupts are enabled, an interrupt request will be generated and sent to the interrupt controller (INTC) module. Interrupt requests are enabled by setting PCSR0[PIE]; the interrupt request flag is PCSR0[PIF] and the request is cleared by setting PCSR0[PIF].

Let's suppose we want to generate periodic interrupts at a frequency of *f* MHz, then the period of the interrupt requests would be 1/*f* µs. Section 24.3.3 presents a formula for calculating the timer "timeout period" which is a function of PCSR0[PRE] and PMR0[PM]

I have posted a spreadsheet on the course website (look for the link labeled PIT Spreadsheet that can be used to determine PCSR[0] and PMR0[PM] to obtain the desired interrupt frequency.

To summarize then, these are the steps that must be followed to configure period interrupts;

1. Clear PCSR0[EN] to disable the timer while it is being configured.
2. Write the selected precaler into PCSR0[PRE].
3. Set PCSR0[DBG] so the counter will stop when execution is halted in the debugger.
4. Set PCSR0[OVW] so a write to PMR0 will immediately cause the value in PCNTR0 to be set to PMR0.
5. Set PCSR0[PIE] to enable interrupts from PIT0.
6. Set PCSR0[PIF] to clear the interrupt request flag (in case it had been set for some reason).
7. Set PCSR0[RLD] so PCNTR0 will be reloaded with PMR0 when PCNTR0 reaches 0x0000.
8. Write PMR0[PM] with the desired value. This will also set PCNTR0 to PMR0 because PCSR0[OVW] was set in Step 4.
9. Determine the interrupt source for PIT0 and configure INTC to recognize and respond to interrupt requests from PIT0.
10. Place the address of the PIT0 ISR into the exception vector table.
11. Make sure to uninhibit interrupts at all levels by clearing SR[I] in your hardware initialization routine.
12. When you are ready to begin generating interrupts, start the counter by setting PCSR0[EN]. To disable, clear PCSR0[EN].

GPT

We can press a button to change the music tempo. SW1 (push button 1) is connected to port TA pin 0. Rather than using polling to check the status of this buttons, for this lab project, we are going to program the General-Purpose Timer (GPT) module to recognize when the button is pressed.

If you look at Table 2-1 in the IMRM you will see that this port pin can each be configured for one of three functions: GPT, PWM, or GPIO. To interface to the switches we will configure this port pin for the GPT function (the primary function). This is accomplished by writing the appropriate bits to PTAPAR. Note, since the pins are being configured not as GPIO pins, it is not necessary to program DDRTA.

Within the GPT, this pin is connected to GPT channels 0 (push button 1). The GPT channel can be configured as general-purpose timers, for the output compare function, or for the input capture function. We will be using the input capture mode.

Look at the GPT block diagram on page 25-2. Locate the box labeled GPTx0 Pin on the right-hand side. This box represents the microcontroller pin for GPT channel 0 (the pin connected to push button 1). The signal coming in on GPTx0 is sent to a box labeled PT0 Logic. To the left of that box is one labeled IOS0, which represents bit 0 in the GPTIOS register (see Sect 25.6.1). Look at Table 25-4 on page 25-5. Note that IOS0 and IOS1 are cleared to enable channel 0 and 1 input capture mode.

The IOS0 box also has an arrow pointing toward a box labeled Edge Detect. Note that Edge Detect is controlled by two bits labeled EDG0A and EDG0B; these are fields within the GPTCTL2 register (see Section 25.6.9). Look at Table 25-12 on page 25-10 and you will see that to detect a signal-change from low-to-high we would program GPTCTL2[EDG0] with 01 and to capture a high-to-low signal change, we would write 10 to GPTCTL2[EDG0]. If we wanted to capture any signal change (low-to-high or high-to-low) we would write 11 to this field. In Lab Project 2, I asked you to answer some questions about the push button switches; in particular, you should have noted that when one of these push buttons is pressed, the signal on the microcontroller pin would change from high to low. Therefore, to detect when the push button is pressed, we would program GPTCTL2[EDG0] with 10.

To the left of the Edge Detect box in the block diagram is a box labeled C0F. This box represents bit 0 in the CF field of register GPTFLG1 (see Sect 25.6.12). Read Table 25-15 on page 25-12 and you will see that these "channel flags" are set when an input capture event occurs, i.e., GPTFLG1[C0F] will be set when push button 1 is pressed and GPTFLG1[C1F] will be set when push button 2 is pressed.

Go back to the GPT block diagram and look for the box labeled Interrupt Logic. Notice that there are two input signals to this box labeled CxI and CxF representing signals C0I, C0F, C1I, C1F, C2I, and C2F. C0I is bit 0 of the CI field with in the GPTIE register (see Sect 25.6.10). Read the description of this field in Table 25-13 on page 25-10.

Putting all of this together, to configure the GPT so we can generate interrupts when push buttons 1 and 2 are pressed, we must:

1. Configure port TA pin 0 for the primary (GPT) function.
2. Clear GPTIOS[IOS0] to enable channel 0 for input capture.
3. Configure GPTCTL2[EDG0] to capture high-to-low signal changes, i.e., button presses.
4. Clear GPTDDR[DDRT0] so the channel 0 pin is an input pin.
5. Set GPTIE[C0I] to enable channel 0 interrupts.
6. Set GPTSCR1[GPTEN] to enable the GPT.

7. Configure the INTC module appropriately to recognize interrupts from GPT channel 0, see Section 3.3.

INTC

Read the introduction and Section 16.1 of Chapter 16. When an interrupt occurs, the interrupting device will send a signal to the ColdFire interrupt controller module. Once the processor determines the "source" of the interrupt by retrieving an 8- bit vector from the INTC, it will use this vector as an index into the exception vector table.

In CodeWarrior, the exception vector table is defined in an IDE-provided source code file named exceptions.c. Look at this file now; the table is defined starting on line 306 as an array named _vect of 256 elements where the data type of each element is vectorTableEntryType; note that vectorTableEntryType is defined on line 300 as a pointer to a void function with no parameters, i.e., void (*vectorTableEntryType)(). Therefore, the exception vector table is an array of 256 function addresses, which are the addresses of the interrupt service routines for all of the interrupt sources. By default, every element of this table (except for the first two), points to a function named asm_exception_handler(), which is defined on line 287, using inline assembly.

The location, in memory, of the exception vector table is controlled by a special supervisor mode register, VBR (Vector Base Register). Turn to Sect 3.2.6 of the IMRM for a description of this register. The upper 12-bits of VBR is used to form the base address, i.e., where the table starts in memory, and the lower 20-bits are assumed to be 0. This means that legal addresses for the table are 0x0000_0000, 0x0010_0000, 0x0020_0000, 0x0030_0000, ..., 0xFFE0_0000, or 0xFFF0_0000.

The linker command file specifies the location in memory of the table. Look in the MEMORY section starting on line 7 and note that the origin of vectorram is 0x2000_0000 and the size is 0x500 bytes. Go to line 117 and note that the LCF defines a global variable named ___VECTOR_RAM which is assigned the address of a section named .vectorram (see line 40); therefore, the value of ___VECTOR_RAM is 0x2000_0000. These global variables (such as __SP_INIT, ___IPSBAR, ___RAMBAR, and so on) defined in the LCF are accessible in our program.

Now go back to Section 16.3.8.1 in the IMRM. Table 16-6 lists the interrupt sources for each interrupt request line going into the INTC module. Scroll down in this table looking for the GPT module; note that GPT channel 0 is interrupt source 44. Thus, we need to write the address of our interrupt service routine for GPT channel 0 (push button 1) into the vector table at the locations for interrupt source 44.

Go back to Section 16.1.1.3, which discusses how the interrupt vector is determined. For INTC0 the 8-bit interrupt vector sent from INTC0 to the microprocessor is 64 + the interrupt source. For GPT channel 0 (source 44) the vector would be 108. Consequently, the address of the ISR for push button 1 must be at index 108 of the table.

Next, we must configure INTC0 registers to set the level and priority of the interrupt sources and to unmask the interrupt requests,

8. For interrupt source 44 write the level into ICR44[IL] and the priority to ICR44[IP].

9. For interrupt source 44 clear bit 12 of IMRH.

Use level 1 priority 7 for push button 1.

Now that you have had a little introduction to the Interrupt Controller Module for interrupts generated by the GPT module, you must determine how to handle interrupts generated by the PIT timer.

Project Requirements

1 The system shall store one song.

2 The song shall contain no more than 256 notes.

3 Each note shall consist of an 8-bit unsigned integer representing the pitch of the note and an 8-bit unsigned integer, which is the duration in ms of the note.

4 Pitches correspond to these 8-bit values: B_3 = 0x00, C_4 = 0x01, C_4# = 0x02, D_4 = 0x03, D_4# = 0x04, E_4 = 0x05, F_4 = 0x06, F_4# = 0x07, G_4 = 0x08, G_4# = 0x09, A_4 = 0x0A, A_4# = 0x0B, ~~B_4# = 0x0C~~, C_5 = 0x0D. No pitch is represented by 0xFF.

5 Durations correspond to these 8-bit values: thirty-secondth = 0x01, thirty-secondth dot = 0x02, sixteenth = 0x03, sixteenth dot = 0x04, eighth = 0x05, eighth dot = 0x06, quarter = 0x07, quarter dot = 0x08, half = 0x09, half dot = 0x0A, whole = 0x0B.

6 Upon starting the program, all LED's shall be turned off.

7 Simultaneously the song shall immediately begin playing.

8 At the beginning of each note, LED *n* shall be turned on.

9 At the end of each note LED *n* shall be turned off.

10 After a note is played there shall be a default pause of 1 second. During this time, the speaker will be silent and LED *n* shall be off.

11 The tempo of the song shall be controlled by push button SW1. The initial tempo of the song is 60 beats per minute (BPM).

12 When SW1 is pressed the delay decreases. The BPM should increase by 10 for each press.

13 When the BPM reaches 120 (500ms delay), each press of SW1 should decrease the BPM by 10.

14 When the BPM reaches the starting value (60 BPM), a press of SW1 causes the BPM to increase again.

Implementation

I am not going to give you a detailed software design but I am going to require that you partition your code into modules with each module implementing common, specific functionality. The required modules are:

1. FST Board Speaker Module
2. GPIO Module
3. Interrupt Controller Module
4. Main Module
5. PIT Module
6. PWM Module
7. Microcontroller Board LED Module

8. Microcontroller Board Push Button Module
9. Song Module
10. Note Module

Each module should consist of a header and source code file, the naming of these files is up to you. Please feel free to use modules from previous projects. As for the song in this project, please feel free to use anything except Twinkle, Twinkle.

What to submit for grading

Put a comment header block at the top of each source code file that contains: (1) the name of the source code file; (2) the lab project number; (3) your name (and your partner's name); (4) your email address (and your partner's email address); (5) the course number and name, CSE325 Embedded Microprocessor Systems; and (6) the semester, Spring 2013.

We will export the project to a directory structure. In CodeWarrior, click File | Export on the main menu. In the Export dialog, expand General. Click on File System. Click Next. In the next dialog, click Select All. Enter a destination directory, e.g., C:\Temp. Click Create Directory Structure for Files. Click Finish. The entire project will be exported to C:\Temp\Proj03 (or whatever name you used for your project when you created it).

Zip this entire directory naming the zip archive **cse325-s13-p04-*lastname*.zip** or **cse325-s13-p04-*lastname1-lastname2*.zip** if you worked with a partner.

Upload the zip archive to Blackboard using the project submission link by the deadline. Consult the online syllabus for the late and academic integrity policies.