UCD School of Electrical and Electronic Engineering

EEEN40280
Digital and Embedded Systems

Microcontroller Programming

Assembly Language and Interrupts

This exercise is designed to develop your assembly language programming skills, and include the use of interrupts. There are two tasks and an assignment. The assignment will be graded.

You have two weeks to complete this exercise, working in a team of two students. The report is due by 15:00 on Friday 9 February. However, you should be able to submit the report by the end of your second laboratory session. Only one report per team is required.

Hardware

The circuit board used is the EVAL-ADuC841QSZ from Analog Devices, based on an ADuC841 microcontroller. The documentation from Analog Devices is available on Blackboard, and also on the lab computers.

Extra components have been added to the boards to facilitate this exercise and another lab assignment – details of these are also available on Blackboard and on the lab computers.

Development System

Run the Keil uVision5 software. On the Project menu, choose Open Project… Navigate to the example project in My Documents\EmbeddedSystems\???day\Example. The project name is blinky. The source file (assembly language program) is in the Source folder.

Build the program using the button on the toolbar (or Project / Build target, or F7). The assembler will put the machine code in the Objects folder and the listing file in the Listings folder.

To download the program to the board, first connect the serial cable to the circuit board and to the computer (there is an extension cable already connected to the computer). Then connect the power supply to the board. The microcontroller will run whatever program is in its program memory.

To set the microcontroller into download mode, press and hold the SERIAL DOWNLOAD button on the board. Then press and release the RESET button. Then release the SERIAL DOWNLOAD button.

To download the program, click the LOAD button on the toolbar in the uVision software. If this button is not available, click the Options for Target button on the toolbar. On the Debug tab, choose the ADI Monitor Driver and make it active, then click OK. (You cannot download if the debug tool is set to Use Simulator.)

Debug

To run the example program in simulation mode, click the Options for Target button on the toolbar. On the Debug tab, choose the Use Simulator option, then click OK. On the toolbar, click the Start/Stop Debug button (or use the Debug menu).

On the debug toolbar, you have options to reset the processor, run the program or step through the program one instruction at a time. You can set breakpoints, program execution will pause at these points, so that you can examine and change registers, memory, etc.

If you step through the program, you should be able to see the registers changing. If you view Port 3 (Peripherals menu) you can see the pin connected to the LED changing state. If the next instruction is a subroutine call, you can use Step Over to execute the subroutine as one step.

You can also debug on the real microcontroller, so that your program interacts with the real hardware connected to the microcontroller. This system is not perfect – it sometimes gives unexpected results.

Exit debug mode. Change the target settings to use the hardware debug mode. In this mode, when the program is running, you cannot monitor what is going on. However, you can set breakpoints **before** you run the program (if you run the program with no breakpoints, you will not be able to get control of the processor again). When execution stops at a breakpoint, you can examine and change registers just as in simulation. You can also execute one step at a time.

Put the microcontroller in serial download mode, as for program download. Start debug mode again. Set a breakpoint at the statement where the state of the LED is about be changed. Run the program. When it stops at the breakpoint, note the state of the LED connected to port 3, pin 4, both in the debug tool and on the board. Click the Step button to execute the next instruction. The state of the LED should have changed in both places.

Now examine I/O port 2 on the Peripherals menu – check that the bit pattern matches the state of the switches connected to the port. Change the switches and check again – the status of the port will only update when you single-step or stop at a breakpoint in the program.

Task 1

Extend the example program, blink.asm, so that each time it changes the state of the LED on port 3, it also updates the LEDs on port 0 to show the state of the switches on port 2. However, the LED on port 0 bit 7 should be ON always – the switch on port 2 bit 7 should be ignored.

Note that the switches are wired so that when a switch is closed, it reads as 0. The LEDs are wired so that an output of 0 will turn on an LED. In both cases, bit 0 is on the left (when the power connector is at the top of the board) – this is the opposite of the normal arrangement.

Download and run your program. Check that it behaves as expected.

Task 2

For this task, you should create a new project. First create a folder for the project, in the working folder for this lab: My Documents\EmbeddedSystems\???day. Create a Source folder within the project folder, just as in the example project.

Download the program timer0.asm from Blackboard and put it in this new Source folder.

In the uVision software, create a new project, in the project folder that you have just created. Choose a simple project name.

Choose the Analog Devices ADuC841 processor. Do NOT include the startup code offered – that is only for programs in C.

Set the Options for Target 1: Clock frequency 11.0592 MHz. On the Debug tab, top right, select ADI Monitor Driver, and make it active. On the Utilities tab, select ADI Monitor Driver, and tick Update Target…

Expand Target 1 in the Project Workspace. Right-click on Source Group 1 and add the program timer0.asm to the project. Build the project – there should be no errors or warnings.

Examine the program – it uses one of the hardware timers to cause an interrupt at regular intervals. The interrupt service routine changes the state of P3.6. For brief information on the timer, see the lecture notes. For more detail, see the ADuC841 data sheet (page 68 in revA).

Download and run this program. Connect P3.6 to the input of the inverter that drives the piezo-electric transducer – pin 11 of the 40106 IC. Then connect the other side of the transducer (point P) to the digital ground rail (DGND). You should be able to hear a sound when the program is running.

By examining the program, calculate the expected frequency of the output signal. Measure the frequency of the output signal using a frequency meter or an oscilloscope. Is it as expected?

Assignment 1

Write a program that does two things at the same time, using interrupts.

1. Your program should generate a square-wave at an adjustable frequency on pin P3.6. Choose frequencies in the range 100 Hz to 5 kHz, to suit the piezo-electric transducer on the board. Use 3 or 4 of the switches to select the required frequency. The frequency should be derived from a hardware timer using interrupts, so that it is not affected by what is happening in the main program.

2. The main program should check another switch and enable or disable the interrupt, so that the sound can be turned on or off. Independent of the interrupts, your program should flash the LED on pin P3.4, at a rate slow enough to be seen. You may use a software delay for this.

Your program should use names or symbols where it is sensible to do so. It should have comments to explain what is happening.

Hints

There are three obvious approaches to this problem, described in outline here:

- Configure a hardware timer to produce the frequency you want. To make this adjustable, you will have to set the re-load value for the timer to different values, based on the switch settings. As the reload value sets the timer period, there will be a non-linear (inverse) relationship between reload value and frequency.
- Configure a hardware timer to give interrupts at a fixed rate, higher than the frequencies you want. Use software to count a number of these interrupts, based on the switch settings, and then change the state of the output pin. As above, this sets period rather than frequency.
- Configure a hardware timer to give interrupts at a fixed rate, much higher than the frequencies you want. Use software to add an adjustable value to a register on each interrupt. Change the state of the output pin each time the register overflows. This gives a frequency proportional to the value added.

You may use any of the hardware timers for this assignment. Timer 0 and timer 1 are essentially the same, but timer 2 has different features and capabilities. You should consider which timer is best for the solution that you have chosen. The interrupt address will be different for each timer...

If you need to do awkward calculations for a small number of possible input values, consider using a look-up table in program memory to store the answers for all the possible inputs. That is usually faster than doing the calculations each time, and can give a smaller program...

Report

Write a short report on your program for assignment 1, starting on the cover sheet provided. Explain briefly how your program works. List the frequencies produced by your program, and show how you calculated those frequencies.

Print your program for assignment 1 and attach it to your report. Submit your report in the lab, or directly to Brian Mulkeen (Room 332, Engineering).