

EECS 373 Fall 1998

Lab 5: Timers & Multiple Interrupts

Introduction:

In this lab, you will modify your lab 4 code so that all events are interrupt driven. Currently, your lab 4 code uses interrupts for serial communication, but your LED flashing is still done with a busy wait. To make the LED flashing interrupt driven, you will use timers to periodically interrupt your program and flash the LED. The interrupt method is a much more effective way to produce the delay because you waste no CPU cycles doing a busy wait.

To show that you will have greater control over the timing with an actual timer, you will use the timer to very rapidly blink the LED. Instead of reading a pattern string, a single variable will indicate the duty cycle (ratio of on-time vs. off-time) for the LED blinking. By flashing the LED rapidly, while controlling the percentage of time that the LED spends in the on state, you can control the apparent intensity (brightness) of the LED. (For example, if your duty cycle is 20%, the LED will be in an on state 20% of the time and have roughly 20% of its normal intensity.) You will implement new command-line instructions in your serial handler to control the brightness.

The MPC823 chip has many timers intended for various different purposes. In addition to the very fast timer that you will be using to control the duty cycle of the LED, there is a real-time counter (RTC) on chip. This counter ticks once per second and can be set to interrupt the processor each time the clock ticks. At program startup, you will initialize this counter to zero, and you will implement a command to print out the number of seconds since program startup.

Finally, to demonstrate the fact that a computer can accomplish many things at once, your main program will be required to generate prime numbers, and you will implement a command to display the largest prime that has been computed at that time.

Since this lab is heavily focused on working with the hardware, it is difficult to simulate this program. You should concentrate on the prelab and on developing and testing individual pieces of code separately. (For example, you should certainly be able to use the simulator to test the prime number generation code that you write. You can also test your ISR code on the simulator by manually setting the PC and other register values and stepping through it.)

NOTE: Read through this entire lab before you begin. There is a great deal of useful information available to you. Focus on understanding the big picture before you begin. If you do not get it, please see a TA in office hours.

Goals:

1. To understand how interrupt controllers are used.
2. To understand how nested interrupts work.
3. To learn about timers and their uses.
4. To learn how to make a machine accomplish several tasks at once.

Definitions:

- SIU – System Interface Unit

Chapter 12 of the manual describes this unit. This unit contains the main MPC823 interrupt controller (among other things).

- CPM – Communications Processor Module

The MPC823 chip contains a second processor (in addition to the PowerPC core) that is responsible for handling low-level communications tasks. Chapter 16 of the manual describes this piece of the processor. (Don't worry, we don't expect you to read *all* of it.)

- Nested Interrupt

A nested interrupt occurs when you are in an Interrupt Service Routine and that routine gets interrupted. Remember that when the processor enters an ISR, interrupts are disabled. You must re-enable them manually in your ISR to allow nested interrupts. You should be doing this for this lab.

Prelab:

NOTE: Important information about each of the components that you will be dealing with can be found further on in the lab.

1. Draw a block diagram showing the interrupt structure for all of the interrupts that you must be able to handle. This block diagram should include the processor, the devices and the interrupt controllers that you are working with.
2. Write C-style pseudocode that shows how your external interrupt handler will work. You only need to show the checks that you need to do to determine which interrupts are called. Remember that you always have to check the SIU on an external interrupt, and you may have to check registers in the CPM as well to determine exactly which interrupt is occurring. (See hints section.)
3. Draw individual flow charts for the handler routines that service the serial interrupt, the Real-Time Counter (RTC) interrupt, and the timer interrupt. Be detailed enough that you indicate what you must do to each register.
4. Write the assembly code required for initializing each of the devices that you will be working with.
5. Write assembly code that reads the IMMR value, and uses it as the base address to access another memory-mapped MPC823 register.
6. Assuming a 24 MHz system clock and an LED blinking rate of 10 kHz, give a range of timer input clock rates that would be reasonable for using the timer to control the LED. Remember that you need to take separate interrupts to turn the LED on and off, and that you need to time those interrupts with an accu-

racy of 1% of the blink rate. The clock rate should be high enough that you can set it once for the entire program and then control the duty cycle by changing the timer reference (limit) register value. Draw a block diagram of all the dividers that the general system clock potentially goes through before being used as the clock input for the timer (not the RTC). Choose a divider value for each of these dividers that achieves a timer clock rate within the range you calculated. Write the values in the divider blocks on your diagram, and indicate the clock frequency on each signal. For an arbitrary LED duty cycle n (in the range 0-100), what value(s) do you need to write to the timer reference register to get the appropriate interrupt timing?

7. Describe an algorithm for calculating prime numbers. Note: Your algorithm does not have to be efficient, but you should be able to get through a few thousand primes if you run your program for a bit.

Program specification:

Your program will have several separate blocks. They include the interrupt service routine, its individual handlers and the main loop of your program. The major blocks are the external interrupt ISR, the real-time counter handler, the serial I/O handler, the timer handler, and the prime number generator.

The External Interrupt ISR:

Because the PowerPC architecture has only a single external interrupt, the ISR will be responsible for determining which device caused the interrupt and calling that specific interrupt handler accordingly. Before you call any of your individual handlers, you should be saving all of your registers, including SRR0 and SRR1. Also, to allow your machine to nest interrupts, you will have to set the External Interrupt Enable (EE) bit in the MSR as early as possible in your ISR (but no earlier!).

The Real-Time Counter(RTC) handler:

The RTC handler will be responsible for handling interrupts from the real-time counter. Every time you handle the counter interrupt, you should increment a counter that you use to store the number of seconds since program startup.

The Serial I/O Handler:

The serial I/O handler for this lab will be very similar to what you did in the previous lab. You will still be able to use all of the serial functions that you called before; the only difference in your serial handler is that you will now be responsible for clearing the proper In-Service bit in the CISR.

If your program is not designed to do so now, you will have to leave the Receive Buffer Full interrupt on at all times. This means that you may potentially get receive interrupts while you are printing out a message. This requires buffering characters that are waiting to be transmitted in memory. Make your buffer a reasonable size (say 1K bytes). In the event of a buffer overrun, you may drop output data, but do not corrupt the contents of the buffer.

To test the other portions of the lab, you will be asked to add additional commands to your handler beyond the current add, sub, etc. A description of all the commands that you should implement can be found below.

Add, +	This command takes two integer parameters (signed words) and displays the sum.
Sub, -	This command takes two integer parameters (signed words) and displays the difference. (Parameter 1 – Parameter 2)
Help, ?	This command should print out a help message which describes all of the available commands. It should be multi line and take up at least 100 bytes. (This is mostly for testing purposes).
Led	This instruction takes a single parameter in the range 0-100 and alters the duty cycle of the timer accordingly. Numbers should be treated as unsigned, and numbers greater than 100 should be treated as 100. If 0 (zero) is input, disable the interrupt and turn the LED off. Remember that you will have to re-enable the interrupt in order to start the routine again.
Time	This command will print the number of seconds since program began.
Prime	This command prints out the largest prime number that has been calculated in decimal format.

The Timer Handler:

(NOTE: There are four timers on the MPC823. Use Timer1.)

This handler will be responsible for handling the interrupts from the timer. Your this handler will be responsible for resetting the timer counter register and setting the value that should be in the timer reference register.

The Prime Number Generator:

The purpose of this portion of the lab is to show that interrupts allow the processor to accomplish a lot of things at once efficiently. The main portion of the program should be continuously calculating prime numbers. You should start with 2 and keep trying to find larger and larger prime numbers. Each time you find a prime, you should save that number in memory and then search for the next larger prime number. It is not necessary for this code to be extremely efficient in finding prime numbers, but it should be able to find prime numbers in the thousands and even higher in a reasonable amount of time.

Details:

Internal Memory Map Register (IMMR)	
The IMMR is described on p12-34 of the manual.	
Most of the device registers that you will be using in this lab are memory mapped. Because there are many different ways that you may want to configure your memory map, it is possible to move the location of these registers. These registers are offset from a base register called IMMR (e.g. the SIPEND register is located at (IMMR & 0xFFFF0000 + 0x10.) The IMMR is a special purpose register (number 638), and it can be read using the mfspr instruction. You should read the value of the MSR into a register at the beginning of each handler and offset from that value.	
Machine Status Register (MSR)	
The bit assignments for the MSR can be found on p6-20	
In the MSR, you will need to Enable External interrupts, and clear the Interrupt Prefix bit. Each time you enter an ISR, the External Interrupt Enable (EE) bit will be cleared to disable all external interrupts. To allow nested interrupts, you will have to reenable the interrupts near the beginning of your ISR.	
System Clock and Reset Control Register (SCCR)	
The bit assignments for the SCCR can be found on p5-3	
The real-time counter has several options for input clocks	
In this lab, we want the RTC to trigger once every second, so to do that, we will want to use the Main Clock Oscillator (OSCM), and divide by four to properly prescale the OSCM.	
SIU Interrupt Controller	
Section 12.3 of the manual deals with the SIU Interrupt Controller	
<u>SIPEND</u> – SIU Interrupt Pending Register p12-7	This register contains a list of interrupts that are currently pending.
<u>SIMASK</u> – SIU Interrupt Mask Register p12-8	This register is used to set a mask indicating which interrupts you are interested in handling. In this lab, you are only interested in handling interrupts for the RTC and the CPM. You can clear the rest of the mask bits.
<u>SIVEC</u> – SIU Interrupt Vector Register p12-10	When handling the external register, you will check this register to determine which event triggered the external interrupt. You will use the value in INTC to vector from a base address into a jump table.
Real-Time Couter	
Section 12.7 of the manual deals with the real-time clock	
<u>RTCSC</u> – Real-Time Clock Status and Control Register p12-18	There are several things that you will have to set up in this register to configure the real-time counter. The RTCIRQ should be set so that it will trigger the Level 0 interrupt. The value that we get from table 12-1 is 00000100b The SEC bit must be cleared each time you handle the interrupt and at startup. The 38K bit should be cleared. Set the SIE bit. Clear the ALE bit Clear the RTF bit Set the RTE bit.
<u>RTC</u> – Real-Time Clock Register p12-19	You should initialize this register to zero on program startup. You can read this register at any time to determine the current number of seconds since the program has begun.

System Integration Timer Keys	
Section 5.4.2.2 describes the Keys addresses for each of the keys are on p3-4 To protect certain critical registers on the processor, you are unable to write to the registers without previously unlocking them. To unlock them, you write the key value (0xFFCAA33) to the register. Without unlocking these registers, they will remain protected, and any writes to them will fail.	
RTCSCK – Real-Time Clock Status and Control Register Key	Writing the key to this register will unlock the RTCSC register
RTCK – Real-Time Clock Register Key	Writing the key to this register will unlock the RTC register
CPM Interrupt Controller	
Section 16.15 of the manual describes the CPM Interrupt controller.	
<u>CICR</u> – CPM Interrupt Configuration Register p16-495	IRL – The Interrupt Request Level (IRL) bits are used to tell the CPM which interrupt it should trigger on the interrupt controller on the SIU. IEN – This bit enables all CPM registers.
<u>CIMR</u> – CPM Interrupt Mask Register p16-498	Timer1 – You must set this bit in the mask to enable timer1. SMC1 – This bit must be set to enable the serial interrupt. The rest of the bits must be cleared.
<u>CISR</u> – CPM Interrupt In-Service Register p16-499	Timer1 & SMC1 – You will be responsible for clearing the bit corresponding to whichever interrupt you serviced. Note: To clear a bit in the register, you must write a 1 to it.
<u>CIVR</u> – CPM Interrupt Vector Register p16-500	You can check the vector number portion of this register to determine which interrupt was called and branch based on it.
<u>CIPR</u> – CPM Interrupt Pending Register	You should clear the bits corresponding to the Timer and the serial device each time you handle the interrupt. To clear the bits, you write a 1 to the memory location.
Timer 1	
Section 16.4 in the manual describes timers. Of special interest is section 16.4.3	
<u>TGCR</u> – Timer Global Configuration Register p16-79	CAS2 – You will want to clear this bit to prevent the cascading of the timers RST1 – You will need to set this bit to enable the timer STP1 – You should clear this bit FRZ1 – This bit should be cleared.
TMRx – Timer Mode Registers p16-80	PS – You will want to set the prescaler to divide the input clock. For this lab, set this to zero. FRR – You must clear this bit to cause the timer to begin ICLK – You will want to set this to 01 ORI – You will want to set this bit to enable the interrupt. GE – This bit should be cleared.
TRRx – Timer Reference Registers p16-81	This register contains the timeout value of the timer. You will want to set this value to a number that will cause the timer interrupt to be fired. You will have to change this value each time in the ISR to create an on/off duty cycle. You will want to clear this at the beginning of your program.
TCNx – Timer Counter Registers p16-82	By clearing this register and the timer reference register, you will cause the timer to interrupt so that you can begin handling the flashing at program startup.

Notes and hints:

1. The main code for your interrupt handler should take care of the register saving, and then use if-then type statements to determine which external interrupt was handled. When you know which handler you have to work with, your program should branch to that point.
2. You can find helpful information in the white MPC823 manual on how many of these things work. If you need help, the book is a very good resource.
3. To allow the board to handle the external interrupt, you will need to enter the following command in the SDS command window: `@ der = 0xFDE7400F`

Procedure (in the lab):

1. After you have verified as much of your program as possible using the simulator, start the SingleStep OnChip debugger and download your program to the target board.
2. Test your program thoroughly. Be sure to try a variety of LED duty cycles while typing 'help' repeatedly to print the help message. This should stress your interrupt handlers the most.
3. Once you have tested your program thoroughly, use the oscilloscope to probe the voltage on the LED (BE CAREFUL). Observe and measure the duty cycle as you vary the intended duty cycle from the command line. Report your findings.
4. Print a listing file of your complete program and demonstrate it to the TA. The TA will sign your listing. Turn the signed copy in with your lab report.

Lab report: (due at the beginning of your next lab section)

1. Briefly summarize the function of your program.
2. Describe how your program operates: the individual steps and algorithms involved. Be thorough but not verbose; two or three paragraphs should be sufficient.
3. Include a well-commented listing of your program. Comments should include register usage (i.e. which variables are kept in which registers), descriptions of all symbols, and explanation of all derived expressions. If you add comments after you demonstrate to the TA, be sure to include both the commented listing and the signed listing.
4. Discuss any difficulties you may have had in getting your program to work correctly: what parts of the program were hard to write initially, what types of bugs did you have to fix, etc.
5. Discuss any limitations your program may have.