EECS 373 Winter 2004

Lab 7: Timers

Requirements

Pre-lab: Individual answers to pre-lab questions are due in your lab section during the week of March 8, 2004. Your group must also have an initial software design completed and entered into SingleStep.

In-lab: The lab demonstration sheet is due by Friday the week of March 8, 2004.

Post-lab: Answers to post-lab questions and are due in your lab section during the week of March 15, 2004.

Value: This lab is worth 5% of your total grade.

Objectives

The purposes of this lab are:

1. To learn how to configure a timer to generate interrupts at a specified frequency.
2. To increase your understanding of and experience with interrupts.

Overview

Timers provide an efficient and flexible source of regular interrupts. In this lab, you will use one of the MPC823's built-in timers to generate a cyclic pattern on the bar-graph display with a user-adjustable frequency.

Design Specification

Your system for Lab 7 will build on your Lab 6 system. You will add a "chaser" display on the bar-graph LEDs in addition to the calculator application and elapsed-time display functions. At any given time, exactly one of the LEDs in the bar-graph display will be on; at a regular interval you will turn off that LED and turn on the adjacent LED, causing the effect of seeing the lit LED "move" along the bar graph. When the lit LED reaches one end of the display, it will either wrap back around to the other end or "bounce" back in the opposite direction, as specified by the user. To control the chaser display, you will add two new commands to the terminal interface:

Command	Action
"i"	Interpret the next word on the command line as an integer indicating the desired interval between chaser display updates in milliseconds. Set the timer frequency accordingly.
"d"	Interpret the next word on the command line as a single-letter direction ("l" for left, "r" for right, or "b" for bounce), and set the direction of the chaser display as indicated. The "l" and "r" settings When the end of the bar graph is reached, the lit LED will wrap around to the other end if the direction is "l" or "r", and will switch directions if the setting is "b".

The chaser display should initially move to the right and wrap around at the end with an update interval of 500 ms. You will use one of the MPC823's built-in timers to generate regular interrupts to update the display. You must generate only the minimum number of interrupts required to update the display. That is, you must implement the ‘i’ command by reconfiguring the timer to generate interrupts at the required frequency, and not by selectively ignoring timer interrupts in software.

Hardware Details

MPC823 Timers (Section 16.4)

The MPC823 timers are part of the “communication processor module”, or CPM. (The communication processor is a complete processor, in addition to the PowerPC CPU, that handles low-level communications tasks.) Use Timer 1 for this lab. Based on the discussion in lecture and careful reading of Section 16.4, you should be able to determine the appropriate settings for nearly all the bits of the timer registers. A few clarifications:

- The STPx and FRZx bits of the TGCR should be cleared.
- If either bit in a TERx register is set and the ORI bit in the corresponding TMRx register is set, an interrupt request will be sent to the CPIC (see below). Your ISR must clear the appropriate bit in this register to turn off the interrupt request to the CPIC.
- The internal general system clock runs at 24 MHz on the MPC823FADS boards in the lab.

MPC823 CPM Interrupt Controller (Section 16.15)

The CPM has its own interrupt controller, the CPIC, which manages all of the interrupts that occur within the CPM. It forwards them via a single interrupt request line to the SIU interrupt controller that you dealt with in Lab 6 (similar to the way the SIU interrupt controller manages a number of interrupts and forwards them via a single interrupt request line to the PowerPC CPU). See Figure 16-133 on p. 16-488. As in Lab 6, your external interrupt ISR will query the SIU interrupt controller to determine the interrupt cause (pushbutton S1, pushbutton S2, the RTC, or the CPIC). If the interrupt cause turns out to be the CPIC, you should call a function that goes through a similar process to determine which of the CPIC interrupt sources is causing the interrupt. (For this lab, Timer1 is the only possible source within the CPIC, but we will be adding additional interrupts at this level in the next lab.)

Even though the CPIC does effectively the same job as the SIU interrupt controller (at a different level), the operation of the two controllers is different. The CPIC does a little more in hardware than the SIU to help implement nested interrupts. Specifically, the CISR automatically tracks the interrupt request(s) that are currently being serviced (i.e., whose ISRs are currently executing) and masks all equal- or lower-priority interrupts. In order to set the correct bit in the CISR, the CPIC must know when the CPU has recognized (acknowledged) a specific interrupt. The CPU provides this information by writing the IACK bit in the CIVR. This tells the CPIC to select the highest-priority interrupt; the CPIC will then provide that interrupt’s vector in the CIVR vector field and simultaneously set that interrupt’s CISR bit (and typically clear that interrupt’s CIPR bit as well). To make this work, the CPU (i.e., your ISR) must also clear the proper CISR bit when it finishes handling the interrupt.

Order of initialization is important for both the CPIC and the timers. In general, you cannot count on the specific values of any registers unless you set them yourself. Therefore you should never enable anything until you have it completely configured. For example, if you enable timer interrupts first and then set up the timer, the previous state of the timer may cause an interrupt before you get around to setting it up the way you want it. Section 16.4.3 spells out exactly how the timers should be initialized.

The key CPIC registers are summarized below:

CICR – CPM Interrupt Configuration Register (p. 16-495)

This register provides the overall CPIC configuration parameters. The IRL bits select the SIU interrupt input (Level0 – Level7) used by the CPIC to signal interrupts to the SIU interrupt controller. Unlike the RTC IRQ field, the level is programmes as a simple binary number. A good value to use here is 4. The IEN bit must be set to enable any CPM interrupts. None of the other bits in the register are relevant for this lab.

CIMR – CPM Interrupt Mask Register (p. 16-498)

This register masks individual CPM interrupt sources, just as the SIMASK register masks SIU interrupt sources. To enable interrupts from a particular CPM interrupt source, you must set its bit in this register. The rest of the bits should be cleared.

CISR – CPM Interrupt In-Service Register (p. 16-499)

Each ISR is responsible for clearing the bit corresponding to whichever interrupt is serviced, typically *after* the interrupt is handled. Note that, regardless of the CIMR bits, the CPIC will not generate an interrupt for a request whose priority is equal to or lower than the highest-priority bit set in the CISR (see Section 16.15.2.3).

CIVR – CPM Interrupt Vector Register (p. 16-500)

This register has the same purpose as the SIU's SIVEC register. However, to read the proper vector number, you must first acknowledge the interrupt by writing a 1 to the IACK bit. In addition to updating the vector number field, this write also automatically sets the corresponding CISR bit and (typically) clears the corresponding CIPR bit.

CIPR – CPM Interrupt Pending Register (p. 16-497)

This register indicates any pending (not yet acknowledged) CPM interrupt requests. These bits are automatically cleared either by writing the IACK bit in the CIVR or by clearing the interrupt source's event register, so you generally do not need to access this register directly.

Pre-lab Assignment

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. Fundamently, a timer consists of source clock, a counter and a comparator. It is can be used to measure time between two events and is characterized by its range and resolution of measurement. The resolution of a timer is the smallest possible interval of measurement and is limited by the period of the source clock. The range of a timer is the largest possible interval of measurement and is limited by the product of timers resolution and the counters capacity. For example, a source clock of 1 ms and an 8-bit count register would allow measurements as large as 255 ms as small as 1 ms.

   The source clock for the CPM counter clock is the MPC823FADS system clock running at 24 MHz. The ICLK and prescaler fields control the scaling of the source clock. Given these details and that the timers cannot be cascaded, answer the following questions.

   a. What is the smallest possible time interval that can be measured or resolved?

   b. What is the range of the timer for part a?

   c. What is the largest possible time interval that can be measured?

d. What is the resolution for part c?

3. To create a 1 ms timer for the lab, the TMRx prescaler field, ICLK field and TRRx register will have to be configured. Of course, there are many possible settings to make a 1 ms timer. What settings will allow for the maximum possible count in multiples of whole milliseconds?

4. Write out the values that you will use to initialize the timer registers and the order in which you will initialize them. That is, rewrite the example from Section 16.4.3 using values appropriate for this lab.

5. Repeat question 4 for the CPIC.

In-lab Procedure

1. Start by writing a program that *only* does the chaser display. You may want to copy your SIU ISR code from Lab 6 to get started. This program should be entirely interrupt driven; the main body should be initialization followed by an infinite loop. Test this program as thoroughly as possible using the simulator.

2. Download your Lab 6 hardware design and your chaser-display program and test it on the lab board.

3. Integrate your chaser-display program with your Lab 6 program. This should require merging only the initialization code and the SIU ISR code. When you merge the initialization, you must call ser_init *before* you initialize anything else.

Demonstration 1.1: Demonstrate your system for the lab instructor.

Post-lab

1. Explain how you organized your code for the chaser lights. Focus on the new functions. Be sure to discuss the implementation of the 'i' and 'd' commands: how do they interact with the timer ISR? What issues did you have to consider avoiding problems or erratic behavior here?

2. 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.

3. 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.

4. As discussed above, the CPIC is designed to make nested interrupts easier. How does the CISR help? The same thing could be done using the SIU interrupt controller. Describe exactly what the SIU ISR would have to do to mimic the behavior of the CPIC CISR. Be sure to consider multiple nested interrupts, i.e. where a low-priority ISR is interrupted by a medium-priority ISR that in turn is interrupted by a high-priority ISR.

Lab 7 Demonstration Sheet

Print this page and present it to your lab instructor when demonstrating the various lab sections. Turn this sheet in with your post lab or when your in lab demonstration is due. You are required to turn in only one demonstration sheet per group.

List Partners Names

____

____

____

D1.1 Demonstrate your working system to your lab instructor.

Lab instructors initials: