Lab 2: General-Purpose Timers and Interrupts

Goals

1. Introduction to timers to be used in the subsequent lab on interrupts
2. Familiarity with the TM4C123 vector table and the ability to modify it
3. Application of interrupts
4. Writing Interrupt Service Routines
5. Emphasizing the ability to extract information from the datasheet to correctly setup registers.

Objective of the lab

Ability to blink a port F LED at a specific rate using a general purpose timer (section A). Then using interrupts to control a GPIO and a timer (section B).

What you need for the lab

1. The EK-TM4C123 Launchpad (http://www.ti.com/tool/EK-TM4C123GXL)
2. TM4C123 data sheet (https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49165887)
3. IAR workbench or other IDE
4. LEDs (https://learn.adafruit.com/all-about-leds/the-led-datasheet)
5. Push buttons (https://www.alps.com/prod/info/E/HTML/Tact/SnapIn/SKHH/SKHHAKA010.html)
6. Debouncing (https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49719211)

Section A. General-Purpose Timers

Timers

The TM4C123GH6PMGeneral-PurposeTimer Module (GPTM) supports programmable timers that can be used to count or to drive events such as I/O, communication, or analog to digital conversions. The GPTM contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each GPTM block provides two timers (referred to as Timer A and Timer B) that can be configured to operate independently or concatenated to operate as one 32-bit or 64-bit timer. There are other timers available on the Tiva C Series microcontrollers such as the System Timer (SysTick) and the PWM timer which can be used for different application. The focus of this lab is to configure the general-purpose Timer A.

Timer Modes:

There are different timer modes that are supported by the microcontroller (please refer to section 11.3.2 of the datasheet for more information). Our objective in this lab is to set a timer to blink an LED periodically every second and therefore we will use the periodic timer mode.

Initialization and Configuration:

In order to use a GPTM successfully, you should follow the following steps as described in section 11.4 in the datasheet

1. Enable the appropriate **TIMERn** bit in the **RCGCTIMER** register. We will use Timer 0 which corresponds to bit 0 of this register.
2. Ensure the timer is disabled by assigning 0 to bit 0 of the register **GPTMCTL**
3. Write the **GPTMCFG** with a value of 0x00000000. This value 0 is to select 32-bit timer configuration
4. Configure **TnMR** field in the **GPTMTnMR** (replace n with A so the register is GPTMTAMR.
5. Configure the direction of the counter to count up or countdown. For this exercise, let's countdown. You need to configure **GPTMTnMR**
6. Load the start value into the GPTM Timer n Interval Load Register (**GPTMTnILR**). We will start the counter at 16,000,000 which is the speed of the microcontroller 16MHZ oscillator (p.57 of the datasheet) so we can have a blink every 1 second.
7. If interrupts are required, set the appropriate bits in the GPTM Interrupt Mask Register **GPTMIMR**. We will skip this step for now till interrupts are used in subsequent labs. You do not need to configure any registers for this step.
8. Set the **TnEN** bit in the **GPTMCTL** register to enable the timer and start counting. Please note that we initially disabled this in step 2 and we are enabling it here by assigning 1 to bit 0 of the register GPTMCTL
9. Poll the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the appropriate bit of the GPTM Interrupt Clear Register (**GPTMICR**). This step should tell us when the counter counted all the way down so we can use a loop to poll this register so it should be placed in the infinite while loop of the main.

Section A Required Task:

1. Update the program you used in lab 1 section A such that you blink one of the 3 LEDs every second.
2. Update the program you used in lab 1 section B such that
   2.1) Please refer to the lab1 regarding the hardware setup.
   2.2) When a button is pressed, the system will respond only if the user holds down the button at least 2 seconds.
   2.3) If the user presses the start/stop button (hold for 2 seconds), but not the passenger button, the system will start with the stop stage (where the red LED is on, and other LEDs are off). After 5 seconds, the system will move to go stage. Then wait for another 5 seconds to change from go stage to stop stage. In other words, the go and stop stage will last for 5 seconds and switch to each other.
   2.4) If the user presses the passenger button (hold for 2 seconds) to indicate a passenger tries to across the street, the system will stop the current stage and move the warn stage immediately. The warn stage will last 5 seconds, and move to stop stage.

3. Re-structure your program into functions and reduce the coding in the main to a minimum. A suggestion is to create a function to initialize the GPIO and another function to initialize the timer and call them both from the main function.

Part B. Interrupts

The Vector Table

The vector table of the TM4C123 Launchpad is shown in Figure 1, which contains the addresses and numbers of the interrupts. On system reset, the vector table is fixed at address 0x0000.0000. The privileged software can write to the Vector Table Offset (VTABLE) register to relocate the vector table starts address to a different memory location, in the range 0x0000.0400 to 0x3FFF.FC00 but it is not our objective in this lab to relocate the vector table.

When you run a program in debugger mode in IAR, you notice that the first block of Disassembly goes from addresses 0x0 to 0x3c (see Figure 2) which matches the vector table contents.

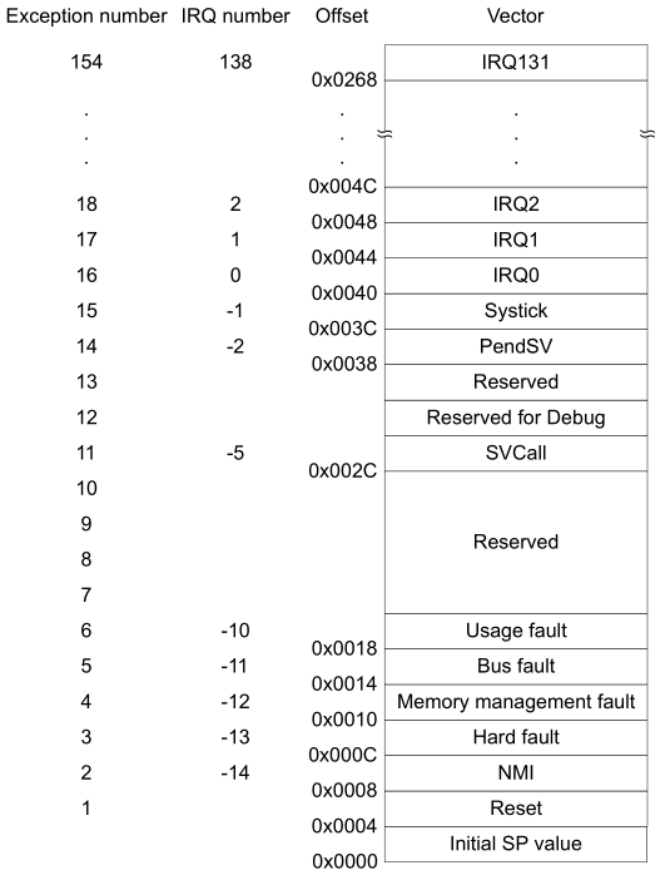

Figure 1 vector table of the TM4C123 LaunchPad (figure 2-6 of the datasheet)

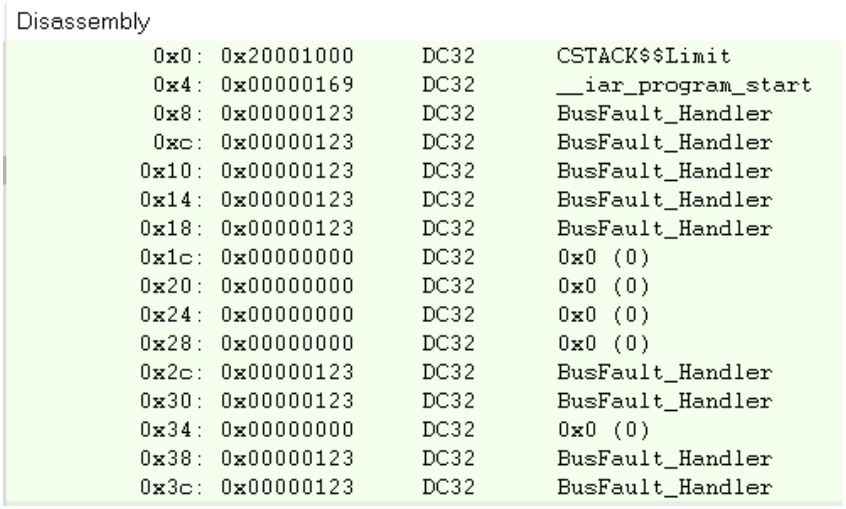

Figure 2 Vector table occupies the first block of memory addresses as seen in IAR debugger

The vector table is implemented in the cstartup_M.c file which is located under the IAR directory. The path of the file is:

C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\src\lib\thumb

If you open this file you will find the vector table defined as an array as shown in Figure 3. Notice the order of the array elements that match the data sheet in figure 1 where the "Reserved" addresses are represented as zeros. To force the vector table array into the ROM, the vector table must be declared as a constant and this is the reason for using the keyword "const". The implementation of each of the vector table handlers is also included in the same file. Figure 4 shows the prototypes of these handlers which is what we will refer to as an Interrupt Service Routine (ISR).

```
const intvec_elem __vector_table[] =
{
  { .__ptr = __sfe( "CSTACK" ) },
  __iar_program_start,

  NMI_Handler,
  HardFault_Handler,
  MemManage_Handler,
  BusFault_Handler,
  UsageFault_Handler,
  0,
  0,
  0,
  0,
  SVC_Handler,
  DebugMon_Handler,
  0,
  PendSV_Handler,
  SysTick_Handler

};
```

Figure 3 Declaration of the Vector table in the cstartup file

```
extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );
```

Figure 4 Prototypes of the ISRs

Our objective is to modify the cstartup_M.c file to add Interrupt requests IRQs to the vector table and implement the interrupt service routine(s) that corresponds to the added IRQs. However, we do NOT want to change the original cstartup_M.c file provided by the IAR software so we will make a copy of it to our IAR project file.

Task B. 1: Opening the cstartup_M.c for Editing

1. Make a copy of your lab 2 section A folder and rename it to lab 2 Section B
2. Make a copy of cstartup_M.c from its original directory to your lab3 directory. Open the workspace in IAR and add the file to your project using the "add files" options.
3. Open the cstartup_M.c file for editing. Note that when it's first opened it'll be locked for editing so right click on file name in the editor tab and choose to save it. This will bring up a message saying that saving will remove the read-only status of the file so hit ok. Now the file can be edited.

Task B. 2: Modifying the vector table

Our objective is to add an ISR to handle the timer we created in section A. Table 2-9 on page 104 of the data sheet lists the interrupts on the TM4C123 controller. The timer 0A is interrupt number 19 which corresponds to location 35 in the vector table. Therefore, the vector array in the cstartup_M.c should be appended accordingly as shown in Figure 5. Timer_Handler is the name of the ISR that we will implement. The name of the ISR is user defined so you can choose any other name for it.

```
SysTick_Handler,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
Timer_Handler
```

Figure 5 Appended vector table

Additionally you should a prototype and function implementation to Timer_Handler like the original functions included in the cstartup_M.c file. You can use the following prototype

```
extern void Timer_Handler(void);
```

and write it after the other prototypes at the top of the file and use the function definition as

```
#pragma call_graph_root = "interrupt"
__weak void Timer_Handler( void ) { while (1) {} }
```

And write it in the same location as other pragmas below the vector array

Now compile the project and open the debugger. You should see the vector table has been extended in the Disassembly window as shown in Figure 6.

Disassembly

```
0x14: 0x000001c9     DC32      BusFault_Handler
0x18: 0x000001cb     DC32      UsageFault_Handler
0x1c: 0x00000000     DC32      0x0 (0)
0x20: 0x00000000     DC32      0x0 (0)
0x24: 0x00000000     DC32      0x0 (0)
0x28: 0x00000000     DC32      0x0 (0)
0x2c: 0x000001cd     DC32      SVC_Handler
0x30: 0x000001cf     DC32      DebugMon_Handler
0x34: 0x00000000     DC32      0x0 (0)
0x38: 0x000001d1     DC32      PendSV_Handler
0x3c: 0x000001d3     DC32      SysTick_Handler
0x40: 0x00000000     DC32      0x0 (0)
0x44: 0x00000000     DC32      0x0 (0)
0x48: 0x00000000     DC32      0x0 (0)
0x4c: 0x00000000     DC32      0x0 (0)
0x50: 0x00000000     DC32      0x0 (0)
0x54: 0x00000000     DC32      0x0 (0)
0x58: 0x00000000     DC32      0x0 (0)
0x5c: 0x00000000     DC32      0x0 (0)
0x60: 0x00000000     DC32      0x0 (0)
0x64: 0x00000000     DC32      0x0 (0)
0x68: 0x00000000     DC32      0x0 (0)
0x6c: 0x00000000     DC32      0x0 (0)
0x70: 0x00000000     DC32      0x0 (0)
0x74: 0x00000000     DC32      0x0 (0)
0x78: 0x00000000     DC32      0x0 (0)
0x7c: 0x00000000     DC32      0x0 (0)
0x80: 0x00000000     DC32      0x0 (0)
0x84: 0x00000000     DC32      0x0 (0)
0x88: 0x00000000     DC32      0x0 (0)
0x8c: 0x000001d5     DC32      Timer_Handler
```

Figure 6 Modified vector table

Task B. 3: Implementing an ISR for the Timer 0A

The objective is to use interrupts to check that the timer 0A has timed out. In section A, we used polling in the while (1) loop to check for the flag and in this lab we want to move this part to an ISR.

4. In section A, we ignore the step that set up the interrupts, you need to enable that by setting the appropriate bits in the **GPTMIMR** register. The Timer 0A is interrupt number 19 so you should configure the **EN0** register accordingly as explained in the lecture slides.
5. Implement the ISR Timer_Handler in the main.c file the same way you would implement any function. Note that whenever the interrupt 19 is enabled, the function Timer_Handler will be executed without the programmer calling it. This means that there is no need to check the GPTMRIS register anymore as we did in section A.
6. Compile and run the program and you should have the same output as obtained in section A but this time you are using interrupts.

Task B. 4: Controlling the Timer from a user switch

Implement an ISR that allows switch PF0 and PF4 to interrupt the circuit. The PF0 switch when pressed should mask the timer's interrupt i.e. the timer should stop working and instead the red LED should be turned on. If switch PF4 is pressed, the timer should turn on and blinks the blue LED every 1 second.

Final Deliverables:

1. A lab report for both section A and B tasks
2. Demonstration of the circuit to your TA
3. Upload the report and source files for both sections to the canvas. One submission is expected per team.