EGH456 Embedded Systems

Laboratory Exercise 2

Timers and Interrupts

Learning Objectives

After completing this laboratory, you should be able to:

- Utilize timers in microcontrollers for event-driven execution.
- Implement interrupts and develop Interrupt Service Routines (ISRs).
- Understand and configure a watchdog timer
- Develop C programs incorporating interrupt handlers and timers.

Reference Materials

- PlatformIO IDE for VSCode Setup
- Cortex M4 Technical Reference Manual
- EK-TM4C1294XL Development Board User's Guide
- TM4C1294NCPDT Microcontroller Datasheet
- TivaWare Peripheral Driver Library User's Guide

Preparation Task: Adding `grlib` and the Touchscreen Drivers to a new project

These steps need to be completed to add graphics functionality to a new project. A touch screen interface is available in the form of a booster pack. The booster packs were created to provide fast and easy prototyping with a broad range of applications. TI provide a graphics library called *grlib*. For the graphics library to work, device specific source and header files need to be included. These are "drivers/Kentec320x240x16_ssd2119_spi.c" and "drivers/touch.c". These files need to be placed in a "src/drivers" folder in your project (you should create one if it doesn't exist). For use in PlatformIO projects, the grlib library files should also be manually added to the "lib" folder of your project. Versions of these files that are compatible with PlatformIO can be found in the previous lab *grlib_demo* example. Copy them into any new project where you wish to use the graphics library. Some changes then need to be made to your main .c file to use these libraries. Follow these steps below to add grlib and the touchscreen drivers to your new project:

1. Copy the grlib folder from the *grlib_demo* example.
2. Copy the following driver files found in your Tivaware SDK (/examples/boards/ek-tm4c1294xl-boostxl-kentec-s1/drivers) folder into your project's `drivers` folder:
   - `drivers/Kentec320x240x16_ssd2119_spi.c`
   - `drivers/Kentec320x240x16_ssd2119_spi.h`
   - `drivers/touch.c`
   - `drivers/touch.h`
3. Include the required libraries in your main C file:

   #include "grlib.h"

   #include "widget.h"

   #include "canvas.h"

   #include "drivers/Kentec320x240x16_ssd2119_spi.h"

   #include "drivers/touch.h"

4. Register the touchscreen interrupt handler by adding:

   extern void TouchScreenIntHandler(void);

   in `startup_ccs.c`, and modify the interrupt vector table to replace default interrupt function `IntDefaultHandler` of the ADC Sequence 3 interrupt vector with `TouchScreenIntHandler`.

Using UART

These steps will assist with connecting to the development board via UART.

PlatformIO has an inbuilt serial monitor that can communicate with the development board via UART and the COM ports. To open the serial monitor, select the plug symbol (Serial Monitor) on the bottom toolbar. This will automatically find available COM ports and allow you to easily set-up a virtual terminal. It is important that the communication configuration (such as baud rate, data bits, parity and handshaking) matches at both ends. The default settings for this port will be displayed when it is initialised. If you wish to use this serial monitor, make sure you use the default baud rate of 9600 when you set-up the UART in your code.

Alternatively, terminal emulation programs such as Putty and HyperTerminal can be connected to your microcontroller through serial (COM) ports, TCP/IP networks, dial-up modems etc. Lab PCs should have licensed version of this software. A free version is available for private use and downloadable from the web.

If both of these programs are not available on your lab computer, download the portable version of putty here

It is a good exercise to learn how to open a serial communication link with HyperTerminal and set parameters such as baud rate, data bits, parity and handshaking. The communication configuration should match at both ends – Hyperterminal on the host PC emulating a terminal and the program running on the application development board. The serial terminal emulated will let you type characters at the keyboard and these can be transmitted to the other end. Similarly, characters received from the other end will be displayed on the terminal which appears as a window on your PC. Typed characters can be echoed locally (default) or this can be turned off - such as for example, your application at the other end is echoing characters or messages back.

Laboratory Tasks

Task A: Timer Interrupts (2 Marks)

Download the example project *timers* from the workshop page on Canvas. Build and run the project, and examine the source code.

1. Find and examine the following macro used in the timer example:

   ```
   #define HWREGBITW(x, b)
   ```

   What is the functionality of this macro and what do the arguments do? Is this macro safe to use within an interrupt and why? **(0.25 marks)**

2. This program eventually reaches the `while(1)` loop in the main function which does not implement any code, effectively creating an infinite loop. So how does any other code execute after this point in the program? **(0.25 marks)**

3. How does the microcontroller determine when to trigger a timer interrupt event and what determines the time between these events? **(0.25 marks)**

4. What happens if two timer interrupts occur simultaneously? Explain using exception handling terminology. **(0.25 marks)**

5. Modify the program as follows: **(1 mark)**

   - Configure **Timer 1** interrupt to run at **0.125Hz** and **Timer 2** interrupt at **0.25Hz**.
   - Use **grlib** to display two counters on the touchscreen, each tracking the number of times a respective timer has triggered.
   - Make the counters update dynamically and reset properly when the a limit of 10 counts is reached.

   Note: Functions such as snprintf() from "stdio.h" are commonly used to print a variable into a string. However, the memory allocation involved in this function requires heap, which is not available in the current project configuration. You could fix the heap or use an alternative approach to update the characters stored in a string. We recommend the `usprintf` function, found in the `ustdlib` library under the `utils` folder. This will help when updating the timer counters dynamically.

Task B: Interrupts (2 Marks)

Download, build, and run the `interrupts` example project.

1. What type of interrupts are used in this example? Are they **hardware, software, or both**? Explain how each interrupt is triggered and handled. **(0.5 mark)**
2. Explain what **tail-chaining** is and how it improves interrupt performance. **(0.5 marks)**
3. Modify the code to **increase the priority of GPIO B** over the other interrupts. What effect does this change have on execution order? **(1.0 marks)**

Unassessed Extension Task: Measuring Tail-Chaining

If you want to observe the effect of tail-chaining:

- Use an **oscilloscope or logic analyzer** to measure the **off-to-on time** of GPIO A, B, and C.
- Compare the execution time for **equal priority interrupts** vs. **increasing priority interrupts**.
- What do you observe? Does tail-chaining reduce latency between consecutive interrupts?

Task C: Watchdog Timer (1 Marks)

Download, build, and run the *watchdog* example project provided on the canvas lab page.

Warning: Incorrectly modifying the watchdog timer may cause the system to enter a constant reset loop, making it difficult to upload new code. If this occurs, you may need to use a **flash erasing tool** or hold the reset button while flashing a new program to recover the microcontroller. See the teaching staff if this occurs and they can help you.

Note: This example introduces new code that uses a **software driver** for the push buttons implemented in "src/drivers/buttons.c".

1. What is the primary function of a **watchdog timer**, and why is it important in embedded systems? Provide an example of a real-world failure that a watchdog timer could prevent. **(0.25 marks)**
2. Modify the watchdog timer's period so that it triggers every **0.2Hz** instead of the default value. **(0.25 marks)**

3. Modify the watchdog timer so that it triggers a system reset if the push button is not pressed within **8 seconds** while keeping the same watchdog rate of 0.2Hz. Ensure you display two types of messages on the serial terminal before and after the system reset to show the correct modification. **(0.5 marks)**

Task D: Stopwatch (3 Marks)

Download, build, and run the *stopwatch* example project. This example provides some starting code to use timers and GPIO interrupts to implement a basic stopwatch. Two user push buttons (SW1 & SW2) will be used to trigger start, stop, and reset functions. This project builds on the *timers* example by adding:

- **A graphical display** for updating the timer on the screen using the **graphics library**.
- **Interrupt-driven button inputs** for starting, stopping, and resetting the timer.

Modifications to Implement: Your task is to familiarise yourself with the provided example program and modify it to include the following functionality:

1. **Initial Display and Instructions (0.5 marks)**
   - When the program starts, the screen should display a **welcome message** and instructions on how to use the stopwatch. The stopwatch should not be running at this stage.
2. **Start and Pause Functionality (0.5 marks)**
   - Pressing the **start button (SW1)** begins counting **in seconds and tenths of a second** (e.g., `12.3s`).
   - Pressing the **start button (SW1)** while running pauses the stopwatch, keeping the current time displayed.
3. **Pause and Resume Logic (0.5 marks)**
   - When paused:
     (a) Pressing the **user reset button (SW2)** clears the timer and returns to the **inactive mode**. **(0.5 marks)**
     (b) Pressing the **start button (SW1)** resumes counting from where it left off. **(0.5 marks)**
4. **Maximum Time Handling (0.5 marks)**
   - If the timer **reaches 20.0 seconds**, it should automatically stop and display **"Time limit reached!"** instead of reverting to inactive mode.

Assessment Criteria

Show modifications and answers to a tutor. Work in pairs or groups of three with tutor permission. Each member must participate.