Lab 3 – Interrupts and Timers

Interrupts. Serial port. Periodic timers. Real-time clock. Flexible timer module.

Introduction

Interrupts are an essential feature of a microcontroller. They enable the software to respond, in a timely fashion, to internal and external hardware events. For example, the reception and transmission of bytes via the UART is more efficient (in terms of processor time) using interrupts, rather than using a polling method. Performance is improved because tasks can be given to hardware modules which "report back" when they are finished.

The periodic interrupt timer (PIT) unit is also an essential feature of a microcontroller that needs to operate as part of a real-time system. It enables periodic tasks, such as real-time control, display updates, key scanning, etc. to be carried out. The real time clock (RTC) operates off a different power-domain to the rest of the K70 chip so that it can operate from a battery. This allows us to maintain the time even when the main power (from the USB) is off. The flexible timer module (FTM) gives us input capture, output compare, and PWM waveform generation capabilities.

Objectives

1. To use interrupts with the serial communication interface.
2. To set up a periodic timer.
3. To implement a real-time clock function.
4. To set up a software interface to timer functions.
5. To expand the implementation of the Tower serial protocol.

Equipment

- 1 TWR-K70F120M-KIT – UTS
- 1 USB cable – UTS
- NXP Kinetis Design Studio

Safety

This is a Category A laboratory experiment. Please adhere to the Category A safety guidelines (issued separately).

Cat. A lab

L3.2

Software Requirements

1. The software is to incorporate all the features of Lab 2. You should also implement or update the public functions and variables, as “advertised” in the supplied header files.
2. A HAL should be written for the four LEDs on the TWR-K70F120M board that allows them to be turned on, off, or toggled.
3. The UART software must use a fully interrupt-driven approach to sending and receiving characters, using FIFO buffers.
4. A real-time clock must be implemented using the RTC module that keeps track of the time-of-day. The user should be able to set the time via a PC (using the Tower Serial Communication Protocol). The RTC should be set up so that an interrupt occurs every second. The RTC ISR should invoke a user callback function (declared in `main.c`) that toggles the **yellow** LED and sends the current time to the PC.
5. A periodic interrupt must be implemented using the PIT with a period of 500 ms. The PIT ISR should invoke a user callback function (declared in `main.c`) that toggles the **green** LED.
6. A HAL must be written for the flexible timer module (FTM) that supports only simple output compare events for all 8 channels. The clock source for the FTM should be the fixed frequency clock (`MCGFFCLK`).
7. On reception of a valid packet from the PC, the **blue** LED must be turned on for a period of one second. This can be accomplished by using the callback facility of the `FTM` module. As an example, you can set up channel 0 of the FTM to produce an output compare interrupt 1 second ahead of the current value of `CNT` that invokes the user callback function (declared in `main.c`) which turns off the LED. You can then turn the blue LED on after a valid packet is received and start the timer linked to channel 0 (which will turn the LED off).

8. Extra commands of the Tower serial protocol to be implemented are:

Tower to PC	PC to Tower
0x0C Time	0x0C Set Time

9. Git must be used for version control. Note that version control will be assessed in Lab 5 based on the development of the software from Lab 1 through to Lab 5.

L3.4

Hints

1. For the FTM, according to section 43.5 “Reset Overview”, there is a certain sequence that should be followed after a reset condition. When your program runs and initialises the FTM, the code sequence should be:

   - Write to `CNTIN`.
   - Write to `MOD`.
   - Write to `CNT`.
   - Write to `CLKS[1:0]` (in `SC`).

2. Read Example 5.1 of the Topic Notes. Pay particular attention to how it satisfies the following:

   - The hardware peripheral’s interrupt arm bit is set.
   - The interrupt source is enabled in the NVIC.
   - A function is declared to be an ISR.
   - `main` enables interrupts via the `__EI()` macro, just before the infinite loop.
   - The vector table in `vectors.c` is manually modified to insert the address of the ISR at the right location.

3. Here’s an example of how the user callback function for the PIT would be used. In `main.c`, the user callback function needs to be defined:

   ```
   void PITCallback(void* arg)
   {
     ...
   }
   ```

   which is then passed as a parameter to initialise the PIT:

   ```
   PIT_Init(CPU_BUS_CLK_HZ, PITCallback, NULL);
   ```

   `PIT_Init` will store the callback parameters in private global variables for later use.

   In `PIT_ISR` we must have the following code to call the user’s function when the interrupt is triggered:

   ```
   ...
   // Call user function
   if (UserFunction)
     (*UserFunction)(UserArguments);
   ```

4. The UART transmitter is an output device that generates an interrupt request when it is ready. Only enable interrupts when there is something to output, otherwise you will create a "crash" because the interrupt will always be triggered by the ready condition, and you can only clear the arm bit by actually sending something.

5. The UART ISR is an example of a situation where we have to poll the interrupt source. See section 5.2.3 of the Topic Notes.

6. Using interrupts means we are now in a "multi-threaded" environment. Identify critical sections where an ISR may interrupt the work of `main`. Don't forget that `main` calls other module's code too, so those functions in other modules that operate in conjunction with ISRs will need careful analysis. You will have to protect your critical sections – read section 5.10 of the Topic Notes.

7. For the RTC, read section 6.3 of the Notes to set the oscillator up. You will need to refer to the top left corner of sheet 4 of the K70 Tower schematic (available on UTSOnline) to determine the required load capacitance of the 32.768 kHz crystal. The text "DNP" next to the crystal load capacitors stands for "Do Not Populate", which is a directive to the printed circuit assembly manufacturer to not "populate" (i.e. physically place) the load capacitors onto the board. The intention of the Tower designer is for the crystal load capacitance to be provided internally by the K70, at the value as stated on the capacitor symbols. To read the RTC in software, see p. 1400 of the K70 Reference Manual. To implement "waiting for the oscillator to be stable", you can just do a for-loop that does nothing except waste the requisite amount of time. For the time to wait, see section 6.3.3.2 of the data sheet for the chip.

L3.6

Marking

1. **The software to be assessed must reside in the remote git repository before the start of your timetabled activity on the date specified in the Timetable in the Learning Guide.**
2. **Please create a "tag" called "Lab3Submission" (no spaces allowed) to the particular commit that you want marked. Markers will then create a branch from this tag called "Lab3Marking" in order to assess it.**
3. **Software marking will be carried out in the laboratory, in the format of a code review.**
4. **Refer to the document "Software Style Guide" for more details of some of the assessment criteria.**

Assessment Criteria

Your lab will be assessed according to the following criteria:

Item	Detail	Evaluation	Mark
Opening comments / function descriptions	File headers and function descriptions are correct.	**E G A P N**	**/0.5**
Naming conventions / code structure	Names and code structure conform to the Software Style Guide.	**E G A P N**	**/0.5**
Doxygen comments	Comments for all functions, variables and modules.	**E G A P N**	**/0.5**
LED HAL	Public functions.	**E G A P N**	**/0.5**
UART interrupts	Interrupts and critical sections.	**E G A P N**	**/1.5**
RTC	Public functions.	**E G A P N**	**/1**
PIT	Public functions.	**E G A P N**	**/1**
FTM	Public functions.	**E G A P N**	**/1.5**
Protocol implementation	Protocol expanded. LED indication.	**E G A P N**	**/1**
		TOTAL	**/8**

L3.8

When we evaluate an assessment item, we will use the following criteria:

Evaluation	Mark (%)	Description
Excellent	100	**All relevant material is presented in a logical manner showing clear understanding, and sound reasoning.** For software – correct coding style, correct software architecture including: modularity; functions; parameters; and types, very efficient implementation (code and time) and/or novel (and correct) code.
Good	75	**Nearly all relevant material is presented with good organisation and understanding.** For software – mostly correct coding style, mostly correct software architecture including: modularity; functions; parameters; and types, reasonably efficient implementation (code and time).
Acceptable	50	**Most relevant material is presented with acceptable organisation and understanding.** For software – inconsistent coding style, reasonable software architecture (but could show improvement in modularity, use of functions, parameters, or types), some code may be prone to errors under certain operating conditions (e.g. input parameters) or usage, occasional inefficient or incorrect code.
Poor	25	**Little relevant material is presented and/or poor organisation or understanding.** For software – Conceptual difficulty of the underlying concepts, numerous coding style errors, functionality missing, poor software architecture, inappropriate or incorrect use of functions, parameters or types. Very inefficient and / or incorrect code.
No attempt	0	**No attempt.** For software – missing modules and/or functionality.

Oral Defence

During the assessment of your work you will be asked questions based on material which you have learnt in the subject and then used to implement the assessment task. You are expected to know exactly how your implementation works and be able to justify the design choices which you have made. If you fail to answer the questions with appropriate substance then you will be awarded **zero** for that component.