EE2028
Microcontroller Programming and Interfacing

Assignment 2 (AY2019/20 Semester 1)

ZAIHEEDA BTE ZAINI (A0187144B)

MUHAMMED ANAS S/O TARIQ (A0182402R)

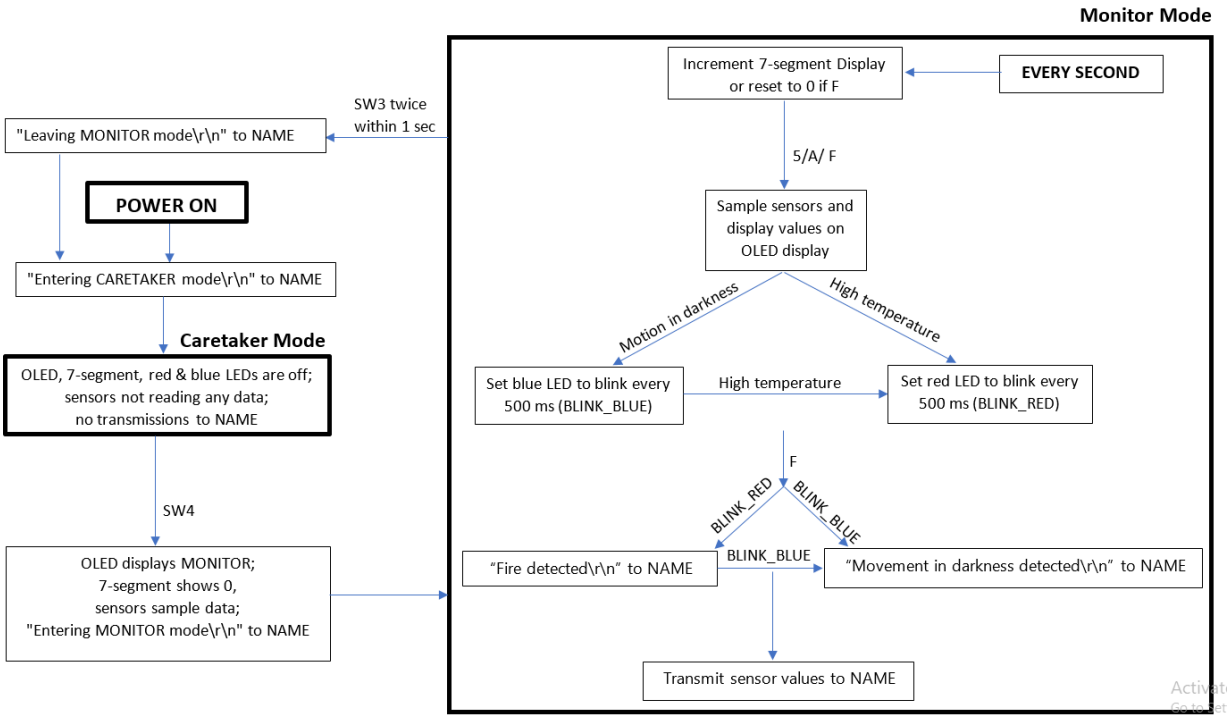
Contents

Introduction and objectives

The aim of this project is to create a care unit for the elderly. Through this project, we will be able to minimise the impact of blackouts and fire on elderly who are alone as we have made the detection of fire and blackouts possible by agency in the event whereby the caretaker is not present. Below is the list of peripherals used and their functionalities:

1. Accelerometer: Detect any movement by the elderly
2. Light Sensor: Detect the ambient light
3. Temperature Sensor: Detect ambient temperature
4. Seven Segment Display: Countdown Timer Display
5. OLED Display: Display readings of accelerometer, temperature and light
6. Blue LED: Low light warning
7. Red LED: High temperature warning
8. Switch 3 (SW3): Enter caretaker mode
9. Switch 4 (SW4): Enter monitor mode
10. NAME: UART terminal program (Tera Term) on a personal computer
11. Repetitive interrupt timer (RIT): Control blinking of red and/or blue LEDs every 500 ms
12. Timer1: Timely update of 7-segment and OLED displays, sampling of sensors and transmission to NAME

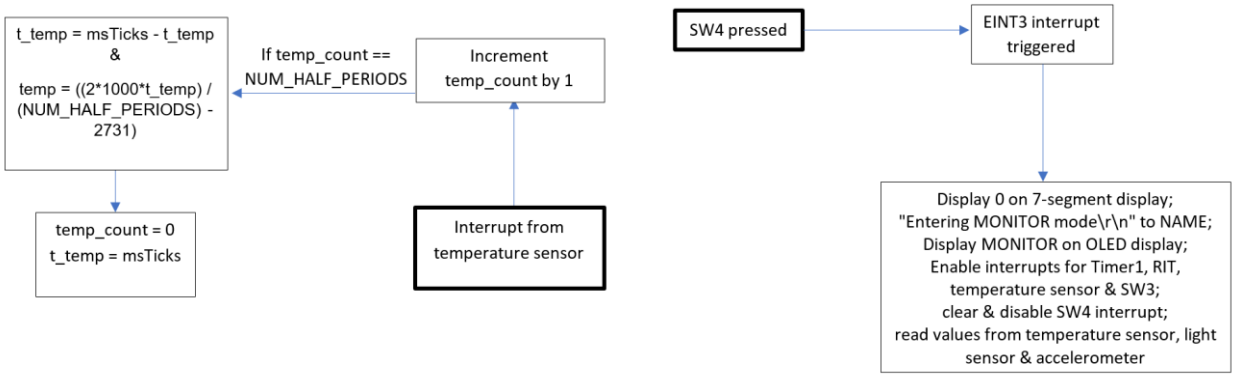
Detailed implementation & flowcharts



Overview

When CUTE is first powered on, it initialises all the required interfaces (I^2C for light sensor and accelerometer, UART for the Xbee RF module and SSP for the OLED and 7-segment displays) and peripherals (7-segment display, OLED display, accelerometer, light sensor, RGB LEDs, SysTick, Timer1 and RIT). It then sends “Entering CARETAKER mode\r\n” to NAME and goes into caretaker mode, whereby no readings are taken, displayed or transmitted and the device is essentially at an idle state. The OLED display, 7-segment display and red and blue LEDs will all be off. The GPIO interrupt for SW4 will be enabled at this point and the interrupts for SW3, temperature sensor, Timer1 and RIT will be disabled. When SW4 is pressed once, the program leaves caretaker mode and enters monitor mode, whereby the interrupts for SW3, temperature sensor, Timer1 and RIT will be enabled and the interrupt for SW4 will be disabled. Readings will then be sampled from the sensors and then every 5 seconds (7-segment shows 5, A or F) thereafter. If there is high temperature, indicative of fire, the red LED will be set to blink every 500 ms. If there is motion in darkness, the blue LED will be set to blink every 500 ms. Both LEDs would blink alternately to each other if both conditions are present at the same time; this blinking of 1 or both LED(s) will carry on for as long as CUTE remains in monitor mode. Every 15 seconds (7-segment shows F), a data transmission would occur to NAME. If the red LED is blinking, “Fire detected\r\n” would be sent first. If the blue LED is blinking, “Movement in darkness detected\r\n” would be sent next. Regardless of the blinking of the LEDs, 1 line of text comprising all the sensor readings will be sent to NAME. When SW3 is pressed twice within 1 second during monitor mode, all the changes brought on earlier by the pressing of SW4 will be reversed, bringing the system back into caretaker mode.

Interrupts for SW4 & temperature sensor (EINT3)



PIO1_4 (P1.31) has been connected horizontally to PIO2_5 (P2.5) via a jumper and a falling edge interrupt will be set on P2.5 when entering caretaker mode by setting bit 5 of the Interrupt Enable for port 2 Falling Edge (IO2IntEnF) register. As SW4 is internally connected to PIO1_4, SW4 has effectively been shorted to P2.5 and a falling edge interrupt will therefore be set for SW4. Hence, once a falling edge interrupt is triggered on P2.5, i.e. SW4 is pressed, the device enters monitor mode; the interrupts for SW3, temperature sensor, Timer1 and RIT will be enabled and the interrupt for SW4 will be cleared by setting bit 2 of the GPIO port 2 interrupt clear (IO2IntClr) register and then disabled. The sensors would be sampled, OLED display will be turned on to show the word “MONITOR” and the 7-segment display will show 0. P1.31 and P2.5 do not need to be initialised via any PINSEL configuration as by default they would be operating the GPIO function with the direction set to input.

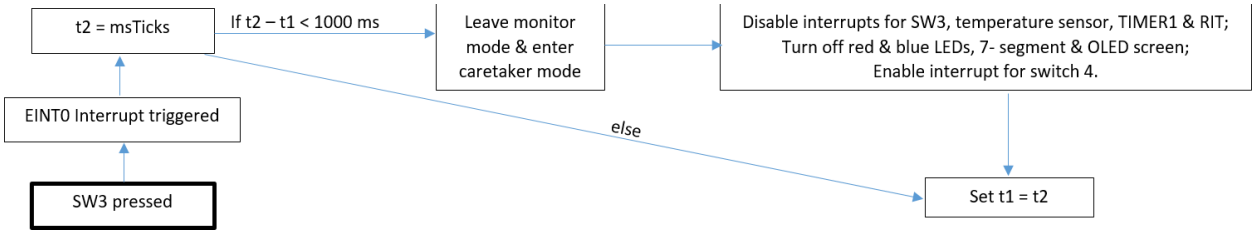
Note: Pressing SW4 during monitor mode has no effect as the interrupt on P2.5 will be disabled.

The output of the MAX6576 temperature sensor is a 50% duty cycle square wave on PIO1_5 (P0.2) with a period (in µs) that is proportional to the Kelvin temperature. By setting bit 2 of Interrupt Enable for port 0 Falling Edge (IO0IntEnF) and Interrupt Enable for port 0 Rising Edge (IO0IntEnR) when entering monitor mode, the rising and falling edge interrupts will be enabled on P0.2 in monitor mode. Similarly, by clearing those 2 bits when exiting monitor mode, the 2 interrupts will be kept disabled during caretaker mode. During monitor mode the number of edges will be counted and once it reaches NUM_HALF_PERIODS, defined as 340, i.e. ~170 cycles, the average period of 1 cycle will be computed and converted to a scalar multiple (10 in this case, due to the TS0/TS1 configuration inputs of the temperature sensor) of the temperature in degrees Celsius:

$$10 \times temperature\ in\ ℃ = \frac{(time\ taken\ for\ 170\ cycles) \times 1000}{170} - 2731$$
$$= \frac{2 \times (time\ taken\ for\ 340\ half-periods) \times 1000}{340} - 2731$$

Note: The temperature will not be recorded in caretaker mode as the interrupts on P0.2 will be disabled

Switch 3 Interrupt (EINT0)

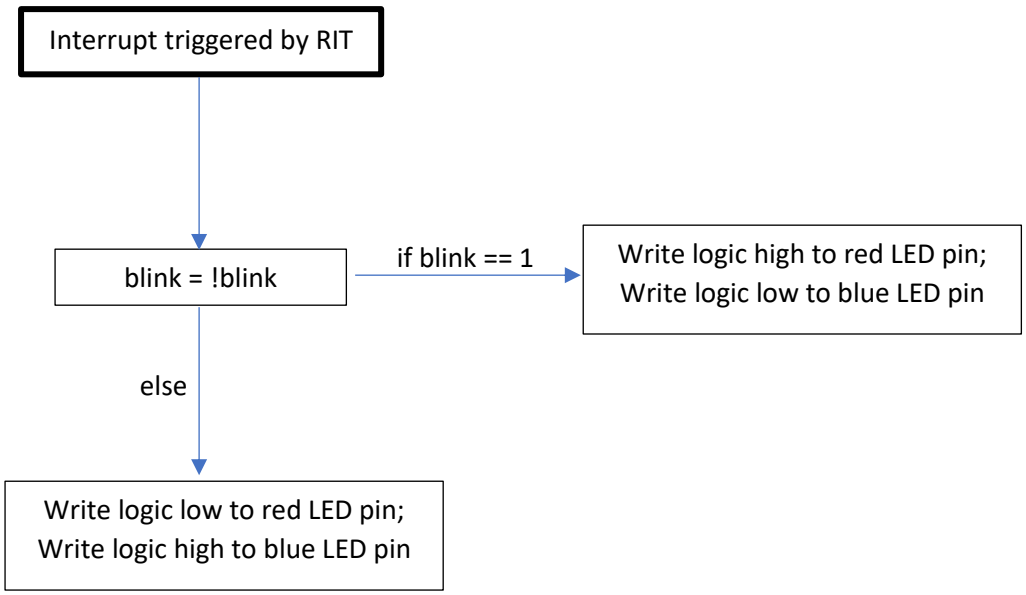


As the EINT0 interrupt could be set on SW3, the J62 jumper has been inserted in the “upper position” so as to connect SW3 to PIO2_9 (P2.10), which can be used for EINT0. Unlike SW4, SW3 (P2.10) is initialised via the PINSEL configuration so as to select function 1 (EINT0) on P2.10 and bit 0 of the External Interrupt Mode (EXTMODE) register will be set to configure the interrupt to be edge sensitive. As it is by default active-low, this results in a falling edge interrupt configuration for SW3. The EINT0 interrupt will be enabled when entering monitor mode, and disabled when leaving monitor mode. Inside the interrupt handler, t1 and t2 are 2 variables that are used to store “time”, in terms of the number of elapsed milliseconds from the start of the program (kept track of by SysTick, using the variable msTicks). When SW3 is pressed in monitor mode, the EINT0 interrupt handler is triggered and the current time is stored in t2 (the time of the previous press would already be in t1). If the difference between t2 and t1 is within 1 second, i.e. 1000 ms, the program leaves monitor mode and enters caretaker mode: the GPIO interrupt for SW4 will be enabled and the interrupts for SW3, temperature sensor, Timer1 and RIT will be disabled. The OLED display and 7-segment display would both be cleared and the red and blue LEDs will be turned off and their pins will be masked. Regardless of whether the difference between t1 and t2 is 1000, t1 will be set equal to t2 at the end to prepare for the next run of the interrupt handler.

Note: t1 is set equal to the time at the start of the program so as to ensure a proper initial value of t1

Note: Pressing SW3 during caretaker mode has no effect as its interrupt will be disabled.

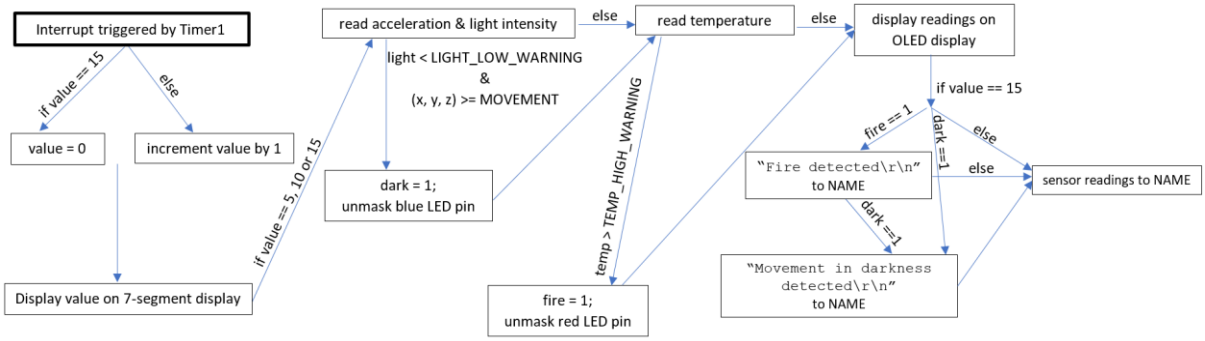
Repetitive Interrupt Timer (RIT)



The repetitive interrupt timer (RIT) features a 32-bit counter (RICOUNTER register) running from a configurable peripheral clock (PCLK) input. When the value in RICOUNTER equals to the 32-bit compare value in the compare (RICOMPVAL) register, an interrupt will automatically be generated. RIT is initialised along with the other peripherals at the start of the program. As RIT is not powered on by default, we need to set bit 16 of the Power Control for Peripherals (PCONP) register to power up RIT. The PCLK of RIT is set to be equal to the system core clock (CCLK) by setting bit 26 of the Peripheral Clock Selection 1 (PCLKSEL1) register. The count is initialised to 0 by clearing the RICOUNTER register and the maximum count value is set to be CCLK/2 (to achieve a frequency of 2 Hz) by writing the value to the RICOMPVAL register. Bits 1 and 3 of the RI Control (RICTRL) register are also set to enable the timer and to ensure that it clears whenever the value of the RICOUNTER register equals the value in RICOMPVAL. Inside the `RIT_IRQHandler`, the variable blink is toggled. The on/off status of both red and blue LEDs are toggled by writing a logic high/low to the LED pins such that the red LED follows blink and the blue LED is opposite to blink, as shown in the above diagram. This means that when both LEDs are required to be blinking at the same time, they will blink alternately. `RIT_IRQHandler` will be disabled when leaving monitor mode and enabled when entering monitor mode.

Note: Although `RIT_IRQHandler` will attempt to toggle the on/off status of both LEDs during monitor mode, whether the LEDs can be turned on/off depends on whether the GPIO pins for the LEDs have been unmasked. Under normal conditions the pins will be masked, so toggling by means of `GPIO_SetValue()` and `GPIO_ClearValue()` will have no effect and the LEDs will remain off. Following detection of fire and/or motion in darkness inside `TIMER1_IRQHandler`, the corresponding LED pin(s) will be unmasked, allowing the LED(s) to be toggled by `RIT_IRQHandler`.

Timer1 Interrupt



The timers in LPC1769 have been designed to count cycles of the peripheral clock (PCLK) or an externally-supplied clock and can optionally generate interrupts or perform other actions at specified timer values, based on match registers. As the 7-segment display needs to be updated every second, the sensors need to be sampled every 5 seconds and the transmission to NAME needs to be done every 15 seconds, Timer1 is used to generate interrupts every second during monitor mode. Similar to RIT, Timer1 is also initialised at the start to supply the same clock signal of system core clock (CCLK) to the PCLK of Timer1 by setting bit 4 of PCLKSEL0. But unlike RIT, Timer1 is always powered on by default so there is no need to set any bit of the Power Control for Peripherals (PCONP) register. The value in the match register 0 (MR0) will be automatically and continuously compared to the Timer1 Counter (TC) register and when the values are equal, actions can be triggered automatically. We therefore set MR0 to the value of CCLK and we set bits 0 and 1 of the match control register (MCR) so that when the value in MR0 equals to the value in TC, TC will be reset and an interrupt will be generated. Finally, we set bit 1 of the timer control register (TCR) so that the counter will be kept in reset until bit 0 is set, which would enable the counter for counting. `TIMER1_IRQHandler` is enabled and bit 0 of TCR is set when entering monitor mode and `TIMER1_IRQHandler` is disabled and bit 1 of TCR will be set when leaving monitor mode, so that the counter and interrupts will be enabled only during monitor mode. For every run of the interrupt handler, the 7 segment display is updated via incrementing/resetting the value variable and setting the 7-segment to display value. At every 5 runs of the interrupt handler (7-segment display is 5, A or F), the sensors are sampled, their values are written to the OLED display and the GPIO pins for the red and/blue LEDs are unmasked and flags are set if the required conditions of fire and/or motion in darkness is fulfilled. At every 15 runs of the interrupt handler (7-segment display is F), the UART transmission to NAME occurs. The flag for fire is checked and an alert is sent if the flag is set. Similarly, the flag for motion in darkness, i.e. dark, is checked after that and an alert is sent if the flag is set. Finally the sensor readings will be written to 1 line of text and transmitted to NAME.

Light sensor, accelerometer and temperature sensor

The ISL29003 light sensor contains photodiodes, which can convert the ambient light input to current outputs. It uses a 16-bit analogue-to-digital converter to convert this current output to a digital signal which is transmitted over I^2C. The `light_read()` function from the BaseBoard library manages the reading and writing of data over I^2C and converting the received data into the unit of lux.

The MMA7455L accelerometer is a surface-machined Micro Electro-mechanical System (MEMS) formed by masking and etching semiconductor materials. It can be visualized as a very small mass suspended in the centre of the device by tiny springs. When an acceleration is applied to the device, the mass deflects along one or more axes inside the device. Built-in circuitry constantly measures the amount of deflection along each axis and translates it into acceleration data which becomes available to be read by an external microcontroller [1]. The raw value generated depends on the chosen sensitivity, which can be ±2g, ±4g or ±8g, whereby g = 9.81 m/s^2 is the acceleration due to Earth's gravity. Similar to the usage of the light sensor, the `acc_read(int8_t *x,int8_t *y,int8_t *z)` function from the BaseBoard library manages the reading and writing of data over I^2C and storing the received raw value into the memory locations passed as parameters into the function.

When `TIMER1_IRQHandler` is triggered during monitor mode and when the 7-segment display shows 5, A or F, the readings from the light sensor and accelerometer are sampled by just simply calling `light_read()` and `acc_read(&x,&y,&z)`, while the reading from the temperature sensor is obtained from the value recorded by the temperature sensor's interrupt handler (EINT3), as explained on page 3. When this light intensity is less than the LIGHT_LOW_WARNING threshold value (set as 50 lx) and when the readings along any of the 3 axes of the accelerometer is more than or equal to the MOVEMENT threshold value (set as raw value 5) at the same time, the dark flag is set and the GPIO pin for the blue LED is unmasked. Similarly, when the temperature is more than the TEMP_HIGH_WARNING threshold value (set as 45 °C), the fire flag is set and the GPIO pin for the red LED is unmasked. As a result, `RIT_IRQHandler` would be able to start toggling the LED(s) from that point forward.

As the BaseBoard driver for the accelerometer uses a sensitivity of ±2g (full range = 4g) in 8-bit mode, there are 2^8 = 256 possible raw values that can be obtained from the accelerometer:

Sensitivity range	Acceleration	Output Value
±2g	-2g	-128
	-g	-64
	0	0
	+g	63
	+2g	127

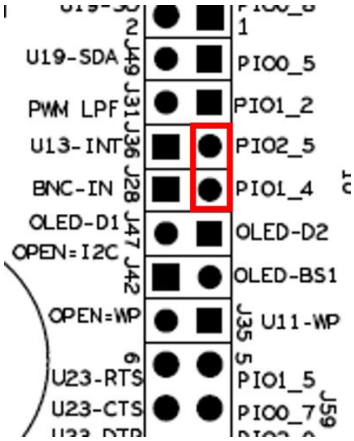
Hence, negative raw values should be interpreted as $a = \frac{raw\ value}{128} \times 2g$, while positive raw values should be interpreted as $a = \frac{raw\ value}{127} \times 2g$. But for convenience, the former or the latter can be used for both cases as 127 ≈ 128. Using units of g is simply a convenient way of relating the magnitude of a measured acceleration to the familiar force of gravity [1].

Problems and solutions (as enhancements)

In earlier versions of our program, we attempted to detect the press of SW4 via polling in a `while(1)` loop and much of our code for monitor mode operation was inside `SysTick_Handler`. Although we managed to successfully enter and exit monitor mode at the start of the program, we realised that after exiting monitor mode, we were usually unable to re-enter monitor mode. In addition, the program was slow (7 segment display did not update exactly at 1 second intervals) and it took multiple presses of SW4 before it was detected. Furthermore, upon looking through the data sheet of the MAX6576 temperature sensor and the library code, we found that that the library function `temp_read()` was inefficient as it stalled the program just to wait for 170 square wave cycles to complete.

We realised that the best solution to the above issues was to completely remove the `while(1)` loop and instead use interrupts to run the whole program so that we could spread the load from `SysTick_Handler` into multiple shorter interrupt handlers, each handling a smaller and more specific task. Hence, we added the repetitive interrupt timer (RIT) to control the blinking of the LEDs, TIMER1 to control the 7-segment display, OLED display and UART transmission and GPIO interrupts to compute the temperature and detect SW4. **The functionality, efficiency and responsiveness of our system has therefore been enhanced by replacing the `while(1)` loop with multiple interrupts.**

GPIO interrupt for SW4



As SW4 is internally connected to P1.31 (via PIO1_4), we knew that it was impossible to set a GPIO interrupt on that pin directly. After looking into the schematics and user manuals, we realised that the adjacent pin, PIO2_5 (P2.5), had already been set as a GPIO interrupt for the light sensor. But as light sensor readings can be obtained rather quickly, we felt that interrupts were not necessary for the light sensor and instead opted to use it for detection of SW4 instead, which was of higher priority. Hence, we connected PIO2_5 to PIO1_4 horizontally using the unused jumper from J28. We therefore effectively shorted SW4 to P2.5. By using the existing GPIO interrupt on that pin, we were then able to detect the press of SW4 instantly through interrupts and it solved the issue of not being able to re-enter monitor mode!

Repetitive interrupt timer (RIT)

As the red and blue LEDs need to be toggled every 500 ms, we configured RIT to generate interrupts with a frequency of 2 Hz (period = 0.5 sec) during monitor mode, such that for each interrupt, the on/off status of the LEDs will toggle (if their GPIO pins have been unmasked).

Timer1

As the updating of the 7-segment display, OLED display, sampling of sensors and transmissions to NAME must occur at regular and accurate intervals, we opted to use Timer1 to generate interrupts every second to manage the above.

GPIO interrupt for temperature sensor

The output of the MAX6576 temperature sensor is a 50% duty cycle square wave with a period (in µs) that is proportional to the Kelvin temperature. The `temp_read` function provided in the baseboard library however uses `for` and `while` loops to wait until a large number of cycles have been generated before attempting to calculate the average period of 1 cycle. This method, although functional, is highly inefficient as it stalls the entire program while waiting for a certain number of cycles to finish and the program does nothing significant during this time. We therefore opted to code our own driver using GPIO interrupts instead, so that instead of stalling the whole program during the computation

of temperature, we just simply keep track of the number of half-cycles (edges), whenever they occur and performing the actual calculation only when the required number of cycles have been generated.

Wired UART issue

However, we were still plagued by a small issue: we were unable to use a baud rate of 115200 for our wired UART transmission even though UART was properly initialised and configured both in our code and in our terminal program (Tera Term/PuTTY). After verifying with the teaching assistant (TA) that our code was correct, we were very frustrated and puzzled as to why baud rates of 115200 and above would not work. We then suspected that the USB cable might have an issue and requested for a replacement cable. Just as we had suspected, it turned out that the USB cable was the issue; the new cable enabled us to use the baud rate of 115200.

Feedback and conclusion

The TA in the lab was very informative and really went all out to ensure that the students really understand the lesson of the day. However, some of the lab assistants were not fluent in English so they were not able to understand and address our questions. Like mentioned earlier, the old equipment caused us unnecessary frustration and a waste of time in trying to figure out the cause of the issues till we changed the USB cable that was worn out due to wear and tear. Although the project is quite challenging for students who are not so proficient at coding, it has definitely been a great exposure for us. There were quite a number of technical problems that we faced but with the knowledge imparted to us by the lecturer and the TA, we managed to solve the issues even if it took us a few days to solve them. We also learnt a few new skills in electrical engineering such as thinking out of the box and importance of doing research and experimentation on our own instead of just relying on the content covered in the lectures and labs. For example, we discovered a way to use interrupts for both switch 4 and the temperature sensor by experimenting with how pins can be connected together via a jumper and by reading the relevant user guides and data sheets. It made us realise that we have to put in the effort to go in search of information and be courageous to try out new ideas in order to improve our system. That was how we were able to discover the different interrupts available for us to make our code more efficient. All in all, we had a mostly great experience with this project.

References

[1] Parallax Inc, “28526-MMA7455-3-Axis-Accelerometer-Documentation-v1.1,” 2009.

[2] Freescale Semiconductor, “MMA7455L,” 2009.