Lab 4: Multithreaded, Interrupt-Driven Multi-sensor Detection

Contents

Abstract

This project involves the design of a multithreaded system on the STM32F4DISCOVERY board. The system concurrently measures the outputs of a temperature sensor and an accelerometer, the latter using DMA. The user can choose between the two modes by tapping on the board. There are two functionalities for each one of these mode which can be selected by pressing the user button. In the temperature mode, the LEDs switch between displaying the changing trend in temperature and flashing. In the accelerometer mode, they are used to determine the edge making the largest positive angle with the horizontal, and to track the direction in which the board is moving. The data is filtered using a moving average filter before being displayed. A simple subtraction gravity filter was found to be good enough when displaying motion. Our timing method was also verfied as accurate.

1 Problem Statement

We were required to develop a program which uses the CMSIS real-time operating system (RTOS) to manage multiple threads. This program has two modes; each of these two modes has two sub-modes for a total of four sub modes. The top level modes are toggled by tapping the device (detected by the accelerometer) while the lower level modes are toggled with a button press. Note that the program is always in one of the sub-modes (i.e. it can't just be in a top-level mode, it must be in a sub-mode which itself is part of a top-level mode).

- Temperature Mode
  - Flashing
  - Temperature
- Accelerometer Mode
  - Angle
  - Motion

Both the temperature and accelerometer functionality must be running concurrently in seperate threads. The current mode only dictates which functiality is displayed on the LEDs. Proper use of OS features such as signals and semaphores must be used to prevent any thread from busy-waiting.

1.1 Temperature mode

This mode has identical functionality to Lab 2.

1.1.1 LED flashing

All the LEDs toggle on and off in unison and at a constant frequency.

1.1.2 Temperature display

The LEDs are used to display the temperature. For each additional 2 degrees celcius, another LED lights up. The display "wraps-around" when there aren't any LEDs left causing all LEDs to turn off. Since there are 4 LEDs, this wrap-around will occur every 8 degrees celcius. The temperature must be read at a frequency of 20 Hz.

1.2 Accelerometer mode

The accelerometer data must be read by the DMA chip. The DMA must be set up to trigger an interupt upon completion which signals the availiability of the data.

1.2.1 Tilt display

If any of the board makes an angle of more than 10 degrees with respect to the horizontal, the LED in line with the direction of the board making the negative angle (below the horizon) should turn on. There can be two LEDs on at the same time if both the pitch and the roll of the board are greater than 10 degrees.

1.2.2 Motion display

LEDs are to be used to display information about the board motion. We are free to come up with the specifics. Gravity has to be filtered out such that motion is displayed the same regardless of board orientation.

1.3 Challenges

The main challenges of this lab were comming up with a good way to filter out gravity in the motion sub-mode and designing how the different functionality will be seperated into threads and how said threads will communicate.

2 Theory and Hypothesis

2.1 Theory

Looking at the theory behind the following systems: Direct Memory Access and CMSIS-RTOS.

2.1.1 Direct Memory Access

Direct memory access (DMA) is used to provide high-speed data transfer between peripherals and memory and between memory and memory while keeping the CPU resources free for other operations. The stm32f4xx_dma library is used to configure the system.

There are two DMA controllers with 8 streams each. Each stream can have up to 8 channels requests and an arbiter is used to handle the priority between DMA requests.

Each stream has 32 FIFO buffers that can be used in FIFO mode or direct mode, and a FIFO threshold level that can be chosen between full, ¾ full, ½ full or ¼ full. For each channel and stream needed, the user must determine the base address of the peripheral used. The peripheral address register can be set to be incremented or not, and likewise for the memory address register. The user can select the width of the peripheral and memory data to be a byte, half a word, or a word. A circular or normal mode can be chosen. Lastly, the user can determine the amount of data to be transferred in a single non interruptable transaction for the peripheral and memory. The last option is only possible of the address increment mode is enabled. [1]

2.1.2 CMSIS-RTOS

The CMSIS-RTOS API is a RTOS interface for Cortex-M processor devices. The API provides thread management functionality as well as semaphore management. Thread management allows us to define create and control threads.

We can define a variety of attributes of a created thread. The initial priority of the thread can be selected, the number of possible thread instances, and the stack size requirements for the thread. Semaphores can be created and we can select the number of available resources. We can use the semaphores to wait on used tokens for a time specified. [2]

2.2 Hypothesis

The accelerometer and temperature sensors are noisy due to inaccuracies introduced by design limitations, manufacturing technologies, external noise and other sources of error. We will be calibrating the accelerometer to compensate for these errors through off-line calibration as well as using a moving average filter for both peripherals. However, we do not expect to get perfect temperature readings or perfect tap detection. We also believe that moving the board with certain intensity may result in false tap detection. We do believe that we user button push will not be detected as a tap, and that we will be able to remove enough noise by selecting an appropriate window width for the filter, as well as through calibration. We also believe that our sensor will be able to detect tilt angles and acceleration angle with small inaccuracies, and that the tapping will work relatively well. We are anticipating the LED peripherals to respond as expected.

3 Implementation

This lab required a new way of architechting code as we needed to exploit concurrency. Our first iteration of the program architecture used osTimers to time the threads. There were two osTimers, one for accelerometer mode and the other for temperature mode. Although they made the design simple and it even worked, we found out that osTimers used soft timers and so didn't have the necessary timing guarentees. To remedy this, we switched to pure osThreads with timing regulated by the TIM3 hardware timer. The TIM3 interrupt handler would set signals flags on each of the two mode threads at appropriate times.

3.1 Design process

The first step of the design process was to determine how the program would switch between modes and how the threads would coordinate with each other to not collide when accessing shared resources such as the LEDs or the button. Once this was done we were able to focus on implementing each mode. The DMA functionality was the trickiest and was developed on it's own in order to isolate any problems we had developing it with problems with the rest of our program. The temperature mode had identical functionality to lab 2, so we just had to restucture it a bit to work as a thread and add some semaphore and signal functionality to allow it to be thread safe and prevent it from busy waiting.

After writing the temperature mode, we used it as a basis for the accelerometer mode. We first did the tilt display as it was the easiest and we then did the motion display. While working on the accelerometer mode, we disabled the top-level mode toggling on tap as we had problems with our accelerometer being very sensitive and detecting a tap when it was really just moving from side to side.

3.2 Program organization

The program consists of three ordinary threads and three interrupt handler threads. Here they are listed.

- Ordinary threads
  - main thread
  - accelerometer mode thread
  - temperature mode thread
- Interrupt handler threads
  - TIM3 tick
  - Tap detected
  - DMA completed

The program flow and the interaction between the threads is summarized in Figure 1 and Figure 2 (they needed to be separated because of their size. The main thread moves access to the IO (LEDs and button) between the two mode threads by holding and releasing two semaphores.

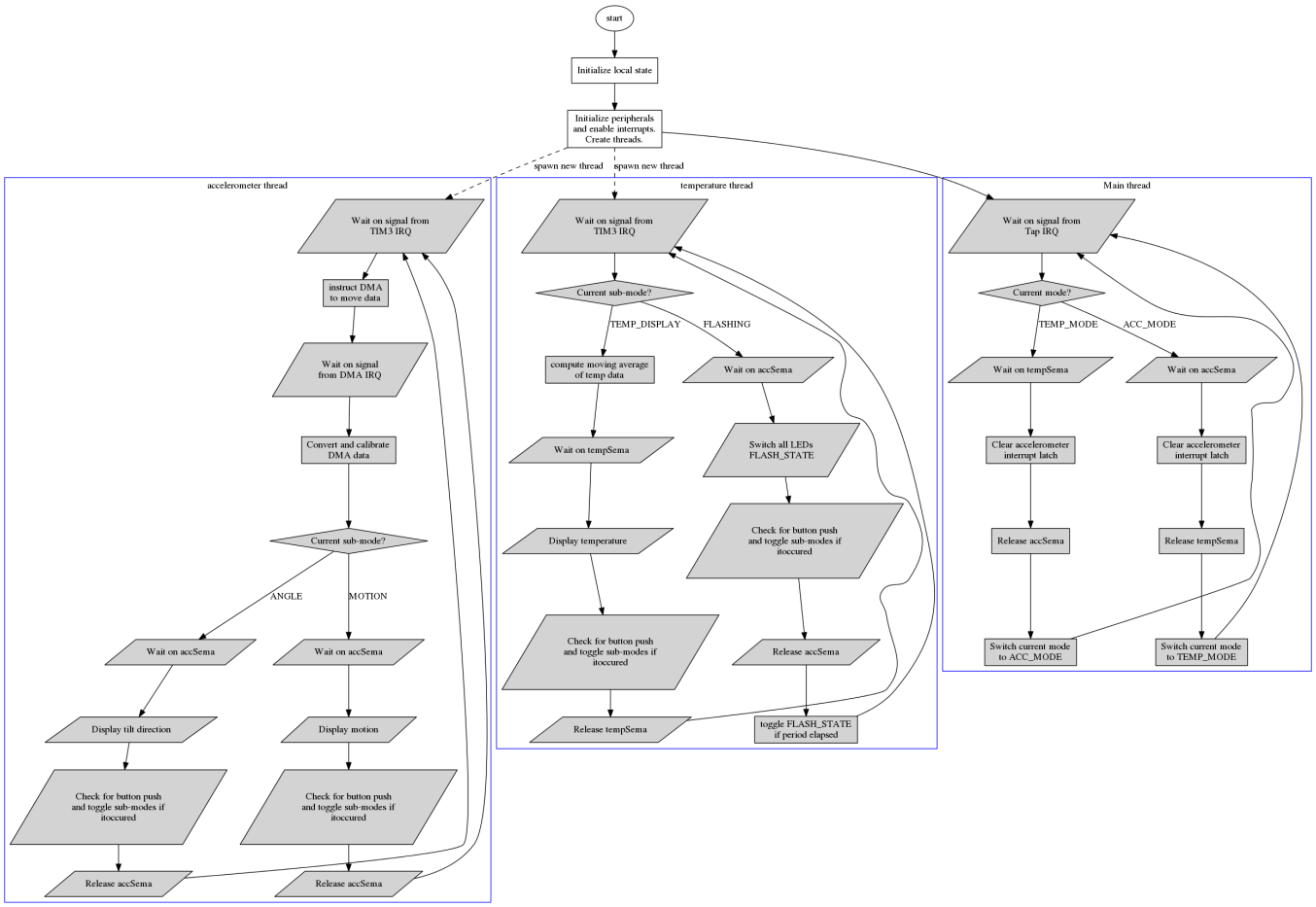

Figure 1: Top level ordinary threads

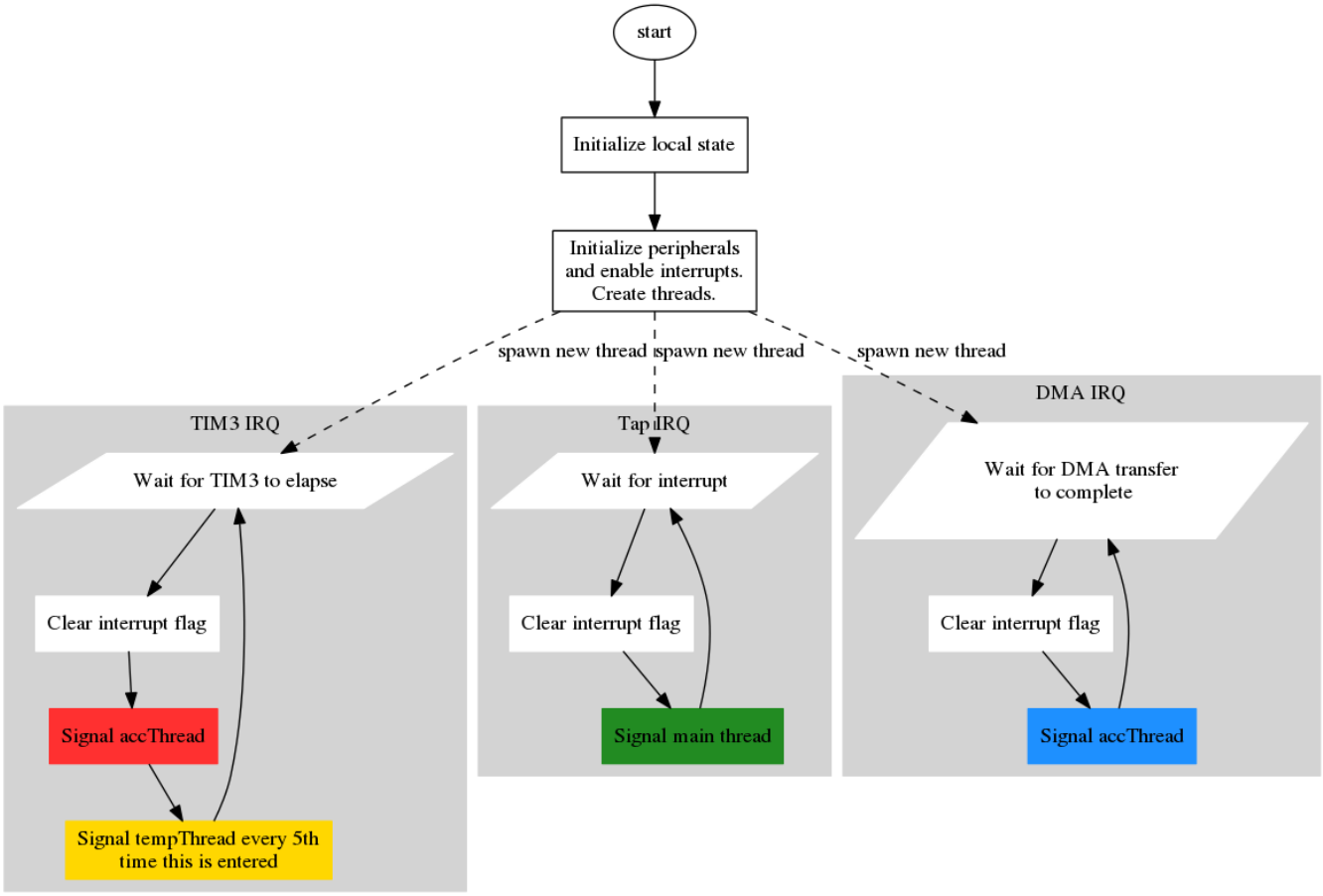

Figure 2: Top level interrupt threads

3.3 Gravity Filtering

Gravity was fitered by subtracting the most recent accelerometer value from the moving average of past accelerometer values. Although this method is crude, it worked well enough and was extremely simple to implement. The added computational cost was a single substraction.

3.4 System configuration

Direct Memory Access was initialized with specific parameters in order to read from the accelerometer. Please refer to previous labs for the initialization parameters for the other peripherals and interrupts.

3.4.1 DMA

We choose DMA2 because that is the one that is linked to SPI1 transfer lines. Channel 3 is selected, and both stream 0 and 3 are initialized because we need access to the SPI1 transfer and receiving lines to access the accelerometer. We can see the DMA2 request mapping in table 1. The following settings are applicable to both streams. We choose the peripheral base address to point to the location of the SPI1 bus since this is the one in concern. We do not increase the peripheral base address because we want to maintain the base address for all transfers. However, the memory base address is incremented for the transfer. Because the data coming from the accelerometer is 8 bits long, we choose a peripheral and data size to be a byte long. A normal mode is selected rather than a circular one so that we control when the repeat occurs, and both streams are set to a high priority. We are using a direct mode for this application therefore threshold altering parameters are not modified.

In this system, the order of priority of interrupts is the timer, followed by tapping and accelerometer. We want to make sure our system is running at a regular interval. Putting any other interrupt before the timer will disrupt the regularity.

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2				DCMI
Channel 2	ADC3	ADC3				CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4			USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX				USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3			TIM8_CH4 TIM8_TRIG TIM8_COM

Figure 3: Table 1 - DMA2 request mapping [1]

The data from the peripheral is entered into a receive array once an interrupt occurs, while the data to the memory is entered into the send array. Before beginning data transfer, we must specify the address to start reading from the SPI. This is done by filling the first byte of the send array with the address of the accelerometer's output data register.

Once we are ready to retrieve data, we enable DMA for transmission or receiving, and wait for an interrupt to occur. Once this is done, the data is automatically filled into the receive register, and the transmission and receiving is disabled.

4 Testing and Observations

To test the system, we ran the program on the board while trying to throw all the corner cases at it. We tapped and shook the device many times while also pushing buttons and made sure it was still behaving properly.

4.1 tempMode

Because the functionality hadn't changed since lab 2, we used the same testing procedure as back then.

4.2 accMode

The tilt display was easy enough to test: tilt it and make sure the correct LEDs light up. We tapped it to make sure the filter waas working properly (LED wouldn't flicker). It passed all the tests.

The motion display was tested by moving it and making sure the correct LEDs light up. We were having issues with motion being detected as taps so we disabled top-level mode toggling for this part. We put the board in various orientations to make sure gravity was being filtered out. Because of our crude gravity filtering mechanism, the LEDs occasionally flickered, but this was rare and performance was in our opinion satisfactory.

5 Conclusion

This project involves the design of a multithreaded system on the STM32F4DISCOVERY board. The system concurrently measures the outputs of a temperature sensor and an accelerometer, the latter using DMA. The user can switch between these modes by tapping on the board. 4 user LEDs are turned on with a specific encoding depending on the functionality setting in each mode. A simple subtraction gravity filter was found to be good enough when displaying motion. Our timing method was also verfied as accurate.

6 Text References

[1] RM0090 Reference manual. STM32F405xx, STM32F405xx, STM32F405xx and STM32F405xx advanced ARM-based 32-bit MCUs . Section 8.

[2] http://bennahill.com/docs/cmsis/CMSIS/Documentation/RTOS/html/. Date last accessed 20/03/2012.