EMBEDDED SYSTEMS LABORATORY

LABORATORY MANUAL

Interrupting the MSP432

AY 22/23

OBJECTIVES

- Learn the difference between polling and interrupts
- Configure and implement hardware interrupts
- Setup and configure the IR-based sensor as a wheel encoder

EQUIPMENT

- Computer or laptop that supports Code Composer Studio (CCS) 9.3 IDE
- Texas Instrument MSP-EXP432P401R LaunchPad Development Kit
- Micro-USB Cable

NOTE:

- Only students wearing fully covered shoes are allowed in the lab due to safety.
- Bring your laptop with Code Composer Studio installed.
- For your understanding and better quiz preparation, take note of the given tasks, especially questions or unexpected code behaviour.

Introduction

Embedded systems can interact with their surrounding environment in many ways. Whether sensing or actuating, there are two primary mechanisms initiating actions: **event-triggered** and **time-triggered**. In this lab, we learn how to use interrupts which are the building blocks for **event-triggered** functionality. Lastly, we will develop code that incorporates interrupts to use the IR sensor as a wheel encoder for measuring the wheel rotation speed.

Task 1: Introduction to Interrupts

A hardware interrupt is an electronic signal that alerts the microprocessor of an event. An interrupt can be triggered by either an internal peripheral (e.g. timer) or an external device (e.g. button).

In the previous lab session, polling was used to detect when a button was pressed. When polling is used, the microprocessor repeatedly checks whether the event has occurred. In the case of a button, the value of the GPIO pin is read to determine if it is high (unpressed) or low (pressed). Once the button is pressed, the microprocessor detects it quickly since it is always active and does nothing but check this single condition, as shown in Figure 1 (bottom).

While polling is a simple way to check for state changes, there's a cost. If the checking interval is too long, there can be a long lag between occurrence and Detection, and you may fail to see the change entirely if the state changes back before you check. A shorter interval will get faster and more reliable Detection and consume much more processing time and power since many more checks will come back negative.

An alternative is to configure an interrupt on the button's GPIO pin such that when a pre-configured trigger condition is met, an interrupt is generated. With this approach, the microprocessor can enter a low-power sleep state and be woken up with the interrupt. The MSP432 can detect signal changes (e.g. rising or falling edges) to generate an interrupt, as shown in Figure 1 (top). If a GPIO pin is configured to be pulled up, a falling edge will occur when the button is pressed, pulling the pin to the ground. Interrupts are thus better suited to handle asynchronous events.

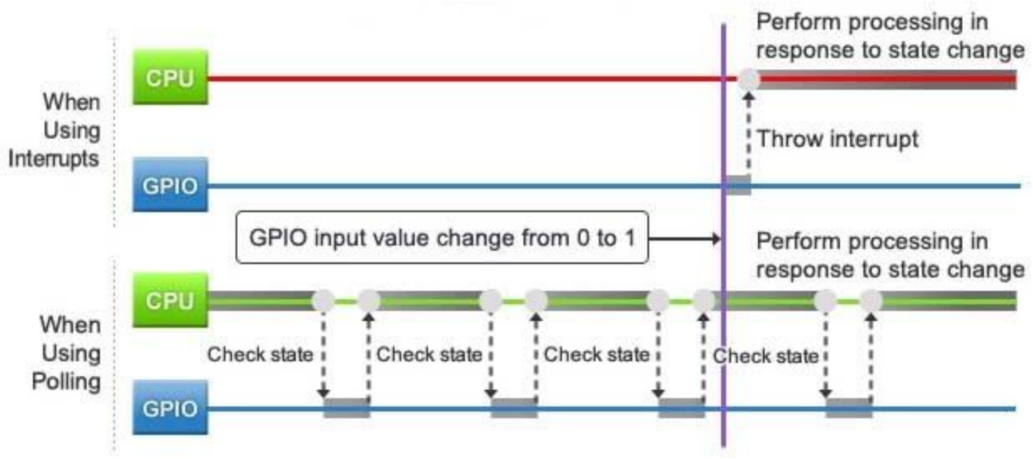
Perform processing in response to state change
CPU
When Using Interrupts
Throw interrupt
GPIO
GPIO input value change from 0 to 1
Perform processing in response to state change
CPU
When Using Polling
Check state
Check state
Check state
Check state
GPIO

Figure 1*: Detecting an event with polling and with hardware interrupts [1].*

Typical microprocessors have multiple states, such as active and sleep, that are important for power management. Depending on the application, the transition between these states can be time-triggered or event-triggered. In both cases, an interrupt initiates the change from sleep to an active state. An event-triggered can come from a GPIO pin, while a time-triggered can come from a timer or a real-time clock (RTC). Figure 2 shows a typical state diagram for these two systems.

Entering the sleep-state requires informing the Power Control Module (PCM) to go into low power mode, e.g. by calling PCM_gotoLPM3(), it enters low power mode 3. Depending on the lower power mode level, the microprocessor will turn off a set of peripherals and internal functions to save energy. Then it will halt (sleep) until an interrupt happens. When the interrupt is received, it will switch to active mode and execute the corresponding interrupt service routine (ISR). Afterwards, it will continue with the program in the main() function.

An event such as a button press will lead to an interrupt. The interrupt flag (IFG) register has a bit assigned to each configured interrupt peripheral. On every occasion the interrupt happens, the corresponding IFG bit is set. Three conditions must be met before the microprocessor is alerted about the event:

1. The pin must be configured to trigger an interrupt.
2. The corresponding interrupt must be enabled.
3. If these two conditions hold, the global interrupt enable (GIE) register is tested. If global interrupts are enabled, the microprocessor then is preempted, and the according to interrupt service routine (ISR) be executed.

This global switch is useful to temporarily turn off all three interrupts to prevent modifying a global variable or avoid preemption between instructions that logically belong together. The interrupt vector table maps the interrupts to the correct ISR.

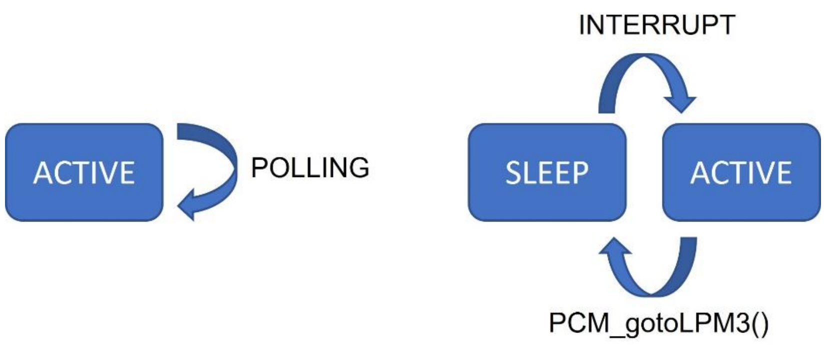

Figure 2*: Polling: The system typically remains in active mode to detect an event.* ***(left)***
Interrupt: The system can be in sleep mode until an event occurs. ***(right)***

A dedicated or grouped interrupt is triggered, depending on the source of the interrupt. For peripherals like GPIO ports, multiple pins could produce the same interrupt. In these cases, it is necessary to query the pin's interrupt vector register to identify the interrupt's exact source. Typically, this is done inside the ISR. Once an ISR identifies the source of the interrupt, it can react in accordance. Typically, ISRs execute in a privileged mode that can mask other interrupts. Hence, ISR should be **<u>as short as possible</u>** and only set application-specific flags to indicate to the microprocessor's main thread to execute the corresponding task in response to an interrupt.

Each interrupt type has a priority such that it is always clear which interrupt is handled first in case multiple interrupts occur at the same time. Furthermore, it is possible to allow nested interrupts, i.e., one interrupt interrupts another interrupt's handling.

Configuring an event-triggered interrupt

On reset, all interrupts are disabled. The following steps are necessary to configure a GPIO event-triggered interrupt on the MSP432.

1. Configure the GPIO pin as an input.
   `GPIO_setAsInputPinWithPullUpResistor()`
2. Select the edge (clock signal) that triggers the interrupt.
   `GPIO_interruptEdgeSelect()`
3. Clear the pin's interrupt flag. This makes sure that no previous interrupts are handled.
   (This step is not mandatory, but it is good practice to do so.)
   `GPIO_clearInterruptFlag()`
4. Set the interrupt enable (IE) bit of the specific GPIO pin
   (enabling the interrupt in the peripheral).
   `GPIO_enableInterrupt()`
5. Set the interrupt enable (IE) bit of the corresponding interrupt source
   (enabling the interrupt in the interrupt controller).
   `Interrupt_enableInterrupt()`
6. Enable interrupts globally (set global interrupt enable (GIE) bit).
   `Interrupt_enableMaster()`

Details of the corresponding API call can be found in DriverLib manual. After these steps, any new event (positive-edge or negative-edge) on the pin will generate an interrupt, preempt the microprocessor, and start executing the corresponding ISR.

Defining an Interrupt Service Routine (ISR)

The Interrupt Service Routine (ISR) is executed if the processor detects the corresponding interrupt. ISR has few restrictions; since it cannot receive arguments or return values, it can only modify global variables. In general, a programmer has the flexibility to choose any arbitrary function to be an ISR.

The code example in Task 1.1, we define an ISR for Port 1 interrupt labelled "*void PORT1_IRQHandler(void)*". Mapping of the interrupt to the ISR is performed in the Interrupt Vector Table (IVT). In code composer, IVT is defined in “startup_msp432p401r_ccs.c”. Each ISR needs to be mapped to an interrupt source. Since the GPIO pins are connected to a grouped interrupt, we first need to figure out which pin triggered the interrupt in the ISR.

Task 1.1: Interrupts with Buttons

A version of the source code utilising the hardware register access can be obtained from xSITe (LAB3_Interrupt.c). You can duplicate the lab 2 project and replace the lab 3 code with it. This sub-task intends to migrate the code to use DriverLib. The code configures an interrupt that detects when button S1 is pressed. This leads LED1 to toggle while pressing button S2 toggles the RGB in LED2.

- Get an overview of the existing GPIO and Interrupt API methods. Chapter 10 & 12 in the DriverLib manual explains its use.
- Configure LED1 and LED2 as output. Make sure all the LEDs are turned off whenever the MSP432 is reset.
- Configure the GPIO pins connected to the buttons S1 and S2 of the MSP432 as input with a pull-up resistor.
- Setup the interrupts by selecting the edge and by clearing the interrupt flags. Select the edge such that an interrupt is triggered when the button is released.

- Enable the interrupts for S1 and S2. For this, enable interrupts for (1) the corresponding GPIO pins (peripheral), (2) the port (interrupt controller) and (3) globally.
- Implement the *PORT1_IRQHandler* interrupt service routine (ISR) such that LED1 and LED2 (RGB LEDs) toggle.
- Ensure that the *PORT1_IRQHandler* ISR has been mapped to the corresponding interrupt source (in this case Port1). This is done in "startup_msp432p401r_ccs.c"
- Compile and run (debug) your program.
- Press the buttons S1 and S2 and verify the resulting action (LEDs).
- Verify that the interrupt is triggered when the button is released, not when it is pushed down.

Task 2: IR-based Wheel Encoder

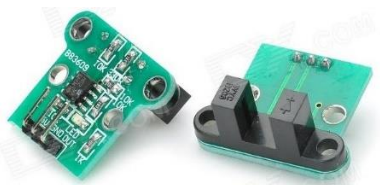

Figure 4 *HC-020K Photoelectric encoders*

The working principle of the encoder shown in Figure 4 is presented in Figure 5 (a). It uses a slotted wheel with a single LED and photodetector pair that generate pulses as the wheel turns, and the speed of an object can be calculated by measuring the pulse duration Δti (i.e. elapsed time or period of a pulse) between successive pulses [2]. It comprises three connections, GND, VCC and OUT. GND and VCC are meant to supply power to the module (in our case via the MSP432's GND and 3.3V pins), while OUT generates the square-pulse signal.

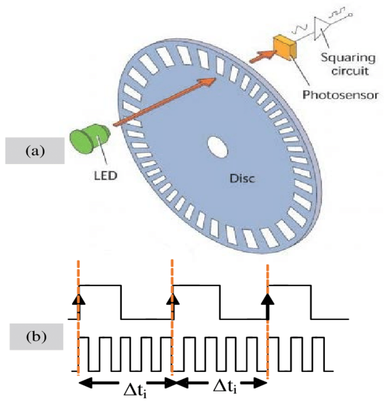

Figure 5 (a) *Encoder principle and* ***(b)*** *operations of measuring elapsed time.* *[3]*

Task 2.1: Testing the Wheel Encoder

This section illustrates the working of a wheel encoder without a single line of source code. This is achieved by using LED1 on the MSP432 via the P1.0 pin. Remember to remove the jumper at P1.0 before connecting the OUT pin on the encoder to the MSP432's P1.0 lower jumper pin (see green in Figure 6). The VCC pin will be connected to a 3.3V pin and GND connected to GND, as shown in Figure 6. Connect the MSP432 to the micro-USB to boot it up.

The following figure illustrates how the encoder is connected to the MSP432:

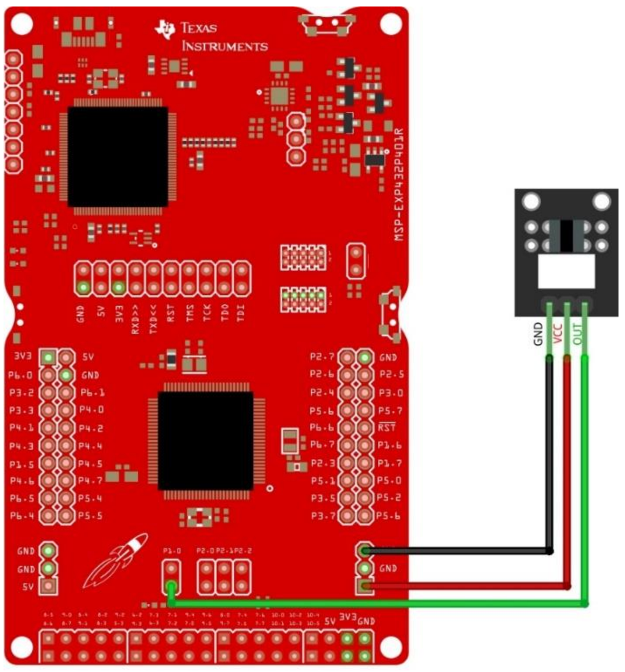

***Figure 6** Hard-wiring the encoder to MSP432's LED*

The encoder's LED will be turned on perpetually, while the photosensor will detect the light from the LED if unblocked and vice-versa. The specifications for the IR sensor is as follows:

- Chip: 74HC14D
- VCC: +3.3V ~ 5 V
- GND: Ground or negative
- OUT: The signal output, link MCU I/O port
- HIGH = Detect obstacles
- LOW = No obstacle

Follow the video guide for "Wheel" followed by the "Wheel Encoder". We are not going to use the motors in this lab session. However, the wheels help illustrate the use of the encoder. Turning the wheel "attached" to the wheel encoder will cause the LED1 to toggle each time it transitions from blocked to unblocked and vice-versa.

Task 2.2: Wheel Encoder using Interrupts

Before proceeding into this sub-task, insert the jumper back into P1.0. Connect the OUT pin from the encoder to the MSP432's P2.5 pin. The GND and VCC still remain the same as in the previous set-up.

This sub-task seeks to set up an interrupt (in this case, an ISR) such that each time the photosensor in the encoder detects a transition from block to unblock (or vice-versa), it will increment the counter. A full rotation comprising 20 transitions (corresponding to the 20 notches on the encoder) will toggle LED1 via the GPIO pin P1.0. The following is the program flow for both the **main function** and **ISR**. Refer to the DriverLib manual to select the correct API calls.

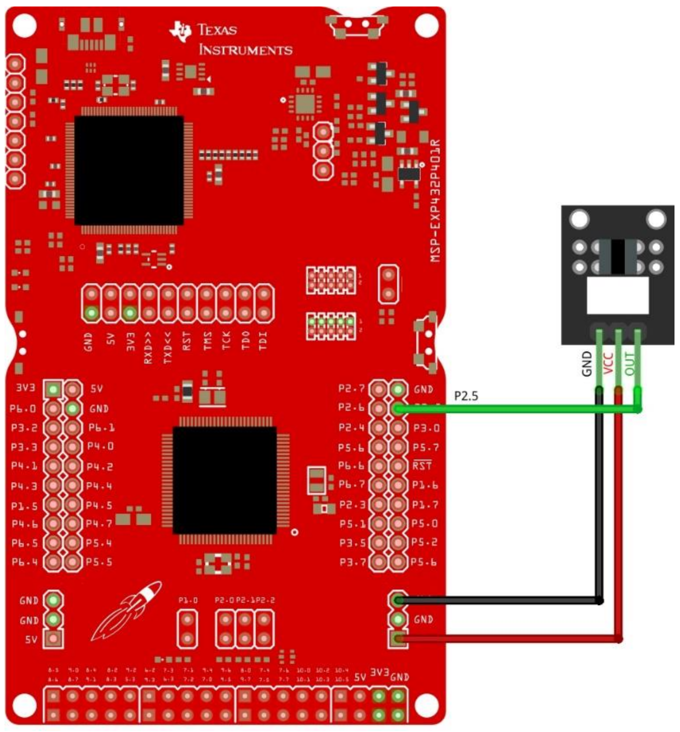

Figure 7 *Connecting the encoder to MSP432*

Main Function

1. Declare a **global** counter variable
2. Disable the watchdog timer
3. Initialise the **global** variable
4. Configure pin P2.5 as **input** (with pull-up resistor)
5. Configure pin P1.0 as **output**
6. **Clear** the interrupt flag for pin P2.5
7. **Enable** interrupt for pin P2.5
8. **Enable** interrupt for Port 2
9. **Enable** master interrupt
10. Forever loop that puts the device to **low power mode 3 state**.

Consider This: Why must step #6 occur before steps #7, #8 and #9? How do you configure the interrupt to ONLY trigger at the rising edge?

Interrupt Service Routine (ISR)

1. Declare an ISR for Port 2 (Hint: PORT2_IRQHandler)
2. Declare a local **integer** variable
3. Get interrupt status for Port 2 and store it into the local variable (declared above)
4. Increment the **global** variable.
5. Check if the **global variable** is equal to 20 (number of notches on the wheel encoder).
    - TRUE: Toggle pin P1.0 and reset the local variable
    - FALSE: Nil
6. Clear the interrupt flag for Port 2 (Hint: use the variable)

Consider This: What could happen if step #6 is performed at step #4?

Submission

Modify Task 2.2 to keep track of the number of full rotations (counter) the wheel makes. This counter will reset to 0 after 7. This number should be presented on the RGB LEDs (P2.0, P2.1 and P2.2). The table below will illustrate how the LEDs should be turned on or off.

COUNTER	RED	GREEN	BLUE
0	OFF	OFF	OFF
1	OFF	OFF	ON
2	OFF	ON	OFF
3	OFF	ON	ON
4	ON	OFF	OFF
5	ON	OFF	ON
6	ON	ON	OFF
7	ON	ON	ON
0	OFF	OFF	OFF
1	OFF	OFF	ON
2	OFF	ON	OFF
3	OFF	ON	ON
4	ON	OFF	OFF
5	ON	OFF	ON
6	ON	ON	OFF
7	ON	ON	ON
…			

Only submit the .C file to DropBox. Include both team members' names.

References

[1] Renesas. "Essentials of Microcontroller Use Learning about Peripherals: Interrupts." Accessed on 19 Sept 2021. (https://www.renesas.com/us/en/support/engineer-school/mcu-programming-peripherals-04-interrupts)

[2] Nurmi, Jarmo, et al. "Micro-electromechanical system sensors in unscented Kalman filter-based condition monitoring of hydraulic systems." 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. IEEE, 2013.

[3] Mones, Zainab & Feng, Guojin & Muo, Ugo & Wang, Tie & Gu, Fengshou & Ball, Andrew. (2016). Performance evaluation of wireless MEMS accelerometer for reciprocating compressor condition monitoring: Proceedings of the International Conference on Power Transmissions 2016 (ICPT 2016), Chongqing, P.R. China, 27–30 October 2016.