The Mechatronics Revolution: Fundamentals and Core Concepts
Lab Assignment 3

Collision Detection using Bump Sensors, General-Purpose Input Outputs (GPIOs) and Interrupt-Driven Programming

3.1 Objective

The main objective of this lab is to interface the bump sensors of the TI-RSLK-Mechkit robot with the MSP432 LaunchPad to detect collisions. You will use MSP432 GPIOs (as inputs and outputs) to read switches and turn on LEDs. This lab also provides an introduction to programming the MSP432 in Code Composer Studio (CCS) using the TI MSP432 Driver Library (DriverLib), and some basics of embedded systems such as edge-triggered interrupts and polling.

In Module 3, we studied microcontroller fundamentals including C Programming, General-Purpose Input Outputs (GPIOs), Polling and Interrupt-Driven I/O. Specifically, the fundamental knowledge required for this lab assignment were provided in the following topics of Module 3:

Module 3, Topic 3, Lessons 1 to 4	General-Purpose Input Output
Module 3, Topic 4, Lessons 1 to 3	Interrupt-Driven Programming

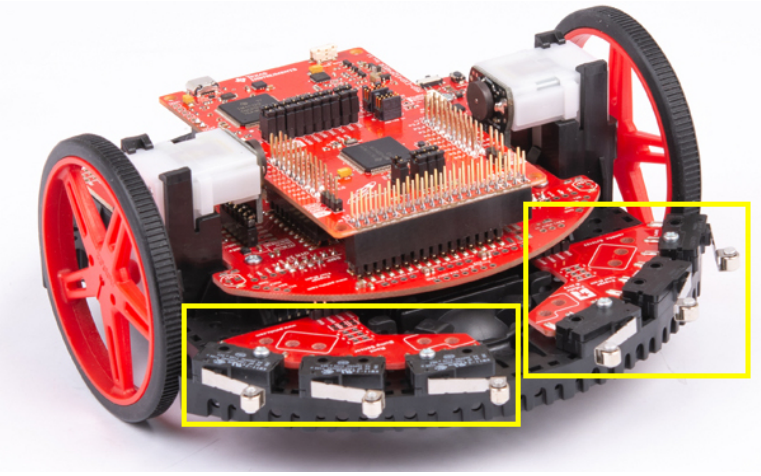


Figure 1: The TI-RSLK-Mechkit robotics system learning kit includes a MSP432 LaunchPad and six bumper sensors that can be used for collision detection.

3.2 Setup

This lab requires the *Code Composer Studio* IDE, the *TI MSP432P401R LaunchPad* and the *TI-RSLK-Mechkit* robot. The lab uses the onboard electronics of the LaunchPad such as switches and LEDs, as well as the 6 bump sensors attached to the TI-RSLK-Mechkit robot.

All components needed for this lab are included in the TI-RSLK-Mechkit. Details on how to assemble the TI-RSLK-Mechkit robot are presented in the Robot Assembly Manual, available under Course Resources on edX (Robot-Assembly-Manual.pdf). Six AA batteries (not included) will be needed to power your robot. You can use rechargeable 1.2V NiMH batteries or disposable 1.5V alkaline batteries.

The MSP432 LaunchPad has two user programmable pushbutton switches and two programmable LEDs as basic input and output components. The switches S1 and S2 are connected to GPIOs P1.1 and P1.4 respectively. The red LED (LED1) is connected to pin P1.0. The multicolored LED (LED2) uses 3 pins for its operation, and is connected to GPIOs P2.0, P2.1, P2.2 for the red, green and blue color respectively.

We will also use the *Texas Instruments MSP432 Driver Library (DriverLib)* in this lab to write programs for the MSP432 LaunchPad.

Two predefined grading macros and a setup function are declared in the `mechrev.h` header file, available under Course Resources on edX (mechrev-header.zip). The output text file generated by the macros will be used to grade the assignment.

The output of the grading macros will be printed to the Code Composer Studio console under the debug mode. Therefore, the LaunchPad USB cable must be connected from robot to PC when collecting the grading outputs.

Warning: Make sure that you have removed the +5V jumper on the MSP432 LaunchPad. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

3.3 Problem Statement

Write a program that monitors the status of the pushbuttons S1 and S2 by polling, and turns the LED1 on and off accordingly. Furthermore, the program requires to detect the bump sensors press using GPIO interrupts, and toggle the multicolored LED2 between the colors accordingly.

Specifically, the program turns on LED1 by pressing and holding either pushbuttons S1 or S2, and off by releasing S1 or S2. That is, LED1 is on only if at least one of the pushbuttons is pressed.

The multicolored LED2 is always on, and should change color between red, green and blue by pressing the bump sensors. Pressing one of the two outer bumps sensors (BMP0 or BMP5) of the TI-RSLK-Mechkit robot should change the color of LED2 to red, the middle bump sensors (BMP1 or BMP4) should change the color of LED2 to green, and pressing one of

Bump Sensor			Color
BMP0	or	BMP5	Red
BMP1	or	BMP4	Green
BMP2	or	BMP3	Blue

Table 1: The LED2 colors associated to each bump sensor.

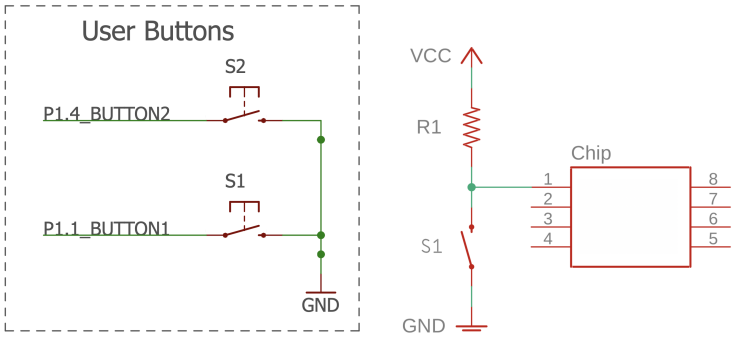
the two inner bumps sensors (BMP2 or BMP3) should change the color of LED2 to blue. That is, if LED2 has become blue after BMP2 (or BMP3) is pressed, it should turn to green when BMP1 or BMP4 is pressed and to red when BMP0 or BMP5 is pressed. The colors associated to each bump sensor are listed in Table 1.

Note: You are required to use "polling" for handling S1 and S2 switches, and use "PORT4 interrupt handler" for detecting the bump sensors press.

3.4 Hardware

3.4.1 Pushbuttons and Pull-Up Resistors

The MSP432 LaunchPad is equipped with two pushbutton switches S1 and S2, with one side connected to the MSP432 GPIO pins P1.1 and P1.4 respectively, and the other side connected to GND, as shown in Figure 2 (a).



(a) (b)

Figure 2: (a) Schematic of S1 and S2 switches on MSP432 (Image courtesy of Texas Instruments), (b) A pull-up resistor, connected to a normally open pushbutton.

By enabling the internal pull-up resistors of the GPIOs P1.1 and P1.4, we can make sure that the inputs are pulled high when the switch is not pressed. This can be visualized as the circuit shown in Figure 2 (b). The S1 and S2 switches are normally open. This will result in the GPIOs P1.1 and P1.4 to be high when the associated switches are not pressed, and become low when their switch is pressed.

Note: The MSP432 LaunchPad has internal pull-up resistors, so you do not need to add physical pull-up resistors to your own circuit. To make use of these internal resistors, use the `MAP_GPIO_setAsInputPinWithPullUpResistor(...)` function from the DriverLib library.

3.4.2 LEDs

The MSP432 LaunchPad is equipped with two user programmable LEDs: LED1 and LED2. LED1 is a red color LED, connected to the GPIO P1.0. LED2 is a three-colored LED, that can provide red, green, and blue colors driven by three GPIO pins P2.0, P2.1 and P2.2 respectively.

The schematic for the LEDs is shown in Figure 3. To turn the LEDs on or off, simply set the associated output pin high or low.

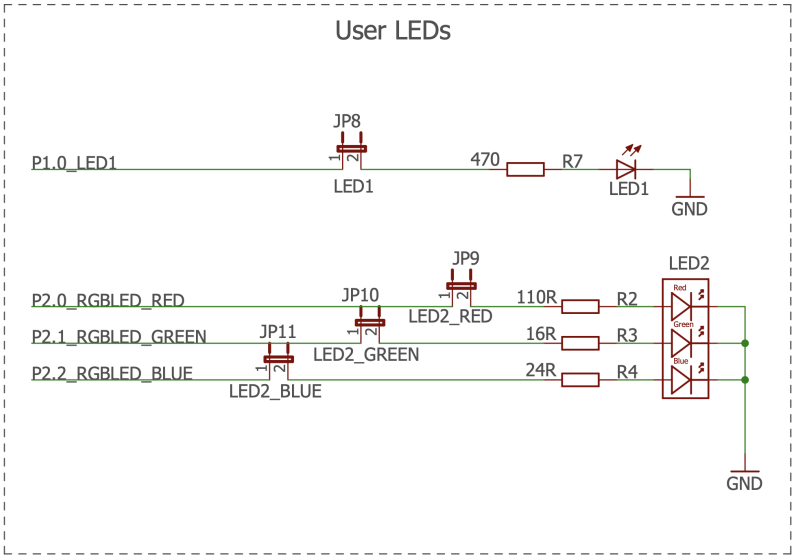


Figure 3: Schematic of LED1 and LED2 on MSP432 LaunchPad (Image courtesy of Texas Instruments).

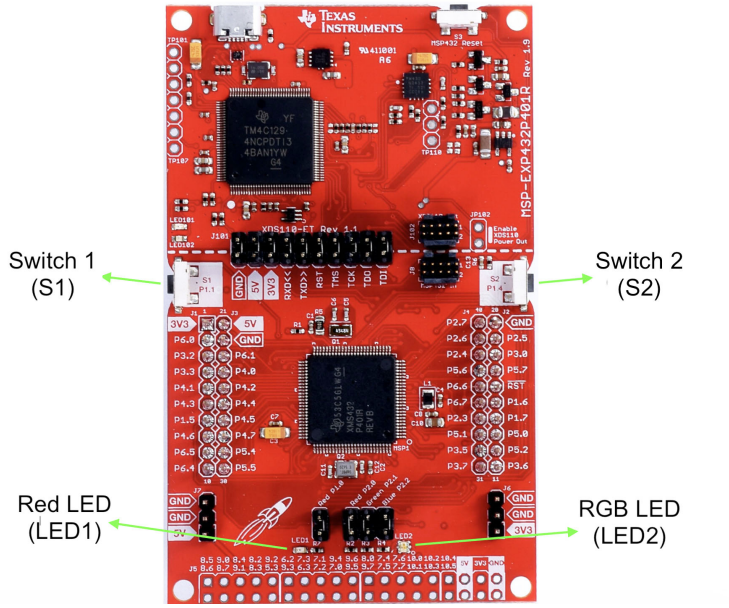


Figure 4: MSP432 LaunchPad Switches and LEDs.

3.4.3 Bumper Switch Boards

The TI-RSLK-Mechkit includes two bumper switch boards, shown in Figure 5. Each bumper switch board has a 4-pin male header that needs to be inserted horizontally into a corresponding 4-pin female header on the front, curved end of the TI-RSLK plastic chassis. These female headers can be found on the underside of the chassis board at the edge of the curved end. For more details on how to connect the bumper switch boards to the TI-RSLK-Mechkit robot, please see the Robot Assembly Manual, available under Course Resources on edX (Robot-Assembly-Manual.pdf).

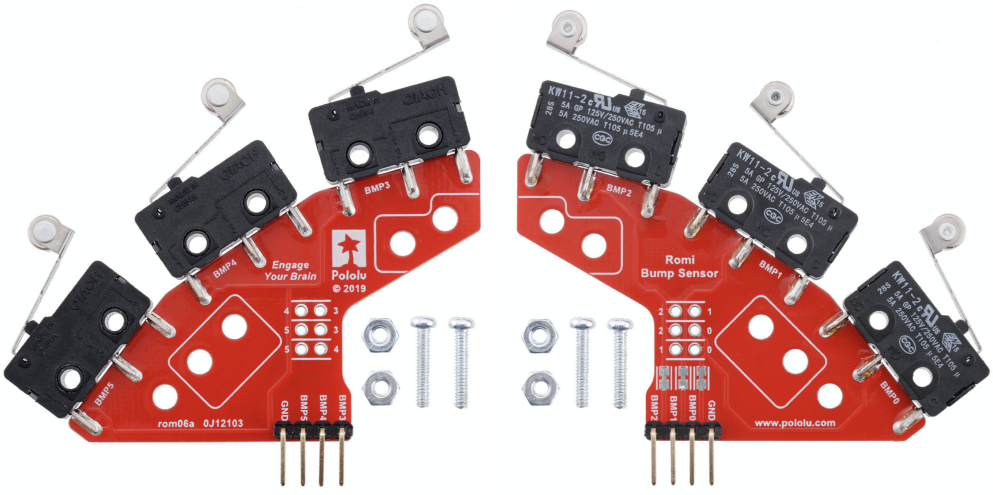


Figure 5: Left and Right Bumper Switch Assembly for TI-RSLK-Mechkit robot (Image Courtesy of Pololu).

Through the chassis board, the six bump sensors BMP0 to BMP5 get connected to MSP432 GPIO pins P4.0, P4.2, P4.3, P4.5, P4.6 and P4.7 respectively. The bump switches are normally open, with the other sides connected to GND.

We can detect collisions by enabling the internal pull-up resistors for the associated GPIOs of MSP432 and detecting the low input. The GPIOs read high when the bump switches are not pressed. The GPIOs can be monitored by either polling or the PORT4 interrupt handler. When using interrupts, a collision can be detected by a *falling edge* happening on each input signal pin.

Note: For this lab assignment, you are required to use "PORT4 interrupt handler" for detecting the bump sensors press.

Table 2 summarizes the MSP432 GPIOs that are directly connected to the Pushbuttons, LEDs and Bump switches.

3.5 Software Requirements

3.5.1 DriverLib

It is recommended to write your program with the help of the functions and macros defined in the Texas Instruments MSP432 Driver Library (DriverLib) (http://www.ti.com/tool/MSPDRIVERLIB). DriverLib is a specialized library created by TI for the MSP series of microcontrollers. It introduces a higher level of abstraction for better readability of your source code. This programming method is an alternative to the typical direct register programming used for many MCU's (we covered DriverLib vs direct register access in Module 3, Topic 3, Lesson 3). The DriverLib library user guide can be found under Course Resources on edX (MSP432-DriverLib-Users-Guide.pdf). This guide provides all the different peripherals of the microcontroller supported by the driver library, their associated functions and the inputs and output arguments of these functions.

The term `MAP_` can be used at the beginning of each function given in DriverLib. This prefix improves speed and performance. The DriverLib header file "driverlib.h" has to be included in the project. This header file provides links/access to all the other headers defined in the driver library such as GPIO, Timers, ADC, etc. In this lab, the GPIO functions are to be used. All DriverLib functions can be accessed simply by including "driverlib.h" as mentioned earlier. Reference the user guide to determine the functions you would like to use and their syntax. For example, to configure the switch S1, the following function is used:

```
MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1,GPIO_PIN0);
```

This function (with above arguments) tells the microcontroller to use pin 0 of port P1 as an input with an internal pull up resistor.

To assist you with this initial program, a source code file with some example GPIO function calls are included with the lab assignment as an additional reference, available under Course Resources on edX (lab3-example.zip).

Device	MSP432 GPIO
Pushbutton1	P1.1
Pushbutton2	P1.4
LED1	P1.0
LED2 Red	P2.0
LED2 Green	P2.1
LED2 Blue	P2.2
BMP0	P4.0
BMP1	P4.2
BMP2	P4.3
BMP3	P4.5
BMP4	P4.6
BMP5	P4.7

Table 2: Pin map of the MSP432 LaunchPad

Warning: With the new versions of the DriverLib library, the base address and module instance names have been changed from `{PERIPHERAL_NAME}_MODULE` to `{PERIPHERAL_NAME}_BASE` per the new device header file definitions. For example, `TIMER_A0_MODULE` to `TIMER_A0_BASE`. Make sure to consult with the DriverLib user guide when using a macro or function. Using the old notation will cause errors when compiling the program with Code Composer Studio.

3.5.2 GIPO Interrupts

For this lab assignment, you are required to use PORT4 interrupt handler to detect bump sensors press.

Since the GPIOs are configured with pull-up resistors, a bump sensor press can be detected by the PORT4 interrupt service routine (ISR) triggered by a *falling edge*.

Below is a very brief code example showing the PORT4 ISR that increments a counter variable when the P4.0 or P4.7 GPIO interrupt is triggered, caused by the bump sensor BMP0 or BMP5.

```
/* GPIO ISR */
void PORT4_IRQHandler(void)
{
    uint32_t status;
    status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P4);
    /* verify that the interrupt is triggered by P4.0 or P4.7 */
    if (status & GPIO_PIN0 || status & GPIO_PIN7)
    {
        /* increment a global counter variable */
        counter++;
```

```
    }

    /* clear the PORT4 interrupt flag */
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, status);
}
```

Note that the GPIO interrupt needs to be enabled in the main code during the initialization by using the following DriverLib functions :

```
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN0 | GPIO_PIN7);
GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN0 | GPIO_PIN7);
GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN0 | GPIO_PIN7,
                                       GPIO_HIGH_TO_LOW_TRANSITION)
Interrupt_enableInterrupt(INT_PORT4);
```

For more information about PORTx interrupt handler, please refer to the MSP432 Reference Manual and MSP432 Peripheral Driver Library User's Guide documents, available under Course Resources on edX (slau356i.pdf and MSP432-DriverLib-Users-Guide.pdf).

3.6 Software Architecture

Note: To create a new CCS project for the MSP432P401R target device, and include the `mechrev.h` header file and the DriverLib library in the project, follow the procedure described in Lab Assignment 1. The `mechrev.h` file and the `driverlib` folder are available under Course Resources on edX (mechrev-header.zip and driverlib.zip).
To get you started with designing the program architecture, a code template for Lab 3 is available under Course Resources on edX (lab-code-templates.zip).

After disabling the watchdog timer by calling the `WDT_A_holdTimer()` function to avoid unnecessary watchdog timer time-out interrupts, call the `mechrev_setup()` function provided by the `mechrev.h` header file to bypass the motor drivers of the TI-RSLK chassis board. See the code provided with Lab Assignment 1 for an example (the code is available under Course Resources on edX (lab1-main.zip)).

Warning: Make sure that you have called `mechrev_setup()` function to disable the built-in motor drivers on the TI-RSLK chassis board. Although we are not using the DC motors in this lab, not disabling the default drivers will cause a conflict with the SN754410 motor driver on the breadboard and will potentially damage the chassis board.

Initialize the P1.1 and P1.4 GPIOs (connected to the S1 and S2 switches of the LaunchPad) as inputs with pull-up resistors.

Initialize the P1.0, P2.0, P2.1, and P2.2 GPIOs as output pins, used for LEDs.

Initialize the P4.0, P4.2, P4.3, P4.5, P4.6 and P4.7 GPIOs as input pins with internal pull-up resistors and interrupt on falling edge, used for bump sensors. Remember to enable the PORT4 interrupt.

Design your code inside the main `while(1)` loop to detect (via polling) whether the S1 and S2 switches are pressed or not. Turn on or off LED1 accordingly, as explained in the Problem Statement in Section 3.3.

Design your code inside the PORT4 ISR function (PORT4_IRQHandler()) to detect bump sensors press. Change the color of LED2 inside the ISR accordingly, as explained in the Problem Statement in Section 3.3.
Note that as mentioned above, LED2 is driven by three GPIO pins P2.0, P2.1, and P2.2, associated to red, green, and blue colors respectively. Therefore, for example, when we want to change the color of LED2 to red, we need to set P2.0 to high and the other two pins (P2.1 and P2.2) to low:

```
/* set LED2 to red color */
GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN0); // red on
GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN1);  // green off
GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN2);  // blue off
// call the grading macro here when generating the grading outputs:
MACRO_LAB3_EVENT(); // see Section 3.9
```

3.7 Expected Performance

Press and hold the LaunchPad S1 switch and verify that the software can recognize the switch status correctly. LED1 should stay on while holding the psuhbutton, and go off when releasing the pushbutton. Repeat the same procedure for S2 and verify the performance.

Next, press and release one of the bump switches and verify that LED2 changes to the associated color, listed in Table 1. Press and release the other 5 bump switches verifying the software can properly recognize each of the 6 switches correctly and toggle LED2 through the correct colors. When you release a bump switch, LED2 should stay on with the last color.

3.8 Troubleshooting

A pushbutton switch does not work:

- Verify that you have initialized the GPIOs of PORT1 and their internal pull-up resistors correctly.
- Set breakpoints using the CCS debugger and verify that the application goes into the polling `if()` statements when pressing a button.

A bump sensor(s) does not work:

- Verify that you have initialized the GPIOs of PORT4 and their internal pull-up resistors correctly.

- Verify that you have initialized the interrupt for PORT4 GPIOs correctly.
- Set breakpoints using the CCS debugger and verify that the application jumps to the interrupt handler when a bump sensor is pressed.
- Look at bump sensors signals with a voltmeter, oscilloscope or logic analyzer. The voltage on the microcontroller pin should be 0 when the associated bump sensors is pressed and 3.3V when it is released. The operation of one switch should not affect the signals on the other switches.
- Look at the port registers in the debugger. With the debugger doing periodic updates, and the software running, you should see the port input register change with a switch press.

3.9 Grading

An output text file with the file extension `.txt` should be uploaded to Vocareum in order for your lab assignment to be graded.

You can generate the grading output text file by the following procedure:

- Make sure that the `mechrev.h` header file is included in your main code.
- You must call `mechrev_setup()` during the initialization of your code (before the main `while(1)` loop). This will disable the chassis board motor drivers.
- You must call `MACRO_LAB3_INIT()` after initializing all the GPIOs and peripherals in your code (right before the main `while(1)` loop).
- You must call `MACRO_LAB3_EVENT()` right after turning on LED1, and after changing the color of LED2 (but not anywhere else in the code).
  Note that when you are changing LED2 color at some part of the code, you should call the macro only once after changing all the associated GPIOs.

  Warning: When calling `MACRO_LAB3_EVENT()` due to a color change for LED2, make sure that you call the macro BEFORE clearing the PORT4 ISR interrupt flag (used for detecting bump sensors presses), otherwise the generated grading output will not be correct.

- After finalizing your program and placing the provided macros in the code, generate the grading output file by following the steps below:
  - Connect the USB debug cable to the MSP432 LaunchPad.
  - Start the debug mode in the Code Composer Studio. The code will be paused at the beginning of the main loop.
  - Press the Resume (F8) button in CCS to start debugging the code.

- Test your code by pressing the LaunchPad pushbuttons (for at least 10 times) and the bump sensors (for at least 10 times) randomly. The grading outputs should start printing in the CCS console.
- Stop the debug mode. Select all the outputs from the CCS console, copy them and paste them into an empty text file. Save the text file with an optional name (with file extension `.txt`) and submit it to Vocareum.

Note: Make sure that the first line in your text file starts with `0xff`, and other lines start with `0xf1`. Otherwise there has been a problem in macro calls. If the first line in the CCS console is something other than `0xff`, do not include that line into the text file that you are going to submit.

Warning: Make sure that the extension of your output text file is `".txt"`. Vocareum cannot open other file types, and would assign a zero grade to your work if the extension is other than `.txt`.

Warning: You can submit the lab assignment to Vocareum for up to 5 times. When *resubmitting* an assignment, make sure to first delete the old output file in the workspace (listed under the "work" directory) and then upload the new file, or use the same name for the new output file so it replaces the old one in the workspace.
DO NOT press the Submit button if there are more than 1 text file uploaded to the workspace, otherwise Vocareum would not be able to open the text file and would assign a zero grade to your work.

For more details on how to include header files, save data as text files, and submit the files to Vocareum, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).