TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

Module 14

Lab 14: Real-time Systems

Lab: Real-time Systems

14.0 Objectives

The purpose of this lab is to interface bump sensors to detect collision, which will be one task that the robot will need to explore its world; see Figure 1.

1. You will use edge-triggered interrupts to detect collisions.
2. You will use shared global variables to communicate between threads.
3. You will use priority to define order of execution when servicing multiple concurrent events.
4. You will profile the time to service an interrupt.
5. You will combine this lab with previous labs to solve a problem.

Good to Know: Interrupts are extremely important for embedded systems, providing a mechanism to implement real-time behavior and multi-threading.

14.1 Getting Started

14.1.1 Software Starter Projects

Look at these three projects:
EdgeInterrupt (edge-triggered interrupt on P1.1 and P1.4),
Lab13_Timers (your solution to lab 13)
Lab14_EdgeInterrupts (starter project for this lab)

14.1.2 Student Resources (in datasheets directory)

Meet the MSP432 LaunchPad (SLAU596*)*
MSP432 LaunchPad User's Guide (SLAU597*)*
Polulu_BumpSwitch_1404.png, mechanical drawing of switch
QTRX, line sensor datasheet

14.1.3 Reading Materials

Chapter 14, "Embedded Systems: Introduction to Robotics"

Good to Know: Edge-triggered interrupts is a useful feature of microcontrollers that are used commonly in embedded systems. In general, external events can be signified as a change in status. Examples include danger, power failure, temperature overload, system faults. If the status is an external digital logic signal, it can be connected to a GPIO input, and the system can request an interrupt on a rising or falling edge of that signal.

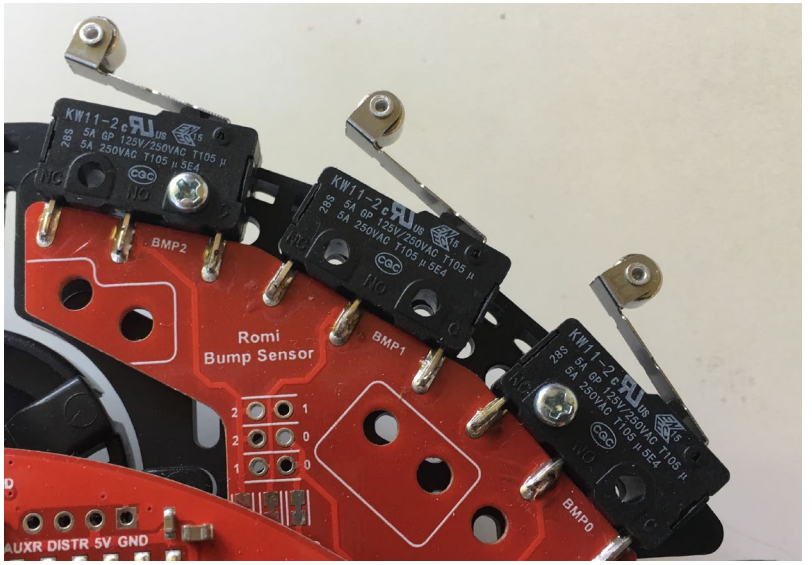

Figure 1. Bump sensors positioned at the front of the robot.

14.1.4 Components needed for this lab (combination of Labs 10 and 12)

All components needed in this lab come with the TI-RSLK Max robot kit (TIRSLK-EVM), Batteries will be required to power your robot.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX kit	TI	TIRSLK-EVM

Table 1 Parts needed for this lab

14.1.5 Lab equipment needed

Oscilloscope (one or two channels at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling

14.2 System Design Requirements

The first goal of this lab is to use edge-triggered interrupts to detect collisions by the bump sensors while the robot is moving. A collision event should cause an

interrupt, and reading the status of the bumper switches should occur in the **interrupt service routine** (ISR).

In the previous labs, we suggested you make the periodic interrupt used for the line sensor (see SysTick Interrupt in Module 10) a high priority because it is a real time measurement. However, once we integrate labs together, the collision task should have a priority even higher priority than the periodic measurements because a collision represents a danger condition that requires immediate service. In lab 5, you attached the bump sensors to the robot and interfaced them to the microcontroller. In this lab, you will shift the software servicing of the sensors from a periodic interrupt to an event-driven trigger.

The second goal of this lab is integrate components into a single hardware software solution that performs a simple but integrated task. This integrated task must include the edge-triggered interrupt interface of the bump sensors.
Feel free to adapt/combine solutions/sensors from previous labs. For example, you could explore a fenced in arena, as shown in Figure 2. When the robot collides into the wall, it could back up a little, turn, and then continue forward.

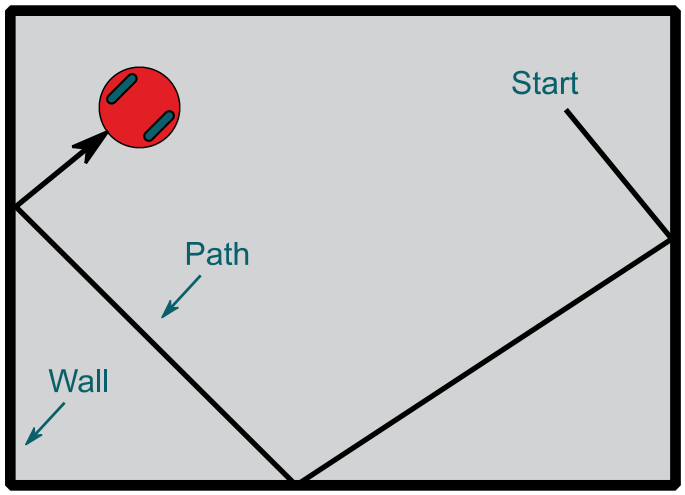

Figure 2. A possible integrated maze is to explore inside a box.

This high-level strategy should be performed separate from the edge-triggered ISR caused by the bump switch touch edge interrupt. For example, you could define three threads

- Periodic Timer_ A1 interrupts to run the high-level strategy
- Edge triggered interrupts for collisions
- Main program initializes and then does nothing in the loop.

14.3 Experiment set-up

You interfaced the QTR line sensor and bump sensors back in Lab 5. Figure 3 shows one possible placement for six sensors. Figure 4 shows a simple electrical circuit for interfacing the switches.

Warning: *TI MSP432 pins are not 5V tolerant; you must power the line sensor and bump sensors with +3.3V.*

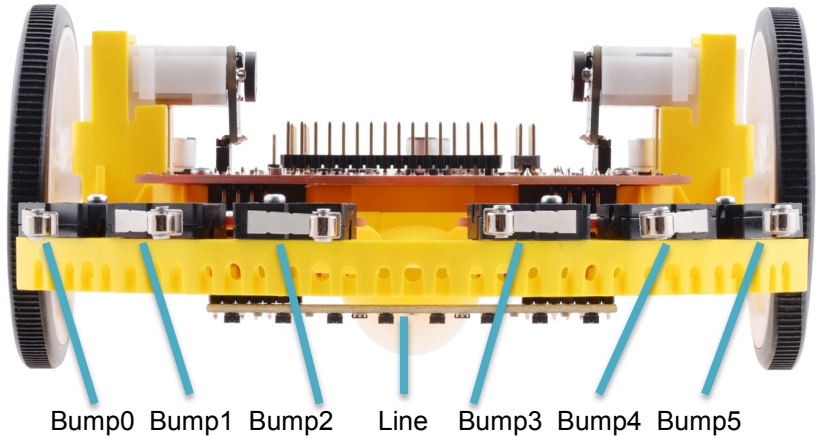

Figure 3. Bump sensors attached to the front of the robot (www.pololu.com).

Warning: Please ensure the +5V jumper on the MSP432 LaunchPad is disconnected or removed. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

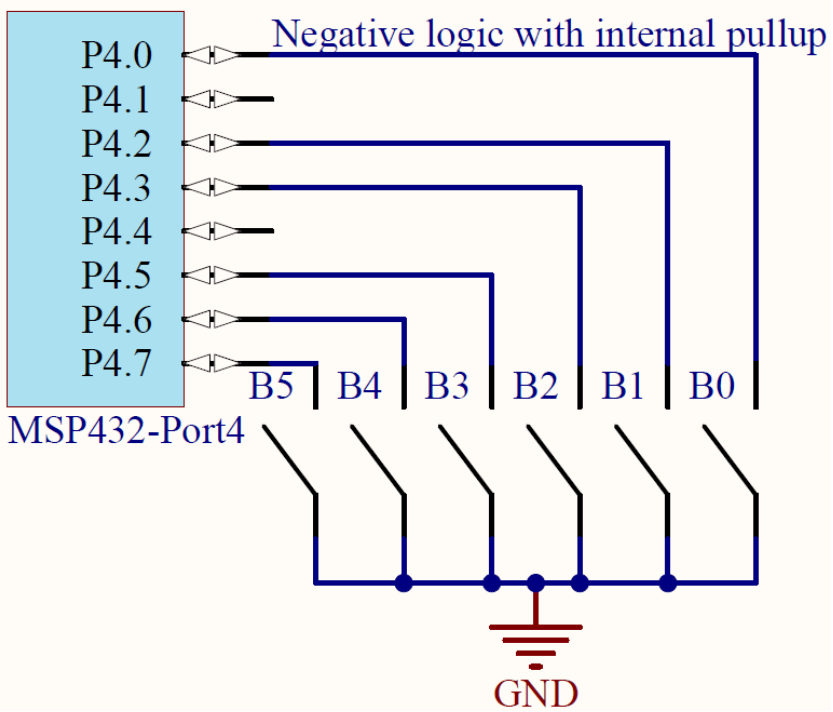

Figure 4. The interface circuit used in RSLK kit. If you are not using the kit remember that only Ports 1 – 6 can trigger interrupts.

14.4 System Development Plan

14.4.1 Interface the switches and motors

If you didn't interface the bump sensors as part of Lab 5, interface the bump sensors to the robot as shown in Figure 4. The motors were interfaced in Labs 5, 12, and 13.

14.4.2 Develop and debug the edge-triggered interrupts

You will write one function to initialize the bump sensors, **BumpInt_Init()**. This function configures the appropriate port pins, enables internal resistors as needed, and **enables edge-triggered interrupts**. You need a way to integrate the low-level device driver code with the high-level robotic system. One way is to place the ISR at the high level. This is a simple approach, but it does intertwine high-level with low-level code. A more elegant solution is to use a hook or function pointer. The user-supplied function is passed from high level to low level dynamically, when the interface is initialized. This high-level function will be called on a collision from your ISR that handles the edge-triggered event. To provide additional functionality, your ISR will pass a 6-bit value from the sensors.

void BumpInt_Init(void(*task)(uint8_t));

Note: In previous labs, you handled collisions within a periodic ISR. If the interrupt period is 10ms, the average latency is 5ms and the worst case latency of a collision event will therefore be 10ms. When running as a high priority edge-triggered interrupt, the latency will be on the order of 1 µs.

You designed and tested the function Motor_Stop as part of Lab 13. For more information on the motors, refer back to Labs 12 and 13. For example, if the robot needs to stop, you define a function

```
uint8_t CollisionData, CollisionFlag;  // mailbox
void HandleCollision(uint8_t bumpSensor) {
  Motor_Stop();
  CollisionData = bumpSensor;
  CollisionFlag = 1;
}
```

HandleCollision is defined within the high level software. When you initialize the bump sensors you pass in a pointer to this high-level routine.

BumpInt_Init(&HandleCollision);

This function will be called from an ISR, so its execution time should be short and bounded. In other words, please avoid long delay loops in the ISRs.

14.4.3 Profiling

Use an oscilloscope and an unused pin to measure the latency of the collision detector. One channel of the scope shows the falling edge (collision) and a second channel shows when the ISR is run. You can use the triple toggle technique to measure both latency (delay from collision to the start of service) and response time (delay from collision until the motors are stopped).

You will need a real scope or logic analyzer (and not TExaS), because the times will be on the order of microseconds *(TExaS has a time resolution of 100µs)*.

14.4.4 Integrated Robotic System

While debugging the integrated system use the dump techniques learned in Lab 10 to record strategic information during the run. Operate the robot for about a minute, and then observe the debug information to verify robot sensors and actuators operated as intended.

If you run the high-level strategy in a periodic interrupt it will be easy to implement a robot command language like

1. **Back Up** slowly for 1 second
2. **Turn Right** slowly for 5 seconds (90 degrees)
3. **Go Forward** quickly for 1 minute (infinite time)
4. **Repeat steps**

Since this task runs in a periodic interrupt, the software has no loops. More specifically, it has no do-while-loops, no while-loos, and no for-loops. This software structure will be very efficient of processor execution time.

On a collision, you stop and restart this simple set of commands

14.5 Troubleshooting

Bump sensors don't work:

- Check the wiring as described in Figure 3. Figure 3 shows negative logic and internal pull-up resistors. Because there are no external resistors, you do need to configure the internal resistors in software.
- Look at signals with a voltmeter, scope or logic analyzer. You should see the voltage on the microcontroller pin be 0 when pressed and 3.3V when 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 the switch.

Robot does not operate properly:

- Don't try to solve the entire project all at once. Break the problem into many small components and test each component separately. After two components are tested, combine those two components and test the two components together. Incrementally add components until the system is complete.
- Use profiling techniques to observe CPU utilization. In particular, measure the percentage time each thread requires. Observe if the execution of one task is preventing another thread from running.
- Use the debugging techniques from Lab 10 to be able to observe inputs and outputs in real time while the robot is operating.

14.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab.

- How does interrupt priority affect the behavior of the robot?
- Assume you knew turning about 90 degrees required you to run the motors for 2 seconds. How would you use interrupts to perform this operation?
- What factors affect latency on this robot?
- How would you use interrupts to run the finite state machine from Lab 6, as shown in Figure 7?
- What should you do in the main program to save power?

14.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. You could extend the system or propose something completely different. For example,

- Integrate Lab 11 so debugging information is displayed on the LCD.
- Integrate Lab 10 so debugging information is recorded into Flash ROM.
- Use debugging features within CCS to perform execution profiling.
- Use debugging features within CCS to perform power profiling on the MSP432. However, most of the power used in the robot is delivered from batteries to the motors, so CCS will not be able to monitor this power. To measure total power, you would need to current measurements from Lab 12.

14.8 Which modules are next?

This was our first of many uses of interrupts in this course. The following modules will build on this module:

Module 15) Interface IR distance sensors to the robot using the ADC.
Module 16) Interface tachometers to the microcontroller and use input capture to measure wheel velocity.
Module 17) Combine modules 12, 13 and 16 to develop closed loop motor controllers. In this module you will be able to spin the motors at a constant speed.

14.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module how to:

- Use edge-triggered interrupts to implement multithreading
- Use global variables to communicate between threads
- Perform execution profiling using port pins and a scope
- Perform high-level tasks on the robot

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265