The Mechatronics Revolution: Fundamentals and Core Concepts
Lab Assignment 5

Serial Communication-Based Control using UART, Distance Sensor Reading using Analog-to-Digital Conversion, and Periodic Interrupts using Timers

5.1 Objective

The main objectives of this lab are: i) to enable the TI-RSLK-Mechkit robot to be controlled by remote commands sent from a computer, ii) to construct a data acquisition system and transmit the collected data to a computer, and iii) to detect objects in front of the robot using an analog distance sensor, sampled by a periodic interrupt.

This lab uses onboard features of the MSP432 LaunchPad including UART, ADC and Timer. To this end, you will write a program that establishes 2-way serial communication between the MSP432 and your PC/laptop via UART. You will also configure the MSP432 Analog to Digital converter (ADC) for continuous conversion of analog distance sensor readings. The Timer_A module of MSP432 will be used to generate periodic interrupts to sample the ADC.

In Module 5, we studied sensors and micorocontoller interfacing, how to interface MSP432 through different types of serial communication protocols such as UART, I2C and SPI, and performing analog-to-digital conversion. We also learned how to stream data to a computer using the UART module of MSP432 LaunchPad. Generating periodic interrupts using timers was covered in Module 3. Specifically, the fundamental knowledge required for this lab assignment were provided in the following topics of the course:

Module 3, Topic 5, Lesson 2	Microcontroller Timers
Module 5, Topic 1, Lesson 4	Sensors for Mechatronic Systems: Proximity Sensors
Module 5, Topic 2, Lessons 1-3	Analog-to-Digital Conversion
Module 5, Topic 3, Lessons 1-4	Universal Asynchronous Receiver-Transmitter (UART)
Module 5, Topic 5, Lessons 1-4	Examples of Sensor-Microcontroller Integration

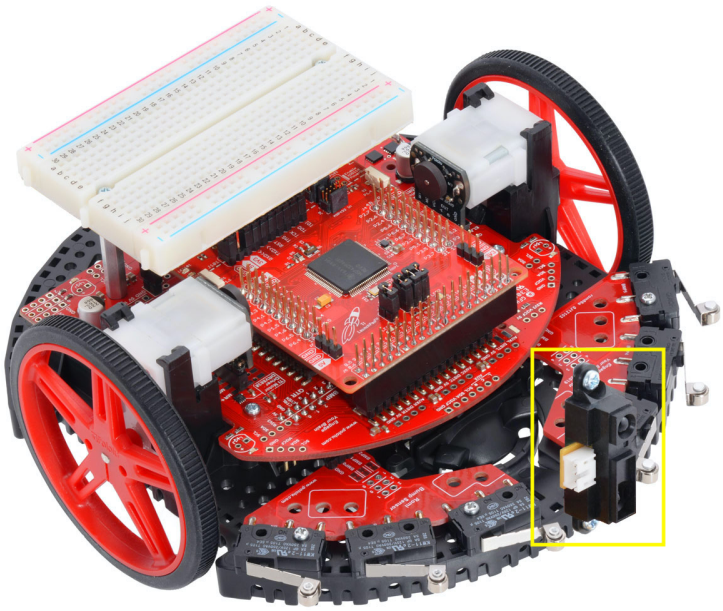


Figure 1: The TI-RSLK-Mechkit robotics system learning kit includes an analog distance sensor that can be used to detect obstacles in front of the robot.

5.2 Setup

This lab requires the *Code Composer Studio* IDE, a *SSH terminal* (e.g. Putty for Windows, Terminal for MacOS, Putty or Terminal for Linux), the *TI MSP432P401R LaunchPad*, and the *TI-RSLK-Mechkit* along with a *Sharp GP2Y0A21YK0F Analog Distance Sensor*.

Note: For more details on how to setup a SSH terminal, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).

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 disposable 1.5V alkaline batteries or rechargeable 1.2V NiMH batteries.

Four 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.

In this lab, the UART data is streamed through the USB debugging cable. Moreover, 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 during the lab and 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.

5.3 Problem Statement

Write a program that reads the analog output signal of the distance sensor at 1 second intervals, calculates the actual distance from the object in front of the robot, and transmits the last reading to the PC upon receiving a specific command over UART.

The output of the analog sensor is connected to pin P6.1 of MSP432, corresponding to the Channel 14 of the ADC14 module. The ADC peripheral converts an analog signal into a digital value consisting of 10 bits. That is, the ADC module must be configured to convert the voltage input in the range of +VCC (3.3V) and GND (0V) into a 10-bit value. The raw ADC digital value should be then translated to the actual "distance" in the range of 4" to 32" (10 cm to 80 cm) using an appropriate conversion equation.

Use Timer_A1 to generate periodic interrupts every 1 second to sample the ADC. *Sampling* is defined as triggering the ADC periodically, enabling the system to collect data in fixed time intervals.

The program needs to transmit and receive data between the PC and MSP432 LaunchPad over UART. It should receive a command data from the SSH terminal console on PC, and upon receiving a specific command, return the latest sensor reading to the PC via UART.

Specifically, the MSP432 should receive data from the SSH terminal over UART, and when a "carriage return" character is received (generated on the keyboard by pressing the "enter" or "return" key, associated to the HEX code 0x0D), it should transmit the latest distance reading as ASCII characters to the SSH terminal, so that the distance value (in *cm*) is printed out on the terminal. It should then wait for the next command to come in, while sampling the distance sensor in background. All other characters received from the SSH terminal should be ignored. This process is repeated in an infinite loop.

Note: For this lab assignment, it is recommended to use the UART receive (RX) interrupt, and use polling for UART transmit (TX).

Note: The LEDs on the MSP432 LaunchPad can be used for debugging purposes, for example to indicate receiving a character over UART.

More details about hardware and software requirements will be discussed in the next sections.

5.4 Background on Required MSP432 Peripherals

5.4.1 UART

Communication between two devices can be serial or parallel. In serial communication, data is transferred one bit at a time, as opposed to multiple bits in parallel communication. Serial communication is typically used in applications where I/O pins are limited and clock synchronization requirements make parallel communication difficult.

Serial communication can be further subdivided into two categories: Synchronous communication and Asynchronous communication. Synchronous communication uses a data channel to transmit data and a clock channel to maintain synchronism. In asynchronous mode, only a data channel is used and data is transmitted at a certain baud rate. The clock channel is absent in asynchronous communication - rather, clock signals and baud rate are separately generated on each individual device. Synchronous communication is used in applications that require continuous communication between two devices whereas asynchronous communication is used when intermittent communication is acceptable or required.

UART is a hardware device that implements asynchronous communication. The communicating devices first agree on a baud rate. Data is transmitted in groups of 8 bits with parity (optional) and other optional bits as available on the communicating devices. The data packet also consists of one start bit and one or two stop bits. All of these settings are matched for both devices during their respective configurations.

MSP432 contains two Enhanced Universal Serial Communication Interface (eUSCI) modules: `eUSCI_A` and `eUSCI_B`. The `A` module supports UART and SPI mode whereas the `B` module supports SPI and I2C modes. MSP432 can be used with up to eight serial communication channels. This lab focuses on configuring and using the MSP432 in the Universal Asynchronous Receiver Transmitter (UART) mode.

Once configured, data can be transmitted by writing to the `TXBUF` register or by using the function `UART_transmitData(EUSCI_A0_BASE)` from the DriverLib library. Similarly, received data is accessed by reading RXBUF or using the function:
`UART_receiveData(EUSCI_A0_BASE)`.

Flags are raised when transmission or reception of data are complete. The transmission complete flag TXIFG is raised when all the data is transmitted and there is no more data in TXBUF. This flag is cleared by writing data to be transmitted to the TXBUF. Similarly, the receive complete flag RXIFG is raised when a complete character (8 bits) is received. This flag is cleared by reading the RXBUF register. When these flags are raised, if enabled, corresponding interrupts are also triggered. The interrupt subroutine can then be used to either transmit new data or to read the received data.

Note: All received data from the SSH terminal console will be encoded in ASCII format, since it will be typed at the keyboard. To determine the HEX equivalent of an ASCII character, consult the ASCII encoding table found at: http://www.asciitable.com/

5.4.2 ADC14

The MSP432 has a user configurable Analog to Digital Conversion (ADC) module known as `ADC14`. There are 24 ADC channels which can be used in parallel or individually. Some of these channels have GPIO pins assigned to them to accept analog signals from external hardware. Each channel can be configured to provide 8-bit, 10-bit, 12-bit or 14-bit outputs. Using more bits will provide better resolution (i.e. smaller voltage increments). But using more bits also increases conversion time because the MSP432 uses a successive approximation ADC. More details on the number of clock cycles required for conversion can be found in

the MSP432 datasheet, available under Course Resources on edX (msp432p401r.pdf).

The first step in configuring the ADC is to determine the number of bits to be used. In this lab we will use *10-bit mode*. Secondly, a conversion mode must be determined. ADC14 offers four different modes of conversion. The one which can be used in this lab is the *Single Channel Single Conversion* mode. In this mode, once a conversion is initiated, the ADC will convert the signal into digital data from only one channel and stop without repeating until it is triggered again (by setting the `SC` bit in the `ADC14CTL0` register). Other modes available are: Multiple Channel Single Conversion, Single Channel Repeat Conversion, and Multiple Channel Repeat Conversion. More details on these modes can be found in the MSP432 Technical Reference Manual, available under Course Resources on edX (slau356i.pdf).

Memory should be allocated to save results of the conversion. A function provided in the DriverLib library (`ADC14_configureConversionMemory(...)`) can be used to configure memory. This function also sets the range of voltages to be used (`AVCC` source for our application, as we need the voltage source to be 3.3V), the channel to be selected and differential vs non-differential (in this lab we will use *non-differential*, so set this parameter to 0).

Warning: Make sure to use the 3.3V as the ADC voltage source, and not the internal voltage reference at 2.5V, as the output of the distance sensor may go up to 3.1V.

In summary, the DriverLib functions that must be called to configure and run the ADC are:

- `ADC14_enableModule`: Enables ADC
- `ADC14_setResolution`: Sets number of bits
- `ADC14_initModule`: Sets clock source and clock rate
- `ADC14_configureSingleSampleMode`: Configures conversion mode
- `ADC14_configureConversionMemory`: Allocates register to hold results
- `ADC14_enableSampleTimer`: Configures manual vs automatic triggering
- `ADC14_enableConversion`: Enables conversion
- `ADC14_toggleConversionTrigger`: Initiates a single conversion (trigger)
- `ADC14_isBusy`: Returns a boolean value that tells if a conversion/sample is in progress
- `ADC14_getResult`: Returns the conversion result for a specified memory channel

Please see the DriverLib User's Guide, available under Course Resources on edX (MSP432-DriverLib-Users-Guide.pdf) for the specific input/output arguments for each function.

After complete configuration, a conversion can be initiated by triggering the ADC using the `ADC14_toggleConversionTrigger()` function. The `ADC_isBusy()` function can be polled to determine if the conversion is still in progress. Once complete, the ADC result can be read using the `ADC14_getResult(...)` function and the conversion trigger can be toggled again to initiate a new conversion.

5.4.3 Timer_A

`Timer_A` of MSP432 is a 16-bit timer/counter which can support multiple capture/compares, PWM outputs, and interval timing. `Timer_A` also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

`Timer_A` features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with pulse width modulation (PWM) capability
- Asynchronous input and output latching

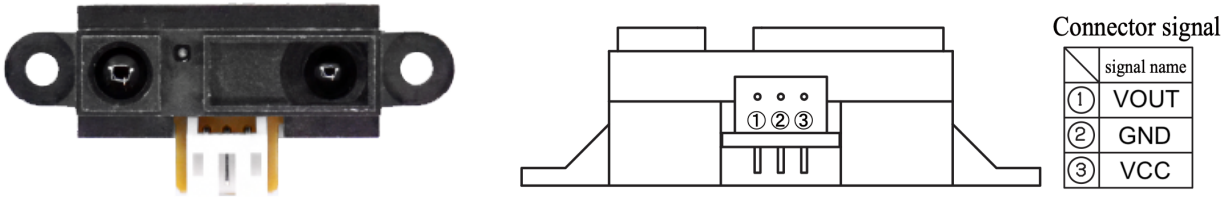
In this lab we use `Timer_A1` in *Up mode* and *Compare mode* to generate a *periodic interrupt* at specific time intervals.

The 16-bit timer/counter register, `TAxR`, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. `TAxR` can be read or written with software. Additionally, the timer can generate an interrupt when it overflows. `TAxR` may be cleared by setting the `TACLR` bit. Setting `TACLR` also clears the clock divider and count direction for up/down mode.

For more information regarding different modes of the `Timer_A`, please refer to MSP432P4xx Family Technical Reference Manual, available under Course Resources on edX (slau356i.pdf).

5.5 Hardware

5.5.1 GP2Y0A21YK0F Distance Sensor



(a) (b)

Figure 2: (a) GP2Y0A21YK0F distance sensor, (b) Pinout.

The GP2Y0A21YK0F distance sensor offers a wide detection range of 10 cm to 80 cm (4" to 32"). The distance is indicated by an analog voltage, so only a single analog input is

required to interface with the sensor. The sensor is powered by 5V, and the output voltage is in the range of 0V to 3.1V.

You will attach a GP2Y0A21YK0F distance sensor to the front of the robot (see the Robot Assembly Manual, available under Course Resources on edX (Robot-Assembly-Manual.pdf)). The sensor should be interfaced with the MSP432 LaunchPad using the `Channel 14` of the `ADC14` module (`A14`). Connect the output pin of the distance sensor directly to the MSP432 pin corresponding to this ADC channel (P6.1). Note that, in general, you can decide what ADC channel you would like to use. However, for grading purposes, you are required to use ADC Channel 14 in this lab.

Note: Although the distance sensor is powered by 5V, its output voltage does not go above 3.1V, so it is safe to connect the output signal directly to MSP432 pins. Remember that MSP432 pins are 3.3V tolerant.

The GP2Y0A21YK0F distance sensor has 3 pins: VOUT, GND, VCC. The pinout for the sensor is illustrated in Figure 2 (b). These 3 pins should be connected to the MSP432 LaunchPad using the 3-pin JST connector included in the TI-RSLK-Mechkit, as shown in the table below:

Distance Sensor Pin	MSP432 LaunchPad Pin
VOUT	P6.1
GND	GND
VCC	5V

Warning: Pay extra attention when wiring the sensor pins to the LaunchPad, as incorrect wiring may cause damage to the sensor. DO NOT rely on the wire colors of the JST connector. The pinout of the sensor is shown in Figure 2 (b), and the correct wiring is listed in the table above.

On my connector the red goes to ground and black is VCC, I stupidly didn't check the spec

The relationship between the sensor's output voltage and the inverse of the measured distance is approximately linear over the sensor's usable range. A graph of the analog output voltage as a function of the distance to a reflective object is shown in Figure 3. You can use this plot to convert the sensor output voltage to an approximate distance by constructing a best-fit line that relates the inverse of the output voltage (V) to distance (cm). To that end, we propose a conversion method in the following.

To have a monotonic behavior for the sensor, we only rely on the sensor readings in the range of 10 cm to 80 cm. In its simplest form, we can use the following equation to convert the output voltage of the sensor to the actual distance:

$$distance\ (cm) = 27.726 * voltage^{-1.2045}$$

For more information about the Sharp GP2Y0A21YK0F analog distance sensor, please refer to https://www.pololu.com/product/136. The datasheet for the Sharp GP2Y0A21YK0F sensor is available at https://www.pololu.com/file/0J85/gp2y0a21yk0f.pdf

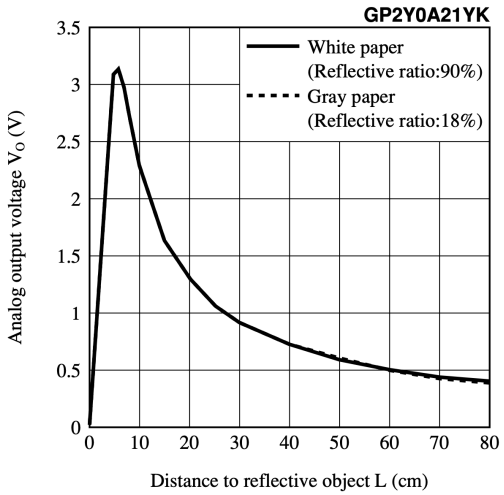


Figure 3: GP2Y0A21YK0F distance vs output voltage. (Image courtesy of Pololu).

5.6 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 5 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.

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 implemented on the breadboard in Lab 4 and will potentially damage the chassis board and/or the SN754410 IC.

Configure the UART module `EUSCI_A0` to enable transmit and receive and to establish a successful communication between the LaunchPad and your computer. Initialize the UART module using the functions from the DriverLib library (an example code was provided in Module 5, Topic 3, Lesson 4 of the course). First select a baud rate to use (standard baud rates include 9600, 38400, 57600 or 115200). The configuration parameters (clock divider and other register values) can be computed from `http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html`. Your UART configuration should use the above baud rate configuration parameters, the *SMCLK clock source*, *no parity*, *LSB first*, *UART mode*, *one stop bit*, and *oversampling*. For this lab, we recommend to use RX interrupt, but no interrupt is needed for TX functionality. Remember

to also configure the pins P1.2 and P1.3 with the primary functionality, associated to the `EUSCI_A0` module.

Configure the ADC14 module `Channel 14` in the *10-bit* single channel single conversion mode. Set the clock source to *SMCLK*, and allocate the register `ADC_MEM0` to hold the results from the channel `A14`, with *non-differential* functionality. Select the `AVCC` source as the voltage reference. Make sure to set the sample timer to *manual iteration*. Do not forget to enable the ADC14 conversion at the end of the initialization.

Configure the pin P6.1 (`A14`) to its tertiary function (which is ADC), by calling the DriverLib function:

```
MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6, GPIO_PIN1,
                                               GPIO_TERTIARY_MODULE_FUNCTION);
```

Configure `Timer_A1` such that it generates a periodic interrupt every 1 second (an example code was provided in Module 3, Topic 5, Lesson 2 of the course). Make sure to enable the `Timer_A1` interrupt and start the counter. Place the manual ADC conversion trigger along with the ADC reading procedure inside the `Timer_A1` interrupt handler (`TA1_0_IRQHandler`). You can also perform the ADC value conversion to the actual voltage and distance inside the `Timer_A1` ISR.

Note: Since all the events in this lab assignment can be handled inside the interrupt handlers, the main `while(1)` loop could remain empty. However, you still need to create an empty `while(1)` loop to prevent the code from exiting the program.

Design your code inside the UART ISR function (`EUSCIA0_IRQHandler()`) to read the received character for each RX interrupt. Upon receiving the carriage return character (`0x0D`) via UART, transmit the latest distance reading (in *cm*) back to the PC via UART. Use the `sprintf()` function to convert the distance variable into ASCII characters to be sent over UART (an example code was provided in Module 5, Topic 5, Lesson 2 of the course). When you convert the data, convert the distance as a floating point number with two decimal place (this can be achieved using the format descriptor "$\%.2f$" as an input to the `sprintf()` function).

```
/* convert the distance variable into an array of ASCII characters */
char stringData[10]; // to hold the ASCII characters to be transmitted out
int numChars = sprintf(stringData, "%.2f", distance_var);
```

Note: When data is transmitted from LaunchPad to the SSH terminal, your code could append a carriage return character (`0x0D`) and a newline character (`0x0A`) at the end to move the curser to the next line in the SSH terminal.

Note: Do not forget to clear the UART RX interrupt flag at the end of the ISR. Not clearing the interrupt flag results in the program not exiting the ISR.

Design your code inside the `Timer_A1` ISR function (`TA1_0_IRQHandler()`) to read the distance sensor output using the ADC14 module at every interrupt. Remember that the `Timer_A1` interrupt is triggered at 1 second intervals. As discussed in Module 5, for a successful ADC reading, you can toggle the conversion trigger, wait until the ADC conversion is complete (you can achieve this by placing the `ADC14_isBusy()` function as the condition for a `while()` loop), and then get the ADC result from the configured memory.
An example code to achieve this behavior could be the following (similar to the example code provided in Module 5, Topic 2, Lesson 3):

```
/* Read the ADC */
ADC14_toggleConversionTrigger();
while( ADC14_isBusy() );
adc_result = ADC14_getResult(ADC_MEM0);
```

Since the entire procedure takes only a few milliseconds, all these functions can be placed inside the timer ISR.

Since we are using the ADC in 10-bit mode, the ADC converts the output voltage of the sensor (which is in range of 0 to 3.3V) to a digital number in the range of 0 to 1023. To convert the digital number to the actual voltage in your program, you can multiply the ADC results by $\frac{3.3}{1023} = 0.0032$ and store the result as a `float` variable:

$$voltage\ (V) = ADC\ value * \frac{3.3}{1023}$$

```
// convert the ADC value to the actual voltage
voltage = adc_result * (3.3/1023.0);
```

You can then convert the GP2Y0A21YK0F sensor output voltage (V) to an approximate distance (cm) by using the following equation:

$$distance\ (cm) = 27.726 * voltage^{-1.2045}\ (V)$$

which can be implemented in C by including the `<math.h>` header file and then using the `pow()` function, as:

```
// convert the voltage to the actual distance in cm
distance_var = 27.726 * pow(voltage,-1.2045);
// call the grading macro here when generating the grading outputs:
MACRO_LAB5_ADC_EVENT(distance_var); // see Section 5.9
```

This equation gives an estimate of the actual distance in centimeters, however the accuracy is enough for the purpose of this lab.

Note: Do not forget to clear the "Capture/Compare" interrupt flag at the end of the timer ISR. Not clearing the interrupt flag results in the program not exiting the ISR.

Note: After an ADC conversion is complete, you can print the results to the CCS console using the `printf()` function for debugging purposes. This can be done by placing the print statement inside the same `Timer_A1` interrupt handler. However, make sure to comment out the `printf()` line when using the grading macros and generating the output text file.

To use `printf()`, you must include the `<stdio.h>` header file. No additional configuration is needed beyond this to use `printf()`.

Consult the DriverLib User's Guide and the MSP432 Technical Reference Manual for guidance as you set up your code, available under Course Resources on edX (MSP432-DriverLib-Users-Guide.pdf and slau356i.pdf).

5.7 Expected Performance

By setting breakpoints and other debugging tools (such as looking at variable values, using LEDs, using `printf()` function, etc.), make sure that the timer interrupt is triggered every 1 second, and the ADC results are read correctly and are converted to the actual voltage value and the actual distance properly.

Connect the robot to the PC using the LaunchPad USB debugging cable, and open the SSH terminal with appropriate UART configuration. Press the "enter" or "return" key to send the ASCII code of `0x0D` to the robot. The program should return the actual distance (as ASCII characters) to the SSH terminal, and the distance (in *cm*) should be printed out on the terminal. Move an object in front of the robot in the range of 10 to 80 cm, and press the enter key repeatedly to get the distance values. Make sure that the distance values are roughly correct.

Note that the robot should not respond to pressing other keys on the keyboard, as the program should ignore the received HEX characters other than `0x0D`.

Note: For more details on how to setup a SSH terminal, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).

5.8 Troubleshooting

Timer interrupt does not occur:

- Verify that the timer interrupt is initialized correctly.
- Set breakpoints using the CCS debugger to check whether the program is stuck in any specific function.
- Check the `Timer_A1` registers using the CCS debugger and verify them with the MSP432 Reference Manual.

UART RX interrupt does not occur:

- Verify that the UART RX interrupt is initialized correctly.
- Make sure that the UART is configured with the correct baudrate and UART parameters during the initialization.
- Verify that the SSH terminal is using the right COM port.
- Verify that the SSH terminal is configured with the right UART parameters.

ADC readings are noisy, not correct:

- Verify that the ADC14 module is initialized correctly.
- After toggling the ADC conversion, make sure that the program waits for the conversion to be complete by checking the `ADC14_isBusy()` function before reading the results.
- Double check the conversion of the raw ADC value to the voltage value and then to the distance value by setting breakpoints using the CCS debugger.

Cannot open the COM port with the SSH terminal to MSP432:

- Make sure that the SSH terminal is using the right COM port.

5.9 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_LAB5_INIT()` after initializing all the GPIOs, Timer, ADC, UART and other peripherals in your code (right before the main `while(1)` loop).
- You must call `MACRO_LAB5_ADC_EVENT(distance_var)` right after reading the ADC value and converting the raw data to distance. The macro input argument `distance_var` is the distance global variable (the variable name is optional) containing the last resulting distance value (in *cm*) from ADC module. For example, if using DriverLib functions to read ADC, you need to call the macro after the function call `ADC14_getResult()` and converting the ADC result to the distance in centimeters.

- You must call `MACRO_LAB5_UART_RX_EVENT(distance_var)` right after receiving a character via UART. The macro input argument `distance_var` is the distance global variable (the variable name is optional) containing the last resulting distance value (in *cm*) from ADC module.
  For example, if using the DriverLib functions to read UART RX, you need to call the macro right after the function call `UART_receiveData(...)`:

```
// receive a character via UART
uint8_t rxData = UART_receiveData(EUSCI_A0_BASE);
// call the grading macro
MACRO_LAB5_UART_RX_EVENT(distance_var);
```

- You must call `MACRO_LAB5_UART_TX_EVENT()` right after transmitting a character via UART. No input argument is needed for this macro.
  For example, if using DriverLib functions to transmit through UART, you need to call the macro after each function call `UART_transmitData(...)`.
  Only use this macro after transmitting each ASCII character of the distance value, resulted from the `sprintf()` function. DO NOT call the macro after transmitting any appended "carriage return" or "new line" characters.

```
// transmit a character via UART
UART_transmitData(EUSCI_A0_BASE, stringData[i]);
// call the grading macro
MACRO_LAB5_UART_TX_EVENT();
```

- 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.
  - Turn the robot on by pressing the Power button on the TI-RSLK chassis board. Make sure that the blue LED on the chassis board is turned on.
  - Start the debug mode in the Code Composer Studio. The code will be paused at the beginning of the main loop. DO NOT press the Resume (F8) button in CCS yet.
  - Start your SSH terminal and connect to the associated COM port.
  - Now press the Resume (F8) button in CCS to start debugging the code. The grading outputs should start printing in the CCS console.
  - Test your code by moving an object in front of the distance sensor and pressing the "enter" or "return" key for at least 20 times over a period of time to read the distance values.
  - 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.

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).

5.10 Optional Module (Non-graded): LCD Display and SPI Communication

Add a basic graphic LCD display module to the TI-RSLK chassis board, and print the distance measurement on it in real time. One such character display is the Sparkfun 10168, which has a SPI serial communication interface.

To add the display to the chassis board, you first need to identify the appropriate pins at the rear of the chassis board. The board has been labeled to help identify the right section (the Sparkfun 10168 section is on top).

More information about attaching an LCD display to the TI-RSLK robot can be found at https://training.ti.com/attaching-display-ti-rslk-max

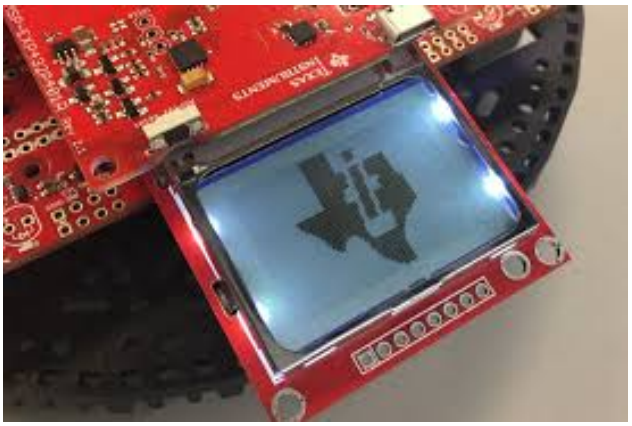


Figure 4: LCD display added to the TI-RSLK chassis board (Image courtesy of Texas Instruments).