IoT Platform:

Tutorial: Parallelism and Inter-Device Communication

Table of Contents

Introduction 3

Things Needed 3

Multithreading 4

UART Communication 6

References 9

| Revision history | | |
| --- | --- | --- |
| **Version** | **Date** | **Comment** |
| 1.0 | 06/15/16 | Initial release |
|  |  |  |
|  |  |  |

# Introduction

**This tutorial assumes that you have completed the “Distance Sensor Tutorial”** and understand the operation of the Sparkfun 9 Degree of Freedom (9DOF) block.

It is also recommended to first complete the IR Beacon and Receiver Array tutorial first.

In this tutorial, you will:

1. Learn about multithreading on a single Intel Edison
2. Learn about UART communication between two Intel Edisons

# Things Needed

* Two Intel Edisons
* Sparkfun blocks:
  + 9DOF Block
  + 2x Battery Block
  + 2x Base Block
  + Soldered GPIO Block from Distance Sensor Tutorial
  + Soldered GPIO Block from IR Receiver Array Tutorial
* 5x HC-SR04 Ultrasonic distance sensor
* Miscellaneous
  + Solder and soldering iron
  + Wires and wire cutter
  + Electric tape
  + Female and male headers

# Multithreading

**Overview**

With two CPU cores, the Intel Edison can run two threads of code simultaneously without interruption. This is useful in situations requiring two different types of tasks may need to be performed simultaneously and where accurate timing is important. In the case of the **pathfinding robot**, one Intel Edison must collect both distance and 9DOF data. The ultrasonic distance sensor requires the Edison to wait for a return signal, then precisely measure the duration of the pulse. Meanwhile, the 9DOF data should be collected with as high a frequency as possible. It is best that the two data collection processes occur simultaneously on separate cores so they do not interfere with each other and degrade the quality of the data.

**Hardware**

With the Distance Sensor Tutorial completed, the only hardware setup that is needed is to stack the Intel Edison, the Base Block, the GPIO Block from the distance sensor tutorial, the 9DOF Block, and the Battery Block

**Demonstration**

Obtain the files contained in the following link and place them into the **home directory of the root account** on the Edison: <https://github.com/zhgary/UCLA_EE180D_Robot/tree/master/integrated_distance_9dof_uart>

Run the following commands to compile and run the code:

**$ gcc -lmraa -pthread pthread2.c LSM9DS0.c -I . -o pthread2**

**$ ./pthread**

Upon success and with functional distance sensors, the console should repeatedly print the numeral “9” and the word “Distance”.

**Software Overview**

The “int main” function in the above program first initializes the GPIO needed to communicate with the distance sensors and 9DOF sensor. If initialization is successful it creates two threads to monitor each type of sensor, then goes to sleep.

Initialization of 9DOF via library functions:

accel **=** accel\_init**();**

set\_accel\_scale**(**accel**,** A\_SCALE\_2G**);**

set\_accel\_ODR**(**accel**,** A\_ODR\_100**);**

a\_res **=** calc\_accel\_res**(**A\_SCALE\_2G**);**

gyro **=** gyro\_init**();**

set\_gyro\_scale**(**gyro**,** G\_SCALE\_245DPS**);**

set\_gyro\_ODR**(**accel**,** G\_ODR\_190\_BW\_70**);**

g\_res **=** calc\_gyro\_res**(**G\_SCALE\_245DPS**);**

Initialization of ultrasonic distance sensors:

int i**;**

**for** **(**i**=**0**;**i**<**5**;**i**++)**

**{**

trig**[**i**]=NULL;**

echo**[**i**]=NULL;**

**}**

trig**[**0**]** **=** mraa\_gpio\_init**(**32**);**

echo**[**0**]** **=** mraa\_gpio\_init**(**46**);**

trig**[**1**]** **=** mraa\_gpio\_init**(**31**);**

echo**[**1**]** **=** mraa\_gpio\_init**(**45**);**

trig**[**2**]** **=** mraa\_gpio\_init**(**33**);**

echo**[**2**]** **=** mraa\_gpio\_init**(**47**);**

trig**[**3**]** **=** mraa\_gpio\_init**(**25**);**

echo**[**3**]** **=** mraa\_gpio\_init**(**13**);**

trig**[**4**]** **=** mraa\_gpio\_init**(**21**);**

echo**[**4**]** **=** mraa\_gpio\_init**(**00**);**

**for** **(**i**=**0**;**i**<**5**;**i**++)**

**{**

**if** **(**trig**[**i**]** **==** **NULL** **||** echo**[**i**]** **==** **NULL)**

**{**

fprintf**(**stderr**,** "Initialize failed"**);**

**return** 1**;**

**}**

**else**

**{**

printf**(**"begin"**);**

**}**

mraa\_gpio\_dir**(**trig**[**i**],** MRAA\_GPIO\_OUT**);**

mraa\_gpio\_dir**(**echo**[**i**],** MRAA\_GPIO\_IN**);**

**}**

Thread creation:

pthread\_t thread1**,** thread2**;**

const char **\***message1 **=** "Thread 1"**;**

const char **\***message2 **=** "Thread 2"**;**

int iret1**,** iret2**;**

//create thread 1

iret1 **=** pthread\_create**(** **&**thread1**,** **NULL,** print\_message\_9DOF\_function**,** **(**void**\*)** message1**);**

**if(**iret1**)**

**{**

fprintf**(**stderr**,**"Error - pthread\_create() return code: %d\n"**,**iret1**);**

exit**(**EXIT\_FAILURE**);**

**}**

//create thread 2

iret2 **=** pthread\_create**(** **&**thread2**,** **NULL,** print\_message\_distance\_function**,** **(**void**\*)** message2**);**

**if(**iret2**)**

**{**

fprintf**(**stderr**,**"Error - pthread\_create() return code: %d\n"**,**iret2**);**

exit**(**EXIT\_FAILURE**);**

**}**

printf**(**"pthread\_create() for thread 1 returns: %d\n"**,**iret1**);**

printf**(**"pthread\_create() for thread 2 returns: %d\n"**,**iret2**);**

//wait for threads to finish, but this will never happen

pthread\_join**(**thread1**,** **NULL);**

pthread\_join**(**thread2**,** **NULL);**

The code that runs in a thread must be contained within a function. In this case it can be seen that the first thread runs print\_message\_9DOF\_function while the second thread runs print\_message\_distance\_function. These functions run infinite loops which repeatedly collect data from the sensors. Note that it is also possible to pass a message to a thread upon its creation, via the fourth argument of the pthread\_create function. This pointer variable becomes the first parameter of the function that the thread runs. This is useful for threads that run the same code but must have some method to distinguish themselves.

# UART Communication

**Overview**

There are occasions where parallel code running on a single Intel Edison is not adequate for the desired embedded or IoT system. One approach is to use multiple Edison devices that communicate with each other.

UART is one of the few methods of wired data transmission between two Intel Edisons, and it offers advantages in speed and reliability. Most interfaces, such as I2C, do not work between two Intel Edisons because one Edison must serve as a slave, which it is not capable of doing. UART is one of the few wired interfaces that can function between two Intel Edisons because it can be made to work despite both Edisons assuming the role of a server or master.

To allow communication between two Intel Edisons, the TX and RX signals of UART must be crossed over. TX of the first Edison should be connected to RX of the second Edison, and vice versa. It goes without saying that the grounds should also be connected.

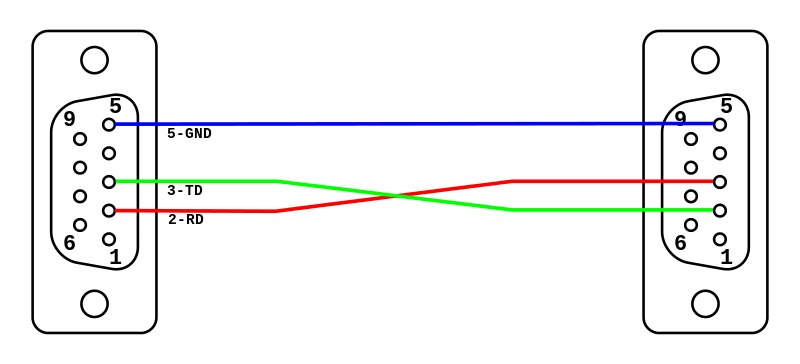


Figure TX / RX Crossover or “Null Modem” on DE-9 connectors

**Hardware**

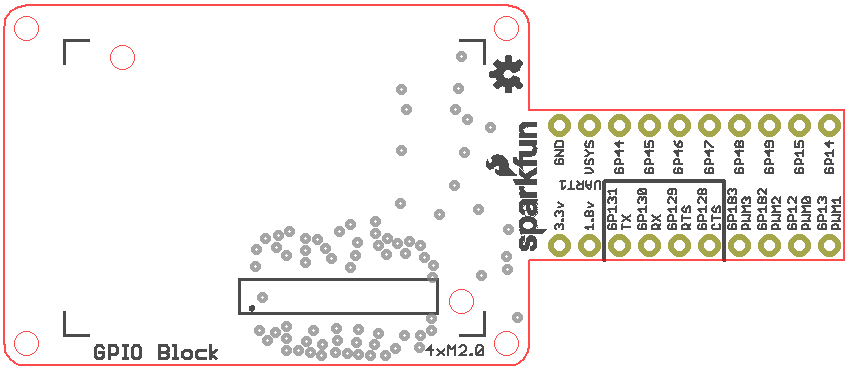
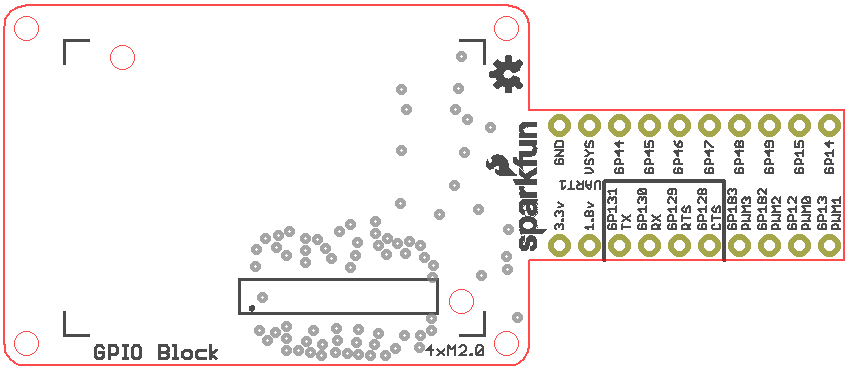


Figure Wiring Diagram

1. The GPIO block used for the Distance Sensor Tutorial and the first part of this tutorial now needs a second ground wire as well as TX and RX wires soldered to it. For attaching the second ground wire, it is possible to desolder, from the GPIO block, the existing ground wire for the distance sensors and further strip the wire so that another ground wire can be twisted around and soldered to it. Then the combination can be soldered back into the GPIO block.
2. Now that ground, TX, and RX wires are soldered to the GPIO block, solder the other ends to a 3 pin male header.
3. Now the other GPIO block needs ground, TX, and RX wires soldered to it. If you have completed the IR Receiver Array Tutorial, use the GPIO block from that tutorial and simply add the wires to the GPIO block.
4. Finally, solder the loose ends of the other ground, TX, and RX wires to a 3 pin female header. Remember to cross over the TX and RX wires, so that when plugged onto the male header from the first GPIO block, TX from the first GPIO block is connected to RX of the second GPIO block and vice versa.
5. For the first Edison, stack the Sparkfun Blocks in the same way as in the first part of this tutorial. For the second Edison, stack the Edison, the Base Block, GPIO Block, and Battery. Finally, connect the UART male and female headers together.

**Demonstration**

1. No extra code is needed for the first Edison with the 9DOF and distance sensors. Simply run **$ ./pthread2**.
2. For the second Edison, obtain code from here: <https://github.com/zhgary/UCLA_EE180D_Robot/tree/master/receiveUART>
3. Compile an run the code on the second Edison with the following commands:  
   **$ gcc -lmraa receiveUART6.c –o receiveUART6  
   $ ./receiveUART6**
4. If the first Edison has not encountered any issues within its program or the sensors, and the Edisons are linked via UART, then upon executing the program in the second Edison, the terminal should display 9DOF and distance sensor data received from the first Edison.

**Software Overview**

Data from 9DOF and distance sensors come in the form of floating point numbers, and in sets of 7 values for the 9DOF sensor, and sets of 5 values for the distance sensor. These must be converted into a format that can be transmitted over UART.

First, the floating point numbers must somehow be sent in the form of 8 bit characters over UART. In this case the code reinterprets the binary data of the floating point numbers as characters using C unions:

//originally found in the 9DOF library

union u**{**

float f**;**

char s**[**4**];**

**};**

//adapted from the above

union du**{**

double d**;**

char s**[**8**];**

**};**

For union u, when a float is stored in, the equivalent data can be found as 4 chars, and the similar is true for union du.

Since the numbers are transmitted in sets, there must be some way to delineate the beginning and end of a set of numbers. An extremely basic way to accomplish this is to use magic numbers, which are values that are unlikely to be mistaken by something else. In this case, a sequence of four 0xFF bytes precede a 9DOF data set, a sequence of four 0xAA bytes precede a distance sensor data set, and the sequence 0xDE, 0xAD, 0xBE, 0xEF follows both types of data sets.

The final issue with sending data through UART is to make sure that only one thread at a time accesses UART. This is accomplished by a mutex lock.

Declaration of the lock:

//this is a global variable

pthread\_mutex\_t mutex1 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

Lock usage. While one thread has the lock, the other thread must wait until the lock is released.

//this occurs in both the 9DOF and distance sensor threads

pthread\_mutex\_lock**(&**mutex1**);**

mraa\_uart\_write**(**uart**,** buffer**,** **sizeof(**buffer**));**

pthread\_mutex\_unlock**(&**mutex1**);**

Meanwhile, the receiver code waits for availability of data in UART and decodes the resulting message. The implementation of this is left to the reader; receiveUART6.c contains an example implementation.

# References

1. https://en.wikipedia.org/wiki/Null\_modem
2. https://cdn.sparkfun.com/datasheets/Dev/Edison/GPIO\_Block.zip