Using MRAA and UPM to connect Sensors to Raspberry Pi

Biss00001 Motion Sensor

Raspberry Pi is a popular platform for learning IoT as it offers a complete Linux server in a tiny platform for a very low cost. This paper provides the steps involved in connecting a PIR Motion Sensor - Biss00001 to Raspberry Pi and reading the sensor values from it using libmraa and libupm libraries.PIR sensors, often referred to as, "Passive Infrared" or "IR motion" sensors, enable you to sense motion. Everything emits a small amount of infrared radiation, and the hotter something is, the more radiation is emitted. PIR sensors are able to detect a change in IR levels of their detection zone (e.g. when a human enters a room) and hence sense motion.

The PIR sensor that is used here has three pins: ground, digital out and 3-5VDC in. At idle, when no motion has been detected, the digital out will remain low, however when motion is detected, the digital out will pulse high (3.3V) and Raspberry Pi is used to sense this!

For this projects, the following is required

Hardware:

- Raspberry PI 2 & SD Card with Raspbian Operating System
- USB Power Supply
- USB Cable
- Breadboard & Jumper Wires
- Biss0001 Motion Sensor

The sensor has 3 Pins. Pin 1 is VCC, Pins 2 is Data, Pin 3 is Ground.

- Connect Pin 1 and Pin 3 is connected to the VCC and GND of Pi (5v) with the help of Breadboard
- Similarly Connect Pin 2(Data) to GPIO 23 [pin 16]
- Led is directly connected to output (Data pin) of Motion Sensor.

Once the connection is done, the next step is to install the libraries in Pi.

A brief description about libmraa and libupm and how we can install it on Raspberry Pi for development in C/C++ is provided below

libmraa - Low Level Skeleton Library for Communication on GNU/Linux platforms

Libmraa is a C/C++ library with bindings to javascript & python to interface with the IO.
The intent is to make it easier for developers and sensor manufacturers to map their sensors & actuators on top of supported hardware and to allow control of low level communication protocol by high level languages & constructs.Use of libmraa does not tie you to specific hardware with board detection done at runtime you can create portable code that will work across the supported platforms.

The intent is to make it easier for developers and sensor manufacturers to map their sensors & actuators on top of supported hardware and to allow control of low level communication protocol by high level languages & constructs.

Steps to install MRAA

Install git,python-dev,pcre,swig and cmake - this are required for building Libmraa library.
The steps to install all these libraries are as follows

sudo apt-get update
sudo apt-get upgrade
apt-cache search pcre
sudo apt-get install libpcre3 libpcre3-dev
sudo apt-get install cmake
sudo apt-get install swig

sudo apt-get install git
apt-get install python-dev

git clone https://github.com/intel-iot-devkit/mraa.git

Building mraa

mkdir mraa/build

cd build
cmake .. -DBUILDSWIGNODE=OFF
make
sudo make install

Important: Make sure you run the final command, “make install” with root or “sudo.”

-DBUILDSWIGNODE turns off NodeJs

To use the library in C or C++ programs, we need to add it to our shared library cache. With root (or using “sudo”), open up the ld.so.conf file:

nano /etc/ld.so.conf
/usr/local/lib/i386-linux-gnu/
You can check to make sure that the cache was updated by typing the command

ldconfig -p | grep mraa

To Compile and Execute a Program using c++

g++ filename.cpp -o outputfilename -lmraa

run your program as sudo ./outputfilename

UPM

UPM is a high level repository for sensors that use MRAA. The list of sensor supported can be found in the link here

http://iotdk.intel.com/docs/master/upm/modules.html

Steps to install UPM

This project depends on libmraa, so that needs to be installed first. Append the install location of mraa pkgconfig to the following environment variable:

PKG_CONFIG_PATH=$PKG_CONFIG_PATH:.../mraa/build/lib/pkgconfig

UPM will attempt to build all directories inside src/ and they must contain individual CMakeLists.txt files.

```
mkdir build
cd build
cmake ..
make
make install
```

The last command will create the include/ and lib/ directories with a copy of the headers and library objects respectively in your build location. Note that doing an out-of-source build may cause issues when rebuilding later on.

Additional features

Changing install path from /usr/local to /usr

```
-DCMAKE_INSTALL_PREFIX:PATH=/usr
```

Building debug build:

```
-DCMAKE_BUILD_TYPE=DEBUG
```

Disabling python module building

```
-DPYTHON_LIBRARY:FILEPATH=/usr/lib/libpython2.7.so.1.0
```

Building c++ examples

```
-DBUILDEXAMPLES=ON
```

Compile and execute program in c++

Include the upm module that u have used in your program as each of the classes listed in these libraries are completely separate so you will need to link against all those that you have used in your program

For example,using grove libraries

```
g++ filename.cpp -o outputfile -lupm-grove -I /usr/local/include/upm
```

If you get an error as “cannot find libupm.grove.so “, u need to export LIBRARY_PATH
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/my_library/path.so.something
$ export LD_LIBRARY_PATH

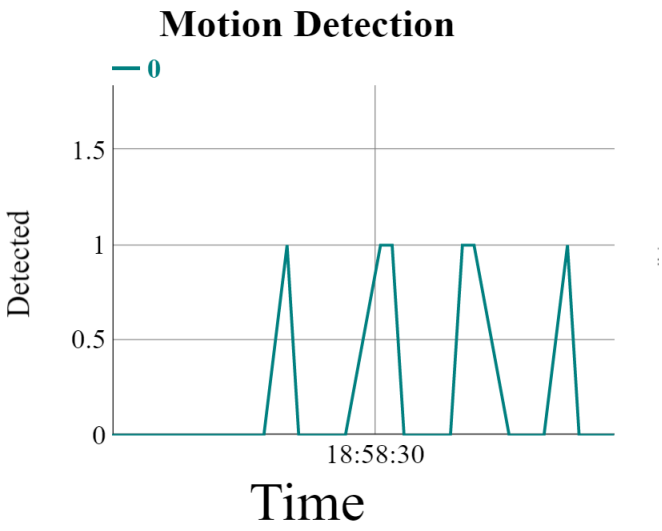
run your program as sudo ./outputfilename

After installing the above libraries,
I used the biss0001.cxx file which was provided in the sample examples section of libupm git repository and edited it to add the instantiate the motion sensor object on the GPIO pin used in the setup and also changed the output to print the date and motion->value();

The program was executed as follows,
g++ motionSensor.cxx -o motionSensor -lupm-biss0001 -I/usr/local/include/upm

Display:

The output of this program is appended to a csv file which is read by the library(Dygraph) and the output is displayed in a html page.Then use your browser to navigate to Pi address/motion.html. The graph looks like this,



Steps to install Apache Webserver and Dygraph

sudo apt-get install apache2
wget -P /var/www http://dygraphs.com/dygraph-combined.js
html file was placed in /var/www/html folder and the below script was written which can be executed to read the output every 1 sec

exec.sh

```
rm /var/www/html/motion.csv
while true; ./motionSensor do  >> /var/www/html/motion.csv; sleep 1; done
```

The html program looks like this,

```
<html>
<head>
<script type="text/javascript"
src="node_modules/dygraphs/dygraph-combined-dev.js"></script>
</head>
<body>
<div id="graphdiv1"
style="width:100%; height:100%"></div>
<script type="text/javascript">
g1 = new Dygraph(
document.getElementById("graphdiv1"),
"motion.csv", // path to CSV file
{title: 'Motion Detection',
legend: 'always',
strokeWidth: 1.5,
visibility: [true],
xLabelHeight: 30,
yLabelHeight: 30,
valueRange:[0,2],
xlabel:'Time',
ylabel:'Detected'}
);

window.intervalId = setInterval(function() {
g1.updateOptions( { 'file': "motion.csv" } );
}, 1000);
</script>
</body>
</html>
```

Grove Temperature and Humidity Sensor [DHT11]

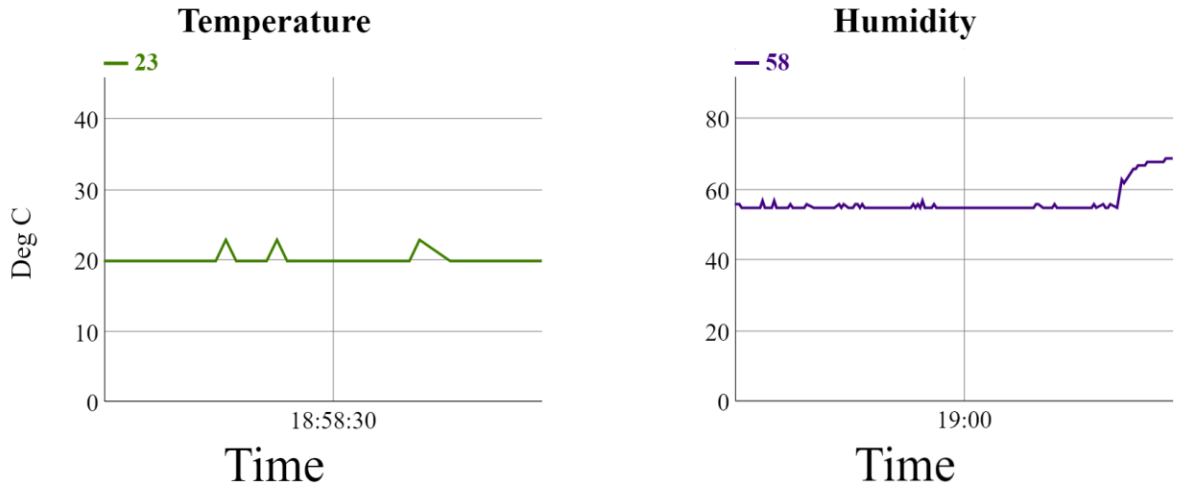
Connection Setup

Connect Pin 1 and Pin 3 is connected to the VCC and GND of Pi (5v) with the help of Breadboard
Similarly Connect Pin 2(Data) to GPIO 17 [pin 11]

Reading Sensor values

The Grove temperature library of UPM reads the AIO input [GroveTemp() object internally invokes mraa_aio_init()]. So I was not able to use this library to read the sensor. None of the other drivers in the library also does not work with DHT11.
A program which was written using mraa api to compute the temperature and humidity results gave incorrect results.
However I was able to read the temperature values using Python GPIO libraries. I plotted a graph using dygraph similar to the steps mentioned in connecting motion sensor and was able to read correct temperature results with it.

The graph looks like this,



Here is the link to the Python Library that I used to read the DHT11 sensor on Raspberry Pi

https://github.com/szazo/DHT11_Python