Peripheral programming with Intel® Edison

1 The MRAA library

Libmraa is a C/C++ library with bindings to Java, Python and JavaScript to interface with the IO on Galileo, Edison & other platforms, with a structured and sane API where port names/numbering matches the board that you are on. 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.

By default, the libmraa is already installed on the Edison. Write the code for mraa_demo.c and its makefile as in Listing 1, Listing 2.

Listing 1: mraa_demo.c

```
#include <mraa.h>

int main(int argc, char** argv)
{
      const char* board_name = mraa_get_platform_name();
      printf("hello mraa\n Version: %s\n Running on %s\n",
            mraa_get_version(), board_name);
      mraa_deinit();
      return MRAA_SUCCESS;
}
```

Listing 2: Makefile

```
all:
      gcc mraa_demo.c -lmraa -o mraa_demo
clean:
      rm mraa_demo
```

Compile and run the above program with **make** and **./mraa_demo**. If all goes well, the program should output the library version and the board's name. The API document of this libmraa can be found at here.

2 Install Screen on Edison

Screen is a full-screen window manager that multiplexes a physical terminal between several processes, typically interactive shells. On Edison, it can be used to monitor the serial port.

1. Copy **screen_4.0.3-r4_core2-32.ipk** to your Edison.
2. On your Edison, issue the command:
   opkg install screen_4.0.3-r4_core2-32.ipk
3. Run **screen -v** to make sure it has been successfully installed.

3 Sparkfun GPIO block

The General Purpose Input/Output (GPIO) Block breaks out the simple GPIO functionality of the Intel Edison. Using selectable level shifters, it is possible to use VSYS or 3.3v logic levels with this Block.

3.1 Board overview

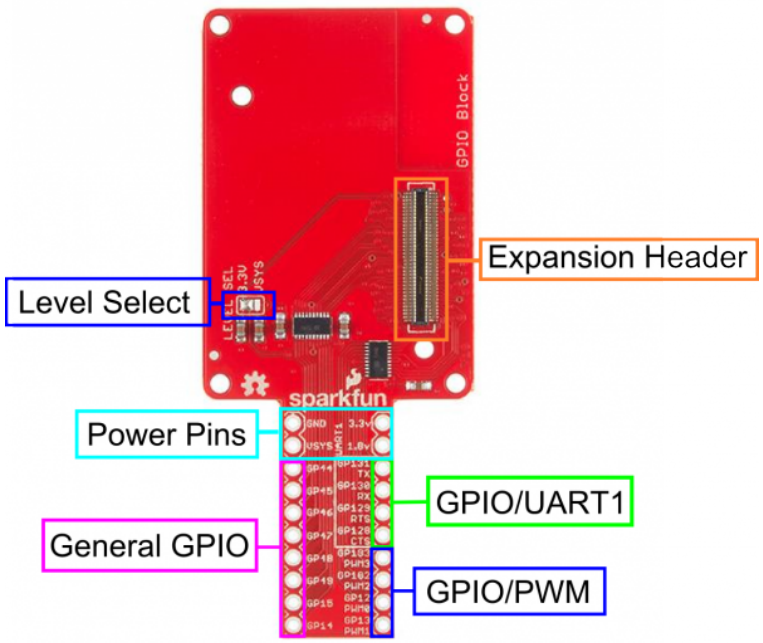

Figure 1: GPIO Block Functional Diagram

- Level Select - Jumper selects GPIO reference voltage, Can be set to 3.3v or VSYS
- Power Pins - Raw access to power pins on Edison
  - GND - Ground pin for all blocks and Edison
  - VSYS - Raw input for Edison and all Blocks.
    - Normal output (with power blocks) 4.0V-4.1V.
    - You can power an Edison through this pin. Acceptable voltages 3.3V-4.5V
  - 1.8v - 1.8v supplied by Edison internal power supply
  - 3.3v - 3.3v supplied by Edison internal power supply
- General GPIO - Genaral use GPIO pins.
- GPIO/UART1 - GPIO pins that can also be used as a second UART. (Useful for GPS receivers and other serial devices)
- GPIO/PWM - GPIO pins capable of generating PWM waveforms. (Useful for LED dimming and Motor control)
- Expansion Header - The 70-pin Expansion header breaks out the functionality of the Intel Edison. This header also passes signals and power throughout the stack. These function much like an Arduino Shield.

3.2 Using the board

Assemble this board under the Base block. Listing 3 shows an example of initializing GP14 as an output pin. Note that libmraa uses different pin numbers[1] so it is essential to assert the **raw** parameter of the Gpio constructor (or else you have to declare pin 36). Compile the program using the makefile in Listing 4.

Listing 3: mraa_gpio.c

```
#include <iostream>
#include <mraa.hpp>

using namespace std;

int main(int argc, char** argv)
{
    mraa::Gpio* gpio = new mraa::Gpio(14, true, true); //or Gpio(36);
    if (gpio == NULL)
    {
        return mraa::ERROR_UNSPECIFIED;
    }

    mraa::Result response = gpio->dir(mraa::DIR_OUT);
    if (response != mraa::SUCCESS)
    {
        mraa::printError(response);
        return 1;
    }

    //Do some stuff ...

    delete gpio;

    return 0;
}
```

Listing 4: Makefile

```
all:
    g++ mraa_gpio.c -lmraa -o mraa_gpio
clean:
    rm mraa_gpio
```

3.3 UART1

The default device file of UART1 on Edison is **/dev/ttyMFD1**. Listing 5 shows an example of sending a string via UART1.

1. Connect two Edison boards' UART1 pins (common ground).

[1] https://iotdk.intel.com/docs/master/mraa/edison.html

2. Issue the following command (same function as TeraTerm on Windows) on one board to monitor the UART1 device file.
   screen /dev/ttyMFD1 115200

3. Compile and run the program on the other board.

Listing 5: mraa_uart.c

```
#include <iostream>
#include <mraa.hpp>

using namespace std;

int main(int argc, char** argv)
{
    mraa::Uart* uart;
    try
    {
        uart = new mraa::Uart("/dev/ttyMFD1");
    } catch (std::exception& e) {
        cout << "Error while setting up raw UART\n";
        std::terminate();
    }
    if (uart->setBaudRate(115200) != mraa::SUCCESS)
    {
        cout << "Error while setting baudrate\n";
    }
    if (uart->setMode(8, mraa::UART_PARITY_NONE, 1) != mraa::SUCCESS)
    {
        cout << "Error while setting mode\n";
    }
    if (uart->setFlowcontrol(false, false) != mraa::SUCCESS)
    {
        cout << "Error while setting flow control\n";
    }

    uart->flush();
    uart->writeStr("Hello");
    delete uart;
}
```

4 Problem 1: UART echo

In this problem, you have to implement an UART echo application that sends back any data it has received.
Submit: source code, makefile, information about CPU & Memory usage.

5 Problem 2: LED controlling

In this problem, you have to implement an application to remote-control a led via TCP. There are 2 programs to complete: **remote_led_server** and **remote_led_client**.

- **remote_led_server** runs on a board which directly connects to the led via GP14. This program waits for commands from clients and processes them.
- **remote_led_client** runs on the other board which resides in the same subnet. The synopsis of this program is as follows:

  ./**remote_led_client** <IPADDR>:<PORT> [commands ...]
  Commands:
  - on: turn on the led.
  - off: turn off the led.
  - blink <time>: blink the led with the delay time measured in millisecond.

  Example: ./remote_led_client 172.28.10.202:5880 blink 1000

Submit: source code, makefile, a protocol document, information about CPU & Memory usage.