CS111
Computer Systems Principles
Section 1E Week 8

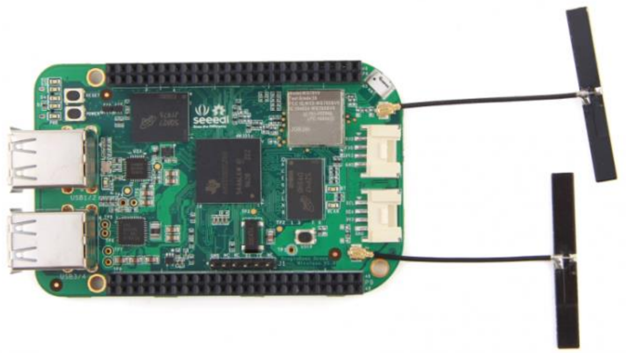
Tengyu Liu

Project 4A

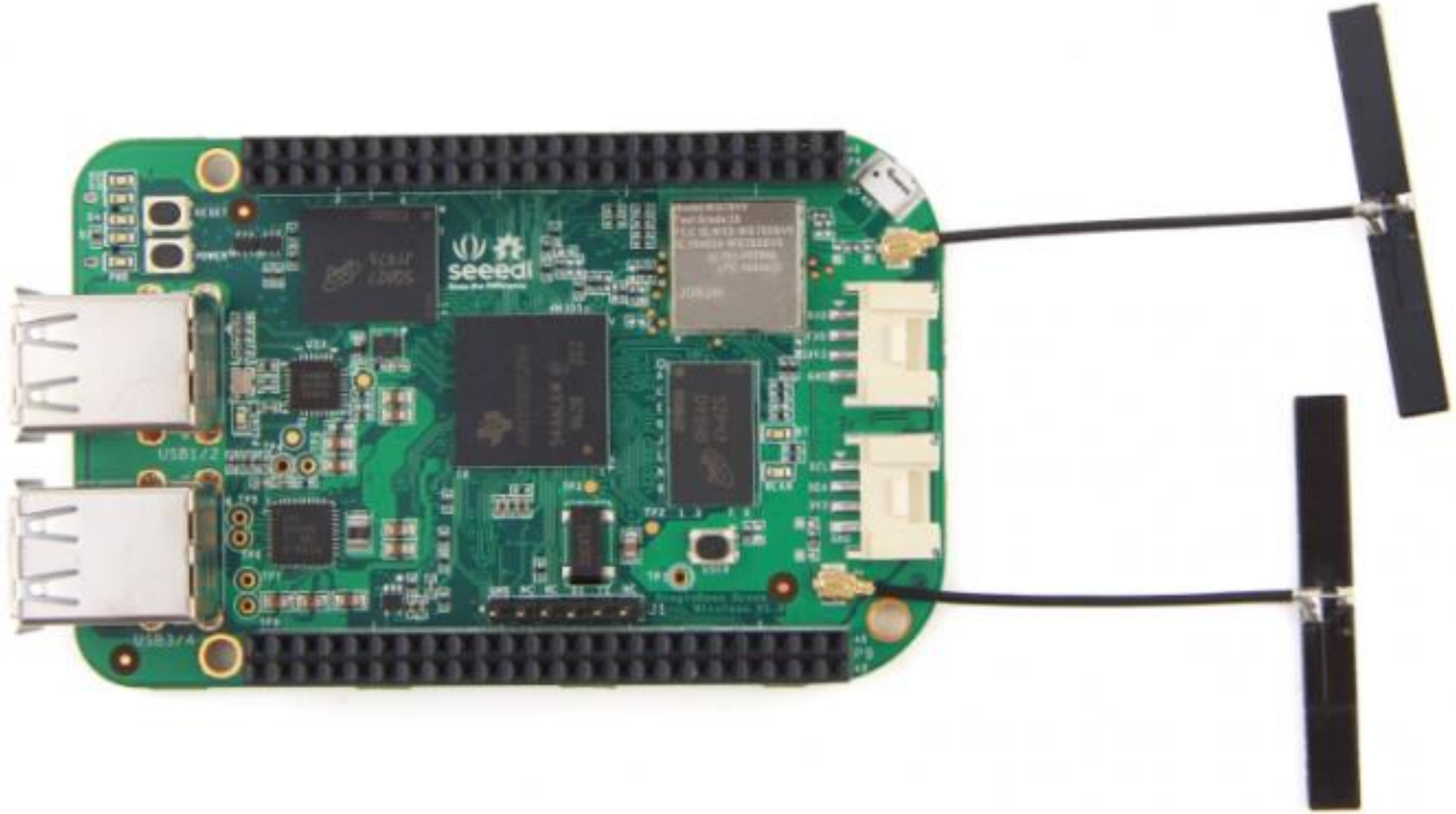
- Objective
  - Make sure that you
    - Have a beaglebone
    - Can log in to your beaglebone
    - Can transfer files from/to beaglebone
    - Have a working beaglebone development environment
    - Can install new packages on your beaglebone
- Deliverables
  - A set of images to prove that you have met the above objectives (check spec)

What is a beaglebone

- A credit-card-sized low-power mini-computer
  - Similar to a Raspberry Pi
- Runs Linux
- Has pins for sensors
- Does not natively support input/output devices like Raspberry Pi
  - Connects to PC console
- We will be using BeagleBone Green Wireless
  - Supports Wi-Fi + Bluetooth Low Energy
  - 1GHz CPU
  - 512MB DDR3 RAM
  - 4GB flash storage

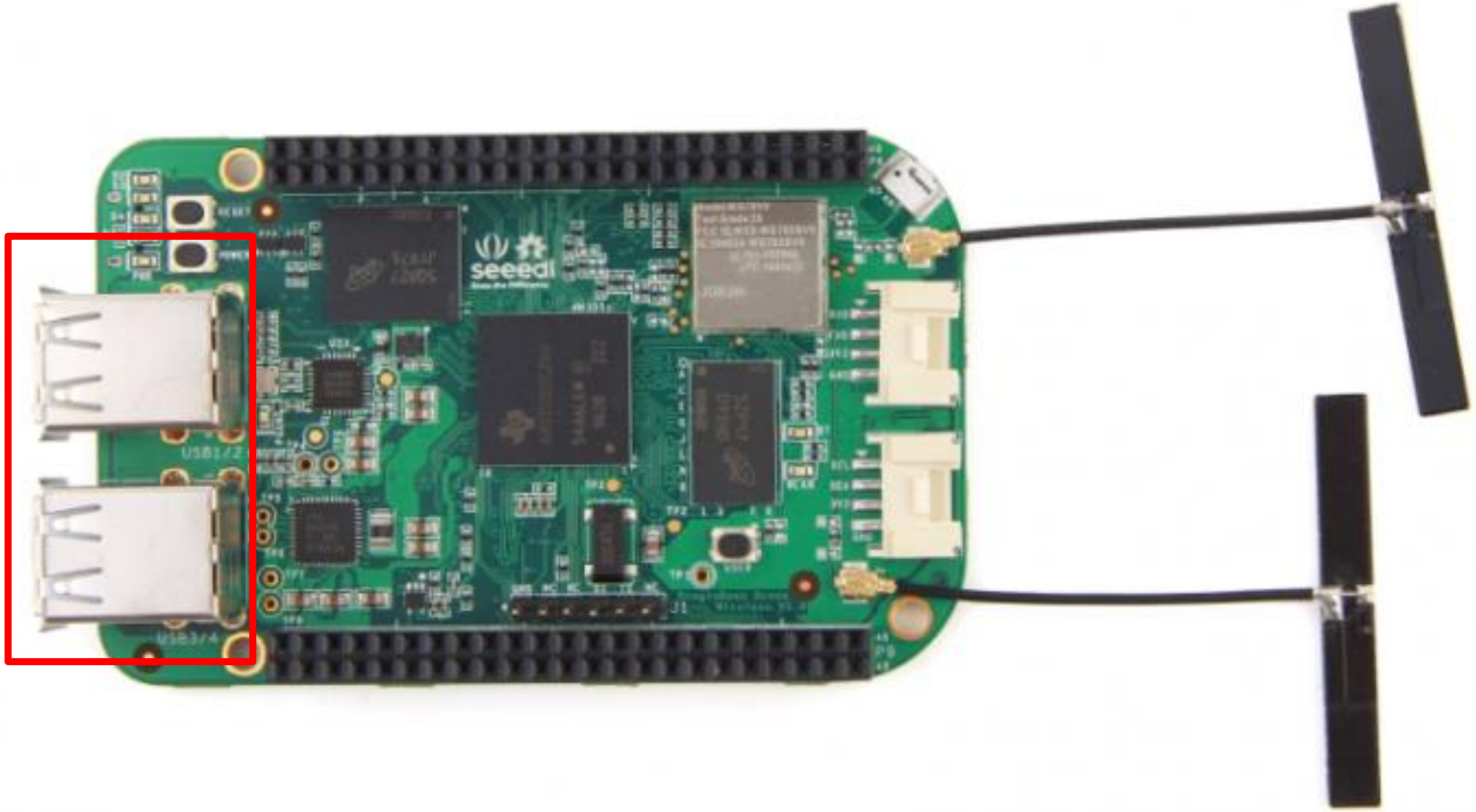


What is a beaglebone



What is a beaglebone

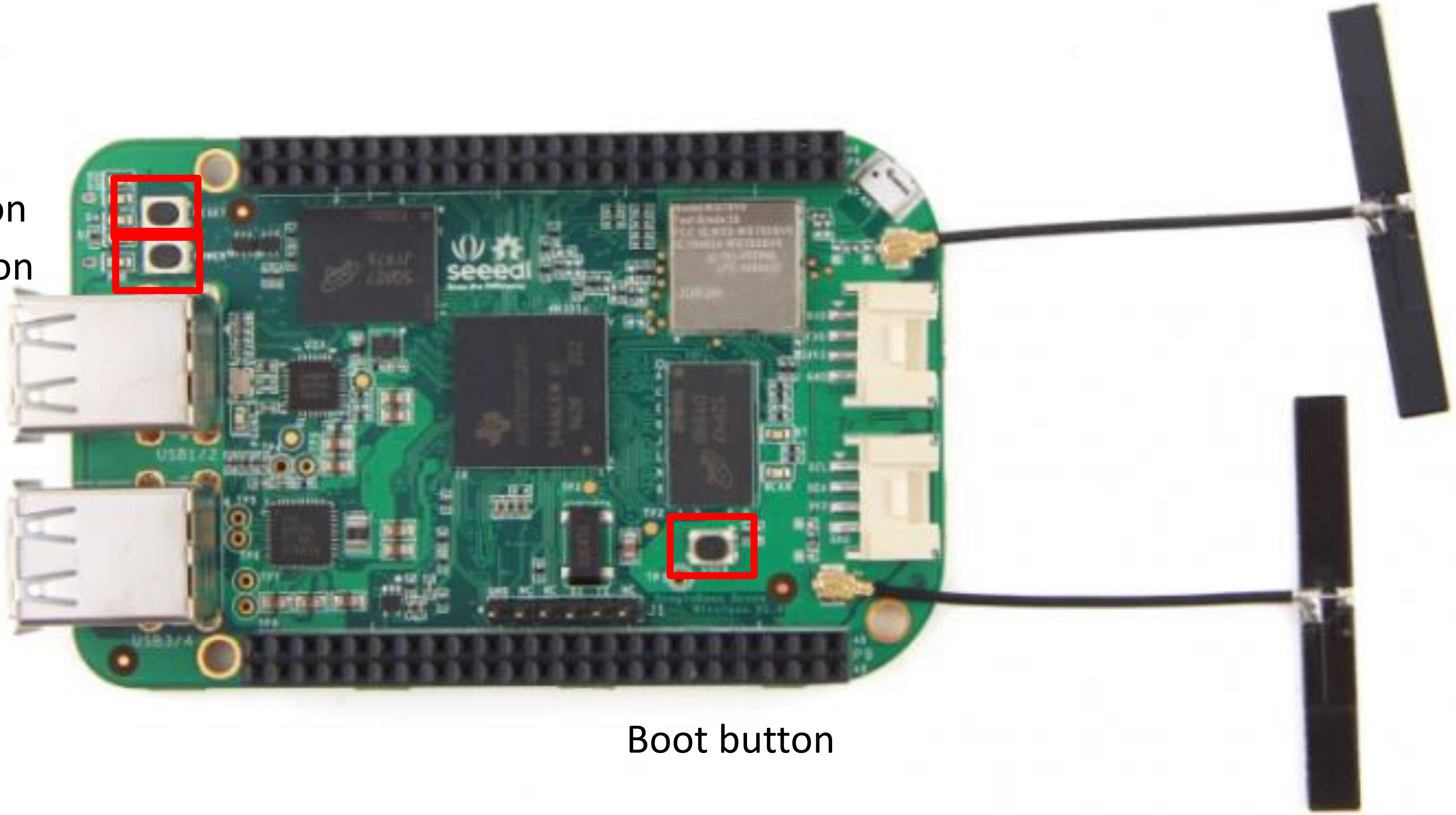
2x2 USB Port



What is a beaglebone

Reset button

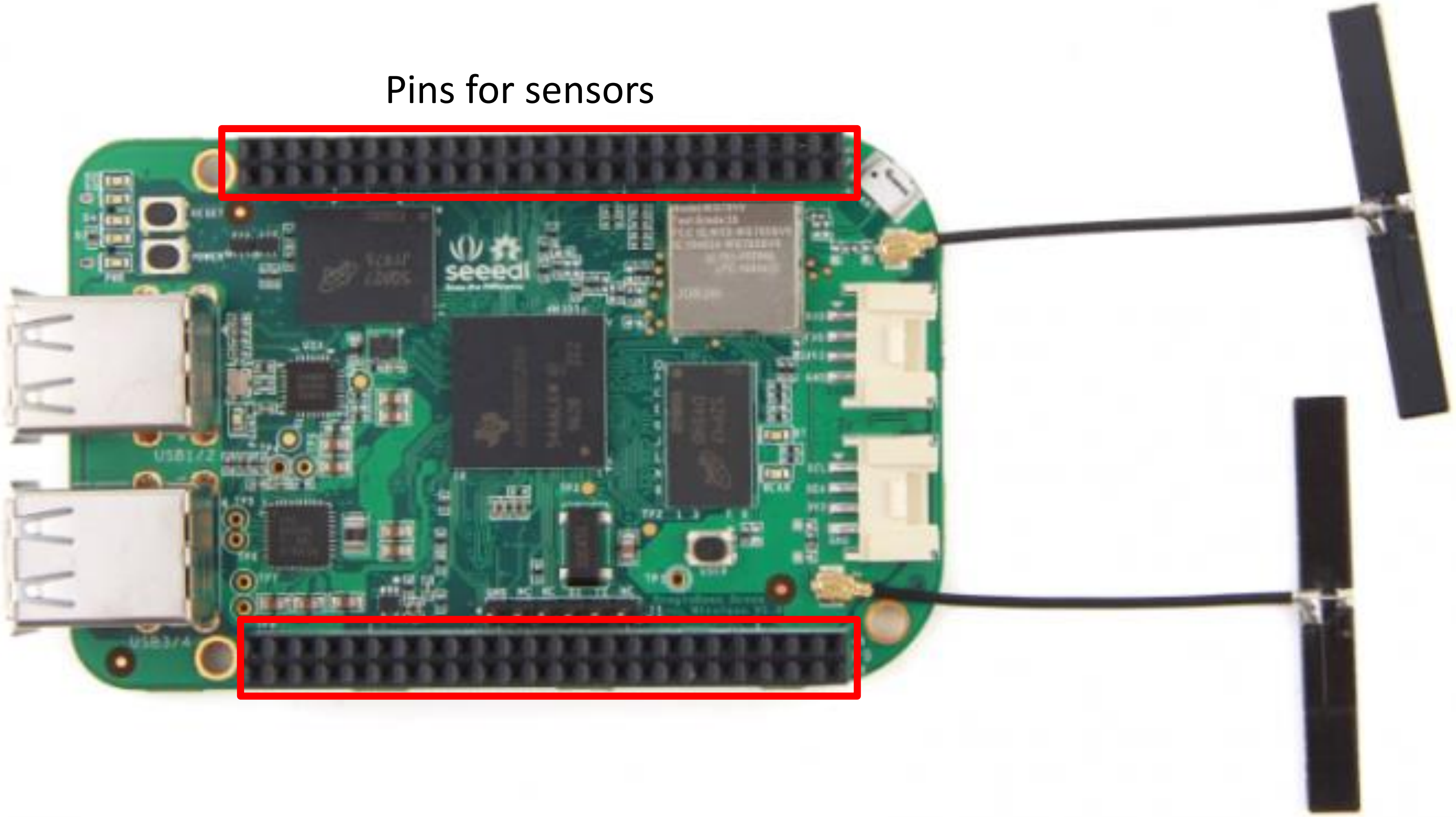
Power button



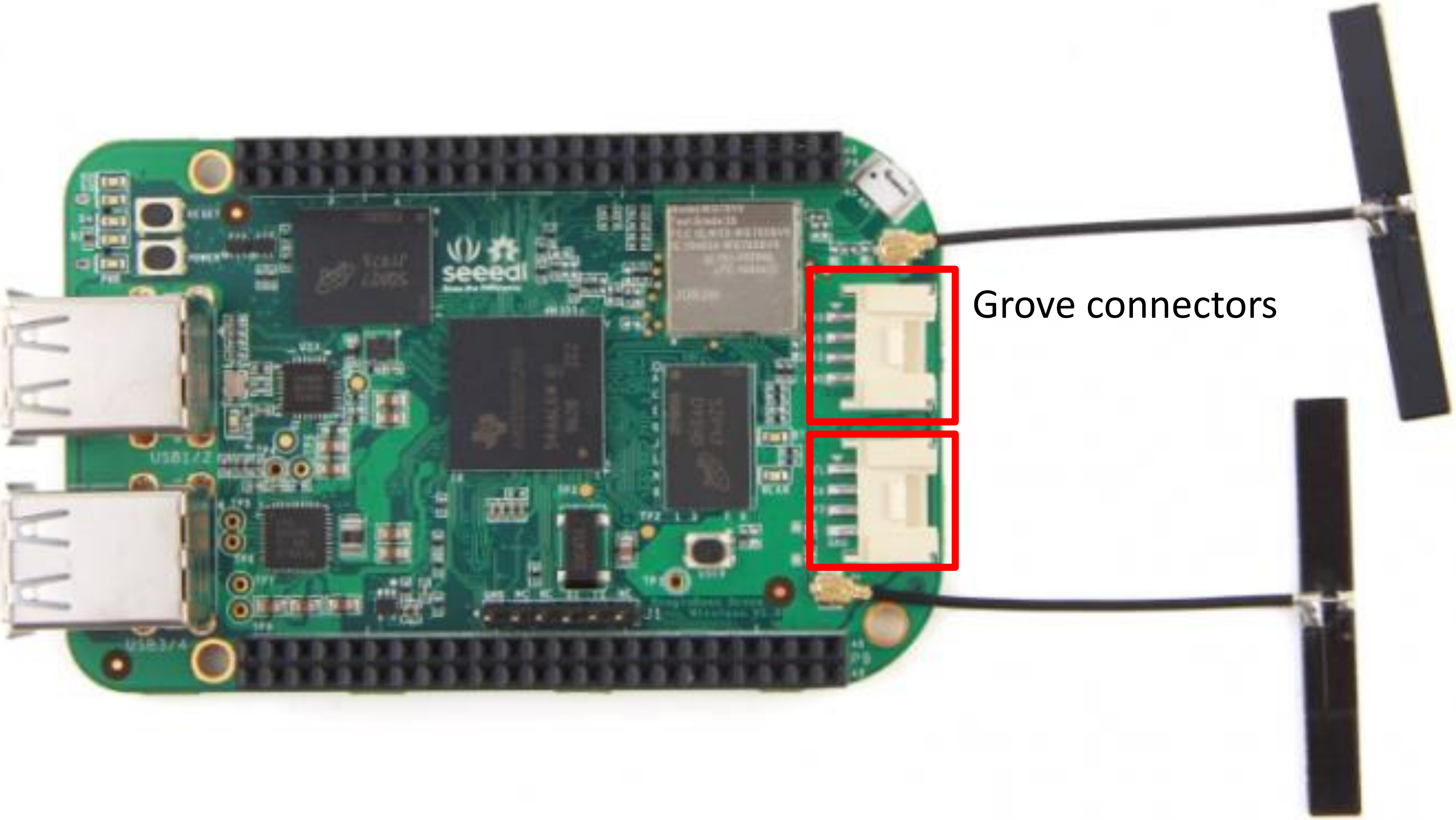
Boot button

What is a beaglebone

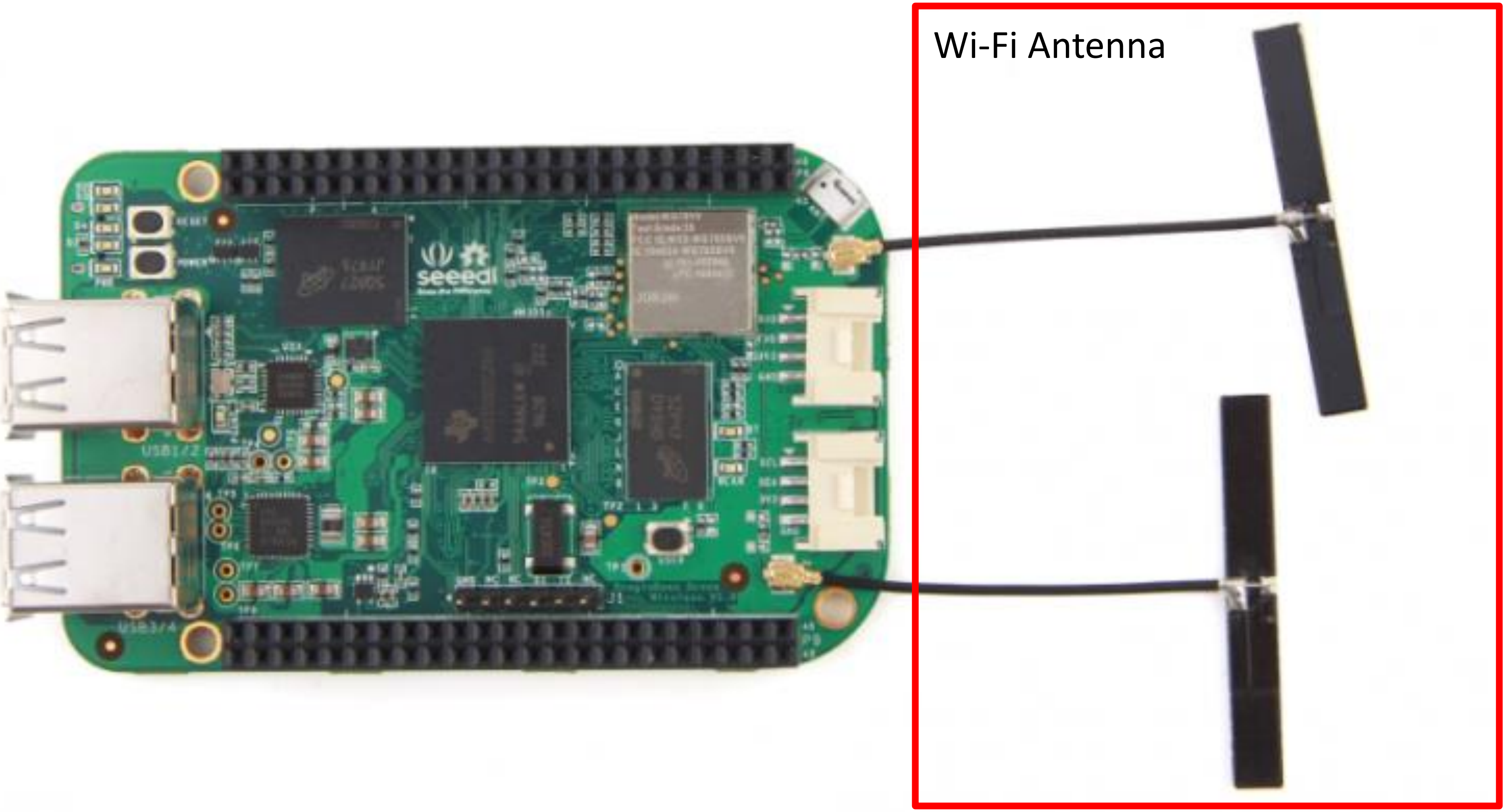
Pins for sensors



What is a beaglebone

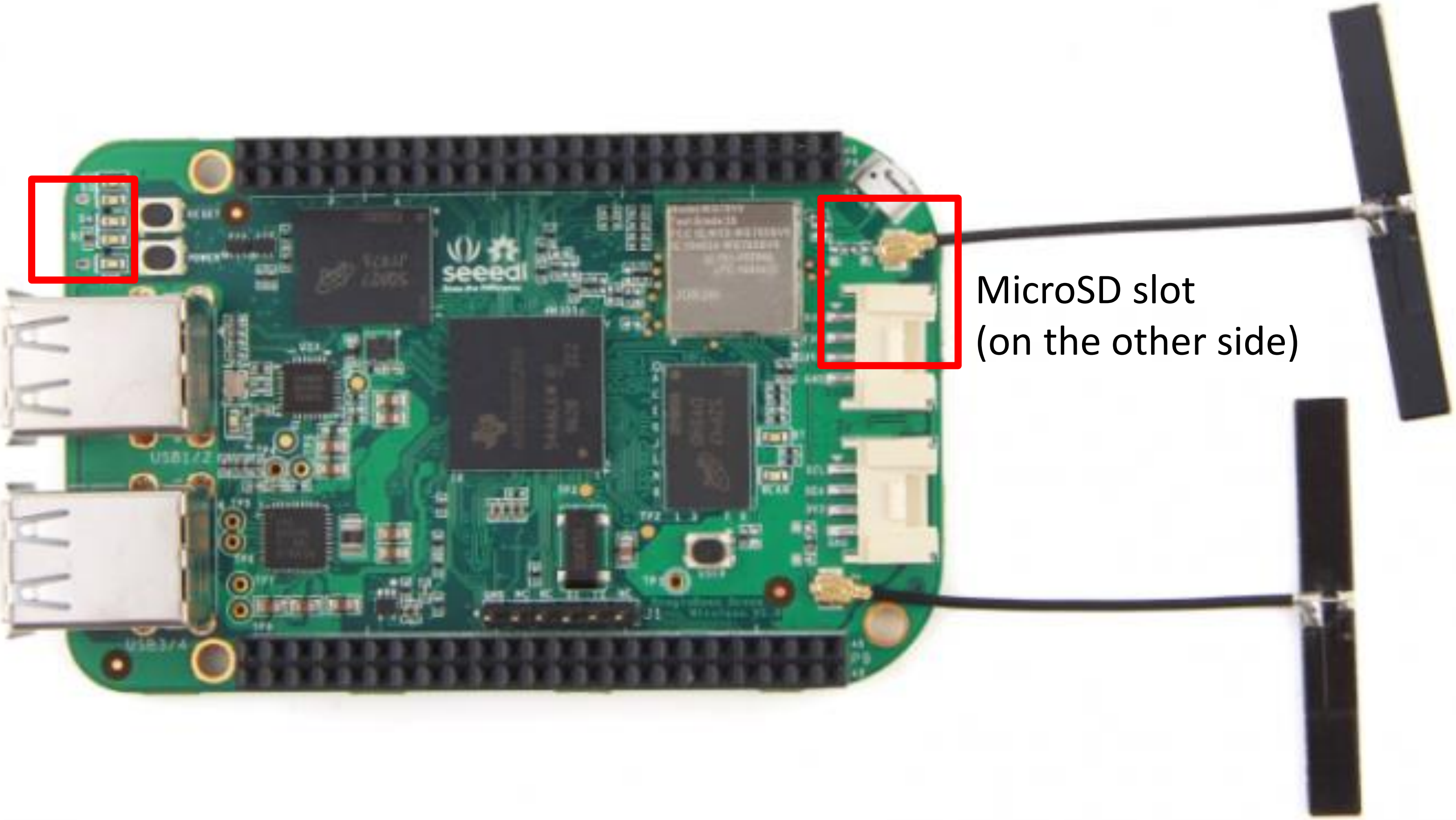


What is a beaglebone



What is a beaglebone

MicroUSB slot
(on the other side)



MicroSD slot
(on the other side)

Boot your beaglebone

- What you need
  - BeagleBone Green Wireless
  - Micro USB – USB cable
  - Your computer
- Use the micro USB cable to connect the beaglebone to your computer
  - Provide power and a development interface
  - Beaglebone runs on a Linux system stored in its flash storage
- Install driver on your computer
  - Instructions available in spec
  - Now you have command line access to your beaglebone

Flashing your beaglebone

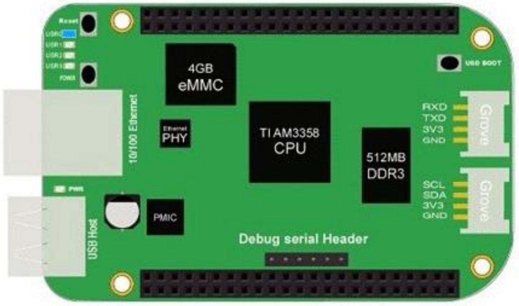
- What you need
  - BeagleBone Green Wireless
  - Micro USB – USB cable
  - MicroSD card
  - Your computer
  - MicroSD card Adapter for your computer

Flashing your beaglebone

- Step 1. Prepare your SD card
  - Download the appropriate image
    - https://debian.beagleboard.org/images/bone-debian-8.7-iot-armhf-2017-03-19-4gb.img.xz
  - Decompress the downloaded image file
    - Image files end in `.img` file extension
  - Install SD card programming utility
    - https://www.balena.io/etcher/
  - Using the SD card programming utility, write the image to the SD card
    - It won’t work if you just copy the img file to the SD card

Flashing your beaglebone

- Step 2. Boot your board off the SD card
  - Power down your beaglebone
  - Insert SD card into your beaglebone
  - Power up your beaglebone with the SD card inserted
    - Your beaglebone should boot from SD card now



Flashing your beaglebone

- Step 3. Flash your beaglebone
  - Edit /boot/uEnv.txt file with sudo privilege
    - Uncomment the last line
    - cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh
  - Reboot your board
  - Observe sequential LED light pattern
    - Could take up to 10 minutes
    - When finish, the LED lights will be off
  - Unplug your board, remove SD card
  - Flashing is complete

Flashing your beaglebone

- Step 4. Reboot without SD card
  - Unplug your board, remove SD card
  - Flashing is complete
  - Reboot your board without SD card inserted
- Step 5. Verify system version
  - Run `lsb_release –a` to check the version of Debian system
  - Should say Release: 8.7

Project 4B – Sensors Input

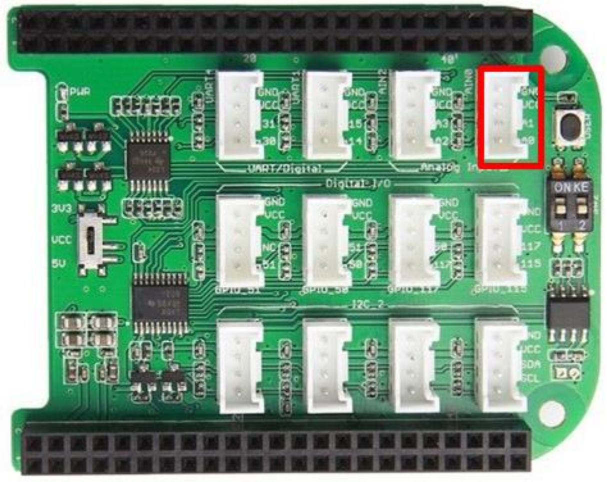
- Create an application that
  - Runs on your beaglebone
  - Read data from external sensors
    - Temperature sensor
    - Button
  - Logs results
- Real world devices are commonly connected. We will explore internet connection in project 4C. For this project, your device will be stand-alone.

Assembling your BeagleBone

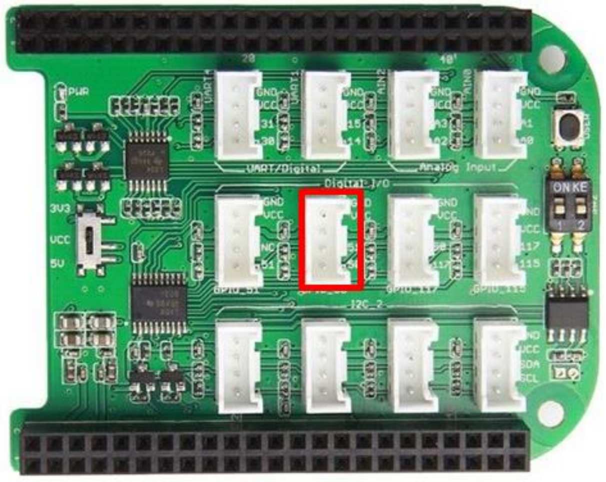
1. Plug the Base Shield in the BeagleBone
2. Plug the temperature sensor to A0/A1
   - Will be **I/O pin #1**
   - Analog device is assumed to provide input
3. Plug the push-button to GPIO 50
   - Will be **I/O pin #60**
   - Need to define if the digital device is used for input or output
4. Turn the voltage on the base cape to 5V
   - This will influence your temperature readings

Assembling your BeagleBone

A0/A1



GPIO 50



Program Breakdown

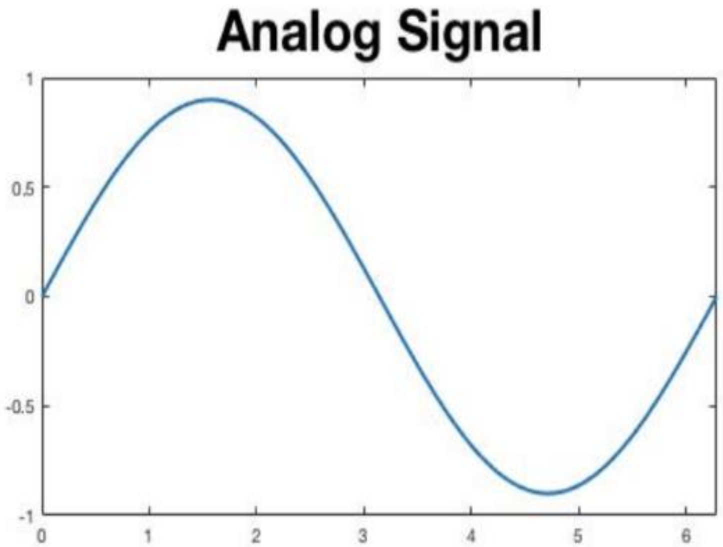
- Goal
  - Read and log temperature periodically
  - From a temperature sensor
  - until a button is pressed
- Arguments
  - period: intervals between consecutive temperature measurements
  - scale: choose to log the temperature in Celsius or Fahrenheit
  - log: choose the log filename
- Command line arguments
  - SCALE, PERIOD
  - START, STOP
  - LOG <TEXT>
  - OFF

Reading from a temperature sensor

- Thermistor is an analog device
  - Thermally sensitive resistor
  - Reading is a resistance
  - You need to convert resistance to temperature
  - Mapping between resistance and temperature is non-linear
- Example code shows temperature in Celsius
  - To convert to Fahrenheit: $F = C \times 1.8 + 32$

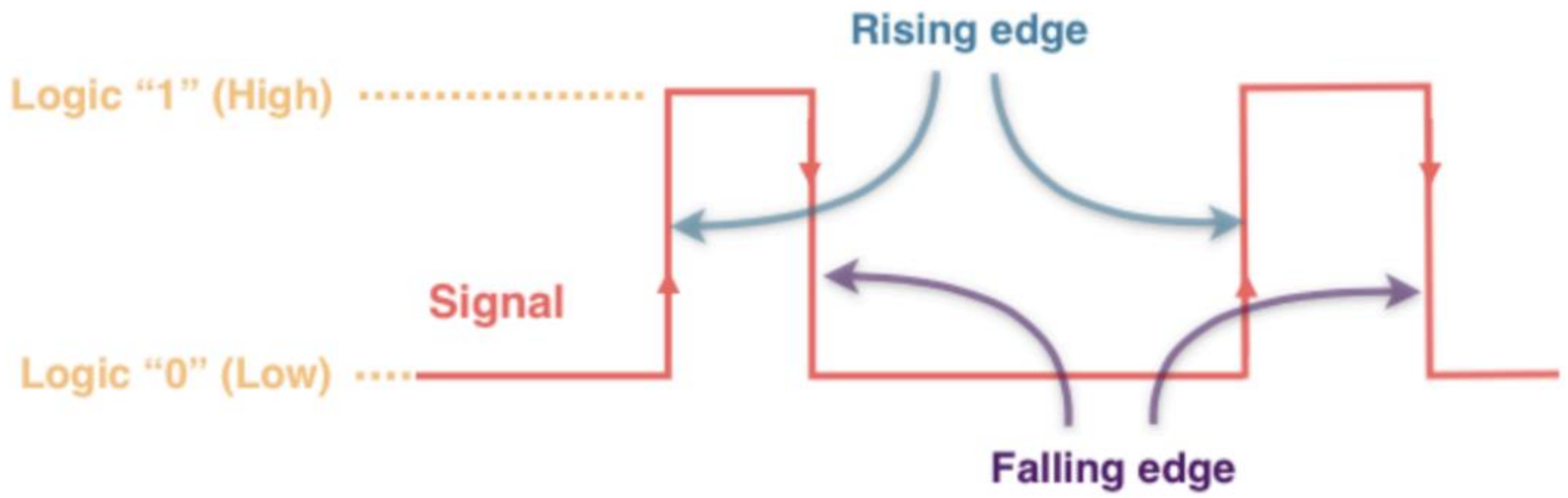
```
int B = 4275; int R = 100000; // B=3975 if v1.0
float R = 1023.0/a-1.0; // a = mraa_aio_read(...);
R = R0*R;

float temperature = 1.0/(log(R/R0)/B+1/298.15)-273.15;
```



Reading from a button

- Buttons have discrete outputs
  - Either logic “1” or logic “0”
  - GPIO



Code for reading from a sensor

- MRAA: I/O library
  1. Include headers
     - MRAA library should be built-in for the beaglebone version of Debian
  2. Allocate sensors variables
  3. Initialize the sensor variables
  4. For GPIO, set the direction of data flow
  5. Read from them
  6. Close the variables

Code for reading from a sensor

- MRAA: I/O library
  1. Include headers
  2. Allocate sensors variables
     - struct mraa_aio_context / mraa_gpio_context
  3. Initialize the sensor variables
  4. For GPIO, set the direction of data flow
  5. Read from them
  6. Close the variables

Code for reading from a sensor

- MRAA: I/O library
    1. Include headers
    2. Allocate sensors variables
    3. Initialize the sensor variables
        - mraa_aio_init / mraa_gpio_init
        - Parameter is the pin#
        - returns mraa_aio_context / mraa_gpio_context
    4. For GPIO, set the direction of data flow
    5. Read from them
    6. Close the variables

Code for reading from a sensor

- MRAA: I/O library
  1. Include headers
  2. Allocate sensors variables
  3. Initialize the sensor variables
  4. For GPIO, set the direction of data flow
     - mraa_gpio_dir
     - On success, returns MRAA_SUCCESS
  5. Read from them
  6. Close the variables

```c
/**
* Gpio Direction options
*/
typedef enum {
   MRAA_GPIO_OUT = 0,      /**< Output. A Mode can also be set */
   MRAA_GPIO_IN = 1,       /**< Input */
   MRAA_GPIO_OUT_HIGH = 2, /**< Output. Init High */
   MRAA_GPIO_OUT_LOW = 3   /**< Output. Init Low */
} mraa_gpio_dir_t;
```

```c
/**
* Set Gpio(s) direction
*
* @param dev The Gpio context
* @param dir The direction of the Gpio(s)
* @return Result of operation
*/
mraa_result_t mraa_gpio_dir(mraa_gpio_context dev, mraa_gpio_dir_t dir);
```

Code for reading from a sensor

- MRAA: I/O library
    1. Include headers
    2. Allocate sensors variables
    3. Initialize the sensor variables
    4. For GPIO, set the direction of data flow
    5. Read from them
        - mraa_aio_read / mraa_gpio_read
        - Parameter is the context
        - Returns an integer
    6. Close the variables

Code for reading from a sensor

- MRAA: I/O library
  1. Include headers
  2. Allocate sensors variables
  3. Initialize the sensor variables
  4. For GPIO, set the direction of data flow
  5. Read from them
  6. Close the variables
     - mraa_aio_close / mraa_gpio_close
     - Parameter is the context object
     - On success, returns MRAA_SUCCESS

Sample program

```c
// flag checked in while loop in main function.
// it is initialized to be 1, or logical true.
// it will be set to 0, or logical false on receipt of SIGINT
sig_atomic_t volatile run_flag = 1;

// this function will set the run_flag to logical false.
void do_when_interrupted(int sig)
{
        // if-statement will verify if the signal is the SIGINT signal before proceeding.
        if (sig == SIGINT)
                run_flag = 0;
}

int main()
{
        // declare buzzer as a mraa_gpio_context variable (GPIO).
        mraa_gpio_context buzzer;
        // initialize MRAA pin 62 (gpio_51) for buzzer.
        buzzer = mraa_gpio_init(62);
        // configure buzzer GPIO interface to be an output pin.
        mraa_gpio_dir(buzzer, MRAA_GPIO_OUT);

        // when SIGINT signal is received, call do_when_interrupted.
        signal(SIGINT, do_when_interrupted);

        // execute if run_flag is logical true (1).
        // break while loop if run_flag is logical false (0).
        while (run_flag) {
                // turn the buzzer on by setting the output to logical true.
                mraa_gpio_write(buzzer, 1);
                sleep(1);
                // turn the buzzer off by setting the output to logical false.
                mraa_gpio_write(buzzer, 0);
                sleep(1);
        }

        mraa_gpio_write(buzzer, 0);
        mraa_gpio_close(buzzer);

        return 0;
}
```

This program emits a beep sound using a buzzer until the user presses Ctrl+C.

Sample program

```c
#include <signal.h>
#include <mraa/gpio.h>

// flag checked in while loop in main function.
// it is initialized to be 1, or logical true.
// it will be set to 0, or logical false on receipt of SIGINT
sig_atomic_t volatile run_flag = 1;

// this function will set the run_flag to logical false.
void do_when_interrupted()
{
        run_flag = 0;
}

int main()
{
        // declare buzzer and button as mraa_gpio_context variables (GPIO).
        mraa_gpio_context buzzer, button;
        // initialize MRAA pin 62 for buzzer and MRAA pin 60 for button.
        buzzer = mraa_gpio_init(62);
        button = mraa_gpio_init(60);

        // configure buzzer GPIO interface to be an output pin.
        mraa_gpio_dir(buzzer, MRAA_GPIO_OUT);
        // configure button GPIO interface to be an input pin.
        mraa_gpio_dir(button, MRAA_GPIO_IN);

        // when the button is pressed, call do_when_interrupted.
        mraa_gpio_isr(button, MRAA_GPIO_EDGE_RISING, &do_when_interrupted, NULL);

        // execute if run_flag is logical true (1).
        // break while loop if run_flag is logical false (0).
        while (run_flag) {
                // turn the buzzer on by setting the output to logical true.
                mraa_gpio_write(buzzer, 1);
                sleep(1);
                // turn the buzzer off by setting the output to logical false.
                mraa_gpio_write(buzzer, 0);
                sleep(1);
        }

        mraa_gpio_write(buzzer, 0);
        mraa_gpio_close(buzzer);
        mraa_gpio_close(button);

        return 0;
}
```

This program emits a beep sound until the user presses the button.

Processing commands

- You need to respond to different commands
  - SCALE, PERIOD
  - START, STOP
  - LOG <TEXT>
  - OFF
- Commands will come from a pipe instead of a keyboard
  - Read may return partial or multiple lines
  - Use a buffer to keep incoming commands and check for valid commands in every iteration
  - Using poll() is appropriate in this context

Processing another input

- Button is another input device, but we cannot poll() on button
  - Why?

Processing another input

- Button is another input device, but we cannot poll() on button
    - There is always a value to be read (either 0 or 1)
- Method 1: check for the button every second
- Method 2: arrange an interrupt when the button is pushed

-DDUMMY

- `-D***` defines a macro to be used by the preprocessor
  - In command line
    - `$ gcc -DDUMMY …`
  - In void main()

```
#ifdef DUMMY
    // include your own dummy mraa implementations that
    // returns a constant when you call mraa_gpio_read or
    // mraa_aio_read and always succeeds for other mraa calls
#else
    // include the real mraa library
#endif
```

- This helps you debug your code without transferring files to beaglebone every time.
- To automate:
  - In your Makefile, run `uname –r`
  - If “`beaglebone`” exists in the resulting output, then compile normally.
  - Otherwise, compile with the `-DDUMMY` flag

FAQ

- Segfaults
  - Likely due to the initialization of your I/O devices. If your sensors aren’t initialized properly, the init function will return NULL
    - Check sensor connection
    - Check pin numbers
    - Run your code from root
    - Flash your card
- We won’t test for tricky edge cases
  - Period=0 or negative
  - Stop and start within a single period
  - Stop and stop, start and start
- On startup, generate first reading before processing input

Questions?

Thank you.