

Add ADC Functionality to RPi with EPI

Raspberry Pi set aside some expansion interfaces to connect with peripheral devices and extend the functionalities, including SPI, I2C, UART and GPIO. However, Raspberry Pi has difficulty in acquiring environmental signals such as temperature, moisture, and air pressure, due to lack of analog input interface. Embedded Pi addressed this issue – Taking the advantage of Embedded Pi Arduino™ interface, we can connect analog sensors to Embedded Pi via Arduino™ Sensor Shield; Raspberry Pi can read ADC values to acquire environmental signals through communication with the Embedded Pi.

Embedded Pi is a Raspberry accessory specially designed by CooCox for Raspberry Pi fans and Arduino™ enthusiasts. As a bridge between Raspberry and Arduino™ shields as well as one between ARM Cortex-M3 and Arduino™ shields, the Embedded Pi integrates and blends the three communities together – Raspberry Pi, Arduino™ and Cortex-M3. For detailed information, please visit <http://www.coocox.org/epi.html>.

This article introduces how to add ADC functionality to the Raspberry Pi using the Embedded Pi.

1. Principle

The MCU STM32F103 on the Embedded Pi is capable of acquiring signals. As the controlled terminal, the Embedded Pi receives real-time command of acquiring ADC values from the Raspberry Pi (refer to [here](#) for the command frame format), decodes the command, starts the Analog-to-Digital Conversion of specified channels, and returns the conversion results to the Raspberry Pi via UART interface. In this case, the Raspberry Pi realized ADC functionality in cooperation with the Embedded Pi.

Using this principle, the Raspberry Pi outputs real-time resistance value of the linear potentiometer and controls the brightness of the LED in the demo below.

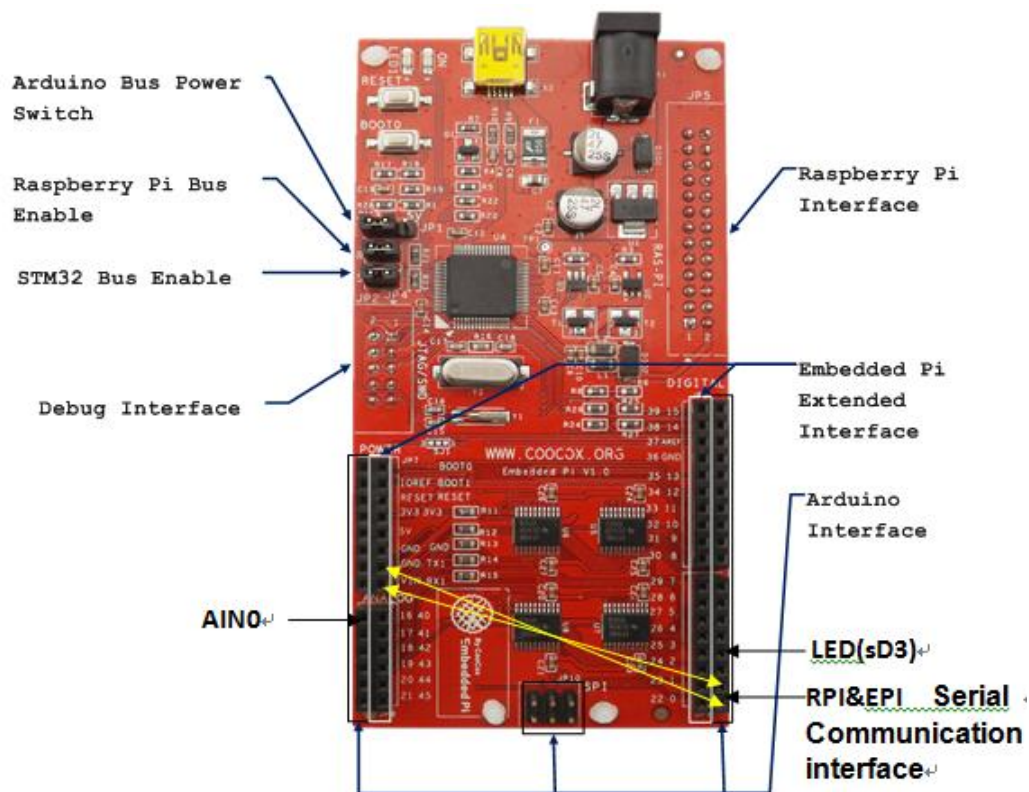
2. Hardware Preparation

2.1 Required Devices

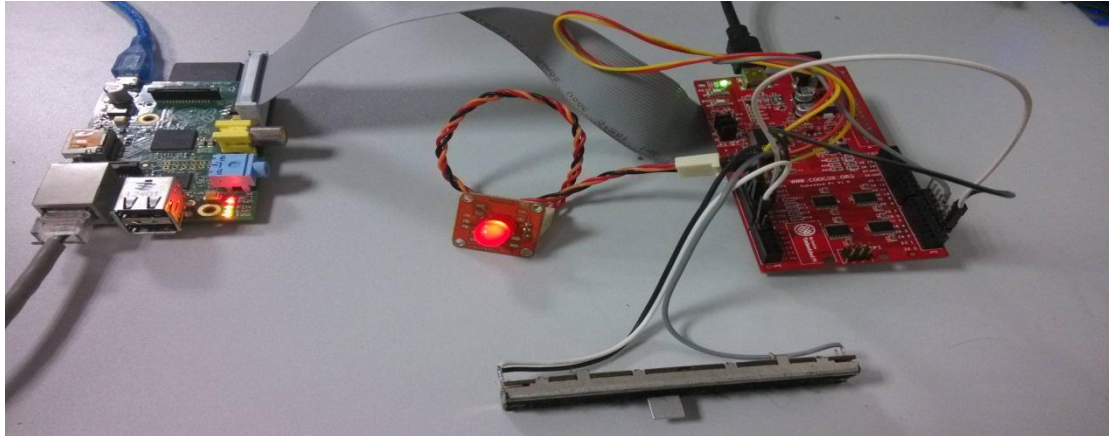
- 1) One piece of Raspberry Pi
- 2) One piece of Embedded Pi
- 3) One linear potentiometer
- 4) One LED
- 5) One 26-pin flat cable
- 6) Several Dupont lines

2.2 Hardware Connection

- 1) Configure JP2 and JP3 to set the Embedded Pi to ST-Adapter mode (For operation mode configuration, refer to [Embedded Pi User Manual](#)).
- 2) Connect the Embedded Pi with the Raspberry Pi using the flat cable (In this case, the UART interface of the Raspberry Pi and the Embedded Pi are connected).
- 3) Connect the linear potentiometer to the Embedded Pi – Connect the analog input to A0 pin of the Embedded Pi, LED to D3 pin of the Embedded Pi.
- 4) Connect the Arduino™ UART interface with the UART interface of Embedded Pi/Raspberry Pi, as the yellow lines in the figure below (The improved version demo will use I2C communication instead, and this step can be skipped then):



So finally your kit should look like the figure below:



2.3 Update the firmware of Embedded Pi

- 1) Since the firmware is to be cloned from GitHub, you need to install Git on the Raspberry Pi first (For the introduction and usage of Git, see http://www.coocox.com/cox/Cox_Github.html):

```
$sudo apt-get install git-core
```

```
$git --version
```

```
git version 1.7.0.4
```

- 2) Clone/Download the firmware to the Embedded Pi after the installation of Git:

```
$git clone git@github.com:machunyu/embeddedpi.git
```

- 3) After the download, enter the directory of 01.AD_Blinky, you will see three folders:

```
$cd demo/demo/01.AD_Blinky
```

```
$ls
```

```
EPI RPI
```

In the EPI folder are programs for the Embedded Pi; In the RPI folder are programs for the Raspberry Pi.

- 4) The UART interface of Raspberry Pi is occupied by the operating system by default for outputting kernel information. The UART0 needs to be configured before being used for communication with the Embedded Pi and downloading the program. Refer to [here](#) for the configuration process.
- 5) Besides the source code of programs, CooCox also provided the executive file ad.bin under directory "EPI/Bin" to run on the Embedded Pi:

```
$cd EPI/Bin
```

```
$ls
```

```
ad.bin
```

6) Download the ad.bin file to Embedded Pi from Raspberry Pi

Refer to [How to program the Embedded Pi with Raspberry Pi](#), and download the ad.bin file to the Embedded Pi using the ISP utility on the Raspberry Pi.

3. The ADC Control Code on Raspberry Pi

To reuse the existing Arduino[™] shield drivers for Linux on Raspberry Pi Debian system, CooCox modified the open source library [arduPi 1.5](#) and added ADC functionality. Click [here](#) to download the customized arduPi library for the Raspberry Pi. You can visit [here](#) to learn the origin and interface introduction of the arduPi library. Here we'd focus on the ADC interface CooCox added based on the Embedded Pi.

3.1 arduPi Library

There are two interface functions for the ADC functionality in the arduPi library.

1) Initialization

```
void ADPi::begin(){  
    Serial.begin(115200);  
}
```

As it is UART interface that is adopted for the communication between the Raspberry Pi and Embedded Pi for now, the initialization of ADC interface is in fact the initialization of the UART interface. Set the baud rate to 115200 to match with the baud rate of Embedded Pi UART interface.

2) Acquire ADC values

```
int ADPi::GetValue(unsigned char channel){  
    .....  
}
```

Below is an example:

```
// file: main.cpp  
//  
//Include arduPi library
```

```
#include "arduPi.h"

void setup()
{
    AD.begin();           // ADC Function Initialization

    pinMode(3, OUTPUT);  // Set sD3 as ouput mode
}

void loop()
{
    int ulADCTmp, ulpwm;

    ulADCTmp = AD.GetValue(0);    // Get the adc digital value of channel 0
    printf("ADC: %d (mV)\n", ulADCTmp*3300/4095);    // Print the adc value

    delay(100);                // Sleep about 100ms
}

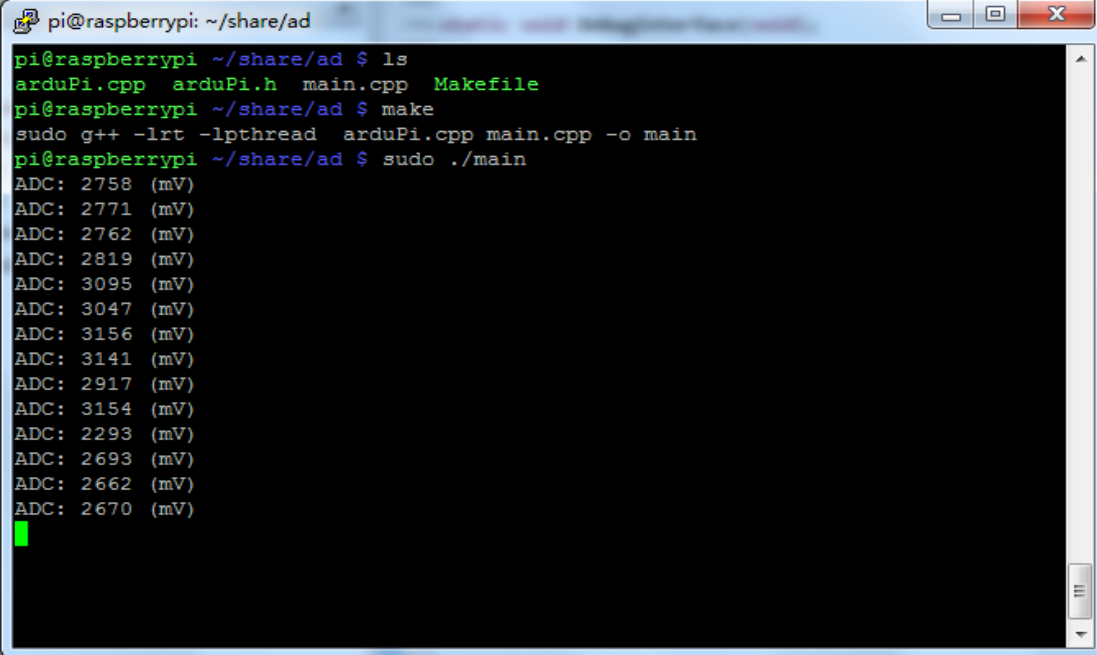
int main (){
    setup();
    while(1){
        loop();
    }
    return (0);
}
```

3.2 Compile and Run

In the arduPi library, the delay() function calls function clock_gettime(), which includes time.h file, thus the gcc compilation parameters should include -lrt. Meanwhile, the two interrupt thread functions attachInterrupt() and detachInterrupt() are in the arduPi, thus the parameter -lpthread should be included too. Compile and run the program:

```
$ g++ -lrt -lpthread main.cpp arduPi.cpp -o main
```

```
$ sudo ./main
```



```
pi@raspberrypi: ~/share/ad
pi@raspberrypi ~/share/ad $ ls
arduPi.cpp arduPi.h main.cpp Makefile
pi@raspberrypi ~/share/ad $ make
sudo g++ -lrt -lpthread arduPi.cpp main.cpp -o main
pi@raspberrypi ~/share/ad $ sudo ./main
ADC: 2758 (mV)
ADC: 2771 (mV)
ADC: 2762 (mV)
ADC: 2819 (mV)
ADC: 3095 (mV)
ADC: 3047 (mV)
ADC: 3156 (mV)
ADC: 3141 (mV)
ADC: 2917 (mV)
ADC: 3154 (mV)
ADC: 2293 (mV)
ADC: 2693 (mV)
ADC: 2662 (mV)
ADC: 2670 (mV)
```

Slide the wiper of the linear potentiometer to different positions, the ADC value output in the Raspberry Pi console changes along, and the LED becomes brighter / darker with greater / smaller ADC value.

4. Follow-up Improvement

With UART communication between the Raspberry Pi and Embedded Pi, it's a bit troublesome to connect extra wires, and inconvenient for other devices to use the UART interface. In the improved version, CooCox will optimize the communication mode and upgrade the firmware of Embedded Pi, using I2C interface which is more flexible for the communication between the Raspberry Pi and Embedded Pi to give full play to Embedded Pi functionalities. Please pay attention to the updates on CooCox [website](#) and [blog](#).

Any problem of using the Embedded Pi, please feel free to [contact us](#).