EECS3215

Analog to Digital Converter

Project

Jianwei Zhong 212115358 Kuanghua Qiao 213172598
4/5/2016

Introduction

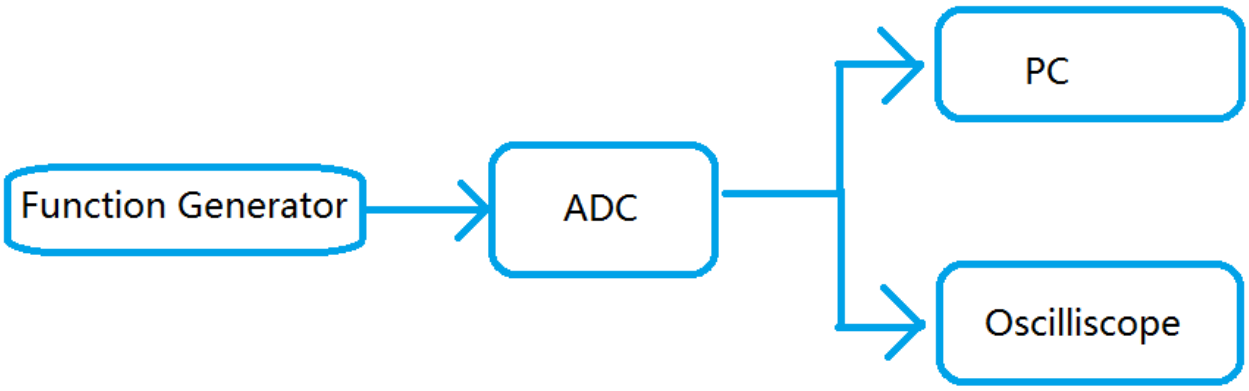
Analog to Digital Converter (ADC) is the one of the most important part in embedded systems. For example, the whole idea of control system in embedded systems is to get back signals after they went through a system so that the whole system can remain stable. In addition, there are several key factors which are very important for designing an ADC, such as sampling rate, resolution, duty cycle and so on. For this project, we design an ADC by using beaglebone board and MATLAB and try to get a maximum sampling rate.

Design

- Theory

There are three ways to design an ADC in beaglebone board. Including using the ADC script in Beaglebone, writing our own ADC function in C and using PRU. In the project, we choose to write ADC function in C instead of using other two. The reasons are that the sampling rate is only 80 Hz by using ADC built-in script and programming PRU is very difficult and complicated.

Circuit Block Diagram



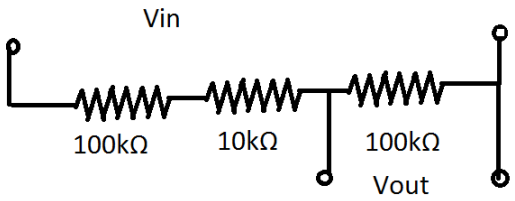
- Equipment
  - Beaglebone Black
  - Oscilloscope
  - 100kΩ Resistor
  - 10kΩ Resistor
  - MATLAB
  - Cloud 9

- Procedure

Firstly, we try to use the built-in ADC script in BB. The sampling frequency is terribly slow---only 83 Hz. Thus we try to use PRU which could be the best way to get maximum sampling frequency. However, as the result, we found out that it is very complicated to write PRU code, for if we acquire maximum sampling rate, it is better to write code in assembly. Consequently, we choose to write our own ADC code in C.

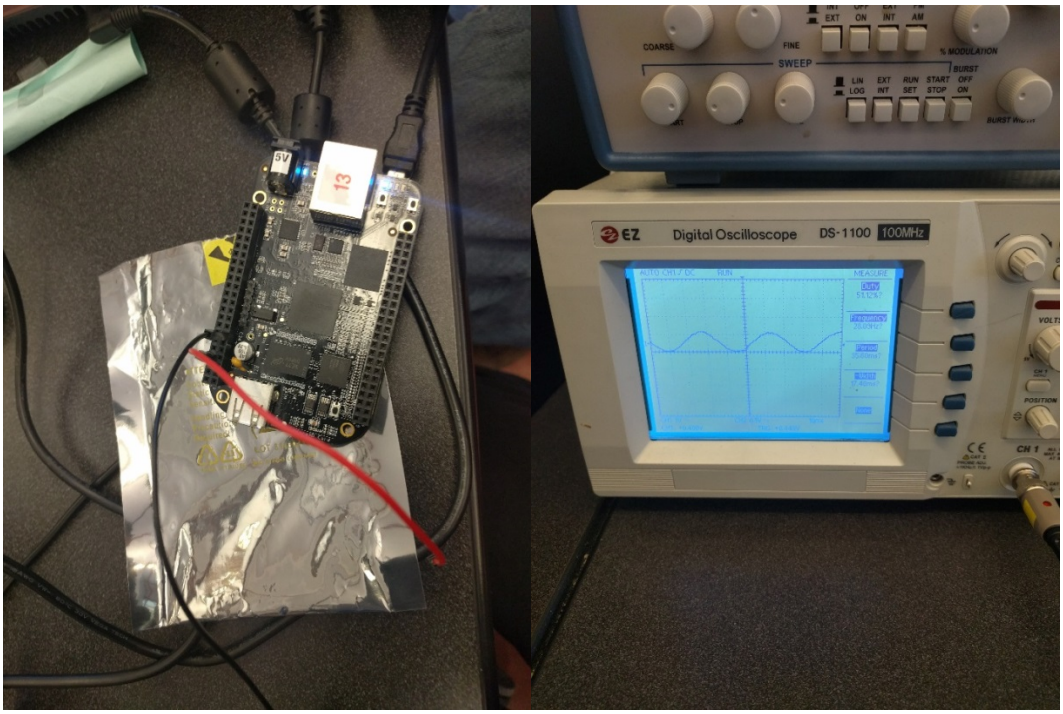
Secondly, during testing our code, we find out that we cannot enable ADC pins on the board in our script. One of the guess is we do not have fully access into the board. But we can still enable ADC pins in the console.

Thirdly, when we test the output signal from the function generator, the minimum voltage of the output function is 2 volt. The maximum voltage of Beaglebone black is 1.8 volt. Thus we decide to use a series of resistors to reduce the input voltage to the Beaglebone board.



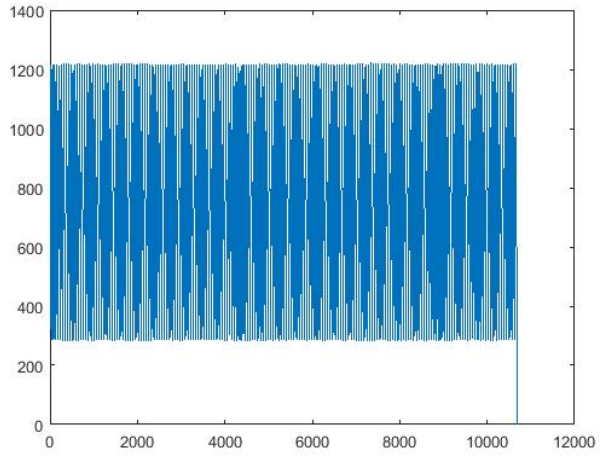
Finally, the script can output a csv file which can be imported into MATLAB.

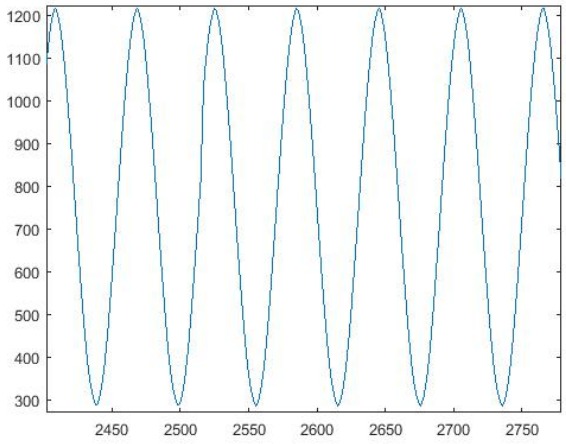
Actual Circuit:



Results

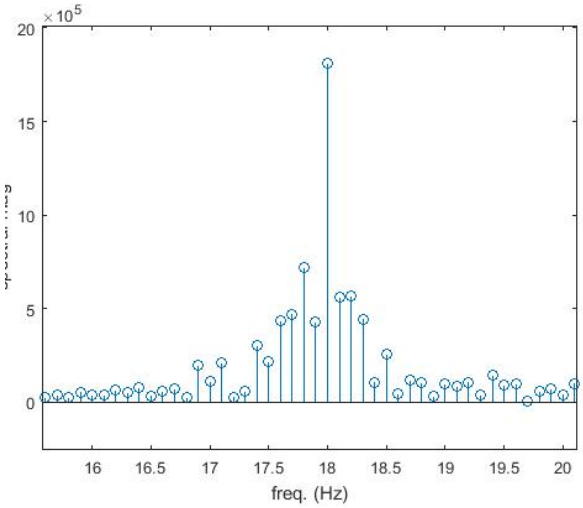
The frequency of input signal is 17Hz.





Sampling signal zooming in

DFT:



Conclusion:

The resolution of ADC in Beaglebone is 12bit. Thus, the output wave form is quite accurate. In addition, we drop the first 0.2 sec after the program begins, because we realized it is very unstable for ADC in the beginning of the time. Secondly, we found out that with increasing recording time, the accuracy of sampling data increases as well. Moreover, because the ADC we design has 1000Hz sampling frequency, the maximum input signal frequency should be 500Hz.

Appendix

```
#include <stdlib.h>
 #include <stdio.h>
 #include <string.h>
 #include <unistd.h>     //close()
 #include <fcntl.h>     //define O_WONLY and O_RDONLY
 #include <time.h>
// #include <dos.h>
 #define Smp_t 10     //This is sampling time

 //Function declarations
 int readADC(unsigned int pin);

 //main program
 int main()
 {    FILE *fp;

      time_t start, end;
      double diff;
      int i,j;
      int adc[1700*Smp_t] = {0};
      //Enable ADC pins within code
      system("echo BB-ADC > /sys/devices/bone_capemgr.*/slots");
      i = j = 0;
      start = time(NULL);
```

```
    //Read ADCs
    while(diff < Smp_t*1.2f){
       adc[i] = readADC(0);
            end = time(NULL);
            diff = (double)(end - start) ;
       if(diff == 0.2)
         j = i;
            i++;
    }
    printf("%d\n",i-j);
    fp = fopen("/var/lib/cloud9/output.csv","w");
    for(;j<=i;j++){
    fprintf(fp,"%d,\n", adc[j]);
    }
    fclose(fp);
    return 0;
}//end main


//Function definitions
int readADC(unsigned int pin)
{
    int fd;          //file pointer
    char val[4];     //holds up to 4 digits for ADC value
    fd = open("/sys/devices/ocp.2/helper.14/AIN0", O_RDONLY);     //open ADC as read only
```

```
    //Will trigger if the ADC is not enabled
    if (fd < 0) {
        perror("ADC - problem opening ADC");
    }//end if
    read(fd, &val, 4);     //read ADC ing val (up to 4 digits 0-1799)
    close(fd);     //close file and stop reading

    return atoi(val);     //returns an integer value (rather than ascii)
}//end read ADC()
```