

Analog to Digital Converters

Lavanya Sajwan

*Worked with Celine Young on the coding

Core 1

$2^{10} = 1024$ different values

$(2^9) + (2^8) + (2^7) + (2^6) + (2^5) + (2^4) + (2^3) + (2^2) + (2) + (2^0) = 1023$

The values range from 1 - 1024

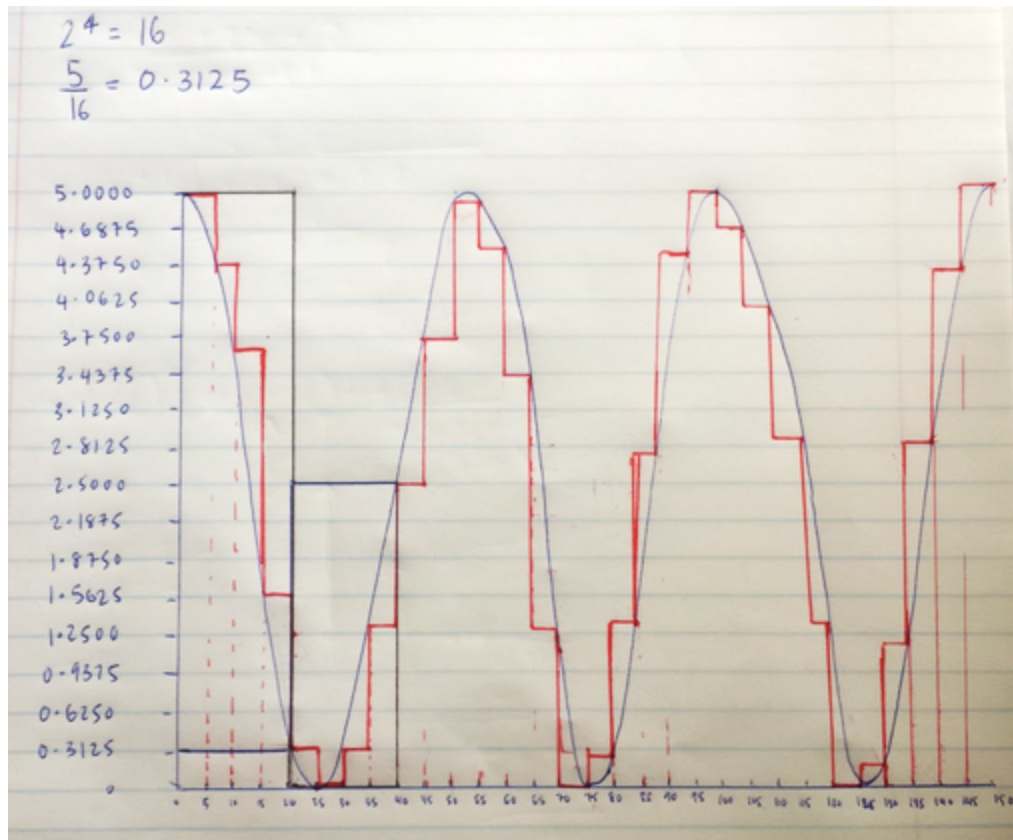
Core 2

$5/1024 = 0.0048828125$ volts

= 0.005 volts

= 5 millivolts

Core 3



(Please ignore the black lines)

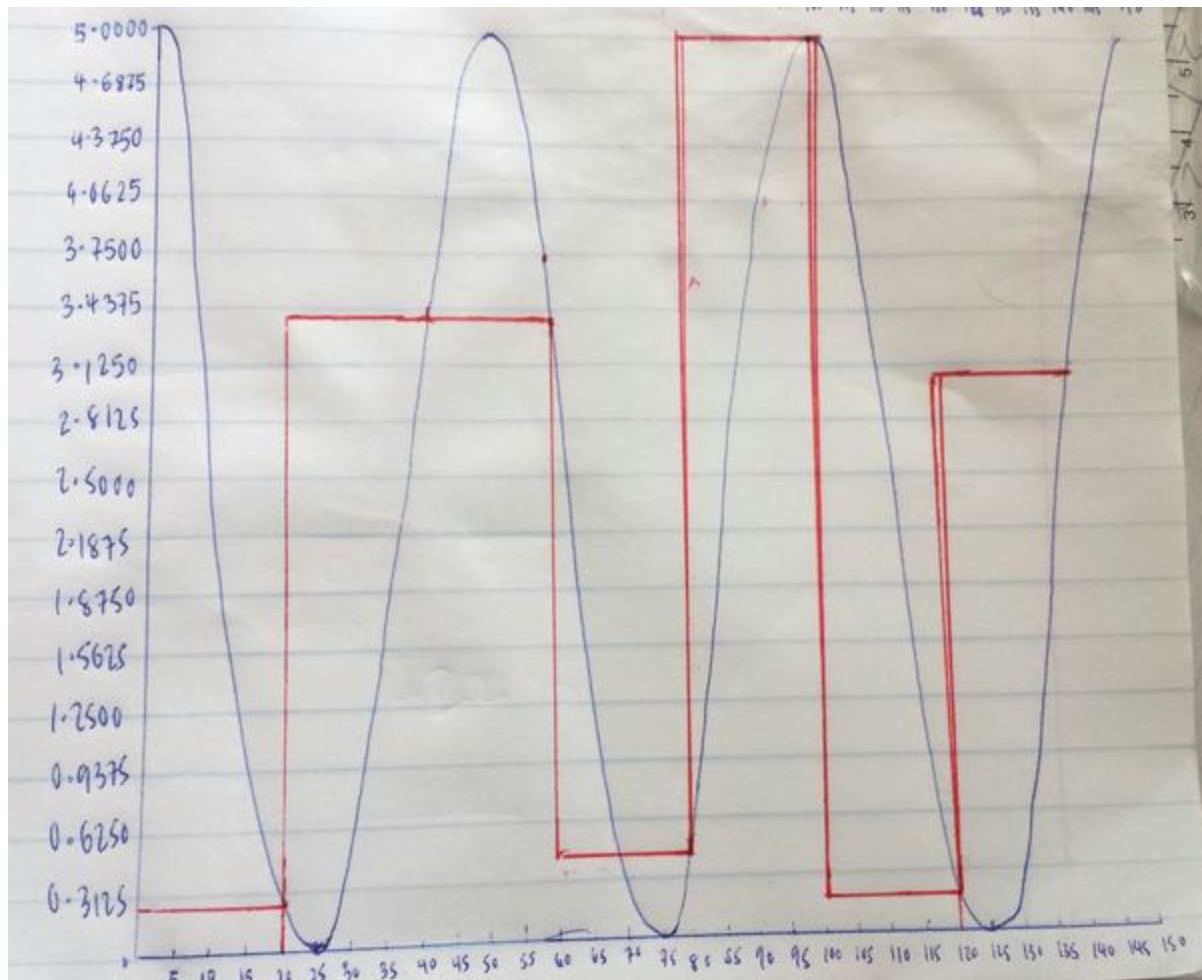
Core 4

$f = 1/T$

$F = 1/0.05$

= 20 Hz

Core 5



Core 6

$$5.25 - 0.25 = 5$$

$$5 / (5.5 - 5.0) = 10$$

10

Core 7

If the signal to noise ratio were less than one the signal would get lost within the noise and will become hard to differentiate.

Core 8

The screenshot shows a C program for Core 8 on the left and its terminal output on the right. The program includes `<stdio.h>` and `<time.h>`, and defines `extern "C"` functions `InitHardware()`, `ReadAnalog(int ch, adc)`, and `Sleep(int sec, int usec)`. The `main` function calls `InitHardware()`, then reads and prints ADC values from channels 0, 2, and 4, with a 1-second sleep between each read. The terminal output, titled "Assignment4", shows the camera initialization process, buffer creation, and the execution of the ADC reads for channels 7, 5, and 3, with a total execution time of 4.806 seconds.

```
1 #include <stdio.h>
2 #include <time.h>
3
4 extern "C" int InitHardware();
5 extern "C" int ReadAnalog(int ch, adc);
6 extern "C" int Sleep(int sec, int usec);
7
8 int main(){
9     InitHardware();
10    int adc_reading;
11    adc_reading = ReadAnalog(0);
12    printf("%d\n", adc_reading);
13    Sleep(1,0);
14    adc_reading = ReadAnalog(2);
15    printf("%d\n", adc_reading);
16    Sleep(1,0);
17    adc_reading = ReadAnalog(4);
18    printf("%d\n", adc_reading);
19    Sleep(1,0);
20
21
22
23
24    return 0;}
25
26
```

```
Assignment4
Init camera output with 320/240
Creating pool with 3 buffers of size 307200
Camera successfully created
ch_adc=7
b2 = 1 b1 = 1 b0 = 1
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=0 rxBuf[1]=0 rxBuf[2]=0
0
ch_adc=5
b2 = 1 b1 = 0 b0 = 1
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=0 rxBuf[1]=0 rxBuf[2]=0
0
ch_adc=3
b2 = 1 b1 = 1 b0 = 0
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=0 rxBuf[1]=0 rxBuf[2]=0
0
Process returned 0 (0x0)   execution time : 4.806 s
Press ENTER to continue.
```

Core 9

The screenshot shows a C program for Core 9 on the left and its terminal output on the right. The program defines `extern "C"` functions `InitHardware()`, `ReadAnalog(int ch, adc)`, and `Sleep(int sec, int usec)`. The `main` function calls `InitHardware()`, then reads and prints ADC values from channels 0, 1, 2, 3, 4, and 5, with a 1-second sleep between each read. The terminal output, titled "101assignment4", shows the camera initialization process, buffer creation, and the execution of the ADC reads for channels 4, 3, and 2, with a total execution time of 7.951 seconds.

```
6 extern "C" int Sleep(int sec, int usec);
7
8 int main(){
9     InitHardware();
10    int adc_reading;
11    int total;
12    total += ReadAnalog(0);
13    printf("%d\n", adc_reading);
14    Sleep(1,0);
15    total += ReadAnalog(1);
16    printf("%d\n", adc_reading);
17    Sleep(1,0);
18    total += ReadAnalog(2);
19    printf("%d\n", adc_reading);
20    Sleep(1,0);
21    total += ReadAnalog(3);
22    printf("%d\n", adc_reading);
23    Sleep(1,0);
24    total += ReadAnalog(4);
25    printf("%d\n", adc_reading);
26    Sleep(1,0);
27    total += ReadAnalog(5);
28    printf("%d\n", adc_reading);
29    printf("%d\n", total/5);
30
31    return 0;}
32
```

```
101assignment4
b2 = 1 b1 = 0 b0 = 1
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=ff rxBuf[1]=ff rxBuf[2]=0
0
ch_adc=4
b2 = 0 b1 = 0 b0 = 1
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=0 rxBuf[1]=0 rxBuf[2]=2
0
ch_adc=3
b2 = 1 b1 = 1 b0 = 0
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=ff rxBuf[1]=ff rxBuf[2]=3
0
ch_adc=2
b2 = 0 b1 = 1 b0 = 0
txBuf[0]=1 txBuf[1]=0 txBuf[2]=0
rxBuf[0]=ff rxBuf[1]=ff rxBuf[2]=2
0
Process returned 0 (0x0)   execution time : 7.951 s
Press ENTER to continue.
```

Completion 1

Humans can only hear between 20Hz and 20kHz. The sampling frequency must be twice that of the maximum frequency that humans can hear so this gives as two samples within one wave and this avoids distortion. This is the Nyquist theorem.

Completion 2

Pin[0] had the sensor connected to it so I expected it to have sensor values picked up from it. However small values from pin[2] and pin[4] were also detected due to the background noise.

Completion 3

The actual values I was able to output were between 3-617. This is different to the bigger range of 1-1024 I calculated in core 1. Because of this the minimum voltage change it would actually detect on an expected range of 0-5V will not be 5 millivolts it would be 8.10 millivolts.

Completion 4

```
1  #include <stdio.h>
2  #include <time.h>
3
4  extern "C" int InitHardware();
5  extern "C" int ReadAnalog (int ch_adc);
6  extern "C" int Sleep(int sec, int usec);
7
8  int main(){
9
10     InitHardware();
11     int adc_reading;
12     int total = 0;
13     int count = 0;
14     int max;
15     int min;
16
17     while(count<5){
18         total+= ReadAnalog(0);
19         printf("%d\n", adc_reading);
20         Sleep(1,0);
21         count++;
22
23         if(adc_reading < min){
24             min = adc_reading;
25         }
26
27         else if(adc_reading > max){
28             max = adc_reading;
29         }
30     }
31
32     printf("%d\n", total/5);
33     printf("%d\n", min);
34     printf("%d\n", max);
35     printf("%d\n", (max-min)/2);
36
37     return 0;
38 }
39
40
```

Challenge 1

Because the maximum frequency that humans can hear is 20kHz, the minimum sample frequency will be double of that which is 40kHz.

Challenge 2

Challenge 3: BONUS MARKS