Name: MANSI UNIYAL
Roll No.: 19EE10039

Experiment 5

Title - Low Pass Filter implementation through Interrupt and pooling implementation on Arduino Hardware

---

Objective:

To Program ATMEGA328p in Arduino Uno to filter the input signal received in the ADC port of the microcontroller and give a filtered output using interrupt and pooling filtering which we can observe in the serial plotter as a virtual output.

Requirements:

1. Arduino Uno Board
2. USB A to USB B cable
3. Arduino IDE

Methodology:

Build the required assembly code to program the ATMEGA328p microcontroller in Arduino. Write the code in Arduino IDE and connect the Arduino Board and observe the signal

Code:

Interrupt filterring:

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

volatile int analogVal;
const byte adc_pin = A0;
int n=33;
int x[33]={0};
//filter coefficients
int h[33] ={0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 10, 14, 16, 20, 24, 26, 26, 26, 24, 20, 16, 14, 10, 6,
4, 2, 0, 0, 0, 0, 0, 0, 0};
//current filter output sample value
float yi;
//filtered output sample value
float yv;
//current input sample value
float xv;

void setup(){
  Serial.begin(9600);
  ADCSRA = 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0;
  ADCSRA |= 1<<ADEN;
```

```
  ADMUX |= 1<<ADLAR;
  ADMUX |= 1<<REFS0;
  ADMUX |= ((adc_pin -14) & 0x07);
}

void loop(){
  ADCSRA |= 1<<ADSC;
  while((ADCSRA & (1<<ADIF)) == 0);
  analogVal = ADCH;
  for(int i=0;i<n-1;i++){
    x[i] = x[i+1];
  }
  x[n-1] = analogVal;

  yi = 0;
  for(int j=0;j<n;j++){                    //convolution
    yi+=(h[j]*x[n-1-j]*1.0)/270;
  }

  yv = 5*((yi*1.0)/255);
  xv = 5*((x[n-1]*1.0)/255);
  Serial.print(xv);
  Serial.print(",");
  Serial.println(yv);
}
```

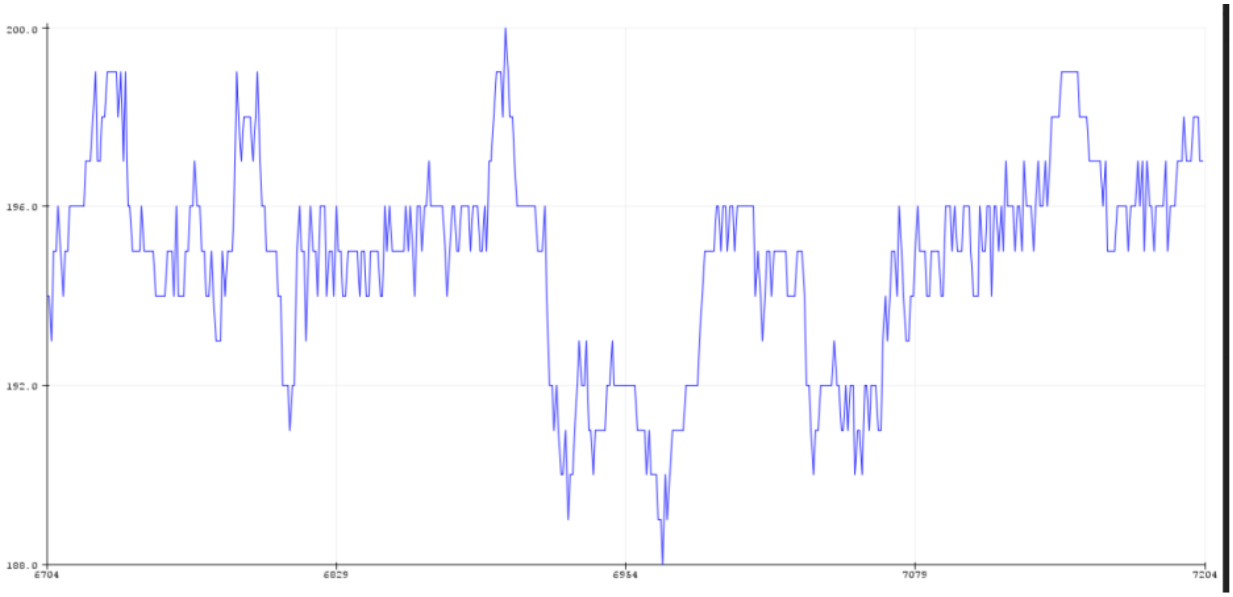
Pooling filtering:

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

volatile int analogVal;
const byte adc_pin = A0;                    //A0 is to be used as input pin
int n=33;
int x[33]={0};
int h[33] ={0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 10, 14, 16, 20, 24, 26, 26, 26, 24, 20, 16, 14, 10, 6,
4, 2, 0, 0, 0, 0, 0, 0, 0};
float yi;           //variable to store the current filter output sample value
float yv;            //variable to store the filtered output sample value in volts
float xv;            //variable to store the current input sample value in volts

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
```

```
  ADCSRA = 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0;        //set clock prescaler to 128
  ADCSRA |= 1<<ADEN;                              //enable ADC
  ADMUX |= 1<<ADLAR;                              //setting output to be left adjusted
  ADMUX |= 1<<REFS0;                              //Vref set to VCC value
  ADMUX |= ((adc_pin -14) & 0x07);                //sending arduino pin value to ADC

}

void loop()
{
  ADCSRA |= 1<<ADSC;                              //start conversion
  while((ADCSRA & (1<<ADIF)) == 0);               //wait for conversion to complete
  analogVal = ADCH;                               //obtain value after conversion
 //shifting the x array to store the latest 33 input samples
  for(int i=0;i<n-1;i++)
  {
    x[i] = x[i+1];
    delay(0.01);
  }
  x[n-1] = analogVal;
  yi = 0;
  //performing convolution based on the formula y(i) = (sum for j = 0 to n-1) x(n-1-j)*h(j)
and scaling appropriately
  for(int j=0;j<n;j++)
  {
    yi+=(h[j]*x[n-1-j]*1.0)/270;
  }
  yv = 5*((yi*1.0)/255);//converting input and output sample values to volts
  xv = 5*((x[n-1]*1.0)/255);
  Serial.print(xv); //printing input and output samples to serial monitor
  Serial.print(",");
  Serial.println(yv);

}
```

Results:

The simulated output as observed in the lab is produced. The Arduino output can be observed in the serial plotter, which shows a filtered output signal of 50Hz frequency with very little noise in the output.



Discussions:

The real-time signal can be produced by swirling the open terminal of the Male-to-Female jumper wire where the generated signal is basically noise. We can also generate real-time signals using a potentiometer.

The interrupt method is beneficial over the polling-based method because in the interrupt method the processor can execute its code without checking for the completion flag in regular intervals. When the filtering is complete, the interrupt signal is sent to the CPU and the CPU executes the necessary ISR while storing the previous PC location at the stack. Thus computational time is saved output pin to D5, +ve terminal to 3V, and -ve pin to ground