Embedded Systems Lab
Expt-5

Pratyush Jaiswal
18EE30021

AIM:

Convert analog input values using ADC and then transmit through UART.

Procedure:

Sinusoidal input wave is taken and given to the internal ADC input pin (ADC0) which converts to digital and then after conversion it is transmitted through TX. Also it is get received to the virtual terminal to see the mapped values from which the matlab plot of the transmitted values are being plotted.

Also the converted value is outputted at PORTB where a DAC is used to convert it to analog & from where a sine wave is generated.

ADC takes a sampling rate of 100 i.e in 1 sec it takes 100 values. It can be changed & according to the frequency according to the Nyquist property.

Date: __/__/____
Page: _____

After the ADC converts the one value, it is transmitted & then other conversion starts.

The output of DAC is in the form of current, so it is being converted into voltage form using an opamp & then being plotted in the oscilloscope.

Here, 10 bits is being taken from ADC & then converted to 8 bits so that it can be outputted through PORTB.

In the virtual terminal, the values are being printed in HEX form.

The result is matching with the input.

In the output oscilloscope output screenshot,

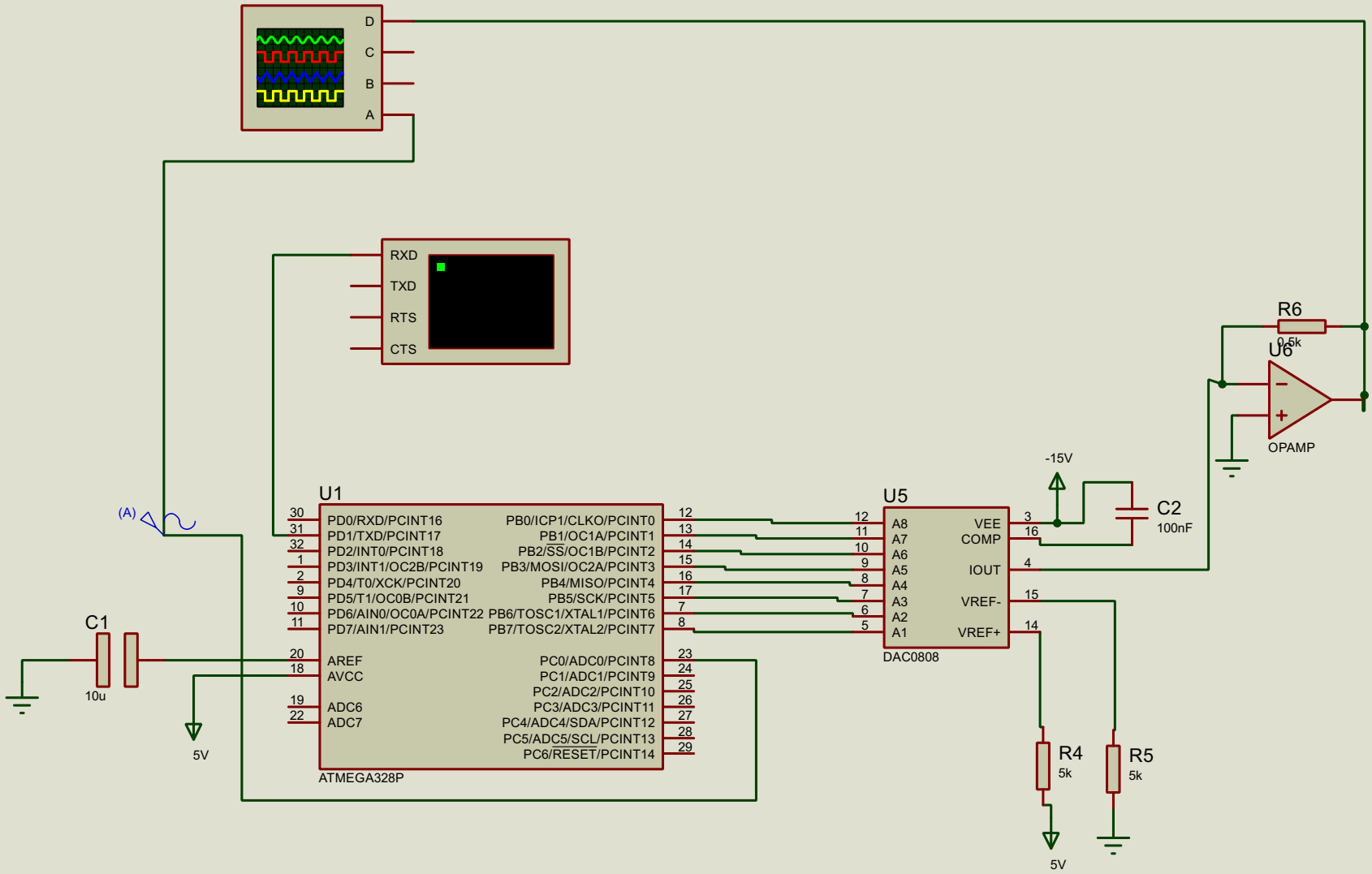
Yellow :- Input Sine Wave

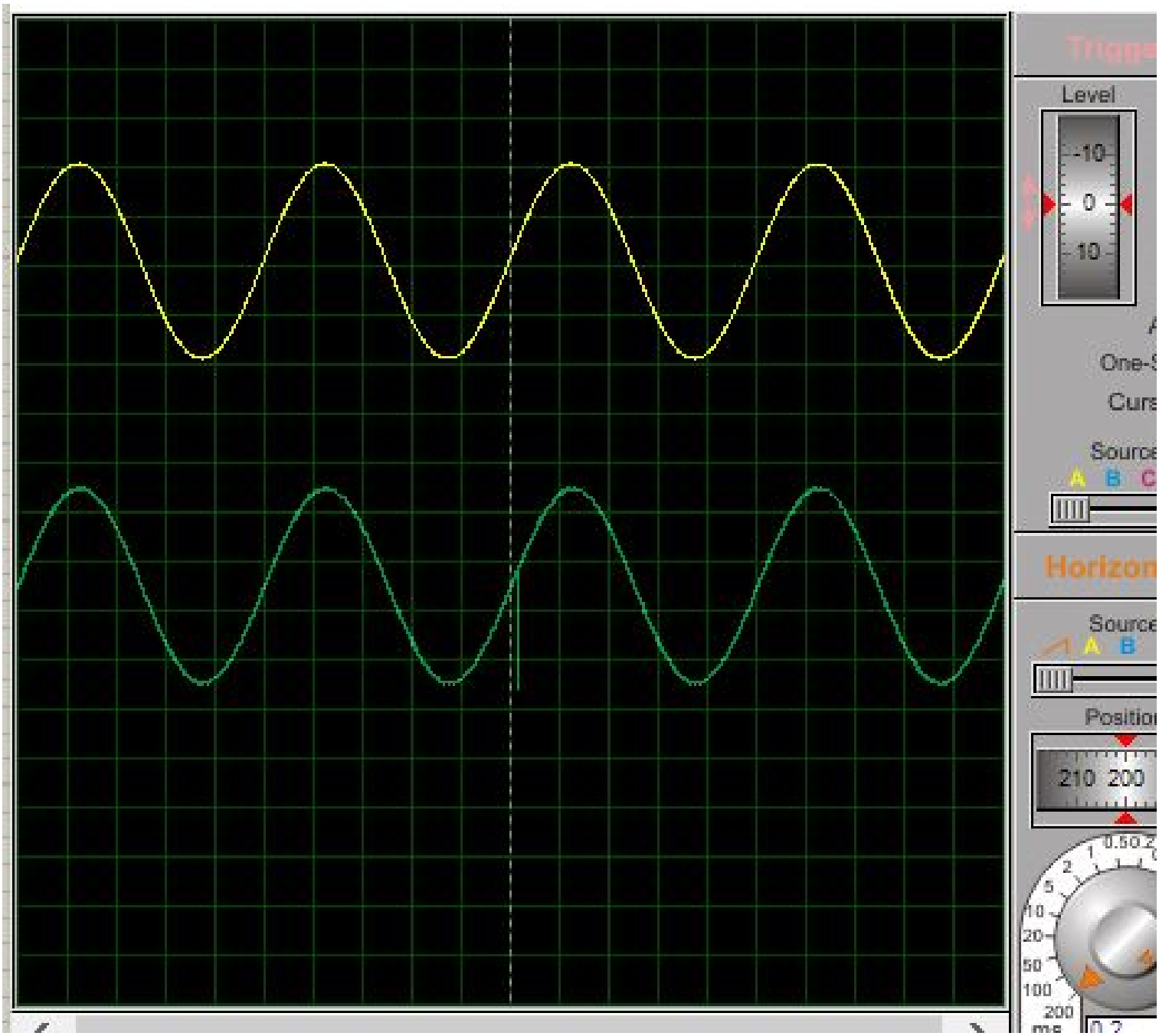
Green :- Output Sine Wave
(from ADC & DAC conversion)

```c
/*
 * GccApplication1.c
 *
 * Created: 04-04-2021 11:46:18
 * Author : Pratyush Jaiswal
 */

#include "avr/interrupt.h"
#include "avr/io.h"
#define F_CPU 1000000UL                     // 1MHz for simulation
#include "util/delay.h"
#include <stdbool.h>
#define BAUD 4800                           // baud rate
#define MYUBRR F_CPU/16/BAUD-1              // baud register for timer
#define SAMPLE_RATE 100                     // number of samples taken in one second

#define TIMER_PRESCALER 64                  // timer prescaler
#define COMPARE ((F_CPU/(TIMER_PRESCALER))/SAMPLE_RATE)-1   // top value for timer


unsigned char lower;                        // for storing the lower 8bits of ADC input
unsigned char upper;                        // for storing the higher 2bits of ADC input

bool done = false;                          // for taking care of the conversion state

void UART_Init(unsigned int ubrr)
{
    UBRR0H = (unsigned char)(ubrr>>8);      //setting baud rate low
    UBRR0L = (unsigned char)ubrr;           // set baud rate upper
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);         // set transmission and receiver bits
    UCSR0C = (1<<USBS0)|(3<<UCSZ00);        // 2 stop bits and 9 bit character size
}

void UART_Transmit(unsigned char data)
{
    while (!(UCSR0A & (1<<UDRE0)));         // wait for empty buffer and transmit
    UDR0 = data;
}
ISR(ADC_vect){                              // one data point conversion done
    lower = ADCL;                           // variable where lower 8 bits are stored
    upper = ADCH;                           // variable where upper 8 bits are stored
    upper &= 0x03;                          // making sure that the other 6
```

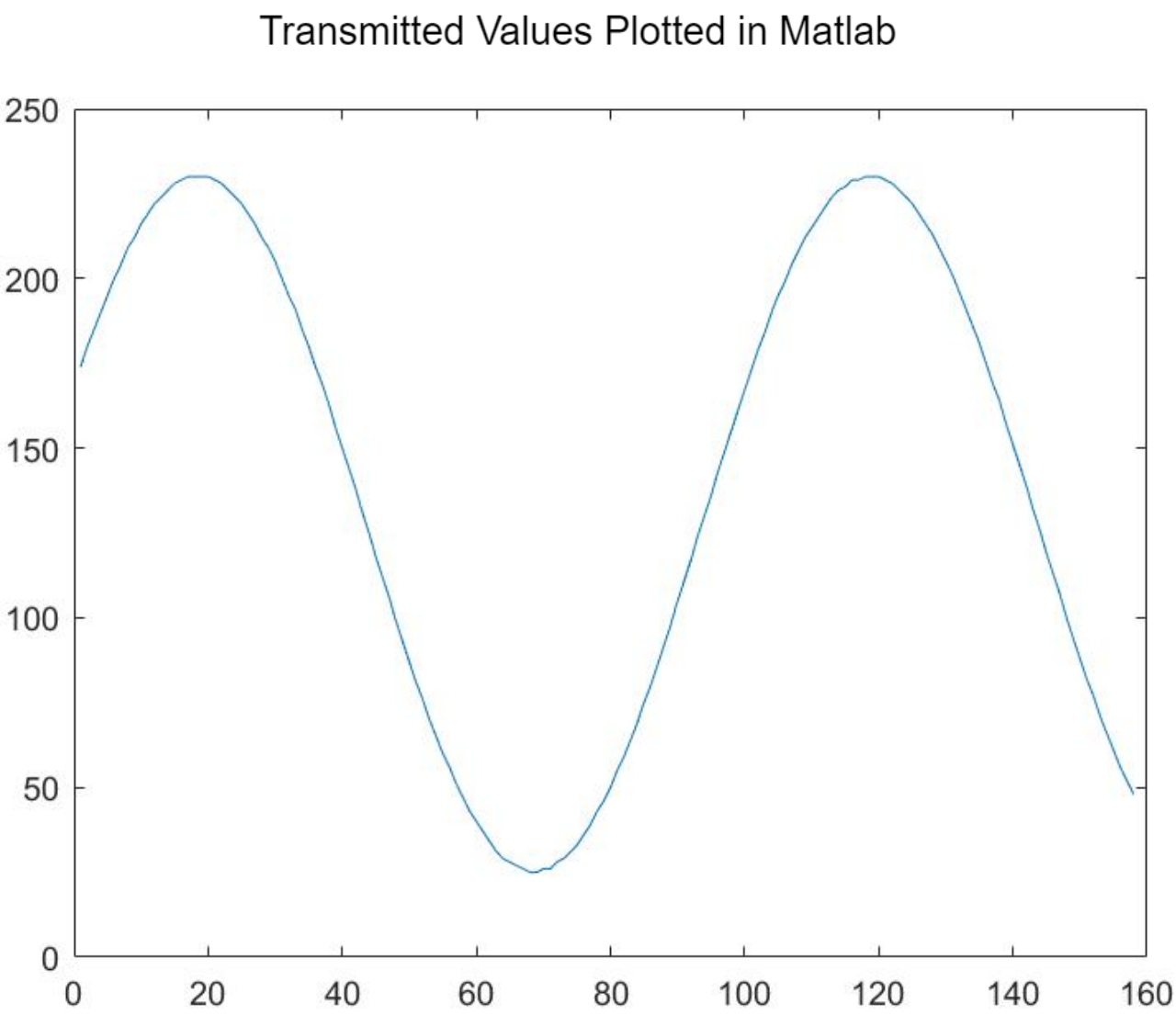
```
        bits are maintained zero
43      done = true;                            // conversion is completed and
          it is ready to be transmitted
44  }
45  EMPTY_INTERRUPT (TIMER1_COMPB_vect);        // timer interrupt when the
    counter overflows, just pass
46
47  int main(void)
48  {
49      UART_Init(MYUBRR);                      // initialization of uart
50      DDRC &= 0xFE;
51      DDRB = 0xFF;
52      TCCR1A = 0;
53      TCCR1B = 0;
54      TCNT1 = 0;
55      TCCR1B = (1<<CS11)| (1<<CS10) | (1<<WGM12);     // CTC, prescaler of 8
56      TIMSK1 = (1<<OCIE1B);                           // interrupt enable
57      OCR1A = COMPARE;                                // top values for timer
58      OCR1B = COMPARE;
59
60      ADCSRA =  (1<<ADEN) |(1<<ADIE) | (1<<ADIF);     // turn ADC on, want
          interrupt on completion
61      ADCSRA |= (1 << ADPS1) | (1 << ADPS0);          // 8 prescaler
62      ADMUX = (1<<REFS0) | (0 & 7);                   // select ADC0 for
          conversion (total 6 ADCs are present)
63      ADCSRB = (1<<ADTS0) | (1<<ADTS2);               // Timer/Counter1
          Compare Match B
64      ADCSRA |= (1<<ADATE);                           // turn on automatic
          triggering
65      DIDR0 |= 0X01;                                  // Disabling Digital
          Input Buffer corresponding to ADC0 to save power
66                                                      // referred to
                      documentation
67      sei();                                          // switching interrupt
          on
68      while(1)
69      {
70          //temp=(upper<<6) + (lower>>2);
71          PORTB =(upper<<6) + (lower>>2);
72          if(done)
73          {
74              cli();                                  // clear global
                  interrupt flag to prevent
75                                                      // any interrupt
                        calling as transmission being done
76              UART_Transmit((upper<<6) + (lower>>2)); // transmit the top 8
                  bits from 10 bits output
77              done = false;                           // transmission over,
                  ready for next value conversion
78              sei();                                  // again switching
                  interrupt on
79          }
80      }
```

```
81      return 0;
82  }
```


D
C
B
A
RXD
TXD
RTS
CTS
R6
0.5k
U6
OPAMP
-15V
C2
100nF
U5
A8
A7
A6
A5
A4
A3
A2
A1
VEE
COMP
IOUT
VREF-
VREF+
DAC0808
U1
PD0/RXD/PCINT16
PD1/TXD/PCINT17
PD2/INT0/PCINT18
PD3/INT1/OC2B/PCINT19
PD4/T0/XCK/PCINT20
PD5/T1/OC0B/PCINT21
PD6/AIN0/OC0A/PCINT22
PD7/AIN1/PCINT23
AREF
AVCC
ADC6
ADC7
PB0/ICP1/CLKO/PCINT0
PB1/OC1A/PCINT1
PB2/SS/OC1B/PCINT2
PB3/MOSI/OC2A/PCINT3
PB4/MISO/PCINT4
PB5/SCK/PCINT5
PB6/TOSC1/XTAL1/PCINT6
PB7/TOSC2/XTAL2/PCINT7
PC0/ADC0/PCINT8
PC1/ADC1/PCINT9
PC2/ADC2/PCINT10
PC3/ADC3/PCINT11
PC4/ADC4/SDA/PCINT12
PC5/ADC5/SCL/PCINT13
PC6/RESET/PCINT14
ATMEGA328P
C1
10u
5V
(A)
R4
5k
R5
5k
5V

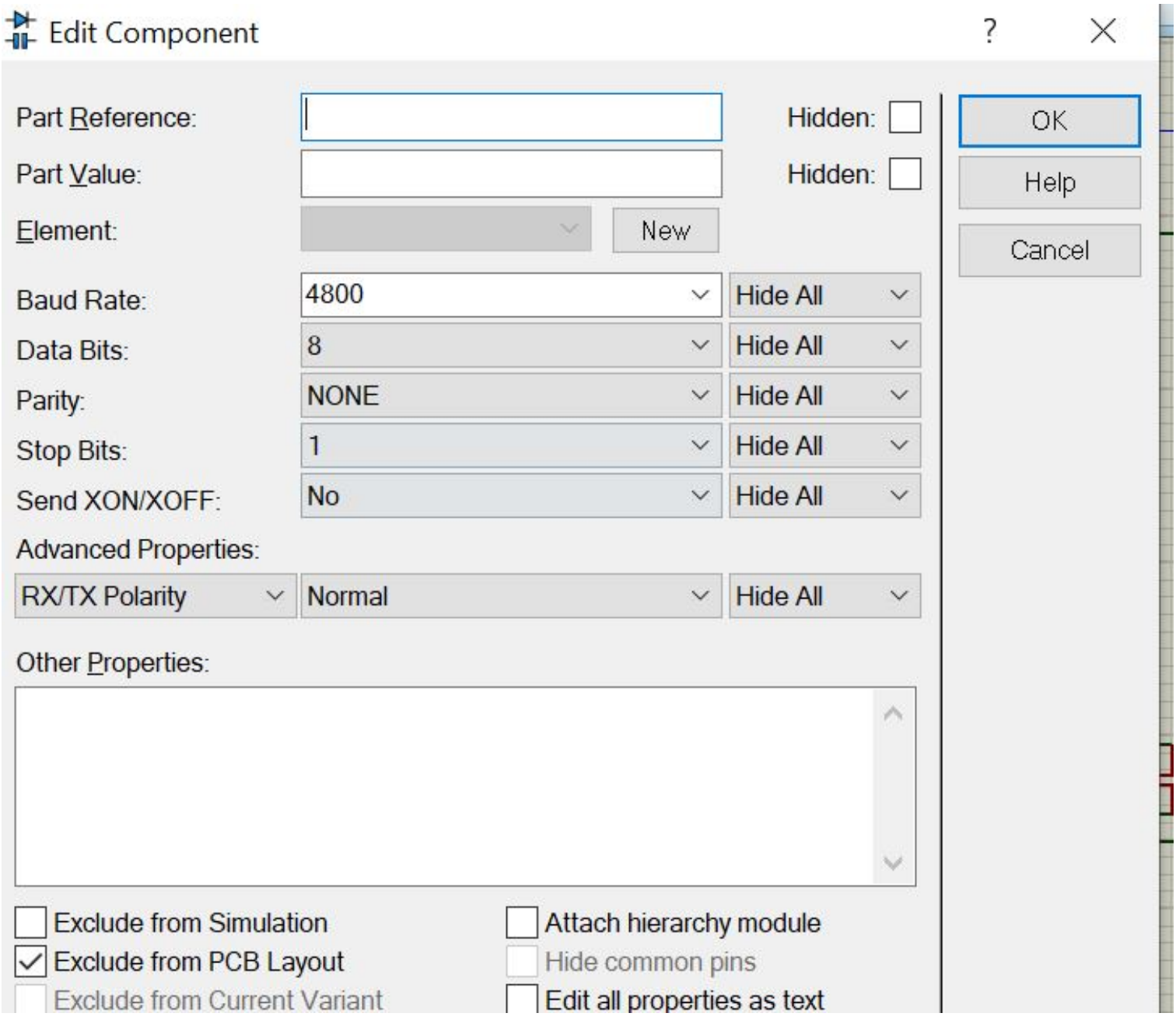

Level
10
0
10
Source
A B C
Source
A B
Position
210 200
0.5
1
2
5
10
20
50
100
200

Virtual Terminal



230 corresponds to 4.5V which is my input sine amp(with offset), so the y-axis can be mapped to the corresponding value.

Virtual Terminal Settings



Date: __/__/____
Page: _____

DISCUSSION:

1. It can be seen that there are some spikes coming in the output in osciloscope. It can be because of the 10 bit manipulation to fit in 8 bit which might lead to some out of track values or error.

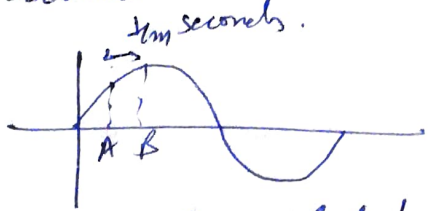
2. The DAC is used for checking whether the ADC converted values are matching with the input signal or not. The result is observed that it is matching with some spikes. (under error permitted %).

3. Sampling frequency should be according to the input wave frequency following the Nyquist theorem.

→ Ideally ADC conversion should be started when the timer overflows, of which the top value is decided by the sampling frequency, but after each conversion we are passing the ADC and transmitting that value too. so the transmission overhead has to be added between two conversion.

So let initially theoretical, we have m samples per second.

So time between two samples taken should be $\frac{1}{m}$ seconds.



Now before B is selected for conversion A has to be transmitted, let the time taken to complete transmission be t seconds. So t seconds have to also added between two samples.

∴ Actual Sampling Rate would be around

$$= \frac{1}{\frac{1}{m} + t} \text{ samples/sec.}$$

The ADC delay if very high as compared to between sample time will also have adverse effect on the output. So while taking the sampling rate we should increase the Nyquist rate from 2 to 2.3 or 2.5 something.

Aliasing proof

Theoretical Sampling rate = 100 samples/s.

Sine frequency = 93Hz.

This clearly violates the Nyquist sampling law, therefore aliasing happens i.e, the output frequency will be less than input.

From DSO,

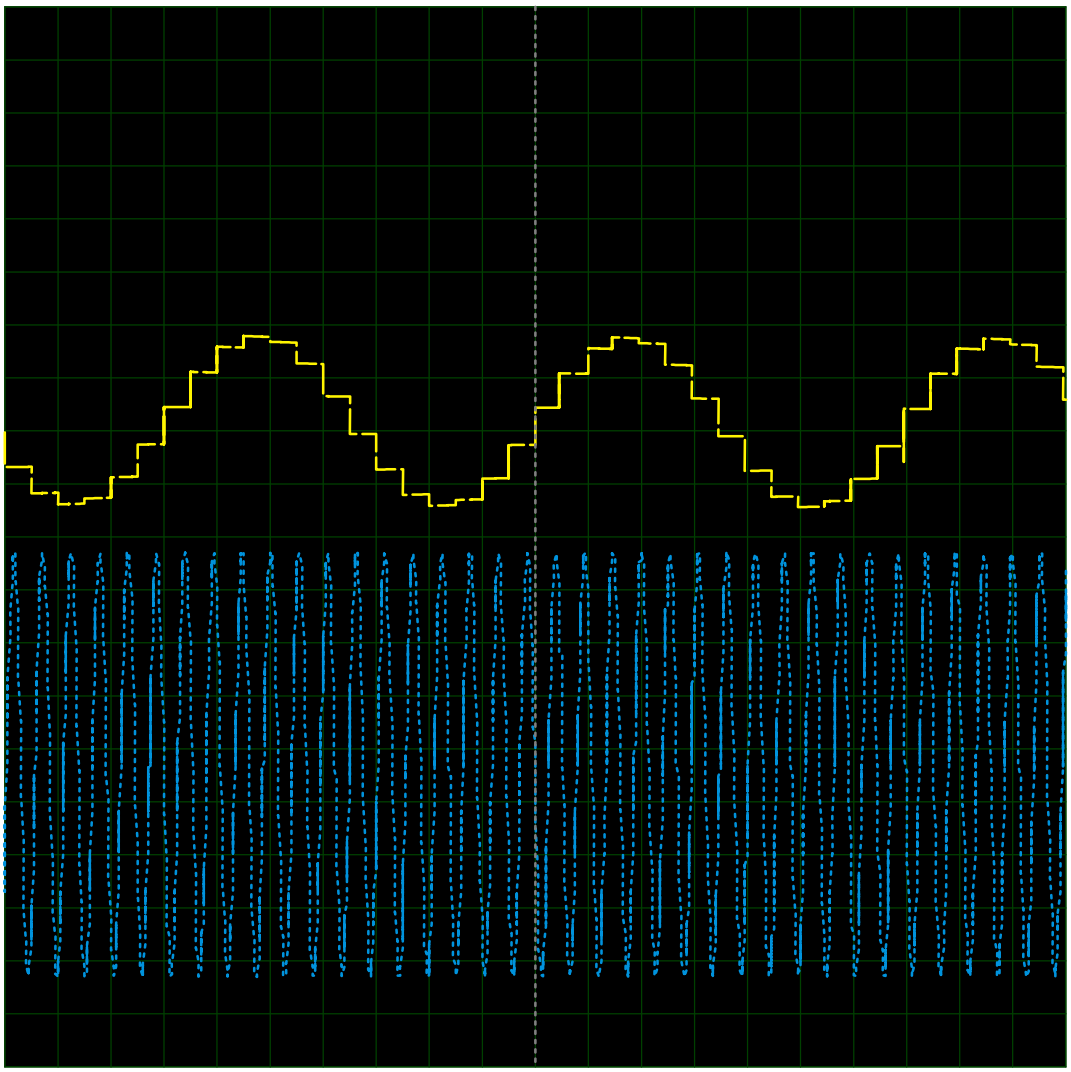
Number of blocks between 2 same value = 7 blocks and 1 block = 20 ms.

$\therefore$ Aliasing frequency $= \frac{1}{7 \times 20 \times 10^{-3}} \approx 7$ Hz.

which is significantly lower than the input sine frequency.

(Here yellow channel is DAC output and blue is input)

DSO Output



	Channel A	Channel B	Channel C	Channel D
V/Div	500.00 mV	500.00 mV	5.00 V	5.00 V
Offset	2.00 V	-4.30 V	-20.00 V	-60.00 V
Invert	Normal	Normal	Normal	Normal
Coupling	AC	AC	Off	Off

	Horizontal
Source	Trace
Position	200.00 mS
S/Div	20.00 mS

	Trigger
Source	Channel A
Level	0.00 V
Coupling	DC
Edge	Rising
Mode	Snapshot

Aliasing Frequency Calculator

This aliasing frequency calculator determines the perceived (reconstructed) frequency f_p of any signal frequency f, which is sampled at any sampling frequency f_s. The calculator also determines the Nyquist frequency for the given sampling frequency. Note that no low-pass or anti-aliasing filter is used to filter higher frequencies, which do not satisfy the Nyquist sampling criterion.

Example: Many musical instruments can easily produce harmonics up to 80 kHz and even more. Many instrumental microphones and sound equipment also have extended frequency response up to 80 kHz. At the same time, the common sampling frequency for digital audio is only 44.1 kHz. Calculate the Nyquist frequency and the perceived frequency for a 39 kHz sine signal if the sampling frequency is 44.1 kHz.

Input

Signal Frequency

f 93 hertz (Hz)

Sampling Rate

f_s 100 hertz (Hz)

Calculate Reset Share

Output

Nyquist Frequency

f_n 50 Hz

Perceived Frequency

f_p 7 Hz

Aliasing occurs because the signal frequency exceeds the Nyquist frequency. The perceived frequency is aliased back to a frequency between 0 and the Nyquist frequency.