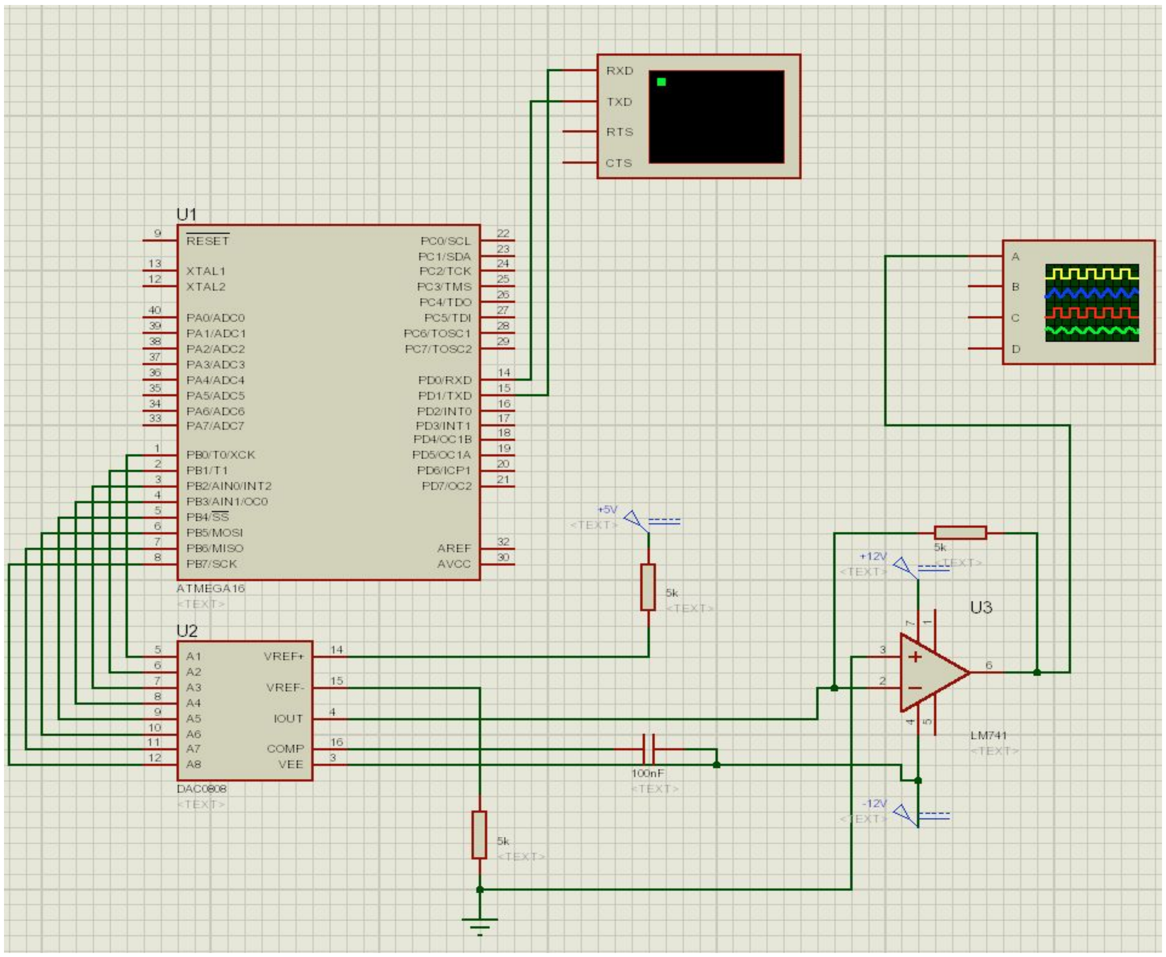
CI - Lap 004 - Wave Generator

Lab Target:

Interfacing 8-bit digital-to-analog with µC to generate different Analog waveforms like sawtooth, square wave, and sine wave.

Theory:

To generate different Analog waveforms using AVR microcontroller it is required to interface a DAC that will convert the Digital inputs given by microcontroller into corresponding Analog outputs and thus it generates different analog waveforms. The DAC output is current equivalent of digital input. So to convert it in to voltage a current to voltage converter is required. This current to voltage converter is build using Op-Amp LM741, and must be connected as follows:



Required Components:

#	Component	Quantity
1	Atmega16L	1
2	8-bit DAC0808	1
3	Operational Amplifier - LM741	1
4	Resistor(s): 5k	3
1	Capacitor(s): 100nF	1

Waveform Generation Approach:

To understand circuit operation we need to understand how microcontroller gives different data to DAC to generate require waveform. Each waveform will be implemented as a separate function, where this function will generate digital values and be interpreted by DAC and generate the required waveform. For simplification the following table shows each function approach:

#	**Waveform**	**Approach**
1	Square Wave	To generate Square Wave the microcontroller gives alternatively 00h (low) and FFh (high) outputs as an input to DAC after some delay. The DAC will generate corresponding alternate low and high Analog outputs through Op-Amp circuit as +12 V and -12 V that will generate Square Wave pattern.
2	Staircase Wave	To generate Staircase Wave the microcontroller first gives 00h (low) output and then after some delay it increases output in steps like 33h, 66h, 99h, CCh and FFh. The DAC will generate Analog output as per these inputs from microcontroller that looks like Staircase Wave.
3	Sine Wave	To generate Sine Wave it is required to make a table of data that contains values calculated using equation (Value = 5 + 5sin(?)) for different angle values like 30o, 60o, 90o,....... of ? Note: the value is Analog output value. The applied digital input must be corresponding to generate this Analog output The values from this table are given to DAC. So DAC will generate corresponding Analog output that generates Sine Wave in output.

4	Triangular Wave	To generate Triangular Wave the microcontroller first gives data from 00h to FFh and then from FFh to 00h. This will generate linearly increasing and decreasing output through Op-Amp that will generate Triangular Wave.

Software Implementation:

We will use the same protocol implemented in the software debugger but with some modifications. Rather than sending address and data we will send waveform # followed by amplitude and frequency, therefore the command will be:

```
@ [0...3] [000...255] [000..255];
```

Code:

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

#include <util/delay.h>

#include "waveform.h"
#include "uart.h"

#define _CMD_START_CNT 1
#define _CMD_END_CNT   1
#define _CMD_WAVE_CNT  1
#define _CMD_AMP_CNT   3
#define _CMD_FRQ_CNT   3

#define FULL_CMD_CNT (_CMD_START_CNT +  _CMD_WAVE_CNT + _CMD_AMP_CNT + _CMD_FRQ_CNT +
_CMD_END_CNT)
#define WAVE_OFFSET  (_CMD_START_CNT)
#define AMP_OFFSET   (_CMD_START_CNT +  _CMD_WAVE_CNT)
#define FREQ_OFFSET  (_CMD_START_CNT +  _CMD_WAVE_CNT + _CMD_AMP_CNT)
#define MARKER_END   (_CMD_START_CNT +  _CMD_WAVE_CNT + _CMD_AMP_CNT + _CMD_FRQ_CNT)
#define MARKER_START (0)

#define WAVEFORM_NUM 4

#define DAC_DDR  DDRB
#define DAC_PORT PORTB
```

```
typedef enum {GENERATE_WAVE, UPDATE_WAVE} states_t;

static uint8_t cmd_buffer[FULL_CMD_CNT];
static void (*waveform[WAVEFORM_NUM])(uint8_t amp, uint8_t freq);
static states_t currentState = GENERATE_WAVE;
static uint8_t amp_value = 0;
static uint8_t freq_value = 0;
static uint8_t waveform_index = WAVEFORM_NUM;


void squareWave(uint8_t amp, uint8_t freq)
{
    // TODO: Place ur code here
    DAC_DDR = 255;
    DAC_PORT = 1;
}

void staircaseWave(uint8_t amp, uint8_t freq)
{
    // Refresh DAC DDR to be output.
    DAC_DDR = 255;

    // Generate waveform.
    DAC_PORT = 0x00;
    _delay_us(200);
    DAC_PORT = 0x33;
    _delay_us(200);
    DAC_PORT = 0x66;
    _delay_us(200);
    DAC_PORT = 0x99;
    _delay_us(200);
    DAC_PORT = 0xCC;
    _delay_us(200);
    DAC_PORT = 0xFF;
    _delay_us(200);
}

void triangleWave(uint8_t amp, uint8_t freq)
{
    // TODO: Place ur code here
    DAC_DDR = 255;
    DAC_PORT = 3;
}

void sineWave(uint8_t amp, uint8_t freq)
{
    // TODO: Place ur code here
    DAC_DDR = 255;
    DAC_PORT = 4;
}

void WAVE_Init(void)
{
```

```c
    uint8_t i;

    /* Init UART driver. */
    UART_cfg my_uart_cfg;

    /* Set USART mode. */
    my_uart_cfg.UBRRL_cfg = (BAUD_RATE_VALUE)&0x00FF;
    my_uart_cfg.UBRRH_cfg = (((BAUD_RATE_VALUE)&0xFF00)>>8);

    my_uart_cfg.UCSRA_cfg = 0;
    my_uart_cfg.UCSRB_cfg = (1<<RXEN)  | (1<<TXEN) | (1<<TXCIE) | (1<<RXCIE);
    my_uart_cfg.UCSRC_cfg = (1<<URSEL) | (3<<UCSZ0);

    UART_Init(&my_uart_cfg);


    /* Clear cmd_buffer. */
    for(i = 0; i < FULL_CMD_CNT; i += 1)
    {
        cmd_buffer[i] = 0;
    }

    /* Initialize waveform array. */
    waveform[0] = squareWave;
    waveform[1] = staircaseWave;
    waveform[2] = triangleWave;
    waveform[3] = sineWave;

    /* Start with getting which wave to generate. */
    currentState = UPDATE_WAVE;
}

void WAVE_MainFunction(void)
{

    // Main function must have two states,
    // First state is command parsing and waveform selection.
    // second state is waveform executing.
    switch(currentState)
    {
        case UPDATE_WAVE:
        {
            UART_SendPayload((uint8_t *)">", 1);
            while (0 == UART_IsTxComplete());

            /* Receive the full buffer command. */
            UART_ReceivePayload(cmd_buffer, FULL_CMD_CNT);

            /* Pull unitl reception is complete. */
            while(0 == UART_IsRxComplete());

            /* Check if the cmd is valid. */
            if((cmd_buffer[MARKER_START] == '@') && (cmd_buffer[MARKER_END] == ';'))
        {
            // Extract amplitude and freq values before sending them to the waveform generator.
            /* Compute amplitude. */
            {
                char _buffer[_CMD_AMP_CNT];
                for(uint8_t i = 0; i < _CMD_AMP_CNT; ++i) { _buffer[i] = cmd_buffer[AMP_OFFSET+i]; }
                amp_value = atoi(_buffer);
            }

            /* Compute frequency. */
            {
                char _buffer[_CMD_FRQ_CNT];
                for(uint8_t i = 0; i < _CMD_FRQ_CNT; ++i) { _buffer[i] = cmd_buffer[FREQ_OFFSET+i]; }
                freq_value = atoi(_buffer);
            }

            /* Compute waveform. */
            {
                waveform_index = cmd_buffer[WAVE_OFFSET] - '0';
            }
        }
        else
        {
          /* Clear cmd_buffer. */
            for(uint8_t i = 0; i < FULL_CMD_CNT; i += 1)
            {
                cmd_buffer[i] = 0;
            }
        }

        // Trigger a new reception.
        UART_ReceivePayload(cmd_buffer, FULL_CMD_CNT);

        UART_SendPayload((uint8_t *)"\r>", 2);
        while (0 == UART_IsTxComplete());
    }
    case GENERATE_WAVE:
    {
        // Execute waveform..
        if(waveform_index < WAVEFORM_NUM)
        {
            waveform[waveform_index](amp_value, freq_value);
        }
        // Keep in generate wave if no command it received.
        currentState = (1 == UART_IsRxComplete()) ? UPDATE_WAVE : GENERATE_WAVE;
        break;
    }
    default: {/* Do nothing.*/}
  }
}
```