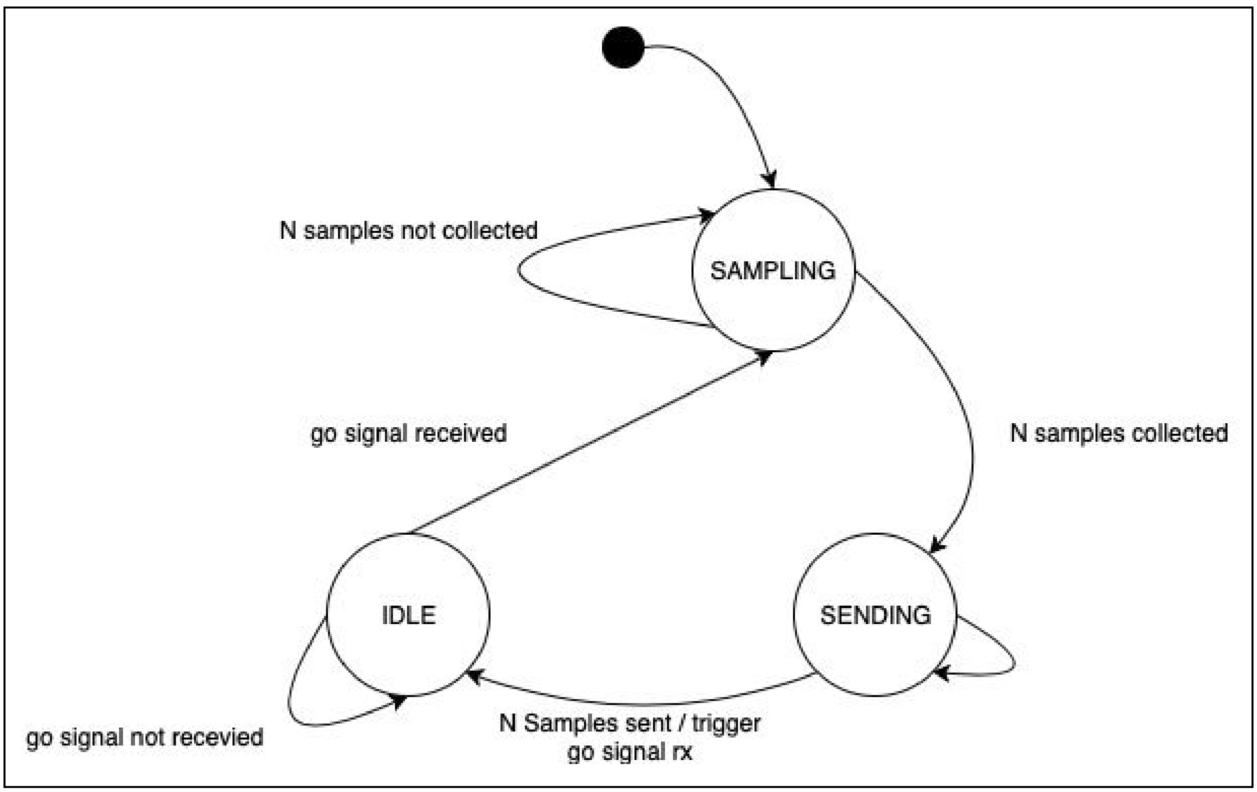
CI - Lap 006 - Oscilloscope

Lab Target:

Using UART with ADC to convert analog signals to digital signals and creating an oscilloscope which can be used for debugging different development boards.

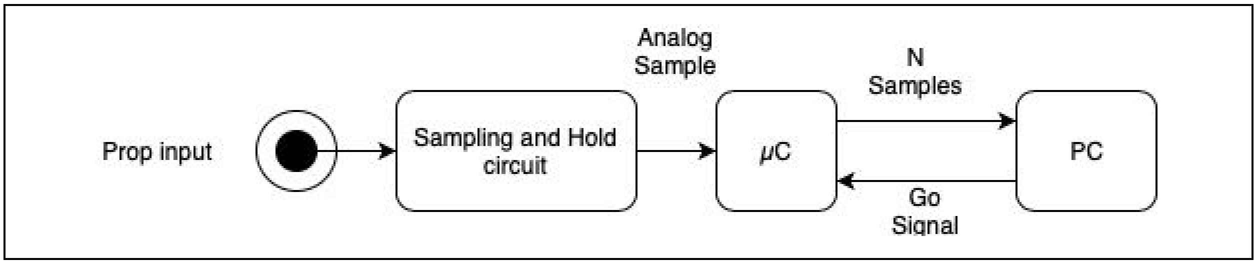
Theory:

Similar to lab 5 (Logic Analyzer) we will use the same state machine but with a little modification, where we will remove the MONITOR state from the state machine diagram and the initial state will be the SAMPLING state. In this state the µC will sample K samples then send them to the PC and wait for the go signal to restart a new sampling state. The following diagram shows the state machine transitions:

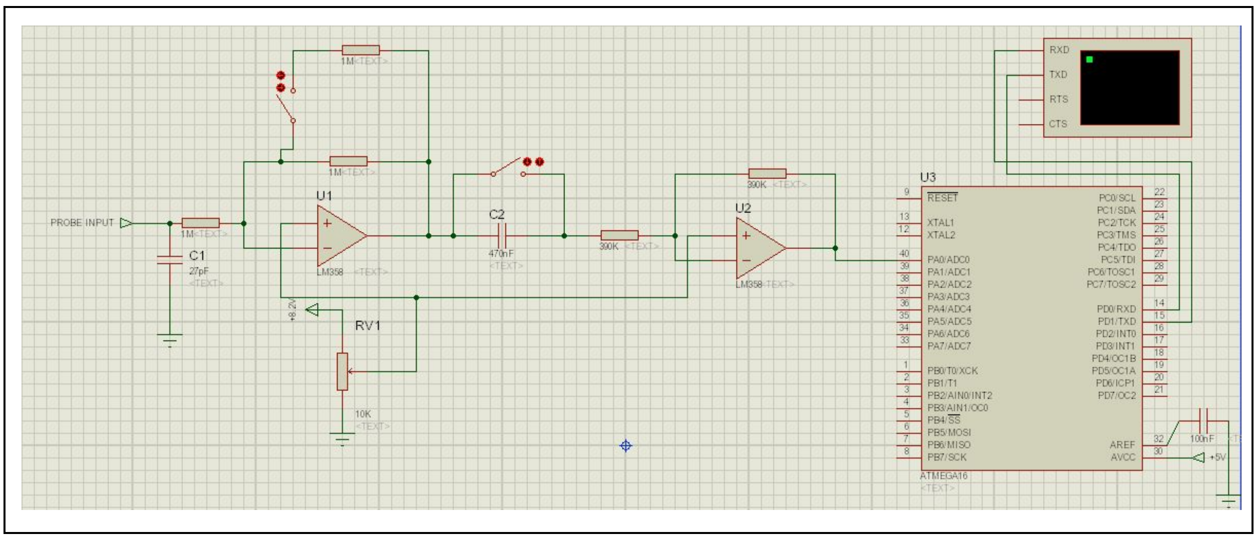


Circuit Theory:

The operating voltage of the circuit is 12V DC. By this voltage, the power supply is producing 2 voltages. +8.2V for IC1 and +5V for IC2 and IC3. This circuit can measure from +2.5V to -2.5V or from 0 to +5V dependent by S1 position (AC or DC input). By using probe with 1:10 division you can measure almost 10 times higher voltages. Moreover, with S2 you can make an extra division by 2 the input voltage. The following figure shows the block diagram:



Circuit Diagram:



States Description:

#	State	Actions
1	SAMPLING	Executes ADC conversion then value and its changing time value and saves them in a buffer of size _SAMPLES_NUM, then checks if the count of captured samples > _SAMPLES_NUM it changes the state to “SENDING” otherwise it returns to “SAMPLING” state.
2	SENDING	Sends the _SAMPLES_NUM samples to the pc and triggers a rx signal from the PC.
3	IDLE	Waits for PC signal to start the whole process again.

Software Implementation:

A sample contains sample value + time sanp value.Therefore each sample will be sent using the following format:

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

Code:

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

#include <util/delay.h>

#include "oscilloscope.h"
#include "uart.h"

#define _CMD_START_CNT 1
#define _CMD_END_CNT   1
#define _CMD_SPACING   1
#define _CMD_PINS_ST   1
#define _CMD_TIME_SNAP 4

#define FULL_SAMPLE_CNT (_CMD_START_CNT + _CMD_PINS_ST +  _CMD_TIME_SNAP + _CMD_END_CNT)

#define _SAMPLE_PIN  (_CMD_START_CNT)
#define _SAMPLE_TIME (_CMD_START_CNT + _CMD_PINS_ST)

#define MARKER_END   (FULL_SAMPLE_CNT - 1)
#define MARKER_START (0)

// Send the following frame for each sample:
// @PIN TIME3 TIME2 TIME1 TIME0;

#define _SAMPLES_NUM 200

#define LOGIC_PORT PINB

typedef enum {SAMPLING, SENDING, IDLE} states_t;


static logic_port_state = 0;
static logic_port_pre_state;
static states_t currentState = SAMPLE;
static uint8_t  analog_samples[_SAMPLES_NUM];
static uint32_t time_snap[_SAMPLES_NUM];
```

```c
uint32_t getTime(void)
{
    // TODO: Place your code here, to compute the elapsed time.
}

uint8_t getADCSample(void)
{
    uint8_t ADCvalue = 0;
    ADCSRA |= (1 << ADSC);
    while(!(ADCSRA & (1<<ADIF))); // waiting for ADIF, conversion complete
    ADCvalue = ADCH;
    return ADCvalue;
}

void OSCI_Init(void)
{
    /* 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);


    // TODO: Place your code here for timer1 initialization to normal mode and keep track
    // to time elapsed.
    {

    }

    // Initialize ADC.
    {
        ADMUX = 0b01100000;    // PA0 -> ADC0, ADLAR=1 (8-bit)
        ADCSRA |= ((1<<ADEN) | (1<<ADSC) | (1<<ADPS1)); // ADC prescaler at 4
    }

    /* Clear cmd_buffer. */
    for(uint8_t i = 0; i < FULL_SAMPLE_CNT; i += 1) { cmd_buffer[i] = 0; }

    /* Start with analog sampling. */
    currentState = SAMPLING;
}

void OSCI_MainFunction(void)
{
    static volatile uint8_t samples_cnt = 0;
```

```
static char _go_signal_buf = 'N';
// Main function must have two states,
// First state is command parsing and waveform selection.
// second state is waveform executing.
switch(currentState)
{
  case SAMPLING:
  {
    // DO here sampling.
    analog_samples[samples_cnt] = getADCSample();
    time_snap[samples_cnt]      = getTime();

    // Increment sample count.
    samples_cnt++;

    // Start sending the collected _SAMPLES_NUM samples.
    currentState = (samples_cnt >= _SAMPLES_NUM) ? SENDING : MONITOR;
    break;
  }
  case SENDING:
  {
    // For _SAMPLES_NUM samples send the construct the buffer.
    static uint8_t _sample_buf[FULL_SAMPLE_CNT];
    for(uint8_t i = 0; i < _SAMPLES_NUM; ++i)
    {
      // Construct the buffer.

      // Add buffer marker
      _sample_buf[MARKER_START] = '@';

      // Add pin value.
      _sample_buf[_SAMPLE_PIN]  = analog_samples[i];

      // Add time snap value.
      _sample_buf[_SAMPLE_TIME + 0] = ((time_snap[samples_cnt] & 0xFF000000) >> 24);
      _sample_buf[_SAMPLE_TIME + 1] = ((time_snap[samples_cnt] & 0x00FF0000) >> 16);
      _sample_buf[_SAMPLE_TIME + 2] = ((time_snap[samples_cnt] & 0x0000FF00) >> 8);
      _sample_buf[_SAMPLE_TIME + 3] = ((time_snap[samples_cnt] & 0x000000FF) >> 0);

      _sample_buf[MARKER_END]   = ';';

      // Send sample.
      UART_SendPayload(_sample_buf, FULL_SAMPLE_CNT);
      while (0 == UART_IsTxComplete());
    }

    // Trigger receiving for go signal.
    UART_ReceivePayload(&_go_signal_buf, 1);
  }
  case IDLE:
  {
    currentState = ((1 == UART_IsRxComplete())&&(_go_signal_buf == 'G')) ? SAMPLING : IDLE;
```

```
            if(currentState == SAMPLING)
            {
                _go_signal_buf = 'N';
                // TODO: Place your code here to reset the timer value.
            }

            break;
        }
        default: {/* Do nothing.*/}
    }
}
```