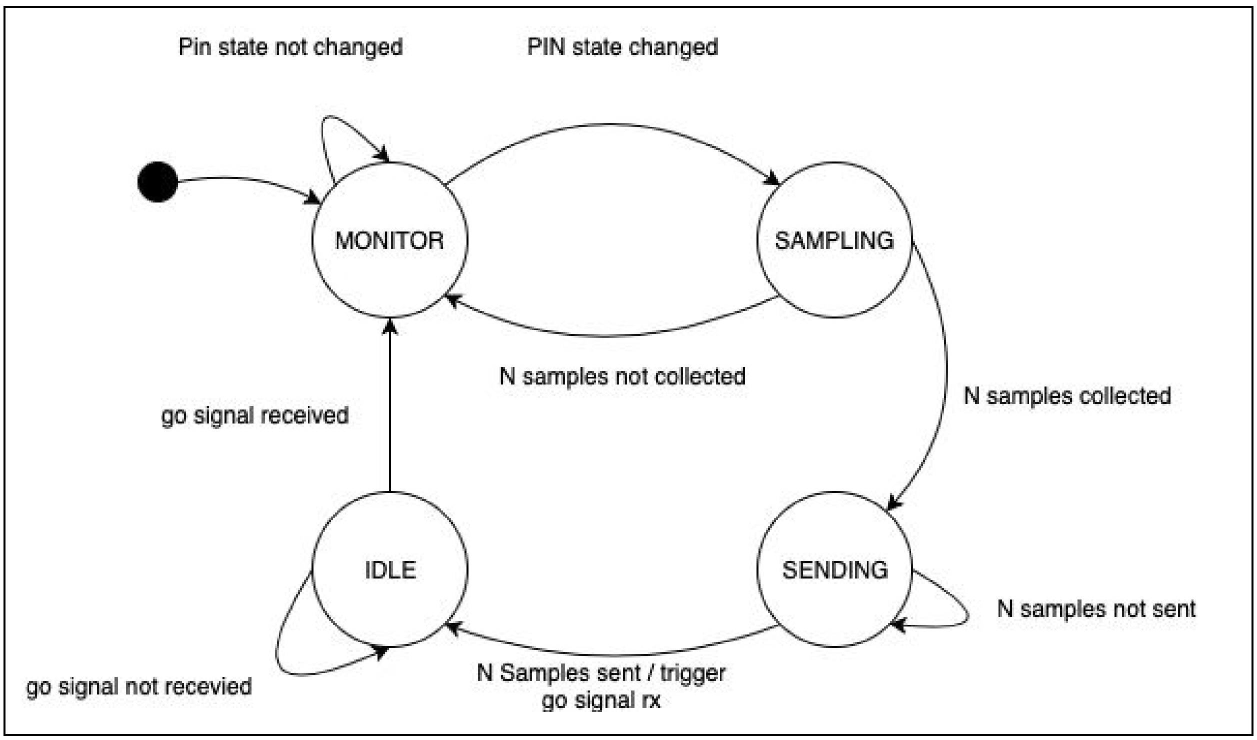
CI - Lap 005 - Logic Analyzer

Lab Target:

Using UART with traditional IO read to capture digital signals and creating a logic analyzer which can be used for debugging different communication protocols like I2C, SPI and others.

Theory:

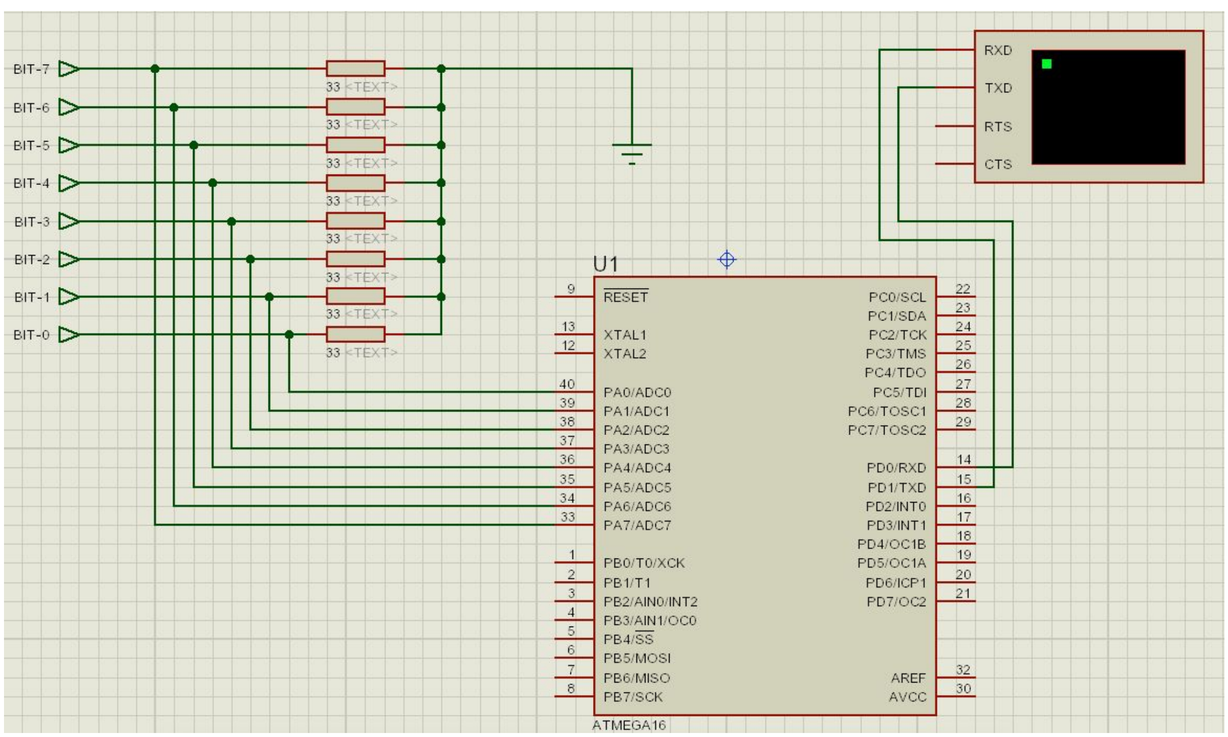
Since digital communication protocols uses digital signal value {0, 1}. Therefore, all we need is to read the µC PIN value then send this value using UART for plotting. To make things more interesting we will develop a state machine to have a robust logic analyzer. This state machine contains 4 states: {MONTOR, SAMPLING, SENDING, IDLE}. The following diagram shows the state machine transitions:



States Description:

#	State	Actions
1	MONITOR	Keep watching Logic_PORT if changed then it changes the state to "SAMPLING".
2	SAMPLING	Reads the Logic_PORT 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 "MONITOR" state.
3	SENDING	Sends the _SAMPLES_NUM samples to the pc and triggers a rx signal from the PC.
4	IDLE	Waits for PC signal to start the whole process again.

Circuit Diagram:



Software Implementation:

A sample contains PORT 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 "logicAnalyzer.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_DDR  DDRB
#define LOGIC_PORT PINB

typedef enum {MONITOR, SAMPLING, SENDING, IDLE} states_t;


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

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

void LOGIC_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.
   {

   }

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

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

void LOGIC_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 MONITOR:
      {
         LOGIC_DDR = 0;
         logic_port_pre_state = logic_port_state;
         logic_port_state     = LOGIC_PORT;
         currentState = (logic_port_pre_state != logic_port_state) ? SAMPLING : MONITOR;
         break;
      }
```

```
case SAMPLING:
{
    // DO here sampling.
    LOGIC_DDR = 0;
    pin_states[samples_cnt] = LOGIC_PORT;
    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]  = pin_states[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')) ? MONITOR : IDLE;

    if(currentState = MONITOR)
    {
        // TODO: Place your code here to reset the timer value.
    }
```

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