Documentation on Programming IR Receiver and Transmitter in Atmega328p and ATMEGA32A respectively

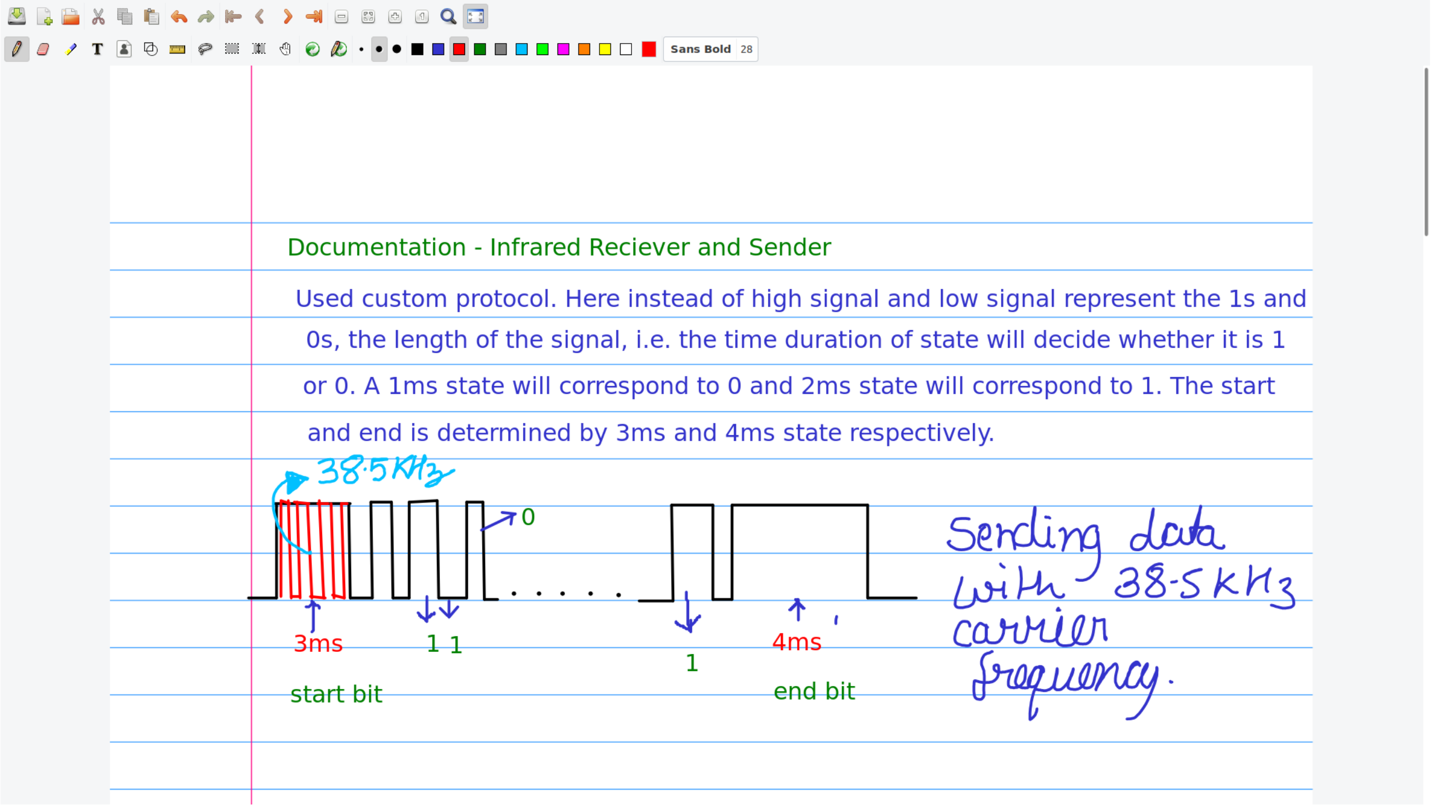
References

https://www.st.com/resource/en/application_note/dm00271524-implementation-of-transmitters-and-receivers-for-infrared-remote-control-protocols-with-stm32cube-stmicroelectronics.pdf

This is the reference I followed.

Although the above reference is based on STM32 microcontrollers but the same logic can be applied for ATMEGA32 as well.

Protocol Used



Documentation - Infrared Reciever and Sender

Used custom protocol. Here instead of high signal and low signal represent the 1s and 0s, the length of the signal, i.e. the time duration of state will decide whether it is 1 or 0. A 1ms state will correspond to 0 and 2ms state will correspond to 1. The start and end is determined by 3ms and 4ms state respectively.

Sending data with 38.5 kHz carrier frequency.

# IR Sender (ATMEGA32A)

We begin by adding the required libraries to our program and defining the frequency of the CPU for _delay_ms() to work properly. Here we also define DATAFRAME, which basically tells us no. of bits of data that is being sent. We also initialize two variables for our need.

In [ ]:

```
#define F_CPU 8000000UL
#include<avr/io.h>
#include<util/delay.h>
#define DATAFRAME

volatile int i;
int data[DATAFRAME];
```

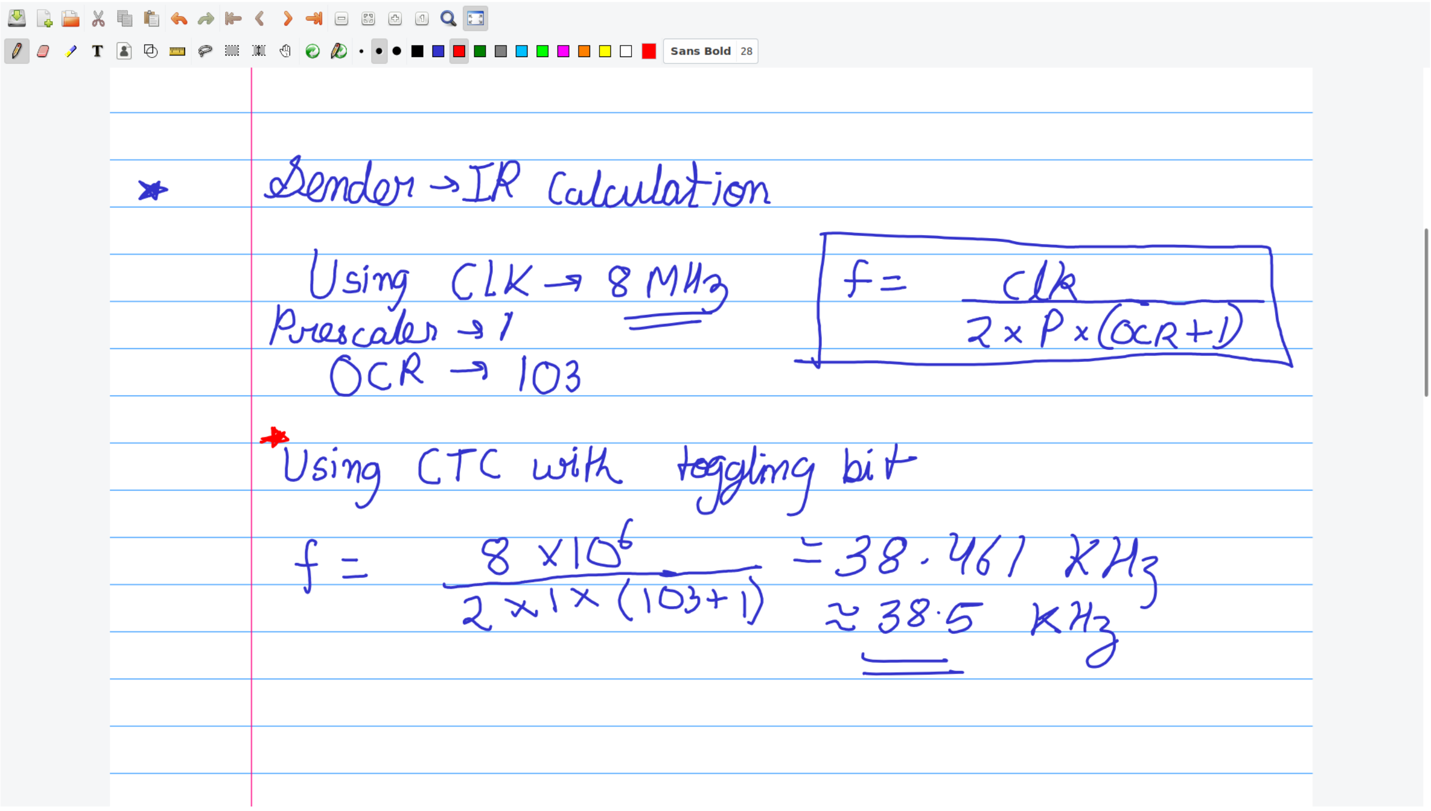
In order to start sending data, we have to know that Infrared communication usually works with a carrier frequency of 36Khz - 40Khz. So for that, we will setup the timer and clock in CTC mode in such a manner that the output is toggled every (1/38kHz) seconds.This creates a signal of 38.5Khz, i.e. PWM with 38.5 Khz frequency

and 50% Duty Cycle.

Here I am using 8 bit timer in CTC mode with prescaler as 1. Output Compare Register will store 103 in order to generate required frequency.

In [ ]:

```
int main()
{

  DDRB = _BV(DDB3);
  OCR0 = 103;
  TCCR0 |= _BV(WGM01);
  TCCR0 |= _BV(CS00);


  sei();


  while (1)
  {

  }

  return 0;
}
```



* Sender → IR Calculation

Using CLK → 8 MHz
Prescaler → 1
OCR → 103

$$\boxed{f = \frac{clk}{2 \times P \times (OCR+1)}}$$

* Using CTC with toggling bit

$$f = \frac{8 \times 10^6}{2 \times 1 \times (103+1)} \simeq 38.461 \text{ KHz} \approx \underline{38.5} \text{ KHz}$$

Now that we have up setup the carrier frequency, we will now define a function that will take in a number and it will convert it into an array of binary numbers which will be sent through IR Led. In our case, we will have two numbers. First one will be integer and the other one will be ID of the bot.

In [ ]:

```
void convert(int num, int botID)
{
  int index = (DATAFRAME-5);
  while (index >= 0)
  {
    data[index] = num & 1;
    index--;
    num >>= 1;
  }
  index = DATAFRAME-1;

  while (index >= DATAFRAME-4)
  {
    data[index] = botID & 1;
    index--;
    botID >>= 1;
  }
}
```

After the successful conversion, our data will be ready to be sent. Hence we will create a send function, that will send data acording to our Protocol.

In [ ]:

```
void send1()
{
  //To Signify Starting Bit - Starts with 3ms pulse

  TCCR0 ^= _BV(COM00);
  _delay_ms(3);


  //1ms pulse if 0 or else 2ms pulse
  for (i = 0; i < DATAFRAME; i++)
  {
    TCCR0 ^= _BV(COM00);
    _delay_ms(1);
    if (data[i])
      _delay_ms(1);
  }

  //To Signify Ending Bit - Ends with 4ms pulse
  TCCR0 ^= _BV(COM00);
  _delay_ms(4);
  TCCR0 ^= _BV(COM00);
}
```

That's it, we are ready to send the data.

In [ ]:

```
//On ATMEGA32 !! clk 8Mhz internal
#define F_CPU 8000000UL //setting up frequency for _delay_ms()
#include<avr/io.h>
#include<util/delay.h>
#define DATAFRAME 20
/////////////////////////////////////////////////////////////////////////
//Global Variables
volatile int i;
int data[DATAFRAME];
/////////////////////////////////////////////////////////////////////////
void convert(int num, int botID) //Converts Number to Array of Binary Digits
{
  int index = (DATAFRAME - 5); //Stores Data - 16bit
  while (index >= 0)
  {
    data[index] = num & 1;
    index--;
    num >>= 1;
  }


  index = DATAFRAME - 1;
  while (index >= DATAFRAME - 4) //Stores Bot ID - 4bit
  {
    data[index] = botID & 1;
    index--;
    botID >>= 1;
  }
}
////////////////////////////////////////////////////////////////////////
void send1()
{
  //To Signify Starting Bit - Starts with 3ms pulse
  TCCR0 ^= _BV(COM00);
  _delay_ms(3);


  //1ms pulse if 0 or else 2ms pulse
  for (i = 0; i < DATAFRAME; i++)
  {
    TCCR0 ^= _BV(COM00);
    _delay_ms(1);
    if (data[i])
      _delay_ms(1);
  }

  //To Signify Ending Bit - Ends with 4ms pulse
  TCCR0 ^= _BV(COM00);
  _delay_ms(4);
  TCCR0 ^= _BV(COM00);
}

////////////////////////////////////////////////////////////////////////
int main()
{

  DDRB = _BV(DDB3);
  OCR0 = 103; //38.5KHz Signal is required
  //TCCR0 |= _BV(COM00);
  TCCR0 |= _BV(WGM01);
  TCCR0 |= _BV(CS00);

  sei();


  while (1)
  {
    convert(67, 4); //Put here the data to send
    send1(); //sends the data
    _delay_ms(1500);
  }

  return 0;
}
////////////////////////////////////////////////////////////////////////////////
```

Now one thing that is left is runtime user input. But unfortunately, I am using ATMEGA32 chip and I don't have a USB to ttl for serial monitor. So I just added delay function.

IR Reciever (ATMEGA328p)

Creating the reciever was the trickiest part. The entire solution for this problem was to use external interrupts. So, initially IR reciever will be connected to external interrupt pin present(PD2). Now, suppose let t=0 when the first signal is just recieved. The signal on interrup pin will be low but as soon as IR Sensor recieves the IR signal, it will become high. We see that the signal rises here, when the communication begins. Hence we will enable rising edge interrupt before recieving the signal. As soon as rising edge is detected, the interrupt subroutine will be called. Now it is quite obvious that the next signal will become low and it will lead to falling edge. Let's say that happens at t=3ms. We will always be monitoring time between rise and fall signals.

The trick is to do all the required calculation in between t=0 and t=3ms, then change the interupt mode to falling edge to detect the time at which signal changes.Then the same logic will be applied further.

In order to measure time between two consecutive edge interrupts, we will be using another Timer interrupt to keep track of time which will be called every 10us where a variable counter will be incremented by 1.

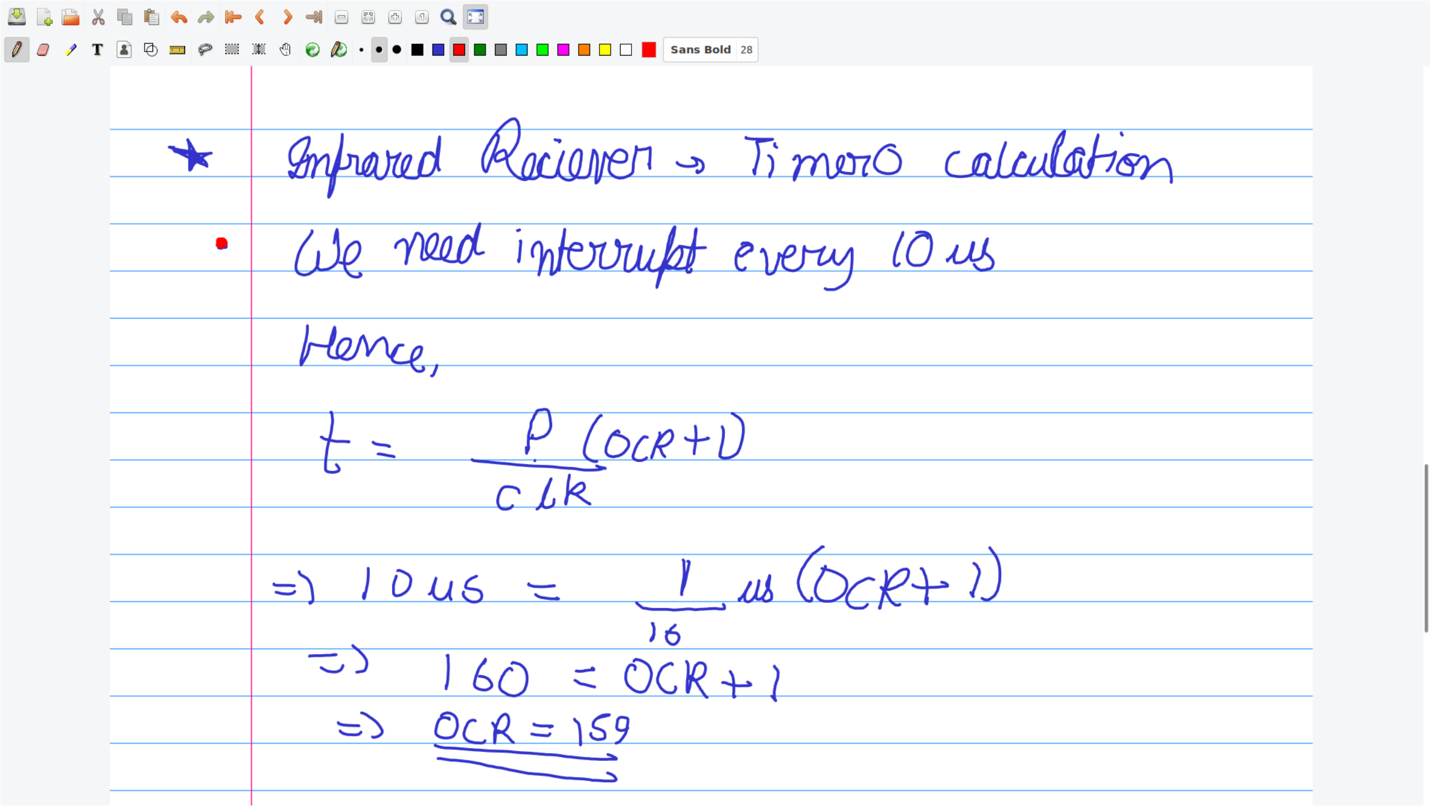
We'll use this calculated time to compare it with our protocol. So suppose if the time dekay between two edge

interrupt was 1ms, then it is 0. If 2ms, then 1. 3ms and 4s delay represents start and stop bit respectively.

In [ ]:

```
//IR // On Atmega328P!!! 16Mhz
/////////////////////////////////////////////////////////////////
#define OCR_VAL 159
#define highSignal 0
#define lowSignal 1
#include <avr/io.h>
#include <avr/interrupt.h>
/////////////////////////////////////////////////////////////////
volatile bool started = false;
/////////////////////////////////////////////////////////////////
void setupTimer();
void setupExternalInterrupt();
void convert();
////////////////////////////////////////////////////////////////
volatile unsigned int timeKeep;
volatile unsigned int data[20];
volatile unsigned int nextSignalTime;
volatile bool nextSignal;
volatile bool state = false;
volatile unsigned int num = 0;
volatile unsigned int inc = 0;
volatile unsigned int botID = 0;
/////////////////////////////////////////////////////////////////
ISR(TIMER0_COMPA_vect)
{
  timeKeep++;
}
/////////////////////////////////////////////////////////////////
ISR(INT0_vect)
{
  nextSignalTime = timeKeep;
  EIMSK &= ~_BV(INT0);

  sei();

  timeKeep = 0;
  TCNT0 = 0;

  if (inc == 16)
    state = true;

  if ((nextSignalTime > 360) && (nextSignalTime < 440) && started)
  {
    Serial.println(String(num)+","+String(botID));
    reset();
  }

  else if (nextSignal == highSignal)
  {
    nextSignal = lowSignal;
    EICRA &= (~((1 << ISC01) | (1 << ISC00)));
    EICRA = _BV(ISC01);
  }

  else if (nextSignal == lowSignal)
  {
    nextSignal = highSignal;
    EICRA |= _BV(ISC00);
  }

  if ((nextSignalTime > 260) && (nextSignalTime < 340))
    started = true;

  else if ((nextSignalTime > 60) && (nextSignalTime < 140))
  {
    if (started)
    {
      if (state)
      {
        botID <<= 1;
        botID += 0;
      }
      else
      {
        num <<= 1;
        num += 0;
      }
      inc++;
    }

  }

  else if ((nextSignalTime > 160) && (nextSignalTime < 240))
  {
    if (started)
    {
      if (state)
      {
        botID <<= 1;
        botID += 1;
      }
      else
      {
        num <<= 1;
        num += 1;
      }
      inc++;
    }

  }
  /*else
    reset();*/




  EIMSK |= _BV(INT0);

}
////////////////////////////////////////////////////////////////
int main()
{
  init();
  Serial.begin(9600);

  PORTD = 0x0;

  setupTimer();
  setupExternalInterrupt();

  nextSignal = highSignal;

  sei();


  while (1)
  {

  }
  return 0;
}
////////////////////////////////////////////////////////////////
void setupTimer()
{
  TCCR0A = _BV(WGM01);
  TCCR0B = _BV(CS00);
  TIMSK0 = _BV(OCIE0A);
  OCR0A = OCR_VAL;
}
```

```
////////////////////////////////////////////////////////////////
void setupExternalInterrupt()
{
  EIMSK = _BV(INT0); //External interrupt on PD2
  EICRA = _BV(ISC00) | _BV(ISC01);
}
///////////////////////////////////////////////////////////////
void reset()
{
  nextSignal = highSignal;   //We expect high signal for the data to be received
  started = false;

  for (num = 0; num < 20; num++)
    data[num] = 0;
  num = 0;
  inc = 0;
  state = false;
  botID = 0;
  EICRA = (_BV(ISC00) | _BV(ISC01));
}
////////////////////////////////////////////////////////////////
```



★ Infrared Reciever → Timer0 calculation

We need interrupt every 10 us

Hence,

$$t = \frac{P\,(OCR+1)}{clk}$$

$$\Rightarrow 10us = \frac{1}{16}us\,(OCR+1)$$

$$\Rightarrow 160 = OCR+1$$

$$\Rightarrow \underline{\underline{OCR = 159}}$$