MEAM 5100 Mechatronics Lab 2

Saurav Agrawal

2.1.1 Get one of the small SPST switches from the ministore and solder wires to it (solid core). Plug the wires from the switch along with a resistor into a breadboard so that the state of the switch can be read by an input port on the ATmega32U4 as either 0V or 5V. Write code that will read the state of the switch and make an external LED turn on when the switch is depressed and turn off when the switch is not pressed. Submit a schematic of your switch, resistors, ATmega32U4 ports and LED and code (as a separate file in gradescope).

Answer: Using the values from the data sheet of the LED we found the corresponding resistance for the LED.

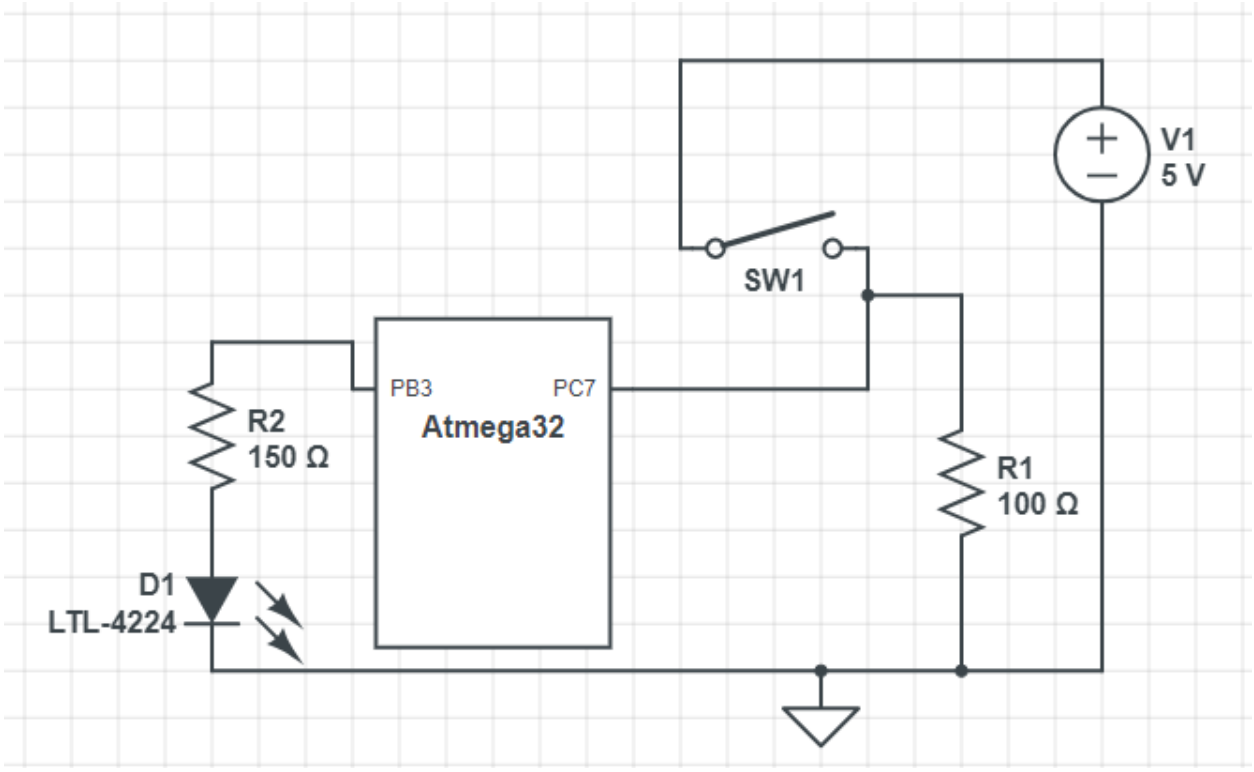


Fig 1: Circuit Diagram

Code:

```
/* Name: main.c (2.1.1)
* Author: Saurav Agrawal
* Copyright: <insert your copyright message here>
* License: <insert your license reference here>
*/
#include "MEAM_general.h"
int main()
```

```
{
 clear(DDRC,7);
 set(DDRB,3);
 while(1)
 {
   if (bit_is_set(PINC, 7))
     { set(PORTB,3);}
   else
    {clear(PORTB,3);}
 }
}
```

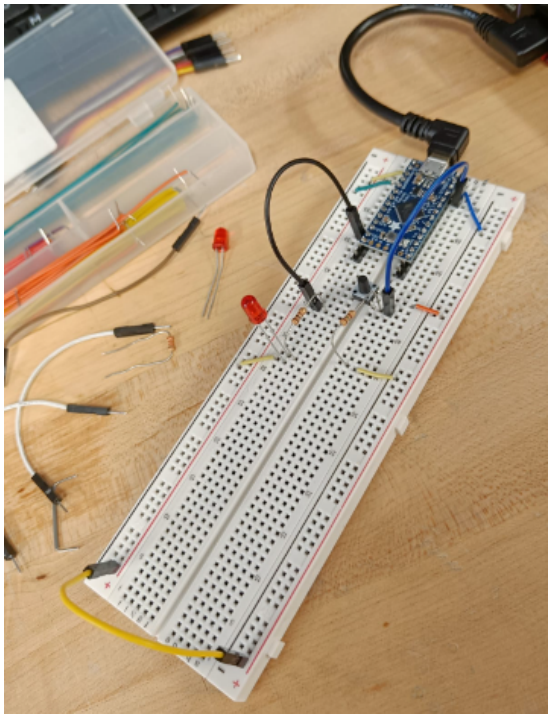


Fig 2 : Circuit using all Components

2.1.2 Add to the code from 2.1.1 so that it prints to a terminal using the m_usb serial commands each time the switch is depressed and each time it is released. Turn on timer3 setting the prescaler to divide by 1024 (use TCCR3B setting CS32:0 to be b101) and print out a time stamp (i.e use m_usb_tx_uint(TCNT3);) each time the switch changes state. Notice that if the switch is bouncing with each press (if you depress slowly, there may be more bounces) you may get multiple printouts with short (or same) time stamp values between them.
Add a passive low pass filter (capacitors and/or resistors) to the switch circuit so the switch is "debounced". Make sure you have a large enough RC time constant so that you don't see multiple bounces on a single press (i.e., 100Hz bounces), but not too large that you distort button presses that occur at roughly 10 times per second (faster than most humans can press). To verify the effect of the RC circuit, use the output of a square wave from a function

generator into the RC circuit and view the output on an oscilloscope. Notice how the signal changes as the frequency goes from below 10Hz to above 100Hz. Submit oscilloscope images for each case (ideally two traces one with output of input signal square wave and one with filtered output).

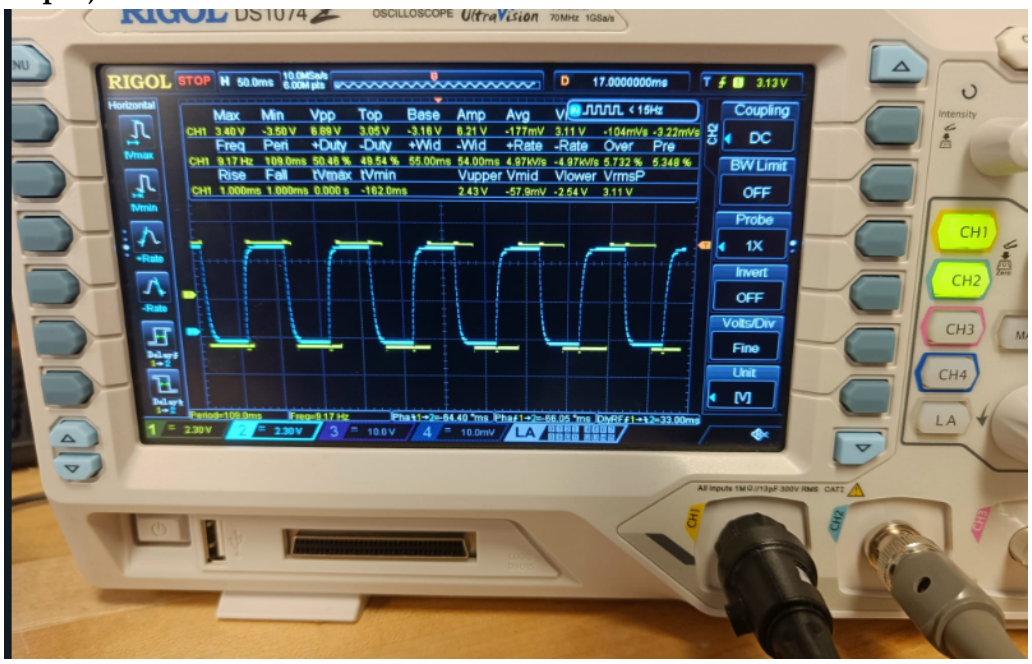


Fig 3: Filtered and Unfiltered square wave at frequency 10 Hz

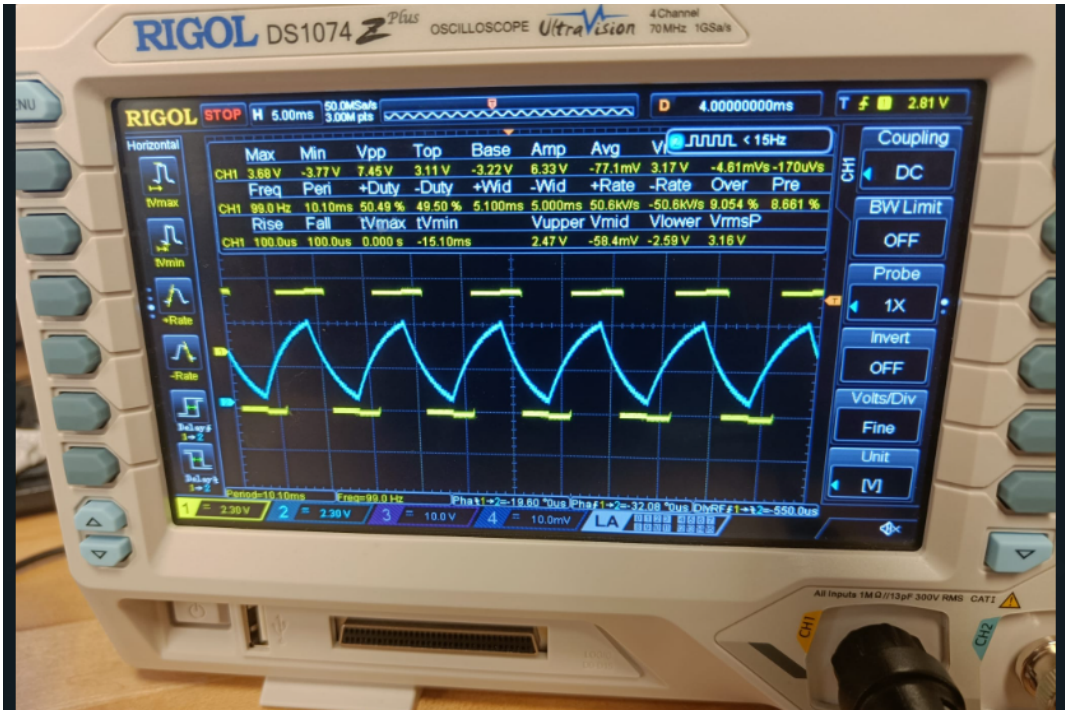


Fig 4: Filtered and Unfiltered square wave at frequency 100 Hz

Square waves have abrupt, sharp transitions from high to low voltage levels and vice versa. When a low pass filter is added, it smoothes out these sharp transitions. The result is that the sharp corners of the square wave are transformed into more gradual slopes, resembling the shape of a saw.
For 10 Hz, the corners are less rounded as compared to that of 100 Hz for the same low pass filter.

2.1.3 Use the input capture function of the timer on the Atmega32U4 to measure how fast you can depress a switch (defined as the time between down presses of the switch). Video gamers sometimes call this "mashing". Make sure that the switch is debounced, so you only get valid presses. Be sure to include time for the ATmega32U4 to print to the USB before exiting main().

Answer:

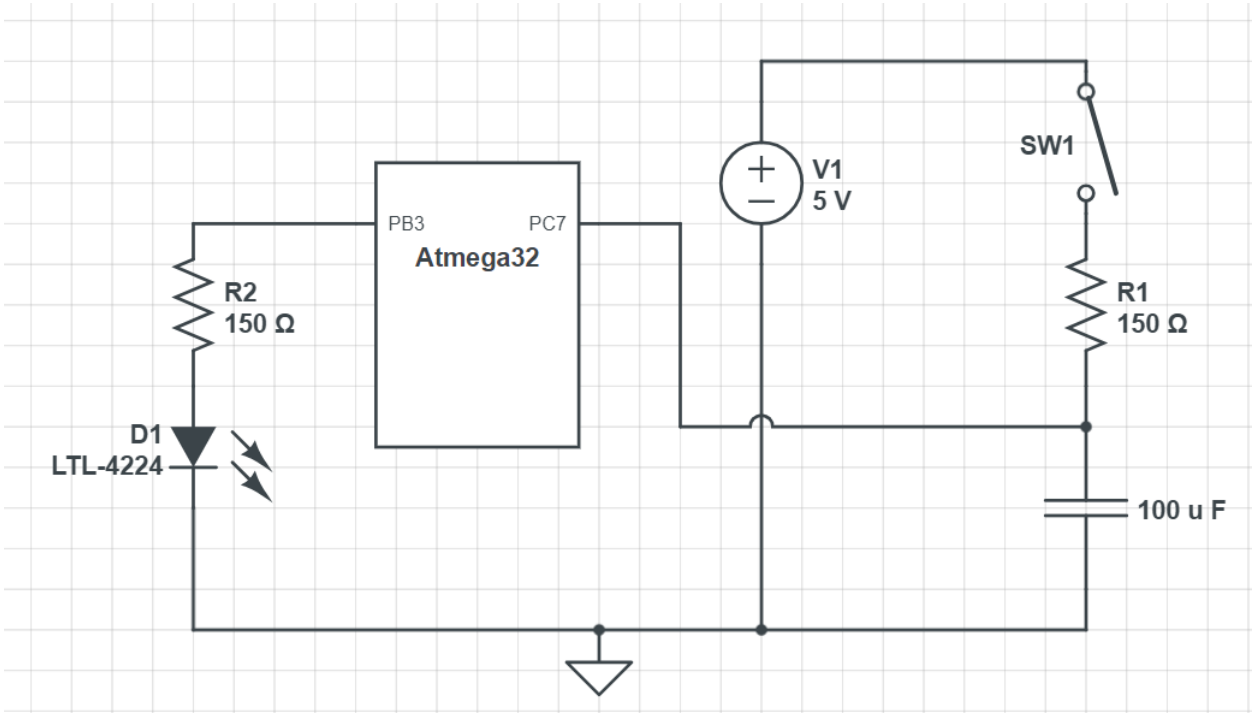


Fig 5 : Circuit of the low pass

Code:

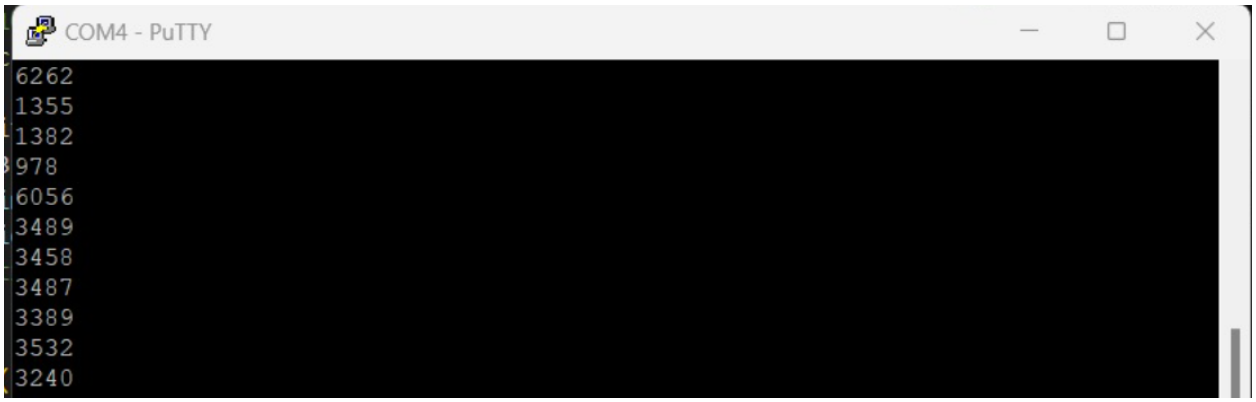
```
/* Name: main.c 2.1.3
* Author: Saurav Agrawal
* Copyright: <insert your copyright message here>
* License: <insert your license reference here>
*/
#include "MEAM_general.h"
#include "m_usb.h"
#define PRINTNUM(x) m_usb_tx_uint(x); m_usb_tx_char(10); m_usb_tx_char(13)
int i;
```

```
int oldtime;
int tperiod;
// char 10 and 13 are carriage return and line feed
void checkPC7()
{
while(!bit_is_set(TIFR3,ICF3));
set(TIFR3,ICF3);
// int oldtime = ICR3;
// int tperiod = ICR3-oldtime;
//m_usb_tx_int(ICR3);
}

int main()
{
  m_usb_init();
  set(TCCR3B,CS30);
  set(TCCR3B,CS32);

  clear(DDRC,7);
  // set(DDRB,3);
  while(1)
  {

   checkPC7();
   oldtime = ICR3;
   checkPC7();
   tperiod = ICR3-oldtime;
   m_usb_tx_int(tperiod);
   m_usb_tx_string("\n");
   _delay_ms(50);
   }
}
```



The fastest time I got is 978 clicks on the counter.

2.2.1 Create the pictured phototransistor circuit and observe the voltage at the point between the transistor and the resistor. This will be the output of the sensing circuit. Observe the voltage at the output on an oscilloscope as more or less light falls on the transistor (e.g., with the normal room lights on, cover or uncover it with your hand). Try the resistor values 10 x's larger and 10 x's smaller. In your write up answer the following question: How does the phototransistors behavior under different conditions change as you change the amount of light that falls on the transistor? Does it reach a valid logic low and valid logic high for the ATmega32U4.

Answer: The resistors 4.7 and 470 Kilo ohms, which are 10 times larger and lower than 47 Kilo ohms, were used to determine logic levels. The voltage level increased as the light on the phototransistor decreased. At 470 K ohms we can find a valid logic high and low for the circuit using the threshold given in the class for Atmega34u.

Resistance	**Room light**	**No light**
4.7 K ohms	**4.4**	**4.9**
47 kohm	**3.3**	**4.7**
470 kohm	**0.08**	**2.6**

Table 1: Comparison of voltage at different light intensities and resistanc**e**

2.2.2: Connect the output of the sensing circuit to a digital input on the ATmega32U4. Use an external LED attached to a pin configured as an output on the ATmega32U4. Create a program that reads the state of the phototransistor circuit and turns OFF the LED if there is light on the transistor and turns ON the LED if there is no light.

```
#include "MEAM_general.h"
int main()
{
 clear(DDRC,7);
 set(DDRB,3);
 while(1)
 {
   if (bit_is_set(PINC, 7))
     { set(PORTB,3);}
   else
    {clear(PORTB,3);}
 }}
```

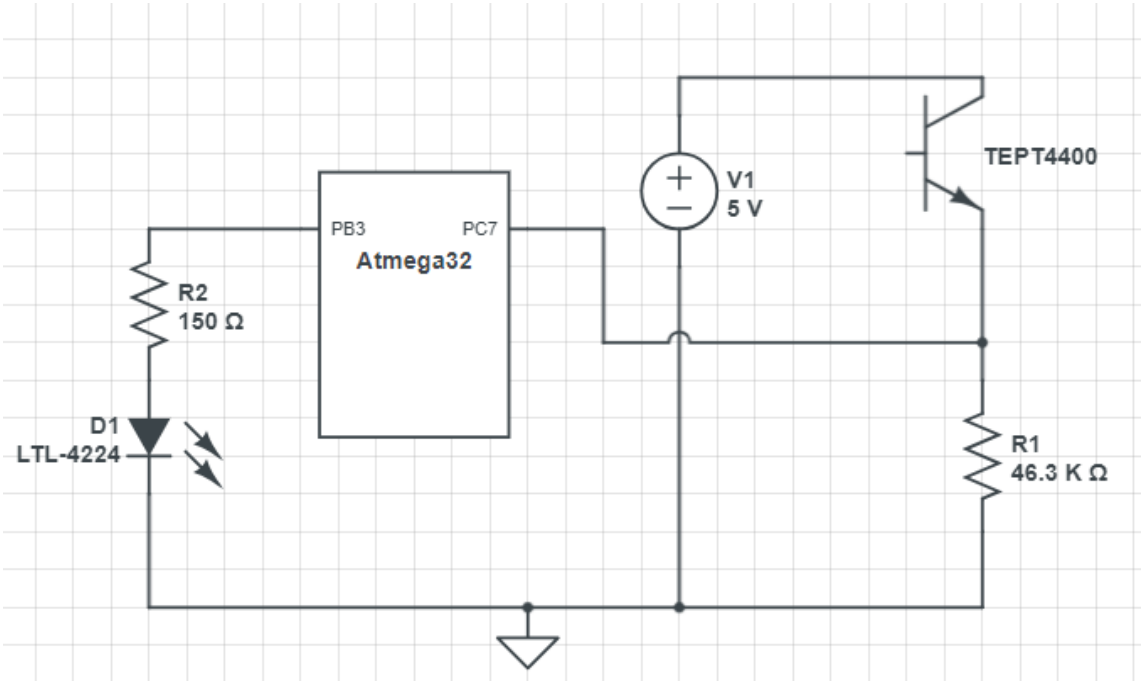


fig 6 : LED blinking circuit

2.3.1 Use a function generator to drive an IR LED "LTE-4208" (from the ministore) with a 5V square wave. Find the datasheet for this LED (on google or digikey) and choose a resistor to put in line with this diode so that the current is close to 30mA. Use a cellphone to verify that the IR LED is blinking. Create a photo detection circuit using an IR phototransistor "LTR-4206E" (from the ministore) that can show the output of the LED using an opamp from

the ministore that creates logic level output (for the different states of the LED being on or off) when the IR phototransistor and IR LED are pointed at each other. Try two frequencies 25 Hz and 662 Hz. Submit a circuitlab circuit diagram (indicate the output - where the scope is measuring) and images of an oscilloscope output of the circuit. One image showing that a frequency of about 25 Hz is being measured by the trigger line, and one snapshot of 662 Hz.

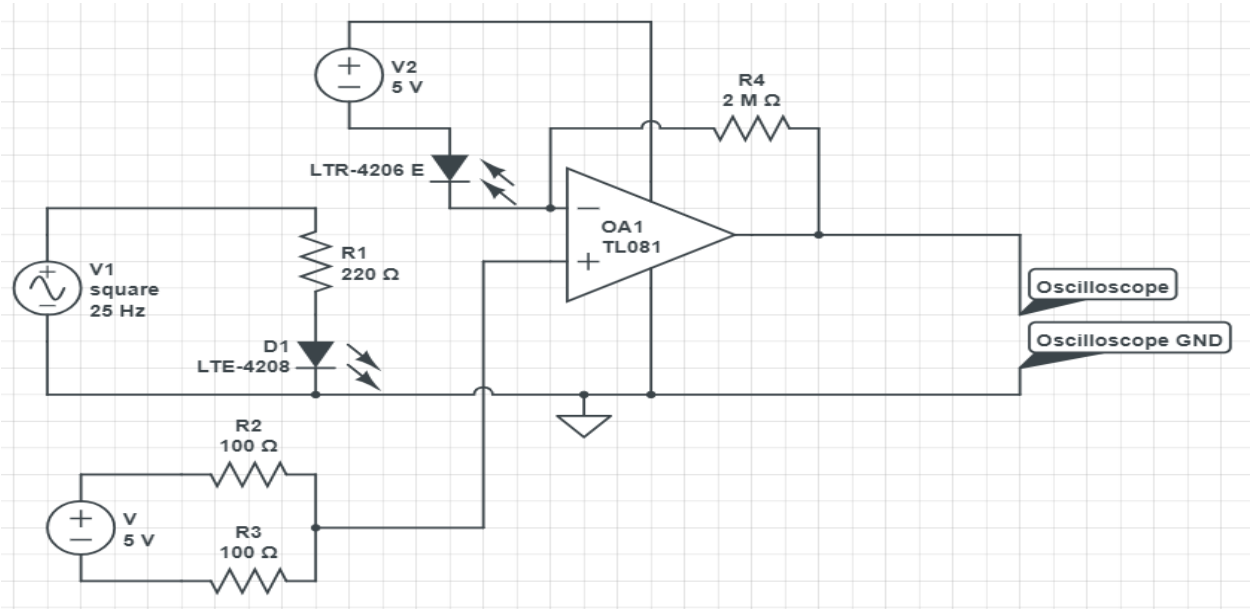


Fig 7: Circuit diagram

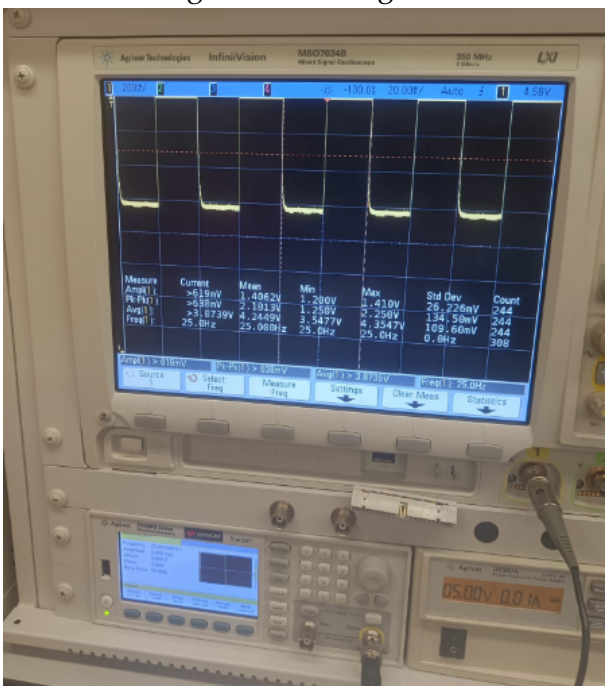


fig 8: Oscilloscope Image for 25 Hz

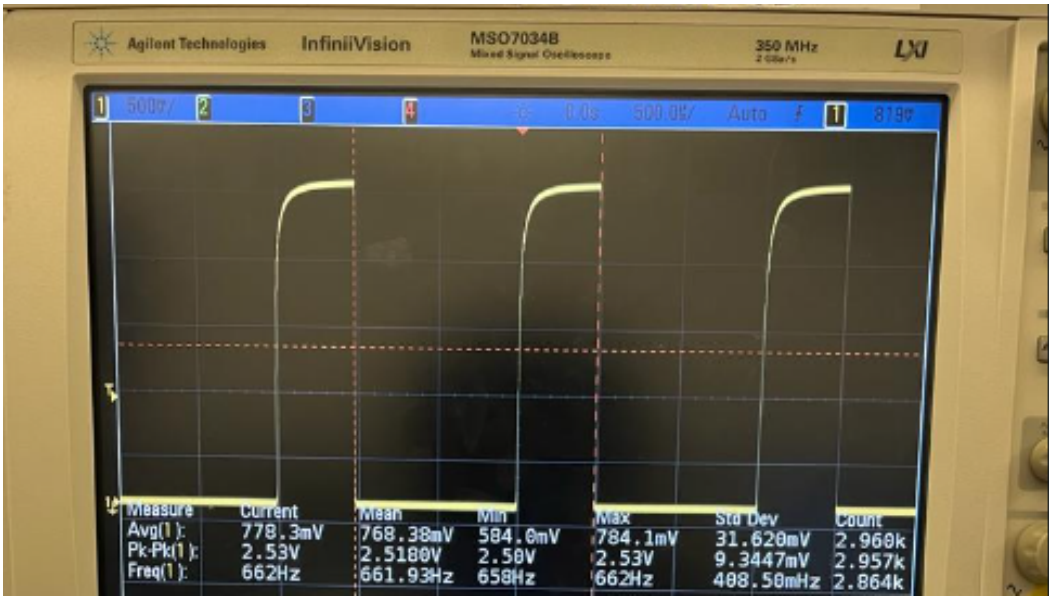


fig 9: Oscilloscope Image for 25 Hz

2.3.2 Connect the detection circuit to a ATmega32U4 and create a program that can detect frequencies of 25 Hz and 662Hz being transmitted and indicate when each frequency is being detected (e.g., turn on a red LED when 25Hz is detected and turn on a green LED when 662 Hz is detected and all LEDs off when neither signal is there). While the LTR-4206E has a visible light filter (the reason it has a dark color) it may still be sensitive to ambient light. The room lights may create 60Hz (120Hz) noise that may confuse your circuit, so, for this part, test with the setup covered so no room light falls on the sensor. Note also that sunlight has lots of IR 940nm frequency so the LTR-4206E will be sensitive to sunlight. You may try to shield the sensor from this ambient light as well. Submit your code, a circuit diagram. Get checked off by a TA (show during office hours and answer TA questions) that your circuit is working as desired.

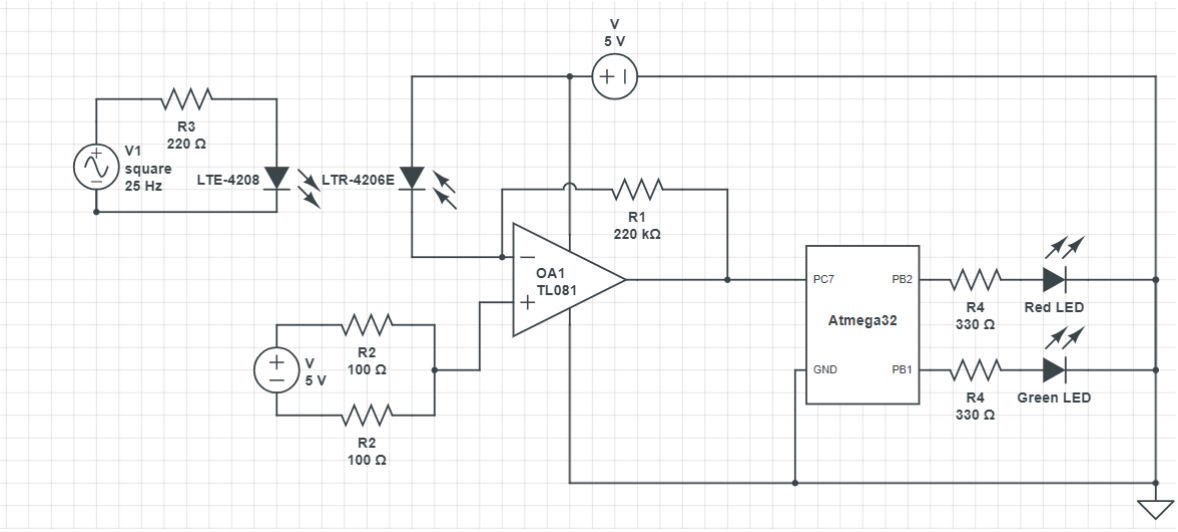


Fig 10 : Circuit diagram

Code:

```
/* Name: main.c 2.3.2
 * Author: Saurav Agrawal
 * Copyright: <insert your copyright message here>
 * License: <insert your license reference here>
 */
 #include "MEAM_general.h"
int find_period()
{    int pt;    // store time of previous rising edge
    int st = TCNT3;  // store start time

    set(TIFR3, ICF3);  // clear flag

    // wait for rising edge, exit if it takes too long
    while (!bit_is_set(TIFR3, ICF3)) {
    }
    int curr_time = ICR3;              // store time
    int diff = curr_time - pt;  // calculate time between edges
    pt = curr_time;                      // update previous time
    return diff;
}
int main(void)
{    _clockdivide(0);     // set clock speed to 16MHz
    set(TCCR3B, CS30);   // set Timer prescaler to 1024
    clear(TCCR3B, CS31);   // set Timer  prescaler to 1024
    set(TCCR3B, CS32);   // set Timer  prescaler to 1024
    clear(DDRC, 7);      // set port C7 as input
    set(DDRB,1);        // set port B1 as output
    set(DDRB,2);        // set port B2 as output
    set(TCCR3B, ICES3);  // set input capture to look for rising edge
    TCNT3 = 0;           // initialize 16-bit timer
    // main infinite loop
    for (;;) {
        // match period to a frequency and turn on matching LED
         int period = find_period();
        if (period > 620 && period < 630) {
            // frequency is 25 Hz
```

```
            clear(PORTB, 1);
            set(PORTB, 2);          }
        else {
            clear(PORTB, 2);
            clear(PORTB, 1);        }
       if (period > 15 && period < 25) {
            // frequency is 662 Hz
            clear(PORTB,2);
            set(PORTB,1);}
        else {
            clear(PORTB, 2);
            clear(PORTB, 1);}
    }
    return 0;         }
```

2.4.1 Create an opamp and/or comparator circuit that can detect the 662 Hz signal even when overhead lights are shining on the sensor, using the same LED transmitting circuit from 2.3.1 (same LED and limited current). Show that the circuit can detect the signal when the transmitter is more than 1.5 meters away from the sensor. Submit a circuit schematic, and an image of the oscilloscope showing the output of the detection circuit.

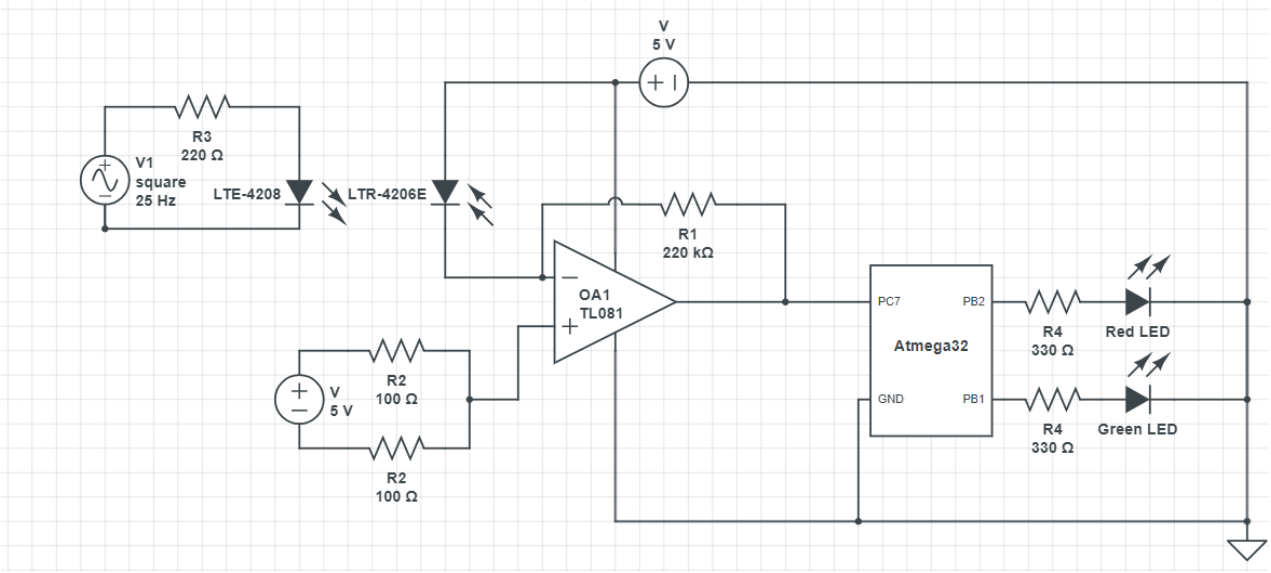


Fig 11 : Circuit diagram

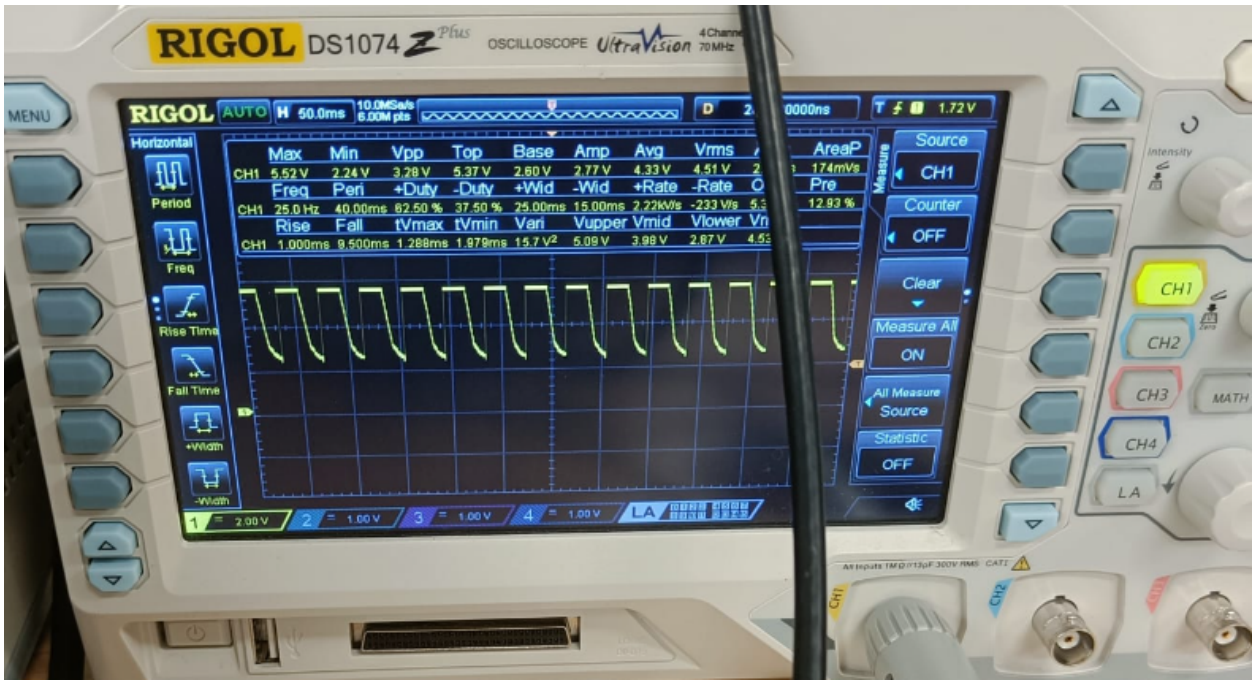


fig 12 : 25 Hz frequency

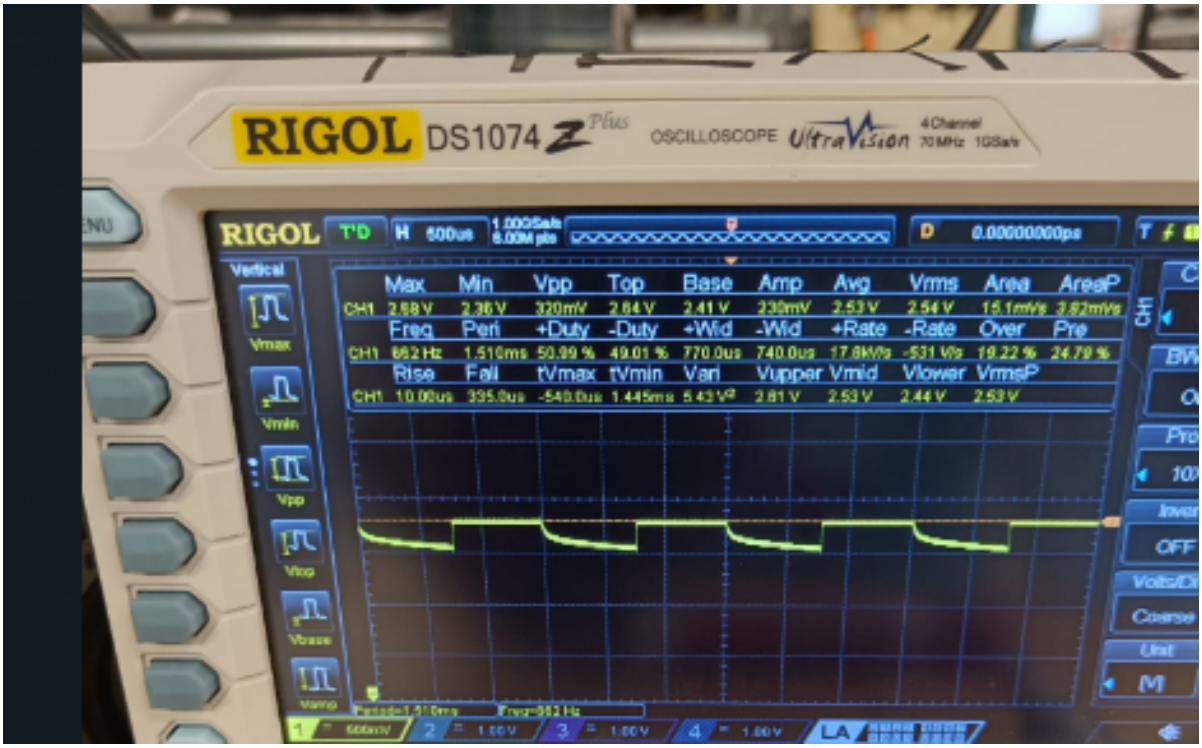


fig 13 : 662 Hz frequency

2.4.2 Create a circuit and code with the ATmega32U4 that can show when 25Hz and/or 662Hz frequencies are present and the transmitting LED is more than 1.5 meters away. At larger distances, aiming the LED and phototransistor at each other is important since they are both lensed. Use a cellphone camera (or borrow one if yours doesn't work) to be sure the LED lens is pointing directly at the phototransistor. With the camera at the point where the phototransistor will be the LED image should be brightest. Note also that your detection system should work even while the sensor is exposed to ambient light (e.g., roomlights or a flashlight). The detection can be indicated by different colored LED's as in 2.3.2. Get a check off from a TA showing detection under different conditions. Answer questions from the TA. Submit code and schematic.

Answer: The code and schematics are the same as 2.3.2 . Since the circuit was able to detect the light from 1.5 m in the same circuit as in from 2.3.2 there was no change in the code and the circuit.
Just the amplitude of the function generator was increased slightly so that it can be seen clearly.

Code:

```
/* Name: main.c 2.3.2
 * Author: Saurav Agrawal
 * Copyright: <insert your copyright message here>
 * License: <insert your license reference here>
 */
 #include "MEAM_general.h"
int find_period()
{    int pt;    // store time of previous rising edge
    int st = TCNT3;  // store start time

    set(TIFR3, ICF3);  // clear flag

    // wait for rising edge, exit if it takes too long
    while (!bit_is_set(TIFR3, ICF3)) {
    }
    int curr_time = ICR3;              // store time
    int diff = curr_time - pt;  // calculate time between edges
    pt = curr_time;                       // update previous time
    return diff;
}
int main(void)
{    _clockdivide(0);     // set clock speed to 16MHz
    set(TCCR3B, CS30);   // set Timer prescaler to 1024
```

```
  clear(TCCR3B, CS31);   // set Timer  prescaler to 1024
  set(TCCR3B, CS32);   // set Timer  prescaler to 1024
  clear(DDRC, 7);      // set port C7 as input
  set(DDRB,1);        // set port B1 as output
  set(DDRB,2);        // set port B2 as output
  set(TCCR3B, ICES3);  // set input capture to look for rising edge
  TCNT3 = 0;           // initialize 16-bit timer
  // main infinite loop
  for (;;) {
      // match period to a frequency and turn on matching LED
       int period = find_period();
      if (period > 620 && period < 630) {
          // frequency is 25 Hz
          clear(PORTB, 1);
          set(PORTB, 2);          }
       else {
          clear(PORTB, 2);
          clear(PORTB, 1);        }
      if (period > 15 && period < 25) {
          // frequency is 662 Hz
          clear(PORTB,2);
          set(PORTB,1);}
       else {
          clear(PORTB, 2);
          clear(PORTB, 1);}
  }
  return 0;        }
```

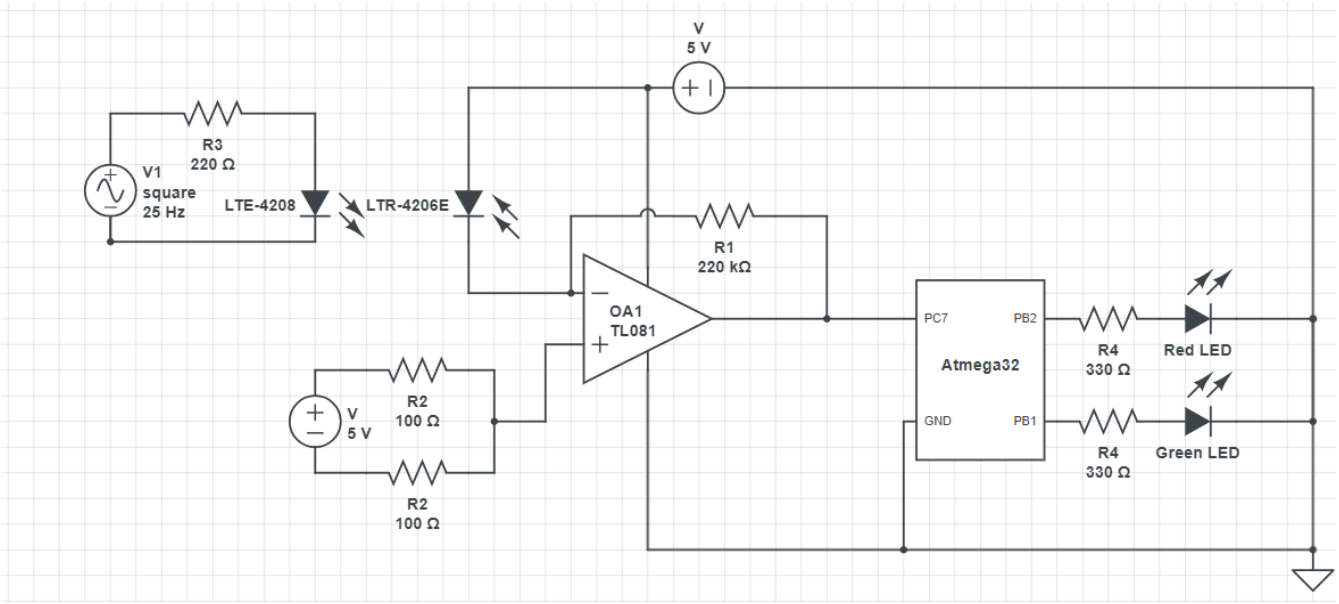


Fig 14 : Circuit diagram

Link to the videos for the questions :

https://www.youtube.com/watch?v=C-OnRrURPtg&list=PLI1MCD4tyefPJeC0Z3hVgUskmzLEJdpnw

(this also includes the video for question 2.4.2 for 1.5 m)