CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

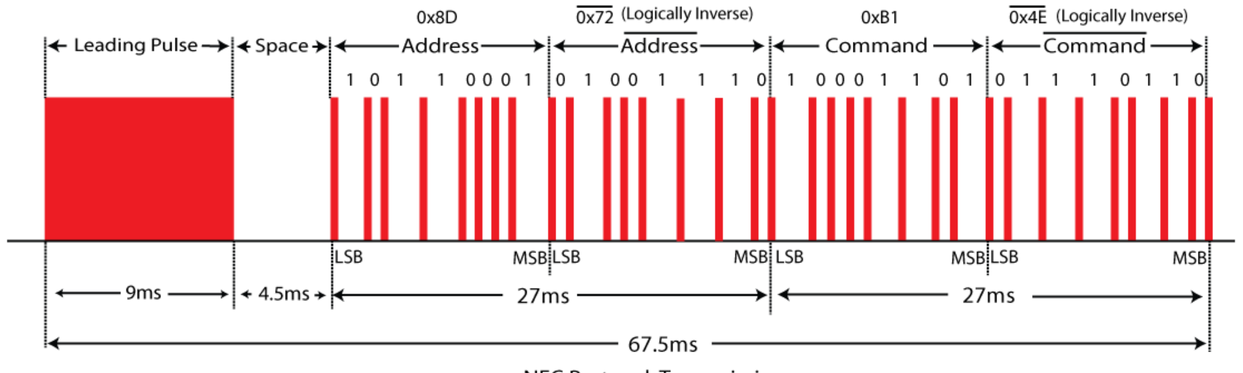
ECE 3301L Spring 2019 Microcontroller Lab Felix Pinai
Session 2

LAB 12: Interface to Infra Red Remote Control

The purpose of this lab is to introduce the student to the technique interfacing to an Infra Red Remote Control using an edge-triggered External interrupt along with a timer.

Lab:

Below is a basic stream of pulse when a key on a remote control is pressed:



NEC Protocol Transmission www.circuitvalley.com

This is based on the NEC protocol transmission. Here are the basic steps on that protocol:

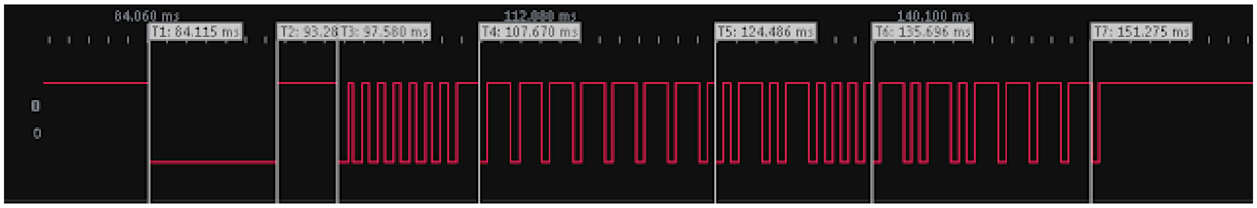
- A leading burst of pulses that lasts up 9 msec. That is the Leading Pulse.
- A space period that lasts 4.5 msec.
- An 8-bit address field for the receiving device
- An 8-bit address inverse of the address
- An 8-bit command field
- An 8-bit inverse of the command field

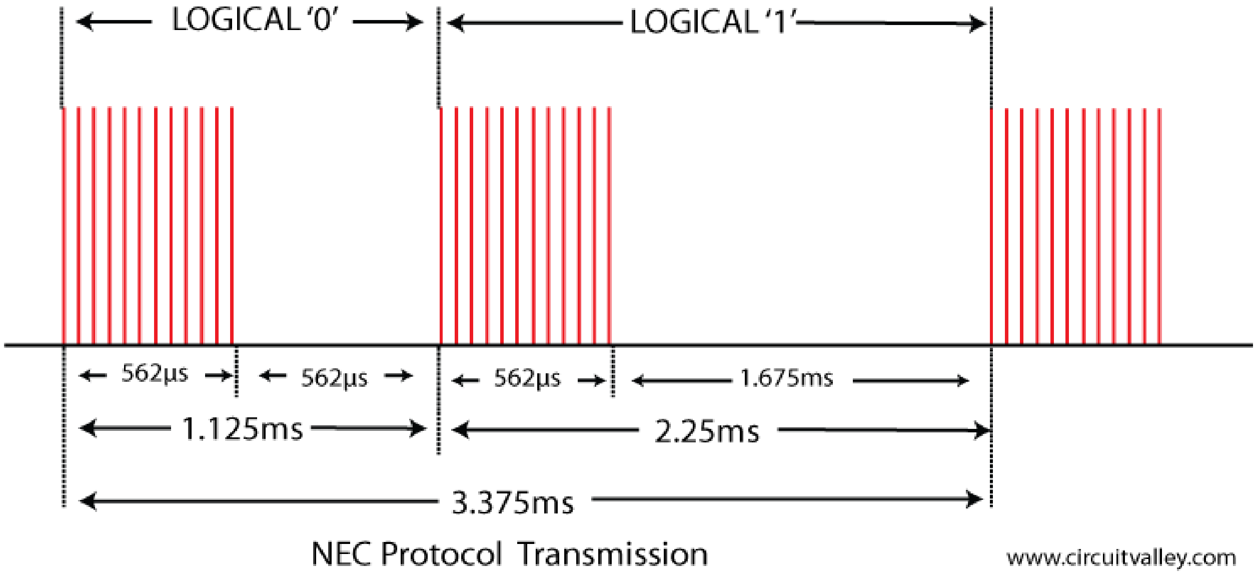
A total of 32 bits should be received after the ‘Space 4.5 msec’ field has been detected. To differentiate for each bit between the logic ‘0’ or the logic ‘1’, the following detection should be applied:

- A logic ‘0’ should be detected when a 562-usec pulse burst is followed by a space period of 562 usec.
- A logic ‘1’ should be detected when a 562-usec pulse burst is followed by a space of 1.675 msec

When the Infrared signal is received by an IR receiver, it will remove the basic modulation frequency from the input signal to only output the ‘envelop’ from that signal. Here is the general sequence of the output (see picture below):

- The output stays at logic ‘1’ when no pulse is received
- The output goes to ‘0’ when the leading pulse is detected. The output should stay at logic ‘0’ for 9 msec.
- When the ‘Space’ is asserted, the output changes to logic ‘1’ for the period of 4.5 msec.
- Next a sequence of 32 bits should be transmitted. When the output changes from ‘1’ to ‘0’ and then back to ‘1’ with the same width of 562 usec, this will indicate that a data logic ‘0’ is asserted. Otherwise, if the output goes from ‘1’ to ‘0’ for a period of 562 usec and then switches from ‘0’ to ‘1’ after a period of 1.675 msec, then a data logic ‘1’ is asserted.





To implement this protocol, we will need to use the Timer for the purpose to measure the elapsed time between two events and the input pin that has the edge-triggered interrupt feature (the INTx pins). Let us assume that we are going to use the INT2 pin to connect to the output of the IR receiver. That output is normally sitting on the logic ‘1’ when no key is pressed on the remote control. A variable ‘nec_state’ will be used and its initial value should be set to ‘0’. That is the first state of the IR decoding sequence. The program should use the pin INT2 to detect the different transitions of the IR signals and update the value of ‘nec_state’ to keep track of the progress of the sequence. At the initial point, INT2 must be programmed to interrupt when the IR output transitions from high to low (to handle the case of the leading pulse)

When an INT2 interrupt occurs, the first task is to read the content of the timer and save into a variable. The next step is to determine to update the state of ‘nec_state’ depending on the time measured.

```
void INT2_ISR():

    • Read Timer1 and save content
    • Clear Timer1 count registers

switch  (nec_state):

case 0:                     // This is to handle first detection of Leading Pulse

    • Clear Timer 1
    • Program Timer1 mode with count = 1usec using System clock running at 8Mhz
    • Enable Timer 1
    • Force bit count (bit_count) to 0
    • Set nec_code = 0
    • Change Edge interrupt of INT2 to Low to High to wait for the end of the Leading Pulse
    • Set nec_state to state 1 and return

case 1:                     // This is to handle the end of the Leading Pulse
                            // and start of Space

    • Check if Timer1 value read is between 8500 usec and 9500 usec
    • If not, force nec_state = 0
    • If yes, change Edge interrupt of INT2 to High to Low to handle end of Space
    • Set nec_state to state 2 and return


case 2:                     // This is to handle the end of Space

    • Check if Timer1 value read is between 4000 usec and 5000 usec
```

- If not, force nec_state = 0
- If yes, change Edge interrupt of INT2 to Low to High to handle the start of bit detection
- Set nec_state to state 3 and return

case 3:   // This is the start of the loop to detect the bit of the data sequence

- Check if Timer1 value read is between 400 usec and 700 usec
- If not, force nec_state = 0
- If yes, change Edge interrupt of INT2 to Low to High
- Set nec_state to state 4 and return

case 4:

- Check if Timer1 value read is between 400 usec and 1800 usec
- If not, force nec_state = 0
- If yes, shift left nec_code by 1 bit
- Check if Timer1 is greater than 1000 usec
- If yes, add 1 to nec_code
- Increment bit_count by 1
- Check if bit_count > 31
- If yes, set nec_ok flag to 1 and set INT2IE = 0
- Else, change Edge interrupt of INT2 to High to Low. And set nec_state to state 3

Below are some initial codes to be used for this exercise.

```
#define _XTAL_FREQ      8000000

void TIMER1_isr(void);
void INT0_isr(void);
unsigned char nec_state = 0;
unsigned char i;
short nec_ok = 0;
unsigned long long nec_code;

void putch (char c)
{
   while (!TRMT);
   TXREG = c;
}
```

```
void init_UART()
{
        OpenUSART (USART_TX_INT_OFF & USART_RX_INT_OFF &
USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
USART_BRGH_HIGH, 25);
        OSCCON = 0x70;
}

void interrupt high_priority chkisr()
{
   if (PIR1bits.TMR1IF == 1) TIMER1_isr();
   if (INTCON3bits.INT2IF == 1) INT2_isr();
}

void TIMER1_isr(void)
{
  nec_state = 0;                         // Reset decoding process
  INTCON2its.INTEDG2 = 0                 // Edge programming for INT2 falling edge
  T1CONbits.TMR1ON=0;                    // Disable T1 Timer
  PIR1bits.TMR1IF=0;                     // Clear interrupt flag
}

void INT2_isr(void)
{
 unsigned int timer;
        INTCON3bits.INT2IF = 0;                  // Clear external interrupt

         if (nec_state != 0)
        {
         timer = (TMR1H << 8) | TMR1L;        // Store Timer1 value
        TMR1H = 0;                            // Reset Timer1
        TMR1L = 0;
         }
        switch(nec_state)
         {
         case 0 :                        // Add your code here based on the provided info

        }
}
```

```
void main()
{
        init_UART();

        OSCCON = 0x70;                  // 8 Mhz
        nRBPU = 0;                      // Enable PORTB internal pull up resistor
        TRISA = 0x01;
        TRISB = 0x07;
        TRISC = 0x01;                   //
        TRISD = 0x00;
        TRISe = 0x00;
        ADCON1 = 0x0F;                   //

         INTCON3bits.INT2IF = 0;         // Clear external interrupt
        INTCON2bits.INTEDG2 = 0;        // Edge programming for INT0 falling edge H to L
        INTCON3bits.INT2IE = 1;         // Enable external interrupt

        TMR1H = 0;                       // Reset Timer1
        TMR1L = 0;                      //
        PIR1bits.TMR1IF = 0;             // Clear timer 1 interrupt flag
        PIE1bits.TMR1IE = 1;             // Enable Timer 1 interrupt

         INTCONbits.PEIE = 1;            // Enable Peripheral interrupt
        INTCONbits.GIE = 1;              // Enable global interrupts
        nec_ok = 0;                     // Clear flag
         nec_code = 0x0;                // Clear code

        while(1)
        {
                if (nec_ok == 1)
                {
                         nec_ok = 0;
                         printf ("NEC_Code = %08lx \r\n", nec_code);
                        INTCON3bits.INT2IE = 1;      // Enable external interrupt
                        INTCON2bits.INTEDG2 = 0; // Edge programming for INT0 falling edge
                }
        }
}
```

TASK A)

Your team will be provided with an IR Remote control. Implement the program above and use the Tera-Term to capture the IR code for each key pressed. In particular, use your table number assignment below to know what column of keys your team should record the value of every key:

Tables 1 – 5: CH- and CH+ keys

Tables 6 – 10: '-' and '+' keys

Tables 11 – 15: '<<' and '>>' keys

Use the printf statement with the print format %08lx in order to print the 8-digit number code in hex. Here are some values for some of the keys:

```
#define CH-          0x00ffa25d
#define Arrow_Left   0x00ff22dd
```

The differences on the connections of this lab versus the previous lab should be the three additional wires to connect the three pins of the TSOP38336 IR receiver respectively to the output (pin 1), ground (pin 2) and +5V (pin 3). Here is the link to the datasheet of that receiver.

http://www.vishay.com/docs/81743/tsop381.pdf

TASK B)

Take the implementation from last week's lab and merge the code of this week's lab to use the remote control to vary the speed of the fan. Use of the UP and DOWN arrow keys of the remote control for the purpose of speed control. Based on your table's number, use the proper colored UP and DOWN keys. The UP and DOWN push button switches from last week's lab should remain operational.