CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

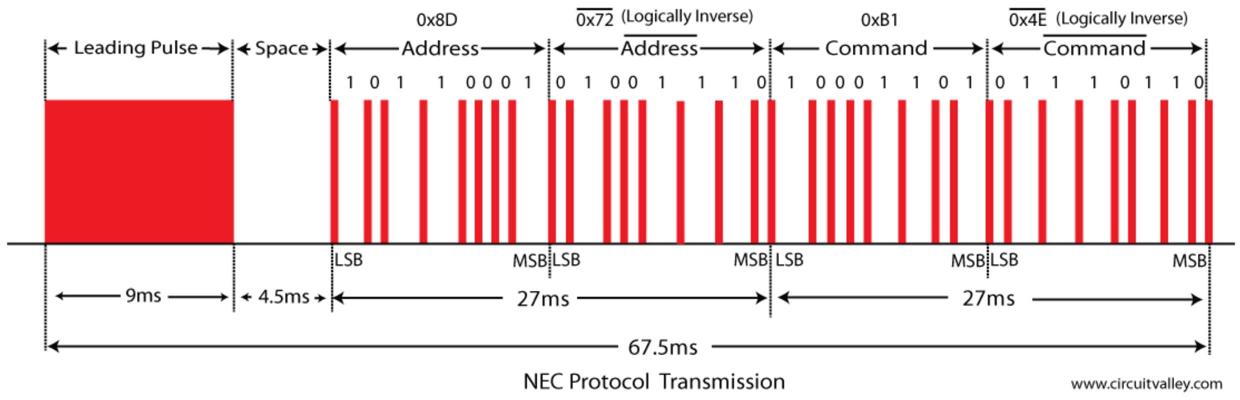
ECE 3301L Spring 2024 Microcontroller Lab Felix Pinai
Session 1

LAB 10: Interface to Infra Red Remote Control using External Interrupts

The purpose of this lab is to introduce the student to the technique to interface to an Infra Red Remote Control.

Lab:

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


0x8D
$\overline{0x72}$ (Logically Inverse)
0xB1
$\overline{0x4E}$ (Logically Inverse)
Leading Pulse
Space
Address
$\overline{\text{Address}}$
Command
$\overline{\text{Command}}$
1 0 1 1 0 0 0 1
0 1 0 0 1 1 1 0
1 0 0 0 1 1 0 1
0 1 1 1 0 1 1 0
LSB
MSB
9ms
4.5ms
27ms
27ms
67.5ms
NEC Protocol Transmission
www.circuitvalley.com

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

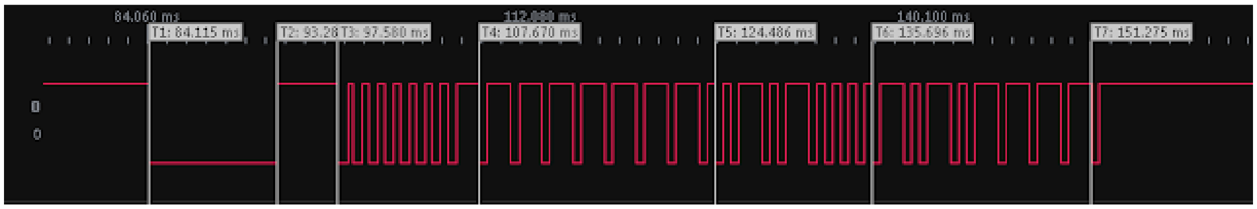
- A leading burst of pulses that lasts up 9 msec.
- A space period that lasts 4.5 msec.
- An 8-bit address field for the receiving device
- An 8-bit address inverse of the address
- An 8-bit command field
- An 8-bit inverse of the command field

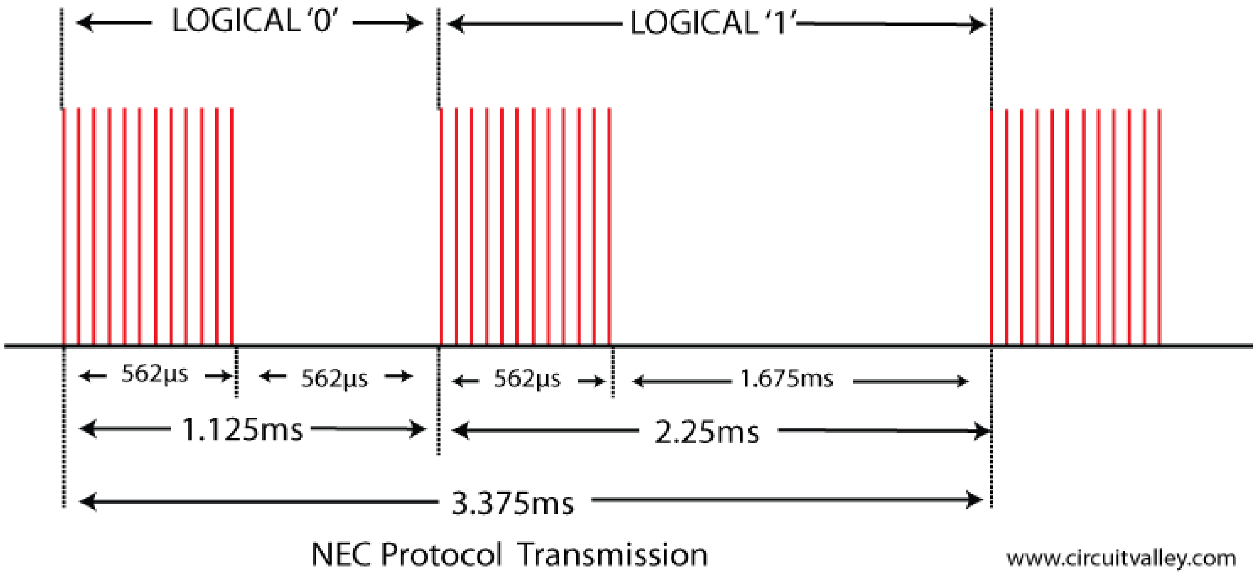
A total of 32 bits should be received after the 'Space 4.5 msec' field has been detected. To differentiate for each bit between the logic '0' or the logic '1', the following detection should be applied:

- 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, the output of that receiver should provide:

- 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, then 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’ for a period of 1.675 msec, then a data logic ‘1’ is asserted.





PART 1)

Refer to the provided schematics.

Use the supplied IR receiver. Connect the power and ground pins as shown on the schematics. The circle shown on the IR receiver is to indicate the half dome side of that part. Place that side on the front end. Connect the output of the receiver to the External INT pin shown on the schematics.

Use the provided remote control to do IR implementation described below.

Use the logic analyzer to capture the waveform of the IR output. Press on any button on the remote control. Set the analyzer on trigger mode to better capture the output.

Once a capture is obtained, verify the following time measurements:

- Leading pulse – should be around 9 msec
- Space – should be about 4.5 msec
- We should have a string of 32 pulses with the logic 1 either 562 usec or 1.675 msec long and the logic 0 at 562 usec long

Take a screen shot of the capture for the report.

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 INT1 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' should 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 INT1 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, INT1 should be programmed to interrupt when the IR output transitions from high to low.

When an INT1 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'.

void INT1_ISR():

- If variable 'Nec_State' is not in state 0 then read the 16-bit from TMR1H and TMR1L to store into variable 'Time_Elapsed'. Clear both registersTMR1H and TMR1L
- Else, perform a switch statement on variable 'Nec_State':

  Case 0:

  - Clear registers TMR1H and TMR1L

- Program Timer 1 mode with count = 1usec using System clock running at 8Mhz
- Enable Timer 1
- Force variable ‘bit count’ to 0
- Set variable ‘Nec_code’ to 0
- Set variable ‘Nec_State’ to 1
- Output value of ‘Nec_State’ to PORT D bits 0-2 to show on the logic analyzer the value of the state machine ‘Nec_State’
- Program Edge interrupt of INT1 to Low to High
- Return

Case 1:

- Check if ‘Time_Elapsed’ has a read value between 8500 usec and 9500 usec
- If yes,
    - force ‘Nec_State’ to state 2.
    - Output of PORTD bits 0-2 the value of ‘Nec_State’
    - Change Edge interrupt of INT1 to High to Low

- If not, we have an error. call function ‘Reset_Nec_State()’ to restart
- Return

Case 2:

- Check if ‘Time_Elapsed’ value read is between 4000 usec and 5000 usec
- If yes,
    - force ‘Nec_State’ to state 3.
    - Output of PORTD bits 0-2 the value of ‘Nec_State’

- If not, do call function ‘Reset_Nec_State()’
- Change Edge interrupt of INT1 to Low to High
- Return

Case 3:

- Check if ‘Time_Elapsed’ value read is between 400 usec and 700 usec
- If yes,
    - force ‘Nec_State’ to state 4.
    - Output of PORTD bits 0-2 the value of ‘Nec_State’

- If not, do call function ‘Reset_Nec_State()’
- Change Edge interrupt of INT1 to High to Low
- Return

Case 4:

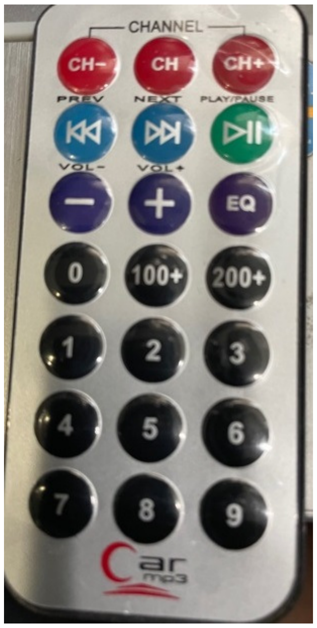
- Check if ‘Time_Elapsed’ value read is between 400 usec and 1800 usec
- If yes,
    - shift left ‘Nec_code’ by 1 bit and store new value back to ‘Nec_code’
    - Check if ‘Time_Elapsed’ is greater than 1000 usec
    - If yes, Increment ‘Nec_code’ by 1
    - If not, do nothing
    - Increment variable ‘bit_count’ by 1

    - Check if ‘bit_count’ > 31
    - If yes:
        - set variable ‘Nec_Button’ to be ‘Nec_code’ shifted right by 8 bits
        - set ‘Nec_State’ to 0
        - Output ‘Nec_code’ to PORTD bits 2-0
        - set ‘Nec_OK’ flag to 1
        - set INT1IE = 0

    - If not:
        - set ‘Nec_State’ to state 3.
        - Output ‘Nec_code’ to PORTD bits 2-0
- If not, do call function ‘Reset_Nec_State()’
- Change Edge interrupt of INT1 to Low to High
- Return

See the attached sample ‘S23_Lab10_S1_sample.c’ file to get started.

In the main routine, the variable ‘Nec_OK’ will be constantly checked. When it is set to 1, then a button on the remote has been detected. The TeraTerm console should show on the screen the **value** of the ‘Nec_Button’ of the key just being pressed.



If you have issues not being able to get the right code, use the logic analyzer to capture the output on PORTD bits 0-2. Use those bits to decode the state machine of your implementation and to determine anything wrong with the code.

When successful, press all the buttons and collect all the codes and then place them consecutively into an array from the order of the buttons on the remote control. See example of ‘**array1**’ in the sample code. Add up to 21 entries in that array.

PART 2)

Once the ‘array1’ is filled, the next step is to create another array called **‘txt1’** with the text values of each button. The text should be 3 characters long terminated with a ‘\0’ to end the string. See the sample code. Make sure to make the string to be **3 characters long**. If the button only has one number, then place a space in front of the number followed by the number and then add another space after that number.

A third array called **‘color’** should reflect the color of each button. Look at the color of each button on the remote. Use the predefined variable shown on the sample code to fill all the 21 entries in this array.

The last task is to output the color of the button onto one of the three RGB LEDs D1, D2, and D3 based on the position that button is located on the remote control. Here are the requirements:

a) **The first 6 buttons (from the top ‘CH+’ to the ‘>>|’) of the remote are assigned to RGB LED D1.**
b) **The next 6 buttons (from ‘-‘ to ‘200+’) of the remote are assigned to RGB LED D2.**
c) **The last 9 buttons (from ‘1’ to ‘9’) of the remote are assigned to RGB LED D3.**

Don’t use if or case statement to select the color for the output.

Use additional arrays to store the color of the button being pressed. Determine the hex value needed to turn on the color for that LED based on the key being pressed. In the case that the LED should be off, then use the value of 0 in order not to turn on that LED. Create an array and store all the values required for that LED.

If a RGB LED uses the same port with another RGB LED, then find the combined hex value to control both LEDs. Create the array that will contain the combined values.

After the arrays are created, simply use the position of the key being pressed with respect to the remote control as the offset to the array. Obtain the hex value from the array and simply output to the associated port to the RGB LED(s).

If a LED is on the same port with another piece of hardware, do the masking process to protect the other hardware bits and then do an OR operation using the value of the array and output to the port.

Press all the buttons on the remote and make sure of the following actions:

1) The proper RGB LED should be turned on and stays on (until the next key is pressed). Only one RGB LED should be on.
2) The LED 'Key Pressed' must be turned on and stays on.
3) Next, a beep should be generated by calling the routine 'Activate_Buzzer()' followed by the routine 'Wait_One_Sec()' and then the function 'Deactivate_Buzzer()' is called to shut off the beep.
4) Turn off the LED 'Key Pressed'.

When complete, demonstrate to the instructor by pressing all the keys on the remote. The LCD should display on the screen the information of the pressed button and its color.