The Sony IR Protocol and PIC BASIC

Mini-Sumo Robotics Project
Worksheet #11b

Credits: Much of the material regarding the Sony IR protocol was provided to me by Ian Mathie and Paul Wytenbroek of BCIT. There is also an excellent web site on IR technology at: http://www.xs4all.nl/~sbp/knowledge/ir/ir.htm

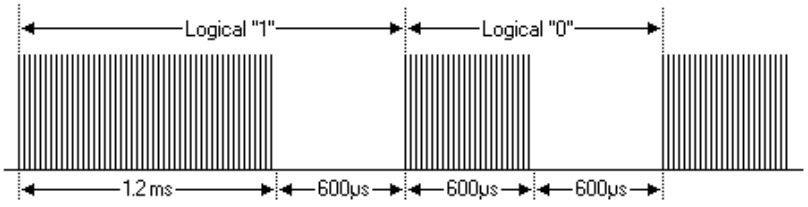
Introduction

If you have been following the worksheets to this point, you should now have your robot responding to an IR signal from a standard infrared remote control. In this worksheet we will look at how to interpret the signal from that remote in order to make your robot perform a variety of tasks. To simplify this process we will move from assembly language, to PIC BASIC, a compiled language.

Fundamentals

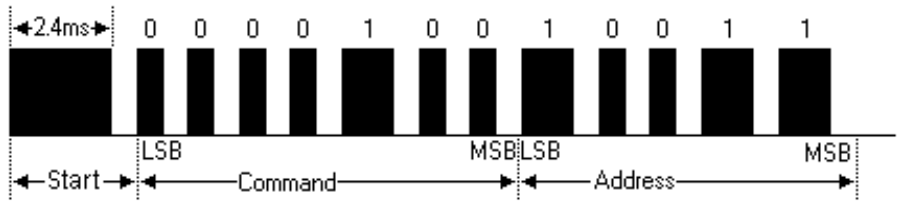
As we learned in the previous worksheet, an IR remote control transmits light at a wavelength of around 1000nm (one one-thousandth of a millimetre), a wavelength that although it is invisible to our eyes, is quite visible to video cameras. If you ever need to check to see if a remote is working, look at it through a video camera. We also learned that the IR signal is pulsed on and off, or "modulated" at a rate of 38,000 times per second or 38 kilohertz (kHz) in order to help the receiver distinguish the IR signal from "background" IR radiation. (The sun, and incandescent light bulbs among other sources also produce 1000nm IR radiation, however they do not modulate it.)

The IR remote will further modulate that 38kHz signal in order to transmit a binary number that can be read by the IR receiver. In the case of the Sony IR protocol (known as SIRC), a 1.2 millisecond (ms) burst of 38kHz IR indicates a "1", while a 0.6 ms burst represents a "0". There is a 0.6 ms pause between bits. A series of ones and zeros looks something like this:



from: http://www.xs4all.nl/~sbp/knowledge/ir/ir.htm

While it is possible to develop a communications protocol using only ones and zeros, it makes things much easier if you know when each set of data starts. To achieve this, there is a special "start bit", represented by a 2.4 ms long pulse. When the receiver senses this long pulse of IR, it knows to expect 12 bits of data to follow in very short order. The entire signal looks something like this:



from: http://www.xs4all.nl/~sbp/knowledge/ir/ir.htm (and corrected)

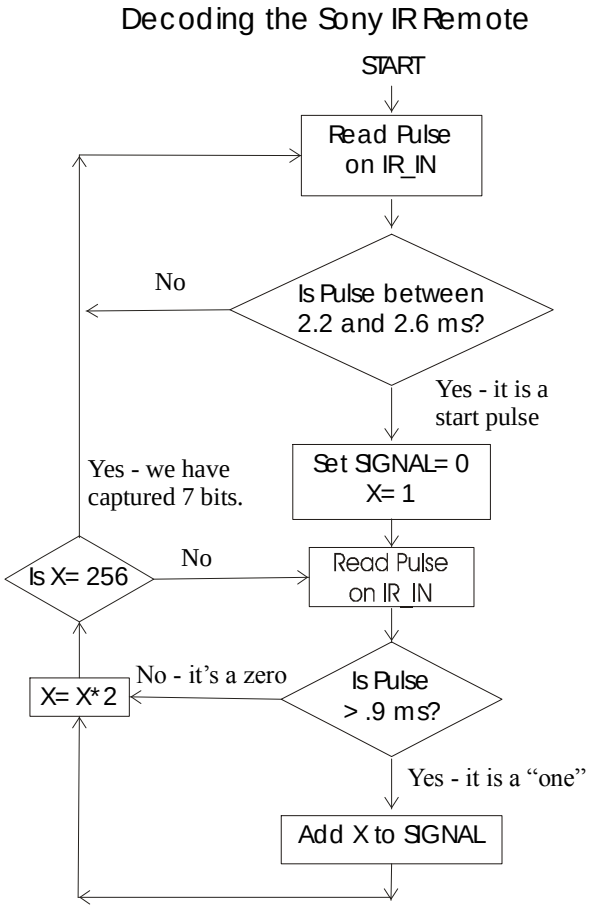
This represents a 12-bit data "word". The first seven bits of the word represent the command that is being sent, while the next five bits tell the receiver the "address" where the command is to be sent. Typically the address would tell a Sony IR receiver whether the command was meant for your television, your VCR, or your DVD player, while the address would carry information regarding which button you pressed.

You will notice the terms "LSB" and "MSB" on the diagram. LSB means "Least Signficant Bit" and MSB means "Most Significant Bit". Typically when we write a binary number, the most significant bit is the first one on the left, and the LSB is the last one to the right. In other words, the seven-bit command sequence transmitted here is actually "0010000", representing a decimal value of 16. This means that you are holding the "channel up" button.

Reading the Signal

Since the PNA4602M IR receiver is designed to de-modulate the 38kHz IR signal, and simply makes it's output go "low" when the signal is being received, we simply need a way to read the duration of each signal. Although timing a signal that lasts less than one one-thousandth of a second might be difficult for us humans, it's easy work for a PIC. The command we use is called "pulsin" (short for "pulse input"). By attaching the output of the IR receiver to a pin on the PIC, and executing the pulsin command we can time the length of the pulse, store the result in a memory location, and then use that "variable" memory location with an "if… then" statement to allow the PIC to make decisions. The correct syntax for the pulsin command is given in the PIC Basic manual.

A flow chart for reading the first seven bits of the Sony IR remote signal is shown to the right. If you were to use the flow chart to analyse the binary signal shown at the top of



page two, the top loop of the flow chart would wait for the 2.4ms long start pulse to be received. (We say we'll accept anything between 2.2 and 2.6 ms just to make sure that we don't miss a signal due to a small timing error.) This signal will come from the IR receiver, which will be attached to a pin on the PIC called "IR_IN". (This can be any pin, so long as we tell PIC BASIC which one it is at the start of the program.) Once the start pulse is received, then "Signal" will be set to zero, and "X" will be set to one. Signal is the variable where we will store the value of the incoming code, while X is a variable that will keep track of which bit in Signal needs to be changed.

At this point we will once again measure a pulse on IR_IN. Looking at the input at the top of page 2, we see that this pulse will be a "zero", so it will be about 0.6 ms long. Since this is less than 0.9 ms, we will not add anything to Signal, will double X (to 2), and loop back to read the pulse length on IR_IN again. This will happen three more times, with X continuing to double each time… 1,2,4,8 etc. until when reading the fifth bit we get a value of "1". This means that the pulse will have been about 1.2 ms, which is greater than 0.9 ms, which means that X will be added to signal. At this point the value of X should be 16, giving us a value of 16 for Signal. This cycle repeats seven times, after which X is equal to 256, and the program returns to the top to await another start pulse.

Although this flow chart does not use the received value to actually DO anything, the code on the next page will implement this flow chart, and POKE the first five bits of the resulting value to PORTA. If you attach LEDs to the outputs from PORTA, you should be able to decode the values of many of the IR remote command buttons.

Now that you can read your IR remote control unit, how could you modify the program below to control the motors on your gear box?

What other things could you make your mini-sumo bot do?

How could you devise an output system so that you could decode all seven bits of the command code? How could you decode the 5 bits of the address code?

Could you use this knowledge to decipher the codes used by other types of remote controls?

Some remote controls do not work with this set up at all. Why do you think this might be? What might they do differently?

 dtengineering@yahoo.ca

```
'IR Decode -- J. Brett, October 2003

'A Pic Basic program that will read the first seven bits of a SONY IR remote
' control signal, and POKE the lower five bits of that value to PORTA

'These Pin-outs can be changed to match your design
Symbol      LED1 = 7            'A simple signalling LED is attached to RB7
Symbol      IR_IN = 1           'The 4602 IR sensor is attached to RB1
Symbol      PORTA = 5           'PORTA controls whether the pins on PORTA are High or Low
Symbol      TRISA  = $85        'TRISA controls whether the pins on PORTA are Inputs or Outputs

'These variables should not need to be changed
Symbol      x= b7                       'a byte variable for the adding routine
Symbol      signal = B5                 'a byte variable for the signal code from the IR remote
Symbol      pulse = W1                  'the length of the IR pulses
                                        ' we use a word variable becuase the length of the pulse
                                        ' may exceed a value of 255 because this is a long pulse

'I always like to have a very simple start up routine.... it helps with troubleshooting
'This Start Up routine flashes LED1 and beeps to let you know things are working
High LED1                               'Signal LED ON
Pause 500                               'wait half a second
Low LED1                                ' turn off the Signal LED
Poke TRISA,0                        'set all PORTA pins to outputs

'The main program begins here
'-------------------------------------------------------------------------------
START:
PulsIn IR_IN,0,pulse                    'read the pulse length from the IR_IN pin
IF pulse < 220 OR pulse > 260 Then START 'if it's not a 2.4ms start bit (or pretty close to it)
                                        ' go back and try again

'aha... it was a start bit, so now we record the next seven pulses to be sent
signal=0                                'set the signal variable back to zero
x=1                                     'set X back to one

LOOP2:
PulsIn IR_IN,0,pulse
IF pulse >90 Then ADD_X                 'this bit is a "one", so set it
READ_IR:                                'a place to come back to
x=x*2
IF x<256 Then LOOP2                     'keep going until we have seven bits

Poke PORTA,signal                       'show the value of the lower five bits of signal on PORTA

GoTo START                              'and go do it all over again


'The following statements work in conjunction with the earlier if...then statements to add the correct
' value to signal when bits are determined to be "on".

ADD_X:
signal=signal+x
GoTo READ_IR

End
```