PIC-based Programmable LED

Scott Hendrickson
Brighton, CO 80601
E-mail: ss@drskippy.net

Abstract

This is a *How-To* for constructing a PIC-based programmable LED. Following Alex Weber's post on *Instructables* on 6 March 2007, I created a programmable LED on a PIC 12F683. The recorded sequence is saved to EEPROM, so it is retained when power is removed from the PIC. The software includes input codes for playing a stored sequence or recording a new sequence. Also included is a description of a simple SOIC chip package prototyping board.

1 Introduction

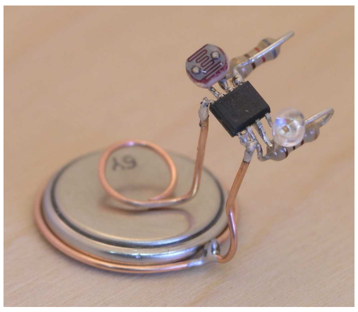

Figure 1: PIC 12F683 Recordable LED.

When I saw Alex Weber's AVR Programmable LED[1], I knew I had to build one for myself. It was immediately clear to me (and to others, see the comments on the *Instructables* post) that this was a great building block for playing with collective behaviors. Once the basic unit is created, it can be replicated quickly. Each unit has input and output and the use of the LDR-visible LED combination for makes it possible for units to communicate and humans to watch.

More recently, Alex has moved this idea along with his "Synchronized Fireflies" post on 6 April 2007[2]. This is a great application!

I used the 12F683 because it has relatively generous memory and both comparators and ADC. An 8-pin package works well because we need only on LED output and a sensor input. The extra pins let us control power to the sensor circuit so we can turn it on or off to conserve power or to ignore input. The 12F683 runs at up to 8 MHz on the internal clock, but 4 MHz is more than enough for our purposes and uses less power.

The code included here stores the recorded sequence in EEPROM on the chip. This allows the sequence to be retained when the chip is powered. The code also include a small code interpretation section allowing the user to play back, record, loop the sequence or put the 12F683 to sleep.

2 The Circuit

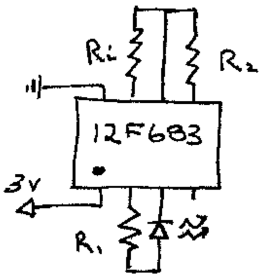

Figure 2: Schematic for Recordable LED.

The circuit used for the Recordable LED closely follows Alex's. See Fig. 2. R_1 is chosen to limit the current through the LED. This circuit uses a 1/4 watt 100 Ω resistor.

R_2 is used to create a voltage divider with the R_i, the LDR, for the sensor input. Choose R_2 depending on your LDR, and the desired sensitivity threshold of the sensor. The circuit show uses $R_2 = 820\ \Omega$. This Recordable LED uses the comparator (instead of Alex's ADC) using the PIC's internal voltage reference for the detector circuit. Just pick a resistance that is closed to a kΩ and play with the threshold voltage by setting V_{ref}.

2.1 Prototype Using The SOIC Package

Obviously this chip package is very small and a little difficult to work with in prototyping. However, it is easy to build a breakout board. Using a Radio Shack DIP board, some headers and a piece of acrylic milled with a Dremel, it was easy to build the simple circuit and set up in-circuit programming at the same time. Four machine screws with nuts on them hold the chip in place by placing downward pressure on the top of the chip. Lightly finger-tightening the nuts held the PIC in place for prototyping. This made development of the software much quicker as there was no need to remove the chip and connect parts every time it was reprogrammed.

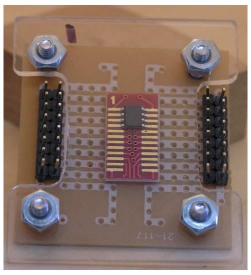

Figure 3: SOIC Prototyping breakout board.

During prototyping, use some simple jumpers to the headers to connect the LED, resistors and LDR and power the circuit with the PICKit2. The PICKit2 even has the capability of varying the supply voltage so it was possible to test the circuit at the planned $V_{dd} = 3V$.

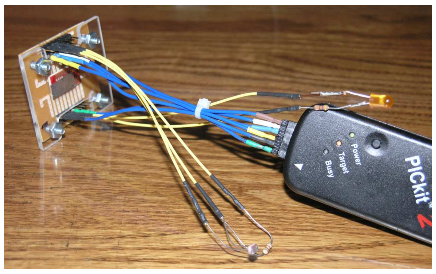

Figure 4: Prototyping the Recordable LED with the breakout board. The PICKit2 is uses for in-circuit programming.

Light Pulses	Action
1	Record a new sequence, play it back immediately, return to command mode
2	Play back stored sequence, return to command mode
3	Play stored sequence indefinitely
Any Other	Put processor to in sleep state

Table 1: Programmable LED commands.

2.2 C Code

The code below was written for the CCS PCW compiler [3]. Because there is a lot of room on the chip and this application doesn't take much EEPROM or program memory, the code was written quick and dirty.

The sequence storage routine is particularly inefficient. It stores the state of the LED at each time step in the first bit of each byte of EEPROM. With onboard EEPROM compactly of 256 bytes, there is plenty of space for a very long sequence. In order to store a very long or additional sequences, one could spend a little time writing a storage routine that packs the bits into bytes for storage.

Another trade-off is made with the command sequence. To use the same recording function as the sequence uses, we just use the top 32 bytes of EEPROM to store the command sequence (calling *record()* with mode *COM*). This saves writing another routine to store command sequences in volatile memory, but also needlessly uses up some of the EEPROM.

Setting V_{ref} is a matter of setting a range bit and value bits in a control register. The CSS compiler makes this very easy with the command `setup`$_\mathtt{v}$`ref(VREF_LOW|15);`. `VREF_LOW` indicates means we are using the low range of voltage reference where,

$$V_{ref} = \frac{VR<3:0>V_{dd}}{24} = \frac{<1111>V_{dd}}{24}. \quad (1)$$

The command decoder counts transitions from light to dark or dark to light and returns this half this number. The commands are shown in the table below.

Looking forward, there should be plenty of room on the chip to create a simple neural network to store many sequences. In this scenario, imputing a sequence that is close to one of the stored sequences would result in the matching output sequence.

Here is a complete software listing. The C file is also available for download at my Web site `http://www.drskippy.com/PIC/LEDRec.c`. If you use a different LDR or want to make customizations you should get this. (You may need to configure the internal voltage reference to use the “HIGH” range.) In you want to use the standard build, the HEX file is also available for download at the same URL, `.../PIC/LEDRec.hex`.

```
#include "W:\CCS_Projects\LEDRec\LEDRec.h"

#define RECORD_END 0x47
#define RECORD_DELAY 150
#define COMMAND_START 0xE0
#define COMMAND_LEN 31
#define COM TRUE
#define REC FALSE

void led_on(void)
{
   output_high(PIN_A5);
   output_low(PIN_A4);
}

void led_off(VOID)
{
   output_low(PIN_A4);
   output_low(PIN_A5);
}

void sensor_on(void)
{
   output_high(PIN_A0);
   output_low(PIN_A2);
}

void sensor_off(void)
{
   output_low(PIN_A0);
   output_low(PIN_A2);
}

void led_blink(int n)
{
   int j;
   for (j=0;j<n;j++)
   {
      delay_ms (75);
      led_on ();
      delay_ms (150);
      led_off ();
      delay_ms (75);
   }
}

void playback(void) {
    char ee_addr;
    char dat;

    ee_addr = 0x00;
    led_off();

    while (ee_addr < RECORD_END) {
        dat = read_eeprom(++ee_addr);
        if (dat == 0x00) led_on();
        else led_off();
        delay_ms(RECORD_DELAY);
    }
}

void record (short mode) {
   short state;
   long max;
   char ee_addr,dat;

   state = FALSE;
   if (mode == REC){
      ee_addr = 0x00;
      max = RECORD_END;
   }
   else {
      ee_addr = COMMAND_START - 1;
      max = COMMAND_START + COMMAND_LEN;
      if (max > 0XFF) led_blink(10);
   }
   sensor_on();
   delay_ms(10); //to get comp ready for first read

   while (ee_addr < max) {
      dat = state^C1OUT;
      write_eeprom(++ee_addr,dat);
      delay_ms(RECORD_DELAY);
   }
   sensor_off();

   // indicate record in memory at startup
   if (mode == REC) write_eeprom(0x00,0x99);
}

int decode_command(void){
   char ee_addr;
   char new, prev;
   int n_changes;

   n_changes = 0;
   ee_addr = COMMAND_START;
   prev = read_eeprom(ee_addr++);
   while (ee_addr < COMMAND_START + COMMAND_LEN) {
      new = read_eeprom(ee_addr++);
      if (new != prev) {
         n_changes++;
         prev = new;
      }
   }
   return (n_changes >> 1);
}

void main()
{
   int command;
   output_a(0);
   setup_adc_ports (NO_ANALOGS);
   setup_adc (ADC_OFF);
   setup_comparator (A1_VR);
   setup_vref (VREF_LOW|15);
   setup_oscillator (OSC_4MHZ);

   if (read_eeprom(0x00)==0x99) playback();
   WHILE (1){
      command = 0;
      while(command == 0)
      {
         led_blink(2);
         record(COM);
         led_blink(2);
         command = decode_command();
      }
      //delay_ms(100);
      //led_blink(command);

      switch (command) {
         case 1:
            record(REC);
            led_blink(2);
         case 2:
            playback();
            led_blink(2);
            break;
         case 3:
            while (1){
               playback();
            }
         default:
            sleep();
      }
   }
}
```

3 Construction

3.1 Parts List

The complete parts list is Table 2.

3.2 Assembly

After I programmed the PIC with the final version of the software, soldering the parts to the SOIC package takes a little patience. Putting a dab of solder on each lead before joining them eases the process. Also, a pair of normally closed tweezers helped to hold the parts while soldering.

It is easy to make a simple battery clip. See Fig. 5. Alex created a very nice one, but a simple loop and make-shift pad does the trick. A small slip of paper acts as an "off" switch when the unit is not in use.

I hope you find this useful [4]. Happy making!

#	Item	Description
	Programmable LED	
1	Microchip PIC 12F683	SOIC package
1	1/4 watt 100Ω resistor	R_1
1	1/4 watt 1000Ω resistor	R_2
1	LED	T1 (small), green 565nm
1	LDR	In bright light, $R \approx 100\Omega$
		In room light, $R \approx 100,000\Omega$
1	3 V battery	
	Proto Board	
1	Circuit Board	Radio Shack 21-117 DIP proto board
1	Adaptor	16-pin Surface mount-to-DIP adaptor socket
-	Headers	2 or 4 strips, 8 pins long
-	wire and solder	

Table 2: Programmable LED Parts

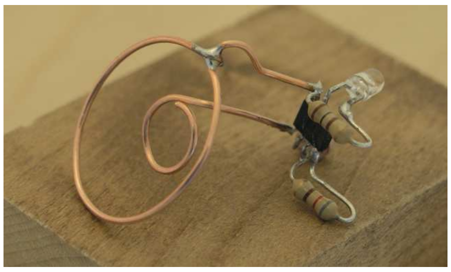

Figure 5: A simple battery clip.

References

[1] *Instructables post by user alex_weber on Mar 6, 2007. http://www.instructables.com/id/ELJXZZVX6JEYVZCV7K*

[2] "Synchronized Fireflies" by user alex_weber on April 6, 2007. http://www.instructables.com/id/ETKA2PCF05JJP4O

[3] http://www.cssinfo.com

[4] This work, including the provide software, is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License, http://creativecommons.org/licenses/by-nc-sa/3.0/. The software is provided without warranty.