A First Amateur Radio PIC Project

Andy Palm N1KSN

In case you haven't noticed, transceivers now use microprocessors extensively to control panel functions, generate RF signals, do digital signal processing, and so on. Many accessories, even kits, use small computers to give equipment functionality that used to be available in only expensive professional models. One line of microcontrollers is very popular with hobbyists, the PIC chip, made by Microchip, Inc. This article describes how you can get started investigating this interesting area and shows a simple ham radio accessory built with a PIC.

There is lots of material on the Internet on PIC chips, but fortunately you can start simply with a couple of main items. First is the PICkit 1 Flash Starter Kit. This package comes with a small programming and development board, a PIC chip, and all the software you will need on a CD. It is still available for only $36. (You will need a PC with a USB port and Windows 98 V2 or higher system.) You can put "PICkit 1" in Google to find a place to buy it. The second thing you will need is the book "123 PIC Microcontroller Experiments for the Evil Genius" by Myke Predko, McGraw-Hill, 2005. This book is currently under $20 at Amazon.com and was written specifically to use the PICkit 1. Besides these two items you will need other small items, discussed in the book, including a couple of PIC16F684 chips, which are under $2.50 each at Mouser Electronics, and maybe a ZIF socket.

The software that comes with the PICkit 1 includes Microchip's Integrated Development Environment (IDE), a system that allows you to write, debug, and download programs from your PC into a PIC chip. Part of this package is a free compiler for a programming language called "c". The book starts you programming the PIC16F684 chip with c. Later in the book you actually start to learn assembly language programming, too. I was at first skeptical that a higher level language like c could be used effectively on such little
computers, but I soon found out how well this version works on PIC chips. The lessons in the book provide code examples that can be immediately used for your own projects.

One item I've long thought of building was a ten minute countdown timer to help me remember to identify myself as per FCC rules when on the air. I always wondered how to build a clock-like timekeeper that was accurate enough. Well, after going through less than half the "Evil Genius" book I had the answer--use a PIC chip and a few other components. The accompanying schematic diagram shows what was needed for the hardware. A seven-segment LED display was selected because it is easy to interface with the PIC and it has a decimal point that I could flash every second. A pushbutton starts the timing sequence and the LED immediately shows "9", representing 9 FULL minutes, plus some number of seconds, remaining. When a minute goes by the displayed digit decreases to "8" and so on. When "0" displays there is a one-minute warning beep. There are three beeps as time runs out and the display goes blank.

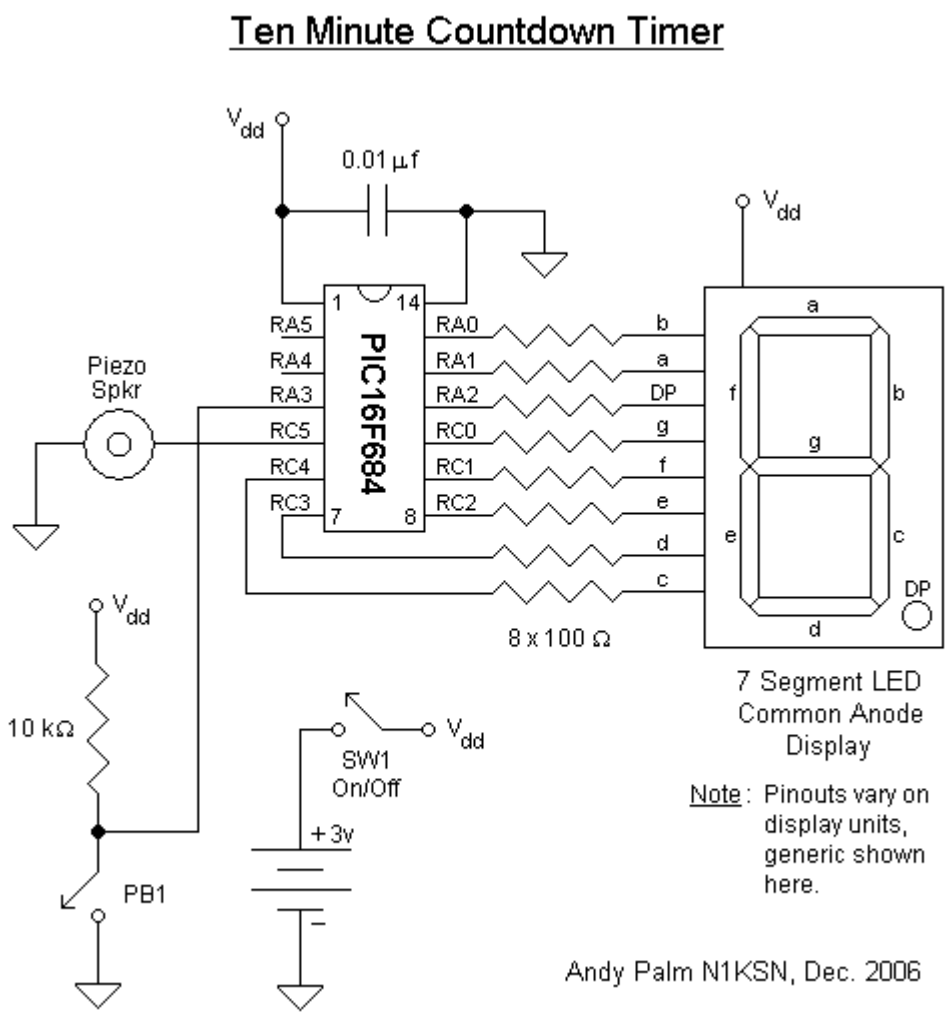
Hopefully you are impressed with how simple the schematic diagram is. It is simple because all the hard work is done in the software. Below this article is the c source code (programming statements) that make it all happen. The PIC chips have certain built-in
hardware modules that simplify many operations. For example, this program uses a timer to count half-second intervals for the main program loop and a pulse width modulator to provide an audio

square wave for the beeps. The book shows you how to use them. Another nice feature about PIC chips is that they have built-in "clocks" that are pretty darn accurate. This means that you don't need to use an external clock or crystal for most projects.

I won't go into detail on the program, but I wanted to mention one feature that was somewhat new to me. Although I'd written computer programs before, I had never done programs designed to work in "real time" where time must be tracked and "events" (like the push of a button) must be recognized right when they happen. One approach to this is a main program loop that never ends. Parts of the loop look for events, make actions, or just wait. To keep track of what should and should not be done during a particular trip through the loop the programmer sets up "state machines." A state machine keeps track of one particular kind of thing, for example, the state of a button. A "state variable" is set to tell the program what particular set of operations should be done. In c you use the "switch" statement to implement a state machine.

This program has two state machines, one to keep track of the push button that starts the timer sequence, and the other to handle the timing, LED display, and piezo speaker (for the beeps). Each time the main loop is executed, the two variables ButtonState and TimerState say which set of actions should be taken. While the timer is active, the loop is executed once every half second. Thus, the display's decimal point can be toggled on and off every half second and you see it flashing once a second.

If you aren't familiar with these concepts, this probably all sounds pretty strange to you. However, the book introduces you to these things and makes it relatively easy. If you always wanted to know something about how these tiny computers work, the PICkit 1 and the Evil Genius book are a combo that is hard to beat as a "fast-track learning path."



Here is the c source code for this project:

/***
TenMinTimer1.c - Ten minute timer using PIC16F684

Uses 7 segment LED display for full minutes remaining, with1 Hz flashing decimal point, warning beep at one minuteremaining, and three final beeps at end of 10 minutes.

Timer sequence is started by release of pushbutton. If doing a restart in middle of an on-going sequence, button must be held down for over one second before releasing.

Assumes 4 MHz clock and 1 MHz instruction cycle. Pins are free for external clock, if desired.

For common anode LED display. Pin assignments for LED are shown below. Other pins are:

Decimal point output on RA2 (low = on, high = off)
Button input on RA3
Sound output on RC5/P1A
RA4 and RA5 are reserved for possible external clock

Subroutine waithalfsec uses internal timer TMR1 to produce a half second delay. Parameter TimeAdj added to TMR1 start value to adjust half second wait time downward.

Beep uses pulse width modulator module with 16x prescaler.
Count for PWM and beep frequency are related by

ToneCount = 1000000 / (Freq * 16) = 62500 / Freq

To maintain an exact 50% duty cycle square wave, round ToneCount to nearest even integer.

Beep is turned on and off using TRISC5.

Reference: 123 PIC Microcontroller Experiments for the Evil Genius, Myke Predko, McGraw-Hill, 2005

Andrew Palm
2006.12.01

**********/

```
#include <pic.h>

__CONFIG(INTIO & WDTDIS & PWRTEN & MCLRDIS & UNPROTECT
        & UNPROTECT & BORDIS & IESODIS & FCMDIS);

#define Button RA3
#define DecPt RA2
#define Down 0
#define Up 1

int k;
int ButtonState, TimerState, TimeCount;
int TimeAdj = 2;  // Value to adjust half second timer
int ToneCount = 26;  // For high freq beep with 16x prescaler

const char LEDDigit[10] = {
// RRRRRRR - PIC16F864 Pin
// AACCCCC
// 1043210
// abcdefg - LED segment
 0b0000001,    // 0 digit, 0/1 = on/off (common anode display)
 0b1001111,    // 1
```

```
 0b0010010,    // 2
 0b0000110,    // 3
 0b1001100,    // 4
 0b0100100,    // 5
 0b0100000,    // 6
 0b0001111,    // 7
 0b0000000,    // 8
 0b0000100};   // 9 digit

/*************************************************************/

waithalfsec(int TimeAdj)
{
  T1CON = 0b00110001;  // Use internal clock, 8x prescaler

  TMR1H = (3036 + TimeAdj) >> 8;    // Set timer high 8 bits
  TMR1L = (3036 + TimeAdj) & 0xFF;  // Set low 8 bits
        // 3036 = 65526 - 500*125 = half sec timer start value

  PEIE = 1;    // Enable peripheral interrupts
  TMR1IF = 0;  // Turn off pending interrupt requests
  TMR1IE = 1;  // Enable TMR1 overflow to request interrupt

  while (!TMR1IF);  // Wait for timer overflow

}  // end waithalfsec

/*************************************************************/

main()
{

  PORTA = 0b000111;  // Turn LED segments and decimal point off
  PORTC = 0b011111;
  CMCON0 = 7;  // Turn off comparators
  ANSEL = 0;   // Turn off ADC

  TRISA = 0b001000;  // RA3 is button input
  TRISC = 0;

  TRISC5 = 1;  // Initialize beep off on RC5/P1A
  T2CON = 0b00000110;      // Turn on PWM with 16x prescaler
  CCP1CON = 0b00001100;    // Enable PWM output
  PR2 = ToneCount;         // Period count for sidetone
  CCPR1L = ToneCount / 2;  // 50% duty cycle square wave
```

```
ButtonState = 0;  // Initialize state machines
TimerState = 0;

while (1 == 1)
{

 switch (ButtonState)  // Button state machine
 {
  case 0:  // (X, U) and entry state
   if (Button == Down) ButtonState = 1;
   break;
  case 1:  // (U, D)
   if (Button == Down) ButtonState = 2;
   break;
  case 2:  // (D, D)
   TRISC5 = 1;  // Shut off beep
   PORTA = 0b000111;  // Turn display off
   PORTC = 0b011111;
   TimerState = 0;
   if (Button == Up)  // (Re)start timer upon button release
   {
    ButtonState = 0;
    TimerState = 1;
    TimeCount = 1200;  // Number of loop cycles in 10 min
   }
   break;
 }

 switch (TimerState)  // Timer state machine
 {
  case 0:  // Do nothing, timer not active
   break;
  case 1:  // Timer is active
   if (TimeCount == 120) TRISC5 = 0;  // Start 1 min warning
   if (TimeCount == 119) TRISC5 = 1;  // Stop warning beep
   if (TimeCount <= 6) TRISC5 = TRISC5 ^ 1;  // Final beeps
   if (TimeCount == 0)  // End timer sequence
   {
    TRISC5 = 1;  // Shut off final beep
    PORTA = 0b000111;  // Turn display off
    PORTC = 0b011111;
    ButtonState = 0;  // Re-initialize
    TimerState = 0;
   }
   else  // Continue timer sequence
   {
```

```
        k = (TimeCount - 1) / 120;  // Display digit
        PORTC = (PORTC & 0b100000) + (LEDDigit[k] & 0b00011111);
        PORTA = (PORTA & 0b111100) + (LEDDigit[k] >> 5);
        DecPt = DecPt ^ 1;  // Toggle decimal point
        TimeCount--;
        waithalfsec(TimeAdj);
      }
      break;
    }

  }  // end loop

}  // end main
```