Safe Lock by TV Remote Controls

Author: *Lilie Yang*
Valmet Automation (Canada) Ltd.
Calgary, Canada
email: liliey@cgy.valmet.com

INTRODUCTION

In this design, a safe box lock controller is developed. Used in many hotels, safe boxes are usually operated by an attached keypad or a credit card. This design is superior to the traditional locking method. It uses a convential television remote control as the command key pad, and uses a 8-pin tiny, yet powerful, PIC12CE519 as the core of the lock unit. Compared with past designs, the innovation here yields a more secure operation, easier to use, and much lower manufacture cost.

THEORY OF OPERATION

In the current market, almost every TV has a wireless remote control. However, different manufactures usually have a different command scheme in their products. It is impossible to cooperate with each manufacture to generate a database to make the lock unit understand each control. Fortunately, as a lock unit, it does not have to understand any TV commands from the remote control. All it has to do is record key sequences entered while doing the lock, and verify that a same key sequence is used while doing the unlock.

In general, all TV remotes share a similar transmit pattern. Figure 1 shows a typical waveform for a key pressed in remote control. By reviewing this figure, we find out that the transmit waveform can always be divided into two sections: data and interval. Furthermore, inside the data section, the '1' and '0' have been represented in different low level periods. The low level period is usually called 'space' in the communication area.

Based on this knowledge, all we have to do is to capture 1s and 0s for each key pressed. To acheive this, we must:

1. Catch the repeated data section.
2. Find a leading space, which is the longest space in the data section.
3. Filter out the second longest space for data 1.
4. Find the shortest space for data 0.

There are many microcontrollers which can be used for this design. However, Microchip's PIC12CE519 is the best fit for overall requirements. It has a power saving function known as "wake up on pin change," a scalable timer, adequate EPROM and RAM for rather complicated programming and, most important, 16 bytes EE memory where locking status and keys can be safely kept. Also, the PIC12CE519 is the most inexpensive one in the 8-bit microcontroller family which can perform the job.

HARDWARE USED IN THIS DESIGN

Figure 2 shows the schematics. The PIC12CE519 is configured using internal Oscillate. The GP3 port takes the output of an infrared module to receive user command, GP4 and GP5 control the output circuits, including LEDs and ON/OFF relays. The GP1 port has a buzzer attached to it which is the audio interface. As the lock will be operated on battery power, a MOS switch is added into the circuit and is controlled by GP0. To alert the user that the battery needs to be replaced, the GP2 port is connected to a low-voltage detector.

SOFTWARE DESIGN

Another advantage of using this PICmicro™ microcontroller is that there are many excellent free or low-cost tools available for use. The simplifiied Source Code in C is shown on page 3. It should be fairly easy to follow. In this kind of application, using high level language can greatly speed up the prototype cycle. The MPLAB™, combined with MPLAB-C, has made the development environment even enjoyable.

Microchip Technology Incorporated, has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work. All property rights, such as patents, copyrights and trademarks remain with author.

BILL OF MATERIALS (BOM)

Item	Part Number
U1	PIC12CE519
T1, T2	2N2907
T3,T4,T5	IRF510
MOD1	IRM-8410 infrared receiver
BZ1	PKM22EPP-40, piezo buzz
VT1	MN1383-J-ND, voltage detector
R1-R5	0.25W resistor
C1	0.1 ceramic
LED1	Green light emitting diode
LED2	Red light emitting diode

FIGURE 1: A TYPICAL PULSE SEQUENCE

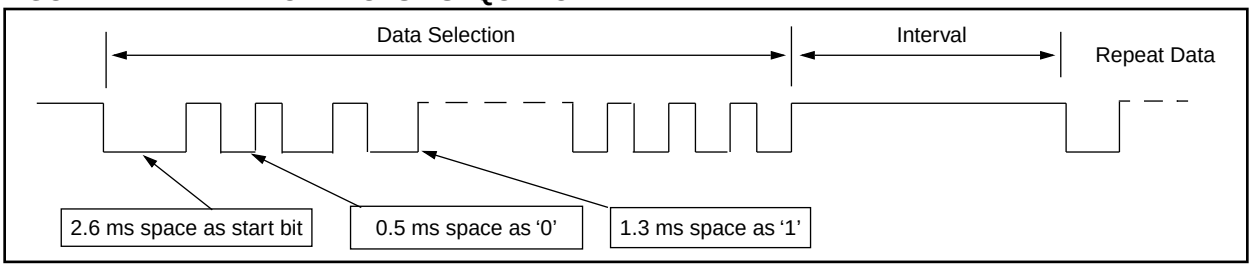
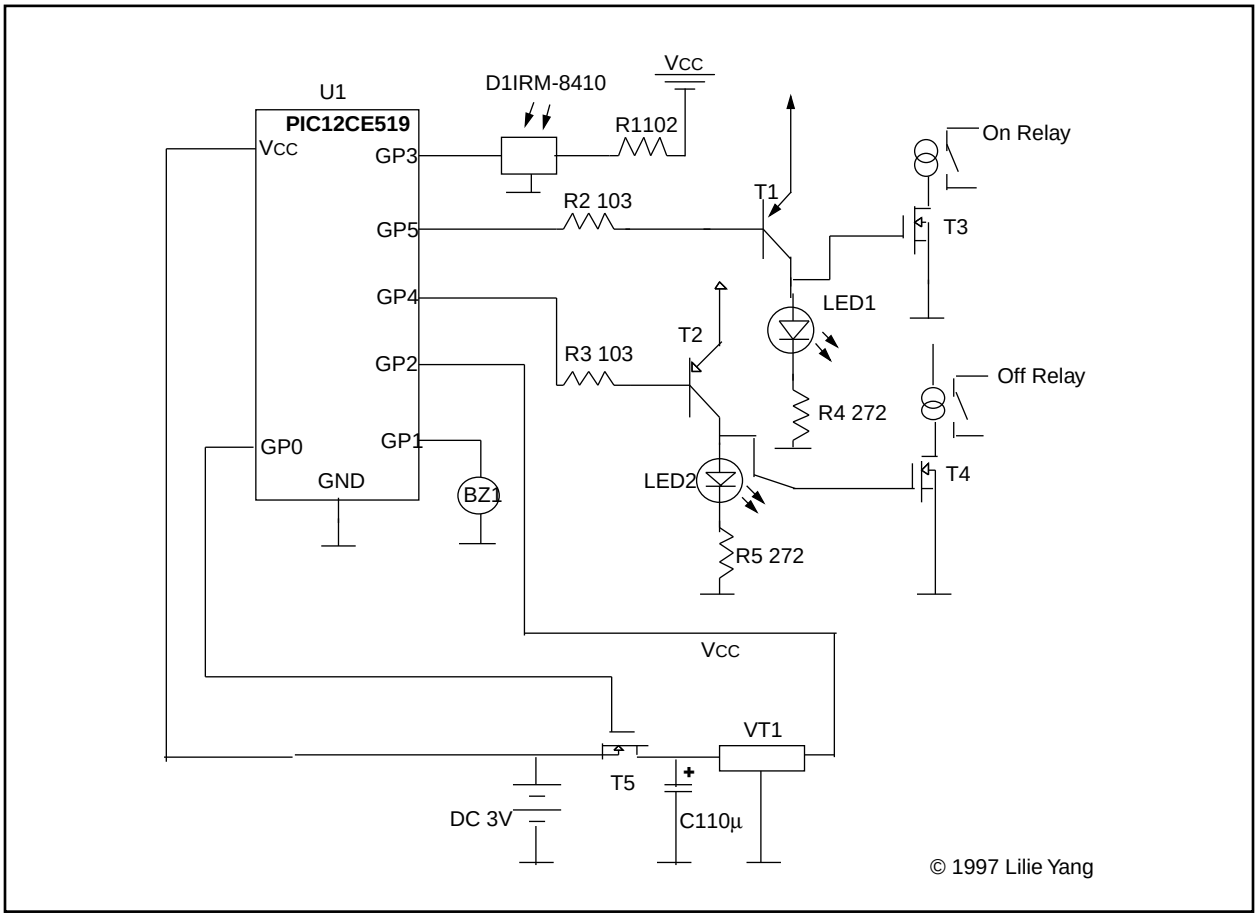


FIGURE 2: SCHEMATICS



© 1998 Microchip Technology Inc.

SOURCE CODE IN C:

```
//
//                                      Safe lock by TV remote controls
//                                      © 1997 Lilie Yang, liliey@cgy.valmet.com
//
#include <12c509.h>

#define MAXBUF                          15          // buffer for pulse
#define MAXBITS                         8           // max number of bits
#define MAXKEYS                         3           // max number of keys

//
// variables used in the program
//
unsigned char pulseBuf[MAXBUF];         // input pulse sequence
unsigned char savedKeys[MAXKEYS];       // saved key, should be kept in Flash memory
unsigned char inComingKeys[MAXKEYS];    // incoming  keys

signed char psIndex;                                // pulse sequence index
signed char keyIndex;                               // key sequence index
signed char tmpIndex;                               // tmp key sequence index

unsigned char timerValue;               // timer value
unsigned char leadingSpcValue;          // timer value for leading space
unsigned char spaceValue;               // timer value for space
unsigned char markValue;                // timer value form mark

unsigned char oneByte;                              // temp variable
unsigned char locked;                               // lock status, should in Flash
unsigned char mainPower;                            // status of main power

//
// function prototypes
//
void main();
void init();
void checkForInput();
void takeOneBit();
void procCmd();
void procOneKey();
void doLocking();
void doUnlocking();

void mainPowerOn();
void mainPowerOff();
void enableInterrupt();
void disableInterrupt();

void beepOneKey();
void beepOK();
void beepError();
void resetTimer();
void resetWDTtimer(unsigned char);

void msDelay (unsigned char ms);
void shortDelay ();
void powerUnlockingRelay();
void powerLockingRelay();
void goSleep();

signed char getPinStatus ();
signed char stopTimer();
signed char wakeOnPinChange();
signed char pinLevelLow();
```

```
//
//           main function
//
void main()
{
   init();                                              // ram check, self test and initializa-
tion
   while (1)                                            // endless loop start from here
   {
      checkForInput();                                  // check to see if any command
      if (timerValue > 0)
              procCmd();                                // process command
      shortDelay();                                     // short delay should be about 100 us
   }
}

void checkForInput()
{
   if (!mainPower)
   {
       mainPowerOn();                // turn on main power
       mainPower = 1;
   }

   resetWDTtimer(100);                                  // set WDT to 100 ms
   goSleep();                                           // read pin before go sleep

   if (wakeOnPinChange())                               // wake up because pin change
       takeOneBit();                                    // read in one bit data
   else                                                 // anything else
   {
      mainPowerOff();                                   // turn off main power
      mainPower = 0;
   }
}

void takeOneBit()
{
    if (pinLeveLow())                                   // pin status must be right
    {
              timerValue = 0;                           // otherwise discard it
              return;
    }

    resetWDTtimer(10);                                  // set WDT to 10 ms
    resetTimer();                                       // clear and restart timer
    if (wakeOnPinChange())
        timerValue = stopTimer();                       // stop timer and get timer value
    else
        timerValue = 0;
}

void procCmd()
{
    if (timerValue >= leadingSpcValue ) // found leading space
    {
              leadingSpcValue = timerValue;             // record longest space value
              psIndex = 0;                              // reset pulse buffer index
    }
    else
    {
              if (timerValue >= markValue)
                  markValue = timerValue;               // record mark value which is '1'
              else if (timerValue > spaceValue)
                  spaceValue = timerValue;              // record space value which is '0'
    }
```


© 1998 Microchip Technology Inc.

```
    if (timerValue == leadingSpcValue)                    // convert timer value to '1' or '0'
        pulseBuf[psIndex] = timerValue;
    else if (timerValue == markValue)
              pulseBuf[psIndex] = 1;
    else
              pulseBuf[psIndex] = 0;

    psIndex++;                                            // increase index for next pulse

    if (psIndex > MAXBITS)                                // has enough bits, store it
             procOneKey();
}

void procOneKey()
{
    disableInterrupt();                                   // disable interrupt for now
    oneByte = 0;                                          // reset tmp variable
    psIndex--;                                            // start with last bit

    while( psIndex > 0)                                   // assembly bits to byte
    {
       oneByte += pulseBuf[psIndex];
       psIndex--;

       if (psIndex > 0)
           oneByte <<= 1;
    }

    inComingKeys[keyIndex] = oneByte;                     // now we got one byte
    keyIndex ++;

    beepOneKey();                                         // beep to ack an key entry
    msDelay(1000);                                        // delay 1 second

    if (keyIndex >= MAXKEYS )
    {
             if (locked)
          doUnlocking();
             else
               doLocking();
        keyIndex = 0;                                     // reset key index
    }
    psIndex = 1;                                          // reset pulse buffer index
    enableInterrupt();                                    // enable interrupt
}

void doLocking()
{
    for (tmpIndex = 0; tmpIndex < MAXKEYS; tmpIndex++)
             savedKeys[tmpIndex] = inComingKeys[tmpIndex];

    beepOK();                                             // beep for OK
    powerLockingRelay();// to set lock on
    msDelay(2000);                                        // sleep 2 seconds
    locked = 1;                                           // set the lock status
}

void doUnlocking()
{
    for (keyIndex = MAXKEYS - 1; keyIndex >= 0; keyIndex --)
             if (inComingKeys[keyIndex] != savedKeys[keyIndex])
             {
               beepError();                               // if found unmatched key, do not con-
tinue
               return;
```

```
            }

    beepOK();                                   // beep for OK
    powerUnlockingRelay();                      // to unlock
    msDelay(2000);                              // sleep 2 seconds
    locked = 0;                                 // set the lock status
}
-END-
```


© 1998 Microchip Technology Inc.