STANDARD
MICROSYSTEMS
CORPORATION

Software Algorithm For Key and Repeat Code Detection in Advanced Consumer Blocks With NEC Framing

By Randy Goldberg

INTRODUCTION

Some of SMSC's Ultra and Super IO's contain an advanced CIRCC block which includes advanced NEC framing detect logic. This logic decodes NEC frames and will identify which key has been depressed on an NEC format remote control. The logic will not however, explicitly identify "repeat codes". A repeat code is what the remote control will send when a key is held down (volume up for example). This application note will show how to use the error detection logic of the advanced CIRCC block to identify repeat codes. Source code for an executable will be provided to show how this can be done. A device driver can be modeled from this code.

Application Note 7.10 is a Consumer IR primer, and is recommended reading if the reader is interested in this application note. Also, for a complete description of the CIRCC block, consult the CIRCC data sheet, which can be found on SMSC web site at www.smsc.com

Consumer Basics

Unlike FIR and SIR (IrDA) which have well defined protocols that can be adhered to by all manufacturers, Consumer IR protocols suffer from a lack of consistency and vary from manufacturer to manufacturer. However at the lowest level, most Consumer protocols can be represented in the same way. A bit rate is defined, a carrier rate is defined, a low level '0' is then represented by a bit rate time with the carrier active, a low level '1' is defined by a bit rate time with no activity on the line. (In the following section (NEC framing), an "old 1" means a standard consumer bit rate time of no activity, and an "old 0", means a standard consumer bite rate time with the carrier active).

80 Arkay Drive
Hauppauge, NY 11788
(631) 435-6000
FAX (631) 273-3123

NEC FRAMING

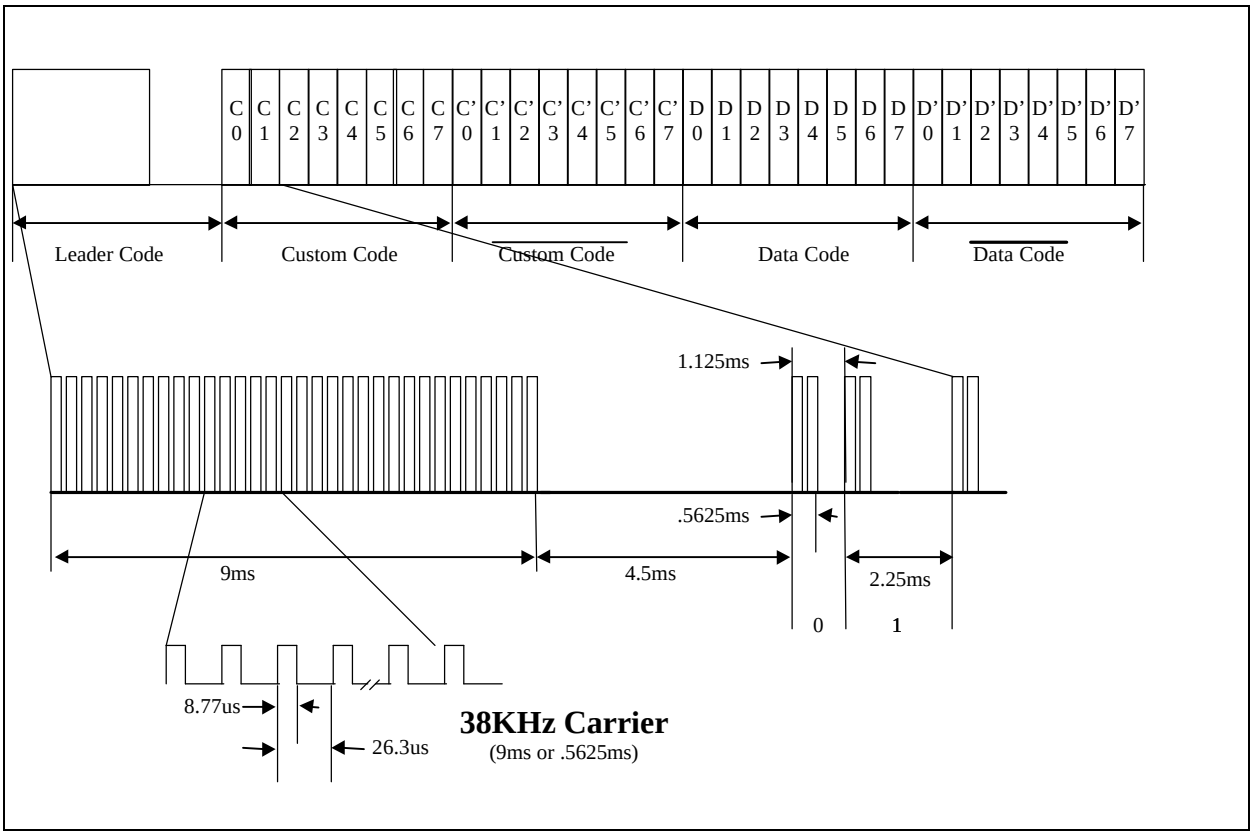

FIGURE 1 - NEC FRAMING FOR A KEY CODE PRESS

The consumer block of the FDC37B77x supports NEC framing in receive mode only.

The NEC framing Data Link Protocol for a **standard key press** consists of a leader code, an 8bit custom code, an 8 bit custom code bar, and 8 bit data code, and an 8 bit data code bar. The custom code can represent the specific device (TV, VCR) and the data code can represent the specific key code. The 'bit' rate is 1.88kHz and the carrier rate is 38kHz. The leader code is 9ms (16 old bits) of carrier and 4.5ms (8 old bits) of idle. The data is a pulse position modulation scheme. An NEC '0' is represented by an old '0' followed by a single old '1'. An NEC '1' is represented by an old '0' followed by a three old '1's.

The NEC framing Data Link Protocol for a **repeat code** consists of a modified leader code of 9ms (16 old bits) of carrier and 2.25ms (4 old bits) of idle, followed by one bit of NEC '0', followed by an idle time of 108ms.

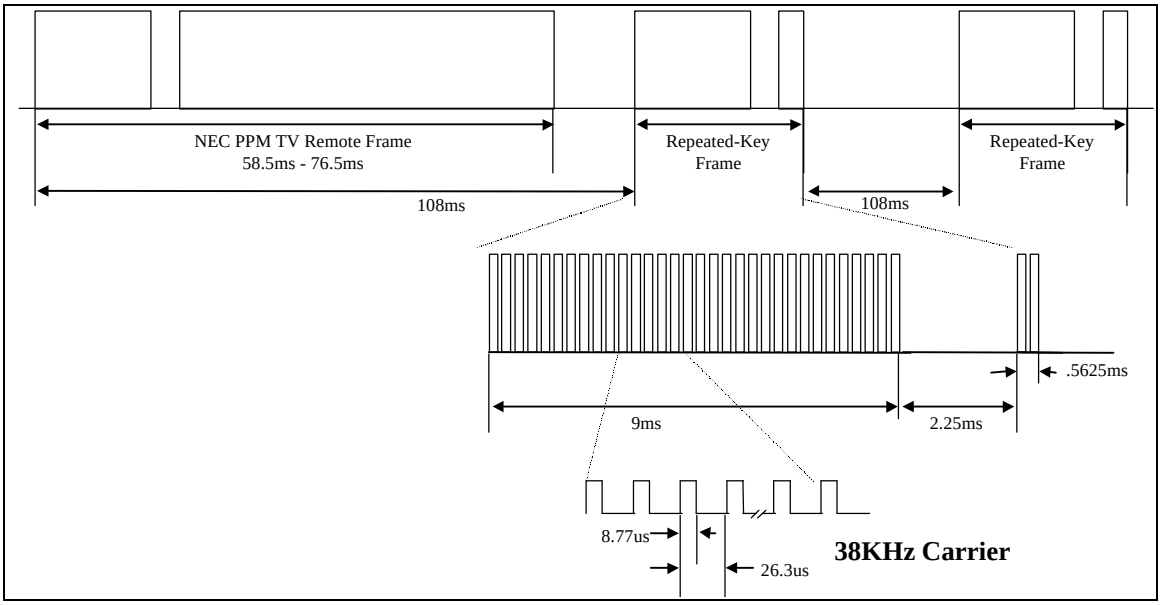

FIGURE 2 - NEC REPEAT KEY FRAME

SMSC'S ADVANCED CIRCC BLOCK

SMSC's advanced CIRCC block can be simplified and summarized in the following way. The block contains logic to detect NEC key press frames. In receive mode, proper carrier frequency rates and NEC framing can be selected, and then the receiver can be enabled. Upon receiving a IR consumer data stream, interrupts will be generated, various status bits will be set, and data may or may not be placed in the FIFO. All of these results can be evaluated to determine if a proper NEC frame occurred, and whether it was a NEC key press, a repeat code, or invalid data.

Key Bits in the CIRCC Block

The software algorithm to determine key press and repeat codes depends on evaluation of certain bits within the CIRCC block. The following is a description of these bits, not as they are necessarily described in the data sheet, but as they will be used for this algorithm.

TABLE 1 – KEY BITS IN THE CIRCC BLOCK

BIT	REGISTER BIT IS IN	BIT DEFINITION AS APPLICABLE TO REPEAT CODE ALGORITHM
Active frame	Interrupt identification register	This bit gets set when any IR transition occurs. It can generate an interrupt, and will be used to detect the beginning of an IR event.
Frame abort	Line status register	This bit gets set if a IR carrier frequency, outside of the range that the receive block has been set for, is detected
Frame error	Line status register	This bit gets set when a valid NEC leader code is detected, and then an NEC protocol violation is detected.
FIFO Not Empty	Bus status register	This register indicates that there is data in the FIFO

THE ALGORITHM

The frame error bit is the key to the repeat code detection algorithm. The frame error bit gets set only if an NEC protocol violation is detected AFTER a valid NEC leader code is detected. (An NEC protocol violation is defined as 2 successive standard (old) consumer bit times of "0" or 3 successive standard (old) bit times of "1", after a valid leader code has been detected. These data strings do not occur during NEC Key press frames with NEC encoded data).

When a error free NEC key press occurs, first the leader code happens, then the custom code, custom code bar, data code, and data code bar occur. These codes (custom and data) do not have protocol violations, the receive block detects them as valid, and places the appropriate codes in the FIFO.

When a NEC repeat code occurs, after the leader code, there is a single NEC encoded “0” bit, followed by a 108ms idle time, followed by the leader code for the next repeat code. The 108ms idle time is identified as an NEC protocol violation, and the Frame error bit is set. Because no valid custom or data codes are sent with a repeat code, no data is placed in the FIFO.

Therefore, after an “NEC” event, if the Frame error bit is not set, and there is data in the FIFO, the event was a valid NEC key press. If the Frame error bit is set, and there is no data in the FIFO, the event was an NEC repeat code.

Also key to the algorithm is a “waiting period”, after any IR activity occurs. A valid NEC key press can be made to generate an interrupt through the FIFO bit in the Interrupt identification register. A repeat code, does not place data in the FIFO, and the Frame error that a repeat code causes is not capable of generating an interrupt. The workaround for this is to use the “active frame” bit to generate an interrupt when any IR activity occurs. Then, a delay of 130ms should be executed, and then the status registers checked. In a real software driver, the interrupt subroutine could be exited while the delay was being executed, so that other system tasks could be done. The reason 130ms was chosen, is that if the “active frame” interrupt occurs at the very beginning of an IR event, 130 ms is enough time for a valid NEC key press frame to occur. It is also enough time to generate 2 repeat codes, which is what is required to generate a frame error.

The following are various “IR events” and how the CIRCC block will respond after 130ms delay:

TABLE 2 – IR EVENTS AND THE EXPECTED RESPONSE

IR ACTIVITY	POSSIBLE SOURCE	CIRCC RESPONSE
Bad ‘spikeish’ IR data	Fluorescent light, poorly aimed remote control	Active frame bit will be set. No NEC data will be placed in the FIFO, and FA may be set. FE will not be set
Valid IR data, but not NEC protocol	Non NEC remote control	Active frame will be set. No data is placed in the FIFO. Frame error is not set. Frame abort bit may be set, depending on the carrier rate frequency.
Valid NEC key code press	NEC format remote control, key press	Active frame bit will be set. NEC data will be placed in the FIFO. FA and FE not set
Valid NEC repeat code	NEC format remote control, key held down	Active frame bit will be set. No NEC data will be placed in the FIFO. FE set, FA not set.

A simple program was written that simply enables the consumer receiver, waits for the presence of IR activity, waits 130ms, and then interrogates the status bits to determine which of the above events occurred. It then prints to the screen the results. The program then repeats.

The program can be used as a guide for programmers writing real system code.

```
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
#include <bios.h>


int cadd=0x370;      // initial configuration address
int cdata=0x371;

int clkadd=0x70;    // real time clock addresses for delay routine
int clkdata=0x71;

int cirbase0=0x300;
int cirbase1,cirbase2,cirbase3,cirbase4,cirbase5,cirbase6,cirbase7;

int i,q,l,m,n,o,p;

int activeframe=0;
int frameerror=0;
int frameabort=0;
int datainfifo=0;
int gooddatainfifo=0;
int data1,data2,data3;
```

```
unsigned long lc;


void hdelay2(int);
int search_for_IRCC_Block(void);
int search_for_CIRCC_Block_which_supports_NECframing(void);



main() {

// generate the consumer runtime registers offsets
cirbase1=cirbase0+1; cirbase2=cirbase0+2; cirbase3=cirbase0+3;
cirbase4=cirbase0+4;cirbase5=cirbase0+5; cirbase6=cirbase0+6;
cirbase7=cirbase0+7;

// make sure the system contains an SMC CIRCC block which supports
// NEC framing
if (
(search_for_IRCC_Block()) &&
search_for_CIRCC_Block_which_supports_NECframing() )
{

// Loop until a key is hit
lc=0;
while (1==1) {
lc=lc+1;

if (kbhit())  break;

//enable the recieve block


outportb(cirbase7,0x70);   // reset block
outportb(cirbase7,0x20);

// Register block 1
outportb(cirbase7,0x21);
outportb(cirbase0,0x30);  // SCE reg A,  enable consumer mode,  half dup off,
                          // RX/TX polarity to low
outportb(cirbase1,0x40);  // SCE reg B,  Select Dedicated IR pins
outportb(cirbase2,0x2);   // FIFO threshold equal 2

// Register block 2
outportb(cirbase7,0x22);
outportb(cirbase0,0xda);  //  Consumer control register,  Enable NEC FRAMING
outportb(cirbase1,0x29);  //  Carrier rate register,  38khz  NEC compatible
outportb(cirbase2,0x37);  //  BIT rate register,  1.88hz  NEC compatible


// Register block 4
outportb(cirbase7,0x22);
outportb(cirbase3,0x00);  //  Set Size register
outportb(cirbase4,0x03);  //

// Register block 0
outportb(cirbase7,0x20);
outportb(cirbase2,0xf0);  //  Enable interrupts
outportb(cirbase5,0x80);  //  Enable receive



/*  wait 10 seconds for key URC activity,   poll the active frame bit of the
interupt identification register.  This bit goes active on any IR activity
 */
l=0;
while (   ((inportb(cirbase1)&0x80)!=0x80) &&   (l<10000)  )  {
hdelay2(1);
```

```
l++ ;
}
if (l<10000) activeframe=0x01;
else
cputs("NO IR activity occured in 5 seconds \n\r");




// if active frame,  then continue with the rest of the routine
if (activeframe==0x01) {


hdelay2(130);   //  wait 130ms,  if a valid NEC frame,  it will take this long
              // to complete



// get status information
frameerror=0;
frameabort=0;
gooddatainfifo=0;
datainfifo=0;


// check frame error bit in the line status register
if ((inportb(cirbase3)&0x20)==0x20) frameerror=0x01;

// check the frame abort bit in the line status register
if ((inportb(cirbase3)&0x04)==0x04) frameabort=0x01;

/* check the FIFO empty bit in the bus status register   if there
is data in the FIFO,  retrieve it */
if ((inportb(cirbase6)&0x80)==0x80) datainfifo=0x01;
if (datainfifo==0x01){
//read fifo data
data1=inportb(cirbase0); data2=inportb(cirbase0); data3=inportb(cirbase0);
if ((data1==0x04)&&(data2==0xfb)) gooddatainfifo=0x01;
    }


//  now evaluate the status,  and decide what kind of data IR activity
//  occurred

// frame abort,  bad data
if (frameabort==0x01) {
cputs("bad data,  frame abort \n\r");}

// active frame,  without data in the FIFO or a frame error means unknown
// Activity
if (  (activeframe==0x01) && (datainfifo==0x00)&& (frameerror==0x00)     ) {
cputs("bad data, unknown IR activity with no status bits set \n\r");}

// Active frame,  and data in the FIFO with no frame error means a
// valid nec code.  If between 0x10 and 0x19,  it is a numerice key
//  0-3 are volume/channel up and down
if (  (activeframe==0x01) && (datainfifo==0x01)&& (frameerror==0x00)     ) {
if (data3==0x00) {cputs("Channel up \n\r");      goto endprint;}
if (data3==0x01) {cputs("Channel down \n\r");    goto endprint;}
if (data3==0x02) {cputs("Volume up \n\r");       goto endprint;}
if (data3==0x03) {cputs("Volume down \n\r");     goto endprint;}
if ( (data3>0x0a)   && (data3<0x1a)   )
{cprintf("Valid NEC Frame,  numeric key:%2.2x\r\n",data3-16);  goto endprint;}
cprintf("Valid NEC Frame,  data code is:%2.2x\r\n",data3);
endprint:
}
```

```
// Active frame,   with no data in the FIFO and the frame error bit set is a
// repeat code
if (  (activeframe==0x01) && (datainfifo==0x00)&& (frameerror==0x01)     ) {
if (data3==0x00) {cputs("Repeat Channel up \n\r");      goto endprint2;}
if (data3==0x01) {cputs("Repeat Channel down \n\r");    goto endprint2;}
if (data3==0x02) {cputs("Repeat Volume up \n\r");       goto endprint2;}
if (data3==0x03) {cputs("Repeat Volume down \n\r");     goto endprint2;}
if ( (data3>0x0a)   && (data3<0x1a)   )
{cprintf("Repeat   numeric key:%2.2x\r\n",data3-16);  goto endprint2;}
cprintf("Frame error with no abort and no data in FIFO  >>>>   Repeat code
:%2.2x\r\n",data3);
endprint2:
}


// rarely,  you can get this case.  If you wait to long to look at the status
// you can get a valid byte,  and a subsequent repeat code
if (  (activeframe==0x01) && (datainfifo==0x01)&& (frameerror==0x01)     ) {
cprintf("Valid NEC Frame,  with a repeat code:%2.2x\r\n",data3);
}


 }


}

  return(0);
}

 }


// end program,  start procedures



//  Use the real time clock to generate an accurate hardware delay
//  with true ms resolution
 void hdelay2 (int count)
{
int i;
int savertca;

outportb (clkadd,0x0a);
savertca=inportb(clkdata);
outportb (clkdata,0x24);
outportb (clkadd,0x0c);

count=count*4;
i=0;
while ( i<count ) {
if((inportb(clkdata)&0x40)==0x40)   i=i+1;
}

outportb (clkadd,0x0a);
outportb (clkdata,savertca);
}



int search_for_IRCC_Block(void)
{
/* this routine will determine if there is an SMC FIR block within the system
by entering configuration space,  and trying to set up the base address of the
FIR block,  If the newly loaded base address of the FIR block,  can be successfully
read back,  it is assumed that an SMC FIR block was found.  An additional check
is to access the runtime registers of the FIR block, and verify that the SMSC
ID bytes within register block 3 are correct
```

```
*/

int test1,test2,test3,test4,test6,test7,test8,test9;

// first look for part at  config space 370
cadd=0x370;
cdata=0x371;
outportb(cadd,0x55);   //enter config
outportb(cadd,0x7); outportb(cdata,0x05);    //  point to logical device 2 (serial
port 2)
outportb(cadd,0x62); outportb(cdata,0x03);   //  set up base address for FIR block at
0x300
outportb(cadd,0x63); outportb(cdata,0x00);
outportb(cadd,0x62);   test1=inportb(cdata);
outportb(cadd,0x63);   test2=inportb(cdata);
outportb(cadd,0x30); outportb(cdata,0x01);  //enable logical device


outportb(cadd,0x20);        //  if 78x,  enable GPIO
if (   inportb(cdata) == 0x44) {
outportb(cadd,0x7); outportb(cdata,0x08); //enable gpio
outportb(cadd,0xe4); outportb(cdata,0x09); //enable gpio
outportb(cadd,0xe5); outportb(cdata,0x08); //enable gpio
                          }


outportb(cadd,0xaa);                        // exit config
// access runtime register of FIR block,  read SMSC ID bytes
outportb(cirbase7,0x23); test3=inportb(cirbase0);
outportb(cirbase7,0x23); test4=inportb(cirbase1);




// now check at config space 3f0
cadd=0x3f0;
cdata=0x3f1;
outportb(cadd,0x55);   //enter config
outportb(cadd,0x7); outportb(cdata,0x05);    //  point to logical device 2 (serial
port 2)
outportb(cadd,0x62); outportb(cdata,0x03);   //  set up base address for FIR block at
0x300
outportb(cadd,0x63); outportb(cdata,0x00);
outportb(cadd,0x62);   test6=inportb(cdata);
outportb(cadd,0x63);   test7=inportb(cdata);
outportb(cadd,0x30); outportb(cdata,0x01);  //enable logical device

outportb(cadd,0x20);        //  if 78x,  enable GPIO
if (   inportb(cdata) == 0x44) {
outportb(cadd,0x7); outportb(cdata,0x08); //enable gpio
outportb(cadd,0xe4); outportb(cdata,0x09); //enable gpio
outportb(cadd,0xe5); outportb(cdata,0x08); //enable gpio
                          }

outportb(cadd,0xaa);                        // exit config
// access runtime register of FIR block,  read SMSC ID bytes
outportb(cirbase7,0x23); test8=inportb(cirbase0);
outportb(cirbase7,0x23); test9=inportb(cirbase1);




if (   ((test1==0x03) && (test2==0x00)&& (test3==0x10)&& (test4==0xb8))
       ||
       ((test6==0x03) && (test7==0x00)&& (test8==0x10)&& (test9==0xb8))

)
{
```

```
 cputs("SMC DEVICE FOUND WITH FIR BLOCK \n\r");
 return(1);
}

else
{
cputs("NO SMC DEVICE FOUND WITH FIR BLOCK \n\r");
return(0);
}
}



int search_for_CIRCC_Block_which_supports_NECframing(void)
/*  this routine checks if the FIR block is capable of decoding NEC frames in
consumer mode.  It does this by looking at a specific run time register
that is there only in Consumer cores that support NEC framing.  Cannot just look
at the CHIP ID register because it may be incremented on future Silicon,  so
it is better just to observe this new register.
*/

{
int test1;


outportb(cirbase7,0x22);
outportb(cirbase3,0x5a); //  write custom code register (offset 3 in block2) to 0x5a
test1=inportb(cirbase3);

cprintf("frameerror:%2.2x\r\n",test1);


if (test1==0x5a)
{
 cputs("FIR BLOCK SUPPORTS NEC FRAMING \n\r");
 return(1);
}

else
{
cputs("FIR BLOCK DOES NOT SUPPORTS NEC FRAMING \n\r");
return(0);

}
}
```