

```

/*****
/*
/*  FILENAME:    mandec.c
/*
/*
/*  ABSTRACT:
/*      Routines to manage a pair of interrupt routines to decode an
/*      incoming Manchester encoded serial data stream. The data decoded
/*      from the stream includes RTC real time data to sync a local
/*      software based RTC function. The data stream updates arrive at
/*      one second intervals.
/*
/*
/*  CHANGE LOG:
/*      DATE      AUTHOR    SDR    DESCRIPTION
/*      28-jan-12  M.Karas   ---    Initial C version as adapted from the
/*                                  assembly language versions used on a
/*                                  previous version of the clock that used
/*                                  a small memory AT81LC4052 MCU from Atmel.
/*
/*
*****/

#include "c8051f340.h"          /* SFR declarations */
#include "GLCD.h"
#include "mandec.h"
#include "debug.h"

/*
**
** global decoder support variables
**
*/

bit decode_enable;             /* flag used to enable/disable the manchester decoder */
bit hilo_bit;                  /* flag used to tracks the high or low pulse interrupt */
                                /* state 1=just captured high width (H->L interrupt */
                                /*      0=just captured low width (H->H interrupt */

unsigned int width;             /* captured width of a timer pulse */
unsigned char xdata bit_buf[32]; /* 256 bits buffer to hold serial bit stream */
unsigned char xdata bit_idx;    /* index to bit position in the bit_buf[] array */
unsigned char code bit_masks[8] = {0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
unsigned char man_state;        /* manchester decoder state machine state variable */

/*
**
** initialize the external interrupt 0.
** enabled as rising edge active for end of low pulse detect.
** sets up the enabled state and clears any flags
** involved with the interrupt service routine
**
*/

void init_x0(void)
{

```

```

EX0 = 0;          /* disable the ext 0 interrupt */
IE0 = 0;          /* clear any pending ext 0 interrupts */
PX0 = 1;          /* make the ext 0 interrupt be high priority */

IT0 = 1;          /* setup ext 0 int to be edge triggered */

IT01CF &= 0xF0;   /* keep bits for interrupt 1 */
IT01CF |= 0x8;    /* Bit 3 - IN0PL - 0 = active low 1 = active high */
/* Bit 2-0 IN0SL[2:0] 0 selects P0.0 */
/*                      1 selects P0.1 */
/*                      2 selects P0.2 */
/*                      3 selects P0.3 */
/*                      4 selects P0.4 */
/*                      5 selects P0.5 */
/*                      6 selects P0.6 */
/*                      7 selects P0.7 */

EX0 = 1;          /* enable the interrupt */
}

/*
**
** initialize the external interrupt 1.
** enabled as falling edge active for end of high pulse detect.
** sets up the enabled state and clears any flags
** involved with the interrupt service routine
**
*/

void init_x1(void)
{
    EX1 = 0;          /* disable the ext 1 interrupt */
    IE1 = 0;          /* clear any pending ext 1 interrupts */
    PX1 = 1;          /* make the ext 1 interrupt be high priority */

    IT1 = 1;          /* setup ext 1 int to be edge triggered */

    IT01CF &= 0x0F;   /* keep bits for interrupt 0 */
    IT01CF |= 0x10;   /* Bit 3 - IN1PL - 0 = active low 1 = active high */
/* Bit 2-0 IN1SL[2:0] 0 selects P0.0 */
/*                      1 selects P0.1 */
/*                      2 selects P0.2 */
/*                      3 selects P0.3 */
/*                      4 selects P0.4 */
/*                      5 selects P0.5 */
/*                      6 selects P0.6 */
/*                      7 selects P0.7 */

    EX1 = 1;          /* enable the interrupt */
}

/*
**

```

```

** routine to initialize timer 1 for /12 clocking from the system clock
** and then set its mode for use in upclock timing of the pulse widths.
**
*/

void init_t1(void)
{
    CKCON &= 0xF4;          /* clear timer 0/1 prescale bits to /12 and */
                           /* clear the T1M bit to select timer 1 to use */
                           /* the prescaled clock */
    TMOD &= 0x0F;          /* clear timer 1 control bits */
    TMOD |= 0x10;          /* select timer 1 in mode 1 16-bit timer */
    ET1 = 0;               /* no interrupts from timer 1 */
    TR1 = 0;               /* keep timer 1 from running till needed */
}

/*
**
** external interrupt 0 service routine
**
** used to respond to the radio receiver outputs rising edges
** this means that the ex0 interrupt needs to be configured for
** rising edge active.
** this captures a timer 1 reading that represents the previous pulse
** low time period.
**
*/

void x0_isr(void) interrupt 0 using 2
{
    TR1 = 0;               /* stop timer 1 to read it */

    if(TF1 == 1)           /* if timer overflow set as maximum pulse width */
    {
        width = 0xFFFF;
    }
    else                   /* no overflow so just read timer's value */
    {
        width = (TH1 << 8) | (TL1);
    }
    TH1 = 0;               /* reset timer for the next pulse */
    TL1 = 0;
    TF1 = 0;               /* clear any pending overflow flag */
    TR1 = 1;               /* let timer 1 run again */

    hilo_bit = 0;          /* clear hilo bit to show LO just measured */

    man_decode();           /* call the manchester decode state machine */

    IE0 = 0;               /* clear the interrupt pending bit */
}

/*

```

```

**
** external interrupt 1 service routine
**
** used to respond to the radio receiver outputs rising edges
** this means that the ex1 interrupt needs to be configured for
** falling edge active.
** this captures a timer 1 reading that represents the previous pulse
** high time period.
**
*/

void x1_isr(void) interrupt 2 using 2
{
    TR1 = 0;                      /* stop timer 1 to read it */

    if(TF1 == 1)                  /* if timer overflow set as maximum pulse width */
    {
        width = 0xFFFF;
    }
    else                          /* no overflow so just read timer's value */
    {
        width = (TH1 << 8) | (TL1);
    }
    TH1 = 0;                      /* reset timer for the next pulse */
    TL1 = 0;
    TF1 = 0;                      /* clear any pending overflow flag */
    TR1 = 1;                      /* let timer 1 run again */

    hilo_bit = 1;                 /* set hilo bit to show HI just measured */

    man_decode();                 /* call the manchester decode state machine */

    IE1 = 0;                      /* clear the interrupt pending bit */
}

/*
**
** initialize the manchester decoder routine to
** initial state and disabled.
**
*/

void man_init(void)
{
    decode_enable = 0;            /* disable the decoding state machine */
    man_state = MAN_STATE_IDLE;   /* set initial state */
    bit_idx = 0;                  /* set bit array index to start */
}

/*
**
** state machine code used to decode the manchester waveform from the
** radio receiver. This is called at each transition interrupt from the

```

```

** INT0 and INT1 interrupts. The HILO_BIT tells us which pulse width has
** just been captured and the 16-bit variable WIDTH holds the captured
** pulse width.
**
*/

void man_decode(void)
{
    switch(man_state)
    {
        /* invalid state */
        default:
            man_state = MAN_STATE_IDLE;           /* reset state machine */
            break;

        /* state waiting for decode enable to become set */
        case MAN_STATE_IDLE:
            if(decode_enable != 0)
            {
                man_state = MAN_STATE_SYNC_HI; /* setup to start looking for 3T.Hi */
                bit_idx = 0;                    /* initialize as for first bit coming */
            }
            break;

        /* state waiting for a 3T.Hi sync pulse. stay here till found */
        case MAN_STATE_SYNC_HI:
            if(hilo_bit != 0)                    /* wait for a high pulse seen */
            {
                /* check for pulse in 3T range */
                if((width > T3_LO) && (width < T3_HI))
                {
                    man_state = MAN_STATE_SYNC_LO; /* setup next to look for 3T.Lo */
                }
                else
                {
                    bit_idx = 0;                  /* ensure ready for first bit */
                }
            }
            break;

        /* state waiting for a 3T.Lo sync pulse. restart at sync high if not seen */
        case MAN_STATE_SYNC_LO:
            if(hilo_bit == 0)                    /* found a low pulse */
            {
                /* check for pulse in 3T range */
                if((width > T3_LO) && (width < T3_HI))
                {
                    man_state = MAN_STATE_FAKE_0; /* setup next to look for 1T.Hi */
                }
                else
                {
                    man_state = MAN_STATE_IDLE;   /* reset state machine */
                }
            }
    }
}

```

```
}
else                                     /* high pulse here is invalid */
{
    man_state = MAN_STATE_IDLE; /* reset state machine */
}
break;

/* state looking for the 1T.Hi for the starting of the Fake 0 byte */

case MAN_STATE_FAKE_0:
    if(hilo_bit != 0)                   /* found a high pulse */
    {
        /* check for pulse in 1T range */
        if((width > T1_LO) && (width < T1_HI))
        {
            man_state = MAN_STATE_1T2T_LO; /* setup next to look for 1T.Hi */
        }
        else
        {
            man_state = MAN_STATE_IDLE; /* reset state machine */
        }
    }
    else                                 /* low pulse here is invalid */
    {
        man_state = MAN_STATE_IDLE; /* reset state machine */
    }
    break;

/* state waiting for 1T.Lo or 2T.Lo */
case MAN_STATE_1T2T_LO:
    if(hilo_bit == 0)                   /* found a low pulse */
    {
        /* check for pulse in 1T range */
        if((width > T1_LO) && (width < T1_HI))
        {
            man_state = MAN_STATE_1T_HI;
            break;
        }
        /* check for pulse in 2T range */
        if((width > T2_LO) && (width < T2_HI))
        {
            put_bit(bit_idx, 1); /* log as '1' bit as received */
            bit_idx++;
            if(bit_idx < SER_BIT_CNT) /* still more bits to process */
            {
                man_state = MAN_STATE_1T2T_HI;
            }
            else
            {
                decode_enable = 0; /* indicate that bit stream received */
                man_state = MAN_STATE_IDLE; /* reset state machine */
            }
            break;
        }
    }
}
```

```
    }
    man_state = MAN_STATE_IDLE;    /* reset state machine */
}
else                                /* high pulse here is invalid */
{
    man_state = MAN_STATE_IDLE;    /* reset state machine */
}
break;

/* state waiting for 1T.Hi */
case MAN_STATE_1T_HI:
    if(hilo_bit != 0)                /* found a high pulse */
    {
        /* check for pulse in 1T range */
        if((width > T1_LO) && (width < T1_HI))
        {
            put_bit(bit_idx, 0);    /* log as '0' bit as received */
            bit_idx++;
            if(bit_idx < SER_BIT_CNT) /* still more bits to process */
            {
                man_state = MAN_STATE_1T2T_LO;
            }
            else
            {
                decode_enable = 0;    /* indicate that bit stream received */
                man_state = MAN_STATE_IDLE; /* reset state machine */
            }
            break;
        }
        man_state = MAN_STATE_IDLE;    /* reset state machine */
    }
    else                                /* low pulse here is invalid */
    {
        man_state = MAN_STATE_IDLE;    /* reset state machine */
    }
    break;

/* state waiting for 1T.Hi of 2T.Hi */
case MAN_STATE_1T2T_HI:
    if(hilo_bit != 0)                /* found a high pulse */
    {
        /* check for pulse in 1T range */
        if((width > T1_LO) && (width < T1_HI))
        {
            man_state = MAN_STATE_1T_LO;
            break;
        }
        /* check for pulse in 2T range */
        if((width > T2_LO) && (width < T2_HI))
        {
            put_bit(bit_idx, 0);    /* log as '0' bit as received */
            bit_idx++;
        }
    }
}
```

```

        if(bit_idx < SER_BIT_CNT)    /* still more bits to process */
        {
            man_state = MAN_STATE_1T2T_LO;
        }
        else
        {
            decode_enable = 0;        /* indicate that bit stream received */
            man_state = MAN_STATE_IDLE; /* reset state machine */
        }
        break;
    }
    man_state = MAN_STATE_IDLE;    /* reset state machine */
}
else                                /* low pulse here is invalid */
{
    man_state = MAN_STATE_IDLE;    /* reset state machine */
}
break;

/* state waiting for 1T.Lo */
case MAN_STATE_1T_LO:
    if(hilo_bit == 0)                /* found a low pulse */
    {
        /* check for pulse in 1T range */
        if((width > Tl_LO) && (width < Tl_HI))
        {
            put_bit(bit_idx, 1);    /* log as '1' bit as received */
            bit_idx++;
            if(bit_idx < SER_BIT_CNT) /* still more bits to process */
            {
                man_state = MAN_STATE_1T2T_HI;
            }
            else
            {
                decode_enable = 0;    /* indicate that bit stream received */
                man_state = MAN_STATE_IDLE; /* reset state machine */
            }
            break;
        }
        man_state = MAN_STATE_IDLE; /* reset state machine */
    }
    else                                /* high pulse here is invalid */
    {
        man_state = MAN_STATE_IDLE; /* reset state machine */
    }
    break;
} /* end of the state machine switch */
} /* end of the decoder routine */

/*
**
** function to put a bit value into the bit array. Entry arguments are the
** bit index and the bit value to store.

```



```
/**
** Note that this does no error checking on the index because the
** bit array is allocated for a full 256 bits.
**
*/

void put_bit(unsigned char index, bit bitval)
{
    unsigned char offset;

    offset = index >> 3;

    if(bitval == 0)
    {
        bit_buf[offset] &= ~bit_masks[index & 0x07];
    }
    else
    {
        bit_buf[offset] |= bit_masks[index & 0x07];
    }
}

/**
**
** function to get a bit value from the bit array. Entry argument is the
** bit index. Bit value is returned.
**
** Note that this does no error checking on the index because the
** bit array is allocated for a full 256 bits.
**
*/

bit get_bit(unsigned char index)
{
    unsigned char offset;

    offset = index >> 3;

    if(bit_buf[offset] & bit_masks[index & 0x07])
    {
        return(1);
    }
    else
    {
        return(0);
    }
}
```