```
/*
 * File:   8x8_matrix_vinyl.c
 * Author: Phil Glazzard
 *
 * Created on 16 August 2016, 17:59
 */
/* This design drives five 8x8 common cathode LED matrices
 * via MAX7219 constant current drivers using SPI on the
 * PIC 16f690 microcontroller
 */
/* PIC pin assignment
 * SDO pin 9 (serial data out} drives DIN of MAX7219
 * SCK pin 11 (serial clock) drives CLK of MAX7219
 * RC6 pin 8 (load) drives CS of MAX7219
 */

#include <xc.h>
// CONFIG
#pragma config FOSC = INTRCIO   // Oscillator Selection bits
#pragma config WDTE = OFF       // Watchdog Timer Enable bit
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select bit
#pragma config CP = OFF         // Code Protection bit
#pragma config CPD = OFF        // Data Code Protection bit
#pragma config BOREN = OFF      // Brown-out Reset Selection bits (BOR disabled)
#pragma config IESO = OFF       // Internal External Switchover bit
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit
#define _XTAL_FREQ 4000000      // 4MHz internal clock
#define SD0 PORTCbits.RC7       // data in 16f690 pin 9
```

```
#define LOAD PORTCbits.RC6      // load 16f690 pin 8

#define SCK PORTBbits.RB6       // clock 16f690 pin 11

//**********************CONSTANTS*********************************************

#define normal_operation_addr 0x0C       //normal operation mode hi byte

#define normal_operation_data 0x01       //normal operation mode lo byte

#define decode_off_addr 0x09             // no BCD decoding needed hi byte

#define decode_off_data 0x00             // no BCD decoding needed lo byte

#define scan_limit_addr 0x0B             // display 8 columns of leds hi byte

#define scan_limit_data 0x07             // display 8 columns of leds lo byte

#define display_intensity_addr 0x0A      // minimum display intensity hi byte

#define display_intensity_data 0x00      // minimum display intensity lo byte

#define no_op_addr 0x00                  // No operation address

#define no_op_data 0x00                  // No operation data

/***************STRING MESSAGE TO DISPLAY*************************************/

#define MESS_CHAR 41                     // number of characters in message array (including spaces and symbols)+1
FOR NULL CHARACTER

//const char message[37] = "vinyl available at rear of store # ^ "; // string of characters stored in EEPROM

//const char message[MESS_CHAR] = "huge range of vinyl albums in store # ^ ";

const char message[] =

{

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space

0x00,0x7F,0x7F,0x04,0x04,0x7C,0x78,0x00, // h

0x3c,0x7c,0x40,0x40,0x7c,0x7c,0x00,0x00, // u

0x98,0xBC,0xA4,0xA4,0xFC,0x7C,0x00,0x00, // g

0x38,0x7C,0x54,0x54,0x5C,0x18,0x00,0x00, // e
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x44,0x7c,0x78,0x4c,0x04,0x1c,0x18,0x00, // r
0x00,0x74,0x54,0x54,0x7c,0x78,0x00,0x00, // a
0x7C,0x7C,0x04,0x04,0x7C,0x78,0x00,0x00, // n
0x98,0xBC,0xA4,0xA4,0xFC,0x7C,0x00,0x00, // g
0x38,0x7C,0x54,0x54,0x5C,0x18,0x00,0x00, // e
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x38,0x7C,0x44,0x44,0x7C,0x38,0x00,0x00, // o
0x48,0x7E,0x7F,0x49,0x03,0x02,0x00,0x00, // f
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x1c,0x3c,0x60,0x60,0x3c,0x1c,0x00,0x00, // v
0x00,0x44,0x7D,0x7D,0x40,0x00,0x00,0x00, // i
0x7C,0x7C,0x04,0x04,0x7C,0x78,0x00,0x00, // n
0x9c,0xbc,0xa0,0xa0,0xfc,0x7c,0x00,0x00, // y
0x00,0x41,0x7F,0x7F,0x40,0x00,0x00,0x00, // l
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x74,0x54,0x54,0x7c,0x78,0x00,0x00, // a
0x00,0x41,0x7F,0x7F,0x40,0x00,0x00,0x00, // l
0x00,0x7f,0x7f,0x48,0x48,0x78,0x30,0x00, // b
0x3c,0x7c,0x40,0x40,0x7c,0x7c,0x00,0x00, // u
0x7C,0x7C,0x18,0x38,0x1C,0x7C,0x78,0x00, // m
0x48,0x5c,0x54,0x54,0x74,0x24,0x00,0x00, // s
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x44,0x7D,0x7D,0x40,0x00,0x00,0x00, // i
0x7C,0x7C,0x04,0x04,0x7C,0x78,0x00,0x00, // n
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x48,0x5c,0x54,0x54,0x74,0x24,0x00,0x00, // s
0x00,0x04,0x3e,0x7f,0x44,0x24,0x00,0x00, // t
0x38,0x7C,0x44,0x44,0x7C,0x38,0x00,0x00, // o
```

```
0x44,0x7c,0x78,0x4c,0x04,0x1c,0x18,0x00, // r
0x38,0x7C,0x54,0x54,0x5C,0x18,0x00,0x00, // e
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x7e,0x81,0x95,0xa1,0xa1-,0x95,0x81,0x7e, //#smiley
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0xc0,0xff,0x7f,0x05,0x05,0x65,0x7f,0x3f, //^note
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //space
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};//space
/*********ASCII to 8x8 bit map LUT*******************************************/
#define ROWS 30
#define COLS 9
const char alphabet[ROWS][COLS] =       // 30 rows by 9 columns
{
   {'a', 0x00,0x74,0x54,0x54,0x7c,0x78,0x00,0x00},
   {'b', 0x00,0x7f,0x7f,0x48,0x48,0x78,0x30,0x00},
   {'c', 0x38,0x7C,0x44,0x44,0x6C,0x28,0x00,0x00},
   {'d', 0x30,0x78,0x48,0x48,0x7F,0x7F,0x00,0x00},
   {'e', 0x38,0x7C,0x54,0x54,0x5C,0x18,0x00,0x00},
   {'f', 0x48,0x7E,0x7F,0x49,0x03,0x02,0x00,0x00},
   {'g', 0x98,0xBC,0xA4,0xA4,0xFC,0x7C,0x00,0x00},
   {'h', 0x00,0x7F,0x7F,0x04,0x04,0x7C,0x78,0x00},
   {'i', 0x00,0x44,0x7D,0x7D,0x40,0x00,0x00,0x00},
   {'j', 0x00,0x40,0xC0,0x80,0xFD,0x70,0x00,0x00},
   {'k', 0x00,0x7F,0x7F,0x10,0x38,0x6C,0x44,0x00},
   {'l', 0x00,0x41,0x7F,0x7F,0x40,0x00,0x00,0x00},
```

```
  {'m', 0x7C,0x7C,0x18,0x38,0x1C,0x7C,0x78,0x00},
  {'n', 0x7C,0x7C,0x04,0x04,0x7C,0x78,0x00,0x00},
  {'o', 0x38,0x7C,0x44,0x44,0x7C,0x38,0x00,0x00},
  {'p', 0x00,0xFC,0xFC,0x24,0x24,0x3C,0x18,0x00},
  {'q', 0x18,0x3c,0x24,0x24,0xfc,0xfc,0x00,0x00},
  {'r', 0x44,0x7c,0x78,0x4c,0x04,0x1c,0x18,0x00},
  {'s', 0x48,0x5c,0x54,0x54,0x74,0x24,0x00,0x00},
  {'t', 0x00,0x04,0x3e,0x7f,0x44,0x24,0x00,0x00},
  {'u', 0x3c,0x7c,0x40,0x40,0x7c,0x7c,0x00,0x00},
  {'v', 0x1c,0x3c,0x60,0x60,0x3c,0x1c,0x00,0x00},
  {'w', 0x3c,0x7c,0x70,0x38,0x70,0x7c,0x3c,0x00},
  {'x', 0x44,0x6c,0x38,0x10,0x38,0x6c,0x44,0x00},
  {'y', 0x9c,0xbc,0xa0,0xa0,0xfc,0x7c,0x00,0x00},
  {'z', 0x4c,0x64,0x74,0x5c,0x4c,0x64,0x00,0x00},
  {'.', 0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x00},
  {'#', 0x7e,0x81,0x95,0xb1,0xb1,0x95,0x81,0x7e}, //#smiley
  {' ', 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, // space
  {'^', 0x08,0x08,0x08,0x2a,0x3e,0x1c,0x08,0x00}  //^arrow
};
void init_ports()
{
  //**********************ANSEL&COMPARATORS SETUP*****************************
  ANSEL = 0x00;              // AD module off
  ANSELH = 0x00;
  CM1CON0 = 0x00;            // Comparator module off
  CM2CON0 = 0x00;
  //**********************TRISbits SETUP**************************************
  TRISBbits.TRISB4 = 1;      // disable SDI facility as we don't want to
                             // receive data back from the slave
```

```
   TRISCbits.TRISC7 = 0;      // OUTPUT to Data in
   TRISCbits.TRISC6 = 0;      // OUTPUT to Load
   TRISBbits.TRISB6 = 0;      // OUTPUT to Clock
   TRISBbits.TRISB5 = 0;      // OUTPUT to diagnostic led
   PORTBbits.RB5 = 0;
/*************************************************************************/

}
void init_spi(void)
{
//***********************SSPCONbits*****************************************
   SSPCONbits.SSPM = 0000;    // SPI Master mode, clock = Fosc/4 = 1MHz
   SSPCONbits.CKP = 0;        // idle state for clock is low
   SSPCONbits.SSPEN = 1;      // enables seial port and configures SCK, SDO
                        //and SDI pins as serial port pins
   SSPCONbits.SSPOV = 0;      // no overflow allowed as we are in Master mode
   SSPCONbits.WCOL = 0;       // no collision
//***********************SSPSTATbits****************************************
   SSPSTATbits.CKE = 1;       // data transmitted on rising edge of clock
   SSPSTATbits.SMP = 1;       // input data sampled at end of output data
   PORTCbits.RC6 = 1;         // Disable Chip Select
 /*************************************************************************/
}

void send_spi_byte(char addr, char data)
{

        SSPBUF = addr;             // send 8 bit address MSB first
        while (!SSPSTATbits.BF)    // wait for 8 bits address data to transmit/ complete
```

```
        {
          ;             // do nothing
        }

      SSPBUF = data;          // send 8 bit data MSB first
      while (!SSPSTATbits.BF)   // wait for 8 bits data / transmit complete
        {
          ;             // do nothing
        }

  }

void send_no_op_spi_byte (void)
{

      SSPBUF = 0x00;          // send 8 bit address MSB first
      while (!SSPSTATbits.BF)   // wait for 8 bits address data to transmit/ complete
        {
          ;             // do nothing
        }

      SSPBUF = 0x00;          // send 8 bit data MSB first
      while (!SSPSTATbits.BF)   // wait for 8 bits data / transmit complete
        {
          ;             // do nothing
        }

  }
/*  initialise matrix 1 */
```

```
void init1_max7219()

{

   PORTCbits.RC6 = 0;

   send_spi_byte (normal_operation_addr,normal_operation_data);//leave shutdown mode and enter normal
operation

   PORTCbits.RC6 = 1;



   PORTCbits.RC6 = 0;

   send_spi_byte (display_intensity_addr,display_intensity_data);// minimum display intensity 1/32

   PORTCbits.RC6 = 1;



   PORTCbits.RC6 = 0;

   send_spi_byte (decode_off_addr,decode_off_data);    // decode mode off

   PORTCbits.RC6 = 1;



   PORTCbits.RC6 = 0;

   send_spi_byte (scan_limit_addr,scan_limit_data); // scan limit = 8 digits multiplexed

   PORTCbits.RC6 = 1;         // LOAD is high

}



/*   initialise matrix 2 */

void init2_max7219()

{

   PORTCbits.RC6 = 0;

   send_spi_byte (normal_operation_addr,normal_operation_data);//leave shutdown mode and enter normal
operation

   send_no_op_spi_byte();

   PORTCbits.RC6 = 1;



   PORTCbits.RC6 = 0;
```

```
   send_spi_byte (display_intensity_addr,display_intensity_data);// minimum display intensity 1/32
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (decode_off_addr,decode_off_data);    // decode mode off
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (scan_limit_addr,scan_limit_data); // scan limit = 8 digits multiplexed
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;         // LOAD is high
}
/*   initialise matrix 3 */
void init3_max7219()
{
   PORTCbits.RC6 = 0;
   send_spi_byte (normal_operation_addr,normal_operation_data);//leave shutdown mode and enter normal
operation
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (display_intensity_addr,display_intensity_data);// minimum display intensity 1/32
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;
```

```
   PORTCbits.RC6 = 0;
   send_spi_byte (decode_off_addr,decode_off_data);    // decode mode off
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (scan_limit_addr,scan_limit_data); // scan limit = 8 digits multiplexed
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;         // LOAD is high
}
/*   initialise matrix 4 */
void init4_max7219()
{
   PORTCbits.RC6 = 0;
   send_spi_byte (normal_operation_addr,normal_operation_data);//leave shutdown mode and enter normal
operation
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (display_intensity_addr,display_intensity_data);// minimum display intensity 1/32
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
```

```
   PORTCbits.RC6 = 1;


   PORTCbits.RC6 = 0;
   send_spi_byte (decode_off_addr,decode_off_data);    // decode mode off
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;


   PORTCbits.RC6 = 0;
   send_spi_byte (scan_limit_addr,scan_limit_data); // scan limit = 8 digits multiplexed
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;         // LOAD is high
}


/*   initialise matrix 5 */
void init5_max7219()
{
   PORTCbits.RC6 = 0;
   send_spi_byte (normal_operation_addr,normal_operation_data);//leave shutdown mode and enter normal
operation
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;
```

```
   PORTCbits.RC6 = 0;
   send_spi_byte (display_intensity_addr,display_intensity_data);// minimum display intensity 1/32
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (decode_off_addr,decode_off_data);    // decode mode off
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;

   PORTCbits.RC6 = 0;
   send_spi_byte (scan_limit_addr,scan_limit_data); // scan limit = 8 digits multiplexed
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   send_no_op_spi_byte();
   PORTCbits.RC6 = 1;         // LOAD is high
}
/********Function reads the ith ascii character from message string and returns this ascii character */
char get_char (char i )
{
   char *char_ptr;            // declare a pointer to char which will read characters from message
   char_ptr = (void*)&message[i];     // pointer points to the ith element of message
```

```
   return *char_ptr;          // returns the ascii character found at &message[i]
}
/*****************************************************************************************
************/

/*********************Function matches ascii character in LUT and returns 8x8
bitmap*****************/

char check_null_char(char i)
{
   char n;
   if(get_char(i) == '\0')     // if we run out of characters in message[], clear n
        {
           n = 0;
        }
   return n;
}
void main(void)
{
/**********INITIALISATION OF PORT PINS, SPI REGISTERS & MAX7219 CHIP***********/
   init_ports();                  // initialise PORTC pins 8, 9,11
   init_spi();                   // initialise SPI registers
   init1_max7219();               // initialise all 5 MAX7219 chips
   init2_max7219();               // individually
   init3_max7219();
   init4_max7219();
   init5_max7219();
/*****************************************************************************/

   char *pm5, *pm4, *pm3, *pm2, *pm1;  // declare five pointers to point to five
```

```
char matrix_5, matrix_4, matrix_3, matrix_2, matrix_1;// consecutive characters
int cnt;                                  // in message[]
char x;
pm5 = (void*)&message[0];       // pm5 (pointer matrix 5) points to address of 1st char
pm4 = (void*)&message[8];       // pm4 points to 2nd char in message[]
pm3 = (void*)&message[16];      // pm3 points to 3rd char in message[]
pm2 = (void*)&message[24];      // pm2 points to 2nd char in message[]
pm1 = (void*)&message[32];      // pm1 points to 1st char in message[]

while(1)
{
   for ( cnt = 0; cnt < 392; cnt++)    //this loop effectively pulls the message[]
   {                            // left across the five 8x8 matrices to create
                                // the scrolling message effect
      for (x = 1; x < 9; x++)        // this loop writes the data the pointer points
      {                         // to in the appropriate 8x8 matrix
         matrix_5 = *pm5;
         PORTCbits.RC6 = 0;
         send_spi_byte(x, matrix_5);
         send_no_op_spi_byte();
         send_no_op_spi_byte();
         send_no_op_spi_byte();
         send_no_op_spi_byte();
         PORTCbits.RC6 = 1;
         pm5 = pm5 + 1;             // pointer now points to next character to the right

         matrix_4 = *pm4;
         PORTCbits.RC6 = 0;
         send_no_op_spi_byte();
```

```
send_spi_byte(x, matrix_4);
send_no_op_spi_byte();
send_no_op_spi_byte();
send_no_op_spi_byte();
PORTCbits.RC6 = 1;
pm4 = pm4 + 1;

matrix_3 = *pm3;
PORTCbits.RC6 = 0;
send_no_op_spi_byte();
send_no_op_spi_byte();
send_spi_byte(x, matrix_3);
send_no_op_spi_byte();
send_no_op_spi_byte();
PORTCbits.RC6 = 1;
pm3 = pm3 + 1;

matrix_2 = *pm2;
PORTCbits.RC6 = 0;
send_no_op_spi_byte();
send_no_op_spi_byte();
send_no_op_spi_byte();
send_spi_byte(x, matrix_2);
send_no_op_spi_byte();
PORTCbits.RC6 = 1;
pm2 = pm2 + 1;

matrix_1 = *pm1;
PORTCbits.RC6 = 0;
```

```
            send_no_op_spi_byte();
            send_no_op_spi_byte();
            send_no_op_spi_byte();
            send_no_op_spi_byte();
            send_spi_byte(x, matrix_1);
            PORTCbits.RC6 = 1;
            pm1 = pm1 + 1;
          }
          __delay_ms(40);                  // delay sets speed of scrolling
          pm5 = (void*)&message[cnt];        // these offsets to the address of each character
          pm4 = (void*)&message[cnt + 8];     // allow the scrolling feature
          pm3 = (void*)&message[cnt + 16];
          pm2 = (void*)&message[cnt + 24];
          pm1 = (void*)&message[cnt + 32];
          if ((void*)*pm1 == "\0")          // when the pointer pm1 finds the null character
          {                            // at the end of message[], it resets all pointer
            pm5 = (void*)&message[0];      // addresses and clears counter variable cnt
            pm4 = (void*)&message[8];
            pm3 = (void*)&message[16];
            pm2 = (void*)&message[24];
            pm1 = (void*)&message[32];
            cnt = 0;
          }
        }
      }
    }
```