EEPROM Manager

Software Modules

ABSTRACT

This routine implements the SPI interface to a Microchip 25xx serial EEPROM.
It can be modified easily as needed.

Contents

1 Introduction

This routine relies on a simple blocking SPI interface, which allows it to be adapted as needed by the user. It also buffers user data into a section of RAM.

2 Software Interface

The eeprom_write function writes the data directly into a buffer that is later taken and written to EEPROM. The eeprom_read function reads the data directly from EEPROM. Finally the eeprom_upate function is periodically called, which flushes the buffer from RAM to EEPROM.

2.1 Board Support Libraries

The archive for this software contains all the necessary header files to enable the code and run the functions: eeprom.c and eeprom.h

2.2 Header File – 'eeprom_cfg.h'

```
/*****************************************************************/
/* THE BELOW DEFINES SHOULD BE CONFIGURED TO MATCH YOUR EEPROM   */
/*****************************************************************/
/* this defines the size of the EEPROM in bytes */
#define EEPROM_CHIP_SIZE       2048
/* this defines the size of the EEPROM's internal page in bytes. */
#define EEPROM_PAGE_SIZE        16

/*****************************************************************/
/* THE BELOW DEFINES SHOULD BE CONFIGURED TO YOUR APPLICATION    */
/*****************************************************************/
/* this defines size of the application's buffer */
#define EEPROM_BUF_SIZE         40
```

2.3 Header File – 'eeprom.h'

```
/*
** Simma Software, Inc.
** https://www.simmasoftware.com
*/
#ifndef __EEPROM_H__
#define __EEPROM_H__
#include <stdint.h>

extern void
eeprom_update( void );

extern uint8_t
eeprom_read( uint16_t addr, uint8_t *buf, uint16_t buf_len );

extern uint8_t
eeprom_write( uint16_t addr, uint8_t *buf, uint16_t buf_len );

#endif
```

2.4 Header File –'bits.h'

```
#ifndef __BITS_H__
#define __BITS_H__

/* bit definitions */
#define B0  (0x000000001)
#define B1  (0x000000002)
#define B2  (0x000000004)
#define B3  (0x000000008)
#define B4  (0x000000010)
#define B5  (0x000000020)
#define B6  (0x000000040)
#define B7  (0x000000080)
#define B8  (0x000000100)
#define B9  (0x000000200)
#define B10 (0x000000400)
#define B11 (0x000000800)
#define B12 (0x000001000)
#define B13 (0x000002000)
#define B14 (0x000004000)
#define B15 (0x000008000)
#define B16 (0x000010000)
#define B17 (0x000020000)
#define B18 (0x000040000)
#define B19 (0x000080000)
#define B20 (0x000100000)
#define B21 (0x000200000)
#define B22 (0x000400000)
```

```
#define B23 (0x000800000)
#define B24 (0x001000000)
#define B25 (0x002000000)
#define B26 (0x004000000)
#define B27 (0x008000000)
#define B28 (0x010000000)
#define B29 (0x020000000)
#define B30 (0x040000000)
#define B31 (0x080000000)

#endif
```

2.5 Source Code – 'eeprom.c'

```
/*
** Simma Software, Inc.
** https://www.simmasoftware.com
** eeprom.c
**
** Implements SPI an interface to Microchip 25xx serial EEPROM chips.
** This routine relies on a simple blocking SPI interface and can be
** easily adapted as needed.  The transmit portion buffers data into
** a section in RAM.  Periodically call the eeprom_update() function
** to write buffered data to the serial SPI EEPROM.
*/
#include <stdint.h>
#include "bits.h"
#include "spi.h"
#include "eeprom.h"
#include "eeprom_cfg.h"



/* the mask size depends on page size */
#if EEPROM_PAGE_SIZE == 16
#define EEPROM_PAGE_MASK        0xf
#elif EEPROM_PAGE_SIZE == 32
#define EEPROM_PAGE_MASK        0x1f
#else
#error unsupported EEPROM page size
#endif


/* commands to eeprom */
#define EEPROM_READ         (B1 | B0)
#define EEPROM_WRITE            (B1)
```

```
#define EEPROM_WRDI             (B2)
#define EEPROM_WREN        (B2 | B1)
#define EEPROM_RDSR        (B2 | B0)
#define EEPROM_WRSR             (B0)



/* this is one self contained buffer,
   we will probably used an arrar of these */
typedef struct {

  uint16_t addr;
  uint8_t buf[ EEPROM_BUF_SIZE ];
  uint16_t len;
  uint16_t index;

} eeprom_buf_t;


eeprom_buf_t eeprom_buf;


/*
** Writes data directly to buffer, which is later written to EEPROM.
** INPUT: addr - EEPROM address
**        buf - buffer where EEPROM data is read from
**        buf_len - amount of data to write
** RETURN 0 - success
**        1 - fail
*/
uint8_t
eeprom_write ( uint16_t addr, uint8_t *buf, uint16_t buf_len )
{
  uint8_t cnt;

  /* return failure if any address exceeds chip's address space */
  if( ((addr + buf_len) > EEPROM_CHIP_SIZE)
      || (buf_len > EEPROM_BUF_SIZE) )
    return 1;

  if( eeprom_buf.len )
    return 1;

  eeprom_buf.len = buf_len;
```

```
  eeprom_buf.addr = addr;
  eeprom_buf.index = 0;

  /* copy over data */
  for( cnt = 0; cnt < buf_len; cnt++ )
    eeprom_buf.buf[ cnt ] = buf[ cnt ];

  return 0;
}



/*
** Reads directly from EEPROM.
** INPUT: addr - EEPROM address
**        buf - buffer where EEPROM data is placed
**        buf_len - amount of data to read
** RETURN 0 - success
**        1 - fail
*/
uint8_t
eeprom_read ( uint16_t addr, uint8_t *buf, uint16_t buf_len )
{
  uint8_t ret;

  /* bail if no buffer */
  if( !buf )
    return 1;

  /* read status register */
  spi_select_eeprom();
  spi_tx( EEPROM_RDSR );
  ret = spi_rx();
  spi_deselect();

  /* is the serial eeprom busy? */
  if( (ret & B0) == 0 ) {

    /* put it in read mode */
    spi_select_eeprom();
    spi_tx( EEPROM_READ );

    /* send address */
    spi_tx( addr >> 8 );
```

```
    spi_tx( addr );

    /* fill buffer */
    for( ret = 0; ret < buf_len; ret++ )
      buf[ ret ] =  spi_rx();

    spi_deselect();

  } else {

    return 1;
  }

  return 0;
}



/*
** Periodic update function which flushes buffer to EEPROM.
*/
void
eeprom_update ( void )
{
  uint8_t tmp, cnt, write_len;

  /* if there isn't anything to write, then don't do anything */
  if( eeprom_buf.len == 0 )
    return;

  /* find out a couple things */
  spi_select_eeprom();
  spi_tx( EEPROM_RDSR );
  tmp = spi_rx();
  spi_deselect();

  /* is the chip busy */
  if( (tmp & B0) != 0 )
    return;

  /* write enable the device */
  spi_select_eeprom();
  spi_tx( EEPROM_WREN );
  spi_deselect();
```

```
  /* is it block protected or write pin enabled? */
  if( tmp & (B3 | B2) ) {

    spi_select_eeprom();
    spi_tx( EEPROM_WRSR );
    spi_tx( 0 );
    spi_deselect();

  } else {

    /* write the starting addr to chip */
    spi_select_eeprom();
    spi_tx( EEPROM_WRITE );
    spi_tx( eeprom_buf.addr >> 8 );
    spi_tx( eeprom_buf.addr );

    /* data can only be written until the end of the eeprom's page.
       the page is either 16 or 32 bytes big (refer to datasheet).
       this statement gives us our max write length */
    write_len = EEPROM_PAGE_SIZE - (eeprom_buf.addr & EEPROM_PAGE_MASK);

    /* if our buffer is less than the max length, then adjust max */
    if( eeprom_buf.len < write_len )
      write_len = eeprom_buf.len;

    /* write all the data we are allowed to */
    for( cnt = 0; cnt < write_len; cnt++ )
      spi_tx( eeprom_buf.buf[eeprom_buf.index+cnt] );

    /* there might still be data left to write, so adjust the variables
       to reflect the amount of information that was just written */
    eeprom_buf.len -= cnt;
    eeprom_buf.index += cnt;
    eeprom_buf.addr += cnt;

    spi_deselect();
  }

  return;
}
```

3 References

1. **Microchip 25xx serial EEPROM Family Datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/22040A.pdf**

4 About Simma Software, the SAE J1939 and UDS Experts

Simma Software, Inc. specializes in real-time embedded software for the automotive industry. Products and services include protocol stacks, bootloaders, device drivers, training, and consultation on the following technologies: J1939, CAN, CAN FD, J1587, J1708, J2497, J1922, J1979, ISO 15765, OBD-II, CANopen, UDS, XCP, NMEA2000, and Secure Boot.