```c
//dash.c

#include "dash.h"
#include <util/delay.h>

int main(void)
{
    //Initialize Button Task
    xTaskCreate(vTaskButtons,BTN_TASK_NAME,BTN_TASK_STACK_SIZE,BTN_TASK_PARAM,BTN_TASK_PRIORITY
,&xTaskHandleButtons);
    //Initialize User Interface Task
    xTaskCreate(vTaskUi,UI_TASK_NAME,UI_TASK_STACK_SIZE,UI_TASK_PARAM,UI_TASK_PRIORITY,&
xTaskHandleUi);
    //Initialize LCD Task
    xTaskCreate(vTaskLcd,LCD_TASK_NAME,LCD_TASK_STACK_SIZE,LCD_TASK_PARAM,LCD_TASK_PRIORITY,&
xTaskHandleLcd);
    //Initialize LED Task
    xTaskCreate(vTaskLeds,LED_TASK_NAME,LED_TASK_STACK_SIZE,LED_TASK_PARAM,LED_TASK_PRIORITY,&
xTaskHandleLeds);
    //Initialize RPM Task
    xTaskCreate(vTaskRpm,DATA_TASK_NAME,DATA_TASK_STACK_SIZE,DATA_TASK_PARAM,DATA_TASK_PRIORITY
,&xTaskHandleData);
    //Initialize CAN Com Task Task
    xTaskCreate(vTaskCom,DATA_COM_NAME,DATA_COM_STACK_SIZE,DATA_COM_PARAM,DATA_COM_PRIORITY,&
xTaskHandleCom);

    //Queues (See header for parameter Definitions and Queue message structures
    xQueueButtonEvents =    xQueueCreate(BTN_EVTS_BFRSIZE,BTN_EVTS_MSGSIZE);
    xQueueLcdEvents =       xQueueCreate(LCD_EVTS_BFRSIZE,LCD_EVTS_MSGSIZE);
    xQueueLedEvents =       xQueueCreate(LED_EVTS_BFRSIZE,LED_EVTS_MSGSIZE);
    xQueueRpmEvents =       xQueueCreate(RPM_EVTS_BFRSIZE,RPM_EVTS_MSGSIZE);

    //Semaphore and Mutex Creation
    xMtxGlobalData=xSemaphoreCreateMutex();
    vSemaphoreCreateBinary(xSemUpdateData);
    vSemaphoreCreateBinary(xSemCan);

    //Configure External Interrupts
    EICRA |=(1<<ISC10)|(1<<ISC11);//|(1<<ISC00)|(1<<ISC01);

    //Enable External Interrupts
    EIMSK |= (1<<INT1)|(1<<INT0);

    //Start the Real time scheduler
    vTaskStartScheduler();
}
```

```
//dash.h
#ifndef DASH_H
#define DASH_H

#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

#include <string.h>

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

//FreeRTOS include files
#include"FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "lcd.h"
#include "mcp2515.h"
#include "IMCP_Defs2.h"

//-----PIN SETTINGS---------
//outputs
#define MCP_CS_PORT PORTB
#define MCP_CS_PIN PINB4
#define MCP_CS_DDR DDRB

//inputs
#define MCP_RX0BF_PORT PIND
#define MCP_RX0BF_PIN PIND2
#define MCP_RX0BF_DDR DDRD

#define MCP_RX1BF_PORT PINB
#define MCP_RX1BF_PIN PB1
#define MCP_RX1BF_DDR DDRB

//IO Defs
#define SPI_PORT    PORTB
#define SPI_DDR     DDRB
#define SPI_SCK     PB7
#define SPI_MISO    PB6
#define SPI_MOSI    PB5
#define SPI_SS      MCP_CS_PIN

//General Macros
#define SetBit(port,pin)    (port|=(_BV(pin)))
#define ClrBit(port,pin)    (port&=~(_BV(pin)))
#define TglBit(port,pin)    (port^=(_BV(pin)))
```

```
#define BitIsSet(port,pin)  (port&_BV(pin))
#define BitIsClr(port,pin)  (!BitIsSet(port,pin))




//Tasks

//Button Handling
void vTaskButtons(void *pvParameters);
    //Initialization Parameters
    xTaskHandle xTaskHandleButtons;
    #define BTN_TASK_NAME       (const signed char * const)("Buttons")
    #define BTN_TASK_STACK_SIZE configMINIMAL_STACK_SIZE
    #define BTN_TASK_PARAM      0
    #define BTN_TASK_PRIORITY   1
    #define BTN_DB_DELAY        50
    #define BTN_LOOP_DELAY      10
    #define BTN_HELD_DELAY      2000
    #define BTN_HELD_COUNT      (BTN_HELD_DELAY/BTN_LOOP_DELAY)

    #define BTN_PUSHED          1
    #define BTN_HELD            2

    //Supporting Functions
    uint8_t CheckButton(volatile uint8_t *bport,uint8_t bpin);

//Lcd Control
void vTaskLcd(void *pvParameters);
    //Initialization Parameters
    xTaskHandle xTaskHandleLcd;
    #define LCD_TASK_NAME       (const signed char * const)("Lcd")
    #define LCD_TASK_STACK_SIZE configMINIMAL_STACK_SIZE
    #define LCD_TASK_PARAM      0
    #define LCD_TASK_PRIORITY   1
    #define LCD_LOOP_DELAY      100

//User Interface task
void vTaskUi(void *pvParameters);
    //Initialization Parameters
    xTaskHandle xTaskHandleUi;
    #define UI_TASK_NAME        (const signed char * const)("Ui")
    #define UI_TASK_STACK_SIZE  configMINIMAL_STACK_SIZE+200
    #define UI_TASK_PARAM       0
    #define UI_TASK_PRIORITY    2




void vTaskLeds(void *pvParameters);
    //Initialization Parameters
        xTaskHandle xTaskHandleLeds;
        #define LED_TASK_NAME       (const signed char * const)("Leds")
        #define LED_TASK_STACK_SIZE configMINIMAL_STACK_SIZE
        #define LED_TASK_PARAM      0
        #define LED_TASK_PRIORITY   1

void vTaskRpm(void *pvParameters);
    //Initialization Parameters
        xTaskHandle xTaskHandleData;
    #define DATA_TASK_NAME      (const signed char * const)("Data")
    #define DATA_TASK_STACK_SIZE    configMINIMAL_STACK_SIZE
    #define DATA_TASK_PARAM     0
    #define DATA_TASK_PRIORITY  3

    void UpdateUiData(void);

    //Page Functions
    void PageMain(void);
    void PageGauges(void);
    void PageDriver(void);
    void PageEcu(void);
    void PageEcuData(void);
    void PageTrsp(void);
    void PageTrspSusp(void);
    void PageTrspTires(void);
    void PageTcm(void);
    void PageTcmData(void);
    void PageTcmRpms(void);
    void PageSysInfo(void);
    void PageDaq(void);
    void PageGps(void);
    void PageGpsData(void);
    void PageGpsTime(void);

    //void vCanCom(void *pvParameters)

    void vTaskCom(void *pvParameters);
    //Initialization Parameters
        xTaskHandle xTaskHandleCom;
    #define DATA_COM_NAME       (const signed char * const)("COM")
    #define DATA_COM_STACK_SIZE configMINIMAL_STACK_SIZE
    #define DATA_COM_PARAM      0
    #define DATA_COM_PRIORITY   4

//Queues

//Button Event Queue
xQueueHandle xQueueButtonEvents;
    //Queue Initialization Defs
    #define BTN_EVTS_MSGSIZE        1
    #define BTN_EVTS_BFRSIZE        1

    //Message Protocol
    #define BTN_UP_PUSHED       1
    #define BTN_UP_HELD         2
    #define BTN_DOWN_PUSHED     3
    #define BTN_DOWN_HELD       4
    #define BTN_LEFT_PUSHED     5
    #define BTN_LEFT_HELD       6
    #define BTN_RIGHT_PUSHED    7
    #define BTN_RIGHT_HELD      8
    #define BTN_CENTER_PUSHED   9
    #define BTN_CENTER_HELD     10
    #define BTN_1_PUSHED        11
    #define BTN_1_HELD          12
    #define BTN_2_PUSHED        13
    #define BTN_2_HELD          14
    #define BTN_3_PUSHED        15
    #define BTN_3_HELD          16
    #define BTN_4_PUSHED        17
    #define BTN_4_HELD          18

//Lcd Event Queue
xQueueHandle xQueueLcdEvents;
    //Lcd Event Structure
    typedef struct LCD_EVENT_t
    {
        uint8_t event;
        uint8_t x;
        uint8_t y;
        char str[21];
    } LCD_EVENT;
    //Queue Initialization Defs
    #define LCD_EVTS_MSGSIZE sizeof(LCD_EVENT)
    #define LCD_EVTS_BFRSIZE 1
    //Event Define
    #define LCD_WRITE           0
    #define LCD_CLEAR_LINE      1
    #define LCD_CLEAR_SCREEN    2
    #define LCD_SCREEN_ON       3
    #define LCD_SCREEN_OFF      4
    #define LCD_REINIT          5

//Rpm Events
xQueueHandle xQueueRpmEvents;
    //Rpm Event Structure
    typedef struct RPM_EVENT_t
    {
        uint32_t last_us,last_ms;
        uint16_t curr_ms,curr_us;
    } RPM_EVENT;
    //Queue Initialization Defs
    #define RPM_EVTS_MSGSIZE sizeof(RPM_EVENT)
    #define RPM_EVTS_BFRSIZE 4

xQueueHandle xQueueLedEvents;
    //LED Event Structure
    typedef struct LED_EVENT_t
    {
        uint8_t led;
        uint16_t period;
    }LED_EVENT;

    //Queue Initialization Defs
    #define LED_EVTS_MSGSIZE sizeof(LED_EVENT)
    #define LED_EVTS_BFRSIZE 5

    //Period Defs
    #define LED_OFF 0
    #define LED_ON  0xFF



xSemaphoreHandle xSemUpdateData;
xSemaphoreHandle xMtxGlobalData;

typedef struct CAR_DATA_t
{
    //Driver
    uint8_t driver_num;

    //RPM
    uint16_t Rpm;
        uint8_t  Rpm_Gauge;

    uint8_t Mph;

    uint8_t can_status;

    //Ecu

        uint8_t ecu_status;

        //Data
        uint8_t brk_ps_r;
        uint8_t brk_ps_l;
        uint8_t thr_pos;
        uint8_t water_tmp;
        uint8_t air_tmp;
        uint8_t batt_volt_raw;

    //DAQ

        uint8_t Daq_Status;

    //Traction Control

        uint8_t traction_control_status;


            //Front
            uint16_t fr_rpm;
            uint16_t fl_rpm;
            uint8_t swag;

            //Rear
            uint16_t rr_rpm;
            uint16_t rl_rpm;
            uint16_t slip;

            //STSLP
            uint8_t slip_ref;

            //Launch Control
            uint8_t lct;

    //Gps
        uint8_t gps_status;

        //Latitude
        uint8_t lat_d1;
        uint8_t lat_do;
        uint8_t lat_m3;
        uint8_t lat_m2;
        uint8_t lat_m1;
        uint8_t lat_m0;

        //Longitude
        uint8_t lng_d1;
        uint8_t lng_do;
        uint8_t lng_m3;
        uint8_t lng_m2;
        uint8_t lng_m1;
        uint8_t lng_m0;

        //Date and Time
        uint8_t day;
        uint8_t month;
        uint8_t year;
        uint8_t hour;
        uint8_t min;
        uint8_t sec;

    //Tires and Suspension

        uint8_t trsp_status;

        //Front Left Temps
        uint16_t fl_tmp_otr;
        uint16_t fl_tmp_mid;
        uint16_t fl_tmp_inr;

        //Front Right Temps
        uint16_t fr_tmp_otr;
        uint16_t fr_tmp_mid;
        uint16_t fr_tmp_inr;
        //Back Left Temps
        uint16_t rl_tmp_otr;
        uint16_t rl_tmp_mid;
        uint16_t rl_tmp_inr;

        //Back Right Temps
        uint16_t rr_tmp_otr;
        uint16_t rr_tmp_mid;
        uint16_t rr_tmp_inr;

        //Suspension Data
        uint16_t fl_susp;
        uint16_t fr_susp;
        uint16_t rr_susp;
        uint16_t rl_susp;
}CAR_DATA;

//Can Message Queue
xSemaphoreHandle xSemCan;

typedef struct CAN_EVENT_t
{
    uint8_t Cmd;
    uint8_t Arg2;
    uint8_t Arg3;
    uint8_t Arg4;
    uint8_t Arg5;
    uint8_t Arg6;
    uint8_t Arg7;
}CAN_EVENT;

typedef struct RPM_STUFF_t
{
    uint8_t Command;
    uint16_t rpm;
    uint16_t timestamp1;
    uint16_t timestamp2;
    uint8_t filler;
}rstuff;

rstuff data_packet;


#endif
```

```c
//dash_buttons.c
#include "dash.h"

#define BTN_UP_PORT         PORTC
#define BTN_UP_DDR          DDRC
#define BTN_UP_PIN          PINC
#define BTN_UP              PC1
#define BTN_UP_EVENT        1

#define BTN_DOWN_PORT   PORTC
#define BTN_DOWN_DDR    DDRC
#define BTN_DOWN_PIN    PINC
#define BTN_DOWN        PC0

#define BTN_LEFT_PORT   PORTD
#define BTN_LEFT_DDR    DDRD
#define BTN_LEFT_PIN    PIND
#define BTN_LEFT        PD7

#define BTN_RIGHT_PORT  PORTC
#define BTN_RIGHT_DDR   DDRC
#define BTN_RIGHT_PIN   PINC
#define BTN_RIGHT       PC3

#define BTN_CENTER_PORT     PORTC
#define BTN_CENTER_DDR      DDRC
#define BTN_CENTER_PIN      PINC
#define BTN_CENTER          PC2

#define BTN_1_PORT      PORTD
#define BTN_1_DDR       DDRD
#define BTN_1_PIN       PIND
#define BTN_1           PD5

#define BTN_2_PORT      PORTD
#define BTN_2_DDR       DDRD
#define BTN_2_PIN       PIND
#define BTN_2           PD6

#define BTN_3_PORT      PORTC
#define BTN_3_DDR       DDRC
#define BTN_3_PIN       PINC
#define BTN_3           PC4

#define BTN_4_PORT      PORTC
#define BTN_4_DDR       DDRC
#define BTN_4_PIN       PINC
#define BTN_4           PC5

void vTaskButtons(void *pvParameters)
{
    //Initialize IO
        //Set as Input: DDR=0
        //PU Resistor Off: PORT=0
    ClrBit(BTN_UP_DDR,BTN_UP);
    ClrBit(BTN_UP_PORT,BTN_UP);
    ClrBit(BTN_DOWN_DDR,BTN_DOWN);
    ClrBit(BTN_DOWN_PORT,BTN_DOWN);
    ClrBit(BTN_LEFT_DDR,BTN_LEFT);
    ClrBit(BTN_LEFT_PORT,BTN_LEFT);
    ClrBit(BTN_RIGHT_DDR,BTN_RIGHT);
    ClrBit(BTN_RIGHT_PORT,BTN_RIGHT);
    ClrBit(BTN_CENTER_DDR,BTN_CENTER);
    ClrBit(BTN_CENTER_PORT,BTN_CENTER);
    ClrBit(BTN_1_DDR,BTN_1);
    ClrBit(BTN_1_PORT,BTN_1);
    ClrBit(BTN_2_DDR,BTN_2);
    ClrBit(BTN_2_PORT,BTN_2);
    ClrBit(BTN_3_DDR,BTN_3);
    ClrBit(BTN_3_PORT,BTN_3);
    ClrBit(BTN_4_DDR,BTN_4);
    ClrBit(BTN_4_PORT,BTN_4);

    uint8_t temp=0;
    uint8_t bevt=0;

    while(1)
    {
        //For each Button
        temp=CheckButton(&BTN_UP_PIN,BTN_UP);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_UP_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_UP_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_UP_PIN,BTN_UP))  vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_DOWN_PIN,BTN_DOWN);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_DOWN_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_DOWN_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_DOWN_PIN,BTN_DOWN))  vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);
            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_LEFT_PIN,BTN_LEFT);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_LEFT_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_LEFT_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_LEFT_PIN,BTN_LEFT))  vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_RIGHT_PIN,BTN_RIGHT);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_RIGHT_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_RIGHT_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_RIGHT_PIN,BTN_RIGHT))    vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_CENTER_PIN,BTN_CENTER);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_CENTER_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_CENTER_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_CENTER_PIN,BTN_CENTER))  vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }
        temp=CheckButton(&BTN_1_PIN,BTN_1);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_1_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_1_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_1_PIN,BTN_1))    vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_2_PIN,BTN_2);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_2_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_2_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_2_PIN,BTN_2))    vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }
        temp=CheckButton(&BTN_3_PIN,BTN_3);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_3_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_3_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_3_PIN,BTN_3))    vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }

        temp=CheckButton(&BTN_4_PIN,BTN_4);
        if(temp)
        {
            if(temp==BTN_PUSHED)    bevt=BTN_4_PUSHED;
            else if(temp==BTN_HELD)
            {
                bevt=BTN_4_HELD;
            }
            xQueueSendToBack(xQueueButtonEvents,&bevt,portMAX_DELAY);
            while(BitIsSet(BTN_4_PIN,BTN_4))    vTaskDelay(BTN_LOOP_DELAY);
            vTaskDelay(BTN_DB_DELAY);

            temp=0;
            bevt=0;
        }
    }
}


//Function that checks the state of the function and returns 0, PUSHED or HELD
uint8_t CheckButton(volatile uint8_t *bport,uint8_t bpin)
{
    if(BitIsSet((*bport),bpin))
    {
        vTaskDelay(BTN_DB_DELAY);
        if(BitIsSet((*bport),bpin))
        {
            for(uint8_t i=0;i<BTN_HELD_COUNT;i++)
            {
                if(BitIsClr((*bport),bpin))
                {
                    return BTN_PUSHED;
                }
                vTaskDelay(BTN_LOOP_DELAY);
            }
            return BTN_HELD;
        }
    }
    return 0;
}
```

```c
//dash_com.c

#include "dash.h"

extern volatile CAR_DATA gData;
uint8_t tx_msg[8];
void vTaskCom(void *pvParameters)
{
            LED_EVENT ledevt;


    //Spi Init
    SPI_PORT |= (1<<SPI_MISO)|(1<<MCP_CS_PIN)|(1<<SPI_MOSI)|(1<<SPI_SCK);
    SPI_DDR |= (1<<SPI_MOSI)|(1<<SPI_SCK)|(1<<MCP_CS_PIN);
    SPI_DDR &= ~((1<<SPI_MISO));

    //Spi Configuration
    SPCR = 0x50;
    SPSR=0x01;

    //Can Intialization
    mcp_v_can_init(CAN_ADDR_DASH);

    //Configure and Enable External Interrupts
    EICRA |=(1<<ISC01);
    EIMSK |= (1<<INT0);

    MCP_RX0BF_p|=(1<<MCP_RX0BF_PIN);

    xSemaphoreTake(xSemCan,portMAX_DELAY);

    //Enable Global Interrupts
    sei();

    uint8_t rx_buf[13];

    uint8_t temp;

    CAN_EVENT can_evt;

    while(1)
    {
        //Wait for Semaphore signal from ISR
        xSemaphoreTake(xSemCan,portMAX_DELAY);

        while(BitIsClr(MCP_RX0BF_PORT,MCP_RX0BF_PIN))
        {
            //Receive the message into the buffer
            mcp_v_CANReceive(rx_buf);
            // data_packet.rpm=gData.Rpm;
            // data_packet.timestamp2=xTaskGetTickCount();

                tx_msg[1]=(uint8_t)((gData.Rpm&0xFF00)>>8);
                tx_msg[2]=(uint8_t)(gData.Rpm&0x00FF);
                uint16_t ts2=xTaskGetTickCount();
                tx_msg[5]=(uint8_t)((ts2&0xFF00)>>8);
                tx_msg[6]=(uint8_t)(ts2&0x00FF);


            mcp_v_CANSend(0,CAN_ADDR_WRL,55,7,tx_msg);
            // for(uint8_t i=0;i<8;i++)
            // {
                // mcp_v_CANReceive(rx_buf);
            // }
        }

    }
}

ISR(INT0_vect)
{
    uint16_t ts1=xTaskGetTickCount();
    tx_msg[3]=(uint8_t)((ts1&0xFF00)>>8);
    tx_msg[4]=(uint8_t)(ts1&0x00FF);


    uint8_t higher;
    xSemaphoreGiveFromISR(xSemCan,&higher);
}
```

```
//dash_data.c

#include "dash.h"

volatile RPM_EVENT isr_rpm;
volatile uint8_t rpm_cnt=0;
volatile uint16_t tick_cnt;
volatile uint8_t higher;
extern volatile CAR_DATA gData;
volatile uint16_t rpm_ms;
volatile uint16_t rpm_us;

// void vTaskDataAndControl(void *pvParameters)
// {
    // uint8_t *pData=(uint8_t *)(&gData);
    // xSemaphoreGive(xMtxGlobalData);

    // xSemaphoreGive(xSemUpdateData);

    // uint8_t bevt;
    // RPM_EVENT rpmevt;
    // uint32_t rpm_temp,rpm_temp2;
    // LED_EVENT ledevt;


        // vTaskDelay(5);
    // }


//}

void vTaskRpm(void *pvParameters)
{
    uint8_t bevt;
    RPM_EVENT rpmevt;
    uint32_t rpm_temp,rpm_temp2;
    LED_EVENT ledevt;
    uint32_t sum;

    uint16_t rpm_buffer[5];
    uint8_t head=0;


    while(1)
    {
        for(uint8_t i=0;i<10;i++)
        {
            if(xQueueReceive(xQueueRpmEvents,&rpmevt,portMAX_DELAY))
            {
                rpm_temp=rpmevt.curr_ms-rpmevt.last_ms;

                rpm_temp=(uint16_t)(((300000*103)/100)/((uint32_t)(rpm_temp)));
                sum=sum+rpm_temp;
                head++;
            }
        }
        sum=sum/11;
        xSemaphoreTake(xMtxGlobalData,portMAX_DELAY);
        gData.Rpm=(uint16_t)sum;
        xSemaphoreGive(xMtxGlobalData);
        xSemaphoreGive(xSemUpdateData);
        vTaskDelay(10);
    }
}
ISR(INT1_vect)
{
    uint8_t higher;

    rpm_ms = xTaskGetTickCount();
    //rpm_us = TCNT1;

        isr_rpm.last_ms=isr_rpm.curr_ms;
        //isr_rpm.last_us=isr_rpm.curr_us;
        isr_rpm.curr_ms=rpm_ms;
        //isr_rpm.curr_us=rpm_us;


        xQueueSendFromISR(xQueueRpmEvents,&isr_rpm,&higher);

}
```

```
//dash_lcd.c

#include "dash.h"

void vTaskLcd(void *pvParameters)
{
    lcd_init(LCD_DISP_ON);

    LCD_EVENT lcd;

    while(1)
    {
        if(xQueueReceive(xQueueLcdEvents,&lcd,portMAX_DELAY))
        {
            if(lcd.event==LCD_CLEAR_SCREEN)
            {
                lcd_clrscr();
            }
            else if(lcd.event==LCD_WRITE)
            {

                lcd.str[20]='\0';
                lcd_gotoxy(lcd.x,lcd.y);
                lcd_puts(lcd.str);
            }
        }

    }

}

#define LED1_PORT   PORTD
#define LED1_DDR    DDRD
#define LED1        PD0

#define LED2_PORT   PORTD
#define LED2_DDR    DDRD
#define LED2        PD1

#define LED3_PORT   PORTD
#define LED3_DDR    DDRD
#define LED3        PD4

#define LED4_PORT   PORTC
#define LED4_DDR    DDRC
#define LED4        PC6

#define LED5_PORT   PORTC
#define LED5_DDR    DDRC
#define LED5        PC7

#define LED6_PORT   PORTA
#define LED6_DDR    DDRA
```

```
#define LED6        PA7

void vTaskLeds(void *pvParameters)
{
    SetBit(LED1_DDR,LED1);
    SetBit(LED2_DDR,LED2);
    SetBit(LED3_DDR,LED3);
    SetBit(LED4_DDR,LED4);
    SetBit(LED5_DDR,LED5);
    SetBit(LED6_DDR,LED6);

    ClrBit(LED1_PORT,LED1);
    ClrBit(LED2_PORT,LED2);
    ClrBit(LED3_PORT,LED3);
    ClrBit(LED4_PORT,LED4);
    ClrBit(LED5_PORT,LED5);
    ClrBit(LED6_PORT,LED6);

    uint16_t led1_clk=0;
    uint16_t led2_clk=0;
    uint16_t led3_clk=0;
    uint16_t led4_clk=0;
    uint16_t led5_clk=0;
    uint16_t led6_clk=0;

    uint16_t led1_per=0;
    uint16_t led2_per=0;
    uint16_t led3_per=0;
    uint16_t led4_per=0;
    uint16_t led5_per=0;
    uint16_t led6_per=0;

    uint16_t tick=0;

    LED_EVENT led;

    while(1)
    {

        if(xQueueReceive(xQueueLedEvents,&led,0))
        {
            if(led.led==1) led1_per=led.period;
            else if(led.led==2) led2_per=led.period;
            else if(led.led==3) led3_per=led.period;
            else if(led.led==4) led4_per=led.period;
            else if(led.led==5) led5_per=led.period;
            else if(led.led==6) led6_per=led.period;
        }

        if(BitIsClr(MCP_RX0BF_PORT,MCP_RX0BF_PIN))
        {
            led6_per=0;
        }
        else
        {
            led6_per=0xFF;
        }

        led1_clk+=tick;
        led2_clk+=tick;
        led3_clk+=tick;
        led4_clk+=tick;
        led5_clk+=tick;
        led6_clk+=tick;



        if(led6_per==0)                     ClrBit(LED6_PORT,LED6);
        else if(led6_per==0xFF)             SetBit(LED6_PORT,LED6);
        else if(led6_clk<(led6_per/2))      SetBit(LED6_PORT,LED6);
        else if(led6_clk<led6_per)          ClrBit(LED6_PORT,LED6);
        else                                led6_clk=0;

        if(led1_per==0)                     ClrBit(LED1_PORT,LED1);
        else if(led1_per==0xFF)             SetBit(LED1_PORT,LED1);
        else if(led1_clk<(led1_per/2))      SetBit(LED1_PORT,LED1);
        else if(led1_clk<led1_per)          ClrBit(LED1_PORT,LED1);
        else                                led1_clk=0;

        if(led2_per==0)                     ClrBit(LED2_PORT,LED2);
        else if(led2_per==0xFF)             SetBit(LED2_PORT,LED2);
        else if(led2_clk<(led2_per/2))      SetBit(LED2_PORT,LED2);
        else if(led2_clk<led2_per)          ClrBit(LED2_PORT,LED2);
        else                                led2_clk=0;

        if(led3_per==0)                     ClrBit(LED3_PORT,LED3);
        else if(led3_per==0xFF)             SetBit(LED3_PORT,LED3);
        else if(led3_clk<(led3_per/2))      SetBit(LED3_PORT,LED3);
        else if(led3_clk<led3_per)          ClrBit(LED3_PORT,LED3);
        else                                led3_clk=0;

        if(led4_per==0)                     ClrBit(LED4_PORT,LED4);
        else if(led4_per==0xFF)             SetBit(LED4_PORT,LED4);
        else if(led4_clk<(led4_per/2))      SetBit(LED4_PORT,LED4);
        else if(led4_clk<led4_per)          ClrBit(LED4_PORT,LED4);
        else                                led4_clk=0;

        if(led5_per==0)                     ClrBit(LED5_PORT,LED5);
        else if(led5_per==0xFF)             SetBit(LED5_PORT,LED5);
        else if(led5_clk<(led5_per/2))      SetBit(LED5_PORT,LED5);
        else if(led5_clk<led5_per)          ClrBit(LED5_PORT,LED5);
        else                                led5_clk=0;



        tick=xTaskGetTickCount();
        vTaskDelay(50);
        tick=xTaskGetTickCount()-tick;




    }
}
```

```c
//dash_ui.c
#include "dash.h"

#define UI_MAIN 1
#define UI_GAUGES 2
#define UI_DRVR 3
#define UI_ECU 4
#define UI_ECU_DATA 5
#define UI_TRSP 6
#define UI_TRSP_SUSP 7
#define UI_TRSP_TIRES 8
#define UI_TCM      9
#define UI_TCM_DATA 10
#define UI_TCM_RPMS 11
#define UI_SYSINFO  12
#define UI_DAQ      13
#define UI_GPS      14
#define UI_GPS_DATA 15
#define UI_GPS_TIME 16

volatile CAR_DATA gData;

LCD_EVENT   Ui_Lcd;
uint8_t     Ui_Btn;
uint8_t     Ui_Page;
LED_EVENT   Ui_Led;
CAR_DATA Ui_Data;

#define UiCheckForButton(evt,pg) if((Ui_Btn)==(evt)) (Ui_Page)=(pg)

void vTaskUi(void *pvParameters)
{
    Ui_Page=UI_MAIN;
    Ui_Btn=0;

    while(1)
    {
            if      (Ui_Page==UI_MAIN)              PageMain();
            //else if   (Ui_Page==UI_GAUGES)            PageGauges();
            // else if  (Ui_Page==UI_DRVR)              PageDriver();
            // else if  (Ui_Page==UI_ECU)               PageEcu();
            // else if  (Ui_Page==UI_ECU_DATA)      PageEcuData();
            // else if  (Ui_Page==UI_TRSP)          PageTrsp();
            // else if  (Ui_Page==UI_TRSP_SUSP)         PageTrspSusp();
            // else if  (Ui_Page==UI_TRSP_TIRES)        PageTrspTires();
            // else if  (Ui_Page==UI_TCM)               PageTcm();
            // else if  (Ui_Page==UI_TCM_DATA)      PageTcmData();
            // else if  (Ui_Page==UI_TCM_RPMS)      PageTcmRpms();
            // else if  (Ui_Page==UI_SYSINFO)           PageSysInfo();
    }
}

void PageMain(void)
```

```c
{
    UpdateUiData();

    char air_str[5];
    char wtr_str[5];
    char can_str[4];
    char lct_str[4];

    Ui_Lcd.event=LCD_WRITE;

    while(Ui_Page==UI_MAIN)
    {

        //Ui_Lcd.event=LCD_CLEAR_SCREEN;
        //xQueueSendToBack(xQueueLcdEvents,&Ui_Lcd,portMAX_DELAY);
        Ui_Lcd.event=LCD_WRITE;

        Ui_Lcd.y=0;
        Ui_Lcd.x=0;

        snprintf_P(Ui_Lcd.str,21,PSTR("Rpm:%4d     Mph:%3d"),Ui_Data.Rpm,Ui_Data.Mph);
        xQueueSendToBack(xQueueLcdEvents,&Ui_Lcd,portMAX_DELAY);


        Ui_Lcd.y=1;
        Ui_Lcd.x=0;
        snprintf_P(Ui_Lcd.str,21,PSTR("Atmp|Wtmp|Volts|Gear"));
        xQueueSendToBack(xQueueLcdEvents,&Ui_Lcd,portMAX_DELAY);

        Ui_Lcd.y=2;
        Ui_Lcd.x=0;


        if(Ui_Data.air_tmp>100)
        {
            snprintf_P(air_str,5,PSTR("OVER"));
            Ui_Led.led=1;
            Ui_Led.period=500;
            xQueueSendToBack(xQueueLedEvents,&Ui_Led,portMAX_DELAY);
        }

        else
        {
            snprintf_P(air_str,5,PSTR("____"));
            Ui_Led.led=1;
            Ui_Led.period=0;
            xQueueSendToBack(xQueueLedEvents,&Ui_Led,portMAX_DELAY);
        }
        if(Ui_Data.water_tmp>200)
        {
            snprintf_P(wtr_str,5,PSTR("OVER"));
            Ui_Led.led=6;
            Ui_Led.period=1000;
            xQueueSendToBack(xQueueLedEvents,&Ui_Led,portMAX_DELAY);

        }
        else
        {
            Ui_Led.led=6;
            Ui_Led.period=0;
            xQueueSendToBack(xQueueLedEvents,&Ui_Led,portMAX_DELAY);
            snprintf_P(wtr_str,5,PSTR("____"));
        }


        if(Ui_Data.lct==2) snprintf_P(lct_str,5,PSTR("LCT"));
        else snprintf_P(lct_str,5,PSTR("___"));

        if(Ui_Data.can_status==1) snprintf_P(can_str,4,PSTR("CAN"));
        else snprintf_P(can_str,4,PSTR("___"));

        snprintf_P(Ui_Lcd.str,21,PSTR("%s %s  %s  %s"),air_str,wtr_str,can_str,lct_str);
        xQueueSendToBack(xQueueLcdEvents,&Ui_Lcd,portMAX_DELAY);

        Ui_Lcd.y=3;
        Ui_Lcd.x=0;

        uint8_t bi=Ui_Data.batt_volt_raw/10;
        uint8_t bf=Ui_Data.batt_volt_raw%10;
        snprintf_P(Ui_Lcd.str,21,PSTR("%3d%c|%3d%c|%2d.%1dV|N123"),Ui_Data.air_tmp,0xDF,Ui_Data.
water_tmp,0xDF,(bi),(bf));
        xQueueSendToBack(xQueueLcdEvents,&Ui_Lcd,portMAX_DELAY);

        while(1)
        {
            vTaskDelay(1000);
            if(xSemaphoreTake(xSemUpdateData,portMAX_DELAY))
            {
                UpdateUiData();
                break;

            }

            Ui_Btn=0;
            if(xQueueReceive(xQueueButtonEvents,&Ui_Btn,0))
            {
                break;
            }

            // UiCheckForButton(BTN_UP_PUSHED,UI_GAUGES);
            // UiCheckForButton(BTN_DOWN_PUSHED,UI_DRVR);
            // UiCheckForButton(BTN_LEFT_PUSHED,UI_DAQ);
            // UiCheckForButton(BTN_RIGHT_PUSHED,UI_ECU);
            // UiCheckForButton(BTN_CENTER_PUSHED,UI_);
            // UiCheckForButton(BTN_1_PUSHED,UI_);
            // UiCheckForButton(BTN_2_PUSHED,UI_);
            // UiCheckForButton(BTN_3_PUSHED,UI_);
            // UiCheckForButton(BTN_4_PUSHED,UI_);
            // UiCheckForButton(BTN_UP_HELD,UI_);
            // UiCheckForButton(BTN_DOWN_HELD,UI_);
            // UiCheckForButton(BTN_LEFT_HELD,UI_);
            // UiCheckForButton(BTN_RIGHT_HELD,UI_);
            // UiCheckForButton(BTN_CENTER_HELD,UI_SYS_INFO);
            // UiCheckForButton(BTN_1_HELD,UI_TGL_LCT);
            // UiCheckForButton(BTN_2_HELD,UI_TGL_DAQ);
            // UiCheckForButton(BTN_3_HELD,UI_);
            // UiCheckForButton(BTN_4_HELD,UI_);




        }
    }
}

void UpdateUiData(void)
{
    uint8_t *p=(uint8_t *)(&Ui_Data);
    uint8_t *g=(uint8_t *)(&gData);

    xSemaphoreTake(xMtxGlobalData,portMAX_DELAY);
    for(uint8_t i=0;i<sizeof(Ui_Data);i++)
    {
        *(p+i)=*(g+i);
    }
    xSemaphoreGive(xMtxGlobalData);

}
```

```
/****************************************************************************
 Title  :   HD44780U LCD library
 Author:    Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
 File:      $Id: lcd.c,v 1.13.2.2 2004/02/12 21:08:25 peter Exp $
 Software:  AVR-GCC 3.3
 Target:    any AVR device, memory mapped mode only for AT90S4414/8515/Mega

 DESCRIPTION
       Basic routines for interfacing a HD44780U-based text lcd display

       Originally based on Volker Oth's lcd library,
       changed lcd_init(), added additional constants for lcd_command(),
       added 4-bit I/O mode, improved and optimized code.

       Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
       4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.

       Memory mapped mode compatible with Kanda STK200, but supports also
       generation of R/W signal through A8 address line.

 USAGE
       See the C include lcd.h file for a description of each function

*****************************************************************************/

// extended by Martin Thomas 3/2004, removed bugs(?), added functions
// and maybe added new bugs
// 6/2005 - update to be compatible with avr-libc 1.2.3, mth

// extended by Andreas Heinzen 8/2008 E-Mail: heinzen@fh-koblenz.de,
// removed bugs(?)
// * correct cursor-positon after scroll
// * redefine  lcd_waitbusy, lcd_getxy (maybe critical, API has changed)
// * correct inactive state after lcd_read
// * made the timing configurable in lcd.h
// added scroll functions for 2-Line Displays
// added 8-Bit IO-MODE
// and maybe added new bugs

#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"

/*
** constants/macros
*/
#define PIN(x) (*(&x - 2))  /* address of data direction register of port x */
#define DDR(x) (*(&x - 1))  /* address of input register of port x          */

#define LCD_FUNCTION_DEFAULT    LCD_FUNCTION_4BIT_2LINES

#define lcd_e_delay()   __asm__ __volatile__( "rjmp 1f\n 1:" );
```

```c
#define lcd_e_high()    LCD_E_PORT  |=  _BV(LCD_E_PIN)
#define lcd_e_low()     LCD_E_PORT  &= ~_BV(LCD_E_PIN)
#define lcd_e_toggle()  toggle_e()
#define lcd_rw_high()   LCD_RW_PORT |=  _BV(LCD_RW_PIN)
#define lcd_rw_low()    LCD_RW_PORT &= ~_BV(LCD_RW_PIN)
#define lcd_rs_high()   LCD_RS_PORT |=  _BV(LCD_RS_PIN)
#define lcd_rs_low()    LCD_RS_PORT &= ~_BV(LCD_RS_PIN)


/*
** function prototypes
*/

static void toggle_e(void);


/*
** local functions
*/




/*************************************************************************
 delay loop for small accurate delays: 16-bit counter, 4 cycles/loop
*************************************************************************/
static inline void _delayFourCycles(unsigned int __count)
{
    if ( __count == 0 )
        __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
    else
        __asm__ __volatile__ (
            "1: sbiw %0,1" "\n\t"
            "brne 1b"                              // 4 cycles/loop
            : "=w" (__count)
            : "0" (__count)
           );
}


/*************************************************************************
delay for a minimum of <us> microseconds
the number of loops is calculated at compile-time from MCU clock frequency
*************************************************************************/
#define delay(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )

/* toggle Enable Pin to initiate write */
static void toggle_e(void)
{
    lcd_e_high();
    lcd_e_delay();
    lcd_e_low();
}
```

```
/*************************************************************************
Low-level function to write byte to LCD controller
Input:    data   byte to write to LCD
          rs     1: write data
                 0: write instruction
Returns:  none
*************************************************************************/

static void lcd_write(uint8_t data,uint8_t rs)
{


    if (rs)
    {   /* write data        (RS=1, RW=0) */
        lcd_rs_high();
    }
    else
    {    /* write instruction (RS=0, RW=0) */
       lcd_rs_low();
    }

    lcd_rw_low();


        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);

        /* output high nibble first */
    if(data & 0x80) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
    else LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
    if(data & 0x40) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
    else LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
    if(data & 0x20) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
    else LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
    if(data & 0x10) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
    else LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
    lcd_e_toggle();

        /* output low nibble */
        if(data & 0x08) LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);
    else LCD_DATA3_PORT &= ~_BV(LCD_DATA3_PIN);
        if(data & 0x04) LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
    else LCD_DATA2_PORT &= ~_BV(LCD_DATA2_PIN);
        if(data & 0x02) LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        else LCD_DATA1_PORT &= ~_BV(LCD_DATA1_PIN);
        if(data & 0x01) LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        else LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);
        lcd_e_toggle();
        /* all data pins high (inactive) */
        LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);


}


/*************************************************************************
Low-level function to read byte from LCD controller
Input:    rs     1: read data
                 0: read busy flag / address counter
Returns:  byte read from LCD controller
*************************************************************************/

static uint8_t lcd_read(uint8_t rs)
{
    uint8_t data;


    if (rs)
        lcd_rs_high();                       /* RS=1: read data      */
    else
        lcd_rs_low();                        /* RS=0: read busy flag */
    lcd_rw_high();                           /* RW=1  read mode      */





        /* configure data pins as input */
        DDR(LCD_DATA0_PORT) &= ~_BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) &= ~_BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) &= ~_BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) &= ~_BV(LCD_DATA3_PIN);

        /* read high nibble first */
        lcd_e_high();
        lcd_e_delay();
        data = 0;
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x10;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x20;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x40;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x80;
        lcd_e_low();

        lcd_e_delay();                       /* Enable 500ns low       */

        /* read low nibble */
        lcd_e_high();
        lcd_e_delay();
        if ( PIN(LCD_DATA0_PORT) & _BV(LCD_DATA0_PIN) ) data |= 0x01;
        if ( PIN(LCD_DATA1_PORT) & _BV(LCD_DATA1_PIN) ) data |= 0x02;
        if ( PIN(LCD_DATA2_PORT) & _BV(LCD_DATA2_PIN) ) data |= 0x04;
        if ( PIN(LCD_DATA3_PORT) & _BV(LCD_DATA3_PIN) ) data |= 0x08;
    lcd_e_low();
        /* all data pins high (inactive) */
        /* configure data pins as output */
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
        LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);
        LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);
        LCD_DATA2_PORT |= _BV(LCD_DATA2_PIN);
        LCD_DATA3_PORT |= _BV(LCD_DATA3_PIN);


    lcd_rw_low();
    return data;
}



/*************************************************************************
loops while lcd is busy, returns address counter
*************************************************************************/
void lcd_waitbusy(void)

{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

}/* lcd_waitbusy */


/*************************************************************************
Move cursor to the start of next line or to the first line if the cursor
is already on the last line.
*************************************************************************/
static inline void lcd_newline(uint8_t pos)
{
    register uint8_t addressCounter=0;



    if ( pos < LCD_START_LINE3 )    addressCounter = LCD_START_LINE2;
    else if ( (pos >= LCD_START_LINE2) && (pos < LCD_START_LINE4) )
    {
        addressCounter = LCD_START_LINE3;
    }
```

```c
    else if ( (pos >= LCD_START_LINE3) && (pos < LCD_START_LINE2) )
    {
        addressCounter = LCD_START_LINE4;
    }


    lcd_command((1<<LCD_DDRAM)+addressCounter);

}/* lcd_newline */


/*
** PUBLIC FUNCTIONS
*/

/*************************************************************************
Send LCD controller instruction command
Input:   instruction to send to LCD controller, see HD44780 data sheet
Returns: none
*************************************************************************/
void lcd_command(uint8_t cmd)
{
    lcd_waitbusy();
    lcd_write(cmd,0);
}


/*************************************************************************
Set cursor to specified position
Input:    x  horizontal position  (0: left most position)
          y  vertical position    (0: first line)
Returns:  none
*************************************************************************/
void lcd_gotoxy(uint8_t x, uint8_t y)
{


    if ( y==0 )
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE1+x);
    else if ( y==1)
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE2+x);
    else if ( y==2)
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE3+x);
    else /* y==3 */
        lcd_command((1<<LCD_DDRAM)+LCD_START_LINE4+x);

}/* lcd_gotoxy */


/*************************************************************************
*************************************************************************/
uint8_t lcd_getxy(void)
{
    register uint8_t c;

    /* wait until busy flag is cleared */
    while ( (c=lcd_read(0)) & (1<<LCD_BUSY)) {}

    /* the address counter is updated 4us after the busy flag is cleared */
    delay(DELAY_READ_DATA); // mt was 4

    /* now read the address counter */
    return (lcd_read(0));  // return address counter
}


/*************************************************************************
Clear display and set cursor to home position
*************************************************************************/
void lcd_clrscr(void)
{
    lcd_command(1<<LCD_CLR);
}


/*************************************************************************
Set cursor to home position
*************************************************************************/
void lcd_home(void)
{
    lcd_command(1<<LCD_HOME);
}



/*************************************************************************
Display character at current cursor position
Input:    character to be displayed
Returns:  none
*************************************************************************/
void lcd_putc(char c)
{
    uint8_t pos;
    pos = lcd_getxy();   // read busy-flag and address counter

    if (c=='\n')
    {
        lcd_newline(pos);
    }
    else
    {
        // mtmt changed order to fix autowrap first write to lcd then
        // check position
        lcd_write(c, 1);
        pos = lcd_getxy();   // read busy-flag and address counter
```

```c
        if ( pos == LCD_START_LINE1+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE2,0);
    else if ( pos == LCD_START_LINE2+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE3,0);
        else if ( pos == LCD_START_LINE3+LCD_DISP_LENGTH )
            lcd_write((1<<LCD_DDRAM)+LCD_START_LINE4,0);
        else if ( pos == LCD_START_LINE4+LCD_DISP_LENGTH ) {
        }
        lcd_waitbusy();

        // lcd_write(c, 1);
    }

}/* lcd_putc */


/*************************************************************************
Display string without auto linefeed
Input:    string to be displayed
Returns:  none
*************************************************************************/
void lcd_puts(const char *s)
/* print string on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = *s++) ) {
        lcd_putc(c);
    }

}/* lcd_puts */


/*************************************************************************
Display string from program memory without auto linefeed
Input:     string from program memory be be displayed
Returns:   none
*************************************************************************/
void lcd_puts_p(const char *progmem_s)
/* print string from program memory on lcd (no auto linefeed) */
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) ) {
        lcd_putc(c);
    }

}/* lcd_puts_p */


/*************************************************************************
Initialize display and select type of cursor
```

```
Input:    dispAttr LCD_DISP_OFF            display off
                   LCD_DISP_ON             display on, cursor off
                   LCD_DISP_ON_CURSOR      display on, cursor on
                   LCD_DISP_CURSOR_BLINK   display on, cursor on flashing
Returns:  none
*************************************************************************/
void lcd_init(uint8_t dispAttr)
{

    /*
     *  Initialize LCD to 4 bit I/O mode
     */
    if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT == &LCD_DATA2_PORT ) && ( &
LCD_DATA2_PORT == &LCD_DATA3_PORT )
      && ( &LCD_RS_PORT == &LCD_DATA0_PORT) && ( &LCD_RW_PORT == &LCD_DATA0_PORT) && (&
LCD_E_PORT == &LCD_DATA0_PORT)
      && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN == 2) && (LCD_DATA3_PIN
 == 3)
      && (LCD_RS_PIN == 4 ) && (LCD_RW_PIN == 5) && (LCD_E_PIN == 6 ) )
    {
        /* configure all port bits as output (all LCD lines on same port) */
        DDR(LCD_DATA0_PORT) |= 0x7F;
    }
    else if ( ( &LCD_DATA0_PORT == &LCD_DATA1_PORT) && ( &LCD_DATA1_PORT == &LCD_DATA2_PORT ) &&
 ( &LCD_DATA2_PORT == &LCD_DATA3_PORT )
           && (LCD_DATA0_PIN == 0 ) && (LCD_DATA1_PIN == 1) && (LCD_DATA2_PIN == 2) && (
LCD_DATA3_PIN == 3) )
    {
        /* configure all port bits as output (all LCD data lines on same port, but control
lines on different ports) */
        DDR(LCD_DATA0_PORT) |= 0x0F;
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
    }
    else
    {
        /* configure all port bits as output (LCD data and control lines on different ports */
        DDR(LCD_RS_PORT)    |= _BV(LCD_RS_PIN);
        //DDR(LCD_RW_PORT)    |= _BV(LCD_RW_PIN);<-RW not used
        DDR(LCD_E_PORT)     |= _BV(LCD_E_PIN);
        DDR(LCD_DATA0_PORT) |= _BV(LCD_DATA0_PIN);
        DDR(LCD_DATA1_PORT) |= _BV(LCD_DATA1_PIN);
        DDR(LCD_DATA2_PORT) |= _BV(LCD_DATA2_PIN);
        DDR(LCD_DATA3_PORT) |= _BV(LCD_DATA3_PIN);
    }

    delay(DELAY_RESET);        /* wait 16ms or more after power-on       */

    /* initial write to lcd is 8bit */
    LCD_DATA1_PORT |= _BV(LCD_DATA1_PIN);  // _BV(LCD_FUNCTION)>>4;
    LCD_DATA0_PORT |= _BV(LCD_DATA0_PIN);  // _BV(LCD_FUNCTION_8BIT)>>4;
    lcd_e_toggle();
    delay(DELAY_INIT_1);         /* delay, busy flag can't be checked here */

    /* repeat last command */
    lcd_e_toggle();
    delay(DELAY_INIT_2);           /* delay, busy flag can't be checked here */

    /* repeat last command a third time */
    lcd_e_toggle();
    delay(DELAY_INIT_2);           /* delay, busy flag can't be checked here */

    /* now configure for 4bit mode */
    //LCD_DATA0_PORT &= ~_BV(LCD_DATA0_PIN);   // LCD_FUNCTION_4BIT_1LINE>>4
    //lcd_e_toggle();

    delay(DELAY_INIT_2);           /* some displays need this additional delay */

    /* from now the LCD only accepts 4 bit I/O, we can use lcd_command() */
    delay(DELAY_INIT_2);           /* some displays need this additional delay */

    /* from now the LCD only accepts 8 bit I/O, we can use lcd_command() */


    lcd_command(LCD_FUNCTION_DEFAULT);      /* function set: display lines  */
    lcd_command(LCD_DISP_OFF);              /* display off                  */
    lcd_clrscr();                           /* display clear                */
    lcd_command(LCD_MODE_DEFAULT);          /* set entry mode               */
    lcd_command(dispAttr);                  /* display/cursor control       */
    DDRA|=(1<<7);
}/* lcd_init */
```

```
#ifndef LCD_H
#define LCD_H
/*************************************************************************
 Title  :   C include file for the HD44780U LCD library (lcd.c)
 Author:    Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
 File:      $Id: lcd.h,v 1.12.2.2 2004/02/12 21:05:59 peter Exp $
 Software:  AVR-GCC 3.3
 Hardware:  any AVR device, memory mapped mode only for AT90S4414/8515/Mega
***************************************************************************/
// extended by Martin Thomas 3/2004, removed bugs(?), added functions
// and maybe added new bugs

// extended by Andreas Heinzen 8/2008 E-Mail: heinzen@fh-koblenz.de,
// removed bugs(?)
// * correct cursor-positon after scroll
// * redefine  lcd_waitbusy, lcd_getxy (maybe critical, API has changed)
// * correct inactive state after lcd_read (isn't realy neccessary)
// * made the timing configurable in lcd.h (look at #define DELAY_.... values)
// added scroll functions for 2-Line Displays
// added 8-Bit IO-MODE
// and maybe added new bugs

// un-extended - by Mark Schaumburg 4/2010

/**
 @defgroup pfleury_lcd LCD library
 @code #include <lcd.h> @endcode

 @brief Basic routines for interfacing a HD44780U-based text LCD display

 Originally based on Volker Oth's LCD library,
 changed lcd_init(), added additional constants for lcd_command(),
 added 4-bit I/O mode, improved and optimized code.

 Library can be operated in memory mapped mode (LCD_IO_MODE=0) or in
 4-bit IO port mode (LCD_IO_MODE=1). 8-bit IO port mode not supported.

 Memory mapped mode compatible with Kanda STK200, but supports also
 generation of R/W signal through A8 address line.

 @author Peter Fleury pfleury@gmx.ch http://jump.to/fleury

 @see The chapter <a href="http://homepage.sunrise.ch/mysunrise/pfleury/avr-lcd44780.html"
target="_blank">Interfacing a HD44780 Based LCD to an AVR</a>
      on my home page.

*/

/*@{*/

#if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
#error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC compiler !"
#endif
```

```
#include <inttypes.h>
#include <avr/pgmspace.h>

/**
 *  @name  Definitions for MCU Clock Frequency
 *  Adapt the MCU clock frequency in Hz to your target.
 */
//#define XTAL 4000000         /**< clock frequency in Hz, used to calculate delay timer */
// mtmt:
#define XTAL F_CPU              /**< clock frequency in Hz, used to calculate delay timer */


#define LCD_PORT         PORTA        /**< port for the LCD lines   */
#define LCD_DATA0_PORT   LCD_PORT     /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT   LCD_PORT     /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT   LCD_PORT     /**< port for 4bit data bit 2 */
#define LCD_DATA3_PORT   LCD_PORT     /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN    0            /**< pin for 4bit data bit 0  */
#define LCD_DATA1_PIN    1            /**< pin for 4bit data bit 1  */
#define LCD_DATA2_PIN    2            /**< pin for 4bit data bit 2  */
#define LCD_DATA3_PIN    3            /**< pin for 4bit data bit 3  */
#define LCD_RS_PORT      PORTA     /**< port for RS line         */
#define LCD_RS_PIN       6            /**< pin  for RS line         */
#define LCD_RW_PORT      PORTA     /**< port for RW line         */
#define LCD_RW_PIN       4            /**< pin  for RW line         */
#define LCD_E_PORT       PORTA     /**< port for Enable line     */
#define LCD_E_PIN        5            /**< pin  for Enable line     */


#define DELAY_RESET     15000
#define DELAY_INIT_1        5000
#define DELAY_INIT_2        100
#define DELAY_READ_DATA     46

#define LCD_LINES           4     /**< number of visible lines of the display */
#define LCD_DISP_LENGTH    20     /**< visibles characters per line of the display */
#define LCD_LINE_LENGTH  0x10     /**< internal line length of the display    */
#define LCD_4BIT_MODE       1     /**using display in 4-Bit-Mode */
#define LCD_START_LINE1  0x00     /**< DDRAM address of first char of line 1 */
#define LCD_START_LINE2  0x40     /**< DDRAM address of first char of line 2 */
#define LCD_START_LINE3  0x14     /**< DDRAM address of first char of line 3 */
#define LCD_START_LINE4  0x54     /**< DDRAM address of first char of line 4 */
#define LCD_WRAP_LINES      0     /**< 0: no wrap, 1: wrap at end of visibile line */


/* instruction register bit positions, see HD44780U data sheet */
#define LCD_CLR               0      /* DB0: clear display                  */
#define LCD_HOME              1      /* DB1: return to home position        */
#define LCD_ENTRY_MODE        2      /* DB2: set entry mode                 */
#define LCD_ENTRY_INC         1      /*   DB1: 1=increment, 0=decrement     */
#define LCD_ENTRY_SHIFT       0      /*   DB2: 1=display shift on           */
#define LCD_ON                3      /* DB3: turn lcd/cursor on             */
```

```
#define LCD_ON_DISPLAY        2      /*   DB2: turn display on              */
#define LCD_ON_CURSOR         1      /*   DB1: turn cursor on               */
#define LCD_ON_BLINK          0      /*     DB0: blinking cursor ?          */
#define LCD_MOVE              4      /* DB4: move cursor/display            */
#define LCD_MOVE_DISP         3      /*   DB3: move display (0-> cursor) ?  */
#define LCD_MOVE_RIGHT        2      /*   DB2: move right (0-> left) ?      */
#define LCD_FUNCTION          5      /* DB5: function set                   */
#define LCD_FUNCTION_8BIT     4      /*   DB4: set 8BIT mode (0->4BIT mode) */
#define LCD_FUNCTION_2LINES   3      /*   DB3: two lines (0->one line)      */
#define LCD_FUNCTION_10DOTS   2      /*   DB2: 5x10 font (0->5x7 font)      */
#define LCD_CGRAM             6      /* DB6: set CG RAM address             */
#define LCD_DDRAM             7      /* DB7: set DD RAM address             */
#define LCD_BUSY              7      /* DB7: LCD is busy                    */

/* set entry mode: display shift on/off, dec/inc cursor move direction */
#define LCD_ENTRY_DEC            0x04   /* display shift off, dec cursor move dir */
#define LCD_ENTRY_DEC_SHIFT      0x05   /* display shift on,  dec cursor move dir */
#define LCD_ENTRY_INC_           0x06   /* display shift off, inc cursor move dir */
#define LCD_ENTRY_INC_SHIFT      0x07   /* display shift on,  inc cursor move dir */

/* display on/off, cursor on/off, blinking char at cursor position */
#define LCD_DISP_OFF             0x08   /* display off                            */
#define LCD_DISP_ON              0x0C   /* display on, cursor off                 */
#define LCD_DISP_ON_BLINK        0x0D   /* display on, cursor off, blink char     */
#define LCD_DISP_ON_CURSOR       0x0E   /* display on, cursor on                  */
#define LCD_DISP_ON_CURSOR_BLINK 0x0F   /* display on, cursor on, blink char      */

/* move cursor/shift display */
#define LCD_MOVE_CURSOR_LEFT     0x10   /* move cursor left  (decrement)          */
#define LCD_MOVE_CURSOR_RIGHT    0x14   /* move cursor right (increment)          */
#define LCD_MOVE_DISP_LEFT       0x18   /* shift display left                     */
#define LCD_MOVE_DISP_RIGHT      0x1C   /* shift display right                    */

/* function set: set interface data length and number of display lines */
#define LCD_FUNCTION_4BIT_1LINE  0x20   /* 4-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_4BIT_2LINES 0x28   /* 4-bit interface, dual line,   5x7 dots */
#define LCD_FUNCTION_8BIT_1LINE  0x30   /* 8-bit interface, single line, 5x7 dots */
#define LCD_FUNCTION_8BIT_2LINES 0x38   /* 8-bit interface, dual line,   5x7 dots */


#define LCD_MODE_DEFAULT     ((1<<LCD_ENTRY_MODE) | (1<<LCD_ENTRY_INC) )




/**
 *  @name Functions
 */



/**
 @brief    Initialize display and select type of cursor
 @param    dispAttr \b LCD_DISP_OFF display off\n
                    \b LCD_DISP_ON display on, cursor off\n
                    \b LCD_DISP_ON_CURSOR display on, cursor on\n
                    \b LCD_DISP_ON_CURSOR_BLINK display on, cursor on flashing
 @return  none
*/
extern void lcd_init(uint8_t dispAttr);

/**
 @brief    Clear display and set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_clrscr(void);


/**
 @brief    Set cursor to home position
 @param    void
 @return   none
*/
extern void lcd_home(void);


/**
 @brief    Set cursor to specified position

 @param    x horizontal position\n (0: left most position)
 @param    y vertical position\n   (0: first line)
 @return   none
*/
extern void lcd_gotoxy(uint8_t x, uint8_t y);

/**
 @brief    Get cursor position

 @param    void
 @return   cursor address counter
*/

extern uint8_t lcd_getxy(void);


/**
 @brief    Display character at current cursor position
 @param    c character to be displayed
 @return   none
*/
extern void lcd_putc(char c);


/**
 @brief    Display string without auto linefeed
 @param    s string to be displayed
 @return   none
```

```
*/
extern void lcd_puts(const char *s);



/**
 @brief    Display string from program memory without auto linefeed
 @param    s string from program memory be be displayed
 @return   none
 @see      lcd_puts_P
*/
extern void lcd_puts_p(const char *progmem_s);



/**
 @brief    Send LCD controller instruction command
 @param    cmd instruction to send to LCD controller, see HD44780 data sheet
 @return   none
*/
extern void lcd_command(uint8_t cmd);

// mtmt new function:
#ifdef LCD_SCROLL_FUNCTION
/**
 @brief    Scroll LCD up
 @param    none
 @return   none
*/
extern void lcd_scrollup(void);
#endif

// mtmt exported for debugging
extern void lcd_waitbusy(void);

/**
 @brief macros for automatically storing string constant in program memory
*/
#define lcd_puts_P(__s)         lcd_puts_p(PSTR(__s))



/*@}*/
#endif //LCD_H
```

```
/*********************************************************
             Controller Area Network
       Hardware Driver for Microchip MCP2515
    v0.9                              Rev: 1/06/05
    (C)2004 by Jeremy Billheimer
*********************************************************/
//updated by Kevin Zimmerschied 2008-05-10
//updated by Michael Moore 2009-5-27
    //added Wrapper functions written by Solomon Williams
    //added more setup information in .h file
    //library no longer relies on spi.c and spi.h

#include "mcp2515.h"


/******************************
 Hardware Drivers for MCP2515:
******************************/
uint8_t mcp_v_spi(uint8_t data)
    {
    SPDR = data;
    loop_until_bit_is_set(SPSR,SPIF);
    return SPDR;
    }//end mcp_v_spi()

void mcp_v_CANSend(uint8_t buffer, uint8_t recipients, uint8_t command, uint8_t argcount,
uint8_t *arg)
    {
    uint8_t SENDData[8];

    if(buffer == 0) while((mcp_u8_readbyte(TXB0CTRL) & 0x08) == 0x08);  //wait until send
buffer is empty
    else if(buffer == 1) while((mcp_u8_readbyte(TXB1CTRL) & 0x08) == 0x08);//wait until send
buffer is empty

    SENDData[0] = command;
    for(uint8_t i=0 ; i < argcount; i++)
        {
            SENDData[i+1]=(*arg);
            arg++;
        }
    mcp_v_loadtxbid(buffer, recipients, 0x00, 0x00, 0x00, argcount+1, SENDData);
    mcp_v_rts(1<<buffer);

    if(buffer == 0) while((mcp_u8_readbyte(TXB0CTRL) & 0x08) == 0x08);  //wait until send
buffer is empty
    else if(buffer == 1) while((mcp_u8_readbyte(TXB1CTRL) & 0x08) == 0x08);//wait until send
buffer is empty

    }//end mcp_v_CANSend()

void mcp_v_CANReceive(uint8_t *recdata)                 //Receives the message from receive
buffer 0 only
```

```
    {
    mcp_v_readrxb0(recdata);                                //Get message from CAN controller
    mcp_v_bitmodify(CANINTF, (1<<RX0IF), 0x00); //Send signal to release buffer for further
messages
    while(mcp_u8_getstatus() & (1<<status_RX0IF));  //Wait until buffer is released
    }//end mcp_v_CANReceive()

void mcp_v_CANSetup(uint8_t address, uint8_t data)
    {
    do
        {
        mcp_v_writebyte(address, data);
        _delay_us(1);
        }while(mcp_u8_readbyte(address) != data);
    }//end mcp_v_CANSetup()

void mcp_v_reset(){

        // DDRD|=(1<<PD0);
        // PORTD|=(1<<PD0);
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(CANRESET);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}


void mcp_v_writebyte(uint8_t address, uint8_t data){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(CANWrite);
    mcp_v_spi(address);
    mcp_v_spi(data);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}


uint8_t mcp_u8_readbyte(uint8_t address){
    uint8_t returnvar;

    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(CANRead);
    mcp_v_spi(address);

    returnvar = mcp_v_spi(0x00);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);

    return(returnvar);
}
```

```
void mcp_v_bitmodify(uint8_t address, uint8_t mask, uint8_t data){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(BitModify);
    mcp_v_spi(address);
    mcp_v_spi(mask);
    mcp_v_spi(data);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}


void mcp_v_loadtxbid(uint8_t buffer, uint8_t SIDH, uint8_t SIDL, uint8_t EID8, uint8_t EID0,
uint8_t nBytes, uint8_t *data){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    if(buffer == 0){
        mcp_v_spi(LoadTXB0ID);
    }
    else if(buffer == 1){
        mcp_v_spi(LoadTXB1ID);
    }
    else if(buffer == 2){
        mcp_v_spi(LoadTXB2ID);
    }

    mcp_v_spi(SIDH);
    mcp_v_spi(SIDL);
    mcp_v_spi(EID8);
    mcp_v_spi(EID0);
    mcp_v_spi(nBytes);

    for( ; nBytes > 0; nBytes--){
        mcp_v_spi(*data);
        data++;
    }

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}


void mcp_v_rts(uint8_t buffers){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(RTSTXB | buffers);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}
```

```c
uint8_t mcp_u8_getstatus(void){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(CANReadStatus);

    uint8_t result = mcp_v_spi(0x00);

    MCP_CS_PORT |= _BV(MCP_CS_PIN);

    return result;
}


void mcp_v_readrxb0(uint8_t *data){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(ReadRXB0ID);
    mcp_v_readrxb(data);
}


void mcp_v_readrxb1(uint8_t *data){
    MCP_CS_PORT &= ~_BV(MCP_CS_PIN);

    mcp_v_spi(ReadRXB1ID);
    mcp_v_readrxb(data);
}


// non public functions below, DO NOT USE!

// reads contents of currently selected RX buffer into memory location
// pointed to by *data.
void mcp_v_readrxb(uint8_t *data)
{
    // read in first 4 bytes of MCP2515 RX buffer, which will be
    // SIDH,SIDL,EID8,EID0,nBytes.
    for(uint8_t i = 0; i <= 3 ; i++)
    {
        *data = mcp_v_spi(0x00);
        data++;
    }
    uint8_t nBytes = mcp_v_spi(0x00) & 0x0F; // get nBytes from buffer next.
    *data = nBytes;
    data++;

    for(uint8_t i = 0; i <= nBytes; i++){
        *data = mcp_v_spi(0x00);
        data++;
    }

    MCP_CS_PORT |= _BV(MCP_CS_PIN);
}
```

```
void mcp_v_can_init(uint8_t address)
{

    mcp_v_reset();
                                  //reset CAN controller
_delay_us(20);                           // There must be a 20us delay after reset or the next few
                              //commands may not be recognized as per 8.1 of datasheet
mcp_v_CANSetup(CNF1     ,0xC1);         //setup bit timing for CAN network

mcp_v_CANSetup(CNF2     ,0xC9);         //setup bit timing for CAN network
mcp_v_CANSetup(CNF3     ,0x02);         //setup bit timing for CAN network
                        //0x60 for all messages
mcp_v_CANSetup(RXB0CTRL ,0x20);         //0x20 for messages with Standard ID only
mcp_v_CANSetup(RXB1CTRL ,0x40);         //messages with Extended ID only (off)
mcp_v_CANSetup(BFPCTRL  ,0x05);     //enable receive buffer full interrupts
mcp_v_CANSetup(CANINTE  ,0x00);     //interrupt setup

mcp_v_CANSetup(RXM0SIDH ,0xFF); //this block of masks and filters apply to
mcp_v_CANSetup(RXM0SIDL ,0x00);             //buffer 0 only WRL_Address_Mask
mcp_v_CANSetup(RXM0EID8 ,0x00);
mcp_v_CANSetup(RXM0EID0 ,0x00);
mcp_v_CANSetup(RXF0SIDH ,address);
mcp_v_CANSetup(RXF0SIDL ,0x00);
mcp_v_CANSetup(RXF0EID8 ,0x00);
mcp_v_CANSetup(RXF0EID0 ,0x00);
mcp_v_CANSetup(RXF1SIDH ,address);
mcp_v_CANSetup(RXF1SIDL ,0x00);
mcp_v_CANSetup(RXF1EID8 ,0x00);
mcp_v_CANSetup(RXF1EID0 ,0x00);

mcp_v_CANSetup(RXM1SIDH ,0xFF);     //this block of masks and filters apply to
mcp_v_CANSetup(RXM1SIDL ,0x00);     //buffer 1 only
mcp_v_CANSetup(RXM1EID8 ,0x00);
mcp_v_CANSetup(RXM1EID0 ,0x00);
mcp_v_CANSetup(RXF2SIDH ,0x00);
mcp_v_CANSetup(RXF2SIDL ,0x00);
mcp_v_CANSetup(RXF2EID8 ,0x00);
mcp_v_CANSetup(RXF2EID0 ,0x00);
mcp_v_CANSetup(RXF3SIDH ,0x00);
mcp_v_CANSetup(RXF3SIDL ,0x00);
mcp_v_CANSetup(RXF3EID8 ,0x00);
mcp_v_CANSetup(RXF3EID0 ,0x00);
mcp_v_CANSetup(RXF4SIDH ,0x00);
mcp_v_CANSetup(RXF4SIDL ,0x00);
mcp_v_CANSetup(RXF4EID8 ,0x00);
mcp_v_CANSetup(RXF4EID0 ,0x00);
mcp_v_CANSetup(RXF5SIDH ,0x00);
mcp_v_CANSetup(RXF5SIDL ,0x00);
mcp_v_CANSetup(RXF5EID8 ,0x00);
mcp_v_CANSetup(RXF5EID0 ,0x00);
```

```
    // mcp_v_bitmodify(CANCTRL  ,0xFF ,0x03);//loop back mode
    // printf_P(PSTR("\rWARNING: CAN Controller in loopback mode!"));
    mcp_v_bitmodify(CANCTRL ,0xE0 ,0x00);           //set mode of operation

    do  {
        _delay_ms(1);
        }while((mcp_u8_readbyte(CANSTAT) & 0xE0) != 0x00);

}
```

```
#ifndef MCP2515_H
#define MCP2515_H

/*********************************************************
               Controller Area Network
    Hardware Driver for Microchip MCP2515
            Header File
    v0.1                  Rev: 1/06/05
    (C)2004 by Jeremy Billheimer
*********************************************************/
//updated by Kevin Zimmerschied 2008-05-10
//updated by Michael Moore 2009-5-27
    //added Wrapper functions written by Solomon Williams
    //added more setup information in .h file
    //library no longer relies on spi.c and spi.h



#include <inttypes.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>



/*
To use this library the SPI bus should be initialized according
to the microcontroller's datasheet. Also, the function
mcp_v_spi() should provide an adequate means for transferring
bytes of data through the SPI bus make sure this function is
compatible with your microcontroller. Be sure that the SPI
control registers are set up properly, an example is shown below.

    SPCR = 0x5C;
    SPSR = 0x01;

Set up IO pins for MCP2515 suggested settings are listed below
DDR 1=out 0=in
MCP_CS - out
MOSI - out
MISO - in
SCK  - out

Be sure that the PORT and PIN settings below reflect your actual
connections between the microcontroller and CAN controller
*/

//-----PIN SETTINGS---------
//outputs
#define MCP_CS_PORT PORTB
#define MCP_CS_PIN PINB4
#define MCP_CS_DDR DDRB
```

```
//inputs
#define MCP_RX0BF_PORT PIND
#define MCP_RX0BF_PIN PIND2
#define MCP_RX0BF_DDR DDRD
#define MCP_RX0BF_p PORTD

#define MCP_RX1BF_PORT PINB
#define MCP_RX1BF_PIN PB1
#define MCP_RX1BF_DDR DDRB

//IO Defs
#define SPI_PORT    PORTB
#define SPI_DDR     DDRB
#define SPI_SCK     PB7
#define SPI_MISO    PB6
#define SPI_MOSI    PB5
#define SPI_SS      MCP_CS_PIN



//-----End PIN SETTINGS-----

//Below is an example initialization routine for the CAN controller
//Copy and paste the code below in to your initialization routine
/*
//------------------Initialize the MCP2515 CAN
Controller-------------------------------------------
//make sure the SPI registers are properly initialized before this section of code
mcp_v_reset();                                  //reset CAN controller
_delay_us(20);                          // There must be a 20us delay after reset or the next few
                                        //commands may not be recognized as per 8.1 of datasheet
mcp_v_CANSetup(CNF1     ,CNF1config);           //setup bit timing for CAN network
mcp_v_CANSetup(CNF2     ,CNF2config);           //setup bit timing for CAN network
mcp_v_CANSetup(CNF3     ,CNF3config);           //setup bit timing for CAN network
mcp_v_CANSetup(RXB0CTRL ,0x20);         //0x20 for messages with Standard ID only
mcp_v_CANSetup(RXB1CTRL ,0x40);         //messages with Extended ID only (off)
mcp_v_CANSetup(BFPCTRL  ,0x05);     //enable receive buffer full interrupts
mcp_v_CANSetup(CANINTE  ,0x01);     //interrupt setup

mcp_v_CANSetup(RXM0SIDH ,_Address_Mask);    //this block of masks and filters apply to
mcp_v_CANSetup(RXM0SIDL ,0x00);             //buffer 0 only
mcp_v_CANSetup(RXM0EID8 ,0x00);
mcp_v_CANSetup(RXM0EID0 ,0x00);
mcp_v_CANSetup(RXF0SIDH ,_Address_Mask);
mcp_v_CANSetup(RXF0SIDL ,0x00);
mcp_v_CANSetup(RXF0EID8 ,0x00);
mcp_v_CANSetup(RXF0EID0 ,0x00);
mcp_v_CANSetup(RXF1SIDH ,_Address_Mask);
mcp_v_CANSetup(RXF1SIDL ,0x00);
mcp_v_CANSetup(RXF1EID8 ,0x00);
mcp_v_CANSetup(RXF1EID0 ,0x00);

mcp_v_CANSetup(RXM1SIDH ,0xFF);     //this block of masks and filters apply to
mcp_v_CANSetup(RXM1SIDL ,0x00);     //buffer 1 only
mcp_v_CANSetup(RXM1EID8 ,0x00);
mcp_v_CANSetup(RXM1EID0 ,0x00);
mcp_v_CANSetup(RXF2SIDH ,0x00);
mcp_v_CANSetup(RXF2SIDL ,0x00);
mcp_v_CANSetup(RXF2EID8 ,0x00);
mcp_v_CANSetup(RXF2EID0 ,0x00);
mcp_v_CANSetup(RXF3SIDH ,0x00);
mcp_v_CANSetup(RXF3SIDL ,0x00);
mcp_v_CANSetup(RXF3EID8 ,0x00);
mcp_v_CANSetup(RXF3EID0 ,0x00);
mcp_v_CANSetup(RXF4SIDH ,0x00);
mcp_v_CANSetup(RXF4SIDL ,0x00);
mcp_v_CANSetup(RXF4EID8 ,0x00);
mcp_v_CANSetup(RXF4EID0 ,0x00);
mcp_v_CANSetup(RXF5SIDH ,0x00);
mcp_v_CANSetup(RXF5SIDL ,0x00);
mcp_v_CANSetup(RXF5EID8 ,0x00);
mcp_v_CANSetup(RXF5EID0 ,0x00);

mcp_v_bitmodify(CANCTRL ,0xE0 ,0x00);           //set mode of operation

do  {
    _delay_ms(1);
    }while((mcp_u8_readbyte(CANSTAT) & 0xE0) != 0x00);
//--------------end initialize CAN controller-------------------------------------------------
*/

#define CANInts 0x05
             // Bit 7 - Message Error
             // Bit 6 - Wakeup
             // Bit 5 - Error
             // Bit 4 - TXB2 Empty
             // Bit 3 - TXB1 Empty
             // Bit 2 - TXB0 Empty
             // Bit 1 - RXB1 Full
             // Bit 0 - RXB0 Full
#define MERRF 7
#define WAKIF 6
#define ERRIF 5
#define TX2IF 4
#define TX1IF 3
#define TX0IF 2
#define RX1IF 1
#define RX0IF 0

#define STATUS 0x00
             //Bit 7 - CANINTF.TX2IF
             //Bit 6 - TXB2CTRL.TXREQ
             //Bit 5 - CANINTF.TX1IF
             //Bit 4 - TXB1CTRL.TXREQ
             //Bit 3 - CANINTF.TX0IF
             //Bit 2 - TXB0CTRL.TXREQ
             //Bit 1 - CANINTF.RX1IF
             //Bit 0 - CANINTF.RX0IF
#define status_TX2IF 7
#define status_TXB2_TXREQ 6
#define status_TX1IF 5
#define status_TXB1_TXREQ 4
#define status_TX0IF 3
#define status_TXB0_TXREQ 2
#define status_RX1IF 1
#define status_RX0IF 0



// A bit timing calculator was created by Michael Moore. If you do not
// have a copy of this please feel free to request it via email at
// michaelmoore44@gmail.com
//below are example settings for bit timing registers
/* Set physical layer configuration
    Fosc = 4.032 MHz
    BRP = 0
    Sync Seg = 1TQ
    Prop Seg = 6TQ
    Phase Seg1 = 6TQ
    Phase Seg2 = 5TQ

    TQ = 2 * (1/Fosc) * (BRP+1)
    Bus speed = 1/(Total # of TQ)*TQ = 100.8 kHz
*/
/*
#define CNF1config 0xC1 //b 0100 0000
#define CNF2config 0xC9 //b 1010 1101
#define CNF3config 0x02 //b 0000 0100
*/

//Settings for a 9.216 MHz crystal 256K baud

#define     CNF1config      0xC1
#define     CNF2config      0xC9
#define     CNF3config      0x02



/*
//Settings for a 18.432 MHz crystal 256K baud

#define     CNF1config      0xC1
#define     CNF2config      0xED
#define     CNF3config      0x04
*/

//16 mhz - 250000baud
//#define       CNF1config      0xC3
//#define       CNF2config      0xC9
//#define       CNF3config      0x02

//MCP2515 instruction set:
```

```
#define CANRESET 0xC0

#define ReadRXB0ID 0x90 // read RX buffer with identifier
#define ReadRXB0 0x92  // read RX buffer, skip the indentifier
#define ReadRXB1ID 0x94
#define ReadRXB1 0x96

#define LoadTXB0ID 0x40 // load TX buffer 0 with identifier
#define LoadTXB0 0x41 // load TX buffer 0, skip identifier

#define LoadTXB1ID 0x42 // load TX buffer 1 with identifier
#define LoadTXB1 0x43 // load TX buffer 1, skip identifier

#define LoadTXB2ID 0x44 // load TX buffer 2 with identifier
#define LoadTXB2 0x45 // load TX buffer 2, skip identifier


#define RTSTXB 0x80 // request to send for no buffers

#define CANWrite 0x02
#define CANRead 0x03
#define CANReadStatus 0xA0
#define CANReadRXStatus 0xB0

#define BitModify 0x05

// end of instruction set

// MCP2515 Control Registers:
#define RXF0SIDH 0x00
#define RXF0SIDL 0x01
#define RXF0EID8 0x02
#define RXF0EID0 0x03
#define RXF1SIDH 0x04
#define RXF1SIDL 0x05
#define RXF1EID8 0x06
#define RXF1EID0 0x07
#define RXF2SIDH 0x08
#define RXF2SIDL 0x09
#define RXF2EID8 0x0A
#define RXF2EID0 0x0B
#define BFPCTRL 0x0C
    #define B1BFS 5
    #define B0BFS 4
    #define B1BFE 3
    #define B0BFE 2
    #define B1BFM 1
    #define B0BFM 0
#define TXRTSCTRL 0x0D
#define CANSTAT 0x0E
#define CANCTRL 0x0F

#define RXF3SIDH 0x10
```

```
#define RXF3SIDL 0x11
#define RXF3EID8 0x12
#define RXF3EID0 0x13
#define RXF4SIDH 0x14
#define RXF4SIDL 0x15
#define RXF4EID8 0x16
#define RXF4EID0 0x17
#define RXF5SIDH 0x18
#define RXF5SIDL 0x19
#define RXF5EID8 0x1A
#define RXF5EID0 0x1B
#define TEC 0x1C
#define REC 0x1D
//#define CANSTAT 0x1E
//#define CANCTRL 0x1F

#define RXM0SIDH 0x20
#define RXM0SIDL 0x21
#define RXM0EID8 0x22
#define RXM0EID0 0x23
#define RXM1SIDH 0x24
#define RXM1SIDL 0x25
#define RXM1EID8 0x26
#define RXM1EID0 0x27
#define CNF3 0x28
#define CNF2 0x29
#define CNF1 0x2A
#define CANINTE 0x2B
#define CANINTF 0x2C
#define EFLG 0x2D
    #define RX1OVR 7
    #define RX0OVR 6
    #define TXBO 5
    #define TXEP 4
    #define RXEP 3
    #define TXWAR 2
    #define RXWAR 1
    #define EWARN 0
//#define CANSTAT 0x2E
//#define CANCTRL 0x2F

#define TXB0CTRL 0x30
#define TXREQ 3
#define TXB0SIDH 0x31
#define TXB0SIDL 0x32
#define TXB0EID8 0x33
#define TXB0EID0 0x34
#define TXB0DLC 0x35
#define TXB0D0 0x36
#define TXB0D1 0x37
#define TXB0D2 0x38
#define TXB0D3 0x39
#define TXB0D4 0x3A
```

```
#define TXB0D5 0x3B
#define TXB0D6 0x3C
#define TXB0D7 0x3D
//#define CANSTAT 0x3E
//#define CANCTRL 0x3F

#define TXB1CTRL 0x40
#define TXB1SIDH 0x41
#define TXB1SIDL 0x42
#define TXB1EID8 0x43
#define TXB1EID0 0x44
#define TXB1DLC 0x45
#define TXB1D0 0x46
#define TXB1D1 0x47
#define TXB1D2 0x48
#define TXB1D3 0x49
#define TXB1D4 0x4A
#define TXB1D5 0x4B
#define TXB1D6 0x4C
#define TXB1D7 0x4D
//#define CANSTAT 0x4E
//#define CANCTRL 0x4F

#define TXB2CTRL 0x50
#define TXB2SIDH 0x51
#define TXB2SIDL 0x52
#define TXB2EID8 0x53
#define TXB2EID0 0x54
#define TXB2DLC 0x55
#define TXB2D0 0x56
#define TXB2D1 0x57
#define TXB2D2 0x58
#define TXB2D3 0x59
#define TXB2D4 0x5A
#define TXB2D5 0x5B
#define TXB2D6 0x5C
#define TXB2D7 0x5D
//#define CANSTAT 0x5E
//#define CANCTRL 0x5F

#define RXB0CTRL 0x60
#define RXB0SIDH 0x61
#define RXB0SIDL 0x62
#define RXB0EID8 0x63
#define RXB0EID0 0x64
#define RXB0DLC 0x65
#define RXB0D0 0x66
#define RXB0D1 0x67
#define RXB0D2 0x68
#define RXB0D3 0x69
#define RXB0D4 0x6A
#define RXB0D5 0x6B
#define RXB0D6 0x6C
```

```
#define RXB0D7 0x6D
//#define CANSTAT 0x6E
//#define CANCTRL 0x6F

#define RXB1CTRL 0x70
#define RXB1SIDH 0x71
#define RXB1SIDL 0x72
#define RXB1EID8 0x73
#define RXB1EID0 0x74
#define RXB1DLC 0x75
#define RXB1D0 0x76
#define RXB1D1 0x77
#define RXB1D2 0x78
#define RXB1D3 0x79
#define RXB1D4 0x7A
#define RXB1D5 0x7B
#define RXB1D6 0x7C
#define RXB1D7 0x7D
//#define CANSTAT 0x7E
//#define CANCTRL 0x7F

// end of control registers

// Function prototypes:
// Public functions:

/**********************************************************************
Function name:
    mcp_v_spi()

Description:
    This function is the general method used by all other mcp
    functions for moving data through the SPI bus

Parameters:
    Data to be transmitted to the slave CAN controller
    0x00 if just receiving data

Returns:
    Data that was clocked in during transmission of byte
    ignore if only transmitting

**********************************************************************/
uint8_t mcp_v_spi(uint8_t data);

/**********************************************************************
Function name:
    mcp_v_CANSend()

Description:
    This function builds and sends CAN messages through buffer 0 or 1
    It is best suited for a protocol that uses SIDH to address nodes
    and that designates the first byte of data in each message as a
    command byte

Parameters:
    buffer - designates which buffer to load with data 0 or 1
    recipients - loaded with the address of the node to receive message
    command - byte informing recipient of how to handle data
    argcount - number of data bytes in payload not including command
    *arg - array containing payload data

Returns:
    none

*********************************************************************/
void mcp_v_CANSend(uint8_t buffer, uint8_t recipients, uint8_t command, uint8_t argcount,
uint8_t *arg);

/*********************************************************************
Function name:
    mcp_v_CANReceive()

Description:
    receives data from buffer 0

Parameters:
    array where message will be stored

Returns:
    none

*********************************************************************/
void mcp_v_CANReceive(uint8_t *recdata);

/*********************************************************************
Function name:
    mcp_v_CANSetup()

Description:
    this function makes sure that all of the registers are correctly
    setup during initialization

Parameters:
    address - address of register being manipulated
    data - data to be stored in address

Returns:
    none

*********************************************************************/
void mcp_v_CANSetup(uint8_t address, uint8_t data);

/*********************************************************************
Function name:
    mcp_v_reset()
```

```
Description:
    Sends the reset command to the MCP2515.

Parameters:
    none

Returns:
    none

**********************************************************************/
void mcp_v_reset(void);

/**********************************************************************
Function name:
    mcp_v_writebyte()

Description:
    Writes a byte to a specified address in the MCP2515.

Parameters:
    address - location of data to be written to.
    data - byte to write to address.

Returns:
    none

**********************************************************************/
void mcp_v_writebyte(uint8_t address, uint8_t data);

/**********************************************************************
Function name:
    mcp_u8_readbyte()

Description:
    Reads a byte from specified address in MCP2515.

Parameters:
    address - location of byte to read.

Returns:
    data byte from address.

**********************************************************************/
uint8_t mcp_u8_readbyte(uint8_t address);

/**********************************************************************
Function name:
    mcp_v_bitmodify()

Description:
    Change a bit or bits at address in the MCP2515.
```

```
Parameters:
    address - location of byte to change.
    mask - 1's set which bits to modify.
    data - values to write to bits. A 1 sets a bit, a 0 clears it.

Returns:
    none

*********************************************************************/
void mcp_v_bitmodify(uint8_t address, uint8_t mask, uint8_t data);

/*********************************************************************
Function name:
    mcp_v_loadtxbid()

Description:
    Loads selected transmit buffer with a message, starting with identifier.

Parameters:
    buffer - transmit buffer to load, 0 to 2.
    SIDH - Standard identifier high byte.
    SIDL - Standard identifier low byte.
    EID8 - Extended identifier high byte.
    EID0 - Extended identifier low byte.
    nBytes - number of bytes in data packet.
    *data - pointer to data to load.

Returns:
    none

*********************************************************************/
void mcp_v_loadtxbid(uint8_t buffer, uint8_t SIDH, uint8_t SIDL, uint8_t EID8, uint8_t EID0,
uint8_t nBytes, uint8_t *data);

/*********************************************************************
Function name:
    mcp_v_rts()

Description:
    Sends the Request-to-Send instruction to the MCP2515.
    The MCP2515 will transmit the contents of the selected transmit
    buffers at the soonest possible time.

Parameters:
    buffers - bit 0 -- Transmit Buffer 0
            - bit 1 -- Transmit Buffer 1
            - bit 2 -- Transmit Buffer 2

Returns:
    none

NOTE:
    a parameter of 0 will result in nothing being transmitted.
    any parameter higher than 7 will give strange results.
    there is no check for these conditions.

*********************************************************************/
void mcp_v_rts(uint8_t buffers);

/*********************************************************************
Function name:
    mcp_u8_getstatus()

Description:
    Retreive status byte from MCP2515

Parameters:
    none

Returns:
    Status

*********************************************************************/
uint8_t mcp_u8_getstatus(void);

/*********************************************************************
Function name:
    mcp_v_readrxb0()

Description:
    Reads data from receive buffer 0 into location pointed to be *data.

Parameters:
    *data - pointer to location to store received data.

Returns:
    none

*********************************************************************/
void mcp_v_readrxb0(uint8_t *data);

/*********************************************************************
Function name:
    mcp_v_readrxb1()

Description:
    Reads data from receive buffer 1 into location pointed to be *data.

Parameters:
    *data - pointer to location to store received data.

Returns:
    none

*********************************************************************/
void mcp_v_readrxb1(uint8_t *data);
```

```c
// private functions, DO NOT USE!
void mcp_v_readrxb(uint8_t *data);

void mcp_v_can_init(uint8_t address);

#endif
```

```
/*********************************************************

IMCP_Defs2.h Version 1.4

Inter Module Communication Protocol Definitions

Revision history located below definitions

*********************************************************/



#ifndef IMCP2_DEFS_H
#define IMCP2_DEFS_H

//************_rules_******************

//Under the IMCP2 protocol only standard messages will be supported. An example of this type of
message can be seen on page 9 of the MCP2515 datasheet.


//Only bits 10 - 3 of the standard identifier will be considered for the addressing of the
modules this is to make addressing modules and setup of filters

//easier because the Standard Identifier high bytes need to be read or written to, to address a
module. (this can be seen in the initialization routine of

//the MCP2515 which is included in the file MCP2515.h) SIDH is defined on pages 19 and 29 of
the MCP2515 datasheet.


//Each person whose module is responsible for providing data on the network for the car should
define how their data will be made available on the network.


//Each data message will be requested by putting the Data_Request command (0x01) in byte 0 in
the Data Field followed by the command id number of the message

//being requested in byte 1 of the Data Field. The corresponding registers can be seen in the
Register Map on page 61 of the MCP2515 Datasheet.


// Handshake_Request [Data_Request] [Handshake][Requestor's Address]

    //0x00 and 0xFF are reserved, do not use them for commands

    #define RESERVED_LOW 0x00
    #define RESERVED_HIGH 0xFF

//(0x01-0x1F) General Commands for all Modules
#define CAN_DATA_REQUEST    0x01
#define CAN_ACTION_REQUEST  0x02
#define CAN_HANDSHAKE       0x03
// Handshake_Request [CAN_DATA_REQUEST] [CAN_HANDSHAKE][Requestor's Address]
// Returns: [CAN_HANDHAKE][Responder's Address]

// -------------------------------- (0x20-0x2F) Engine Control Unit Module
--------------------------------
#define CAN_ECU_DATA        0x20 //[Brake Press. R][Brake Press. F][Throttle PS][W. Temp][A.
Temp][Batt. Volt]
    //Send:[CAN_DATA_REQUEST][CAN_ECU_DATA][Requestor's Address]
    //Receive:[CAN_ECU_DATA][Brake Press. R][Brake Press. F][Throttle PS][W. Temp][A.
Temp][Batt. Volt]




// (0x30-0x3F) Dash Module Commands
#define CAN_DASH_DATA       0x30    //[RPM.H][RPM.L]
    //Send: [CAN_DATA_REQUEST][CAN_DASH_DATA][Requestor's Address]
    //Receive: [CAN_DASH_DATA][RPM.H][RPM.L]

// (0x40-0x4F) Datalogger Module Commands
#define CAN_DAQ_START       0x40
    //Send: [CAN_ACTION_REQUEST][CAN_DAQ_START][Char1][Char2][Char3][Char4][Char5][Char6][Char7]
#define CAN_DAQ_STOP        0x41
    //Send: [CAN_ACTION_REQUEST][CAN_DAQ_STOP]




// -------------------------------- (0x50-0x5F) Traction Control System Module
--------------------------------
#define CAN_TCS_FRONT    0x50    //[FRRPM1][FRRPM0][FLRPM1][FLRPM0][SWAG]
    //Send: [CAN_DATA_REQUEST][CAN_TCS_FRONT][Requestor's Address]
    //Receive: [CAN_TCS_FRONT][FRRPM1][FRRPM0][FLRPM1][FLRPM0][SWAG]

#define CAN_TCS_REAR     0x51    //[RRRPM1][RRRPM0][RLRPM1][RLRPM0][SLIP1][SLIP0]
    //Send: [CAN_DATA_REQUEST][CAN_TCS_REAR][Requestor's Address]
    //Receive: [CAN_TCS_REAR][RRRPM1][RRRPM0][RLRPM1][RLRPM0][SLIP1][SLIP0]

#define CAN_TCS_STSLP 0x52
//Send: [CAN_TCS_STSLP][REF_SLIP][Requestor's Address]
    //Receive: [CAN_TCS_STSLP][REF_SLIP]

#define CAN_TCS_LCTRL 0x53
//Send: [CAN_TCS_LCTRL][LC_SET][Requestor's Address] -- LC_Set is a filler here
    //Receive: [CAN_TCS_LCTRL][LC_SET] -- LC_Set is 1 and responds after launch controller is set

#define CAN_TCS_LCOFF 0x54
//Receive: [CAN_TCS_LCOFF]




// -------------------------------------------- (0x60-0x6F) Gps Module
--------------------------------------------
#define CAN_GPS_LAT      0x60    //[LAT_D1][LAT_D0][LAT_M3][LAT_M2][LAT_M1][LAT_M0]
    //Send: [CAN_DATA_REQUEST][CAN_GPS_LAT][Requestor's Address]
    //Receive: [CAN_GPS_LAT][LAT_D1][LAT_D0][LAT_M3][LAT_M2][LAT_M1][LAT_M0]
```

```
#define CAN_GPS_LONG     0x61    //[LONG_D1][LONG_D0][LONG_M3][LONG_M2][LONG_M1][LONG_M0]
    //Send: [CAN_DATA_REQUEST][CAN_GPS_LONG][Requestor's Address]
    //Receive: [CAN_GPS_LONG][LONG_D1][LONG_D0][LONG_M3][LONG_M2][LONG_M1][LONG_M0]

#define CAN_GPS_DT       0x62    //[DAY][MONTH][YEAR][HOUR][MIN][SEC]
    //Send: [CAN_DATA_REQUEST][CAN_GPS_DT][Requestor's Address]
    //Receive: [CAN_GPS_DT][DAY][MONTH][YEAR][HOUR][MIN][SEC]







// (0x70-0x7F) Wireless Module

// (0x80-0x8F) Accelerometer

// (0x90-0x9F) Broadcast Messages

// (0xA0-0xAF) Bootloader Messages

// (0xB0-0xBF) Suspension & Temperature Module
#define CAN_TIRE_TEMPS      0xB0
    //Send:[CAN_DATA_REQUEST][CAN_TIRE_TEMPS][Requestor's Address]
    //Receive: [CAN_TIRE_FL_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
    //Receive: [CAN_TIRE_FR_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
    //Receive: [CAN_TIRE_RL_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
    //Receive: [CAN_TIRE_RR_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
#define CAN_TIRE_FL_TEMPS   0xB1
    //Send:[CAN_DATA_REQUEST][CAN_TIRE_FL_TEMPS][Requestor's Address]
    //Receive: [CAN_TIRE_FL_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
#define CAN_TIRE_FR_TEMPS   0xB2
    //Send:[CAN_DATA_REQUEST][CAN_TIRE_FR_TEMPS][Requestor's Address]
    //Receive: [CAN_TIRE_FR_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
#define CAN_TIRE_RL_TEMPS   0xB3
    //Send:[CAN_DATA_REQUEST][CAN_TIRE_RL_TEMPS][Requestor's Address]
    //Receive: [CAN_TIRE_RL_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
#define CAN_TIRE_RR_TEMPS   0xB4
    //Send:[CAN_DATA_REQUEST][CAN_TIRE_RR_TEMPS][Requestor's Address]
    //Receive: [CAN_TIRE_RR_TEMPS][Outer.H][Outer.L][Middle.H][Middle.L][Inner.H][Inner.L]
#define CAN_SUSP_POSITION   0xB5
    //Send:[CAN_DATA_REQUEST][SUSPENSION_POSITION][Requestor's Address]
    //Receive: [SUSPENSION_POSITION][FL.L][FR.L][RL.L][RR.L][FL.H2,FR.H2,RL.H2,RR.H2]    //A2D
is 10Bit,High 2 bits are packed in byte 6



// CAN Received message structure

// each of these is the position in the returned message array where the described item is
located
```

```
// for example, the first DATA byte is located in the 6th element of the returned message array

#define CAN_R_SIDH 0 // Standard Identifier High

#define CAN_R_SIDL 1 // Standard Identifier Low

#define CAN_R_EID8 2 // Extended ID High (unused)

#define CAN_R_EID0 3 // Extended ID Low (unused)

#define CAN_R_M_size 4 // Message size (number of bytes)

#define CAN_R_Command 5 // IMCP command

#define CAN_R_Arg1 6 // Argument 1

#define CAN_R_Arg2 7 // Argument 2

#define CAN_R_Arg3 8 // Argument 3

#define CAN_R_Arg4 9 // Argument 4

#define CAN_R_Arg5 10 // Argument 5

#define CAN_R_Arg6 11 // Argument 6

#define CAN_R_Arg7 12 // Argument 7


// CAN Sending message structure

// The data for the message being sent has a smaller pointer as SIDH,SIDL,EID8,EID0,nBytes are
populated in the function and not the data pointer

#define CAN_S_Arg1 0 // Argument 1

#define CAN_S_Arg2 1 // Argument 2

#define CAN_S_Arg3 2 // Argument 3

#define CAN_S_Arg4 3 // Argument 4

#define CAN_S_Arg5 4 // Argument 5

#define CAN_S_Arg6 5 // Argument 6

#define CAN_S_Arg7 6 // Argument 7

// Module Addresses

// 0 will not be used

#define CAN_ADDR_WRL  1
```

```
#define CAN_ADDR_GPS  2

#define CAN_ADDR_DAQ  4    //CompactRIO Datalogger

#define CAN_ADDR_ECU  5

#define CAN_ADDR_DASH 6

#define CAN_ADDR_TCS  7

#define CAN_ADDR_SUSP 8    //Suspension Travel & Tire Temp module

#define CAN_ADDR_BROADCAST 0xFE //this may be used in the future


#endif


/***************Revision History*****************************************************************

Date: 2010/02/23 Created by:     MM Version: 1.0 Created Document
Date: 2010/03/05 Modified by:    MS Version: 1.1 Created Command Sections, defined new Ecu Module
                                                 and General Commands
Date: 2010/03/07 Modified by:    MS Version: 1.2 Added Command to get RPM bytes from the Dash
module
Date: 2010/03/15 Modified by:    MN Version: 1.3 Added Suspension & Tire Temp module address
                                                 Added Commands to get susp position & tire temps
Date: 2010/03/15 Modified by:    MN Version: 1.4 Changed CAN Sending message structure to reflect
                                                 updates to the send message function, Changed
module
                                                 address syntax to allow for easier parsing.
*********************************************************************************************/
```