Appendix B

# Code in VPU:

can.h

#ifndef \_CAN\_H\_

#define \_CAN\_H\_

#define POOLSIZE 14

#define DATASIZE 5

#define ENG\_SPEED\_ (0x15)

#define ENG\_RPM\_ (0x16)

#define ENG\_OILPRESSURE\_ (0x21)

#define ENG\_OILLEVEL\_ (0x20)

#define ENG\_TEMP\_ (0x25)

#define ENG\_GEAR\_POS\_ (0x26)

#define BD\_ (0x30)

#define SAF\_ (0x32)

#define INF\_OT\_ (0x64)

#define INF\_TIME\_ (0x65)

#define ENG\_GAS\_ (0x22)

#define INF\_ODO\_ (0x60)

#define INF\_TRIP\_ (0x58)

#define INF\_GEN\_ (0x63)

typedef union{

struct{

uint16\_t ID;

uint8\_t LENGTH;

uint8\_t DATA[DATASIZE];

};

*uint64\_t* reg;

} DATAPOOL\_TYPE;

typedef struct {

\_\_IO DATAPOOL\_TYPE Pool[POOLSIZE];

} CAN\_DATAPOOL;

void initDatapool();

void initDatapoolIDs();

void setDatapool(uint16\_t id, uint8\_t length, uint8\_t data[]);

void setMessage(uint16\_t id, uint8\_t data1, uint8\_t data2, uint8\_t data3, uint8\_t data4, uint8\_t data5);

#endif /\* \_CAN\_H\_ \*/

can.c

#include "sam.h"

#include "can.h"

CAN\_DATAPOOL datapool;

CAN\_DATAPOOL \*ptrCAN = &datapool;

void initDatapool() {

for(uint8\_t i = 0; i < POOLSIZE; i++) {

ptrCAN->Pool[i].ID = 0;

ptrCAN->Pool[i].LENGTH = 0;

for(uint8\_t j = 0; j < DATASIZE; j++) {

ptrCAN->Pool[i].DATA[j] = 0;

}

}

}

void initDatapoolIDs() {

ptrCAN->Pool[0].ID = ENG\_SPEED\_;

ptrCAN->Pool[0].LENGTH = DATASIZE;

ptrCAN->Pool[0].DATA[0] = 0x0A;

ptrCAN->Pool[0].DATA[1] = 0x00;

ptrCAN->Pool[1].ID = ENG\_RPM\_;

ptrCAN->Pool[1].LENGTH = DATASIZE;

ptrCAN->Pool[1].DATA[0] = 0x40;

ptrCAN->Pool[1].DATA[1] = 0x1F;

ptrCAN->Pool[2].ID = ENG\_OILPRESSURE\_;

ptrCAN->Pool[2].LENGTH = DATASIZE;

ptrCAN->Pool[2].DATA[0] = 0x32;

ptrCAN->Pool[2].DATA[1] = 0x00;

ptrCAN->Pool[3].ID = ENG\_OILLEVEL\_;

ptrCAN->Pool[3].LENGTH = DATASIZE;

ptrCAN->Pool[3].DATA[0] = 0x32;

ptrCAN->Pool[3].DATA[1] = 0x00;

ptrCAN->Pool[4].ID = ENG\_TEMP\_;

ptrCAN->Pool[4].LENGTH = DATASIZE;

ptrCAN->Pool[4].DATA[0] = 0x5A;

ptrCAN->Pool[4].DATA[1] = 0x00;

ptrCAN->Pool[5].ID = ENG\_GEAR\_POS\_;

ptrCAN->Pool[5].LENGTH = DATASIZE;

ptrCAN->Pool[5].DATA[0] = 0x02;

ptrCAN->Pool[6].ID = BD\_;

ptrCAN->Pool[6].LENGTH = DATASIZE;

ptrCAN->Pool[6].DATA[2] = 0x02;

ptrCAN->Pool[6].DATA[4] = 0x01;

ptrCAN->Pool[7].ID = SAF\_;

ptrCAN->Pool[7].LENGTH = DATASIZE;

ptrCAN->Pool[7].DATA[2] = 0x03;

ptrCAN->Pool[7].DATA[4] = 0x01;

ptrCAN->Pool[8].ID = INF\_OT\_;

ptrCAN->Pool[8].LENGTH = DATASIZE;

ptrCAN->Pool[8].DATA[0] = 0x20;

ptrCAN->Pool[9].ID = ENG\_GAS\_;

ptrCAN->Pool[9].LENGTH = DATASIZE;

ptrCAN->Pool[9].DATA[0] = 0x32;

ptrCAN->Pool[10].ID = INF\_TIME\_;

ptrCAN->Pool[10].LENGTH = DATASIZE;

ptrCAN->Pool[10].DATA[0] = 0x03;

ptrCAN->Pool[10].DATA[2] = 0x3B;

ptrCAN->Pool[11].ID = INF\_GEN\_;

ptrCAN->Pool[11].LENGTH = DATASIZE;

ptrCAN->Pool[11].DATA[0] = 0x0F;

ptrCAN->Pool[12].ID = INF\_ODO\_;

ptrCAN->Pool[12].LENGTH = DATASIZE;

ptrCAN->Pool[13].ID = INF\_TRIP\_;

ptrCAN->Pool[13].LENGTH = DATASIZE;

}

void setDatapool(uint16\_t id, uint8\_t length, uint8\_t data[]) {

if(length != 0) {

for(uint8\_t i = 0; i < POOLSIZE; i++) {

if(ptrCAN->Pool[i].ID == id) {

ptrCAN->Pool[i].LENGTH = length;

for(uint8\_t j = 0; j < DATASIZE; j++) {

ptrCAN->Pool[i].DATA[j] = data[j];

}

}

}

}

}

void setMessage(uint16\_t id, uint8\_t data1, uint8\_t data2, uint8\_t data3, uint8\_t data4, uint8\_t data5) {

for(uint8\_t i = 0; i < POOLSIZE; i++) {

if(ptrCAN->Pool[i].ID == id) {

ptrCAN->Pool[i].DATA[0] = data1;

ptrCAN->Pool[i].DATA[1] = data2;

ptrCAN->Pool[i].DATA[2] = data3;

ptrCAN->Pool[i].DATA[3] = data4;

ptrCAN->Pool[i].DATA[4] = data5;

}

}

}

config\_gclk.h

#ifndef CONFIG\_GCLK\_H\_

#define CONFIG\_GCLK\_H\_

#define NULL 0

void gclkInit( );

void gclkGENDIV( uint32\_t factor, uint32\_t id );

void gclkGENCTRL( uint32\_t \_U\_( standby ), uint32\_t \_U\_( selection ), uint32\_t \_U\_( output\_en ),

uint32\_t \_U\_( output\_off ), uint32\_t \_U\_( duty ), uint32\_t \_U\_( enable ), uint32\_t source,

uint32\_t id );

void gclkCLKCTRL( uint16\_t \_U\_( lock ), uint16\_t \_U\_( enable ), uint16\_t generator, uint16\_t id );

#define myGclkInit gclkInit

#endif /\* CONFIG\_GCLK\_H\_ \*/

config\_gclk.c

#include "sam.h"

#include "config\_gclk.h"

void gclkInit( ) {

//gclkGENDIV( 0x0, 0x04 );

//GENEN=1 generator enabled, SRC=0x6 OSC8M oscillator, ID=0x4 GCLKGEN4

gclkGENCTRL( NULL, NULL, NULL, NULL, NULL, 1, GCLK\_GENCTRL\_SRC\_OSC8M\_Val, 0x0 );

gclkGENCTRL( NULL, NULL, NULL, NULL, NULL, 1, GCLK\_GENCTRL\_SRC\_OSC8M\_Val, 0x2 );

//CLKEN=1 enable, GEN=0x4 GCLKGEN4, ID=0x17 SERCOM3\_CORE

//gclkCLKCTRL( NULL, 1, GCLK\_CLKCTRL\_GEN\_GCLK0\_Val, GCLK\_CLKCTRL\_ID\_SERCOM3\_CORE\_Val );

gclkCLKCTRL( NULL, 1, GCLK\_CLKCTRL\_GEN\_GCLK4\_Val, GCLK\_CLKCTRL\_ID\_SERCOMX\_SLOW\_Val );

gclkCLKCTRL( NULL, 1, GCLK\_CLKCTRL\_GEN\_GCLK2\_Val, GCLK\_CLKCTRL\_ID\_TCC2\_TC3\_Val );

}

void gclkGENDIV( uint32\_t factor, uint32\_t id ) {

//Single 32-bit write

GCLK->GENDIV.reg = GCLK\_GENDIV\_DIV( factor ) | GCLK\_GENDIV\_ID( id );

}

void gclkGENCTRL( uint32\_t \_U\_( standby ), uint32\_t \_U\_( selection ), uint32\_t \_U\_( output\_en ),

uint32\_t \_U\_( output\_off ), uint32\_t \_U\_( duty ), uint32\_t \_U\_( enable ), uint32\_t source,

uint32\_t id ) {

//Single 32-bit write

GCLK->GENCTRL.reg = ( \_U\_( standby ) << GCLK\_GENCTRL\_RUNSTDBY\_Pos )

| ( \_U\_( selection ) << GCLK\_GENCTRL\_DIVSEL\_Pos )

| ( \_U\_( output\_en ) << GCLK\_GENCTRL\_OE\_Pos ) | ( \_U\_( output\_off ) << GCLK\_GENCTRL\_OOV\_Pos )

| ( \_U\_( duty ) << GCLK\_GENCTRL\_IDC\_Pos ) | ( \_U\_( enable ) << GCLK\_GENCTRL\_GENEN\_Pos )

| GCLK\_GENCTRL\_SRC( source ) | GCLK\_GENCTRL\_ID( id );

while( GCLK->STATUS.bit.SYNCBUSY ) {}

}

void gclkCLKCTRL( uint16\_t \_U\_( lock ), uint16\_t \_U\_( enable ), uint16\_t generator, uint16\_t id ) {

//Single 16-bit write

GCLK->CLKCTRL.reg = ( \_U\_( lock ) << GCLK\_CLKCTRL\_WRTLOCK\_Pos ) | ( \_U\_( enable ) << GCLK\_CLKCTRL\_CLKEN\_Pos )

| GCLK\_CLKCTRL\_GEN( generator ) | GCLK\_CLKCTRL\_ID( id );

}

config\_pm.h

#ifndef CONFIG\_PM\_H\_

#define CONFIG\_PM\_H\_

void pmInit();

void pmAPBCMASK( uint32\_t value );

#define myPmInit pmInit

#endif /\* CONFIG\_PM\_H\_ \*/

config\_pm.c

#include "sam.h"

#include "config\_pm.h"

void pmInit() {

//APBC clock for SERCOM3 is enabled

pmAPBCMASK( PM\_APBCMASK\_SERCOM3 );

pmAPBCMASK( PM\_APBCMASK\_TC3 );

}

void pmAPBCMASK( uint32\_t value ) {

PM->APBCMASK.reg |= value;

}

config\_port.h

#ifndef CONFIG\_PORT\_H\_

#define CONFIG\_PORT\_H\_

void portInit( );

void portProps( uint8\_t group, uint8\_t pin, uint8\_t value, uint8\_t type );

void portDir( uint32\_t group, uint32\_t offset, uint32\_t value);

void portOut( uint32\_t group, uint32\_t offset, uint32\_t value );

#define myPortInit portInit

#define myPortOut portOut

#endif /\* CONFIG\_PORT\_H\_ \*/

config\_port.c

#include "sam.h"

#include "config\_port.h"

void portInit( ) {

//PROPERTIES: pin configuration and peripheral function

portProps( 0, PIN\_PA22, 0x1, PORT\_PMUX\_PMUXE\_C ); //SDA

portProps( 0, PIN\_PA23, 0x1, PORT\_PMUX\_PMUXO\_C ); //SCL

portProps( 0, PIN\_PA14, 0x1, PORT\_PMUX\_PMUXE\_E ); //PWM PIN

}

void portProps( uint8\_t group, uint8\_t pin, uint8\_t value, uint8\_t type ) {

//GROUP: PuertoA = 0, PIN position = PINCFG register

PORT->Group[ group ].PINCFG[ pin ].reg = value; //configuration

//PERIPHERAL FUNCTION if needed

if( PORT->Group[ group ].PINCFG[ pin ].bit.PMUXEN == 1 ) {

pin = ( pin >> 1 ); //obtain MUX register

PORT->Group[ group ].PMUX[ pin ].reg |= type; //odd or even

}

}

void portDir( uint32\_t group, uint32\_t offset, uint32\_t value ) {

if( offset == PORT\_DIRSET\_OFFSET ) {

PORT->Group[ group ].DIRSET.reg = value; //output direction

} else if( offset == PORT\_DIRCLR\_OFFSET ) {

PORT->Group[ group ].DIRCLR.reg = value; //input direction

}

}

void portOut( uint32\_t group, uint32\_t offset, uint32\_t value ) {

if( offset == PORT\_OUTSET\_OFFSET ) {

PORT->Group[ group ].OUTSET.reg = value; //logic level "1"

} else if( offset == PORT\_OUTCLR\_OFFSET ) {

PORT->Group[ group ].OUTCLR.reg = value; //logic level "0"

}

}

config\_sercomI2C.h

#ifndef CONFIG\_SERCOMI2C\_H\_

#define CONFIG\_SERCOMI2C\_H\_

#define NULL 0

#define SLAVE\_ADDR 0x49

void sercomI2CSInit( );

void sercomI2CSEnable( );

void sercomI2CSCTRLA( uint32\_t time\_out, uint32\_t stretch, uint32\_t speed, uint32\_t s\_extend\_time, uint32\_t sda\_hold,

uint32\_t pin\_usage, uint32\_t standby, uint32\_t mode, uint32\_t enable, uint32\_t reset );

void sercomI2CSCTRLB( uint32\_t acknowledge, uint32\_t command, uint32\_t mode, uint32\_t automatic\_ack, uint32\_t group,

uint32\_t smart );

void sercomI2CSADDR( uint32\_t mask, uint32\_t ten\_bit, uint32\_t address, uint32\_t gen\_call );

#define mySercomI2CSInit sercomI2CSInit

#define mySercomI2CSEnable sercomI2CSEnable

#endif /\* CONFIG\_SERCOMI2C\_H\_ \*/

config\_sercomI2C.c

#include "sam.h"

#include "config\_sercomI2C.h"

void sercomI2CSInit( ) {

sercomI2CSCTRLA( NULL, NULL, 0x0, NULL, 0x3, NULL, NULL, SERCOM\_I2CM\_CTRLA\_MODE\_I2C\_SLAVE\_Val, 0, NULL );

sercomI2CSCTRLB( NULL, NULL, 0x1, 1, NULL, 1 );

sercomI2CSADDR( SLAVE\_ADDR, NULL, NULL, NULL );

}

void sercomI2CSEnable( ) {

SERCOM3->I2CS.INTFLAG.reg = SERCOM\_I2CS\_INTFLAG\_MASK; //clear the FLAGS

SERCOM3->I2CS.CTRLA.reg |= SERCOM\_I2CS\_CTRLA\_ENABLE; //I2C enabled

while( SERCOM3->I2CS.SYNCBUSY.bit.ENABLE ){ } //wait until I2C is enabled

}

void sercomI2CSCTRLA( uint32\_t time\_out, uint32\_t stretch, uint32\_t speed, uint32\_t s\_extend\_time, uint32\_t sda\_hold,

uint32\_t pin\_usage, uint32\_t standby, uint32\_t mode, uint32\_t enable, uint32\_t reset ) {

//Main configurations

SERCOM3->I2CS.CTRLA.reg |= SERCOM\_I2CS\_CTRLA\_MODE( mode );

SERCOM3->I2CS.CTRLA.bit.LOWTOUTEN |= time\_out;

SERCOM3->I2CS.CTRLA.reg |= SERCOM\_I2CS\_CTRLA\_SDAHOLD( sda\_hold ) | SERCOM\_I2CS\_CTRLA\_SPEED( speed );

//Other configurations

SERCOM3->I2CS.CTRLA.bit.SCLSM |= stretch;

SERCOM3->I2CS.CTRLA.bit.SEXTTOEN |= s\_extend\_time;

SERCOM3->I2CS.CTRLA.bit.PINOUT |= pin\_usage;

SERCOM3->I2CS.CTRLA.bit.RUNSTDBY |= standby;

SERCOM3->I2CS.CTRLA.bit.SWRST |= reset;

//SERCOMx must stay disabled until all its configured

SERCOM3->I2CS.CTRLA.bit.ENABLE |= enable;

}

void sercomI2CSCTRLB( uint32\_t acknowledge, uint32\_t command, uint32\_t amode, uint32\_t automatic\_ack, uint32\_t group,

uint32\_t smart ) {

//Main configurations

SERCOM3->I2CS.CTRLB.reg |= SERCOM\_I2CS\_CTRLB\_AMODE( amode );

SERCOM3->I2CS.CTRLB.bit.AACKEN |= automatic\_ack;

SERCOM3->I2CS.CTRLB.bit.SMEN |= smart;

//Other configurations

SERCOM3->I2CS.CTRLB.bit.ACKACT |= acknowledge;

SERCOM3->I2CS.CTRLB.reg |= SERCOM\_I2CS\_CTRLB\_CMD( command );

SERCOM3->I2CS.CTRLB.bit.GCMD |= group;

}

void sercomI2CSADDR( uint32\_t mask, uint32\_t ten\_bit, uint32\_t address, uint32\_t gen\_call ) {

//Main configurations

SERCOM3->I2CS.ADDR.reg |= SERCOM\_I2CS\_ADDR\_ADDRMASK( mask ) | SERCOM\_I2CS\_ADDR\_ADDR( address );

//Other configurations

SERCOM3->I2CS.ADDR.bit.TENBITEN |= ten\_bit;

SERCOM3->I2CS.ADDR.bit.GENCEN |= gen\_call;

}

i2c.h

#ifndef I2C\_H\_

#define I2C\_H\_

uint8\_t i2cSReceive();

uint8\_t i2cSReceive8bit();

void i2cSSend(uint8\_t\* send\_buff, uint8\_t pos);

void i2cSSendID(uint16\_t send\_buff);

void i2cSSend8bit( uint8\_t data );

void i2cSSend16bit( uint16\_t data );

void SAcknowledge( uint32\_t ack );

#define myI2CSReceive i2cSReceive

#define myI2CSSend i2cSSend

#define myI2CSSendID i2cSSendID

#endif /\* I2C\_H\_ \*/

i2c.c

#include "sam.h"

#include "i2c.h"

void SAcknowledge( uint32\_t ack ) {

SERCOM3->I2CS.CTRLB.bit.ACKACT |= ack;

}

uint8\_t i2cSReceive() {

uint8\_t receive\_data;

//CASE 2. Address packet is accepted - Write flag set

SAcknowledge( 0 ); //configure for acknowledge

while( !SERCOM3->I2CS.STATUS.bit.CLKHOLD) {} //waits until salve hold message

if( SERCOM3->I2CS.STATUS.bit.DIR == 0 ) { //if MASTER is transmitting

//Acknowledge for packet is send automatically because AACKEN is enabled

receive\_data = i2cSReceive8bit( ); //receive data packet(s)

//After receiving data, ack is send automatically because SMART MODE is enabled

}

return receive\_data;

SERCOM3->I2CS.INTFLAG.bit.AMATCH |= 1;

SERCOM3->I2CS.INTFLAG.bit.DRDY |= 1;

}

uint8\_t i2cSReceive8bit() {

uint8\_t data;

data = SERCOM3->I2CS.DATA.reg;

while( !SERCOM3->I2CS.INTFLAG.bit.DRDY ) { } //wait until data packet is received

return data; //read character received

}

void i2cSSend(uint8\_t\* send\_buff, uint8\_t pos) {

SAcknowledge( 0 ); //configure for acknowledge

while(!SERCOM3->I2CS.STATUS.bit.CLKHOLD){}

if( SERCOM3->I2CS.STATUS.bit.DIR == 1 ) { //if MASTER is transmitting

//Acknowledge for packet is send automatically because AACKEN is enabled

i2cSSend8bit(send\_buff[pos]); //receive data packet(s)

//After receiving data, ack is send automatically because SMART MODE is enabled

}

SERCOM3->I2CS.INTFLAG.bit.AMATCH |= 1;

SERCOM3->I2CS.INTFLAG.bit.DRDY |= 1;

}

void i2cSSendID(uint16\_t send\_buff) {

SAcknowledge( 0 ); //configure for acknowledge

while(!SERCOM3->I2CS.STATUS.bit.CLKHOLD){}

if( SERCOM3->I2CS.STATUS.bit.DIR == 1 ) { //if MASTER is transmitting

//Acknowledge for packet is send automatically because AACKEN is enabled

i2cSSend16bit(send\_buff); //receive data packet(s)

//After receiving data, ack is send automatically because SMART MODE is enabled

}

SERCOM3->I2CS.INTFLAG.bit.AMATCH |= 1;

SERCOM3->I2CS.INTFLAG.bit.DRDY |= 1;

}

void i2cSSend8bit( uint8\_t data ) {

SERCOM3->I2CS.DATA.reg=SERCOM\_I2CS\_DATA\_DATA(data);

while(!SERCOM3->I2CS.INTFLAG.bit.DRDY){ }

}

void i2cSSend16bit( uint16\_t data ) {

SERCOM3->I2CS.DATA.reg=SERCOM\_I2CS\_DATA\_DATA(data);

while(!SERCOM3->I2CS.INTFLAG.bit.DRDY){

}

}

spi.h

#ifndef SPI\_H\_

#define SPI\_H\_

#define SLAVE\_SD\_CARD 0

#define SLAVE\_CAN\_0 1

#define SLAVE\_CAN\_1 2

void initSPI(void);

uint8\_t spiSend(uint8\_t data);

uint8\_t spiSS(uint8\_t device);

uint8\_t spiSR(uint8\_t device);

#endif /\* SPI\_H\_ \*/

spi.c

#include "sam.h"

#include "spi.h"

void initSPI(void) {

// SLAVE SELECT PIN 8 ARDUINO

REG\_PORT\_DIRSET0 = PORT\_PA06;

REG\_PORT\_OUTSET0 = PORT\_PA06;

// SLAVE SELECT PIN 9 ARDUINO

REG\_PORT\_DIRSET0 = PORT\_PA07;

REG\_PORT\_OUTSET0 = PORT\_PA07;

// SLAVE SELECT PIN 10 ARDUINO

REG\_PORT\_DIRSET0 = PORT\_PA18;

REG\_PORT\_OUTSET0 = PORT\_PA18;

// INIT SPI PROTOCOL

PM->APBCMASK.bit.SERCOM1\_ = 1;

GCLK->CLKCTRL.reg = GCLK\_CLKCTRL\_CLKEN | GCLK\_CLKCTRL\_GEN\_GCLK0 | GCLK\_CLKCTRL\_ID\_SERCOM1\_CORE;

while(GCLK->STATUS.bit.SYNCBUSY);

const SERCOM\_SPI\_CTRLA\_Type ctrla = {

.bit.DORD = 0, // MSB first

.bit.CPHA = 0, // Mode 0

.bit.CPOL = 0,

.bit.FORM = 0, // SPI frame

.bit.DIPO = 3, // MISO on PAD[3]

.bit.DOPO = 0, // MOSI on PAD[0], SCK on PAD[1], SS\_ on PAD[2]

.bit.MODE = 3 // Master

};

SERCOM1->SPI.CTRLA.reg = ctrla.reg;

const SERCOM\_SPI\_CTRLB\_Type ctrlb = {

.bit.RXEN = 1, // RX enabled

.bit.MSSEN = 0, // Manual SC

.bit.CHSIZE = 0 // 8-bit

};

SERCOM1->SPI.CTRLB.reg = ctrlb.reg;

SERCOM1->SPI.BAUD.reg = 2; // Rate is clock / 2

// Mux for SERCOM1 PA16,PA17,PA19

const PORT\_WRCONFIG\_Type wrconfig = {

.bit.WRPINCFG = 1,

.bit.WRPMUX = 1,

.bit.PMUX = MUX\_PA16C\_SERCOM1\_PAD0,

.bit.PMUXEN = 1,

.bit.HWSEL = 1,

.bit.PINMASK = (uint16\_t)((PORT\_PA16 | PORT\_PA17 | PORT\_PA19) >> 16)

};

PORT->Group[0].WRCONFIG.reg = wrconfig.reg;

SERCOM1->SPI.CTRLA.bit.ENABLE = 1;

while(SERCOM1->SPI.SYNCBUSY.bit.ENABLE);

}

uint8\_t spiSend(uint8\_t data) {

uint8\_t ret;

while(SERCOM1->SPI.INTFLAG.bit.DRE == 0);

SERCOM1->SPI.DATA.reg = data;

while(SERCOM1->SPI.INTFLAG.bit.TXC == 0);

while(SERCOM1->SPI.INTFLAG.bit.RXC == 0);

ret = SERCOM1->SPI.DATA.reg;

return ret;

}

uint8\_t spiSS(uint8\_t device)

{

spiSend(0xFF);

int ret = 0;

switch (device)

{

case SLAVE\_SD\_CARD:

REG\_PORT\_OUTCLR0 = PORT\_PA18;

break;

case SLAVE\_CAN\_0:

REG\_PORT\_OUTCLR0 = PORT\_PA07;

break;

case SLAVE\_CAN\_1:

REG\_PORT\_OUTCLR0 = PORT\_PA06;

break;

default:

ret = 1;

break;

}

return ret;

}

uint8\_t spiSR(uint8\_t device)

{

int ret = 0;

switch (device)

{

case SLAVE\_SD\_CARD:

REG\_PORT\_OUTSET0 = PORT\_PA18;

break;

case SLAVE\_CAN\_0:

REG\_PORT\_OUTSET0 = PORT\_PA07;

break;

case SLAVE\_CAN\_1:

REG\_PORT\_OUTSET0 = PORT\_PA06;

break;

default:

ret = 1;

break;

}

return ret;

}

timers.h

#ifndef TIMERS\_H\_

#define TIMERS\_H\_

#define DUTYCHANGE 655

#define BASECYCLE 49152

void timersInit();

void decrease\_duty\_cycle ();

void increase\_duty\_cycle ();

void setPWM (int duty);

#endif /\* TIMERS\_H\_ \*/

timers.c

#include "sam.h"

#include "timers.h"

void timersInit(){

//Configure delay

TC4->COUNT16.CTRLA.bit.MODE = 0;

TC4->COUNT16.CTRLA.bit.PRESCALER = 4;

TC4->COUNT16.INTFLAG.bit.OVF = 1; //Limpiar la bandera de overflow

TC4->COUNT16.COUNT.bit.COUNT = 60536; //Inicializar el COUNT

TC4->COUNT16.CTRLA.bit.PRESCSYNC = 1; //Configurar la sincronización del prescalador y el counter

TC4->COUNT16.CTRLA.bit.ENABLE = 1; //Habilitar el timer

while(TC4->COUNT16.STATUS.bit.SYNCBUSY);

//Configure PWM

TC3->COUNT16.CTRLA.bit.PRESCSYNC = 2;

TC3->COUNT16.CTRLA.bit.WAVEGEN = 2;

TC3->COUNT16.CTRLA.bit.PRESCALER = 0;

TC3->COUNT16.CTRLA.bit.MODE = 0;

TC3->COUNT16.CC[0].reg = BASECYCLE;

TC3->COUNT16.CTRLA.bit.ENABLE = 1; //Habilitar el timer

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

}

void increase\_duty\_cycle () {

TC3->COUNT16.CTRLA.bit.ENABLE = 0;

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

uint16\_t duty = TC3->COUNT16.CC[0].reg;

if (TC3->COUNT16.CC[0].reg < 52427){

duty = duty + DUTYCHANGE;

TC3->COUNT16.CC[0].reg = duty;

}

TC3->COUNT16.CTRLA.bit.ENABLE = 1;

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

}

void decrease\_duty\_cycle () {

TC3->COUNT16.CTRLA.bit.ENABLE = 0;

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

uint16\_t duty = TC3->COUNT16.CC[0].reg;

if (TC3->COUNT16.CC[0].reg > 49152){

duty = duty - DUTYCHANGE;

TC3->COUNT16.CC[0].reg = duty;

}

TC3->COUNT16.CTRLA.bit.ENABLE = 1;

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

}

void setPWM (int duty) {

TC3->COUNT16.CTRLA.bit.PRESCSYNC = 2;

TC3->COUNT16.CTRLA.bit.WAVEGEN = 2;

TC3->COUNT16.CTRLA.bit.PRESCALER = 0;

TC3->COUNT16.CTRLA.bit.MODE = 0;

TC3->COUNT16.CC[0].reg = (duty \* DUTYCHANGE);

TC3->COUNT16.CTRLA.bit.ENABLE = 1;

while(TC3->COUNT16.STATUS.bit.SYNCBUSY);

}

uart.h

#ifndef UART\_H\_

#define UART\_H\_

//SALIDA TX ENTRADA RX

#define PA10\_TX 0x01

#define PA11\_RX 0x07

#define Type\_C 0x2u;

//calculo BAUDRATE con Fref=8Mhz, Fbaud=9600 S=16

//#define BAUDRATESET(x) uint64\_t br = (uint64\_t)65536 \* (8000000 - 16 \* x) / 8000000;

void initUART(void);

#endif /\* UART\_H\_ \*/

uart.c

#include "sam.h"

#include "uart.h"

/\* Replace with your library code \*/

void initUART(void) {

/\* APBCMASK \*/

/\* SERCOM 0 enable\*/

PM->APBCMASK.reg |= PM\_APBCMASK\_SERCOM0;

/\*GCLK configuration for sercom0 module: using generic clock generator 0, ID for sercom0, enable GCLK\*/

GCLK->GENCTRL.reg=GCLK\_GENCTRL\_SRC\_OSC8M|GCLK\_GENCTRL\_ID(0)|

GCLK\_GENCTRL\_GENEN;

GCLK->CLKCTRL.reg = GCLK\_CLKCTRL\_ID(SERCOM0\_GCLK\_ID\_CORE) |

GCLK\_CLKCTRL\_CLKEN | GCLK\_CLKCTRL\_GEN(0);

/\*CONFIGURAR PA10 y PA11 como salidas tipo C (SERCOM 0) \*/

PORT->Group[0].PINCFG[PIN\_PA10].reg= PA10\_TX;

PORT->Group[0].PINCFG[PIN\_PA11].reg= PA11\_RX;

PORT->Group[0].PMUX[5].bit.PMUXE=Type\_C;

PORT->Group[0].PMUX[5].bit.PMUXO=Type\_C;

//desahibilitar UART para inicializacion

SERCOM0->USART.CTRLA.bit.ENABLE=0;

/\* configure SERCOM0 module for UART as Standard Frame, 8 Bit size, No parity, BAUDRATE:9600\*/

/\*DORD 0x1 MSB first

MODE 0x1 USART with internal clock

RXPO 0x3 SERCOM PAD[3] is used for data reception

TXPO 0x1 SERCOM PAD[2] TX pin location

DEFINES:

1)SERCOM\_USART\_CTRLA\_DORD (\_U(0x1) << SERCOM\_USART\_CTRLA\_DORD\_Pos)

2)SERCOM\_USART\_CTRLA\_MODE\_USART\_INT\_CLK (0x1 << 2)

3)SERCOM\_USART\_CTRLA\_RXPO(value) (SERCOM\_USART\_CTRLA\_RXPO\_Msk & ((value) << SERCOM\_USART\_CTRLA\_RXPO\_Pos) \*/

SERCOM0->USART.CTRLA.reg =

SERCOM\_USART\_CTRLA\_DORD | SERCOM\_USART\_CTRLA\_MODE\_USART\_INT\_CLK |

SERCOM\_USART\_CTRLA\_RXPO(3) | SERCOM\_USART\_CTRLA\_TXPO(1);

while(SERCOM0->USART.SYNCBUSY.reg){} // waiting loading

*uint64\_t* br = (*uint64\_t*)65536 \* (8000000 - 16 \* 9600) / 8000000;

SERCOM0->USART.BAUD.reg = (uint16\_t)br;

while(SERCOM0->USART.SYNCBUSY.reg){} // waiting loading

SERCOM0->USART.CTRLB.reg = SERCOM\_USART\_CTRLB\_RXEN | SERCOM\_USART\_CTRLB\_TXEN | SERCOM\_USART\_CTRLB\_CHSIZE(0/\*8 bits\*/);

while(SERCOM0->USART.SYNCBUSY.reg){} // waiting loading

SERCOM0->USART.CTRLA.reg |= SERCOM\_USART\_CTRLA\_ENABLE;

}

main.c

#include "sam.h"

#include "config\_gclk.h"

#include "config\_pm.h"

#include "config\_port.h"

#include "config\_sercomI2C.h"

#include "i2c.h"

#include "uart.h"

#include "myprintf.h"

#include "spi.h"

#include "mcp\_can.h"

#include "mcp\_can\_dfs.h"

#include "can.h"

#include "timers.h"

#define CAN0\_SPEED CAN\_500KBPS

#define CAN1\_SPEED CAN\_100KBPS

#define DUTYCYCLE 60

extern CAN\_DATAPOOL datapool;

extern CAN\_DATAPOOL \*ptrCAN;

uint8\_t pos, data\_received, pool, i;

uint8\_t data[5] = {0, 0, 0, 0, 0};

uint16\_t id = 0;

uint8\_t len = 0;

int main(void) {

/\* Initialize the SAM system \*/

SystemInit();

/\* Initialize UART \*/

initUART();

SYSCTRL->OSC8M.bit.PRESC = 0;

/\* Configure the PORTto define pins properties and directions \*/

portInit();

/\* Configure PM to enable APBx clock for the corresponding peripheral \*/

pmInit();

/\* Configure GCLK to enable and define clock source for the corresponding peripheral \*/

gclkInit();

/\* Configure SERCOMx to define operating modes, pads, baud rate, character size, etc... \*/

sercomI2CSInit();

/\* Initialize PWM timers \*/

timersInit();

/\* Clear the peripheral FLAGs and enable the peripheral \*/

sercomI2CSEnable();

initSPI();

/\* Initialize Datapool \*/

initDatapool();

initDatapoolIDs();

/\* Initialize CAN \*/

canBegin(SLAVE\_SD\_CARD, CAN0\_SPEED);

///\* Sets the PWM duty cycle \*/

setPWM(DUTYCYCLE);

while (1) {

readMsgBufID(&id, &len, data);

setDatapool(id, len, data);

//for(int i = 0; i < POOLSIZE; i++) {

//myprintf("ID: %02X DLC: %d BUFFER: %02X %02X %02X %02X %02X\n", ptrCAN->Pool[i].ID, ptrCAN->Pool[i].LENGTH, ptrCAN->Pool[i].DATA[0], ptrCAN->Pool[i].DATA[1],

//ptrCAN->Pool[i].DATA[2], ptrCAN->Pool[i].DATA[3], ptrCAN->Pool[i].DATA[4]);

//}

//myprintf("\n\n");

///\* Backlight setting \*/

//setPWM(ptrCAN->Pool[6].DATA[0]);

/\* Datapool \*/

for (pos = 0; pos < POOLSIZE; pos++) {

for (i = 0; i <= DATASIZE; i++){

data\_received = myI2CSReceive();

if (data\_received == 0) {

pool = pos;

myI2CSSendID(ptrCAN->Pool[pos].ID);

}

else{

if ( data\_received > 0 && data\_received <= DATASIZE) {

data\_received--;

myI2CSSend(ptrCAN->Pool[pool].DATA, data\_received);

}

}

}

}

}

}

mcp\_can\_dfs.h

#ifndef \_MCP2515DFS\_H\_

#define \_MCP2515DFS\_H\_

#define MAX\_CHAR\_IN\_MESSAGE 8

/\*

\* Begin mt

\*/

#define TIMEOUTVALUE 50

#define MCP\_SIDH 0

#define MCP\_SIDL 1

#define MCP\_EID8 2

#define MCP\_EID0 3

#define MCP\_TXB\_EXIDE\_M 0x08 /\* In TXBnSIDL \*/

#define MCP\_DLC\_MASK 0x0F /\* 4 LSBits \*/

#define MCP\_RTR\_MASK 0x40 /\* (1<<6) Bit 6 \*/

#define MCP\_RXB\_RX\_ANY 0x60

#define MCP\_RXB\_RX\_EXT 0x40

#define MCP\_RXB\_RX\_STD 0x20

#define MCP\_RXB\_RX\_STDEXT 0x00

#define MCP\_RXB\_RX\_MASK 0x60

#define MCP\_RXB\_BUKT\_MASK (1<<2)

/\*

\*\* Bits in the TXBnCTRL registers.

\*/

#define MCP\_TXB\_TXBUFE\_M 0x80

#define MCP\_TXB\_ABTF\_M 0x40

#define MCP\_TXB\_MLOA\_M 0x20

#define MCP\_TXB\_TXERR\_M 0x10

#define MCP\_TXB\_TXREQ\_M 0x08

#define MCP\_TXB\_TXIE\_M 0x04

#define MCP\_TXB\_TXP10\_M 0x03

#define MCP\_TXB\_RTR\_M 0x40 /\* In TXBnDLC \*/

#define MCP\_RXB\_IDE\_M 0x08 /\* In RXBnSIDL \*/

#define MCP\_RXB\_RTR\_M 0x40 /\* In RXBnDLC \*/

#define MCP\_STAT\_RXIF\_MASK (0x03)

#define MCP\_STAT\_RX0IF (1<<0)

#define MCP\_STAT\_RX1IF (1<<1)

#define MCP\_EFLG\_RX1OVR (1<<7)

#define MCP\_EFLG\_RX0OVR (1<<6)

#define MCP\_EFLG\_TXBO (1<<5)

#define MCP\_EFLG\_TXEP (1<<4)

#define MCP\_EFLG\_RXEP (1<<3)

#define MCP\_EFLG\_TXWAR (1<<2)

#define MCP\_EFLG\_RXWAR (1<<1)

#define MCP\_EFLG\_EWARN (1<<0)

#define MCP\_EFLG\_ERRORMASK (0xF8) /\* 5 MS-Bits \*/

/\*

\* Define MCP2515 register addresses

\*/

#define MCP\_RXF0SIDH 0x00

#define MCP\_RXF0SIDL 0x01

#define MCP\_RXF0EID8 0x02

#define MCP\_RXF0EID0 0x03

#define MCP\_RXF1SIDH 0x04

#define MCP\_RXF1SIDL 0x05

#define MCP\_RXF1EID8 0x06

#define MCP\_RXF1EID0 0x07

#define MCP\_RXF2SIDH 0x08

#define MCP\_RXF2SIDL 0x09

#define MCP\_RXF2EID8 0x0A

#define MCP\_RXF2EID0 0x0B

#define MCP\_CANSTAT 0x0E

#define MCP\_CANCTRL 0x0F

#define MCP\_RXF3SIDH 0x10

#define MCP\_RXF3SIDL 0x11

#define MCP\_RXF3EID8 0x12

#define MCP\_RXF3EID0 0x13

#define MCP\_RXF4SIDH 0x14

#define MCP\_RXF4SIDL 0x15

#define MCP\_RXF4EID8 0x16

#define MCP\_RXF4EID0 0x17

#define MCP\_RXF5SIDH 0x18

#define MCP\_RXF5SIDL 0x19

#define MCP\_RXF5EID8 0x1A

#define MCP\_RXF5EID0 0x1B

#define MCP\_TEC 0x1C

#define MCP\_REC 0x1D

#define MCP\_RXM0SIDH 0x20

#define MCP\_RXM0SIDL 0x21

#define MCP\_RXM0EID8 0x22

#define MCP\_RXM0EID0 0x23

#define MCP\_RXM1SIDH 0x24

#define MCP\_RXM1SIDL 0x25

#define MCP\_RXM1EID8 0x26

#define MCP\_RXM1EID0 0x27

#define MCP\_CNF3 0x28

#define MCP\_CNF2 0x29

#define MCP\_CNF1 0x2A

#define MCP\_CANINTE 0x2B

#define MCP\_CANINTF 0x2C

#define MCP\_EFLG 0x2D

#define MCP\_TXB0CTRL 0x30

#define MCP\_TXB1CTRL 0x40

#define MCP\_TXB2CTRL 0x50

#define MCP\_RXB0CTRL 0x60

#define MCP\_RXB0SIDH 0x61

#define MCP\_RXB1CTRL 0x70

#define MCP\_RXB1SIDH 0x71

#define MCP\_TX\_INT 0x1C // Enable all transmit interrup ts

#define MCP\_TX01\_INT 0x0C // Enable TXB0 and TXB1 interru pts

#define MCP\_RX\_INT 0x03 // Enable receive interrupts

#define MCP\_NO\_INT 0x00 // Disable all interrupts

#define MCP\_TX01\_MASK 0x14

#define MCP\_TX\_MASK 0x54

/\*

\* Define SPI Instruction Set

\*/

#define MCP\_WRITE 0x02

#define MCP\_READ 0x03

#define MCP\_BITMOD 0x05

#define MCP\_LOAD\_TX0 0x40

#define MCP\_LOAD\_TX1 0x42

#define MCP\_LOAD\_TX2 0x44

#define MCP\_RTS\_TX0 0x81

#define MCP\_RTS\_TX1 0x82

#define MCP\_RTS\_TX2 0x84

#define MCP\_RTS\_ALL 0x87

#define MCP\_READ\_RX0 0x90

#define MCP\_READ\_RX1 0x94

#define MCP\_READ\_STATUS 0xA0

#define MCP\_RX\_STATUS 0xB0

#define MCP\_RESET 0xC0

/\*

\* CANCTRL Register Values

\*/

#define MODE\_NORMAL 0x00

#define MODE\_SLEEP 0x20

#define MODE\_LOOPBACK 0x40

#define MODE\_LISTENONLY 0x60

#define MODE\_CONFIG 0x80

#define MODE\_POWERUP 0xE0

#define MODE\_MASK 0xE0

#define ABORT\_TX 0x10

#define MODE\_ONESHOT 0x08

#define CLKOUT\_ENABLE 0x04

#define CLKOUT\_DISABLE 0x00

#define CLKOUT\_PS1 0x00

#define CLKOUT\_PS2 0x01

#define CLKOUT\_PS4 0x02

#define CLKOUT\_PS8 0x03

/\*

\* CNF1 Register Values

\*/

#define SJW1 0x00

#define SJW2 0x40

#define SJW3 0x80

#define SJW4 0xC0

/\*

\* CNF2 Register Values

\*/

#define BTLMODE 0x80

#define SAMPLE\_1X 0x00

#define SAMPLE\_3X 0x40

/\*

\* CNF3 Register Values

\*/

#define SOF\_ENABLE 0x80

#define SOF\_DISABLE 0x00

#define WAKFIL\_ENABLE 0x40

#define WAKFIL\_DISABLE 0x00

/\*

\* CANINTF Register Bits

\*/

#define MCP\_RX0IF 0x01

#define MCP\_RX1IF 0x02

#define MCP\_TX0IF 0x04

#define MCP\_TX1IF 0x08

#define MCP\_TX2IF 0x10

#define MCP\_ERRIF 0x20

#define MCP\_WAKIF 0x40

#define MCP\_MERRF 0x80

/\*

\* speed 16M

\*/

#define MCP\_16MHz\_1000kBPS\_CFG1 (0x00)

#define MCP\_16MHz\_1000kBPS\_CFG2 (0xD0)

#define MCP\_16MHz\_1000kBPS\_CFG3 (0x82)

#define MCP\_16MHz\_500kBPS\_CFG1 (0x00)

#define MCP\_16MHz\_500kBPS\_CFG2 (0xF0)

#define MCP\_16MHz\_500kBPS\_CFG3 (0x86)

#define MCP\_16MHz\_250kBPS\_CFG1 (0x41)

#define MCP\_16MHz\_250kBPS\_CFG2 (0xF1)

#define MCP\_16MHz\_250kBPS\_CFG3 (0x85)

#define MCP\_16MHz\_200kBPS\_CFG1 (0x01)

#define MCP\_16MHz\_200kBPS\_CFG2 (0xFA)

#define MCP\_16MHz\_200kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_125kBPS\_CFG1 (0x03)

#define MCP\_16MHz\_125kBPS\_CFG2 (0xF0)

#define MCP\_16MHz\_125kBPS\_CFG3 (0x86)

#define MCP\_16MHz\_100kBPS\_CFG1 (0x03)

#define MCP\_16MHz\_100kBPS\_CFG2 (0xFA)

#define MCP\_16MHz\_100kBPS\_CFG3 (0x87)

/\*

#define MCP\_16MHz\_100kBPS\_CFG1 (0x03)

#define MCP\_16MHz\_100kBPS\_CFG2 (0xBA)

#define MCP\_16MHz\_100kBPS\_CFG3 (0x07)

\*/

#define MCP\_16MHz\_95kBPS\_CFG1 (0x03)

#define MCP\_16MHz\_95kBPS\_CFG2 (0xAD)

#define MCP\_16MHz\_95kBPS\_CFG3 (0x07)

#define MCP\_16MHz\_83k3BPS\_CFG1 (0x03)

#define MCP\_16MHz\_83k3BPS\_CFG2 (0xBE)

#define MCP\_16MHz\_83k3BPS\_CFG3 (0x07)

#define MCP\_16MHz\_80kBPS\_CFG1 (0x03)

#define MCP\_16MHz\_80kBPS\_CFG2 (0xFF)

#define MCP\_16MHz\_80kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_50kBPS\_CFG1 (0x07)

#define MCP\_16MHz\_50kBPS\_CFG2 (0xFA)

#define MCP\_16MHz\_50kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_40kBPS\_CFG1 (0x07)

#define MCP\_16MHz\_40kBPS\_CFG2 (0xFF)

#define MCP\_16MHz\_40kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_33kBPS\_CFG1 (0x09)

#define MCP\_16MHz\_33kBPS\_CFG2 (0xBE)

#define MCP\_16MHz\_33kBPS\_CFG3 (0x07)

#define MCP\_16MHz\_31k25BPS\_CFG1 (0x0F)

#define MCP\_16MHz\_31k25BPS\_CFG2 (0xF1)

#define MCP\_16MHz\_31k25BPS\_CFG3 (0x85)

#define MCP\_16MHz\_20kBPS\_CFG1 (0x0F)

#define MCP\_16MHz\_20kBPS\_CFG2 (0xFF)

#define MCP\_16MHz\_20kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_10kBPS\_CFG1 (0x1F)

#define MCP\_16MHz\_10kBPS\_CFG2 (0xFF)

#define MCP\_16MHz\_10kBPS\_CFG3 (0x87)

#define MCP\_16MHz\_5kBPS\_CFG1 (0x3F)

#define MCP\_16MHz\_5kBPS\_CFG2 (0xFF)

#define MCP\_16MHz\_5kBPS\_CFG3 (0x87)

#define MCPDEBUG (0)

#define MCPDEBUG\_TXBUF (0)

#define MCP\_N\_TXBUFFERS (3)

#define MCP\_RXBUF\_0 (MCP\_RXB0SIDH)

#define MCP\_RXBUF\_1 (MCP\_RXB1SIDH)

#define MCP2515\_OK (0)

#define MCP2515\_FAIL (1)

#define MCP\_ALLTXBUSY (2)

#define CANDEBUG 1

#define CANUSELOOP 0

#define CANSENDTIMEOUT (200) /\* milliseconds \*/

/\*

\* initial value of gCANAutoProcess

\*/

#define CANAUTOPROCESS (1)

#define CANAUTOON (1)

#define CANAUTOOFF (0)

#define CAN\_STDID (0)

#define CAN\_EXTID (1)

#define CANDEFAULTIDENT (0x55CC)

#define CANDEFAULTIDENTEXT (CAN\_EXTID)

#define CAN\_5KBPS 1

#define CAN\_10KBPS 2

#define CAN\_20KBPS 3

#define CAN\_31K25BPS 4

#define CAN\_33KBPS 5

#define CAN\_40KBPS 6

#define CAN\_50KBPS 7

#define CAN\_80KBPS 8

#define CAN\_83K3BPS 9

#define CAN\_95KBPS 10

#define CAN\_100KBPS 11

#define CAN\_125KBPS 12

#define CAN\_200KBPS 13

#define CAN\_250KBPS 14

#define CAN\_500KBPS 15

#define CAN\_1000KBPS 16

#define CAN\_OK (0)

#define CAN\_FAILINIT (1)

#define CAN\_FAILTX (2)

#define CAN\_MSGAVAIL (3)

#define CAN\_NOMSG (4)

#define CAN\_CTRLERROR (5)

#define CAN\_GETTXBFTIMEOUT (6)

#define CAN\_SENDMSGTIMEOUT (7)

#define CAN\_FAIL (0xff)

#define CAN\_MAX\_CHAR\_IN\_MESSAGE (8)

#endif

mcp\_can.h

#include "sam.h"

#ifndef MCP\_CAN\_H\_

#define MCP\_CAN\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SIMPLE FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void mcp2515\_reset(void);

uint8\_t mcp2515\_readRegister(const uint8\_t address);

void mcp2515\_readRegisterS(const uint8\_t address, uint8\_t values[], const uint8\_t n);

void mcp2515\_setRegister(const uint8\_t address, const uint8\_t value);

void mcp2515\_setRegisterS(const uint8\_t address, const uint8\_t values[], const uint8\_t n);

void mcp2515\_modifyRegister(const uint8\_t address, const uint8\_t mask, const uint8\_t data);

uint8\_t mcp2515\_readStatus(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* COMPOUND FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INIT SEQUENCES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t mcp2515\_setCANCTRL\_Mode(const uint8\_t newmode);

uint8\_t mcp2515\_configRate(const uint8\_t canSpeed);

void mcp2515\_initCANBuffers(void);

uint8\_t mcp2515\_init(const uint8\_t canSpeed);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MANAGEMENT TOOLS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void mcp2515\_write\_id(const uint8\_t mcp\_addr, const uint8\_t ext, const uint16\_t id);

void mcp2515\_write\_mf(const uint8\_t mcp\_addr, const uint8\_t ext, const uint32\_t id );

void mcp2515\_read\_id(const uint8\_t mcp\_addr, uint8\_t\* ext, uint16\_t\* id);

void mcp2515\_write\_canMsg(const uint8\_t buffer\_sidh\_addr);

void mcp2515\_read\_canMsg(const uint8\_t buffer\_sidh\_addr);

void mcp2515\_start\_transmit(const uint8\_t mcp\_addr);

uint8\_t mcp2515\_getNextFreeTXBuf(uint8\_t \*txbuf\_n);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START MCP SLAVE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t init\_Mask(uint8\_t num, uint32\_t ulData);

uint8\_t init\_Filt(uint8\_t num, uint8\_t ext, uint32\_t ulData);

uint8\_t setMsg(uint16\_t id, uint8\_t ext, uint8\_t len, uint8\_t \*pData);

uint8\_t clearMsg(void);

uint8\_t sendMsg(void);

uint8\_t sendMsgBuf(uint16\_t id, uint8\_t ext, uint8\_t len, uint8\_t \*buf);

uint8\_t readMsg(void);

uint8\_t readMsgBuf(uint8\_t \*len, uint8\_t buf[]);

uint8\_t readMsgBufID(uint16\_t \*ID, uint8\_t \*len, uint8\_t buf[]);

uint8\_t checkReceive(void);

uint32\_t getCanId(void);

uint8\_t isRemoteRequest(void);

uint8\_t isExtendedFrame(void);

uint8\_t canBegin(uint8\_t slave, uint8\_t speedset);

void slaveSelect(uint8\_t slave);

#endif /\* MCP\_CAN\_H\_ \*/

mcp\_can.c

#include "mcp\_can\_dfs.h"

#include "mcp\_can.h"

#include "spi.h"

#include "sam.h"

uint8\_t m\_nExtFlg; /\* identifier xxxID \*/

/\* either extended (the 29 LSB) \*/

/\* or standard (the 11 LSB) \*/

uint16\_t m\_nID; /\* can id \*/

uint8\_t m\_nDlc; /\* data length: \*/

uint8\_t m\_nDta[MAX\_CHAR\_IN\_MESSAGE]; /\* data \*/

uint8\_t m\_nRtr; /\* rtr \*/

uint8\_t m\_nfilhit;

uint8\_t device;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SIMPLE FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void mcp2515\_reset(void) {

// Reset Instruction = 0xC0

spiSS(device);

spiSend(MCP\_RESET);

spiSR(device);

}

uint8\_t mcp2515\_readRegister(const uint8\_t address) {

uint8\_t ret;

spiSS(device);

spiSend(MCP\_READ);

spiSend(address);

ret = spiSend(0x00);

spiSR(device);

return ret;

}

void mcp2515\_readRegisterS(const uint8\_t address, uint8\_t values[], const uint8\_t n) {

uint8\_t i;

spiSS(device);

spiSend(MCP\_READ);

spiSend(address);

// mcp2515 has auto-increment of address-pointer

for (i=0; i<n && i<CAN\_MAX\_CHAR\_IN\_MESSAGE; i++) {

values[i] = spiSend(0x00);

}

spiSR(device);

}

void mcp2515\_setRegister(const uint8\_t address, const uint8\_t value) {

spiSS(device);

spiSend(MCP\_WRITE);

spiSend(address);

spiSend(value);

spiSR(device);

}

void mcp2515\_setRegisterS(const uint8\_t address, const uint8\_t values[], const uint8\_t n) {

uint8\_t i;

spiSS(device);

spiSend(MCP\_WRITE);

spiSend(address);

for (i=0; i<n; i++)

{

spiSend(values[i]);

}

spiSR(device);

}

void mcp2515\_modifyRegister(const uint8\_t address, const uint8\_t mask, const uint8\_t data) {

spiSS(device);

spiSend(MCP\_BITMOD);

spiSend(address);

spiSend(mask);

spiSend(data);

spiSR(device);

}

uint8\_t mcp2515\_readStatus(void) {

uint8\_t ret;

spiSS(device);

spiSend(MCP\_READ\_STATUS);

ret = spiSend(0x00);

spiSR(device);

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* COMPOUND FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INIT SEQUENCES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t mcp2515\_setCANCTRL\_Mode(const uint8\_t newmode) {

uint8\_t i;

mcp2515\_modifyRegister(MCP\_CANCTRL, MODE\_MASK, newmode);

i = mcp2515\_readRegister(MCP\_CANCTRL);

i &= MODE\_MASK;

if ( i == newmode )

{

return MCP2515\_OK;

}

return MCP2515\_FAIL;

}

uint8\_t mcp2515\_configRate(const uint8\_t canSpeed) {

uint8\_t set, cfg1, cfg2, cfg3;

set = 1;

switch (canSpeed)

{

case (CAN\_5KBPS):

cfg1 = MCP\_16MHz\_5kBPS\_CFG1;

cfg2 = MCP\_16MHz\_5kBPS\_CFG2;

cfg3 = MCP\_16MHz\_5kBPS\_CFG3;

break;

case (CAN\_10KBPS):

cfg1 = MCP\_16MHz\_10kBPS\_CFG1;

cfg2 = MCP\_16MHz\_10kBPS\_CFG2;

cfg3 = MCP\_16MHz\_10kBPS\_CFG3;

break;

case (CAN\_20KBPS):

cfg1 = MCP\_16MHz\_20kBPS\_CFG1;

cfg2 = MCP\_16MHz\_20kBPS\_CFG2;

cfg3 = MCP\_16MHz\_20kBPS\_CFG3;

break;

case (CAN\_31K25BPS):

cfg1 = MCP\_16MHz\_31k25BPS\_CFG1;

cfg2 = MCP\_16MHz\_31k25BPS\_CFG2;

cfg3 = MCP\_16MHz\_31k25BPS\_CFG3;

break;

case (CAN\_33KBPS):

cfg1 = MCP\_16MHz\_33kBPS\_CFG1;

cfg2 = MCP\_16MHz\_33kBPS\_CFG2;

cfg3 = MCP\_16MHz\_33kBPS\_CFG3;

break;

case (CAN\_40KBPS):

cfg1 = MCP\_16MHz\_40kBPS\_CFG1;

cfg2 = MCP\_16MHz\_40kBPS\_CFG2;

cfg3 = MCP\_16MHz\_40kBPS\_CFG3;

break;

case (CAN\_50KBPS):

cfg1 = MCP\_16MHz\_50kBPS\_CFG1;

cfg2 = MCP\_16MHz\_50kBPS\_CFG2;

cfg3 = MCP\_16MHz\_50kBPS\_CFG3;

break;

case (CAN\_80KBPS):

cfg1 = MCP\_16MHz\_80kBPS\_CFG1;

cfg2 = MCP\_16MHz\_80kBPS\_CFG2;

cfg3 = MCP\_16MHz\_80kBPS\_CFG3;

break;

case (CAN\_83K3BPS):

cfg1 = MCP\_16MHz\_83k3BPS\_CFG1;

cfg2 = MCP\_16MHz\_83k3BPS\_CFG2;

cfg3 = MCP\_16MHz\_83k3BPS\_CFG3;

break;

case (CAN\_95KBPS):

cfg1 = MCP\_16MHz\_95kBPS\_CFG1;

cfg2 = MCP\_16MHz\_95kBPS\_CFG2;

cfg3 = MCP\_16MHz\_95kBPS\_CFG3;

break;

case (CAN\_100KBPS): /\* 100KBPS \*/

cfg1 = MCP\_16MHz\_100kBPS\_CFG1;

cfg2 = MCP\_16MHz\_100kBPS\_CFG2;

cfg3 = MCP\_16MHz\_100kBPS\_CFG3;

break;

case (CAN\_125KBPS):

cfg1 = MCP\_16MHz\_125kBPS\_CFG1;

cfg2 = MCP\_16MHz\_125kBPS\_CFG2;

cfg3 = MCP\_16MHz\_125kBPS\_CFG3;

break;

case (CAN\_200KBPS):

cfg1 = MCP\_16MHz\_200kBPS\_CFG1;

cfg2 = MCP\_16MHz\_200kBPS\_CFG2;

cfg3 = MCP\_16MHz\_200kBPS\_CFG3;

break;

case (CAN\_250KBPS):

cfg1 = MCP\_16MHz\_250kBPS\_CFG1;

cfg2 = MCP\_16MHz\_250kBPS\_CFG2;

cfg3 = MCP\_16MHz\_250kBPS\_CFG3;

break;

case (CAN\_500KBPS):

cfg1 = MCP\_16MHz\_500kBPS\_CFG1;

cfg2 = MCP\_16MHz\_500kBPS\_CFG2;

cfg3 = MCP\_16MHz\_500kBPS\_CFG3;

break;

case (CAN\_1000KBPS):

cfg1 = MCP\_16MHz\_1000kBPS\_CFG1;

cfg2 = MCP\_16MHz\_1000kBPS\_CFG2;

cfg3 = MCP\_16MHz\_1000kBPS\_CFG3;

break;

default:

set = 0;

break;

}

if (set) {

mcp2515\_setRegister(MCP\_CNF1, cfg1);

mcp2515\_setRegister(MCP\_CNF2, cfg2);

mcp2515\_setRegister(MCP\_CNF3, cfg3);

return MCP2515\_OK;

}

else {

return MCP2515\_FAIL;

}

}

void mcp2515\_initCANBuffers(void) {

uint8\_t i, a1, a2, a3;

uint8\_t std = 0;

uint8\_t ext = 1;

uint32\_t ulMask = 0x00000000;

uint32\_t ulFilt = 0x00000000;

//uint32\_t ulMask = 0x07FF0000;

//uint32\_t ulFilt = 0x03F80000;

//uint32\_t ulMask = 0x07FF0000;

//uint32\_t ulFilt = 0x06170000;

//uint32\_t ulMask = 0x07FF0000;

//uint32\_t ulFilt = 0x04420000;

//uint32\_t ulMask = 0x07FF0000;

//uint32\_t ulFilt = 0x06100000;

//uint32\_t ulMask = 0x07FF0000;

//uint32\_t ulFilt = 0x03E00000;

mcp2515\_write\_mf(MCP\_RXM0SIDH, ext, ulMask); /\*Set both masks to 0 \*/

mcp2515\_write\_mf(MCP\_RXM1SIDH, ext, ulMask); /\*Mask register ignores ext bit \*/

/\* Set all filters to 0 \*/

mcp2515\_write\_mf(MCP\_RXF0SIDH, ext, ulFilt); /\* RXB0: extended \*/

mcp2515\_write\_mf(MCP\_RXF1SIDH, std, ulFilt); /\* RXB1: standard \*/

mcp2515\_write\_mf(MCP\_RXF2SIDH, ext, ulFilt); /\* RXB2: extended \*/

mcp2515\_write\_mf(MCP\_RXF3SIDH, std, ulFilt); /\* RXB3: standard \*/

mcp2515\_write\_mf(MCP\_RXF4SIDH, ext, ulFilt);

mcp2515\_write\_mf(MCP\_RXF5SIDH, std, ulFilt);

/\* Clear, deactivate the three \*/

/\* transmit buffers \*/

/\* TXBnCTRL -> TXBnD7 \*/

a1 = MCP\_TXB0CTRL;

a2 = MCP\_TXB1CTRL;

a3 = MCP\_TXB2CTRL;

for (i = 0; i < 14; i++) { /\* in-buffer loop \*/

mcp2515\_setRegister(a1, 0);

mcp2515\_setRegister(a2, 0);

mcp2515\_setRegister(a3, 0);

a1++;

a2++;

a3++;

}

mcp2515\_setRegister(MCP\_RXB0CTRL, 0);

mcp2515\_setRegister(MCP\_RXB1CTRL, 0);

}

uint8\_t mcp2515\_init(const uint8\_t canSpeed) { /\* mcp2515init \*/

uint8\_t res;

mcp2515\_reset();

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

res = mcp2515\_setCANCTRL\_Mode(MODE\_CONFIG);

if(res > 0)

{

for(int aux=0;aux<500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

/\* set boadrate \*/

if(mcp2515\_configRate(canSpeed))

{

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

if ( res == MCP2515\_OK ) {

/\* init canbuffers \*/

mcp2515\_initCANBuffers();

/\* interrupt mode \*/

mcp2515\_setRegister(MCP\_CANINTE, MCP\_RX0IF | MCP\_RX1IF);

/\* enable both receive-buffers \*/

/\* to receive messages \*/

/\* with std. and ext. identifiers \*/

/\* and enable rollover \*/

mcp2515\_modifyRegister(MCP\_RXB0CTRL, MCP\_RXB\_RX\_MASK | MCP\_RXB\_BUKT\_MASK, MCP\_RXB\_RX\_STDEXT | MCP\_RXB\_BUKT\_MASK);

mcp2515\_modifyRegister(MCP\_RXB1CTRL, MCP\_RXB\_RX\_MASK, MCP\_RXB\_RX\_STDEXT);

/\* enter normal mode \*/

res = mcp2515\_setCANCTRL\_Mode(MODE\_NORMAL);

if(res)

{

for(int aux=0;aux<1500;aux++)

// We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

}

return res;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MANAGMENT TOOLS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void mcp2515\_write\_id(const uint8\_t mcp\_addr, const uint8\_t ext, const uint16\_t id) {

uint16\_t canid;

uint8\_t tbufdata[4];

canid = (uint16\_t)(id & 0x0FFFF);

if (ext == 1) {

tbufdata[MCP\_EID0] = (uint8\_t) (canid & 0xFF);

tbufdata[MCP\_EID8] = (uint8\_t) (canid >> 8);

canid = (uint16\_t)(id >> 16);

tbufdata[MCP\_SIDL] = (uint8\_t) (canid & 0x03);

tbufdata[MCP\_SIDL] += (uint8\_t) ((canid & 0x1C) << 3);

tbufdata[MCP\_SIDL] |= MCP\_TXB\_EXIDE\_M;

tbufdata[MCP\_SIDH] = (uint8\_t) (canid >> 5);

}

else {

tbufdata[MCP\_SIDH] = (uint8\_t) (canid >> 3);

tbufdata[MCP\_SIDL] = (uint8\_t) ((canid & 0x07) << 5);

tbufdata[MCP\_EID0] = 0;

tbufdata[MCP\_EID8] = 0;

}

mcp2515\_setRegisterS(mcp\_addr, tbufdata, 4);

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

}

void mcp2515\_write\_mf( const uint8\_t mcp\_addr, const uint8\_t ext, const uint32\_t id )

{

uint16\_t canid;

uint8\_t tbufdata[4];

canid = (uint16\_t)(id & 0x0FFFF);

if ( ext == 1)

{

tbufdata[MCP\_EID0] = (uint8\_t) (canid & 0xFF);

tbufdata[MCP\_EID8] = (uint8\_t) (canid >> 8);

canid = (uint16\_t)(id >> 16);

tbufdata[MCP\_SIDL] = (uint8\_t) (canid & 0x03);

tbufdata[MCP\_SIDL] += (uint8\_t) ((canid & 0x1C) << 3);

tbufdata[MCP\_SIDL] |= MCP\_TXB\_EXIDE\_M;

tbufdata[MCP\_SIDH] = (uint8\_t) (canid >> 5 );

}

else

{

tbufdata[MCP\_EID0] = (uint8\_t) (canid & 0xFF);

tbufdata[MCP\_EID8] = (uint8\_t) (canid >> 8);

canid = (uint16\_t)(id >> 16);

tbufdata[MCP\_SIDL] = (uint8\_t) ((canid & 0x07) << 5);

tbufdata[MCP\_SIDH] = (uint8\_t) (canid >> 3 );

}

mcp2515\_setRegisterS( mcp\_addr, tbufdata, 4 );

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

}

void mcp2515\_read\_id(const uint8\_t mcp\_addr, uint8\_t\* ext, uint16\_t\* id) {

uint8\_t tbufdata[4];

\*ext = 0;

\*id = 0;

mcp2515\_readRegisterS(mcp\_addr, tbufdata, 4);

\*id = (tbufdata[MCP\_SIDH]<<3) + (tbufdata[MCP\_SIDL]>>5);

if ((tbufdata[MCP\_SIDL] & MCP\_TXB\_EXIDE\_M) == MCP\_TXB\_EXIDE\_M)

{

/\* extended id \*/

\*id = (\*id<<2) + (tbufdata[MCP\_SIDL] & 0x03);

\*id = (\*id<<8) + tbufdata[MCP\_EID8];

\*id = (\*id<<8) + tbufdata[MCP\_EID0];

\*ext = 1;

}

}

void mcp2515\_write\_canMsg(const uint8\_t buffer\_sidh\_addr) {

uint8\_t mcp\_addr;

mcp\_addr = buffer\_sidh\_addr;

mcp2515\_setRegisterS(mcp\_addr+5, m\_nDta, m\_nDlc); /\* write data bytes \*/

if ( m\_nRtr == 1) /\* if RTR set bit in byte \*/

{

m\_nDlc |= MCP\_RTR\_MASK;

}

mcp2515\_setRegister((mcp\_addr+4), m\_nDlc); /\* write the RTR and DLC \*/

mcp2515\_write\_id(mcp\_addr, m\_nExtFlg, m\_nID); /\* write CAN id \*/

}

void mcp2515\_read\_canMsg(const uint8\_t buffer\_sidh\_addr) { /\* read can msg \*/

uint8\_t mcp\_addr, ctrl;

mcp\_addr = buffer\_sidh\_addr;

mcp2515\_read\_id(mcp\_addr, &m\_nExtFlg, &m\_nID);

ctrl = mcp2515\_readRegister(mcp\_addr-1);

m\_nDlc = mcp2515\_readRegister(mcp\_addr+4);

if ((ctrl & 0x08)) {

m\_nRtr = 1;

}

else {

m\_nRtr = 0;

}

m\_nDlc &= MCP\_DLC\_MASK;

mcp2515\_readRegisterS(mcp\_addr+5, &(m\_nDta[0]), m\_nDlc);

}

void mcp2515\_start\_transmit(const uint8\_t mcp\_addr) { /\* start transmit \*/

mcp2515\_modifyRegister(mcp\_addr-1 , MCP\_TXB\_TXREQ\_M, MCP\_TXB\_TXREQ\_M);

}

uint8\_t mcp2515\_getNextFreeTXBuf(uint8\_t \*txbuf\_n) { /\* get Next free txbuf \*/

uint8\_t res, i, ctrlval;

uint8\_t ctrlregs[MCP\_N\_TXBUFFERS] = {MCP\_TXB0CTRL, MCP\_TXB1CTRL, MCP\_TXB2CTRL};

res = MCP\_ALLTXBUSY;

\*txbuf\_n = 0x00;

//

/\* check all 3 TX-Buffers \*/

for (i=0; i<MCP\_N\_TXBUFFERS; i++) {

ctrlval = mcp2515\_readRegister(ctrlregs[i]);

if ( (ctrlval & MCP\_TXB\_TXREQ\_M) == 0) {

\*txbuf\_n = ctrlregs[i]+1; /\* return SIDH-address of Buffe \*/

/\* r \*/

res = MCP2515\_OK;

return res; /\* ! function exit \*/

}

}

return res;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START MCP SLAVE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t init\_Mask(uint8\_t num, uint32\_t ulData) {

uint8\_t res = MCP2515\_OK;

uint8\_t ext =0;

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

res = mcp2515\_setCANCTRL\_Mode(MODE\_CONFIG);

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

if(res > 0){

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

if((ulData & 0x80000000) == 0x80000000)

ext = 1;

if (num == 0){

mcp2515\_write\_mf(MCP\_RXM0SIDH, ext, ulData);

}

else if(num == 1){

mcp2515\_write\_mf(MCP\_RXM1SIDH, ext, ulData);

}

else res = MCP2515\_FAIL;

res = mcp2515\_setCANCTRL\_Mode(MODE\_NORMAL);

if(res > 0){

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

return res;

}

uint8\_t init\_Filt(uint8\_t num, uint8\_t ext, uint32\_t ulData) {

uint8\_t res = MCP2515\_OK;

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

res = mcp2515\_setCANCTRL\_Mode(MODE\_CONFIG);

if(res > 0)

{

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

switch(num)

{

case 0:

mcp2515\_write\_mf(MCP\_RXF0SIDH, ext, ulData);

break;

case 1:

mcp2515\_write\_mf(MCP\_RXF1SIDH, ext, ulData);

break;

case 2:

mcp2515\_write\_mf(MCP\_RXF2SIDH, ext, ulData);

break;

case 3:

mcp2515\_write\_mf(MCP\_RXF3SIDH, ext, ulData);

break;

case 4:

mcp2515\_write\_mf(MCP\_RXF4SIDH, ext, ulData);

break;

case 5:

mcp2515\_write\_mf(MCP\_RXF5SIDH, ext, ulData);

break;

default:

res = MCP2515\_FAIL;

}

res = mcp2515\_setCANCTRL\_Mode(MODE\_NORMAL);

if(res > 0)

{

for(int aux=0;aux<1500;aux++) // We need to wait about X microsec to let it load

{

;//this works like a NOPE instruction

}

return res;

}

for(int aux=0;aux<1500;aux++) // We need to wait about X microseconds to let it load

{

;//this works like a NOPE instruction

}

return res;

}

uint8\_t setMsg(uint16\_t id, uint8\_t ext, uint8\_t len, uint8\_t \*pData) {

int i = 0;

m\_nExtFlg = ext;

m\_nID = id;

m\_nDlc = len;

for(i = 0; i<MAX\_CHAR\_IN\_MESSAGE; i++)

{

m\_nDta[i] = \*(pData+i);

}

return MCP2515\_OK;

}

uint8\_t clearMsg() {

m\_nID = 0;

m\_nDlc = 0;

m\_nExtFlg = 0;

m\_nRtr = 0;

m\_nfilhit = 0;

for(int i = 0; i<m\_nDlc; i++ )

m\_nDta[i] = 0x00;

return MCP2515\_OK;

}

uint8\_t sendMsg() {

uint8\_t res, res1, txbuf\_n;

uint16\_t uiTimeOut = 0;

do {

res = mcp2515\_getNextFreeTXBuf(&txbuf\_n); /\* info = addr. \*/

uiTimeOut++;

} while (res == MCP\_ALLTXBUSY && (uiTimeOut < TIMEOUTVALUE));

if(uiTimeOut == TIMEOUTVALUE)

{

return CAN\_GETTXBFTIMEOUT; /\* get tx buff time out \*/

}

uiTimeOut = 0;

mcp2515\_write\_canMsg(txbuf\_n);

mcp2515\_start\_transmit(txbuf\_n);

do

{

uiTimeOut++;

res1= mcp2515\_readRegister(txbuf\_n); /\* read send buff ctrl reg \*/

res1 = res1 & 0x08;

}while(res1 && (uiTimeOut < TIMEOUTVALUE));

if(uiTimeOut == TIMEOUTVALUE) /\* send msg timeout \*/

{

return CAN\_SENDMSGTIMEOUT;

}

return CAN\_OK;

}

uint8\_t sendMsgBuf(uint16\_t id, uint8\_t ext, uint8\_t len, uint8\_t \*buf) {

setMsg(id, ext, len, buf);

return sendMsg();

}

uint8\_t readMsg() {

uint8\_t stat, res;

stat = mcp2515\_readStatus();

if (stat & MCP\_STAT\_RX0IF) /\* Msg in Buffer 0 \*/

{

mcp2515\_read\_canMsg(MCP\_RXBUF\_0);

mcp2515\_modifyRegister(MCP\_CANINTF, MCP\_RX0IF, 0);

res = CAN\_OK;

}

else if (stat & MCP\_STAT\_RX1IF) /\* Msg in Buffer 1 \*/

{

mcp2515\_read\_canMsg(MCP\_RXBUF\_1);

mcp2515\_modifyRegister(MCP\_CANINTF, MCP\_RX1IF, 0);

res = CAN\_OK;

}

else

{

res = CAN\_NOMSG;

}

return res;

}

uint8\_t readMsgBuf(uint8\_t \*len, uint8\_t buf[]) {

uint8\_t rc;

rc = readMsg();

if (rc == CAN\_OK) {

\*len = m\_nDlc;

for(int i = 0; i<m\_nDlc; i++) {

buf[i] = m\_nDta[i];

}

} else {

//\*len = 0;

}

return rc;

}

uint8\_t readMsgBufID(uint16\_t \*ID, uint8\_t \*len, uint8\_t buf[]) {

uint8\_t rc;

rc = readMsg();

if (rc == CAN\_OK) {

\*len = m\_nDlc;

\*ID = m\_nID;

for(int i = 0; i<m\_nDlc && i < MAX\_CHAR\_IN\_MESSAGE; i++) {

buf[i] = m\_nDta[i];

}

} else {

//\*len = 0;

}

return rc;

}

uint8\_t checkReceive(void) {

uint8\_t res;

res = mcp2515\_readStatus(); /\* RXnIF in Bit 1 and 0 \*/

if (res & MCP\_STAT\_RXIF\_MASK)

{

return CAN\_MSGAVAIL;

}

else

{

return CAN\_NOMSG;

}

}

uint32\_t getCanId(void) {

return m\_nID;

}

uint8\_t isRemoteRequest(void) {

return m\_nRtr;

}

uint8\_t isExtendedFrame(void) {

return m\_nExtFlg;

}

uint8\_t canBegin(uint8\_t slave, uint8\_t speedset) {

uint8\_t res;

device = slave;

res = mcp2515\_init(speedset);

if (res == MCP2515\_OK)

return CAN\_OK;

else

return CAN\_FAILINIT;

}

void slaveSelect(uint8\_t slave) {

spiSend(0xFF);

device = slave;

}

# Code in GPU:

main.c

/\* Description: main app

\* Author: Sharif Nasser Kadamani

\* ID: A00820367

\* Semester I - TEA AD18

\*/

#include <stdio.h>

#include <unistd.h>

#include <time.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <string.h>

#include <linux/i2c-dev.h>

#include <errno.h>

#include <pthread.h>

#include "graph\_link.h"

#include "i2c\_datapool.h"

#define SUCCESS 0

struct datapool pool[POOLSIZE];

int seconds = 0, minutes = 0, hours = 0, state = 0;

void \*get\_time(void \*arg) {

while(1) {

seconds++;

state = !state;

if(seconds == 60) {

minutes++;

seconds = 0;

}

if (minutes == 60)

{

hours++;

minutes = 0;

}

if (hours == 24)

{

hours = 0;

}

delay(1000);

}

}

int main(int argc, char \*argv[]) {

int ret;

*pthread\_t* tid;

*pthread\_attr\_t* tattr;

/\* Create second thread for TFT display updating \*/

*pthread\_attr\_init*(&tattr);

*pthread\_attr\_setdetachstate*(&tattr, *PTHREAD\_CREATE\_DETACHED*);

ret = *pthread\_create*(&tid, &tattr, get\_time, NULL); //CAMBIAR FUNCION

if(ret != SUCCESS) {

*printf*("error: could not start thread for time update\n");

}

/\* Initialize link with graphs in channel \*/

ret = init\_graph\_link();

while(1) {

receive\_datapool();

if(ret == 1) {

update\_graph();

}

else {

*printf*("error: could not init graph linkage\n");

}

delay(100);

}

/\* Ends link with graphs \*/

end\_graph\_link();

return SUCCESS;

}

i2c\_datapool.h

#define SAF\_ID (0x32)

#define INF\_OT\_ID (0x64)

#define INF\_TIME\_ID (0x65)

#define INF\_GEN\_ID (0x63)

#define INF\_ODO\_ID (0x60)

#define INF\_TRIP\_ID (0x58)

#define ENG\_FUEL\_ID (0x22)

#define ENG\_SPEED\_INDEX get\_position\_datapool(ENG\_SPEED\_ID)

#define ENG\_RPM\_INDEX get\_position\_datapool(ENG\_RPM\_ID)

#define ENG\_OIL\_LEVEL\_INDEX get\_position\_datapool(ENG\_OIL\_LEVEL\_ID)

#define ENG\_OIL\_PRESSURE\_INDEX get\_position\_datapool(ENG\_OIL\_PRESSURE\_ID)

#define ENG\_TEMP\_INDEX get\_position\_datapool(ENG\_TEMP\_ID)

#define ENG\_GEAR\_POS\_INDEX get\_position\_datapool(ENG\_GEAR\_POS\_ID)

#define BD\_INDEX get\_position\_datapool(BD\_ID)

#define SAF\_INDEX get\_position\_datapool(SAF\_ID)

#define INF\_OT\_INDEX get\_position\_datapool(INF\_OT\_ID)

#define INF\_TIME\_INDEX get\_position\_datapool(INF\_TIME\_ID)

#define INF\_GEN\_INDEX get\_position\_datapool(INF\_GEN\_ID)

#define INF\_ODO\_INDEX get\_position\_datapool(INF\_ODO\_ID)

#define INF\_TRIP\_INDEX get\_position\_datapool(INF\_TRIP\_ID)

#define ENG\_FUEL\_INDEX get\_position\_datapool(ENG\_FUEL\_ID)

struct datapool{

char \*name;

int id;

int length;

int data[5];

};

int delay(long int milliseconds);

int get\_position\_datapool(int id);

void receive\_datapool();

#endif // I2C\_DATAPOOL\_H\_INCLUDED

i2c\_datapool.c

/\* Description: Code in charge of I2C communication for datapool actualization

\* Author: Sharif Nasser Kadamani

\* ID: A00820367

\* Semester I - TEA AD18

\*/

#include <stdio.h>

#include <unistd.h>

#include <time.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <string.h>

#include <linux/i2c-dev.h>

#include <errno.h>

#include "i2c\_datapool.h"

#define I2C\_ADDR 0x49

#define ID\_INDEX 0

extern struct datapool pool[POOLSIZE];

int delay(long int milliseconds) {

struct *timespec* t;

// seconds

t.*tv\_sec* = milliseconds / 1000;

// nanoseconds

t.*tv\_nsec* = (milliseconds % 1000) \* 1000000;

return *nanosleep*(&t, NULL);

}

int get\_position\_datapool(int id) {

for(int pos = 0; pos < POOLSIZE; pos++) {

if(pool[pos].id == id) {

return pos;

}

}

return -1; /\* Si no encuentra el ID \*/

}

void receive\_datapool() {

char buff\_i2c, \*filename\_i2c = "/dev/i2c-2";

int file\_i2c, i, j;

int index\_data = ID\_INDEX;

file\_i2c = open(filename\_i2c, *O\_RDWR*);

if(file\_i2c < 0) {

*printf*("error opening file: %s \n", *strerror*(*errno*));

}

else if(ioctl(file\_i2c, I2C\_SLAVE, I2C\_ADDR) < 0){

*printf*("ioctl error: %s \n", *strerror*(*errno*));

}

else {

for(i = 0; i < POOLSIZE; i++) {

buff\_i2c = index\_data;

*write*(file\_i2c, &buff\_i2c, 1);

*read*(file\_i2c, &buff\_i2c, 1);

pool[i].id = buff\_i2c;

for(j = 0; j < DATASIZE; j++) {

index\_data++;

buff\_i2c = index\_data;

*write*(file\_i2c, &buff\_i2c, 1);

*read*(file\_i2c, &buff\_i2c, 1);

pool[i].data[j]= buff\_i2c;

}

*printf*("ID: %02X | DATA: %02X | %02X | %02X | %02X | %02X\n", pool[i].id, pool[i].data[0], pool[i].data[1], pool[i].data[2], pool[i].data[3], pool[i].data[4]);

index\_data = ID\_INDEX;

}

*printf*("\n");

}

*close*(file\_i2c);

}

graph\_link.h

#ifndef \_GRAPH\_LINK\_H\_

#define \_GRAPH\_LINK\_H\_

#define TRUE 1

#define FALSE 0

#define SPEED\_LIMIT (200)

#define RPM\_LIMIT (10000)

#define FUEL\_LIMIT (100)

#define OIL\_LIMIT (60)

#define BATTERY\_LIMIT (17)

#define ENG\_TEMP\_MAX (95)

#define ENG\_TEMP\_MIN (85)

#define DOS16 (255)

#define CUATRO16 (65536)

#define SEIS16 (16777216)

int init\_graph\_link(void);

void update\_graph();

void end\_graph\_link(void);

#endif // \_GRAPH\_LINK\_H\_

graph\_link.c

/\* Description: app that links code with graph in channel cluster, updates graphs and ends link

\* Author: Sharif Nasser Kadamani

\* ID: A00820367

\* Semester I - TEA AD18

\*/

#include <stdio.h>

#include <stdint.h>

#include <greio.h>

#include <string.h>

#include <time.h>

#include "graph\_link.h"

#include "i2c\_datapool.h"

#include "ClusterIO\_events.h"

#define SUCCESS 0

extern struct datapool pool[POOLSIZE];

extern int seconds, minutes, hours, state;

gre\_io\_t \*send\_handle;

gre\_io\_serialized\_data\_t \*nbuffer = NULL;

cluster\_update\_event\_t event\_data;

int init\_graph\_link(void) {

/\* Connect to a channel to send messages \*/

send\_handle = gre\_io\_open("cluster", GRE\_IO\_TYPE\_WRONLY);

if(send\_handle == NULL) {

*printf*("can't open send handle ClusterIO.gapp\n");

return FALSE;

}

return TRUE;

}

void update\_graph() {

/\* printf("Graph link loop... press CTRL + C to end\n"); \*/

int ret = 0;

/\* Current value variables \*/

static *int8\_t* temp\_ext = 0;

static uint8\_t speed = 0;

static uint8\_t fuel = 0;

static uint8\_t oil = 0;

static uint8\_t battery = 0;

static uint8\_t gear\_position = 0;

static uint8\_t engine\_ind = 0;

static uint8\_t brake\_ind = 0;

static uint8\_t l\_arrow\_ind = 0;

static uint8\_t r\_arrow\_ind = 0;

static uint8\_t seatbelt\_ind = 0;

static uint8\_t highbeam\_ind = 0;

static uint8\_t abs\_ind = 0;

static uint8\_t swerve\_ind = 0;

static uint8\_t seatbelt\_state = 0;

static uint8\_t hour\_time = 0;

static uint8\_t minute\_time = 0;

static uint16\_t rpm = 0;

static *uint64\_t* trip = 0;

static *uint64\_t* odometer = 0;

/\* Update value variables \*/

*int8\_t* temp\_ext\_received = pool[INF\_OT\_INDEX].data[0];

uint8\_t speed\_received = pool[ENG\_SPEED\_INDEX].data[0];

uint8\_t fuel\_received = pool[ENG\_FUEL\_INDEX].data[0];

uint8\_t oil\_received = pool[ENG\_OIL\_PRESSURE\_INDEX].data[0];

uint8\_t battery\_received = pool[INF\_GEN\_INDEX].data[0];

uint8\_t gear\_position\_received = pool[ENG\_GEAR\_POS\_INDEX].data[0];

uint8\_t eng\_temp\_received = pool[ENG\_TEMP\_INDEX].data[0];

uint8\_t brake\_ind\_received = pool[BD\_INDEX].data[2];

uint8\_t hour\_time\_received = pool[INF\_TIME\_INDEX].data[0];

uint8\_t seatbelt\_state\_received = pool[SAF\_INDEX].data[2];

uint8\_t minute\_time\_received = pool[INF\_TIME\_INDEX].data[2];

uint8\_t arrow\_received = pool[BD\_INDEX].data[4];

uint8\_t swerve\_ind\_received = pool[INF\_GEN\_INDEX].data[3];

uint8\_t abs\_ind\_received = pool[INF\_GEN\_INDEX].data[4];

uint8\_t highbeam\_ind\_received = pool[INF\_GEN\_INDEX].data[2];

uint16\_t rpm\_received = pool[ENG\_RPM\_INDEX].data[0] + (pool[ENG\_RPM\_INDEX].data[1] \* 16 \* 16);

*uint64\_t* odometer\_received = pool[INF\_ODO\_INDEX].data[0] +

(pool[INF\_ODO\_INDEX].data[1] \* DOS16) +

(pool[INF\_ODO\_INDEX].data[2] \* CUATRO16) +

(pool[INF\_ODO\_INDEX].data[3] \* SEIS16);

*uint64\_t* tip\_received = pool[INF\_TRIP\_INDEX].data[0] +

(pool[INF\_TRIP\_INDEX].data[1] \* DOS16) +

(pool[INF\_TRIP\_INDEX].data[2] \* CUATRO16) +

(pool[INF\_TRIP\_INDEX].data[3] \* SEIS16);

*memset*(&event\_data, 0, sizeof(event\_data));

event\_data.speed = speed;

event\_data.rpm = rpm;

event\_data.fuel = fuel;

event\_data.l\_arrow\_ind = l\_arrow\_ind;

event\_data.r\_arrow\_ind = r\_arrow\_ind;

event\_data.hour\_time = hour\_time;

event\_data.minute\_time = minute\_time;

event\_data.oil = oil;

event\_data.temp\_ext = temp\_ext;

event\_data.gear\_position = gear\_position;

event\_data.engine\_ind = engine\_ind;

event\_data.brake\_ind = brake\_ind;

event\_data.seatbelt\_ind = seatbelt\_ind;

event\_data.swerve\_ind = swerve\_ind;

event\_data.highbeam\_ind = highbeam\_ind;

event\_data.abs\_ind = abs\_ind;

event\_data.seatbelt\_state = seatbelt\_state;

event\_data.odometer = odometer;

event\_data.trip = trip;

event\_data.battery = battery;

/\* Speed update \*/

if(speed\_received > speed && speed <= SPEED\_LIMIT) {

speed++;

}

if(speed\_received < speed) {

speed--;

}

/\* RPM update \*/

if(rpm\_received > rpm && rpm <= RPM\_LIMIT) {

rpm = rpm + 100;

}

if(rpm\_received < rpm) {

rpm = rpm - 100;

}

/\* Fuel update \*/

if(fuel\_received > fuel && fuel <= FUEL\_LIMIT) {

fuel++;

}

if(fuel\_received < fuel) {

fuel--;

}

/\* Fuel update \*/

if(oil\_received > oil && oil <= OIL\_LIMIT) {

oil++;

}

if(oil\_received < oil) {

oil--;

}

if(battery\_received > battery && battery <= BATTERY\_LIMIT) {

battery++;

}

if(battery\_received < battery) {

battery--;

}

/\* Indicator initialization \*/

engine\_ind = 0;

brake\_ind = 0;

seatbelt\_ind = 0;

swerve\_ind = 0;

abs\_ind = 0;

highbeam\_ind = 0;

/\* Indicator update \*/

if(eng\_temp\_received > ENG\_TEMP\_MAX) {

engine\_ind = 255;

}

if(brake\_ind\_received == 1) {

brake\_ind = 255;

}

if(seatbelt\_state\_received >= 1 && seatbelt\_state\_received <= 3) {

seatbelt\_ind = 255;

}

if(abs\_ind\_received == 1) {

abs\_ind = 255;

}

if(swerve\_ind\_received == 1) {

swerve\_ind = 255;

}

if(highbeam\_ind\_received == 1) {

highbeam\_ind = 255;

}

switch(arrow\_received) {

case 0:

l\_arrow\_ind = 0;

r\_arrow\_ind = 0;

break;

case 1:

l\_arrow\_ind = state \* 255;

r\_arrow\_ind = 0;

break;

case 2:

l\_arrow\_ind = 0;

r\_arrow\_ind = state \* 255;

break;

case 3:

l\_arrow\_ind = state \* 255;

r\_arrow\_ind = state \* 255;

break;

}

hour\_time = hour\_time\_received + hours;

minute\_time = minute\_time\_received + minutes;

if(hour\_time >= 24) {

hour\_time = hour\_time - 24;

}

if(minute\_time >= 60) {

minute\_time = minute\_time - 60;

}

odometer = odometer\_received + trip;

trip = tip\_received;

seatbelt\_state = seatbelt\_state\_received;

gear\_position = gear\_position\_received;

temp\_ext = temp\_ext\_received;

/\* Send a named event with an additional string payload \*/

nbuffer = gre\_io\_serialize(nbuffer, NULL,

CLUSTER\_UPDATE\_EVENT,

CLUSTER\_UPDATE\_FMT,

&event\_data,

sizeof(event\_data));

if (!nbuffer) {

*fprintf*(*stderr*, "can't serialized data to buffer, exiting\n");

}

else {

gre\_io\_send(send\_handle, nbuffer);

if (ret < 0) {

*fprintf*(*stderr*, "send failed, exiting\n");

}

}

}

void end\_graph\_link(void) {

/\* Release the buffer memory, close the send handle \*/

gre\_io\_free\_buffer(nbuffer);

gre\_io\_close(send\_handle);

}

ClusterIO\_events.h

#define CLUSTER\_UPDATE\_EVENT "cluster\_update"

#define CLUSTER\_UPDATE\_FMT "8u1 trip 8u1 odometer 2u1 rpm 1u1 fuel 1u1 battery 1u1 oil 1u1 speed 1u1 gear\_position 1s1 temp\_ext 1u1 l\_arrow\_ind 1u1 r\_arrow\_ind 1u1 swerve\_ind 1u1 brake\_ind 1u1 highbeam\_ind 1u1 seatbelt\_ind 1u1 abs\_ind 1u1 engine\_ind 1u1 hour\_time 1u1 minute\_time 1u1 seatbelt\_state"

typedef struct {

*uint64\_t* trip;

*uint64\_t* odometer;

uint16\_t rpm;

uint8\_t fuel;

uint8\_t battery;

uint8\_t oil;

uint8\_t speed;

uint8\_t gear\_position;

*int8\_t* temp\_ext;

uint8\_t l\_arrow\_ind;

uint8\_t r\_arrow\_ind;

uint8\_t swerve\_ind;

uint8\_t brake\_ind;

uint8\_t highbeam\_ind;

uint8\_t seatbelt\_ind;

uint8\_t abs\_ind;

uint8\_t engine\_ind;

uint8\_t hour\_time;

uint8\_t minute\_time;

uint8\_t seatbelt\_state;

} cluster\_update\_event\_t;