/\*

\*\*

\*\* Copyright 2008 Onyx Engineering

\*\* This file is licensed MIT

\*\* File: lua\_serial.c

\*\* Adds low level linux serial support to lua

\*\*

\*\*

\*\*

\*\*

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <termios.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <assert.h>

#include "cf\_controller\_rs485.h"

static void setBaudRate( int handle, int speed );

static void computeCheckSum( char \*checkSum, char \*message, int size);

static char tx\_buffer[4096];

static int handle = -1;

static int lua\_rsRawRead( lua\_State \*L )

{

char string[256];

int size;

assert( handle > 0);

memset( string,0,sizeof(string ));

size = read( handle, string, 255 );

lua\_pushinteger( L, size );

if( size > 0 )

{

lua\_pushlstring( L, string, size );

}

else

{

lua\_pushnil(L );

}

return 2;

}

static int lua\_rsWrite( lua\_State \*L )

{

char \*string;

size\_t size;

char checkSum[3];

assert(handle > 0 );

string = (char \*)lua\_tolstring(L, 1, &size);

computeCheckSum(checkSum,string,size);

strcpy(tx\_buffer,string);

strcat(tx\_buffer,checkSum);

write( handle, tx\_buffer, size+2);

return 0;

}

typedef enum

{

LOOKING\_FOR\_HEADER = 0,

BUILDING\_MESSAGE = 1,

LOOKING\_FOR\_CHECK\_SUM\_1 = 2,

LOOKING\_FOR\_CHECK\_SUM\_2 = 3,

}RS485\_RX\_STATES;

static unsigned int rd\_state = LOOKING\_FOR\_HEADER;

static unsigned int rd\_index;

static unsigned int checkSum;

static unsigned int checkSum\_m;

static unsigned int delimeter\_count;

static char unsigned rd\_messageBuffer[ 4096];

static int lua\_rsGetMessage ( lua\_State \*L )

{

int returnValue;

int loopFlag;

int size;

unsigned char temp[2];

returnValue = 0;

loopFlag = 1;

delimeter\_count = 0;

while( loopFlag != 0)

{

size = read( handle, &temp, 1 );

if( rd\_index >= sizeof(rd\_messageBuffer))

{

rd\_state = LOOKING\_FOR\_HEADER;

}

//printf(" data %d state %d del cnt %d \r\n",temp[0], rd\_state, delimeter\_count);

if( size > 0 )

{

switch( rd\_state )

{

case LOOKING\_FOR\_HEADER:

if( temp[0] == '{' )

{

checkSum = temp[0];

checkSum &= 0xffff;

rd\_index = 0;

delimeter\_count = 1;

rd\_state = BUILDING\_MESSAGE;

rd\_messageBuffer[rd\_index] = temp[0];

rd\_index +=1;

}

else

{

; // stay in looking for header state

}

break;

case BUILDING\_MESSAGE:

rd\_messageBuffer[ rd\_index ] = temp[0];

checkSum += temp[0];

checkSum &= 0xffff;

rd\_index += 1;

switch( temp[0])

{

case '{':

delimeter\_count += 1;

break;

case '}':

delimeter\_count -= 1;

break;

}

if( delimeter\_count == 0)

{

rd\_state = LOOKING\_FOR\_CHECK\_SUM\_1;

}

break;

case LOOKING\_FOR\_CHECK\_SUM\_1:

checkSum\_m = temp[0];

checkSum\_m &= 0xff;

rd\_state = LOOKING\_FOR\_CHECK\_SUM\_2;

break;

case LOOKING\_FOR\_CHECK\_SUM\_2:

checkSum\_m |= ((temp[0]<<8)&0xff00);

lua\_pushinteger(L,checkSum);

lua\_pushinteger(L,checkSum\_m);

lua\_pushlstring(L,(char\*) rd\_messageBuffer,(unsigned) rd\_index );

returnValue = 3;

rd\_state = LOOKING\_FOR\_HEADER;

goto exit;

break;

}

}

else

{

loopFlag = 0;

}

}

exit:

return returnValue;

}

static int lua\_rsOpen( lua\_State \*L )

{

char \*device;

int speed;

rd\_state = LOOKING\_FOR\_HEADER;

device = (char \*)lua\_tostring(L,1);

speed = lua\_tointeger( L, 2 );

handle = open(device, O\_RDWR | O\_NONBLOCK );

if( handle > 0 )

{

setBaudRate( handle , speed );

lua\_pushboolean(L,1);

}

else

{

lua\_pushboolean(L,0);

}

return 1;

}

static int lua\_rsClose( lua\_State \*L )

{

int handle;

rd\_state = LOOKING\_FOR\_HEADER;

close(handle);

handle = -1;

return 0;

}

static int lua\_rsHelp( lua\_State \*L )

{

printf("list of low level rs commands commands \r\n\r\n");

printf("rs485.open(device,speed) -- opens serial port \r\n");

printf(" --- where 0 = 9600 \n");

printf(" --- where 1 = 19200 \n");

printf(" --- where 2 = 38K \n");

printf(" --- where 3 = 56K \n");

printf(" --- where 4 = 115K \n");

printf("rs485.close() -- closes serial port \n");

printf("rs485.rawRead() \n");

printf(" -- returns size string \n");

printf("rs485.getMessage( ) \n");

printf("rs485.write(handle,string) \n");

printf(" -- builds and writes API packet \n");

printf("rs485.help() -- dumps out commands \n");

return 0;

}

static int lua\_description( lua\_State \*L )

{

lua\_pushstring(L,"RS485 Interface");

return 1;

}

static const struct luaL\_reg lua\_zigbee[] =

{

{"description", lua\_description},

{"open", lua\_rsOpen },

{"close", lua\_rsClose },

{"rawRead", lua\_rsRawRead },

{"write", lua\_rsWrite },

{"getMessage", lua\_rsGetMessage },

{"help", lua\_rsHelp },

{ NULL, NULL },

};

int lua\_cf\_controller\_rs485\_load( lua\_State \*L )

{

luaL\_openlib(L,"rs485",lua\_zigbee,0 );

return 1;

}

/\*

\*\*

\*\* Static Support Functions

\*\*

\*\*

\*/

static void setBaudRate( int handle, int speed )

{

struct termios reading;

tcgetattr( handle,&reading);

switch( speed)

{

case 0:

cfsetispeed( &reading, B9600);

cfsetospeed( &reading, B9600);

break;

case 1:

cfsetispeed( &reading, B19200);

cfsetospeed( &reading, B19200);

break;

case 2:

cfsetispeed( &reading, B38400);

cfsetospeed( &reading, B38400);

break;

case 3:

cfsetispeed( &reading, B57600);

cfsetospeed( &reading, B57600);

break;

case 4:

cfsetispeed( &reading, B115200);

cfsetospeed( &reading, B115200);

break;

}

reading.c\_cflag |= CREAD;

reading.c\_cc[VMIN]=1;

reading.c\_cc[VTIME]=0;

reading.c\_lflag &= ~ICANON; /\* unbuffered input \*/

reading.c\_iflag = IGNPAR;

reading.c\_oflag = 0;

tcsetattr( handle,TCSANOW,&reading);

}

static void computeCheckSum( char \*checkSum, char \*message, int size)

{

int temp;

int i;

temp = 0;

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

{

temp += \*message;

temp &= 0xffff;

message++;

}

\*checkSum++ = temp & 0xff;

\*checkSum++ = (temp>> 8)& 0xff;

}