#include <Arduino.h>

#include <Wire.h>

#include <SPI.h>

#include "DataFlash.h"

#include "telitGPRS.h"

#include "clock.h"

#include "cache.h"

#include "sensors.h"

// EEPROM Layout of 4096 bytes - atmega128l

// "Cache" (record buffer ) used from byte 100 to 4055

// The devices ID registered in the databse is stored at the start, and consists of 2 fields (isSet - uint8, deviceID - uint32)

// The struct takes byte 0 - 4 inclusive.

// NB: This to change since using the 4mb DataFlash IC

// CoCT IOT Telemetry defines

#define MSG\_TYPE\_AUTH 0x0

#define MSG\_TYPE\_TIME 0x1

#define MSG\_TYPE\_DATA 0xA

#define MSG\_TYPE\_PING 0xF

#define COCT\_SITE\_ID 0x10

const byte i2cPin = PIN\_PG1; // - enable i2c communications with the tiny13A

const byte softPowerPin = PIN\_PD7; // HIGH = MCU is powered else LOW MCU switches off.

static const int csPin = 8; // Pins for the Data Flash chip

static const int resetPin = 8;

static const int wpPin = 7;

DataFlash dataflash;

clock clk;

cache che(100, 66, 16, 247);

telitGPRS gprs(Serial1);

sensors inputs;

uint16\_t msPause, msCount;

uint8\_t buffer[16];

uint32\_t msBufferSent;

uint8\_t cntSends;

// uint32\_t tsCheckClock;

uint32\_t tsActivity;

uint32\_t tsSensorActivity;

void debug(const \_\_FlashStringHelper \*string)

{

gprs.unlatchSerial();

Serial1.print(string);

gprs.latchSerial();

}

void debug(const char\* fmt, ...)

{

int count;

va\_list args;

byte b;

va\_start(args, fmt);

count = strlen(fmt);

gprs.unlatchSerial();

for (int i = 0; i < count; i++)

{

if (fmt[i] == '%')

{

if (fmt[i+1] == '%')

{

Serial1.write(fmt[i++]);

} else {

switch(fmt[i+1])

{

case 'd': Serial1.print(va\_arg(args, int32\_t));

break;

case 'i': Serial1.print(va\_arg(args, int32\_t));

break;

case 'l': Serial1.print(va\_arg(args, uint32\_t));

break;

case 'w': Serial1.print(va\_arg(args, uint16\_t)); // use this for byte values, as the print doesn't work with byte.

break;

case 'c': Serial1.print((char)va\_arg(args, int16\_t));

break;

case 's': Serial1.print(va\_arg(args, char\*));

break;

case 'h': b = (uint8\_t) va\_arg(args, uint16\_t);

if (b < 16) Serial1.write(48);

Serial1.print(b, HEX);

break;

default: ;

}

i++;

}

} else

Serial1.write(fmt[i]);

}

gprs.latchSerial();

va\_end(args);

}

void bufferSensorValues()

{

uint8\_t readBuffer[16];

uint32\_t temp = clk.get();

uint32\_t values = 0;

memset(&readBuffer[0], 0, sizeof(readBuffer));

memcpy(&readBuffer[0], &temp, 4);

values = inputs.getValue(0) & 0x3FF;

temp = inputs.getValue(1) & 0x3FF;

values += (temp << 10);

temp = inputs.getValue(2) & 0x3FF;

values += (temp << 20);

memcpy(&readBuffer[4], &values, 4);

values = inputs.getValue(5);

temp = inputs.getValue(6);

// values += (temp << 16);

memcpy(&readBuffer[8], &values, 4);

memcpy(&readBuffer[12], &temp, 4);

tsSensorActivity = clk.get();

if (!che.add(&readBuffer[0], 16)) debug(F("FAILED to cache reading :-("));

}

int calcCRC(int crc, uint8\_t \* ptr, int count)

{

char i;

while (--count >= 0)

{

crc = crc ^ (int) \*ptr++ << 8;

i = 8;

do

{

if (crc & 0x8000)

crc = crc << 1 ^ 0x1021;

else

crc = crc << 1;

} while(--i);

}

return crc;

}

bool sendPacket(const uint8\_t type, const bool ack, const uint32\_t id, uint8\_t\* data, const uint16\_t len)

{

bool result = false;

uint16\_t header;

uint16\_t crc;

uint16\_t packetLen;

int32\_t tmp;

memset(&header, 0, sizeof(header));

memset(&crc, 0, sizeof(crc));

packetLen = sizeof(header) + sizeof(id) + len + sizeof(crc);

header = packetLen + (0x00 << 9) + ((type & 0xF) << 11);

if (ack) header += (0x1 << 15);

tmp = 0;

tmp = calcCRC(tmp, (uint8\_t \*) &header, sizeof(header));

tmp = calcCRC(tmp, (uint8\_t \*) &id, sizeof(id));

if (len > 0) tmp = calcCRC(tmp, (uint8\_t \*) data, len);

crc = tmp & 0xFFFF;

result = gprs.sendOpen(packetLen) &&

gprs.sendData((uint8\_t \*) &header, sizeof(header)) &&

gprs.sendData((uint8\_t \*) &id, sizeof(id)) &&

((len == 0) || gprs.sendData(data, len)) &&

gprs.sendData((uint8\_t \*) &crc, sizeof(crc)) &&

gprs.sendClose();

if (!result) gprs.closeUdpConnection();

if (result)

{

cntSends++;

tsActivity = clk.get();

}

return result;

}

bool sendPacketTime()

{

uint16\_t\* milliseconds;

uint32\_t\* seconds;

uint32\_t\* marker;

uint8\_t buf[10];

marker = (uint32\_t\*) &buf[0];

seconds = (uint32\_t\*) &buf[4];

milliseconds = (uint16\_t\*) &buf[8];

\*milliseconds = 0;

\*marker = clk.get(seconds);

return sendPacket(MSG\_TYPE\_TIME, false, COCT\_SITE\_ID, buf, 10);

}

bool handleIncomingDataFromServer()

{

uint8\_t \*data;

uint16\_t length;

uint32\_t \*temp;

uint32\_t \*seconds;

uint16\_t \*milliseconds;

int crc = 0;

uint8\_t version = 0;

uint8\_t type = 0;

bool ack = false;

uint32\_t id = 0;

uint32\_t test;

// uint32\_t difference = 0;

if (gprs.recv())

{

cntSends = 0;

gprs.data(&data, &length);

temp = (uint32\_t \*) data;

if ((\*temp & 0x1FF) == length)

{

crc = calcCRC(crc, data, length - 2);

temp = (uint32\_t \*) (data + length - 2);

if ((crc & 0xFFFF) == (\*temp & 0xFFFF))

{

// Length and CRC passed !

memcpy(&id, data+2, sizeof(id));

temp = (uint32\_t \*) (data+1);

version = (\*temp >> 1) & 0x3;

type = (\*temp >> 3) & 0xF;

ack = (((\*temp >> 7) & 0x1) == 1);

data += 6;

if (version == 0)

{

// Everything should be ready now to simply check response and handle data

if (ack)

{

// Responses to data sent are handled here

switch (type)

{

case MSG\_TYPE\_TIME:

seconds = (uint32\_t \*) data;

milliseconds = (uint16\_t \*) (data+4);

temp = (uint32\_t \*) (data+6);

test = clk.set(\*seconds, \*milliseconds, \*temp);

//temp = (uint32\_t \*) data;

//difference = clk.get();

//difference = (difference > \*temp) ? difference - \*temp : \*temp - difference;

debug("Remote clock is %l.%w, with marker = %l... %l\r\n", \*seconds, \*milliseconds, \*temp, test);

//if (!clk.isSet() || (difference > 5)) clk.set(\*temp - 1);

//tsCheckClock = clk.get() + 360;

break;

case MSG\_TYPE\_DATA:

if (msBufferSent > 0)

{

msBufferSent = 0;

// Delete the oldest record from the cache

che.shift();

}

break;

case MSG\_TYPE\_PING:

break;

default:

debug("Received ACK for unknown message type (%h)\r\n", type);

}

} else {

// Requests from the server are handled here

debug("No logic to handle request (%h) from server.\r\n", type);

}

}

tsActivity = clk.get();

}

}

}

return true;

}

void onReceiveI2C(int numBytes)

{

uint8\_t id;

uint32\_t pulseInterval = 0;

uint16\_t pulseCount = 0;

uint32\_t value;

uint8\_t count = 0;

while (Wire.available())

{

value = (uint8\_t) Wire.read();

if (count == 0)

{

id = value;

} else if (count < 5) {

pulseInterval += (value << (8 \* (count - 1)));

} else pulseCount += (value << (8 \* (count - 5)));

count++;

}

debug("I2C: id=%w, ms=%l, cnt=%w.\r\n", id, pulseInterval, pulseCount);

if ((count == 7) && (id < 3)) inputs.setPulse(id, pulseInterval / pulseCount);

// if (count == 3) inputs.setPulse(id, pulseInterval & 0xFFFF);

}

void setup()

{

uint8\_t status;

DataFlash::ID id;

uint16\_t counter = 0;

pinMode(i2cPin, OUTPUT);

pinMode(softPowerPin, OUTPUT);

digitalWrite(i2cPin, HIGH);

digitalWrite(softPowerPin, HIGH);

Serial1.begin(57600);

Wire.onReceive(onReceiveI2C);

Wire.begin(3);

msPause = 0;

msCount = 0;

memset(&buffer[0], 0, sizeof(buffer));

msBufferSent = 0;

cntSends = 0;

tsSensorActivity = 0;

tsActivity = 0;

// DataFlash configuration

SPI.begin();

dataflash.setup(csPin);

dataflash.begin();

status = dataflash.status();

dataflash.readID(id);

debug("DataFlash: status(%h) manufacturer(%h) d0(%h) d1(%h) ext(%h)", status, id.manufacturer, id.device[0], id.device[1], id.extendedInfoLength);

}

void loop()

{

msPause = 0;

// get the GSM engine up and running !

if (!gprs.isOnline())

{

if (!gprs.isEnabled() && !gprs.enable()) msPause = 5000;

if ((msPause == 0) && gprs.isRegistered() && gprs.isGprsRegistered())

{

// configure and connect here

if (gprs.isActive())

{

if (!gprs.openUdpConnection())

{

gprs.deactivate();

msPause = 5000;

debug(F("Gprs active but connection failed"));

}

} else {

if (!(gprs.signalStrength() && gprs.configure() && gprs.activate() && gprs.openUdpConnection()))

{

msPause = 5000;

// Retry 3 times then disable the modem and start again

debug(F("First gprs connection failed"));

}

}

} else msPause = 1000;

} else {

msPause = 5000;

// First process any data back from the server

handleIncomingDataFromServer();

// if (!clk.isSet() || ((tsCheckClock > 0) && (clk.get() > tsCheckClock)))

if (clk.checkClockNow())

{

// First things first, need to set the clock as soon as the device comes online

if (sendPacketTime()) msPause = 10000;

} else {

// Business as normal

if ((msBufferSent == 0) && (che.size() > 0) && che.next(&buffer[0], sizeof(buffer))) msBufferSent = millis() - 10000;

if ((msBufferSent != 0) && (millis() - msBufferSent > 9999))

{

if (sendPacket(MSG\_TYPE\_DATA, false, COCT\_SITE\_ID, buffer, sizeof(buffer)))

{

msPause = 10000;

msBufferSent = millis();

}

} else if (clk.get() - tsActivity > 60) {

if (sendPacket(MSG\_TYPE\_PING, false, COCT\_SITE\_ID, NULL, 0)) msPause = 10000;

}

}

}

if (msPause == 0) msPause = 100;

msCount = 0;

if (gprs.hasEventFired\_NOCARRIER()) gprs.closeUdpConnection();

if (gprs.hasEventFired\_SRING()) msPause = 0;

while (msCount < msPause)

{

if (gprs.isReadPending()) break;

if (((msCount % 1000) == 0) && clk.isSet() && inputs.read() && (inputs.percentageChangeOf(1) || (clk.get() - tsSensorActivity > 3600)))

{

//debug("Got new values for inputs");

bufferSensorValues();

inputs.rollValues();

break;

}

msCount += 100;

delay(100);

}

}