/\*\* =========================================================================

\* @file DRWI\_SIM7080LTE.ino

\* @brief Example for DRWI CitSci LTE sites.

\*

\* This example shows proper settings for the following configuration:

\*

\* Mayfly v1.0 board

\* EnviroDIY SIM7080 LTE module (with Hologram SIM card)

\* Hydros21 CTD sensor

\* Campbell Scientific OBS3+ Turbidity sensor

\*

\* @author Sara Geleskie Damiano <sdamiano@stroudcenter.org>

\* @copyright (c) 2017-2022 Stroud Water Research Center (SWRC)

\* and the EnviroDIY Development Team

\* This example is published under the BSD-3 license.

\*

\* Hardware Platform: EnviroDIY Mayfly Arduino Datalogger

\*

\* DISCLAIMER:

\* THIS CODE IS PROVIDED "AS IS" - NO WARRANTY IS GIVEN.

\* ======================================================================= \*/

// ==========================================================================

// Defines for TinyGSM

// ==========================================================================

/\*\* Start [defines] \*/

#ifndef TINY\_GSM\_RX\_BUFFER

#define TINY\_GSM\_RX\_BUFFER 64

#endif

#ifndef TINY\_GSM\_YIELD\_MS

#define TINY\_GSM\_YIELD\_MS 2

#endif

/\*\* End [defines] \*/

// ==========================================================================

// Include the libraries required for any data logger

// ==========================================================================

/\*\* Start [includes] \*/

// The Arduino library is needed for every Arduino program.

#include <Arduino.h>

// EnableInterrupt is used by ModularSensors for external and pin change

// interrupts and must be explicitly included in the main program.

#include <EnableInterrupt.h>

// Include the main header for ModularSensors

#include <ModularSensors.h>

#include <AM2315C.h>

/\*\* End [includes] \*/

// ==========================================================================

// Data Logging Options

// ==========================================================================

/\*\* Start [logging\_options] \*/

// The name of this program file

const char\* sketchName = "GHWX1.cpp";

// Logger ID, also becomes the prefix for the name of the data file on SD card

const char \*LoggerID = "GHX1";

// How frequently (in minutes) to log data

const uint8\_t loggingInterval = 15;

// Your logger's timezone.

const int8\_t timeZone = -6; // UTC-6 hours - CST - not sprung forward for daylight savings

// NOTE: Daylight savings time will not be applied! Please use standard time!

// Set the input and output pins for the logger

// NOTE: Use -1 for pins that do not apply

const int32\_t serialBaud = 57600; // Baud rate for debugging

const int8\_t greenLED = 8; // Pin for the green LED

const int8\_t redLED = 9; // Pin for the red LED

const int8\_t buttonPin = 21; // Pin for debugging mode (ie, button pin)

const int8\_t wakePin = 31; // MCU interrupt/alarm pin to wake from sleep

// Mayfly 0.x D31 = A7

const int8\_t sdCardPwrPin = -1; // MCU SD card power pin

const int8\_t sdCardSSPin = 12; // SD card chip select/slave select pin

const int8\_t sensorPowerPin = 22; // MCU pin controlling main sensor power

/\*\* End [logging\_options] \*/

// ==========================================================================

// Wifi/Cellular Modem Options

// ==========================================================================

/\*\* Start [sim\_com\_sim7080] \*/

// For almost anything based on the SIMCom SIM7080G

#include <modems/SIMComSIM7080.h>

// Create a reference to the serial port for the modem

HardwareSerial& modemSerial = Serial1; // Use hardware serial if possible

const int32\_t modemBaud = 9600; // SIM7080 does auto-bauding by default, but

// for simplicity we set to 9600

// Modem Pins - Describe the physical pin connection of your modem to your board

// NOTE: Use -1 for pins that do not apply

const int8\_t modemVccPin = 18;

// MCU pin controlling modem power --- Pin 18 is the power enable pin for the

// bee socket on Mayfly v1.0, use -1 if using Mayfly 0.5b or if the bee socket

// is constantly powered (ie you changed SJ18 on Mayfly 1.x to 3.3v)

const int8\_t modemStatusPin = 19; // MCU pin used to read modem status

const int8\_t modemSleepRqPin = 23; // MCU pin for modem sleep/wake request

const int8\_t modemLEDPin = redLED; // MCU pin connected an LED to show modem

// status

// Network connection information

const char\* apn =

"hologram"; // APN connection name, typically Hologram unless you have a

// different provider's SIM card. Change as needed

// Create the modem object

SIMComSIM7080 modem7080(&modemSerial, modemVccPin, modemStatusPin,

modemSleepRqPin, apn);

// Create an extra reference to the modem by a generic name

SIMComSIM7080 modem = modem7080;

/\*\* End [sim\_com\_sim7080] \*/

// ==========================================================================

// Using the Processor as a Sensor

// ==========================================================================

/\*\* Start [processor\_sensor] \*/

#include <sensors/ProcessorStats.h>

// Create the main processor chip "sensor" - for general metadata

const char\* mcuBoardVersion = "v1.1";

ProcessorStats mcuBoard(mcuBoardVersion);

/\*\* End [processor\_sensor] \*/

// ==========================================================================

// Maxim DS3231 RTC (Real Time Clock)

// ==========================================================================

/\*\* Start [ds3231] \*/

#include <sensors/MaximDS3231.h>

// Create a DS3231 sensor object

MaximDS3231 ds3231(1);

/\*\* End [ds3231] \*/

// ==========================================================================

// Sensirion SHT4X Digital Humidity and Temperature Sensor

// Built in on Mayfly 1.x

// ==========================================================================

/\*\* Start [sensirion\_sht4x] \*/

#include <sensors/SensirionSHT4x.h>

// NOTE: Use -1 for any pins that don't apply or aren't being used.

const int8\_t SHT4xPower = sensorPowerPin; // Power pin

const bool SHT4xUseHeater = true;

// Create an Sensirion SHT4X sensor object

SensirionSHT4x sht4x(SHT4xPower, SHT4xUseHeater);

// Create humidity and temperature variable pointers for the SHT4X

Variable\* sht4xHumid =

new SensirionSHT4x\_Humidity(&sht4x, "478413da-8244-4d34-855e-7c74b390e7ad");

Variable\* sht4xTemp =

new SensirionSHT4x\_Temp(&sht4x, "65d7964a-5bfd-4afd-a7d4-f1f677993ff1");

/\*\* End [sensirion\_sht4x] \*/

// ==========================================================================

// AOSong AM2315(c) - BJ Digital Humidity and Temperature Sensor

// ==========================================================================

/\*\* Start [ao\_song\_am2315] \*/

#include "AM2315c.h"

#include "sensors/AOSongAM2315.h"

#include <Wire.h>

// NOTE: Use -1 for any pins that don't apply or aren't being used.

const int8\_t AM2315Power = sensorPowerPin; // Power pin

// Create an AOSong AM2315 sensor object

TwoWire\* myI2C = &Wire;

int8\_t powerPin = AM2315Power;

uint8\_t measurementsToAverage = 3; // or any other value you want

AOSongAM2315 am2315(myI2C, powerPin, measurementsToAverage);

// Create humidity and temperature variable pointers for the AM2315

Variable\* am2315Humid =

new AOSongAM2315\_Humidity(&am2315, "0af7a334-7200-4ade-853d-09c66afb6f58");

Variable\* am2315Temp =

new AOSongAM2315\_Temp(&am2315, "5c5f2d60-497c-4c34-b6fb-6c074a56b200");

/\*\* End [ao\_song\_am2315] \*/

// ==========================================================================

// Creating the Variable Array[s] and Filling with Variable Objects

// ==========================================================================

/\*\* Start [variable\_arrays] \*/

Variable \*variableList[] = {

new ProcessorStats\_Battery(&mcuBoard), // Battery voltage (ProcessorStats\_Battery)

new SensirionSHT4x\_Humidity(&sht4x), // Relative humidity (Sensirion\_SHT40\_Humidity)

new Modem\_SignalPercent(&modem), // Percent full scale (EnviroDIY\_LTEB\_SignalPercent)

new SensirionSHT4x\_Temp(&sht4x), // Temperature (Sensirion\_SHT40\_Temperature)

new AOSongAM2315\_Temp(&am2315), // Temperature (AOSong\_AM2315\_Temp)

new AOSongAM2315\_Humidity(&am2315) // Relative humidity (AOSong\_AM2315\_Humidity)};

};

// All UUID's, device registration, and sampling feature information can be

// pasted directly from Monitor My Watershed.

// To get the list, click the "View token UUID list" button on the upper right

// of the site page.

// \*\*\* CAUTION --- CAUTION --- CAUTION --- CAUTION --- CAUTION \*\*\*

// Check the order of your variables in the variable list!!!

// Be VERY certain that they match the order of your UUID's!

// Rearrange the variables in the variable list ABOVE if necessary to match!

// Do not change the order of the variables in the section below.

// \*\*\* CAUTION --- CAUTION --- CAUTION --- CAUTION --- CAUTION \*\*\*

// Replace all of the text in the following section with the UUID array from

// MonitorMyWatershed

/\* clang-format off \*/

// --------------------- Beginning of Token UUID List ---------------------

const char \*UUIDs[] = // UUID array for device sensors

{

"16c3d920-5060-4831-af32-20a6160636f4", // Battery voltage (EnviroDIY\_Mayfly\_Batt)

"84852a10-2872-4f01-94d4-4e25c60cfc6a", // Relative humidity (Sensirion\_SHT40\_Humidity)

"9f140560-6783-4127-b459-cf2f5191ddd6", // Percent full scale (EnviroDIY\_LTEB\_SignalPercent)

"7b1df8a7-9115-462f-a802-85b7b04b8c21", // Temperature (Sensirion\_SHT40\_Temperature)

"5c5f2d60-497c-4c34-b6fb-6c074a56b200", // Temperature (AOSong\_AM2315\_Temp)

"0af7a334-7200-4ade-853d-09c66afb6f58" // Relative humidity (AOSong\_AM2315\_Humidity)

};

const char \*registrationToken = "84c74f11-11cc-438d-9824-845926153594"; // Device registration token

const char \*samplingFeature = "565522f2-c1a9-4edb-87c9-b1d400a11753"; // Sampling feature UUID

// ----------------------- End of Token UUID List -----------------------

/\* clang-format on \*/

// Count up the number of pointers in the array

int variableCount = sizeof(variableList) / sizeof(variableList[0]);

// Create the VariableArray object

VariableArray varArray(variableCount, variableList, UUIDs);

/\*\* End [variable\_arrays] \*/

// ==========================================================================

// The Logger Object[s]

// ==========================================================================

/\*\* Start [loggers] \*/

// Create a new logger instance

Logger dataLogger(LoggerID, loggingInterval, &varArray);

/\*\* End [loggers] \*/

// ==========================================================================

// Creating Data Publisher[s]

// ==========================================================================

/\*\* Start [publishers] \*/

// Create a data publisher for the Monitor My Watershed/EnviroDIY POST endpoint

#include <publishers/EnviroDIYPublisher.h>

EnviroDIYPublisher EnviroDIYPOST(dataLogger, &modem.gsmClient,

registrationToken, samplingFeature);

/\*\* End [publishers] \*/

// ==========================================================================

// Working Functions

// ==========================================================================

/\*\* Start [working\_functions] \*/

// Flashes the LED's on the primary board

void greenredflash(uint8\_t numFlash = 4, uint8\_t rate = 75) {

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

digitalWrite(greenLED, HIGH);

digitalWrite(redLED, LOW);

delay(rate);

digitalWrite(greenLED, LOW);

digitalWrite(redLED, HIGH);

delay(rate);

}

digitalWrite(redLED, LOW);

}

// Reads the battery voltage

// NOTE: This will actually return the battery level from the previous update!

float getBatteryVoltage() {

if (mcuBoard.sensorValues[0] == -9999) mcuBoard.update();

return mcuBoard.sensorValues[0];

}

// ==========================================================================

// Arduino Setup Function

// ==========================================================================

/\*\* Start [setup] \*/

void setup() {

// Start the primary serial connection

Serial.begin(serialBaud);

// Print a start-up note to the first serial port

Serial.print(F("Now running "));

Serial.print(sketchName);

Serial.print(F(" on Logger "));

Serial.println(LoggerID);

Serial.println();

Serial.print(F("Using ModularSensors Library version "));

Serial.println(MODULAR\_SENSORS\_VERSION);

Serial.print(F("TinyGSM Library version "));

Serial.println(TINYGSM\_VERSION);

Serial.println();

// Initialize the AM2315 sensor

if (!am2315.begin()) {

Serial.println("AM2315 sensor initialization failed!");

} else {

Serial.println("AM2315 sensor initialized successfully!");

}

// Start the serial connection with the modem

modemSerial.begin(modemBaud);

// Set up pins for the LED's

pinMode(greenLED, OUTPUT);

digitalWrite(greenLED, LOW);

pinMode(redLED, OUTPUT);

digitalWrite(redLED, LOW);

// Blink the LEDs to show the board is on and starting up

greenredflash();

pinMode(20, OUTPUT); // for proper operation of the onboard flash memory

// chip's ChipSelect (Mayfly v1.0 and later)

// Set the timezones for the logger/data and the RTC

// Logging in the given time zone

Logger::setLoggerTimeZone(timeZone);

// It is STRONGLY RECOMMENDED that you set the RTC to be in UTC (UTC+0)

Logger::setRTCTimeZone(0);

// Attach the modem and information pins to the logger

dataLogger.attachModem(modem);

modem.setModemLED(modemLEDPin);

dataLogger.setLoggerPins(wakePin, sdCardSSPin, sdCardPwrPin, buttonPin,

greenLED);

// Begin the logger

dataLogger.begin();

// Note: Please change these battery voltages to match your battery

// Set up the sensors, except at lowest battery level

if (getBatteryVoltage() > 3.4) {

Serial.println(F("Setting up sensors..."));

varArray.setupSensors();

}

/\*\* Start [setup\_sim7080] \*/

modem.setModemWakeLevel(HIGH); // ModuleFun Bee inverts the signal

modem.setModemResetLevel(HIGH); // ModuleFun Bee inverts the signal

Serial.println(F("Waking modem and setting Cellular Carrier Options..."));

modem.modemWake(); // NOTE: This will also set up the modem

modem.gsmModem.setBaud(modemBaud); // Make sure we're \*NOT\* auto-bauding!

modem.gsmModem.setNetworkMode(38); // set to LTE only

// 2 Automatic

// 13 GSM only

// 38 LTE only

// 51 GSM and LTE only

modem.gsmModem.setPreferredMode(1); // set to CAT-M

// 1 CAT-M

// 2 NB-IoT

// 3 CAT-M and NB-IoT

/\*\* End [setup\_sim7080] \*/

// Sync the clock if it isn't valid or we have battery to spare

if (getBatteryVoltage() > 3.55 || !dataLogger.isRTCSane()) {

// Synchronize the RTC with NIST

// This will also set up the modem

dataLogger.syncRTC();

}

// Create the log file, adding the default header to it

// Do this last so we have the best chance of getting the time correct and

// all sensor names correct

// Writing to the SD card can be power intensive, so if we're skipping

// the sensor setup we'll skip this too.

if (getBatteryVoltage() > 3.4) {

Serial.println(F("Setting up file on SD card"));

dataLogger.turnOnSDcard(

true); // true = wait for card to settle after power up

dataLogger.createLogFile(true); // true = write a new header

dataLogger.turnOffSDcard(

true); // true = wait for internal housekeeping after write

}

// Call the processor sleep

Serial.println(F("Putting processor to sleep\n"));

dataLogger.systemSleep();

}

/\*\* End [setup] \*/

// ==========================================================================

// Arduino Loop Function

// ==========================================================================

/\*\* Start [loop] \*/

// Use this short loop for simple data logging and sending

void loop() {

// Note: Please change these battery voltages to match your battery

// At very low battery, just go back to sleep

if (getBatteryVoltage() < 3.4) {

dataLogger.systemSleep();

}

// At moderate voltage, log data but don't send it over the modem

else if (getBatteryVoltage() < 3.55) {

dataLogger.logData();

}

// If the battery is good, send the data to the world

else {

dataLogger.logDataAndPublish();

}

}

/\*\* End [loop] \*/