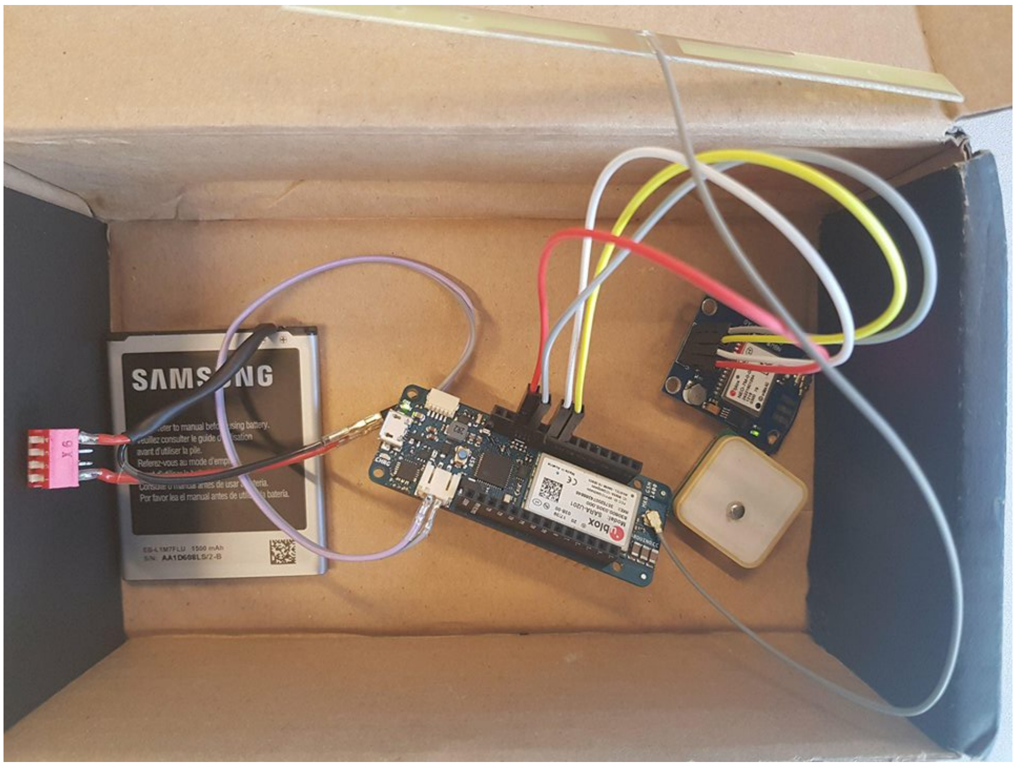
GPS Tracker

F18-ITAMS



Casper Boye Pedersen
201504480

Lasse Østerberg Sangild
20114274

Advisor
Henning Hargaard

Abstract

The focus of this project has been to investigate a possible way of building a portable location tracking device, capable of sending it's location to a remote server. The device had, at a minimum, to be able to locate itself and send the location to a remote server.

The chosen modules that make up the GPS Tracker are a NEO-7M GPS module, an Arduino MKR GSM 1400 combined micro controller and GSM module and a Ciseco B026 SPI to micro SD socket. The SD card socket has not been implemented.

The modules has been successfully tested on their own and as part of a system context. The GPS Tracker is, at the end of development, able to locate itself and send the location data to a remote server.

Contents

Chapter 1
Introduction

As different Global Navigation Satellite Systems (GNSS) are developed (GPS, GLONASS, GALILEO and BeiDou) and low power devices (IoT) blooms, it seems obvious to combine them. The combination can lead to different quality of life improvements, such as coffee being ready when you get home, health care personnel being alerted if a person leaves the premises, the ability to find something you have lost and so forth.

1.1 Project Description

The scope of this project will be to create a low power battery powered unit able to locate itself anywhere in the world and send the acquired data to a server for storage and/or analysis.

1.2 Terminology

Below in table 1.1 is shown a list of terms used throughout the report describing each name for clarification purposes.

Name	**Description**
µC	Micro controller
µSD	Micro SD

Table 1.1: List of terminologies.

Chapter 2

Requirements

In this chapter the requirements for the system will be described. No use cases have been produced as the target of this project is rather simple. Instead the MoSCoW below will be used to describe the functions of the system and the prioritization.

- **Must**
  - Communicate with a server over wast distances.
  - Retrieve global positional data.
  - Send location data to server.
- **Should**
  - Save location data to non-volatile memory.
  - Employ power saving functions when not being used.
- **Could**
  - Allow for wireless updates.
- **Won't**
  - Log movement data.

2.1 Non-functional Requirements

- Should be able to store at least 1440 locations (one data point per minute for 24 hours).

Chapter 3

Analysis

To fulfil the specified requirements in Requirements a module to locate the system, save the location and send the location must be present. Additionally a controller is needed to facilitate communication between the modules. This system interfaces with a server as shown in figure 3.1

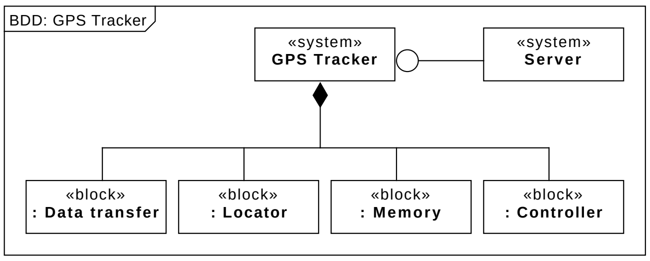


Figure 3.1: A high level BDD showing the parts of the GPS Tracker and the interface with a server.

3.1 Controller

The controller handles the interfacing between the other blocks. It allows them to go into low power modes when not used, and wakes them up when necessary. It should also, according to the requirements, be a low power device. These specifications can be handled by a µC. This fits very well with this project being part of a course on µC.

3.2 Data Transfer

The Data Transfer block is responsible for sending the data stored on the GPS Tracker to a remote server for backup or analysis. This can be done in a number of ways, most plausible are transmission over wifi, mobile networks or satellite.

Wifi is very fast, but is also very limited in range and availability. As this module is meant to be taken ”on-the-move” the wifi technology does not offer the range required by the requirements section.

Mobile networks be it GPRS, 2G, 3G or 4G, all offer long range communications available almost anywhere on the planet. Modules are readily available, and the only requirement is the use of a SIM card, of which the acquisition and use is rather cheap. As the data is merely GPS data the amount of data sent could be low, and the speed of the chosen network should not be a focus.

Satellite Internet access is available anywhere on the planet. It covers the globe using satellites, and the signal is only hindered by the weather. The latency is high, 638 ms [5], but this should not be an issue in a system as this. The drawbacks are that the equipment needed is rather large, and that the cost is 300 € [5].

LoRaWAN is a technology primarily developed to cater for the many IoT devices, which are developed at the time. It offers low speeds but high ranges with few masts. The issue with LoRaWAN is that it primarily covers urban areas, and coverage of out of town areas is minimal at the moment.

3.3 Locator

There are a few location services available, but one stands out.

GSM triangulation uses available GSM masts, their location and their signal strengths to calculate and approximate location of the device. This solution could possibly be paired with the Data Transfer module, allowing for fewer modules to be incorporated into the GPS Tracker. The issue with GSM triangulation is that the location accuracy drastically drops when GSM towers are not available or the reception settings are poor.

GNSS is the go to solution for low power devices requiring knowledge of their location. It is available world-wide and is free of charge, whether you choose the American GPS, Russian GLONASS or European GALILEO. The accuracy is often less than 10 m.

3.4 Memory

Storing memory is a vital function of the GPS Tracker if the Data Transfer module is unable to establish a connection to the server, and to maximize power savings. Storing data allows for only waking up the Locator module often, and save the data for transmission in larger packages.

EEPROM grants access to a small volume of non-volatile data storage. It might even be included on the chosen controller, allowing the Memory module to be internal. An issue might be the storage volume, as the requirements specify at least 1440 data points.

USB Flash Drives are readily available and the size of them is can be very small. The available storage can be massive and is often in excess of 1 GB. In order for a µM to interact with a USB drive, it will most likely need to use an external USB controller, which would be a secondary module needed.

SD cards are very much like USB Flash Drives in that they can also store large amounts of data. SPI is a free to use interaction method with SD cards, which most µC can use.

Chapter 4

Architecture

This section describes the overall system architecture, the chosen platforms for development and how the different subsystems communicate with each other.

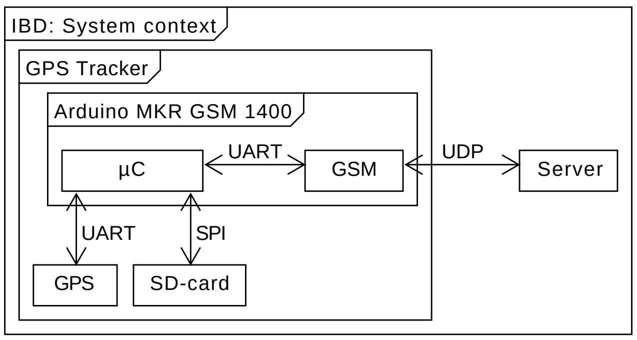


Figure 4.1: Overall IBD, including types of connection.

4.1 Choice of components

The physical entities necessary to solve the tasks of the blocks identified in figure 4.1 are as follows:

Controller and Data transfer can be performed by a single module. The Arduino MKR GSM 1400 which includes a SAMD21G18A ARM µC and a u-blox SARA-U201-03B-00 High Speed Packet Access (HSPA) with 2G fallback module for internet access[1]. They communicate using one of the two UARTs available on the Arduino MKR GSM 1400 board. The SAMD21G18A ARM µC uses $<$15 mA while running [3, p. 791-794]. The SARA-U201 uses a maximum of 1.9 A when transferring data [7, p. 26] and less than 1 mA when in power-saving mode. Alternatively the known Arduino MEGA2560 with an external GSM module could be used, but this has been opted against in favour of using an integrated low power module.

Locator has to be able to locate itself globally. This is solved using the GNSS module, u-blox NEO-7M-0-000, from Embedded Stock. The module is able to use GPS (with SBAS and QZSS) and GLONASS, allowing for global positioning. It also uses a maximum of 67 mA [6, p. 17] and is the low power version from the NEO-7 series. The breakout-board exposes the UART connection, which can be used to communicate with the controller.

Memory has to be non-volatile and able to store at least 1440 locations. This is accomplished using a SD-card with 2 GB of memory, allowing for each location to be up to 1.39 MB. The connection to the Ciseco B026 SD-card breakout board[4] is SPI which is directly available on the Arduino MKR GSM 1400 module.

4.2 Communication

With the chosen components, the interfaces between them can be identified, as shown in figure 4.1. The communication between the SAMD21G18A and the SARA-U201 module are presoldered to UART. The breakout-board for the NEO-7M exposes UART ports and the SD-card breakout-board exposes SPI ports. Connection to the server is chosen to be UDP with an acknowledge from the server.

4.2.1 SAMD21G18A and SARA-U201

As is standard with GSM modules, the SARA-U201 employs a AT command interface through the UART. This means that commands and data to and from the module is in ascii, and that commands is started by "AT+" and end with <CR><LF>.

The implementation and design of communications between SAMD21G18A and SARA-U201 will follow specification laid out in the u-blox AT Commands Manual [9].

In general each communication will follow a given flow, the controller starts by sending an AT command:

```
AT+CREG?\r\n
```

Responding to each message the GSM module responds with the command it read:

```
AT+CREG?\r
```

Following command confirmation, if it is a recognised command, the GSM will potentially send a respond message with status. This responds always starts with a '+' symbol.

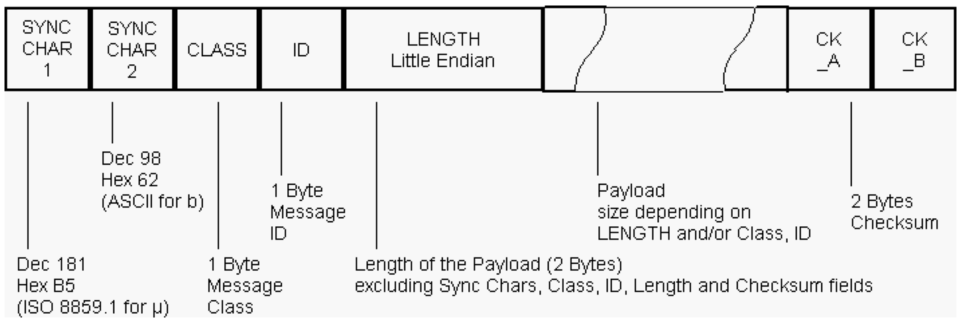

SYNC CHAR 1
SYNC CHAR 2
CLASS
ID
LENGTH Little Endian
CK _A
CK _B
Dec 98 Hex 62 (ASCII for b)
1 Byte Message ID
Payload size depending on LENGTH and/or Class, ID
2 Bytes Checksum
Dec 181 Hex B5 (ISO 8859.1 for µ)
1 Byte Message Class
Length of the Payload (2 Bytes) excluding Sync Chars, Class, ID, Length and Checksum fields

Figure 4.2: Overview of the UBX protocol [8, p. 73].

```
AT+CREG: 0,1\r
```

Finally a last message will be send, with the result code of the command. This will either be ”OK” or ”ERROR”. If the command was not recognised the result will always be ”ERROR”.

```
OK\r\n
```

4.2.2 Arduino MKR GSM 1400 and NEO-7M

The NEO-7M UART uses the frame 8N1, with a baud rate of 4800, 9600, 19 200, 38 400, 57 600 or 115 200. For easy debugging, the 9600 baud rate is chosen.

4.2.2.1 Protocol

The chosen GPS module is able to communicate using NMEA, UBX or RTCM protocol [8]. It has been chosen to communicate using the UBX protocol, as it, asides from the GPS location data, can be used to set up the modules internal workings, such as power save mode.

Figure 4.2 shows the UBX protocol. The sync chars are static and the CLASS and ID define the function which the NEO-7M is meant to perform. The checksum is calculated using the CLASS, ID, length and payload bytes, as described in [8, p. 74]. Two types of responses from the NEO-7M. If the command is a setup command, the NEO-7M answers with ACK or NACK [8, p. 80]. Any other command will respond with the answer described in the protocol. See [8, p. 73-183] for the full protocol, or below for an example.

The most used UBX command is UBX-NAV-PVT [8, p. 160], which returns the location of the NEO-7M. The communication is shown in table 4.1. Answers are available from the later performed tests, but can be seen in appendix B and appendix C.

	Command	**Description**
Sync Char 1	0xB5	Hex value for the ascii char µ.
Sync Char 2	0x62	Hex value for the ascii char b.
CLASS	0x01	Navigation Results.
ID	0x07	Navigation Position Velocity Time Solution.
Length	0x0000	Length of payload.
Payload	NONE	Here no payload is needed, as the CLASS and ID tells the NEO-7M what to return.
CK_A	0x08	Checksum part A.
CK_B	0x19	Cheksum part B.

Table 4.1: Example communication using the UBX protocol.

Chapter 5
Design

5.1 Sequence Diagrams

The sequence diagrams, figures 5.1 to 5.3 below, are meant to give an understanding of the inner workings of the GPS Tracker.

Initiation of the system is described in figure 5.1. It shows the µC initiating itself and then the external blocks. The GPS module is polled until it returns valid location data. The time returned from the GPS allows the µC to know the time and run the desired actions either at certain intervals, e.g. every 10 minutes, or at specific times, e.g. every full hour. When valid data has been retrieved it is saved to the SD card and the GPS and GSM modules are put to sleep to conserve power. The acquisition of GPS location is shown in figure 5.2. The GPS starts in a sleep mode and is awoken to find it's location. When the location is found and is valid, the GPS is put back to sleep and the data is saved to the SD card. To upload data to the server, the GSM must be awoken and data must be retrieved from the SD card, as shown in figure 5.3. When the data has been sent the GSM is put back to sleep.

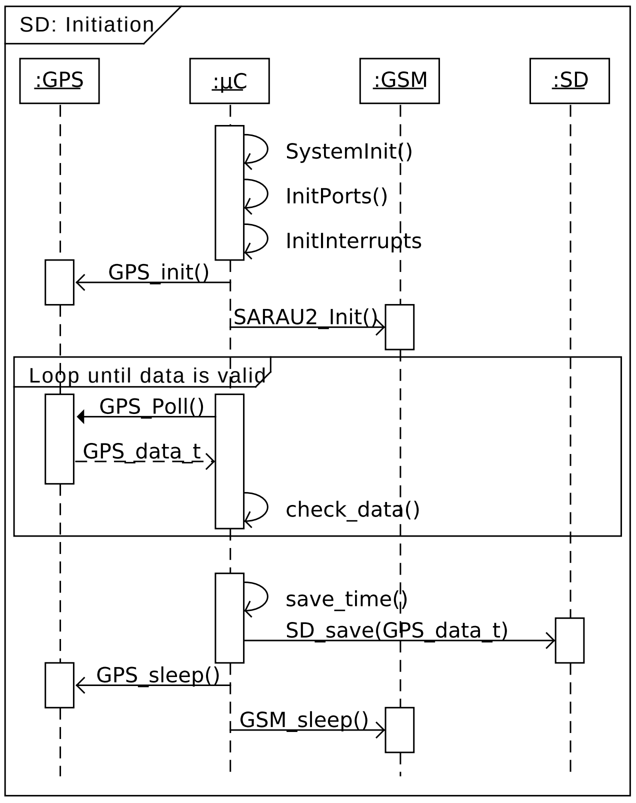


Figure 5.1: Sequence diagram showing the initiation of the system.

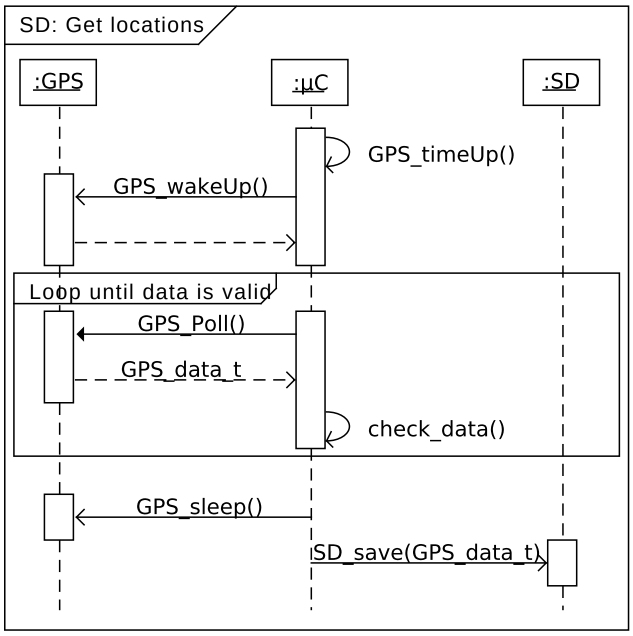


Figure 5.2: Sequence diagram describing the acquisition and saving of GPS data.

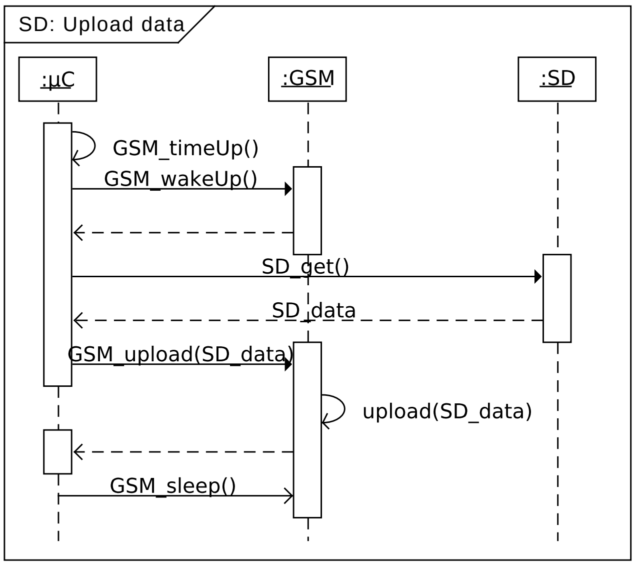


Figure 5.3: Sequence diagram showing the process of sending saved data to the server.

5.2 Controller - SAMD21G18A

General flow of the system is linear, where the modules are initialized, and the GPS polling is looped. Since the communication protocols are asynchronous, it was decided to make the serial ports interrupt based. Meaning that the main thread will not prevent data from being sent or received, while also making the send and receive functions non blocking.

5.2.1 UART - BaudRate

Both SARA-U201 and NEO-7M supports the usage of auto baud rate detection, meaning that they will respond to a message in the same baud rate as they receive. But due to consistency, both modules is set to the selected baud rate during their set up configuration. For simplicity a baud rate of 9600 bit/s was chosen, specification for the UART is shown in table 5.1.

Baud Rate	9600
Data Bits	8
Parity	None
Stop Bits	1

Table 5.1: Baud rate definition used for serial com ports on SAMD21G18A.

5.3 GSM - SARA-U201

To facilitate the functionality of the SARA-U201 module, a modem specific driver will be developed. This driver will contain functions to configure SARA-U201 for sending and receiving data through UDP packets.

5.3.1 UART

Communication with the SARA-U201 module, will be done through a UART connection. The module contains an ability to auto detect the baud rate of the first transmission, and use this baud rate to respond. In addition it supports the standard baud rates: 1200, 2400, 4800, 9600, 19 200, 38 400 and 5 higher rates. 9600 baud was selected with focus on stability and the ability to debug the communications.

5.3.2 AT Command Interface

Through the application a limited amount of commands is necessary, the command flow is displayed on figure 5.4. While functionality of the commands used are described, in appendix A.

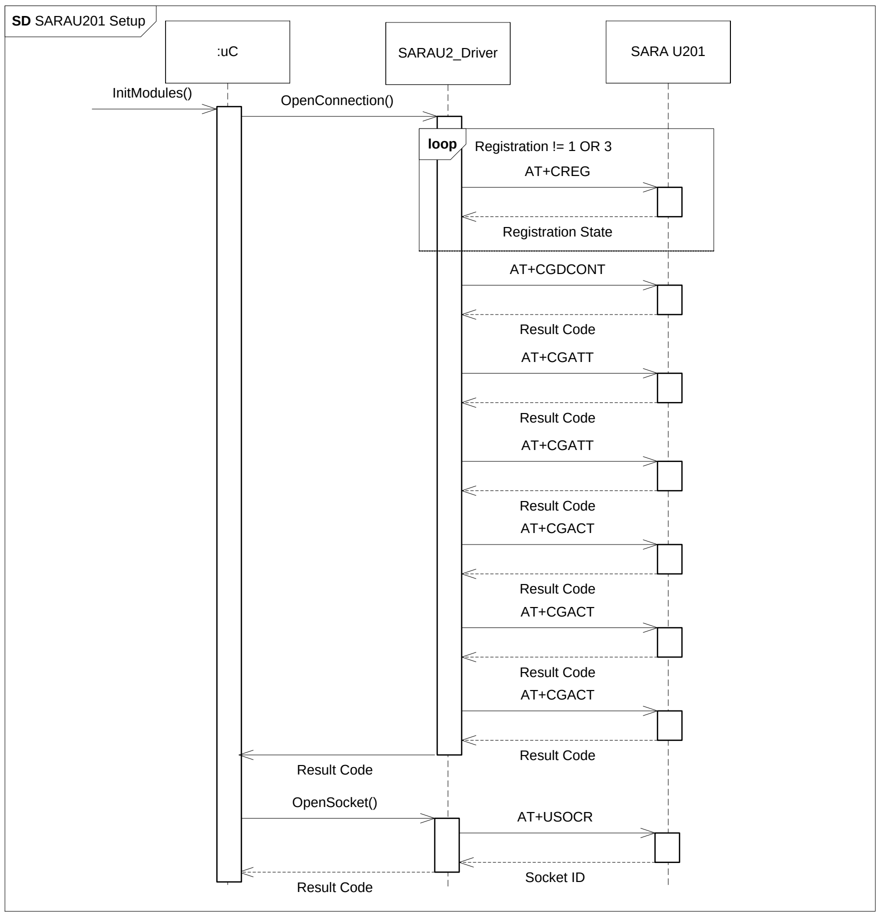


Figure 5.4: Sequence diagram of AT configuration of SARA-U201 module.

Noteworthy in figure 5.4 is, the calls between SARAU2 Driver and SARA U201 should be seen as UART transmissions, and not direct method calls. The transmission is facilitated by the an UART driver.

5.4 GPS - NEO-7M

The design of is shown as a flow diagrams in figures 5.5 and 5.6. After an initiation is run for the GPS module, it only returns data when something is sent to it.

As seen in figure 5.5, the first two steps construct the message according to the protocol described in section 4.2.2.1. Then the message is sent, and the response header is located in the UART buffer, as shown in figure 5.6, where every unused byte before the header is removed with the `popByte()` function.

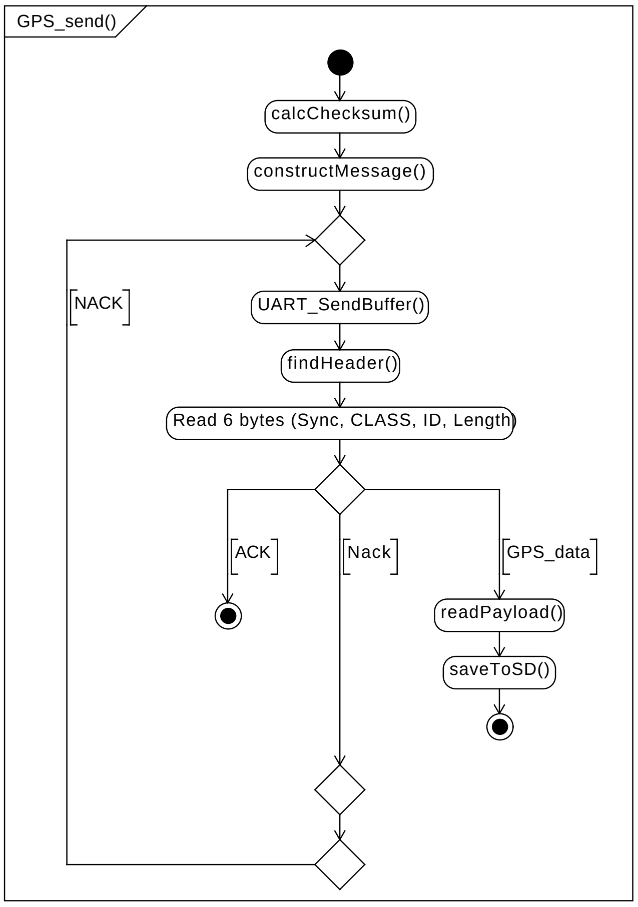


Figure 5.5: Flow diagram for `GPS_send()`.

When the received data has been located, the first six bytes, including the length of the payload are loaded, and the type of message is determined. If the response is an ACK, the program can continue, but if it is a NACK, the message has to be sent again. If the response contains GPS_data, the data must be read, interpreted and saved to the SD-card.

The GPS module is put to sleep whenever a valid fix has been received, to reduce the power consumption, and it is allowed because the direction and speed of the GPS module is not at interest in this project. Sleeping the module requires GLONASS to be turned of. Additionally the module is put into pedestrian mode, as this is the mode in which the GPS will be tested.

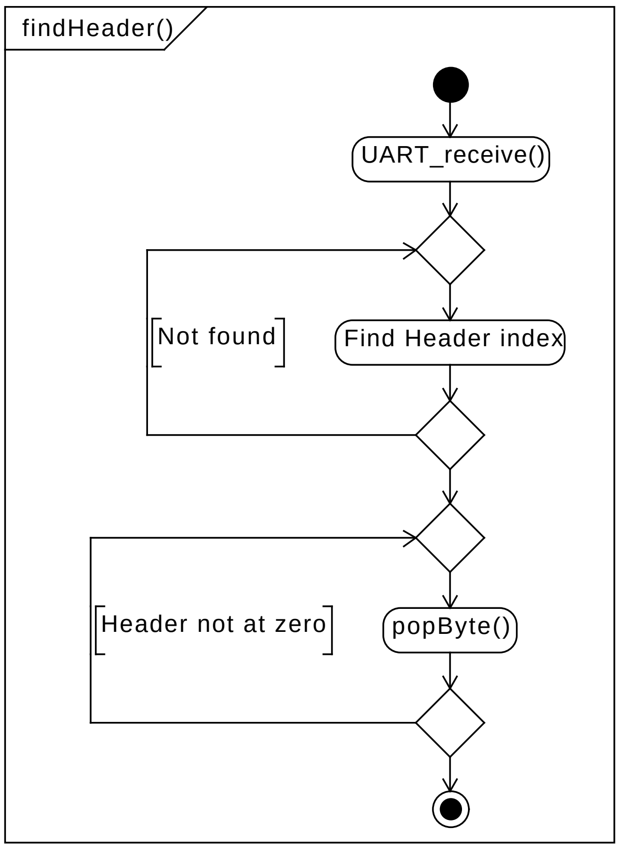


Figure 5.6: Flow diagram for `findHeader()`

5.5 SD-card - Ciseco B026

The SD-card is to be used without a file system, as the stored data will be hex values of known size. The necessary functions to write will be

```
uint8_t SD_save();
GPS_data_t SD_load();
uint8_t SD_delete();
```

where `GPS_data_t` is a struct containing the time and location of a given GPS data point. `SD_delete()` is meant to delete any data points already sent to the remote server.

Chapter 6

Implementation

6.1 SAMD21G18A

This section is dedicated to describing the configurations required on the SAMD21G18A platform. Each subsection will deal with the registers and values needed for each module.

6.1.1 Programming framework

Both Arduino and Atmel provides a framework for coding purposes for the Arduino MKR GSM 1400 board. Since Arduino sketches are very high level programming, this was considered to far from the processor. Atmel studio provide ASF(advanced software frame) for their processor, which includes drivers and helpfull tools. At the start of the project, it was assumed that ASF would be to high level also, resulting in the group programming in C without any major framework. Code examples and configurations was later researched in ASF.

6.1.2 Clock

The SAMD21G18A is able to run with a clock frequency of 48 MHz, but this clock is not started automatically. It is therefore necessary to activate, not just the clock called *DFLL48M*, but also set up the reference clock, the generic clocks and the required multiplexers.

During operation SAMD21G18A is able to maintain a given frequency through a DFLL(Digital Frequency Lock Loop). This system relies on a secondary clock as a reference source for the main clock. In the following list it is shown in what order the clocks and registers should be activated in order to get a stable system.

1. Enable XOSC32K clock (External on-board 32.768Hz oscillator). Used as reference for 48 MHz clock.

2. Set Generic Clock Generator (1) source to XOSC32K.

3. Active and set reference for Generic Clock Multiplexer.
4. Enable DFLL48M.
5. Set DFLL48M as source for Generic Clock Generator 0.
6. Edit Pre scaler for OSCM clock to obtain 8 MHz.
7. Set OSC8M as source for Generic Clock Generator 3.

Going through the data sheet of SAMD21G18A the group was not able to configure the clock to run correctly. It was therefore assessed, that using a free open source implementation from example code[2], would speed up progress and result in a working prototype. The example sketch was import into atmel studio through it's sketch import system, and a startup.c file was created setting up the clock.

6.1.3 UART: Configuration

This section will describe the processor specific register configuration needed to get the UART running on SAMD21G18A. Mainly there are two control register A and B, but also a register for interrupt enabling and one for baud rate.

table 6.1, table 6.2 and table 6.3 contains values for data, if a bit or variable is not mentioned in the table, assume it to be 0.

Name	Value (Sercom2 | | Sercom5)
Data Order	0
Communication Mode	0
Frame Format	0
Receive Data Pinout	1 | | 3
Transmit Data Pinout	0 | | 1
Sample Rate	1
Operating Mode	1

Table 6.1: Control A register values.

Name	Value
Receiver Enable	1
Transmitter Enable	1
Parity Mode	0
Stop Bit Mode	0
Character Size	0

Table 6.2: Control b register values.

Name	Value
Receive Complete Interrupt Enable	1
Transmit Complete Interrupt Enable	1

Table 6.3: Interrupt Enable register values.

6.1.4 Baud Rate

To implement baud rate calculation, the data sheet for SAMD21G18A was consulted. The group expected to use asynchronous arithmetic baud, and started to set up the implementation of this. After having a functional UART set up, an oscilloscope was used to auto detect the baud rate. After several attempts the baud rate was still unstable, having a range from 9300 -> 9800 bits/s. This meant that the NEO-7M and SARA-U201 module communications was scrambled at times.

Trying to understand the UART set up better, an example from Arduino sketches was reviewed. In the example asynchronous fractional baud calculation was used, and the group decided to test out this approach. Although nothing was noticeably change in the control registers, from the previous tests, the UART was now stable. Following will be a example of fractional baud value calculation.

$$BAUD = \frac{f_{ref}}{S * f_{BAUD}} - \frac{FP}{8}$$

Following snippet is taken from baud calculation in the program source code:

```
uint32_t baudTimes8 = (SystemCoreClock * 8) / (16 *
↪  uartSetup.baudRate);
uint16_t baudReg = ((baudTimes8 % 8) << 13) | (baudTimes8 >> 3);
```

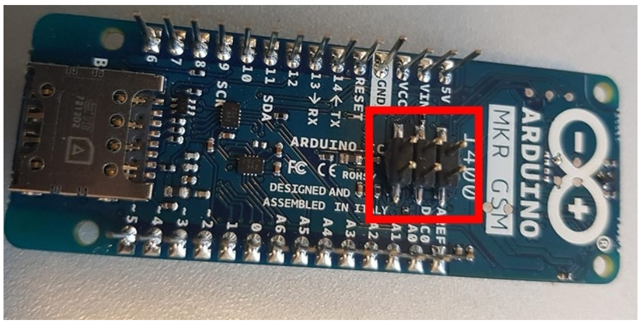


Figure 6.1: Arduino Arduino MKR GSM 1400 board with SWS pins attached.

6.1.5 Debugging

In circuit debugging was achieved through a SWD connection on the Arduino MKR GSM 1400 board. As shown on figure 6.1 a set of 6 pins was soldered on pads at the bottom of the Arduino MKR GSM 1400. Using an Atmel Ice MKII, debugging could be done through JTAG wires, as SWD is a subset of the JTAG connection.

6.2 GSM - SARA-U201

6.2.1 PDP Context

In order for the GSM module to send data packets out through GPRS, a PDP context is created through the command:

AT+CGDCONT=1,"IP","www.internet.mtelia.dk"

The command describes what packet protocol to use (IP) and the APN of the cellular network. When a context is declared and the device is registered on a network, the context can be activated. Once activated the module is assigned an IP, which will be used to send data through.

6.2.2 PSD

Creating sockets on SARA-U201 requires a internal PDP context, also described as a PSD (Packet Switch Data) profile. This is set up similar to the external context, just with another command:

AT+UPSD=0,1,"www.internet.mtelia.dk"

Defining context 0, using 1(APN) and "www.internet.mtelia.dk" as apn. Once both context has been activated, a socket is created, in this case a UDP socket, and data can be sent to the server.

```
struct GPS_data_t{
  uint16_t year;
  uint8_t month;
  uint8_t date;
  uint8_t hour;
  uint8_t minute;
  uint8_t second;
  uint32_t lat;
  uint32_t lon;
  // valid = 0, means data is valid.
  uint8_t valid;
};
```

Listing 6.1: GPS data struct.

6.3 GPS - NEO-7M

The location data received from the NEO-7M GPS is stored in a struct defined as shown in listing 6.1. The members are defined in sizes allowing the received data to be copied directly into the struct using `memcpy()`. This is made easy as the endianness of the NEO-7M and the SAMD21G18A match, both are little-endian.

6.4 µSD - Ciseco B026

The µSD card saving functionality was not implemented due to time constraints.

Chapter 7

Test

In this section an short introduction to how this project has used test from beginning to the end will be described.

7.1 Module tests

The goal of the module tests is to determine if the used modules are able to perform the actions demanded by the Requirements.

Module tests has been performed on the SARA-U201 GSM module, NEO-7M GPS module, and the Ciseco B026 µSD-card socket with µSD-card inserted.

7.1.1 GSM SARA-U201

The module test of the GSM SARA-U201 tests if the module is able to send a simple string to a known server.

The used test command is

AT+USOST=*SOCK*,"*IP*",*PORT*,11,"Hello World"

Where *SOCK* is decided by the SARA-U201 module, and *IP* and *PORT* are decided by the server.

Action	**Expected outcome**	**Outcome**
Send test command, via a USB to UART module, to Arduino MKR GSM 1400 coded with an Arduino Sketch forwarding the input to the SARA-U201.	The server log shows "Hello World".	Success. For more information see transcript in appendix appendix D.

Table 7.1: Module test of SARA-U201 GSM module.

7.1.2 GPS NEO-7M

The GPS module is tested by connecting it to a USB to UART module and sending the appropriate UBX commands. The setting is outdoors as to achieve the most precise localization.

Before this, the below command CFG-PRT (Port Configuration for UART) [8, p. 119-120], must be sent to the GPS module, to only enable UBX commands in and out, and set the baud rate to 9600:
0xB562060014000100000D0080000080250000010001000000000009A79

Action	Expected outcome	Outcome
Send *0xB562010700000819* (UBX-NAV-PVT/Get location) from computer to GPS module via USB to UART with 9600 baud.	Receive 92 B according to [8, p. 160-161].	92 B received. Output shown in appendix B.
Input coordinates achieved from above test into Google Maps.	Pin is placed at the location above command was issued at.	Pin is placed at the location the command was issued.

Table 7.2: Module test of NEO-7M GPS module.

7.1.3 Ciseco B026 socket and µSD-card

Before the test, use *dd* on a Linux device, to clear the µSD-card with `dd if=/dev/zero of=/dev/sdX`, where sdX is the µSD-card.

Action	Expected outcome	Outcome
Use an Arduino Sketch driver for the Arduino MKR GSM 1400 to save the string "Hello World" to the µSD-card via SPI. Remove the µSD-card from the Ciseco B026 socket and read it on a Linux device using *dd* to a file.	The output file shows "Hello World".	

Table 7.3: Module test of Ciseco B026 socket and µSD-card.

7.2 Integration tests

The integration tests are much like the module tests, except for the Arduino MKR GSM 1400 being the facilitator of the data.

7.2.1 Arduino MKR GSM 1400 with SARA-U201

The used test command is

AT+USOST=*SOCK*,"*IP*",*PORT*,11

Where *SOCK* is decided by the SARA-U201 module, and *IP* and *PORT* are decided by the server.

Action	**Expected outcome**	**Outcome**
Send test command from Arduino MKR GSM 1400 to SARA-U201 via UART.	SARA-U201 responds with @. Debugger shows @ in buffer	Hex value for @ (0x40) shown in debugger.
Send "Hello World"	The server log shows "Hello World".	The server log shows "Hello World".

Table 7.4: Integration test of Arduino MKR GSM 1400 and SARA-U201 GSM module.

7.2.2 Arduino MKR GSM 1400 with NEO-7M

Before this, the command CFG-PRT (Port Configuration for UART) [8, p. 119-120], must be sent to the GPS module.

Action	Expected outcome	Outcome
Send *0xB562010700000819* (UBX-NAV-PVT/Get location) from Arduino MKR GSM 1400, with debugger on, to GPS module via UART with 9600 baud.	Receive 92 bytes according to [8, p. 160-161].	92 B received. Output shown in appendix C.
Input coordinates achieved from above test into Google Maps.	Pin is placed at the location above command was issued at.	Pin is placed at the location the command was issued.

Table 7.5: Integration test of Arduino MKR GSM 1400 and NEO-7M GPS module.

7.2.3 Arduino MKR GSM 1400 with Ciseco B026 and µSD-card

Before the test, use *dd* on a Linux device, to clear the µSD-card with `dd if=/dev/zero of=/dev/sdX`, where sdX is the µSD-card.

Action	Expected outcome	Outcome
Use the Arduino MKR GSM 1400 to save the string "Hello World" to the µSD-card via SPI. Remove the µSD-card from the Ciseco B026 socket and read it on a Linux device using *dd* to a file.	The output file shows "Hello World".	

Table 7.6: Integration test of Arduino MKR GSM 1400 and Ciseco B026 socket and µSD-card.

7.3 Acceptance Tests

The goal of the acceptance test is to test the system as a whole. Due to time constraints, two acceptance tests are produced, one with and one without the Ciseco B026 and µSD-card.

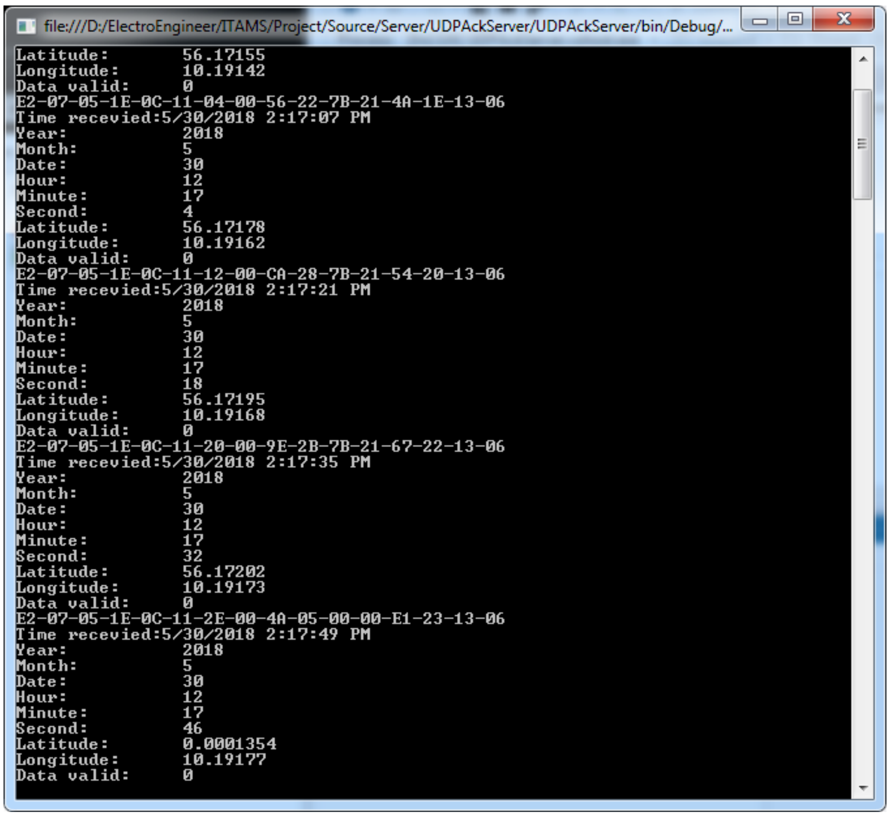
Acceptance tests are carried out outdoors to ensure good GPS signal quality.

Action	Expected outcome	Outcome
Power on GPS Tracker. Wait 5 min.	One valid location data point is shown on the server.	
Wait 5 min.	The server still shows one valid data point.	
Wait 15 min.	The server shows 15 to 20 valid location data points.	
A GPS location from the server is put into Google Maps.	Google Maps shows the location where the GPS Tracker was turned on.	

Table 7.7: Acceptance test of GPS Tracker with Ciseco B026 socket and µSD-card.

Action	Expected outcome	Outcome
Power on GPS Tracker. Wait 5 min.	1 to 5 valid data points are shown on the server.	The server receives $\approx$ 5 valid values pr. minute, but stops receiving after 2 min, due to an unknown interrupt error on the SAMD21G18A. Output shown in figure 7.1.
Wait 5 min.	5 to 10 valid data points are shown on the server.	
Wait 15 min.	The server shows 15 to 25 valid location data points.	
A GPS location from the server is put into Google Maps.	Google Maps shows the location where the GPS Tracker was turned on.	The location is correct. See figure 7.2.

Table 7.8: Acceptance test of GPS Tracker without Ciseco B026 socket and µSD-card.


file:///D:/ElectroEngineer/ITAMS/Project/Source/Server/UDPAckServer/UDPAckServer/bin/Debug/...

```
Latitude:        56.17155
Longitude:       10.19142
Data valid:      0
E2-07-05-1E-0C-11-04-00-56-22-7B-21-4A-1E-13-06
Time recevied:5/30/2018 2:17:07 PM
Year:            2018
Month:           5
Date:            30
Hour:            12
Minute:          17
Second:          4
Latitude:        56.17178
Longitude:       10.19162
Data valid:      0
E2-07-05-1E-0C-11-12-00-CA-28-7B-21-54-20-13-06
Time recevied:5/30/2018 2:17:21 PM
Year:            2018
Month:           5
Date:            30
Hour:            12
Minute:          17
Second:          18
Latitude:        56.17195
Longitude:       10.19168
Data valid:      0
E2-07-05-1E-0C-11-20-00-9E-2B-7B-21-67-22-13-06
Time recevied:5/30/2018 2:17:35 PM
Year:            2018
Month:           5
Date:            30
Hour:            12
Minute:          17
Second:          32
Latitude:        56.17202
Longitude:       10.19173
Data valid:      0
E2-07-05-1E-0C-11-2E-00-4A-05-00-00-E1-23-13-06
Time recevied:5/30/2018 2:17:49 PM
Year:            2018
Month:           5
Date:            30
Hour:            12
Minute:          17
Second:          46
Latitude:        0.0001354
Longitude:       10.19177
Data valid:      0
```

Figure 7.1: Server log of acceptance test of GPS Tracker without µSD-card.

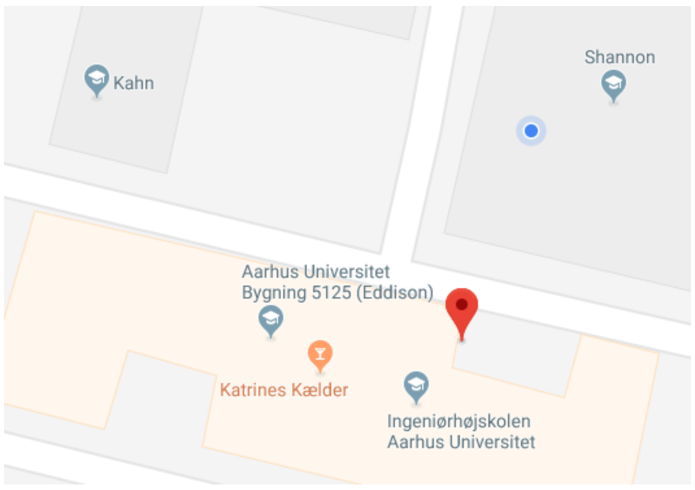


Figure 7.2: Google maps of one of the locations shown in figure 7.1. The blue dot is the actual location, the red is the GPS coordinate. Some of the deviation might be due to the use of float and not double on the server, for storing the coordinates.

Chapter 8

Conclusion

The goal of the project has been to create a GPS Tracker able to locate itself and send the location to a remote server, possibly sending at specified intervals to conserve power. The final prototype incorporates an ARM µC, a GSM cellular module and a GPS module.

A great deal of work went into getting to know the SAMD21G18A µC, this was a hindrance to the work flow. As the SAMD21G18A is built on the SAM structure and not the AVR structure, a lot of new registers had to be understood. A solution to this could have been to use Arduino Sketches or the Atmel ASF framework, but the team felt this might have led to less learning from the project. Another solution to the unknown architecture could have been to use the known ATmega2560.

The interfacing with the GSM and GPS modules were fairly straightforward, but was again hindered by the above described lack of knowlede of the SAMD21G18A µC. This showed in an unreliable UART communication, which was not made stable until the very end of the project.

An issue encountered with the GSM module was the need for sudden bursts of current, the antenna needed. These bursts are up to 2 A, which a standard USB power supply is not able to supply. The solution was to attach a 3.8 V mobile phone battery to the Arduino MKR GSM 1400.

The final prototype is able to locate itself. It is not able to save location data over extended periods without internet access. The GPS data gathered is sent to a remote server, available for backup or analysis.

Chapter 9

Bibliography

Datasheets

[3] Microchip Technologies Inc. **32-bit ARM-Based Microcontrollers**. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/40001882A.pdf.

[6] u-blox. **NEO-7, Data Sheet**. URL: https://www.u-blox.com/sites/default/files/products/documents/NEO-7_DataSheet_%28UBX-13003830%29.pdf.

[7] u-blox. **SARA-U2 series, Data Sheet**. URL: https://www.u-blox.com/sites/default/files/SARA-U2_DataSheet_%28UBX-13005287%29.pdf.

[8] u-blox. **NEO-7, Protocol Description**. URL: https://www.u-blox.com/sites/default/files/products/documents/u-blox7-V14_ReceiverDescrProtSpec_%28GPS.G7-SW-12001%29_Public.pdf.

[9] u-blox. **u-blox Cellular Modules - AT Commands Manual**. URL: https://www.u-blox.com/sites/default/files/u-blox-CEL_ATCommands_%28UBX-13002752%29.pdf.

Webpages

[1] Arduino. **Arduino MKR GSM 1400**. URL: https://store.arduino.cc/mkr-gsm-1400.

[2] **Arduino Sketch: SerialPassthrough.ino**. URL: https://github.com/arduino/Arduino/blob/master/build/shared/examples/04.Communication/SerialPassthrough/SerialPassthrough.ino.

[4] Rapid Electronics Limited. **Ciseco B026 I/O POD microSD Card Socket 3.3V SPI Communication**. URL: https://www.rapidonline.com/ciseco-b026-i-o-pod-microsd-card-socket-3-3v-spi-communication-73-4833.

[5] **Satllite Internet Access**. URL: https://en.wikipedia.org/wiki/Satellite_Internet_access.

Appendix A

SARA-U201 AT communication

Name	Description	Command and expected *Response*
AT+CCID	Used to get the sim card ID, which could be used as identifier for the transmissions.	AT+CCID *+CCID:<CCID>*
AT+CREG?	Used to read the current network status. Returns CREG: 0,? The return value ? describes if the modem is registered on the network.	AT+CREG? *+CREG: 0,<state>*
AT+CGD-CONT	Set up IP and APN for the network provider for the sim card	AT+CGDCONT=1,"IP","www.internet.mtelia.dk" *OK*
AT+CGATT	Attaches the GSM to the GPRS service before proceeding to activate internet protocols	AT+CGATT=1 *OK*
AT+CGACT	Activate the PDP context of the GPRS connection set up with the cgdcont command	AT+CGACT=1,1 *OK*
AT+UPSD	Set up APN for the internal context in the module, necessary for creating data sockets	AT+UPSD=0,1,"www.internet.mtelia.dk" *OK*
AT+UPSDA	Activate the internal context and allow the module to bin sockets for data transfer	AT+UPSDA=0,3 *OK*
AT+USOCR	Set up a socket for internet protocol transmission, use 17 for UDP protocol. Returns the socket id, for future use.	AT+USOCR=17 *+USOCR: 1*
AT+USOST	Send LENGTH amount of bytes through bound SOCK, to the stated IP on PORT. SARA-U201 will respond with @ after which the binary data can be send	AT+USOST=0,"IP",PORT,LENGTH @ Hello World *OK*

Table A.1: Table of used AT command to control SARA-U201 module. Each command must be ended with <CR><LF>

Appendix B

GPS module test data

Desc	Code (hex)	Meaning
Header	b562	
ID	0107	
Length	5400	
iTOW	f0000a19	GPS Time of week
year	e207	2018
month	05	April
day	18	24th
hour	14	20
min	29	41
sec	08	08
valid	07	Fully Resolved
tAcc	1b000000	
nano	677c0000	
fixType	03	3D-fix
flags	01	Valid fix
reserved0	0a	
numSV	06	Number of satellites: 6
lon	285a1406	Longitude: 10.199 709 6
lat	b8dd7a21	Latitude: 56.170 027 999 999 995
height	2e560100	
hMSL	36af0100	
hAcc	0d260000	
vAcc	48360000	
velN	a1ffffff	
velE	1d000000	
velD	b0ffffff	
gSpeed	63000000	
heading	e77e6001	
sAcc	4d030000	
headingAcc	c615e800	
pDOP	e700	
reserved2	0000	
reserved3	84d31100	
checksum	b05c	

Table B.1: Example of GPS answer to UBX-NAV-PVT[8, p. 160-161]. Hex code is little endian.

Appendix C

GPS integration test data

Desc	Code (hex)	Meaning
Header	b562	
ID	0107	
Length	5400	
iTOW	f074381c	GPS Time of week
year	e207	2018
month	05	May
day	19	25th
hour	0b	11
min	1e	30
sec	2c	44
valid	07	Fully Resolved
tAcc	2d000000	
nano	604b0400	
fixType	03	3D-fix
flags	01	Valid fix
reserved0	0a	
numSV	08	Number of satellites: 8
lon	20241306	Longitude: 10.191 772 799 999 999
lat	65377b21	Latitude: 56.172 323 7
height	28da0100	
hMSL	25330100	
hAcc	d3390000	
vAcc	53450000	
velN	1a000000	
velE	7b000000	
velD	11010000	
gSpeed	7e000000	
heading	59c38901	
sAcc	52030000	
headingAcc	4c8f3c00	
pDOP	e100	
reserved2	0000	
reserved3	84d3a437	
checksum	0020	

Table C.1: Example of GPS answer to UBX-NAV-PVT[8, p. 160-161]. Read from debugger. Hex code is little endian.

Appendix D

Sara-U201 Acceptance Test - Transcript

- 25-05-2018 14:12:??.??? [TX] -
  AT+CGREG?<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+CGREG=<CR>
  <CR><LF>
  +CGREG: 0,1<CR><LF>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+CGDCONT=1,″IP”,″www.internet.mtelia.dk”<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+CGDCONT=1,″IP”,″www.internet.mtelia.dk”<CR>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+CGATT=1<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+CGATT=1<CR>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+CGACT=1,1<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+CGACT=1,1<CR>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+CGDCONT?<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+CGDCONT?<CR>
  <CR><LF>
  +CGDCONT: 1, ″IP”, ″www.internet.mtelia.dk”,″10.225.178.101″,0,0<CR><LF>
  <CR><LF>
  OK<CR><LF>

- 25-05-2018 14:12:42.254 [TX] -
  AT+UPSD=0,1,"www.internet.metelia.dk"<CR><LF>
- 25-05-2018 14:12:42.375 [RX] -
  AT+UPSD=0,1,"www.internet.metelia.dk"<CR>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:54.954 [TX] -
  AT+USPDA=0,3<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+USPDA=0,3<CR>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+USOCR=17<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+USOCR=17<CR>
  <CR><LF>
  +USOCR: 0<CR><LF>
  <CR><LF>
  OK<CR><LF>
- 25-05-2018 14:12:??.??? [TX] -
  AT+USOST=0,"188.114.136.5",30000,11,"Hello World"<CR><LF>
- 25-05-2018 14:12:??:??? [RX] -
  AT+USOST=0,"188.114.136.5",30000,11,"Hello World"<CR>
  <CR><LF>
  +USOST: 0,11<CR><LF>
  <CR><LF>
  OK<CR><LF>