Tracking System

Suaineadh satellite experiment
Bachelor Thesis

Name  Martine Selin Carl Brengesjö
Email  mseli@kth.se carlbre@kth.se

Examiner  William Sandqvist
Supervisors  Adam Wujek, Jerker Skogby
Abstract

The purpose of this bachelor thesis is to present a tracking system for the Suaineadh satellite experiment. The experiment is a part of the REXUS (Rocket Experiments for University Students) program and the objective is to deploy a foldable web in space. The assignment of this thesis is to develop a tracking system to find the parts from the Suaineadh experiment that will land on Earth. It is important to find the parts and recover all the data that the experiment performed during the travel in space. The implementation of this thesis investigates two different ways to track and find the experiment. The first way is to locate the experiment module by a Global Positioning System (GPS) and send the coordinates to a satellite modem, controlled by a programmed microprocessor. The other way is by using a radio beacon that sends a specific radio frequency. The results of this thesis presents a prototype for the tracking system with a GPS and the satellite modem and code example for the microprocessor. It also presents a working radio frequency beacon system on a Printed Circuit Board. The thesis had some unexpected incidents and had to change some directives. This rendered the work to take longer time then estimated. Despite the difficulties resulted this thesis in a working system to track the experiment.

Sammanfattning

Contents

1 Introduction and background 1
   1.1 The Furushiki technique ................................... 1
   1.2 The Suaineadh experiment .................................. 2

2 Problem definition 2
   2.1 Localisation .................................................. 2
   2.2 Transfer information of location .............................. 3
   2.3 Challenges in space .......................................... 3

3 Solving the assignment for the thesis 4
   3.1 Recover the parts by the coordinates ....................... 4
   3.2 Recover the parts by a Radio Frequency ..................... 4
   3.3 Printed Circuit Board with GPS and RF ..................... 4
   3.4 Functional testing .......................................... 5
      3.4.1 Hardware development ................................... 5
      3.4.2 Software development ................................. 5

4 Implementation of the project 5
   4.1 Literature study ............................................. 6
   4.2 Evaluation of equipment .................................... 6
      4.2.1 Programmable Integrated Circuit ...................... 6
      4.2.2 Radio Frequency Beacon ............................... 6
      4.2.3 Data transmission ...................................... 6
      4.2.4 Global Positioning System ........................... 8
   4.3 Software design ............................................ 11
      4.3.1 Goals for software implementation for PIC .......... 12
      4.3.2 Configurations for the satellite modem .............. 12
      4.3.3 The PIC programming for modem and GPS ............ 12
      4.3.4 Testing of the modem .................................. 13
      4.3.5 Problems in the programming phase with STX modem . 15
      4.3.6 GPS tracking system .................................. 16
      4.3.7 Changes in guidelines ................................ 16
      4.3.8 RF beacon ............................................. 16
   4.4 Designing of the PCB ....................................... 17
      4.4.1 Changes in guidelines ................................ 19
   4.5 Construction of PCB ......................................... 19
   4.6 The PIC code for the RF Beacon ................................
   4.7 Testing of the RF beacon board ............................... 21
      4.7.1 Testing setup .......................................... 21
      4.7.2 Power interrupts ...................................... 21
      4.7.3 No transmission before liftoff ......................... 22
      4.7.4 PIC code and State machine ............................ 23
      4.7.5 Battery testing ....................................... 23
   4.8 Completion .................................................. 24

5 Discussion 25
   5.1 Areas of improvement ..................................... 25
      5.1.1 Antenna for the STX2 modem .......................... 25
      5.1.2 Testing ................................................ 25
      5.1.3 Radio transmission .................................... 25
   5.2 Retrospect .................................................. 25

6 Conclusion 27
7 References

A RF beacon
   A.1 Circuit diagram of the RF beacon .................. 29
   A.2 RF Beacon top ...................................... 30
   A.3 RF Beacon bottom ................................. 33

B Code
   B.1 Modem testing code ............................... 38
   B.2 GPS testing code ................................. 38
   B.3 RF Code ......................................... 39
1 Introduction and background

The space has been a source of fascination for mankind for a very long time. In our quest to understand more about the universe, new technologies have been developed to reach out beyond our current borders. The space stations and shuttles give us possibilities to observe more of our surroundings. Satellites around the globe simplify our everyday lives with communication and positioning of locations and directions. The need for power in these systems is an ever-increasing task for authorities and companies building them. The traditional approach is to mount static solar panels to generate electricity. The panels can be folded to meet space requirements for the space shuttles. As new technologies develop, for example more flexible panels, the need for a technique to deploy a foldable fabric in vacuum need to be verified. An alternative use for this is a web that robots can travel along to help astronauts with transport of tools and equipment or for large-scale communication antennae. The Japanese origami technique called Furushiki has inspired the technique for folding fabrics in space.

1.1 The Furushiki technique

The Furushiki technique is a traditional Japanese way to fold large fabrics to occupy a small volume. Attempts to adapt the technique in space have been made before. One of the first was conducted by Japanese scientists in which mathematical and computer simulations is used to model the space web[1]. Two ways of deployment were tested, one in which satellites span the web to stabilize the expansion and to hold the fabric in place, and the second approach is where the fabric is rotated and the centrifugal force holds the material straight. The test also compared the result of different positions of the space web relative to Earth. A practical evaluation of the Furushiki technique is needed to expand the knowledge of non-gravity deployment. The new experiment is called Suaineadh.

Figure 1: Square web
1.2 The Suaineadh experiment

The Suaineadh satellite experiment is a student project to send out a test module in space by a sounding rocket. The rocket is launched from Esrange, located in the north of Sweden, and the experiment is a part of the REXUS program. The program is a platform for rocket-borne experiments for university students, that launches two rockets each year. REXUS is a collaboration between the German Aerospace Center (DLR) and the Swedish National Space Board (SNSB). The program is available to other European countries through the European Space Agency (ESA). The experiments are modules delivered by the Rexus rocket. The Suaineadh experiment will prove the feasibility of a novel concept for deploying a square web in space, illustrated in figure 1. The module is called Central HUB And Daughter (CHAD) and is mounted in the nosecone of the rocket. Figure 2 displays how the architecture looked from the beginning of working with the experiment. After returning to Earth the visual and test data of the deployment need to be extracted. Therefore it is necessary to locate and recover the module.[2]

2 Problem definition

After the launch of the module for the Suaineadh experiment, the search for the module begins. The purpose of this bachelor thesis is to design and construct the tracking system to find the parts of the experiment. It is important to locate the module and recover all the experiment data. It is also important to take care of the experimental parts. In the following subsections we will describe different problem areas, the first one is localization of the module and the following are transfer information and the last one is about the challenges in space.

2.1 Localisation

To recover all the data from the Suaineadh satellite experiment, it is important to locate where the module has landed. A trajectory history and sent route information will help the rescue team to recover all the parts thanks to a working tracking system. The construction of the tracking system need to have as small parts as possible to fit in the rocket and the parts should have a low voltage so the system can stay alive for at least five hours after landing. During the time in space the tracking system need to receive data about the location through the GPS, which should support high altitude and a good reception to most of the satellites.

If the tracking system falls apart after landing on Earth, it is possible that the modem will stop sending positions. Therefore it is necessary to have a more reliable system to make sure that it recover all the data. The RF beacon system will be used to transmit the signals and have a range of more than ten km. It is essential to have its own power system if the main tracking system experience power failure.
2.2 Transfer information of location

100 seconds after start of experiment the tracking system will start. It is important that the power for transmitting units like RF beacon and the GlobalStar modem shall be off until CHAD separates. If the transmitting units are on, there is a risk of interfering with the launching from Esrange. Figure 2 shows the whole architecture of the Suaineadh satellite experiment and the relation of the tracking system is shown at the top right corner of the picture.

The tracking system will get the coordinates for the experiment module and send the necessary data further to a satellite modem. If the tracking system should have a power interruption or restart, it will still be able to send the correct data or storage it. For the helicopters to find and locate the experimental module, the RF beacon system need to be operational 5h after landing on Earth.

2.3 Challenges in space

Operating in space provides different challenges than within the atmosphere. There is no air, no protection from the sun’s ultraviolet radiation and no defense against space debris. This experiment is only exposed to the harsh conditions for a limited duration and only a few are needed to be considered. There are two main requirements that the tracking system has to be able to withstand. First, the temperature in the mesosphere is as low as -80°C. As the exposure is short, the guideline for the Suaineadh experiment is that the system should be able to withstand temperatures around -40°C. This impacts the choice of components and can be tested using a vacuum chamber after construction to ensure the operation of the tracking system. Second, the launch of the rocket and final impact requires a robust system. The tracking system have to be constructed to sustain vibrations and to be sturdy enough that it should survive the impact.
3 Solving the assignment for the thesis

To solve the problems during the development of a working tracking system, is to either use to locate the experiment by coordinates from a Global Positioning System, GPS or use a Radio Frequency (RF) beacon system. To manage this comprehensive project it has been divided so that Martine primarily has been handling the GPS and the modem while Carl’s main responsibility has been the RF beacon.

3.1 Recover the parts by the coordinates

To recover the parts by coordinates a microprocessor will be used to collect the information from the GPS and send it to a satellite modem. One of the goals is to create the software for the PIC that will handle the data from GPS and communicate with the modem.

3.2 Recover the parts by a Radio Frequency

The assignment is to design and implement a small board with a beacon that will send a radio frequency signal. Batteries need to keep the transmission on for at least five hours.

3.3 Printed Circuit Board with GPS and RF

We need to design a board for the tracking system to get a steady connection with all the components and to be able to mount it in the module. Then the components are connected electrically using conductive pathways, tracks or signal traces. +5 V are provided from the module. As the RF beacon need enforced package to withstand the impact. The size that the PCB have to follow is the standard size PC/104, 96x90 mm. Larger will take too much volume in the module. Figure 3 describes how the parts that can be used in the assignment.

![Diagram of the tracking system](image)

Figure 3: Original scheme for the required parts on the PCB

In the centre of the picture there is a microprocessor (PIC) that will send a start signal to the RF (Radio Frequency) beacon and GPS (Global Position System) to start receiving and transmitting signals. There are two ways to recover the data from the experiment and the most important one is the RF beacon. The
RF beacon will transmit a radio signal straight to the base station and the signal will be picked up by helicopters collecting the debris. The GPS to the right in figure 3, is the second way to locate the position of the module. It will transmit GPS data back to the PIC and the PIC will send the position data to the modem, then the modem will transmit it to web portal that could be picked up at the base station.

3.4 Functional testing

The experiment have to be carefully tested as there are no possibilities to correct mistakes and manually restart the process when the rocket has been launched. The functions have to be evaluated and if any obstacle appears it has to be resolved. Some key elements have to be specifically verified. The modem and GPS have to go through testing of the main function and to determine the limits of the connection and if there are any angle to the satellites, that results in degraded performance. The PIC have to save the state to be able to resume operations if there is a power failure and if a restart is necessary. This have to be verified by disconnecting the power source. The RF beacon have to be operational five hours after impact.

3.4.1 Hardware development

The testing of the system can only be done when the system is completed. The testing will include:

- Verification that no transmission is performed before the experiment is started.
- The complete cycle from the initialization until the transmission of the RF beacon ends.
- Runtime of the beacon.
- The systems ability to handle power interrupts.

3.4.2 Software development

Testing of software involves validating and verifying the software program. To check if the program works as expected and test how it will react on unexpected events.

4 Implementation of the project

A literature study was made in the beginning of this bachelor project. The aim was to find as much information as possible about rocket experiment and increase the understanding of the Suaineadh project. Followed by the literature study we needed to find suitable parts for the development of the tracking systems for the Suaineadh project. After that a structure for the tracking system was built.
4.1 Literature study

A literature study was conducted after first having an information search. This information has mainly focused on articles in previous experiments with REXUS project. Libraries have been visited, where books have been read to increase knowledge in areas such as design and implementation of C code and MPLAB. Several websites have also been studied following a tip from a tutor on the project company in the topic "Programming C". The study has increased our understanding of the project and problems to create the tracking system.

4.2 Evaluation of equipment

There are many parts from different manufacturers that should fit on the PCB. In this section of the study we will evaluate the essential parts of programmable integrated circuit, radio frequency beacon, transmission and the global positioning system. As this is a student project, the price is a main factor as long as the components fulfills the requirements.

4.2.1 Programmable Integrated Circuit

Today it is popular to program the microcontroller circuits because of its low price and convenient size. In modern PIC processors, there is also a special hardware, for Universal Asynchronous Receiver/-Transmitter, (UART) communication.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Pin</th>
<th>Price($)</th>
<th>RAM (bytes)</th>
<th>Operating voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microchip</td>
<td>PIC16F628</td>
<td>18</td>
<td>10</td>
<td>225</td>
<td>3-5.5V</td>
</tr>
<tr>
<td>Microchip</td>
<td>PIC12F629</td>
<td>8</td>
<td>5</td>
<td>64</td>
<td>3-5.5V</td>
</tr>
<tr>
<td>Microchip</td>
<td>PIC16F690</td>
<td>20</td>
<td>7</td>
<td>256</td>
<td>2-5V</td>
</tr>
</tbody>
</table>

A comparison among the most common programmable integrated circuits shows that PIC16F690 is the most affordable on the market. PIC16F690 has a low price and high pin count. It can store 256 bytes of EEPROM data in the flash memory and operating voltage is low 2-5 volts. The PIC16F690 meets all requirements and should be enough to be a part of the GPS tracking system.

4.2.2 Radio Frequency Beacon

The most common Radio Frequency (RF) beacon, TX1, is enough for our needs. It must have separate power for an independent operation. The range of the transmitter is over 10 km. The rate of the transmitted data is up to 10 kbps. There is a possibility to send data through the RF beacon, however this feature will not be used. The available frequency is 173.250 MHz.

4.2.3 Data transmission

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Orbit</th>
<th>Price($)</th>
<th>Activation fee</th>
<th>Variable data cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globalstar</td>
<td>STX-2</td>
<td>More south</td>
<td>100</td>
<td>40</td>
<td>0.15/msg</td>
</tr>
<tr>
<td>Iridium</td>
<td>9602</td>
<td>The polar</td>
<td>500</td>
<td>not specified</td>
<td>not specified</td>
</tr>
</tbody>
</table>

Martine Selin, Carl Brengesjö  
November 8, 2011
The two modems from Globalstar and Iridium use separated networks in different satellites. A third satellite network, Thuraya, exists but have a similar coverage to Globalstar but no comparable advantages. The networks of the satellites use different orbits and are differently priced.

Comparing Globalstar and Iridium, Globalstar has a lower initial cost but the networks of the satellites orbit farther from the polar circle which needs an unrestricted view to the south for a good reception from the surface. The Iridium orbits closer to the north pole and provides good coverage and constant signal strength. The Globalstar network was used in the SQUID-project [3], an earlier REXUS experiment, so knowledge is accessible if needed. The earlier experience is the main factor for choosing the Globalstar.

**Transmission of data through the simplex modem**

![STX2 Diagram](image)

Figure 4: Installation of the STX2

The PIC supplies power to the STX2 and directly after startup the modem will be set into standby mode. To set up a communication with the PIC and the modem it is necessary to first to do a handshake, see figure 4 for connections. The PIC needs to send a Request To Send (RTS) signal to the modem. If it receives an answer on the Clear To Send (CTS) pin from the modem, it is OK to send a message. If the STX does not respond within 20 ms with CTS, the PIC should consider the STX2 busy and wait a minute to resend RTS.

The primary interface for the STX2 for communication is the TTL asynchronous serial port (UART). The PIC will send the data package from Rx pin and after the last byte of data it will deactivating RTS. The STX2 will then deactivates the CTS line and send a Response Serial Packet on the Modems Tx pin, see figure 5. The STX2 then starts the first transmission attempt of the packet after completion the STX2 returns to Standby Mode.

Simplex messages are transmitted via the Globalstar low-earth orbit satellite network, using an uplink-only connection (one-way data transmission), see figure 6.

The serial port works with the serial limitations of 9600bps, 8 data bits, no
parity, 1 stop bit. Each command from the DTE to the modem (STX2) is sent in a serial packet.

The serial packet format can be seen in figure 7.

4.2.4 Global Positioning System

The Global positioning system is used to acquire coordinates for the Suaineadh experiment module. The GPS module should acquire reception at high altitudes and have low power consumption.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Size (mm)</th>
<th>Price ($)</th>
<th>Max Altitude (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniTraQ</td>
<td>GT-320FW</td>
<td>34x34</td>
<td>38</td>
<td>25000</td>
</tr>
<tr>
<td>GlobalSat</td>
<td>ET-318</td>
<td>15,2x14</td>
<td>54</td>
<td>18000</td>
</tr>
</tbody>
</table>

The price is the main reason for choosing the GT-320FW, see figure 8. It is a compact GPS receiver with an integrated patch antenna that measures 34x34x10mm and weighs only 20 grams. It runs on low-voltage 3.6 to 6v DC which is good for designing the PCB. This module has a special high-altitude firmware version loaded to support operation at altitudes in excess of 25 kilometers.
GPS module specifications

The GPS module communicates through the serial interface at RS232 (Recommended Standard 232) or LV TTL (Low Voltage Transistor-Transistor Logic) level. The LV TTL interface will be used due to it being easier to get equipment for troubleshooting. While connecting the serial interface on the GPS module, it will start sending out messages with the NMEA (National Marine Electronics Association’s) protocol, at 4800 baud automatically every second.

The protocol will send out seven different strings of data about the GPS receiver, the one we are interested starts with $GPGGA, (GPS FIX DATA) then we have the information about the time, position and position-fix related data.

The serial data format:

$GPGGA, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, M, <10>, M, <11>, <12>, *, <13>, <CR>, <LF>

Example:

$GPGGA,104549.04,2447.2038,N,12100.4990,E,1,06,01.7,00078.8,M,0016.3,M,*5C<CR><LF>

The explanation about the $GPGGA information is described in figure 9. The position coordinates and the definition of data is explained in the next subsection.

The GPS was tested to verify if the reception was strong and position was
correct. The test was made with three similar GPS modules and on two different locations in Stockholm. First in Kista and then in Solna, the results are shown in the table below. It is not a big difference between all three of them.

Kista: Latitude, 59.4051 Longitude, 17.94172

<table>
<thead>
<tr>
<th>GPS 1</th>
<th>59.405013</th>
<th>17.941627</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS 2</td>
<td>59.404798</td>
<td>17.941750</td>
</tr>
<tr>
<td>GPS 3</td>
<td>59.404980</td>
<td>17.941588</td>
</tr>
</tbody>
</table>

Solna: Latitude, 59.343377 Longitude, 18.001764

<table>
<thead>
<tr>
<th>GPS 1</th>
<th>59.343563</th>
<th>18.002463</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS 2</td>
<td>59.343563</td>
<td>18.002593</td>
</tr>
<tr>
<td>NO FIX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Definition of the data**

$GPGGA$ specifies that following is the time and the values of the latitude and longitude coordinates. The time is given in the format of 121030.00 for hours, minutes and seconds. The latitude and longitude are a way to specify locations around the globes described as angles from the equator and from the north-south line specified as the Coordinated Universal Time, UTC, also known as GMT (Greenwich Mean Time), that runs through London, England. The latitude is
specified between 0 and 90 and whether the location is north or south of the equator. The longitude is between 0 and 180 and a description if the location is west or east of UTC, see figure 10. The coordinates from the GPS-module is given in degrees, minutes and seconds which is a different way to describe the angle without writing with decimal numbers. The degree is divided in minutes (1/60 of a degree) and minutes in seconds (1/60 of a minute).

![Figure 10: Lines of longitude (left) and latitude (right)](image)

4.3 Software design

The 16F690 PIC is manufactured by the company Microchip Technology. PICkit 2 Development Programmer will be used for programming the microcontroller. The software recommended from the manufacturer is called MPLAB Integrated Development Environment (IDE), a 32-bit application that runs on Microsoft Windows. The code will be written in C and compiled by a CC5X B Knudsen.
C Compiler. [4] The micro-controller also has a UART. This enables to translate data among serial forms. For debugging and check that correct data is transferred from the PIC, a USB-serial-TTL Cord with integrated electronics used. The terminal to check the bytes from the TTL cable and was a program called AccessPort, a good advanced serial monitor. It is easy to see the results in both HEX code and ASCII and to shift between the different baud rates needed to communicate with the GPS, PIC and modem. Figure 11 shows how the test and set up for the communication with the STX modem could look like.

4.3.1 Goals for software implementation for PIC

The aim of programming the PIC, is to get the GPS to send the right information of data to the modem. It will include to get the coordinates from the GPS and build the correct package to send to the modem. The modem must also be configured to communicate with the PIC and send information about it states.

The code should be easy to understand and it should be easy to change variables, partly to change the time variables. The code should also be built with smart features so it is easy to modify and understand its purpose.

The goal for programming of the functions required for the PIC:

- Initialize the GPS and sample data from the GPS module.
- Select the right data to transfer to the modem.
- Build a package to send to the modem with the correct CRC calculations.
- Initialize the modem to be able to receive data.
- Send a valid package to the modem.

For a function diagram see appendix

4.3.2 Configurations for the satellite modem

The coordinates of the Suaineadh experiment need to be transmitted to pinpoint the location of the module after returning to Earth. Satellite communication has limited data transfer rate and it is necessary to only send essential data. To send data to satellites the tracking system need a modem and an antenna for transmission. The transmission from the module can begins no earlier than 100 second after the separation from the rocket and will last until impact. The data collected will then be used to locate the module. The modem will have access to a 3.3 V-line to power the modem and the antenna.

4.3.3 The PIC programming for modem and GPS

The main program has two states, one to get the GPS signals and one to communicate with the modem.

In the first state it initializes the GPS by setting the baud rate to 4800 which is necessary to receive the GPS signals correct. It then starts to receive a valid string of data according to specifications in the function GetGPSCoordinates.
It will check for the GPPGA string and skips some of the characters that is not needed. If it does not find a valid string it will output a location error, and then the state will try to check for another string. When it has found a valid string the program will save the data in a global variable and go to state two to try sending the package.

The first thing state two is doing is to change the baud rate to 9600 for the communication with the STX-2 modem. It will then try to generate a valid cyclic redundancy check, CRC, for the data message. When it has generated the CRC for the package, the PIC will try to send the package by setting the RTS low, it will then wait for the modem to answer with CTS low. Then the PIC will send a byte from the RX port on the modem. If the TX pin is connected to the modem the modem will send an acknowledgement package and if the CRC is wrong the modem will not respond with anything.

The modem then tries to send the package with the connected antenna, how fast and how often the package is sent is set by the settings we have configured when the modem was initialized.

4.3.4 Testing of the modem

To check if the RTS and CTS works, it is possible to send a test package to see if the modem will respond to it.

![Figure 12: STX modem #1 is responding](image)

According to the specifications in the STX2 data sheet [6] it is possible to use a specific package to request the firmware or ID of the modem. Figure 12 shows a picture from AccessPort how the STX modem #1 is responding to the packages.

User request the modem to return its unit ID: 0xAA 0x05 0x01 CRC: 0x50 0xD5

Sending a request the modem to return its unit ID: 0xAA 0x05 0x01 CRC: 0x50 0xD5
Response from the modems

| STX #1  | 0xAA 0x09 0x01 0x00 0x0E 0x4C 0x1F 0x25 0xBB |
| STX #2  | 0xAA 0x09 0x01 0x00 0x0E 0x4C 0x1B 0x01 0xFF |

Sending to requests Firmware version: 0xAA 0x05 0x05 CRC : 0x74 0x93

Response from the modems

| STX #1  | 0xAA 0x0E 0x07 0x00 0x0E 0x4C 0x1F 0x00 0x05 0x0C 0x1E 0x00 0x0C 0x9D |
| STX #2  | 0xAA 0x0E 0x07 0x00 0x0E 0x4C 0x1B 0x00 0x05 0x0C 0x1E 0x00 0xA0 0x8D |

To test if the CRC is correct, you need to change the two last bytes to go through the CRC function.

Error message could look something like this: 0xAA 0x05 0xFF 0xA1 0xCB

This value 0xFF means that the modem answers and corresponds to a non acknowledgment (NAK).

In figure 13, the answer from the modem is shown. The hex code with the red underline indicate that the modem has acknowledged the package and is now going into send mode. The package following, the modem sends a non acknowledgment, highlighted by the hex code with a red underline. The last bytes are when the remaining number of transmission attempts.

0x00 No pending transmission 0xxx Number of remaining transmission attempts

To connect the wired cables directly to the modem and they were folded so they broke off many times. To solder several times on the modem is probably not good for the circuits.
4.3.5 Problems in the programming phase with STX modem

There were many problems during the programming phase, the first problems we encountered were with the CTS and RTS. According to the first documentation the signals at the start should be the inverse (low) and this was incorrect. When connecting the modem we saw that the RTS and CTS was high. Then the problem was to synchronize these signals in the code for the microprocessor.

One additional problem was with the given CRC calculation from the STX2 modem specifications was not correctly implemented in the PIC. This led to that the package sent from the PIC to the modem was incorrect and then the modem did not answer with any packages. This problem was solved by rewriting the code for the CCX compiler.

Original CRC function from the STX2 Data sheet [7].

```
uint16_t crc16(uint16_t crc, uint8_t *ptr, int length) {
    auto uint16_t i;
    while (length --){
        crc = crc ^ (uint16_t) *ptr++;
        for (i=0;i<8;i++){
            if (crc & 0x0001)
                crc = (crc >> 1) ^ 0x8408;
            else
                crc >>= 1;
        }
    }
    return crc;
}
```

The PIC was not able to preform certain tasks like shifting bits and it was also necessary that you had to divide too complex expressions, see in code below.

```
/* Modified CRC function for STX2 modem */
uns16 crc16(uns16 crc, uns8 *ptr, int length) {
    uns16 i;
    while (length != 0){
        length --;
        crc = (crc ^ (uns16) *ptr++);
        for (i=0;i<8;i++){
            if (crc & 0x0001){
                crc = (crc / 2);
                crc = crc ^ 0x8408;
            }
            else
                crc /= 2;
        }
    }
    return crc;
}
```

The first package we received from the STX2 was correct and the ID was also correct. But when the modem starts to try to send the packages to the satellites, the error code busy was returned from every attempt to send another one. We
were able to ask the modem for how many tries it had left, but it still is not able to send the package to the satellites. One possible option is that it could be something wrong with the antenna or the antenna connection.

Sometimes the modem will lock itself without any identifiable reason, it did not answer any response. A restart of the modem will fix the problem.

CHAD can provide power for the tracking system module but not all components and parts can use the electricity provided. The design ensures safe operation of the tracking system.

### 4.3.6 GPS tracking system

This system has several requirements. The primary requirement comes from that the modem and antenna uses a voltage of +3.3V. To easy provide the lower voltage, a voltage regulator will be used. A regulator can give a specified level of voltage although it wastes the difference as heat. The input voltage will be the +5 V line and the modem drain a maximum of 50 mA in active mode and the antenna drains 550 mA at max. Although the drain is relatively high it is motivated to use a regulator to provide the correct voltage. The transmission will be during a short timespan so the loss is acceptable compared to a more efficient but advanced solution. A Low-Dropout (LDO) regulator that meets these requirement is MCP1827 from Microchip. The LDO has a maximum output of 1.5 A and requires 0.12 mA to drive. The LDO also have a Shutdown Control Input, SHDN, so the PIC can control when to give power to the modem and antenna.

### 4.3.7 Changes in guidelines

Due to time restrictions and a wish to prioritizing from the supervisors the creation of schematics had to be laid off as there was not enough time. Instead the RF beacon was to be focused on as it was regarded more critical and less prone for errors.

### 4.3.8 RF beacon

To increase the robustness of the RF beacon service and the redundancy of the whole system, it needs to be constructed as a separate system that can sustain the transmission after the impact. The transceiver, TX1, broadcast a signal with a frequency of 173.250 MHz from an antenna. To keep the RF beacon running after the impact it needs a separate power system that meet the higher demand on durability. As there can not be any transmission before launch but to limit the dependence of the main system the beacon system will have batteries to keep the transmission running for several hours after the impact.

The first idea for activating the beacon was by activating a D-latch in the module from the PIC of the GPS system, see figure 14. Though if the signal from the GPS system is cut off or malfunctioning the beacon would not work and a more independent solution had to be chosen. Next evaluated solution was to prohibit the beacon from the GPS PIC initially which would make the beacon activate after liftoff no matter what happens to the GPS and STX-2 modem. This would could be constructed by using a separate PIC-processor powered by batteries,
see figure 15. The solution was too reliant on the GPS PIC though, and as the project ran out of time, priorities had to be made.

Instead the final version became to use the signals provided by the CHAD to control the beacon and let the beacon change from the power provided from the CHAD system to batteries for the beacon without interrupts. This solution with the power switch was provided by supervisor Adam Wujek. The beacon is controlled by the signals LO (Lift Off), SODS (Start/Stop of Data Storage), and SS1 (Separation Sensor 1) as well as a D-latch. The SODS controls parallel transistors connected to the D-latches two outputs which allows switching between batteries and the power line, see figure 16. Before launch the beacon is switched to the main power line from CHAD. Before liftoff the power switch is controlled by the SODS signal, however after LO and SODS the beacon system remain on battery power even if SODS turns high again as seen by Figure 17.

The RF beacon is turned on by the PIC when the correct state in the runtime has been reached. The circuit diagram from Adam Wujek can be found in the Appendix.

4.4 Designing of the PCB

The foundation for a good PCB design is a good schematic picture of the components that has been simulated to work before the designing of the PCB. Using a free PCB design-tool, Eagle, the schedule is transformed to a PCB scheme.
and the components and connections can be given a logical layout that makes good use of the available size of the board. Because of the limited volume in the module of the REXUS rocket the board have to be as small as possible but still in a form factor that is possible to mount. The first step in creating the board layout is to have all the components. Many of the components is available from the manufacturer’s webpage. The components from Microchip, like the PIC16F690, comes in a Ultra Librarian format, created by Accelerated Design. This format can then be freely converted for Eagle and many other popular programs as Pspice. However, not all components are available from the manufacturer, like the TX1, so a component had to be created for Eagle. In Eagle you can create a symbol for the circuit schedule and then you make a package that will be the footprint on the PCB. These are combined in a device where the connectors of the symbol is matched with the package. This make it possible to use the circuit diagram for the RF beacon the TX1 were created in both an upright and a version laying down against the PCB (and also a RX1, the receiver, to complement it). The satellite modem was created as well but never used in the end. The TX1 package and STX-2 symbol is shown in figure 18 and 19.

Figure 18: Package of TX1

Figure 19: Symbol of STX-2

The second step is to lay all the components in a way that minimizes crossing the electrical connections on the layout. After that it is a time-consuming work to lay, what Eagle has named, the buses that will be etched onto the final board. For this board a two-sided design was used with as many components of Surface Mount Device (SMD) type to optimize the space available and to make it possible to change side using vias if needed to bypass any crossing lines. Still, the complexity is increasing exponentially as the board size decreases.
4.4.1 Changes in guidelines

When the first draft of the design was completed the question was raised if it was possible to decrease the size of the beacon board from 96x90mm to 60x60mm, a decrease with almost 60% of the surface. Many hours were laid on two completely different layout foundations but the complexity expanded until the every new bus demanded rerouting of increasingly more old buses. The task seems possible with more time and skill but the final design is instead some optimizations, minor changes and fixes to the first design draft to eradicate any design error of buses placed too close. See figure 20.

![Figure 20: PCB Design for the RF beacon](image)

4.5 Construction of PCB

To construct a PCB it is common to order it from a company but cheaper to etch it oneself. As the price point when ordering from a company is very high, the first board were etched and the second version would be ordered from a firm. When etching the PCB is constructed using a print out of the design on a transparent sheet. The PCB is then lighted with UV-light to form the tracks and the holes drilled for the pads to mount the components. The construction of the PCB was conducted by Jerker Skogby, supervisor of CHAD, because of earlier experience with constructing PCBs. When the PCB was constructed all components had to be soldered on, a not negligible task with many small components to mount and several vias to connect. The result can be seen in figure 21.
4.6 The PIC code for the RF Beacon

To control the RF beacon to start the transmission there is a PIC that will respond to the input signals from CHAD. After the release from the rocket it waits before to start the beacon. The signals follow a scheme and the programming is centered around a State machine that moves between states as the signals change and activates light emitting diodes (LED) accordingly to know in which stage the program is. Active is digital zero except for SS1 for which digital one is active.

The state machine begins in the Start state and lights a LED to indicate the initialization. When given the SODS signal as the Documentation starts, the state is changed to Battery and another LED is lit. As the launch occurs it enters the Liftoff state and the Liftoff-LED is lit. When the rocket has reached the correct altitude and the experiment detaches from the rocket it enters the Wait state. After 100 seconds the State machine transits to Transmit state, lights the Transmit-LED and starts the TX1 beacon. See figure 22.

The beacon sends its state to CHAD with I/O status with every state change and every second. The package have the following specification:
4.7 Testing of the RF beacon board

The RF beacon was constructed to switch seamless between power from CHAD and power from batteries. When the system is powered on the SODS signal it activates the battery usage. Following the instructions no transmissions can occur before launch so that has to be tested as well. As the experiment can not be rebooted the code in the PIC has to be carefully tested so it follows the designated path.

• Verification that no transmission is performed before the experiment is started. Follow the signal changes and verify that the TX1 is not activated before the Transmit state is entered.

• The complete cycle from the initialization until the transmission of the RF beacon is turned on. Feed the beacon with the signals from CHAD and verify that the battery change is working, that the wait is initiated as expected and that the transmission starts according to specification.

• Runtime of the beacon. From that the transmission begins the beacon should be active for more than 5 hours.

4.7.1 Testing setup

The testing of the RF beacon was performed with a finished CHAD-module. The module was connected in the same way as in the finally assembly. The CHAD module provides the LO, SODS, SS1 signals as well as the power needed for the Beacon.

4.7.2 Power interrupts

The RF beacon was switched on using the CHAD power and measuring the power from the batteries. No power from the batteries were drained. When activating the SODS, the beacon switched to batteries indicated by the led.

Figure 23: Power from battery
The power switch works as expected as indicated by LED27 where the power is provided by batteries, see figure 23. When SODS is deactivated the system falls back on CHAD power and LED26 is lit, see figure 24.

4.7.3 No transmission before liftoff

When the TX1 is switched on it is indicated by LED1. Running the sequence showed that the system behaved as expected with RF completely turned off during CHAD power, see figure 25, and when time to transmit it was turned on, see figure 26.

Figure 24: Power from CHAD

Figure 25: TX1 deactivated

Figure 26: TX1 activated
4.7.4 **PIC code and State machine**

To test that the code and the State machine works as expected, the signals is fed to the RF beacon and output as well as the state of the LEDs controlled by the PIC showed that the sequence was followed through according to specification. See figure 27.

4.7.5 **Battery testing**

Because of shortage of time, the runtime of the batteries could not be tested. However the beacon uses two 9V batteries for redundancy. For testing, regular alkaline batteries were used as the Lithium-ion batteries which has higher energy density were sold out. The energy is in the batteries is higher than those that the experiment SQUID used last year so the runtime should be the same or longer, although not verified.
4.8 Completion

The result of the project is a PCB for the RF Beacon system with a code for the PIC-processor were sent to Scotland for system assembling and testing on the 24th of October. The result can be seen in figure 28. After system testing it will be replaced by professionally constructed board.

![RF Beacon board](image)

Figure 28: RF Beacon board

The board for the GPS and modem is not fully realized because of short timeline and the unexpected events. The results for this is a scheme of how to connect all the parts and listen to the signals on the ports. The code is attached for configuration of the modem and programming of the PIC.
5 Discussion

During this project there has been challenges, and the result as such, that are important to reflect on. If anyone would continue where this project has landed these are the things to think about.

5.1 Areas of improvement

To construct a similar work there are things to consider doing differently and areas where the work can be complemented.

5.1.1 Antenna for the STX2 modem

We did not make the antenna for the modem to work, one possibility that we could have improved are the antenna connection. Buy a new more expensive that might have worked better. It is really hard to radio frequency signals throughout a test breadboard.

5.1.2 Testing

There are several areas the testing can be extended to to improve the testing and also to lower the risk of faults. First the battery should be tested fully to see if it can sustain more than 5 hours of sending with the TX1. If it is necessary the battery type, battery capacity or some fundamental design of the Beacon PCB may have to be changed. Other areas to test is the ability to withstand the temperatures in space but also the cold weather in north part of Sweden. This can also affect the batteries so a temperature testing would have to include all of the equipment. Vibration tests were not possible to conduct due to practical reasons but as the rocket ignites the whole module will be subjected to high stress and it would be better to test beforehand.

5.1.3 Radio transmission

The radio transmission of the RF beacon was never fully tested in this thesis and although the documentation of the TX1 describes the quarter-wave whip as basically a wire connected to the RF out there was not enough time to test the actual transmission before shipping. This is a critical function and should be tested carefully if possible.

5.2 Retrospect

In retrospect, we can reflect on why the work has taken longer than we estimated. The parts took a longer time to ship, and it resulted in some weeks of not being able to program and learn about how the parts work. During this time we had time to write on the feasibility study and set up a functional working space for the programming. As a conclusion from this thesis it would have been better to try to condense all works from the beginning, however, this is difficult when one is not too familiar with all the work that is involved. From our side we have planned poorly, and breaks during the work coupled with poor time management led to a lack of cohesion with our supervisors.

If you work with technology products that are uncommon in conventional electronics stores, it is harder to find a solution to any problems that may arise.
When you need expert help and support from abroad, it is time consuming and difficult to obtain. What you can do is contact people who have worked with it before and there are risks that they will be unresponsive.

Working with the supervisors of CHAD has mostly been helpful as they have tried to patiently assist us if we have needed their knowledge and support.

Sometimes too good as it is not only one time when they have demanded that the solution provided fit requirements not stated before, but obvious to them. This takes more time and can be a little frustrating. The new guidelines for the PCB for example was a diversion from the first direction of "as small as possible" to a goal that in the end weren’t reachable but at the same time time-consuming to pursue.
6 Conclusion

So we did our bachelor thesis on building a tracking system for the Suaineadh satellite experiment. During these 16 weeks we achieved knowledge about how a tracking system is built. How to locate a module with the GPS coordinates and send package data through a satellite modem. We also achieved knowledge on how to program a microprocessor and finally, how to design and build a PCB.

Building the tracking system was a much larger task than we first anticipated and we had no previous experience in this subject. We thought the task of using a GPS and satellite modem sounded exciting and we wanted to increase our understanding of how this worked. It has been exciting but it was also a lot of hard work that we had not accounted for. The consequence of this is that it took longer than we previously expected.

We have gained valuable experience for the future in dealing with all the challenges and obstacles that came with this thesis.

Finally, we hope that the system is going to work and hold for the experiments in space.
7 References

[1] Furoshiki Satellite - A Large Membrane Structure As A Novel Space System  
Shinchi Nakasuka, Takahira Aok, Ichiro Ikeda and Yuichi Tsuda (2001)

25 Jan (2011)

Aug (2010)


A RF beacon

Figure 29: Circuit diagram of the RF beacon
Figure 30: RF Beacon top

Figure 31: RF Beacon bottom
/*
 * STX2 Modem - Testing for Tracking system - Suaineadh
 * Martine Selin 2011–11–01
 */

#include <16F690.h>
#define PACKAGE_LENGTH 14
#define SEND_IDMESSAGE 0
#define GET_CONFIG 1
#define SEND_MESSAGE 2
#define TRIES_LEFT 3
#pragma bit STATELED @ PORTC.2 /* Toggle State LED */

void initserial( void );
void putchar( char );
void delay10( char );
void printf( const char *string, uns8 variable);

void send_message();
void get_configuration();
void set_configuration(int tries, int min_interval, int max_interval);
void get_number_of_tries_left();
void send_package(uns8 *char_ptr, int length);
void send_IDmessage();
uns16 crc16(uns16 crc, uns8 *ptr, int );
void modem_sendbyte(unsigned int);
void initSTX2( void );

void main( void )
{
    /* initialize PIC for correct baudrate 9600 STX2 modem */
    initserial();

    int mode = 1;
    PORTC.0 = 1; /* LED/PICKIT2 for RTS SET HIGH */

    /* Set configurations for the modem */
    set_configuration(5, 12, 30);
    delay10(20);
    while(1){
        /* Test states for the modem */
        switch(mode){
            /* case SEND_IDMESSAGE will ask for the modems ID */
            case SEND_IDMESSAGE:
                STATELED = 1;
                break;
        }
    }
}
delay10(200);  /* 2000 ms delay */

    /* Try to send a ID message */
    send_IDmessage();
    STATELED = 0;
    mode = GET_CONFIG;
    break;

    /* case GET_CONFIG will ask for the modems configurations */
    case GET_CONFIG:

        STATELED = 1;
        delay10(200);  /* 2000 ms delay */

        /* Try to get the modems configuration */
        get_configuration();
        STATELED = 0;
        mode = SEND_MESSAGE;
        break;

    /* case SEND_MESSAGE will send a fixed message */
    case SEND_MESSAGE:

        STATELED = 1;
        delay10(200);  /* 2000 ms delay */

        /* Try to send a fixed message */
        send_message();
        STATELED = 0;
        mode = TRIES_LEFT;
        break;

    /* case TRIES_LEFT ask the modem how many tries it have left */
    case TRIES_LEFT:

        STATELED = 1;
        delay10(200);  /* 2000 ms delay */

        /* Set to GPS to the first mode */
        get_number_of_tries_left();
        STATELED = 0;

        /* Set to GPS to the first mode */
        mode = SEND_IDMESSAGE;
        break;
    } /* End switch */

} /* End main */


FUNCTIONS

unsigned crc16(unsigned crc, unsigned *ptr, int length)
{
    unsigned i;

    while(length != 0){
        length --;
        crc = (crc ^ (unsigned) *ptr++);
        for(i=0;i<8;i++){
            if(crc & 0x0001) {
                crc = (crc / 2);
                crc = crc ^ 0x8408;
            } else {
                crc /= 2 ;
            }
        }
    }

    return crc;
}
Suáíneadh sa tellite experiment A RF bea con return crc;

} return crc;

void send_IDmessage () {

/* Example package:
 0xAA 0x05 0x05 CRC : 0x74 0x93
 0xAA 0x05 0x01 CRC : 0x50 0xD5
*/

char test_package[5];
test_package[0]=0xAA; /* Preamble Fixed pattern 0xAA */
test_package[1]=0x05; /* Message length */
test_package[2]=0x05; /* Ask for ID */
test_package[3]=0x00; /* CRC */
test_package[4]=0x00; /* CRC */

uns16 CrcValue=0;
CrcValue = ~crc16(0xFFFF,test_package,3);
test_package[3]=(uns8) (CrcValue&0xFF);
test_package[4]=(uns8) (CrcValue/256);
send_package(test_package, 5);
}

/* Send message function have a fixed own created message right now for testing, The smallest package you could send is 5 bytes (with Preamble, length, CMD and CRCx2) It is possible to send a datapackage with GPS coordinates up to 144 bytes. */

void send_message () {

char package[14];

package[0]=0xAA; /* Preamble Fixed pattern 0xAA */
package[1]=0x0E; /* Message length */
package[2]=0x00; /* Command: Send the package */
package[3]=0x00; /* Data from 4–12 */
package[4]=0x3D;
package[5]=0xDD;
package[6]=0x85;
package[7]=0xC6;
package[8]=0xC5;
package[9]=0x5D;
package[10]=0x01;
package[11]=0x00;
package[12]=0x00; /* CRC */
package[13]=0x00; /* CRC */

uns16 CrcValue=0;
CrcValue = ~crc16(0xFFFF,package,PACKAGE_LENGTH-2);
package[12]=(uns8) (CrcValue&0xFF); /* Mask values */
package[13]=(uns8) (CrcValue/256); /* Shift 8 at CrcValue/2^8*/
send_package(package, 14);
}

/* Get config will set the ask the modem for the configurations */

void get_configuration () {

char package[5];
package[0]=0xAA;
package[1]=0x05;
package[2]=0x07;
package[3]=0x00;
package[4]=0x00;

uns16 CrcValue = 0;
CrcValue = ~crc16(0xFFFF,package,3);
package[3]=(uns8) (CrcValue&0xFF); /*Maskar v rden */

Martine Selin, Carl Brangesjö 33 November 8, 2011
package[4]=(uns8) (CrcValue/256); /*Shift 8 bytes CrcValue/2^8*/
send_package(package, 5);
}
/* Set config, number of tries and min_interval (Sek*5) and max_interval (Sek*5) */
void set_configuration(int tries, int min_interval, int max_interval) {
    char package[14];
    package[0]=0xAA; /* Preamble Fixed pattern 0xAA */
    package[1]=0x0E; /* Length */
    package[2]=0x06; /* Command */
    package[3]=0x00; /* Unit ID */
    package[4]=0x0E;
    package[5]=0x4C;
    package[6]=0x1F; /* End of Unit ID */
    package[7]=0x00; /* RF */
    package[8]=tries; /* Number of tries */
    package[9]=min_interval; /* Minimum time between intervals */
    package[10]=max_interval; /* Maximum time between intervals */
    package[11]=0x00; /* TX Power */
    package[12]=0x00; /* MSB CRC */
    package[13]=0x00; /* LSB CRC */

    uns16 CrcValue = 0;
    CrcValue = ~crc16(0xFFFF, package, 12);
    package[12]=(uns8) (CrcValue&0xFF); /* Choose the last bytes */
    package[13]=(uns8) (CrcValue/256); /* Shift 8 bits CrcValue/2^8 */
    send_package(package, 14);
}
/* Get number of tries left from the modem */
void get_number_of_tries_left() {
    char package[5];
    package[0]=0xAA; /* Preamble Fixed pattern 0xAA */
    package[1]=0x05; /* Length */
    package[2]=0x04; /* Command */
    package[3]=0x00;
    package[4]=0x00;

    uns16 CrcValue = 0;
    CrcValue = ~crc16(0xFFFF, package, 3);
    package[3]=(uns8) (CrcValue&0xFF);
    package[4]=(uns8) (CrcValue/256);
    send_package(package, 5);
}
/* Send a char array to the modem (Complete package) */
void send_package(uns8 *char_ptr, int length) {
    /* RTS SET LOW */
    PORTC.0 = 0;
    while (PORTA.1 != 0);
    /* Counter for sending 1 packet, or else the modem can lock it self */
    int onepackage = 0;
    /* 20 ms of try to send a package */
    int time;
    for(time = 0; time<20; time++){
        /* OBS!! CHANGE 1 to ZERO FOR SEND!*/
        if(PORTA.1 == 0 && onepackage == 0){
            int i;
for (i=0;i<length;i++){
    modem_sendbyte(*char_ptr);
    char_ptr++;
}

} /* RTS SET HIGH TO INDICATE LAST BYTE */
PORTC.0 = 1;
onepackage = 1;

delay10(100); /* 1000 msec delay */
delay10(200); /* 2000ms delay */

} /* initialise PIC16F690 serialcom port */
void initserial( void )
{
    /* One start bit, one stop bit, 8 data bit, no parity. 9600 Baud. */
    TRISC = 0x00;  // Set All on PORTC as Output
    TRISA = 0x03;  // Input for RA0 and RA1
    ANSEL.0 = 0; /* No AD on RA0 */
    ANSEL.1 = 0; /* No AD on RA1 */
    TXEN = 1; /* transmit enable */
    SYNC = 0; /* asynchronous operation */
    TX9 = 0; /* 8 bit transmission */
    SPEN = 1;
    BRGH = 0; /* settings for 6800 Baud */
    BRG16 = 1; /* @ 4 MHz-clock frequency */
    SPBRG = 25; /* 25 for 9600 */
    CREN = 1; /* Continuous receive */
    RX9 = 0; /* 8 bit reception */
    ANSELH.3 = 0; /* RB5 not AD-input but serial_in */

} /* initialise PIC16F690 serialcom port */
void initSTX2( void )
{
    SPBRG = 25; /* For 9600 baud */
}

void modem_sendbyte(unsigned int TX_BYTE){
    while (!TXIF) ;
    TXREG = TX_BYTE;
}

void putchar( char d_out ) /* sends one char */
{
    while (!TXIF) ; /* wait until previous character transmitted */
    TXREG = d_out;
}

/* Delay function by William Sandqvist */

void delay10( char n)
/*
   Delays a multiple of 10 milliseconds using the TMR0 timer
   Clock : 4 MHz     =>  period T = 0.25 microseconds
   1 IS = 1 Instruction Cycle = 1 microseconds
   error: 0.16 percent. B Knudsen.
*/
{
    char i;

Martine Selin, Carl Brengesjö 35 November 8, 2011
OPTION = 7;
do {
    i = TMR0 + 39; /* 256 microsec * 39 = 10 ms */
    while ( i != TMR0);
}
while ( --n > 0);

// Debugging outprints function by William Sandqvist */
void printf(const char *string, uns8 variable) {
    char i, k, m, a, b;
    for(i = 0 ; ; i++)
    {
        k = string[i];
        if( k == '0') break; // at end of string
        if( k == '%') /* insert variable in string */
        {
            i++;
            k = string[i];
            switch(k)
            {
                case 'd': // %d signed 8 bit
                    if( variable.7 ==1) putchar('-');
                    else putchar('+');
                    if( variable > 128) variable = -variable; // no break!
                    case 'u': // %u unsigned 8 bit
                    a = variable/100;
                    putchar('0'+a); // print 100's
                    b = variable%100;
                    a = b/10;
                    putchar('0'+a); // print 10's
                    a = b%10;
                    putchar('0'+a); // print 1's
                    break;
                case 'b': // %b BINARY 8 bit
                    for( m = 0 ; m < 8 ; m++ )
                    {
                        if( variable.7 ==1) putchar('1');
                        else putchar('0');
                        variable = rl(variable);
                    }
                    break;
                case 'c': // %c 'char'
                    putchar(variable);
                    break;
                case '%':
                    putchar('%');
                    break;
                default: // not implemented
                    putchar('!');
            }
        }
    }
    else putchar(k);
}
/ *  
GPS – Testing for Suaineadh – Tracking system  
Martine Selin  2011–11–01

+5V----Vdd  16F690  Vss----GND  
|RA5     |  RA0|  |RA4     |  RA1|  |RA3     |  RA2|  |RC5     |  RO0|  |RC4     |  RC1|  |RC3     |  RC2|  |RC6     |  RB4|  |RC7     |  RB5/Rx| >GPS Signals in  
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

/*  
* B Knudsen Cc5x C–compiler – not ANSI–C */  
//include "16F690.h"

#pragma config |= 0x00D4  
#define MAX_STRING 23  
#define PACKAGELENGTH 14  
#define DOLLAR '$'  
#define LOCATION_ERROR 1  
#define LOCATION_OK 0  

void initserial( void );  
void initGPS( void );  
void FlushReceiverBuffer( void );  
void putchar( char );  
char getchar( void );  
void delay10( char );  
void printf( const char *string , uns8 variable );  
int check_digit( char c );  
int GetGPSCoordinates( char * );  
uns16 crc16( uns16 crc , uns8 *ptr , int );

void main( void )
{
    /* initialize PIC for correct baudrate 4800 */  
    initserial();  
    delay10(200); /* 2000 ms delay */  
    FlushReceiverBuffer();  
    char input_gps[MAX_STRING]; /* buffer for input string */  
    int mode = 0; /* Start at mode 0, receive string from GPS */

    int locate = 0;  
    int num;  
    int notvalidstring = 0;

    PORTC.0 = 1; /* LED just for check the PIC is still active */  
    delay10(200);  

    while(1)
    {
        /*The program got 2 states, one to get a string from GPS */  
        switch(mode)
        {
            case 0:
                /* GET GPS POSITION */  

                Martine Selin, Carl Brengesjö  37  
November 8, 2011
locate = 0;
int numberofgprstries = 0;

/* Set baud rate to 4800 to communicate with PIC */
/* initGPS(); */
delay10(200); /* 200 ms Delay after init */

/* Try to get a valid GPS String */
while (locate == LOCATION_ERROR || numberofgprstries < 10 ){
    printf("Trying to get GPS data: \n\r", 0);
    numberofgprstries++;

    if( GetGPSCoordinates( &input_gps[0] ) ) {
        printf("Found a $GPS String: \n\r", 0);
        printf(input_gps, 0);
        /* Check if the string is valid and how many '0' in it */
        int zero = 0;
        int validnumbers=0;
        int n;
        for( n=0;n<MAX_STRING;n++){
            char inputnumb = input_gps[n];
            if( check_digit(inputnumb) ){
                validnumbers++;
                if( inputnumb == '0'){
                    zero++;
                }
            }
        } // End forloop
        /* Check if the string from the modem is ok */
        if( zero < 9 && validnumbers>6){
            printf("The string is valid \n\r", 0);
            delay10(200);
            locate = LOCATION_OK;
            mode = 1;
            break;
        }
    } else if ( GetGPSCoordinates( &input_gps[0] ) == 0){
        printf("No_valid_string_from_function \n\r",0);
        notvalidstring++;
        mode = 0;
        delay10(200);
    } // Whileloop

    printf("Number_of_tries_is_over, set_mode_3 t%d\n\r", numberofgprstries);
    mode = 3;
    break;
}

case 1:
delay10(255);
//send_message();
//get_number_of_tries_left();

char createpackage[14];
printf("Send_package_to_modem: \n\r",0);
printf(input_gps, 0);
createpackage[0]=0xAA; /* Preamble Fixed pattern 0xA0 */
createpackage[1]=0x0E; /* Message length */
createpackage[2]=0x00;  /* Message send */
int i;
char c;
for( i=0;i<12;i++){
    if(input_gps[i] != ','){
        c = input_gps[i];
        createpackage[2+i] = c;
    }
}
/*
 printf("To whole package is:\n\r",0);
 printf(createpackage,0);
*/
// printf(input_gps[21],0);

// uns16 CrcValue=0;
// CrcValue = ~crc16(0xFFFF,input_gps,21);
// input_gps[22]=(uns8) (CrcValue&0xFF);
// input_gps[23]=(uns8) (CrcValue / 256);

delay10(200);
/* Set to GPS next mode */
mode = 0;
break;

case 3:
    printf("Send error message to modem:\n\r",0);
    //send_errormessage();
    /* Set to GPS mode again */
    mode = 0;
    break;
} /*Switchcase over*/
}

/* *************************************************** */
/* FUNCTIONS */
/* *************************************************** */

/* Modified CRC function for STX2 modem */
uns16 crc16(uns16 crc, uns8 *ptr, int length)
{
    uns16 i;
    while(length != 0){
        length--;
        crc = (crc ^ *(uns16) *ptr++);
        for(i=0;i<8;i++){
            if(crc & 0x0001){
                crc = (crc / 2);
                crc =crc ^ 0x8408;
            }
            else
                crc /=2 ;
        }
    }
    return crc;
}

/* initialise PIC16F690 serialcom port */
void initserial( void )
{
    /* One start bit, one stop bit, 8 data bit, no parity, 9600 Baud. */
Suaineadh sa tellite experiment A RF bea con

/* One start bit, one stop bit, 8 data bit, no parity. 9600 Baud. */

TRISC = 0x00; // Set All on PORTC as Output
TRISA = 0x03; // Input for RA0 and RA1

ANSEL = 0; // No AD on RA0
ANSEL = 0; // No AD on RA1

// TRISA.0 = 0; /* RTS output */
// TRISA.1 = 1; /* CTS input */

TXEN = 1; /* transmit enable */
SYNC = 0; /* asynchronous operation */
TX9 = 0; /* 8 bit transmission */
SPEN = 1;

BRGH = 0; /* settings for 6800 Baud */
BRG16 = 1; /* @ 4 MHz-clock frequency */
SPBRG = 51; /* For 48 baud, 25 for 9600 */

CHEN = 1; /* Continuous receive */
RX9 = 0; /* 8 bit reception */
ANSELH.3 = 0; /* RB5 not AD-input but serial_in */

} /* initialise PIC16F690 serialcom port */
void initGPS( void ) {
    SPBRG = 51; /* For 4800 baud */
}

int check_digit(char c) {
    if ((c >= '0') || (c <= '9'))
        return 1;
    return 0;
}

} /* Get GPS Coordinates picks out the GPPGA string */
int GetGPSCoordinates( char * string ) {
    char charCount, r, c, g, s;
    int i, j;
    FlushReceiverBuffer();
    r = getchar();
    if(r == DOLLAR){
        for( i=0; i<5; i++ ){
            g = getchar();
        }
    /* Check for GPPGA String */
    if(g == 'A'){
        s = getchar();
        /* Skip the ',' character */
        do{g = getchar();} while(g != ',');
        /* Save latitude coordinates */
        for( charCount = 0; ; charCount++ ){
            c = getchar();
            string[charCount] = c;
        /* Store character */
        if( charCount == (MAX_STRING) )
            /* end of input and 'E' character */
            return LOCATION_ERROR;
        else if(c=='E' || c=='W ')
            string[charCount] = c;
        /* Store characters */
        string[charCount+1] = '\0';
        /* add "end of string"
        }*/
    }
}
return LOCATION_OK;
}
}
return LOCATION_ERROR;

void FlushReceiverBuffer (void) {
    char trash;
    trash = RCREG; /* the two char's that locked the receiver */
    trash = RCREG; /* are read and ignored */
    CREN = 0; /* the unlock procedure ... */
    CREN = 1;
}

char getchar (void) /* receives one char */ {
    char d_in;
    while (RCIF == 0); /* wait for char */
    d_in = RCREG;
    return d_in;
}

void putchar (char d_out) /* sends one char */ {
    while (!TXIF); /* wait until previous character transmitted */
    TXREG = d_out;
}

/* Delay function by William Sandqvist */
void delay10 (char n) /*
    Delays a multiple of 10 milliseconds using the TMR0 timer

    Clock: 4 MHz => period T = 0.25 microseconds
    1 IS = 1 Instruction Cycle = 1 microsecond
    error: 0.16 percent. B Knudsen. */
{
    char i;

    OPTION = 7;
    do {
        i = TMR0 + 39; /* 256 microsec * 39 = 10 ms */
        while (i != TMR0);
    } while (--; n > 0);
}

/* Debugging outprints function by William Sandqvist */
void printf (const char *string, uns8 variable) {
    char i, k, m, a, b;
    for (i = 0; ; i++) {
        k = string[i];
        if (k == '\0') break; // at end of string
        if (k == '%') // insert variable in string
            { i++; k = string[i];
                switch (k)
                { case 'd': // %d signed 8bit

    Martine Selin, Carl Brengesjö 41 November 8, 2011
if ( variable == 11) putchar('-');
else putchar('_');
if ( variable > 128) variable = -variable; // no break!
case 'u': // %u unsigned 8bit
    a = variable / 100;
    putchar('0'+a); // print 100's
    b = variable % 100;
    a = b / 10;
    putchar('0'+a); // print 10's
    a = b % 10;
    putchar('0'+a); // print 1's
    break;
case 'b': // %b BINARY 8bit
    for ( m = 0 ; m < 8 ; m++ )
        if ( variable == 1) putchar('1');
        else putchar('0');
        variable = rl(variable);
    break;
case 'c': // %c 'char'
    putchar(variable);
    break;
case '%':
    putchar('%');
    break;
default: // not implemented
    putchar('!');
}
else putchar(k);
}
Code to test if the timer interrupt is working as expected

RF Beacon – Suaineadh tracking system

/*
* B Knudsen Cc5x C-compiler – not ANSI-C */

#include "int16CXX.h"
#include "16F690.h"
#include "pic.h"

#pragma origin 4

#pragma config |= 0x00D4
#define HIGH 1
#define LOW 0
#define ACTIVE 0 // signal active, active by low signal
#define INACT 1 // signal inactive
#define TIMER 200

/ * N A M E S F O R T H E I N P U T S F O R P I C * /
#pragma bit SeparationSignal @ PORTA.5 /* INVERTED */
#pragma bit LiftOffLatch @ PORTC.4
#pragma bit SODS @ PORTC.7 /* SODS = Start of Data Storage */
#pragma bit TX @ PORTB.7 /* TX to send status package */

/ * N A M E S F O R T H E O U T P U T S * /
#pragma bit SignalLogic1 @ PORTA.4
#pragma bit SignalLogic2 @ PORTB.4
#pragma bit RX @ PORTB.5 /* RX to receive status messages if needed to implement */

/ * N A M E S F O R T H E L E D S * /
#pragma bit INTILED @ PORTB.6 /* INIT LED Light for PIC, LEDs */
#pragma bit LIFTOFFLED6 @ PORTC.0 /* LED For Lift off, LED6 */
#pragma bit SODSLED7 @ PORTC.1 /* SODS LED to check the data, LED7 */
#pragma bit RELLED8 @ PORTC.2 /* ReleaseLED from CHAD, LEDs */

#pragma shared Allocation

// states
#define STATE_UNKNOWN 0
#define STATE_START 1
#define STATE_TEST 2
#define STATE_LIFTOFF 3
#define STATE_BATTERY 4
#define STATE_RELEASE 5
#define STATE_WAIT 6
#define STATE_TRANSMIT 7
#define STATE_MAX 8 // maximum number of state
Suaineadh sa tellite experiment A RF bea con

# define TX_BUF_LEN 15
# define DeviceID 0x03
# define EndByte 0x
# define PayloadSize 4
# define TIMER_WAIT 100

// functions
void initPic( void );
void init_timer0(void);
void delayIO( char );
void state_machine(void);
void send_status(void);
bit get_LO(void);
bit get_SODS(void);
bit get_SS1(void);
void set_SignalLogic1(bit value);
void set_SignalLogic2(bit value);
void set_INITLED(bit value);
void set_LIFTOFFLED6(bit value);
void set_SODSLED7(bit value);
void set_RELLED8(bit value);
void change_state(uint8 new_state);
void state_machine_timer_reset(void);
void state_machine_timer_increase(void);
bit state_machine_timer_is_elapsed(uint16 comp_val);
interrupt Timer0_IRS(void);
uint8 calc_checksum(uint8 *data, uint8 length);

// Variables
uint8 state_cur; // current status
char TX_buf[TX_BUF_LEN];
uint8 uart_tx_len; // length of packet
uint8 tmpu8;
uint8 tmp_timer;
uint16 timer;
uint16 state_machine_timer;

void interrupt Timer0_IRS(void) {
    // GIE = 0; not necessary
    // W, STATUS (and PCLATH if required)
    char sv_FSR = FSR; // save FSR if required
    static uint8 tx_pointer;
    if (TOIE & & TOIF) // Timer0 interrupt timer enabled and overflowed
        { // Timer0 interrupt timer enabled and overflowed
            TOIF = 0; // Clear flag
            tmp_timer = tmp_timer + 1;
            if(tmp_timer > 17){ // Every 1 second
                tmp_timer = 0;
                // send status and increase the state machine timer
                timer = timer + 1;
                /*if(INITLED){
                    set_INITLED(LOW);
                } else{
                    set_INITLED(HIGH);
                }*/
                send_status(); // Time to send status
                state_machine_timer_increase(); // and increase the state machine timer
            }
        } else{
            if(TXIF & & TXIE){ // UART TX interrupt
                TXREG=TX_buf[tx_pointer];
            }
        }
}
tx_pointer++;  
if (tx_pointer==uart_tx_len)  
    tx_pointer=0;  
    TXIE=0; // disable the TX interrupt
}

FSR = rsv_FSR;  // restore FSR if saved
int_restore_registers // W, STATUS (and PCLATH if required)
/
NOTE: GIE is AUTOMATICALLY cleared on interrupt entry and set to 1 on exit (by RETIE). Setting GIE to 1 inside the interrupt service routine will cause nested interrupts if an interrupt is pending. Too deep nesting may crash the program!
/  
GIE = 1;
}

bit get_LO(void) {
    return LiftOFFLatch;  
// Returns the value of the Liftoff from the latch
}

bit get_SODS(void) {
    return SODS;
}

bit get_SS1(void) {
    return SeparationSignal;
}

void set_SignalLogic1(bit value) {  
    SignalLogic1 = value;
}

void set_SignalLogic2(bit value) {  
    SignalLogic2 = value;
}

void set_INITLED(bit value) {  
    INITLED = value;
}

void set_LIFTOFFLED6(bit value) {  
    LIFTOFFLED6 = value;
}

void set_SODSLED7(bit value) {  
    SODSLED7 = value;
}

void set_RELLED8(bit value) {  
    RELLED8 = value;
}

void change_state(uint8 new_state) {
    state_cur=new_state;  
// send_status();
}

void state_machine_timer_reset(void) {
    state_machine_timer=0;
}

void state_machine_timer_increase(void) {
    state_machine_timer+=;
}
bit state_machine_timer_is_elapsed(u8 comp_val) {
    if (state_machine_timer >= comp_val) { return 1; } else { return 0; }
}

// TX send, modified code from Adam Wujek
void send_status(void) {
    // Send I/O status and state over serial TX
    // TX_buf[] = 0;
    u8 io_tmp;
    // First four bytes
    TX_buf[0] = 'A';
    TX_buf[1] = 'A';
    TX_buf[2] = 'A';
    TX_buf[3] = 'A';
    // Size and ID
    TX_buf[4] = PayloadSize;
    TX_buf[5] = DeviceID;
    // Time
    tmpu8 = state_machine_timer >> 8;
    TX_buf[6] = tmpu8;
    TX_buf[7] = state_machine_timer & 0xFF;
    TX_buf[8] = tmp_time;
    // Current internal state
    TX_buf[9] = state_cur;
    // I/O values in Payload, 8 bit
    tmpu8 = (u8) get_LO();
    io_tmp = tmpu8;
    // LO
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) get_SODS();
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) get_SS1();
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) Signal1;
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) Signal2;
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) LED6;
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) LED7;
    io_tmp = io_tmp << 1;
    tmpu8 = (u8) LED8;
    io_tmp = io_tmp << 1;
    TX_buf[10] = io_tmp;
    TX_buf[11] = 0;
    TX_buf[12] = 0;
    // Checksum
    TX_buf[TX_BUF_LEN-2] = calc_checksum(TX_buf, TX_BUF_LEN-2);
    TX_buf[TX_BUF_LEN-1] = EndByte; // endbyte
}

void main(void) {
    /* Initialisation of PICF690 */
    initPIC();
    /*Set all the LEDs to low on port RC.0 - RC.1 */
    INITLED = LOW;
}

Martine Selin, Carl Brengesjö
46
November 8, 2011
LiftonLED6 = LOW;
SODSLED7 = LOW;
RELLED8 = LOW;

/* Set the outputs SignalLogic (RA4 and RB1) to LOW */
SignalLogic1 = LOW;
SignalLogic2 = LOW;
timer = 0;
int tTimer = 0;
init_timer0();
INITLED = HIGH; // Ready
while(1){
    state_machine();
}

/* ************************************************** */
/* FUNCTIONS */
/* ************************************************** */

// Calculate the checksum for the TX package, code from Adam Wujek
uns8 calc_checksum(uns8 *data, uns8 length){
    //return *data;
    //calculate checksum of Basic Packet format
    uns8 i;
    uns16 temp_value=0;
    for (i=0;i<length;i++){
        //length-1 is index of last message byte,
        //don't calculate for last two bytes (checksum and end_byte)
        temp_value+=data[i];
    }
    return (uns8)(temp_value&0xff);
}

/* The State machine

| First state | <--> | Lift off state | <--> | Battery state | <--> |

|<--> | Release state | <--> | Wait state (300s) | <--> | Transmit state |

Input signals; LO = Lift off;
SODS = Start of Documentation; SS1 = Separation signal 1
Output signals; SignalLogic1 – 2
Output LED; INITLED, LOLED, SODSLED, SS1LED
*/

// The State machine, modified code from Adam Wujek
void state_machine(void){
    if(state_cur>STATE_MAX){state_cur=STATE_UNKNOWN}; state_cur=STATE_START;
    // fix wrong status

    switch(state_cur){
    // First state
    case STATE_START:
        if (get_SODS()==ACTIVE){
            set_SODSLED7(HIGH);
            change_state(STATE_BATTERY);
        }
        break;

    // When SODS active, Light up SODSLED
    case STATE_BATTERY:

}
if (get_SODS() == INACT) {
    set_SODSLED7(LOW);
    change_state(STATE_START);
}
else if (get_LO() == ACTIVE) {
    set_LIFTOFFLED6(HIGH);
    change_state(STATE_LIFTOFF);
} break;

// When LO is active, Light up LOLED
	case STATE_LIFTOFF:
    if (get_LO() == INACT) {
        set_LIFTOFFLED6(LOW);
        change_state(STATE_BATTERY);
    }
    else if (get_SS1() == ACTIVE) {
        set_RELLED8(HIGH);
        state_machine_timer_reset();
        change_state(STATE_WAIT);
    }
    break;

// Wait before transmission
	case STATE_WAIT:
    tmp8 = state_machine_timer_is_elapsed(TIMER_WAIT);
    // due to limitations in compiler
    if (get_SS1() == INACT) {
        set_RELLED8(LOW);
        change_state(STATE_LIFTOFF);
        state_machine_timer_reset();
    }
    else if (tmp8) {
        change_state(STATE_TRANSMIT);
        set_SignalLogic1(ACTIVE);
        set_SignalLogic2(ACTIVE);
    }
    break;

// Start transmission of the beacon
	case STATE_TRANSMIT:
    if (get_SS1() == INACT) {
        set_SignalLogic1(INACT);
        set_SignalLogic2(INACT);
        change_state(STATE_LIFTOFF);
        // state_machine_timer_reset();
    }
    break;
    default:
    break;
} // end switch

} // end state_machine

/* initialise PIC16F690 */
void initPIC( void ) {

    ANSEL = 0; // Set PORT to Digital I/O
    ANSELH = 0; // Set PORTS as Digital I/O

    TRISA.5 = 1;
    TRISA.4 = 0;

    TRISB.4 = 0; // Output to logic */
    TRISB.6 = 0; // Output to Initilize LEDPIC */
SUAINAHD satellite experiment

TRISC = 0b11111000; /* RC0,RC1,RC2 out, RC3,RC4,RC5, RO6, RC7 in */

TXEN = 1; /* transmit enable */
TXIE = 0; /* disable TX int */
SYNC = 0; /* asynchronous operation */
TX9 = 0; /* 8 bit transmission */
SPEN = 1;

RXEN = 1; /* settings for 4800 Baud */
BRG16 = 1; /* 0 4 MHz-clock frequency */
SPBRG = 8; /* 25 = 9600 Baud // 51=4800 Baud // 18 = 115200 */

void init_timer0(void){
    PSA = 0; // prescaler assigned to Timer 0, check also WTG for it
    PS0 = 1; // timer 0 prescaler to 1:256
    PS1 = 1;
    PS2 = 1;
    T0CS = 0; // internal clock
    TMRO = 0;
    T0IE = 1; // enable the interrupt
    GIE = 1;
}