Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

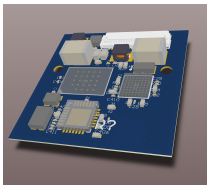
Distributed Computing

Building the Next Generation of GPS Receivers 2.0

Group Project

Ferdinand von Hagen, Robin Rump
vhagenf@student.ethz.ch, rrump@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich



Supervisors:
Pascal Bissig, Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

November 30, 2016

Contents

1 Introduction **1**

2 Requirements **2**

3 Overview of Redesigned Board **4**

3.1 Processor 4

3.2 Power Management 5

3.3 Storage 5

3.4 GPS receiver 6

3.5 Solving Power Issues 6

4 Programming **8**

5 Evaluation and Testing **9**

5.1 Testing the PARC 9

5.2 Power analysis 9

6 Outlook **12**

Bibliography **13**

A Appendix Chapter **A-1**

A.1 Configure the GPS chip A-1

A.1.1 Read sample data from flash A-1

A.1.2 Erase the flash A-1

A.2 PCB files A-2

Chapter 1

Introduction

This project is based on the semester thesis "Building the Next Generation of GPS receivers" by David Etienne[2]. His task was to build a GPS signal sampling device which samples and stores raw GPS data to be later used to post-compute the GPS-coordinates on a backend computer using Manuel Eichelberger's Exhaustive Shift Matching (ESM) algorithm.[1]

We carefully analyzed the issues found by David Etienne in the first design regarding the sampling of the GPS front end data and its power supply. Then we tried to solve the identified issues. In this work, we follow the same ideas, but improve on circuit board size and power consumption. In addition to these improvements, the requirements slightly changed as improvements in the localization algorithm allow for less accurate time synchronization.

Solutions to the issues found are proposed and evaluated. We will discuss how we chose our components, and which requirements could be met.

Chapter 2

Requirements

Our main goal and our requirements are very similar to the original ones outlined in David Etiennes' semester thesis.
Repeatedly sample raw GPS data with a sample rate of at least 8 MHz over a time period of up to 2 milliseconds.

However, building upon his research, the requirements for the project are as follows.

Requirement 1. Selecting a processor that is capable of recording GPS samples at the required sampling rate of 8 MHz

As it will be explained in the processor section, the previously used processor was not capable of sampling the data at 8MHz. We needed to chose a new core unit with a different approach / interface, in order to be able to capture the data at the desired sample frequency.

Requirement 2. Analyse peripherals (such as flash storage etc.) with regard to size and power consumption and replace if necessary

The SD card's power consumption and size needs to be analysed. The power supply and power management has to be adopted for the new hardware.

Requirement 3. Reduce PCB footprint by designing a micro-scale PCB with the desired changes (new processor, new GPS interface, new power supply)

The new board should showcase the possible size of such a tracking device and should meet a size that would make it possible to attach the board to a bird. Considering the size of the components we aim for a size around 2cm x 2cm.

Requirement 4. Write a new firmware to perform all tasks such as the sampling and time keeping, connection between the USB bus and the storage device

The software has to be adopted to the new processor and the hardware changes have to be taken care of in software.

Requirement 5. Evaluation of power consumption

After completition of the PCB and software, we need to make sure, that our goals regarding power consumption and functionality are met.

Chapter 3

Overview of Redesigned Board

3.1 Processor

Replacing the XMS microcontroller was the first and most difficult requirement to fulfill. David Etienne used a quite new and low power CPU with a core frequency of 48MHz to receive the 4-bit parallel data stream from the GPS front end. Due to limitations of the DMA on the chip, he was unable to receive samples with the desired 8 MHz sample frequency. During our research, we ran into similar issues. It was almost impossible to find adequate data about the DMA speeds on the processors we looked at. As most of the low power processors share memory lanes, DMA data transfers are dependent on the traffic on the memory lanes caused by CPU activity and the delays due to CPU activity or synchronization with the GPS frontend could easily add up to at least 8 clock cycles or more per transfer. Hence, using these DMAs to sample GPS data at 8 MHz would require processors with a core frequency above 64 MHz. The necessary synchronization between the DMA transfer and GPS frontend would further complicate this approach. A solution entirely based on the DMA transfer from ports to memory was therefore not feasible.

Our next idea was to use four integrated SPI (serial peripheral interface) modules on a processor to read the data and process it. That would have left us with enough time to transfer the data using the core's DMA. Unfortunately, CPUs with at least 5 SPI Interfaces (4 for the GPS sampling and one for data transfer from/to the memory) are rather rare. Also, we would have been forced to find a way to synchronize the four SPI modules in order to be able to reconstruct the samples.

We decided to look for something entirely new. The solution implemented in this group project is based on an ATMEL SAM4L processor[4]. This is a member of a family of flash micro controllers based on the high performance 32-bit ARM Cortex-M4 RISC processor which runs at frequencies up to 48MHz. It

features a parallel capture module (PARC) capable of sampling data from an 8-bit parallel bus operating up to 24 MHz. The PARC can simultaneously push the 4 GPS sample bits into a buffer within the memory of the core. In our tests we were able to reliably process data at 20MHz, far above the required 8MHz sample frequency.

3.2 Power Management

Our system needs to be able to be powered from a small battery. The most commonly used power source for micro-scale, low power systems are coin cells. Our initial research was entirely based on the CR2032, an commonly used $3V$ Lithium Coin Cell with $230mAh$ charge[7][8]. However, when we first drafted the sketch for the new PCB, it became apparent that the coin cell would be larger than the PCB. After doing power calculations (which will be discussed very soon), we decided to switch to the CR1632, which has a lower capacity of $140mAh$, but also is 4mm smaller in diameter.

In order to be able to power the system based on a coin cell, the system power consumption when sleeping and peak power consumption when active had to be limited. As the power consumption of the processor and GPS front end could not be lowered, we had to reduce the current consumption of the remaining two components of the system, data storage and power management.

3.3 Storage

According to various articles [3] [10] [9], the current consumption and power-on cycles of SD cards are pretty unpredictable. Moreover, a SD card can easily consume up to $100mA$ in order to power the internal charge pumps of the SD card. Finally, the size of a normal microSD card reader to be soldered onto a PCB is about the same size as our final design. We needed to find a new solution, which allows storing enough data, while taking up less space than a SD card solution while using as little current as possible. Coin cells have a rather high internal impedance of at least 20Ω [8]. Hence drawing high currents require large backup capacitors to keep the supply voltage stable which in turn increases current consumption due to the leakage current of the capacitors and the size of the PCB. This makes SD cards an unfavourable choice for this type of application.

We were looking for a low power flash chip with a capacity of at least 66MB. This is based on the assumption, that we want to sample every hour for one year: $24 * 356 * 8kB = 66MB$. We selected the Micron N25Q00A[6]. It comes

with a standard SPI bus, does not require an additional charge pump and has a maximum operating current consumption of $20mA$. At 1 GBit storage, it actually allows us to store almost two years of data.

3.4 GPS receiver

The market for commercial, small, low-power GPS chips with raw data output is small. The MAX2769[5] has already been used in the Hermes v1 project, and we continued using it for this project. There are few other options available which all provided similar or worse performance without having any additional features.

3.5 Solving Power Issues

After all chips had been selected we made a first power consumption estimation. Based on the datasheets, we estimated that every sample would wakeup the system for about $25ms$ ($4ms$ for processor initialization, $5ms$ for the MAX startup and sample time, $0.5ms * 16$ for write access). This accumulates to a total active time of the system in two years of $(24 * 356 * 2 * 25ms)$ $427.2s = 7.12$ minutes. We calculated, that in the worst case scenario the system would draw $100mA$. This is equivalent to $(7.12/60 * 100mA) = 11.86mAh$ of consumed charge. Coin cells degrade over time, so to account for degradation, based on half of the nominal coin cell capacity, our standby current would have to be lower than $((70mAh - 12mAh)/(24 * 356 * 2h)) = 3.4\mu A$ for two years of lifetime.

The processor requires $1.5\mu A$ in backup mode while the real time clock (RTC) remains active. This gives us another $1.9\mu A$ to be available for various leakage currents or other devices. Unfortunately, neither the memory nor the MAX2769 have a proper standby mode. Also, as mentioned above, the coin cells voltage decreases over time. As a consequence, we had to add a power management chip. We decided to use the AMS AS1310 as it would cut off the GPS front end and memory chip completely during standby mode and use less than $100nA$ itself. It also has an integrated buck-boost converter to provide a stable $3V$ power supply for these chips, even when the coin cell degrades and does not provide $3V$.

In order to be able to make the PCB as small as possible, all components where designed with the smallest footprints available. This allowed us to integrate all components onto a PCB with an area of 23.4mm x 18.8mm including the GPS antenna and coin cell holder. Along with it, we designed a debugging / interface chip. It can be connected through a flat band cable. On it, you find male

headers to debug the system using a logic analyzer as well as a mini USB port to configure the device and read the sample data with a computer (see appendix). The weight of the system is as low as 4.9g (board: 3.1g, coin cell: 1.8g, flat band cable: 0.1g).

Chapter 4

Programming

We sample the data from the MAX2769 onto the flash chip, using a combination of PARC and SPI. PARC, the parallel input capture, allows the processor to transfer up to 8 bits (although we will only be using 4) directly into a buffer within the processor's memory (via the DMA). Once completed, we process the data and transfer it to the processor's memory through the SPI bus.

At the very beginning of our program, we check for a positive power supply from the computer. If a computer is to be found attached, the chip switches into what we call the "USB mode". The chip uses the standard USB mass storage device protocol. In order to avoid overhead, all data is stored binary, without a filesystem. Therefore, you can copy the contents of the flash chip onto a computer using a UNIX "dd" command. It will take somewhere between 2.5-5 minutes to copy the data onto the PC.

Chapter 5

Evaluation and Testing

5.1 Testing the PARC

As we could not properly analyze the received GPS data (as the data has no specific, predictable structure), we had to use other methods to verify that our data capturing works. During our initial tests on an evaluation kit of the processor (SAM4L8 Xplained Pro), we used a Raspberry Pi to generate a known batch of sample data. We would then capture it with the PARC interface. The core had no problem to sample data as fast as 20MHz without any data errors.

5.2 Power analysis

In order to make sure our finished device would agree with our calculations made during its design, we had to run a power analysis. We loaded the software onto the chip and had it sample a couple of times and then attach it to a power analyzer and see its current usage during regular operation.

On Figures 5.2 and 5.1, you can see both the digital signal of the chip over one sample along with its current consumption. You can clearly differentiate between six phases.

1. The processor starts the core voltage regulator and stabilizes it's clock. This may take a while, since we send it to the lowest sleep mode (backup mode) after each sample.

2. The processor has booted and has activated the power domain through the AS1310 power management IC for the memory and GPS front end.

3. The GPS front end is initialized, waits for the clock to stabilize and samples $2ms$ worth of data.

4. The data in the buffer is transformed into a binary format suitable for the flash. The MAX2769 is shut down; the flash initialized.

Figure 5.1: Logic analyzer output for one sample cycle

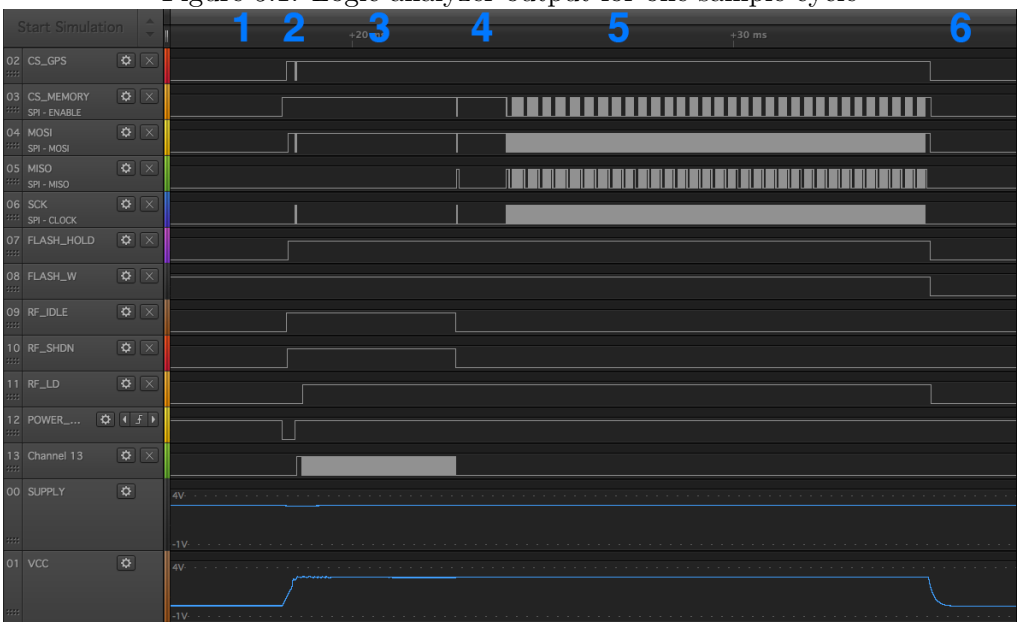


Figure 5.2: Power analysis of one sample cycle

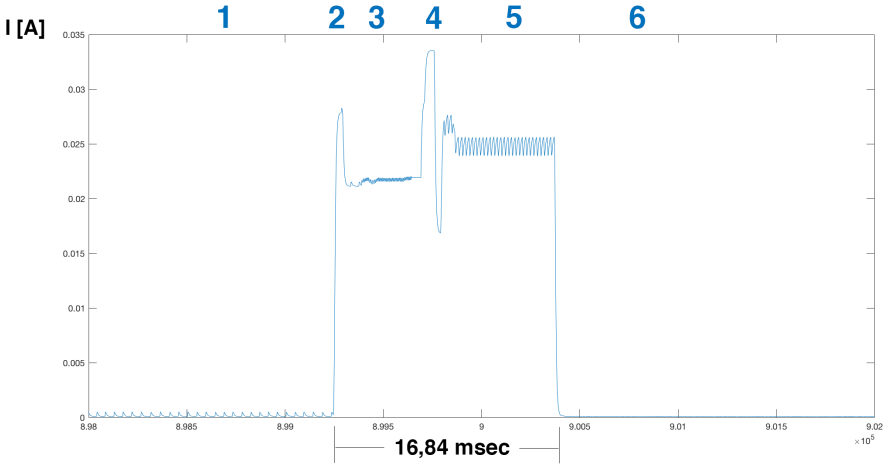
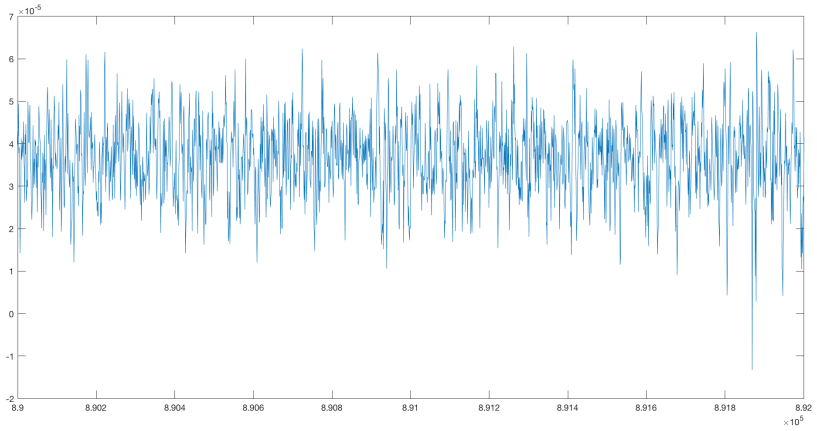


Figure 5.3: Power analysis of standby current



5. 8kB worth of sampled data is transferred to the flash through 32 page programs at 256 bytes each (note the 32 little peaks on the image).

6. The AS1310 cuts the power to all peripheral devices and the processor enters backup power mode. Power consumption drops to $50\mu A$.

As we can see very well the active power consumption with around 25mA is very well within our expected range. Our active time at 16.84ms is lower than our initial estimates. Unfortunately, the standby power consumption is hovering around 50uA, which is 10 times higher than the expected value and makes one or two years lifetime infeasible. The sampled data was later exported for analysis. More testing needs to be done and the settings of the GPS front end must be adjusted to be able to analyze the GPS performance.

Chapter 6

Outlook

The processor, flash and other components selected during this group project turned out to be perfectly suited for this type of application. We were able to achieve an active power consumption that is well within the range of a coin cell powered application and a lot lower than our worst case estimation. However, the standby current is way higher than expected. We were unable to identify the exact reason for the high current, but we expect it to be a combination of a minor bug in the PCB design regarding the resistors, a software bug and measurement inaccuracy. It seems that something prevents the processor from disabling all peripheral ports during backup power mode which leads to a higher power consumption.

Unfortunately, we do not have a test board where we could test the processor without any connected devices. It would be very important to be able to test all functions of the processor without any connected devices like debug adapters to be able to debug the low power modes of the processor.
The next important step would be to find the best settings for the GPS front end and check the antenna design for maximum performance. The measured active power consumption leaves room for other improvements too. Smaller buffer capacitors could be used and would further decrease the standby power consumption and would therefore allow smaller coin cells. Even smaller PCBs could then be achieved. During development the adapter board used for debugging and programming turned out to be extremely helpful. A more sophisticated adapter board with more debugging options and added time synchronization would be good to have.

Bibliography

[1] M. Eichelberger. Rethinking GPS Localisation. Master's thesis, ETH Zurich, Switzerland, 2015.

[2] D. Etienne. Building the Next Generation of GPS Receivers. Semester thesis, ETH Zurich, Switzerland, 2016.

[3] TI: Coin cells and peak current draw. http://www.ti.com/lit/wp/swra349/swra349.pdf. Last accessed: 30.11.2016.

[4] Atmel SAM4L Datasheet. http://www.atmel.com/Images/Atmel-42023-ARM-Microcontroller-ATSAM4L-Low-Power-LCD_Datasheet.pdf. Last accessed: 30.11.2016

[5] MAX2769 Datasheet. https://datasheets.maximintegrated.com/en/ds/MAX2769.pdf. Last accessed: 30.11.2016.

[6] Micron Flash Datasheet. https://www.micron.com/~/media/documents/products/data-sheet/nor-flash/serial-nor/n25q/n25q_1gb_1_8v_65nm.pdf. Last accessed: 30.11.2016.

[7] CR2032a Datasheet. https://cdn-shop.adafruit.com/datasheets/maxell_cr2032_datasheet.pdf. Last accessed: 30.11.2016.

[8] CR2032b Datasheet. http://data.energizer.com/PDFs/cr2032.pdf. Last accessed: 30.11.2016.

[9] MicronSD Datasheet. http://www.farnell.com/datasheets/1836582.pdf. Last accessed: 30.11.2016.

[10] KingstonSD Datasheet. http://www.electronics123.net/amazon/datasheet/Kingston%20Micro-SD%20Specification.pdf. Last accessed: 30.11.2016.

Appendix A

Appendix Chapter

A.1 Configure the GPS chip

The GPS chip can be dynamically configured, without recompiling its software. When connected via USB to a computer, you will see with two USB drives: "GPS DATA" and "GPS CONFIG". Using UNIX "dd" on "GPS CONFIG", you can read and write your configuration (such as interval between samples etc.) directly into the permanent flash of the processor on the chip. The flash drives do not have a file system. The first 64 bytes of the "GPS CONFIG" are the actual config bytes; followed by 4032 unused do not care bytes (USB requires us to have a minimum of 8 sectors, 512 bytes each). Table A.1 on page A-9 explains the order and defaults of the configuration file.

```
/* Read from chip */
sudo dd if=/dev/diskX of=config_read.img bs=512
/* Write to chip */
sudo dd if=config_write.img of=/dev/diskX bs=512
```

A.1.1 Read sample data from flash

As you might have guessed, the data is located on the second USB drive, "GPS DATA". This is a readonly USB drive (see A.1.2 for erasing). You can read it using the UNIX dd command mentioned in A.1. The binary image consists of continuous *8KB samples*. Within one sample there are 16384 data points representing the data received from the MAX2769 bits. The order of the bits is is *Q1Q0I0I1* with I1 being the LSB.
Once you hit a continuous stream of *0xFFFFFF*, you will know you have reached the part of the flash that has not been programmed / reached yet.

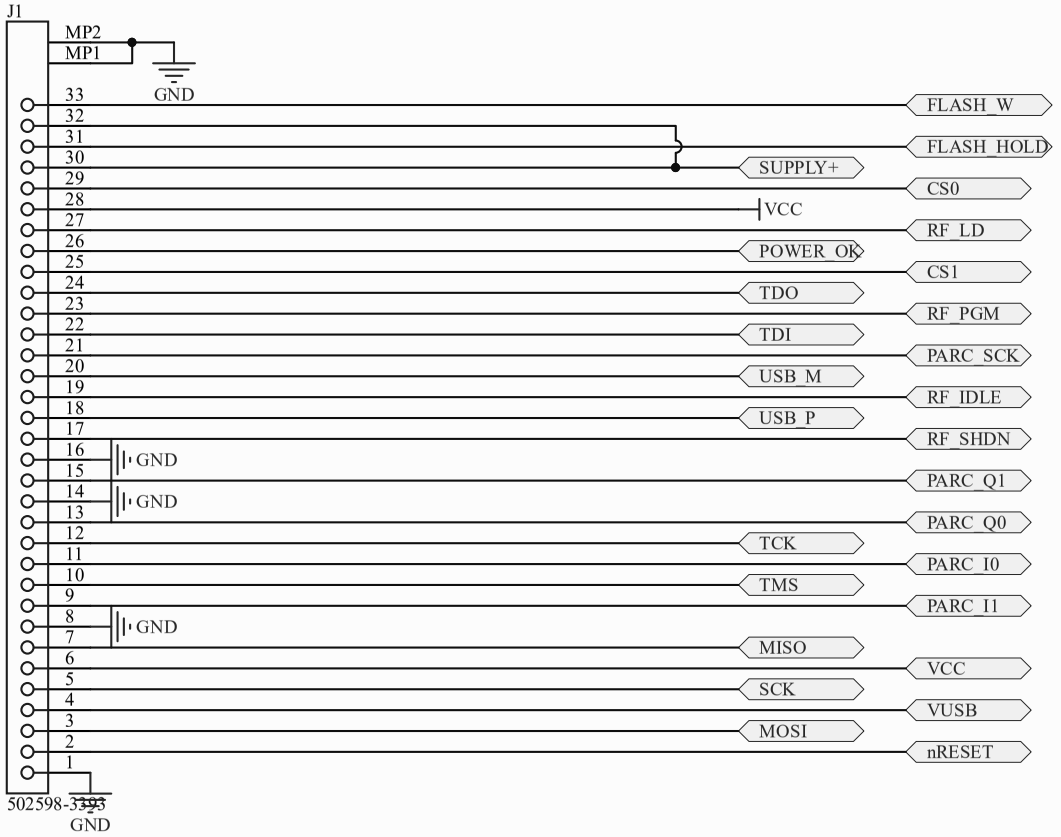
A.1.2 Erase the flash

Table A.1 on page A-9 contains the field *flag register*. This config field is used to erase the chip. If you set the field to *0x01*, once you reconnect the device,

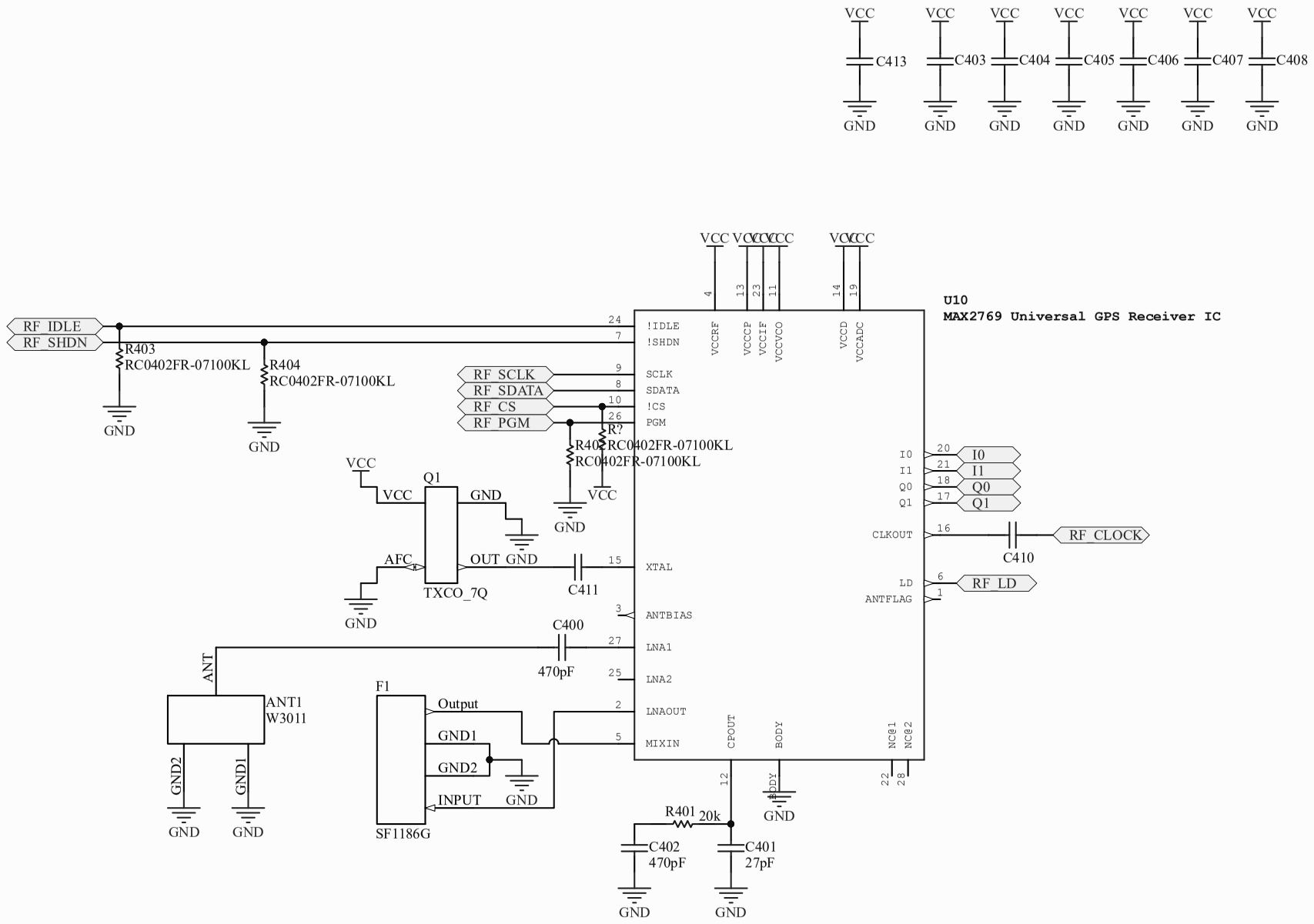
the GPS chip will trigger a bulk erase on the flash chip. While the bulk erase is being executed, you will not see the flash drives mounted in your computer.
Warning: The bulk erase will clear every single page within the flash chip. According to the datasheet, this typically takes around 1040 seconds(!), but can last even longer. Make sure **not to** detach the device from the USB port before the flash has been fully erased.
Once the bulk erase is done, the flash drives will reappear in your computer. It will clear the *flag register* and set it back to *0x10*. This way, you will know when you can safely detach the GPS device from your computer.

A.2 PCB files

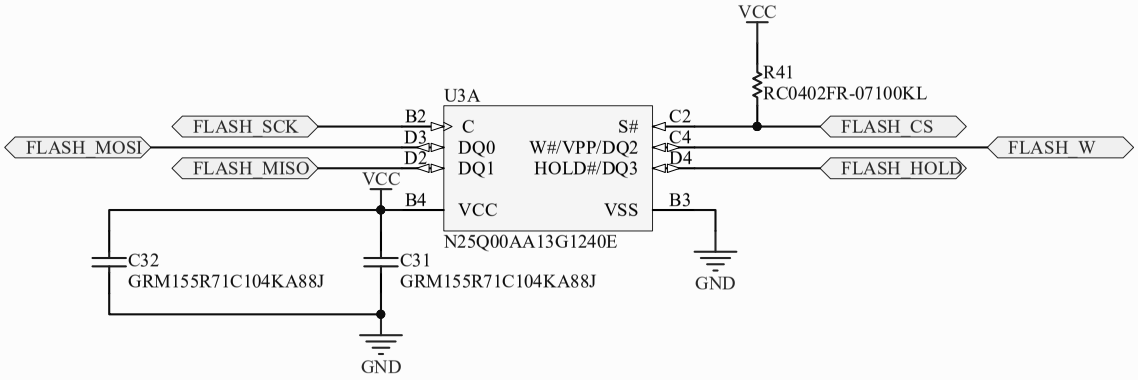
Attached are the circuit files.

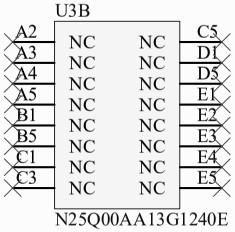


Title		
Size A	Number	Revision
Date: 29/11/2016		Sheet of
File: Y:\Development\..\Main.SchDoc		Drawn By:

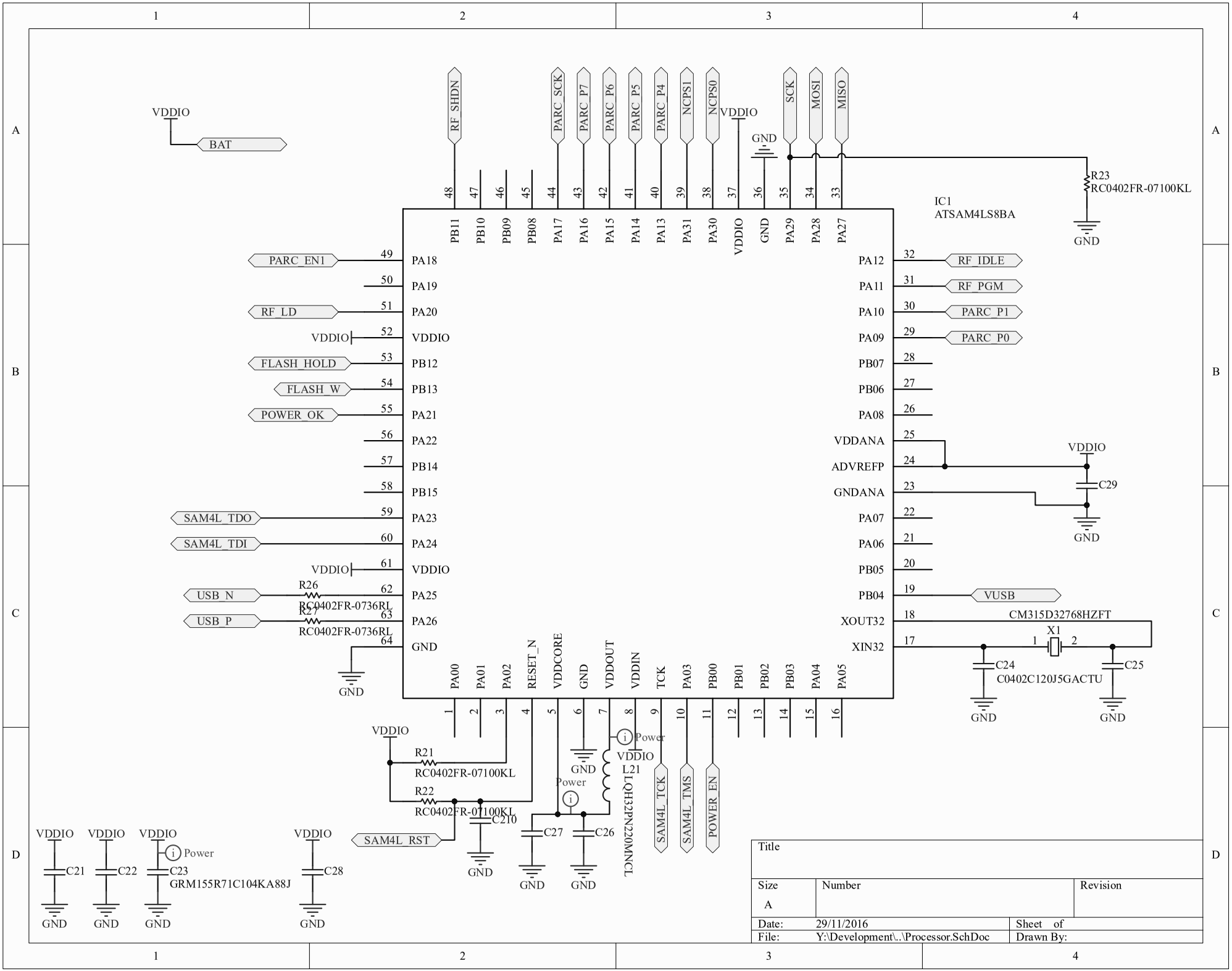


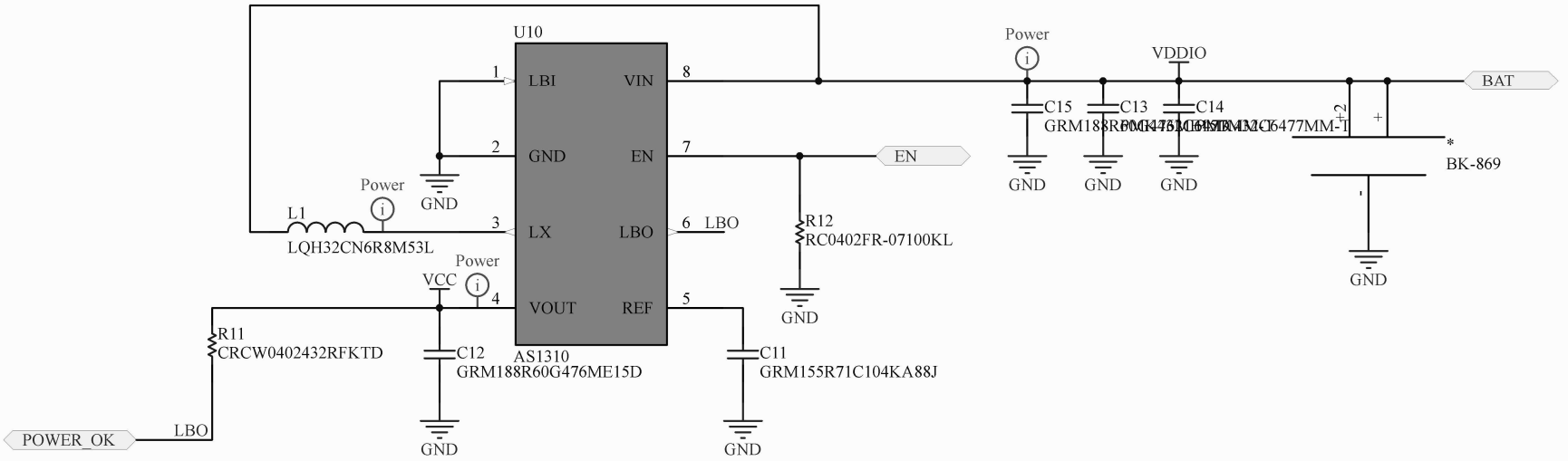
Title		
Size A	Number	Revision
Date: 29/11/2016		Sheet of
File: Y:\Development\..\RFFrontend.SchDoc		Drawn By:





Title		
Size: A	Number	Revision
Date: 29/11/2016		Sheet of
File: Y:\Development\..\Memory.SchDoc		Drawn By:


VDDIO
BAT
RF_SHDN
PARC_SCK
PARC_P7
PARC_P6
PARC_P5
PARC_P4
NCPS1
NCPS0
VDDIO
GND
SCK
MOSI
MISO
R23
RC0402FR-07100KL
GND
IC1
ATSAM4LS8BA
48 PB11
47 PB10
46 PB09
45 PB08
44 PA17
43 PA16
42 PA15
41 PA14
40 PA13
39 PA31
38 PA30
37 VDDIO
36 GND
35 PA29
34 PA28
33 PA27
PARC_EN1 49 PA18
50 PA19
RF_LD 51 PA20
VDDIO 52 VDDIO
FLASH_HOLD 53 PB12
FLASH_W 54 PB13
POWER_OK 55 PA21
56 PA22
57 PB14
58 PB15
SAM4L_TDO 59 PA23
SAM4L_TDI 60 PA24
VDDIO 61 VDDIO
USB_N R26 RC0402FR-0736RL 62 PA25
USB_P R27 RC0402FR-0736RL 63 PA26
64 GND
GND
32 PA12 RF_IDLE
31 PA11 RF_PGM
30 PA10 PARC_P1
29 PA09 PARC_P0
28 PB07
27 PB06
26 PA08
25 VDDANA
24 ADVREFP
23 GNDANA
22 PA07
21 PA06
20 PB05
19 PB04 VUSB
18 XOUT32 CM315D32768HZFT
17 XIN32
X1
1
2
C24
C0402C120J5GACTU
C25
GND
GND
GND
VDDIO
C29
GND
1 PA00
2 PA01
3 PA02
4 RESET_N
5 VDDCORE
6 GND
7 VDDOUT
8 VDDIN
9 TCK
10 PA03
11 PB00
12 PB01
13 PB02
14 PB03
15 PA04
16 PA05
VDDIO
R21
RC0402FR-07100KL
R22
RC0402FR-07100KL
SAM4L_RST
C210
GND
Power
GND
C27
GND
Power
C26
GND
VDDIO
L21
LQH32PN220MNCL
SAM4L_TCK
SAM4L_TMS
POWER_EN
VDDIO
VDDIO
VDDIO
Power
C21
C22
C23
GRM155R71C104KA88J
VDDIO
C28
GND
GND
GND
GND
Title
Size
A
Number
Revision
Date: 29/11/2016
Sheet of
File: Y:\Development\..\Processor.SchDoc
Drawn By:



Title			
Size A	Number		Revision
Date:	29/11/2016	Sheet of	
File:	Y:\Development\..\Power.SchDoc	Drawn By:	

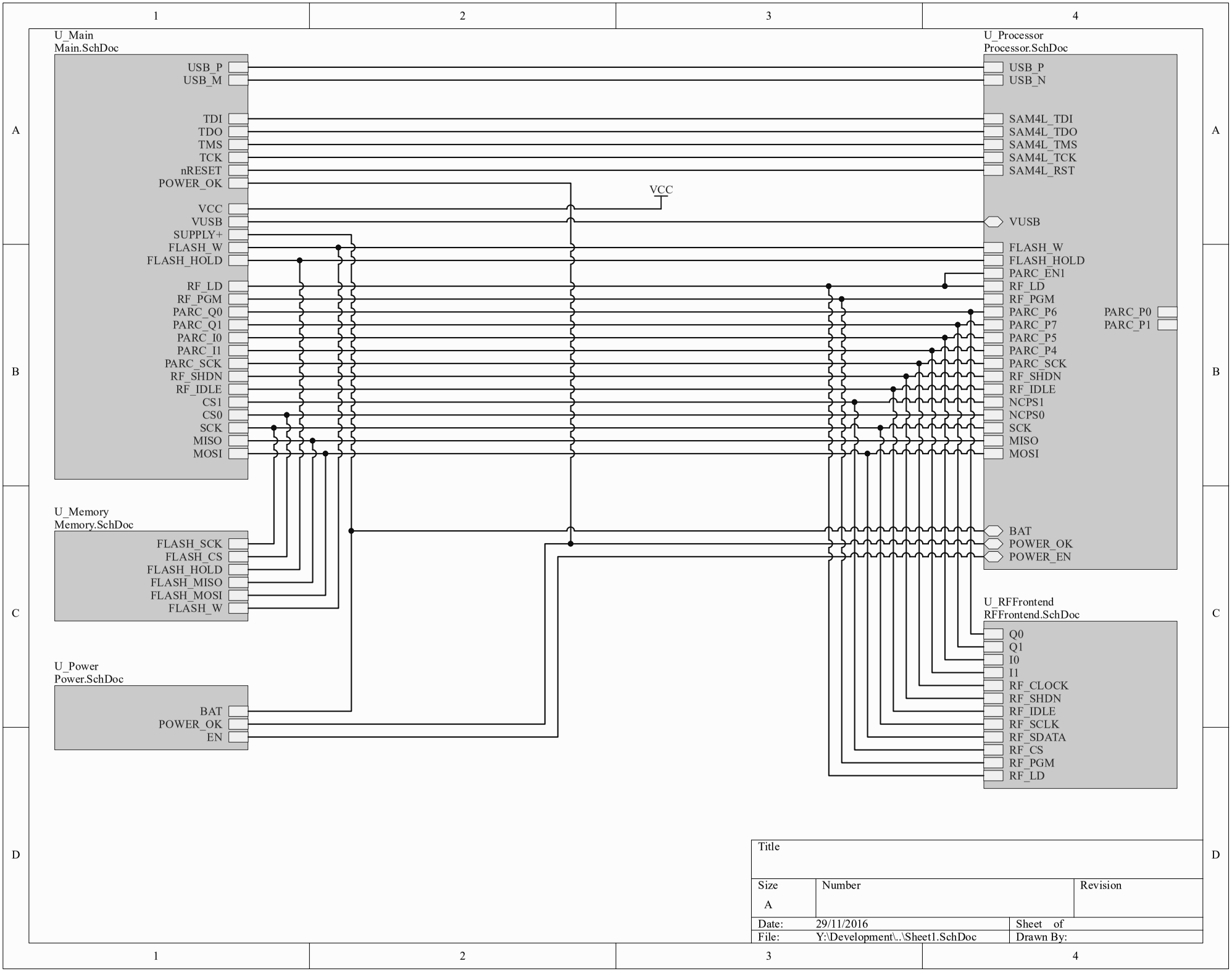

U_Main
Main.SchDoc
USB_P
USB_M
TDI
TDO
TMS
TCK
nRESET
POWER_OK
VCC
VUSB
SUPPLY+
FLASH_W
FLASH_HOLD
RF_LD
RF_PGM
PARC_Q0
PARC_Q1
PARC_I0
PARC_I1
PARC_SCK
RF_SHDN
RF_IDLE
CS1
CS0
SCK
MISO
MOSI
U_Memory
Memory.SchDoc
FLASH_SCK
FLASH_CS
FLASH_HOLD
FLASH_MISO
FLASH_MOSI
FLASH_W
U_Power
Power.SchDoc
BAT
POWER_OK
EN
VCC
U_Processor
Processor.SchDoc
USB_P
USB_N
SAM4L_TDI
SAM4L_TDO
SAM4L_TMS
SAM4L_TCK
SAM4L_RST
VUSB
FLASH_W
FLASH_HOLD
PARC_EN1
RF_LD
RF_PGM
PARC_P6
PARC_P7
PARC_P5
PARC_P4
PARC_SCK
RF_SHDN
RF_IDLE
NCPS1
NCPS0
SCK
MISO
MOSI
PARC_P0
PARC_P1
BAT
POWER_OK
POWER_EN
U_RFFrontend
RFFrontend.SchDoc
Q0
Q1
I0
I1
RF_CLOCK
RF_SHDN
RF_IDLE
RF_SCLK
RF_SDATA
RF_CS
RF_PGM
RF_LD
Title
Size
A
Number
Revision
Date:
29/11/2016
Sheet of
File:
Y:\Development\..\Sheet1.SchDoc
Drawn By:

Table A.1: Config for GPS chip

uint32_t	written	0x01020304	Unless "written" equals its default, chip will reset config
uint32_t	max_config_1	0xA2951A30	MAX Register CONF1
uint32_t	max_config_2	0x85502881	MAX Register CONF2
uint32_t	max_config_3	0xEAFF1F42	MAX Register CONF3
uint32_t	max_pll_conf	0x9CC00083	MAX Register PLLCONF
uint32_t	max_div	0x0C000804	MAX Register DIV
uint32_t	max_fdiv	0x80000705	MAX Register FDIV
uint32_t	max_strm	0x80000006	MAX Register STRM
uint32_t	max_clk	0x10061B27	MAX Register CLK
uint32_t	samples	4096	Number of samples
uint32_t	flags	0x00	Flag register (Explanation: A.1.2)
uint32_t	interval	3600	Interval between the samples (in sec)
uint32_t	user1	0	Don't care, store your own data
uint32_t	user2	0	Don't care, store your own data
uint32_t	user3	0	Don't care, store your own data
uint32_t	user4	0	Don't care, store your own data