Mixed Signal, Hard-Software-Co-Design
(Embedded Systems)

GPS Evaluator

Characterization of oscillators

Tobias Himmer (24111)
Michael Wydler (24113)
Karl-Heinz Zimmermann (24112)

19.01.2013

Hochschule
Ravensburg-Weingarten
University of Applied Sciences

Contents

1 Basics

The GPS Evaluator shall display the system's position as reported by the GPS module (the position is retrieved using at least 3 satellites). The system (Figure 1.1) uses the GPS clock as reference signal for further time measurements on the Spartan 3E board. The clock deviation of both, the internal system clock and the external oscillator is calculated. The calculated values and the GPS position are sent to a server. It is also possible to connect multiple smartphones to the server, which also send their current position. The position of the GPS board and the smartphones can be displayed using any[1] webbrowser.

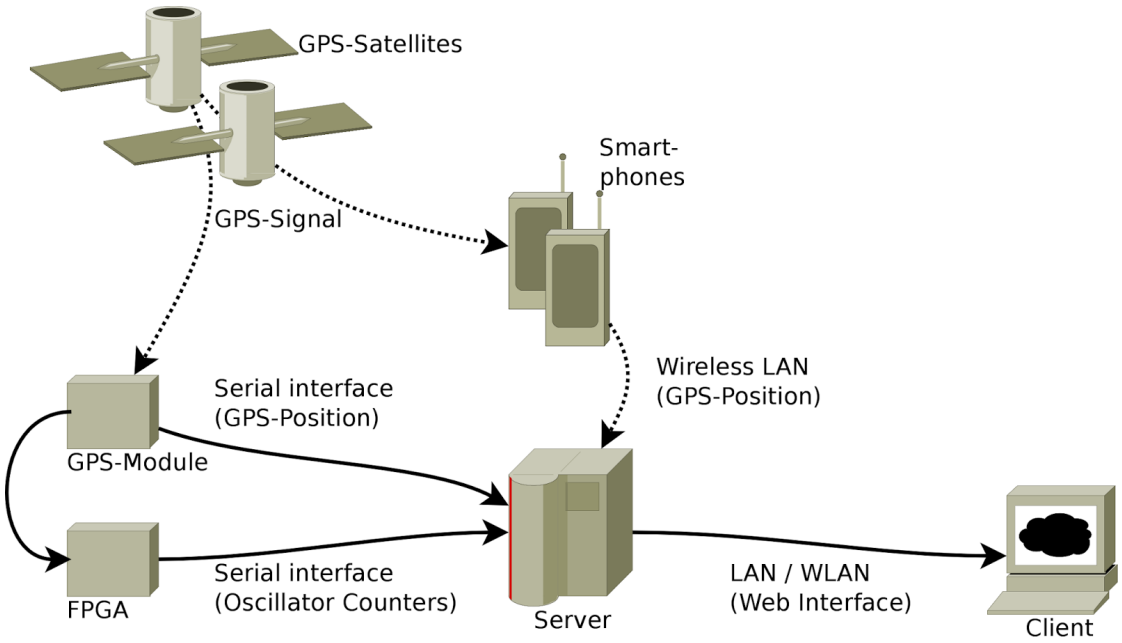


Figure 1.1: Overview of the complete system.

[1] Internet Explorer excluded

1.1 Concept of GPS

The GPS is using the time-delay of the travelling signal to determine the current position. Therefor at least 3 satellites are required (Figure 1.2). The satellites need a synchronized and very accurate clock to get an accurate position. A fourth satellite is required to get an accurate time signal. This time signal is well suited for characterizing an oscillator's frequency and hence used as the reference time signal in this project.

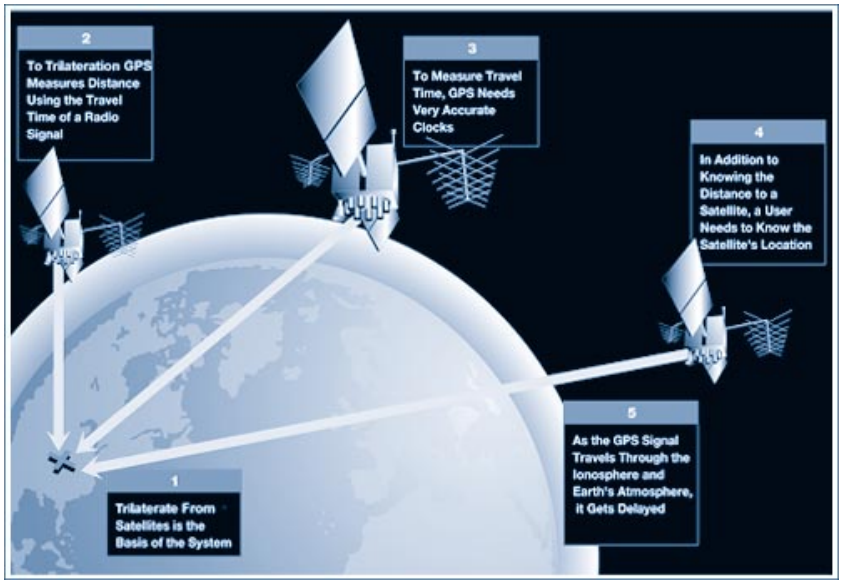


Figure 1.2: Concept of GPS

For more information, please see [2].

1.2 NMEA Protocol

NMEA 0183 is a standardised communication protocol, which was defined by the National Marine Electronics Association (NMEA). It is used for the communication between GPS devices and computers. The data sent with this protocol is structured in so-called sentences and has the following format:

`$SDDBT,22.3,f,6.8,M,3.7,F*3F<cr><lf>`

The "$" character indicates a new sentence and the "*" character indicates the end. The "*" is followed by the check-sum (XOR of all characters between the "$" and "*") of the sentence.

For this implementation only 3 of over 50 possible sentences are used:

- **GPRMC**
  Recommended Minimum Sentence C, supported by every GPS device. Contains information about date, time, status and mode.
- **GPGGA**
  Contains the most important information about position, height and accuracy.
- **GPGSA**
  Contains information about fixed satellites, fix quality and accuracy.

For detailed information on MNEA sentences refer to [1] and [3].

2 FPGA

2.1 Overview

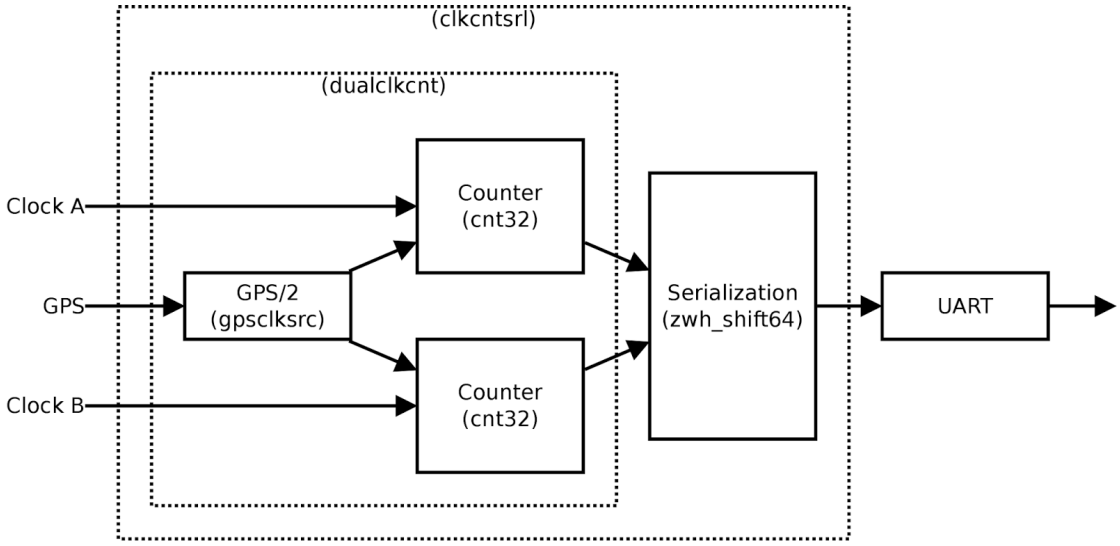


Figure 2.1: Overview

Clock A is used as system clock - e.g. edge detectors use this clock source as reference. Thus, clock A should always be faster than clock B!

The internal circuit for the FPGA is divided into the following tasks:

- Generation of the halved GPS clock signal as well as detecting rising and falling edges of the halved GPS clock signal.
- Counting of the frequency for both oscillator inputs in parallel.
- Serializing the counter values.
- Sending the serialized counter values to a computer for visualization.

These tasks are divided into two phases, each one second in duration, depending on the halved GPS clock signal:

- Clock signal equals logical one: Both connected frequencies are counted until the end of phase 1, then held for further processing.
- Clock signal equals logical zero: Serializing and sending the previously determined counter values via UART. At the end of phase two the counters are reset into their initial state to prepare for counting frequencies in the next cycle.

Therefore every two seconds, a new pair of counter values is generated and sent to the connected device. Figure 2.2 illustrates one measurement and transmission cycle.

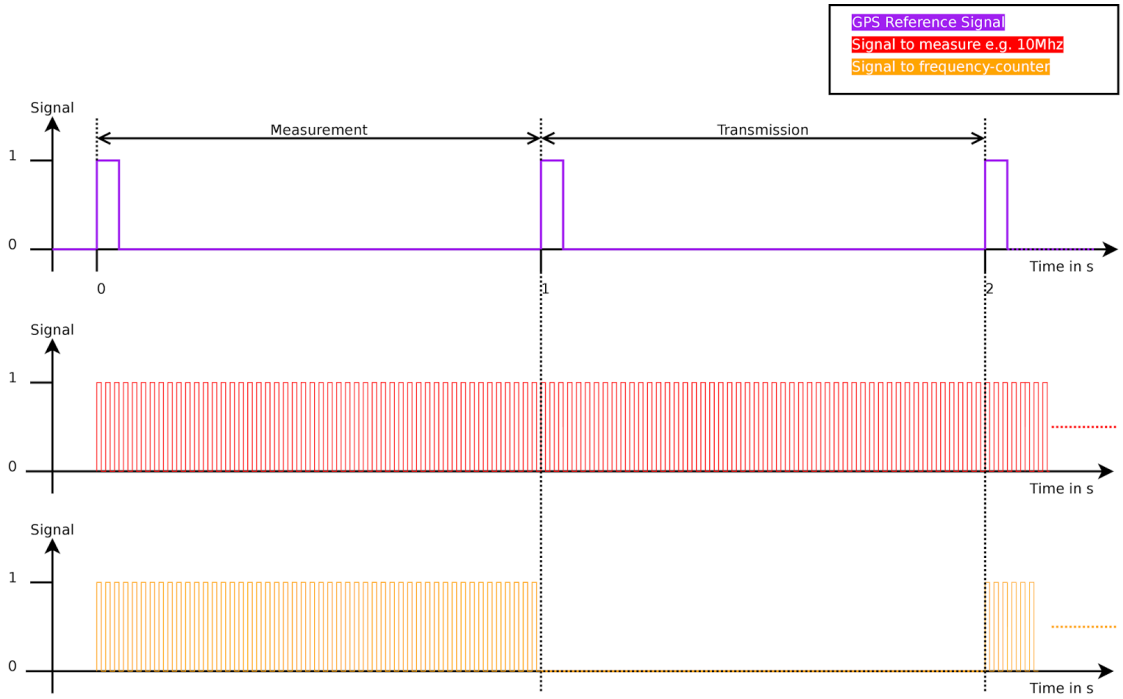


Figure 2.2: The two phases of the FPGA

2.2 GPS Clock

The GPS clock passes a toggle flip-flop to generate the halved GPS clock (Figure 2.3). Additionally the falling and rising edges are detected: A pulse on the rising edge output marks the beginning of a measurement-phase, whereas a falling edge marks the end of the measurement-phase and the beginning of a transmission-phase.

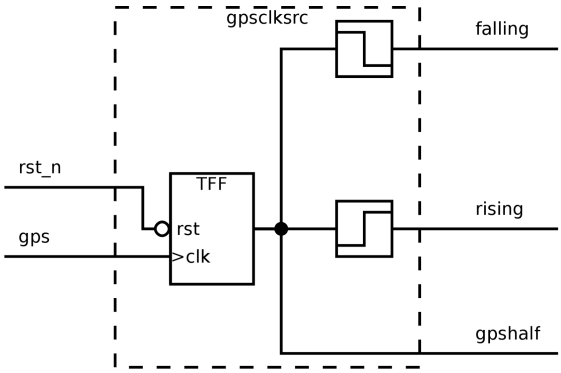


Figure 2.3: GPS clock divider and edge detectors

2.3 2x32-Bit Binary Counter

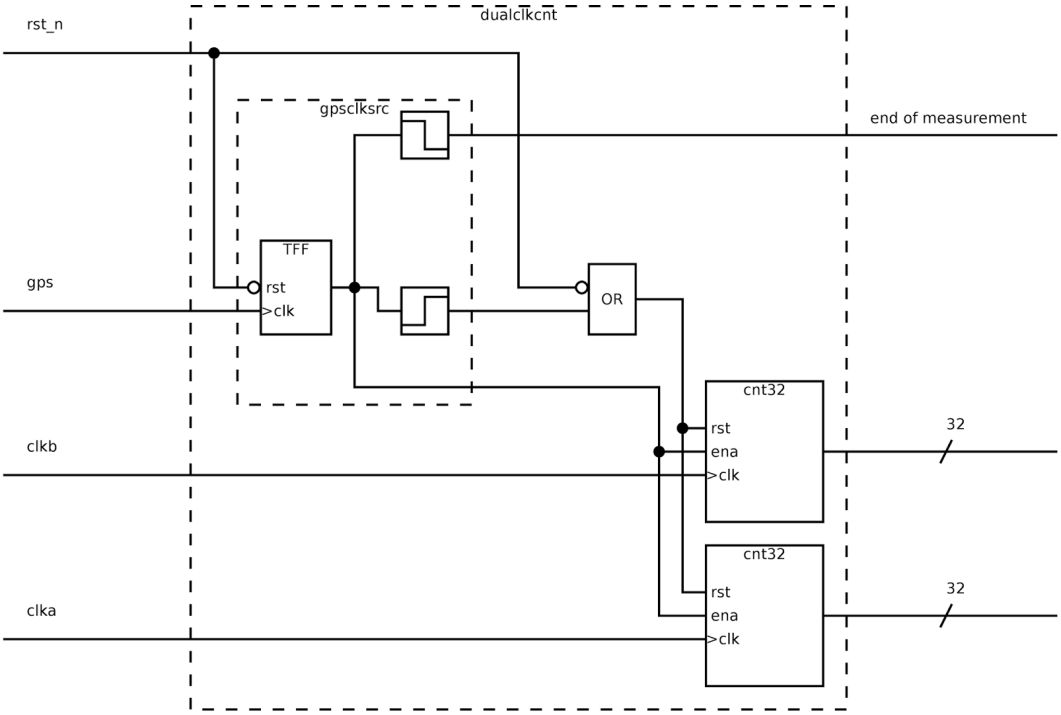


Figure 2.4: 32-bit binary counters

Frequency counting is implemented using a 32-bit binary counter for each oscillator input (Figure 2.4). No clock dividers for the oscillator inputs are used in this design. Each counter ranges from 0 to 4.294.967.295 (2^{32}-1), hence frequencies up to ~4.294GHz could be counted for one second.

2.4 Serialization

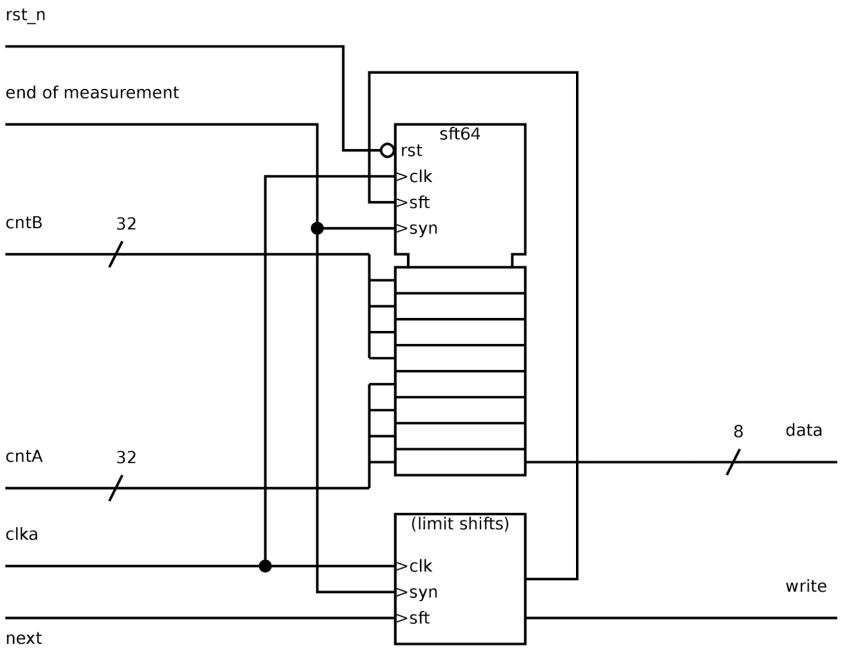


Figure 2.5: Serialization of two 32-bit counter values

The serial transmitter (Figure 2.5) is only capable of sending one byte at a time, thus a shift register is needed to serialize the 64 bits into 8 bit chunks.

Each time a new pair of counter values is generated, the values are loaded into the shift register ("`syn`" signal) and a "`write`" pulse is sent to the UART to initiate a transmission. As soon as the UART is capable of consuming one more byte, the "`next`" signal is triggered by the UART. This causes a shift to the next byte and a subsequent "`write`" pulse until 8 bytes are sent (7 shifts total).

2.5 UART

Based upon http://www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/1999f/UART/txmit.vhd but modified interface behaviour and replaced baud rate generation with a more flexible clock divider that is not limited to dividers to the power of two.

The two counter values are sent in little-endian[1] format as seen in figure 2.6.

[1] http://en.wikipedia.org/wiki/Little-endian

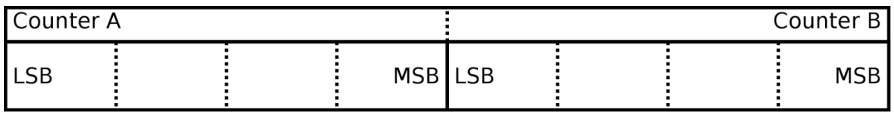


Figure 2.6: Binary format for counter values

Example of a transmission and it's decoding:

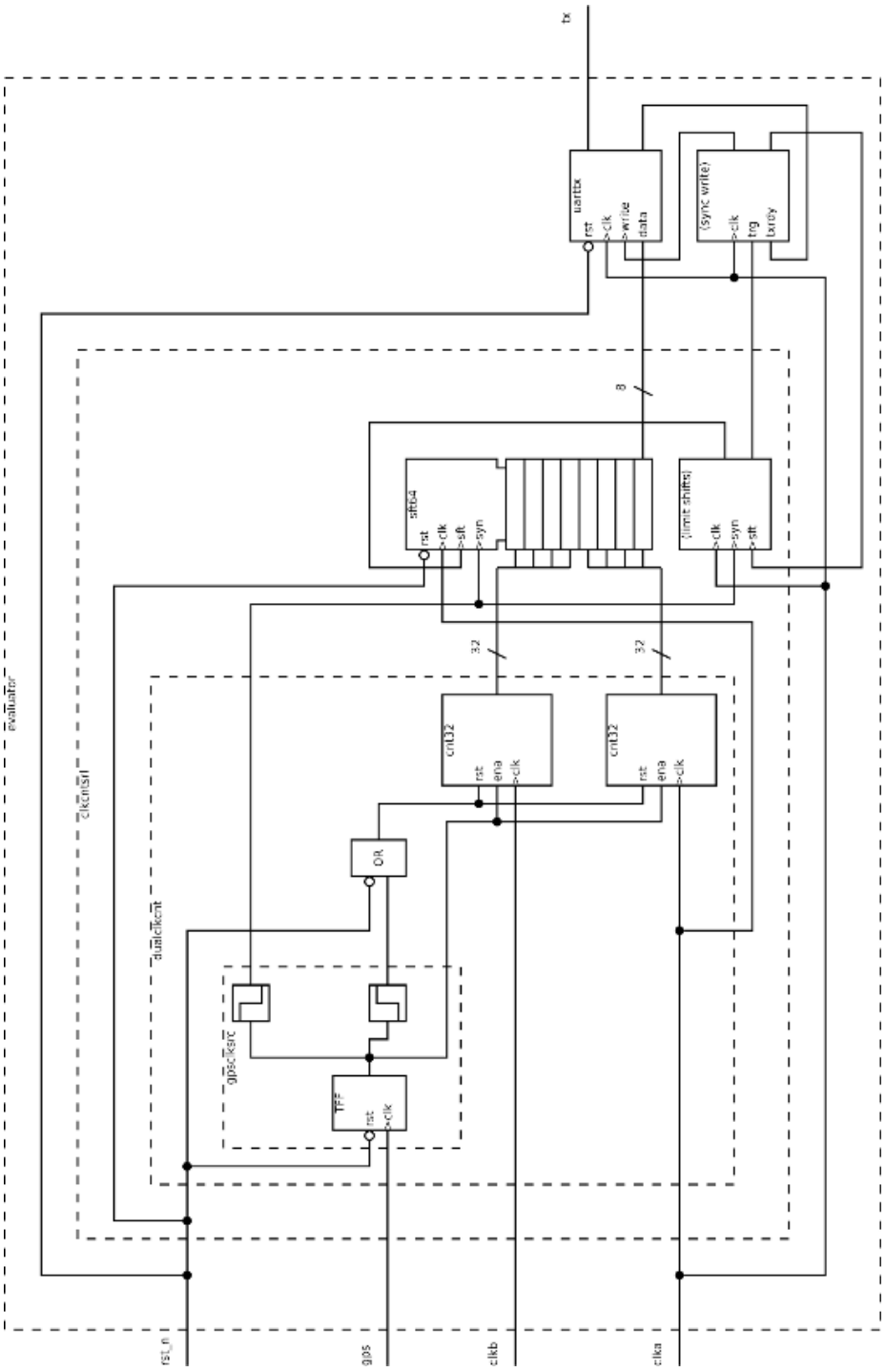
FPGA sends:

`0xA4 0xF0 0xFA 0x02` for counter A and
`0xF8 0x96 0x98 0x00` for counter B

Server decodes:

50,000,036 (~50MHz) for counter A and
10,000,120 (~10MHz) for counter B

2.6 Detailed Schematic



3 Server

The server is programmed using Python[1]. Python is used because it is a high-level programming language, which provides a large and comprehensive standard library.

To manage the application environment, the Python package "virtualenv" is used. With this package it is possible to create a local environment and only install/activate necessary packages (no root privileges required). For this application only the Python standard library and two additional packages are needed:

argparse==1.2.1 (to manage program arguments)

pyserial==2.6 (to read data from serial/usb ports)

To run the server, an Apache Webserver[2] is required. Also the `mod_rewrite` and `mod_proxy` modules must be activated. For security reasons, a rewrite of the event-stream URL to a relative application URL is applied. This is achieved by a `.htaccess` file in the root directory of the website itself:

```
<IfModule mod_rewrite.c>
        RewriteEngine on
        RewriteRule events http://localhost:8001/ [NC,L,P]
</IfModule>
```

3.1 Components

The complete program is split into four threads to provide parallel processing. All threads have a shared message queue to pass messages from one thread to another. The main program only sets up all threads and starts them. If the main program is terminated, all threads are terminated as well.

[1] http://en.wikipedia.org/wiki/Python_(programming_language)
[2] http://httpd.apache.org/

The server can be started with the following command:
`python manage.py runserver [--debug]`

The option "`--debug`" can be used to simulate a connected FPGA- and GPS board. It will generate random values for the two counters and set a fixed GPS position. The server can be stopped by pressing `Ctrl+C` on the keyboard. This will stop all threads and quit the program.

3.1.1 FPGAThread

This thread is reading eight bytes (four bytes for each counter value) from the serial port connected to the FPGA in an endless loop. This byte-array is unpacked (little-endian) into two integer variables. In the next step, these values are encoded as JSON data and finally put into the shared message queue (Figure 3.1).

JSON example:

```
json-data = {
    "cnt1": "50000120",
    "cnt2": "10000048"
}
```

Message example:

```
event: clk\ndata: json-data\n\n
```

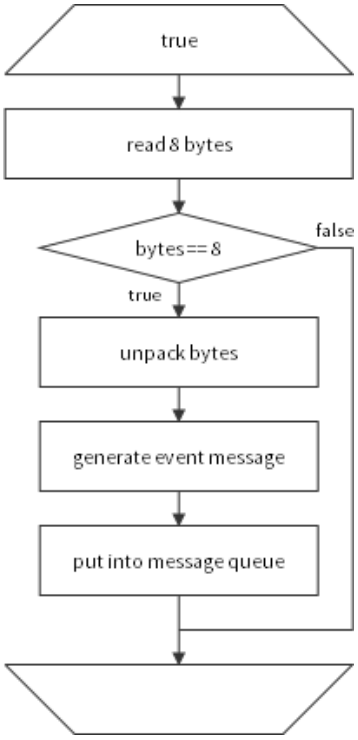


Figure 3.1: FPGA thread activity diagram

3.1.2 GPSThread

This thread (Figure 3.2) uses an endless loop to read every byte from the GPS module. Until a new line character is received, all characters are merged into a sentence. If a sentence is complete, the required information is parsed from the sentence. Latitude and longitude must be converted from the degrees/minutes/seconds-format into the decimal degrees format. After JSON encoding, the data is also put into the shared message queue.

JSON example:

```
json-data = {
    "time": "17:47:25",
    "date": "18.01.2013",
    "latitude": "...",
    "longitude": "...",
    "altitude": "...",
    ...
}
```

Message example:

```
event: gps\ndata: json-data\n\n
```

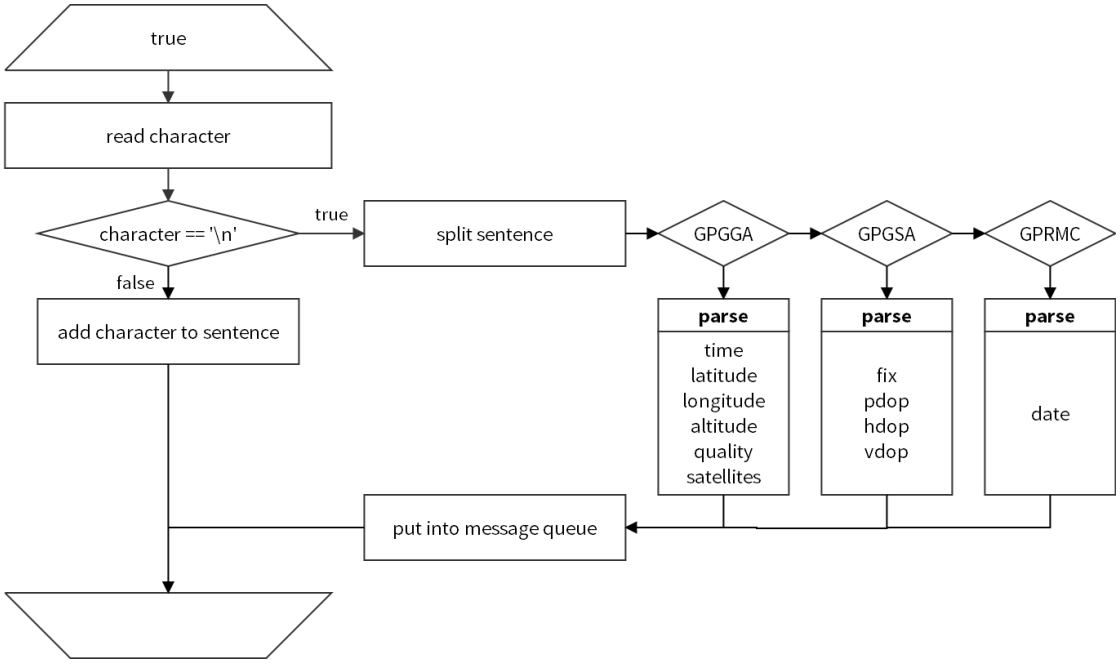


Figure 3.2: GPS thread activity diagram

3.1.3 TCPThread

This thread (Figure 3.3) is responsible for the TCP connections to the smartphones. As this is a multithreaded server, it is possible to connect multiple smartphones simultaneously. The

data is also put into the shared message queue. To close the connection, the smartphone's client has to send a message containing "bye".

JSON example:

```
json-data = {
    "latitude": "...",
    "longitude": "...",
    "altitude": "...",
}
```

Message example:

```
event: ext\ndata: json-data\n\n
```

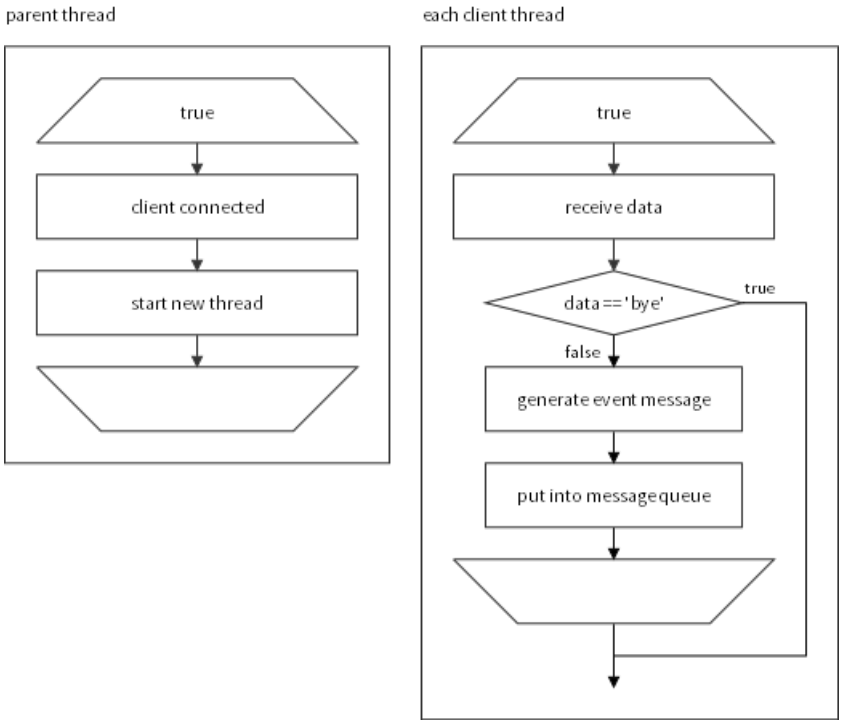


Figure 3.3: TCP thread activity diagram

3.1.4 SSEThread

This thread (Figure 3.4) is used to transmit the events from the FPGA-, GPS- and TCP-Thread to the connected web-client. Basically this thread is a HTTP-Server, whose content-type is changed to "`text/event-stream`". Whenever a new event is available in the shared message queue, it will be sent to the client.

Stream example:

```
event: type\ndata: data\n\n
```

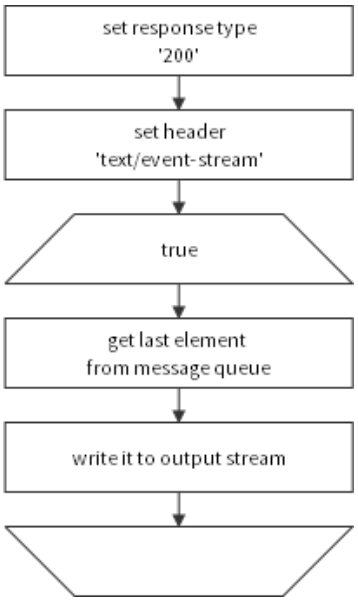


Figure 3.4: SSE thread activity diagram

3.2 Activity Diagram

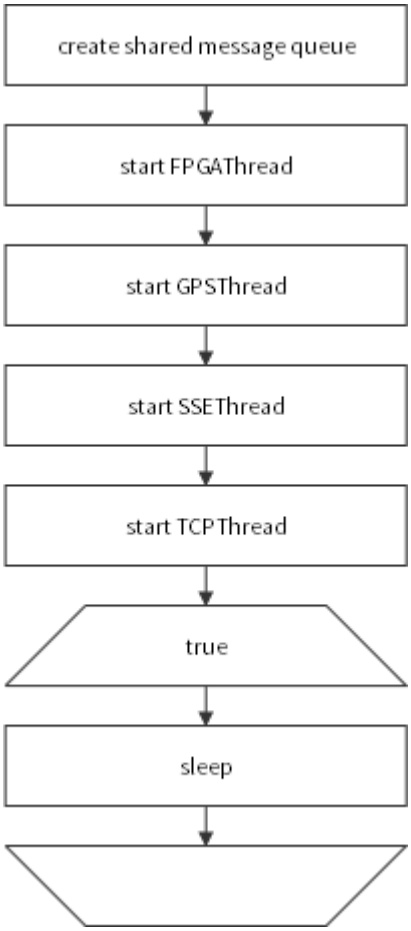


Figure 3.5: Activity diagram

4 Client

4.1 Web-Interface

To display the collected information, a HTML5[1] web-interface has been developed. This web-interface (Figure 4.1) can be opened with every browser[2]. JavaScript[3] (with jQuery), a client-side scripting language is used to receive new information from the server and update the interface.

Figure 4.1: User-Interface

The page can be split into three components. The first component is the map. This map uses the Google Maps JavaScript API[4] to display the current position of the GPS board at the center of the map. In addition, the position of connected Android smartphones is displayed (see Chapter 4.2). It is also possible to display the path of the last 100 positions and an estimated precision (as circle around the icon) of the GPS board.

[1] http://en.wikipedia.org/wiki/HTML5
[2] The complete web-interface has been tested with Google Chrome, so we recommend to use it.
[3] http://en.wikipedia.org/wiki/JavaScript
[4] https://developers.google.com/maps/documentation/javascript/?hl=de

The second component is the information HUD on the left side. All important information from the GPS board, like position, fix quality, number of satellites can be found there. The last component is the characterization of oscillators. Therefor we used the Highcharts[5] JavaScript library, to display two graphs at the head of the page. With these graphs the frequency of the oscillators (only the last 20 samples, the value range is adjusted dynamically) is visualized.

The communication with the server is realized via server-send-events[6] (sse). These events are generated by the SSEThread (see Chapter [SSETHREAD]) in the server application. On the client side, it is sufficient to connect to this event-stream and react to the events. The basic functionality is shown in the next example.

Example:

```
var source = new EventSource('/gps/events');
source.addEventListener('clk', function(e) { do something with data }, false);
source.addEventListener('gps', function(e) { do something with data }, false);
source.addEventListener('ext', function(e) { do something with data }, false);
```

Because the data is encoded as JSON, it is trivial to extract the required information - JavaScript has a build-in function called `JSON.parse(e.data)`. This function parses the JSON data and returns the data as object or array.

4.2 Android

Since the web-interface already displays the GPS position of the GPS module, it was obvious that other devices may be shown on the screen, too.

Android is well suited for this task, as within the Android device it is possible to retrieve the GPS information of the smartphone itself. This GPS dataset is transferred to the server, that displays all connected smartphones. The developed application, is called an Android "App" and is shown in the following figures.

Figure 4.4 shows the design of the application.

The diagram shows that at first, the user has to configure the network connection to the server by entering it's IP-Address and port. After this step the connection to the server can be opened. If it is not possible to establish a connection, the App will repeat the connection attempt until a connection is established or the user cancels the connection process. If the connection is established, the App listens to the GPS data, which is provided by the Android OS. Two parameters can be chosen that define the update condition of the GPS position:

[5] http://www.highcharts.com/
[6] http://en.wikipedia.org/wiki/Server-sent_events

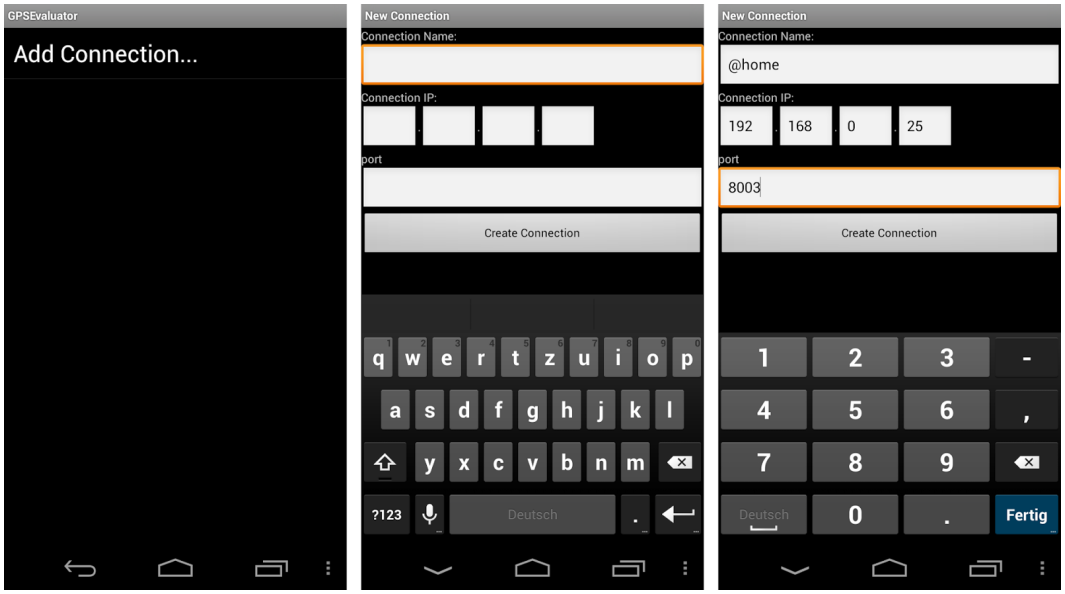


Figure 4.2: Adding a connection

A minimum change of the position and additionally a minimum time interval between two updates can be defined. (current implementation: min. distance change=1 meter, min. time change=1 second)

Android calls the method which is responsible for updating the server. This method shown on the right side of the diagram, is running in parallel to the main process. The method receives the GPS information as an argument. This GPS dataset is converted to a JSON string which contains the Android's Device-ID, latitude, longitude and the altitude. Further, this generated JSON string is transmitted to the server which shows a symbol for each smartphone on the map. This task is repeated until the user stops the App. As the server is multithreaded, it is possible to connect any number of smartphones. While testing the GPS device it turns out, that occasionally there is a difference of about 10-20 meters from the real position. So with three devices (one GPS device and two Android devices e.g.) it is possible to make a triangle interpolation to minimize the location error of the single devices. This referencing process can for example be done at beginning, but it is currently unimplemented.

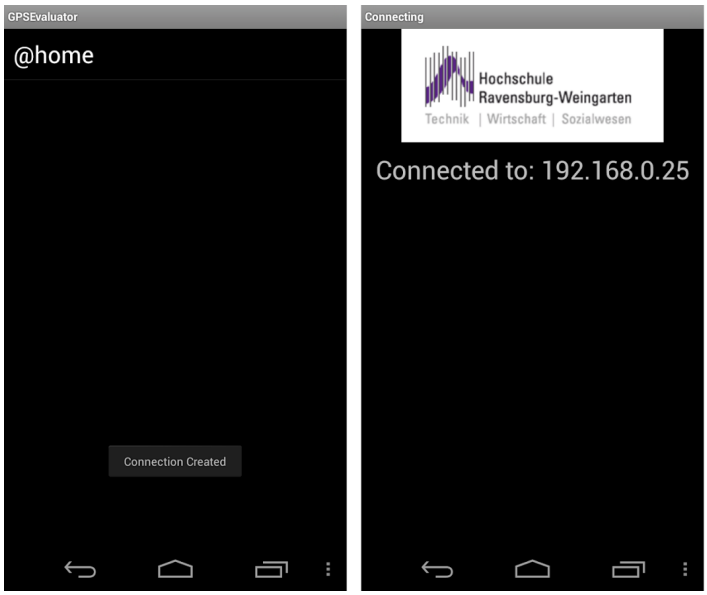


Figure 4.3: Connect to server

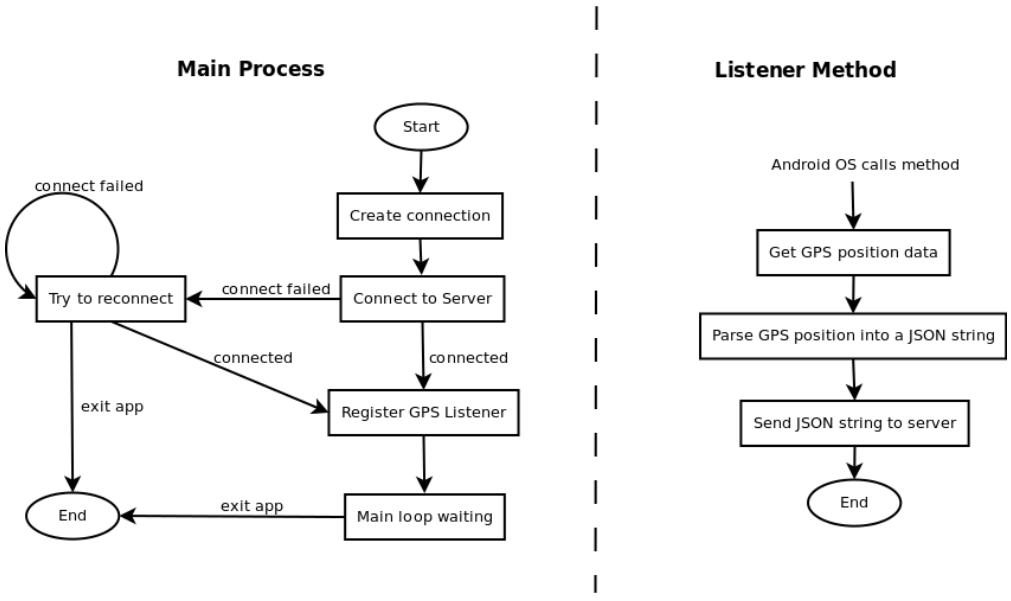


Figure 4.4: Android activity diagram

5 Statistics

As shown in Chapter 2.3, a 32-bit counter for each oscillator is used. As a result no dividers are necessary. The obtained counter values are 100% accurate, as every pulse of the oscillator is counted.

It seems that the counter values fluctuate about plus minus three. A possible explanation for these differences could be that the noise on the GPS clock signal affects the accuracy of the counting period. Among other things, this effect depends on the cable length between GPS module and FPGA board.

For more detailed statistics, more sample data as currently available would be necessary.

6 Problems

The one-second signal between GPS module and FPGA is prone to noise and other disturbances caused by long wires. These circumstances manifest in incorrect counter values.

Unclean rising and/or falling edges produced measurement phases shorter than one second. The most common error appeared after the falling edge on the signal: The oscillations after the falling edge resulted in falsely detected rising-edges that produced counter values amounting a fifth of the expected values (the one-second signal of the GPS module has a high-time of a fifth of a second). This effect emerged only on certain device configurations.

To mitigate this problem, shorter wires have been used. Additionally some experimentation was done on where to connect the individual device's power supplies for best signal quality. Following images are taken from an oscilloscope connected to the GPS clock signal:

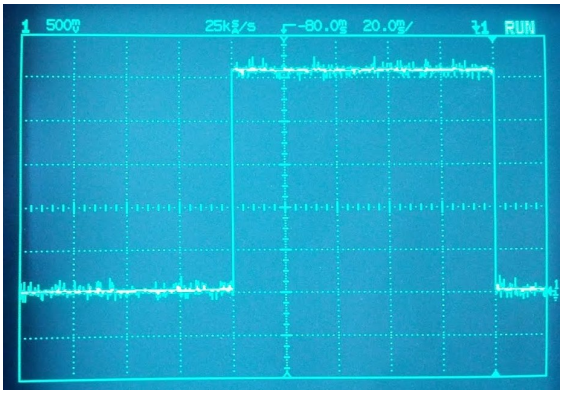


Figure 6.1: GPS signal pulse

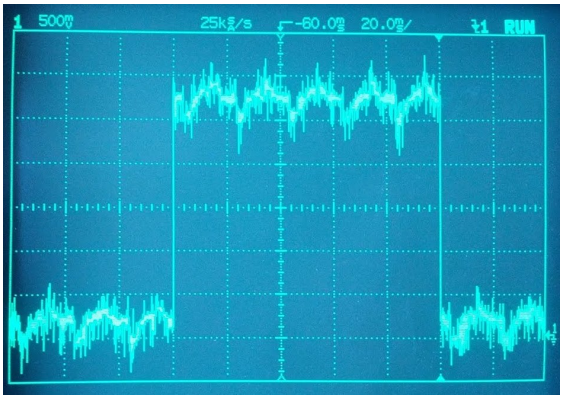


Figure 6.2: Noisy GPS signal pulse

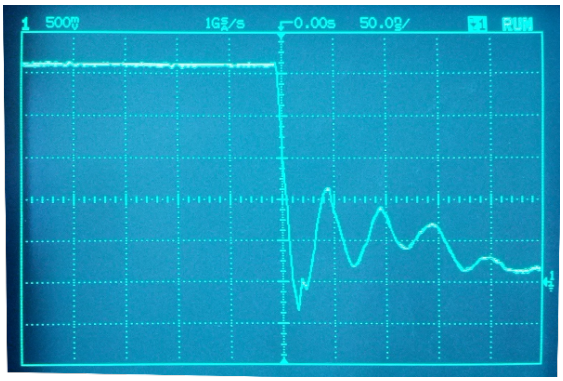


Figure 6.3: Falling-edge oscillations

7 Summary

For us as computer scientists this project was a great experience. Common computer scientists only program the software running on some hardware. Until this course, nobody of us had experience with designing and implementing hardware circuits itself using VHDL. At the beginning we had some problems getting used to the idea of VHDL programming, as VHDL is a descriptive language that differs from common programming languages like C, C++ or Java.

It was more effective to learn about hardware-software-co-design working on a hands-on project than just learning the theoretical approach. There are many possibilities on how to use GPS information in applications, so we implemented additional functionality in our project (e. g. display current position on map, connect Android smartphones).

The logic is realized in VDHL and could successfully be synthesised on our target system. Both oscillator frequencies are measured correctly and sent to the server. Also the GPS board and all connected smartphones transfer their position information to the server. The server forwards all this information to the client, which in turn displays the data in a user-friendly manner.

https://github.com/wydler/gps-evaluator

Bibliography

[1] Glenn Baddeley. Gps - nmea sentence information. http://aprs.gids.nl/nmea/, 2001.

[2] GPSWorld. The role of gps in precise time and frequency dissemination. http://ilrs.gsfc.nasa.gov/docs/timing/gpsrole.pdf, 1990.

[3] Wikipedia. Nmea 0183. http://en.wikipedia.org/wiki/NMEA_0183.