A Fully Generated GPS Receiver

Daniel Grubb, Olivia Hsu, Zhaokai Liu, Aviral Pandey,
Dinesh Parimi, Ruocheng Wang, Zhongkai Wang, Nick Werblun

***Abstract*— Designing individual GPS receivers for positioning and navigation applications with many different GPS signals available is an expensive and time-consuming process. Current GPS chips are largely designed as single instances, which can be prohibitively expensive for low volume applications. This paper presents an RTL generator for an L1 C/A GPS receiver, designed and tested in the CHISEL ecosystem, and interfaced with the RISC-V-based Rocket Chip. An integrated hardware/software approach was adopted to create a parameterized and extendable design that can be deployed to an FPGA and run C code, allowing for decoding of GPS signals and interfacing with other devices. Generated instances of the GPS receiver blocks were able to perform critical GPS tasks in simulation, such as GPS signal acquisition and GPS signal tracking.**

I. INTRODUCTION

As one of the most widely used satellite navigation systems, GPS has become ubiquitous in everyday life, and much of the signal decoding is performed in digital ASICs or FPGAs. Historically, civilian devices have used the L1 C/A channel, but as the deployment of GPS Block III approaches, these devices will be expected to transition to make use of the new channels. The design process for receivers for these new channels could be improved by taking advantage of the ”generators, not modules” philosophy of CHISEL, allowing for greater design reusability through inheritance and parameterization. In this document, we explore the design space of a CHISEL generator for an L1 C/A GPS receiver.

Current state-of-the-art GPS receivers typically use an FFT based approach for GPS signal acquisition because of the improvement in acquisition time over serial search methods. A more recent development that can potentially improve acquisition time further is the use of the sparse FFT [7], which takes advantage of the sparse nature of the correlation of GPS signals. Also, GPS signals are relatively slow, some GPS receivers perform calculations for the tracking system loop (code loop and carrier loop) in software rather than in hardware.

II. GPS SIGNAL SPECIFICATION

This design is targeting the L1 C/A GPS signal - the only legacy civilian channel. The carrier frequency is 1575.42 MHz. The PRN codes are of length 1023 and are transmitted at 1.023 MHz. With a high autocorrelation and a very low cross-correlation, the PRN codes allow the receiver to uniquely identify the transmitting satellite. The navigation data bits are modulated on top of this using a BPSK scheme at 50 Hz.

In bit space, the basic unit of a GPS transmission is a subframe - 300 bits of data organized into ten 30-bit words. Each of these words contains 24 bits of data and 6 bits of parity that are used in the receiver to verify the integrity of the received packet. Five subframes make up one frame, which constitutes a full transmission, and each subframe contains different data about the satellite's position, its health, and other relevant information. Data is transmitted at a rate of 50 bits per second.

III. DESIGN

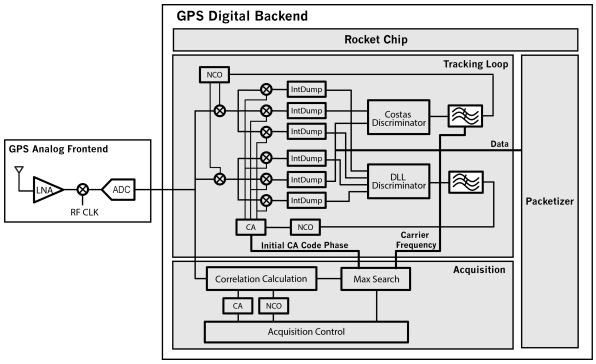

Fig. 1. The High-level Architecture of the GPS L1 C/A Receiver

A. Acquisition Loop

The acquisition loop is designed to get the correlation given the specified frequency and code phase of a satellite, then pass the coarse frequency and code phase with maximum correlation to the tracking loop if a satellite is found.

1) Serial Search: Though being quite slow, the simplest way to find out the optimal frequency and phase is to use the serial search, which is realized by obtaining the correlation of only one frequency and code phase at a time, and let the controller sweep the frequency & code phase. The expression of the correlation is: $R^2[m] = [\sum(x[n] \cdot CA[n+m] \cdot cos[\Omega n])]^2 + [\sum(x[n] \cdot CA[n+m] \cdot sin[\Omega n])]^2$

Where Ω represents the frequency and m represents the code phase. This can be realized by several small multipliers, two digital integrators and a large square sum block, as shown in Fig.2. Notice that we are using the same ADC, cosine and sine signal data if we are getting the correlation of different code phases but a fixed frequency, we can actually acquire the correlation of different code phases at the same time by saving the CA code in the shift registers, which is sometimes called parallel search or say hybrid search

shown in Fig.2. In this case we can simultaneously get the correlation of all the code phases of one frequency at a time so that we only need to sweep the frequency. Searching the all the code phases in parallel reduces the acquisition time significantly, but also increases the chip area substantially. To handle this issue, the acquisition loop has parameterized number of correlators. For example, there are totally 16384 code phases in this design, but since the tracking loop can finally find out the resolute code phase at cost of longer tracking time if given a coarse code phase which is less than 1 chip (16 code phases) away from the accurate one, we only need to acquire the correlation per 16 code phases, which means there will be only 1023 correlators instead of 16384 ones. For safety in this design, we are still acquiring correlation per 8 code phases, which means there are 2046 correlators in parallel.

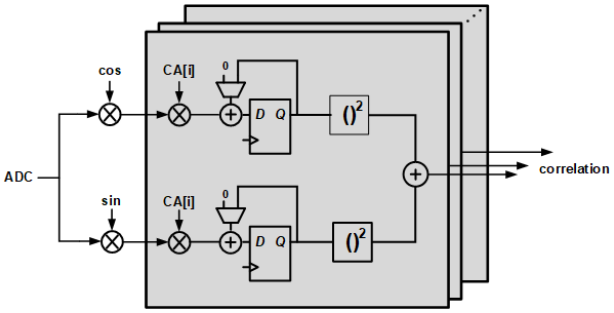

Fig. 2. The Diagram of Parallel Search

2) FFT Search: The linear search algorithm can also be performed by correlating received data with the replica code by circularly shifting the replica code. This circular convolution is simply a multiplication in the frequency domain:

$$R[m] = x[n] \bigotimes CA[-n] = \mathcal{F}^{-1}(\mathcal{F}(x[n]) \cdot \mathcal{F}(CA[n])^*)$$

where fast Fourier transform(FFT) and its inverse(IFFT) are used to calculate the correlation

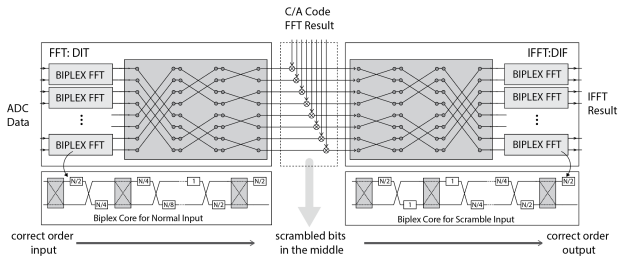

Fig. 3. The Diagram of FFT Search

The diagram of FFT/IFFT generator is in Fig. III-A.2 The FFT/IFFT blocks implemented in a two-stage pipelined architecture. Because of the large number of points needed of each FFT/IFFT operation, this architecture is necessary for reducing the hardware cost. For FFT, the first stage is Biplex FFT which takes input data stream and perform FFT on a subset of data. The second stage is a direct FFT that finish the rest of FFT operations using the output from Biplex FFT.

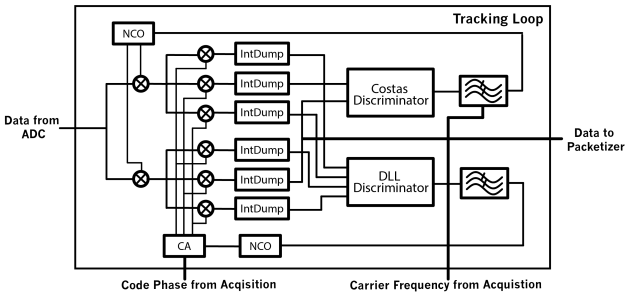

Fig. 4. Block Diagram of the Tracking Loop

For a simple FFT architecture, with correct input bit order the output of is in bit-reverse order. In this FFT generator, because of some bit unscrambling happens between two stages, the output order is not very straightforward. But correct order is necessary to perform IFFT and find where correlation happens. Another way is implement the IFFT in a way that can accept scrambled input but give the result with correct order. In this way, there is no need for extra memory and re-order logic between FFT and IFFT, the latency and hardware cost is reduce.

B. Tracking Loop

The Tracking Loop ensures that the receiver remains locked to the frequency and code phase of the satellite. The loop generates a replica carrier using a Numerically Controlled Oscillator (NCO) that locks onto the phase of the transmitted GPS signal. The replica carrier and received carrier are locked using a Costas Loop. The tracking loop also generates a replica C/A code for the satellite it is tracking. The phase of the replica C/A code is locked to the received signal using a Delay Locked loop (DLL). Figure 4 shows a block diagram of the tracking loop.

1) Carrier Recovery: In order to perform carrier recovery, we utilize the modulation insensitive Costas loop with a Frequency Locked Loop (FLL) assist. The integrated prompt I and Q samples are passed into the Costas and FLL discriminators and then to their loop filters. The Costas discriminator we use is the optimal two quadrant arctangent, and can be represented as follows:

$$\phi_e = \text{ATAN}\left(\frac{Q_p}{I_p}\right)$$

Along with a Costas discriminator for phase error, we have also implemented frequency discriminator. We utilize the optimal four quadrant arctangent discriminator:

$$\omega_e = \frac{\text{ATAN2}\left(Q_p, I_p\right)}{\Delta t}$$

where Δt represents the time difference between two integrated samples of Q_p and I_p. The frequency and phase error is combined through a 3rd order loop filter represented by the block diagram shown in 5. The frequency error sees a 2nd order filter and the phase error sees a 3rd order filter

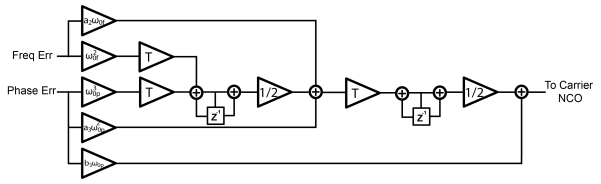

Fig. 5. 3rd Order Loop Filter

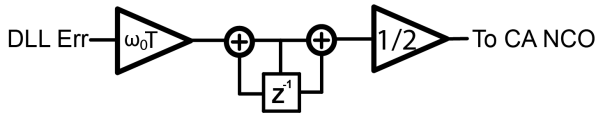

Fig. 6. 1st Order Loop Filter

because frequency is the derivative of phase. The coefficients a_3, b_3, and a_2 are selected based on the results described in [4].

C. Code Phase Recovery

The code phase recovery loop ensures that replica code generated at the receiver is matched to the code phase of the satellite. The code phase tracking loop utilizes 6 correlators per channel, an Early, Prompt, and Late correlator for both the in-phase component and the quadrature component. The Early and Late correlators are spaced half a code phase apart in time, requiring the use of another NCO that is double the frequency of the nominal NCO. The DLL discriminator we use is a normalized non-coherent "Early-minus-Late" power function and can be represented as:

$$E = I_E^2 + Q_E^2 \tag{1}$$

$$L = I_L^2 + Q_L^2 \tag{2}$$

$$\Delta\phi_c = \frac{1}{2}\frac{E-L}{E+L} \tag{3}$$

The DLL discriminator output is passed through a first order loop filter represented by the block diagram shown in 6. We selected to use a DLL bandwidth of 3 Hz. The coefficients a and b were selected based on the bilinear transform of and continuous time first order filter.

$$H(s) = \frac{1}{1+\frac{s}{\omega}} \rightarrow H(z) = \frac{1+z^{-1}}{\left(1+\frac{2}{T\omega}\right)+\left(1-\frac{2}{T\omega}\right)z^{-1}}$$

Implementing the discriminators in the carrier and code tracking loops required the use of arctangents as well as division. Our design does not rely on software to perform these functions. Instead, we use a parameterizable CORDIC to implement them. Since the CORDIC takes many cycles, we also implement a carrier and code recovery finite state machine to coordinate the connections between the CORDICs, discriminators, and loop filters.

D. Ranging

The process of computing user position happens in three steps entirely in software. In general, computation of user position is done using ephemeris data transmitted by the satellites to compute SV positions which, in addition to the computed GPS time, can be used to compute user position, given at least four satellites.

1) Parameter extraction from hardware: The packetizer module in the receiver acts as the bit-level interface to the Rocket core. It is divided into three submodules with specific functions:

- **Parser**: Searches the incoming data stresm for the 8-bit GPS preamble, which identifies the start of a subframe - the basic unit of a GPS transmission, consisting of ten words of 30 bits each. Once the preamble is detected, the next 300 bits are written into registers and passed on through a ready/valid interface.
- **Parity checker**: Each 30-bit word contains 24 data bits (D1-D24) and 6 parity bits (D25-D30) that are used to verify the integrity of the word. The parity calculation involves the 24 data bits, plus the last two bits of the previously-received word, and is calculated as in Fig. 8. The parity checker outputs the result of the parity check on each of the ten words, then passes its data to the next stage.
- **Param extractor**: an extra layer of combinational logic that identifies the subframe, extracts the corresponding ephemeris parameters, and maps them to the processor's memory map.

$$\begin{aligned}
D_{25} &= D_{29}^{*} \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \oplus d_{18} \\
&\quad \oplus d_{20} \oplus d_{23} \\
D_{26} &= D_{30}^{*} \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{18} \oplus d_{19} \\
&\quad \oplus d_{21} \oplus d_{24} \\
D_{27} &= D_{29}^{*} \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{19} \\
&\quad \oplus d_{20} \oplus d_{22} \\
D_{28} &= D_{30}^{*} \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{20} \\
&\quad \oplus d_{21} \oplus d_{23} \\
D_{29} &= D_{30}^{*} \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 \oplus d_{10} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \\
&\quad \oplus d_{21} \oplus d_{22} \oplus d_{24} \\
D_{30} &= D_{29}^{*} \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus d_{13} \oplus d_{15} \oplus d_{19} \oplus d_{22} \\
&\quad \oplus d_{23} \oplus d_{24}
\end{aligned}$$

Fig. 7. GPS parity calculation. [6]

2) Space vehicle (SV) position calculation: SV position calculation is done by using the extracted ephemeris data and solving a set of known equations to correct for orbital motion, relativity, etc [1]. Some of these equations have no close-formed solution, and must be solved iteratively. Doing these calculations accurately is vital for the user position calculation, as knowing the receiver's distance from the satellites is the ultimate goal.

The returned coordinates are in ECEF (Earth-Centered, Earth-Fixed) coordinates, as shown below.

The code converts the hardware elapsed time into "GPS time," or corrected time delay after lock on. It then takes the extracted hardware parameters and does unit conversions and scaling as described in [1] for proper computation, before returning values for user position calculation.

3) User position calculation: The positioning algorithm determines the GPS receiver's location by combining the calculated positions of the tracked satellites and the propagation time for the GPS signal from the satellite to the receiver extracted from the tracking loop. The time is extracted from

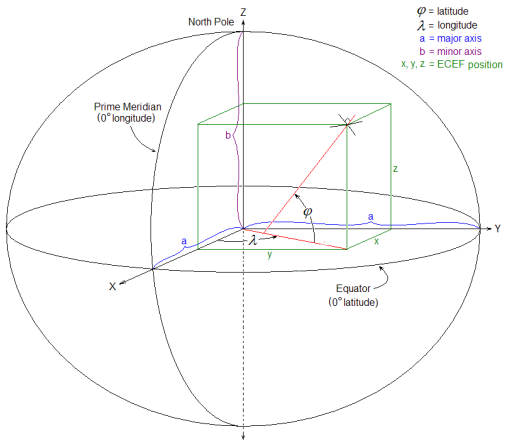

Fig. 8. ECEF Coordinates Source: https://en.wikipedia.org/wiki/ECEF

the tracking loop by keeping track of the time elapsed from a known GPS second that is embedded in the navigation message. This is achieved in hardware by tracking the number of 50Hz bits (20ms each), milliseconds (20ms per bit), CA code chips (0.977us each), and the NCO accumulator in the tracking loop. Taking a snapshot of these trackers will give the GPS time at which the signal was transmitted, since the tracking loop essentially generates a model of the GPS signal it is tracking at the point in time at which it was transmitted from the space vehicle (SV). We can use the approximate altitude of the GPS satellites to approximate a receive time from the transmit time (the bias of this approximation is later solved for). All four of the transmit times (one for each SV being tracked) are then referenced to this nominal receive time. This timing data is then passed to the processor for the positioning calculation.

The receiver location can then be solved for using an iterative algorithm, as described in detail in [1] and also in the doc/Rangind.md in the linked Github repository. At a high level, the algorithm chooses a nominal starting location (ie. (0,0,0)) and time bias with respect to the real receive time (ie. 0s), then uses the approximately measured navigation times and satellite locations to calculate a set of nominal pseudoranges. The pseudorange (PR) in this case is defined as:

$$PR_i = [(x_n - x_{sv,i})^2 + (y_n - y_{sv,i})^2 + (z_n - z_{sv,i})^2]^{1/2} + T_{bias,n} * C$$

where $x_n, y_n, z_n, T_{bias_n}$ are the nominal coordinates and time bias that get updated each iteration, $x_{sv,i}, y_{sv,i}, z_{sv,i}$ are the location coordinates of the ith SV (i = 1,2,3,4), and C is the speed of light. This equation is essentially calculating the distance from the nominal receiver position to the SV position plus a correction term for the time bias. This set of equations, one for each of the four tracked satellites, can be linearized in terms of calculated pseudoranges and time deltas. With a matrix inversion, an update to the nominal location is calculated, along with an error magnitude for the current iteration. The calculated update is applied to the initial nominal position and time bias, and is repeated until the error magnitude of the receiver location is below the desired error threshold.

IV. DESIGN METHODOLOGY

The first step in the design process was the creation of behaviorally-accurate models of the different modules in Python, along with testbenches. This allowed module behavior to be abstracted away from implementation details, and verify the correctness of all algorithms used.

Next, interfaces between modules were established to prevent later conflicts, and CHISEL modules for each subsystem were developed, with their corresponding testbenches. Unit-testing at this level and comparison with the behavioral models served as the primary guideline for model correctness. Real recorded GPS data was used to generate test vectors for the different blocks.

Finally, these submodules were integrated into top-level modules for the two major hardware components of the design - the acquisition loop and the tracking loop. The top-level tracking loop was successfully interfaced with Rocket Chip through a register map. Verilog was then generated and synthesized via Hammer for the integrated Rocket/tracking system. A similar process will be adopted for the acquisition loop, after which full-system verification with ranging software can begin.

V. RESULTS

All subsystems were unit-tested under RTL simulation, and verified using the behavioral Python models we had previously constructed.

The acquisition loop is verified with Verilator and FIRRTL simulator embedded in CHISEL framework. With real GPS data[2], the optimal frequency and code phase is located successfully. In Fig. 9, we are showing the result by searching 40 frequencies with 500Hz frequency steps and 2046 code phases with 8 code phase step.

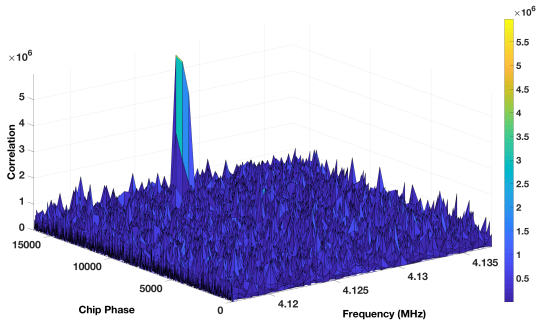

Fig. 9. Parallel Search Result

The tracking loop was verified using an RTL simulation, a python model, and a mixed Scala simulation. The Scala simulation was a cosimulation of the NCO and CA code generator RTL modules run in Verilator driven by a Scala program that calculated the loop discriminator and filter responses. The Scala simulation allows for easy co-integration

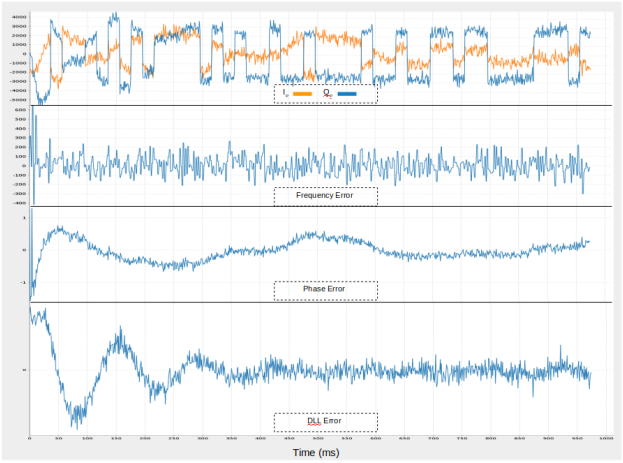

Fig. 10. Tracking Loop Result

of ideal blocks with real RTL modules for comprehensive testing. It also verifies that we are capable of tracking real GPS data and locking to the frequency and phase of the received signal. Fig. 10 shows the tracking loop locking onto frequency and phase of the carrier as well as the code phase of the C/A replica.

The tracking loop was connected to Rocketchip RISC-V core using a register map for the Packetizer module. The generated Verilog was then synthesized using Vivado via Hammer.

Although not fully integrated with hardware, the ranging software framework has some verified testing. Using [5] which provides a set of decoded satellite ephemeris data, we were able to show that the SV position code is able to correctly calculate the position of a SV. We were able to verify that the positioning code was able to correctly calculate the position of a receiver at the BWRC in Berkeley, CA using a set of dummy SV positions and transmit times. The calculated position in ECEF coordinates are as follows: $x : -2691.466, y : -4262.826, z : 3894.033$, which is very close to the dummy target location. The framework for the full ranging computation exists, but requires the fully completed receiver to be fully tested, since time measurements are taken from the tracking loop, along with the navigation message itself. Once the hardware is completed, we will be able to verify the timing measurement itself, along with the positioning calculations all together.

VI. CONCLUSIONS

The CHISEL generator workflow was well-suited to design the architecture of a GPS receiver. The NCO design, for example, came in two flavors - one with an additional sine output, and one without. While this might have necessitated two separate modules in a conventional HDL, one could simply inherit from an existing module in CHISEL, allowing for improved design reusability.

CHISEL's DSP types were an invaluable asset to debugging the design of the various feedback loops inside the tracking blocks. DSP Real type simulations allow us to verify our calculations and ensure that implementation in CHISEL corresponds to the Python model. Fixed Point types are capable of inferring the correct positions of the binary point after various operations are applied. This greatly simplifies implementing DSP computation and makes RTL intuitive to read and understand. Furthermore, CHISEL is implemented as a DSL in Scala, a fully featured modern programming language. We used Scala features to automate calculating various parameters as well as generating plots of simulation results.

The ability to integrate designs with an existing, proven CPU core in the form of RocketChip was invaluable, as the floating-point calculations and highly iterative nature of the algorithm were better suited to a software realization than to a hardware design. The RocketChip workflow includes simple constructs for interfacing hardware with software, enabling the combination of both approaches in a complex system design.

ACKNOWLEDGMENT

The authors would like to thank Professor Borivoje Nikolic, Dr. Sijun Du, and Paul Rigge from the University of California, Berkeley, for supporting and advising on this project.

REFERENCES

[1] Doberstein, Dan. 4. Fundamentals of GPS Receivers A Hardware Approach, Springer, 2012, pp. 62-78.
[2] Samples of GNSS Signal Records. [Online]. http://gfix.dk/matlab-gnss-sdr-book/gnss-signal-records/
[3] Teunissen, Peter. Montenbruck, Oliver. Handbook of Global Navigation Satellite Systems, Springer, 2017.
[4] Kaplan, Elliott. Hegarty, Christopher. Understanding GPS Principles and Applications, Artech House, 2006.
[5] "Raw GPS Signal Samples Data Set for Testing GPS Receivers." Jks. Accessed December 4, 2018. http://www.jks.com/gps/gps.html.
[6] Tsui, James Bao-Yen. Fundamentals of Global Positioning System Receivers: A Software Approach, Wiley, 2000.
[7] H. Hassanieh, Faster GPS via the Sparse Fourier Transform. Mobicom, August 2012.