This is the accepted version of the following article: A flexible FPGA implementation of GNSS signal acquisition circuit using high level synthesis, 13th International Symposium on Computing and Networking Workshops (CANDARW 2025), pp. 365–367 (11/2025), which has been published in final form at https://doi.org/10.1109/CANDARW68385.2025.00070. The article was presented at 16th International Workshop on Advances in Networking and Computing (WANC-16), a workshop of CANDAR 2025.

A flexible FPGA implementation of GNSS signal acquisition circuit using high level synthesis

Naoki Fujieda and Rei Yokoyama
Department of Electrical and Electronics Engineering,
Faculty of Engineering,
Aichi Institute of Technology,
Toyota, Aichi, Japan
nfujieda@aitech.ac.jp (Naoki Fujieda)

Takuji Ebinuma
Department of Astronautics and Aeronautics,
College of Science and Engineering,
Chubu University,
Kasugai, Aichi, Japan
ebinuma@isc.chubu.ac.jp (Takuji Ebinuma)

***Abstract*—A GNSS (Global Navigation Satellite System) receiver is an attractive material to learn signal processing for wireless systems. In digital design using an FPGA, high level synthesis (HLS) has become common as a mean to implement a complicated algorithm rapidly. This paper presents an HLS implementation of the serial search method of GNSS signal acquisition. The implemented code supports any combination of parallelization for the three dimensions: PRN (Pseudo-Random Number) codes, Doppler frequencies, and code offsets of PRN. Learners will understand how to optimize code for HLS and how parallelization policies affect the synthesized circuit and system design. According to our evaluation with a PYNQ-Z1 evaluation board, when the degree of parallelization was set to 528, the proposed circuit completed the signal acquisition process for 32 satellites in 3.095 seconds, which was 507.6 times faster than the circuit without being parallelized. The number of required LUTs was 29,872, which was only 21.1 times larger.**

I. Introduction

A GNSS (Global Navigation Satellite System) receiver is an attractive learning material of signal processing for wireless systems. It determines its own position based on radio waves from earth satellites. Some countries operate their own GNSS systems, such as GPS (Global Positioning System), Galileo, and BeiDou. GNSS is a familiar subject for many learners as they use it daily. Signals are available for free when we get outside. Learning signal processing on GNSS can motivate learners to proceed to more advanced topics, such as GNSS spoofing attacks [7] and lunar navigation systems [9], or to investigate other wireless systems.

One of an important process for a GNSS receiver is signal acquisition. It checks if a signal from each satellite has been received or not. If received, it also gives the Doppler frequency and the spread code offset of that signal. Signal acquisition methods are mainly classified into serial search and parallel search [5]. In the serial search, the correlation power between the input and a replica signal is calculated for each possible pair of the frequency and the offset. The parallel search determines their peak using fast Fourier transform (FFT). The serial search has a simpler, more easily understandable code organization, but requires much larger amount of computation. To achieve both short execution time and simple organization, a possible strategy is to apply some optimization and parallelization to the serial search.

Meanwhile, there have been increasing demands for engineers who understand FPGA (Field-Programmable Gate Array) design and development for wireless systems, especially using high level synthesis (HLS). Recent wireless systems have many requirements: high-speed and low-latency processing, flexibility to adapt them to new protocols and technologies, and reduction of time to market. Characteristics of FPGA and HLS fit them well. Processing signals on wireless systems at a processor or an FPGA is called software-defined radio (SDR). Products for SDR that feature FPGA are available for purchase, such as USRP and Red Pitaya. For an AMD's FPGA, a highly optimized HLS tool called Vitis HLS [10] is available. By combining synthesized circuits with an FPGA SoC platform called PYNQ, software drivers can be written in Python, which enables a more efficient design and verification flow in FPGA development.

Based on the aforementioned backgrounds, we present a new FPGA implementation of a GNSS signal acquisition circuit. Its development principles are threefold. First, the serial search is adopted as a signal acquisition method. Second, the circuit is written in C++ synthesizable with Vitis HLS. Third,

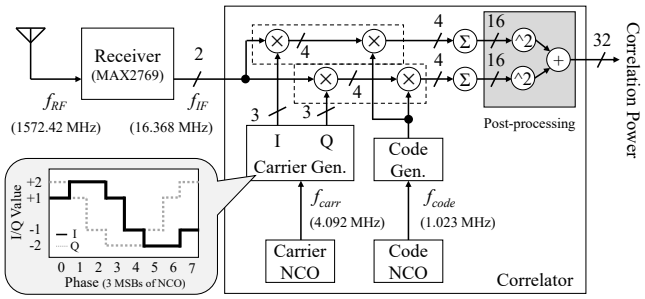

Fig. 1. Overview of correlator in the signal acquisition algorithm.

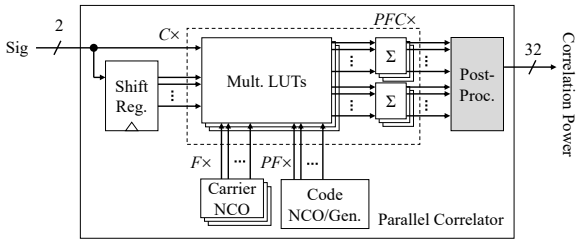

Fig. 2. Organization of the proposed parallel correlator.

the circuit can be verified in real machine of PYNQ. As far as we know, there isn't any implementation according to all of these principles. Existing FPGA implementations of the serial search [4], [6], [8] are written in HDL (Hardware Description Language) and not suitable for grasping its process quickly. GNSS-SDR [2] is a software implementation that targets a processor but its code is very complicated in order to realize real-time processing by software. Hard SyDR [3] has most similar principles to ours, which adopts Vitis HLS and the PYNQ platform. However, it implements the parallel search and includes no viewpoints of learning. This paper describes how the proposed circuit achieves both ease of understanding code organization and moderate, customizable performance. The relationship between the degree of parallelization and the execution time or the amount of hardware is also evaluated using the PYNQ-Z1 evaluation board.

II. GNSS Signal Acquisition

In this research, an open-source C program of the serial search for GPS signals [1] is used as a basis of our implementation. Figure 1 describes the overview of the correlator, a main component of signal acquisition, in this program. A GPS signal of frequency f_{RF} is first down-converted by a front-end receiver to intermediate frequency f_{IF}. The program receives a sequence of intermediate signals, quantized to 2 bits.

The correlator generates two kinds of replica signals: a down-converted carrier and a C/A code, spread code of GPS. The replicas are frequency controlled by corresponding NCOs (Numerically Controlled Oscillators), or a kind of counter circuits. The carrier replica has in-phase and quadrature components, both of which are generated from three MSBs (Most Significant Bits) of the carrier NCO and output as 3-bit signed integers. The C/A code is a pseudo-random number (PRN) sequence of 1,023 bits, predefined per satellite. The code replica is output bit by bit (also called chip) and interpreted as a sign bit. The code completes one cycle in one millisecond.

For each component, the product of the input, the carrier replica, and the code replica are accumulated for one millisecond. After that, the sum of the squares of them is calculated and output as a relative correlation power. If it exceeds a predefined threshold, the signal is considered acquired.

In the serial search method, the correlation power is calculated and compared for each of the possible pairs of the Doppler frequency and the code offset. Due to Doppler shift of a satellite, the frequency observed by the receiver is slightly shifted. The base program [1] set its range as $\pm$ 6,000 Hz and its step as 500 Hz. The code offset is searched through a whole cycle of 1,023 chips, with a step of 0.5 chip. In this research, we target all of the 32 satellites of GPS. Therefore, the number of times of the correlation calculation in total becomes $25 \times 2{,}046 \times 32 = 1{,}636{,}800$. Converted to the number of multiply-add operations, it amounts to about 53.6 billion.

III. Optimization and Parallelization

The development of the proposed circuit was roughly separated into two phases. First, the base program was optimized for HLS implementation, giving exactly the same results as the base program. The optimization includes the use of types of arbitrary bit width, replacement of C/A code generator with a ROM, and replacement of multiplications of input and repricas with table look-up. Then, the multiply-add operations are parallelized for higher performance. The parallelization is done for all three dimensions of the search space: the PRN sequence, the Doppler frequency, and the code offset. In this paper, the degrees of parallelization for the respective dimensions are denoted as P, F, and C. For simplicity, we set the degree for each dimension to a divisor of the number of possible values for the variable. For example, the value of C increases as 1, 2, 3, 6, 11, 22, 31, 33, and so on.

Figure 2 describes the organization of the parallelized correlator. The input is delayed for 8 samples per 0.5 chip of the code offset, using a shift register. The correlator has F copies of the carrier NCO, as they have different increment values with the frequencies. The code NCO and the code generator are not fully duplicated. The value of P affects only the output bit width of the ROM. The effect of Doppler frequency on the code replica is up to one cycle of delay. It is enough for them to be modified partially to output F code replicas of P bits. Finally, C intermediate inputs, F carrier phases, and FP code replicas are given to the parallelized multipliers, that have been replaced with LUTs. The post-processing part is not parallelized: the correlation power is calculated sequentially from each pair of accumulators.

The main loop of the implemented C++ code has only 39 lines even including comments, blank lines, and pragmas. The whole function to be high level synthesized is within 150 lines. The code is very clear even though various degrees of parallelization can be applied.

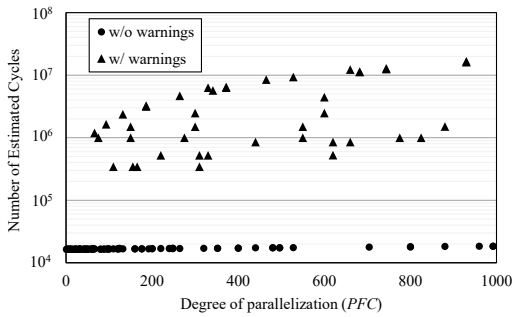

Fig. 3. Estimated number of cycles to complete the target function.

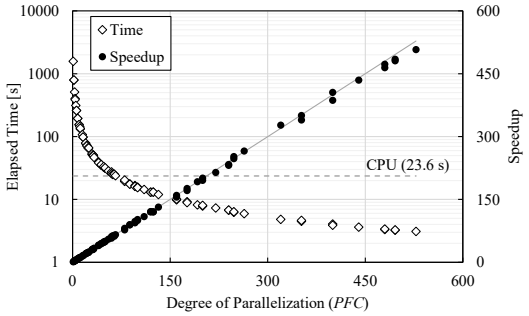

Fig. 5. Time elapsed for the signal acquisition task and speedup ratio.

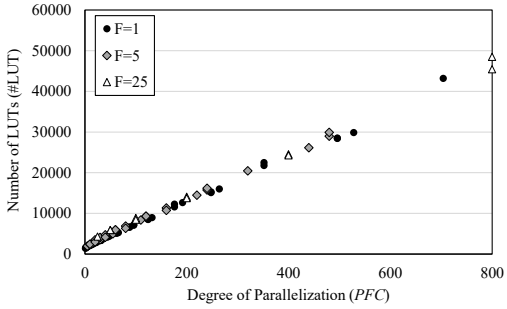

Fig. 4. Number of LUTs required for synthesized circuit.

IV. Evaluation

We evaluate how the values of the degree of parallelization P, F, and C affect the results of high level synthesis, the execution time, and the amount of hardware. The syntheses are conducted on a Windows 10 PC that has a Core i7-14700K processor and 64 GB of main memory, using Vitis HLS 2022.1 and Vivado 2022.1. The target platform is PYNQ 3.0.1 on a PYNQ-Z1 evaluation board. The measurement program of the execution time is written in Python, which runs a task of correlation calculation for all of the 1,636,800 sets.

First, we collected the estimated number of cycles to complete the function of the correlator from the reports of high level synthesis. Figure 3 plots the results. For the all figures in this section, the X-axis represents the total degree of parallelization, i.e., $P \times F \times C$. The Y-axis of this figure is the estimated number of cycles, expressed in the logarithmic scale. In the cases where the expected number of cycles ($\sim$ 16k) were not obtained, or triangles in the figure, an HLS 200–960 warning was recorded that said the tool failed to flatten the main loop. We found that we got a warning when FC was set to a larger number than 62. We could guess that there were some undocumented restrictions to 64 or more times of loop iteration.

Second, we conducted logic syntheses to the circuits that gave the expected number of cycles. Figure 4 depicts the number of LUTs required for the circuits, obtained from the reports of logic syntheses. The cases where F was set to 1, 5, and 25 were plotted as circles, diamonds, and triangles, respectively. The figure clearly describes that it can be basically approximated by a linear function. The same trend was observed in the number of flip-flops. For all of the cases, 1.5 ($\times$ 36 kbit) block RAMs and 2 DSP units were used. The Y-intercept of the approximate line was 1,867, which mainly correspond to the interface and the post-processing circuits. We also observed that the number of LUTs became slightly large when the value of F was large, which was probably due to the replication of the carrier NCOs.

Finally, we measured the time taken to complete the signal acquisition task on real machine. Figure 5 depicts the time (diamond) and its speedup ratio (circle). The execution time is shown in the logarithmic scale of the left axis, while the speedup ratio follows the right axis. The gray dotted line represents the execution time of the CPU execution (23.6 seconds) using the same PC as the syntheses. The figure indicates a linear performance improvement to the degree of parallelization. When $(P, F, C) = (16, 1, 33)$, the execution time became 3.095 seconds, which corresponded to 507.6 times of speedup. On the other hand, the number of LUTs became 29,872, only 21.1 times larger. Considering 53.6 billion multiply-add operations required for the process of signal acquisition, the effective performance of the correlator became 17.3 Gop/s, which was 16.4% of the peak performance under the operating frequency of 100 MHz.

V. Conclusion

In this paper, we proposed an FPGA implementation of GNSS signal acquisition circuit using high level synthesis. It kept a simple code organization, balanced the performance with the amount of hardware or other limitations, and had the performance enough for practical use. The code and the evaluation environment of the proposed circuit are available at https://github.com/nfproc/gpsacq-hls.

We are planning to make necessary preparations to actually use the proposed circuit for education. We will develop reference designs to support the signal tracking and to capture actual GNSS signals. We would like to provide learners of signal processing for wireless systems with these codes, and feed the results back to future development.

REFERENCES

[1] CQ Publishing Co., Ltd. RF World No. 13 Download Service (in Japanese). [Online]. Available: https://www.rf-world.jp/bn/RFW13/RFW13DLS.shtml

[2] C. Fernández-Prades *et al.*, "GNSS-SDR: an open source tool for researchers and developers," in *24th International Technical Meeting of the Satellite Division of the Institute of Navigation*, 2011, pp. 780–794.

[3] A. Grenier *et al.*, "Hard SyDR: A Benchmarking Environment for Global Navigation Satellite System Algorithms," *Sensors*, vol. 24, no. 2, pp. 409:1–409:22, 2024.

[4] D. M. Harris. Hardware/Software Codesign for Wireless Systems. Harvey Mudd Collage. [Online]. Available: https://pages.hmc.edu/harris/class/e168b/

[5] R. Khan *et al.*, "Acquisition Strategies of GNSS Receiver," in *International Conference on Computer Networks and Information Technology*, 2011, pp. 119–124.

[6] P. J. Mumford *et al.*, "The Namuru Open GNSS Research Receiver," in *19th International Technical Meeting of the Satellite Division of the Institute of Navigation*, 2006, pp. 2847–2855.

[7] D. Schmidt *et al.*, "A Survey and Analysis of the GNSS Spoofing Threat and Countermeasures," *ACM Computing Surveys*, vol. 48, no. 4, pp. 64:1–64:31, 2016.

[8] A. M. Shapiro, "FPGA-based Real-time GPS Receiver," Master's thesis, Cornell University, 2010.

[9] T. Tanaka *et al.*, "Systems design results of LunaCube: Dual-satellite lunar navigation system with 6U-CUBESATs," *Acta Astronautica*, vol. 216, pp. 318–329, 2024.

[10] Xilinx Inc., *Vitis High-Level Synthesis User Guide*, UG1399 (v2021.1), 2021.