Accurate Events Synchronization in a System-on-Chip Navigation Receiver

Benoit Priot, Arnaud Dion, Guillaume Beaugendre, Raghuveer Kasaraneni

Accurate Events Synchronization in a System-on-Chip Navigation Receiver

Benoît Priot
ISAE-SUPAERO
University of Toulouse, France
benoit.priot@isae-supaero.fr

Arnaud Dion
ISAE-SUPAERO
University of Toulouse, France
arnaud.dion@isae-supaero.fr

Guillaume Beaugendre
ISAE-SUPAERO
University of Toulouse, France
guillaume.beaugendre@isae-supaero.fr

Raghuveer Kasaraneni
ISAE-SUPAERO
University of Toulouse, France
raghuveer.kasaraneni@isae-supaero.fr

***Abstract*—A System-On-Chip design and synchronization details of a navigation receiver are presented. The architecture of the GNSS receiver is easily modifiable and offers the capability of accurate time management, thanks to the use of a co-design approach. The purpose of such a platform is to allow real time validation of research algorithms. A secondary application is education, as this platform can be used to study signal demodulation and navigation.**

The receiver is fully functional, but further developments are still undergoing. Results demonstrate accuracy, flexibility and ease of use of the system.

***Index Terms*—GNSS receiver, System-On-Chip**

I. Introduction

Event and timing management in a Global Navigation Satellite Systems (GNSS) receiver is critical. Indeed, the standard position, velocity and timing (PVT) solution is based on the propagation delay estimation from the receiver to each satellite in view. Thus, GNSS receivers typically involve high sampling frequency, event synchronization and low latency closed control loops.

Designing a real embedded receiver to study algorithms is a complex task for most research laboratories. There is two usual solutions to tackle this problem : the Field-Programmable Gate Arrays (FPGA) based System-On-Chip (SOC) or pure software using SDR approach. Due to the implementation of matrix of binary operators, architectures based on FPGA allow an interesting concurrent and real-time signal processing [1]. The latency of computations in an FPGA is completely deterministic by their very construction, downto a clock cycle. However, designing applications to be ported on FPGA is usually complex and lacks the flexibility of software application for fixed architecture processor. The typical solution is then to use the Software Defined Radio (SDR) approach where all the computations are performed by high performance/high frequency Digital Signal Processors (DSP). Software receivers are already proposed to scientific or industrial community [2]. However, as the architecture is fixed, they are not suited when targeting highly integrated embedded applications or intended for performance, such as power consumption, memory usage or high computations. Whatever the performance of the DSP, it is limited by the fixed number of operators and by the bandwidth of the memory [3]. The classical approaches of hardware (HW) and software (SW) design are very different, as are their design languages : VHDL and C. We used a modern approach based on SystemC language [5] and High Level Synthesis (HLS) tools in a coherent and common design flow for HW and SW. Such design environment allows to emulate both HW and SW module execution before implementation. It is a great advantage for validating and debugging, as well as making it evolve rapidly according to the emulation results.

The purpose of this work is to develop a GNSS platform for research labs and academic institutions. It offers various possibilities:

- Critical management of time which is the core of a GNSS receiver,
- Full access to the architecture,
- Rapid and easy reconfigurations of the architecture,
- Validation on HW targets of the algorithms developed on SW (Matlab or Octave) receiver.

First, a review of the co-design flow and the distributed GNSS architecture will be presented. We will demonstrate the versatility of the approach, from algorithm to architecture. Then a description of the innovative time management and illustrative results will be shown. As stated previously, this is the core of a receiver, and control loop latency should be bounded as tight as possible.

II. System design

In order to create a flexible distributed architecture for a bi-constellation (GPS/Galileo) GNSS receiver, new seamless co-design and high level synthesis tools have been used. The receiver, so-called low-level layer, supports the demodulation of signals, the acquisition and tracking of satellites in view, and leads to the generation of pseudo-ranges (and Doppler frequencies) between the receiver and the satellites. The navigator, so-called high-level layer, converts the receiver-to-satellite pseudo-ranges into PVT solutions. From this generic

Funded by Occitanie region.

architecture, a standard receiver/navigator is implemented. It is designed to operate with GPS L1 C/A and Galileo E1 signals. The development, based on SystemC language [5], is supported by the implementation of a sequential functional simulator in Matlab or Octave (Matlab compatible GNU software). This flow is presented in fig. 1. From the co-design architecture developed on dedicated third party tools, a real-time implementation of the standard receiver/navigator is performed on a hardware platform. Such implementation is based on existing development boards, i.e., the Zedboard (based on Xilinx Zynq-7000), with a radio-frequency (RF) front-end extension board. Such generic development boards are therefore inexpensive. An earlier version of this receiver has been used to assess the performances of computational error mitigation techniques [6] due to current leakage in components or due to low battery level.

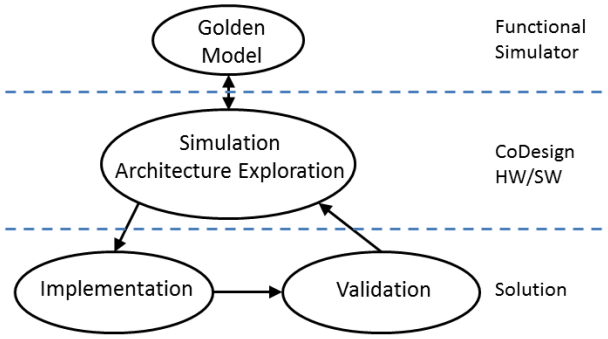

Fig. 1. Seamless design flow.

A simplified block diagram of the architecture considered in this contribution is shown in fig. 2, with the following blocks:

1) RF Front End which provides samples coming from either an antenna or a file to the system
2) The Data IQ Handler which hands out the incoming samples to the appropriates blocks.
3) The Acquisition which searches for satellites from the received signal
4) Tracking loops which follow satellite signals
5) The Manager deals both with acquisition and tracking processes. It also initializes and monitors the tracking loops.
6) The Measurer which converts observable coming from tracking loops into raw measurements (pseudo-range, Doppler frequency, carrier phase), and the Navigator which calculates the PVT solution.

SpaceStudio is a C/C++ framework created by SpaceCodesign for developing applications to speed-up performance using CPU and FPGA without knowing the underlying hardware infrastructure of these technologies [7]. The development of applications is intuitively partitioned to target heterogeneous computing platforms (e.g., System-On-Chip). Designers explore, analyze, profile, and validate designs using the SpaceStudio solution. Using SpaceStudio co-design, we performed the partitioning HW / SW with the SystemC language that

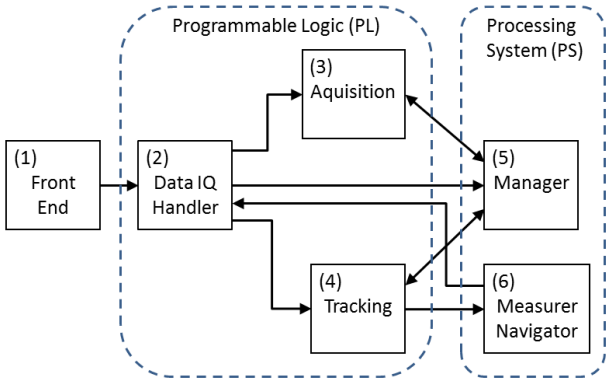

Fig. 2. block diagram of the architecture.

allows to generate IP with Vivado High Level Synthesis. This novel implementation technique offers more flexibility as no VHDL coding is required except for the interfaces with the front-end. Thus, except for loop filters, the tracking loops are HW-based. Fast computation capabilities (acquisition Fast Fourier Transform (FFT) and channel tracking) are supported by the HW part (PL). High level computation algorithms with less time constraints (loop filters, raw measurement, navigator) are supported by the SW part using a softcore processor (PL) as well as the ARM processor (PS).

III. Event and time management

Events refer to a given process of the receiver associated to a specific time. They can be related to a specific temporal characteristic of the GNSS signal or to a time-based functionality of the receiver. Based on the architecture of the system, events can be classified into four classes: new sample reception, acquisition and tracking channel initialization, tracking, and measurement.

A. New sample reception

Compared to the architecture presented in [8], the novelty of this project is the direct connection of the front-end to tracking loops. Indeed, no buffer is present at the input of the tracking channels. In addition, the tracking loops are parallel and autonomous. Due to this design, the computational load of the ARM processor is reduced as it only needs to configure the different tracking channels (for initialization and measurement process) and to retrieve the observations when available. The direct connection also implies that samples must be processed as soon as they arrive, because a single loss of sample will strongly affect the performance of the receiver.

In our architecture, the Data IQ handler block is used to manage and to dispatch the incoming samples from the GNSS receiver front-end. The Data IQ handler updates a sample counter register, referred to as N_s, which is accessible for all other blocks requiring time synchronization. The receiver reference time t_{rx} is then defined by:

$$t_{rx} = t_{rx}^0 + N_s/F_s \quad (1)$$

where t_{rx}^0 is the initial reference time referenced against GPS time, and F_s is the sampling frequency.

The sample counter is crucial for the synchronization and timing between the different blocks because all the other events will be related to a specific sample, as described further.

B. Acquisition and tracking channel initialization

In our architecture, we use a single acquisition block for the whole tracking channels. First, the signal is stored over a fixed period of 8.192ms. The value given by the sample counter for the first recorded sample is also memorized. This value is referred to as N_s^{acqui}. Then, we compute the correlation between the signal and a local replica using a FFT-based correlation implementation. This allows to estimate the code delay τ, Doppler frequency f_d and signal-to-noise ratio. In case of detection, this information is sent to the channel manager in order to compute the value of the sample counter corresponding to the beginning of a code PRN of the satellite. This value, referred to as N_s^{init}, is computed following the equation:

$$N_s^{init} = N_s^{acqui} + F_s(kT_c^{rx} - \tau) \tag{2}$$

k is an arbitrary integer chosen such as:

$$N_s^{init} > N_s + F_s T_{computation} \tag{3}$$

where N_s is the current value of the sample counter, and $T_{computation}$ is the computation and communication delay of the channel manager.

This equation takes into account the estimated duration of the received PRN code affected by the Doppler effect. This value is referred to as:

$$T_c^{rx} = T_c/(1 + f_d/f_{L1}) \tag{4}$$

where T_c is the standard duration of a PRN code (1 ms for GPS L1 C/A code), and f_{L1} is the value of L1 frequency. Then the manager initializes a tracking channel by sending the PRN ID of the satellite, the Doppler frequency and N_s^{init} to the tracking block. Satellite signal tracking automatically starts when the value of the sample counter register is equal to N_s^{init}.

Using Equation (2) and (4) it is possible to bound the maximum time lapse for this initialization, depending on the precision of acquisition results and the dynamic of the vehicle. For example, with a Doppler frequency error of 200Hz and a time lapse of 5s, the error on the estimated starting time of the code PRN is about $0.635\mu s$. This value is lower than the chip period, which is $0.9775\mu s$ for GPS L1 C/A and Galileo E1-B/C signals. Thus the tracking channels can be initialized. Also, the dynamic of the vehicle can also be taken into account, as an acceleration of 1m/s^2 induce a Doppler rate of 5.255Hz/s, and a maximum Doppler frequency error of 26.3Hz after 5s.

Therefore, this functional design allows us to take into account the non-deterministic delay resulting from computation process and from communications between blocks while ensuring the synchronization of tracking loops with the incoming signal.

C. Tracking

In our architecture, the tracking loops are divided into four functional blocks as shown in the diagram presented in fig. 3. The different blocks are defined by their activation time. They are executed following a cascaded chain of events, and have different working frequencies. These blocks are defined as follow:

1) the NCO_Correlator, activated at each new sample coming from the Data IQ handler,
2) the Integrator Synchronizer, activated at each end of code coming from the NCO_Correlator,
3) the LoopEstimator, activated at each integrator output coming from the Integrator Synchronizer,
4) the DataDecoder, activated at each end of bit coming from the Integrator Synchronizer.

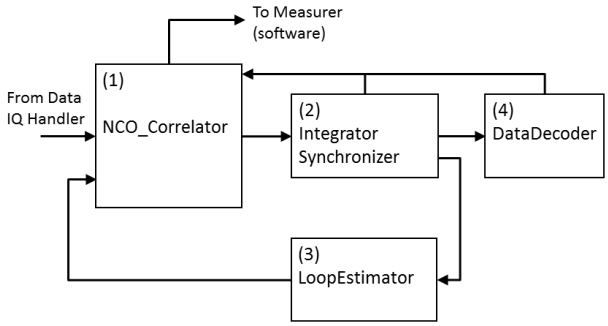

Fig. 3. block diagram of the tracking channels.

The NCO_Correlator block is composed of three functional blocks: carrier NCO, code NCO and correlators. At each new sample, the carrier phase ϕ and code phase φ_c of the respective NCO are incremented. Once the code phase reaches the end of the PRN code table (1023 chips for GPS L1 C/A signal), an End of Code event is triggered, and outputs of the correlators are sent to the Integrator Synchronizer block.

The Integrator Synchronizer block is composed of the bit synchronization and of the integrators. At each End of Code event, the value of the code counter N_c is incremented and is used to trigger two events. One is the End of Integration event, which triggers the send of the outputs of the integrators to the LoopEstimator. The second is the End of Bit event, which triggers the send of the bit value to the DataDecoder block, and triggers the reset of the code counter.

The LoopEstimator block is composed of the phase and delay discriminators, and of the phase and delay loop filters. It updates the commands of the NCO, and estimates the Doppler frequency. It does not generate any event as the NCO processes the latest commands values available using dedicated registers. A delay can be observed between the End of Code from the NCO_Correlator block and the update of the NCO command, which could be reduced by the optimization of computation in the tracking loop.

The DataDecoder block main task is to decode the frame of the navigation message in order to retrieve information

such as ephemeris data of the satellites and time, using the bitstream from Integrator Synchronizer block. It also includes a preamble detector, a bit counter N_b, and a frame counter N_f. The frame counter is updated thanks to the timing information of the navigation message.

D. Measurement

The purpose of measurements is to retrieve the valuable information associated to the same reference time t_{rx} from all the tracking channels. This information is sent to the Measurer Navigator block in order to elaborate raw measurements, in particular pseudo-range measurements. The pseudo-ranges are computed following the equation:

$$PR = (t_{rx} - t_{sv})/c \tag{5}$$

where c is the speed of light, t_{rx} is the receiver reference time associated with the measurement, and t_{sv} is the satellite transmission time, given by:

$$t_{sv} = T_{chip}\varphi_c + T_c N_c + T_b N_b + T_f N_f \tag{6}$$

where T_{chip} is the chip period, T_c is the PRN code period, T_b is the bit period, and T_f is the frame period. As described previously, φ_c comes from NCO_Correlator, N_c comes from Integrator Synchronizer, and N_b and N_f come from DataDecoder.

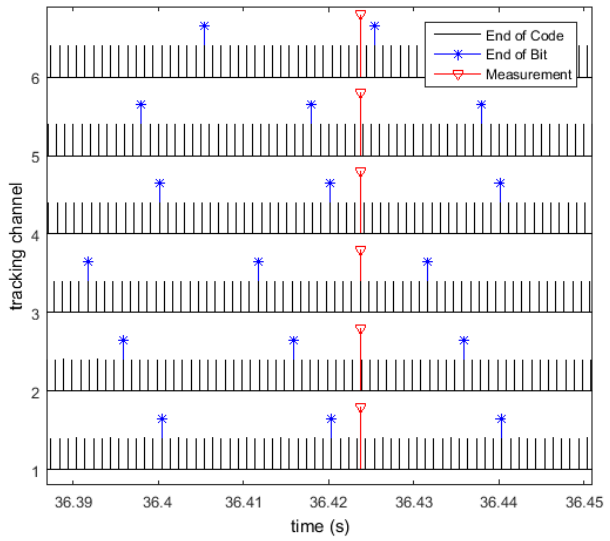

Fig. 4. Event time diagram.

In order to synchronize the measurements for the whole tracking channels, we use the Data IQ handler block to trigger a Measurement event when the value of the sample counter register is equal to N_s^{meas} by using a Do_Measurement flag. Then the NCO_correlator block of each channel, which is the only block of the tracking channel activated at each sample, is tasked to fetch the data and send them to the Measurer Navigator block.

However, due to the cascaded architecture of the tracking loops, a challenge is to wait for the Integrator Synchronizer and the DataDecoder to finish their computation, while not blocking the NCO_Correlator process. The solution selected for our architecture, illustrated in fig. 5, is described as follow: First, the NCO_Correlator saves the value of the carrier phase and code phase. Then, in case of End of Code, we set a Measurement_Enable flag to zero, and resume NCO_Correlator process. When Measurement_Enable is set to one, the NCO_Correlator collects remaining data and send it to the Measurer Navigator block. This process ensures the coherency and the datation of the raw measurements, even in case computation or communication delay.

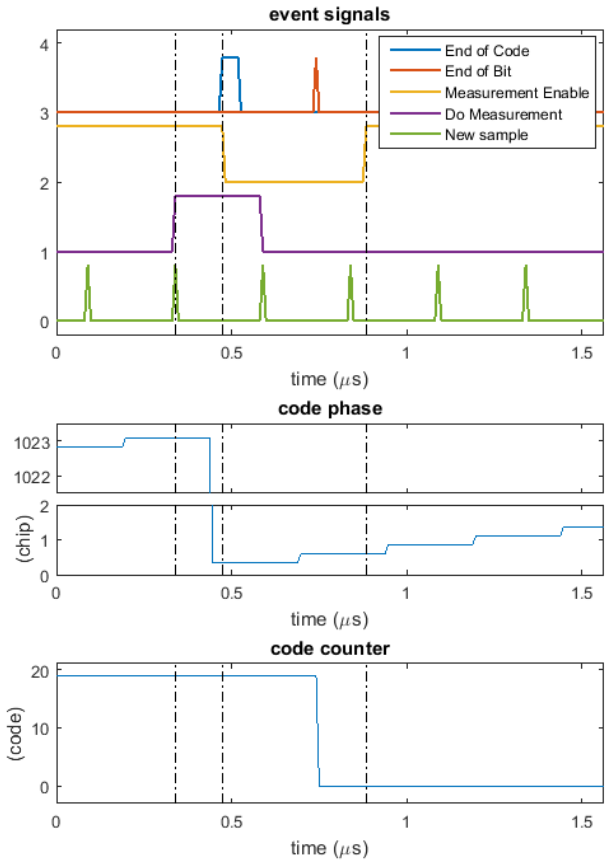

Fig. 5. Measure event time diagram.

IV. Results and Discussion

The receiver has been completely implemented with six GPS channels and is fully functional. It can be extended to as many channels as can be fitted in the HW. For simulation and debbuging purposes we used a recorded signal. We sampled the GNSS signal with an SBX board mounted on an ETTUS USRP X310. We are working to integrate the Analog Device AD9361 RF board with the FPGA.

The test of this architecture has shown the capability to initialize the channels from the results of the acquisition, regardless of the computing delay of the acquisition. The results obtained when using tracking loops (integrators, discriminators, Doppler frequency and C/N0) are shown in fig. 6, when

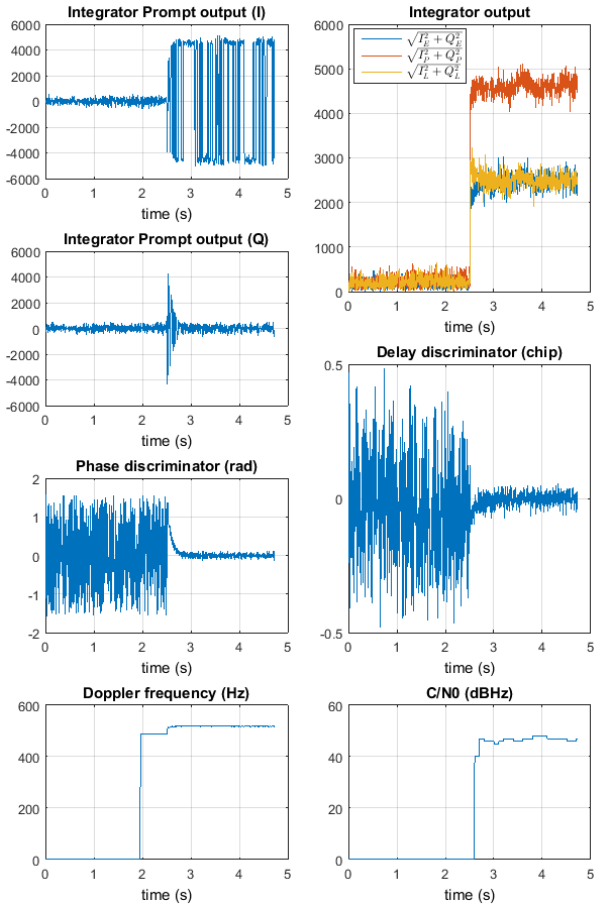

Fig. 6. Test results of the tracking loops.

initialized with acquisition outputs. The acquisition process start at 0s. At approximately 1.9s, the acquisition provides its results to the manager which configures the tracking loop. We can see that the Doppler frequency is initialized. At 2.5 s, the tracking loop starts the tracking process. From that moment, the loop converges and work properly. In this test, the initialization delay and the tracking start delay have been increased intentionally, in order to validate channel initialization.

In addition to the development of the aforementioned co-design architecture, one of the expected outputs is a Vivado project (or EDK) compatible with the hardware platform, targeted and validated on such platform.

V. Conclusion

This work demonstrates that co-design GNSS architecture is of great interest to fix the complexity and non-flexibility issues related to the use of FPGA. It offers both the capability of a critical management of time and of an open and easily modifiable architecture. The high level modelling and simulation allows early key decisions and shortens the development process [4].

Acknowledgment

This work has been funded by Occitanie region.

The authors would like to thank Dr Gaël Pagès for the work on the software manager, as well as Space Codesign for the technical support. The authors would also like to thank Dr Anne-Emmanuelle Priot for her help and support.

References

[1] F. B. Muslim, L. Ma, M. Roozmeh and L. Lavagno, "Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis," in IEEE Access, vol. 5, pp. 2747-2762, 2017.

[2] C. Fernandez-Prades et al., "GNSS-SDR - An open source Global Navigation Satellite Systems software-defined receiver," Available online at https://gnss-sdr.org/, Accessed: November 15,2018.

[3] C. Rowen, "Engineering the complex SOC," Prentice Hall, 2004.

[4] A. Dion, E. Boutillon, V. Calmettes, E. Liegon, "A Flexible Implementation of a Global Navigation Satellite System (GNSS) receiver for on-board satellite navigation," 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP).

[5] T. Wieman, B. Bhattacharya, T. Jeremiassen, C. Schroder and B. Vanthournout, "An Overview of Open SystemC Initiative Standards Development," in IEEE Design & Test of Computers, vol. 29, no. 2, pp. 14-22, April 2012.

[6] M. M. Hafidhi, E. Boutillon, A. Dion, "Demo: Localisation in a faulty digital GPS receiver," 2016 Conference on Design and Architectures for Signal and Image Processing (DASIP).

[7] L. Filion, M.-A. Cantin, L. Moss, G. Bois, M. Aboulhamid, "A SystemC Framework for Fast Exploration of Hardware/Software Systems," 2007 Conference on Design and Verification , San Jose, 2007.

[8] M. Majoral, C. Fernandez-Prades, J. Arribas, "Implementation of GNSS Receiver Hardware Accelerators in All-Programmable System-On-Chip Platforms,".