MANTECH Publications

Journal of Power Electronics and Drives
Volume 8, Issue 3, September-December 2023

System on Chip Based RTC in Power Electronics

R. Dorothy[1], Sasilatha. T[2]

Department of Electrical and Electronics Engineering (Marine)

AMET University, Chennai, India

Email: *eei@journal.uad.ac.id*

Abstract

Current control systems and emulation systems (also known as Hardware-in-the-Loop, HIL or Processor-in-the-Loop, PIL) for high-end power-electronic applications typically consist of a large number of components and interconnecting buses. These components typically include a micro controller for communication and high level control, a DSP for real-time control, an FPGA section for fast parallel actions and data acquisition, multiport RAM structures or bus systems as an interconnect System-on-Chip (SoC) technology integrates a significant number of these functionalities into a single semiconductor chip. This provides the benefit of a decrease in both space and costs, in addition to an increase in the speed of communication within the company. These kinds of systems become increasingly significant not only for scientific study but also for applications in industry. The System-on-Chip (SoC) that serves as an example here combines a fast processor system (FPGA) with a Dual-Core ARM 9 hard processor system (HPS), and there are also fast interlinks between these individual components. For SoC systems to be able to provide real-time control and emulation, the accompanying software and firmware principles need to be carefully considered. This paper explains how to use the resources of the SoC in the most effective way possible and analyses the difficulties that are generated by the internal structure of the SoC. The utilisation of asymmetric multiprocessing is the primary concept here: One of the cores operates in hard real time using a bare-metal operating system. On the second core, a "real-time" Linux operating system handles service operations and communication respectively. The Field Programmable Gate Array (FPGA) is utilised for flexible process-oriented interfaces (such as

A/D, D/A, and switching signals), quasi-hard-wired protection, and the exact timing of the real-time control cycle. This method of implementation is well-known and is even occasionally proposed; but, to the best of the author's knowledge, it is only seldom put into practise and rarely documented within the context of demanding real-time control or emulation. In the article, the method of implementation is broken down in great detail, including the process interfaces, and the study also addresses the benefits and drawbacks of the selected idea. The outcomes of the measurements provide evidence of the solution's qualities.

***Keywords**: SoC, control, cache interference, multiprocessing*

INTRODUCTION

A real time clock (RTC) module, a crystal oscillation circuit, and a voltage supply circuit are all components that may be found on a system-on-a-chip. Both the RTC module, which provides time capabilities, and the crystal oscillation circuit, which generates an oscillation, are connected together in this design. The crystal oscillation circuit and at least a section of the RTC module can each get their own supply voltage thanks to the coupling that occurs in the voltage supply circuit. The use of multicore architectures for real-time control applications is a tough subject that requires the developers to take great attention when decomposing the problem in order to make the most of the parallel processing that is available [1-2]. In the context of fast prototyping systems and/or low effort development projects, it is unusual for this sort of optimization to have been recorded up to this point. In a multiprocessing context, the coupled employment of two kernels places an excessive demand on the implementation of the control code and the code for communication.

It is difficult to anticipate how the activities will interact with one another and what benefits will be obtained. There is a possibility that interrupting service procedures will result in intriguing side consequences. This work proposes a general split of the service activities and the control tasks, which ultimately results in a tight function-based parallelization and a virtually full decoupling of the real-time control cycle from ancillary services. The real-time core directly interacts with the FPGA, which is a proven approach [3, 4]. This allows for precise timing. The first half of this discussion will focus on analysing the requirements for

real-time control systems. Following this, a structure that satisfies these prerequisites will be constructed. Interfaces for connection to processes, such as the configuration of a test bench, are described here. Last but not least, the functioning of the platform and its peripherals is proved by benchmark testing and measurement results collected at a laboratory test set-up. In [5], which also provides approximately 70 more citations, there is a full explanation of a comparable method that does not require real-time functionality. As a result, the focus of this work is limited to a few carefully chosen citations [6].

Because of the necessity for quick prototyping and small series control systems, extra constraints have been placed on the control system's adaptability in terms of both its hardware and its software. During the process of development, the computational performance has to be increased; the requirement to optimise for performance concerns should be avoided; a user-friendly code should be supplied; and extra resources for debugging jobs may be necessary. In addition to this, it is essential to implement standard functions out of the box, such as measurement, pulse-pattern generation (such as PWM), trace functionality, and the user interface [7]. This means that an individual implementation is not required.

On the side of the process interface, the requirements might shift fast, and it is essential that it be easy to add new measures or other kinds of physical measuring interfaces at any moment. On the other hand, both the system and its associated expenditures have to be kept to a minimum. It's possible that a modular, flexible idea with a variety of standard modules is the best option to go with.

When testing new control algorithms, there is a potential for the power electronic components to be put at risk. Therefore, it seems sense to provide a safety layer that is HPS-independent but also quick and adaptable. An FPGA-based limit check enables reactions in a few microseconds, which is quick enough to safeguard sensitive semiconductors. Additionally, this type of check may be designed separately from the control code. The entirety of these needs may be summed up as follows, and the newly presented system will be capable of meeting them all:

At this time, the majority of SoC design data layout is also done manually, which presents two major challenges: (1) the development time is significant, which is not acceptable for

meeting the requirements of the modern day time to market, and (2) the quality of the solution varies depending on the expertise of the designer.

The Proposed Solution

SoCs are built with the ability to start up from a single source. In comparison to the present heterogeneous multi-chip control systems, this results in a substantial simplification of the boot procedure as well as firmware updates [8]. Utilizing a "high-level" operating system like Linux is the most effective method for achieving a user-friendly and straightforward integration of tasks. A "real-time" edition of Linux is a smart choice since it provides increased speed and a more predictable response time. When applied to this scenario, the definition of "real time" calls for a more in-depth discussion:

• While "real-time" cycles of one hundred microseconds or less are often required for power electronics systems, the bare-metal technology employed here and detailed later on in this work readily hits 20 microseconds (if the control code is compact enough)

• “real-time” Linux prevents the long-term blocking of interruptions by service activities (such as network traffic), but the operating system does not provide the complete prioritisation of particular processes, which is necessary for achieving a low cycle time. When seen from the perspective of power electronics, this does not constitute "real time."

• Commercial real-time operating systems offer a real-time functionality that is comparable to that which is described in this article; however, these systems are typically unable to avoid all interruptions and instead must focus their attention entirely on the control code.

It is abundantly clear that, from the point of view of usability, a "high-level" operating system such as Linux, which has a large number of pre-fabricated services and functions, is better. However, the relationship between these service duties and the most critical real-time control process is a difficult one. It would be ideal if the control process had control over all interrupt services inside the operating system; however, this would make it more difficult to integrate service functions. In addition, reprogramming all of the components—including the firmware, software, and FPGA structure—would be difficult to do and require a lot of work [9]. The approach that is being utilised here incorporates the benefits that are offered by both

systems (see Figure 1). The "real-time" Linux operating system and many service operations are being provided by one of the cores. These include updates to the software and firmware through TCP/IP, features for controlling the process, such as set-point value transfer, and logging capabilities (e.g. streaming measured data to network-attached storage, NAS). A bare-metal programme that contains the control duties is run by the second core of the computer [10]. It is not directly connected to any gadgets from the outside world (with the exception of a very fast FPGA link to transmit data in both directions for process interaction). As a direct result of this, the control latency is almost completely independent from the service operations that are now being carried out. On the other hand, as you will see in the next example, the control execution is affected by the interference that cannot be avoided because of the shared access to the Level 2 cache and the on-board SD RAM (Figure 2). In the following section of this study, the quantity of interference, as well as probable explanations and countermeasures, will be studied.

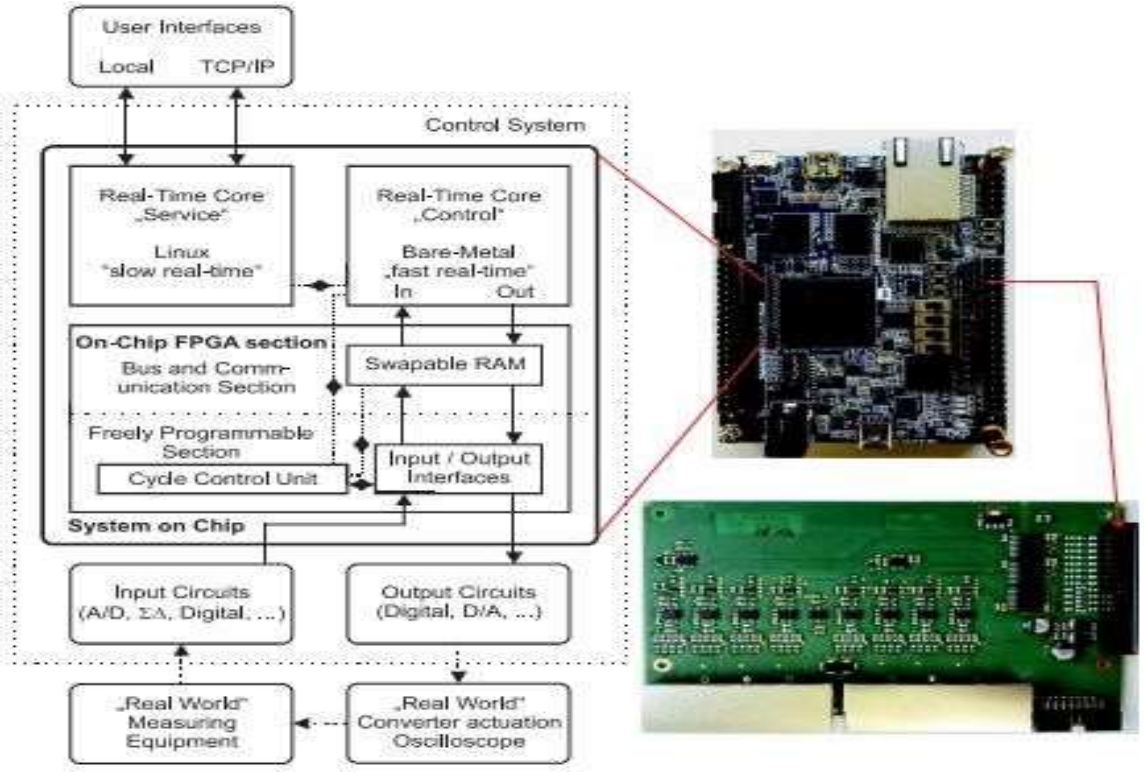

Figure 1. left: Basic structure of the used SoC platform, right: photo of DE0-Nano-SoC andPeripheral (8 Channel ADC-board)

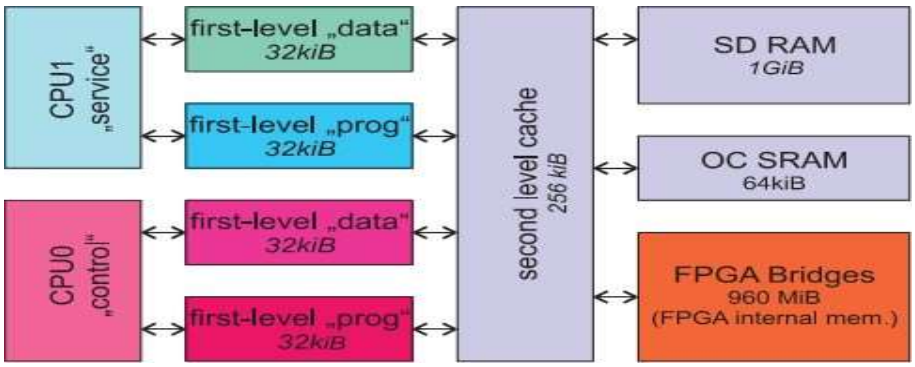

Figure 2: Cache and interface structure of the SoC

Memory Interfaces and Interferences

Both of the ARM9's CPUs share a shared second-level cache, which is connected to the memory through the ARM9 architecture (cp. Figure 2). Since the on chip RAM, also known as OC RAM, is considerably quicker than the external SD RAM, it is the one that is chosen for the data interchange between the two CPUs. One gigabyte of RAM is provided via the SD RAM. The remaining 704 megabytes are set aside as shared memory (which is directly controlled by the bare-metal OS), and they are used almost exclusively for trace data. 256 megabytes are set aside for the Linux operating system, and 64 megabytes are set aside for the bare-metal operating system. The activities that are carried out in relation to the programme and data that are stored in the first-level cache that is connected with one core are completely independent of the actions that are carried out by the other core. Unfortunately, it is necessary for the control to access the FPGA-Bridge for control purposes and the SD RAM for trace functionality during each and every control cycle (typically ranging from 20 microseconds to 500 microseconds, depending on the type of control). Both of these functions are necessary for the control. In order to accomplish this, interaction with the second level cache is required, which results in interference with the service core. According to our observations, the CPU load of the service core never rises over 1%, which results in a little amount of access to the second level cache.

Executing code that needs a significant amount of access to the SD RAM, on the other hand, may generate significant interference between the control activities and the service duties. An illustration of such interference is provided in Figure 3. In advanced CMOS technology, optimization of the SRAM array structure was performed with the goal of improving energy efficiency [6].

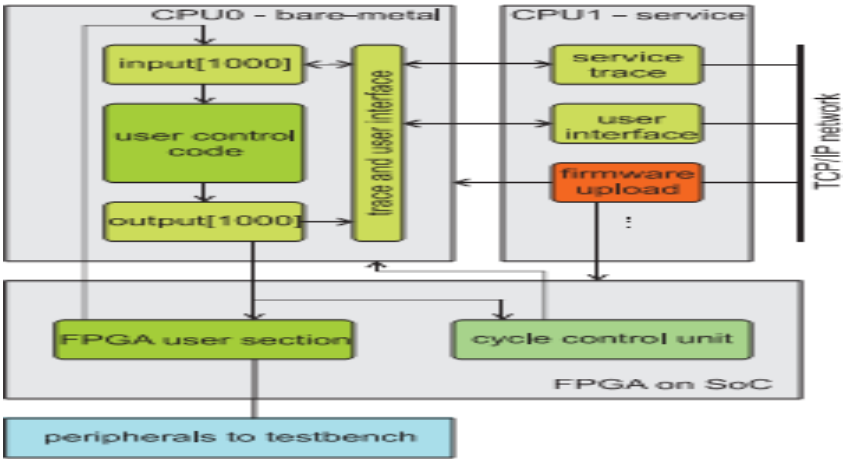

Figure 3. Example for such interference

During typical operation, the time required to complete the execution of the chosen control code (cp, t < 0 s) ranges from 28 µs to 34 µs. This time includes the acquisition of measured data and the functioning of the trace. When in this condition, the control code will place the measured data in a circular buffer in preparation for its subsequent storage in a data file (which is then provided by the service core). It is essential to take note that the configuration of the system provides for adjustable cycle times (e.g. for random PWM or PLL-controlled cycle-time adjustment in normal PWM mode). As a consequence of this, every measurement that has been kept has its own (and potentially unique) cycle time. It is expected that the data file will contain both a pre-defined pre-trigger sequence as well as a post-trigger sequence if a trigger event occurs.

A Linux service function will get the data ready for storage in a data file when the trigger event occurs at t=0 s. In order to accomplish this, it has to locate the moment at which the pre-trigger sequence begins. This cannot be inferred from the cycle time due to the fact that the cycle time is subject to change. The service function is required to do a cumulative addition of all cycles moving backwards into the past until it meets the pre-trigger duration that was specified. A significant volume of data is read extremely quickly from the storage device's dynamic random access memory (SD RAM) (the service core has nothing else to do and provides high calculation power). As a consequence, there are many cache misses, and the second-level cache must be loaded from the side of the service core. Because of this, the amount of time required for the bare-metal core to access memory grows, and the amount of

time spent executing the control code reaches a brief high of up to 68 µs. This is more than twice the average value, and it is very possible that it will violate the permitted control cycle duration, which will result in the system tripping. The data access rate will eventually settle into a more manageable range after this startup of the tracing service. It is helpful in this situation to have write access to slower storage devices or network connections. The interference with the cache is still present, but it is now much easier to tolerate. As a direct result of this, service jobs in processes with significant data access need to have their pace purposely slowed down. This does not constitute a major performance detriment: The real-time needs or priorities of the service core are not very stringent. In addition, it is not difficult to incorporate pauses into self-made service routines. On the other hand, it is obvious that software running on the Linux core might cause interference with the real-time control; therefore, software components ought to be rigorously vetted before being used: Even a brief load surge lasting only a few microseconds might cause the real-time control code's real-time criteria to be violated.

The Peripheral Concept

In the field of power electronics, where many kilowatts are involved, two different signal standards are typically used. Current signals in the ranges of ±20 mA to ±200 mA are most frequently employed as standards for analogue numbers. Signals that are based on current have a high noise immunity, which makes it possible to make connections over long distances even in areas with high EMI levels. There is a wide variety of sensors available, including those for isolating current and voltage probes, in addition to temperature sensors. Fiber optics offer a quick and easy solution for the transmission of digital signals such as those used in communication, digital signal transmission (for example, sigma-delta-modulated signals), and PWM signals. They are almost completely immune to EMI and their transmission latency is almost completely independent of the cable length, which enables exact timing. In addition to this, they offer virtually complete isolation, eliminating the need for extra galvanic isolations.

To handle multiple tasks a modular platform is developed. Each of the two GPIO slots can be equipped with an adapter- board for four of the following peripheral modules:

- 8 channel current input analogue to digital converter
    - Several signal standards up to ± 200 mA (directly compatible with common

currentsensors)
  - Up to 1 MSPS-1µs sampling time
  - 12 bit resolution
  - Real parallel sampling of all 8 values

- 8 channel current output digital to analogue converter
  - Several signal standards up to ± 100 mA
  - Up to 1 MSPS-1µs sampling time
  - 12 bit resolution
  - Real parallel output of all 8 values

- 8 channel versatile link fibre optic
  - Input/output direction arbitrary during assembly
  - 50 MSPS
  - Highly isolated and robust converter coupling.

The adapter board itself provides the following features:

- Fast FPGA-Watchdog functionality providing save switching signals during boot and programming phase of the FPGA
- Sufficient ground connection for all connections
- Supply of 3.3 V, 5.0 V and ±15 V
- Four Module Slots
- TWI-interface option for timing uncritical port expansions (e.g. contactor, temperatures, hardware-user-interface: switches, LED's, As all these components connect to the FPGA, flexible adaption by suitable PCB and adapted VHDL code is straightforward.

User Code Integration and Base Functions

Figure 3 shows how a structure that includes the required interfaces has been developed for an efficient development chain. All of the typical duties for cycle-based control are encased in a precompiled bare-metal framework, which can be coupled with a precompiled user code by employing a standard gcc compiler. This framework can also be used to perform cycle-based control. The user code is performed in an iterative manner, with the user code having the ability to determine the timing, which is then exactly implemented by the FPGA. The

FPGA takes readings of the measurements in a synchronised fashion with the control cycle, and it is able to supply the user code with up to one thousand float values every control cycle. These are compiled into an input array that lives within the memory of the computer. The direct integration of HPS and FPGA enables a memory swap between the FPGA and HPS within one FPGA cycle (10 nanoseconds). Following the memory swap, the HPS is able to access the newly accessible values directly, with the speed of memory access being the sole limiting factor (see above). During the control cycle, the values that are sent to the FPGA and any peripherals that are coupled to the FPGA are gathered in an output array that has room for a thousand float values. At the conclusion of the control cycle, the memory region that stores the output values is likewise swapped, and the FPGA is therefore given access to this information as a result. The user area of the FPGA may have some VHDL code in it, which would provide measurements, precalculations (like mean value computations), margin checks, and pulse pattern creation that is appropriate for the control of the test bench. The individual aspects may be readily incorporated into the whole. During runtime, the service core is able to make configurations to the functionality of the trace. It is feasible to trace a selection of the values that were entered and those that were output. In addition, the service core has the capability of uploading fresh firmware to the bare metal and FPGA. Over a TCP/IP network, the entire system is completely programmable and may be controlled by the user.

RESULTS AND DISCUSSION

A 200 kW 3-phase, two-level converter that is attached to an induction machine is being driven by the control platform. In the user control code area, an open loop control has been created so that inappropriate behaviour is immediately noticeable. A closed-loop control method would eliminate the majority of deviations from the norm. The cycle time is currently set at 200 microseconds, which corresponds to the converter's maximum valve switching frequency.

For this application, the average computation time is roughly 13 microseconds. This time includes the collecting of measurement data and the delivery of commands to the FPGA. The valve control signals are generated by a PWM that is implemented in the user part of the FPGA. These signals are then transferred to the converter using fibre optics. They immediately link to the ADC-module of the control system and translate the converter

currents. These current transducers deliver +/-200 mA for +/- nominal current. The ADC-Module takes readings of the current at a rate that is synced with the PWM reference; as a result, the results of the measurements only display the fundamental frequency. The findings demonstrate the behaviour that was anticipated, and further scope measurements validate the trace's appropriate operational capabilities.

CONCLUSION

Based on the findings that were reported in this work, the idea that was adopted and the SoC device that was picked may be characterised as a solution for the control and emulation of power electronic systems that is highly performant, efficient, and cost-effective. The incorporation of an FPGA portion in addition to two ARM 9 cores results in increased flexibility and adaptability. A dependable foundation for real-time systems may be achieved through the utilisation of asymmetrical multiprocessing to delegate jobs to each of the processor's two cores. Both the control core (bare-metal OS, for the real-time control code, with a cycle time of less than 20 microseconds) and the service core (Linux OS, for communication and auxiliary services) operate separately, with each being optimised specifically for the tasks that they are responsible for. On the other hand, one must exercise caution with regard to the Linux component of the system: The interference that is created by having a second level cache and SD RAM that supports both cores is not insignificant, since it is both inherent and inevitable. If the Linux core makes frequent and rapid access to SD RAM, this might significantly increase the amount of time the real-time core must spend doing calculations. The programme that is executing on the Linux system should be made to enter obligatory pauses (with no or low access to SD RAM), since this is the recommended countermeasure. When integrating third-party functions into the Linux system, however, this process must be approached with caution because it is straightforward for self-made software components.

REFERENCES

1. Dong C, Zeng H. Minimizing Stack Memory for Hard Real-time Applications on Multicore Platforms.
2. Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden. 2014; 1-6.
3. Paun V, Monsuez B, Baufreton P. *On the Determinism of Multi- core Processors.*

French Singaporean Workshop on Formal Methods and Applications (FSFMA), Singapore. 2013; 32-46.

4. Abourida S, Belanger J, Dufour C. *Real-time HIL Simulation of a Complete PMSM Drive at 10 µs Time Step*. European Conference on Power Electronics and Applications, Dresden. 2005; 9.
5. Vincke R, Messiaen A, Boydens J. Hybrid FPGA/Multi-core CPUs for Industrial Applications. *Annual Journal of Electronics*. 2013. ISSN 1314-0078.
6. Vardhan H, Akin B, Jin H. A Low-Cost, High-Fidelity Processor-in-the-Loop Platform. *IEEE Power Electronics Magazine*, 2016; 18- 41.
7. Ashwin JS, Praveen JS, Manoharan N. Optimization of SRAM Array Structure for Energy Efficiency Improvement in Advanced CMOS Technology. *Indian Journal of Science and Technology*, 7(S6): 35- 39.
8. Jyothi B, M.Venugopala Rao. Carrier-Based PWM Technique for Inverter-Fed Multiphase Induction Motor. *International Journal of Electrical and Computer Engineering (IJECE)*. October 2016; 6(5): 1967-1984.
9. Trio Adiono, Aditya F. Ardyanto, Nur Ahmadi, Idham Hafizh, Septian G. P. Putra. An SoC Architecture for Real-Time Noise Cancellation System Using Variable Speech PDF Method. *International Journal of Electrical and Computer Engineering (IJECE)*. December 2015; 5(6): 1336- 1346.
10. Suman S, Sharma KG, Ghosh PK. 250 MHz Multiphase Delay Locked Loop for Low Power Applications. *International Journal of Electrical and Computer Engineering (IJECE)*. 2017; 7(6): 3323-3331.
11. Rajalakshmi C, ArGunasekaran IK. Low-power High-speed Amplification Using Filterbank for Digital Hearing Aids. *International Journal of MC square Scientific Research (IJMSR)*. 2016; 8(1):60-73.