MANTECH Publications

Journal of Power Electronics and Drives
Volume 7, Issue 2, May-August 2022

An Investigation into the Use of System on Chip Real Time Clocks in Power Electronics

Dr. Mudit Mishra[1], Prabhat Jain[2], Vijay Gupta[3]

Associate Professor[1], Students[2, 3]

Department of ECE

Sri Sairam College of Engineering

Email: *prabhatjain25aug@gmail.com*

Abstract

For high-end power-electronic applications, current control and emulation systems (also known as Hardware-in-the-Loop, HIL or Processor-in-the-Loop, PIL) often consist of a large number of components and interconnected buses. A microcontroller for communication and high-level control, a DSP for real-time control, an FPGA section for quick parallel actions and data acquisition, multiport RAM structures, or bus systems as an interface are common components. A substantial number of these capabilities are integrated into a single semiconductor chip through System-on-Chip (SoC) technology. This has the advantage of saving space and money while also increasing the speed of communication inside the firm. These kind of systems are becoming increasingly important not just for scientific research but also for industrial applications. The System-on-Chip (SoC) used here combines a fast processor system (FPGA) with a Dual-Core ARM 9 hard processor system (HPS), with fast interconnects between these separate components. The supporting software and firmware concepts must be carefully studied for SoC systems to enable real-time control and emulation. This article describes how to use the SoC's resources as efficiently as possible and examines the issues caused by the SoC's internal structure. The fundamental principle here is the use of asymmetric multiprocessing: One of the cores runs a bare-metal operating system in hard real time. A "real-time" Linux operating system conducts service activities and communication on the second core. FPGAs are used for flexible process-oriented interfaces (such as A/D, D/A, and switching signals),

quasi-hard-wired protection, and precise timing of the real-time control cycle. This implementation technique is well-known and is even occasionally proposed; but, to the best of the author's knowledge, it is only seldom used and rarely documented in the context of demanding real-time control or emulation. The technique of implementation is broken out in great length in the paper, including the process interfaces, and the research also examines the pros and cons of the chosen notion. The results of the measurements demonstrate the solution's properties.

Keywords: *Multiprocessing, Control, Cache Interference, Soc*

INTRODUCTION

A system-on-a-chip may include a real time clock (RTC) module, a crystal oscillation circuit, and a power supply circuit. This design connects the RTC module, which offers time capabilities, and the crystal oscillation circuit, which creates an oscillation. Because of the coupling that occurs in the voltage supply circuit, the crystal oscillation circuit and at least a piece of the RTC module can each get their own supply voltage. The use of multicore architectures for real-time control applications is a difficult issue that necessitates careful decomposition of the problem in order to make the most of the parallel processing that is available [1-2]. It is rare for this type of optimization to have been reported up to this point in the context of quick prototyping systems and/or low effort development initiatives. The linked use of two kernels in a multiprocessing environment creates an undue demand on the implementation of control code and communication code.

It is impossible to predict how the activities will interact and what advantages will be gained. There is a chance that disrupting service routines will have interesting secondary effects. This paper offers a broad separation of service activities and control tasks, resulting in strict function-based parallelization and a nearly complete decoupling of the real-time control cycle from ancillary services. The real-time core communicates directly with the FPGA, which is a tried and reliable method [3, 4]. This enables exact timing. The requirements for real-time control systems will be examined in the first half of this talk. Following that, a structure that meets these requirements will be built. This section describes

interfaces for connecting to processes, such as the setting of a test bench. Last but not least, benchmark testing and measurement data acquired at a laboratory test setup demonstrate the platform's and its peripherals' functionality. [5] has a detailed discussion of a related approach that does not require real-time capabilities, as well as nearly 70 more citations. As a result, the emphasis of this study is restricted to a few carefully selected sources [6].

Because of the need for fast prototyping and small series control systems, additional limits have been imposed on the adaptability of the control system in terms of both hardware and software. During the development phase, computational performance must be raised; the necessity to optimise for performance concerns must be avoided; user-friendly code must be provided; and more resources for debugging activities may be required. Furthermore, standard features like as measurement, pulse-pattern generating (such as PWM), trace capabilities, and the user interface must be implemented out of the box [7]. This means that no one-time implementation is necessary.

On the process interface, needs may change quickly, and it is critical that it be simple to add new measurements or alternative types of physical measuring interfaces at any time. On the other hand, the system and its related costs must be maintained to a minimum. A modular, adaptable idea with a range of common components may be the best alternative to go with.

There is a possibility of putting power electronic components at risk while testing new control algorithms. As a result, it becomes logical to provide a safety layer that is HPS-independent while simultaneously being rapid and adaptive. An FPGA-based limit check allows for responses in a few microseconds, which is fast enough to protect critical semiconductors. Furthermore, this form of check may be designed independently of the control code. The following needs may be summarised, and the newly provided system will be capable of addressing them all:

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

THE PROPOSED SOLUTION

SoCs are designed to be bootable from a single source. This results in a significant simplification of the boot operation as well as firmware upgrades when compared to current heterogeneous multi-chip control systems [8]. Using a "high-level" operating system, such as Linux, is the most effective way to provide user-friendly and easy task integration. A "real-time" edition of Linux is a wise choice since it enables faster and more predictable reaction times. When applied to this circumstance, the notion of "real time" necessitates a more detailed examination:

- While "real-time" cycles of 100 microseconds or fewer are frequently required for power electronics systems, the bare-metal technique used here and discussed later in this work easily achieves 20 microseconds (if the control code is compact enough)

- “real-time” Although Linux minimises long-term disruptions caused by service activities (such as network traffic), the operating system does not support complete prioritisation of certain tasks, which is required to achieve a short cycle time. From the standpoint of power electronics, this does not represent "real time."

- Commercial real-time operating systems provide real-time capability similar to that described in this article; however, these systems are often unable to avoid all interruptions and must instead focus their attention exclusively on the control code.

It is plainly evident that a "high-level" operating system, such as Linux, with a huge number of pre-built services and functions, is superior in terms of usability. However, the link between these service responsibilities and the most crucial real-time control procedure is complex. It would be ideal if the control process had control over all interrupt services inside the operating system; however, this would make integrating service functions more challenging. Furthermore, upgrading all of the components—including the firmware, software, and FPGA structure—would be complex and time-consuming [9]. The technique employed here integrates the advantages provided by both systems (see Figure 1). One of the cores is responsible for the "real-time" Linux operating system and various service functions. These include software and firmware upgrades over TCP/IP, process control features such as set-point value transmission, and logging capabilities (e.g. streaming measured data to network-attached

storage, NAS). The second core of the computer runs a bare-metal software that handles control functions [10]. It is not directly linked to any external devices (with the exception of a very fast FPGA link to transmit data in both directions for process interaction). As a result, the control latency is nearly entirely independent of the service activities that are presently being performed. However, as shown in the following example, control execution is hampered by interference that cannot be avoided due to shared access to the Level 2 cache and the on-board SD RAM (Figure 2). The quantity of interference, as well as possible reasons and countermeasures, will be investigated in the next portion of this study.

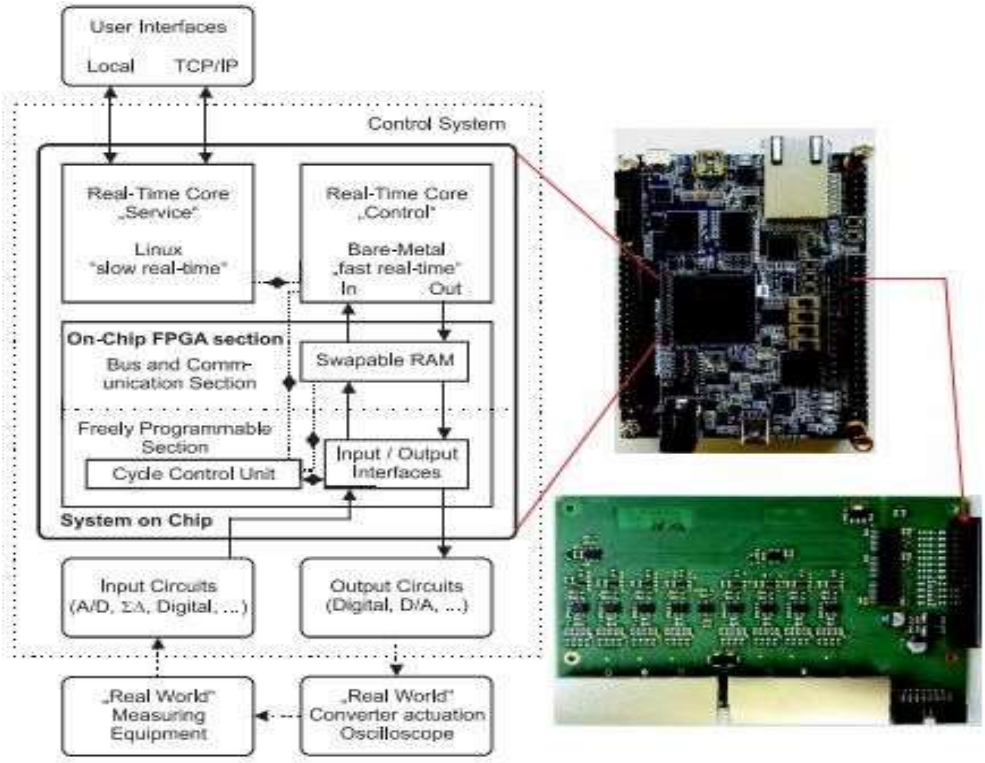
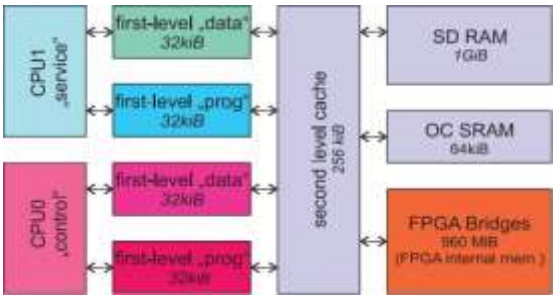


Figure 1. left: Basic structure of the used SoC platform, right: photo of DE0-Nano-SoC



and Peripheral (8 Channel ADC-board)

Figure 2: Cache and interface structure of the SoC

MEMORY INTERFACES AND INTERFERENCES

Both ARM9 CPUs have a common second-level cache, which is linked to memory through the ARM9 architecture (cp. Figure 2). Because on-chip RAM, also known as OC RAM, is far faster than external SD RAM, it is used for data interchange between the two CPUs. The SD RAM provides one gigabyte of RAM. The remaining 704 megabytes are designated as shared memory (directly managed by the bare-metal OS) and are nearly entirely utilised for trace data. The Linux operating system gets 256 megabytes, whereas the bare-metal operating system gets 64 megabytes. The activities performed in respect to the programme and data stored in the first-level cache attached to one core are fully independent of the operations performed by the other core. Unfortunately, the control must contact the FPGA-Bridge for control reasons and the SD RAM for trace functionality throughout each control cycle (typically ranging from 20 microseconds to 500 microseconds, depending on the type of control). Both of these functions are required for control. Interaction with the second level cache is necessary to do this, which results in interference with the service core. According to our observations, the service core's CPU demand seldom exceeds 1%, resulting in very little access to the second level cache.

Executing code that requires extensive access to the SD RAM, on the other hand, may cause severe interference between control and service tasks. Figure 3 is an example of such interference. The SRAM array structure was optimised in advanced CMOS technology with the purpose of boosting energy efficiency [6].

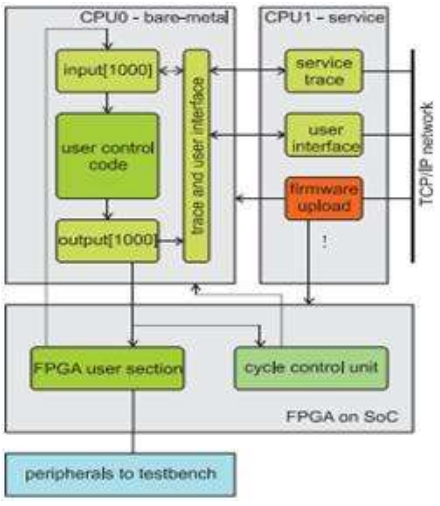


Figure 3 Example for such interference

During normal operation, the time required to complete the execution of the selected control code (cp, t < 0 s) varies between 28 µs and 34 µs. This time period comprises the collection of measured data as well as the operation of the trace. When this occurs, the control code will store the measured data in a circular buffer before storing it in a data file (which is then provided by the service core). It is critical to note that the system's setup allows for changeable cycle times (e.g. for random PWM or PLL-controlled cycle-time adjustment in normal PWM mode). As a result, each measurement that has been recorded has its own (possibly unique) cycle time. If a trigger event happens, it is expected that the data file will contain both a pre-defined pre-trigger sequence and a post-trigger sequence.

When the trigger event happens at t=0 s, a Linux service function will prepare the data for storage in a data file. It must first determine when the pre-trigger sequence begins in order to achieve this. This cannot be deduced from the cycle time since the cycle time is subject to change. The service function must do a cumulative accumulation of all cycles going backwards in time until it reaches the pre-trigger duration.

A large amount of data is read from the storage device's dynamic random access memory (SD RAM) in record time (the service core has nothing else to do and provides high calculation power). As a result, there are frequent cache misses, and the second-level cache must be loaded from the service core's side. As a result, the time required for the bare-metal core to access memory increases, and the time spent running control code reaches a temporary peak of up to 68 µs. This is more than double the average value, and it is quite likely that it will exceed the allowed control cycle time, causing the system to trip.

Following the launch of the tracing service, the data access rate will finally settle into a more tolerable range. It is advantageous to have write access to slower storage devices or network connections in this case. The cache interference is still present, but it is much easier to accept today. As a result, service tasks in processes that need extensive data access must deliberately slow down.

This is not a significant performance detriment: The service core's real-time demands or priorities are not extremely rigorous. Furthermore, pauses are simple to implement into self-created service

routines. On the other hand, it is evident that software operating on the Linux core may interfere with real-time control; hence, software components should be thoroughly tested before use: Even a minor load surge lasting a few microseconds may violate the real-time control code's real-time criterion.

THE PERIPHERAL CONCEPT

Two separate signal standards are frequently employed in the field of power electronics, where several kilowatts are involved. Current signals in the ±20 mA to ±200 mA range are most commonly used as analogue number standards. Signals based on current have a strong noise immunity, allowing connections to be made over long distances even in places with high EMI levels. There are several sensors available, including those for isolating current and voltage probes, as well as temperature sensors.

Fiber optics provide a rapid and easy method for digital signal transfer, such as those used in communication, digital signal transmission (such as sigma-delta-modulated signals), and PWM signals. They are virtually fully resistant to EMI, and their transmission delay is nearly independent of cable length, allowing for precise timing. Furthermore, they provide nearly full isolation, avoiding the need for further galvanic isolations.

A modular platform is being designed to handle numerous jobs. Each of the two GPIO slots may be outfitted with an adapter board for four of the peripheral modules listed below:

8 channel current input analogue to digital converter

- Several signal standards up to ± 200 mA (directly compatible with common current sensors)
- 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 depicts the creation of a framework that comprises the necessary interfaces for an efficient development chain. All of the conventional cycle-based control chores are encased in a precompiled bare-metal framework, which may be linked with precompiled user programmes using a regular gcc compiler. Cycle-based control may also be performed using this framework. The user code is executed iteratively, with the user code determining the time, which is then precisely implemented by the FPGA. The FPGA reads the measurements in synchrony with the control cycle, and it can feed the user code with up to a thousand float values every control cycle. These are compiled into an input array that is stored in the computer's memory. The direct integration of HPS and FPGA allows for memory swapping between the FPGA and HPS during a single FPGA cycle (10 nanoseconds). Following the memory swap, the HPS may directly access the newly available values, with memory access speed being the only limiting factor (see above). During the control cycle, the values delivered to the FPGA and any peripherals attached to the FPGA are grouped in an output array with space for a thousand float values. At the end of the control cycle, the memory area that holds the output values is also swapped, and the FPGA now has access to this information. The user region of the FPGA may contain VHDL code that provides measurements, precalculations (such as mean value computations), margin checks, and pulse pattern generation suited for test bench control. Individual components can be easily integrated into the overall. The service core has the ability to configure the trace's functionality during runtime. It is possible to follow a subset of the values that were

input and printed. Furthermore, the service core is capable of uploading new firmware to the bare metal and FPGA. The entire system is totally programmable and may be controlled by the user through a TCP/IP network.

RESULTS AND DISCUSSION

The control platform drives a 200 kW three-phase, two-level converter connected to an induction machine. An open loop control has been introduced in the user control code section to make incorrect behaviour obvious. The bulk of departures from the standard would be eliminated by a closed-loop control approach. The cycle time is now set to 200 microseconds, which corresponds to the maximum valve switching frequency of the converter.

The average calculation time for this application is around 13 microseconds. This time period involves the collection of measurement data as well as the transmission of commands to the FPGA. A PWM implemented in the user section of the FPGA generates the valve control signals. These signals are then sent through fibre optics to the converter. They instantly connect to the control system's ADC-module and translate the converter currents. These current transducers provide +/-200 mA for a nominal current of +/- The ADC-Module collects current readings at a rate that is synchronised with the PWM reference; as a result, the measurements only show the fundamental frequency. The findings support the predicted behaviour, and additional scope measurements corroborate the trace's operating capabilities.

CONCLUSION

Based on the findings provided in this study, the idea that was adopted and the SoC device that was chosen may be described as a highly performant, efficient, and cost-effective solution for the control and emulation of power electronic systems. The addition of an FPGA component in addition to two ARM 9 cores increases flexibility and adaptability. A stable basis for real-time systems may be built by delegating work to each of the processor's two cores via asymmetrical multiprocessing. Both the control core (bare-metal OS, for real-time control code with a cycle time of less than 20 microseconds) and the service core (Linux OS, for communication and auxiliary services) work independently, with each optimised especially for the tasks for which it is responsible. On the other hand, the Linux component of the system should be approached with caution: The interference caused by a second level

cache and SD RAM that supports both cores is considerable, as it is both inherent and unavoidable. If the Linux core makes frequent and quick access to SD RAM, the amount of time the real-time core needs spend completing computations may grow dramatically. Because this is the suggested countermeasure, the software running on the Linux system should be forced to enter mandatory pauses (with no or limited access to SD RAM). However, though it is simple for self-made software components, integrating third-party functions into the Linux system must be treated with prudence.

REFERENCES

1. 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.

2. 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.

3. Dong C, Zeng H. Minimizing Stack Memory for Hard Real-time Applications on Multicore Platforms. Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden. 2014; 1-6.

4. 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.

5. 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.

6. 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.

7. Vincke R, Messiaen A, Boydens J. Hybrid FPGA/Multi-core CPUs for

Industrial Applications. Annual Journal of Electronics. 2013. ISSN 1314-0078.

8. 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.

9. Vincke R, Messiaen A, Boydens J. Hybrid FPGA/Multi-core CPUs for Industrial Applications. Annual Journal of Electronics. 2013. ISSN 1314-0078.