An FPGA-based Real-Time Event Sampler

Niels Penneman[1,2], Luc Perneel[3],
Martin Timmerman[1,3], and Bjorn De Sutter[1,2]

[1] Electronics and Informatics Department, Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel, Belgium

[2] Computer Systems Lab, Ghent University
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
{niels.penneman|bjorn.desutter}@elis.ugent.be

[3] Dedicated Systems Experts
Bergensesteenweg 421 B12, 1600 St-Pieters-Leeuw, Belgium
{l.perneel|m.timmerman}@dedicated-systems.info

Abstract. This paper presents the design and FPGA-implementation of a sampler that is suited for sampling real-time events in embedded systems. Such sampling is useful, for example, to test whether real-time events are handled in time on such systems. By designing and implementing the sampler as a logic analyzer on an FPGA, several design parameters can be explored and easily modified to match the behavior of different kinds of embedded systems. Moreover, the trade-off between price and performance becomes easy, as it mainly exists of choosing the appropriate type and speed grade of an FPGA family.

Keywords: real-time testing, event sampling, logic analyzer, FPGA

1 Introduction

Real-time (RT) computing systems have constrained reaction times, i.e., deadlines have to be met in response to events. Ensuring that the constraints are met on a device for a given operating system and set of applications becomes difficult as soon as either of them shows non-trivial behavior. For hard RT systems guarantees have to be provided, which is often done by means of worst-case execution time analysis and by relying on predictable algorithms and hardware [1, 2].

Given the criticality of the RT behavior for safety, quality of service, and other user-level requirements, extensive testing of the RT behavior is often performed on a full system. Hence precise methods are needed for observing that RT behavior. Some requirements of these methods are that (1) they should be fast and accurate to allow the correct observation of events happening at high rates, (2) they should be non-intrusive to make sure that the system under test (SUT) behaves as similar to the final system as possible, (3) the methods should be configurable for different measuring contexts, given the wide range of RT deadlines and of application behaviors, and (4) given that relatively few developers will test RT behavior, the required infrastructure needs to be cheap.

On many embedded systems, the rate at which events occur is so high that software-only solutions cannot meet the first two requirements. This paper presents an FPGA-based event sampler which can be used in a hardware-software cooperative approach. The SUT emits signals through hardware when events occur, which are then captured by the sampler. This system will be (1) fast enough because it runs on an FPGA, (2) reconfigurable by altering design parameters or by choosing different FPGAs, (3) cheap because it does not require high-end FPGAs, and (4) non-intrusive because the required changes to the SUT are minimal. Our proposal is in line with today's use of FPGAs to speed up EDA tasks such as software-hardware co-design and system simulation.

The remainder of this paper is structured as follows. Section 2 provides more background information on the problem of RT event sampling. The design of an FPGA-based sampler is presented in Section 3, after which Section 4 evaluates the performance obtained with this design, and Section 5 draws conclusions.

2 Real-Time Sampling System

The goal of our system is to sample signals emitted by the SUT, timestamp them, and send them to the tester's workstation for further interpretation.

Hardware signals corresponding to events on the SUT have to be emitted with the least possible intrusion on its behavior. Furthermore, the emitted signals have to reach the logic analyzer with predictable, fixed latency; otherwise, precise tracking of RT behavior is not possible. This rules out several existing communication interfaces such as PCI, PCI Express and IEEE 1149.1 JTAG.

By contrast, General Purpose Input Output (GPIO) interfaces can be controlled with fixed latency, using memory-mapped IO through GPIO registers. The adaptation of a RT OS to emit signals via a GPIO interface upon events is then limited to manual instrumentation of the code, adding at most a few instructions per event. Similar uses of GPIO can be found in, e.g., [3, 4].

Six commercially available logic analyzers have been evaluated [5–10]. None of these devices can sample for a prolonged time with high accuracy. Firstly, most of them sample continuously, instead of only storing samples as events occur. Hence, large amounts of data are generated. Secondly, the available memories tend to be too small to capture a reasonable amount of events.

It is clear that both a large memory and event-based sampling are key to solving our problem. For that reason, our design captures data from eight input channels: four channels for actual test data, two channels for interrupt generation testing, one channel for the SUT to signal an error, and one channel for timestamp counter overflow detection (optionally configurable as external input [11]).

Figure 1 shows an overview of our design. Samples are 32 bits in size, of which 8 map to the input signals, and 24 are dedicated to the timestamp. The timestamp is provided by a counter, operating at the sampling frequency. In order to reconstruct the exact timing, counter overflows are also stored as events. A control block on the data path enables the sampler to start and stop registering events. Users can interact with this block through the management interface. The

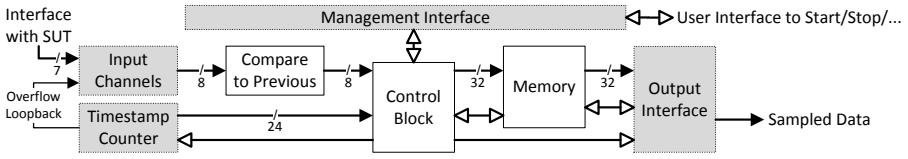

Fig. 1. Conceptual design of the logic analyzer device. Black arrows indicate data edges, while white arrows indicate control signals.

Device	Logic Elements	9kb Memory Blocks	Total RAM Bits	PLLs	Global Clock Networks
EP3C25	24,624	66	608,256	4	20
EP3C40	39,600	126	1,161,216	4	20

Table 1. Overview of some relevant features of targeted Altera Cyclone III devices.

control block may also stop the sampling process whenever errors are detected, and provides reset functionality as a recovery mechanism. The samples stored in internal memory can be retrieved through the output interface.

3 Logic Analyzer Design

This section discusses the different components of our design as a so-called System on a Programmable Chip (SOPC) on Altera's Cyclone III FPGA devices. Although our design will also work on more advanced FPGAs, our design choices will be based on the EP3C25F324C8 FPGA. Some properties of the Cyclone III device family are presented in Table 1.

3.1 Communication with the Workstation

As indicated on the right in Figure 1, the sampler needs to transmit sampled data to the workstation of the tester. Moreover, the tester must be able to control the sampler. Ethernet is fast enough for our purpose and future-proof with respect to commonly used workstation configurations. Although hardware TCP/IP stack implementations exist [12, 13], they are either limited in functionality or overly complex in terms of hardware and resource usage. Alternatively, Altera provides a SOPC development environment [14] with ready-to-use components [15–17], including the Nios II soft-core CPU and an Ethernet MAC suitable to run a software TCP/IP stack. Opting for this solution requires us to design a custom SOPC. The CPU can then be used to run both the TCP/IP stack and the software control over the whole sampler.

In the structure of the whole RT sampler design, the five components on the top right of Figure 2 implement the Ethernet interface. The Ethernet MAC core has two streaming interfaces, one for transmitting (TX) and one for reading (RX) data. Each of these interfaces needs to be connected to the data memory using scatter-gather DMA controller cores. The operation of these cores is defined

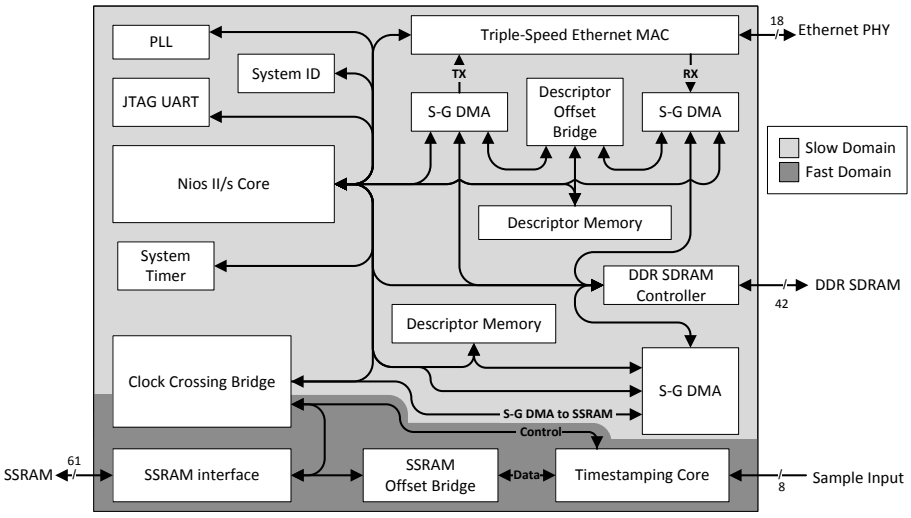

Fig. 2. Structure of the whole logic analyzer design.

through DMA descriptors. In order to improve performance, a separate on-chip memory region is allocated to hold these descriptors. The Nios II CPU connects to all of these components using memory-mapped interfaces. The Ethernet MAC and DMA cores implement slave ports and interrupt senders through which they can be controlled and monitored. The CPU also needs access to the DMA descriptor memory to allocate and initialize descriptors.

The software TCP/IP implementation running on the Nios II core controls all this hardware. Output data to be transmitted to the tester's workstation is obtained from external DDR SDRAM via the DDR SDRAM controller. The rationale for this type of memory is explained in Section 3.3.

3.2 Control over the Logic Analyzer

The Nios II CPU, on which the control software will run, comes in three different configurations [16]: economy, standard and fast. We chose the standard configuration: the fast configuration is too large, offering features our software cannot exploit, while the economy version does not offer enough performance.

The software [11] is built on the MicroC/OS-II RT OS [18]. Alternatives are available, such as uClinux [19] and eCos [20]. Their Nios II ports [21] were not considered because they were either outdated or lacked integration support with recent Altera software versions. The TCP/IP stack is provided by InterNiche.

The initial memory footprint of the software varies between 1256 and 1350 kB, depending on how much debug code is included. As detailed in Section 3.3, samples need to be moved from the SSRAM into the SDRAM before they can be

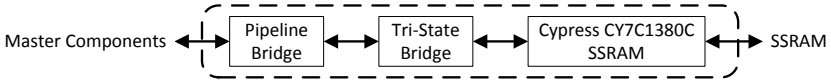

Fig. 3. SOPC components constituting the SSRAM interface.

sent over the network. Therefore, a buffer is allocated statically in the SDRAM with the size of the SSRAM, which accounts for 1024 kB of the footprint.

3.3 Memory Architecture

Our logic analyzer requires memories to store the timestamped signals before they are being transferred to the tester's workstation. We opted for an approach with three types of memory: on-chip RAM, SSRAM, and DDR SDRAM.

SSRAM provides burstable and hence predictable storage. The timestamping core acts as a master to this memory to store samples through the interface shown in Figure 3. The DMA controller at the bottom of Figure 2 also acts as a master to transfer samples from the SSRAM to the SDRAM through the clock crossing bridge. Clock crossing bridges add their address as an offset to the address of their slave components. Since Altera requires SOPC designs to use a flat memory model, offset bridges must be introduced to compensate for this behavior.

In order to prevent other components from interfering with the single-port SSRAM, the slower SDRAM is used as general-purpose data memory. This means that the Ethernet interface also gets its data from the SDRAM. Samples are transferred to this SDRAM from the SSRAM in a fast and predictable manner by the DMA controller, controlled by the software. By filling the buffer internal to the timestamping core instead of writing samples directly to the SSRAM, the SSRAM port can be freed to allow transfers to the SDRAM. Because the software controls both processes (unlike the Ethernet interface, which is also dependent on external factors), it is capable of scheduling the reads and writes to the SSRAM without blocking the timestamping core unnecessarily.

3.4 Timestamping Core

The gateware for the actual timestamping needs to fit in Altera's SOPC model to allow interaction with other components such as the CPU and memory devices. For that reason the timestamping core is designed as a custom component following the Altera SOPC standards. This core samples external signals, adds timing information and manages an integrated on-chip buffer.

The pipelined data flow within the timestamping core is depicted in Figure 4. The timestamp counter is continuously incremented at the sampling frequency. First, raw input from the FPGA pins is combined with the value of the timestamp counter into a sample. Next, the input bits are compared with the relevant bits of the previous sample. When sampling is enabled and the inputs are different, the new sample is enqueued in the on-chip FIFO buffer.

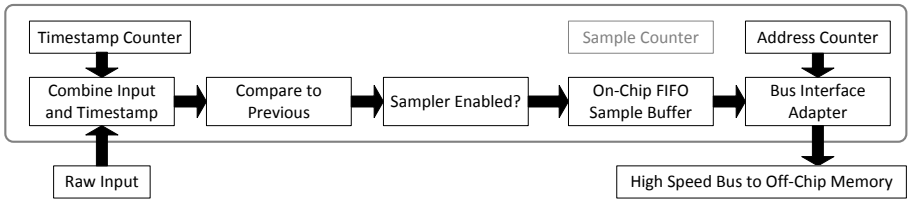

Fig. 4. Pipelined data flow within the timestamping core.

The on-chip FIFO buffer provides independent read and write ports. Adapter logic enables the read side to operate as a memory-mapped bus master. The address to write to in the SSRAM is provided by the address counter, which is incremented on each write operation. A separate sample counter keeps track of the number of samples written, and is incremented at the same time.

Due to the design of the memory-mapped master interface, the address counter starts just before the base address of the external memory. While the number of stored samples could be derived from the rightmost 19 bits of the destination address, doing so would require a 19-bit addition. Although reporting on this number is generally not a critical operation, detecting that the off-chip memory is full is however critical. In the separate sample counter, bit 19 indicates this condition. Therefore, using separate counters outperforms the solution with a 19-bit addition.

3.5 Clock Domains and Other Logic

The whole design was split in two clock domains, because the control over the whole system is less time-critical than the components involved in sampling, allowing optimal placement on the FPGA of the latter. Both domains interface with each other through a clock crossing bridge as depicted in Figure 2.

3.6 Tuning the Design

The sampling accuracy of our design can obviously be improved by using faster FPGAs or by using bigger ones on which the additional resources increase the freedom of the fitter, which will hence be able to reach higher clock speeds.

The accuracy can also be increased by introducing a third clock domain for the timestamping core, which runs at a higher clock rate than the SSRAM interface. In this configuration, the size of the FIFO buffer internal to the timestamping core will determine the maximum burst size. Larger buffers allow longer bursts, but also require larger FPGAs. Since our design uses 63 out of 66 memory blocks on our target device, there is barely any room left to experiment with these features. Using more memory blocks severely limits the freedom of the fitter, and hence results in a significant drop of the maximum sampling frequency.

Logic Elements	RAM bits	9kb Memory Blocks	PLLs	Pins
16,670 (67.7%)	238,382 (39.2%)	63 (95.5%)	2 (50.0%)	130 (60.2%)

Table 2. Resource utilization of the complete design on an EP3C25F324 FPGA.

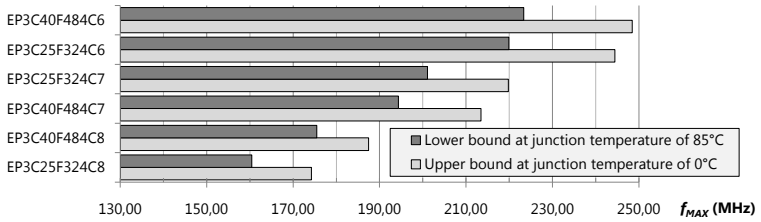

Fig. 5. Maximum sampling frequency obtained on several Cyclone III FPGAs.

4 Performance Evaluation

All synthesis was performed with Altera Quartus II v9.0 [14], targeting the Altera Cyclone III FPGA starter kit. Table 2 shows the resource utilization of the complete design on the target FPGA. Figure 5 shows the maximum sampling frequency on different Cyclone III devices. For each device, the SDRAM interface was set to operate at the maximum frequency according to its speed grade.

To interpret these results, one should know that the EP3C40F484 devices are next in line to EP3C25F324 devices when it comes to available LEs, M9K blocks, and configurable IO pins; detailed specifications are shown in Table 1. The larger devices result in a speed gain for the fastest (C6) and slowest (C8) speed grades. However, the difference for the C6 grade is not significant, as results may slightly vary with pin assignments or different synthesis parameters.

We can draw the following conclusions: Except for the C8 speed grade, the resource utilization of the device on the FPGA does not harm its performance. For sampling frequencies up to 150 MHz, the smallest, cheapest and slowest FPGA (EP3C25F324C8) is sufficient. For sampling frequencies up to 200 MHz, the C6 speed grade of the same size (EP3C25F324C6) is a safe bet.

5 Conclusions

This paper presented the design of an FPGA-based RT event sampler that can be used to test the RT behavior of embedded systems. It supports fast, non-intrusive sampling, and is cheap because low-cost FPGAs suffice. Its performance scales very well with different FGPA classes and speed grades. Furthermore, by changing some design parameters, such as buffer sizes, a wide range of RT behaviors can be targeted easily, e.g., with or without long bursts of events.

References

1. Sha, L., Abdelzaher, T., Årzén, K.E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: A historical perspective. Real-Time Syst. **28**(2-3) (2004) 101–155
2. Buttazzo, G.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications (Real-Time Systems Series). 2nd edn. Springer (2005)
3. Choudhuri, S., Givargis, T.: FlashBox: a system for logging non-deterministic events in deployed embedded systems. In: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing. (2009) 1676–1682
4. Chen, X., Zhang, D., Yang, H.: Design and implementation of a single-chip ARM-based USB interface JTAG emulator. IEEE International Symposium on Embedded Computing (2008) 272–275
5. Active Technologies: AT-LA500 USB logic analyzer (2008) http://www.activetechnologies.it/02products/Atla/00Overview/text.htm.
6. CWAV Corporation: USBee DX Test Pod Users Manual. 3.1 edn. (2008) http://www.usbee.com/dxmanual.pdf.
7. Intronix Test Instruments Corporation: Intronix LA1034 LogicPort PC-based logic analyzer with USB interface (2008) http://www.pctestinstruments.com.
8. Janatek Electronic Designs: Annie-USB PC-Based Logic Analyzer: User's Manual. 2nd edn. (2008) http://www.janatek.co.za/annie-usb_main.htm.
9. Janatek Electronic Designs: LA-Gold-36 PC-based logic analyzer (2008) http://www.janatek.co.za/la-gold-36_main.htm.
10. Link Instruments Corporation: IO-3200 USB logic analyzer and pattern generator for windows (2008) http://www.linkinstruments.com/logana32.htm.
11. Penneman, N.: A renewed sampler system for evaluating and benchmarking (RT)OS. Master's thesis, Vrije Universiteit Brussel (2009)
12. Sutton, P., Brennan, J., Partis, A., Peddersen, J.: VHDL IP stack (2001) http://www.itee.uq.edu.au/ peters/xsvboard/stack/stack.htm.
13. ADESCOM: Wire-Speed Internet: IP Core for VoIP and IPTV Internet (2009) http://www.adescom.com/ipac1.htm.
14. Altera Corporation: Quartus II Handbook. 9.0.0 edn. (2009) http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf.
15. Altera Corporation: Triple Speed Ethernet MegaCore Function User Guide. 9.0 edn. (2009) http://www.altera.com/literature/ug/ug_ethernet.pdf.
16. Altera Corporation: Nios II Processor Reference Handbook. 9.0.0 edn. (2009) http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.
17. Altera Corporation: Using high-performance DDR, DDR2, and DDR3 SDRAM with SOPC Builder. Application Note 517, Altera Corporation (2008) http://www.altera.com/literature/an/an517.pdf.
18. Labrosse, J.J.: MicroC/OS-II: The Real Time Kernel. 2nd edn. CMP Media, Inc., USA (2002)
19. Arcturus Networks Incorporated: uClinux™– embedded Linux microcontroller project. http://www.uclinux.org.
20. eCos: embedded configurable operating system. http://ecos.sourceware.org.
21. Nios Community Forum: http://www.niosforum.com