A Catalog of Hardware Acceleration Techniques for Real-Time Reconfigurable System on Chip

Neil Bergmann, Peter Waldeck, John Williams
School of ITEE, University of Queensland
{n.bergmann; waldeck; jwilliams}@itee.uq.edu.au

Abstract

The new technology of reconfigurable System-on-Chip is shown to be a good match to the requirements of real-time embedded systems. In particular, the judicious use of specialised data processing peripherals can reduce the CPU load significantly and greatly ease the task of guaranteeing that real-time deadlines are met in complex multi-processing real-time systems. A catalog of other possible uses for the reconfigurable logic resources on such a chip which can assist in improving real-time system performance is also presented.

1. Introduction

As FPGAs reach mega-gate size, it now becomes feasible to implement a complete microcontroller, consisting of CPU, peripherals, and a limited amount of program and data memory on a single FPGA. We call such a system a reconfigurable System-on-Chip (rSoC).

The concept on an rSoC can be extended to include systems where a hardwired CPU is incorporated on the die along with the FPGA circuitry, such as those offered by Xilinx [1], Altera [2], Atmel [3] and Triscend [4]. Additionally, we extend this concept of rSoC to include those systems where external memory chips (RAM, CPU program ROM, FPGA configuration ROM, Flash) are added to the integrated CPU-plus-peripherals chip.

RSoC technology is seen as being particularly useful for a number of different scenarios:

- Embedded systems where a single chip microcontroller is unavailable with the required set of peripherals, eg. a system requiring a microcontroller with 32 PWM outputs.
- For prototyping of ASIC SoC solutions in embedded applications.
- For cases where the range of peripherals changes in different operating modes.
- For implementation of real-time embedded systems, where custom hardware peripherals can improve real-time response rates.

We are particularly interested in the last of these.

2. Real-time System-on-Chip

A useful distinction is to define three types of computer systems [5] – transformational, interactive and reactive (or real-time). We have argued in an earlier paper [6] that reconfigurable computing can provide significant advantages for real-time computing, and we summarise the core of the argument here.

For conventional "transformational" and "interactive" computer systems, the goal is to complete each individual task or sub-task in as short a time as possible. Since each sub-task is usually a relatively long and complex set of operations, overall performance is greatly improved by techniques which increase the *average rate* at which instructions are executed. Such tasks include pipelining, memory caching, and multiple instruction issue.

The goal of real-time systems, especially hard real-time systems is quite different: to guarantee that a response can be made to an input signal by a fixed deadline, even in the worst case situation. Average instruction execution rates tend to be less important than worst case calculation times. For complex real-time systems with many tasks, many signals, and many deadlines, a software solution to a real-time multi-tasking environment leads to a very conservative computing solution. A powerful processor is needed to guarantee that response times always are met, even for very rare conjunctions of events.

Being able to respond to inputs with specific hardware modules has obvious advantages. Many hardware units can all operate in parallel, so that individual response times are much less variable and easier to guarantee, even as the number of tasks increases. Parallel hardware is not so affected by issues such as task swapping, scheduling, interrupt service, and critical sections, which complicate real-time software solutions.

While we can easily argue the potential advantages of reconfigurable real-time systems, our eventual research goal is to clearly demonstrate and prove these advantages through theoretical analysis and practical case studies.

One key step in our research program is identifying different ways in which the characteristics of reconfigurable system-on-chip technology can be used

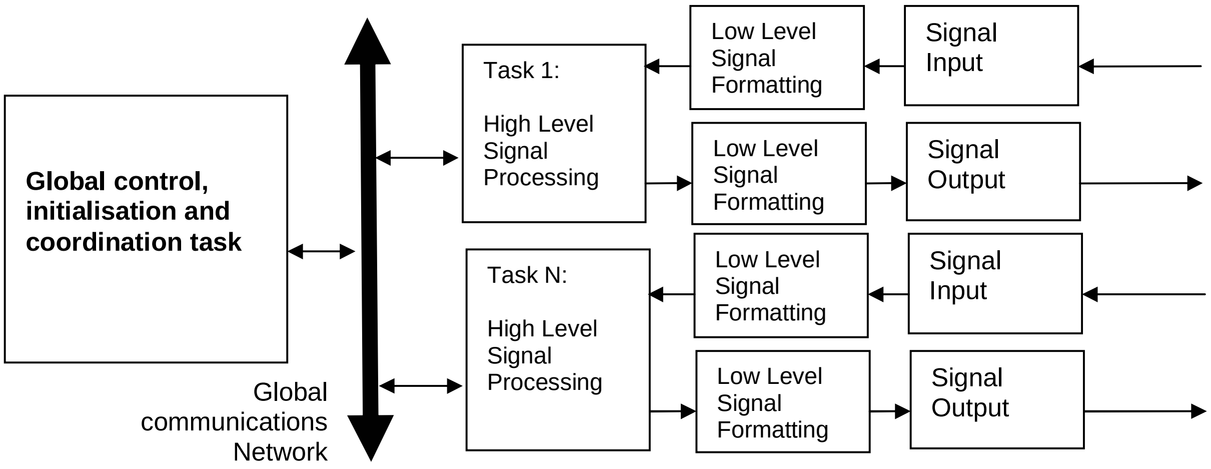
Global control, initialisation and coordination task
Global communications Network
Task 1: High Level Signal Processing
Task N: High Level Signal Processing
Low Level Signal Formatting
Low Level Signal Formatting
Low Level Signal Formatting
Low Level Signal Formatting
Signal Input
Signal Output
Signal Input
Signal Output

Figure 1: A Hierarchy of communicating real-time tasks

advantageously for real-time system implementation. That is the purpose of this paper.

We will catalog a large number of different possible mechanisms for enhancing the implementation of real-time systems through the use of rSoC technology.

3. A model of RT system implementation

Figure 1 shows a simplified framework for describing multiple real-time tasks. We will use this framework to reason about how different tasks can be assisted by rSoC technology.

For a given set of tightly coupled inputs and outputs (whose processing constitutes a single ‘task’), data needs to be input from external sensors or communication channels, it needs to be assembled into meaningful words, packets or streams of signals, these signals are then processed into meaningful output sets, which are disassembled and sent to actuators or other systems.

In addition, there is a need to transmit data between cooperating tasks, and also between individual I/O processing tasks and a global controlling task which sets overall system parameters, and collects overall system status information.

Conventionally, a real-time system consists of a CPU, memory, plus a number of different I/O peripherals such ADCs, DACs, UARTs, GPIO, and timers.

In such systems, using the framework above, the hardware peripherals typically handle signal I/O and some initial signal formatting (eg. converting RS232 bits into words). Once a meaningful signal is received or sent, an interrupt is sent to initiate software to deal with additional signal formatting and to signal the appropriate software task that there is data for processing. Other software tasks deal with global signal processing. A Real-time Operating System (RTOS) coordinates and schedules the tasks and assists with interprocess communication.

rSoC technology can improve real-time system performance by providing additional hardware support for these operations.

4. RSoC support for real-time systems

The rest of the paper describes some of our early ideas about how rSoC technology can be used to enhance the implementation of real-time embedded systems. All of these techniques have the potential to improve system performance. Much of our research in the near future will be trying to determine if these potential advantages can be realised.

4.1. Customised peripherals

Ordinary peripherals (timers, UARTs etc) necessarily tend to be general purpose. RSoC technology allows individual peripherals to be customised to the data and the tasks that they deal with. This can reduce the overall gate count of peripherals by removing any unused features, or alternatively improve overall system performance by adding non-standard features.

For example, if a UART is used for a communications channel which uses neither parity checking or hardware handshaking, and which operates at a fixed baud rate, then the UART can be much simplified.

Alternatively, if digital samples are received through an ADC from a transducer at a regular sample rate, and these need to be tested to ensure that they are within an allowable operating range, then a customised ADC interface can do this range-checking with a small number of extra gates, and only signal the CPU when data is out of range.

Customising peripherals extends not just to individual peripherals, but also to the mix of peripherals provided. Microcontroller manufacturers provide a large range of different microcontroller models for the same CPU, each with a different mix of peripherals (eg. Motorola offer more than 20 variations on their 68HC11 processor [7]). Usually, on any given microcontroller in a system, many of these peripherals are unused. Alternatively, unusual mixtures of peripherals (say 6 UARTS and 12 PWM outputs) mean that no microcontroller has the right set.

4.2. Hardware data formatting

For specific applications, there may be higher levels of information organisation than just single input and output samples which conventional peripherals deal with. For example, communications over a particular RS232 channel might always be used within a higher communications protocol, with specific packet structures. Customised rSoC peripherals can do the front-end data processing to group bytes into packets, and only signal the CPU when a whole packet is available for processing, reducing the CPU load. In such cases the "Low Level Signal Formatting" operations of Figure 1 are done in programmable hardware.

4.3. Hardware signal processing

Often, real-time processing of repetitive data streams consists of several layers of processing. There is some simple processing that needs to be done on every sample, and then higher level processing which needs to be done less frequently on larger sets of data. For example, in a car ignition system, a spark needs to be generated at a specific interval after a crankcase sensor signals that a piston is at TDC (Top Dead Centre). This spark signal needs to be generated on every combustion cycle. The particular timing interval for the spark may change as a function of engine speed, engine temperature, but this does not need to be updated every cycle.

Hardware is well suited to simple, fast, regular computation, whilst software is more suited to slower, more complex, variable computation.

In rSoC technology we can build custom signal processing hardware to deal with the very simple, regular high-speed signal processing, and leave the more complex processing for software. This migrates some of the "High Level Signal Processing" modules in Figure 1 from hardware to software.

4.4. Customised CPUs

If the rSoC has either a soft processor core, or a hard processor core which is extensible with reconfigurable logic, then another option is to explore new architectural features in the CPU to better support real-time system operation. We list some possibilities below.

4.4.1. Customised interrupt handlers. A major cause of unpredictable response times for real-time software tasks involves servicing interrupts. Typically, interrupts are used to signal the arrival of new input data, to signal the need for new output data, or to signal the end of a specified scheduling time slice. Servicing an interrupt requires a context switch upon entry and typically a scheduling operation upon exit. Multiple levels of interrupts, and disabling of interrupts during critical sections adds to processing delays in responding to interrupts. Novel interrupt handlers might include, for example, delayed interrupt handling to allow for more efficient handling of incoming and outgoing data when the appropriate task is next scheduled, or parallel interrupt processing hardware which can deal with moving data samples to memory buffers and marking pending I/O tasks as ready without interrupting the main CPU tasks through a context switch.

4.4.2. Customised schedulers. Implementing efficient scheduling algorithms can have a major effect on real-time system performance at meeting deadlines. Static scheduling algorithms (eg. implemented with fixed priorities) provide quick but inflexible scheduling. Dynamic scheduling, such as deadline monotonic [8], provides good scheduling orders, but incurs additional software overhead.

Implementing complex scheduling algorithms in hardware has only been practical in the past for high volume ASICs with fixed function. RSoC technology provides a mechanism where hardware scheduling algorithms can be altered to meet the particular process mix at a specific time. Additional hardware timers, for example, can provide accurate measures of parameters such as time to next deadline. Hardware algorithms can also pre-empt a task at any time that another task becomes more urgent, not just at the end of fixed time slices.

4.4.3. Specialised multi-processing support. Guarantees of real-time performance become harder as more and more tasks are added to a system, because of the larger number of context switches, which are effectively wasted overhead time on the CPU.

Another fruitful area of exploration would be to investigate fast context-switching algorithms. RISC processors typically use register windowing to reduce the context switch time during subroutine calls. A similar approach can be taken with real-time systems, but using an array of internal register banks, one per process. A single register, perhaps under the control of a hardware scheduler, would control the currently active context. A change to this register would make a new context (including PC, stack pointer, etc) available to the CPU, effectively giving a zero-cycle context switch. In a conventional processor, it would be problematic to decide how many context banks to have – in an rSoC the number can be decided precisely for each different application.

4.4.4. Specialised instruction sets. A soft processor core can be customised with a unique instruction set, depending on the mix of processes which it is running. DSP algorithms, for example, might require instructions for 12 bit arithmetic. A limiting-addition instruction might limit the answer to MAXINT if there is an overflow. Special instructions might be used to add or scale A-law companded PCM voice samples. If any instructions in the "standard" instruction set are unused, then some hardware savings can be made.

4.4.5. Customised memory address decoding. Address decoding hardware is needed in a microcontroller to select appropriate addresses for peripherals and memory blocks. In an rSoC, one can gain some additional flexibility by adding specialised address decoding to speed up some operations. For example, if a CPU has hardware support for multiple contexts, then the same context index could be used to provide context-dependent memory addressing.

Important addresses for a process, such as data and status registers for peripherals, could be mapped into CPU-register addresses, which can be accessed in a single cycle, rather than via a slower memory or peripheral addresses. Context-mapped addresses could also allow many tasks to each run within a small memory address space (64 k) and take advantage of smaller address calculation even though the overall system memory is significantly larger. Such context-mapped addresses would also allow for separate module compilation, even in the absence of a conventional MMU.

4.4.6. Multiple CPUs. We have so far assumed a single multi-tasked CPU assisted by parallel hardware modules processing peripheral data. Moving tasks to hardware improves the predictability of their timing, but this is as much a result of the fact that a processing unit is dedicated to that one task, as it is to the fact that the task is implemented in hardware rather than software. These same tasks implemented in parallel hardware can also be implemented in parallel software through the use of multiple, small CPUs rather a single larger, faster CPU. In an rSoC, each small CPU could be customised to its individual task, and also customised to deal with the specific inter-processor communications required for this specific application.

4.4.7. Customised memory structures. To accompany multiple CPUs, there might need to be a shared memory system which allows different tasks to share data in an efficient way. In a general purpose multi-processor architecture, memory architectures tend to be very symmetric and regular. There is no need for such regularity with an application specific system, which can have combinations of multi-port and single port memories as needed.

5. Conclusions

Conventional desktop CPUs use techniques such as very high speed (>1GHz) CPU clocks, pipelining, multi-level caching, branch prediction and multiple instruction issue in order to produce very high average instruction per second figures. A simple rSoC processor cannot hope to compete with this raw, average computation rate.

Unfortunately, these same techniques which speed up average instruction rate of desktop CPUs also tend to make the worst case execution rate for individual tasks many times slower than the average. These systems do not handle large numbers of unpredictable context switches well. For this reason, rSoC processors are better able to compete with conventional processors in the real-time domain, and this is seen as a fruitful area for research exploration. In particular, there are many avenues for exploring how the additional programmable hardware resources on an rSoC can be used to advantage to improve real-time system performance.

This paper has identified many areas for additional research. It still remains for each of these areas to be more fully explored, and this will be the basis for significant research in the future. We are currently building a prototyping platform for real-time rSoC experiments that will allow us to pursue more of these ideas [9]. It is hoped that this paper might provide some insights so that others might also investigate what appears to be a very promising new research domain.

Acknowledgment. The support of the Australian Research Council for this work is gratefully acknowledged.

6. References

[1] Xilinx, "Xilinx FPGA Product Tables", on-line at www.xilinx.com

[2] Triscend, "A7 Configurable System-on-Chip" on-line at www.triscend.com

[3] Altera, "About Excalibur Embedded Processor Solutions" on-line at www.altera.com

[4] "FPSLIC – Field Programmable System Level Integrated Circuits" at www.atmel.com

[5] G. Berry, "The Esterel v5 Language Primer" at ftp://ftp.esterel.org/esterel/pub/papers/primer.pdf

[6] N. W. Bergmann, G. Brebner, and J.P. Gray, "Reconfigurable Computing and Reactive Systems" *Proceedings of the Australasian Workshop on Parallel and Real-Time Systems: PART '00*, Newcastle, November, 2000

[7] "M68HC11 Family", on-line at e-www.motorola.com

[8] K. Tindell, "Deadline Monotonic Analysis", Embedded System Programming, 13(6), June 2000

[9] N.W. Bergmann, J.A. Williams, Peter Waldeck, "Egret: A Flexible Platform for Real-Time Reconfigurable Systems-on-Chip", Engineering of Reconfigurable Systems and Algorithms: ERSA '03, Las Vegas, June 2003.