Hardware support for real-time reconfigurable system-on-chip

Peter J. Waldeck[a] and Neil W. Bergmann[a]

[a]School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, Australia

ABSTRACT

This paper introduces a computer architecture suitable for embedded real-time applications where low power consumption is a requirement. This is achieved through the use of a hybrid hardware-software system. A system architecture is proposed which allows for modules of a system to be implemented at run-time in either hardware or software. Implementation choices may be made dynamically based on the loading of the host microprocessor, in a multi-tasking environment. An approach to inter-module communication is described, along with how this is affected by dynamic configuration. Some research goals are identified, including investigating the effects on real-time performance, power consumption and the design process involved in reconfigurable systems.

Keywords: reconfigurable hardware, real-time systems, microblaze

1. INTRODUCTION

Modern complex embedded real-time systems require significant computational power while guaranteeing latency and timing performance. Guaranteeing the performance of a multi-tasked system often requires a far more powerful processor than is practical when low power consumption is required.

Hybrid hardware-software systems have a number of advantages over traditional microprocessor-based software systems or custom ASIC hardware solutions. The implementation of control-flow algorithms is difficult in hardware, while algorithms involving significant parallel arithmetic operations may be difficult to realize in a microprocessor-based software solution. Hybrid hardware-software systems allow for parallelism to be exploited in hardware, while leaving control of the overall system in software. This may result in superior real-time performance, as argued in an earlier paper.[1]

Various factors need to be minimized when designing embedded systems - cost of manufacture, cost of development, power consumption and many others. Use of hybrid hardware-software solutions may present advantages in these areas.

2. MOTIVATION

Hybrid hardware-software architectures have been investigated for over a decade. There are many examples of systems developed to take advantage the power of reconfigurable hardware. Systems such as GARP, ,[2] Chimaera ,[3] PipeRench [4] and MorphoSys [5] are typically aimed at general-purpose applications. They aim to augment the central processor's power with a reconfigurable array, thus achieving greater computational power.

Work has been done investigating dynamic configuration of hardware, aiming to treat hardware modules in the same manner as software - able to be swapped in and out of a hardware array.[6] Switching of computation between hardware and software implementations dynamically has not been investigated. The effects of this switching are also to be investigated.

Work has been performed examining the power requirements of reconfigurable architectures.[7] This work does not investigate the tradeoffs involved in dynamically switching between hardware and software approaches.

This research also aims to examine the effect on real-time performance (in terms of response time and determinism of the system) of introducing a dynamically reconfigurable hardware platform. This has not been investigated previously.

Further author information: (Send correspondence to Peter Waldeck.)
Peter Waldeck.: E-mail: waldeck@itee.uq.edu.au, Telephone: +67 7 3365 8305
Prof. Neil Bergmann: E-mail: n.bergmann@itee.uq.edu.au, Telephone: +67 7 3365 1182

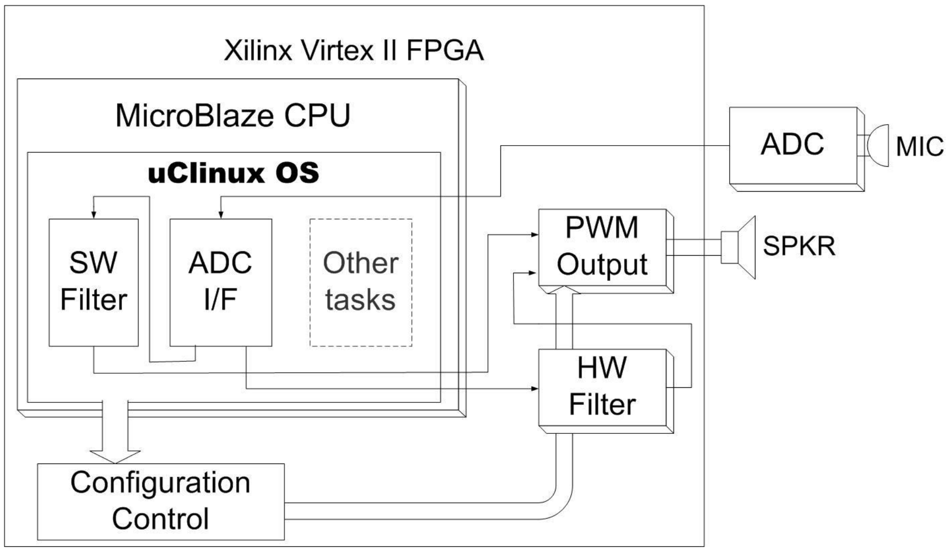

Figure 1. System-level architecture

3. SYSTEM ARCHITECTURE

Developing an architecture to meet the requirements of a low-power, embedded, multi-tasking real-time system presents a significant challenge. The system must be capable of adapting to different processing loads, while continuing to meet real-time deadlines.

Typical DSP-type applications require regular, repetitive operations to be performed on streaming data. This data may be required to be passed to a number of different functional units for calculations. Some of these calculations may be implemented in hardware functional units, thus freeing the processor to perform other tasks.

The proposed system consists of a soft processor (Xilinx MicroBlaze[8]) configured into the FPGA fabric, along with suitable communication structures - a high-performance Local Memory Bus (LMB), the lower data rate On-chip Peripheral Bus (OPB) and Fast Simplex Links (FSL) as required. The LMB allows guaranteed fast access to on-chip Block RAM (BRAM). Peripheral hardware devices can be located physically on the OPB, allowing for 32-bit transfers between the master processor and slave peripherals. Alternatively, peripheral hardware may be connected directly to the processor (or each other) by means of FSL's - dedicated 32-bit, point-to-point, unidirectional links, with built-in FIFO's. This range of communication structures allows the designer to use the most appropriate structure for each particular module.

An example architecture is shown in Figure 1. This example implements a very simple system - a filter capable of being implemented in either software or hardware, depending on the loading of the processor at the time. This system is intended as a proof-of-concept for further, more complex systems involving a number of modules.

3.1. Module Integration

All modules are to be developed in both software and hardware where possible - the combination of hardware and software used is under the control of the operating system. All modules will be present in hardware. Each of these modules is not required to be in operation at any particular time. Physically, the hardware modules are connected to others through a communication structure, consisting of FSL links. During periods of low processor load, hardware modules may be disabled and implemented in software, allowing power to be saved in the system. During periods of high demand on the resources of the processor, software modules may be switched to hardware, allowing for algorithms to be implemented in hardware without interaction with software. All

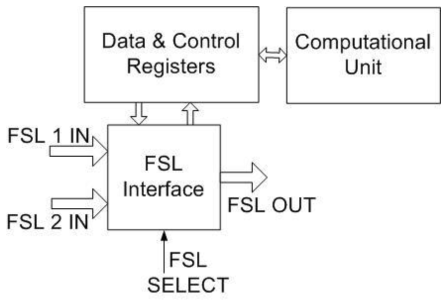

Figure 2. Block diagram of hardware module

hardware modules may be enabled under software control to allow for the construction of dynamic hardware-software hybrid systems.

3.2. Hardware Modules

Hardware interfaces must be simple to allow for customisation to specific applications without significant rework. They should also allow for flexibility to communicate data or control signals. To this end, a structure is proposed in Figure 2. This structure shows a computational unit, which will perform a fixed function, such as filtering, along with suitable control registers. These registers will allow simple control of the unit, such as setting the baudrate for an I/O device. The FSL interface allows connection of the module within the system. The input for this particular module may be selected from one of two sources at run-time, using the FSL SELECT signal. For example, these two sources could be the microprocessor and some other hardware module. Each module will need to be customised in order to appropriately connect suitable bus interfaces.

3.3. Software Modules

Software modules will be developed as processes under the control of the operating system. This is discussed further in the following section.

4. OPERATING SYSTEM

The operating system controlling the system is to be based on uClinux ,[9] a port of Linux to microprocessors lacking a memory management unit, such as MicroBlaze. It is a full-featured operating system, supporting networking and a variety of file systems. The role of the operating system will be to coordinate software processes and control configuration of modules appropriately. This will involve keeping a record of the implementation status of each module and performing the transitions between hardware and software implementations.

5. COMMUNICATION STRATEGIES

Communication between modules presents a unique challenge - each module requires no knowledge of other modules it is communicating with. This will allow for extremely flexible system configuration at run-time. Communication between modules falls into three categories: software-software (SW-SW), hardware-software (HW-SW) and hardware-hardware (HW-HW) communication. In order to allow for dynamic configuration of module interaction, this communication must be consistent across all categories. In order to achieve this, we must examine the nature of the information to be passed between modules.

5.1. Control signals

Control signals are usually simple boolean signals, determining the status of a module, or determining whether particular actions should be performed. In terms of SW-SW communication, simple inter-process message-passing protocols can allow for boolean signals to be exchanged. In HW-SW communication, the FSL interface allows for specific control signals to be communicated. HW-HW communication is achieved in the same manner - the FSL interface between hardware modules is identical to the interface with the microprocessor, allowing identical communication.

5.2. Data signals

Within most DSP-type applications there are two distinct types of data - signal data (e.g. audio or video) and calculation data (e.g. filter coefficients). The target system we plan to use is a telecommunications device, using audio data consisting of 8-bit samples being processed at 10kHz. Filter coefficients within the system must be updated after each audio sample has been processed. This regularity of processing allows for simple timed communication. SW-SW communication is achieved through the use of standard message passing. HW-SW communication is achieved through driver functions accessing the FSL. In the case of audio data, this is likely to take the form of single audio samples in one FSL transaction, due to the low bandwidth demand of audio signals (8-bit samples at 10kHz). Filter coefficients would be updated over the course of a number of FSL transactions, most likely transferring multiple coefficients per transaction, due to the higher bandwidth required. HW-HW communication will be performed in an identical manner due to the identical FSL communication interface.

6. DYNAMIC CONFIGURATION

Run-time configuration of the modules is a clear goal. This will allow for a system's computational power to vary dependent on which tasks are implemented in hardware. Run-time alterations to interconnections between modules will introduce overheads and therefore impact on the ability of the system to respond to external stimuli. Minimisation of these overheads is essential.

Analysis of the transitions reveals the likely extent of overheads, which will be largely dependent on the length and accuracy of the filters required by the system and hence the amount of data required to be transferred between software and hardware implementations.

6.1. SW to HW

Assuming a module is currently implemented in software, in order for this module to be transferred to hardware, a number of steps need to be taken:

- The computational unit of the hardware module needs to be initialised. This may involve operations such as transferring a set of coefficients to a hardware filter.
- The calculation must be finished in software before the module may be transferred to hardware. The overhead involved in transferring a filter calculation in operation is likely to be prohibitive. This results in a guaranteed delay of at most 1 sample.
- Communication structures need to be configured. This requires altering configuration of all modules interacting with the changed module in order to reflect the altered implementation status.

6.2. HW to SW

The transfer from hardware to implementation in software requires similar steps to the reverse operation:

- Again, we must wait until the calculation for that sample is finished before transferring to a software implementation.
- ”De-initialisation” of the hardware module - this will most likely involve reading out filter coefficients and resetting the filter to be available for future implementation.
- Communications structures again need to be changed to reflect the changed status of the module.

7. RESEARCH GOALS

This research aims to explore a number of questions. These include:

- **Determinism**- Real-time systems must be able to guarantee certain response times and/or execution times. Typically this will require a powerful processor which is idle most of the time, in order to guarantee the ability to meet these deadlines. This architecture may enable system designers to more easily verify worst-case execution times due to the ability of the hardware to dynamically increase effective processing power through enabling extra hardware units.
- **Power Consumption**- In modern embedded devices, particularly mobile devices, low power consumption is essential. Real-time systems typically require more computational power to implement a particular function, in order to be able to guarantee their response time under load. This system may allow a smaller, more power-efficient processor, while still guaranteeing real-time response.
- **Design process**- The design process involved in this architecture is relatively simple, once basic hardware blocks and associated software drivers are set up. If these are well designed, they could allow a system designer to build a custom system with ease. This will require a tightly defined interface methodology for both hardware and software modules in order to retain compatibility with previously developed modules.

  Scalability is an issue to be investigated - there are a fixed number of FSL channels on the processor (8 masters, 8 slaves in the current MicroBlaze), restricting the size of the module network able to be attached to the processor. Increasing the number of links on the processor requires an increase in the number of instructions, and hence instruction decoding logic.

The process of answering these questions is that of designing an example DSP system described in an earlier paper.[10] This system will allow us to test the real-time performance and power consumption of the architecture.

8. CONCLUSION

This paper has outlined an approach to embedded real-time system design which allows for powerful DSP algorithms to be implemented in either hardware or software, dependent on processor loading. Mechanisms for implementing modules of an algorithm in both hardware and software have been examined, along with requirements for dynamic configuration of hardware structures. This will allow for an operating system to alter the hardware configuration dynamically and safely. An example system will be built, allowing various issues to be investigated, including the effects on real-time performance, power consumption, and the design process.

REFERENCES

1. N. W. Bergmann, G. Brebner, and J. P. Gray, "Reconfigurable computing and reactive systems," *Australasian Workshop on Parallel and Real-Time Systems (PART '00)* , 2000. Newcastle, Australia.
2. J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines* , pp. 12–21, April 1997.
3. Z. A.Ye, A. Moshovos, S. Hauck, and P. Banerjee, "Chimaera: a high-performance architecture with a tightly-coupled reconfigurable functional unit," *Computer Architecture, 2000. Proceedings of the 27th International Symposium on* , pp. 225–235, June 2000.
4. S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "Piperench: A reconfigurable architecture and compiler," *Computer* **33**, pp. 70–77, April 2000.
5. H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and T. Lang, "Morphosys: An integrated reconfigurable architecture," *Proceedings of NATO RTO Symposium of System Concepts and Integration* , April 1999.
6. V. Nollet, J.-Y. Mignolet, T. Bartic, D. Verkest, S.Vernalde, and R. Lauwereins, "Hierarchical run-time reconfiguration managed by an operating system for reconfigurable systems," *Engineering of Reconfigurable Systems and Algorithms, ERSA'03, Proceedings of the International Conference on* , pp. 81–87, June 2003.

7. M. Wan, H. Zhang, M. Benes, and J. Rabaey, “A low-power reconfigurable data-flow driven dsp system,” *Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on* , pp. 191–200, October 1999.
8. “Microblaze soft processor,” 2003. http://www.xilinx.com/microblaze.
9. “uClinux Embedded Linux/Microcontroller Project,” 2003. http://www.uclinux.org.
10. P. Waldeck and N. Bergmann, “Dynamic hardware-software partitioning on reconfigurable system-on-chip,” *System-on-Chip for Real-Time Applications, 2003. Proceedings. The 3rd IEEE International Workshop on* , pp. 102–105, June 2003.