Egret: A Flexible Platform for Real-Time Reconfigurable Systems on Chip

Neil Bergmann, John Williams, Peter Waldeck
School of ITEE, University of Queensland,
Brisbane, Australia

Abstract: The new technology of reconfigurable System-on-Chip is a good match to the requirements of real-time embedded systems. However, the complicated design of such systems remains an obstacle to the widespread practical adoption of this technology. One method for reducing the incremental cost of new designs is to develop a generic platform for the design, prototyping and implementation of reconfigurable real-time systems. This paper presents our requirements for this platform, which we have dubbed Egret (Experimental Generic Reconfigurable Embedded Target). We describe our initial thinking on the design of this platform, and outline some of the high-level design choices made so far.

I. Introduction

Field Programmable Gate Arrays (FPGAs) have long been a popular implementation technology for random logic, glue logic and microprocessor peripherals.

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 of 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 and interfacing chips are added to the integrated CPU-plus-peripherals chip.

rSoC technology is seen as being particularly useful for a number of different scenarios, such as embedded systems where a single chip microcontroller is unavailable with the required set of peripherals, prototyping of ASIC SoC solutions in embedded applications and for cases where the range of peripherals changes in different operating modes.

Another scenario of particular interest to us is for implementation of real-time embedded systems, where custom hardware peripherals can improve real-time response rates, and the ability to create customised computer architectures facilitates rapid application prototyping.

II. Real-Time System-on-Chip

We have argued in an earlier paper [5] that reconfigurable computing can provide significant advantages for real-time computing, and we summarise the core of the argument here.

The goal of real-time systems, especially hard real-time systems is often to guarantee that a response can be made to an input signal by a fixed deadline, even in the worst case situation. Worst case calculation times tend to be more important than average instruction execution rates. 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 response times even for very rare conjunctions of events.

Being able to respond to inputs with customised hardware modules has obvious advantages. Many parallel 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.

Our research goal is to clearly demonstrate and prove these advantages through theoretical analysis and practical case studies.

III. FPGA Platforms

Modern high speed and high-capacity FPGAs, with hardwired processor, memory and peripherals, provide a type of design platform for reconfigurable computing, since they reduce the number of choices about how to implement particular functions.

Xilinx's Lysaght argues that in order to manage the complexity of modern mega-gate FPGAs, and to produce new designs quickly and efficiently, it will become essential for companies to develop their own higher level reconfigurable computing platforms [6]. This includes hardware platforms (perhaps particular types of FPGAs, and particular peripheral libraries), software platforms (such as operating systems), and design tool flows (for software, hardware, and systems).

As part of our research into reconfigurable computing for real-time systems, we have commenced the design of such a platform for real-time systems, and we present our initial work in the rest of the paper.

IV. Platform Specifications

No platform can be universal in its applicability, even within the domain of reconfigurable computing for real-time systems. We have chosen a particular slice through the design universe for our platform, which we call Egret (Experimental Generic Reconfigurable Embedded Target).

The primary objectives for our platform design are twofold:

- To further our research into reconfigurable system-on-chip for real-time systems, and
- To provide a platform that students (especially undergraduate and coursework masters project students) can use to rapidly prototype new reconfigurable, embedded computing applications.

Based on these objectives, we have first specified some general requirements, and are in the process of deciding how best to implement these.

A. *Modularity*

A modular system is one in which a particular system can be composed from a selection of modules from within a larger general purpose pool of modules.

In the case of Egret, we require modularity in three domains:

- Physical modularity allowing arbitrary combinations of core and peripheral modules to be plugged together,
- Logical modularity permitting communication and control with the physical modules, and FPGA IP cores assembled to implement the required set of processors and on-chip peripherals required for the system,
- Software modularity providing device driver, networking, data management and application specific code requirements of the system.

B. *Flexibility and Extensibility*

We require that the hardware and software design of the platform should be easily extensible to handle systems which require different levels of memory and processing power, as well as different networking and external signal interfacing.

C. *Plug and Play*

It is required that (as far as practicable), circuit components should be able to be added in a plug-and-play fashion. For example, addition of a new hardware module with a network interface should automatically load the appropriate device drivers etc, into the network stack, without requiring a complete rebuild of the operating system.

In lieu of full run-time hot-swap capabilities, our immediate goal is a platform which performs power-on module identification, and configures the software operating system accordingly. In this scenario it is reasonable to assume that the rSoC FPGA itself is appropriately configured at design time with the requisite on-chip peripherals.

D. *Vendor Independence*

The platform should not mandate the choice of a single vendor's FPGAs. This implies some kind of hardware abstraction layer, such as that offered by a conventional modern operating system.

The first instantiation of the platform is likely to be for one particular vendor, however design decisions made during the prototyping process must keep the vendor independence goal in mind.

E. *Simple Design Tool Chain*

Our experience of reconfigurable system-on-chip design is that it is complicated, and has a very steep learning curve. We require that our platform support a simple design tool chain, so that simple real-time embedded system designs can be accomplished on the platform without excessive lead times. Conversely, experienced designers should be able to make full use of the capabilities of the reconfigurable computing fabric.

F. *Reconfigurability*

As far as it is practical, the platform should take advantage of the design flexibility offered by the use of a central reconfigurable system-on-chip. Existing modular computer architectures tend to utilize shared bus architectures (e.g. PCI, ISA etc), via which peripherals communicate with the CPU and each other. A side-effect of this is that every physical module must support the entire wide bus interface. Instead, our approach is to focus peripheral control as much as possible in the central rSoC, using logical peripheral modules as described above. The physical interface to modules is made as narrow as possible, simplifying module design.

G. *Research Support*

The platform is not primarily a platform for prototyping commercial designs. Instead, it needs to be able to support our current and planned research projects. These include:

- Hardware-Software tradeoffs in reconfigurable system-on-chip implementations of real-time systems,
- Flexible on-chip interconnection networks for reconfigurable system-on-chip,
- Flexible, re-usable microprocessor peripheral core designs for reconfigurable system-on-chip, and
- Applicability of reconfigurable system-on-chip for particular application domains, including audio processing, video processing, aerospace and satellite systems.

H. *Ease of Manufacture*

Whilst it is not envisaged that any of our initial designs will immediately be converted to commercial products, the platform should support a reasonable commercialization path.

This would entail keeping the same logical system architecture but instantiating it on custom circuit boards. The rSoC microprocessor/peripheral cores and operating system/application software should migrate relatively unchanged.

V. Platform Design

Having provided some specifications, we next investigate our initial design choices for our platform.

A. *Software Operating System*

The choice of an appropriate operating system (OS) environment will drive many of the other design choices. Coarsely there are two competing approaches –

microkernel vs. operating system. Microkernels have the benefit of small memory footprints, good support for real-time scheduling, but suffer from limited support and tools and non-standard development and operating environments. On the other hand operating systems are usually well supported and provide familiar development environments, but have large memory footprints and often incur significant execution time overheads. Another axis of choice is commercial vs. open source.

One complication is the rather simple hardware architectures of most rSoC processor cores, in particular the lack of memory management units (MMUs) to implement virtual memory. This excludes many Unix-like systems, including FreeBSD, Mach and off-the-shelf Linux.

uClinux (*"you-see-linux"*) is a port of the Linux operating system to embedded processors lacking an MMU [7]. Originally targeting the Motorola 68K series of processors, it now supports several architectures including ARM.

In common with most Unix-like operating systems, uClinux lacks hard real-time support. However, this can be achieved through use of the real time extensions such as RTAI and RTLinux, wherein the operating system runs as a low priority user-mode process [8]. The combination of uClinux and RTAI/RTLinux is a promising approach, and is discussed further below in the context of the embedded processor selection.

From an application programming perspective, uClinux offers an interface almost identical to standard Linux, including command shells, C library support and Unix system calls. Thus, the platform can mature gracefully onto more powerful embedded processors in a reasonably transparent manner.

B. Embedded Processor

The pool of potential processors for the platform includes soft and hard cores from a variety of vendors, such as the PowerPC (hard-core) and Microblaze (soft-core) from Xilinx, various ARM platforms (hard-cores from Altera and Triscend), and the NIOS soft-core, also from Altera. Other processors include the venerable 8051 and AVR. Third party soft-core microprocessors are also plentiful.

Based on the factors presented in the previous section, we have decided to target uClinux, initially on the Xilinx Microblaze architecture, with a view to introducing the RTLinux extensions for real-time support. At the time of writing the uClinux port is more or less complete.

C. Modules - functionality

To maintain maximum flexibility, we aim for a large selection of relatively small and simple boards (modules) which are assembled together to prototype a specific system.

Our planned initial set of modules is as follows:

- rSoC core – contains the FPGA with CPU, plus sufficient RAM and Flash to run the operating system, and base configuration of the FPGA hardware,
- Memory boards – provide different combinations of RAM and Flash memory,
- Networking boards – initially offering 10/100 Megabit Ethernet, and

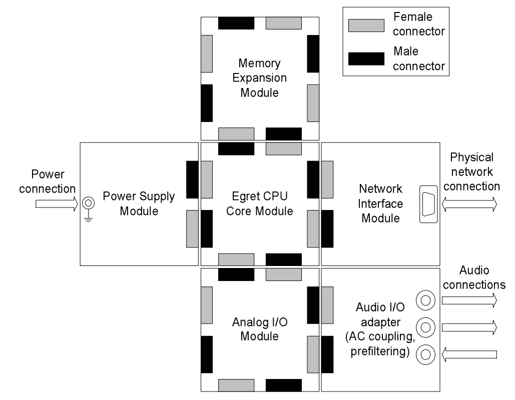

Figure 1. Example tiling layout for planar topology.

- General purpose I/O (GPIO) board – adds a number of general purpose digital and analog inputs and outputs, including audio.

Future modules potentially include video I/O and camera interfaces, multichannel audio I/O, networking modules such as wired (CAN bus, etc.) and wireless (Bluetooth, IEEE802.11 etc.), navigation and positioning (GPS) and application specific transducer modules.

D. Logical Interfaces

Any one prototyped system is likely to consist of several modules with a single rSoC core. Ultimately the goal is for an automated self-configuration process to identify connected hardware modules at startup, configure the FPGA with appropriate peripheral driver cores (such as I2C, SPI and one-wire, or custom peripheral interfacing cores), and configure operating system software drivers accordingly. For maximum flexibility this process may also support negotiation between the rSoC core and each peripheral module over which physical interface pins to use.

E. Physical Interfaces

Initially we are not especially interested in physical miniaturisation, and the first version of the platform, Egret v0.1 will use 75mm x 75mm square circuit boards, with connectors potentially along all four edges. The rSoC core would always have all four edges populated with connectors.

Two potential physical interconnection schemes have emerged – horizontal tiling and vertical stacking. Each has benefits and drawbacks which we discuss briefly below. The cardinal compass directions North, South, East and West denote the four sides of the Egret core.

1) Horizontal Tiling

Boards are designed so that they can tile the plane in any configuration. The connectors along each edge of every board are identical, and (at least physically), any two boards can be plugged together. Each edge has symmetrical male and female connector beside each other**.** Figure 1 shows an example tiling arrangement, while the connector structure is represented in Figure 2.

Benefits of this approach include easy physical access to boards and components for hardware debugging, and fairly simple physical connector and circuit board design. A challenge associated with horizontal tiling is that to support configurations with greater than four modules fanning around the Egret core, modules must offer some through-connect capability.

2) *Vertical Stacking*

Under this scheme, boards are designed to stack vertically. All signals connect vertically through the edges of each board, and each module taps into a subset of these signals to achieve its interface with the CPU. Each edge has four sub connectors, a male and female pair on the bottom and on the top. In this way multiple modules stack on top of each other, with four rotational symmetries.

All pins on all edges are common between modules. To share this resource, modules have a so-called "active edge". During stacking, modules are rotated such that their active face connects with one side (N, S, E or W) of the Egret core. If just one half of an edge is sufficient for a particular module then such modules may also be flipped, doubling to eight the number of stacking orientations.

For Egret v.0.1 we have opted for the planar tiling approach, mainly to simplify CPU and peripheral module development and debugging. Stackable modules will be investigated as a possibility in subsequent revisions.

F. Connector Pin Assignments

For either the tiling or stacking configurations, careful design of the electrical (pin) interfaces is required. To physically achieve the general tiling capability, symmetrical structures of male and female connectors are required. Within this, symmetry in electrical connections is also necessary. Thus, all module identification and power/ground signals must be duplicated and reflected in a manner similar to that illustrated in Figure 2.

Otherwise, pins on connectors are unreserved, and each peripheral module is free to use them as it sees fit. The CPU card has all of these uncommitted pins connected to FPGA user I/O pins, which can be reprogrammed as inputs, outputs, left tristate, or driven to ground for improved EMC.

To ensure that all connector edges are compatible, voltages are duplicated on a pair of male and female connectors. Ground pins are liberally interspersed among signal pins for protection against electromagnetic interference.

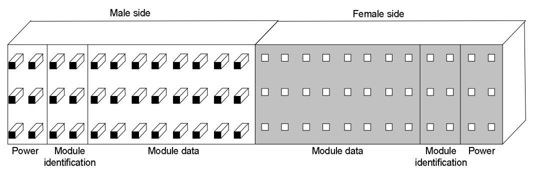

Figure 2. Illustration of symmetrical dual male/female connector scheme, showing power, module identification, and module specific pins.

VI. Conclusions

Egret is a new architecture designed specifically to support research and development into real time reconfigurable systems on chip. The high level requirements of the architecture have been developed and described.

In response to these specifications, a preliminary architectural design has been sketched out, and named Egret v0.1. The platform will run an embedded Linux-based operating system called uClinux with real-time (RTLinux) extensions. The port to the Xilinx Microblaze softcore processor is almost complete.

A core suite of hardware modules has been chosen for development, and candidates for future modules identified. Future support for plug-and-play run-time system reconfiguration and other advanced capabilities is anticipated, but with a short term focus on achieving a working platform to be configured manually.

Egret is a work in progress. Current and future efforts include the port of uClinux and real time extensions to the Microblaze softcore processor, development of module identification protocols to support boot-time operating system configuration, and hardware design of Egret core rSoC module and primary peripheral modules (memory, ethernet and general purpose IO).

Acknowledgment

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

References

[1] Xilinx, "Xilinx FPGA Product Tables," 2003, <http://www.xilinx.com/products/tables/fpga.htm> [accessed 7th Feb., 2003].

[2] Altera, "Excalibur Devices," 2003, <http://www.altera.com/products/devices/arm/arm-index.html> [accessed 7th Feb., 2003].

[3] Atmel, "Field Programmable System Level Integrated Circuits," 2003, <http://www.atmel.com/products/FPSLIC/> [accessed 7th Feb., 2003].

[4] Triscend, "A7 Configurable System-on-Chip," 2003, <http://www.triscend.com/products/a7.htm> [accessed 7th Feb., 2003].

[5] N. W. Bergmann, G. Brebner, and J. P. Gray, "Reconfigurable Computing and Reactive Systems," presented at Australasian Workshop on Parallel and Real-Time Systems (PART '00), Newcastle, Australia, 2000.

[6] P. Lysaght, "FPGAs as Meta-Platforms for Embedded Systems," presented at IEEE International Conference on Field Programmable Technology (FPT '02), Hong Kong, 2002.

[7] A. Rubini and J. Corbet, *Linux Device Drivers*, 2nd ed: O'Reilly and Associates, 2001.

[8] M. Barabanov, "A Linux-based Real-Time Operating System," Masters Thesis, New Mexico Institute of Mining and Technology, Sorocco, New Mexico, 1997.