

Predictable Reactive Processors for Next Generation Computing: A Proposal

Partha S Roop
Centre for Embedded Intelligence (CEMBI)
Department of Electrical & Computer engineering
University of Auckland

School of Engineering Report Number 662
February 2008

SUMMARY

Researchers at Auckland University and Kiel University have been working on a family of processors they term as *reactive processors* [20]. Reactive architectures are specialized embedded processors that directly interact with the environment through a specialized instruction set architecture (ISA). The hallmark of reactive processing is timing predictability and efficient resource usage, using the synchronous model of execution as exemplified by the Esterel programming language. The initial idea was proposed by Salcic and Roop of Auckland University in 2002 and was subsequently explored and extended by the research team of von Hanxleden et al. at Kiel University. Predictable execution of embedded systems using processors that guarantee timing has also gained momentum through the concept of Precision timed or PRET machines recently advocated by Edwards and Lee at DAC 2007 [8]. While architectures since the early 80s went for average case performance increase as the main metric of success, Edwards and Lee argue that “it is time for a new era of processors whose temporal behaviour is as easily controlled as their logical function”. There is good evidence that the reactive processing approach has much to offer here, and the goal of the work proposed here is to investigate how the diverse approaches taken at Auckland and Kiel can be combined and extended. Specifically, we would like (1) to investigate alternative ISAs that offer predictable timing, starting from high-level programs given in Esterel and possibly other languages, such as Java, (2) to investigate suitable models for distributed reactive processing, and (3) to build the foundations for a family of next-generation reactive processors that guarantee predictable execution.

1. INTRODUCTION AND BACKGROUND

Ubiquitous embedded systems have caught the attention of researchers over the past decade. Typical embedded applications ranging from complex aircraft flight controllers to simple digital cameras require worst case guarantees on their timing performance and hence are called real-time systems. General-purpose processors, being highly speculative, are not ideally suited to implement such real-time embedded systems. Though many researchers are looking at the matching of application requirements to processor technology, the design of processors with predictability so that it directly matches the real-time applications is still an open research problem which is yet to be fully solved.

Architecture support for predictable execution is not a new idea however. The Multiple Active Context System (MACS) architecture [1] was proposed in 1991. MACS is a cache-less architecture where multiple task contexts are tracked in hardware and a round robin schedule is used to issue an instruction from a different task context in each cycle. Simulation results have established that MACS was able to match and some times exceed conventional cache based architectures. More recently, Simultaneous Multithreaded Processors (SMT) has been advocated for real-time scheduling [2]. SMT exploits dynamic pipeline scheduling to achieve both instruction and thread-level parallelism [3, 4, 5]. While this approach suffices for soft real-time systems, they are unsuitable for hard real-time systems due to the speculative nature of dynamic pipeline scheduling. SMTs, however may just exploit thread-level parallelism to achieve the same effect as MACS architecture. For example, in [6] a concurrent event handling architecture is proposed where an additional event handler thread is spawned so that the initial real-time task is not preempted.

More recently, Ip and Edwards [7] looked at the problem of providing ISA support for precise timing of software. They provide a deadline instruction that controls a set of cycle accurate timers and its parameters are a specific timer together with the count value of this timer. When executed, this instruction initiates the down counting of this timer and waits until the timeout happens. Once this happens, the counter is reloaded and the control reaches the instruction following the deadline instruction. Thus, if a deadline instruction is coded inside a loop, the period of the loop is at least that of the deadline. This technique will thus work for hard real-time systems requiring short deadlines that can't be achieved using the precision of periodic interrupts alone. However, this research is not a complete solution for predictable architectures for embedded computing.

The idea of predictable architectures has gained momentum with two leading researchers (Edwards and Lee) proposing the concept of a Precision Timed Machine (or the PRET machine) in the recent Design Automation Conference (DAC 2007) [8]. The Edwards and Lee proposal is driven by the fact that safety-critical embedded systems such as aircraft flight control need certification. The certification of such software is extremely expensive due to the following reason advocated by the authors "if a manufacturer expects to produce a plane for 50 years, it needs a 50-year stock pile of fly-by-wire components that are made from the same mask set on the same production line. Even a slight change might affect the timing and require the software to be re-certified". Logically, the authors go on to argue that the different levels of abstraction invented by computer scientists have failed to take timing into consideration. The PRET proposal while being driven by the same idea of predictable execution, differs from all these in a significant aspect. Unlike all existing approaches, PRET advocates the development of a suitable ISA that offers predictability. Existing proposals for predictable execution were primarily developed to aid RTOS implementation. PRET, on the other hand, argues that "nearly every abstraction provided by computing has failed our aircraft manufacturer". For example, while ISA is developed to hide hardware details from the software developer, ISA has no guarantees for preserving timing properties that are so critical at the level of the software. Similarly, the timing of an RTOS may affect the concurrent tasks and RTOS do not offer deterministic concurrency. Hence, PRET proposal advocates some solutions along the lines of: (1) the usage of ISA that achieves precise timing with low overheads, (2) introducing

timed semantics for programming languages and (3) offering deterministic concurrency that are a side effect of the programming language being used.

While the PRET proposal is a major shift from the last 25 years of RISC based computing, Auckland University researchers have been looking at the creation of PRET machines without having to forgo the body of work around RISC (such as the use of simple instructions which are kept separate from the usual memory access instructions). Rop at al. started looking at this problem since 2001 and developed the concept of reactive processors [9, 10]. A reactive processor is a specialized embedded processor that directly interacts with the environment through IO ports and a specialized instruction set architecture (ISA). Such processors have ISA support for preemption and priority resolution and do not support conventional interrupts. We have already established that such a design results in more compact code which is also more efficient compared to either interrupts or polling [10]. Subsequently, we looked at the problem of directly executing synchronous languages [11] using reactive processors. Synchronous languages have precise semantics and hence have been languages of choice for designing safety-critical embedded systems [12]. They offer both predictable execution while guaranteeing properties of determinism and reactivity. Hence, they are used in particular to program the air flight control system and nuclear power plant control software. The proposed approach of direct execution of synchronous languages allows almost one-to-one mapping of high-level language features on the ISA level of a reactive processor. This not only can improve code size and execution time, but can potentially improve verifiability at all design abstraction levels. While Auckland researchers worked primarily on improving the average case, researchers from Kiel University extended the idea of reactive processors towards predictable execution [13, 14] using the concept of WCET based execution. The research question we would like to pose based on this prior research is the following “Can we exploit and substantially extend the body of work around reactive processors to achieve predictable real-time processing of distributed embedded applications?” This is obviously a huge research question, also posed by Lee [15] in his recent proposal on Cyber Physical Systems (or CPS are time critical, distributed embedded systems that control many complex physical processes). Lee’s proposal is motivated by the limitations of all design abstraction levels of current computing where all these abstractions ignore timing. Hence, in the conclusions of his report Lee advocates “to fully realize the potential of CPS, the core abstractions of computing need to be rethought. Incremental increments will, of course, continue to help. But effective orchestration of software and physical processes requires semantic models that reflect properties of interest in both”. Reactive processors offer a paradigm shift from the conventional von Neumann model of interrupt oriented reactive interaction. Can we now utilize reactive processors to achieve PRET machines of the future and achieve some steps towards CPS?

2. History of Reactive Processors and Relationship to Predictability

2.1 Reactive Architectures from Auckland University: An embedded system continuously interacts with its adjoining environment (sensors and actuators). When implementing this interaction on conventional processors, we have to rely on either interrupts or polling. Polling is a simple mechanism that checks for sensor values at regular intervals and hence wastes CPU cycles. Interrupts, on the other hand, combine preemption and priority in a single package and are much more efficient compared to polling. The main drawbacks of interrupts are: (1) unpredictability in the latency of an interrupt and (2) the associated code is very complex and difficult to debug. Reactive processors were motivated by the need for providing alternative predictable and structured models of interactions with the environment and were developed by us at Auckland University. The main idea centred on an asynchronous hardware block, called the Abort Handling Block (or AHB). AHB deals with preemption and priority that works asynchronously with the main control unit of the processor. Preemption and priority were achieved using ISA extensions: we implemented a preemption instruction called ABORT that wraps a body of code. The body executes until the preemption condition happens. Once this happens, program control jumps to a specified point (called a continuation address) outside the body. Priority is achieved using simple nesting of ABORT instructions where the outer aborts have higher priority. Using these extensions we demonstrated that preemption and priority resolution can be achieved always within fixed bounded delay of at most one instruction cycle. The first set of processors called REFLIX [9, 10, 16] and REPIC [11] extended conventional

non-pipelined processors. However, these simple architectures demonstrated a very structured ISA driven approach to environment interaction that is both efficient and predictable [17]. Subsequently, we proposed a fully pipelined version called REMIC (or reactive microprocessor) [18] that was a custom built reactive microprocessor. REMIC had a three-stage pipeline, a reactive functional unit (RFU) for environment interaction through a set of direct signal lines and a few reactive instructions in the ISA. Figure 1 provides the top-level view of this architecture. Precise timing could be achieved in REMIC by associating timers with signal inputs and programming them through AWAIT, EMIT or ABORT instructions [18]. This is similar in spirit to what is proposed later in [7]

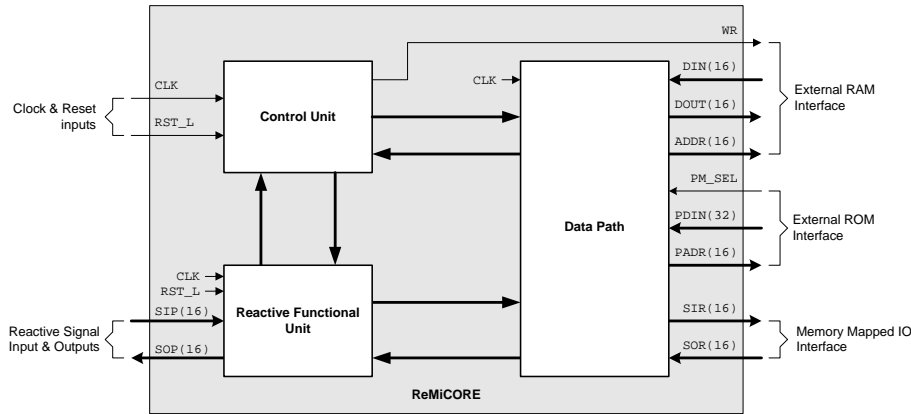


Figure1: REMIC reactive processor showing asynchronous RFU and the CU (reproduced from [19])

Subsequently, we proposed several reactive multiprocessors for handling both synchronous and asynchronous concurrent processes that are essential for supporting higher-level abstractions (language-level) of embedded computing. The first extension was an asymmetric multiprocessor called HiDRA (Hybrid Reactive architecture) [19] that combined a set of REMIC cores to provide direct support for a set of asynchronous concurrent tasks. A master processor can spawn these tasks on slave processors and point to point communication and synchronization between master and slave is achieved through reactive signals and associated reactive instructions (EMIT, AWAIT and ABORT). The corresponding reactive ISA, called CRAL (concurrent reactive assembler language), is a first attempt at providing architectural support for predictable mechanisms for asynchronous processes through direct ISA support (fixed latency of task creation, communication among tasks and so on). Subsequently, we developed a symmetric shared memory multiprocessor called EMPEROR (see Figure 2) by combining several reactive processors that were synchronized using a thread control block. EMPEROR, unlike HiDRA, provided an approach for distributing a set of synchronous threads [11] in a multiprocessor.

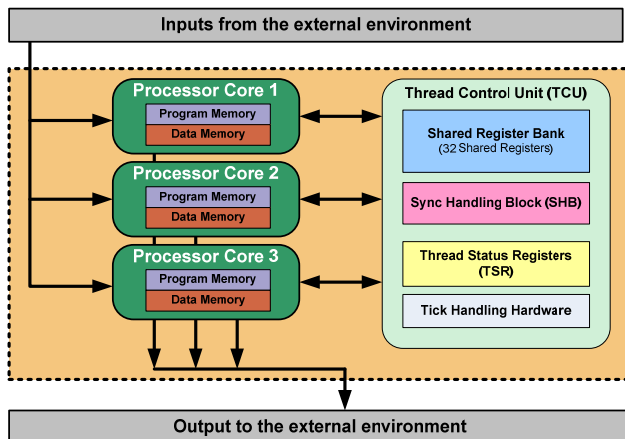


Figure 2: The EMPEROR Architecture.

Table 1 summarizes the primary difference between conventional and reactive processors.

<i>Features</i>	<i>Reactive Processors</i>	<i>Conventional Processors</i>
Execution progression	Evolves in discrete instants separated by “tick delimiting instructions”	Evolves continuously
Preemption	Accomplished through event reaction block with implicit priority resolution and context switching in hardware	Accomplished through interrupt mechanism requiring explicit priority resolution, context saving and restoration in software
Concurrency	Synchronous parallel execution and broadcast communication between threads	Asynchronous execution requiring explicit message passing/rendezvous for communication between threads
View of the environment	Changes at discrete instants. Inputs are latched at the beginning and outputs are sustained till the end of a “tick”.	Changes continuously Inputs can be read at any time, and outputs can be sustained for any duration.

Table 1: Reactive versus conventional processors (reproduced from [20]).

2.2 Link to high-level languages:

The reactive architectures developed at Auckland University were inspired by the well established synchronous language Esterel [21]. Synchronous languages are languages of choice for designing safety-critical embedded systems since they combine precise semantics with the synchronous execution model to avoid the well known *delta-delay issues* with VHDL like hardware description languages. All “correct” synchronous programs (in Esterel, for example) are always deterministic (will guarantee the same sequence of outputs for the same input sequence) and reactive (will not reject valid input stimulus to the system). Thus, such programs provide efficient techniques for formal verification [22]. However, software implementations of Esterel programs rely on conventional architectures with their usual deficiencies (discussed earlier) at the ISA level for providing predictability. Hence, an obvious (though non-trivial) approach should be to explore the usage of reactive architectures in compiling Esterel.

Esterel executes synchronously relative to ticks of a logical clock that provide synchronous execution of the Esterel threads. Preemptions are also synchronized with respect to these ticks (either at the start of the tick called *strong preemptions* or at the end of the tick called *weak preemptions*). Moreover, preemptions may or may not be immediate (at the first instant when they are active). This leads to quite complex preemption handling mechanisms in hardware. Moreover, there is a need to deal with surface and depth behaviours [23] and detection of signal presence to preserve Esterel semantics. The EMPEROR [11] architecture is a first attempt at providing ISA support for direct execution of Esterel. Here each processor executes until it hits a *pause* instruction which marks its *local tick*. The *global tick* (which corresponds to the Esterel tick) is detected using barrier synchronization in the TCU when all threads have reached their local ticks. Such a logical-tick based approach leads to quite compact code that has almost one to one correspondence with the original Esterel source. A compiler called EEC2 (EMPEROR Esterel compiler) [24] generates code for distributed execution using dynamic signal resolution along with surface and depth behaviours [23] in the control flow graph.

More recently, we have proposed a multithreaded architecture for direct execution of Esterel. The proposed processor, STARPro [25], has a synchronized Abort Handling Block (AHB) and a Thread Handling Block (THB) within an Esterel support Unit (ESU). The ESU maintains task contexts in hardware (THB) to emulate synchronous execution of Esterel using a single processor pipeline.

2.3 Related efforts on Reactive Processors:

While the Auckland architectures have primarily exploited the average case performance of Esterel, a parallel effort on ISA driven reactive computing is from Kiel University. Kiel researchers have also examined predictable execution of Esterel. They have proposed a family of processors, called the KEP (Kiel Esterel Processor) family [13, 14] that provide hardware support for Esterel using a set of hardware watchers for detecting aborts and support priority based scheduling of Esterel threads whose context is maintained in hardware. In fact, while the Auckland effort primarily looked at multiprocessor options (except for their latest processor that is multithreaded), the KEP family explored the multithreaded option from the start. Hence, they could support many more threads and preserved the semantics of the language faithfully. More importantly, they executed Esterel programs either using Auckland-like logical ticks or worst case execution time (WCET) based fixed tick lengths. The latter approach was proposed for predictable execution of tight control loops. The KEP family of processors from Kiel together with the STARPro processor from Auckland have demonstrated that it is possible to execute a high-level language that has direct features for timing and reactivity using PRET machines that faithfully preserve the high-level specification directly.

Orthogonal to this effort on reactive processing hardware is the research effort in the direction of virtual machines that effectively execute high-level specifications directly by having ISA support for such languages. Prominent amount such effort is the VM approach to the execution of Esterel in the CEC compiler of Edwards [26]. More recently, Malik et al. have developed a Tandem Virtual Machine (TVM) Approach [27] for the execution of a reactive Java extension called SystemJ. TVM combines a control virtual machine that can directly execute reactive constructs while interacting with the standard JVM that executes standard Java methods. The TVM approach, unlike the VM approach of CEC, can thus execute both control and data effectively and much more efficiently compared to standard JVM. The focus of the VM approach to Esterel execution is code size reduction while the TVM approach focuses on both efficiency and size of the resultant code. However, none of these VM approaches have explored the predictability issue.

In summary, initial reactive processors provided predictable environment interaction and priority resolution. Predictable timing could also be achieved through cycle accurate timers being connected directly to the signals of the processors [16]. Predictable execution of asynchronous processes explored in the HiDRA model. Later processors focused on direct execution of Esterel using either VM based or ISA based reconfigurable processor approach.

Questions still remaining: While WCET based execution of Esterel achieves some predictability, it is still not fully reliable (timing overruns may be a possibility). Also, is WCET analysis alone the best way of achieving predictable execution? Finally, is Esterel-like language the best option to move towards the future of predictable reactive computing? We would like examine these questions in the following.

3. OVERALL AIM OF THE RESEARCH

3.1 General and long-term goals:

To explore new abstractions for computing for next-generation embedded systems. To study the link between languages, networks, operating systems and ISA in relation to predictable execution semantics of future computing.

3.2 Specific objectives of the current research:

The intention of this proposal is to conduct a general investigation on (1) to investigate alternative ISAs that offer predictable timing, starting from high-level programs given in Esterel and possibly other languages, (2)

to investigate suitable models for distributed reactive processing, and (3) to build the foundations for a family of next-generation reactive processors that guarantee predictable execution.

3.3 Advancement of Knowledge and Understanding:

We strongly believe that next generation computing systems will be highly distributed embedded systems controlling diverse physical processes that demand predictable execution [15]. Such systems have been termed as Cyber Physical Systems or CPS by Lee [15]. The two proposals by Lee and Edwards [8] and Lee [15] argue for a rethink on design abstractions in computing to tackle such systems in the near future. We also believe that such efforts must start as soon as possible. However, this proposal argues that researchers at Auckland University and Kiel University have independently attempted to develop a new abstraction at the ISA level called reactive processing for over 5 years. This abstraction, however, provides only partial answers to realize the PRET machine dream first and CPS dream subsequently.

However, the objectives (1), (2) and (3) when achieved will lead to the realization of possibly the first prototype PRET machine and the associated software tools to program such machines. Hence, this proposal will be a leading activity in the world towards the achievement of some of the objectives of the ambitious CPS proposal. Once this project is completed, the investigators from Auckland and Kiel will continue to build on the findings of the current proposal to solve other critical issues in the CPS proposal such as networking and distribution issues.

4. RESEARCH QUESTIONS AND METHODOLOGY

Lee points out in [15] that “to realize the full potential of CPS, we will have to rebuild computing and networking abstractions”. This is obviously a big task and requires significant research effort from all major embedded system research groups globally. *The goal of this project is not to solve this challenge.* Rather, our goal is to closely examine the link between the CPS challenge in general and the PRET initiative in particular and the research on reactive processing carried out over the past five years. As discussed, reactive processing provided a new ISA oriented abstraction for better environment interaction that is highly predictable when compared with other abstractions on conventional processors such as interrupts. *Can we exploit the reactive abstraction to its fullest potential by developing reactive PRET machines?* We would also explore the link between such architectures and languages for the design of future distributed embedded systems.

We will exploit the following research questions during this project.

1. **ISA design issues:** What kind of ISA support is necessary for predictable reactive architectures?

The KEP3a architecture [13] may be considered as a predictable architecture as it can execute using fixed tick length that is determined using WCET analysis. However, WCET analysis is a very complex process and requires abstract interpretation [28, 29] (though reactive architectures are better candidates for WCET analysis of code). Without WCET analysis, timing overruns of the executing code is a possibility in KEP3a. We seek to explore reactive architectures where timing predictability is achieved without just relying on WCET analysis. These architectures provide instructions for signal manipulation (through memory mapped signals), preemption, priority, delay and local tick generation (pause) and thread management. Each of these instructions take fixed processor cycles. Precise timing of loops may possibly be achieved by associating cycle accurate timers with reactive instructions to achieve deadline and delay. However, the real challenge happens when we think of the link between the ISA and the language? Also, what kind of concurrent processes do we need to support and why?

2. **Language design issues:** What kind of models of concurrency to be supported at the architecture level for predictability, compositionality and distribution?

When dealing with reactivity at the ISA level, we started off by incorporating a RFU (reactive functional unit) [18] that worked asynchronously with the main control unit of the processor. Such asynchronous interaction leads to quite efficient and more importantly predictable implementation of preemption together with priority resolution. However, when trying to execute Esterel, we had to synchronize the RFU with the main control unit to achieve synchronous preemptions. This led to the introduction of an instruction called CHKABORT [11, 24, 25] which acted as a synchronization point between the two units. CHKABORT has to be inserted at specific points in the code during compilation to enable the checking of strong and weak preemptions. While moving from purely asynchronous reactive blocks to synchronous ones enabled Esterel like execution, it defeated the primary purpose of dealing with the asynchronous environment using an asynchronous block in the processor. This realization brings us back a “full circle” and in this proposal as we seek to revert back to the original asynchronous model. Hence, we need to make adjustments at the language level rather than the ISA level. As argued later, this will facilitate better language design for tackling both predictability and distribution – two key issues for CPS style future computing.

Since this research is a 6 month exercise, it appears prudent to stick to the synchronous framework. We have had several years of research experience both at Auckland [9-11, 16-20] and Kiel [13, 14, 20] on the direct execution of Esterel. However, as predictability was not our main concern, we continued with the full Esterel V5 language as our specification language. In this proposal we seek to revert back to the asynchronous interaction between the abort handling block and the control unit. Hence the research question would be – “what kind of adjustments do we need to make at the semantic level to realize a version of Esterel so that the tight synchrony between the reactive functional unit and the main control unit of the processor is no longer a necessity?” One possible option could be that all preemptions to events have a one tick delay (i.e., preemptions happen with respect to the status of an event in the previous instant). This will not only ease the compilation process, it may also facilitate better ISA that aids predictability.

Another issue is that of composition of synchronous processes (Esterel-like threads) [21]. This is essential, if we want to design distributed systems using Esterel seamlessly. Due to the synchronous model in Esterel (where a consumer thread may react to the events generated by a producer thread in the same instant), non-causal thread compositions are a possibility [12]. Hence, Esterel compilers must perform a causality analysis to reject such non-causal compositions as incorrect Esterel programs. This analysis is either very costly, in case full constructiveness analysis is performed, or quite conservative, in case one (conservatively) approximates causality by acyclicity. Distribution efforts on synchronous specifications currently generate a sequential program from the original concurrent source and then distribute this [30]. Hence, such approaches ignore the specification of concurrency in the original program. Can we achieve composition of several precompiled Esterel sources (separate compilation [31]) so that these programs may either be executed on a single processor or distributed across a network of processors (nodes) seamlessly? Some possible ways to deal with this may be to delay all reactions to events (both presence and absence) by one tick. This approach would be similar in spirit to the SL language [32] where reaction to presence is done synchronously while the reaction to absence is delayed. SL programs do run-time resolutions of signals as absence can be determined only at the end of a tick unlike presence (which is detected when a signal is emitted). This is unlike our recent approach to compilation and execution of IEC61499 function blocks [33] where we have proposed that response to only registered signals are allowed. As a consequence, we avoid the slow run-time resolution of signals in SL. Such approaches aid compositionality and may also help in separate compilation, which will be a key to achieve component-oriented designs using future synchronous languages.

Once the ISA design and language design issues have been tackled, we will embark on a new family of reactive processors and the associated tools. This objective will be only partially completed within the project duration and will be continued when the PI returns to UoA after his sabbatical.

Stages of this research

Stage A (January to March 2009): ISA design for predictability – This stage will include a detailed and up to date literature review on predictable architectures and languages for predictability. It will then design a new ISA and corresponding processor architecture. We will closely examine the link between existing reactive and predictable architectures.

Stage B (April to June 2009): Language Design for predictable execution – This stage will examine existing synchronous languages, compositionality and predictable execution models from such languages. We will then modify / restrict the semantics of an existing language to achieve predictable execution on the architecture developed in stage A.

REFERENCES

1. B. Cogswell and Z. Segall, "MACS: A Predictable Architecture for Real-Time Systems", in proc. Real-Time Systems Symposium, IEEE CS Press, 2001.
2. R. Jain, C. J. Huges, S. V. Adve, "Soft real-time scheduling on simultaneous multithreading processors", in proc. Real-Time Systems Symposium, IEEE CS Press, 2002.
3. S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous multithreading: A platform for next-generation processors", IEEE Micro, 17(5), pp. 12-19, 1997.
4. J. Lo, S. J. Eggers, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, "Converting thread level parallelism to instruction level parallelism via simultaneous multithreading", ACM Transaction on Computers, 15(3), pp. 322-354, August 1997.
5. J. Redstone, S. Eggers, H. Levy, "An Analysis of Operating System Behaviour on a Simultaneous Multithreaded Architecture", in proc. ASPLOS 2000.
6. S. W. Keckler, A. Chang, W. S. Lee, S. Chatterjee, and W. J. Daly, "Concurrent event handling through multithreading", IEEE Transactions on Computers, 48(9), pp. 903-916, 1999.
7. N. J. H. Ip and S. A. Edwards, "A Processor Extension for Cycle-Accurate Real-Time Software", in proc. of IFIP International Conference on Embedded and Ubiquitous Computing (EUC), 2006.
8. S. A. Edwards and E. A. Lee, "A Case for Precision Timed (PRET) Machine", in proc. DAC 2007.
9. Z. Salcic, P. S. Roop, M. Biglari-Abhari and A. Bigdeli, "REFLIX: A processor core for reactive embedded applications", in proc. 12th FPL, LNCS No. 2438, pp. 945-954, 2002.
10. P. S. Roop, Z. Salcic, M. Biglari-Abhari and A. Bigdeli, "A new reactive processor with architectural support for control dominated embedded systems", in proc. VLSI Design, IEEE CS Press, 2003.
11. P. S. Roop, Z. Salcic, M. W. S. Dayaratne, "Towards Direct Execution of Esterel Programs on Reactive Processors", 4th ACM International Conference on Embedded Software (EMSOFT 04), Pisa, Italy, September 27-29, 2004.
12. A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, R. De Simone, "The synchronous languages 12 years later", Proceedings of the IEEE, 91(1), 2003.
13. X. Li, M. Boldt and R. von Hanxleden, "Mapping Esterel onto a multithreaded embedded processor", in proc. 12th ASPLOS, 2006.
14. X. Li and R. von Hanxleden, "A concurrent reactive Esterel processor based on multithreading", in proc. ACM symposium on applied computing (SAC), 2006.
15. E. A. Lee, "Computing Foundations and Practice for Cyber-Physical Systems: A Preliminary Report", Technical Report No. UCB/EECS-2007-72, Department of Electrical Engineering & Computer Sciences, 2007.
16. Z. Salcic Z., P. S. Roop, M. Biglari-Abhari M. and A. Bigdeli, "REFLIX: A Framework of a Novel Processor Core for Reactive Embedded Applications", Elsevier Journal of Microprocessors and Microsystems, Vol. 28, pages 13-25, 2004.
17. A. Wahid, "Architectural Support for Real-Time Scheduling in Embedded Microprocessors", Master of Engineering thesis, University of Auckland, 2006.
18. Z. Salcic, D. Hui, P. S. Roop and M. Biglari-Abhari, "REMIC – Design of a Reactive Embedded Microprocessor Core", Asia-South Pacific Design Automation Conference, Shanghai, January 2005.
19. Z. Salcic, D. Hui, P. S. Roop and M. Biglari-Abhari, "HiDRA - A Reactive Microprocessor Architecture for Heterogeneous Embedded Systems", Elsevier Journal of Microprocessors and Microsystems, 30(2), pp. 72-85, 2006.
20. R. von Hanxleden, X. Li, P. S. Roop, Z. Salcic and L. H. Yoong, "Reactive Processing for Reactive Systems", European Research Consortium for Informatics and Mathematics News (ERCIM News), vol. 66, p28-29, 2006.
21. G. Berry and G. Gonthier, "The Esterel Synchronous Programming Language: design, semantics and implementation", Science of Computer Programming, 19(2), pp. 87-152, 1992.
22. A. Bouali, "XeVe: An Esterel Verification Environment", In proc. Of the 10th International Conference on Computer Aided Verification (CAV'98), vol. 1427, LNCS.
23. D. Potop-Butucaru, "Optimizations for faster simulation of Esterel programs", Ph.D. dissertation, Ecole des Mines, Paris, France, 2002.
24. L. H. Yoong, P. S. Roop, Z. Salcic, "Compiling Esterel for Distributed Execution", in proc. Synchronous Languages, Applications and Programming 2006, March 25- April 2, Vienna, Austria.

25. S. Yuan, S. Andalam, L. H. Yoong, P. S. Roop, Z. Salcic, "STARPro: A New Multithreaded Direct Execution Platform for Esterel", SLA++P, 2008 (accepted).
26. Stephen A. Edwards and Jia Zeng, "Code Generation in the Columbia Esterel Compiler", EURASIP Journal on Embedded Systems, 2007.
27. A. Malik, Z. Salcic and P. S. Roop, "SystemJ Compilation using the Tandem Virtual Machine Approach", ACM TODAES (under review).
28. H. Theiling and C. Ferdinand and R. Wilhelm, "Fast and precise WCET prediction by separate cache and path analysis", Real Time Systems, 18(2/3), pp. 157-179, 2000.
29. C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, S. Thesing and R. Wilhelm, "Reliable and precise WCET determination for a real-life processor", International Workshop on Embedded Software, EMSOFT 01, LNCS vol 2211, 2001.
30. A. Girault, "A survey of automatic distribution method for synchronous programs", in proc. International Workshop on Synchronous Languages, Applications and Programs, SLAP'05, ENTCS, Edinburgh, UK, April 2005.
31. K. Schnider, J. Brandt and E. Vecchie, "Modular compilation of synchronous programs", IFIP conference on distributed and parallel embedded systems, vol. 25/2006, Springer, 2006.
32. F. Boussinot and R. de Simone, "The SL Synchronous Language", IEEE Transactions on Software Engineering, 22(4), pp. 256-266, 2006.
33. L. H. Yoong, P. S. Roop, V. Vyatkin and Z. Salcic, "A Synchronous Approach for IEC61499 function Block Implementation", IEEE Transactions on Computers (under review).
34. Auckland Reactive Processors Page on <http://www.ece.auckland.ac.nz/~roop/ReactiveProcessors.php>
35. Kiel Esterel Processors Page on <http://www.informatik.uni-kiel.de/rtsys/kep/>