

# REMIC – Design of a Reactive Embedded Microprocessor Core

Zoran Salcic, Dong Hui, Partha Roop and Morteza Biglari-Abhari  
 Department of Electrical and Computer Engineering, Auckland University  
 20 Symonds St, Auckland, New Zealand, e-mail: z.salcic@auckland.ac.nz

**Abstract** – Reactivity on external events is an important feature of almost all embedded systems. In this paper we present the design of a new, reactive embedded microprocessor called REMIC, that supports reactivity in a new way following the paradigm of synchronous system level language Esterel. The rationale for REMIC design, its novel features with the design details and some performance figures are presented to demonstrate its suitability for embedded systems. Besides single processor systems, REMIC can be easily combined into multiple processor architectures that support real concurrency.

## I Introduction – Motivation and Background

Embedded systems are application specific systems that continuously interact with their environment and react to the events generated from the environment. The fast and efficient reaction to an external event required by these systems is called *reactivity* in this paper. Embedded applications may be broadly classified as either control-dominated or data-dominated. In control-dominated systems, input events arrive asynchronously and the arrival time of an event is very critical. Data-dominated systems, on the other hand, have inputs arriving at regular intervals and the value of an input event is more critical than its time of arrival. Real-life embedded systems are most often hybrid in their nature combining control- and data-dominated behaviors which also communicate with each other [1]. Data-dominated behaviors are usually nicely described using programming languages. Control-dominated behaviors, on the other hand, usually have to handle interrupts that represent the external random events. Moreover, a real-time operating system (RTOS) is needed to combine interrupt-driven and other parts of an application.

System-level languages, such as such as SystemC [2], and Esterel [3], can be used to describe heterogeneous embedded applications. Reactivity and concurrency are two of the main features of such languages. While they are suitable for specification capture and simulation, they are difficult to map on the target architectures that satisfy real-time and reactivity constraints.

Since reactivity is critical to control-dominated (part of) embedded systems, this paper first explores the suitability of architectures for reactivity. Reactivity demands efficient signal emission, signal polling, preemption support, priority resolution and concurrency support. Conventional way of signal emission and signal polling is through memory-mapped or IO-mapped schemes and using memory access or specialized IO instructions. These

approaches are inefficient compared to the use of direct signal lines and instruction set architecture (ISA) support for signal manipulations as advocated in this paper.

Preemption is fundamental to control-dominated systems for handling exceptions. Traditional processors use interrupts, which combine preemption with context saving, to handle such exceptions. Vectored multiple-level interrupts are also used for priority resolution. This flexibility, however, makes interrupts inefficient for reactive applications. An alternative mechanism called ABORT has been proposed in [4, 5], where handling of reactivity was mixed with control of traditional style processors. The proposed designs had no support for handling real concurrency. In this paper we propose a systematic approach for handling reactivity through a reactivity handling block that may be used in a plug-and-play fashion. The proposed processor architecture is fully pipelined and also supports concurrency through the use of multiple processor cores.

## II. The Need for Reactive Processors

System-on-Chip (SoC) is an approach for embedded systems design that creates the need for the design of custom (configurable) processors. Examples of configurable processor cores include X32V [6], Xtensa [7], ARC [8] and Altera NiOS [9]. They all use traditional architectures and mechanisms to interact with external environment. The need for processors that are suitable for handling both control-driven and data-driven behaviors in a more efficient way that it is done in traditional processors is demonstrated in [4, 5]. However, those processors are based on the existing and simple cores with the constraints of original cores. In this paper we present the design of a new reactive processor, called REMIC (**R**eactive **M**ICroprocessor) that is designed to extend the concept to multiple processor case and provide:

1. *Efficient mapping of control-dominated applications on processor ISA*, thus better supporting performance and code density for control-dominated applications.
2. *Support for direct execution of reactive language features*, thus reducing the overhead when compiling those languages onto target processor ISA.
3. *Support for concurrency and multiple processor configurations*, by providing mechanism for concurrent processes synchronization and communication.

## III. REMIC – Reactive Embedded Processor Core

REMIC design follows the main ideas of Esterel with a set of native instructions for control-dominated applications in addition to instructions found in traditional processors. These new native instructions provide direct support for *delay*, *signal emission*, *priority* and *preemption*. Concurrency is achieved by a multiple REMIC cores, where new signal manipulation instructions can be used to implement efficiently synchronization of concurrent tasks running on separate processors. The key REMIC features that facilitate reactive applications are:

- One Signal Input Port (SIP) and one Signal Output Port (SOP) enable direct mapping of Esterel pure (binary) signals [3]. Simultaneous emission of multiple signals in one clock cycle is supported in order to preserve the instantaneity principle of Esterel.
- Two user programmable internal timers are implemented to generate the timeout signals, which can be used internally or externally.
- ABORT instruction is introduced to handle Esterel-like preemption. Program code can be wrapped up in the abort statements, also allowing nesting of ABORTs, and immediately abandoned when an external event on the specified SIP input occurs.
- Other instructions including EMIT, SUSTAIN, PRESENT, SAWAIT, TAWAIT, CAWAIT are added to support Esterel-like reactive features (Table I).

Figure 1 illustrates the REMIC architecture and partition of functionality. REMIC consists of a traditional RISC-type pipelined microprocessor datapath, reactive functional unit (RFU) for handling external and internal signals, and processor control unit. The design is modular in a way that datapath with control unit (together referred to as the non-reactive core), can be used for customization by adding various functional units, the RFU being just one of them. REMIC has Harvard-type architecture with 32-bit wide program and 16-bit wide data memory.

#### A. Native Reactive Instruction Set

The hardware implementations of Esterel-like instructions result in better performance and much more efficient compilation of Esterel programs [10]. REMIC has a group of seven native instructions that support reactive processing. They are shown in Table I.

Table I Instructions supporting reactivity

Features	Instruction Syntax
Signal emission	EMIT <i>signal(s)</i>
Signal sustenance	SUSTAIN <i>signal(s)</i>
Signal polling	SAWAIT <i>signal</i>
Delay	TAWAIT <i>clock(s), prescaler</i>
Conditional signal polling	CAWAIT <i>signal1, signal2, address</i>
Signal presence	PRESENT <i>signal, address</i>
Preemption	ABORT <i>signal, address</i>

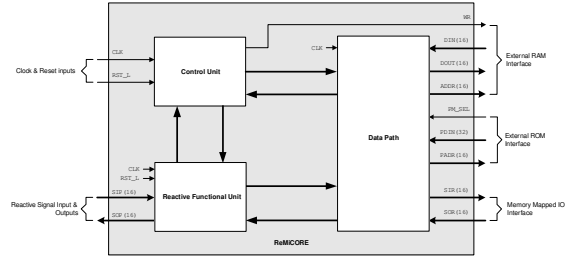


Figure 1 REMIC block diagram

All REMIC instructions are 32 bits long. Instruction formats of some reactive instructions are illustrated below.

#### EMIT - Signal Emission

EMIT, with the format as shown below, is used to generate external output signals through the SOP. The signals last for one clock cycle. Bits 24-9 of the instruction are mapped to bits 15-0 of the SOP.

31-30	29-23	24-9	8-0
AM(2)	OC(5)	Signals(16)	Unused(4)

#### SAWAIT – Signal Polling

SAWAIT, with the format as shown below, is used to poll for a specified signal from the SIP. REMIC stays in a wait state until the signal occurs in the environment.

31-30	29-23	24-9	8-5	4-0
AM(2)	OC(5)	Unused(16)	SIG(4)	Unused(4)

#### ABORT - Preemption

ABORT, with the format as shown below, is the most crucial reactive instruction because it is introduced to support preemption with priorities. An ABORT instruction becomes active from the instant it is executed until either (1) it reaches the continuation address, or (2) an event on one of the SIP inputs occurs that preempts all unexecuted instructions within the body. Bits 24-9 of the instruction specify the abort continuation address and bits 8-5 specify the abort signal that is encoded to one of the SIP inputs.

31-30	29-23	24-9	8-5	4-0
AM(2)	OC(5)	Continuation Address(16)	SIG(4)	Unused(5)

#### B. Example Program

In order to illustrate the effectiveness of the native reactive instructions we use the example of the pump controller, which is described as follows [11]:

“A pump controller is used to control the operation of a pump inside a mine which may have high methane levels. The pump is used to pump out water (whenever the water level exceeds the desired level) provided the methane level is below the desired level (RIGHT-METHANE). Whenever,

methane level goes above this desired level (NOT-RIGHT-METHANE), the controller must stop the pump and wait until right methane level is restored. If at any time, however, the methane level is too high (NOT-RIGHT-METHANE is only marginally high) then the pump must be stopped immediately and an ALARM must be generated. Pumping is stopped until right methane level is restored”.

A REMIC implementation of the specification, which includes declarations of signals, is shown in Figure 2. It maps into 13 program memory words. When using traditional processor instructions, the specification requires typically at least three times more instructions [5].

```

HighMethaneLevel  &EQU 0 ; SIP[0]
NotRightMethane   &EQU 1 ; SIP[1]
HighWaterLevel    &EQU 2 ; SIP[2]
LowWaterLevel     &EQU 3 ; SIP[3]
RightMethane      &EQU 4 ; SIP[4]
START_PUMP        &EQU $0001 ; SOP[0]
STOP_PUMP         &EQU $0002 ; SOP[1]
STOP_PUMP_ALARM   &EQU $0006 ; SOP[2]=ALARM

LOOP0  ABORT @HighMethaneLevel ACT_HML
LOOP1  ABORT @NotRightMethane ACT_NRM
LOOP2  SAWAIT @HighWaterLevel
        EMIT #START_PUMP
        SAWAIT @LowWaterLevel
        EMIT #STOP_PUMP
        JMP LOOP2
ACT_NRM EMIT #STOP_PUMP
        SAWAIT @RightMethane
        JMP LOOP1
ACT_HML EMIT #STOP_PUMP_ALARM
        SAWAIT @RightMethane
        JMP LOOP0
        &END

```

Figure 2 REMIC Program for Pump Controller Example

## IV. REMIC Design

### A. Datapath

REMIC datapath, shown in Figure 3, contains a number of registers used for data storage and internal operations:

- *Programmer visible registers.* The working register file (RF) contains 8 16-bit registers that can be used for data storage or as address registers. The stack pointer SP can be loaded with the LDSP instruction. With each stack operation, it is automatically incremented or decremented so that it always points to the next stack location in the data memory. The Flag registers are updated by each arithmetic operation.
- *Programmer invisible registers.* A number of registers are used for internal operations and not directly accessible by the user. These registers are, a 32-bit instruction register (IR), a 16-bit address register (AR), a 16-bit program counter (PC) and six pipeline registers (POP, POC, PSX, PSY, PSZ, PPC) used hold

the intermediate data required at different pipeline stages.

An extra stage, pipelined opcode register (PPOC), is added to support a new state in the control unit, which in non-reactive core has three states. The new state will be discussed in the next section.

### B. Abort Handling Block

The abort handling block (AHB), shown in Figure 4, is the main part of the RFU. Its control unit is implemented using a finite state machine in which each state represents one abort level starting from A0 to A4. A0 is the state where no ABORT is loaded and A1 is the state where the first ABORT is loaded and so on. Figure 5 illustrates the state diagram of the AHB control unit.

The AHB can be easily upgraded to support more than four levels of nested aborts by adding new states to the AHB control unit. Additional active abort status registers (AASR) and active abort address registers (AAAR) are needed to support the new states, each of which represents an additional nested abort level. Since the upgrade will not change the interface between the AHB and the rest of the processor, it requires no modifications on the processor control unit and datapath.

### C. Control Unit

An event handling mechanism is added to the non-reactive control unit to support the preemptive abort event. In the beginning of each state, the control unit checks for the occurrence of the pending abort event flag (PAEF) signal, and if detected, it loads the PC with the continuation address provided by the RFU and then restarts the pipeline operation. If not, the program execution continues in its normal flow. Unlike the interrupt handling mechanism that saves the return address before jumping to the service routine, the abort handling mechanism does not save any contexts and jumps to the continuation address immediately after the PAEF signal is detected.

### D. Multiple-core REMIC

An approach to support real concurrency using reactive cores has been proposed in [12]. A hierarchical architecture that contains a master REMIC processor and two slave REMIC processors is illustrated in Figure 6. Process(or) communication is performed using shared memory blocks between master and each of the slave processors, and synchronization is implemented using signals and reactive native operations on signals. By using signal emission, wait and signal preemption, various types of asynchronous communication channels and message passing primitives are easily implemented. This way the architecture effectively supports GALS type model of computation [12].

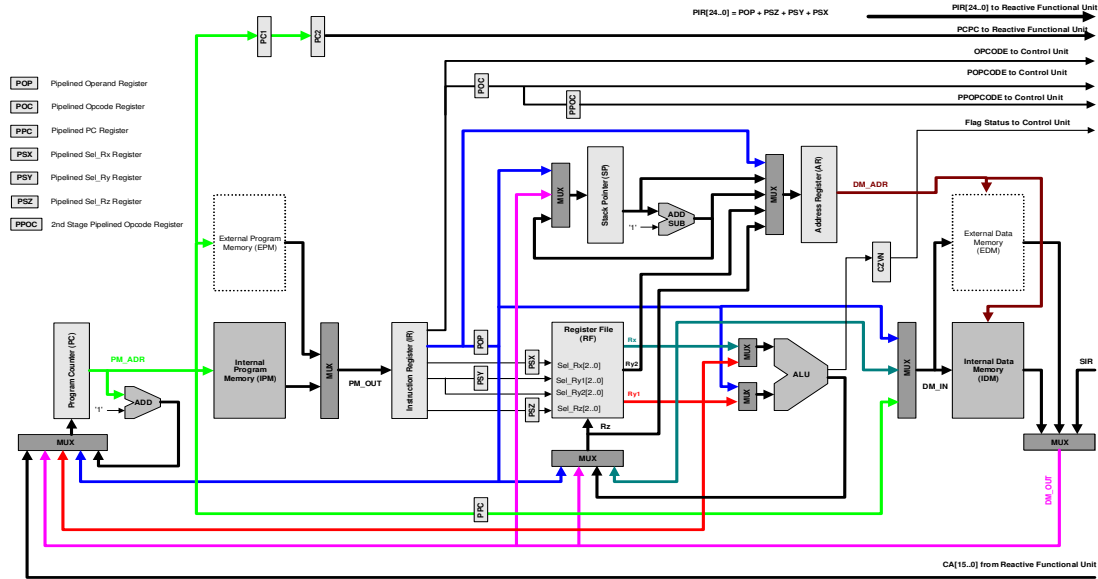


Figure 3 REMIC Data Path

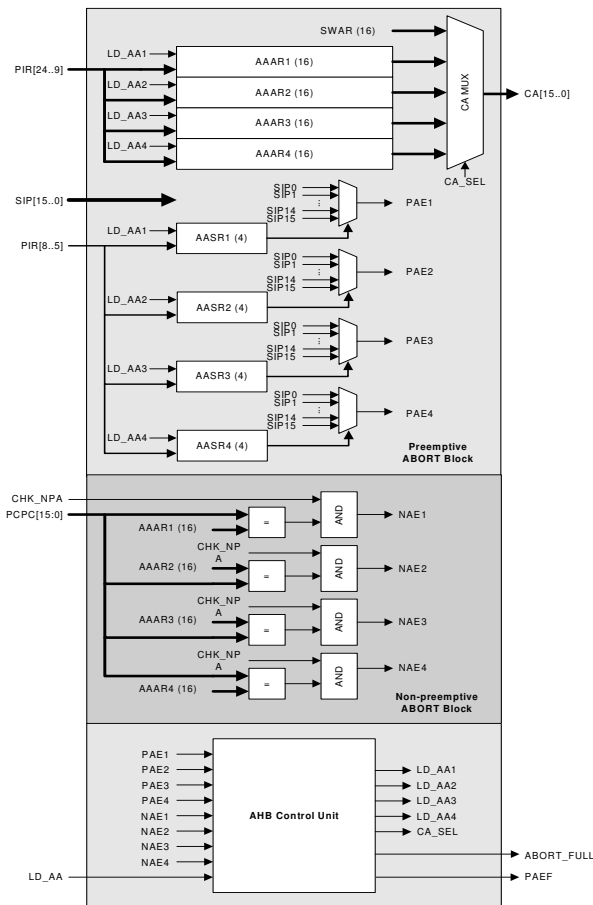
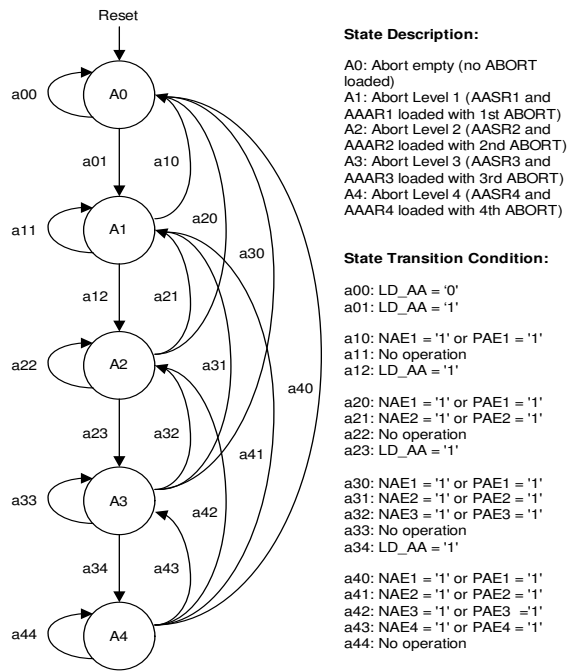


Figure 4 Abort Handling Block



**State Description:**

- A0: Abort empty (no ABORT loaded)
- A1: Abort Level 1 (AASR1 and AAAR1 loaded with 1st ABORT)
- A2: Abort Level 2 (AASR2 and AAAR2 loaded with 2nd ABORT)
- A3: Abort Level 3 (AASR3 and AAAR3 loaded with 3rd ABORT)
- A4: Abort Level 4 (AASR4 and AAAR4 loaded with 4th ABORT)

**State Transition Condition:**

- a00: LD\_AA = '0'
- a01: LD\_AA = '1'
- a10: NAE1 = '1' or PAE1 = '1'
- a11: No operation
- a12: LD\_AA = '1'
- a20: NAE1 = '1' or PAE1 = '1'
- a21: NAE2 = '1' or PAE2 = '1'
- a22: No operation
- a23: LD\_AA = '1'
- a30: NAE1 = '1' or PAE1 = '1'
- a31: NAE2 = '1' or PAE2 = '1'
- a32: NAE3 = '1' or PAE3 = '1'
- a33: No operation
- a34: LD\_AA = '1'
- a40: NAE1 = '1' or PAE1 = '1'
- a41: NAE2 = '1' or PAE2 = '1'
- a42: NAE3 = '1' or PAE3 = '1'
- a43: NAE4 = '1' or PAE4 = '1'
- a44: No operation

Figure 5 AHB Control Unit State Transition Diagram

V. Results

REMIC has been designed using traditional design flow and has been synthesized for the FPGA implementation. The results of synthesis and some program size comparisons

between non-reactive and reactive core and programs that implement the same functionality using standard microprocessors are shown in Table II.

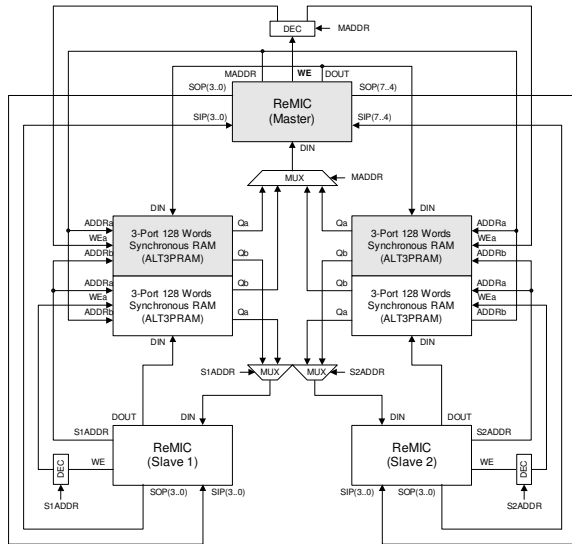


Figure 7 Multiple REMIC System

Table II Quantitative results and comparisons

EP20K200EF device	Non-reactive core	REMIC
Logic Elements	1240	2018
Maximum System Clock (MHz)	29.90	21.70

Application	Abort Level	Non-reactive core (words)	REMIC (words)
Seat Belt Controller	1	23	6
Pump Controller	2	60	13
Elevator Controller	0	86	25
Traffic Light Controller	2	68	16

Application	ReMIC (16-bit words)	8051 (bytes)	68HC11 (bytes)	NIOS16 (16-bit words)
Pump Controller	26	35	56	80
Elevator Controller	50	45	79	116
Traffic Light Controller	32	70	114	147

The multiple core REMIC architecture has been first prototyped in standard FPGA devices. Even the smallest APEX device [9] accommodates a configuration with a master and three slave cores. Availability of internal memory blocks in the FPGA device enabled full implementation of the multiple core circuit with the communication channels that are using shared memories. The performance of multiple processor ReMIC architecture was compared with the protocol stack implementation using MIPS R3000 in [13] is shown in Table III. The generated processor architecture has one master and one slave core. The memory requirements and execution times show substantial improvement, which may be attributed to direct support for reactivity and concurrency.

Table III Performance comparison of example application

Implementation	Data mem. (bytes)	Code size (bytes)		Exec. Time (cycles)	
		MP	SP	MP	SP
MultiCORE	64	160	24	810	28
MultiCORE (total)	64	184		838	
MIPS (1 task)	160	1008		4283	
MIPS (3 tasks)	352	1632		4161	

## VI. Conclusions

REMIC is a novel processor core that has built in features for handling pure binary input and output signals in Esterel sense. These features are provided to the user in the form of an extended ISA by increasing processor capabilities to deal with interaction with external environment. This makes programming of reactive situations much simpler compared to traditional programming techniques. It directly supports a number of key Esterel features and enables generation of very efficient machine code. The processor is suitable not only for single processor systems, but also for multiple processor architectures, effectively supporting real concurrency and GALS model of computation.

## References

- [1] S. Edwards et al., Design of Embedded Systems: Formal Models, Validation and Synthesis, Proceedings of the IEEE, Vol. 85, No. 3, 1997.
- [2] SystemC web site, [www.systemc.org](http://www.systemc.org)
- [3] F. Boussinot and R. de Simone, "The ESTEREL language," *Proceedings of the IEEE*, vol. 79, pp. 1293-1304, 1991.
- [4] Z. Salcic, P. S. Roop, M. Biglari-Abhari, A. Bigdeli, "REFLIX: A Processor Core with Native Support for Control Dominated Embedded Applications", Elsevier Journal of Microprocessors and Microsystems, Vol. 28, pp. 13-25, 2004
- [5] Z.Salcic and P.Roop, "Customizing Processors Cores to Support Reactivity", Proceedings of the International Conference on Engineering of Reconfigurable systems and Algorithms, CSREA Press, pp. 194-200, June 2004
- [6] D. Zier et al, "X32V: A Design of a Configurable Processor Core for Embedded Systems", Proceedings of the International Conference on Embedded Systems & Applications, CSREA Press, pp. 123-129, 2004
- [7] R.E. Gonzalez, "Xtensa: Aconfigurable and Extensible Processor", IEEE Micro, vol. 20, No. 2, March/April 2000
- [8] A. Senthia, "Solving System on Chip Design Challenges with the ARCform Development Platform", ARC Cores Ltd, San Jose, CA, 2001
- [9] Altera products literature, [www.altera.com](http://www.altera.com)
- [10] P.Roop, Z.Salcic, M.W.Dayartne, Towards Direct Execution of Esterel Programs on Reactive Processors, to be published in Proceedings of the ACM EMSOFT-04, Pisa Italy, September 2004
- [11] Gomaa H., *Designing concurrent distributed and real-time applications with UML*, Addison-Wesley, 2000
- [12] Z.Salcic, P.Roop, D. Hui and I.Radojevic, "HiDRA: A New Architecture for Heterogeneous Embedded Systems", Proceedings of the International Conference on Embedded Systems & Applications, CSREA Press, pp.164-170, June 2004
- [13] L. Lavagno and E. Sentovich, ECL: A Specification Environment for System-Level Design, DAC 1999.