A Multi-Threaded Reactive Processor

Xin Li  Marian Boldt  Reinhard v. Hanxleden

Real-Time Systems and Embedded Systems Group
Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Germany
www.informatik.uni-kiel.de/rtsys

ASPLOS’06
24 October 2006
Reactive vs. Non-Reactive Systems

Transformational systems *numerical computation programs, compilers* . . .

Interactive systems *operating systems, databases* . . .
Reactive vs. Non-Reactive Systems

Transformational systems: numerical computation programs, compilers...

Interactive systems: operating systems, databases...

Reactive systems: process controllers, signal processors...
Why “Reactive Processing”? 

Control flow on traditional (non-embedded) computing systems:

- Jumps, conditional branches, loops
- Procedure/method calls

Control flow on embedded, reactive systems: all of the above, plus
Why “Reactive Processing”? 

Control flow on traditional (non-embedded) computing systems:

▶ Jumps, conditional branches, loops
▶ Procedure/method calls

Control flow on embedded, reactive systems: all of the above, plus

▶ Concurrency
▶ Preemption
Why “Reactive Processing”?  

Control flow on traditional (non-embedded) computing systems:
- Jumps, conditional branches, loops
- Procedure/method calls

Control flow on embedded, reactive systems: all of the above, plus
- Concurrency
- Preemption

The problem: mismatch between traditional processing architectures and reactive control flow patterns
- Processing overhead, e.g. due to OS involvement or need to save thread states at application level
- Timing unpredictability
Reactive Processing Part I: The Language

Have chosen **Esterel**:

- Created in the early 1980's
- For programming control-dominated reactive systems
- Used as intermediate language for Statechart (Safe State Machines)
- Textual imperative language with reactive control flow constructs
  - Concurrency
  - Weak/strong abortion
  - Exceptions
  - Suspension
- A synchronous language
- Deterministic behavior, clean semantics
- Currently undergoing IEEE standardization
Reactive Processing Part II: The Execution Platform

- **Hardware**
  - Custom Hardware
  - Environment

- **Software**
  - COTS Assembler
  - COTS-μC
  - Environment

- **Co-design**
  - COTS Assembler
  - COTS-μC
  - Custom Hardware
  - Environment
Reactive Processing Part II: The Execution Platform

- **Hardware**
  - Custom Hardware
  - Environment

- **Software**
  - COTS Assembler
  - COTS-μC
  - Environment

- **Co-design**
  - COTS Assembler
  - COTS-μC
  - Custom Hardware
  - Environment

- **Patched Processor**
  - Extended Assembler
  - PIC Core
  - Extension
  - Environment

- **Esterel Processor**
  - Esterel Assembler
  - Esterel-μC
  - Environment
Why bother?

Reactive processing yields

- Low power requirements
- Deterministic control flow
- Predictable timing
- Short design cycle
Why bother?

Reactive processing yields

➤ Low power requirements
➤ Deterministic control flow
➤ Predictable timing
➤ Short design cycle

Can use reactive processor

➤ in stand alone, small reactive applications
Why bother?

Reactive processing yields

- Low power requirements
- Deterministic control flow
- Predictable timing
- Short design cycle

Can use reactive processor

- in stand alone, small reactive applications
- as building block in SoC designs
Overview

Introduction

The Kiel Esterel Processor
  The Esterel Language
  Instruction Set Architecture
  Processor Architecture
  Compiler

Experimental Results

Summary and Outlook
The Esterel Language

Logical Ticks

- Execution is divided into \textit{ticks}
- \textbf{Synchrony hypothesis}: Outputs generated from given inputs occur at the same tick

Signals

- \textit{Present} or \textit{absent} throughout a tick
- Used to communicate internally and with the environment
The Esterel Language

Logical Ticks

- Execution is divided into *ticks*

- **Synchrony hypothesis:** Outputs generated from given inputs occur at the same tick

Signals

- *Present* or *absent* throughout a tick

- Used to communicate internally and with the environment

```plaintext
module ABRO:
  input A, B, R;
  output O;
  loop
    abort
      [ await A
      ||
        await B ];
    emit O
    halt;
  when R
  end loop;
end module
```
The Esterel Language

Logical Ticks
- Execution is divided into *ticks*
- **Synchrony hypothesis:** Outputs generated from given inputs occur at the same tick

Signals
- *Present* or *absent* throughout a tick
- Used to communicate internally and with the environment

```
module ABRO:
  input A, B, R;
  output O;
  loop
    abort
    [ await A
      ||
      await B ];
    emit O
    halt;
  when R
end loop;
end module
```
The Esterel Language

Logical Ticks

- Execution is divided into ticks
- **Synchrony hypothesis:** Outputs generated from given inputs occur at the same tick

Signals

- *Present* or *absent* throughout a tick
- Used to communicate internally and with the environment

```
module ABRO:
  input A, B, R;
  output O;
  loop
    abort
    [ await A
    ||
    await B ];
    emit O
    halt;
  when R
  end loop;
end module
```
The Esterel Language

Logical Ticks

▶ Execution is divided into ticks
▶ Synchrony hypothesis:
  Outputs generated from given inputs occur at the same tick

Signals

▶ Present or absent throughout a tick
▶ Used to communicate internally and with the environment

module ABRO:
  input A, B, R;
  output O;
  loop
    abort
      [ await A
        ||
        await B ];
    emit O
    halt;
  when R
end loop;
end module

Tick

A

B

0
The Esterel Language

Logical Ticks

- Execution is divided into *ticks*
- **Synchrony hypothesis:** Outputs generated from given inputs occur at the same tick

Signals

- *Present* or *absent* throughout a tick
- Used to communicate internally and with the environment
The Esterel Language

Logical Ticks

- Execution is divided into *ticks*
- **Synchrony hypothesis:** Outputs generated from given inputs occur at the same tick

Signals

- *Present* or *absent* throughout a tick
- Used to communicate internally and with the environment

```
module ABRO:
  input A, B, R;
  output 0;
  loop
    abort
    [ await A
      ||
      await B ];
    emit 0
    halt;
  when R
  end loop;
end module
```
Candidates for the Instruction Set

Esterel kernel statements

- \( \| \)
- suspend \( S \) when \( S \)
- trap \( T \) in \( \) exit \( T \) \( \) end trap
- pause
- signal \( S \) in \( \) end
- emit \( S \)
- present \( S \) then \( \) end
- nothing
- loop \( \) end loop
- ;
Candidates for the Instruction Set

Esterel kernel statements

- \( || \)
- suspend \( \ldots \) when \( S \)
- trap \( T \) in \( \ldots \) exit \( T \) \( \ldots \) end trap
- pause
- signal \( S \) in \( \ldots \) end
- emit \( S \)
- present \( S \) then \( \ldots \) end
- nothing
- loop \( \ldots \) end loop
- ;

Derived statements

- [weak] abort \( \ldots \) when \( S \)
- await \( S \)
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

This redundancy improves space/time efficiency
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

This redundancy improves space/time efficiency

\begin{tabular}{|l|}
\hline
TOS: & \texttt{\% trap T in} \\
A0: & \texttt{\% loop} \\
PAUSE & \texttt{\% pause;} \\
PRESENT S,A1 & \texttt{\% present S then} \\
EXIT TOE, TOS & \texttt{\% exit T} \\
A1: & \texttt{\% end present} \\
GOTO A0 & \texttt{\% end loop} \\
TOE: & \texttt{\% end trap;} \\
\hline
\end{tabular}
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

*This redundancy improves space/time efficiency*

<table>
<thead>
<tr>
<th>T0S:</th>
<th>% trap T in</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0:</td>
<td>% loop</td>
</tr>
<tr>
<td>PAUSE</td>
<td>% pause;</td>
</tr>
<tr>
<td>PRESENT S,A1</td>
<td>% present S then</td>
</tr>
<tr>
<td>EXIT T0E, T0S</td>
<td>% exit T</td>
</tr>
<tr>
<td>A1:</td>
<td>% end present</td>
</tr>
<tr>
<td>GOTO A0</td>
<td>% end loop</td>
</tr>
<tr>
<td>TOE:</td>
<td>% end trap;</td>
</tr>
</tbody>
</table>
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

*This redundancy improves space/time efficiency*

```
TOS: % trap T in
A0: % loop
    PAUSE % pause;
    PRESENT S,A1 % present S then
    EXIT TOE, TOS % exit T
A1: % end present
    GOTO A0 % end loop
TOE: % end trap;
```

≡

```
AWAIT S % await S
```
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

*This redundancy improves space/time efficiency*

```
T0S:  % trap T in
A0:   % loop
      PAUSE % pause;
      PRESENT S, A1 % present S then
      EXIT TOE, T0S % exit T
A1:   % end present
      GOTO A0 % end loop
TOE:  % end trap;

AWAIT S % await S
```

- Refined ISA to reduce HW usage

**Example:** `abort` can translate to

- `ABORT` in the most general case
- `LABORT` if no other `[L]ABORTS` are included in abort scope
- `TABORT` if neither `||` nor other `[L|T]ABORTS` are included
The KEP Instruction Set

- Includes all kernel statements
- In addition, some derived statements

*This redundancy improves space/time efficiency*

```
T0S: % trap T in
A0: % loop
    PAUSE % pause;
    PRESENT S, A1 % present S then
    EXIT TOE, T0S % exit T
A1: % end present
    GOTO A0 % end loop
TOE: % end trap;
```

- Refined ISA to reduce HW usage

Example: `abort` can translate to

- `ABORT` in the most general case
- `LABORT` if no other `[L]ABORTS` are included in abort scope
- `TABORT` if neither `||` nor other `[L|T]ABORTS` are included

- Furthermore: valued signals, pre, delay expressions, ...
The Kiel Esterel Processor Architecture

- Reactive Core
  - Decoder & Controller, Reactive Block, Thread Block
- Interface Block
  - Interface signals, Local signals, ...
- Data Handling
  - Register file, ALU, ...

Xin Li, Marian Boldt, Reinhard v. Hanxleden
The Architecture of the Reactive Core

The Diagram illustrates the architecture of the multi-threaded reactive processor, including components such as Watcher0, Decoder, Watchers Prioritization Controller, Local Watchers, Thread Watchers, Await, Present, Scheduler, and various signals like Tick, innerOp, innerAddr, innerMark, innerData, countSPEC, wrPreemption, wrWAIT, InstrCLK, cclk, rdAbort, weakFlag, rdSuspend, etc. The processor is designed to handle multiple threads and reactive signals efficiently.
The Compilation Challenge: Thread Dependencies

```
module Example:
output O;
signal A,R in
|
  weak abort
  sustain R;
  when immediate A;
  emit O
|
  await R;
  emit A
|
end signal
end module
```
The KEP Compiler

Thread scheduling:

1. Construct Concurrent KEP Assembler Graph (CKAG)
2. Compute thread priorities/\( ids \) that respect dependencies
3. Generate PAR and PRI0 statements accordingly
The KEP Compiler

Thread scheduling:
1. Construct Concurrent KEP Assembler Graph (CKAG)
2. Compute thread priorities/ids that respect dependencies
3. Generate PAR and PRI0 statements accordingly

Other tasks:
- Analyze Watcher requirements
- Map Esterel statements to KEP refined ISA
The KEP Compiler

Thread scheduling:
1. Construct Concurrent KEP Assembler Graph (CKAG)
2. Compute thread priorities/\textit{ids} that respect dependencies
3. Generate PAR and PRI0 statements accordingly

Other tasks:
- Analyze Watcher requirements
- Map Esterel statements to KEP refined ISA

Optimizations:
- Dead code elimination, based on CKAG
- “Undismantling” of kernel statements
Introduction
The Kiel Esterel Processor
Experimental Results
Summary and Outlook

The Esterel Language
Instruction Set Architecture
Processor Architecture
Compiler

Example Compilation

module Example:
output O;
signal A,R in
[
  weak abort
  sustain R;
  when immediate A;
  emit O
  ||
  await R;
  emit A
];
end signal
end module

% module Example
OUTPUT 0
[L00,T0] EMIT _TICKLEN,#12
[L01,T0] SIGNAL A
[L02,T0] SIGNAL R
[L03,T0] PAR 2,A0,1
[L04,T0] PAR 1,A1,2
[L05,T0] PARE A2,2
[L06,T1] A0: WABORTI A,A3
[L07,T1] A4: EMIT R
[L08,T1] PRIO 1
[L09,T1] PRIO 2
[L10,T1] PAUSE
[L11,T1] GOTO A4
[L12,T1] A3: EMIT 0
[L14,T2] EMIT A
[L15,T0] A2: JOIN 0
[L16,T0] HALT
Example—Execution Trace

Scheduling criteria: 1. active, 2. highest priority, 3. highest id

module Example:
output 0;
signal A,R in
[  
  weak abort sustain R;
  when immediate A;
  emit 0
  ];
end signal
end module

- Tick 1 -
! reset;
% In:
% Out: R
T0: L01, L02, L03, L04, L05
T1: L06, L07, L08
T2: L13
T1: L09, L10
T0: L15
- Tick 2 -
% In:
% Out: A R O
T1: L10, L11, L07, L08
T2: L13, L14
T1: L09, L10, L12
T0: L15, L16
- Tick 3 -
% In:
% Out:
T0: L16
Overview

Introduction

The Kiel Esterel Processor

Experimental Results
  KEP Evaluation Platform
  Performance
  Scalability

Summary and Outlook
The KEP Evaluation Platform

- Highly automated process, currently using 470+ benchmarks
- End to end validation of hardware and compiler against “trusted” reference (Esterel Studio)
- Detailed performance measurements
Performance

Memory usage

- **Unoptimized:** 25–94% (83% avg) reduction of memory usage (Code+RAM)
- **Optimized:** Yield further 5% to 30+% improvements

Speed

- **WCRT speedup:** typically $>4x$
- **ACRT speedup:** typically $>5x$
- Optimizations yield further improvements

Power

- **Peak energy usage reduction:** 46–84% (75% avg)
- **Idle (= no inputs) energy usage reduction:** 58–97% (86% avg)
### The worst-/average-case reaction times comparison

<table>
<thead>
<tr>
<th>Module Name</th>
<th>MicroBlaze</th>
<th>KEP3a-Unoptimized</th>
<th>KEP3a-optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCRT V5</td>
<td>V7</td>
<td>CEC</td>
</tr>
<tr>
<td>abcd</td>
<td>1559</td>
<td>954</td>
<td>1476</td>
</tr>
<tr>
<td>abcdef</td>
<td>2281</td>
<td>1462</td>
<td>1714</td>
</tr>
<tr>
<td>eight.but</td>
<td>3001</td>
<td>1953</td>
<td>2259</td>
</tr>
<tr>
<td>chan_prot</td>
<td>754</td>
<td>375</td>
<td>623</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>487</td>
<td>230</td>
<td>397</td>
</tr>
<tr>
<td>runner</td>
<td>566</td>
<td>289</td>
<td>657</td>
</tr>
<tr>
<td>example</td>
<td>467</td>
<td>169</td>
<td>439</td>
</tr>
<tr>
<td>ww.button</td>
<td>1185</td>
<td>578</td>
<td>979</td>
</tr>
<tr>
<td>greycounter</td>
<td>1965</td>
<td>1013</td>
<td>2376</td>
</tr>
<tr>
<td>tcint</td>
<td>3580</td>
<td>1878</td>
<td>2350</td>
</tr>
<tr>
<td>mca200</td>
<td>75488</td>
<td>29078</td>
<td>12497</td>
</tr>
</tbody>
</table>

The worst-/average-case reaction times, in clock cycles, for the KEP3a and MicroBlaze

- **WCRT speedup**: typically $>4x$
- **ACRT speedup**: typically $>5x$
- **Optimizations can yield further improvements**
# Memory Usage

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel LOC</th>
<th>MicroBlaze Code+Data (byte)</th>
<th>KEP3a-Unopt. Code+Data (byte)</th>
<th>KEP3a-opt. Code (word)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V5 V7 CEC</td>
<td>V5 V7 CEC</td>
<td>V5 V7 CEC</td>
<td>V5 V7 CEC</td>
</tr>
<tr>
<td>abcd</td>
<td>160</td>
<td>6680 7928 7212</td>
<td>168 1.05</td>
<td>756 0.11</td>
</tr>
<tr>
<td>abcdef</td>
<td>236</td>
<td>9352 9624 9220</td>
<td>252 1.07</td>
<td>1134 0.12</td>
</tr>
<tr>
<td>eight_but</td>
<td>312</td>
<td>12016 11276 11948</td>
<td>336 1.08</td>
<td>1512 0.13</td>
</tr>
<tr>
<td>chan_prot</td>
<td>42</td>
<td>3808 6204 3364</td>
<td>66 1.57</td>
<td>297 0.09</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>27</td>
<td>2668 5504 2460</td>
<td>38 1.41</td>
<td>171 0.07</td>
</tr>
<tr>
<td>runner</td>
<td>31</td>
<td>3140 5940 2824</td>
<td>39 1.22</td>
<td>175 0.06</td>
</tr>
<tr>
<td>example</td>
<td>20</td>
<td>2480 5196 2344</td>
<td>31 1.55</td>
<td>139 0.06</td>
</tr>
<tr>
<td>ww_button</td>
<td>76</td>
<td>6112 7384 5980</td>
<td>129 1.7</td>
<td>580 0.10</td>
</tr>
<tr>
<td>greycounter</td>
<td>143</td>
<td>7612 7936 8688</td>
<td>347 2.43</td>
<td>1567 0.21</td>
</tr>
<tr>
<td>tcint</td>
<td>355</td>
<td>14860 11376 15340</td>
<td>437 1.23</td>
<td>1968 0.17</td>
</tr>
<tr>
<td>mca200</td>
<td>3090</td>
<td>104536 77112 52998</td>
<td>8650 2.79</td>
<td>39717 0.75</td>
</tr>
</tbody>
</table>

- **Unoptimized:** 83% avg reduction of memory usage (Code+RAM)
- **Optimized:** May yield further 5% to 30+% improvements
Scalability

Synthesis results for Xilinx 3S1500-4fg-676\(^1\)

<table>
<thead>
<tr>
<th>Thread Count</th>
<th>Slices</th>
<th>Gates (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1295</td>
<td>295</td>
</tr>
<tr>
<td>10</td>
<td>1566</td>
<td>299</td>
</tr>
<tr>
<td>20</td>
<td>1871</td>
<td>311</td>
</tr>
<tr>
<td>40</td>
<td>2369</td>
<td>328</td>
</tr>
<tr>
<td>60</td>
<td>3235</td>
<td>346</td>
</tr>
<tr>
<td>80</td>
<td>4035</td>
<td>373</td>
</tr>
<tr>
<td>100</td>
<td>4569</td>
<td>389</td>
</tr>
<tr>
<td>120</td>
<td>5233</td>
<td>406</td>
</tr>
</tbody>
</table>

- 48 valued signals
  - *up to 256 possible*
- 2 Watchers, 8 Local Watchers
  - *either up to 64 possible*
- 1k (1024) instruction words
  - *up to 16k possible*
- 128 registers (in word)
  - *up to 512 possible*
- 16-bits (65536) max counter value
- Frequency is stable (around 60 MHz)

\(^1\)For comparison, a MicroBlaze implementation requires around 1k slices and 309k gates; a two threads EMPEROR platform requires around 2k slices.
Summary Reactive Processors

Processor supports reactive control flow directly, at hardware level

- “Watchers” monitor preemption signals
  
  *No need for polling, interrupts*

- Support for concurrency
  
  *Multi-threading or multi-processing*

- Synchronous model of computation
  
  *Perfectly deterministic, predictable timing*
Related Work/Contributions

RePIC [Roop et al.’04]/EMPEROR [Yoong et al.’06]

- Multi-processing patched reactive processor
- Three-valued signal logic + cyclic executive
Related Work/Contributions

RePIC [Roop et al.’04]/EMPEROR [Yoong et al.’06]
- Multi-processing patched reactive processor
- Three-valued signal logic + cyclic executive

Kiel Esterel Processor 1–3
- Multi-threading custom reactive processor
- Provides most Esterel primitives, but still incomplete
- No compilation scheme to support concurrency
Related Work/Contributions

RePIC [Roop et al.’04]/EMPEROR [Yoong et al.’06]
- Multi-processing patched reactive processor
- Three-valued signal logic + cyclic executive

Kiel Esterel Processor 1–3
- Multi-threading custom reactive processor
- Provides most Esterel primitives, but still incomplete
- No compilation scheme to support concurrency

KEP3a (this work)
- Provides all Esterel primitives
- Refined ISA
- Compiler exploits multi-threading
Outlook

- Improve priority assignments
- Speedup signal expression computations with external logic block
- WCRT analysis with concurrency
- Extend to Esterel v7
- KEP in Esterel—e.g., to produce Esterel virtual machine
- Combination with multi-core (for data handling)
- Adaptation to non-Esterel languages
Outlook

- Improve priority assignments
- Speedup signal expression computations with external logic block
- WCRT analysis with concurrency
- Extend to Esterel v7
- KEP in Esterel—e.g., to produce Esterel virtual machine
- Combination with multi-core (for data handling)
- Adaptation to non-Esterel languages

Thanks!

Questions/Comments?
Appendix

KEP3a Instruction Set + Architecture
  Esterel-Type Instructions
  Handling Concurrency
  Handling Preemption
  Handling Exceptions
  WCRT Self-Monitoring

The Compiler
  Three Compilation Steps
  The Concurrent KEP Assembler Graph
  Cyclicity
  Constraints

Further Measurements
  Code Characteristics and Compilation Times
  Speed, Size, Power, Scalability
  Analysis of context switches
  Another Example

Summary
  Multi-processing vs. Multi-threading
  Comparison of Synthesis Options
  Application Scenarios
### Instruction Set Summary 1/2

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR Prio, startAddr [, ID]</td>
<td>$p \parallel q$</td>
<td>Fork and join. An optional ID explicitly specifies the ID of the created thread.</td>
</tr>
<tr>
<td>PARE endAddr</td>
<td></td>
<td>Set the priority of the current thread.</td>
</tr>
<tr>
<td>JOIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIO Prio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[W]ABORT [n,] S, endAddr</td>
<td>[weak] abort</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when [$n$] $S$</td>
<td></td>
</tr>
<tr>
<td>[W]ABORTI S, endAddr</td>
<td>[weak] abort</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when immediate $S$</td>
<td></td>
</tr>
<tr>
<td>SUSPEND[I] S, endAddr</td>
<td>suspend</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>when [immediate] $S$</td>
<td></td>
</tr>
<tr>
<td>EXIT TrapEnd[,TrapStart]</td>
<td>trap $T$ in</td>
<td>Exit from a trap, TrapStart and TrapEnd specify trap scope. Unlike GOTO, check for concurrent EXITS and terminate enclosing $\parallel$</td>
</tr>
<tr>
<td></td>
<td>exit $T$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end trap</td>
<td></td>
</tr>
</tbody>
</table>

$S$ can be one of the following:
1. $S$: signal status (present/absent)
2. PRE($S$): previous status of signal
3. TICK: always present

$n$ can be one of the following:
1. #data: immediate data
2. reg: register contents
3. ?$S$: value of a signal
4. PRE(?$S$): previous value of a signal
### Instruction Set Summary 2/2

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAUSE</td>
<td>pause</td>
<td>Wait for a signal. AWAIT TICK is equivalent to PAUSE</td>
</tr>
<tr>
<td>AWAIT [n,] (S)</td>
<td>await [n] (S)</td>
<td></td>
</tr>
<tr>
<td>AWAIT[I] (S)</td>
<td>await [immediate] (S)</td>
<td></td>
</tr>
<tr>
<td>CAWAITS</td>
<td>await [immediate] (S)</td>
<td>wait for several signals in parallel</td>
</tr>
<tr>
<td>CAWAIT[I] (S, addr)</td>
<td>case [immediate] (S) (do) (end)</td>
<td></td>
</tr>
<tr>
<td>CAWAITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGNAL (S)</td>
<td>signal (S) in ...end</td>
<td>Initialize a local signal (S)</td>
</tr>
<tr>
<td>EMIT (S [, {#data}reg])</td>
<td>emit (S) [((val))]</td>
<td>Emit (valued) signal (S)</td>
</tr>
<tr>
<td>SUSTAIN (S [, {#data}reg])</td>
<td>sustain (S) [((val))]</td>
<td>Sustain (valued) signal (S)</td>
</tr>
<tr>
<td>PRESENT (S, elseAddr)</td>
<td>present (S) then ...end</td>
<td>Jump to (elseAddr) if (S) is absent</td>
</tr>
<tr>
<td>NOTHING</td>
<td>nothing</td>
<td>Do nothing</td>
</tr>
<tr>
<td>HALT</td>
<td>halt</td>
<td>Halt the program</td>
</tr>
<tr>
<td>GOTO (addr)</td>
<td>loop ...end loop</td>
<td>Jump to (addr)</td>
</tr>
<tr>
<td>CALL (addr)</td>
<td>call (P)</td>
<td>call a procedure, and return from the procedure</td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling Concurrency

Execution status of a single thread

The status of the whole program, as managed by the Thread Block
Handling Concurrency

A thread has its

- thread id
- address range and independent program counter
- priority value
  - assigned when a thread is created
  - dynamically changed via PRI0 instruction
- status flags
  - ThreadEnable
  - ThreadActive
Handling Preemption

Watcher contains

Enable Watcher (EW)
- Watches the PC (Program Counter)
- Compares PC
- Preemption enabled?

Trigger Watcher (TW)
- Watches the Signal
- Counts down the counter (abortion)
- Preemption active?

% Esterel
abort
  weak abort
  p;
  when S2;
  q;
  when S1;

% KEP Assembler
ABORT S1,A1
WABORT S2,A0
  p
A0:  q
A1:
Watcher Refinement

Thread Watcher
- belongs to a thread directly
- can neither include concurrent threads nor other preemptions
- least powerful, but also cheapest

Local Watcher
- may include concurrent threads and also preemptions handled by a Thread Watcher
- cannot include another Local Watcher

Watcher
- may include concurrent threads and any preemptions
- most powerful, but also most expensive
Handling Exceptions

Exception

- has its address range
- sets an exitFlag
  - cleared when reaching the end of the trap scope
  - effects control at the join point
- can be overridden based on the corresponding trap scopes (address range)

```esterel
% Esterel
trap T1 in
  trap T2 in
  [ p;
    exit T1;
    ||
  q;
    exit T2; ];
end trap;
r;
end trap;

% KEP Assembler
T1S: T2S:
  PAR 1,A1,1
  PAR 1,A2,2
  PARE A3
A1: p
  EXIT T1,T1S
A2: q
  EXIT T2,T2E
A3: JOIN
T2E:r
T1E:
```
WCRT (Tick Length) Self-Monitoring

- OscClk: external clock; InstrClk: instructions; Tick: logical ticks
- Emitting special signal `_TICKLEN` configures Tick Manager with WCRT
- TickWarn pin indicates WCRT timing violation
Step 1: Construct Concurrent KEP Assembler Graph

module: Example

% module Example
OUTPUT 0

[L00,T0]  EMIT _TICKLEN,#12
[L01,T0]  SIGNAL A
[L02,T0]  SIGNAL R
[L03,T0]  PAR 2,A0,1
[L04,T0]  PAR 1,A1,2
[L05,T0]  PARE A2,2
[L06,T1]  A0: WABORTI A,A3
[L07,T1]  A4: EMIT R
[L08,T1]  PRIO 1
[L09,T1]  PRIO 2
[L10,T1]  PAUSE
[L11,T1]  GOTO A4
[L12,T1]  A3: EMIT 0
[L14,T2]  EMIT A
[L15,T0]  JOIN 0
[L16,T0]  HALT
Step 2: Compute Thread Priorities/\textit{ids}

- Compute priority for current tick at each node
- Compute priority for next tick at tick boundaries
Step 2: Compute Thread Priorities/\( ids \)

- Compute priority for current tick at each node
- Compute priority for next tick at tick boundaries
- Priority within tick must not increase
- Initialize tick boundaries with lowest priority, compute priorities backwards
Step 2: Compute Thread Priorities/\textit{ids}

- Compute priority for current tick at each node
- Compute priority for next tick at tick boundaries
- Priority within tick must not increase
- Initialize tick boundaries with lowest priority, compute priorities backwards
- Judicious traversal of CKAG allows to compute each priority just once
  - Facilitates correctness argument
  - Complexity linear in CKAG size
Step 3: Generate PAR/PRI0 Statements

- Enforce that a statement is always executed with same priority, irrespective of control flow
- Must consider priorities for current and for next tick
- Again linear complexity
CKAG Node Types

The CKAG distinguishes the following sets of nodes:

**D**: Delay nodes (octagons)
  - PAUSE, AWAIT, HALT, SUSTAIN

**F**: Fork nodes (triangles)
  - PAR/PARE

**T**: Transient nodes (rectangles/inverted triangles)
  - EMIT, PRESENT, etc. (rectangles)
  - JOIN nodes (inverted triangles)

**N**: Set of all nodes, \( N = D \cup F \cup T \)
The Concurrent KEP Assembler Graph (CKAG)

Define

- for each fork node $n$:
  - $n$.join: the JOIN statement corresponding to $n$,
  - $n$.sub: the transitive closure of nodes in threads generated by $n$.

- for abort nodes $n$ ([L|T][W]ABORT[I], SUSPEND[I]):
  - $n$.end: the end of the abort scope opened by $n$,
  - $n$.scope: the nodes within $n$’s abort scope.

- for all nodes $n$:
  - $n$.prio: the priority that the thread executing $n$ should be running with

- for $n \in D \cup F$,
  - $n$.prionext: the priority that the thread executing $n$ should be resumed with in the subsequent tick.
CKAG Dependency Types

Define dependencies

\( n_{\text{dep}_i} \): the dependency sinks with respect to \( n \) at the current tick (the *immediate dependencies*)

\( n_{\text{dep}_d} \): the dependency sinks with respect to \( n \) at the next tick (the *delayed dependencies*)

Induced by emissions of strong abort trigger signals and corresponding delay nodes within the abort scope
CKAG Successor Types

Define following types of successors for each $n$:

- $n.suc_c$: the control successors.
- $n.suc_w$: the weak abort successors
- $n.suc_s$: the strong abort successors
- $n.suc_f$: the flow successors

the set $n.suc_c \cup n.suc_w \cup n.suc_s$

For $n \in F$ we also define the following fork abort successors

- $n.suc_wf$: the weak fork abort successors
- $n.suc_sf$: the strong fork abort successors
Program Cycle

An Esterel program is considered cyclic iff the corresponding CKAG contains a path from a node to itself, where for all nodes $n$ and their successors along that path, $n'$ and $n''$, the following holds:

$$
n \in D \land n' \in n.suc_w
$$
$$\lor \quad n \in F \land n' \in n.suc_c \cup n.suc_{wf}
$$
$$\lor \quad n \in T \land n' \in n.suc_c \cup n.dep_i
$$
$$\lor \quad n \in T \land n' \in n.dep_d \land n'' \in n'.suc_c \cup n'.suc_s \cup n'.suc_{sf}.
$$
Constraints

A correct priority assignment must fulfill the following constraints, where \( m, n \) are arbitrary nodes in the CKAG

**Constraint (Dependencies)**

- \( \text{For } m \in n.\text{dep}_i: \ n.\text{prio} > m.\text{prio} \)
- \( \text{For } m \in n.\text{dep}_d: \ n.\text{prio} > m.\text{prionext} \)

**Constraint (Intra-Tick Priority)**

- \( \text{For } n \in D \text{ and } m \in n.\text{suc}_w, \text{ or } n \in F \text{ and } m \in n.\text{suc}_c \cup n.\text{suc}_wf, \text{ or } n \in T \text{ and } m \in n.\text{suc}_c: } \ n.\text{prio} \geq m.\text{prio} \)
Computing Thread Priorities

Constraint (Inter-Tick Priority for Delay Nodes)

- $\forall m \in n.suc_c \cup n.suc_s: \ n.prionext \geq m.prio$

Constraint (Inter-Tick Priority for Fork Nodes)

- $n.prionext \geq n.join.prio$
- $\forall m \in n.suc_{sf}: \ n.prionext \geq m.prio$
Computing Thread Priorities

module Edwards02:
input S, I;
output 0;

signal A,R in
  every S do
    await I;
    weak abort
    sustain R;
  when immediate A;
  emit 0;
  loop
    pause;
    present R then
      emit A;
      end present
  end loop
  end every
end signal
end module

module Edwards02-dism:
input S,I;
output 0;

signal A, R in
  abort
  loop
    pause
    end loop
  when S;
  loop
    abort
      [ abort
        loop
          pause
          end loop
        when I;
        weak abort
        loop
          emit R;
          pause
          end loop
        when immediate A;
        emit 0
      ]
      % cont...
  end loop
end signal
end module

% cont...

||

loop
  pause;
  pause;
  present R then
    emit A
    end present
  end loop
end loop
when S
end loop
end module
Optimized Priority Assignment

```plaintext
INPUT S,I
OUTPUT  O
[L00,T0]  EMIT  _TICKLEN,#20
[L01,T0]  SIGNAL  A
[L02,T0]  SIGNAL  R
[L03,T0]  AWAIT  S
[L04,T0]  A2:  LABORT  S,A3
[L05,T0]  PAR  1,A4,1
[L06,T0]  PAR  1,A5,2
[L07,T0]  PARE  A6,1
[L08,T1]  A4:  TABORT  I,A7
[L09,T1]  A8:  PRIO  3
[L10,T1]  PAUSE
[L11,T1]  PRIO  1
[L12,T1]  GOTO  A8
[L13,T1]  A7:  TWABORTI  A,A9
[L14,T1]  A10:  EMIT  R
[L15,T1]  PRIO  1
[L16,T1]  PRIO  3
[L17,T1]  PAUSE
[L18,T1]  GOTO  A10
[L19,T1]  A9:  EMIT  0
[L20,T2]  A5:A11:  PAUSE
[L21,T2]  PRIO  2
[L22,T2]  PAUSE
[L23,T2]  PRESENT  R,A12
[L24,T2]  EMIT  A
[L25,T2]  A12:  PRIO  1
[L26,T2]  GOTO  A11
[L27,T0]  A6:  JOIN
[L28,T0]  A3:  GOTO  A2
```

⇒

```plaintext
INPUT S,I
OUTPUT  O
[L00,T0]  EMIT  _TICKLEN,#20
[L01,T0]  SIGNAL  A
[L02,T0]  SIGNAL  R
[L03,T0]  AWAIT  S
[L04,T0]  A2:  LABORT  S,A3
[L05,T0]  PAR  3,A4,1
[L06,T0]  PAR  2,A5,2
[L07,T0]  PARE  A6,1
[L08,T1]  A4:  AWAIT  I
[L09,T1]  A7:  TWABORTI  A,A9
[L10,T1]  A10:  EMIT  R
[L11,T1]  PRIO  1
[L12,T1]  PRIO  3
[L13,T1]  PAUSE
[L14,T1]  GOTO  A10
[L15,T1]  A9:  EMIT  0
[L17,T2]  PAUSE
[L18,T2]  PRESENT  R,A12
[L19,T2]  EMIT  A
[L20,T2]  A12:  GOTO  A11
[L21,T0]  A6:  JOIN
[L22,T0]  A3:  GOTO  A2
```
Code Characteristics and Compilation Times

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Threads</th>
<th>Preemptions</th>
<th>CKAG</th>
<th>Preemption handled by</th>
<th>Compiling Time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cnt</td>
<td>Max Depth</td>
<td>Cnt</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DepthCon</td>
<td></td>
<td>Depth</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Esterel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEP3a (Unoptimized)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEP3a (optimized)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MicroBlaze</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: In the mca200, the watcher refinement reduces the hardware requirements from 4033 slices (if all preemptions were handled by general purpose Watchers) to 3265 slices (19% reduction).
## Worst-/Average-Case Reaction Times

<table>
<thead>
<tr>
<th>Module Name</th>
<th>MicroBlaze</th>
<th>KEP3a-Unoptimized</th>
<th>KEP3a-optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCRT</td>
<td>ACRT</td>
<td>WCRT Ratio to</td>
</tr>
<tr>
<td></td>
<td>V5 V7 CEC</td>
<td>V5 V7 CEC</td>
<td>best MB</td>
</tr>
<tr>
<td>abcdd</td>
<td>1559 954 1476</td>
<td>1464 828 1057</td>
<td>135 0.14</td>
</tr>
<tr>
<td>abcdef</td>
<td>2281 1462 1714</td>
<td>2155 1297 1491</td>
<td>201 0.14</td>
</tr>
<tr>
<td>eight_but</td>
<td>3001 1953 2259</td>
<td>2833 1730 1931</td>
<td>267 0.14</td>
</tr>
<tr>
<td>chan_prot</td>
<td>754 375 623</td>
<td>683 324 435</td>
<td>117 0.31</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>487 230 397</td>
<td>456 214 266</td>
<td>54 0.23</td>
</tr>
<tr>
<td>runner</td>
<td>566 289 657</td>
<td>512 277 419</td>
<td>36 0.12</td>
</tr>
<tr>
<td>example</td>
<td>467 169 439</td>
<td>404 153 228</td>
<td>42 0.25</td>
</tr>
<tr>
<td>ww_button</td>
<td>1185 578 979</td>
<td>1148 570 798</td>
<td>72 0.12</td>
</tr>
<tr>
<td>greycounter</td>
<td>1965 1013 2376</td>
<td>1851 928 1736</td>
<td>528 0.52</td>
</tr>
<tr>
<td>tcint</td>
<td>3580 1878 2350</td>
<td>3488 1797 2121</td>
<td>408 0.22</td>
</tr>
<tr>
<td>mca200</td>
<td>75488 29078 12497</td>
<td>73824 24056 11479</td>
<td>2862 0.23</td>
</tr>
</tbody>
</table>

The worst-/average-case reaction times, in clock cycles, for the KEP3a and MicroBlaze:

- **WCRT speedup**: typically > 4x
- **ACRT speedup**: typically > 5x
- **Optimizations yield further improvements**
Memory Usage

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel LOC</th>
<th>MicroBlaze Code+Data (byte)</th>
<th>KEP3a-Unopt. Code+Data (byte)</th>
<th>KEP3a-opt. Code (word)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V5 V7 CEC</td>
<td>V5 V7 CEC (best)</td>
<td>Code (word)</td>
<td>Code (word)</td>
</tr>
<tr>
<td>abcd</td>
<td>160</td>
<td>6680 7928 7212</td>
<td>168 1.05</td>
<td>756 0.11</td>
</tr>
<tr>
<td>abcddef</td>
<td>236</td>
<td>9352 9624 9220</td>
<td>252 1.07</td>
<td>1134 0.12</td>
</tr>
<tr>
<td>eight_but</td>
<td>312</td>
<td>12016 11276 11948</td>
<td>336 1.08</td>
<td>1512 0.13</td>
</tr>
<tr>
<td>chan_prot</td>
<td>42</td>
<td>3808 6204 3364</td>
<td>66 1.57</td>
<td>297 0.09</td>
</tr>
<tr>
<td>reactor controlled</td>
<td>27</td>
<td>2668 5504 2460</td>
<td>38 1.41</td>
<td>171 0.07</td>
</tr>
<tr>
<td>runner</td>
<td>31</td>
<td>3140 5940 2824</td>
<td>39 1.22</td>
<td>175 0.06</td>
</tr>
<tr>
<td>example</td>
<td>20</td>
<td>2480 5196 2344</td>
<td>31 1.55</td>
<td>139 0.06</td>
</tr>
<tr>
<td>ww_button</td>
<td>76</td>
<td>6112 7384 5980</td>
<td>129 1.7</td>
<td>580 0.10</td>
</tr>
<tr>
<td>greycounter</td>
<td>143</td>
<td>7612 7936 8688</td>
<td>347 2.43</td>
<td>1567 0.21</td>
</tr>
<tr>
<td>tcint</td>
<td>355</td>
<td>14860 11376 15340</td>
<td>437 1.23</td>
<td>1968 0.17</td>
</tr>
<tr>
<td>mca200</td>
<td>3090</td>
<td>104536 77112 52998</td>
<td>8650 2.79</td>
<td>39717 0.75</td>
</tr>
</tbody>
</table>

- **Unoptimized**: 83% avg reduction of memory usage (Code+RAM)
- **Optimized**: May yield further 5% to 30+% improvements
## Power Consumption

<table>
<thead>
<tr>
<th>Module Name</th>
<th>MicroBlaze (82mW@50MHz)</th>
<th>KEP3a² (mW)</th>
<th>Ratio (KEP to MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
<td>Peak</td>
<td>Idle</td>
</tr>
<tr>
<td>abcd</td>
<td>69</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>abcdef</td>
<td>74</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>eight_but</td>
<td>74</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>chan_prot</td>
<td>70</td>
<td>28</td>
<td>12</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>76</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>runner</td>
<td>78</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>example</td>
<td>77</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>ww_button</td>
<td>81</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>greycounter</td>
<td>78</td>
<td>44</td>
<td>33</td>
</tr>
<tr>
<td>tcint</td>
<td>80</td>
<td>18</td>
<td>10</td>
</tr>
</tbody>
</table>

- Peak energy usage reduction: 75% avg
- Idle (= no inputs) energy usage reduction: 86% avg

²Based on Xilinx 3S200-4ft256, requires an additional 37mW as quiescent power for the chip itself
Scalability

Synthesis results for Xilinx 3S1500-4fg-676

<table>
<thead>
<tr>
<th>Thread Count</th>
<th>Slices</th>
<th>Gates (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1295</td>
<td>295</td>
</tr>
<tr>
<td>10</td>
<td>1566</td>
<td>299</td>
</tr>
<tr>
<td>20</td>
<td>1871</td>
<td>311</td>
</tr>
<tr>
<td>40</td>
<td>2369</td>
<td>328</td>
</tr>
<tr>
<td>60</td>
<td>3235</td>
<td>346</td>
</tr>
<tr>
<td>80</td>
<td>4035</td>
<td>373</td>
</tr>
<tr>
<td>100</td>
<td>4569</td>
<td>389</td>
</tr>
<tr>
<td>120</td>
<td>5233</td>
<td>406</td>
</tr>
</tbody>
</table>

- 48 valued signals
  - up to 256 possible
- 2 Watchers, 8 Local Watchers
  - either up to 64 possible
- 1k (1024) instruction words
  - up to 64k possible
- 128 registers (in word)
  - up to 512 possible
- 16-bits (65536) max counter value
- Frequency is stable (around 60 MHz)

Note: In the mca200, the watcher refinement reduces the hardware requirements from 4033 slices (if all preemptions were handled by general purpose Watchers) to 3265 slices (19% reduction).

For comparison, a MicroBlaze implementation requires around 1k slices.

\(^3\)
## Analysis of Context Switches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abcd</td>
<td>16513</td>
<td>3787</td>
<td>1521</td>
<td>3082</td>
<td>1243</td>
</tr>
<tr>
<td>abcdef</td>
<td>29531</td>
<td>7246</td>
<td>3302</td>
<td>6043</td>
<td>2519</td>
</tr>
<tr>
<td>eight_but</td>
<td>39048</td>
<td>10073</td>
<td>5356</td>
<td>8292</td>
<td>3698</td>
</tr>
<tr>
<td>chan_prot</td>
<td>5119</td>
<td>1740</td>
<td>707</td>
<td>990</td>
<td>438</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>151</td>
<td>48</td>
<td>29</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>runner</td>
<td>5052</td>
<td>704</td>
<td>307</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>example</td>
<td>208</td>
<td>60</td>
<td>2</td>
<td>26</td>
<td>9</td>
</tr>
<tr>
<td>ww_button</td>
<td>292</td>
<td>156</td>
<td>92</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>greycounter</td>
<td>160052</td>
<td>34560</td>
<td>14043</td>
<td>26507</td>
<td>12725</td>
</tr>
<tr>
<td>tcint</td>
<td>80689</td>
<td>33610</td>
<td>16769</td>
<td>5116</td>
<td>2129</td>
</tr>
<tr>
<td>mca200</td>
<td>982417</td>
<td>256988</td>
<td>125055</td>
<td>242457</td>
<td>105258</td>
</tr>
</tbody>
</table>
## Edwards02: Esterel to KEP

```plaintext
module Edwards02:
  input S, I;
  output O;

  signal A, R in
  every S do
    await I;
    weak abort
    sustain R;
    when immediate A;
    emit O;
  end every

  loop
    p
  end loop

  await S;
  loop
    abort
    p;
    halt
    when S
  end loop

  sustain S
  loop
    emit S;
    pause;
  end loop

  loop
    p
  end loop

  A:
    p;
    goto A
```

### Code Characteristics and Compilation Times
- **Input:** S, I
- **Output:** O
- **Statement:**
  - [L00, T0] `EMIT _TICKLEN, #20`
  - [L01, T0] `SIGNAL A`
  - [L02, T0] `SIGNAL R`
  - [L03, T0] `AWAIT S`
  - [L04, T0] `A2: LABORT S, A3`
  - [L05, T0] `PAR 1, A4, 1`
  - [L06, T0] `PAR 1, A5, 2`
  - [L07, T0] `PARE A6, 1`
  - [L08, T1] `A4: TABORT I, A7`
  - [L09, T1] `A8: PRIO 3`
  - [L10, T1] `PAUSE`
  - [L11, T1] `PRIO 1`
  - [L12, T1] `GOTO A8`
  - [L13, T1] `A7: TWABORTI A, A9`
  - [L14, T1] `A10: EMIT R`
  - [L15, T1] `PRIO 1`
  - [L16, T1] `PRIO 3`
  - [L17, T1] `PAUSE`
  - [L18, T1] `GOTO A10`
  - [L19, T1] `A9: EMIT 0`
  - [L20, T2] `A5: A11: PAUSE`
  - [L21, T2] `PRIO 2`
  - [L22, T2] `PAUSE`
  - [L23, T2] `PRESENT R, A12`
  - [L24, T2] `EMIT A`
  - [L25, T2] `A12: PRIO 1`
  - [L26, T2] `GOTO A11`
  - [L27, T0] `A6: JOIN`
  - [L28, T0] `A3: GOTO A2`
Edwards02: a Possible Execution Trace

module Edwards02:
  input S, I;
  output O;

  signal A, R in
  every S do
    await I;
    weak abort
    sustain R;
    when immediate A;
    emit O;
  end
end signal

end module

await S;
loop
  abort
  p;
  halt
  when S
end loop

loop
  emit S;
  pause;
end loop

sustain S

loop
  p;
end every
end loop
end signal

loop
  p;
end loop

A:
p; goto A

Tick
S I
R R
R A
O

Xin Li, Marian Boldt, Reinhard v. Hanxleden
module Edwards02:
input S, I;
output 0;

signal A, R in
every S do
  await I;
  weak abort
  sustain R;
  when immediate A;
  emit 0;
end every loop
pause;
pause;
present R then
  emit A;
end present
end loop
end signal
end module

INPUT S, I
OUTPUT 0

[L00, T0] EMIT _TICKLEN, #20
[L01, T0] SIGNAL A
[L02, T0] SIGNAL R
[L03, T0] AWAIT S
[L04, T0] A2: LABORT S, A3
[L05, T0] PAR 1, A4, 1
[L06, T0] PAR 1, A5, 2
[L07, T0] PARE A6, 1
[L08, T1] A4: TABORT I, A7
[L09, T0] A8: PRI 0 3
[L10, T0] PAUSE
[L11, T1] PRI 0 1
[L12, T1] GOTO A8
[L13, T1] A7: TWABORTI A, A9
[L14, T1] A10: EMIT R
[L15, T1] PRI 0 1
[L16, T1] PRI 0 3
[L17, T1] PAUSE
[L18, T1] GOTO A10
[L19, T1] A9: EMIT 0
[L20, T2] A5: A11: PAUSE
[L21, T2] PRI 0 2
[L22, T2] PAUSE
[L23, T2] PRESENT R, A12
[L24, T2] EMIT A
[L25, T2] A12: PRI 0 1
[L26, T2] GOTO A11
[L27, T0] A6: JOIN
[L28, T0] A3: GOTO A2

- Tick 1 -
  ! reset;
% In: % Out:
[L01, T0] [L02, T0] [L03, T0]

- Tick 2 -
% In: S % Out:
[L03, T0] [L04, T0] [L05, T0]
[L06, T0] [L07, T0]
[L20, T2] [L08, T1]
[L09, T1] [L10, T1]
[L27, T0]

- Tick 3 -
% In: I % Out: R
[L10, T1] [L13, T1]
[L14, T1] [L15, T1]
[L20, T2] [L21, T2] [L22, T2]
[L16, T1] [L17, T1] [L27, T0]

- Tick 4 -
% In: % Out: A R O
[L17, T1] [L18, T1]
[L14, T1] [L15, T1]
[L22, T2] [L23, T2] [L24, T2]
[L25, T2] [L26, T2] [L20, T2]
[L16, T1] [L17, T1] [L19, T1]
[L27, T0]
Multi-processing vs. Multi-threading

Multi-processing (EMPEROR/RePIC)

- Esterel thread $\approx$ one independent RePIC processor
- Thread Control Unit handles the synchronization and communication
- Three-valued signal representation
- `sync` command to synchronize threads

Multi-threading (KEP)

- Esterel thread $\approx$ several registers
- priority-based scheduler
- `PRIO` command to synchronize threads
## Comparison of Synthesis Options

<table>
<thead>
<tr>
<th>Cost</th>
<th>HW</th>
<th>SW</th>
<th>Co-design</th>
<th>Reactive Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Area</td>
<td>++/−</td>
<td>+</td>
<td>+</td>
<td>−/−</td>
</tr>
<tr>
<td>Memory</td>
<td>++</td>
<td>−</td>
<td>−/−</td>
<td>+</td>
</tr>
<tr>
<td>Power Usage</td>
<td>++</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>Appl. Design Cycle</td>
<td>−</td>
<td>++</td>
<td>+/−</td>
<td>++</td>
</tr>
<tr>
<td>Speed</td>
<td>++</td>
<td>−</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Flexibility</td>
<td>−</td>
<td>++</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Scalability</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>−</td>
</tr>
</tbody>
</table>

Multi-processing vs. Multi-threading:

- Speed: +++, −, +
- Flexibility: −, ++, −, +
- Scalability: +, ++, +
- Logic Area: ++/−, +, +
- Memory: ++, −, −
- Power Usage: ++, −, −
- Appl. Design Cycle: −, ++, +/−
Scenario I: DSP + Reactive Processor

- DSP
- Reactive Processor
- IPs
- Global Memory
- Communication Backplane
Scenario II: DSP + HW Block + Reactive Processor
Scenario III: HW Block + Reactive Processor

- HW blk
- Global Memory
- Communication Backplane
- Reactive Processor
- IPs
Possible Co-Design Development Flow

Reactive processing . . .

- permits a simple mapping strategy
- allows optimizations on high-level
- can meet stricter constraints than classical architectures
- permits a better tradeoff between all cost factors

Application Description
(Esterel + e.g. Lustre/Simulink)

Co-simulation/verif.

Mapping

Reactive Processor Synthesis


System Constraints (e.g. WCET, area, etc.)

HW Synthesis


HW Block Impl.

Implementation of App.
Possible Co-Design Development Flow

Reactive processing . . .

- permits a simple mapping strategy
- allows optimizations on high-level
- can meet stricter constraints than classical architectures
- permits a better tradeoff between all cost factors

Thanks/Comments?