Light-Weight, Predictable Reactive Processing—The Kiel Esterel Processor

Xin Li Reinhard von Hanxleden

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

University Booth at DATE’07
Nice, 19 April 2007
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
  Instruction Set Architecture
  Processor Architecture
  WCRT Self-Monitoring
  KEP Evaluation Platform

Experimental Results

Summary and Outlook
The KEP Instruction Set

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

*This redundancy improves space/time efficiency*
The KEP Instruction Set

- Includes all Esterel 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;
```
The KEP Instruction Set

- Includes all Esterel 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;
```
The KEP Instruction Set

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

*This redundancy improves space/time efficiency*

```plaintext
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 Esterel 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;
```

- 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
```

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

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

This redundancy improves space/time efficiency

\[
\begin{align*}
\text{TOS:} & \quad \% \text{trap } T \text{ in} \\
\text{A0:} & \quad \% \text{loop} \\
& \quad \text{PAUSE} \quad \% \text{pause} \\
& \quad \text{PRESENT S, A1} \quad \% \text{present } S \text{ then} \\
& \quad \text{EXIT TOE, TOS} \quad \% \text{exit } T \\
\text{A1:} & \quad \% \text{end present} \\
& \quad \text{GOTO A0} \quad \% \text{end loop} \\
\text{TOE:} & \quad \% \text{end trap} \\
\end{align*}
\]

\[
\begin{align*}
\text{AWAIT S} & \quad \% \text{await } S
\end{align*}
\]

- Refined ISA to reduce HW usage
  - Example: abort can translate to
    - \text{ABORT} in the most general case
    - \text{LABORT} if no other [L]ABORTS are included in abort scope
    - \text{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, ...
The Architecture of the Reactive Core
WCRT Self-Monitoring

- Worst Case Reaction Time: max. instruction cycles per logical tick
- Synchronicity + static schedule + KEP ISA $\rightarrow$ WCRT analysis practical
- OscClk: external clock; InstrClk: instructions; Tick: logical ticks
- Emitting special signal _TICKLEN configures Tick Manager with WCRT
- TickWarn pin indicates WCRT timing violation

% KEP Assembler
% module OVERRUN
INPUT D
OUTPUT A,B,C
EMIT _TICKLEN, #3
EMIT A
EMIT B
PAUSE
EMIT A
EMIT B
EMIT C
AWAIT D

Running on a KEP2 implemented on a Memec V2MB1000 Development Board at a rate of $T_{osc} = 41.67\ ns$ (24 MHz), the waveform was recorded by an Agilent 1683A Logic Analyzer. In OVERRUN, the first EMIT statement sets TICKLEN to three; in other words, the module claims that $V_{ticklen}$, the maximal number of instructions executed within a tick, is at most three. If TICKLEN is larger than $V_{ticklen}$, it means that the ticks last longer than is necessary to finish tick computations before the next tick starts; if TICKLEN is smaller than $V_{ticklen}$, this means that we run the risk of timing violations. Setting TICKLEN to some value, in this case three, activates the Tick Manager, which from then on will on the one hand ensure that ticks that complete in less than three instructions will be padded until they are three instruction cycles long, and on the other hand will signal a timing violation if a tick is not completed within three instructions. In the example, the first logical tick lasts three instruction cycles. In the second tick, the controller has to execute five instructions until the AWAIT statement is executed. Hence, the TickWarn signal will be set high when the fourth instruction cycle arrives to indicate the tick length timing violation.

The goal of the WCRT analysis presented here is to automatically deduce a value for TICKLEN that is just large enough to never induce a timing violation; ideally, we achieve $TICKLEN = V_{ticklen}$.
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
Overview

Introduction

The Kiel Esterel Processor

Experimental Results
  Performance
  Scalability

Summary and Outlook
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>abcddef</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></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abcd</td>
<td>160</td>
<td>6680 7928 7212</td>
<td>168 1.05 756 0.11</td>
<td>164 0.93</td>
</tr>
<tr>
<td>abcdef</td>
<td>236</td>
<td>9352 9624 9220</td>
<td>252 1.07 1134 0.12</td>
<td>244 0.94</td>
</tr>
<tr>
<td>eight_but</td>
<td>312</td>
<td>12016 11276 11948</td>
<td>336 1.08 1512 0.13</td>
<td>324 0.94</td>
</tr>
<tr>
<td>chan_prot</td>
<td>42</td>
<td>3808 6204 3364</td>
<td>66 1.57 297 0.09</td>
<td>62 0.94</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>27</td>
<td>2668 5504 2460</td>
<td>38 1.41 171 0.07</td>
<td>34 0.89</td>
</tr>
<tr>
<td>runner</td>
<td>31</td>
<td>3140 5940 2824</td>
<td>39 1.22 175 0.06</td>
<td>27 0.69</td>
</tr>
<tr>
<td>example</td>
<td>20</td>
<td>2480 5196 2344</td>
<td>31 1.55 139 0.06</td>
<td>28 0.94</td>
</tr>
<tr>
<td>ww_button</td>
<td>76</td>
<td>6112 7384 5980</td>
<td>129 1.7 580 0.10</td>
<td>95 0.74</td>
</tr>
<tr>
<td>greycounter</td>
<td>143</td>
<td>7612 7936 8688</td>
<td>347 2.43 1567 0.21</td>
<td>343 1</td>
</tr>
<tr>
<td>tcint</td>
<td>355</td>
<td>14860 11376 15340</td>
<td>437 1.23 1968 0.17</td>
<td>379 0.87</td>
</tr>
<tr>
<td>mca200</td>
<td>3090</td>
<td>104536 77112 52998</td>
<td>8650 2.79 39717 0.75</td>
<td>8650 1</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**

<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
Summary Reactive Processors

Processor supports reactive control flow directly, at hardware level

► “Watchers” monitor preemption signals
   \textit{No need for polling, interrupts}

► Support for concurrency
   \textit{Multi-threading or multi-processing}

► Synchronous model of computation
   \textit{Perfectly deterministic, predictable timing}

Not covered here:

► Compilation (challenge: obey signal dependencies)
Summary Reactive Processors

Processor supports reactive control flow directly, at hardware level

- “Watchers” monitor preemption signals
  \textit{No need for polling, interrupts}
- Support for concurrency
  \textit{Multi-threading or multi-processing}
- Synchronous model of computation
  \textit{Perfectly deterministic, predictable timing}

Not covered here:

- Compilation (challenge: obey signal dependencies)
- WCRT analysis (initializes Tick Manager)
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*

Not covered here:

- Compilation (challenge: obey signal dependencies)

- WCRT analysis (initializes Tick Manager)

- HW/SW co-synthesis (combinational logic speeds up expression computation)
Outlook

- Improve priority assignments
- 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
- 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

www.informatik.uni-kiel.de/rtsys/kep/
Outlook

- Improve priority assignments
- 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

www.informatik.uni-kiel.de/rtsys/kep/

Thanks!

Questions/Comments?
Appendix

KEP3a Instruction Set + Architecture
   Esterel and the KEP Instruction Set Architecture
   Handling Concurrency
   Handling Preemption

The Compiler
   Compiler Overview
   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
   Related Work/Contributions
   Multi-processing vs. Multi-threading
   Comparison of Synthesis Options
   Application Scenarios
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

LogicalTicks

- 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

```e
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 0
  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 0;
  loop
    abort
    [ await A
      ||
      await B ];
    emit 0
    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

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

<table>
<thead>
<tr>
<th>Tick</th>
<th>A</th>
<th>B</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

Tick

<table>
<thead>
<tr>
<th>Tick</th>
<th>A</th>
<th>B</th>
<th>R</th>
<th>R</th>
<th>O</th>
</tr>
</thead>
</table>

A | B | R | R | O |
Candidates for the Instruction Set

Esterel kernel statements

- ||
- suspend ... 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 ... when $S$
- trap $T$ in ... exit $T$ ... end trap
- pause
- signal $S$ in ... end
- emit $S$
- present $S$ then ... end
- nothing
- loop ... end loop
- ;

Derived statements
- [weak] abort ... when $S$
- await $S$
- ...
### 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>[ ]</td>
<td>Fork and join.</td>
</tr>
<tr>
<td>PARE endAddr</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>JOIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIO Prio</td>
<td></td>
<td>Set the priority of the current thread</td>
</tr>
<tr>
<td>[W]ABORT [n,] S, endAddr</td>
<td>[weak] abort ... when [n] S</td>
<td>S can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. S: signal status (present/absent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. PRE(S): previous status of signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. TICK: always present</td>
</tr>
<tr>
<td></td>
<td>n can be one of the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. #data: immediate data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. reg: register contents</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. ?S: value of a signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. PRE(?S): previous value of a signal</td>
<td></td>
</tr>
<tr>
<td>[W]ABORTI S, endAddr</td>
<td>[weak] abort ... when immediate S</td>
<td></td>
</tr>
<tr>
<td>SUSPEND[I] S, endAddr</td>
<td>suspend ... when [immediate] S</td>
<td></td>
</tr>
<tr>
<td>EXIT TrapEnd[,TrapStart]</td>
<td>trap T in exit T end trap</td>
<td>Exit from a trap, TrapStart and TrapEnd specify trap scope. Unlike GOTO, check for concurrent EXITs and terminate enclosing</td>
</tr>
</tbody>
</table>
### Instruction Set Summary 2/2

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PAUSE</strong></td>
<td>pause</td>
<td>Wait for a signal. AWAIT TICK is equivalent to PAUSE</td>
</tr>
<tr>
<td><strong>AWAIT [n,] S</strong></td>
<td>await [n] S</td>
<td></td>
</tr>
<tr>
<td><strong>AWAIT[I] S</strong></td>
<td>await [immediate] S</td>
<td></td>
</tr>
<tr>
<td><strong>CAWAITS</strong></td>
<td>await case [immediate] S do end</td>
<td>wait for several signals in parallel</td>
</tr>
<tr>
<td><strong>CAWAIT[1] S, addr</strong></td>
<td>await S do</td>
<td></td>
</tr>
<tr>
<td><strong>CAWAIT</strong></td>
<td>await S do end</td>
<td></td>
</tr>
<tr>
<td><strong>SIGNAL S</strong></td>
<td>signal S in ...end</td>
<td>Initialize a local signal S</td>
</tr>
<tr>
<td>**EMIT S [, {#data</td>
<td>reg}]**</td>
<td>emit S [(val)]</td>
</tr>
<tr>
<td>**SUSTAIN S [, {#data</td>
<td>reg}]**</td>
<td>sustain S [(val)]</td>
</tr>
<tr>
<td><strong>PRESENT S, elseAddr</strong></td>
<td>present S then ...end</td>
<td>Jump to elseAddr if S is absent</td>
</tr>
<tr>
<td><strong>NOTHING</strong></td>
<td>nothing</td>
<td>Do nothing</td>
</tr>
<tr>
<td><strong>HALT</strong></td>
<td>halt</td>
<td>Halt the program</td>
</tr>
<tr>
<td><strong>GOTO addr</strong></td>
<td>loop ...end loop</td>
<td>Jump to addr</td>
</tr>
<tr>
<td><strong>CALL addr</strong></td>
<td>call P</td>
<td>call a procedure, and return from the procedure</td>
</tr>
<tr>
<td><strong>RETURN</strong></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

% Esterel
[
  p
  ||
  q
];

% KEP Assembler
   PAR 1,A0,1
   PAR 1,A1,2
   PARE A2
A0:  p
A1:  q
A2:  JOIN
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)
The Compilation Challenge: Thread Dependencies

```eclipse
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
```

Diagram:
```
Tick
```
```
R     R  
A     O
```
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/\textit{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/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
Example Compilation

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

```plaintext
% 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
```

- 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 D
  T1: L10, L11, L07, L08
  T2: L13, L14
  T1: L09, L10, L12
  T0: L15, L16
- Tick 3 -
  % In:
  % Out:
  T0: L16
Step 1: Construct Concurrent KEP Assembler Graph

% 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 O
[L14,T2] EMIT A
[L15,T1] A2: JOIN 0
[L16,T0] HALT
Step 2: Compute Thread Priorities/\( ids \)

- 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

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

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

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:

\[
\begin{align*}
\forall n &\in D \land n' \in n.\text{suc}_w \\
\lor n &\in F \land n' \in n.\text{suc}_c \cup n.\text{suc}_w \\
\lor n &\in T \land n' \in n.\text{suc}_c \cup n.\text{dep}_i \\
\lor n &\in T \land n' \in n.\text{dep}_d \land n'' \in n'.\text{suc}_c \cup n'.\text{suc}_s \cup n'.\text{suc}_sf.
\end{align*}
\]
Constraints

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

**Constraint (Dependencies)**

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

**Constraint (Intra-Tick Priority)**

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

Constraint (Inter-Tick Priority for Delay Nodes)

- For all $m \in n.\text{suc}_c \cup n.\text{suc}_s$: $n.\text{prionext} \geq m.\text{prio}$

Constraint (Inter-Tick Priority for Fork Nodes)

- $n.\text{prionext} \geq n.\text{join}.\text{prio}$
- For all $m \in n.\text{suc}_f$: $n.\text{prionext} \geq m.\text{prio}$
Computing Thread Priorities

module Edwards02:
  input S, I;
  output O;

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 loop
end every
end module

module Edwards02-dism:
  input S, I;
  output O;
  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...
    ||
  loop
    pause;
    when I;
    weak abort
    loop
      emit R;
      pause
    end loop
    when S
  end loop
end signal
end module

% cont...

||
  loop
  pause;
  when S
  end loop
% cont...
KEP3a Instruction Set + Architecture
The Compiler
Further Measurements
Summary

Compiler Overview
The Concurrent KEP Assembler Graph
Cyclicity, Constraints
## Optimized Priority Assignment

<table>
<thead>
<tr>
<th>Line</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[L00,T0]</td>
<td><code>EMIT _TICKLEN,#20</code></td>
<td></td>
</tr>
<tr>
<td>[L01,T0]</td>
<td><code>SIGNAL A</code></td>
<td></td>
</tr>
<tr>
<td>[L02,T0]</td>
<td><code>SIGNAL R</code></td>
<td></td>
</tr>
<tr>
<td>[L03,T0]</td>
<td><code>AWAIT S</code></td>
<td></td>
</tr>
<tr>
<td>[L04,T0]</td>
<td><code>A2: LABORT S,A3</code></td>
<td></td>
</tr>
<tr>
<td>[L05,T0]</td>
<td><code>PAR 1,A4,1</code></td>
<td></td>
</tr>
<tr>
<td>[L06,T0]</td>
<td><code>PAR 1,A5,2</code></td>
<td></td>
</tr>
<tr>
<td>[L07,T0]</td>
<td><code>PARE A6,1</code></td>
<td></td>
</tr>
<tr>
<td>[L08,T1]</td>
<td><code>A4: TABORT I,A7</code></td>
<td></td>
</tr>
<tr>
<td>[L09,T1]</td>
<td><code>A8: PRIO 3</code></td>
<td></td>
</tr>
<tr>
<td>[L10,T1]</td>
<td><code>PAUSE</code></td>
<td></td>
</tr>
<tr>
<td>[L11,T1]</td>
<td><code>PRIO 1</code></td>
<td></td>
</tr>
<tr>
<td>[L12,T1]</td>
<td><code>GOTO A8</code></td>
<td></td>
</tr>
<tr>
<td>[L13,T1]</td>
<td><code>A7: TWABORTI A,A9</code></td>
<td></td>
</tr>
<tr>
<td>[L14,T1]</td>
<td><code>A10: EMIT R</code></td>
<td></td>
</tr>
<tr>
<td>[L15,T1]</td>
<td><code>PRIO 1</code></td>
<td></td>
</tr>
<tr>
<td>[L16,T1]</td>
<td><code>PRIO 3</code></td>
<td></td>
</tr>
<tr>
<td>[L17,T1]</td>
<td><code>PAUSE</code></td>
<td></td>
</tr>
<tr>
<td>[L18,T1]</td>
<td><code>GOTO A10</code></td>
<td></td>
</tr>
<tr>
<td>[L19,T1]</td>
<td><code>A9: EMIT 0</code></td>
<td></td>
</tr>
<tr>
<td>[L20,T2]</td>
<td><code>A5:A11: PAUSE</code></td>
<td></td>
</tr>
<tr>
<td>[L21,T2]</td>
<td><code>PRIO 2</code></td>
<td></td>
</tr>
<tr>
<td>[L22,T2]</td>
<td><code>PAUSE</code></td>
<td></td>
</tr>
<tr>
<td>[L23,T2]</td>
<td><code>PRESENT R,A12</code></td>
<td></td>
</tr>
<tr>
<td>[L24,T2]</td>
<td><code>EMIT A</code></td>
<td></td>
</tr>
<tr>
<td>[L25,T2]</td>
<td><code>A12: PRIO 1</code></td>
<td></td>
</tr>
<tr>
<td>[L26,T2]</td>
<td><code>GOTO A11</code></td>
<td></td>
</tr>
<tr>
<td>[L27,T0]</td>
<td><code>A6: JOIN</code></td>
<td></td>
</tr>
<tr>
<td>[L28,T0]</td>
<td><code>A3: GOTO A2</code></td>
<td></td>
</tr>
</tbody>
</table>
### Code Characteristics and Compilation Times

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel</th>
<th>KEP3a (Unoptimized)</th>
<th>KEP3a (Optimized)</th>
<th>MicroBlaze</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Threads</td>
<td>Preemptions</td>
<td>CKAG Preemption handled by Compiling</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cnt Max</td>
<td>Depth Conc</td>
<td>Cnt Max</td>
<td>Dep Max</td>
<td>Prio</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abc</td>
<td>4 2 4</td>
<td>20 2</td>
<td>211 36</td>
<td>3 30</td>
<td>0 4 3</td>
</tr>
<tr>
<td>abcdef</td>
<td>6 2 6</td>
<td>30 2</td>
<td>313 90</td>
<td>3 48</td>
<td>0 6 5</td>
</tr>
<tr>
<td>eight</td>
<td>8 8</td>
<td>40 4</td>
<td>415 168</td>
<td>3 66</td>
<td>0 8 7</td>
</tr>
<tr>
<td>chan_prot</td>
<td>5 3 4</td>
<td>6 1</td>
<td>80 4 2</td>
<td>2 10</td>
<td>0 0</td>
</tr>
<tr>
<td>reactor</td>
<td>3 2 3</td>
<td>5 1</td>
<td>51 5 1</td>
<td>0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>runner</td>
<td>2 2 2</td>
<td>9 3</td>
<td>61 0 1</td>
<td>0</td>
<td>3 2</td>
</tr>
<tr>
<td>example</td>
<td>2 2 2</td>
<td>4 2</td>
<td>36 2 3</td>
<td>6</td>
<td>0 1</td>
</tr>
<tr>
<td>ww_button</td>
<td>13 3 4</td>
<td>27 2</td>
<td>194 0 1</td>
<td>0</td>
<td>0 5</td>
</tr>
<tr>
<td>greycounter</td>
<td>17 3 13</td>
<td>19 2</td>
<td>414 53</td>
<td>6 58</td>
<td>0 4</td>
</tr>
<tr>
<td>tcint</td>
<td>39 5 17</td>
<td>18 2</td>
<td>583 65</td>
<td>3 20</td>
<td>0 1</td>
</tr>
<tr>
<td>mca200</td>
<td>59 5 49</td>
<td>64 4</td>
<td>11219 129 11 190</td>
<td>2 14 48</td>
<td>11.25</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).
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**

<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>ACRT V5</td>
<td>WCRT Ratio to</td>
</tr>
<tr>
<td></td>
<td>V7</td>
<td>CEC</td>
<td>best MB</td>
</tr>
<tr>
<td>abcd</td>
<td>1559</td>
<td>1464</td>
<td>135 0.14</td>
</tr>
<tr>
<td>abcdef</td>
<td>2281</td>
<td>2155</td>
<td>201 0.14</td>
</tr>
<tr>
<td>eight_but</td>
<td>3001</td>
<td>2833</td>
<td>267 0.14</td>
</tr>
<tr>
<td>chan_prot</td>
<td>754</td>
<td>683</td>
<td>117 0.31</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>487</td>
<td>456</td>
<td>54 0.23</td>
</tr>
<tr>
<td>runner</td>
<td>566</td>
<td>512</td>
<td>36 0.12</td>
</tr>
<tr>
<td>example</td>
<td>467</td>
<td>404</td>
<td>42 0.25</td>
</tr>
<tr>
<td>ww_button</td>
<td>1185</td>
<td>1148</td>
<td>72 0.12</td>
</tr>
<tr>
<td>greycounter</td>
<td>1965</td>
<td>1851</td>
<td>528 0.52</td>
</tr>
<tr>
<td>tcint</td>
<td>3580</td>
<td>3488</td>
<td>408 0.22</td>
</tr>
<tr>
<td>mca200</td>
<td>75488</td>
<td>73824</td>
<td>2862 0.23</td>
</tr>
</tbody>
</table>
## Memory Usage

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel LOC</th>
<th>MicroBlaze Code+Data (byte) V5</th>
<th>V7</th>
<th>CEC</th>
<th>KEP3a-Unopt. Code (word) abs. rel.</th>
<th>KEP3a-Unopt. Code+Data (byte) abs. rel.</th>
<th>KEP3a-opt. Code (word) abs. rel.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>V5</td>
<td>V7</td>
<td>CEC</td>
<td>best</td>
<td>best</td>
<td>best</td>
</tr>
<tr>
<td>abcd</td>
<td>160</td>
<td>6680</td>
<td>7928</td>
<td>7212</td>
<td>168.1.05</td>
<td>756.0.11</td>
<td>164.0.93</td>
</tr>
<tr>
<td>abcdef</td>
<td>236</td>
<td>9352</td>
<td>9624</td>
<td>9220</td>
<td>252.1.07</td>
<td>1134.0.12</td>
<td>244.0.94</td>
</tr>
<tr>
<td>eight but</td>
<td>312</td>
<td>12016</td>
<td>11276</td>
<td>11948</td>
<td>336.1.08</td>
<td>1512.0.13</td>
<td>324.0.94</td>
</tr>
<tr>
<td>chan_prot</td>
<td>42</td>
<td>3808</td>
<td>6204</td>
<td>3364</td>
<td>66.1.57</td>
<td>297.0.09</td>
<td>62.0.94</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>27</td>
<td>2668</td>
<td>5504</td>
<td>2460</td>
<td>38.1.41</td>
<td>171.0.07</td>
<td>34.0.89</td>
</tr>
<tr>
<td>runner</td>
<td>31</td>
<td>3140</td>
<td>5940</td>
<td>2824</td>
<td>39.1.22</td>
<td>175.0.06</td>
<td>27.0.69</td>
</tr>
<tr>
<td>example</td>
<td>20</td>
<td>2480</td>
<td>5196</td>
<td>2344</td>
<td>31.1.55</td>
<td>139.0.06</td>
<td>28.0.94</td>
</tr>
<tr>
<td>ww_button</td>
<td>76</td>
<td>6112</td>
<td>7384</td>
<td>5980</td>
<td>129.1.7</td>
<td>580.0.10</td>
<td>95.0.74</td>
</tr>
<tr>
<td>greycounter</td>
<td>143</td>
<td>7612</td>
<td>7936</td>
<td>8688</td>
<td>347.2.43</td>
<td>1567.0.21</td>
<td>343.1</td>
</tr>
<tr>
<td>tcint</td>
<td>355</td>
<td>14860</td>
<td>11376</td>
<td>15340</td>
<td>437.1.23</td>
<td>1968.0.17</td>
<td>379.0.87</td>
</tr>
<tr>
<td>mca200</td>
<td>3090</td>
<td>104536</td>
<td>77112</td>
<td>52998</td>
<td>8650.2.79</td>
<td>39717.0.75</td>
<td>8650.1</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>KEP3a $^2$ (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

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

Synthesis results for Xilinx 3S1500-4fg-676³

<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).

³For comparison, a MicroBlaze implementation requires around 1k slices and 309k gates; a two threads EMPEROR platform requires around 2k slices.

Xin Li, Reinhard von Hanxleden
The Kiel Esterel Processor
Slide 51
## Analysis of Context Switches

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Instr's total</th>
<th>CSs at same priority</th>
<th>CSs due to PRI0</th>
<th>Module Name</th>
<th>Instr's total</th>
<th>CSs at same priority</th>
<th>CSs due to PRI0</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcde</td>
<td>16513</td>
<td>3787 4.36</td>
<td>1521 0.40</td>
<td>3082 0.19</td>
<td>1243 0.33</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>abcd</td>
<td>29531</td>
<td>7246 4.08</td>
<td>3302 0.46</td>
<td>6043 0.20</td>
<td>2519 0.35</td>
<td>0.42</td>
<td></td>
</tr>
<tr>
<td>eight_but</td>
<td>39048</td>
<td>10073 3.88</td>
<td>5356 0.53</td>
<td>8292 0.21</td>
<td>3698 0.37</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>chan_prot</td>
<td>5119</td>
<td>1740 2.94</td>
<td>707 0.41</td>
<td>990 0.19</td>
<td>438 0.25</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>151</td>
<td>48 3.15</td>
<td>29 0.60</td>
<td>0 0</td>
<td>0 0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>runner</td>
<td>5052</td>
<td>704 7.18</td>
<td>307 0.44</td>
<td>0 0</td>
<td>0 0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>208</td>
<td>60 3.47</td>
<td>2 0.30</td>
<td>26 0.13</td>
<td>9 0.15</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>ww_button</td>
<td>292</td>
<td>156 1.87</td>
<td>92 0.59</td>
<td>0 0</td>
<td>0 0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>greycounter</td>
<td>160052</td>
<td>34560 4.63</td>
<td>14043 0.41</td>
<td>26507 0.17</td>
<td>12725 0.37</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>tcint</td>
<td>80689</td>
<td>33610 2.4</td>
<td>16769 0.50</td>
<td>5116 0.06</td>
<td>2129 0.06</td>
<td>0.42</td>
<td></td>
</tr>
<tr>
<td>mca200</td>
<td>982417</td>
<td>256988 3.82</td>
<td>125055 0.49</td>
<td>242457 0.25</td>
<td>105258 0.41</td>
<td>0.43</td>
<td></td>
</tr>
</tbody>
</table>
Edwards02: Esterel to KEP

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

  signal A,R in
    every S do
      p
      end
    end

  loop
    abort
      p;
      halt
      when S
    end loop

  sustain S
    emit S;
    pause;
    end loop
  end

end module
```

```
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
```
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 every
end module

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

```
loop
  emit S;
  pause;
end loop
```

```
sustain S
```

```
loop
  p;
end loop
```

```
A:
  p;
goto A
```

Tick
---
S I R R A O
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;
pause;
present R then emit A;
end present
end loop
end every
end signal
end module

- Tick 1 -
! reset;
% In:
% Out:
[L00,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] [L16,T1] [L17,T1] [L18,T1]
[L19,T1] [L20,T2] [L21,T2] [L22,T2] [L27,T0]
- Tick 4 -
% In: A R O
[L17,T1] [L18,T1]
[L14,T1] [L15,T1] [L22,T2] [L23,T2] [L24,T2]
[L25,T2] [L26,T2] [L27,T0] [L28,T0] A3: GOTO A2
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
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
- `PRI0` command to synchronize threads
### Comparison of Synthesis Options

<table>
<thead>
<tr>
<th></th>
<th>HW</th>
<th>SW</th>
<th>Co-design</th>
<th>Reactive Processor</th>
<th>Multi-processing</th>
<th>Multi-threading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>++</td>
<td>--</td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Flexibility</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td></td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>Scalability</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td></td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Logic Area</td>
<td>++/-</td>
<td>+</td>
<td>+</td>
<td></td>
<td>--</td>
<td>+/-</td>
</tr>
<tr>
<td>Memory</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td></td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Power Usage</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td></td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Appl. Design Cycle</td>
<td>--</td>
<td>++</td>
<td>+/-</td>
<td></td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>
Scenario I: DSP + Reactive Processor

- DSP
- Reactive Processor
- IPs
- Communication Backplane
- Global Memory
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
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?