A Concurrent Reactive Esterel Processor Based on Multi-Threading

Xin Li, Reinhard v. Hanxleden

Christian-Albrechts Universität Kiel
Department of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
www.informatik.uni-kiel.de/inf/von-Hanxleden/

SAC 2006, Dijon, France
Overview

Introduction

- Esterel for Reactive Systems
- Esterel Synthesis Options
- Multi-processing vs. Multi-threading
Overview

Introduction
- Esterel for Reactive Systems
- Esterel Synthesis Options
- Multi-processing vs. Multi-threading

The Kiel Esterel Processor
- Architecture Overview
- Handling Concurrency
- An Example
Overview

Introduction
- Esterel for Reactive Systems
- Esterel Synthesis Options
- Multi-processing vs. Multi-threading

The Kiel Esterel Processor
- Architecture Overview
- Handling Concurrency
- An Example

Experimental Results
- KEP3 Evaluation Platform
- Resource Usage
- Scalability
- Code and RAM Sizes
Overview

Introduction
  Esterel for Reactive Systems
  Esterel Synthesis Options
  Multi-processing vs. Multi-threading

The Kiel Esterel Processor
  Architecture Overview
  Handling Concurrency
  An Example

Experimental Results
  KEP3 Evaluation Platform
  Resource Usage
  Scalability
  Code and RAM Sizes

Summary and Outlook
The Synchronous Language Esterel

- Created in the early 1980's
- Describes behavior of reactive systems
- Concurrency + numerous forms of preemption
- Deterministic behavior, clean semantics
Esterel Synthesis Options

- **Hardware**
  - Custom Hardware
  - Environment

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

- **Co-design**
  - COTS Assembler
  - COTS-μC
  - Custom Hardware
  - Environment
Esterel Synthesis Options

- **Hardware**
  - Custom Hardware
  - Environment

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

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

- **Patched Processor**
  - Extended Assembler
  - PIC
  - Extension
  - Environment
Esterel Synthesis Options

**Hardware**
- Custom Hardware
  - Environment

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

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

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

**Esterel Processor**
- Esterel Assembler
  - Esterel-μC
    - Environment
Multi-processing vs. Multi-threading

- **Multi-processing (EMPEROR/RePIC)**
  - an Esterel thread ≈ one independent RePIC processor
  - Thread Control Unit handles the synchronization and communication
  - `sync` command to synchronize threads

- **Multi-threading (KEP3)**
  - an Esterel thread ≈ several registers
  - priority-based scheduler
  - `PRI0` command to schedule threads
Overview

Introduction
   Esterel for Reactive Systems
   Esterel Synthesis Options
   Multi-processing vs. Multi-threading

The Kiel Esterel Processor
   Architecture Overview
   Handling Concurrency
   An Example

Experimental Results
   KEP3 Evaluation Platform
   Resource Usage
   Scalability
   Code and RAM Sizes

Summary and Outlook
Kiel Esterel Processor Architecture

- Reactive Core
  - Decoder & Controller
  - Reactive Block
  - Thread Block

- Interface Block
  - Interface signals
  - Local signals
  - ...

- Data Handling
  - Register file
  - ALU
  - ...

Introduction
The Kiel Esterel Processor
Experimental Results
Summary and Outlook

Architecture Overview
Handling Concurrency
An Example

Kiel Esterel Processor 3

- Thread Block
  - subPC Register File
  - Thread Controller

- Reactive Block
  - Register File
  - ALU
  - MUX

- Interface Block
  - Input/output Signals

- Interface signals
- Local signals
- ...

- Data Handling
  - Register file
  - ALU
  - ...

- Reactive Core
  - Decoder & Controller
  - Reactive Block
  - Thread Block

- Interface Block
  - Interface signals
  - Local signals
  - ...

- Data Handling
  - Register file
  - ALU
  - ...

Xin Li, Reinhard v. Hanxleden
Handling Concurrency
Handling Concurrency

```plaintext
[  
    await A
    'alert(||)'
    await B
]
emit 0;
```
Handling Concurrency

```plaintext
[ await A `\alert{||}`
  await B
];
emit 0;
⇒
PAR 1,P1
PAR 1,P2
PARE P3
P1: AWAITS A
P2: AWAITS B
P3: JOIN
  EMIT 0
```

Xin Li, Reinhard v. Hanxleden
An Example

% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
  [ await I;
    weak abort
    sustain R;
    when immediate A;
    emit O;
  ]
  loop
    pause;
    pause;
    present R then
      emit A;
    end
  end loop ]
end every
end signal
end module

Tick 0: main thread waits for signal S
An Example

```
% Esterel
module EXAMPLE:
  input S,I;
  output O;
  signal A,R in
  every S do
    [ await I;
      weak abort
      sustain R;
      when immediate A;
      emit O;
    ]
    loop
      pause;
      pause;
      present R then
        emit A;
      end
    end loop
  end every
end signal
end module
```

Tick 0: main thread waits for signal S

Tick 1: S is present; 1st thread waits for I, 2nd thread pauses
An Example

% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
  [ await I;
    weak abort
    sustain R;
    when immediate A;
  emit O;
  ]
  loop
    pause;
    pause;
    present R then
      emit A;
    end
  end loop ]
end every
end signal
end module

Tick 0: main thread waits for signal S
Tick 1: S is present; 1st thread waits for I, 2nd thread pauses
Tick 2: I is present; 1st thread sustains R, 2nd thread pauses again
An Example

% Esterel
module EXAMPLE:
  input S, I;
  output O;
  signal A, R in
  every S do
    [ await I;
      weak abort
      sustain R;
      when immediate A;
      emit O;
      ||
      loop
      pause;
      pause;
      present R then
      emit A;
      end
    end loop ]
  end every
end signal
end module

Tick | S | I
--- | --- | ---
     | R | R
     | A | O

Tick 0: main thread waits for signal S
Tick 1: S is present; 1st thread waits for I, 2nd thread pauses
Tick 2: I is present; 1st thread sustains R, 2nd thread pauses again
Tick 3: 1st thread sustains R; 2nd thread therefore emits A and hence (weakly) aborts 1st thread
Scheduling must obey signal dependencies!
An Example

```e
%% Esterel
module EXAMPLE:
  input S,I;
  output O;
  signal A,R in
  every S do
    [ await I;
      weak abort
      sustain R;
      when immediate A;
      emit O;
    ]
  loop
    pause;
    pause;
    present R then
    emit A;
  end
  end loop ]
end every
end signal
end module
```

**Tick 0:** main thread waits for signal $S$

**Tick 1:** $S$ is present; 1st thread waits for $I$, 2nd thread pauses

**Tick 2:** $I$ is present; 1st thread sustains $R$, 2nd thread pauses again

**Tick 3:** 1st thread sustains $R$; 2nd thread therefore emits $A$ and hence (weakly) aborts 1st thread

**Scheduling must obey signal dependencies!**

**Tick 4+:** etc.
An Example

```esterel
% Esterel
module EXAMPLE:
    input S,I;
    output O;
    signal A,R in
    every S do
        [ await I;
        weak abort
        sustain R;
        when immediate A;
        emit O;
        ]
        loop
        pause;
        pause;
        present R then
        emit A;
        end
        end loop
    end every
end signal
end module
```
An Example

```esterel
% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
  [ await I;
   weak abort
   sustain R;
   when immediate A;
   emit O;
   ||
   loop pause;
   pause;
   present R then
   emit A;
   end
   end loop ]
end every
end signal
end module
```

```
\begin{align*}
\text{await } & S; \\
\text{loop } & \text{abort } P; \\
\text{halt } & \text{when } S \\
\text{end loop }
\end{align*}
```

```
\begin{align*}
\text{sustain } & \text{’$$S$$’} \\
\text{loop } & \text{emit } S; \\
\text{pause } & \text{end loop }
\end{align*}
```

```
\begin{align*}
\text{loop } & P \\
\text{end loop }
\end{align*}
```

```
\begin{align*}
\text{A: } & P; \\
\text{goto } & A
\end{align*}
```
An Example

% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
[ await I;
  weak abort
  sustain R;
  when immediate A;
  emit O;
  ]
  | loop
  pause;
  pause;
  present R then
  emit A;
end
end every
end signal
end module

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

loop
  emit S;
  pause;
  end loop

A:
  p;
  goto A

% KEP3 ASM
INPUT S,I
OUTPUT O
SIGNAL A,R
[00] await S
[01] A0: abort S,A4
[02] PAR 3,P1 % Fork
[03] PAR 2,P2
[04] PARE P3 % Prios:
[05] P1: await I % 3
[06] WABORTI A,A2 % 3
[07] A1: emit R % 3
[08] PRIO 1 % 1
[09] PRIO 3 % 3
[10] PAUSE % 3
[12] A2: emit O % 3
[14] PAUSE % 2
[16] EMIT A % 2
[18] P3: JOIN % Join
[19] HALT
[20] A4: GOTO A0

Xin Li, Reinhard v. Hanxleden
A Concurrent Reactive Esterel Processor
An Example

% KEP3 ASM
INPUT S,I
OUTPUT O
SIGNAL A,R

[00]  AWAIT S
[01]  A0:  ABORT S,A4
[02]  PAR 3,P1  % Fork
[03]  PAR 2,P2
[04]  PARE P3  % Prios:
[05]  P1:  AWAIT I  % 3
[06]  WABORTI A,A2  % 3
[07]  A1:  EMIT R  % 3
[08]  PRIO 1  % 1
[09]  PRIO 3  % 3
[10]  PAUSE  % 3
[12]  A2:  EMIT O  % 3
[14]  PAUSE  % 2
[16]  EMIT A  % 2
[18]  P3:  JOIN  % Join
[19]  HALT
[20]  A4:  GOTO A0

Tick 0: main thread stays at [00]
An Example

```
% KEP3 ASM
INPUT S, I
OUTPUT O
SIGNAL A, R
[00] AWAIT S
[01] A0: ABORT S, A4
[02] PAR 3, P1  % Fork
[03] PAR 2, P2
[04] PARE P3  % Prios:
[05] P1: AWAIT I  % 3
[06] WABORTI A, A2  % 3
[07] A1: EMIT R  % 3
[08] PRIO 1  % 1
[09] PRIO 3  % 3
[10] PAUSE  % 3
[12] A2: EMIT O  % 3
[14] PAUSE  % 2
[16] EMIT A  % 2
[18] P3: JOIN  % Join
[19] HALT
[20] A4: GOTO A0
```

Tick 0: main thread stays at [00]

Tick 1: main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]
An Example

```
% KEP3 ASM
INPUT S,I
OUTPUT O
SIGNAL A,R

[00] AWAIT S
[01] A0: ABORT S,A4
[02] PAR 3,P1 % Fork
[03] PAR 2,P2
[04] PARE P3 % Prios:
[05] P1: AWAIT I % 3
[06] WABORTI A,A2 % 3
[07] A1: EMIT R % 3
[08] PRIO 1 % 1
[09] PRIO 3 % 3
[10] PAUSE % 3
[12] A2: EMIT O % 3
[14] PAUSE % 2
[16] EMIT A % 2
[18] P3: JOIN % Join
[19] HALT
[20] A4: GOTO A0
```

Tick 0: main thread stays at [00]

Tick 1: main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]

Tick 2: main thread stays at [18], 1st thread stays at [10], 2nd thread stays at [14]
An Example

```
% KEP3 ASM
INPUT S,I
OUTPUT O
SIGNAL A,R

[00] AWAIT S
[01] A0: ABORT S,A4
[02] PAR 3,P1 % Fork
[03] PAR 2,P2
[04] PARE P3 % Prios:
[05] P1: AWAIT I % 3
[06] WABORTI A,A2 % 3
[07] A1: EMIT R % 3
[08] PRIO 1 % 1
[09] PRIO 3 % 3
[10] PAUSE % 3
[12] A2: EMIT O % 3
[14] PAUSE % 2
[16] EMIT A % 2
[18] P3: JOIN % Join
[19] HALT
[20] A4: GOTO A0
```

Tick

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
</tr>
</tbody>
</table>

**Tick 0:** main thread stays at [00]
**Tick 1:** main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]
**Tick 2:** main thread stays at [18], 1st thread stays at [10], 2nd thread stays at [14]
**Tick 3:** main thread stays at [18], 1st thread terminates, 2nd thread stays at [13]
An Example

```
% KEP3 ASM
INPUT S,I
OUTPUT O
SIGNAL A,R
[00]    AWAIT S
[01] A0: ABORT S,A4
[02]    PAR 3,P1  % Fork
[03]    PAR 2,P2
[04]    PARE P3  % Prios:
[05] P1:    AWAIT I   % 3
[06]    WABORTI A,A2 % 3
[07] A1:    EMIT R   % 3
[08]    PRIO 1   % 1
[09]    PRIO 3   % 3
[10]    PAUSE   % 3
[12] A2:    EMIT O   % 3
[14]    PAUSE   % 2
[16]    EMIT A   % 2
[18] P3:    JOIN   % Join
[19]    HALT
[20] A4:    GOTO A0
```

Tick 0: main thread stays at [00]
Tick 1: main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]
Tick 2: main thread stays at [18], 1st thread stays at [10], 2nd thread stays at [14]
Tick 3: main thread stays at [18], 1st thread terminates, 2nd thread stays at [13]
Tick 4+: etc.
The Test Harness

1. Validate correctness of architecture (+ compiler)
2. Measure performance (cycle accurate)
Resource Usage KEP3 vs. EMPEROR

<table>
<thead>
<tr>
<th></th>
<th>KEP3-A</th>
<th>-B</th>
<th>-C</th>
<th>-D</th>
<th>-E</th>
<th>EMPEROR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output</td>
<td>11/11</td>
<td>16/16</td>
<td>11/11</td>
<td>32/32</td>
<td>24/24</td>
<td>24/24</td>
</tr>
<tr>
<td>Valued Input/Output</td>
<td>2/2</td>
<td>2/2</td>
<td>3/3</td>
<td>3/3</td>
<td>2/2</td>
<td>2/2</td>
</tr>
<tr>
<td>Thread Number</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Preemption Nest</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>4+4</td>
</tr>
<tr>
<td>Counter Value Range</td>
<td>255</td>
<td>65535</td>
<td>65535</td>
<td>65535</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Variable Register Num</td>
<td>16</td>
<td>64</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>64+64</td>
</tr>
<tr>
<td>Datapath Width [bits]</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Logic Cells</td>
<td>1670</td>
<td>2474</td>
<td>2692</td>
<td>4020</td>
<td>2086</td>
<td>4761</td>
</tr>
<tr>
<td>Max Osc Freq [MHz]</td>
<td>52.75</td>
<td>45.31</td>
<td>39.96</td>
<td>39.48</td>
<td>42.68</td>
<td>35.38</td>
</tr>
<tr>
<td>Instruction Freq [MHz]</td>
<td>17.58</td>
<td>15.10</td>
<td>13.32</td>
<td>13.16</td>
<td>14.23</td>
<td>8.84</td>
</tr>
</tbody>
</table>

- KEP3 is highly configurable
- Resource usage less than 50% of comparable EMPEROR
- Higher clock frequency
- Fewer clock cycles per instruction
Scalability

<table>
<thead>
<tr>
<th>Thread Cnt</th>
<th>KEP3</th>
<th>EMPEROR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Logic Cells</td>
<td>2086</td>
</tr>
<tr>
<td>4</td>
<td>2170</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2306</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2466</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2946</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>3758</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>4768</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>5564</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Max Osc Freq [MHz]</td>
<td>42.68</td>
</tr>
<tr>
<td>126</td>
<td>42.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
<tr>
<td></td>
<td>42.68</td>
<td>40.26</td>
</tr>
</tbody>
</table>

- Resource usage of KEP3 with 102 threads comparable to EMPEROR2 with two threads
- Still have higher clock (and instruction) frequency
### Code and RAM Sizes

<table>
<thead>
<tr>
<th>Module</th>
<th>Thread Cnt/ Preemption nesting depth</th>
<th>Code Size (in words)</th>
<th>RAM Usage (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Microblaze</td>
<td>KEP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V5</td>
<td>V7</td>
</tr>
<tr>
<td>BAT_DIAG</td>
<td>1/1</td>
<td>487</td>
<td>1274</td>
</tr>
<tr>
<td>VER_ACC_DIAG</td>
<td>1/1</td>
<td>433</td>
<td>1229</td>
</tr>
<tr>
<td>BELT</td>
<td>2/3</td>
<td>617</td>
<td>1255</td>
</tr>
<tr>
<td>ABCD</td>
<td>4/1</td>
<td>1357</td>
<td>1547</td>
</tr>
<tr>
<td>RUNNER</td>
<td>2/5</td>
<td>688</td>
<td>1323</td>
</tr>
<tr>
<td>ARBITER12</td>
<td>36/1</td>
<td>3162</td>
<td>1703</td>
</tr>
<tr>
<td>LONG_SPEED_STRAT</td>
<td>1/1</td>
<td>573</td>
<td>1306</td>
</tr>
</tbody>
</table>

Comparison of the codes sizes in words (one word equals four bytes), and comparison of RAM usage in bytes.

- 88% reduction of code size
- 54% reduction of RAM usage
- can be optimized further
The Kiel Esterel Processor

- ... is developed for the direct execution of Esterel programs
- ... employs a multi-threading reactive architecture
- ... supports Esterel’s concurrency and preemption in a very precise, direct and efficient way
- ... allows to combine Esterel control constructs in an arbitrary fashion
- ... is configurable and efficient
Outlook

KEP

- Implementation in Esterel!
Outlook

KEP
- Implementation in Esterel!

Compiler
- Optimize priority assignments
Outlook

KEP
- Implementation in Esterel!

Compiler
- Optimize priority assignments

Both
- HW/SW-Codeign—implement compound logical expressions in customizable HW block
Outlook

KEP
- Implementation in Esterel!

Compiler
- Optimize priority assignments

Both
- HW/SW-Codeign—implement compound logical expressions in customizable HW block

Thanks!
Questions/Comments?
Appendix

Reactive Processors

Instruction Set

Handling Concurrency

Handling Preemption
Reactive Processors

- Patched Reactive Processor
  - a COTS processor core is combined with an external hardware block
  - implements several additional Esterel-style instructions
  - a multi-processing architecture for implementing concurrency
Reactive Processors

- **Patched Reactive Processor**
  - a COTS processor core is combined with an external hardware block
  - implements several additional Esterel-style instructions
  - a multi-processing architecture for implementing concurrency

- **Custom Reactive Processor**
  - a full-custom reactive core
  - instruction set and data path are tailored for Esterel
  - a multi-threading architecture for implementing concurrency
## Instruction Set 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 \</td>
<td>q ) ]</td>
</tr>
<tr>
<td>PARE endAddr 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 ( \ldots ) when [( n )] S</td>
<td>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</td>
</tr>
<tr>
<td>[w]ABORT[I] S, endAddr</td>
<td>[weak] abort ( \ldots ) when [immediate] S</td>
<td></td>
</tr>
<tr>
<td>SUSPEND[I] S, endAddr</td>
<td>suspend ( \ldots ) when [immediate] S</td>
<td></td>
</tr>
<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>
</tbody>
</table>
### Instruction Set 2/2

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL $S$</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>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>CLRC/SETC</td>
<td></td>
<td>Clear/set carry bit</td>
</tr>
<tr>
<td>LOAD reg, n</td>
<td></td>
<td>Load/store register</td>
</tr>
<tr>
<td>{SR</td>
<td>SRC</td>
<td>SL</td>
</tr>
<tr>
<td>{ADD[C]</td>
<td>SUB[C]</td>
<td>MUL} reg, n</td>
</tr>
<tr>
<td>{ANDR</td>
<td>ORR</td>
<td>XORR} reg, n</td>
</tr>
<tr>
<td>CMP reg, n</td>
<td></td>
<td>Compare, branch conditionally.</td>
</tr>
<tr>
<td>JW cond, addr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling Concurrency

A thread has its

▶ independent PC
▶ address range
Handling Concurrency

A thread has its

- independent PC
- address range
- priority value
  - assigned when a thread is created
  - dynamically changed via PRI0 instruction
  - 255 priority levels

status flags
- ThreadEnable: enabled or disabled
- ThreadActive: active or inactive

parent thread

A Concurrent Reactive Esterel Processor Slide 23
Handling Concurrency

A thread has its

- independent PC
- address range
- priority value
  - assigned when a thread is created
  - dynamically changed via PRI0 instruction
  - 255 priority levels
- status flags
  - ThreadEnable: *enabled* or *disabled*
  - ThreadActive: *active* or *inactive*
Handling Concurrency

A thread has its

- independent PC
- address range
- priority value
  - assigned when a thread is created
  - dynamically changed via PRI0 instruction
  - 255 priority levels
- status flags
  - ThreadEnable: enabled or disabled
  - ThreadActive: active or inactive
- parent thread
  - a thread is blocked by any of its active sub threads
Handling Concurrency

Executing sub threads

- all *enabled* threads are *active* in a new tick
- an active sub thread blocks its parent thread
- the highest priority/earliest generated thread gets to execute
- a thread is *inactive* in the current tick when a non-instantaneous instruction is executed
- a tick is finished when all of threads are *inactive*
Handling Concurrency

Terminating sub threads

- comparing the expected fetch address with thread’s address range
  - equal: finishes its body
  - greater: terminated by an outer abortion or exception
- thread is disabled (and inactive)
Handling Preemption
Inside/Outside Preemption Range Watching (IOPRW)

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?
# Comparison of Implementations

<table>
<thead>
<tr>
<th></th>
<th>Hardware</th>
<th>Software</th>
<th>Co-design</th>
<th>Patched Processor</th>
<th>Esterel Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</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>
</tr>
<tr>
<td>Esterel Compliance</td>
<td>++</td>
<td>++</td>
<td>+/-</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Cost</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>
</tr>
</tbody>
</table>

Note: ++ = best; − − = worst.

E.g. Cost ++ means very low production costs.