Multi-Threaded Reactive Processing

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

SYNCHRON’06
November 2006
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
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
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 Statecharts (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 (v7)
Reactive Processing Part II: The Execution Platform
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
Overview

Introduction

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

The Compiler

Wrap-Up
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
```
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 \( S \)
- ...
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;
```

- 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, ...

Kiel Esterel Processor 3

- Instruction Memory
- Decoder & Controller
- Thread Controller
- subPC Register File
- Tick Manager
- Address Multiplexer
- Instruction Fetch
- Tick, TickWarn & InstrCk
- ALU
- Register File
- MUX
- OscCik
- Reset

Input/output Signals
Interface Block
The Architecture of the Reactive Core

Diagram showing the architecture of the reactive core with various components and signals such as cclk, PC[15:0], innerOp[4:0], innerAddr[8:0], innerMark[5:0], innerData[15:0], countSPEC[15:0], wrPreemption, wrAWAIT, wrPAR, InstrClk, Tick, ThreadID[6:0], and Unicode Signal connected to different blocks like Watcher0, Decoder, Watchers Priority Controller, Local Watchers, or Thread Watchers.
Overview

Introduction

The Kiel Esterel Processor

The Compiler

The Concurrent KEP Assembler Graph (CKAG)
Handling Thread Dependencies

Wrap-Up
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
module EXAMPLE:
output A, R;
end module

% module Example
OUTPUT A, R
module EXAMPLE:
output A,R;
[
|
|
]
end module

module: EXAMPLE

% module Example
OUTPUT A,R
module EXAMPLE:
output A, R;
[
 ||
]
end module

% module Example
OUTPUT A, R
module EXAMPLE:
output A,R;
[

]
end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0:

L08: A1:
module EXAMPLE:
  output A,R;
  [ weak abort
    when A ||
  ]
end module
```plaintext
module EXAMPLE:
output A,R;
[
  weak abort
  when A
||
]
end module
```

```plaintext
% module Example
OUTPUT A,R
L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0:
L08: A1:
```

```plaintext
module: EXAMPLE

L03 fork

L04 A0
L08 A1

L04 WABORT A,A3
```
module EXAMPLE:
output A,R;
[
  weak abort 
  when A 
  ||
]
end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L08: A3:A1:
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
when A
||
]
end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L08: A3:A1:

module: EXAMPLE

L03 fork

L04 A0
L08 A1

L04 WABORT A,A3
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
  when A
][
end module

% module Example
OUTPUT A,R
L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L08: A3:A1:

L03 fork

L04 A0
L08 A1

L04 WABORT A,A3

L05 A4

L05 EMIT R

L06 PAUSE

L07 GOTO A4
module EXAMPLE:
output A,R;
[
    weak abort sustain R
    when A
]
end module

% module Example
OUTPUT A,R
L01:    PAR 1,A0,1
L02:    PAR 1,A1,2
L03:    PARE A2
L04:    A0: WABORT A,A3
L08:    A3:A1:

L03 fork
L04 A0
L08 A1
L04 WABORT A,A3
L05 A4
L05 EMIT R
L06 PAUSE
L07 GOTO A4
L08 A3
module EXAMPLE:
output A,R;
[
    weak abort
    sustain R
    when A
]
]
end module

% module Example

OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:
module EXAMPLE:
output A,R;
[
  weak abort sustain R when A
  ||
  pause;
]
end module

% module Example
OUTPUT A,R
L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:
module EXAMPLE:
output A,R;
[
   weak abort sustain R when A ||
   pause;
]
end module

% module Example

OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
  when A
||
  pause;
]
end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:PAUSE
module EXAMPLE:
output A,R;
[
    weak abort
    sustain R
    when A
    ||
    pause;
    present R then
    emit A
    end present
]
end module
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
  when A
  ||
  pause;
  present R then
  emit A
  end present
]
end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:PAUSE
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
  when A
[]
  pause;
  present R then emit A
  end present
] end module

% module Example
OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:PAUSE
L09: PRESENT R,A5
L10: EMIT A
L11: A5:
module EXAMPLE:
output A,R;
[
  weak abort
  sustain R
when A
||
  pause;
  present R then
  emit A
  end present
] end module

% module Example

OUTPUT A,R

L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:PAUSE
L09: PRESENT R,A5
L10: EMIT A
L11: A5:
module EXAMPLE:
output A,R;
[ weak abort sustain R when A || pause; present R then emit A end present ]
end module

% module Example
OUTPUT A,R
L01: PAR 1,A0,1
L02: PAR 1,A1,2
L03: PARE A2
L04: A0: WABORT A,A3
L05: A4: EMIT R
L06: PAUSE
L07: GOTO A4
L08: A3:A1:PAUSE
L09: PRESENT R,A5
L10: EMIT A
L11: A5:A2:JOIN
L12: HALT
Dealing With Dependencies

1. Add dependencies to CKAG
2. Compute statement priorities
3. Assign initial thread prios
4. Insert PRI0 stmts
Dealing With Dependencies

% module Example

OUTPUT A,R
EMIT _TICKLEN,#14

[L01,T0,P2] PAR 2,A0,2
[L02,T0,P2] PAR 1,A1,1
[L03,T0,P2] PARE A2,2
[L04,T2,P2] A0: WABORT A,A3
[L05,T2,P2] A4: EMIT R
[L06,T2,P1] PRIO 1
[L07,T2,P2] PRIO 2
[L08,T2,P1/2] PAUSE
[L09,T2,P2] GOTO A4
[L10,T2,P2] A3: NOTHING
[L11,T1,P1] A1: PRIO 2
[L12,T1,P1/2] PAUSE
[L13,T1,P2] PRESENT R,A5
[L14,T1,P2] EMIT A
[L15,T1,P1] A5: PRIO 1
[L16,T1,P1] NOTING
[L17] A2: JOIN 0
[L18,T0,P1/1] HALT
Example—Execution Trace

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

```
module EXAMPLE:
  output A,R;
  [
    weak abort
    sustain R;
  when A
  ||
    pause;
    present R then
    emit A
  end present
] end module
```

```
% module Example

OUTPUT A,R
EMIT _TICKLEN,#14
[L01,T0,P2] PAR 2,A0,2
[L02,T0,P2] PAR 1,A1,1
[L03,T0,P2] PARE A2,2
[L04,T2,P2] A0: WABORT A,A3
[L05,T2,P2] A4: EMIT R
[L06,T2,P1] PRIO 1
[L07,T2,P2] PRIO 2
[L08,T2,P1/2] PAUSE
[L09,T2,P2] GOTO A4
[L10,T2,P1] A3: NOTHING
[L11,T1,P1] A1: PRIO 2
[L12,T1,P1/2] PAUSE
[L13,T1,P2] PRESENT R,A5
[L14,T1,P2] EMIT A
[L15,T1,P1] A5: PRIO 1
[L16,T1,P1] NOTHING
[L17] A2: JOIN 0
[L18,T0,P1/1] HALT
```

- Tick 1 -
  ! reset;
  % In:
  % Out: R
T0: L01,L02,L03
T2: L04,L05,L06
T1: L11,L12
T2: L07,L08
T0: L17
- Tick 2 -
  % In:
  % Out: A R O
T2: L08,L09,L05,L06
T1: L12,L13,L14,L15,L16
T2: L07,L08,L10
T0: L17,L18
- Tick 3 -
  % In:
  % Out:
T0: L18

---

The Concurrent KEP Assembler Graph (CKAG)
Handling Thread Dependencies
CKAG for tcint
tcint ThreadId’s
tcint ThreadId’s with Thread Dependencies
Overview

Introduction

The Kiel Esterel Processor

The Compiler

Wrap-Up

KEP Evaluation Platform
Experimental Results
Summary & Outlook
Identify that benchmark
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
## Code Characteristics and Compilation Times

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel</th>
<th>KEP3a (Unoptimized</th>
<th>optimized)</th>
<th>MicroBlaze</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Threads</td>
<td>Preemptions</td>
<td>CKAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cnt</td>
<td>max</td>
<td>nodes dep.</td>
<td>max priority instr</td>
</tr>
<tr>
<td></td>
<td>depth</td>
<td>max</td>
<td>max depth</td>
<td>max pr.io</td>
</tr>
<tr>
<td></td>
<td>conc</td>
<td>max</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>depth</td>
<td>max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abc</td>
<td>4</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>cd</td>
<td>4</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>abcd</td>
<td>4</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>abcdf</td>
<td>4</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>eight but</td>
<td>8</td>
<td>2</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>chan prot</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>reactor ctrl</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>runner</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>example</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>ww button</td>
<td>13</td>
<td>3</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>grey button</td>
<td>17</td>
<td>3</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>tcint</td>
<td>39</td>
<td>5</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>mca200</td>
<td>59</td>
<td>5</td>
<td>49</td>
<td>4</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 V5</td>
<td>ACRT V5</td>
<td>WCRT Ratio to ACRT Ratio</td>
</tr>
<tr>
<td></td>
<td>V7 CEC</td>
<td>V7 CEC</td>
<td>best MB best MB</td>
</tr>
<tr>
<td>abcd</td>
<td>1559 954 1476</td>
<td>1464 828 1057</td>
<td>135 0.14 87 0.11</td>
</tr>
<tr>
<td>abcdef</td>
<td>2281 1462 1714</td>
<td>2155 1297 1491</td>
<td>201 0.14 120 0.09</td>
</tr>
<tr>
<td>eight_but</td>
<td>3001 1953 2259</td>
<td>2833 1730 1931</td>
<td>267 0.14 159 0.09</td>
</tr>
<tr>
<td>chan_prot</td>
<td>754 375 623</td>
<td>683 324 435</td>
<td>117 0.31 60 0.19</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>487 230 397</td>
<td>456 214 266</td>
<td>54 0.23 45 0.21</td>
</tr>
<tr>
<td>runner</td>
<td>566 289 657</td>
<td>512 277 419</td>
<td>36 0.12 15 0.05</td>
</tr>
<tr>
<td>example</td>
<td>467 169 439</td>
<td>404 153 228</td>
<td>42 0.25 24 0.16</td>
</tr>
<tr>
<td>ww_button</td>
<td>1185 578 979</td>
<td>1148 570 798</td>
<td>72 0.12 51 0.09</td>
</tr>
<tr>
<td>greycounter</td>
<td>1965 1013 2376</td>
<td>1851 928 1736</td>
<td>528 0.52 375 0.40</td>
</tr>
<tr>
<td>tcint</td>
<td>3580 1878 2350</td>
<td>3488 1797 2121</td>
<td>408 0.22 252 0.14</td>
</tr>
<tr>
<td>mca200</td>
<td>75488 29078 12497</td>
<td>73824 24056 11479</td>
<td>2862 0.23 1107 0.10</td>
</tr>
</tbody>
</table>

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

- **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 (word)</th>
<th>KEP3a-opt. Code (word)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>V5   V7    CEC</td>
<td>abs.    rel.     abs.    rel.     abs.    rel.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1]   [2]  (best)</td>
<td>[3]    [3]/[1]  [4]    [4]/[2]  [5]    [5]/[3]</td>
<td></td>
</tr>
<tr>
<td>abcd</td>
<td>160</td>
<td>6680  7928  7212</td>
<td>168    1.05       756    0.11       164    0.93</td>
<td></td>
</tr>
<tr>
<td>abcsdf</td>
<td>236</td>
<td>9352  9624  9220</td>
<td>252    1.07       1134  0.12       244    0.94</td>
<td></td>
</tr>
<tr>
<td>eight_but</td>
<td>312</td>
<td>12016 11276 11948</td>
<td>336    1.08       1512  0.13       324    0.94</td>
<td></td>
</tr>
<tr>
<td>chan_prot</td>
<td>42</td>
<td>3808  6204  3364</td>
<td>66     1.57       297    0.09       62     0.94</td>
<td></td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>27</td>
<td>2668  5504  2460</td>
<td>38     1.41       171    0.07       34     0.89</td>
<td></td>
</tr>
<tr>
<td>runner</td>
<td>31</td>
<td>3140  5940  2824</td>
<td>39     1.22       175    0.06       27     0.69</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>20</td>
<td>2480  5196  2344</td>
<td>31     1.55       139    0.06       28     0.94</td>
<td></td>
</tr>
<tr>
<td>ww_button</td>
<td>76</td>
<td>6112  7384  5980</td>
<td>129    1.7        580    0.10       95     0.74</td>
<td></td>
</tr>
<tr>
<td>greycouner</td>
<td>143</td>
<td>7612  7936  8688</td>
<td>347    2.43       1567  0.21       343    1</td>
<td></td>
</tr>
<tr>
<td>tcint</td>
<td>355</td>
<td>14860 11376 15340</td>
<td>437    1.23       1968  0.17       379    0.87</td>
<td></td>
</tr>
<tr>
<td>mca200</td>
<td>3090</td>
<td>104536 77112 52998</td>
<td>8650   2.79       39717 0.75       8650  1</td>
<td></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 ¹ (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

¹ Based 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)

---

2 For comparison, a MicroBlaze implementation requires around 1k slices and 309k gates; a two threads EMPEROR platform requires around 2k slices
## 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>total ratio</td>
<td>rel. abs. ratio</td>
<td>total rel. abs.</td>
<td>rel. abs. rel. rel.</td>
</tr>
<tr>
<td>abcd</td>
<td>16513</td>
<td>3787</td>
<td>4.36</td>
<td>1521</td>
<td>3082</td>
</tr>
<tr>
<td>abcdef</td>
<td>29531</td>
<td>7246</td>
<td>4.08</td>
<td>3302</td>
<td>6043</td>
</tr>
<tr>
<td>eight_but</td>
<td>39048</td>
<td>10073</td>
<td>3.88</td>
<td>5356</td>
<td>8292</td>
</tr>
<tr>
<td>chan_prot</td>
<td>5119</td>
<td>1740</td>
<td>2.94</td>
<td>707</td>
<td>990</td>
</tr>
<tr>
<td>reactor_ctrl</td>
<td>151</td>
<td>48</td>
<td>3.15</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>runner</td>
<td>5052</td>
<td>704</td>
<td>7.18</td>
<td>307</td>
<td>0</td>
</tr>
<tr>
<td>example</td>
<td>208</td>
<td>60</td>
<td>3.47</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>ww_button</td>
<td>292</td>
<td>156</td>
<td>1.87</td>
<td>92</td>
<td>26</td>
</tr>
<tr>
<td>greycounter</td>
<td>160052</td>
<td>34560</td>
<td>4.63</td>
<td>14043</td>
<td>26507</td>
</tr>
<tr>
<td>tcint</td>
<td>80689</td>
<td>33610</td>
<td>2.4</td>
<td>16769</td>
<td>5116</td>
</tr>
<tr>
<td>mca200</td>
<td>982417</td>
<td>256988</td>
<td>3.82</td>
<td>125055</td>
<td>242457</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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*
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 (e.g., for data handling)
- Adaptation to non-Esterel languages
- Apply compilation approach to standard processors ("multi-threaded simulation")
- To go further:
  http://www.informatik.uni-kiel.de/rtsys/kep/
Identify that benchmark . . .

Thanks!
Questions/Comments?
Overview

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

The Compiler

Further Measurements

Summary
### 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</td>
<td></td>
</tr>
<tr>
<td>PARE endAddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOIN</td>
<td></td>
<td>Set the priority of the current thread</td>
</tr>
<tr>
<td>PRIOR Prio</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<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></td>
<td>$n$ can be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. #data: immediate data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. reg: register contents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. ?$S$: value of a signal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. PRE(?$S$): previous value of a signal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Esterel Syntax</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<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[I] S, addr</strong></td>
<td>await S do</td>
<td></td>
</tr>
<tr>
<td><strong>CAWAITE</strong></td>
<td></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><strong>EMIT S [, {#data|reg}]</strong></td>
<td>emit S [(val)]</td>
<td>Emit (valued) signal S</td>
</tr>
<tr>
<td><strong>SUSTAIN S [, {#data|reg}]</strong></td>
<td>sustain S [(val)]</td>
<td>Sustain (valued) signal S</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 PRIIO instruction
- **status flags**
  - ThreadEnable
  - ThreadActive

```esterel
% Esterel
[
  p
  ||
  q
];
```

```kep
% 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)

```plaintext
% Esterel
trap T1 in
  trap T2 in
  [ p;
    exit T1;
  ]
  q;
  exit T2; ];
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:
```
Overview

KEP3a Instruction Set + Architecture

The Compiler
  The Concurrent KEP Assembler Graph
  Kep Thread Ids
  Priority Assigning
  Cyclicity

Further Measurements

Summary
Esterel Kep Compiling Scheme

- parsing and module expanding is done by Columbia Esterel Compiler (CEC)
- result is an Esterel abstract syntax tree (AST)
- dismantle some more complex esterel statements to simple statements
- creating assembler program and Concurrent KEP Assembler Graph (CKAG) by recursive visiting of the AST
- do computations and optimizations on CKAG
  - e.g. do priority assigning to ensure correct behavior
  - e.g. do statement collapsing to use less statements
- printing to KEP assembler or viewing the CKAG
Concurrent KEP Assembler Graph (CKAG)

- represents the control flow of KEP assembler
- needed to realize more complex compiling computations
- each node contains a KEP statement and *Thread Id*
- different kind of nodes
  - instantaneous: TransientNode, LabelNode
  - not instantaneous: DelayNode
  - parallel: ForkNode, JoinNode
- different kind of edges
  - sequence control flow
  - abort preemption, weak abort preemption,
  - exit preemption
Kep Thread Id

- contains an unique integer value and the parent's thread pointer
- the main thread has no parent thread (NULL pointer)
- a sequence of fork nodes declares the sub-thread id's
- Kep Thread Id structure has much similarity to a tree
- three important relations between thread id's
  - subthread
  - concurrent
  - sequence
Thread Id Example 1

- T1 and T2 are sub-threads of T0
- T1 and T2 are concurrent to each other
- no sequence behaviour
Thread Id Example 2

- \{T1, ..., T7\} are sub-threads of T0
- T4 and T5 are sub-threads of T3
- \{T1, T2, T3\}, \{T1, T2, T4, T5\} and \{T6, T7\} are in each case concurrent to each other
- \{T1, T2, T3, T4, T5\} are in sequence to \{T6, T7\}
Priority Assigning Scheme

- determining *signal dependencies*
- relevant for our concern are only *concurrent* signal dependencies, which we call *priority dependencies*
- run priority assigning algorithm fulfilling all priority constraints
- if assigning algorithm ends successfully:
  - setting priorities of PAR and PARE statements
  - inserting priority statements according to the priority assigning if necessary
Signal Dependencies

- a writer-reader pair of the same signal $S$ defines a **signal dependency**

- writer statements: EMIT, SUSTAIN, SIGNAL, EXIT, SETV

- reader statements: PRESENT, AWAIT, PAUSE, HALT, SUSTAIN, EMIT
Signal Dependency Example 1

EMIT S

PRESENT S, A0
Signal Dependency Example 2

- SETV S,#2
- EMIT T,?S
- PRESENT T,A1
Priority Dependencies

- A priority dependency is defined as a signal dependency with concurrent `KepThreadIds`.
- Execution order can and must be determined by priorities respectively thread id’s.
- Priority dependencies are only relevant when executable within the same tick, otherwise superfluous.
- Analysis is conservative in a way that superfluous dependencies may exist.
Priority Dependency Example

```
... PAR*
EMIT S
...
PRESENT S, A1
...
PRESENT S, A0
...```

Xin Li, Marian Boldt, Reinhard v. Hanxleden
Constraints

- a priority dependency \((w, r)\) causes a writer constraint:
  \[
  w.prio >_w r.prio :=
  \]
  \[
  w.prio > r.prio \lor (w.prio = r.prio \land w.id > r.id)
  \]

- instantaneous execution order causes control flow constraints: priorities never grow instantaneously

- node: the control flow successors of DelayNodes causes no constraints because they do not represent instantaneous execution
Assigning Algorithm

- try to fulfill all constraints by a DFS like algorithm
- basic idea to compute the priority $n.prio$ of a node $n$:
  \[
  n.prio = \max(\{n.inst\_succ.prio\} \cup \{n.reader.prio + 1\})
  \]
- for a DelayNode $d$ additionally the priority $d.prio\_next$ of the next instant is computed: this ensures termination of a one pass traversal
  \[
  d.prio\_next = \max(\{n.next\_succ.prio\} \cup \{n.reader.prio + 1\})
  \]
Priority Dismantling

- sometimes we cannot fulfill the constraints because of statements which are writer and reader at the same time:
  - SUSTAIN S
  - EMIT S, expr
- separate writing and reading part by dismantling
- priority statements can be inserted in between
The Concurrent KEP Assembler Graph

Kep Thread Ids

Priority Assigning

Cyclicity

PAR*

WABORTI A,A3

AWAIT R

SUSTAIN R

EMIT A

JOIN 0
PAR*

WABORTI A,A3

A4

AWAIT R

EMIT R

EMIT A

PAUSE

GOTO A4

JOIN 0

A3
PAR*

WABORTI A,A3

A4

EMIT R

PRIO 1

AWAIT R

PRIO 2

EMIT A

PAUSE

GOTO A4

A3

JOIN 0
The Concurrent KEP Assembler Graph

Kep Thread Ids

Priority Assigning

Cyclicity
PAR*

EMIT V1,?V2

PRESENT V1,A3

EMIT V2,#3

A3
PAR*

EMIT V1, V2

PRESENT V1, A3

EMIT V2, #3

A3

i

2

1

f

t

i
PAR

EMIT V1

PRESENT V1,A3

EMIT V2

SETV V2,#3

SETV V1,?V2

A3
Thread Id Optimization

- same priority:
  - thread id has scheduling relevance
  - no priority statements needed
  - try to optimize thread id’s

- thread dependency: priority dependency modulo nodes/statements

- optimization: assign dependency fathers higher thread id than children

- node: sub-thread constraints must be obtained
Superfluous Priority Dependencies

- Priority dependencies which are not executable within the same instant are called superfluous dependencies
- in general difficult to compute
- special case:
  - search least common fork node (nodes are concurrent)
  - make instant analysis starting with fork node
  - if one node has only instantaneous and the other not instantaneous paths: remove Priority Dependency
PAR*

EMIT S

PAUSE

PRESENT S,A0
PAR*

- inst
- \( \ldots \)

EMIT S
PAUSE

- \( \ldots \)

PRESENT S, A0
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 \( n \):
  - \( n.\text{join} \): the JOIN statement corresponding to \( n \),
  - \( n.\text{sub} \): the transitive closure of nodes in threads generated by \( n \).

- for abort nodes \( n \) ([L|T][W]ABORT[I], SUSPEND[I]):
  - \( n.\text{end} \): the end of the abort scope opened by \( n \),
  - \( n.\text{scope} \): the nodes within \( n \)'s abort scope.

- for all nodes \( n \):
  - \( n.\text{prio} \): the priority that the thread executing \( n \) should be running with

- for \( n \in D \cup F \),
  - \( n.\text{prionext} \): 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 $n$:

- $n.\text{suc}_c$: the control successors.
- $n.\text{suc}_w$: the weak abort successors
- $n.\text{suc}_s$: the strong abort successors
- $n.\text{suc}_f$: the flow successors
  
  the set $n.\text{suc}_c \cup n.\text{suc}_w \cup n.\text{suc}_s$

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

- $n.\text{suc}_{wf}$: the weak fork abort successors
- $n.\text{suc}_{sf}$: 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:

$$n \in D \land n' \in n.suc_w$$

$$\lor n \in F \land n' \in n.suc_c \cup n.suc_{wf}$$

$$\lor n \in T \land n' \in n.suc_c \cup n.dep_i$$

$$\lor n \in T \land n' \in n.dep_d \land n'' \in n'.suc_c \cup n'.suc_s \cup n'.suc_{sf}.$$
Overview

KEP3a Instruction Set + Architecture

The Compiler

Further Measurements

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

Summary
Code Characteristics and Compilation Times

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Esterel Threads</th>
<th>Preemptions Threads</th>
<th>CKAG</th>
<th>Preemption handled by</th>
<th>Compiling Time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cnt Max Depth</td>
<td>Preemptions Cnt Max Depth</td>
<td>Nodes Dep. Max Priority Instr</td>
<td>Local Thread Watcher Watcher</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ethanol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abc</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>abcd</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>eight</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>chan</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>prot</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>reactor</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>ctrl</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>runner</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>example</td>
<td>17</td>
<td>3</td>
<td>13</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>ww_button</td>
<td>39</td>
<td>5</td>
<td>17</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>greycounter</td>
<td>59</td>
<td>5</td>
<td>49</td>
<td>64</td>
<td>4</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 V5</td>
<td>ACRT V5</td>
<td>WCRT Ratio to ACRT Ratio</td>
</tr>
<tr>
<td></td>
<td>V7</td>
<td>CEC</td>
<td>best MB</td>
</tr>
<tr>
<td></td>
<td>CEC</td>
<td></td>
<td>Unopt</td>
</tr>
<tr>
<td>abcde</td>
<td>1559</td>
<td>954</td>
<td>1476</td>
</tr>
<tr>
<td>abcd</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 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 (word)</th>
<th>KEP3a-opt. Code (word)</th>
</tr>
</thead>
<tbody>
<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
## Power Consumption

<table>
<thead>
<tr>
<th>Module Name</th>
<th>MicroBlaze (82mW@50MHz)</th>
<th>KEP3a $^3$ (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

$^3$Based 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)

---

4 For comparison, a MicroBlaze implementation requires around 1k slices and 309k gates; a two threads EMPEROR platform requires around 2k slices.
### Analysis of Context Switches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>2</td>
<td>26</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
    loop
      p
    end loop
  end every
  end signal
end module
```

```
  every S do
    loop
      abort
      p;
      halt
    when S
    end loop
  end S
  loop
    emit S;
    pause;
  end loop

  loop
    p
  end loop

  A:
    p;
    goto A
```

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

  sustain S
  loop
    emit S;
    pause;
  end loop

  loop
    p
  end loop
```

```
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: TWABORT 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

```verilog
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 loop
end module
```

```verilog
loop
  p
end loop
≡
await S;
loop
  abort
  p;
  halt
  when S
  end loop
```

```verilog
sustain S
≡
loop
  emit S;
  pause;
  end loop
```

```verilog
loop
  p
end loop
≡
A:
  p;
  goto A
```

```
Tick S I
----------
R R

A
O
```
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

  loop
    pause;
  pause;
  present R then
    emit A;
  end present
  end loop
end signal
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, T1] 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

- Tick 1 -
! reset;
% In:
% Out:
[L01, T0] [L02, T0] [L03, T0]
- Tick 2 -
% In: S
% Out:
[L04, T0] [L05, T0] [L06, T0] [L07, T0]
[L08, T1] [L09, T1] [L10, T1] [L11, T1] [L12, T1]
[L13, T1] [L14, T1] [L15, T1] [L16, T1] [L17, T1] [L18, T1] [L19, T1]
[L20, T2] [L21, T2] [L22, T2] [L23, T2] [L24, T2] [L25, T2] [L26, T2] [L27, T0]
- Tick 3 -
% In: I
% Out: R
[L10, T1] [L11, T1] [L12, T1] [L13, T1] [L14, T1] [L16, T1] [L17, T1] [L18, T1] [L19, T1] [L20, T2] [L21, T2] [L22, T2] [L23, T2] [L24, T2] [L25, T2] [L26, T2] [L27, T0]
- Tick 4 -
% In: A R O
[L17, T1] [L18, T1] [L19, T1] [L20, T2] [L21, T2] [L22, T2] [L23, T2] [L24, T2] [L25, T2] [L26, T2] [L27, T0]
Overview

KEP3a Instruction Set + Architecture

The Compiler

Further Measurements

Summary

Multi-processing vs. Multi-threading
Comparison of Synthesis Options
Application Scenarios
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
- \texttt{sync} command to synchronize threads

Multi-threading (KEP)

- Esterel thread $\approx$ several registers
- priority-based scheduler
- \texttt{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
Scenario II: DSP + HW Block + Reactive Processor

\[\text{HW blk} \rightarrow \text{Communication Backplane} \rightarrow \text{DSP} \rightarrow \text{Reactive Processor} \rightarrow \text{Global Memory} \rightarrow \text{IPs}\]
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

Thanks/Comments?