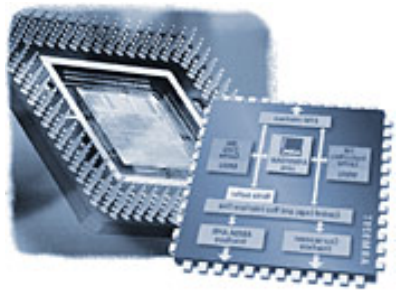
A Concurrent Reactive Esterel Processor Based on Multi-Threading

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

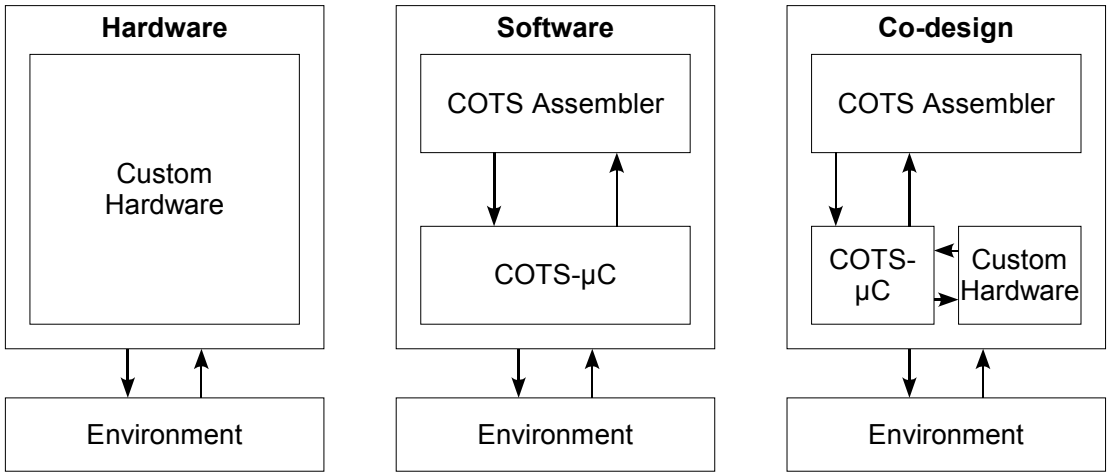
Overview

Overview

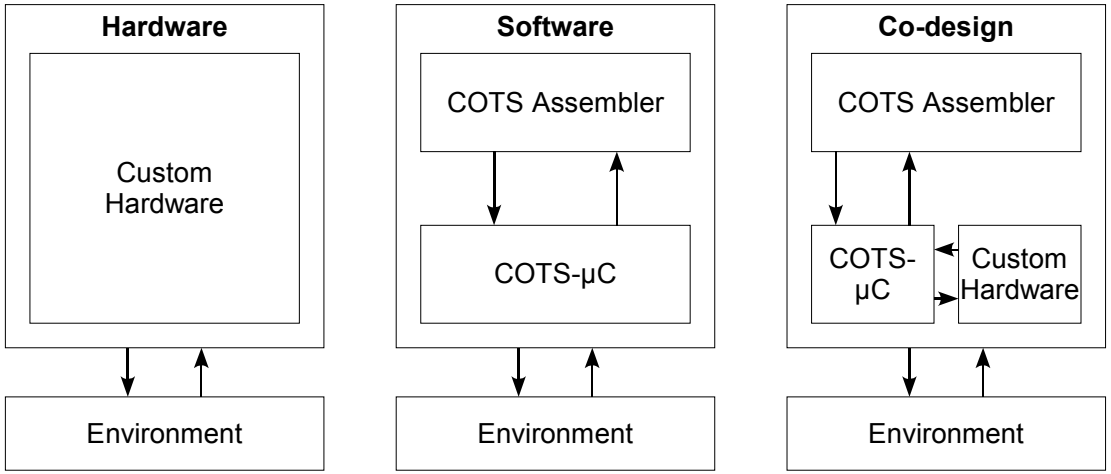
The Synchronous Language Esterel

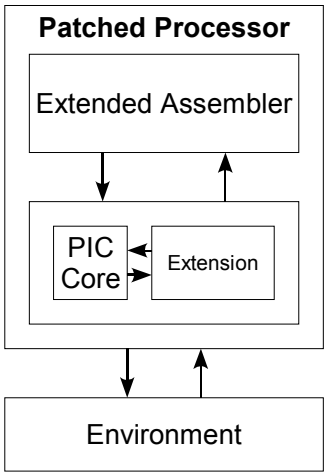
- Created in the early 1980's
- Describes behavior of reactive systems
- Concurrency + numerous forms of preemption
- Deterministic behavior, clean semantics

Esterel Synthesis Options

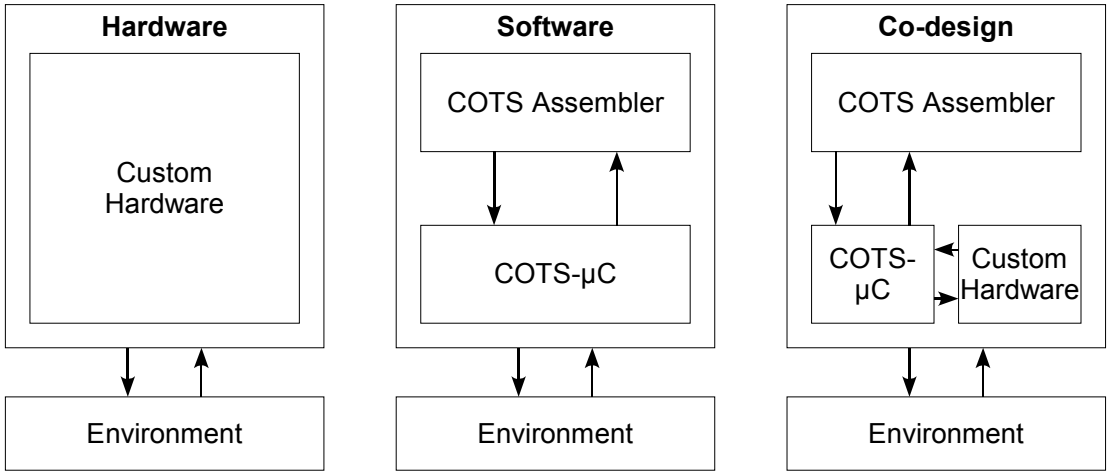


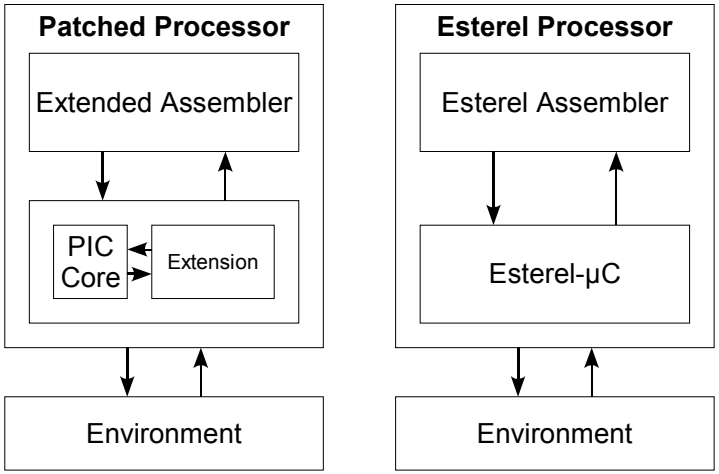
Esterel Synthesis Options



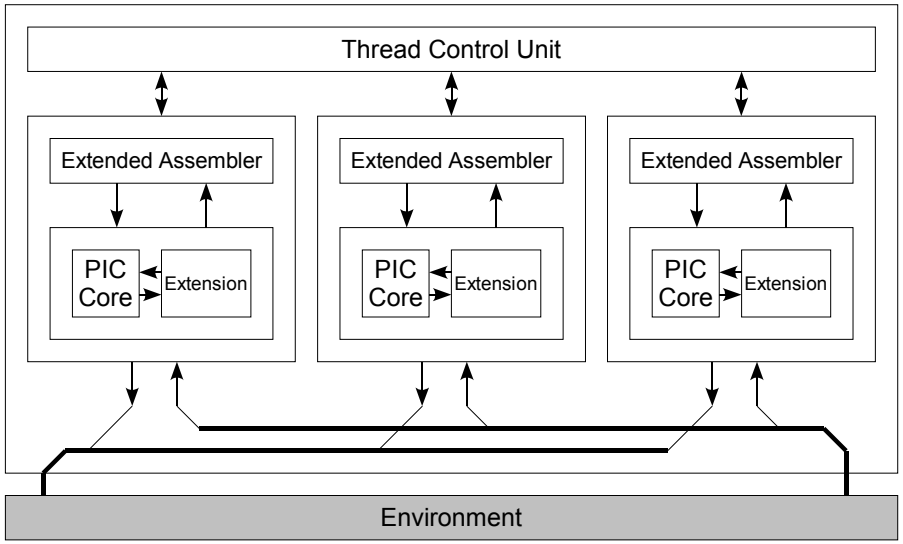


Esterel Synthesis Options

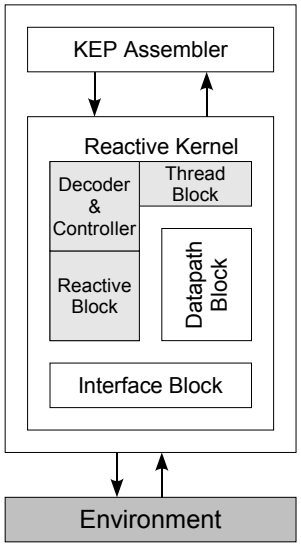




Multi-processing vs. Multi-threading



Multi-processing (EMPEROR/RePIC)



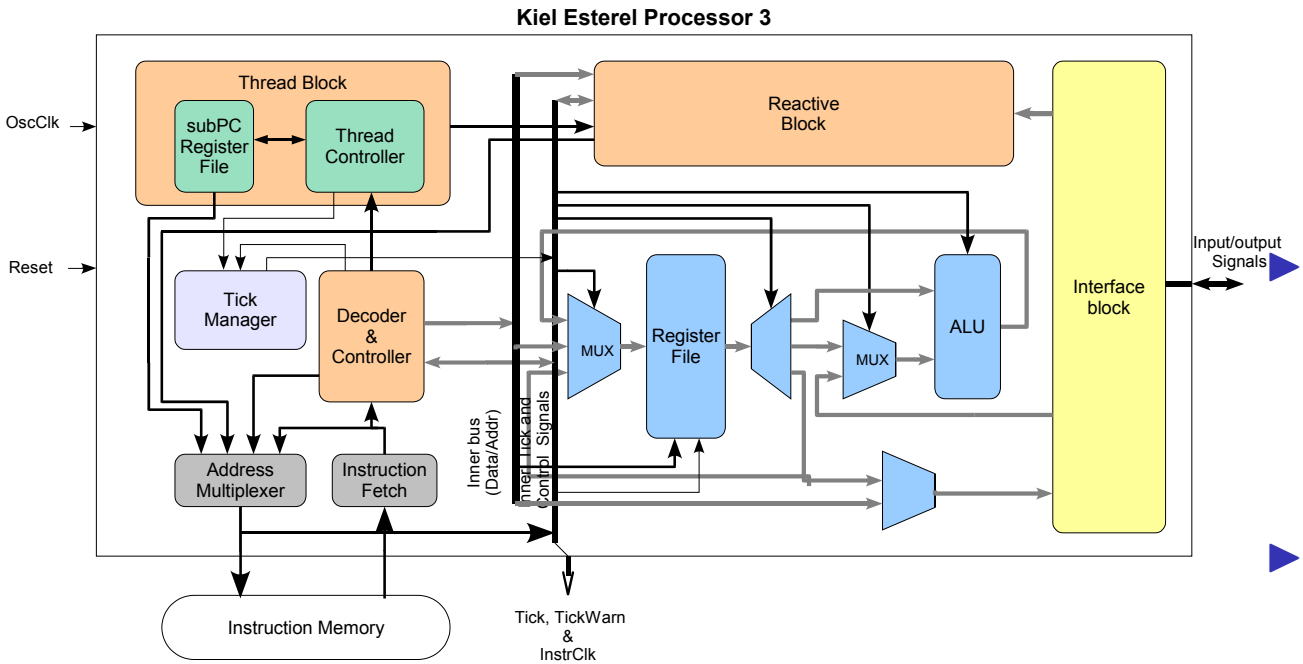
Multi-threading (KEP3)

- an Esterel thread ≈ one independent RePIC processor
- Thread Control Unit handles the synchronization and communication
- `sync` command to synchronize threads

- an Esterel thread ≈ several registers
- priority-based scheduler
- `PRIO` command to schedule threads

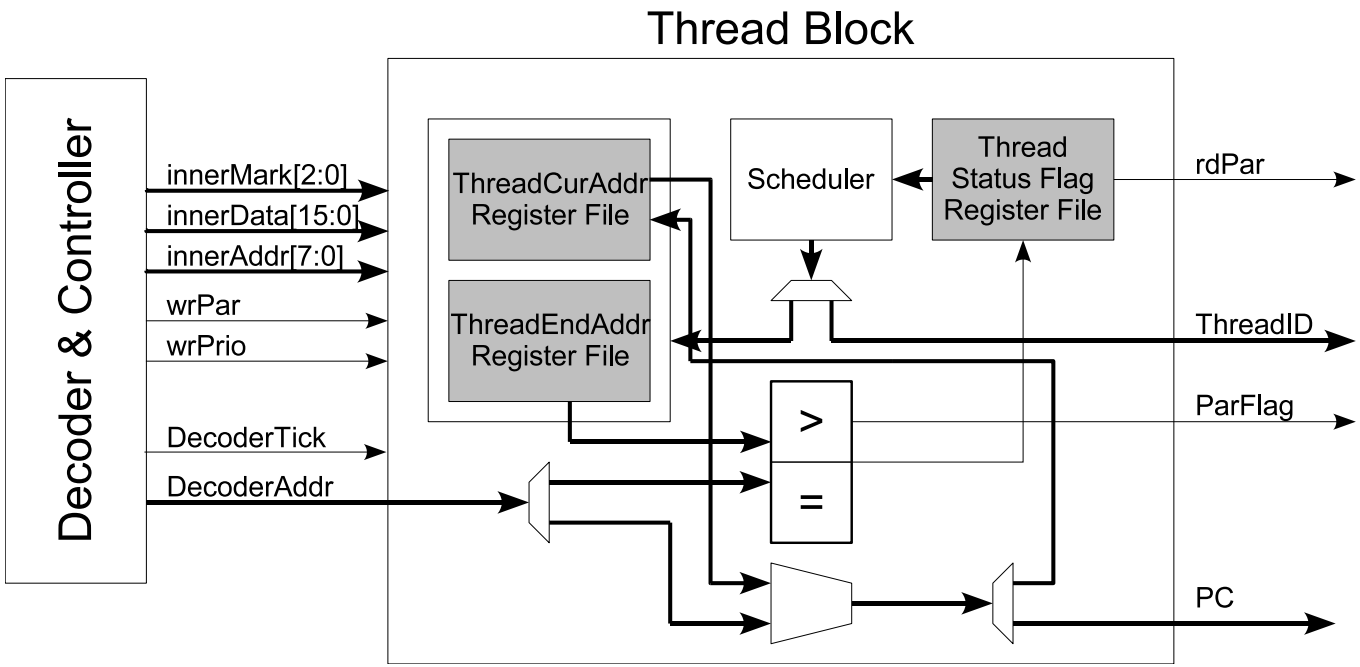
Overview

Kiel Esterel Processor Architecture



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

Handling Concurrency

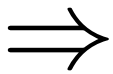


Handling Concurrency

```
[
  await A
‘\alert{||}’
  await B
];
emit O;
```

Handling Concurrency

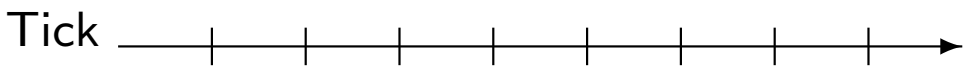
```
[
  await A
‘\alert{||}’
  await B
];
emit O;
```



```
    PAR 1,P1
    PAR 1,P2
    PARE P3
P1: AWAIT A
P2: AWAIT B
P3: JOIN
    EMIT O
```

An Example

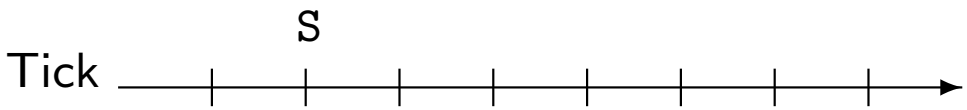
```
% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
  [ await I;
    weak abort
      sustain R;
    when immediate A;
    emit O;
  ||
    loop
      pause;
      pause;
      present R then
        emit A;
      end
    end loop ]
end every
end signal
end module
```



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
```

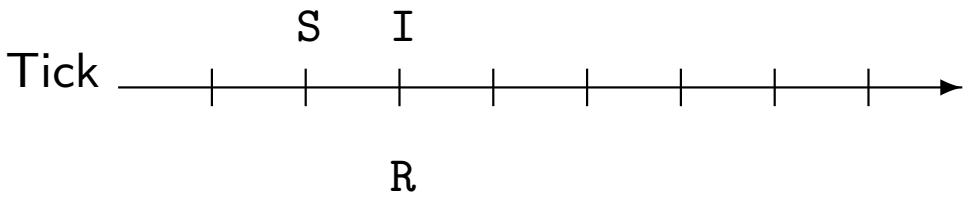


Tick 0: main thread waits for signal S

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

An Example

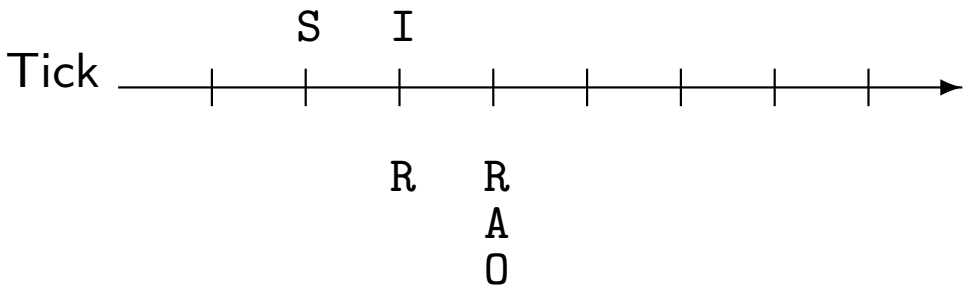
```
% Esterel
module EXAMPLE:
input S,I;
output O;
signal A,R in
every S do
  [ await I;
    weak abort
      sustain R;
    when immediate A;
    emit O;
  ||
    loop
      pause;
      pause;
      present R then
        emit A;
      end
    end loop ]
end every
end signal
end module
```



- 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 0: main thread waits for signal S

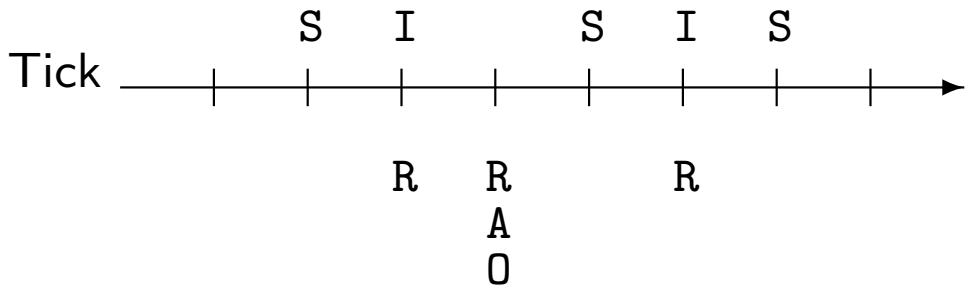
Tick 1: S is present; 1st thread waits for I, 2nd thread pauses

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

```
% 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
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
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
```

```
every S do
  p
end
```

≡

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

```
sustain '$$$'
```

≡

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

```
loop
  p
end loop
```

≡

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

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
```

```
every S do
  p
end
```

≡

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

```
sustain '$$$'
```

≡

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

```
loop
  p
end loop
```

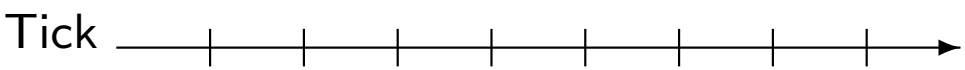
≡

```
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
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 2
[18] P3:  JOIN          % Join
[19]      HALT
[20] A4:  GOTO A0
```

An Example

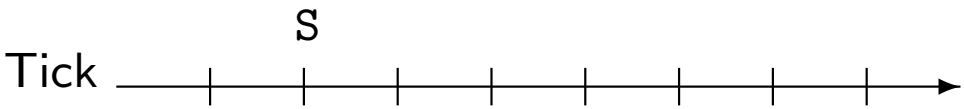
```
          % KEP3 ASM
          INPUT S,I
          OUTPUT O
          SIGNAL A,R
[00]      AWAIT S
[01] A0:  ABORT S,A4
[02]      PAR 3,P1      % Fork
[03]      PAR 2,P2
[04]      PARE P3       % Prios:
[05] P1:  AWAIT I       % 3
[06]      WABORTI A,A2  % 3
[07] A1:  EMIT R        % 3
[08]      PRIO 1        % 1
[09]      PRIO 3        % 3
[10]      PAUSE         % 3
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 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
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 2
[18] P3:  JOIN          % Join
[19]      HALT
[20] A4:  GOTO A0
```

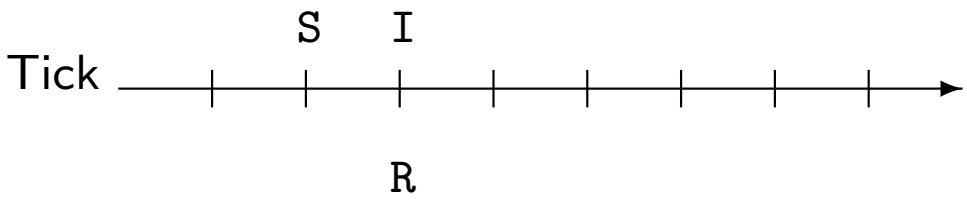


Tick 0: main thread stays at [00]

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
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 2
[18] P3:  JOIN          % Join
[19]      HALT
[20] A4:  GOTO A0
```



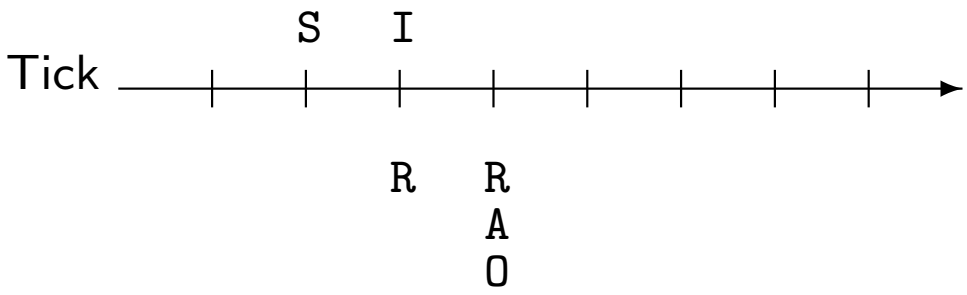
Tick 0: main thread stays at [00]

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
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 2
[18] P3:  JOIN          % Join
[19]      HALT
[20] A4:  GOTO A0
```



Tick 0: main thread stays at [00]

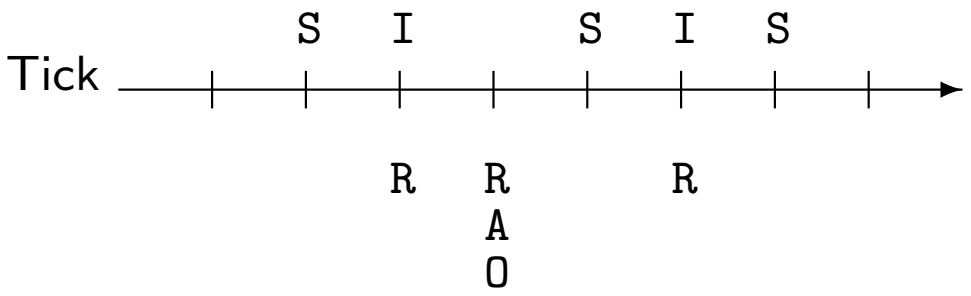
Tick 1: main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]

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]

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
[11]      GOTO A1       % 3
[12] A2:  EMIT O        % 3
[13] P2:  PAUSE         % 2
[14]      PAUSE         % 2
[15]      PRESENT R,A3  % 2
[16]      EMIT A        % 2
[17] A3:  GOTO P2       % 2
[18] P3:  JOIN          % Join
[19]      HALT
[20] A4:  GOTO A0
```



Tick 0: main thread stays at [00]

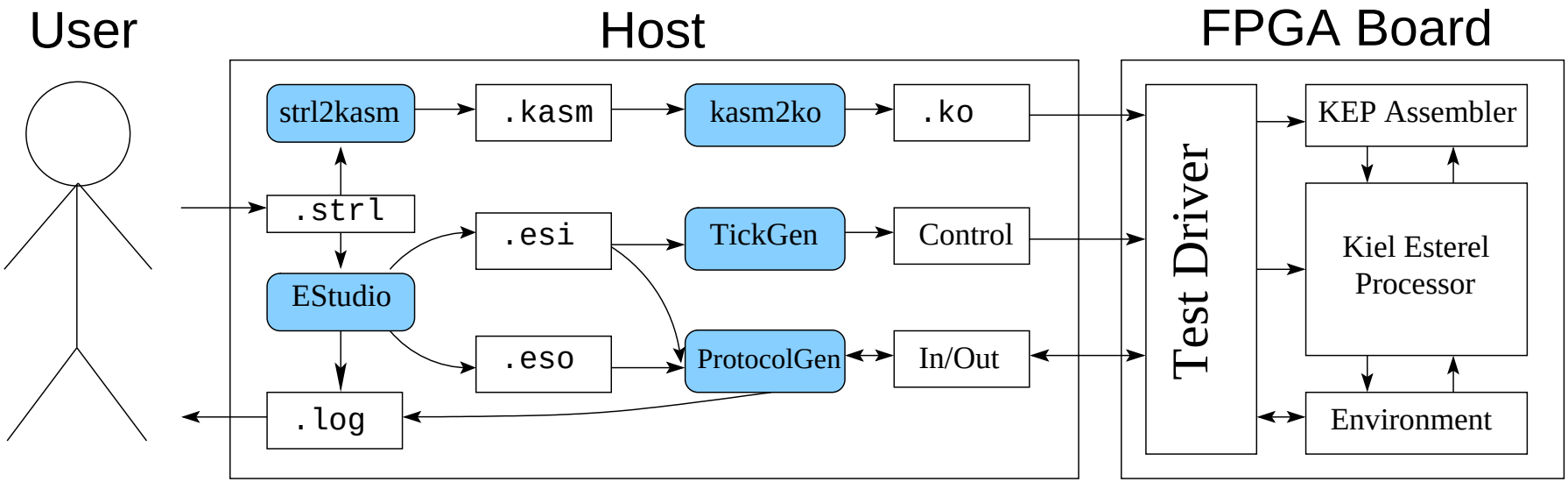
Tick 1: main thread stays at [18], 1st thread stays at [05], 2nd thread stays at [13]

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

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

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

Scalability

	KEP3								EMPEROR2
Thread Cnt	2	4	8	16	32	64	102	126	2
Logic Cells	2086	2170	2306	2466	2946	3758	4768	5564	4761
Max Osc Freq [MHz]	42.68	42.68	42.68	42.51	42.68	42.68	40.26	40.26	35.38

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

Code and RAM Sizes

Module	Thread Cnt/ Preemption nesting depth	Code Size (in words)				RAM Usage (in bytes)			
		Microblaze			KEP	Microblaze			KEP
		V5	V7	CEC		V5	V7	CEC	
BAT_DIAG	1/1	487	1274	378	63	80	76	56	16
VER_ACC_DIAG	1/1	433	1229	303	41	72	68	52	12
BELT	2/3	617	1255	483	31	84	84	52	14
ABCD	4/1	1357	1547	1396	107	112	112	504	24
RUNNER	2/5	688	1323	608	37	88	84	60	14
ARBITER12	36/1	3162	1703	3909	317	256	172	88	156
LONG_SPEED_STRAT	1/1	573	1306	319	56	80	72	52	12

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

Summary

- 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-Codesign—implement compound logical expressions in customizable HW block

Outlook

KEP

- Implementation in Esterel!

Compiler

- Optimize priority assignments

Both

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

Thanks!
Questions/Comments?

Appendix

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

Mnemonic, Operands	Esterel Syntax	Notes
PAR *Prio*, *startAddr* [, *ID*] PARE *endAddr* JOIN	[*p* || *q*]	Fork and join. An optional *ID* explicitly specifies the ID of the created thread.
PRIO *Prio*		Set the priority of the current thread
[W]ABORT [*n*,] *S*, *endAddr*	[weak] abort ... when [*n*] *S*	*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
[W]ABORT[I] *S*, *endAddr*	[weak] abort ... when [immediate] *S*	
SUSPEND[I] *S*, *endAddr*	suspend ... when [immediate] *S*	
PAUSE	pause	Wait for a signal. AWAIT TICK is equivalent to PAUSE
AWAIT [*n*,] *S*	await [*n*] *S*	
AWAIT[I] *S*	await [immediate] *S*	

Instruction Set 2/2

Mnemonic, Operands	Esterel Syntax	Notes
SIGNAL *S*	signal *S* in ...end	Initialize a local signal *S*
EMIT *S* [, {#*data*|*reg*}]	emit *S* [(*val*)]	Emit (valued) signal *S*
SUSTAIN *S* [, {#*data*|*reg*}]	sustain *S* [(*val*)]	Sustain (valued) signal *S*
PRESENT *S*, *elseAddr*	present *S* then ...end	Jump to *elseAddr* if *S* is absent
NOTHING	nothing	Do nothing
HALT	halt	Halt the program
GOTO *addr*	loop ...end loop	Jump to *addr*
CLRC/SETC		Clear/set carry bit
LOAD *reg*, *n*		Load/store register
{SR|SRC|SL|SLC|NOTR} *reg*		Shift/negate
{ADD[C]|SUB[C]|MUL} *reg*, *n*		Add, subtract (with carry), multiply
{ANDR|ORR|XORR} *reg*, *n*		Logical operations
CMP *reg*, *n* JW *cond*, *addr*		Compare, branch conditionally.

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 `PRIO` instruction
  - 255 priority levels

Handling Concurrency

A thread has its

- independent PC
- address range
- priority value
  - assigned when a thread is created
  - dynamically changed via `PRIO` 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 `PRIO` 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

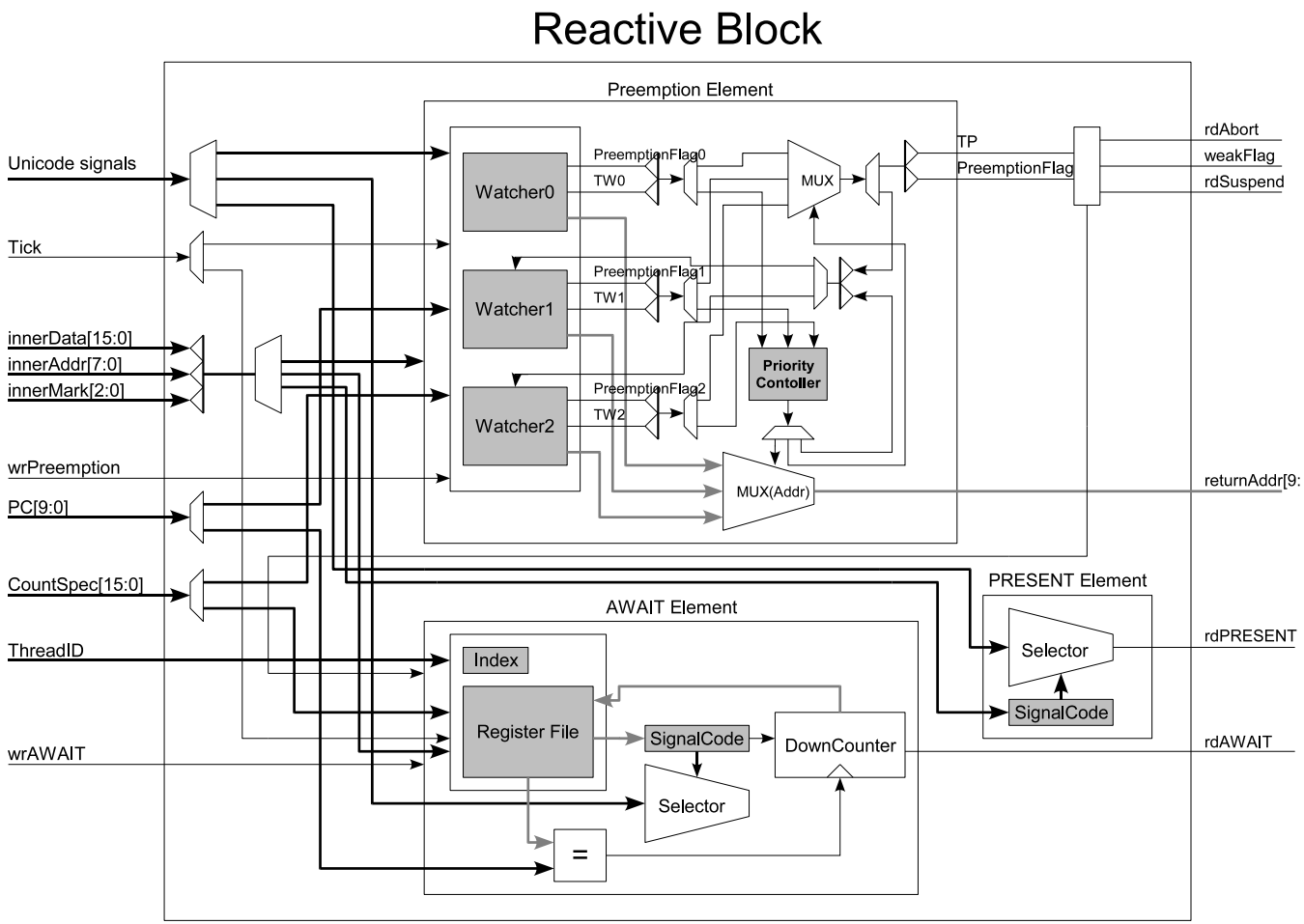
- all *enabled* threads are *active* in a new tick
- an active sub thread blocks its parent thread
- the highest priority/earliest generated thread gets to execute
- a thread is *inactive* in the current tick when a non-instantaneous instruction is executed
- a tick is finished when all of threads are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

Handling Concurrency

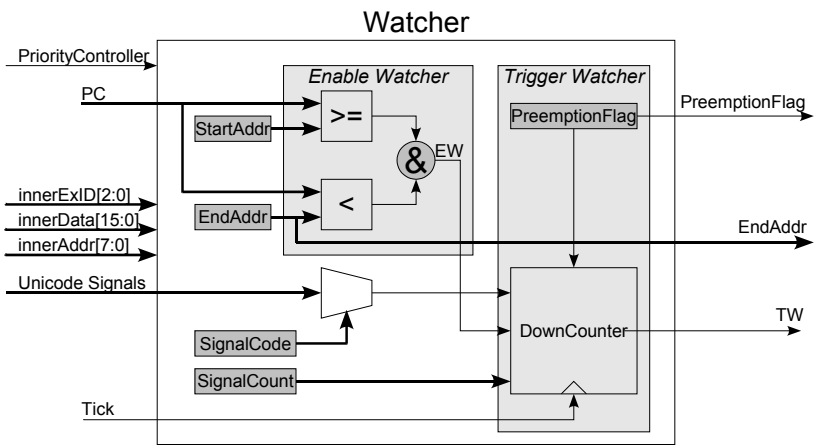
Terminating sub threads

- 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

	Hardware	Software	Co-design	Patched Processor	Esterel Processor
Speed	++	−	+	+	+
Flexibility	− −	++	−	+/−	+
Esterel Compliance	++	++	+/−	+	++
Cost	+	− −	−	−	+
Appl. Design Cycle	− −	++	+/−	++	++

Note: ++ = best; − − = worst.

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