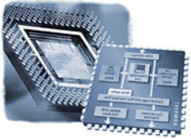
An Esterel Processor with Full Preemption Support and its Worst Case Reaction Time Analysis

Reinhard v. Hanxleden

Joint work with Xin Li, Jan Lukoschus, Marian Boldt, Michael Harder

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/`

CASES 2005, San Francisco



The Synchronous Language Esterel

- Created in the early 1980's
- Describes behavior of reactive systems
- Concurrency + numerous forms of preemption
- Deterministic behavior, clean semantics
- Time is divided into discrete instances

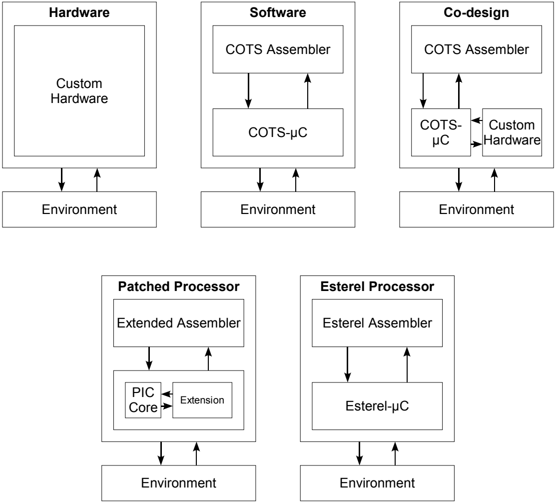
Conceptually: Reactions are instantaneous (Synchrony hypothesis)

In reality: Computation of reactions does take time—bounded by Worst Case Reaction Time (WCRT)

☺ Esterel semantics bounds computation effort per reaction

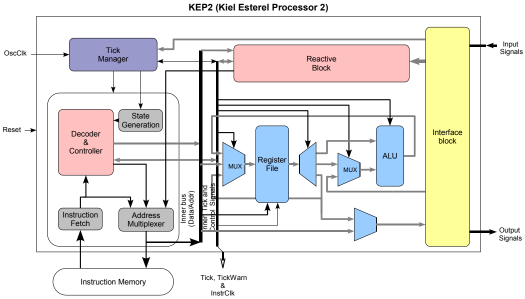
☹ Typical synthesis path re-introduces timing uncertainties

Esterel Synthesis Options



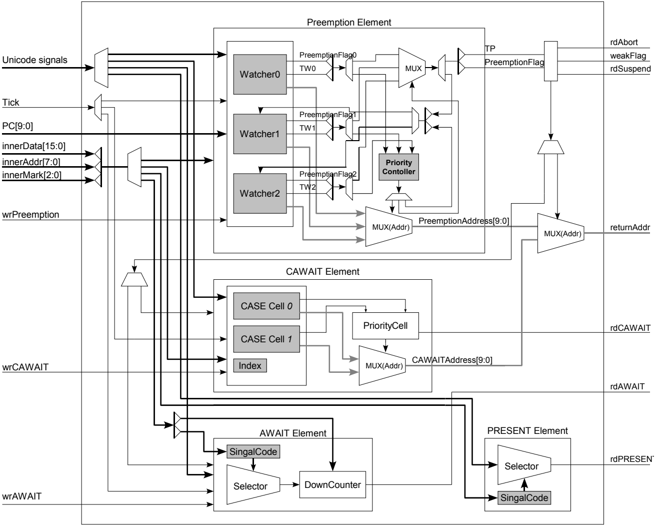
Overview

Kiel Esterel Processor Architecture

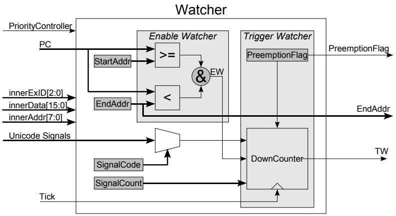


- Reactive Core
  - Decoder & Controller
  - Reactive Block
- Interface Block
  - Interface signals
  - Local signals
  - ...
- Data Handling
  - Register file
  - ALU
  - ...

Reactive Block



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?

KEP Assembler—An Example

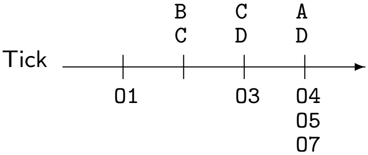
```
% Esterel
weak abort
  suspend
    abort
      emit O1;
      await D;
      emit O2;
    when C;
    emit O3;
    await D;
    emit O4;
  when B;
  emit O5;
  await D;
  emit O6;
when A;
emit O7;
halt;
```

⇒

```
    % KEP2 ASM   (Addresses)
    WABORT 1,A,A2      (0)(1)
      SUSPEND 1,B,A1   (2)(3)
        ABORT 1,C,A0   (4)(5)
          EMIT O1         (6)
          AWAIT D         (7)
          EMIT O2         (8)
A0:
        EMIT O3           (9)
        AWAIT D          (10)
        EMIT O4          (11)
A1:
      EMIT O5            (12)
      AWAIT D            (13)
      EMIT O6            (14)
A2:
    EMIT O7              (15)
    HALT                 (16)
```

Inside/Outside Preemption Range Watching

```
    % KEP2 ASM    (Addresses)
    WABORT 1,A,A2      (0)(1)
      SUSPEND 1,B,A1   (2)(3)
        ABORT 1,C,A0   (4)(5)
          EMIT O1         (6)
          AWAIT D         (7)
          EMIT O2         (8)
A0:
        EMIT O3           (9)
        AWAIT D          (10)
        EMIT O4          (11)
A1:
      EMIT O5            (12)
      AWAIT D            (13)
      EMIT O6            (14)
A2:
    EMIT O7              (15)
    HALT                 (16)
```



- Tick 0: Configure Watchers, emit O1 (Instruction 6), wait for D (7)
- Tick 1: Suspension B takes priority over abortion C; freeze at (7)
- Tick 2: Strong abortion C takes priority over awaiting D; jump to (9), continue to (10)
- Tick 3: Weak abortion A triggered; execute (10–12), fetch and abort (13), jump to (15), continue to (16)

Comparison of KEP with RePIC [Roop+EMSOFT04]

```
% Esterel
...
weak abort
  ...
  emit Z;
  await B;
  emit X;
  await C;
  emit W;
when A;
emit Y;
...
```

```
% RePIC ASM
...
ldaaddr A0
abort 0 A
...
emit Z
chkabort 0
$0: await
present B
goto $0
emit X
chkabort 0
$1: await
present C
goto $1
emit W
A0: emit Y
...
```

```
% KEP ASM
...
WABORT 1,A,A0
  ...
  EMIT Z
  AWAIT B
  EMIT X
  AWAIT C
  EMIT W
A0:
EMIT Y
...
```

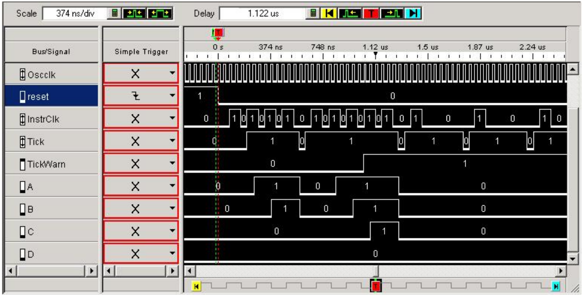
KEP is Configurable—and Efficient!

	KEP2-A	KEP2-B	KEP2-C	KEP2-D	KEP2-E	RePIC
AWAIT Cases	2	2	2	2	2	2
Preemption Nesting	2	2	2	4	4	4
Counter Value Range	1	255	1	255	1	1
Input/Output Signals	11/11	16/16	11/11	16/16	12/12	12/12
Valued Input/Output Signals	2/2	2/2	2/2	2/2	1/1	1/1
Datapath Width	8	8	16	16	8	8
Logic Cell Cnt	1092	1270	1384	1972	1488	2068
Max Osc Freq(MHz)	54.11	47.93	44.97	41.46	42.87	40.27
Instruction Freq(MHz)	18.04	15.93	14.99	13.82	14.29	10.1

WCRT (Tick Length) Self-Monitoring

- OscClk: external clock; InstrClk: instructions; Tick: logical ticks
- Emitting special signal `_TICKLEN` configures Tick Manager with WCRT
- TickWarn pin indicates WCRT timing violation

```
% KEP Assembler
% module OVERRUN
INPUT D
OUTPUT A,B,C

EMIT _TICKLEN, #3
EMIT A
EMIT B
PAUSE
EMIT A
EMIT B
EMIT C
AWAIT D
```



Overview

Problem Statement

Want to transform Esterel program into KEP Assembler,

WCRT information

```
% Esterel
module ABRT:

input A;
output S, T;

abort
  emit S;
  halt;
  emit T
when A

end module
```

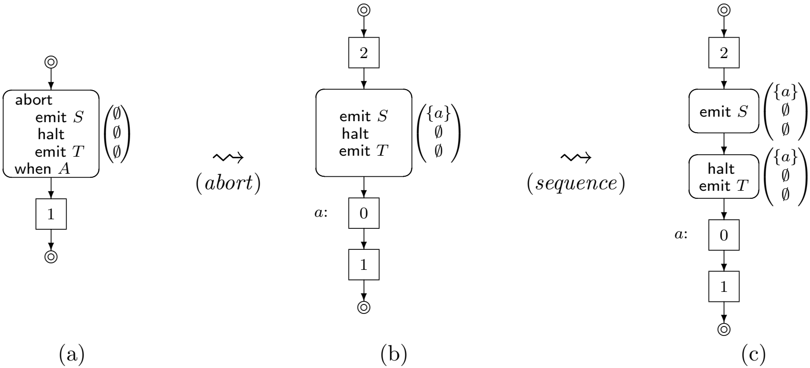
$\Longrightarrow$

```
    % KEP Assembler
    % module ABRT
    INPUT A;
    OUTPUT S, T;

    EMIT _TICKLEN, #4
                           % T0
    ABORT 1, A, A0         % N1.2
      EMIT S               % N1.3
      HALT                 % N1.4, T2
      EMIT T
A0:                        % N3.0
    HALT                   % N3.1, T4
```

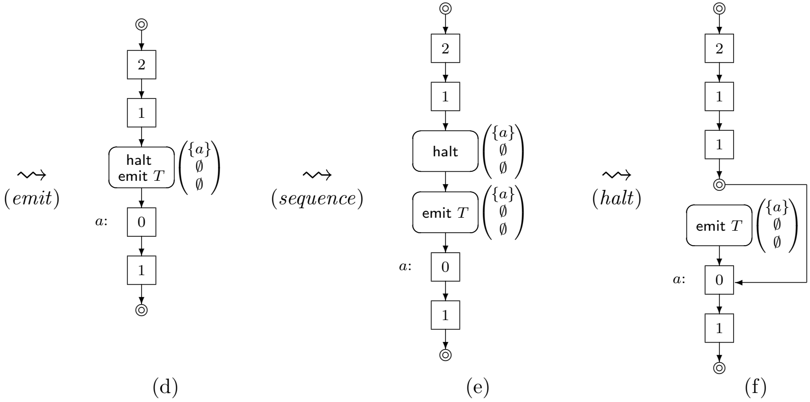
Overview of WCRT Analysis

- Implementation as add-on to Columbia Esterel Compiler (CEC)
- Three basic steps:
  1. Construct KEP Assembler Graph (KAG)
  2. Determine longest path within reaction
  3. Bound concrete reaction times

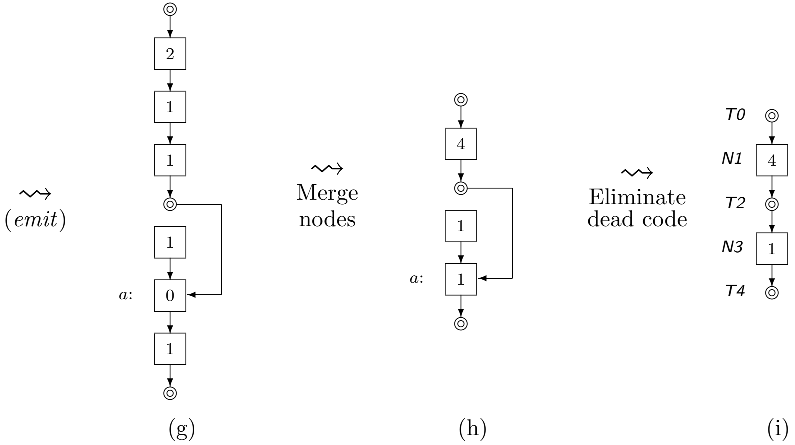
Example of KAG Construction (1/3)



Example of KAG Construction (2/3)



Example of KAG Construction (3/3)



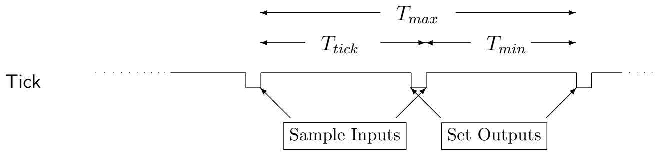
Determining the Longest Path

```
int max_instant_len(Nodes N) {

  forall n ∈ N do {
    n.visited := false
    n.len := ⊥
  }
  T := {n ∈ N | n.terminal}

  return max_{n∈T} get_len(n)
}
```

```
int get_len(Node n) {
  if (n.len = ⊥) {
    if (n.visited) {
      n.len := ∞ // Instantaneous loop!
    }
    else {
      n.visited := true
      n.len := n.cost + max_{s∈(n.succs\T)} get_len(s)
    }
  }
  return n.len
}
```

Complexitiy $\mathcal{O}(|N| + |E|)$—which is optimal!

Concrete Reaction Times



$V_{ticklen}$: Instruction count determined by WCRT analysis

T_{osc}: Basic clock rate of processor

$$T_{min} \leq T_{react} < T_{max} \quad (1)$$
$$T_{min} = (3V_{ticklen} + 1) * T_{osc} \quad (2)$$
$$T_{tick} = T_{min} + T_{osc} \quad (3)$$
$$T_{max} = T_{min} + T_{tick} \quad (= (6V_{ticklen} + 3) * T_{osc}) \quad (4)$$

Example: $V_{ticklen} = 5$, $T_{osc} = 41.67ns$ (24 MHz)
$\Rightarrow 666.72ns \leq T_{react} < 1375.11ns$

Summary

- The Kiel Esterel Processor
  - ... completely and accurately implements Esterel preemption primitives
  - ... is more efficient than patched processor approach
  - ... implements data handling (valued signals) and `pre` operator
  - ... and has predictable processing times, taking advantage of Esterel's predictable control flow!
- Have developed WCRT analysis algorithm of linear complexity
- Can synthesize self-checking KEP code with WCRT annotations

Outlook

KEP

- Concurrent KEP with dynamically interleaved execution (KEP 3)
- Other extensions, *e. g.* direct handling of `immediate` triggers
- Implementation in Esterel!

WCRT Analysis

- Further tighten results, *e. g.* by analysing possible configurations/signal statusses to prune paths (model checking)
- Use WCRT analysis to guide compiler optimizations!

Note: *Figures 3–6 in printed proceedings are broken—please download electronic version*

Thanks! Questions/Comments?

Part I

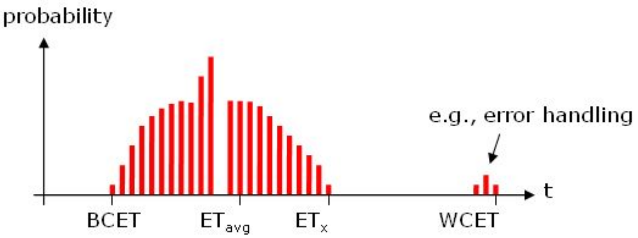
Appendix

Worst-Case Reaction Time Analysis

Worst Case Execution Time (WCET): Considers single application
[Li/Malik/Wolfe 1995 + many others]

Worst Case Response Time (WCRespT): WCET + task interference
[*e. g.*, Tan/Mooney 2005]

Worst Case Reaction Time (WCRT): WCRespT + HW interface



www.wcet.at

Classical Difficulties with WCET Analysis

- Unpredictable control flow
  - Unbounded number of loops
  - Interrupts
- Difficult to predict hardware
  - Caches
  - Pipelining
- *... however, this picture changes for synchronous programming on a reactive processor!*

Instruction Set Overview

- Initializtion of preemption blocks takes two cycles
- All other instructions take only one cycle
- 32-bit instruction word
- 16-bit inner data bus
- 10-bit instruction memory address bus
- 30 instructions
- Replace majority of Esterel statements directly
- Syntax translation for remaining statements
- ☹ No || operator (yet)
- Approaches to fix this:
  - Multi-processor architecture
  - Sequentialize concurrent Esterel programs into KEP2-Assembler
  - Multi-threaded architecture (KEP3)

Instruction Set 1/4

Mnemonic, Operands	Cycles	Corresponding Esterel Statement
ABORT n,S,endAddr(,startaddr)	2	abort...when n S
WABORT n,S,endAddr(,startaddr)	2	weak abort...when n S
SUSPEND 1,S,endAddr(,startaddr)	2	suspend...when S
AWAIT S	1	await S
AWAIT n,S	1	await n S
PAUSE	1	await Tick/Pause
AWAIT n,Tick	1	await n Tick
CAWAIT S_n,$S_n$startaddr	1	await case
CAWAITE S_n,$S_n$startaddr	1	await case
EMIT S	1	emit S
EMIT S,#data	1	emit S(data)
EMITR S,reg	1	emit S(var_reg)
SUSTAIN S	1	sustain S
SUSTAIN S,#data	1	sustain S(data)
SUSTAINR S,reg	1	sustain S(var_reg)
HALT	1	halt
NOTHING	1	nothing
PRESENT S,elseaddr	1	present S then .. else .. end present
SIGNAL S	1	signal S in
SIGNAL PRE(S)	1	

Instruction Set 2/4

Mnemonic, Operands	Cycles	Corresponding Esterel Statement
`CALL addr`	1	call *subroutine*
`RET`	1	
`GOTO addr`	1	
`JW Z,elseaddr`	1	
`JW L,elseaddr`	1	
`JW G,elseaddr`	1	
`JW GE,elseaddr`	1	
`JW LE,elseaddr`	1	
`JW EE,elseaddr`	1	
`JW NE,elseaddr`	1	
`CLRC`	1	
`SETC`	1	
`SR reg`	1	
`SRC reg`	1	
`NOTR reg`	1	

Instruction Set 3/4

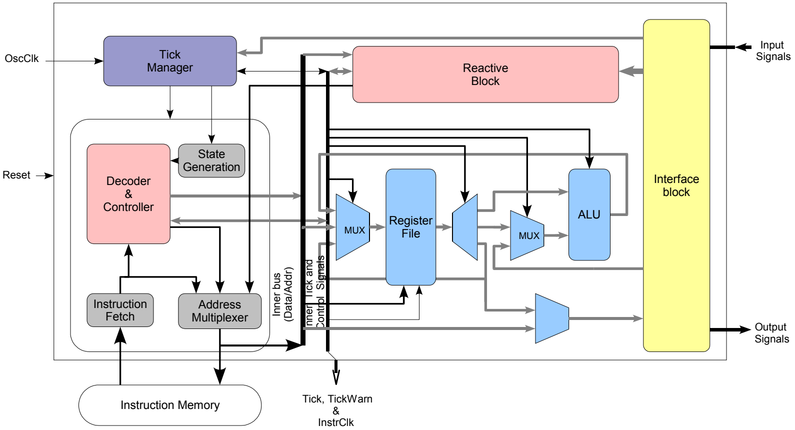
Mnemonic, Operands	Cycles	Corresponding Esterel Statement
LOAD REG,#data	1	var_REG:=data
LOAD REG,reg	1	var_REG:=var_reg
LOAD REG,?S	1	var_REG:=?S
LOAD REG,pre(?S)	1	var_REG:=pre(?S)
ADD REG,#data	1	var_REG:=var_REG + data
ADD REG,reg	1	var_REG:=var_REG + var_reg
ADD REG,?S	1	var_REG:=var_REG + ?S
ADD REG,pre(?S)	1	var_REG:=var_REG + pre(?S)
ADDC REG,#data	1	
ADDC REG,reg	1	
ADDC REG,?S	1	
ADDC REG,pre(?S)	1	
SUB REG,#data	1	var_REG:=var_REG - data
SUB REG,reg	1	var_REG:=var_REG - var_reg
SUB REG,?S	1	var_REG:=var_REG - ?S
SUB REG,pre(?S)	1	var_REG:=var_REG - pre(?S)
SUBC REG,#data	1	
SUBC REG,reg	1	
SUBC REG,?S	1	
SUBC REG,pre(?S)	1	

Instruction Set 4/4

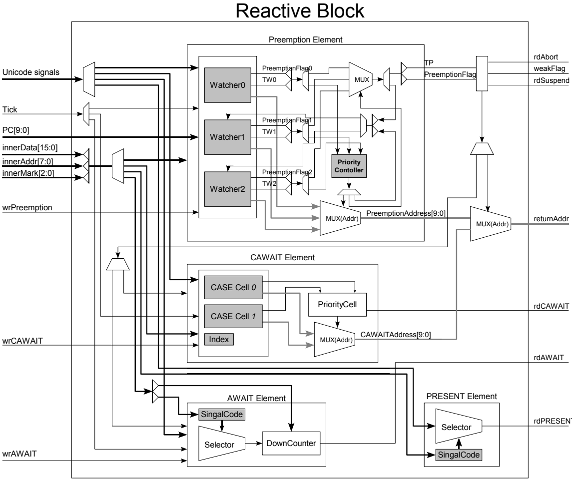
Mnemonic, Operands	Cycles	Corresponding Esterel Statement
MUL REG,#data	1	var_REG:=var_REG * data
MUL REG,reg	1	var_REG:=var_REG * var_reg
MUL REG,?S	1	var_REG:=var_REG * ?S
MUL REG,pre(?S)	1	var_REG:=var_REG * pre(?S)
ANDR REG,#data	1	
ANDR REG,reg	1	
ANDR REG,?S	1	
ANDR REG,pre(?S)	1	
ORR REG,#data	1	
ORR REG,reg	1	
ORR REG,?S	1	
ORR REG,pre(?S)	1	
XORR REG,#data	1	
XORR REG,reg	1	
XORR REG,?S	1	
XORR REG,pre(?S)	1	
CMP REG,#data	1	
CMP REG,reg	1	
CMP REG,?S	1	
CMP REG,pre(?S)	1	

Reactive Core

KEP2 (Kiel Esterel Processor 2)



Other Elements of the Reactive Block



- Preemption Element
  - ABORT
  - WABORT
  - SUSPEND
- AWAIT Element
  - AWAIT
  - PAUSE
- CAWAIT Element
  - CAWAIT
  - CAWAITE
- PRESENT Element
  - PRESENT

Interface Block

The Interface Block . . .

- Emits signals
- Receives and registers signals
- Recodes signals
- Supports Esterel pre operator directly
  - Esterel V5.91 and later version
  - pre(S): The previous status of signal S
  - pre(?S): The value of signal S in the previous instant

Interface Block

```
%KEP2 Assembler preillu
INPUTV A,B
INPUT C,D,E,F
OUTPUTV G,H
OUTPUT I,J,K,L
VAR X
  EMIT _TICKLEN,#5      (1)
  AWAIT A               (2)
  PRESENT C,A0          (3)
    EMIT I              (4)
A0:
  PAUSE                 (5)
  PRESENT PRE(I),A1     (6)
    EMIT G,#25          (7)
    LOAD X,PRE(?A)      (8)
    EMITR H,X           (9)
A1:
  HALT
(10)
```

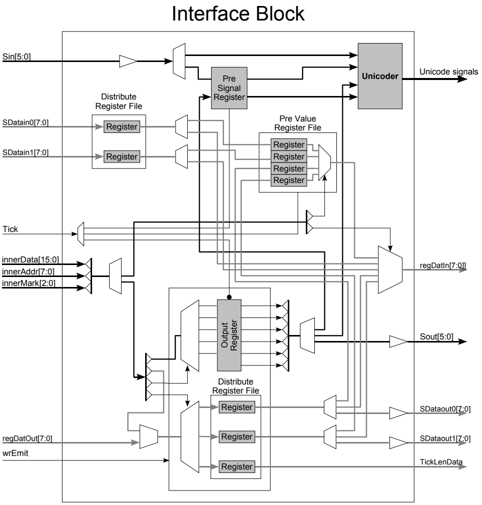
Signal unicode

$d_7 - d_2$	d_1	d_0
Signal code	Input	pre

Unicode

Signal	$d_7 - d_2$	d_1	d_0
_TICKLEN	000000	0	0
A	000001	1	0
B	000010	1	0
C	000011	1	0
G	000001	0	0
H	000010	0	0
I	000011	0	0
PRE(A)	000001	1	1
PRE(I)	000011	0	1

Interface Block




```
    %KEP2 Assembler preillu
    INPUTV A,B
    INPUT C,D,E,F
    OUTPUTV G,H
    OUTPUT I,J,K,L
    VAR X
      EMIT _TICKLEN,#5     (1)
      AWAIT A              (2)
      PRESENT C,A0         (3)
        EMIT I             (4)
    A0:
      PAUSE                (5)
      PRESENT PRE(I),A1    (6)
        EMIT G,#25         (7)
        LOAD X,PRE(?A)     (8)
        EMITR H,X          (9)
    A1:
      HALT
(10)
```

- Initial tick
- Second tick: A and C occur, ?A=20
- Third tick

Compiler

```
kepcmp [-w resource|speed|all] [-s value] [-v
valuedsignalnum] [-d datawidth] -i filename
```

- -w resource/speed/all
  - The Watcher optimization strategy
- -s *value*
  - Maximum amount of input/output signals
- -v *valuedsignalnums*
  - Maximum amount of valued input/output signals
- -i *filename*
  - Assembler language file name

Resource Optimization (Default)

```
%Esterel
module RUNNER
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
every Morning do
  abort
    abort
      sustain Walk;
    when 100 Meter;
    abort
      every Step do
        emit Jump
      end every;
    when 15 Second;
    sustain Run;
  when Lap;
end every;
end module;
```

```
%KEP2 Assembler
%
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
  AWAIT MORNING
A0:
  ABORT 1,MORNING,A3
    ABORT 1,LAP,A3
      ABORT 100,METER,A1
        SUSTAIN WALK
A1:
      ABORT 15,SECOND,A2
        AWAIT STEP
A5:
        ABORT 1,STEP,A4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

Resource Optimization (Default)

```
%KEP2 Assembler
%
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
  AWAIT MORNING
A0:
  ABORT 1,MORNING,A3
    ABORT 1,LAP,A3
      ABORT 100,METER,A1
        SUSTAIN WALK
A1:
      ABORT 15,SECOND,A2
        AWAIT STEP
A5:
        ABORT 1,STEP,A4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

```
%KEP2 Assembler
%Standard (Resource Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
   AWAIT MORNING
A0:
$0:ABORT 1,MORNING,A3,$0
$1: ABORT 1,LAP,A3,$1
$2:   ABORT 100,METER,A1,$2
        SUSTAIN WALK
A1:
$3:   ABORT 15,SECOND,A2,$3
        AWAIT STEP
A5:
$4:     ABORT 1,STEP,A4,$4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

Speed Optimization

```
%KEP2 Assembler
%Standard (Resource Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
   AWAIT MORNING
A0:
$0:ABORT 1,MORNING,A3,$0
$1: ABORT 1,LAP,A3,$1
$2:   ABORT 100,METER,A1,$2
        SUSTAIN WALK
A1:
$3:   ABORT 15,SECOND,A2,$3
        AWAIT STEP
A5:
$4:     ABORT 1,STEP,A4,$4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

```
%KEP2 Assembler
%(Speed Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
ABORT 1,MORNING,A3,$0
ABORT 1,LAP,A3,$1
ABORT 100,METER,A1,$2
ABORT 15,SECOND,A2,$3
ABORT 1,STEP,A4,$4
   AWAIT MORNING
A0: $0: $1: $2:
        SUSTAIN WALK
A1: $3:
        AWAIT STEP
A5: $4:
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

Tradeoff Optimization

```
%KEP2 Assembler
%Standard (Resource Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
   AWAIT MORNING
A0:
$0:ABORT 1,MORNING,A3,$0
$1: ABORT 1,LAP,A3,$1
$2:   ABORT 100,METER,A1,$2
        SUSTAIN WALK
A1:
$3:   ABORT 15,SECOND,A2,$3
        AWAIT STEP
A5:
$4:     ABORT 1,STEP,A4,$4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

- Abortion MORNING: Watcher0
- Abortion LAP: Watcher1
- Abortion METER: Watcher2
- Abortion SECOND: Watcher2
- Abortion STEP: Watcher3

Tradeoff Optimization

```
%KEP2 Assembler
%Standard (Resource Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
   AWAIT MORNING
A0:
$0:ABORT 1,MORNING,A3,$0
$1: ABORT 1,LAP,A3,$1
$2:   ABORT 100,METER,A1,$2
        SUSTAIN WALK
A1:
$3:   ABORT 15,SECOND,A2,$3
        AWAIT STEP
A5:
$4:     ABORT 1,STEP,A4,$4
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

```
%KEP2 Assembler
%(Tradeoff Optimization)
input Morning,Meter,Step
input Second,Lap
output Walk,Jump,Run
ABORT 1,MORNING,A3,$0
ABORT 1,LAP,A3,$1
ABORT 1,STEP,A4,$4
   AWAIT MORNING
A0: $0: $1:
$2:   ABORT 100,METER,A1,$2
        SUSTAIN WALK
A1:
$3:   ABORT 15,SECOND,A2,$3
        AWAIT STEP
A5: $4:
          EMIT JUMP
          HALT
A4:
        GOTO A5
A2:
      SUSTAIN RUN
      HALT
A3:
    GOTO A0
```

Comparison of Watcher Optimized Strategies

RUNNER case

- Resource Optimization (Standard)
  - 9 instruction cycles
  - 4 Watchers
  - The number of Watchers depends on the nest levels
- Speed Optimization
  - 3 instruction cycles
  - 5 Watchers
  - The number of Watchers depends on the preemption instruction numbers
- Tradeoff Optimization
  - 5 instruction cycles
  - 4 Watchers
  - The number of Watchers depends on the nest levels

Comparison of Esterel Synthesis Options

	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.

Comparison of Functions

KEP2	RePIC
Reactive kernel	Traditional microcontroller + patch block
Semi-custom (scalable) elements	Fixed patch block
Supports full Esterel preemption constructs (`abort`/`weak abort`/`suspend`) directly and exactly	Sequence assemblers to implement `abort`/`weak abort` indirectly
Supports valued signal and counter directly (e. g. `emit S(27)`, `abort...when 50 S`)	No such function
Supports pre(S),pre(?S) in architecture	No such function
Esterel optimized datapath, adaptive 8-bit/16-bit wide	Traditional microcontroller datapath, 8-bit wide
Supports multiply instruction	No such function
Predictable tick length	Unforeseen tick length

Comparison of Esterel Direct Execution Ability

Esterel	KEP2	RePIC
abort	√	×
abort (with counter)	√	×
weak abort	√	×
weak abort (with counter)	√	×
suspend	√	×
await	√	√
await (with counter)	√	×
case await	√	√
emit/sustain	√	√
emit/sustain (with carried data)	√	×
present	√	√
halt	√	×

Comparison of Data Handling

Curve (contained in the mca200)

	KEP2 (16-bit)	AT89C55 (8-bit)	Microblaze (32-bit)
Code size (in word)	188	5862	5292
Code size (in byte)	752	5862	21168
RAM Usage	9	37+369	20+116
Instruction cycle	70	2681	499