WCRT Algebra and Interfaces for Esterel-Style Synchronous Processing

Michael Mendler (University of Bamberg)
Reinhard v. Hanxleden, Claus Traulsen (University of Kiel)

DATE 2009
Nice, 21 April 2009

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Outline

Reactive Processing—Why Bother?

- Deterministic behavior
- Predictable timing, simplified WCRT analysis

Reactive Processing—Why Bother?

- Deterministic behavior
- Predictable timing, simplified WCRT analysis
- Low resource requirements per high-level operation
  - fast and compact code
  - minimal power consumption

Reactive Processing—Why Bother?

- Deterministic behavior
- Predictable timing, simplified WCRT analysis
- Low resource requirements per high-level operation
  - fast and compact code
  - minimal power consumption
- Traceability—direct mapping from language semantics to execution platform

Reactive Processing—Approach

Program in reactive programming language. Here, use synchronous language Esterel:

- Deterministic behavior
- Clean mathematical semantics
- Discrete timing model

Reactive Processing—Approach

Program in reactive programming language. Here, use synchronous language Esterel:

- Deterministic behavior
- Clean mathematical semantics
- Discrete timing model

Reactive Instruction Set Architecture directly support reactive control flow:

- Concurrency
- Preemption

Reactive Processing—Approach

Program in reactive programming language. Here, use synchronous language Esterel:

- Deterministic behavior
- Clean mathematical semantics
- Discrete timing model

Reactive Instruction Set Architecture directly support reactive control flow:

- Concurrency
- Preemption

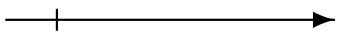
Esterel

```
% Esterel

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

Esterel

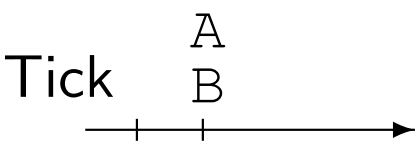
```
% Esterel

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

Tick



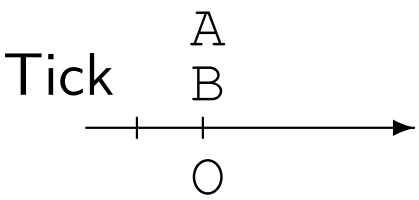
Esterel

```
% Esterel

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



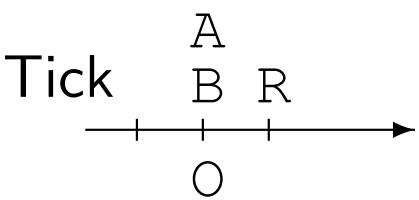
Esterel

```
% Esterel

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



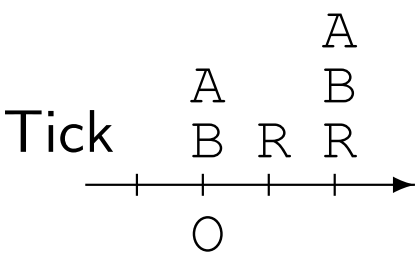
Esterel

```
% Esterel

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



Esterel

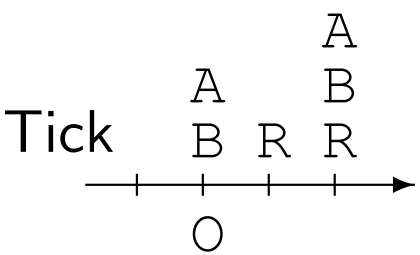
```
% Esterel

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



The Kiel Esterel Processor (KEP)

```
% Esterel

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

```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

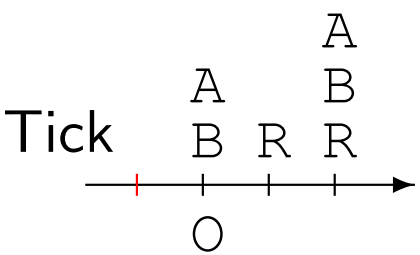


The Kiel Esterel Processor (KEP)

```
% Esterel

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

```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

```
% KEP Execution Trace
 - Tick  1 -
% In:
% Out:
% RT: 7
ABORT L2
PAR L3 PAR L4 PARE L5
AWAIT L9 AWAIT L7
JOIN L11
```

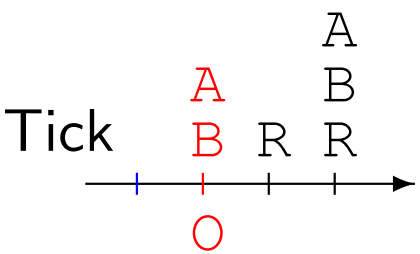


The Kiel Esterel Processor (KEP)

```
% Esterel

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

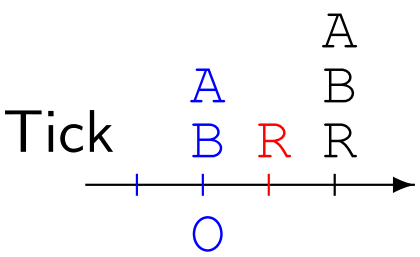
```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

```
% KEP Execution Trace
 - Tick  1 -
% In:
% Out:
% RT: 7
ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  2 -
% In: A B
% Out: O
% RT: 5
AWAIT_L9 AWAIT_L7 JOIN_L11
EMIT_L12 HALT_L13
```



The Kiel Esterel Processor (KEP)

```
% Esterel

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

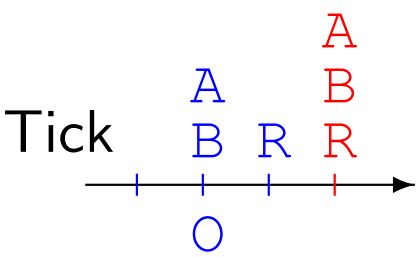



```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

```
% KEP Execution Trace
 - Tick  1 -
% In:
% Out:
% RT: 7
ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  2 -
% In: A B
% Out: O
% RT: 5
AWAIT_L9 AWAIT_L7 JOIN_L11
EMIT_L12 HALT_L13
- Tick  3 -
% In: R
% Out:
% RT: 9
HALT_L13 GOTO_L15 ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
```

The Kiel Esterel Processor (KEP)

```
% Esterel

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

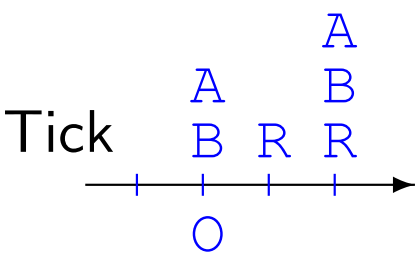



```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

```
% KEP Execution Trace
 - Tick  1 -
% In:
% Out:
% RT: 7
ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  2 -
% In: A B
% Out: O
% RT: 5
AWAIT_L9 AWAIT_L7 JOIN_L11
EMIT_L12 HALT_L13
- Tick  3 -
% In: R
% Out:
% RT: 9
HALT_L13 GOTO_L15 ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  4 -
% In: A B R
% Out:
% RT: 11
AWAIT_L9 AWAIT_L7 JOIN_L11
GOTO_L15 ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7 JOIN_L11
```

The Kiel Esterel Processor (KEP)

```
% Esterel

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




```
% KEP Assembler

INPUT A,B,R
OUTPUT O

[L01] A0:
[L02]      ABORT R,A1
[L03]      PAR 1,A2,1
[L04]      PAR 1,A3,2
[L05]      PARE A4
[L06] A2:
[L07]      AWAIT A
[L08] A3:
[L09]      AWAIT B
[L10] A4:
[L11]      JOIN
[L12]      EMIT O
[L13]      HALT
[L14] A1:
[L15]      GOTO A0
```

```
% KEP Execution Trace
 - Tick  1 -
% In:
% Out:
% RT: 7
ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  2 -
% In: A B
% Out: O
% RT: 5
AWAIT_L9 AWAIT_L7 JOIN_L11
EMIT_L12 HALT_L13
- Tick  3 -
% In: R
% Out:
% RT: 9
HALT_L13 GOTO_L15 ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7
JOIN_L11
- Tick  4 -
% In: A B R
% Out:
% RT: 11
AWAIT_L9 AWAIT_L7 JOIN_L11
GOTO_L15 ABORT_L2
PAR_L3 PAR_L4 PARE_L5
AWAIT_L9 AWAIT_L7 JOIN_L11
```

KEP Instruction Cycles

Esterel Source	KEP Assembler	Cycles
input **output**	INPUT *I* OUTPUT *O*	0
[p_1 || ... || p_n]	PAR ... PAR PARE ... JOIN	$n+1$ 1
emit *S* [(*val*)]	EMIT *S* [, {#*data*|*reg*}]	1
present *S* **then** ... **end**	PRESENT *S*, *elseAddr*	1
[weak] **abort** ... **when** [immediate] *S*	[W]ABORT[I] *S*, *endAddr* ... *endAddr*:	1
pause	PAUSE	1
await [immediate] *S*	AWAIT[I] *S*	1
loop ... end loop	*addr*:... GOTO *addr*	1

Worst Case Reaction Time (WCRT)

- Defined as upper bound for longest instantaneous path
- Measured in KEP instruction cycles
- Maximum time to react to given input with according output

Worst Case Reaction Time (WCRT)

- Defined as upper bound for longest instantaneous path
- Measured in KEP instruction cycles
- Maximum time to react to given input with according output

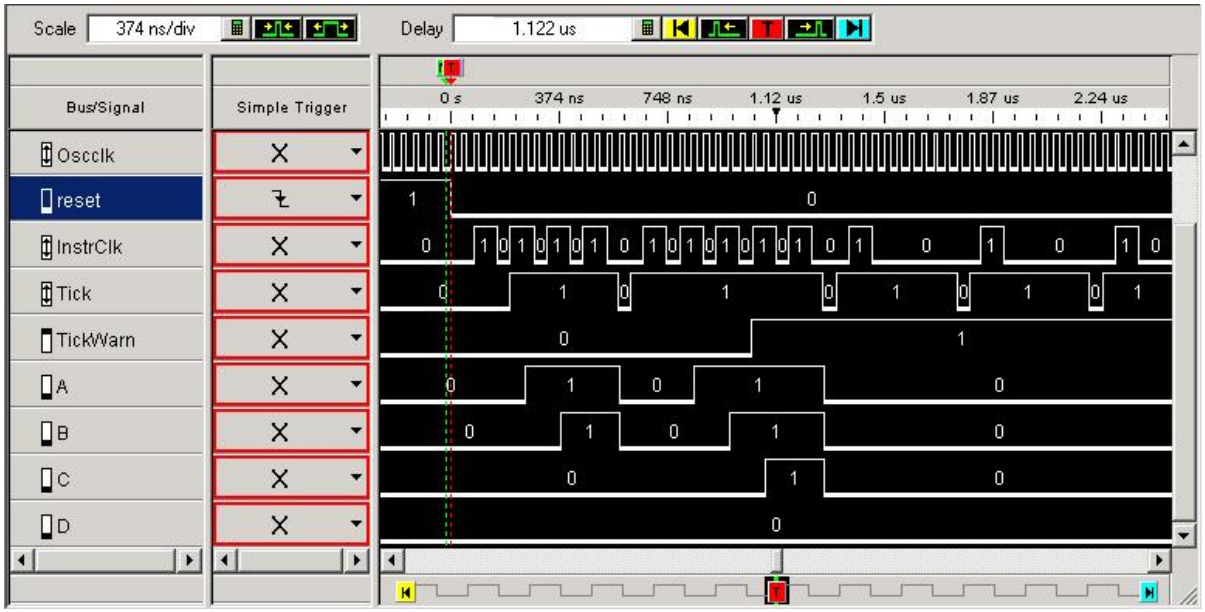
Usage of WCRT:

- Safely determine whether deadlines are met
- Can eliminate reaction time jitter of KEP by setting variable `_TICKLEN` according to WCRT

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
```



Outline

Related Work

- Classical WCET Analyses
  (eg, overview in [Wilhelm *et al.*, ACM TECS '08])
- WCRT analysis for sequential version of KEP
  [Li *et al.*, CASES'05]
- WCRT analysis based on CKAG traversal
  [Boldt *et al.*, ENTCS'08]
- Reactive Processing: eg, *StarPro*
  [Yuan *et al.*, SLA++P'08]
- Precision Timed Architecture
  [Lickly *et al.*, CASES'08]

WCRT vs. WCET

Worst Case Execution Time

- Compute maximal execution time for piece of code

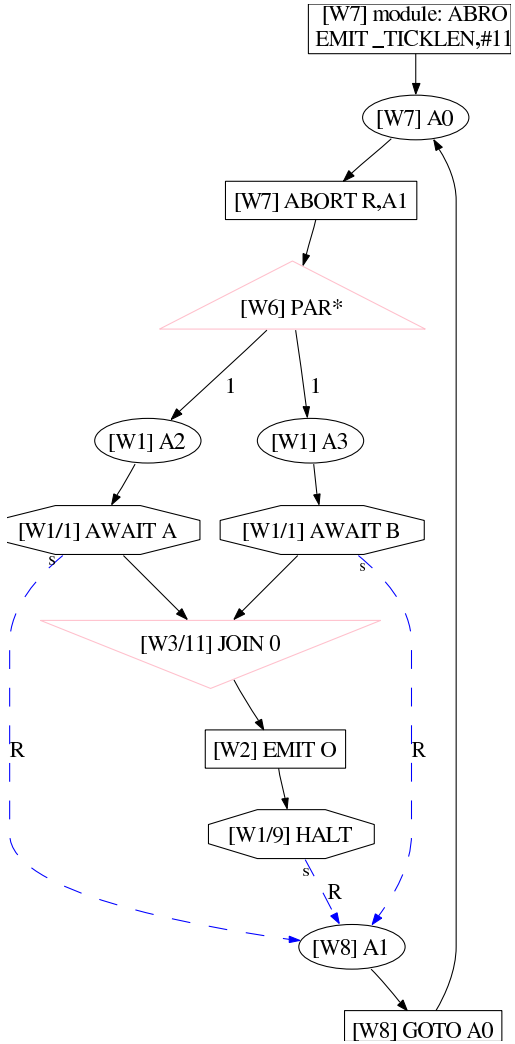
Worst Case Reaction Time

- Compute maximal time to react:
  one valid program state to another
- Similar to stabilization time of circuits

WCRT as Longest Path

```
    EMIT _TICKLEN,#11
A0: ABORT R,A1
    PAR 1,A2,1
    PAR 1,A3,2
    PARE A4,1
A2: AWAIT A
A3: AWAIT B
A4: JOIN 0
    EMIT O
    HALT
A1: GOTO A0
```

- Compute longest path between delay-nodes
- Abstract data-dependencies
- *Ad-hoc* optimizations

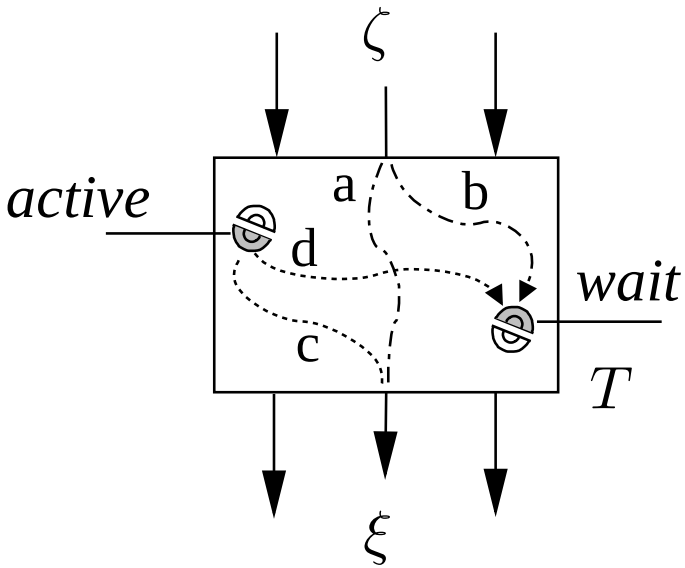


Outline

Interfaces

- Approach: use interface algebra to express WCRT
- Solid theoretical basis
- Allows refinement, eg. considering data-dependencies
- Modular computation (dynamic programming)
- Computation: $(max, +)$-algebra on timing matrix

Node Types



$$T = \begin{pmatrix} d_{thr} & d_{src} \\ d_{snk} & d_{int} \end{pmatrix} : (\zeta \vee active) \supset (\circ\xi \oplus \circ wait)$$

Interface Types

$$D{:}\phi \supset \psi$$

- Delay Matrix

$$D = \begin{pmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & & \vdots \\ d_{m1} & \cdots & d_{mn} \end{pmatrix}$$

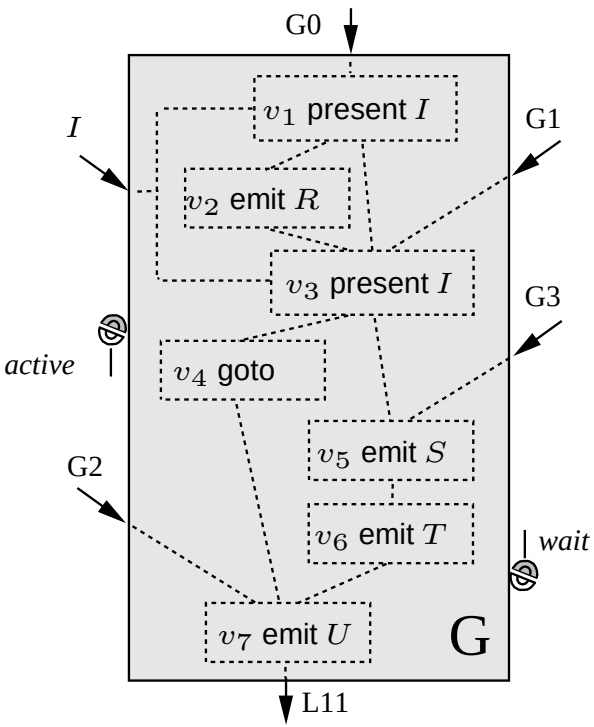
- Input Control

$$\phi = \zeta_1 \vee \zeta_2 \vee \cdots \vee \zeta_m$$

- Output Control

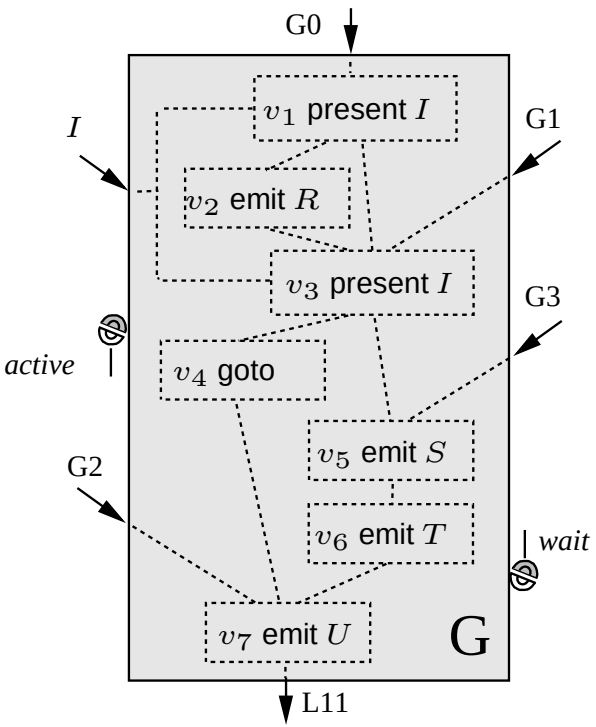
$$\psi = \circ\zeta_1 \oplus \circ\zeta_2 \oplus \cdots \oplus \circ\zeta_n$$

Expressing the WCRT



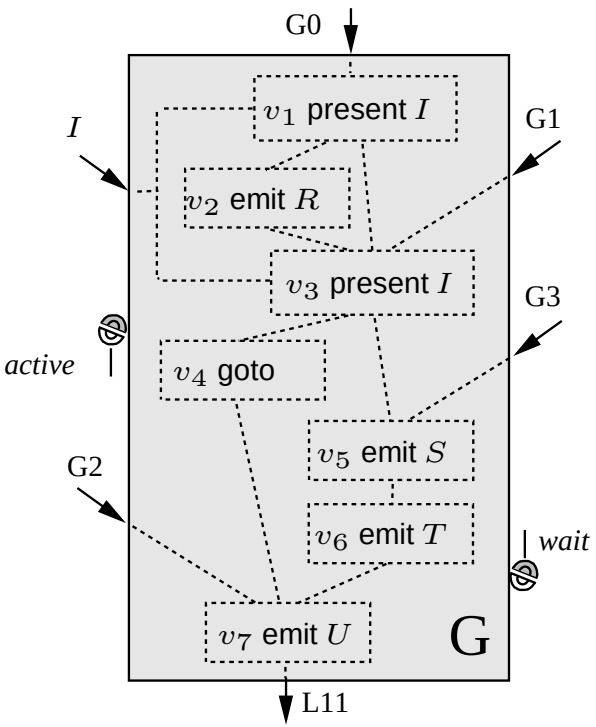
- $(6) : G0 \supset \circ L11$

Expressing the WCRT



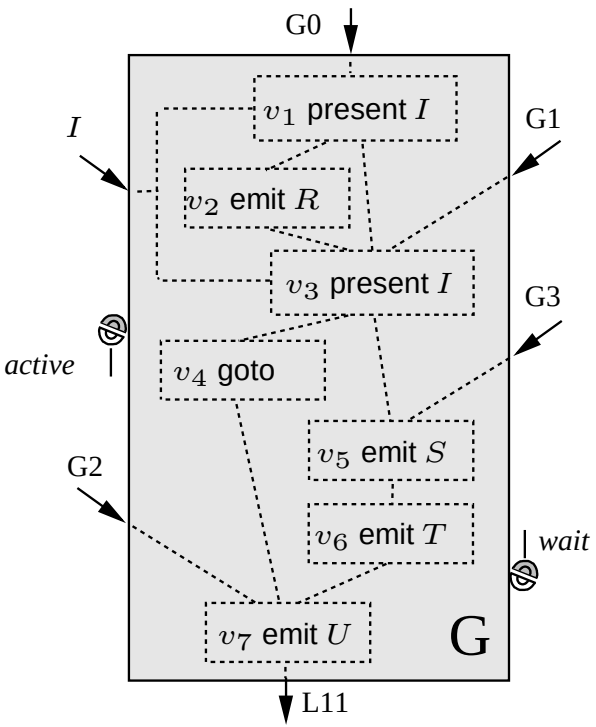
- $(6) : G0 \supset \circ L11$
- $(6, 4, 3, 1) :$
  $(G0 \vee G1 \vee G3 \vee G2) \supset \circ L11$

Expressing the WCRT



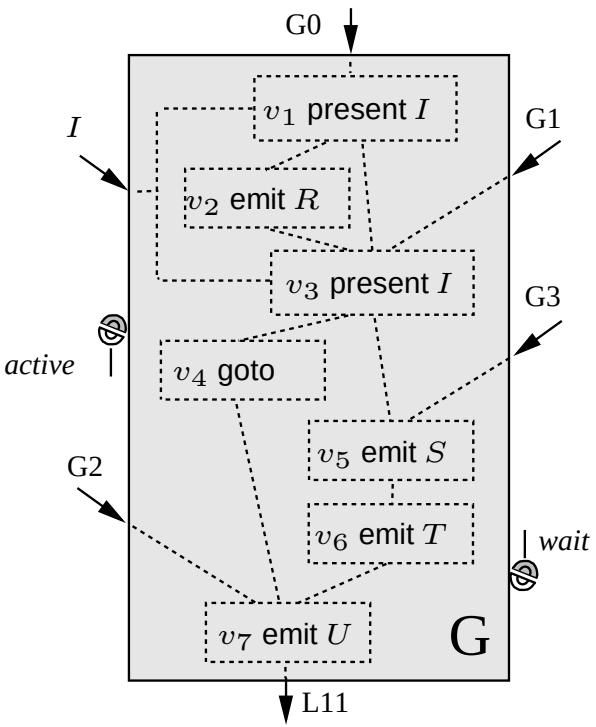
- $(6) : G0 \supset \circ L11$
- $(6, 4, 3, 1) :$
  $(G0 \vee G1 \vee G3 \vee G2) \supset \circ L11$
- $(5, 5, 3, 4, 3, 1) :$
  $((G0 \wedge I) \vee (G0 \wedge \neg I) \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$

Expressing the WCRT



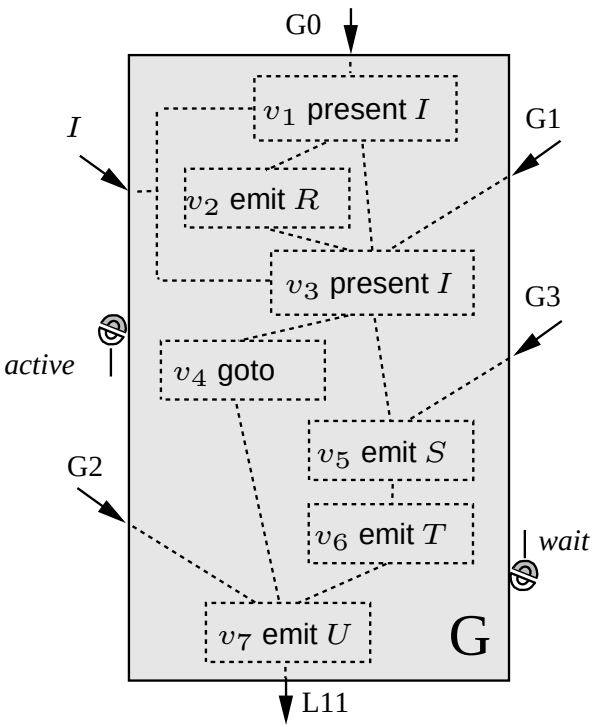
- $(6) : G0 \supset \circ L11$
- $(6,4,3,1) :$
  $(G0 \vee G1 \vee G3 \vee G2) \supset \circ L11$
- $(5,5,3,4,3,1) :$
  $((G0 \wedge I) \vee (G0 \wedge \neg I) \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$
- $(5,3,4,3,1) : (G0 \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$

Expressing the WCRT



- $(6) : G0 \supset \circ L11$
- $(6,4,3,1) :$
  $(G0 \vee G1 \vee G3 \vee G2) \supset \circ L11$
- $(5,5,3,4,3,1) :$
  $((G0 \wedge I) \vee (G0 \wedge \neg I) \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$
- $(5,3,4,3,1) : (G0 \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$
- $(5,3,4,1) : (G0 \vee ((G1 \wedge I) \oplus$
  $G3) \vee (G1 \wedge \neg I) \vee G2) \supset \circ L11$

Expressing the WCRT



- $(6) : G0 \supset \circ L11$
- $(6,4,3,1) :$
  $(G0 \vee G1 \vee G3 \vee G2) \supset \circ L11$
- $(5,5,3,4,3,1) :$
  $((G0 \wedge I) \vee (G0 \wedge \neg I) \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$
- $(5,3,4,3,1) : (G0 \vee (G1 \wedge I) \vee$
  $(G1 \wedge \neg I) \vee G3 \vee G2) \supset \circ L11$
- $(5,3,4,1) : (G0 \vee ((G1 \wedge I) \oplus$
  $G3) \vee (G1 \wedge \neg I) \vee G2) \supset \circ L11$
- $(5) : G0 \supset \circ L11$

Implementation

- Implemented an WCRT mechanism based on interface algebra in KEP compiler
- Identify blocks with threads
- Compute the through, source, sink and internal WCRT for each thread independently

Implementation

- Implemented an WCRT mechanism based on interface algebra in KEP compiler
- Identify blocks with threads
- Compute the through, source, sink and internal WCRT for each thread independently
- So far, all outgoing transitions from a thread are abstracted into one
- Do not consider traps yet, replace these by local signals + abortions
- Does not yet distinguish immediate/delayed aborts

Experimental Results

- Compared with existing, graph-based WCRT-analysis mechanism
- Set of benchmarks from Estbench test suite (Columbia)

	WCRT	
Module name	Graph	Interface
abro	11	11
atds	60	34
mca200	1779	1782
runner	20	16
tcint	191	126
watch	11	12

Conclusion

- Benefits:
  - Flexibility—can balance precision and analysis effort
  - Handling of control data
  - Systematic treatment of parallel execution
- Implemented basic ideas—promising first results

Outlook

- Delayed abortion + traps
- Consider thread priorities
- Formalize semantics of KEP

Outlook

- Delayed abortion + traps
- Consider thread priorities
- Formalize semantics of KEP

Thanks!
Questions/Comments?