#### The Kiel Reactive Processor Reactive Processing beyond the KEP

#### Claus Traulsen

Christian-Albrechts Universität zu Kiel

#### Synchron 2007

29. November 2007





CIAU

General Idea The Kiel Esterel Processor (KEP)

### Outline

#### Reactive Processing

General Idea The Kiel Esterel Processor (KEP)

#### Instruction Set

Candidates Scade

#### The KReP

Basic Ideas Example Execution Outlook

### The Problem

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

### The Problem

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
- Precise timing

## The Problem

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
- Precise timing

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

## Solution: Reactive Processing

Just another application (class) specific processor

- Deterministic control flow
- Predictable timing
- Short design cycle
- Low power requirements

## Solution: Reactive Processing

Just another application (class) specific processor

- Deterministic control flow
- Predictable timing
- Short design cycle
- Low power requirements
- Can use reactive processor
  - in stand alone, small reactive applications

## Solution: Reactive Processing

Just another application (class) specific processor

- Deterministic control flow
- Predictable timing
- Short design cycle
- Low power requirements
- Can use reactive processor
  - in stand alone, small reactive applications
  - as building block in SoC designs

# **KEP:** Current Status

- Reactive Processor for Esterel
- "Directly" executes Esterel
- Implements Esterel v5 (nearly) completely
  - Includes concurrency
  - Includes valued signals (but no combine)

# KEP: Current Status

- Reactive Processor for Esterel
- "Directly" executes Esterel
- Implements Esterel v5 (nearly) completely
  - Includes concurrency
  - Includes valued signals (but no combine)
- Complete toolchain
  - Compiler from Esterel to KEP assembler uses Columbia Esterel Compiler as front-end
  - Compiler from KEP assembler
  - Testbench for automatic testing Generate trace files (esi/eso) and compare to Esterel-Studio

General Idea The Kiel Esterel Processor (KEP)

### KEP: Recent work

Reimplementation in Esterel v7

- Better maintainability
- Medium size Esterel example:
   > 4000 lines of Esterel code
- SoftKEP for fast testing



General Idea The Kiel Esterel Processor (KEP)

### KEP: Recent work

Reimplementation in Esterel v7

- Better maintainability
- Medium size Esterel example:
   > 4000 lines of Esterel code
- SoftKEP for fast testing

Connection to real world

- So far the KEP executes trace files
- Have it control some "real" environment



General Idea The Kiel Esterel Processor (KEP)

### KEP: Drawbacks

- Input limited to Esterel
- Niche between hardware and software generation Even more true for KEP on FPGA
- Single core with multiple threads (Logical concurrency, not suited for acceleration)

General Idea The Kiel Esterel Processor (KEP)

#### The Kiel Reactive Processor

Key Idea: Have a wider field of application

1. Compile other languages to the KEP Assembler

General Idea The Kiel Esterel Processor (KEP)

### The Kiel Reactive Processor

Key Idea: Have a wider field of application

- 1. Compile other languages to the KEP Assembler
- 2. Have a new reactive Processor But keep the good parts of the KEP:
  - Support for concurrency and preemption
  - Precise timing
  - Easy compilation

#### Candidates Scade

### Outline

#### Reactive Processing

General Idea The Kiel Esterel Processor (KEP)

#### Instruction Set

Candidates Scade

#### The KReP

Basic Ideas Example Execution Outlook

### Requirements for the Instruction Set

- Should grab key idea of reactive processing!
- Direct support for:
  - Preemption
  - Concurrency
- Timing is an essential part of semantics
- Implementable
- Easy use as target language

### Requirements for the Instruction Set

- Should grab key idea of reactive processing!
- Direct support for:
  - Preemption
  - Concurrency
- Timing is an essential part of semantics
- Implementable
- Easy use as target language
- Similar requirements as for real-time languages
- $\sim$  Use them as a base language (Just like Esterel for the KEP)

## Requirements for the Base Language

- Widly used for programming reactive systems
- Well defined (formal semantics)
- Small kernel
- Support for reactive control-flow

Lets take a look at other languages ...

# A Survey on Possible Languages (Esterel like)

- IC Not more used than Esterel
  - Only use as intermediate language
  - Probably easier?

# A Survey on Possible Languages (Esterel like)

- IC Not more used than Esterel
  - Only use as intermediate language
  - Probably easier?
- - Comparison to KASM might be interesting

# A Survey on Possible Languages (Esterel like)

- IC Not more used than Esterel
  - Only use as intermediate language
  - Probably easier?
- BAL ► Can be generated from Esterel
  - Comparison to KASM might be interesting
- - Not widly spread (yet?)

## A Survey on Possible Languages (Mainstream)

 $\mathsf{Simulink}/\mathsf{Stateflow}$ 

- Widely used
- Semantics complicated (and changing)
- Unclear how to implement

# A Survey on Possible Languages (Mainstream)

 $\mathsf{Simulink}/\mathsf{Stateflow}$ 

UML Stacharts

Widely used

- Semantics complicated (and changing)
- Unclear how to implement
- Clearly wide spread
- Not well defined
- Not deterministic
- Hard to implement efficiently

## A Survey on Possible Languages (Mainstream)

 $\mathsf{Simulink}/\mathsf{Stateflow}$ 

UML Stacharts

Real Time Java

Widely used

- Semantics complicated (and changing)
- Unclear how to implement
- Clearly wide spread
- Not well defined
- Not deterministic
- Hard to implement efficiently
- Clearly wide spread
- Real-Time is only an "add-on"
- Concurrency is not deterministic

## A Survey on Possible Languages (Lustre like)

Lustre

- Mainly concurrent equations
  - Precise and simple semantics
  - Compilation is quite efficient
  - Might get benefit from parallel execution (Multicore)

## A Survey on Possible Languages (Lustre like)

Lustre

- Mainly concurrent equations
  - Precise and simple semantics
  - Compilation is quite efficient
  - Might get benefit from parallel execution (Multicore)

Scade

- Adds automata to Lustre
- Special hardware might be useful for
  - Deep hierarchy of automata
  - Parallelism

### Which one to choose?

So far, our candidate is Scade because ...

- clear and simple semantics
- mixing of dataflow and automata (introduced in Lucid Synchrone by Pouzet)
- import from Simulink/Stateflow (SystemC/TLM, AADL)
- easier than Esterel
  - no schizophrenia
  - simple causality analysis
  - preemption more restricted

### Which one to choose?

So far, our candidate is Scade because ...

- clear and simple semantics
- mixing of dataflow and automata (introduced in Lucid Synchrone by Pouzet)
- import from Simulink/Stateflow (SystemC/TLM, AADL)
- easier than Esterel
  - no schizophrenia
  - simple causality analysis
  - preemption more restricted

But there are some drawbacks:

- Not too widely used
- No open tool, no "Scade community"

#### Candidates Scade

# Code Generation

Software:

- Efficient compilation to C code
- Automata are first transformed to dataflow
- Makes code lengthy and hard to read

Hardware:

- Can generate hardware (just as for Lustre)
- But not currently done by the SCADE tool





### Outline

#### Reactive Processing

General Idea The Kiel Esterel Processor (KEP)

#### Instruction Set

Candidates Scade

#### The KReP

Basic Ideas Example Execution Outlook Reactive Processing Bas Instruction Set Exa The KReP Out

Basic Ideas Example Execution Outlook

### Aims of the KReP

- Faster than software
- Precise timing
  - Stall when too fast
  - Balance workload (WCET not ACET)
  - Need very good WCET analysis

Reactive Processing Bas Instruction Set Exa The KReP Our

Basic Ideas Example Execution Outlook

### Aims of the KReP

- Faster than software
- Precise timing
  - Stall when too fast
  - Balance workload (WCET not ACET)
  - Need very good WCET analysis
  - Be precise and fast!

Reactive Processing Bas Instruction Set Exa The KReP Our

Basic Ideas Example Execution Outlook

### Aims of the KReP

- Faster than software
- Precise timing
  - Stall when too fast
  - Balance workload (WCET not ACET)
  - Need very good WCET analysis
  - Be precise and fast!
- Parallel Execution
- Support for automata

Basic Ideas Example Execution Outlook

### Overview of the Architecture

CIAU



Slide 19

## A Simple Program



Execute on two cores:



Image: A matrix

< E

### A Simple Program



Execute on two cores:

P1 
$$L1 = I1 + 2$$
  
P2  $L2 = L1 * I1$ 

Image: A matrix

< E

Basic Ideas Example Execution Outlook

### Parallel Execution: Step 0



< E

Basic Ideas Example Execution Outlook

### Parallel Execution: Step 1



- ( E

Basic Ideas Example Execution Outlook

### Parallel Execution: Step 2



A D b 4 A b

A B + A B +

Basic Ideas Example Execution Outlook

### Parallel Execution: Step 3



→ Ξ → → Ξ →

### Parallel Execution: Step 4



CAU

4 A<sup>™</sup> ▶

Basic Ideas Example Execution Outlook

### Parallel Execution: Wrap-Up

Similar to distribution of Lustre programs

- New hardware for synchronization
- Still have a global clock.

Basic Ideas Example Execution Outlook

### Parallel Execution: Wrap-Up

Similar to distribution of Lustre programs

- New hardware for synchronization
- Still have a global clock.

Can as well be seen as multicore execution:

- ► Do we really need new hardware ....
- ... or can we use COTS multicore and some SW instead?

#### Basic Ideas Example Execution Outlook

## Dealing with Automata

- 1. Compile to dataflow
  - + Can use existing tools
    - Lengthy code
    - Code for transitions is executed each tick
- 2. Special instructions for automata
  - + Better performance
  - + More information for WCRT analysis
    - Unclear how to preempt
    - Have to deal with concurrency inside atomata

#### Execution of automata

Execution of an automaton (from the Scade Language Primer):

- $1. \ \ {\rm Determine \ the \ selected \ state}$
- 2. If the state has outgoing transitions, evaluate all the guards of the strong transitions and inspect them
- 3. Determine the active state
- 4. Compute actions in the active state
- 5. If no strong transition has fired, evaluate and inspect all the guards of the weak transitions

### Execution of automata

Execution of an automaton (from the Scade Language Primer):

- $1. \ \ {\rm Determine \ the \ selected \ state}$
- 2. If the state has outgoing transitions, evaluate all the guards of the strong transitions and inspect them
- 3. Determine the active state
- 4. Compute actions in the active state
- 5. If no strong transition has fired, evaluate and inspect all the guards of the weak transitions

Could directly implement this algorithm,

#### Execution of automata

Execution of an automaton (from the Scade Language Primer):

- $1. \ \ {\rm Determine \ the \ selected \ state}$
- 2. If the state has outgoing transitions, evaluate all the guards of the strong transitions and inspect them
- 3. Determine the active state
- 4. Compute actions in the active state
- 5. If no strong transition has fired, evaluate and inspect all the guards of the weak transitions

Could directly implement this algorithm, but have to traverse complete hierarchy in each step.

### Automata and Watcher

Idea: Similar to the KEP

- Watcher unit checks whether a transition is triggered
- Watcher is initialized when state is entered
- Watcher gives new PC
- But multicore: More like the Emperor
  - 1. Each processor has one watcher (no parallelism/easy)
  - One watcher for multiple processors SYNC instruction asks watcher unit at begin/end of each tick

Advantage of Scade: Preemption is more constrained than in Esterel

### Related Work

Processors:

- ► KEP by Li
- Emperor/STARPro by Roop *et al.*
- JOP (java optimized processor) by Schoeberl
- PRET (Precision timed machine) by Lee and Edwards

### Related Work

Processors:

- ► KEP by Li
- Emperor/STARPro by Roop *et al.*
- JOP (java optimized processor) by Schoeberl
- ▶ PRET (Precision timed machine) by Lee and Edwards

Distributed execution of Lustre:

- ocrep/screp by Girault
- Lustre and TTA by Curic et al.

▶ ...

 
 Reactive Processing Instruction Set
 Basic Ideas

 The KReP
 Outlook

### Conclusion

ъ

イロト イポト イヨト イヨト

 
 Reactive Processing Instruction Set
 Basic Ideas

 The KReP
 Outlook

### Conclusion

#### None yet



A 3 1 A 3 1

ъ

### Current Status

- Hacked a prototype in Esterel v7
- Can execute Counter on three cores
- Object code generated by hand

Reactive Processing Bas Instruction Set Exa The KReP Out

Basic Ideas Example Execution Outlook

## Future Work /Open Questions

- 1. Integrate Automata
  - How to preempt efficiently
  - Reassign work to processors
- 2. Think about the timing
  - Precise WCRT
  - Timing guarantees in the ISA
- 3. Automatic compilation
  - How to balance the workload

Reactive Processing Bas Instruction Set Exa The KReP Out

Basic Ideas Example Execution Outlook

## Future Work /Open Questions

- 1. Integrate Automata
  - How to preempt efficiently
  - Reassign work to processors
- 2. Think about the timing
  - Precise WCRT
  - Timing guarantees in the ISA
- 3. Automatic compilation
  - How to balance the workload

#### Any comments are welcome!