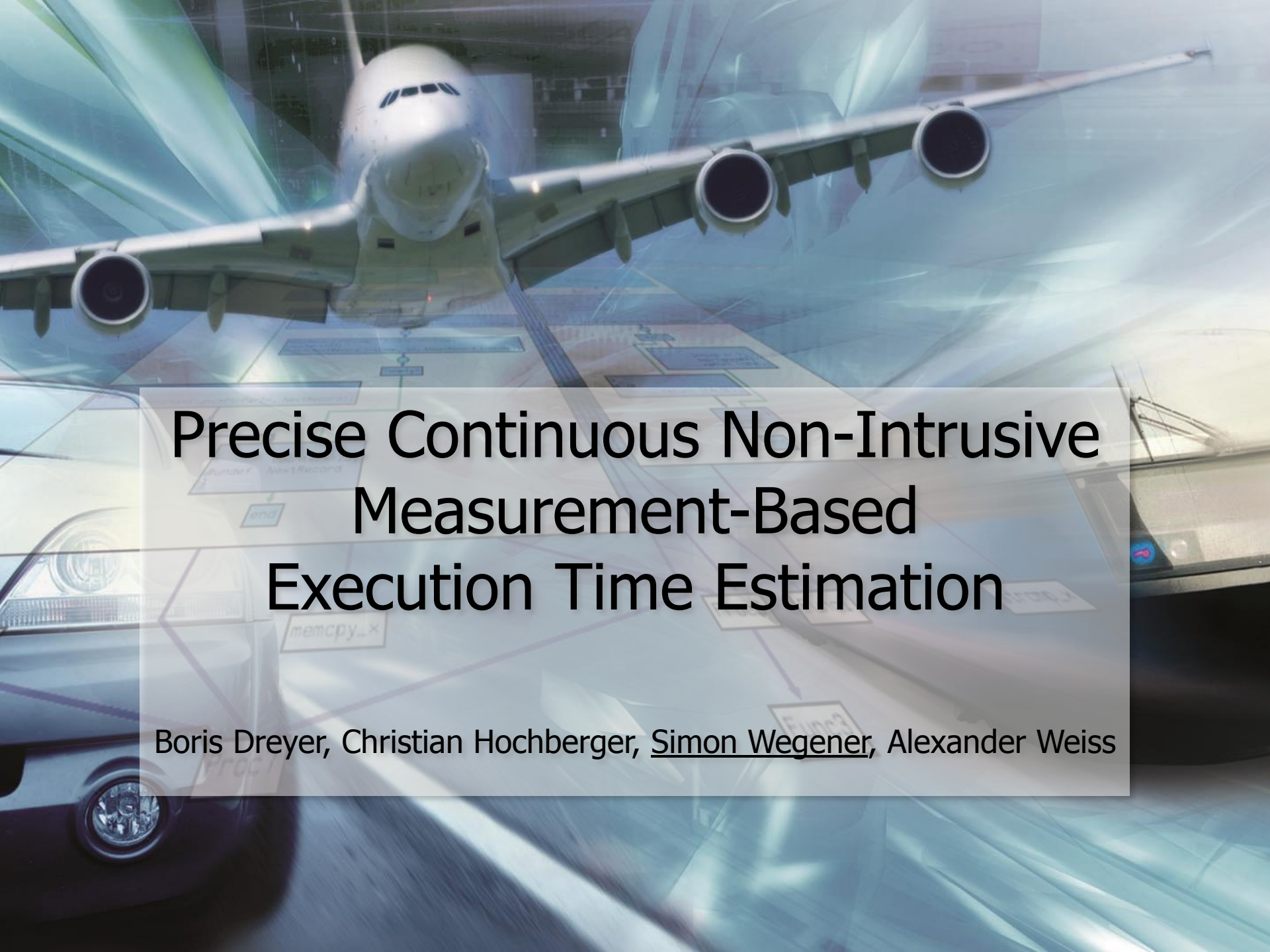


Precise Continuous Non-Intrusive Measurement-Based Execution Time Estimation

Boris Dreyer, Christian Hochberger, Simon Wegener, Alexander Weiss

CONIRAS

This work was funded within the project CONIRAS by the German Federal Ministry for Education and Research with the funding ID 01IS13029. The responsibility for the content remains with the authors.

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

AbsInt

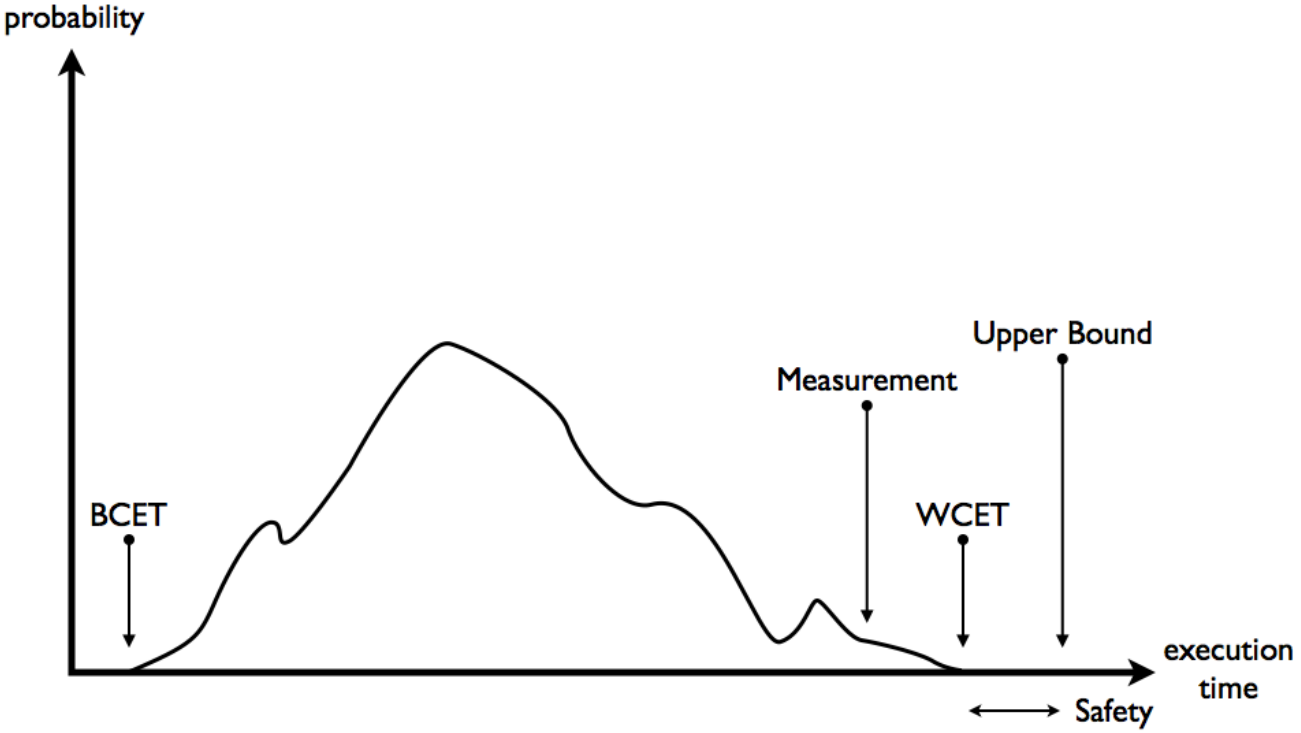
ACCEMIC
accelerate your microcontroller design

TECHNISCHE
UNIVERSITÄT
DARMSTADT

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR SOFTWARETECHNIK
UND PROGRAMMIERSPRACHEN

The Setting

The Timing Problem



Problem Solved?

Reinhard Wilhelm et al.:
The Worst-case Execution Time Problem
— Overview of Methods and Survey of Tools

CONCLUSIONS

„The problem of determining upper bounds on execution times for single tasks and for quite complex processor architectures has been solved. Several commercial WCET tools are available and have experienced very positive feedback from extensive industrial use.“

ACM Transactions on Embedded Computing Systems, Vol. 7, No. 3, April 2008.

Well, almost…

- Unpredictable delays due to interference on multi-cores
- Unpredictable hardware (e.g. random replacement policies for caches)
- Undocumented hardware

⇒ Very pessimistic results!

AUTOMOTIVE · RAILWAY · AVIONICS
MULTICORE SYSTEMS

Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement

ECRTS'14

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

PREDATOR : ICT project in the 7th Framework Program of the EU

Predictability Considerations in the Design of Multi-core Embedded Systems

ERTS 2010

Measurements?

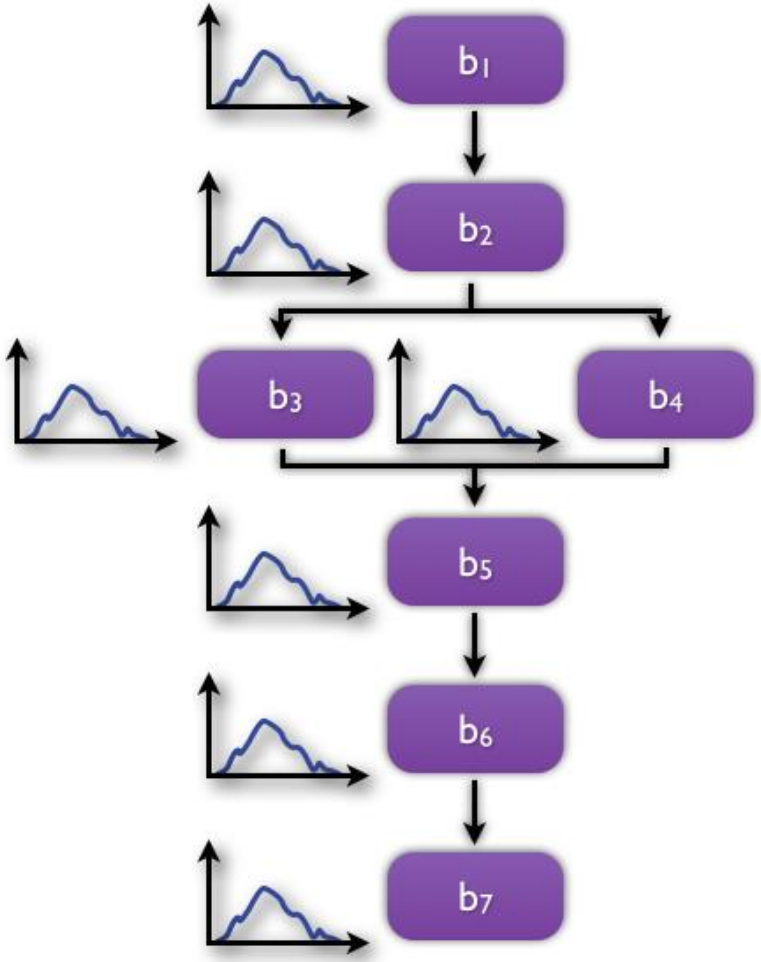
Karsten Schmidt et al.:

Non-Intrusive Tracing at First Instruction

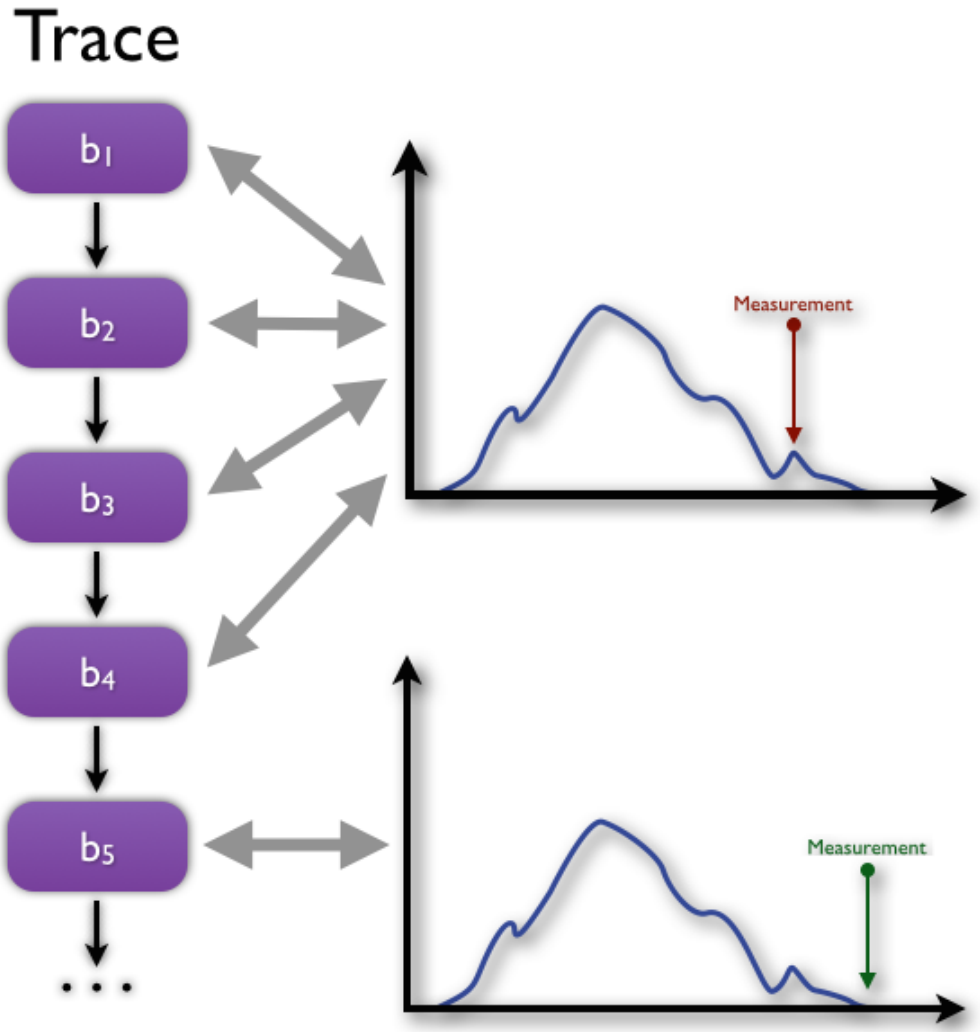
> "In the selected use case, a simple regression test, first a function trace for a longer time span, and then later on an instruction trace for a dedicated time span were generated. This approach (instead of tracing everything across a long time span) is necessary due to the limited bandwidth to components outside of the microcontroller."

SAE Technical Paper 2015-01-0176

Catching the Worst Case



Catching the Worst Case



Measurements?

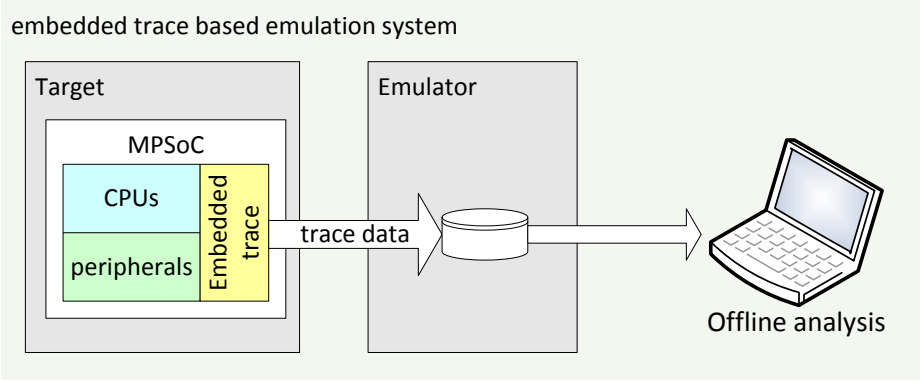
Karsten Schmidt et al.:

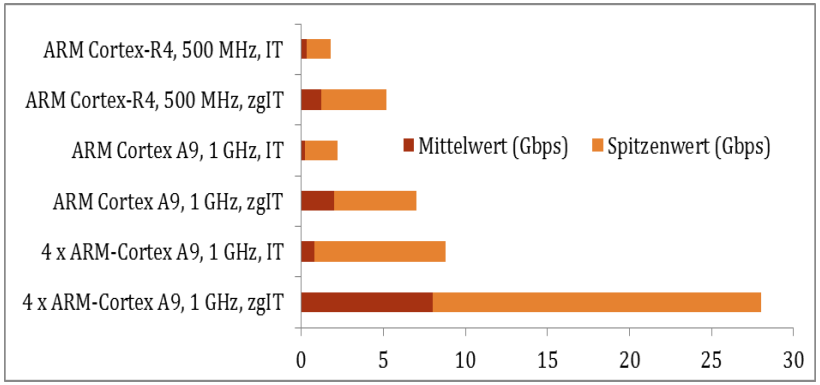
Non-Intrusive Tracing at First Instruction

> "The art of tracing lies in the selection of the proper abstraction level and time in order to perform safe regression tests or identify potential issues as fast as possible. Especially when working under high time pressure with high costs related to the tests, or when working under extreme conditions, […], efficiency in tracing is crucial."

SAE Technical Paper 2015-01-0176

State of The Art: Embedded Trace





Bandwith (average):
0,3 .. 4 Bit / Instruction (non-cycle accurate)
5 .. 8 Bit / Instruction (cycle accurate)
8 Bit / Instruction (data trace)

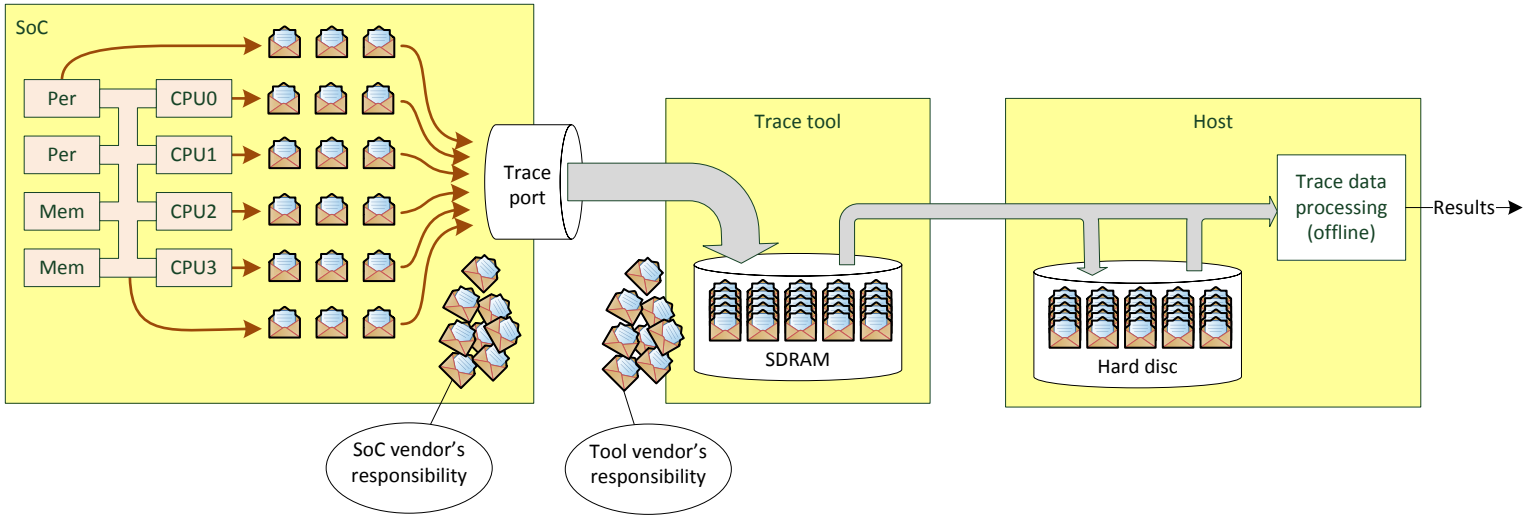
MPSoC, 4 CPUs, 1 GHz
Cycle accurate instruction + data trace
4 x 14 Bit x 1 GHz => approx. **28 Gbit/s**
(+ timestamps)
(+ bus master trace)
(+ peak bandwidth)

Time stamps	CPU 3
Instructions	
Data read	
Data written	
Time stamps	CPU 2
Instructions	
Data read	
Data written	
Time stamps	CPU 1
Instructions	
Data read	
Data written	
Time stamps	CPU 0
Instructions	
Data read	
Data written	

MPSoC (4 CPUs)

AbsInt ACCEMIC accelerate your microcontroller design
TECHNISCHE UNIVERSITÄT DARMSTADT
UNIVERSITÄT ZU LÜBECK INSTITUT FÜR SOFTWARETECHNIK UND PROGRAMMIERSPRACHEN

Trace Recording and Offline Processing

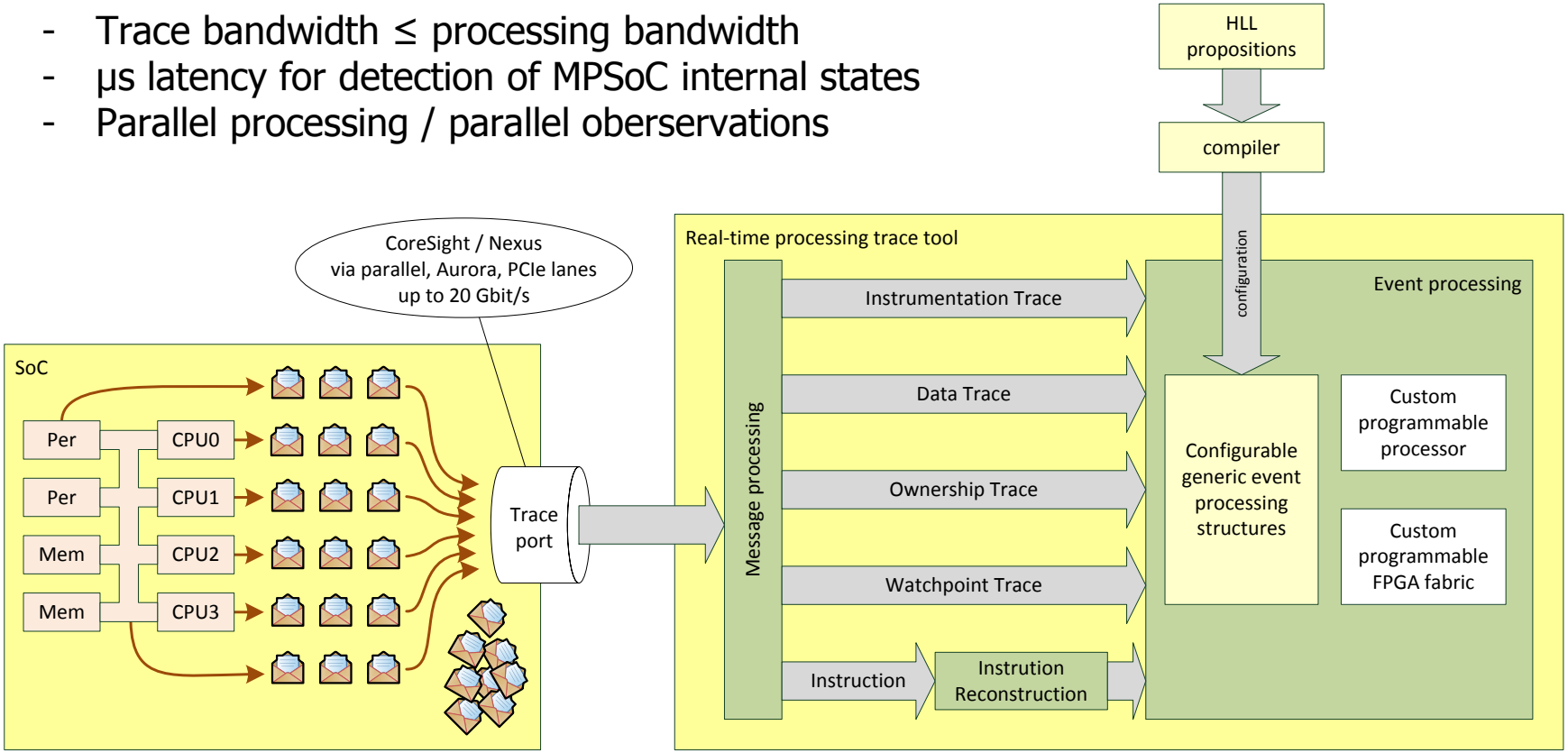


Drawbacks:

- Not all trace messages go through the SoC's trace port due to limited bandwidth
- Processing bandwidth of host trace port is smaller than trace port bandwidth
- Huge latency for the detection of MPSoC internal states
- Restricted to on-chip trigger logic

Solution: Online Analysis of Trace Data

- Trace bandwidth ≤ processing bandwidth
- µs latency for detection of MPSoC internal states
- Parallel processing / parallel oberservations



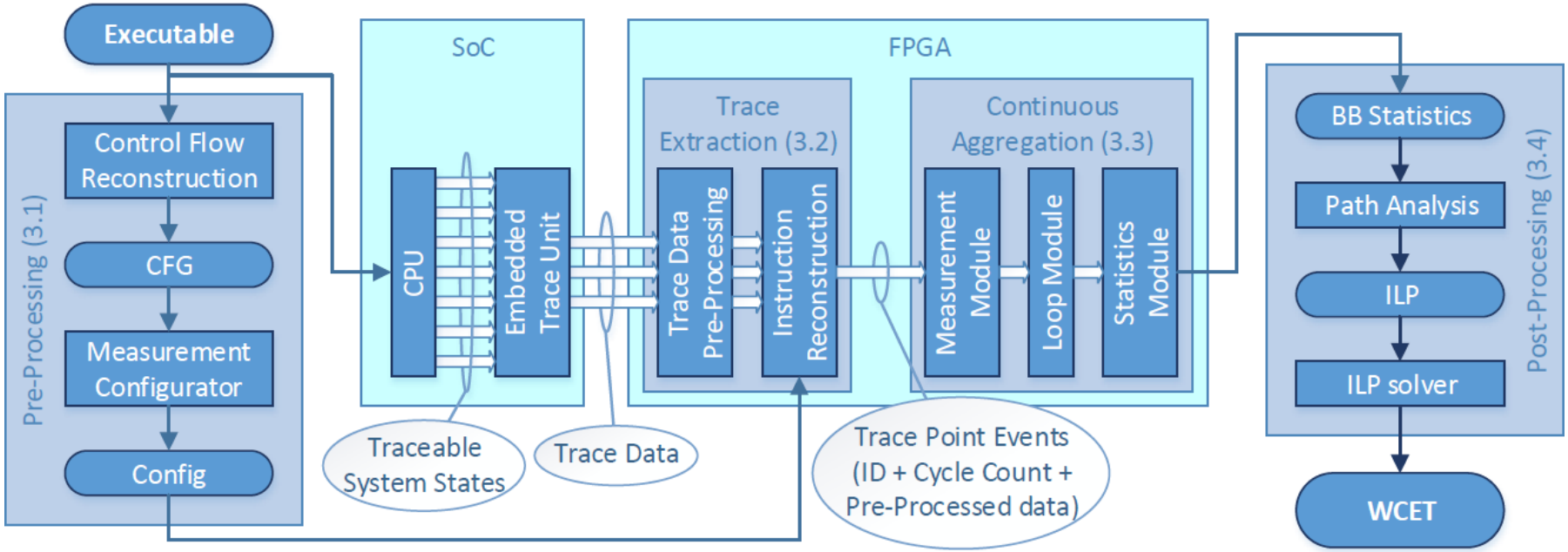
AbsInt ACCEMIC accelerate your microcontroller design

TECHNISCHE UNIVERSITÄT DARMSTADT

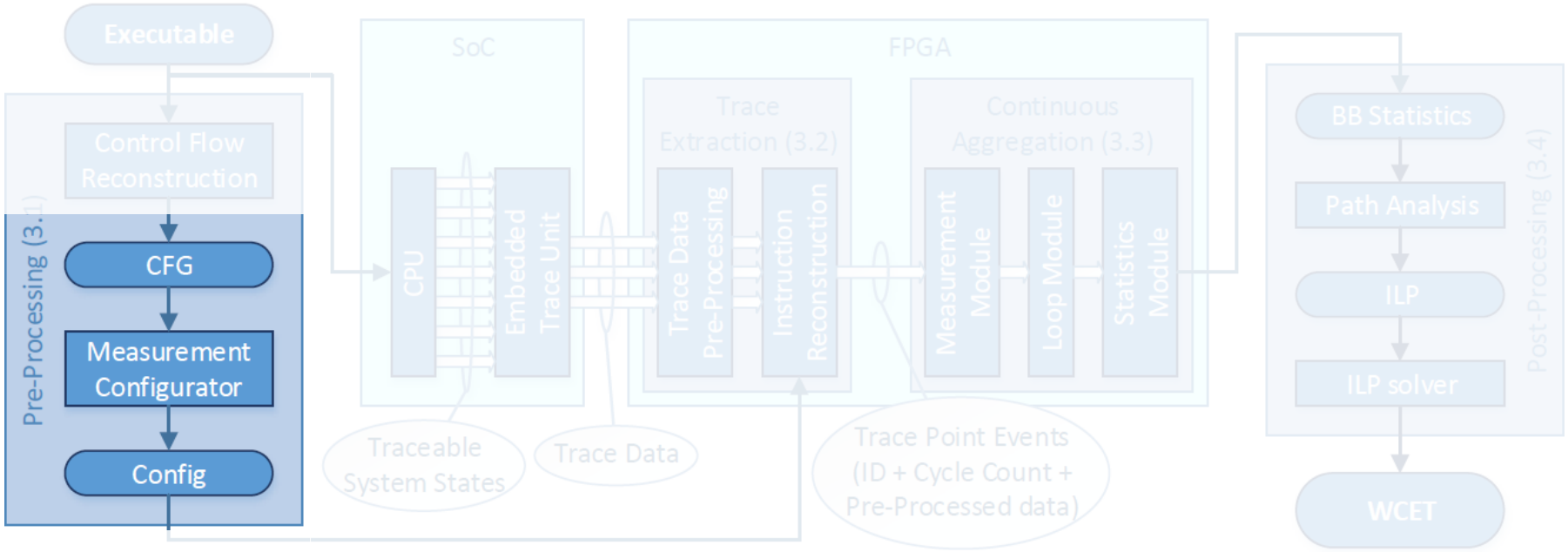
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR SOFTWARETECHNIK UND PROGRAMMIERSPRACHEN

Our Approach

Workflow

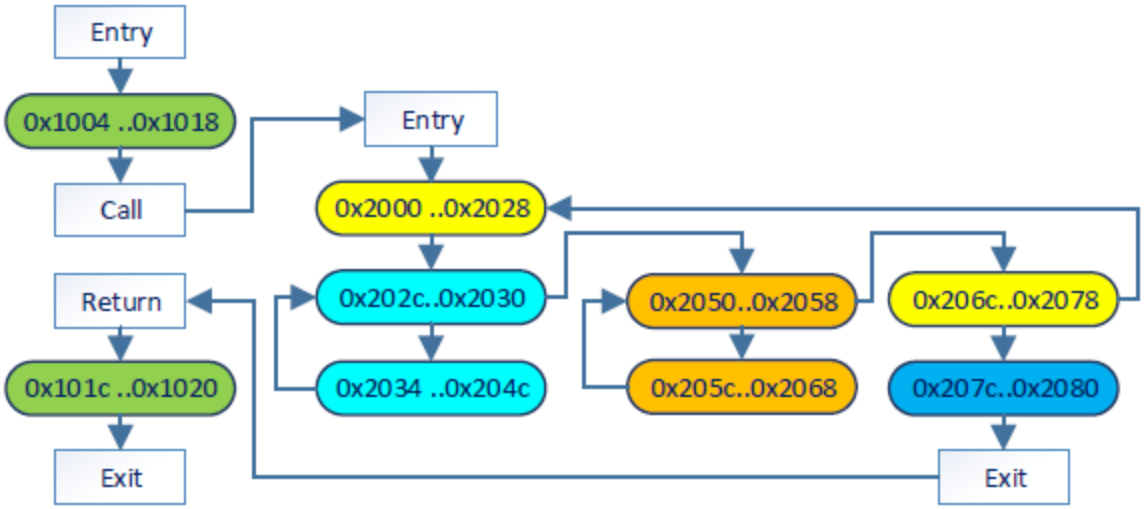

Executable
Pre-Processing (3.1)
Control Flow Reconstruction
CFG
Measurement Configurator
Config
SoC
CPU
Embedded Trace Unit
Traceable System States
Trace Data
FPGA
Trace Extraction (3.2)
Trace Data Pre-Processing
Instruction Reconstruction
Continuous Aggregation (3.3)
Measurement Module
Loop Module
Statistics Module
Trace Point Events (ID + Cycle Count + Pre-Processed data)
Post-Processing (3.4)
BB Statistics
Path Analysis
ILP
ILP solver
WCET

Workflow


Executable
Pre-Processing (3.1)
Control Flow Reconstruction
CFG
Measurement Configurator
Config
SoC
CPU
Embedded Trace Unit
Traceable System States
Trace Data
FPGA
Trace Extraction (3.2)
Trace Data Pre-Processing
Instruction Reconstruction
Continuous Aggregation (3.3)
Measurement Module
Loop Module
Statistics Module
Trace Point Events (ID + Cycle Count + Pre-Processed data)
Post-Processing (3.4)
BB Statistics
Path Analysis
ILP
ILP solver
WCET

Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {              // L1
    …;
    for (…;…;…)     // L2
      …;
    for (…;…;…)     // L3
      …;
  } while (…);
}
```



Events

(address, tag, level, distinctor, call, entry, exit, return)

Events are generated for each basic block.

Events

(address, tag, level, distinctor, call, entry, exit, return)

The address of the basic block‘s first instruction.

Events

(address, tag, level, distinctor, call, entry, exit, return)

A custom field used to efficiently address statistic records in memory

Events

(address, tag, level, distinctor, call, entry, exit, return)

The loop nesting level of the basic block.

Events

(address, tag, level, distinctor, call, entry, exit, return)

A field to distinguish two loops with the same nesting level that lie directly after another.

Events

(address, tag, level, distinctor, call, entry, exit, return)

Is another routine called in the basic block?

Events

(address, tag, level, distinctor, call, entry, exit, return)

Is a routine entered with this basic block?

Events

(address, tag, level, distinctor, call, entry, exit, return)

Is a routine exited with this basic block?

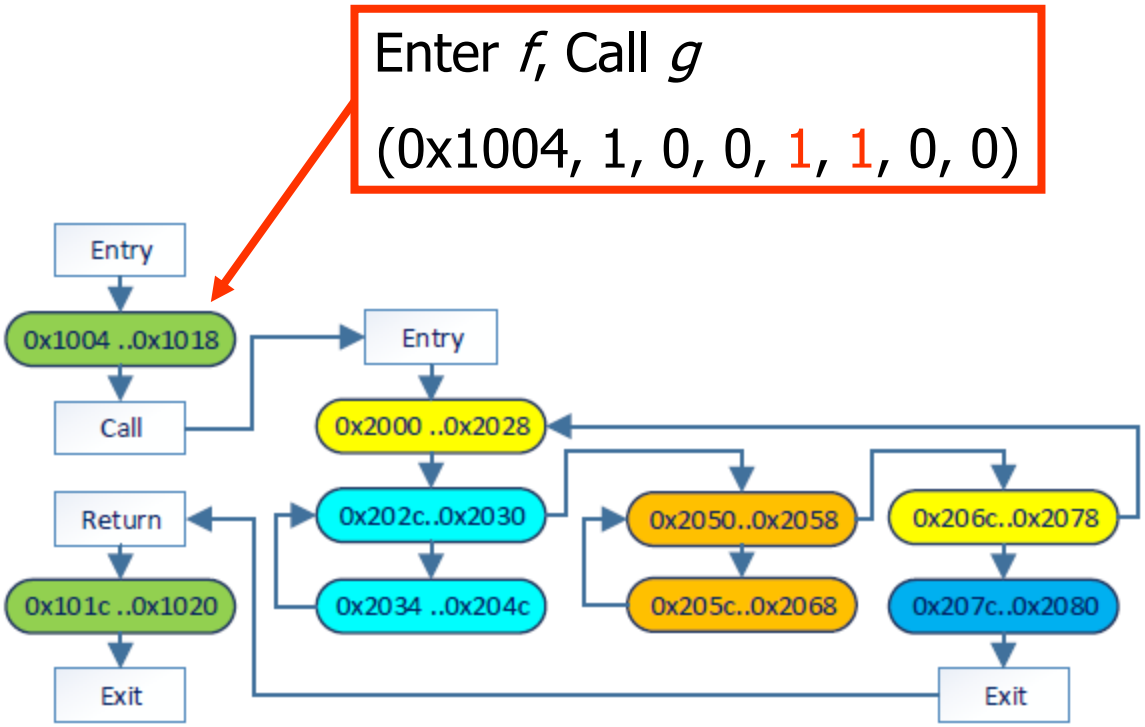
Events

(address, tag, level, distinctor, call, entry, exit, return)

Did a return from a called routine happen?

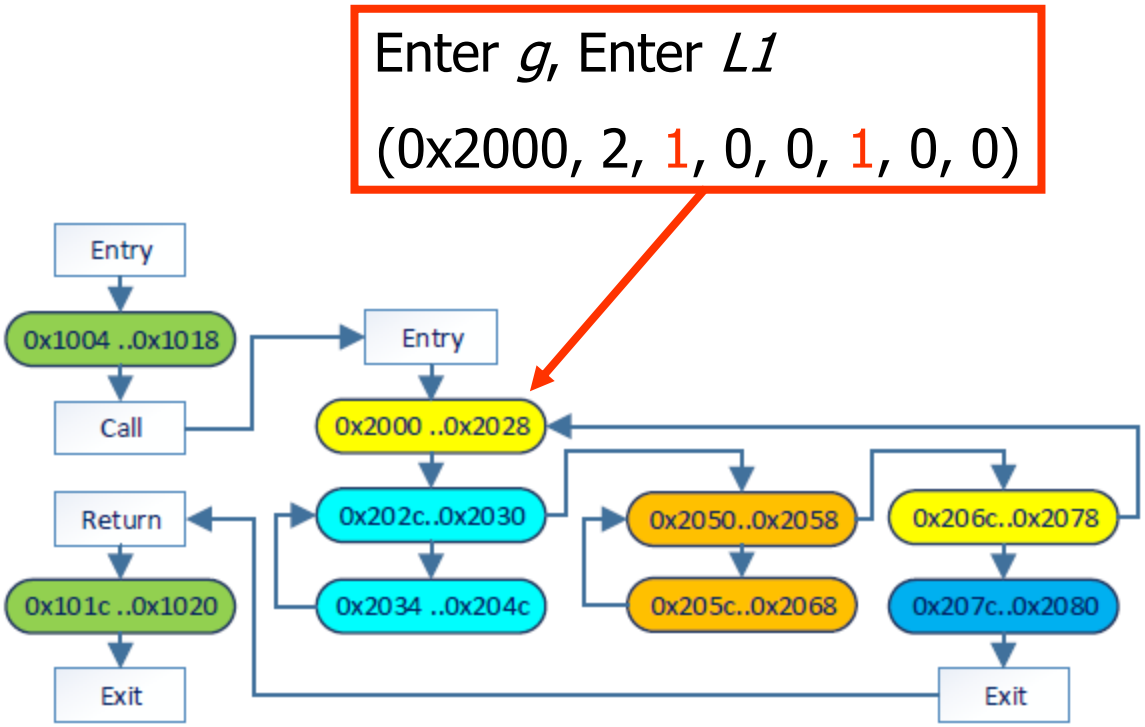
Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {              // L1
    …;
    for (…;…;…)     // L2
      …;
    for (…;…;…)     // L3
      …;
  } while (…);
}
```



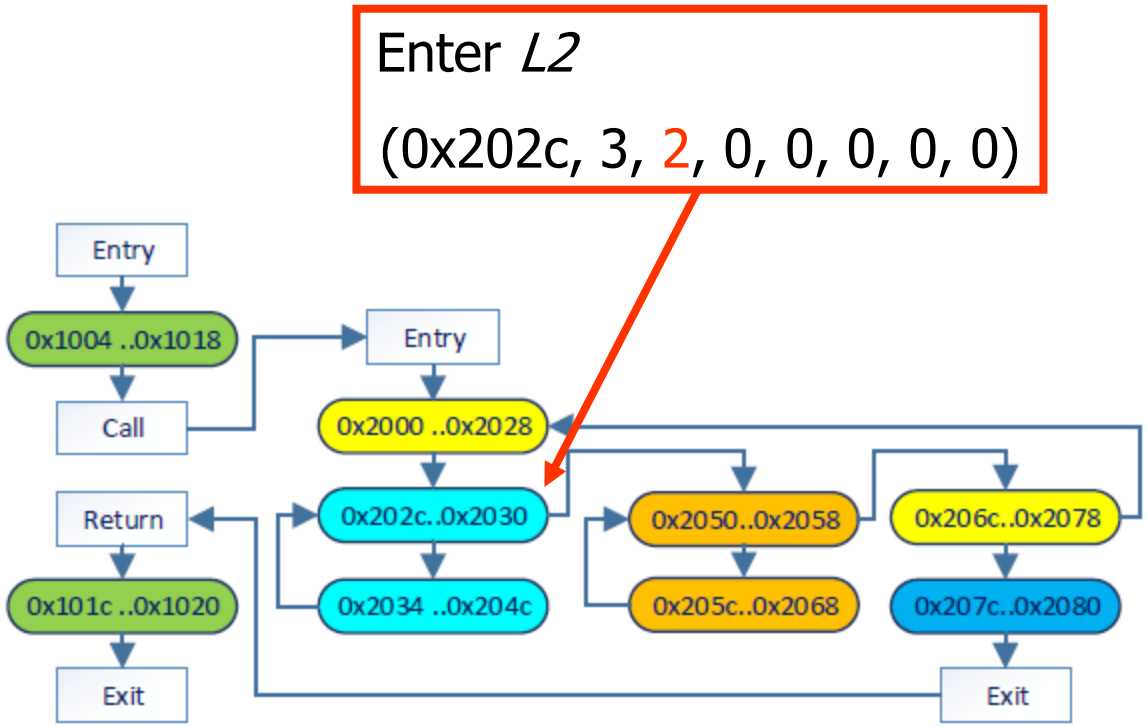
Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {                 // L1
    …;
    for (…;…;…)        // L2
      …;
    for (…;…;…)        // L3
      …;
  } while (…);
}
```



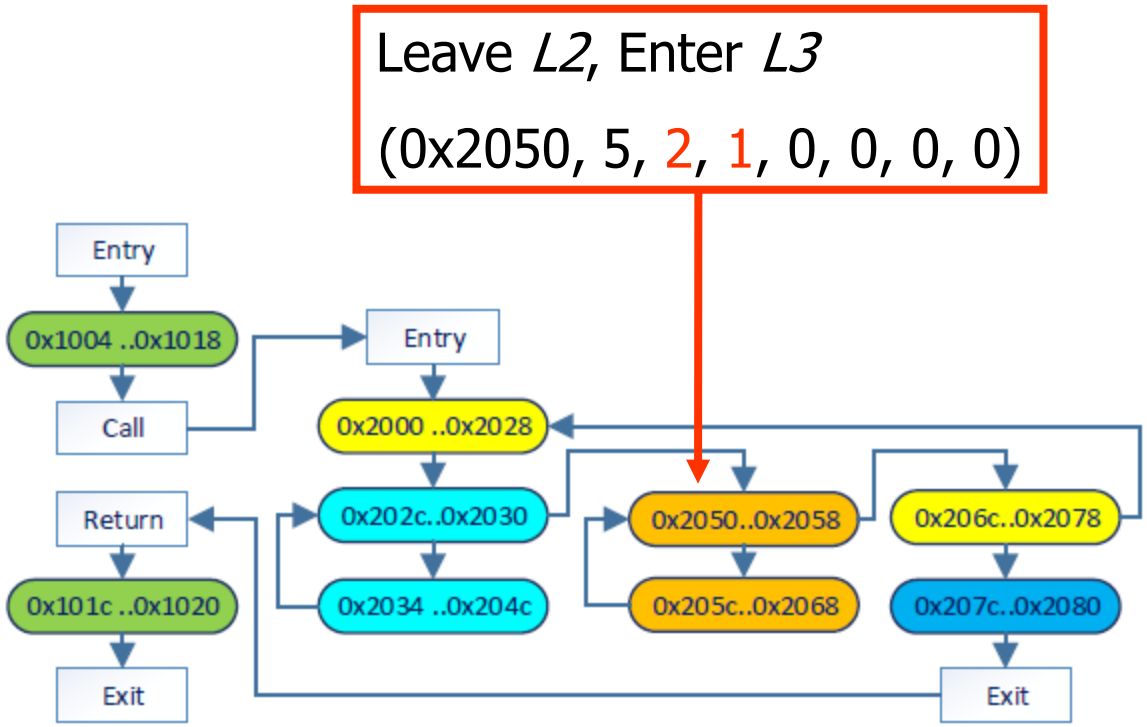
Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {              // L1
    …;
    for (…;…;…)     // L2
      …;
    for (…;…;…)     // L3
      …;
  } while (…);
}
```



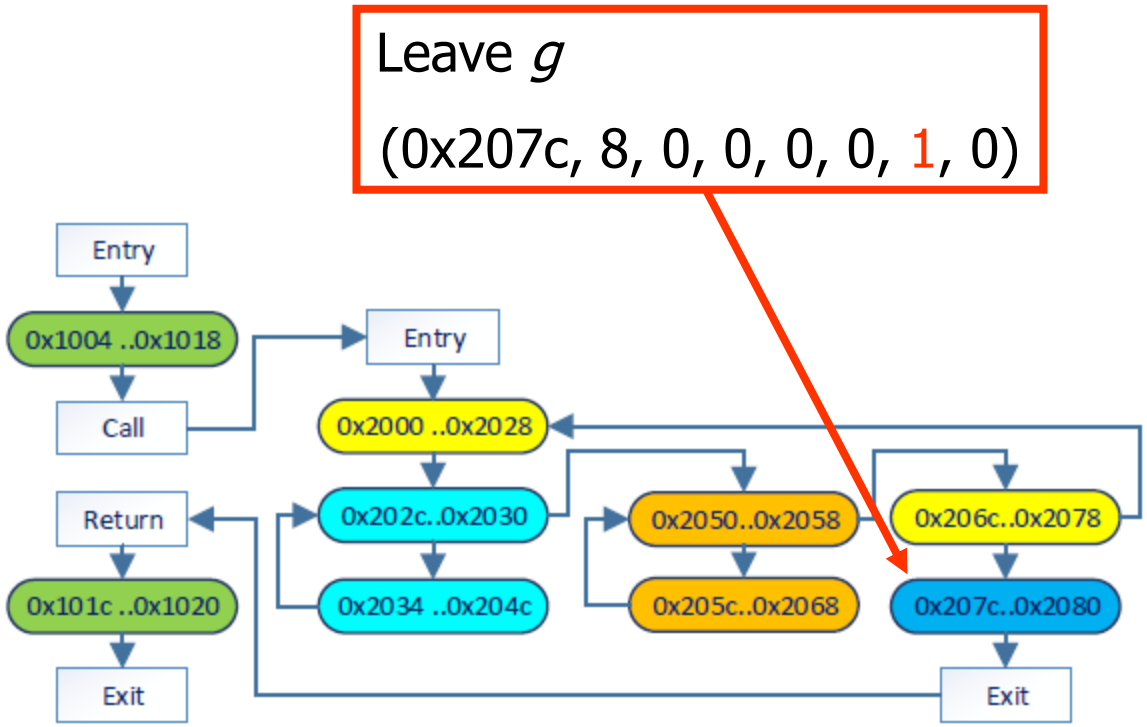
Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {              // L1
    …;
    for (…;…;…)     // L2
      …;
    for (…;…;…)     // L3
      …;
  } while (…);
}
```

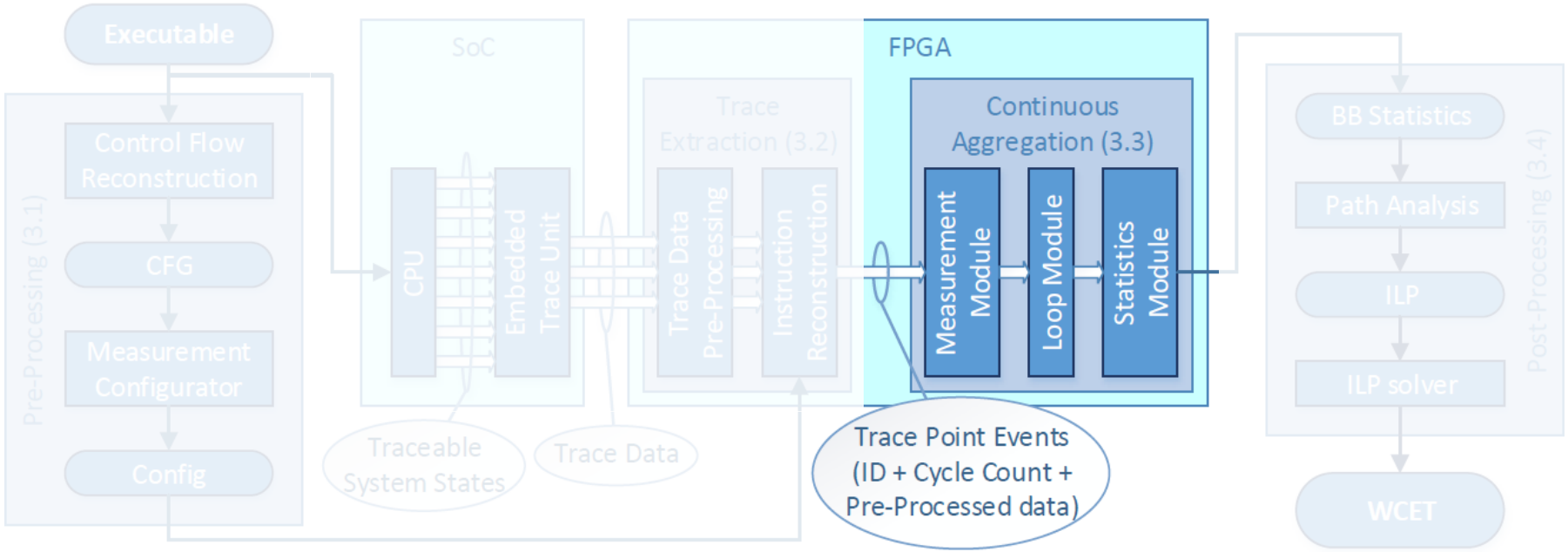


Measurement Configurator

```
void f (…)
{
  g(…);
}
void g (…)
{
  do {              // L1
    …;
    for (…;…;…)     // L2
      …;
    for (…;…;…)     // L3
      …;
  } while (…);
}
```



Workflow


Executable
Pre-Processing (3.1)
Control Flow Reconstruction
CFG
Measurement Configurator
Config
SoC
CPU
Embedded Trace Unit
Traceable System States
Trace Data
Trace Extraction (3.2)
Trace Data Pre-Processing
Instruction Reconstruction
FPGA
Continuous Aggregation (3.3)
Measurement Module
Loop Module
Statistics Module
Trace Point Events (ID + Cycle Count + Pre-Processed data)
Post-Processing (3.4)
BB Statistics
Path Analysis
ILP
ILP solver
WCET

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time of BB @ 0x1004 8 cycles

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time of BB @ 0x2034 2 cycles

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time
of BB @ 0x2034
1 cycle

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time of BB @ 0x2034 1 cycle

Pure maximisation gives 2 cycles.

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time of BB @ 0x2034
1 cycle

Pure maximisation gives 2 cycles.
We want to take typical cache behaviour into account!

Event Stream

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0
(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48
(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56
(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57
(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Execution Time of BB @ 0x2034
1 cycle

Pure maximisation gives 2 cycles.

Idea: Distinguish between first and all other loop iterations.

Loop Iteration State Machine

- A state is a data structure that keeps track of the call stack, the loop nesting level and whether we are in the first or in a later iteration of a loop.
- Calling a routine / returning from a routine changes the state.
- Entering / leaving a loop changes the state.

States

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

The pointer to the active row.

States

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

Is the row valid?

States

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

The id identifies a loop or a routine.

States

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

The index discriminates between first and further iterations.

State Transformations

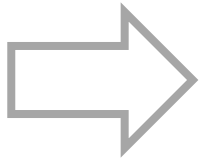
	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

(0x1004, 1, 0, 0, 1, 1, 0, 0) Routine Call,

(0x2000, 2, 1, 0, 0, 1, 0, 0) Increased Loop Nesting Level

State Transformations

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0



	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x1004, 1, 0, 0, 1, 1, 0, 0)
(0x2000, 2, 1, 0, 0, 1, 0, 0)

Routine Call,
Increased Loop Nesting Level

State Transformations

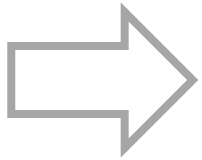
	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x2000, 2, 1, 0, 0, 1, 0, 0) Entering a loop

(0x202c, 3, 2, 0, 0, 0, 0, 0)

State Transformations

	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0



	valid	id	index
→	1	3	0
	1	2	0
	0	0	0
	1	1	0

(0x2000, 2, 1, 0, 0, 1, 0, 0)

(0x202c, 3, 2, 0, 0, 0, 0, 0)

Entering a loop

State Transformations

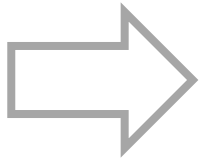
	valid	id	index
→	1	3	0
	1	2	0
	0	0	0
	1	1	0

(0x2034, 4, 2, 0, 0, 0, 0, 0)
(0x202c, 3, 2, 0, 0, 0, 0, 0)

Recursively entering a loop

State Transformations

	valid	id	index
→	1	3	0
	1	2	0
	0	0	0
	1	1	0



	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0

(0x2034, 4, 2, 0, 0, 0, 0, 0)
(0x202c, 3, 2, 0, 0, 0, 0, 0)

Recursively entering a loop

State Transformations

	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0

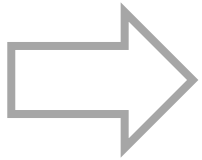
(0x202c, 3, 2, 0, 0, 0, 0, 0)

(0x2050, 5, 2, 1, 0, 0, 0, 0)

Entering a different loop

State Transformations

	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0



	valid	id	index
→	1	5	0
	1	2	0
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0)
(0x2050, 5, 2, 1, 0, 0, 0, 0)

Entering a different loop

State Transformations

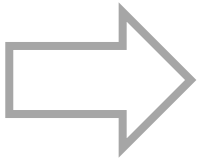
	valid	id	index
→	1	5	1
	1	2	0
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) Leaving a loop

(0x206c, 7, 1, 0, 0, 0, 0, 0)

State Transformations

	valid	id	index
→	1	5	1
	1	2	0
	0	0	0
	1	1	0



	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0)

(0x206c, 7, 1, 0, 0, 0, 0, 0)

Leaving a loop

State Transformations

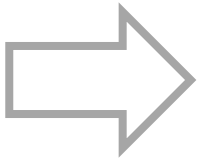
	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x206c, 7, 1, 0, 0, 0, 0, 0) Leaving a loop

(0x207c, 8, 0, 0, 0, 0, 1, 0)

State Transformations

	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0



	valid	id	index
	0	0	0
	0	0	0
→	1	8	0
	1	1	0

(0x206c, 7, 1, 0, 0, 0, 0, 0)
(0x207c, 8, 0, 0, 0, 0, 1, 0)

Leaving a loop

State Transformations

	valid	id	index
	0	0	0
	0	0	0
→	1	8	0
	1	1	0

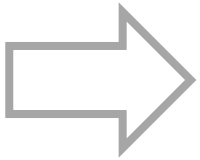
(0x207c, 8, 0, 0, 0, 0, 1, 0)

(0x101c, 9, 0, 0, 0, 0, 1, 1)

Leaving a routine

State Transformations

	valid	id	index
	0	0	0
	0	0	0
→	1	8	0
	1	1	0



	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

(0x207c, 8, 0, 0, 0, 0, 1, 0)

(0x101c, 9, 0, 0, 0, 0, 1, 1)

Leaving a routine

Example

Example

	first	further
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	0	0	0

(?, ?, ?, ?, 1, ?, ?, ?)

Example

	first	further
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

(?, ?, ?, ?, 1, ?, ?, ?)

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0

Example

	first	further
1	8	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x1004, 1, 0, 0, 1, 1, 0, 0) @ t+0

(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8

Example

	valid	id	index
→	1	3	0
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+8

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30

Example

	first	further
1	8	0
2	22	0
3	16	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
→	1	3	0
	1	2	0
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+30

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46

Example

	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	0
4	2	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+46

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48

Example

	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	8
4	2	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+48

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56

Example

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
→	1	3	1
	1	2	0
	0	0	0
	1	1	0

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+56

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57

Example

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

	valid	id	index
→	1	5	0
	1	2	0
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+57

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

Example

	valid	id	index
→	1	5	0
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	0
6	0	0
7	0	0
8	0	0
9	0	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+59

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+71

Example

	valid	id	index
→	1	5	1
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	0
6	2	0
7	0	0
8	0	0
9	0	0

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+71

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+73

Example

	valid	id	index
→	1	5	1
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	6
6	2	0
7	0	0
8	0	0
9	0	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+73

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+79

Example

	valid	id	index
→	1	5	1
	1	2	0
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	6
6	2	1
7	0	0
8	0	0
9	0	0

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+79

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+80

Example

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	6
6	2	1
7	0	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
→	1	2	0
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+80

(0x206c, 7, 1, 0, 0, 0, 0, 0) @ t+83

Example

	first	further
1	8	0
2	22	0
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
→	1	2	1
	0	0	0
	1	1	0

(0x206c, 7, 1, 0, 0, 0, 0, 0) @ t+83

(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+91

Example

	valid	id	index
→	1	3	0
	1	2	1
	0	0	0
	1	1	0

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

(0x2000, 2, 1, 0, 0, 1, 0, 0) @ t+91

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+102

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	3	0
	1	2	1
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+102

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+110

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	3	1
	1	2	1
	0	0	0
	1	1	0

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+110

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+112

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	3	1
	1	2	1
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+112

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+119

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	3	1
	1	2	1
	0	0	0
	1	1	0

(0x2034, 4, 2, 0, 0, 0, 0, 0) @ t+119

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+120

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	5	0
	1	2	1
	0	0	0
	1	1	0

(0x202c, 3, 2, 0, 0, 0, 0, 0) @ t+120

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+122

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	5	0
	1	2	1
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+122

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+126

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	6
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	5	1
	1	2	1
	0	0	0
	1	1	0

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+126

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+127

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	5	1
	1	2	1
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+127

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+135

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
→	1	5	1
	1	2	1
	0	0	0
	1	1	0

(0x205c, 6, 2, 1, 0, 0, 0, 0) @ t+135

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+136

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	0
8	0	0
9	0	0

	valid	id	index
	0	0	0
→	1	2	1
	0	0	0
	1	1	0

(0x2050, 5, 2, 1, 0, 0, 0, 0) @ t+136

(0x206c, 7, 1, 0, 0, 0, 0, 0) @ t+139

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	4
8	0	0
9	0	0

	valid	id	index
	0	0	0
	0	0	0
→	1	8	0
	1	1	0

(0x206c, 7, 1, 0, 0, 0, 0, 0) @ t+139

(0x207c, 8, 0, 0, 0, 0, 1, 0) @ t+143

Example

	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	4
8	6	0
9	0	0

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	1	1	0

(0x207c, 8, 0, 0, 0, 0, 1, 0) @ t+143

(0x101c, 9, 0, 0, 0, 0, 1, 1) @ t+149

Example

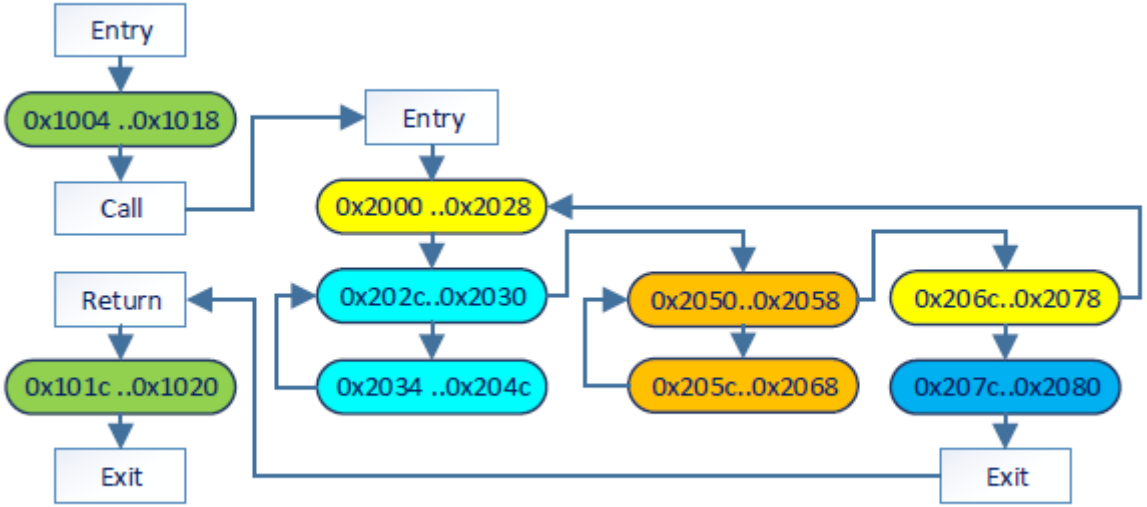
	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	4
8	6	0
9	6	0

	valid	id	index
	0	0	0
	0	0	0
	0	0	0
→	0	0	0

(0x101c, 9, 0, 0, 0, 0, 1, 1) @ t+149

(?, ?, ?, ?, ?, ?, ?, 1) @ t+155

Example



	first	further
1	8	0
2	22	11
3	16	8
4	2	1
5	12	8
6	2	1
7	8	4
8	6	0
9	6	0

Overall execution time estimate:

- Our method: 191 cycles
- Context insensitive: 258 cycles

Implementation

FPGA Implementation

- We are currently building a prototype based on a Xilinx Virtex 7 FPGA (xc7v585t) and RLDRAM.
- Most complicated part is the connection to the SoC's trace interface (i.e. event generation).

Resource Usage on FPGA

Loop Iteration State Machine (~ 180 MHz)

- 3 BRAMs
- 470 Registers
- 398 LUTs

Statistics Module (~ 130 MHz)

- 7 BRAMs
- 481 Registers
- 406 LUTs

Conclusion

The CONIRAS Approach

Precise

- We account for typical (instruction) cache behaviour.

Continuous

- We perform direct online aggretation at runtime.

Non-intrusive

- We use the hardware support of modern SoCs.

Measurement-Based Execution Time Estimation

The CONIRAS Approach

⇒ Reduced pessimism

⇒ Improved observability

⇒ Scalability

⇒ Statistics instead of piles of data


AbsInt