Department of Computer Science

pls Programmierbare Logik & Systeme GmbH

A High-Level Language and Compiler to Configure the Multi-Core Debug Solution

Jens Braunes
pls Development Tools

Rainer G. Spallek
Dresden University of Technology

Überblick

1. Teil

Debug-Technologie: On-Chip Trace

Einordnung, Herausforderungen

2. Teil

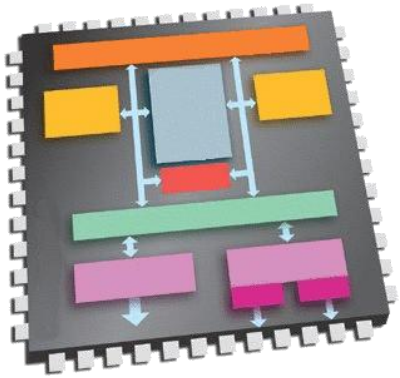
Softwareunterstützung für MCDS

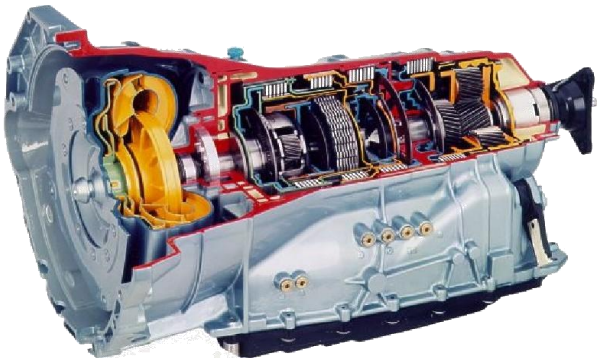
A High-Level Language and Compiler to Configure the Multi-Core Debug Solution (VALID'09)

1. TEIL: DEBUG-TECHNOLOGIE: ON-CHIP TRACE

Einordnung, Herausforderungen

Debug-Szenarien eingebetteter Systeme

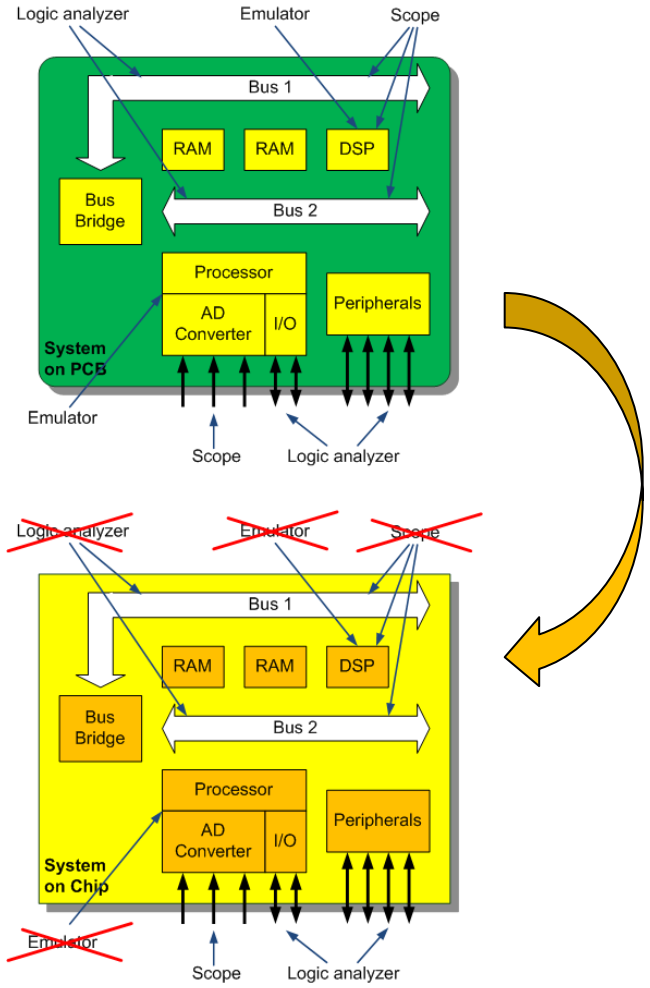




- Debugging komplexer Multi-Core SoCs
  - CPU + DSP System
  - Multi-Bus
  - Multi-CPU System
  - Multi-DSP System
- Debugging eingebetteter Software zur Laufzeit:
  - Motor im Laborbetrieb (Labcar)
  - Fahrzeug auf Teststrecke
- Entfernung zw. Target und Debugger
  - Kabel und Schnittstellen
  - Zugang zum Target
- Code Profiling, Performance Analyse

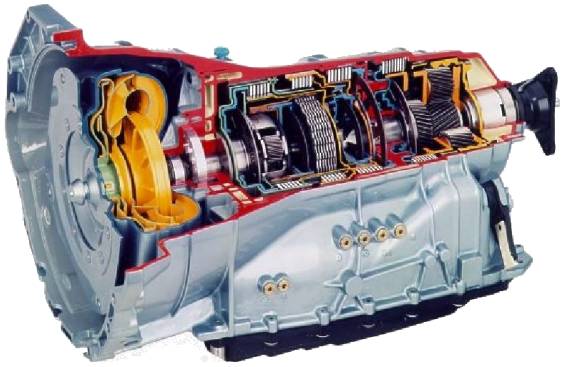
Problemstellungen und Herausforderungen

- Sichtbarkeit / Beobachtbarkeit
  - SoC hat keine „natürlichen" Zugangspunkte
  - Kein Zugang zu internen Bussen od. Schnittstellen



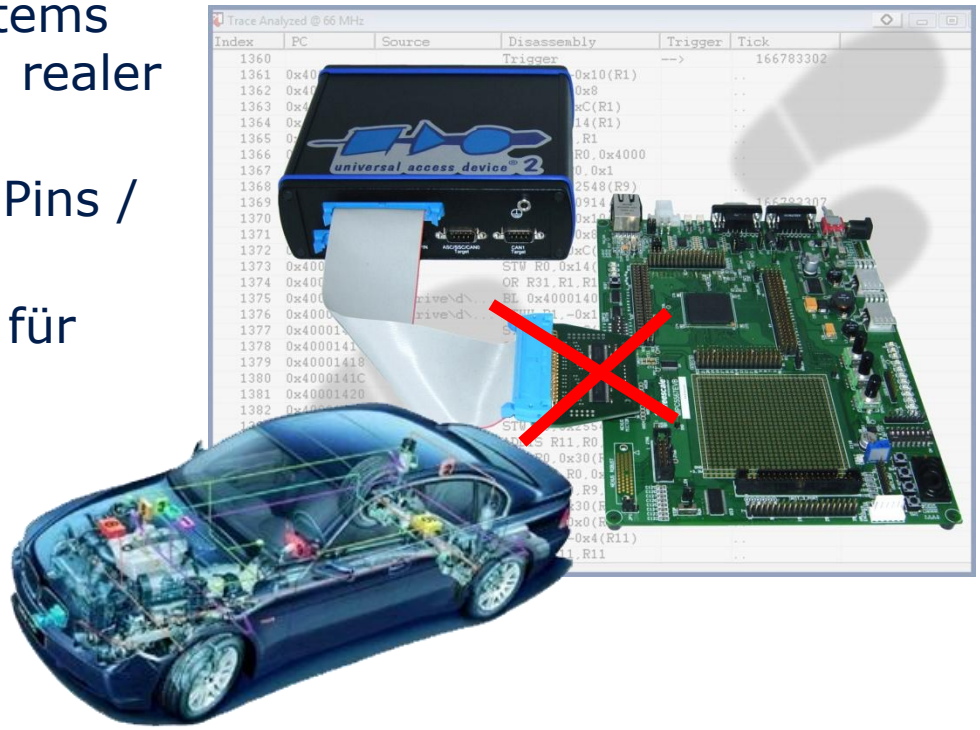
Problemstellungen und Herausforderungen

- „Echtzeitfähigkeit“ – Debugging kritischer Systeme zur Laufzeit
  - Beeinflussung des Laufzeitverhaltes durch den Debugger, z.B.
    - gemeinsamer Buszugriff
    - Halt für Zustandsabfrage
  - Debugging innerhalb der „natürlichen“ Umgebung
    - Motor im Laborbetrieb (Labcar)
    - Fahrzeug auf Teststrecke



Problemstellungen und Herausforderungen

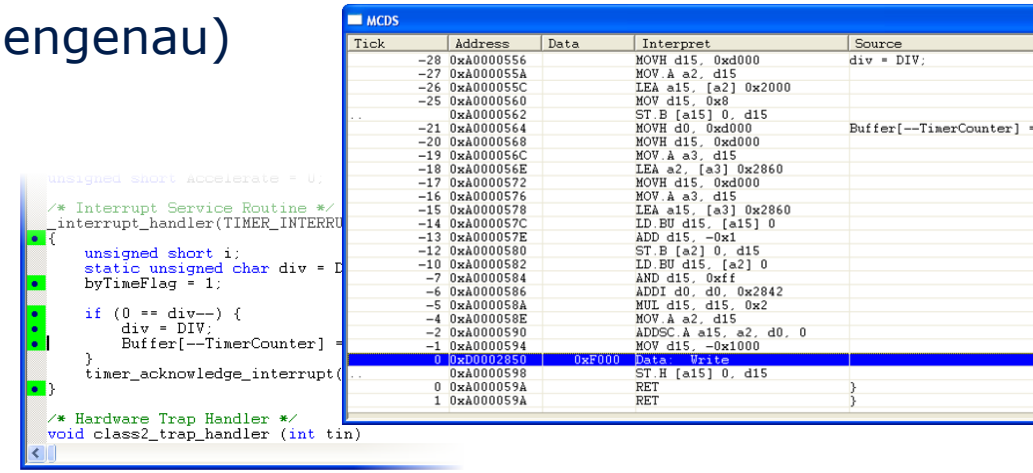
- Debuggen des realen Systems (z.B. Motorsteuergerät) in realer Umgebung
  - Begrenzte Anzahl der Pins / Leitungen nach außen
  - Begrenzte Bandbreite für Debugging (Trace)



↳ **On-Chip Debug Support mit Trace-Fähigkeiten und Low-Pin-Count Interface**

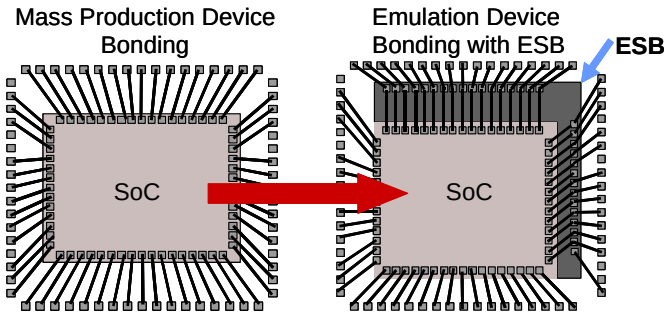
Multi-Core Debug Solution (MCDS)

- Systembeobachtung (zyklengenau)
  - Program Trace
  - Data Trace
  - Ownership Trace
  - Status Trace
  - Watchpoint Trace



- Non-invasive
  - Keine Beeinflussung des Laufzeitverhaltens

- On-Chip Trace Memory
  - Keine teure Trace-Schnittstelle
  - Gleiches Bond-Out mit und ohne MCDS



2. TEIL
SOFTWAREUNTERSTÜTZUNG FÜR MCDS

A High-Level Language and Compiler to Configure the Multi-Core Debug Solution

Debug Challenges and Solutions

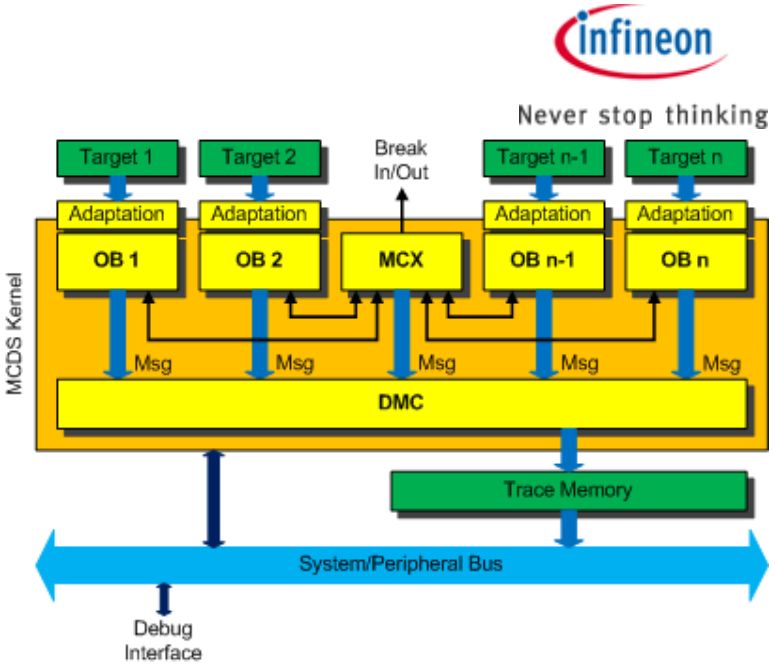
- New challenges when debugging and analyzing state-of-the-art SoCs:
  - Multi-Core / multi-bus
  - Applications with hard real-time demands
  - Multiple IPs – single chip: components not visible / accessible for debugging
  - Low pin counts (automotive and industrial applications)
- Solution: On-chip trace and debug units
  - High degree of observability without affecting the run-time behavior (non-invasive trace)
  - Configurable trace recording:
    - Definition of analysis tasks, tailored to the users' needs
    - Capture only relevant data to save trace memory and simplify analysis
- E.g. ARM CoreSight, Infineon's MCDS (Multi-Core Debug Solution)

Outline

- MCDS (Multi-Core Debug Solution)
- Tool support for MCDS – Objective
- High Level Language for Trace Qualification and Compiler
- Case Study
- Conclusion

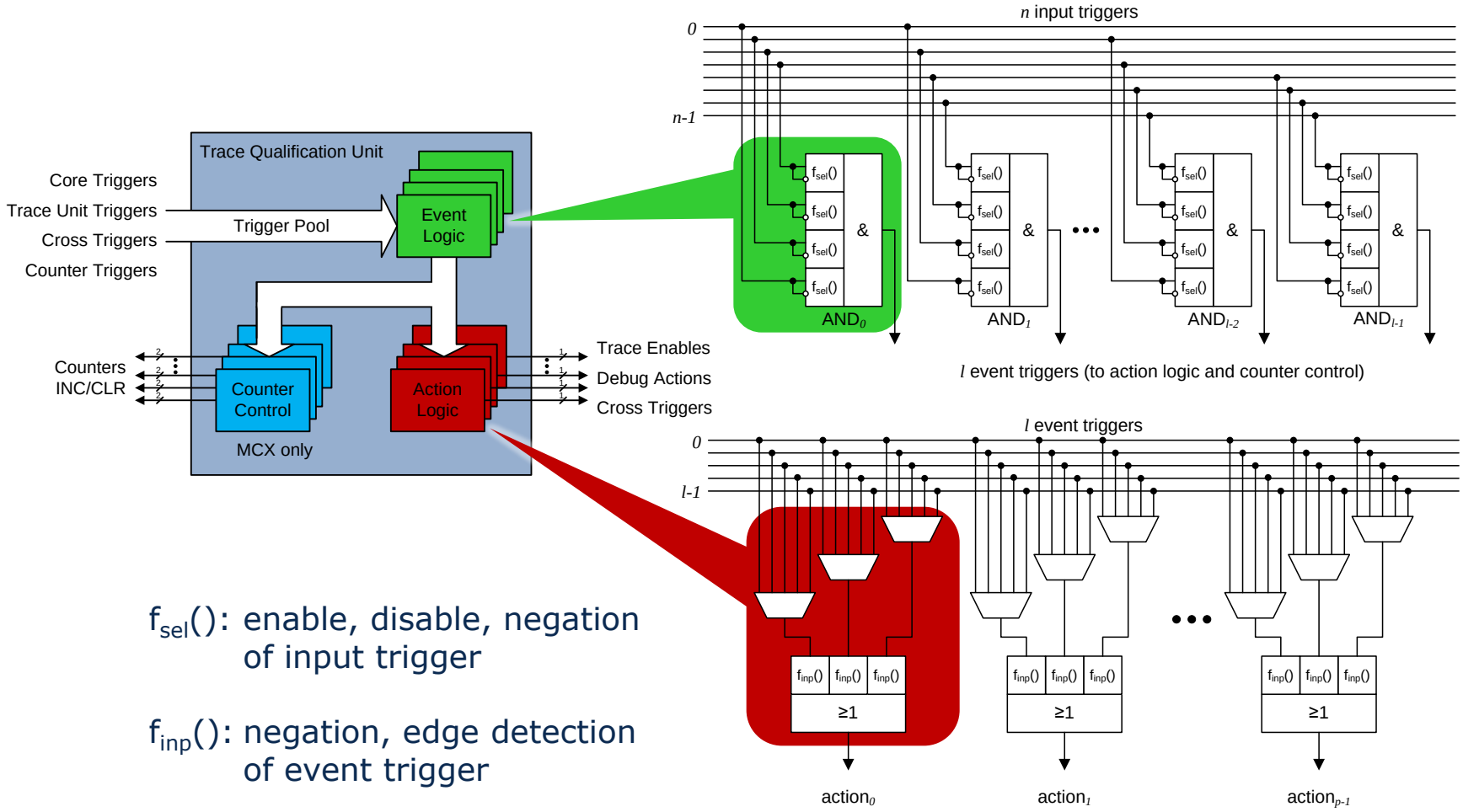
Multi-Core Debug Solution (MCDS)

- Configurable and scalable trigger, trace qualification, and trace compression IP block (by Infineon)
- For multi-core / multi-bus SoCs
- Simultaneous recording of multiple trace sources – one single, time aligned trace stream
- On-chip trace memory to overcome trace port bottleneck
- Record only relevant data → complex trace qualification and trigger logic
- Cross triggers to distribute events between OBs
- ~500 registers for configuration



OB	observation block
MCX	multi-core cross connect
DMC	data management controller
Msg	trace messages

Trace Qualification and Trigger Logic



$f_{sel}()$: enable, disable, negation of input trigger

$f_{inp}()$: negation, edge detection of event trigger

Objective

- Create an interface to configure trace and debug tasks for MCDS
- Challenges:
  - Multiple trace sources – one single trace stream
  - Multi-Core / multi-bus → requires cross triggering
  - Powerful but complex trace qualification and trigger logic
- Approach:
  - High level language and compiler for trace qualification
  - Provides overall system view – no separation between trace sources
  - Supports state machines to define complex but manageable analysis tasks

High Level Language for Trace Qualification

- Quite similar to high level programming languages
- Statements to initiate debug and trace actions (e.g. enabling trace recording, generating core events, break targets, …)
- if-else to enable actions based on conditions
- Statements to build state machines (state labels, goto).

```
TPC.PCcore1 pc1;
TPC.PCcore2 pc2;
TSignal Enter = pc1 == 0xD400049A;
TSignal Leave = pc2 == 0xD40005BC;
TPC.BusAddr addr;
SharedMem = addr == 0xF0050040;
// Write to bus from any busmaster:
TData.BusAccess access;
MemWrite = 0x200<=(access & 0x20e)<=0x20F;
TData.BusData data;

// State machine
state0:
   if (Enter) then
      goto state0;
state1:
   store pc1;
   store pc2;
   emit mcx.tick_enable;
   if (MemWrite and SharedMem) {
      trigger;
      store addr;
      store data;
   }
   if (Leave) {
      goto state0;
   }
```

Transforming State Machines

- State machines not directly supported – counters (MCX) represent states
- Transformation into combinatorial logic:
  - Transformation of edge detection (if needed):

    $x_1 \wedge \text{rise}(x_2) \wedge x_3 \ldots x_n \rightarrow \text{rise}(x_1 \wedge x_2 \wedge x_3 \ldots x_n)$

    y becomes true (high) if at least one input x_i has a transition from low to high meanwhile all other inputs are already high.
  - Combinatorial terms → canonical form
  - Simplification

```
s1:
  if (x1) {
    act1;
    goto s2;
  } else {
    act2;
  }
s2:
  if (x2) {
    act1;
    act2;
  }
```

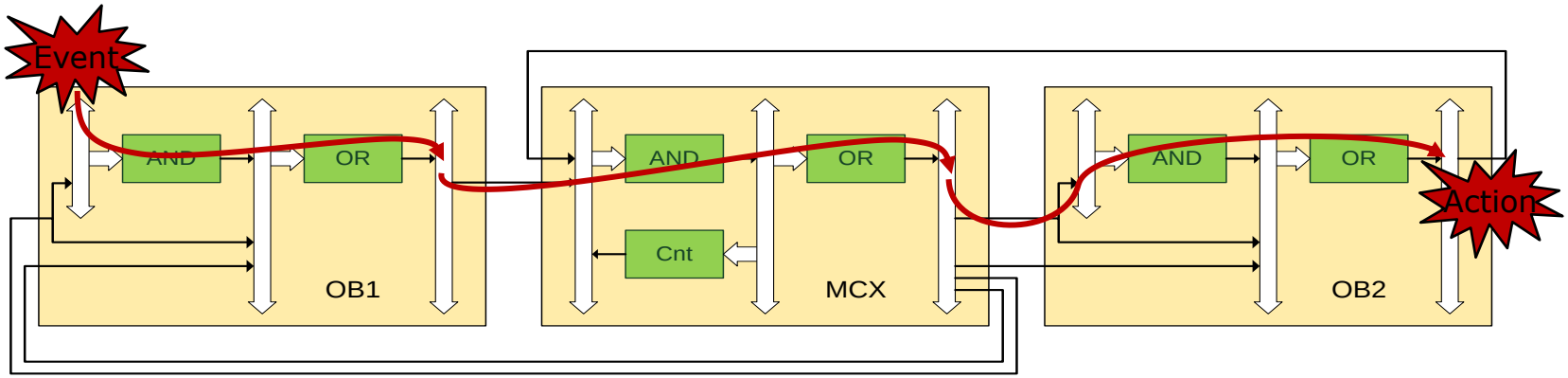
Combinatorial terms:

$y_{goto} = (s1 \wedge x1) \vee (s2 \wedge x2)$

$y_{act1} = (s1 \wedge x1) \vee (s2 \wedge x2)$

$y_{act2} = (s1 \wedge \overline{x1}) \vee (s2 \wedge x2)$

Cross Trigger Routing



$$y_{OB1} = (x_{1\ OB1} \wedge x_{2\ OB2}) \vee (x_{3\ OB2} \wedge x_{4\ OB2})$$

$$ct_{1\ OB2\rightarrow MCX} = x_{2\ OB2}$$

$$ct_{3\ MCX\rightarrow OB1} = ct_{1\ OB2\rightarrow MCX}$$

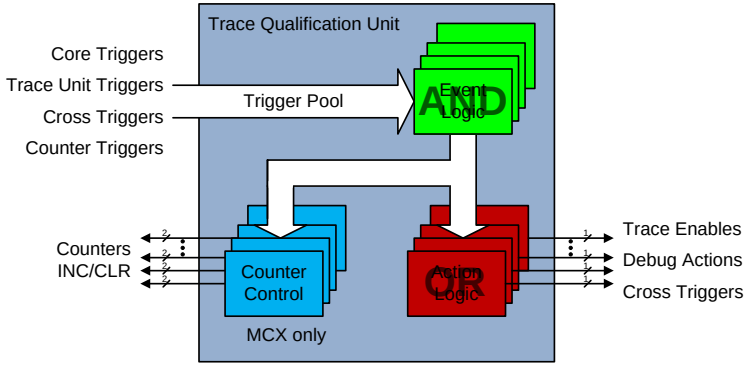
$$ct_{2\ OB2\rightarrow MCX} = (x_{3\ OB2} \wedge x_{4\ OB2})$$

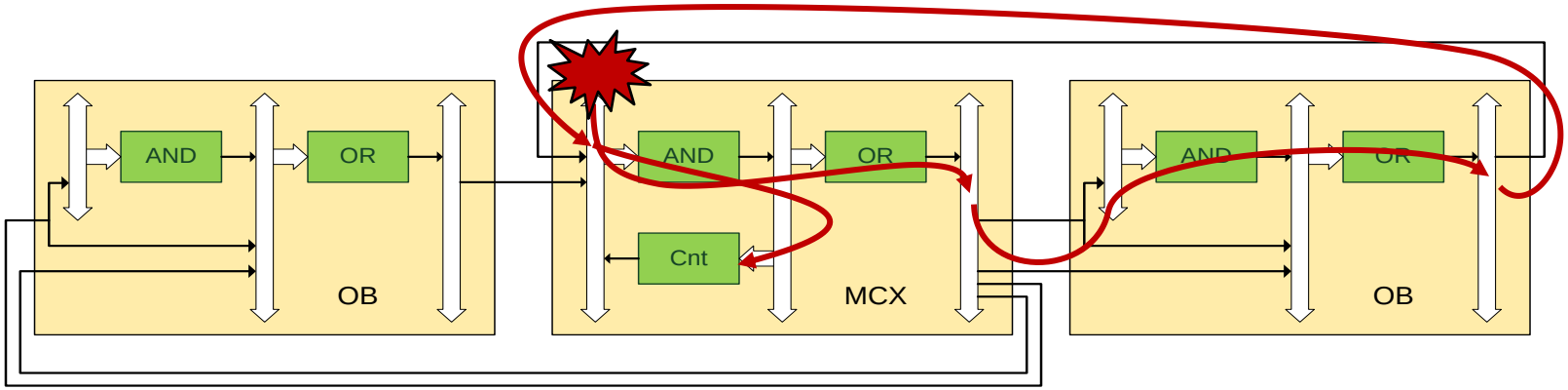
$$ct_{4\ MCX\rightarrow OB1} = ct_{2\ OB2\rightarrow MCX}$$

$$y_{OB1} = (x_{1\ OB1} \wedge ct_{3\ MCX\rightarrow OB1}) \vee ct_{4\ MCX\rightarrow OB1}$$

Cross Trigger Routing

- Managing the lack of "OR" functionality in Counter Control
  - Use trace qualification logic of spare OB
  - At the expense of timing





Resource Management

Two critical resources in MCDS:

1. **Fixed assignment of input events to AND blocks**
   Find that AND block which is utilized best for a given conjunction of input triggers

Resource Management

Two critical resources in MCDS:

2. Limited number of cross trigger lines between OBs and MCX

Reuse of cross trigger lines

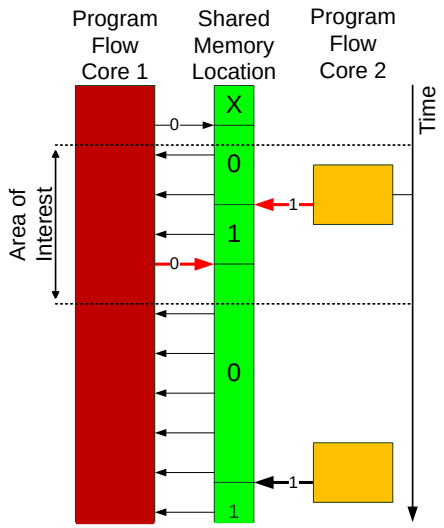
$$y_{OB2} = x_{1\ OB1} \wedge X_{2\ OB1} \wedge X_{3\ MCX}$$
$$y_{MCX} = x_{1\ OB1} \wedge X_{2\ OB1} \wedge X_{3\ MCX}$$

$\longrightarrow$

$$ct_{OB1 \rightarrow MCX} = x_{1\ OB1} \wedge X_{2\ OB1}$$
$$ct_{MCX \rightarrow OB2} = ct_{OB1 \rightarrow MCX} \wedge X_{3\ MCX}$$
$$y_{OB2} = ct_{MCX \rightarrow OB2}$$
$$y_{MCX} = ct_{OB1 \rightarrow MCX} \wedge X_{3\ MCX}$$

Case Study

- Multi-core interaction via shared memory




```
TPC.PCcore1 pc1;
TPC.PCcore2 pc2;
TSignal Enter = pc1 == 0xD400049A;
TSignal Leave = pc2 == 0xD40005BC;
TPC.BusAddr addr;
SharedMem = addr == 0xF0050040;
// Write to bus from any busmaster:
TData.BusAccess access;
MemWrite = 0x200<=(access & 0x20e)<=0x20F;
TData.BusData data;

// State machine
state0:
   if (Enter) then
      goto state0;
state1:
   store pc1;
   store pc2;
   emit mcx.tick_enable;
   if (MemWrite and SharedMem) {
      trigger;
      store addr;
      store data;
   }
   if (Leave) {
      goto state0;
   }
```

Case Study

- Multi-core interaction via shared memory

Source file characteristics	
# of debug actions	6
# of states	2
# of state transitions	2
# of input events[1]	4
# of conditions	3

[1] From trigger pools of all used OBs.

```
TPC.PCcore1 pc1;
TPC.PCcore2 pc2;
TSignal Enter = pc1 == 0xD400049A;
TSignal Leave = pc2 == 0xD40005BC;
TPC.BusAddr addr;
SharedMem = addr == 0xF0050040;
// Write to bus from any busmaster:
TData.BusAccess access;
MemWrite = 0x200<=(access & 0x20e)<=0x20F;
TData.BusData data;

// State machine
state0:
   if (Enter) then
      goto state0;
state1:
   store pc1;
   store pc2;
   emit mcx.tick_enable;
   if (MemWrite and SharedMem) {
      trigger;
      store addr;
      store data;
   }
   if (Leave) {
      goto state0;
   }
```

Case Study

```
MCDS_TCACT17    = 0xC0000000
MCDS_MCXACT4    = 0xC4000000
MCDS_TCACT4     = 0xD4000000
MCDS_TCACT5     = 0xD4000000
MCDS_MCXACT44   = 0xC4000000
MCDS_PCPACT4    = 0xD4000000
MCDS_PCPACT5    = 0xD4000000
MCDS_MCXACT28   = 0xC4000000
MCDS_SPBACT10   = 0xC0000000
MCDS_MCXACT29   = 0xC9000000
MCDS_MCXACT49   = 0xC4000000
MCDS_SPBACT3    = 0xC1000000
MCDS_SPBACT5    = 0xC1000000
MCDS_SPBACT2    = 0xC1000000
MCDS_SPBACT4    = 0xC1000000
MCDS_TCACT16    = 0xC1000000
MCDS_TCEVT0     = 0xFFFFFBFF
MCDS_MCXEVT8    = 0xF7FFFEFF
MCDS_MCXEVT8    = 0xF7FFFEFF
MCDS_MCXEVT4    = 0xFFFFFFFE
MCDS_MCXEVT4    = 0xFFFFFFFE
MCDS_MCXEVT4    = 0xFFFFFFFE
MCDS_SPBEVT0    = 0xFFFEFFFE
MCDS_MCXEVT9    = 0xFEFFBFFF
MCDS_MCXEVT4    = 0xFFFFFFFE
MCDS_SPBEVT1    = 0xEFFEFFFE
MCDS_SPBEVT1    = 0xEFFEFFFE
MCDS_SPBEVT1    = 0xEFFEFFFE
MCDS_SPBEVT1    = 0xEFFEFFFE
MCDS_TCEVT1     = 0xFFFFEFFF
MCDS_MCXEVT10   = 0xFEFFFFBF
MCDS_MCXEVT10   = 0xFEFFFFBF
MCDS_MCXLMT1    = 0x0
MCDS_MCXLMT0    = 0x0
MCDS_TCIPBND0   = 0xD400049A
MCDS_TCIPRNG0   = 0x2
MCDS_SPBACBND0  = 0x200
MCDS_SPBACRNG0  = 0xF
MCDS_SPBACMSK0  = 0x20E
MCDS_SPBEABND0  = 0xF0050040
MCDS_SPBEARNG0  = 0x0
MCDS_TCIPBND1   = 0xD40005BC
MCDS_TCIPRNG1   = 0x2
MCDS_MCXCCL1    = 0x8A88
MCDS_MCXCCL0    = 0x8A88
```

Characteristics of compilation results	
# of occupied debug blocks	4[1]
# of additional input events	2[2]
# of cross triggers[3]	6
Cross trigger reuse rate[4]	2.2
# of different action conditions	7
# of MCDS registers to set	46

1 Two processor observation blocks, one bus observation block and MCX.

2 Two counters representing the states.

3 Adds one additional condition each.

4 Cross triggers used by 13 action conditions.

Conclusion

- MCDS provides a high degree of system observability without affecting the run-time behavior
- Powerful trace qualification and trigger logic requires tool support
- High level language and compiler for trace qualification
  - Provides overall system view – no separation between trace sources
  - Supports state machines to define complex but manageable analysis tasks
  - Compiler rids the user from dealing with cross triggers and resources management
- MCDS as well as the configuration language and compiler are already in industrial use where effective debug support is essential for better product quality and shorter product development cycles.