Alexander Weiss & Thomas Preußer
Accemic Technologies

MIPI HTI(SM), PTI(SM) and STP(SM) –
The Bases for Next-Generation Analyses of Multicore Processors
Motivation
Motivation: Complex Systems Challenges

Burkacky 2018 (McKinsey Center for Future Mobility®):
“Snowballing complexity is causing significant software-related quality issues, as evidenced by millions of recent vehicle recalls.”

Where software is, there are errors.

The defect potential increases with system complexity.
The defect removal efficiency decreases with increasing system complexity.

Growing and self-reinforcing problem

Key capability: System observation
Embedded System Observation

Challenges
• Real-time Control
• Concurrency
• Elusive Heisenbugs
• Rare Mandelbugs

Key 1: Non-Intrusiveness
• Maintain application timing
• Avoid phantom synchronization

Key 2: Continuity
• No systematic observation limits
• Online trace data processing

Debug
Health Monitoring
Runtime Verification at System Level

Log & Store
SW Instrumentation

printf(…)

Probe Effect
State of Affairs
The Crux of System Integration

The Past

System Bus Probe

Instruction Memory (ROM)

Data Memory (RAM)

Logic Analyzer

- Instruction Addresses
- Data Access
- Timing

Processor

CPU

Peripherals

Need

- On-Chip Probes
- Data Filters
- High-Speed Interfaces

The Present

Data Volume
- Multiple players
- Working at GHz speeds.

Clueless System Bus
- Caches hide memory access.
- On-chip memory swallows all access info.
- Multicore systems prevent effect attribution.
Embedded Trace Options

**Option 1 - Intrusive**
- Compete with application for memory bandwidth and space.

**Option 2 - Bounded**
- Size of trace buffer limits time.

**Option 3 - Hard**
- Processing must match line rate.
MIPI Protocols in Action
Carrying Execution Trace Data Off-Chip

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STP</td>
<td>MIPI System Trace Protocol</td>
</tr>
<tr>
<td>HTI</td>
<td>MIPI High-Speed Trace Interface</td>
</tr>
<tr>
<td>TWP(SM)</td>
<td>MIPI Trace Wrapper Protocol</td>
</tr>
<tr>
<td>PTI</td>
<td>MIPI Parallel Trace Interface</td>
</tr>
</tbody>
</table>
# MIPI PHY + Link Layer Technologies

## PTI
- Parallel little-endian TRC_DATA (up to 32)
- Synchronous to shared TRC_CLK (up to ~300 MHz)
- Mechanisms for:
  - Signal skew calibration,
  - Physical, by-pin slicing for parallel point-to-point streams, and
  - Data striping for mismatched protocol/interface bit widths.

## HTI
- Trace transmission over (multiple) high-speed serial lanes.
- **Aurora** 8B/10B simplex with up to 6 lanes (HTI) or 8 lanes (HTIv1).
- Up to 12.5 Gbps per lane.
- UFC Messages for sideband info: trigger, overflow, link health, aux.
- Session management (incl. disabled, init, idle, run, test).
MIPI Transport Layer Technologies

STP – System Trace Protocol
- Multiplexing among up to 65536 sources at the granularity of 4-bit nibbles.
- Autarchic timestamping and syncing.
- Sideband packets for user payload and flagging.
- Stream synchronization facilities.
- Data integrity protection.

TWP – Trace Wrapper Protocol
- Multiplexing among up to 111 sources at byte granularity.
- Designed for very low overhead.
Peering with the ARM CoreSight Stack

TWP = TPIU Formatter

HTI = HSSTP over Aurora
Recap: Key Embedded Trace Properties

Key 1: Non-Intrusiveness
• Maintain application timing
• Avoid phantom synchronization
• MIPI Trace protocols and interfaces
• Standard interfaces for (non-intrusive) trace output (USB, Ethernet, PCIe)

Key 2: Continuity
• No systematic observation limits
• Online trace data processing
Enabling Online Trace Data Processing
Benefits & Use Cases

Benefits

• Evade all observation time limits.
• Provide instantaneous reaction opportunities.
• Avoid intrusive SW instrumentation.

Use Cases

• Continuous runtime verification:
  – Formal constraints-based anomaly detection.
  – Complex triggers on rare Mandelbug occurrences.
• Structural coverage measurements of high-level integration & system tests.
A Taste of the Challenge

```c
// Count all encounters of 42
static unsigned matches = 0;
static void count42(int const a) {
    if(a == 42) matches++;
}

int main() {
    ...
    for(int i = 0; i < n; i++) {
        int arg = f(i, ...);
        count42(arg);
    }
    printf("Matches:%0d\n", matches);
}
```

- Occasional synchronization
- Only branches and diversions:
  - Taken / not taken indicators
  - Delta-encoded indirect targets

<table>
<thead>
<tr>
<th>Resolved Return</th>
<th>Compressed Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSB</td>
<td>PSB</td>
</tr>
<tr>
<td>FUP 0x400220</td>
<td>FUP 0x400220</td>
</tr>
<tr>
<td></td>
<td>– Synchronization –</td>
</tr>
<tr>
<td></td>
<td>– Synchronization –</td>
</tr>
<tr>
<td>PSBEND</td>
<td>PSBEND</td>
</tr>
<tr>
<td>TNT N</td>
<td>TNT N</td>
</tr>
<tr>
<td></td>
<td>– Conditional branch @0x400103 not taken –</td>
</tr>
<tr>
<td>TIP 0x___0228</td>
<td>TIP 0x___0228</td>
</tr>
<tr>
<td></td>
<td>– Return –</td>
</tr>
<tr>
<td>TNT T</td>
<td>TNT T</td>
</tr>
<tr>
<td></td>
<td>– Conditional branch @0x40022C taken (loop) –</td>
</tr>
</tbody>
</table>
Setup for Measuring the Structural Coverage
Structural Coverage Reporting

```c
#define CEDAR
#include "cedar_ipt.h"
#endif

#include <stdlib.h>
#include <stdio.h>

struct bin_tree {
    int data;
    struct bin_tree * right, *left;
};
typedef struct bin_tree node;

void insert(node ** tree, int val) {
    node *temp = NULL;
    if (!(*tree)) {
        temp = (node *)malloc(sizeof(node));
        temp->right = temp->right = NULL;
        temp->data = val;
        *tree = temp;
        return;
    }
    if (val < (*tree)->data) {
        insert(&(*tree)->left, val);
    } else {
        insert(&(*tree)->right, val);
    }
}```
Higher-Level Coverage Metrics for Safety Certification

- What branches constitute a decision?
- What paths were part of an execution?
- Does the path set satisfy MC/DC coverage?

```c
if((a&&b) || c) {
    ...
    ...
}
```

```
40121C: cmpl $0x0,-0x24(%rbp)
401220: je 401228 <main+0x106>
401222: cmpl $0x0,-0x28(%rbp)
401226: jne 40122e <main+0x10c>
401228: cmpl $0x0,-0x2c(%rbp)
40122C: je 401234 <main+0x112>
```
Conclusion
### Importance of MIPI Standards

- Capable vehicles for architecture-independent trace transport.
- Enable **reuse** of lower-level trace layers.
- Allow vendors to **focus** efforts on added high-level functionality.
- Accelerate innovation.
- Reduce time to market.

### Challenges for Online Trace Data Processing

- Standardisation of COTS-based trace interfaces
- Increase available trace bandwidth
- Unbounded timing slack between multiplexed streams (TWP/STP)
- Defined buffering limits on carried protocols would be helpful
ADDITIONAL RESOURCES

• C. Jones and O. Bonsignour, The Economics of Software Quality. Addison-Wesley, 2011. https://lmy.de/5gXVT
• Thomas Preuβer et al.: Everything You Always Wanted to Know About Embedded Trace, accepted by IEEE Computer, preprint available at https://lmy.de/l4Hs5
• The CEDARtools online monitoring system: https://lmy.de/9vAuL
THANK YOU!