A hardware coprocessor for Zynq-based Dynamic Information Flow Tracking

CryptArchi 2016

June 22, La Grande Motte

Muhammad Abdul WAHAB

Supervisors:

Christophe MOY

Pascal COTRET

Outline

Context: Dynamic Information Flow Tracking

Related Work

Coresight components

Proposed Global Approach

Conclusion

Motivation

Security

- Software security is still a major threat
- More systems are online, vulnerable
- Threats have multiplied

Need an approach that is

- Flexible
- Practical
- Fast

DIFT: Dynamic Information Flow Tracking

Information flow

Information flow is the transfer of information from an information container c_1 to c_2 in a given process P.

$$c_1 \underset{P}{\rightarrow} c_2$$

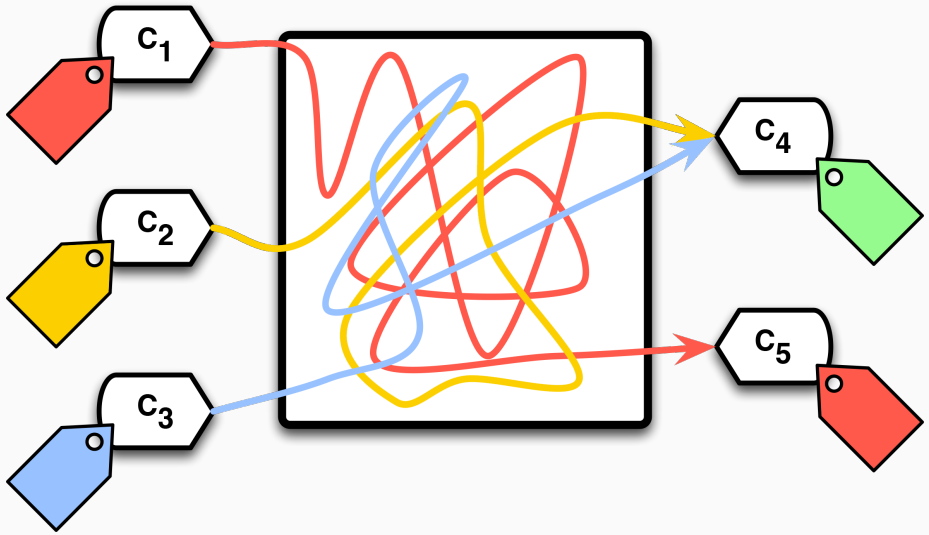
Example

```
int p = 3;
int s = 42;
int x = p + s;
```

DIFT: Dynamic Information Flow Tracking

Principle

- We attach labels called tags to such containers and specify an information flow policy, i.e. relations between tags
- At runtime, we propagate tags to reflect information flows that occur and detect any policy violation



DIFT: Dynamic Information Flow Tracking

DIFT principle

1. DIFT taints data from untrusted sources
   - Extra tag bit per word marks if untrusted
2. Propagate taint during program execution
   - Operations with tainted data produce tainted results
3. Check for unsafe uses of tainted data

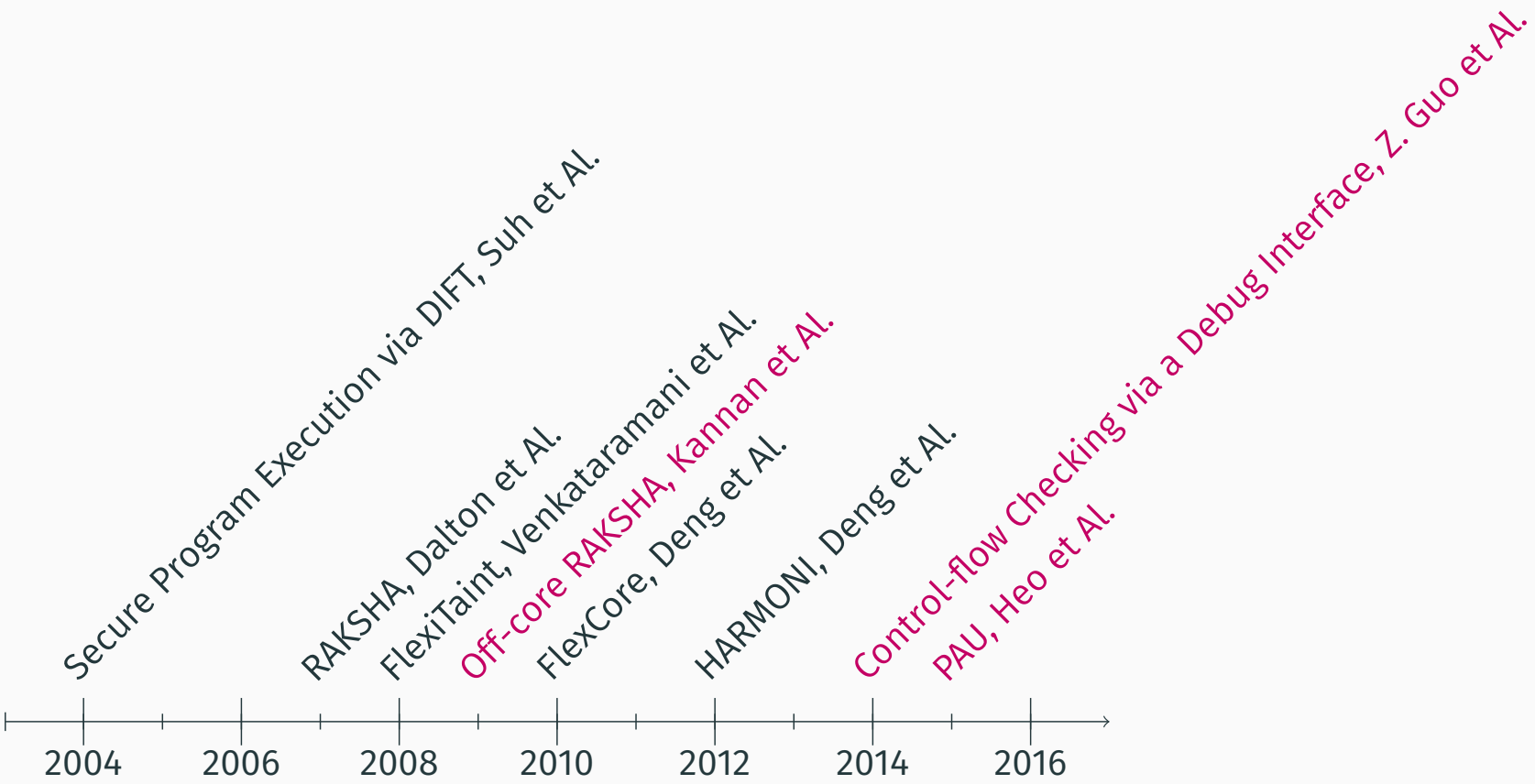
Motivation

Protection from low-level and high-level threats

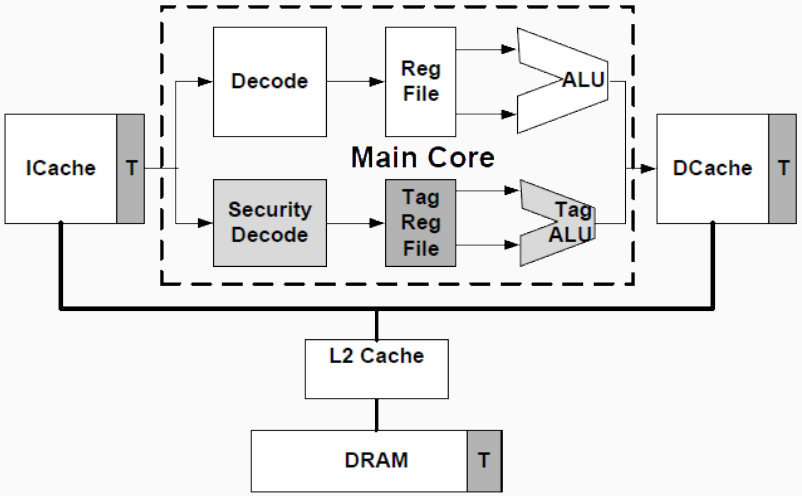
DIFT: Example

Example code	Tag initialization	Tag propagation	Tag check
p = 3;	$\underline{p} \leftarrow$ public		
s = 42;	$\underline{s} \leftarrow$ secret		
x = p + s;		$\underline{x} \leftarrow \underline{p} + \underline{s} = \underline{s}$;	
print(x);			if ($\underline{x}$!= public) `raise interruption`

Recent works on DIFT



Related work



In-core DIFT[1]

Pro

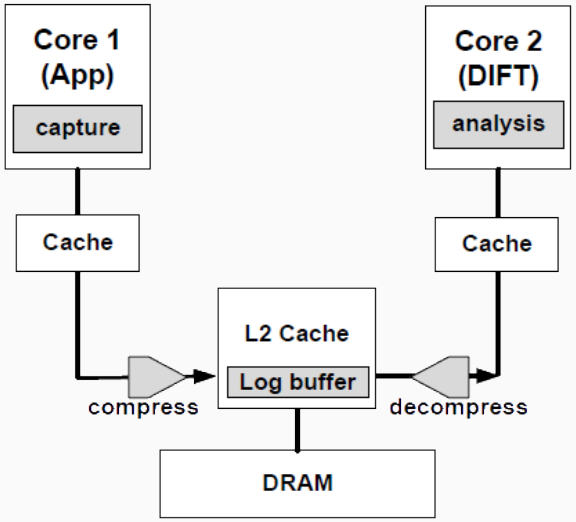
- Minimal impact on speed

Cons

- Tag propagation in core pipeline
- Not portable

[1] Dalton, Kannan, and Kozyrakis, “Raksha: A Flexible Information Flow Architecture for Software Security”

Related work



Offloading DIFT[2]

Pros

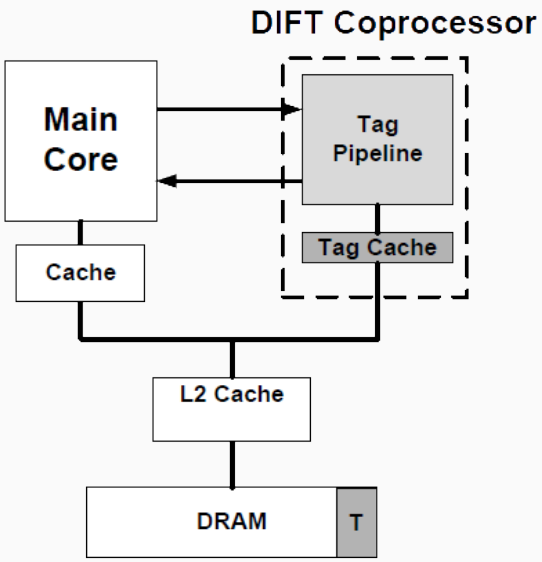
- Separate processor for DIFC controls
- Tag compression/de-compression

Cons

- Inter-core logic
- Power consumption x 2

[2] Nagarajan et al., “Dynamic Information Tracking on Multicores”

Related work



Off-core DIFT[3]

Pros

- Detached coprocessor
- Nearly no modification of the main processor

Cons

- Main core may stall
- Frequency issues

[3] Kannan, Dalton, and Kozyrakis, “Decoupling dynamic information flow tracking with a dedicated coprocessor”

Summary

DIFT Approaches	In-core[1]	Offloading[2]	Off-core[3]
Hardcore portability	-	+	++
Time Overhead	+	-	++
Surface Overhead	+	-	+
Main CPU	Softcore (Leon 3)		
Frequency	Same for both cores		

Objectives

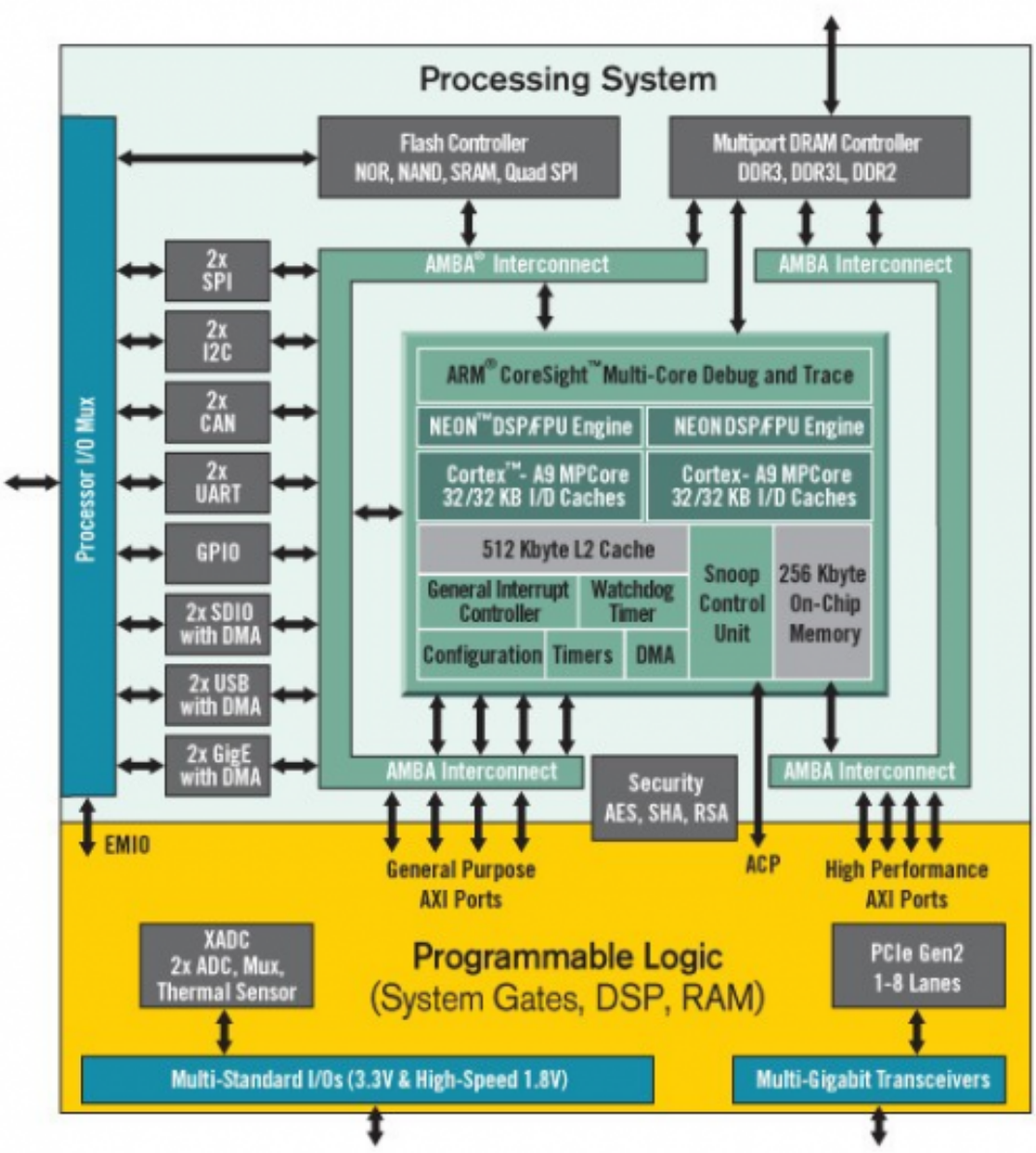
Main Goals

- Approach based on a non-modified CPU with a standard Linux and generic binaries
- Flexible and portable solution
- Low performance overhead
- Few false positives/negatives

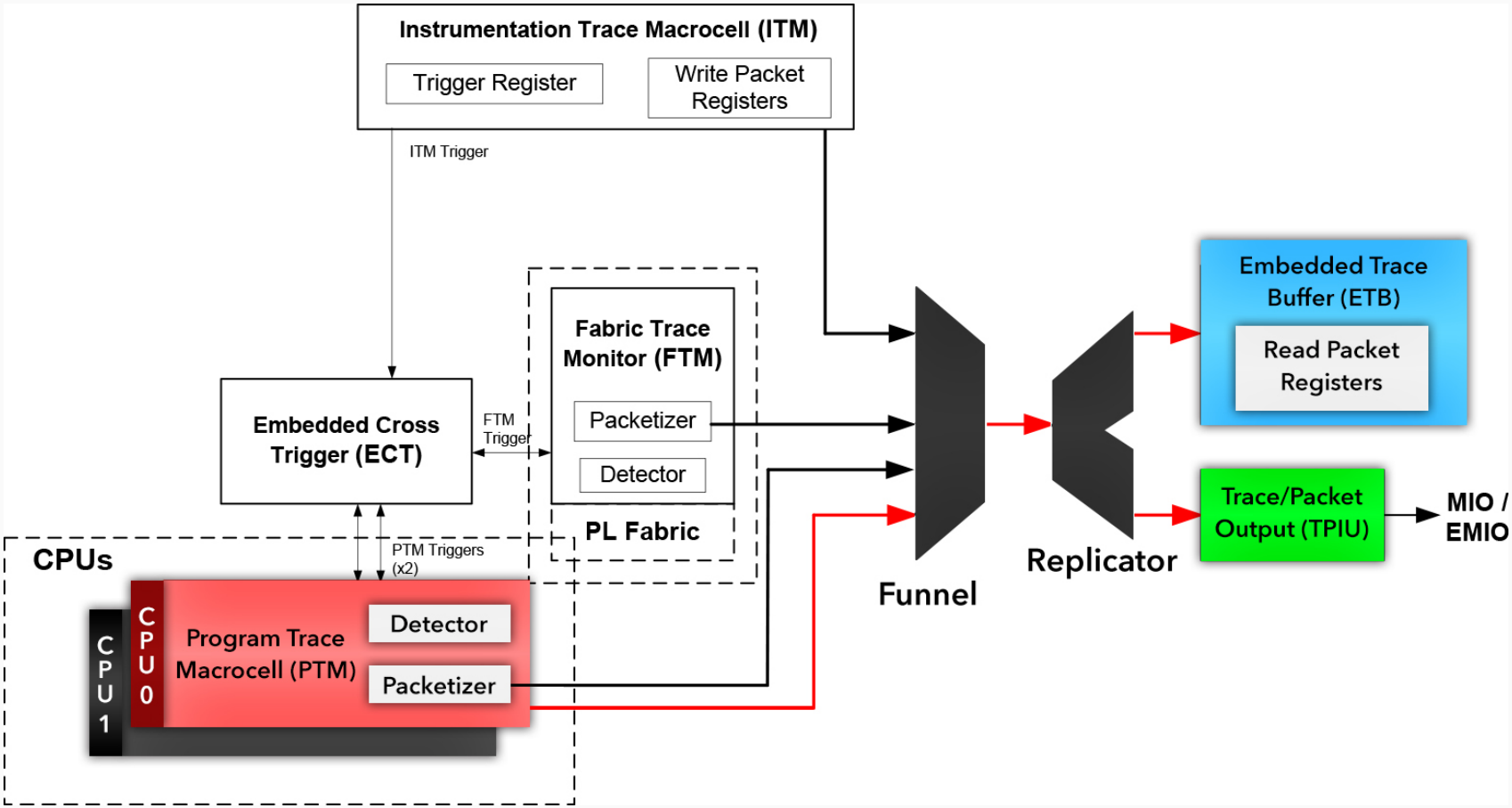
Constraints

- Zedboard: Zynq SoC (ARMv7 architecture, Z7020)
- Standard Linux
- Preliminary Threat Model: All PS and PL interfaces are considered secure

Coresight components



Coresight components



A set of IP blocks providing HW assisted system tracing[4]

[4] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/DDI0314H_coresight_components_trm.pdf

PFT trace Example

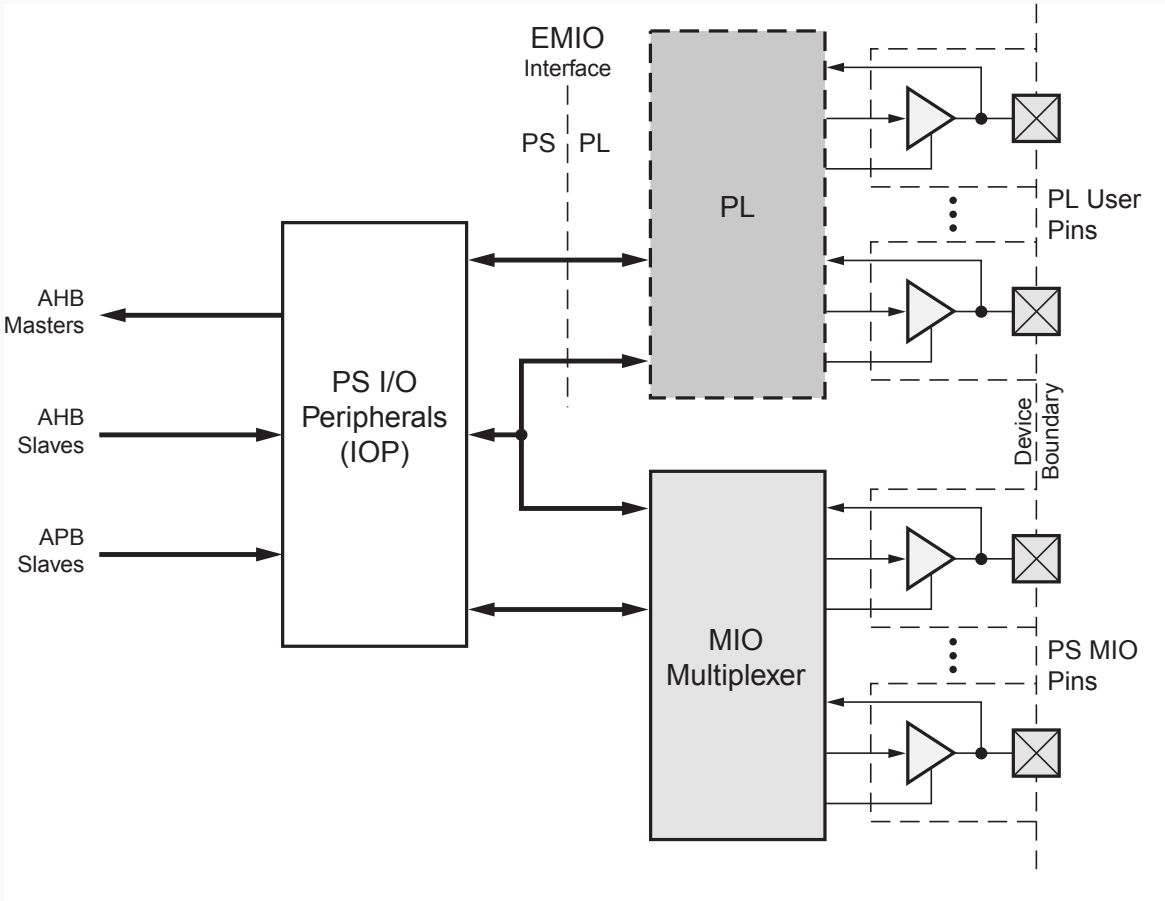
Address	Instruction	Trace, if any, with explanation
0x1000	MOV	-
0x1004	ADD	-
0x1008	B 0x1100	Direct branch taken. E atom generated.
0x1100	MOV	-
0x1104	LDR	-
0x1108	ADD	-
0x110C	CMP	-
0x1110	BNE 0x1104	Direct branch taken. E atom generated.
0x1104	LDR	-
0x1108	ADD	-
0x110C	CMP	-
0x1110	BNE 0x1104	Direct branch taken. E atom generated.

PFT trace Example

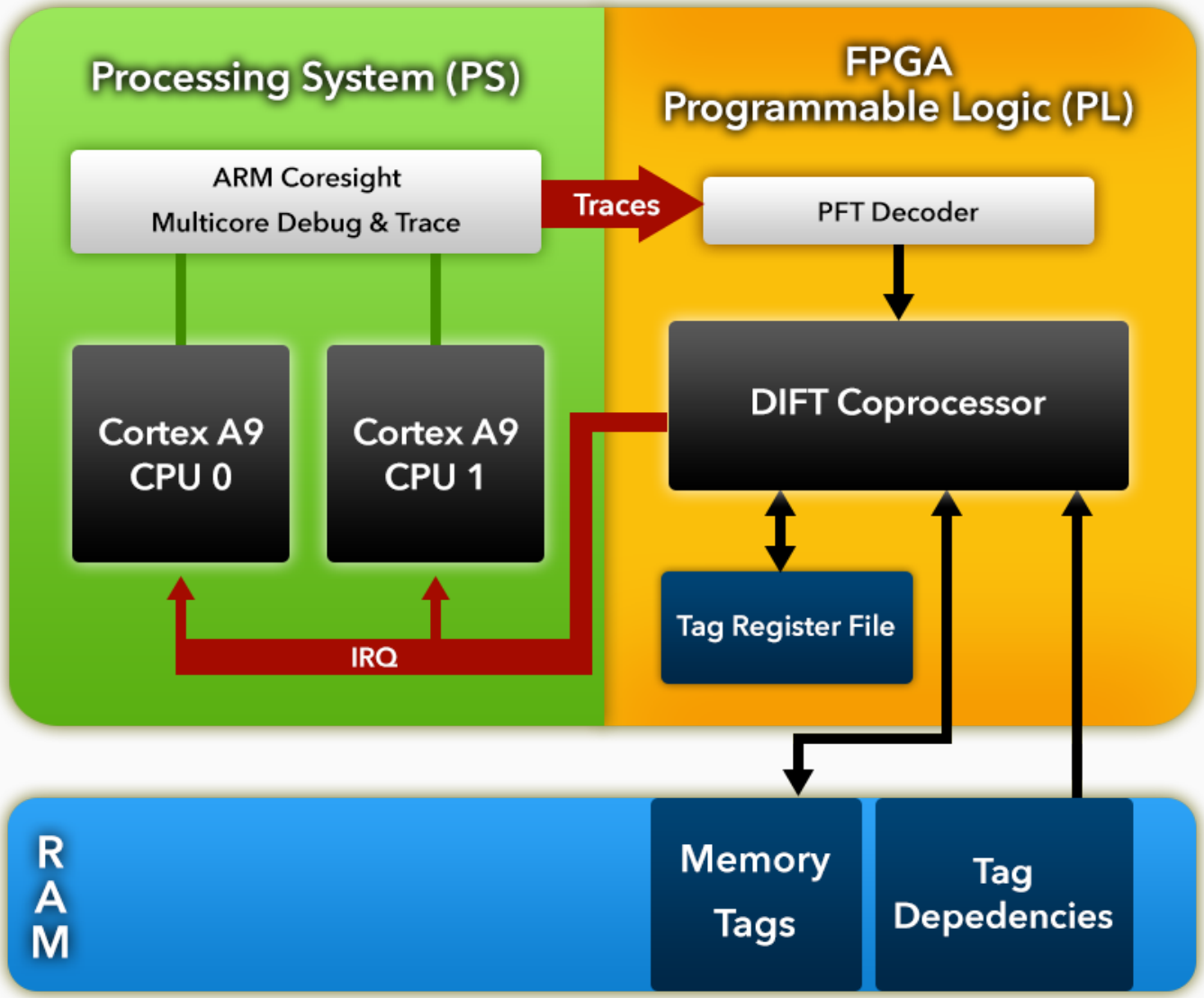
Address	Instruction	Trace, if any, with explanation
0x1104	LDR	-
0x1108	ADD	-
0x110C	CMP	-
0x1110	BNE 0x1104	Direct branch not taken. N atom generated
0x1114	LDR	-
0x1118	STR	-
		Indirect branch taken.
0x111C	MOV PC, Rn	Generate branch address packet indicating new address, 0x2000.
0x2000	MOV	-

Trace recovery on the PL

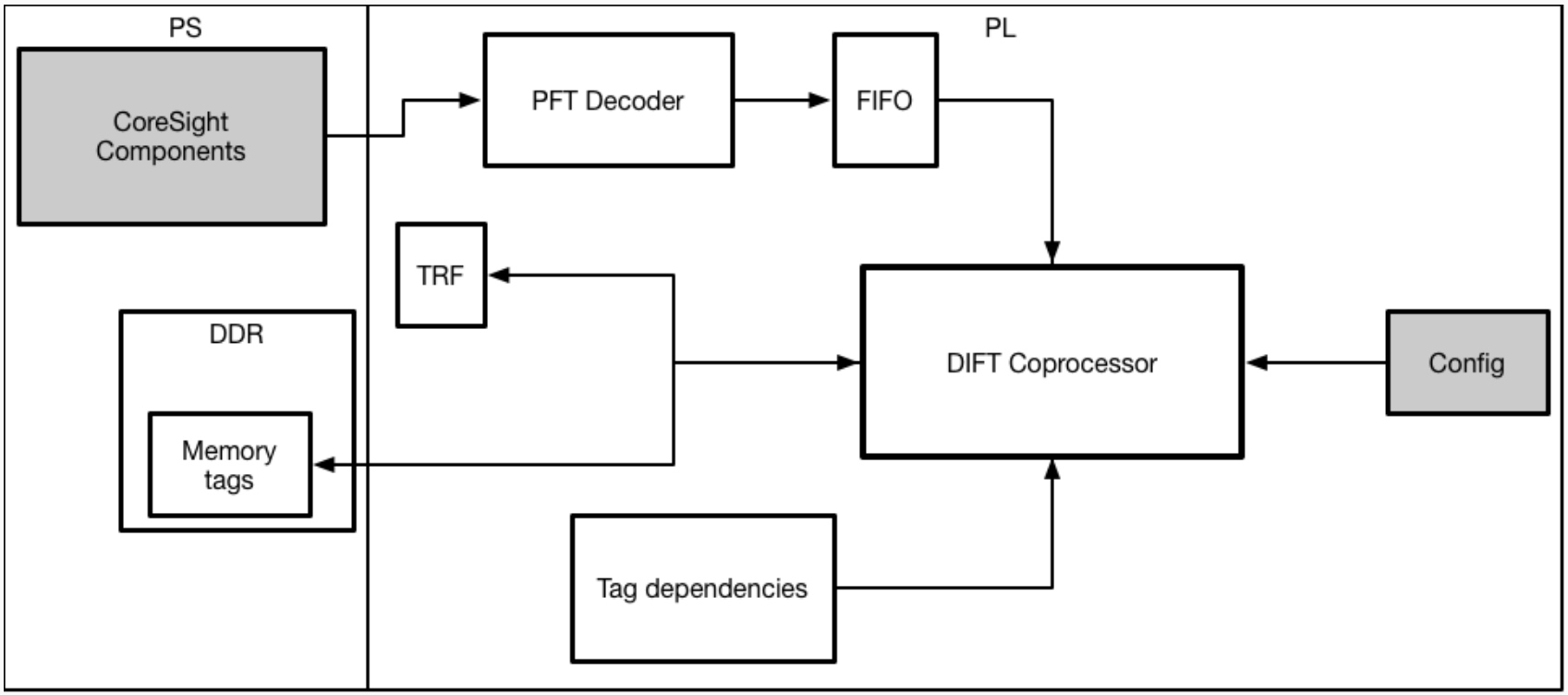
- EMIO: Peripheral port to Programmable Logic (PL)
- Facilitates connection to peripheral in programmable logic



Global Architecture

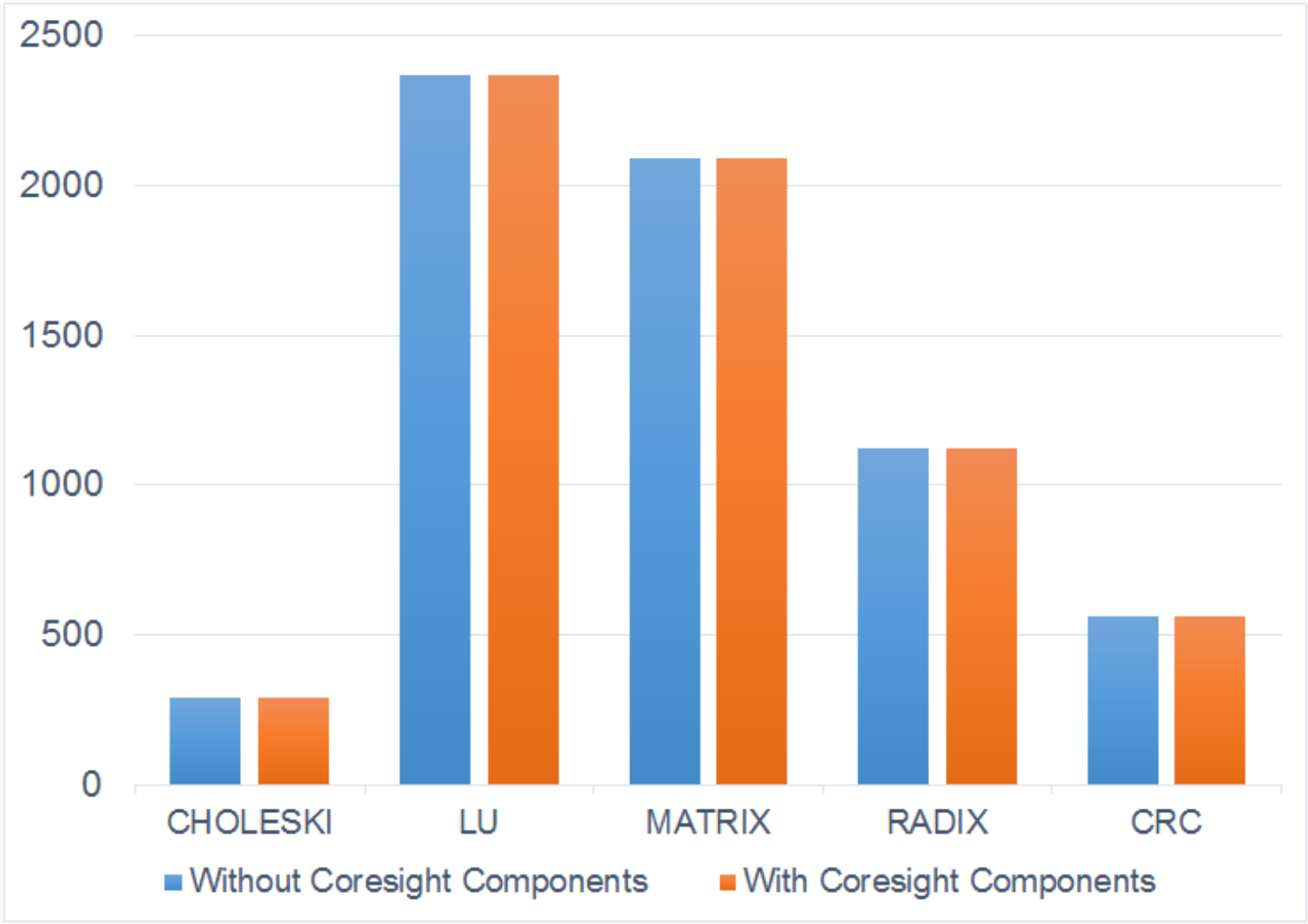


Global Architecture



Results - Coresight Components Overhead

Average execution time overhead 0.07%

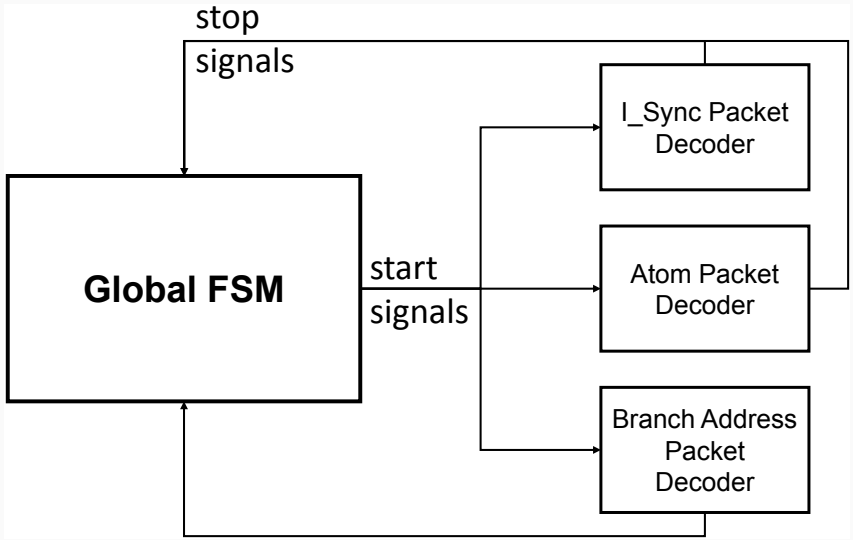


Average execution time (in μs) for MiBench programs

Results - PFT Decoder

Packet name	Header
A-SYNC	0x00
I-SYNC	0x08
Atom	0x80
Branch Address	0x01
Waypoint Update	0x72
Trigger	0x0c
Context ID	0x6e
TimeStamp	0x3c
Exception return	0x42
Ignore	0x66

PFT packets



PFT Decoder

Current status

- Linux Coresight driver
- Trace Decoder
- Tag Register File (TRF)
- Tag dependencies (Static Analysis)
- First prototype under development

IP Name	Slice LUTs	Slice Registers	Slice
Microblaze	824	530	300
PFT Decoder	308 (37%)	222 (42%)	110(37%)
TRF	49 (6%)	64 (12%)	13 (4%)

IP size for Zedboard (Zynq Z7020)

Conclusion

- CoreSight PTM allows to obtain runtime information
- Simple prototype under construction
- Approach allows to work with every hard core with debug components
  - Intel Processor Trace
  - STM (TI)
  - BHRB (PowerPC)
- Possible to extend DIFT on multicore SoC
- Few challenges ahead
  - Frequency issues
  - SW-HW synchronization
  - DIFT Coprocessor security

References I

- Dalton, Michael, Hari Kannan, and Christos Kozyrakis. “Raksha: A Flexible Information Flow Architecture for Software Security”. In: SIGARCH Comput. Archit. News. 35.2 (June 2007), pp. 482–493.
- Deng, Daniel Y and G Edward Suh. “High-performance parallel accelerator for flexible and efficient run-time monitoring”. In: Dependable Systems and Networks (DSN). IEEE. 2012, pp. 1–12.
- Deng, Daniel Y et al. “Flexible and efficient instruction-grained run-time monitoring using on-chip reconfigurable fabric”. In: IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society. 2010, pp. 137–148.
- Heo, Ingoo et al. “Implementing an Application-Specific Instruction-Set Processor for System-Level Dynamic Program Analysis Engines”. In: ACM TODAES. 20.4 (2015), p. 53.

References II


Kannan, Hari, Michael Dalton, and Christos Kozyrakis. “Decoupling dynamic information flow tracking with a dedicated coprocessor”. In: Dependable Systems & Networks, 2009. IEEE. 2009, pp. 105–114.


Nagarajan, Vijay et al. “Dynamic Information Tracking on Multicores”. In: INTERACT. 2008.


Suh, G Edward et al. “Secure program execution via dynamic information flow tracking”. In: Acm Sigplan Notices. Vol. 39. 11. ACM. 2004, pp. 85–96.


Venkataramani, Guru et al. “Flexitaint: A programmable accelerator for dynamic taint propagation”. In: HPCA 2008. IEEE. 2008, pp. 173–184.