Input to the Commission on Enhancing National Cybersecurity

Submission date: September 6, 2016

Joint submission made by:

Benjamin Gittins
Chief Technical Officer
b.gittins@synaptic-labs.com
+356 9944 9390

Ronald Kelson
Chief Executive Officer
r.kelson@synaptic-labs.com
+356 9944 9390

Synaptic Laboratories Ltd.
www.synaptic-labs.com
13 Nadur Heights,
Nadur NDR-1390,
MALTA, Europe

Designers of safe and secure computing and communication architectures. Developers of general-purpose soft IP for FPGA devices, to increase security and performance, and to reduce circuit area.

Topic of this submission:

Significant Progress In The Design Of Backwards Compatible, Cache-coherent, Trustworthy and Dependable computing: Synaptic Labs' Safe and Secure Real-Time platform with collaborators such as Intel PSG

RFI topic areas this submission relates to:

- Cybersecurity Research and Development
- Critical Infrastructure Cybersecurity
- Identity and Access Management
- Internet of Things
- International Markets

Submission contents:

(1) A 1 page executive summary for this comment, in the format requested by the RFI, which "identifies the topic addressed, the challenges, and the proposed solution, recommendation, and/or finding." We have inserted headings that match these points in the executive summary.

(2) Detailed technical information about the technologies being developed within the SSRT project are hosted on the European Union's Mixed Criticality Forum website (www.mixedcriticalityforum.org/projects/detail/?tx_dreams_fep[project]=12). Each year the Cluster holds a by invitation only one day workshop to showcase the latest advances in the field. In 2016 the SSRT project was selected for promotion at the Workshop held on 18 March at the 5 day European event for electronic system design and testing, the Design, Automation and Test in Europe (Date '16) Conference, Dresden, 14-18 March 2016. The Workshop slideshow of 44 slides / 22 pages in length (including the text spoken on the day) is attached to this submission.
https://contrex.offis.de/home/images/downloads/date2016/07_20160311-DATE-2016-SLL-Costs-HW-014-Website.pdf

(3) Brian Snow. We need assurance! In ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference, pages 3–10, Washington, DC, USA, Dec. 2005. IEEE Computer Society. Full text published online on the ACASC website. (https://www.acsac.org/2005/papers/Snow.pdf)

Significant Advances In The Design Of Backwards Compatible, Trustworthy and Dependable computing: Synaptic Labs' Safe and Secure Real-Time platform with collaborators such as Intel PSG

1 Page Executive Summary

RFI Topics: Cybersecurity Research and Development, Critical Infrastructure Cybersecurity, Identity and Access Management, Internet of Things, International Markets

Problem: Brian Snow (Formerly U.S. National Security Agency for 30+ years, designing secure products and systems, including 12 years as Technical Director) states: "For a one-word synopsis of computer design philosophy, it was and is: SHARING. In the security realm, the one word synopsis is **SEPARATION**. .... So today, making a computer secure requires imposing a "separation paradigm" on top of an architecture built to share. That is tough! Even when partially successful, the residual problem is going to be covert channels." Real-time experts state: "In safety critical and mission critical systems ... it is important to assign applications with different requirements to different partitions with different criticality levels ... Partitions should be isolated functionally, **temporally** and **securely** ... Unfortunately, modern COTS architectures are **not** built to provide strong isolation guarantees." From a safety perspective, in 2012 Airbus' Benoît Triquet stated multicore processors represent "a major challenge how to adequately deploy them for safety applications they were typically not designed specifically for. ... Temporal behaviour has been much less addressed .. **Airbus ... have found very few multicore chips that can ever hope to be useable for avionics.**"

Progress being made: The Safe and Secure Real-time Project is the answer to published top priority Government and industry needs for high performance, area efficient, real-time capable, multi-core (and many-core) computers, on which both general purpose and real-time software can run **concurrently at the same instant in time with higher performance**. The single-core performance of software running on soft cores accessing FLASH and SDRAM in FPGA will be more competitive than current single-core systems. On the SSRT architecture with many cores and many bus-master peripherals, real-time software will be as easy to write, verify, certify and maintain as it is today on single-bus master computer architectures.

To achieve this, several years of foundational cross-domain research and industrial needs analysis has led to universal computer architecture designs for multiple industries that are commercially viability in the soft logic of COTS FPGA chips (and that can then be developed as hard-macro's in FPGA or as ASIC micro controller chips). Our designs systematically eliminate or control all timing and performance problems at the source, in the hardware, with no changes needed to application software and no changes or minimal porting for O/S, while ensuring that all shared memory and message passing paradigms and all real-time operating system types (ARINC-653, AUTOSAR 2.0, ...) can be supported. The SSRT project is processor agnostic.

The SSRT architecture is designed to be extended to create Synaptic Labs', independently reviewed, Trustworthy resilient universal Secure Infrastructure Platform (TruSIP). TruSIP is designed to provide high-assurance security controls that prevent the public cloud provider and their hardware and software suppliers from maliciously or unintentionally learning or exposing the value of the cloud client's data, even though the data is being processed in the cloud. This includes protection against malware hidden in the hardware or software employed in the cloud infrastructure used to provide services to customers.

The recommendation: We respectfully propose that the Commission's detailed recommendations to strengthen cybersecurity should include the following points:

1. Perform a high-level survey to identify, catalogue and evaluate the viability of all candidate next-generation **cache-coherent** mixed criticality real-time **multi-core** computer architectures that provide backwards compatibility of general purpose and real-time software running on COTS real-time operating systems and that can scale performance near-linearly for between 2 to 4 cores. (Based on our survey of all published real-time computing architectures as of 2014, there are very few cache-coherent designs.)
2. Perform a high-level security aware Failure Mode and Effects Analysis of today's COTS real-time capable computing architectures that considers the impact of identified safety and security flaws wrt. the stakeholders in critical infrastructure and cyber-physical applications (automotive, industrial control, avionics, aerospace). Quantify the costs to the global community of those security flaws. Quantify the returns of developing a "fit for purpose" high-assurance real-time computing platform. Fund the top 5 candidate next-generation computing solutions that are credibly trustworthy and dependable, ensuring sufficient diversity between the research agendas / techniques. Ensure equal access and adequate support for (and team building around) innovative small-to-medium sized enterprises.

For more information, visit the project summary hosted on the website of the EU Mixed-Criticality Cluster (www.mixedcriticalityforum.org/projects/detail/?tx_dreams_fep[project]=12) See also the invited presentation at the EU funded Mixed-Criticality Cluster's workshop at the 5 day Design, Automation and Test in Europe (Date '16) Conference, Dresden, 14-18 March 2016 which is attached.

Sincerely, Benjamin Gittins and Ronald Kelson.

Very brief introduction to only a few aspects of S/Labs' Safe and Secure Real-Time (SSRT) cache-coherent shared memory computing architecture

Already under development - No major barriers remain

- The most complex component of SSRT at Tech. Readiness Level 5 (TRL-5)
- Some discrete technologies are also advancing to market now in products

High performance

- With clock cycle deterministic time partitioning between ALL cores and ALL bus-master peripherals
- Satisfies both general purpose AND time-analysability, safety and security requirements

Industry agnostic, cross domain

- IoT, auto, avionics, industrial control, ...

Benjamin GITTINS cto@pqs.io
Ronald KELSON ceo@pqs.io

Overcomes all major barriers with:

- **Faster per-core *and* across cores performance for best effort *and* real time workloads**
  - Near linear scalable software performance over 1 to 28 cores
  - Guaranteed wire-speed shared memory bandwidth for peripherals
- **Faster total system performance:**
  - With any combination of memory intensive mixed-criticality tasks distributed across cores
- **Support for:**
  - All popular *(and* emerging) timing analysis techniques; and
  - Asymmetric and symmetric multi-processing for all RTOS types
- **Easy implementation** in most FPGA's (and ASIC) - vendor neutral

Our strategy and roadmap for developing SSRT

1. **Create a vendor neutral shared memory architecture (done)**
   a. Rework any components found in general-purpose computer architectures that reduce:
      a. time analyzability or
      b. software performance

   b. Our interconnect is explicitly designed to support *unmodified* COTS cores and peripherals to support rapid market up-take:

      i. Time-analysable soft-cores within the FPGA

      ii. Coupled with dual ARM cores within the hard processing subsystem of SoC FPGA

      iii. Coupled with quad-core Intel Xeon cores external to the FPGA

SYNAPTIC LABORATORIES LTD.
Concepts for Composable Dependable Architectures &
Costs of Hardware-Support for Dependability (DATE 2016)

Our strategy and roadmap for developing SSRT

1. **Create a vendor neutral shared memory architecture (done)**
2. **Already advancing a 4 phase commercialisation roadmap**

   a. Develop parts of the architecture in stand-alone products for FPGA (started)

   b. Complete implementations of SSRT architectures for FPGA coupled to ASIC

      i. Employ an enhanced time-analysable shared memory cache

      ii. Employ a tiny time-analysable MMU that introduces zero timing jitter

   c. Upgrade the Nios II soft-core to employ our Cache and MMU directly

   d. Full ASIC implementation/s of SSRT

Introducing SSRT over 9 Parts

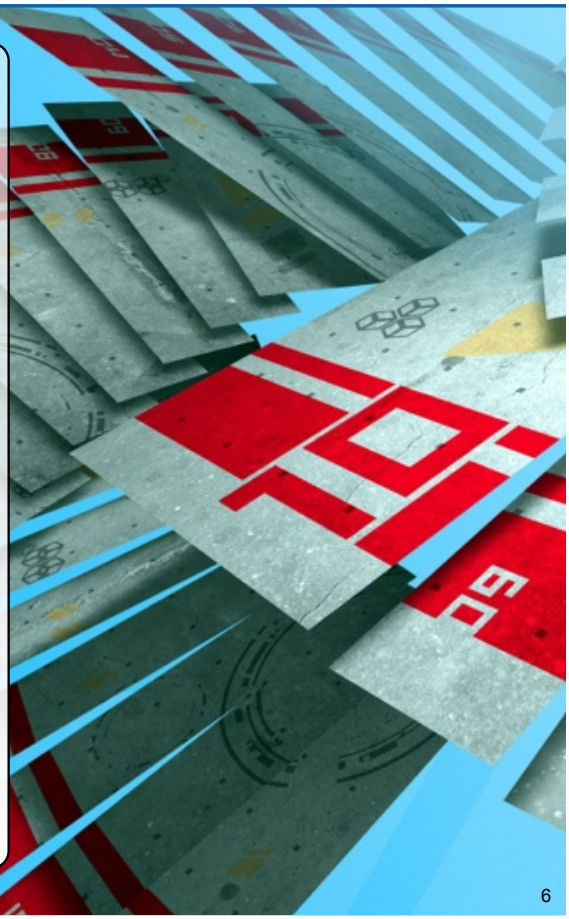
Slide number(s)		
6 - 12	Part 1:	**1-core** SSRT configuration
13 - 20	Part 2:	**2-core** SSRT configuration
21 - 22	Part 3:	**3-core** SSRT configuration (ARM and Nios II cores)
23 - 27	Part 4:	**8-core** SSRT configuration
28 - 30	Part 5:	**14-core** SSRT configuration (Xeon and Nios II cores)
31 - 32	Part 6:	**28-core** SSRT configuration
33 - 34	Part 7:	SSRT's support for multiple timing analysis techniques
35 - 37	Part 8:	Broad across-industry expectations wrt. realtime architectures and How SSRT satisfies those industrial requirements
38 - 42	Part 9:	S/Labs' 4 phase commercialisation roadmap for SSRT
43 - 44		Closing statement

Part 1:

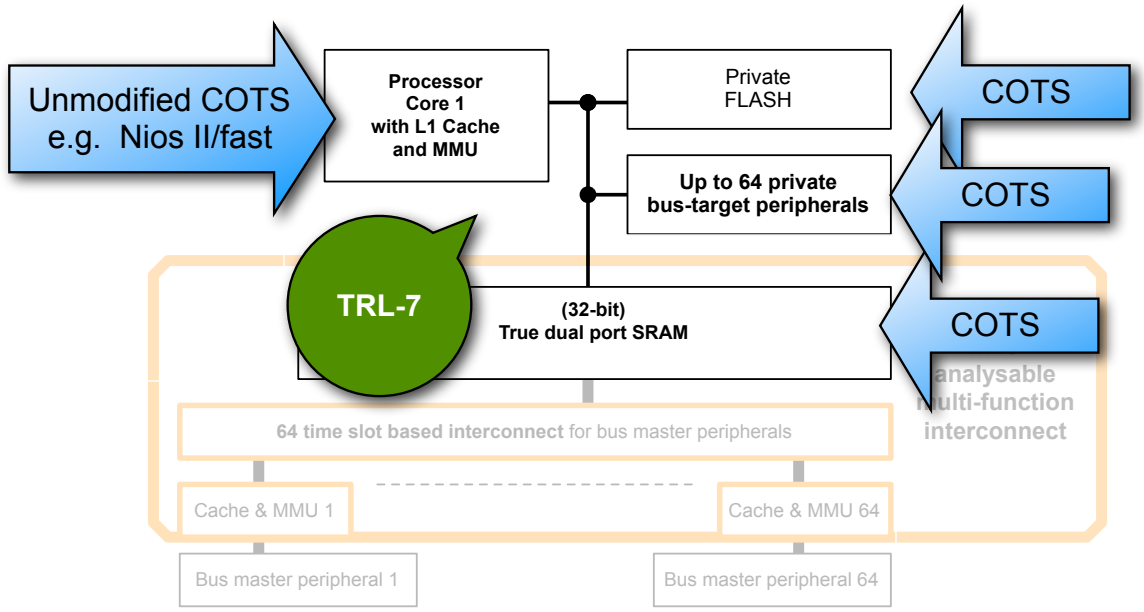
A fully time-analysable unmodified COTS single-core, multi bus-master SSRT configuration

that has faster best-effort & real-time performance than today's single-core architectures

(using comparable components)

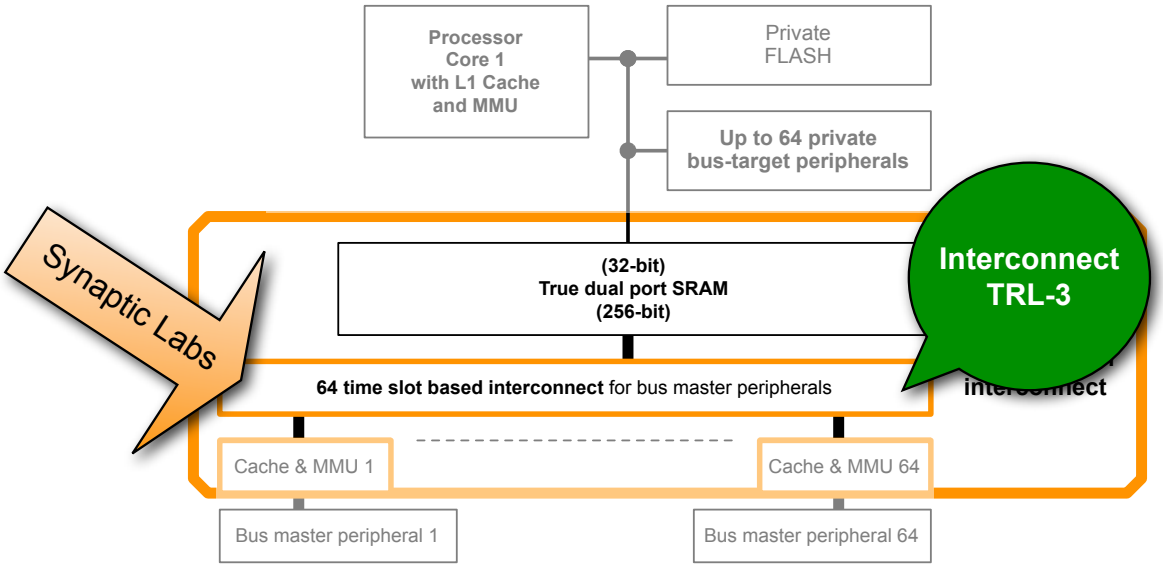


Single-core design with 64 bus master peripherals



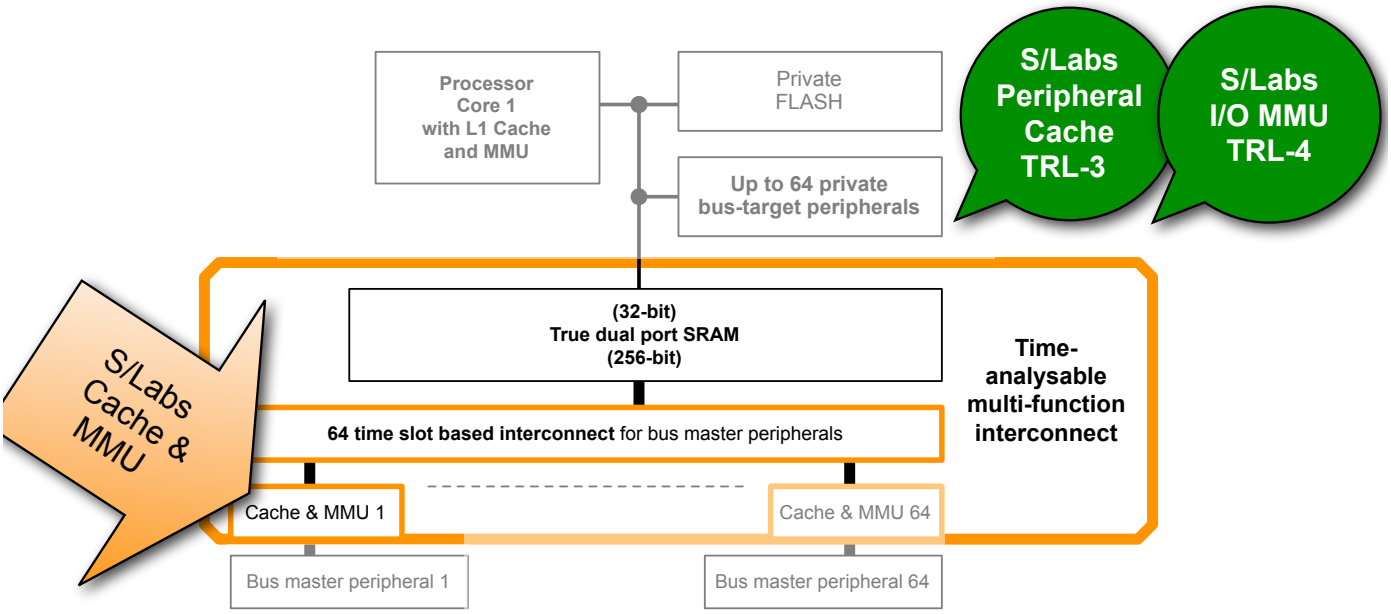
- The single unmodified COTS core has 100% time deterministic access to:
  - COTS Private Flash
  - Up to 64 private COTS bus-target peripherals
  - One port of the true dual-port SRAM which stores shared memory

Single-core design with 64 bus master peripherals



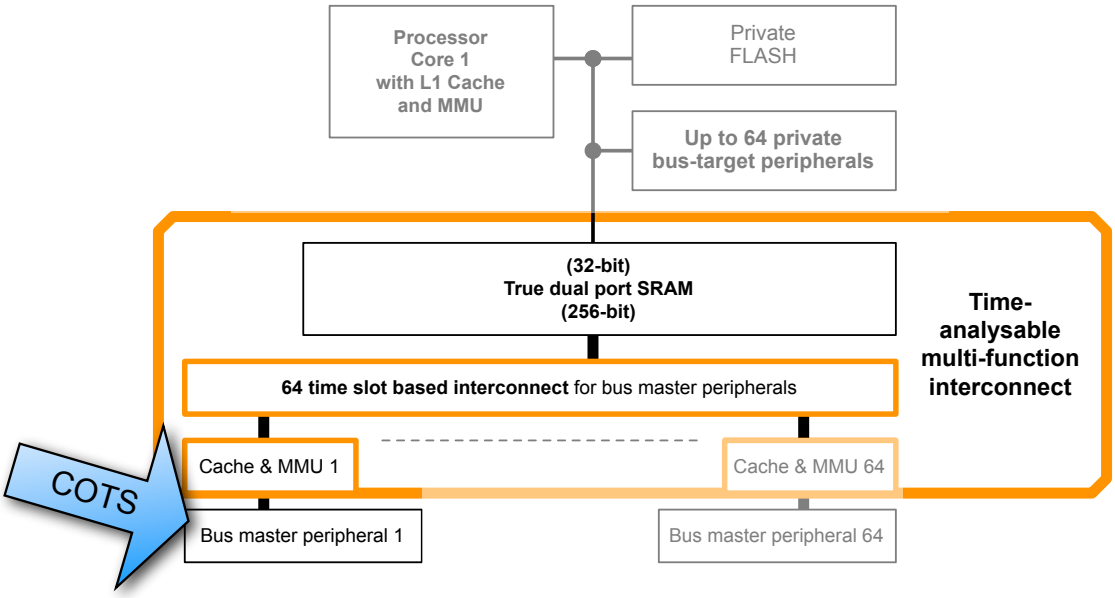
- Employs S/Labs' up to 64 time slot based interconnect for bus master peripherals
  - 256-bit wide data path to the true dual-port SRAM
  - Each time slot is exactly 1 clock-cycle in duration
  - The worst case access latency for 1 time slot is only 63 clock-cycles
  - Permits allocation of multiple time slots to any very high-bandwidth peripherals

Single-core design with 64 bus master peripherals



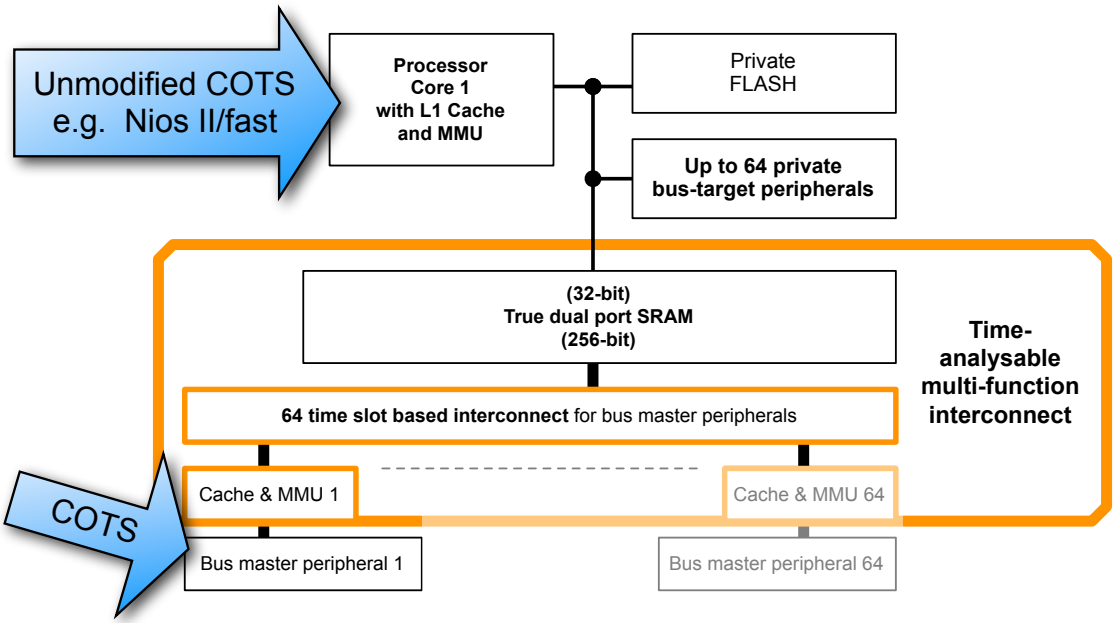
- Each bus-master peripheral is paired with a tiny fully-associative cache & I/O MMU
  - Accelerates contiguous rd/wr operations by that COTS bus-master peripheral
    - Combines 32-bit wide writes into a 256-bit contiguous write operation to SRAM
    - Prefetches 256-bits of data from SRAM on a 32-bit read-miss by the peripheral

Single-core design with 64 bus master peripherals



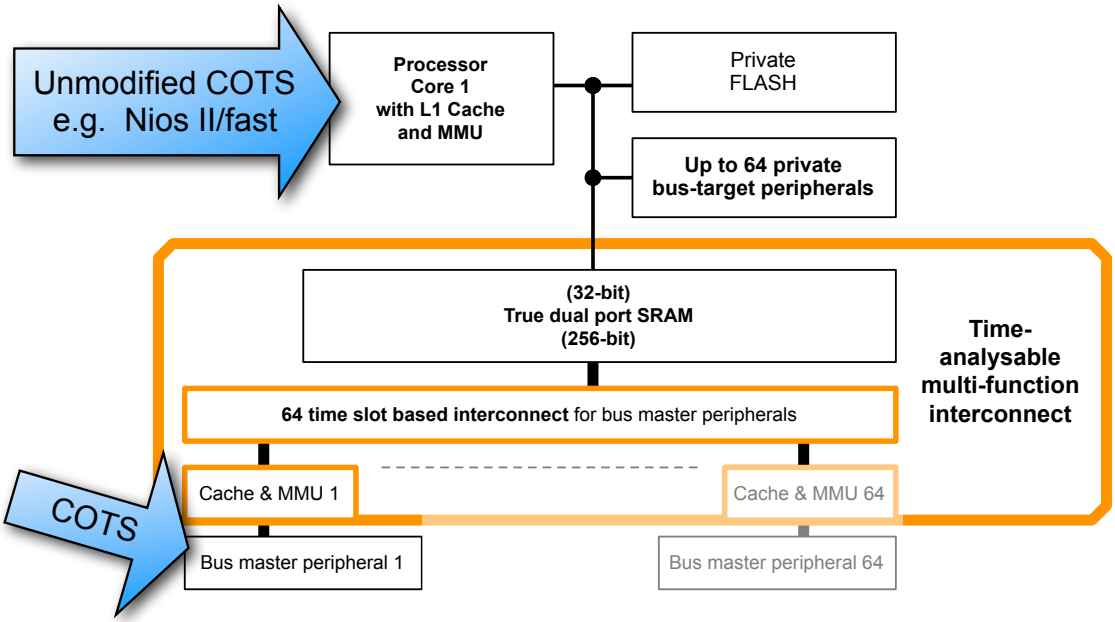
- It is trivial to configure SSRT at design-time to ensure that each COTS bus-master peripheral has **guaranteed wire-speed bandwidth with low latency**

Architecture enables superior ACET performance



- **SSRT is the *ideal* high-performance single-core ACET architecture**
  - **Compared to today's COTS single-core microcontroller architectures,** the average case execution time of software running on the core is FASTER because:
    - there is no timing interference from bus-master peripheral activity

Architecture enables superior WCET performance



- **SSRT is the *ideal* high-performance single-core WCET architecture:**
  - The upper-bound WCET analysis of software running on that core is as tight as a single-core system with no bus-master peripherals
  - All bus-master peripherals are intrinsically guaranteed wire-speed bandwidth with guaranteed tight upper-bound access latencies at all times

Part 2:

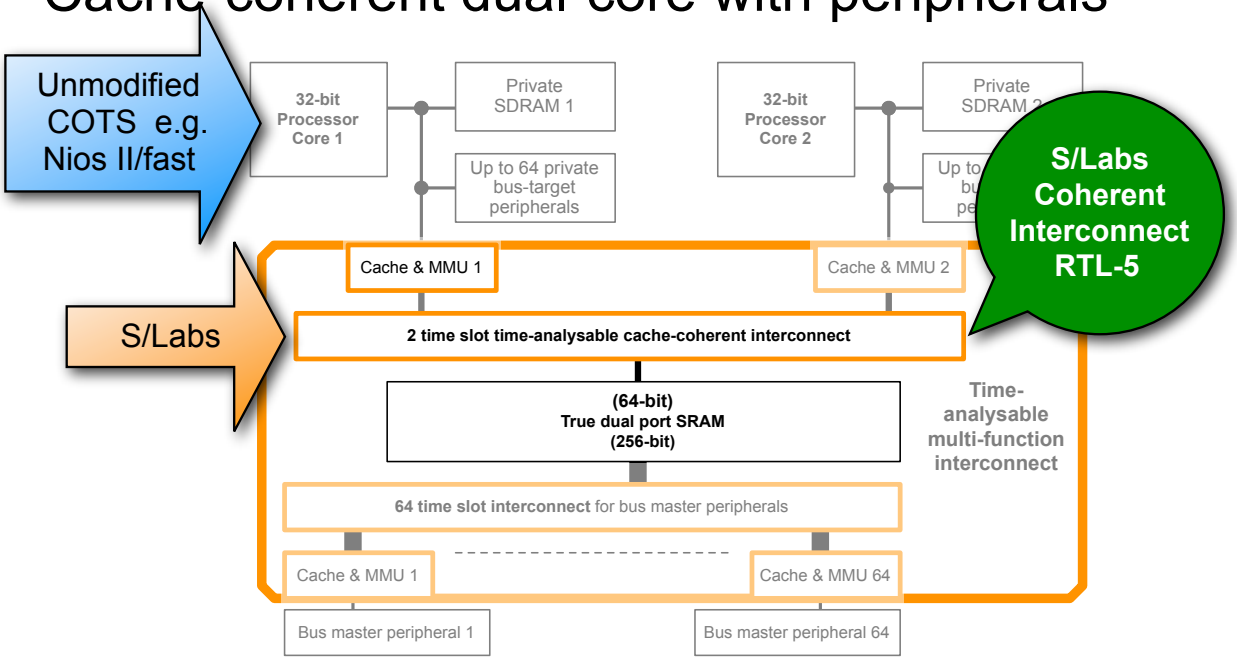
An unmodified COTS dual-core, cache coherent, SSRT configuration

Achieves higher bandwidth access to shared memory than today's multi-core architectures

(using comparable components)

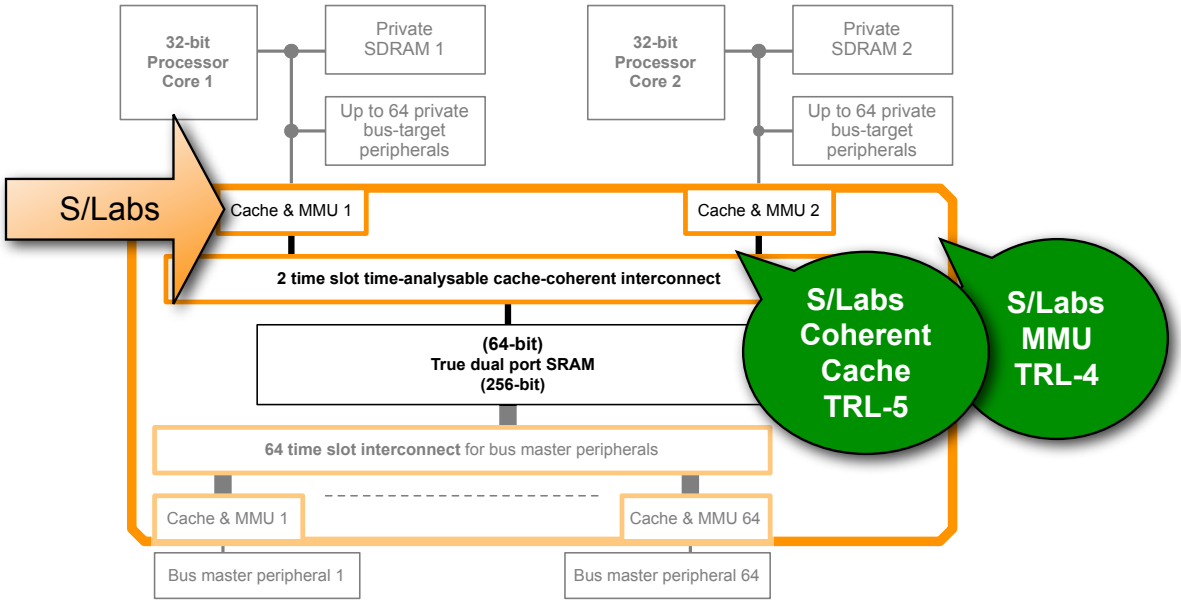
Employs the technologies described in the single-core SSRT configuration

Cache-coherent dual-core with peripherals



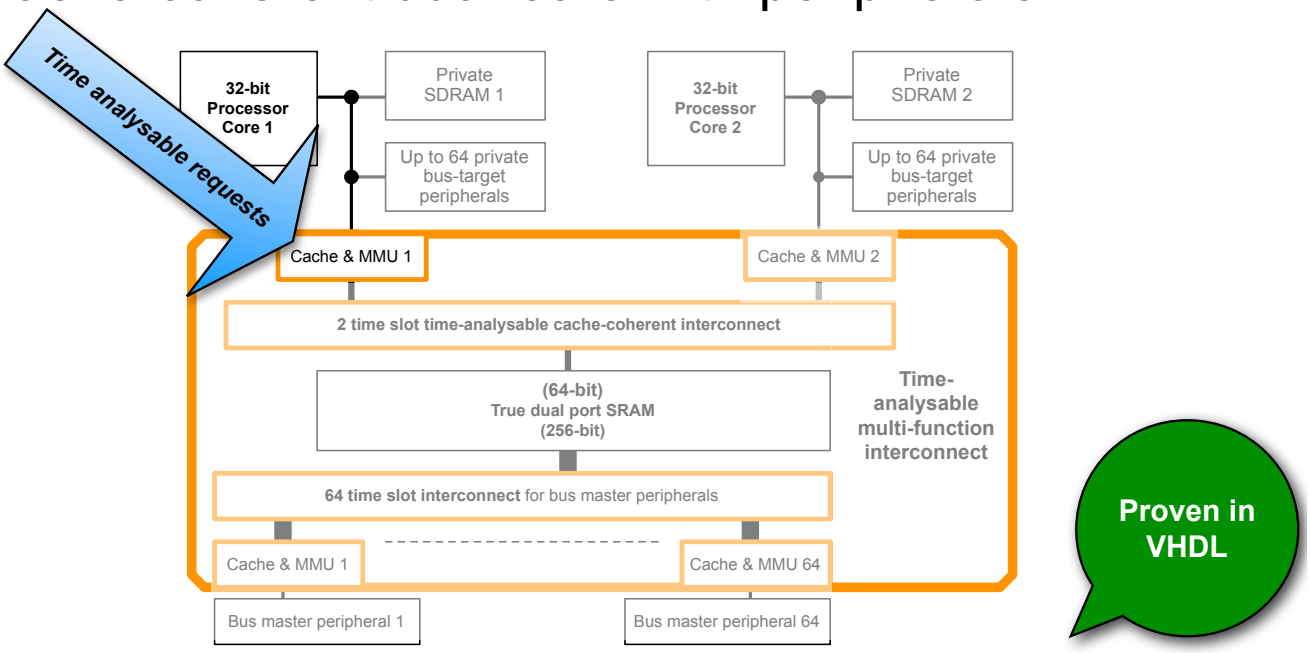
- The dual-core configuration adds S/Labs' time-analysable **cache-coherent** time slot based interconnect which has:
  - 2 time slots for 2 cores
  - 64-bit wide data path (2x the width of the 32-bit processor word length)

Cache-coherent dual-core with peripherals



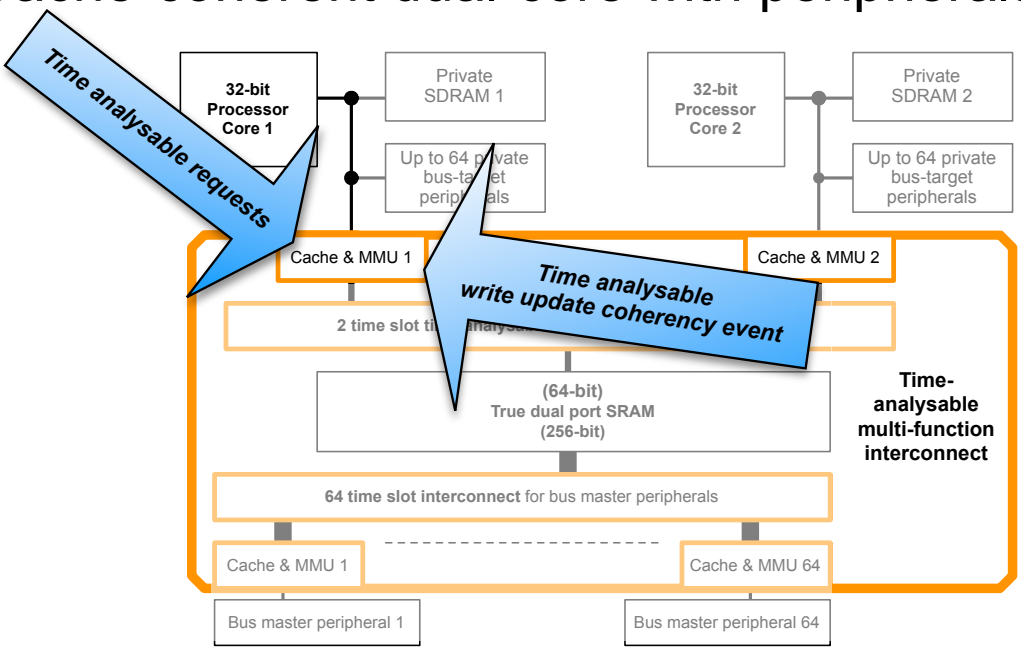
- There are 2 caches *and* 2 constant time MMU connected to that interconnect
  - Each of S/Labs' coherent caches employ:
    - a fully-associative, true LRU (or random) cache-line eviction scheme; and
    - a time-analysable "write-update" (snarfing) cache coherency scheme

Cache-coherent dual-core with peripherals



- The time analysable memory transfer requests issued by the processor core to the cache of the interconnect ...

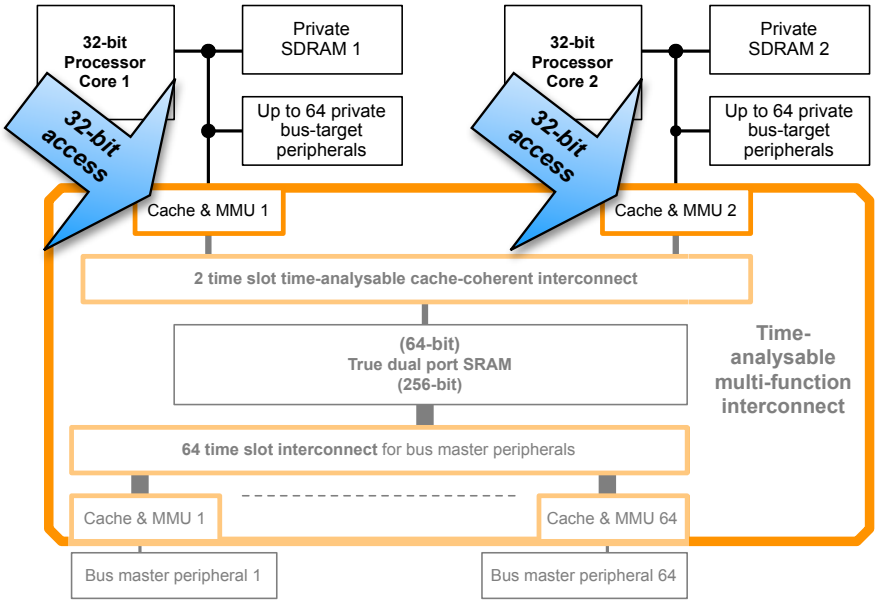
Cache-coherent dual-core with peripherals



Proven in VHDL

- The time analysable memory transfer requests issued by the processor core to the cache of the interconnect experience **zero timing interference** wrt. all write coherency events issued over that interconnect

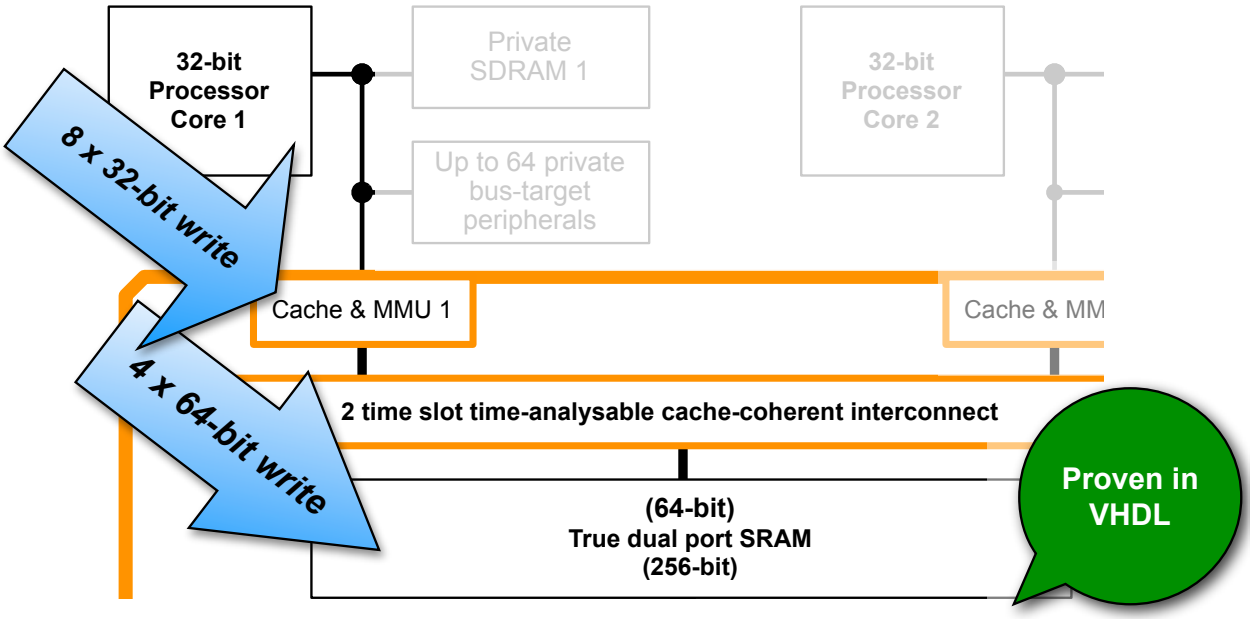
Cache-coherent dual-core with peripherals



Proven in VHDL

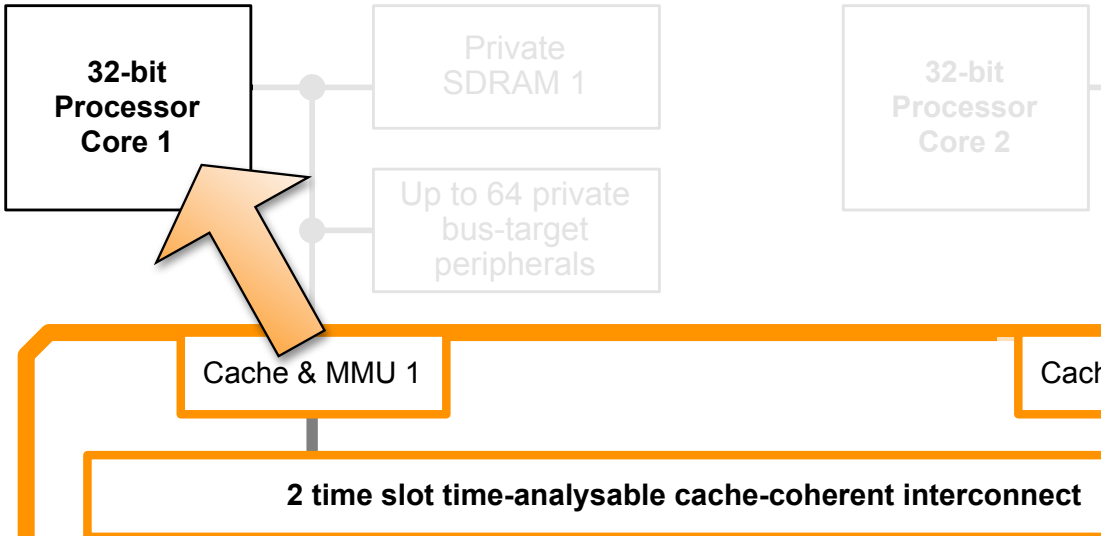
- In a *N* core configuration of the SSRT architecture:
  - Every core can issue
    N 32-bit wide memory requests
    every
    N clock cycles to its cache

Cache-coherent dual-core with peripherals



- In a *N* core configuration of the SSRT architecture:
  - We linearly scale the cache-line and bus widths with the number of cores to **ensure that the peak bandwidth between cores SCALES LINEARLY**
  - For example eight 32-bit write operations issued over 8 clock cycles can be combined into four 64-bit write requests issued issued over 8 clock cycles with zero timing jitter introduced from unrelated memory transfer requests

In phase 3: Slightly modifying COTS cores



- To gain additional per-core performance in phase 3 of our commercialisation roadmap, we plan to implement our
  - **coherent cache; and**
  - **smaller and faster MMU**

  directly into the Nios II/fast soft-core processor pipeline

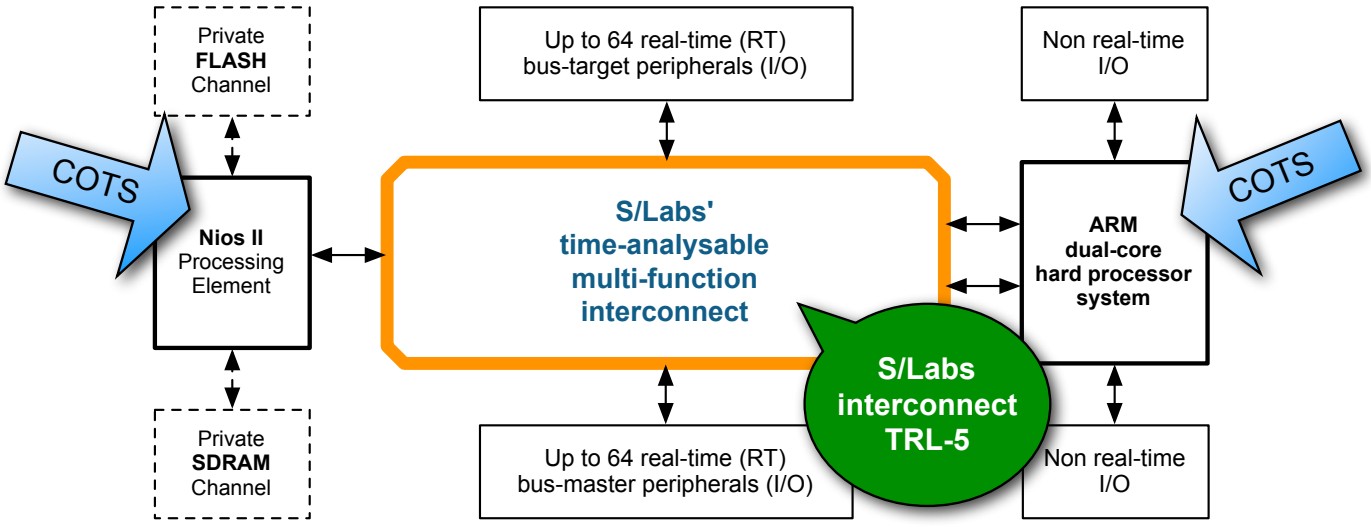
Part 3:

An unmodified COTS tri-core, cache coherent, SSRT configuration

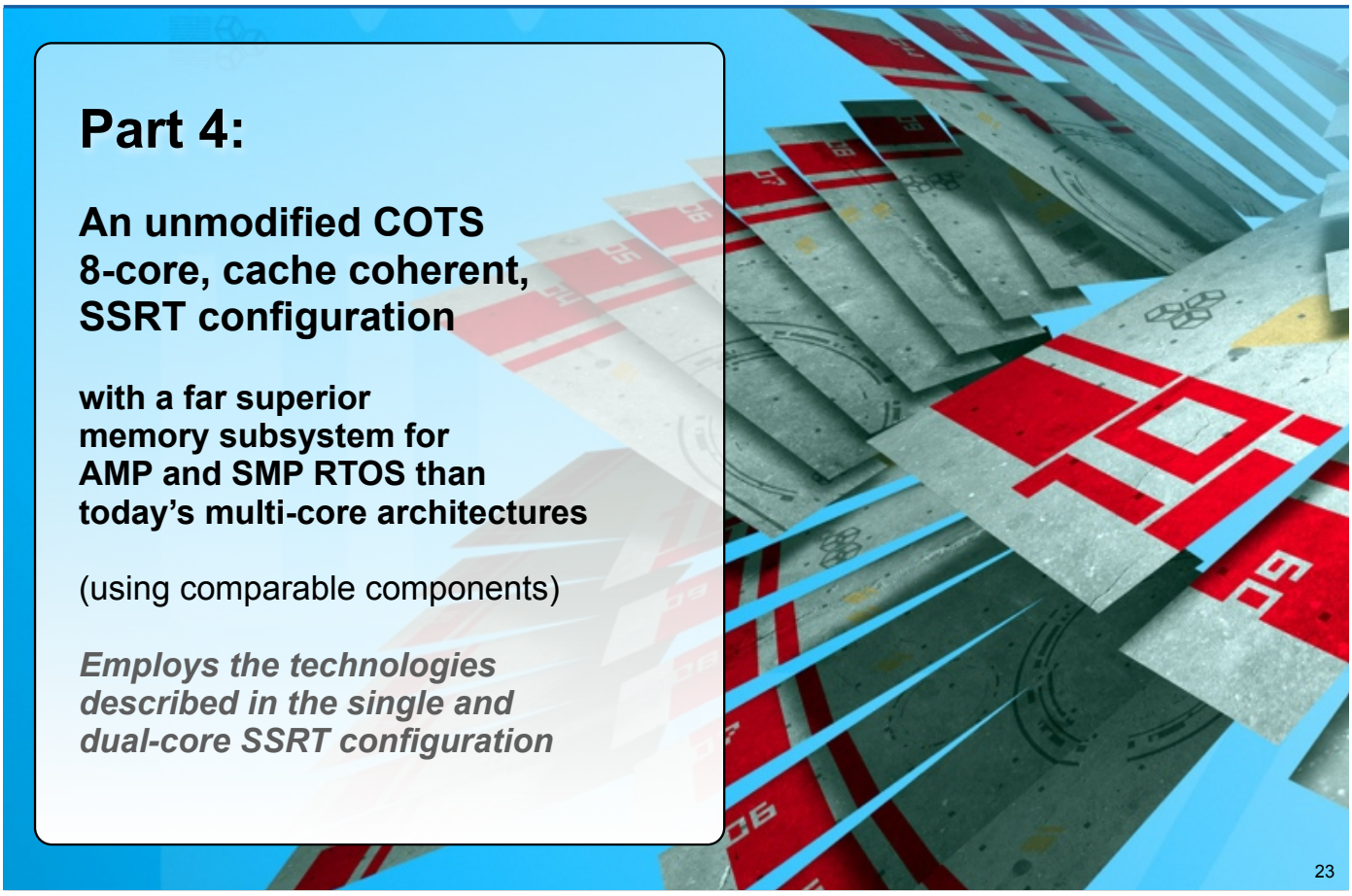
This example employs dual hard macro ARM cores and a time-analysable soft Nios II core in one SoC FPGA

Employs the technologies described in the single and dual-core SSRT configurations

Use the fast ARM cores of Altera SoC FPGA



- The unmodified Nios II core and COTS peripherals located in the programmable logic of the FPGA experience zero timing interference from the fast unmodified ARM cores and COTS peripherals located in the hard processing subsystem of the FPGA



SYNAPTIC LABORATORIES LTD.
Concepts for Composable Dependable Architectures &
Costs of Hardware-Support for Dependability (DATE 2016)

SSRT is carefully designed for use with SDRAM

- Wrt. 32-byte burst access to a DDR SDRAM running @ 800 MHz per data pin:

  Peak performance of an **8-bit** wide SDRAM in **OPEN**-page mode (6.4 Gb/s)

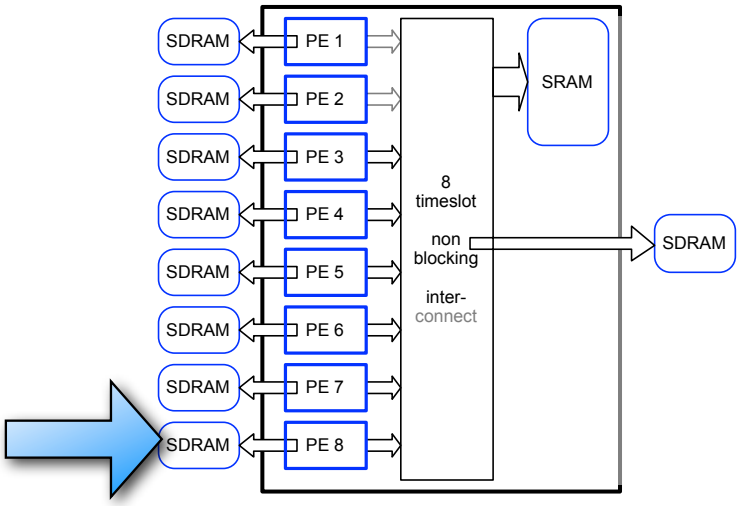
  is 1.56x faster than the

  peak performance of a **64-bit** wide SDRAM in **CLOSED**-page mode (4.1 Gb/s)

SSRT is carefully designed for use with SDRAM

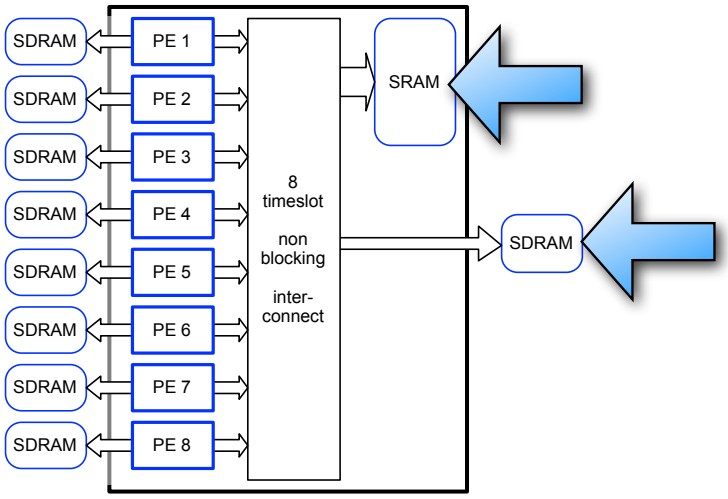
- To take advantage of this fact:
  - SSRT's memory subsystem is carefully designed to maximise SDRAM row-hits on narrow, open-page mode, private SDRAM channels
  - This increases the effective memory bandwidth per data pin of SDRAM while controlling hardware costs in a multi-core context

Simplified view of the 8-core memory subsystem



- Private narrow SDRAM channels are used to store all executable code and private data that is accessed by its core
- This self-evidently results in LINEAR scalability of private code + private data access in AMP and SMP tasks wrt. the number of cores

Simplified view of the 8-core memory subsystem



- In SMP contexts:
  - All data shared between the threads of an application running on 2 or more cores can be mapped to either shared SDRAM or shared SRAM

Part 5:

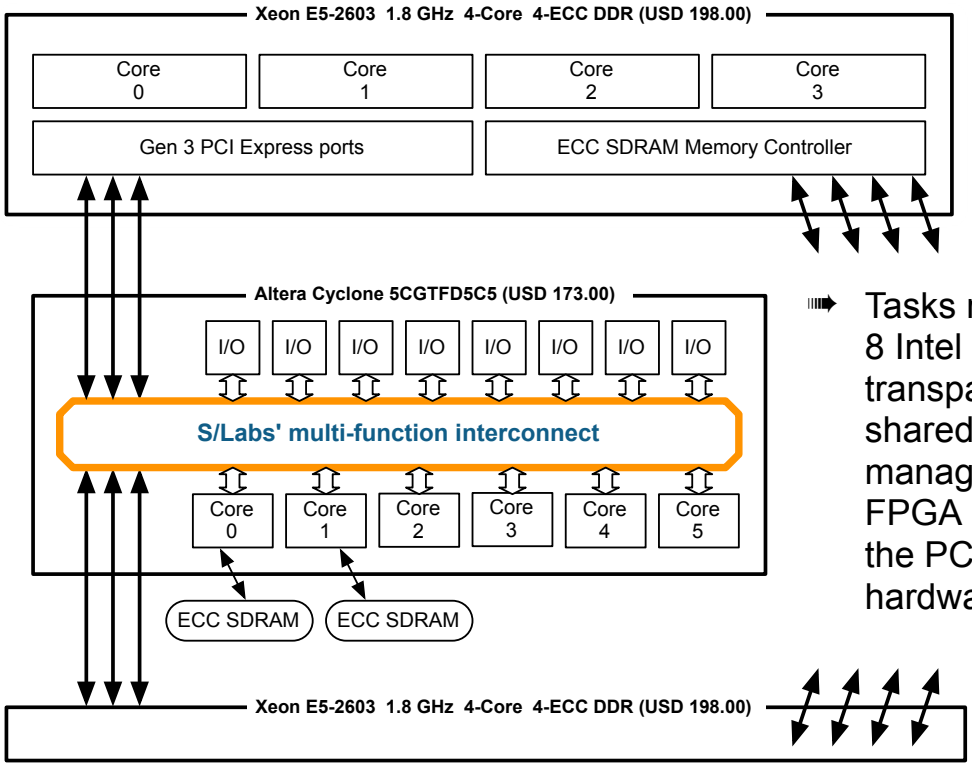
An unmodified COTS 14 core, shared-memory configuration of SSRT that employs:

8x Intel Xeon processor cores

6x Intel PSG Nios II/fast cores

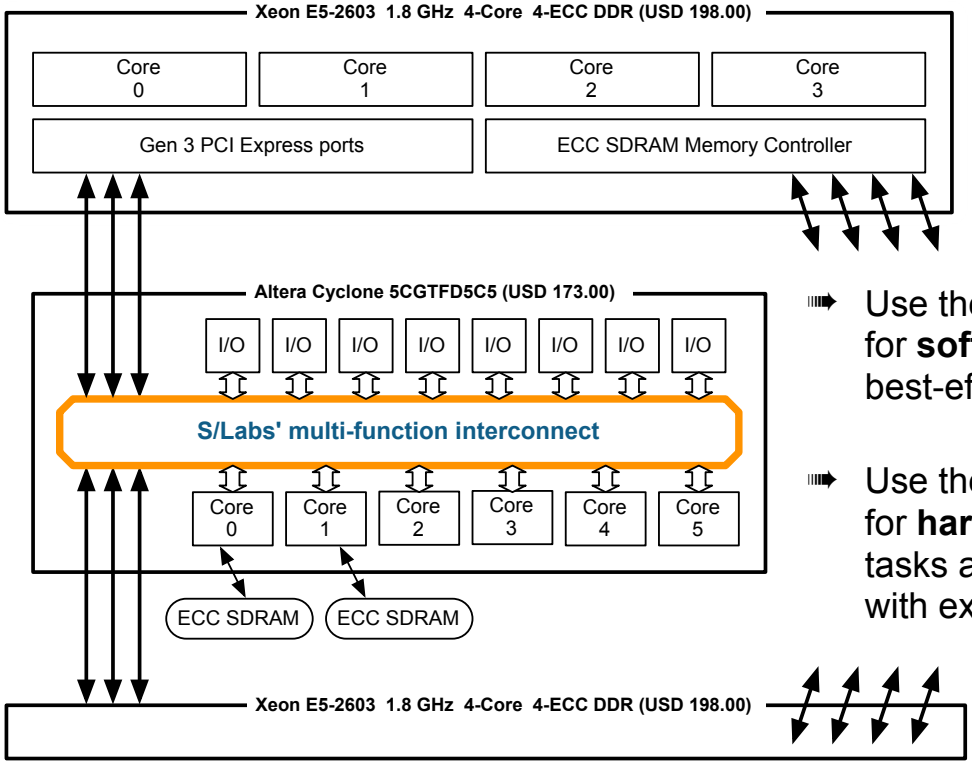
Employs the technologies described in the single and dual-core SSRT configuration

Coupling 2 Xeon E5-2603 chips to our interconnect



- Tasks running on the 8 Intel Xeon cores can transparently access the shared memory managed on the FPGA by using the PCI express hardware

Coupling 2 Xeon E5-2603 chips to our interconnect



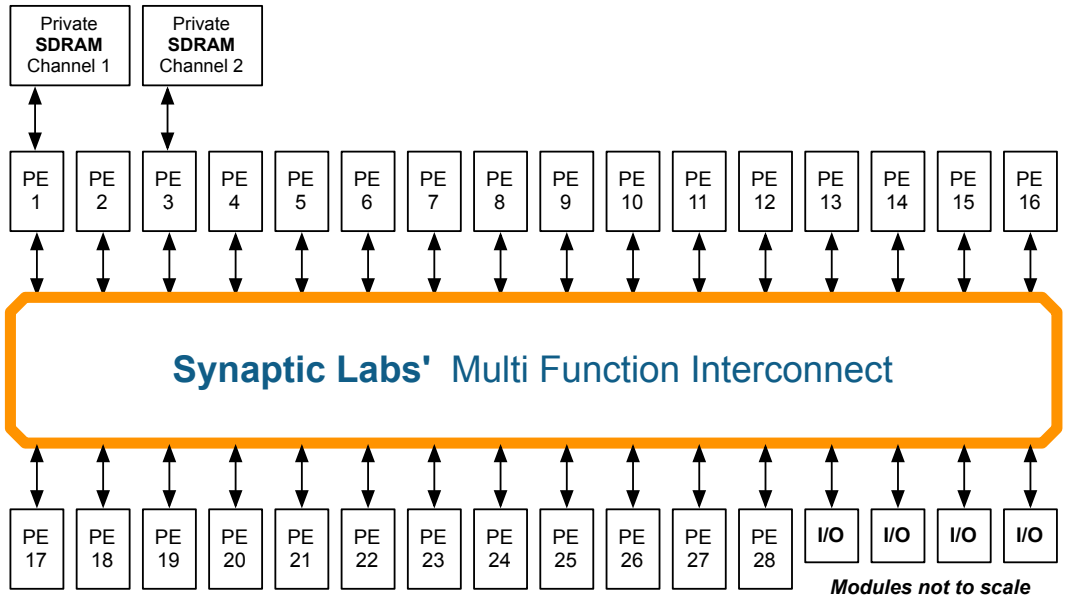
- Use the 2 Intel Xeon chips for **soft** real-time and best-effort tasks
- Use the Altera FPGA for **hard** real-time tasks and peripherals with excellent performance

Part 6:

An unmodified COTS 28-core, cache-coherent, SSRT configuration optimised for digital signal processing

Employs the technologies described in the single and dual-core SSRT configuration

S/Labs' interconnect in 28-core configuration



- In this illustration, a 1024-bit wide cache coherent interconnect, is configured for 28-wide symmetric multi processing (SMP)
  - Supports a small number of I/O peripherals (Ethernet, CAN, ...)

SYNAPTIC LABORATORIES LTD.
Concepts for Composable Dependable Architectures &
Costs of Hardware-Support for Dependability (DATE 2016)

SSRT supports multiple timing analysis techniques

- Integrated hardware support for
  multiple timing analysis techniques **in parallel**
  makes SSRT the ideal real-time platform
  - Supports rapid switching between
    true LRU and random cache-line eviction
    in a fully-associative cache
    to support different timing analysis schemes
- The end-user can therefore select and compare
  different timing analysis algorithms
  or
  use different timing analysis techniques for different tasks
- We explicitly track the evolving requirements of:
  - AbsInt, Rapita, Proxima, ...
  - *We acknowledge and are grateful for the long term collaboration support from AbsInt*

Part 8:

Broad across-industry expectations wrt. realtime architectures

and

How the SSRT architecture satisfies those requirements

SYNAPTIC LABORATORIES LTD.
Concepts for Composable Dependable Architectures &
Costs of Hardware-Support for Dependability (DATE 2016)

Broad across-industry expectations

- **Real-time capabilities in multi-core architectures are almost universally viewed as**

  SECOND TIER PRIORITIES

  when compared to maintaining ***backwards compatibility*** in the form of

  best-effort software price / performance ratios that are roughly competitive against COTS non-real time multi-core solutions

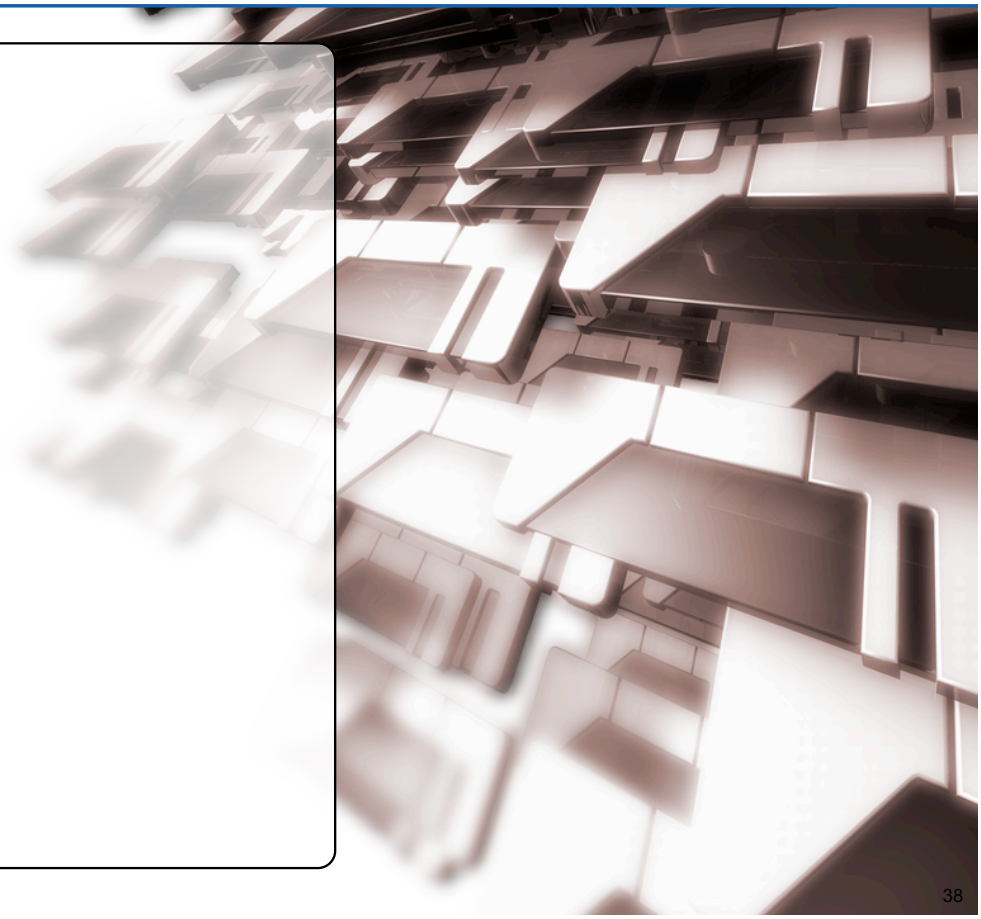
- ✓ ***S/Labs' SSRT cache-coherent shared memory architecture will simultaneously achieve FASTER real-time and best-effort software performance:***
  - ✓ *by employing faster more efficient tech. at comparable costs*
  - ✓ *increasing software performance by reducing contention*

Broad across-industry expectations

- Specifically, end-users are looking for:
  1. **Support for (cache-coherent) *shared-memory* software**
     - Based on our analysis, time-analysable message passing architectures cannot deliver commercially competitive cache-coherent shared memory performance over 2 or more cores
  2. **Support for their trusted AMP and *SMP* real-time operating systems**
  3. **Support running *memory intensive* best-effort tasks fast while *concurrently* running *memory intensive* time-analysable tasks *with tight bounds***
  4. **Support for *existing* instruction sets and development tool suites**
     - Using either low-area COTS cores and/or high-performance COTS cores
     - Support for their trusted / mandated task scheduling schemes
     - Robust space and time partitioning, priority driven scheduling, ...
  5. **Support for *their* preferred time-analysis scheme(s) with tight bounds**
     - The computer architecture **must not impose/dictate** a timing method on to the end-user
- Only solutions that satisfy all 5 points are candidates to achieve ***sufficient broad industry*** acceptance to achieve economies of scale in manufacture
  - This is why competing real-time multi-core proposals are **not** widely adopted today
  - ✓ **Only S/Lab's multi-core architecture satisfies all these 5 points and more...**

Part 9:

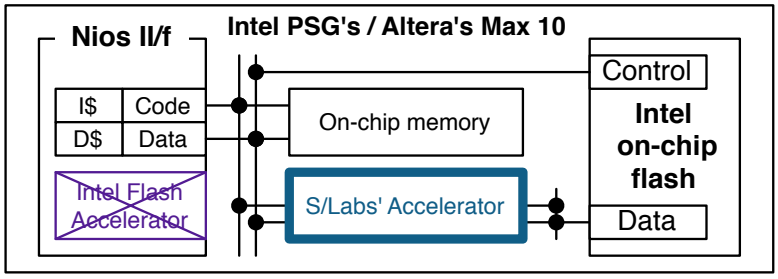
S/Labs' 4 phase commercialisation roadmap for SSRT



S/Labs' 4 phase commercialisation roadmap

1. **Develop a range of private fully-associative cache technologies that *accelerate* per-core performance for best-effort tasks in FPGA**
   a. **For example:**
   a *tiny* flash accelerator for up to 4x faster best-effort software performance on the Nios II/f when executing code from the on-chip FLASH of the Altera Max10® FPGA

   Up to 4x faster and 41x smaller than Altera's Flash Accelerator which was designed specifically to accelerate the Max 10 on-chip flash



Stand-alone product available for license today

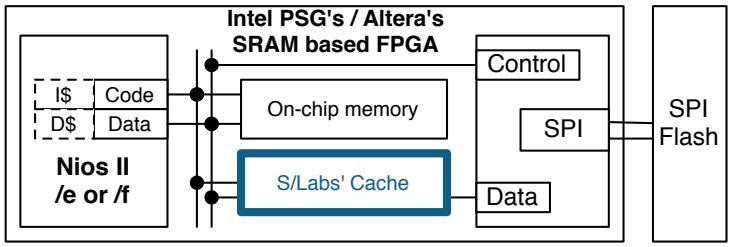
- *This tiny cache (35 4LUT, 1KB SRAM) informs the commercial development of the tiny caches for bus-master peripherals in SSRT*

S/Labs' 4 phase commercialisation roadmap

1. **Develop a range of private fully-associative cache technologies that *accelerate* per-core performance for best-effort tasks in FPGA**
   b. **S/Labs up to 32 way, fully associative cache with true LRU cache-line eviction**

   Accelerate on *and* off-chip FLASH and SDRAM on the Nios II/e e.g. our 4-way, 4KB **L1** cache on **Nios II/e**@100MHz wins up to **44x** acceleration of industry standard benchmarks run from 25 MHz serial flash

   Add our 4-way 1KB **L2** cache to **Nios II/f**@100MHz with 4K L1 I$ to win up to **1.8x** acceleration of industry standard benchmarks run from 1xSPI@25 MHz.



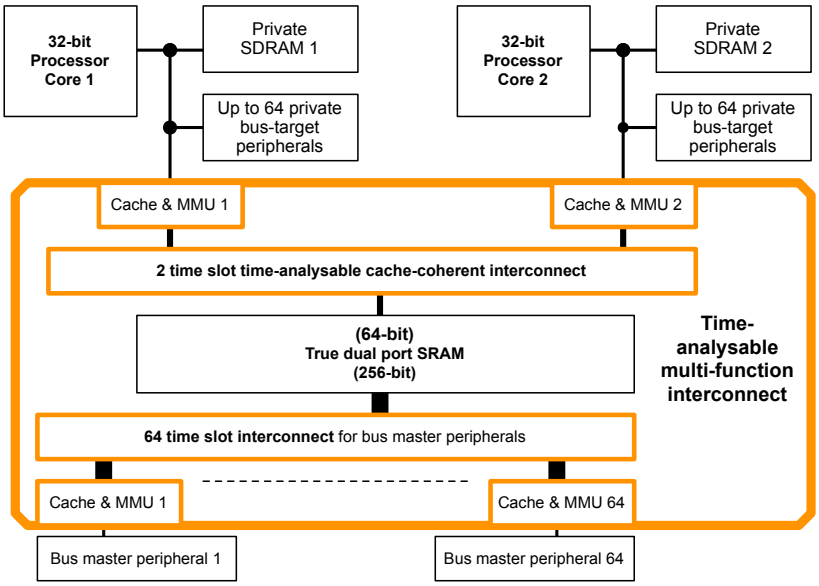
Stand-alone product available for license today

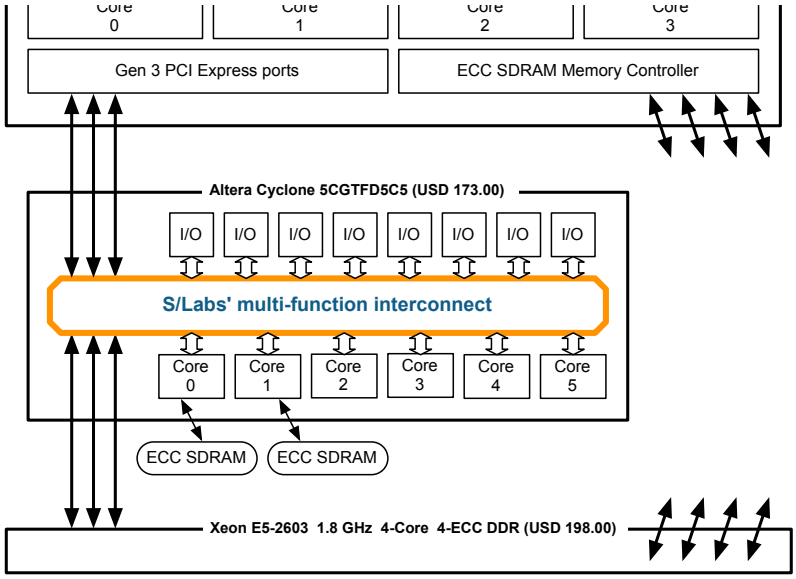
- *Our small fast cache designs inform the commercial design of the SSRT fully-associative write-update cache module with true-LRU and random cache-line eviction policies*

S/Labs' 4 phase commercialisation roadmap

2. Improve both the per core performance ***and*** the total system performance in a wide-range of shared memory mixed criticality FPGA applications, when compared to today's COTS FPGA based solutions

   Such as our 2 core solution previously discussed





4. Implement our field-proven technology as a hard-macro in a SoC FPGA device or as dedicated ASIC chips

In summary, only S/Labs' SSRT architectures permits:

- ACET and WCET performance gains to be won in FPGA (and ASIC)
- a commercially viable, incremental development process that is producing stand-alone products in addition to the final architecture
- the development of high performance commercially viable FPGA solutions in soft logic, that can be coupled to COTS ASIC cores, to inform future high-volume fully ASIC based solutions

S/Labs solutions address the pragmatic real-world needs of the broadest range of end-users / industries

www.synaptic-labs.com

Thank you for your attention
All enquiries welcome

Benjamin GITTINS cto@pqs.io

Ron KELSON ceo@pqs.io

We Need Assurance!

Brian Snow
U. S. National Security Agency
bdsnow@nsa.gov

Abstract

When will we be secure? Nobody knows for sure – but it cannot happen before commercial security products and services possess not only enough functionality to satisfy customers' stated needs, but also sufficient assurance of quality, reliability, safety, and appropriateness for use. Such assurances are lacking in most of today's commercial security products and services. I discuss paths to better assurance in Operating Systems, Applications, and Hardware through better development environments, requirements definition, systems engineering, quality certification, and legal/regulatory constraints. I also give some examples.

1. Introduction

This is an expanded version of the "Distinguished Practitioner" address at ACSAC 2005 and therefore is less formal than most of the papers in the proceedings.

I am very grateful that ACSAC chose me as a distinguished practitioner, and I am eager to talk with you about what makes products and services secure.

Most of your previous distinguished practitioners have been from the open community; I am from a closed community, the U.S. National Security Agency, but I work with and admire many of the distinguished practitioners from prior conferences.

I spent my first 20 years in NSA doing research developing cryptographic components and secure systems. Cryptographic systems serving the U.S. government and military spanning a range from nuclear command and control to tactical radios for the battlefield to network security devices use my algorithms.

For the last 14 years, I have been a Technical Director at NSA (similar to a chief scientist or senior technical fellow in industry) serving as Technical Director for three of NSA's major mission components: the Research Directorate, the Information Assurance Directorate, and currently the Directorate for Education and Training (NSA's Corporate University). Throughout these years, my mantra has been, "Managers are responsible for doing things right; Technical Directors are responsible for finding the right things to do."

There are many things to which NSA pays attention in developing secure products for our National Security Customers to which developers of commercial security offerings also need to pay attention, and that is what I want to discuss with you today.

2. Setting the context

The RSA Conference of 1999 opened with a choir singing a song whose message is still valid today: "Still Haven't Found What I'm Looking For". The reprise phrase was . . . *"When will I be secure? Nobody knows for sure. But I still haven't found what I'm looking for!"*

That sense of general malaise still lingers in the security industry; why is that? Security products and services should stop malice in the environment from damaging their users. Nevertheless, too often they fail in this task. I think it is for two major reasons.

First, too many of these products are still designed and developed using methodologies assuming random failure as the model of the deployment environment rather than assuming malice. There is a world of difference!

Second, users often fail to characterize the nature of the threat they need to counter. Are they subject only to a generic threat of an opponent seeking some weak system to beat on, not necessarily theirs, or are they subject to a targeted attack, where the opponent wants something specific of theirs and is willing to focus his resources on getting it?

The following two simple examples might clarify this.

Example 1: As a generic threat, consider a burglar roaming the neighborhood wanting to steal a VCR. First, understand his algorithm: Find empty house

(dark, no lights) try door; if open, enter, if VCR – take. If the door is resistant, or no VCR is present, find another dark house.

Will the burglar succeed? Yes, he will probably get a VCR in the neighborhood. Will he get yours? What does it take to stop him? Leave your lights on when you go out (9 cents a kilowatt-hour) and lock your door. That is probably good enough to stop the typical generic burglar.

Example 2: As a targeted threat, assume you have a painting by Picasso worth $250,000 hanging above your fireplace, and an Art thief knows you have it and he wants it. What is his algorithm? He watches your house until he sees the whole family leave. He does not care if the lights are on or not. He approaches the house and tries the door; if open, he enters. If locked, he kicks it in. If the door resists, he goes to a window. If no electronic tape, he breaks the glass and enters. If electronic tape is present, he goes to the siding on the house, rips some off, then tears out the fiberboard backing, removes the fiberglass insulation, breaks though the interior gypsum board, steps between the studs, and finally takes the painting and leaves.

It takes more effort to counter a targeted threat. In this case, typically a burglar alarm system with active polling and interior motion sensors as a minimum (brick construction would not hurt either). With luck, this should be enough to deter him. If not, at least there should be increased odds of recovery due to hot pursuit once the alarms go off.

There is no such thing as perfect security; you need to know how much is enough to counter the threat you face, and this changes over time.

3. What do we need?

NSA has a proud tradition during the past 53 years of providing cryptographic hardware, embedded systems, and other security products to our customers. Up to a few years ago, we were a sole-source provider. In recent years, there has come to be a commercial security industry that is attractive to our customers, and we are in an unaccustomed position of having to "compete." There is nothing wrong with that. *If* industry can meet our customer's needs, so be it.

Policy and regulation still require many of our customers to accept Government advice on security products. However, they really press us to recommend commercial solutions for cost savings and other reasons. Where we can, we do so. However, we do not do it very often because we still have not found what *we* are looking for – assurance.

Assurance is essential to security products, but it is missing in most commercial offerings today. The major shortfall is absence of assurance (or safety) mechanisms in *software*. If my car crashed as often as my computer does, I would be dead by now.

In fact, compare the software industry to the automobile industry at two points in its history, the 1930s and today. In 1930, the auto industry produced cars that could go 60 mph or faster, looked nice, and would get you from here to there. Cars "performed" well, but did not have many "safety features." If you were in an accident at high-speed, you would likely die.

The car industry today provides air bags, seat belts, crush zones, traction control, anti-skid braking, and a host of other safety details (many required by legislation) largely invisible to the purchaser. Do you *regularly* use your seat belt? If so, you realize that users *can* be trained to want and to use assurance technology!

The software security industry today is at about the same stage as the auto industry was in 1930; it provides performance, but offers little safety. For both cars and software, the issue is really assurance.

Yet what we need in security products for high-grade systems in DoD is more akin to a military tank than to a modern car! Because the environment in which our products must survive and function (battlefields, etc.) has malice galore.

I am looking forward to, and need, convergence of government and commercial security products in two areas: assurance, and common standards. Common standards will come naturally, but assurance will be harder – so I am here today as an evangelist for assurance techniques.

Many vendors tell me that users are not willing to pay for assurance in commercial security products; I would remind you that Toyota and Honda penetrated U.S. Markets in the 70's by differentiating themselves from other brands by improving reliability and quality! What software vendor today will become the "Toyota" of this industry by selling robust software?

4. Assurance: first definition

What do I mean by assurance? I'll give a more precise definition later, but for now it suffices to say that assurance work makes a user (or accreditor) more confident that the system works as intended, without flaws or surprises, even in the presence of malice.

We analyze the system at design time for potential problems that we then correct. We test prototype devices to see how well they perform under stress or when used in ways beyond the normal specification. Security acceptance testing not only exercises the product for its expected behavior given the expected

environment and input sequences, but also tests the product with swings in the environment outside the specified bounds and with improper inputs that do not match the interface specification. We also test with proper inputs, but in an improper sequence. We anticipate malicious behavior and design to counter it, and then test the countermeasures for effectiveness. We expect the product to behave safely, even if not properly, under any of these stresses. If it does not, we redesign it.

I want functions *and* assurances in a security device. We do not "beta-test" on the customer; if my product fails, someone might die.

Functions are typically visible to the user and commanded through an interface. Assurances tend to be invisible to the user but keep him safe anyway.

Examples would be thicker insulation on a power wire to reduce the risk of shock, and failure analysis to show that no single transistor failure will result in a security compromise.

Having seat belts in a car provides a safety function. Having them made of nylon instead of cotton is the result of assurance studies that show nylon lasts longer and retains its strength better in the harsh environment of a car's interior.

Assurance is best addressed during the initial design and engineering of security systems – not as after-market patches. The earlier you include a security architect or maven in your design process, the greater is the likelihood of a successful and robust design. The usual quip is, "He who gets to the interface first, wins".

When asked to predict the state of "security ten years from now," I focus on the likely absence of assurance, rather than the existence of new and wonderful things.

Ten years from now, there will still be security-enhanced software applications vulnerable to buffer overflow problems. These products will not be secure, but will be sold as such.

Ten years from now, there will still be security-enhanced operating systems that will crash when applications misbehave. They will not be secure either.

Ten years from now, we will have sufficient functionality, plenty of performance, but not enough assurance.

Otherwise, predicting ten years out is simply too hard in this industry, so I will limit myself to about five years. Throughout the coming five-year span, I see little improvement in assurance, hence little true security offered by the industry.

5. The current state of play

Am I depressed about this state of affairs? Yes, I am. The scene I see is products and services sufficiently robust to counter many (but not all) of the "hacker" attacks we hear so much about today, but not adequate against the more serious but real attacks mounted by economic enemies, organized crime, nation states, and yes, terrorists.

We will be in a truly dangerous stance: we will think we are secure (and act accordingly) when in fact we are not secure.

The serious enemy knows how to hide his activities. What is the difference between a hacker and a more serious threat such as organized crime? The hacker wants a *score*, and bragging rights for what he has obviously defaced or entered. Organized crime wants a *source*, is willing to work long, hard, and quietly to get in, and once in, wants to stay invisible and continue over time to extract what it needs from your system.

Clearly, we need confidence in security products; I hope we do not need a major bank-failure or other disaster as a wake-up call before we act.

The low-level hackers and "script-kiddies" who are breaking systems today and are either bragging about it or are dumb enough to be caught, are providing some of the best advertising we could ask for to justify the need for assurance in security products.

They demonstrate that assurance techniques (*barely*) adequate for a benign environment simply will not hold up in a malicious environment, so we *must* design to defeat malice. Believe me – there is malice out there, beyond what the "script-kiddies" can mount.

However, I do fear for the day when the easy threats are countered – that we may then stop at that level, rather than press on to counter the serious and pernicious threats that can stay hidden.

During the next several years, we need major pushes and advances in three areas: Scalability, Interoperability, and Assurance. I believe that market pressures will provide the first two, but not the last one – assurance.

There may or may not be major breakthroughs in new security functions; but we really do not need many new functions or primitives – if they come, that is nice. If they do not, we can make do with what we have.

What we really need but are not likely to get is greater levels of assurance. That is sad, because despite the real need for additional research in assurance technology, the real crime is that we fail to

use fully that which we already have in hand! We need to better use those confidence-improving techniques that we do have, and continue research and development efforts to refine them and find others.

I am not asking for the development of new science; the safety and reliability communities (and others) know how to do this – go and learn from them.

You are developers and marketers of security products, and I am sorry that even as your friend I must say, "Shame on you. You should build them better!" It is a core quality-of-implementation issue. The fact that teen-age hackers can penetrate many of your devices from home is an abysmal statement about the security-robustness of the products.

6. Assurance: second definition

It is time for a more precise definition. Assurances are confidence-building activities demonstrating that

1. $ The system's security policy is internally consistent and reflects the requirements of the organization,
2. $ There are sufficient security functions to support the security policy,
3. $ The system functions meet a desired set of properties and *only* those properties,
4. $ The functions are implemented correctly, and
5. $ The assurances *hold up* through the manufacturing, delivery, and life cycle of the system.

We provide assurance through structured design processes, documentation, and testing, with greater assurance provided by more processes, documentation, and testing.

I grant that this leads to increased cost and delayed time-to-market – a severe one-two punch in *today's* marketplace; but your customers are growing resistive and are beginning to expect, and to demand, better products *tomorrow*. They are near the point of chanting, "I'm mad as hell, and I'm not going to take it anymore!"

Several examples of assurance techniques come to mind; I will briefly discuss some in each of the following six areas: operating systems, software modules, hardware features, systems engineering, third party testing, and legal constraints.

7. Operating systems

Even if operating systems are not truly secure, they can at least remain benign (not actively malicious) if they would simply enforce a digital signature check on every critical module prior to each execution. Years ago, NSA's research organization wrote test code for a UNIX system that did exactly that. The performance degraded about three percent. This is something that is doable!

Operating Systems should be self-protective and enforce (at a minimum) separation, least-privilege, process-isolation, and type-enforcement.

They should be aware of and enforce security policies! Policies drive requirements. Recall that Robert Morris, a prior chief scientist for the National Computer Security Center, once said: "Systems built without requirements cannot fail; they merely offer surprises – usually unpleasant!"

Given today's common hardware and software architectural paradigms, operating systems security is a major primitive for secure systems – you will not succeed without it. This area is so important that it needs all the emphasis it can get. It is the current "black hole" of security.

The problem is innately difficult because from the beginning (ENIAC, 1944), due to the high cost of components, computers were built to share resources (memory, processors, buses, etc.). If you look for a one-word synopsis of computer design philosophy, it was and is SHARING. In the security realm, the one word synopsis is SEPARATION: keeping the bad guys away from the good guys' stuff!

So today, making a computer secure requires imposing a "separation paradigm" on top of an architecture built to share. That is tough! Even when partially successful, the residual problem is going to be covert channels. We really need to focus on making a secure computer, not on making a computer secure – the point of view changes your beginning assumptions and requirements!

8. Software modules

Software modules should be well documented, written in certified development environments, (ISO 9000, SEI-CMM level five, Watts Humphrey's Team Software Process and Personal Software Process (TSP/PSP), etc.), and *fully* stress-tested at their interfaces for boundary-condition behavior, invalid inputs, and proper commands in improper sequences.

In addition to the usual quality control concerns, *bounds checking* and *input scrubbing* require special attention. For bounds checking, verify that inputs are of the expected type: if numeric, in the expected range; if character strings, the length does not exceed the internal buffer size. For input scrubbing, implement reasonableness tests: if an input should be a single word of text, a character string containing multiple words is wrong, even if it fits in the buffer.

A strong quality control regime with aggressive bounds checking and input scrubbing will knock out the vast majority of today's security flaws.

We also need good configuration control processes and design modularity.

A good security design process requires review teams as well as design teams, and no designer should serve on the review team. They cannot be critical enough of their own work. Also in this world of multi-national firms with employees from around the world, it may make sense to take the national affinity of employees into account, and not populate design and review teams for a given product with employees of the SAME nationality or affinity. Half in jest I would say that if you have Israelis on the design team put Palestinians on the review team; or if Germans are on one, put French on the other. . . .

Use formal methods or other techniques to assure modules meet their specifications exactly, with no extraneous or unexpected behaviors – especially embedded malicious behavior.

Formal methods have improved dramatically over the years, and have demonstrated their ability to reduce errors, save time, and even save dollars! This is an under-exploited and very promising area deserving more attention.

I cite two examples of formal methods successes: The Microsoft SLAM static driver verifier effort coming on line in 2005, and Catherine Meadows' NRL Protocol Analyzer detecting flaws in the IKE (Internet Key Exchange) protocol in 1999. You may have your own recent favorites.

As our systems become more and more complex, the need for, and value of, formal methods will become more and more apparent.

9. Hardware features

Consider the use of smartcards, smart badges, or other hardware tokens for especially critical functions. Although more costly than software, when properly implemented the assurance gain is great. The form-factor is not as important as the existence of an isolated processor and address space for assured operations – an "Island of Security," if you will. Such devices can communicate with each other through secure protocols and provide a web of security connecting secure nodes located across a sea of insecurity in the global net.

I find it depressing that the hardware industry has provided hardware security functionality (from the Trusted Platform Group and others) now installed in processors and motherboards that is not yet accessed or used by the controlling software, whether an OS or an application.

10. Security systems engineering

How do we get high assurance in commercial gear?

a) How can we trust, or

b) If we cannot trust, how can we safely use, security gear of unknown quality?

Note the difference in the two characterizations above: *how we phrase the question may be important*. For my money, I think we need more focus on how to use safely security gear of unknown quality (or of uncertain provenance).

I do not have a complete answer on how to handle components of unknown quality, but my thoughts lean toward systems engineering approaches somewhat akin to what the banking industry does in their systems. No single component, module, or person knows enough about the overall transaction processing system to be able to mount a successful attack at any one given access point. To be successful the enemy must have access at multiple points and a great deal of system architecture data.

Partition the system into modules with "blinded interfaces" and limited authority where the data at any one interface are insufficient to develop a complete attack. Further, design cooperating modules to be "mutually suspicious," auditing and alarming each other's improper behavior to the extent possible.

For example: if you are computing interest to post to accounts there is no need to send the complete account record to a subroutine to adjust the account balance. Just send the current balance and interest rate, and on return store the result in the account record. Now the interest calculating subroutine *cannot* see the data on the account owner, and therefore cannot target specific accounts for theft or other malicious action. We need to trust the master exec routine, but minimize the number of subroutines we need to trust. Yes, I know this is over-simplified, but you get my drift.

In addition, to guard against "unintended extra functionality" within given hardware modules or software routines, the development philosophy needs to enforce something akin to "no-lone zones" in that no single designer or coder can present a "black-box" (or proprietary?) effort to the system design team that is tested only at its interfaces and is then accepted.

Review all schematics and code (in detail, line by line) for quality and "responsive to stated requirement" goals. This review should be by parties independent of the designer. This is expensive, but not

far from processes required today in many quality software development environments to address reliability and safety concerns.

This of course requires all tools (compilers, CAD support, etc.) used in the development environment to be free of malice; that can be a major hurdle and a difficult assurance task in and of itself (remember the Thompson compiler in "Reflections on Trusting Trust, CACM 1983)!

The "Open Source" movement may also provide value in this area. There are pluses and minuses with open source, but from the security viewpoint, I believe it is primarily a plu**s.**

Further architectural constraints may be imposed to make up for deficiencies in certain modules. Rather than (or in addition to) encryption in application processes prior to transmission to other sites which could be bypassed or countered by a malicious operating system, you might require site-to-site transmissions to go through an encrypting modem or other in-line, non-bypassable link encryptors.

Link encryption in addition to application layer encryption is an example of a "Defense in Depth" strategy that attempts to combine several weak or possibly flawed mechanisms in a fashion robust enough to provide protection at least somewhat stronger than the strongest component present.

Synergy, where the strength of the whole is greater than the sum of the strength of the parts, is highly desirable but not likely. We must avoid at all costs the all-too-common result where the system strength is less than the strength offered by the strongest component, and in some worst cases less than the weakest component present. Security is so very fragile under composition; in fact, secure composition of components is a major research area today.

Good *system* security design today is an art, not a science. Nevertheless, there are good practitioners out there that can do it. For instance, some of your prior distinguished practitioners fit the bill.

This area of "safe use of inadequate components" is one of our hardest problems, but an area where I expect some of the greatest payoffs in the future and where I invite you to spend effort.

11. Third party testing

NIST (and NSA) provide third-party testing in the National Information Assurance Partnership Laboratories (NIAP labs), but Government certification programs will only be successful if users see the need for something other than vendor claims of adequacy or what I call "proof by emphatic assertion – Buy me, I'm Good."

If not via NIST or other government mechanism, then the industry must provide *third-party* mediation for vendor security claims via consortia or other mechanisms to provide *independent* verification of vendor claims *in a way understandable by users*.

12. Market/legal/regulatory constraints

Market pressures are changing, and may now help drive more robust security functionality. The emergence of e-commerce in the past decade as a driver for secure internet financial transactions is certainly helpful, as is the entertainment industry's focus on digital rights management. These industries certainly want security laid on correctly and robustly!

I hope citizens will be able to use the emerging mechanisms to protect personal data in their homes, as well as industry using the mechanisms to protect industry's fiscal and intellectual property rights. It is simply a matter of getting the security architecture right.

I wonder if any of the industry consortia working on security for digital rights management and/or electronic fiscal transactions have citizen advocates sitting on their working groups.

Lawsuits might help lead to legal "fitness-for-use" criteria for software products – much as other industries face today. This could be a big boon to assurance – liability for something other than the quality of the media on which a product is delivered!

Recall that failure to deliver expected functionality can be viewed, in legal parlance, as providing an "attractive nuisance" and is often legally actionable.

One example is a back yard swimming pool with no fence around it. If a neighbor's child drowns in it, you can be in deep trouble for providing an attractive nuisance. Likewise, if you do a less than adequate job of shoveling snow from your walk in winter (providing the appearance of usability) you can be liable if someone slips on the ice you left on the surface. Many software security products today are attractive nuisances!

All you need do is to Google "Software Quality Lawsuits" or a similar phrase, and you can find plenty of current examples of redress sought under law for lack of quality in critical software. Do not attempt to manage defects in software used in life-critical applications. Remove them during the development and testing processes! People have died due to poor software in medical devices, and the courts are now engaged; the punitive awards can be significant.

One example of a lawsuit already settled: *General Motors Corp. v. Johnston* (1992). A truck stalled and was involved in an accident because of a defect in a PROM, leading to the death of a seven-year old child. An award of $7.5 million in punitive damages against GM followed, in part due to GM knowing of the fault, but doing nothing.

There are social processes outside the courts that can also drive vendors toward compliance with quality standards.

One of the most promising recent occurrences in the insurance industry was stated in the report of Rueschlikon 2005 (a conference serving the insurance industry). Many participants felt that, "The insurance industry's mechanisms of premiums, deductibles, and eligibility for coverage can incent best practices and create a market for security . . . This falls in line with the historic role played by the insurance industry to create incentives for good practices, from healthcare to auto safety . . . Moreover, the adherence to a set of best practices suggest that if they were not followed, firms could be held liable for negligence."

Bluntly, if your security product lacks sufficient robustness in the presence of malice, your customers will have to pay more in insurance costs to mitigate their risks.

How the insurance industry will measure best practices and measure compliance are still to be worked out, but I believe *differential* pricing of business disaster recovery insurance based in part on quality/assurance (especially of security components) is a great stride forward in bringing market pressure to bear in this area!

13. Summary

In closing, I reiterate that what we need most in the future is more assurance rather than more functions or features. The malicious environment in which security systems must function *absolutely requires* the use of strong assurance techniques.

Remember: most attacks today result from failures of assurance, not failures of function.

Rather than offer predictions, try for a self-fulfilling prophecy – each of us should leave this conference with a stronger commitment to using available assurance technology in products! It is not adequate to *have* the techniques; we must *use* them!

We have our work cut out for us; let's go do it.

In closing, I would like to thank Steven Greenwald, Brad Martin, and Greg Shipley for their insights and help in preparing this article.