Secure Integration of Cryptographic Primitives

Patrick Schaumont
schaum@vt.edu
ECE Department, Virginia Tech

A System contains Hardware & Software

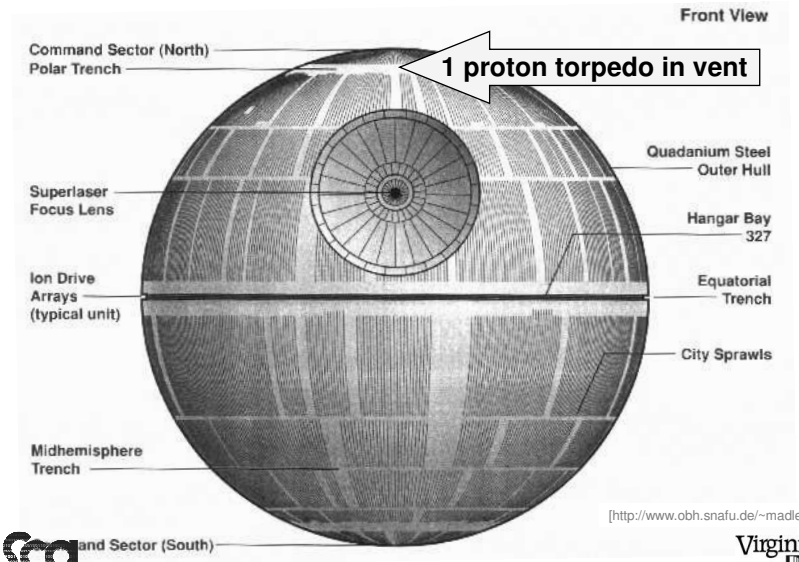
Smart Card

Mobile Biometrics

Networked Sensors

- Systems are driven by major processor architectures
  - X86, ARM, 8051, PIC
  - But real products differentiate using hardware
- Secure hardware presents a key value proposition

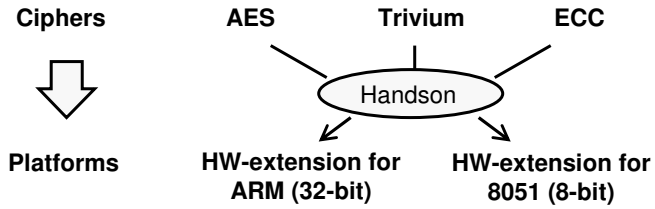
Key Value = Protect the weakest link



[http://www.obh.snafu.de/~madley/starwars/]

Objectives of this talk

- Provide an appreciation for the strengths and weaknesses of hardware and software in secure embedded systems
- Demonstrate different integration strategies for hardware and software
- Experiment with sample applications and platforms



Intended Audience

- Cryptographic Engineers
  - Improve crypto implementations with a system-level approach
- Hardware Engineers
  - Address the system integration issues of their components
- Secure Software Engineers
  - looking to leverage advantages of secure hardware
- Experimentalists

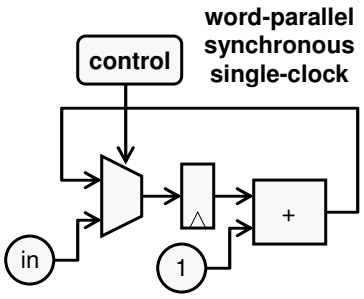
Organization of this talk

- Part 1: Hardware versus Software
  - Intro, HW/SW Codesign in Secure Applications
  - Hands-on - Design of a simple cipher
- Part 2: Hardware-Software Interfacing
  - HW/SW Interface Design and Modeling
  - Hands-on - Integration of a simple cipher
- Part 3: Alternative and Secure HW/SW Interfaces
  - Alternative HW/SW Interfaces
  - Hands-on - Optimization of a cipher system

Part 1
Hardware
versus Software

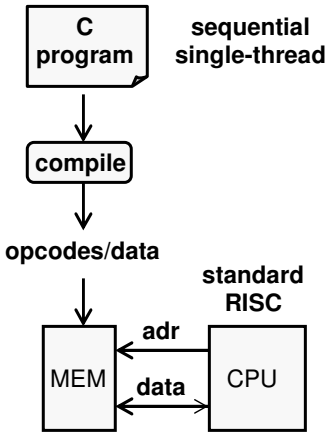
Hardware and Software - Assumptions

Hardware



'RTL' Register Transfer Level

Software



HW/SW, strengths and weaknesses

Hardware

- Parallel Execution of Ops
- Fixed in time (cycles), Variable in resources (area)
- Timing constraints are easy, Area constraints are hard.
- Flexibility is hard
- Complex Data Processing
- Modeling != Implementation
- IP (Intellectual Property) is hard to find, hard to transfer

Software

- Sequential Execution of Ops
- Fixed in resources, Variable in execution time
- Timing constraints are hard, Area constraints are easy(ier)
- Flexibility is easy
- Complex Control Processing
- Modeling == Implementation
- IP is easy to find, hard to transfer

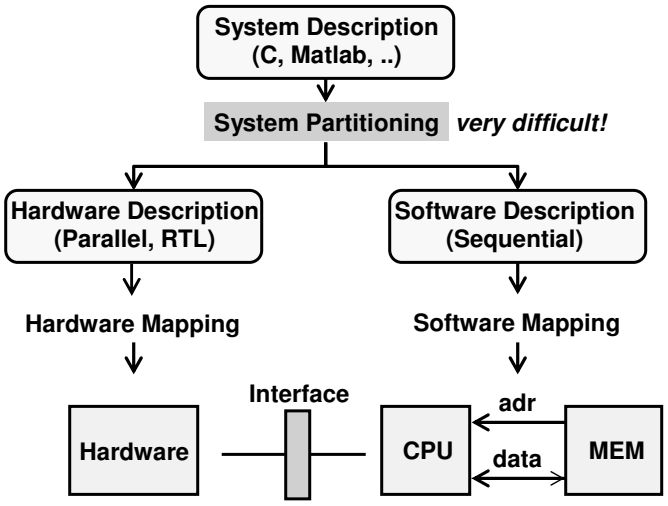
HW/SW, strengths and weaknesses

Hardware

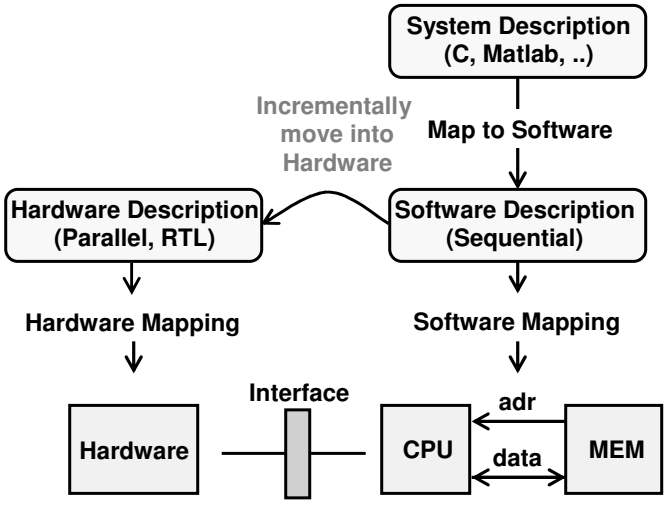
Software

In many respects, Hardware and Software use dual design philosophies that lead to dual design results

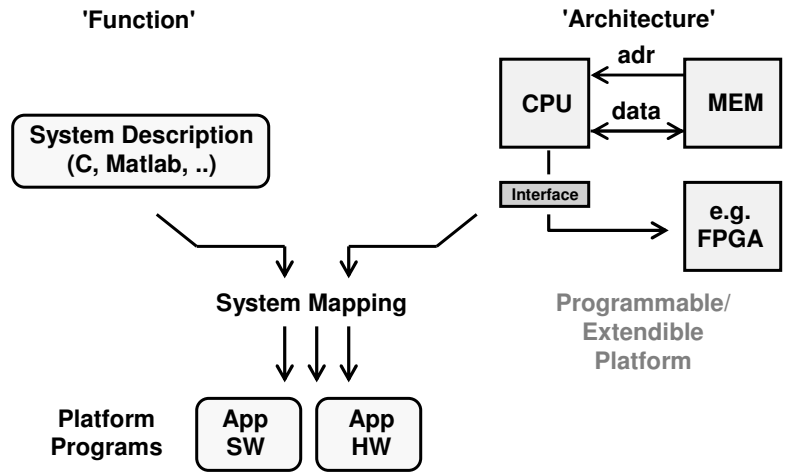
Concurrent Design of Hardware/Software



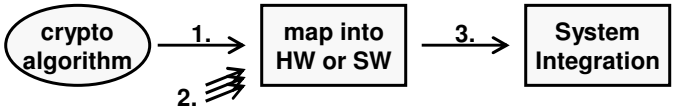
Variant 1 - Accelerate Software



Variant 2 - Platform-Based Design



Mapping Algorithms - Generic Concerns



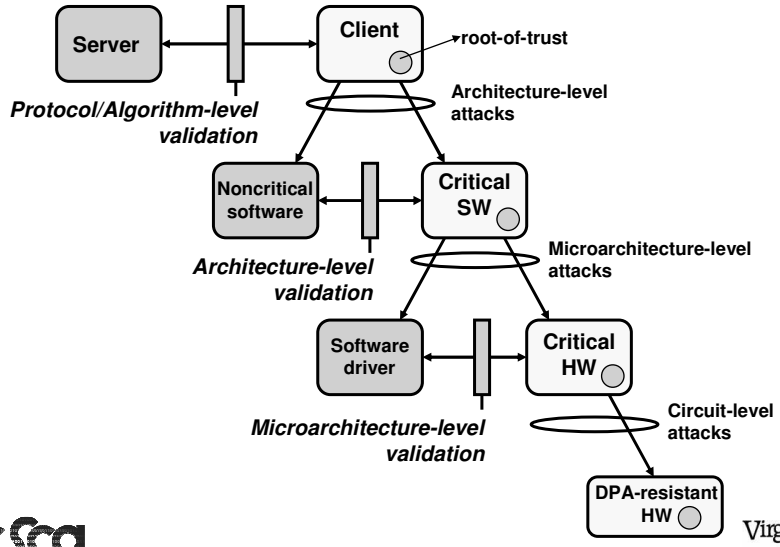
Hardware

1. Perfect match possible
2. Flexibility is hard
   - Multiple Crypto Algorithms
   - Encryption & Decryption
   - Modes of Operation
3. System Communication is hard

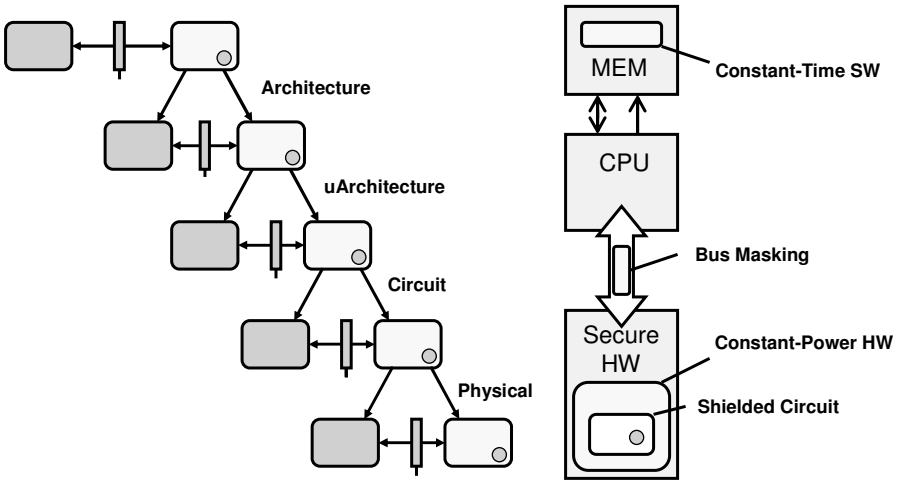
Software

1. Approximate match
   - Storage
   - Computations
   - Communication
2. Flexibility is easy
3. System Communication is easy

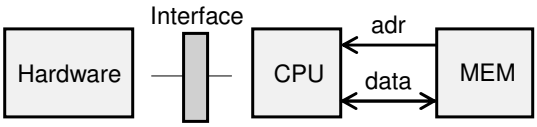
Mapping Algorithms - Security Concerns



Structured Side-Channel Defenses



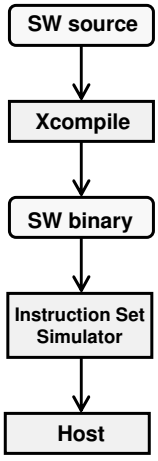
HW/SW Modeling - Abstraction Levels



Level	Communication	Computation	
Algorithm	Primitive Operations	Untimed Processes	HW == SW
TLM – PV	Transactions	Untimed Processes	HW and SW are distinct
TLM – PV-T	Transactions	Timed Processes	
RTL	Cycle Accurate	RTL	
Gates	Sub-cycle Accurate	Gates	

PV(-T) = programmers' view (with time)

Software Modeling and Simulation



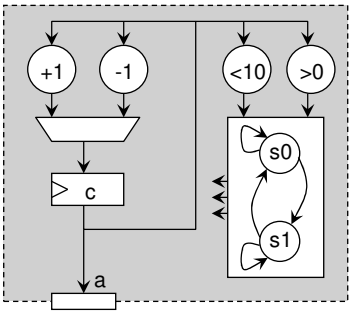
- *Native* SW simulation: directly on host machine
  - + No Xcompile, No ISS
  - - No Timing, only functional simulation
- *Interpreted* SW simulation: uses Xcompile, ISS, simulate per instruction
  - + Most flexible, most accurate
  - - Slow (100x .. 1000X)
- *Compiled* SW simulation: uses Xcompile, translate SW binary to host binary
  - + Much faster then interpreted (10x .. 100x)
  - - Not universal, requires specialized translator
- *Instruction-accurate* vs *Cycle-accurate*
- Simulators trade off speed, accuracy, visibility

GEZEL

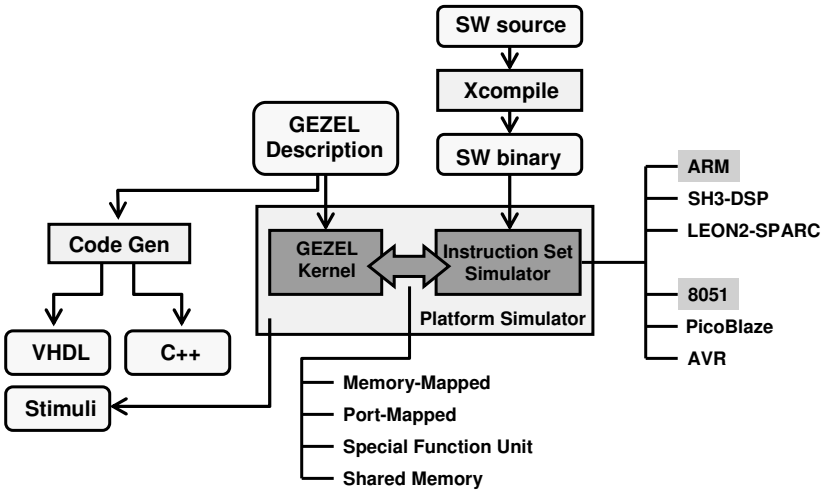
- Cycle-based Hardware Description Language
  - Deterministic and Implementation-oriented
  - Based on split control/datapath modeling (FSMD)
  - Easy to learn and use - 11-page LRM
- Hardware Simulation Kernel
  - Open-source (C++) with co-simulation backend
  - Library block concept
  - Toggle/Operation Profiler
- VHDL/Testvector Backend

Example FSMD Model

```
dp updown(out a : ns(4)) {
  reg c  : ns(4);
  sfg inc { c = c + 1;
            a = c;     }
  sfg dec { c = c - 1;
            a = c;     }
}

fsm ctl_updown(updown) {
  initial s0;
  state   s1;
  @s0 if (c < 10) then (inc) -> s0;
                  else (dec) -> s1;
  @s1 if (c > 0)  then (dec) -> s1;
                  else (inc) -> s0;
}
```



Hardware/Software Codesign

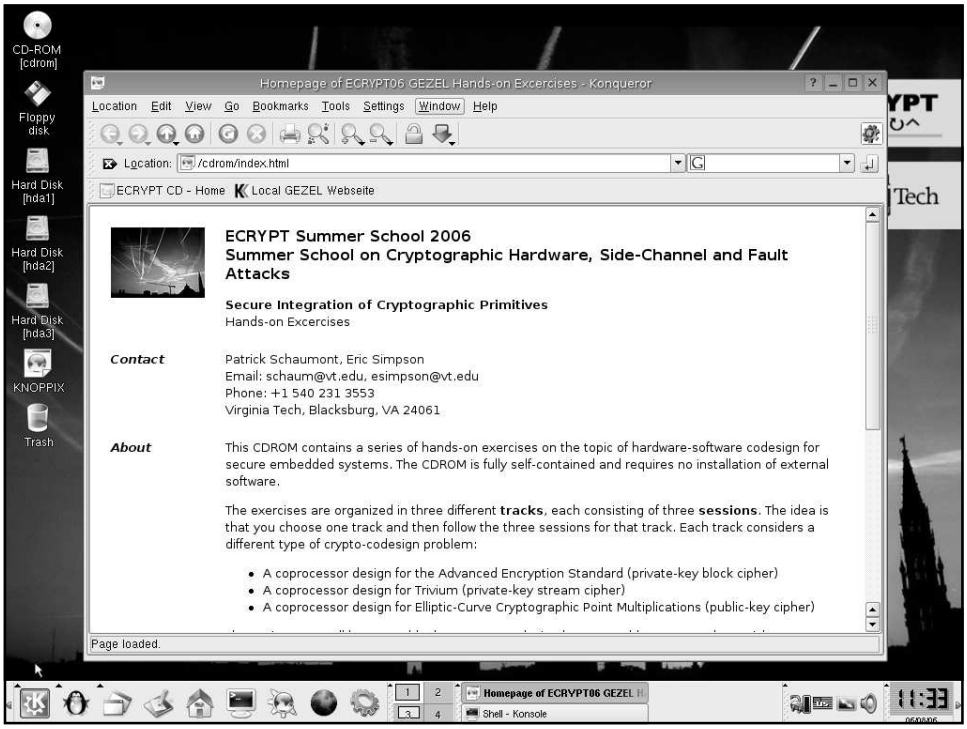


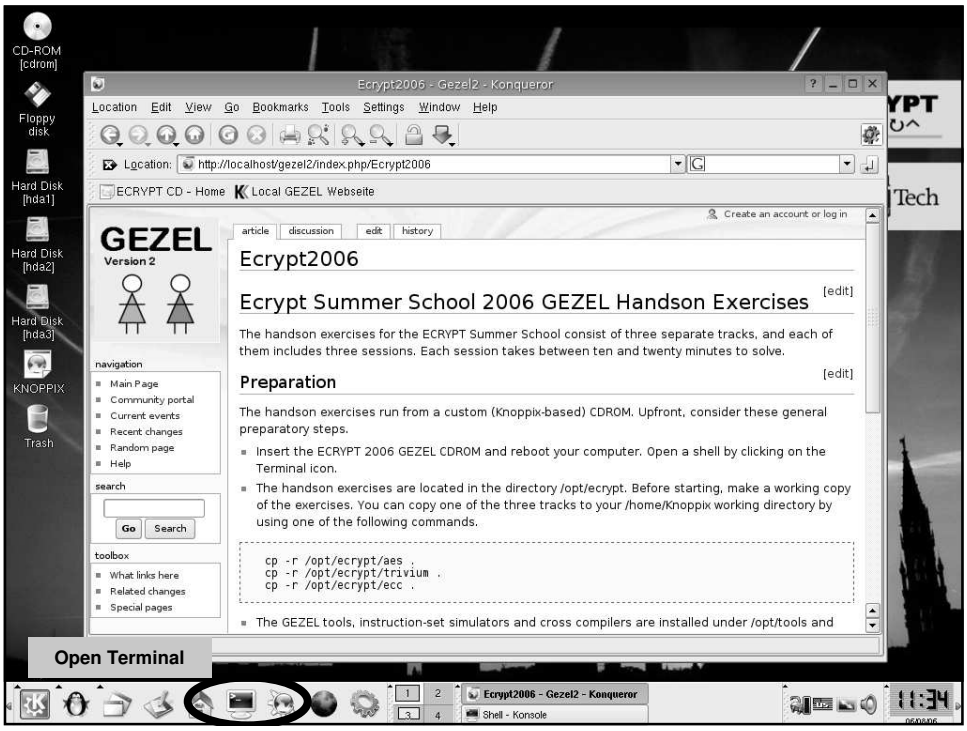
Summary

- Hardware and software
  - Strengths and weaknesses
  - Approaches for codesign
- Secure hardware and software
  - Tree of trust
- Modeling of hardware and software
  - Software modeling - cross-compilation, instruction-set simulation
  - Hardware modeling - GEZEL

A closer look

- Three possible tracks:
  - Advanced Encryption Standard
  - Trivium
  - Elliptic Curve Point Multiplier
- Step 1: Hardware development



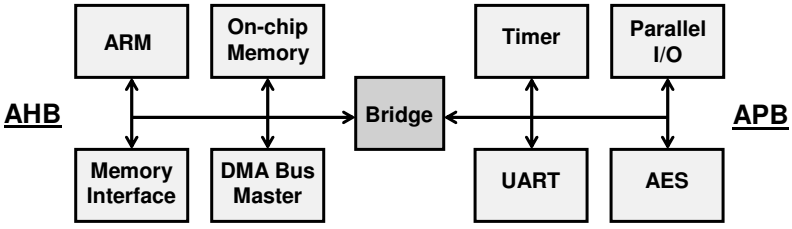


Part 2
Hardware-Software Interfacing

Hardware-Software Interfacing

- Microprocessor Bus
- Memory-mapped Hardware Encapsulation
- Synchronization
- Interface Models in GEZEL

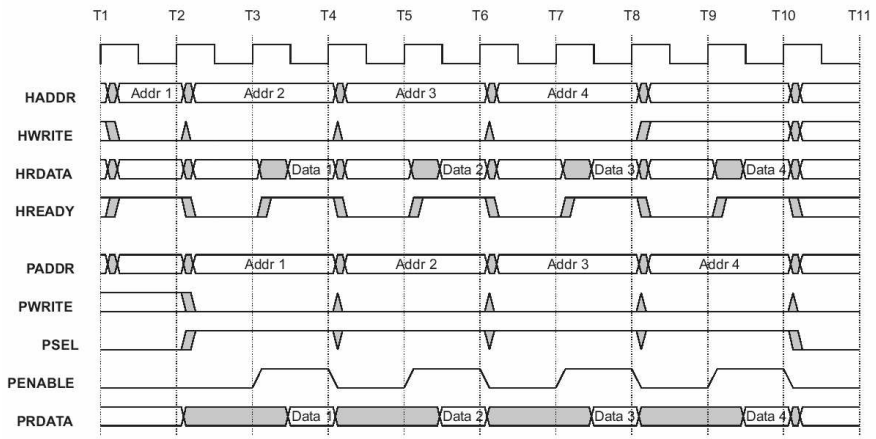
Microprocessor Bus - AMBA



- High-speed bus
- Unidirectional wires
- Multi-master
- Pipelined transfers
- Burst transfers
- Split transfer (decouple M-S)

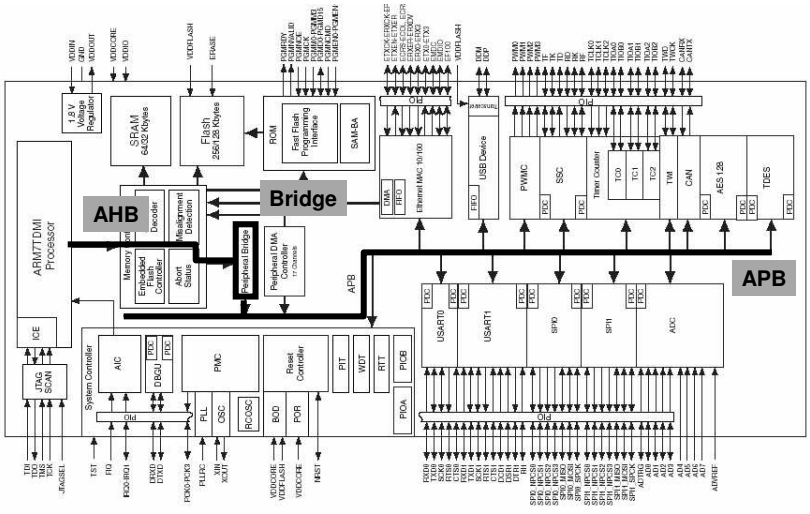
- Single-master bus (bridge)
- Unidirectional wires
- Simple read/write transfers

Sample Transfer: Periph->Bridge->ARM

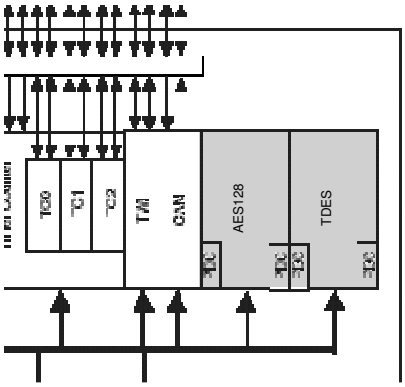


[AMBA Specification Rev. 2.0]

Example - AHB/APB in Atmel AT91SAM7

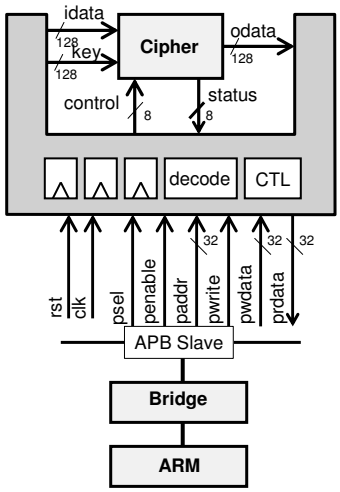


Example - AHB/APB in Atmel AT91SAM7



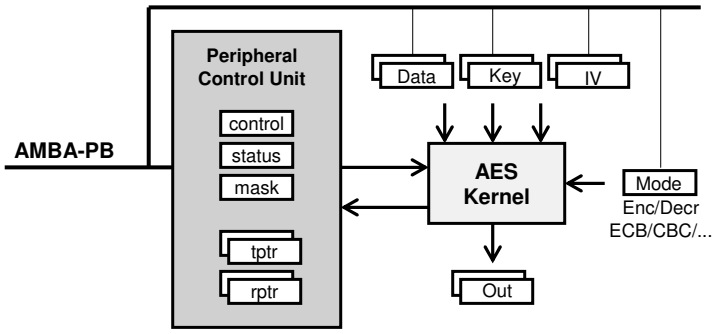
- AES and Triple-DES Hardware
- Multiple Modes of Operation
  - ECB, CBC, OFB, CFB, CTR
- Memory-Mapped Registers
  - 22 for AES
  - 18 for TDES
- Flexible operation mechanisms
  - Stream or last-word-only
  - Interrupts
  - Direct Memory Access

Hardware Encapsulation



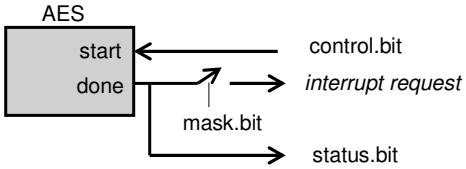
- Adapt Hardware I/O to bus
  - Match wordlengths
  - Multiplex multiple inputs/outputs
- Define instruction-set
  - HW I/O multiplexing
  - HW mode selection
  - Start/stop control
- Key issue: Balance communication and computation!
  - Data I/O from SW to HW does not benefit from CPU cache
  - Control handshaking of SW with HW does not benefit from CPU pipeline

Example: AES in AT91SAM7

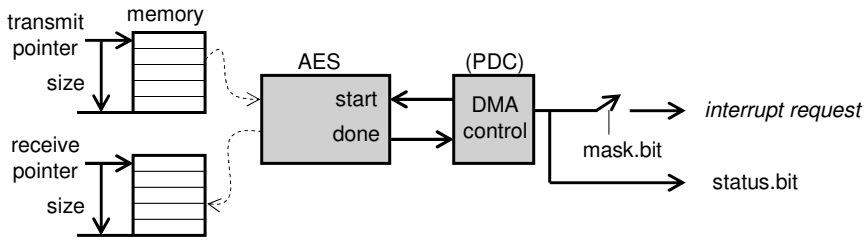


AES Control in AT91SAM7

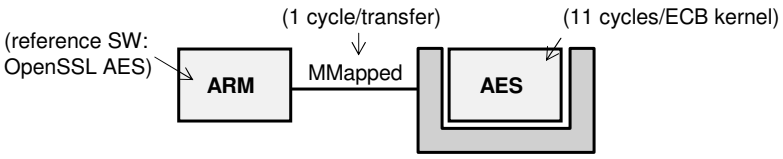
- Software-controlled operation

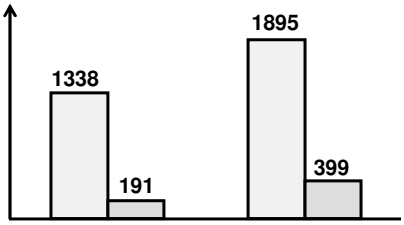


- Direct-Memory Access Controlled Operation



Encapsulation Overhead for AES





(Profiling cycle counts show approx. 10% overhead)

SW SW->HW->SW

- Using optimizing compiler:
  - HW has 7X improvement over optimized SW openSSL AES
  - But HW is over 17X slower than hardware kernel by itself!
- Using non-optimizing compiler:
  - HW has only 4.7X improvement over SW, 36X slower than kernel
  - Data movement is bottleneck

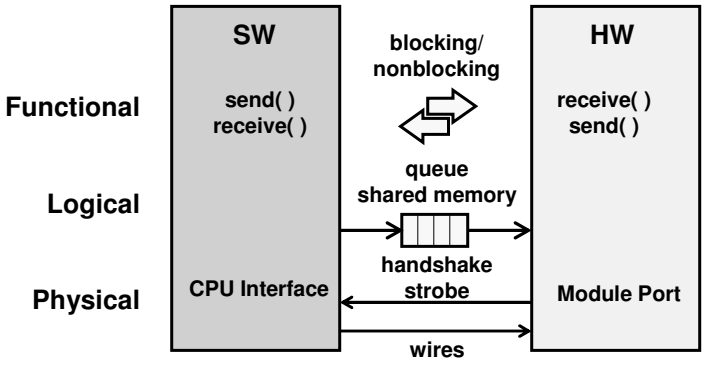
..and there's additional latency within SW!

	ROM/ RAM	Complexity	Typical Latency*
Large OS systems	**64M**	**> 100 processes**	**1000's**
Medium OS systems	**16M**	**A few 10's processes**	
Small OS systems	**4M**	**A few processes**	
Virtual Machines	**1M**	**A layered program**	**100's**
Custom OS MicroKernels	**256K**	**A few C programs**	**10's**
Microcontrollers	**64K**	**C program**	**1's**
MicroPrograms	**16K**	**Micro/Asm programs**	
	4K		
	1K		
	256		

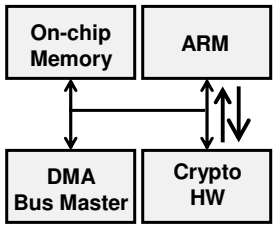
* Latency = Cycle Count Latency to go from a User Application to HW and back

Synchronization

- Hierarchy of activities to maintain data precedence
- Has profound impact on CPU/HW internal operation
  - Pipelining (HW/SW), wait states (HW), cache (SW)



Synchronization: Message Passing

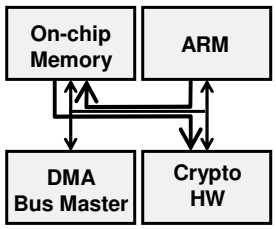



```
void send( int d) {
   *data = d;         // define data
   *req = !(*req);    // handshake
    while (*ack != *req) ;
}
```

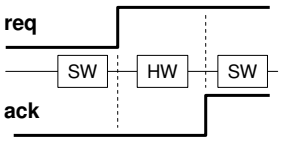
16 cycles round-trip
(with full optimization)

```
dp rcv(in d: ns(32);
       in req : ns(1); out ack : ns(1)) {
   reg rack : ns(1);
   reg rd : ns(32);
   always {
     rack = req;
     ack = rack;
     rd  = (ack != req) ? d : rd;
   }
}
```

Synchronization: Shared Memory




```
volatile unsigned int *shared =
  (volatile unsigned int *) 0x80001000;

int accumulate() {
   *req = !(*req);
    // hardware has access here
    while (*ack != *req) ;

    for (i=0; i<..; i++)
      acc += shared[i];
    return acc;
}
```

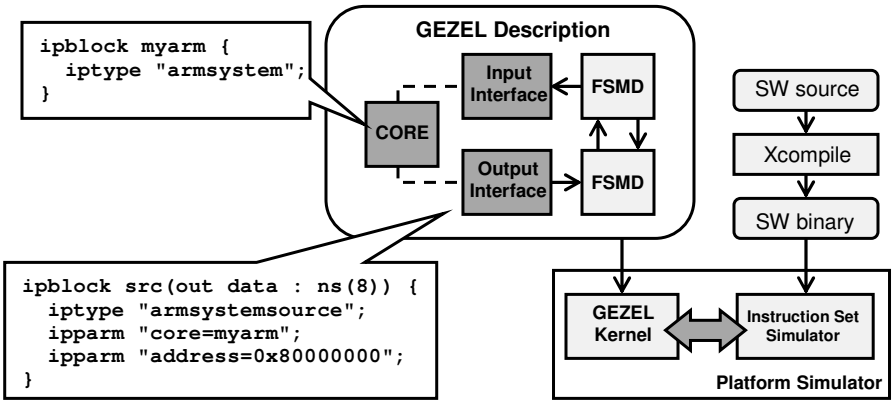


7.2 cycles per word
(accumulating 20K words in SW
from a 4K buffer written by hardware)

- Further enhancement using DMA

GEZEL Modeling

- IPBLOCK: User-defined Simulator Primitive to capture interface between HW model and core



Summary

- Microprocessor Bus
  - AMBA AHB/APB
  - Atmel AT91SAM7 microcontroller
- Hardware-microprocessor bus interfaces
  - AES in the AT91SAM7
  - Overhead of encapsulation
- Hardware-software communication
  - Synchronization - shared memory

A closer look

- at two target platforms:
  - ARM with Memory-mapped interface
  - 8051 with Port-mapped interface
- Step 2: Coprocessor Integration
  - AES
  - Trivium
  - ECC

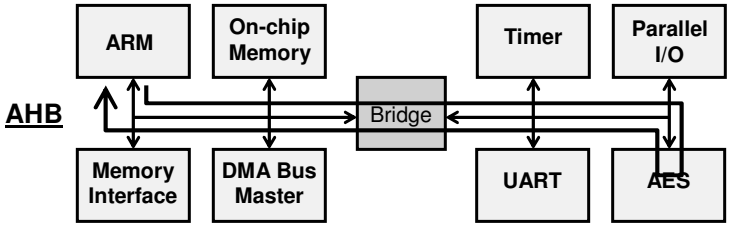
Part 3
Alternative and Secure Hardware-Software Interfaces

Alternative and Secure HW/SW interfaces

- Alternative Integration Strategies
  - Custom Instructions
  - Loosely coupled processors
- Secure Partitioning
  - Oracle partitioning
  - Process Isolation
- Beyond Hardware-Software Codesign
  - Components and Platforms

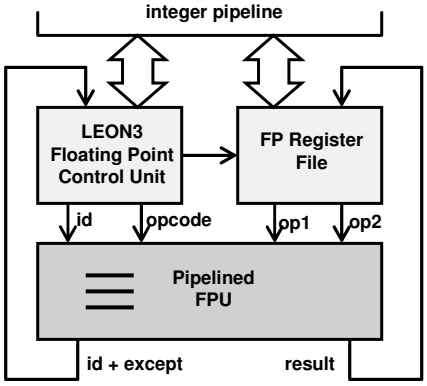
Alternative Integration Strategies

1. Pull computation into the CPU
   - Special Function Units in processor (ASIP)
   - Coprocessor bus and Dedicated Ports



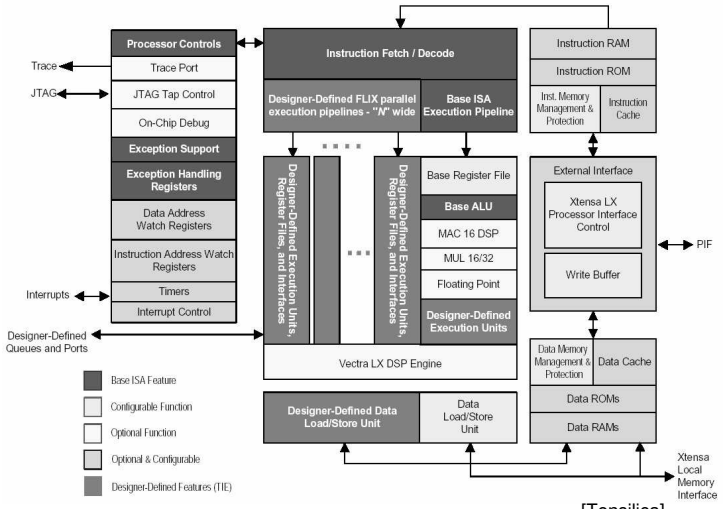
2. Push communication out of the CPU
   - Network on Chip
   - Shared Memory & Direct Memory Access
   - Streaming processing

Pull into CPU - LEON3 Example



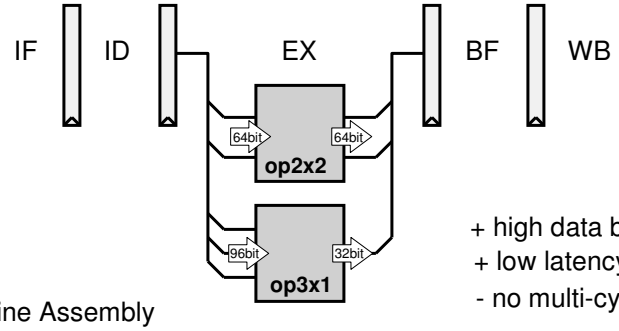
- Make use of dedicated processor instructions
  - Floating-point
  - General Coprocessor
- Tightly integrates to processor pipeline: pipeline stall, pipeline flush
- Compared to standard bus (APB), data bandwidth typically x8
  - Larger wordlength
  - One transfer/cycle

Pull into CPU - Xtensa Example



[Tensilica]

Pull into CPU – SimIt-ARM Example



+ high data bandwith
+ low latency
- no multi-cycle operation

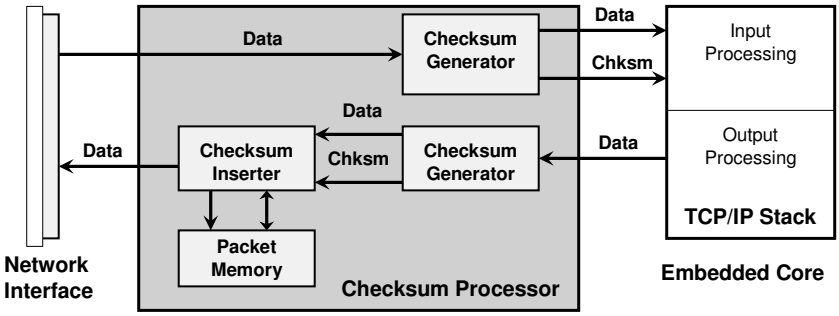
Inline Assembly

```
#define OP2x2_1(D1,D2,S1,S2) \
        asm volatile ("smullnv %0, %1, %2, %3": \
                      "=&r"(D1),"=&r"(D2): \
                      "r"(S1),"r"(S2));
```

Use in C, e.g.

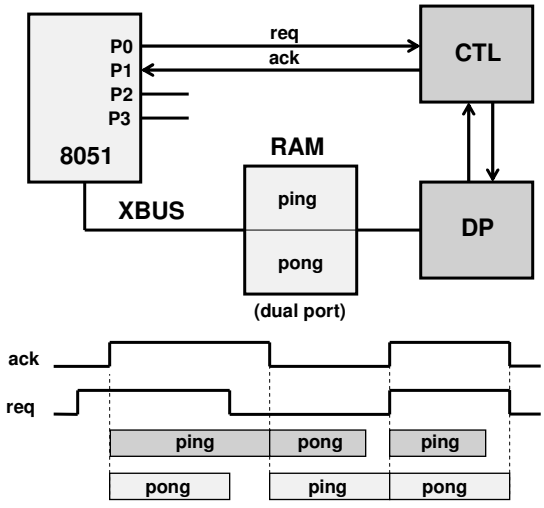
```
OP2x2_1(a, b, c, d);
```

Push out of CPU: Checksum Example

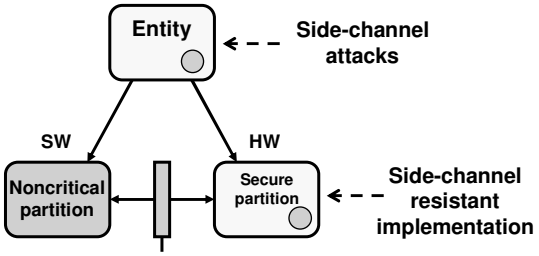


- Compared to LEON2/50MHz (using Virtex-2)
  - 66X energy savings for checksum function in HW over SW
  - 33x performance improvement
  - at 32% area overhead due to additional HW

Push out of CPU – Ping Pong Buffer Example



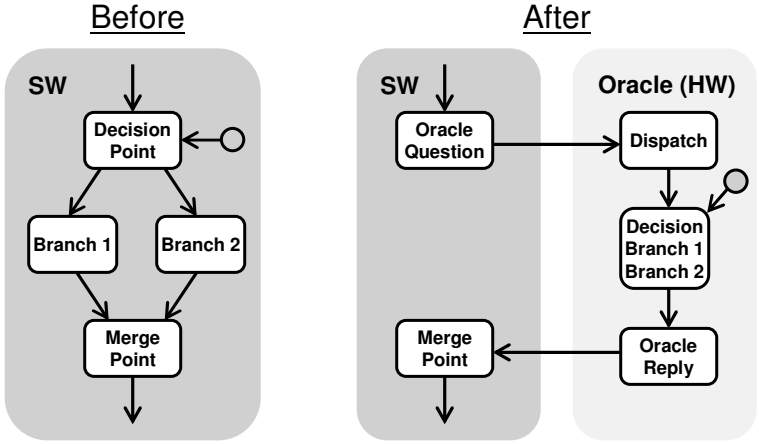
Secure Partitioning



- Partition a design on the basis of side channel leakage of the root-of-trust
  - Oracle Partitioning
  - Process Isolation

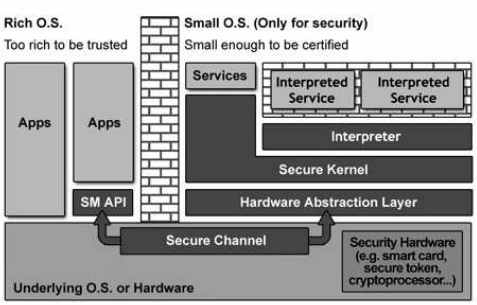
Oracle Partitioning

- Hide control-flow side channels by deferring decisions to an oracle



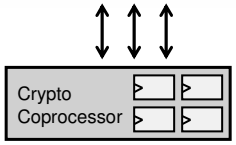
Process Isolation

See also A. Tanenbaum's Article in IEEE Computer, May 2006.



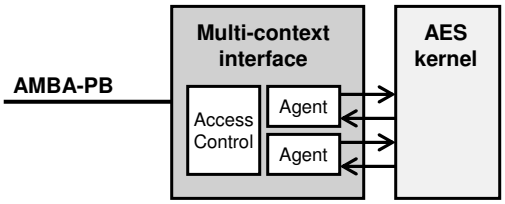
http://www.trusted-logic.com/
(now ARM TrustZone)

Need to extend the concept of isolation into new processor hardware



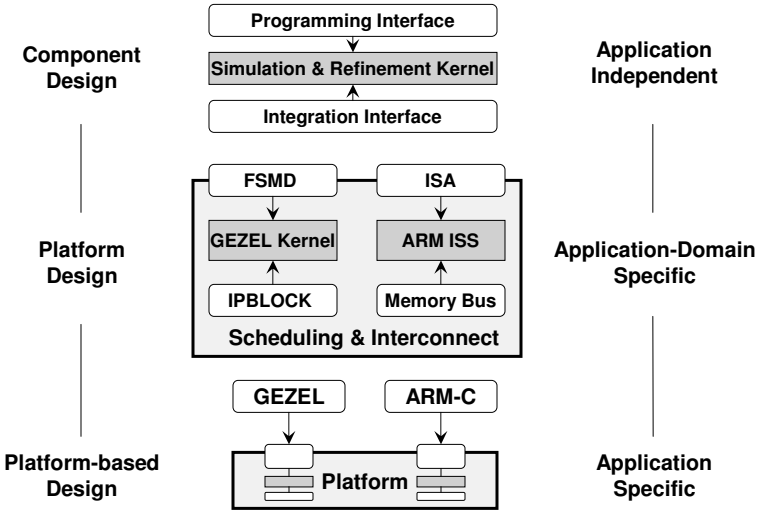
Example: Context-switched AES

[Herwin Chan]

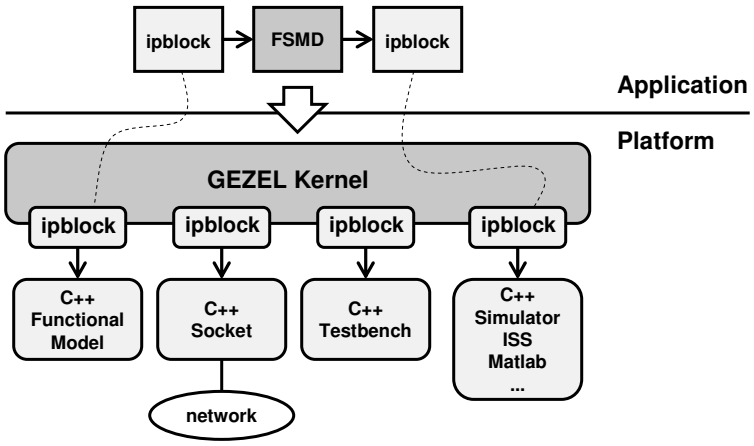


- Agents manage coprocessor state (registers) locally
- Agents establish a secure channel to application SW
  - Access Control generates a unique random number nonce
  - Subsequent SW accesses are authenticated by nonce
- Context switch much faster than using software
  - About 16X for typical AES including mode-state

Components and Platforms



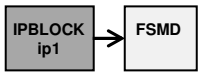
Building platforms using ipblock



Building platforms using ipblock

GEZEL Spec

```
ipblock ip1(out data : ns(8)) {
  iptype "myblock";
  ipparm "parm1=myparm";
}
```



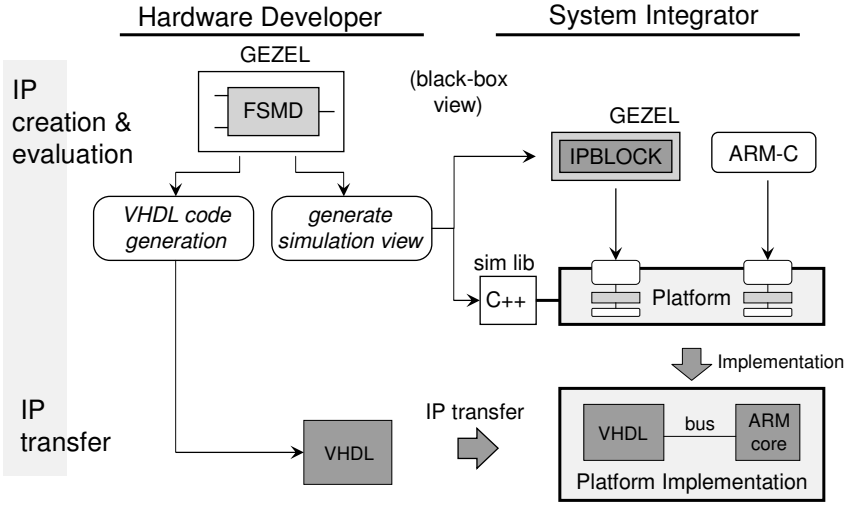
C++ Implementation of ipblock

```
class myblock {
  public:
   myblock();
   void run();              // called once per cycle
   void setparm(char *p); // called with p = "parm1.."
   ..
};
```

`libmyblock.so` Dynamically Compiled Library

@runtime

IP Reuse and Exchange



Summary

- Alternative Hardware-Software Interfaces
  - Custom Instructions
  - Loosely coupled coprocessors
- Secure Hardware-Software Interfacing
  - Partitioning - Oracle example
  - Isolation
- Ideas in Platform design
  - Extensibility through ipblock
  - Reusability through ipblock

A closer look

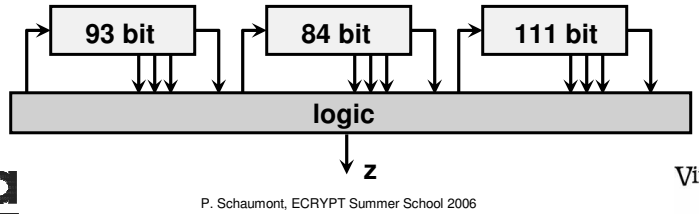
- at two alternative target platforms:
  - ARM with special-function units
  - 8051 with shared-memory interface
- Step 3: Coprocessor optimization
  - AES
  - Trivium
  - ECC

Design Examples

Example 1: Trivium in GEZEL

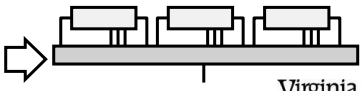
Pseudocode: [De Canniere et al, eSTREAM]

```
for i = 1 to N do
   t1 = s66 ^ s93
   t2 = s162 ^ s177
   t3 = s243 ^ s288
   zi = t1 ^ t2 ^ t3
   t1 = t1 ^ s91 & s92 ^ s171
   t2 = t2 ^ s175 & s176 ^ S264
   t3 = t3 ^ s286 & s287 ^ s69
  (  s1,   s2, ...,  s93) = (t3,   s1, ...,  s92)
  ( s94,  s95, ..., s177) = (t1,  s94, ..., s176)
  (s178, s179, ..., s288) = (t2, s178, ..., s287)
end for
```



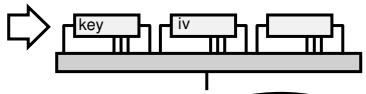
Ex1: Trivium Kernel in GEZEL

```
dp trivium(in  si : ns(288);  // state input
           out so : ns(288);  // state output
           out z  : ns(1)) {  // crypto bit out
  sig  t1,  t2,  t3 : ns(  1);
  sig t11, t22, t33 : ns(  1);
  sig saa           : ns( 93);
  sig sbb           : ns( 84);
  sig scc           : ns(111);
  always {
    t1  = si[ 65] ^ si[ 92];
    t2  = si[161] ^ si[176];
    t3  = si[242] ^ si[287];
    z   = t1 ^ t2 ^ t3;
    t11 = t1 ^ (si[ 90] & si[ 91]) ^ si[170];
    t22 = t2 ^ (si[174] & si[175]) ^ si[263];
    t33 = t3 ^ (si[285] & si[286]) ^ si[ 68];
    saa = si[  0: 92] # t33;
    sbb = si[ 93:176] # t11;
    scc = si[177:287] # t22;
    so  = scc # sbb # saa;
  }
}
```



Ex1: Trivium Key Schedule in GEZEL

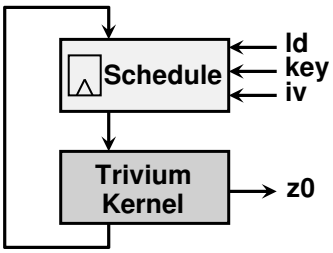
```
dp keyschedule(in  ld : ns(1);     // reload key & iv
               in  iv : ns(80);    // initialization vector
               in key : ns(80);    // key
               out  e : ns(1);     // output valid
               in  si : ns(288);   // state input
               out so : ns(288)) { // state output
  reg s        : ns(288); // state register
  reg cnt      : ns(11);  // initialization counter
  sig saa      : ns( 93);
  sig sbb      : ns( 84);
  sig scc      : ns(111);
  sig cte      : ns(111);
  always {
    saa  = ld ? key        : si[  0: 92];
    sbb  = ld ? iv         : si[ 93:176];
    cte  = 7;
    scc  = ld ? (cte << 108) : si[177:287];
    s    = scc # sbb # saa;
    so   = s;
    cnt  = ld ? 1152 : (cnt ? cnt - 1 : cnt);
    e    = (cnt ? 0 : 1);
  }
}
```



+ Control & Initialization

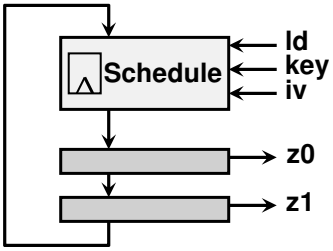
Ex1: 1-bit per cycle Trivium

```
dp triviumtop(in  ld : ns(1);     // reload key & iv
              in  iv : ns(80);    // initialization vector
              in key : ns(80);    // key
              out  z : ns(1);     // encrypted output
              out  e : ns(1)) {   // output valid
  sig si, so0 : ns(288);
  sig z0      : ns(1);
  use keyschedule(ld, iv, key, e, si, so0);
  use trivium    (so0, si, z0);
  always {
    z = z0;
  }
}
```



Ex1: 2-bit per cycle Trivium

```
dp trivium1 : trivium
dp triviumtop(in  ld : ns(1);     // reload key & iv
              in  iv : ns(80);    // initialization vector
              in key : ns(80);    // key
              out  z : ns(2);     // encrypted output
              out  e : ns(1)) {   // output valid
  sig si, so0, so1 : ns(288);
  sig z0, z1       : ns(1);
  use keyschedule(ld, iv, key, e, si, so1);
  use trivium    (so0, so1, z0);
  use trivium1   (so1,  si, z1);
  always {
    z = z0 # z1;
  }
}
```

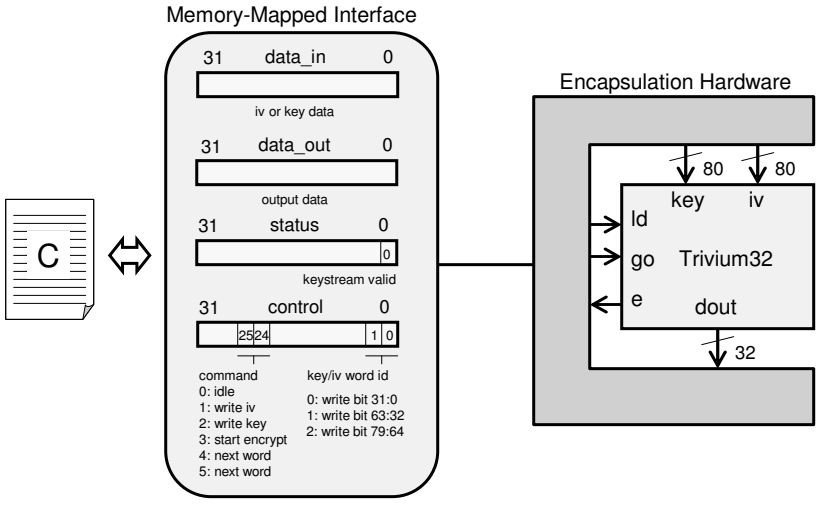


Ex1: GEZEL -> implementation

GEZEL -> (fdlvhd) -> VHDL -> (synplify, ise) -> FPGA

	Spartan3 XS400 -4	Virtex4 XC4VLX15 -12
Trivium1		
Flip-Flop	288 + 11	288 + 11
Slices (4-LUT)	160 (317)	161 (318)
Post-PAR clock	4.226 ns	2.019 ns
Trivium8		
Flip-Flop	288 + 8	288 + 8
Slices (4-LUT)	181 (353)	181 (351)
Post-PAR clock	3.987 ns	1.94 ns
Trivium32		
Flip-Flop	288 + 6	288 + 6
Slices (4-LUT)	267 (516)	267 (516)
Post-PAR clock	4.637 ns	2.075ns

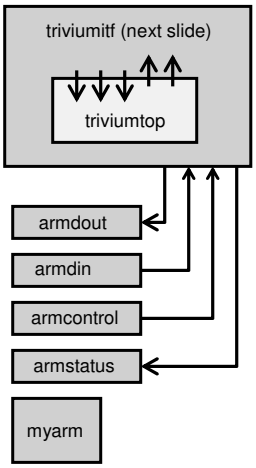
Example 2: Trivum Coprocessor for ARM



Ex2: Encapsulation Interface (1)

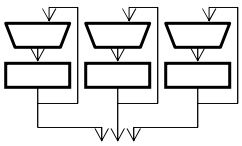
```
dp triviumtop(in  ld : ns(1);     // reload key & iv
              in  go : ns(1);     // stop/go signal
              in  iv : ns(80);    // initialization vector
              in key : ns(80);    // key
              out  z : ns(32);    // encrypted output
              out  e : ns(1)) {   // output valid
  sig so0, si : ns(288);
  use keyschedule(ld, go, iv, key, e, si, so0);
  use trivium320 (so0, si, z);
}
ipblock myarm {
  iptype "armsystem";     // instantiate ARM
  ipparm "exec=trivium";  // load executable 'trivium'
}
ipblock armdout(in data : ns(32)) {
  iptype "armsystemsink";      // data-out interface
  ipparm "core=myarm";         // connects to myarm
  ipparm "address=0x80000000"; // memory-map
}
ipblock armdin(out data : ns(32)) {
  iptype "armsystemsource";    // data-in interface
  ipparm "core=myarm";         // connects to myarm
  ipparm "address=0x80000004"; // memory-map
}
ipblock armstatus(in data : ns(32)) {
  iptype "armsystemsink";      // status interface
  ipparm "core=myarm";         // connects to myarm
  ipparm "address=0x80000008"; // memory-map
}
ipblock armcontrol(out data : ns(32)) {
  iptype "armsystemsource";    // control interface
  ipparm "core=myarm";         // connects to myarm
  ipparm "address=0x8000000C"; // memory map
}
```

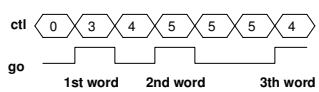
GEZEL System Topology



Ex2: Encapsulation Interface (2)

```
dp triviumitf(in  din    : ns(32);
              out dout   : ns(32);
              in  ctl    : ns(32);
              out status : ns(32)) {
  sig ld, go, e  : ns(1);
  sig iv, key    : ns(80);
  sig z          : ns(32);
  use triviumtop(ld, go, iv, key, z, e);
  reg ivr, keyr          : ns(80);
  sig ivr0, ivr1, ivr2 : ns(32);
  sig key0, key1, key2 : ns(32);
  reg oldread          : ns(3);
  always {
   // program new IV
   ivr0   = ((ctl[24:26] == 0x1) & (ctl[0:1] == 0x0)) ? din : ivr[31: 0];
   ivr1   = ((ctl[24:26] == 0x1) & (ctl[0:1] == 0x1)) ? din : ivr[63:32];
   ivr2   = ((ctl[24:26] == 0x1) & (ctl[0:1] == 0x2)) ? din : ivr[79:64];
   ivr    = ivr2 # ivr1 # ivr0;  iv = ivr;

   // program new KEY
   key0   = ((ctl[24:26] == 0x2) & (ctl[0:1] == 0x0)) ? din : keyr[31: 0];
   key1   = ((ctl[24:26] == 0x2) & (ctl[0:1] == 0x1)) ? din : keyr[63:32];
   key2   = ((ctl[24:26] == 0x2) & (ctl[0:1] == 0x2)) ? din : keyr[79:64];
   keyr   = key2 # key1 # key0;  key = keyvr;

   ld     = ((ctl[24:26] == 0x3) ? 1 : 0);  // control start
   status = e; // read status
   dout   = z; // read output data

   // trivium kernel control
   oldread = ((ctl[24:26]));
   go      = ((ctl[24:26] == 0x4) & (oldread ==0x5)) |
             ((ctl[24:26] == 0x5) & (oldread ==0x4)) |
             ((ctl[24:26] == 0x3) & (oldread ==0x0));
  }
}
```

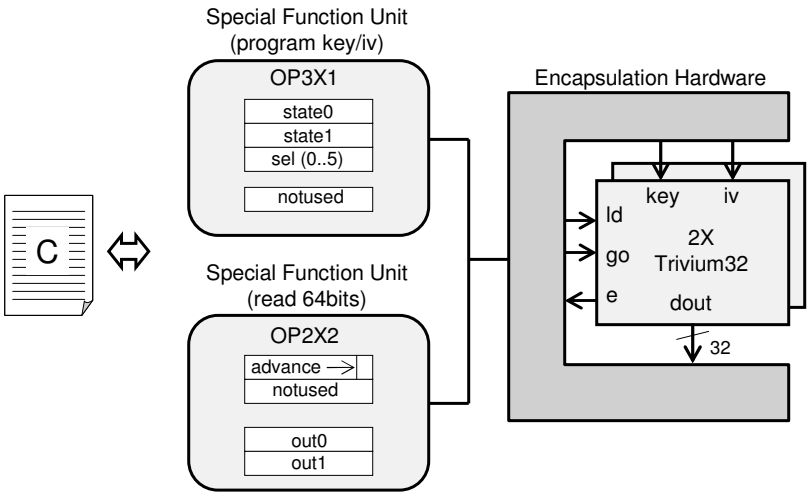




Ex2: Driver program in C

```
int main() {
  volatile unsigned int *data   = (unsigned int *) 0x80000004;
  volatile unsigned int *ctl    = (unsigned int *) 0x8000000C;
  volatile unsigned int *output = (unsigned int *) 0x80000000;
  volatile unsigned int *status = (unsigned int *) 0x80000008;

  // program iv
  *ctl  = (1 << 24);       // word 0
  *data = 0;
  *ctl  = (1 << 24) | 0x1; // word 1
  *ctl  = (1 << 24) | 0x2; // word 2

  // program key
  *ctl  = (2 << 24);       // word 0
  *data = 0x80;
  *ctl  = (2 << 24) | 0x1; // word 1
  *data = 0;
  *ctl  = (2 << 24) | 0x2; // word 2

  // run the key schedule
  *ctl  = 0;
  *ctl  = (3 << 24);  // start pulse

  while (! *status) {
    *ctl = (4 << 24);
    if (*status) break;
    *ctl = (5 << 24);
  }
  //  key stream is ready here
}
```

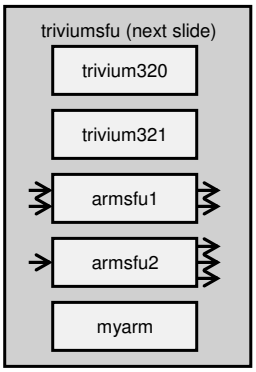

Example 3: Trivum Function Unit for ARM



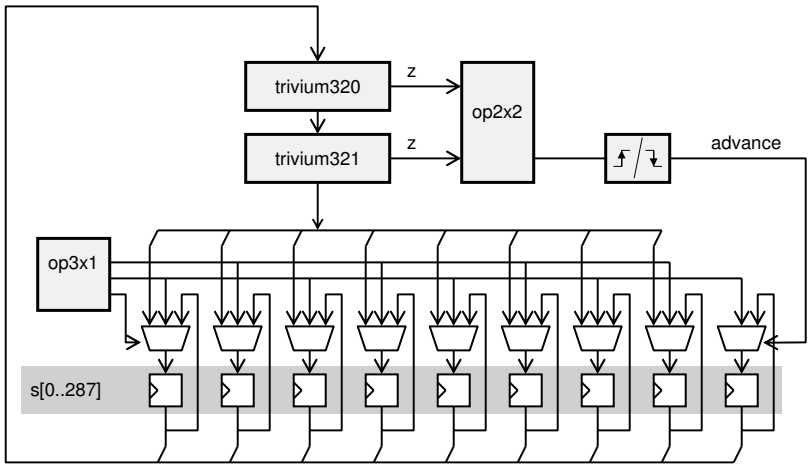
Ex3: Encapsulation Interface (1)

```
ipblock myarm {
  iptype "armsystem";
  ipparm "exec=trivium";
}
ipblock armsfu1(out d1, d2 : ns(32);
                in  q1, q2 : ns(32)) {
  iptype "armsfu2x2";
  ipparm "core = myarm";
  ipparm "device = 0";
}
ipblock armsfu2(out d1, d2, d3 : ns(32);
                in  q1         : ns(32)) {
  iptype "armsfu3x1";
  ipparm "core = myarm";
  ipparm "device = 0";
}
dp trivium320(in  si : ns(288);
              out so : ns(288);
              out z  : ns(32)) {
  ..
}
dp trivium321 : trivium320
```

GEZEL System Topology



Ex3: Encapsulation Interface (2)



Ex3: Driver program in C

```
#include "armsfu.h"
int main() {
  int z1, z2, i;
  unsigned int stream[512];

  int key1 = 0x80;
  int key2 = 0xe0000000;
  OP3x1_1(z1,key1, 0, 1);
  OP3x1_1(z1,  f0, 0, 2);
  OP3x1_1(z1,  f0, 0, 3);
  OP3x1_1(z1,  f0, 0, 4);
  OP3x1_1(z1,key2, 0, 5);
  OP3x1_1(z1,  f0, 0, 0);
  for (i=0; i<9; i++) {
    OP2x2_1(z1, z2, 1, 0);
    OP2x2_1(z1, z2, 0, 0);
   }

  for (i=0; i<128; i++) {
    OP2x2_1(z1, z2, 1, 0);
    stream[4*i]   = z1;
    stream[4*i+1] = z2;
    OP2x2_1(z1, z2, 0, 0);
    stream[4*i+2] = z1;
    stream[4*i+3] = z2;
  }
  return 0;
```

Key schedule

Key stream

Further Reading and References

- **Methodology**
- D. Hwang, P. Schaumont, K. Tiri, I. Verbauwhede, "Securing Embedded Systems," IEEE Security and Privacy Magazine, March-April 2006.
- D. Hwang, P. Schaumont, S. Yang, I. Verbauwhede, "Multi-level Design Validation in a Secure Embedded System," Proceedings of the 2005 High Level Design and Validation Workshop, November 2005.
- A. Jerraya, W. Wolf, "Hardware/Software Interface Codesign for Embedded System Design," IEEE Computer, February 2005.
- E. Lee. "Embedded Software." Advances in Computers (M. Zelkowitz, editor), Vol. 56, Academic Press, London, 2002.
- G. de Micheli, R. Ernst, W. Wolf. "Readings in Hardware/Software Codesign." The Morgan Kaufmann Systems On Silicon Series, Elsevier, Norwell, MA, 2001.
- N. Potlapally, S. Ravi, A. Raghunathan, and N. Jha. "Analyzing the energy consumption of security protocols." 2003 International Symposium on Low Power Electronics and Design (ISLPED 2003), 30-35, 2003.
- A. Ravi, A.Raghunathan, N. Potlapally,and M. Sankaradass. "System design methodologies for a wireless security processing platform." 39th Design Automation Conference, 777-782, 2002.
- C. Rowen, "Engineering the Complex SoC, Fast, Flexible Design with Configurable Processors," 2004, Prentice Hall Modern Semiconductor Series.
- K. Sakiyama, L. Batina, P. Schaumont, and I. Verbauwhede, "HW/SW Co-design for TA/SPA-resistant Public-Key Cryptosystems," In ECRYPT Workshop, CRASH - CRyptographic Advances in Secure Hardware, 8 pages, 2005.
- P. Schaumont, I. Verbauwhede, "Domain-specific co-design for embedded security," IEEE Computer, April 2003.

Further Reading and References

- **Methodology**
- P. Schaumont, D. Hwang, I. Verbauwhede, "Platform-based design for an embedded fingerprint authentication device," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 24(12):1929-1936, December 2005.
- I. Verbauwhede, P. Schaumont, "Skiing the embedded systems mountain," ACM Transactions on Embedded Computing Systems (Special issue on embedded systems education), August 2005.
- **Modeling**
- M. Davio, J. P. Deschamps. and A. Thayse. "Digital Systems with Algorithm Implementation." John Wiley & Sons, New York, 1983.
- A. Donlin, "Transaction Level Modeling: Flows and Use Models," Proc. ISSS/CODES 2004.
- S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli. "Design of Embedded Systems: Formal Models, Validation, and Synthesis." Proceedings of the IEEE, 85(3):366—390, March 1997.
- F. Maraninchi, J. Cornet, L. Maillet-Contoz, "Definition of Transactional abstraction levels needed for a precise architecture evaluation in the System on Chip's Design Flow," presented at 12th Synchronous Workshop, 2005, online.
- W. Qin and S. Malik. "Flexible and formal modeling of microprocessors with application to retargetable simulation." Design, Automation and Test in Europe 2003, 556-561, 2003.
- A. Sangiovanni-Vincentelli, and G. Martin. "Platform-based Design and Software Design Methodology for Embedded Systems." IEEE Design and Test Magazine, 23-33, November-December 2001.
- P. Schaumont, D. Ching, I. Verbauwhede, "An Interactive Codesign Environment for Domain-specific Coprocessors," ACM Transactions on Design Automation of Electronic Systems, January, 2006.

Further Reading and References

- **Modeling**
- P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, and G. Essink. "Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach." 2004 International Conference on Hardware/Software codesign and system synthesis (CODES+ISSS), 206-217, 2004.
- **Coprocessor Architecture**
- F. Barat, and R. Lauwereins. "Reconfigurable Instruction Set Processors: A Survey." IEEE International Workshop on Rapid System Prototyping, 168-173, 2000.
- H. Chan, P. Schaumont, I. Verbauwhede, "Process Isolation for Reconfigurable Hardware," Proc. of the 2006 Engineering of Reconfigurable Systems and Algorithms (ERSA), June 2006.
- H. Eberle, S. Shantz, B. Gupta, N. Gura, L. Rarick, L. Spracklen, "Accelerating next-generation public-key cryptosystems on general-purpose CPUs," IEEE Micro March-April 2005.
- J. Großschädl, P. Ienne, L. Pozzi, S. Tillich, and A. K. Verma. Combining Algorithm Exploration with Instruction Set Design: A Case Study in Elliptic Curve Cryptography. Proc. of the 9th Conference on Design, Automation and Test in Europe (DATE 2006), Munich, Germany, March 6-10, 2006.
- D. Talla, C. Hung, R. Talluri, F. Brill, D. Smith, D. Brier, B. Xiong, and D. Huynh. "Anatomy of a Portable Digital Mediaprocessor." IEEE Micro, 24(2):32-39, March/April 2004.
- S. Yang, P. Schaumont, and I. Verbauwhede, "Microcoded Coprocessor for Embedded Secure Biometric Authentication Systems," IEEE/ACM/IFIP International Conference on Hardware - Software Codesign and System Synthesis(CODES+ISSS'05), Sept. 2005.
- **Coprocessor-software Interfacing Issues**
- A.F. Harvet, "DMA Fundamentals on various PC Platforms," Application Note 011, National Instruments, April 1991, online.

Further Reading and References

- **Coprocessor-software Interfacing Issues**
- Y. Matsuoka, P. Schaumont, K. Tiri, and I. Verbauwhede, "Java cryptography on KVM and its performance and security optimization using HW/SW co-design techniques," Proc. Int. Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2004), pp. 303-311, September 2004.
- P. Schaumont, K. Sakiyama, A. Hodjat, I. Verbauwhede, "Embedded software integration for coarse-grain reconfigurable architectures," 2004 Reconfigurable Architectures Workshop (RAW 2004), April 2004
- **Components**
- Atmel Semiconductor Datasheet, "AT91 ARM Thumb-based Microcontrollers," April 2006, online.
- ARM Ltd, "AMBA Specification v. 2.0," May 1999, online.
- ARM Ltd. "Amba Axi Protocol v1.0." ARM IHI 0022B; <http://www.arm.com/ products/solutions/axi_spec.html>.
- Tensilica, "Xtensa LX Microprocessor," Overview Handbook, online. <http://www.tensilica.com>