Hardware implementation of SP module with PAX cryptoprocessor

YU-YUAN CHEN and RUBY B. LEE

PRINCTON UNIVERSITY TECHNICAL REPORT, APRIL 2008

Abstract:

This report describes an implementation of the Secret Protecting (SP) architecture features in an SP-module in VHDL. It can be integrated with any processor core. In this report, we integrate with the PAX cryptoprocessor designed at Princeton University.

1. SP module implementation

Overview

This is an implementation of the Secret Protecting (SP) [1][6] module that can be added to any processor. SP is a small set of architectural features that can be added to a processor, System-on-Chip (SOC) or multicore chip, to provide hardware-anchored protection of sensitive data, together with a Trusted Software Module (TSM). This implementation includes Authority-mode SP [1] and also User-mode SP [6].

In this report the implementation of SP is added to the base ISA of the PAX cryptoprocessor [3][4][5] designed at Princeton University. A detailed diagram of the SP hardware [2] is given in the SP module in Figure 1. For more information about the PAX cryptoprocessor and SecureCore project that incorporates SP architecture, please refer to [7] and [8].

In this implementation, only Level 1 split caches (L1 Instruction cache and L1 Data cache) are implemented (see Figure 2), mainly for limited space reasons for VHDL to FPGA implementations. Hence, two encryption/hashing engines (one for instruction and one for data) are preferred due to possible contentions of using the engine between CIC (code integrity checking) and secure_load / secure_store instructions in the pipelined implementation. In a microprocessor, a Level 2 unified cache is typically also present on-chip, hence only one encryption/hashing engine would be required at the L2 cache to (off-chip) external memory interface. Since Level 3 caches may also be present on-chip, the SP module is added to the last level of on-chip cache, where a cache-miss would result in having to go off-chip.

The functions of the signals in Figure 1 are explained in Table 1. The `_s` in the signal names signifies a VHDL signal.

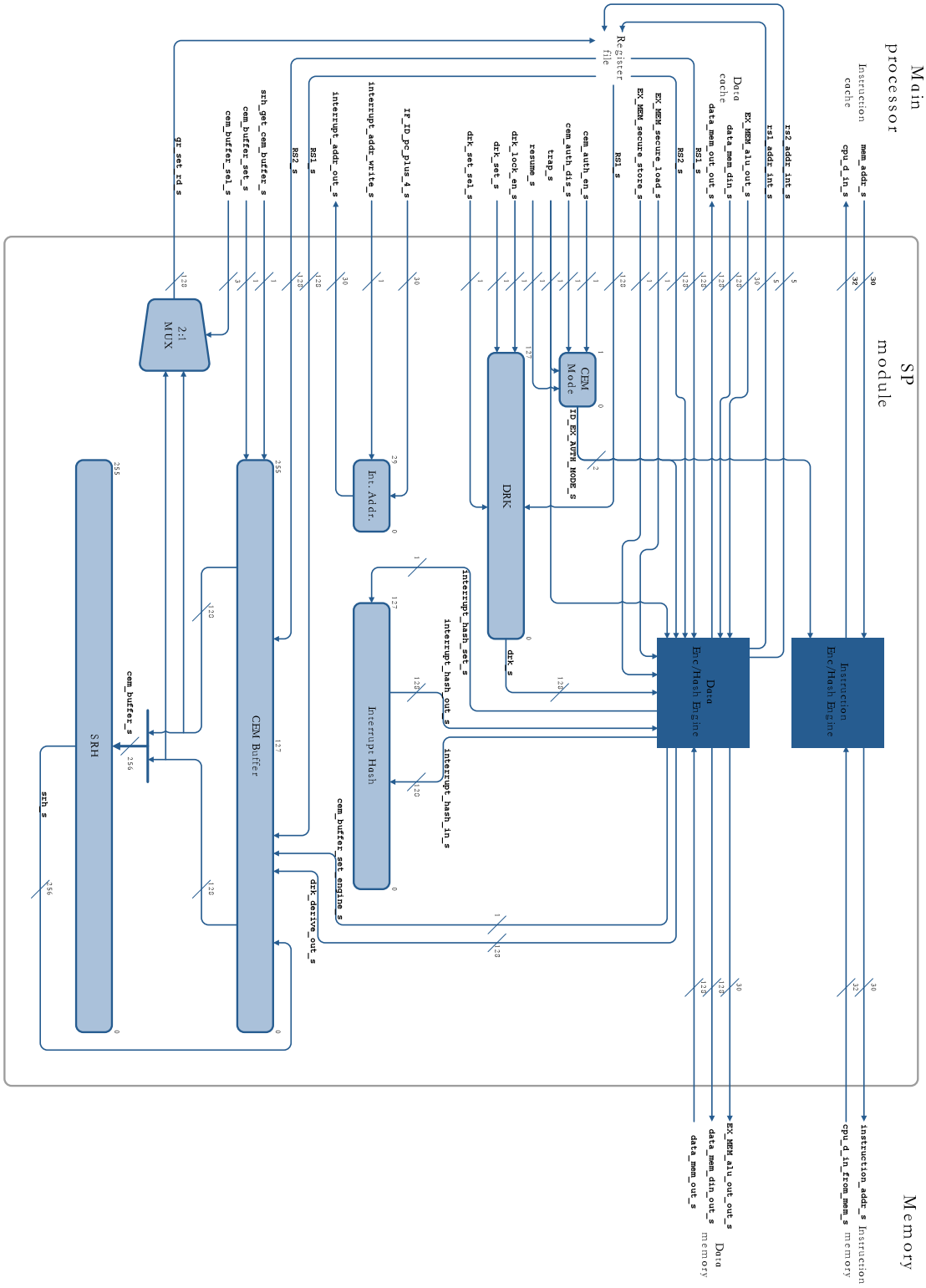


Figure 1: SP module

Table 1: function descriptions of the signals in SP.

Signal name	Function
mem_addr_s	Instruction address from I-cache
instruction_addr_s	Instruction address to instruction memory
cpu_din_from_mem_s	Instruction from instruction memory
cpu_d_in_d	Instruction to I-cache
d_cache_mem_addr_s	Data address from D-cache
d_cache_mem_addr_out_s	Data address to data memory
data_to_mem_s	Data from D-cache (`store`)
data_to_mem_out_s	Data to data memory (`store`)
data_mem_out_s	Data from data memory (`load`)
data_mem_out_out_s	Data to D-cache (`load`)
rs1_addr_int_s	Register index to read the register values upon a software interrupt
rs2_addr_int_s	
trap_s	Software interrupt
resume_s	Resume from software interrupt
interrupt_addr_write_s	Write the return address into `interrupt address register` upon a software interrupt
interrupt_addr_out_s	The value of return address in the `interrupt address register`
interrupt_hash_set_s	Set the value of interrupt hash register
interrupt_hash_out_s	The value of the interrupt hash register
interrupt_hash_in_s	The hash value calculated from enc/hash engine to be set into the interrupt hash register upon a software interrupt
RS1_s	Register values from register file (used for `trap`, `drk.set` and `gr.get`)
RS2_s	
d_cache_secure_load_s	Signal for `secure_load` from D-cache
d_cache_secure_store_s	Signal for `secure_store` from D-cache
cem_auth_en_s	Signifies enter active authority CEM mode
cem_auth_dis_s	Signifies exit active authority CEM mode
drk_lock_en_s	Lock DRK register
drk_set_s	Set the value of DRK register
drk_set_sel_s	Select which part of the DRK register to be set
IF_ID_pc_plus_4_s	The value of `pc+4` in the `IF-ID` stage pipeline register

srh_get_cem_buffer_s	srh.get to set CEM buffer register
cem_buffer_set_s	gr.get to set CEM buffer register
cem_buffer_sel_s	Select which part of the cem buffer to be set (gr.get) and retrieved (gr.set)
gr_set_rd_s	The value to be set into general register of gr.set
ID_EX_AUTH_MODES_S	Value of the CEM mode register
drk_s	Value of the DRK register
cem_buffer_set_engine_s	drk.derive to set the CEM buffer register
drk_derive_out_s	The value of the derived key of drk.derive
cem_buffer_s	The value of the CEM buffer register
srh_s	The value of the SRH register

The signals between SP and the processor are illustrated in Figure 2.

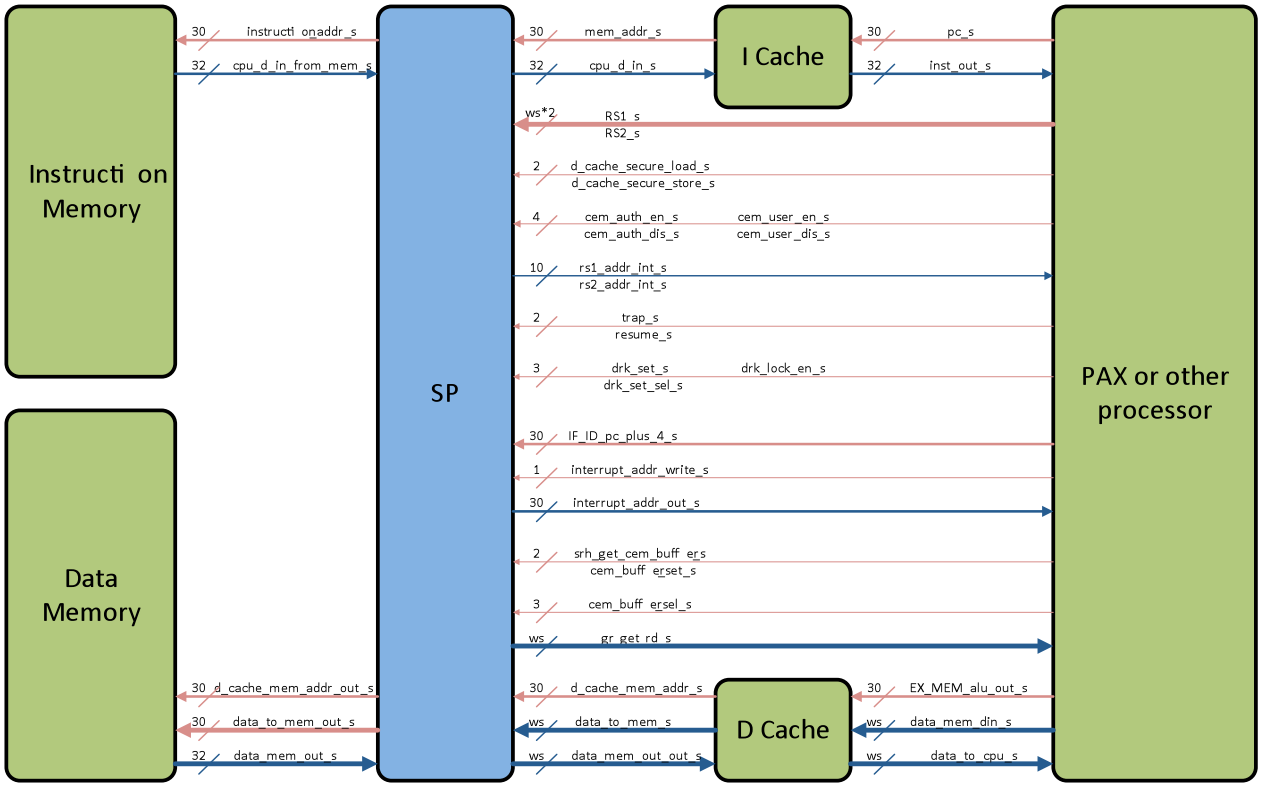


Figure 2: signals between SP and the processor.

SP instructions encoding

SP introduces 18 new instructions, 11 for authority-mode and 7 for user-mode SP. Table 2 shows the encoding for PAX specifically; other encoding can be used to a different processor.

Table 2: SP instructions encoding (the functions of user-mode are currently not implemented).

SP mode	Instruction Class	Mnemonic	Opcode	Subop	Notes
Authority mode	Initialize	drk.set.sel. Rs1, Rs2	010100	000000	sel = 0
				000001	sel = 1
		drk.lock		000010	
	Master Root Secres / CEM Register Access	drk.derive Rs1, Rs2	010101	N/A	
		srh.get	011100	000001	
		srh.set		000010	
		gr.get.sel Rs1, Rs2	011110	000000	sel = 000
				000001	sel = 001
				000010	sel = 010
				000011	sel = 011
				000100	sel = 100
				000101	sel = 101
				000110	sel = 110
				000111	sel = 111
		gr.set.sel Rd		001000	sel = 000
				001001	sel = 001
				001010	sel = 010
				001011	sel = 011
				001100	sel = 100
				001101	sel = 101
				001110	sel = 110
				001111	sel = 111
	CEM	begin_cem.a	000010	000001	
		end_cem.a		000010	
Shared	Secure Memory	secure_load Rs, Rs, imm	010001		
		secure_store Rd, Rs, imm	011001		
User mode	CEM	begin_cem.u	000010	000100	
		end_cem.u		001000	
	Master	umk.get.sel	010110		

	Secrets				
	Initialize	dmk.set	010100	001000	
		dmk.lock		010000	
		umk.set		100000	
	Virtualization	cem_save.u	001000		
		cem_restore.u	001001		

TSM code/data alignment

Since the CIC encryption/hashing engine is placed between external memory and the leve-1 cache, only a cache miss will trigger the engine to check the integrity of secure code/data. If a secure code/data has been brought into the cache before CEM mode is active, that particular code/data will not be checked by the engine. Two possible solutions can solve this issue. The first one is to flush the cache line after CEM becomes active and bring back the cache line into the cache again, so that it is checked by the engine. However, this approach will require non-secure code/data that co-exists with secure code/data in the same cache line to be included in the calculation of hash, which is unnecessary. The second approach is to force alignment of secure code/data to the line size of instruction cache, so that the execution of first secure code/data will automatically trigger a cache miss and bring in a cache line of secure code/data. For the current implementation, we force flush for secure data while force alignment for secure code.

A requirement for TSM code resulting from the forced alignment is that the compiler has to make sure that `begin_cem` is always placed at the last word of a cache line, so that the following TSM code will miss in the cache and automatically be checked by CIC. Whether or not the TSM code is called as a function or inserted inline with the application code does not compromise the security of TSM code as long as the above requirement is met.

2. Integration of PAX and SP

PAX top level diagram

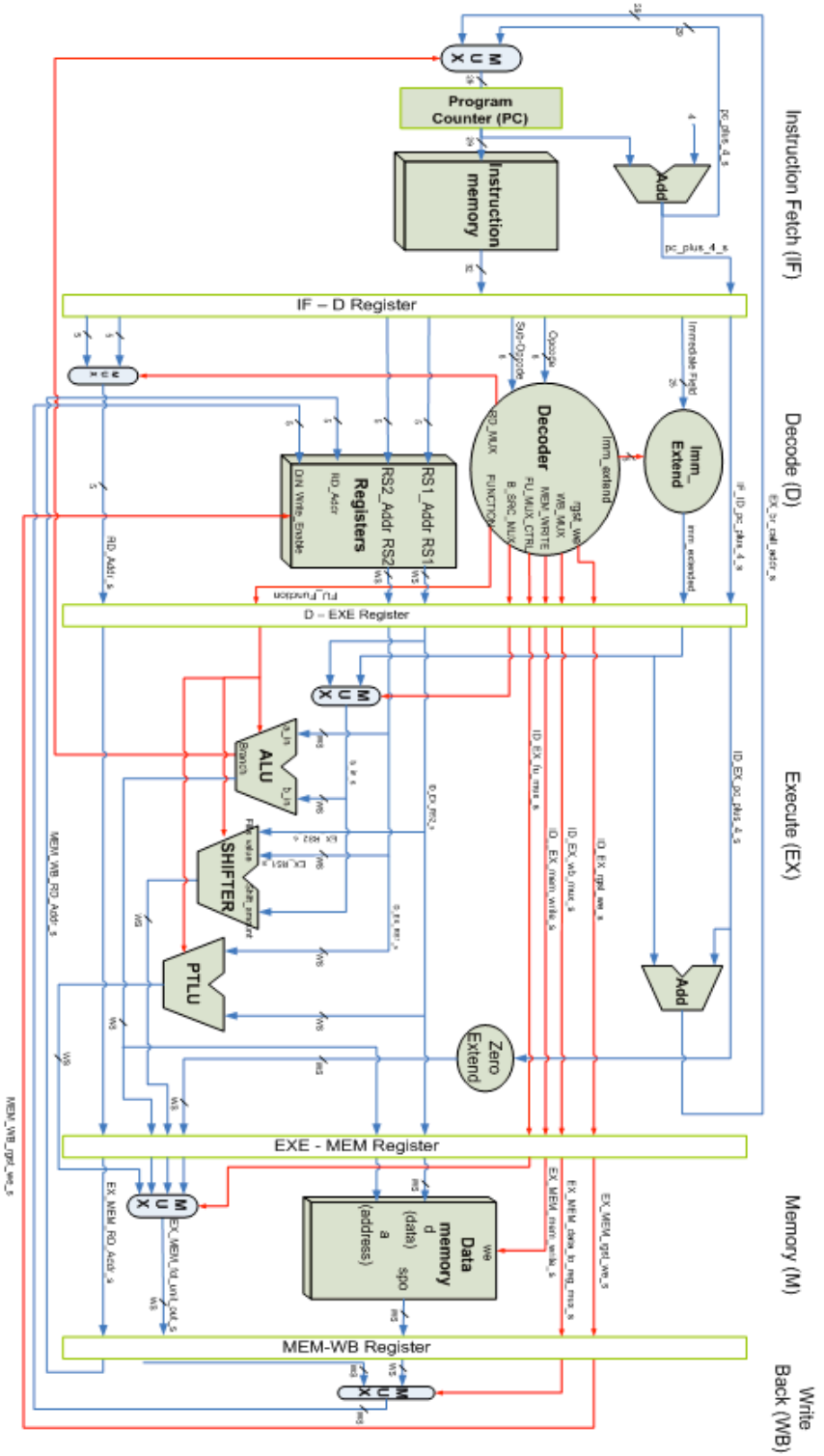


Figure 3: pipeline implementation of PAX.

SP encoding in PAX

SP instructions are encoded into empty slots of appropriate categories of PAX instruction sets. Table 3 shows the encodings for SP instructions with the PAX ISA. Similar encodings of SP instructions can be done for other processors.

Table 3: SP-PAX instruction encoding table.

Instruction	Opcode		
CALL	00	00	01
Begin CEM	00	00	10
End CEM			
LDZ	00	01	00
LDK	00	01	01
RET	00	01	10
TRAP	00	01	11
RESUME			
CEM save	00	10	00
CEM restore			
Secure load	01	00	01
LW	01	00	10
LD8	01	00	11
DRK set	01	01	00
DRK lock			
DMK set			
DMK lock			
UMK set			
DRK derive	01	01	01
Secure store	01	10	01
SW	01	10	10
SW8	01	10	11
SRH get	01	11	00
SRH set			
GR get	01	11	10
GR set			
SUBi	10	00	01

Instruction	Opcode		
ORi	10	00	11
XORi	10	01	00
SLLi	10	01	01
SRAi	10	01	10
ShRP	10	10	10
BGU	10	11	00
BGEU	10	11	01
BG	10	11	10
BGE	10	11	11
AND	11	00	00
OR			
XOR			
NOT			
ADDw			
SUBw			
PERM.1	11	00	10
BEQ	11	01	10
BNE	11	01	11
Bfmul.lo	11	00	01
Bfmul.hi			
Shuffle.lo	11	10	10
Shuffle.hi			
Rev			
ptw	11	10	11
ptr.x.ctr	11	11	00
ptr.s.ctr			
ptr.o			
pti	11	11	01

ADDi	10	00	00
ANDi	10	00	10

LD16	11	11	10
ST16	11	11	11

PAX design file changes

Several parts of PAX have to be modified to incorporate the introduction of SP components. The changes of design files of PAX are listed in Table 4.

Table 4: modifications to PAX design files.

File	Added signals	Function
decoder.vhd	INTERRUPT_ADDR_WRITE	Write-enable for interrupt address register
	TRAP	Specify a `trap` instruction
	RESUME	Specify a `resume` instruction
	SECURE_LOAD	Specify a `secure_load` instruction
	SECURE_STORE	Specify a `secure_store` instruction
	ENGINE_FUNC	Specify the enc/hash engine function
	SRH_SET	Specify a `srh_set` instruction
	SRH_GET_CEM_BUFFER	Specify a `srh_get` instruction
	GR_SEL	`sel` signal used for `gr.get` and `gr.set`
	CEM_BUFFER_SET	enable `write` signal for CEM buffer
	DRK_SET	Specify a `drk_set` instruction
	DRK_SET_SEL	`sel` signal used for `drk_set`
	DRK_LOCK_EN	Specify a `drk_lock` instruction
	CEM_USER_EN	Specify a `begin_cem.u` instruction
	CEM_AUTH_EN	Specify a `begin_cem.a` instruction
	CEM_USER_DIS	Specify a `end_cem.u` instruction
	CEM_AUTH_DIS	Specify a `end_cem.a` instruction
EX_MEM_reg.vhd	d_cache_stall	Pipeline stall signal from data cache
	ID_EX_engine_func	`ENGINE_FUNC` in pipeline stage `ID_EX`
	ID_EX_secure_load	`secure_load` in pipeline stage `ID_EX`
	ID_EX_secure_store	`secure_store` in pipeline stage `ID_EX`
	EX_MEM_ENGINE_FUNC	`ENGINE_FUNC` in pipeline stage `EX_MEM`
	EX_MEM_SECURE_LOAD	`secure_load` in pipeline stage `EX_MEM`
	EX_MEM_SECURE_STORE	`secure_store` in pipeline stage `EX_MEM`
EX_Mux.vhd	gr_set_out	Extra signal for writing into registers
ID_EX_reg.vhd	d_cache_stall	Pipeline stall signal from data cache
	engine_func	`ENGINE_FUNC` from decoder

	secure_load	SECURE_LOAD from decoder
	secure_store	SECURE_STORE from decoder
IF_ID_reg.vhd	d_cache_stall	Pipeline stall signal from data cache
PAX_pack.vhd	constant FROM_GR_SET	Add an extra control signal to EX_Mux
pc_next.vhd	d_cache_stall	Pipeline stall signal from data cache
	cache_busy	Signal to indicate instruction fetch stall due to cache access time

SP components design files

This section describes the design files added by introducing SP components into PAX.

Table 5: SP design files.

File	Note
CEM_buffer_mux.vhd	multiplexer to select which part of CEM buffer for gr.set.sel Rd
CEM_buffer_reg.vhd	implements CEM buffer register
CEM_mode_reg.vhd	implements CEM mode register
DRK_reg.vhd	implements DRK register
enc_hash_engine.vhd	implements enc/hash engine for data cache
i_cache_engine.vhd	implements enc/hash engine for instruction cache
Interrupt_addr_reg.vhd	implements interrupt address register
Interrupt_hash_reg.vhd	implements interrupt hash register
SRH_reg.vhd	implements SRH register

Level-1 cache design files

This section describes the design files added by introducing level-1 instruction and data cache into PAX. The first two files that deal with bit-vector arithmetic are used for the internal data format in the caches. Bit-vectors and std_logic vectors in VHDL are essentially the same except simulation purposes.

Table 6: level-1 cache design files.

File	Note
bv_arithmetic-body.vhd	Function body of *bit-vector* arithmetic
bv_arithmetic.vhd	Function declaration of *bit-vector* arithmetic
cache_types.vhd	Define cache write strategy types (*write-back* or *write-through*)
d_cache-behaviour.vhd	Behavior model of data cache

d_cache.vhd	Entity declaration of data cache
dlx_types-body.vhd	Implements the package body of `dlx_types`
dlx_types.vhd	Defines subtypes of signals of different width
i_cache-behaviour..vhd	Behavior model of instruction cache
i_cache.vhd	Entity declaration of instruction cache
mem_types.vhd	Defines the types of the widths of memory bus

The cache model is a modified version of Peter J. Ashenden [9]. The current setup is outlined in the following table:

Table 7: cache parameters for both instruction and data cache for PAX- 128.

Parameter	Value
Cache size	32 KB
Line size	64 Bytes
Associativity	1 (direct-mapped)
Write strategy	Write through
Hit time	1 cycle
Miss penalty	16 cycles (data cache)
	4 cycles (instruction cache)
Clock cycle	20 ns

3. Simulation

Testing SP functionality

We use the test assembly code given below to test the correct operations of SP components.

Test assembly code

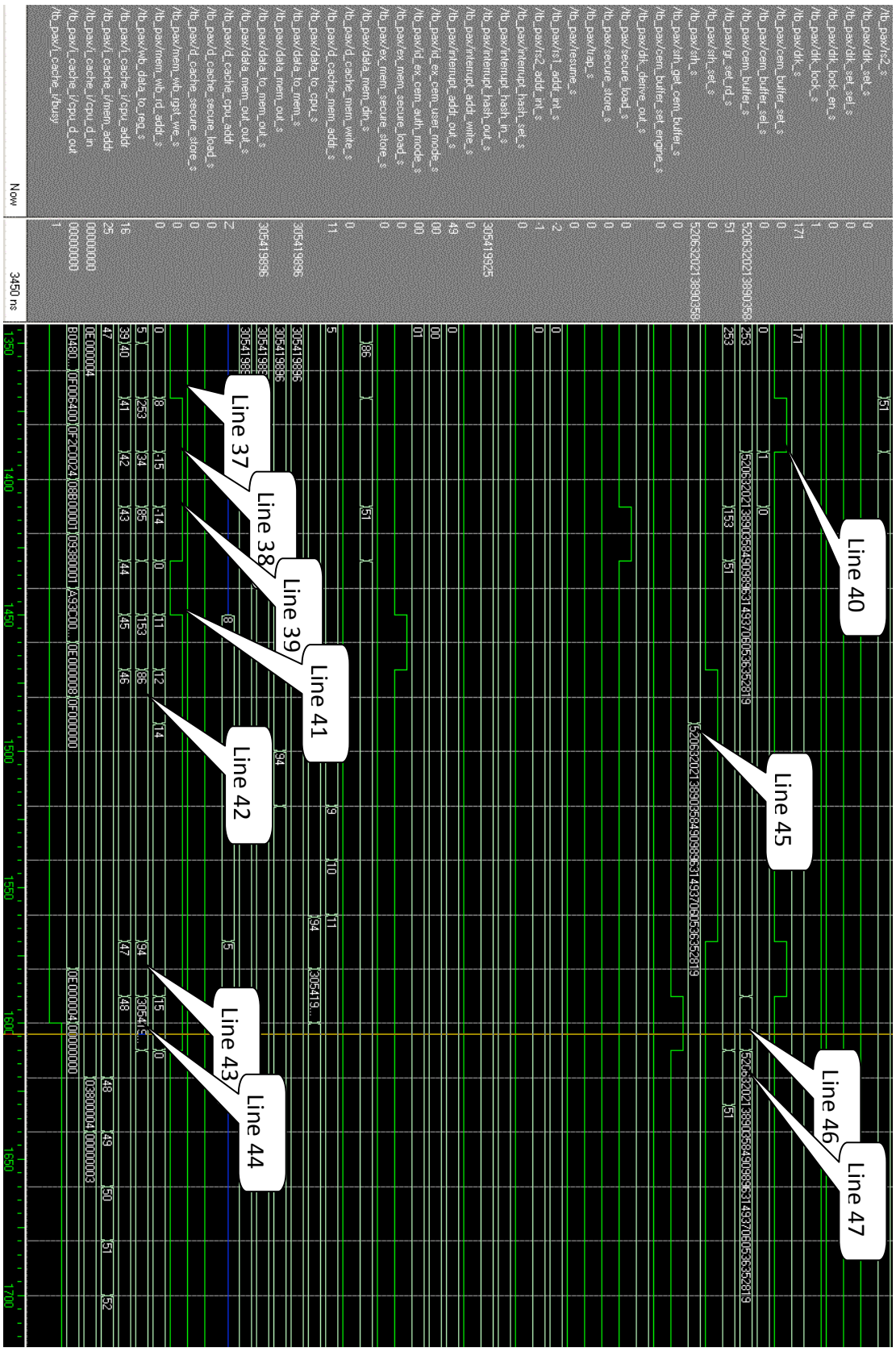
```
pc   Instruction                    Comments
@ put initial constants in registers and memory location for later
0    addi r1, r0, #0xAB             @ r1 = 0xAB (171)
1    addi r2, r0, #0x56             @ r2 = 0x56 (86)
2    loadi.k.1 r8, #0x1234
3    loadi.k.0 r8, #0x5678          @ r8 = 0x12345678
4    store.16 r8, r0, #0x04         @ mem[0x04] = 0x12345678 (305419896)
 @ setting up the DRK and lock the DRK register
5    drk.set.0 r1, r0               @ drk = 0xAB (171)
```

```
6    drk.lock                    @ this is simulating machine bootup
@ start CEM section
7    begin_cem.a
8    Nop
9    call #0x22                  @ call TSM code
 @ end CEM section
10   end_cem.a
 @ some memory accesses to verify the values stored in memory locations
11   load r14, r0, #0x04         @ r14 = 0x12345678 (305419896)
12   load r13, r0, #0x08         @ r13 = 0x5E (94)
13   secure_store r2, r0, #0x08  @ mem[0x08] = 0x5E (94)

@ ================================================
@ Start of TSM code
@ setting up some register values for later
32   addi r3, r0, #0x99          @ r3 = 0x99 (153)
33   addi r4, r0, #0x33          @ r4 = 0x33 (51)
 @ do a secure store to memory to put the encrypted value
34   secure_store r2, r0, #0x08  @ mem[0x08] = 0x5E (94) (0x5E = 0x56 xor 0x08)
35   store.16 r8, r0, #0x05      @ mem[0x05] = 0x12345678 (305419896)
 @ ask for a derived key
36   drk.derive r2, r0           @ cem_buffer = 0xFD (253)
                                 @ drk xor r2 = 0xAB xor 0x56 = 0xFD
37   xor r8, r1, r2              @ r8 = 0xFD (253)
38   addi r9, r0, #0x22          @ r9 = 0x22   (34) (dummy instruction)
39   addi r10, r0, #0x55         @ r10 = 0x55 (85) (dummy instruction)
 @ test the functions of gr.get and gr.set
40   gr.get.0 r3, r4             @ cem_buffer =
                                 0x99000000000000000000000000000033
                                 @  = 52063202138903584909896314937060536352819
41   gr.set.1 r11                @ r11 = 0x99 (153)
 @ do secure load and normal load to the same memory location and expect to get
different values, one decrypted and one encrypted
42   secure_load r12, r0, #0x08  @ r12 = 0x56 (86) (0x56 = 0x5E xor 0x08)
43   load r14, r0, #0x08         @ r14 = 0x5E (94)
44   load r15, r0, #0x05         @ r15 = 0x12345678 (305419896)
@ test the srh.set and srh.get
45   srh.set                     @ srh = 0x99000000000000000000000000000033
```

```
                                    @  = 52063202138903584909896314937060536352819
46   gr.get.0 r0, r0                @ cem_buffer = 0x00
47   srh.get                        @ cem_buffer =
                                    0x990000000000000000000000000000033
                                    @  = 52063202138903584909896314937060536352819
@ test the trap and resume instructions
48   Trap
49   nop x 6
55   drk.set.0 r10, r0              @ trying to set drk = 0x55 (85) but illegal
56   nop x 6
62   Resume
63   nop x 12
75   Ret
@End of TSM code
@=================================================
```

Simulation snapshot



Simulation of AES-128 with new I-cache and D-cache

We also tested our PALMS-group's optimized AES-128 software program [5]. This ran correctly with the new SP module and cache additions, including the initialization of the AES tables and cache-misses in the new I-cache and D-cache (which were not present in the earlier PAX simulations).

In the AES startup phase, there was a 29.5% overhead due to the I-cache misses in fetching of the AES code into the empty I-cache, and the D-cache misses for AES execution. This demonstrates the correct functioning of the new caches, since the previous PAX simulations was equivalent to all cache hits.

In the steady-state AES phase, each round of AES took 2 cycles with no cache misses, as before.

4. Future work

Writing Applications

Writing an application with PAX assembly code without any compiler support would be difficult. Possible solutions include writing a small application with key storage structure using C code and translating the compiled assembly into PAX assembly.

Register spilling

Proper compiler support has to be added to make sure any secure data stored in the registers cannot be spilled out to memory during TSM execution. Otherwise, the security provided by CEM would be broken and secrets potentially leaked out of the processor.

HMAC of secure data

Unlike secure code, secure data cannot put the hash at the end of a cache line in that the hash would include the entire cache line. Possible solutions include storing the hash in the "other half" of the memory address space [6], such that each read from a secure data would require two memory reads, one for the data and another for the hash of the data.

Encryption/Hashing engine

The current implementations of the encryption/hashing engines are simple `XORs` with either the memory addresses for secure code/data or with DRK for DRK derive. This is because we are not interested in an optimal encryption/hashing

design for this project. Future work would include an engine that does AES or some other cipher/s and hash function/s. Note that since PAX does AES a lot faster than most special purpose AES engines, it is possible to use PAX as SP's encryption/hashing engine. It would require the code for HMAC, encryption or decryption to be stored in a particular area of the PAX memory, so that when SP requires encryption/hashing the control would jump to the code that handles them. If PAX were also acting as the main processor running the application in a single-core processor chip, this would disrupt the application's execution and incur overhead to manage the switch between the application's code and encryption/hashing. However, in a multi-core chip, PAX-SP can serve as an on-chip input-output processor that does security processing.

5. References

[1] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security,* 2007, pp. 389-400.

[2] J. Dwoskin and R. Lee, "SP Processor Architecture Reference Manual," *Technical Report CE-L2007-009,* 11/21/2007.

[3] M. Fiskiran and R. B. Lee, "On-chip lookup tables for fast symmetric-key encryption," in ASAP '05: Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors, 2005, pp. 356-363.

[4] Murat Fiskiran and Ruby B. Lee, "PAX: A Datapath-Scalable Minimalist Cryptographic Processor for Mobile Environments," *Embedded Cryptographic Hardware: Design and Security*, Nadia Nedjah and Luiza de Macedo Mourelle, eds., Nova Science, NY, ISBN 1-59454-145-0, September 2004.

[5] Ruby B. Lee, Murat Fiskiran, Michael Wang, Yedidya Hilewitz, Yu-Yuan Chen, "PAX: A Cryptographic Processor with Parallel Table Lookup and Wordsize Scalability," Princeton University Department of Electrical Engineering Technical Report CE-L2007-010, November 2007.

[6] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin and Z. Wang, "Architecture for protecting critical secrets in microprocessors," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on,* 2005, pp. 2-13.

[7] PAX processor project, http://palms.ee.princeton.edu/pax

[8] SecureCore project, http://securecore.princeton.edu/

[9] Peter J. Ashenden, DLX processor. http://ghdl.free.fr/dlx/cache.9.html