国立大学法人
電気通信大学
The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

「Course」 RISC-V Computer System Integration

「 Lecture 08」 Cryptosystem: Integration with SHA, AES, & RSA Modules

Phạm Công Kha
Hoàng Trọng Thức

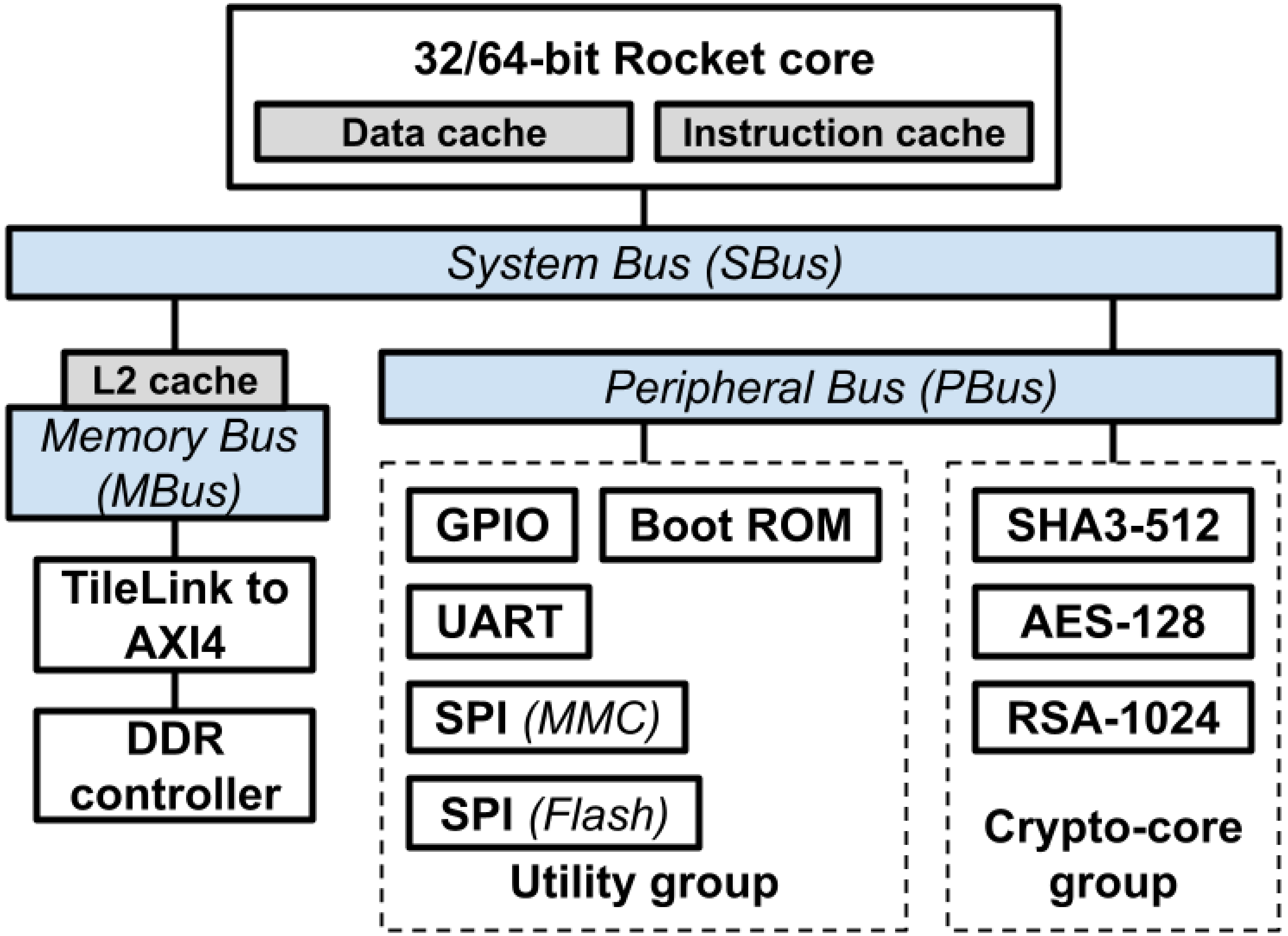
Tháng 9/2023

Outline

Outline

1. Preparation
2. Hash function: SHA3-512
3. Cipher function: AES-128
4. Crypto-key scheme: RSA-1024
5. Practice

1. Preparation (1/5) The wanted system architecture

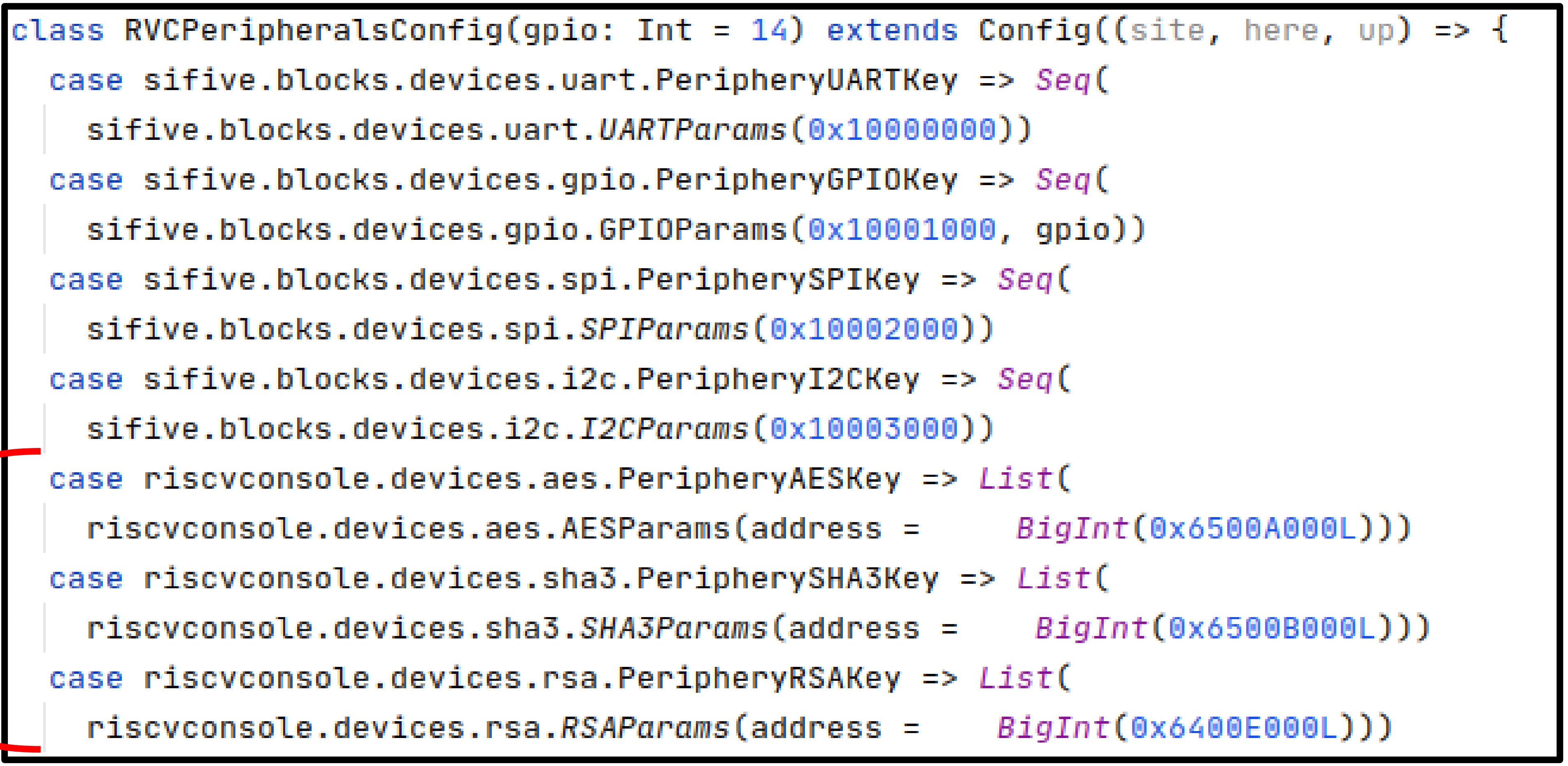


We'll have the completed system looks like this:

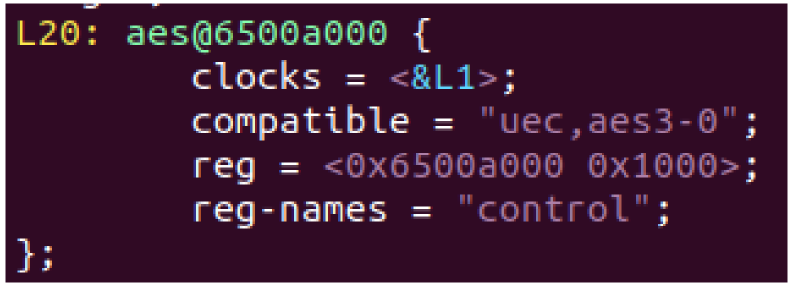
- Single-core 32/64-bit Rocket
- A DDR controller for main memory
- Peripherals include two groups: utility & crypto-cores
- Utility group consists of modules necessary for working such as boot ROM, UART, SPI for MMC, etc.
- Crypto-core group consists of **SHA3** *(512-bit)*, **AES** *(128-bit)*, and **RSA** *(1024-bit)*.

1. Preparation (2/5) The wanted final results

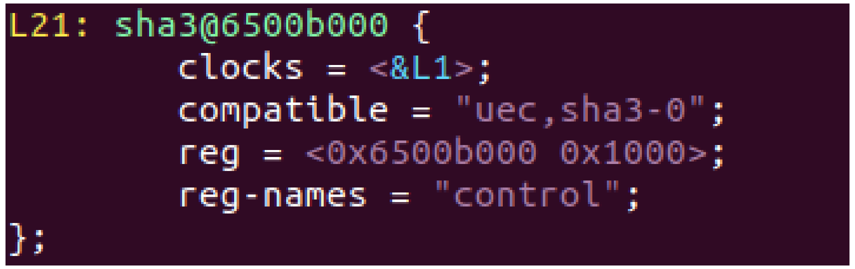
Three crypto-cores of AES, SHA3, and RSA are added to the system:



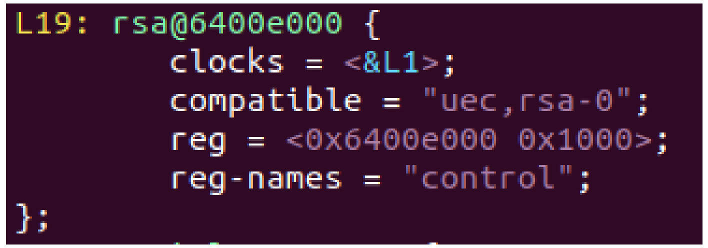
```
class RVCPeripheralsConfig(gpio: Int = 14) extends Config((site, here, up) => {
  case sifive.blocks.devices.uart.PeripheryUARTKey => Seq(
    sifive.blocks.devices.uart.UARTParams(0x10000000))
  case sifive.blocks.devices.gpio.PeripheryGPIOKey => Seq(
    sifive.blocks.devices.gpio.GPIOParams(0x10001000, gpio))
  case sifive.blocks.devices.spi.PeripherySPIKey => Seq(
    sifive.blocks.devices.spi.SPIParams(0x10002000))
  case sifive.blocks.devices.i2c.PeripheryI2CKey => Seq(
    sifive.blocks.devices.i2c.I2CParams(0x10003000))
  case riscvconsole.devices.aes.PeripheryAESKey => List(
    riscvconsole.devices.aes.AESParams(address =     BigInt(0x6500A000L)))
  case riscvconsole.devices.sha3.PeripherySHA3Key => List(
    riscvconsole.devices.sha3.SHA3Params(address =    BigInt(0x6500B000L)))
  case riscvconsole.devices.rsa.PeripheryRSAKey => List(
    riscvconsole.devices.rsa.RSAParams(address =    BigInt(0x6400E000L)))
```



```
L20: aes@6500a000 {
        clocks = <&L1>;
        compatible = "uec,aes3-0";
        reg = <0x6500a000 0x1000>;
        reg-names = "control";
};
```



```
L21: sha3@6500b000 {
        clocks = <&L1>;
        compatible = "uec,sha3-0";
        reg = <0x6500b000 0x1000>;
        reg-names = "control";
};
```



```
L19: rsa@6400e000 {
        clocks = <&L1>;
        compatible = "uec,rsa-0";
        reg = <0x6400e000 0x1000>;
        reg-names = "control";
};
```

1. Preparation (3/5) Get Verilog sources

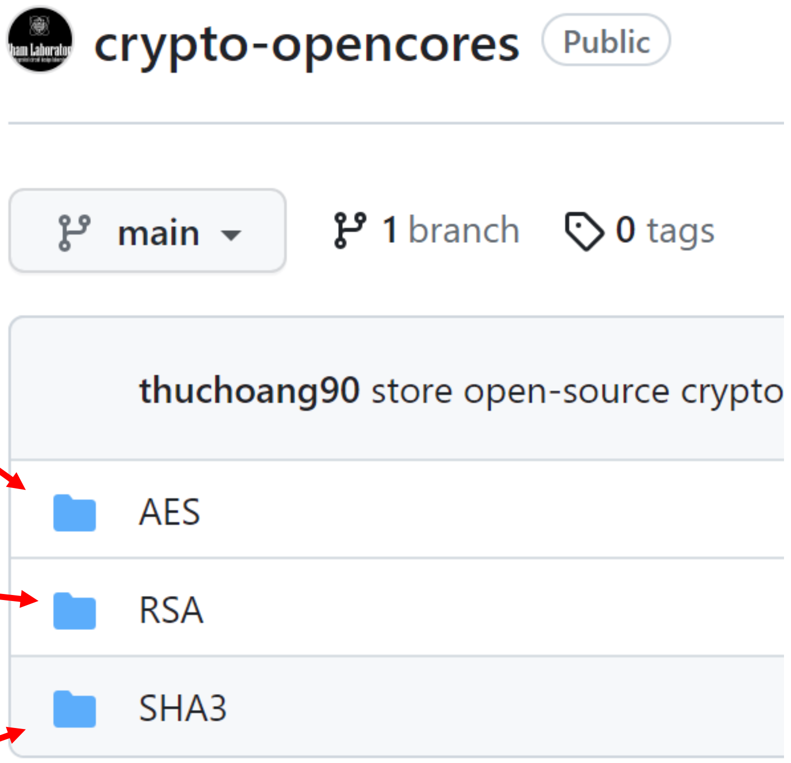
You can get the Verilog sources of the three modules in here:

```
$ git clone https://github.com/uec-hanken/crypto-opencores.git
```

AES *(128-bit)*

RSA *(1024-bit)*

SHA3 *(512-bit)*



crypto-opencores Public

thuchoang90 store open-source crypto

AES

RSA

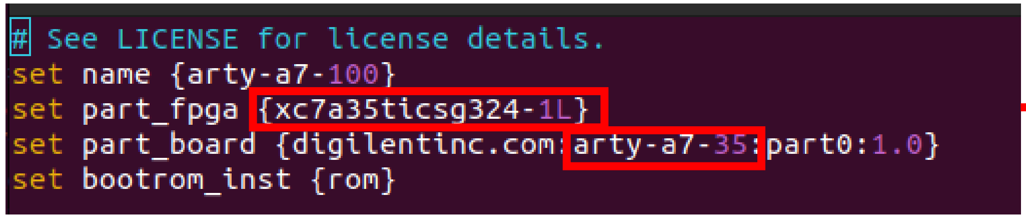
SHA3

1. Preparation (4/5) Use Arty-A7-100T

The *three crypto-cores* are quite **big**, so the **35T** version of **Arty-A7** is not going to be enough. We have to switch back to the **100T** version.

```
$ vi hardware/fpga-shells/xilinx/arty_a7_100/tcl/board.tcl
```

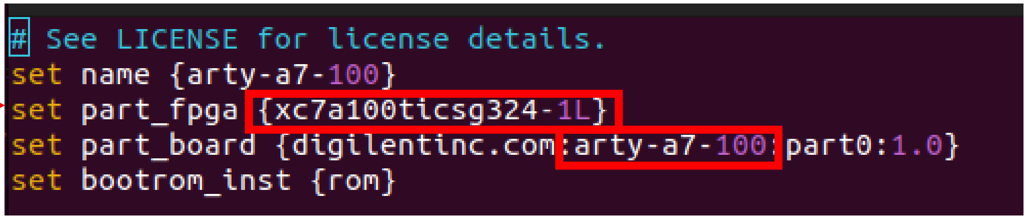
Change from here:



```
# See LICENSE for license details.
set name {arty-a7-100}
set part_fpga {xc7a35ticsg324-1L}
set part_board {digilentinc.com:arty-a7-35:part0:1.0}
set bootrom_inst {rom}
```

(type `i` *to write and* `esc` *to release)*

To here:



```
# See LICENSE for license details.
set name {arty-a7-100}
set part_fpga {xc7a100ticsg324-1L}
set part_board {digilentinc.com:arty-a7-100:part0:1.0}
set bootrom_inst {rom}
```

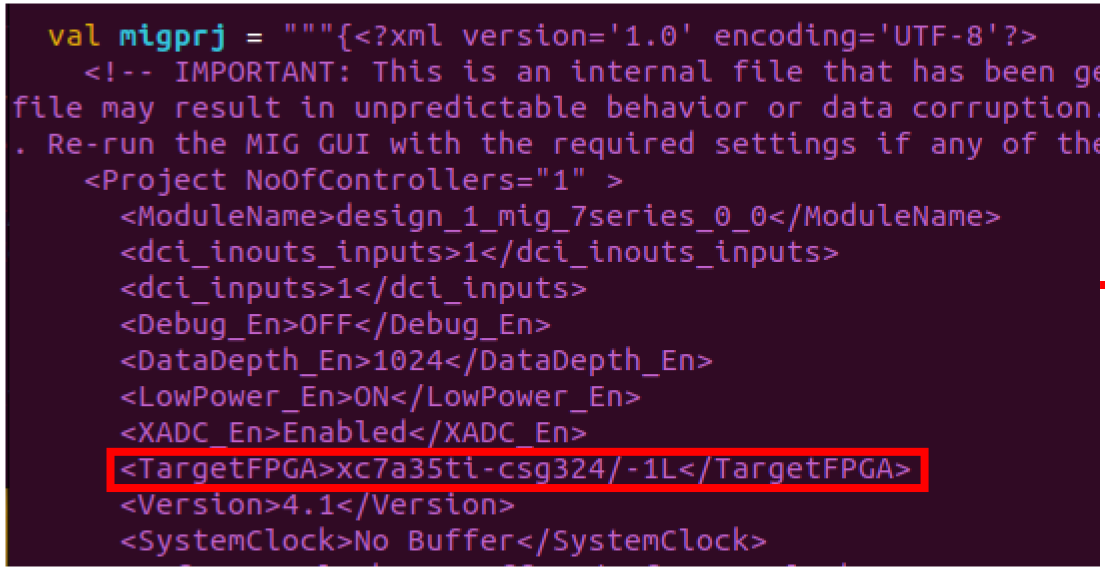
(type `:wq` *to save and exit)*

1. Preparation (5/5) Use Arty-A7-100T

The *three crypto-cores* are quite **big**, so the **35T** version of **Arty-A7** is not going to be enough. We have to switch back to the **100T** version.

```
$ vi hardware/fpga-shells/src/main/scala/ip/xilinx/arty100tmig/arty100tmig.scala
```

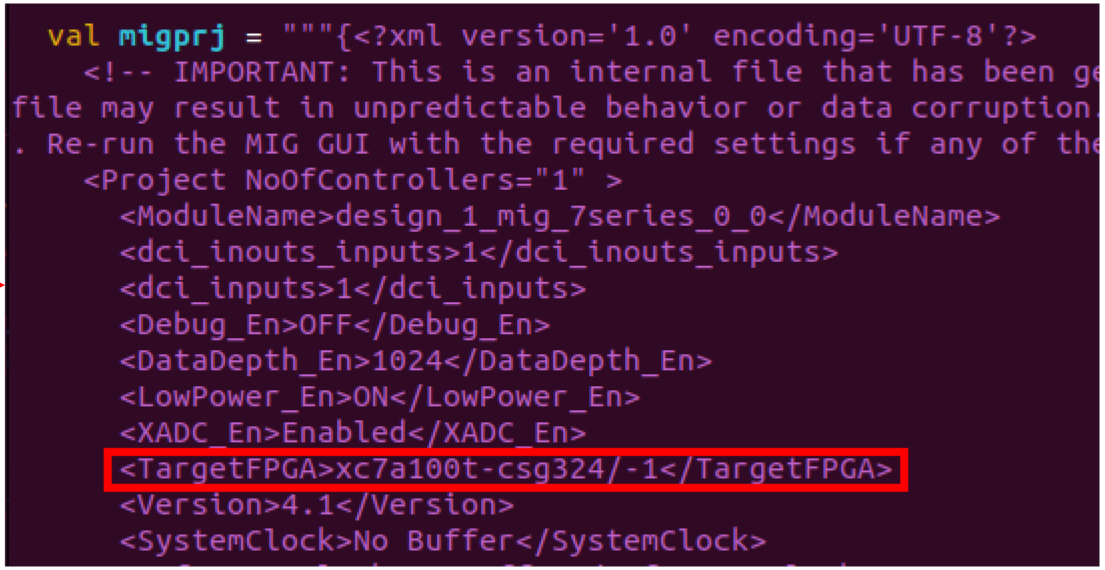
Change from here:



```
  val migprj = """{<?xml version='1.0' encoding='UTF-8'?>
    <!-- IMPORTANT: This is an internal file that has been ge
file may result in unpredictable behavior or data corruption.
. Re-run the MIG GUI with the required settings if any of the
    <Project NoOfControllers="1" >
      <ModuleName>design_1_mig_7series_0_0</ModuleName>
      <dci_inouts_inputs>1</dci_inouts_inputs>
      <dci_inputs>1</dci_inputs>
      <Debug_En>OFF</Debug_En>
      <DataDepth_En>1024</DataDepth_En>
      <LowPower_En>ON</LowPower_En>
      <XADC_En>Enabled</XADC_En>
      <TargetFPGA>xc7a35ti-csg324/-1L</TargetFPGA>
      <Version>4.1</Version>
      <SystemClock>No Buffer</SystemClock>
```

(type `i` to write and `esc` to release)

To here:



```
  val migprj = """{<?xml version='1.0' encoding='UTF-8'?>
    <!-- IMPORTANT: This is an internal file that has been ge
file may result in unpredictable behavior or data corruption.
. Re-run the MIG GUI with the required settings if any of the
    <Project NoOfControllers="1" >
      <ModuleName>design_1_mig_7series_0_0</ModuleName>
      <dci_inouts_inputs>1</dci_inouts_inputs>
      <dci_inputs>1</dci_inputs>
      <Debug_En>OFF</Debug_En>
      <DataDepth_En>1024</DataDepth_En>
      <LowPower_En>ON</LowPower_En>
      <XADC_En>Enabled</XADC_En>
      <TargetFPGA>xc7a100t-csg324/-1</TargetFPGA>
      <Version>4.1</Version>
      <SystemClock>No Buffer</SystemClock>
```

(type `:wq` to save and exit)

国立大学法人 電気通信大学 The University of Electro-Communications

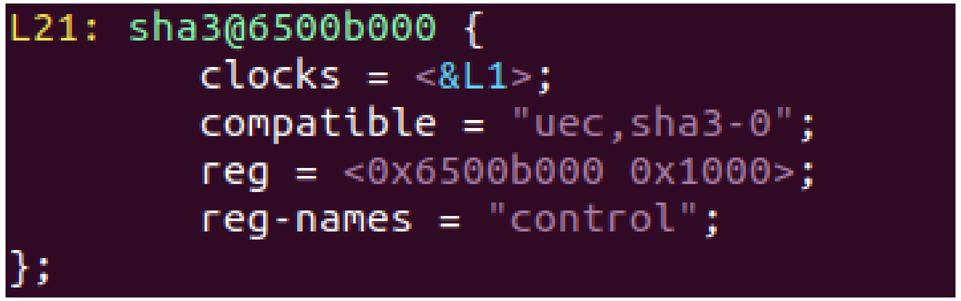
Pham Laboratory
Integrated circuit design laboratory

Outline

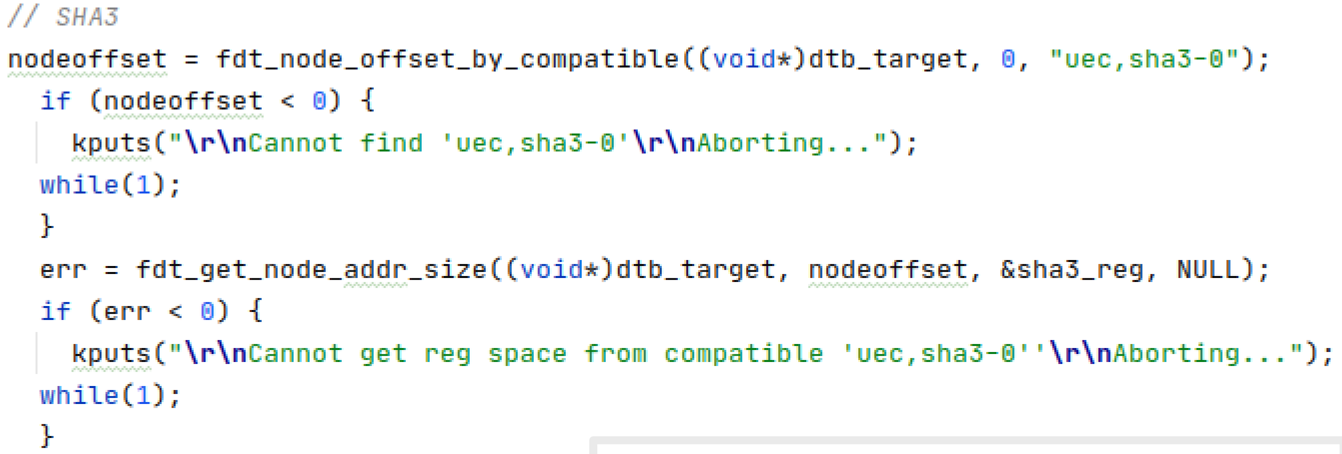
1. Preparation
2. Hash function: SHA3-512
3. Cipher function: AES-128
4. Crypto-key scheme: RSA-1024
5. Practice

2. Hash function: SHA3 (1/10) The wanted result

The **SHA3** module is added to the system.

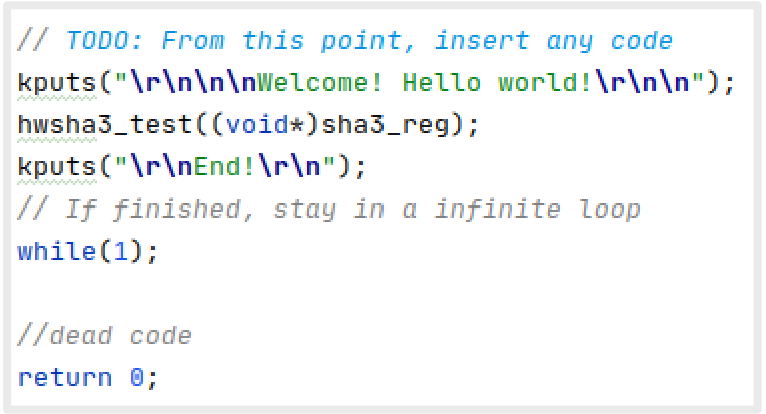


```
L21: sha3@6500b000 {
        clocks = <&L1>;
        compatible = "uec,sha3-0";
        reg = <0x6500b000 0x1000>;
        reg-names = "control";
};
```



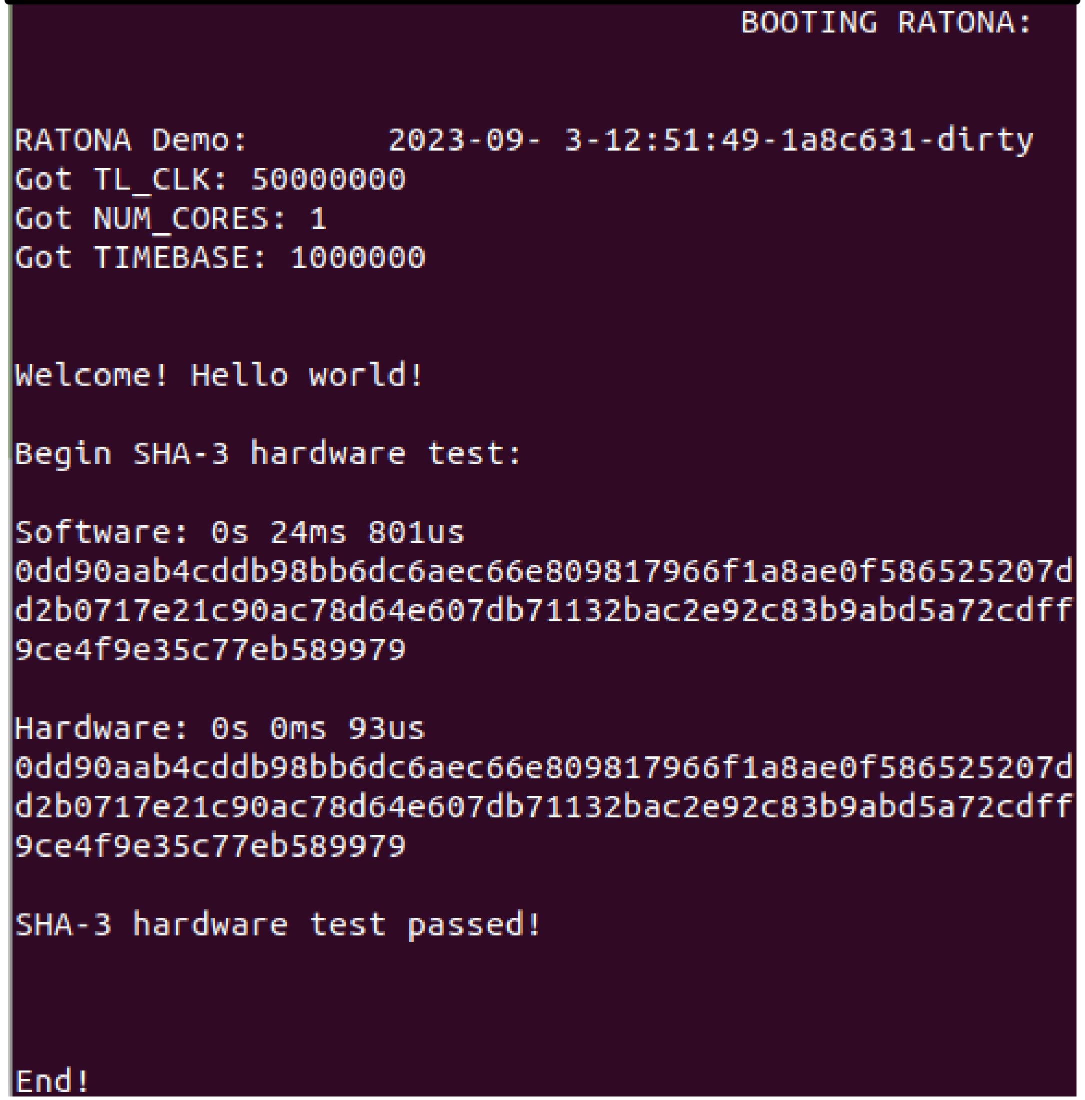
```
// SHA3
nodeoffset = fdt_node_offset_by_compatible((void*)dtb_target, 0, "uec,sha3-0");
  if (nodeoffset < 0) {
    kputs("\r\nCannot find 'uec,sha3-0'\r\nAborting...");
  while(1);
  }
  err = fdt_get_node_addr_size((void*)dtb_target, nodeoffset, &sha3_reg, NULL);
  if (err < 0) {
    kputs("\r\nCannot get reg space from compatible 'uec,sha3-0''\r\nAborting...");
  while(1);
  }
```

Get the module pointer in software and run the hardware test



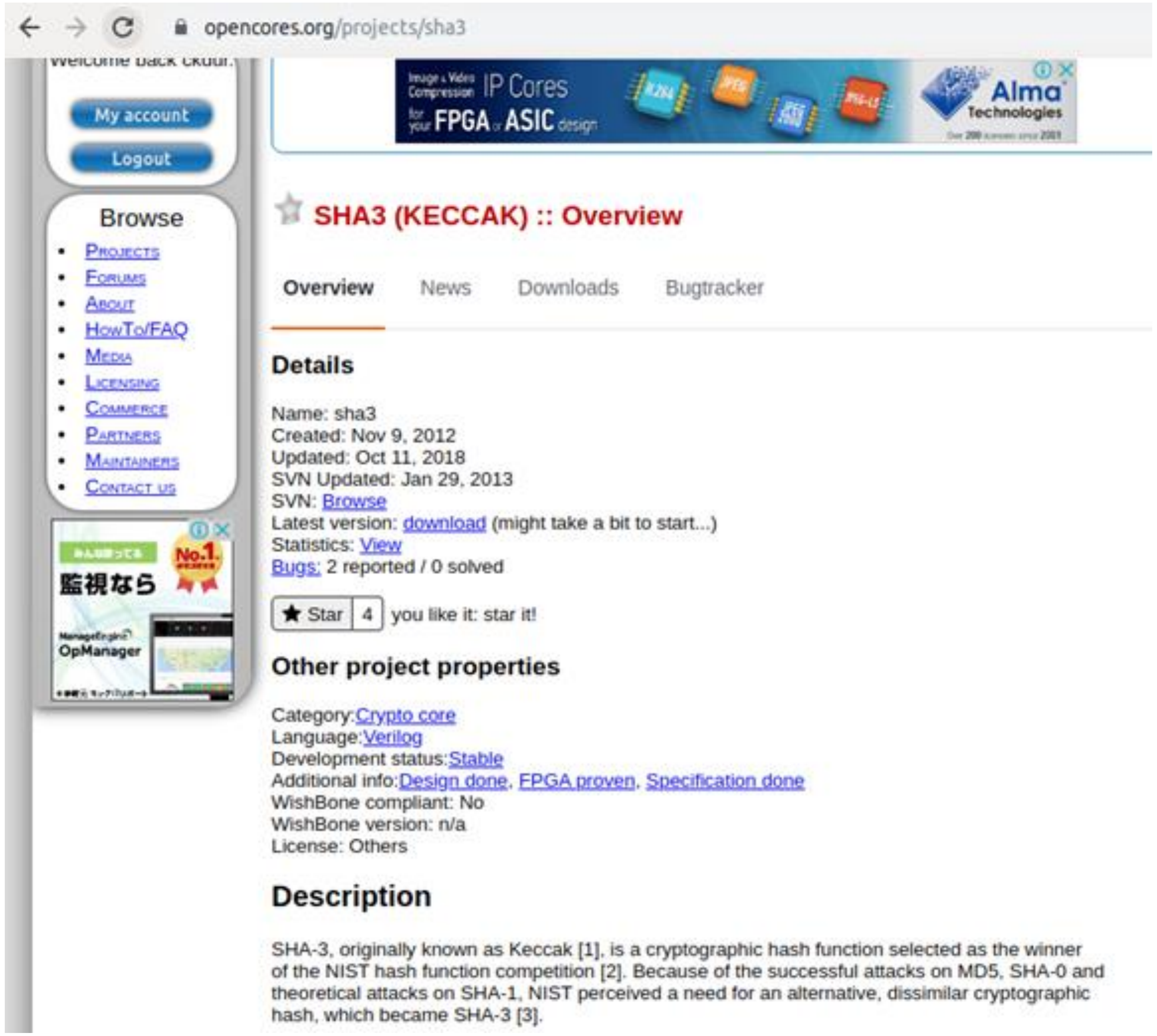
```
// TODO: From this point, insert any code
kputs("\r\n\n\nWelcome! Hello world!\r\n\n");
hwsha3_test((void*)sha3_reg);
kputs("\r\nEnd!\r\n");
// If finished, stay in a infinite loop
while(1);

//dead code
return 0;
```

Hardware test in software:



```
                                    BOOTING RATONA:


RATONA Demo:       2023-09- 3-12:51:49-1a8c631-dirty
Got TL_CLK: 50000000
Got NUM_CORES: 1
Got TIMEBASE: 1000000


Welcome! Hello world!

Begin SHA-3 hardware test:

Software: 0s 24ms 801us
0dd90aab4cddb98bb6dc6aec66e809817966f1a8ae0f586525207d
d2b0717e21c90ac78d64e607db71132bac2e92c83b9abd5a72cdff
9ce4f9e35c77eb589979

Hardware: 0s 0ms 93us
0dd90aab4cddb98bb6dc6aec66e809817966f1a8ae0f586525207d
d2b0717e21c90ac78d64e607db71132bac2e92c83b9abd5a72cdff
9ce4f9e35c77eb589979

SHA-3 hardware test passed!



End!
```

2. Hash function: SHA3 (2/10) Open-source module



SHA3 (KECCAK) :: Overview

Details

Name: sha3
Created: Nov 9, 2012
Updated: Oct 11, 2018
SVN Updated: Jan 29, 2013
SVN: Browse
Latest version: download (might take a bit to start...)
Statistics: View
Bugs: 2 reported / 0 solved

you like it: star it!

Other project properties

Category:Crypto core
Language:Verilog
Development status:Stable
Additional info:Design done, FPGA proven, Specification done
WishBone compliant: No
WishBone version: n/a
License: Others

Description

SHA-3, originally known as Keccak [1], is a cryptographic hash function selected as the winner of the NIST hash function competition [2]. Because of the successful attacks on MD5, SHA-0 and theoretical attacks on SHA-1, NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3 [3].

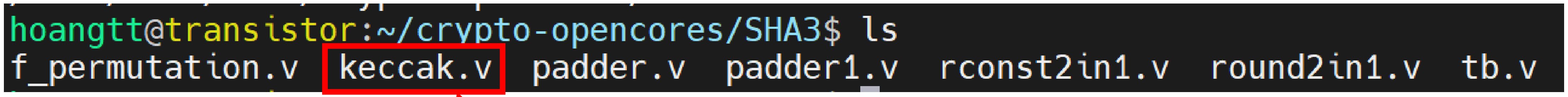
Using available open-sources core:
https://opencores.org/projects/sha3

The core is written in Verilog HDL code

Compare to the original design, the core presented in this class has been optimized for a better performance by:

- Cut-off not using parts
- Rewrite in true RTL code for better synthesis result
- Adding buffer/register to increase speed

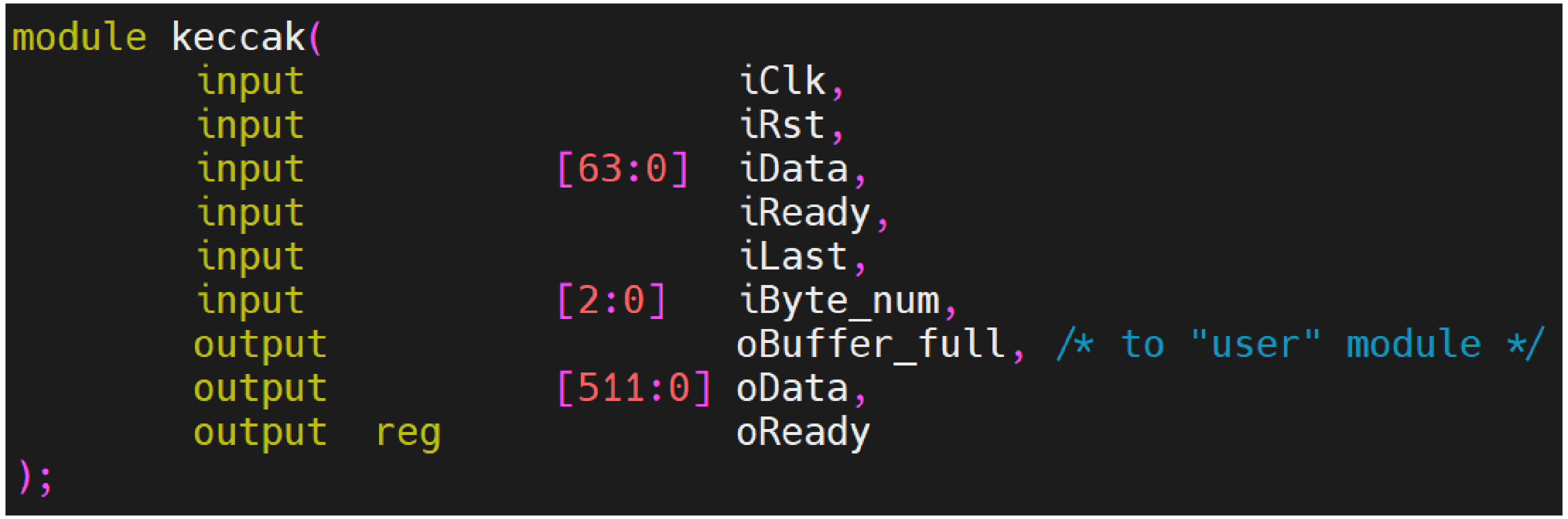
2. Hash function: SHA3 (3/10) Core ports



```
hoangtt@transistor:~/crypto-opencores/SHA3$ ls
f_permutation.v  keccak.v  padder.v  padder1.v  rconst2in1.v  round2in1.v  tb.v
```

TOP file

The **keccak** module's ports.



```
module keccak(
        input                   iClk,
        input                   iRst,
        input           [63:0]  iData,
        input                   iReady,
        input                   iLast,
        input           [2:0]   iByte_num,
        output                  oBuffer_full, /* to "user" module */
        output          [511:0] oData,
        output  reg             oReady
);
```

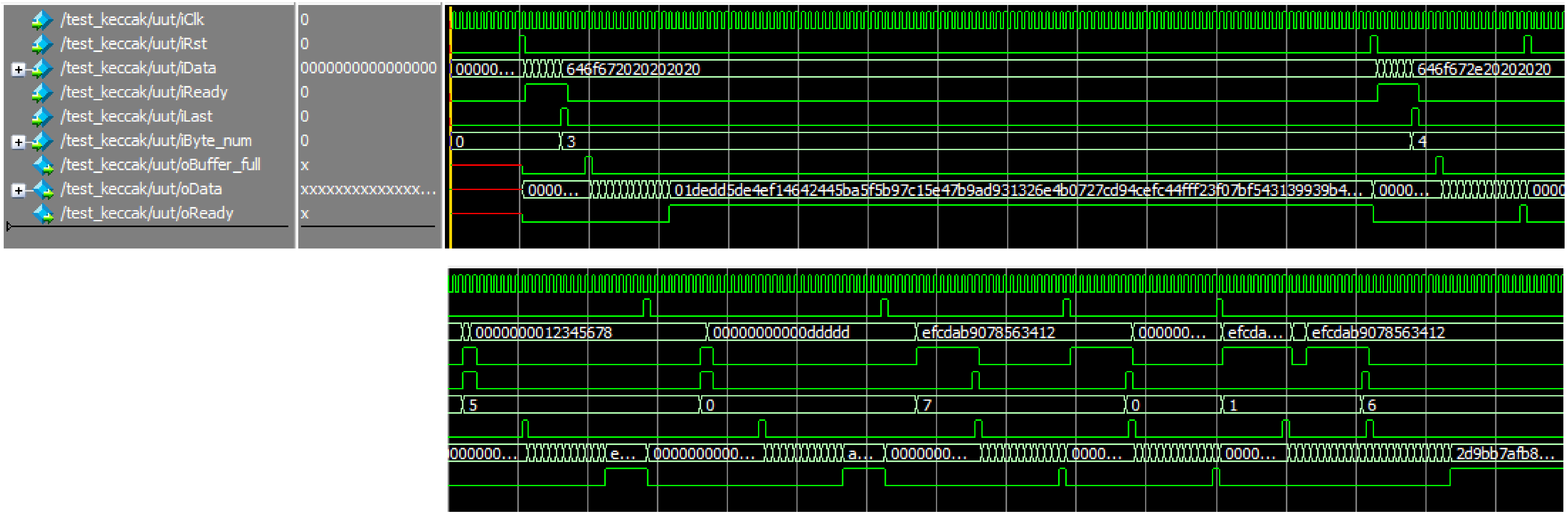
2. Hash function: SHA3 (4/10) Core ports

The **keccak** module's ports.

Name	**Width**	**Direction**	**Description**
iClk	1	In	Clock
iRst	1	In	Synchronous reset
iData	64	In	Input data
iByte_num	3	In	The byte length of *iData*
iReady	1	In	Valid for *iData*
iLast	1	In	Current *iData* is last or not
oBuffer_full	1	Out	Buffer is full or not
oData	512	Out	Hash result
oReady	1	Out	Valid for *oData*

2. Hash function: SHA3 (5/10) Waveform

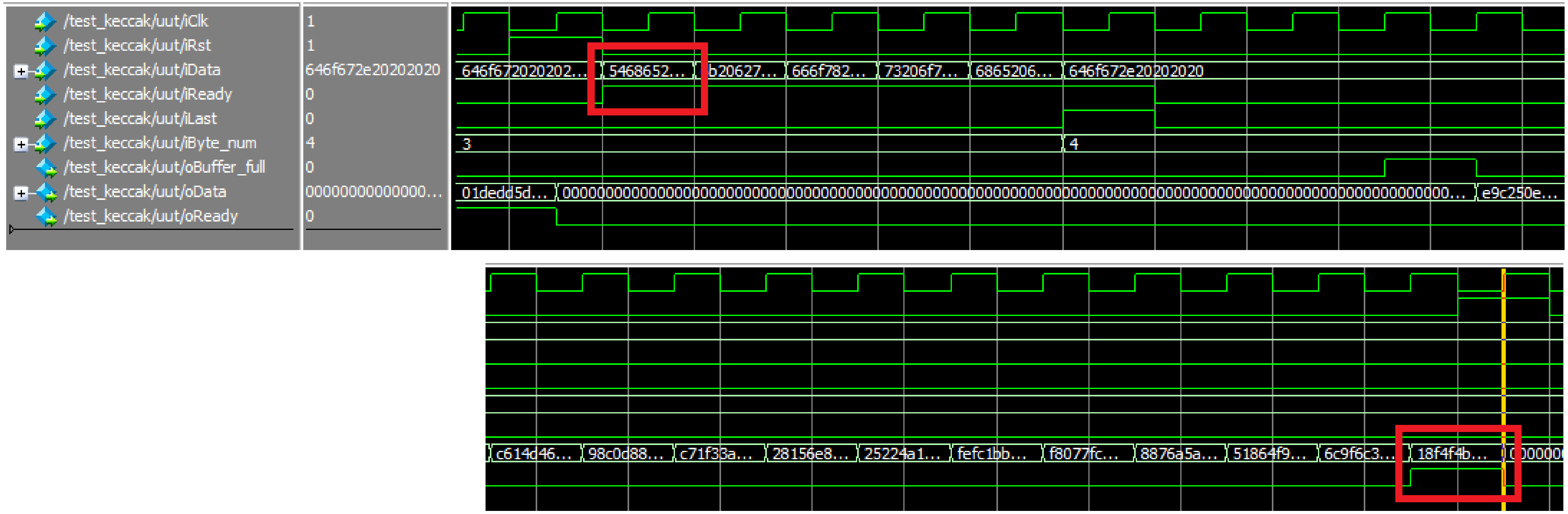
Example waveform of the **SHA3** module



```
/test_keccak/uut/iClk           0
/test_keccak/uut/iRst           0
/test_keccak/uut/iData          0000000000000000
/test_keccak/uut/iReady         0
/test_keccak/uut/iLast          0
/test_keccak/uut/iByte_num      0
/test_keccak/uut/oBuffer_full   x
/test_keccak/uut/oData          xxxxxxxxxxxxxx...
/test_keccak/uut/oReady         x
```

2. Hash function: SHA3 (6/10) Waveform

Close-up of one transaction from the first data in to the hash result out

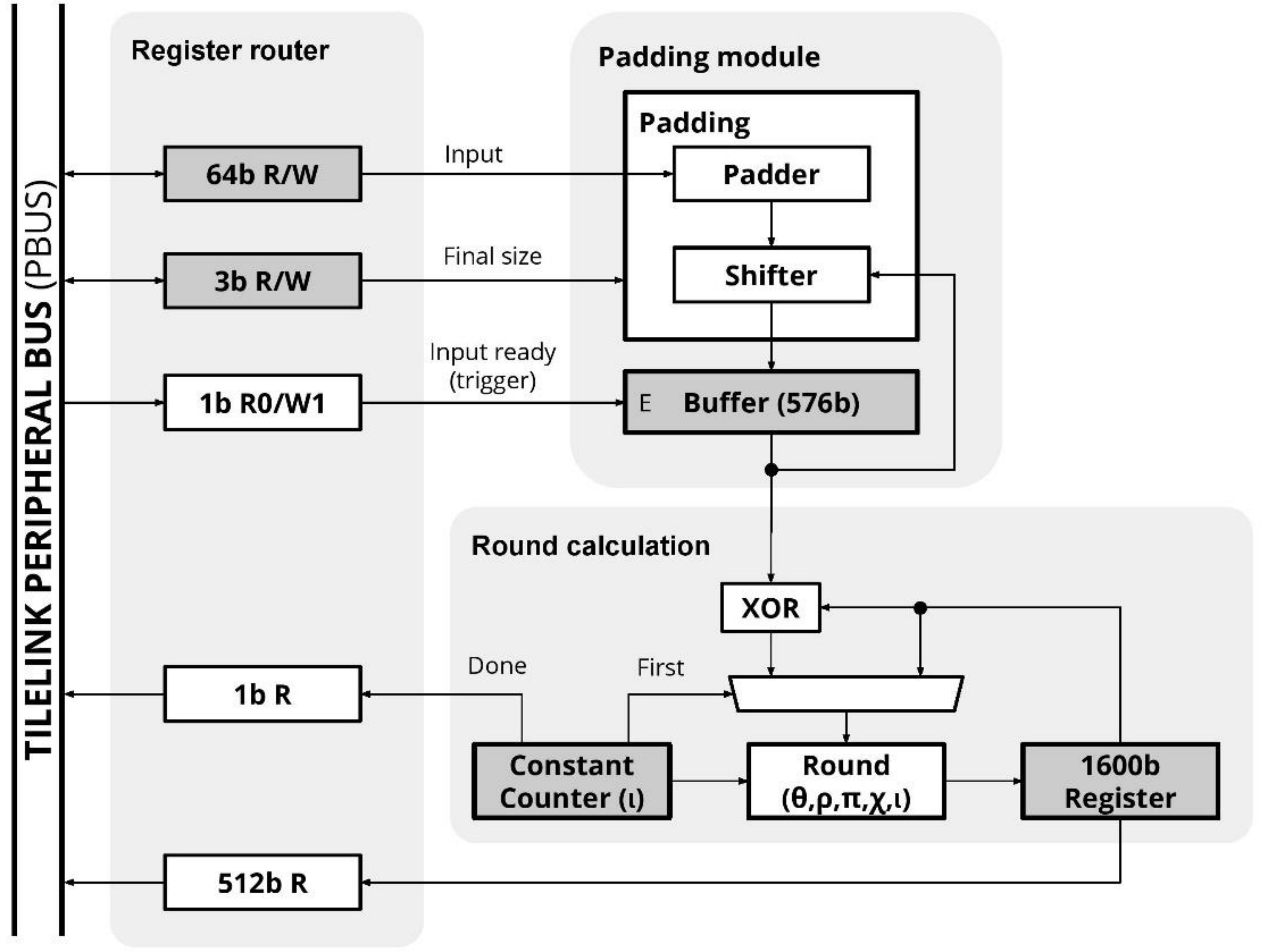


```
/test_keccak/uut/iClk          1
/test_keccak/uut/iRst          1
/test_keccak/uut/iData         646f672e20202020
/test_keccak/uut/iReady        0
/test_keccak/uut/iLast         0
/test_keccak/uut/iByte_num     4
/test_keccak/uut/oBuffer_full  0
/test_keccak/uut/oData         00000000000000...
/test_keccak/uut/oReady        0
```

```
iData:      646f672020202...  5468652...  b20627...  666f782...  73206f7...  6865206...  646f672e20202020
iByte_num:  3  4
oData:      01dedd5d...  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000...  e9c250e...
```

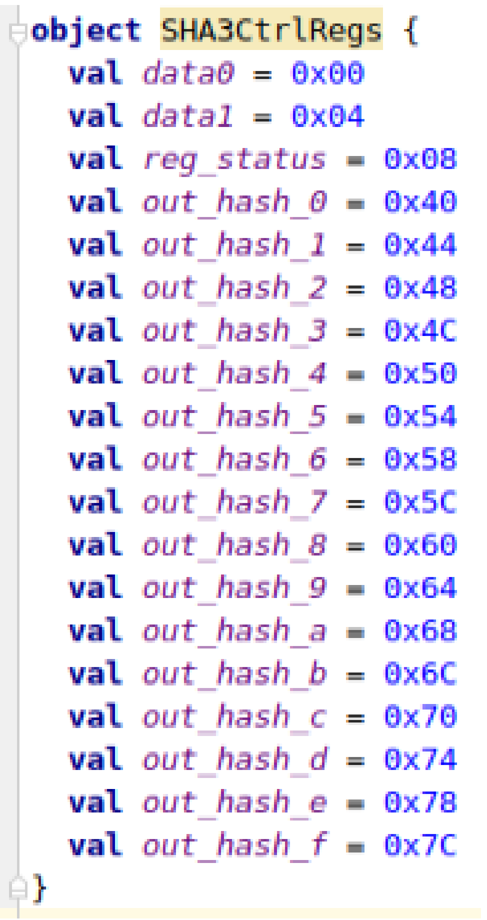
```
oData:  c614d46...  98c0d88...  c71f33a...  28156e8...  25224a1...  fefc1bb...  f8077fc...  8876a5a...  51864f9...  6c9f6c3...  18f4f4b...  000000
```

2. Hash function: SHA3 (7/10) Register-map

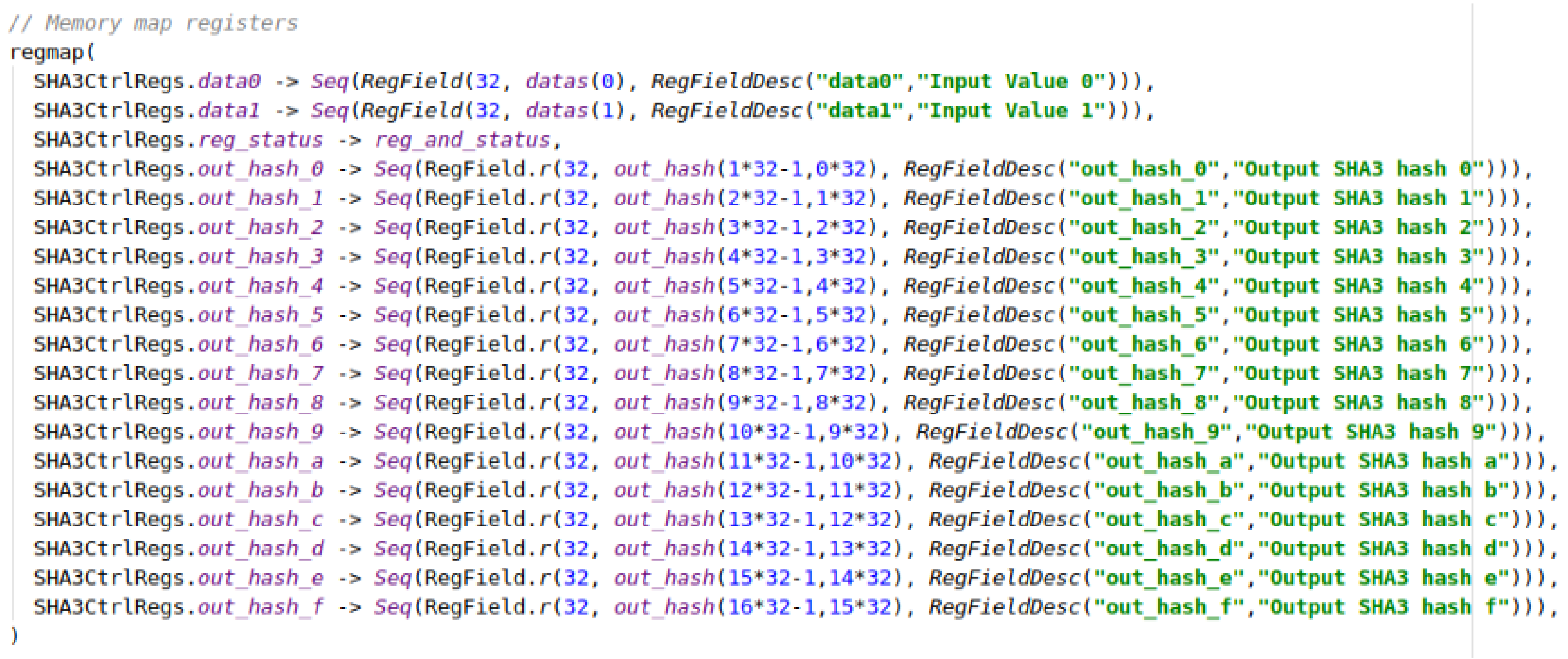


SHA3 module's register-map

2. Hash function: **SHA3** (8/10) Register-map



```scala
object SHA3CtrlRegs {
  val data0 = 0x00
  val data1 = 0x04
  val reg_status = 0x08
  val out_hash_0 = 0x40
  val out_hash_1 = 0x44
  val out_hash_2 = 0x48
  val out_hash_3 = 0x4C
  val out_hash_4 = 0x50
  val out_hash_5 = 0x54
  val out_hash_6 = 0x58
  val out_hash_7 = 0x5C
  val out_hash_8 = 0x60
  val out_hash_9 = 0x64
  val out_hash_a = 0x68
  val out_hash_b = 0x6C
  val out_hash_c = 0x70
  val out_hash_d = 0x74
  val out_hash_e = 0x78
  val out_hash_f = 0x7C
}
```



```scala
// Memory map registers
regmap(
  SHA3CtrlRegs.data0 -> Seq(RegField(32, datas(0), RegFieldDesc("data0","Input Value 0"))),
  SHA3CtrlRegs.data1 -> Seq(RegField(32, datas(1), RegFieldDesc("data1","Input Value 1"))),
  SHA3CtrlRegs.reg_status -> reg_and_status,
  SHA3CtrlRegs.out_hash_0 -> Seq(RegField.r(32, out_hash(1*32-1,0*32), RegFieldDesc("out_hash_0","Output SHA3 hash 0"))),
  SHA3CtrlRegs.out_hash_1 -> Seq(RegField.r(32, out_hash(2*32-1,1*32), RegFieldDesc("out_hash_1","Output SHA3 hash 1"))),
  SHA3CtrlRegs.out_hash_2 -> Seq(RegField.r(32, out_hash(3*32-1,2*32), RegFieldDesc("out_hash_2","Output SHA3 hash 2"))),
  SHA3CtrlRegs.out_hash_3 -> Seq(RegField.r(32, out_hash(4*32-1,3*32), RegFieldDesc("out_hash_3","Output SHA3 hash 3"))),
  SHA3CtrlRegs.out_hash_4 -> Seq(RegField.r(32, out_hash(5*32-1,4*32), RegFieldDesc("out_hash_4","Output SHA3 hash 4"))),
  SHA3CtrlRegs.out_hash_5 -> Seq(RegField.r(32, out_hash(6*32-1,5*32), RegFieldDesc("out_hash_5","Output SHA3 hash 5"))),
  SHA3CtrlRegs.out_hash_6 -> Seq(RegField.r(32, out_hash(7*32-1,6*32), RegFieldDesc("out_hash_6","Output SHA3 hash 6"))),
  SHA3CtrlRegs.out_hash_7 -> Seq(RegField.r(32, out_hash(8*32-1,7*32), RegFieldDesc("out_hash_7","Output SHA3 hash 7"))),
  SHA3CtrlRegs.out_hash_8 -> Seq(RegField.r(32, out_hash(9*32-1,8*32), RegFieldDesc("out_hash_8","Output SHA3 hash 8"))),
  SHA3CtrlRegs.out_hash_9 -> Seq(RegField.r(32, out_hash(10*32-1,9*32), RegFieldDesc("out_hash_9","Output SHA3 hash 9"))),
  SHA3CtrlRegs.out_hash_a -> Seq(RegField.r(32, out_hash(11*32-1,10*32), RegFieldDesc("out_hash_a","Output SHA3 hash a"))),
  SHA3CtrlRegs.out_hash_b -> Seq(RegField.r(32, out_hash(12*32-1,11*32), RegFieldDesc("out_hash_b","Output SHA3 hash b"))),
  SHA3CtrlRegs.out_hash_c -> Seq(RegField.r(32, out_hash(13*32-1,12*32), RegFieldDesc("out_hash_c","Output SHA3 hash c"))),
  SHA3CtrlRegs.out_hash_d -> Seq(RegField.r(32, out_hash(14*32-1,13*32), RegFieldDesc("out_hash_d","Output SHA3 hash d"))),
  SHA3CtrlRegs.out_hash_e -> Seq(RegField.r(32, out_hash(15*32-1,14*32), RegFieldDesc("out_hash_e","Output SHA3 hash e"))),
  SHA3CtrlRegs.out_hash_f -> Seq(RegField.r(32, out_hash(16*32-1,15*32), RegFieldDesc("out_hash_f","Output SHA3 hash f"))),
)
```

SHA3 module’s register-map

2. Hash function: SHA3 (9/10) Register-map

SHA3 module's register-map

Address	Mode	Name	Description
`0x00 - 0x04`	RW	Data	64-bit data in to hash
`0x08`	RW	Status	Configuration and current Status
`0x0C - 0x3C`	--	Reserved	Reserved
`0x40 - 0x7C`	R	Out_Hash	512-bit SHA-3 hash

Status Register (0x08)

Bits	Mode	Name	Description
[3:0]	RW	byte_num	Number of Bytes to hash (1 to 7. 0 is all)
[7:4]	--	Reserved	Reserved
[8]	R	busy	Busy
[9]	R	buff_full	Buffer Full
[10]	R	out_notready	Output Not Ready
[15:11]	--	Reserved	Reserved
[16]	W	commit	Write '1' to commit current input to the Hash calculation
[17]	RW	last	Indicates last portion of data in the input stream
[23:18]	--	Reserved	Reserved
[24]	RW	reset	Resets the hash calculator

2. Hash function: SHA3 (10/10) Software pseudo-code

Software pseudo-code for using **SHA-3** accelerator via registers

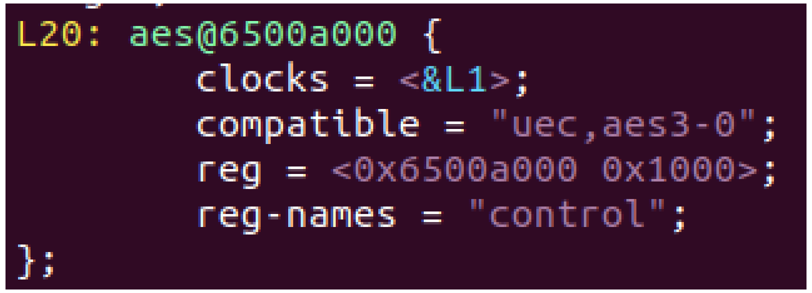
SHA3(message, size)

```
1. ref = 0;
   SHA3[size] = 0;
2. while(size >= 8)
       SHA3[block,0:7] = message[ref + 0:7];
       SHA3[trigger] = 1;
       ref += 8;
       size -= 8;
   endwhile
3. SHA3[size] = size;
   SHA3[block,0:7] = message[ref + 0:7];
   SHA3[last] = 1;
   SHA3[trigger] = 1;
4. Wait for SHA3[done];
   Return SHA3[result];
```

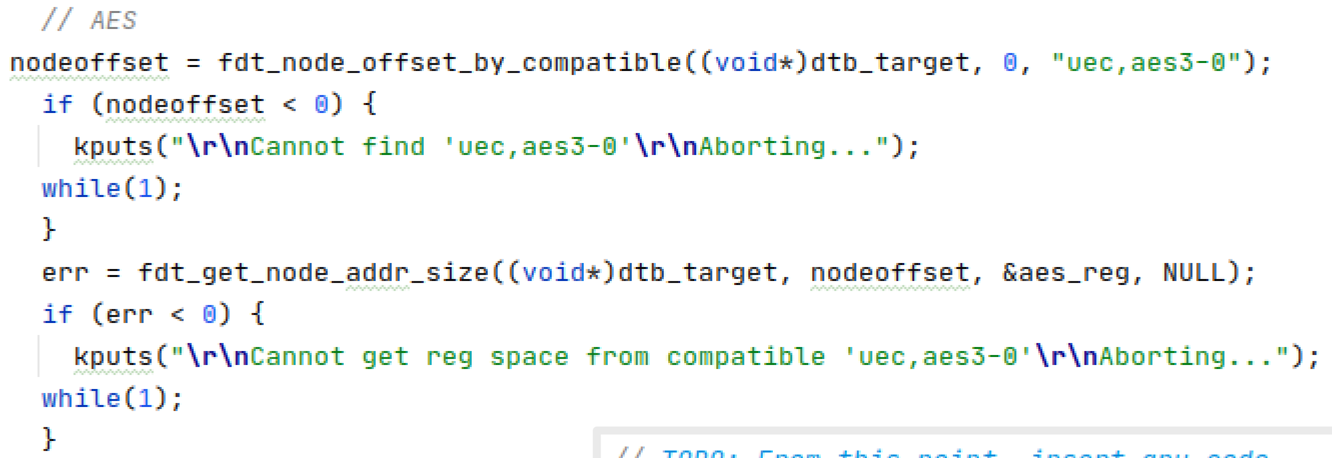
Outline

3. Cipher function: AES (1/9) The wanted result

The **AES** module is added to the system.

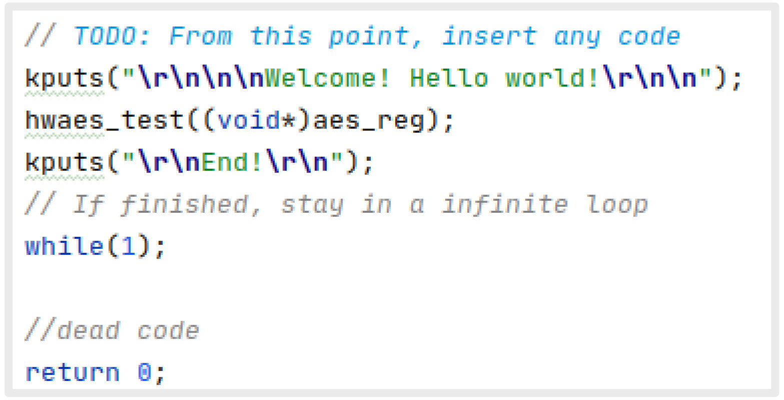


```
L20: aes@6500a000 {
        clocks = <&L1>;
        compatible = "uec,aes3-0";
        reg = <0x6500a000 0x1000>;
        reg-names = "control";
};
```



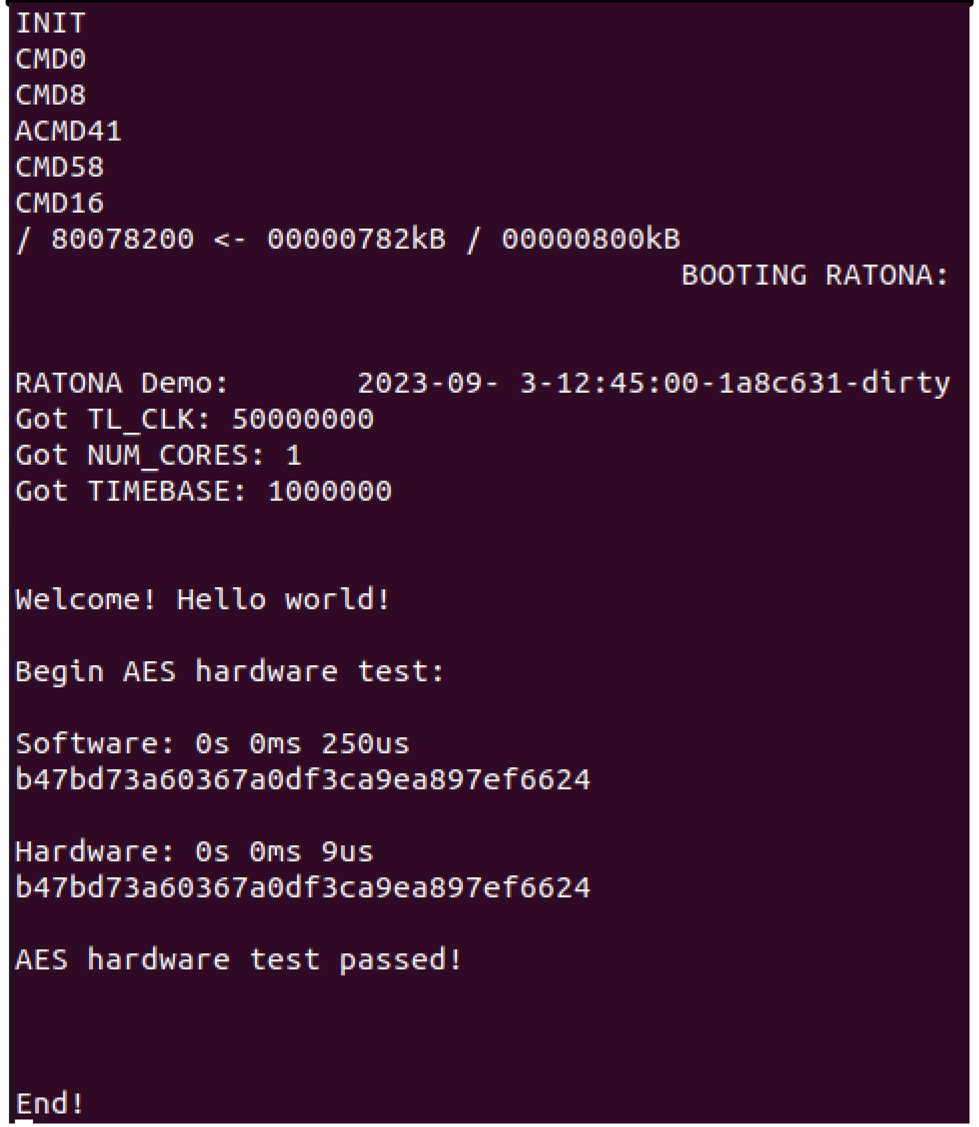
```
  // AES
nodeoffset = fdt_node_offset_by_compatible((void*)dtb_target, 0, "uec,aes3-0");
  if (nodeoffset < 0) {
    kputs("\r\nCannot find 'uec,aes3-0'\r\nAborting...");
  while(1);
  }
  err = fdt_get_node_addr_size((void*)dtb_target, nodeoffset, &aes_reg, NULL);
  if (err < 0) {
    kputs("\r\nCannot get reg space from compatible 'uec,aes3-0'\r\nAborting...");
  while(1);
  }
```

Get the module pointer in software and run the hardware test



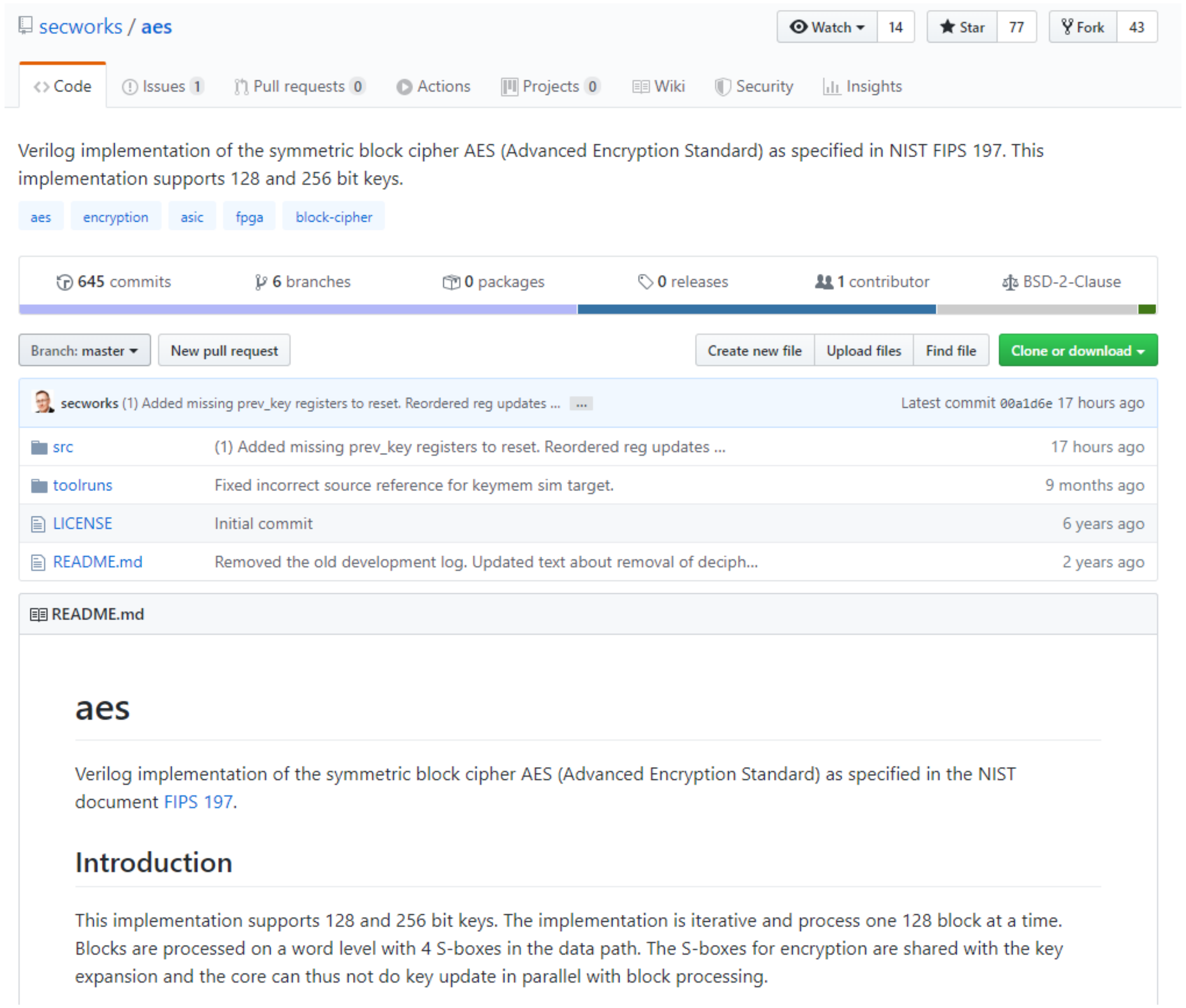
```
// TODO: From this point, insert any code
kputs("\r\n\n\nWelcome! Hello world!\r\n\n");
hwaes_test((void*)aes_reg);
kputs("\r\nEnd!\r\n");
// If finished, stay in a infinite loop
while(1);

//dead code
return 0;
```

Hardware test in software:



```
INIT
CMD0
CMD8
ACMD41
CMD58
CMD16
/ 80078200 <- 00000782kB / 00000800kB
                                      BOOTING RATONA:



RATONA Demo:       2023-09- 3-12:45:00-1a8c631-dirty
Got TL_CLK: 50000000
Got NUM_CORES: 1
Got TIMEBASE: 1000000


Welcome! Hello world!

Begin AES hardware test:

Software: 0s 0ms 250us
b47bd73a60367a0df3ca9ea897ef6624

Hardware: 0s 0ms 9us
b47bd73a60367a0df3ca9ea897ef6624

AES hardware test passed!



End!
```

3. Cipher function: AES (2/9) Open-source module



secworks / aes

Watch 14 | Star 77 | Fork 43

Code | Issues 1 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights

Verilog implementation of the symmetric block cipher AES (Advanced Encryption Standard) as specified in NIST FIPS 197. This implementation supports 128 and 256 bit keys.

aes encryption asic fpga block-cipher

645 commits | 6 branches | 0 packages | 0 releases | 1 contributor | BSD-2-Clause

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

secworks (1) Added missing prev_key registers to reset. Reordered reg updates ... Latest commit 00a1d6e 17 hours ago

src	(1) Added missing prev_key registers to reset. Reordered reg updates ...	17 hours ago
toolruns	Fixed incorrect source reference for keymem sim target.	9 months ago
LICENSE	Initial commit	6 years ago
README.md	Removed the old development log. Updated text about removal of deciph...	2 years ago

README.md

aes

Verilog implementation of the symmetric block cipher AES (Advanced Encryption Standard) as specified in the NIST document FIPS 197.

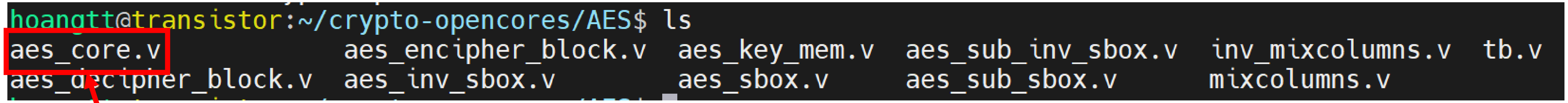
Introduction

This implementation supports 128 and 256 bit keys. The implementation is iterative and process one 128 block at a time. Blocks are processed on a word level with 4 S-boxes in the data path. The S-boxes for encryption are shared with the key expansion and the core can thus not do key update in parallel with block processing.

Using available open-sources core
https://github.com/secworks/aes

- Configuration of -128 or -256 can be changed on the fly
- Four SBOXes & four InvSBOXes are made by Lookup-table
- Fully hardware encryption / decryption

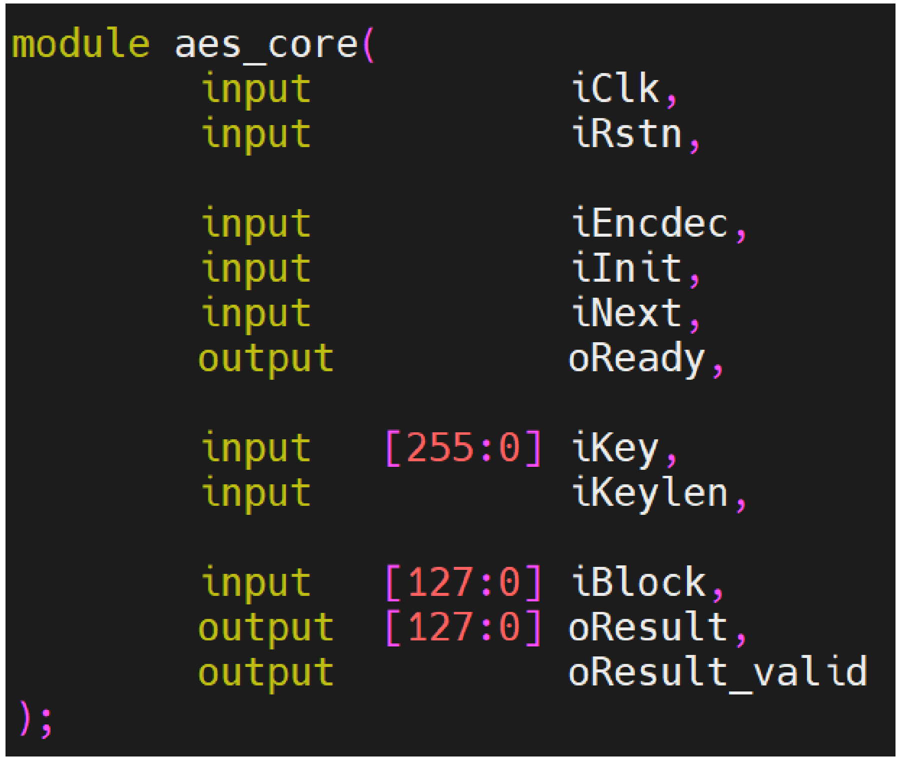
3. Cipher function: AES (3/9) Core ports



```
hoangtt@transistor:~/crypto-opencores/AES$ ls
aes_core.v              aes_encipher_block.v  aes_key_mem.v  aes_sub_inv_sbox.v  inv_mixcolumns.v  tb.v
aes_decipher_block.v    aes_inv_sbox.v        aes_sbox.v     aes_sub_sbox.v      mixcolumns.v
```

TOP file

The **aes_core** module's ports.



```
module aes_core(
        input               iClk,
        input               iRstn,

        input               iEncdec,
        input               iInit,
        input               iNext,
        output              oReady,

        input     [255:0]   iKey,
        input               iKeylen,

        input     [127:0]   iBlock,
        output    [127:0]   oResult,
        output              oResult_valid
);
```

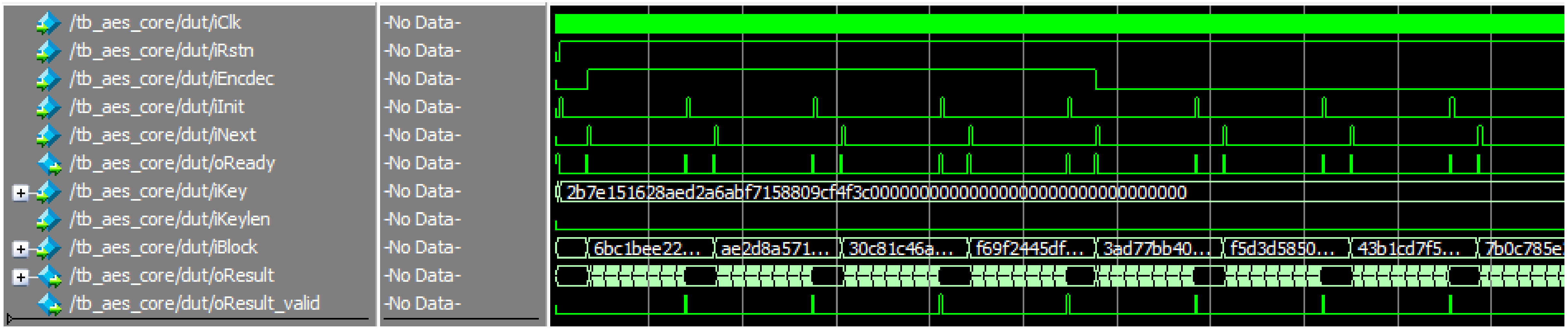
3. Cipher function: AES (4/9) Core ports

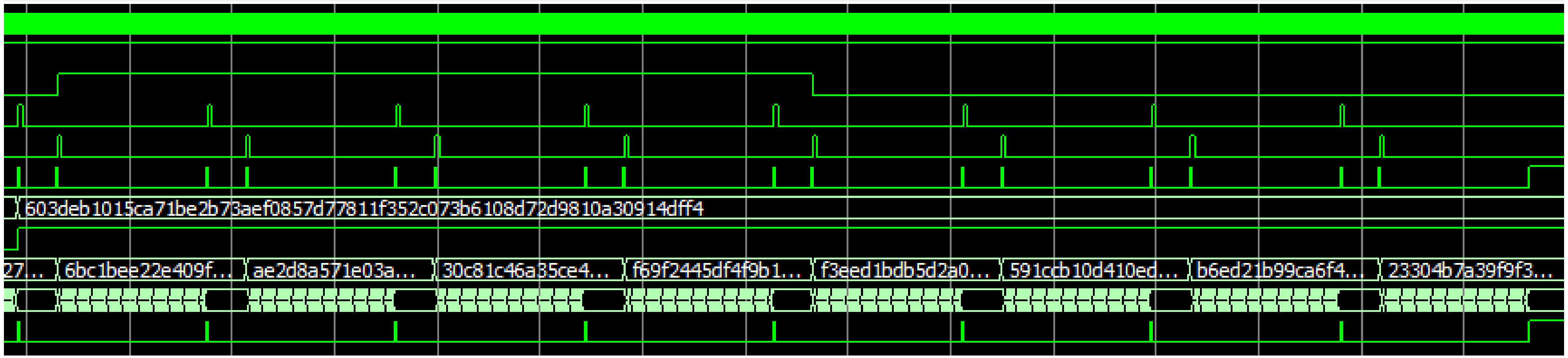
The **aes_core** module's ports.

Name	**Width**	**Direction**	**Description**
iClk	1	In	Clock
iRstn	1	In	Synchronous reset
iEncdec	1	In	0: Decrypt; 1: Encrypt
iInit	1	In	Specify the init round, *KeyExpansion* round. When asserted, the key is updated via *iKey* and *iKeylen*
iNext	1	In	Valid for *iBlock*
oReady	1	Out	Signal that the core is ready for a new *iInit* or *iNext*
iKey	256	In	Key in
iKeylen	1	In	0: use 128-bit key; 1: use 256-bit key
iBlock	128	In	128-bit block data in
oResult	128	Out	128-bit block data out
oResult_valid	1	Out	Valid for *oResult*

3. Cipher function: AES (5/9) Waveform

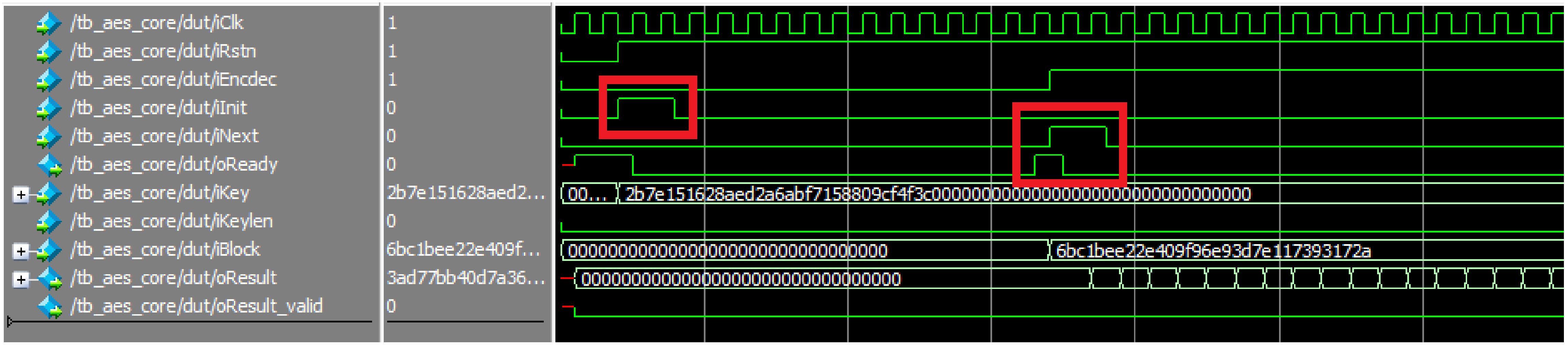
Example waveform of the **AES** module





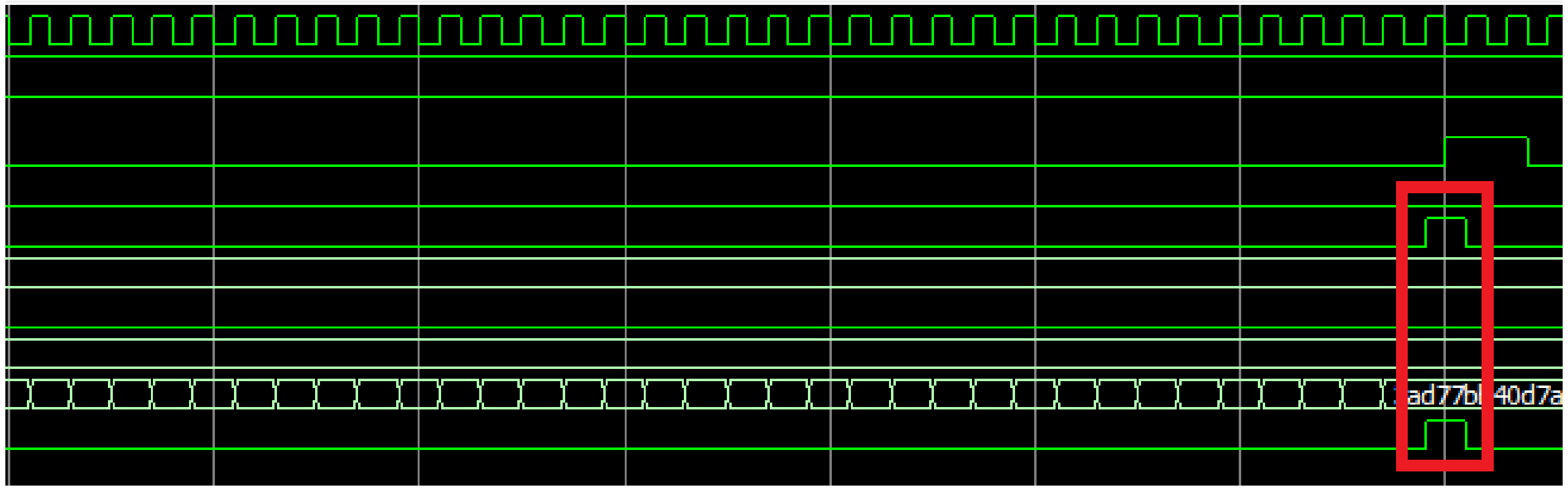
3. Cipher function: AES (6/9) Waveform

Close-up of one encipher block transaction.

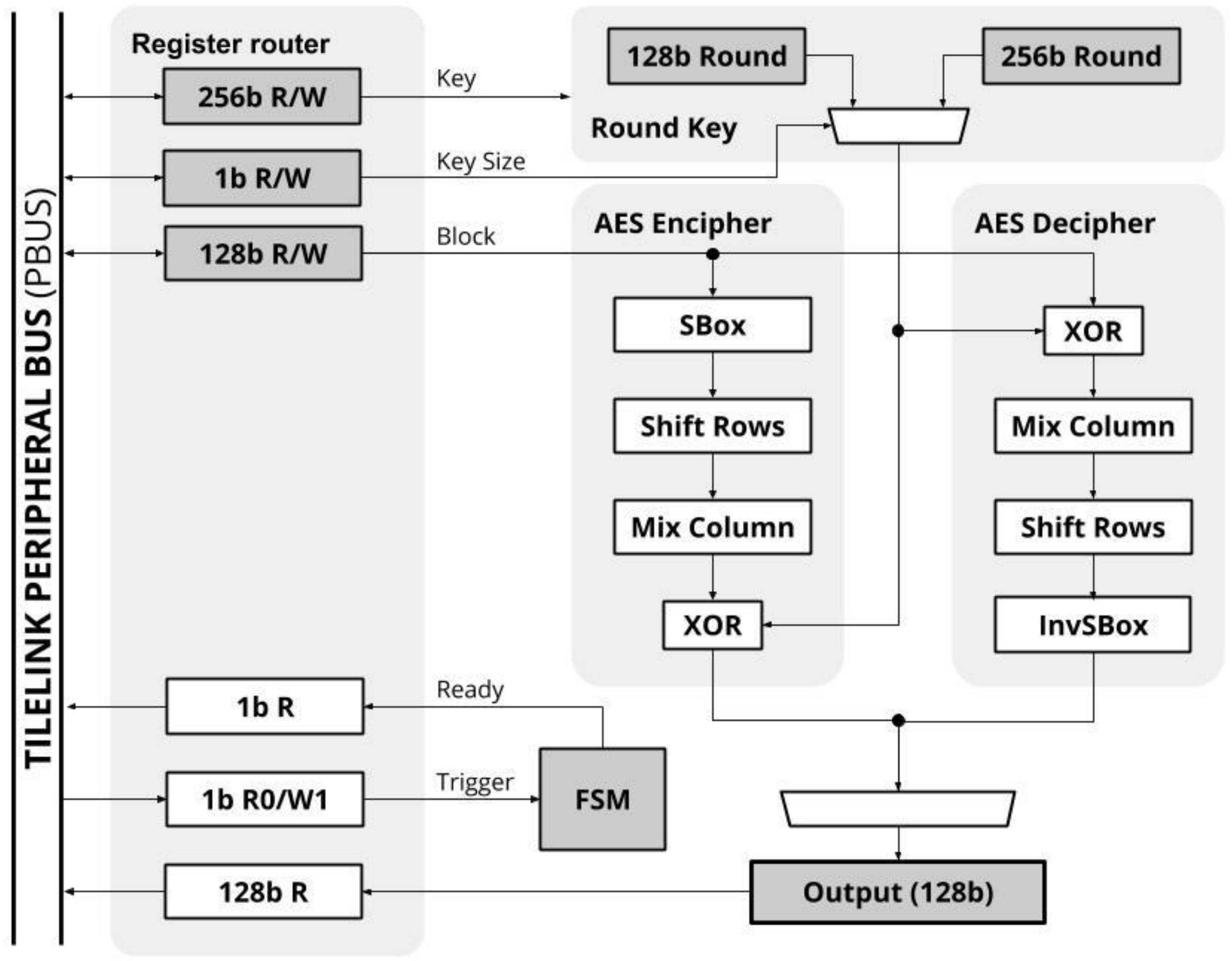


```
/tb_aes_core/dut/iClk           1
/tb_aes_core/dut/iRstn          1
/tb_aes_core/dut/iEncdec        1
/tb_aes_core/dut/iInit          0
/tb_aes_core/dut/iNext          0
/tb_aes_core/dut/oReady         0
/tb_aes_core/dut/iKey           2b7e151628aed2...
/tb_aes_core/dut/iKeylen        0
/tb_aes_core/dut/iBlock         6bc1bee22e409f...
/tb_aes_core/dut/oResult        3ad77bb40d7a36...
/tb_aes_core/dut/oResult_valid  0
```

00...
2b7e151628aed2a6abf7158809cf4f3c00000000000000000000000000000000
00000000000000000000000000000000
6bc1bee22e409f96e93d7e117393172a
00000000000000000000000000000000



3. Cipher function: AES (7/9) Register-map



AES module's register-map

3. Cipher function: AES (8/9) Register-map

AES module’s register-map

Address	Mode	Name	Description
`0x000 - 0x01C`	RW	Key	128 or 256-bit Key
`0x020 - 0x02C`	RW	Block	Block to Cypher or De-cypher
`0x030 - 0x03C`	R	Result	Cypher-text or decypher-text
`0x040 - 0xFF4`	--	Reserved	Reserved
`0xFF8`	RW	Config	Configuration of the AES
`0xFFC`	RW	Status	Status of AES calculation

Configuration of the AES (0xFF8)

Bits	Mode	Name	Description
`[0]`	RW	encdec	Encode or Decode AES
`[1]`	RW	keylen	Length of the Key (0 for 128, 1 for 256)

Status of AES calculation (0xFFC)

Bits	Mode	Name	Description
`[0]`	W	init	Write ‘1’ to encode or Decode AES
`[1]`	RW	keylen	Length of the Key (0 for 128, 1 for 256)
`[2]`	R	ready	AES ready

3. Cipher function: AES (9/9) Software pseudo-code

The software pseudo-code for using **AES** accelerator via registers

AES_cypher(text, is256, first)

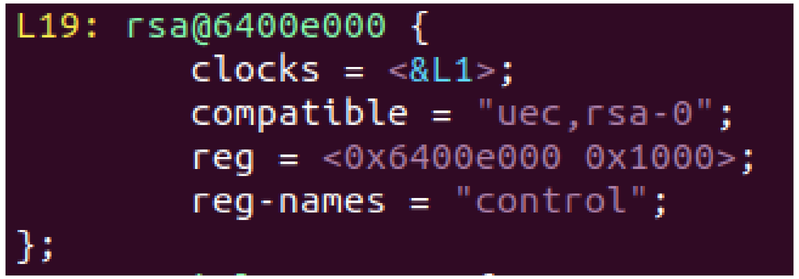
```
1.cyphertext = [];
  AES[config,size] = is256;
2.if(first) {
      AES[trigger,key_expansion] = 1;
      Wait for AES[done];
  }
3.AES[block] = text;
  AES[trigger,data] = 1;
4.Wait for AES[done];
  Return AES[result]
```

Outline

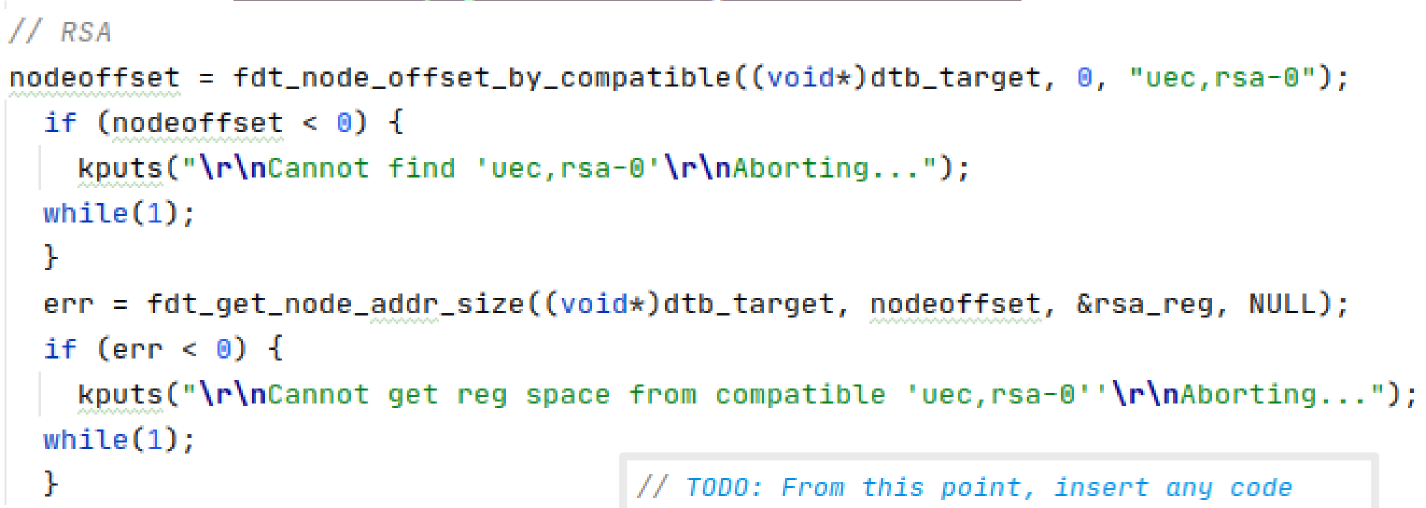
1. Preparation
2. Hash function: SHA3-512
3. Cipher function: AES-128
4. Crypto-key scheme: RSA-1024
5. Practice

4. Crypto-key scheme: RSA (1/6) The wanted result

The **RSA** module is added to the system.

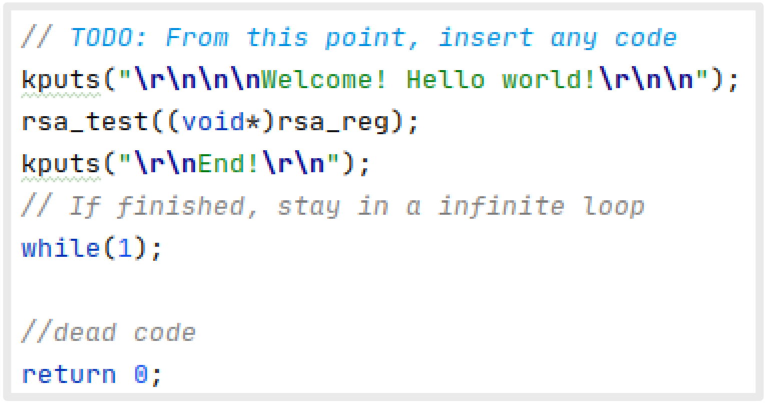


```
L19: rsa@6400e000 {
        clocks = <&L1>;
        compatible = "uec,rsa-0";
        reg = <0x6400e000 0x1000>;
        reg-names = "control";
};
```



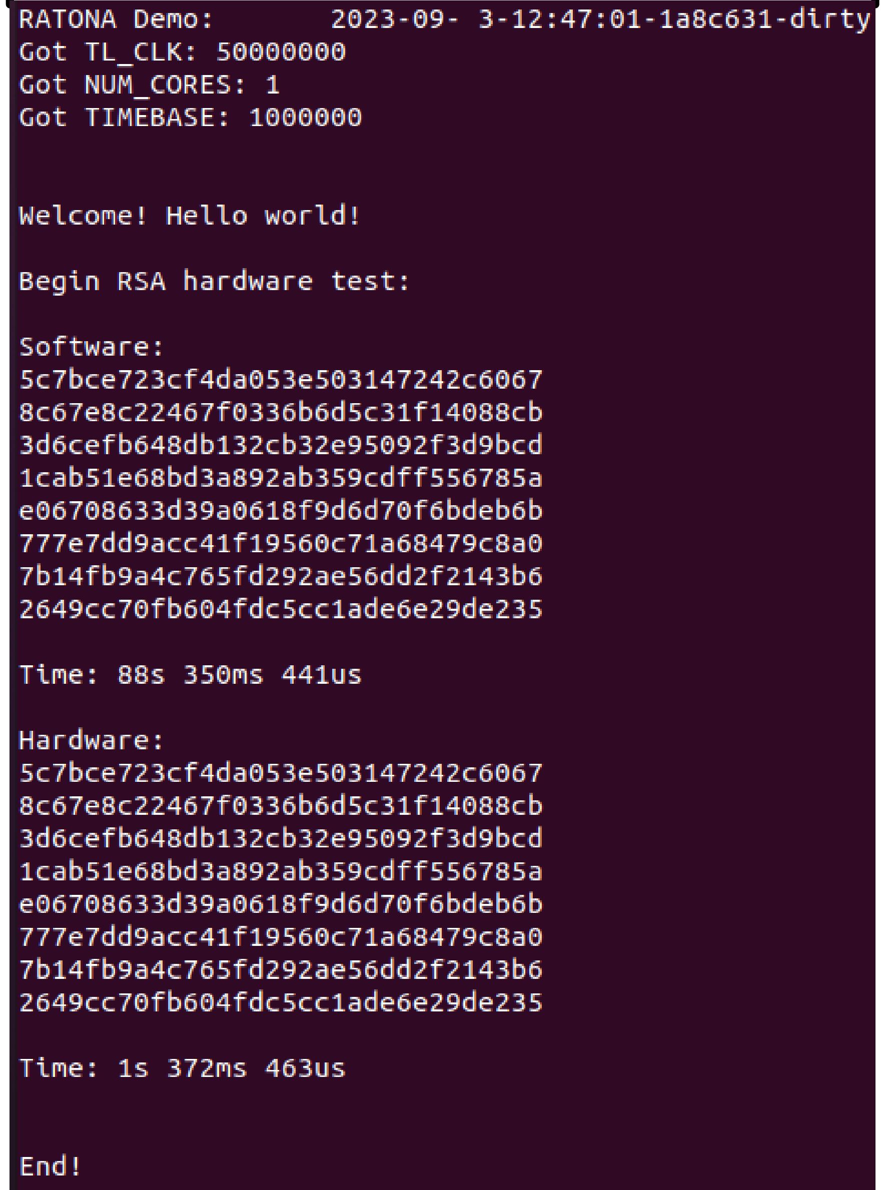
```
// RSA
nodeoffset = fdt_node_offset_by_compatible((void*)dtb_target, 0, "uec,rsa-0");
  if (nodeoffset < 0) {
    kputs("\r\nCannot find 'uec,rsa-0'\r\nAborting...");
  while(1);
  }
  err = fdt_get_node_addr_size((void*)dtb_target, nodeoffset, &rsa_reg, NULL);
  if (err < 0) {
    kputs("\r\nCannot get reg space from compatible 'uec,rsa-0''\r\nAborting...");
  while(1);
  }
```

Get the module pointer in software and run the hardware test



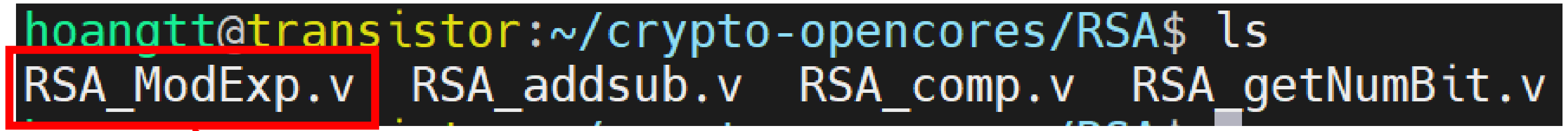
```
// TODO: From this point, insert any code
kputs("\r\n\n\nWelcome! Hello world!\r\n\n");
rsa_test((void*)rsa_reg);
kputs("\r\nEnd!\r\n");
// If finished, stay in a infinite loop
while(1);

//dead code
return 0;
```

Hardware test in software:



```
RATONA Demo:       2023-09- 3-12:47:01-1a8c631-dirty
Got TL_CLK: 50000000
Got NUM_CORES: 1
Got TIMEBASE: 1000000


Welcome! Hello world!

Begin RSA hardware test:

Software:
5c7bce723cf4da053e503147242c6067
8c67e8c22467f0336b6d5c31f14088cb
3d6cefb648db132cb32e95092f3d9bcd
1cab51e68bd3a892ab359cdff556785a
e06708633d39a0618f9d6d70f6bdeb6b
777e7dd9acc41f19560c71a68479c8a0
7b14fb9a4c765fd292ae56dd2f2143b6
2649cc70fb604fdc5cc1ade6e29de235

Time: 88s 350ms 441us

Hardware:
5c7bce723cf4da053e503147242c6067
8c67e8c22467f0336b6d5c31f14088cb
3d6cefb648db132cb32e95092f3d9bcd
1cab51e68bd3a892ab359cdff556785a
e06708633d39a0618f9d6d70f6bdeb6b
777e7dd9acc41f19560c71a68479c8a0
7b14fb9a4c765fd292ae56dd2f2143b6
2649cc70fb604fdc5cc1ade6e29de235

Time: 1s 372ms 463us


End!
```

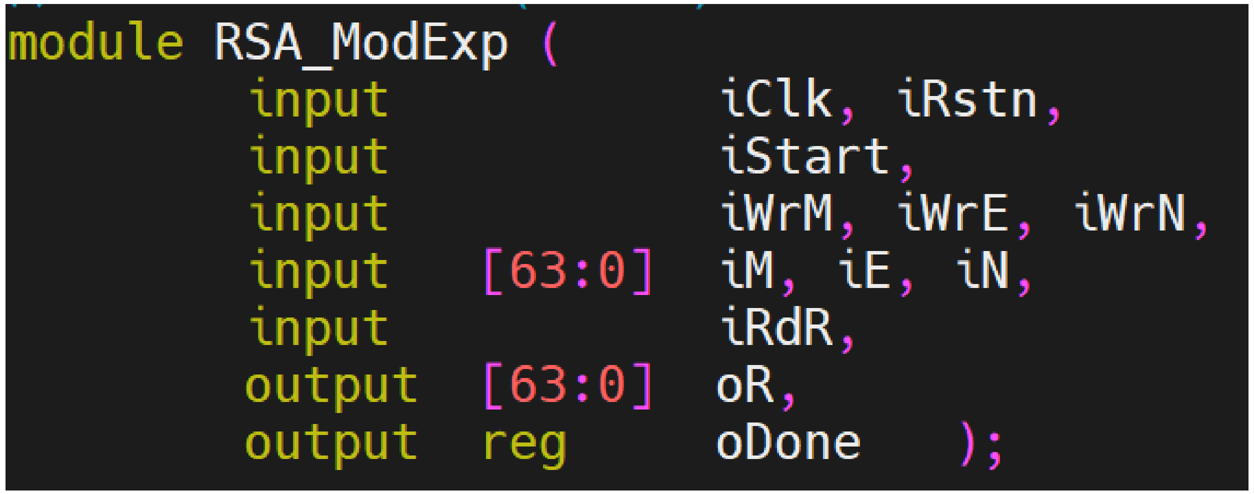
4. Crypto-key scheme: **RSA** (2/6) Core ports



```
hoangtt@transistor:~/crypto-opencores/RSA$ ls
RSA_ModExp.v  RSA_addsub.v  RSA_comp.v  RSA_getNumBit.v
```

TOP file

The **RSA_ModExp** module's ports.



```
module RSA_ModExp (
        input                iClk, iRstn,
        input                iStart,
        input                iWrM, iWrE, iWrN,
        input        [63:0]  iM, iE, iN,
        input                iRdR,
        output       [63:0]  oR,
        output       reg     oDone    );
```

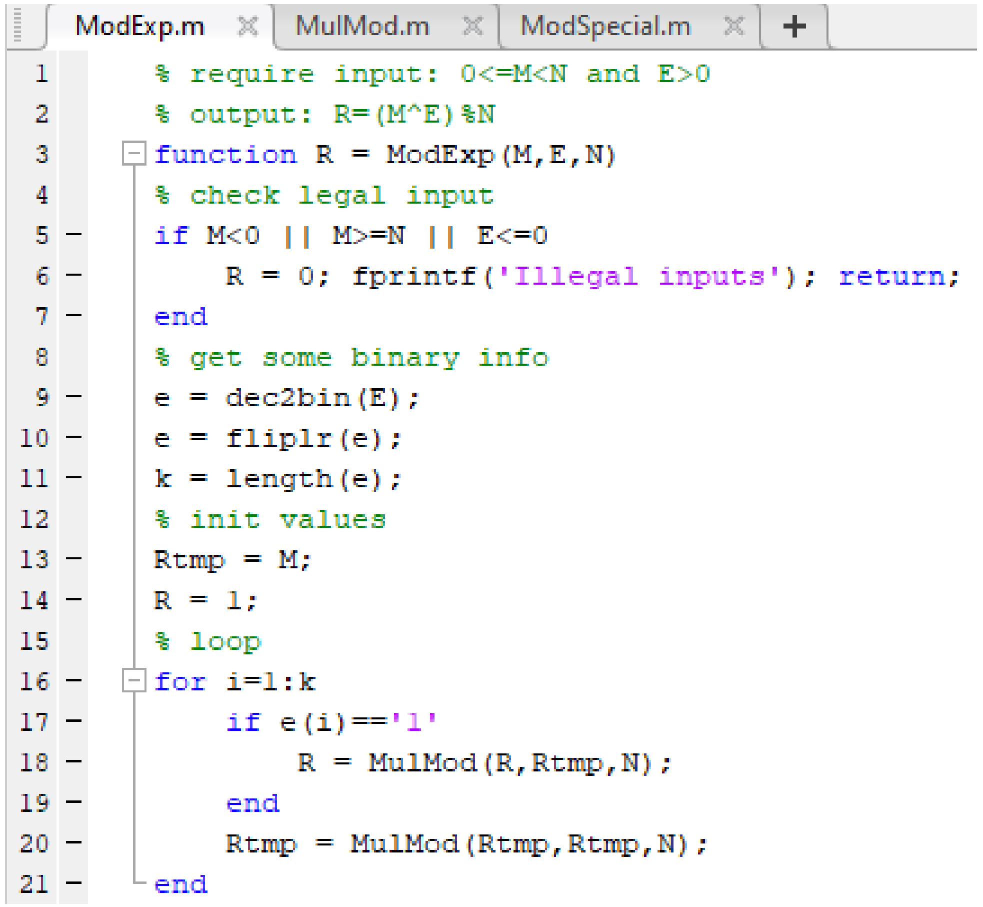
4. Crypto-key scheme: RSA (3/6) Core ports

The **RSA_ModExp** module's ports.

PIN	DIR	WIDTH	Description
		Control signals	
iClk	Input	1	Clock
iRstn	Input	1	Reset low
iStart	Input	1	Start computing
iWrM	Input	1	Write M
iWrE	Input	1	Write E
iWrN	Input	1	Write N
iRdR	Input	1	Read Result
		Input Data	
iM	Input	64	M
iE	Input	64	E
iN	Input	64	N
		Output Data	
oR	Output	64	Result
oDone	Output	1	Finish computing

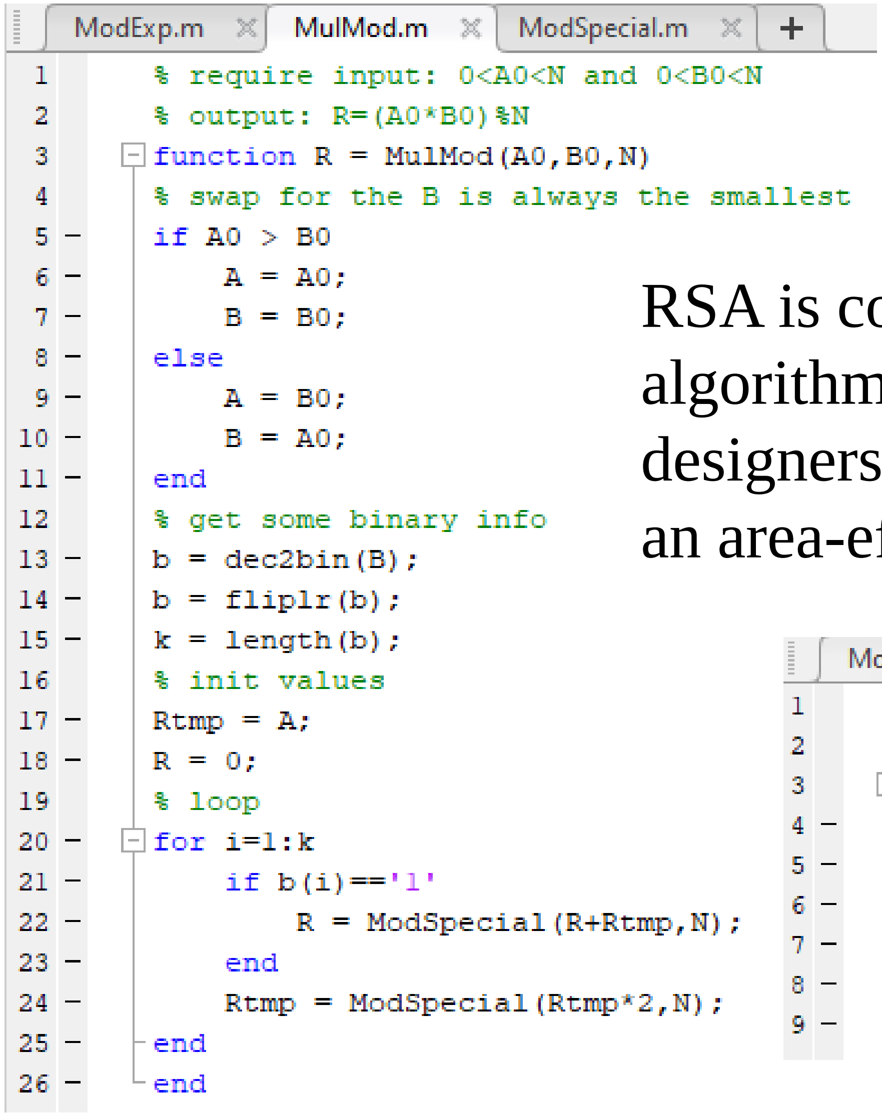
4. Crypto-key scheme: RSA (4/6) Design overview

The following are the Matlab codes represented for our implementation.



ModExp.m | MulMod.m | ModSpecial.m | +

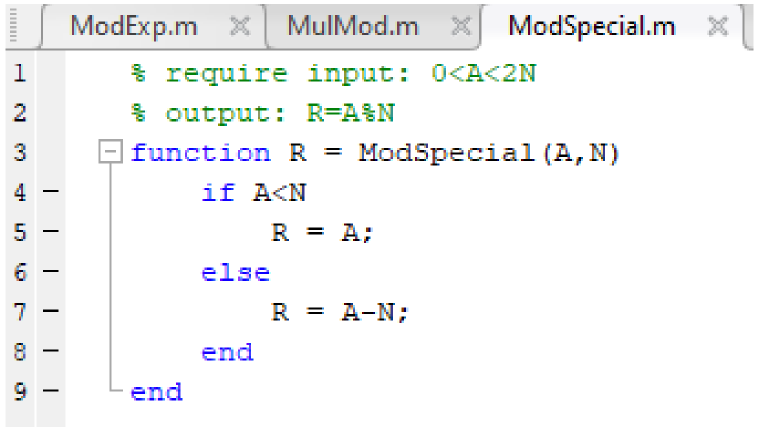
```
% require input: 0<=M<N and E>0
% output: R=(M^E)%N
function R = ModExp(M,E,N)
% check legal input
if M<0 || M>=N || E<=0
    R = 0; fprintf('Illegal inputs'); return;
end
% get some binary info
e = dec2bin(E);
e = fliplr(e);
k = length(e);
% init values
Rtmp = M;
R = 1;
% loop
for i=1:k
    if e(i)=='1'
        R = MulMod(R,Rtmp,N);
    end
    Rtmp = MulMod(Rtmp,Rtmp,N);
end
```



ModExp.m | MulMod.m | ModSpecial.m | +

```
% require input: 0<A0<N and 0<B0<N
% output: R=(A0*B0)%N
function R = MulMod(A0,B0,N)
% swap for the B is always the smallest
if A0 > B0
    A = A0;
    B = B0;
else
    A = B0;
    B = A0;
end
% get some binary info
b = dec2bin(B);
b = fliplr(b);
k = length(b);
% init values
Rtmp = A;
R = 0;
% loop
for i=1:k
    if b(i)=='1'
        R = ModSpecial(R+Rtmp,N);
    end
    Rtmp = ModSpecial(Rtmp*2,N);
end
end
```

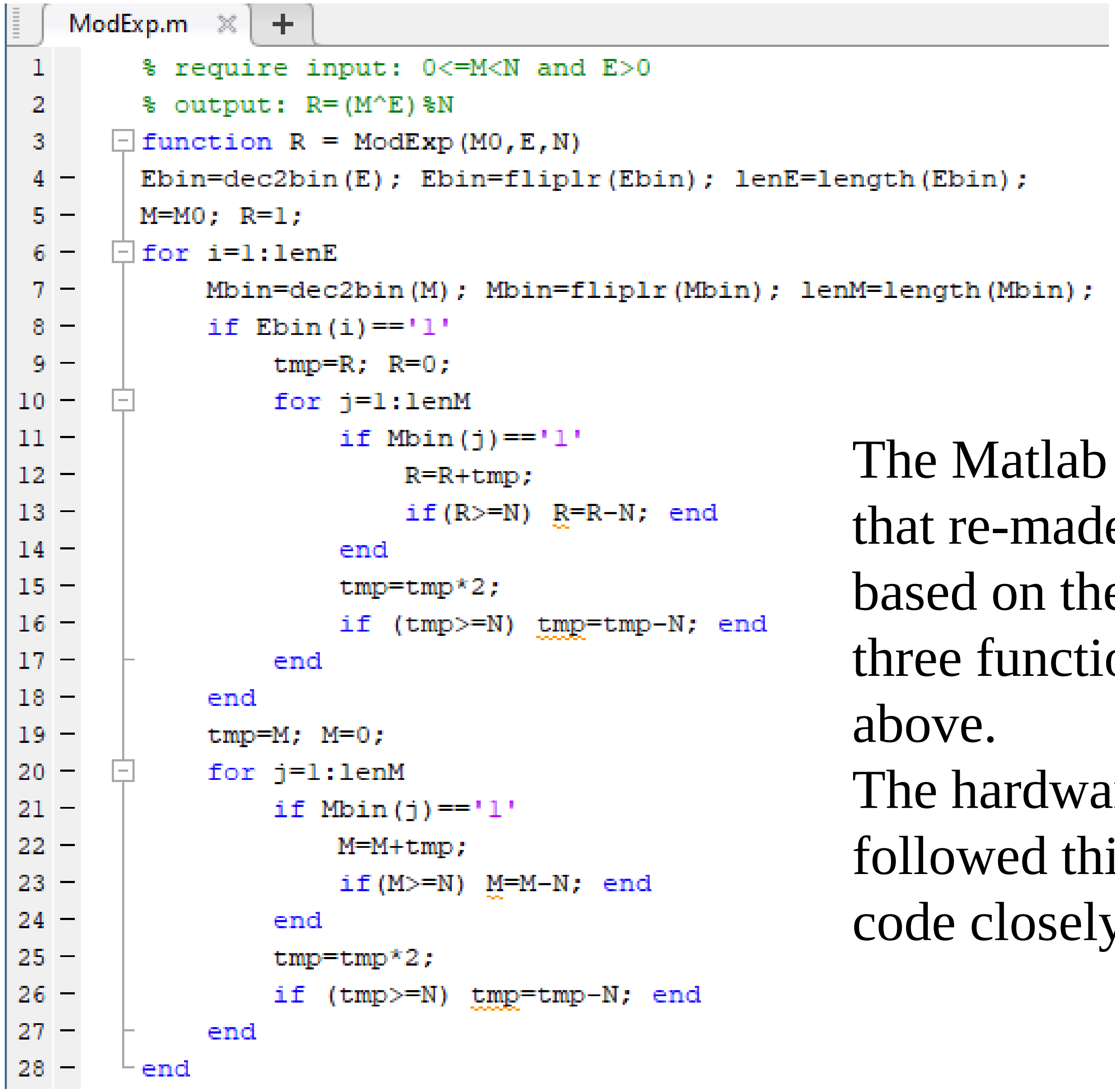
RSA is considered a costly algorithm for hardware designers → we were aiming for an area-effective architecture.



ModExp.m | MulMod.m | ModSpecial.m

```
% require input: 0<A<2N
% output: R=A%N
function R = ModSpecial(A,N)
    if A<N
        R = A;
    else
        R = A-N;
    end
end
```

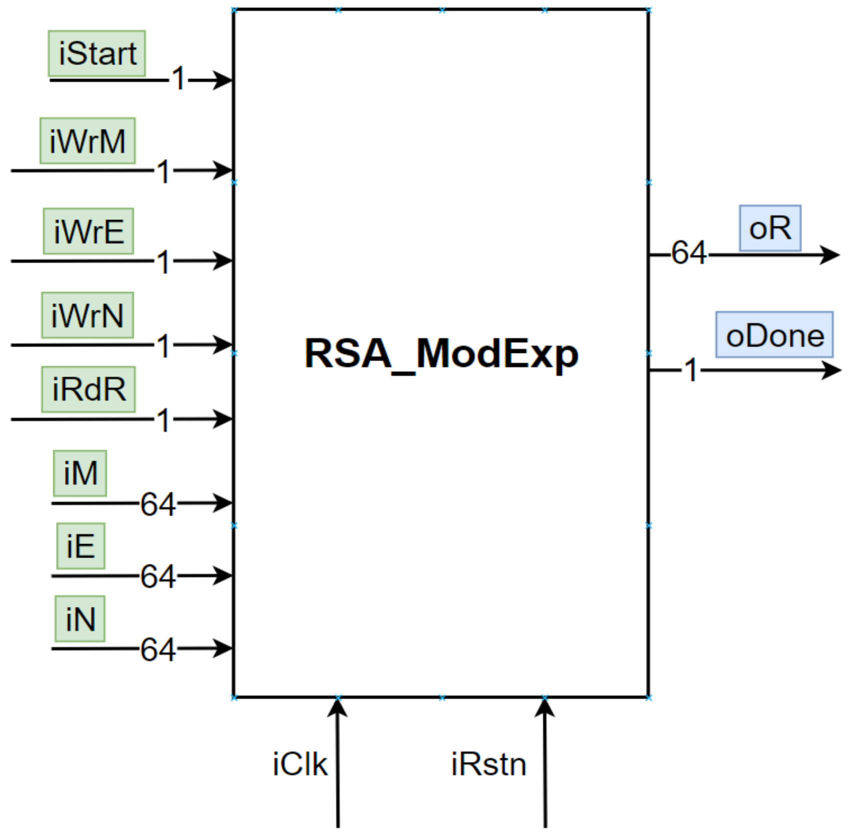
4. Crypto-key scheme: RSA (5/6) Design overview



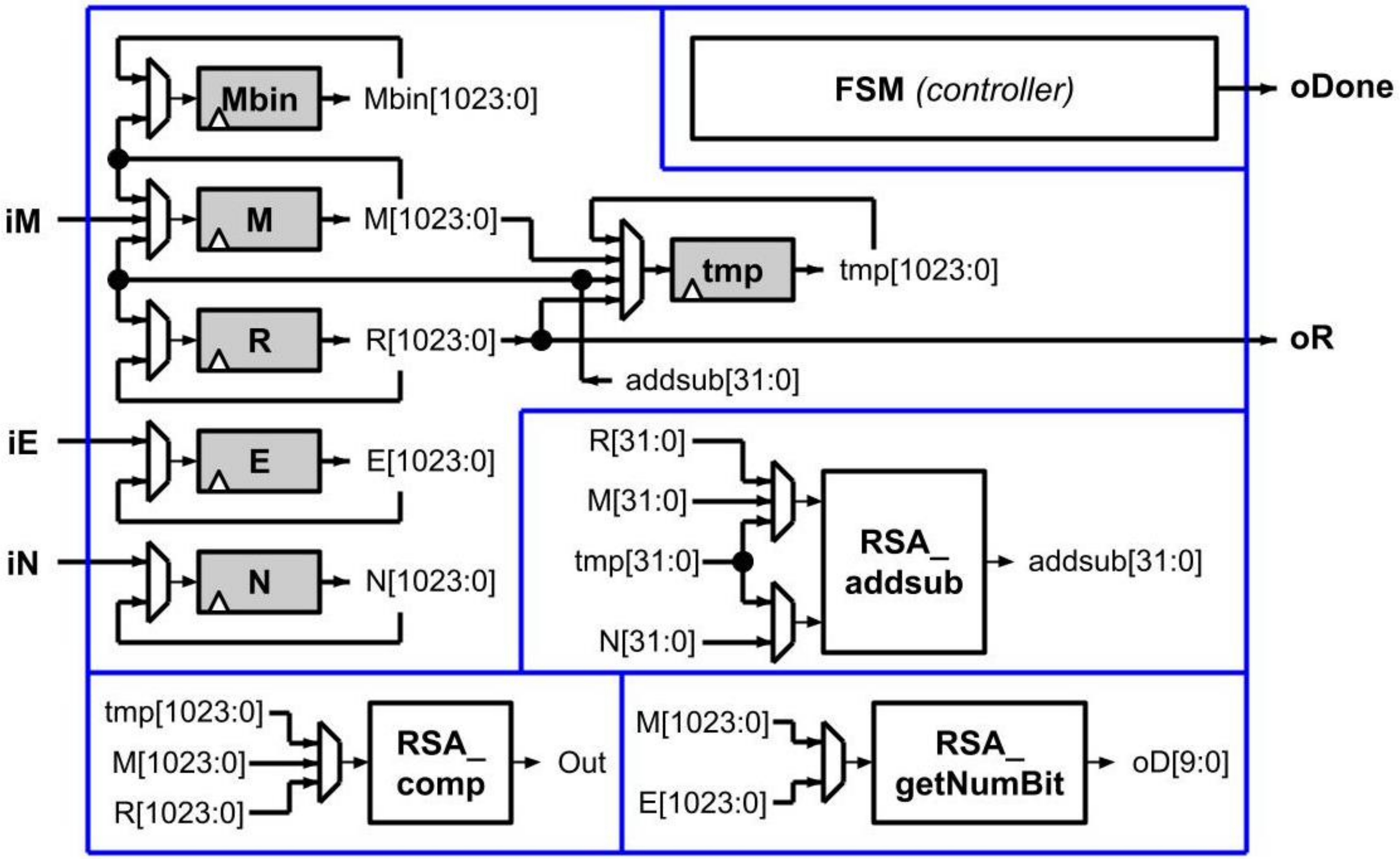
```
% require input: 0<=M<N and E>0
% output: R=(M^E)%N
function R = ModExp(M0,E,N)
Ebin=dec2bin(E); Ebin=fliplr(Ebin); lenE=length(Ebin);
M=M0; R=1;
for i=1:lenE
    Mbin=dec2bin(M); Mbin=fliplr(Mbin); lenM=length(Mbin);
    if Ebin(i)=='1'
        tmp=R; R=0;
        for j=1:lenM
            if Mbin(j)=='1'
                R=R+tmp;
                if(R>=N) R=R-N; end
            end
            tmp=tmp*2;
            if (tmp>=N) tmp=tmp-N; end
        end
    end
    tmp=M; M=0;
    for j=1:lenM
        if Mbin(j)=='1'
            M=M+tmp;
            if(M>=N) M=M-N; end
        end
        tmp=tmp*2;
        if (tmp>=N) tmp=tmp-N; end
    end
end
```

The Matlab code that re-made based on the three functions above.
The hardware followed this code closely.

Top view of the RSA module.
Function: $oR = iM^{iE} \ mod \ iN$
Note: RSA Genkey is not included.



4. Crypto-key scheme: RSA (6/6) Design overview



RSA module block diagram:

- Six 1024-bit registers are used to store computational values.
- Three submodules, addsub, comp, and getNumBit are used for $\pm$, $<$, and get the number of meaning LSBs. The addsub submodule operates on 32-bit at a time $\rightarrow$ trade-off speed for area cost.
- Finally, an FSM is used as the controller for all the activities in the module.

国立大学法人
電気通信大学
The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

Outline

5. Practice (1/1)

Exercise 1:

Add the given **SHA3**, **AES**, and **RSA** modules to the existing Rocket computer system.

THANK YOU

Tháng 9/2023