Nial Stewart Developments Ltd

FPGA & High Speed Digital Design

FPGA Lock IP
User Guide

Rev 2.0

1. Overview

FPGAs are much more flexible than ASICs, they are however much more difficult to secure. External configuration devices mean PCBs are easily copied. IP has also been difficult to license, especially for evaluation purposes. Bitstream encryption is available in some devices but often not in high volume lower cost devices.

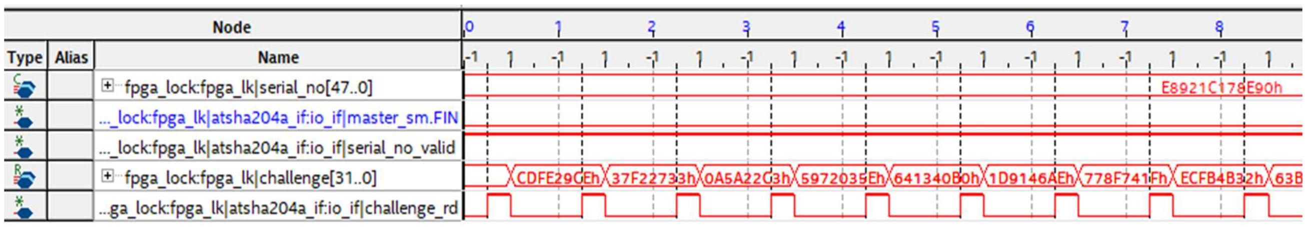
The **FPGA Lock** core enables FPGA images, IP and hardware to be secured with a tiny cheap external device, one IO pin and a small logic footprint.

The core works with the Microchip ATSHA204A. This performs a NIST SHA-256 hash with a user programmed 256 bit secret key, a core generated 256 bit random "challenge" and the device's serial number, then returns the result. The core reads the devices serial number then performs the same hash with the serial number, challenge and secret key internally. If the ATSHA204A generated results match the internally calculated values the core confirms a properly configured device is present. If a correctly programmed device is not present FPGA functionality can be disabled.

The ATSHA204A variant used is a three pin SOT-23 device, the interface to it is driven by a single FPGA pin. The device operates with a 2.0V to 5.5V supply, communications voltage 1.8V to 5.5V.

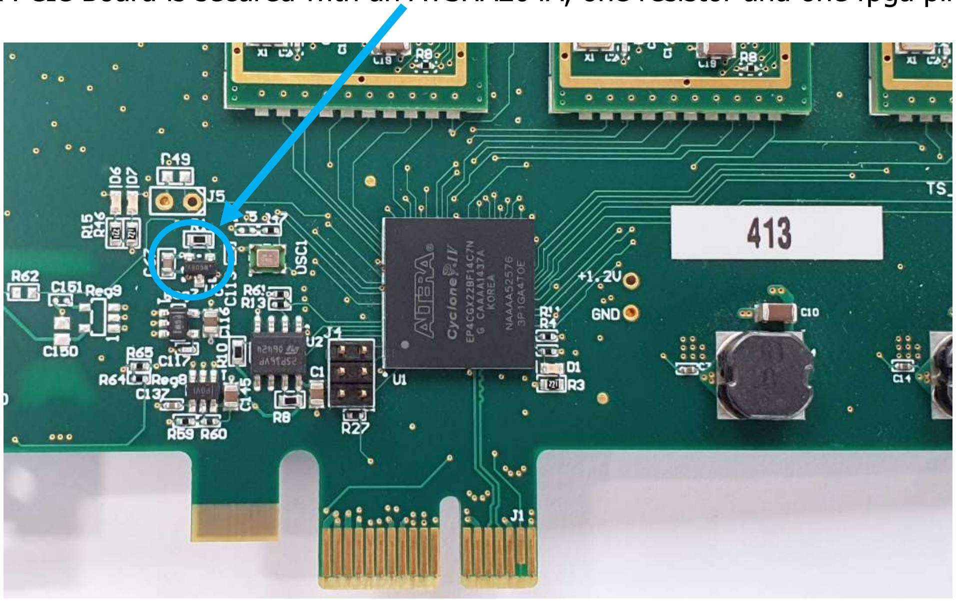
The devices are guaranteed to have a unique 72 bit ID, 24 bits are static so the 48 variable ID bits are reported by the core when a hash test is triggered. This can also be used for inventory control.

A random 256 bit number is generated for each challenge, the target device serial number is also used in the hash. One device's serial number and the random challenges for a hash check are captured below.

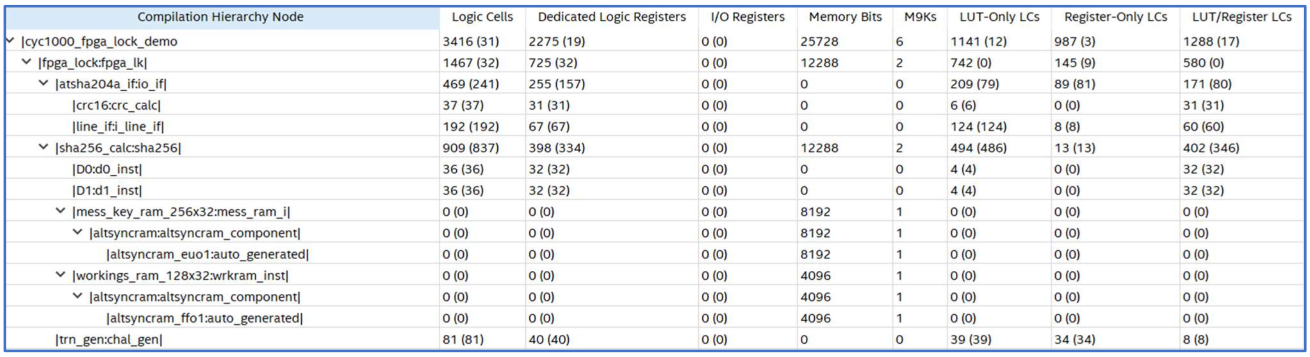


Type	Alias	Name	0 – 8
		fpga_lock:fpga_lk|serial_no[47..0]	E8921C178E90h
		..._lock:fpga_lk|atsha204a_if:io_if|master_sm.FIN	
		..._lock:fpga_lk|atsha204a_if:io_if|serial_no_valid	
		fpga_lock:fpga_lk|challenge[31..0]	CDFE29CEh, 37F22733h, 0A5A22C3h, 5972035Eh, 641340B0h, 1D9146AEh, 778F741Fh, ECFB4B32h, 63B
		...ga_lock:fpga_lk|atsha204a_if:io_if|challenge_rd	

A PCIe Board is secured with an ATSHA204A, one resistor and one fpga pin.



The core is designed to be compact and use as few FPGA resources as possible (< 750 registers, 2 block rams) but be easily placed and routed to meet design constraints. The Quartus resources summary from a design targeted at a CYC1000 evaluation board is shown here. Note the design also includes an instance of the **KEY Writer** and a signal tap instance.



Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M9Ks	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
|cyc1000_fpga_lock_demo	3416 (31)	2275 (19)	0 (0)	25728	6	1141 (12)	987 (3)	1288 (17)
|fpga_lock:fpga_lk|	1467 (32)	725 (32)	0 (0)	12288	2	742 (0)	145 (9)	580 (0)
|atsha204a_if:io_if|	469 (241)	255 (157)	0 (0)	0	0	209 (79)	89 (81)	171 (80)
|crc16:crc_calc|	37 (37)	31 (31)	0 (0)	0	0	6 (6)	0 (0)	31 (31)
|line_if:i_line_if|	192 (192)	67 (67)	0 (0)	0	0	124 (124)	8 (8)	60 (60)
|sha256_calc:sha256|	909 (837)	398 (334)	0 (0)	12288	2	494 (486)	13 (13)	402 (346)
|D0:d0_inst|	36 (36)	32 (32)	0 (0)	0	0	4 (4)	0 (0)	32 (32)
|D1:d1_inst|	36 (36)	32 (32)	0 (0)	0	0	4 (4)	0 (0)	32 (32)
|mess_key_ram_256x32:mess_ram_i|	0 (0)	0 (0)	0 (0)	8192	1	0 (0)	0 (0)	0 (0)
|altsyncram:altsyncram_component|	0 (0)	0 (0)	0 (0)	8192	1	0 (0)	0 (0)	0 (0)
|altsyncram_euo1:auto_generated|	0 (0)	0 (0)	0 (0)	8192	1	0 (0)	0 (0)	0 (0)
|workings_ram_128x32:wrkram_inst|	0 (0)	0 (0)	0 (0)	4096	1	0 (0)	0 (0)	0 (0)
|altsyncram:altsyncram_component|	0 (0)	0 (0)	0 (0)	4096	1	0 (0)	0 (0)	0 (0)
|altsyncram_ffo1:auto_generated|	0 (0)	0 (0)	0 (0)	4096	1	0 (0)	0 (0)	0 (0)
|trn_gen:chal_gen|	81 (81)	40 (40)	0 (0)	0	0	39 (39)	34 (34)	8 (8)

The **KEY Writer** design is provided as part of the FPGA Lock IP package. This allows users to quickly program and lock their custom secret key on their PCBs.

In a target design, delaying the trigger of the hash check for a period could allow CEMs to build and test designs with full functionality. When the assembled/ tested boards are received the ATSHA204A can be programmed in situ with a quickly loaded FPGA image with the **Key Writer** module to permanently enable functionality after power on.

The secret key can be programmed in the ATSHA204A at the factory (contact Microchip for details), or in a stand-alone programmer.

The FPGA Lock and Key Writer are provided in clear VHDL to allow user verification of operation.

Demonstration

A quick 2 minute demonstration of the FPGA Lock and Key Writer functionality can be seen here…

https://youtu.be/yH0eSVunF2M

The ATSHA204A devices are delivered with a default set of keys. In the demonstration a test is done against a fresh device. This fails to match the expected secret key hash result so the ‘enable’ output goes inactive. The secret key is then programmed and another test is performed, this time the result matches and the ‘enable’ output remains active.

2. Implementation

2.1 FPGA Lock

2.1.1 FPGA Lock Port Descriptions

Port	Type	In/Out	Description
sim	Generic		Set to '0'
bit_period	Generic		Number of System clock cycles in 4.45us (rounded up)
clk	std_logic	In	System clock
rst	std_logic	In	Active high reset synchronous to System clock
dio	std_logic	Inout	Bi-directional IO Pin interface to the ATSHA204A.
go	std_logic	In	Single system clock width trigger to send the challenge and do a hash test.
serial_no	std_logic_vector(47..0)	Out	48 Bit ATSHA204A Serial number
serial_no_valid	std_logic	Out	Flag that is set when the serial number has been read and is valid.
enable	std_logic	Out	Enable output to the FPGA, this is the result of a test of the ATSHA204A. This is reset to '1' so functionality defaults to enabled. When the check is completed if it fails this is driven to '0'.
hash_tested	std_logic	Out	Flag that the challenge and hash comparison has completed.
timeout_error	std_logic	Out	No device has been detected.
glitch_error	std_logic	Out	A start bit has been detected from the ATSHA204A but when checked at a half bit period it is no longer present.

2.1.2 FPGA Lock Instantiation

The FPGA Lock is instantiated as follows…

```
fpga_lk:entity work.fpga_lock
generic map
  (   sim              => '0', -- in std_logic := '0';
      bit_period       => "0110111101") -- in std_logic_vector(9 downto 0))  -- # Clocks 
for 4.45 us
port map (
      clk              => clk,              -- in std_logic;
      rst              => rst,              -- in std_logic;
      go               => fpga_lock_go,     -- in std_logic;
      dio              => fpga_lock_dio,    -- inout std_logic;
      serial_no        => serial_no,        -- out std_logic_vector(47 downto 0);
      ser_no_valid     => serial_no_valid, -- out std_logic;
      enable           => enable        ,   -- out std_logic;
      hash_tested      => hash_tested  ,    -- out std_logic;
      timeout_error    => timeout_error,    -- out std_logic;
      glitch_error     => glitch_error      -- out std_logic;
      );
```

2.2 Key Writer

2.2.1 Key Writer Port Descriptions

Port	Type	In/Out	Description
sim	Generic		Set to '0'
bit_period	Generic		Number of System clock cycles in 4.45us (rounded up)
clk	std_logic	In	System clock
rst	std_logic	In	Active high reset synchronous to System clock
dio	std_logic	Inout	Bi-directional IO Pin interface to the ATSHA204A.
go	std_logic	In	Single system clock width trigger to start the key_write process
key_written	std_logic	Out	Flag that the key write process has completed.
serial_no	std_logic_vector(47..0)	Out	N/A
serial_no_valid	std_logic	Out	N/A
enable	std_logic	Out	N/A
hash_tested	std_logic	Out	N/A
timeout_error	std_logic	Out	N/A
glitch_error	std_logic	Out	N/A

2.2.2 Key Writer Instantiation

```
key_wr:entity work.key_writer
generic map
      (sim              => '0',           -- in std_logic := '0';
      bit_period        => "0110111101")  -- in std_logic_vector(9 downto 0));
port map (
      clk               => clk,           -- in std_logic;
      rst               => rst,           -- in std_logic;
      go                => key_wr_go,     -- in std_logic;
      key_written       => key_wr_done,   -- out std_logic;
      dio               => key_wr_dio,    -- inout std_logic;
      serial_no         => open,
      ser_no_valid      => open,
      enable            => open,
      hash_tested       => open,
      timeout_error     => open,
      glitch_error      => open,
      debug             => open
      );
```

2.3 File list

2.3.1 FPGA Lock

For **Intel** designs add the following files to the FPGA project.

fpga_lock.vhd
atsha204a_if.vhd
crc16.vhd
trn_gen.vhd
line_if.vhd
sha256_calc.vhd
mess_key_ram_256x32.vhd
workings_ram_128x32.vhd

For **Xilinx** designs add
fpga_lock.vhd
atsha204a_if.vhd
crc16.vhd
trn_gen.vhd
line_if.vhd
sha256_calc.vhd
inferable_dp_ram_1clk.vhd
xil_mess_key_ran_256x32.vhd
xil_workings_ram_128x32.vhd

2.3.2 Key Writer

For **Intel** designs add the following files to the FPGA project.

key_writer.vhd
key_wr_atsha204a_if.vhd
crc16.vhd
line_if.vhd
mess_key_ram_256x32.vhd

For **Xilinx** designs add

key_writer.vhd
key_wr_atsha204a_if.vhd
crc16.vhd
line_if.vhd
inferable_dp_ram_1clk.vhd
xil_mess_key_ran_256x32.vhd

2.4 Constraints

If the system clock is defined and the trigger inputs are synchronous the only thing that needs constrained is the bidirectional dio pin. The communication 'bit period' is 445uS so only a very loose constraint is needed. The following creates a virtual clock and applies a 500ps input and output delay constraint. These constraints work in both Quartus and Vivado and should be added to your design's *.sdc or *.xdc file.

```
create_clock -name virt_inout_clk -period 1000
set_clock_groups -asynchronous -group [get_clocks virt_inout_clk]

set_output_delay -clock [get_clocks virt_inout_clk] 500.000 [get_ports fpga_lock_io]
set_input_delay -clock [get_clocks virt_inout_clk] 500.000 [get_ports fpga_lock_io]
set_output_delay -clock [get_clocks virt_inout_clk] 500.000 [get_ports key_wr_io]
set_input_delay -clock [get_clocks virt_inout_clk] 500.000 [get_ports key_wr_io]
```

With Xilinx designs the following must be added to the *.XDC to allow the combinatorial loops in the random number generator module (change the hierarchy as appropriate).

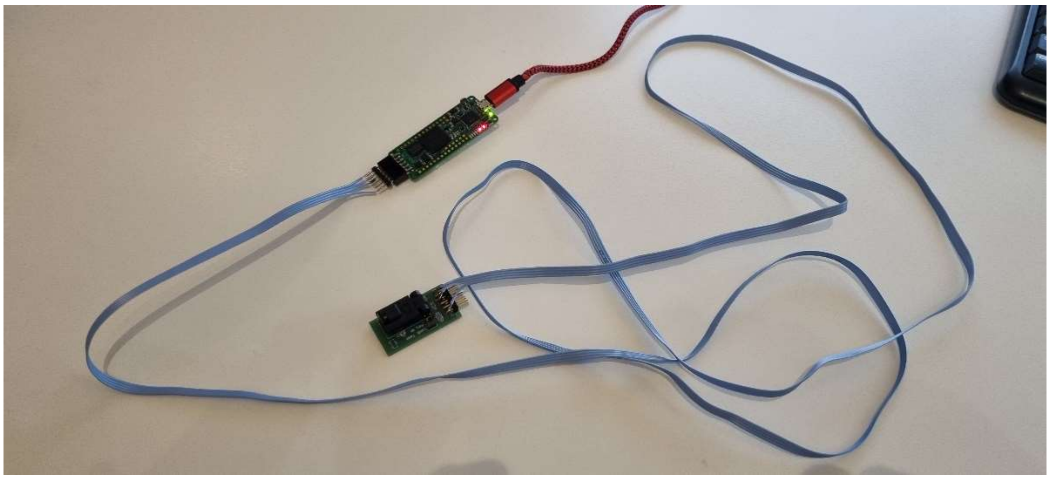
```
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets {fpga_lk/real_challenge.chal_gen/ring5[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets {fpga_lk/real_challenge.chal_gen/ring7[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets {fpga_lk/real_challenge.chal_gen/ring9[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets {fpga_lk/real_challenge.chal_gen/ring11[0]}]
set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets {fpga_lk/real_challenge.chal_gen/ring13[0]}]
```

2.5 Signal Integrity

Communications with the ATSHA204A is via an 'open drain' interface with an external pull up.

Communication is relatively slow with a bit period of 4.45us however it is the user's responsibility to ensure the integrity of the link between the FPGA and ATSHA204A.

A successful hash test down 2M of ribbon cable, this was a 3.3V link with a 2.2K pullup.



2.6 Setting a Custom 256 Bit Key

The simulations and example builds in the delivered IP all use a secret key of 0xA5A5A5A51111111122222222333333334444444455555555666666667777777

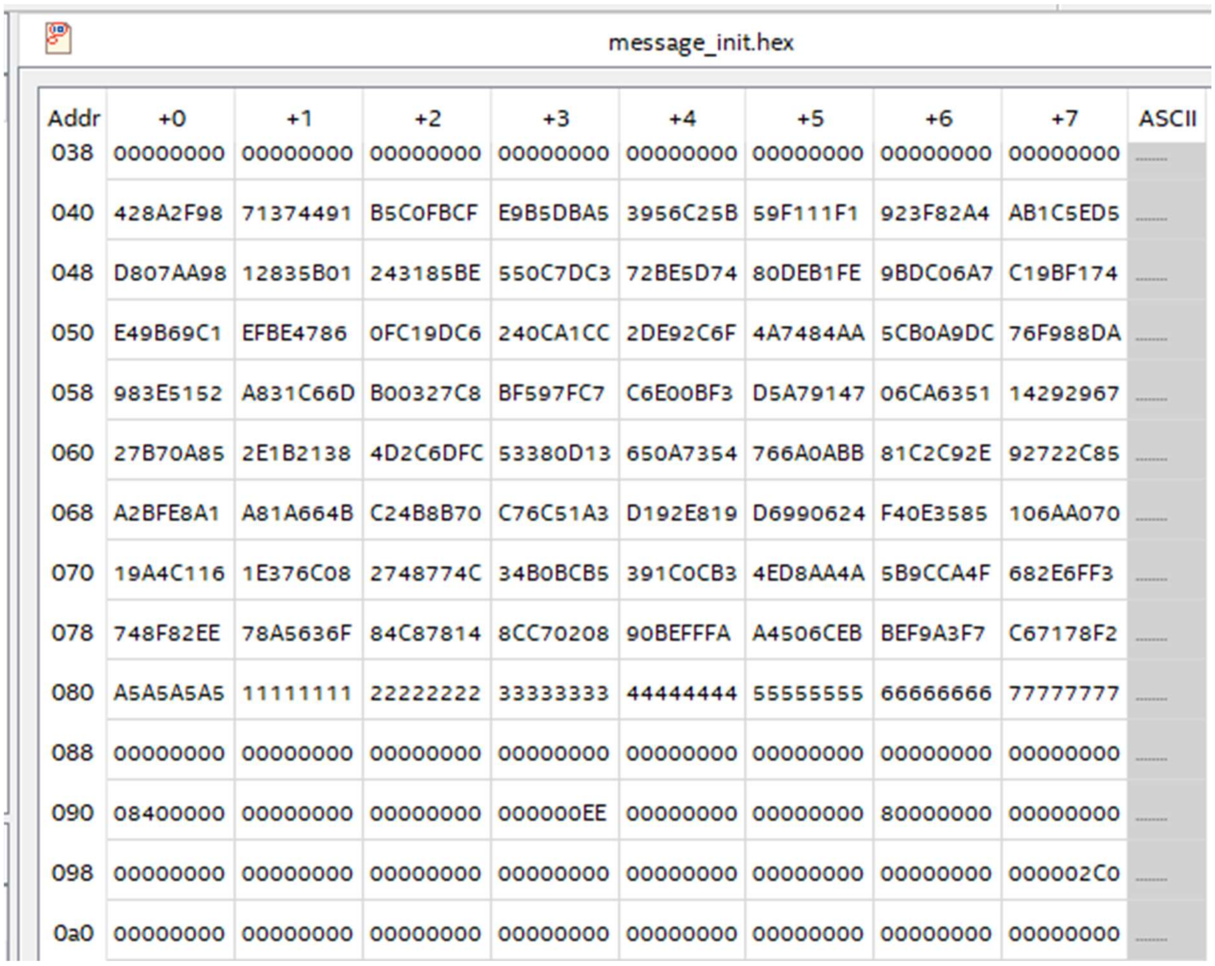
For user implementation a unique 256 bit secret key should be chosen.

2.6.1 Intel Designs

In both the FPGA_Lock and Key_Writer designs the secret key is defined in "message_init.hex".

In Quartus, File-> Open then change the file type selection to 'All Files'. Navigate to the design\vhdl directory, select the "messge_init.hex" then when prompted set the Word Size to 32.

Quartus displays the file contents as below..



message_init.hex

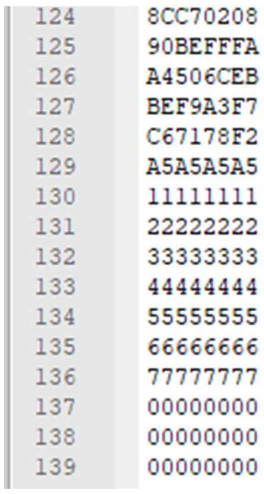
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
038	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
040	428A2F98	71374491	B5C0FBCF	E9B5DBA5	3956C25B	59F111F1	923F82A4	AB1C5ED5	....
048	D807AA98	12835B01	243185BE	550C7DC3	72BE5D74	80DEB1FE	9BDC06A7	C19BF174	....
050	E49B69C1	EFBE4786	0FC19DC6	240CA1CC	2DE92C6F	4A7484AA	5CB0A9DC	76F988DA	....
058	983E5152	A831C66D	B00327C8	BF597FC7	C6E00BF3	D5A79147	06CA6351	14292967	....
060	27B70A85	2E1B2138	4D2C6DFC	53380D13	650A7354	766A0ABB	81C2C92E	92722C85	....
068	A2BFE8A1	A81A664B	C24B8B70	C76C51A3	D192E819	D6990624	F40E3585	106AA070	....
070	19A4C116	1E376C08	2748774C	34B0BCB5	391C0CB3	4ED8AA4A	5B9CCA4F	682E6FF3	....
078	748F82EE	78A5636F	84C87814	8CC70208	90BEFFFA	A4506CEB	BEF9A3F7	C67178F2	....
080	A5A5A5A5	11111111	22222222	33333333	44444444	55555555	66666666	77777777	....
088	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
090	08400000	00000000	00000000	000000EE	00000000	00000000	80000000	00000000	....
098	00000000	00000000	00000000	00000000	00000000	00000000	00000000	000002C0	....
0a0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....

The secret key is stored in addresses 0x80 to 0x87, this can be modified and saved to whatever is required. Care should be taken not to modify any other values in this file.

2.6.2 Xilinx Designs

In both the FPGA_Lock and Key_Writer designs the secret key is defined in the plain text files "xil_message_init.hex".

The secret key is defined in the text file between lines 129 and 136. This should be set to whatever value the user has chosen.



```
124    8CC70208
125    90BEFFFA
126    A4506CEB
127    BEF9A3F7
128    C67178F2
129    A5A5A5A5
130    11111111
131    22222222
132    33333333
133    44444444
134    55555555
135    66666666
136    77777777
137    00000000
138    00000000
139    00000000
```

Again, care should be taken not to modify any other values in this file.

Note: The same secret key must be set in the FPGA_Lock and FPGA_Writer designs for the FPGA hash test to work.

3. TestBench

A basic testbench is provided that demonstrates consecutive FPGA_Lock tests against two devices with the example 0xA5A5… key and a challenge of 8* x”11111111”.

The device IDs and returned HASH test results that the testbench drives are the signal tap results from two real ATSHA204A devices. In the FPGA_Lock module if the sim generic is set to ‘1’, the ‘random’ challenge is set to 8 x”11111111”.

When triggered the core wakes the ATSHA204A then reads its unique device ID. A Hash test is then triggered with a write of the random challenge and the parameters to be included in the hash (eg include the device ID). When the device returns the result each 32 bit word is compared to the internally calculated values and the Enable output de-asserted if these do not match.

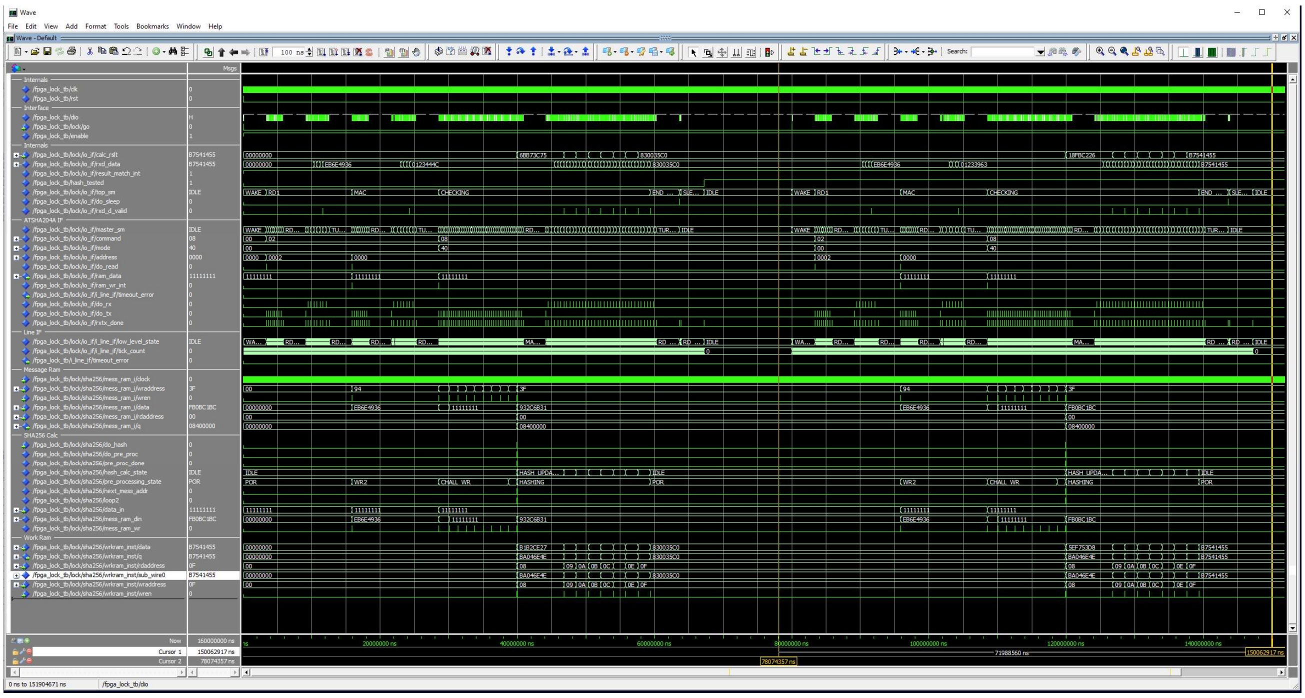
The testbench can be modified to return an incorrect value and the output will be disabled.

Scripts are provided to run the testbench with the Intel Starter version of Modelsim. To run the testbench….

3.1 Intel Testbench

Run up Modelsim
File -> Change Directory   to the FPGA_Lock/Modelsim directory
In the Transcript window type….
“source Intel_Compile_Run_FPGA_Lock_testbench.do”

A ‘work’ subdirectory should be created then the design and testbench compiled into it, then run with the wave window loaded with a range of pertinent signals. The test should run for 160ms.



The Intel version of Modelsim has the altera_mf library pre-compiled, if using a different version this library must be compiled from the Quartus source and mapped to in modelsim.ini

3.2 Xilinx Testbench

Run up Modelsim
File -> Change Directory   to the FPGA_Lock/Modelsim directory
In the Transcript window type
“source Xilinx_Compile_Run_FPGA_Lock_testbench.do”

A ‘work’ subdirectory should be created then the design and testbench compiled into it, then run with the wave window loaded with a range of pertinent signals. The test should run for 160ms.

4. Example Builds

Two very basic example Intel and Xilinx designs are provided to demonstrate the implementation of the FPGA_Lock and Key_Writer designs.

The top level design files are…
CYC1000_FPFA_Lock_Demo\vhdl\cyc1000fpga_lock_demo.vhd
Artix7_FPGA_Lock_Demo\vhdl\artix7_fpga_lock_demo.vhd

Examining these will show how easily the FPGA Lock and Key Writer modules are included in a design.

The Project files are….
CYC1000_FPFA_Lock_Demo\Intel\cyc1000fpga_lock_demo.qpf
Artix7_FPGA_Lock_Demo\Artix7_FPGA_Lock_Demo\Artix7_FPGA_Lock_Demo.xpr

The projects are targeted at low end devices and use Quartus Lite 18.1 and Vivado 2018.3 so they can be opened and rebuilt by more recent versions of the tools.

4.1 Intel

An example design is provided targeting the Cyclone 10 on Arrow CYC1000 development board. The contains the **FPGA Lock** and **Key Writer** modules but they drive separate IO pins as they would in separate implementations.

The FPGA Lock IO is on FPGA pin D16, this is connected to the PMOD header D3, the Key Writer IO to FPGA pin F16, PMOD-D2.

PMOD Header J8 looking into the connector

3.3V	**GND**	**D3** FPGA Lock IO	**D2** Key Writer IO	**D1** Sel Key Wr	**D0** Sel Lock Test
3.3V	**GND**	**D7**	**D6**	**D5**	**D4**

LEDs
LED0 – (Nearest the USB connector) – FPGA Functionality Enable/ result of FPGA Lock Test
LED2 – Lock Test Completed
LED4 – Timeout error
LED6 – Glitch Error
LED7 – Key Write Done

User Pushbuttons
PB3 – (Nearest the PMOD connector) – Do FPGA Lock Test
PB2 – Do Key Write

There is only one user button on this board so two inputs are provided to select whether a Lock Test or Key Write will be triggered.

Implementing both cores in the design allows the secret key to be programmed in an ATSHA204A, then a hash test done to confirm it has been correctly configured. This arrangement could be used in production.

4.2 Xilinx

An example design is provided targeting the Artix 7 on the Digilent Arty Development board, this contains both the **FPGA Lock** and **KEY Writer** modules.

The FPGA Lock IO is on FPGA pin V11, this is connected to the PMOD header JC-D3, the Key Writer IO to FPGA pin V10, PMOD JC-D2.

PMOD Header JC looking into the connector

3.3V	**GND**	**D3** FPGA Lock IO	**D2** Key Writer IO	**D1**	**D0**
3.3V	**GND**	**D7**	**D6**	**D5**	**D4**

The FPGA Lock test is triggered by BTN0 (furthest away from the USB connector)
A Key Write is triggered by BTN1.

LD4 - FPGA Functionality Enable/ result of FPGA Lock Test
LD5 – Lock Test Completed
LD6 – Timeout Error
LD7 – Glitch Error

LD0 – Goes red -> green to show the Key Write has been performed.

5. Purchase

Licensing is on a 'site' basis, the purchase of a site licence will allow the use of the core in any projects without restriction.

Please contact nial@nialstewartdevelopments.co.uk for full licensing details and with any queries about purchasing the core.

User Guide Changelog

Version	Date	Changes
1.0	09/05/2023	First Customer Release
2.0	10/01/2024	Inferrable rams used for the Xilinx build. Improved Key Writer implementation description Constraints updated Testbench scripts improved Example design descriptions updated