PRESENT DECODER
documentation

Krzysztof Gajewski
and opencores.org

www.opencores.org

gajos@opencores.org

Change History

Rev.	Chapter	Date	Description	Reviewer
0.1	all	2014/05/25	First draft	K. Gajewski

Contents

1 Introduction

Present is "ultra-lightweight" block cipher developed by A. Bogdanov et al. and proposed in 2007 [1]. It uses 64 bit data block and 80 bit or 128 bit key. This cipher consists of 32 rounds, during which:

- round key is added to plaintext
- plaintext goes through sBoxes (substitution boxes)
- plaintext after sBoxes goes through pLayer (permutation layer)
- round key is updated

After that, ciphertext feeds out the output. Briefly algorithm was shown in Fig. 1. In subprojects

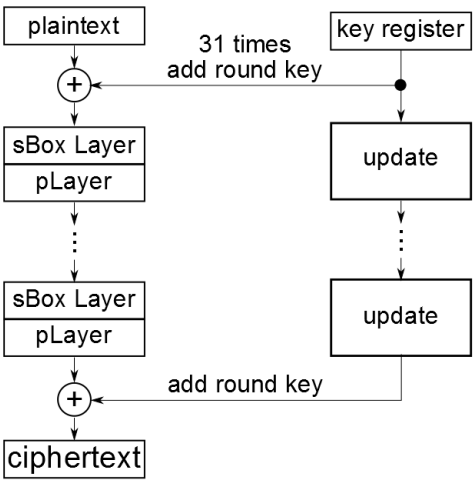

Figure 1: Briefly block scheme of the PRESENT block cipher

`Pure` and `PureTesting` Present coder components was presented. In this project Present decoder was presented. Decoding key is firstly generated, basing on the key used for data coding. Next, input data are decoded (taking into account "inverse" direction to the presented in Fig. 1), and at last feeds the output. This core works with 80 bit key. Target was Xilinx® Spartan 3E XC3S500E [2] on Spartan 3E Starter Board [3] made by Digilent®.

2 Interface

Top level component of Present decoder was shown in Fig. 2. All inputs and outputs are synchronous except `reset` signal and sampled at rising edge of the clock. Type for all signals is `STD_LOGIC` or `STD_LOGIC_VECTOR`.

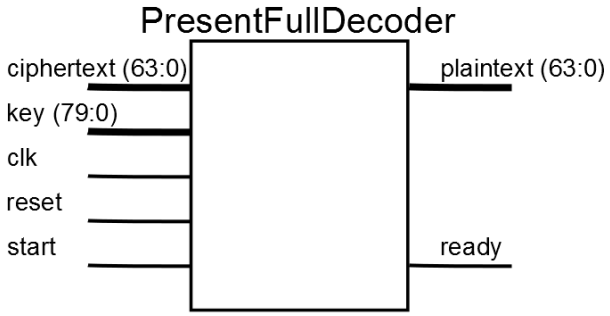

Figure 2: Top level component of Present decoder

Signal name	Width	In/Out	Description
`ciphertext`	64	in	input data which have to be decoded.
`key`	80	in	secret key used for input data decoding (the same which was used for data encoding).
`clk`	1	in	clock signal for the component
`reset`	1	in	*Asynchronous* reset signal.
`start`	1	in	signal which starts decoding process.
`plaintext`	64	out	decoded text output.
`ready`	1	out	signal informing about end of decoding process. "0" - wait until end of data decoding. "1" - data at the `ciphertext` output are valid, you can read them.

Table 1: Input/Output signals of Present Decoder component

3 Internal structure and state machine workflow

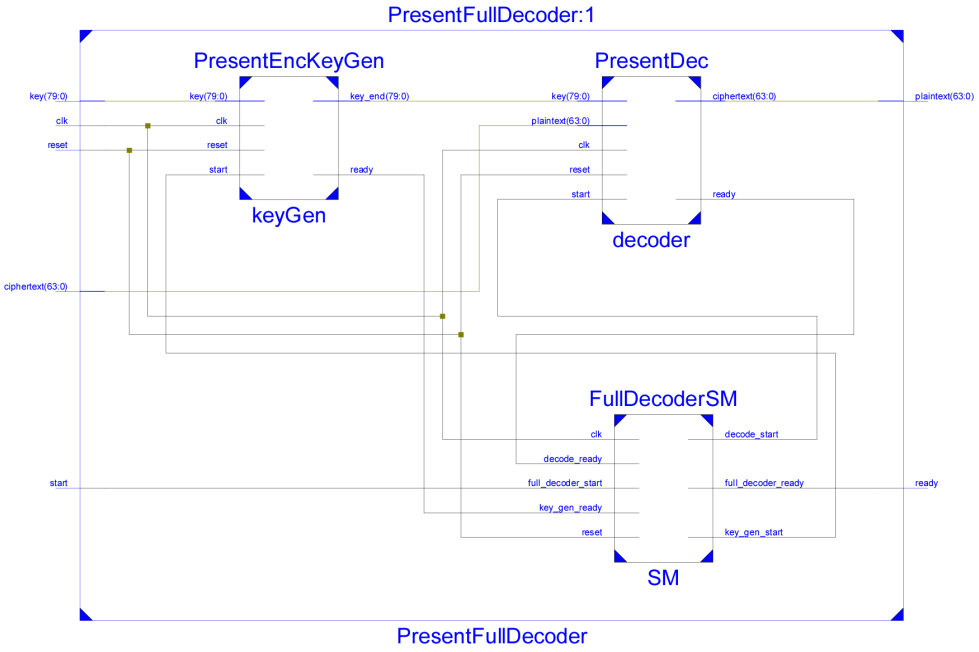

Figure 3: Internal datapath between main components in Present decoder.

Internal datapath between main components was shown in Fig. 3. They are responsible for:

- `PresentEncKeyGen` - key generator for decoding process. Before decoding stage, key need to be prepared to the 'appropriate value'. This value is signalled by `ready = '1'`. It is almost the same core as in `Present` subproject, but truncated from text encoding part.
- `PresentDec` - subcomponent responsible for ciphertext decoding. It is working in similar way as Present cipher, but is working in inverse way.
- `FullDecoderSM` - State machine controlling overall decoding process.

More information about cipher core and key generation process can be found in `./Present/doc/present_pure.pdf` file ("Present" subproject documentation).

State machine of the `PresentDec` component was shown in Fig. 4. It consist of three states `NOP`, `SM_START` and `READY`. The way of work of this state machine is the same as in the `Present` subproject, but the counter is counting down instead of counting up.

State machine of the `FullDecoderSM` component was shown in Fig. 5. It consist of four states `NOP`, `KG_START`, `DEC_START` and `READY`. `NOP` is default state after resetting the core. This state is active as long as `full_decoder_start` = '0'.

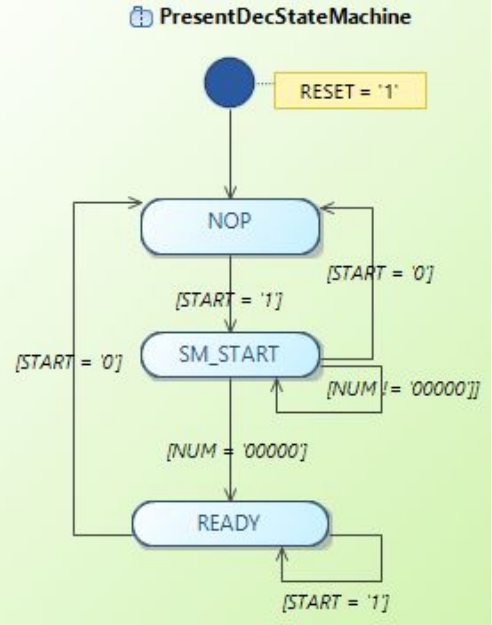

Figure 4: State machine of the Present component

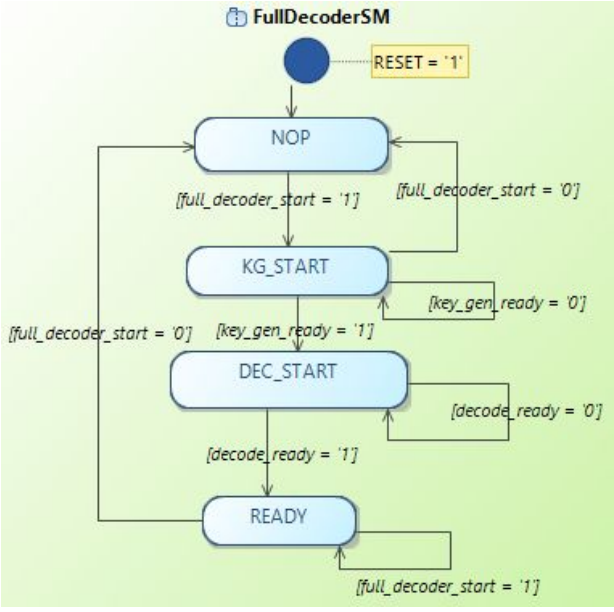

Figure 5: State machine of the Present decoder in main component.

When `full_decoder_start` = '1' key generation for the decoding process starts. Proper `key` and `ciphertext` must feed the input before. `KG_START` state is active as long `PresentEncKeyGen` is generating the key. Key generation ends, when `PresentEncKeyGen` sets the `ready` signal to '1'. When the `ready` signal is set to '1', the state is changing.

During DEC_START state decoding process appears. State machine is in this state until PresentDec ends its works. The end of decoding is signalled by setting the ready signal to '1' by the PresentDec component. Then, the state is changing.

READY sets the ready signal of the PresentFullDecoder to '1'. It is idle-like state, when the user can reads the output of the Present decoder. The state machine is in this state until the user sets the full_decoder_start to '0'.

4 FPGA implementations

The component has only been verified on a Xilinx® Spartan 3E XC3S500E FPGA in FG320 package and synthesized with Xilinx ISE 14.2. Appropriate setup files was prepared with use of ISE Project Navigator, but Makefile scripts was also written. Suitable files was stored in `./Decode/syn/XC3ES500/` directory. Implementation in FPGA device was done in another sub-project called `DecodeTesting`. Makefile was tested in Windows 8 with use of Cygwin for 64-bit Windows.

Synthesis results was given in Fig. 4

Xilinx ®Spartan 3E XC3S500E FPGA in FG320 package			
Parameter	**Used**	**Available**	**Utilisation**
Number of Slices	354	4656	7%
Number of Slice Flip Flops	240	9312	2%
Number of 4 input LUTs	402	9312	4%
Number of bonded IOBs	212	232	91%
Number of GCLKs	1	24	4%
Minimum period	5.023ns	-	-
Maximum Frequency	199 MHz	-	-

Table 2: Synthesis results for Spartan 3E XC3S500E

Possible change in used FPGA device may be possible in steps given below[1]:

1. Copy `./Decode/syn/XC3ES500/` directory to another one like `./Decode/syn/YOUR_FPGA_SYMBOL/`
2. Go to `./Decode/syn/XC3ES500/` directory.
3. In `PresentEnc.xst` file modify the line `-p xc3s500e-5-fg320` to `-p YOUR_FPGA_SYMBOL`
4. In `Makefile` file modify the line `PLATFORM=xc3s500e-fg320-5` to `PLATFORM=YOUR_FPGA_SYMBOL`

[1] This solution was not tested and is based on my own observations. Additional care should be taken with *.UCF files. You can make this modifications on your own risk

5 Simulation

Self-checking test bench were provided to the components used for Present encoder. They are stored in `./Decode/bench/vhdl` directory. Suitable configuration files and Makefile used for running test bench was stored in `./Decode/sim/rtl_sim/bin` directory. Appropriate test vectors was taken from [1].

Makefile was prepared to make "manual run" of tests. If You want to perform it without gui, remove `-gui` option in Makefaile.

6 Troubleshooting

During work with Windows 8 64-bit and and Xilinx® ISE 64-bit some problems may occur:

1. Xilinx may be unable to open projects in Project Navigator.
2. When you run `make` in Cygwin and perform testbench it would be unable to open ISIM gui.
3. When you run ISIM gui (*.exe test bench file) it hangs out or anti virus protection opens.

To solve problems listed above you have to perform steps listed below:

1. You have to rename libraries `libPortabilityNOSH.dll` to `libPortability.dll` from `nt64` directories (http://www.gadgetfactory.net/2013/09/having-problems-installing-xilinx-ise-on-windows-8-64bit-here-is-a-fix-video-included/)
2. Firstly, install Cygwin X11 (http://stackoverflow.com/questions/9393462/cannot-launch-git-gui-using-cygwin-on-windows)
3. Temporary switch off anti virus protection.

7 License and Liability

References

[1] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe, “Present: An ultra-lightweight block cipher,” in *Cryptographic Hardware and Embedded Systems - CHES 2007*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds. Springer Berlin Heidelberg, 2007, vol. 4727, pp. 450–466. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74735-2_31

[2] Xilinx. (2014, Feb.) Spartan-3e fpga family data sheet. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[3] Digilent. (2014, Feb.) Spartan 3e starter board. [Online]. Available: http://www.digilentinc.com/Products/Detail.cfm?Prod=S3EBOARD