PRESENT DECODER ("DECODE TESTING") - WITH COMMUNICATION CHANNEL WITH PC

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

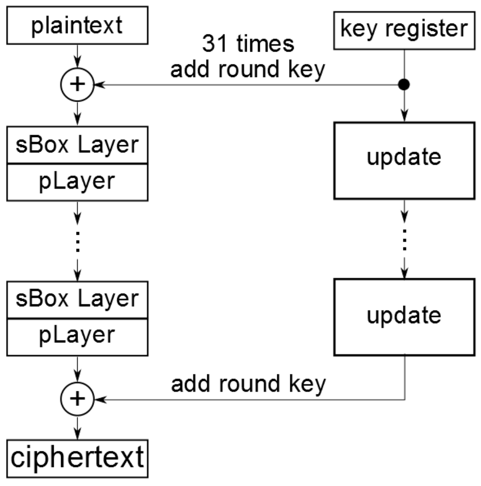

Figure 1: Briefly block scheme of the PRESENT block cipher

In subprojects `Pure` and `PureTesting` Present coder components was presented. In this project Present decoder was presented. It was attached by shifting registers to RS-232 core developed by Digilent® to enable communication with PC. 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 Decode Testing was shown in Fig. 2. The number of inputs and outputs was limited due to RS-232 component in communication interface. All inputs and outputs are synchronous except `reset` signal in state machines and sampled at rising edge of clock. All signals are `STD_LOGIC`.

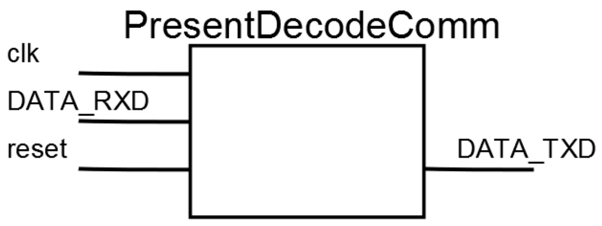

Figure 2: Top level component of Present Decode Testing

Signal name	Width	In/Out	Description
`clk`	1	in	Clock signal for the component.
`DATA_RXD`	1	in	Input data signal.
`reset`	1	in	*Asynchronous* / *Synchronous* reset signal.
`DATA_TXD`	1	out	Output data signal.

Table 1: Input/Output signals of Present Decode Testing component

3 Internal structure and state machine workflow

Internal datapath between components was shown in fig. 3. All control signals, `clk` and `reset` was omitted for clearance. In these schamatic `keyReg`, `textReg` and `outReg` are shift registers enabling conversion of the input/output serial data into parallel data. They are respectively:

- `keyReg` - shift register for the key used during decoding,
- `textReg` - shift register for the text to be decoded (to be more clearly `ciphertext`),
- `outReg` - shift register for the output data to be sendend by RS232 (to be more clearly `plaintext`).

`PresentFullDecoder` - is the decoding core. It was described in `./Decode/doc/present_decode.pdf` file ("Decode" subproject documentation). `RS232` is the serial communication core developed by Digilent® responsible for the communication with PC computer. `PresentDecodeCommSM` is state machine which manage communication with PC and data conversion before and after data decoding in `PresentFullDecoder` component.

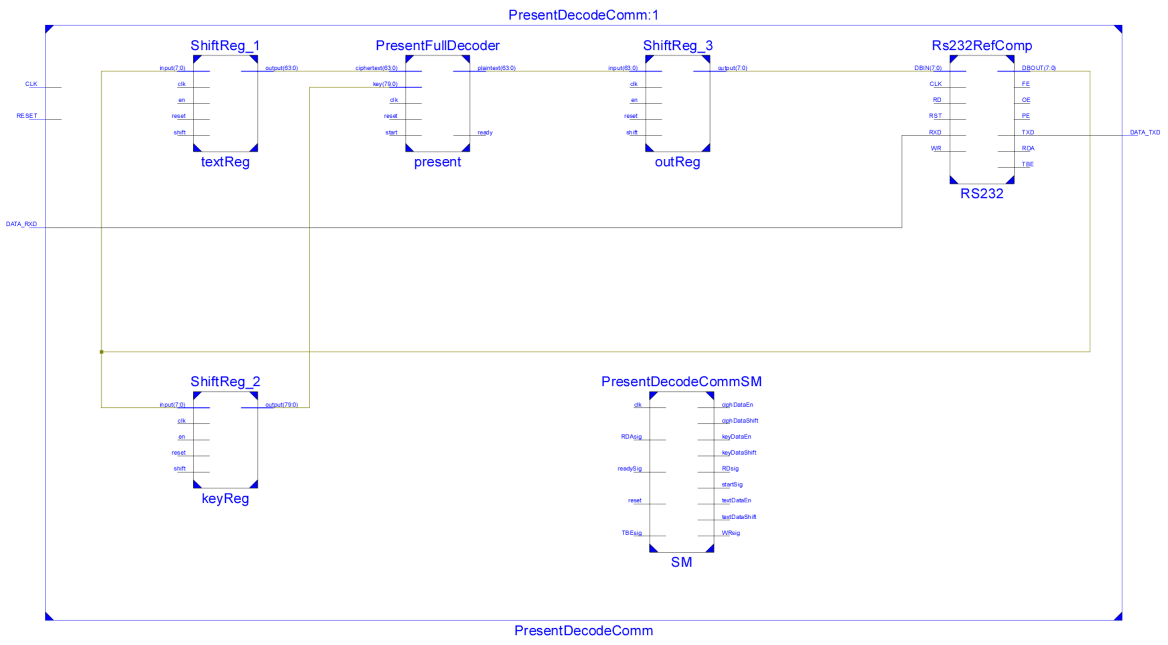

Figure 3: Internal structure of Present Decode core with communication environment.

State machine states and transition between them was shown in fig. 4.

In fact it is the same State machine as in Present `Pure Testing` subproject, but due to its length, will be reminded. State machine consist of following states:

- `NOP` - this is initial state of the state machine. It is set up after resetting the system. If any data appear in the RS-232 input (`RDAsig = '1'`), this state will be changed.

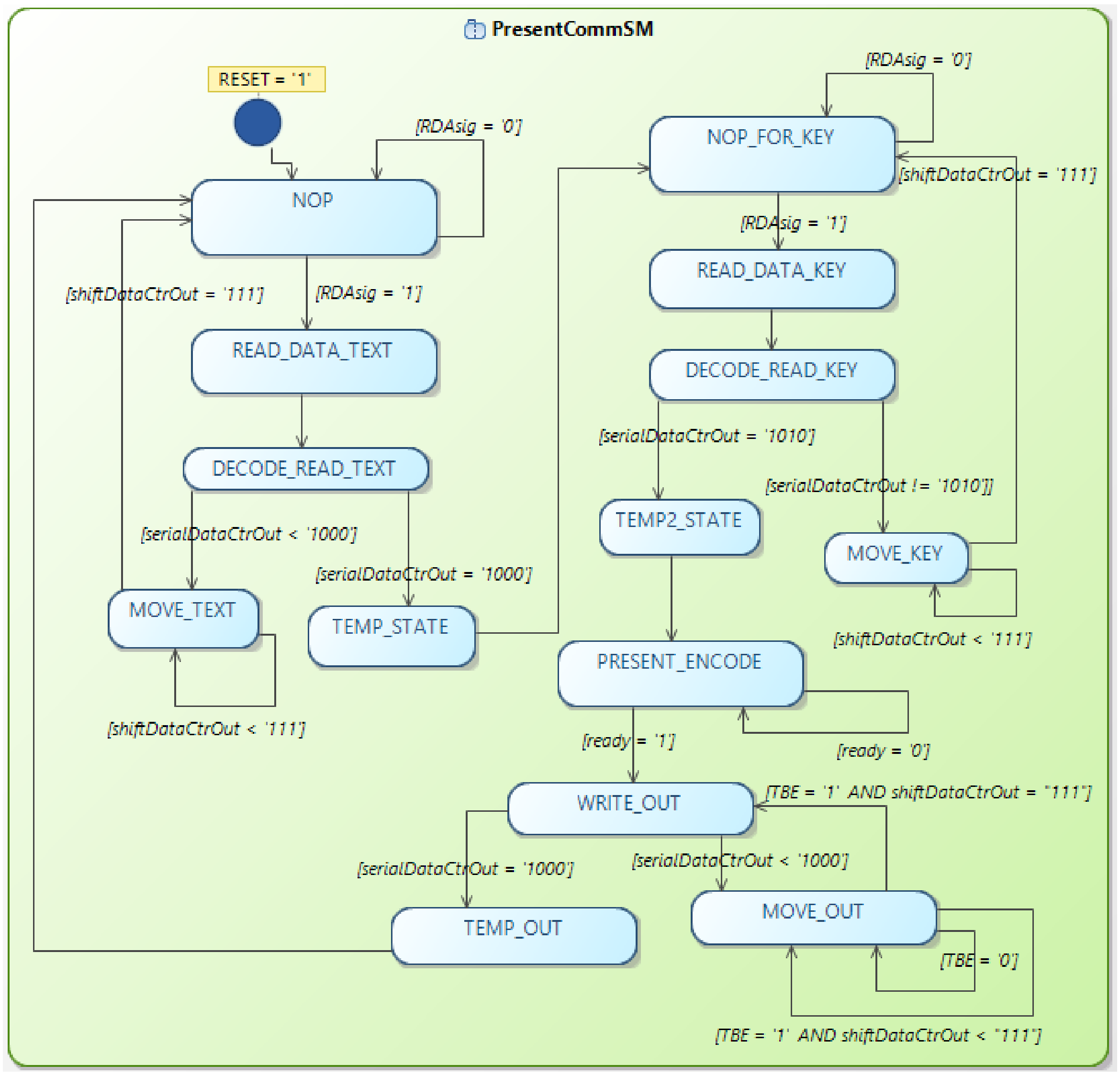
PresentCommSM
RESET = '1'
NOP
[RDAsig = '0']
[shiftDataCtrOut = '111']
[RDAsig = '1']
READ_DATA_TEXT
DECODE_READ_TEXT
[serialDataCtrOut < '1000']
[serialDataCtrOut = '1000']
MOVE_TEXT
TEMP_STATE
[shiftDataCtrOut < '111']
NOP_FOR_KEY
[RDAsig = '0']
[shiftDataCtrOut = '111']
[RDAsig = '1']
READ_DATA_KEY
DECODE_READ_KEY
[serialDataCtrOut = '1010']
[serialDataCtrOut != '1010']
TEMP2_STATE
MOVE_KEY
[shiftDataCtrOut < '111']
PRESENT_ENCODE
[ready = '1']
[ready = '0']
WRITE_OUT
[TBE = '1' AND shiftDataCtrOut = "111"]
[serialDataCtrOut = '1000']
[serialDataCtrOut < '1000']
TEMP_OUT
MOVE_OUT
[TBE = '0']
[TBE = '1' AND shiftDataCtrOut < "111"]

Figure 4: State machine of the Present Decode cipher with added communication component

- `READ_DATA_TEXT` / `READ_DATA_KEY` - This state informs the RS-232 component that input data was read (by write enable in `keyReg` register).
- `DECODE_READ_TEXT` / `DECODE_READ_KEY`- In this state the number of performed data reading iterations are checked. Because one RS-232 packet was set to 8 bytes - 8 iterations need to be performed for reading full 64 bit text data input (10 iterations for reading full 80 bit key data input).
- `TEMP_STATE` / `TEMP2_STATE` / `TEMP_OUT` - Here the counter is prepared for key reading / decoding / next "decoding session".
- `MOVE_TEXT` / `MOVE_KEY` / - Due to serial data in RS-232 component are stored in 8 bit register, they need to be shifted in appropriate place in given shift registers. It is performed by 8

shifts made in 8 clock cycles.

- `NOP_FOR_KEY` - Kind of `NOP` or wait state until 'key' data will arrive.
- `PRESENT_ENCODE` - In this state Present decoding is performed. This state is active until Present component informs about ending of the decoding process (`readySig = '1'`). [In this part the naming may be a little bit confusing, but this not interfere the work of the core]
- `WRITE_OUT` - state responsible for immediate sending decoded data. It is performed as many number as 64 bits of decoded data will be sent by the RS-232 component to the PC (similarly to "`DECODE...`" states).
- `MOVE_OUT` - it is similar state to the previous `MOVE...` states, but here additionally state machine must wait until output data buffer will be prepared for next data which have to be sent.

No "lost data" checking, and data correction protocol was performed. It was assumed "ideal channel" for communication. Some states could be "merged" into one state but it will involve more expanded control logic.

4 FPGA implementations

The component has been verified on a Xilinx® Spartan 3E XC3S500E FPGA in FG320 package and synthesized with Xilinx ISE 14.2. It was also implemented and practically tested on Spartan 3E Starter Board made by Digilent®. Appropriate setup files was prepared with use of ISE Project Navigator, but Makefile scripts was also written. Suitable files was stored in `./DecodeTesting/syn/XC3ES500/` directory. 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**	**Utilization**
Number of Slices	533	4656	11%
Number of Slice Flip Flops	530	9312	4%
Number of 4 input LUTs	582	9312	5%
Number of bonded IOBs	4	232	1%
Number of GCLKs	2	24	8%
Minimum period	6.343ns	-	-
Maximum Frequency	157 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 `./DecodeTesting/syn/XC3ES500/` directory to another one like `./DecodeTesting/syn/YOUR_FPGA_SYMBOL/`
2. Go to `./DecodeTesting/syn/XC3ES500/` directory.
3. In `PresentDecodeComm.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`

WARNING!!! With this core there exist one issue related to communication process. It was observed, that every **first** decoding process after resetting the core, returns incorrect data. Resending data and performing new calculations fix that. So, using this core it is desirable to force performing first "dummy" calculations after every reset.

[1]This solution was not tested and is based on my own observations. Additional care should be taken with *.UCF files - this supplied with this project should be appropriate only for Spartan 3E Starter Board made by Digilent®. You can make this modifications on your own risk

5 Simulation and software

5.1 Simulation

Self-checking test bench were provided to the components used for Present encoder with RS-232 communication. They are stored in `./DecodeTesting/bench/vhdl` directory. Suitable configuration files and Makefile used for running test bench was stored in `./DecodeTesting/sim/rtl_sim/bin` directory. Appropriate test vectors was taken from [1]. In `PresentDecodeCommTB.vhd` file with suitable test files stored in `./DecodeTesting/sim/rtl_sim/bin/test` directory simulation of RS-232 communication was prepared. Due to that only this one test bench is not self checking. Observation and testing of the communication in this case will be most comfortable using isim gui.

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

5.2 Software

With this project two tool programs written in Java was included:

- `PresentDataGenerator` (class with the same file name)
- "GUI Application" which consist of two classes (Communication.java and Window.java)

They were brought into Eclipse project, which can be easy imported. It was tested with Eclipse Indigo version.

First of them is used to prepare data for `PresentDecodeCommTB`. It can be used by:

- Setting `drive`, `data` and `key` variables with hexadecimal values as it is desired.
- Running the compiler and running program.

On its output it sends set of bits which are sequentially sent to the PresentDecodeComm component during test bench.

"GUI application" enables communication with PC by use of RS-232 connection. RS-232 communication in Java is delivered by `rxtx` library. It was partly based on tutorial which can be found at [4]. This program can be used as follow:

- After connecting FPGA board to the RS-232 port click the "Connect" button.
- To the "Data" and "Key" write suitable hexadecimal data used for encoding.
- Press "Send" button.
- Answer should appear in "Log" box in hexadecimal values.

These programs were not prepared for unusual cases, so entering intended inappropriate values (like non hexadecimal values) are not recommended.

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

[4] H. Poon. (2014, May) Serial communication in java with example program. [Online]. Available: http://henrypoon.wordpress.com/2011/01/01/serial-communication-in-java-with-example-program/