HDL Project – CRC generation

MEM – Embedded Systems with constrained ressources

V02, 21.01.2017	Add concept and a few details	ANM

Learning targets

- Digital design
  - Understand and implement basic digital building blocks like flip-flops, counters, state machines.
  - Analyze how a complex design can be partitioned in building blocks to facilitate group work, allow for easier implementation and verification.
- FPGA design flow
  - Simulate and verify modules (building blocks)
  - Simulate and verify the whole design.
  - Synthesize the design in an FPGA.
  - Verify and debug the FPGA implementation.
  - Validate that the overall design satisfies the *voice of the customer*.

Introduction

A memory check shall be implemented by a Cyclic Redundancy Check (CRC). CRC is an error-detection method commonly used in digital communication (e.g. Bluetooth) and storage devices to detect accidental changes to raw data.

CRC are popular due to their simple implementation in digital hardware:

- Flip-flops arranged as a shift registers with feedback → linear feedback shift register (LFSR).
- XOR logic gates to implement modulo2 addition.

The CRC was invented by W. Wesley Peterson in 1961; the 32-bit CRC function of Ethernet and many other standards is the work of several researchers and was published in 1975. (Wikipedia, 2017)

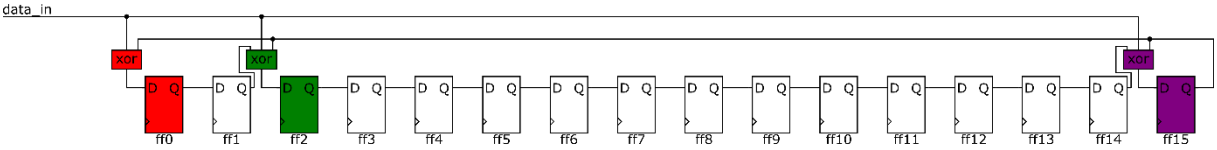
The CRC algorithm that shall be used in this project is the so called “CRC-16-IBM” CRC. It is used amongst others in USB2.0.

CRC algorithm

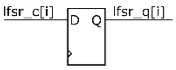
The CRC structure is usually specified by its polynomial. The polynomial denotes which bits of the CRC word are used for xor-ing. It is denoted either as HEX- number or polynomial:

- crc[15:0]=1+x^2+x^15+x^16;
- 0x8005

Both notations refer to the following structure:



CRC-16-IBM: 0x8005 = 0b1000_0000_0000_0101
--> crc[15:0]=1+x^2+x^15+x^16;




```
always_comb begin
   lfsr_c[0] = lfsr_q[15] ^ data_in;
   lfsr_c[1] = lfsr_q[0];
   lfsr_c[2] = lfsr_q[1] ^ lfsr_q[15] ^ data_in;
   lfsr_c[3] = lfsr_q[2];
   lfsr_c[4] = lfsr_q[3];
   lfsr_c[5] = lfsr_q[4];
   lfsr_c[6] = lfsr_q[5];
   lfsr_c[7] = lfsr_q[6];
   lfsr_c[8] = lfsr_q[7];
   lfsr_c[9] = lfsr_q[8];
   lfsr_c[10] = lfsr_q[9];
   lfsr_c[11] = lfsr_q[10];
   lfsr_c[12] = lfsr_q[11];
   lfsr_c[13] = lfsr_q[12];
   lfsr_c[14] = lfsr_q[13];
   lfsr_c[15] = lfsr_q[14] ^ lfsr_q[15] ^ data_in;
end
```

After all bits are processed sequentially the result of the CRC check is the content of the 16 flip flops.

Parallel CRC algorithm

To increase data throughput the serial CRC algorithm can be paralleled. This is especially beneficial if a byte organized memory shall be checked.

Details can be found in the white paper “A Practical Parallel CRC Generation Method” (https://ilias.fhv.at/goto_ilias_fhv_at_file_307397_download.html)

Project specification

The content of a memory with 1024 8bit memory cells shall be processed by a CRC checker and then be compared to a pre-defined number to verify that the memory content is valid.

1. Load a data byte.
2. Process it with the CRC algorithm
3. If at last memory cell compare to defined result and output “crc_ok”, else go to step 1.

Concept

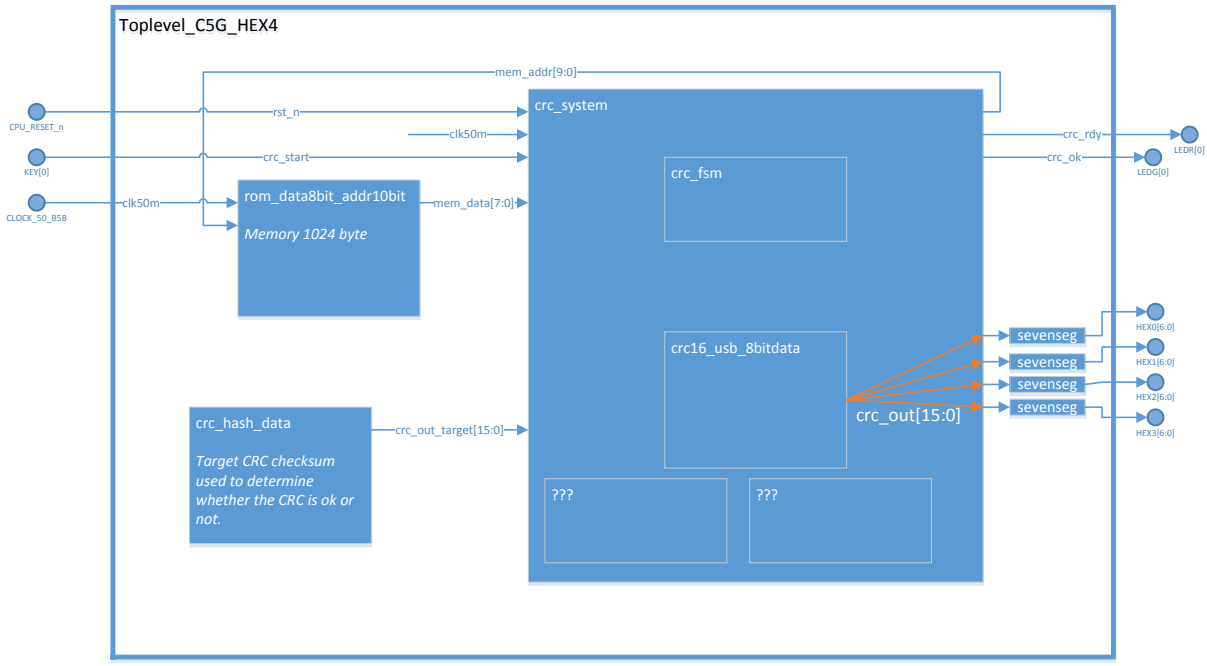


Figure 1: Possible toplevel block diagram of the memory check.

Module memory

A memory containing 1024 8bit elements. The memory content is defined by the text file "rom_data8bit_addr10bit.txt".

Module crc_fsm

A state machine that controls the CRC process.

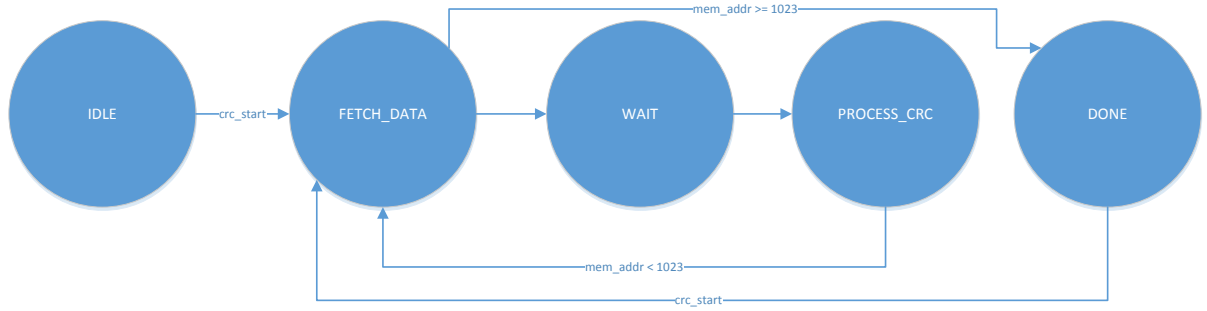


Figure 2: Possible state machine that controls the CRC check.

Module crc_check

A CRC16 check with the polynom CRC-16-IBM 0x8005 (crc[15:0]=1+x^2+x^15+x^16) shall be used.

The Verilog code of the CRC algorithm can be generated automatically (e.g. http://outputlogic.com/?page_id=321).

Project template

In order to facilitate a fast project development a template project is available as zip file in ILIAS. The content of this zip file is

- ./fpga: Contains a Quartus Project "Toplevel_C5G_HEX4" that already contains
  - A toplevel *Toplevel_C5G_HEX4.sv*
  - All pin assignments

    - The *memory rom_data8bit_addr10bit.v* and constant *crc_hash_data.v*
    - The expected crc checksum value for the uncorrupted memory is 0xbd38.
- ./src_sv
    - The crc source file *crc16_usb_8bitdata.sv*.
    - The sevensegment decoder *sevenseg.v*.
- ./sim_sv
    - TCL script sim_tb_Toplevel_C5G_HEX4.tcl that shall be used to simulate the toplevel in Modelsim.
    - Testbench *tb_Toplevel_C5G_HEX4.sv* that verifies that the created CRC system processes a
        - Correct memory content → crc_out == crc_out_target
        - Memory with one bit error → crc_out != crc_out_target
        - Memory with two bit errors → crc_out != crc_out_target

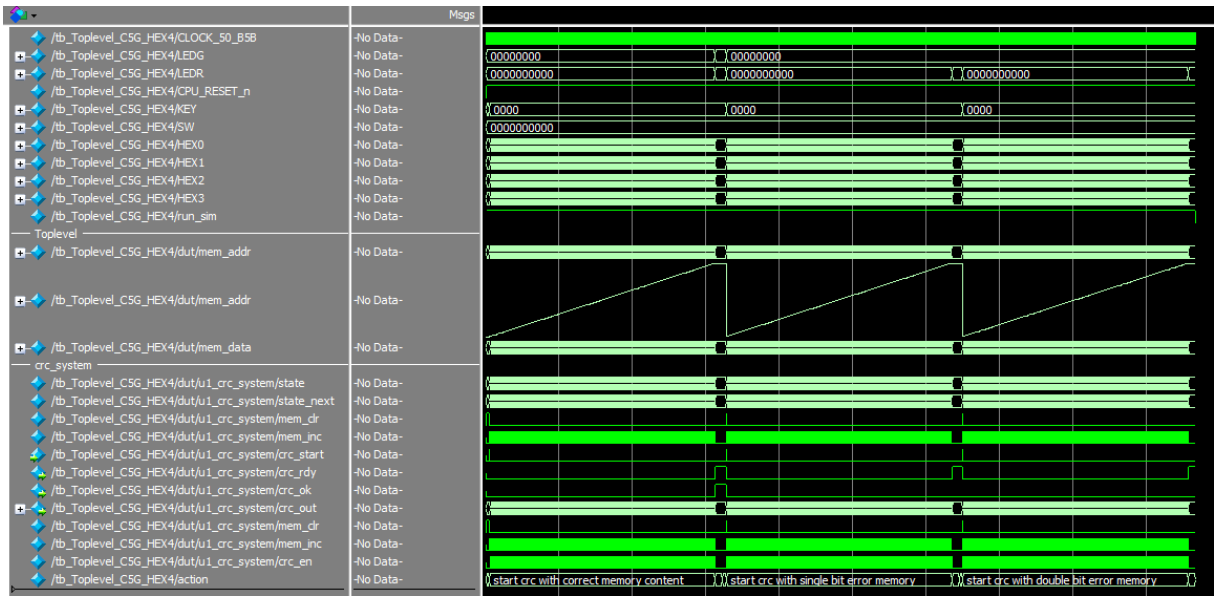


Figure 3: Wave window of the toplevel testbench simulation.

The project can be simulated and synthesized but is not complete (and does not pass the test bench). The missing part is the content of the crc_system.sv module. It is your task to

- Come up with a concept for the crc_system module and to draw a block diagram.
- Finalize the project (coding, verification/simulation, synthesis) and to demonstrate the running system in hardware (demonstrator/proof of concept)

Expected results

- Digital design (20cp)
    - Concept and block diagram showing all IOs, submodules and toplevel wiring (drawing) (5cp).
    - RTL design of toplevel and all submodules (5cp).
    - Test benches that can be used to check the functionality of the submodules (5cp).
    - Passing the already existing toplevel testbench *tb_Toplevel_C5G_HEX4.sv* (5cp).
- FPGA implementation (5 cp)
    - Compile the project for the FPGA board.

- Document the device utilization.
- Document the RTL block diagram. Does it match the expected diagram?
- Proof of concept → working prototypes that implements all features.

References

Wikipedia. (2017, Jan 16). *Cyclic redundancy check*. Retrieved from Cyclic redundancy check: https://en.wikipedia.org/wiki/Cyclic_redundancy_check