

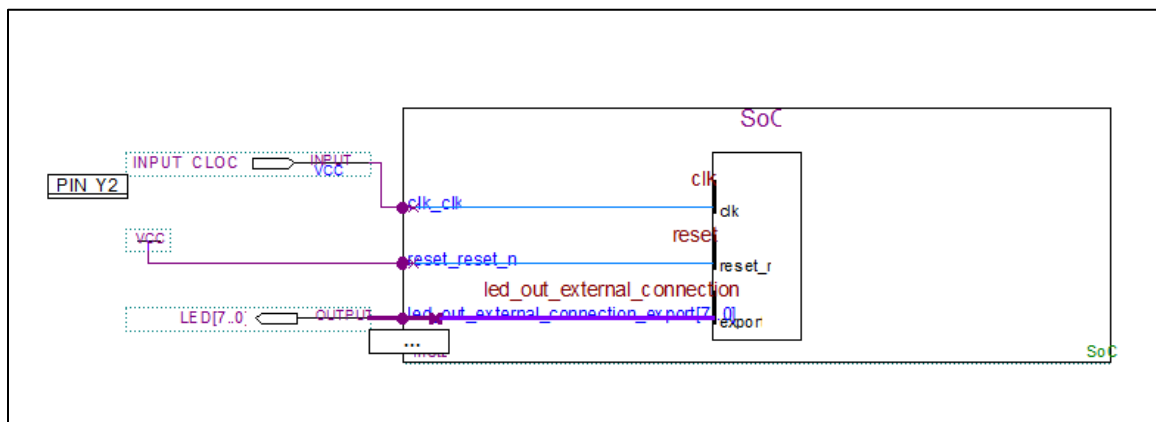
CO503: Advanced Embedded Systems

Practical 2 – Processor Customization

E/17/168

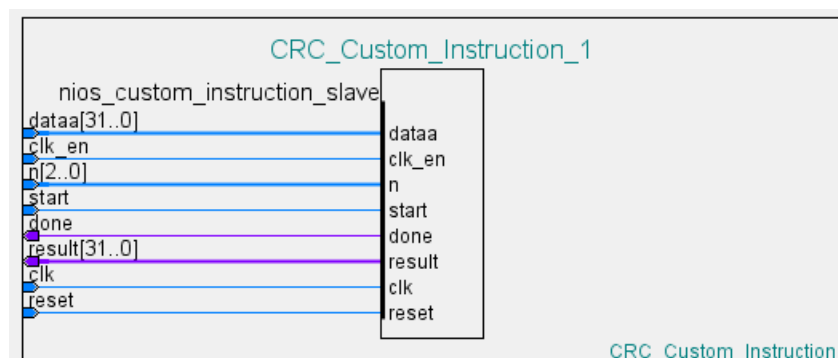
K.H.S.P.Kodagoda

1. Initially, a System-on-Chip was created according to the lab 1. The Nios II/s processor was used as the CPU of the SoC. The custom instruction logic could be directly connected to the Nios II arithmetic logic unit.



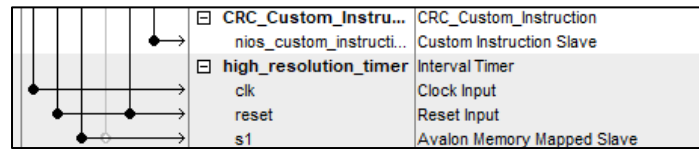
System on Chip and input/output signals.

2. Then the new component was added to the library in Qsys. It was a custom hardware for performing modulo-2 division. This can be used to do cyclic redundancy check. This CRC calculation consists of an iterative algorithm with XOR and shift operations. The new component could be added to the library by using component editor of the Qsys. There were two Verilog files which are called “CRC_Component.v” and “CRC_Custom_Instruction.v”. They were added as synthesis files of the component. These files describe this component’s implementation. The parameters and signals found in the top-level module will be used for this component’s parameters and signals.



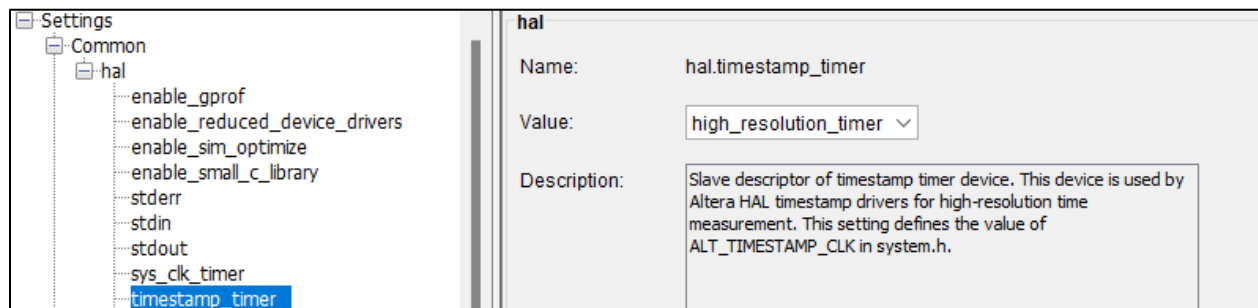
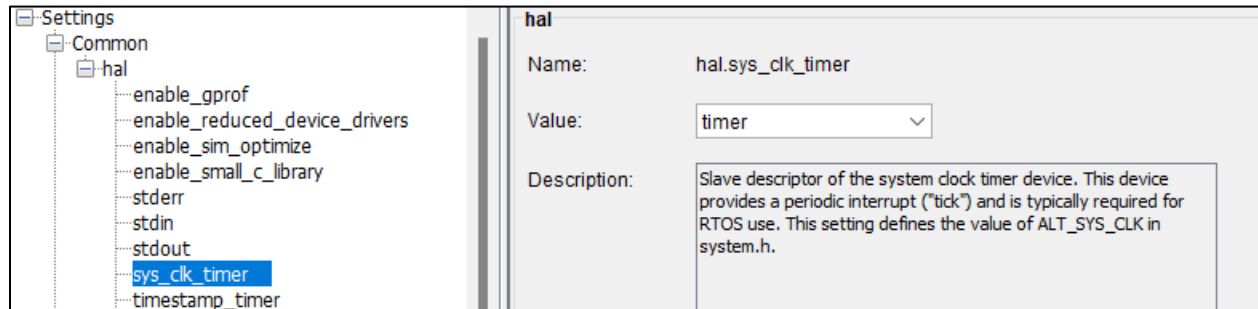
Block Diagram of the CRC Component.

3. Then CRC component and high-resolution timer were added to the Qsys design.



CRC component and high-resolution timer.

4. After design was compiled and uploaded to the FPGA board. Then software for the FPGA board was implemented by using Eclipse and as the system clock timer normal timer was used and as timestamp timer high resolution timer was used.



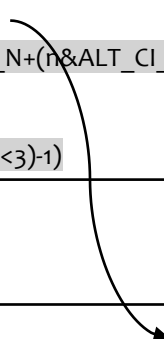
5. After that “ci_crc.c, ci_crc.h, crc_main.c, crc.c, crc.h” files were imported to the project. The it was built, and it gave an error inside the “ci_crc.c” file.

File Name	Description
crc.c	Software CRC algorithm run by the Nios II processor.
crc.h	Header file for crc.c.
ci_crc.c	Program that accesses CRC custom instruction.
ci_crc.h	Header file for ci_crc.c.

Defining line of a macro `#define CRC_CI_MACRO (n, A)` was not correct and from “system.h” file `ALT_CI_CRC_CUSTOM_INSTRUCTION_o(n,A)` function was imported and by using it that error was corrected.

```
/*
 * Custom instruction macros
 *
 */

#define ALT_CI_CRC_CUSTOM_INSTRUCTION_o(n,A)
__builtin_custom_ini(ALT_CI_CRC_CUSTOM_INSTRUCTION_o_N+(n&ALT_CI_CRC_CUSTOM_INSTRUCTION_o_N_MASK),(A))
#define ALT_CI_CRC_CUSTOM_INSTRUCTION_o_N 0x0
#define ALT_CI_CRC_CUSTOM_INSTRUCTION_o_N_MASK ((1<<3)-1)
```



```
#define CRC_CI_MACRO(n, A) ALT_CI_CRC_CUSTOM_INSTRUCTION_o(n,A)
```

6. Then it was compiled and run. It gave below result.

```
Processing time for each implementation
-----
Software CRC = 49 ms
Optimized software CRC = 07 ms
Custom instruction CRC = 01 ms

Processing throughput for each implementation
-----
Software CRC = 1560 Mbps
Optimized software CRC = 978 Mbps
Custom instruction CRC = 2048 Mbps

Speedup ratio
-----
Custom instruction CRC vs software CRC = 77
Custom instruction CRC vs optimized software CRC = 53
Optimized software CRC vs software CRC= 1
```

Conclusion

The idea of a CRC is like that of a checksum bit. With a CRC you compute a checksum code from a message, a stream of bytes. Then, at a later point, you re-compute the checksum over the message and verify that you have obtained the same checksum. This process ensures the integrity of data. In this code there are three implementations of same process.

1. A slow, bit-serial implementation in software.
2. A faster software implementation that uses the table-lookup method.
3. A custom-instruction design that uses an unrolled CRC computation that evaluates the CRC of up to 32 message bits in one clock cycle.

Processing time for each implementation	
Software CRC	49 ms
Optimized software CRC	07 ms
Custom instruction CRC	01 ms
Processing throughput for each implementation	
Software CRC	1560 Mbps
Optimized software CRC	978 Mbps
Custom instruction CRC	2048 Mbps
Speedup ratio	
Custom instruction CRC vs software CRC	77
Custom instruction CRC vs optimized software CRC	53
Optimized software CRC vs software CRC	1

According to the result slowest process was bit-serial implementation in software. It took 49ms and the software implementation with look up tables took 07ms. But custom-instruction design of CRC only took 1ms. So here we could observe that customized hardware implementation is faster and for special operations this kind of hardware implementations can be used. Also processing throughput of the custom CRC was high and it could 2048 Mb data per second. So, hardware implementation is better for special operations and to do that processor can be customized by adding new hardware components to a processor.