CO503: Practical 2 - Processor Customization

Group 3

E/19/155, E/19/166

Introduction

The practical builds a System-on-Chip with a modified NiosII Processor using FPGA design tools. The purpose of the customisation is to enhance the efficiency of the modulo-2 division operation in the network device's Cyclic Redundancy-Check (CRC) algorithm.

Exercises

1. To execute the modulo-2 division, add a custom instruction to the NiosII processor's current MIPS ISA.
2. Use XOR and shift operations to implement the necessary hardware capability to enable the custom instruction.
3. Utilize the CRC algorithm with the recently added custom instruction, and assess how well it performs in comparison to pure software implementations.

Implementation

IP blocks with the bare minimum of components were used in the hardware implementation to support the CRC custom instruction.

Used IP blocks,

1. clk - clk / reset signals for the Soc
2. timer0 - For interval timer operation
3. high_res_timer - To calculate time intervals between implementations in software/hardware
4. sysid - Gives a unique ID to the SoC which prevents accidental downloading of software.
5. cpu - Processor of the SoC
6. jtag_uart - UART connection with the computer
7. onchip_mem - Data / Instruction code save

Hardware Implementation

To handle the custom instruction, we must develop a new component after building the base system on Qsys with the aforementioned components.
Using the supplied Verilog files, CRC_Custom_Instruction.v & CRC_Component.v, the component is constructed. We can use the files to create a single interface called Neos Custom Instruction Slave.
of the Customized Instruction Slave kind
The clock cycle type is set to Variable with operands set to 1, and the clock cycles are set to 0 as parameters.
The signals were configured using the interface signal kinds after the interface was defined.
We can connect the component's custom instruction slave to the CPU's custom instruction master after it has been created.
The freshly created SoC can then be imported to finish the BDF file.

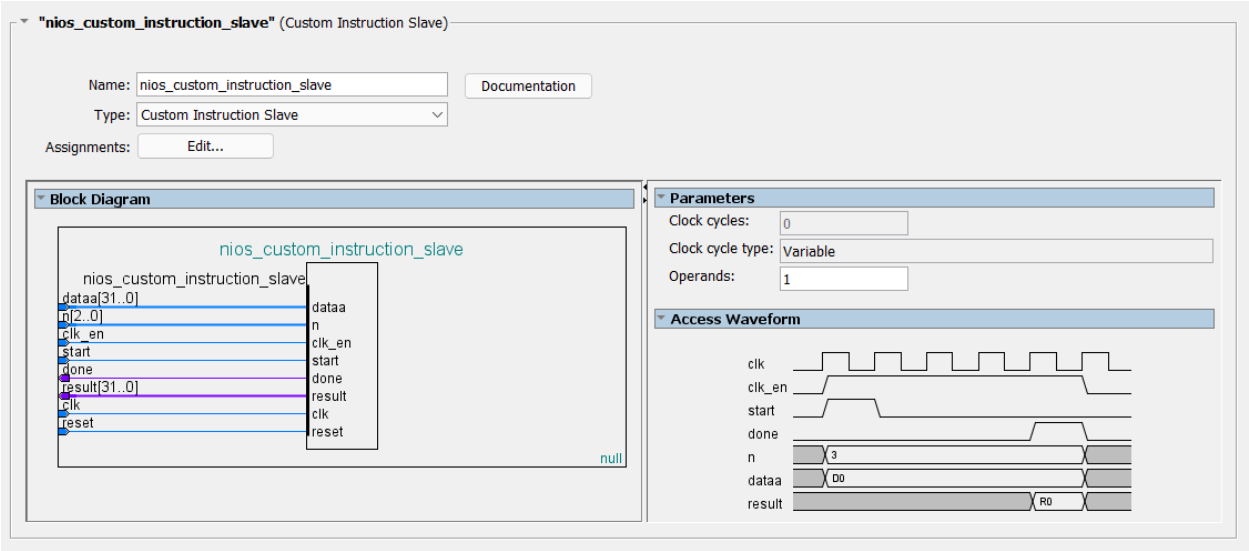

Figure 01: CRC Custom Instruction Interface

Name	Interface	Signal Type	Width	Direction
dataa	nios_custom_instruction_slave	dataa	32	input
n	nios_custom_instruction_slave	n	3	input
clk_en	nios_custom_instruction_slave	clk_en	1	input
start	nios_custom_instruction_slave	start	1	input
done	nios_custom_instruction_slave	done	1	output
result	nios_custom_instruction_slave	result	32	output
clk	nios_custom_instruction_slave	clk	1	input
reset	nios_custom_instruction_slave	reset	1	input

Figure 02: Signals of CRC Custom Instruction

Software Implementation

After initializing the project, files provided by the software must be imported.

ci_crc.h - Header file for ci_crc.h
ci_crc.c - Access the CRC custom instruction
crc.h - Header for crc.c

crc.c - Software CRC algorithm run by the Nios II processor
crc_main.c - Main program that populates random test data and execute CRC in both software and custom instruction, validates output and reports processing time.

- Setting timer0 as the sys_timer and high_res_timer as the timestamp timer would need to be changed.
- After the BSP is produced, the project can be constructed.
- The first mistake we get here indicates that CRC_MACRO(n,A) is not defined.
- At the hardware level, this function is used to manage the CRC custom instruction.
- We can tell from the error that this function's definition has to be updated to match the one that the system.h provides.
- When utilizing Qsys, the system.h file is generated in tandem with the SoC.
- Examining the system.h file reveals the definition for the ALT_CI_CRC_CUSTOM(n,A) as well as the commented CRC custom instruction section.
- Since CRC_MACRO(n,A) is required to access this, we can copy and paste ALT_CI_CRC_CUSTOM(n,A) at the definition in ci_crc.c.

Performance Comparison

In comparing the performance of software CRC, optimized software CRC with lookup tables, and custom instruction CRC, significant differences arise. The software CRC implementation processes each bit individually, leading to slower execution due to its bit-wise operations. While it requires minimal memory usage, it lacks optimization for speed. The optimized software CRC with lookup tables improves speed by performing byte-by-byte operations, reducing redundant division operations. However, it increases memory usage due to the lookup table and incurs overhead in table generation. Conversely, the custom instruction CRC leverages hardware acceleration through Nios II's custom instructions, enabling faster execution of iterative algorithms involving XORs and shifts. This approach significantly enhances CRC calculation speed, outperforming both software implementations. However, it may require additional hardware resources and design complexity. Overall, the custom instruction CRC offers the highest performance by trading off hardware complexity for improved computational efficiency.

```
+------------------------------------------------------------+
| Comparison between software and custom instruction CRC32   |
+------------------------------------------------------------+


System specification
--------------------
System clock speed = 50 MHz
Number of buffer locations = 32
Size of each buffer = 256 bytes


Initializing all of the buffers with pseudo-random data
-------------------------------------------------------
Initialization completed


Running the software CRC
------------------------
Completed


Running the optimized software CRC
----------------------------------
Completed


Running the custom instruction CRC
----------------------------------
Completed


Validating the CRC results from all implementations
---------------------------------------------------
All CRC implementations produced the same results


Processing time for each implementation
---------------------------------------
Software CRC = 438 ms
Optimized software CRC = 295 ms
Custom instruction CRC = 05 ms


Processing throughput for each implementation
---------------------------------------------
Software CRC = 16384 Mbps
Optimized software CRC = 1213 Mbps
```

Figure 03: Comparison of Results(1)

```
Custom instruction CRC = 978 Mbps


Speedup ratio
-------------
Custom instruction CRC vs software CRC = 81
Custom instruction CRC vs optimized software CRC = 54
Optimized software CRC vs software CRC= 1
```

Figure 04: Comparison of Results(2)

Additional Errors

1. Output Error when running
   A "CRC outputs not the same" problem was reported in the run report after the aforementioned faults were fixed and the code was executed on hardware.Examining the message reveals that the output of the CRC custom instruction is 0x0.

2. Error when building (found in BSP)
   Redefining previously defined variables that are already present in the bsp is what caused these problems. The board support package, or BSP, generates mistakes because it is always trying to optimize the program. This can be resolved by taking out the redefinitions, cleaning the bsp, and building it.

3. Errors on memory allocation
   We had adjusted the onchip memory to the default values in our initial build. There is not enough memory here. 256KB of memory can be allocated to address this problem.

4. An overflow error occurred during calculations in both the Software CRC and Optimized Software CRC.

   To address this issue, adjustments were made to the data types of the counting parameters and other parameters from 'alt_u32' to 'alt_u64' and from 'unsigned long' to 'unsigned long long', respectively.

While the results of software installation vary. Returning to QSYS, we saw that, in contrast to the documentation, our implementation of the custom instruction has three interfaces: CLK RESET, the custom instruction slave, and one. Observing the signals indicates that, as opposed to coming from the nios_custom_instruction_slave, the CLK and RESET get the signals independently. The problem was fixed by eliminating these unnecessary interfaces and allocating the proper signals.