ABHISHEK NAYAK
PUID: 33791158
nayak19@purdue.edu

CRC Custom Instruction Design

ECE 695R: System-on-Chip Design (Homework #4)

14th October 2021

OVERVIEW

In this Homework, I have designed a custom instruction for the CRC computation in the 802.11 MAC protocol. I was provided with reference software that implements the compute-intensive parts of the original IEEE 802.11b "Wi-Fi" MAC protocol.

DESIGN DESCRIPTION

Combinational Design

The combinational (single cycle) custom instruction implements one step of CRC computation. The instruction takes as inputs the 32- bit CRC and 8-bits of data and produces the updated CRC. The final XOR operation (crc_temp = crc_temp ^ $0xFFFFFFFF$) is implemented in software by flipping all the bits of crc_temp. The CRC table is stored internally to the custom instruction HW.

Multicycle Design

In multicycle design, the extended multicycle custom instruction supports 6 operations –

- Loading the value of CRC into custom instruction HW
- Computing CRC on 1-byte of data
- Computing CRC on 2-bytes of data
- Computing CRC on 3-bytes of data
- Computing CRC on 1-word of data
- Reading the value of CRC from the custom instruction HW.

The wait cycles for 2-bytes CRC computation, 3-bytes CRC computation, 1-word CRC computation are 2 cycles, 3 cycles, 4 cycles respectively. Otherwise, it is a 1- cycle wait (basically, the processor can process new instruction the following cycle). In this way, the extended multicycle design implements variable latency.

The software loads the initial CRC value into the custom instruction HW. If the start or end bytes of payload/frame are unaligned, they are handled by performing sub-word CRC computations (on 1-byte/2-byte/3-byte of data) in the custom instruction HW. For the remaining bytes of payload/frame, the software performs CRC computation on 1-word of data in the custom instruction HW. The updated CRC is read out from the custom instruction HW and is flipped to get the final CRC value.

Extra Credit Design

The extended multicycle custom instruction is enhanced to support 7 operations – the 6 operations of the multicycle design and additionally, computing CRC on 2-words of data.

The wait cycle for the 2-words CRC computation is 8 cycles. The other wait cycles are the same as in the multicycle design.

Software is also enhanced to support sending doubleword data to the custom instruction HW when possible. If the bytes in the payload/frame cannot be sent in blocks of doubleword data, it sends word by word to the custom instruction HW. The start and end bytes of payload/frame, if unaligned, are handled in a similar manner as in the multicycle design.

PERFORMANCE MEASUREMENTS RESULTS

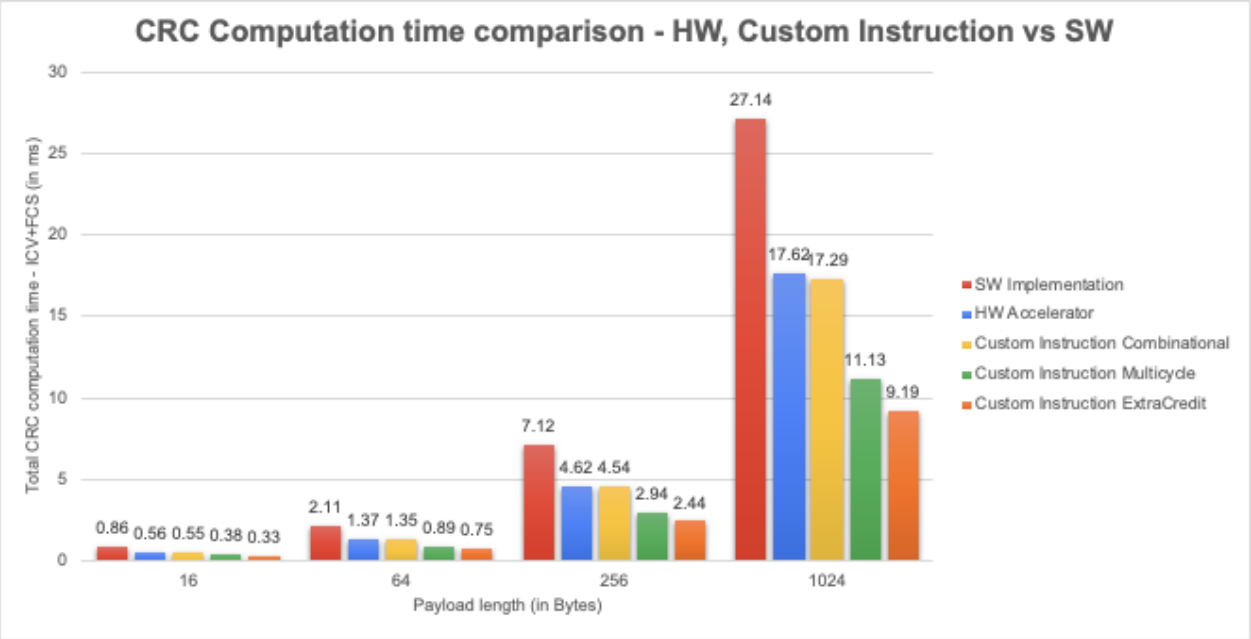
The performance calculator peripheral is also added to the SoC for accurately timing the reference software and Custom instruction implementation.

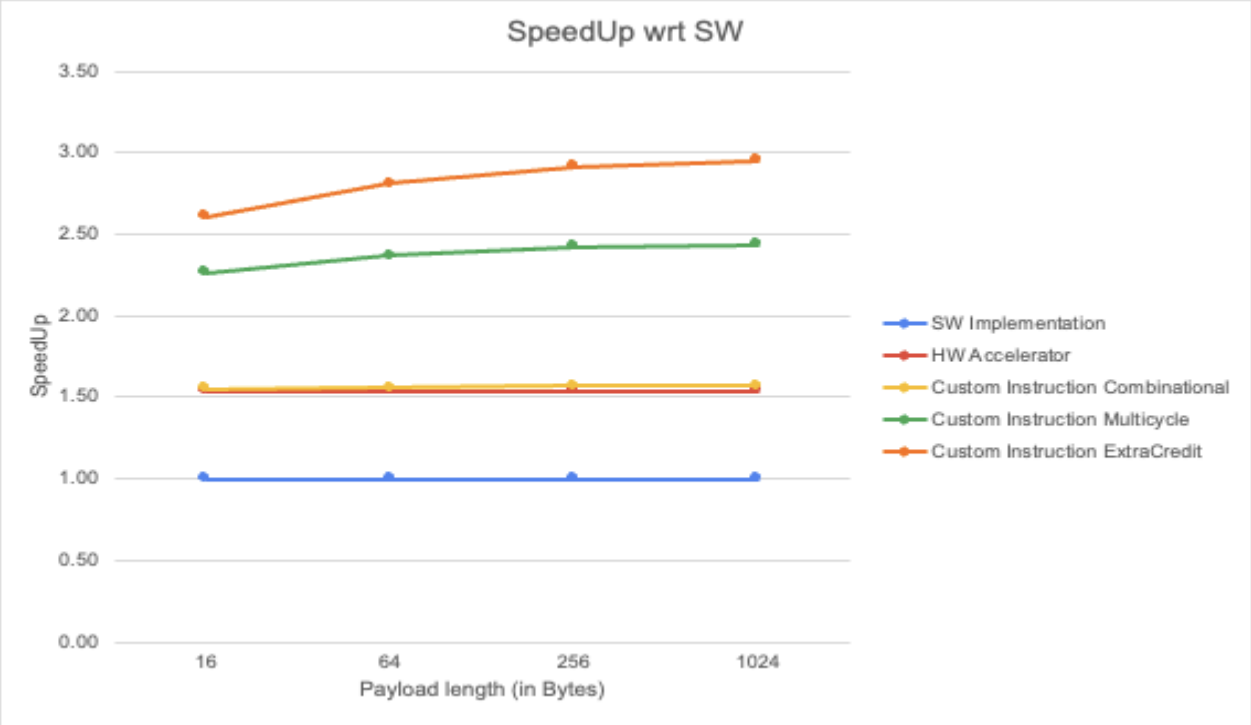
The execution time of the CRC computation (ICV time + FCS time) is measured for a single frame of different payload lengths - 16B, 64B, 256B, and 1024B.

The execution times shown below are measured across the reference software implementation (SW Basic), HW Accelerator (implemented in HW#3), Combinational Custom instruction, Multicycle Custom instruction, and the Extra credit design. The speedup achieved with respect to the reference software implementation is shown for the CRC computation time.

Payload length (B)	Total CRC computation (ms)				
	SW	HW Accelerator	CI Combinational	CI Multicycle	CI ExtraCredit
16	0.86	0.56	0.55	0.38	0.33
64	2.11	1.37	1.35	0.89	0.75
256	7.12	4.62	4.54	2.94	2.44
1024	27.14	17.62	17.29	11.13	9.19

Payload length (B)	SpeedUp (wrt to SW)				
	SW	HW Accelerator	CI Combinational	CI Multicycle	CI ExtraCredit
16	1.00	1.54	1.55	2.26	2.61
64	1.00	1.54	1.56	2.37	2.81
256	1.00	1.54	1.57	2.42	2.92
1024	1.00	1.54	1.57	2.44	2.95





It is noticed that the overall CRC computation speeds up by **1.54x** (with respect to SW implementation) when using the HW Accelerator.

The combinational custom instruction design speeds up the CRC evaluation by **~1.55x**. Whereas, the extended multicycle custom instruction design achieves a speedup of almost **~2.4x**. Of the 3 strategies for the custom instruction design, the extra credit custom instruction design strategy outperforms the extended multicycle custom instruction design by about **1.2x**.

ANALYSIS OF RESULTS

Comparison of performance of Custom instruction wrt HW Accelerator

The combinational custom instruction achieves the same performance (1.55x) as the primitive design for the HW accelerator (1.54x). The combinational custom instruction does not stall the processor pipeline and neither does the HW accelerator. The processor and the HW accelerator can operate concurrently. The processor sends the payload/frame data to the HW accelerator through the Avalon bus. However, this does not halt the processor from processing subsequent instructions.

The extended multicycle custom instruction stalls the processor pipeline (for 3 more cycles) for the CRC computation on each word of payload/frame data. These additional cycles are negligible when compared to the total cycles spent on the CRC computation. In the case of the HW accelerator, the processor and HW accelerator can operate concurrently.

Comparison of Accelerator vs Custom instructions as two approaches to HW/SW partitioning

In summary, the Multicycle Custom Instruction approach outperforms the HW Accelerator by a significant margin with better design optimizations. Although a possible downside of Custom Instruction is that stalls the processor pipeline. But, the HW accelerator can operate concurrently with the processor. Also, the custom instruction approach limits the number of inputs and outputs that a custom instruction HW can handle at a time. For large-scale designs with multiple inputs and outputs, the HW accelerator is the better approach to HW/SW partitioning.