EE2003: Course Project 2021 Implementation and Analysis of Peripheral for NanoJPEG Decoder in PicoRV32 Environment

Bachotti Sai Krishna Shanmukh EE19B009
Katari Hari Chandan EE19B032
Potta Muni Asheesh EE19B048

December 8, 2021

1 Introduction

We aim to identify the bottleneck in the NanoJPEG decoder running on a 186 $\times$ 170 JPEG kitten image in PicoRV32 processor in it's default configuration (regular). Later we design two peripherals and use Verilog as the RTL Language. We use AXI4- Lite Bus for interfacing between the processor and peripheral. For ease of faster simulation verilator is used. Due to resource constraints, we restricted ourselves to Yosys Open synthesis suite for synthesis of design.

2 Finding the Bottleneck

With the help of "stats_helper.c" module, we could find the approximate number of clock cycles and number of instructions between any two points in our code with the *get_num_cycles* function and *get_num_instr* function respectively. The entire njDecode() function is broadly split down by a switch-case into six different functions: SOF, DQT, DHT, DRI, Decode Scan and Skip marker operations. There is also a default state which conditionally implements Skip marker. We used *get_num_cycles* function at these different cases to identify the part consuming more clock cycles.

```
Default Operation : 920 cycles.
DQT Operation :5107 cycles.     SOF Operation :7489 cycles.
DHT Operation :7342181 cycles.  Decode Scan Operation :83065112 cycles.
Decoded the image in 133972744 cycles.
Cycle counter ........169506305
```

```
Instruction counter ..40099394
CPI: 4.01
```

Clearly, Decode Scan operation takes more clock cycles than any other. Hence, we decided to further investigate it. We observe that the row and column IDCT eat up a large share of clock cycles and hence the bottleneck for the entire code.

```
njDecode Block active cycles :82436686 cycles
In njDecode Block, IDCT active cycles :66552893 cycles
No. of row and col IDCT instructions :15796746
```

3 Row and Column IDCT Peripheral Design

From the description of PicoRV32 processor[1], we can see that the CPI of ALU (Reg + Reg, Reg + Imm) is 3. We assume that the ALU stage for an ALU instruction (like add,sub) is busy for atmost 1 clock cycle. Also, given in the manual that a barrel shifter would take as much time as any other ALU instruction. Using this information, we assumed that a 32 bit adder and shifter have time delays equivalent to 1 clock cycle duration in our peripheral too.

We designed two different modules for Row IDCT and Column IDCT respectively. All data computations in our peripheral are done combinationally. However, we also have a clock, reset and counter to put some required features like resetting the module and generating a ready signal when the data outputs are ready. The combinational logic is mostly made from additions and shift operations only. All multiplications can be written as combination of addition and shift operations, as one of the operands is known before hand. From our delay assumptions stated above, we found the critical path by simulating with extrinsic time delays in Verilog. One must note that this was done only for simulation purpose and the delays were removed for synthesis part. For rowidct module, the IF-path and ELSE-path critical delay are 1 and 16 clock cycle equivalents respectively. For colidct module, the IF-path and ELSE-path critical delay are 3 and 17 clock cycle equivalents respectively. More information can be seen in timing folder. After finding the critical path in clock cycle equivalents, we decided appropriate counter value at which the ready signal can be set high.

For synthesis, we used Yosys Open Synthesis suite and our Verilog designs synthesized well, with no latches inferred and zero warnings, without compromising the functionality. More info can be found in row/col workspace in project folder. We have used Yosys-synthesized verilog files in makefile execution.

4 Analysis of Peripheral Performance

After interfacing the peripheral through well routed address map in AXI-4 Lite bus, we could successfully run the NanoJPEG decoder code. Below is the clock cycle analysis after peripheral interfacing is done.

```
Default Operation :936 cycles.  DQT Operation :5101 cycles.
SOF Operation :7506 cycles.     DHT Operation :6782667 cycles.
Decode Scan Operation :28366758 cycles.
Decoded the image in  78721156 cycles
Cycle counter ........114050004
Instruction counter ..27032167
CPI: 4.04
njDecode Block active cycles :27752658 cycles
In njDecode Block, IDCT active cycles :11696803 cycles
No. of row and col IDCT instructions:2710662
```

- We can observe that the number of cycles in Decode scan operation has decreased by a factor of 2.93.
- Within the Decode scan operation, the total IDCT active cycles has decreased by a factor of 5.69 i.e., the IDCT computation is now faster with peripheral.
- Before the peripheral accessing was done, the IDCT operation needs 15.8M RISC-V 32I instructions (approx.) whereas after the peripheral accessing is done, the IDCT operation needs 2.7M RISC-V 32I instructions.
- Although, the decrease in number of instructions looks appealing, the instruction mix has now more loads and stores and the CPI of LOAD/STORE is 5 for PicoRV32 processor.
- We can observe that CPI of IDCT computation is 4.21 before peripheral access and 4.31 after peripheral access. This explains, the very little change observed in overall CPI from 4.01 to 4.04.

Division of Work

- Inferring NanoJPEG software implementation - EE19B009,EE19B032, EE19B048.
- Identifying Bottleneck - EE19B032
- Row IDCT Verilog Implemenation, design testing and simulation, and synthesis - EE19B009
- Column IDCT Verilog Implementation, design testing and simulation, and synthesis - EE19B048
- AXI4 Lite Bus interfacing, module instantiating, defining memory map addresses and related functions for peripheral accessing - EE19B009, EE19B048
- Peripheral Performance Analysis - EE19B009, EE19B032, EE19B048
- Documenting the progress and findings of project - EE19B032

Reference(s):
1. https://github.com/YosysHQ/picorv32