Domain Selection & Software Reference

ELEC 5803

Jesse Levine (101185127)

1 Objective

The objective of this report and for phase 2 of the project is to select an application domain and apply a software reference to the baseline core.

In this report the baseline core is extended to be able to perform the multiplication operation, in order to be able to utilize the core for the chosen domain. The exteded core is then tested on a simplified test and its performance is then measured (Area, Clock Cycle, Cycle Count, etc..).

2 Domain Selection

The chosen application domain is Machine Learning, specifically Inference Acceleration. This project targets the acceleration of the Softmax operation, defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

While matrix multiplication dominates early layers of neural networks, the Softmax layer is the critical bottleneck in Transformer and Classifier architectures due to high-latency operations (exponentiation, division, and accumulation).

Calculating e^{z_i} via software emulation (Taylor series) requires dozens of cycles per element which necessiates a full vector summation before any single output can be normalized. Furthermore, divison latency, which is particularly costly on RV32i cores that lack native hardware division and multiplication.

By targetting Softmax, this project will demonstrate how specialized hardware can eliminate these serial bottlenecks and synchronization barriers.

3 Baseline Limitations

The baseline core adheres to the RV32I base integer instruction set. This ISA does not include native support for multiplication or division. As such, before attempting to write and test on a Softmax kernel, the baseline core was extended to the RV32IM ISA which includes hardware instructions for multiplication, which is discussed in the following Section 4.

4 Intruction Extension

The extension was implemented within the cpu function in riscv32i.cc. The decoding logic was updated to recognize the following parameters:

1. **Opcode:** 0x33 (OPCODE_R)
2. **Funct7:** 0x01 (FUNCT7_M)
3. **Funct3:** 0x00 (FUNCT3_MUL)

```
case OPCODE_R:
    // Support for RV32M: Multiplication Extension
    if (func7 == FUNCT7_M && func3 == FUNC3_MUL) {
        res = src1 * src2;
    } else {
        // Standard RV32I R-type instructions (ADD, SUB, SLT, etc.)
        switch (funcx) {
            case FUNCX_ADD: res = src1 + src2; break;
            case FUNCX_SUB: res = src1 - src2; break;
            // ...
        }
    }
    break;
```

5 Testing & Verification

TO verify the integration of the RV32M multiplication extension, a two-stage verification process was done: C-simulation and implementation resource analysis.

5.1 Kernel Test

A dedicated test kernel, mult.c was written. The program performs the following operations:

1. Loads two integer operands from memory addresses 0x100 and 0x104.
2. Executes the * operator. Because the RISC-V toolchain now targets the extended ISA, this mpas directly to the newly implemented OPCODE_R / FUNC3_MUL hardware path instead of a software library call.
3. Stores the results to 0x108 and triggers an exall to singal completion.

mult.c was then compiled to mult.txt:

PC	**Machine Code (Hex)**	**Instruction / Description**
0x00	`10002783`	`lw a5, 256(x0)` Load word from memory address 256 into `a5`
0x04	`10402703`	`lw a4, 260(x0)` Load word from memory address 260 into `a4`
0x08	`02e787b3`	`mul a5, a5, a4` Multiply `a5` and `a4`, store result in `a5` (RV32M)
0x0C	`10f02423`	`sw a5, 264(x0)` Store word from `a5` to memory address 264
0x10	`00000073`	`ecall` Environment call (system call / program exit)
0x14	`00008067`	`ret` Return from subroutine (`jalr x0, 0(x1)`)

Table 1: RV32IM machine code execution trace

5.2 CSIM

The testbench loaded the compiled mult.txt machine code (as seem in Table 1 into the HLS model's memory. the CSim successfully verified the hardware logic, confirming the decoder correctly idenitifes the instruction and the execution unit produces mathematically correct results:

Test #	**ECALL PC**	**Operation**	**Result**
1	0x00000010	1×1	PASS (1)
2	0x00000010	1×2	PASS (2)
3	0x00000010	10×10	PASS (100)
4	0x00000010	2×20	PASS (40)

Table 2: CSIM validation results for RV32IM multiplication program

5.3 Implementation & Resource Results

The design was synthesized and implemented for the Xilinx PYNQ-Z1 (xc7z020-clg400-1). The implementation resulted in the following resource implementation (Table 3):

Resource Type	**Usage**
SLICE	379
LUT	1302
FF	598
DSP	3
BRAM	2

Table 3: Post-implementation FPGA resource utilization

References