FPGA Lab – Memory Hierarchy

Objectives

In the second part, MicroBlaze will be expanded with an IO module to support multiple peripherals and a memory hierarchy.

1. Expand the IO bus, to support multiple peripherals, by adding a timer. This will be later used to measure application execution time.
2. Improve performance by adding hardware arithmetic support, and a data cache.
3. Study the performance over various architectural features using a matrix multiplication example and a handful of special kernels.

Adding multiple IO components

To add multiple peripherals to the IO bus, multiplexed access is needed. AXI infrastructure IP provides the necessary functionality for connecting blocks together. Some examples include:

1. NxM crossbars for connecting N master devices to M slave slave devices. The architecture of the crossbar can be optimised for resources through multiplexing or performance by reproducing paths so non-blocking transfers can occur concurrently.
2. AXI protocol conversion (e.g. conversion between AXI3 and AXI4, or AXI4-lite to AXI4), data width conversion and clock conversion blocks.
3. A variety of AXI data generators, performance monitors, and verification.

To expand the number of peripherals MicroBlaze can support, an AXI crossbar is required on the master data port. Add a crossbar and perform necessary rewiring to attach:

1. Debug module
2. AXI timer

Validate the design.

Adding DDR3 memory controller and Data Cache Interface

The Zybo Z7 board comes with 1GB of DDR3 memory. However, this memory is connected to the hardened memory controller in the realm of the processing system. Source the add_ext_memory.tcl script to update the block diagram with the pre-configured memory controller.

Using the advanced configuration tab inside Microblaze add a data cache of 16kB capacity, a line length of four words, and enable write access in write-through mode. Also, set/verify the base and high address to correspond to the region of the external memory.

Hard implementation for arithmetic units can significantly improve performance, check the boxes to include a barrel shifter and 64-bit integer multiplier.

Add a logic analyser to the cache port. Optionally add probes to the LMB and data bus.

Validate the design, generate the bitstream and export the .xsa file.

Software

Once again update the hardware platform with the new .xsa, and create a new project.

1. Copy the provided C source files for the naive matrix multiplication. Notice the timer.c/timer.h files contain basic functions for configuring the timer in 64-bit mode and measuring execution time between two points in a c program.
2. The vanilla matrix multiplication implementation is contained in matmul.c/matul.h. Run some tests using the following options:
    1. Optimise the code O0 → O3
    2. Enable/disable hardware multiplier/shifter through compiler options.
    3. Enable / Disable the caches to see their impact!
3. Set up the Logic Analyser in Vivado and observe AXI transactions on the cache bus while executing your c program.

Hint: Before running your benchmarks consider invalidating the cache. Small data arrays may already reside in the cache at the start of the experiment. This can happen if they were randomly generated in the previous step.

Homework – Part 2/2

By reconfiguring the hardware platform, performing compiler optimisations, code re-writing or otherwise, report on the characteristics and subsequently improve the baseline performance of kernels 1-4. Finally, discuss which architectural features improve or degrade each kernel's performance.

Kernel 1

```C
void kernel1 (int *A, int size, int offset) {
	int i;
	for(i = 0; i < size-offset; i++)
		A[i] += A[i +offset] ;
}
```

Kernel 2

```C
void kernel2 (int *A, int size) {
	int i;
	for(i = 3; i < size; i++)
		A[i] = A[i - 1] + A[i - 2] * A[i - 3];
}
```

Kernel 3

```C
void kernel3 (float *h, float *w, int idx, int size) {
	for (int i = 0; i < ARRAY_SIZE; ++i)
		h[idx[i]] = h[idx[i]] + w[i];
}
```

Kernel 4

```C
float kernel4 (float *A, float *B, int size) {
	float sum = 0;
	for (int i = 0; I < size; i++) {
		float diff = A[i] - B[i];
		if (diff > 0) sum = (sum + diff);
	}

	return sum;
}
```

Guidelines and Deliverables:

This part of the homework contributes the remaining 60% of the final mark.

Deliverables:

1. Hardware demonstration for both parts.
2. A short report. The report should include:
    1. Block diagram of the hardware platform(s)
    2. Maximum length of 5 pages, excluding appendix
    3. Modified source code with relevant comments as appendix

The homework can be attempted collaboratively in groups of maximum three people.

References

1. MicroBlaze Processor Reference Guide (UG984)