Lab 7: SMIPS with DRAM and Caches

Due: Friday November 14, 2014

1 Introduction

By now you have a 6-stage pipelined SMIPS processor with target and direction branch predictors. Unfortunately your processor is limited to running programs that can fit in a 256 KB FPGA block RAM. This works fine for the small benchmark programs we are running, such as a 250 item quicksort, but most interesting applications are (much) larger than 256 KB. Luckily the FPGA boards we are using have 1 GB of DDR3 DRAM accessible by the FPGA. This is great for storing large programs, but this may hurt the performance since DRAM has long latencies when reading data from them.

This lab will focus on using DRAM instead of block RAM for main program and data storage to store larger programs and adding caches to reduce the performance penalty from long latency DRAM loads.

First you will write a translator module that takes memory requests from the CPU and translates them to memory requests for DRAM. This module will enable a larger storage space for your programs, but it will see a large decrease in performance since your are reading from DRAM almost every cycle. Next you will implement a cache to reduce the amount of times you need to read from the DRAM, therefore improving your processors performance. Lastly you will synthesize your design for an FPGA and run very large benchmarks that require DRAM and very long benchmarks that require an FPGA.

2 DRAM Interface

The KC705 FPGA board you will use in this class has 1 GB of DDR3 DRAM. DDR3 memory has a 64 bit wide data bus, but 8 64 bit chunks are sent per transfer, so effectively it acts like a 512 bit wide memory. DDR3 memories have high throughput, but they also have high latencies for reads.

The Sce-Mi interface generates a DDR3 controller for us, and it takes in `MemoryClient` interface to connect to it. The typedefs provided for you in this lab use types from BSV's Memory package. Here are some of the typedefs related to DDR3 memory:

```
typedef 24 DDR3AddrSize;
typedef Bit#(DDR3AddrSize) DDR3Addr;
typedef 512 DDR3DataSize;
typedef Bit#(DDR3DataSize) DDR3Data;
typedef TDiv#(DDR3DataSize, 8) DDR3DataBytes;
typedef Bit#(DDR3DataBytes) DDR3ByteEn;
typedef TDiv#(DDR3DataSize, DataSize) DDR3DataWords;

// The below typedef is equivalent to this:
// typedef struct {
//     Bool        write;
//     Bit#(64)    byteen;
//     Bit#(24)    address;
//     Bit#(512)   data;
// } DDR3_Req deriving (Bits, Eq);
typedef MemoryRequest#(DDR3AddrSize, DDR3DataSize) DDR3_Req;

// The below typedef is equivalent to this:
// typedef struct {
//     Bit#(512)   data;
// } DDR3_Resp deriving (Bits, Eq);
typedef MemoryResponse#(DDR3DataSize) DDR3_Resp;

// The below typedef is equivalent to this:
// interface DDR3_Client;
//     interface Get#( DDR3_Req ) request;
//     interface Put#( DDR3_Resp ) response;
// endinterface;
typedef MemoryClient#(DDR3AddrSize, DDR3DataSize) DDR3_Client;
```

2.1 DDR3_Req

The requests for DDR3 reads and writes are different than the requests of `FPGAMemory`. The biggest difference is the byte enable, `byteen`.

- `write` – Boolean specifying if this request is a write request or a read request.
- `byteen` – Byte enable, specifies which bytes will be written or read. You will probably want to set this to all 1's. You can do that with the literal `'1` (note the apostrophe).
- `address` – Address for read or write request. DDR3 memory is addressed in 512-bit chunks, so address 0 refers to the first 512 bits, and address 1 refers to the second 512-bits. This is very different than the byte addressing used in the SMIPS processor.
- `data` – Data value used for write requests.

2.2 DDR3_Resp

DDR3 memory only sends responses for reads just like `FPGAMemory`. The memory response type is a structure instead of just `Bit#(512)` so you will have access the `data` field of the response in order to get the `Bit#(512)` value.

2.3 DDR3_Client

The `DDR3_Client` interface is made up of a `Get` subinterface and a `Put` subinterface. This interface is exposed by the processor, and the Sce-Mi infrastructure connects it to the DDR3 controller. You do not need to worry about constructing this interface because it is done for you in the example code.

2.4 Example Code

Here is some example code showing how to construct the FIFOs for a DDR3 memory interface along with the initialization interface for DDR3. This example code is provided in `DDR3Example.bsv`.

```
import GetPut::*;
import ClientServer::*;
import Memory::*;

// other packages and type definitions

module mkProc(Proc);
    Fifo#(2, DDR3_Req)  ddr3ReqFifo  <- mkCFFifo;
    Fifo#(2, DDR3_Resp) ddr3RespFifo <- mkCFFifo;
    MemInitIFC ddr3InitIfc <- mkMemInitDDR3( ddr3ReqFifo );
    Bool memReady = ddr3InitIfc.done;

    // optional: easy to use wrapper and interfaces for ddr3 FIFOs
    WideMem wideMemWrapper <- mkWideMemFromDDR3( ddr3ReqFifo, ddr3RespFifo );
    Vector#(2, WideMem) wideMems <- mkSplitWideMem( wideMemWrapper );
    // Data cache should use wideMems[0]
    // Instruction cache should use wideMems[1]

    // lots of other code

    // This rule is needed for running the lab on the FPGA
    rule drainMemResponses( !cop.started );
        ddr3RespFifo.deq;
    endrule

    // other interface methods

    interface WideMemInitIfc memInit = ddr3InitIfc;
    interface DDR3_Client ddr3client = toGPClient( ddr3ReqFifo, ddr3RespFifo );
```

```
endmodule
```

2.5 Sharing the DRAM Interface

The example code exposes a single interface with the DRAM, but you have two modules that will be using it: an instruction cache and a data cache. If they both send requests to `ddr3ReqFifo` and they both get responses from `ddr3RespFifo`, it is possible for their responses to get mixed up. To handle this, you need a separate FIFO to keep track of the order the responses should come back in. Each load request is paired with an enqueue into the ordering FIFO that says who should get the response.

To simplify this for you, you have been given modules that you can use to split the DDR3 FIFOs into two `WideMemory` interfaces. These modules have been included in the sample code above. If you choose to use them, you need to look at the code and interfaces yourself to figure out how they function and how to use the interfaces.

3 Migrating Code from Previous Lab

The provided code for this lab is very similar, but there are a few differences to note. Most of the differences are displayed in the provided example code `DDR3Example.bsv`.

3.1 Modified Proc Interface

The Proc interface now only has a single memory initialization interface to match the unified DDR3 memory. The width of this memory initialization interface has been expanded to 512 bits per transfer. The new type of this initialization interface is `WideMemInitIfc` and it is implemented in `WideMemInit.bsv`.

3.2 Empty Files

The two processor implementations for this lab: `WithoutCache.bsv` and `WithCache.bsv` are initially empty. You should copy over the code from either `SixStageBHT.bsv` or `SixStageBonus.bsv` as a starting point for these processors. `Bht.bsv` is also empty, so you will have to copy over the code from the previous lab for that too.

4 `WithoutCache.bsv` – Using the DRAM Without a Cache

Exercise 1 (10 Points): Implement a module `mkTranslator` in `Cache.bsv` that takes in some interface related to DDR3 memory (`WideMemory` for example) and returns a `Cache` interface. This module should not do any caching, just translation from `MemReq` to requests to DDR3 (`WideMemReq` if using `WideMemory` interfaces) and translation from responses from DDR3 (`CacheLine` if using `WideMemory` interfaces) to `MemResp`. This will require some internal storage to keep track of which word you want from the cache line that comes back from main memory. Integrate `mkTranslator` into a six stage pipeline in the file `WithoutCache.bsv`. You can build this processor by running `build -v withoutcache` from `scemi/sim/`, and you can test this processor by running `./run_assembly withoutcache` and `./run_benchmarks withoutcache` from `scemi/sim/`.

Discussion Question 1 (5 Points): Record the results for `./run_benchmarks withoutcache`. What IPC do you see for each benchmark?

5 `WithCache.bsv` – Using the DRAM With a Cache

By running the benchmarks with simulated DRAM, you should have noticed that your processor slows down a lot. You can speed up your processor again by remembering previous DRAM loads in a cache as described in class.

Exercise 2 (20 Points): Implement a module `mkCache` to be a direct mapped cache that allocates on write misses and writes back only when a cache line is replaced. This module should take in a `WideMem` interface (or something similar) and expose a `Cache` interface. Use the `typedefs` in `CacheTypes.bsv` to size your cache and for the `Cache` interface definition. Incorporate this cache in the same pipeline from `WithoutCache.bsv` and save it in `WithCache.bsv`. You can build this processor by running `build -v withcache` from `scemi/sim/`, and you can test this processor by running `./run_assembly withcache` and `./run_benchmarks withcache` from `scemi/sim/`.

Discussion Question 2 (5 Points): Record the results for `./run_benchmarks withcache`. What IPC do you see for each benchmark?

6 Running Large Programs

This section was added on November 9th.

By adding support for DDR3 memory, you processor can now run larger programs than the small benchmarks we have been using. Unfortunately, these larger programs take longer to run, and in many cases, it will take too long to wait for the simulation to finish. Now is a great time to try FPGA synthesis. By implementing your processor on an FPGA, you will be able to run these large programs much faster since the design is running in hardware instead of software.

Exercise 3 (0 Points, but you should still totally do this): Before synthesizing for an FPGA, lets try looking at a program that takes a long time to run in simulation. The program `./run_mandelbrot` runs a benchmark that prints a square image of the Mandelbrot set using 1's and 0's. Run this benchmark to see how slow it runs in real time. Please don't wait for this benchmark to finish, just kill it early using ctrl-c.

6.1 Synthesizing for FPGA

There is some extra code in the course locker to enable FPGA synthesis. To get this code, go to the `scemi` directory and run the following command:

```
cp -r /mit/6.175/extra-lab-code/lab7/fpga_kc705 .
```

Now that you have the code, you can start FPGA synthesis for `WithCache.bsv` by going into the `fpga_kc705` folder and executing the command:

```
vivado_setup build -v
```

This command will take a lot of time (about one hour) and a lot of resources. You will probably want to select a vlsifarm server that is under a light load. You can see how many people are logged in with `w` and you can see the resources being used with `top`.

Once this has completed, you can submit your FPGA design for testing on the shared FPGA board by running the command `./submit_bitfile` and you can check the results with `./get_results`. These results will look similar to the simulation results, but they will include larger benchmarks (`*.large.vmh`) from `/mit/6.175/fpga-programs`.

If you want to check the status of the FPGA, you can run the command `./fpga_status`.

Exercise 4 (10 Points): Synthesize `WithCache.bsv` for the FPGA and send your design to the shared FPGA for execution. Get the results for the normal and large benchmarks and add them to discussion.txt

Discussion Question 3 (10 Points): How many cycles does the Mandelbrot program take to execute in your processor? The current FPGA design has an effective clock speed of 50 MHz. How long does the Mandelbrot program take to execute in seconds? Estimate how much of a speedup you are seeing in hardware versus simulation by estimating how long (in wall clock time) it would take to run `./run_mandelbrot` in simulation.

6.2 Note on the FPGA from the TA

The shared FPGA for this class is running in a computer that is sitting under my desk in my office. If you are interested in seeing the board, or seeing it run in real-time, drop by office hours or make an appointment to come by some other time.

Also, if you have any problems with it, please e-mail me as soon as possible. The infrastructure is not very stable, but it is easy for me to fix if you let me know as soon as you have a problem.