University of Pennsylvania
Department of Electrical and Systems Engineering
Electronic Design Automation

ESE535, Spring 2009 — Assignment #2 — Monday, February 2

Part A Due: Monday, February 16, beginning of class.
Part B Due: Wednesday, February 25, beginning of class.

Resources: You are free to use any books, articles, notes, or papers as references. Provide citations in your writeup as appropriate.

Collaboration: Please work independently on this assignment. You may discuss general algorithmic strategies and help each other with the compiler, build environment, and debugging, but each student should develop his or her own solution. If you do discuss strategy or getting debugging help, please acknowledge it in your writeup.

Writeup: Writeup should be in an electronically readable format (HTML or PDF preferred—I do not want to decipher handwriting or hand-drawn figures). State any assumptions you need to make.

Part A

Develop and implement a scheduling routine for minimizing the time required to evaluate a dataflow graph on a fixed set of heterogeneous resources. We provide the infrastructure and a sample scheduling routine (fcfs). You only need to provide an alternate scheduler. Note that the distribution of operator types is provided as an input to the program. Your scheduler should work for any distribution provided.

Turnin:

1. Your code (no binary files, but in an archive like the provided support so it can be unpacked and built; if you add any files, the `make` file should be updated to build the application with your additions)

2. Explanation of your scheduling strategy and a description of your algorithm and the code that supports it

3. Summary of your results on the provided benchmark set. (table with columns: benchmark, provided fcfs cycles, your cycles, % improvement, you scheduler runtime in seconds; a final row should summarize the geometric mean of your improvements across the benchmark set.) Provide two tables for the two resource cases listed in the provided `make` file.

4. Explanation of any tuning, benchmarking, and profiling you performed to arrive at your current solution. Where appropriate show quantitative graphs or tables.

5. If you add any options or otherwise change how the code is run, provide an explanation of how to run your scheduler.

Part B

Develop and implement a routine to minimize the time required to evaluate the dataflow graph by selecting the number of resources of each type obeying an area bound. This extends the problem above. Again, we provide a simple example that performs this provisioning and you need to provide a superior alternative.

Turnin:

1. Your code (as with part A)
2. Explanation of your scheduling strategy and a description of your algorithm and the code that supports it
3. Summary of your results on the provided benchmark set. (table with columns: benchmark, provided uniform ops cycles, your selected operator distribution, your cycles, % improvement, you scheduler runtime in seconds; a final row should summarize the geometric mean of your improvements across the benchmark set.) Provide two tables for the two area bounds listed in the provided `make` file.
4. Explanation of any tuning, benchmarking, and profiling you performed to arrive at your current solution. Where appropriate show quantitative graphs or tables.
5. If you add any options or otherwise change how the code is run, provide an explanation of how to run your provisioning scheduler.

Operator Model The model for operators for this assignment is that they are all internally pipelined to accept one new set of inputs on each cycle. Since they are pipelined, they do not produce a result until several cycles later (*e.g.* the adder takes 8 cycles to produce a result); you cannot use a result until it has had time to go all the way through the operator pipeline. Each different operator type has a different number of pipeline stages. This information is recorded in the library (modeled by `library.c`). No extra time is charge for communication among operators for this assignment. This particular library models double-precision floating point operations implemented at high throughput (around 300 MHz) on a modern FPGA.

Provided Code Base Pickup the code in `assign2.tar` from `~ese535/spring2009/assign2.tar` on `eniac`. Unpack it with `tar -xvf assign2.tar`. Run `make` to build. This should produce two executables `driver2a` and `driver2b` that you can run. With no arguments either driver will produce a usage message. The `Makefile` in the `test` subdirectory runs each driver on a few test cases and provides an example of how to use it. At the moment there are only two example input files. We will update `assign2.tar` to include additional test cases in the next few days. You will need to change the `-scheduler` and `-provision` options in the `test/Makefile` to get it to run your code.

For this assignment, we provide you a stub for your scheduler in the file `your_scheduler.c` and your provisioner in the file `your_provision.c`. Currently these routines do nothing useful. You should complete the routines for Parts A and B respectively.

A quick overview of code:

- `driver2a.c` — contains the `main` function which drives the fixed resource scheduler. You will most likely **not** need to modify the driver, but may do so if you find it useful.
- `driver2b.c` — contains the `main` function which drives the area-bound provision scheduler. You will most likely **not** need to modify the driver, but may do so if you find it useful.
- `graph.c`, `graph.h` – represents the dataflow graph and includes code for reading the graph. You will definitely need to work with the graph, so you will need to use these routines. You may want to add fields to the graph or graph nodes to support your algorithm.
- `library.c`, `library.h` – this contains information about the physical operators onto which you are scheduling. In a real system, the library would be read from a file or a database. You will need to use these routines to find out the number of pipeline stages in each operator and the area of each operator.
- `platform.c`, `platform.h` – this records the provisioned resource set, holds the schedule you create, and can print the schedule. You will need to use routines here to perform your actual assignment of graph nodes to operators and time slots.
- `check.c`, `check.h` – code to check if a schedule is valid. You should not need to touch this code. It is called from both drivers to validate the schedule. This exists to help both you and I check that the schedules you produce are complete and legal.
- `fcfs.c`, `fcfs.h` – this is an implementation of a **dumb** first-come-first-serve scheduler in the spirit of the psuedocode shown on Assignment 1. It is provided (1) to give you

something to beat and (2) to illustrate how to use all the other routines (*e.g.* graph, platform, queue).

- `uniformmops.c`, `uniformmops.h` – like fcfs, this is a **dumb** provisioner that simply implements n of each operator for the largest n that fits in the allowed area. Again, this illustrates the form of the result for the provisioner and its use of other routines.
- `queue.c`, `queue.h` – a simplistic queue used by the fcfs scheduler that may or may not be useful to you. Note the restrictions stated in the source.
- `util.c`, `util.h` — various utilities. You should not need to touch. You may or may not want to use some of these utilities.

Caveat: The code was newly written for this assignment. While I have tried to test it, like any recently developed code it may contain bugs. Let me know if you have any problems. Similarly, I may need to provide updated source as I fix bugs or add additional functionality.

Graph File Format The provided code `graph.c` will take care of reading the graph files. Nonetheless, it may be useful for you to read the .gph files. The first line is the number of nodes in the graph. Each line after the first describes a node in the graph. The first number is the identifier for the node. The second is the operator type (corresponding to the operator types defined in `library.h`. The remaining numbers node identifiers for the inputs to the node. Input nodes have no inputs. Operations like add, multiply, and divide have two inputs, while sqrt, exp, and log have a single input. These are all defined in `library.c`. The # symbol is the comment character. Everything after the # symbol is a comment.