LSU EE 4755 Homework 5 Due: 22 November 2022

For instructions visit `https://www.ece.lsu.edu/koppel/v/proc.html`. *For the complete Verilog for this assignment without visiting the lab follow* `https://www.ece.lsu.edu/koppel/v/2022/hw05.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw05.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Assignment Background

As we should know the synthesis program, given a Verilog description of a module, writes a design file with an optimized version of the module mapped to the chosen technology. For this assignment the chosen technology is the same Oklahoma University ASIC process we've been using throughout the semester.

An important skill for those writing Verilog descriptions is to estimate the cost and performance of those synthesized modules. In this assignment we'll look at how well the synthesis program handles the different modules we considered for computing the floating-point expression $v_0^2 + v_0 v_1 + v_1^2$. We will consider the combinational, sequential, and pipelined modules covered in class.

A synthesis script will be used to synthesize these modules, plus three arithmetic unit modules, plus additional modules created for the solution to this problem. To complete the assignment the output of the script must be understood and the synthesis script must be modified. The output of the synthesis script is similar to the output of the scripts used in prior assignments, so it should be familiar. Modifying the script will be something new, and might be a challenge for some of you. It is okay to seek help modifying the script from classmates and others, though the solutions to the problems themselves must be completed individually.

Modules

This assignment includes modules for the combinational, sequential, and pipelined implementations of the multi-step computation. They are named `ms_comb`, `ms_seq`, and `ms_pipe`. For comparison the assignment also includes modules containing a single floating-point unit, they are named `try_mult`, `try_add`, and `try_sq` (square).

Four additional modules are provided for experimentation, `m1_func`, `m1_comb`, `m1_seq`, and `m1_pipe`. These modules initially perform the computation $v_0 + v_0 v_1 + v_1^2$, but they can be modified to perform other computations. Module `m1_func` is used by the testbench to obtain a correct value, so modify it first so that it computes the desired computation. Then modify the others that you want to synthesize. (The synthesis program does not care whether a module passes the testbench, but no conclusion can be drawn from the area and delay of module that does not work correctly.)

All of these modules have the same parameters and ports, though not every module uses every port. For example, only `ms_seq` and `ms_pipe` are sequential so that the `clk` and `reset` ports on the others serve no function. These unused ports will be eliminated during optimization so they won't affect cost or timing.

Module Parameters and Floating Point Format

The modules used in this assignment all have the same parameters, these parameters specify the floating-point number format to be used. The first parameter, `wsig`, specifies the number of bits in the significand (fractional part) of the floating point number. The default value is 23, which is the same as an IEEE 754 single (C `float`). The second parameter, `wexp`, is the number of bits in the exponent. The default value is 8, which matches an IEEE single. The third parameter, `ieee`, specifies whether the IEEE floating-point format should be strictly followed. The default value

is 1, which means yes; a 0 means that special cases do not have to be handled correctly. These include `NaN` (not a number) and subnormal values. The size of the floating point number using these parameters is `1+wexp+wsig`, the extra `1` is for the sign bit.

For this assignment all modules are instantiated with `ieee=0`. This is done to explore the fuller range of optimization possibilities and also to reduce the time needed for synthesis.

The sample synthesis runs consider two formats, IEEE single in which `wsig=23` and `wexp=8`, and the ML-friendly BF16 (informally known as brain float) in which `wsig=7` and `wexp=8`. The advantage of BF16 for machine learning is that it is half the size of a single, and with a 7-bit significand, requires half the energy for multiplication than the older 16-bit FP16 format. For us the big advantage is that it takes less time to synthesize than a single.

Testbench

The testbench exercises the six modules, `ms_comb`, `ms_seq`, `ms_pipe`, `m1_comb`, `m1_seq`, and `m1_pipe` instantiated with a significand size of 7 and 23. They should all initially pass. As with other testbenchs in this class, a line will be printed for the first few module errors, and a tally will be provided for each module and size. The testbench uses `ms_func` to determine the correct output of the `ms` modules and `m1_func` to determine the correct output of the `m1` modules. When modifying the `m1` modules be sure to also modify `m1_func` so that the testbench can show you whether your modified modules do what you think they are doing.

The Synthesis Script

As with past assignments, the modules in the assignment file should be synthesized using the script `syn.tcl`. Unlike other assignments, this script will have to be modified.

The synthesis script itself is written in TCL (Tool Control Language, the abbreviation is pronounced tickle) a scripting language chosen by Cadence for scripting their EDA software. (Nowadays Python would be used. If it were up to me it would be Perl. But it's TCL.) Documentation for TCL can be found at `https://tmml.sourceforge.net/doc/tcl/`. This describes TCL, not the functionality needed to run Genus or other tools. For Genus-specific commands see the synthesis documentation linked to `https://www.ece.lsu.edu/koppel/v/ref.html`.

For this assignment it should not be necessary to use new Genus commands, just to change which modules are synthesized and which parameters to instantiate with. For that, one needs only a rudimentary knowledge of TCL, perhaps what can be learned just by looking at `syn.tcl`.

The synthesis script starts by setting some script variables, using the TCL `set` command, and by setting Genus attributes, using the Genus `set_db` command:

```
set verilog_source hw05.v
set syn_level "high"
set spew_file "spew.log"
set report_file "syn-report.log"
set_db syn_global_effort $syn_level
set rpt_chan [open $report_file w]
puts "Synthesizing at effort level \"$syn_level\"\n"
```

As one might guess `syn_level` is the amount of effort used for synthesis. Possible values are `none`, `low`, `medium`, and `high`. These initial lines are followed by the definition of a TCL procedure `syn_mod`, which emits the commands needed to synthesize a module, followed by commands to retrieve the area and delay of the synthesized module. A line of text is written showing the area and delay. It should not be necessary to modify `syn_mod` for this assignment.

Module `syn_mod` is called in a loop nest near the end of the file:

```
# List of combinational modules.
```

```
set mods_comb { ms_comb try_mult try_add try_sq }
set delay_targets { 100 0.1 }
set mods { try_mult try_add try_sq }
set mods { ms_comb ms_seq ms_pipe try_mult try_add try_sq }
set wsigs { 7 14 23 }

foreach delay_target $delay_targets {
    foreach ws $wsigs {
        foreach mod $mods {
            syn_mod $mod $delay_target " $ws 8 0 "
        }
    }
}
```

The loop nest above synthesizes each of the modules listed in `mods` (that's the inner loop). Each of these six modules is synthesized for each significand size found in `wsigs`. These modules are synthesized with each delay constraint in `delay_target`. For the code above there would be a total of $2 \times 6 \times 3$ synthesis runs. That would probably take hours.

The first `set` line writes variable `mods_comb` with a list of combinational modules. This variable must be updated with any new combinational modules that you use. Variable `mods` is set twice, first to a list of the arithmetic modules, then those are replaced with a list of the arithmetic modules and our multi-step modules. (Because of the second assignment the first assignment has no effect.) If one wanted to only synthesize the arithmetic modules one would comment out the second `mods` line. There is no need to use a loop nest. It is possible to write a `syn_mod` call for each synthesis, for example:

```
set delay_targets { 100 }
set wsigs { 7 14 23 }

syn_mod try_mult 5 "7 8 0"
syn_mod try_mult 5 "7 6 0"

# Exit before the loop nest.
close $rpt_chan
quit
foreach delay_target $delay_targets {
```

The example above does two synthesis runs. The `5` is the delay target and the quoted part are the parameters. (The parameters must be quoted so that they are read as a single argument to `syn_mod`.) In the example above, `try_mult` is synthesized with two exponent sizes, 8 bits and 6 bits, both are synthesized with a delay target of 5 ns.

To synthesize a new module (for example, one you wrote) add the name to one of the `mod` lists, or just use the name on a direct call to `syn_mod` as in the example above. **Iff the module is combinational** add the module to `mods_comb`. Not adding a combinational module to `mods_comb` will result in an error. Adding a sequential module to `mods_comb` will result in incorrect timing.

Synthesis Script Output

The synthesis script `syn.tcl` is run using the command `genus -files syn.tcl`. The run starts with a substantial amount of header output, including warnings, copyright information, and system information. Some is shown below:

```
[cyc.ece.lsu.edu] % genus -files syn.tcl
```

```
2022/11/13 16:52:05 WARNING This OS does not appear to be a Cadence supported Linux configuration.
2022/11/13 16:52:05 For more info, please run CheckSysConf in <cdsRoot/tools.lnx86/bin/checkSysConf <productId>
TMPDIR is being set to /tmp/genus_temp_566634_cyc.ece.lsu.edu_koppel_nvftYI
Cadence Genus(TM) Synthesis Solution.
Copyright 2022 Cadence Design Systems, Inc. All rights reserved worldwide.
Cadence and the Cadence logo are registered trademarks and Genus is a trademark
of Cadence Design Systems, Inc. in the United States and other countries.

[16:52:12.338826] Configured Lic search path (21.01-s002): /apps/linux/cadence/share/license/license.dat:/opt/pgi/license.dat
```

The output of the script proper (as opposed to Genus, the synthesis program) starts with an announcement of the synthesis effort level followed by a table of synthesis results:

```
Synthesizing at effort level "high"

Module Name                              Area   Delay   Delay     Synth
                                              Actual  Target     Time
ms_comb_wsig7_wexp8_ieee0              600190  12.219     0.1 ns    423 s
ms_seq_wsig7_wexp8_ieee0               445400   5.754     0.1 ns    236 s
ms_pipe_wsig7_wexp8_ieee0              797327   5.678     0.1 ns    309 s
ms_comb_wsig14_wexp8_ieee0            1363980  14.391     0.1 ns    707 s
```

Each line of the table shows the result of one synthesis run. The `Module Name` column shows the name of the module followed by the parameter values used in its instantiation. In the sample above three different modules are synthesized, `ms_comb`, `ms_seq`, and `ms_pipe`. Module `ms_comb` is synthesized once with significand of 7 bits and once with a significand of 14 bits.

The `Area` *column* shows the area given by the Genus `report area` command. The units are relative to the OSU technology. *The* `Delay Actual` *column* shows the length of critical path through the module in units of nanoseconds. *The* `Delay Target` *column* shows the delay constraint that the synthesis program was set to meet. In the example above the constraint is 0.1 ns, which means the critical path can be no longer than 0.1 ns. This constraint was intentionally set to an impossibly low value, to determine the minimum delay that the synthesis program could achieve. Normally the delay constraint is set to something achievable, perhaps 4 ns in the example above, and the synthesis program would generate the least expensive design that meets the delay constraint. *The* `Synth Time` *column* shows the wall-clock (elapsed) time needed to perform the synthesis. The wall-clock time is shown to help plan the synthesis runs, it does not directly affect or describe the design itself.

Problem 1: In class we considered three ways of implementing `multi_step`, the modules that computed $v_0^2 + v_0 v_1 + v_1^2$: A combinational version, a sequential version, and a pipelined version. Appearing below are the results from synthesizing these three modules, named `ms_comb`, `ms_seq`, and `ms_pipe`, followed by results of synthesizing modules consisting only of the Chipware floating-point multiplier, adder, and a multiplier with the same value used for both operands. These are synthesized with a large delay constraint, meaning that the cost has been minimized.

Module Name	Area	Delay Actual	Delay Target	Synth Time
ms_comb_wsig23_wexp8_ieee0	1597692	75.142	100.0 ns	229 s
ms_seq_wsig23_wexp8_ieee0	945919	29.324	100.0 ns	111 s
ms_pipe_wsig23_wexp8_ieee0	1866509	28.273	100.0 ns	205 s
try_mult_wsig23_wexp8_ieee0	525991	28.231	100.0 ns	62 s
try_add_wsig23_wexp8_ieee0	339036	27.396	100.0 ns	53 s
try_sq_wsig23_wexp8_ieee0	297753	25.504	100.0 ns	38 s
ms_comb_wsig7_wexp8_ieee0	375767	34.708	100.0 ns	75 s
ms_seq_wsig7_wexp8_ieee0	275858	15.305	100.0 ns	34 s
ms_pipe_wsig7_wexp8_ieee0	526000	14.466	100.0 ns	62 s
try_mult_wsig7_wexp8_ieee0	94274	9.346	100.0 ns	13 s
try_add_wsig7_wexp8_ieee0	140221	14.196	100.0 ns	21 s
try_sq_wsig7_wexp8_ieee0	57802	6.085	100.0 ns	8 s

(*a*) Based on the data above, show the latency and throughput of each module for the 23-bit significand. It might be necessary to look at the module descriptions (Verilog code) to answer this question.

(*b*) For each of the two significand sizes, show that the delay of the three `ms` modules are what one would expect given the delays of the three arithmetic modules.

(*c*) Using the cost of the arithmetic units, show that the cost of `ms_comb` is lower than expected, but the cost of `ms_seq` and `ms_pipe` are about or perhaps a little more than what one would expect.

Problem 2: It is welcome that the cost of `ms_comb` is lower than what one would expect based on the cost of the arithmetic units. There are several possible reasons for this, for example the synthesis program may be simplifying the two adders used in computations such as $a + b + c$ or it may be sharing hardware used to process the common b operand in expressions like $a \times b$ and $b \times c$, or perhaps it may even be transforming $v_0^2 + v_0 v_1 + v_1^2$ into $(v_0 + v_1)^2 - v_0 v_1$. Or maybe the costs for the arithmetic units shown in the table are higher than they should be.

Perform a set of synthesis runs to provide evidence for a reason that `ms_comb` cost less than its constituent parts. Consider the possible reasons given above, or one of your own. These synthesis runs can operate on one of the existing modules, a slightly modified version of the modules, or something wholly different. The modules `m1_comb`, `m1_seq`, `m1_pipe` can be used for experimentation. See the Modules section above.

Describe the results of these experiments and how they convincingly support a particular reason for the lower cost. Data from a single synthesis run, or a series of very similar runs will not be considered convincing.

The Verilog file for this assignment will be collected, but submit the answers to this question on paper or by E-mail. Please E-mail PDF files. Sending word processor source files as a final product is unprofessional, even if they are TEX files.

In your writeup:

- Indicate how you believe the synthesis program is optimizing `ms_comb`.
- Describe the modules you synthesized to come to this conclusion, and the results of synthesis. Most credit will be given for this part of the assignment.
- Explain why your experiments show that the lower cost was not due to other optimizations.