Computation II - 5EIB0
mMIPS lab 2: changing the critical path v2.1

In case of a branch instruction (e.g. jump, branch equal, etc.) the Branch Control unit (`branch_ctrl` in the mMIPS decides whether to jump to another address. The `branch_ctrl` module is located in the MEM stage (see the mMIPS schematic). After every branch instruction the LCC compiler makes sure there is still one instruction to be executed before the jump to another address is made (if needed), or if no such instruction could be found, a NOP is inserted. Since the decision whether to take the branch or not is made in the MEM stage, the mMIPS doesn't know whether to fetch instructions from the new address (i.e. the branch is taken) or from the old address (i.e. the branch is not taken). This means that a so called branch hazard occurs for two cycles and thus the hazard module has to inserts two NOP instructions.

A good way to improve the performance of the mMIPS is by reducing the number of hazards that occur. In this lab we will reduce the number of branch hazards. To do this, we will move the `branch_ctrl` module from the MEM stage to the EX stage. We will also look at synthesis in order to determine the critical path of the mMIPS and the way this path is affected by hardware changes.

Note: It is very important to make regular backups of your work. For the final assignment it is important you keep track of the improvements in cycles and frequency of every change you make to the design.

1 Synthesis and the critical path

The maximum frequency at which your mMIPS can run is determined by the critical path. The critical path is the longest path from the output of a register (latch/flip flop) through combinatorial logic to the input of a register. The delay of this critical path is partially caused by logic (i.e. the path through logical gates such as an adder module) and for the other part by routing (i.e. the time for the signals to travel over the wires).

When Vivado synthesizes the Verilog code, it determines the critical path and the worst negative slack. The worst negative slack is the time that the critical path has left until the next clock pulse. These values are calculated using a clock frequency of $100MHz$ (This is the standard clock frequency of the fpga) and can be found in the utilization report and timing summary. The worst negative slack T_{WNS} together with the set clock frequency f_s determines the maximum frequency ($f = 1/((1/f_s) - T_{WNS})$). The total execution time is given by $T = fC$. Because we want to minimize the total execution time, maximizing the maximum clock frequency is one of our main goals. In order to track our progress, we will first synthesize the original mMIPS. This allows us to determine the maximum clock frequency f of the original mMIPS design.

1. Open the mMIPS (which you also used in lab 1) in vivado by running the command `make vivado`.
2. Synthesize the mMIPS by clicking on "Run Synthesis". You will now see some log output in the console. You will get a warning that some ports are not connected, this is normal.
3. When the synthesis has completed, you should open the "Utilization Report" and "Timing Summary". This can be done by clicking on "Report Utilization" and "Report Timing Summary". Use the default settings in the popups.
4. Take a look at the reports and locate the following information:
   - The percentage of slice LUTs that is used in the design. (see the "Utilization" report) (you can display percentages by clicking on the "%")
   - The Worst Negative Slack (WNS). (see the "Timing Summary")
   - The critical path of the "clk" signal. (click on the Worst Negative Slack number)

5. At which registers does the critical path start and end? What logic can be found in between these registers? What percentage of the delay on the critical path is due to logic? What percentage is due to the route? Using this information you should locate the critical path on the mMIPS schematic.

6. Calculate the maximum clock frequency using the Worst Negative Slack and a set clock frequency of $100MHz$.

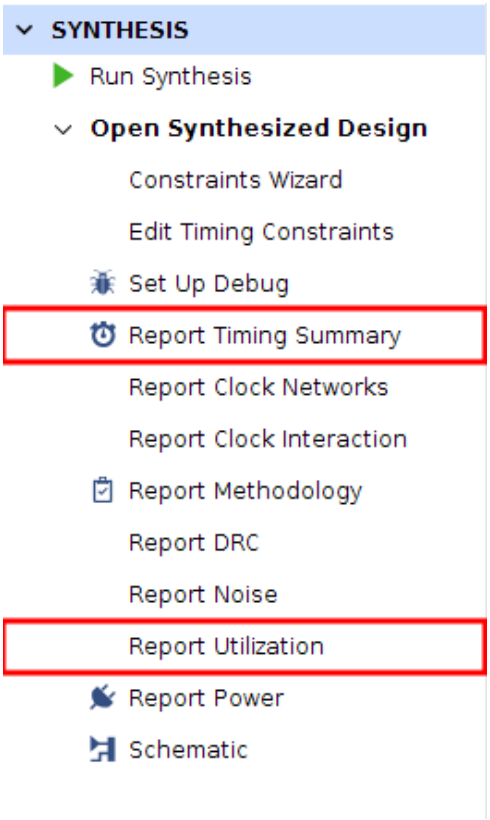

Figure 1: Synthesize

2 Moving the Branch Control module

In order to move the `branch_ctrl` module, we have to remove some registers and wires (registers between the EX and MEM stage) and reconnect the branch control to the proper wires in the EX stage. If all is done correctly, the branching decision will be made in the EX stage. Note that the mMIPS will however not function properly. We should also adapt the hazard module. This will be done in the last steps of this exercise.

Make sure you have a backup of your work from lab 1 before starting this exercise! If you copied the complete directory make sure to run 'make clean-vivado' in the new folder before opening vivado again.

All registers/wires that are to be deleted are marked red in figure 2.

1. Open the file `mmips.v`. In this file all modules and wires connecting the relevant modules are defined (excluding the `dmem` and `imem` modules).

2. From this file, delete the following modules and signals:

   - Register module `ex_alu_zero`.
   - Register module `ex_mux5`.
   - Register module `ex_ctrl_mem_branch`.
   - Wire `bus_ex_alu_zero`.
   - Wire `bus_ex_mux5`.

- Wire `bus_ex_ctrl_mem_branch`.

3. In the same file, you should reconnect the signals that were the input to these registers to the `branch_ctrl` and `mux1`.
   - Connect `bus_alu_zero` to the `AluZero` input of module `branch_ctrl`.
   - Connect `bus_id_ctrl_mem_branch` to the `BranchOp` input of module `branch_ctrl`.
   - Connect `bsu_mux5` to the `in1` input of the module `mux1`.
4. There is still an input to the `hazard` module unconnected. Previously the wire `bus_ex_ctrl_mem_branch` was connected to this input. Why was this wire connected to the `hazard` module? This wire (and so the input port) can be removed from the mMIPS. Why? To answer these questions you should also look at `hazard.v` and try to understand how the `hazard` module detects branch hazards. (Note: the `hazard` module is already connected to `bus_id_ctrl_mem_branch`).
5. Adapt the `hazard` module such that it works correctly with the new placement of the branch control module. That is, the `hazard` module should only insert 1 NOP instruction after a branch instruction. The next instruction still needs to be prefetched in the last branch hazard cycle (which is one cycle sooner as before!). In addition you should remove the rudimentary input of the `hazard` module and it's connection to `bus_ex_ctrl_mem_branch`.
6. Try and simulate your new design. Fix any errors (if needed). Verify that the mMIPS works correctly by comparing the output dump file to the reference dump you created in lab 1. *Hint: to quickly check for errors, modify the filter code so it runs on a smaller portion of the image (e.g. 8x8 pixels) and compare it with a reference output (based on an image with the same dimensions) of the original mMIPS. However, always check your hardware and software also with the full program to ensure that you do not overlook some bugs!*

3 Determining the performance gain

The performance gain is defined as the (percentage) reduction of the total execution time of a program running on the processor. The total execution time T is defined as the number of cycles C divided by the maximum frequency f of the processor: $T = C/f$. Note that T is in general not the same as the simulation time given by icarus after a finished simulation. icarus computes this simulation time using a theoretical clock frequency which is not related to the actual maximal clock frequency of your mMIPS. The latter frequency can only be obtained after synthesis. The performance gain G is defined as: $G = \frac{T_{original} - T}{T_{original}} \cdot 100\%$, in which case a performance gain of 100% means the program runs in half the time compared to the running time on the original processor.

1. Synthesize the top level module of your modified mMIPS. Determine the maximum clock frequency f from the synthesis report.
2. Calculate the performance gain using the number of clock cycle C and maximal frequency f of the original mMIPS and your modified mMIPS.
3. What is the critical path of your new mMIPS design? Did the critical path change after you moved the branch ctrl module to a different pipeline stage? How could this critical path be improved further? What happens to the performance gain when you apply this change to the critical path?

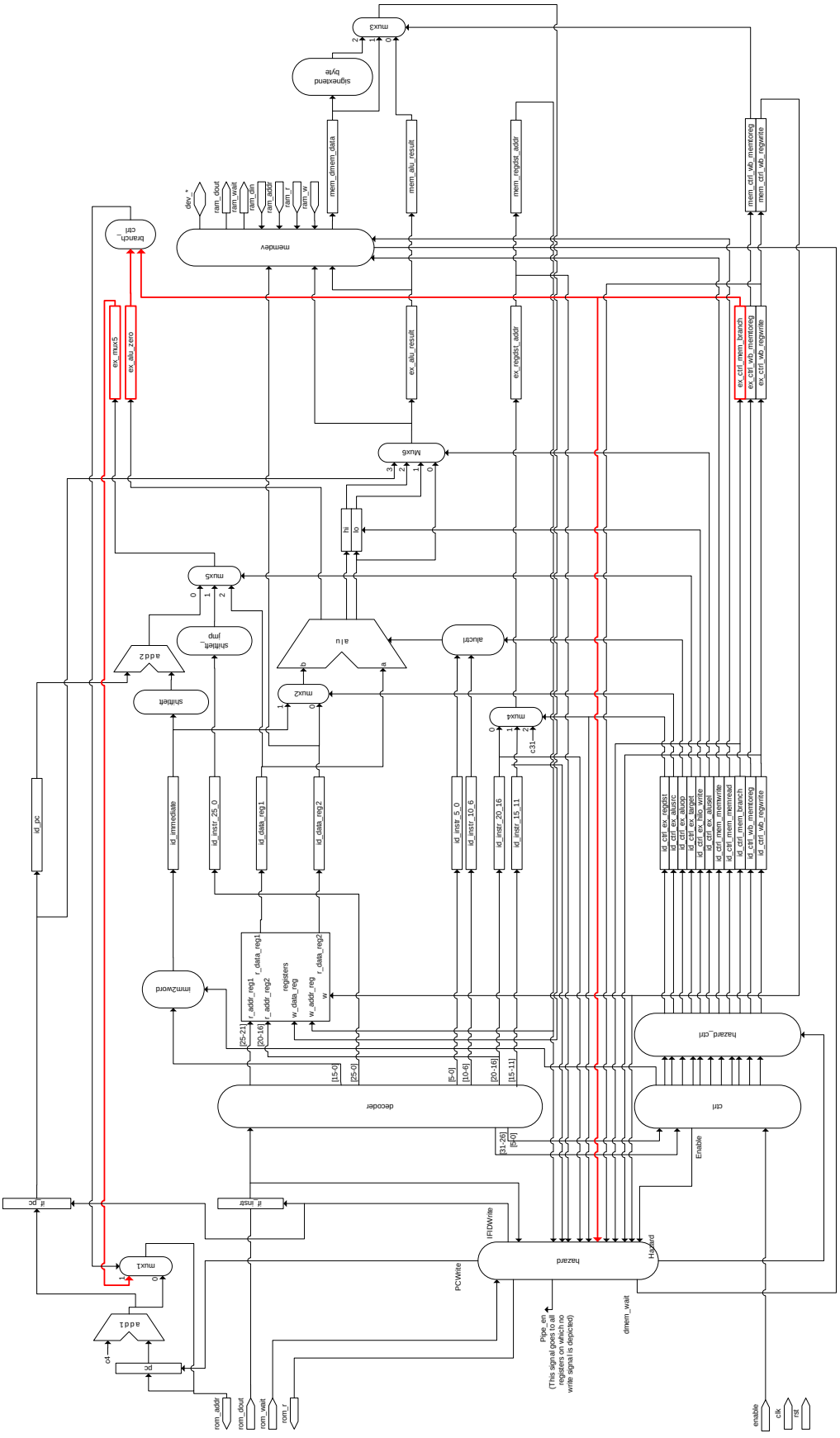

Figure 2: Modules that are to be removed