ASIP Laboratory - Session 5

Paul Georg Wagner Majd Mansour

June 10, 2017

Exercise 1: Creating the applications

The first exercise of this session consisted of migrating the existing bubble sort implementation from the last session to a new framework, allowing it to run on the FPGA board. In order to do so, we copied our not optimized bubble sort implementation without the custom `bgeu` instruction to the two provided framework files *Bubblesort_ LCD_ forSW.s* and *Bubblesort_ LCD_ forHW.s*. The software version of the code is targeted to run with dlxsim and ModelSim, while the hardware version can be compiled for the FPGA board. Because the FPGA board requires four NOPs after each instruction, we changed the bubble sort algorithm for the hardware version slightly, in order to meet this requirement.

For the first exercise we compiled the software version (*Bubblesort_ LCD_ forSW.s*) and executed it using dlxsim and ModelSim. The output of dlxsim is given in listing 1. Unsurprisingly the bubble sort implementation still works and takes 6195 cycles to execute. Afterwards we also simulated the code using ModelSim, which resulted in the same output (see figure 1). However, with ModelSim the number of required cycles is much higher (6943). This discrepancy occurs because of the way ModelSim executes the program. In general, the values that ModelSim outputs are more reliable.

Listing 1: Output of dlxsim

```
Initialise array...
Array:
45 75 342 54 7 86 92 235
4 42 99 78 63 352 21 634
6 77 346 23
BubbleSort: sorting...
Array:
4 6 7 21 23 42 45 54
63 75 77 78 86 92 99 235
342 346 352 634
Array Sorted: YES
Number of cycles needed: 6195
```

Exercise 2: Implementing Project Version1

In order to compile the *Bubblesort_ LCD_ forHW.s* file into a binary that can be uploaded to the FPGA, several steps are necessary. First of all we created a Xilinx ISE project and added the VHDL files of the unmodified CPU to the project. Then we generated a bitstream for the target FPGA. Finally, a call to `make fpga` merged the generated bitstream and the compiled code to a binary that can be uploaded to the FPGA board using `make upload`.

When running the framework directly on the FPGA board, the output of the code can be seen on the UART terminal. The output of the code is displayed in figure 2. Apparently the code also runs correctly on the FPGA board. However, the code takes a total of 16529

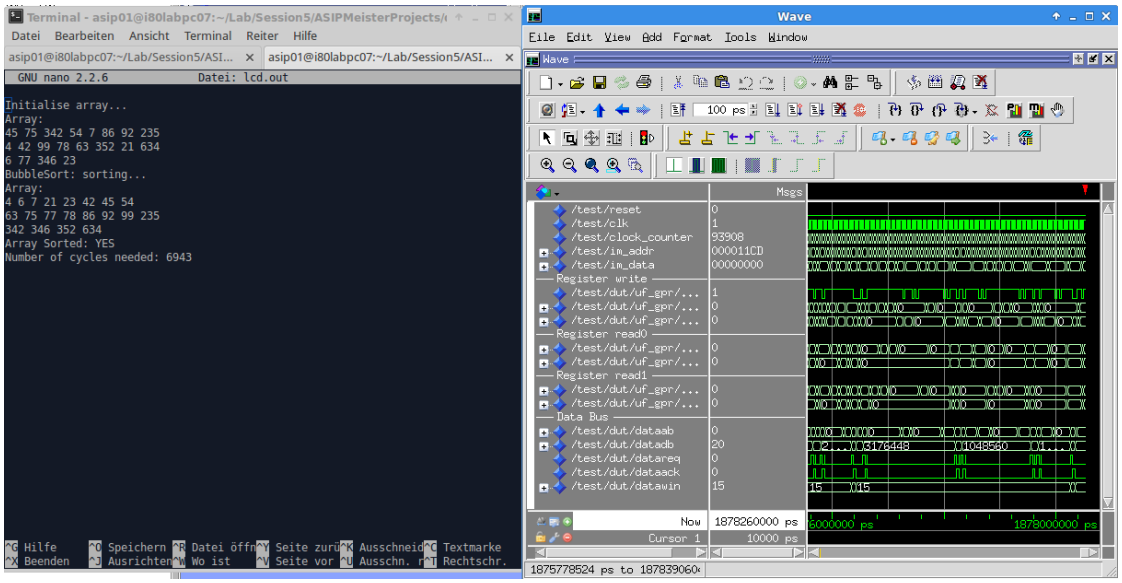

Figure 1: Output of ModelSim.

cycles on the FPGA board, so way more than what either dlxsim or ModelSim required. This is because of the additional NOPs that had to be added to the code in order to meet the requirement of having at least four NOPs after each instruction. We tested the framework with 100 MHz timing as well as 50 MHz timing. In each case, the output was correct and the number of cycles did not change.

a) *If your design works correctly, find out the design statistics (speed and area).*

The Xilinx ISE tool calculates various design statistics regarding achieved speed and required area while compiling the code to a bitstream. However, the framework that is used for the FPGA board contains overhead such as code to communicate with the LCD as well as the UART terminal. Hence when compiling the framework for uploading it to the board, the given results are not particularly reliable. In order to avoid deceptive design statistics, the code can be compiled a second time using only the framework files provided in *ISE_ Benchmark*, which internally maps all the CPU connections in order to decrease the area needed for the framework, thus giving more accurate design statistics.

With the reduced project, Xilinx ISE outputs various statistics regarding the achievable FPGA speed and the required area. For the bubble sort algorithm along with the unmodified CPU, a total of 998 out of 17280 (5%) FPGA slices and 3260 out of 69120 (4%) look-up tables are necessary. Furthermore, the minimal amount of time for a clock cycle that is compatible with the project has been determined as 6.316 nanoseconds. This results in a maximum achievable clock frequency of $\frac{1}{6.316\text{ ns}} = 158.328$ MHz. Up to this processor speed the code should be capable of producing correct results.

b) *Compute the accurate time (in ms) required to sort the 20 numbers.*

The amount of time that is necessary for sorting the numbers arises from the processor speed and the number of required cycles. The highest tested processor speed is 100 MHz, while the number of cycles the bubble sort algorithm requires is 16529 cycles. This results in an execution time of $\frac{16529\text{ cycles}}{100\text{ MHz}} = 1.6529 * 10^{-4}$ s $= 165.29$ ns. If the processor is operated at only 50 MHz, the required execution time doubles.

c) *Analyse the time and find the critical path.*

In order to calculate the minimal amount of time that is required for a single clock cycle, the Xilinx ISE tool analyzes paths through the data flow graph of the input code and

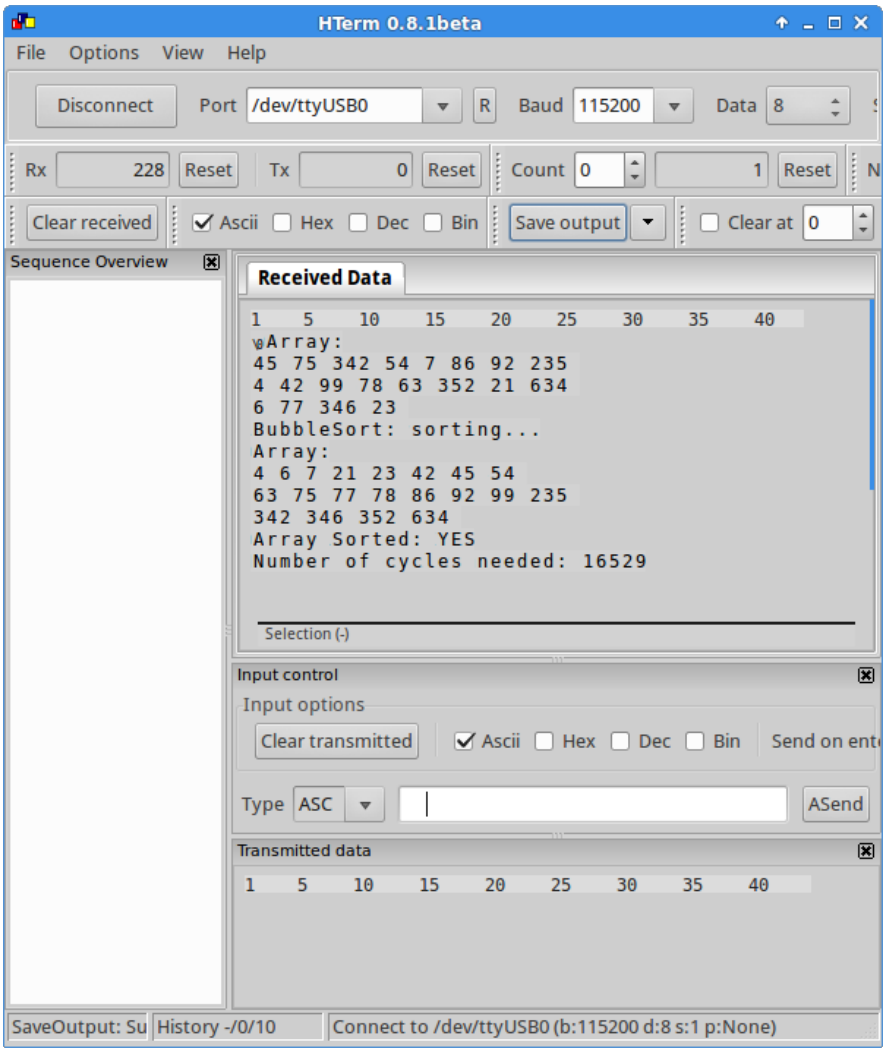

Figure 2: Output of hterm when running the project on the FPGA board.

schedules operations to different cycles. The critical path is the longest path through the data flow graph, which decides how much time has to be reserved for a single clock cycle. As the required time for the critical path increases, the maximum achievable processor frequency decreases. For this project, the Xilinx ISE tool analyzed a total of 234260 different paths and identified the critical path time as 6.255 ns (see listing 2). Along with clock path skew and clock uncertainty we get the already mentioned minimal amount of time for a single clock cycle as $6.255\ \text{ns} + 0.026\ \text{ns} + 0.035\ \text{ns} = 6.316\ \text{ns}$.

Listing 2: Critical path analysis

```
Slack (setup path):     -1.316ns (requirement - (data path - clock path skew + uncertainty))
  Source:               CPU0/UA_CTRL/UA_CW_EXE/DOUT_33 (FF)
  Destination:          CPU0/UA_PREG10/DOUT_2 (FF)
  Requirement:          5.000ns
  Data Path Delay:      6.255ns (Levels of Logic = 7)
  Clock Path Skew:      -0.026ns (0.135 - 0.161)
  Source Clock:         clk_in_BUFGP rising at 0.000ns
  Destination Clock:    clk_in_BUFGP rising at 5.000ns
  Clock Uncertainty:    0.035ns

  Clock Uncertainty:          0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):  0.070ns
    Total Input Jitter (TIJ):   0.000ns
    Discrete Jitter (DJ):       0.000ns
    Phase Error (PE):           0.000ns

    Maximum Data Path: CPU0/UA_CTRL/UA_CW_EXE/DOUT_33 to CPU0/UA_PREG10/DOUT_2
    Location             Delay type         Delay(ns)  Physical Resource
                                                       Logical Resource(s)
    -------------------------------------------------  -------------------
    SLICE_X42Y88.CQ      Tcko                  0.450   CPU0/UA_CTRL/UA_CW_EXE/DOUT<33>
                                                       CPU0/UA_CTRL/UA_CW_EXE/DOUT_33
    SLICE_X46Y87.B1      net (fanout=45)       1.156   CPU0/UA_CTRL/UA_CW_EXE/DOUT<33>
    SLICE_X46Y87.B       Tilo                  0.094   CPU0/UF_ALU0/pre_result<23>189
                                                       CPU0/UF_ALU0/bin<19>1
    SLICE_X45Y91.D6      net (fanout=4)        0.666   CPU0/UF_ALU0/bin<19>
    SLICE_X45Y91.D       Tilo                  0.094   CPU0/UF_ALU0/a0/carry_1_or0000166
                                                       CPU0/UF_ALU0/a0/carry_1_or0000166
    SLICE_X44Y91.B1      net (fanout=9)        0.769   CPU0/UF_ALU0/a0/carry_1_or0000166
    SLICE_X44Y91.B       Tilo                  0.094   CPU0/UA_CTRL/UA_VALID_EXE/DOUT
                                                       CPU0/UF_ALU0/a0/carry_0_or0000233_SW3
    SLICE_X42Y92.A2      net (fanout=1)        0.774   N623
    SLICE_X42Y92.A       Tilo                  0.094   N545
                                                       CPU0/UF_ALU0/a0/u16_1/carry_1_or00001
    SLICE_X46Y93.D6      net (fanout=7)        0.481   CPU0/UF_ALU0/a0/u16_1/carry<1>
    SLICE_X46Y93.D       Tilo                  0.094   CPU0/UA_PREG08/DOUT<24>
                                                       CPU0/UF_ALU0/pre_result<24>171
    SLICE_X45Y94.A1      net (fanout=2)        0.857   CPU0/uf_alu0_result<24>
    SLICE_X45Y94.A       Tilo                  0.094   CPU0/UA_PREG08/DOUT<27>
                                                       CPU0/UF_ALU0/Z_cmp_eq0000216
    SLICE_X43Y93.C4      net (fanout=1)        0.509   CPU0/UF_ALU0/Z_cmp_eq0000216
    SLICE_X43Y93.CLK     Tas                   0.029   CPU0/UA_PREG10/DOUT<2>
                                                       CPU0/UF_ALU0/Z_cmp_eq0000281
                                                       CPU0/UA_PREG10/DOUT_2
    -------------------------------------------------  -------------------
    Total                                      6.255ns (1.043ns logic, 5.212ns route)
                                                       (16.7% logic, 83.3% route)
```

Exercise 3: Implementing Project Version2

After running the first bubble sort version on the FPGA, we also compiled the second version, which uses the `bgeu` instruction of the modified CPU to speed up the calculation. Again, we first simulated the bubble sort algorithm using dlxsim and ModelSim. The output of dlxsim is given in listing 3.

Listing 3: Output of dlxsim for the updated project

```
Initialise array...
Array:
45 75 342 54 7 86 92 235
4 42 99 78 63 352 21 634
6 77 346 23
BubbleSort: sorting...
Array:
4 6 7 21 23 42 45 54
63 75 77 78 86 92 99 235
```

```
342 346 352 634
Array Sorted: YES
Number of cycles needed: 4709
```

Unsurprisingly the use of the new `bgeu` instruction reduced the required number of cycles. The achieved speedup results to $\frac{6195}{4709} \approx 1.32$. Again, the output of ModelSim results in a slightly larger value, for reasons already mentioned.

Afterwards we compiled the hardware project file into a binary that for uploading it to the FPGA and recompiled it using only the *ISE_Benchmark* framework files in order to get reliable design statistics. The output of the FPGA board when running this project version is displayed in figure 3. Apparently the new `bgeu` instruction also runs correctly on the FPGA board. This time the code takes a total of 14015 cycles to complete, which is a speedup of $\frac{16529}{14015} \approx 1.18$ compared to the version without the `bgeu` instruction. We again tested the framework with 100 MHz timing as well as 50 MHz timing. In each case, the output was correct and the number of cycles did not change.

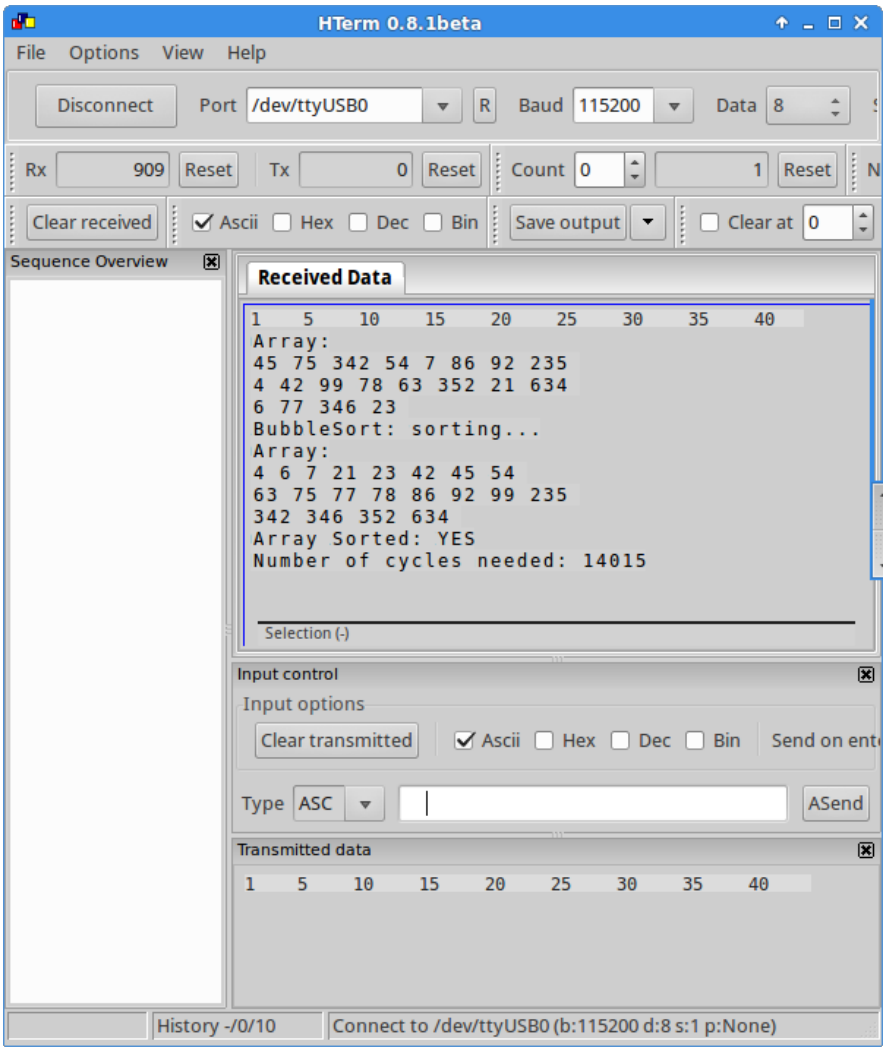

Figure 3: Output of hterm when running the updated project on the FPGA board.

a) *If your design works correctly, find out the design statistics (speed and area).*

When compiling the bubble sort algorithm for the updated CPU, Xilinx ISE determines that a total of 1078 out of 17280 (6%) FPGA slices and 3382 out of 69120 (4%) look-up tables are necessary. This is more than what has been used for the previous CPU, which is not a surprising result given that the updated CPU has more functionality than

before – including an additional adder resource. Furthermore, the minimal amount of time for a clock cycle that is compatible with the updated CPU has been determined as 7.039 ns nanoseconds. This results in a maximum achievable clock frequency of $\frac{1}{7.039 \text{ ns}} = 142.066$ MHz. So the additional functionality decreases the achievable maximum clock frequency by about 16 Mhz. However, since the clock frequency of the available FPGA board can only be set to a maximum of 100 Mhz, this is not a problem.

b) *Compute the accurate time (in ms) required to sort the 20 numbers.*

The highest tested processor speed is 100 MHz, while the number of cycles the updated version of the bubble sort algorithm requires is 14015 cycles. This results in an execution time of $\frac{14015 \text{ cycles}}{100 \text{ MHz}} = 1.4015 * 10^{-4}$ s $= 140.15$ ns. If the processor is operated at only 50 MHz, the required execution time doubles.

c) *Analyse the time and find the critical path.*

For the updated project, the Xilinx ISE tool analyzed a total of 542382 different paths and identified the critical path time as 6.873 ns (see listing 4). Along with clock path skew and clock uncertainty we get the minimal amount of time for a single clock cycle as 6.873 ns + 0.131 ns + 0.035 ns = 7.039 ns. This time is higher than in the previous project, because the additional functionality of the CPU results in a longer critical data flow path.

d) *Compare the design statistics between the two versions.*

The design statistics between the two projects are in general quite similar, but also differ at some crucial points. As could be expected, the updated CPU version uses more resources and hence takes more area on the FPGA board, most notably resulting in an increased amount of used FPGA slices (1078 instead of 998) and look-up tables (3382 instead of 3260). Furthermore, the updated CPU has a longer critical path delay of 7.039 nanoseconds instead of 6.316 nanoseconds. This results in a reduced maximum achievable frequency of 142.066 MHz instead of 158.328 MHz.

e) *How does this affect the execution time (in cycles and ms)?*

The higher area costs and reduced speed of the updated CPU version are countered by the reduced number of CPU instructions that the updated bubble sort program takes. Since this bubble sort algorithm can replace the previous `slte`/`bnez` instruction pair (along with eight intermediate NOPs for the hardware version) by only one `bgeu` instruction (along with four trailing NOPs), the amount of necessary CPU instructions reduces. While the first bubble sort algorithm took 16529 CPU cycles, the new version only takes 14015 CPU cycles. Both CPU versions have a maximum frequency higher than 100 MHz, which is the highest value that can be set for the available FPGA board. Hence both versions can be executed with a clock frequency of 100 MHz, which results in a total execution time of only 140.15 nanoseconds compared to the 165.29 nanoseconds of the previous version.

So all in all the updated CPU has a higher area usage and lower maximum frequency, but manages to reduce the execution time of the sample program by more than 25 nanoseconds.

Listing 4: Critical path analysis of the updated version

```
Slack (setup path):     -2.039ns (requirement - (data path - clock path skew + uncertainty))
  Source:               CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_FFd2 (FF)
  Destination:          CPU0/UA_PREG32/DOUT_13 (FF)
  Requirement:          5.000ns
  Data Path Delay:      6.873ns (Levels of Logic = 8)
  Clock Path Skew:      -0.131ns (1.288 - 1.419)
  Source Clock:         clk_in_BUFGP rising at 0.000ns
  Destination Clock:    clk_in_BUFGP rising at 5.000ns
  Clock Uncertainty:    0.035ns

  Clock Uncertainty:          0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter (TSJ):  0.070ns
    Total Input Jitter (TIJ):   0.000ns
    Discrete Jitter (DJ):       0.000ns
    Phase Error (PE):           0.000ns

  Maximum Data Path: CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_FFd2 to CPU0/UA_PREG32/DOUT_13
    Location              Delay type          Delay(ns)  Physical Resource
                                                         Logical Resource(s)
    -------------------------------------------------  -------------------
    SLICE_X92Y82.BQ       Tcko                  0.471   CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_FFd1_1
                                                        CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_FFd2
    SLICE_X87Y85.B5       net (fanout=892)      0.842   CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_FFd2
    SLICE_X87Y85.B        Tilo                  0.094   CPU0/UF_MUL0/mulu/current_state_FSM_FFd34
                                                        CPU0/UA_CTRL/UA_PROCFSM/cur_state_FSM_Out01
    SLICE_X86Y83.B6       net (fanout=406)      0.376   CPU0/uf_div0_reset
    SLICE_X86Y83.B        Tilo                  0.094   CPU0/UF_DIV0/divu/current_state_FSM_FFd34
                                                        CPU0/UF_DIV0/divu/current_state_FSM_FFd35-In1
    SLICE_X86Y83.A5       net (fanout=4)        0.271   CPU0/UF_DIV0/divu/current_state_FSM_FFd35-In
    SLICE_X86Y83.A        Tilo                  0.094   CPU0/UF_DIV0/divu/current_state_FSM_FFd34
                                                        CPU0/UA_CTRL/do_stall_ID_SW0_SW0
    SLICE_X87Y77.A6       net (fanout=3)        0.492   N527
    SLICE_X87Y77.A        Tilo                  0.094   CPU0/UA_CTRL/advance_stage_IF75
                                                        CPU0/UA_CTRL/do_stall_ID
    SLICE_X70Y76.B6       net (fanout=14)       0.862   CPU0/UA_CTRL/do_stall_ID
    SLICE_X70Y76.B        Tilo                  0.094   CPU0/UF_GPR/Mmux_data_out0_829
                                                        CPU0/UA_MUX03/DOUT<1>1
    SLICE_X69Y79.A5       net (fanout=256)      0.844   CPU0/ua_mux03_dout<1>
    SLICE_X69Y79.A        Tilo                  0.094   CPU0/UA_PREG06/DOUT<13>
                                                        CPU0/UF_GPR/Mmux_data_out0_813
    SLICE_X69Y79.C2       net (fanout=1)        0.749   CPU0/UF_GPR/Mmux_data_out0_813
    SLICE_X69Y79.CMUX     Tilo                  0.392   CPU0/UA_PREG06/DOUT<13>
                                                        CPU0/UF_GPR/Mmux_data_out0_34
                                                        CPU0/UF_GPR/Mmux_data_out0_2_f7_3
    SLICE_X87Y79.C6       net (fanout=1)        0.981   CPU0/uf_gpr_data_out0<13>
    SLICE_X87Y79.CLK      Tas                   0.029   CPU0/UA_PREG32/DOUT<14>
                                                        CPU0/UA_MUX05/DOUT<13>1
                                                        CPU0/UA_PREG32/DOUT_13
    -------------------------------------------------  -------------------
    Total                                       6.873ns (1.456ns logic, 5.417ns route)
                                                        (21.2% logic, 78.8% route)
```