TFE4141 Design of Digital Systems 1

Assignment 5: Datapath design

**Mega adder requirements**

In this assignment we will design a large adder circuit. This adder circuit must satisfy a set of requirements as described in this section.

**Functional requirements**

The design takes two 128-bit numbers a and b, add these together and produces a 128-bit result y.

**Interface requirements**

The design has two interfaces. One interface for sending data in to the design and one interface that is used for sending data out of the circuit.

The input interface is used for sending in the two 128-bit numbers a and b in chunks of 32-bits at a time.

These numbers are added together and produces a 128-bit number y that is sent out of the circuit also in 32-bit chunks.

The 128-bit numbers are chopped up in 32-bit words as illustrated in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **127..96** | **95..64** | **63 ..32** | **31 ..0** |
| a3 | a2 | a1 | a0 |
| b3 | b2 | b1 | b0 |
| y3 | y2 | y1 | y0 |

Table 1. Partitioning the 128-bit operands into four 32-bit words.

Both the input and the output interfaces are valid/ready based handshake interfaces.

* data\_<in/out> must have valid data when data\_<in/out>\_valid is high.
* Data is transferred on the interface when data\_<in/out>\_valid and data\_<in/out>\_ready are high both in the same clock cycle.
* If data\_<in/out>\_valid is high and the corresponding ready signal is low in a particular clock cycle, then data\_<in/out>\_valid must be high also in the following clock cycle and the data\_<in/out> signal must retain its previous value. This situation is illustrated for the data transfer of a3 and y3 in Figure 1 and Figure 2.

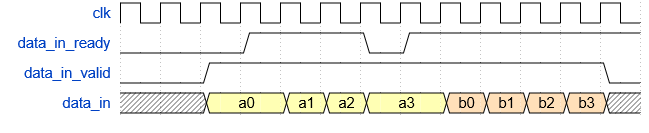


Figure 1. Input interface.

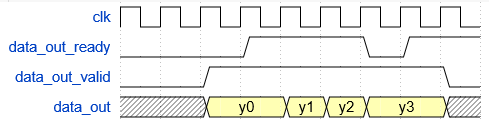


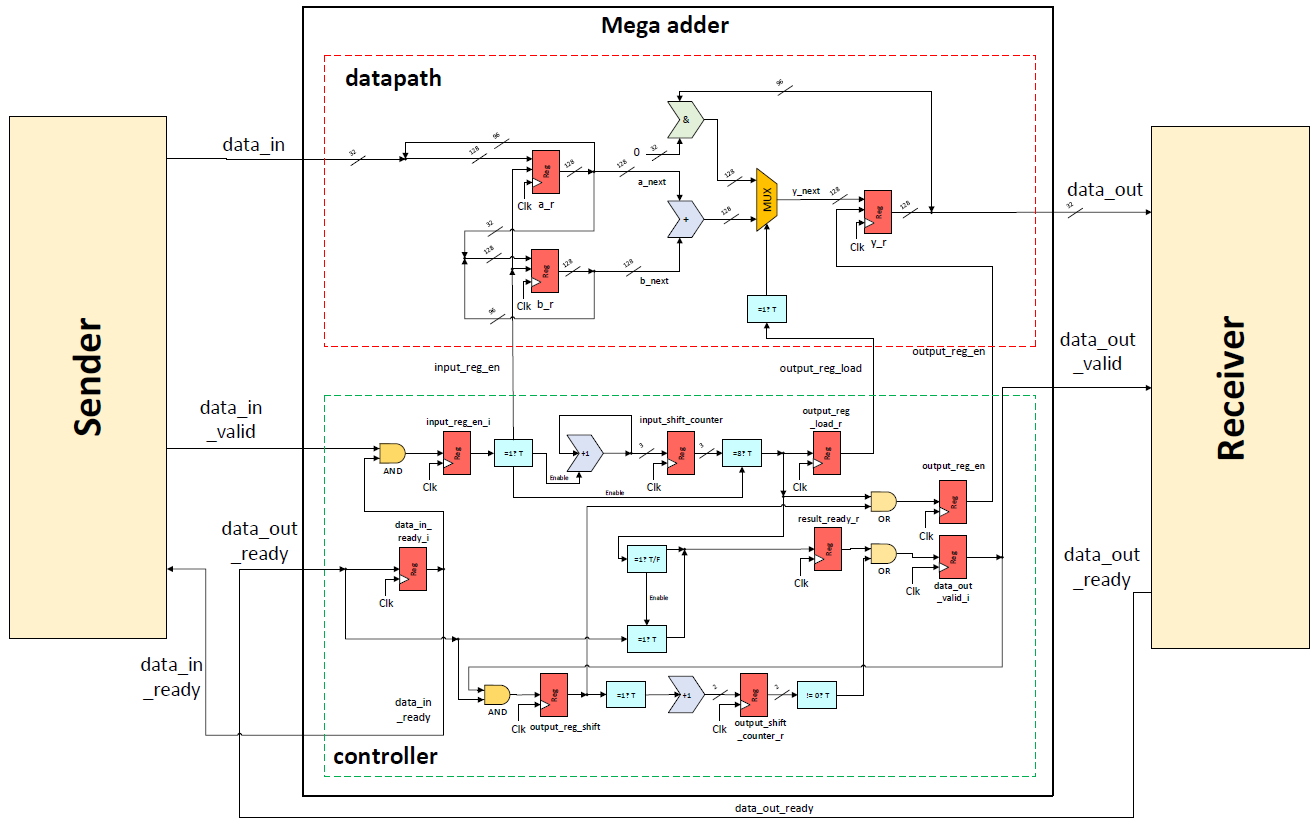
Figure 2. Output interface.

**Performance requirements**

The design must be able to run at minimum 200MHz

**Task 1: Draw a block diagram.**

Examine the code for the mega\_adder. Create a block diagram for the design. Create a block diagram for both the datapath and the control path.



**Note: All registers defined are connected to the reset signal (to restart them when necessary) and most of them to clock signal too (to update them with synchronization). To make the diagram more readable, such connections have been either omitted or shorted.**

**Task 2: Simulate the circuit.**

Run the simple testbench that has been made for the design. Check that the design produces the correct result.

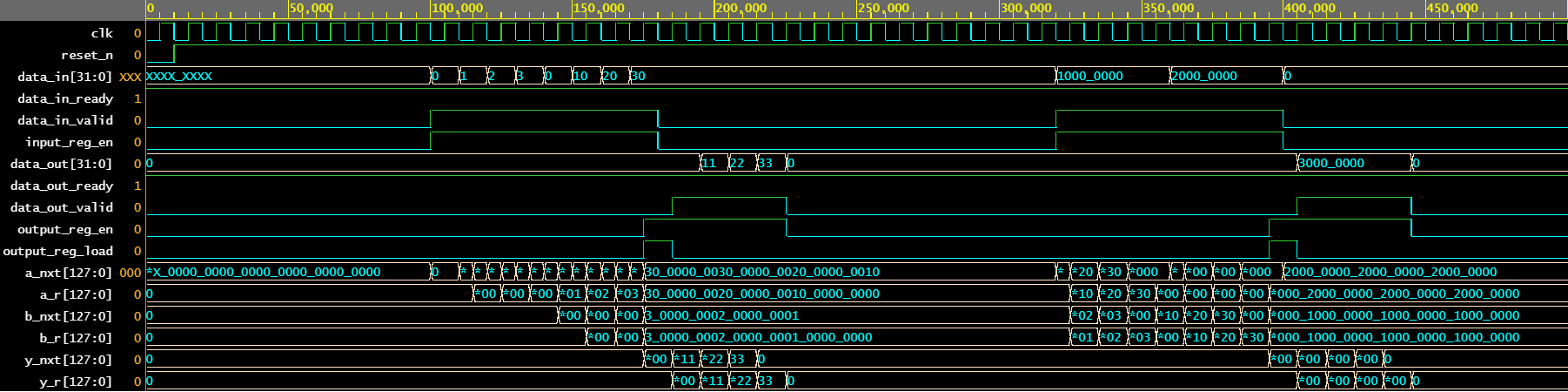
**The test bench works perfectly. A brief overall of what we can see in the graph (shown below):**

**Every time both data\_in\_ready and data\_in\_valid are 1 (input\_reg\_en = 1), whatever there is in data\_in (32 bits) is saved into the most significant part of a\_next (not clocked process, so not a register). At the same time, a\_next is stored in a\_r (clocked process, so a register), shifting the value 32-bits to the right. In the same way, the 32-bits least significant part of a\_r goes into the 32-bits most significant part of b\_next, set later in b\_r in the same way as shown with a\_r.**

**When output\_reg\_load = 1, the addition of a\_next and b\_next is stored in y\_next (not clocked process, so not a register), clearing the 32-bits least significant part (just outputted, so useless to keep) once such a condition goes back to 0.**

**Every time both data\_out\_ready and data\_out\_valid are 1 (output\_reg\_en = 1), y\_next is stored in y\_r (clocked process, so a register), which is continuously being sent to data\_out.**

**Inputs are filled in 8 chunks of 32-bits (4 for each input [a0, a1, a2, a3] & [b0, b1, b2, b3]) and output is sent out in 4 chunks [y0, y1, y2, y3].**



**As one can see above, this test bench performs two independent tests:**

* **A = 0..3 0..2 0..1 0..0 & B = 0..30 0..20 0..10 0..0  so Y = 0..33 0..22 0..11 0..0**
* **A = 2..0 2..0 2..0 2..0 & B = 1..0 1..0 1..0 1..0  so Y = 3..0 3..0 3..0 3..0**

**Task 3: Count the number of flip flops that should be inferred during synthesis.**

Count the number of flip-flops that will be inferred during synthesis. Do this by counting the number of flip-flops that will be inferred by each process in the design and add up the numbers.

Run synthesis and check that the number of flip flops you counted matches that inferred during synthesis.

**For the datapath, the code shows there are 6 signals of 128 bits each (a\_next, a\_r, b\_next, b\_r, y\_next, y\_r). Only a\_r, b\_r, y\_r are used in clocked processes, which means these 3 signals are (3x128) 384 registers (so a\_next, b\_next, y\_next are just groups of wires). With the same approach for the controller, this one has 7 registers, 2 1-bit inputs and 5 1-bit outputs (2 internal outputs to communicate with datapath and 2 external outputs). That makes 391 registers (391 1-bit flip flops) in total.**

**Now using Vivado, with synthesized design done, it shows the same information: the controller uses 7 registers and the datapath 384 registers, so 391 registers in total. Moreover, the implementation design shows again the same numbers, which means the design cannot be simplified anymore.**

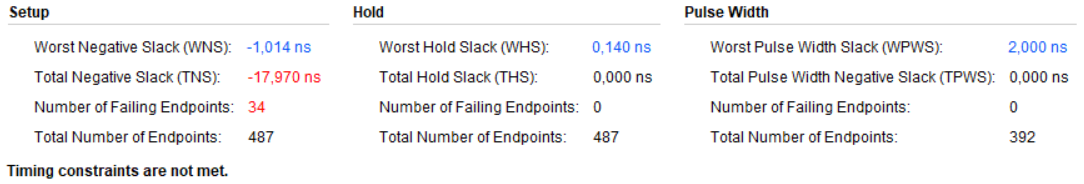
**Task 4: Synthesize the design and report the max clock frequency.**

Synthesize the design by pressing “Run Synthesis”. Constraints have been set up so that the target frequency is 200MHz. The synthesis tool will not be able to meet those constraints.

**The project has been requested to have a period of 5ns, which means a frequency of 1/5ns = 200MHz, so this value has been added in the design as a constraint.**

**Unfortunately, the synthetized design report informs that this period will not be met because it is not time enough to perform all requested calculations. A negative slack time of 1ns approx. is shown. That is to say that the frequency will not be 200MHz, but 1/(5ns + 1ns) = 166MHz.**

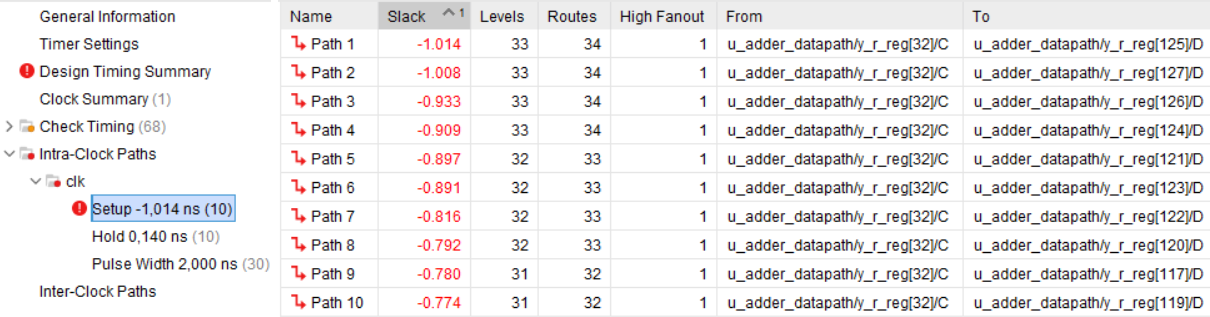
**Once the correct period is met, then it should be used in the testbench for testing purposes.**



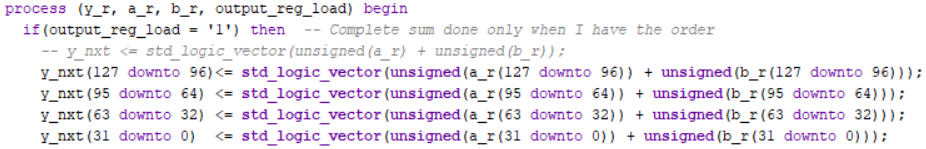
**Task 5: Find the critical path in the design and improve the timing.**

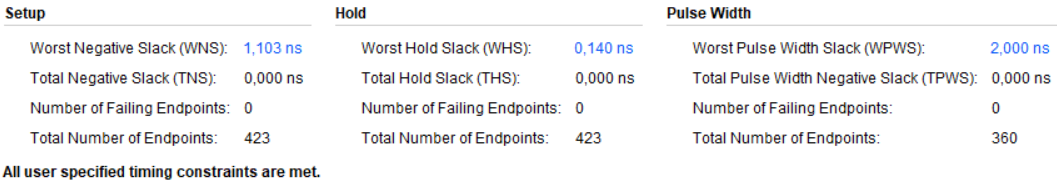
Locate the critical path in the design and improve the timing. This will most likely require a change in the microarchitecture. Draw a block diagram for the design that will have a shorter critical path.

**With the negative slack time of 1ns, going to the option “Timing summary🡪Intra-Clock Paths 🡪clk🡪Setup”, it shows that the delay problem is experienced in the most significant part of y\_r (so during the 128 bits addition). This would be the critical path.**



**As one possible solution to solve this problem, if the signal y\_next is updated through 4 different additions of 32 bits each (so a0+b0, a1+b1, a2+b2, a3+b3) instead of just one addition of 128 bits, the design fits inside the timing constraint of 200MHz without problems (with a positive slack time of 1,1ns).**



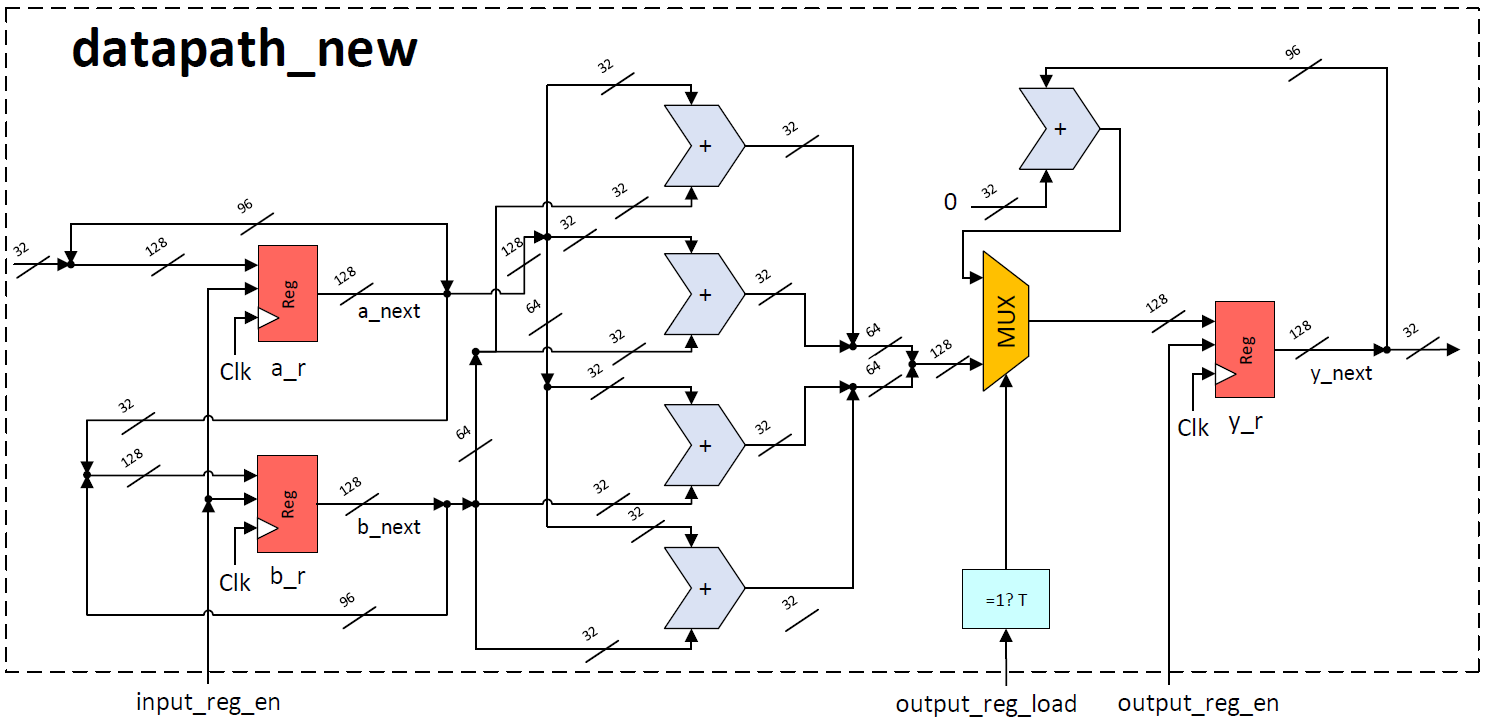


**Task 6: Write RTL code, test and synthesize the new design.**

Implement the design you suggested in Task 5. Check that it is functions correctly and that the timing constrains are met.

**The new design has a positive (not negative!) slack time of 1,1ns, so there is no problem to meet the requested timing constraint.**

**The new microarchitecture in the datapath would be:**



**Appendix A: Reporting timing and editing constraints.**

The screenshot below shows that the clock period is set up to be 5ns. This corresponds to a clock frequency of 200MHz. When you press the “Report Timing Summary” to the left, the “Design Timing Summary” window will appear.   
  
The “Design Timing Summary” window below shows that we have a negative slack of 1ns. This means that the max frequency is around 166MHz instead of the required 200MHz.

If you want to change the constrains, you can just press “Edit Timing Constraints” and change the period of the clock from 5ns to e.g. 10ns. Hopefully the design will be able to meet those relaxed constraints.

