IN3160V22 — Oblig 9
Pipelining and bits in arithmetic calculations

Martin Mihle Nygaard <martimn@ifi.uio.no>

1. We have two numbers a and b which is 16 bits each. How many bits is the result of the sum of these two numbers (i.e. $a + b$)?

Let both a and b be 2^{16}, the highest value a 16-bit number can hold.* Then we have:

$$\log_2(a+b) = \log_2\left(2 \times 2^{16}\right) = \log_2\left(2^{17}\right) = 17.$$

*Or possibly $2^{16} - 1$ if you exclude 0, but it doesn't really matter for the final results, just makes the calculations messier.

2. How many bits is the result of mulitiplying these two 16 bits numbers (i.e. $a \cdot b$)?

With a and b as before, we have

$$\log_2(a \cdot b) = \log_2\left(2^{16} \times 2^{16}\right) = \log_2\left(2^{32}\right) = 32.$$

3. We now have four numbers: a, b, c, and d who are 16 bits each. How many bits is the result of adding all these numbers $a + b + c + d$?

Same logic:

$$\log_2(a+b+c+d) = \log_2\left(4 \times 2^{16}\right) = \log_2\left(2^{18}\right) = 18.$$

4. We now have a additional number e which is 16 bits. How many bits is the result of $(a + b + c + d) \times e$?

And finally, using the previous result:

$$\log_2\left((a+b+c+d) \times e\right) = 18 + \log_2\left(2^{16}\right) = 18 + 16 = 34.$$

5. Draw datapath diagrams for `compute` and `compute_pipelined`.

(a) See Figure 1.

(b) See Figure 2.

6. Implement the new module `compute_pipelined` in synthesizable VHDL based on the given compute RTL architecture. The given testbench `tb_compute_pipelined` shall be used to ensure that the `compute_pipelined` architecture works as required.

The code for the module is attached as a separate file, but is also quoted in Listing 1. In my tests, it completed the attached `tb_compute_pipelined.vhd` test bench without issue.

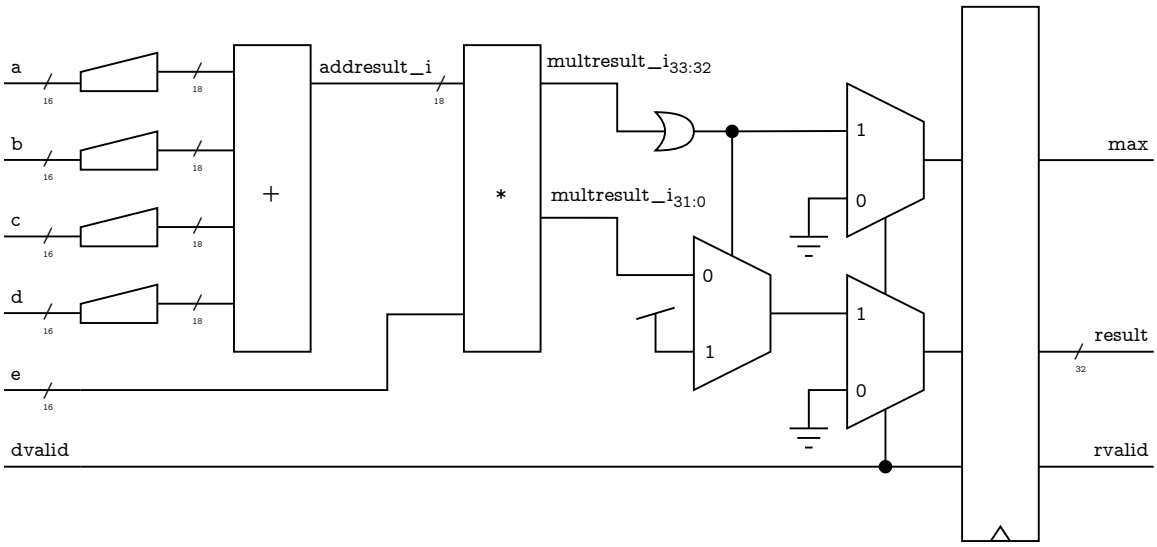

Figure 1: Datapath diagram for the `compute` module. Reset functionality not included. I guess the add operation, +, should technically have been a tree of three two-input blocks instead, but I have abstracted this away for simplicity.

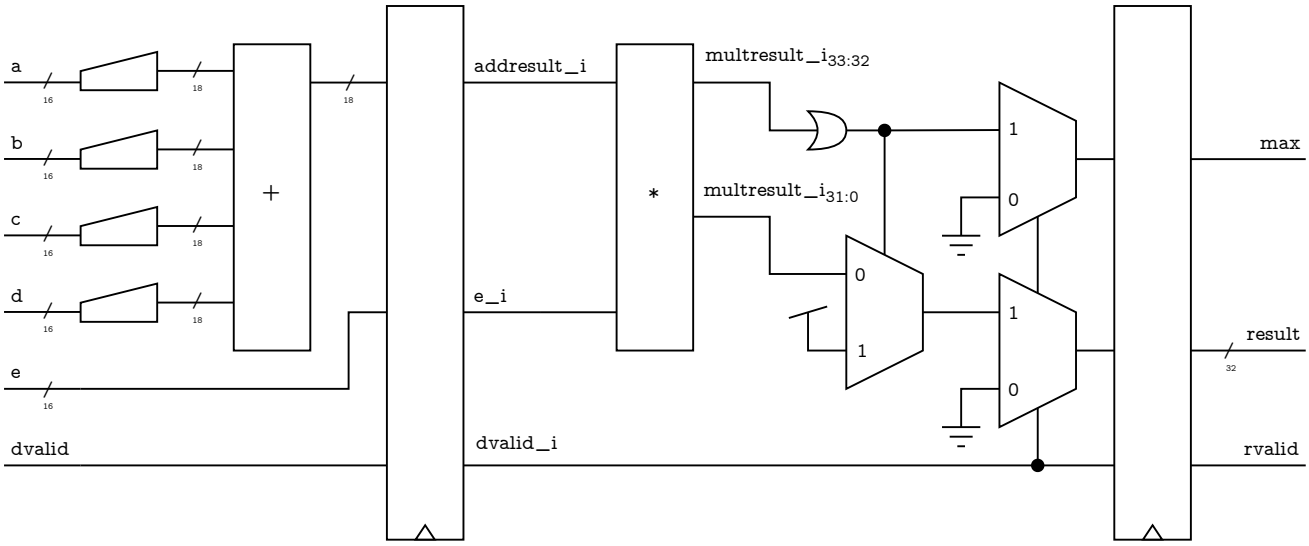

Figure 2: Datapath diagram for the `compute_pipelined` module. Reset functionality not included.

Listing 1: Source file `compute_pipelined_rtl.vhd`. Modified from the included `compute_rtl.vhd`. I've tried to put a comment on all my changes, but see Figure 2 for a more intuitive understanding of the new signals I've introduced.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

architecture rtl of compute_pipelined is
  -- Initialize intermediary signals. I.e first block of registers.
  signal addresult_i  : unsigned(17 downto 0);
  signal e_i          : unsigned(15 downto 0);
  signal dvalid_i     : std_logic;
begin

  process (rst, clk) is
    -- `multresult_i` is still a variable, since it's not stored in a
    -- register, just immediately passed along next logic block. But
    -- `addresult_i` is now a signal.
    variable multresult_i : unsigned(33 downto 0);
  begin
    if rst = '1' then
      result <= (others => '0');
      max    <= '0';
      rvalid <= '0';
      addresult_i  <= (others => '0'); -- New reset targets here.
      e_i          <= (others => '0'); -- XXX
      dvalid_i     <= '0';             -- XXX
    elsif rising_edge(clk) then
      if (dvalid = '1') then -- Check `dvalid` for addr. logic
        addresult_i  <= (unsigned("00" & a) + unsigned("00" & b)) +
                        (unsigned("00" & c) + unsigned("00" & d));
        e_i      <= unsigned(e); -- Propagate next `e`
      end if;
      if (dvalid_i = '1') then -- Check `dvalid_i` for mult. logic
        multresult_i := addresult_i * e_i; -- Use propagated `e`
        if (multresult_i(33 downto 32) = "00") then
          result <= std_logic_vector(multresult_i(31 downto 0));
          max    <= '0';
        else
          result <= (others => '1');
          max    <= '1';
        end if;
      else
        result <= (others => '0');
        max    <= '0';
      end if;
      dvalid_i <= dvalid;   -- Propagate `dvalid`
      rvalid   <= dvalid_i; -- Propagate `dvalid_i`
    end if;
  end process;

end architecture rtl;
```

7. How many registers/flip-flops are used in the module `compute`?

We need to store the `max`, `result` and `rvalid` signals between clock cycles. These are the output signals. This is $1 + 32 + 1 = 34$ bits (I think).

TODO: Check with Vivado's synthesis tool.

8. How many registers/flip-flops are used in the module `compute_pipeline`?

We need to also store `dvalid_i`, `e_i` and `addresult_i` in addition the same signals as in `compute`. This is $1 + 16 + 18 + [34] = 69$ bits.

TODO: Check with Vivado's synthesis tool.