jan.kleinsorge [☺] tu-dortmund.de

Exercises to
Cyber-physical System Fundamentals
Summer term 2014

Assignment 3

(10 Points)

To be solved in the weeks starting on Monday, June 2, 2014

This assignment is to be solved in sessions on June, 4th and 11th.

Hints:

- For this session we use GHDL and GtkWave under Cygwin. To prepare your compilation/simulation environment, launch the cygwin terminal from the desktop and type “startxwin”. Within the new terminal window, navigate (UNIX paths! “c:” corresponds to “/cygdrive/c/”) to the source code folder (for example ‘cd /cygdrive/c/Users/-cilab13XX/Desktop”). Type “make” to compile your source files, type “make view” to run the waveform viewer. Use the source code template “a1_tpl” for an initial test. It is readily compilable.
- Lecture slides 2.06/40ff provide additional material in VHDL.
- For internet access, specifiy the http proxy “proxy.cs.tu-dortmund.de”, port 3128 in your browser.

3.1 Semantics of VHDL-simulations (3 Points)

VHDL features two different timing models called *inertial* (default) and *transport*. The former model supports the simulation of inherent lag in circuitries. Consider the following source code. Understand its semantics by simulation. What happens when lines 21 and 22 are changed from “. . . => . . . ” to “. . . => transport . . . ”? What is the meaning of the two timing models in this example and how does it manifest itself?

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.all;
4
5  entity e_inertial is
6  end e_inertial;
7
8  architecture rtl of e_inertial is
9    signal u, v, w : std_logic;
10 begin
11
12   p1: process
13   begin
14     u <= '0', '1' after 10 ns, '0' after 15 ns, '1' after 20 ns,
15          '0' after 25 ns, '1' after 30 ns;
16     wait;
17   end process p1;
18
19   p2: process (u)
20   begin
21     v <= u after 4 ns;
22     w <= u after 6 ns;
23   end process p2;
24
25 end rtl;
```

3.2 VHDL Syntax (7 Points)

In the lecture, a full-adder written in VHDL has been sketched . Extend this initial example to form an N-bit ripple carry adder (RCA). Such an adder passes the carry-bit bit from one “stage” to the next, hence the term *ripple carry*.

Implement a 4-bit RCA like shown, based on the source code template “a2_tpl”. The template already provides basic structural elements, a driver component for the simulation, some example code for reference and comments (see also the slides for reference). Compile and test your solution.

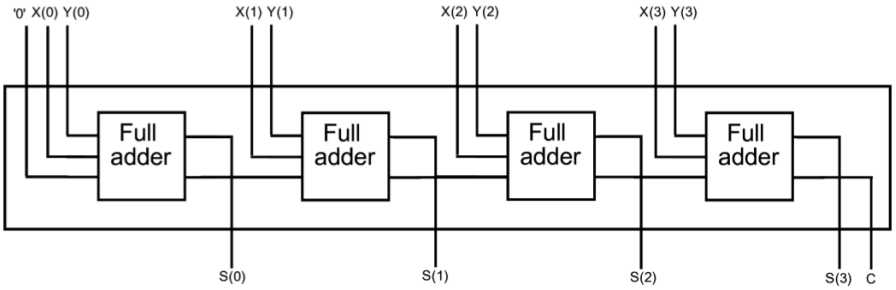

Figure 1: Ripple Carry Adder

General notes:

Dates and additional information can be found on the lecture website (can also be found via EWS). The assignments will be typcically be published **Mondays** on a weekly basis and have to be solved in the lab session of the same week. To pass the labs, a minimum of 50% of the total points must be achieved in the first half and the second half, respectively.