Prof. Bernd Finkbeiner, Ph.D.
Dipl.-Inf. Rüdiger Ehlers
Dipl.-Inf. Hans-Jörg Peter
Michael Gerke, B.Sc.

Reactive Systems Group
Saarland University
Due: Thursday, 4th December 2008
before the lecture, i.e., 2:10pm

Embedded Systems 08/09 – Problem Set 6

Problem 1 (VHDL: 4-Bit Register) (40 pts.)

In hardware design, one usually uses a global clock signal to establish a discrete model of time. Figure 1 shows such a clock signal where each period between two rising edges represents a certain (discrete) moment of time. Using this abstraction, it is possible to describe the behaviour of circuits in terms of clock cycle indexes. Here, SIG^i denotes the stabilized value of signal SIG during the *first* half ($\texttt{clk} = 1$) of cycle i.

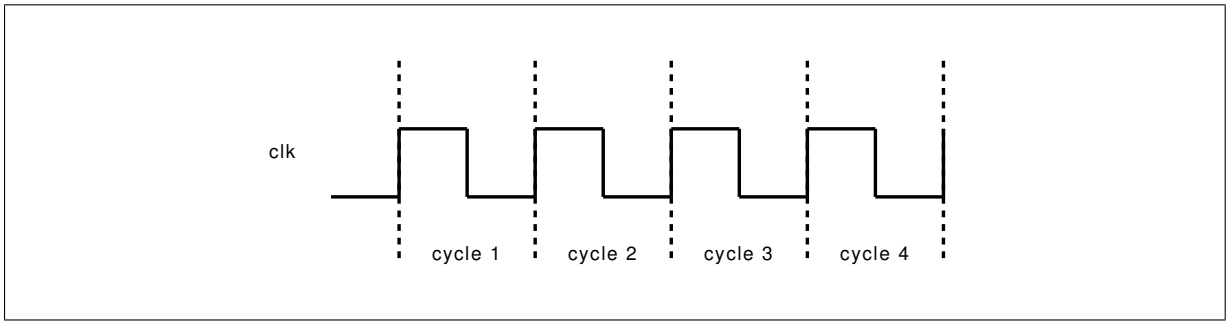

Figure 1: Timing diagram of a typical clock signal

An n-bit register is a standard circuit in hardware design that stores n-bit values. A register has an n-bit input IN, an 1-bit input EN, and an n-bit output OUT. It works as follows: whenever $EN = 1$ the register stores IN, OUT always gives the last value that has been stored in a previous cycle, even if $EN = 1$ in the current cycle. More formally, let IN^i, OUT^i, and EN^i be the input, output, and enabled signals of the i^{th} cycle, respectively, then the behaviour is defined as follows:

$$OUT^{i+1} = \begin{cases} IN^i, & \text{if } EN^i = 1 \\ OUT^i, & \text{otherwise} \end{cases}$$

The input and output interface of a 4-bit register is described by the following VHDL declaration:

```
entity register is
   port(
     in3, in2, in1, in0, en, clk: in bit;
     out3, out2, out1, out0: out bit);
end register;
```

Start with this declaration to develop a 4-bit register in VHDL. [1] When writing the structural architectures, you can use flip-flops and basic gates (NOT, AND, OR, NAND, XOR, NOR) as your starting subentities.

1. Write a VHDL *behavioral* architecture implementing a 4-bit register. (15 pts.)
2. Write a VHDL *structural* architecture implementing a 4-bit register. (25 pts.)

Problem 2 (VHDL: 4-Bit Multiplier) (60 pts.)

The input and output interface of a 4-bit multiplier is described by the following VHDL declaration:

```
entity multiplier is
   port(
     a3, a2, a1, a0, b3, b2, b1, b0: in bit;
     c3, c2, c1, c0, d: out bit);
end multiplier;
```

Start with this declaration to develop a saturating 4-bit multiplier in VHDL. Let

$$a = 8a_3 + 4a_2 + 2a_1 + a_0\,,$$
$$b = 8b_3 + 4b_2 + 2b_1 + b_0\,,$$
$$c = 8c_3 + 4c_2 + 2c_1 + c_0\,.$$

Multiplication modulo 16 is denoted by $*_{16}$ and saturating multiplication modulo 16 is denoted by $*^s_{16}$, i.e.,

$$a *^s_{16} b = \begin{cases} ab\,, & \text{if } ab \leq 15 \\ 15\,, & \text{otherwise}\,. \end{cases}$$

You must ensure that:

- $c = a *_{16} b$, for the 4-bits multiplier;
- $c = a *^s_{16} b$, for the saturating 4-bits multiplier;
- in all cases: $d = 1$ iff there was an overflow, i.e., $a + b > 15$.

When writing the structural architectures, you can use full adders, half adders, and basic gates (NOT, AND, OR, NAND, XOR, NOR) as your starting subentities.

1. Write a VHDL *behavioral* architecture implementing a *saturating* 4-bit multiplier. (20 pts.)
2. Write a VHDL *structural* architecture implementing a *saturating* 4-bit multiplier. (40 pts.)

Hint: Consider the school method for multiplying two binary numbers:

$$\begin{array}{lcccccccc}
 & (& 0 & 0 & 0 & a_3 & a_2 & a_1 & a_0 &) \wedge b_0 \\
+ & (& 0 & 0 & a_3 & a_2 & a_1 & a_0 & 0 &) \wedge b_1 \\
+ & (& 0 & a_3 & a_2 & a_1 & a_0 & 0 & 0 &) \wedge b_2 \\
+ & (& a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 &) \wedge b_3 \\
+ & \text{the carries} & & & & & & & & \\
\hline
\text{Sum} = & c_6 c_5 c_4 c_3 c_2 c_1 c_0 & & & & & & & &
\end{array}$$

[1] Note that `clk` is the clock signal whose rising edges define the cycles.