Adder

Task:

Implement a 2-bit adder using 1-bit full adder and 1-bit half adder as components (Figure 1) that are connected together in a top-level module. Describe both components in VHDL. Prepare two implementations where VHDL components are instantiated in:

- VHDL top-level file
- Block Design top-level file

Simulate and implement each project on FPGA development board.

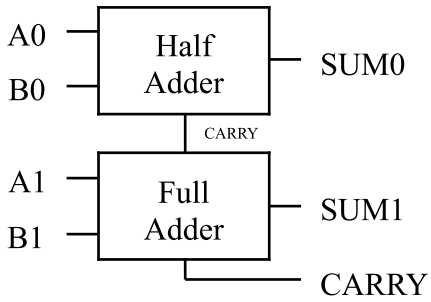

Figure 1: 2-bit Adder

Full Adder Example

The system can be represented as a collection of components and their interconnections using structural design approach. In this way the hardware is described as schematic or netlist, where internal structure of the interconnected components is hidden. The components can be either regular structures (like logic gates, adders, comparators, registers, etc.) or previously defined sources, which are used within another higher level design to create a hierarchy.

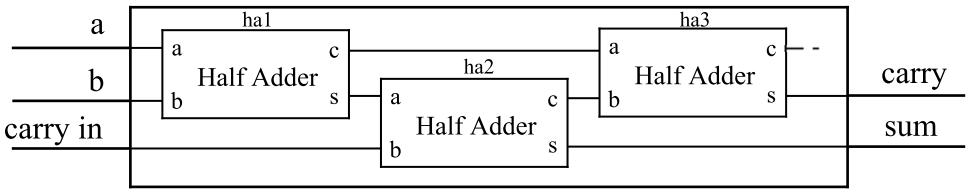

Figure 2: Full Adder Block Diagram

A simple 1-bit full adder circuit has three inputs and two outputs. It can be built from three

half adders as shown in Figure 2. Note that the carry output of the *ha3* half adder is left floating, as it would always equal to '0' (other half adders cannot have their carry outputs equal to '1' both at the same time).

In order to construct the full adder in Figure 2, the half adder should be designed first (refer to the example in the previous Comparator lab). When it is ready, a top-level source file is needed to instantiate and to connect half adders.

One option is to create a block design as a top-level source file. Then half adder design can be packaged into IP and then instantiated from the list of IPs. However, for such trivial design IP generation may not be needed. As an alternative, VHDL source file can be added to the block design as an RTL module. Right-click on an empty space in the block design and select *Add Module* option. Highlight half adder design and click *OK* to place it in the block design. Add two more half adders and connect them. The end result should look similar to the one shown in Figure 3. Note that carry output of the half adder "HA_0" is left floating. The logic associated with that output (AND gate) will be automatically removed during implementation phase.

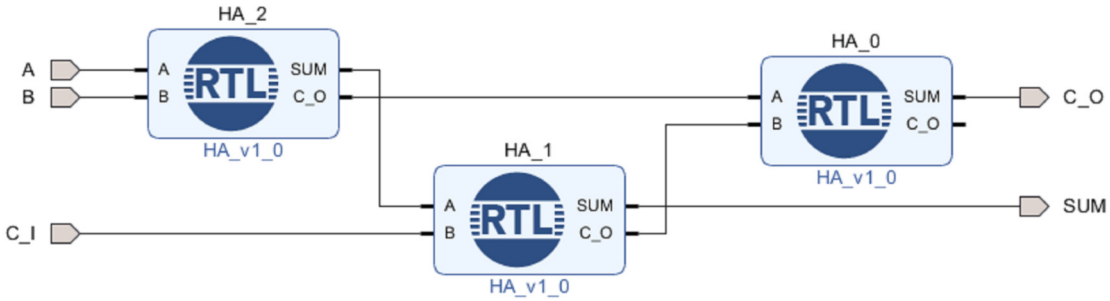

Figure 3: Full Adder Block Design

A similar approach is possible for the VHDL top-level source file using structural description style. The half adder design can be used as a component, thus effectively creating a textual description of a schematic in Figure 2. Component is a piece of conventional VHDL code, which can be used within another description. This allows to partition, share and reuse VHDL code.

Once the half adder has been described, it can be used to form full adder design by declaring it as a component in the declarative part of the full adder architecture (before **begin** keyword). The component declaration is very similar to entity declaration with keywords **entity** substituted for keyword **component** (Listing 1).

Listing 1: VHDL Declaration of Component Half Adder

```
component half_adder is
        port (a, b: in std_logic;
              sum, carry: out std_logic);
end component;
```

The next step is to instantiate a component in the architecture body. Component instantiation starts with a label, followed by component's name. The label is required to differentiate between multiple instances of the same component. The final component instantiation part is mapping of component's inputs/outputs to the ports and signals of the higher level description. The mapping can be done in two ways as shown in Listing 2:

Listing 2: a) Explicit and b) Implicit Instantiation of Component Half Adder

```
ha1: half_adder
port map (a => ha1_a,
          sum => ha1_sum,
          b => ha1_b,
          carry => ha1_carry);
```

a)

```
ha1: half_adder
port map (ha1_a, ha1_b, ha1_sum, ha1_carry);
```

b)

The explicit instantiation associates each input/output of the component (to the left of "=>" operator) with a corresponding signal (to the right of "=>" operator) of the higher level description. Note, that the order, in which the ports are being mapped, does not correspond to the order in component declaration. This is due to the fact that association is being stated explicitly. On the other hand, in the implicit instantiation the signals are ordered exactly as in component declaration. This in turn allows to omit the "=>" operator.

Listing 3 shows three implicitly instantiated *half_adder* components, which form the full adder as shown in Figure 2. Note, that the order of components is not important, since they

are working concurrently (in parallel). The schematic, which can be reconstructed from this structural VHDL description, should be identical to the one in Figure 2.

Listing 3: Structural Description of Full Adder

```
ha3: half_adder port map (c_a, c_b, carry, open);
ha1: half_adder port map (a, b, s_a, c_a);
ha2: half_adder port map (s_a, carry_in, sum, c_b);
```

Note, that in order to form internal connection between components, three additional signals must be declared: *c_a*, *s_a*, *c_b*. Note, that the same signal (e.g., *c_a*) is used as output for one component (*ha1*) and as input for the other component (*ha3*). This signal can be viewed as corresponding wire from full adder schematic in Figure 2. The floating *carry* output of the *ha3* half adder is specified with the keyword **open**.