Catapult C Lab 2

Understanding Interface Synthesis

Introduction

This lab covers some of the basic concepts of Catapult C interface synthesis technology. Catapult C Synthesis employs "Interface Synthesis Technology" to allow variables on an un-timed C function interface to be bound to components that are described in RTL and have implicit timing on the ports. This association of interface variables with interface components is one of the mechanisms that Catapult C uses to embed timing in a design. This lab will illustrate the behavior of the basic "wire" interface as well as interfaces with a "data valid" output.

Using "wire" Interfaces

The Catapult C synthesis library provides some basic components, which mainly pass the process level handshake to the outside world. These components can be selected as 'Architectural Constraints' in the GUI or as commands. The "wire" interface components are the default component. The first section of this lab covers the behavior of the input and output "wire" interface components. These components are define as:

'mgc_in_wire' (default for inputs)

results in a single input signal '`z`' without control signal. The input is assumed to be registered outside of the design and ready whenever the process reads it.

'mgc_out_stdreg' (default for outputs)

results in an output signal '`z`' without any control signals. The output is registered in the data path. The register value is stable until the next write operation occurs.

Set the working directory

Launch Catapult C and set the working directory to the lab2 directory

Add the input file

- ✓ Add the lab2.cpp file to the Catapult project. This design adds two arrays of 16-bit signed integers element by element. It uses the Catapult bit-accurate types to set the bit widths of the port variables.

```
#define nums 2
//Include Catapult bit accurate types
#include <ac_int.h>

#pragma design top
void lab2(int16 a[nums], int16 b[nums], int17 c[nums]){

    c[0] = a[0] + b[0] + b[1];
    c[1] = a[1] - b[0] - b[1];
}
```

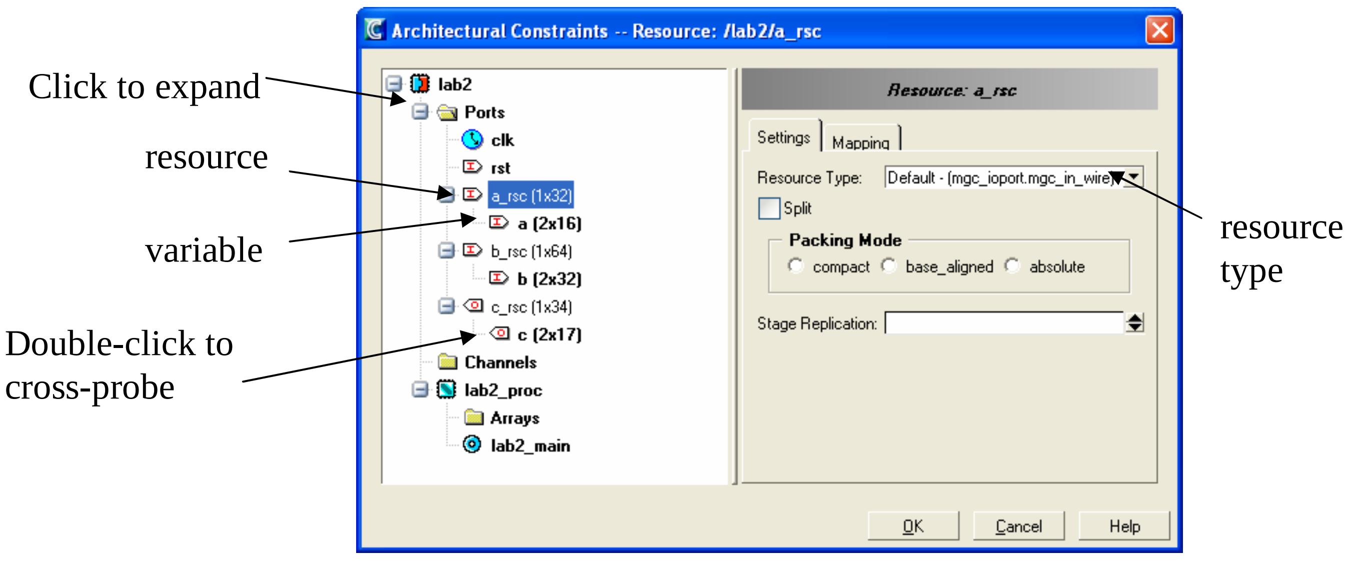
Setup Design

- ✓ Click on “Setup Design” in the design bar and set the technology to Altera Stratix II (Default part and speed grade). Set the clock frequency to **400MHz**

Architectural Constraints

The architectural constraints window provides a view of the ports, arrays and loops in the design, which are all things that can be constrained or optimized. The top section is a list of the ports that will be put on the hardware design. There are several ports that can’t be represented by C++ code. These are automatically added to the design: clock, reset, and an optional enable as well as start and done handshaking. The data in the source code is mapped to inputs, outputs or inout resources. Arrays on the interface have a different Icon, but are still of type input, output, or inout.

- ✓ Click on the “Architectural Constraints” icon in the design bar. This will bring up a dialog, which will show you the ports and loops in your design. Expand the ports folder by clicking on the plus sign in the left pane of the dialog box (Figure 1)



Resource: a_rsc

Resource Type: Default - (mgc_ioport.mgc_in_wire)

Split

Packing Mode: compact / base_aligned / absolute

Stage Replication:

Figure 1 Architectural Constraints

Checking resource mapping

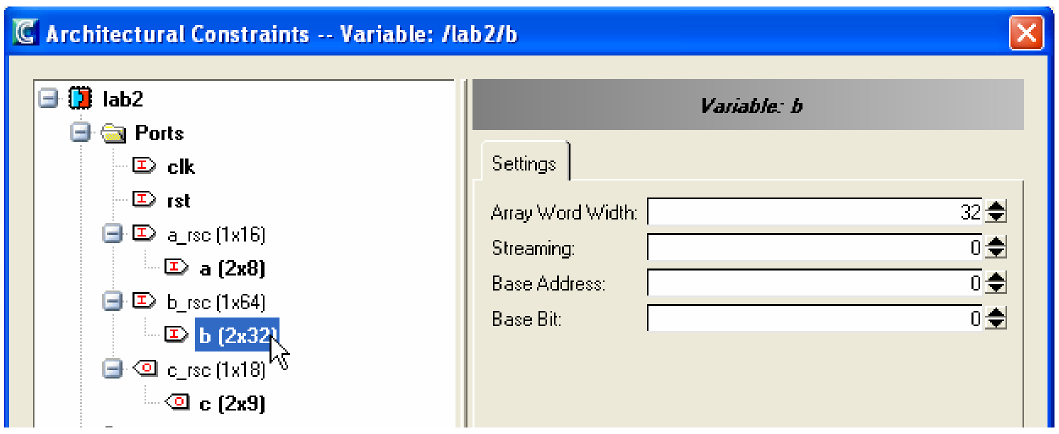
- ✓ Click on each resource to verify that the resource type is set to “mgc_in_wire” for inputs and “mgc_out_stdreg” for outputs.

Checking bit widths

- ✓ Click on the plus sign next to each resource to expand it and click on each variable to verify that the word widths are correct.

Remember that the inputs should each be 16-bits wide. Note that the “b” input word with is indicated as 32 bits (Figure 2). This is because it has been “accidentally left as an integer type.

- ✓ Edit the C++ source code and make “b” an int16.
- ✓ Save the C++ source
- ✓ Go back to architectural constraints and verify that the bit widths are correct.



Variable: b

Array Word Width: 32

Streaming: 0

Base Address: 0

Base Bit: 0

Figure 2 Architectural Constraints – Checking Bit Widths

- ✓ Double-click on one of the variables to cross-probe to the C++ source.

Generate RTL

- ✓ Click on the "Generate RTL" icon in the design bar.
- ✓ Double click on the "rtl.vhdl" or the "rtl.v" icon in their respective folder in the output files folder to open the file in the text editor (Note: you should have enabled Verilog output in lab2 to use the rtl.v file).
- ✓ Click on the "Find" icon in the design bar and search for "ENTITY lab2 IS" (VHDL) or "module lab2"(Verilog). You should see:

```
ENTITY lab2 IS
  PORT(
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    a_rsc_z : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    b_rsc_z : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    c_rsc_z : OUT STD_LOGIC_VECTOR (33 DOWNTO 0)
  );
END lab2;

Or

module lab2 (
  clk, rst, b_rsc_z, a_rsc_z, c_rsc_z
);
  input clk;
  input rst;
  input [31:0] b_rsc_z;
  input [31:0] a_rsc_z;
  output [33:0] c_rsc_z;
```

- ✓ Examine the port widths and note that Catapult has "flattened the arrays for a,b, and c. Thus the two 16-bit elements of "a" are mapped to a single 32-bit port.
- ✓ Try cross probing to the C++ source by clicking on the RTL ports that are highlighted in bold text.

Simulation

- ✓ Double-click on the "Simulate rtl.vhdl" or "Simulate rtl.v" icon in the Verification->Modelsim folder. This will launch Modelsim and compile rtl.vhdl or rtl.v, including all of the necessary libraries.
- ✓ Once Modelsim has finish compiling enter the following command at the command prompt:

For VHDL output

source ../../sim1Vhd.tcl

For Verilog output

urce ../../sim1Vlg.tcl

This will compile lab2_tb1.vhd (Located in the lab2 directory), which is a simple VHDL test bench that will exercise the RTL from Catapult. The test bench generates new input values for a[0] and a[1] every clock cycle after reset has been asserted. The b[0] and b[1] inputs are set equal to zero. This means that the algorithm is essentially copying a[0] and a[1] to c[0] and c[1] respectively.

Looking at the waveform display in Modelsim, we see that although the values of a[0] and a[1] are changing every clock cycle, the output on c[0] and c[1] is only changing every three clock cycles after the first algorithm iteration (Figure 3). Why is this?

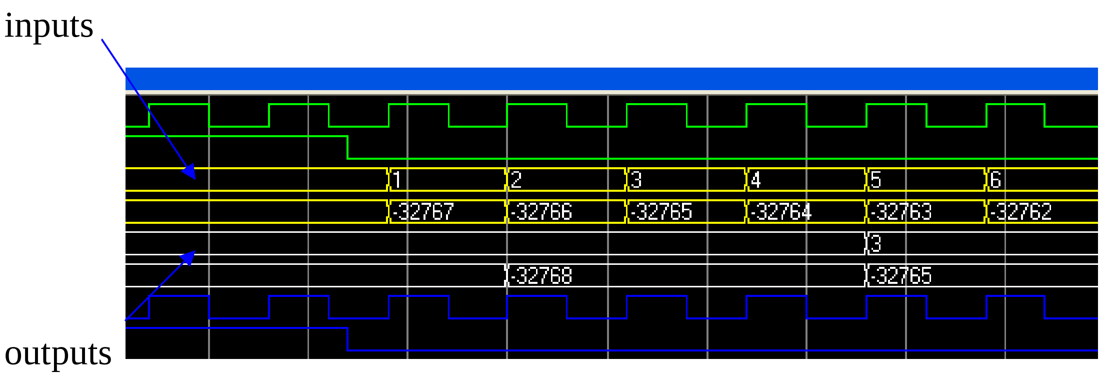


Figure 3 Simulation using “wire” interfaces

The reason why the output is only written every three clock cycles can be seen by looking at the Gantt chart (Figure 4).

✓ Click on schedule to launch the Gantt chart

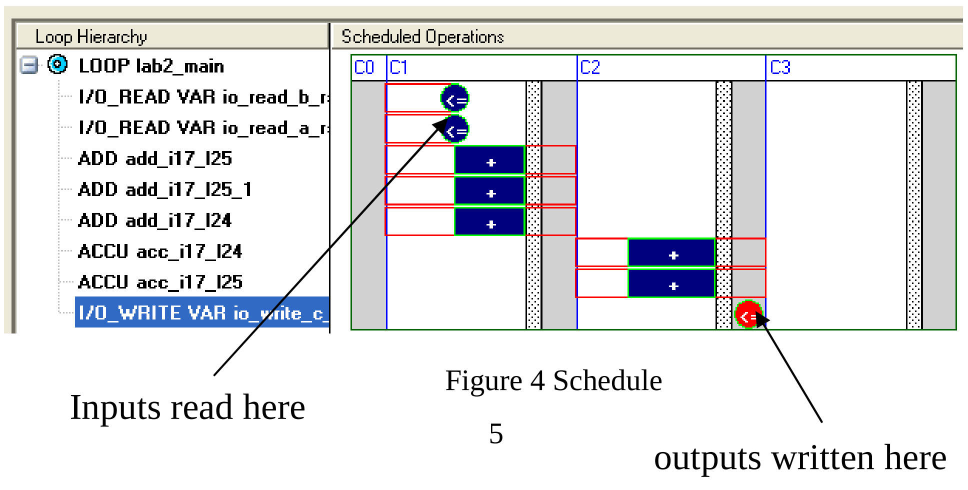


Figure 4 Schedule

It can be seen from figure 4 that each iteration of the algorithm, or main loop, takes three clock cycles. Thus the inputs and outputs are only read/written every three clock cycles. In addition to that the reads and writes are unconditional so even though the values of a[0] and a[1] are changing every clock cycle, they are only being read by the RTL every third clock cycle.

✓ Close Modelsim

Using "wire" Interfaces with an acknowledge

Catapult also provides interface components with acknowledge control signals that are asserted each time an interface is read or written. The next section of this lab covers the use of these interface components. The components are:

'mgc_in_wire_en'

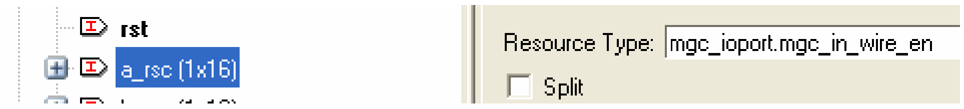
results in an input signal '**z**' and a single control output **lz**. The input is assumed to be registered outside the design and ready whenever the design reads it. Reading the input is acknowledged with setting **lz** to '1' at the clock, where the data is taken over.

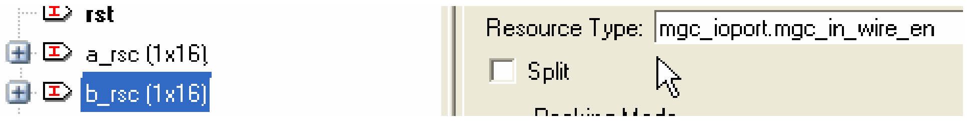
'mgc_out_stdreg_en'

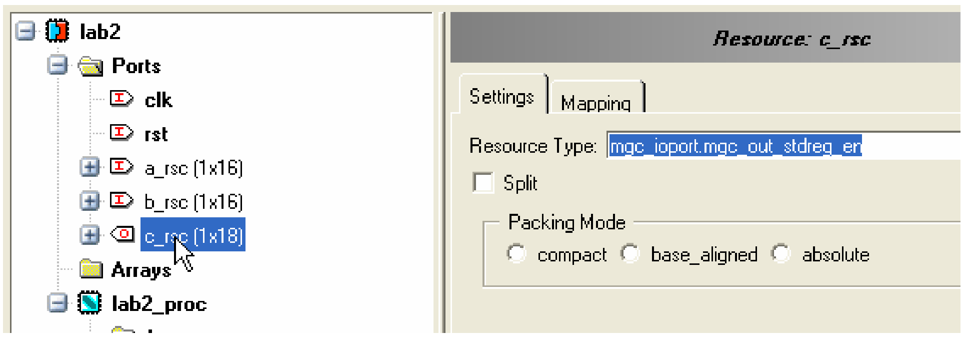
results in an output signal '**z**' and a single control output **lz**. Each output value is valid for one full cycle and each write is flagged with the signal **lz** set to '1'. The output is undefined in all other cycles, when the control **lz** is '0' .

Architectural Constraints

✓ Click on "Architectural Constraints" and set both the "a" and "b" resource types to "mgc_inwire_en. Set the "c" resource type to "mgc_out_stdreg_en (Figure 5).


rst
a_rsc (1x16)
Resource Type: mgc_ioport.mgc_in_wire_en
Split


rst
a_rsc (1x16)
b_rsc (1x16)
Resource Type: mgc_ioport.mgc_in_wire_en
Split



Resource Type: mgc_ioport.mgc_out_stdreg_en

☐ Split

Packing Mode: ○ compact ○ base_aligned ○ absolute

Figure 5 Setting Resource Types

Generate RTL

- ✓ Click on the "Generate RTL" icon in the design bar.
- ✓ Double click on the rtl.vhdl or rtl.v icon in the output files folder to open the file in the text editor.
- ✓ Click on the "Find" icon in the design bar and search for "ENTITY lab2 IS" (VHDL) or "module lab2"(Verilog). You should see:

```
ENTITY lab2 IS
  PORT(
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    b_rsc_lz : OUT STD_LOGIC;
    b_rsc_z : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    a_rsc_lz : OUT STD_LOGIC;
    a_rsc_z : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    c_rsc_lz : OUT STD_LOGIC;
    c_rsc_z : OUT STD_LOGIC_VECTOR (33 DOWNTO 0)
  );
END lab2;
```

Or:

```
module lab2 (
  clk, rst, b_rsc_lz, b_rsc_z, a_rsc_lz, a_rsc_z, c_rsc_lz, c_rsc_z
);
  input clk;
  input rst;
  output b_rsc_lz;
  input [31:0] b_rsc_z;
  output a_rsc_lz;
  input [31:0] a_rsc_z;
  output c_rsc_lz;
  output [33:0] c_rsc_z;
```

- ✓ Examine the port widths and note that “a”, “b” and ”c” each have an “_lz” port added that will be asserted any time the port is read or written.

Simulation

- ✓ Double-click on the “Simulate rtl.vhdl” or “Simulate rtl.v” icon in the Verification->Modelsim folder. This will launch Modelsim and compile rtl.vhdl or rtl.v, including all of the necessary libraries.

- ✓ Once Modelsim has finish compiling enter the following command at the command prompt:

For VHDL output

```
source ../../sim2Vhd.tcl
```

For Verilog output

```
source ../../sim2Vlg.tcl
```

This will compile lab2_tb2.vhd (Located in the lab2 directory), which is a simple VHDL test bench that will exercise the RTL from Catapult. The test bench generates new input values for a[0] and a[1] every time “a_rsc_lz” has been asserted. The b[0] and b[1] inputs are set equal to zero. This means that the algorithm is essentially copying a[0] and a[1] to c[0] and c[1] respectively.

Looking at the waveform display in Modelsim, we see that the values of a[0] and a[1] are changing every time “a_rsc_lz” has been asserted. The outputs on c[0] and c[1] are only changing every time c_rsc_lz is asserted and exactly match the inputs on a[0] and a[1] (Figure 6).

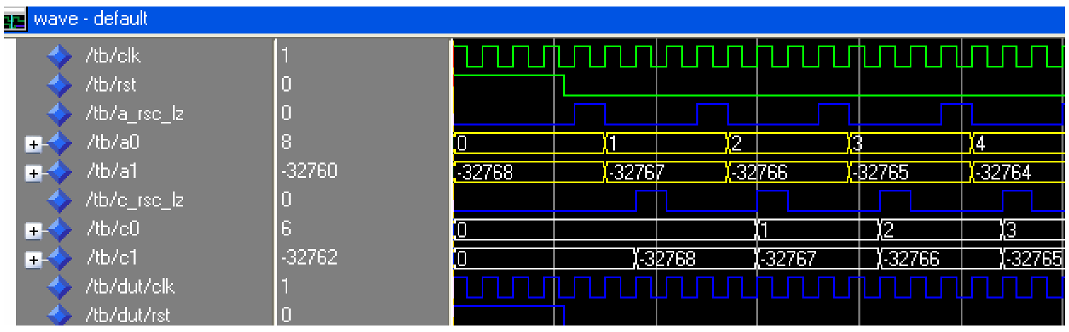


Figure 5 Simulation using “mgc_inwire_en”

- ✓ Close all windows and select File->new project

END