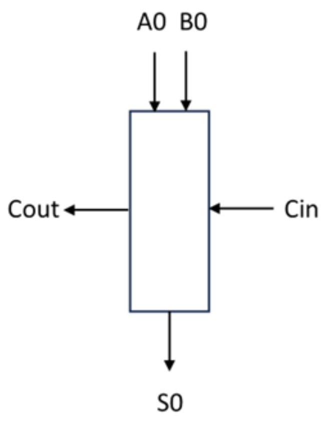
Comp3402 The Full Adder

C.B.Price January 2024

Purpose	(i) To learn how to use the Structural Architecture, (ii) To apply this to construct a full adder
Files Required	Vivado software (on machines, free download) and zipped projects
ILO Contribution	LO 5
Send to Me	nix
Homework	Read chapter 13

1. Prep for coding the Adder Bit-Slice

(a) Here's a single bit slice which has three inputs, Cin, A0 and B0 and two outputs, the sum S0 and carry out Cout. Create the truth table for all 8 possible input values.



C_{in}	A0	B0	S0	C_{out}

(b) Look at the rows where S0 = 1. Create a logical mini-term for each row. E.g., if you have a row with Cin=0, A0 = 1, B0 = 0, then the mini-term will be **A . ~B . ~C.**

(c) Combine the 4 mini-terms to create a logical expression for **S0 =**

(d) Repeat (b) and (c) to get an expression for Cout.

2. Coding the Adder Bit-Slice

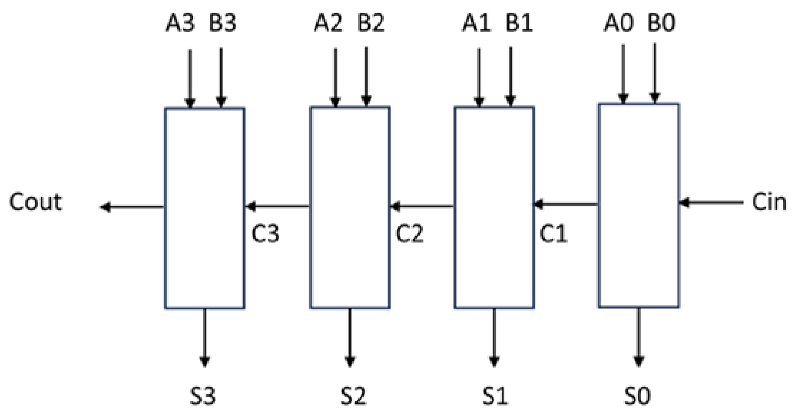
(a) Fire up Vivado and open the source file **fullAdderSlice.vhd**. You will see that 8 mini-terms have been declared for you, the first 4 are for the sum and terms 5-8 are for the carry.

(b) Code your expressions for the 8 mini-terms you have found above. Here's an example how to code the mini-term **A . ~B . ~C:** **A and not(b) and not(C)**.

(c) Now code the expressions for sum **S** and for **Cout**. Do not attempt to synthesise, we must complete the second source file.

3. **Assembling the complete 4-bit Adder.**

Here's the *structure* of our 4-bit added made from 4 of the Adder Bit-Slices you have just coded.



(a) Open up the source file **FourBit_Full_Adder.vhd** where the Structural Architecture is used. You will see the following code which defines the **Component fullAdderSlice.**

```
COMPONENT fullAdderSlice
  Port (A : in std_logic;
        B : in std_logic;
        Cin : in std_logic;
        S : out std_logic;
        Cout : out std_logic);
END COMPONENT;
```

The code which defines how the bit-slice components are connected is shown in the following begin-end block, though this is incompelete

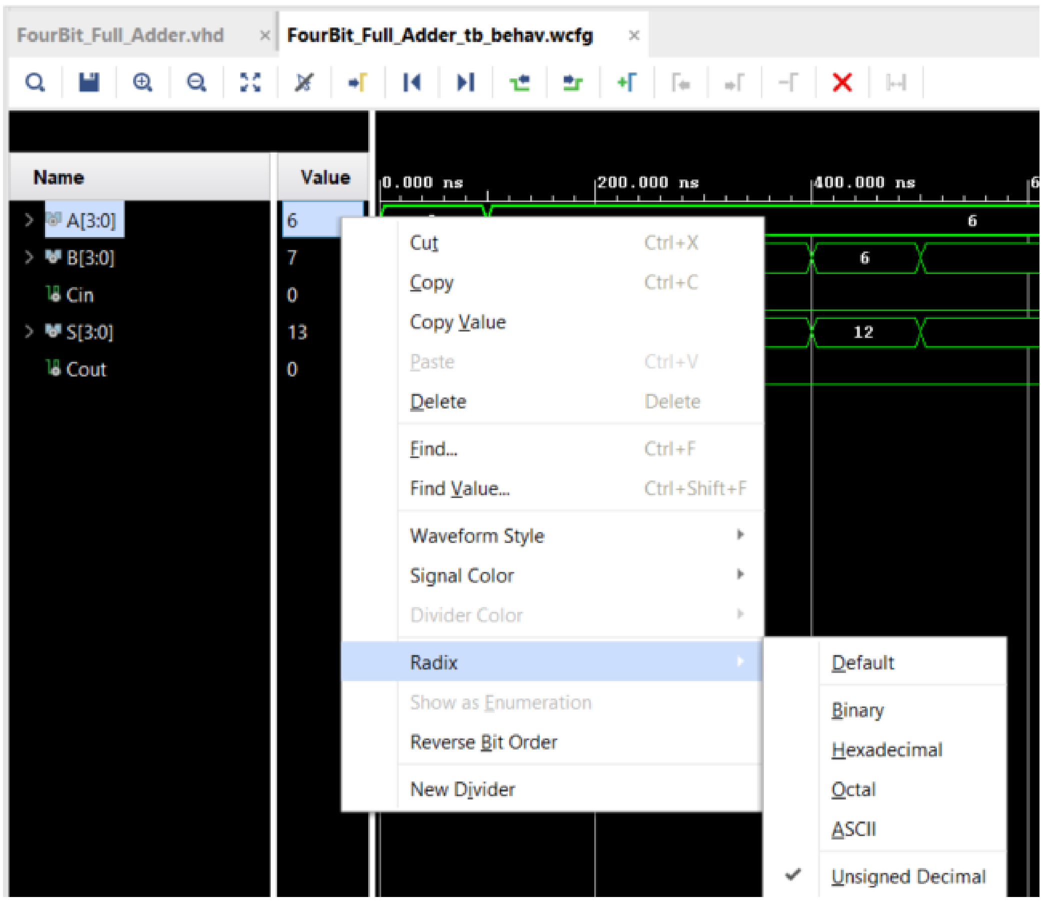
```
BEGIN
stage0: fullAdderSlice PORT MAP ( X(0), Y(0), Cin,  S(0), C(1) ) ;
stage1: fullAdderSlice PORT MAP (     ,     ,     ,     ,      ) ;
stage2: fullAdderSlice PORT MAP (     ,     ,     ,     ,      ) ;
stage3: fullAdderSlice PORT MAP (     ,     ,     ,     , Cout ) ;
END Structural ;
```

Fill in the missing signals. You need to use signals from the **Entity** FourBit_Full_Adder declared up top, specifically in the **Port**.

(b) Now it's time to synthesise the whole circuit.

4. **Running the TestBench**

Run the associated test-bench to check the synthesis. You might like to change the radix of the display so it shows unsigned decimals. You can do this by right-clicking on the value fields:

FourBit_Full_Adder.vhd
FourBit_Full_Adder_tb_behav.wcfg
Name
Value
A[3:0] 6
B[3:0] 7
Cin 0
S[3:0] 13
Cout 0
0.000 ns
200.000 ns
400.000 ns
6
6
12
Cut Ctrl+X
Copy Ctrl+C
Copy Value
Paste Ctrl+V
Delete Delete
Find... Ctrl+F
Find Value... Ctrl+Shift+F
Waveform Style
Signal Color
Divider Color
Radix
Show as Enumeration
Reverse Bit Order
New Divider
Default
Binary
Hexadecimal
Octal
ASCII
✓ Unsigned Decimal