ELEC5566M

FPGA Design for System on Chip

Assignment 1: 2-Bit x 3-Bit Multiplier

**201715540**

**Junnan Liu**

**el23jl2@leeds.ac.uk**

**GitHub username: JannLeo**

**Appendix:**

**MultiplierFirstRow.v**

/\*

\* MultiplierFirstRow

\* ----------------------------

\* By: Junnan Liu

\* For: University of Leeds

\* Date: 2th March 2024

\*

\* Description

\* ------------

\* The module is a submodule which comprises first row of multiplier

\*

\*/

**module** MultiplierFirstRow **(**

**input** **[**1**:**0**]** m**,**

**input** **[**1**:**0**]** q**,**

**input** cIn**,**

**output** **[**1**:**0**]** qOut**,**

**output** cOut**,**

**output** mOut**,**

**output** s

**);**

//The A and B is the variable used in Adder1Bit to add each other.

**wire** A**,**B**;**

// Instantiate A and B to calculate Two addends of 1-bit adder

**and(**A**,**q**[**1**],**m**[**0**]);**

**and(**B**,**q**[**0**],**m**[**1**]);**

//Instantiate the Adder1Bit to act as Full Adder

Adder1Bit adder0 **(**

**.**cin **(**cIn**),**

**.**a **(**A**),**

**.**b **(**B**),**

**.**sum **(**s**),**

**.**cout**(**cOut**)**

**);**

// According to the Submodule for First Row of Multiplier

//The qOut is equal to the q0 and q1.

assign qOut[1:0]=q[1:0];

// According to the Submodule for First Row of Multiplier

// The mOut is equal to m[0]

assign mOut=m[0];

endmodule

**MultiplierRemainingRow.v:**

/\*

\* MultiplierFirstRow

\* ----------------------------

\* By: Junnan Liu

\* For: University of Leeds

\* Date: 2th March 2024

\*

\* Description

\* ------------

\* The module is a submodule which comprises remaining rows of multiplier

\*

\*/

**module** MultiplierRemainRow **(**

**input** pp**,**

**input** m**,**

**input** q**,**

**input** cIn**,**

**output** qOut**,**

**output** cOut**,**

**output** mOut**,**

**output** s

**);**

//The A and B is the variable used in Adder1Bit to add each other.

**wire** A**,**B**;**

// Instantiate A and B to calculate Two addends of 1-bit adder

**and(**A**,**q**,**m**);**

// The B is equal to the pp

**assign** B **=** pp**;**

//Instantiate the Adder1Bit to act as Full Adder

Adder1Bit adder1 **(**

**.**cin **(**cIn**),**

**.**a **(**A**),**

**.**b **(**B**),**

.sum (s),

.cout(cOut)

);

// According to the Submodule for Remaining Rows of Multiplier

//The qOut is equal to the q.

assign qOut = q;

// According to the Submodule for Remaining Rows of Multiplier

//The mOut is equal to the m.

assign mOut = m;

endmodule

**Adder1Bit.v**

/\*

\* Single Bit Adder

\* ----------------------------

\* By: Thomas Carpenter

\* For: University of Leeds

\* Date: 28th December 2017

\*

\* Description

\* ------------

\* The module is a simple 1-bit Adder using Gate Level Verilog

\*

\*/

**module** Adder1Bit **(**

// Declare input and output ports

**input** a**,**

**input** b**,**

**input** cin**,**

**output** cout**,**

**output** sum

**);**

// Declare several single-bit wires that we can

// use to interconnect the gates. You can use

// any name you like as long as it contains only

// a-z, A-Z, underscore (\_), and 0-9. Names can't

// start with a digit.

**wire** link1**,**link2**,**link3**;**

// Instantiate gates to calculate sum output

**xor(**link1**,**a**,**b**);**

**xor(**sum**,**link1**,**cin**);**

// Instantiate gates to calculate carry (cout) output

**and(**link2**,**a**,**b**);**

**and(**link3**,**cin**,**link1**);**

**or** **(**cout**,**link2**,**link3**);**

**endmodule**

**Multiplier2x3.v:**

/\*

\* Main Module for 2bit x 3bit Multiplier

\* ----------------------------

\* By: Junnan Liu

\* For: University of Leeds

\* Date: 2th March 2024

\*

\* Description

\* ------------

\* The module is a main Module for 2bit x 3bit Multiplier

\*

\*/

// The Multiplier2x3 module conatins 2 input: 2 bits multiplier m and 3 bits multiplier q

// It also contains the 5 bits output named p

**module** Multiplier2x3 **(**

**input** **[**1**:**0**]** m**,**

**input** **[**2**:**0**]** q**,**

**output** **[**4**:**0**]** p

**);**

// These are the variables for each submodule of the cOut multiplier.

**wire** cOut\_1**,**cOut\_2**,**cOut\_3**;**

// This variable represents the count of '1's in the MultiplierFirstRow of qOut.

**wire** **[**1**:**0**]** qOut1**;**

//This variable represents the count of '2's in the MultiplierFirstRow of qOut.

**wire** **[**1**:**0**]** qOut2**;**

// This variable represents the count of '3's in the MultiplierRemainRow of qOut.

**wire** qOut3**;**

// This variable represents the count of '4's in the MultiplierRemainRow of qOut.

**wire** qOut4**;**

// This variable represents the count of '2's in the MultiplierFirstRow of s.

**wire** s\_2**;**

// These are the variables for each submodule of the mOut multiplier.

**wire** mOut\_1**,**mOut\_2**,**mOut\_3**,**mOut\_4**;**

// This variable represents the count of '2's in the MultiplierFirstRow of m.

**wire** **[**1**:**0**]**m\_2**;**

// assign the value to m\_2

**assign** m\_2**[**0**]=**m**[**1**];**

**assign** m\_2**[**1**]=**1'b0**;**

// Calculate the p[0] through q[0] & m[0]

**and(**p**[**0**],**q**[**0**],**m**[**0**]);**

// Instantiate MultiplierFirstRow1 to calculate qOut1, cOut\_1, mOut\_1 and p[1]

MultiplierFirstRow MultiplierFirstRow1**(**

**.**m**(**m**[**1**:**0**]),**

**.**q**(**q**[**1**:**0**]),**

**.**cIn**(**1'b0**),**

**.**qOut**(**qOut1**),**

**.**cOut**(**cOut\_1**),**

**.**mOut**(**mOut\_1**),**

**.**s**(**p**[**1**])**

**);**

// Instantiate MultiplierFirstRow2 to calculate qOut2, cOut\_2, mOut\_2 and s\_2

MultiplierFirstRow MultiplierFirstRow2**(**

**.**m**(**m\_2**),**

**.**q**(**qOut1**),**

**.**cIn**(**cOut\_1**),**

**.**qOut**(**qOut2**),**

**.**cOut**(**cOut\_2**),**

**.**mOut**(**mOut\_2**),**

**.**s**(**s\_2**)**

**);**

// Instantiate MultiplierRemainRow3 to calculate qOut3, cOut\_3, mOut\_3 and p[2]

MultiplierRemainRow MultiplierRemainRow3**(**

**.**pp**(**s\_2**),**

**.**m**(**mOut\_1**),**

**.**q**(**q**[**2**]),**

**.**cIn**(**1'b0**),**

**.**qOut**(**qOut3**),**

**.**cOut**(**cOut\_3**),**

**.**mOut**(**mOut\_3**),**

**.**s**(**p**[**2**])**

**);**

// Instantiate MultiplierRemainRow4 to calculate qOut4, p[4], mOut\_4 and p[3]

MultiplierRemainRow MultiplierRemainRow4**(**

**.**pp**(**cOut\_2**),**

**.**m**(**mOut\_2**),**

**.**q**(**qOut3**),**

**.**cIn**(**cOut\_3**),**

.qOut(qOut4),

.cOut(p[4]),

.mOut(mOut\_4),

.s(p[3])

);

endmodule

**simulation/Multiplier2x3\_tb.v:**

// /\*!\notex{

// Multiplier2x3 Test Bench

// ---------------------------

// By: Junnan Liu

// For: University of Leeds

// Date: 2th March 2024

//

// Short Description

// -----------------

// This is a simple test bench module to test the Multiplier2x3.

// module with a few stimulii.

//

// }!\*/

// Timescale indicates unit of delays.

// `timescale unit / precision

// Where delays are given as:

// #unit.precision /\*!\notex{

//

// Let's stick with a "unit" of 1ns. You may choose the "precision".

//

// e.g for `timescale 1ns/100ps then:

// #1 = 1ns

// #1.5 = 1.5ns

// #1.25 = 1.3ns (rounded to nearest precision)

// }!\*/

`timescale 1 ns**/**100 ps

// Test bench module declaration

// Always end test bench module names with \_tb for clarity, and use no port list

**module** Multiplier2x3\_tb**;**

// Test Bench Generated Signals

**reg** **[**1**:**0**]** m**;**

**reg** **[**2**:**0**]** q**;**

// DUT Output Signals

**wire** **[**4**:**0**]** p**;**

// Device Under Test. Instance typically named "dut", or "ModuleName\_dut".

Multiplier2x3 Multiplier2x3\_dut**(**

**.**m**(**m**),**

**.**q**(**q**),**

**.**p**(**p**)**

**);**

//We can then calculate the expected value using alternate method to the DUT.

**wire** **[**4**:**0**]** expected\_value**;** //1-bit wider than sum to include the carry out.

**assign** expected\_value **=** m**\***q**;** //In this case we use the behavioural + operator

// Integer for-loop

**integer** i**;**

**integer** j**;**

// Test Bench Logic

**initial** **begin**

//Print to console that the simulation has started. $time is the current sim time.

$display**(**"%d nstsimulation started"**,**$time**);**

//Monitor changes to any values listed, and automatically print to the console

//when they change. There can only be one $monitor per simulation.

$monitor**(**"%d ns\tm=%b\tq=%h\tp=%h\t Expected:\tvalue=\%h\t"**,**

$time**,**m**,**q**,**p**,**expected\_value**);**

// Loop for 2bit multiplier m

**for(**i**=**0**;**i**<**4**;**i**=**i**+**1**)begin**

// Loop for 3bit multiplier q

**for(**j**=**0**;**j**<**8**;**j**=**j**+**1**)begin**

m**=**i**[**1**:**0**];**

q**=**j**[**2**:**0**];**

**#**10**;**//wait 1o units.end

$display**(**"%d ns\tsimulation Finished"**,**$time**);** //Finished

**end**

**end**

**end**

**endmodule**