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**

# 2-bit by 3-bit Multiplier Assessment

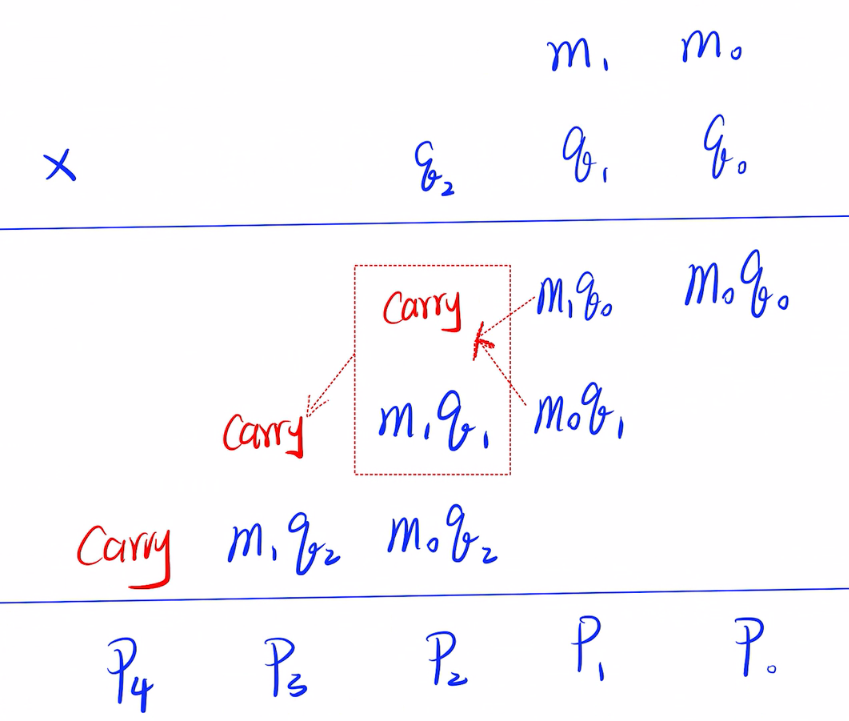
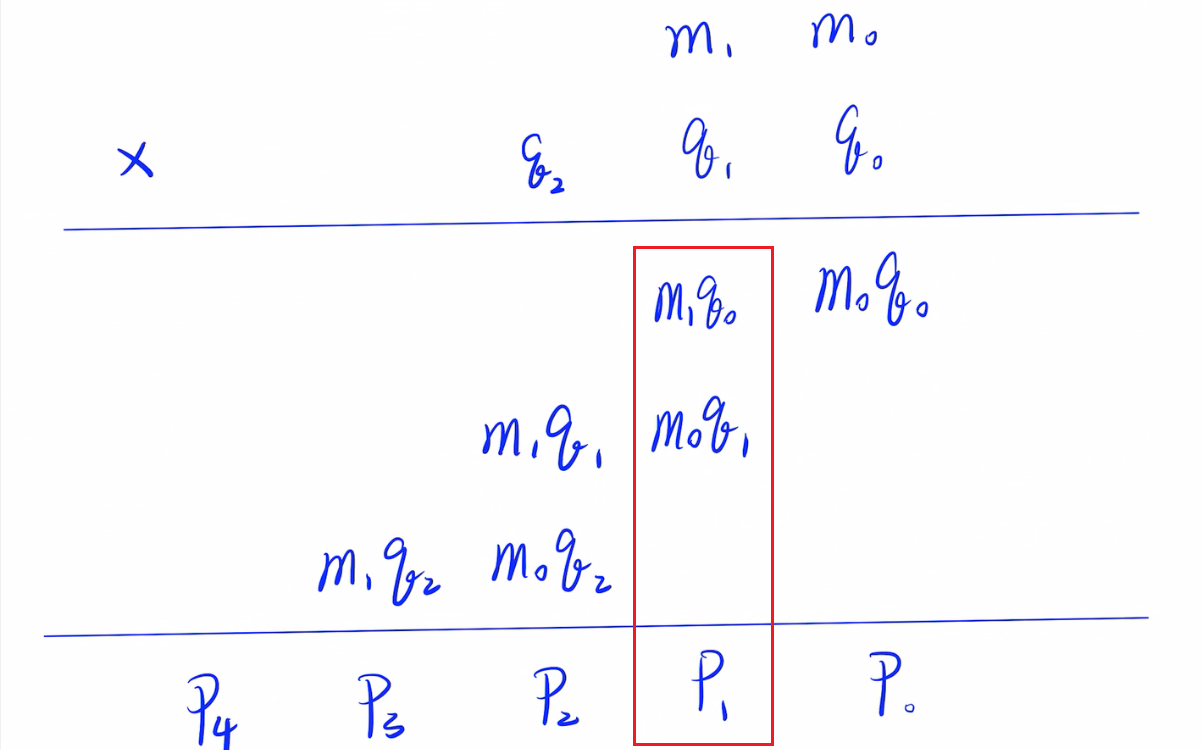
The describes of following files are provided in Table 1 Description of project-related documents.

#### Implementation principle

* + After knowing the variables of multiplier, we need to know the structure of the 2-Bit by 3-Bit Multiplier. As the figure shown in Figure 1 The mark of 2-Bit by 3-Bit Multiplier Structure, we can see there have 4 parts need us to achieve. Thus, we need to figure out what input and output of these multiplier in submodule.

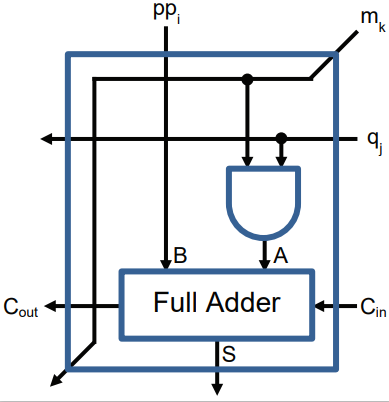
##### To calculate the p0, the p0 is easy to calculate since it is equal to m0 x q0.

* + To calculate the p1, according to the Figure 2 Submodule for First Row of Multiplier, we can see the Function of this struct is to calculate S and Cout through Table 2 The meanings of different variable in Submodule for First Row of Multiplier.
    - According to the principle of above figure, the picture shown below left illustrate the theory of the 2 bits by 3 bits multiplier. The red frame shows below is just like the function of 1st.First Row of Multiplier (Also it generates the carry to 2nd.First Row of Multiplier).



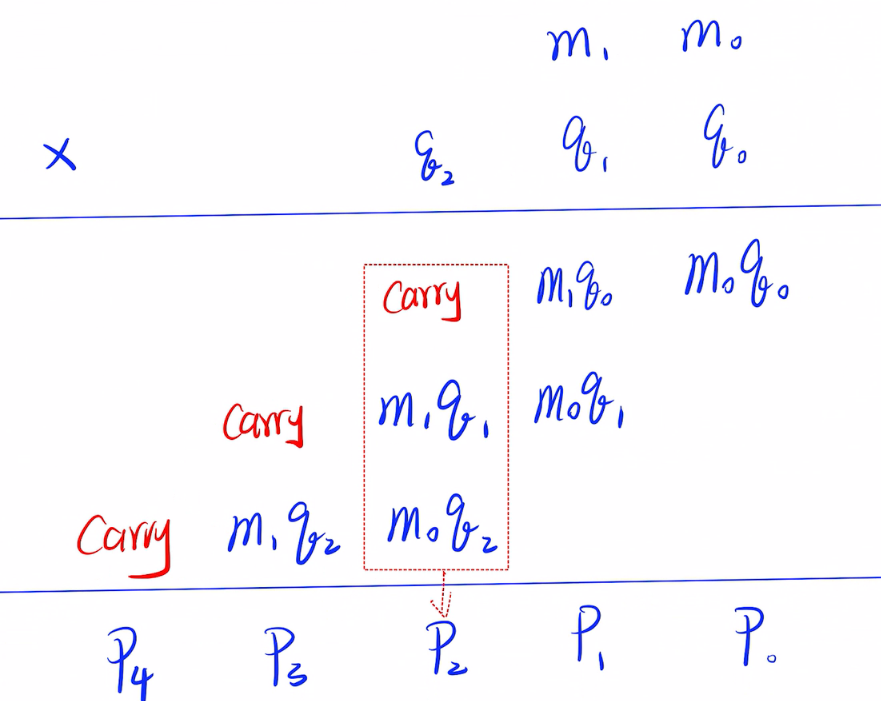
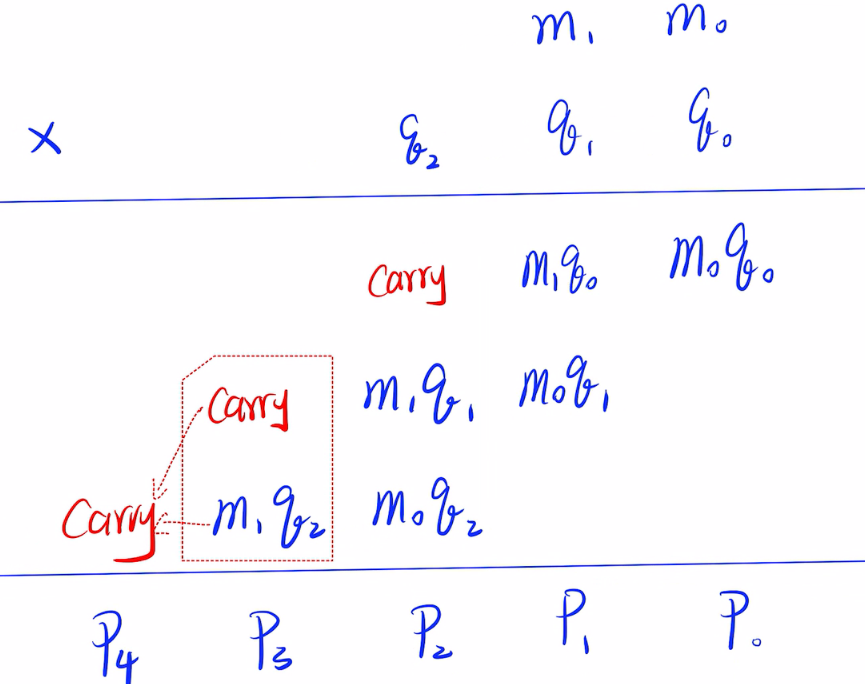
* + The main function of 2th.First Row of Multiplier is to add the carry of last part (calculate P1) and m1 x q1 to generate the S and carry to 3th.Remaining Rows of Multiplier to calculated (As the figure shown above right).
  + To calculate the p3 and p4, we need to see the structure of Remaining Rows of Multiplier:

|  |  |
| --- | --- |
| **Variable** | **How to calculate** |
| **A** | qj x mk |
| **B** | ppi = S in the previous line |
| **S** | A+B=(qj x mk) + ppi |
| **Cout** | carry of the S |



* + We can see the Function of this struct is to calculate S and Cout through above table.

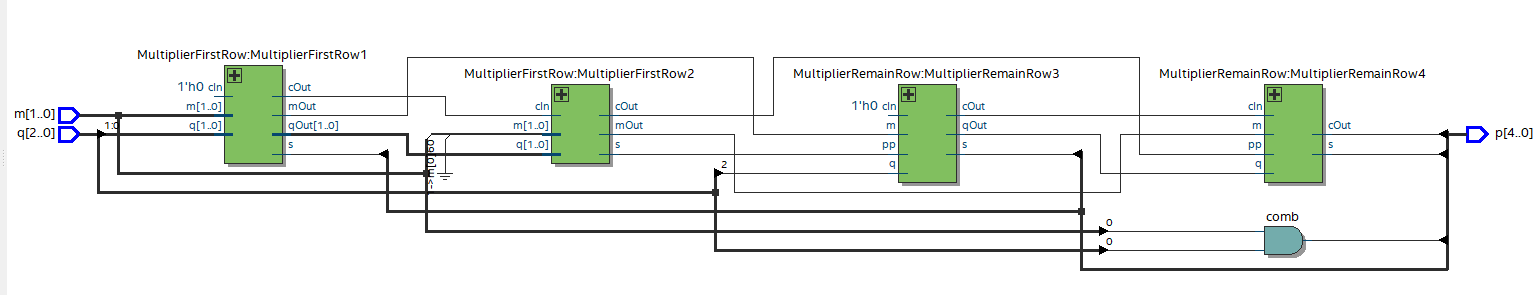
##### According to the principle of above figure, the picture shown below left illustrate the theory of the 2 bits by 3 bits multiplier. The red frame shows below is to show the main function of 3rd.Remaining Rows of Multiplier is to calculate p2 = carry + m1 x q1 + m0 x q2.

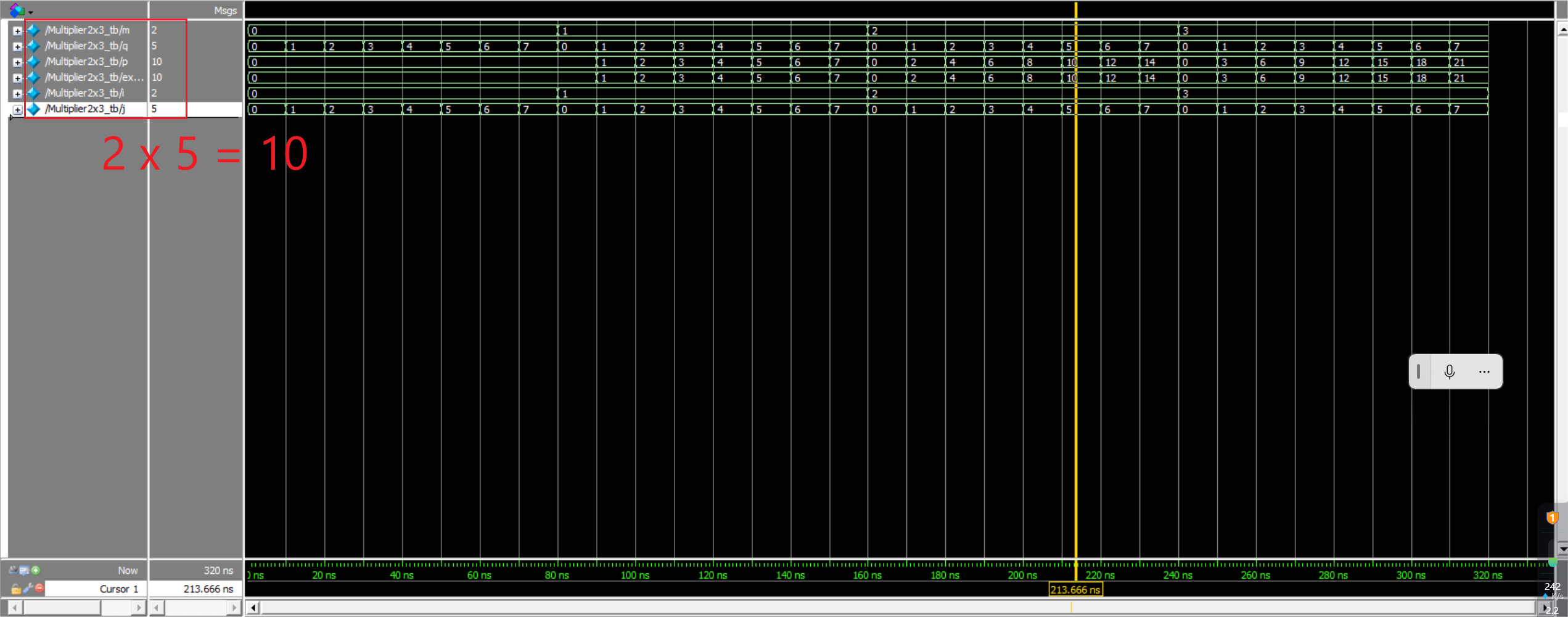
##### As for p3 and p4, it is be calculated in 4th. Remaining Rows of Multiplier. The 4th. Remaining Rows of Multiplier function is mainly using the carry calculated from 3rd. Remaining Rows of Multiplier and m1 x q2 to calculate p3 and generate the carry to act as p4 (as figure shown above right).

#### Output and Simulation

The RTL viewer output of simulation as follows, and it shows the code matches the function of the design task diagrams:



* + The mainly function of the test bench of the code is to simulation the process of 2 bits by 3 bits multiplier. The figure below is the output of simulation using the code of test bench, from which we can see when i = 2 and j = 5, the m would be equal to 2 , q would be equal to 5 and it can be calculated that both of the p and expected\_value are 10, which shows that the algorithm of multiplier2x3 is correct.



**Appendix:**

|  |  |
| --- | --- |
| **File** | **Purpose** |
| **Multiplier2x3.v** | Main Module for 2bit x 3bit Multiplier |
| **MultiplerFirstRow.v** | Submodule which comprises first row of multiplier |
| **MultiplerRemainingRow.v** | Submodule which comprises remaining rows of multiplier |
| **Multiplier2x3.sdc** | Generic timing constraints file |
| **Structure/\*** | Diagrams of the multiplier structure |
| **simulation/Multiplier2x3\_tb.v** | Test bench of the 2bit x 3bit Multiplier |
| **Adder1Bit.v** | Submodule which comprises 1-bit Adder |

Table 1 Description of project-related documents

|  |  |
| --- | --- |
| **Variable** | **How to calculate** |
| **A** | q1 x mk |
| **B** | q0 x mk+1 |
| **S** | A + B = q1 x mk + q0 x mk+1 |
| **Cout** | Carry of S |

Table 2 The meanings of different variable in Submodule for First Row of Multiplier

图示, 示意图

描述已自动生成

Figure 1 The mark of 2-Bit by 3-Bit Multiplier Structure

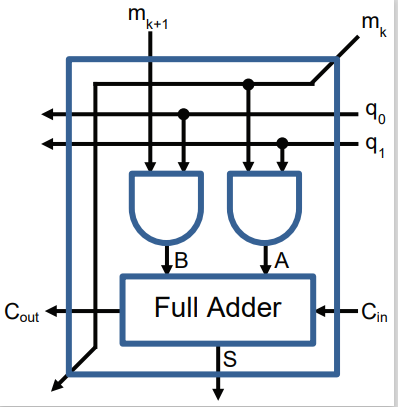


Figure 2 Submodule for First Row of Multiplier

**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**

**Reference:**

**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

\*

\*/

[1] S. Freear, D. D. Cowell, D. T. Carpenter, and D. H. Clegg, ‘Assignment 1: 2-Bit x 3-Bit Multiplier’.