**113學年度**

**國立中山大學**

**課程名稱:硬體描述語言**

RTL設計:

**授課教師:蕭勝夫**

**學生學號/班級/姓名: B103040045/資工114/楊貽婷**

**Adder Designs Using Verilog Structural, Dataflow, and Behavioral Modeling**

**作業/成果報告/專題**

1. Structure level: 將Full Adder以邏輯閘組合的方式撰寫出來，最後在32-bit ripple carry adder當中用generate語法呼叫出32個Full adder計算32-bit加法。
2. Dataflow level: 用assign語法，直接描寫加法。
3. Behavior level: 用always語法偵測Input port是否有變化，再執行計算。
4. Structure level with register: 在第一點架構上，計算完的結果先經過D-flip flop再送到output port。
5. Dataflow level with register: 在第二點架構上，計算完的結果先經過D-flip flop再送到output port。
6. Behavior level with register: 在第三點架構上，計算完的結果先經過D-flip flop再送到output port。

Testbench: 將會用到的腳位宣告好，再呼叫6種不同設計的加法器建立instances，在initial begin – end區塊，用$random隨機分派值，之後將預期輸出跟加法器做比對，檢查是否有錯。

* Pre\_syn\_testbench跟Post\_syn\_testbench的差別為: 有沒有引用sdf檔案。
* RTL波形圖 (pre\_simulation)

此步驟的目的為驗證RTL程式正確，發現當三種方式所寫出的combinational ripple carry adder，一發現input值更新後，也會同時馬上更新output。但如果將這種combinational加入D-flip-flop後，在positive edge時才會更新output數值。

一張含有 螢幕擷取畫面, 電子產品, 行, 鮮豔 的圖片

自動產生的描述

* Gate-level (delay optimize)波形圖

將上述RTL程式經過DC compiler後，在經由post simulation驗證所得出的波形圖與RTL波型圖表現一致。

一張含有 螢幕擷取畫面, 電子產品, 鮮豔, 行 的圖片

自動產生的描述

* 請觀察三種modeling的數據與波型是否相同，並解釋你認為的原因:

三種modeling在波形圖上的表現，不論是數據還是波型都相同，可能原因為這三種modeling是以不同描述方式撰寫出相同功能。例如structure-level常以hierarchy的結構，從小module慢慢組成大module，但在behavior-level為電路功能的行為描述。

* 數據表格 (提供在下頁):
  + 從combination circuit設計加入暫存器後，面積和延遲皆會增大。
  + 從三種modeling來看的話，dataflow和behavior數值一樣，且在area-optimized and delay-optimized分別有最小面積和最小延遲。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Area () | | | Delay  (ns) | Power (W) | | |
| CL | SL | Total | Dynamic  (uW) | Leakage  (nW) | Total  (mW) |
| adder\_structure | Delay | 92.482562 | 0 | 92.482562 | 0.473287 | 118.4376 | 88.0144 | 0.1185 |
| Area | 48.107521 | 0 | 48.107521 | 0.766279 | 35.8175 | 27.4608 | 0.35845 |
| Between | 67.132802 | 0 | 67.132802 | 0.619783 | 59.7878 | 50.9201 | 0.59839 |
| adder\_structure\_reg | Delay | 94.970882 | 30.792960 | 125.763843 | 0.486066 | 310.9370 | 111.876 | 0.311 |
| Area | 48.107521 | 30.792960 | 78.900481 | 0.780511 | 155.4213 | 52.9503 | 0.1555 |
| Between | 67.547522 | 30.792960 | 98.340482 | 0.618289 | 211.3923 | 78.3415 | 0.2115 |
| adder\_dataflow | Delay | 105.909122 | 0 | 105.909122 | 0.086346 | 577.5765 | 95.2921 | 0.5777 |
| Area | 33.488641 | 0 | 33.488641 | 0.890143 | 23.1684 | 17.7645 | 0.23186 |
| Between | 52.254721 | 0 | 52.254721 | 0.488245 | 53.9102 | 32.3345 | 0.53943 |
| adder\_dataflow\_reg | Delay | 109.693442 | 30.792960 | 140.486402 | 0.10077 | 144.10 | 122.8804 | 1.4411 |
| Area | 33.488641 | 30.792960 | 64.281601 | 0.906512 | 126.1914 | 43.2527 | 0.1262 |
| Between | 52.099201 | 30.792960 | 82.892161 | 0.503641 | 238.7401 | 56.7549 | 0.2388 |
| adder\_behavior | Delay | 105.909122 | 0 | 105.909122 | 0.086346 | 577.5765 | 95.2921 | 0.5777 |
| Area | 33.488641 | 0 | 33.488641 | 0.890143 | 23.1684 | 17.7645 | 0.23186 |
| Between | 52.254721 | 0 | 52.254721 | 0.488245 | 53.9102 | 32.3345 | 0.53943 |
| adder\_behavior\_reg | Delay | 109.693442 | 30.792960 | 140.486402 | 0.10077 | 1.4410 | 122.8804 | 1.4411 |
| Area | 33.488641 | 30.792960 | 64.281601 | 0.906512 | 126.1914 | 43.2527 | 0.1262 |
| Between | 52.099201 | 30.792960 | 82.892161 | 0.503641 | 238.7401 | 56.7549 | 0.2388 |

* 心得:

先前在必修課「數位系統實驗」上已有撰寫Verilog的經驗，但本次作業還是花費許久時間在上面。其中一個可能原因為先前所使用的軟體為Vivado並不是這堂課所要求的VCS + nWave，因此花了不少時間適應。加上先前Vivado只需要按按鈕就好，不像這次作業一樣需要對Pre simulation、compile、Post simulation做相關額外操作，所以在操作上研究許久。還有一點是開啟nWave時，等待時間等蠻久的，雖能順利操作，但就是跑得有點慢。

在撰寫RTL程式的部分沒甚麼難處，而利用nWave如上段所述花了一些時間，到這邊為止為我很熟悉的流程，另外值得一提的是，非常感謝助教已經提供好相關模板並且幾乎可以沿用，讓我不用重新研究，在Testbench部分也解釋得不錯，省下不少自己研究的時間。而之後的compile過程簡直是惡夢，在這邊卡了很久，因為調period一直調不好，所以大概跑了上百次，而且總共有18種組合，是個非常枯燥乏味的流程。Post simulation跟Pre simulation的流程差不多，但是中途遇到shell檔無法執行，透過上網搜尋和詢問ChatGPT解決。