디지털 논리2 및

컴퓨터 공학 기초 실험2   
보고서

과제제목: Carry Look-ahead Adder (CLA)

실험일자: 2017년 09월 08일 (금)

제출일자: 2017년 09월 14일 (목)

학 과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 금요일 5, 6, 7

학 번: 2012722028

성 명: 장한별

1. 제목 및 목적
   1. 제목

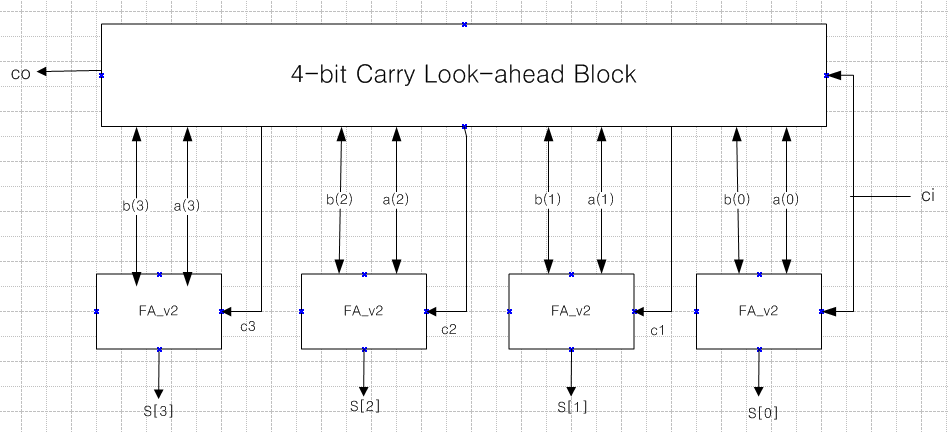
Carry Look-ahead Adder (CLA)

* 1. 목적

4-bit Ripple Carry Adder(RCA) 를 이용하여 32-bit RCA를 설계한다. RCA 의 시간이 오래 걸린다는 단점을 보완한 Carry Look-ahead Adder(CLA)의 원리를 이해하고 32-bit CLA 를 설계한다. RCA 와 CLA 의 결과는 같지만 Timing Analysis 를 이용해 주파수, 즉, 계산 속도 차이를 비교한다.

1. 원리(배경지식)
2. **Carry Look-ahead Adder** .

Ripple Carry Adder의 단점은 최하위 bit의 Full Adder에 나온 Carry out 값이 그 다음 bit의 Full Adder의 Carry in 값으로 들어간다. 이런 방식으로 차례로 Carry가 넘어가야 다음 bit의 Full Adder가 계산되므로 bit가 많아질수록 시간이 오래 걸릴 수 밖에 없는 구조이다. 이러한 단점을 보완한 방식이 Carry Look-ahead Adder(CLA) 다. 두 개의 Input 값과 처음의 Carry in이 입력되면 각각의 bit의 Full Adder에서 나오는 Carry out 값이 계산되어 속도가 빨라지게 된다. 이렇듯 Carry 값을 미리 볼 수 있어서 Carry Look-ahead Adder라는 이름이 불려진다



**<4-bit Carry Look-ahead Adder>**

위의 그림은 볼 수 있듯 Sum의 값은 각각의 Full Adder에서 나오고 Carry 값은 Carry Look-ahead Adder 에서 나온다. 따라서 Full Adder를 구현할 때에는 Carry out 값을 고려하지 않아도 된다.Carry Look-ahead Adder에서 Carry out값을 계산하기 위한 Generation signal(Gi) 과Propagation signal(Pi)은

Gi = Ai Bi Pi  = Ai + Bi

로 나타낸다. Generation signal(Gi) 이란 input A, B에서 Carry가 발생 할 수 있는 경우이기 때문에 A와 B가 모두 1이여야만 Carry가 발생하기 때문에 Gi = Ai Bi 로 나타낼 수 있고 Propagation signal(Pi)이란 Carry in이 들어 올 경우 A 또는 B가 1일 경우 Carry가 발생하기 때문에 Pi = Ai + Bi 로 나타낼 수 있다. 따라서 Carry Out을 Generation signal과 Propagation signal이 발생할 경우를 종합한

 · ·

 · ··

로 나타낼 수 있다. 위의 식을 적용하여 4-bit CLA를 위한 Carry를 구할 수 있다.

·

··· · ··

·· · ··

 · ·· ···

·· · ·· ···

 · ·· ··· ····

|  |
| --- |
|  |

**<32-bit Carry Look-ahead Adder>**

4-bit Carry Look-ahead Adder(CLA) 8개를 직렬로 instance하여 32-bit Carry Look-ahead Adder(CLA)를 위의 그림처럼 설계 할 수 있다.

1. 설계 세부사항
2. **4-bits Carry Look-ahead Adder**

**i) I/O Description**

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | a | 4-bit | Input data A |
| b | 4-bit | Input data B |
| ci | 1-bit | Carry in |
| Output | s | 4-bit | Sum |
| co | 1-bit | Carry out |
| Wire | c1 | 1-bit | Internal carry |
| c2 | 1-bit | Internal carry |
| c3 | 1-bit | Internal carry |

**<I/O Description>**

4-bit CLA이기 때문에 Input a,b 는 4비트이고, ci(Carry in)이 Input 으로 들어간다.

output으로는 s(sum) 과 co(carry out) 이 있고, Wire c1,c2,c3가 있다.

**ii) Module Description**

|  |  |  |
| --- | --- | --- |
| Classification | Name | Description |
| Module | cla4 | 4-bit carry look-ahead adder |
| Instance | U0\_fa\_v2 | Full Adder |
| U1\_fa\_v2 |
| U2\_fa\_v2 |
| U3\_fa\_v2 |
| U4\_fa\_v2 | Carry generation |

**<Module Description>**

기존의 full adder 의 co(carry out)은 필요가 없게 된다. 그렇기에 fa\_v2module을 4개 instance하여 sum 만 출력하도록 한다.

**2) 32-bits Carry Look-ahead Adder**

**i) I/O Description**

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | a | 32-bit | Input data A |
| b | 32-bit | Input data B |
| ci | 1-bit | Carry in |
| Output | co | 1-bit | Carry out |
| s | 32-bit | Sum |
| Wire | c1 | 1-bit | Internal carry |
| c2 | 1-bit | Internal carry |
| c3 | 1-bit | Internal carry |
| c4 | 1-bit | Internal carry |
| c5 | 1-bit | Internal carry |
| c6 | 1-bit | Internal carry |
| c7 | 1-bit | Internal carry |

**< I/O Description >**

Input으로는 32bit의 a,b 그리고 1bit의 ci(carry in)이 있다.

Output 으로는 32bit 의 s(Sum), 1bit의 co(carry out) 이 있다. 그리고 4bit CLA에서 계산한 각각의 carry값들을 전달해주기위해 1 bit의 wire c1, c2, c3, c4, c5, c6, c7가 사용된다.

**ii) Module Description**

|  |  |  |
| --- | --- | --- |
| Classification | Name | Description |
| Module | cla32 | 32-bit CLA |
| Instance | U0\_cla4 | 4-bit CLA |
| U1\_cla4 |
| U2\_cla4 |
| U3\_cla4 |
| U4\_cla4 |
| U5\_cla4 |
| U6\_cla4 |
| U7\_cla4 |

**<Module Description>**

4-bit CLA를 8개 instance하여서 구현하였다.

1. **32-bits CLA with clock**

**i) I/O Description**

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | clock | 1-bit | clock |
| a | 32-bit | Input data A |
| b | 32-bit | Input data B |
| ci | 1-bit | Carry in |
| Output | co\_cla | 1-bit | Carry out |
| s\_cla | 32-bit | Sum |
| Register | reg\_a | 32-bit | Register A |
| reg\_b | 32-bit | Register B |
| reg\_ci | 1-bit | Register carry in |
| reg\_s\_cla | 32-bit | Register sum |
| reg\_co\_cla | 1-bit | Register carry out |
| Wire | wire\_s\_cla | 32-bit | Wire sum |
| wire\_co\_cla | 1-bit | Wire carry out |

**< I/O Description >**

앞뒤로 Register를 연결함으로써 계산시간을 알아볼 수 있다. Input으로는 32bit의 a, b 1bit의 clock, ci 가 있고, Output으로 32bit 의 Sum(s\_cla), 1bit의 carry out(co\_cla)이 있다. Register 에는 32bit의 Input reg\_a, reg\_b 와 1bit의 reg\_ci 있고, 32bit의 Output reg\_s\_cla 와 1bit의 reg\_co\_cla가 있다. 그리고 Register를 연결시켜주는 32bit의 wire\_s\_cla 와 1bit의 wire\_co\_cla 가 있다.

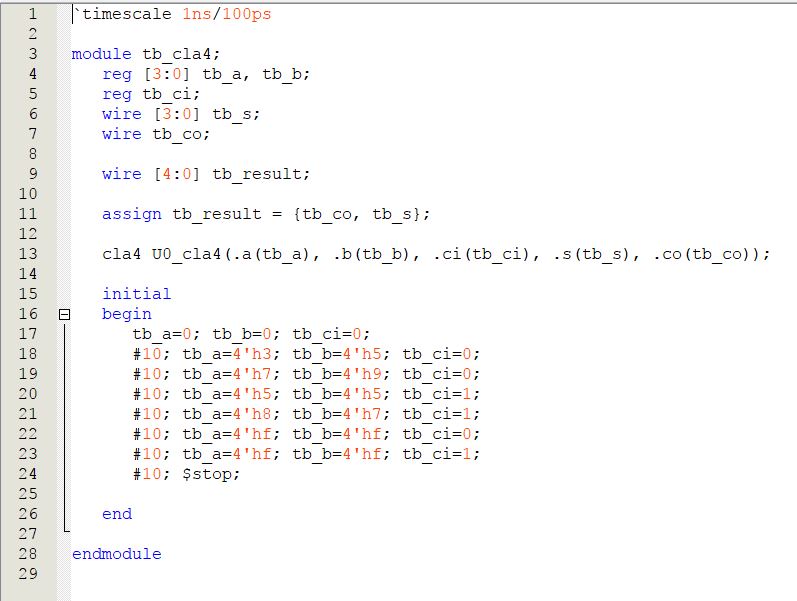
**ii) Module Description**

|  |  |  |
| --- | --- | --- |
| Classification | Name | Description |
| Module | cla\_clk | 32-bit CLA with clock |
| Instance | U0\_cla32 | 32-bit CLA |

**< Module Description>**

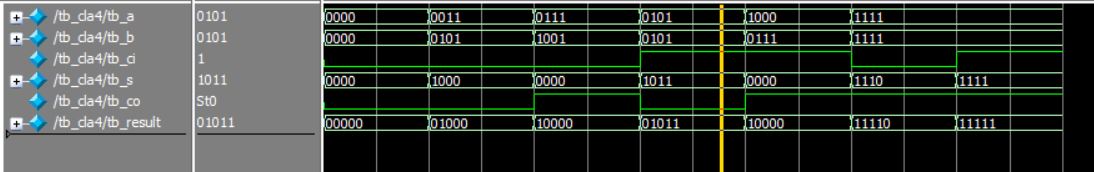
32-bit CLA에 register를 instance 하여 구현하였다.

1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과
2. **4-bits CLA**



**<4-bit CLA 의 TestBench>**

00112 + 01012 의 결과, 01112 + 10012 의 결과, 01012 + 01012 의 결과, 01012 + 01012 의 결과, 10002 + 01112 의 결과, 11112 +11112 의 결과를 확인하는 TestBench 이다.

**<4-bit CLA 의 WaveForm>**

00112 + 01012 의 결과를 확인하면 각각의 bit에서 carry가 발생하는데 10002 가 나온 것으로 보아 정확히 작동했음을 확인할 수 있다.

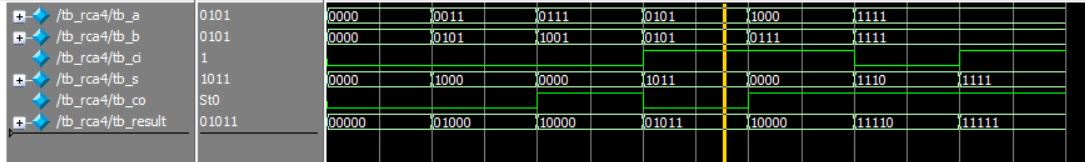
01112 + 10012 : 합이 16인데 4bit 연산 시 나타낼 수 있는 값은 최대 15 이므로 overflow가 발생한다. 따라서 sum 의 값은 0000, carry out 은 1, 5bit로 구성된 result 에선 16이라는 정확한 값이 나옴을 확인할 수 있다.

01012 + 01012 : 처음 carry in 값이 1이므로 최하위 bit와 2bit 위의 bit가 carry 가 발생함을 알 수 있다. 따라서 10112 라는 값이 나온다.

10002 + 01112 : overflow가 발생하지만 5bit로 구성된 result 에선 정확한 값이 출력됨을 확인할 수 있다.

11112 + 11112 : overflow가 발생하지만 result 에서 정확한 값이 출력되는 것으로 보아 carry와 sum 부분이 작동이 잘 되는 것을 확인할 수 있다.

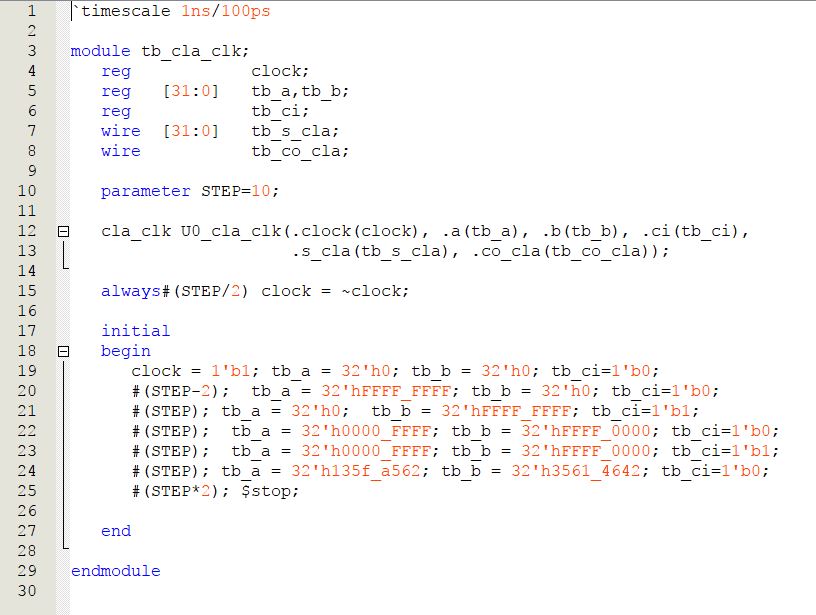
11112 + 11112 : 처음 carry in 값이 1 이므로 sum값이 1, co 값이 1로 출력되어 result는 111112 가 된다.



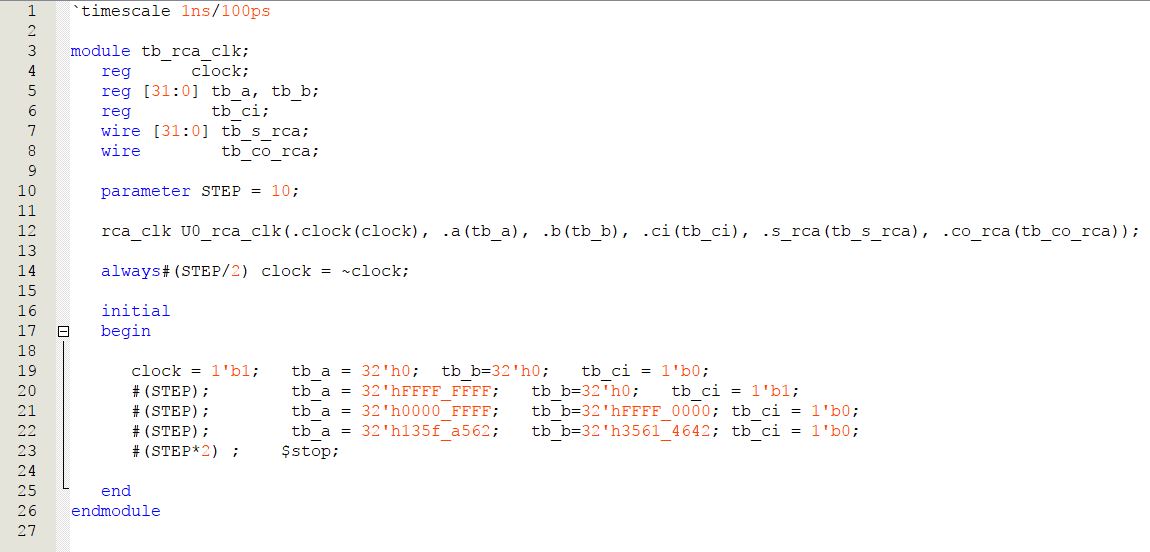
**<4-bit RCA 의 WaveForm>**

위 그림은 4-bit RCA 의 WaveForm 이다. 4-bit CLA 의 WaveForm의 결과와 동일함을 확인할 수 있다.

1. **32-bits CLA with Register / 32-bits RCA with Register**

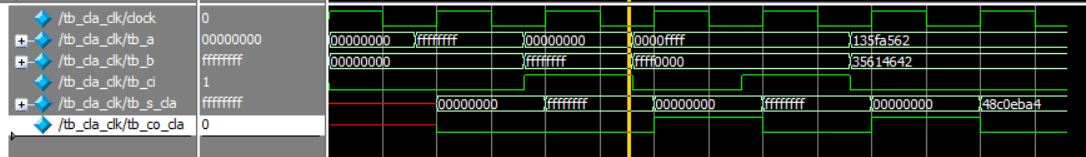


**<32-bits CLA with Register 의 TestBench>**

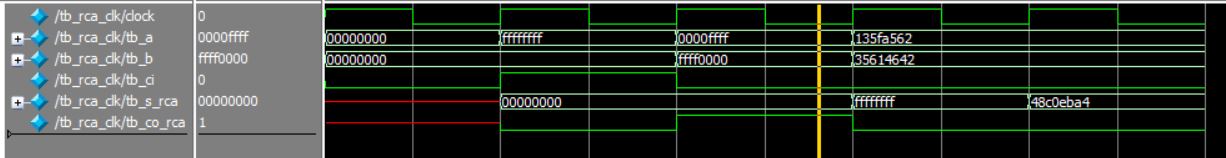


**<32-bits RCA with Register 의 TestBench>**

32-bits CLA with Register 의 TestBench 와 32-bits RCA with Register 의 TestBench 에 같은 값을 넣어주어 어떤 결과가 나오는지 확인하도록 한다.



**<32-bits CLA with Register 의 WaveForm>**

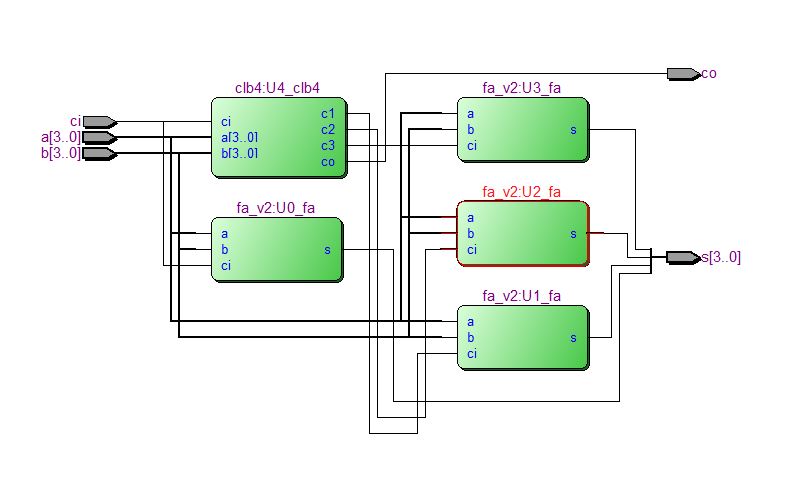


**<32-bits RCA with Register 의 WaveForm>**

32-bits CLA with Register 의 WaveForm 와 32-bits RCA with Register 의 WaveForm 는 같은 형태를 보인다. 정확히 어떤 차이가 있는지 Flow Summary 와 Fmax Summary를 통해 확인해 보도록 한다.

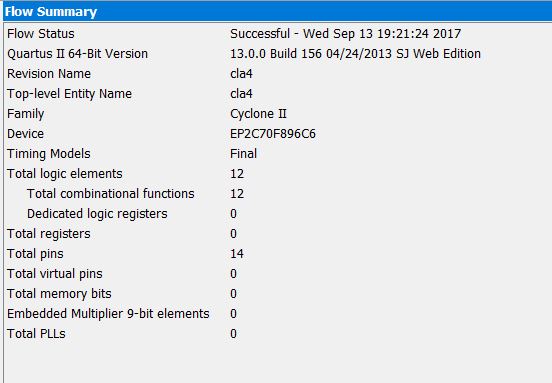
* 1. 합성(synthesis) 결과

1. **4-bits CLA**



**<4-bits CLA 의 RTL Map Viewer >**

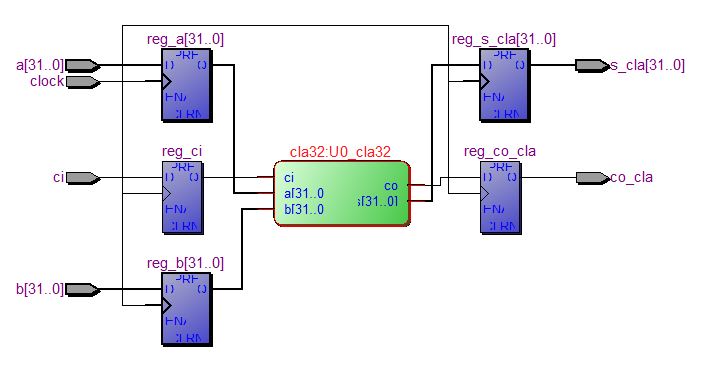
RTL Map Viewer를 통해 4-bits CLA 가 Full Adder 4개로 구현됨을 시각적으로 쉽게 확인할 수 있다.

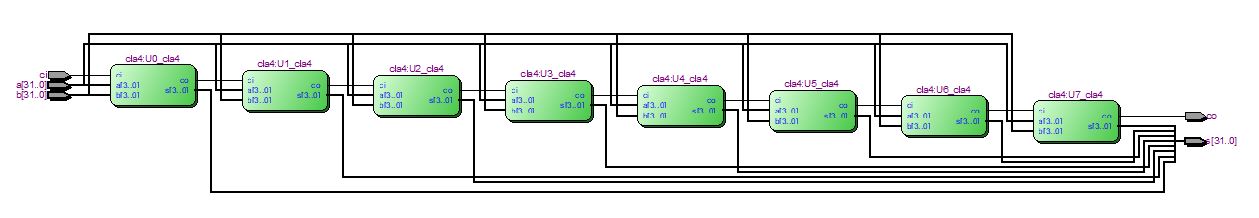


**<4-bits CLA 의 Flow Summary>**

Total logic elements 는 12 Total pins은 14 인 것을 확인할 수 있다.

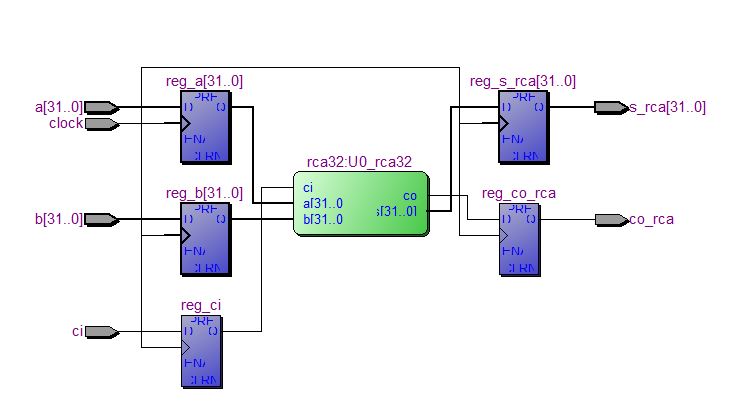
1. **32-bits CLA with Register / 32-bits RCA with Register**

**<32-bits CLA with Register 의 RTL Map Viewer 1>**

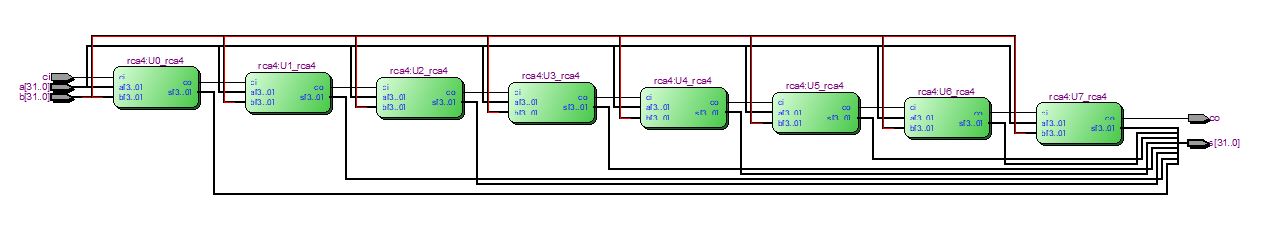
****

**<32-bits CLA with Register 의 RTL Map Viewer 2>**

32-bit CLA 는 4-bits CLA를 8개를 직렬로 연결한 것과 같이 구성되어 있는 것을 확인할 수 있다.

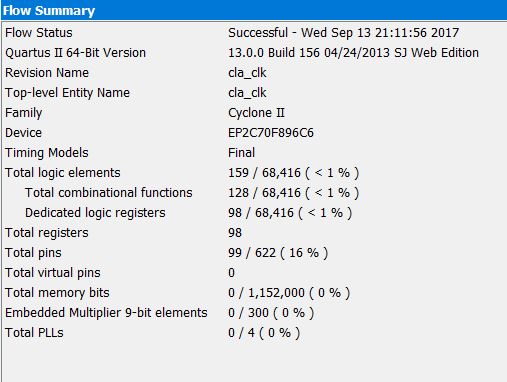


**<32-bits RCA with Register 의 RTL Map Viewer 1>**



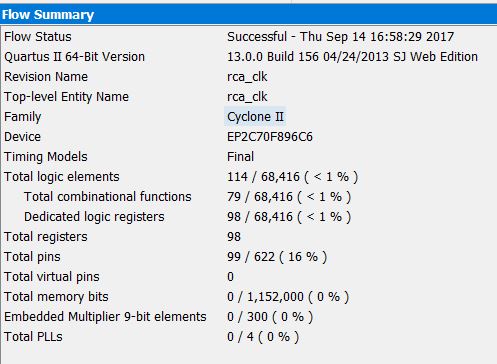
**<32-bits RCA with Register 의 RTL Map Viewer 2>**

32-bit RCA 는 4-bits RCA를 8개를 직렬로 연결한 것과 같이 구성되어 있는 것을 확인할 수 있다.



**<32-bits CLA with Register 의 Flow Summary>**

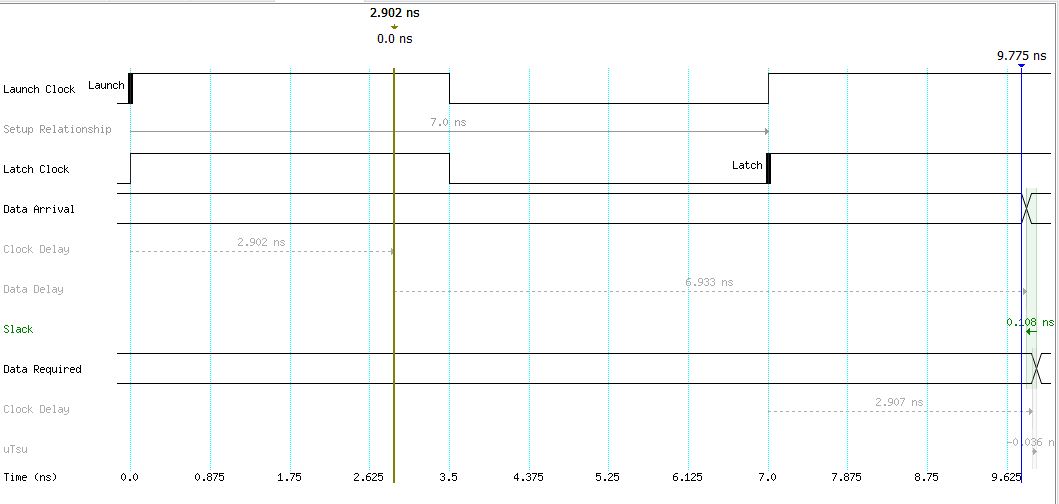
Total logic elements 를 확인하면 159 라는 것을 확인할 수 있다.



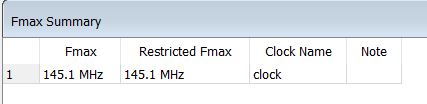
**<32-bits RCA with Register 의 Flow Summary>**

Total logic elements 를 확인하면 114 라는 것을 확인할 수 있다.

32-bits CLA 의 Total logic elements는 159 이고 32-bits RCA 의 Total logic elements 는 114 인 것을 보아 32-bits CLA 의 사이즈가 32-bits RCA 의 사이즈 보다 크다는 것을 확인할 수 있다. 속도 면에서는 어떤 차이가 있는지 WaveForm in Timing Analysis를 확인하도록 한다.

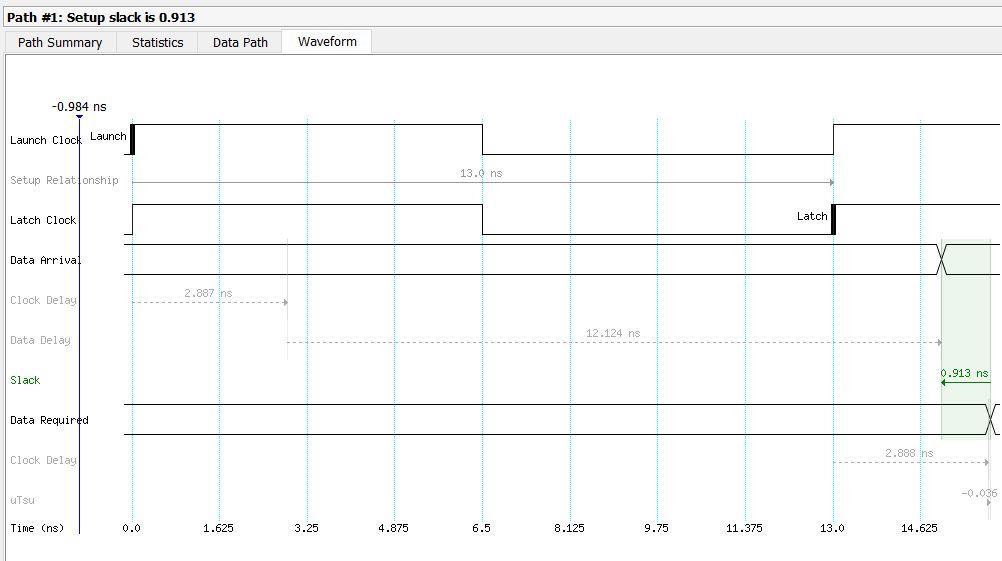


**<32-bits CLA with Register 의 WaveForm in Timing Analysis>**

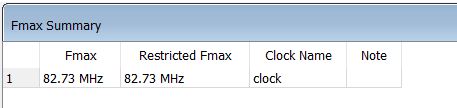
****

**<32-bits CLA with Register 의 Fmax Summary>**

Clock을 7ns로 변경시켜 주면 Slack이 양수로 됨을 확인할 수 있다. 그때 32-bits CLA 의 속도는 145.1 MHz 임을 알 수 있다.



**<32-bits RCA with Register 의 WaveForm in Timing Analysis >**

****

**<32-bits RCA with Register 의 Fmax Summary>**

Clock을 13ns로 변경시켜 주면 Slack이 양수로 됨을 확인할 수 있다. 그때 32-bits RCA 의 속도는 82.73 MHz 임을 알 수 있다.

32-bits CLA 의 속도는 145.1 MHz 이고, 32-bits RCA 의 속도는 82.73 MHz 이다. 따라서 32-bits CLA 가 32-bits RCA 보다 약 1.75 배 더 빠르다는 것을 알 수 있다.

1. 고찰 및 결론
   1. 고찰

Carry out 값을 미리 계산해 주기위해 Generation signal 과 Propagation signal 의 개념을 이해하고 직접 Boolean equation 을 작성하는 것이 힘들었다. 하지만 Carry Look-ahead Adder의 원리를 이해하기 위해 필수적인 과정임을 깨닫고 차근차근 계산을 해보니 끝까지 완수할 수 있었다.

Timing Analysis를 이용해서 확인하는 부분이 익숙하지 않아서 실습 초기에 실수가 많았다. 처음엔 Slack이 음수로 나와 당황했지만 Clock period를 양수로 변경시켜서 Fmax Summary를 통해 속도를 확인할 수 있었다.

* 1. 결론

Carry look-ahead Adder9(CLA) 가 어떻게 구성되어 있는지 명확하게 알 수 있는 실험이었고, 4-bits CLA 를 설계한 후 4-bits CLA 8개를 instance 하여 32-bits CLA 를 설계할 수 있었다. 지난 실습에서 설계했던 4-bits RCA 를 이용해 4-bits RCA 8개를 instance 하여 32-bits RCA를 설계할 수 있었다. 그 후 32-bits CLA 와 32-bits RCA 가 어떤 구성으로 이루어져 있는지 비교하고 Flow Summary 의 total logic elements를 확인하여 CLA의 사이즈가 RCA 의 사이즈보다 크다는 것을 알 수 있었고, Fmax Summary를 확인하여 CLA의 속도가 RCA의 속도보다 빠르다는 것을 알 수 있었다.

즉, RCA 는 계산 속도는 느리지만 크기가 작고, CLA 는 계산 속도는 빠르지만 크기가 크다는 특성을 비교할 수 있었다.

실제로 bit 수가 적을 때(16bits 미만) 는 Carry 값을 미리 계산하는 CLA가 RCA 보다 느리므로 16bits 미만의 연산에서는 구현이 간단하고 사이즈가 작은 Ripple Carry Adder를 사용하고, 16bits 이상의 연산에선 Carry Look-ahead Adder 를 사용하는 것이 효율적이다.

1. 참고문헌

공진흥 / 컴퓨터공학기초실험2 / 새빛관303호(광운대학교) / 2017년.

이준환 / 디지털논리회로2 / 참빛관B101호(광운대학교) / 2017년.