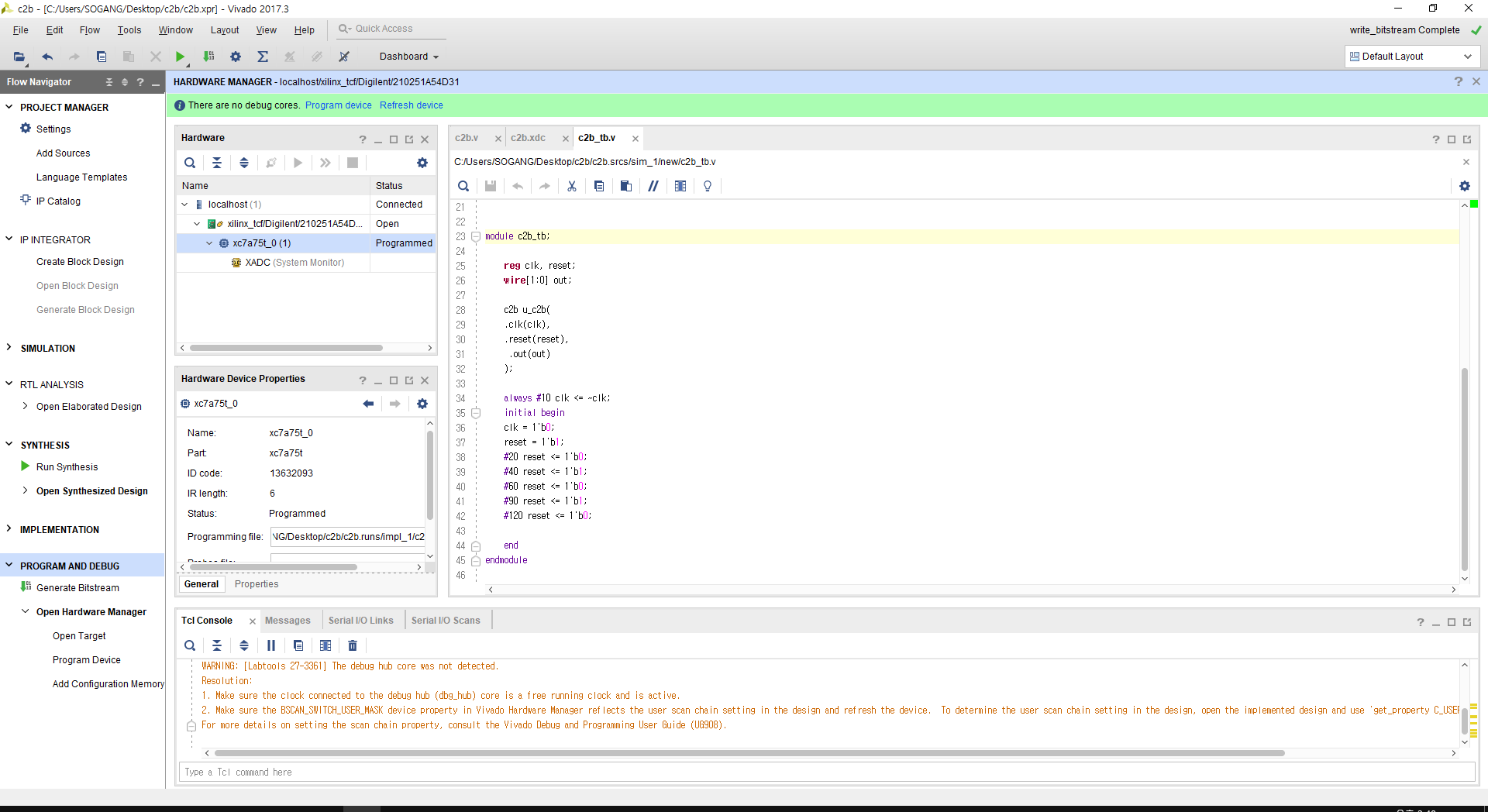
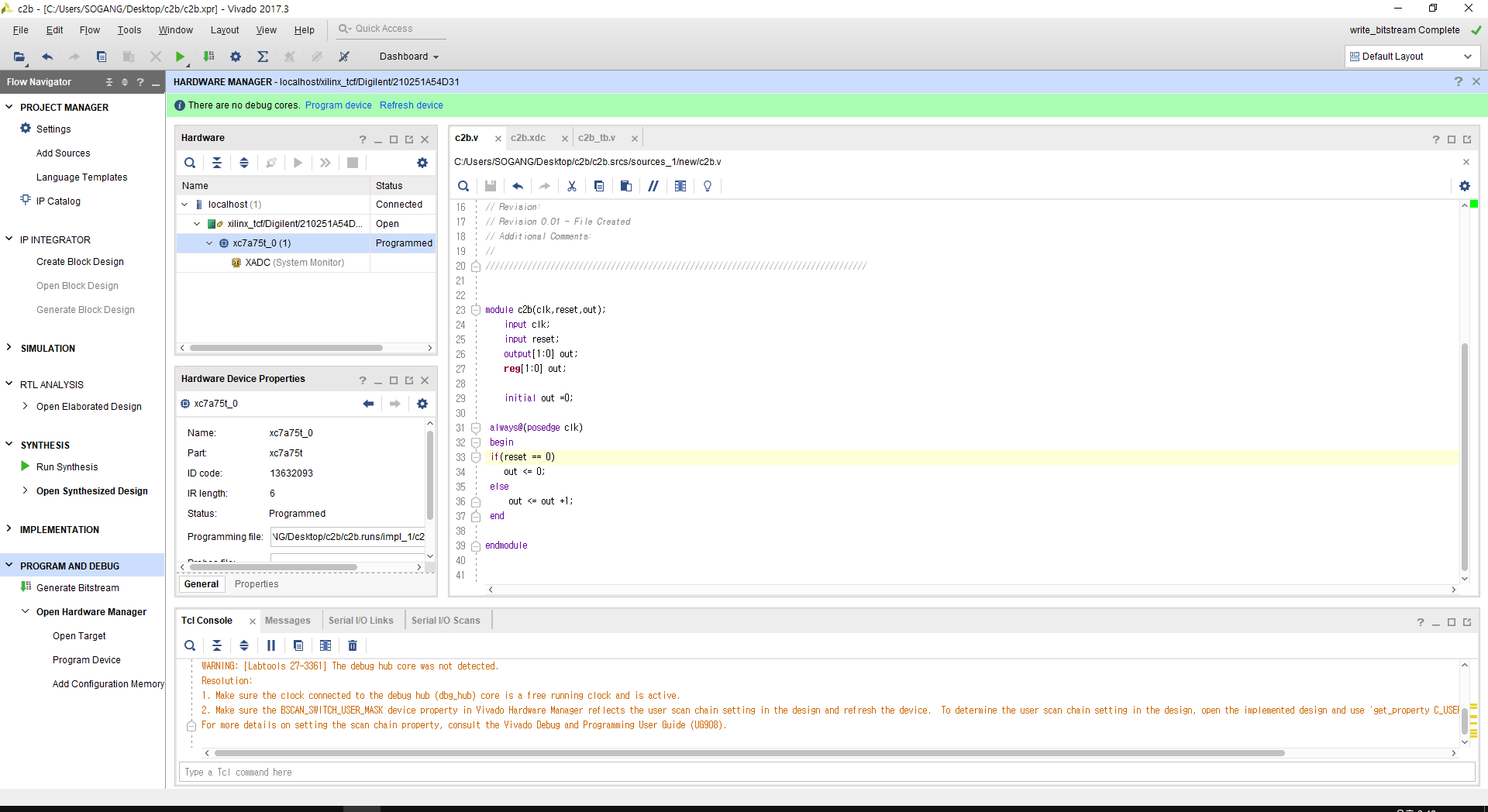
12주차 결과보고서

전공: 아트엔테크놀로지 학년: 3학년 학번: 20191172 이름: 함승우

1. 2-bit counter 의 결과 및 Simulation 과정

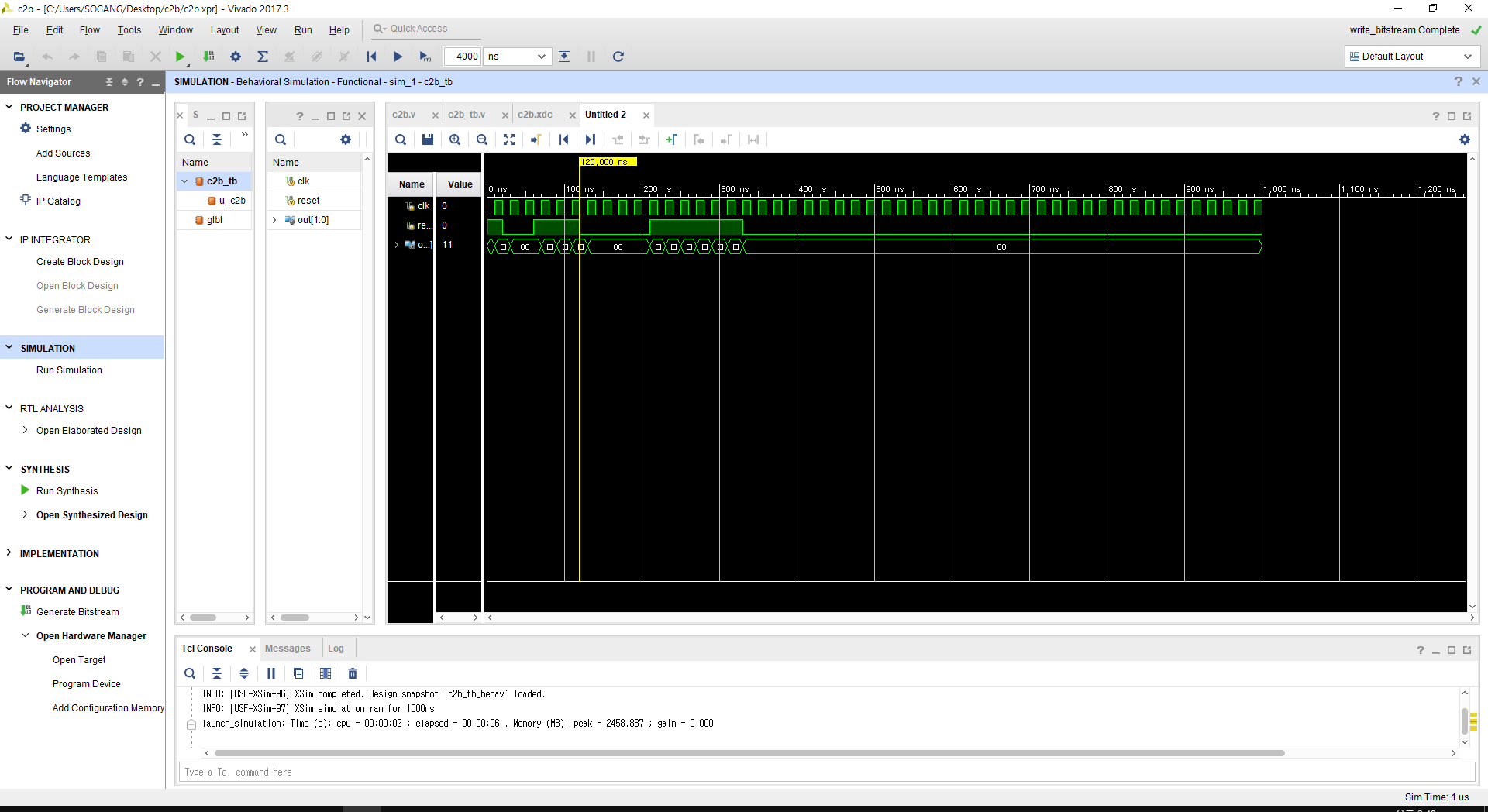
2 bit counter는 출력이 00, 01, 10, 11로 나타나는 계수기이다. 본 실습에서는 reset을 누른 상태에서, 즉 reset의 입력이 1일 때, clock이 변화할 때마다 계수기의 output이 1씩 증가하는 것으로 설계했다. 이때 reset 버튼에 손을 떼게 되면, 즉 reset의 입력이 0이 되었을 때, clock이 변화하면 계수기의 output이 00으로 초기화된다.

* Verilog Coding

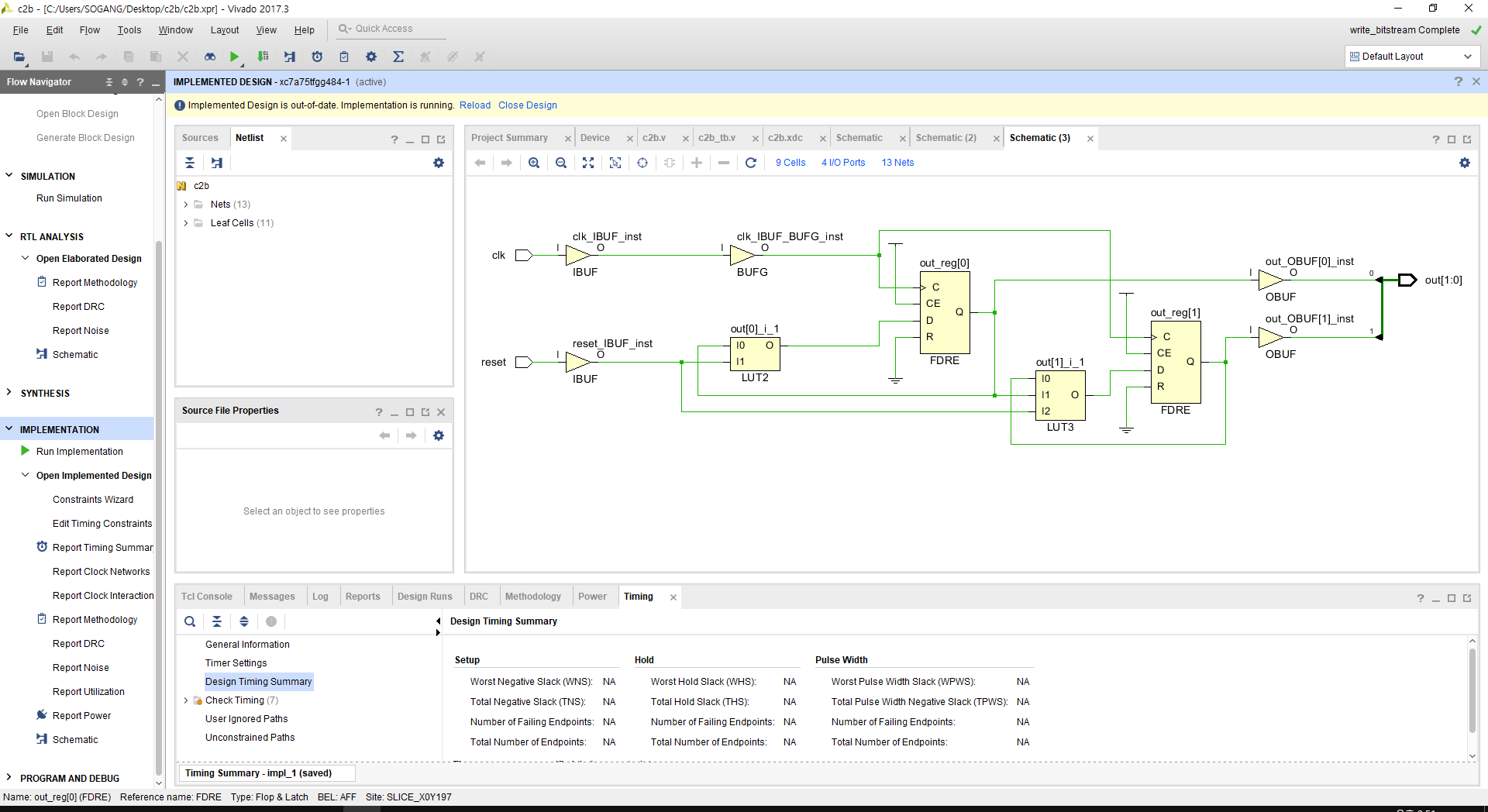


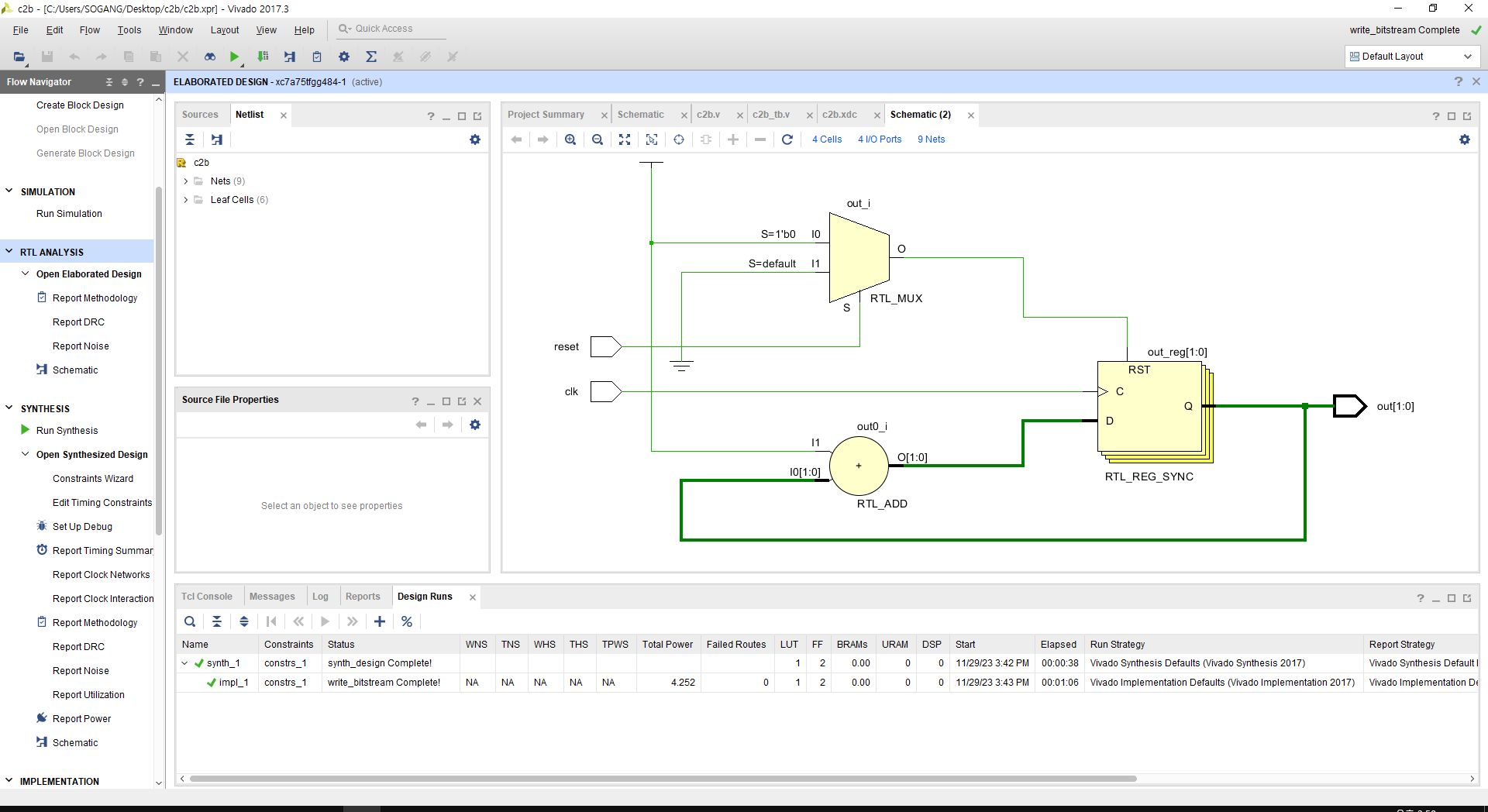
이 Verilog 코드에서 reset이 0일 경우 output이 00으로 출력되도록 설계하였다. 또한, reset이 0이 아닌 1일 경우, output이 하나씩 더하는 식으로 설계하였다. tb파일에서는 reset의 1의 설정, 0의 설정을 반복하여 나타나게 설계하였다.

* Simulation 과정 및 Schematic



output을 배열로 설정했기에 위 그림과 같은 형태로 나타나고, reset이 1일 때, 계속하여, clock이 변화할 때마다 +1 되는 것을 알 수 있다. 예를 들어, 120인 경우에, tb 파일에서 reset의 값이 0으로 변환된다는 것을 알 수 있다. 이때 simulation결과를 잘 보면, 120부터 output의 값이 00으로 되는 것을 알 수 있다. 또한, 그 다음 reset이 1이 되었을 때, clock이 변화할 때마다, output의 값이 변화하고, 1씩 증가하는 것을 확인할 수 있다.



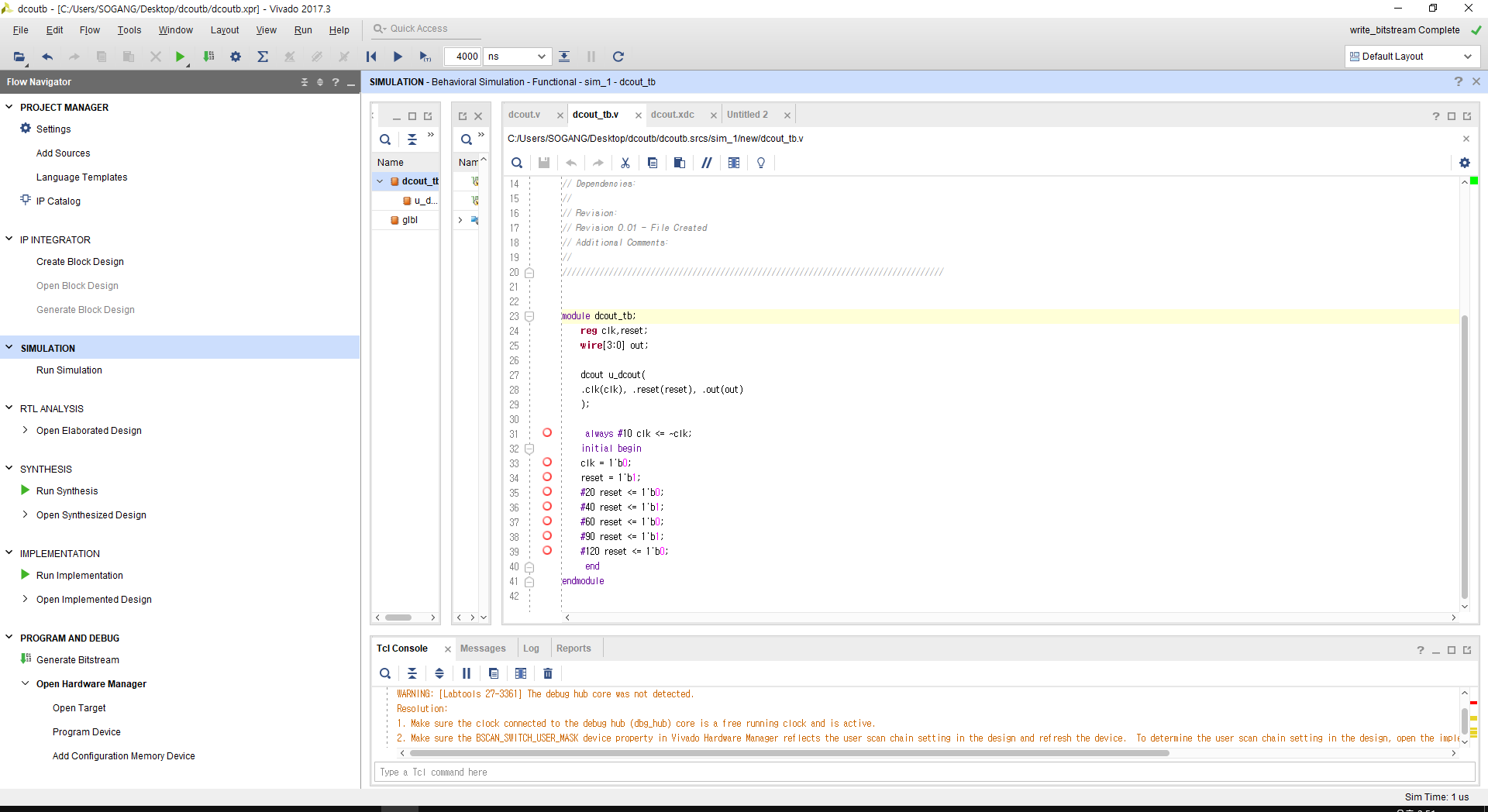
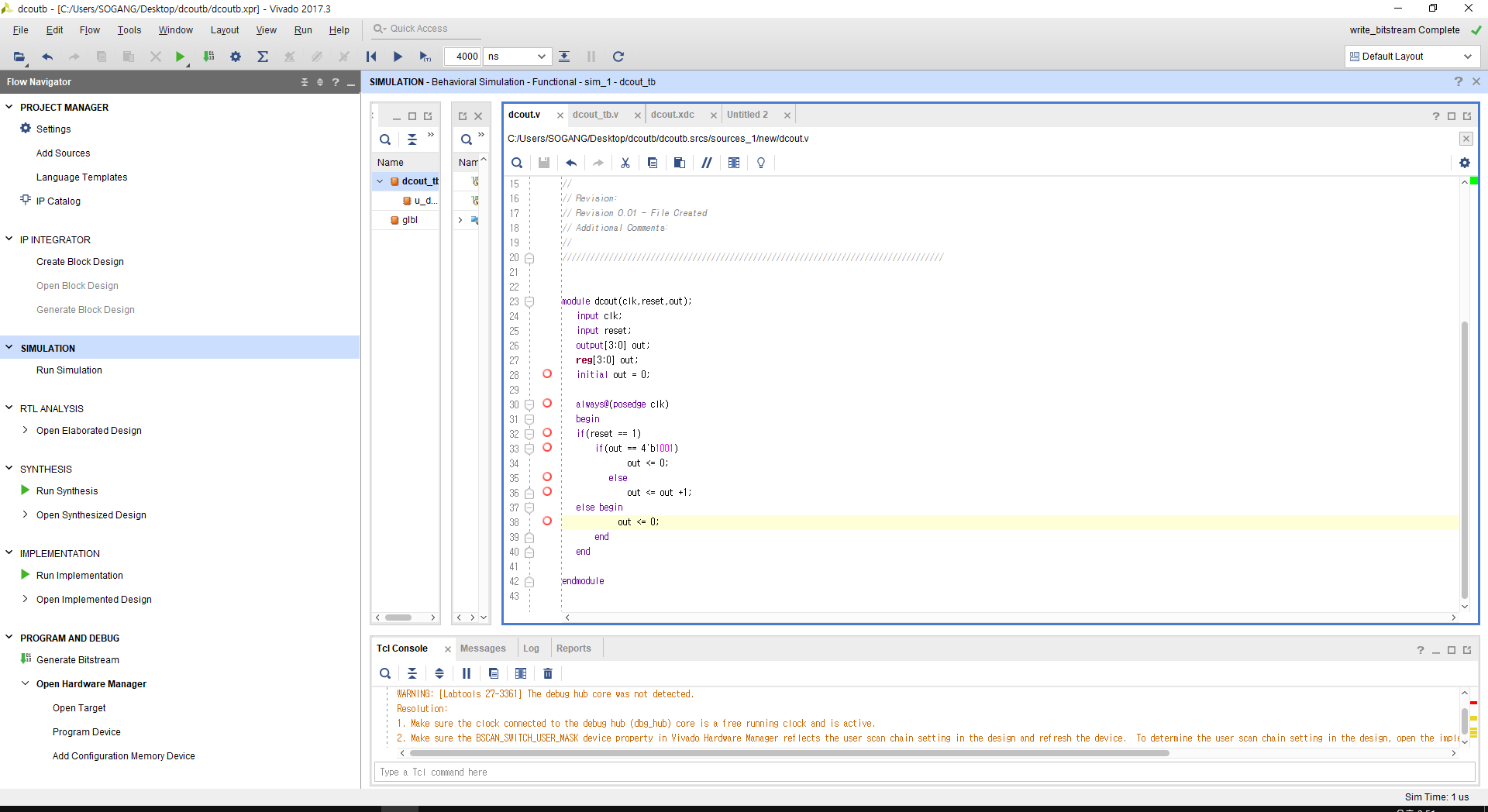


2. 4-bit decade counter의 결과 및 Simulation 과정에 대해서 설명하시오.

(verilog source code, 출력 예시/schematic 포함, 과정 상세히 적을 것)

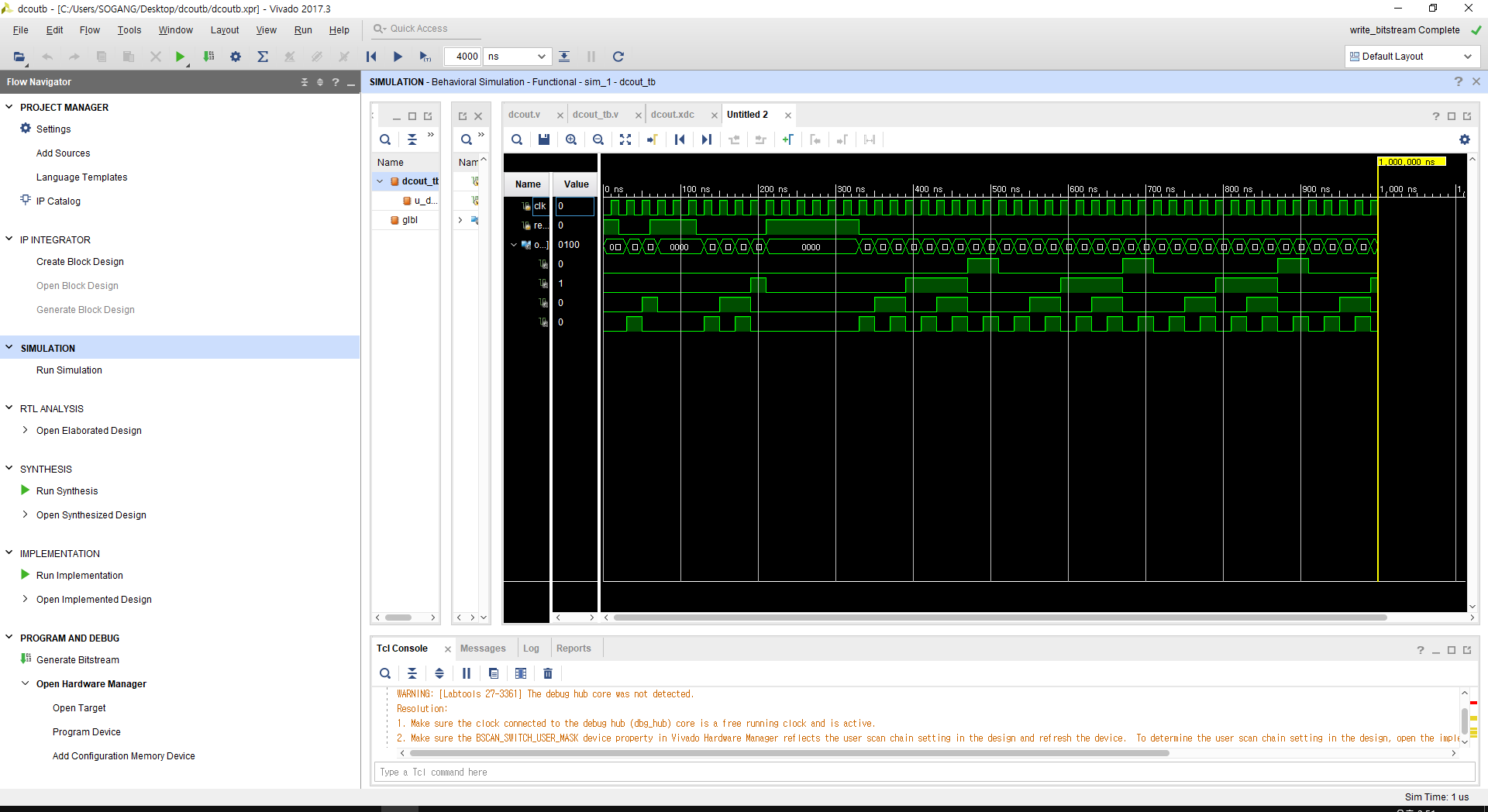
4- bit decade counter는 Binary Code로 4비트의 수를 계수하는 Counter이다. 본 실습에서는 위와 마찬가지로 reset 버튼을 누르고, clock의 변화가 있을 때마다 output의 값이 1씩 증가했다. output은 0~9까지 표현하므로, 이 값은 4 bit로 0000 ~ 1001까지 출력한다.

* Verilog coding

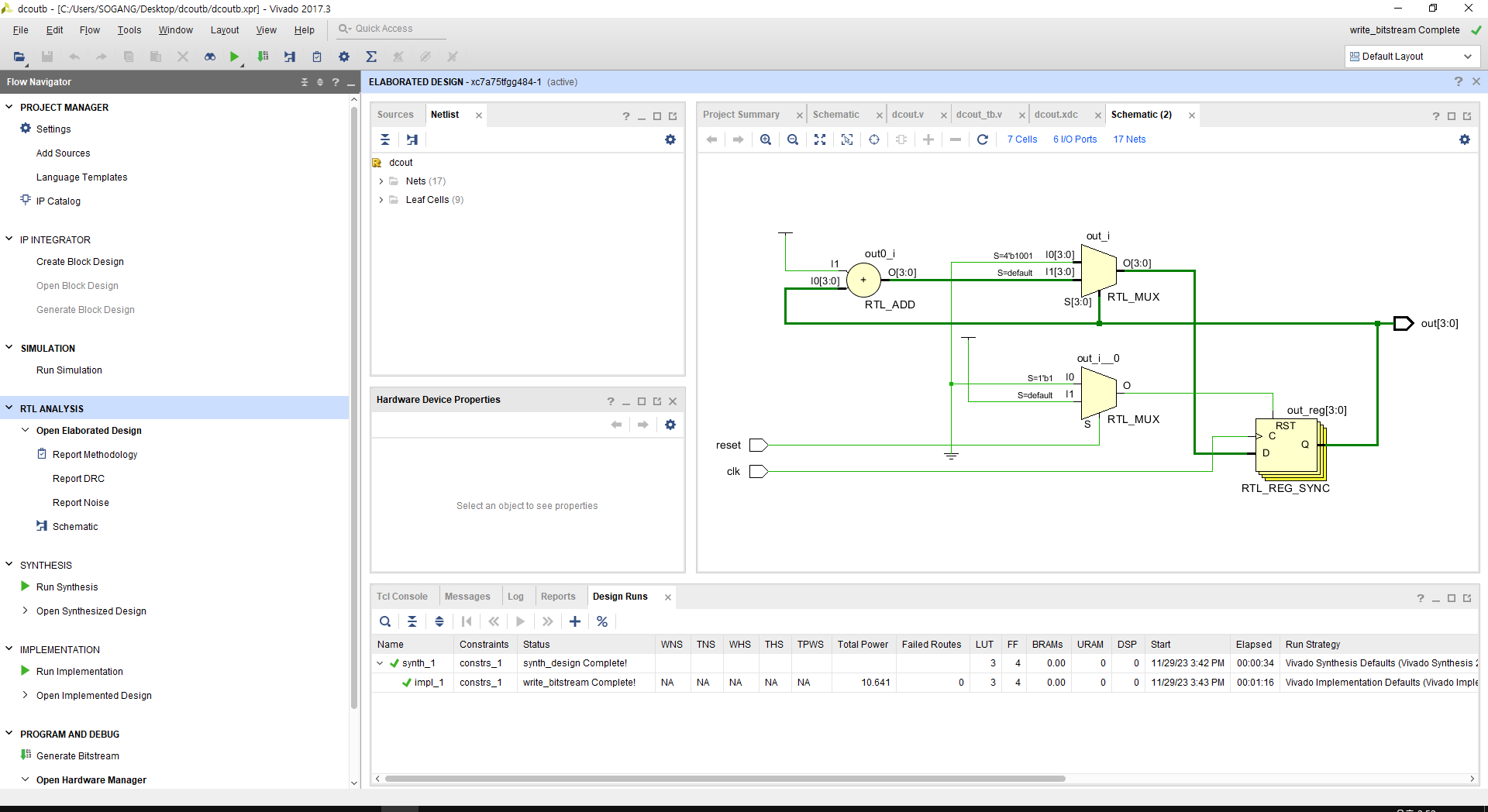
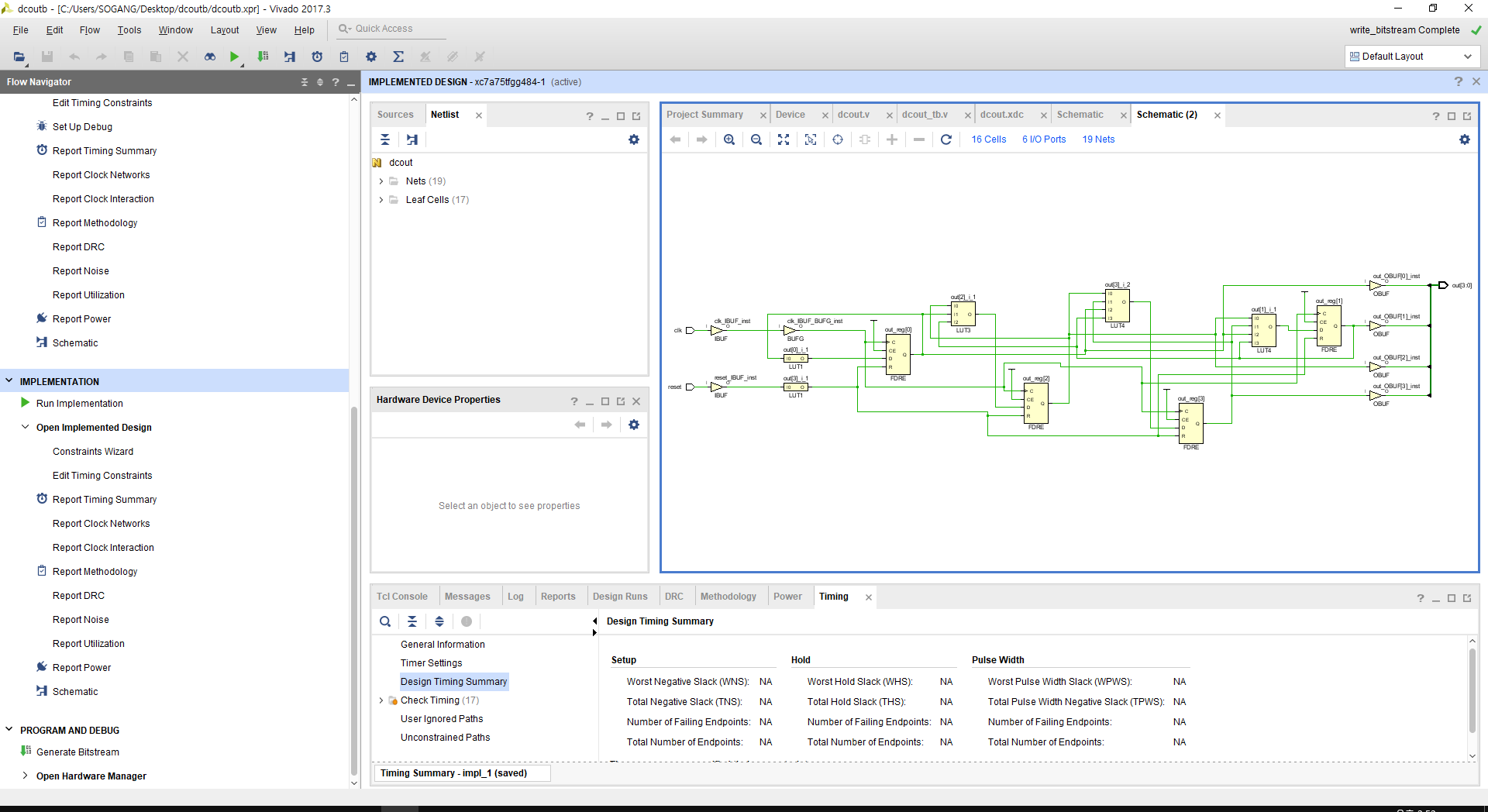


위와 비슷하게 reset이 1인 경우에 output이 +1이 되고, reset이 0인 경우에 output은 0이 된다. 하지만, 위 이진 카운터와는 다르게 output은 4 bit기 때문에 4’b0000 이런 형태로 초기화를 하고, 표현할 수 있는 최대치인 1001에 도달했을 때 output을 다시 0으로 초기화하는 것에 있어서 차이가 있다.

* Simulation 과정 및 Schematic



위 이진 카운터가 동일하게, reset에 결과들이 영향을 받고, clock이 10 단위로 변화하는데, 이 변화에 맞춰서 reset이 조건을 만족한다면, 계속 +1이 되는 것을 확인할 수 있다. 또한, 500ns 부근을 보면, output의 값이 1001인데 이는 Binary code로 표현할 수 있는 최대치이다. 따라서 다음 clock의 변화 때, output의 값은 0이 된다. 이 코드의 Schematic는 이렇다.



3. 4-bit 2421 decade counter 의 결과 및 Simulation 과정에 대해서 설명하시오.

(verilog source code, 출력 예시/schematic 포함, 과정 상세히 적을것)

4 – bit 2421 decade counter는 앞서 진행했던 8421 Binary code decade counter와 유사하다. 동작하는 방식은 동일하나, 0부터 9까지 표현하는 방식에 있어서 다르다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8421 code | | | | | 2421 code | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 5 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 6 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 7 | 1 | 1 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 8 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 | 9 | 1 | 1 | 1 | 1 |

2421 code는 4와 5 사이를 기준으로 서로 1의 보수 관계이다. 즉, 4와 5, 3과 6, 2와 7 이런 식으로 보수의 형태를 띄게 된다.

- Verilog 코드

코드를 짠 아이디어는 우선, 8421 code로 작성을 한다. 그 후 2421 code로 변환하는 것이다.

8421의 code는 위 2번 실습과 동일하고, 위 2번 실습을 통해서 나온 output을 차례대로 bout[3], bout[2], bout1[1], bout[0]을 통해서 2421 code를 설계했다.

1. out[0]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | X | X | X | X |
| 10 | 0 | 1 | X | X |

cout[0] = bout[0]

1. out[1]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | X | X | X | X |
| 10 | 1 | 1 | X | X |

bout[3] + bout[2]’bout[1] + bout’[1]bout[0]bout[2]

1. out[2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 1 | 1 |
| 11 | X | X | X | X |
| 10 | 1 | 1 | X | X |

bout[3] + bout[2]bout[1] + bout[2]bout[0]’

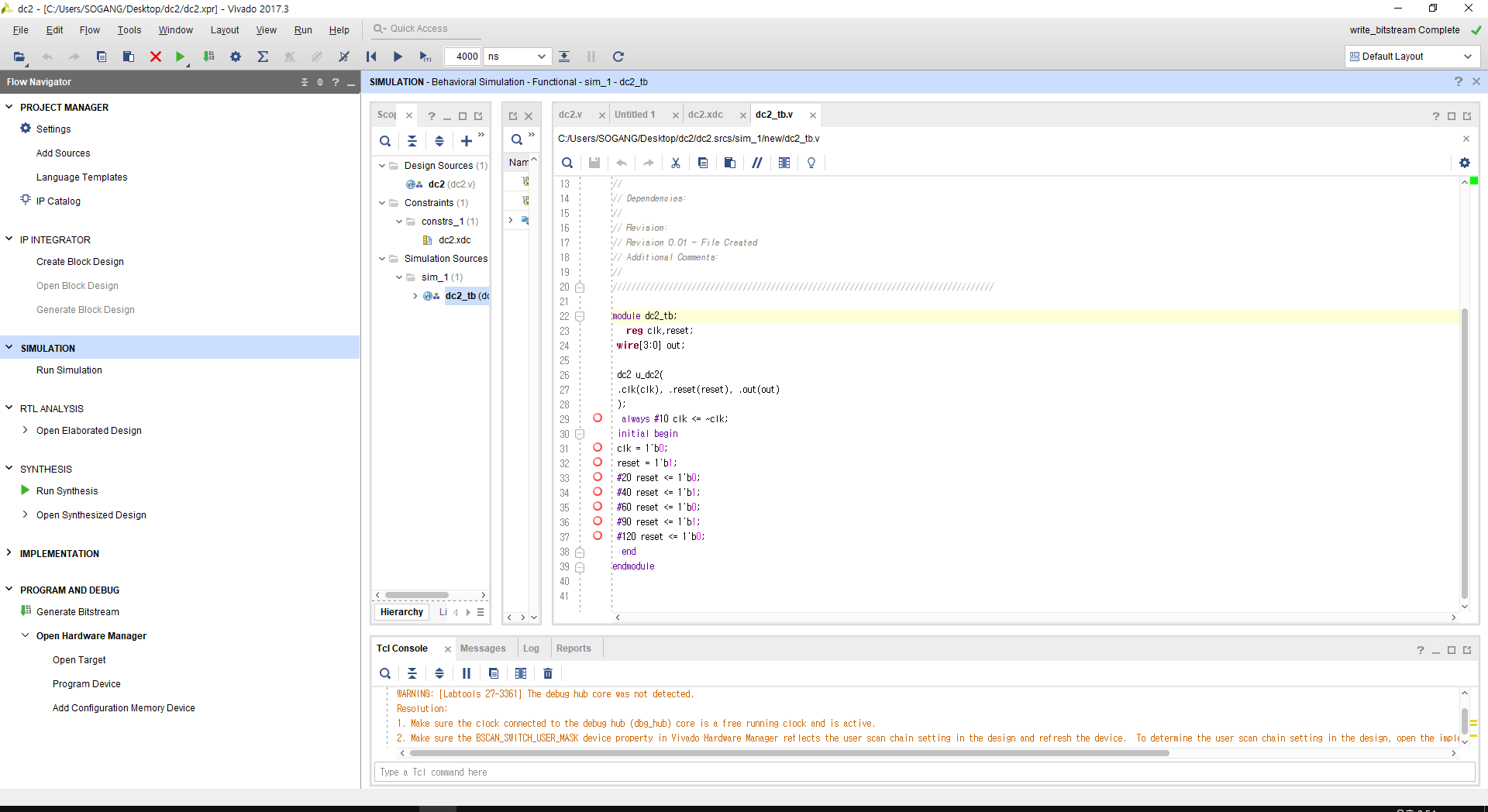
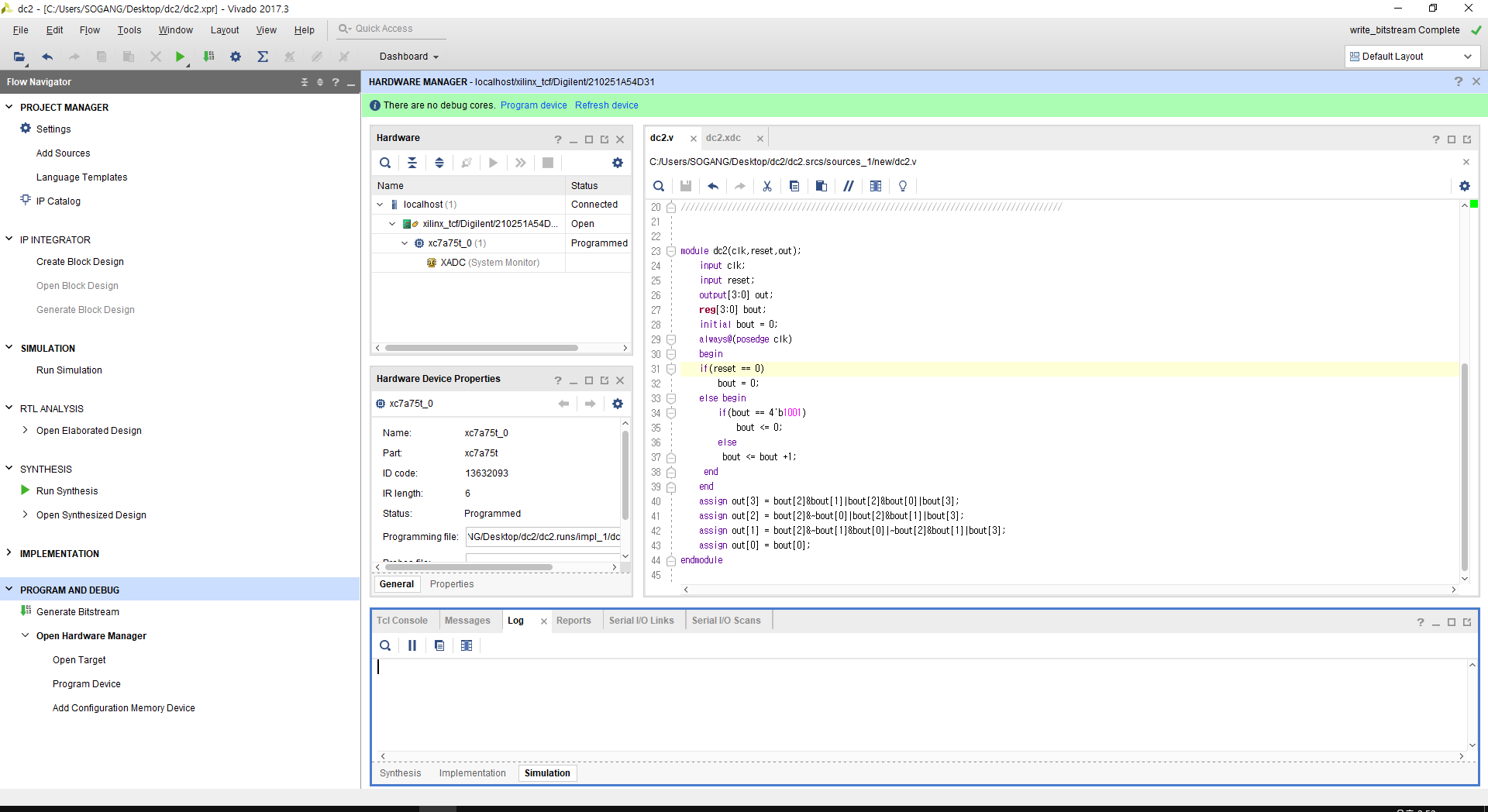
1. out[3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | X | X | X | X |
| 10 | 1 | 1 | X | X |

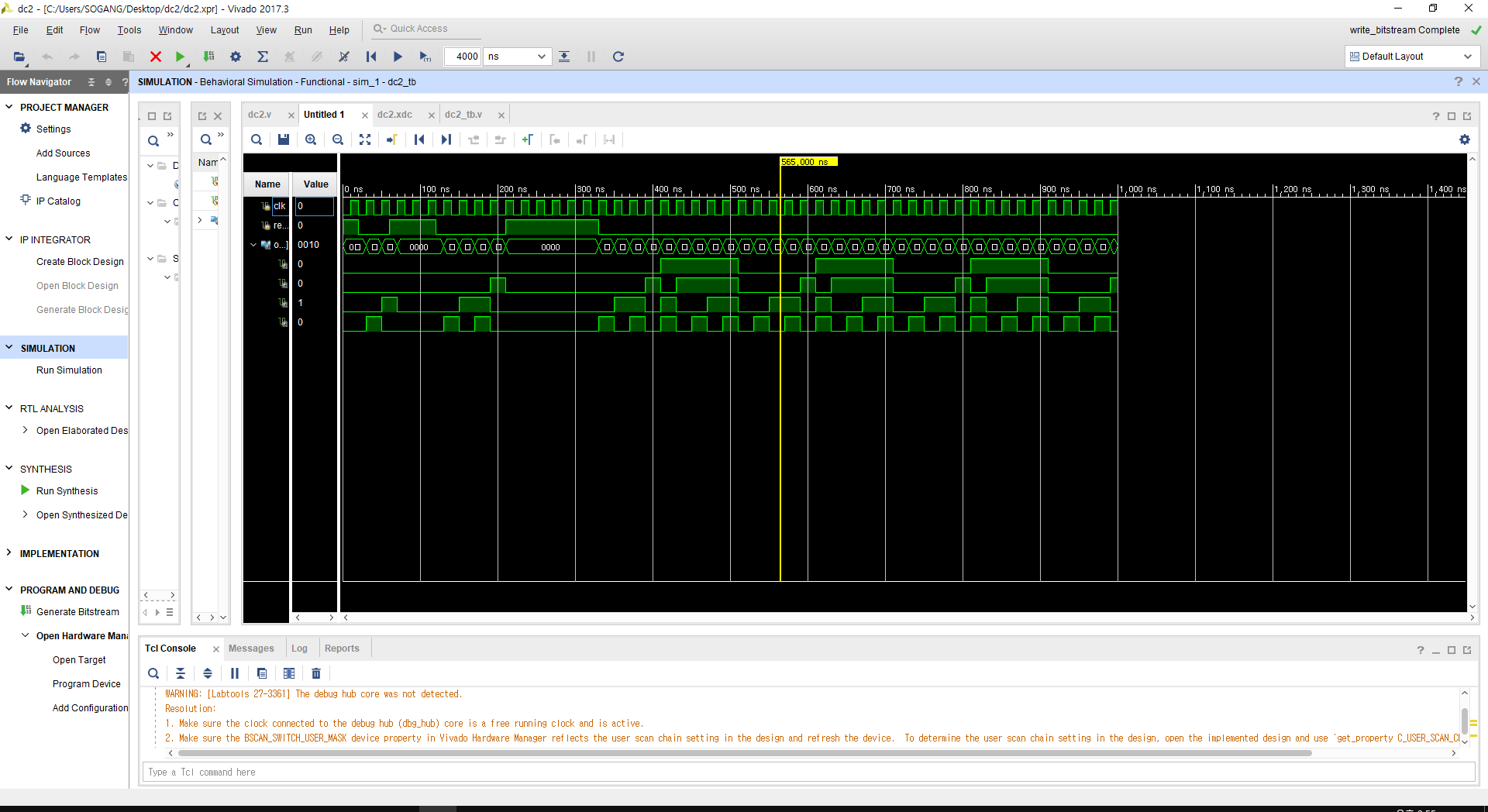
bout[3] + bout[2]bout[0] + bout[2]bout[1]

따라서 Verilog coding은 이렇다.

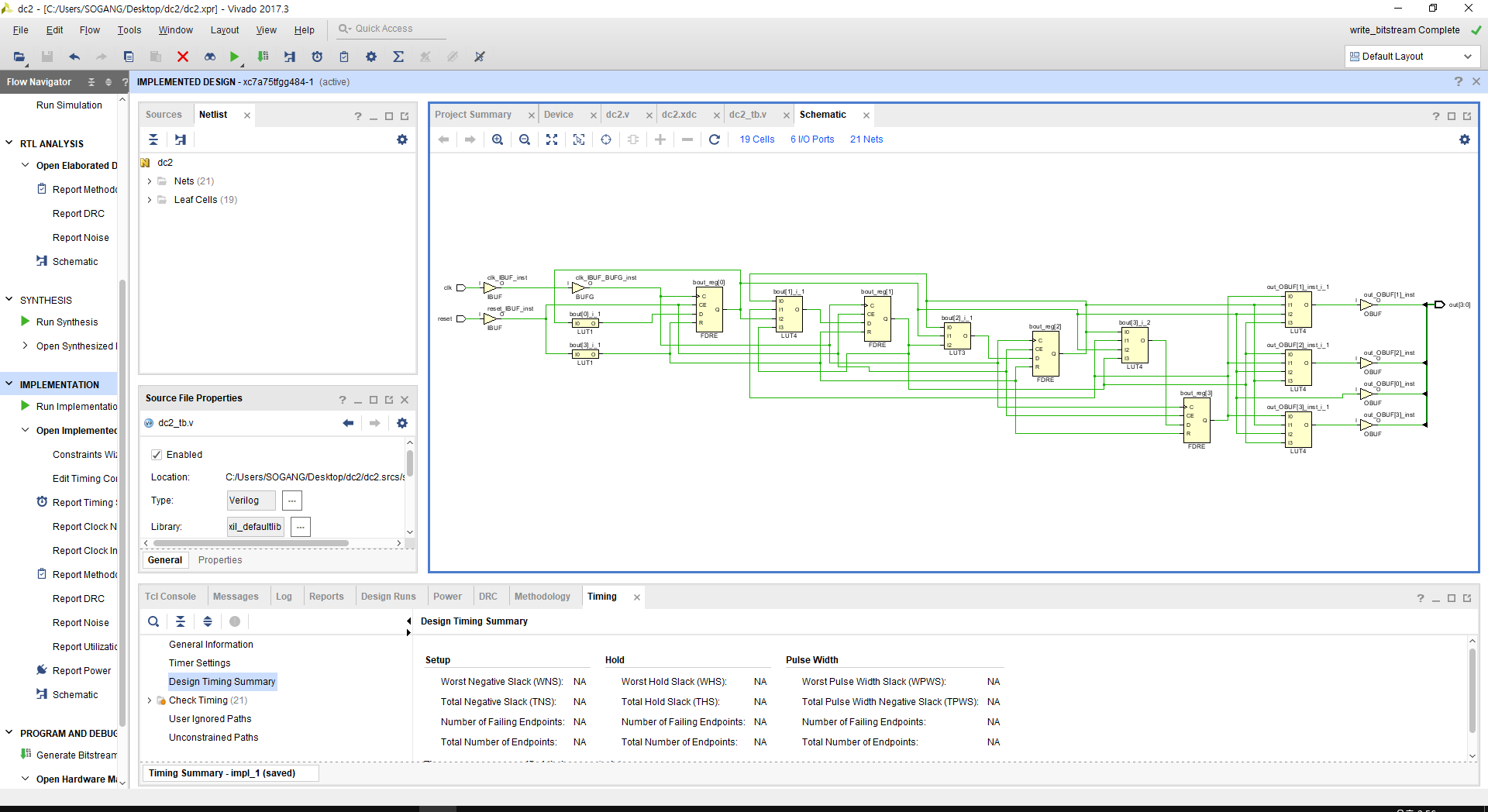
* Verilog code

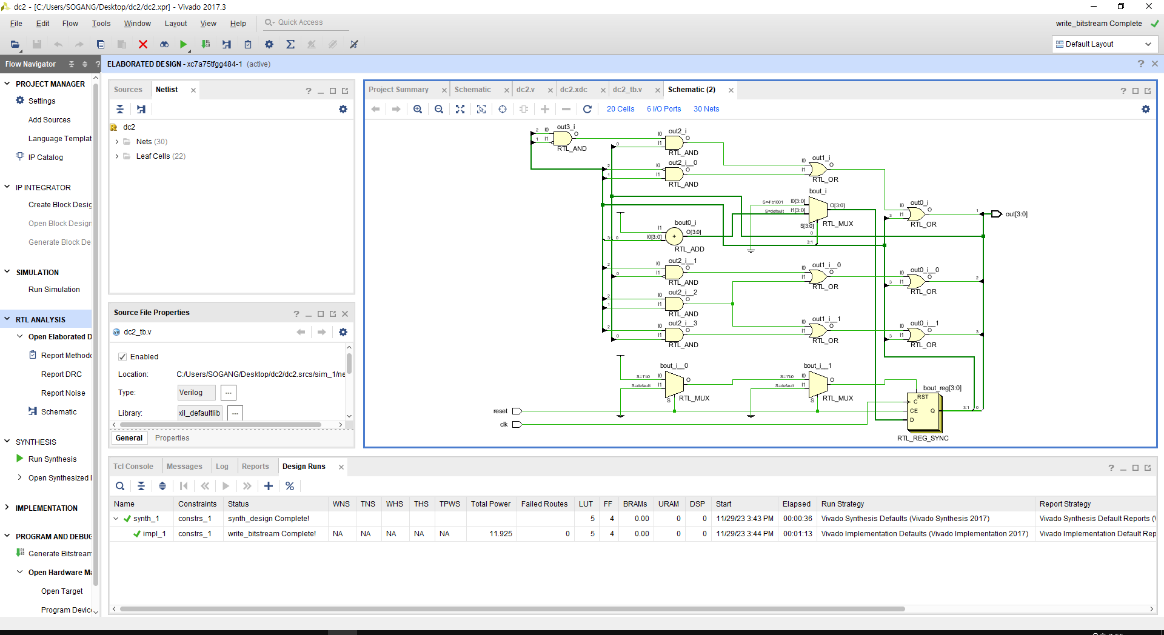


* Simulation 과정 및 Schematic



위, 8421 code와 동일하게 작동하지만, 2421 code라는 점에서 살짝 다르다. reset이 초기화되고, output이 차례대로 출력되는 순서대로 보자면, 0000, 0010, 0011, 0100 후에 output[3]부분이 1로 출력 되면서 1011이 출력된다. 그후는, 위 작성한 2421 code와 동일하게 출력된다.





4. 결과 검토 및 논의 사항.

output을 array 형식으로 받고 출력했다. reset에서 손을 뗄 때 계수기에 나타난 output을 0으로 만들었고, 그렇지 않는다면 clock이 변화할 때 마다 output을 하나씩 증가시켰다. 2진 카운터와 decade counter는 reset == 1일 때 output의 배열에 1씩 더해주는 것으로 구현했고, 2421 counter도 원리는 비슷하나. 8421 code와 0~9까지 표현함에 있어서 다르기 때문에, 8421 counter로 우선 구현하고, decade counter의 4 bit 결과 값에 따라서 카르노 맵을 그려서 2421 counter를 구현하였다.

1. 추가 이론 조사 및 작성.

본 실습에서 2421 counter를 작성했지만, 다른 Binary code 중 5211 code를 작성해보자. 우선, 앞선 실습 3에서 작성했듯이 8421 code로 count를 한 다음, 5211 code로 변환하는 과정만 필요하다고 가정하자.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8421 code | | | | | 5211 code | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 4 | 0 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 5 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 6 | 1 | 0 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 7 | 1 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 8 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 | 9 | 1 | 1 | 1 | 1 |

이에 따라 카르노 맵을 작성하면,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 | | 00 | 0 | 1 | 1 | 1 | | 01 | 1 | 0 | 0 | 0 | | 11 | X | X | X | X | | 10 | 0 | 1 | X | X | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 | | 00 | 0 | 0 | 0 | 1 | | 01 | 1 | 0 | 0 | 1 | | 11 | X | X | X | X | | 10 | 1 | 1 | X | X | |
| out[0] =bout[0]’bout[1]’bout[2] + bout[0]bout[2]’ + bout[1]bout[2]’ | out[1] = bout[1]bout[0]’ + bout[3] + bout[1]’bout[0]’bout[2] |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 | | 00 | 0 | 0 | 1 | 0 | | 01 | 1 | 0 | 1 | 0 | | 11 | X | X | X | X | | 10 | 1 | 1 | X | X | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | bout[1]bout[0]  bout[3]bout[2] | 00 | 01 | 11 | 10 | | 00 | 0 | 0 | 0 | 0 | | 01 | 0 | 1 | 1 | 1 | | 11 | X | X | X | X | | 10 | 1 | 1 | X | X | |
| out[2] = bout[3] + bout[1]’bout[0]’bout[2] + bout[1]bout[0] | out[3] = bout[3] + bout[0]bout[2] + bout[1]bout[2] |