디지털 논리2 및

컴퓨터 공학 기초 실험2   
보고서

과제제목: FIFO

실험일자: 2017년 10월 13일 (금)

제출일자: 2017년 10월 26일 (목)

학 과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 금 5, 6, 7

학 번: 2012722028

성 명: 장 한 별

1. 제목 및 목적
   1. 제목

FIFO (First In First Out)

* 1. 목적

이전 실습 때 구현한 register file을 사용하여 READ와 WRITE기능을 하는 FIFO를 구현하고 이해하도록 한다. FIFO를 설계하는 데에 있어서 FSM의 State diagram을 이해하고 구현할 수 있도록 하고, 이를 Verilog로 설계할 수 있도록 한다. 또한 이를 Testbench를 통해 잘 구현이 되는지 확인하며 다시 한번 이해하는 데에 목적이 있다.

1. 원리(배경지식)
2. Queue

|  |
| --- |
| <Queue> |
|  |
| 위 그림은 queue 자료구조를 그림으로 나타낸 그림이다. Queue의 특징은 FIFO(First In First Out) 즉, 먼저 들어온 데이터가 먼저 나가는 구조이다. 따라서 1,2,3,4,5 이렇게 데이터가 들어왔으면 빠져나갈 때 1,2,3,4,5의 순서대로 빠져 나가게 된다. |

1. State

|  |
| --- |
| 1. READ: dequeue 가 잘 된 상태  2. WRITE: enqueue 가 잘 된 상태  3. RD\_ERROR: dequeue가 잘 되지 않은 상태 (empty일떄 read를 한 것)  4. WR\_ERROR: enqueue가 잘 되지 않은 상태 (full 일때 enqueue를 한 것)  5. IDLE : READ나 WRITE를 하지않고 아무런 operation을 수행하지 않은 상태 |

1. 설계 세부사항

< FIFO>

|  |
| --- |
| <State diagram> |
|  |
| 위 그림은 fifo의 state diagram 이다. 총 5개의 state에 각 신호에 따라 state가 변하는 것을 확인할 수 있다. 이때 각 화살표에 써져있는 nop 은 wr\_en이 0 rd\_en이 0일때이고, wr\_con은 wr\_en이 1, rd\_en이 0, data\_count 가 8보다 작을 때의 신호이다. 또한 full\_con은 wr\_en이 1, rd\_en이 0, data\_count가 8일때이고, rd\_con은 wr\_en이 0, rd\_en 이 1, data\_count가 0보다클때이고, 마지막으로 empty\_con은 wr\_en이 0, rd\_en이 1, data\_count 이 0일때이다. |

|  |  |
| --- | --- |
| State encoding | |
| IDLE | 000 |
| WRITE | 001 |
| READ | 010 |
| WR\_ERROR | 011 |
| RD\_ERROR | 100 |
| 위 표는 각 state를 binary encoding을 통해 해당하는 값으로 설정한 내용이다. | |

|  |  |
| --- | --- |
| <State operation> | |
| State | Operations |
| IDLE | No change |
| WRITE | mem[tail]=din, tail++, data\_count++, we=1 |
| READ | dout=mem[head], head++, data\_count--, re=1 |
| WR\_ERROR | No change |
| RD\_ERROR | No change |
| 위 표는 각 state일 때의 작동을 설명한 내용이다. IDLE, WR\_ERROR, RD\_ERROR일때는 어떠한 작동도 하지않고, WRITE일때는 din을 tail에 넣어주고, data\_count를 1 늘려주고, write enable 신호를 1로 set해준다. 또한 READ일때는 출력으로 head값을 출력해주고, head를 1 늘려주고, data\_count를 1 빼주고, read enable 신호를 1로 set해준다. | |

|  |  |  |
| --- | --- | --- |
| <Output design> | | |
|  | FULL | EMPTY |
| Data\_count == 0 | 0 | 1 |
| Data\_count == 8 | 1 | 0 |
| others | 0 | 0 |
| 위 표는 full과 empty를 design한 내용이다. Data count가 0일때는 empty가 1로 set되고, data\_count가 8일때는 full이 1로 set된다. | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| <Output design> | | | | | |
|  | IDLE | WRITE | READ | WR\_ERROR | RD\_ERROR |
| WR\_ACK | 0 | 1 | 0 | 0 | 0 |
| WR\_ERR | 0 | 0 | 0 | 1 | 0 |
| RD\_ACK | 0 | 0 | 1 | 0 | 0 |
| RD\_ERR | 0 | 0 | 0 | 0 | 1 |
| 위 표는 각 state에서 flag의 값들의 변화를 설정해준 표이다. IDLE일때는 모든 flag가 0이고, WRITE일때는 WR\_ACK 만 1이고 READ일때는 RD\_ACK만 1, WR\_ERROR일때는 WR\_ERR만 1, RD\_ERROR일때는 RD\_ERR만 1이 되도록 설정하였다. | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| <I/O Description> | | | |
| Port | Name | Bandwidth | Description |
| Input | clk | 1-bit | Clock |
| reset\_n | 1-bit | Reset |
| rd\_en | 1-bit | Read enable |
| wr\_en | 1-bit | Write enable |
| din | 32-bit | Data in |
| Output | dout | 32-bit | Data out |
| full | 1-bit | Full flag |
| empty | 1-bit | Empty flag |
| wr\_ack | 1-bit | Write acknowledge flag |
| wr\_err | 1-bit | Write error flag |
| rd\_ack | 1-bit | Read acknowledge flag |
| rd\_err | 1-bit | Read error flag |
| data\_count | 4-bit | Count of data |
| Wire | head | 3-bit | Head address |
| next\_head | 3-bit | Next head address |
| tail | 3-bit | Tail address |
| next\_tail | 3-bit | Next tail address |
| state | 3-bit | current address |
| next\_state | 3-bit | Next state |
| next\_data\_count | 4-bit | Next count of data |
| we | 1-bit | Write enable |
| re | 1-bit | Read enable |
| to\_mux | 32-bit | From register file to mux |
| to\_ff | 32-bit | From mux to flip flop |
| 위 표는 FIFO의 Input/Output Description이다. | | | |

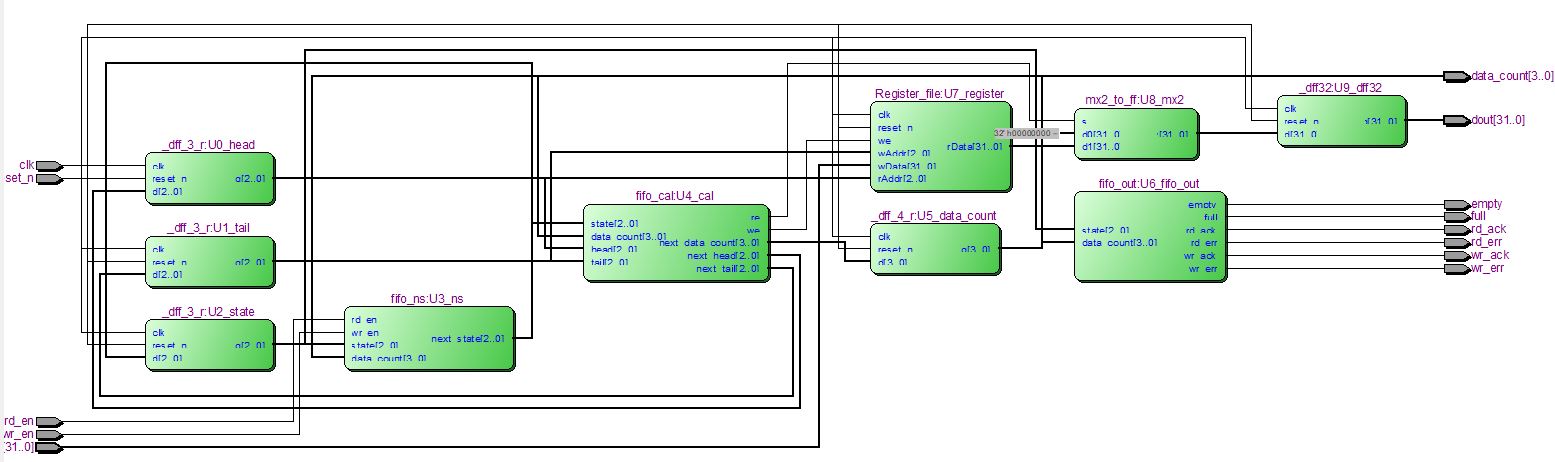
|  |  |  |
| --- | --- | --- |
| <Module Description> | | |
| Classification | Name | Desciption |
| Module | fifo | First in First out Queue |
| Instance | U0\_head | 3bit d flip flop for head |
| U1\_tail | 3bit d flip flop for tail |
| U2\_state | 3bit d flip flop for state |
| U3\_ns | Next state logic |
| U4\_cal | Calculate address logic |
| U5\_data\_count | 4bit d flip flop for data count |
| U6\_fifo\_out | Output logic |
| U7\_register | Register file |
| U8\_mx2 | From register file to flip flop mx |
| U9\_dff | 32bit d flip flop for data out |
| 위 표는 FIFO의 Module Description이다. 위 10개의 모듈들을 instance하여 설계하였음을 확인할 수 있다. | | |

1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\fi.test1.JPG |
| 위 그림은 FIFO Testbench Waveform 의 앞 부분이다. 먼저 reset을 0, rd\_en과 wr\_en 모두 0으로 초기화한 후 시작했다. 먼저 데이터를 넣지 않은 다음에 rd\_en을 1로 set 했을 때, rd\_err 가 뜨는 것을 확인할 수 있고, 그 이후로 wr\_en 신호를 주면서 큐에 값들을 변화시키면서 차근차근 넣었고, enqueue를 함과 동시에 data\_count 가 1씩 늘어나는 것을 확인할 수 있다. 이때 중간에 read를 한 번 했고, dequeue 가 잘 되었고, dout 에 처음 넣은 값 ffff0000이 출력되었음을 확인할 수 있다. 그 이후로 계속 enqueuer(write) 했고, data\_count가 8이 되고 나서 계속 write 했지만 full 상태로 변했기 때문에 wr\_err 가 1로 뜨는 것을 확인할 수 있다. |

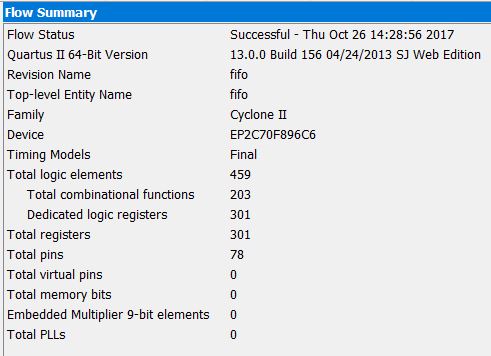
|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\fi.test2.JPG |
| 위 그림은 FIFO Testbench Waveform 의 뒷부분이다. 위에 이어 설명하도록 한다. 그 이후 rd\_en을 1로 즉, dequeue를 진행했다. 이때, dout의 값을 확인하면 처음 queue에 넣은 순서대로 FIFO 의 성질을 그대로 유지하며 넣은 순서대로 출력이 되었음을 확인할 수 있고, 이때 read가 제대로 되면서 re\_ack 신호가 1로 set 되는 것을 확인할 수 있다. dequeue 중 간에 enqueue를 한 번 실행하면서 enqueue가 되는지 확인했고, 확실히 동작함을 확인할 수 있다. 이때 모든 data를 dequeue하고나서 dequeue할 data 가 없으면 empty flag 가 1로 set 되고 계속 read를 하게 되면 rd\_err 가 뜨는 것을 확인할 수 있다. 그 이후 다시 enqueue를 하며 제대로 실행되는지 확인했고, 정확하게 출력되고있음을 확인할 수 있다. |

* 1. 합성(synthesis) 결과



< FIFO 의 RTL Map Viewer >

위 그림은 FIFO 의 RTL Map Viewer 이다. 해당하는 input 값들을 넣어주었을 때 올바른 output 값들이 나오도록 설계하였고, design 했던 table 대로 구현이 되어있음을 한 눈에 확인할 수 있다.



< FIFO의 Flow Summary>

위 그림은 FIFO 의 Flow Summary 이다. Total logic elements 가 459, total register가 301, total pins이 78 임을 확인할 수 있다.

1. 고찰 및 결론
   1. 고찰

이번 실습 시간에는 전에 구현했던 Register File을 이용해서 FIFO를 설계하는 실습이었다. 처음 FIFO를 설계하는 과정에서 강의 자료에 있는 state diagram을 이해하는 부분에는 크게 어려움이 없었지만 Verilog로 설계하는 과정에서 no operation일 때 어떻게 작동을 해야되는지에 대한 어려움이 있었다. 디지털 논리 회로2 인터넷 강의를 참고하려 했지만, design 자체가 조금 달라서 설계하는데 조금 어려움이 있었다.

처음 설계 할 때는 state가 IDLE 일 때, 모든 값을 0으로 초기화하는 식으로 설계했다. 그에 따른 문제가 발생했고, 이후 그 전 값들을 유지하고 read 신호와 write 신호만 0으로 set 한다는 것을 이해한 후 문제를 해결할 수 있었다.

또한 설계를 모두 끝난뒤, 컴파일 하는 과정에서 오류가 수백가지 이상이 나와 너무 당황했었지만 case문 에서 default 를 제대로 설정을 안해주었거나, if 문에서 마지막 else를 선언해 주지 않고 else if 로만 설계하면 오류가 생기는 점을 확인했다. 따라서 if 문이나 case 문에서 else 와 default를 항상 설정을 하지 않으면 latch가 생성되어 오류가 된다는 것을 알게되었고, 이러한 예외처리를 항상 유의하며 코드를 구현해야겠다는 생각이 들었다.

* 1. 결론

QUEUE는 FIFO(first in first out) 의 설정을 띄고있는 자료구조이다. 이러한 자료구조를 Register file과 instance하여 data들을 해당 read,write 신호에 따라 read하고, write 할 수 있는 자료구조를 verilog를 통해 설계하였고, 또한 flag들을 설정하여 제대로 진행이 되고 있는지 안되고 있는지를 한눈에 확인할 수 있었다. 이러한 성질들은 c언어와 유사한 점이 굉장히 많았다. 같은 자료구조를 다른 언어를 통해 구현할 수 있다는 점이 굉장히 흥미로웠고, 설계 과정은 비슷하나, 설계 방법이 언어마다 차이가 굉장히 많을 것으로 생각이 들었다.

또한 case문과 if문을 사용할 때 예외 처리의 중요성을 다시 한번 깨닳았다. 해당 state에서 가능한 모든 부분을 cover해야하기 때문에 예외처리를 항상 해주어야 되고, 앞으로 많은 구조들을 설계할 때 이 부분은 항상 명심해야 할 것이다.

1. 참고문헌

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

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