SoC 를 위한 *Peripheral* 설계

[*Serial Bus* ➔ *UART Controller* 구현]

[*Reference*]

- *https://ko.wikipedia.org/wiki/%EC%A7%81%EB%A0%AC_%ED%86%B5%EC%8B%A0*
- *https://ko.wikipedia.org/wiki/UART*
- *MicroBlaze.v15 [IHIL]*
- *https://electriceng.tistory.com/422*

Serial Bus

➢ *Serial Bus*(직렬 통신)

- **직렬 통신**(Serial Bus, 시리얼 버스)은 연속적으로 통신 채널이나 컴퓨터 버스를 거쳐 한 번에 하나의 비트 단위로 데이터를 전송하는 과정
- 여러 개의 병렬 채널을 갖춘 링크 위에서 동시에 여러 개의 비트를 보내는 병렬 통신과 대조적
- 컴퓨터에서 데이터 처리가 병렬로 되는데, 통신을 위해 병렬 통신을 하려면 여러개의 채널이 필요
- 거리와 비용을 고려하면 채널이 많을 경우 병렬 통신은 문제가 될 수 있다. 결국 병렬로 처리되는 데이터를 통신할 때 시간으로 나누어 차례대로 전송함으로써 문제를 해결
- 직렬 통신에서 데이터가 계속되어 전송되면, 각 비트를 구별할 방법이 필요
- Digital Circuit에서 수신된 데이터의 비트가 시간적으로 어디서부터 시작이고 끝인지를 알 필요
- 데이터 비트를 복구하기 위해 데이터의 시간적 위치를 알리기 위해 동기신호를 보내는 경우와 동기 신호 없이 신호 자체에서 데이터 비트를 복원하는 방식으로 구분
  - 동기 방식 : 데이터 신호와는 별도로 동기신호를 함께 전송
  - 비동기 방식 : 데이터 신호만을 보내고 각각의 방식에 따라 데이터 비트를 찾아냄
  - 집적회로(IC)들은 여러 개의 핀을 갖추고 있을 수록 더 비싸서, 수많은 IC들은 핀들의 수를 줄이기 위해 속도가 중요하지 않을 시점에 직렬 버스를 사용하여 데이터를 전송
  - 값싼 직렬 버스의 예 : UART, SPI, I^2C 등…

Serial Bus ➔ 종류(1) : UART

➢ *Serial Bus : **UART**(**U**niversal **A**synchronous serial **R**eceiver and **T**ransmitter)*

- **비동기식(*Asynchronous*) 통신**. 데이터 수신의 타이밍을 위하여 Clock 라인을 사용하지 않음.
- 비동기식 통신이기 때문에 보내는 데이터를 어떻게 해석할지가 중요. 데이터 해석을 위해서는 아래와 같은 정보가 필요. 송수신 측에서는 아래와 같은 규칙들을 지정하고 데이터를 해석.
  - ✓ ***Baud Rate***(보레이트)라고 하며 초당 많은 심볼(Symbol, 의미 있는 데이터 묶음)을 얼마만큼 전송할지에 대한 정보
  - ✓ **데이터 길이** : 송수신 데이터 길이(8bit or 7bit)
  - ✓ **패리티 비트** : 패리티 비트 사용 않는 경우, Even(짝수) 패리티 사용, Odd(홀수) 패리티 사용
  - ✓ **정지 비트** : 정지 비트(1 or 2개)

비트 수	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
	시작 비트 (Start bit)	*5–8 데이터 비트*								*패리티 비트 (parity bit)*	*종료 비트 (Stop bit(s))*
	Start	*Data 0*	*Data 1*	*Data 2*	*Data 3*	*Data 4*	*Data 5*	*Data 6*	*Data 7*	*Parity*	*Stop*

- **시작 비트** : 통신의 시작을 의미하며 한 비트 시간 길이 만큼 유지한다. 지금부터 정해진 약속에 따라 통신을 시작
- **데이터 비트** : 5~8비트의 데이터 전송을 한다. 몇 비트를 사용할 것인지는 해당 레지스터 설정에 따라 결정
- **패리티 비트** : 오류 검증을 하기 위한 패리티 값을 생성하여 송신하고 수신쪽에 오류 판단한다. 사용안함, 짝수, 홀수 패리티 등의 세가지 옵션으로 해당 레지스터 설정에 따라 선택할 수 있다. '사용안함'을 선택하면 이 비트가 제거
- **끝 비트** : 통신 종료를 알린다. 세가지의 정해진 비트 만큼 유지해야 한다. 1, 1.5, 2비트로 해당 레지스터 설정에 따라 결정

Serial Bus ➔ 종류(2) : SPI

➢ *Serial Bus : SPI(Serial Peripheral Interface Bus)*

- 동기식(Synchronous) 시리얼 통신 방식. 데이터 수신의 타이밍을 위하여 Clock 라인을 사용.
- 직렬 주변기기 인터페이스 버스(Serial Peripheral Interface Bus) 또는 SPI 버스는 아키텍처 전이중(Full Duplex) 통신 모드로 동작하는 모토로라 아키텍처에 이름을 딴 동기화 직렬 데이터 연결 표준
- 아래와 같은 4가지 선으로 구성
  - ✓ MOSI : 마스터 출력, 슬레이브 입력 (마스터로부터의 출력). Master Out, Slave In
  - ✓ MISO : 마스터 입력, 슬레이브 출력 (슬레이브로부터의 출력). Master In, Slave Out
  - ✓ SCK : 직렬 클럭 (마스터로부터의 출력). 데이터 전송 타이밍 동기화를 위한 Clock
  - ✓ SS(CS) : 슬레이브 셀렉트 (active low, 마스터로부터의 출력). Slave Select(또는 Chip Select)는 데이터 수신할 기기 선택을 위한 신호로 사용
- 마스터가 SS를 통해 신호를 전송할 슬레이브를 선택
- 마스터는 MOSI 를 통해서 SCLK에 동기화된 신호를 전송
- I2C가 2개의 선이 필요한 것과 달리 SPI는 선을 추가로 더 필요

Serial Bus ➔ 종류(3) : I²C

➢ *Serial Bus : I²C*

- 동기식(Synchronous) 시리얼 통신 방식. 데이터 수신의 타이밍을 위하여 Clock 라인을 사용.
- 필립스에서 개발한 직렬 버스로 마더보드, 임베디드 시스템, 휴대 전화 등에 저속의 주변 기기를 연결하기 위해 사용
- I²C 는 풀업 저항이 연결된 직렬 데이터(SDA)와 직렬 클럭(SCL)이라는 두 개의 양 방향 오픈 컬렉터 라인을 사용
- 최대 전압은 +5 V이며, 일반적으로 +3.3 V 시스템이 사용되지만 다른 전압도 가능
- 마스터(Master)와 슬레이브(Slave)가 존재
- 아래와 같은 2가지 선으로 구성
  - ✓ SDA : Data 송수신
  - ✓ SCK : Clock 전송
- 슬레이브마다 지정된 주소 값을 가지고 데이터를 주고 받는데, 데이터를 주고 받을 때 반드시 주소 값을 붙여서 전송
- SPI가 여러 개의 선이 필요한 것과 달리 2개의 선만 가지고 통신이 가능

SPI(Serial Peripheral Interface Bus)

Serial Bus ➔ SPI(1)

➢ *Serial Bus : SPI(Serial Peripheral Interface Bus) (1)*

- SPI통신은 I2C통신과 같은 통신방법의 한 종류이지만 통신하는 방법이 다름
  - I2C통신은 한사람이 보낼때는 다른사람이 받고만 있어야하는 무전기와 같은 통신방식
  - SPI 통신은 한사람이 데이터를 보내면서 데이터를 받을 수 있는 전화같은 방식
- 어느 통신방법을 쓸 것인가에 대하여는 사용자들의 몫이기 때문에 장단점등을 잘 확인하여 사용

❖ SPI 통신의 장점

1. 완전한 전이중(Full duplex) 통신 : 양방향 통신이다. 위에서 설명한 것처럼 말하면서 들을수 있다는 이야기.
2. 전송되는 비트에 대한 완전한 프로토콜 유연성 : 최대 16비트까지 맘대로 길이를 조절 가능
3. 전송기가 불필요 : 흔히 말하는 트랜시버를 사용할 필요가 없음
4. 매우 단순한 하드웨어 인터페이스 처리 : 아주 단순한 센서나 메모리에서 많이 사용
5. IC 패키지에 4개의 핀만 사용
6. 최대 클럭이 제한되지 않아 속도 제한이 없음
7. Push-Pull 출력(Open Drain이 아닌)을 사용하여 상호간에 같은 전압을 사용하여 시그널 정합성과 고속 지원
8. I2C 보다 낮은 소비 전력
9. 슬레이브는 마스터가 보내주는 클럭만을 사용하고 정확성이 떨어져도 문제 없음

Serial Bus ➔ SPI(2)

➢ *Serial Bus : SPI(Serial Peripheral Interface Bus) (1)*

- PI통신은 I2C통신과 같은 통신방법의 한 종류이지만 통신하는 방법이 다름
  - I2C통신은 한사람이 보낼때는 다른사람이 받고만 있어야하는 무전기와 같은 통신방식
  - SPI 통신은 한사람이 데이터를 보내면서 데이터를 받을 수 있는 전화같은 방식
- 어느 통신방법을 쓸 것인가에 대하여는 사용자들의 몫이기 때문에 장단점등을 잘 확인하여 사용

❖ SPI 통신의 단점

1. 하드웨어 슬레이브 인식이 없음
2. 슬레이브에 의한 하드에워 흐름제어가 없음
3. 오류 검사 프로토콜이 정의되어 있지 않음 (에러 체킹 지원)
4. 노이즈 스파이크에 영향을 받는 경향이 있음
5. RS-232, CAN 버스보다 비교적 더 짧은 거리에서 동작 (칩간 통신에서만 주로 사용)
6. 하나의 마스터 장치만 지원
7. 인밴드(디폴트 SPI wire)를 통해 주소가 지원되지 않아 다수의 슬레이브를 사용 시 별도의 아웃밴드(칩 셀렉트 라인)를 통해 슬레이브를 선택
8. Hot 플러그를 지원하지 않음

Serial Bus ➔ SPI(3)

- *SPI(Serial Peripheral Interface Bus) ➔ Applications*

  (1) EEPROM, 플래시 메모리 (2) MMC or SD Card (3) LCD (4) RTC(Real Time Clock)

  (5) 각종 센서류

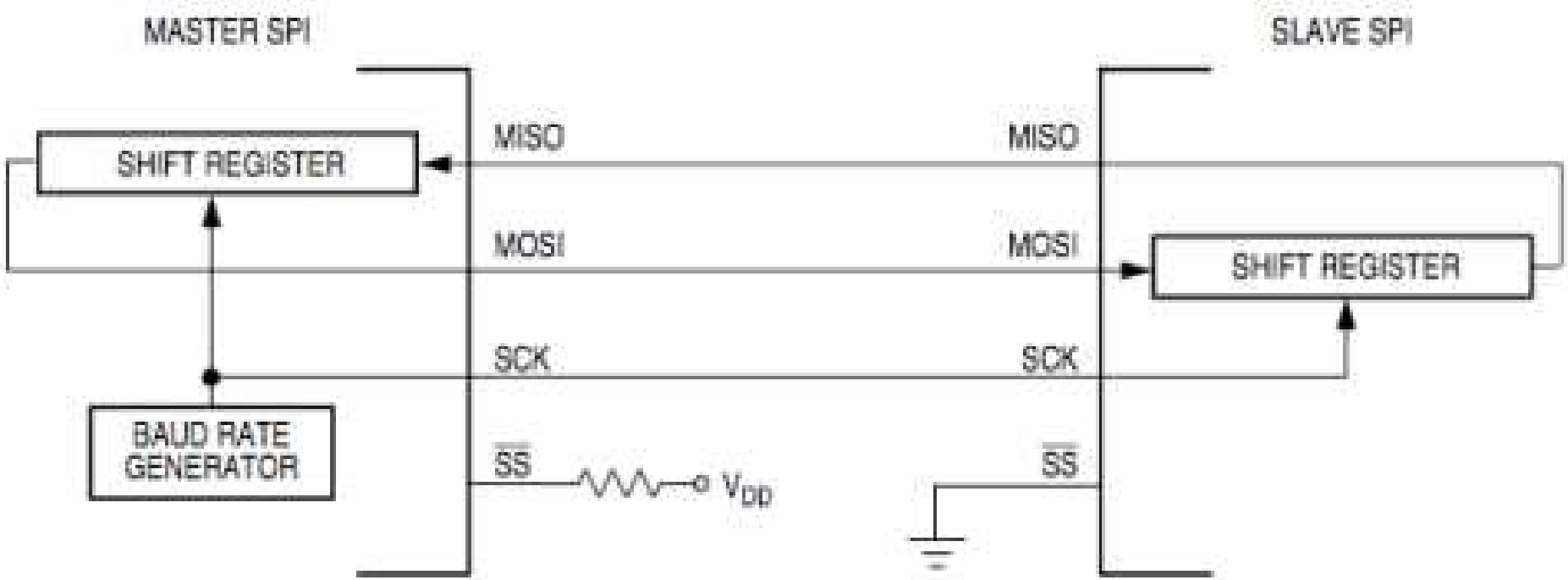
- *SPI(Serial Peripheral Interface Bus) ➔ SPI Pin Description*

1. I2C통신에서는 데이터를 전송하는 SDA(Serial Data)선과 클럭을 전송하는 SCL(Serial Clock)선 두개만 필요
2. SPI통신에서는 송신과 수신을 위한 MOSI(Master Out Slave IN)선과 MISO(Master In Slave Out)선, 클럭(SCLK)선, SS(Slave Select)선 이렇게 4개가 필요
3. MOSI(Master Out Slave IN) ➔ 이름 그대로 마스터에서 나와서 슬레이브에게 가는 데이터를 전송하는 선
4. MISO(Master In Slave Out) ➔ 슬레이브에게 나와서 마스터에게 가는 데이터를 전송하는 선 (I2C통신이 전 이중통신이 안됐다면 SPI는 선이 구분되어 있기 때문에 송수신이 동시에 가능.다만 SPI통신은 Master-Slave 관계(주인-종)처럼 Slave는 Master에게 통신은 마음대로 하지 못함.)
5. SPI의 데이터 송수신은 동시에 이루어지기 때문에 데이터 송신이 곧 수신. 이 데이터 송수신의 타이밍을 결정짓는 것은 Master장치
6. SCLK, CLK(Clock) 은 동기화 신호를 위한 Clock
7. SS(Slave Select) ,CS(Chip Select)선 ➔ Master장치가 여러개의 Slave 장치와 통신할 때 하나를 선택하여 통신을 하는데 그 때 선택을 하는 역할의 신호선, 한마디로 어디로 보낼지 결정하는 신호선

Serial Bus ➔ SPI(4)

➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* 통신의 동작 구조(1)

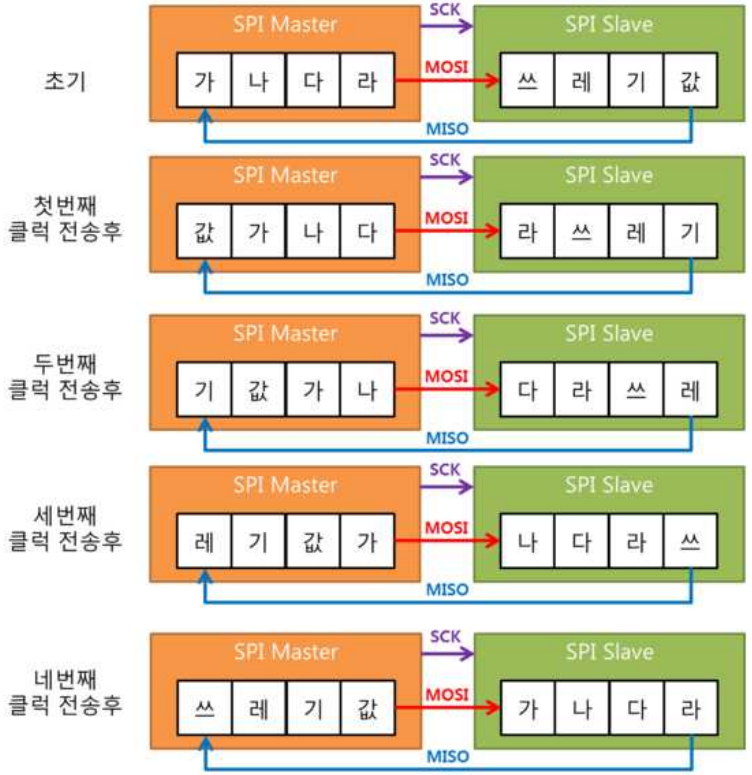
1. SPI 장치들은 Shift Register를 가지고 있음
2. 기본적으로 MSB부터 전송되는데 특정 컨트롤러는 LSB부터 전송을 수행시키는 방법도 지원
3. 그림처럼 SHIFT REGISTER에 의해서 데이터가 이동
4. 마스터에서 어떤 값을 슬레이브로 보낸다고 가정했을 때, 보내는 1Clock의 신호 마다 1bit의 data가 이동
5. 밀어내기 식으로 data가 들어간다고 생각하면 됨

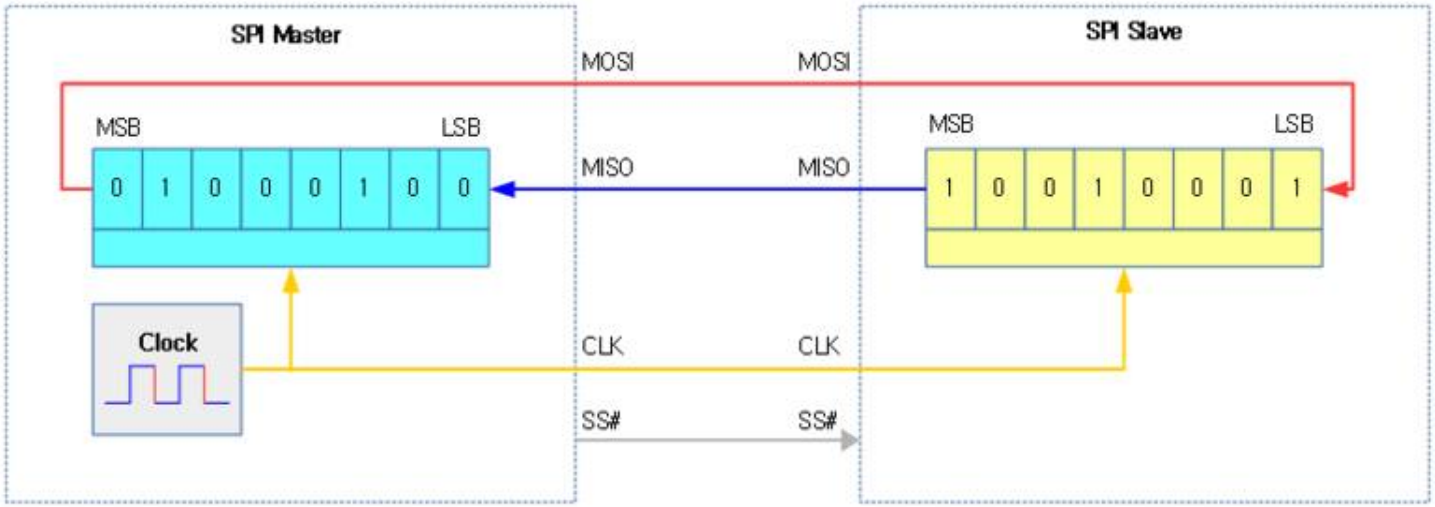


Serial Bus ➔ SPI(5)

➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* 통신의 동작 구조(2)

1. SPI 장치들은 Shift Register를 가지고 있음
2. 기본적으로 MSB부터 전송되는데 특정 컨트롤러는 LSB 부터 전송을 수행시키는 방법도 지원
3. 그림처럼 SHIFT REGISTER에 의해서 데이터가 이동
4. 마스터에서 어떤 값을 슬레이브로 보낸다고 가정했을 때, 보내는 1Clock의 신호 마다 1bit의 data가 이동
5. 밀어내기 식으로 data가 들어간다고 생각하면 됨





Serial Bus ➔ SPI(6)

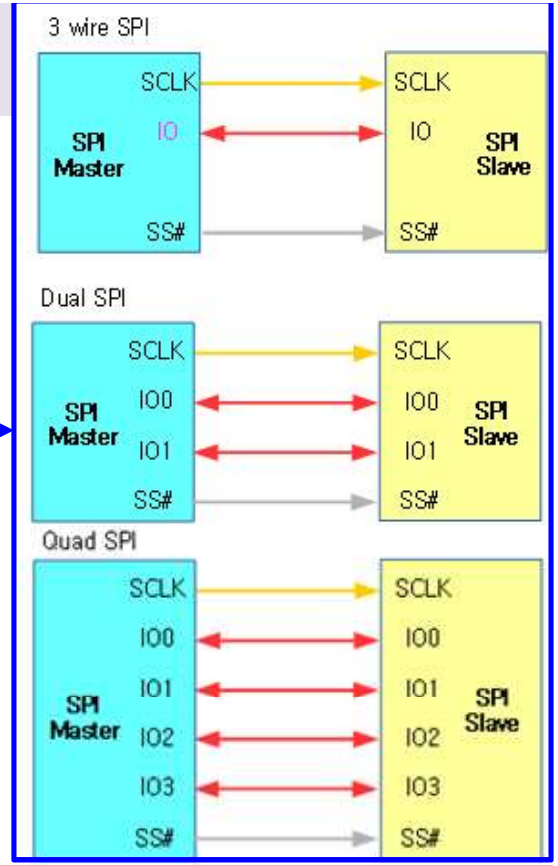
➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* 전송모드

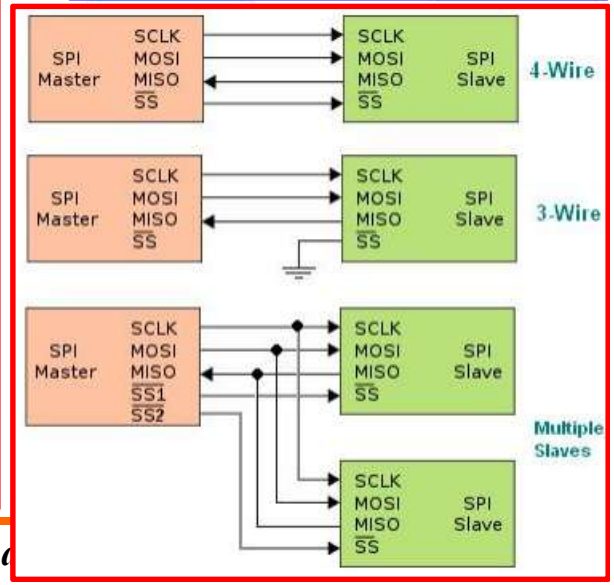
❖ 3 wire SPI
- Half Duplex 입출력 wire를 1개 지원
- 저속 EEPROM 및 센서류에서 사용

❖ Dual SPI
- Half Duplex 입출력 wire를 2개 지원한다. (단방향 속도가 2배)

❖ Quad SPI
- Half Duplex 입출력 wire를 4개 지원한다. (단방향 속도가 4배)
- SPI-Nor 플래시 및 SPI-Nand 플래시에 자주 사용

❖ 다른 예시 3-Wire, 4-Wire
- 1번이 4-Wire 방식, 2번이 3-Wire 방식, 3번이 다중 슬레이브 연결 시의 결선도
- 1번의 경우, 일반적으로 연결하는 방식이며, SS를 통해 data 전송 알림 신호를 보내 data를 전달하는 방식.
- 2번의 경우, Slave장치 에서 SS를 GND에 묶어두면 Slave장치는 항상 data를 받을 수 있는 상태가
- 3번의 경우, Slave를 여러개 연결할 경우 SS1, SS2를 통해 data 값을 전달하는 방식



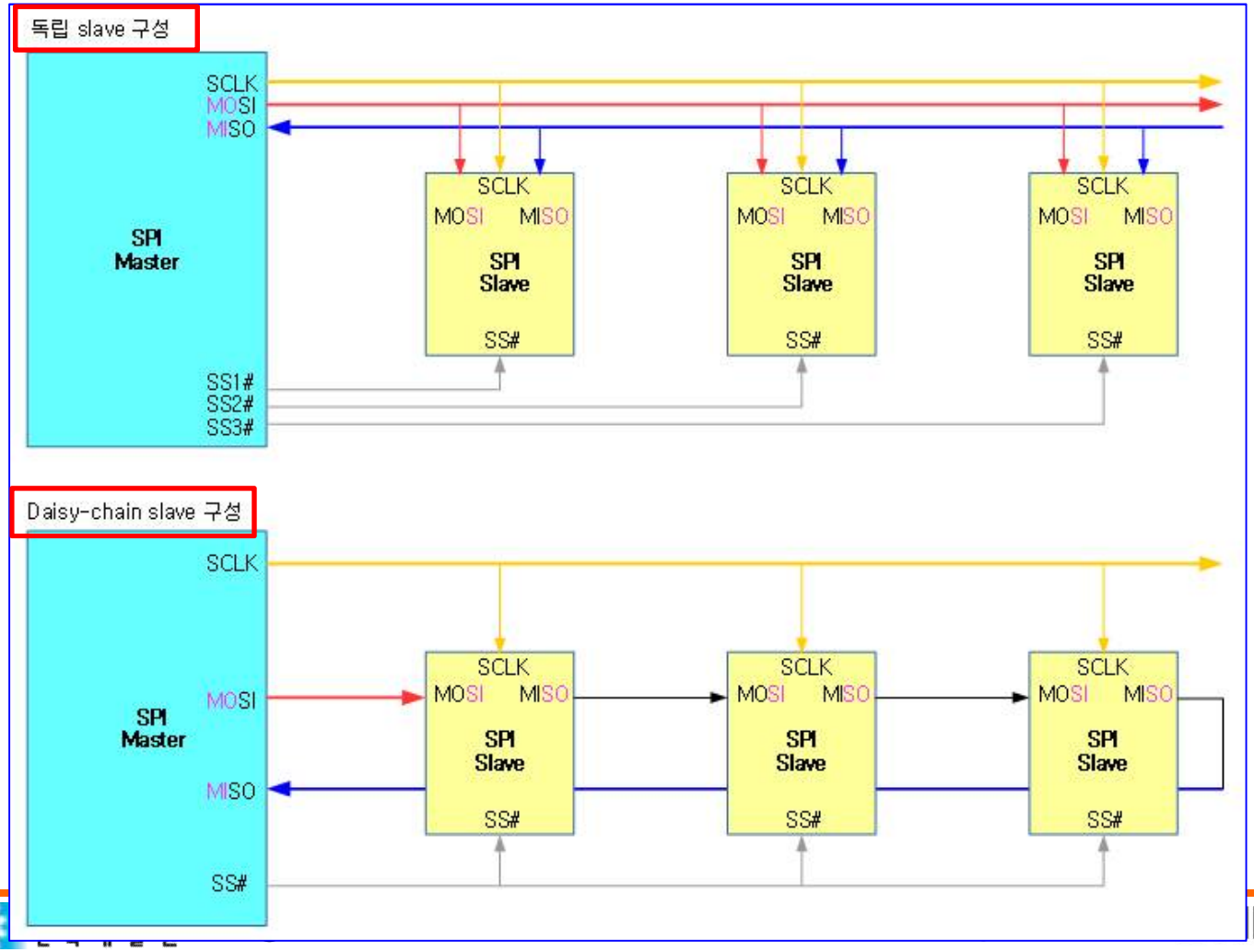


Serial Bus ➔ SPI(7)

➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* 연결 구성 방식

❖ SPI 버스를 구성하는 방법으로 다음과 같이 두 가지 방식을 사용

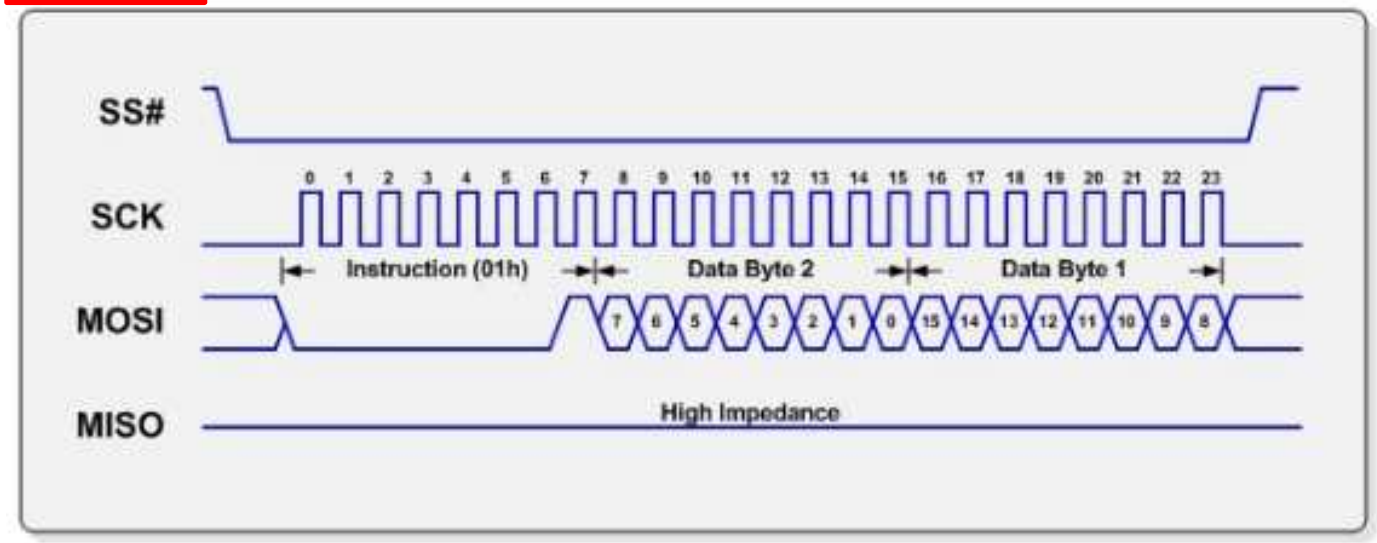
- 독립 slave 구성 방식 ➔ SPI 디바이스를 선택하여 사용하는 방식
- Daisy-chain slave 구성 방식 ➔ 데이터 비트를 서로 밀어내는 방식



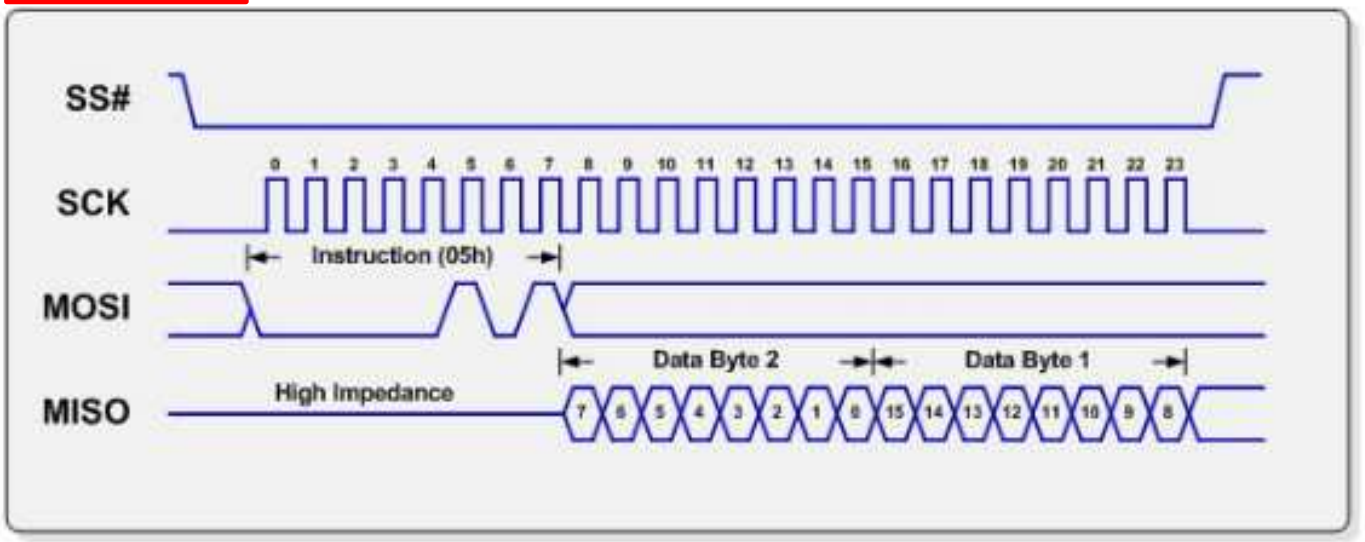
Serial Bus ➔ SPI(8)

➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* : 간단한 SPI의 Write 및 Read 동작(1)

Write



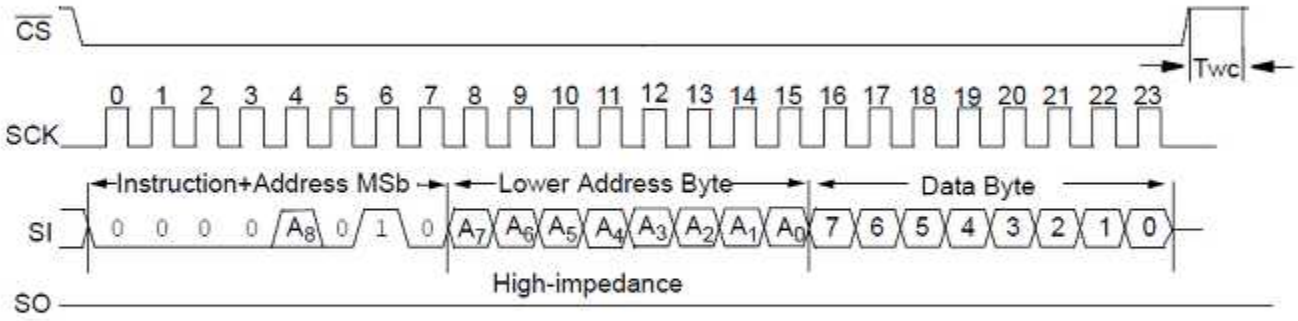
Read



Serial Bus ➔ SPI(9)

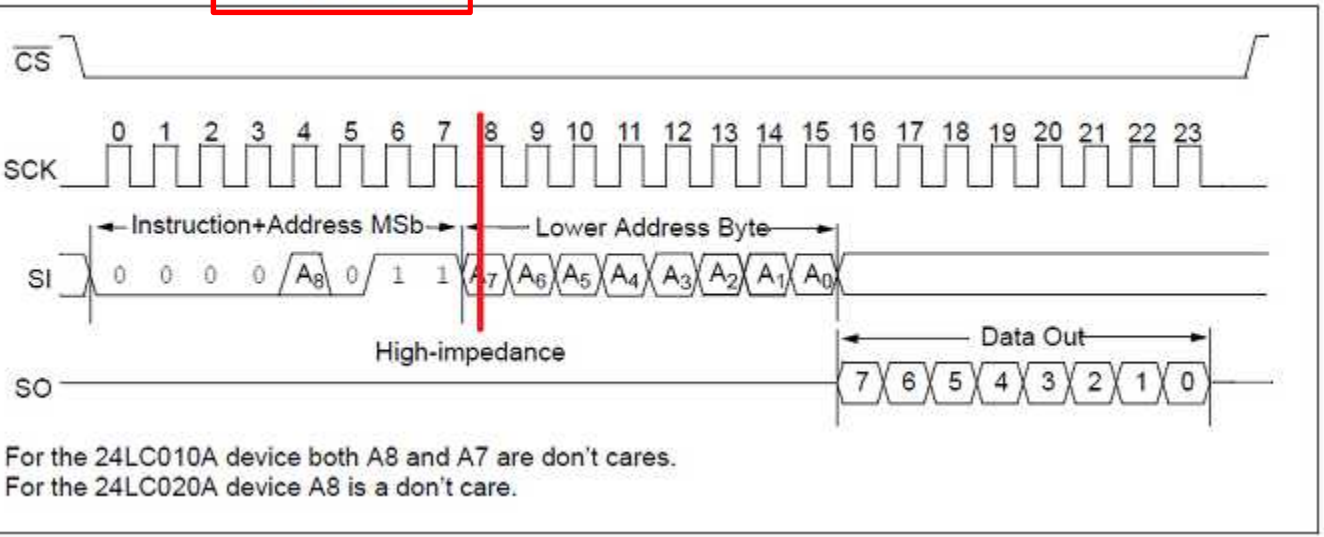
➢ *SPI(Serial Peripheral Interface Bus)* ➔ *SPI* : 간단한 SPI의 Write 및 Read 동작(2)

FIGURE 3-2: BYTE WRITE SEQUENCE



For the 24LC010A device both A8 and A7 are don't cares.
For the 24LC020A device A8 is a don't care.

FIGURE 3-1: READ SEQUENCE



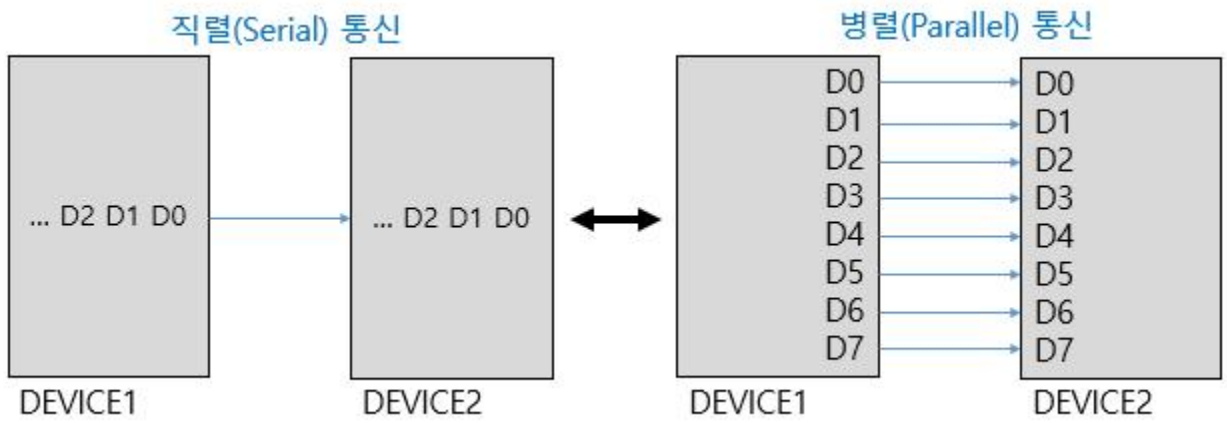
For the 24LC010A device both A8 and A7 are don't cares.
For the 24LC020A device A8 is a don't care.

UART(Universal Asynchronous serial Receiver and Transmitter)

Serial Bus ➔ UART(1)

➢ *Serial Bus : **UART**(**U**niversal **A**synchronous serial **R**eceiver and **T**ransmitter) (1)*

- 직렬 통신 : 1개의 입출력 핀을 통해 8개 비트를 한 번에 전송하는 방법
- 병렬(Parallel) 통신 : n비트의 데이터를 전송하기 위해 n개의 입출력 핀을 사용하는 방법

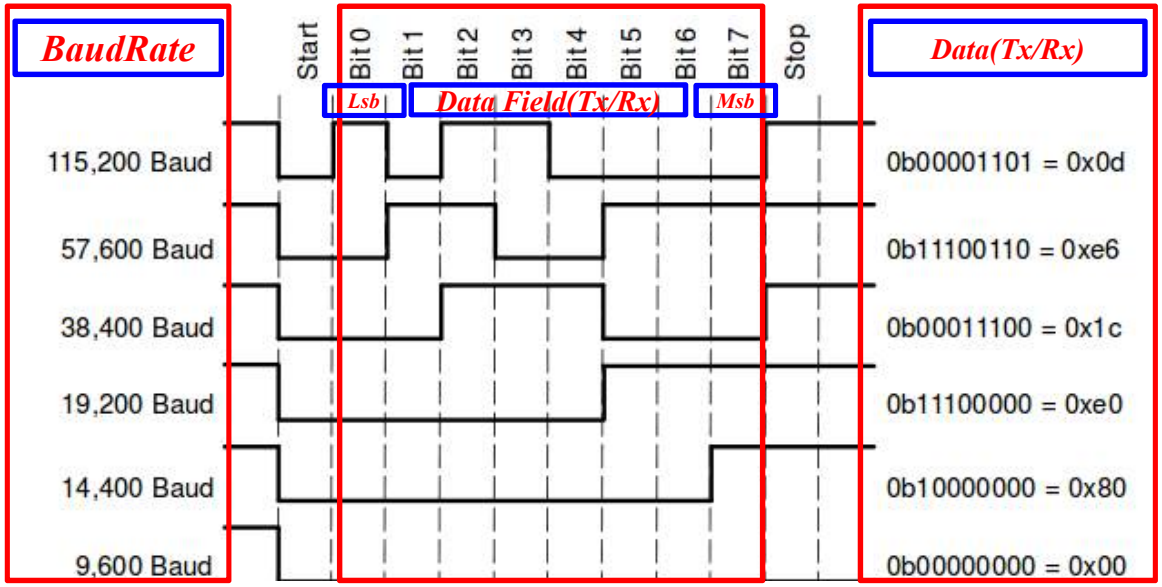


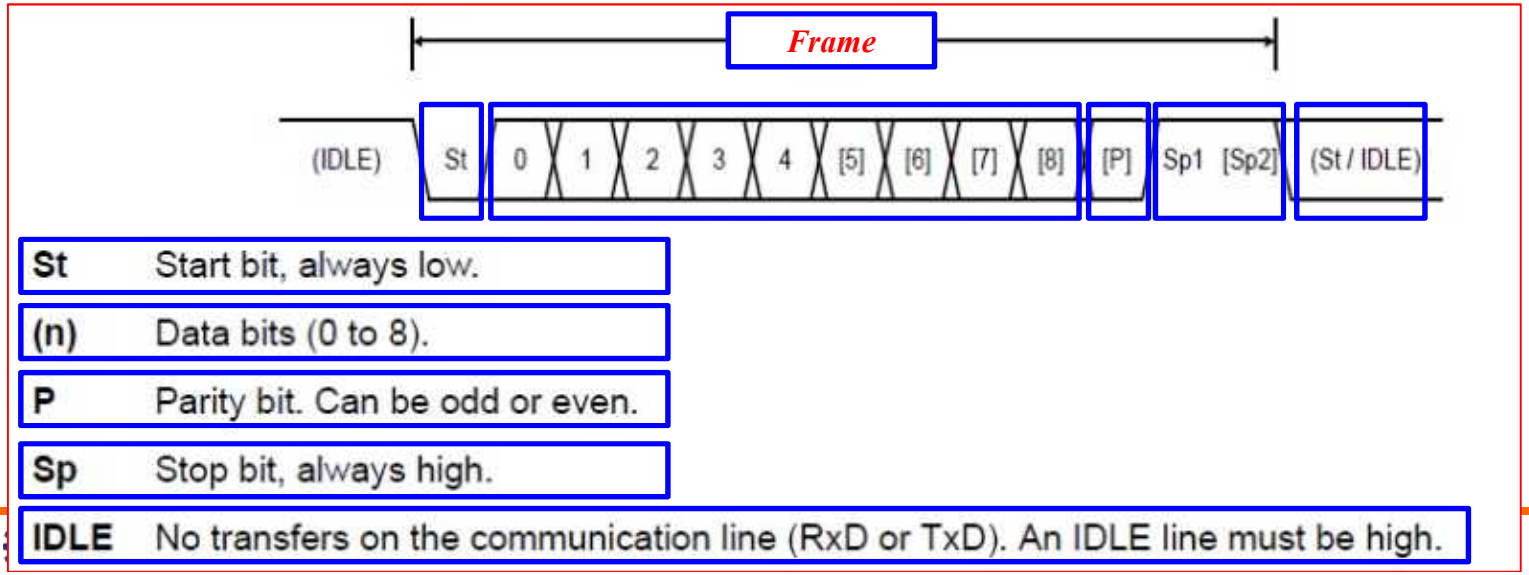
- **MCU 등에서는 직렬 통신 방식**이며, 그 중 **많이 쓰이는 방법 중 하나가 *UART***
- **시리얼 통신을 사용하기 위해서는 보내는 쪽(Tx)와 받는 쪽(Rx)에서 약속을 정해야**하는데, 이를 **프로토콜(*Protocol*)**이라고 함
- 디지털 회로에서 0과 1의 값만 처리할 수 있으므로 0은 GND, 1은 VCC로 데이터 전송하고, 받는 쪽에서는 GND와 VCC를 0과 1의 이진 값으로 변환하여 사용
- 보내는 쪽(Tx)와 받는 쪽(Rx)이 원활하게 데이터를 처리하려면, **데이터를 보내는 속도에 대하여 약속(프로토콜, Protocol)이 정해져 있어야 함**

Serial Bus ➔ UART(2)

➢ *Serial Bus : UART(Universal Asynchronous serial Receiver and Transmitter) (2)*

- ***UART***에서는 **보내는 쪽(Tx)과 받는 쪽(Rx) 에서 데이터를 보내는 속도**를 **보레이트(*Baud Rate*)**로 정함

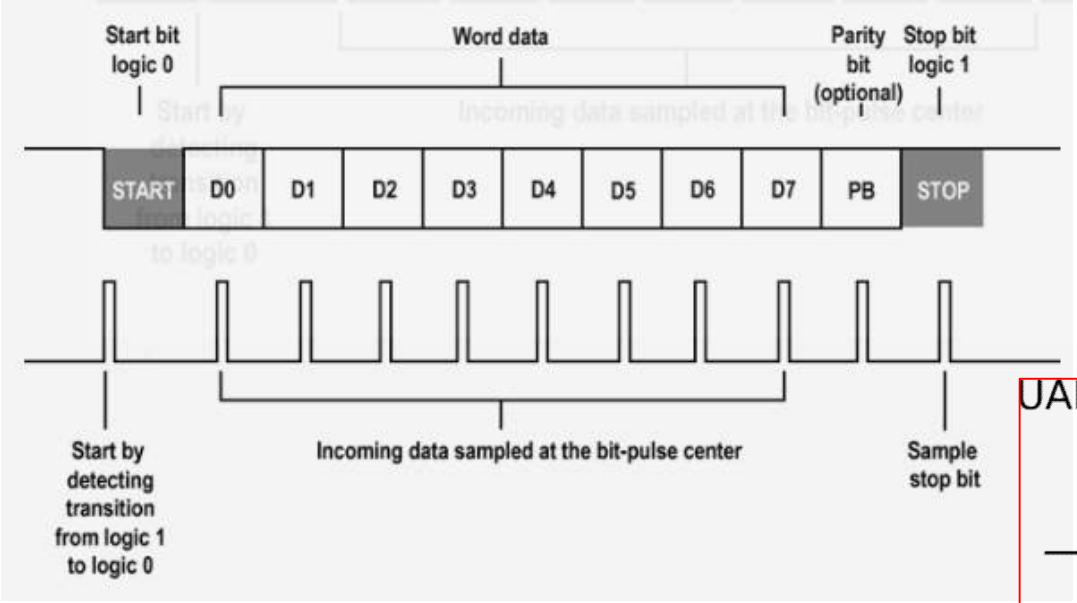


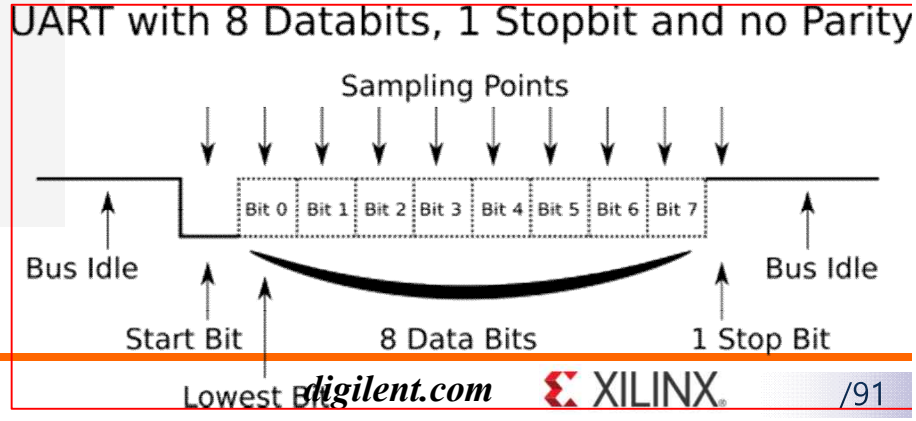


Serial Bus ➔ UART(3)

➢ *Serial Bus : UART(Universal Asynchronous serial Receiver and Transmitter)(3)*

- Tx 와 Rx 가 동일한 속도로 데이터를 주고 받는다고 하여 정확하게 통신이 되지는 않음
- Tx는 항상 데이터를 보내는 것이 아니며, 필요한 경우에만 데이터를 전송하므로, **Rx 는 Tx가 언제 데이터를 보내는지와 Tx에서 보내는 데이터의 시작이 어디서부터인지 알아낼수 있는 방법이 필요**
  ➔ **'0'의 시작비트(*St, Start Bit*)와 '1'의 정지 비트(*Sp , Stop Bit*)를 사용**
- ***UART*는 바이트(1byte=8bit) 단위 통신**을 주로 사용, **시작비트와 정지 비트가 추가되어 10비트의 데이터를 전송**(패리티 비트를 사용하지 않는 경우)





UART Controller Implementation

- ***UART TX** Implementation*
- ***FIFO_16x8** Implementation*
- ***UART RX** Implementation*
- ***UART Controller** Implementation*
- ***UART Loopback** Implementation*

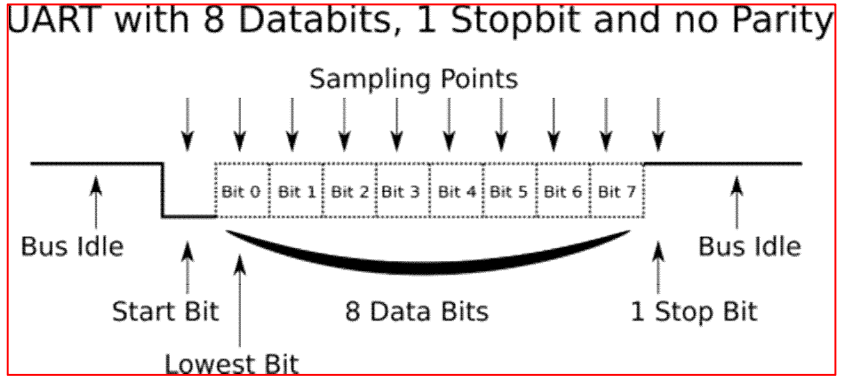
UART Controller Implementation

➢ ***UART****(Universal Asynchronous serial Receiver and Transmitter) Controller Implementation*

- UART는 비동기식(Asynchronous), 전이중(Full-Duplex) 통신 방식의 통신 프로토콜
- UART를 이용하여 RS-232, RS-485, RS-422 구현
- ***UART Frame*** 구조

Start Bit	Data Field	Parity bit	Stop Bits
1	5 ~ 9	0 or 1 or 2	1 or 2

  - ✓ ***IDLE*** : Logic High (1)
  - ✓ ***Start bit*** : 1-bit, Logic Low (0)
  - ✓ ***Data Field*** : 5 ~ 9 bits
  - ✓ ***Parity bit*** : 0 or 1 bits (***None***, ***Odd*** Parity, ***Even*** Parity)
  - ✓ ***Stop bit*** : 1 or 2 bits, Logic High (1)

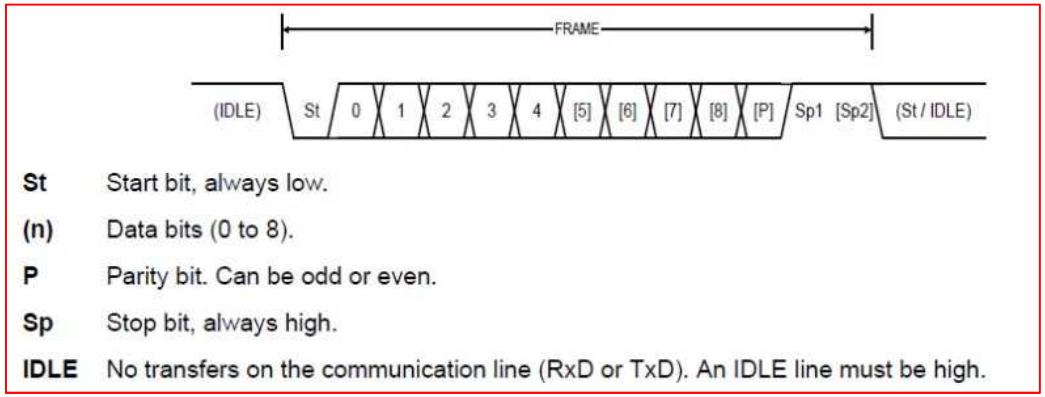


- ***IDLE*** 상태는 ***Logic High(1)***
- ***Start bit*** 는 1-bit 로 구성, ***Logic Low(0)***
- ***Data Field*** 는 5 ~9 bits까지 가능(보통 ***8bit***)
- ***Parity bit***는 사용하지 않는 경우***(None, '00')***, Odd Parity, Even Parity 를 사용하는 경우***('01', '10')***
- ***Parity*** 계산은 Data Field 의 bit 값으로 계산
- ***Stop bit***는 1-bit or 2-bits로 구성되고, ***Logic High(1)***

UART Controller Implementation

➢ UART Controller Implementation

- ***UART Controller Spec***.
  - ❖ ***Baudrate*** : 2,400 ~ 115,200 (그 이상도 가능하나 ***UART Transmitter*** 성능에 따라 다름)
  - ❖ ***Data Field*** : 8bits 로 고정 (8bits 가 아닌 경우는 거의 없음)
  - ❖ ***Parity bit*** : ***none, odd parity, even parity*** 지원
  - ❖ ***Stop bit*** : ***1 or 2 bits*** 지원
  - ❖ 수신단(***UART Receiver Stage***) : ***16 bytes FIFO*** 지원
  - ❖ ***Error bit*** 정의
    - ✓ ***overrun error*** : 수신 버퍼 full
    - ✓ ***frame error*** : stop bit error
    - ✓ ***parity error*** : parity bit error

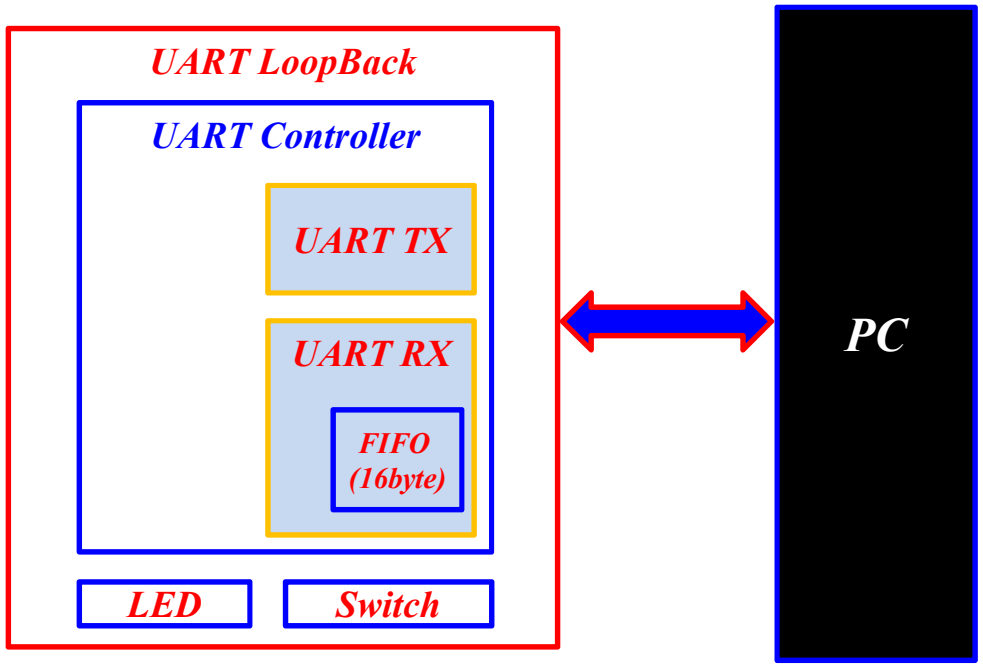


UART Controller Implementation

➢ UART Controller Implementation

▪ UART Controller System Block

- ❖ ***UART TX*** : *Uart Tx Module*
- ❖ ***UART RX*** : *Uart Rx Module, 16bytes FIFO* 포함
- ❖ ***UART Controller*** : *UART Tx/Rx Controller*
- ❖ ***UART LoopBack*** : *LoopBack Test*, PC로부터 데이터를 수신하면 그대로 PC로 전송
- ❖ ***Led*** : 상태(***Tx/Rx/Error***) 표시 ➔ *Tx available, Rx available, overrun error, frame error, parity error*
- ❖ ***SW*** : 모드 설정 (***baudrate, parity, stop bit***)



UART Controller Implementation

➢ UART Controller Implementation

▪ Baudrate

- ❖ ***UART Controller Main Clock***은 ***100MHz***를 사용
- ❖ 넓은 ***Baudrate***를 설정할 수 있도록 Baudrate 설정 레지스터는 16bits를 사용

➢ 100Mhz일 때, 각 Baudrate 에 따른 설정값 ➔ [***100M / baudrate*** = 변환 값(수식 값)]

baudrate	수식 값	정수값으로 변환	baudrate2	오차율
2400	41666.67	41667	2400.0	0.00%
4800	20833.33	20833	4800.1	0.00%
19200	5208.33	5208	19201.2	0.01%
38400	2604.17	2604	38402.5	0.01%
57600	1736.11	1736	57603.7	0.01%
76800	1302.08	1302	76804.9	0.01%
115200	868.06	868	115207.4	0.01%
230400	434.03	434	230414.7	0.01%
460800	217.01	214	467289.7	1.41%
921600	108.51	109	917431.2	-0.45%
1843200	54.25	54	1851851.9	0.47%

- ✓ **"정수값으로 변환"한 값들은 "수식 값"을 반올림한 값**
- ✓ **"baudrate2"는 "정수값으로 변환한 값"을 이용해서 실제로 계산된 Baudrate [= *100M* / 변환 값(수식 값)]**

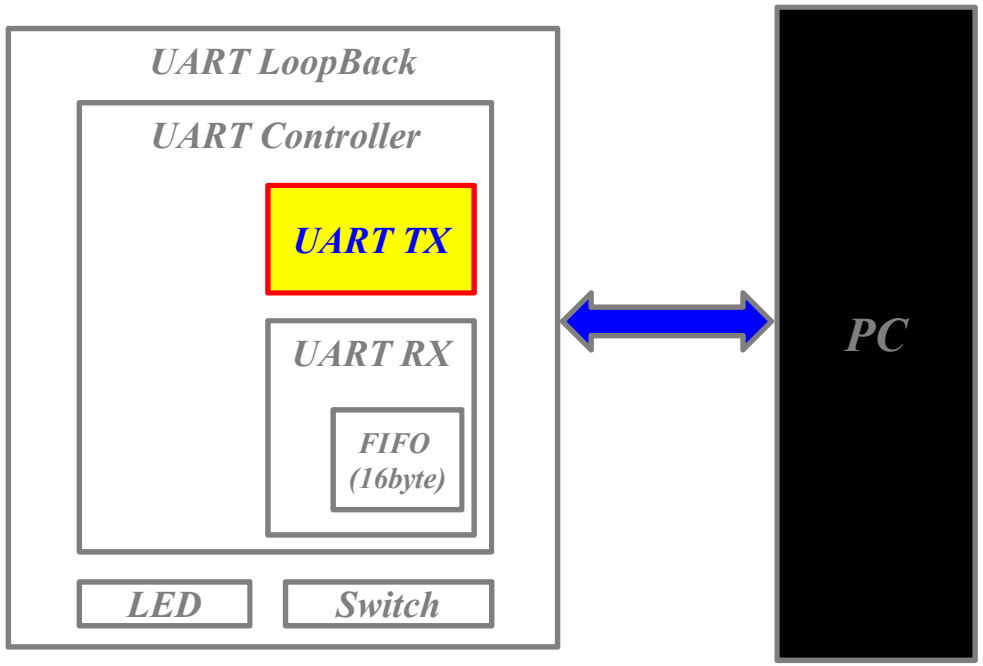
UART Controller Implementation

- ***UART TX** Implementation*
- *FIFO_16x8 Implementation*
- *UART RX Implementation*
- *UART Controller Implementation*
- *UART Loopback Implementation*

UART Controller Implementation

➢ UART Controller Implementation

- **UART Controller System Block**
  - ❖ **UART TX**: Uart Tx Module
  - ❖ UART RX : Uart Rx Module, 16bytes FIFO 포함
  - ❖ UART Controller : UART Tx/Rx Controller
  - ❖ UART LoopBack : LoopBack Test, PC로부터 데이터를 수신하면 그대로 PC로 전송
  - ❖ Led : 상태를 표시함, Tx available, Rx available, overrun error, frame error, parity error
  - ❖ SW : 모드 설정 (baudrate, parity, stop bit)



UART Controller Implementation

➢ ***UART TX*** *Implementation*

1. Port 정의

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation

✓ ***UART_TX*** ➔ ***input & out port***를 정의

- ***UART_TX*** Module 을 이용하여 데이터를 전송하기 위해서는, ***buadrate, parity_sel, stop_sel, tdata***를 설정 ➔ ***send flag***를 ***enable***
- 전송이 완료되면, ***done flag***가 ***active***

signal	in/out	size	description
reset	***in***	[0]	*reset, Active Low*
mclk	***in***	[0]	*clock, 100Mhz*
baudrate	***in***	[15:0]	baudrate 설정
parity_sel	***in***	[1:0]	parity 설정 [***0 : none, 1 : even, 2 : odd***]
stop_sel	***in***	[0]	stop bits 설정 [***0 : 1bit, 1 : 2bits***]
tdata	***in***	[7:0]	send data
send	***in***	[0]	send flag ➔ High 일 때 전송 시작, pulse 로 입력
txd	***out***	[0]	*uart_txd*
done	***out***	[0]	송신 완료 *flag [1 : idle* 상태 or 송신 완료] , [*0* : 송신 중]

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation

✓ ***UART_TX ➔ input & out port***를 정의

Uart_Tx . v (1)

```
1.  module Uart_Tx(
2.              reset     ,
3.              mclk      ,
4.              baudrate  ,
5.              parity_sel ,  // 0 : none,  1 : even, 2 : odd
6.              stop_sel   ,  // 0 : 1bits, 1 : 2bits
7.              tdata     ,
8.              send      ,
9.              txd       ,
10.             done
11.);
12. input               reset     ;
13. input               mclk      ;
14. input   [15:0]      baudrate  ;
15. input   [1:0]       parity_sel ;
16. input               stop_sel  ;
17. input   [7:0]       tdata     ;
18. input               send      ;
19. output              txd       ;
20. output              done      ;
```

✓ 1 ~ 20 : port 선언

✓ reset ➔ in [0] : reset, Active Low

✓ mclk ➔ in [0] : clock, 100Mhz

✓ *baudrate* ➔ in [15:0] : baudrate 설정 [16bit = 65,536],

[100M/65,536 = 1,520(변환값)]

✓ *parity_sel* ➔ in [1:0] : parity 설정 [*0* : none, *1* : *even*, *2* : odd]

✓ stop_sel ➔ in [0] : stop bits 설정 [*0* : 1bit, *1* : 2bits]

✓ tdata ➔ in [7:0] : send data , 8bit

✓ send ➔ in [0] : send flag ➔ *High* 일 때 전송 시작 : pulse 입력

✓ txd ➔ out [0] : *uart_txd*

✓ done ➔ out [0] : 송신 완료 flag [1 : idle 상태 or 송신 완료] ,

[0 : 송신 중]

UART Controller Implementation

- ***UART TX*** *Implementation*
  1. *Port 정의*
  2. *State 정의*

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation

✓ UART_TX ➔ state 및 state flag 정의

Uart_Tx . v (2)

```
21.  //----------------------------------------------
22.  // 1) define state
23.  reg                 m_state ;
24.  parameter           M_IDLE    = 1'b0 ;
25.  parameter           M_RUN     = 1'b1 ;
26.  //----------------------------------------------
27.  // 2) state flag
28.  wire                s_idle  = (m_state == M_IDLE ) ? 1'b1 : 1'b0 ;
29.  wire                s_run   = (m_state == M_RUN  ) ? 1'b1 : 1'b0 ;
```

➔ *send flag*

✓ 23 ~ 25 : state 정의

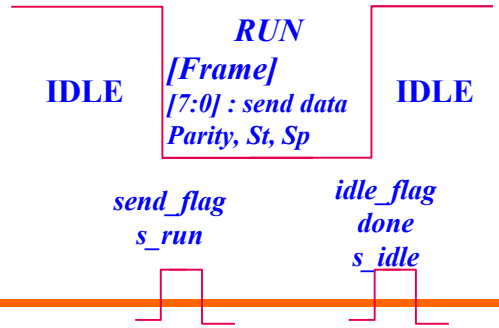
➔ IDLE, RUN 2개로 정의 : IDLE 상태에서 send flag가 active 되면 RUN 상태가 되면서 데이터를 전송

➔ 전송이 완료되면 done flag를 enable 하고 IDLE 모드

✓ 28 ~ 29 : state flag 정의

➔ m_state == 1'b0 이면 ➔ *s_idle = 1'b1* , 아니면 1'b0

➔ m_state == 1'b1 이면 ➔ *s_run = 1'b1* , 아니면 1'b0



UART Controller Implementation

- **UART TX** Implementation
  1. Port 정의
  2. State 정의
  3. Code Implementation

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation

✓ UART_TX ➔ cnt1 & cnt2 & parity 정의

baudrate	수식 값	정수값으로 변환	baudrate2	오차율
2400	41666.67	41667	2400.0	0.00%
4800	20833.33	20833	4800.1	0.00%
19200	5208.33	5208	19201.2	0.01%
38400	2604.17	2604	38402.5	0.01%
57600	1736.11	1736	57603.7	0.01%
76800	1302.08	1302	76804.9	0.01%
115200	868.06	868	115207.4	0.01%
230400	434.03	434	230414.7	0.01%
460800	217.01	214	467289.7	1.41%
921600	108.51	109	917431.2	-0.45%
1843200	54.25	54	1851851.9	0.47%

Uart_Tx . v (3)

```
30.  // ----------------------------------------
31.  // 3) code implementation

32.  reg      [15:0]          cnt1;          // baudrate counter
33.  always @(posedge mclk or negedge reset)
34.  begin
35.          if(~reset)      cnt1 <= 16'b0;
36.          else            cnt1 <= s_idle ? 16'b0 : (cnt1==baudrate) ? 16'b0 : cnt1+1'b1;
37.  end

38.  reg      [3:0]           cnt2;          // bit counter
39.  always @(posedge mclk or negedge reset)
40.  begin
41.          if(~reset)  cnt2 <= 4'b0;
42.          else            cnt2 <= s_idle ? 4'b0 : (cnt1==baudrate) ? cnt2+1'b1 : cnt2;
43.  end

44.  wire     tdata_xor = tdata[0]^tdata[1]^tdata[2]^tdata[3]^tdata[4]^tdata[5]^tdata[6]^tdata[7];

45.  reg                      parity;
46.  always @(posedge mclk or negedge reset)
47.  begin
48.          if(~reset)  parity <= 1'b0;
49.          else            parity <= (parity_sel==2'b01) ? tdata_xor : ~tdata_xor ;
50.  end
```

xnor

✓ 32 ~ 37 : baudrate까지 counting 하는 *counter(cnt1, 16bit)* 정의, 1bit duration

➔ s_idle 이 '0' 이고 cnt1 이 baudrate 이 아니면 cnt1을 1씩 up

✓ 38 ~ 43 : 전송하는 *bit counter(cnt2, 4bit)*정의

➔ s_idle 이 '0' 이고 cnt1 이 baudrate 이면 cnt2을 1씩 up

✓ 44 ~ 50 : 전송할 데이터의 Parity bit 를 계산

➔ *parity_sel = 2'b01 : even* ➔ *XOR* 연산

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation ➔ UART_TX ➔ txd 정의

Uart_Tx . v (4)

```
51.  reg                    txd;
52.  always @(posedge mclk or negedge reset)
53.  begin
54.          if(~reset)  txd <= 1'b1;
55.          else        txd <= s_idle ? 1'b1 :
56.  ((cnt2==4'd0) && (cnt1==16'd0)) ? 1'b0    :          // start bits
57.  ((cnt2==4'd1) && (cnt1==16'd0)) ? tdata[0] :          // data
58.  ((cnt2==4'd2) && (cnt1==16'd0)) ? tdata[1] :
59.  ((cnt2==4'd3) && (cnt1==16'd0)) ? tdata[2] :
60.  ((cnt2==4'd4) && (cnt1==16'd0)) ? tdata[3] :
61.  ((cnt2==4'd5) && (cnt1==16'd0)) ? tdata[4] :
62.  ((cnt2==4'd6) && (cnt1==16'd0)) ? tdata[5] :
63.  ((cnt2==4'd7) && (cnt1==16'd0)) ? tdata[6] :
64.  ((cnt2==4'd8) && (cnt1==16'd0)) ? tdata[7] :
65.
66.  ((parity_sel==2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :
67.
68.  ((parity_sel==2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :
69.  ((parity_sel==2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
70.
71.  ((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
72.  ((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
73.
74.  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
75.  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
76.  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd11) & (cnt1==16'd0)) ? 1'b1 :
77.  txd ;
78.  end
79.  wire                   done = s_idle;
```

➔ *txd* ➔ *[ouput , 1bit , Uart_txd : Transmitter bit]*

➔ *txd*
➔ *s_idle*
➔ *cnt2*
➔ *cnt1*
➔ *tdata*
➔ *parity_sel*
➔ *stop_sel*
➔ *Parity*

➔ *tdata* ➔ *[input , 8bit , send data]*

➔ *parity_sel , // 0 : none, 1 : even, 2 : odd*
➔ *stop_sel , // 0 : 1bits, 1 : 2bits*

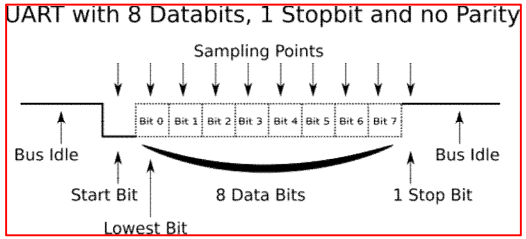
➔ *stop_sel* ➔ *[input , stop bits 설정 , 0:1bit , 1:2bit]*

➔ *done* ➔ *out [0] : 송신 완료 flag [1 : idle 상태 or 송신 완료] , [0 : 송신 중]*

✓ 51 ~ 78 : 전송되는 비트 txd 를 생성

➔ *txd* 는 s_idle 이 아니면 조건에 따라 start bits을 내보내거나 tdata를 내보냄

➔ 조건 : cnt1이 16'd0일 때 cnt2의 카운트에 맞게 tdata[8bit(send data, start bit 포함 =)] 각 비트 정보(tdata)



Next Slide

✓ 79 : done 을 생성

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation ➔ UART_TX ➔ txd 정의

Next Slide

Uart_Tx . v (4)

```
51.  reg                    txd;
52.  always @(posedge mclk or ne
53.  begin
54.          if(~reset)  txd <= 1'
55.          else
56.                  ((cnt2==4
57.                  ((cnt2==4
58.                  ((cnt2==4
59.                  ((cnt2==4
60.                  ((cnt2==4
61.                  ((cnt2==4
62.                  ((cnt2==4
63.                  ((cnt2==4'b7) && (cnt1==16'd0)) ? tdata[6] :
64.                  ((cnt2==4'd8) && (cnt1==16'd0)) ? tdata[7] :
65.
66.                  ((parity_sel==2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :
67.
68.                  ((parity_sel==2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :
69.                  ((parity_sel==2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
70.
71.                  ((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
72.                  ((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
73.
74.                  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
75.                  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
76.                  ((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd11) & (cnt1==16'd0)) ? 1'b1 :
77.                  txd ;
78.  end

79.  wire                   done = s_idle;
```

➔ *txd*
➔ *s_idle*
➔ *cnt2*
➔ *cnt1*
➔ *tdata*
➔ *parity_sel*
➔ *stop_sel*
➔ *Parity*

[표] Parity 값과 stop 값에 따라 송신되는 bit 구성

Parity	Stop	Start	Data field	Parity	Stop-1	Stop-2
0	1	0	1 ~ 8	-	9	-
0	2	0	1 ~ 8	-	9	10
1 or 2(!=0)	1	0	1 ~ 8	*9*	10	-
1 or 2(!=0)	2	0	1 ~ 8	*9*	10	11

➔ *tdata ➔ [input , 8bit , send data]*

➔ *stop_sel ➔ [input , stop bits 설정 , 0:1bit , 1:2bit]*

➔ *done ➔ out [0] : 송신 완료 flag [1 : idle 상태 or 송신 완료] , [0 : 송신 중]*

✓ 51 ~ 78 : 전송되는 비트 txd 를 생성

➔txd 는 s_idle 이 아니면 조건에 따라 start bits을 내보내거나 tdata를 내보냄

➔조건 : cnt1이 16'd0일 때 cnt2의 카운트에 맞게 tdata(8bit, send data)각 비트 정보(tdata)

✓ 79 : done 을 생성

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation ➔ UART_TX ➔ txd 정의

[표] Parity 값과 stop 값에 따라 송신되는 비트 구성

Parity	Stop	Start	Data field	*Parity*	Stop-1	Stop-2
0	1	0	1 ~ 8	-	9	-
0	2	0	1 ~ 8	-	9	10
1 or 2(!=0)	1	0	1 ~ 8	*9*	10	-
1 or 2(!=0)	2	0	1 ~ 8	*9*	10	11

```
51.  reg
52.  alw
53.  beg
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.  end
79.  wire                    done = s_idle;
```

➔ *stop_sel ➔ [input , stop bits 설정 , 0:1bit , 1:2bit]*

➔ *parity_sel ➔ in [1:0] : parity 설정 [0 : none, 1 : even, 2 : odd]*

```
((parity_sel==2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :

((parity_sel==2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? 1'b1 :
((parity_sel==2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :

((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
((parity_sel!=2'b0) & ~stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :

((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd9 ) & (cnt1==16'd0)) ? parity :
((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd10) & (cnt1==16'd0)) ? 1'b1 :
((parity_sel!=2'b0) &  stop_sel & (cnt2==4'd11) & (cnt1==16'd0)) ? 1'b1 :
txd ;
```

✓ 51 ~ 78 : 전송되는 비트 txd 를 생성

➔txd 는 s_idle 이 아니면 조건에 따라 start bits을 내보내거나 tdata를 내보냄

➔조건 : cnt1이 16'd0일 때 cnt2의 카운트에 맞게 tdata(8bit, send data)각 비트 정보(tdata)

✓ 79 : done 을 생성

UART Controller Implementation

➢ Uart_Tx.xpr

➢ UART TX Implementation ➔ Code Implementation ➔ UART TX ➔ 상태 천이 구현(m_state)

Uart_Tx . v (5)

✓ 82 ~ 92 : m_state 의 상태 천이 구현

```
80.  // ----------------------------------------
81.  // 4) state transition : m_state
82.  always @(posedge mclk or negedge reset)
83.  begin
84.          if(~reset)      m_state <= 1'b0;
85.          else            m_state <= (s_idle  & send) ? M_RUN :
86.
87.    (s_run & (parity_sel==2'b0) & (stop_sel==1'b0) & (cnt2==4'd10)) ? M_IDLE :
88.    (s_run & (parity_sel==2'b0) & (stop_sel==1'b1) & (cnt2==4'd11)) ? M_IDLE :
89.    (s_run & (parity_sel!=2'b0) & (stop_sel==1'b0) & (cnt2==4'd11)) ? M_IDLE :
90.    (s_run & (parity_sel!=2'b0) & (stop_sel==1'b1) & (cnt2==4'd12)) ? M_IDLE :
91.    m_state ;
92.  en
93.  en
```

Parity	Stop	Start	Data field	Parity	Stop-1	Stop-2	IDLE
0	1	0	1 ~ 8	-	9	-	10
0	2	0	1 ~ 8	-	9	10	11
1 or 2(!=0)	1	0	1 ~ 8	9	10	-	11
1 or 2(!=0)	2	0	1 ~ 8	9	10	11	12

✓ s_idle =‘1’ 이고 send=‘1’ 이면 M_RUN(데이터 전송 상태)

✓ s_run=‘1’ 상태에서 천이 조건 구현

➔ m_state
➔ s_run
➔ parity_sel
➔ stop_sel
➔ cnt2
➔ M_IDLE

Uart_Tx . v (2)

```
21.  // ----------------------------------------
22.  // 1) define state
23.  reg                m_state ;
24.  parameter           M_IDLE    = 1'b0 ;
25.  parameter           M_RUN     = 1'b1 ;
26.  // ----------------------------------------
27.  // 2) state flag
28.  wire                s_idle  = (m_state == M_IDLE ) ? 1'b1 : 1'b0 ;
29.  wire                s_run   = (m_state == M_RUN  ) ? 1'b1 : 1'b0 ;
```

➔ send flag

✓ 23 ~ 25 : state 정의

➔ IDLE, RUN 2개로 정의 : IDLE 상태에서 send flag가 active 되면 RUN 상태가 되면서 데이터를 전송

➔ 전송이 완료되면 done flag를 enable 하고 IDLE 모드

✓ 28 ~ 29 : state flag 정의

➔ m_state == 1’b0 이면 ➔ s_idle = 1’b1 , 아니면 1’b0

➔ m_state == 1’b1 이면 ➔ s_run = 1’b1 , 아니면 1’b0

UART Controller Implementation

- **UART TX** Implementation
  1. Port 정의
  2. State 정의
  3. Code Implementation
  4. Test Bench

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation ➔ UART_TX ➔ Simulation 검증

Uart_Tx _tb. v (1)

```
1.  `timescale 1ns / 1ps

2.  module Uart_Tx_tb();

3.  reg                 reset, mclk;

4.  initial  begin
5.           reset = 0;
6.           mclk = 0;
7.
8.  #10000 reset = 1;
9.  end

10. always  #5          mclk = ~mclk;          // 100 Mhz

11. reg      [9:0]      cnt;
12. always @(posedge mclk or negedge reset)
13. begin
14.          if(~reset)     cnt <= 10'b0;
15.          else           cnt <= (cnt==10'd1023) ? 10'd1023 : cnt+1'b1;
16. end
```

✓ 3 ~ 10 : reset, mclk 생성

✓ 11 ~ 16 : send 신호를 생성하기 위한 counter(cnt, 10bit) 생성 [10'd1023 = 10'h3ff = 10'b0011_1111_1111]

UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation ➔ UART_TX ➔ Simulation 검증

Uart_Tx _tb. v (2)

```
17.  wire    [15:0]     baudrate  = 16'd868;  // [100M/868 =115,200 ]
18.  wire    [7:0]      tdata     = 8'h55;  // 8'b0101_0101
19.  wire    [1:0]      parity_sel = 2'b01;  // even
20.  wire               stop_sel  = 1'b1 ;   // 2bit

21.  reg                send;
22.  always @(posedge mclk or negedge reset)
23.  begin
24.          if(~reset)     send <= 1'b0;
25.          else           send <= (cnt==10'd1000) ? 1'b1 : 1'b0;
26.  end

27.  Uart_Tx            Uart_Tx_U0(
28.          .reset     (reset     ),
29.          .mclk      (mclk      ),
30.          .baudrate  (baudrate  ),
31.          .parity_sel (parity_sel ),
32.          .stop_sel  (stop_sel  ),
33.          .tdata     (tdata     ),
34.          .send      (send      ),
35.          .txd       (txd       ),
36.          .done      (done      )
37.  );
38.  endmodule
```

baudrate	수식 값	정수값으로 변환	baudrate2	오차율
2400	41666.67	41667	2400.0	0.00%
4800	20833.33	20833	4800.1	0.00%
19200	5208.33	5208	19201.2	0.01%
38400	2604.17	2604	38402.5	0.01%
57600	1736.11	1736	57603.7	0.01%
76800	1302.08	1302	76804.9	0.01%
115200	868.06	868	115207.4	0.01%
230400	434.03	434	230414.7	0.01%
460800	217.01	214	467289.7	1.41%
921600	108.51	109	917431.2	-0.45%

✓ 17 ~ 20 : Uart_Tx 입력 신호

➔ baudrate : 115,200

➔ tdata(송신 데이터) : 0x55 (8'b0101_0101)

➔ parity : even

➔ stop bit : '1' ➔ 2bit

✓ 21 ~ 26 : send flag 생성 [10'd1000 = 10'h3e8 = 10'b0011_1110_1000]

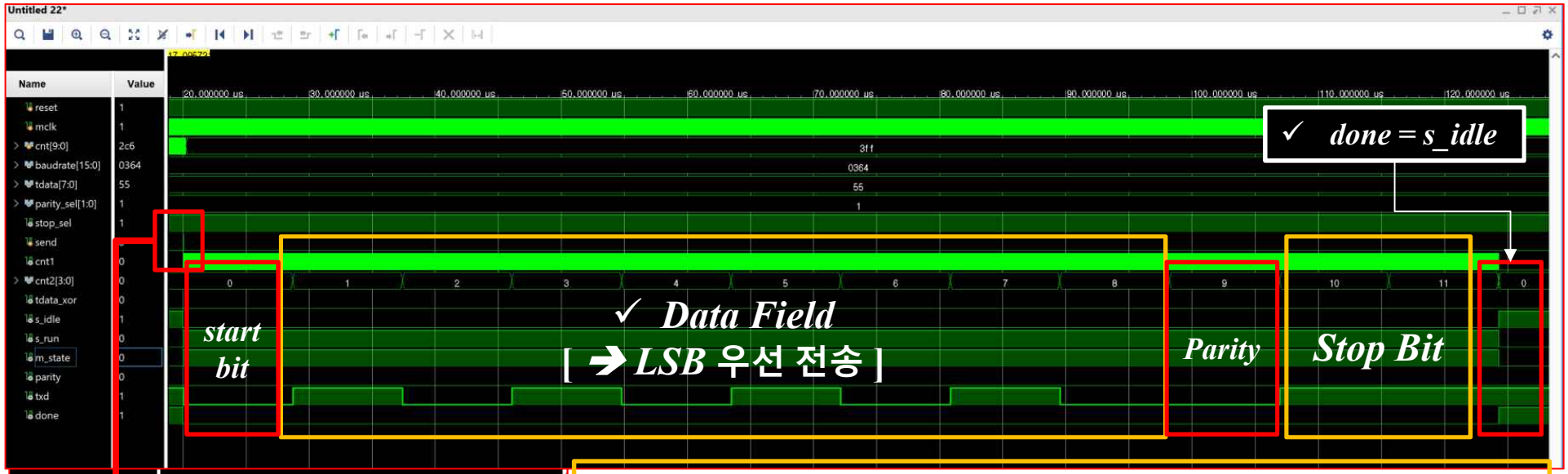
✓ 44 ~ 50 : Uart_Tx module

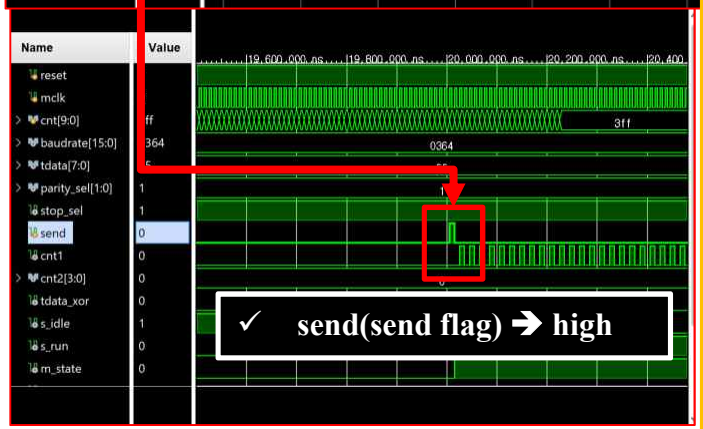
UART Controller Implementation

➢ *Uart_Tx.xpr*

➢ *UART TX Implementation* ➔ Code Implementation

❖ UART_TX ➔ Simulation 검증 ➔ Uart_Tx : Top Module 지정 ➔ Run Simulation





- ✓ reset ➔ in [0] : reset, Active Low
- ✓ mclk ➔ in [0] : clock, 100Mhz
- ✓ baudrate ➔ in [15:0] : baudrate 설정
- ✓ parity_sel ➔ in [1:0] : parity 설정 [0 : none, 1 : even, 2 : odd]
- ✓ stop_sel ➔ in [0] : stop bits 설정 [0 : 1bit, 1 : 2bits]
- ✓ tdata ➔ in [7:0] : send data
- ✓ send ➔ in [0] : send flag ➔ H 일 때 전송을 시작 : pulse 입력
- ✓ txd ➔ out [0] : uart_txd
- ✓ done ➔ out [0] : 송신 완료 flag [1 : idle 상태 or 송신 완료] , [0 : 송신 중]

UART Controller Implementation

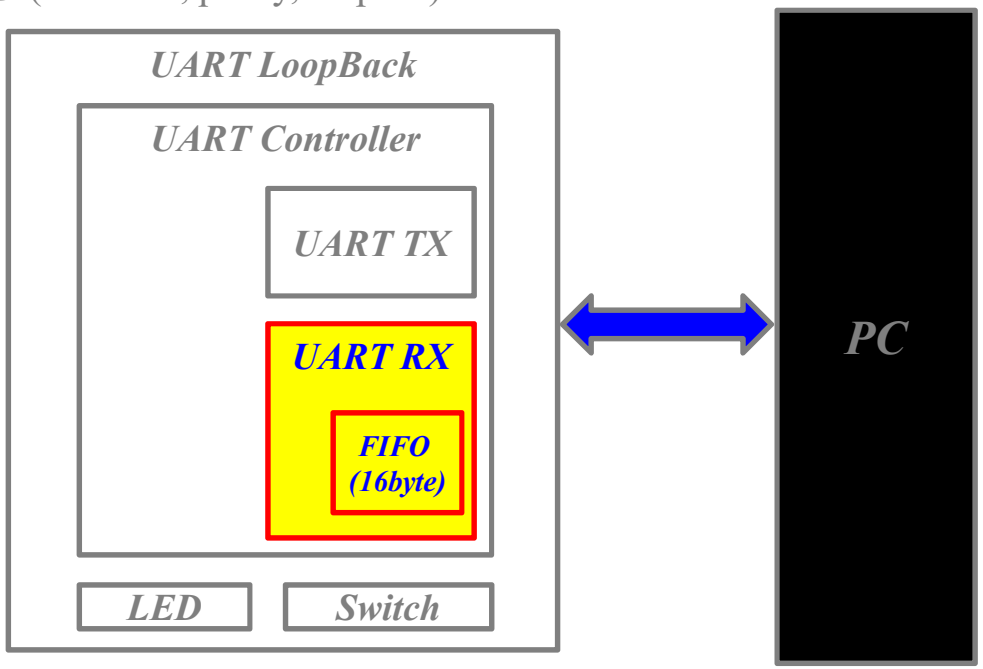
- *UART TX Implementation*
- *FIFO_16x8 Implementation*
- *UART RX Implementation*
- *UART Controller Implementation*
- *UART Loopback Implementation*

UART Controller Implementation

➢ Uart_Rx.xpr

➢ UART Controller Implementation

- **UART Controller System Block**
  - ❖ UART TX : Uart Tx Module
  - ❖ **UART RX : Uart Rx Module, 16bytes FIFO 포함**
  - ❖ UART Controller : UART Tx/Rx Controller
  - ❖ UART LoopBack : LoopBack Test, PC로부터 데이터를 수신하면 그대로 PC로 전송
  - ❖ Led : 상태를 표시함, Tx available, Rx available, overrun error, frame error, parity error
  - ❖ SW : 모드 설정 (baudrate, parity, stop bit)



UART Controller Implementation

- *FIFO_16x8 Implementation*

UART Controller Implementation

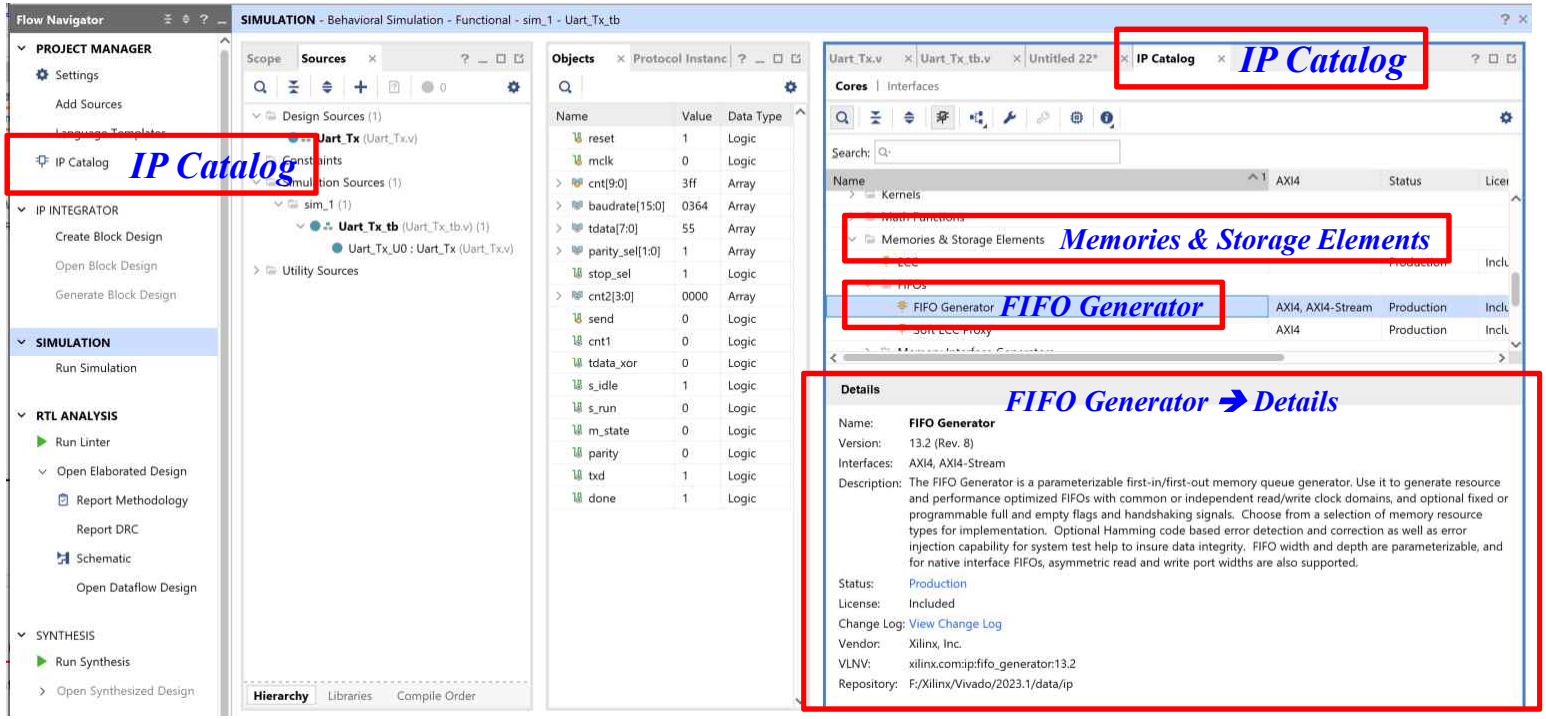
➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함
- 데이터가 수신되었을 때, 바로 데이터를 읽지 않으면 그 다음 데이터가 들어오면 이전 데이터를 잃어버리게 되는데, 이를 방지하기 위하여 내부에 FIFO를 가지고 있음

➔ UART Controller들이 16바이트 FIFO를 가지고 있기 때문에 16바이트 FIFO 구현 실습

➔ FIFO 생성 : [PROJECT MANAGER] ➔ [IP Catalog] 클릭 ➔ [Memories & Storage Elements] ➔ [FIFOs] ➔ [FIFO Generator] 더블 클릭 ➔ [FIFO Generator] 실행



UART Controller Implementation

➢ *Uart_Rx.xpr*

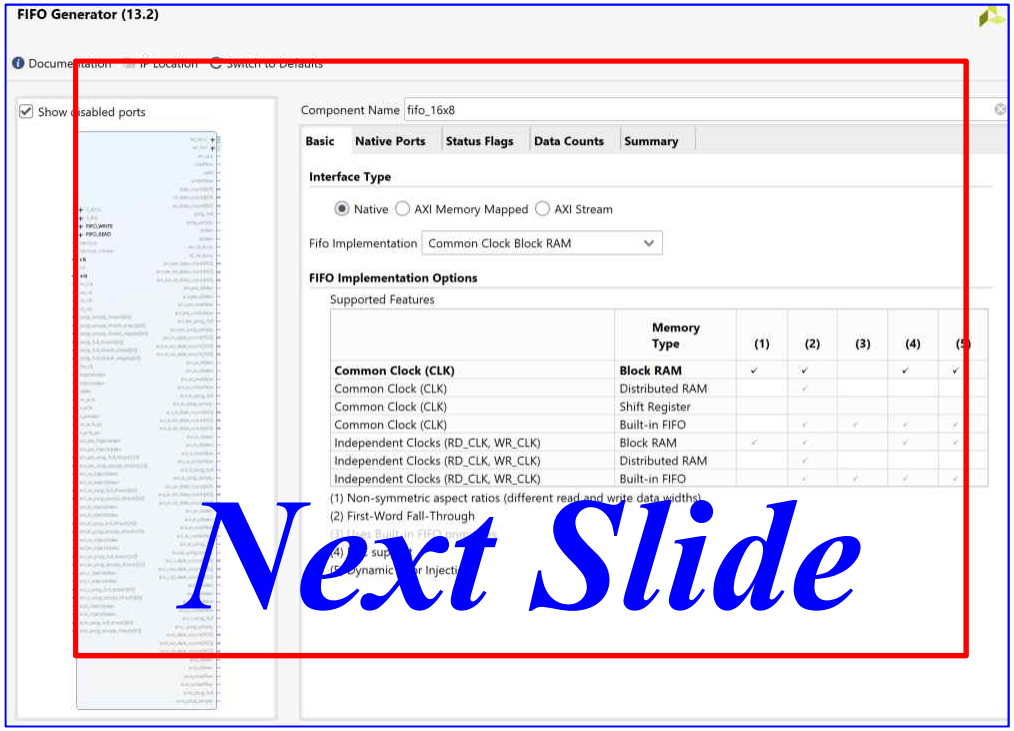
➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- *Uart Rx* 모듈을 구현 ➔ *Uart Rx* 모듈은 내부에 *16*바이트 *FIFO*를 포함

➔ FIFO 생성 : [PROJECT MANAGER] ➔ [IP Catalog] 클릭 ➔ [Memories & Storage Elements] ➔ [FIFOs] ➔ [FIFO Generator] 더블 클릭 ➔ [FIFO Generator] 실행

➔ Component Name : *fifo_16x8* 로 입력 ➔[***Basic***]

➔ *Interface Type : Native* ➔ *FIFO Implementation Option* : *Common Clock Block RAM* 을 선택 : *read/write* 동일 *clock*을 사용



Next Slide

UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ **FIFO Generator (13.2)**

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: fifo_16x8 — *fifo_16x8*

Basic | Native Ports | Status Flags | Data Counts | Summary

Interface Type

Native — (•) Native ( ) AXI Memory Mapped ( ) AXI Stream

Fifo Implementation: Common Clock Block RAM — *Common Clock Block RAM*

FIFO Implementation Options

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	**Block RAM**	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection

Show disabled ports

FIFO_WRITE
FIFO_READ
clk
srst

UART Controller Implementation

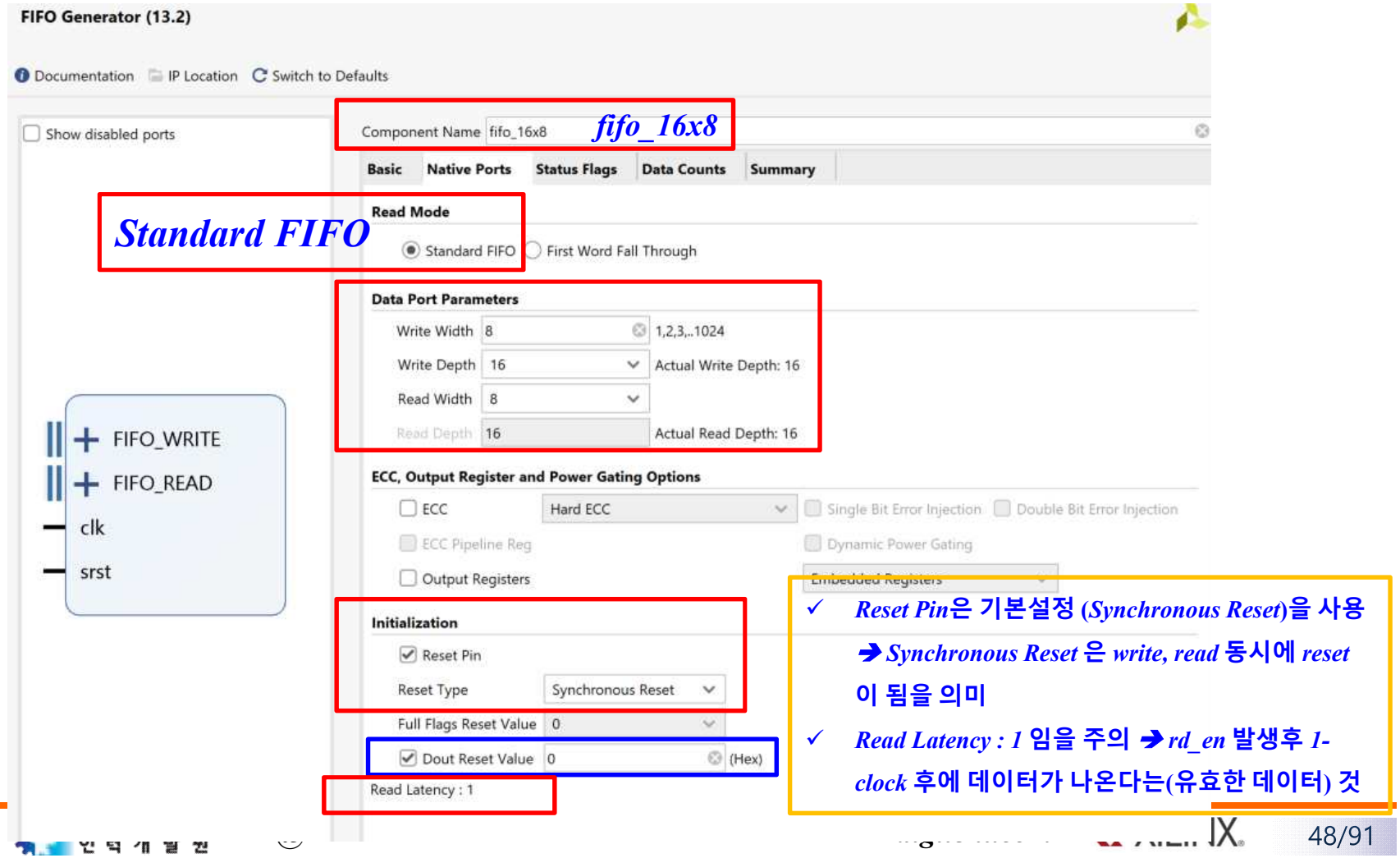
➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함

➔ Component Name : fifo_16x8 로 입력 ➔[***Native Ports***]

➔ ***[Read Mode : Standard FIFO], [Write Wdith : 8], [Write Depth : 16], [Read Width : 8]*** ➔ 입력/선



✓ ***Reset Pin*은 기본설정 (*Synchronous Reset*)을 사용 ➔ *Synchronous Reset* 은 *write, read* 동시에 *reset* 이 됨을 의미**

✓ ***Read Latency : 1* 임을 주의 ➔*rd_en* 발생후 *1-clock* 후에 데이터가 나온다는(유효한 데이터) 것**

UART Controller Implementation

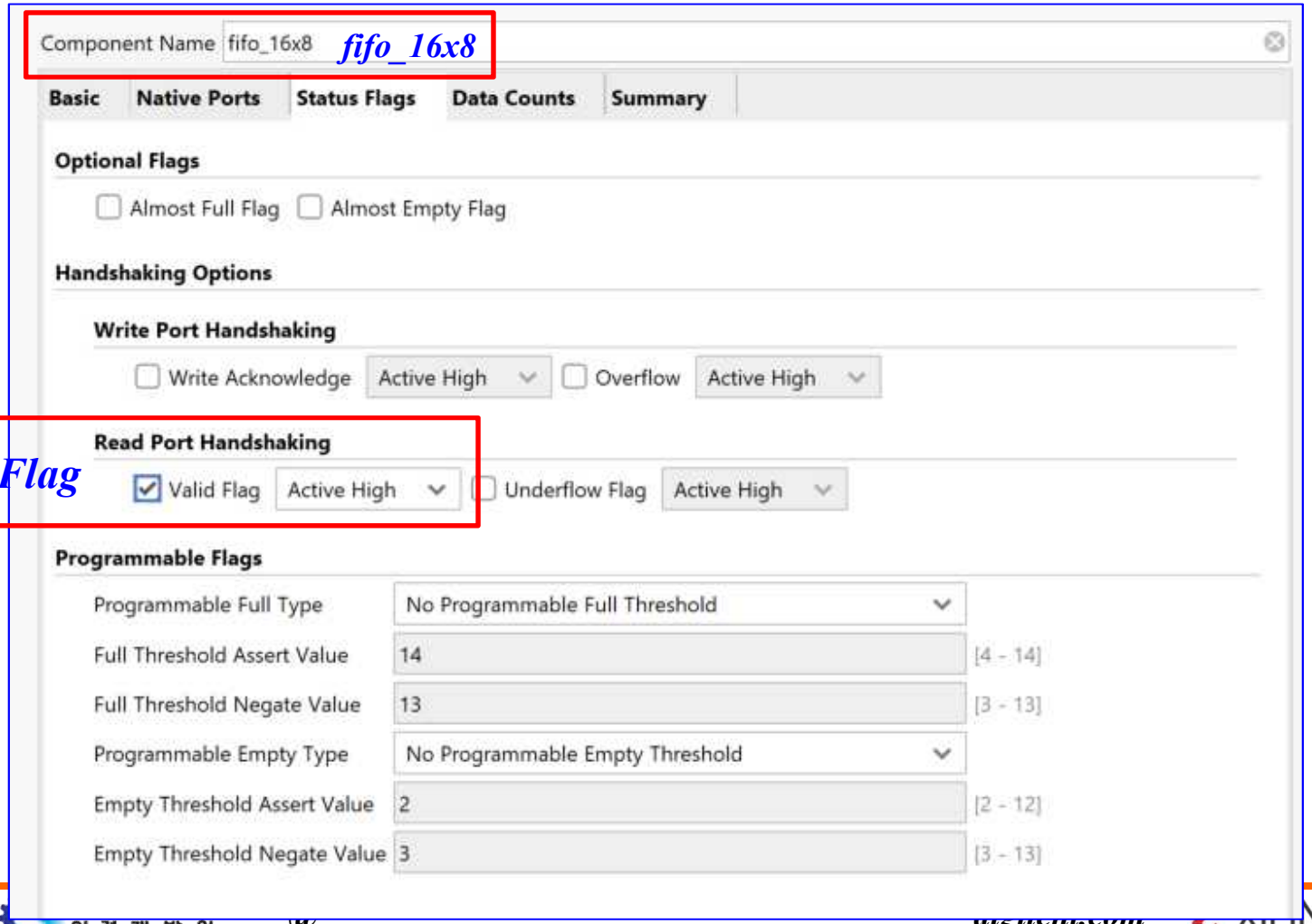
➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함

➔ Component Name : fifo_16x8 로 입력 ➔[***Status Flags***]

➔ ***Read Port Handshaking*** ➔ ***Valid Flag : Check*** ➔ Valid Flag는 FIFO에 Read 할 데이터가 남아 있음을 알려줌



Valid Flag

UART Controller Implementation

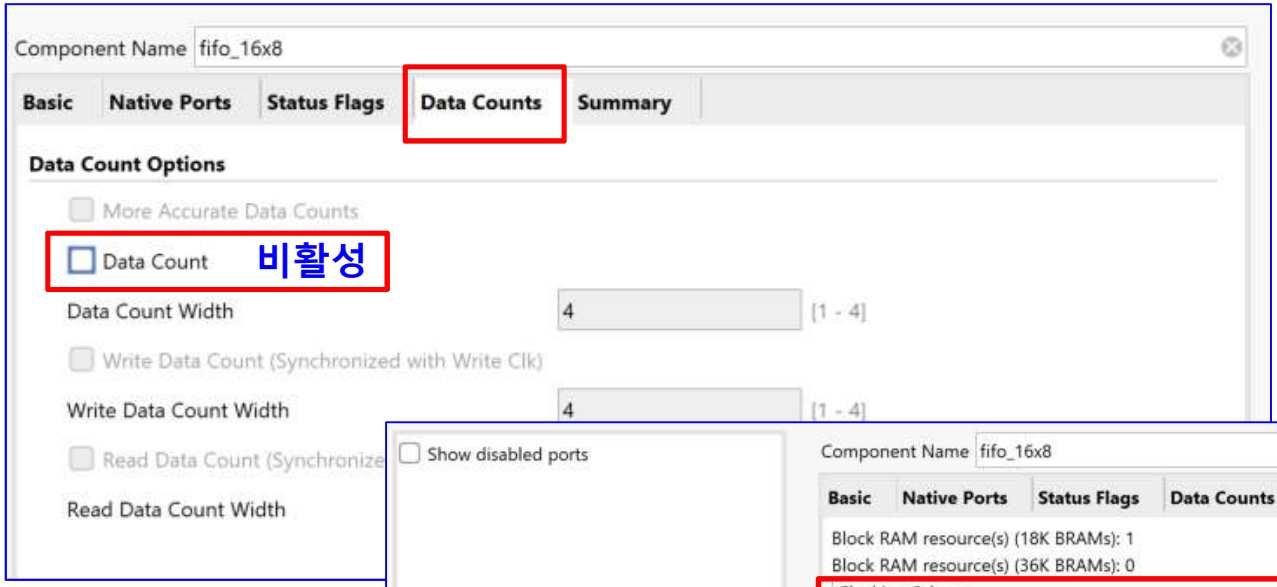
➢ *Uart_Rx.xpr*

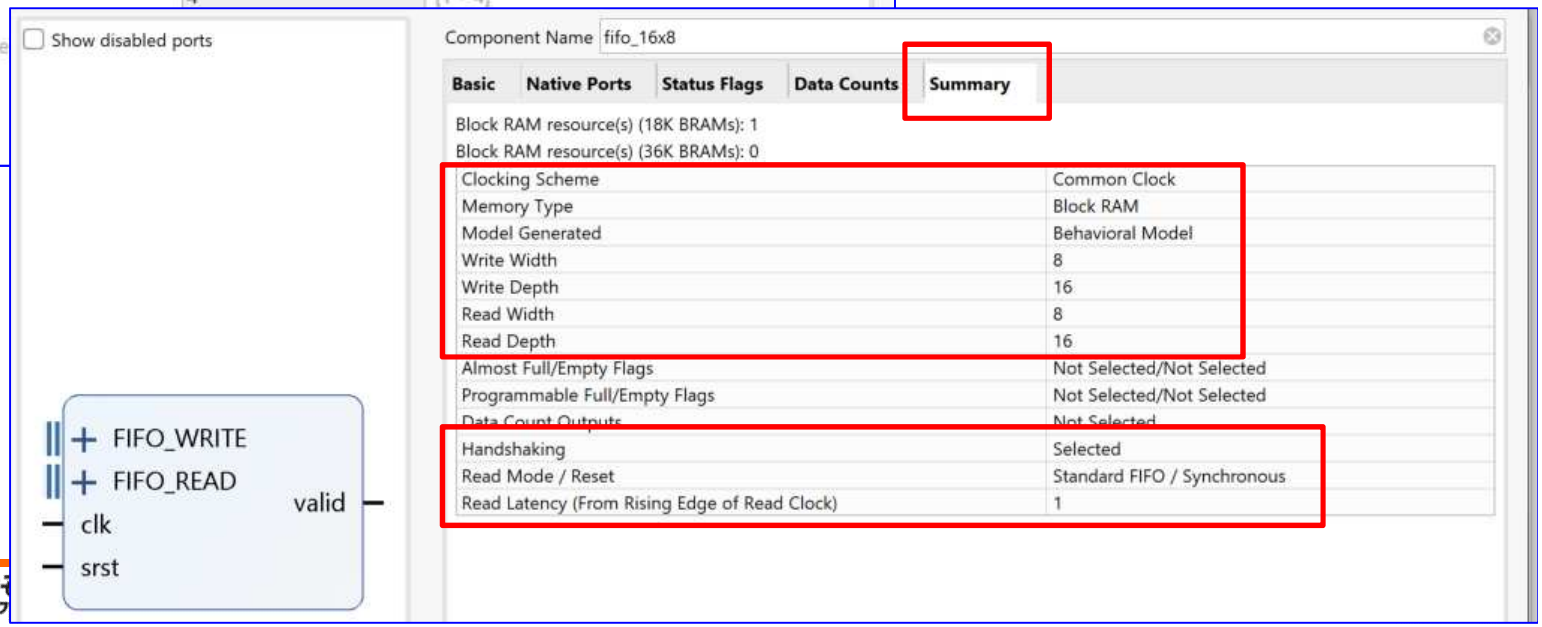
➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함

➔ Component Name : fifo_16x8 로 입력 ➔[***Data Counts***] : Default , [***Summary***]

➔ Data Count를 Check 하면 현재 몇 바이트가 FIFO에 남아있는지를 알려주는 옵션





UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함

➔ Component Name : fifo_16x8 로 입력 ➔생성된 [***FIFO(16 byte)***]

fifo_16x8.veo

```
1.  //---------- Begin Cut here for INSTANTIATION Template ---
2.  // INST_TAG
3.  fifo_16x8 your_instance_name (
4.    .clk(clk),      // input wire clk
5.    .srst(srst),    // input wire srst(synchronous reset)
6.    .din(din),      // input wire [7 : 0] din
7.    .wr_en(wr_en),  // input wire wr_en
8.    .rd_en(rd_en),  // input wire rd_en
9.    .dout(dout),    // output wire [7 : 0] dout
10.   .full(full),    // output wire full
11.   .empty(empty),  // output wire empty
12.   .valid(valid)  // output wire valid
13. );
```

UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ UART_RX ➔ ***FIFO(16 byte) Implementation***

- Uart Rx 모듈을 구현 ➔ Uart Rx 모듈은 내부에 16바이트 FIFO를 포함

➔ Component Name : fifo_16x8 로 입력 ➔생성된 **[*FIFO(16 byte)*]**

fifo_16x8.veo

```
1.  //---------- Begin Cut here for
    INSTANTIATION Template ---// INST_TAG
2.  fifo_16x8 your_instance_name (
3.    .clk(clk),      // input wire clk
4.    .srst(srst),    // input wire srst
5.    .din(din),      // input wire [7 : 0] din
6.    .wr_en(wr_en),  // input wire wr_en
7.    .rd_en(rd_en),  // input wire rd_en
8.    .dout(dout),    // output wire [7 : 0] dout
9.    .full(full),    // output wire full
10.   .empty(empty),  // output wire empty
11.   .valid(valid)  // output wire valid
12. );
```

- ✓ ***full*** : *fifo*가 full 이 되어서 더이상 데이터를 write 할 수 없음을 알려주는 flag [Active High]
- ✓ ***din*** : write data input
- ✓ ***wr_en*** : write strobe [Active High]
- ✓ ***empty*** : *fifo* 가 비어 있음을 알려주는 flag [Active High]
- ✓ ***dout*** : read data output
- ✓ ***rd_en*** : read strobe, Active High
- ✓ ***valid*** : *fifo*에 읽지 않은 데이터가 남아 있음을 알려주는 flag, [Active High]
- ✓ ***clk*** : clock
- ✓ ***srst*** : reset [Synchronous Reset , Active High]

UART Controller Implementation

- **UART RX** Implementation
  1. Port 정의
  2. State 정의
  3. Code Implementation
  4. State Transition
  5. Test Bench

UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ Code Implementation : ***UART_RX*** ➔ ***input & out port*** 정의

- ***Uart_Rx Module*** 은 rxd을 통하여 데이터가 수신되면 : ***rvalid flag*** ➔ ***active high(1)***
  - ✓ 상위 모듈에서는 *rvalid flag*확인 ➔ 만일 *rvalid flag*가 *active(1)* 되면 수신 데이터가 있다는 것을 의미 ➔이때 *ren*을 " *1* "로 만들어서 *rdata*를 통하여 수신된 데이터를 read ➔ rvalid가 low가 될 때까지 데이터 read.
  - ✓ rvalid가 low가 되면 더이상 읽을 데이터가 없음을 나타내기 때문에 ➔ ren : low

signal	in/out	size	description
reset	in	[0]	*reset, Active Low*
mclk	in	[0]	*clock, 100Mhz*
baudrate	in	[15:0]	*baudrate 설정*
parity_sel	in	[1:0]	*parity 설정 [0 : none, 1 : even, 2 : odd]*
stop_sel	in	[0]	*stop bits 설정 [0 : 1bit, 1 : 2bits]*
ren	in	[0]	*fifo read strobe*
rdata	*out*	[7:0]	fifo read data output
rvalid	*out*	[0]	fifo read valid, fifo 에 읽을 데이터가 있음을 알려줌, active high
overrun	*out*	[0]	*fifo full error, active high*
frame_err	*out*	[0]	*stop bit error, active high*
parity_err	*out*	[0]	*parity bit error, active high*
rxd	*in*	[0]	*uart 수신 데이터 RXD (Input Data)*

UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ Code Implementation ➔ UART_RX ➔ ***input & out port***를 정의

uart_rx. v (1)

```
1.  module uart_rx(
2.              reset , mclk , baudrate ,
3.              parity_sel ,
4.              // 0 : none,  1 : even, 2 : odd
5.              stop_sel  ,   // 0 : 1bits, 1 : 2bits
6.              ren       ,
7.              rdata     ,
8.              rvalid    ,   // rdata valid flag
9.              overrun   ,   // rx fifo full
10.             frame_err ,// stop bit error
11.             parity_err ,  // parity error
12.             rxd
13.);
14.input                    reset     ;
15.input                    mclk      ;
16.input    [15:0]          baudrate  ;
17.input    [1:0]           parity_sel ;
18.input                    stop_sel  ;
19.input                    ren       ;
20.output   [7:0]           rdata     ;
21.output                   rvalid    ;
22.output                   overrun   ;
23.output                   frame_err ;
24.output                   parity_err ;
25.input                    rxd       ;
```

uart_rx. v (2)

```
26.  // 1) define state
27.  reg      [1:0]          m_state ;
28.  parameter               M_IDLE      = 2'b0 ;
29.  parameter               M_RECEIVE   = 2'd1 ;
30.  parameter               M_DONE      = 2'd2 ;

31.  // 2) state flag
32.  wire     s_idle    = (m_state == M_IDLE ) ? 1'b1 : 1'b0 ;
33.  wire     s_receive = (m_state == M_RECEIVE ) ? 1'b1 : 1'b0 ;
34.  wire     s_done    = (m_state == M_DONE ) ? 1'b1 : 1'b0 ;
```

✓ 1 ~ 25 : *port* 선언

✓ 26 ~ 30 : *state*를 정의

➔ *IDLE, RECEIVE, DONE* 3개의 *state*

➔ *IDLE* 상태에서 *rxd*를 통하여 데이터가 들어오면 *RECEIVE* 상태가 되어 데이터를 수신 ➔ 데이터 수신이 완료되면 DONE 상태

➔ *DONE* 상태에서는 *frame_error, parity_error*를 *Check* 하고, 또 수신된 데이터를 *fifo*에 저장

➔ *DONE*상태 에서의 일이 끝나면 다시 IDLE 상태

✓ 31 ~ 34 : *state flag*를 정의

UART Controller Implementation

➢ ***UART RX*** *Implementation*

1. *Port 정의*
2. *State 정의*
3. *Code Implementation*
4. *State Transition*
5. *Test Bench*

UART Controller Implementation

➢ Uart_Rx.xpr

➢ UART RX Implementation ➔ Code Implementation ➔ UART_RX ➔ **code implementation**

uart_rx. v (3)

```
35.  // 3) code implementation
36.  reg                 rxd_1d, rxd_2d, rxd_3d;
37.  wire                rxd_nedge = ~rxd_2d & rxd_3d;
38.  always @(posedge mclk or negedge reset)
39.  begin
40.          if(~reset)  begin
41.                      rxd_1d <= 1'b1;
42.                      rxd_2d <= 1'b1;
43.                      rxd_3d <= 1'b1;
44.          end
45.          else        begin
46.                      rxd_1d <= rxd;
47.                      rxd_2d <= rxd_1d;
48.                      rxd_3d <= rxd_2d;
49.          end
50.  end
```

- ✓ 36 ~ 50 : rxd input 데이터의 negative edge를 생성 ➔ 수신은 rxd(input)의 negative edge(Start bit) 부터 시작
- ✓ 51 ~ 56 : baudrate에 따라서 각 bit의 duration을 구함
- ✓ 57 ~ 62 : 수신되는 각 비트의 counter 를 생성

```
51.  reg      [15:0]       cnt1;
52.  always @(posedge mclk or negedge reset)
53.  begin
54.          if(~reset)    cnt1 <= 16'b0;
55.          else          cnt1 <= ~s_receive ? 16'b0 : (cnt1==baudrate) ? 16'b0 : cnt1+1'b1;
56.  end
```

- ✓ 51 ~ 56 : baudrate에 따라서 각 bit의 duration을 구함

```
57.  reg      [3:0]        cnt2;
58.  always @(posedge mclk or negedge reset)
59.  begin
60.          if(~reset)    cnt2 <= 4'b0;
61.          else          cnt2 <= ~s_receive ? 4'b0 : (cnt1==baudrate) ? cnt2+1'b1 : cnt2;
62.  end
```

- ✓ 57 ~ 62 : 수신되는 각 비트의 counter 를 생성

UART Controller Implementation

➢ Uart_Rx.xpr

➢ UART RX Implementation ➔ Code Implementation ➔ UART_RX ➔ code implementation

uart_rx. v (4)

```
63.  reg      [7:0]        rxd_data;
64.  always @(posedge mclk or negedge reset)
65.  begin
66.          if(~reset)    rxd_data <= 8'b0;
67.          else          begin
68.                        rxd_data[0] <= ((cnt2==4'd1) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[0];
69.                        rxd_data[1] <= ((cnt2==4'd2) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[1];
70.                        rxd_data[2] <= ((cnt2==4'd3) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[2];
71.                        rxd_data[3] <= ((cnt2==4'd4) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[3];
72.                        rxd_data[4] <= ((cnt2==4'd5) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[4];
73.                        rxd_data[5] <= ((cnt2==4'd6) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[5];
74.                        rxd_data[6] <= ((cnt2==4'd7) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[6];
75.                        rxd_data[7] <= ((cnt2==4'd8) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_data[7];
76.          end
77.  end
```

✓ 63 ~ 77 : rxd에서 수신되는 data 값을 구함 ➔ 데이터를 읽는 point는 bit의 중간 위치 : 즉 cnt1의 값이 buadrate/2가 되는 지점

```
78.  wire                  cal_parity =
     rxd_data[0]^rxd_data[1]^rxd_data[2]^rxd_data[3]^rxd_data[4]^rxd_data[5]^rxd_data[6]^rxd_data[7];

79.  reg                   cal_parity2;
80.  always @(posedge mclk or negedge reset)
81.  begin
82.          if(~reset)    cal_parity2 <= 1'b0;
83.          else          cal_parity2 <= (parity_sel==2'b01) ? cal_parity : (parity_sel==2'b10) ? ~cal_parity : 1'b0 ;
84.
85.  end
```

```
86.  reg                   rxd_parity;
87.  always @(posedge mclk or negedge reset)
88.  begin
89.          if(~reset)    rxd_parity <= 1'b0;
90.          else          rxd_parity <= ((cnt2==4'd9) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d : rxd_parity;
91.  end
```

UART Controller I

➢ *UART RX Implementation* ➔ Code Imp

Parity	Stop	Start	Data field	Parity	Stop-1	Stop-2
0	1	0	1 ~ 8	-	9	-
0	2	0	1 ~ 8	-	9	10
1 or 2(!=0)	1	0	1 ~ 8	9	10	-
1 or 2(!=0)	2	0	1 ~ 8	9	10	11

```
92.  reg                  stop_bit1;
93.  always @(posedge mclk or negedge reset)
94.  begin
95.  if(~reset)  stop_bit1 <= 1'b0;
96.  else       stop_bit1 <= ((parity_sel==2'b00) & (cnt2==4'd9 ) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d :
97.                          ((parity_sel!=2'b00) & (cnt2==4'd10) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d :
98.                           stop_bit1 ;
99.  end
```

✓ 92 ~ 99 : 수신된 데이터 에서 stop_bit1을구함

```
100. reg                  stop_bit2;
101. always @(posedge mclk or negedge reset)
102. begin
103. if(~reset)  stop_bit2 <= 1'b0;
104. else       stop_bit2 <= ((parity_sel==2'b00) & (cnt2==4'd10) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d :
105.                         ((parity_sel!=2'b00) & (cnt2==4'd11) & (cnt1=={1'b0, baudrate[15:1]})) ? rxd_3d :
106.                          stop_bit2 ;
107. end
```

✓ 100 ~ 107 : 수신된 데이터 에서 stop_bit2를 구함

```
108. reg      [2:0]        cnt_done;
109. always @(posedge mclk or negedge reset)
110. begin
111.          if(~reset)   cnt_done <= 3'b0;
112.          else         cnt_done <= ~s_done ? 3'b0 : cnt_done+1'b1;
113. end
```

✓ 108 ~ 113 : DONE 상태에서 사용하는 counter를 생성

UART Controller Implementation

- **UART RX** Implementation
  1. Port 정의
  2. State 정의
  3. Code Implementation
  4. State Transition
  5. Test Bench

UART Controller Implem

➢ *UART RX Implementation* ➔ Code Implementation ➔ U

Parity	Stop	Start	Data field	Parity	Stop-1	Stop-2
0	1	0	1 ~ 8	-	9	-
0	2	0	1 ~ 8	-	9	10
1 or 2(!=0)	1	0	1 ~ 8	9	10	-
1 or 2(!=0)	2	0	1 ~ 8	9	10	11

✓ 114 ~ 124 : 상태 전이를 구현

```
114. // 4) state transition
115. always @(posedge mclk or negedge reset)
116. begin
117.         if(~reset)     m_state <= 1'b0;
118.         elss           m_state <= ( s_idle    & rxd_nedge ) ? M_RECEIVE :
119.                        ((parity_sel==2'b0) & (stop_sel==1'b0) & (cnt2==4'd9 ) & (cnt1==baudrate)) ? M_DONE    :
120.                        ((parity_sel==2'b0) & (stop_sel==1'b1) & (cnt2==4'd10) & (cnt1==baudrate)) ? M_DONE    :
121.                        ((parity_sel!=2'b0) & (stop_sel==1'b0) & (cnt2==4'd10) & (cnt1==baudrate)) ? M_DONE    :
122.                        ((parity_sel!=2'b0) & (stop_sel==1'b1) & (cnt2==4'd11) & (cnt1==baudrate)) ? M_DONE    :
123.                        ( cnt_done==3'd7) ? M_IDLE    :  m_state ;
124. end
```

✓ 125 ~ 133 : *frame_err (stop bit error)* 구현

```
125. // 5) uart receive state & fifo
126. reg                    frame_err;
127. always @(posedge mclk or negedge reset)
128. begin
129.         if(~reset)     frame_err <= 1'b0;
130.         else           frame_err <= ((cnt_done==3'd5) & (stop_sel==1'b0)) ? ~stop_bit1  : // stop_bit1 = 1 ➔ frame_err = 0
131.                        ((cnt_done==3'd5) & (stop_sel==1'b1)) ? ~(stop_bit1&stop_bit2) : // (stop_bit1 & stop_bit2) = 1 ➔ frame_error =0
132.                        frame_err ;
133. end
```

✓ 134 ~ 141: *parity_err* 구현

```
134. reg                    parity_err;
135. always @(posedge mclk or negedge reset)
136. begin
137.         if(~reset)  parity_err <= 1'b0;
138.         else              parity_err <= (parity_sel==2'b0) ? 1'b0 :
139.                             ((cnt_done==3'd5) & (cal_parity2==rxd_parity)) ? 1'b0 :
140.                             ((cnt_done==3'd5) & (cal_parity2!=rxd_parity)) ? 1'b1 :  parity_err ;
141. end
```

UART Controller Implementation

➢ Uart_Rx.xpr

➢ UART RX Implementation ➔ Code Implementation ➔ UART_RX ➔ code implementation

uart_rx.v (7)

```
142. wire            fifo_full ;
143. wire            overrun = fifo_full;
```

✓ 142 ~ 143 : overrun (fifo_full)을 구현

```
144. wire            fifo_empty;
145. wire            rvalid = ~fifo_empty;
```

✓ 144 ~ 145 : rvalid (read available)을 구현

```
146. wire    [7:0]   fifo_din = rxd_data ;
147. wire            fifo_wen = (cnt_done==3'd2) ? 1'b1 : 1'b0;
148. wire            fifo_ren = ren ;
149. wire    [7:0]   fifo_dout;
150. wire    [7:0]   rdata = fifo_dout;
151. // wire         rvalid = valid ;
```

✓ 146 ~ 150 : fifo에 관련된 신호를 생성

```
152. fifo_16x8       rxd_fifo (
153.         .clk    (mclk     ),    // input wire clk
154.         .srst   (1'b0     ),    // input wire srst(active high)
155.         .din    (fifo_din ),    // input wire [7 : 0] din
156.         .wr_en  (fifo_wen ),    // input wire wr_en
157.         .rd_en  (fifo_ren ),    // input wire rd_en
158.         .dout   (fifo_dout),    // output wire [7 : 0] dout
159.         .full   (fifo_full),    // output wire full
160.         .empty  (fifo_empty),   // output wire empty
161.         .valid  (         )     // output wire valid
162.         //.valid  (rvalid )     // output wire valid

163. );

164. endmodule
```

✓ 151 ~ 161 : fifo_16x8 module

➔ fifo valid 신호를 사용하지 않고 fifo_empty 신호를 사용하여 rvalid 신호를 생성

➔ valid 신호 사용 가능 ➔ [wire rvalid = valid ;]

signal	in/out	size	description
ren	in	[0]	fifo read strobe
rdata	out	[7:0]	fifo read data output
rvalid	out	[0]	fifo read valid, fifo 에 읽을 데이터가 있음을 알려줌, active high
overrun	out	[0]	fifo full error, active high
rxd	in	[0]	uart 수신 데이터 RXD

UART Controller Implementation

- ***UART RX*** *Implementation*
  1. *Port 정의*
  2. *State 정의*
  3. *Code Implementation*
  4. *State Transition*
  5. *Test Bench*

UART Controller Implementation

➢ Uart_Rx.xpr

➢ UART RX Implementation ➔ Code Implementation ➔ UART_RX ➔ **Simulation Verification**

uart_rx_tb. v (1)

```
1.  `timescale 1ns / 1ps
2.  module uart_rx_tb();

3.  reg                 reset, mclk;

4.  initial     begin
5.              reset = 0;
6.              mclk = 0;
7.
8.  #10000     reset = 1;
9.  end

10. always  #5          mclk = ~mclk;  // 100 Mhz

11. reg         [19:0]  cnt;
12. always @(posedge mclk or negedge reset)
13. begin
14.             if(~reset)      cnt <= 20'b0;
15.             else            cnt <= cnt+1'b1;
16. end

17. wire        [15:0]  baudrate  = 16'd868; // 16'h364 , 115,200
18. wire        [7:0]   tdata     = 8'h55;  // 0101_0101
19. wire        [1:0]   parity_sel = 2'b01; // even
20. wire                stop_sel  = 1'b1 ;  // 2bit

21. reg                 send;
22. always @(posedge mclk or negedge reset)
23. begin
24.             if(~reset)      send <= 1'b0;
25.             else            send <= (cnt==20'd1000) ? 1'b1 : 1'b0;
26. end // 20'd1_000 ➔ 20'h3e8 ➔ 20'b0011_1110_1000
```

✓ 2 ~ 16 : reset, mclk, 내부에서 사용할 counter(cnt)를 생성

✓ 17 ~ 20 : 각 모듈에 입력될 신호 생성

✓ 21 ~ 26 : send flag 생성

uart_rx_tb. v (2)

```
27. reg         ren;
28. always @(posedge mclk or negedge reset)
29. begin
30.             if(~reset)      ren <= 1'b0;
31.             else            ren <= (cnt==20'd14000) ? 1'b1 : 1'b0;
32. end

33. wire                txd  ;
34. wire                done ;
35. uart_tx     uart_tx(
36. .reset (reset ),  .mclk (mclk ), .baudrate   (baudrate ),
37.  .parity_sel (parity_sel ), .stop_sel   (stop_sel  ), .tdata (tdata ),
38.  .send (send  ), .txd(txd  ), .done (done  )
39. );

40. wire                rxd = txd  ;
41. wire        [7:0]   rdata      ;
42. wire        rvalid  , overrun  , frame_err  , parity_err ;

43. uart_rx     uart_rx(
44. .reset      (reset     ), .mclk      (mclk      ),
45. .baudrate   (baudrate  ), .parity_sel (parity_sel ),
46. .stop_sel   (stop_sel  ),
47. .rdata      (rdata     ),
48. .ren        (ren       ),
49. .rvalid     (rvalid    ),
50. .overrun    (overrun   ),
51. .frame_err  (frame_err ),
52. .parity_err (parity_err ),
53. .rxd        (rxd       )
54. );
55. endmodule
```

✓ 27 ~ 32:수신된 데이터를 fifo에서 읽기 위한 fifo ren 신호 생성

✓ 33 ~ 55 : uart_tx 모듈

➔ uart_tx 모듈에서 전송된 데이터를 uart_rx 모듈의 입력

➔ 이때 baudrate, parity_sel, stop_sel은 uart_tx, uart_rx 모듈에서 동일한 값을 사용

UART Controller Implementation

➢ *Uart_Rx.xpr*

➢ *UART RX Implementation* ➔ Code Implementation ➔ UART_RX ➔ ***Simulation Verification***

❖ UART_RX ➔ Simulation 검증 ➔ uart_rx : Top Module 지정 ➔ ***Run Simulation***

➔ ren이 High 가 되었을 때, rdata값이 0x55 (0101_0101)

➔ ***frame_err, parity_err, overrun : ' 0 '***

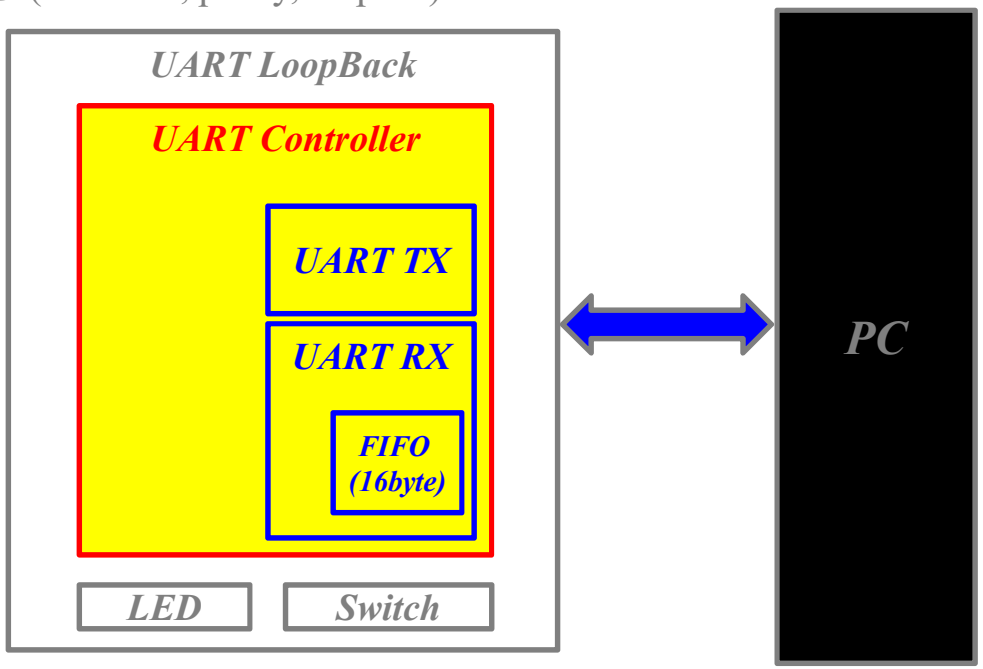
➔ rvalid 신호는 fifo에 데이터가 저장되면 "1"로 되고, fifo에서 데이터를 읽으면 "0"

UART Controller Implementation

- UART TX Implementation
- FIFO_16x8 Implementation
- UART RX Implementation
- **UART Controller Implementation**
- UART Loopback Implementation

➢ UART Controller Implementation

- **UART Controller System Block**
  - ❖ UART TX : Uart Tx Module
  - ❖ ***UART RX : Uart Rx Module, 16bytes FIFO 포함***
  - ❖ ***UART Controller : UART Tx/Rx Controller***
  - ❖ UART LoopBack : LoopBack Test, PC로부터 데이터를 수신하면 그대로 PC로 전송
  - ❖ Led : 상태를 표시함, Tx available, Rx available, overrun error, frame error, parity error
  - ❖ SW : 모드 설정 (baudrate, parity, stop bit)



UART Controller Implementation

➢ Uart_Controller_exam.xpr

➢ UART Controller Implementation ➔ **Code Implementation**

uart_controller. v (1)

```
1.  `timescale 1ns / 1ps

2.  module uart_controller(
3.              reset, mclk, baudrate,
4.              parity_sel , // 0 : none,  1 : even, 2 : odd
5.              stop_sel   ,    // 0 : 1bits, 1 : 2bits
6.              tdata      ,    // tx data
7.              send       ,    // send flag
8.              trdy       ,    // tx ready
9.              txd        ,    // uart_txd
10.             rxd        ,    // uart_rxd
11.             ren        ,    // read enable
12.             rdata      ,    // rx data
13.             rvalid     ,    // rdata valid flag
14.             overrun    ,    // rx fifo full
15.             frame_err  ,    // stop bit error
16.             parity_err      // parity error
17. );
18. input       reset , mclk     ;
19. input       [15:0]           baudrate   ;
20. input       [1:0]            parity_sel ;
21. input       stop_sel   ;
22. input       [7:0]tdata     ;
23. input       send       ;
24. output      trdy  , txd;
25. input       rxd , ren;
26. output      [7:0]            rdata      ;
27. output      rvalid, overrun, frame_err, parity_err ;
```

uart_controller. v (2)

```
28. wire                         trdy  ;
29. wire                         txd   ;
30. uart_tx     uart_tx(
31.             .reset     (reset     ),
32.             .mclk      (mclk      ),
33.             .baudrate  (baudrate  ),
34.             .parity_sel (parity_sel ),
35.             .stop_sel  (stop_sel  ),
36.             .tdata     (tdata     ),
37.             .send      (send      ),
38.             .done      (trdy      ),
39.             .txd       (txd       )
40. );

41. wire        [7:0]            rdata     ;
42. wire                         rvalid    ;
43. wire                         overrun   ;
44. wire                         frame_err ;
45. wire                         parity_err ;

46. uart_rx     uart_rx(
47.             .reset     (reset     ),
48.             .mclk      (mclk      ),
49.             .baudrate  (baudrate  ),
50.             .parity_sel (parity_sel ),
51.             .stop_sel  (stop_sel  ),
52.             .rdata     (rdata     ),
53.             .ren       (ren       ),
54.             .rvalid    (rvalid    ),
55.             .overrun   (overrun   ),
56.             .frame_err (frame_err ),
57.             .parity_err (parity_err ),
58.             .rxd       (rxd       )
59. );

60. endmodule
```

❖ 1 ~ 27 : port 선언

➔ *baudrate, parity_sel, stop_sel* : uart_tx, uart_rx 모듈에 공통 사용

➔ *tdata* : uart_tx 모듈을 통하여 전송하는 데이터, 8bits

➔ *send* : uart_tx 모듈에 전송을 시작하라는 start flag

➔ *trdy* : uart_tx 모듈에 전송할 준비가 되었음을 알려주는 flag, uart_tx 모듈의 idle 상태와 같음. 즉 idle 상태에 있으면 전송 가능함으로 인식

➔ *rvalid* : uart_rx 모듈에 수신한 데이터가 있음을 알려주고, uart_rx 모듈의 수신 fifo에 수신한 데이터가 있음을 알려줌

➔ *ren, rdata* : uart_rx 모듈에 수신한 데이터가 있을 때, ren을 active 로 만들어서 수신한 데이터를 rdata로 받음.

➔ *overrun, frame_err, parity_err* : 수신한 데이터에 에러가 발생

❖ 58 – 70 : uart_tx module

❖ 72 – 91 : uart_rx module

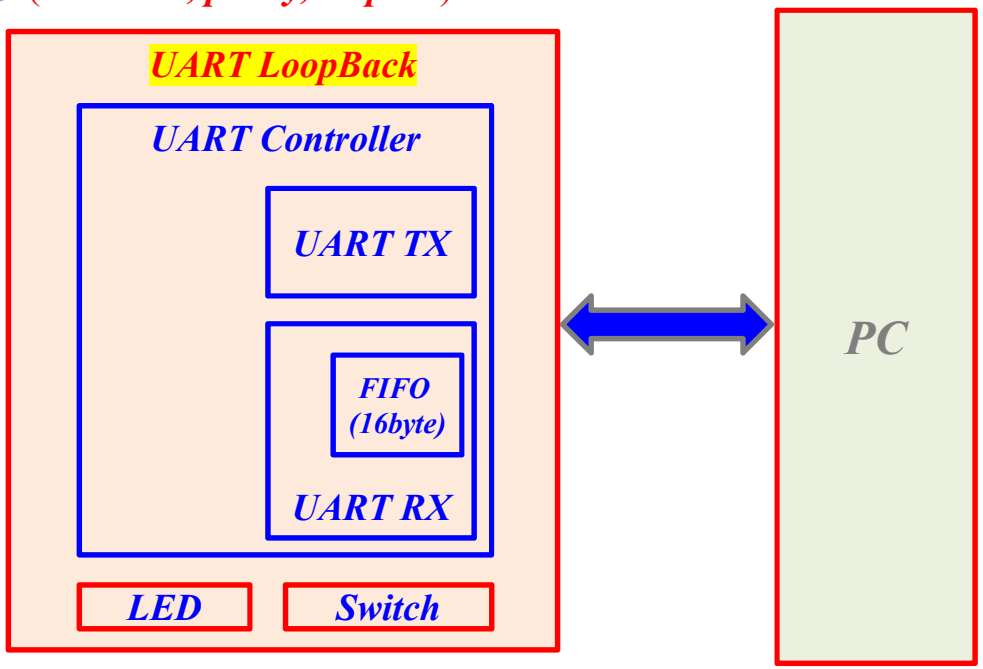
UART Controller Implementation

- UART TX Implementation
- FIFO_16x8 Implementation
- UART RX Implementation
- UART Controller Implementation
- **UART Loopback Implementation**

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Controller Implementation

- **UART Controller System Block**
  - ❖ UART TX : Uart Tx Module
  - ❖ ***UART RX : Uart Rx Module, 16bytes FIFO 포함***
  - ❖ UART Controller : UART Tx/Rx Controller
  - ❖ ***UART LoopBack : LoopBack Test***, PC로부터 데이터를 수신하면 그대로 PC로 전송
  - ❖ ***Led*** : 상태를 표시 ➔ ***Tx available, Rx available, overrun error, frame error, parity error***
  - ❖ ***SW*** : 모드 설정 (***baudrate, parity, stop bit***)



UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ *UART Loopback Implementation* ➔ ***Uart Controller Verification : LoopBack Test***

➔ PC와 연결해서 PC에서 전송한 데이터를 그대로 다시 PC로 전송을 구현

➔ 코드 구현후에 ***Bitstream(.bit or .bin)***을 생성 ➔ 보드에 ***Program Device*** ➔ PC와 연결 및 테스트

➢ 구성

❖ 보드에 있는 Switch와 LED를 이용하여 여러가지 모드를 설정하고 결과를 표시

3) Stop Bit 설정

SW3	Stop bit
0	1 bit
1	2 bit

2) Parity 설정

SW2	SW1	Parity
0	0	none
0	1	even
1	0	odd
1	1	none

1) Baudrate 설정

SW0	Baudrate
0	115,200
1	38,400

4) LED 정보

LED	LD5	LD4	LD2	LD1	LD0
status	trdy(tx ready, tx done)	rvalid(rx valid)	overrun_error	frame_error	parity_error

UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation ➔ Code Implementation ➔* ***Input and Output Port*** *정의*

- ❖ PC에서 uart_txd를 통하여 데이터를 전송하면 uart_rx 모듈에서 데이터를 수신하고 수신 버퍼에 전송
- ❖ 수신 버퍼에 데이터가 있으면 이 데이터를 읽어서 uart_tx 모듈을 통하여 PC로 데이터를 전송
- ❖ sw[3:0]을 통하여 UART 모드[*Baudrate(SW0), Parity Bit(SW1 & SW2), Stop Bit(SW3)*]을 설정
- ❖ PC의 **터미널 프로그램(*Windows Application Program*)**과 환경을 맞추어야 정상적인 데이터 송수신을 구현
- ❖ 보드와 PC 프로그램 간에 설정 값이 다르면 에러가 발생하고, 이를 LED를 통해 확인

signal	in/out	size	description
reset	*in*	[0]	*reset, Active Low*
clk	*in*	[0]	*clock, 100Mhz*
uart_rxd	*in*	[0]	***uart_rxd***
uart_txd	*out*	[0]	***uart_txd***
sw	*in*	[3:0]	*user switch, mode 설정*
led	*out*	[7:0]	*user led, 상태 표시*

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ **Input and Output Port** 정의

uart_loopback. v (1)

```
1.  `timescale 1ns / 1ps
2.
3.  module uart_loopback(
4.              reset    ,
5.              clock    ,
6.              uart_rxd ,
7.              uart_txd ,
8.              sw       ,
9.              led
10.);
11.input        reset  , clock , uart_rxd ;
12.output       uart_txd ;
13.input        [3:0]          sw      ;
14.output       [7:0]          led     ;

15.// 1) define state
16.reg          [1:0]          m_state ;
17.parameter                   M_IDLE     = 2'd0 ;
18.parameter                   M_FREAD    = 2'd1 ;
19.parameter                   M_SEND     = 2'd2 ;

20.// ----------------------------------------
21.// 2) state flag
22.wire                        s_idle   = (m_state == M_IDLE  ) ? 1'b1 : 1'b0 ;
23.wire                        s_fread  = (m_state == M_FREAD ) ? 1'b1 : 1'b0 ;
24.wire                        s_send   = (m_state == M_SEND  ) ? 1'b1 : 1'b0 ;
```

1) Baudrate 설정

SW0	Baudrate
0	115,200
1	38,400

3) Stop Bit 설정

SW3	Stop bit
0	1 bit
1	2 bit

2) Parity 설정

SW2	SW1	Parity
0	0	none
0	1	even
1	0	odd
1	1	none

- 1 ~ 14 : *port* 선언
- 11 : input ➔ *reset, clock, uart_rxd*
- 12 : output ➔ *uart_txd*
- 13 : sw ➔ UART Mode 설정
- 14 : led ➔ UART Status, Error 표시
- 16 – 19 : state 정의
  - ➔ *M_IDLE* 상태에서 uart_rxd 로부터 데이터를 수신하면 *M_FREAD* 상태
  - ➔ *M_SEND* 상태에서는 수신 버퍼에 저장된 데이터를 읽어서 송신(uart_tx)모듈로 보내고, uart_txd를 통하여 데이터를 전송
  - ➔ 데이터 전송이 완료되면 *M_IDLE* 상태
- 라인 48 – 50 : state flag 정의

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ **uart_loopback. v**

uart_loopback. v (2)

```
25. // 3) code implementation

26. reg         [2:0]       cnt_fread;
27. always @(posedge clock or negedge reset)
28. begin
29.             if(~reset)      cnt_fread <= 3'b0;
30.             else            cnt_fread <= ~s_fread ? 3'b0 : cnt_fread+1'b1 ;
31. end

32. reg                     ren;
33. always @(posedge clock or negedge reset)
34. begin
35.             if(~reset)      ren <= 1'b0;
36.             else            ren <= (cnt_fread==3'd2) ? 1'b1 : 1'b0;
37. end

38. wire        [7:0]       rdata;
39. reg         [7:0]       tdata;
40. always @(posedge clock or negedge reset)
41. begin
42.             if(~reset)      tdata <= 8'b0;
43.             else            tdata <= (cnt_fread==3'd7) ? rdata : tdata ;
44. end

45. reg                     send;
46. always @(posedge clock or negedge reset)
47. begin
48.             if(~reset)      send <= 1'b0;
49.             else            send <= (cnt_fread==3'd7) ? 1'b1 : 1'b0 ;
50. end
```

➢ 26 – 31 : FREAD 상태에서 사용하는 counter 생성

➢ 32 – 37 : *Receive FIFO* 에서 데이터를 읽기 위한 ren 생성

➢ 38 – 44 : *Receive FIFO* 에서 읽은 데이터(*rdata*)를 uart_tx 모듈을 통해 송신하기 위한 데이터(tdata) 생성

➔ *Receive FIFO*에서 데이터를 읽을 때 딜레이가 발생 ➔ 즉 ren이 Active 되고 약 *1~2 clock* 후에 rdata가 유효한 데이터가 됨 ➔ 따라서 *ren*은 *cnt_fread : 3* 에서 active 되고, tdata를 생성하는 것은 cnt_fread : 7

➢ 45 – 50 : 데이터 송신을 위한 send flag 생성

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ **uart_loopback. v**

uart_loopback. v (3)

```
51. reg         [2:0]          cnt_send;
52. always @(posedge clock or negedge reset)
53. begin
54.             if(~reset)  cnt_send <= 3'b0;
55. else   cnt_send <= ~s_send ? 3'b0 : (cnt_send==3'd7) ? 3'd7 : cnt_send+1'b1 ;
56. end
```

```
57. wire                    trdy   ;
58. wire                    rvalid ;
```

```
59. // 4) state transition
60. always @(posedge clock or negedge reset)
61. begin
62.             if(~reset)      m_state <= 1'b0;
63.             else            m_state <= (s_idle  & rvalid & trdy) ? M_FREAD  :
64.                             (s_fread & (cnt_fread==3'd7)      ) ? M_SEND   :
65.                             (s_send  & (cnt_send==3'd7) & trdy ) ? M_IDLE   :
66.                             m_state ;
67. end
```

```
68. wire        [15:0]          baudrate   = ~sw[0] ? 16'd868  :  // 115,200
69.                             16'd2604 ;   //  38,400
70.
71. wire        [1:0]           parity_sel = (sw[2:1]==2'b00) ? 2'b00 : // none
72.                             (sw[2:1]==2'b01) ? 2'b01 : // even
73.                             (sw[2:1]==2'b10) ? 2'b10 : // odd
74.                             2'b00 ;
75.
76. wire                        stop_sel   = ~sw[3] ? 1'b0 :  // 1-stop
77.                             1'b1 ;  // 2-stop
```

- 51 – 56 : SEND 상태에서 사용하는 counter 생성
- 57 : tx ready, uart_tx 모듈에서 생성되는 신호
- 58 : rx data available, uart_rx 모듈에서 생성되는 신호
- 61 ~ 68 : 상태 전이 구현

➔ rvalid, trdy 신호가 active 되면 M_FREAD 상태

➔ 수신 FIFO로 부터 데이터를 읽은 후, M_SEND 상태

➔ 전송이 완료되면 (trdy active) M_IDLE 상태

- 69 ~ 78 : sw 에 따른 모드 설정

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ **uart_loopback. v**

uart_loopback. v (4)

```
79. wire                    uart_txd  ;
80. wire                    overrun   ;
81. wire                    frame_err ;
82. wire                    parity_err ;
83. uart_controller         uart_controller(
84.             .reset      (reset),
85.             .mclk       (clock),
86.             .baudrate   (baudrate),
87.             .parity_sel (parity_sel),
88.             .stop_sel   (stop_sel),

89.             .tdata      (tdata),
90.             .send       (send),
91.             .trdy       (trdy ),
92.             .txd        (uart_txd),

93.             .rxd        (uart_rxd),
94.             .ren        (ren),
95.             .rdata      (rdata),
96.             .rvalid     (rvalid),
97.             .overrun    (overrun),
98.             .frame_err  (frame_err ),
99.             .parity_err (parity_err)
100.);
```

➢ 79 – 100 : *uart_controller* 모듈

```
101.wire    [7:0]   led = {2'b0, trdy, rvalid, 1'b0, overrun, frame_err, parity_err};

102.endmodule
```

➢ 101 : led에 상태를 표시

UART Controller Implementation

- UART TX Implementation
- FIFO_16x8 Implementation
- UART RX Implementation
- UART Controller Implementation
- **UART Loopback Implementation**
  - ✓ **Loopback Test Bench**

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ Simulation ➔ **uart_loopback_tb. v**

uart_loopback_tb. v (1)

```
1.  `timescale 1ns / 1ps

2.  module uart_loopback_tb();

3.  reg         reset, mclk;

4.  initial     begin
5.              reset = 0;
6.              mclk = 0;
7.
8.  #10000      reset = 1;
9.  end

10. always  #5 mclk = ~mclk;                // 100 Mhz

11. reg         [9:0]           cnt;
12. always @(posedge mclk or negedge reset)
13. begin
14.             if(~reset)      cnt <= 10'b0;
15.             else            cnt <= (cnt==10'd1023) ? 10'd1023 : cnt+1'b1;
16. end

17. wire        [15:0]          baudrate   = 16'd868;
18. wire        [7:0]           tdata      = 8'h55;        // 0101_0101
19. wire        [1:0]           parity_sel = 2'b00;
20. wire                        stop_sel   = 1'b0 ;
```

➢ uart_tx 모듈에서 출력되는 txd 신호를 uart_loopback 으로 입력해서 수신과 송신이 정상적으로 이루어지는지를 확인

➢ uart_tx 모듈에서 생성된 txd 신호를 uart_loopback의 uart_rxd 입력으로 인가

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ UART Loopback Implementation ➔ Code Implementation ➔ Simulation ➔ **uart_loopback_tb. v**

uart_loopback_tb. v (2)

```
21. reg         send;
22. always @(posedge mclk or negedge reset)
23. begin
24.             if(~reset)  send <= 1'b0;
25.             else                send <= (cnt==10'd1000) ? 1'b1 : 1'b0;
26. end

27. wire        txd;
28. uart_tx     uart_tx(
29.             .reset      (reset), .mclk (mclk ), .baudrate (baudrate ),
30.             .parity_sel (parity_sel ),
31.             .stop_sel   (stop_sel),
32.             .tdata      (tdata),
33.             .send       (send),
34.             .txd        (txd ),
35.             .done       (done)
36. );

37. wire        uart_txd ;
38. wire        [7:0]           led;
39. uart_loopback               uart_loopback(
40.             .reset      (reset      ),
41.             .clock      (mclk       ),
42.             .uart_txd   (uart_txd   ),
43.             .uart_rxd   (txd        ),
44.             .sw         (4'b00      ),
45.             .led        (led        )
46. );

47. endmodule
```

➢ uart_tx 모듈에서 생성된 txd 신호를 uart_loopback 의 uart_rxd 입력으로 인가

UART Controller Implementation

- UART TX Implementation
- FIFO_16x8 Implementation
- UART RX Implementation
- UART Controller Implementation
- **UART Loopback Implementation**
  - ✓ Loopback Test Bench
  - ✓ **Simulation Result**

UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ ***Code Implementation*** ➔ ***Simulation Result***

➔ uart rxd로 수신된 데이터가 uart txd를 통해 그대로 송신상태 이동을 확인

➔ s_idle 상태에서 데이터 수신이 완료되면 s_fread 상태

➔ s_fread 상태에서 전송이 시작되면 s_send 상태

➔ 전송이 완료되면 s_idle 상태

UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ ***Code Implementation*** ➔ ***Simulation Result***

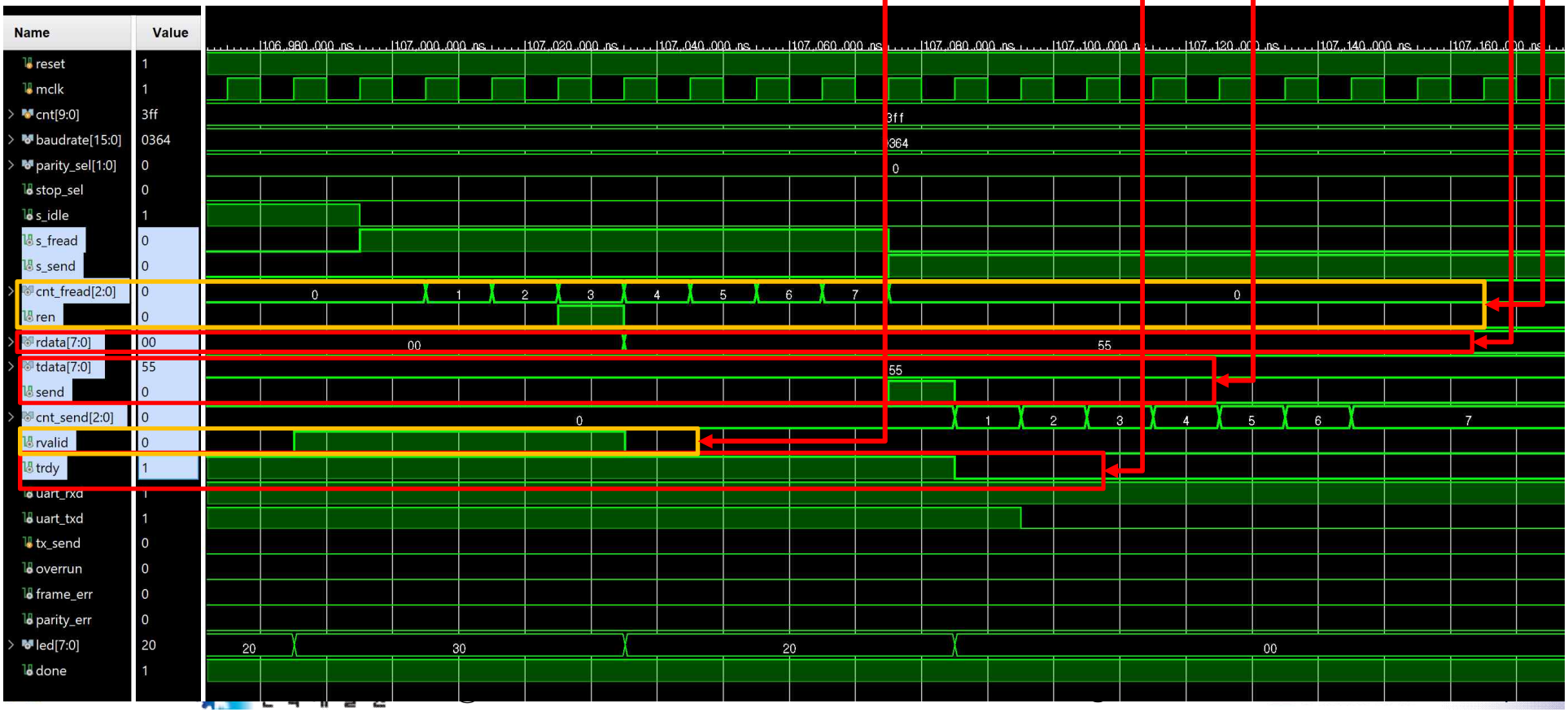
➔ cnt_fread : 3 일 때, ren 이 enable

➔ cnt_fread : 4 일 때 수신 버퍼의 데이터가 rdata로 로드

➔ tdata에 rdata 값이 저장됨과 동시에 send flag가 enable

➔ trdy 는 전송이 시작되면 0가 되고 전송이 완료되면 1

➔ rvalid 는 수신 버퍼에 데이터가 있으면 1, 없으면 0



UART Controller Implementation

- UART TX Implementation
- FIFO_16x8 Implementation
- UART RX Implementation
- UART Controller Implementation
- UART Loopback Implementation
- **XDC & Program Device**

UART Controller Implementation ➢ Uart_Controller_exam.xpr

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ ***UART Loopback*** ➔ ***Bitstream(xdc)***

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clock]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock]
## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports {reset}]
## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {led[7]}]
##USB-RS232 Interface
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports uart_rxd]
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports uart_txd]
## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ *UART Loopback* ➔ ***USB-UART(Basys3)***

5 USB-UART Bridge (Serial Port)

The Basys 3 includes an FTDI FT2232HQ USB-UART bridge (attached to connector J4) that allows you to use PC applications to communicate with the board using standard Windows COM port commands. Free USB-COM port drivers, available from www.ftdichip.com under the "Virtual Com Port" or VCP heading, convert USB packets to UART/serial port data. Serial port data is exchanged with the FPGA using a two-wire serial port (TXD/RXD). After the drivers are installed, I/O commands can be used from the PC directed to the COM port to produce serial data traffic on the B18 and A18 FPGA pins.

Two on-board status LEDs provide visual feedback on traffic flowing through the port: the transmit LED (LD18) and the receive LED (LD17). Signal names that imply direction are from the point-of-view of the DTE (Data Terminal Equipment), in this case the PC.

The FT2232HQ is also used as the controller for the Digilent USB-JTAG circuitry, but the USB-UART and USB-JTAG functions behave entirely independent of one another. Programmers interested in using the UART functionality of the FT2232 within their design do not need to worry about the JTAG circuitry interfering with the UART data transfers, and vice-versa. The combination of these two features into a single device allows the Basys 3 to be programmed, communicated with via UART, and powered from a computer attached with a single Micro USB cable. The connections between the FT2232HQ and the Artix-7 are shown in Fig. 6.

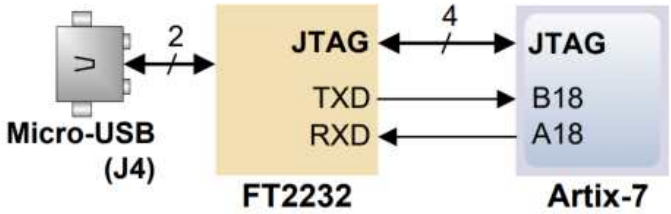
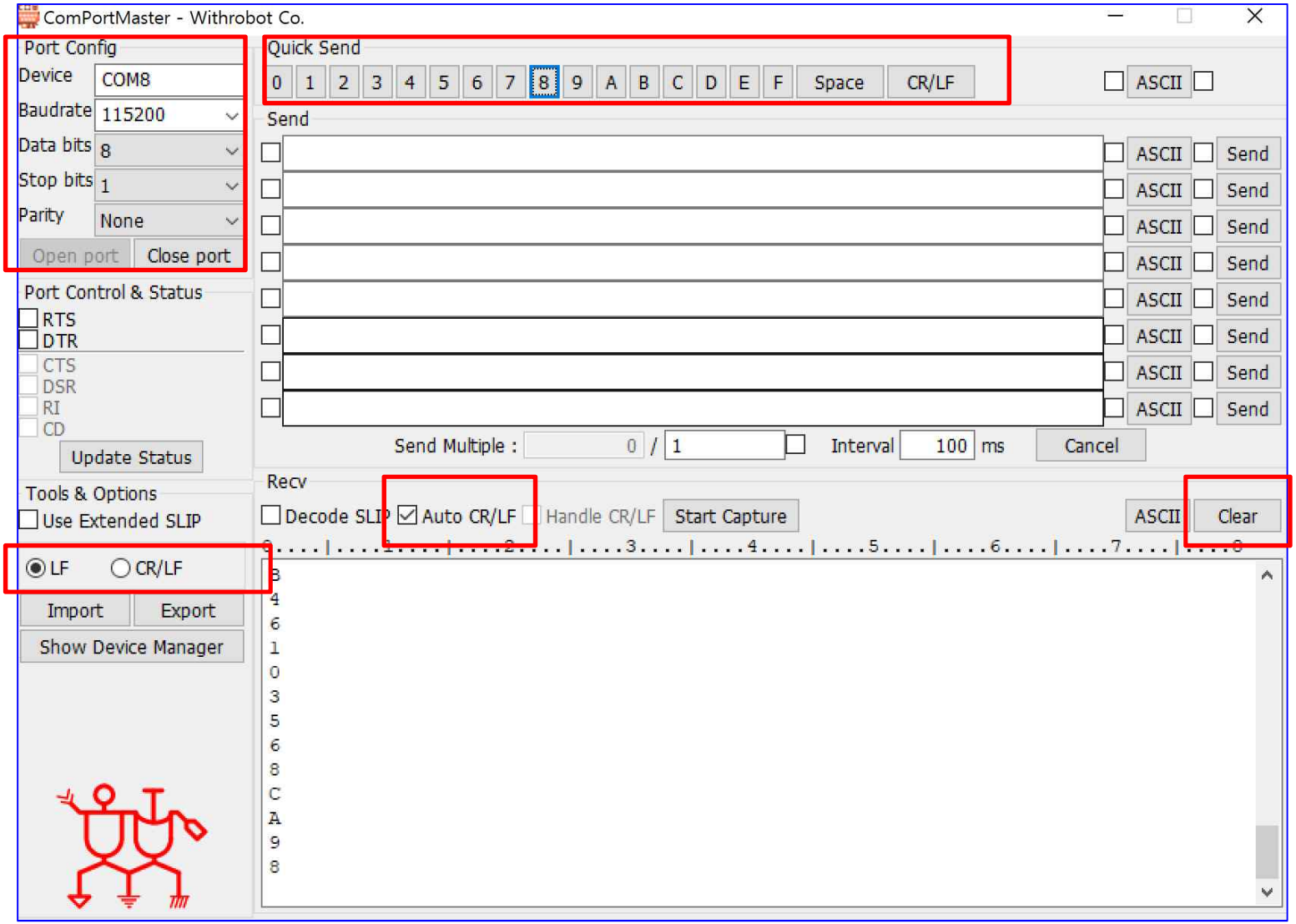


Figure 6. Basys 3 FT2232HQ connections.

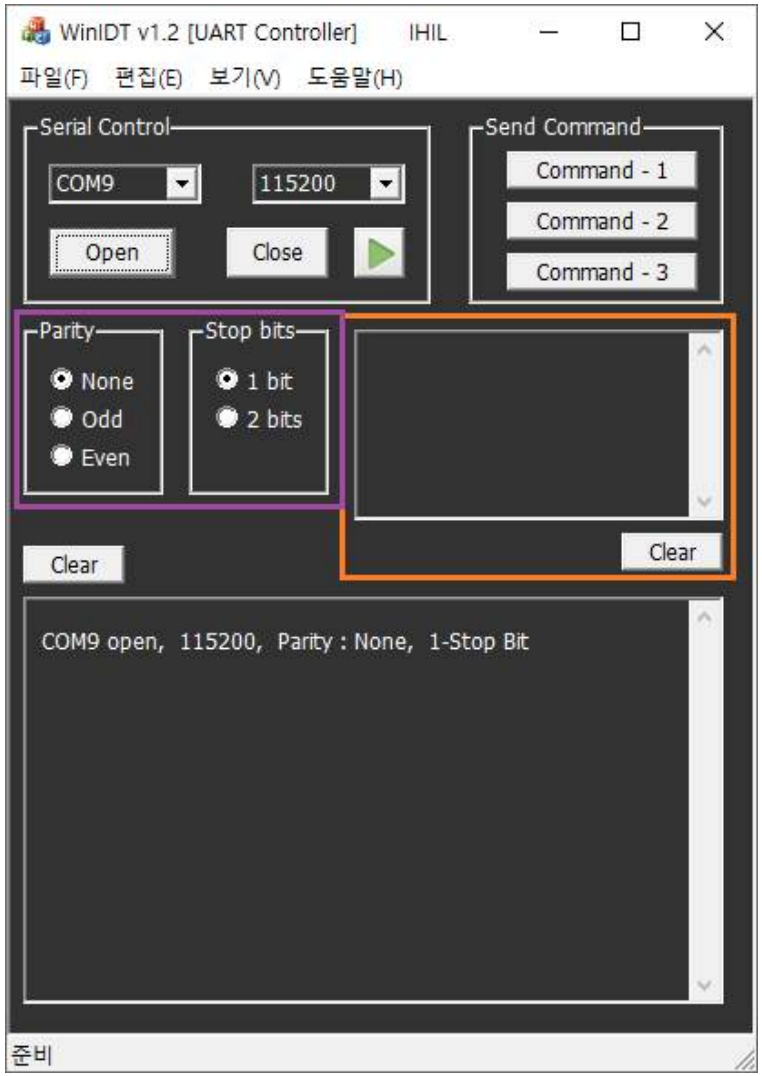
UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ Bitstream 생성 ➔ ***Check Result***



UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ Bitstream 생성 ➔ ***Check Result***



- **Serial Control : Com Port, Baudrate 설정후 Open 클릭➔ Com port 가 Open ➔ Close 클릭하면 Close**
- **Send Command : command-1 클릭하면 "0~9"까지 전송되고, command-2 클릭하면 "A~Z"까지 전송되고, command-3 클릭하면 "a~z"까지 전송**
- **보라색 : Parity, Stop bits 을 설정**
- **주황색 : 문자를 입력하면 보드로 데이터가 전송**
- **하단 에디터 : 보드에서 수신된 데이터**

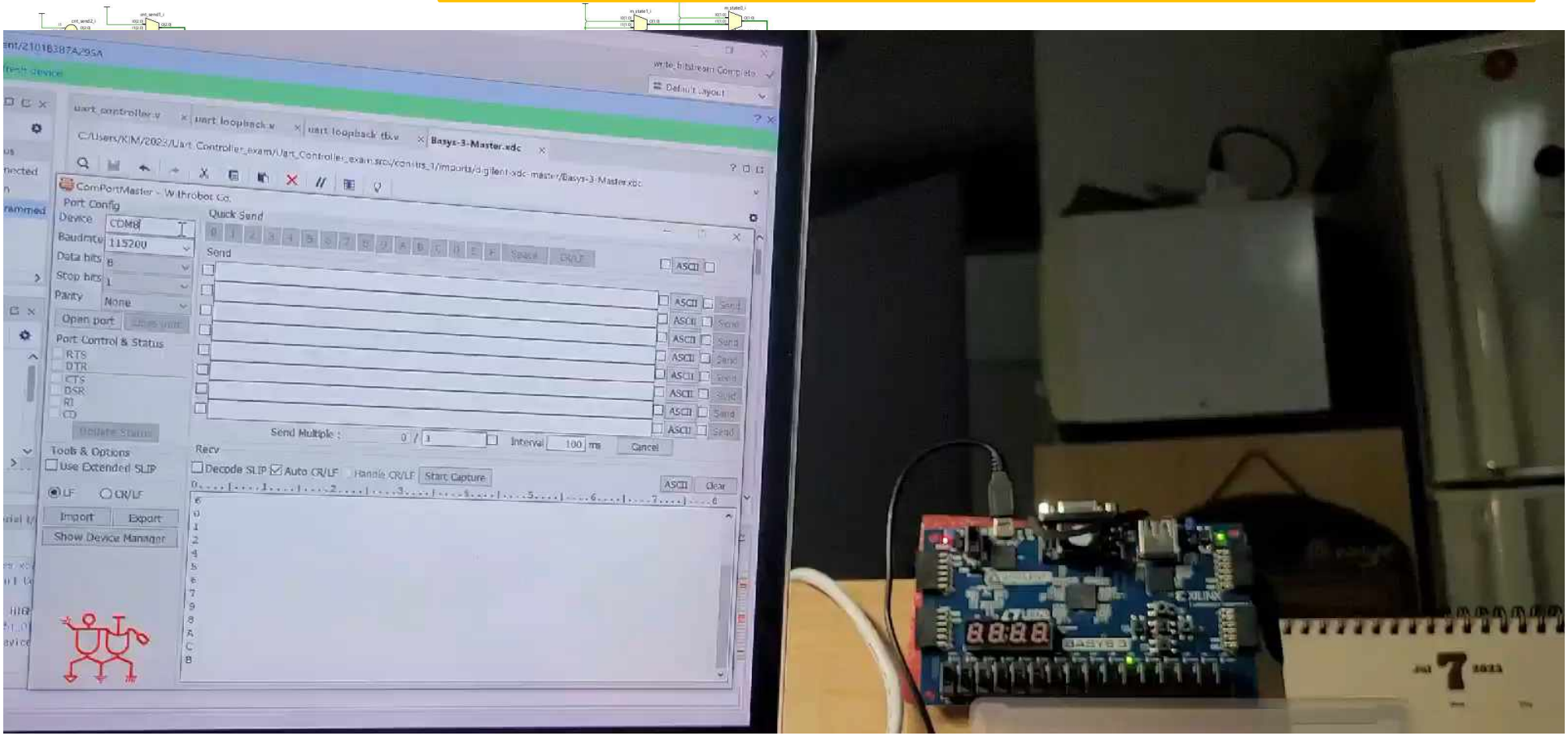
UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ Bitstream 생성 ➔ 결과 확인

➔ [Test-1 : bps : 115,200, parity : none, stop bits : 1bits]

- 보드의 SW을 모두 0 로 만들고 전원을 인가 ➔ PC 프로그램의 설정을 115200, None, 1 bit 로 설정➔ Open을 클릭 ➔ command-1, 2, 3 버튼을 클릭하고 송신 에디터 창에 문자를 입력하고 결과를 확인

❖ **전송한 데이터가 그대로 표시 & 에러 led는 점등되지 않고, trdy led만 점등**



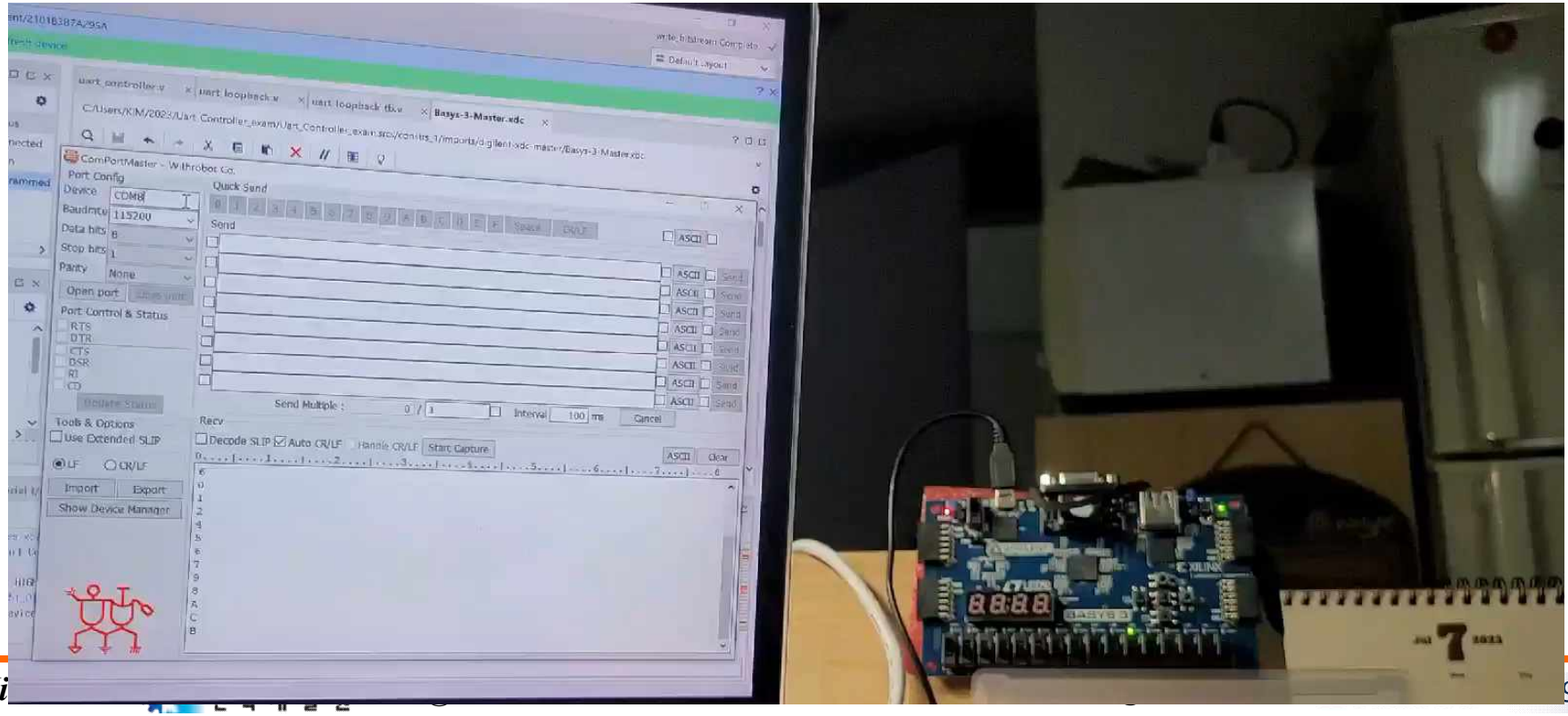
UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

➢ *UART Loopback Implementation* ➔ Code Implementation ➔ Bitstream 생성 ➔ 결과 확인

➔ [Test-2 : 보드는 None Parity, PC 프로그램은 Odd Parity 로 설정]

- Close 버튼을 클릭 ➔ Com Port를 Close ➔ Parity를 Odd로 선택 ➔ Open 클릭 ➔ 동일하게 데이터를 전송하고 결과를 확인

- **LD1 이 점등 : LD1은 frame_error (stop bit 에러) ➔ PC에서는 Odd Parity bit를 전송하는데, 보드는 Parity 설정이 None으로 설정되어 있기 때문에 Parity bit가 1이면 정상적인 Stop bit로 인식하는데, Parity bit가 0로 입력되면 Stop bit 오류로 인식하게 됨 ➔ 이 상태에서 보드의 Parity 설정을 Odd (SW2 : 1, SW1 : 0)로 설정하면 에러가 발생하지 않는 것 확인**



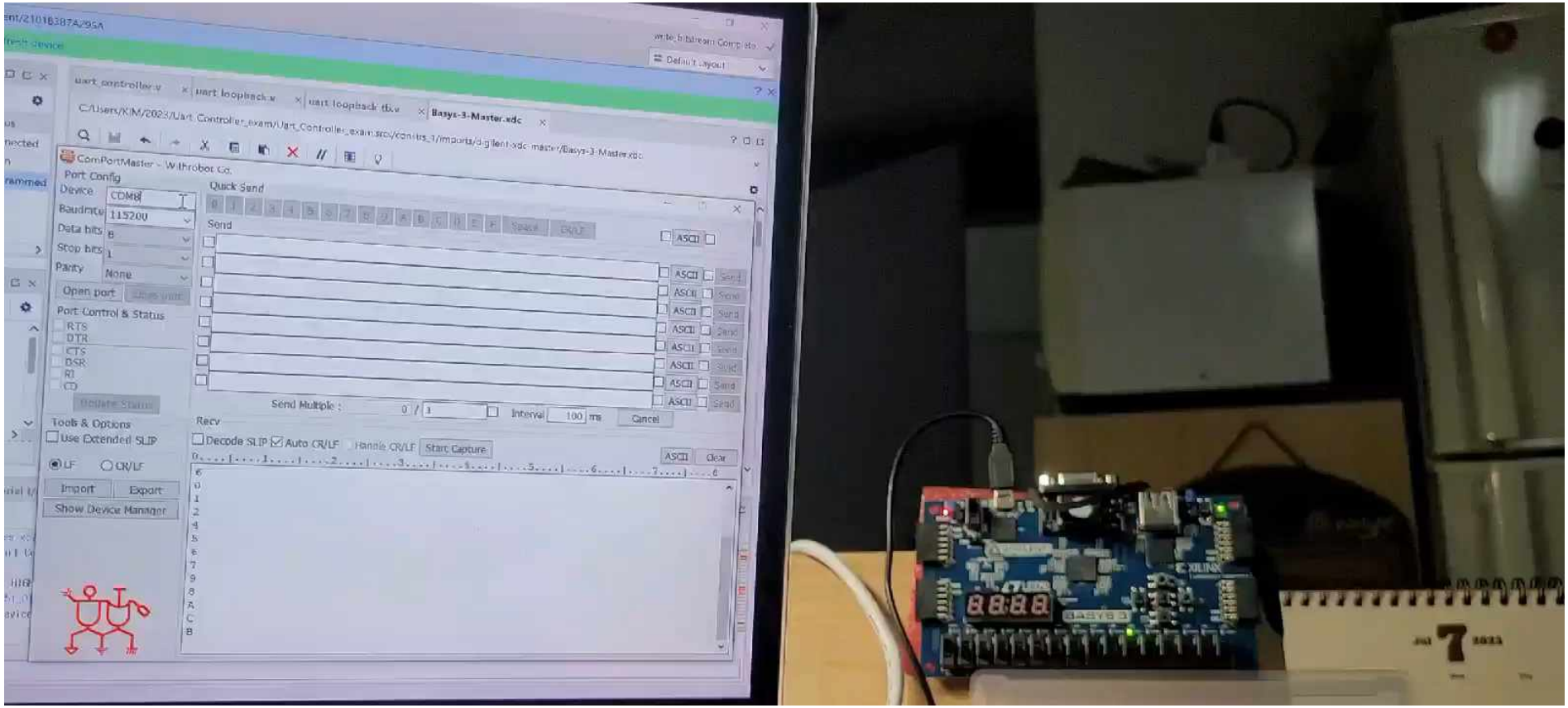
UART Controller Implementation ➢ *Uart_Controller_exam.xpr*

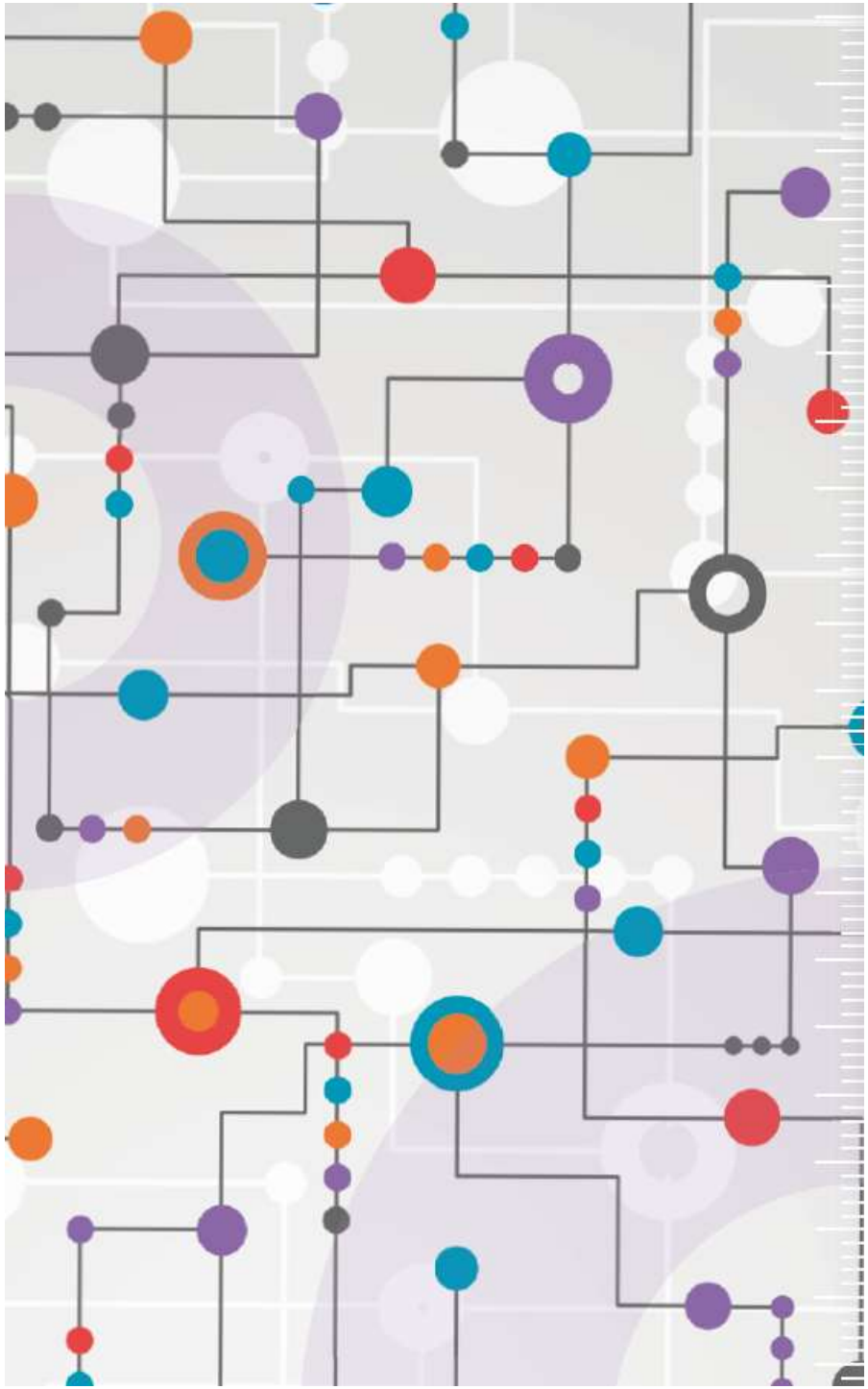
➢ *UART Loopback Implementation* ➔ Code Implementation ➔ Bitstream 생성 ➔ 결과 확인

➔ [Test-3 : PC는 Odd Parity, 보드는 Even Parity로 설정]

- 보드의 Parity 설정을 Even (SW2 : 0, SW1 : 1)으로 설정 ➔ 데이터를 전송하고 결과를 확인

- **LD0 (Parity Error)가 점등 : 보드의 Parity 설정을 Even (SW2 : 0, SW1 : 1)으로 설정하고 , PC에서는 Odd Parity bit를 전송**
  ➔ 보드의 Parity 설정을 Odd (SW2 : 1, SW1 : 0)로 설정하면 에러가 발생하지 않는 것 확인





수고하셨습니다.