

# TI DSP, MCU, Xilinx Zynq FPGA

## 프로그래밍 전문가 과정

**2018.07.18**

**강사 – Innova Lee(이상훈)**  
gcccompil3r@gmail.com

**학생 – 안상재**  
sangjae2015@naver.com

## 1. SPI 통신 (Serial Peripheral Interface) 개념

# 1. SPI 통신 (Serial Peripheral Interface) 개념

## 1-1. SPI 특징

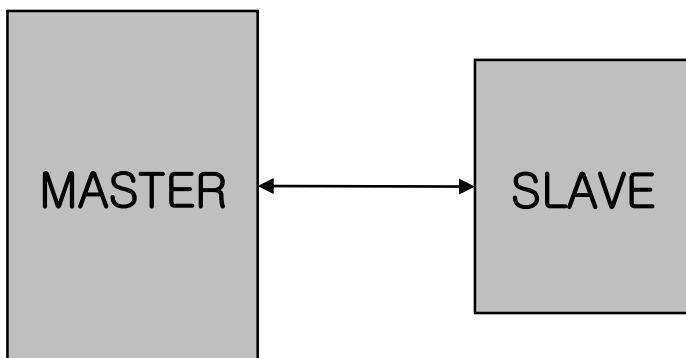
- SPI는 Motorola/Freescale사에서 처음 개발하였음.
- 임베디드 시스템 내부의 모듈, IC간 연결 및 통신 시 주로 사용됨. (온도센서, EEPROM 메모리)
- 단거리 전이중 전송방식 (ex : 전화 통화)

## 1-2. SPI 버스

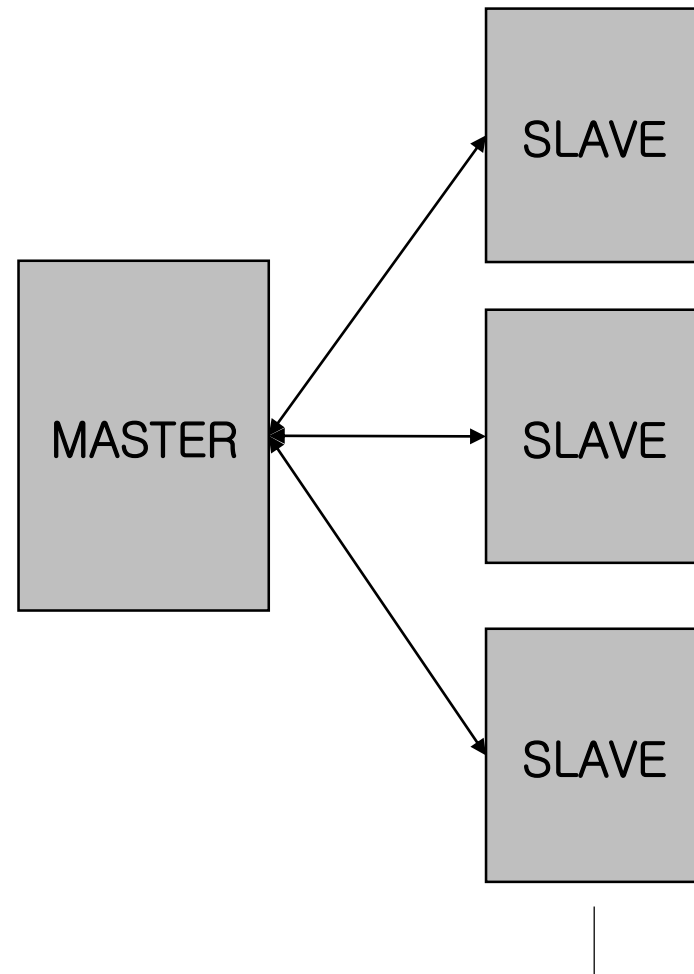
- 1:N 통신 (1개의 마스터가 다수의 슬레이브를 제어함)
- 클럭 라인을 별도로 사용하는 동기식(synchronous) 직렬 데이터 버스 방식  
⇒ 동기식 통신이란 클럭선(CLK) 을 사용해서 클럭의 엣지마다 데이터를 1bit 전송하는 방식을 말함.  
⇒ 비동기식 통신의 대표적인 예로는 UART 통신이 있음.
- 물리적으로 분리된 송신 및 수신 데이터 선로를 사용하므로 전이중 통신이 가능한 장점이 있음.
- 주소 영역을 사용하지 않으므로 여러 개의 슬레이브 장치에 대한 어드레싱을 위해 각 장치별 Chip Select(CS) 신호선이 추가되어야 하는 문제점이 있음.
- SPI 버스 상의 마스터 노드는 클럭을 제공하고 슬레이브 장치를 선택하는 기능을 가진다.  
나머지 노드는 슬레이브 노드로써 개별적인 Slave Select(chip select) 선로에 의해 선택됨.

- SCLK(Serial Clock) : 마스터가 생성하는 클럭 신호이다.
  - MOSI(Master Output, Slave Input) : 마스터가 출력하는 직렬 데이터 신호이다. MSB부터 송신된다.
  - MISO(Master Input, Slave Output) : 마스터가 수신하는 직렬 데이터 신호이다.
  - CS(Chip Select) : 마스터에 의한 슬레이브 선택용 제어 신호이다. SS은 보통 active low 방식이지만 그 반대의 경우도 있다. 선택되지 않은 장치의 MISO 신호값은 하이 임피던스값을 유지한다.
-

1-3. 1:1 통신과 1:N 통신



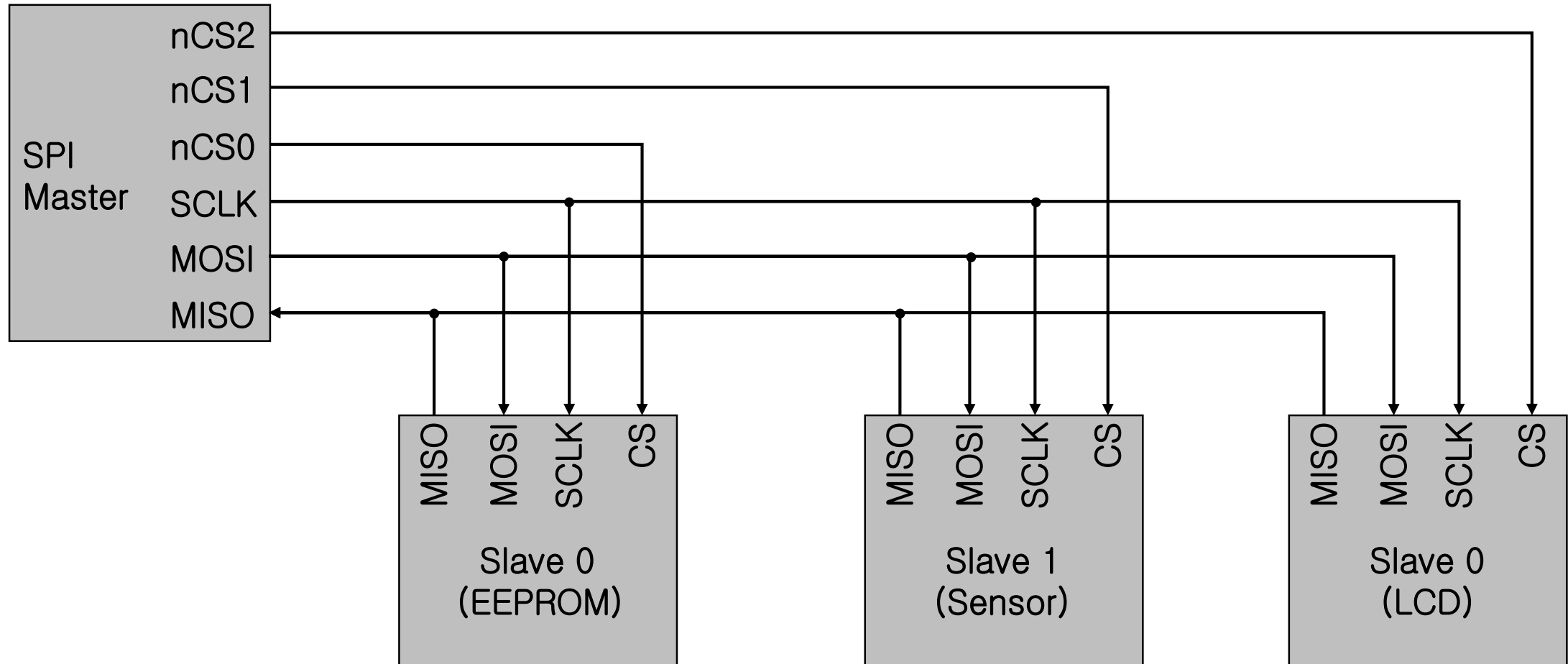
a) 1:1 통신



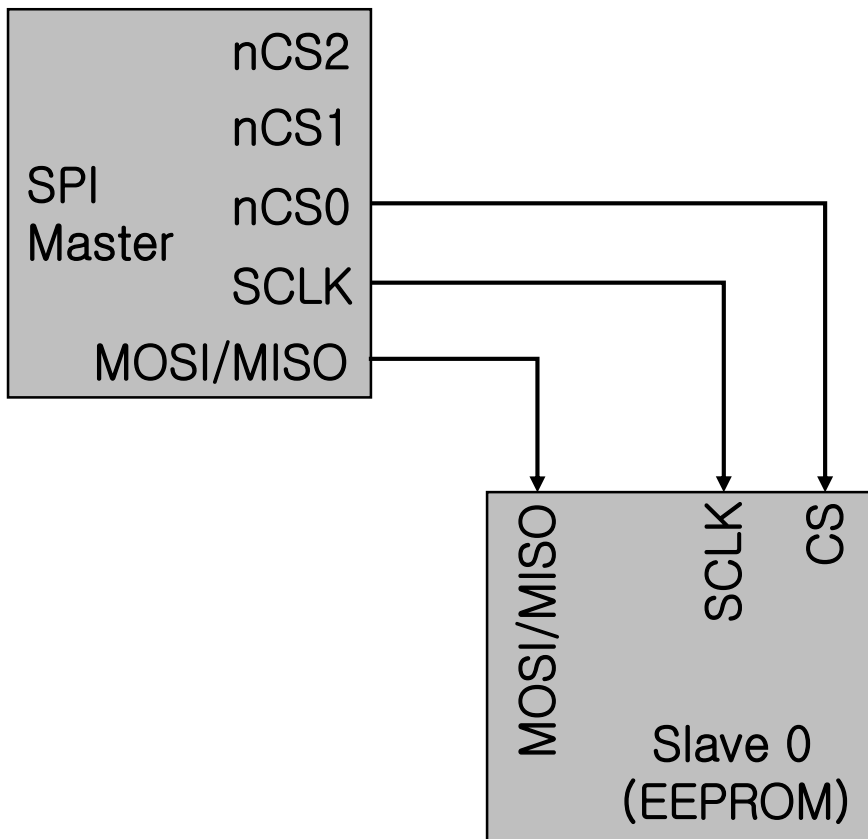
b) 1:N 통신

## 1-4. SPI 버스의 종류

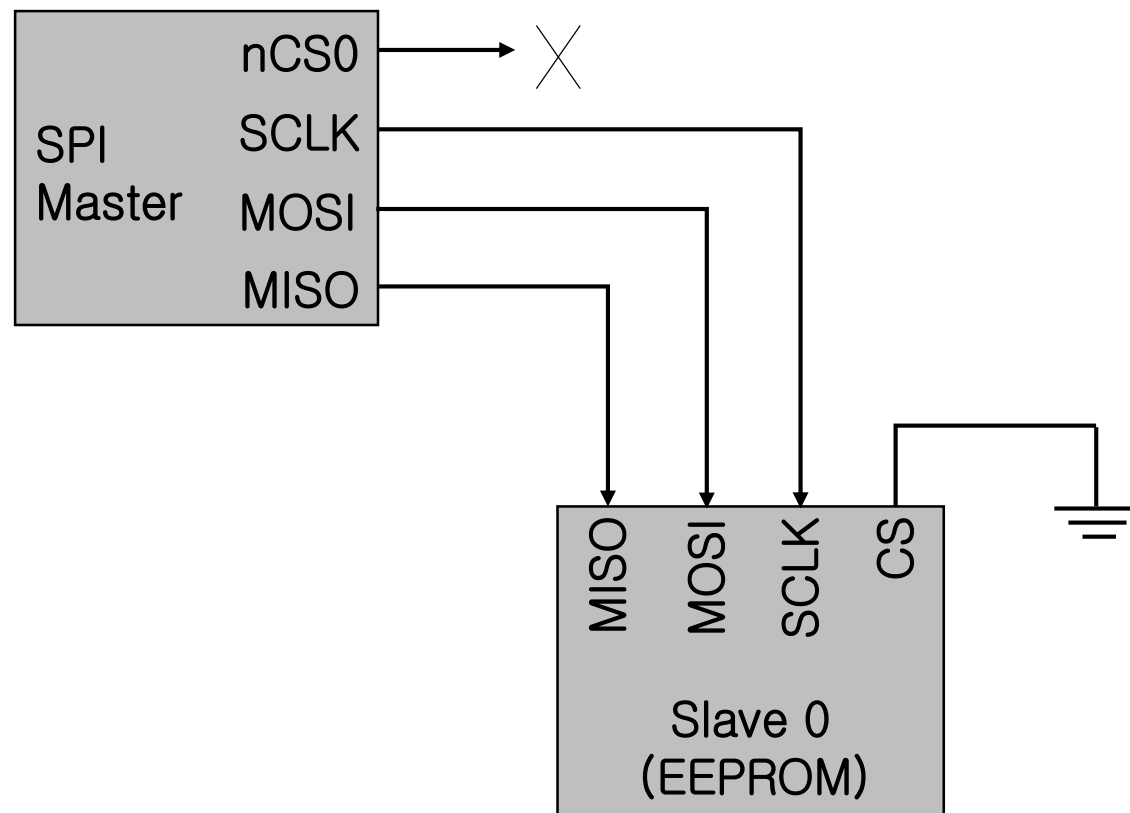
(a) 4-wire SPI



(b) 3-wire SPI/Half-duplex (MicroWire)



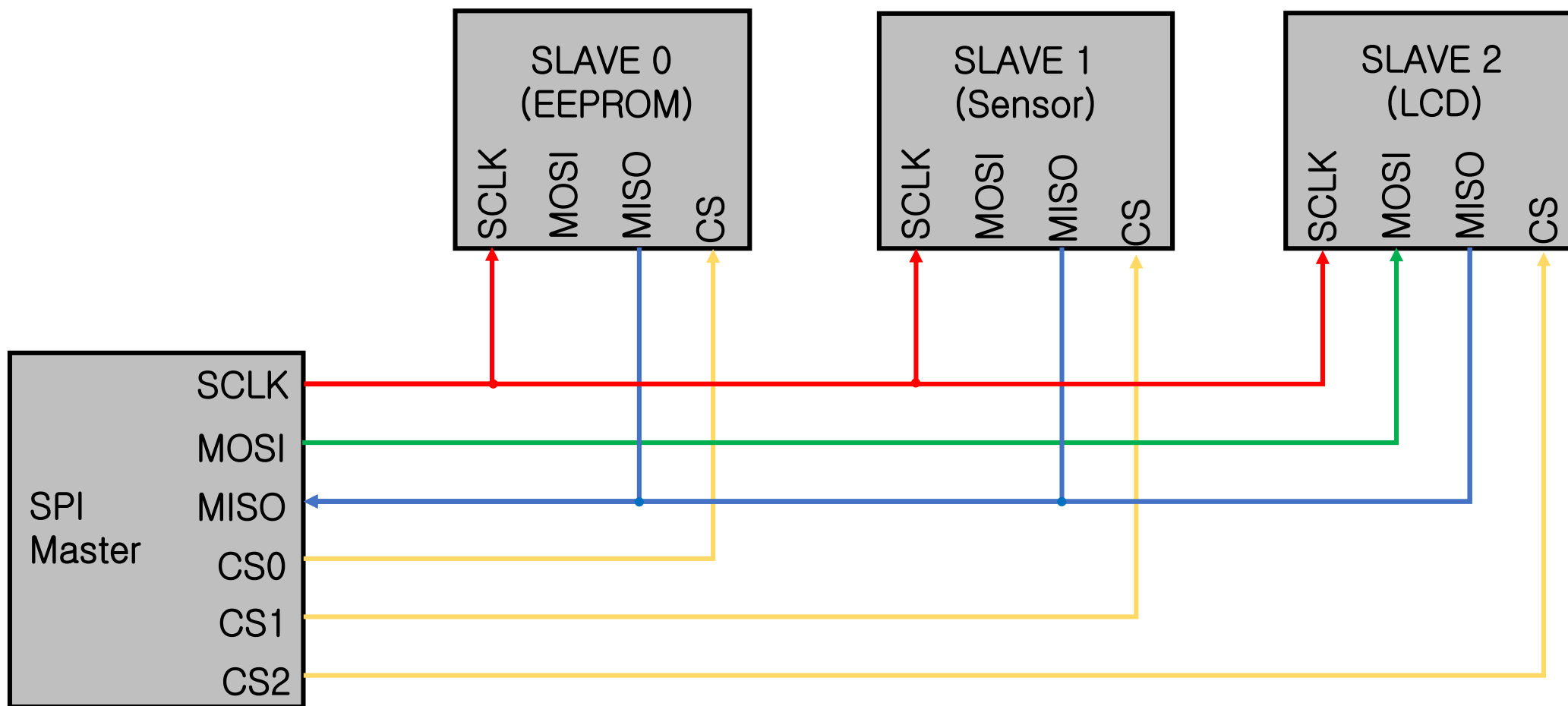
(c) 3-wire SPI (1:1 연결용)



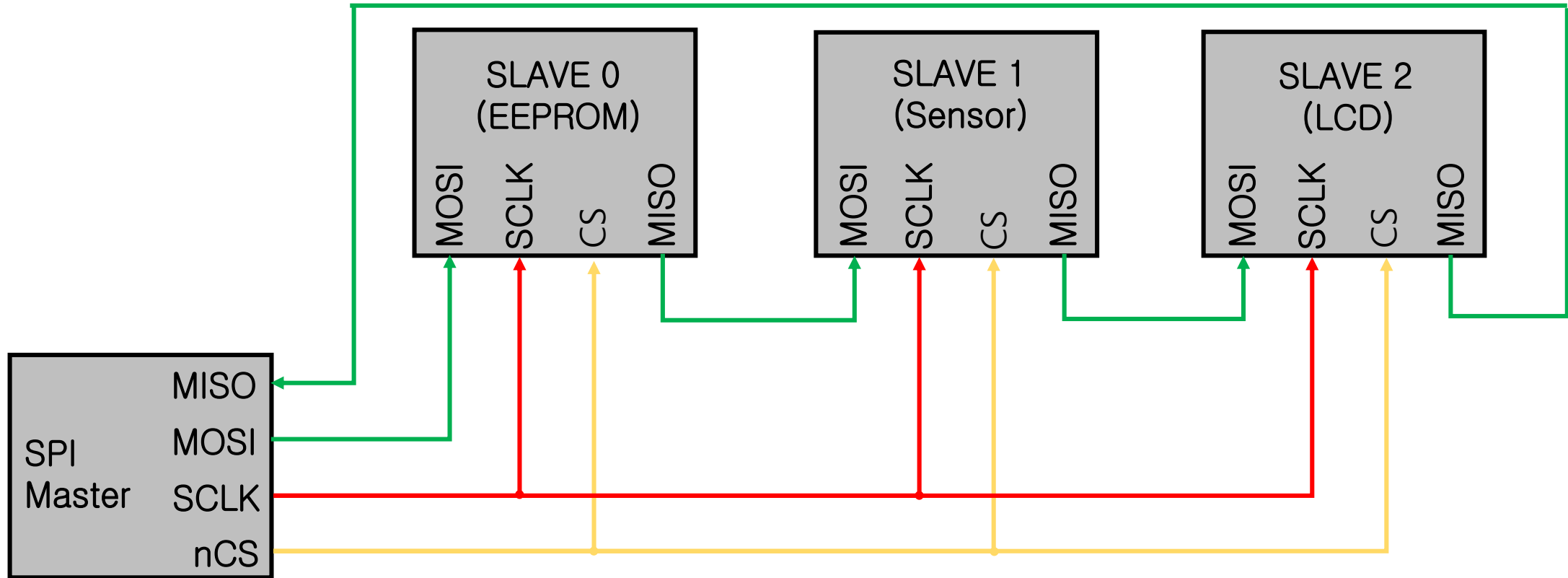


## 1-5. SPI 버스의 구성 (다중 SPI 슬레이브 장치를 연결하는 경우)

### 1) 개별적인 슬레이브 연결 구성 (Independent slave Configuration)

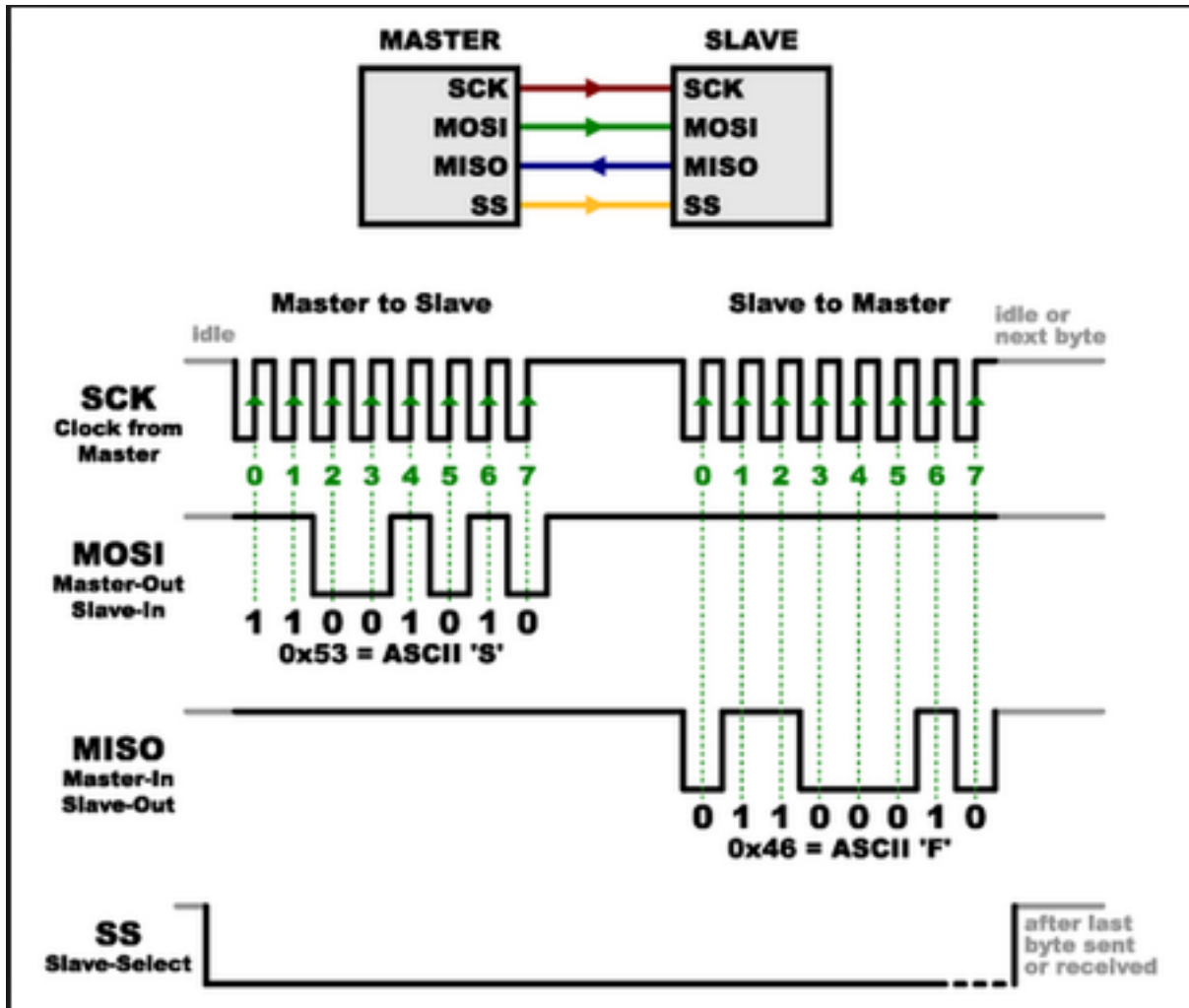


## 2) 데이터 체인 연결 구성 (Independent slave Configuration)



=> 1개의 Chip Select 선로만 사용하여도 다수의 슬레이브에게 데이터를 전달할 수 있음.

## 1-6. SPI 통신의 방식



- 마스터->슬레이브 : SS핀이 Low인 상태에서,  
클럭의 엣지마다 MOSI 핀을 통해 1bit씩 전송

- 슬레이브->마스터 : SS핀이 Low인 상태에서,  
클럭의 엣지마다 MISO 핀을 통해 1bit씩 전송

(SS가 Low로 내려가면, SS핀에 연결되어 있는 슬레이브와 통신을 하게 됨)

## *2. TMS570 보드의 SPI 통신*

## 2. TMS570 보드의 SPI 통신

### 2-1. SPI 특징

- 고속 동기식 통신.
- 2bits ~ 16bits 길이의 데이터 비트 스트림 전송, 프로그래밍으로 데이터 전송 속도 제어 가능.
- SPI통신은 shift registers, display drivers, analog-to-digital converters 와 같은 장치들을 이용해서 외부 I/O 또는 주변장치들과 인터페이스 함.
- TMS570 은 MibSPI를 지원한다. MibSPI 는 SPI의 확장이고, 두 가지 모드를 지원한다.
  - Compatibility Mode
  - Multi-buffer Mode

#### \* Compatibility Mode

- MibSPI 의 Compatibility mode는 SPI 와 100% 호환이 된다.

#### \* Multi-buffer Mode

- MibSPI의 Multi-buffer mode는 MibSPI에만 적용이 되고 SPI에는 적용이 되지 않는다.

#### \* MibSPI와 SPI의 차이는 SPI 는 각각의 슬레이브에 대한 설정을 할 수 없지만, MibSPI는 각각의 슬레이브에 대한 개별적인 설정이 가능하다.

- ex) 10MHZ와 5MHZ의 클럭 주파수를 지원하는 슬레이브 장치 2개가 마스터와 연결이 되어있다면, 각각의 슬레이브를 위한 개별적인 설정이 가능하다.

## 2-2. SPI 의 핀 설정

Table 28-1. Pin Configurations

| Pin     | Master Mode                           |   | Slave Mode                                  |  |
|---------|---------------------------------------|---|---|--|
| SPICLK  | Drives the clock to external devices  |   | Receives the clock from the external master |  |
| SPISOMI | Receives data from the external slave |   | Sends data to the external master           |  |
| SPISIMO | Transmits data to the external slave  |   | Receives data from the external master      |  |
| SPIENA  | SPIENA disabled:<br>GIO               | SPIENA enabled:<br>Receives ENA signal from<br>the external slave | SPIENA disabled:<br>GIO                     | SPIENA enabled:<br>Drives ENA signal from the<br>external master     |
| SPICS   | SPICS disabled:<br>GIO                | SPICS enabled:<br>Selects one or more slave<br>devices            | SPICS disabled:<br>GIO                      | SPICS enabled:<br>Receives the CS signal<br>from the external master |

**NOTE:**

1. When the SPICS signals are disabled, the chip-select field in the transmit data is not used.
2. When the SPIENA signal is disabled, the **SPIENA** pin is ignored in master mode, and not driven as part of the SPI transaction in slave mode.

Table 28-2. MibSPI/SPI Configurations

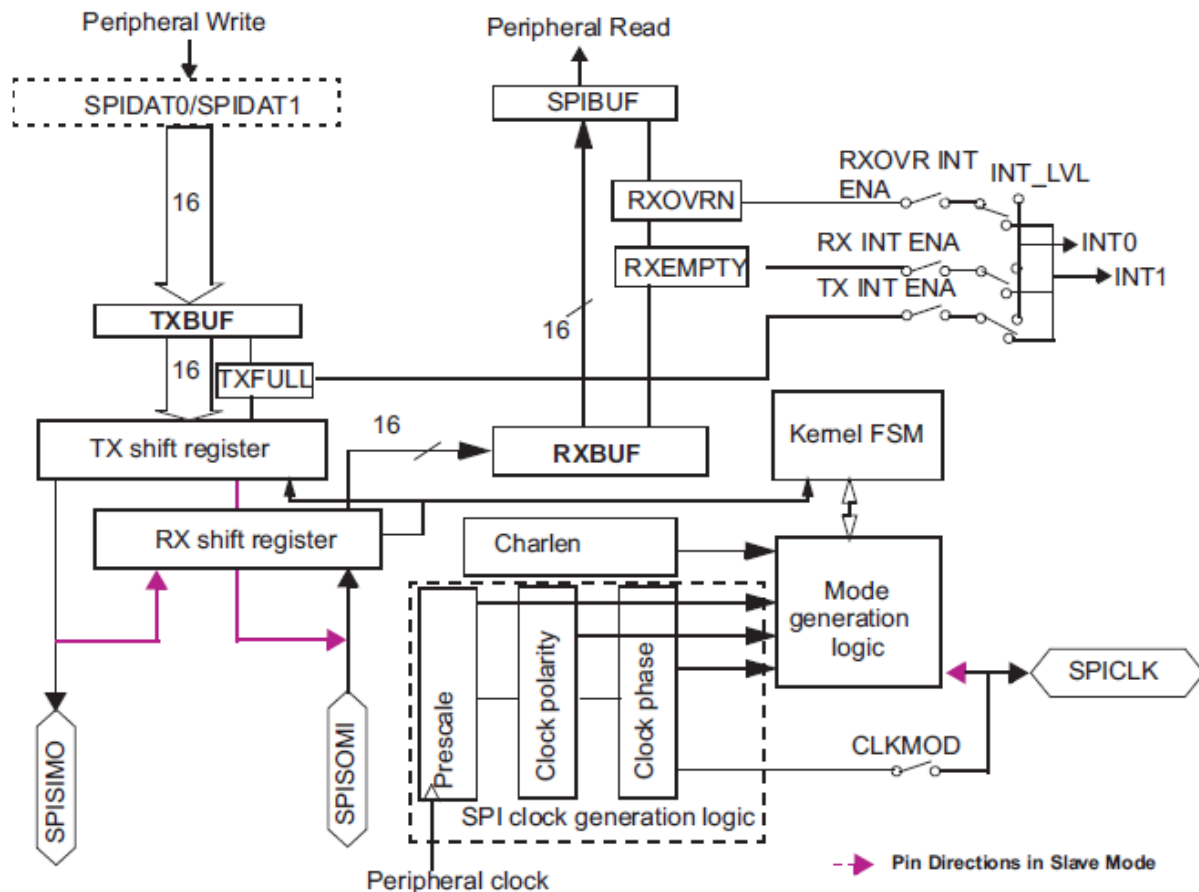
| MibSPIx/SPIx | I/Os   |
|--------------|--|
| MibSPI1      | MIBSPI1SIMO[1:0], MIBSPI1SOMI[1:0], MIBSPI1CLK, MIBSPI1nCS[5:0], MIBSPI1nENA |
| MibSPI2      | MIBSPI2SIMO[1:0], MIBSPI2SOMI[1:0], MIBSPI2CLK, MIBSPI2nCS[5:0], MIBSPI2nENA |
| MibSPI3      | MIBSPI3SIMO[1:0], MIBSPI3SOMI[1:0], MIBSPI3CLK, MIBSPI3nCS[5:0], MIBSPI3nENA |
| MibSPI4      | MIBSPI4SIMO[1:0], MIBSPI4SOMI[1:0], MIBSPI4CLK, MIBSPI4nCS[5:0], MIBSPI4nENA |
| MibSPI5      | MIBSPI5SIMO[1:0], MIBSPI5SOMI[1:0], MIBSPI5CLK, MIBSPI5nCS[5:0], MIBSPI5nENA |
| SPI1         | SPI1SIMO, ZSPI1SOMI, SPI1CLK, SPI2nCS[1:0], SPI1nENA                         |
| SPI2         | SPI2SIMO, ZSPI2SOMI, SPI2CLK, SPI2nCS[1:0], SPI2nENA                         |
| SPI3         | SPI3SIMO, ZSPI3SOMI, SPI3CLK, SPI3nCS[1:0], SPI3nENA                         |

## 2-3. SPI Mode의 동작

- SPI는 소프트웨어에 의해 마스터, 슬레이브 중에 어떤 것으로 동작할 것인지 설정 가능함.
- MASTER bit (SPIGCR1[0]) 는 SPISIMO 와 SPISOMI pins의 설정을 선택함.
- CLKMOD bit (SPIGCR1[1])은 내부 클럭, 외부 클럭 소스 중에 어떤 것을 사용할 것인지 결정함.
- SPICS 핀은 다중 슬레이브 장치, 또는 싱글 슬레이브 장치와 통신할 때 메시지의 범위를 정하기 위해 사용 가능함.
- master mode에서 SPIDAT1 레지스터에 데이터를 쓰면, SPICS 핀은 통신할 슬레이브를 선택하기 위해 자동으로 해당 슬레이브에 물림.
- SPIENA 핀에 의해 제공되는 Handshaking mechanism 은 슬레이브가 마스터로부터 데이터를 수신할 준비가 안되어 있다면 클럭 신호를 지연시킴.

## 2-4. SPI Mode 동작 블록 다이어그램

Figure 28-1. SPI Functional Logic Diagram



### \* 데이터 송신 과정

- TX shift register 와 TXBUF가 비어 있다면, SPIDAT의 데이터는 바로 TX shift register로 복사된다.
- TX shift register가 이미 꽉차있거나 shifting 과정이고, TXBUF가 비어있다면 SPIDAT의 데이터는 TXBUF로 복사되고 동시에 TXFULL flag는 1이 된다.
- 데이터의 shift 동작이 완료되면, TXBUF의 데이터는 TX shift register로 복사되고, TXFULL flag는 0으로 클리어 된다. (다음 데이터 fetch 가능)

### \* 데이터 수신 과정

- SPIBUF 와 RXBUF가 모두 비어있다면, RX shift register의 데이터는 SPIBUF로 복사된다. 동시에 SPIBUF의 RXEMPTY flag는 클리어 된다.
- SPIBUF가 이미 full 이라면, RX shift register의 데이터는 RXBUF로 복사된다.
- SPIBUF가 CPU 또는 DMA에 의해 read되면 RXBUF의 데이터는 SPIBUF로 복사된다.
- SPIBUF와 RXBUF가 모두 full 이면, RXBUF는 덮어 쓰여진다.

1 This is a representative diagram, which shows three-pin mode hardware.  
 2 TXBUF, RXBUF, and SHIFT\_REGISTER are user-invisible registers.  
 3 SPIDAT0 and SPIDAT1 are user-visible, and are physically mapped to the contents of TXBUF.  
 4 SPISIMO, SPISOMI, SPICLK pin directions depend on the Master or Slave Mode.

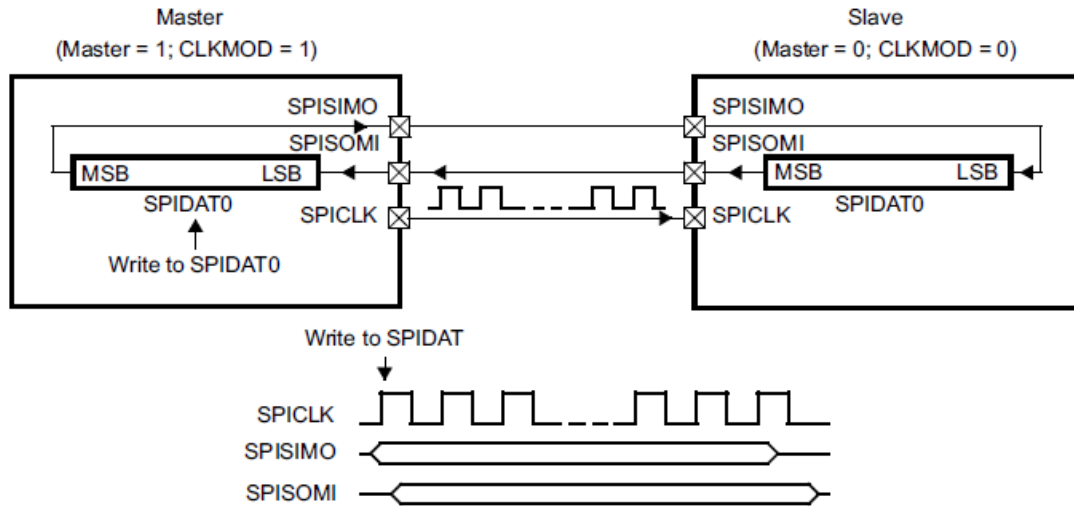


## 2-5. Physical Interface

### 2-5-1. Three-Pin mode

- \* 사용하는 핀
- SPISIMO
  - SPISOMI
  - SPICLK

Figure 28-6. SPI Three-Pin Operation



#### \* 마스터 모드

- 마스터 모드에서는 SPICLK 핀의 serial clock 신호를 출력으로 제공
- 데이터는 SPISIMO 핀에서 송신하고, SPISOMI 핀에서 수신함.
- 클럭의 엣지마다 데이터를 1bit씩 송,수신함.

#### \* 슬레이브 모드

- 슬레이브 모드에서는 SPICLK 핀은 serial clock 신호를 위한 입력으로 사용된다.  
(외부에 연결된 마스터에 의해 공급됨)
- SPISIMO 핀을 통해 읽혀진 데이터는 외부에 연결된 마스터에 의해 클럭을 수신하면, RX shift register로 이동한다.

## 2-5-2. Four-Pin Mode

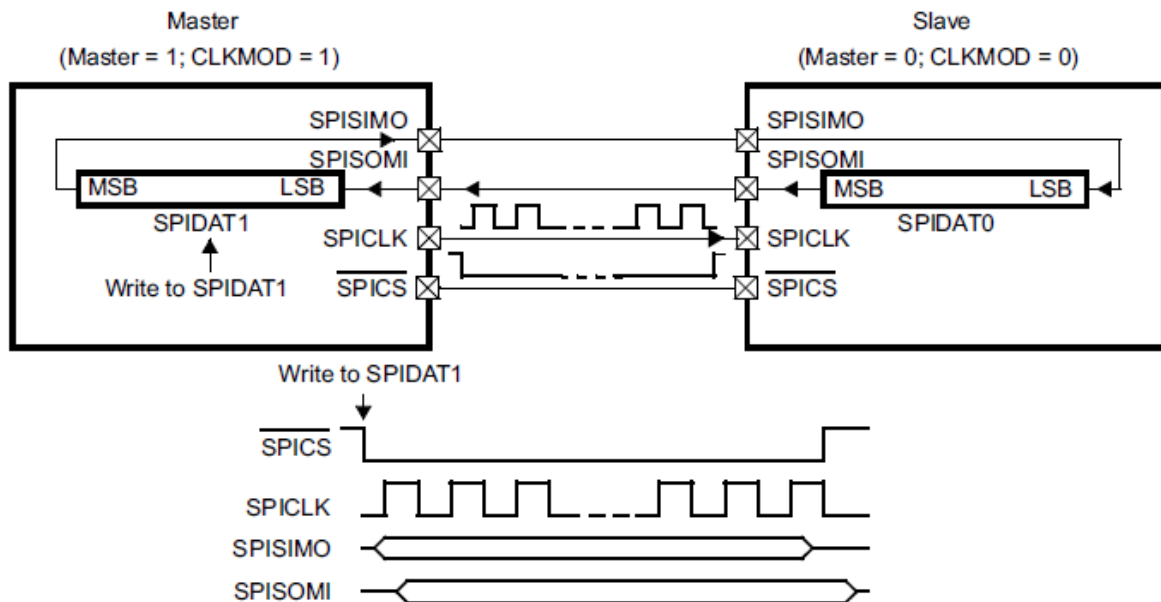
### 2-5-2-1. Four-Pin Option with SPICS

\* 사용하는 핀

- SPISIMO
- SPISOMI
- SPICLK
- SPICS

=> spi 통신을 4 핀을 사용해서 할 때는 SPICS핀을 사용하는 경우와 SPIENA 핀을 사용하는 경우 2가지가 있다.

Figure 28-7. Operation with SPICS



#### \* SPICS 핀

- 마스터 모드에서 chip select 신호는 슬레이브를 선택하기 위해 사용된다.
- 슬레이브 모드에서 chip select 신호는 통신을 enable 또는 disable 하기위해 사용됨.
- chip select 기능은 SPICS 핀들 중에 하나를 chip select 로 설정함으로써 enabled 됨.

#### \* Multiple Chip Selects (SPICS핀을 여러개 사용하는 경우)

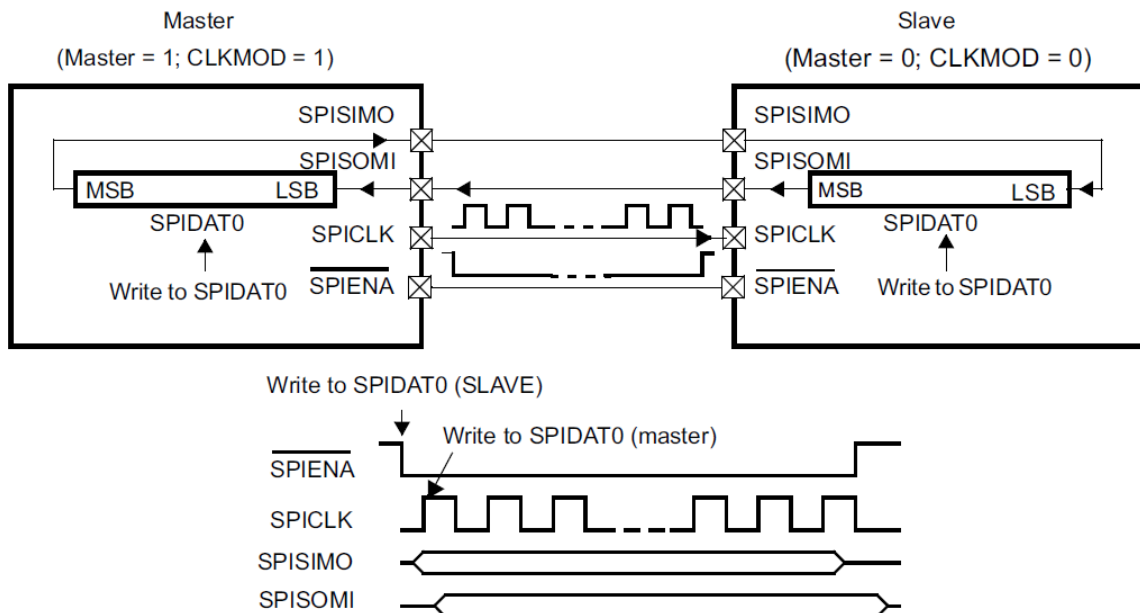
- SPICS 핀은 SPIPC0 레지스터에서 functional pins로 설정됨.
- SPIDEF 레지스터에서 SPICS 핀의 디폴트 패턴이 설정됨.
- 여러 슬레이브들이 각각 다른 chip-select polarity를 가지고 활성화되는 것을 허용한다.
- SPIDAT1레지스터의 CSNR 필드에서 SPICS 핀의 drive state 가 설정됨.
- 슬레이브 모드에서는 SPICS 입력 핀의 활성화 값 0에 의해 SPI 통신이 활성화됨.

## 2-5-2-2. Four-Pin Option with SPIENA

\* 사용하는 핀

- SPISIMO
- SPISOMI
- SPICLK
- SPIENA

Figure 28-8. Operation with SPIENA



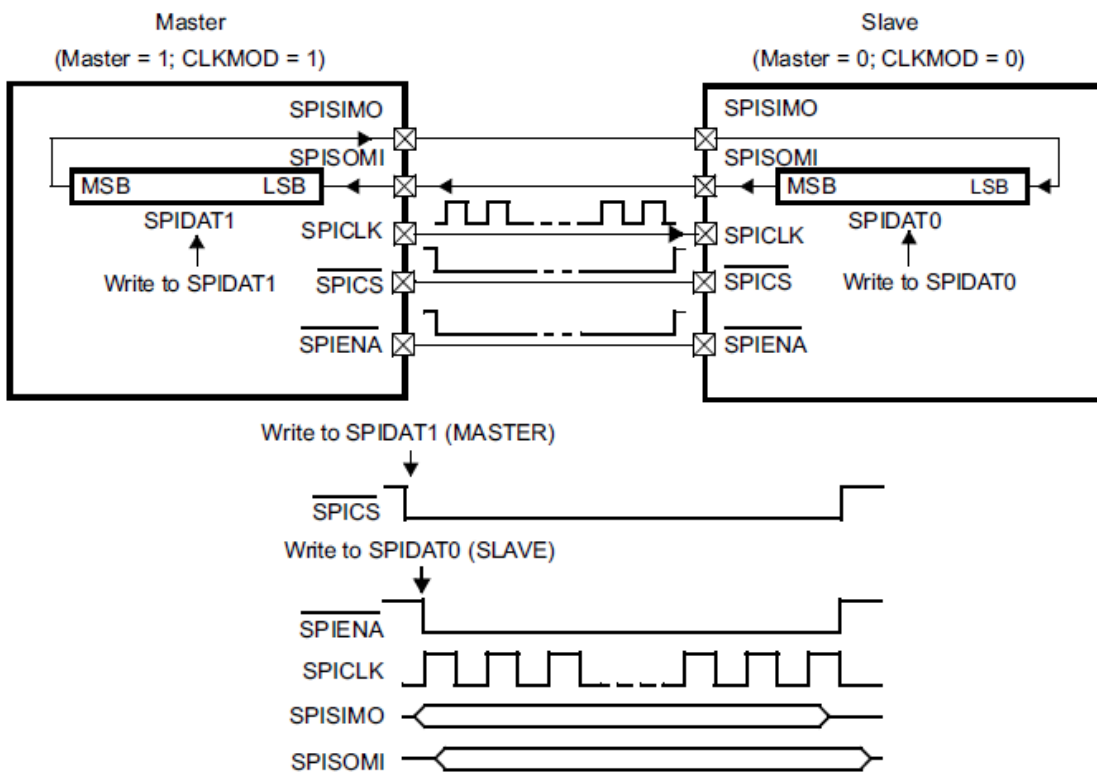
### \* SPIENA 핀

- SPIENA 핀은 WAIT 신호 핀으로 동작한다.
- SPIENA 핀 상의 슬레이브로부터의 active-low 신호는 마스터가 클럭 펄스 신호를 출력하는 것을 허용한다. (high 신호는 마스터가 클럭 신호를 중지하게 함)
- high-impedance 모드(ENABLE\_HIGHZ = 1)에서 슬레이브가 한 문자의 수신을 완료하면 SPIENA핀을 high-impedance 모드로 둔다.
- push-pull 모드(ENABLE\_HIGHZ = 0)에서 슬레이브가 한 문자의 수신을 완료하면 SPIENA핀에 1을 출력한다.
- 새로운 데이터가 슬레이브의 TX shift register에 쓰여진 후에, 슬레이브는 다음 데이터의 전송을 위해 SPIENA 핀을 다시 low로 내린다.
- 마스터 모드에서 SPIENA 핀이 functional로 설정되면, 입력 핀으로 동작한다.
- 슬레이브 모드에서 SPIENA 핀이 functional로 설정되면, 출력 핀으로 동작한다.

### 2-5-3. Five-Pin Operation (Hardware Handshaking)

- \* 사용하는 핀
- SPISIMO
- SPISOMI
- SPICLK
- SPICS
- SPIENA

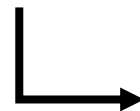
Figure 28-9. SPI Five-Pin Option with  $\overline{\text{SPIENA}}$  and  $\overline{\text{SPICS}}$



=> Five-pin 동작은 three-pin 모드의 기능과 enable pin, chip-select pins를 제공한다.

#### \* SPIENA 핀

- High-impedance 모드(ENABLE\_HIGHZ = 1) 에서 슬레이브는 디폴트로 SPIENA 핀을 high-impedance로 한다.
- push-pull 모드(ENABLE\_HIGHZ = 0) 에서 슬레이브는 디폴트로 SPIENA 핀을 high로 올린다.
- 2가지 모드에서 슬레이브의 shift register에 새로운 데이터가 쓰여지면, SPIENA핀을 low로 내려간다.
- 슬레이브가 마스터에 의해 선택되지 않으면, 슬레이브는 SPIENA 핀을 high로 출력한다.
- 다중 슬레이브 장치들이 공통 SPIENA핀에 연결되어 있을때, 모든 슬레이브들은 각각의 SPIENA핀을 high-impedance 모드로 설정해야 한다.



**\* SPICS 핀**

- 마스터 모드에서 SPICS 핀은 functional 핀으로 동작함. (핀은 출력 모드)
  - 마스터의 SPIDAT1/SPIDAT0 레지스터에 데이터를 쓰면, 자동으로 SPICS신호를 low로 내림.
  - 마스터는 데이터의 전송을 완료한 후에, SPICS 핀을 다시 high로 올림.
  - 슬레이브 모드에서 SPICS 핀은 SPI functional inputs으로 동작함.
-

## ***2-6. Advanced Module Configuration Options***

### ***2-6-1. Data Formats***

- SPIDAT1 레지스터의 DFSEL[1:0] 에서 전송할 데이터의 포맷을 설정할 수 있다. (데이터 포맷 0,1,2,3)
  - shift direction은 데이터의 MSB와 LSB중 어느 것을 먼저 보낼지 선택하기 위해 사용될 수 있다.
  - 수신된 데이터는 shift direction과 data word length에 무관하게 자동으로 저장되고, 데이터를 송신할 때는 내부 shift register가 설정된 shift direction과 data word length에 따라 데이터를 전송한다.
  - 송수신된 데이터의 에러 검출을 하기 위해, 홀수 또는 짝수 패리티 비트가 데이터 워드 끝에 추가될 수 있다.
  - 마스터는 같은 슬레이브에게 2번의 연속적인 접근을 할 수 있기 때문에, 8-bit delay counter 는 슬레이브가 데이터를 갱신하는 데 필요한 delay time을 제공하는 데 사용될 수 있다.
  - CHARLEN[4:0]는 data word의 bits 수(2 to 16)를 결정한다.
  - 송,수신되는 데이터의 길이는 마스터와 슬레이브에게 같은 길이로 프로그램되어야 한다.
- 단, 여러 개의 chip-selects가 사용될 때, 다른 길이의 데이터 설정을 가진 여러 개의 슬레이브가 존재할 수 있다.

\* 데이터의 송,수신시 포맷

**NOTE:** Data must be right-justified when it is written to the SPI for transmission irrespective of its character length or word length.

Figure 28-10 shows how a 12-bit word (0xEC9) needs to be written to the transmit buffer to be transmitted correctly.

Figure 28-10. Format for Transmitting an 12-Bit Word

송신 →

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| x   | x   | x   | x   | 1   | 1   | 1  | 0  | 1  | 1  | 0  | 0  | 1  | 0  | 0  | 1  |

**NOTE:** The received data is always stored right-justified regardless of the character length or direction of shifting and is padded with leading 0s when the character length is less than 16 bits.

Figure 28-11 shows how a 10-bit word (0x0A2) is stored in the buffer once it is received.

Figure 28-11. Format for Receiving an 10-Bit Word

수신 →

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0  | 1  | 0  | 1  | 0  | 0  | 0  | 1  | 0  |

## 2-6-2. Clocking Modes

=> SPICLK 핀은 클럭의 phase, polarity에 따라 4가지 모드로 동작하며, 이러한 4가지 옵션은 SPI가 다양한 종류의 시리얼 장치와 인터페이스가 가능하게 만들어준다.

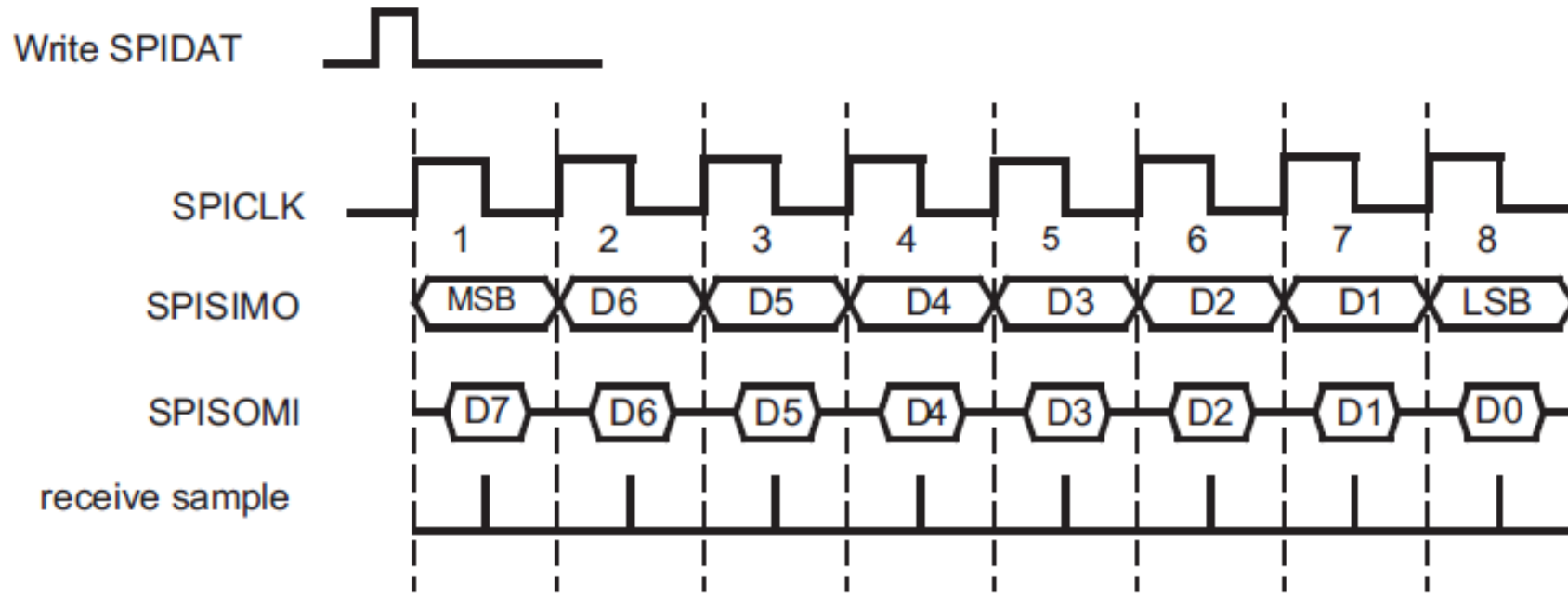
**Table 28-3. Clocking Modes**

| POLARITY | PHASE | Action   |
|----------|-------|--|
| 0        | 0     | Data is output on the rising edge of SPICLK. Input data is latched on the falling edge.  |
| 0        | 1     | Data is output one half-cycle before the first rising edge of SPICLK and on subsequent falling edges. Input data is latched on the rising edge of SPICLK.  |
| 1        | 0     | Data is output on the falling edge of SPICLK. Input data is latched on the rising edge.  |
| 1        | 1     | Data is output one half-cycle before the first falling edge of SPICLK and on subsequent rising edges. Input data is latched on the falling edge of SPICLK. |



1) *Polarity = 0 and Phase = 0*

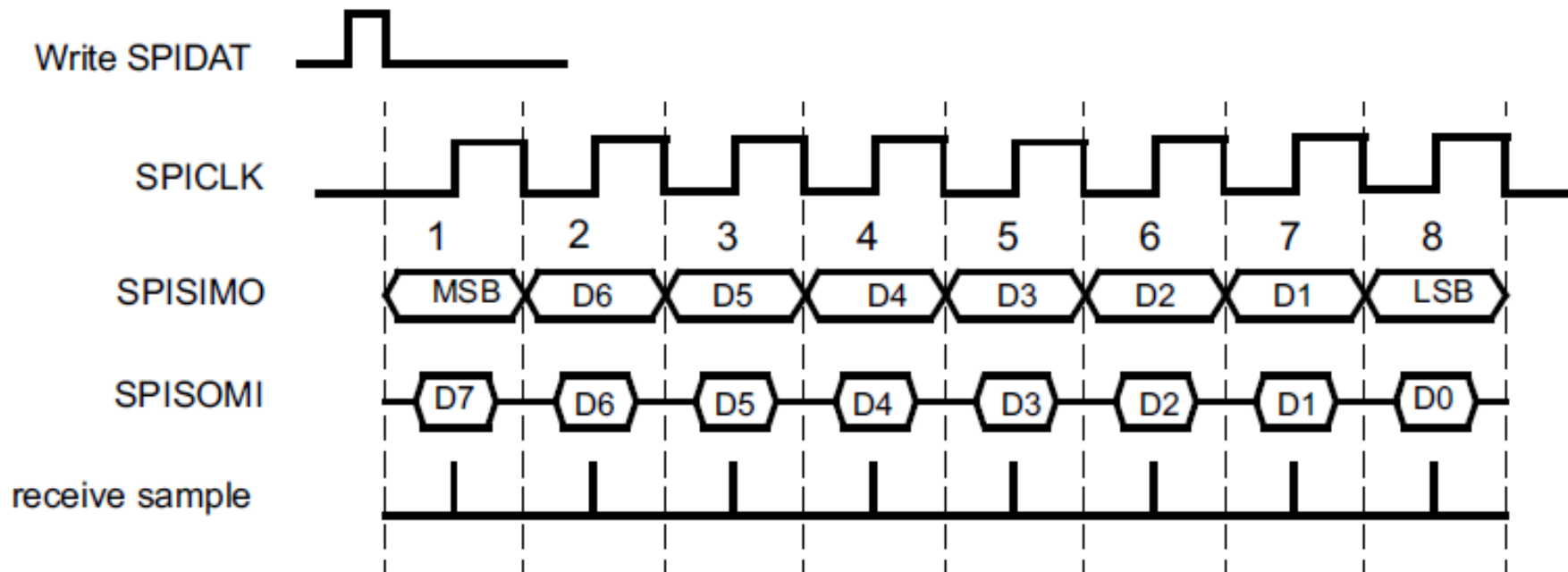
Figure 28-12. Clock Mode with Polarity = 0 and Phase = 0



Data is output on the rising edge of SPICLK.  
Input data is latched on the falling edge of SPICLK.

## 2) Polarity = 0 and Phase = 1

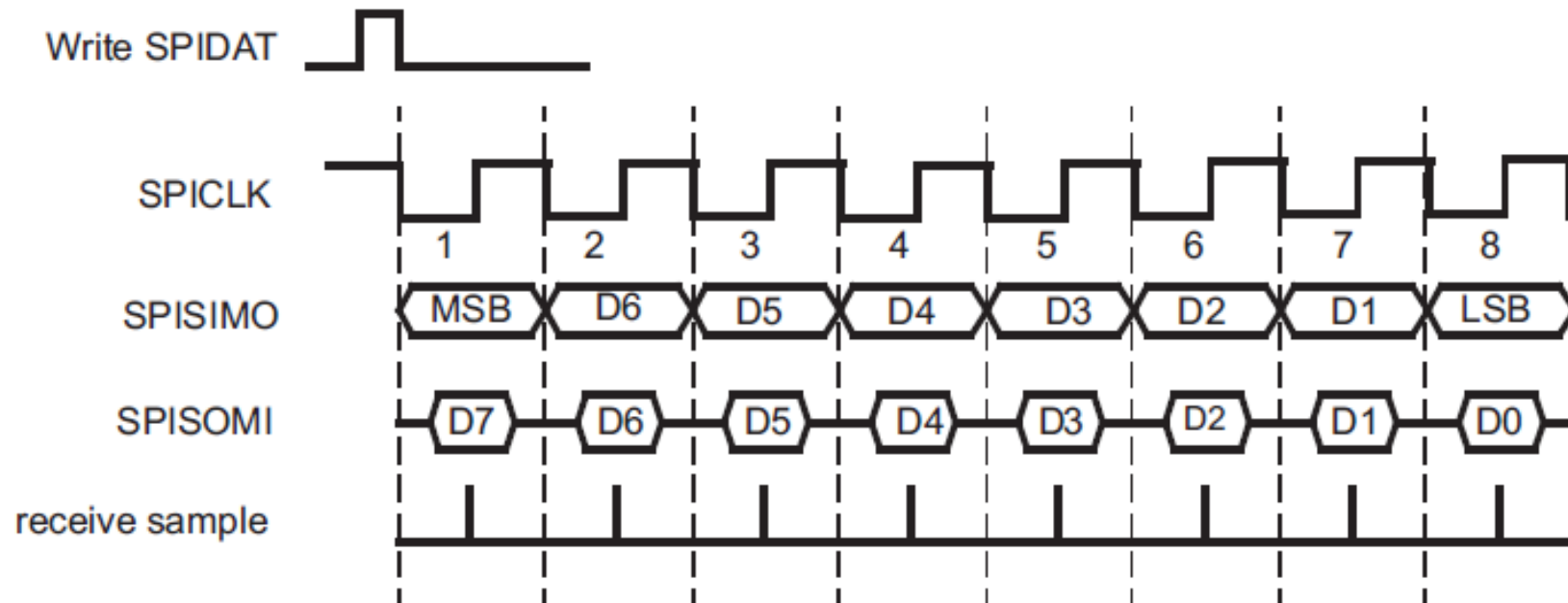
Figure 28-13. Clock Mode with Polarity = 0 and Phase = 1



Data is output one-half cycle before the first rising edge of SPICLK and on subsequent falling edges of SPICLK  
Input data is latched on the rising edge of SPICLK

### 3) *Polarity = 1 and Phase = 0*

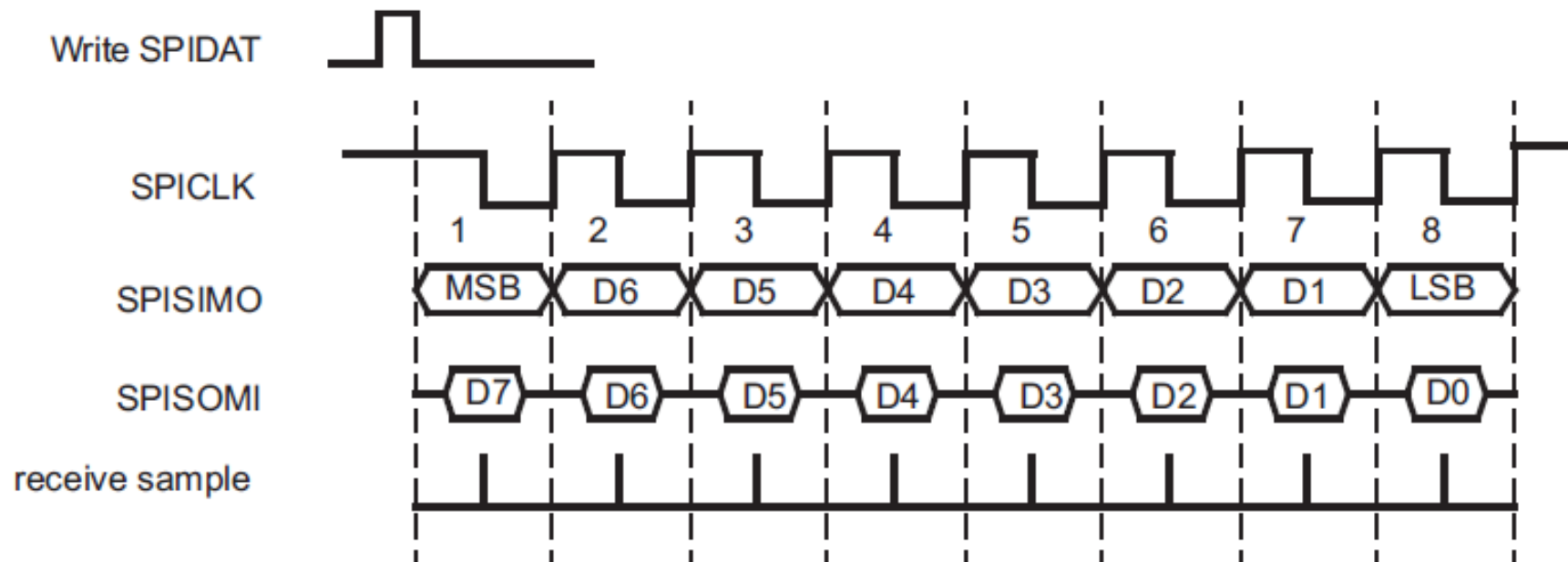
Figure 28-14. Clock Mode with **Polarity = 1** and **Phase = 0**



Data is output on the falling edge of SPICLK.  
Input data is latched on the rising edge of SPICLK.

#### 4) *Polarity = 1 and Phase = 1*

Figure 28-15. Clock Mode with **Polarity = 1** and **Phase = 1**

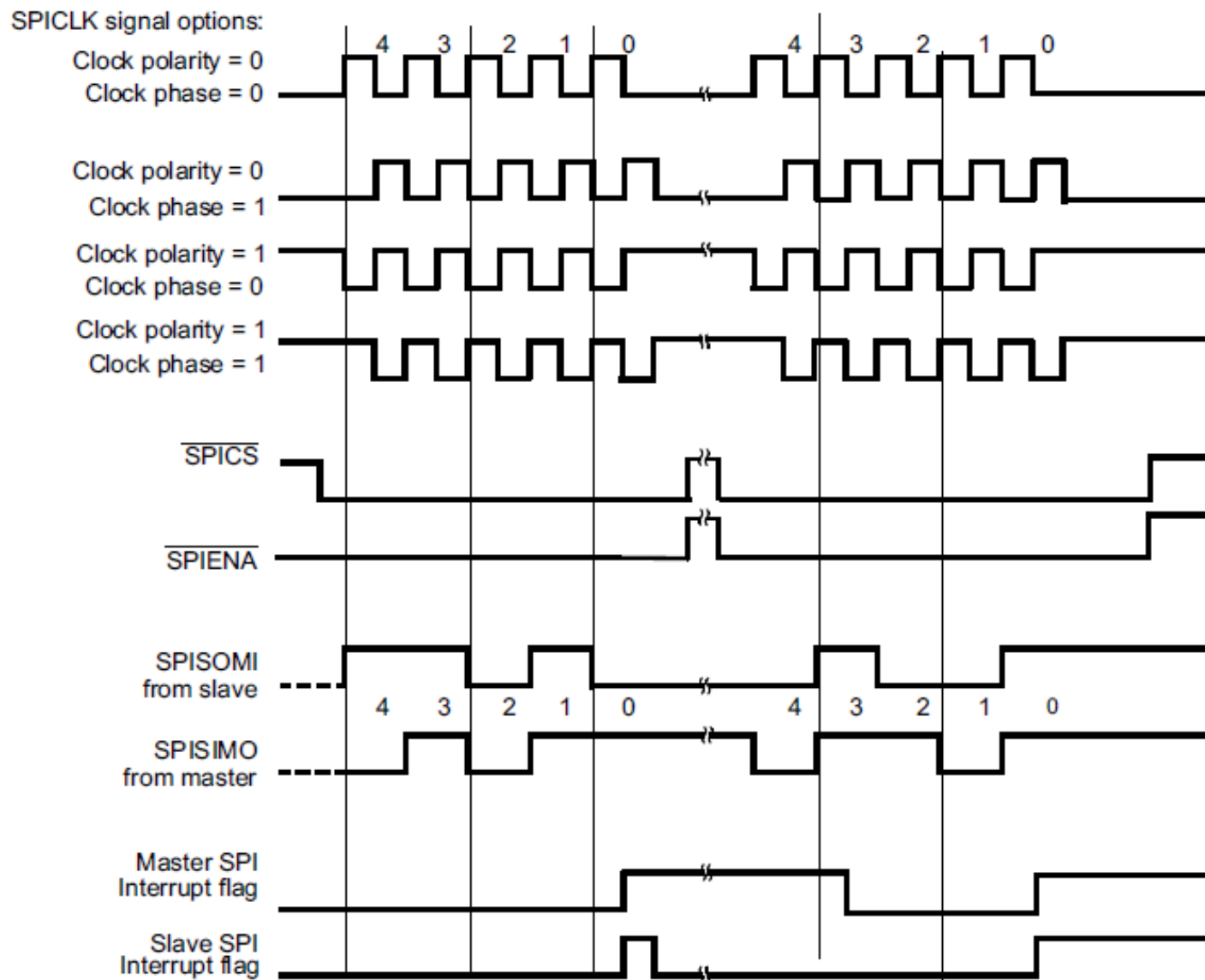


Data is output one-half cycle before the first falling edge of SPICLK and on the subsequent rising edges of SPICLK.  
Input data is latched on the falling edge of SPICLK.

\* 5-Pin Option 모드의  
타이밍 다이어그램

Figure 28-16 illustrates a SPI data transfer between two devices using a character length of five bits.

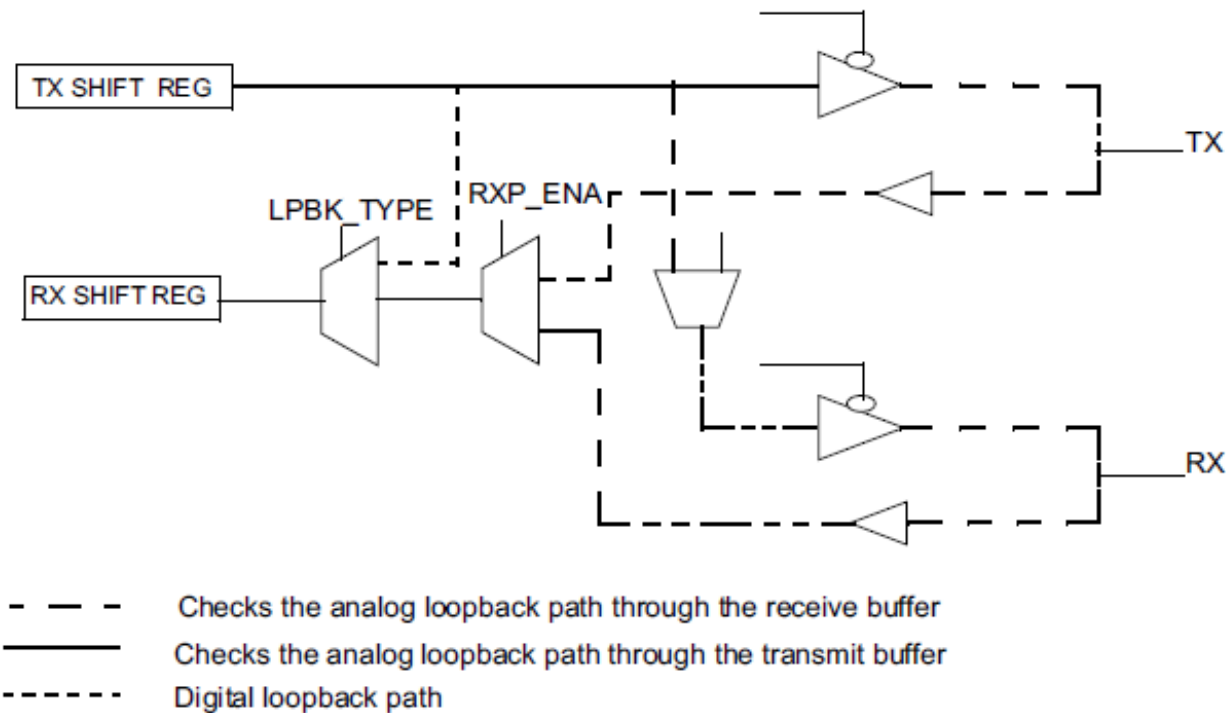
Figure 28-16. Five Bits per Character (5-Pin Option)



### 2-6-3. Input/Output Loopback Test Mode

- Input/Output Loopback Test 모드는 외부 인터페이스 없이 모든 Input/Output 핀 및 SPI에 대한 테스트를 지원한다.
- CPU가 transmit buffer에 데이터를 쓰고, receive buffer가 올바른 transmit data를 갖고 있는지 체크할 수 있다.
- transmit data는 receive-data line를 통해 다시 fed back 된다.

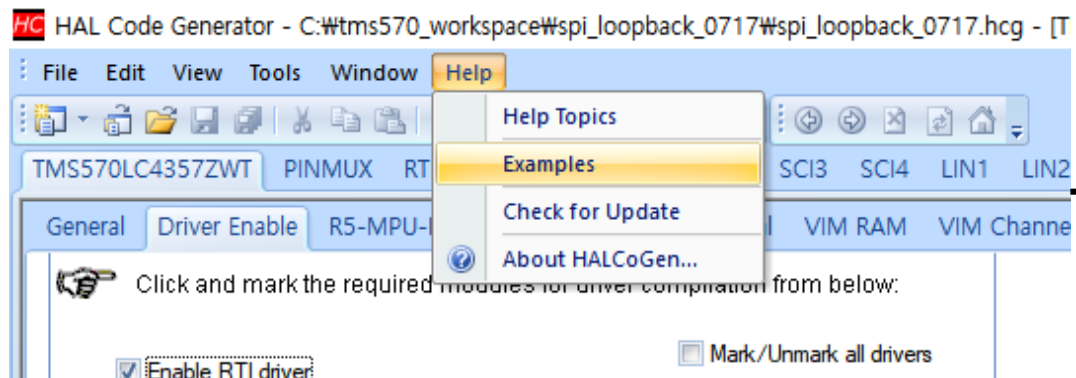
Figure 28-31. I/O Paths During I/O Loopback Modes



This diagram is intended to illustrate loopback paths and therefore may omit some normal-mode paths.

### *3. TMS570보드 기반의 SPI 통신 구현*

### 3-1. Loopback Test



| 이름                  | 수정된 날짜           | 유형    | 크기 |
|---------------------|------------------|-------|----|
| RM42x_41x           | 2018-05-03 오후... | 파일 폴더 |    |
| RM44x               | 2018-05-03 오후... | 파일 폴더 |    |
| RM46x               | 2018-05-03 오후... | 파일 폴더 |    |
| RM48x               | 2018-05-03 오후... | 파일 폴더 |    |
| RM57Lx              | 2018-05-03 오후... | 파일 폴더 |    |
| TMS470M             | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LC43x         | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LS04x_03x_02x | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LS09x_07x     | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LS12x_11x     | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LS20x         | 2018-05-03 오후... | 파일 폴더 |    |
| TMS570LS31x_21x     | 2018-05-03 오후... | 파일 폴더 |    |

문서  
ILC43x

|   |                  |      |      |
|---|------------------|------|------|
| example_i2cMaster_TX_RX                 | 2017-08-14 오후... | C 파일 | 6KB  |
| example_i2cSlave_TX_RX                  | 2017-08-14 오후... | C 파일 | 5KB  |
| example_Lin_Digital_Loopback            | 2017-08-14 오후... | C 파일 | 5KB  |
| example_Lin_Digital_Loopback_interrupts | 2017-08-14 오후... | C 파일 | 5KB  |
| example_mibspi_loopback                 | 2017-08-14 오후... | C 파일 | 6KB  |
| example_mibspi_trigger_tick             | 2017-08-14 오후... | C 파일 | 7KB  |
| example_mibspiDMA                       | 2017-08-14 오후... | C 파일 | 10KB |
| example_rtiBlinky                       | 2017-08-14 오후... | C 파일 | 6KB  |
| example_SafetyLib                       | 2017-08-24 오후... | C 파일 | 32KB |
| example_sci_dma                         | 2017-08-14 오후... | C 파일 | 9KB  |
| example_sci_uart_9600                   | 2017-08-14 오후... | C 파일 | 5KB  |
| example_sys_cache                       | 2017-08-14 오후... | C 파일 | 23KB |
| example_sys_cache                       | 2017-08-14 오후... | H 파일 | 5KB  |
| example_TI_Fee_Write_Read               | 2017-08-14 오후... | C 파일 | 6KB  |

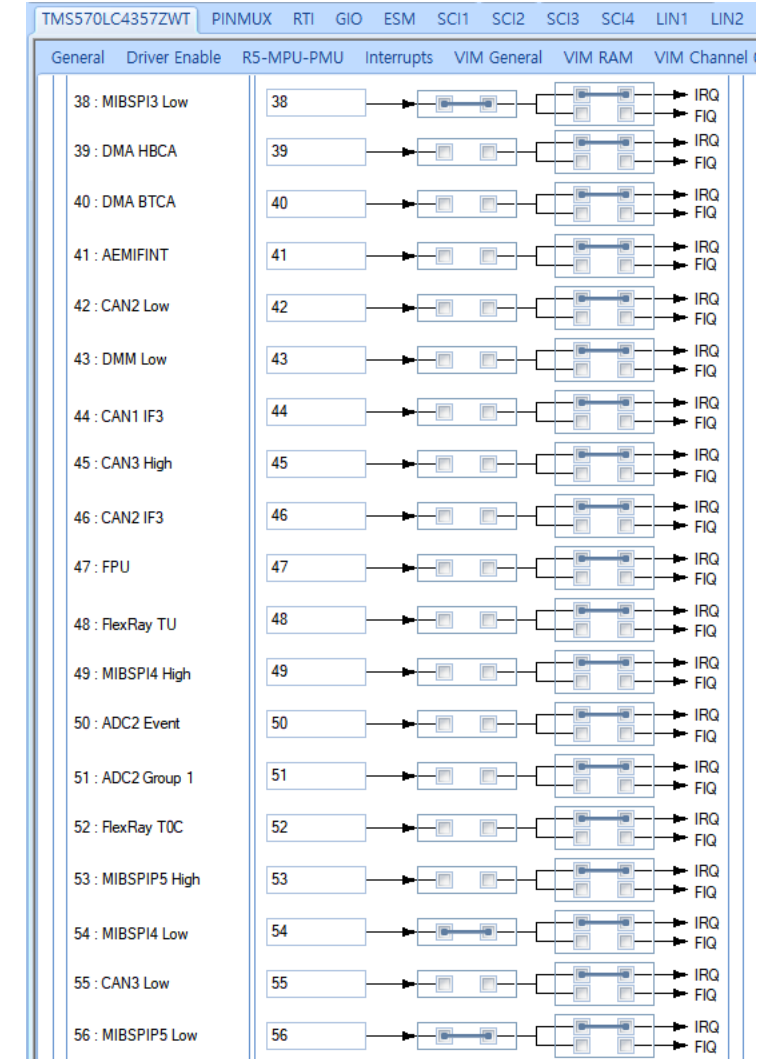
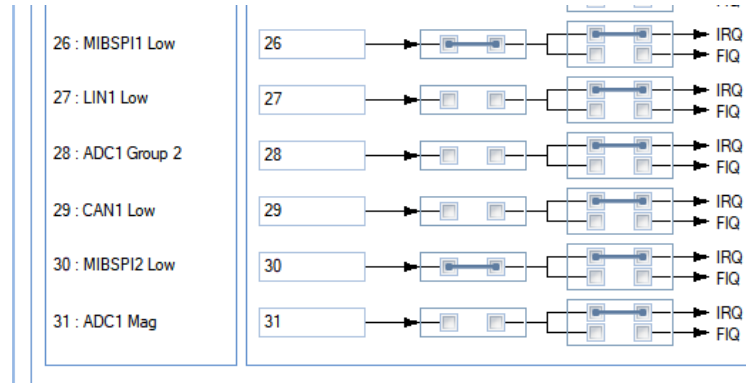
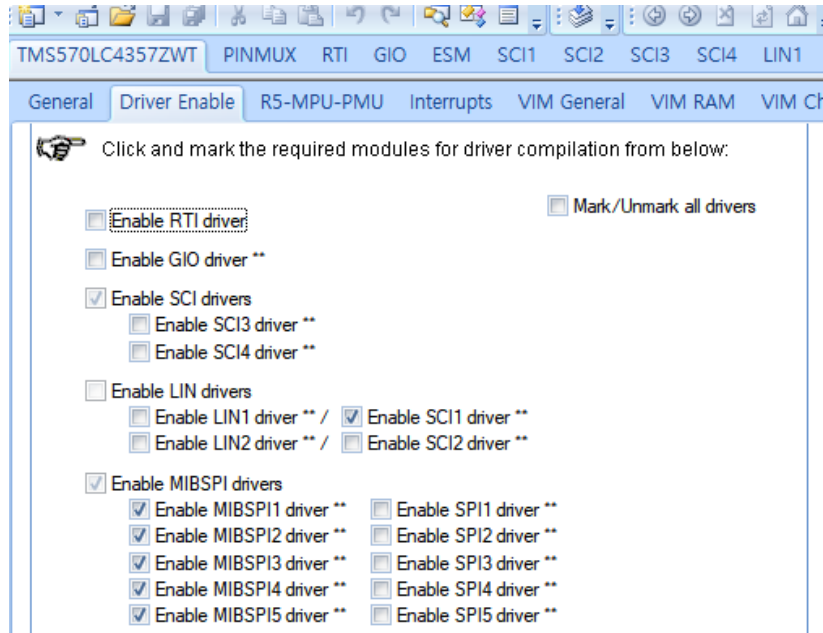
유형: C 파일  
크기: 5.62KB  
수정된 날짜: 2017-08-14 오후 2:30

=> HALCOGEN에서 제공하는 예제 코드를 사용함!



## 3-1. Loopback Test

### 3-1-1. HALCOGEN 설정



=> Loopback을 테스트 해보기 위한 MIBSPI1~5와 디버깅을 위해 SCI1을 Enable 한다.

=> MIBSPI1~5 의 VIM Channel configuration을 Low로 설정한다.  
=> 나머지 세부설정은 디폴트 값으로 그대로 둔다.

## 3-1-2. 코드

```
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_mibspi.h"
#include "HL_esm.h"
#include "HL_sci.h"
#include "HL_sys_core.h"
#include <string.h>
```

```
#define D_COUNT 20
```

```
uint32 cnt=0, error=0, tx_done=0;
uint8 tx_data1[D_COUNT] = "mibspiREG1WrWn";
uint8 tx_data2[D_COUNT] = "mibspiREG2WrWn";
uint8 tx_data3[D_COUNT] = "mibspiREG3WrWn";
uint8 tx_data4[D_COUNT] = "mibspiREG4WrWn";
uint8 tx_data5[D_COUNT] = "mibspiREG5WrWn";
uint16 rx_data1[D_COUNT] = {0};
uint16 rx_data2[D_COUNT] = {0};
uint16 rx_data3[D_COUNT] = {0};
uint16 rx_data4[D_COUNT] = {0};
uint16 rx_data5[D_COUNT] = {0};
```

*// 전송할 데이터*

*// Loopback에 의해 수신할 변수*

```
void main(void)
```

```
{
```

```
    _enable_IRQ_interrupt();
```

```
    scilnit();
```

```
    mibspiInit();
```

```
    mibspiEnableLoopback(mibspiREG1, Digital_Lbk);
```

```
// mibspi Loopback 을 Enable 시킴.
```

```
    mibspiEnableLoopback(mibspiREG2, Digital_Lbk);
```

```
    mibspiEnableLoopback(mibspiREG3, Digital_Lbk);
```

```
    mibspiEnableLoopback(mibspiREG4, Digital_Lbk);
```

```
    mibspiEnableLoopback(mibspiREG5, Digital_Lbk);
```

```
    mibspiEnableGroupNotification(mibspiREG1, 0, 1);
```

```
// mibspi의 송신 완료 인터럽트를 Enable 시킴.
```

```
    mibspiEnableGroupNotification(mibspiREG2, 0, 1);
```

```
    mibspiEnableGroupNotification(mibspiREG3, 0, 1);
```

```
    mibspiEnableGroupNotification(mibspiREG4, 0, 1);
```

```
    mibspiEnableGroupNotification(mibspiREG5, 0, 1);
```

```
    mibspiSetData(mibspiREG1, 0, &tx_data1[0]);
```

```
// mibspi의 송신 그룹의 버퍼에 송신할 데이터를 셋팅함.
```

```
    mibspiSetData(mibspiREG2, 0, &tx_data2[0]);
```

```
    mibspiSetData(mibspiREG3, 0, &tx_data3[0]);
```

```
    mibspiSetData(mibspiREG4, 0, &tx_data4[0]);
```

```
    mibspiSetData(mibspiREG5, 0, &tx_data5[0]);
```

```
    mibspiTransfer(mibspiREG1, 0);
```

```
// mibspi 의 버퍼에 있는 데이터를 송신함.
```

```
    mibspiTransfer(mibspiREG2, 0);
```

```
    mibspiTransfer(mibspiREG3, 0);
```

```
    mibspiTransfer(mibspiREG4, 0);
```

```
    mibspiTransfer(mibspiREG5, 0);
```

```
    while(1); // 대기
```

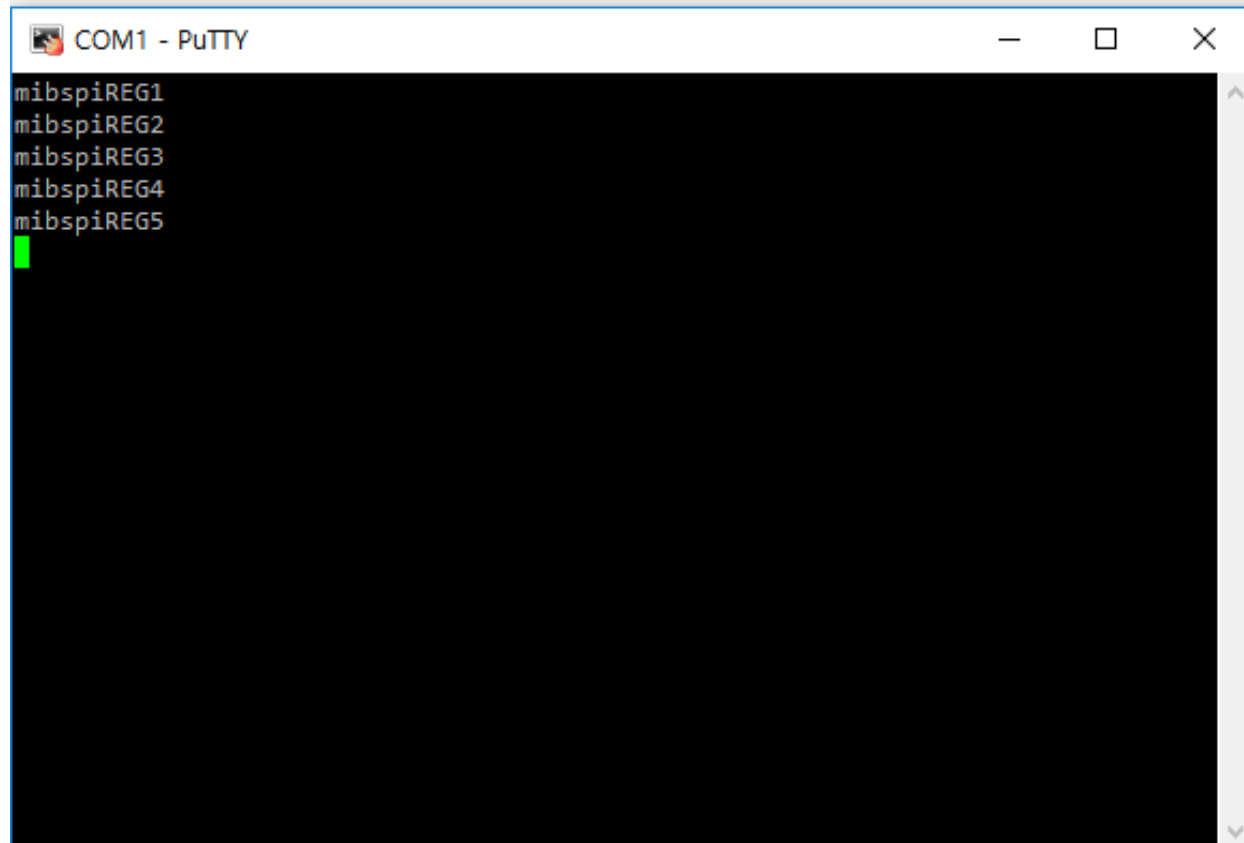
```
}
```

```
void mibspiGroupNotification(mibspiBASE_t *mibspi, uint32 group)    // mibspi 송신 완료 인터럽트
{
    uint16 * data;

    if(mibspi == mibspiREG1)    // mibspi의 종류에 따라 분류함
        data = &rx_data1[0];
    if(mibspi == mibspiREG2)
        data = &rx_data2[0];
    if(mibspi == mibspiREG3)
        data = &rx_data3[0];
    if(mibspi == mibspiREG4)
        data = &rx_data4[0];
    if(mibspi == mibspiREG5)
        data = &rx_data5[0];

    mibspiGetData(mibspi, group, data);    // 송신한 데이터를 수신함.
    sciSend(sciREG1, strlen(data), data);    // 수신한 데이터를 디버깅하기 위해 터미널 창에 출력함.
}
```

### 3-1-3. 출력



```
COM1 - PuTTY
mibspiREG1
mibspiREG2
mibspiREG3
mibspiREG4
mibspiREG5
█
```

=> 송신한 mibspiREG1~5가 그대로 수신되는 것이 확인됨.

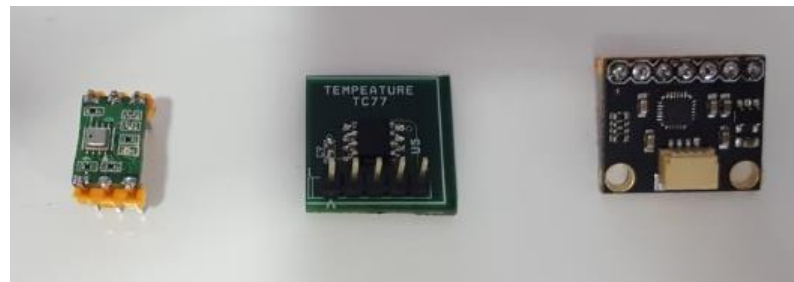
**\* 문제점**

=> CORTEX R5F MCU가 고성능 MCU라 옵션이 많아서 데이터 시트를 읽어도 이해가 쉽지 않음.  
(mibspi vs spi)

**\* 다음 주 계획**

=> 모듈을 이용해서 다중 슬레이브 제어하기!

=> eQEP, eCAP 학습 및 구현하기!



좌측 부터

- OSTSen-E280 (습도, 압력, 온도 센서 모듈)
- TC77(온도 센서)
- MPU9250 (9축 자이로/가속도/지자기 센서 모듈)