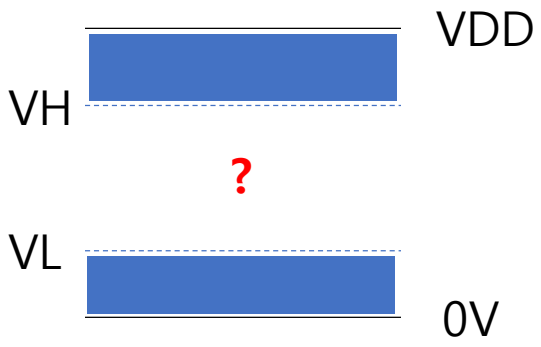
4. ADC

4. ADC

- **ADC(Analog to Digital Converter)**
- 아날로그 전압을 디지털 값으로 바꿔주는 전압 측정 장치이다.
- **아날로그 기준 전위에** 대한 값을 반환한다.

Q. 아날로그 기준 전위는 왜 중요한가?

기준 전위(VDD)가 바뀌어도 결과값에 큰 영향을 주지 않는다.



디지털

기준 전위(VDDA)가 바뀌면 결과값도 비례해서 바뀐다.

$$\frac{1}{3.3} = 0.303$$

$$\frac{1}{3.4} = 0.294$$

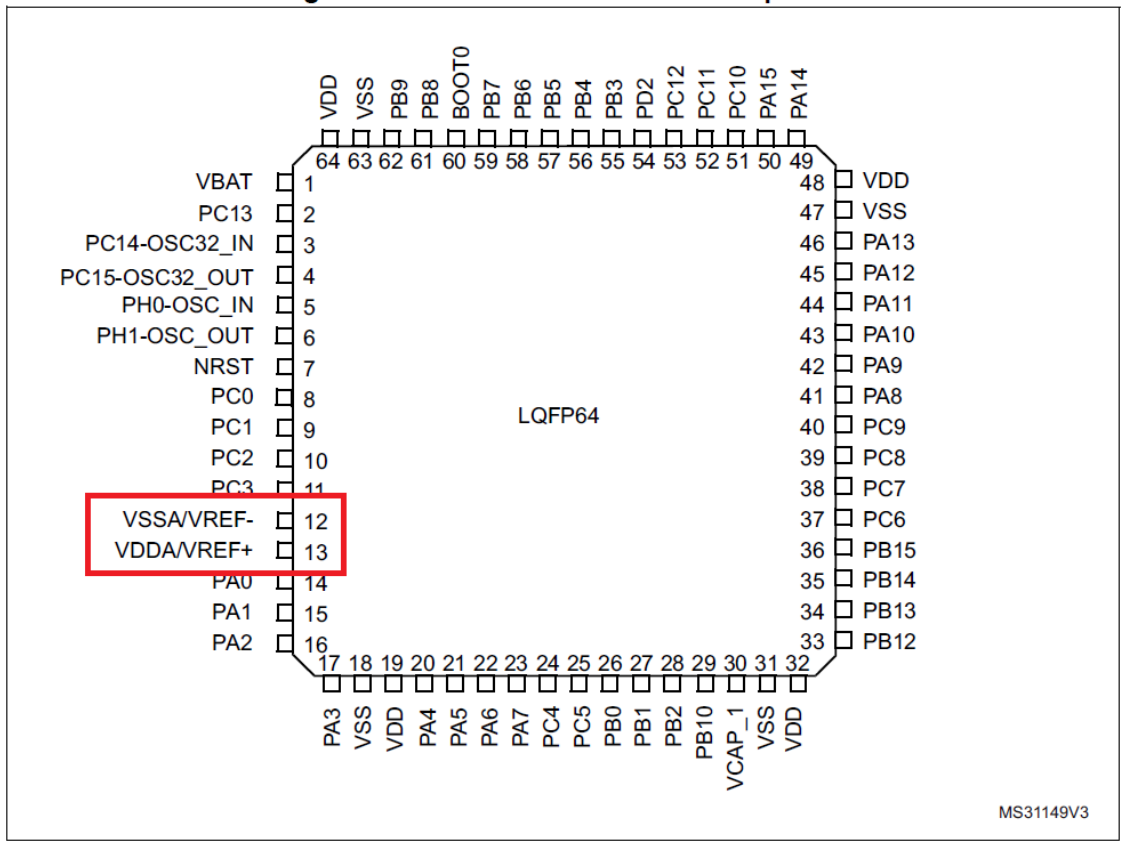
아날로그

4. ADC

Q. 아날로그 기준 전위는 왜 중요한가?

- 따라서 대부분의 칩들은 아날로그 기준 전위에 대한 **별도 핀이 존재**한다.

Figure 10. STM32F446xC/xE LQFP64 pinout



1. The above figure shows the package top view.

4. ADC

Q. ADC의 성능은 무엇으로 결정되는가?

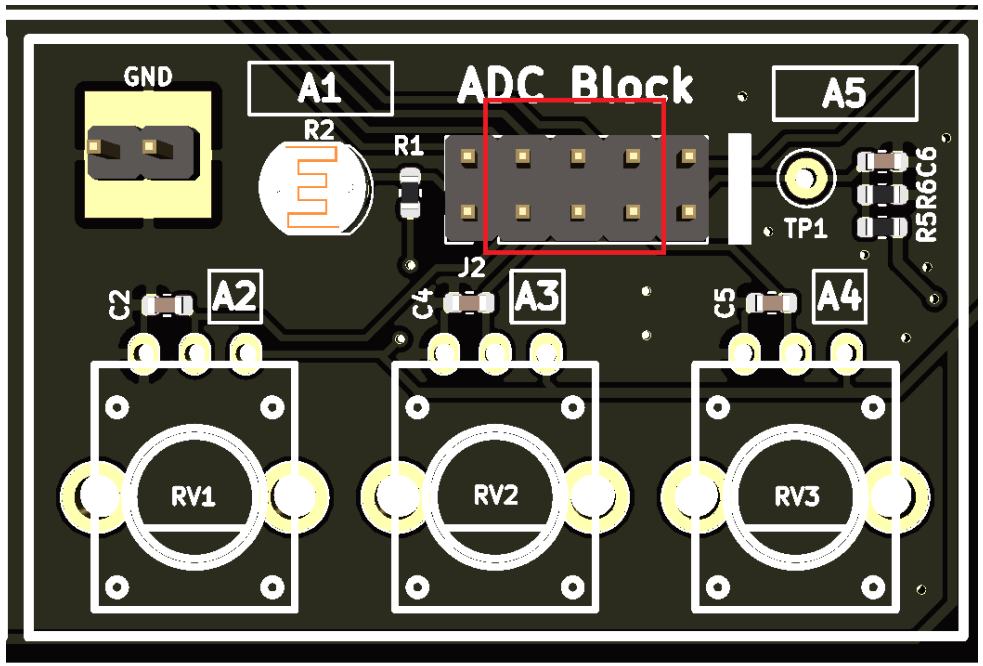
- **Resolution** : ADC는 항상 몇 Bit resolution인지가 명시 되어있다.
  만약 12bit ADC라면 0~VDDA 를 4096등분한다.

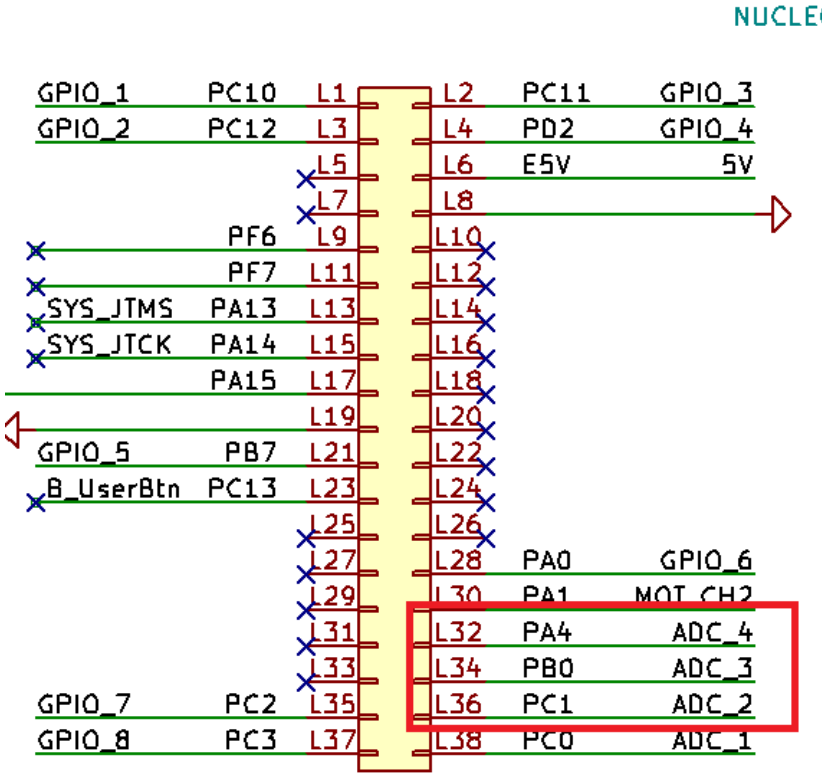
- **Sampling Rate** : 아날로그 전압을 디지털로 변환하는 작업은 상당히 긴 시간을 필요로 한다.
  Sampling Rate는 얼마나 빨리 전압 값을 변환할 수 있는지를 나타내 주는 지표이다.

4. ADC

• ADC 실습

- 보드의 ADC 블록을 아래 그림과 같이 점퍼선으로 연결해, Nucleo에 연결해 준다.
- 세 개의 가변저항은 **Nucleo의 PC1, PB0, PA4**에 연결되어 있다.
- 세 가변저항에서 측정된 전압을 **Polling, Interrupt, DMA**의 방식으로 받아본다.

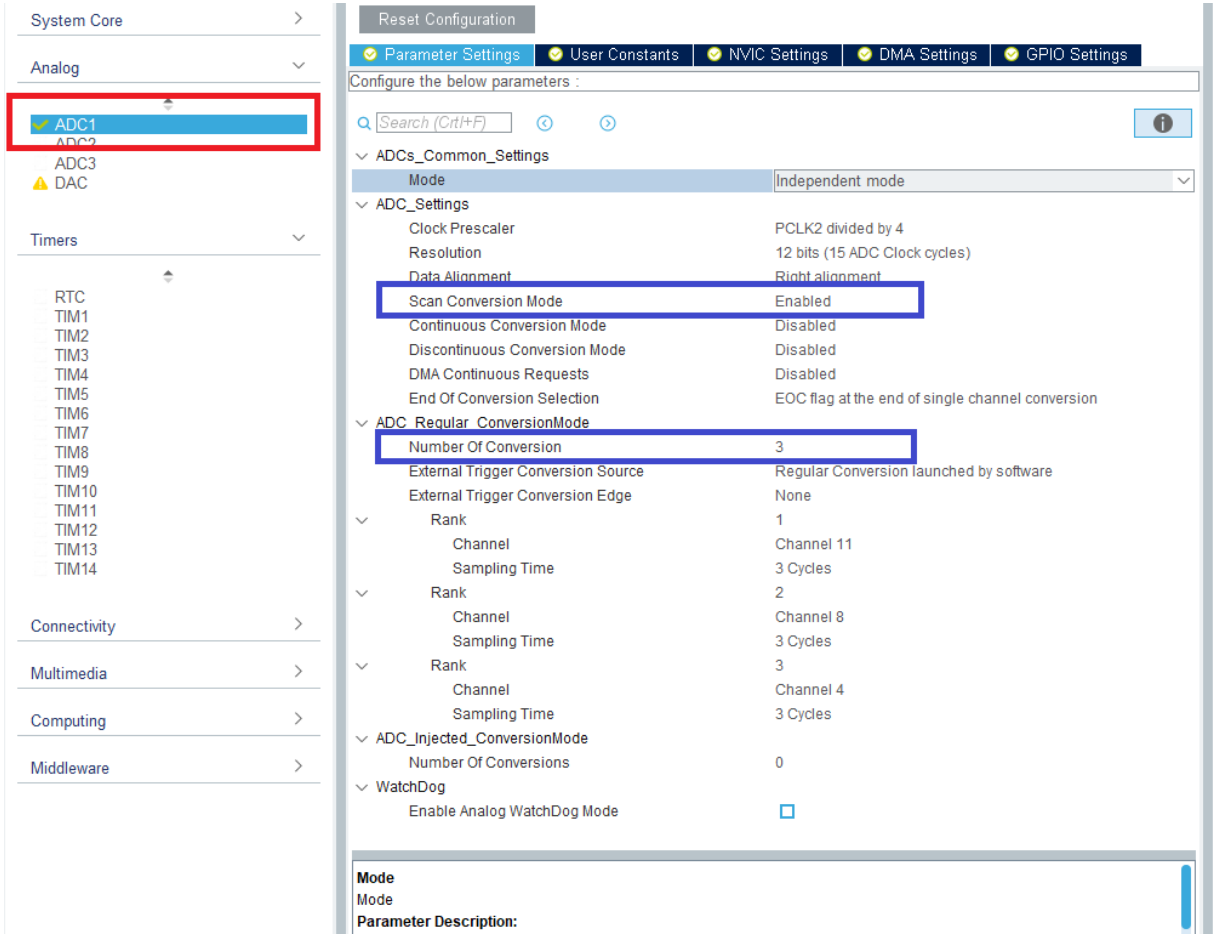


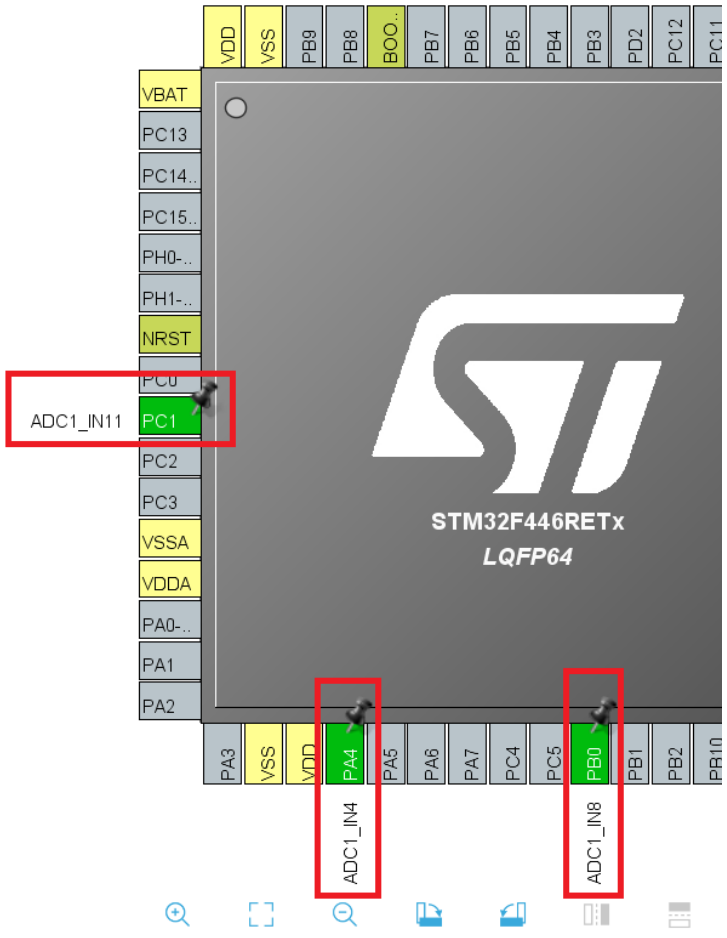


4. ADC

• ADC 실습 (Polling)

- 아래와 같이 세팅한 후, 코드를 생성한다.
- ADC 설정 부분은 아래를 잘 보고 주의해서 동일하게 설정한다.





4. ADC

• ADC 실습 (Polling)

- Main.c 상단에 ADC 값을 저장할 변수를 선언한다.

```
/* Private user code ---------------
/* USER CODE BEGIN 0 */
uint16_t adcValues[3];
/* USER CODE END 0 */
```

4. ADC

• ADC 실습 (Polling)

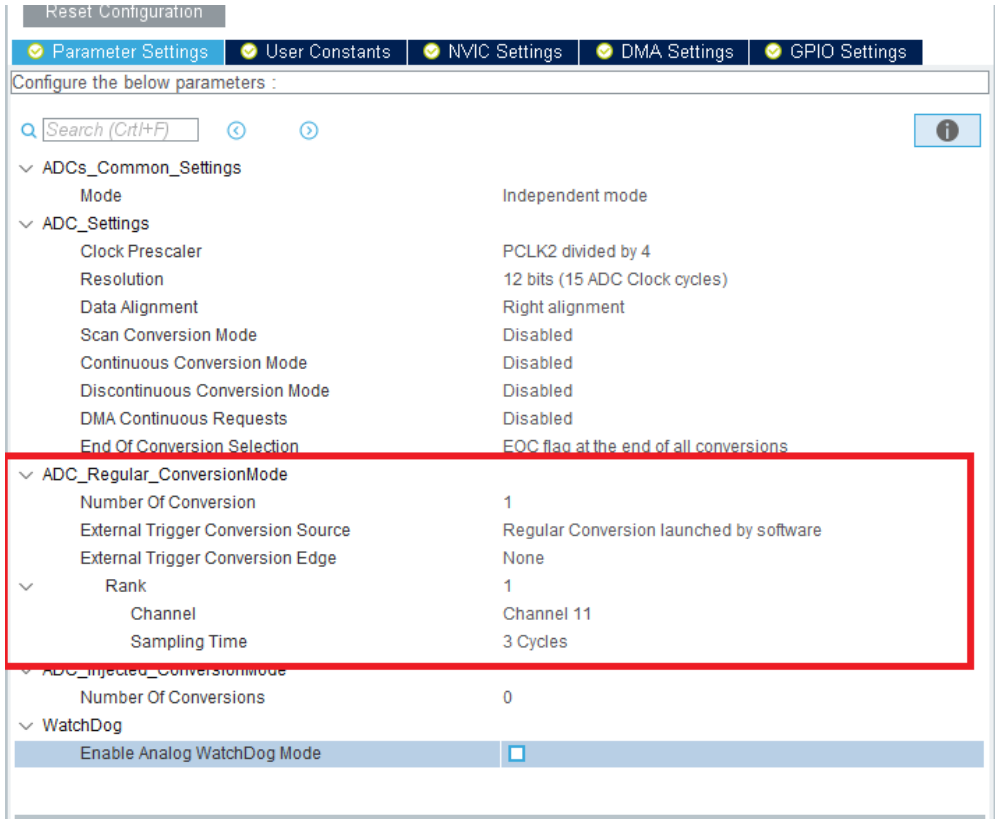
- **HAL_ADC_Start**(ADC Instance)
- **HAL_ADC_PollForConversion**(ADC Instance, Timeout)
- 아래와 같이 입력한 후, Live Watch를 통해 adcValues값의 변화를 보자.

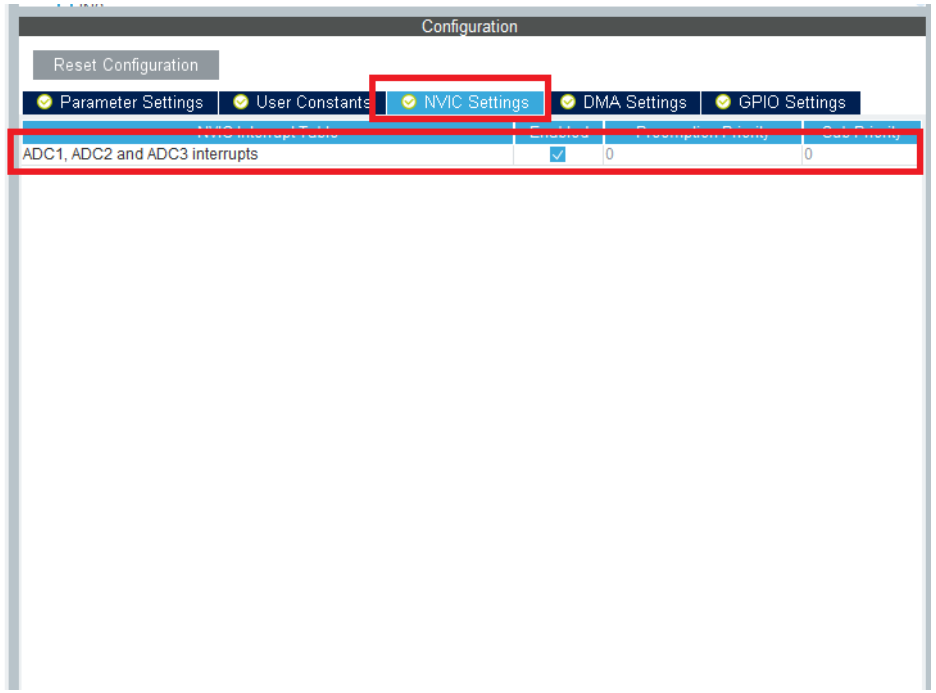
```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_ADC_Start(&hadc1);

  // Rank 1 Conversion
  HAL_ADC_PollForConversion(&hadc1, 1000);
  adcValues[0] = HAL_ADC_GetValue(&hadc1);

  // Rank 2 Conversion
  HAL_ADC_PollForConversion(&hadc1, 1000);
  adcValues[1] = HAL_ADC_GetValue(&hadc1);

  // Rank 3 Conversion
  HAL_ADC_PollForConversion(&hadc1, 1000);
  adcValues[2] = HAL_ADC_GetValue(&hadc1);

  HAL_Delay(10);

  /* USER CODE END WHILE */
```

4. ADC

• ADC 실습 (Interrupt)

- ADC Interrupt 방식은 구조상 여러 채널을 받는데 부적합하여 한 채널만 받기로 한다.
- ADC 설정에서 1채널만 Conversion하도록 설정하고, NVIC에서 인터럽트를 등록한다.
- 코드를 생성한다.





4. ADC

• ADC 실습 (Interrupt)

- 인터럽트 구문을 따라 들어가면, 최종적으로 다음과 같은 ***Weak 함수***로 빠지는 것을 알 수 있다.

- 해당 함수를 복사하여 main.c 적당한 곳에 함수를 만들자.

```
*/
void ADC_IRQHandler(void)
{
  /* USER CODE BEGIN ADC_IRQn 0 */

  /* USER CODE END ADC_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC_IRQn 1 */

  /* USER CODE END ADC_IRQn 1 */
}
```

```
    if (HAL_IS_BIT_CLR(hadc->State, HAL_ADC_STATE_INJ_BUSY))
    {
      SET_BIT(hadc->State, HAL_ADC_STATE_READY);
    }
  }

  /* Conversion complete callback */
  HAL_ADC_ConvCpltCallback(hadc);

  /* Clear regular group conversion flag */
  __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_STRT | ADC_FLAG_EOC);
}
```

```
  * @retval None
  */
__weak void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(hadc);
  /* NOTE : This function Should not be modified, when the callback is needed
            the HAL_ADC_ConvCpltCallback could be implemented in the user file
  */
}
```

4. ADC

• ADC 실습 (Interrupt)

- Main.c 적당한 곳에 다음과 같이 작성하자.

```
/* Private user code ---------------------------------------
/* USER CODE BEGIN 0 */
uint16_t adcValue;
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
  if(hadc == &hadc1){
    adcValue = HAL_ADC_GetValue(&hadc1);
  }
}
/* USER CODE END 0 */
```

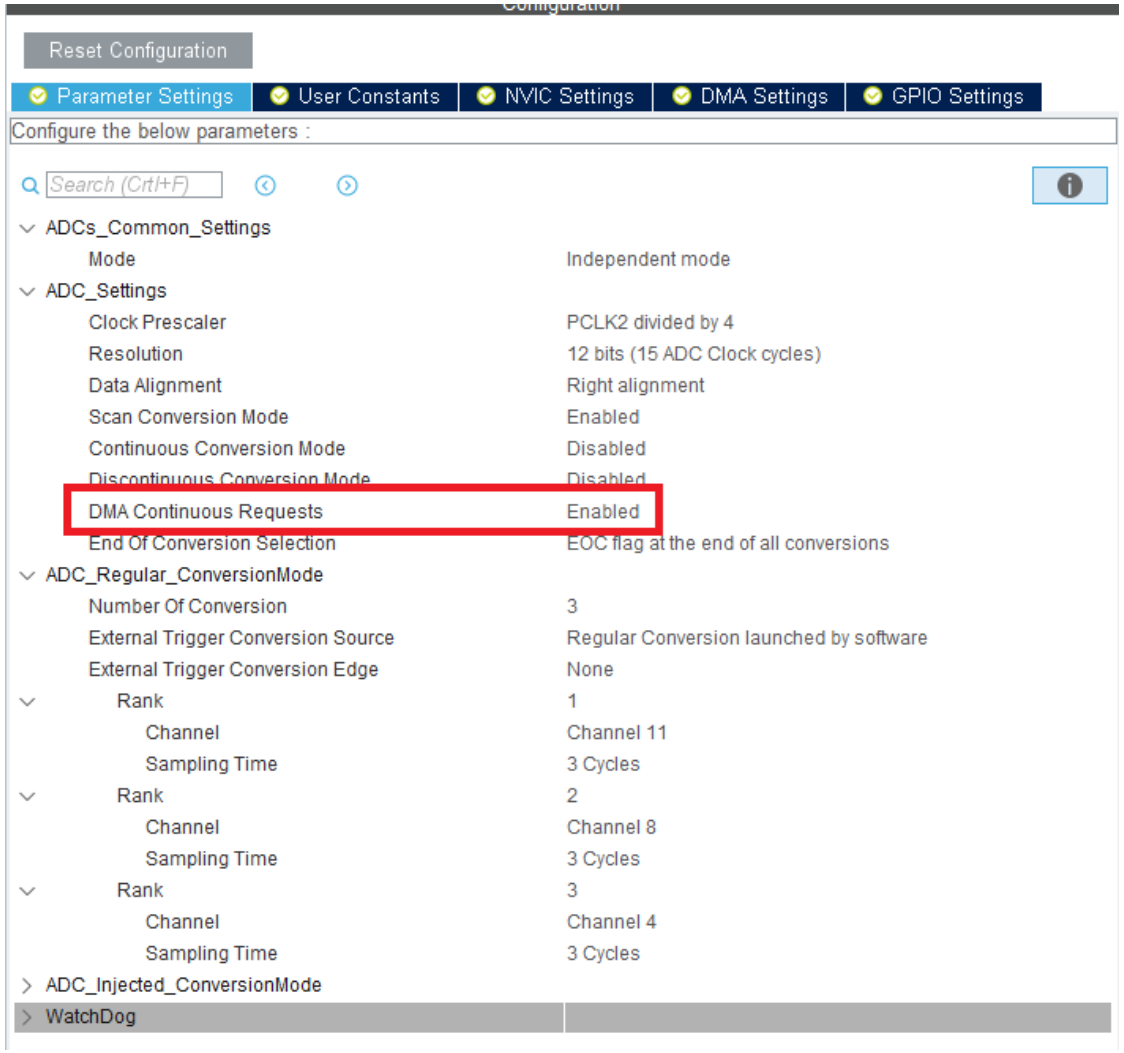
- While문 안에 다음과 같이 작성하여 주기적으로 인터럽트 변환을 실행하자.
- 펌웨어를 업로드 하고, Live Watch를 통해서 변수 값을 관찰하자.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_ADC_Start_IT(&hadc1);
  HAL_Delay(10);
  /* USER CODE END WHILE */
```

4. ADC

• ADC 실습 (DMA)

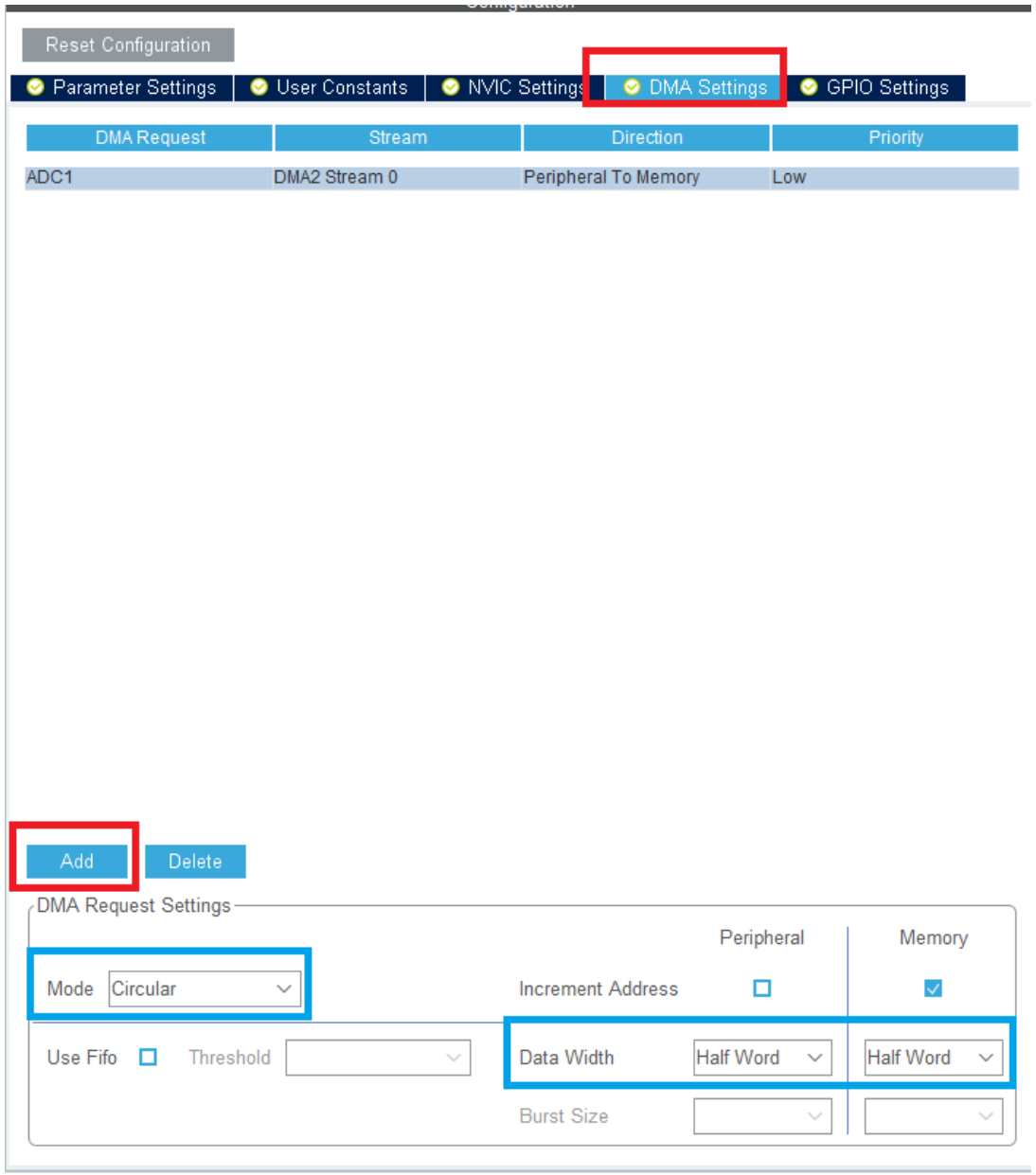
- ADC 세팅을 다음과 같이 바꾼다.
- 이때 DMA Continuous Requests를 Enabled 해준다.



4. ADC

• ADC 실습 (DMA)

- DMA 세팅을 다음과 같이 설정한 후 코드를 생성한다.



4. ADC

• ADC 실습 (DMA)

- Main.c 상단에 ADC 값을 저장할 변수를 선언한다.

```
/* Private user code ---------------
/* USER CODE BEGIN 0 */
uint16_t adcValues[3];
/* USER CODE END 0 */
```

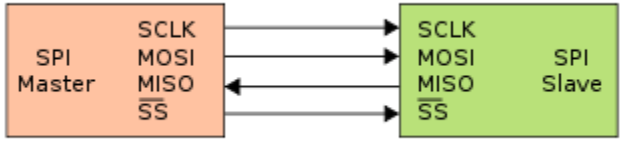
- 다음과 같이 DMA를 시작하여, DMA가 지속적으로 ADC에게 요청을 하도록 하고, 일정 주기로 ADC를 시작시켜 준다.
- 해당 코드를 업로드 하고 변수를 관찰해 보자.

```
MX_DMA_Init();
MX_ADC1_Init();
/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adcValues, 3);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_ADC_Start(&hadc1);
  HAL_Delay(10);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
```

5. SPI

5. SPI

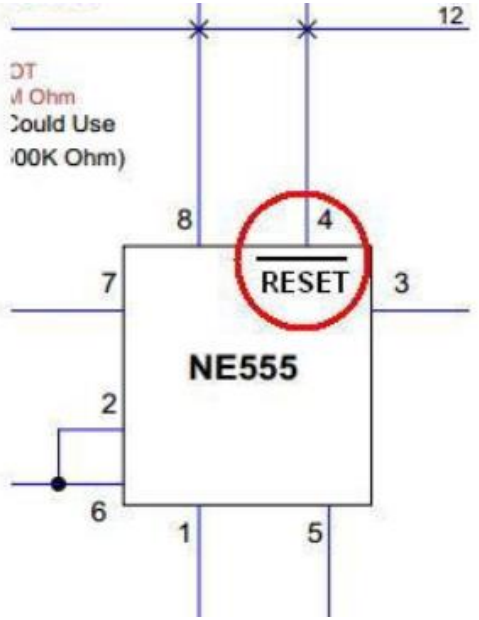
- SPI(Serial Peripheral Interface)



- SCK, MOSI, MISO, NSS의 4선으로 구성되며, 통신속도가 매우 빨라 고속 통신에 많이 사용된다.
- 각 통신선의 역할은 다음과 같다.
  - **SCK** : 통신 클럭이다. 모든 통신은 **이 클럭에 맞춰서** 이루어지며, 해당 클럭을 **생성하는 측을 Master**, 해당 클럭을 **받는 측을 Slave**라 한다.
  - **MOSI** : Master Out Slave In / 마스터에서 슬레이브로 전송되는 라인
  - **MISO** : Master In Slave Out / 슬레이브에서 마스터로 전송되는 라인
  - **NSS** : Chip Select***(Active Low)*** / 통신할 칩을 선택하는 라인

Q. Active Low란?

A. **Low가 되었을 때** 기능이 활성화 됨을 의미.
일반적으로 접두어 ***N,~,/*** 를 붙이거나, 위쪽에 바를 붙여서 표시한다.



Q. SPI는 Synchronous 통신인가 Asynchronous 통신인가?

A. Synchronous

5. SPI

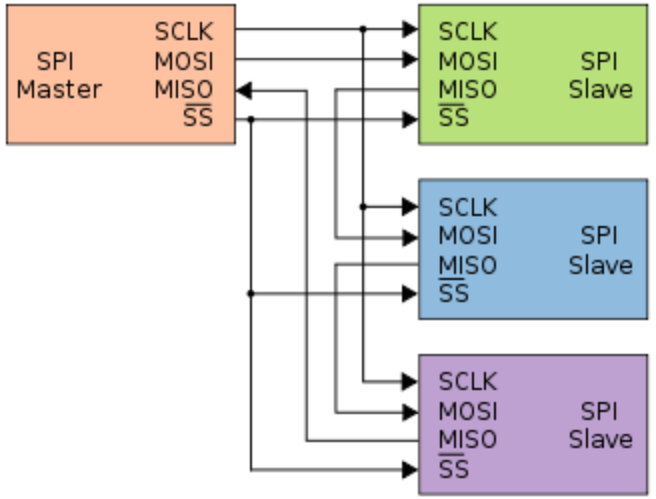
- SPI(Serial Peripheral Interface)

- SPI는 NSS 핀을 통하여 특정 Slave를 선택할 수 있다.
- 따라서 SPI는 **1:N 통신**이 가능하다.
- 하지만 SPI는 연결 선이 많아 1:N으로 연결하여 사용하는 경우는 많이 없다.

Q. Full Duplex / Half Duplex란?

A. **Full Duplex**란 데이터를 **보내면서 동시에 받을 수 있는 시스템**을 말한다. 반대로 **Half Duplex**는 **보낼 때는 받지 못하고, 받을 때는 보내지 못하는 시스템**을 말한다.

SPI는 Full Duplex / Half Duplex 모두 가능하다.



5. SPI

• SPI Frame

- 뭔가 미묘하게 다르다...?

Sensor A

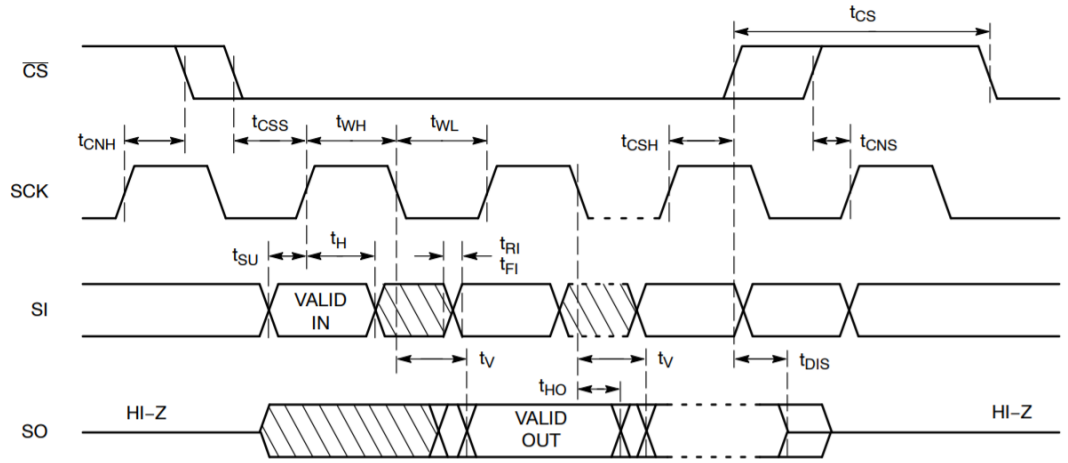
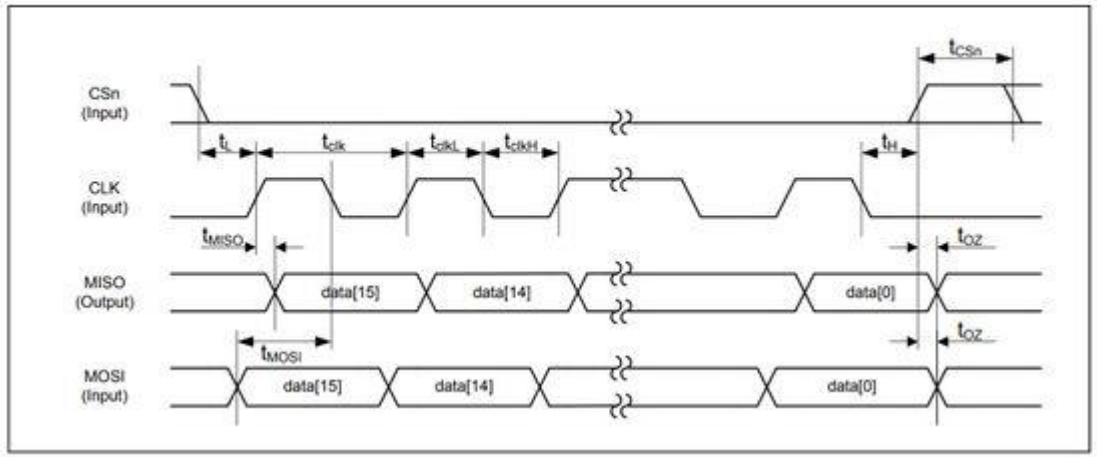


Figure 2. Synchronous Data Timing

Sensor B

Figure 14:
SPI Timing Diagram



5. SPI

• SPI Frame

- 뭔가 미묘하게 다르다...?

Sensor A

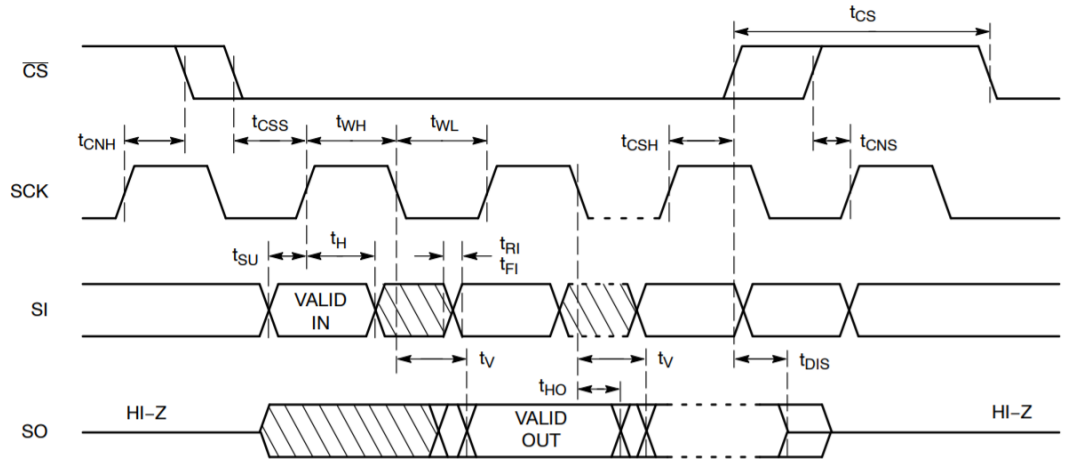
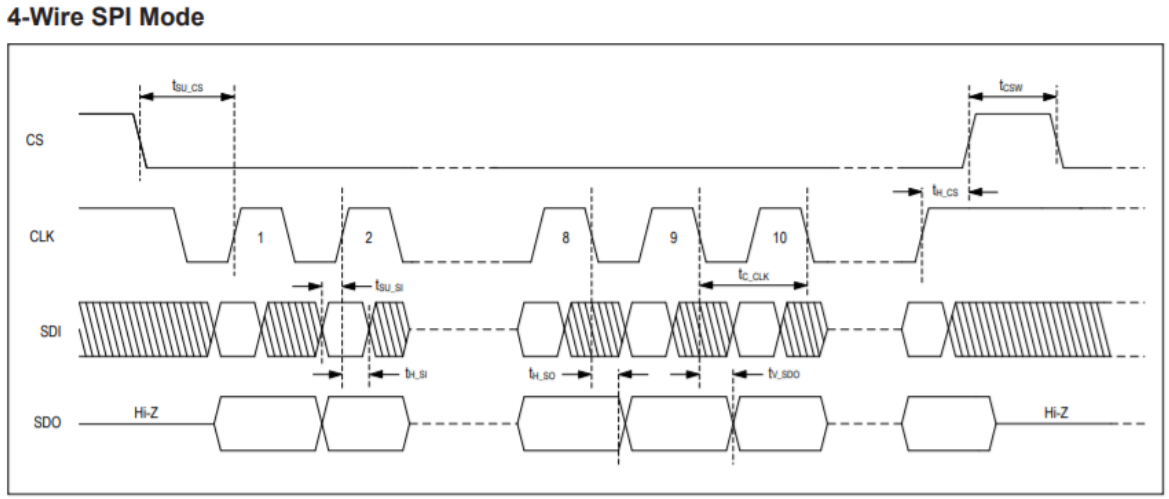


Figure 2. Synchronous Data Timing

Sensor C



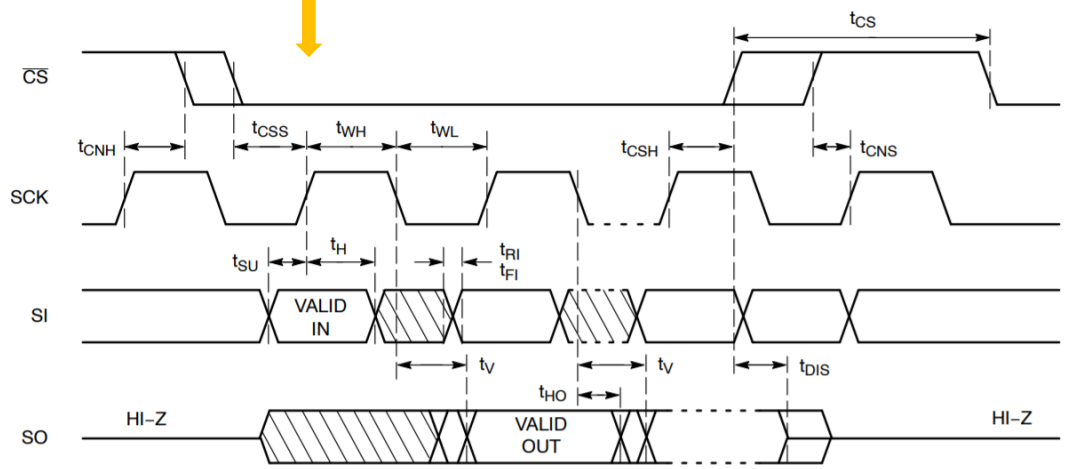
5. SPI

- **SPI Frame**

CPHA = 1 Edge

- 뭔가 미묘하게 다르다...?

Sensor A



CPOL = Low

Figure 2. Synchronous Data Timing

- **CPOL (Clock Polarity) :** 클럭의 IDLE 상태의 Polarity
- **CPHA (Clock Phase) :** 데이터를 읽는 타이밍 (첫번째 edge 혹은 두번째 edge)

5. SPI

• SPI Frame

- 뭔가 미묘하게 다르다...?

Sensor A

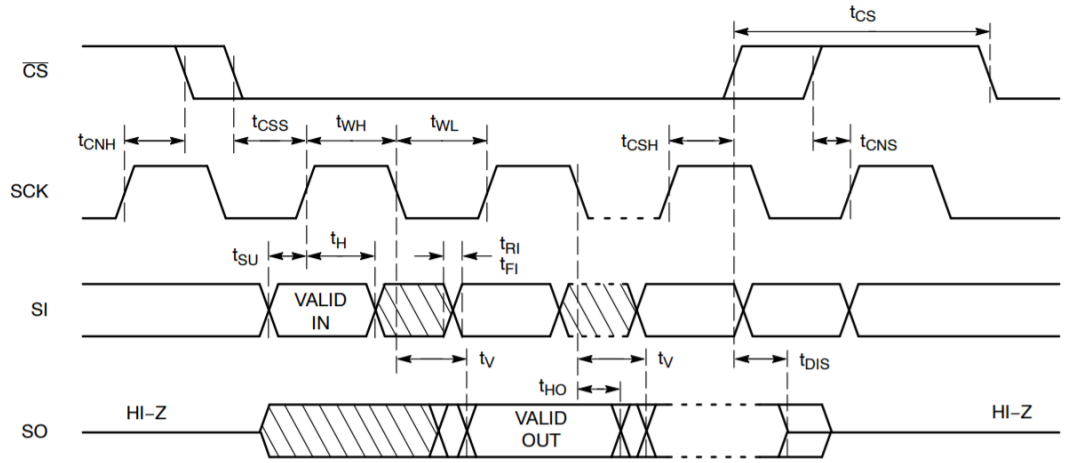
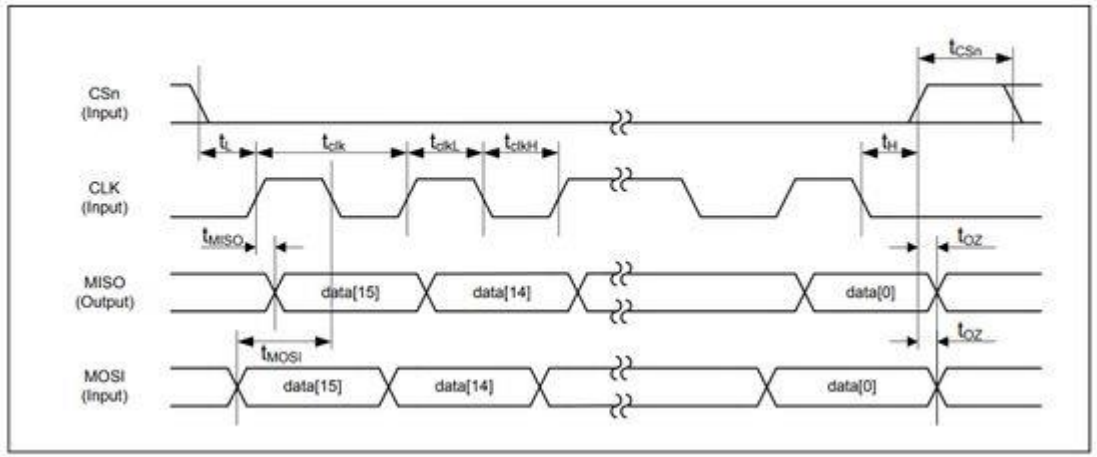


Figure 2. Synchronous Data Timing

CPOL = Low
CPHA = 1 Edge

Sensor B

Figure 14:
SPI Timing Diagram



CPOL = Low
CPHA = 2 Edge

5. SPI

• SPI Frame

- 뭔가 미묘하게 다르다...?

Sensor A

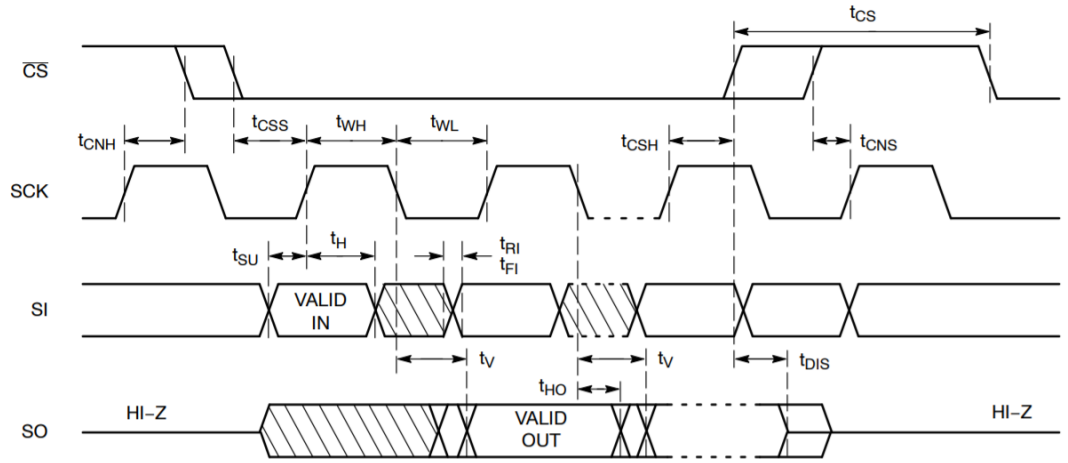
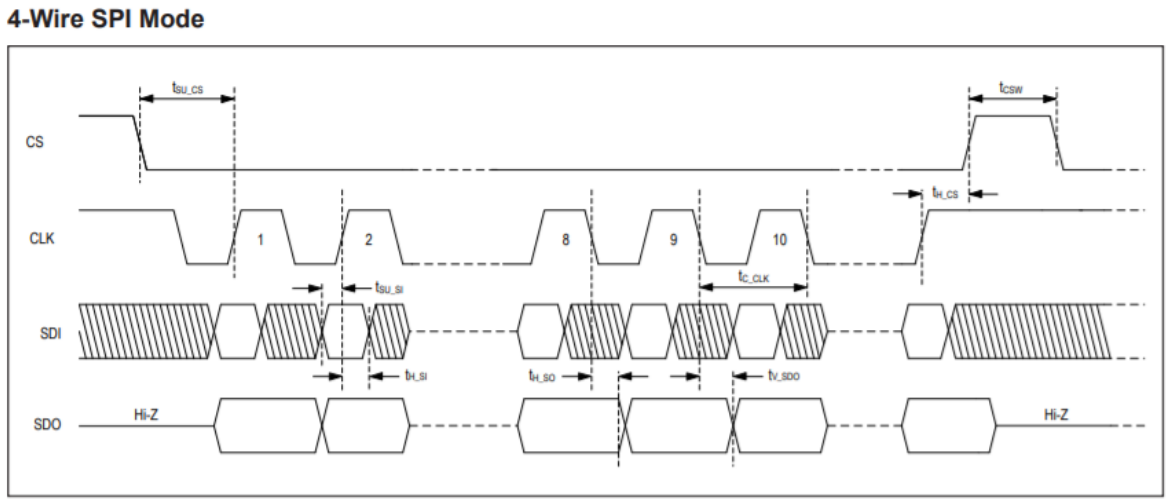


Figure 2. Synchronous Data Timing

CPOL = Low
CPHA = 1 Edge

Sensor C



CPOL = High
CPHA = 2 Edge