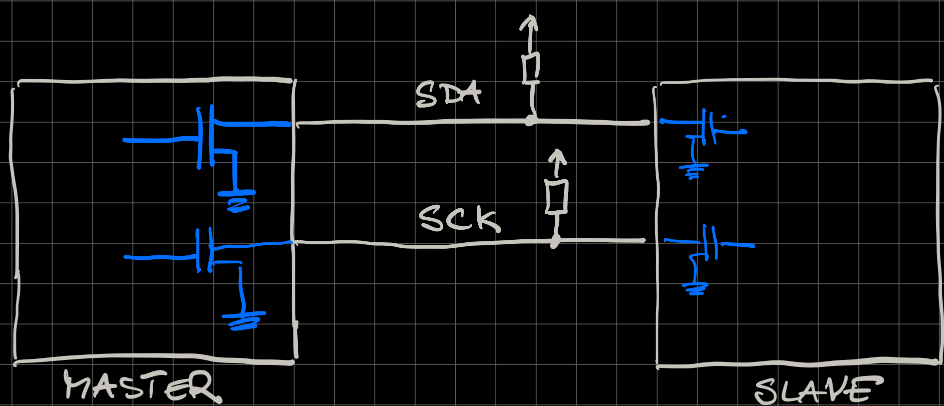
SINHRONSKA SERIJSKA KOMUNIKACIJA

* I2C

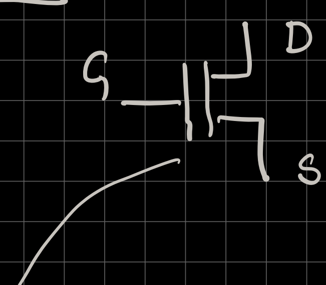
- 2 signala za komunikacijo:
  - SDA : Serial Data
  - SCK : Serial Clock

Fizična plast:

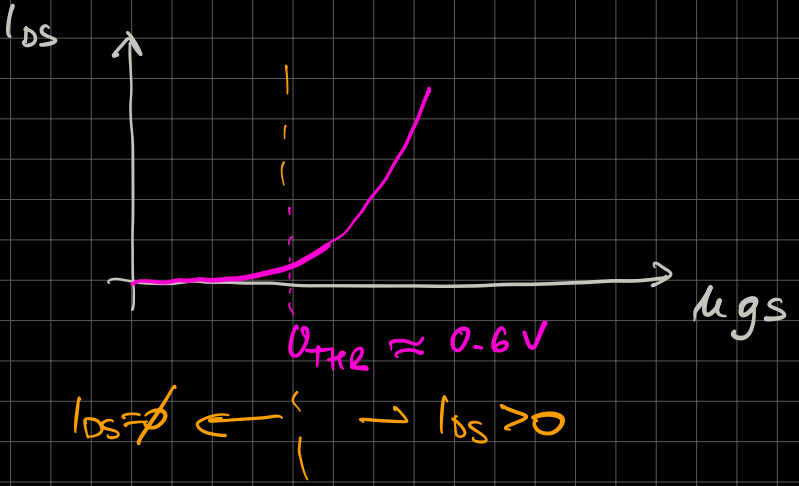


Vem, da je politično nekorektno, ampak v tem trenutku ni boljših izrazov

Open - Drain



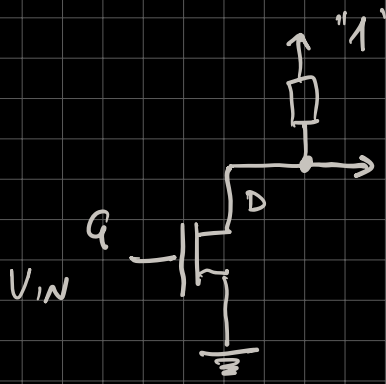
delovanje: (zelo poenostavljeno)



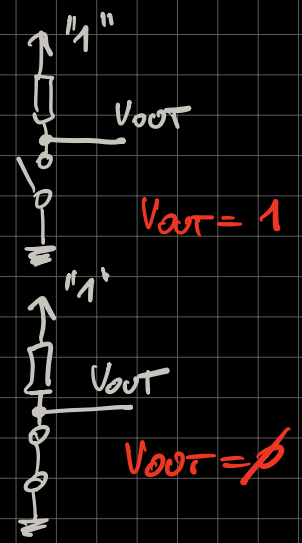
Nas zanima način delovanja tega stikala:

U_G = "1" ⟶

U_G = "0" ⟶



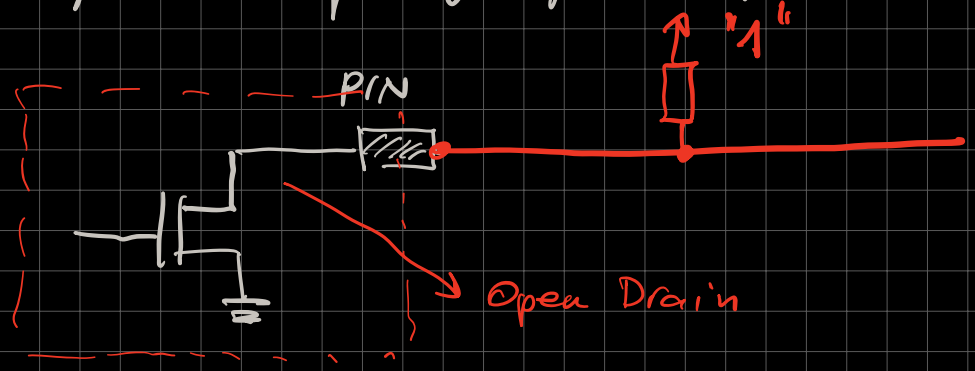
⇒ $V_{IN} = \begin{cases} 0 : \\ 1 : \end{cases}$



- Imamo na izhodu stabilno logično stanje = "1"

- Obtičemo pričemu samo "0" / na izhodu sicer je prisotna "1"

Ta princip se uporablja pri serijski komunikaciji:



Protokol

→ prenos se vedno začne z enim START bitom

→ prenos se zaključi s STOP bitom

→ prenos ima 2 fazi:
NASLOVNA in PODATKOVNA

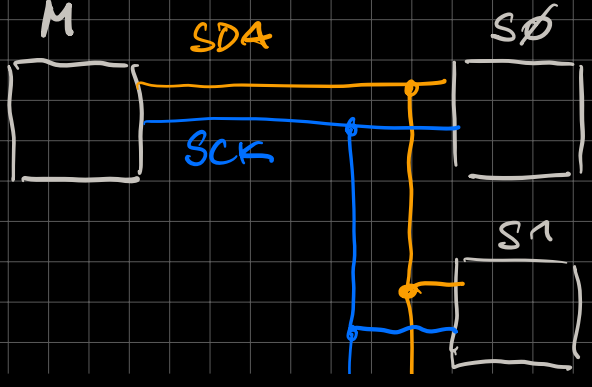
⇓ ⇓

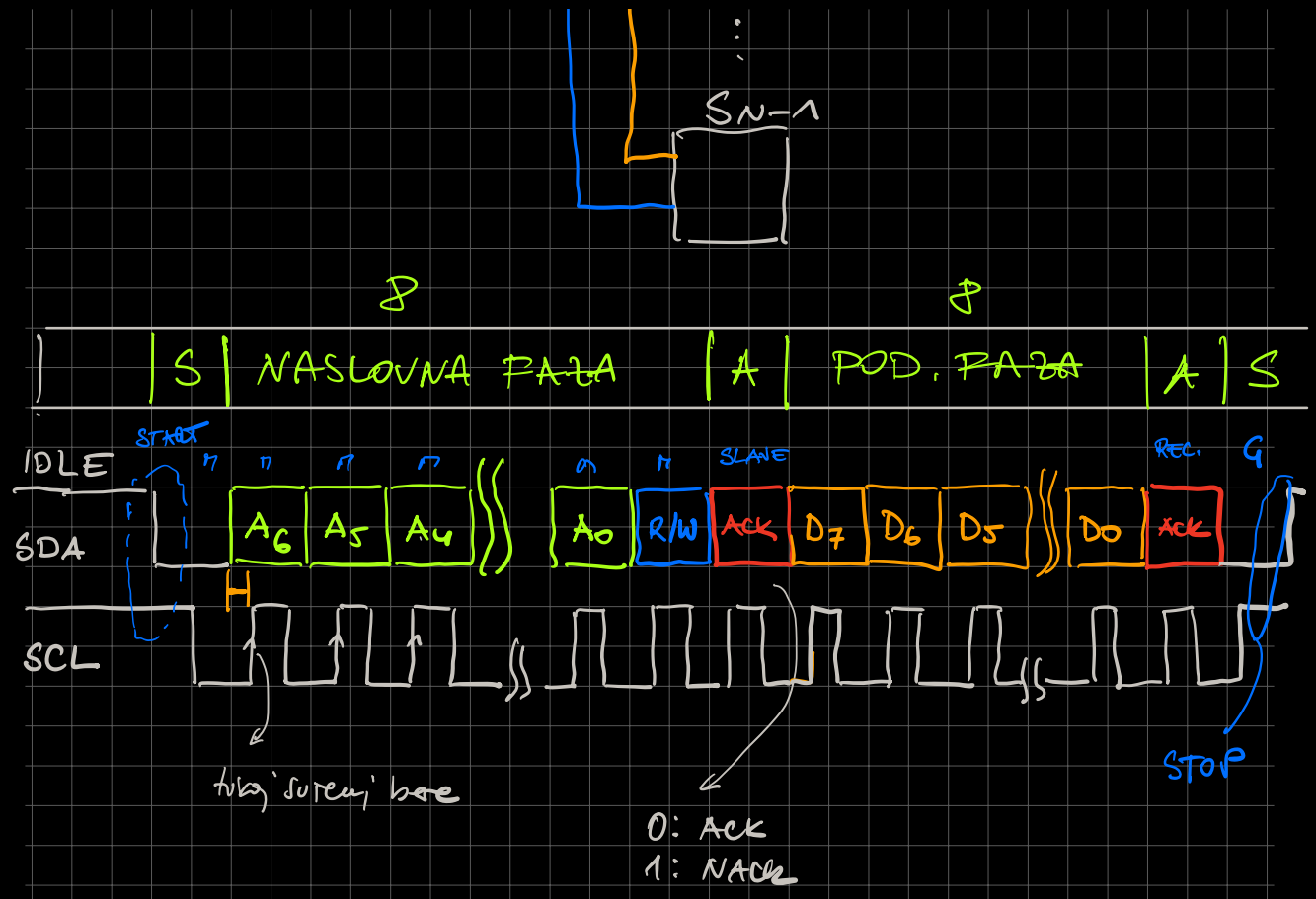
pošljemo 8 bitni NASLOV senzorja

8 bitov podatkov

→ po 8 bitih prenosa obvezno sledi t.i. ACK bit

"One MASTER, many SLAVES"





Možen je tudi eksplozijski (BURST) prenos podatkov:

S — NASLOVNA FAZA — A — PODATEK — A — PODATEK — A — ... — STOP

HAL:

```
typedef struct {
  I2C_TypeDef                 *Instance;  /* I²C registers base address      */
  I2C_InitTypeDef             Init;       /* I²C communication parameters    */
  uint8_t                     *pBuffPtr;  /* Pointer to I²C transfer buffer */
  uint16_t                    XferSize;   /* I²C transfer size               */
  __IO uint16_t               XferCount;  /* I²C transfer counter            */
  DMA_HandleTypeDef           *hdmatx;    /* I²C Tx DMA handle parameters    */
  DMA_HandleTypeDef           *hdmarx;    /* I²C Rx DMA handle parameters    */
  HAL_LockTypeDef             Lock;       /* I²C locking object              */
  __IO HAL_I2C_StateTypeDef   State;      /* I²C communication state         */
  __IO HAL_I2C_ModeTypeDef    Mode;       /* I²C communication mode          */
  __IO uint32_t               ErrorCode;  /* I²C Error code                  */
} I2C_HandleTypeDef;
```

```c
typedef struct
{
  uint32_t Timing;              /*!< Specifies the I2C_TIMINGR_register value.
                                     This parameter calculated by referring to I2C initialization section
                                     in Reference manual */

  uint32_t OwnAddress1;         /*!< Specifies the first device own address.
                                     This parameter can be a 7-bit or 10-bit address. */

  uint32_t AddressingMode;      /*!< Specifies if 7-bit or 10-bit addressing mode is selected.
                                     This parameter can be a value of @ref I2C_ADDRESSING_MODE */

  uint32_t DualAddressMode;     *!< Specifies if dual addressing mode is selected.
                                     This parameter can be a value of @ref I2C_DUAL_ADDRESSING_MODE */

  uint32_t OwnAddress2;         /*!< Specifies the second device own address if dual addressing mode is selected
                                     This parameter can be a 7-bit address. */

  uint32_t OwnAddress2Masks;    /*!< Specifies the acknowledge mask address second device own address if dual addressing
                                     mode is selected.
                                     This parameter can be a value of @ref I2C_OWN_ADDRESS2_MASKS */

  uint32_t GeneralCallMode;     /*!< Specifies if general call mode is selected.
                                     This parameter can be a value of @ref I2C_GENERAL_CALL_ADDRESSING_MODE */

  uint32_t NoStretchMode;       /*!< Specifies if nostretch mode is selected.
                                     This parameter can be a value of @ref I2C_NOSTRETCH_MODE */

} I2C_InitTypeDef;
```

Ta je SLAVE

2.2.4 I2C timing register

The I2C timing register is defined as the following table shows:

Table 3. Timing register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]			
rw								rw				rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]								SCLL[7:0]							
rw								rw							

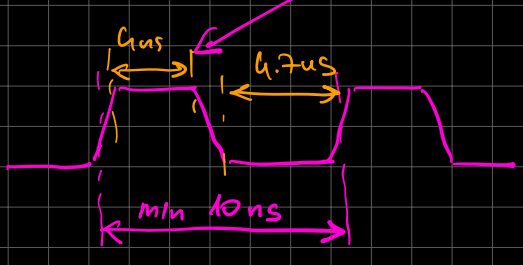
PRESC[3:0] is used to prescale I2C clock source (I2CCLK); it allows the generation of a divided clock. The period of this divided clock t_{PRESC} is defined by:

$t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$

I2S standard predvideva 3 načine?

Table 2. I2C timings specification (see I2C specification, rev.03, June 2007)

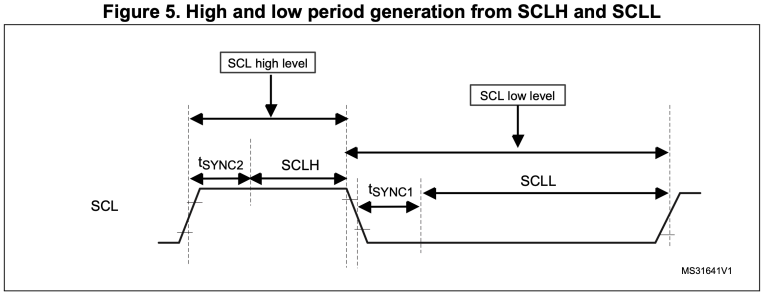
Symbol	Parameter	Standard		Fast mode		Fast mode +		Unit
		Min	Max	Min	Max	Min	Max	
f_{SCL}	SCL clock frequency	0	100	0	400	0	1000	KHz
t_{LOW}	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	µs
t_{HIGH}	High Period of the SCL clock	4		0.6		0.26	-	µs



4 + 6 us
H L
5 , 5 us

SCLH[7:0] and SCLL[7:0] are used to configure I2C speed frequency when master mode is selected. SCLH generates the high period of the SCL clock (t_{HIGH}) and SCLL generates the low period of the SCL clock (t_{LOW}). The figure below shows how these timings are deduced:

Figure 5. High and low period generation from SCLH and SCLL



SCLDEL[3:0] is used to program the data setup time ($t_{SU;DAT}$) as shown in the following figure:

Figure 3. Data setup time generation from SCLDEL

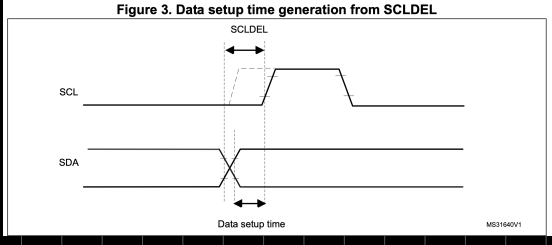
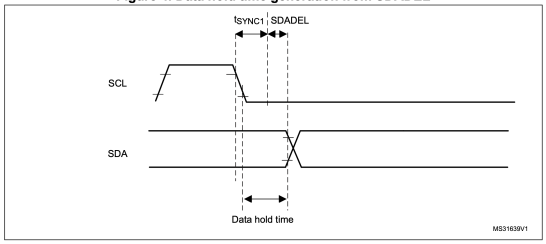


Figure 4. Data hold time generation from SDADEL



duration depends on these parameters:

Komunikacija v polling načinu:

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
                                          uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

```
HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
                                         uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

Prekinitveni način:

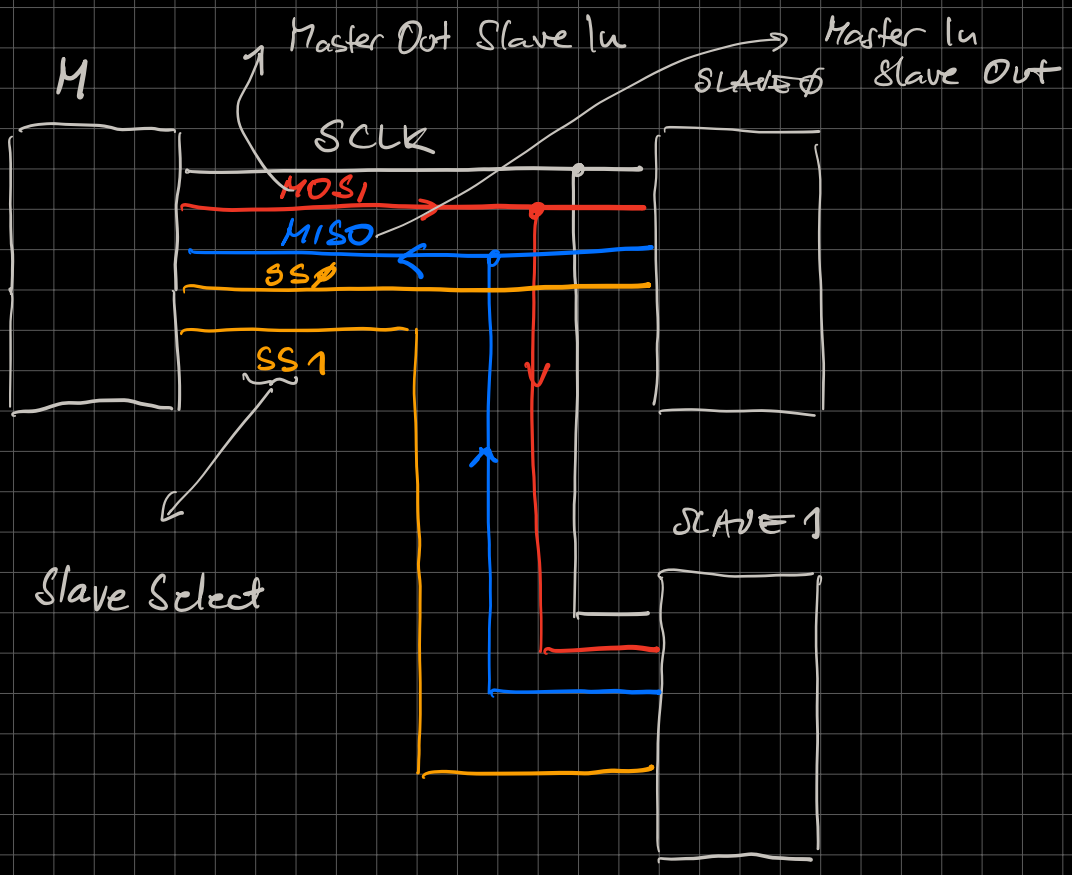
```
HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
                                             uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress,
                                            uint8_t *pData, uint16_t Size);
```

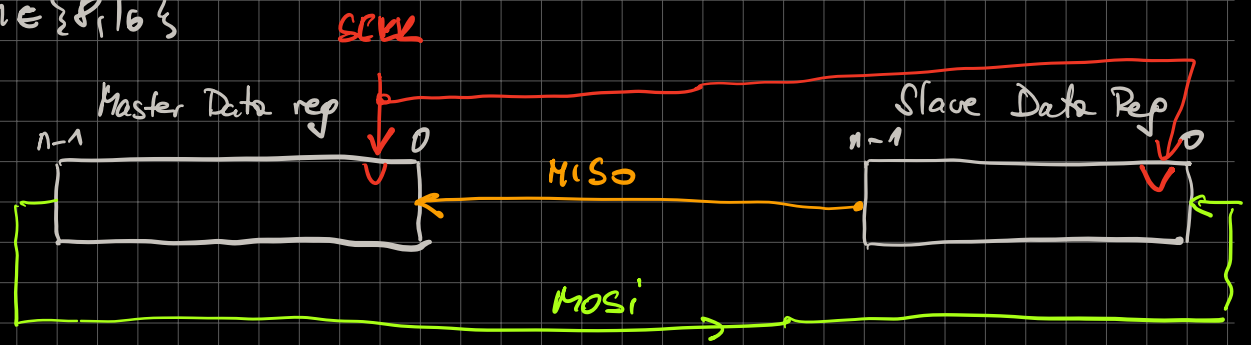
SPI

→ omogoča FULL-DUPLEX komunikacijo

→ SPI uporablja 4 žice



$n \in \{8, 16\}$



Vsako urino periodo gre MSB bit iz Slave v LSB bit postopoma ali MSB bit iz Master v LSB bit Slave-a

FULL Duplex

→ ni start/stop bita
→ ni naslova fona

→ Vsako urino periodo se en bit zapiše v register ob prehodu ure iz neaktivnega stanja v aktivno stanje

↓

SPI omogoča izbiro aktivnega stanja ure

↓

CPOL (Clock Polarity)

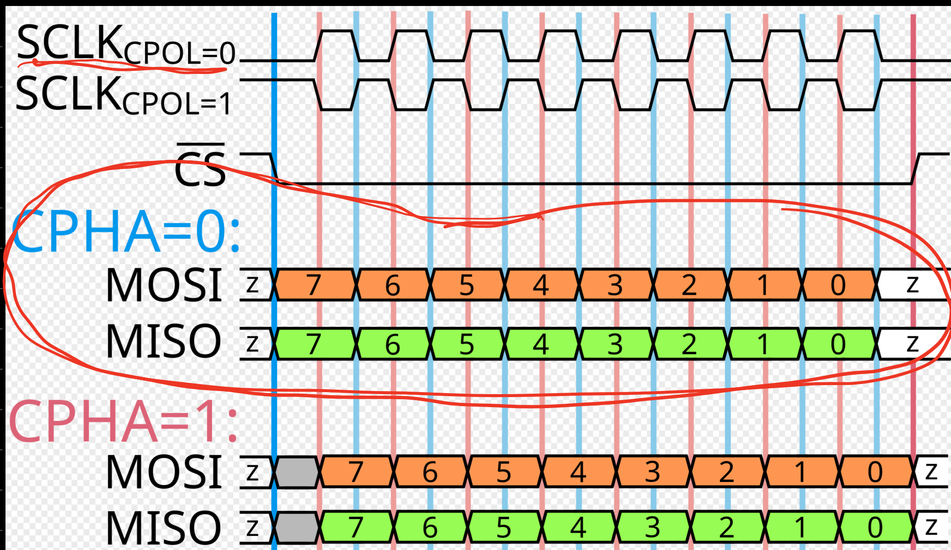
"L" → aktivno je 1, neaktivno je 0

"H" → aktivno "0", neaktivno "1"

SPI omogoča nastavitev, kdaj odčitki pridejo na vodilo (pred prvo urino fronto ali ne)

↓

CPHA → Clock Phase



SPI definito 4 modalità differenti

CPOL	CPHA	MODE
0	0	0
0	1	1
1	0	2
1	1	3

```
typedef struct __SPI_HandleTypeDef {
  SPI_TypeDef               *Instance;    /* SPI registers base address */
  SPI_InitTypeDef           Init;         /* SPI communication parameters */
  uint8_t                   *pTxBuffPtr;  /* Pointer to SPI Tx transfer Buffer */
  uint16_t                  TxXferSize;   /* SPI Tx Transfer size */
  __IO uint16_t             TxXferCount;  /* SPI Tx Transfer Counter */
  uint8_t                   *pRxBuffPtr;  /* Pointer to SPI Rx transfer Buffer */
  uint16_t                  RxXferSize;   /* SPI Rx Transfer size */
  __IO uint16_t             RxXferCount;  /* SPI Rx Transfer Counter */
  DMA_HandleTypeDef         *hdmatx;      /* SPI Tx DMA Handle parameters   */
  DMA_HandleTypeDef         *hdmarx;      /* SPI Rx DMA Handle parameters   */
  HAL_LockTypeDef           Lock;         /* Locking object                 */
  __IO HAL_SPI_StateTypeDef State;        /* SPI communication state */
  __IO uint32_t             ErrorCode;    /* SPI Error code */
} SPI_HandleTypeDef;
```

```
typedef struct {
  uint32_t Mode;                /* Specifies the SPI operating mode. */
  uint32_t Direction;           /* Specifies the SPI bidirectional mode state. */
  uint32_t DataSize;            /* Specifies the SPI data size. */
  uint32_t CLKPolarity;         /* Specifies the serial clock steady state. */
  uint32_t CLKPhase;            /* Specifies the clock active edge for the bit capture. */
  uint32_t NSS;                 /* Specifies whether the NSS signal is managed by
                                   hardware (NSS pin) or by software */
  uint32_t BaudRatePrescaler;   /* Specifies the Baud Rate prescaler value which will be
                                   used to configure the SCK clock. */
  uint32_t FirstBit;            /* Specifies whether data transfers start
                                   from MSB or LSB bit. */
  uint32_t TIMode;              /* Specifies if the TI mode is enabled or not. */
  uint32_t CRCCalculation;      /* Specifies if the CRC calculation is enabled or not. */
  uint32_t CRCPolynomial;       /* Specifies the polynomial used for the CRC calculation. */
} SPI_InitTypeDef;
```

- Mode: MASTER / SLAVE
- Direction: FULL / HALF
- DataSize: 8/16
- NSS: SW / HW
- BaudRatePrescaler: 2^0 – 2^7
- FirstBit: MSB / LSB

```c
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size,
                                   uint32_t Timeout);
```

```c
HAL_StatusTypeDef HAL_SPI_Receive(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size,
                                  uint32_t Timeout);
```

HALF

```c
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
                                          uint8_t *pRxData, uint16_t Size,
                                          uint32_t Timeout);
```

→ Full Duplex

```c
HAL_StatusTypeDef HAL_SPI_Transmit_IT(SPI_HandleTypeDef *hspi, uint8_t *pData,
                                      uint16_t Size);
HAL_StatusTypeDef HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData,
                                     uint16_t Size);
HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t *pTxData,
                                             uint8_t *pRxData, uint16_t Size);
```