Osnove mikroprocesorske elektronike

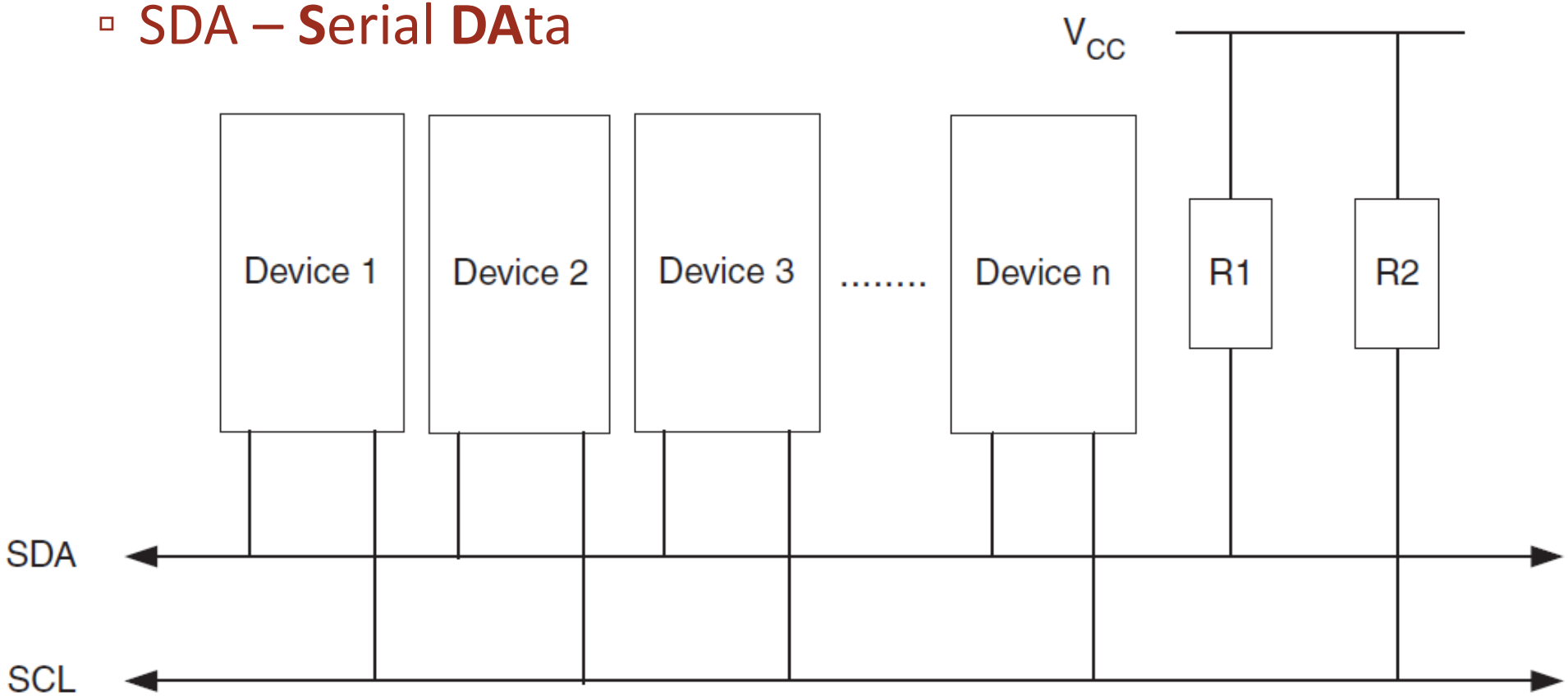
Marko Jankovec

Sinhrona komunikacijska vodila

Primeri I^2C

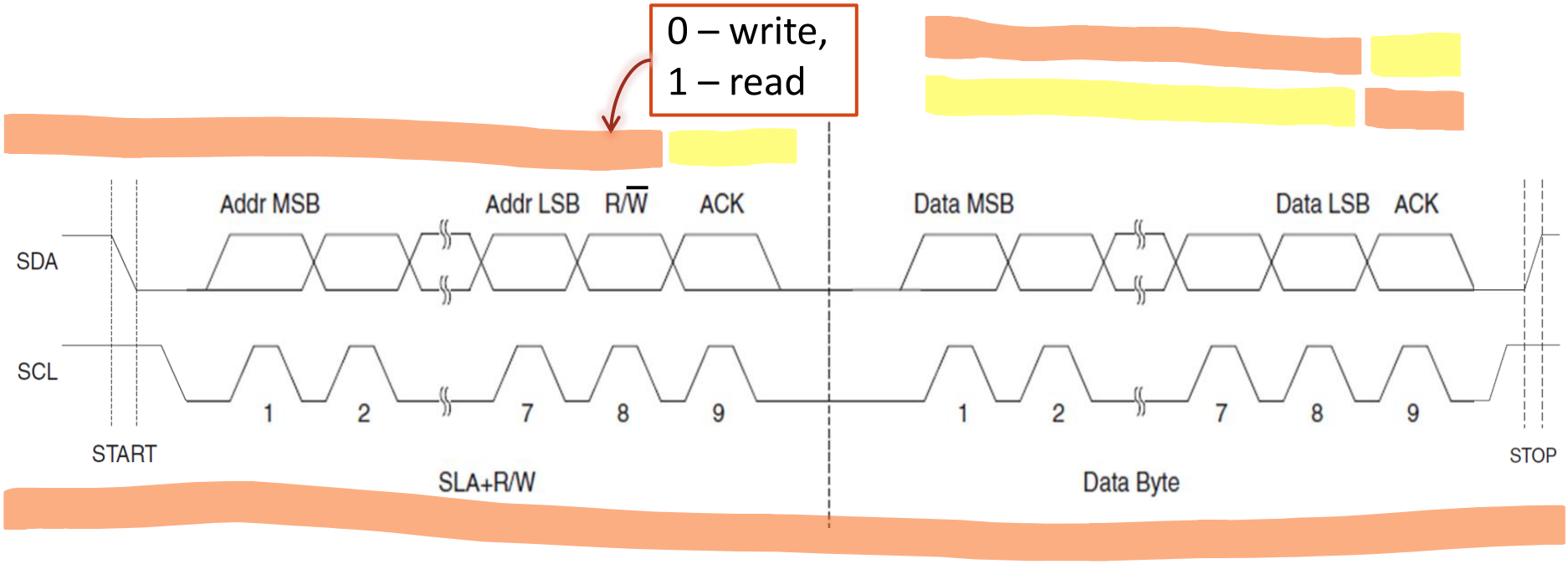
Vodilo IIC (Inter Integrated Circuit) – I²C

- Sinhrono vodilo
  - SCL – **S**erial **CL**ock
  - SDA – **S**erial **DA**ta

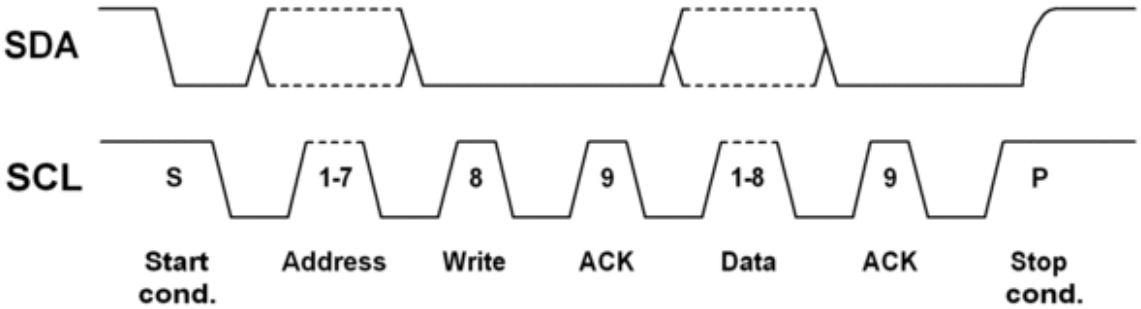


Tipičen potek komunikacije I2C

Gospodar (Master)
Suženj (Slave)



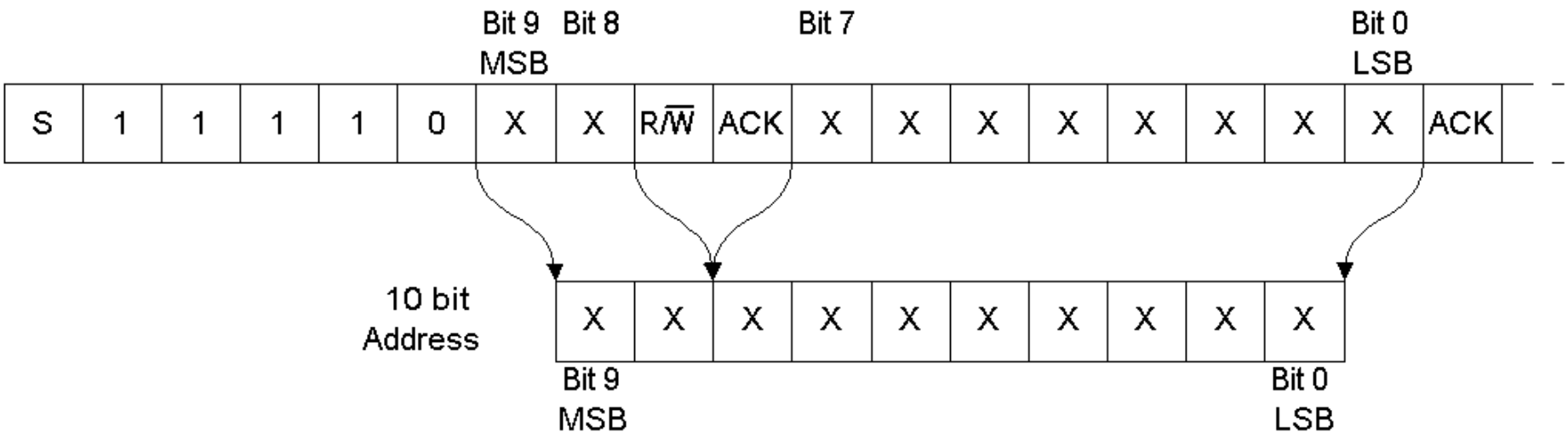
Naslovni prostor – 7 bitov



0000000 0	General Call
0000000 1	Start Byte
0000001 X	CBUS Addresses
0000010 X	Reserved for Different Bus Formats
0000011 X	Reserved for future purposes
00001XX X	High-Speed Master Code
11110XX X	10-bit Slave Addressing
11111XX X	Reserved for future purposes

https://www.i2c-bus.org/addressing/

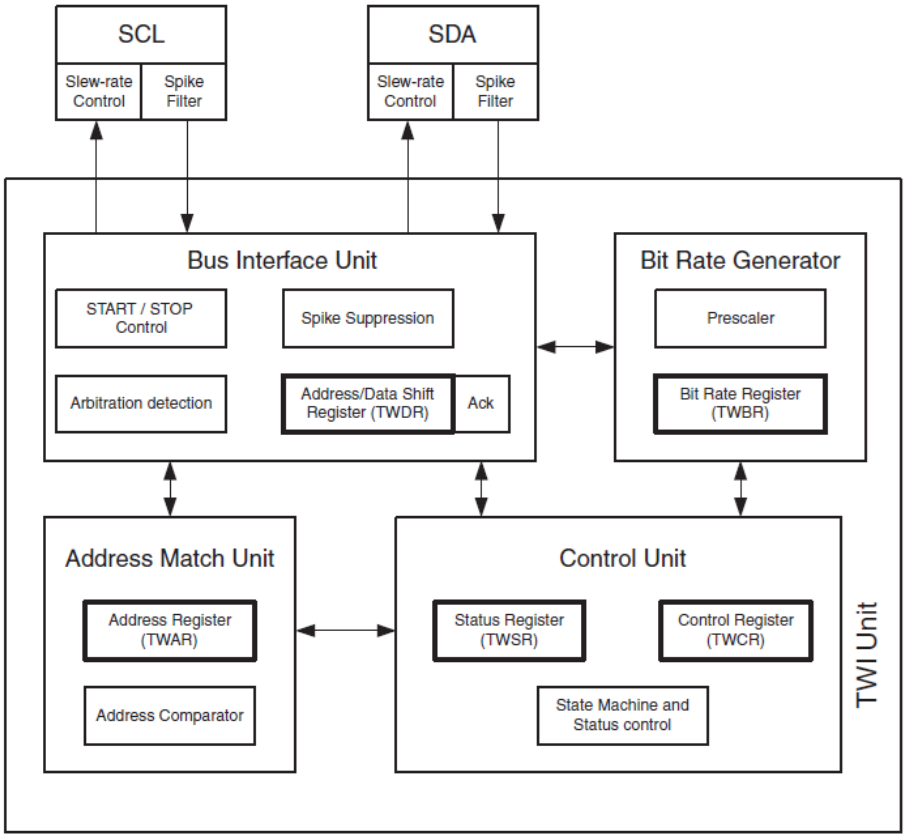
10-bitno naslavljanje



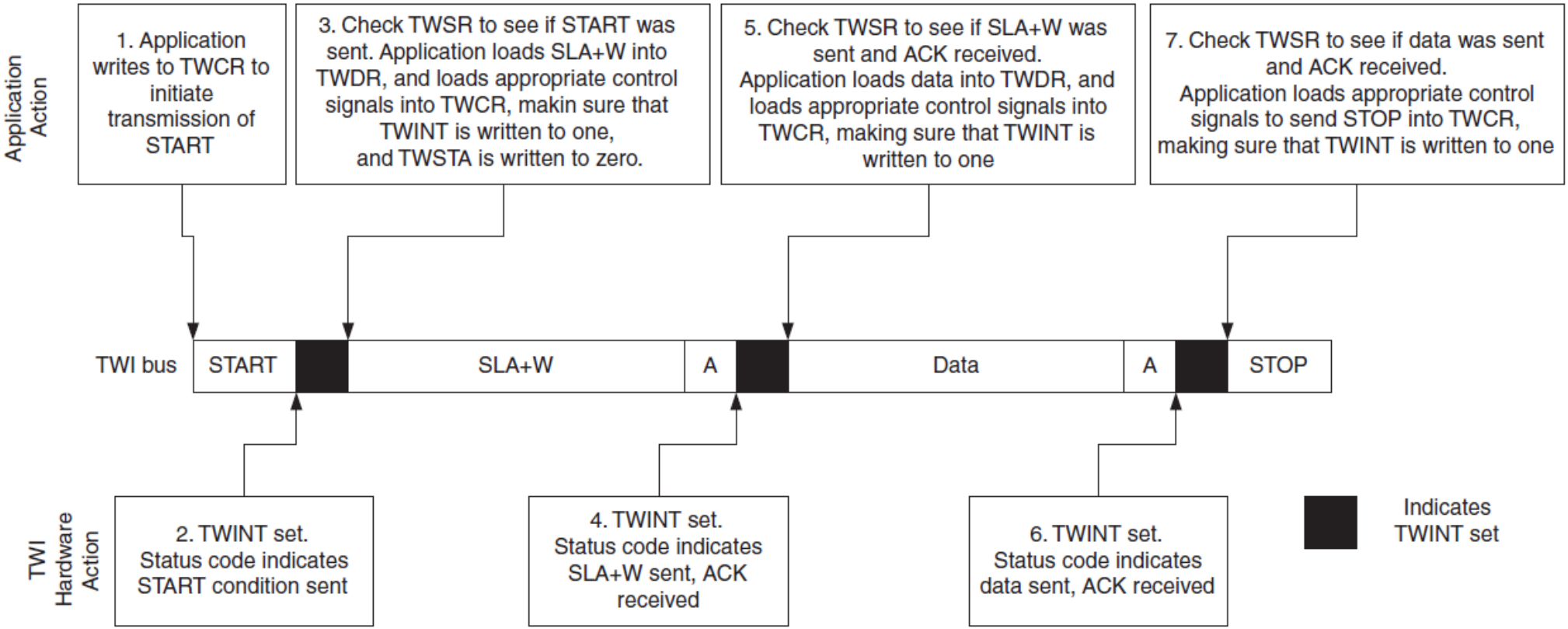
https://www.i2c-bus.org/addressing/10-bit-addressing/

Modul I²C pri AVR (TWI)

- Two TWI instances TWI0 and TWI1
- Simple, yet Powerful and Flexible Communication Interface, only two Bus Lines Needed
- Both Master and Slave Operation Supported
- Device can Operate as Transmitter or Receiver
- 7-bit Address Space Allows up to 128 Different Slave Addresses
- Multi-master Arbitration Support
- Up to 400kHz Data Transfer Speed
- Slew-rate Limited Output Drivers
- Noise Suppression Circuitry Rejects Spikes on Bus Lines
- Fully Programmable Slave Address with General Call Support
- Address Recognition Causes Wake-up When AVR is in Sleep Mode
- Compatible with Philips' I²C protocol



Uporaba vmesnika TWI



Primer kode za vmesnik TWI

```
unsigned char TWI_Start(unsigned char sla, unsigned char n)
{
        TWI_N = n;
        TWI_SLA = sla;    // address+R/W
        TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)|(1<<TWIE); // send start condition
        while (TWI_N!=255);  // čakaj, dokler se prejšnja akcija ne zaključi
        return TWI_SLA;  //return error code
}
```

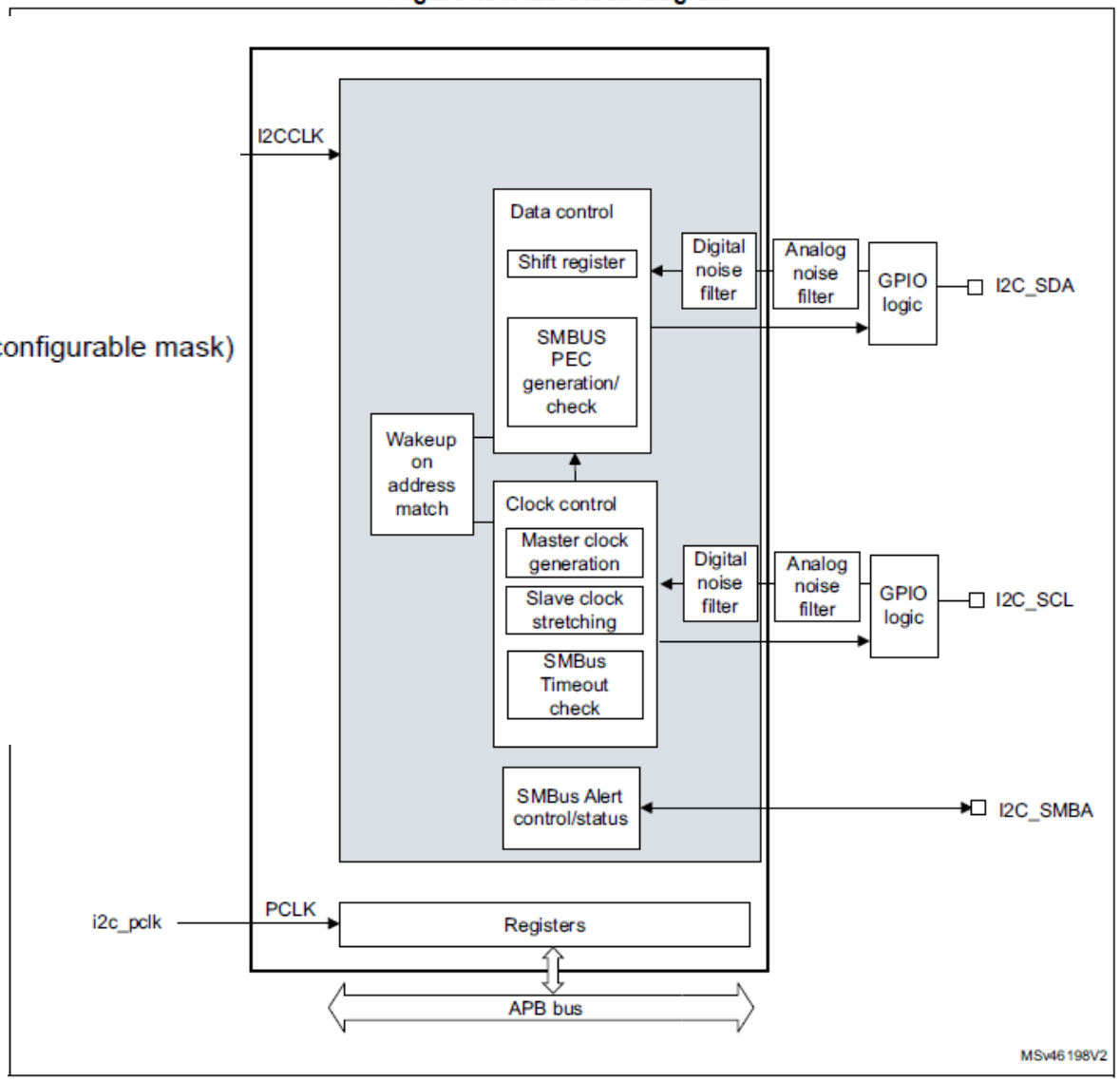
Prekinitvena rutina TWI

```
interrupt [TWI] void twi_isr(void)
{
  switch (TWSR&0xF8)
  {
    case TWI_START: // start condition ack
            TWDR = TWI_SLA; //load address + R/W
            TWI_SLA=0;    // indicates no error
            TWCR = (1<<TWINT) |(1<<TWEN)|(1<<TWIE); // start
            break;

    case TWI_MT_SLA_ACK: //master write slave address ackn
            TWI_N--;
            TWDR = TWI_BUFFER[TWI_N];  // load data
            TWCR = (1<<TWINT) |(1<<TWEN)|(1<<TWIE); // start
            break;

    case TWI_MT_DATA_ACK://master write data transfer ack
            if (TWI_N)
            {
              TWDR = TWI_BUFFER[TWI_N-1]; // load data
              TWCR = (1<<TWINT) |(1<<TWEN)|(1<<TWIE); // start
            }
            else
            {
              TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);// stop
            }
            TWI_N--;
            break;

    case TWI_MR_SLA_ACK: //master read slave address ack
      TWI_N--;
      TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA)|(1<<TWIE); //enable
receiving data
      break;

    case TWI_MR_DATA_ACK:    //master read slave data received ack
      TWI_BUFFER[TWI_N]= TWDR;             //shranimo podatek
      TWI_N--;
      if (TWI_N)
        TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA)|(1<<TWIE);
        //enable receiving data and acknowledge
        else TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWIE);
        // enable receiving data and not acknowledge - last byte read
      break;

    case TWI_MR_DATA_NACK: //master read slave data received not
acknowledged
      TWI_BUFFER[TWI_N]= TWDR; /shranimo zadnji byte podatka
      TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO); // send stop condition
      TWI_N--;
    break;

    default:    //error occured during TWI transmission
      TWI_SLA=TWSR&0xF8;      // store TWSR for error inspection
      TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO); // send stop condition
      TWI_N=255;              // indicates TWI termination

    break;
  }
}
```

Modul I2C pri STM32

- I²C bus specification rev03 compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode
  - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
  - All 7-bit addresses acknowledge mode
  - General call
  - Programmable setup and hold times
  - Easy to use event management
  - Optional clock stretching
  - Software reset
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters

Figure 630. I2C block diagram



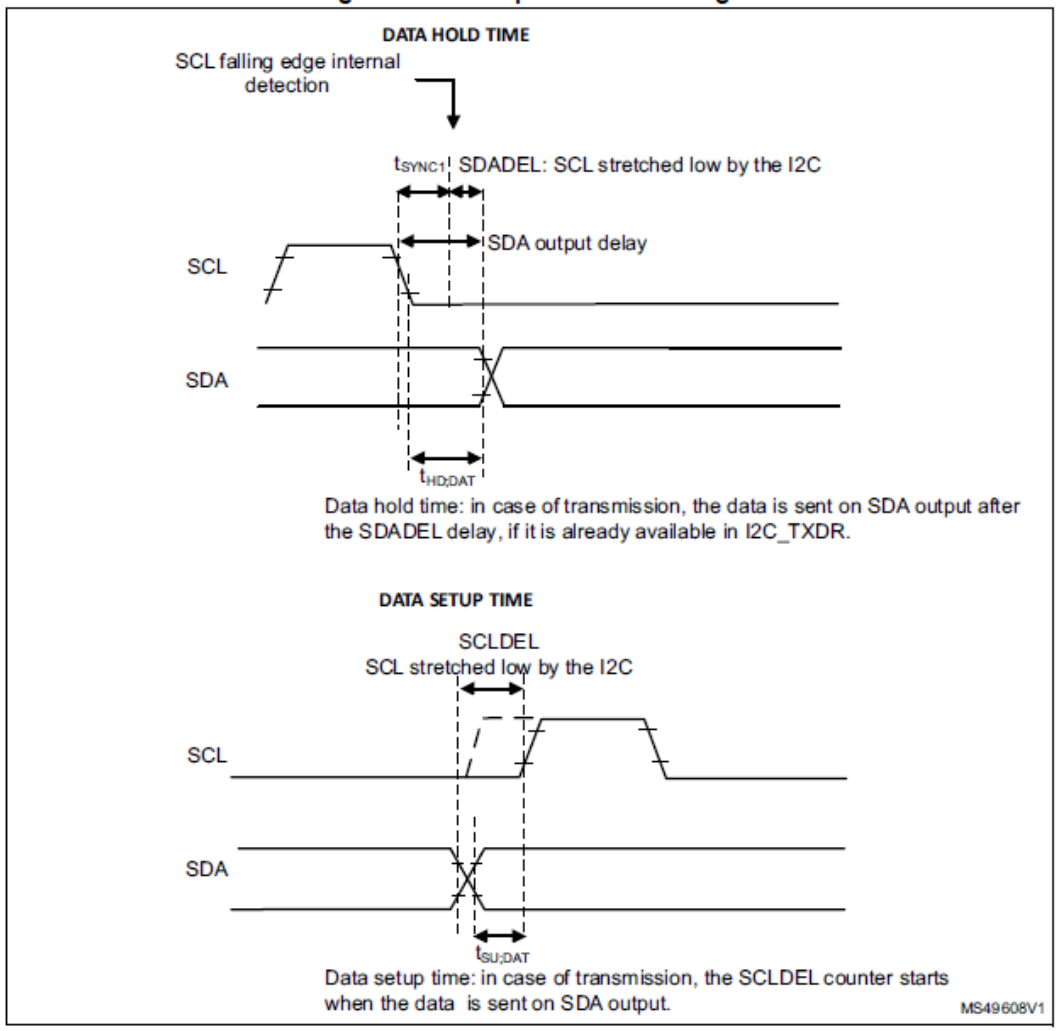
Analogni in digitalni filter za odpravo šuma

Table 381. Comparison of analog vs. digital filters

-	Analog filter	Digital filter
Pulse width of suppressed spikes	≥ 50 ns	Programmable length from 1 to 15 I2C peripheral clocks
Benefits	Available in Stop mode	– Programmable length: extra filtering capability versus standard requirements – Stable length
Drawbacks	Variation vs. temperature, voltage, process	Wakeup from Stop mode on address match is not available when digital filter is enabled

Nastavitve zakasnitev

Figure 632. Setup and hold timings



Strojno podprt protokol

Hardware transfer management

The I2C has a byte counter embedded in hardware in order to manage byte transfer and to close the communication in various modes such as:

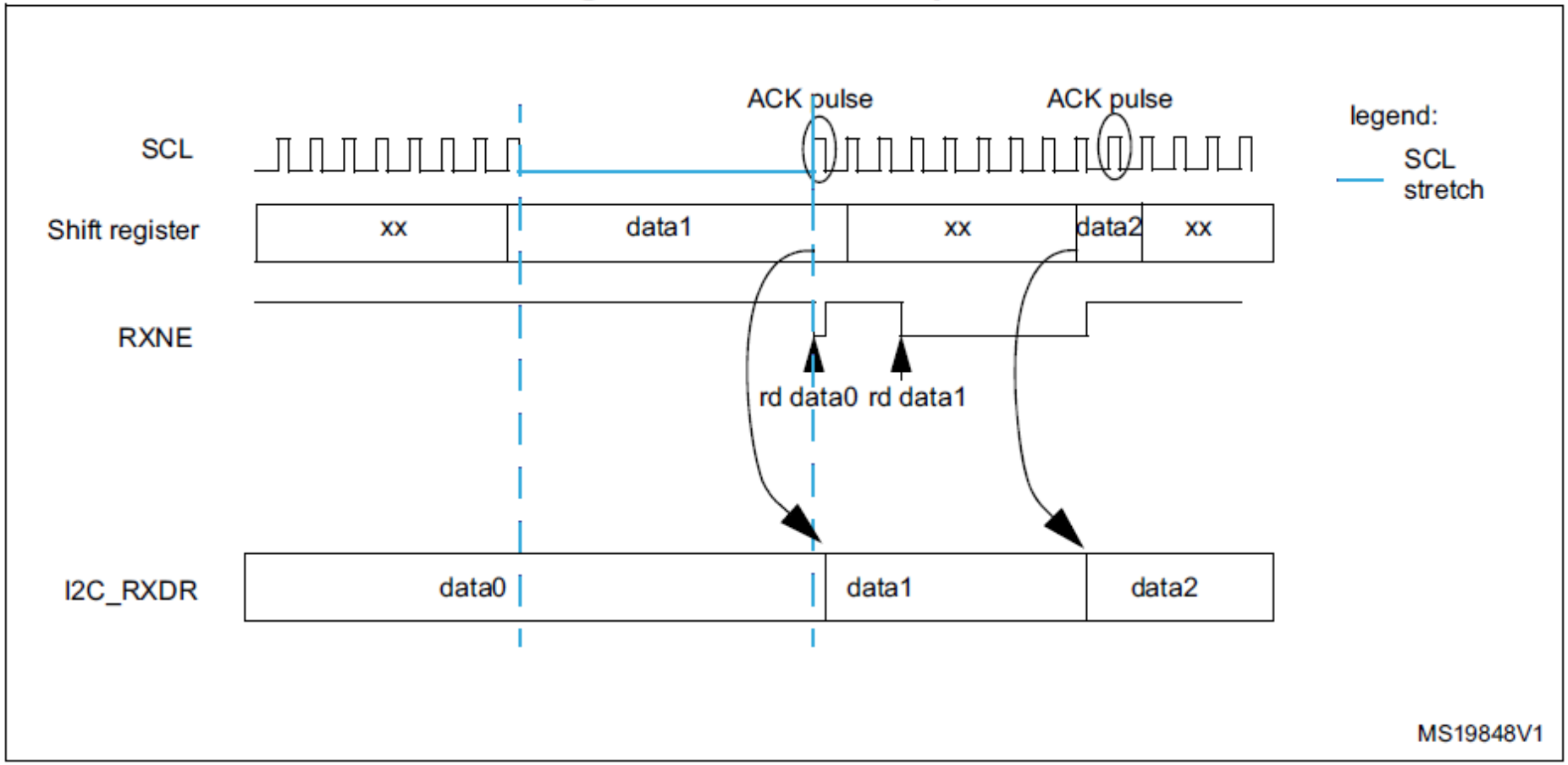
- NACK, STOP and ReSTART generation in master mode
- ACK control in slave receiver mode
- PEC generation/checking when SMBus feature is supported

The byte counter is always used in master mode. By default it is disabled in slave mode, but it can be enabled by software by setting the SBC (Slave Byte Control) bit in the I2C_CR2 register.

RM0440

Primer sprejema podatka

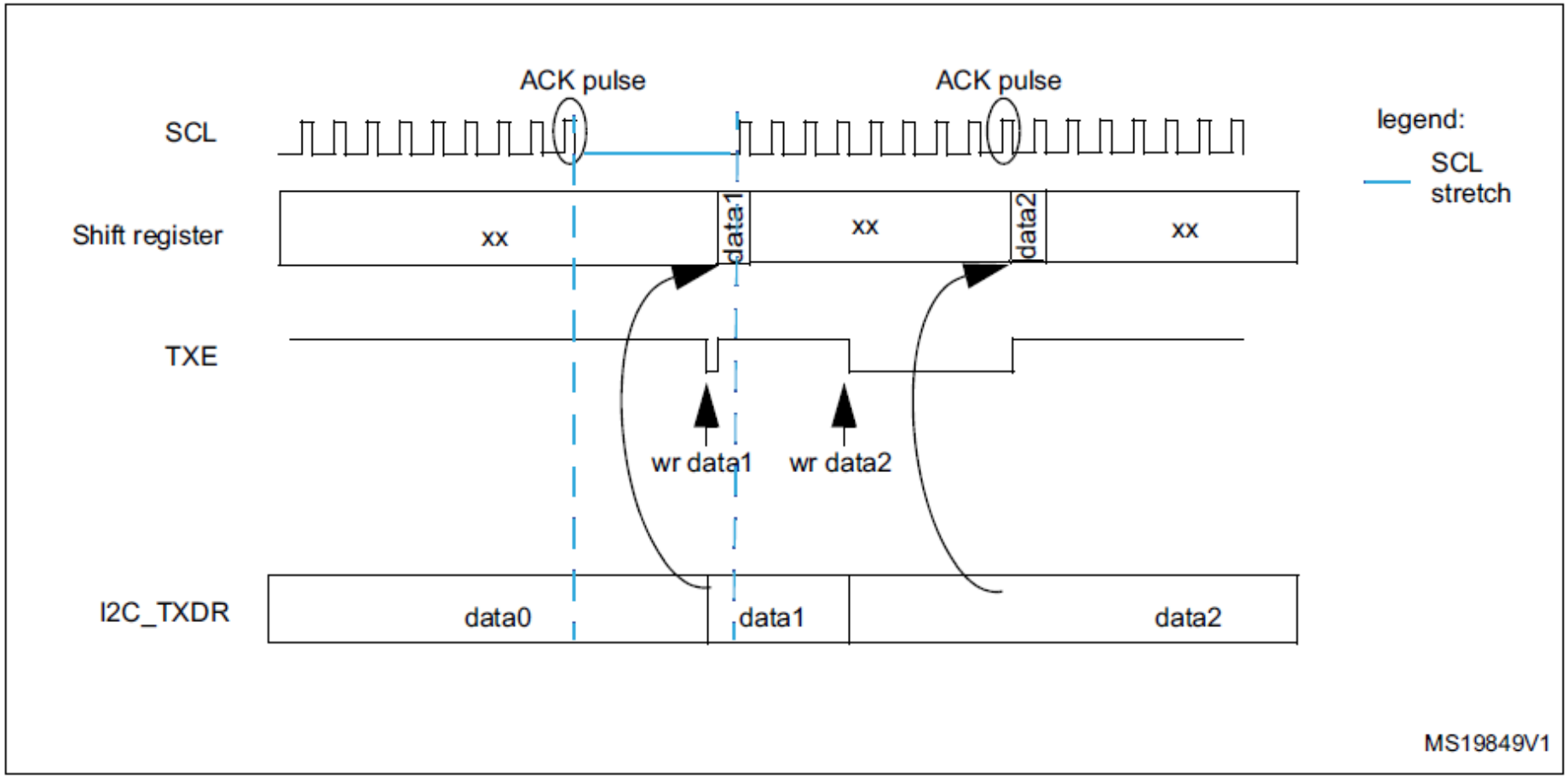
Figure 634. Data reception



RM0440

Primer pošiljanja podatka

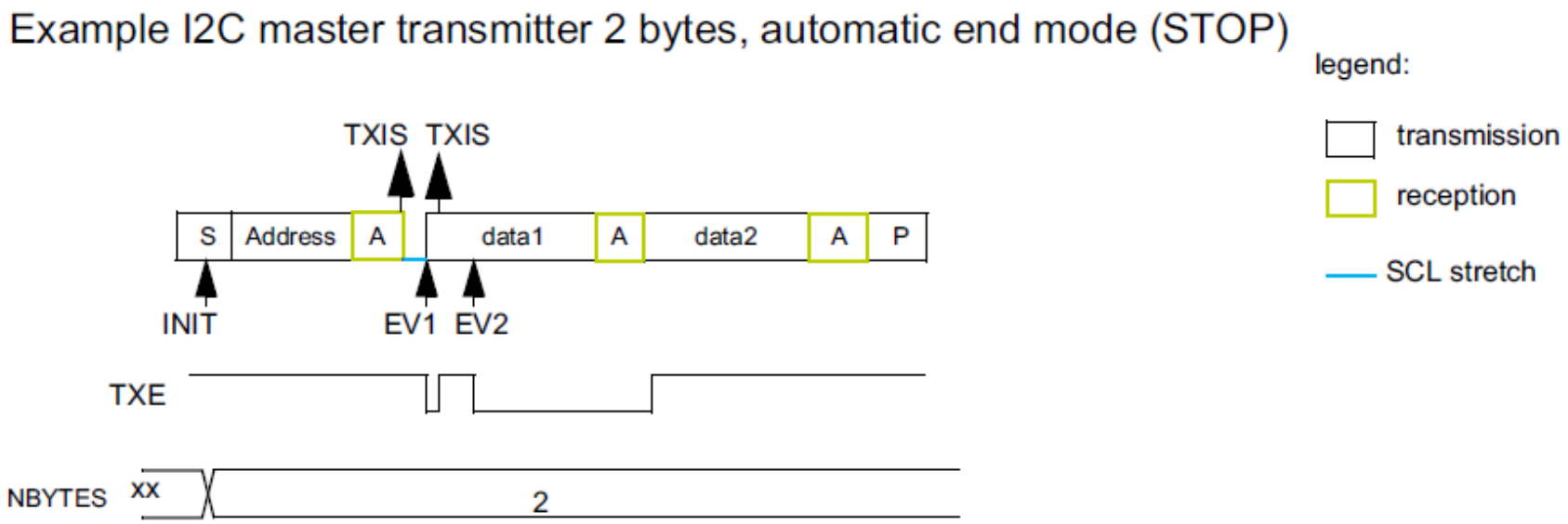
Figure 635. Data transmission



RM0440

Master pošiljanje dveh podatkov

Figure 649. Transfer bus diagrams for I2C master transmitter



INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START

EV1: TXIS ISR: wr data1

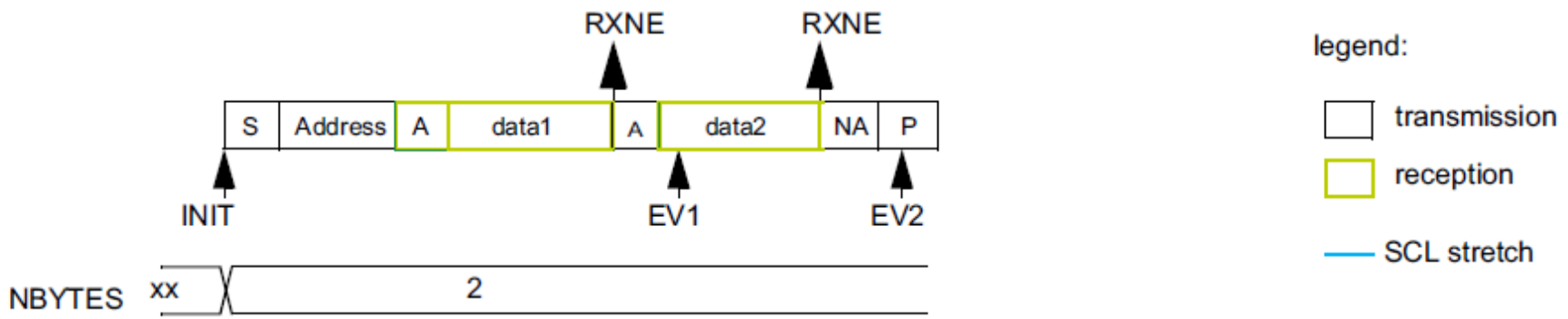
EV2: TXIS ISR: wr data2

RM0440

Master sprejem dveh podatkov

Figure 652. Transfer bus diagrams for I2C master receiver

Example I2C master receiver 2 bytes, automatic end mode (STOP)



INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

RM0440

Detekcija napak

Bus Error (BERR) – This error occurs when the I2C interface detects an external Stop or Start condition during an address or a data transfer.

Acknowledge Failure (AF) – This error occurs when the interface detects a non-acknowledge bit.

Arbitration Lost (ARLO) – This error occurs when the I2C interface detects an arbitration lost condition.

Overrun/Underrun Error (OVR) – An **overrun error** can occur in slave mode when clock stretching is disabled and the I2C interface is receiving data. The interface has received a byte and the data in DR has not been read before the next byte is received by the interface. **Underrun error** can occur in slave mode when clock stretching is disabled and the I2C interface is transmitting data. The interface has not updated the DR with the next byte before the clock comes for the next byte.

Nastavitve I2C v CubeMX

Categories | A->Z

System Core
- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- WWDG

Analog

Timers

Connectivity
- FDCAN1
- FDCAN2
- FDCAN3
- FMC
- I2C1
- I2C2
- I2C3
- I2C4
- IRTIM
- LPUART1
- QUADSPI1
- SPI1

Mode

I2C | I2C

Configuration

Reset Configuration

NVIC Settings | DMA Settings | GPIO Settings | Parameter Settings | User Constants

Configure the below parameters :

Search (Ctrl+F)

Timing configuration	
Custom Timing	Disabled
I2C Speed Mode	Standard Mode
I2C Speed Frequency (K...	100
Rise Time (ns)	10
Fall Time (ns)	10
Coefficient of Digital Filter	5
Analog Filter	Enabled
Timing	0x30908CCF
Slave Features	
Clock No Stretch Mode	Disabled
General Call Address De...	Disabled
Primary Address Length ...	7-bit
Dual Address Acknowled...	Disabled
Primary slave address	0

41.7.5 I2C timing register (I2C_TIMINGR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: No wait states

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	rw					rw	rw	rw	rw	rw	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	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Uporaba brez prekinitev

```
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);

HAL_StatusTypeDef HAL_I2C_Slave_Transmit(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_StatusTypeDef HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);

HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout);
```

https://forum.digikey.com/t/using-the-stm32cube-hal-i2c-driver-in-master-mode/15122

Uporaba s prekinitvami

```
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);
HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Write_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Mem_Read_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size,
uint32_t XferOptions);
HAL_StatusTypeDef HAL_I2C_EnableListen_IT(I2C_HandleTypeDef *hi2c);
HAL_StatusTypeDef HAL_I2C_DisableListen_IT(I2C_HandleTypeDef *hi2c);
HAL_StatusTypeDef HAL_I2C_Master_Abort_IT(I2C_HandleTypeDef *hi2c, uint16_t DevAddress);
```

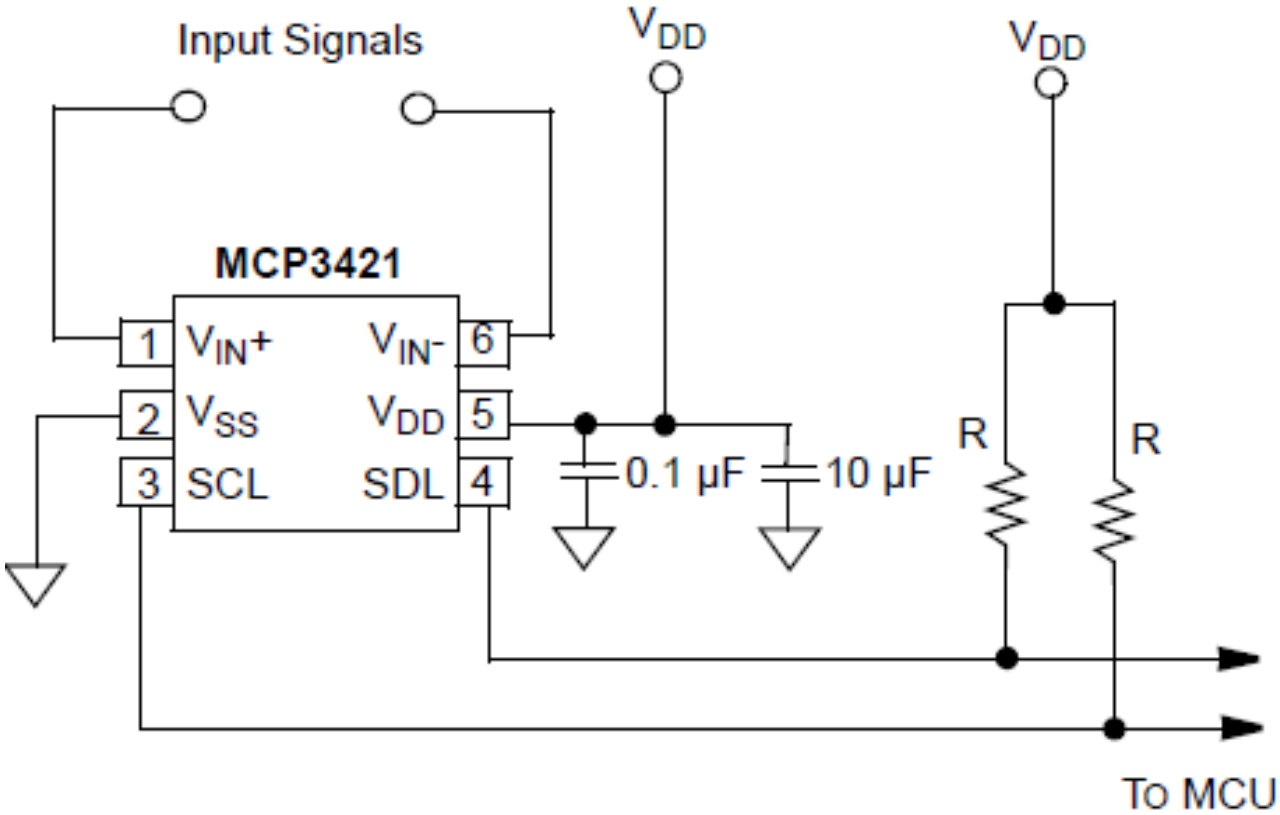
https://deepbluembedded.com/stm32-i2c-tutorial-hal-examples-slave-dma/

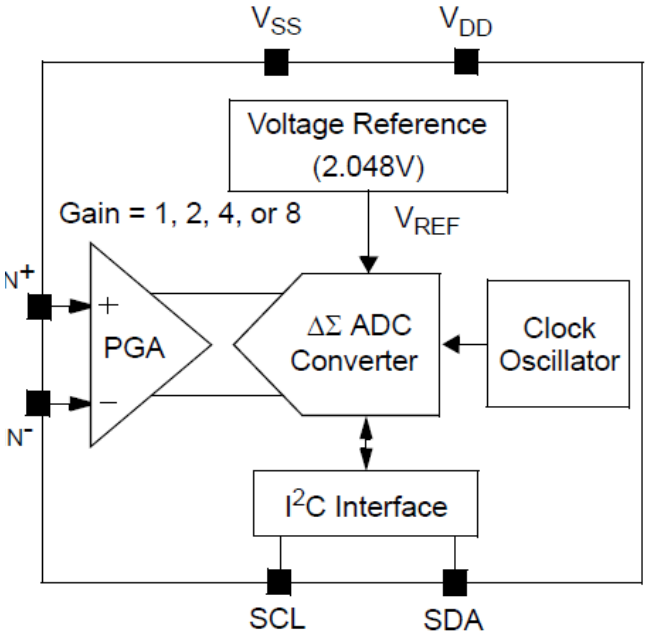
Uporaba z DMA

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size);

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t XferOptions);

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size, uint32_t XferOptions);
```

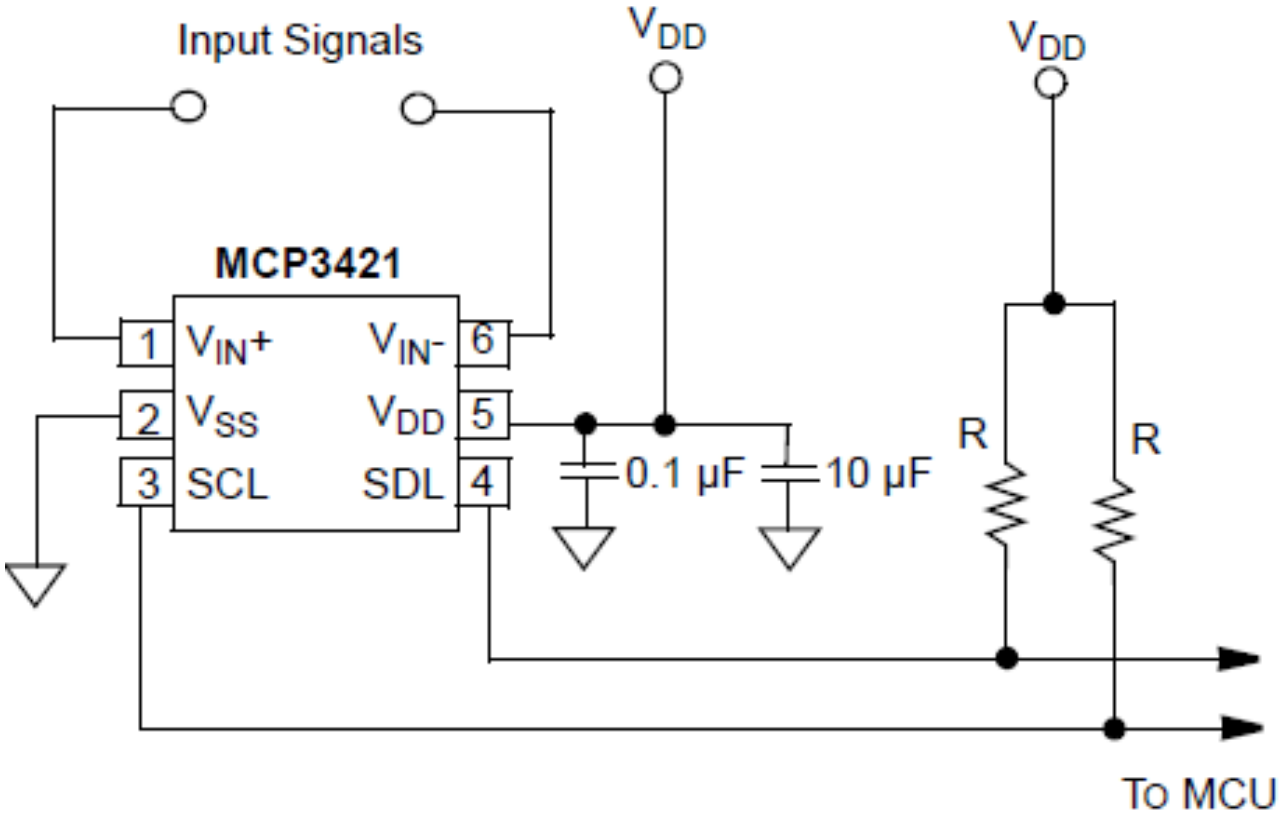
https://deepbluembedded.com/stm32-i2c-tutorial-hal-examples-slave-dma/

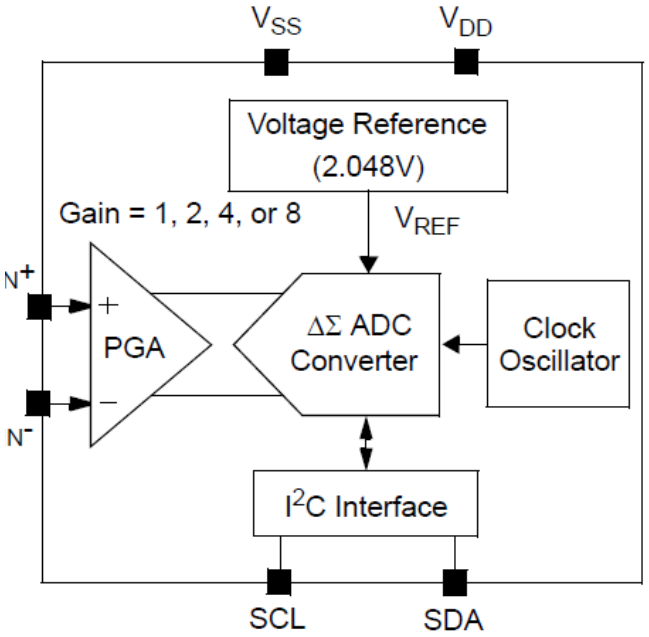
18-bitni Σ-Δ AD pretvornik MCP3421



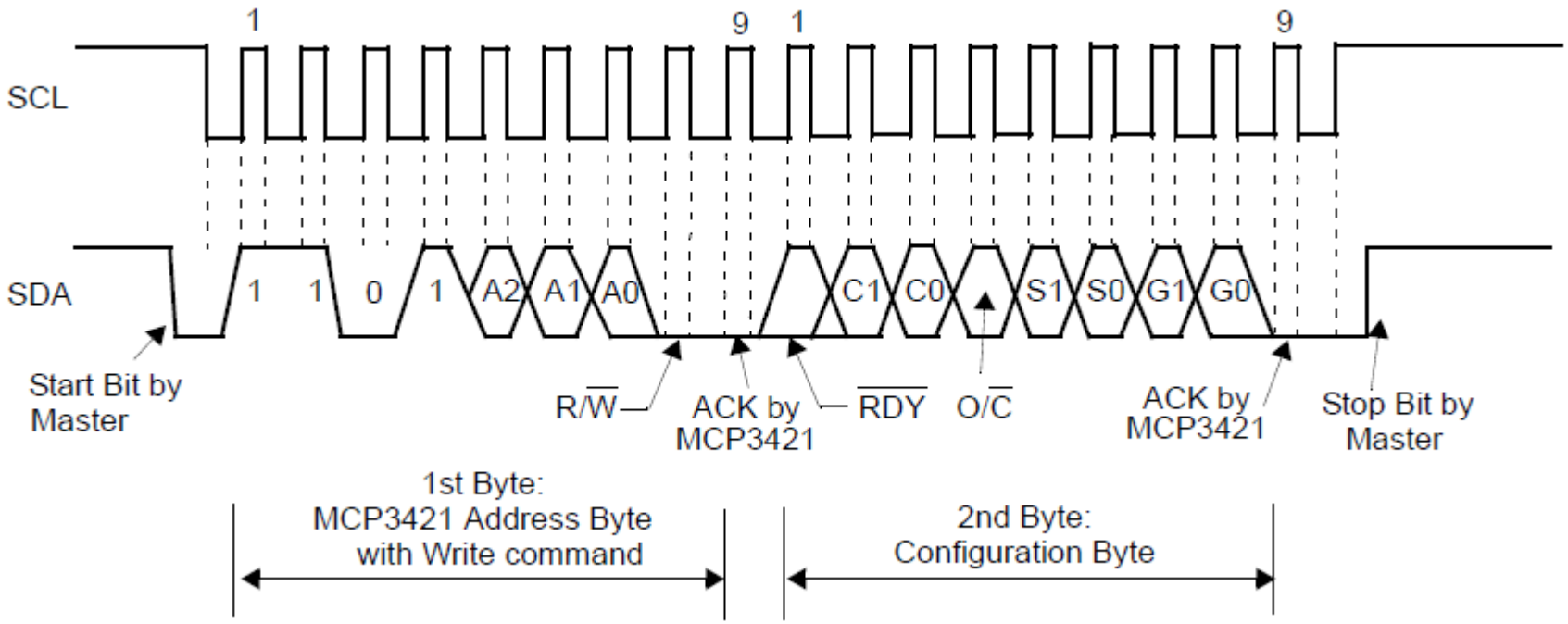


18-bitni Σ-Δ AD pretvornik MCP3421

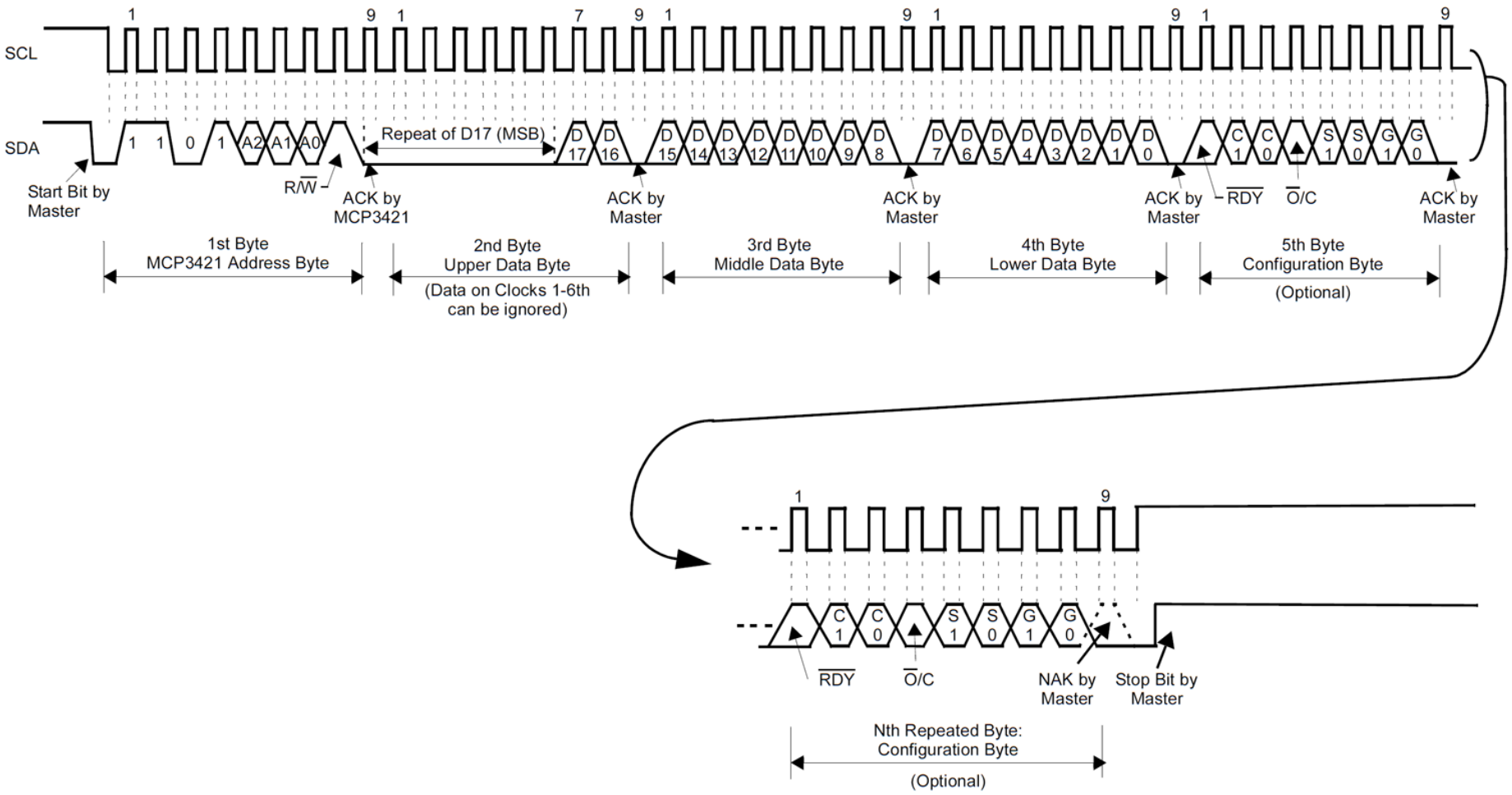




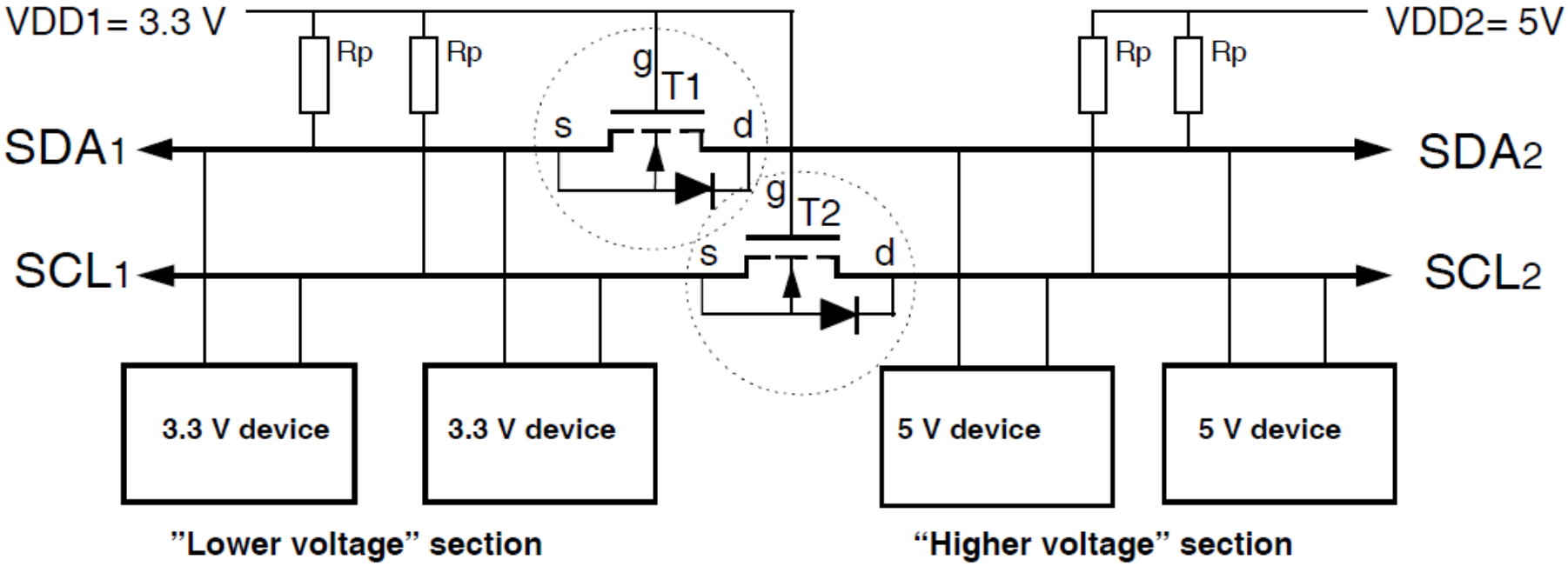
Pisanje nastavitve



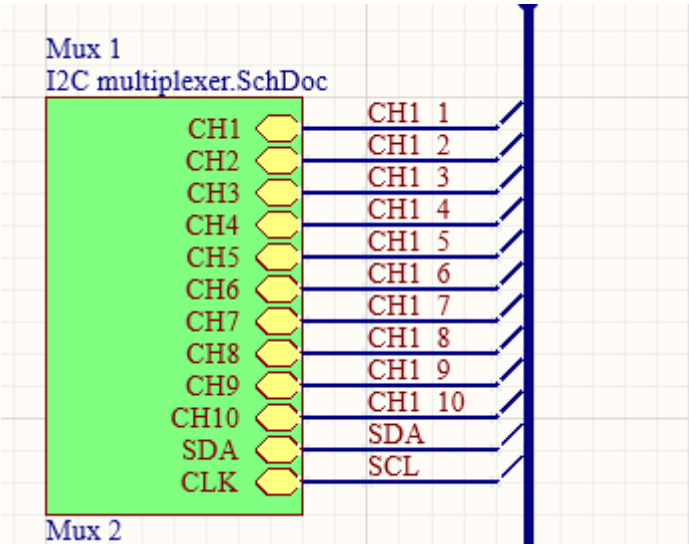
Branje vrednosti

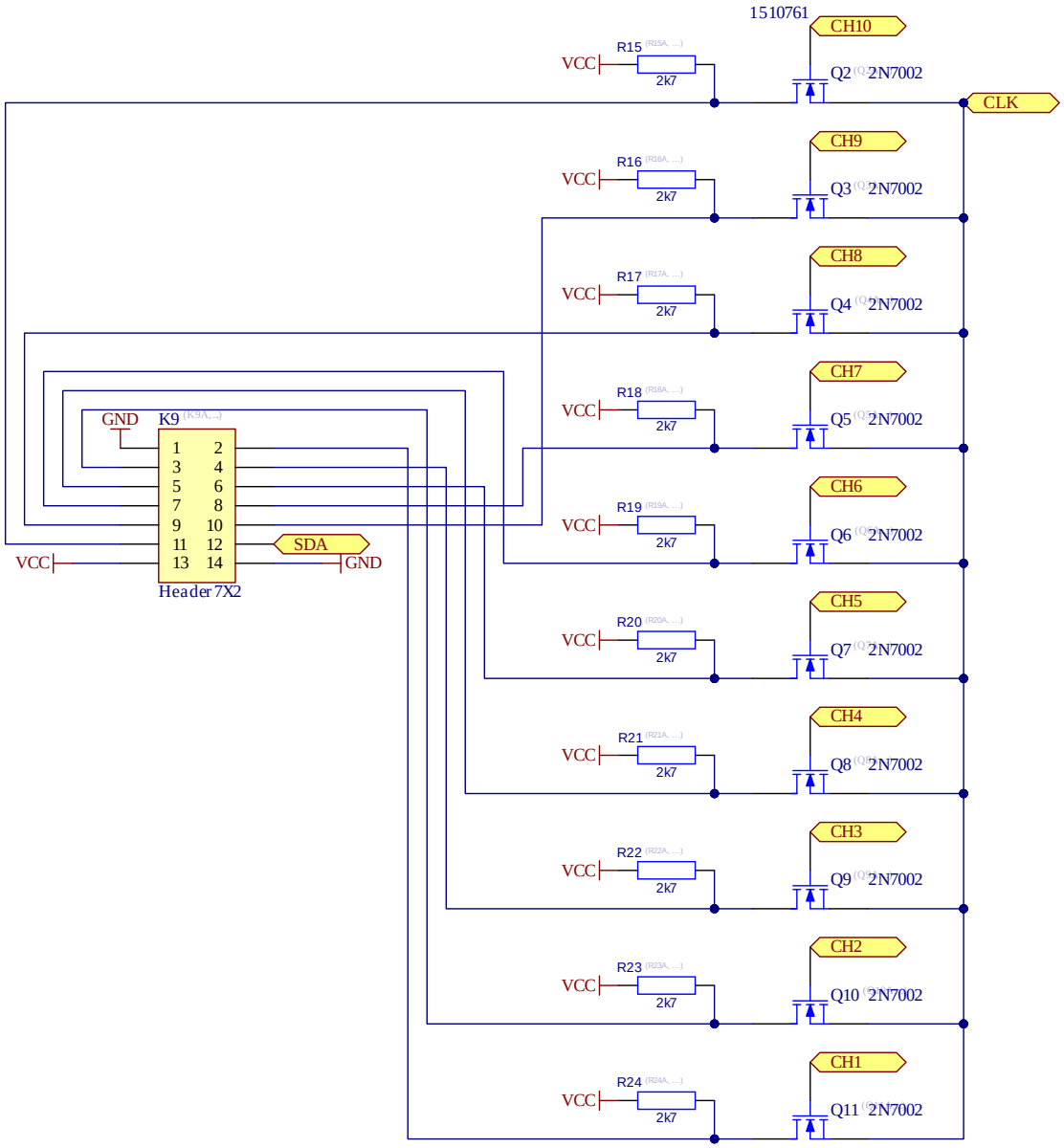


Pretvarjanje logičnih nivojev



I2C multiplekser





I2C multiplekser