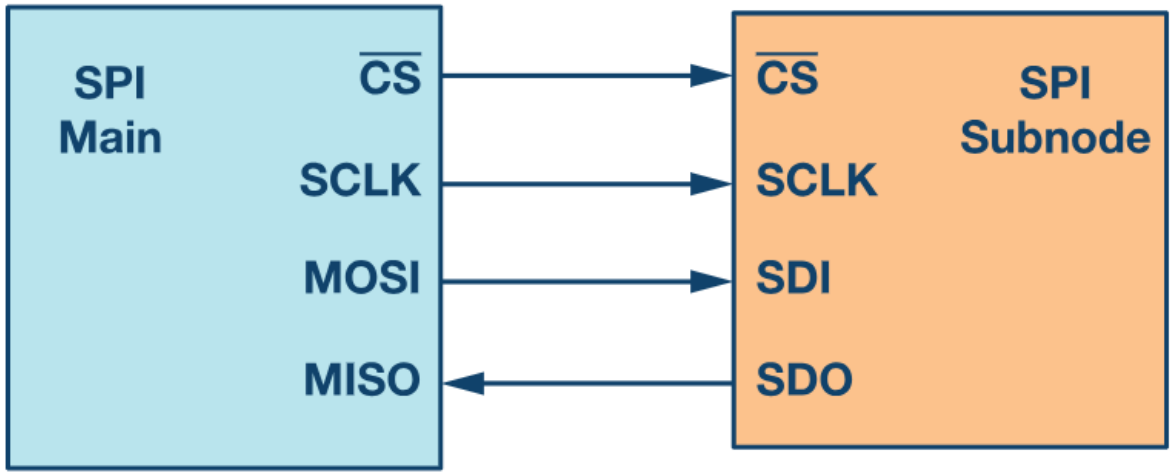
فصل ششم : SPI and I2C

6

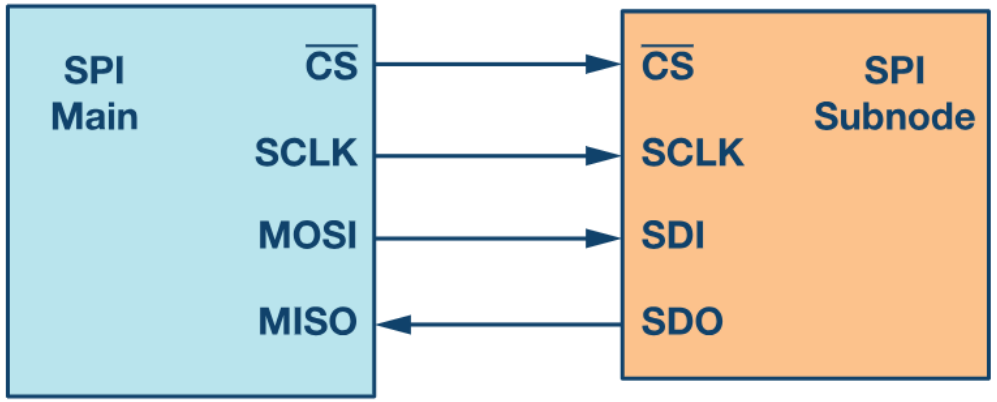
Serial Peripheral Interface (SPI)



- یکی از پروتکل‌های محبوب ارسال و دریافت دیتا به صورت سریال می‌باشد.
- اغلب بین پردازنده و یک Peripheral استفاده می‌شود نظیر ADCs، DACs، Sensors، SRAMs و ...
- رایج‌ترین نوع SPI، از 4 پایه استفاده می‌کند.
  - Master Output, Slave Input (MOSI)
  - Master Input, Slave Output (MISO)
  - Serial Clock (SCLK/SC)
  - Chip Select (CS)
- باتوجه به وجود پایه Serial Clock، این پروتکل از نوع سنکرون می‌باشد! (کاملا مشابه با USART)

Serial Peripheral Interface (SPI)

- در حالت استاندارد، هر پروتکل SPI می‌تواند شامل یک Master و چندین Slave باشد. (چگونه!؟)
- سیگنال Clock همواره توسط Master ارسال می‌گردد. (بنابراین این پایه در سمت Master از نوع خروجی در سمت Slave از نوع ورودی است.
- سیگنال Chip Select به منظور انتخاب Slave می‌باشد. (به اینصورت که به ازای هر Slave، یک سیگنال Chip Select اضافه می‌شود. در سمت Master از نوع خروجی و در سمت Slave از نوع ورودی است)
- پایه MOSI برای ارسال دیتا از Master به Slave می‌باشد.
- پایه MISO برای ارسال دیتا از Slave به Master می‌باشد.
- باتوجه به وجود پایه‌های MISO و MOSI، این پروتکل یک ارتباط دوطرفه می‌باشد.



تبادل داده در SPI

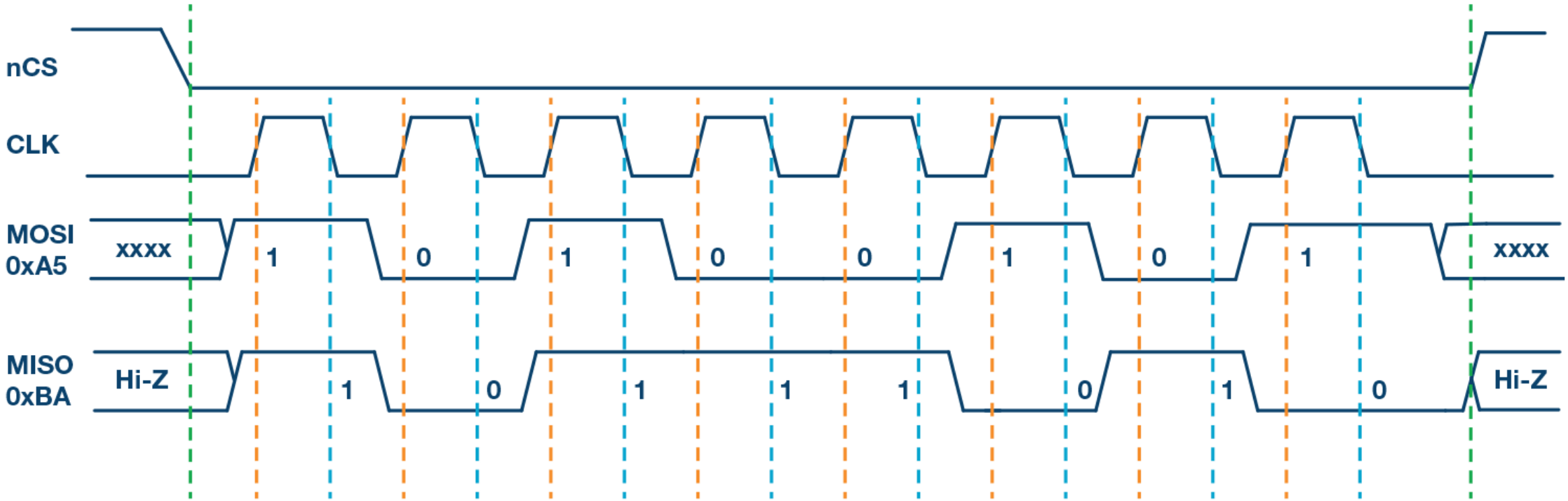
- انتقال داده با انتخاب Slave آغاز می‌شود. (پایه CS مربوط به Slave مربوط Low می‌شود.
- در گام بعد سیگنال Clk بایستی توسط Master تولید گردد.
- باتوجه به اینکه ارتباط SPI ازنوع سنکرون می‌باشد، Master و Slave می‌توانند بصورت همزمان دیتا ارسال و دریافت نمایند. (دیتا از سمت Master به کمک پایه MOSI، به صورت سریال برای Slave ارسال می‌گردد. همچنین از سمت Slave به کمک پایه MISO، به صورت سریال اطلاعات شیفت داده شده و برای Master ارسال می‌گردد.)
- عملیات شیفت داده و نمونه برداری به کمک لبه‌های Clock تنظیم می‌شود (لبه‌های بالارونده و پایین رونده)

تنظیمات مربوط به Clock

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

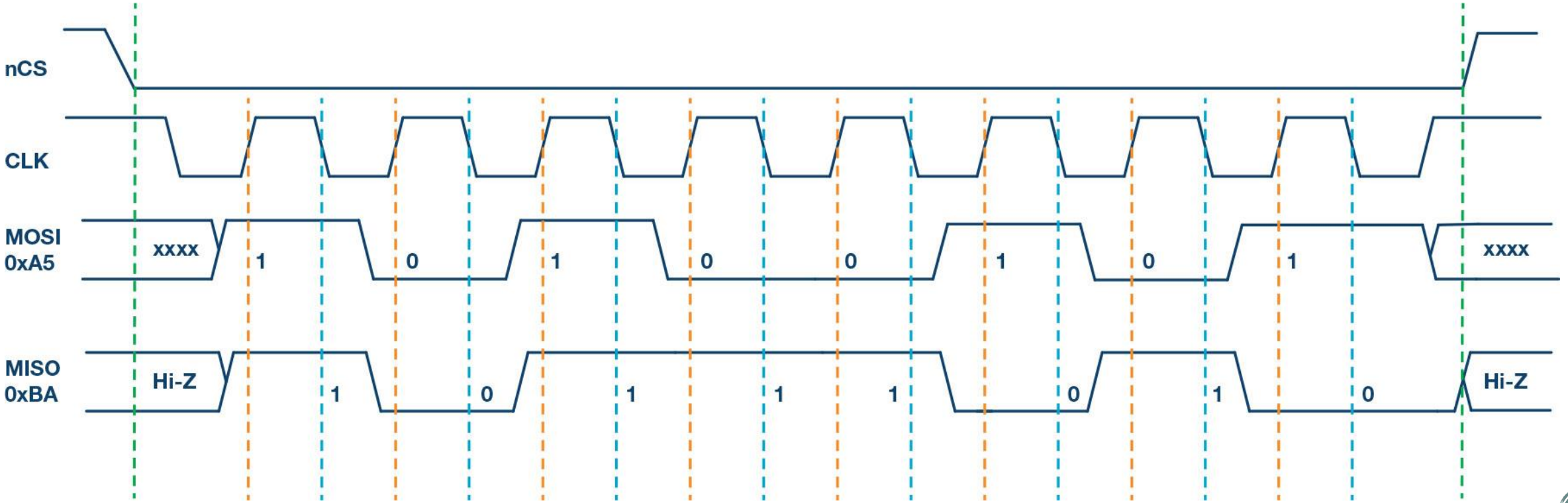
تنظیمات مربوط به Clock

- CPOL = 0 CPHA = 0
- خطوط نارنجی نمونه برداری خطوط آبی شیفت داده



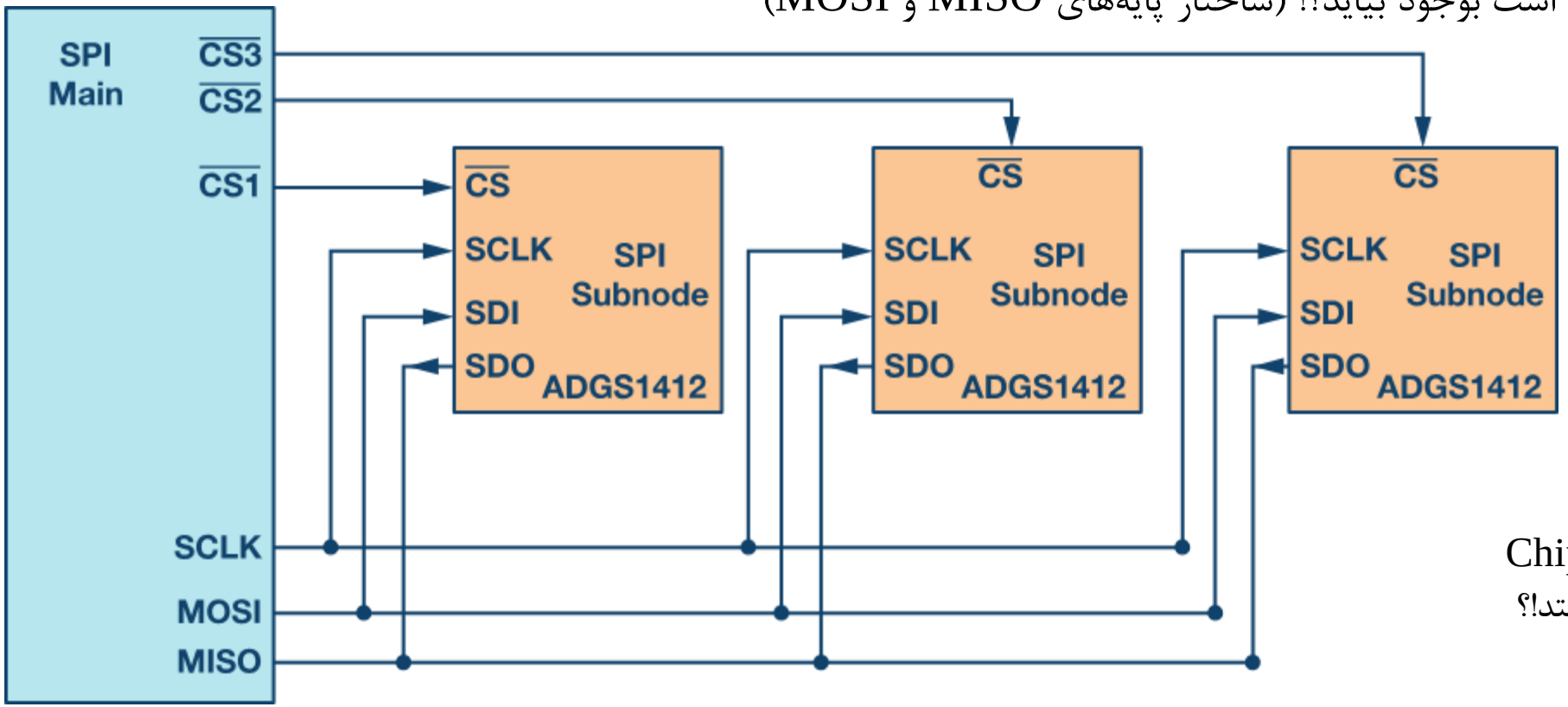
تنظیمات مربوط به Clock

- CPOL = 1 CPHA = 1
- خطوط نارنجی نمونه برداری خطوط آبی شیفت داده



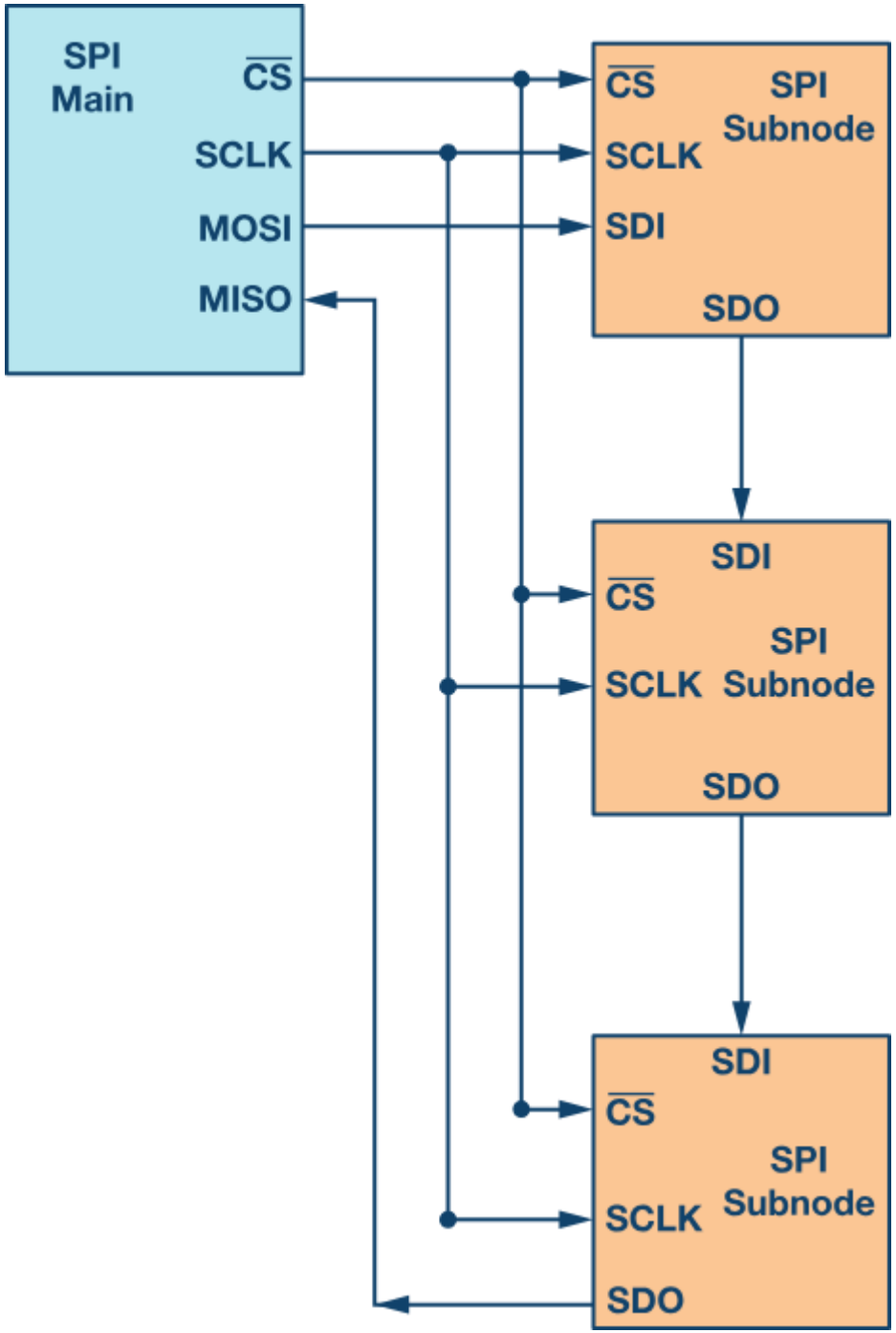
پروتکل SPI با چند Slave

- در این ساختار به ازای هر Slave، بایستی یک پایه در Master اضافه شود (برای Chip Select)
- اگر تعداد بلوک‌های Slave زیاد باشد، به منظور کاهش تعداد پایه‌های Master، می‌توان از دیکدر استفاده نمود.
- در این ساختار چه مشکلی ممکن است بوجود بیاید؟! (ساختار پایه‌های MISO و MOSI)

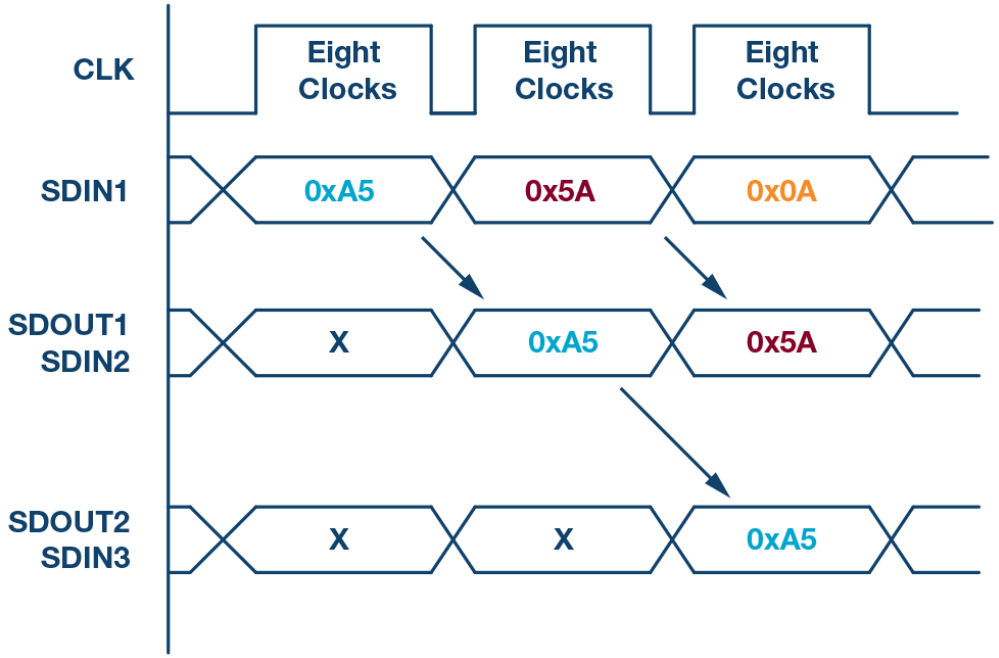


اگر بیش از یک Chip Select
فعال شود، چه اتفاقی می‌افتد!؟

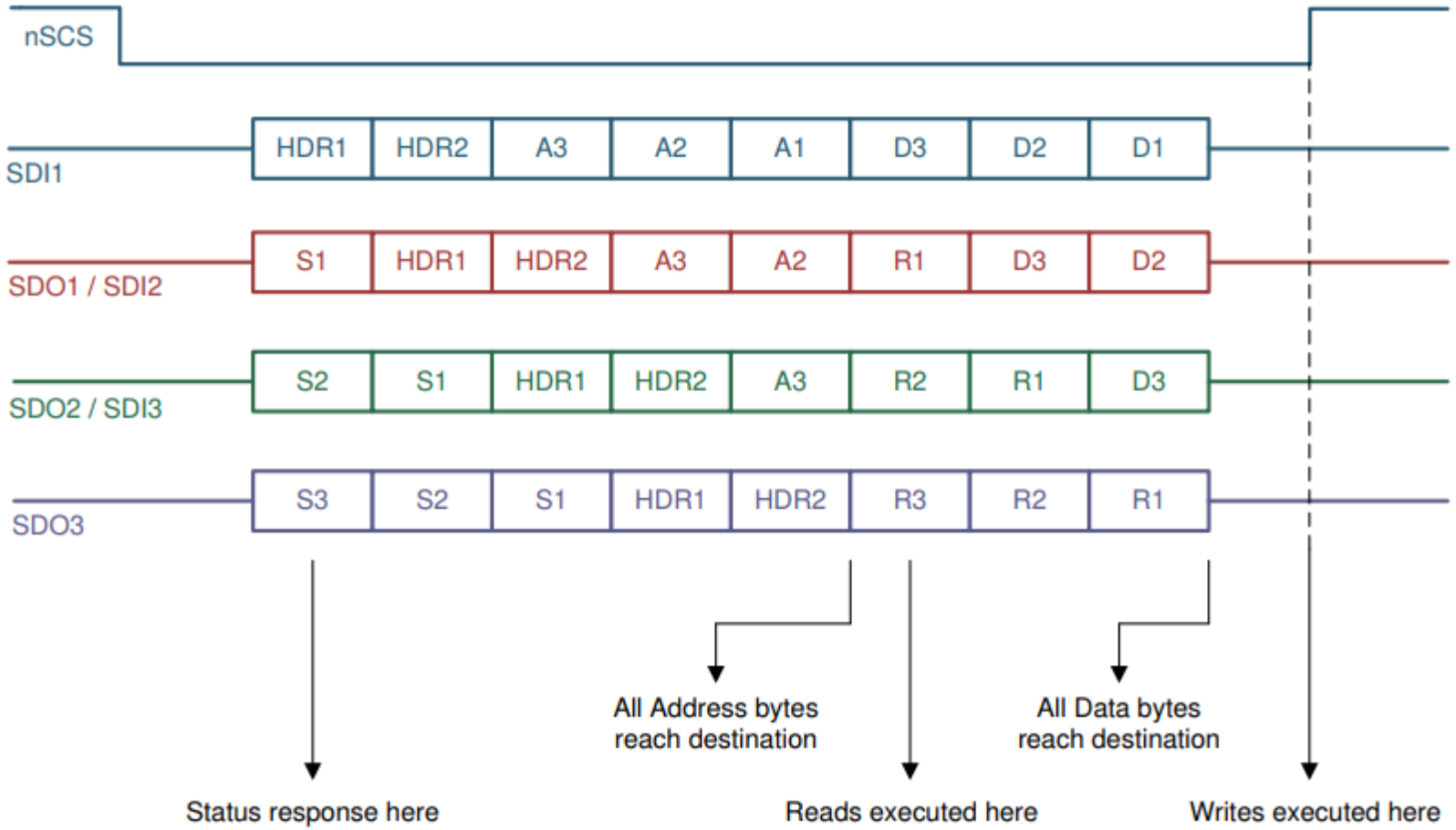
Daisy-Chain Method



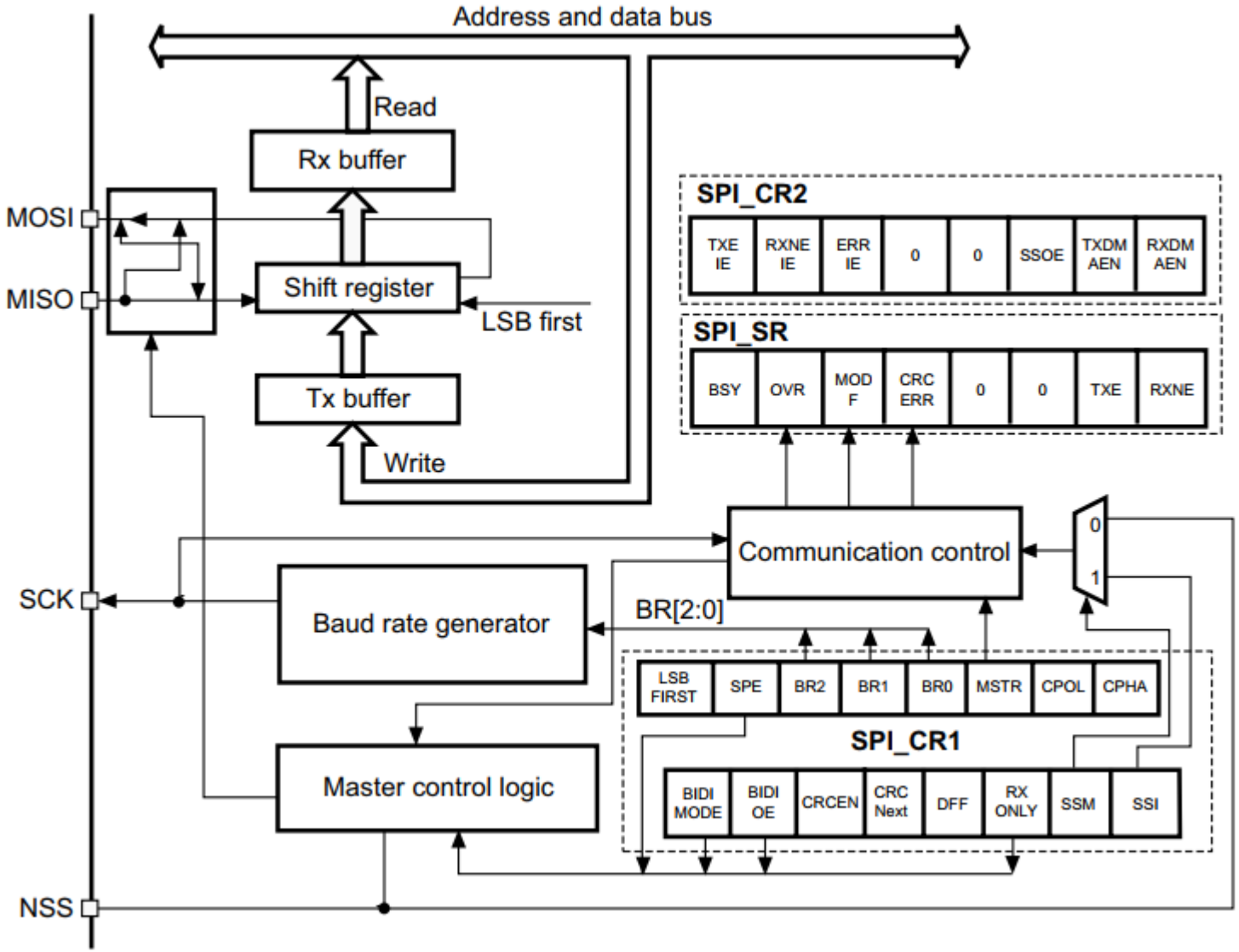
- در این ساختار پایه‌های CS، همگی به یکدیگر متصل می‌شوند.
- در این به صورت سری اطلاعات از یک Slave به Slave بعدی منتقل می‌شود.
- مزایا :
- معایب :



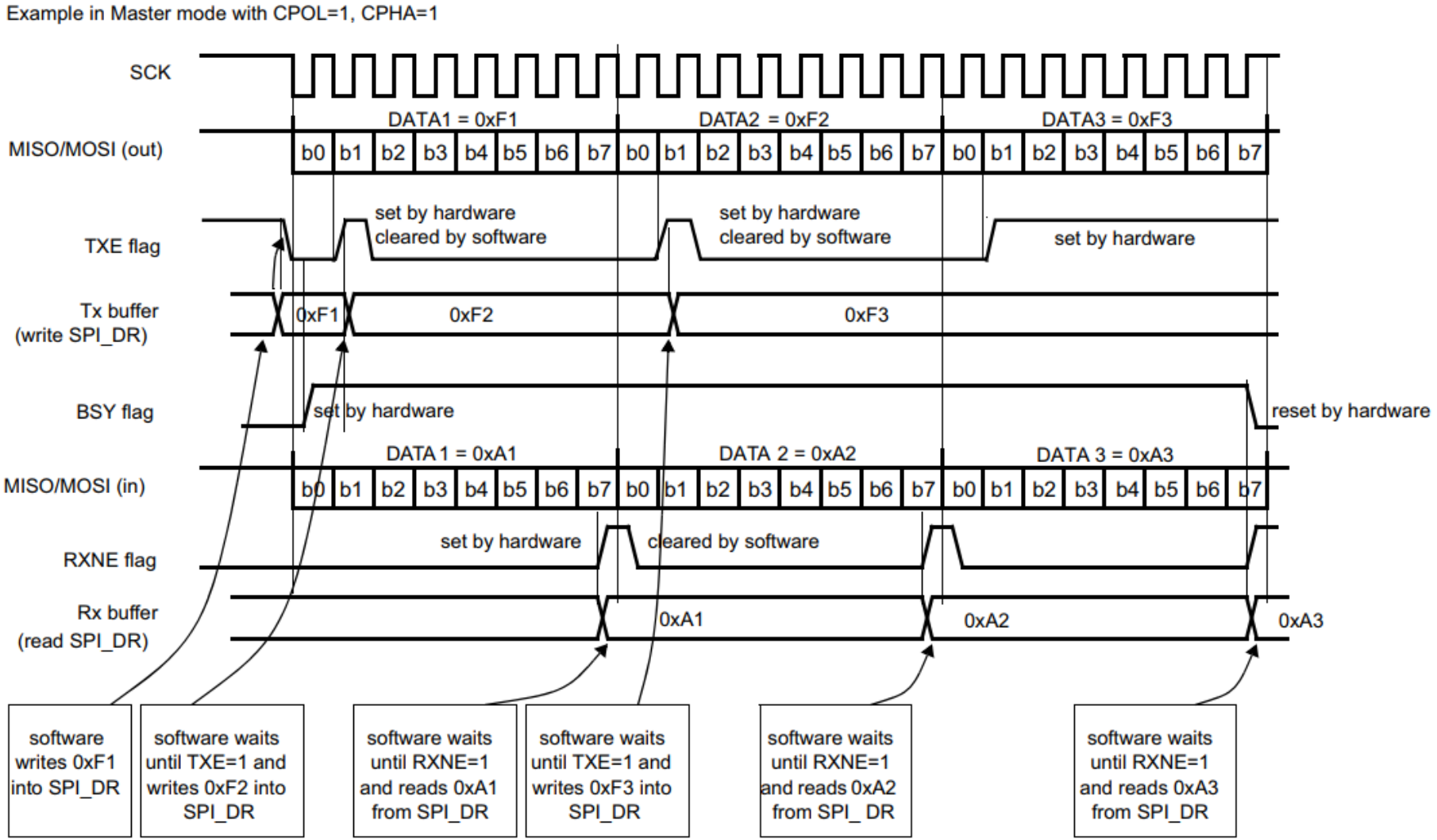
Daisy-Chain Method



STM32F10x SPI



STM32F10x SPI



توابع HAL کاربردی در SPI

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

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

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

توابع HAL کاربردی در SPI

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)

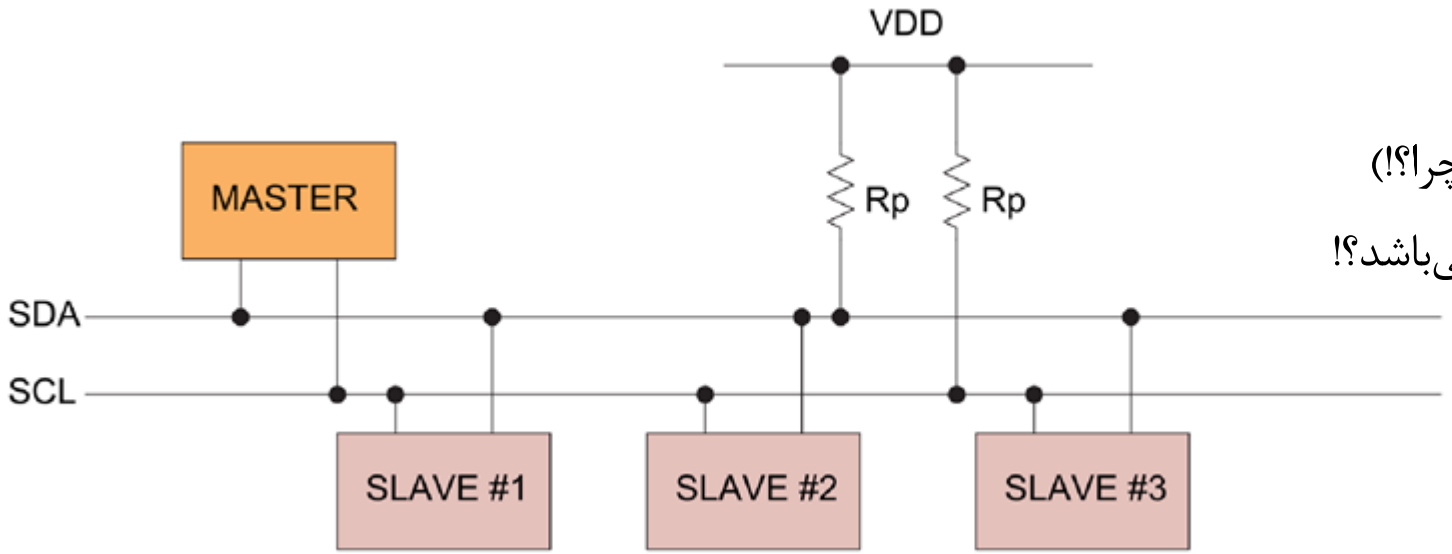
void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

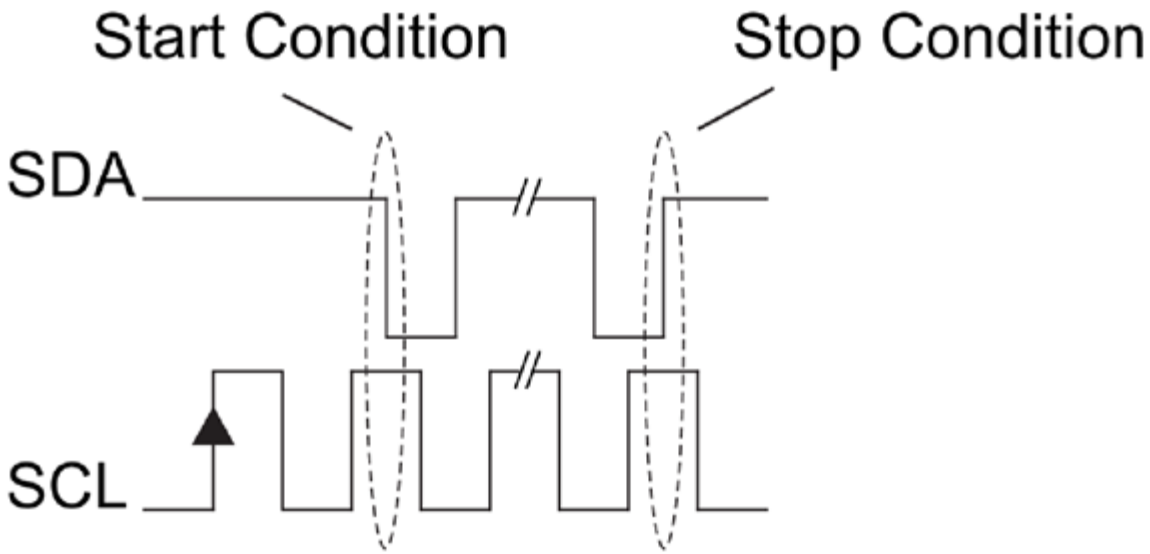
Inter-Integrated Circuit (I2C)

- با توجه به نام این ارتباط بیشتر به منظور تبادل داده میان دو یا چند IC طراحی شده است (ارتباطات کوتاه)
- ارتباط بر پایه Master و Slave می‌باشد. (وظایف Master شامل تامین Clock، ارسال آدرس Slave و خواندن/نوشتن از/در Slave می‌باشد).
- Slave تنها زمانی پاسخ می‌دهد که از سوی Master فراخوانی شده باشد. بنابراین هیچگاه Slave شروع‌کننده تبادل داده نیست.
- ارتباط ار نوع سریال و شامل دو پایه می‌باشد.
  - Serial Data Line (SDA)
  - Serial Clock Line (SCL)
- توجه شود بایستی زمین تمامی مدار مشترک باشد! (چرا؟!)
- ارتباط باوجود یک پایه دیتا، ولی به صورت دوطرفه می‌باشد؟!



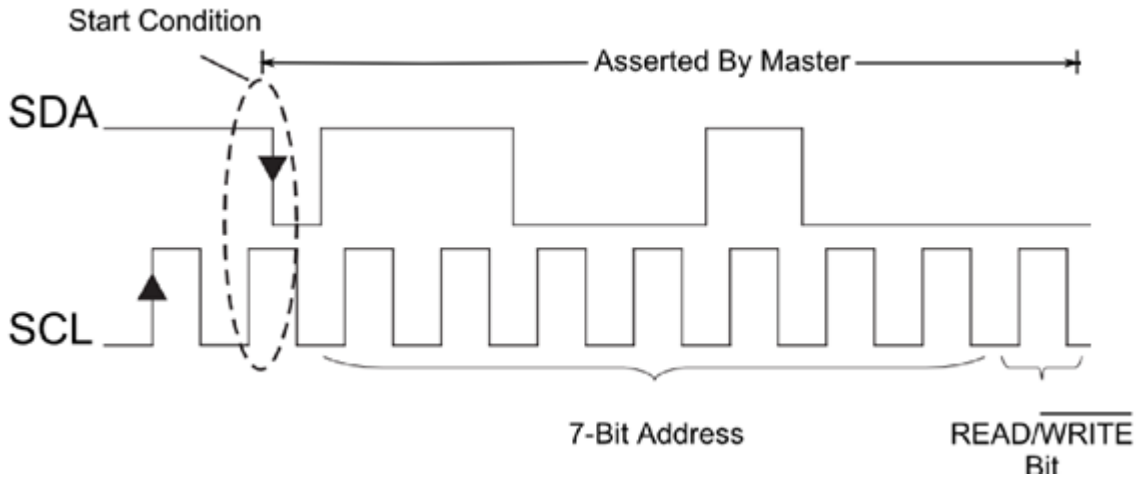
Start and Stop conditions

- پیش از شروع تبادل داده، Master به منظور آگاه‌سازی Slaveها، سیگنال Start Bit را ارسال می‌کند.
- سیگنال Start Bit شامل یک بیت 0 می‌باشد هنگامی که clock یا سیم SCL در وضعیت High یا 1 قرار دارد.
- در پروتکل I2C، تغییر وضعیت پایه SDA تنها در سطوح Low یا 0 از Clock مجاز است ولی برای Start bit و Stop bit، این تغییر هنگامی رخ می‌دهد که سیگنال clock در وضعیت High باشد.
- در انتهای ارسال داده‌ها نیز، سیگنال Stop bit توسط Master ارسال می‌گردد تا Slave مطلوب دیگر منتظر دریافت داده نباشد و وضعیت باس در حالت idle قرار بگیرد.



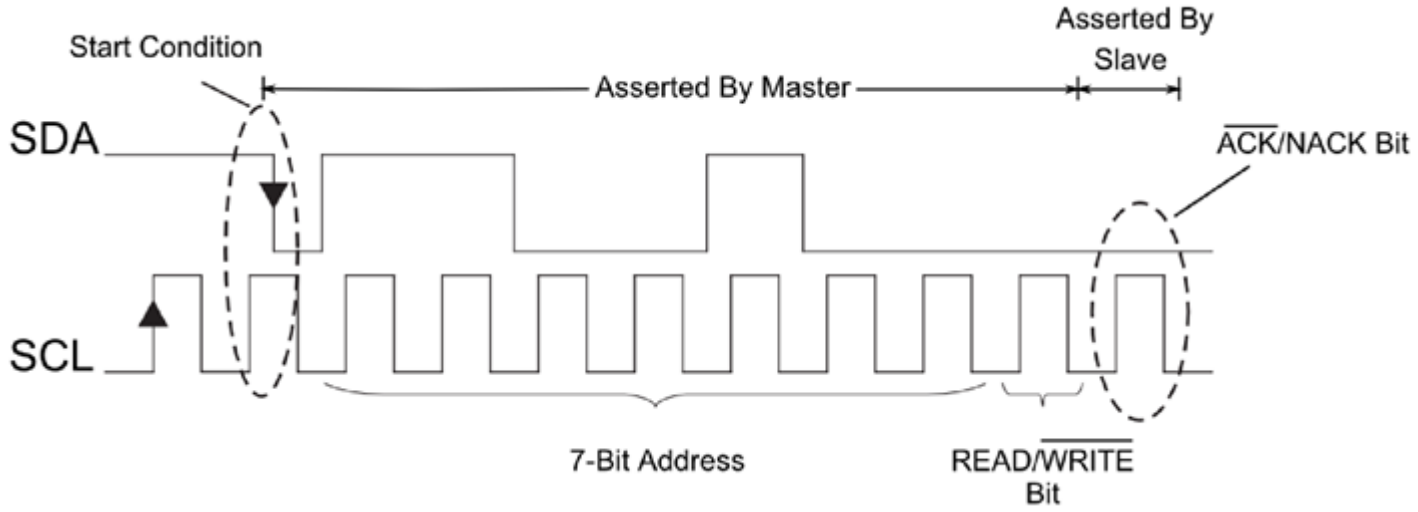
Address Byte

- پس از ارسال Start bit توسط Master، تمامی Slaveهای آماده، منتظر دریافت آدرس می‌مانند.
- Master بایستی آدرس Slave مطلوب خود را بر روی باس قرار دهد (معمولا با ارزش‌ترین بیت ابتدا ارسال می‌گردد)
- در بسته آدرس شامل 1 بایت یا 8 بیت می‌شود (7بیت نخست مربوط به آدرس و بیت آخر مربوط به عملیات خواندن یا نوشتن می‌باشد.)
- با 7 بیت آدرس می‌توان تا 128 Slave مختلف را به باس متصل کرد (به پارامترهای دیگری نیز وابسته است).
- به منظور نوشتن در Slave، بایستی بیت R/W 0 باشد و به منظور خواندن بایستی مقدار آن 1 شود.



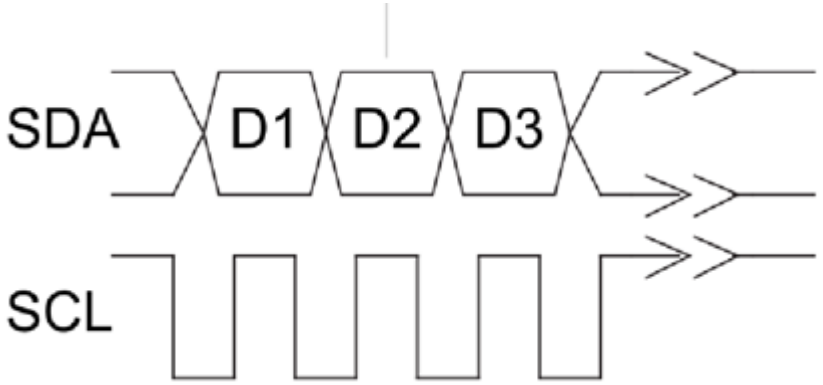
Acknowledge (ACK)/ Not Acknowledge (NACK)

- به منظور اطمینان Master از حضور Slave با آدرس ارسالی، بایستی پاسخی توسط Slave ارسال گردد.
- در هنگام ارسال آدرس تمامی Slaveها در حال گوش کردن به آدرس ارسالی توسط Master می‌باشند (مداری در داخل Slave ها وجود دارد که آدرس ارسالی توسط Master را با آدرس خودشان مقایسه می‌کند.)
- در هر قسمت از مقایسه آدرس، اگر عدم تطبیق مشخص شود مدار ادامه مقایسه را متوقف می‌کند (بنابراین هرگز سیگنالی بر روی باس قرار نمی‌دهد.)
- در صورت تطابق آدرس با یکی از Slaveها، اگر آن Slave در حالت آماده بکار باشد، سیگنال ACK = 0 را ارسال می‌کند (اصطلاحا، باس را پایین می‌کشد (۱ بیت)).



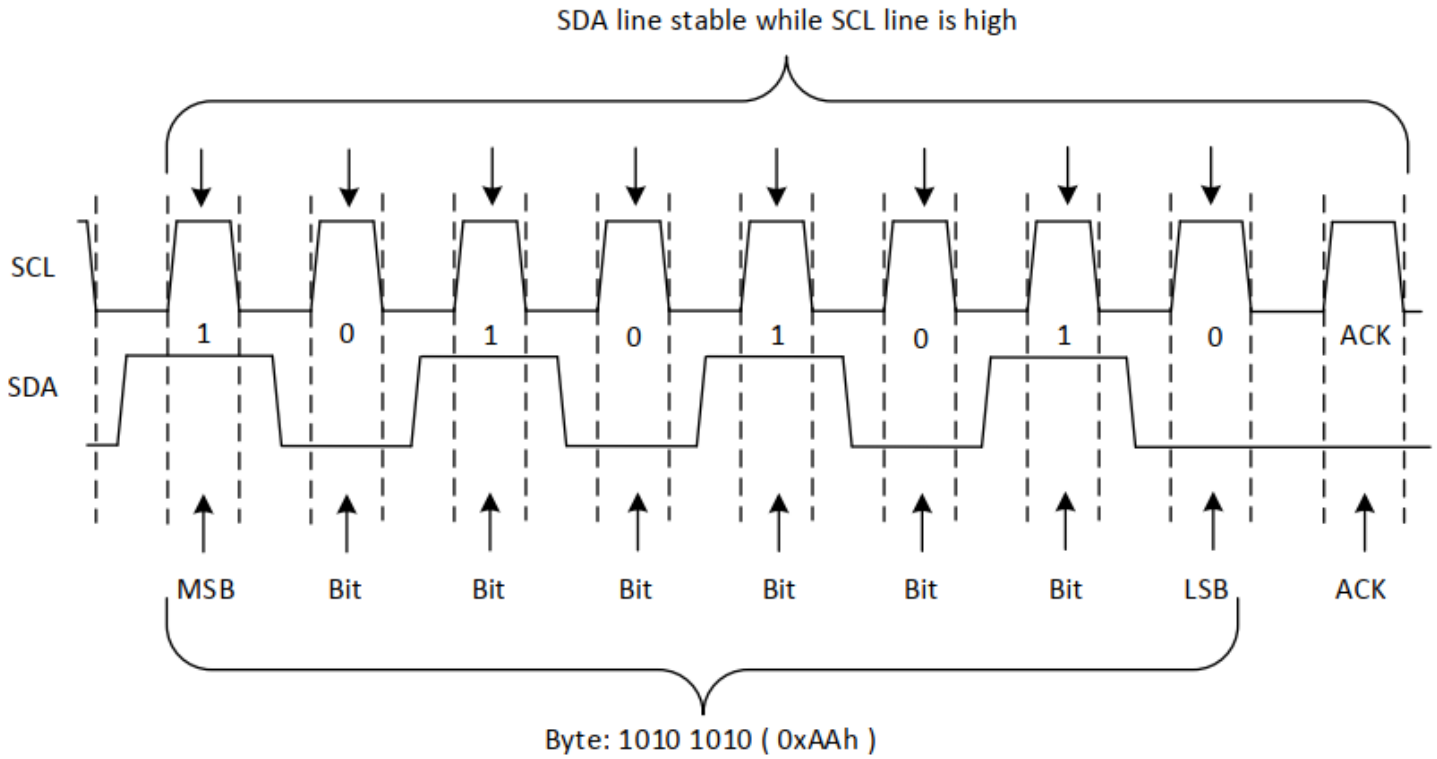
Data Byte

- در صورت دریافت ACK از Slave دو سناریو وجود دارد :
  - Master سیگنال RW = 0 را ارسال کرده است (Master قصد نوشتن داده دارد، بنابراین داده‌ها با فرمت بایت و با شروع از باارزش‌ترین بیت، ارسال می‌گردد)
  - Master سیگنال RW = 1 را ارسال کرده است (Master قصد خواندن داده دارد، بنابراین داده‌ها با فرمت بایت و با شروع از باارزش‌ترین بیت، توسط Slave انجام می‌شود)
- توجه شود که در پروسه خواندن داده، هنگامی که Slave آدرس خود را تشخیص می‌دهد، بیت ACK را ارسال و سپس کنترل باس SDA را بر عهده می‌گیرد، پس از ارسال هر 8 بیت، یک بین ACK یا NACK، توسط Master ارسال می‌گردد. مادامی که Master قصد دارد داده دریافت نماید، بیت Ack و پس از رسیدن به آخرین بایت مطلوب، مقدار NACK را ارسال می‌کند و کنترل باس SDA را برعهده گرفته و Stop Bit را ارسال می‌کند.



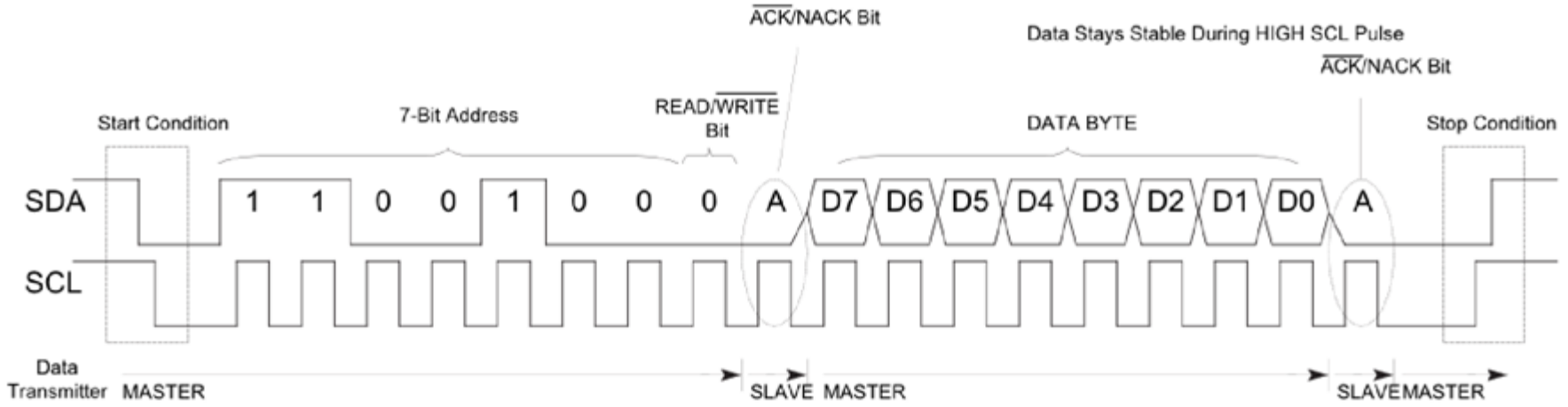
Data Byte

- هنگامی که Slave کنترل باس SDA را برعهده می‌گیرد بازهم Master است که سیگنال Clock را ارائه می‌دهد.
- نمونه برداری در میانه سطح بالای SCL انجام می‌شود (سیگنال SDA تنها هنگامی SCL در وضعیت Low قرار دارد می‌تواند تغییر کند، بنابراین بهترین زمان برای نمونه برداری، میانه سطح High در SCL می‌باشد).



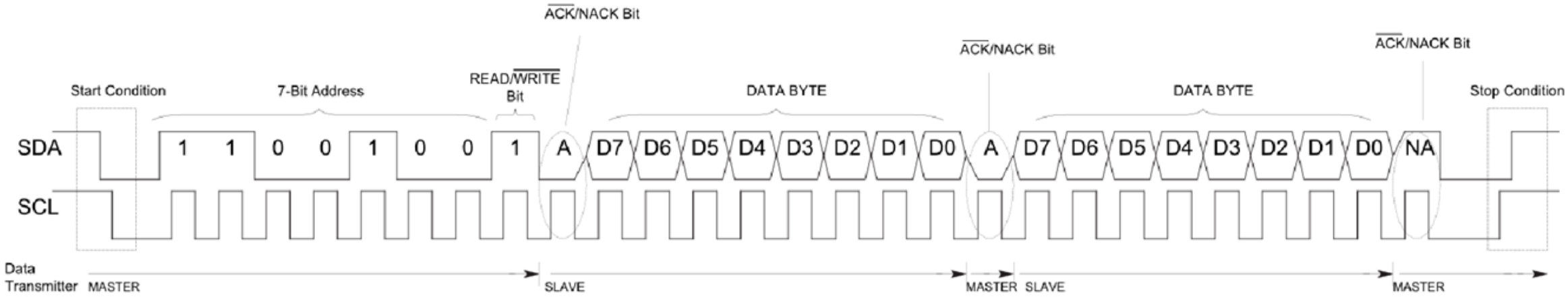
Write to a device

- در هنگام نوشتن در یک Device، بایستی پس از استارت بیت و آدرس، بیت 0 به عنوان RW ارسال گردد.
- در گام بعد، Slave اگر آماده باشد، بیت ACK را ارسال کرده و منتظر دریافت دیتا می‌شود.
- در گام بعد، Master پس از ارسال هر بسته 8 بیتی دیتا، منتظر دریافت ACK/NACK از سوی Slave می‌ماند.
- با دریافت بیت ACK، به ارسال دیتا ادامه می‌دهد و با دریافت بیت NACK می‌تواند مجدد برای ارسال داده تلاش نماید و یا به عملیات پایان دهد.
- به منظور پایان دادن به تراکنش با Slave، بایستی استاپ بیت ارسال گردد، در غیر اینصورت هر دیتایی بر روی باس قرار گیرید، توسط Slave پیشین پاسخ داده خواهد شد



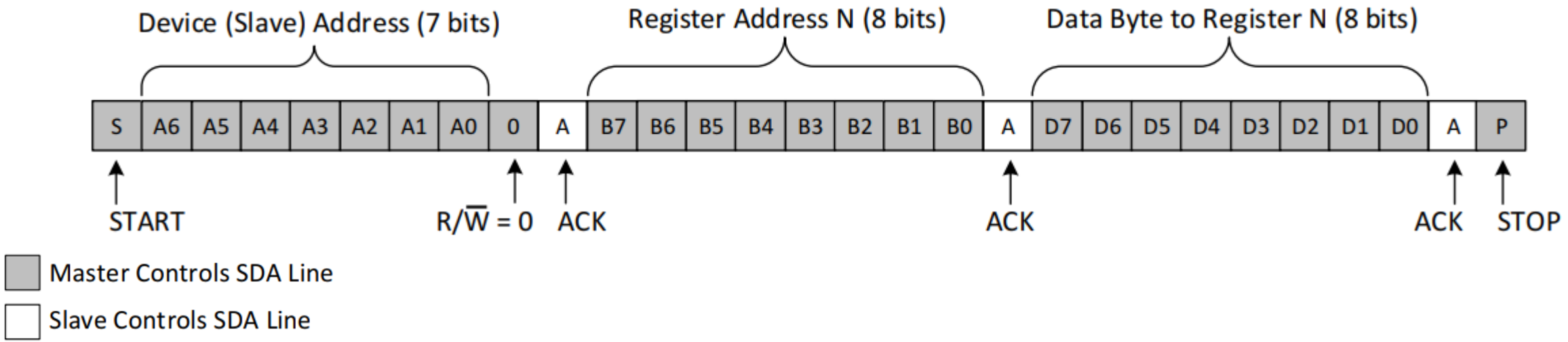
Read from a device

- در هنگام خواندن از یک Device، بایستی پس از استارت بیت و آدرس، بیت 1 به عنوان RW ارسال گردد.
- در گام بعد، Slave اگر آماده باشد، بیت ACK را ارسال کرده و کنترل باس SDA را به عهده می‌گیرد و شروع به ارسال داده می‌کند.
- در گام بعد، Slave پس از هر بسته 8 بیتی دیتا که ارسال می‌کند، منتظر بیت ACK/NACK از Master می‌ماند.
- با دریافت بیت ACK، به ارسال دیتا ادامه می‌دهد و با دریافت بیت NACK به آن پایان می‌دهد و کنترل باس به Master می‌دهد.



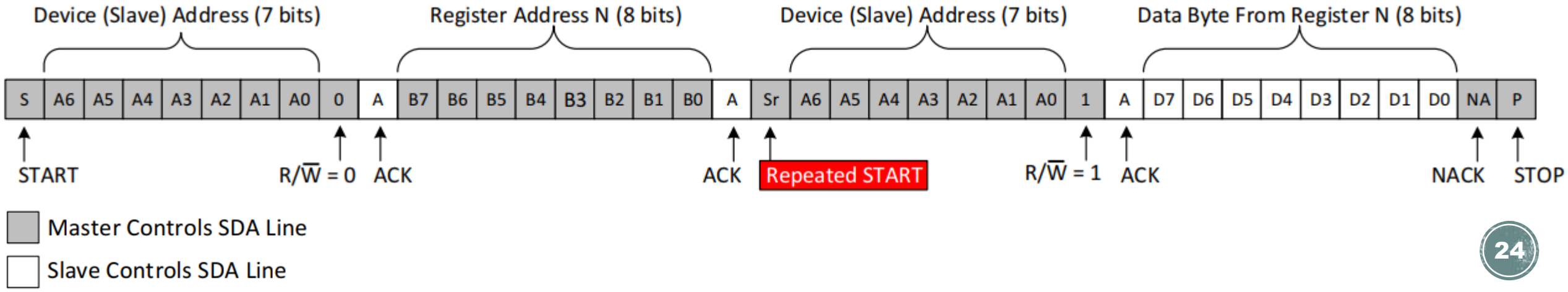
Write to a device’s register

- در صورتی که Slave خود دارای رجیسترهایی باشد، نظیر تراشه‌های حافظه، علاوه بر آدرس خود تراشه در باس I2C، بایستی آدرس رجیستری از آن را که قصد داریم در آن دیتایی بنویسیم نیز ارسال نماییم.
- در این راستا، پس از ارسال استارت بیت، آدرس Slave به همراه بیت R/W = 0 را ارسال می‌کنیم. پس از دریافت بیت ACK از سوی Slave، آدرس رجیستری از Slave که قصد داریم در آن دیتا ذخیره نماییم را ارسال می‌کنیم. در گام آخر نیز پس از دریافت ACK، از سوی Slave، دیتا را ارسال می‌کنیم. در انتها و به منظور پایان تراکنش، استاپ بیت ارسال می‌گردد.
- سوال : در صورتی که بخواهیم مجموعه‌ای داده ارسال نماییم، بایستی آدرس را پیش از هر بار دیتا ارسال نماییم!؟



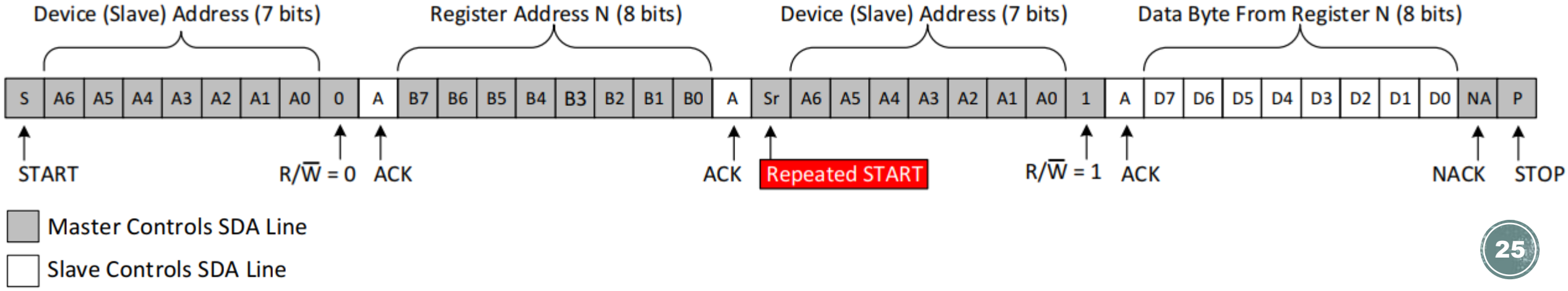
Read from a device's register

- در صورتی که Slave خود دارای رجیسترهایی باشد، نظیر تراشه‌های حافظه، علاوه بر آدرس خود تراشه در باس I2C، بایستی آدرس رجیستری از آن را که قصد داریم از آن دیتایی بخوانیم را نیز ارسال نماییم.
- در این راستا، پس از ارسال استارت بیت، آدرس Slave به همراه بیت R/W = 0 را ارسال می‌کنیم. پس از دریافت بیت ACK از سوی Slave، آدرس رجیستری از Slave که قصد داریم در از آن دیتا بخوانیم را ارسال می‌کنیم. در گام بعد بایستی مجدد آدرس Slave را اینبار به همراه بیت R/W=1 ارسال نماییم تا Slave متوجه شود که قصد خواندن اطلاعات را داریم، بنابراین بایستی مجدد یک استارت بیت دیگر ارسال نماییم سپس آدرس و R/W=1 را ارسال کنیم. در این مرحله پس از ارسال ACK توسط Slave، کندل باس را به دست می‌گیرد و شروع به ارسال داده می‌کند تا بیت NACK را در انتهای یک بسته توسط Master دریافت نماید.
- در انتها و به منظور پایان تراکنش، استاپ بیت ارسال می‌گردد.



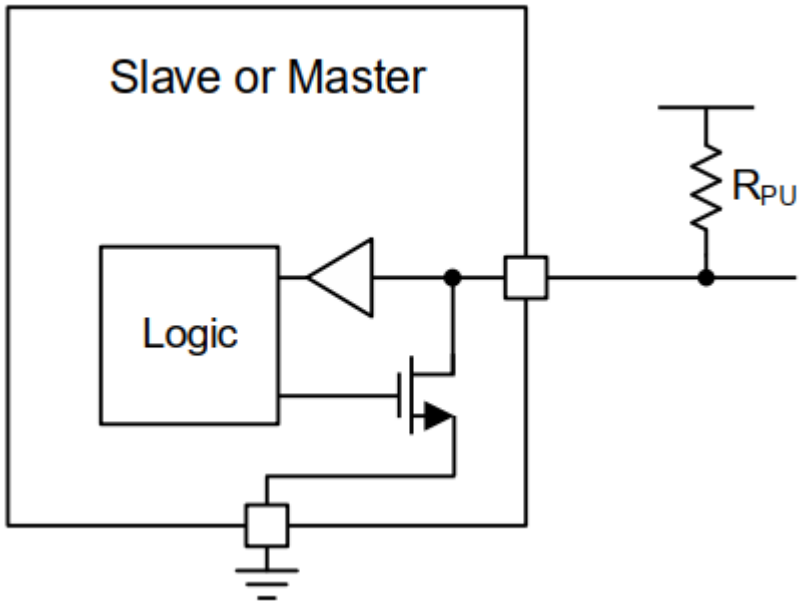
Read from a device's register

- در صورتی که Slave خود دارای رجیسترهایی باشد، نظیر تراشه‌های حافظه، علاوه بر آدرس خود تراشه در باس I2C، بایستی آدرس رجیستری از آن را که قصد داریم از آن دیتایی بخوانیم را نیز ارسال نماییم.
- در این راستا، پس از ارسال استارت بیت، آدرس Slave به همراه بیت R/W = 0 را ارسال می‌کنیم. پس از دریافت بیت ACK از سوی Slave، آدرس رجیستری از Slave که قصد داریم در از آن دیتا بخوانیم را ارسال می‌کنیم. در گام بعد بایستی مجدد آدرس Slave را اینبار به همراه بیت R/W=1 ارسال نماییم تا Slave متوجه شود که قصد خواندن اطلاعات را داریم، بنابراین بایستی مجدد یک استارت بیت دیگر ارسال نماییم سپس آدرس و R/W=1 را ارسال کنیم. در این مرحله پس از ارسال ACK توسط Slave، کندل باس را به دست می‌گیرد و شروع به ارسال داده می‌کند تا بیت NACK را در انتهای یک بسته توسط Master دریافت نماید.
- در انتها و به منظور پایان تراکنش، استاپ بیت ارسال می‌گردد.

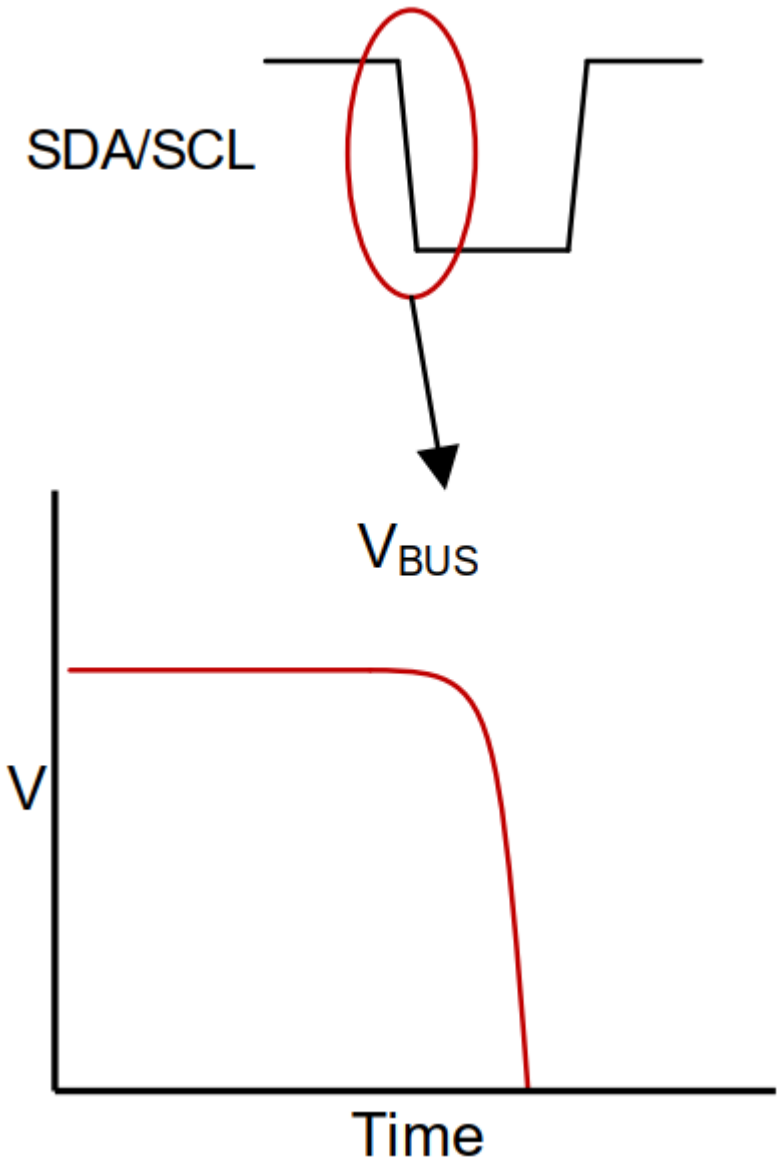


Structure of SDA/SCL Line

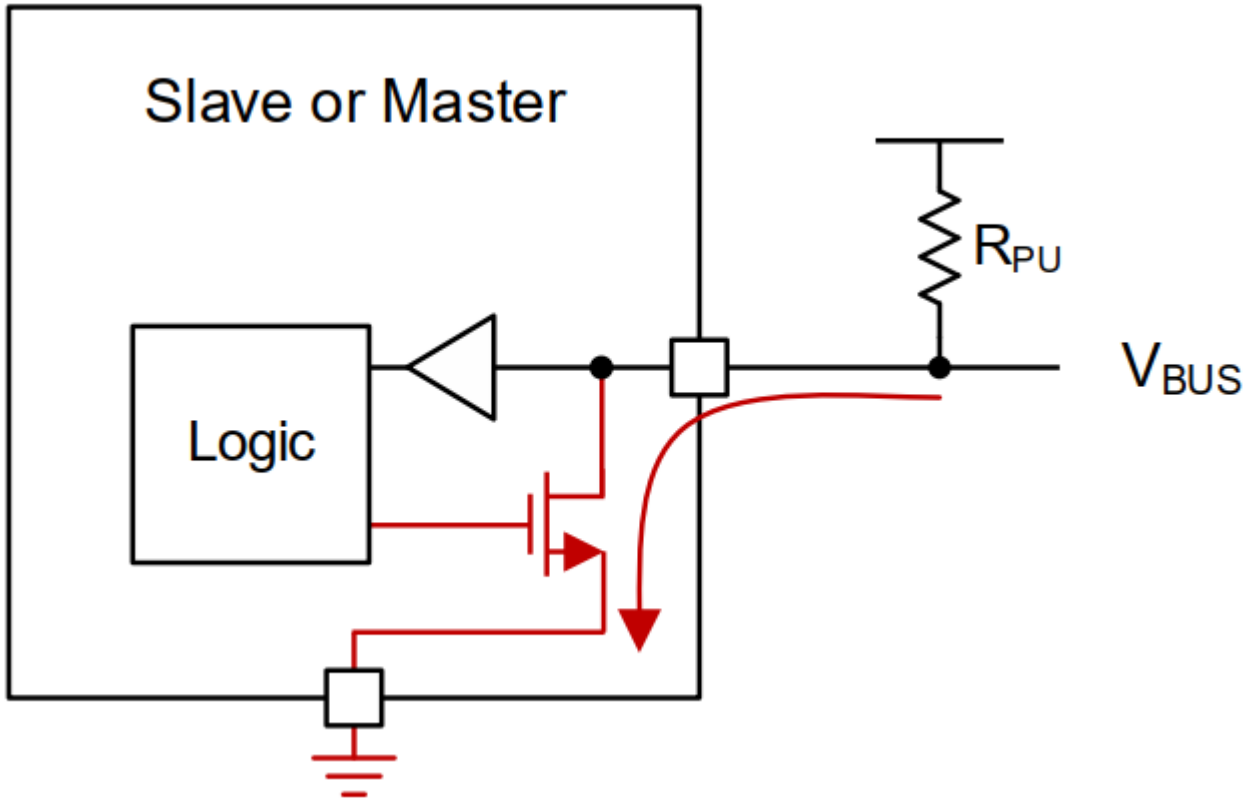
- باتوجه به اینکه در ساختار I2C هر دو پایه SCL و SDA به صورت ارتباط دوطرفه می‌باشند، بایستی معماری آن‌های به صورت Open Drain/Collector باشد.
- هنگامی که Slave در حالت غیرفعال می‌باشد، مدار Logic همواره ترانزیستور را خاموش نگه می‌دارد که سبب می‌شود تا سیگنال ارسالی توسط Slave، همواره NACK باشد.



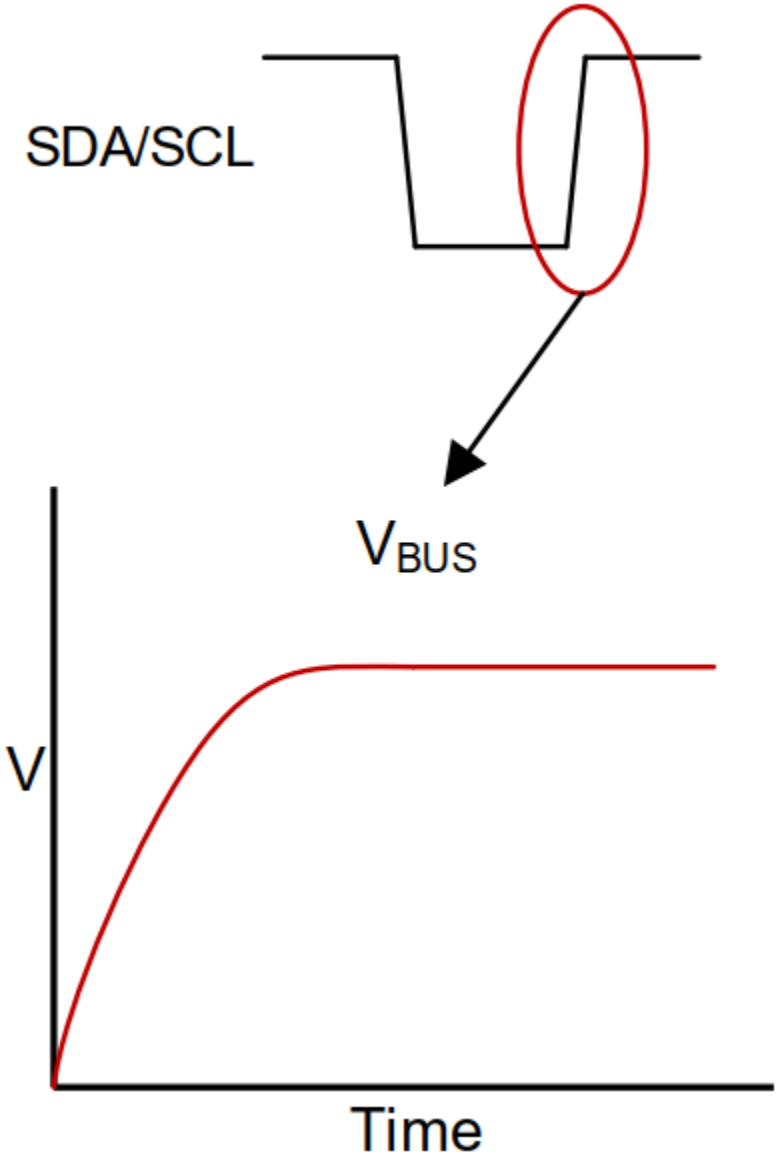
Open Drain Pulling Low



$$\tau = RC = Ron \times Cparasitic$$

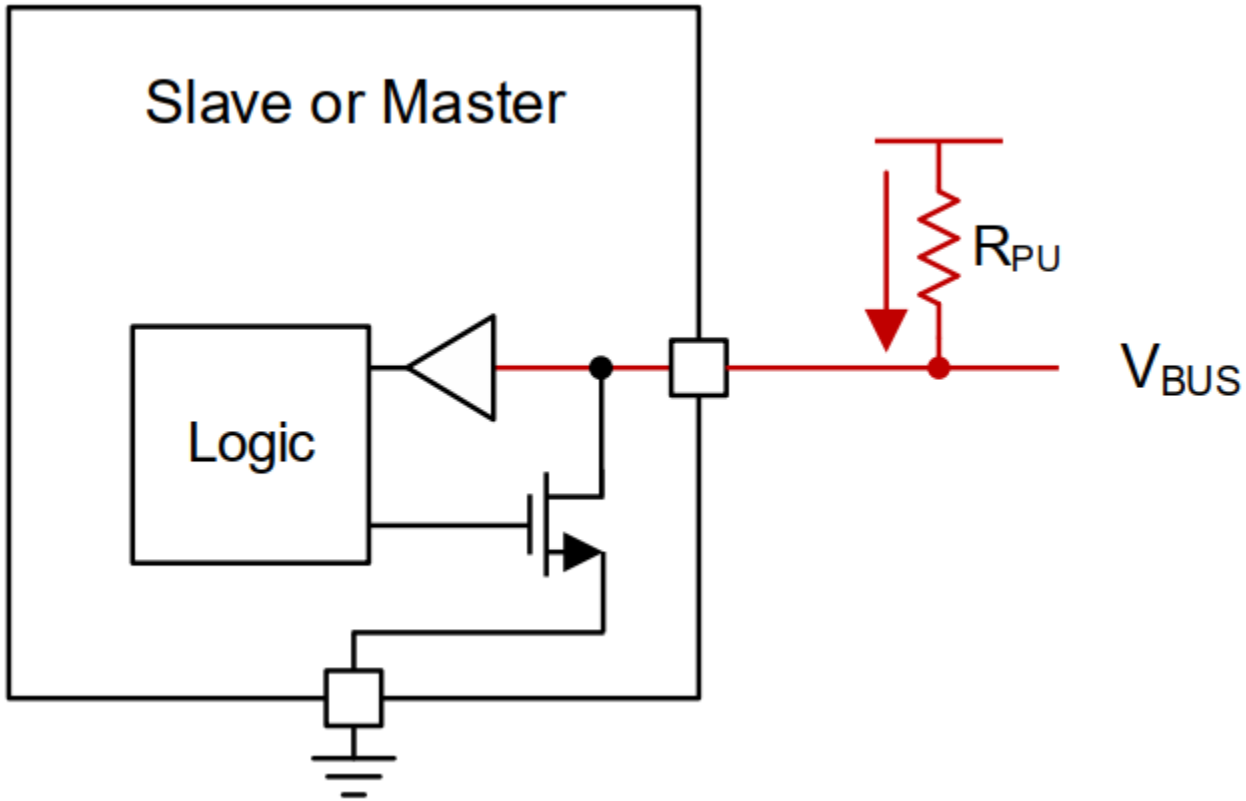


Open Drain Releasing Bus



$$\tau = RC = Rpu \times Cparasitic$$

$$Rpu \gg Ron \rightarrow Trise \gg Tfall$$



Open Drain Releasing Bus

- حداکثر تعداد Slaveهای موجود در یک باس I2C باتوجه به 7 بیت فضای آدرس‌دهی، می‌تواند 128 باشد.
- هر Slave یا Master دارای خازن‌های پارازیتیک می‌باشد، بنابراین با افزودن آن‌ها به باس، ظرفیت خازنی کلی باس افزایش می‌یابد، از طرفی بایستی Trise کوچکتر از حداکثر سرعت باس باشد، بنابراین یکی از عوامل محدود کننده سرعت باس، خازن پارازیتیک آن می‌باشد که نبایستی از 400 pF بیشتر شود (چرا!؟)
- باس I2C دارای دو سرعت استاندارد می‌باشد :
  - Standard Mode (SM) : 100 Kbit/s
  - Fast Mode (FM) : 400 Kbit/s
- مقاومت Pull-up معمولا بین 1k تا 10k انتخاب می‌شود. (مصالحه توان و سرعت).
- سرعت‌های بالاتر امکان پذیر است!؟

Multi Master and Clock Stretching

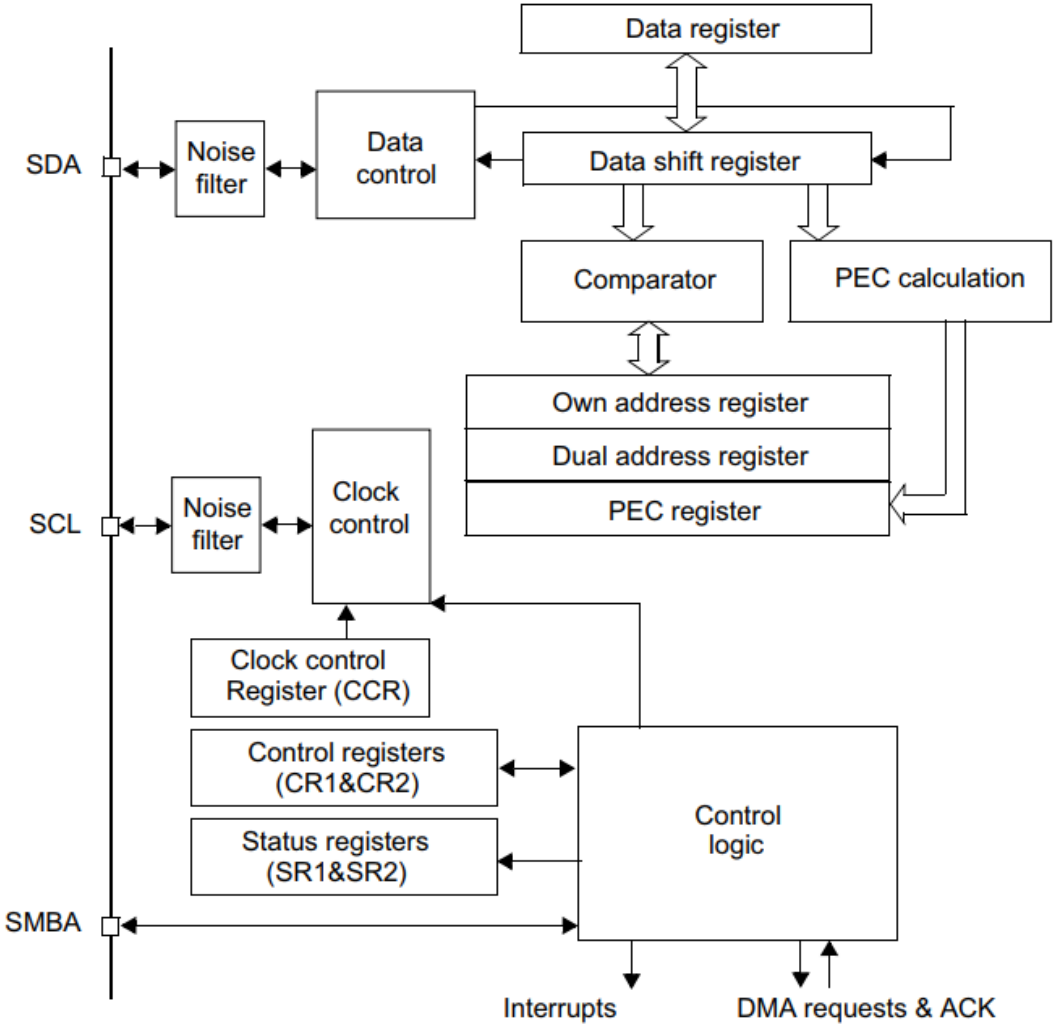
- هنگامی که چند Master در باس وجود دارند، سیگنال هر کدام زودتر سیگنال Start Bit را ارسال نمایند، کنترل باس را به عهده می‌گیرند
  - سوال : اگر هر دو هر زمان این سیگنال را ارسال نمایند چه می‌شود!؟

- در برخی کاربردها قابلیت Clock Stretching توسط Slave وجود دارد، به این معنا که Slave می‌تواند سیگنال SCL را Low نگه دارد تا شرایط لازم برای از سرگیری تبادل داده را بدست آورد، سپس SCL را رها می‌کند.

Advantage & Disadvantage

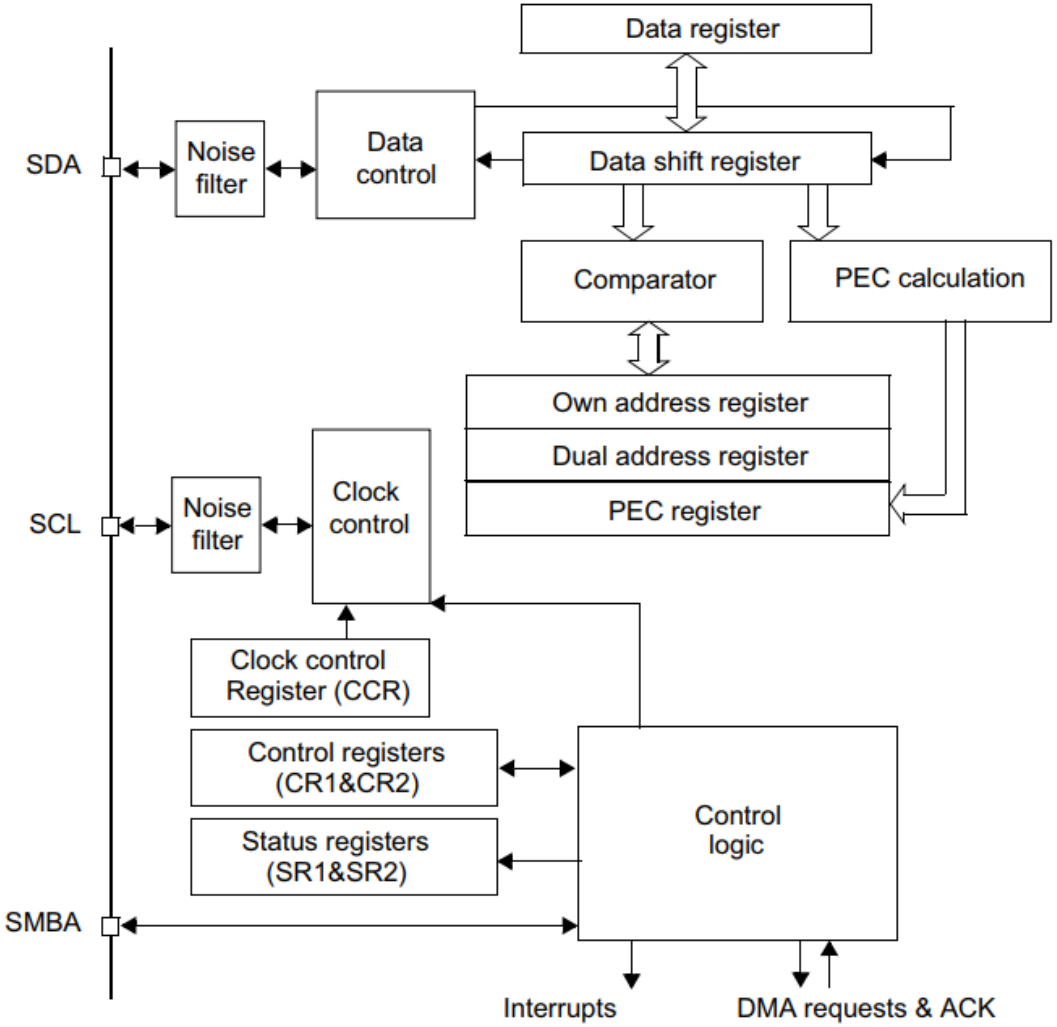
- Disadvantages
  - مشکل در آدرس مشترک میان Slaveها
  - در صورت پشتیبانی از Multi Master و Clock Stretching، کنترل باس کمی پیچیده می‌شود.
  - افزایش بیت‌های اضافه نظیر استارت، استاپ، ACK/NACK و آدرس که منجر به کاهش Throughput می‌گردد.
  - محدودیت‌های سخت‌افزاری نظیر ظرفیت خازنی باس
  - باس اشتراکی که با خراب شدن یکی از Slaveها ممکن است کل باس از کار بیفتد.
- Advantages
  - استفاده از دوسیم و ارتباط سنکرون
  - قابلیت Multi Master
  - خطایابی بالا باتوجه به وجود بیت‌های ACK/NACK

STM32F10x I2C Block Diagram



- Packet Error Checking
- SMBA

STM32F10x I2C Block Diagram



- Packet Error Checking
- SMBA

توابع HAL کاربردی در I2C

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)

توابع HAL کاربردی در I2C

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)

توابع HAL کاربردی در I2C

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)