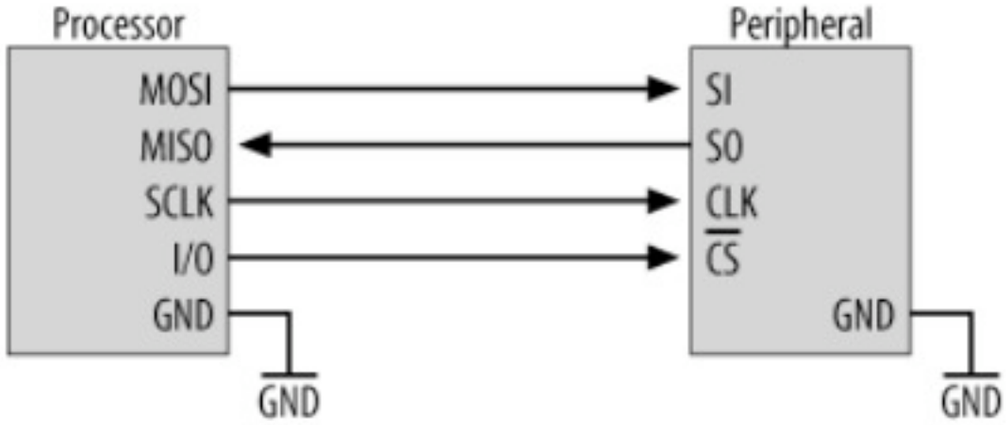
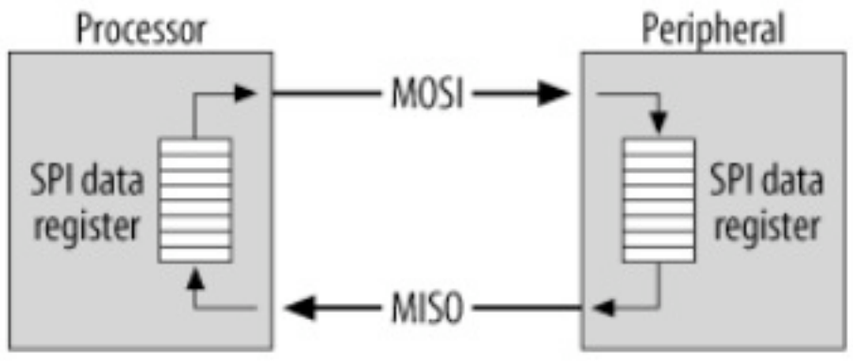
SPI

Serial Peripheral Interface (SPI)

- Invented by Motorola
- Uses four signals
  - Master-out Slave-in (MOSI)
  - Master-in Slave-out (MISO)
  - Serial Clock (SCK)
  - Slave Select (/SS), or Chip Select (/CS)
- See also: Chapter 7 of Catsoulis's book

Basic SPI Connection



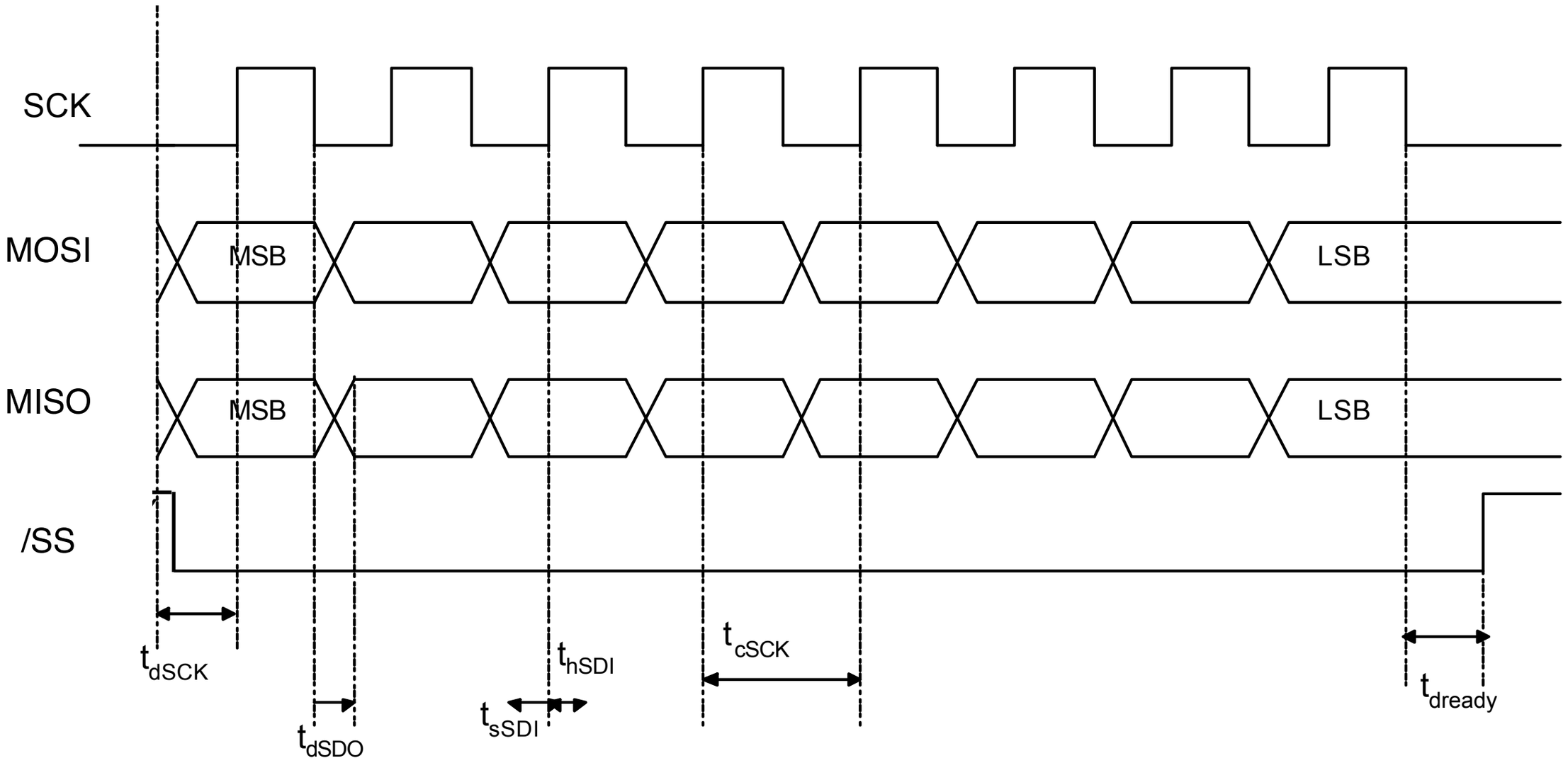


- MOSI, aka SI (slave in) or SDI (slave data in)
- MISO, aka SO or SDO
- Serial Clock: named SCLK or just SCK
- Chip select /CS or Slave Select /SS
- Swap byte

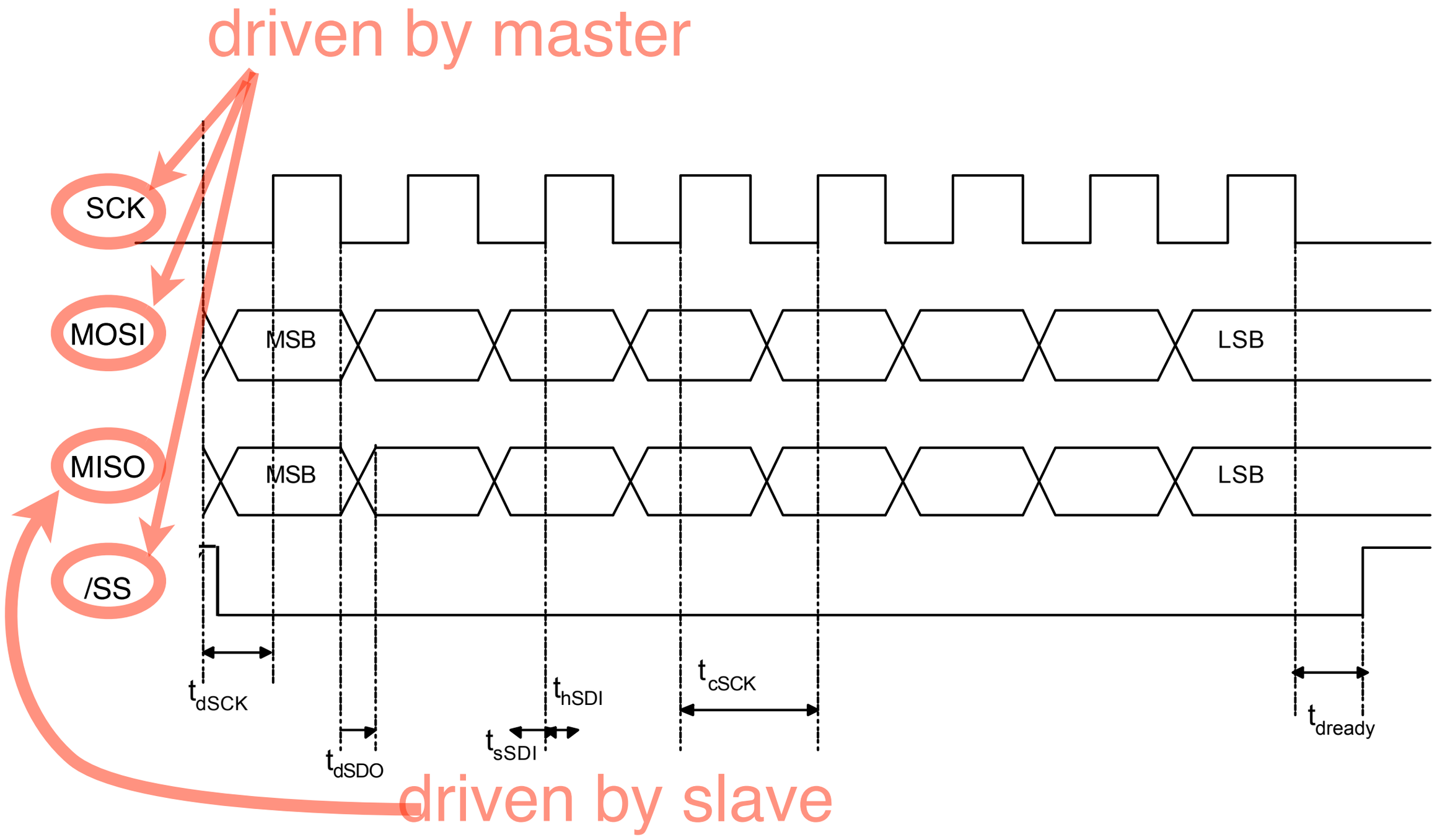
Master-Slave protocol

- Master
  - Asserts Slave Select (/SS)
  - Drives the clock (SCK)
- Data
  - Master and slave swap data, 1 bit per SCK
  - Full duplex

SPI timing example



SPI timing example



Issues

- Synchrony
- Polarity, Phase
- Slave notification
- Application-specific protocol
- Bus topologies
- MCU integration vs. software controller

Synchrony

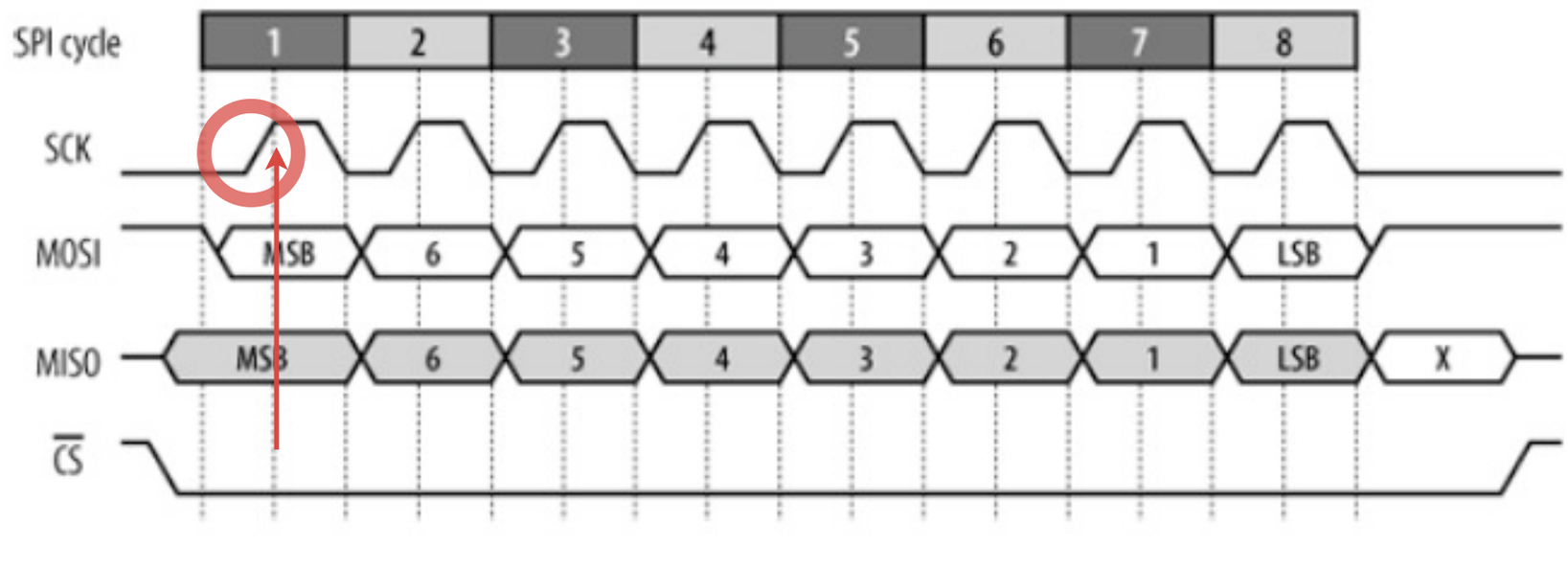
- Is it synchronous or asynchronous?
  - Synchronous to the master's SCK
  - But SCK doesn't mean fixed clock rate!
- Master can drive SCK as slowly as it wants
  - 0Hz to 1-2 MHz; could be 8MHz-32MHz
  - Slave is responsible for keeping up; but Master should be aware of slowest slave

SPI polarity and phase

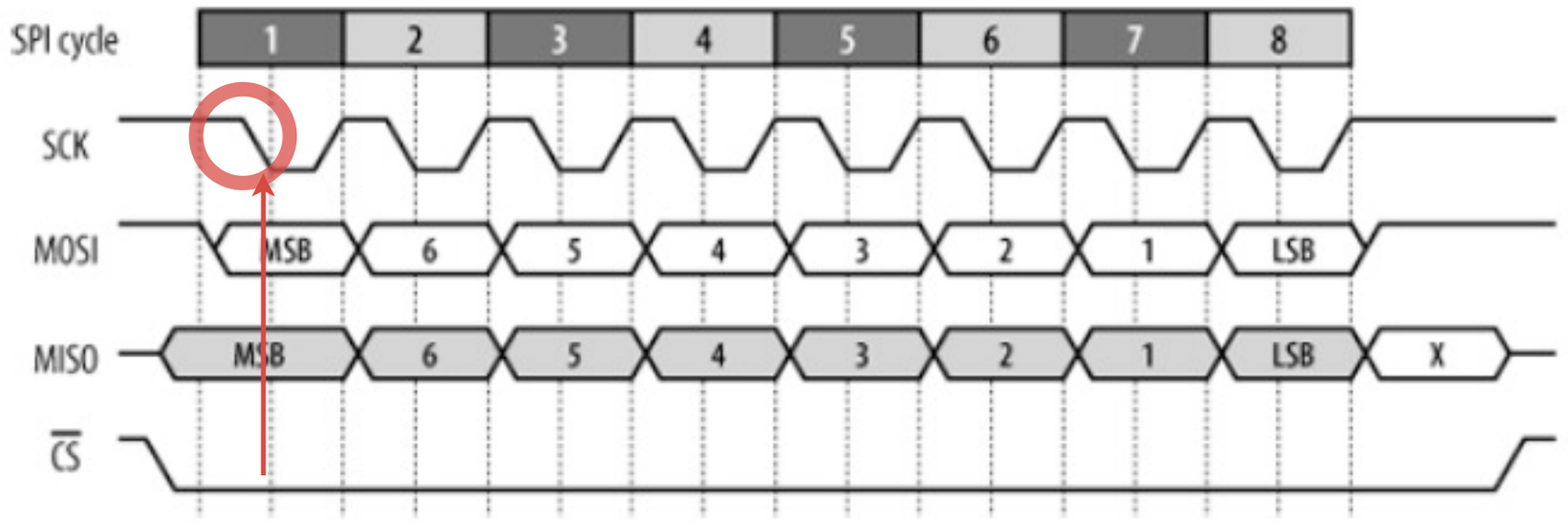
- SCK **Polarity**
  - Low when idle, data valid on *rising edge*
  - High when idle, data valid on *falling edge*
- SCK **Phase**
  - Phase zero: first edge after /CS asserted
  - Phase one: second edge after /CS asserted

Phase-zero SCK

low polarity, rising edge

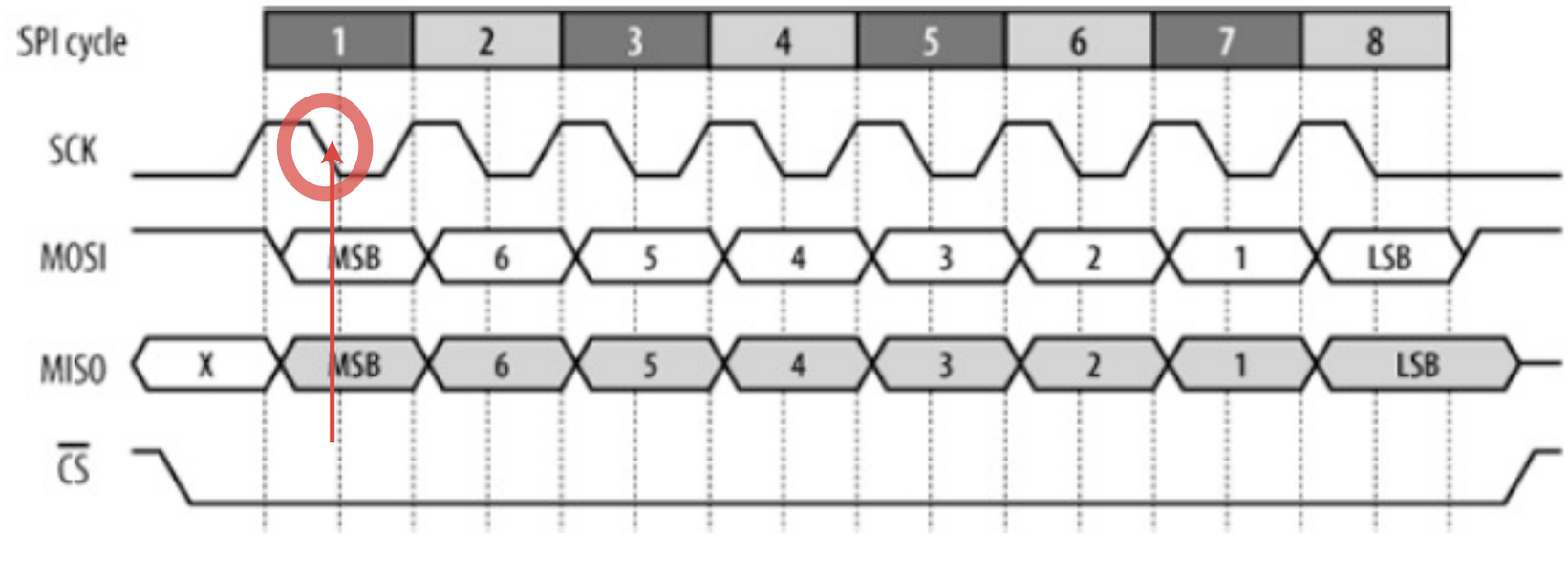


high polarity, falling edge

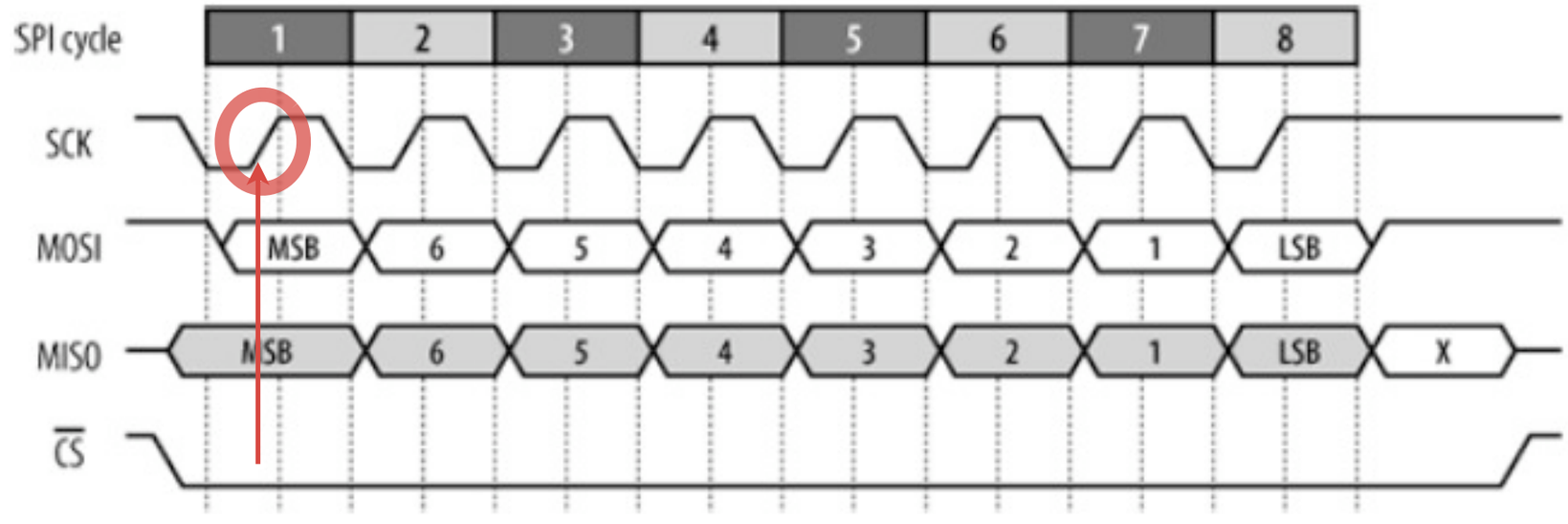


Phase-one SCK

low polarity, after rising edge



high polarity, after falling edge



Slave notification

- What if Slave has data to send to Master?
  - Slave cannot initiate an SPI transaction
    => Master has to keep polling
- Solution:
  - Extra "data-ready" signal (IRQ) from slave to an interrupt line on MCU
  - Not part of SPI; device-specific

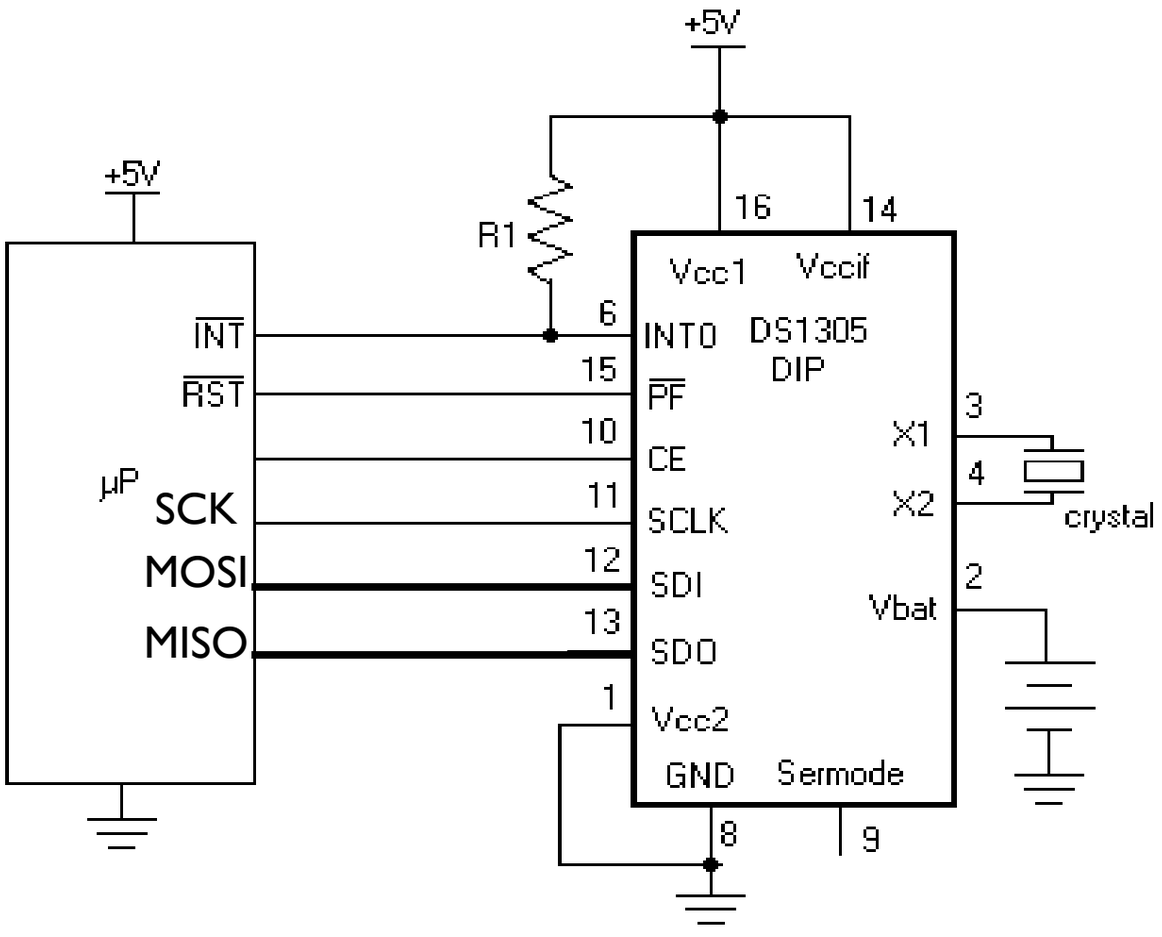
How is SPI used?

- SPI itself provides a "transport" mechanism
- Slave device defines own formats. Typically,
  - Master asserts /SS
  - Byte 0: Master sends command or register address; Slave sends padded byte
  - Byte 1...: data to/from slave
  - Master deasserts /SS

Example: SPI-based clock & calendar chip

- Maxim DS 1305 (DS => Dallas Semi)
- Keeps track of
  - year up to 2100, month, day, hour, min, sec, ms
  - alarm => generates interrupt
  - voltage monitor
- http://datasheets.maxim-ic.com/en/ds/DS1305.pdf

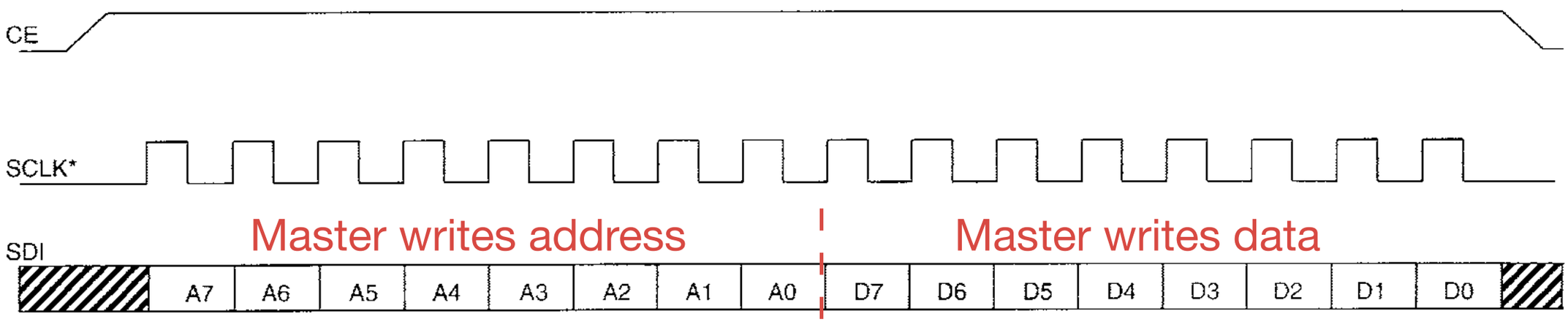
SPI Connection for Alarm Clock chip



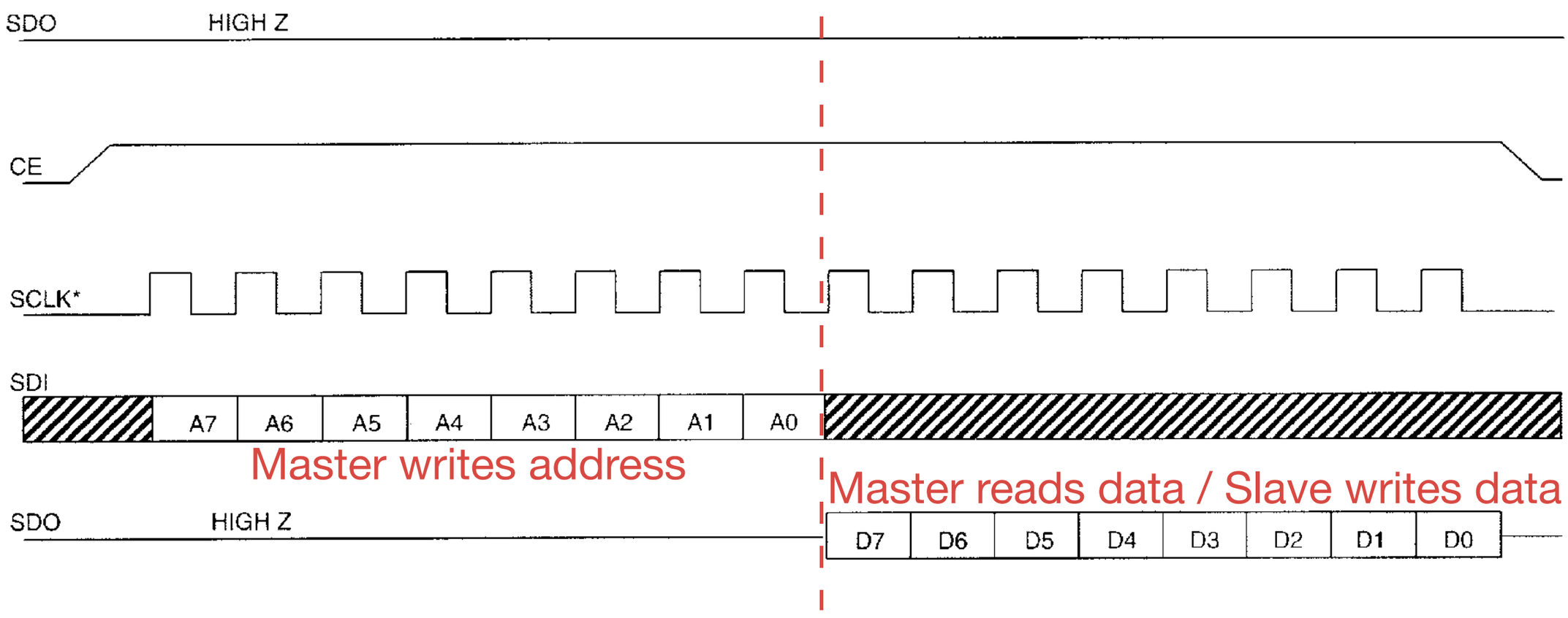
- SerMode
  - 1: SPI mode
  - 0: 3-wire mode
- Interrupt

DS1305 SPI single-byte transaction

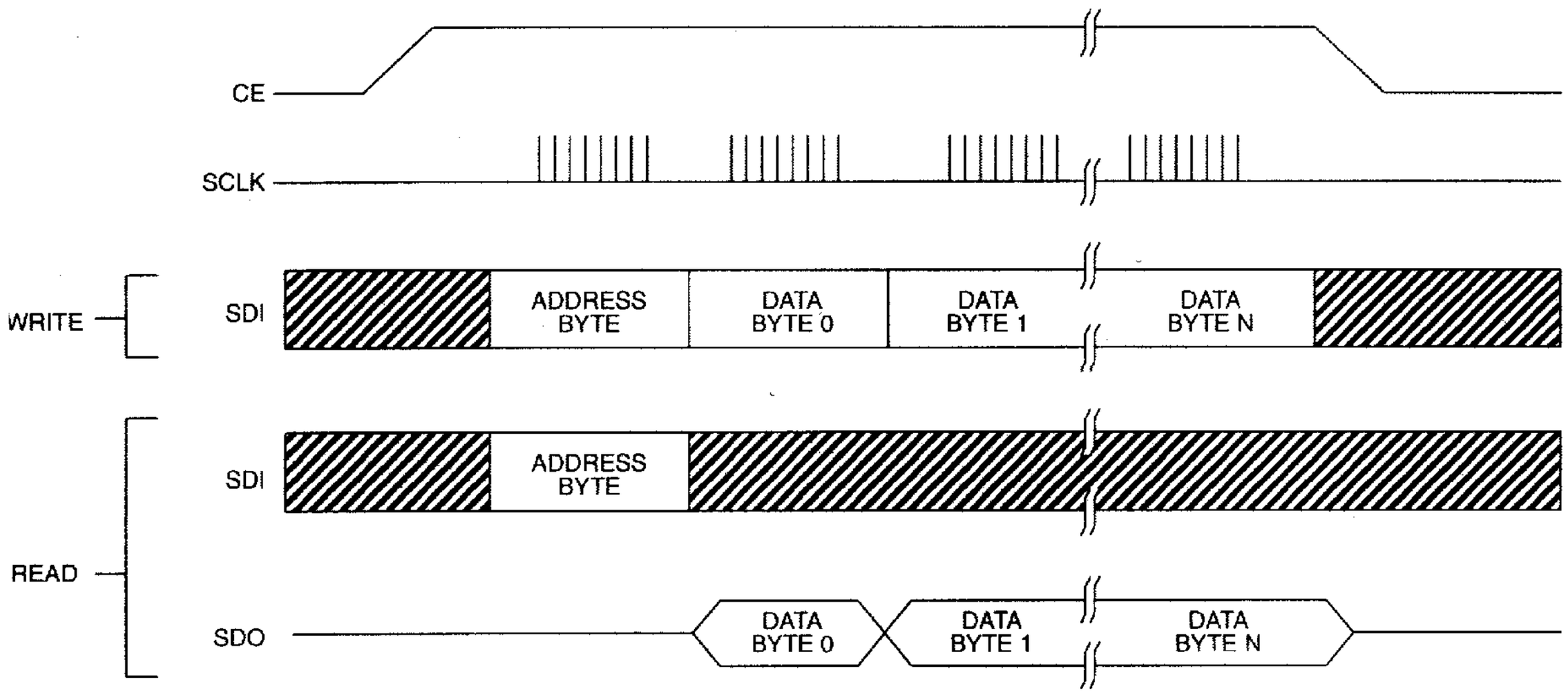
Write



Read



DS 1305 SPI multi-byte transaction (keep CE high)



DS 1305 registers

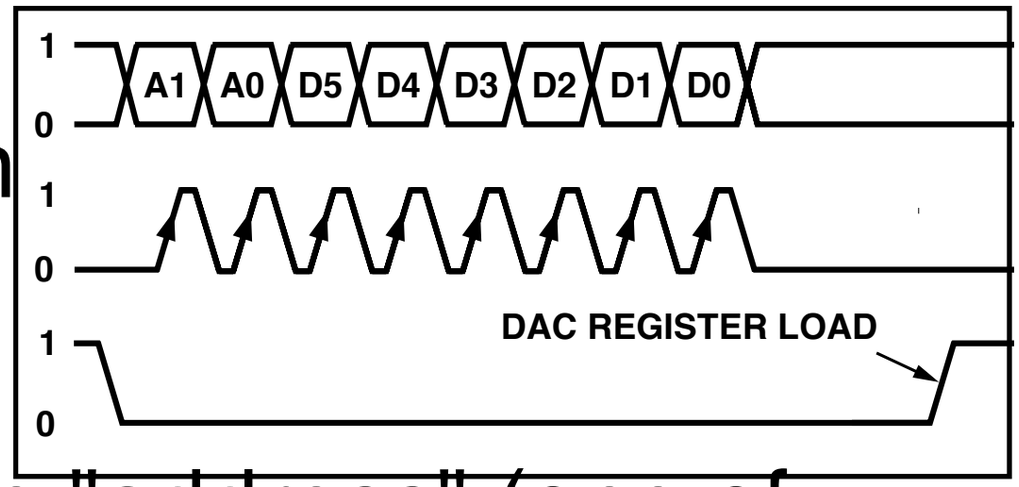
HEX ADDRESS		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	RANGE
READ	WRITE									
00H	80H	0	10 Seconds			Seconds				00–59
01H	81H	0	10 Minutes			Minutes				00–59
02H	82H	0	12	P	10 Hour	Hours				01–12 + P/A
				A						00–23
			24	10						
03H	83H	0	0	0	0	Day				1–7
04H	84H	0	0	10 Date		Date				1–31
05H	85H	0	0	10 Month		Month				01–12
06H	86H	10 Year				Year				00–99
—	—	Alarm 0								—
07H	87H	M	10 Seconds Alarm			Seconds Alarm				00–59
08H	88H	M	10 Minutes Alarm			Minutes Alarm				00–59
09H	89H	M	12	P	10 Hour	Hour Alarm				01–12 + P/A
				A						00–23
			24	10						
0AH	8AH	M	0	0	0	Day Alarm				01–07
—	—	Alarm 1								—
0BH	8BH	M	10 Seconds Alarm			Seconds Alarm				00–59
0CH	8CH	M	10 Minutes Alarm			Minutes Alarm				00–59
0DH	8DH	M	12	P	10 Hour	Hour Alarm				01–12 + P/A
				A						00–23
			24	10						
0EH	8EH	M	0	0	0	Day Alarm				01–07
0FH	8FH	Control Register								—
10H	90H	Status Register								—
11H	91H	Trickle Charger Register								—
12–1FH	92–9FH	Reserved								—
20–7FH	A0–FFH	96 Bytes User RAM								00–FF

Single vs Burst mode in DS 1305

- Burst => just continue reading more bytes
  - hardware automatically incr. address by 1
  - wraps around if accessing user RAM
- Example: start reading at address 0
  - [10s, s], [10m, m], [10h, h], [day], ...
  - can read all or part in burst, or read any single component of the date/time record.

Another SPI example: AD5203

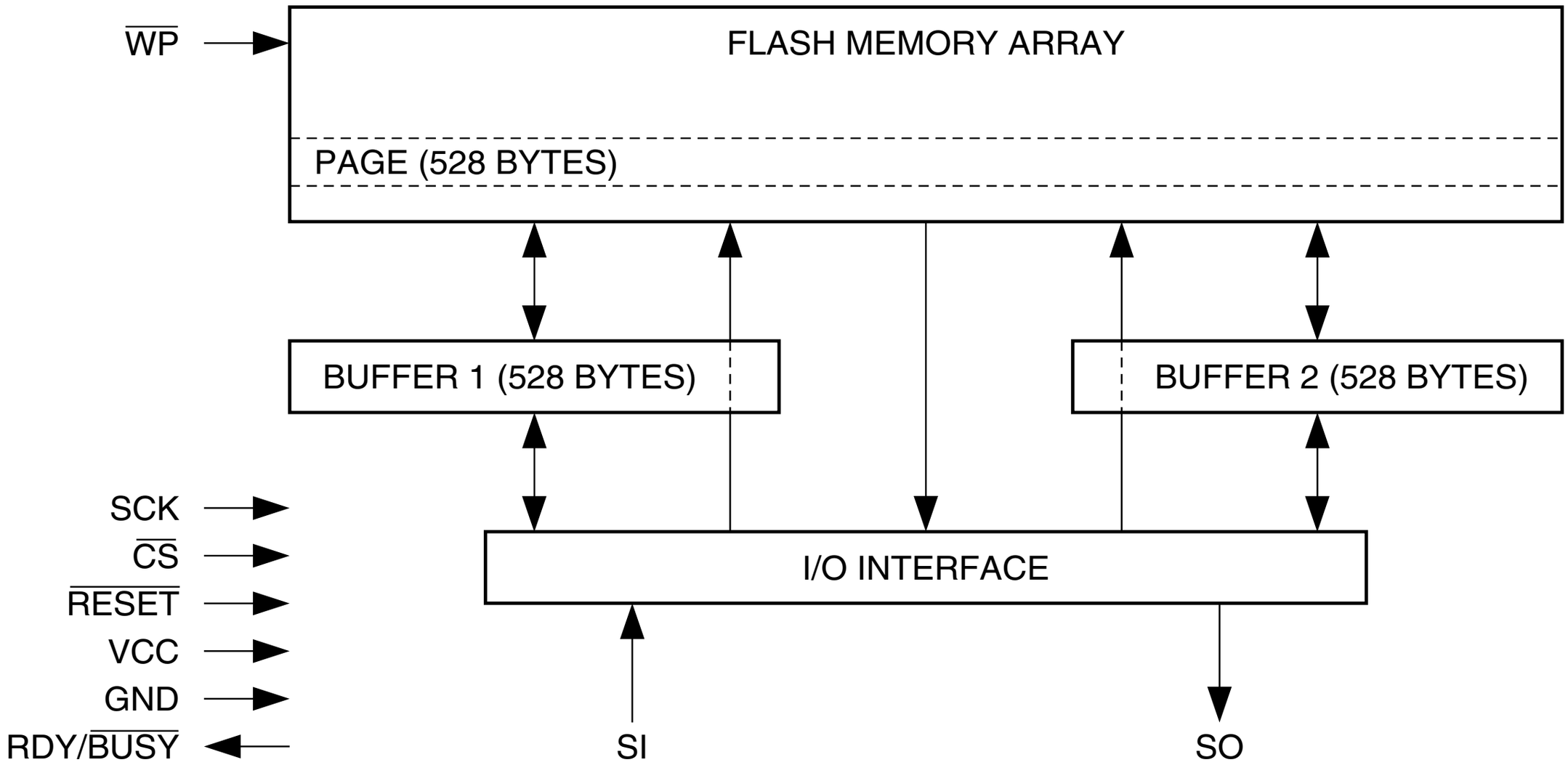
- Four digital potentiometers in one
  - Selected by two bits
  - Set in 6-bit resolution



- SPI transaction
  - shift two bits in for the "address" (one of four)
  - shift six bits in for the potentiometer setting

http://www.analog.com/static/imported-files/data_sheets/AD5203.pdf

Ex.: AT45DB161 flash memory



http://www.atmel.com/dyn/products/product_card.asp?family_id=616&family_name=DataFlash&part_id=1890

More SPI Ex: SPI flash memory: AT45DB161

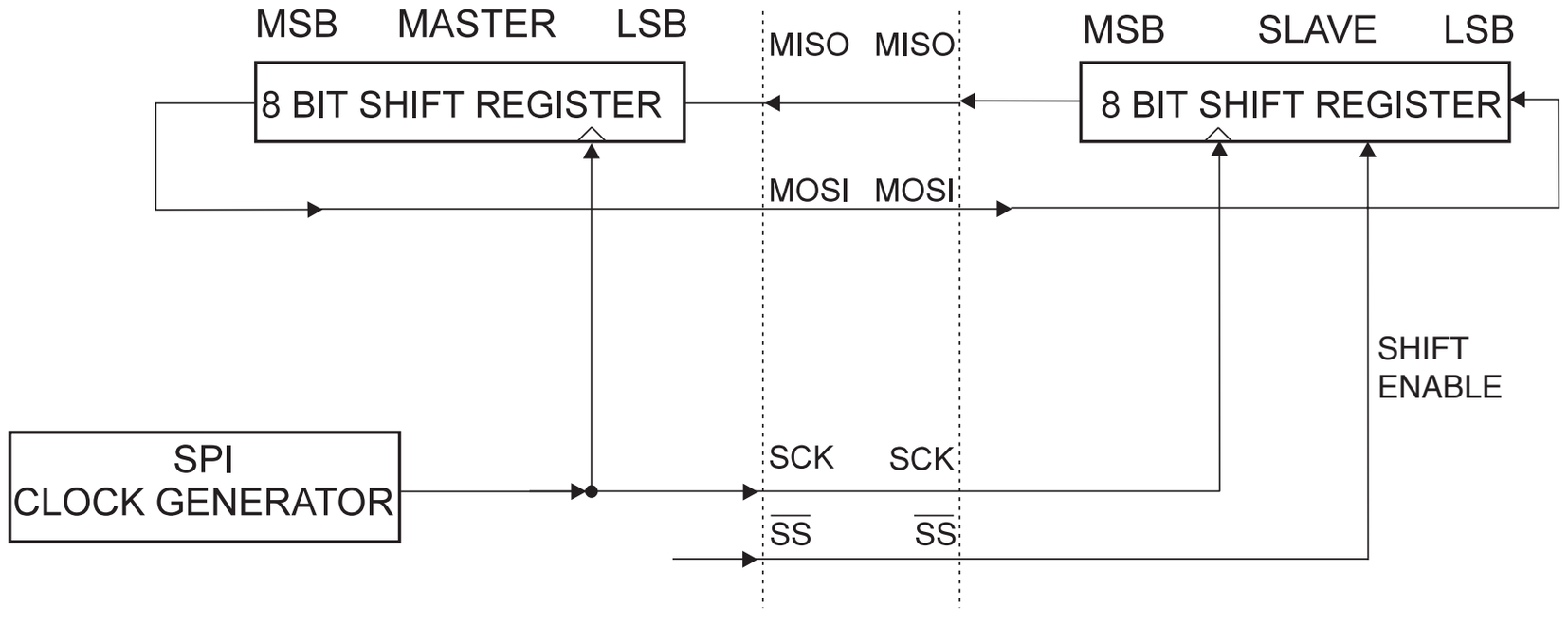
- To use: 8-bit Opcodes followed by address
  - address could be within buffer or the flash memory
    - page size: 512 bytes
- Access: Status register vs. data
- Data sheet: http://www.atmel.com/dyn/resources/prod_documents/doc0807.pdf

Some Opcodes for AT45DB161

read mem	52H	main mem page read, 12-bit page address (and then wait cycle, shift data)
read buffer	54H	Buffer1 read
	56H	Buffer2 read
"read" mem to buffer	53H	Main mem page to Buffer1 xfr
	55H	Main mem page to Buffer 2 xfr
write buffer	84H	Buffer1 write, followed by 10-bit buffer address
erase	81H	Page erase
	50H	block erase.
write page thru buffer	58H	auto page rewrite thru Buffer1
	59H	auto page rewrite thru Buffer2

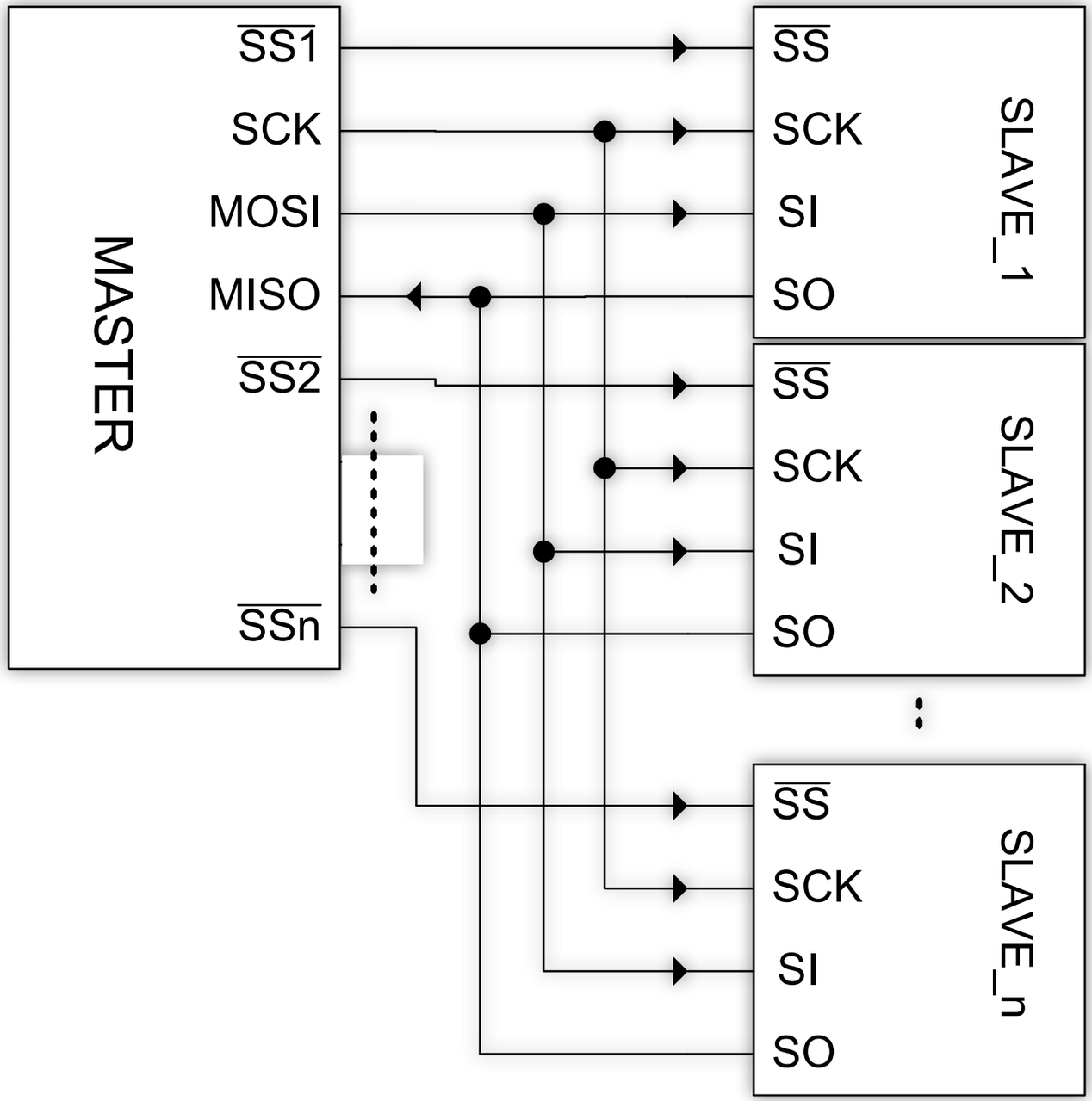
SPI as a bus -- multiple devices

- One SPI master
- Share SCK, MOSI, MISO lines
- One /SS per slave; conceptually point-to-point



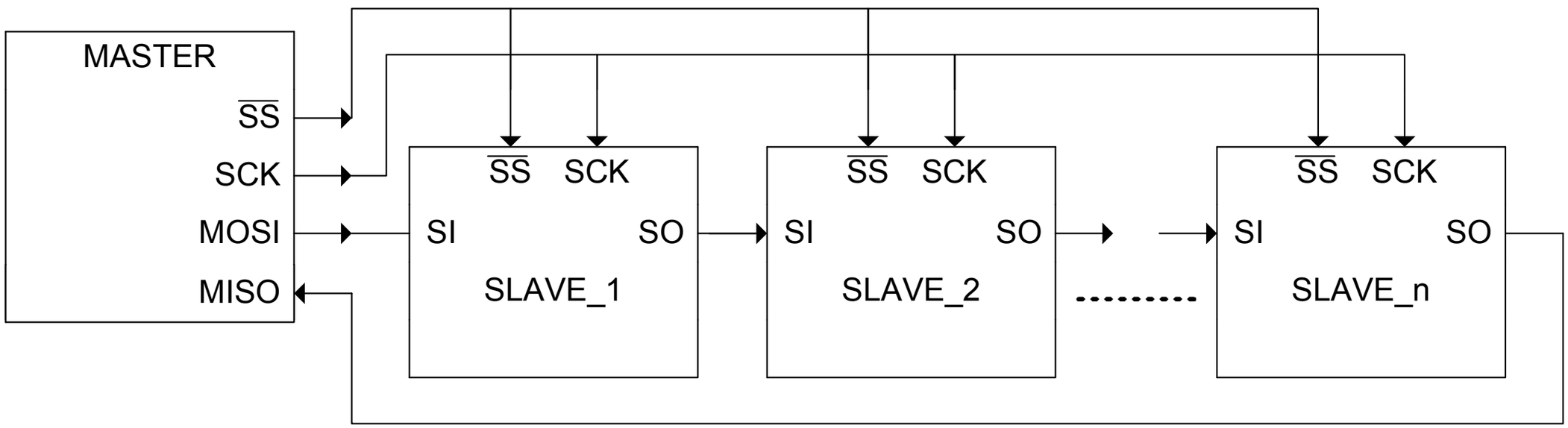
Basic SPI bus topology

- One Master, Multi-Slave
- shared MISO, MOSI, SCK
- one /SS per slave



Cascaded topology

- One /SS selects everybody!
- SO of one slave feeds into SI of next, then wrap around back to master



SPI issues

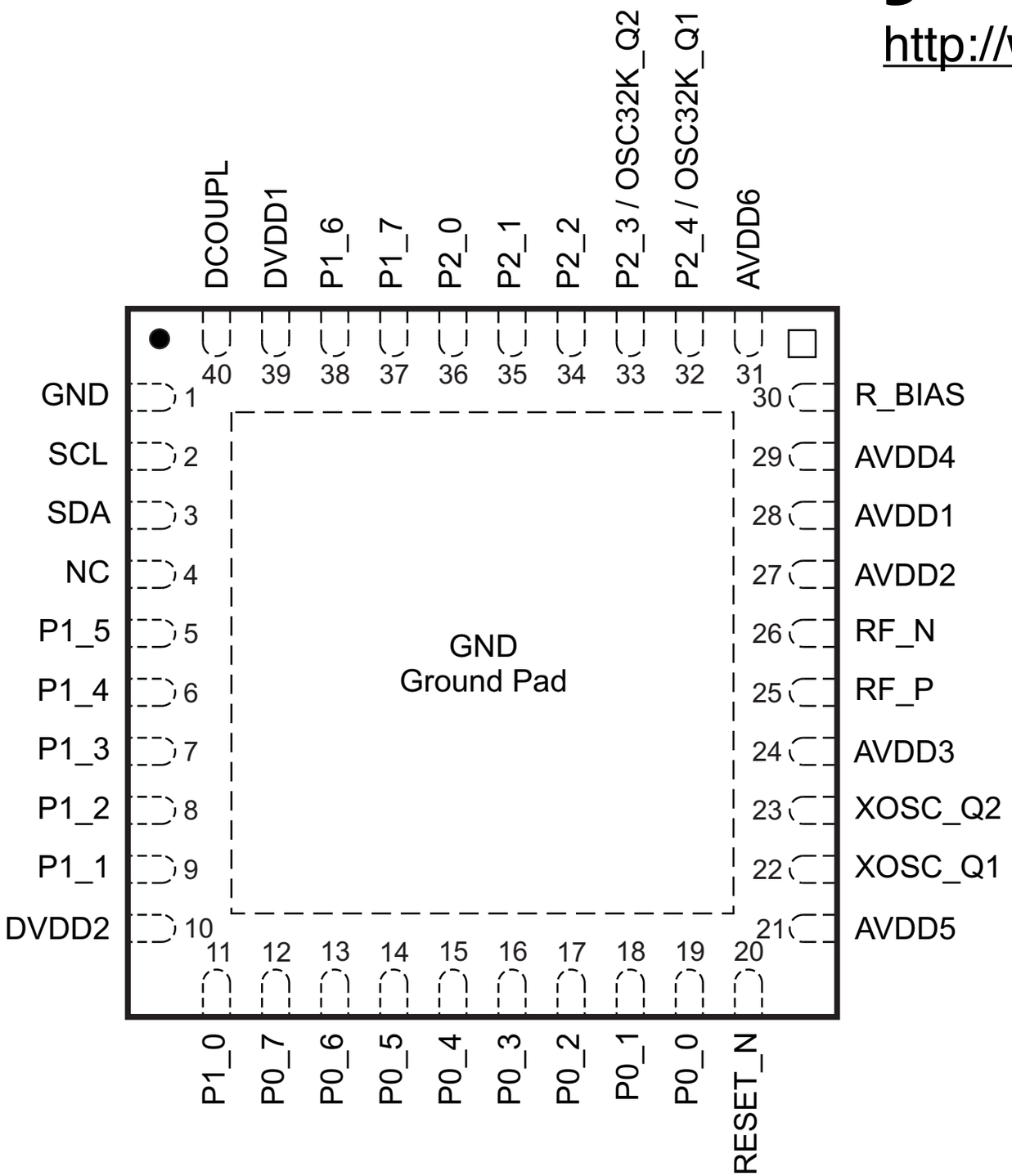
- MCU integration and programming
  - integrated controller
  - software controller
- Programmable Master/Slave role
- Slave-to-slave transfer

Integrated SPI controller in MCUs

- Many modern MCUs have built-in SPI
  - Programmable clock rate, 1,2,4,8MHz etc
  - read / write SFR, poll flag, interrupt
    => similar to serial port programming
- Difference: one register w/ read buffer (8051 SBUF is actually one name for two registers!)

Case Study: CC2541

http://www.ti.com/lit/ds/symlink/cc2541.pdf



- Where is it?
- Look for "USART" in the Users guide
- Latest copy at

http://www.ti.com/lit/ug/swru191f/swru191f.pdf

SPI Pin mapping on CC2541

Table 7-1. Peripheral I/O Pin Mapping

Periphery/ Function	P0								P1								P2				
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	4	3	2	1	0
ADC	A7	A6	A5	A4	A3	A2	A1	A0													T
Operational amplifier						O	–	+													
Analog comparator			+	–																	
USART 0 SPI			C	SS	MO	MI															
Alt. 2											M0	MI	C	SS							
USART 0 UART			RT	CT	TX	RX															
Alt. 2											TX	RX	RT	CT							
USART 1 SPI			MI	M0	C	SS															
Alt. 2									MI	M0	C	SS									
USART 1 UART			RX	TX	RT	CT															
Alt. 2									RX	TX	RT	CT									
TIMER 1		4	3	2	1	0															
Alt. 2	3	4												0	1	2					
TIMER 3												1	0								
Alt. 2									1	0											
TIMER 4															1	0					
Alt. 2																		1			0
32-kHz XOSC																	Q1	Q2			
DEBUG																			DC	DD	
OBSSEL											5	4	3	2	1	0					

http://www.ti.com/lit/ug/swru191f/swru191f.pdf

SFRs for SPI configuration on CC254x

- Selecting SPI mode
  - USARTx: set UxCSR.MODE = 0 (where x = 0 or 1)
- Selecting alt1 or alt2 SPI config:
  - USARTx: set PERCFG.UxCFG = 0 or 1
- Configure for master or slave
  - USARTx: set UxCSR.SLAVE bit

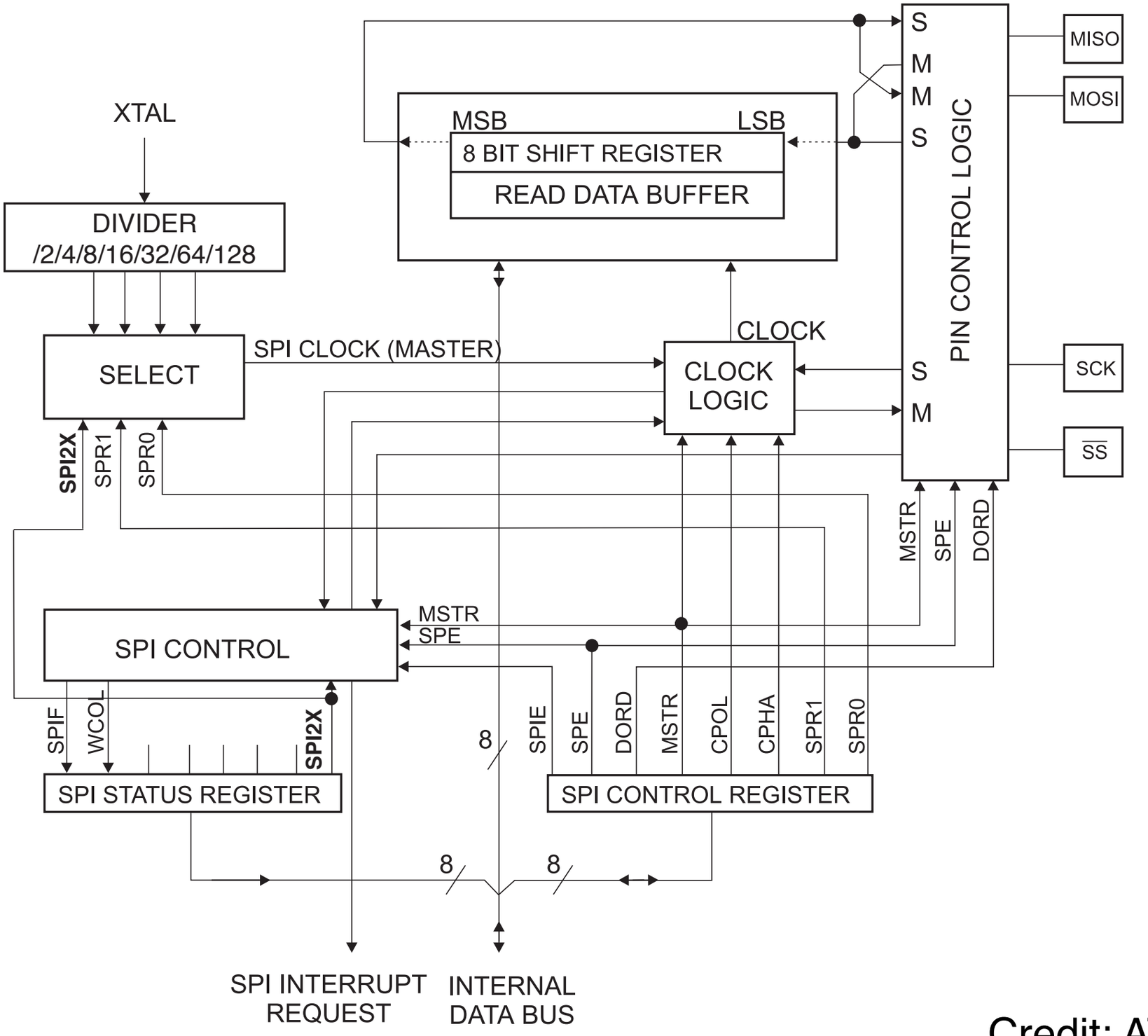
SFRs for SPI configurations for CC254x

- Polarity: UxGCR.POL
- Phasing: UxGCR.CPHA
- Endian: UxGCR.ORDER
- Master clock rate: UxBAUD.BAUD_M

SFRs for SPI operation

- Status:
  - UxCSR.RX_BYTE (1: byte received)
  - UxCSR.TX_BYTE (1: transmit ready)
  - UxCSR.ACTIVE (1: active as slave)
- data buffer
  - UxDBUF (same name for Rx & Tx regs)

SPI controller for Atmel



Credit: ATMEL

SPI Programming for ATMega168 (,48, 88)

- Can be configured as master or slave
- Registers
  - SPCR (control register)
  - SPDR (data register)
  - SPSR (status register)

SPCR: SPI "control" register (configuration)

- Bit 7: SPIE (SPI interrupt enable)
- Bit 6: SPE (SPI enable)
- Bit 5: DORD (data order) = endian
- Bit 4: MSTR (master/slave select)
- Bit 3: CPOL (clock polarity)
- Bit 2: CPHA (clock phase)
- Bit 1,0: SPR1,0 (clock rate) div by 2..128

SPSR: status register

- Bit 7: SPIF (SPI interrupt flag)
- Bit 6: WCOL (write collision)
- Bit 5..1: reserved
- Bit 0: SPI2x (double speed)

SPI programming as master

- Write to SPDR
  - MCU starts SPI clock generator, shift bits
  - MCU sets SPIF flag to indicate end of Tx
    - programmer must assert own /SS!
- Rx *Double buffering* (e.g., on ATMega168)
  - Separate registers for Rx and Tx
  - can still read last Rx'd byte while swapping byte

MCU as SPI slave

- /SS becomes input
  - No data shifted unless /SS asserted
- Writing: when master decides to read!
  - slave wrote to SPDR first, and data gets read (swapped) by master later
- Reading: upon SPIF (finishes byte swap)
  - slave should read SPDR ASAP or else it may be overwritten

ATMega168 master SPI mode (MSPIM)

- USART can be configured as SPI master (not as slave)
- UCPOLn, UCPHAn for polarity/phase
- TXCn, RXCn flags: xmit ready, data ready =>TXCn must be cleared explicitly
- USART has extra features
  - Double buffering, WCOL (write collision)

I²C

I²C Bus

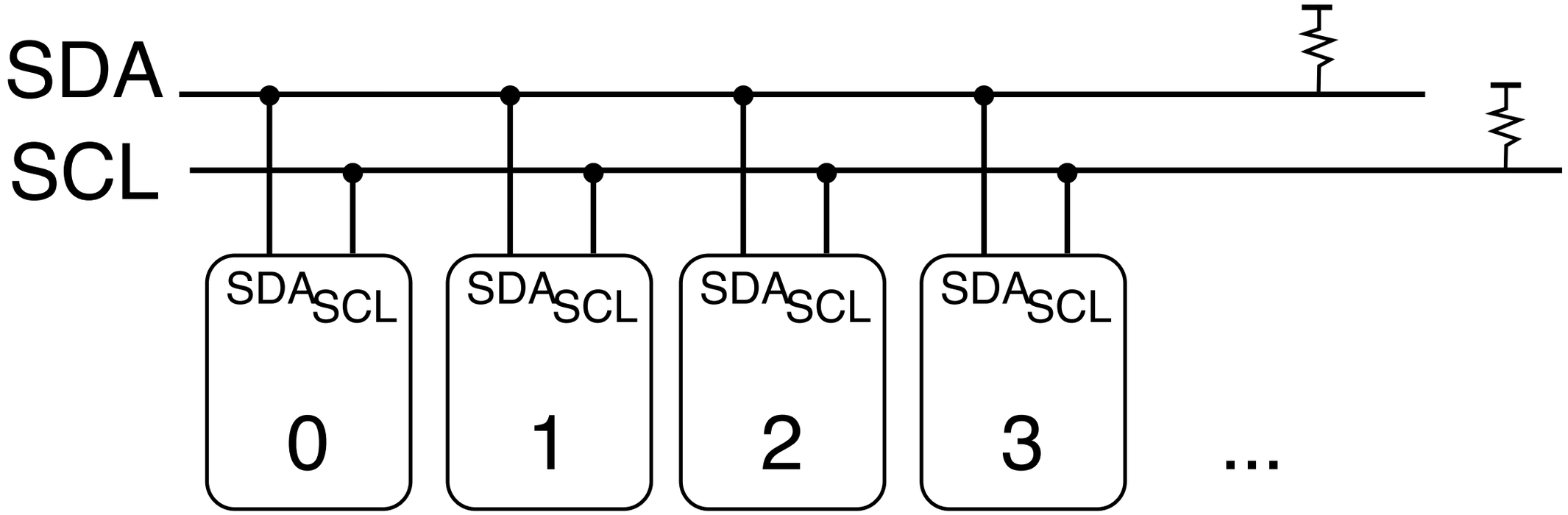
- Developed by Philips / Signetics (now NXP)
- Application: connect audio/video boxes
- Objective: minimize # of wires, glue logic
- Issues
  - Resolving different clock speeds
  - Multiple access: arbitration, addressing
- Reading: Chapter 8 of Catsoulis's book

I²C: Inter-IC Circuit

- Serial bus, sharing two wires (plus ground)
  - SDA: serial data
  - SCL: serial clock
- That's it! *no separate CE or CS per device!*
  - I²C defines own protocol w/ arbitration
  - Multi-master, priority-based

I²C topology

- They should share the same GND, too!
- Both SDA and SCL should be pulled up

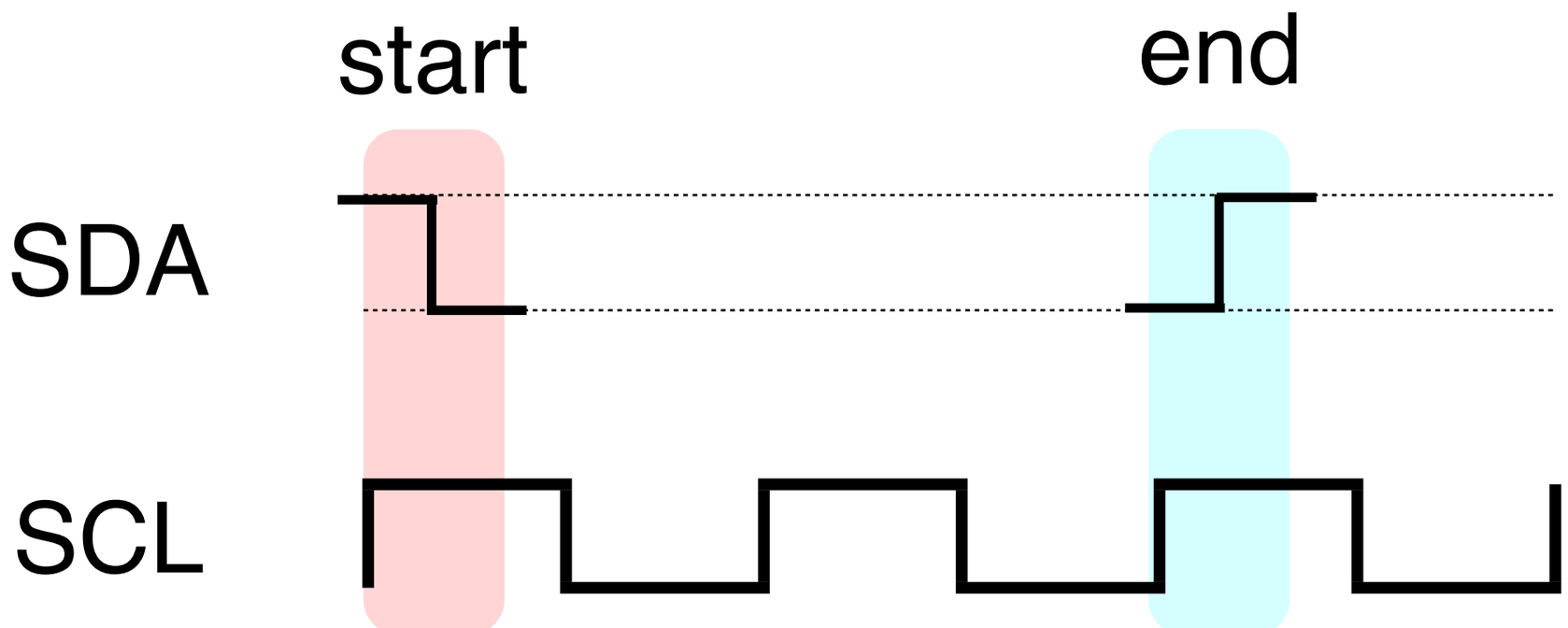


I²C protocol: three phases

1. Start: by any node as if it's master
    - All nodes on the bus should pay attention
2. Normal transaction:
    - write slave ID, R/W, and data transfer
3. End:
    - by the current master

Start and End conditions

- Normally, SDA should be stable when SCL high
- Start: SDA falls while SCL high
- End: SDA rises while SCL high

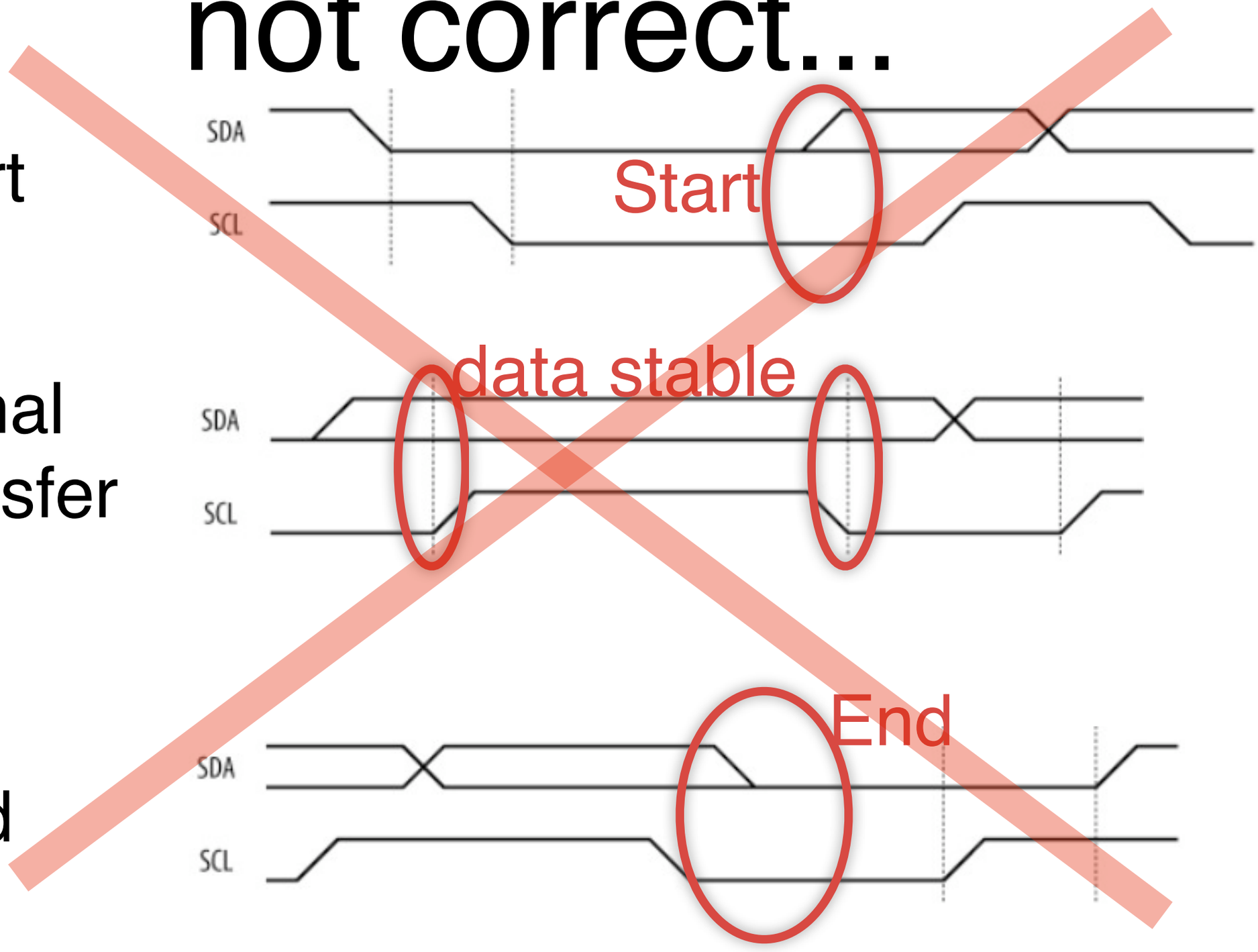


Catsoulis's book - not correct...

Start

Normal bit transfer

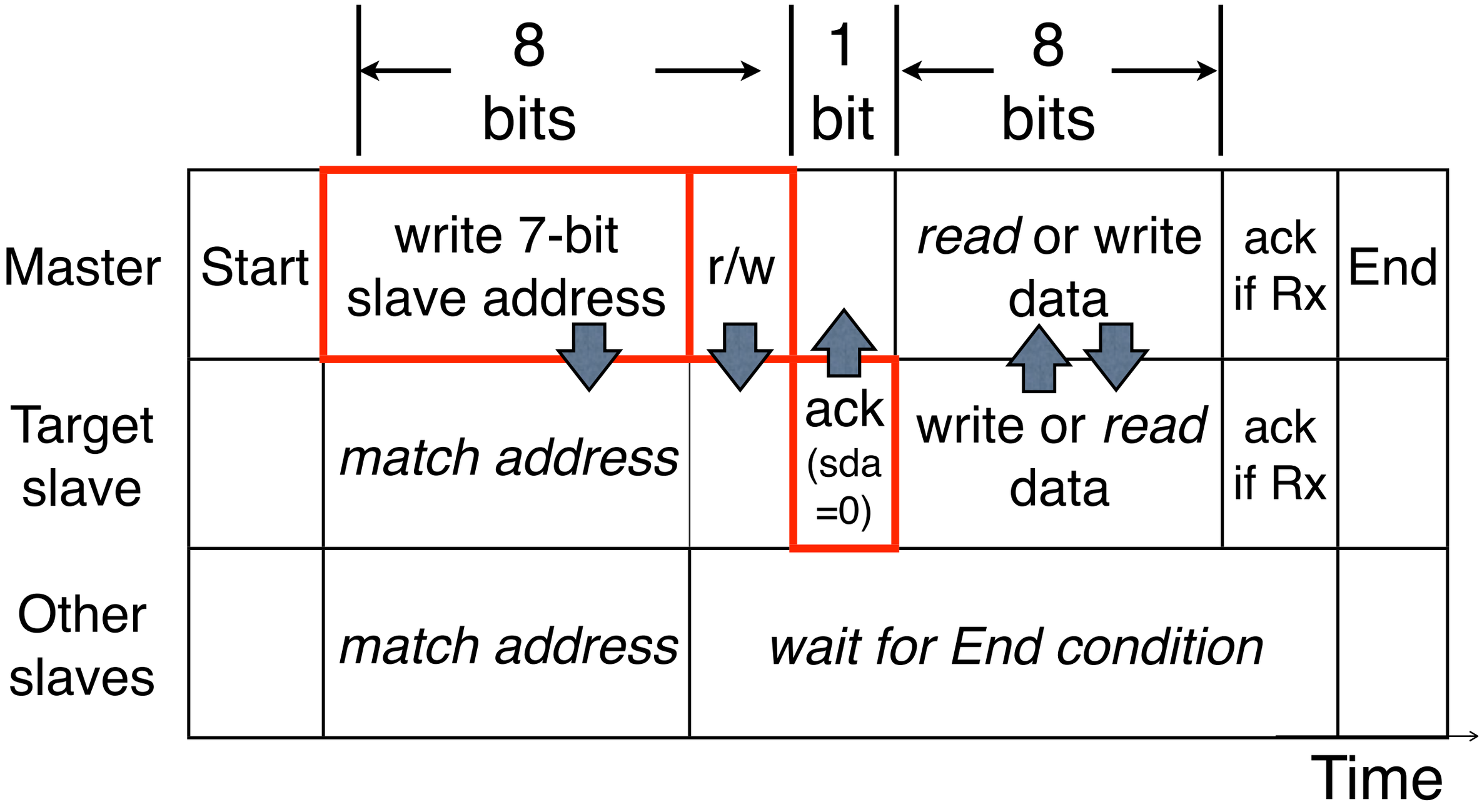
End



Slave addressing

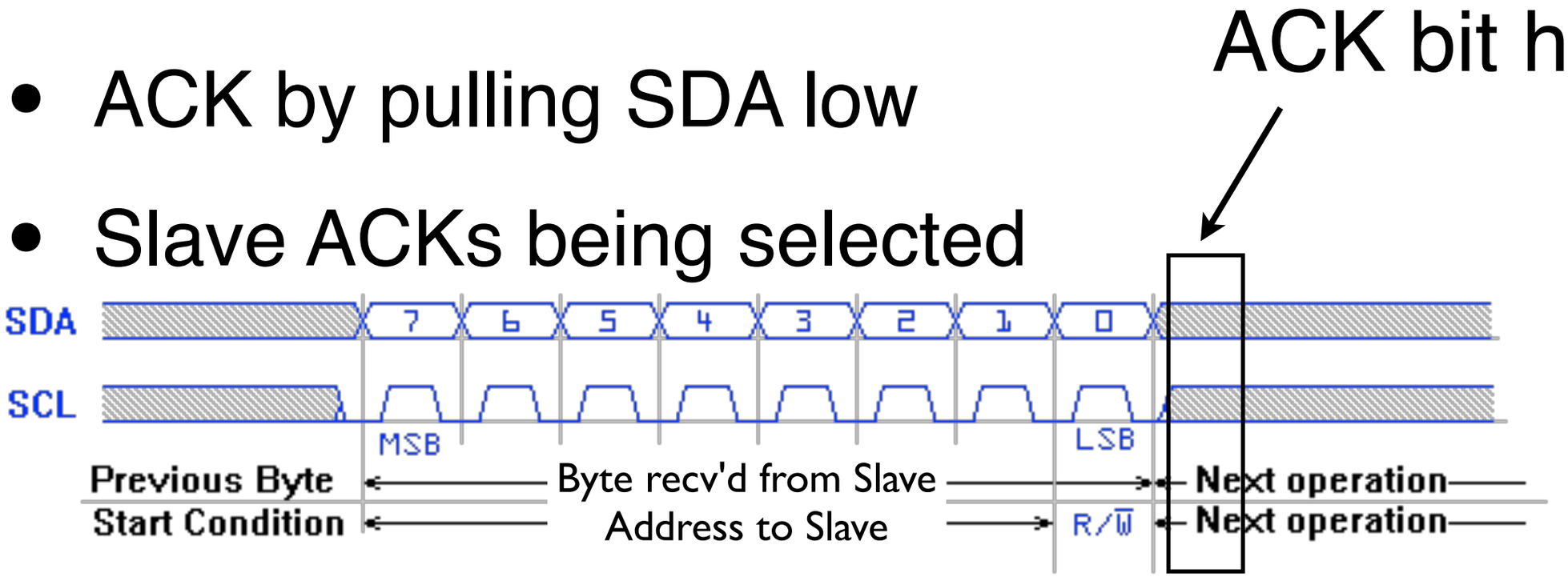
- Master writes slave ID (7-bit)
  - Every node compares its own ID
  - if ID matches, then the slave is selected
    - **slave Acks by pulling SDA low**
    - transfers data, then End
- Note: END can come any time!

Typical sequence

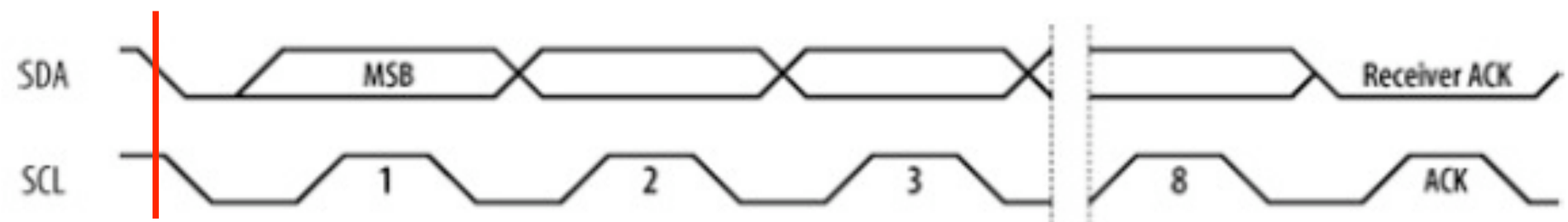


Acknowledgments

- ACK by pulling SDA low
- Slave ACKs being selected

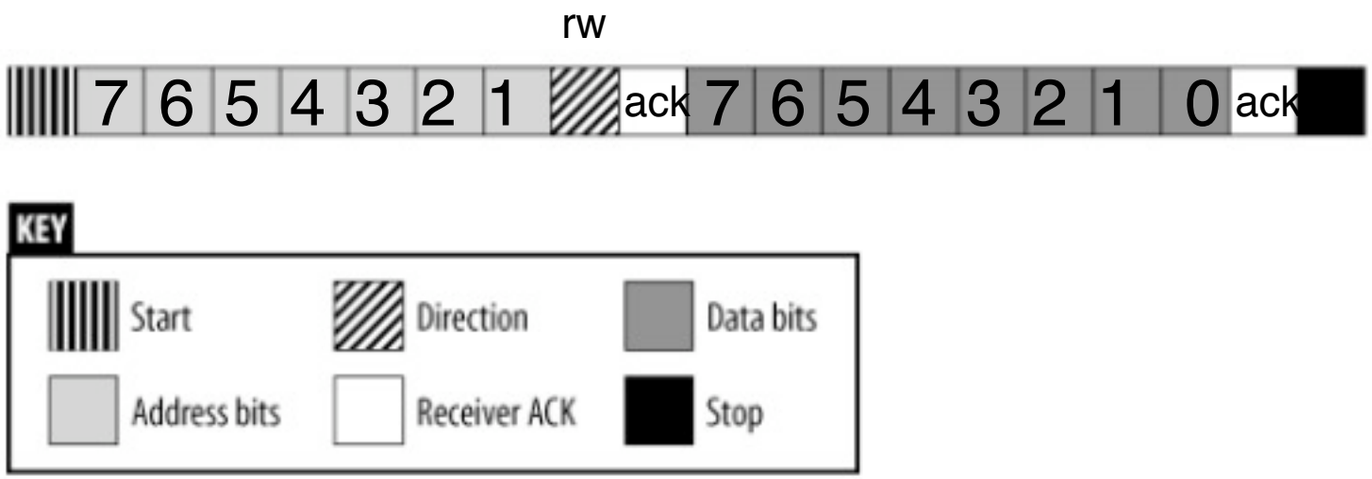


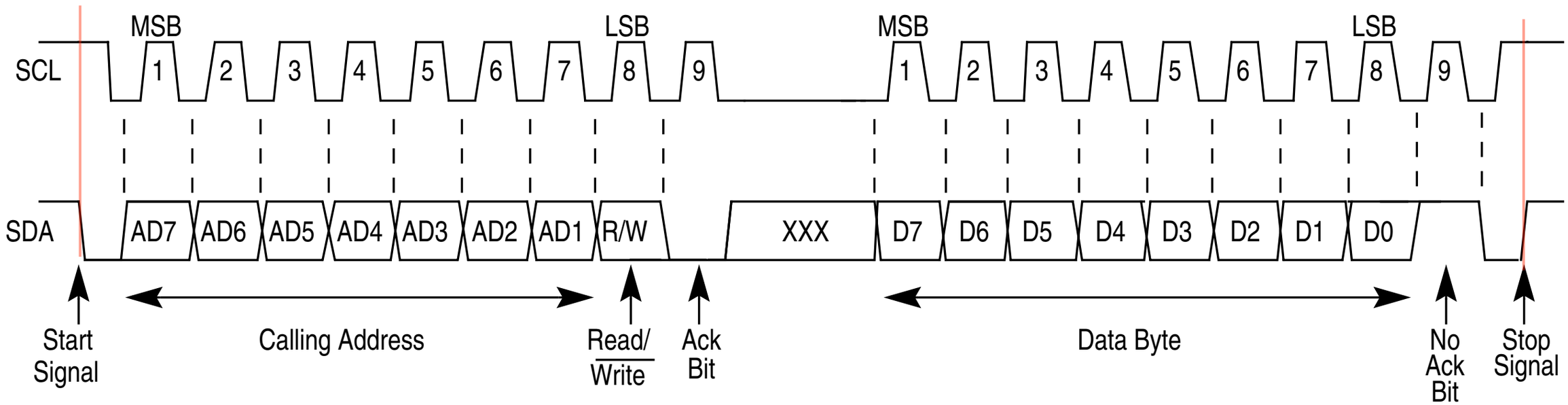
- Master ACKs to slave after receiving a byte



start

Regular I²C transaction





Issues

- Bus Arbitration
- Clock Synchronization
- Repeated Start and acknowledgment
- Special Addressing
- Extensions

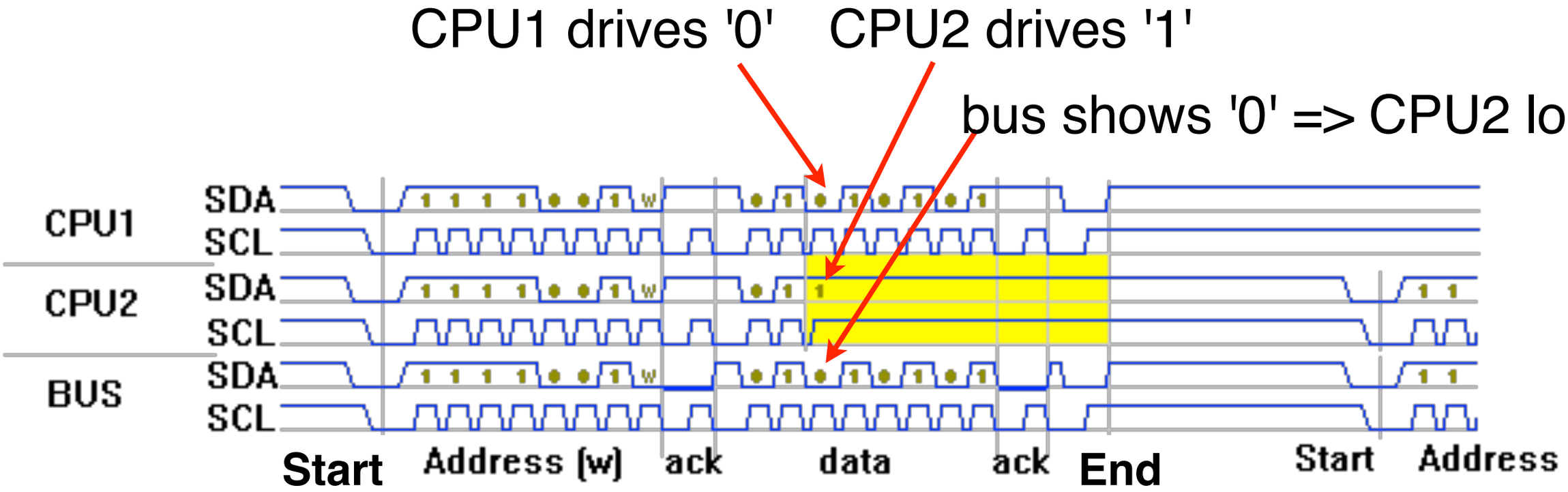
Bus Arbitration

- What if multiple nodes want to be master?
  - They should all watch for Start cond., but what if they missed it? (just powered on)
- Solution: "watch what you drive"
  - If different => you lose!
  - Wait for End condition, then start.

Arbitrate by "Watching what you drive"

- SCL, SDA lines are pull-up
  - Conceptually open-collector output:
- '0': pull-down by someone
  - Whoever drives '0' always wins arbitration
- '1': disconnected
  - Whoever wants '1' may lose!
  - Wait for End condition and re-try

Bus arbitration example



both CPU1, CPU2 drive identical values	CPU2 loses, backs off & waits for End	Idle	New trans.

credit: http://www.esacademy.com/faq/i2c/general/i2carbit.htm

Clocking Synchronization

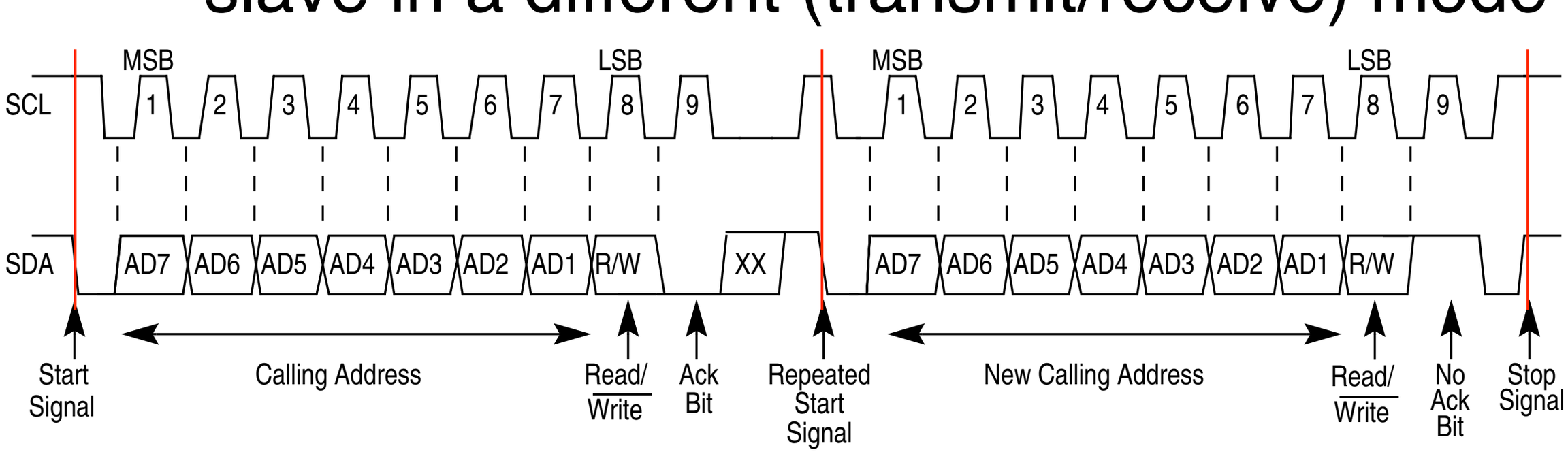
- Data is valid only during SCL = high
- Master drives SCL low, release to go high
- Slow device can hold SCL low
  - Master waits till SCL high
  - Works at the slowest speed of all!
  - Idea borrowed from GPIB (IEEE 488)

Clock sync. & Bus arbitration mechanisms

- Advantages
  - simple mechanism, need no add'l wires
  - Bus is utilized even during arbitration
    - Force backoff by pulling both SCL,SDA '0'
- Disadvantages
  - Can deadlock if SCL output is stuck low

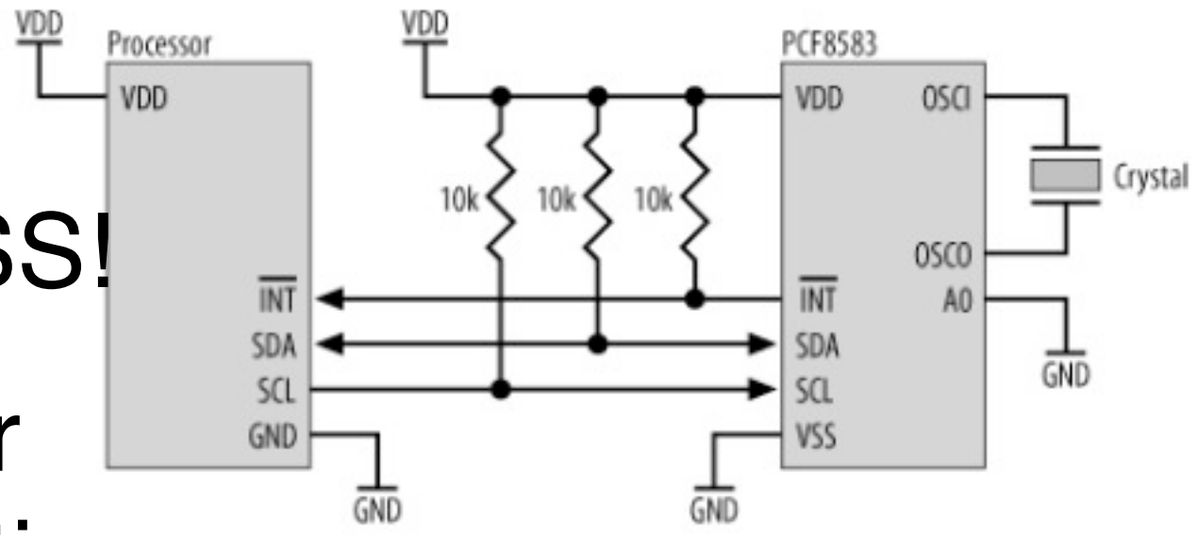
Option to continue an I²C transaction

- "Repeated Start" signal:
  - to communicate w/ another slave or same slave in a different (transmit/receive) mode

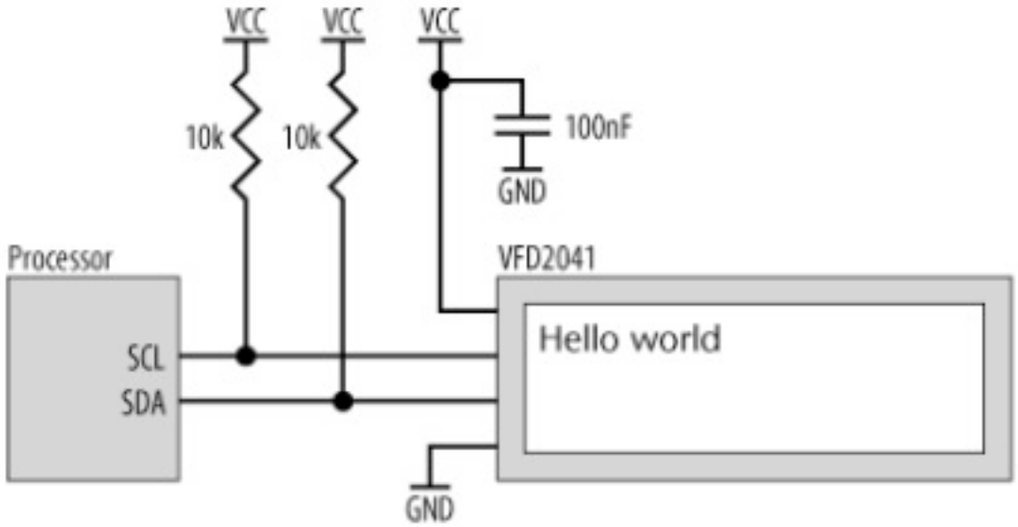


Ex. I²C app.: PCF8583 clock/calendar chip

- Simple connection
  - SDA, SCL, no /SS!
  - separate /INT for interrupt notification
- Need pull-up resistors
- Data sheet: http://www.nxp.com/acrobat_download/datasheets/PCF8583_5.pdf



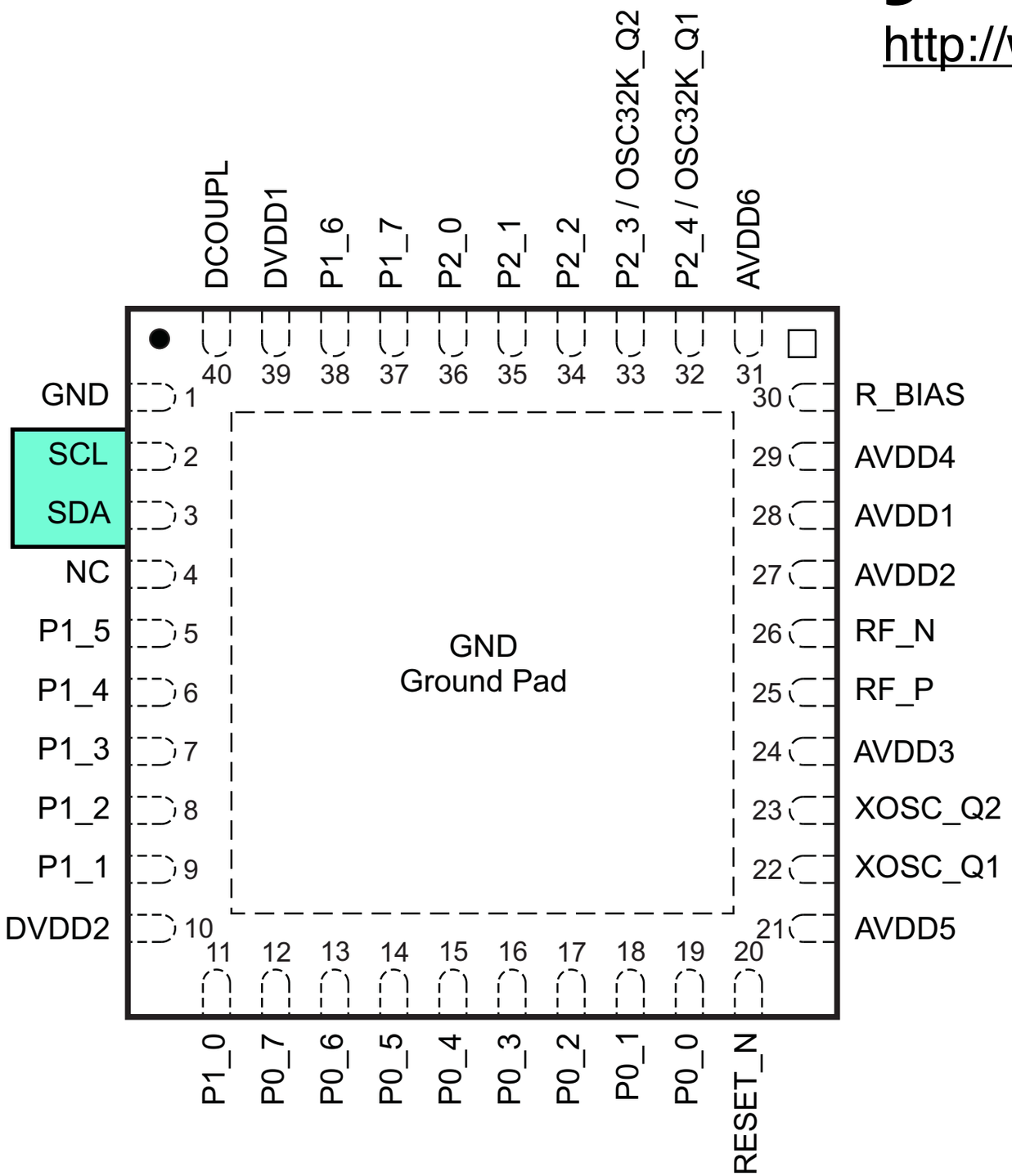
Another Ex. I²C app.: LCD VFD2041



- Both I²C and UART
- up to 80 ch x 4 lines
- Data sheet: http://www.matrixorbital.ca/manuals/LCDVFD_series/VFD2041/VFD2041_150.pdf

Case Study: CC2541

http://www.ti.com/lit/ds/symlink/cc2541.pdf



- Where is I2C?
- Look for SDL, SDA

http://www.ti.com/lit/ug/swru191f/swru191f.pdf

I2C on CC2541: Configuration

- I2CADDR.ADDR: own address
- I2CCFG.CR[2:0]: 123-533KHz clock
- I2CCFG.ENS1: enable I2C
- I2CWC.{SCL,SDA}PUE: pull-up enable
- I2CWC.{SCL,SDA}OE: output enable

I2C on CC2541: Control and Operation

- I2CCFG.{STA,STO}: Generate Start, Stop Condition
- I2CDATA: I2C data in/out
- I2CSTAT.STC: status code