EEL 4742 – Embedded Systems
Module 9 – SPI & Pixel Display

Hadi M Kamali

Department of Electrical and Computer Engineering (**ECE**)
University of Central Florida

Office Location/phone: HEC435 – (407) 823-0764
webpage: https://www.ece.ucf.edu/~kamali/
e-mail: kamali@ucf.edu

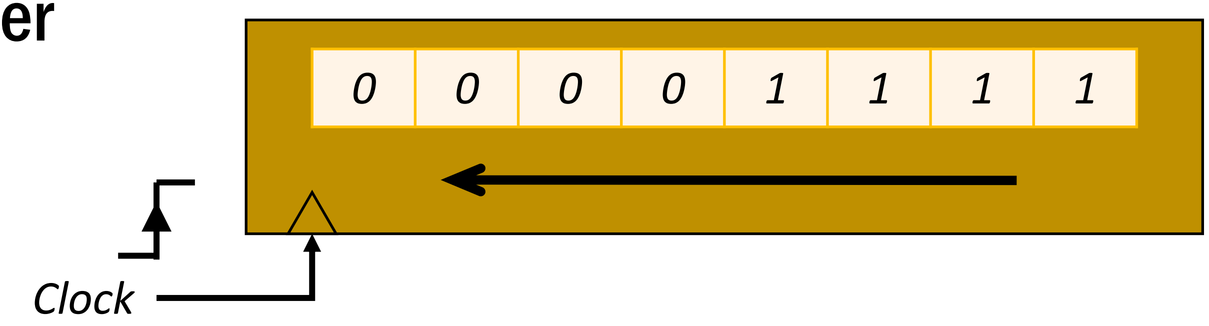
HAVEN Research Group
https://haven.ece.ucf.edu/

Serial Peripheral Interface (SPI)

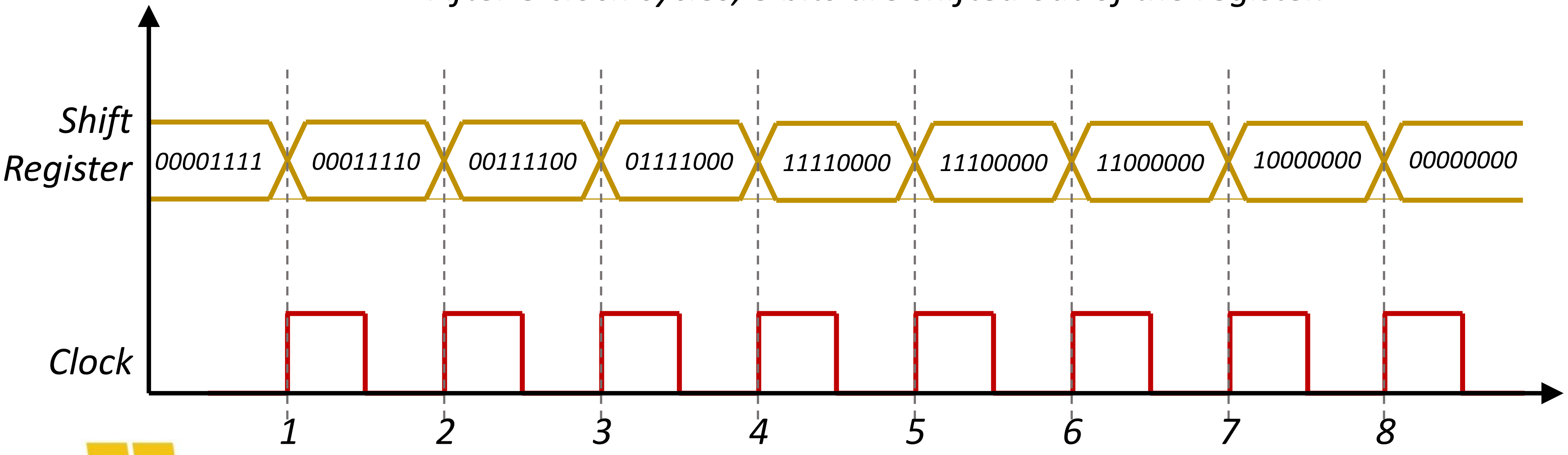
- Serial Peripheral Interface (SPI)
  - Another serial communication protocol
  - Similar to I2C and unlike UART → It is synchronous!
  - Unlike I2C and UART → It is not a standard
    - Terminologies might differ depending on the manufacturer
  - It is based on 4 wires → 4-wire communication
  - It is a master-slave protocol (leader-follower)
  - Full-duplex communication
    - Communication occurs in both directions (simultaneously)

The Basics of Serializing

- Let’s recap on the operation of shift register

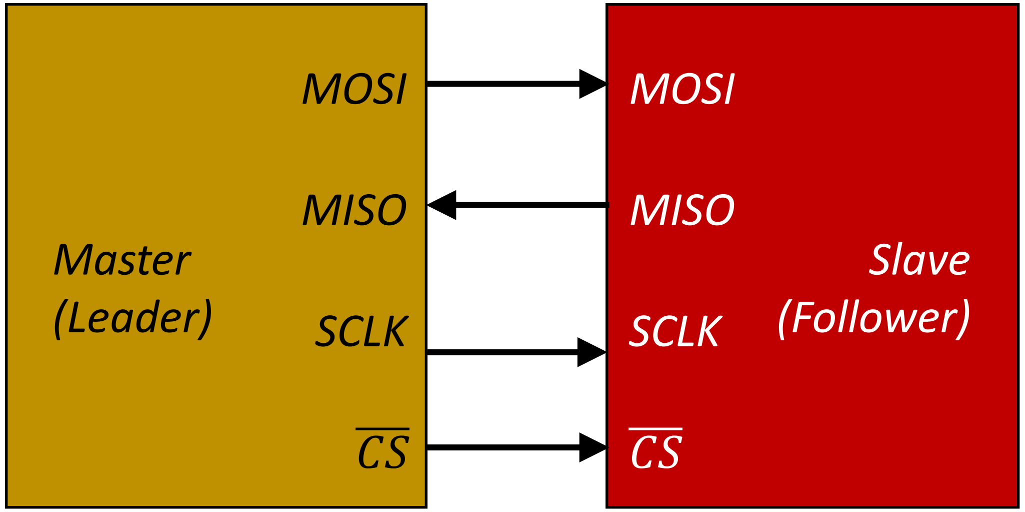


An example 8-bit left shift register.
After 8 clock cycles, 8 bits are shifted out of the register.



Signaling in SPI Communication

- 4-wire Communication
  - MOSI >> Master OUT Slave IN
  - MISO >> Master IN Slave OUT
  - SCLK >> Serial (synchronous) clock source
  - ~CS >> Chip select
    - Chip select is active low

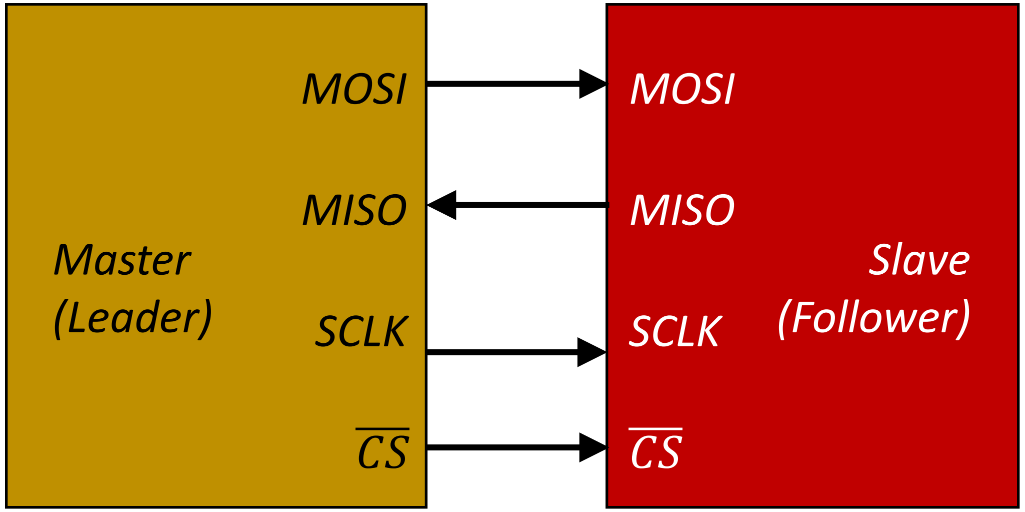


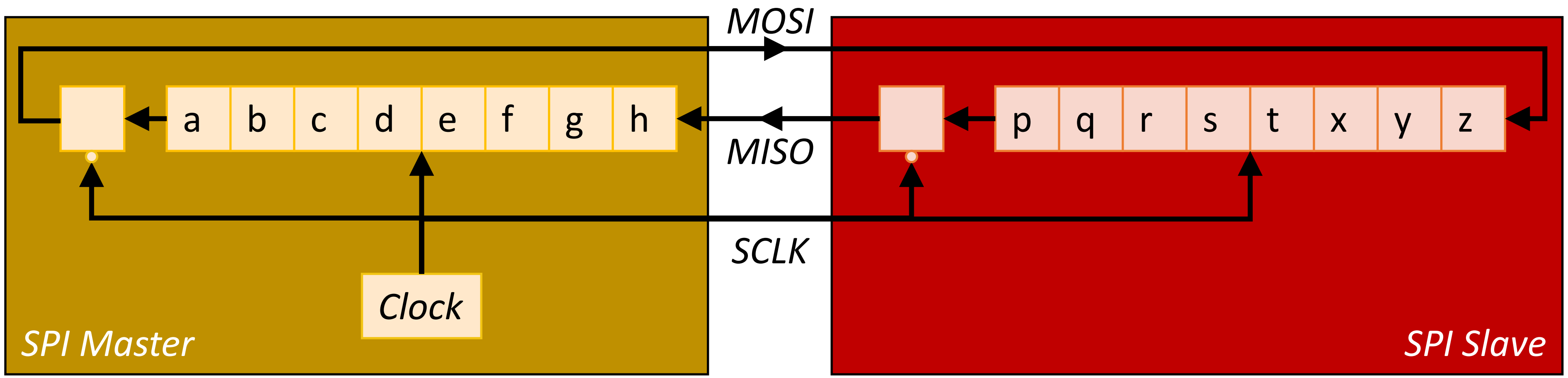
SPI Communication Flow

- The communication occurs in two stages

*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*





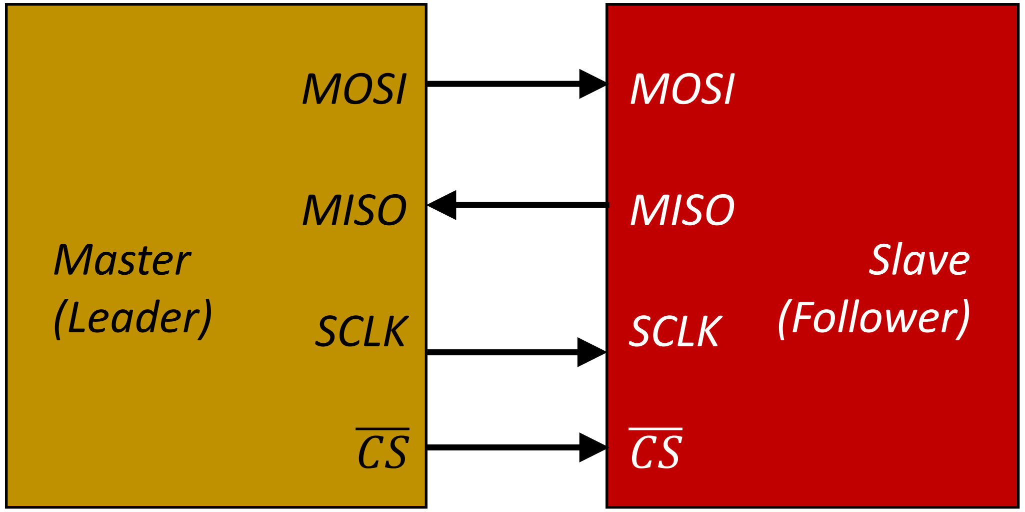
SPI Communication Flow

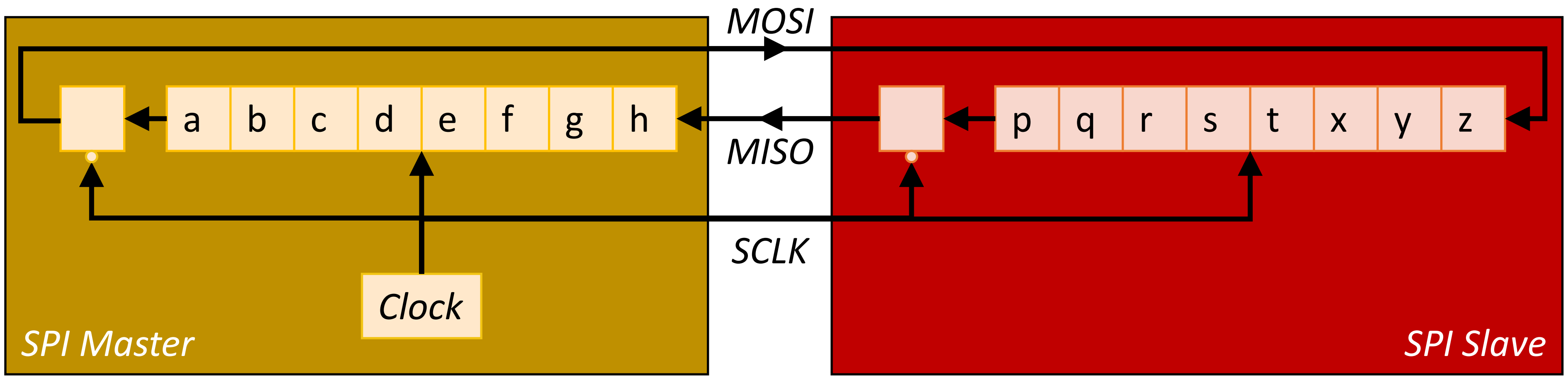
- **The communication occurs in two stages**

*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*

leading edge is a rising edge.
trailing edge is a falling edge.





SPI Communication Flow

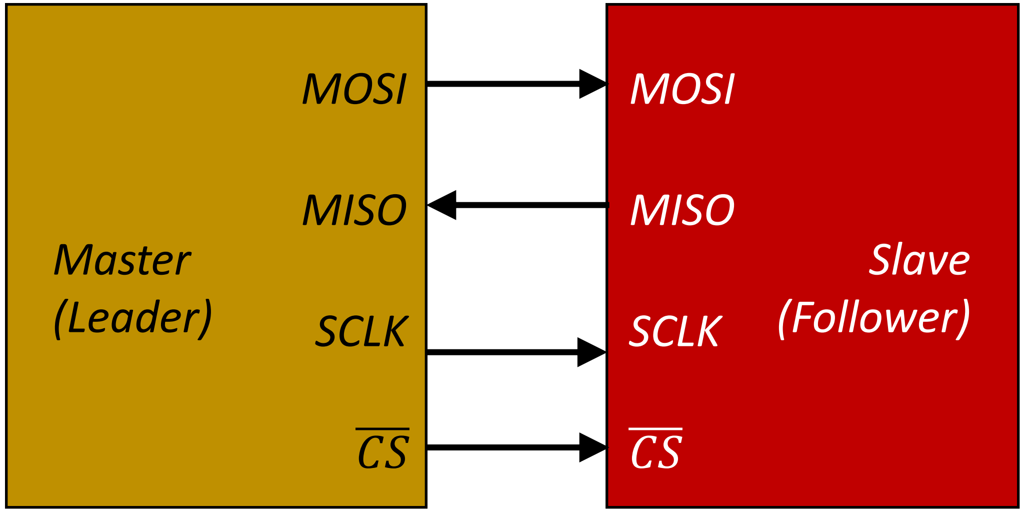
- **The communication occurs in two stages**

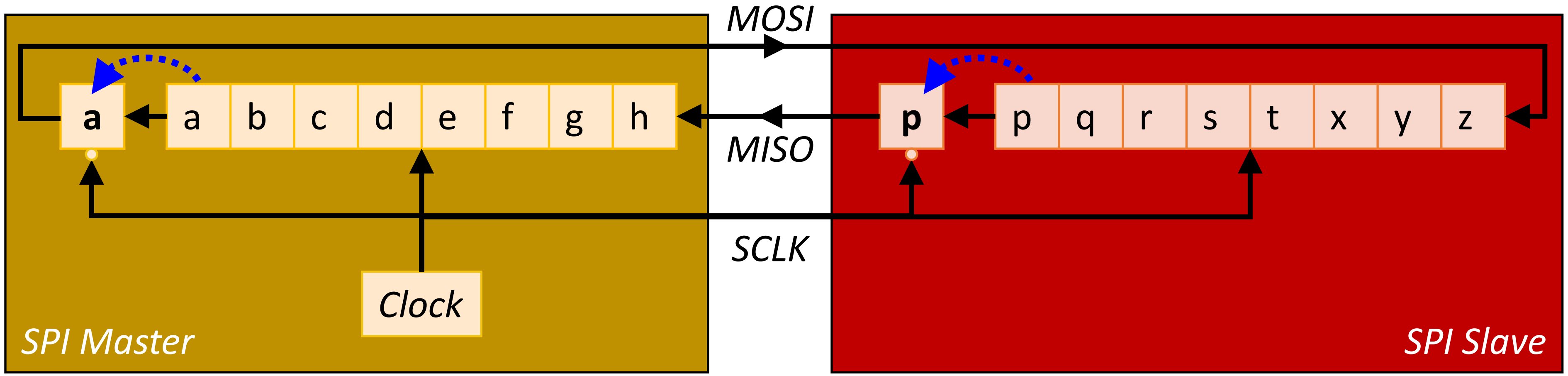
*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*

*At **leading** edge*

The MSBs are latched!



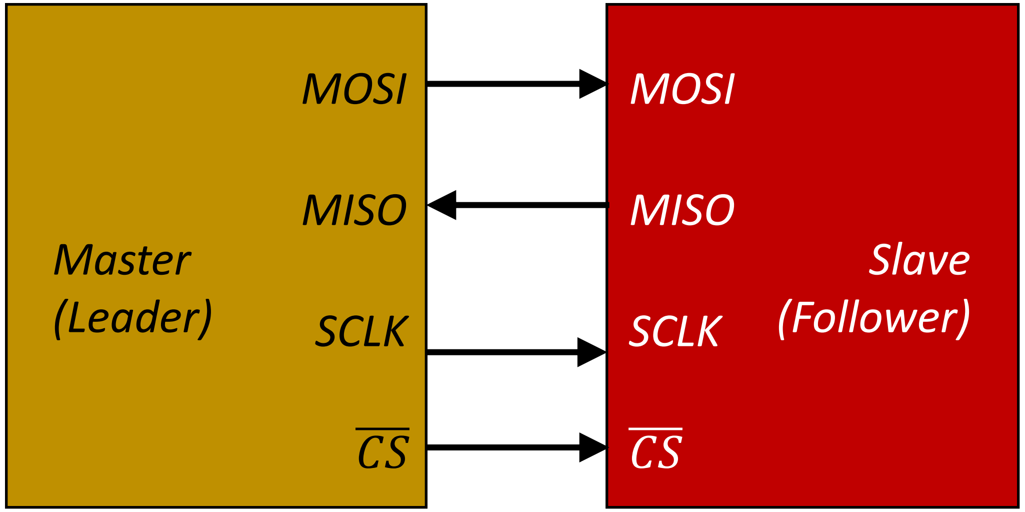


SPI Communication Flow

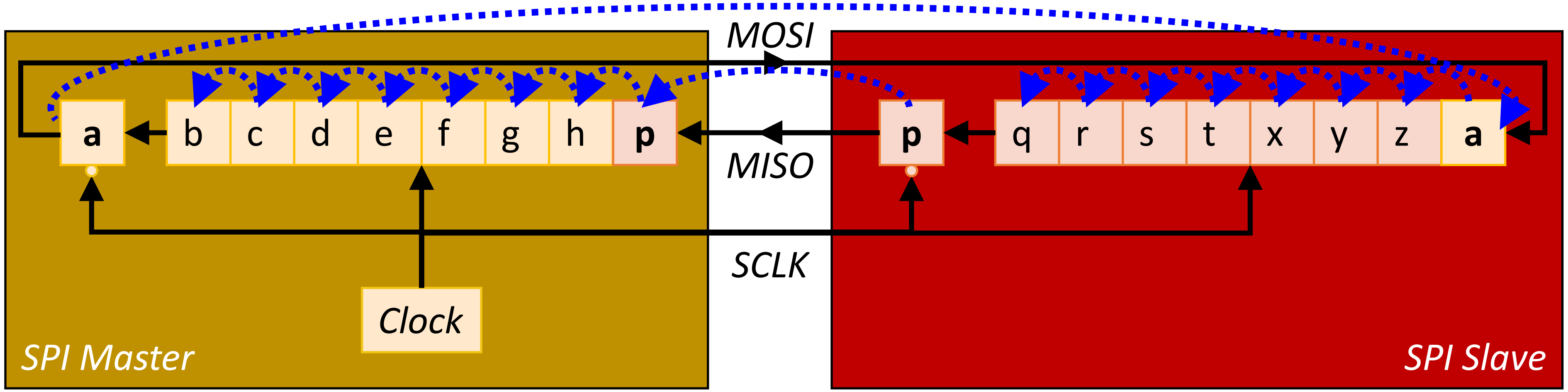
- **The communication occurs in two stages**

*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*



*At **falling** edge*
All registers are left-shifted.
1 bit is transferred between devices.



SPI Communication Flow

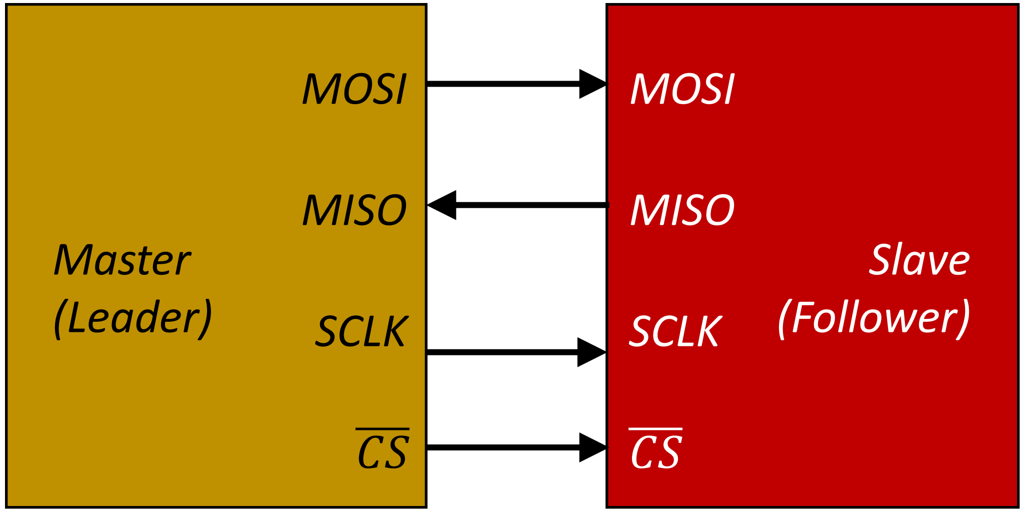
- **The communication occurs in two stages**

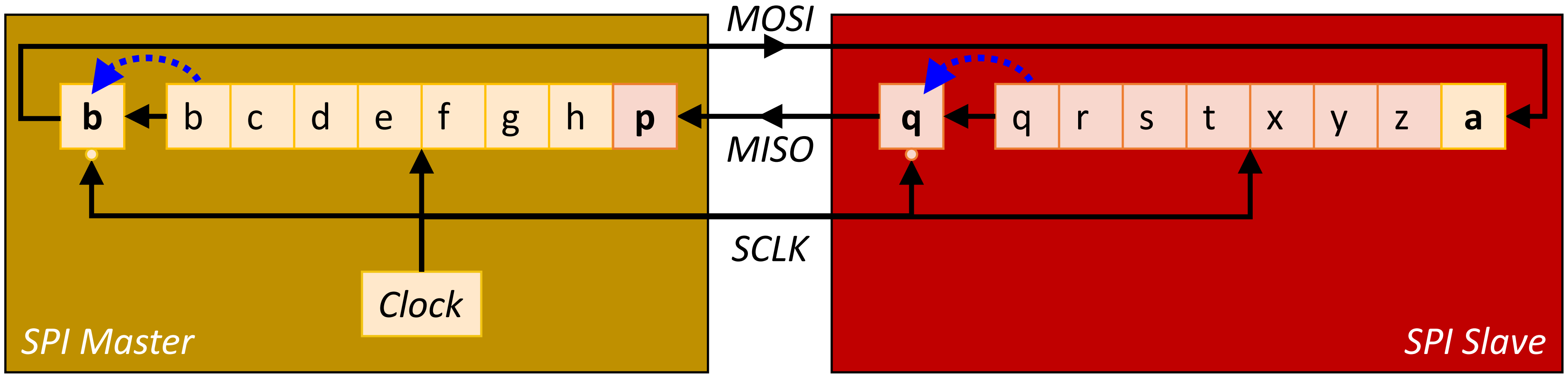
*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*

*At **leading** edge*

The MSBs are latched!



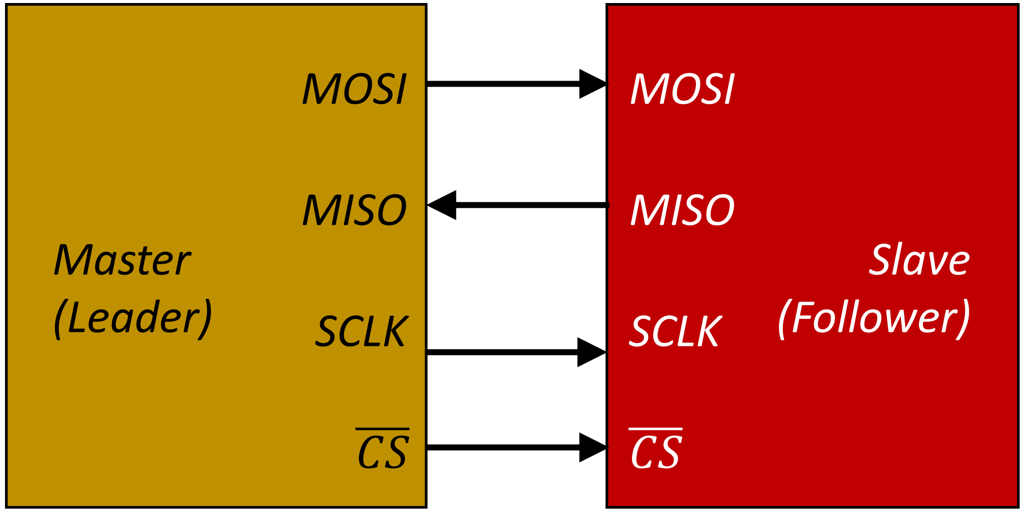


SPI Communication Flow

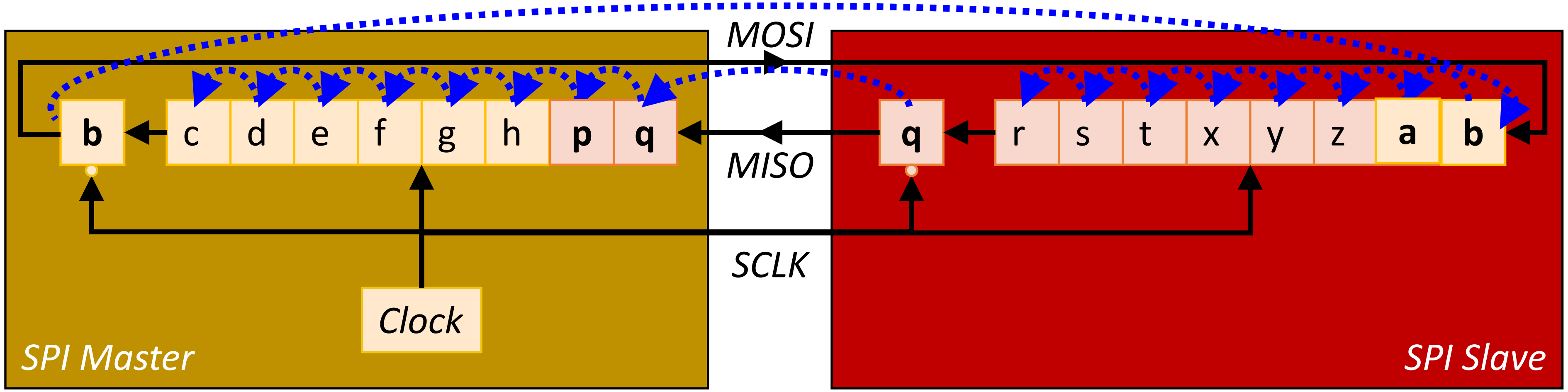
- The communication occurs in two stages

*(stage 1) During the **leading** clock edge, the msb is latched in both the devices.*

*(Stage 2) During the **trailing** clock edge, the register values from both devices are shifted into the two devices.*



*At **falling** edge*
All registers are left-shifted.
1 bit is transferred between devices.



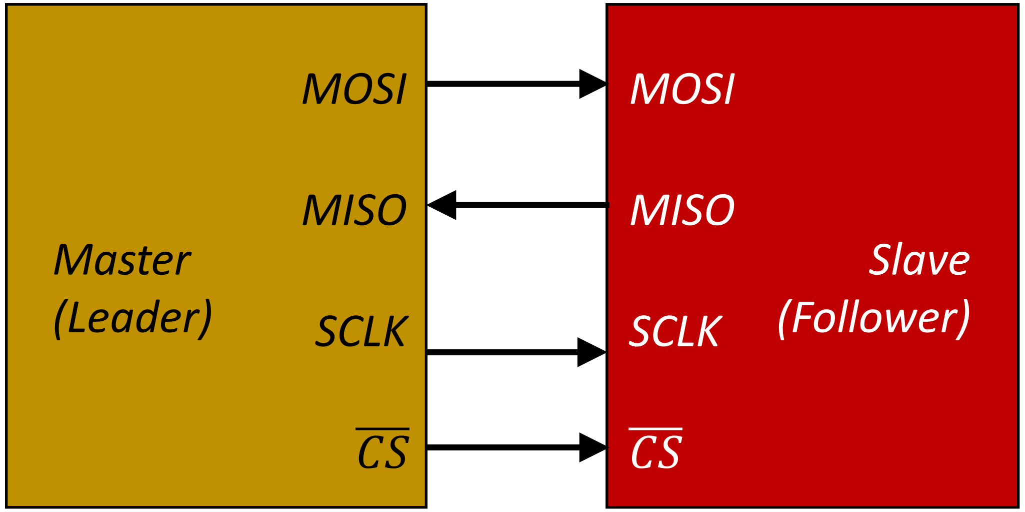
SPI Communication Flow

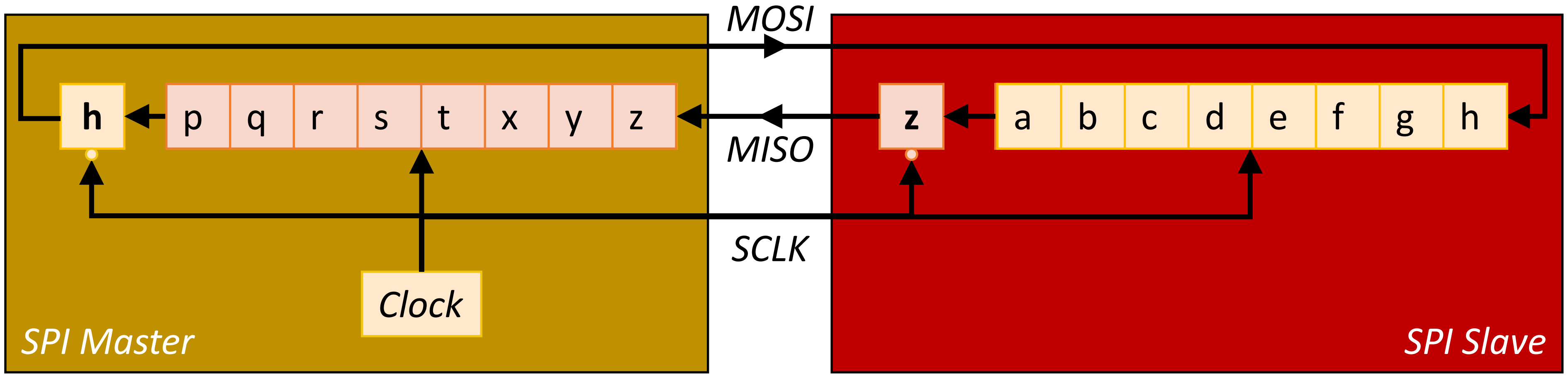
- **The communication occurs in two stages**

(stage 1) During the ***leading*** *clock edge, the msb is latched in both the devices.*

(Stage 2) During the ***trailing*** *clock edge, the register values from both devices are shifted into the two devices.*

After 8 full clock cycles,
8 bits are transferred between devices!





SPI Communication Modes

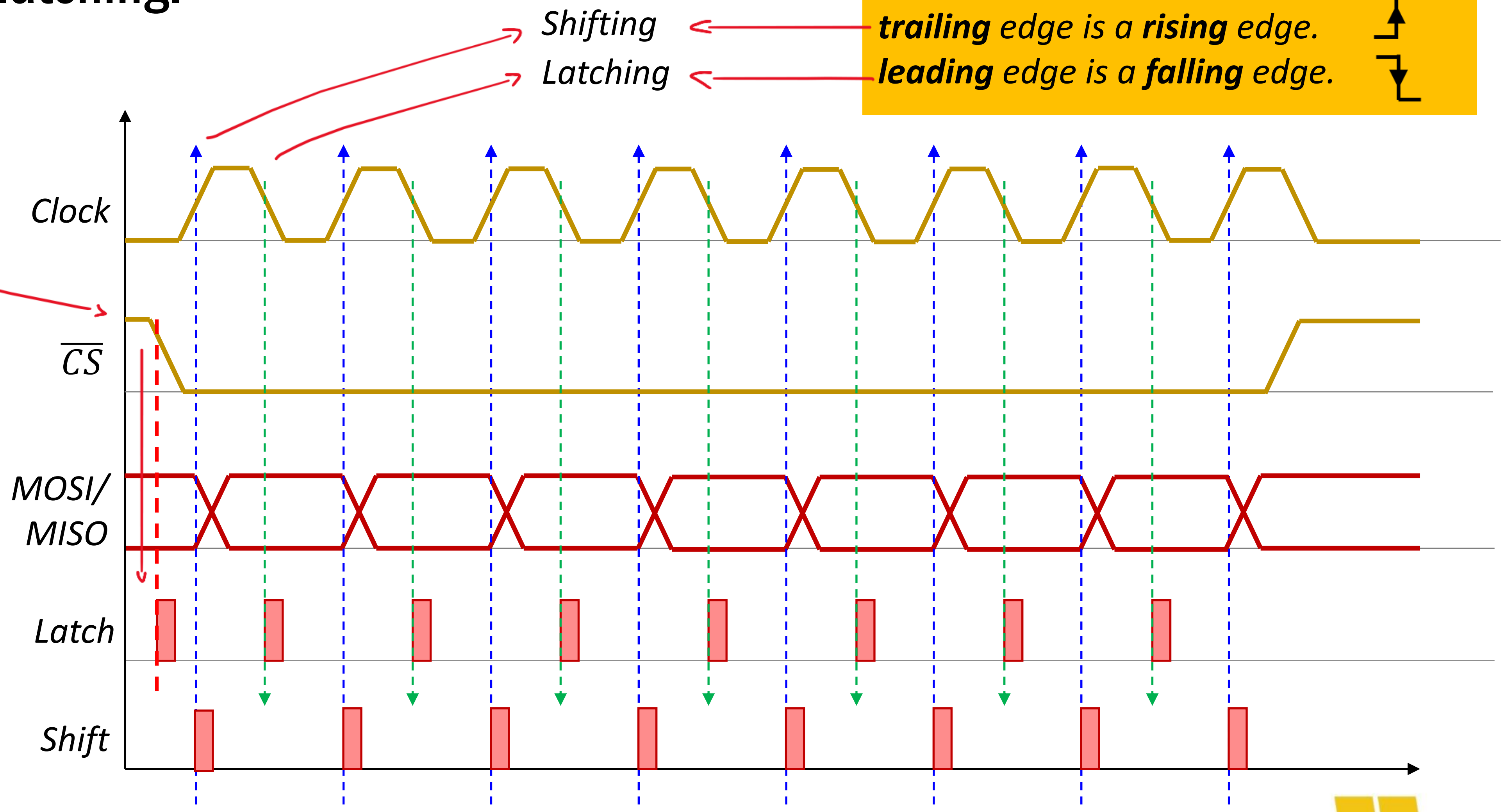
- $\overline{CS}$ causes the first latching.

Clock phase **(CKPH)**	Clock polarity **(CKPL)**
0	0

Clock is low when SPI is inactive.

trailing *edge is a* ***rising*** *edge.*

leading *edge is a* ***falling*** *edge.*



SPI Communication Modes

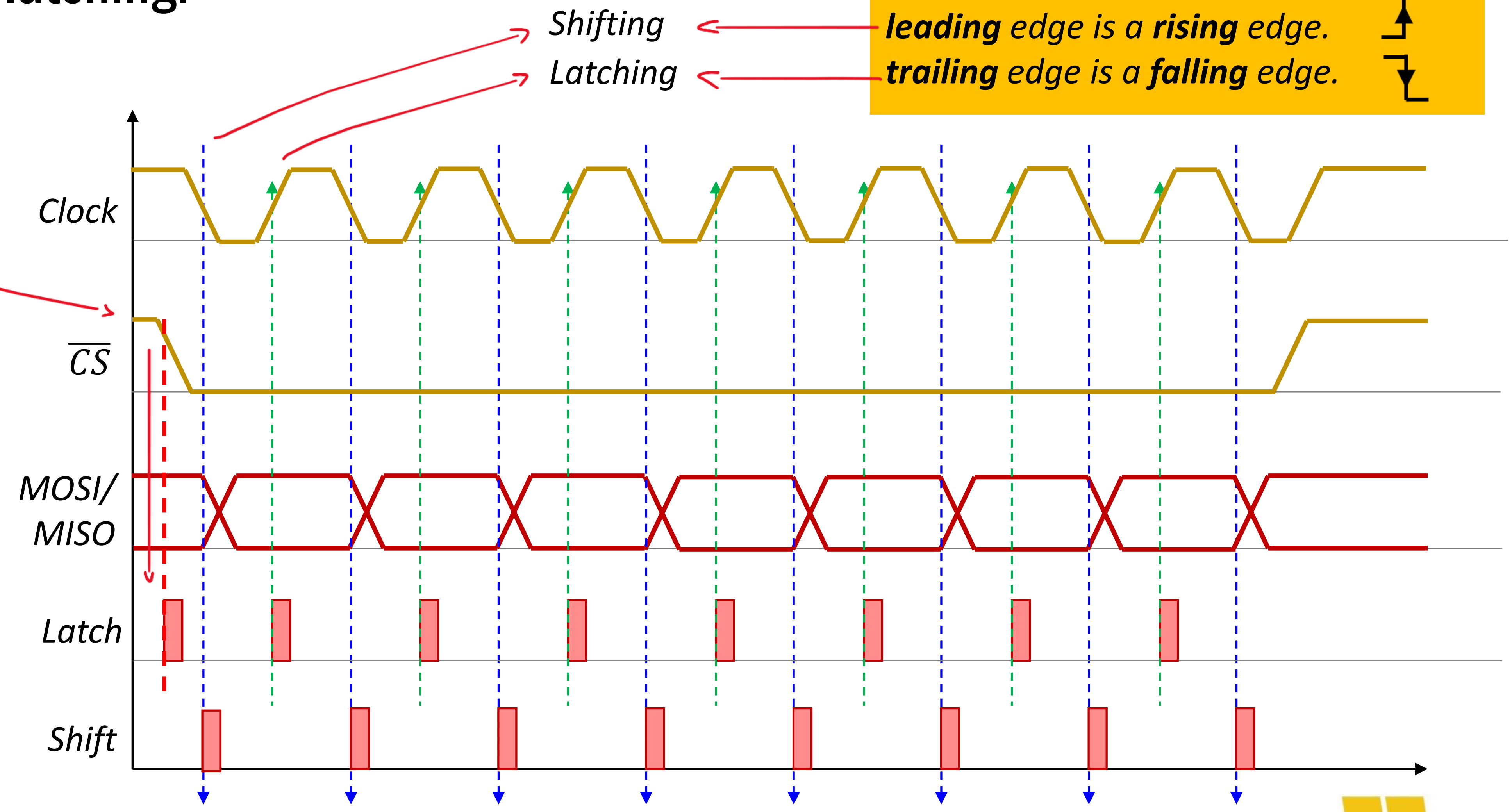
- $\overline{CS}$ causes the first latching.

Clock phase (CKPH)	Clock polarity (CKPL)
0	1

Clock is high when SPI is inactive.

leading edge is a **rising** edge.

trailing edge is a **falling** edge.



SPI Communication Modes

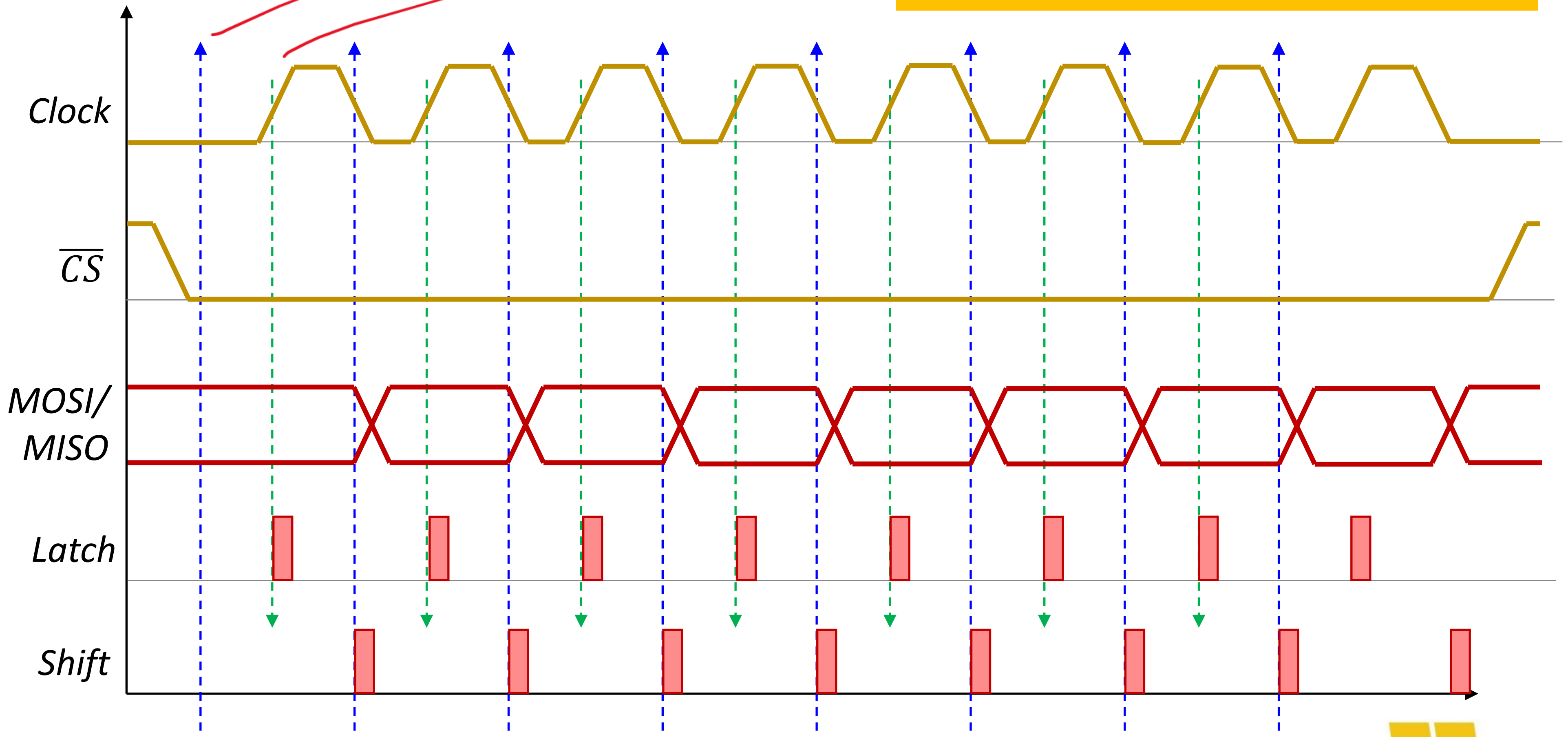
- $\overline{CS}$ has no impact on the first latching.

Shifting ← ***trailing*** *edge is a* ***rising*** *edge.*

Latching ← ***leading*** *edge is a* ***falling*** *edge.*

Clock phase ***(CKPH)***	*Clock polarity* ***(CKPL)***
1	*0*

Clock (phase) is shifted.



SPI Communication Modes

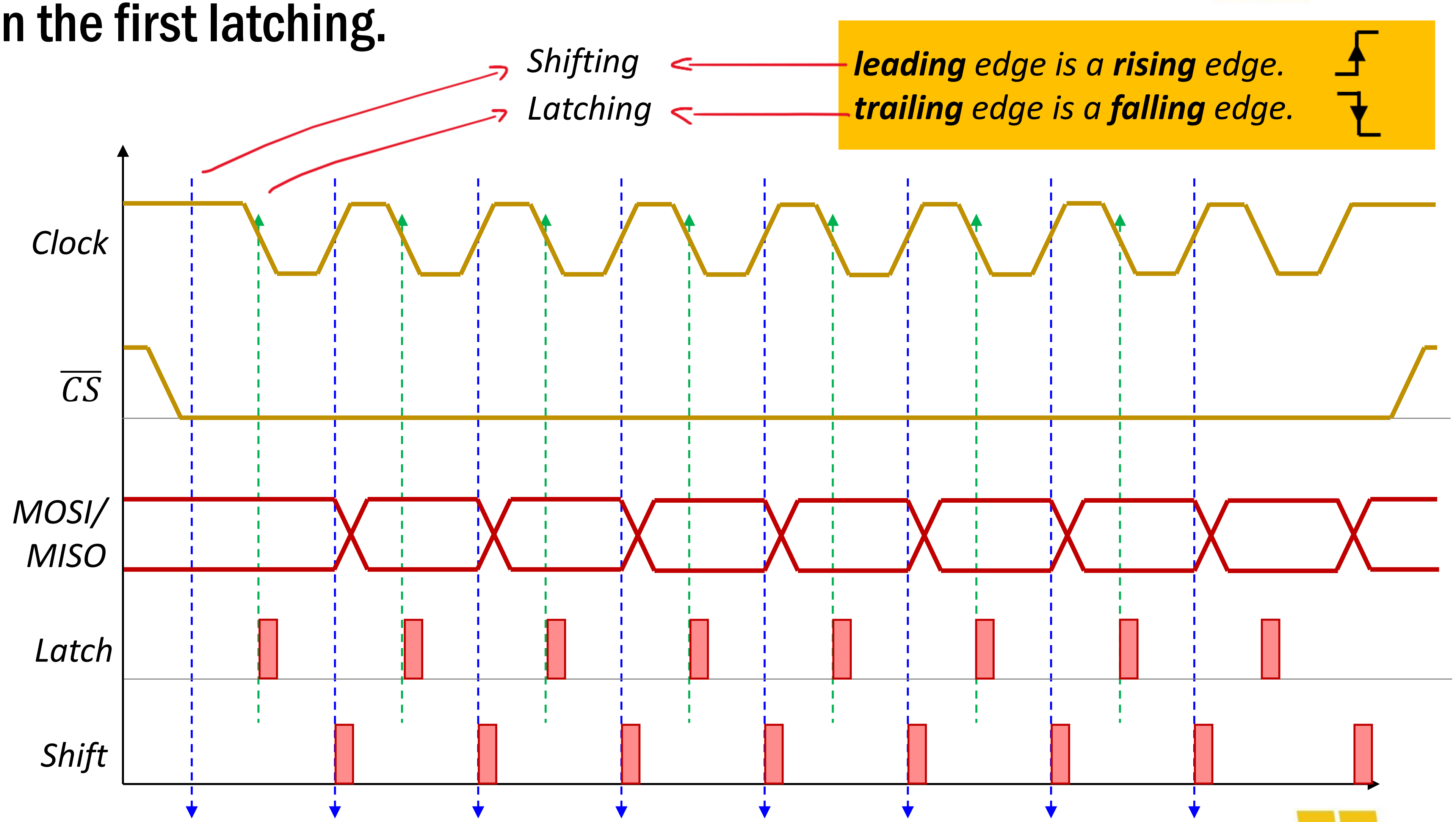
- $\overline{CS}$ has no impact on the first latching.

leading edge is a ***rising*** edge.

trailing edge is a ***falling*** edge.

*Clock phase **(CKPH)***	*Clock polarity **(CKPL)***
1	*1*

Clock (phase) is shifted.



SPI Communication Modes

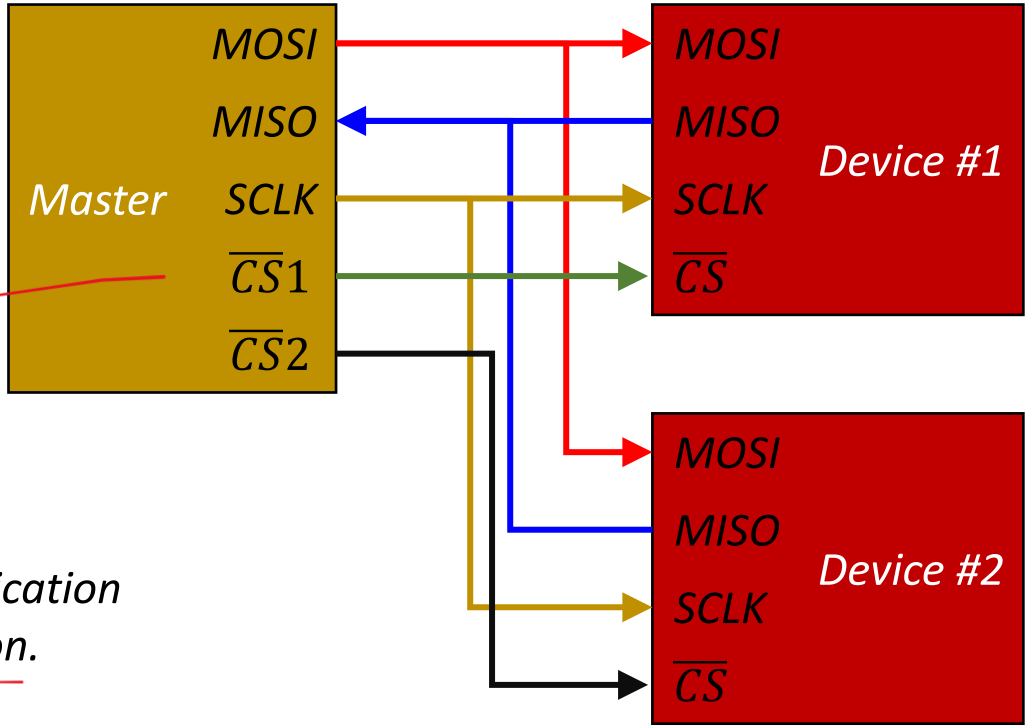
- The polarity and phase of the clock signal
  - can be modified using the CKPL and CKPL bits resulting in 4 different modes.

	Clock Phase	Clock Polarity
Mode 0	*0*	*0*
Mode 1	*0*	*1*
Mode 2	*1*	*0*
Mode 3	*1*	*1*

The SPI protocol is ***not a standard****.*
The modes and the definition of clock phase and clock polarity
varies from device to device. ***Always check the documentation.***

SPI Communication – Multiple Devices

- Master controls the clock!
- Master controls the receiving slave
  - No addressing mechanism
  - Through CS signaling (Master disables/enables)

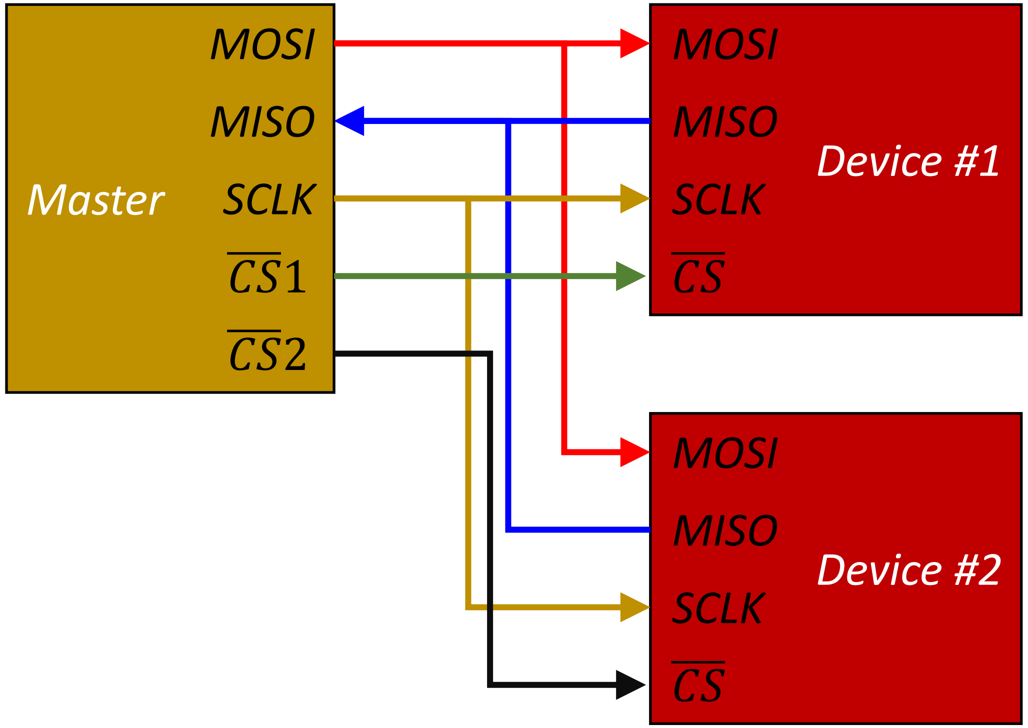


If $\overline{CS}1$ is low, then the device #1 is selected for communication and device #2 must not interfere with the communication.

SPI Communication – Multiple Devices

- Master controls the clock!
- Master controls the receiving slave

(Q) How many pins are required at the master side to support n slaves?



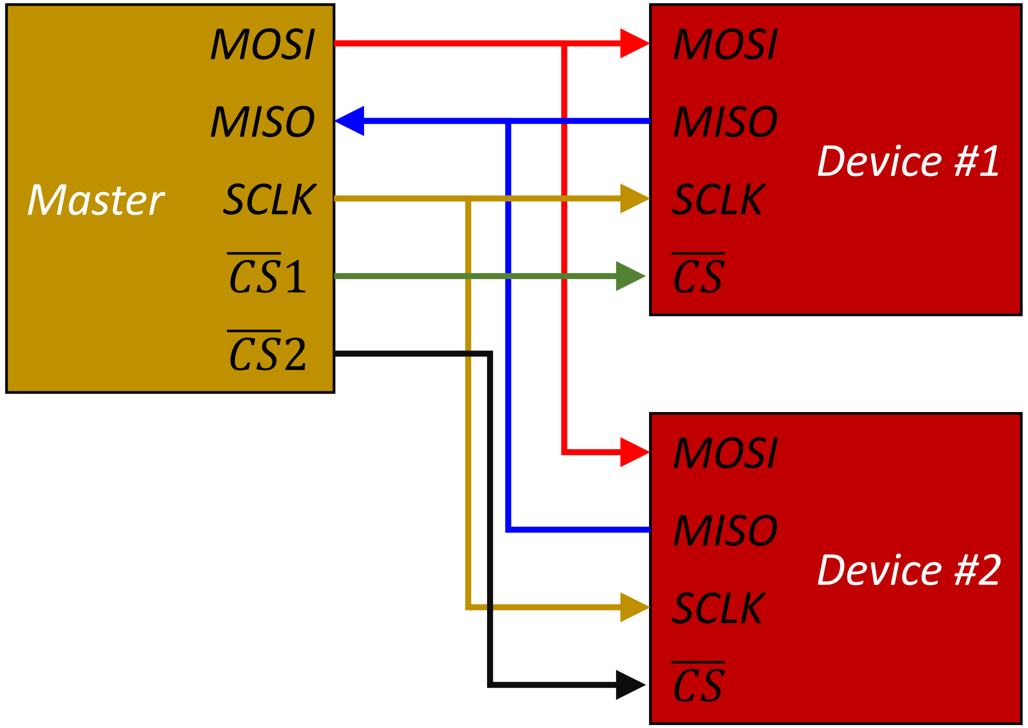
SPI Communication – Multiple Devices

- Master controls the clock!
- Master controls the receiving slave

(Q) How many pins are required at the master side to support n slaves?

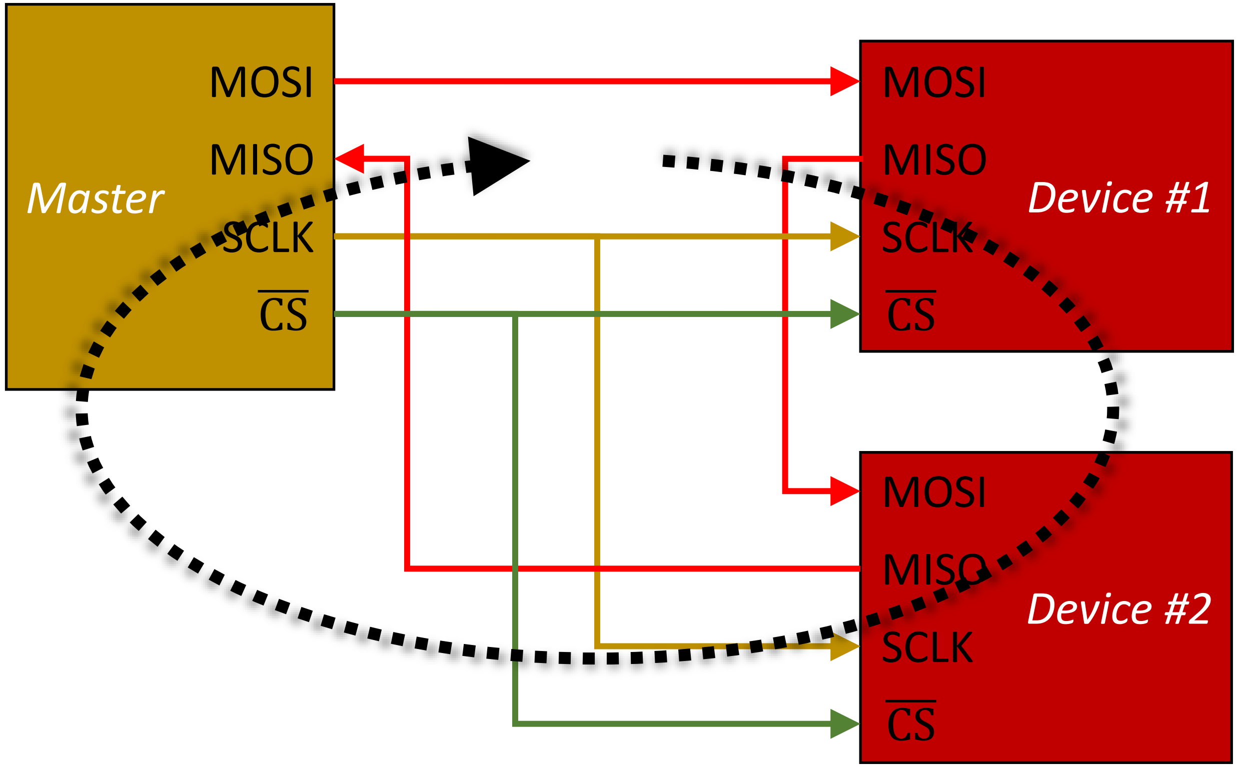
(A) Each slave device with one dedicated CS → n pins!

Not very efficient for the master!

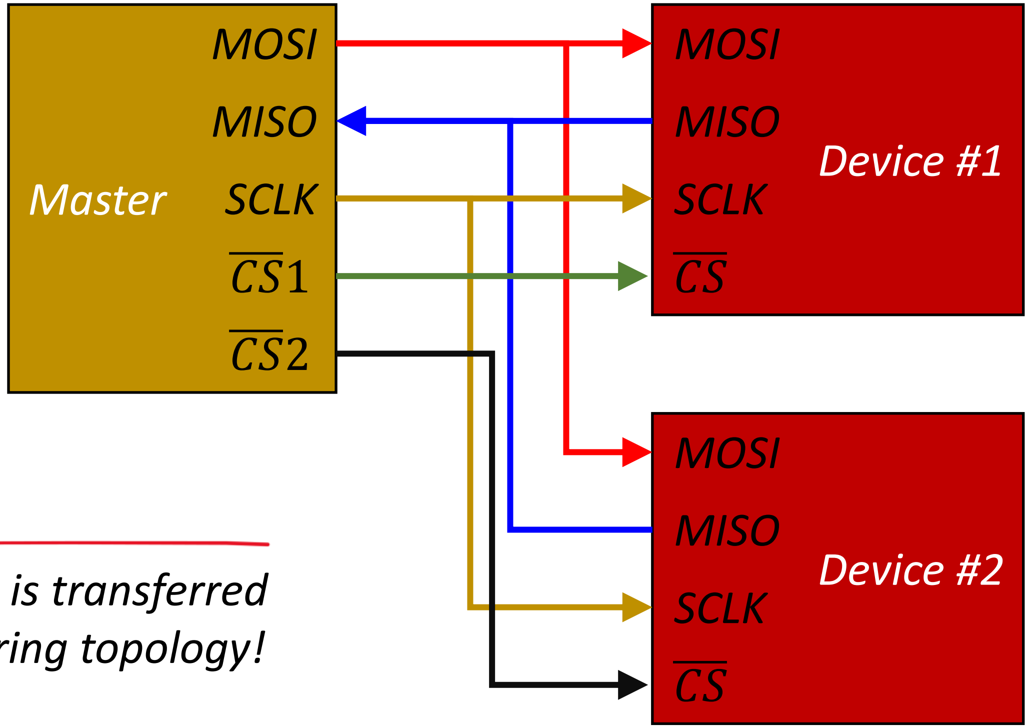


SPI Communication using Daisy Chaining

- Use of Daisy chaining architecture

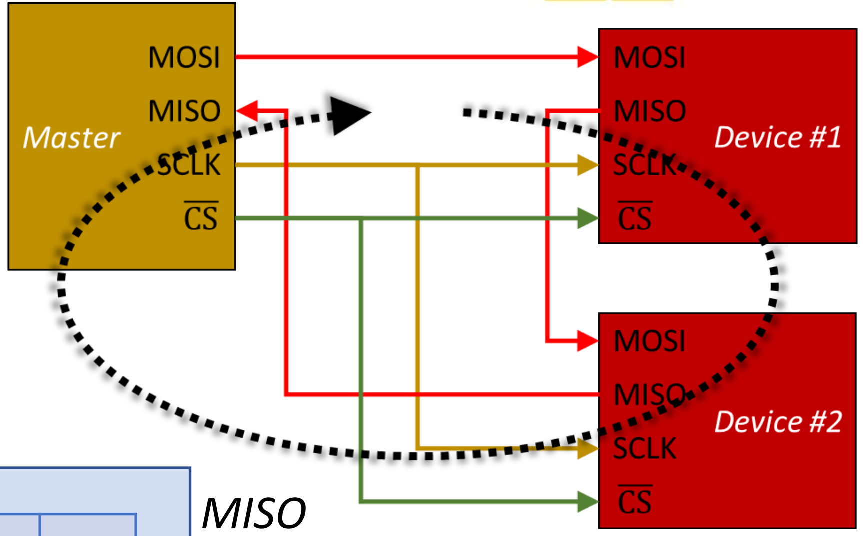


data is transferred in a ring topology!

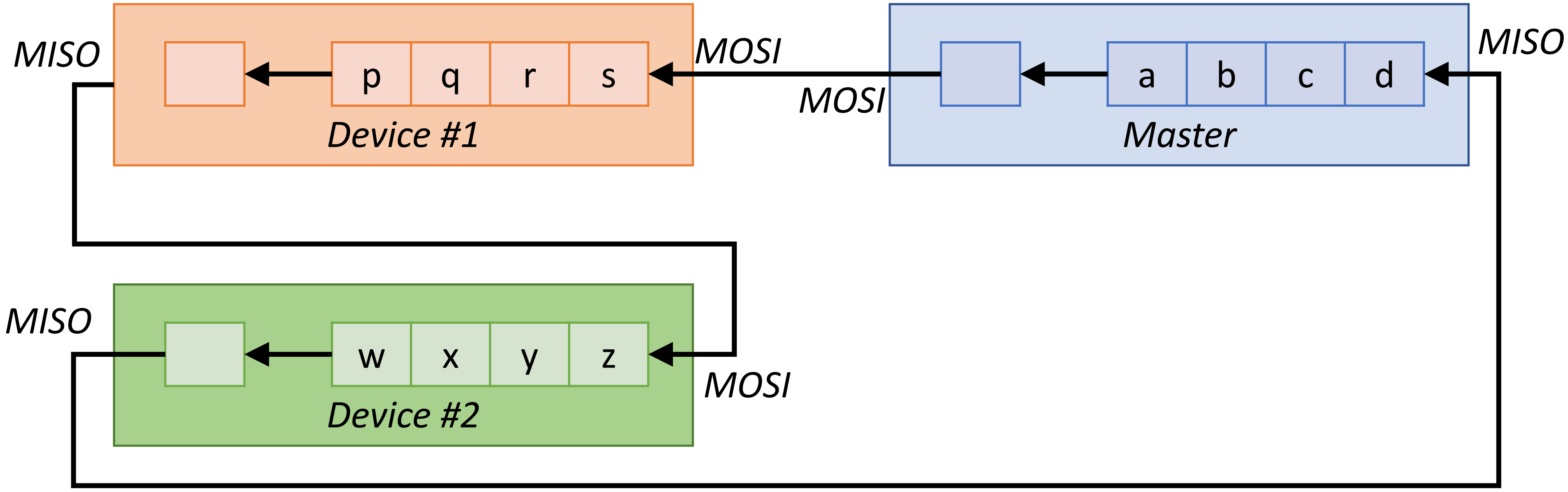


SPI Communication using Daisy Chaining

- Use of Daisy chaining architecture

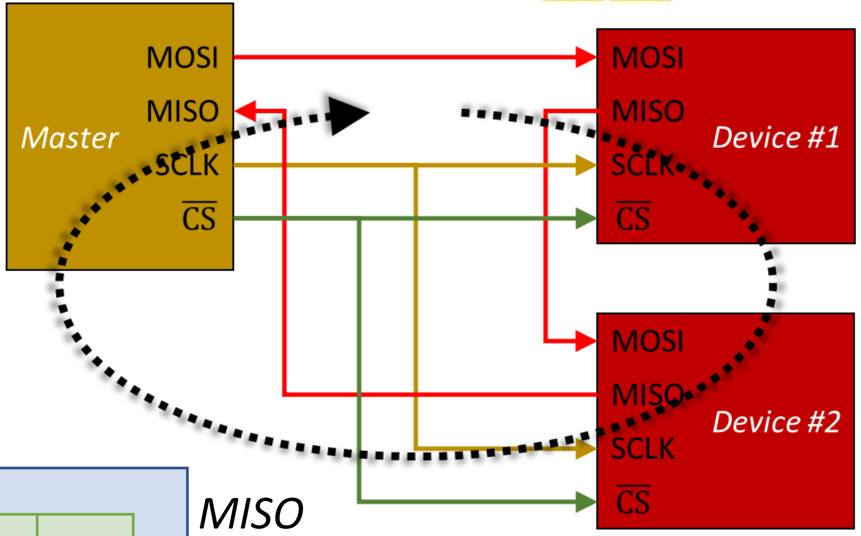


Example of daisy chaining with 4 bits

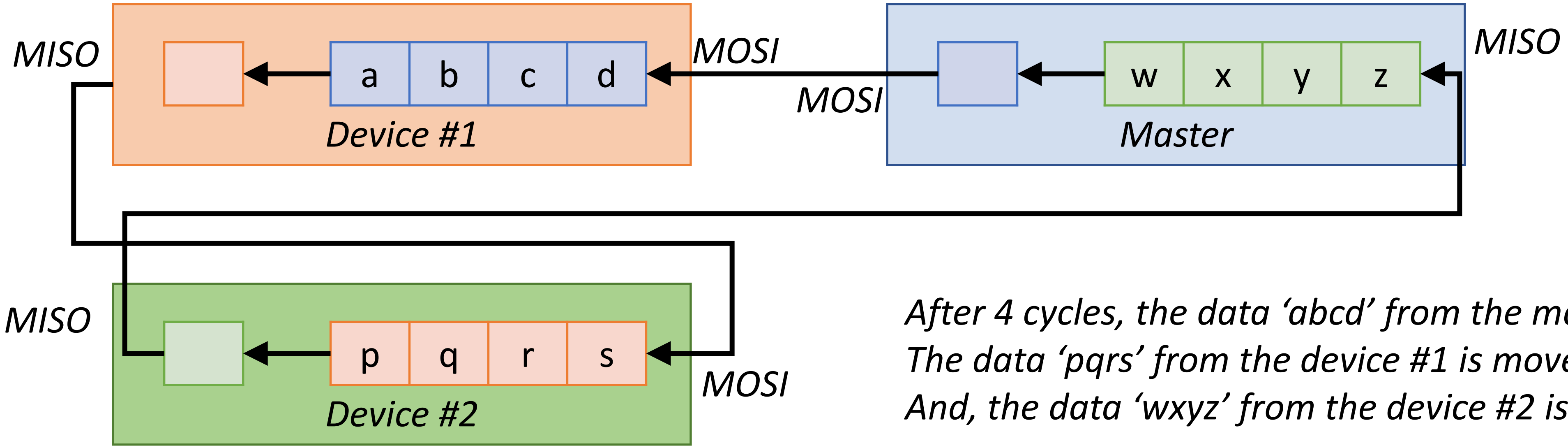


SPI Communication using Daisy Chaining

- Use of Daisy chaining architecture



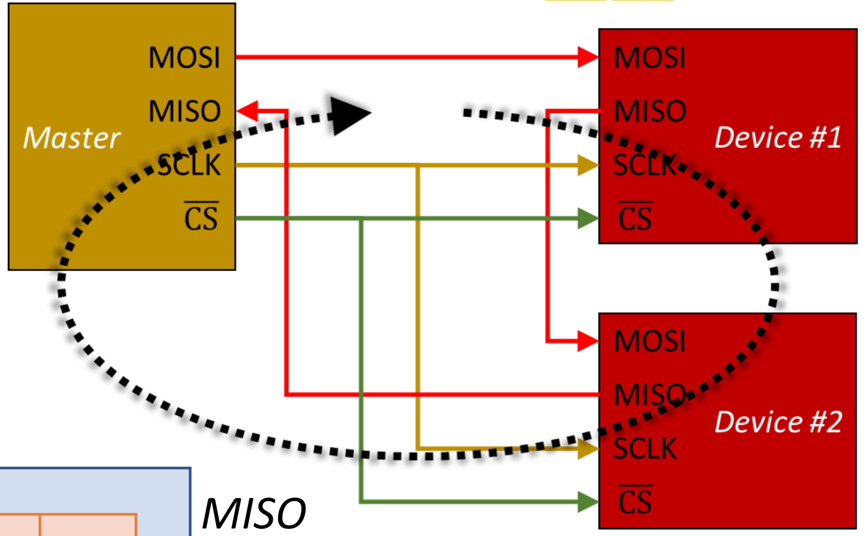
*Example of daisy chaining with 4 bits – **after 4 clock cycles***



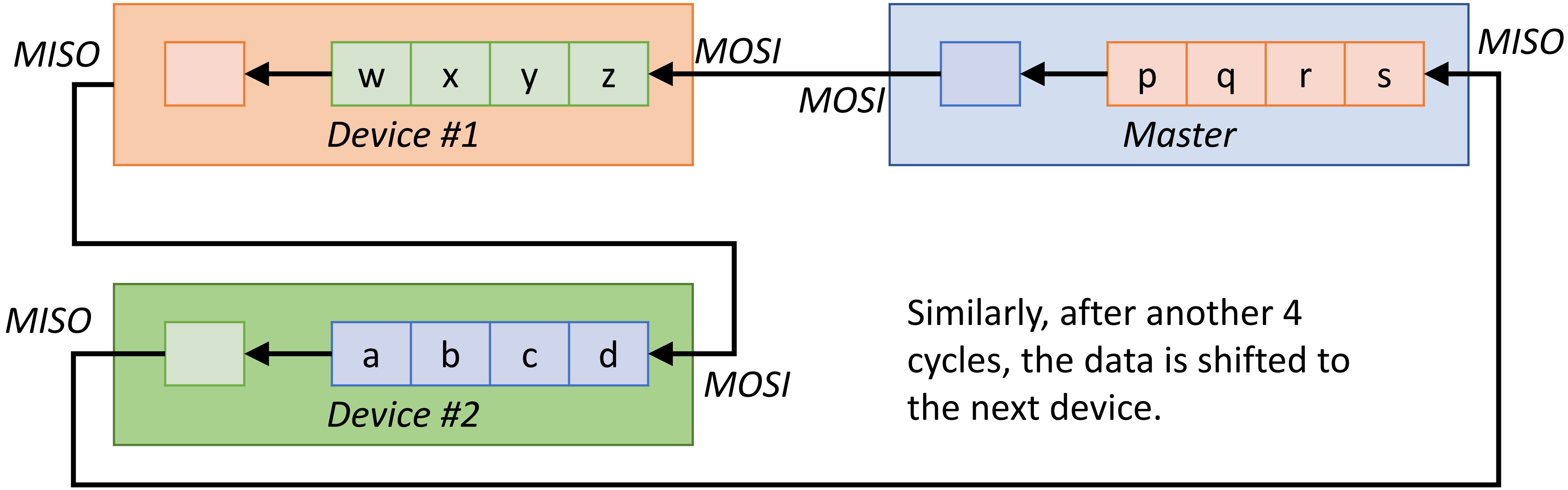
After 4 cycles, the data 'abcd' from the master is moved to device #1.
The data 'pqrs' from the device #1 is moved to device #2.
And, the data 'wxyz' from the device #2 is moved to the master.

SPI Communication using Daisy Chaining

- Use of Daisy chaining architecture



*Example of daisy chaining with 4 bits – **after 4 clock cycles***



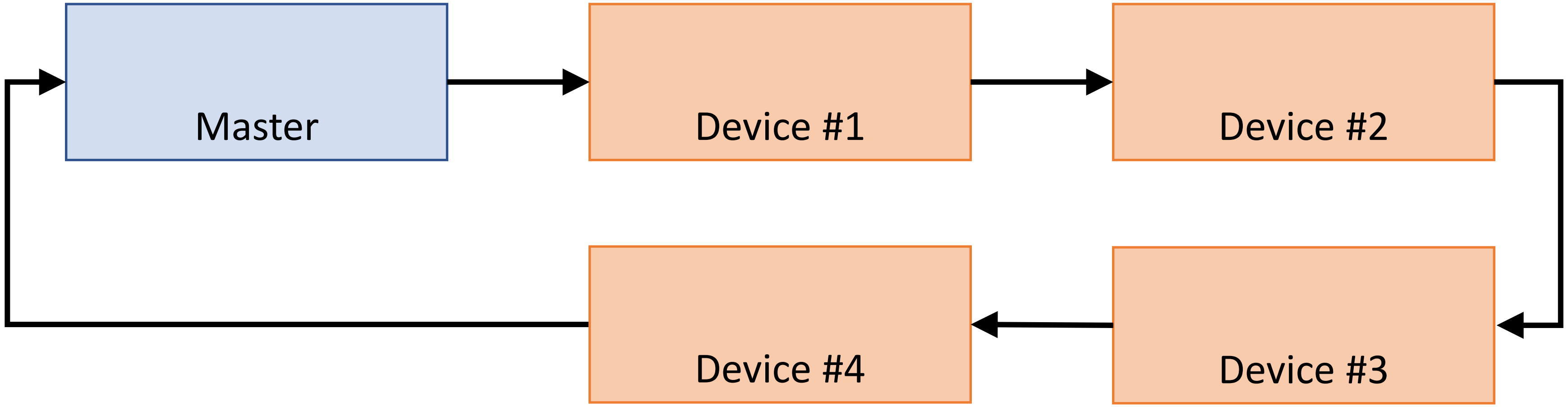
Similarly, after another 4 cycles, the data is shifted to the next device.

SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?

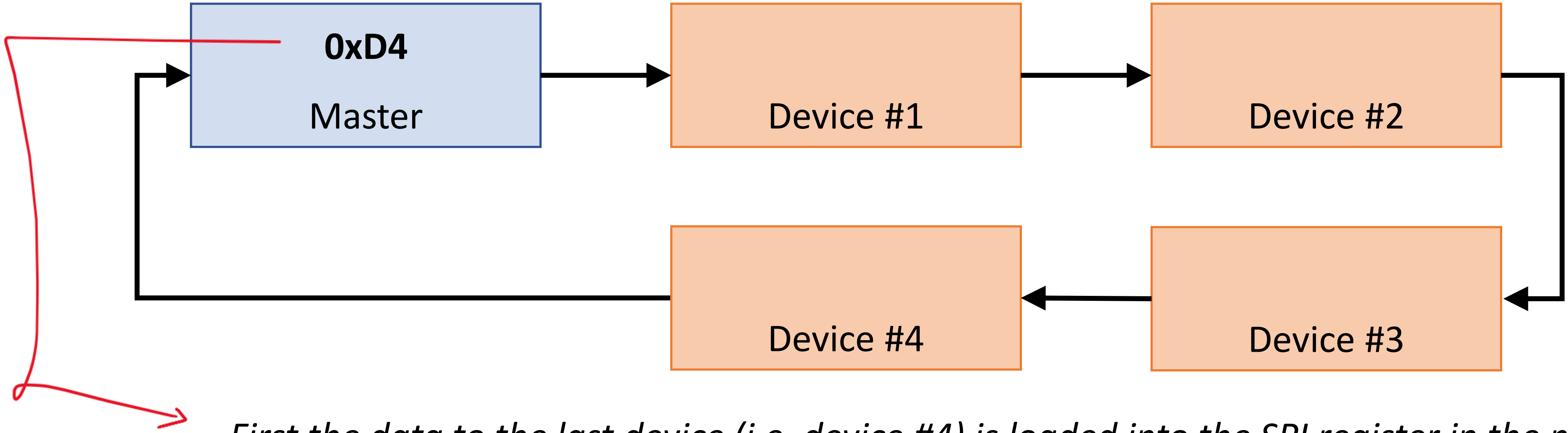


SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?



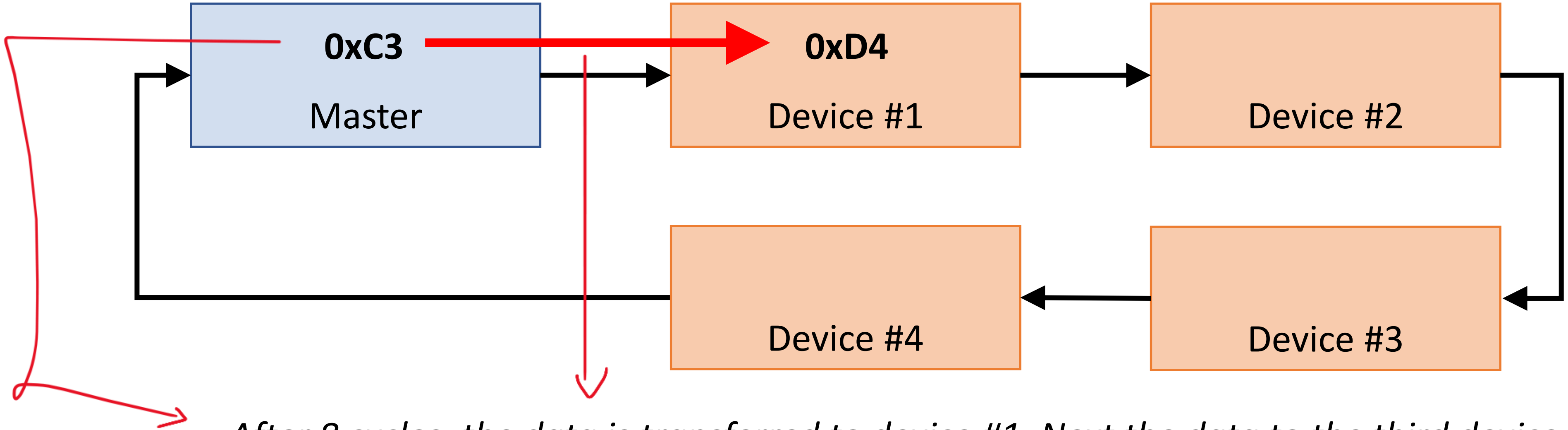
First the data to the last device (i.e. device #4) is loaded into the SPI register in the master.

SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?



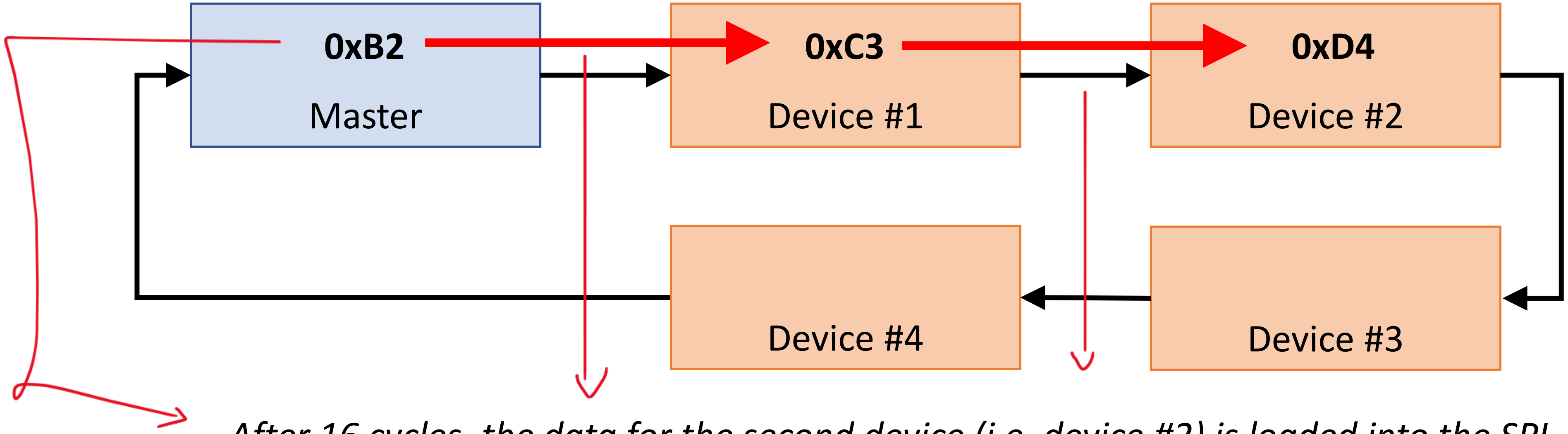
After 8 cycles, the data is transferred to device #1. Next the data to the third device (i.e. device #3) is loaded into the SPI register in the master.

SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?



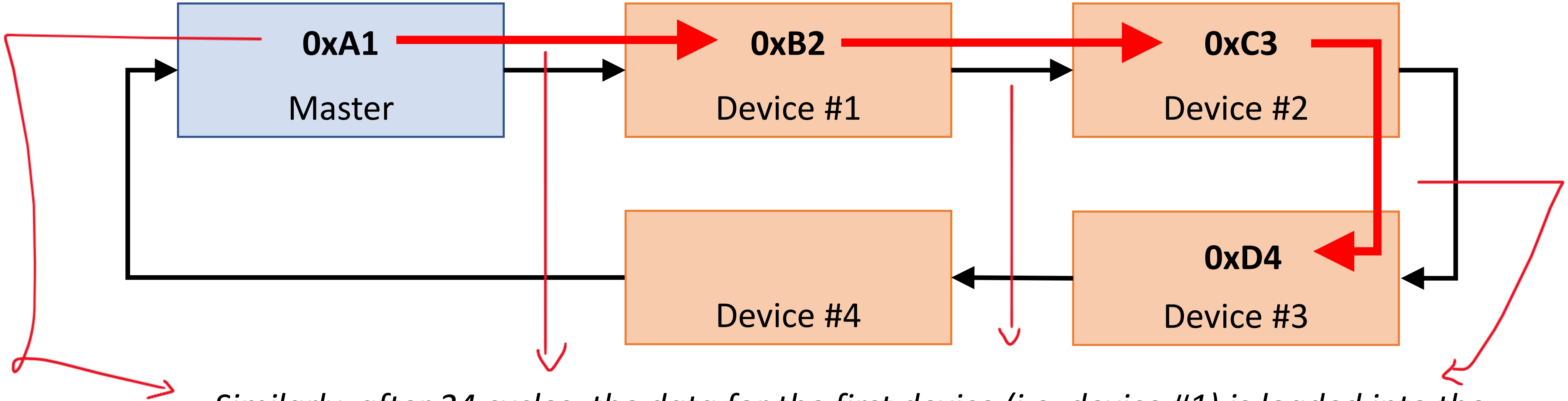
After 16 cycles, the data for the second device (i.e. device #2) is loaded into the SPI register in the master.

SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?



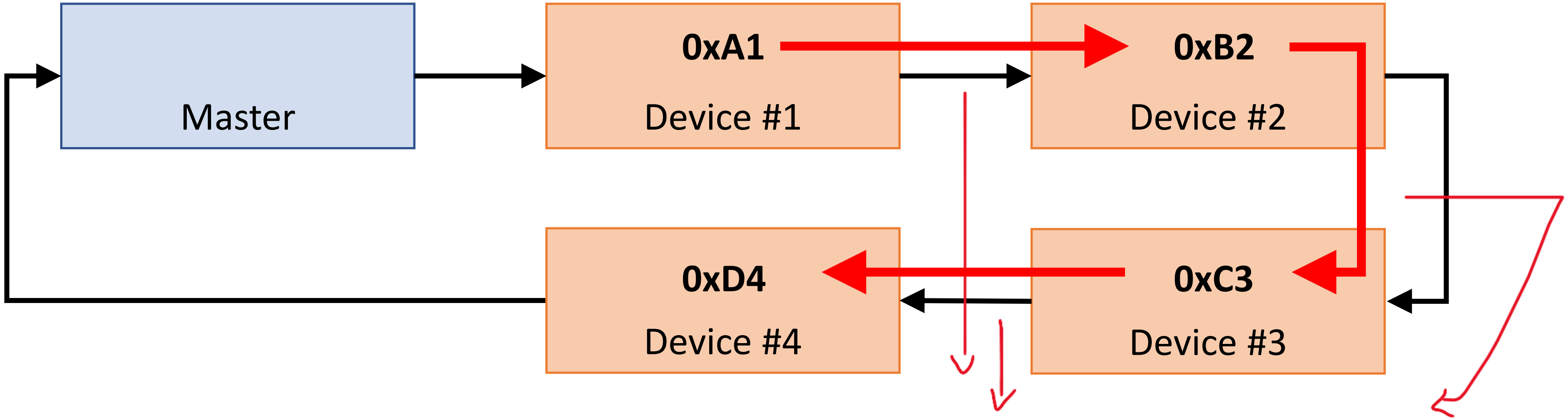
Similarly, after 24 cycles, the data for the first device (i.e. device #1) is loaded into the SPI register in the master.

SPI Communication using Daisy Chaining

- An example of SPI Communication
  - The SPI master wants to send 0xA1 to device #1
  - The SPI master wants to send 0xB2 to device #2
  - The SPI master wants to send 0xC3 to device #3
  - The SPI master wants to send 0xD4 to device #4

How is the communication done?

How many clock cycles will it take for the communication to complete?



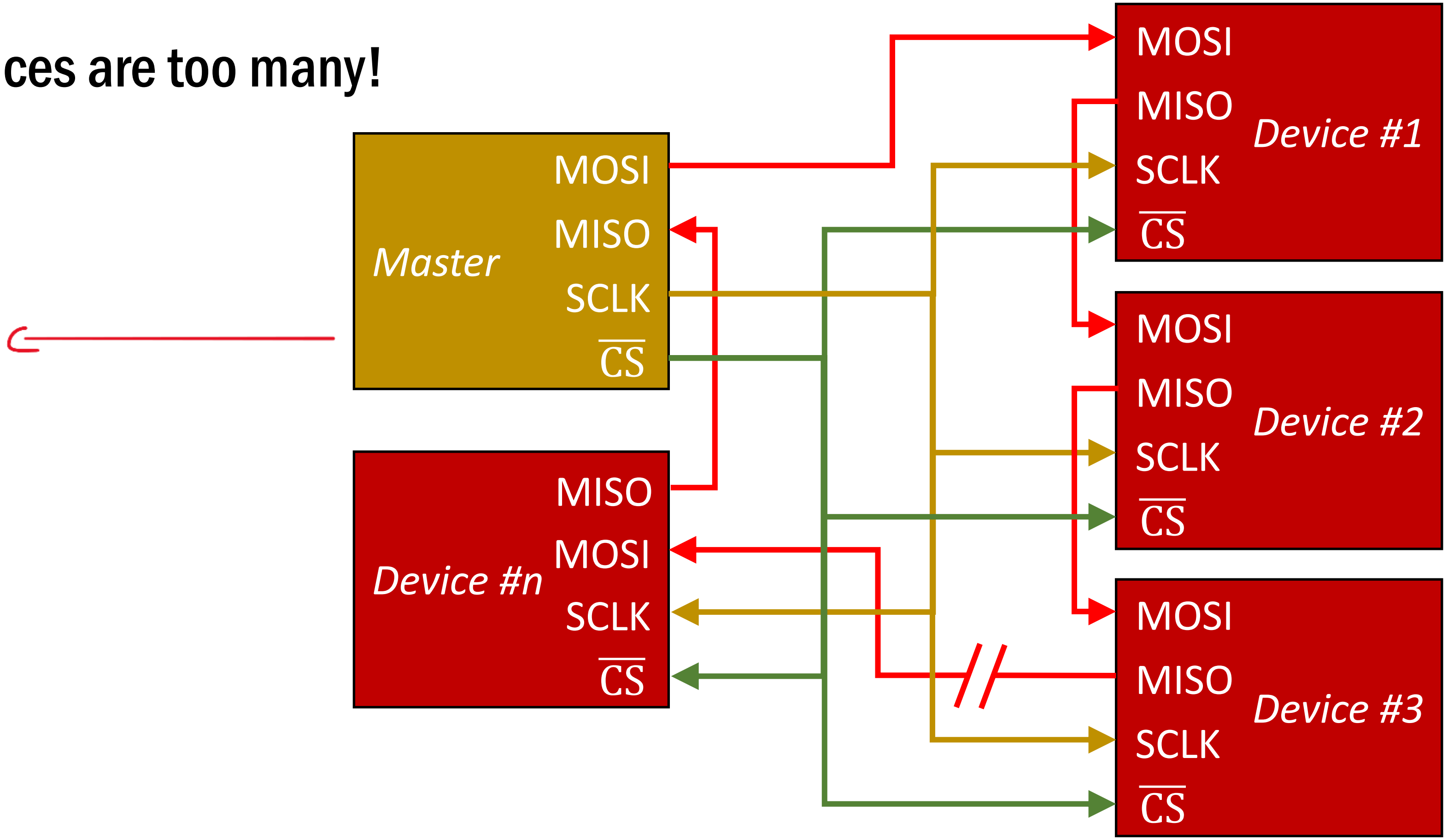
After 32 cycles, the data is sent to each of the 4 devices.

SPI Communication using Daisy Chaining

- Daisy chain is good for minimizing the number of pins
  - Not really good w.r.t. the delay
  - Particularly if the number of devices are too many!

Depending on the order of devices
*→ **Up to (n).(bw)***

n: number of devices
bw: size of the data



eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols
    - e.g., UART
    - <u>**e.g., serial peripheral interface (SPI)**</u>
    - e.g., inter-integrated Circuit (I2C)
- In MSP430FR6989
  - There are two implementations of eUSCI_A
    - eUSCI_A0 and eUSCI_A1
  - There are two implementations of eUSCI_B
    - eUSCI_B0 and eUSCI_B1

eUSCI_A supports

- *UART*
- ***SPI***

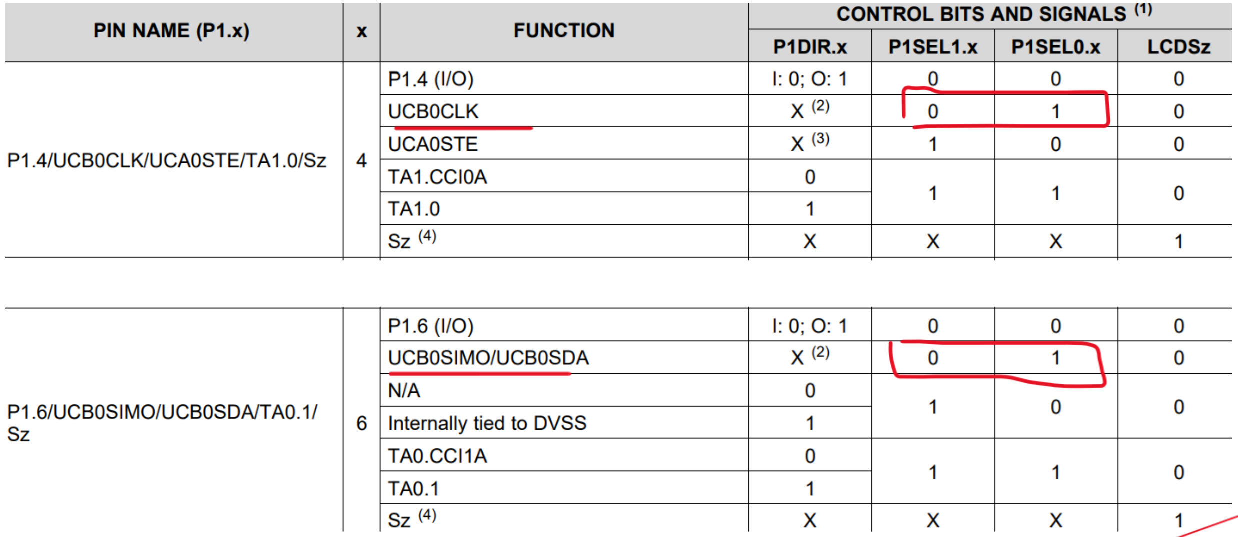
eUSCI_B supports

- *I2C*
- ***SPI***

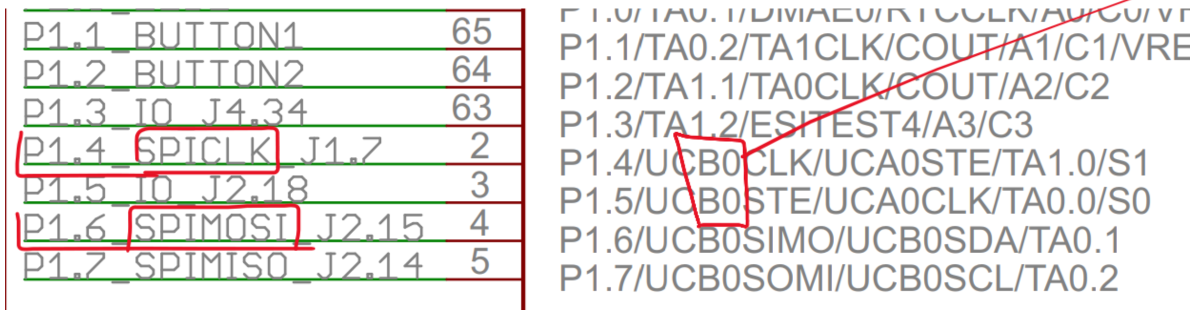
*The boosterpack SPI bus is connected to the eUSCI_B module via **pins P1.4** and **P1.6**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols



PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS [1]			
			P1DIR.x	P1SEL1.x	P1SEL0.x	LCDSz
P1.4/UCB0CLK/UCA0STE/TA1.0/Sz	4	P1.4 (I/O)	I: 0; O: 1	0	0	0
		UCB0CLK	X [2]	0	1	0
		UCA0STE	X [3]	1	0	0
		TA1.CCI0A	0	1	1	0
		TA1.0	1			
		Sz [4]	X	X	X	1
P1.6/UCB0SIMO/UCB0SDA/TA0.1/Sz	6	P1.6 (I/O)	I: 0; O: 1	0	0	0
		UCB0SIMO/UCB0SDA	X [2]	0	1	0
		N/A	0	1	0	0
		Internally tied to DVSS	1			
		TA0.CCI1A	0	1	1	0
		TA0.1	1			
		Sz [4]	X	X	X	1



		P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VF
P1.1_BUTTON1	65	P1.1/TA0.2/TA1CLK/COUT/A1/C1/VRE
P1.2_BUTTON2	64	P1.2/TA1.1/TA0CLK/COUT/A2/C2
P1.3_IO_J4.34	63	P1.3/TA1.2/ESITEST4/A3/C3
P1.4_SPICLK_J1.7	2	P1.4/UCB0CLK/UCA0STE/TA1.0/S1
P1.5_IO_J2.18	3	P1.5/UCB0STE/UCA0CLK/TA0.0/S0
P1.6_SPIMOSI_J2.15	4	P1.6/UCB0SIMO/UCB0SDA/TA0.1
P1.7_SPIMISO_J2.14	5	P1.7/UCB0SOMI/UCB0SCL/TA0.2

eUSCI_A supports

- *UART*
- ***SPI***

eUSCI_B supports

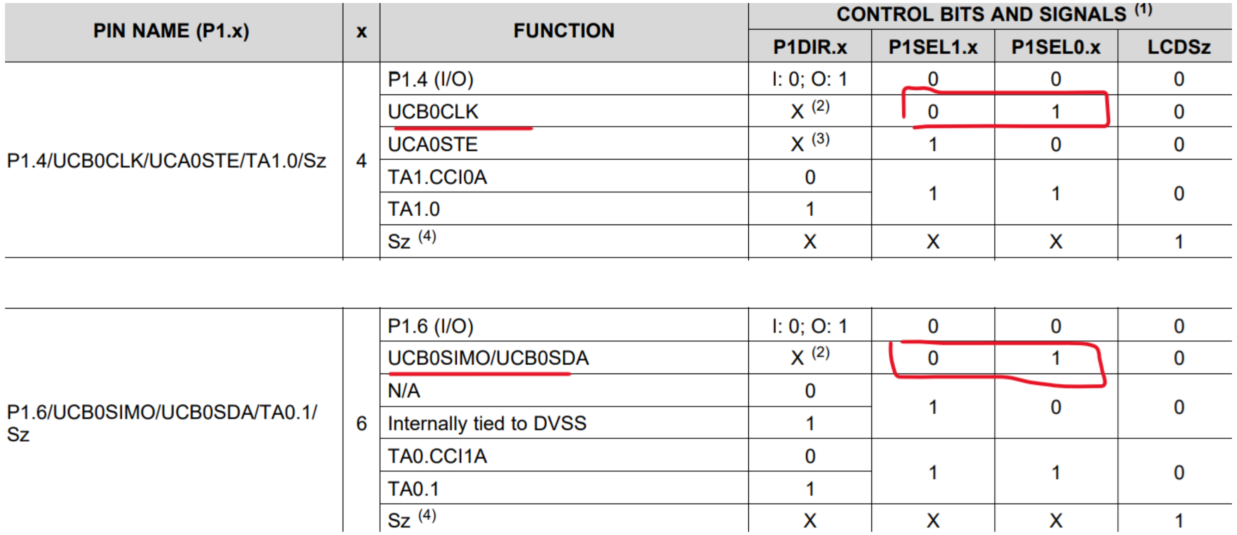
- *I2C*
- ***SPI***

We are using eUSCI_B here!

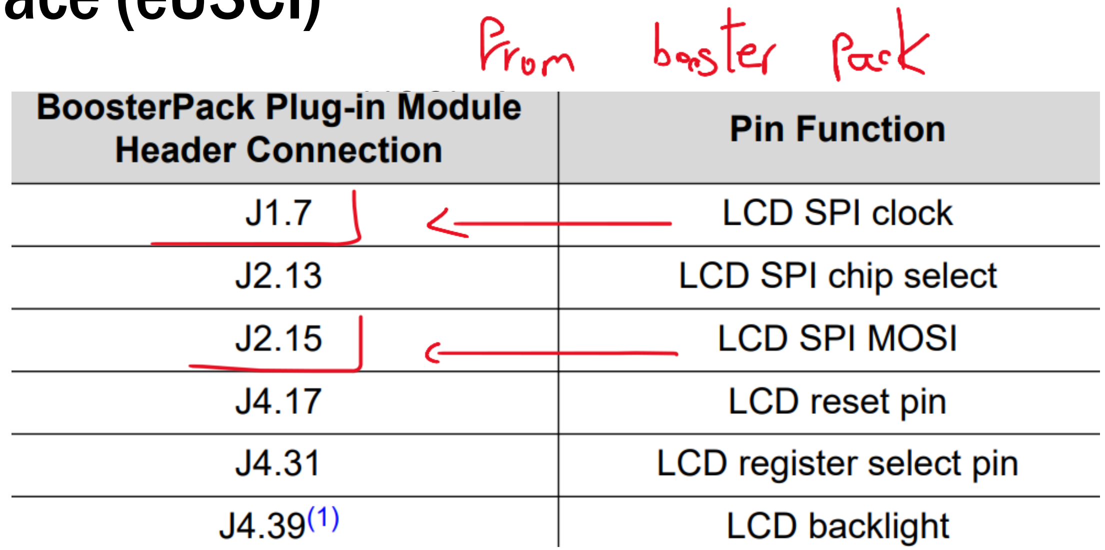
*The boosterpack SPI bus is connected to the eUSCI_A module via **pins P1.4** and **P1.6**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols

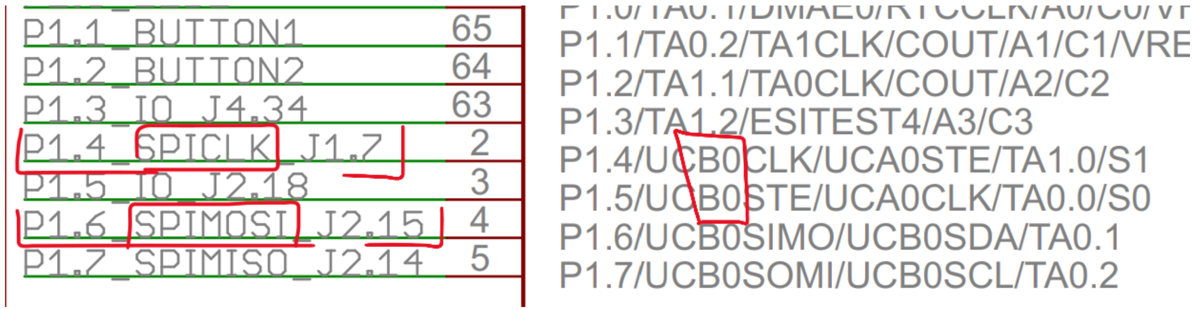


PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS (1)			
			P1DIR.x	P1SEL1.x	P1SEL0.x	LCDSz
P1.4/UCB0CLK/UCA0STE/TA1.0/Sz	4	P1.4 (I/O)	I: 0; O: 1	0	0	0
		UCB0CLK	X (2)	0	1	0
		UCA0STE	X (3)	1	0	0
		TA1.CCI0A	0	1	1	0
		TA1.0	1			
		Sz (4)	X	X	X	1
P1.6/UCB0SIMO/UCB0SDA/TA0.1/ Sz	6	P1.6 (I/O)	I: 0; O: 1	0	0	0
		UCB0SIMO/UCB0SDA	X (2)	0	1	0
		N/A	0	1	0	0
		Internally tied to DVSS	1			
		TA0.CCI1A	0	1	1	0
		TA0.1	1			
		Sz (4)	X	X	X	1



From boster Pack

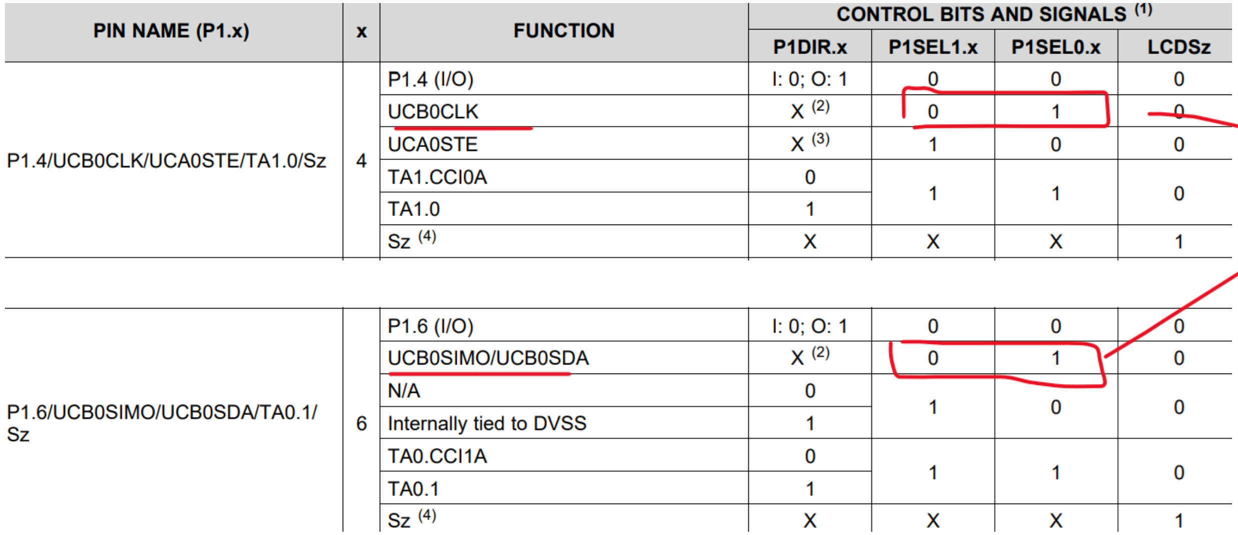
BoosterPack Plug-in Module Header Connection	Pin Function
J1.7	LCD SPI clock
J2.13	LCD SPI chip select
J2.15	LCD SPI MOSI
J4.17	LCD reset pin
J4.31	LCD register select pin
J4.39(1)	LCD backlight


P1.1_BUTTON1 65
P1.2_BUTTON2 64
P1.3_IO_J4.34 63
P1.4_SPICLK_J1.7 2
P1.5_IO_J2.18 3
P1.6_SPIMOSI_J2.15 4
P1.7_SPIMISO_J2.14 5
P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VF
P1.1/TA0.2/TA1CLK/COUT/A1/C1/VRE
P1.2/TA1.1/TA0CLK/COUT/A2/C2
P1.3/TA1.2/ESITEST4/A3/C3
P1.4/UCB0CLK/UCA0STE/TA1.0/S1
P1.5/UCB0STE/UCA0CLK/TA0.0/S0
P1.6/UCB0SIMO/UCB0SDA/TA0.1
P1.7/UCB0SOMI/UCB0SCL/TA0.2

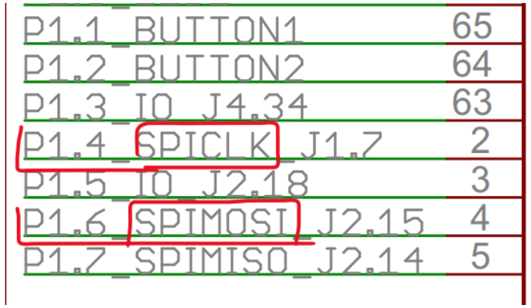
*The boosterpack SPI bus is connected to the eUSCI_A module via **pins P1.4** and **P1.6**.*
Refer to the boosterpack and launchpad user guide for detailed information on all the connected pins.

eUSCI Module - Revisiting

- Enhanced Universal Serial Communication Interface (eUSCI)
  - Supports multiple serial communication protocols

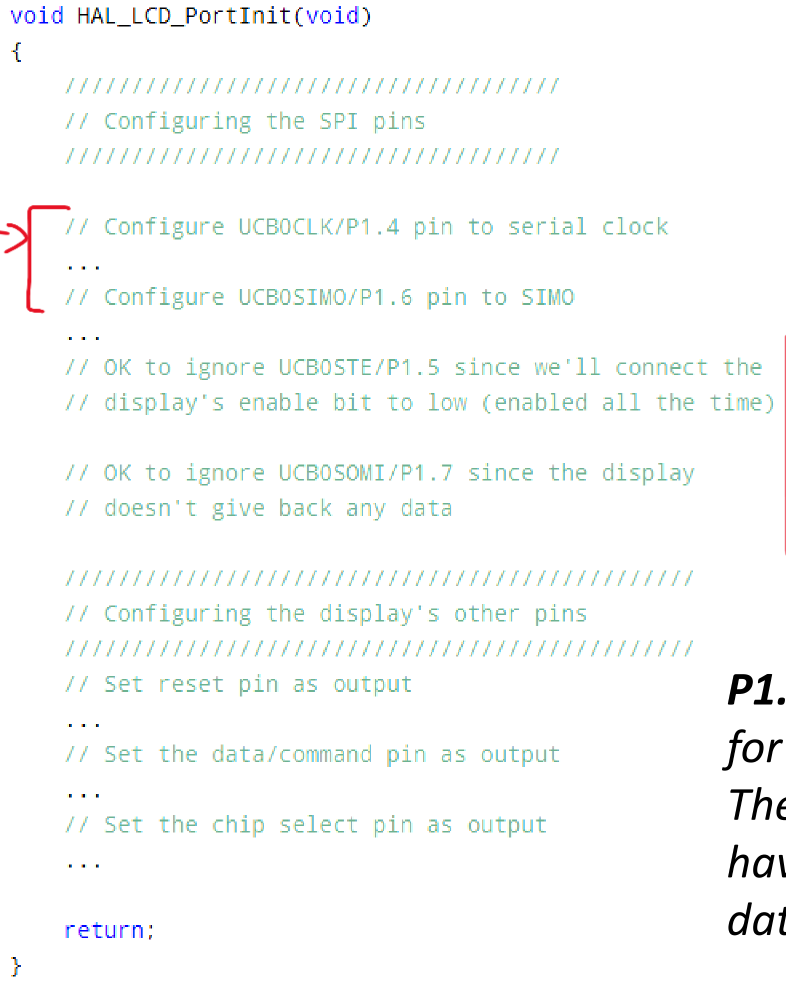


PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS (1)			
			P1DIR.x	P1SEL1.x	P1SEL0.x	LCDSz
P1.4/UCB0CLK/UCA0STE/TA1.0/Sz	4	P1.4 (I/O)	I: 0; O: 1	0	0	0
		UCB0CLK	X (2)	0	1	0
		UCA0STE	X (3)	1	0	0
		TA1.CCI0A	0	1	1	0
		TA1.0	1			
		Sz (4)	X	X	X	1
P1.6/UCB0SIMO/UCB0SDA/TA0.1/Sz	6	P1.6 (I/O)	I: 0; O: 1	0	0	0
		UCB0SIMO/UCB0SDA	X (2)	0	1	0
		N/A	0	1	0	0
		Internally tied to DVSS	1			
		TA0.CCI1A	0	1	1	0
		TA0.1	1			
		Sz (4)	X	X	X	1



P1.1_BUTTON1	65
P1.2_BUTTON2	64
P1.3_IO_J4.34	63
P1.4_SPICLK_J1.7	2
P1.5_IO_J2.18	3
P1.6_SPIMOSI_J2.15	4
P1.7_SPIMISO_J2.14	5

P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VF
P1.1/TA0.2/TA1CLK/COUT/A1/C1/VRE
P1.2/TA1.1/TA0CLK/COUT/A2/C2
P1.3/TA1.2/ESITEST4/A3/C3
P1.4/UCB0CLK/UCA0STE/TA1.0/S1
P1.5/UCB0STE/UCA0CLK/TA0.0/S0
P1.6/UCB0SIMO/UCB0SDA/TA0.1
P1.7/UCB0SOMI/UCB0SCL/TA0.2



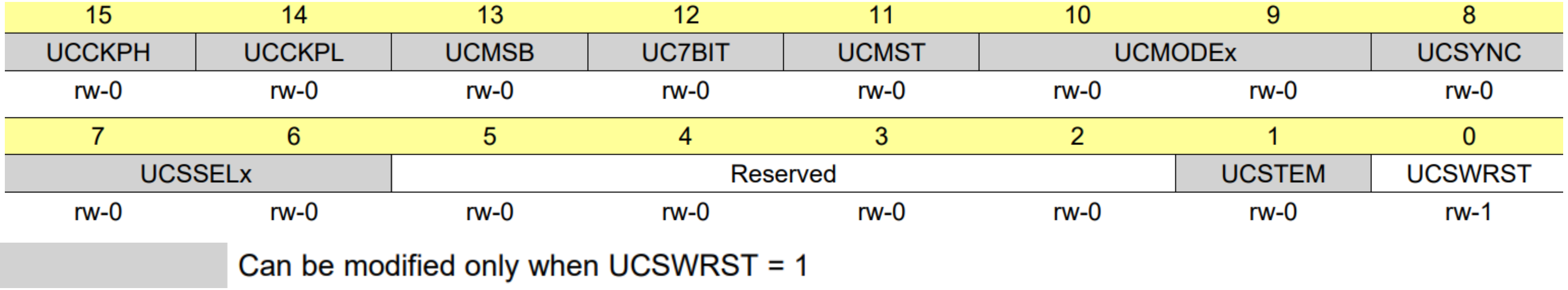
```
void HAL_LCD_PortInit(void)
{
    /////////////////////////////////////
    // Configuring the SPI pins
    /////////////////////////////////////

    // Configure UCB0CLK/P1.4 pin to serial clock
    ...
    // Configure UCB0SIMO/P1.6 pin to SIMO
    ...
    // OK to ignore UCB0STE/P1.5 since we'll connect the
    // display's enable bit to low (enabled all the time)

    // OK to ignore UCB0SOMI/P1.7 since the display
    // doesn't give back any data

    /////////////////////////////////////////////////
    // Configuring the display's other pins
    /////////////////////////////////////////////////
    // Set reset pin as output
    ...
    // Set the data/command pin as output
    ...
    // Set the chip select pin as output
    ...

    return;
}
```

***P1.5** and **P1.7** are for return data! The slave might have no return data*

SPI Registers

- eUSCI_A control register 0

UCAxCTLW0



15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1

- UCCKPH
  - Determining the clock phase select →

0: shifted on the first edge and latched on the following edge

1: latched on the first edge and shifted on the following edge

- UCCKPL
  - Determining the clock polarity select →

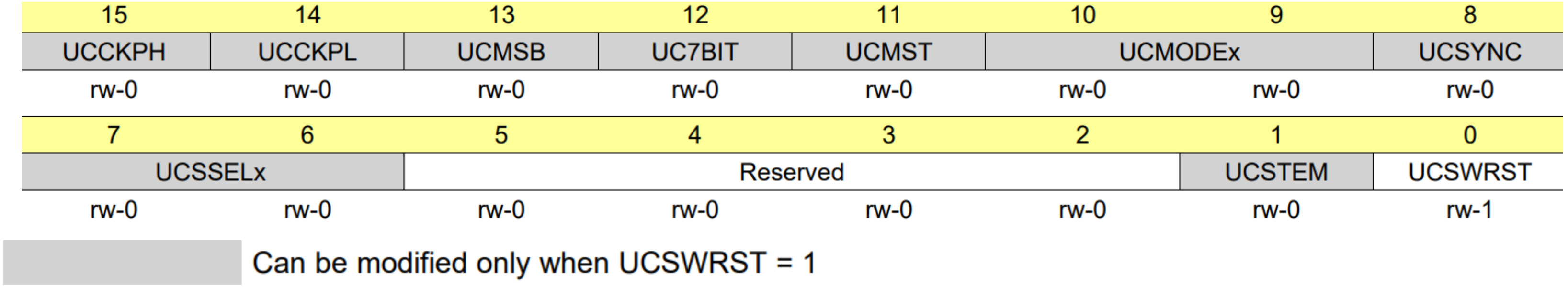
0: the inactive state is low

1: the inactive state is high

SPI Registers

- eUSCI_A control register 0

UCAxCTLW0



15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1

- UCMSB
  - Determining the order of the shift register

0: LSB first

1: MSB first

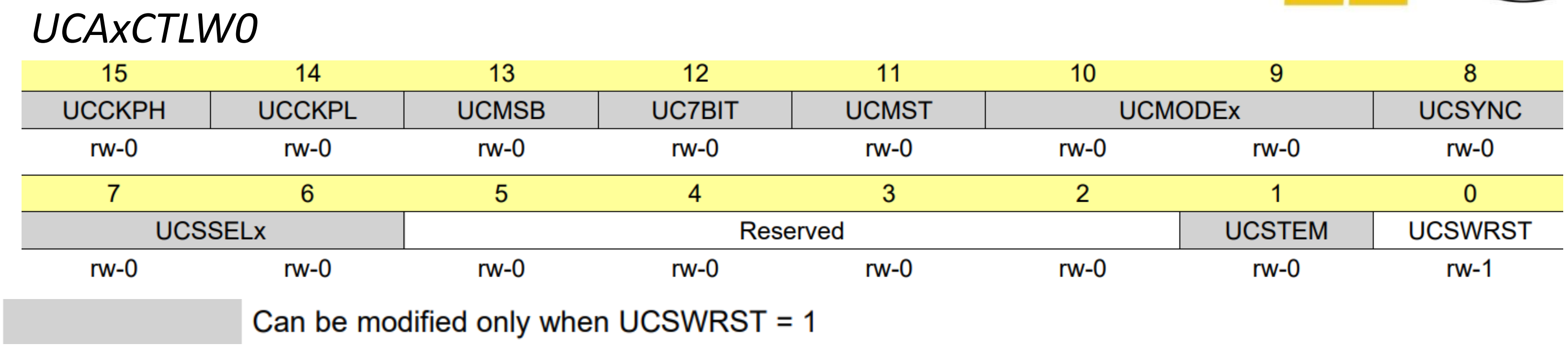
- UC7BIT
  - Determining the character length

0: 8-bit data

1: 7-bit data

SPI Registers

- eUSCI_A control register 0



UCAxCTLW0

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1

- UCMST
  - Determining master mode

0: slave mode

1: master mode

- UCMODEx
  - Determining the synchronous mode

00: 3-bit SPI

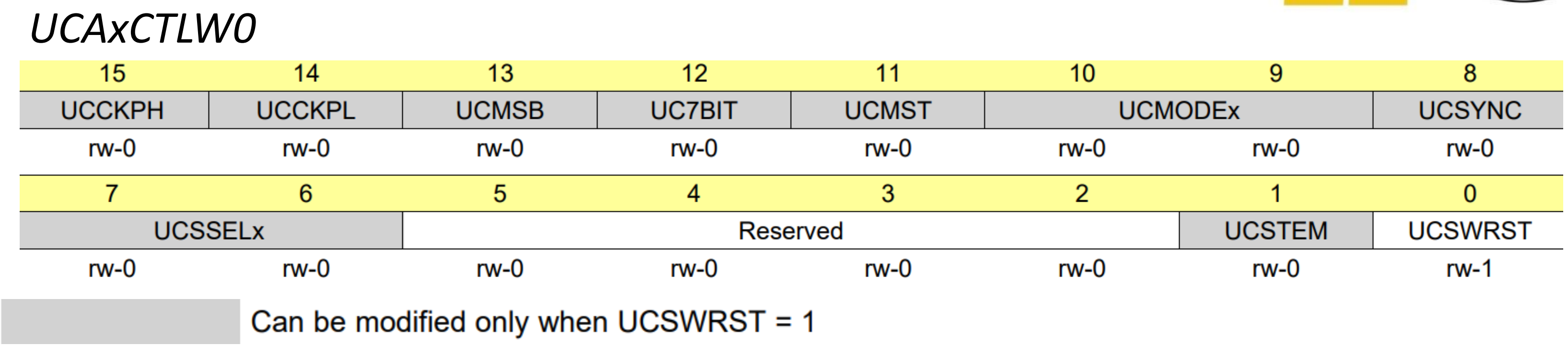
01: 4-bit SPI with UCxSTE active high

10: 4-bit SPI with UCxSTE active low

Slave Transfer En

SPI Registers

- eUSCI_A control register 0



UCAxCTLW0

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1

- UCSSELx
  - Determining the clock source

00: UCxCLK in slave mode
01: ACLK in master mode
11: SMCLK in master mode

- UCWRST
  - Determining the reset mode

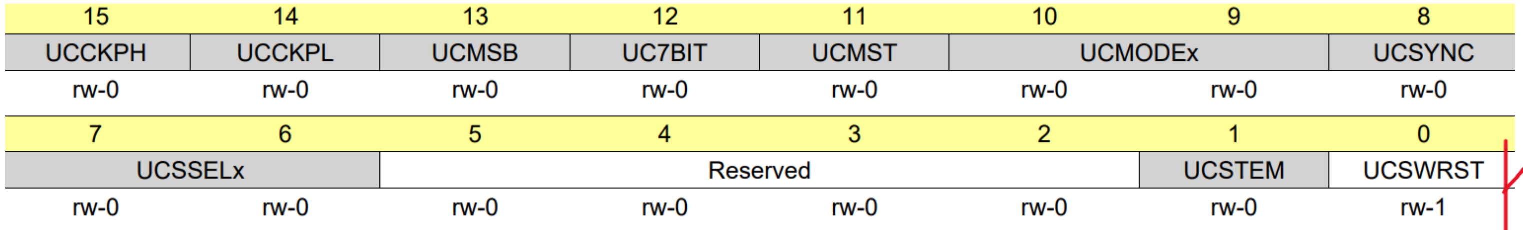
0: reset is not active

1: reset is active

SPI Registers

- **eUSCI_A control register 0**

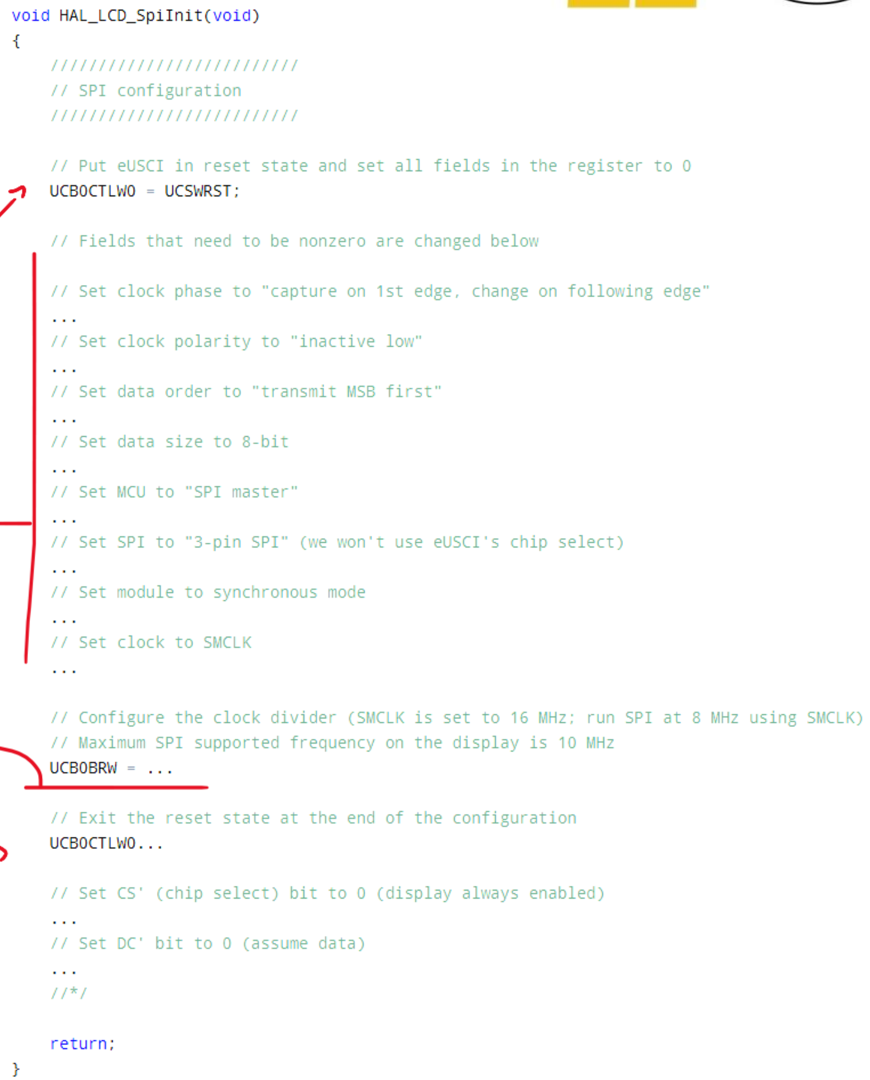
UCAxCTLW0



15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCSSELx		Reserved				UCSTEM	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1

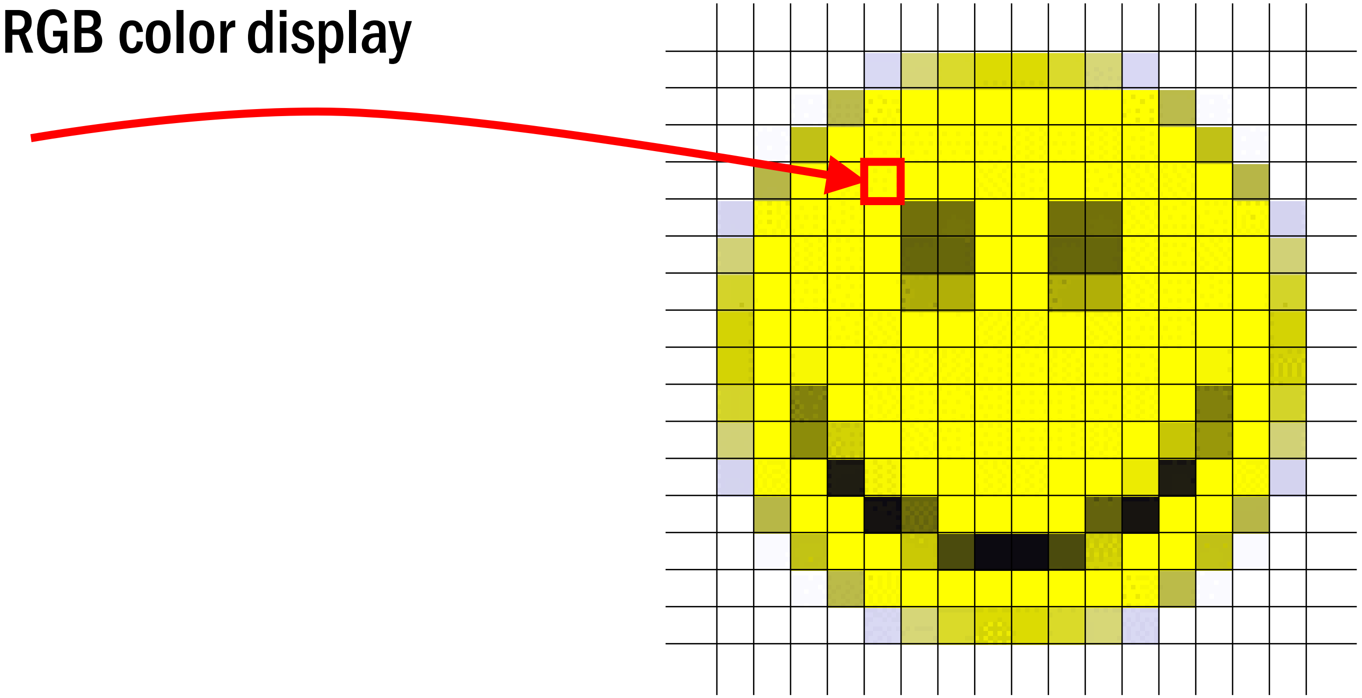
The divider you need for the clock



```
void HAL_LCD_SpiInit(void)
{
    ///////////////////////////
    // SPI configuration
    ///////////////////////////

    // Put eUSCI in reset state and set all fields in the register to 0
    UCB0CTLW0 = UCSWRST;

    // Fields that need to be nonzero are changed below

    // Set clock phase to "capture on 1st edge, change on following edge"
    ...
    // Set clock polarity to "inactive low"
    ...
    // Set data order to "transmit MSB first"
    ...
    // Set data size to 8-bit
    ...
    // Set MCU to "SPI master"
    ...
    // Set SPI to "3-pin SPI" (we won't use eUSCI's chip select)
    ...
    // Set module to synchronous mode
    ...
    // Set clock to SMCLK
    ...

    // Configure the clock divider (SMCLK is set to 16 MHz; run SPI at 8 MHz using SMCLK)
    // Maximum SPI supported frequency on the display is 10 MHz
    UCB0BRW = ...

    // Exit the reset state at the end of the configuration
    UCB0CTLW0...

    // Set CS' (chip select) bit to 0 (display always enabled)
    ...
    // Set DC' bit to 0 (assume data)
    ...
    //*/

    return;
}
```

LCD Pixel Display

- LCD is made of pixels (matrix of dots)
  - Each pixel could be either black/white or RGB color display
- Possibly touch-sensing capability
- TI BoosterPack
  - 128 x 128 pixels RGB display
  - Each pixel is represented by 18 bits
    - (6 bits for each – Red, Green, Blue)
    - 2^18=262144≈ 262K color display
      (i.e. each pixel is capable of displaying 262K different colors)



LCD Display Organization

- The LCD display component has
  - screen
  - built-in controller and
  - LED for backlighting

- The built-in controller is a microcontroller
  with a communication protocol
  to interface with the LCD display.

MSP430FR6989

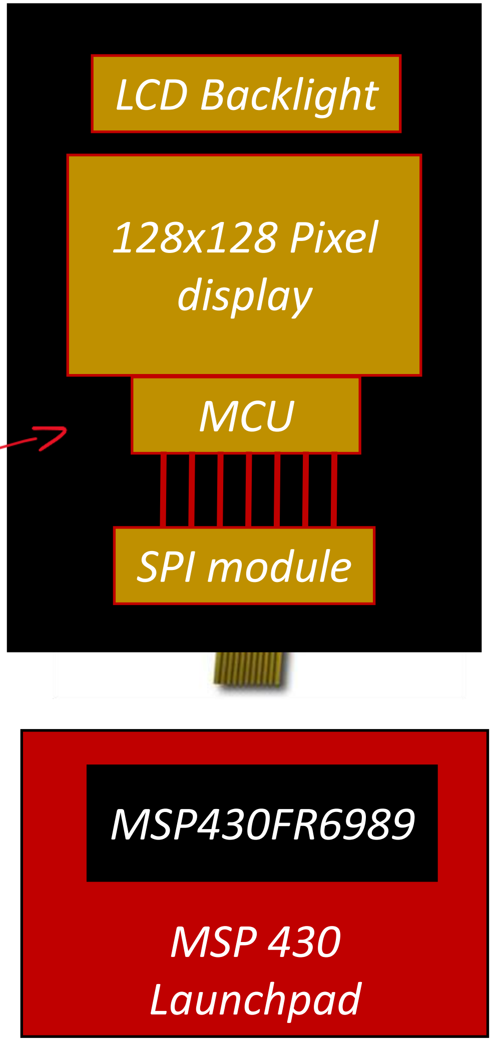
MSP 430 Launchpad

LCD Display Organization

- The LCD display component has
  - screen
  - built-in controller and
  - LED for backlighting

- The built-in controller is a microcontroller with a communication protocol to interface with the LCD display.

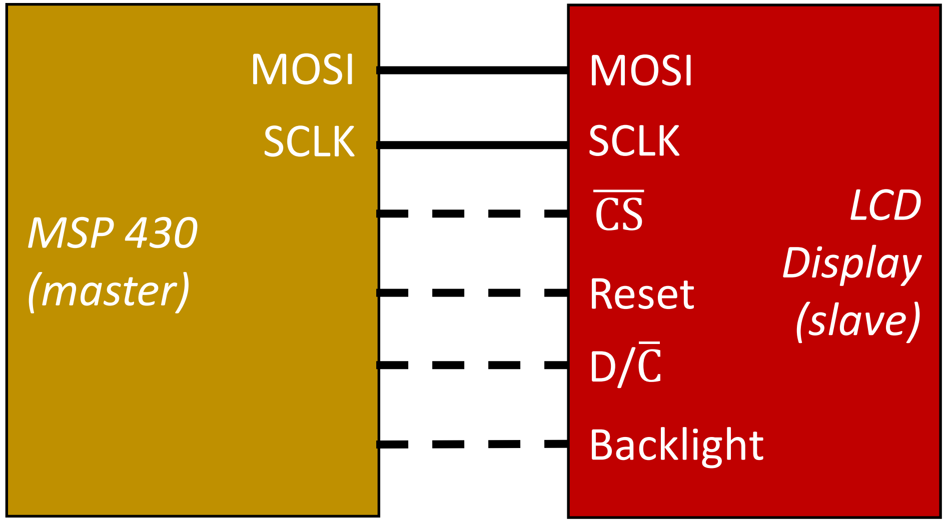
In our boosterpack, a Sitronix ST7735S MCU is integrated into the LCD display module, and it supports SPI communication protocol.



LCD Display Organization

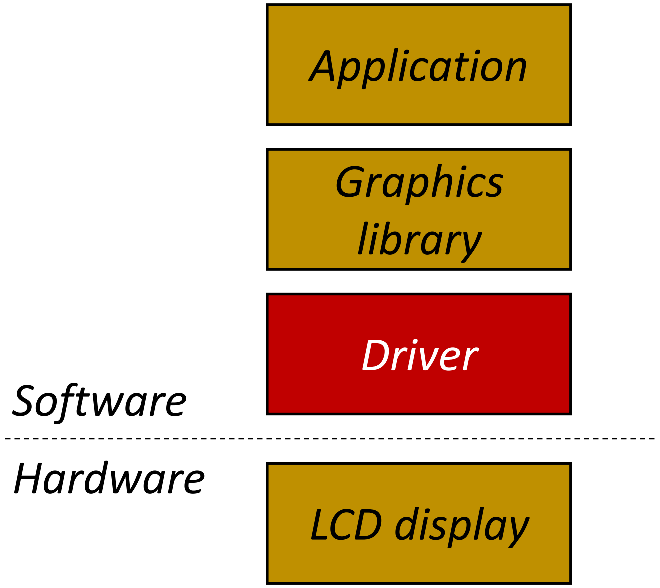
- The communication between MSP430 and LCD controller
  - via a half-duplex 3-wire SPI.

  - D/~C indicates whether the SPI byte is a command or a data
  - A low pulse on the reset pin resets the display MCU
  - Backlight pins are used to turn ON the backlight LED
  - Chip select is held low by connecting to ground.



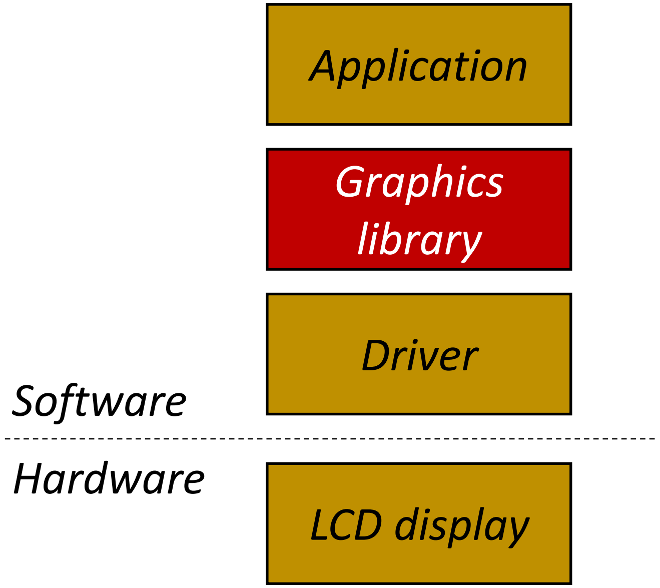
Software Stack for LCD Display

- Driver performs the low-level operations on the display.
  - Providing reset signal
  - Sending a byte of data/command
  - Drawing a single pixel
- Drivers are device specific
  - Two different devices of MSP430 might have different driver files
- Drivers are also specific to the display
  - Displays from different manufacturers will have different drivers.



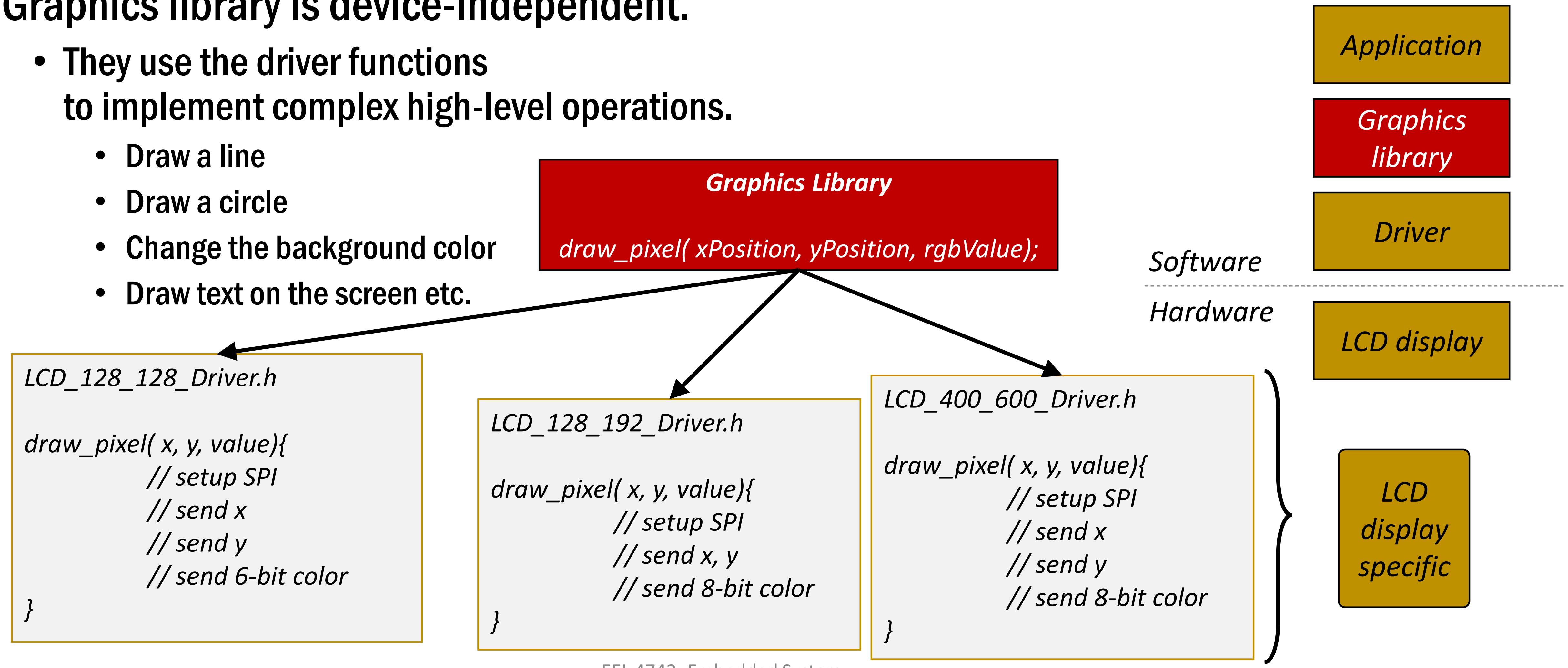
Software Stack for LCD Display

- Graphics library is device-independent.
  - They use the driver functions
    to implement complex high-level operations.
    - Draw a line
    - Draw a circle
    - Change the background color
    - Draw text on the screen
    - Draw an image
    - …



Software Stack for LCD Display

- Graphics library is device-independent.
  - They use the driver functions to implement complex high-level operations.
    - Draw a line
    - Draw a circle
    - Change the background color
    - Draw text on the screen etc.



Graphics Library

```
draw_pixel( xPosition, yPosition, rgbValue);
```

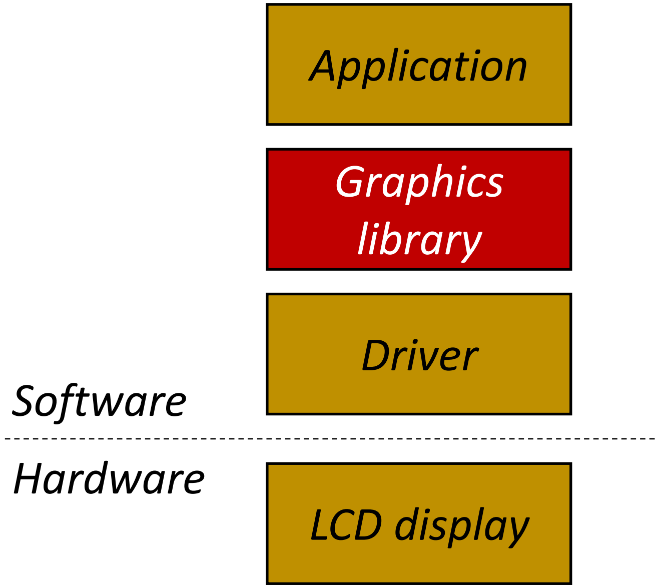
```
LCD_128_128_Driver.h

draw_pixel( x, y, value){
                // setup SPI
                // send x
                // send y
                // send 6-bit color
}
```

```
LCD_128_192_Driver.h

draw_pixel( x, y, value){
                // setup SPI
                // send x, y
                // send 8-bit color
}
```

```
LCD_400_600_Driver.h

draw_pixel( x, y, value){
                // setup SPI
                // send x
                // send y
                // send 8-bit color
}
```

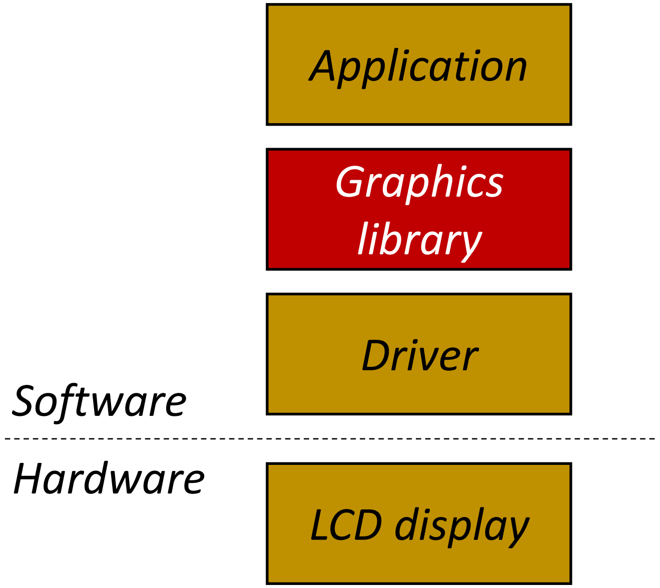
Software Stack for LCD Display

- Graphics library provided by TI
  - Provides an API for developing applications
    - Draw lines, circles, text, rectangles etc.
    - Draw buttons, sliders, check boxes (useful for touchscreens)
  - Provides different types of fonts
  - Image reformer utility
    - Converts images into a data structure C file, which can then be compiled for the chosen display



Software Stack for LCD Display

- An example of pseudocode using the graphic library


Application

Graphics library

Driver

Software

Hardware

LCD display

Graphics_drawImage(…) $\xrightarrow{\text{grlib.h}}$ *For drawing the images*

Image Reformer *is a utility that converts images into C code for use with MSP430 Graphics Library.*

from
.jpeg
.gif
.png
.bmp
.tif

based on the color depth per pixel
1 (max 2 colors)
2 (max 4 colors)
4 (max 16 colors)
8 (max 256 color)

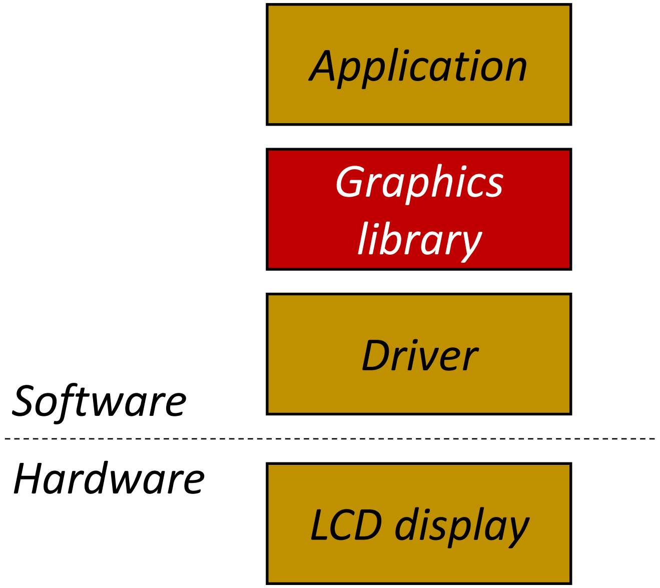
Determining the size
(+ compression)

to
.c

logo.c (one example converted)

Software Stack for LCD Display

- Graphics library provided by TI
    - Color translation of the graphic library



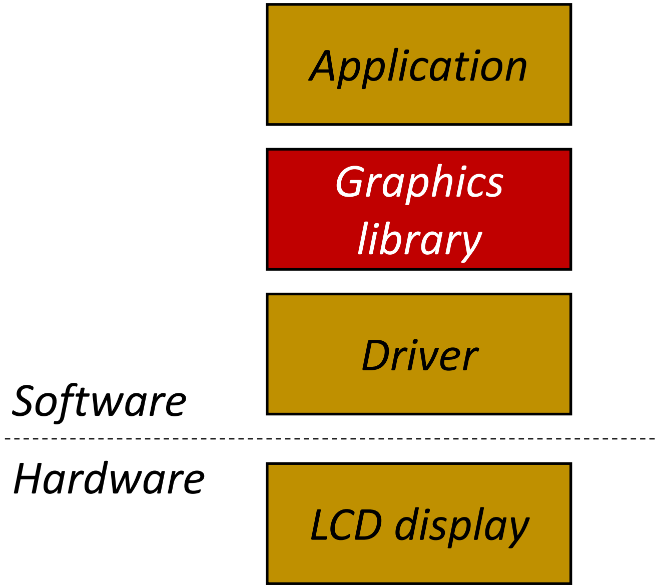
High resolution color display supports ***8-bits for each color****.*
So, RGB color is represented by a ***24-bit value****.*

	Red value	Green value	Blue value	
Red	255	0	0	🟥
Green	0	255	0	🟩
Blue	0	0	255	🟦
White	255	255	255	⬜
Black	0	0	0	⬛
Gold	255	192	0	🟨

How many different colors can be shown using a 24-bit representation? → = $2^{24} \approx$ 16 million colors

Software Stack for LCD Display

- An example of pseudocode using the graphic library



```
#define  COLOR_GOLD  0x00FFC000
…
Graphics_context        LCD_context;
Init_context ( &LCD_context);
Set_background_color( &LCD_context, COLOR_BLACK);

Set_foreground_color( &LCD_context, COLOR_GOLD);
Draw_string_centered( &LCD_context, “Hello World”, 64, 64);

Set_foreground_color( &LCD_context, COLOR_RED);
Draw_line_horizontal( &LCD_context, 20, 100, 50);
```

Since the LCD only supports 6-bit values, the driver scales the 8-bit value into 6-bit values

Red=255, Green=192, Blue=0
8-bit values

→

Red=63, Green=47, Blue=0
6-bit values

Thank You!

Questions?

Email: kamali@ucf.edu
UCF HEC 435 (407) 823 – 0764
https://www.ece.ucf.edu/~kamali/

HAVEN Research Group
https://haven.ece.ucf.edu/

HAVEN ASSURANCE VERIFICATION UCF UNIVERSITY OF CENTRAL FLORIDA