The University of Texas at Arlington

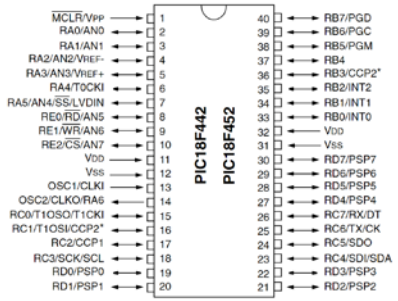
Lecture 14
Communication Peripherals

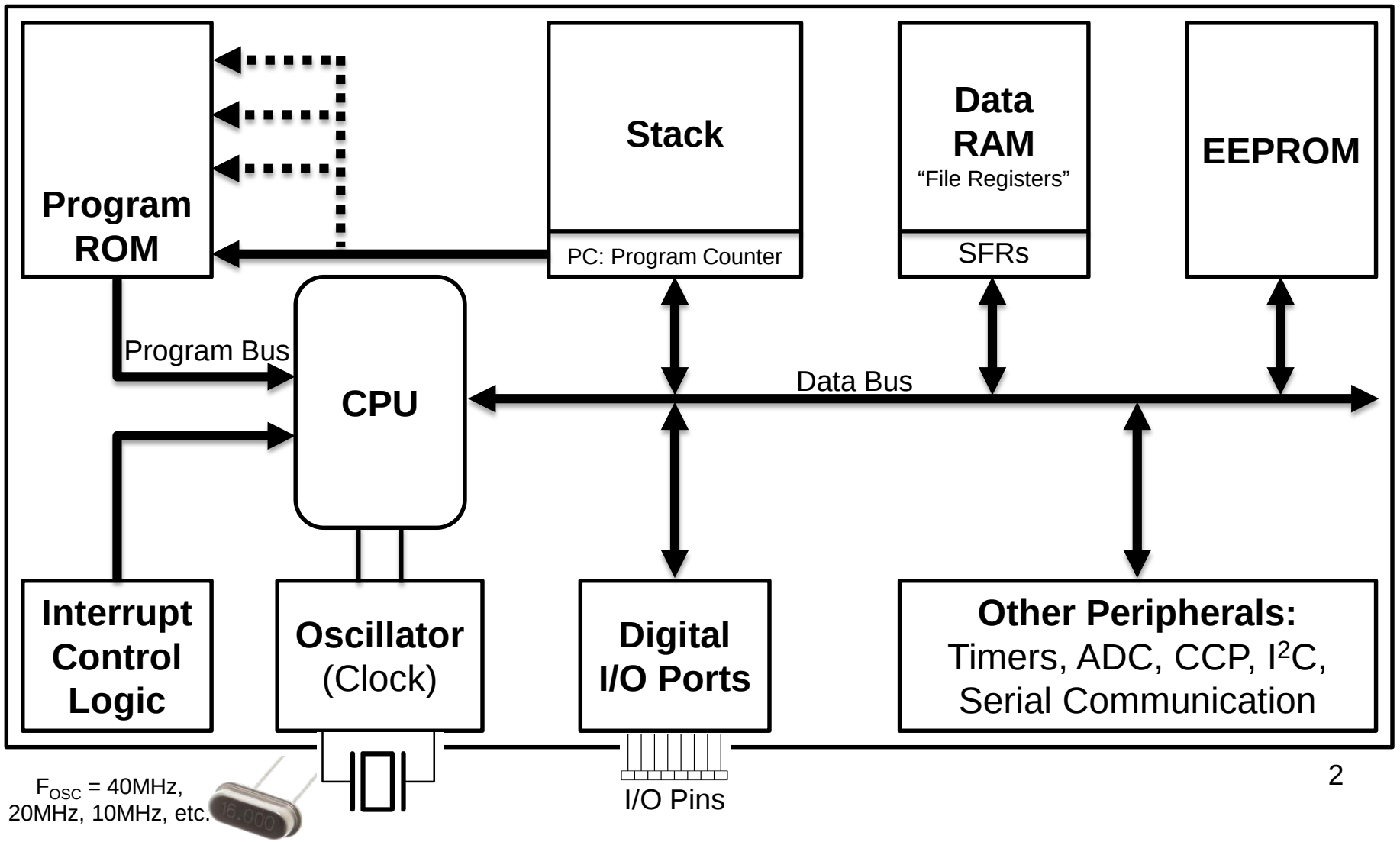
CSE 3442/5442

Embedded Systems 1

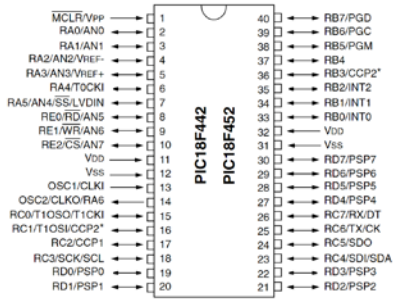
Based heavily on slides by Dr. Gergely Záruba and Dr. Roger Walker

High-Level View of a PIC18 Microcontroller





Communication Peripherals



Program ROM

Stack

PC: Program Counter

Data RAM

"File Registers"

SFRs

EEPROM

Program Bus

CPU

Data Bus

Interrupt Control Logic

Oscillator (Clock)

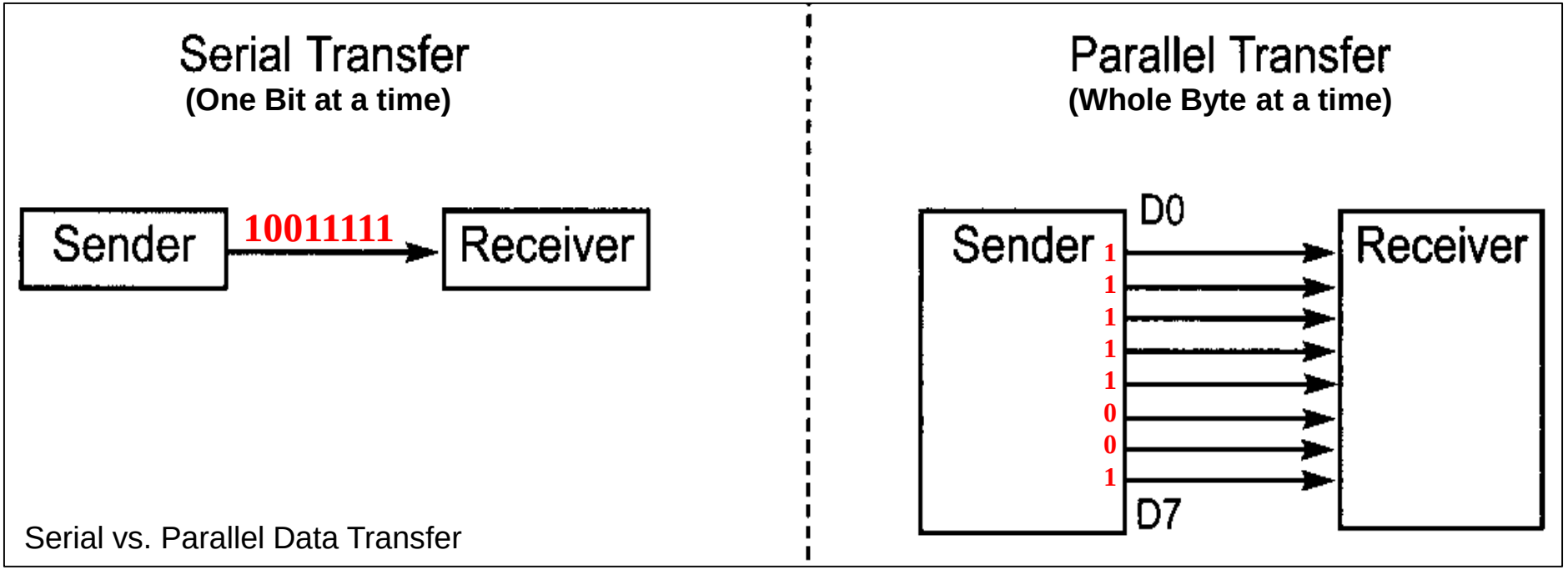
Digital I/O Ports

Other Peripherals:
Timers, ADC, CCP, **I²C**, **Serial Communication**

F_{OSC} = 40MHz, 20MHz, 10MHz, etc.

I/O Pins

Serial vs. Parallel Communication



Serial vs. Parallel Data Transfer

- **Fewer connections**
  - Cheaper
- **Great for far distances**
- **Slower**

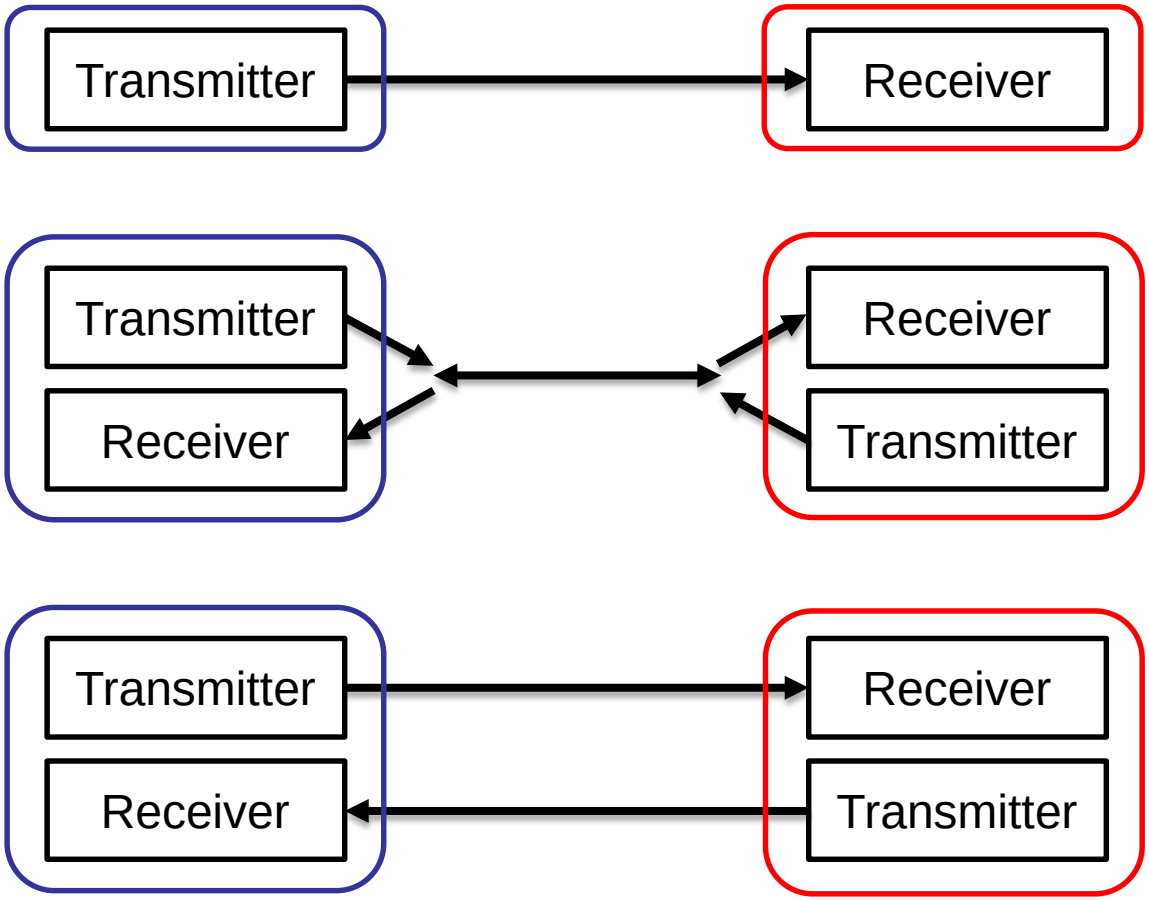
- **Many connections**
  - Expensive
- **More points of failure**
- **Faster**

Serial Data Transfer Methods

- **Synchronous**
  - Sender and Receiver share a clock signal
  - High data transfer rate
  - “Blocks” of data at a time
- **Asynchronous**
  - Sender provides a sync signal to the Receiver before starting each transmission
    - No shared clock but must agree upon a data-rate
  - Single bytes at a time
  - Slower data transfer rate but more flexible

Serial Data Transfer Types

- **Simplex**
  - one-way
- **Half-Duplex**
  - one line
- **Full-Duplex**
  - two lines

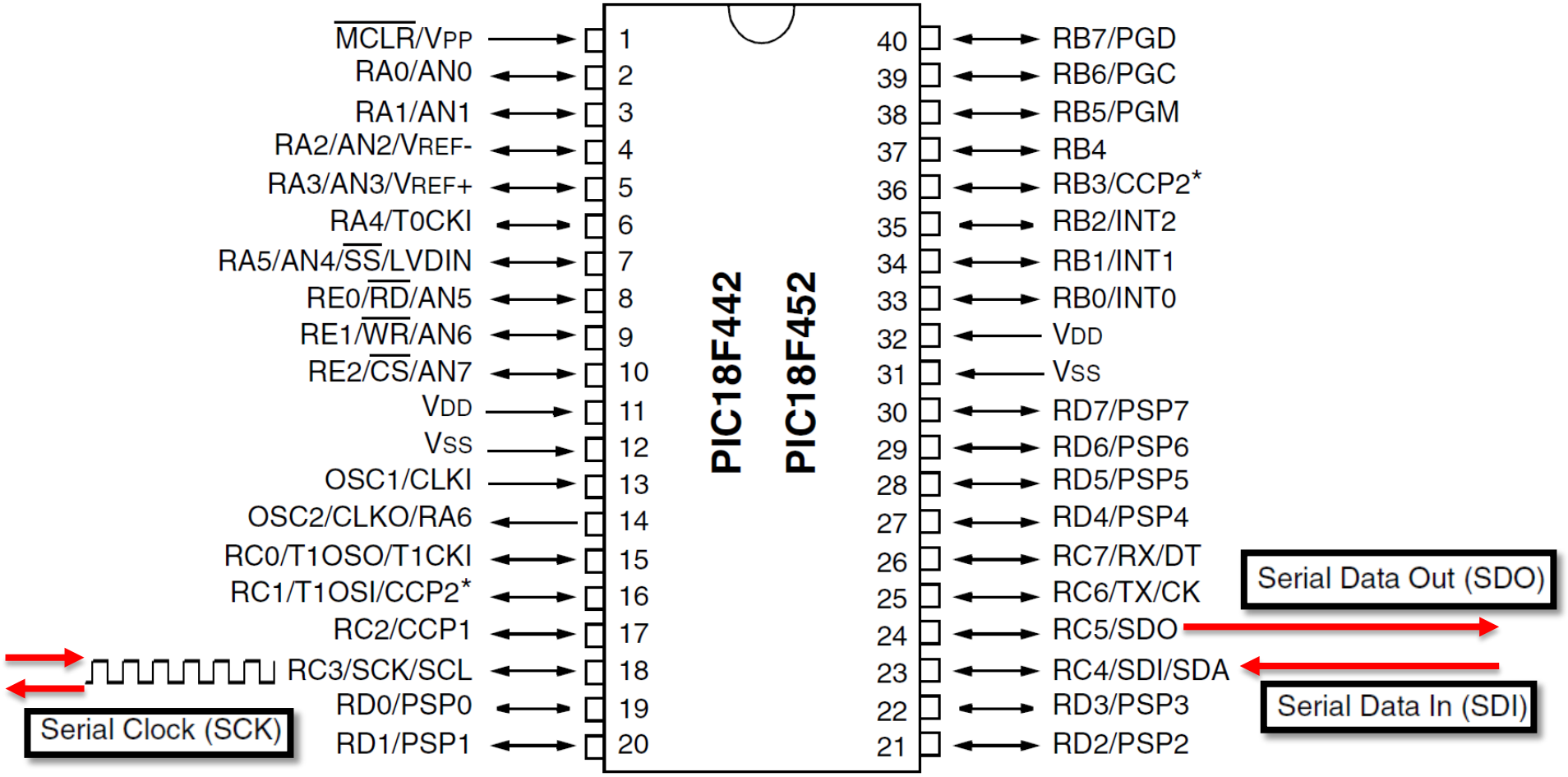


PIC Communication Peripherals (4 Modes)

- **MSSP: Master Synchronous Serial Port Module**
  - **SPI**: Serial Peripheral Interface
  - **I²C**: Inter-Integrated Circuit
    - Full Master Mode
    - Multi-Master Mode
    - Slave Mode
- **USART: Universal Synchronous/Asynchronous Receiver/Transmitter Module**
  - **ART**: Asynchronous (full-duplex)
  - **SRT**: Synchronous (half-duplex)
    - Master
    - Slave

MSSP SPI Mode

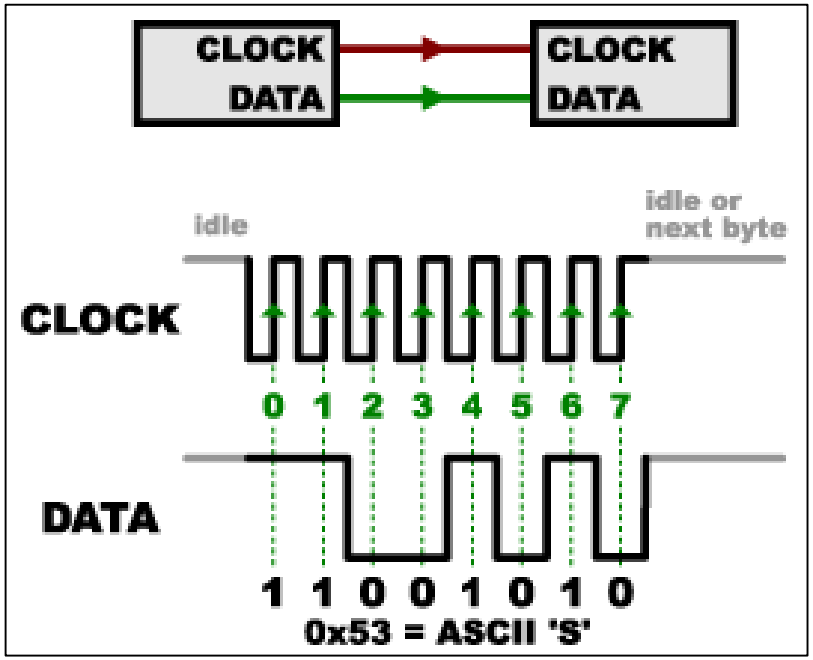
The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

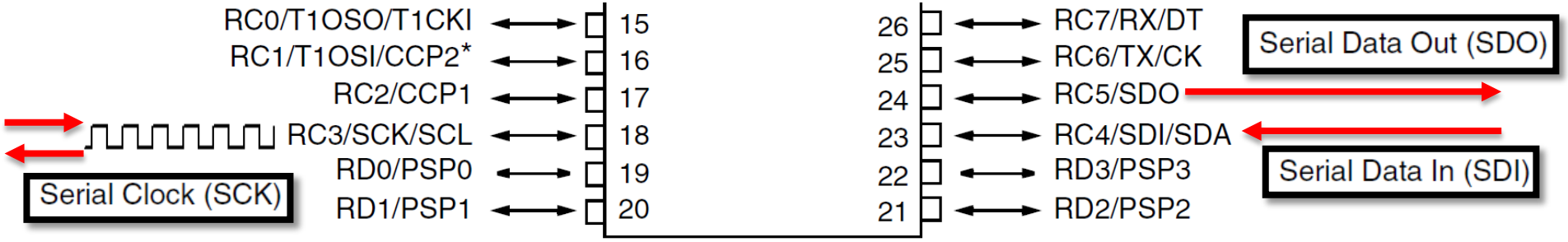


MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

Generic Example:
Clock's Idle State = HIGH
Clock Data Out/In on RISING EDGE





MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

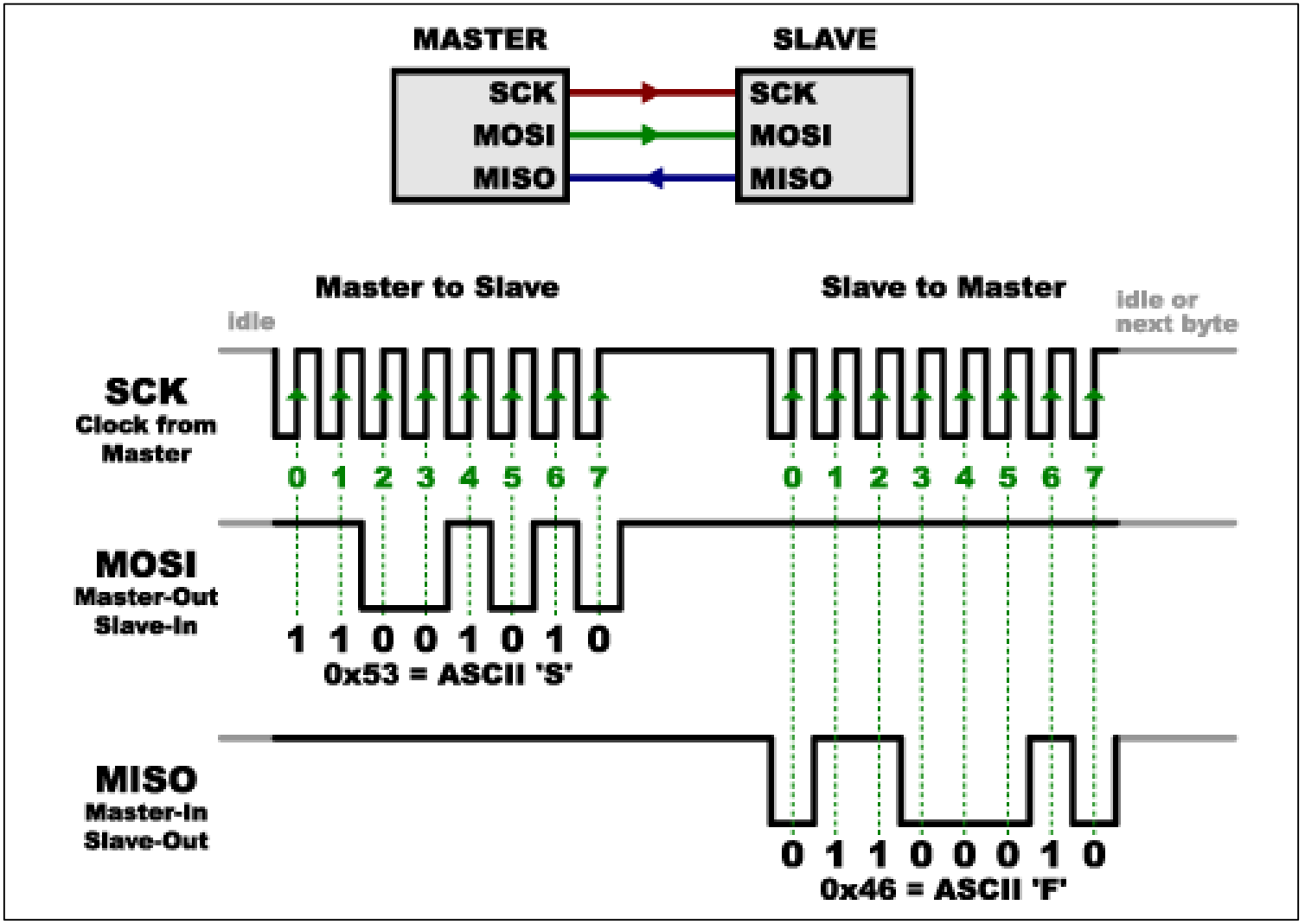
- SPI is a sync. **Data Exchange** protocol
- As data is clocked out, new data is clocked in
  - A FULL duplex data transmission occurs for each SPI clock cycle (if desired)

The MSSP module has four registers for SPI mode operation. These are:

- MSSP Control Register1 (SSPCON1)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer (SSPBUF)
- MSSP Shift Register (SSPSR) - Not directly accessible

MSSP SPI Mode

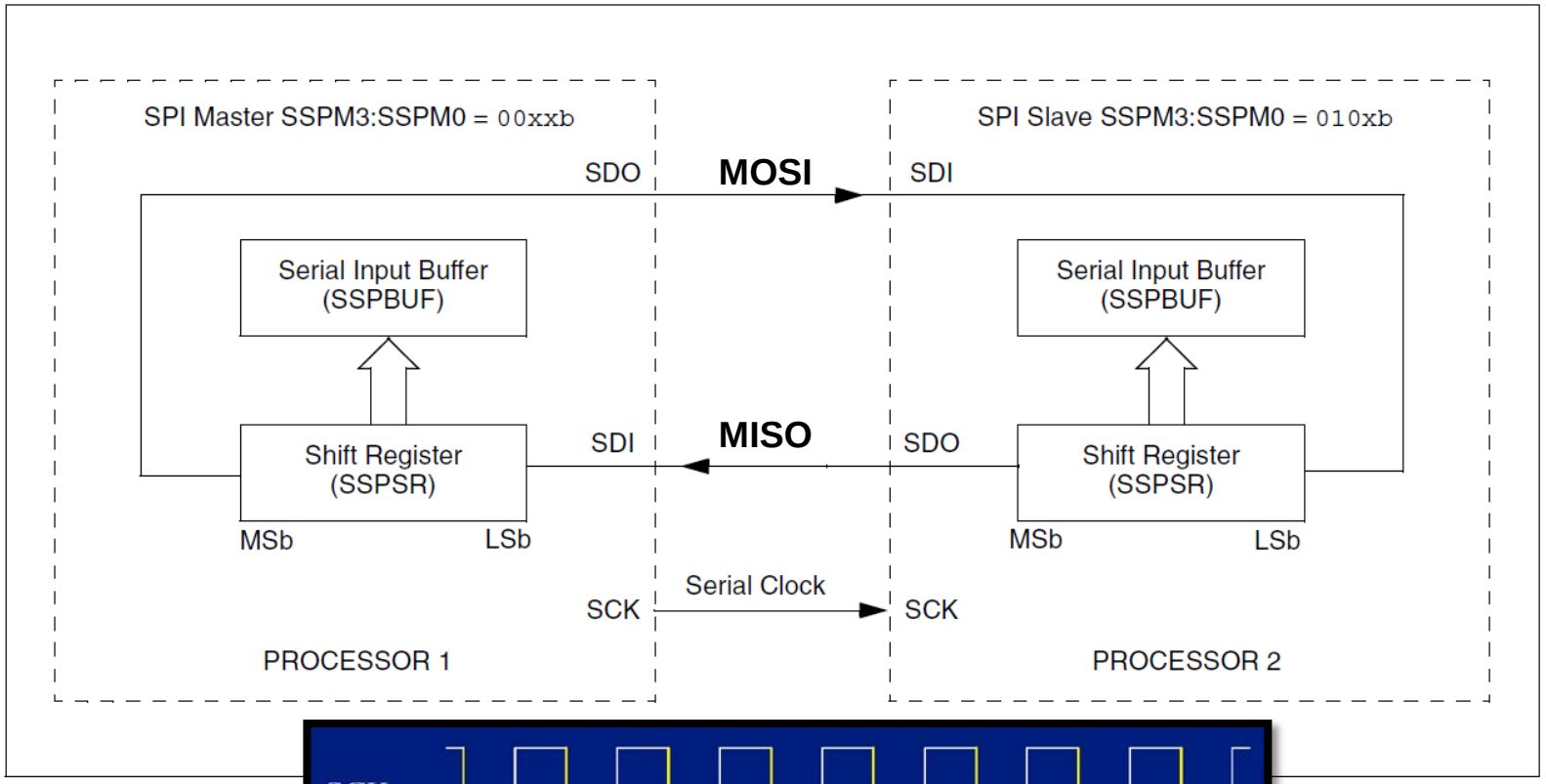
The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

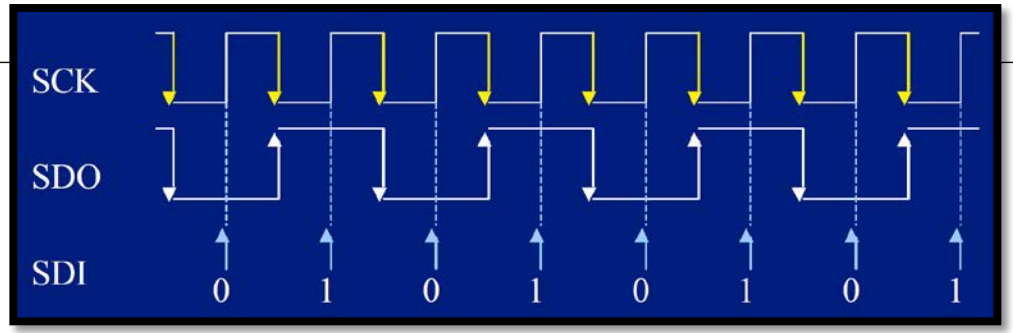


MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

FIGURE 15-2: SPI MASTER/SLAVE CONNECTION





MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

15-1: SSPSTAT: MSSP STATUS REGISTER (SPI MODE)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit

SPI Master mode:

1 = Input data sampled at end of data output time

0 = Input data sampled at middle of data output time

SPI Slave mode:

SMP must be cleared when SPI is used in Slave mode

bit 6 **CKE:** SPI Clock Edge Select

When CKP = 0:

1 = Data transmitted on rising edge of SCK

0 = Data transmitted on falling edge of SCK

When CKP = 1:

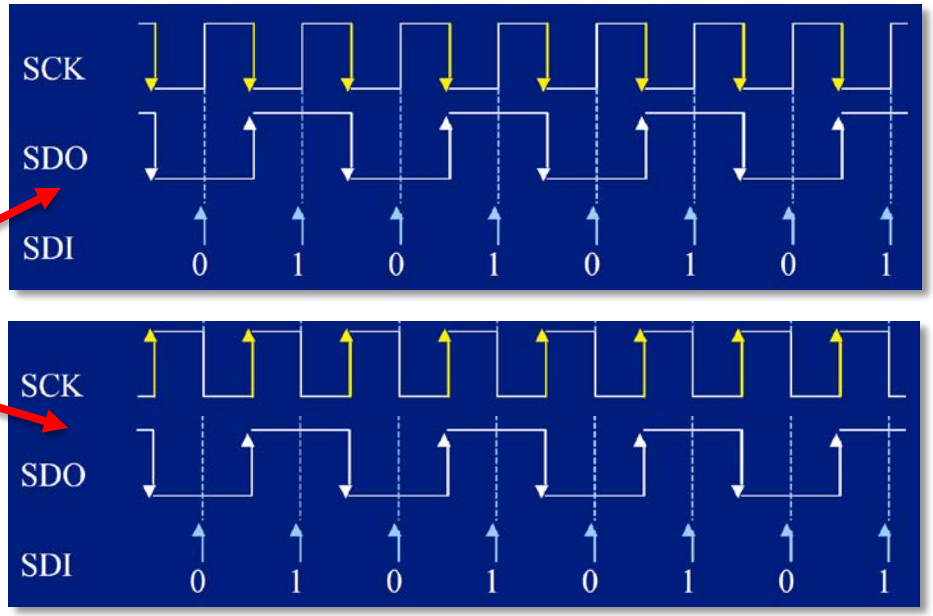
1 = Data transmitted on falling edge of SCK

0 = Data transmitted on rising edge of SCK

bit 0 **BF:** Buffer Full Status bit (Receive mode only)

1 = Receive complete, SSPBUF is full

0 = Receive not complete, SSPBUF is empty



MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

15-2: SSPCON1: MSSP CONTROL REGISTER1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit (Transmit mode only)
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit
SPI Slave mode:
1 = A new byte is received while the SSPBUF register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in Slave mode.The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow (must be cleared in software).
0 = No overflow

Note: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
1 = Enables serial port and configures SCK, SDO, SDI, and $\overline{SS}$ as serial port pins
0 = Disables serial port and configures these pins as I/O port pins

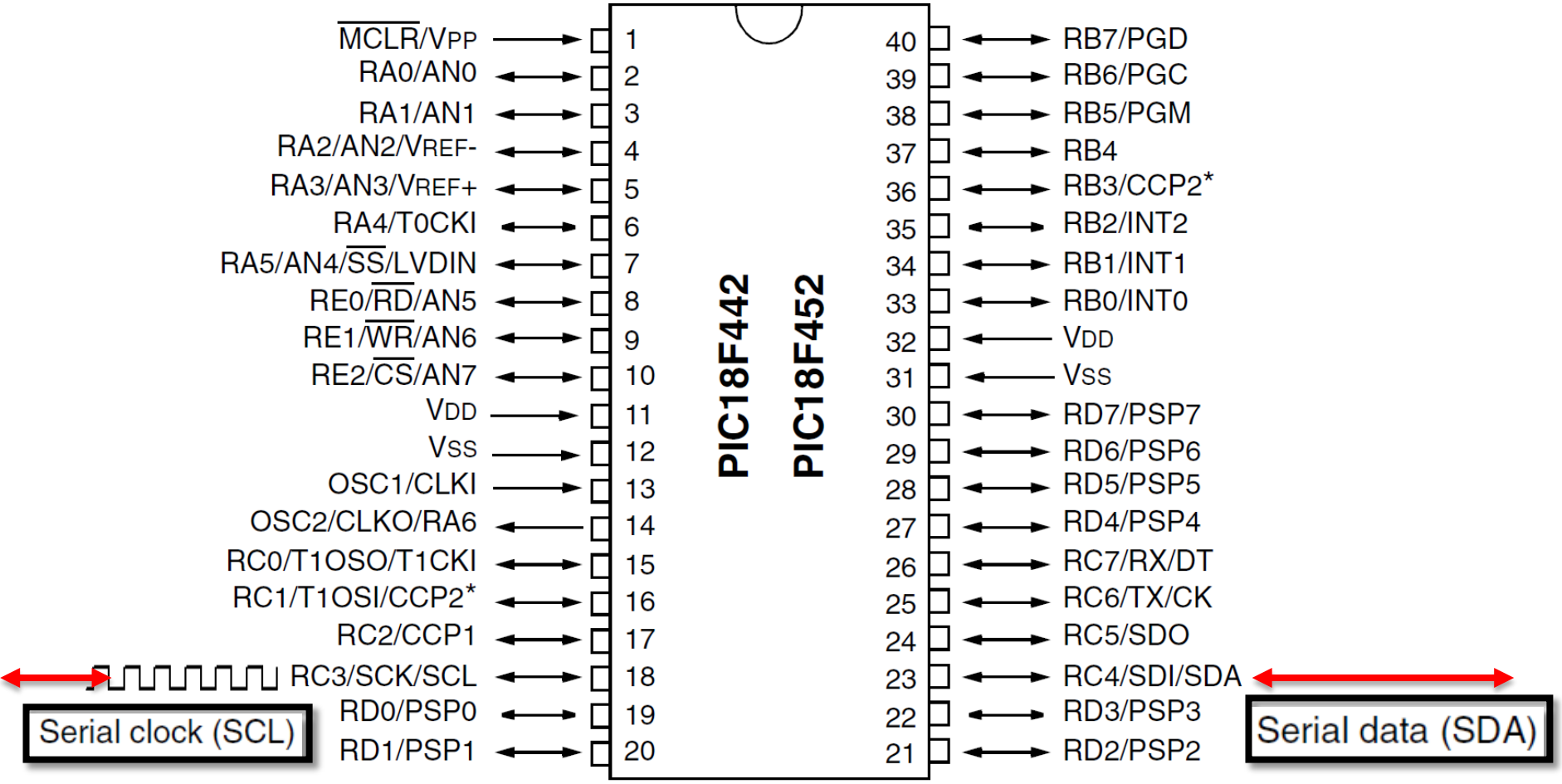
Note: When enabled, these pins must be properly configured as input or output.

bit 4 **CKP:** Clock Polarity Select bit
1 = IDLE state for clock is a high level
0 = IDLE state for clock is a low level

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
0101 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin
0100 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = FOSC/64
0001 = SPI Master mode, clock = FOSC/16
0000 = SPI Master mode, clock = FOSC/4

MSSP I²C Mode

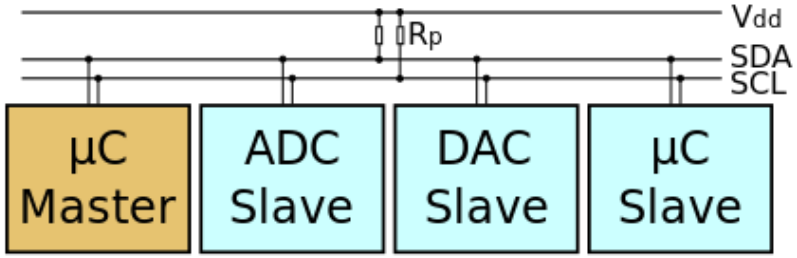
The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit



MSSP
I²C Mode

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit

- I²C is a sync. bi-directional protocol (2 lines)
  - "Acknowledge" System
  - Master-Slave relationships
  - Multi-Slave & Multi-Master
  - Shared Bus



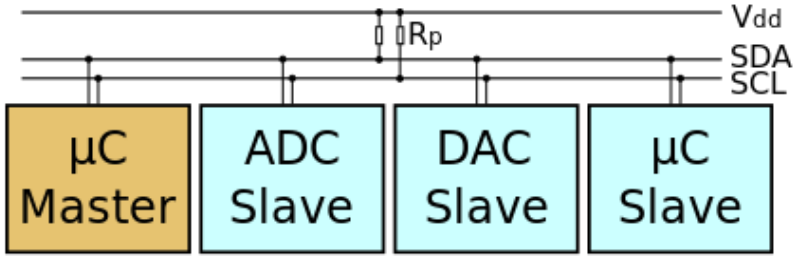
The MSSP module has six registers for I²C operation. These are:

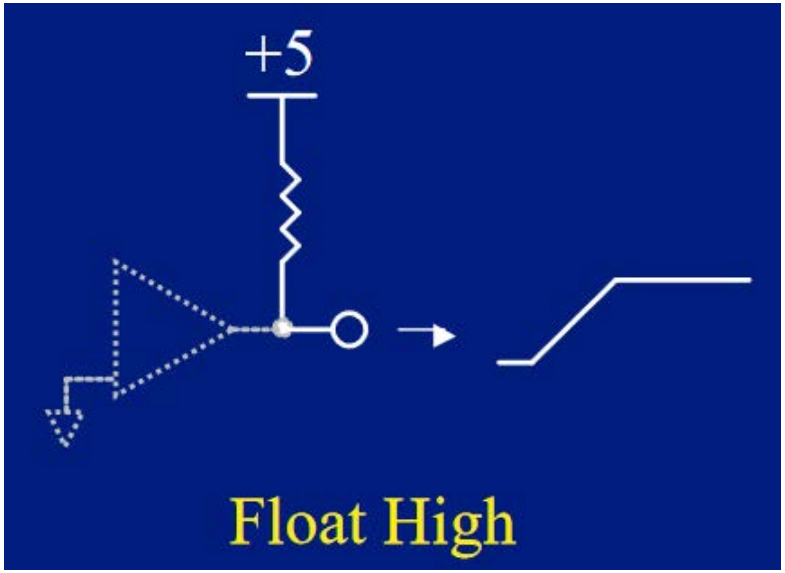
- MSSP Control Register1 (SSPCON1)
- MSSP Control Register2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer (SSPBUF)
- MSSP Shift Register (SSPSR) - Not directly accessible
- MSSP Address Register (SSPADD)

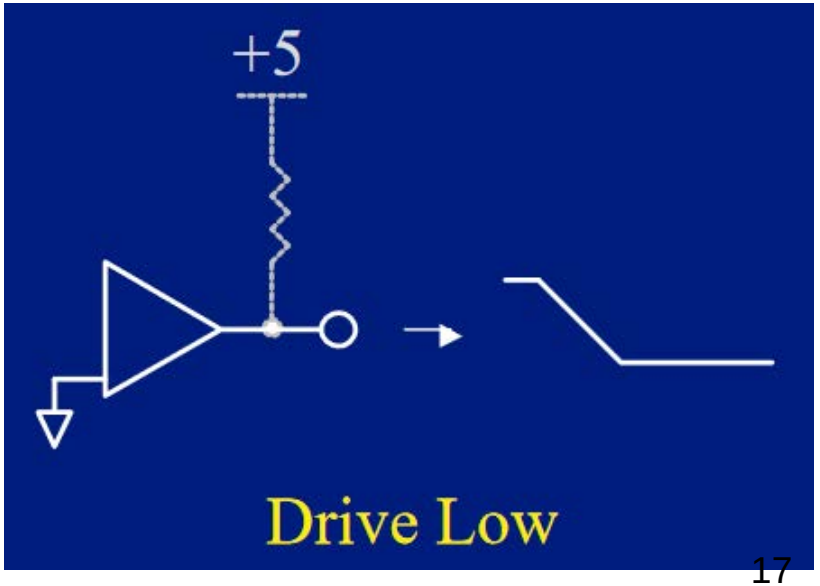
MSSP I²C Mode

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit

- I²C has two signal levels (open drain)
  - Float HIGH (Logic 1)
  - Drive LOW (Logic 0)



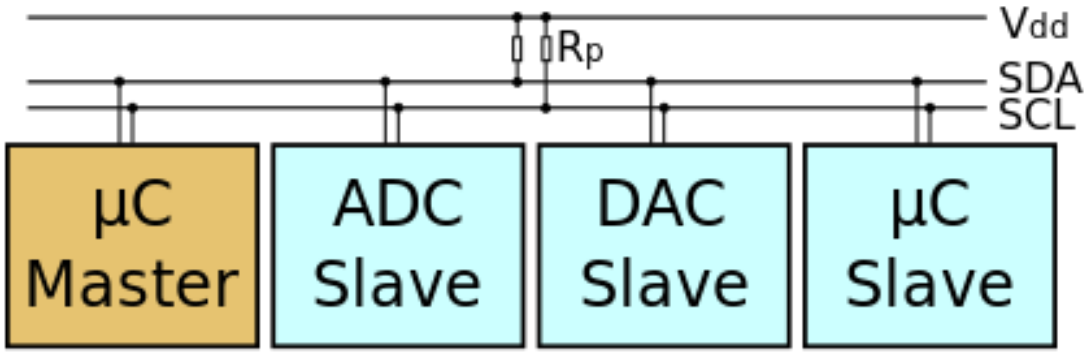


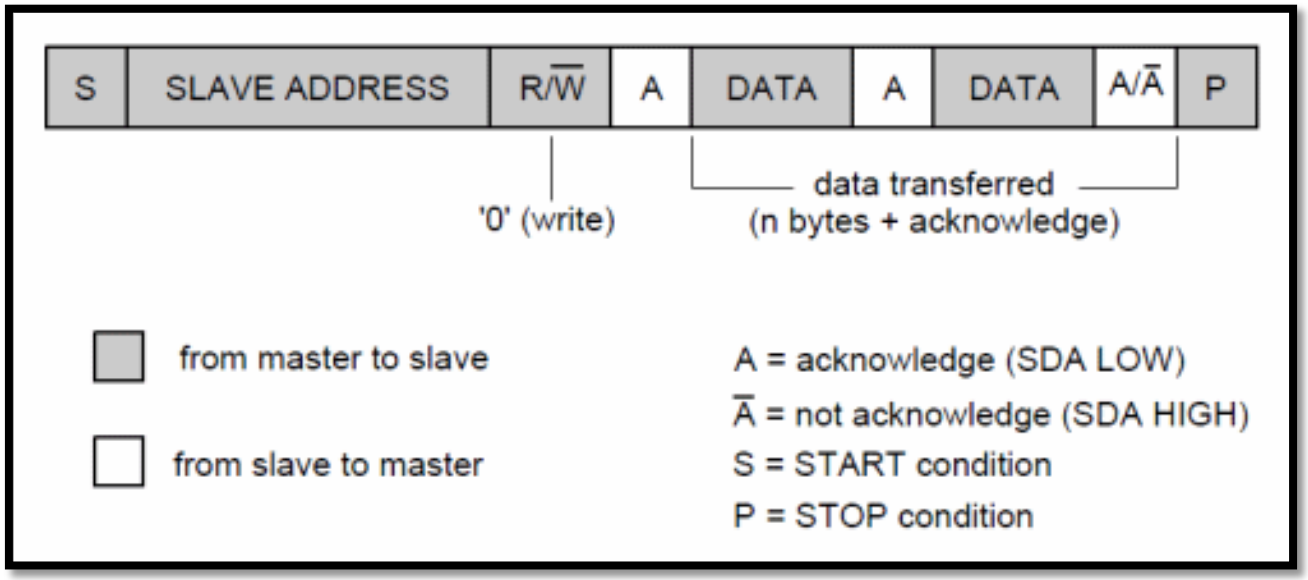


MSSP I²C Mode

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit

Master **writing** to a Slave

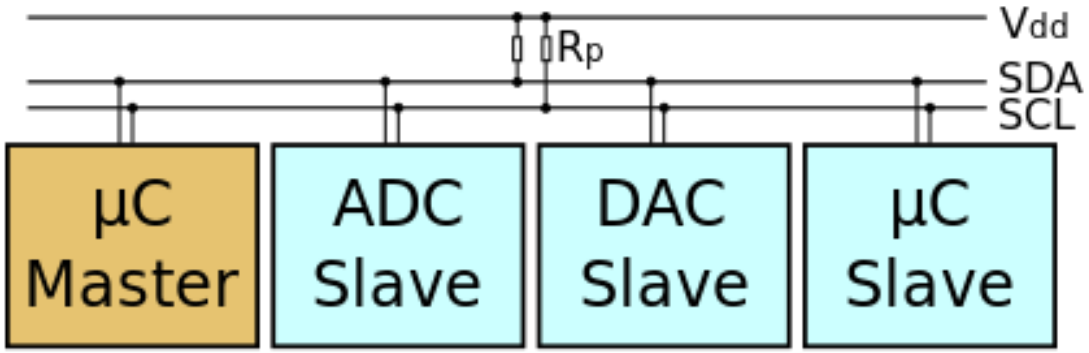


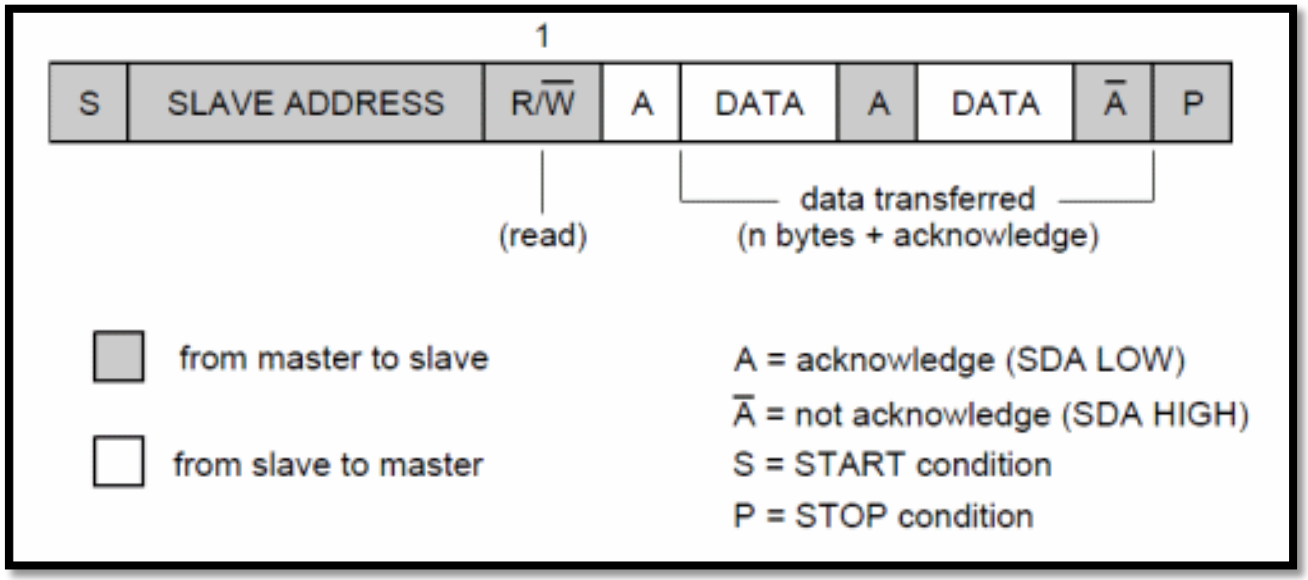


MSSP I²C Mode

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit

Master **reading** from a Slave

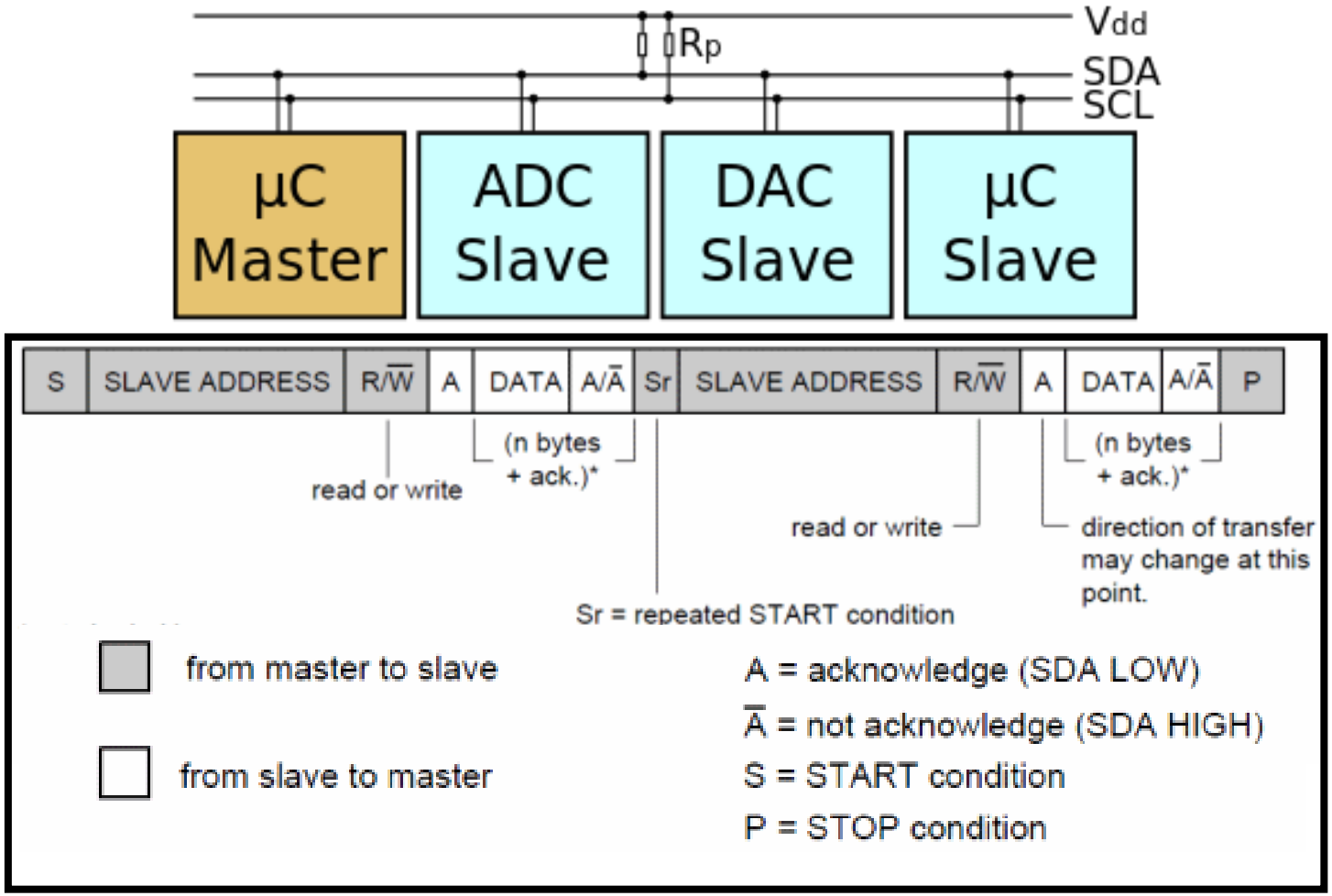




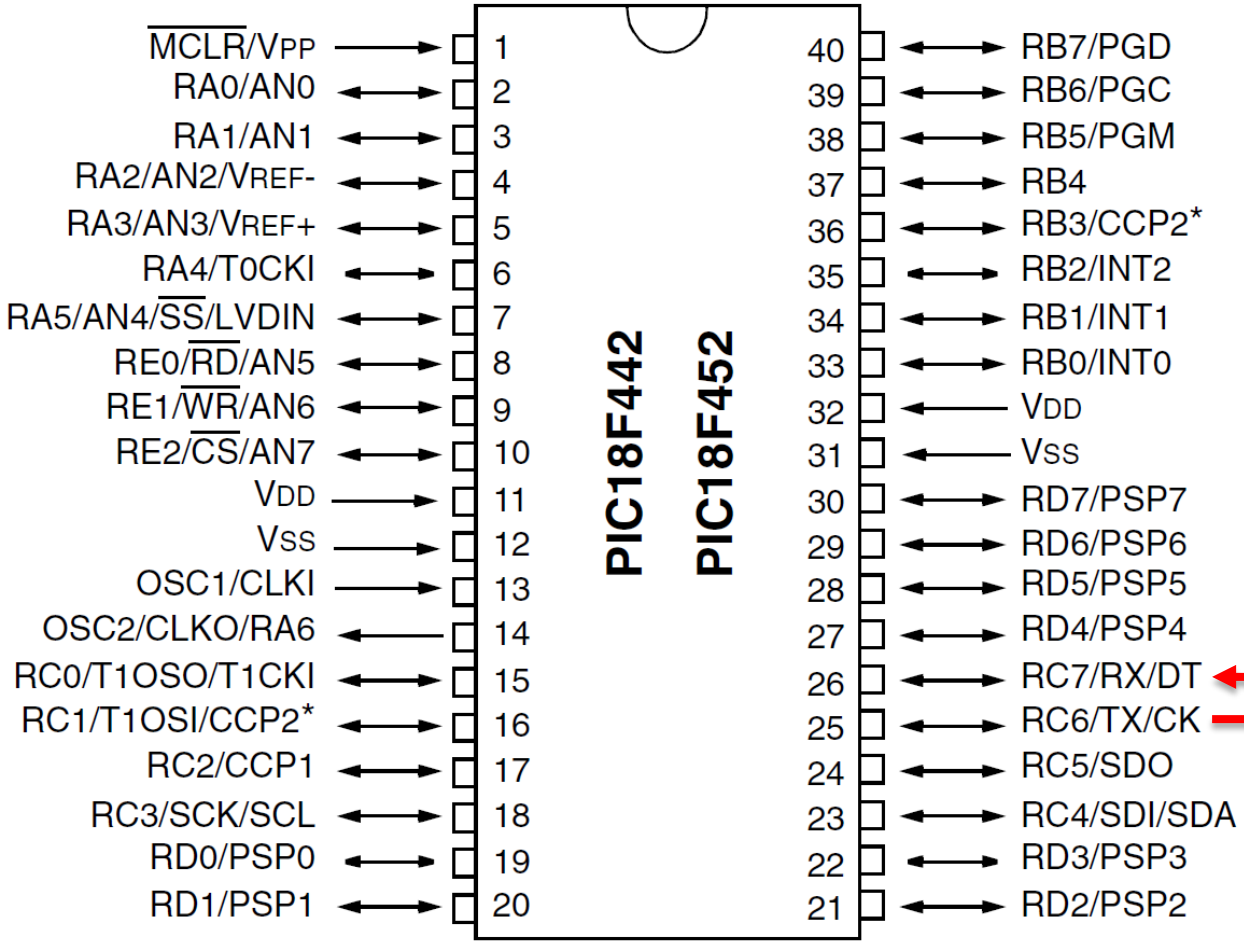
MSSP I²C Mode

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit

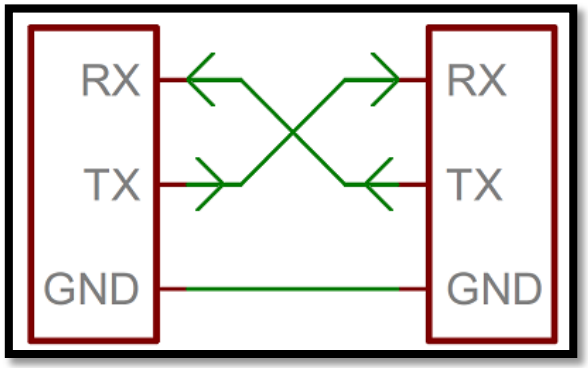
Master **reading** from a Slave (special cond.)



USART
Async. Full-Duplex



Crisscross the RX/TX Pins!!



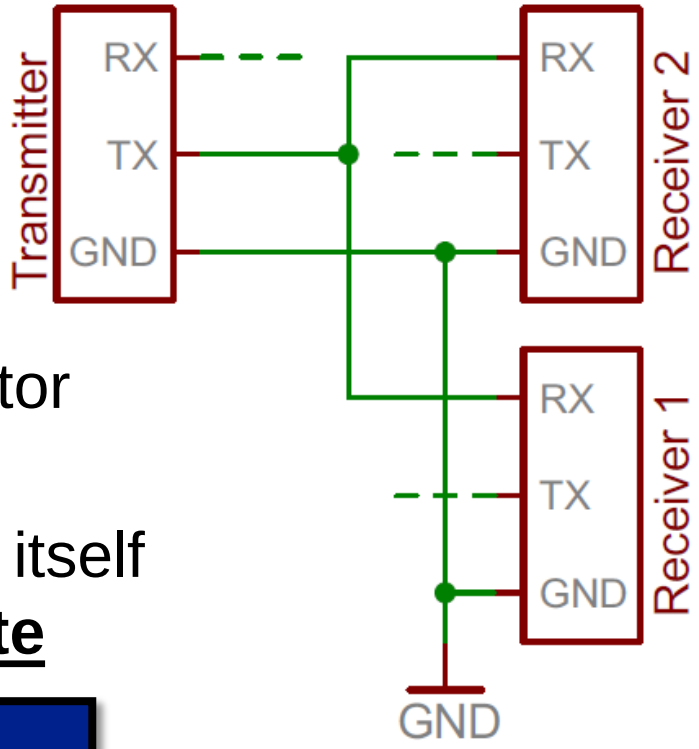
RX: Receive Data Pin

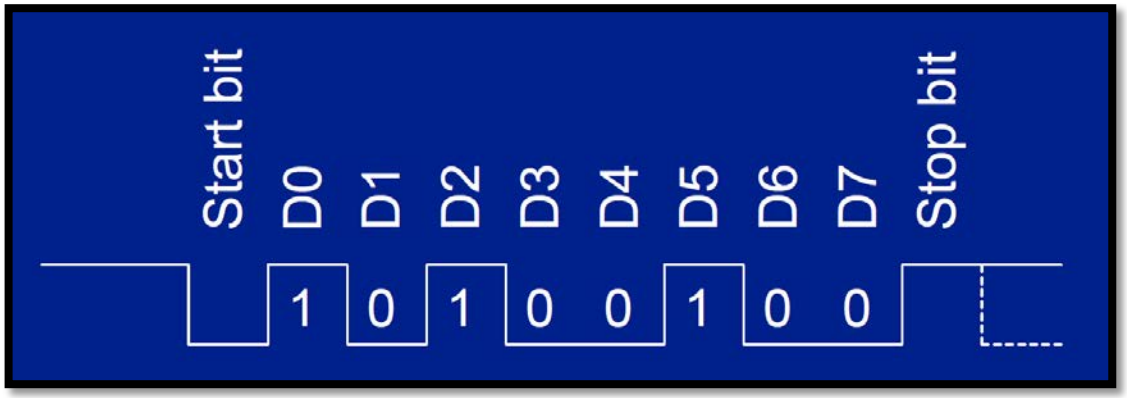
TX: Transmit Data Pin

No shared clock

USART: Async.

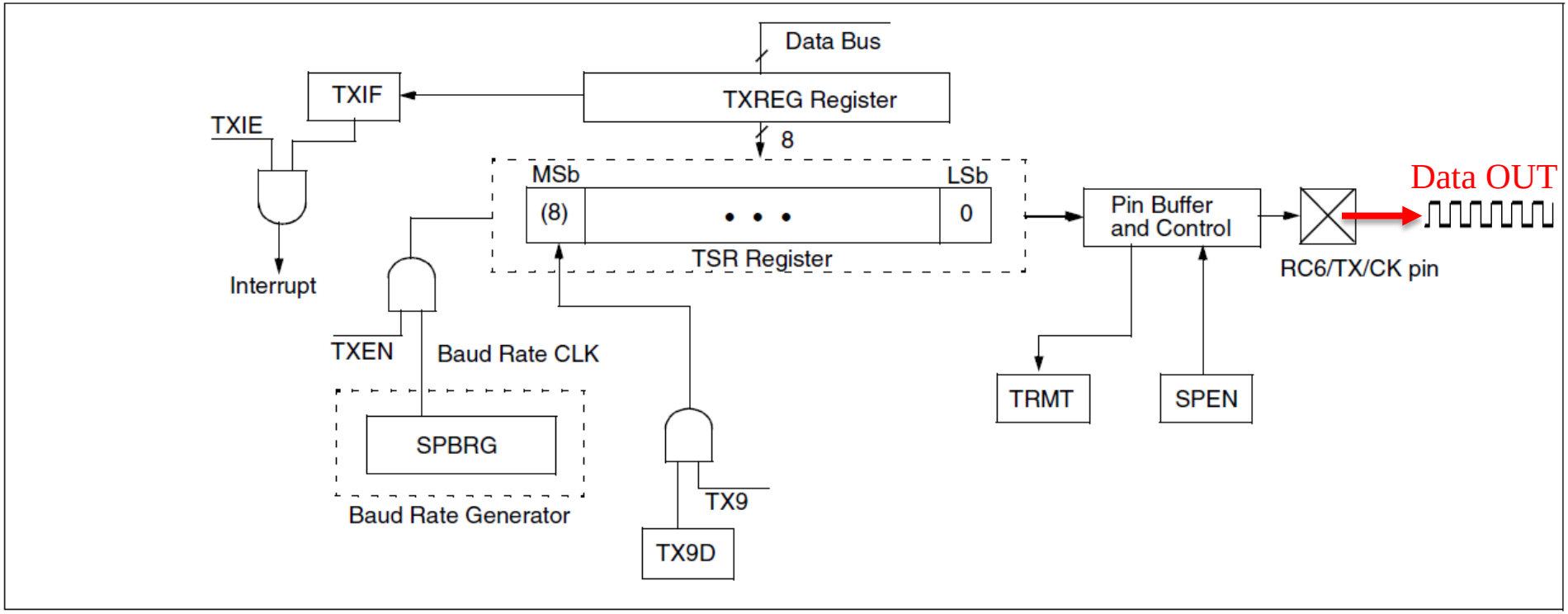
- **TXSTA**: Transmit Status and Control
  - **TXREG**: data to be sent out
- **RCSTA**: Receive Status and Control
  - **RCREG**: where received data is put
- **SPBRG**: Serial Port Baud Rate Generator
  - define the data rate
- Instead of a shared clock, now the data itself is sent out at a settable rate, **BAUD Rate**





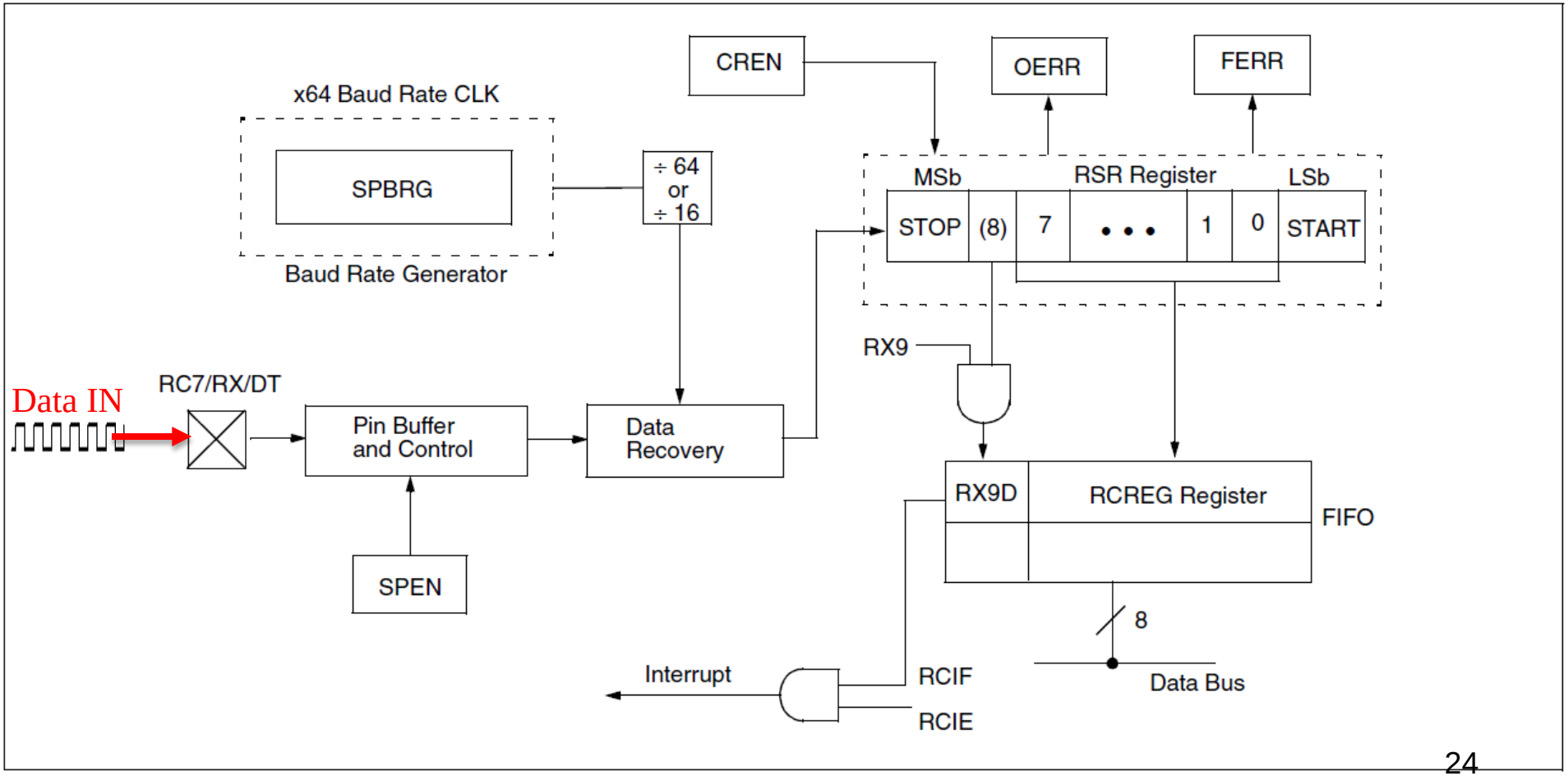
USART: Transmission
TX Output Pin

FIGURE 16-1: USART TRANSMIT BLOCK DIAGRAM



USART: Reception
RX Input Pin

FIGURE 16-4: USART RECEIVE BLOCK DIAGRAM



BAUD Rate in bps (bits per second)

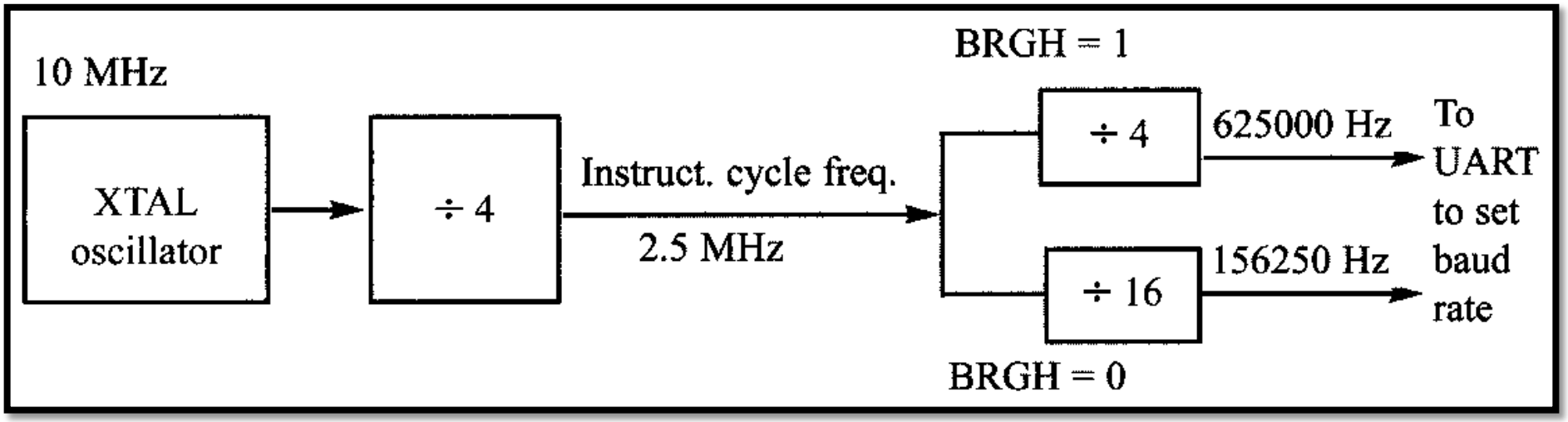


TABLE 16-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

Legend: X = value in SPBRG (0 to 255)

BAUD Rate Ex: F_{OSC} = 16 MHz Want 9,600 bps (Low Speed)

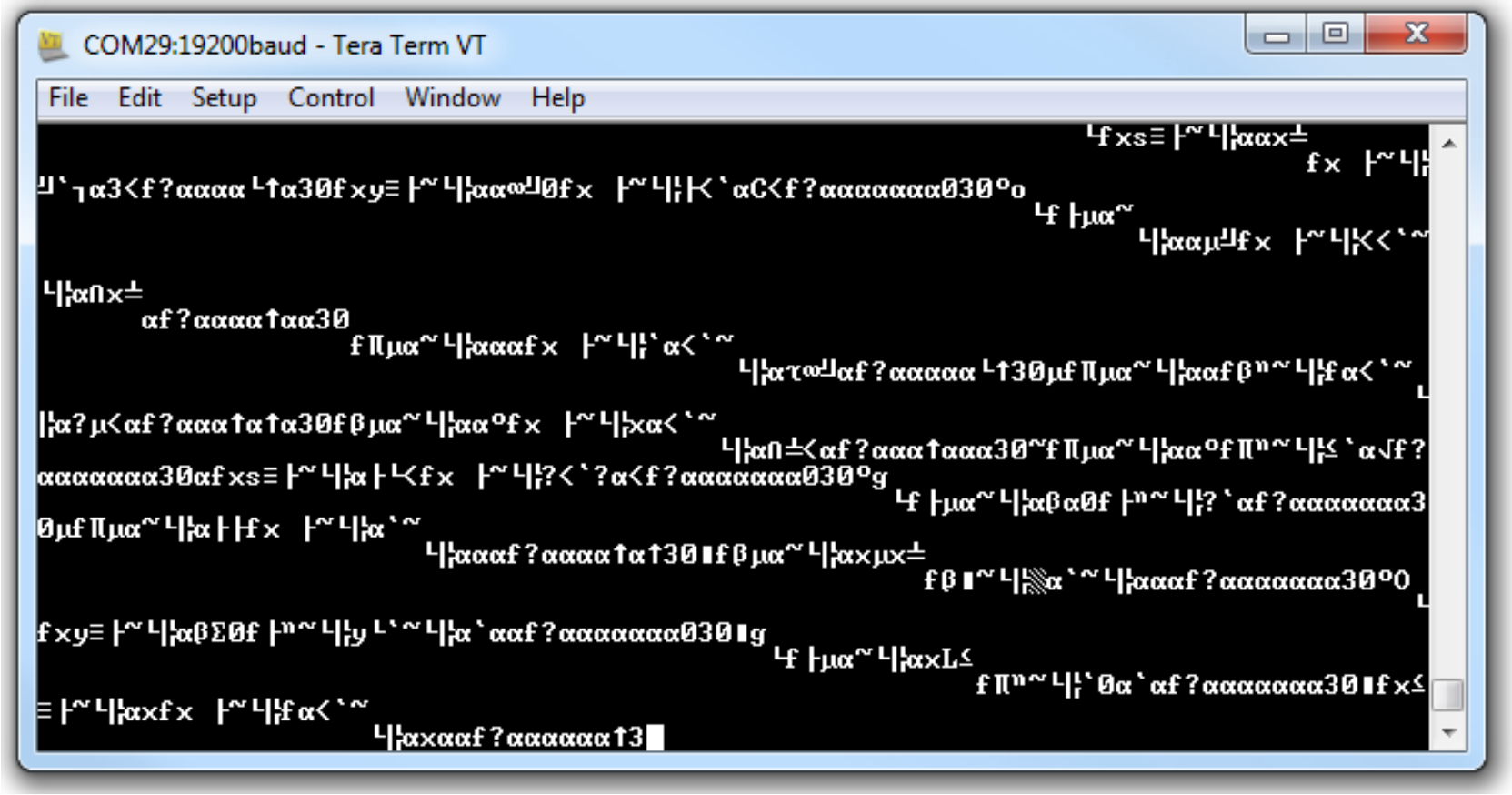
Desired Baud Rate	=	Fosc / (64 (X + 1))
Solving for X:		X = value in SPBRG (0 to 255)
X	=	((Fosc / Desired Baud Rate) / 64) – 1
X	=	((16000000 / 9600) / 64) – 1
X	=	[25.042] = 25 ← SPBRG
Calculated Baud Rate	=	16000000 / (64 (25 + 1))
	=	9615
Error	=	(Calculated Baud Rate – Desired Baud Rate) / Desired Baud Rate
	=	(9615 – 9600) / 9600
	=	0.16%

TABLE 16-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

Legend: X = value in SPBRG (0 to 255) ←

BAUD Rate Mismatch



Common BAUD Rates (Low Speed)

TABLE 16-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (Kbps)	Fosc = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	2.40	-0.07	214	2.40	-0.15	162	2.40	+0.16	129
9.6	9.62	+0.16	64	9.55	-0.54	53	9.53	-0.76	40	9.47	-1.36	32
19.2	18.94	-1.36	32	19.10	-0.54	26	19.53	+1.73	19	19.53	+1.73	15
76.8	78.13	+1.73	7	73.66	-4.09	6	78.13	+1.73	4	78.13	+1.73	3
96	89.29	-6.99	6	103.13	+7.42	4	97.66	+1.73	3	104.17	+8.51	2
300	312.50	+4.17	1	257.81	-14.06	1	NA	-	-	312.50	+4.17	0
500	625	+25.00	0	NA	-	-	NA	-	-	NA	-	-
HIGH	625	-	0	515.63	-	0	390.63	-	0	312.50	-	0
LOW	2.44	-	255	2.01	-	255	1.53	-	255	1.22	-	255

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:

1 = High speed

0 = Low speed

Synchronous mode:

Unused in this mode

Common BAUD Rates (High Speed)

CSE@UTA

TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (Kbps)	Fosc = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	9.60	-0.07	214	9.59	-0.15	162	9.62	+0.16	129
19.2	19.23	+0.16	129	19.28	+0.39	106	19.30	+0.47	80	19.23	+0.16	64
76.8	75.76	-1.36	32	76.39	-0.54	26	78.13	+1.73	19	78.13	+1.73	15
96	96.15	+0.16	25	98.21	+2.31	20	97.66	+1.73	15	96.15	+0.16	12
300	312.50	+4.17	7	294.64	-1.79	6	312.50	+4.17	4	312.50	+4.17	3
500	500	0	4	515.63	+3.13	3	520.83	+4.17	2	416.67	-16.67	2
HIGH	2500	-	0	2062.50	-	0	1562.50	-	0	1250	-	0
LOW	9.77	-	255	8,06	-	255	6.10	-	255	4.88	-	255

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:

1 = High speed

0 = Low speed

Synchronous mode:

Unused in this mode

R 16-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in SYNC mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data
Can be Address/Data bit or a parity bit.

R 16-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9:** 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode - Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode - Slave:
Don't care

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

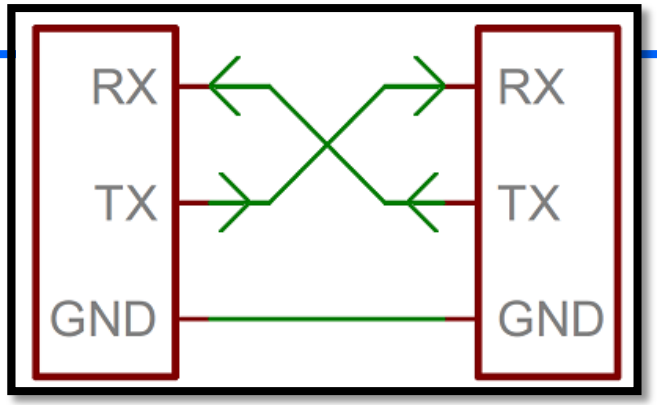
bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enable interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as parity bit

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

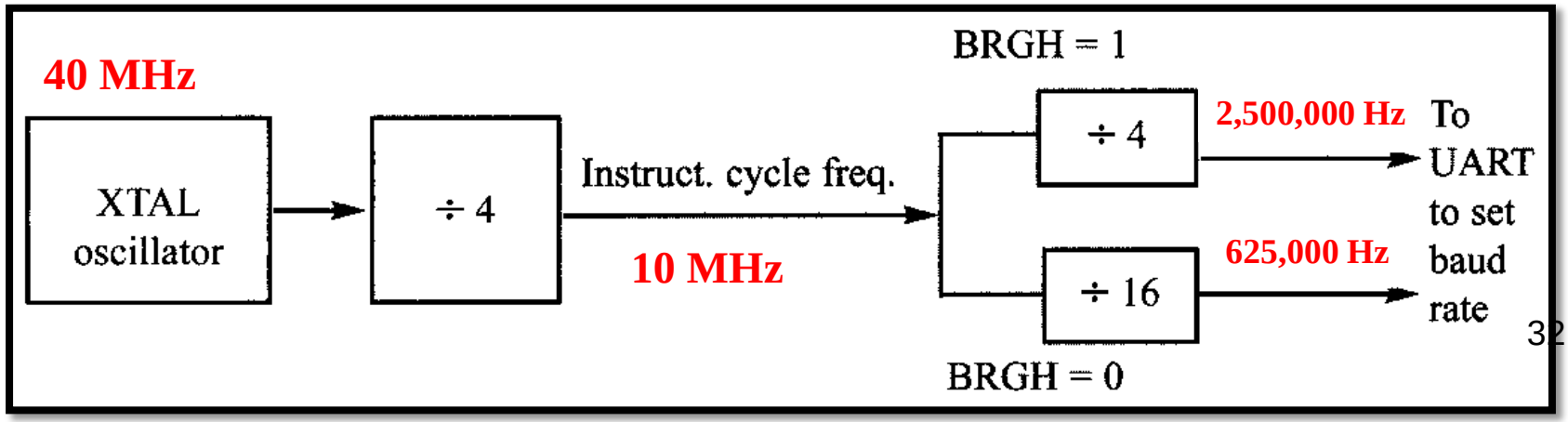
bit 1 **OERR:** Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data
This can be Address/Data bit or a parity bit, and must be calculated by user firmware.

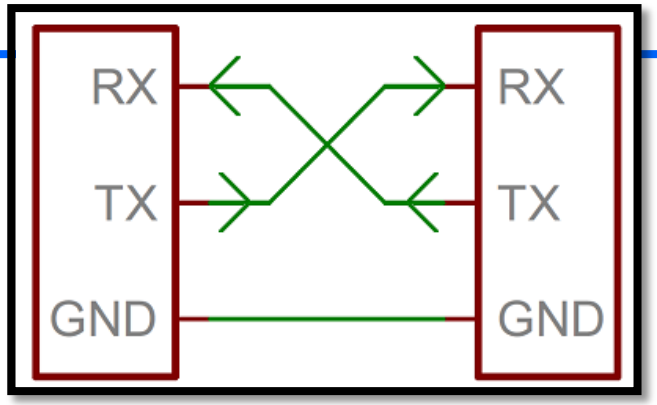
USART Async. Example



- F_{osc} = 40 MHz
- find **X** = SPBRG (0-255)
- Desired BAUD Rate = **9600 bps**
  - High Speed: BAUD = $\frac{F_{osc}}{4*4(X+1)} = \frac{F_{osc}}{16(X+1)}$
  - Low Speed: BAUD = $\frac{F_{osc}}{4*16(X+1)} = \frac{F_{osc}}{64(X+1)}$

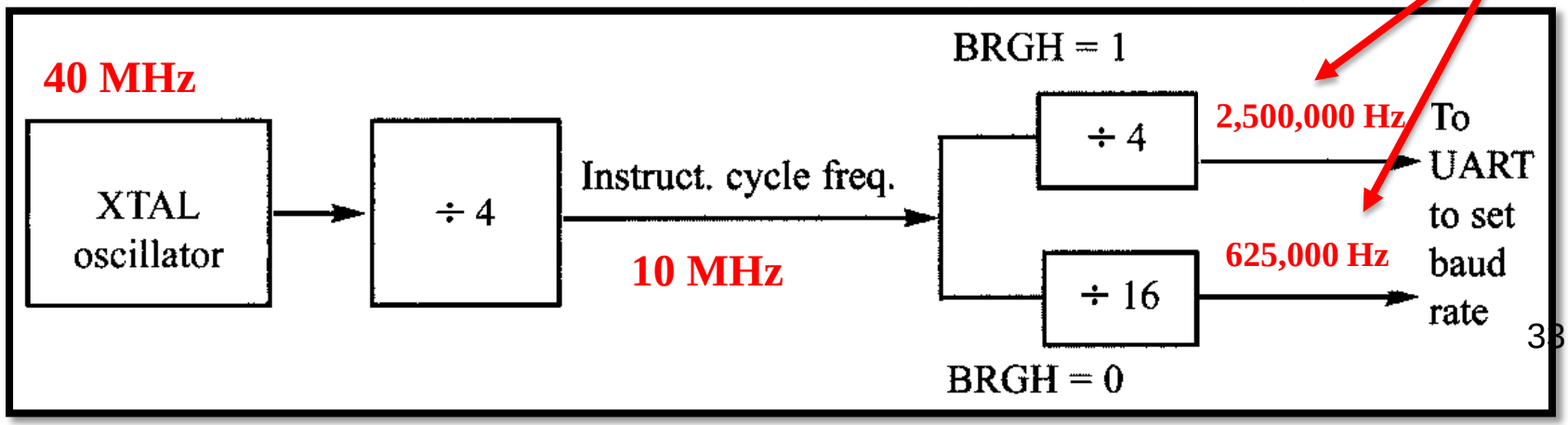


USART Async. Example



- F_{osc} = 40 MHz
- find **X** = SPBRG (0-255)
- Desired BAUD Rate = **9600 bps**
  - High Speed: $BAUD = \frac{F_{osc}}{4*4(X+1)} = \frac{F_{osc}}{16(X+1)}$
  - Low Speed: $BAUD = \frac{F_{osc}}{4*16(X+1)} = \frac{F_{osc}}{64(X+1)}$

Want to slow down to 9600



USART Async. Example

- **F_{osc} = 40 MHz,** find **X** = SPBRG (0-255)
- Desired BAUD Rate = **9600 bps**
  - High Speed: 9600 = $\frac{F_{osc}}{16(X+1)}$ → X = $\frac{40MHz}{16 * 9600}$ - 1 = **259.417**
  - Low Speed: 9600 = $\frac{F_{osc}}{64(X+1)}$ → X = $\frac{40MHz}{64 * 9600}$ - 1 = **64.104**

USART Async. Example

- **F_{osc} = 40 MHz,** find **X** = SPBRG (0-255)
- Desired BAUD Rate = **9600 bps**
  - High Speed: 9600 = $\frac{F_{osc}}{16(X+1)}$ → X = $\frac{40MHz}{16 * 9600}$ - 1 = **259.417**
  - Low Speed: 9600 = $\frac{F_{osc}}{64(X+1)}$ → X = $\frac{40MHz}{64 * 9600}$ - 1 = **64.104**

> 255

<= 255

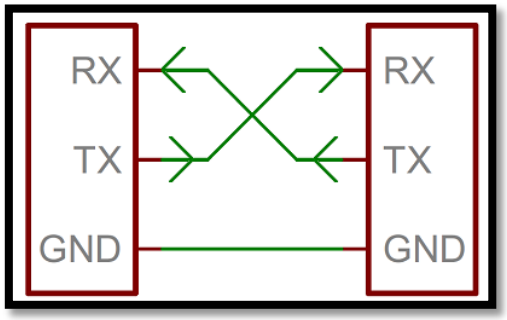
- Select ***Low Speed*** and ***SPBRG = 64***

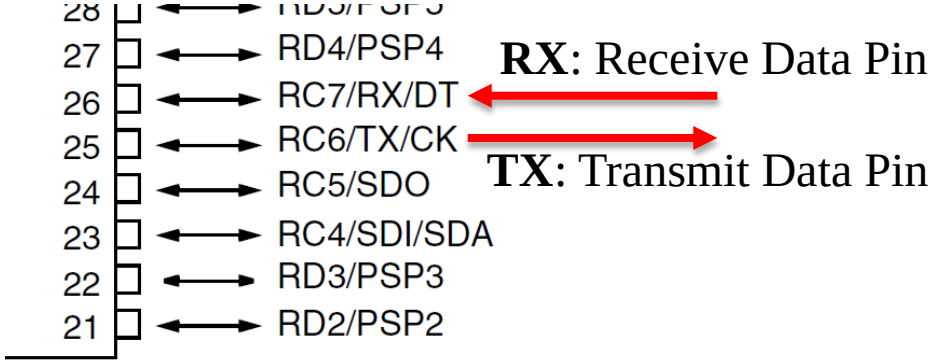
TABLE 16-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (Kbps)	Fosc = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBI valu (decir
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	2.40	-0.07	214	2.40	-0.15	162
9.6	9.62	+0.16	64	9.55	-0.54	53	9.53	-0.76	40
19.2	18.94	-1.36	32	19.10	-0.54	26	19.53	+1.73	19

USART: Async. Steps (Polling Ex.)

Reception






```
1  TRISCbits.RC7 = 1;   //RX is INPUT
2  SPBRG = 64;          //Low Speed: 9,600 BAUD for Fosc = 40 MHz
3  TXSTAbits.SYNC = 0; //Async. Mode
4  TXSTAbits.BRGH = 0; //Low Speed BAUD Rate
5  RCSTAbits.RX9 = 0;   //8-bit Reception
6  RCSTAbits.SPEN = 1; //Serial Port ENABLED (RX and TX active)
7  RCSTAbits.CREN = 1; //Enable Continuous Receiver (turned ON)
8
9  while(PIR1bits.RCIF == 0);  //Wait for incoming data
10 myVar = RCREG;              //Reading RCREG clears RCIF flag
```

USART: Async. Steps (Polling Ex.)

Reception

```
1  TRISCbits.RC7 = 1;   //RX is INPUT
2  SPBRG = 64;          //Low Speed: 9,600 BAUD for Fosc = 40 MHz
3  TXSTAbits.SYNC = 0; //Async. Mode
4  TXSTAbits.BRGH = 0; //Low Speed BAUD Rate
5  RCSTAbits.RX9 = 0;   //8-bit Reception
6  RCSTAbits.SPEN = 1; //Serial Port ENABLED (RX and TX active)
7  RCSTAbits.CREN = 1; //Enable Continuous Receiver (turned ON)
8
9  while(PIR1bits.RCIF == 0);  //Wait for incoming data
10 myVar = RCREG;              //Reading RCREG clears RCIF flag
```

PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 5 **RCIF**: USART Receive Interrupt Flag bit

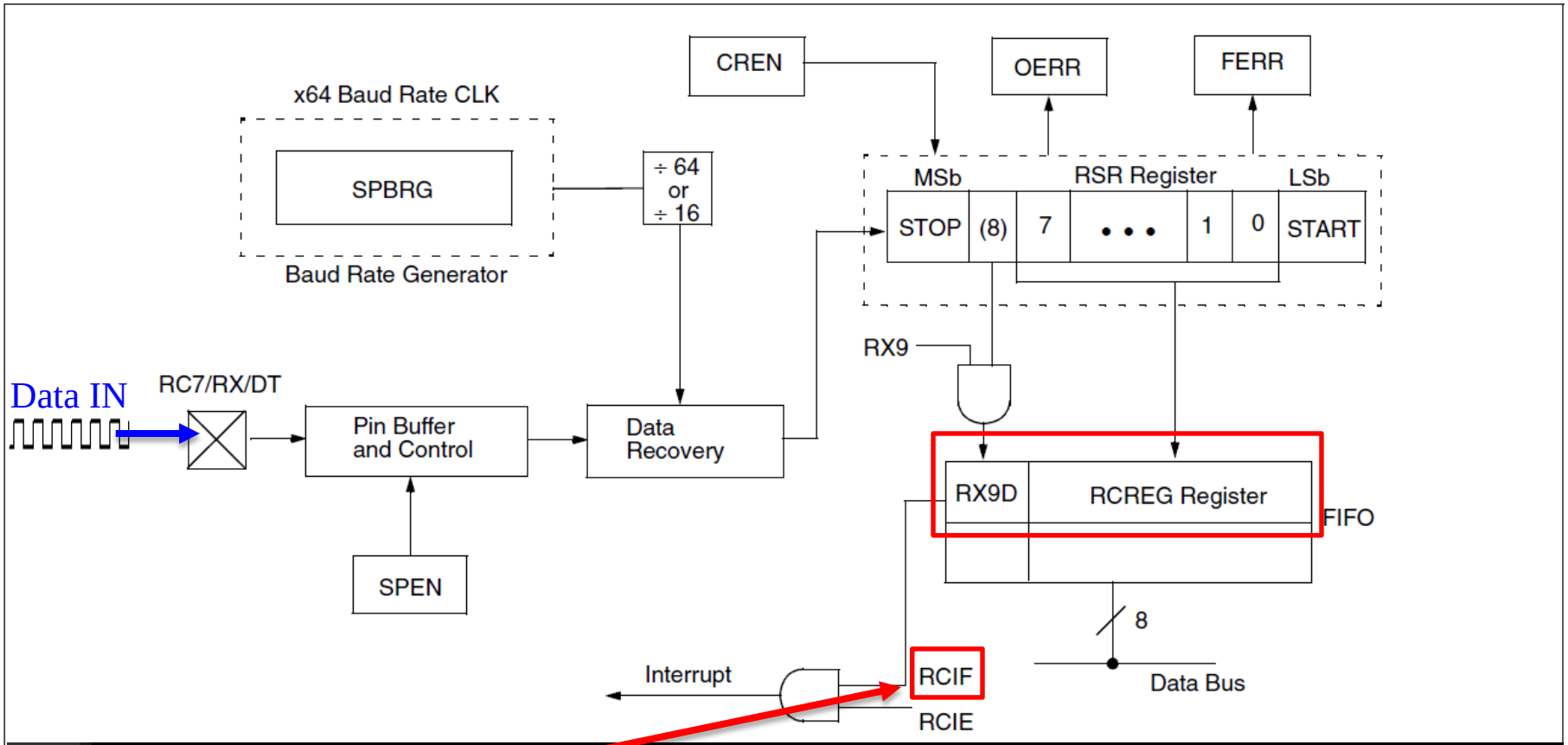
1 = The USART receive buffer, RCREG, is full (cleared when RCREG is read)

0 = The USART receive buffer is empty → not full (not every bit has arrived yet)

USART: Reception

CSE@UTA

FIGURE 16-4: USART RECEIVE BLOCK DIAGRAM




```
9  while(PIR1bits.RCIF == 0);   //Wait for incoming data
10 myVar = RCREG;               //Reading RCREG clears RCIF flag
```

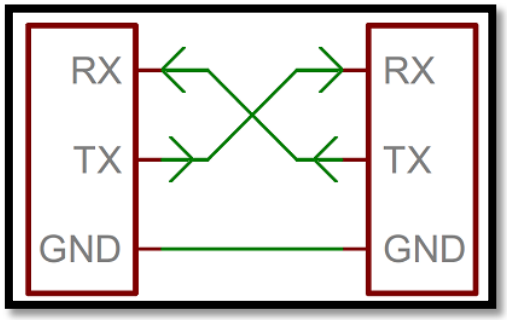
bit 5 **RCIF**: USART Receive Interrupt Flag bit

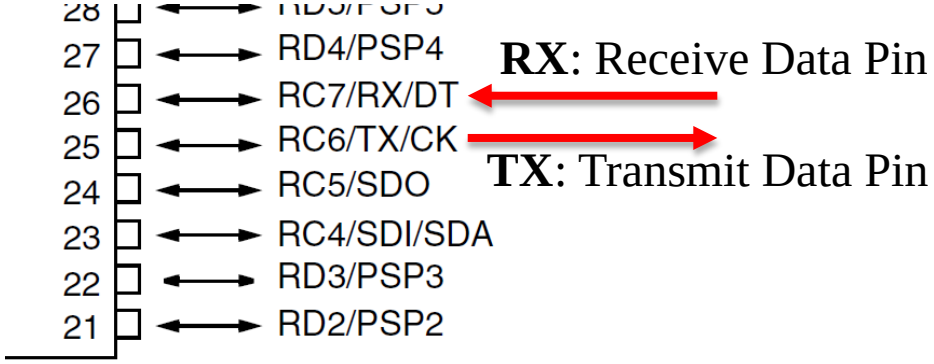
1 = The USART receive buffer, RCREG, is full (cleared when RCREG is read)

0 = The USART receive buffer is empty → not full (not every bit has arrived yet)

USART: Async. Steps (Polling Ex.)

Transmission






```
16 TRISCbits.RC6 = 0;   //TX is OUTPUT
17 SPBRG = 64;          //Low Speed: 9,600 BAUD for Fosc = 40 MHz
18 TXSTAbits.SYNC = 0; //Async. Mode
19 TXSTAbits.BRGH = 0; //Low Speed BAUD Rate
20 RCSTAbits.RX9 = 0;   //8-bit Reception
21 RCSTAbits.SPEN = 1; //Serial Port ENABLED (RX and TX active)
22 TXSTAbits.TXEN = 1; //Enable Transmitter (turned ON)
23
24 while(TXSTAbits.TRMT == 0); //Wait until possible previous
25                             //        transmission is done
26 TXREG = 'P';                //Send one 8-bit byte out
```

Typo
TXSTAbits.TX9 = 0;

USART: Async. Steps (Polling Ex.)

Transmission

```
16 TRISCbits.RC6 = 0;   //TX is OUTPUT
17 SPBRG = 64;          //Low Speed: 9,600 BAUD for Fosc = 40 MHz
18 TXSTAbits.SYNC = 0; //Async. Mode
19 TXSTAbits.BRGH = 0; //Low Speed BAUD Rate
20 RCSTAbits.RX9 = 0;   //8-bit Reception
21 RCSTAbits.SPEN = 1; //Serial Port ENABLED (RX and TX active)
22 TXSTAbits.TXEN = 1; //Enable Transmitter (turned ON)
23
24 while(TXSTAbits.TRMT == 0); //Wait until possible previous
25                             //        transmission is done
26 TXREG = 'P';                //Send one 8-bit byte out
```

Typo
TXSTAbits.TX9 = 0;

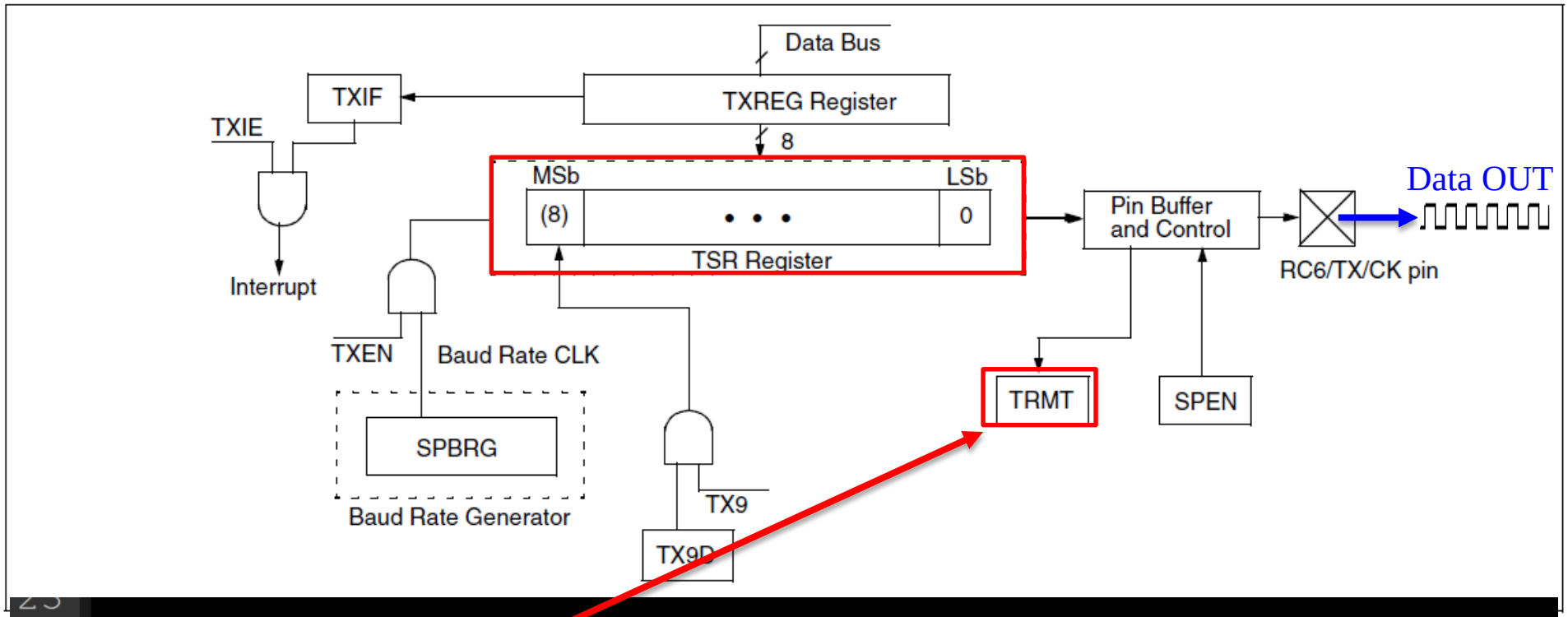
TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full → not empty (still has something in it)

USART: Transmission

FIGURE 16-1: USART TRANSMIT BLOCK DIAGRAM




```
24 while(TXSTAbits.TRMT == 0); //Wait until possible previous
25                             //        transmission is done
26 TXREG = 'P';                //Send one 8-bit byte out
```

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full → not empty (still has something in it)

Additional Information

- SPI
  - https://www.youtube.com/watch?v=9hMsNOwY5AQ
  - https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi
  - http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf
  - https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
- I2C
  - https://www.youtube.com/watch?v=fm13tIe5wSc
  - https://learn.sparkfun.com/tutorials/i2c
  - http://ww1.microchip.com/downloads/en/DeviceDoc/i2c.pdf
  - https://en.wikipedia.org/wiki/I%C2%B2C
- USART – Async.
  - http://ww1.microchip.com/downloads/en/DeviceDoc/USART.pdf