Chuyên đề II
Vi điều khiển và ứng dụng

Serial Communication

Truyền thông nối tiếp

- 2 the Universal Asynchronous Receiver and Transmitters (UARTs)
- 2 the SPI synchronous serial interfaces
- 2 the I 2 C synchronous serial interfaces

Synchronous serial

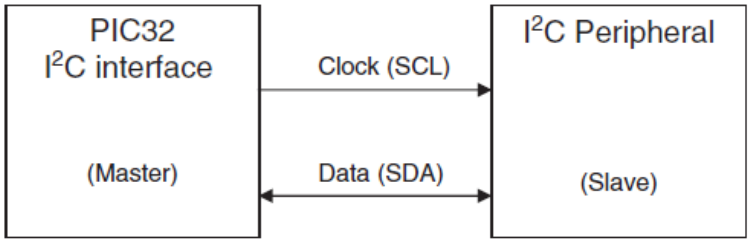


Figure 8.1: I²C interface block diagram.

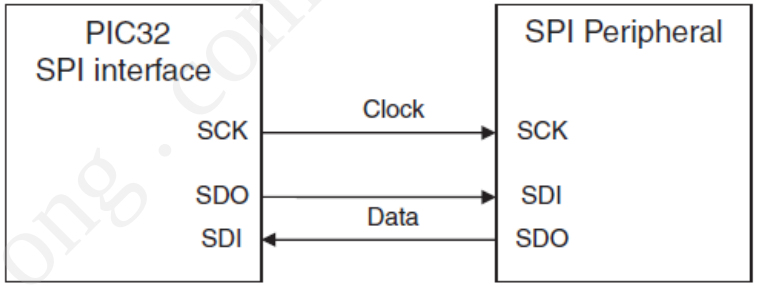


Figure 8.2: SPI interface block diagram.

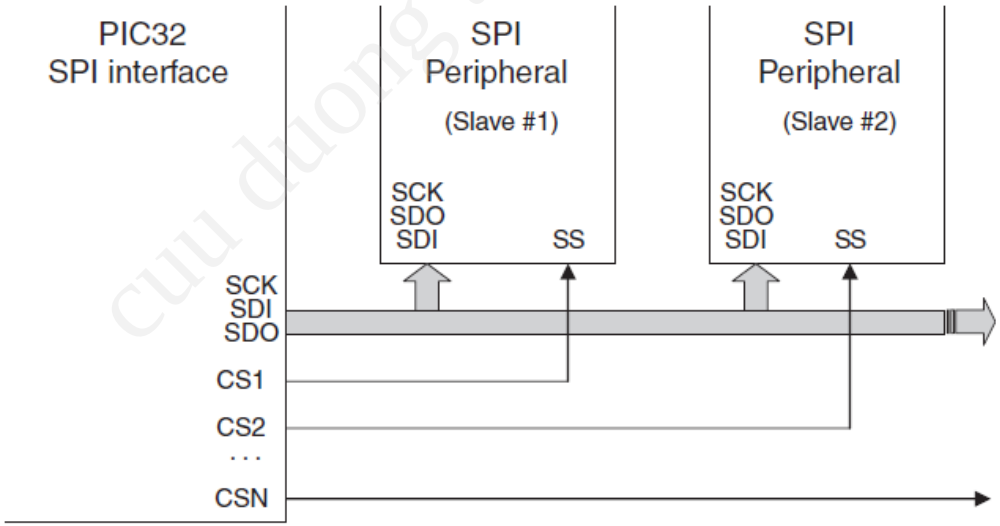
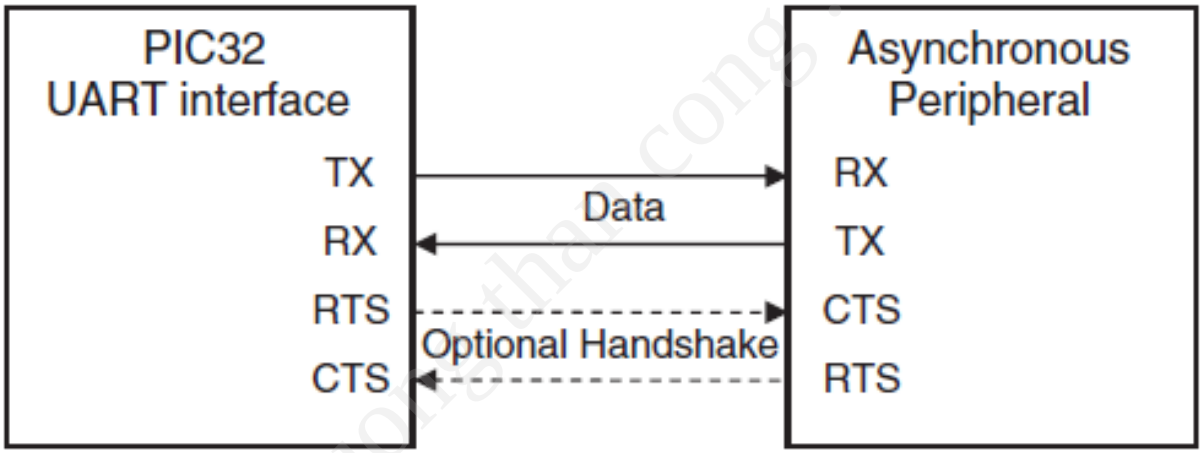


Figure 8.3: SPI bus block diagram.

Asynchronous Serial Interfaces



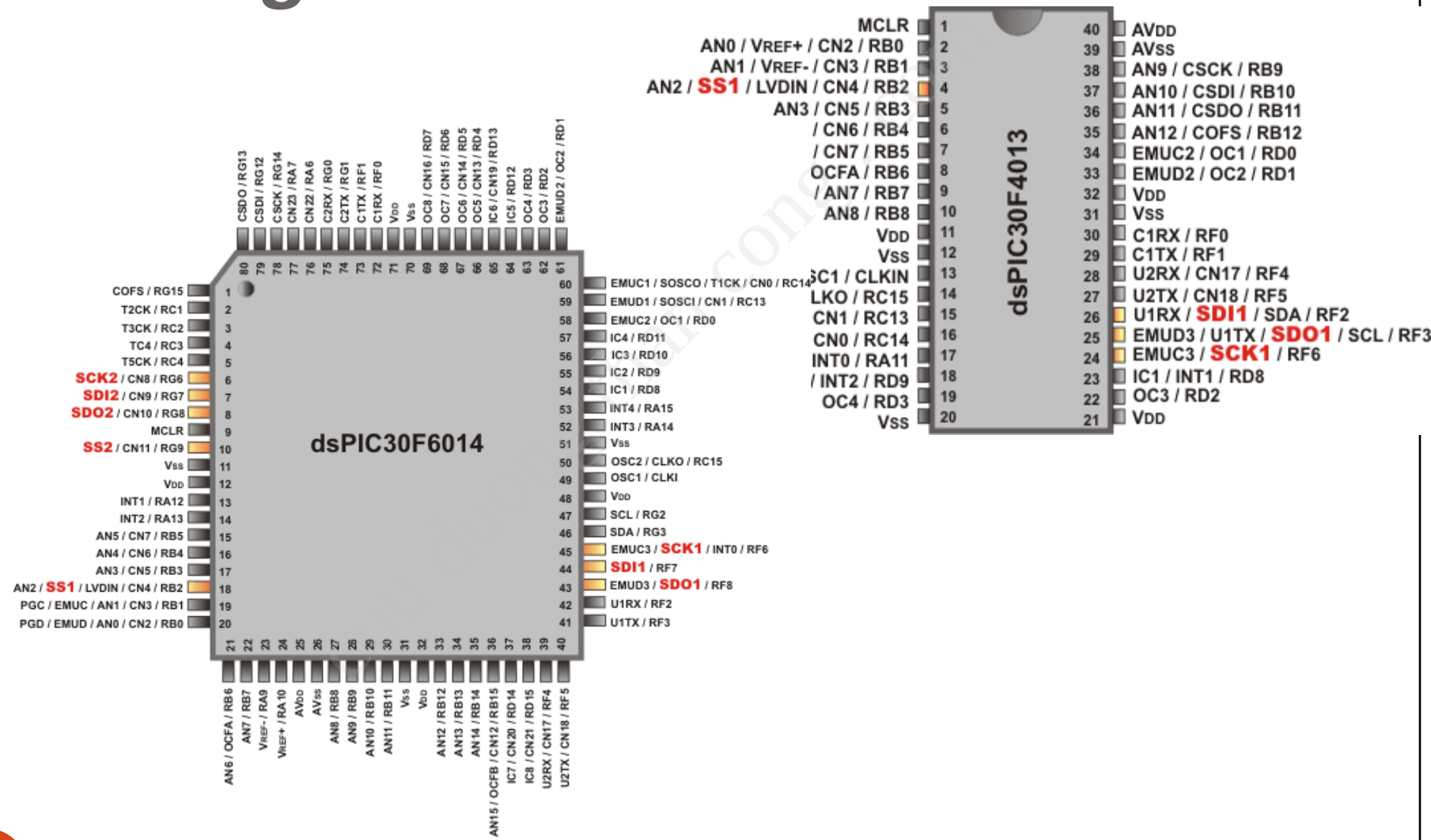
Some comparisons

	Synchronous		Asynchronous
Peripheral	**SPI**	**I^2C**	**UART**
Max bit rate	20 Mbit/s	1 Mbit/s	500 kbit/s
Max bus size	Limited by number of pins	128 devices	Point to point (RS232), 256 devices (RS485)
Number of pins	3 + n × CS	2	2(+2)

More

	Synchronous		Asynchronous
Peripheral	**SPI**	**I²C**	**UART**
Pros	Simple, low cost, high speed	Small pin count, allows multiple masters	Longer distance (use transceivers for improved noise immunity)
Cons	Single master, short distance	Slowest, short distance	Requires accurate clock frequency
Typical application	Direct connection to many common peripherals on same PCB	Bus connection with peripherals on same PCB	Interface with terminals, personal computers, and other data acquisition systems
Examples	Serial EEPROMs (25CXXX series), MCP320X A/D converter, ENC28J60 Ethernet controller, MCP251X CAN controller . . .	Serial EEPROMs (24CXXX series), MCP98XX temperature sensors, MCP322x A/D converters . . .	RS232, RS422, RS485, LIN bus, MCP2550 IrDA interface . . .

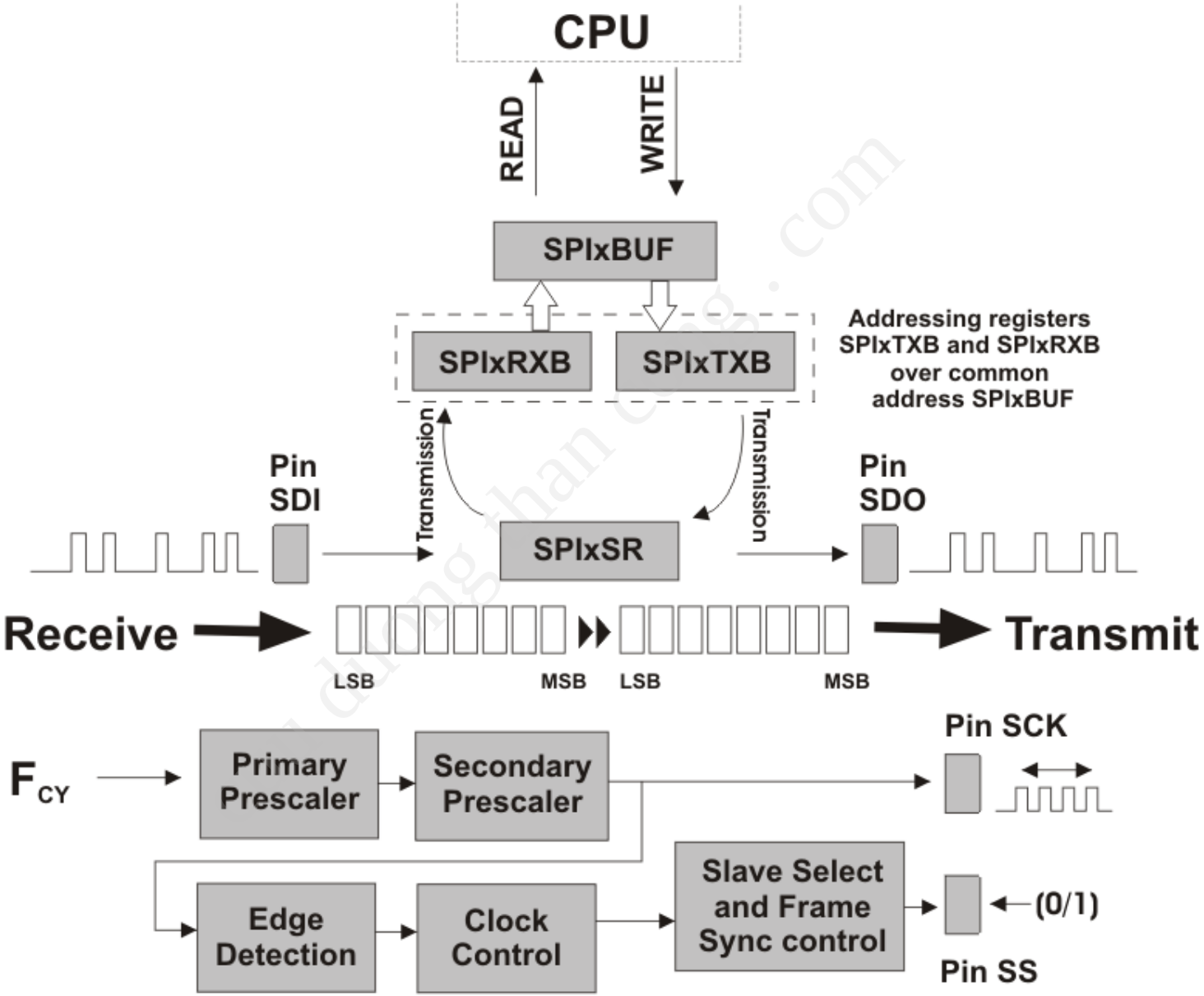
Synchronous Communication Using the SPI Modules



SPI applications

- Interfacing with memory devices
  - Serial EEPROMs - e.g., 25xx256
- Interfacing with codecs
- Control Ports or PCM Data
- Interfacing with communication chips

  Bluetooth
- Boot Loader



Giải thích các bit điều khiển

NAME	ADR	15	14	13	12-7	6	5	4	3	2	1	0	RESET STATE
SPI1STAT	0x0220	SPIEN	-	SPISIDL	-	SPIROV	-	-	-	-	SPITBF	SPIRBF	0x0000

Table 9-3 Status and control register SPI1STAT

```
SPIEN - SPI enable bit (SPIEN=0 disables module, SPIEN=1 enables module)
SPISIDL - Stop in IDLE mode bit (SPISIDL=0 continue operation, SPISIDL=1 discontinue operation)
SPIROV - Receive overflow flag bit
SPITBF - SPI transmit buffer full status bit (SPITBF=0 transmit started, SPIxTBF empty,
         SPITBF=1 transmit not yet started, SPIxTBF is full)
SPIRBF - SPI receive buffer full status bit (SPIRBF=0 receive is not complete
         SPIxRXB is empty, SPIRBF=1 receive complete SPIxRXB is full)
```

Bít điều khiển

NAME	ADR	15	14	13	12	11	10	9	8	7	6	5
SPI1CON	0x0222	-	FRMEN	SPIFSD	-	DISSDO	MODE16	SMP	CKE	SSEN	CKP	MSTEN

Table 9-4a Control register SPI1CON

4	3	2	1	0	RESET STATE
SPRE<2:0>			PPRE<1:0>		0x0000

Table 9-4b continued

FRMEN – Framed SPI support bit
SPIFSD – Frame sync pulse direction control on SSx pin bit
(SPIFSD=0 frame sync pulse output (master), SPIFSD=1 frame sync pulse input (slave))
DISSDO – Disable SDOx pin bit
MODE16 – Word/byte communication select bit (MODE16=0 8-bit mode, MODE16=1 16-bit mode)
SMP – SPI data input sample phase bit (Master mode: SMP=0 input data sampled at middle of data output time, SMP=1 input data sampled at end of data output time;
Slave mode: SMP must be cleared)

CKE – SPI clock edge select bit (CKE=0 serial output data changes on transition from IDLE clock state to active clock state, CKE=1 serial output data changes on transition from active clock state to IDLE clock state)

SSEN – Slave select enable bit
(SSEN=0 SS1 pin not used by module, SSEN=1 SS1 pin used for slave mode)

CKP – Clock polarity select bit (CKO=0 IDLE state for clock is a low level, active state is a high level, CKO=1 IDLE state for clock is a high level, active state is a low level)

MSTEN – Master mode enable bit (MSTEN=0 slave mode, MSTEN=1 master mode)
SPRE<2:0> – Secondary prescale (master mode) bits
000 – secondary prescale 8:1

SPI overview

- Serial transmission and reception of 8-bit or 16-bit data
- Full-duplex, synchronous communication
- Compatible with Motorola's SPI and SIOP interfaces
- 3-wire interface
- Supports 4 different clock formats and serial clock speeds up to 10 Mbps
- Buffered Transmission and Reception

SPI - Master / Slave

- SPI module can be configured as Master or Slave
- In any SPI data transfer, there is a single Master and a single Slave
  - Selected by MSTEN bit, SPIxCON<5>
  - Master generates serial clock pulse (on SCK pin)
  - SCK frequency determined by Primary
- Prescaler bits (PPRE) and Secondary Prescaler (SPRE) bits in SPIxCON register

Fsck = Fcy / (PPRE * SPRE)

$$F_{SCK} = \frac{F_{CY}}{primary\_prescaler \bullet secondary\_prescaler}$$

Examples of the SCKx frequencies as functions of the primary and secondary prescaler settings are shown in Table 9-1.

FCY = 30 MHZ		SECONDARY PRESCALER SETTINGS				
		1:1	2:1	4:1	6:1	8:1
Primary prescaler settings	1:1	30 000	15 000	7 500	5 000	3 750
	4:1	7 500	3 750	1 875	1 250	938
	16:1	1 875	938	469	313	234
	64:1	469	234	117	78	59
Fcy = 5 Mhz						
Primary prescaler settings	1:1	5 000	2 500	1 250	833	625
	4:1	1 250	625	313	208	156
	16:1	313	156	78	52	39
	64:1	78	39	20	13	10

Table 9-1 SCKx frequencies

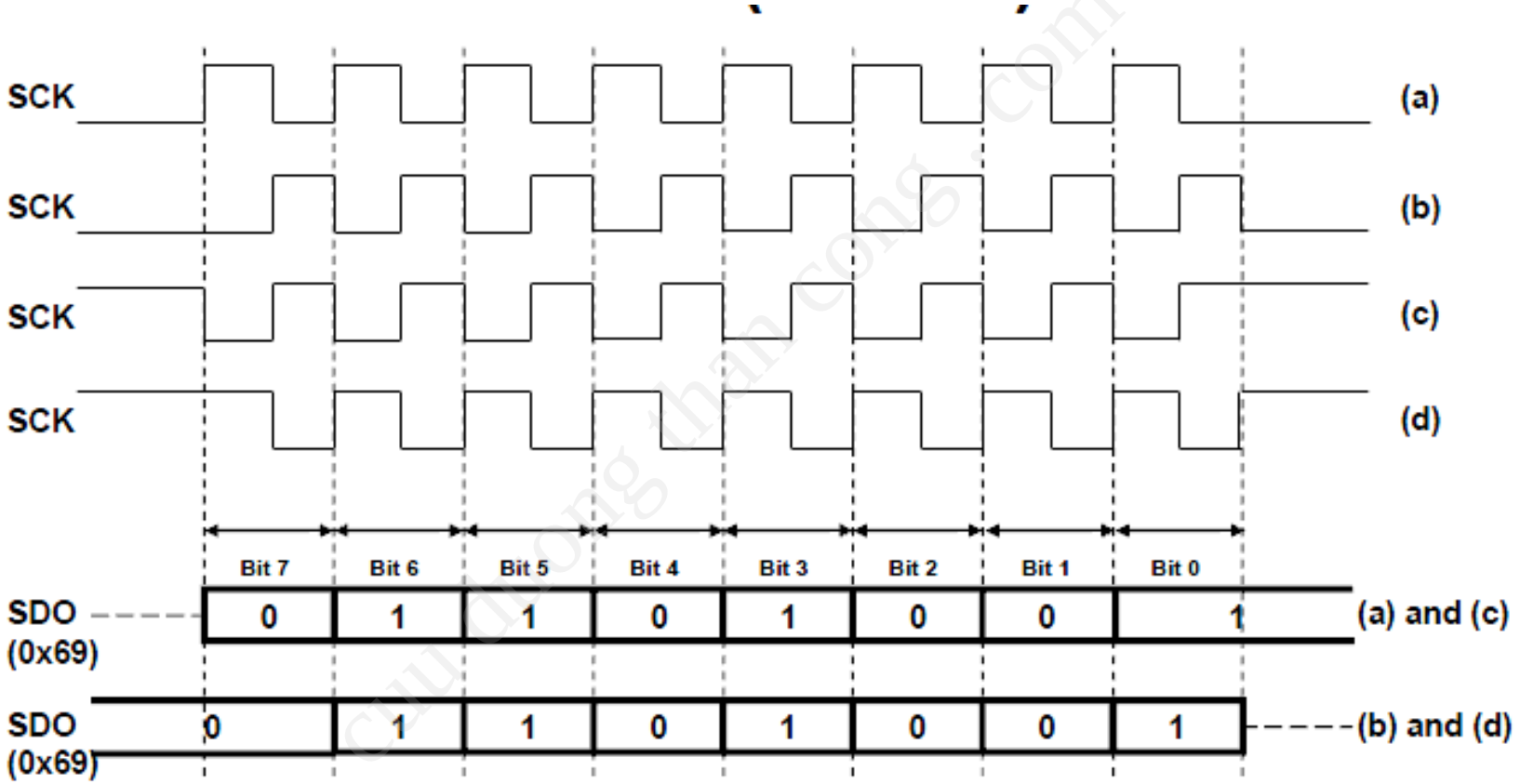
SPI - Serial Clock Formats

- 4 clock formats - set by CKP and CKE bits

in the SPIxCON register

- SCK is low when module is idle, SDO changes on clock going high (CKP=0, CKE=0)
- SCK is low when module is idle, SDO changes on clock going low (CKP=0, CKE=1)
- SCK is high when module is idle, SDO changes on clock going low (CKP=1, CKE=0)
- SCK is high when module is idle, SDO changes on clock going high (CKP=1, CKE=1)

Ví dụ



(a) CKP = 0, CKE = 0 (b) CKP = 0, CKE = 1 (c) CKP = 1, CKE = 0 (d) CKP = 1, CKE = 1

SPI - Transmission

- Module is enabled by setting SPIEN bit in the SPIxSTAT register
  - Transmission begins when data is written into the Master's Transmit Buffer
  - SCK pulses are generated by the Master only when SPIxSR contains data
  - Transmission can be disabled by setting the DISSDO bit in the SPIxCON register
- SPIxBUF is buffered
  - You can write SPIxBUF while data is being shifted out through SPIxSR
  - SPITBF bit in the SPIxSTAT register indicates that the Transmit Buffer is full
  - Wait until SPITBF = 0 to write data
  - Transmission of the new data starts as soon as SPIxSR is idle

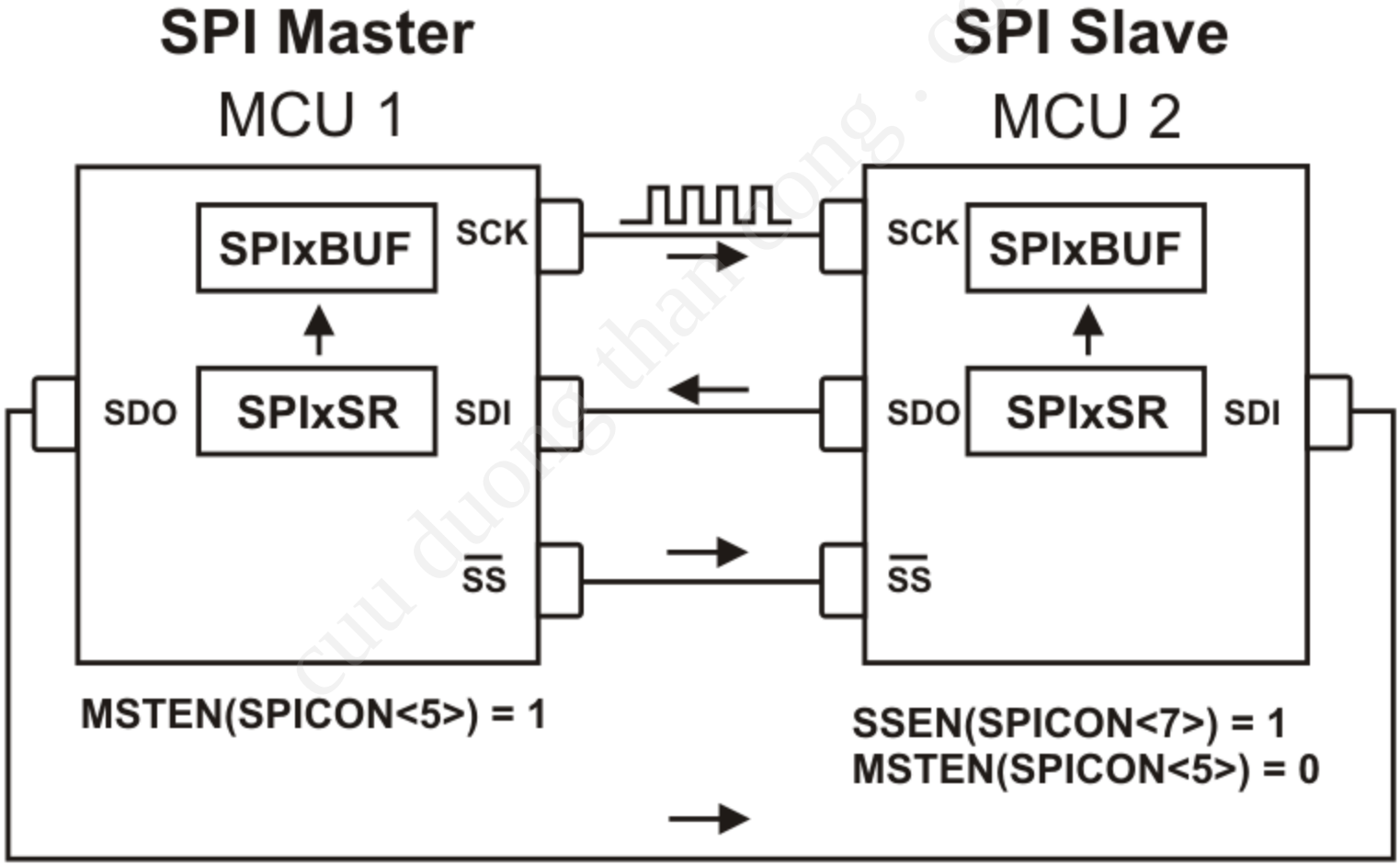
SPI - Reception

- Reception occurs concurrently with the transmission
  - When all bits of data have been shifted in through SPIxSR, SPIxSR contents are transferred to Receive Buffer
  - SPI interrupt (indicated by SPIIF bit and enabled by SPIIE bit) is generated so that buffer can be read
- SPIxBUF subject to Receive Overflow
- SPIRBF bit in the SPIxSTAT register = 1 indicates that the Receive Buffer is full
- SPIxBUF must be read before new data is completely shifted in

When receive overflow occurs...

- New data not transferred to Receive Buffer
- SPIROV bit in SPIxSTAT is set

Configuration



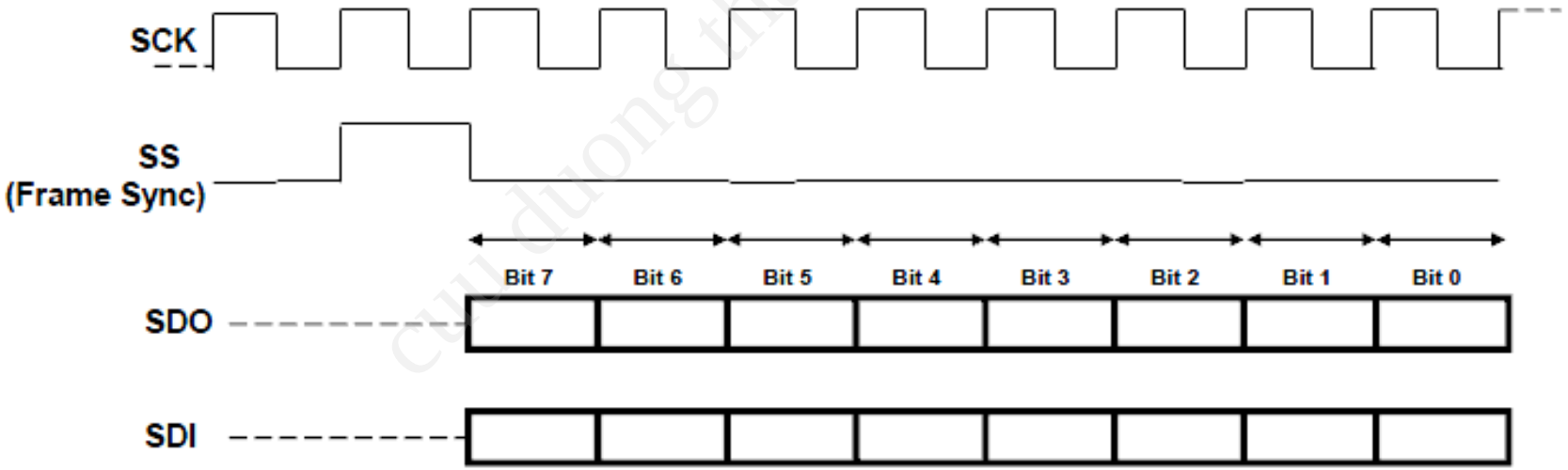
SPI - Data Sizes

- 8-bit and 16-bit data communication
- SPI operation is identical for both data sizes, except number of bits transmitted
  - For 8-bit data, Master generates 8 SCK pulses
  - For 16-bit data, Master generates 16 SCK pulses
- 16-bit operation is selected by setting the

MODE16 bit in the SPIxCON register

SPI - Framed SPI

- SPI supports Frame Synchronization
  - Enabled by setting FRMEN bit in the SPIxCON register
  - SCK pulses are continuous in this mode



SPI - Framed SPI

- Frame Master generates Frame Sync pulses
  - Frame Master or Slave mode is selected by clearing or setting the SPIFSD bit in the SPIxCON register
  - Shifting of data starts only after a Frame Sync pulse is generated on the SS pin
- 4 possible Framed SPI modes
  - SPI Master, Frame Master
  - SPI Master, Frame Slave
  - SPI Slave, Frame Master
  - SPI Slave, Frame Slave

SPI - chức năng phụ

- Slave Select (SS) pin functionality
  - In this mode, the Slave functions only as
  - long as the SS pin is driven low
  - Enabled by setting SSEN bit in the SPIxCON register
- Slave Wake-up from SLEEP
  - Since SCK pulses are provided by the Master, SPI Slave can function in SLEEP
  - Slave Reception wakes up the device from SLEEP

Ví dụ ghép nối với Serial EEROM 25L256

```
// 1. init the SPI peripheral
#define SPI_CONF 0 x 8120 // SPI on, 8-bit master, CKE=1,CKP=0
TCSEE = 0; // make SSEE pin output
CSEE = 1; // de-select the EEPROM
SPI2CON = SPI_CONF; // select mode and enable


    // send one byte of data and receive one back at the same time
    int writeSPI2( int i)
    {
        SPI2BUF = i; // write to buffer for TX
        while( !SPI2STATbits.SPIRBF); // wait for transfer complete
        return SPI2BUF; // read the received value
    }//writeSPI2
```

Đọc Serial ROM

```
// 25LC256 Serial EEPROM commands
#define SEE_WRSR 1 // write status register
#define SEE_WRITE 2 // write command
#define SEE_READ 3 // read command
#define SEE_WDI 4 // write disable
#define SEE_STAT 5 // read status register
#define SEE_WEN 6 // write enable
```

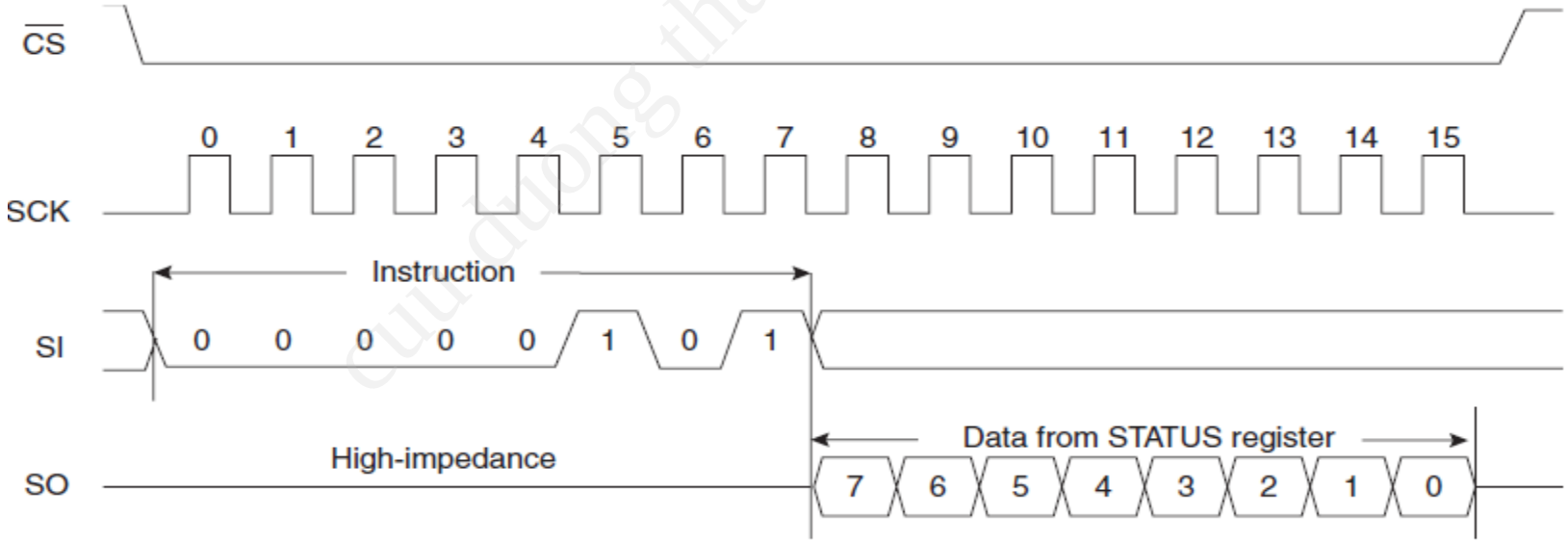


Figure 8.7: The complete Read Status Register command timing sequence.

Ví dụ chương trình đầy đủ

```
int writeSPI2( int i)
{
    SPI2BUF = i;                        // write to buffer for TX
    while( !SPI2STATbits.SPIRBF);       // wait for transfer complete
    return SPI2BUF;                     // read the received value
}//writeSPI2

main ()
{
  int i;
  // 1. init the SPI peripheral
  TCSEE = 0;                            // make SSEE pin output
  CSEE = 1;                             // de-select the Serial EEPROM
  SPI2CON = SPI_CONF;                   // select mode and enable SPI2
  SPI2BRG = SPI_BAUD;                   // select clock speed
  // main loop
  while( 1)
  {
    // 2. Check the Serial EEPROM status
      CSEE = 0;                         // select the Serial EEPROM
      writeSPI2( SEE_STAT);             // send a READ STATUS COMMAND
      i=writeSPI2( 0);                  // send/receive
      CSEE=1;                           // deselect terminate command
```

Writing/read Data to the EEPROM

```
// send a Write command
CSEE = 0; // select the Serial EEPROM
    writeSPI2( SEE_WRITE); // send command, ignore data
    writeSPI2( ADDR_MSB); // send MSB of memory address
    writeSPI2( ADDR_LSB); // send LSB of memory address
    writeSPI2( data); // send the actual data
// send more data here to perform a page write
CSEE = 1; // start actual EEPROM write cycle
// send a Write command
CSEE = 0; // select the Serial EEPROM
    writeSPI2( SEE_READ); // send command, ignore data
    writeSPI2( ADDR_MSB); // send MSB of memory address
    writeSPI2( ADDR_LSB); // send LSB of memory address
    data=writeSPI2( 0); // send dummy, read data
// read more data here sequentially incrementing the address
CSEE = 1; // terminate the read sequence
```

Read 32bit values

```
int readStatus( void)
{
  // Check the Serial EEPROM status
  int i;
  CSEE = 0;
  writeSPI2( SEE_STAT);
  i = writeSPI2( 0);
  CSEE = 1;
  return i;
} // readStatus
```

```
int readSEE( int address)
{ // read a 32-bit value starting at an even address

  int i;
  // wait until any work in progress is completed
  while ( readStatus() & 0x1); // check WIP

  // perform a 16-bit read sequence (two byte sequential read)
  CSEE = 0;                          // select the Serial EEPROM
  writeSPI2( SEE_READ);              // read command
  writeSPI2( address >>8);           // address MSB first
  writeSPI2( address & 0xfc);        // address LSB (word aligned)
  i = writeSPI2( 0);                 // send dummy, read msb
  i = (i<<8)+ writeSPI2( 0);         // send dummy, read lsb
  i = (i<<8)+ writeSPI2( 0);         // send dummy, read lsb
  i = (i<<8)+ writeSPI2( 0);         // send dummy, read lsb
  CSEE = 1;
  return ( i);
}// readSEE
```

Write 32bit values

```
void writeEnable( void)
  {
  // send a Write Enable command
  CSEE = 0;                    // select the Serial EEPROM
  writeSPI2( SEE_WEN);         // write enable command
  CSEE = 1;                    // deselect to complete the command
}// writeEnable

void writeSEE( int address, int data)
{ // write a 32-bit value starting at an even address

  // wait until any work in progress is completed

  while ( readStatus() & 0x1) // check the WIP flag

  // Set the Write Enable Latch
  writeEnable ();

  // perform a 32-bit write sequence (4 byte page write)
  CSEE = 0;                    // select the Serial EEPROM
  writeSPI2( SEE_WRITE);       // write command
  writeSPI2( address>>8);      // address MSB first
  writeSPI2( address & 0xfc); // address LSB (word aligned)
  writeSPI2( data >>24);       // send msb
  writeSPI2( data >>16);       // send msb
  writeSPI2( data >>8);        // send msb
  writeSPI2( data);            // send lsb
  CSEE = 1;
}// writeSEE
```

DAC ví dụ MCP4921

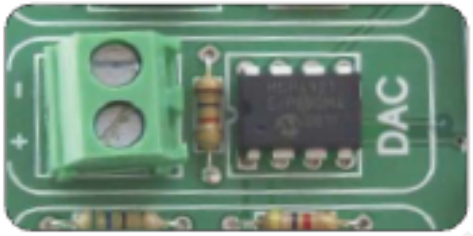


Figure 17 DAC

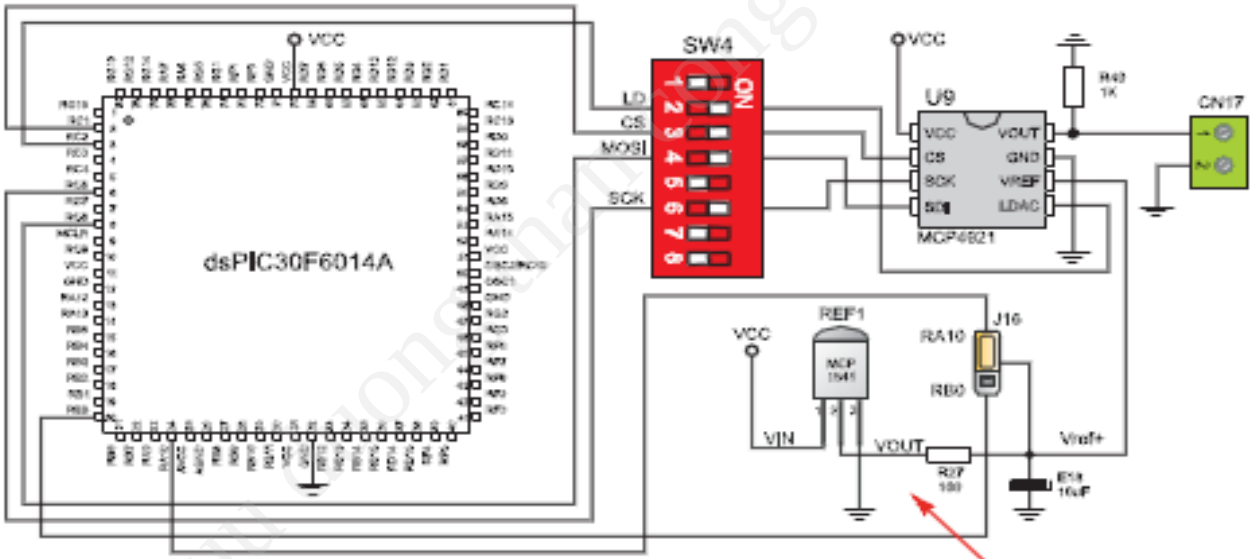


Figure 18 DAC circuit diagram

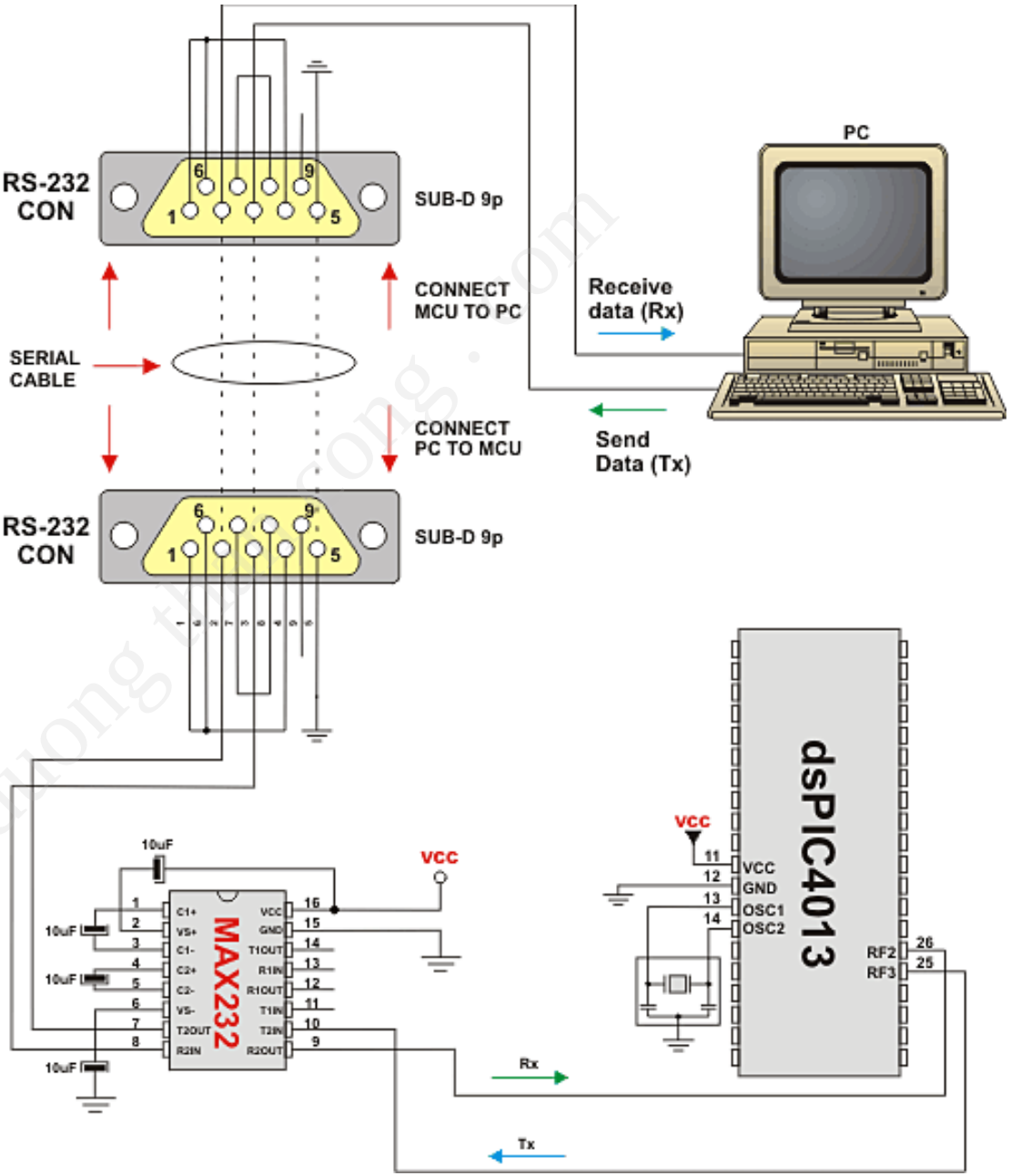
Ví dụ phần mềm DAC

```
const char CS_PIN = 0;
unsigned int value;

void InitMain() {
  ADPCFG = 0xFFFF;                    // Set AN pins as digital

  Spi_Init();                         // Initialize SPI module

  TRISF.CS_PIN = 0;                   // Set CS pin as output
}//~

// DAC increments (0..4095) --> output voltage (0..Vref)
void DAC_Output(unsigned int valueDAC) {
 char temp;

  PORTF.CS_PIN = 0;                   // Select DAC module

  // Send 2 bytes of valueDAC variable
  temp = (valueDAC >> 8) & 0x0F;      // Prepare hi-byte for transfer
                                      // It's a 12-bit number, so only
                                      // lower nibble of high byte is used
  temp |= 0x30;                       // Set MCP4921 control bits
  Spi_Write(temp);                    // Send data via SPI

  temp = valueDAC;                    // Prepare lo-byte for transfer
  Spi_Write(temp);                    // Send data via SPI

  PORTF.CS_PIN = 1;                   // Deselect DAC module
}//~
```

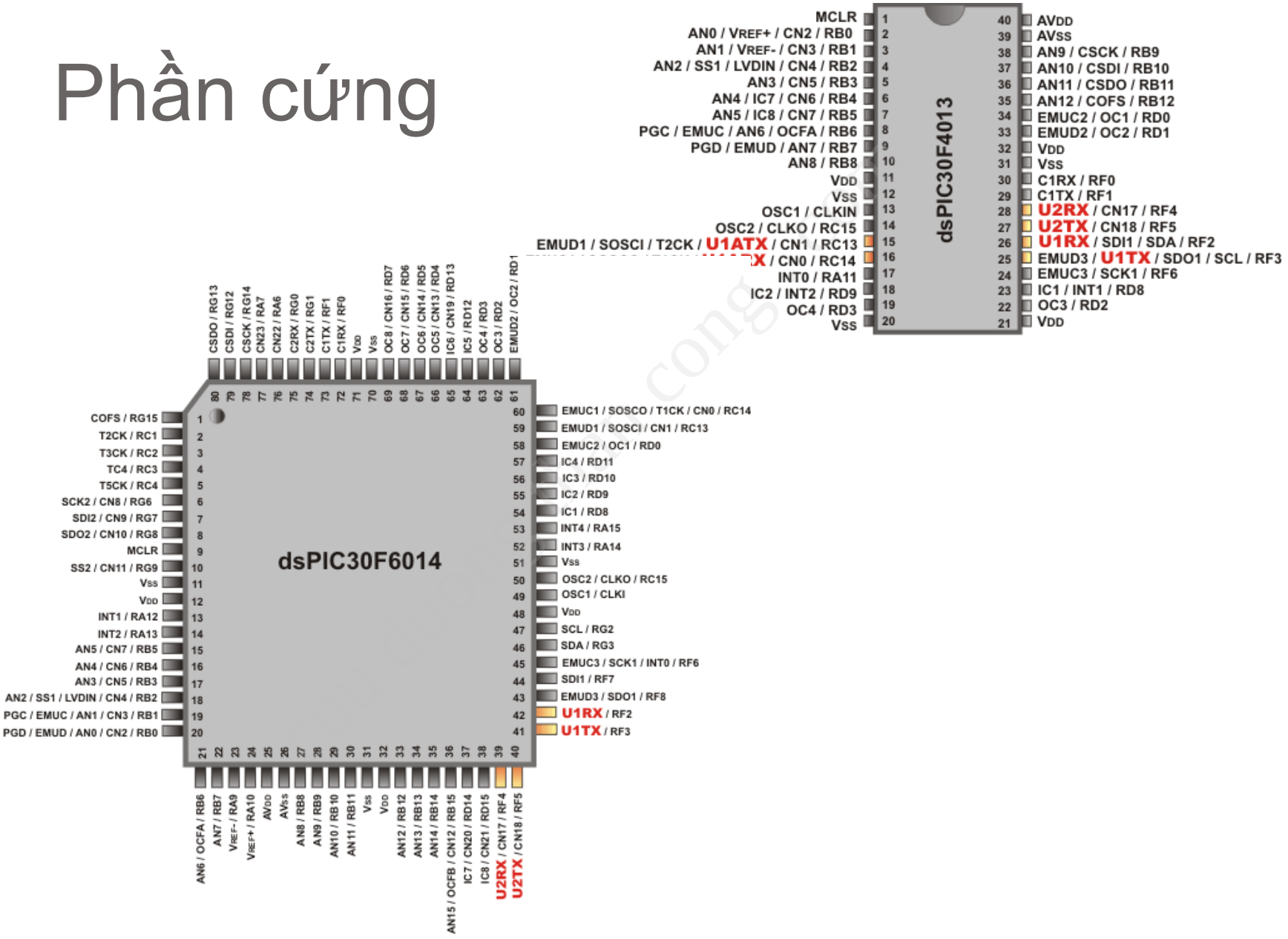
Serial Communications using the dsPIC30F UART Module

- Serial transmission and reception
- 8-bit or 9-bit data
- Full-duplex, asynchronous communication
- Support for communication protocols such
- as RS-232, RS-422, RS-485 and LIN
- 4-deep Transmit and Receive buffers
- Transmit and Receive interrupts
- Error detection
- Support for receiver addressing

Cấu hình truyền thống



Phần cứng



Các thanh ghi

NAME	ADR	15	14	13	12	11	10	9
U1MODE	0x020C	UARTEN	-	USIDL	-	-	ALTIO	-
U1STA	0x020E	UTXISEL	-	-	-	UTXBRK	UTXEN	UTXBF
U1TXREG	0x0210	-	-	-	-	-	-	-
U1RXREG	0x0212	-	-	-	-	-	-	-
U1BRG	0x0214	Baud-rate generator prescale						
U2MODE	0x0216	UARTEN	-	USIDL	-	-	ALTIO	-
U2STA	0x0218	UTXISEL	-	-	-	UTXBRK	UTXEN	UTXBF
U2TXREG	0x021A	-	-	-	-	-	-	-
U2RXREG	0x021C	-	-	-	-	-	-	-
U2BRG	0x021E	Baud-rate generator prescale						

Table 10-2 Description of UART module registers

8	7	6	5	4	3	2	1	0	RESET STATE
-	WAKE	LPBACK	ABAUD	-	-	PDSEL<1:0>		STSEL	0x0000
TRMT	URXISEK<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA	0X0000
UTX8	Transmit register								0x00uu
Transmit register									0x00uu
-	WAKE	LPBACK	ABAUD	-	-	PDSEL<1:0>		STSEL	0x0000
TRMT	URXISEK<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA	0X0000
UTX8	Transmit register								0x00uu
Transmit register									0x00uu

Giải thích thanh ghi

```
UARTEN - UART enable bit (UARTEN=0 UART is disabled, UARTEN=1 UART is enabled)
USIDL - Stop in IDLE mode bit (UISDL=0 continue operation in IDLE mode,
        USIDL=1 discontinue operation in IDLE mode)
ALTIO - UART alternate I/O selection bit (ALTIO=0 UART communicates using UxTX
        and UxRX I/O pins, ALTIO=1 UART communicates using UxATX and UxARX I/O pins)
WAKE - Enable wake-up on START bit detect during SLEEP mode bit
LPBACK - UART loop back mode select bit
ABAUD - Auto baud enable bit
PDSEL<1:0> - Parity and data selection bits
      00 - 8-bit data, no parity
      01 - 8-bit data, even parity
      10 - 8-bit data, odd parity
      11 - 9-bit data, no parity
STSEL - STOP  selection bit (STSEL=0 one STOP bit, STSEL=1 two STOP bits)
UTXISEL - Transmission interrupt mode selection bit (UTXISEL=0 interrupt when a character is
tranferred to the transmit shift register, UTXISEL=1 interrupt when a character is tranferred
to the transmit shift register and the  transmit buffer becomes empty)
```

Thanh ghi (tiếp)

```
UTXBRK - Transmit break bit ( UTXBRK=0 UxTX pin operates normally,
         UTXBRK=1 UxTX pin is driven low, regardless of transmitter state)
UTXEN - Transmit enable bit (UTXEN=0 UART transmitter disabled,
        UTXEN=1 UART transmitter enabled)
UTXBF - Transmit buffer full status bit (UTXBF=0 transmit buffer is not full,
        UTXBF=1 Transmit buffer is full)
TRMT - Transmit shift register is empty bit (TRMT=0 transmit shift register is not empty,
transmission in progress, TRMT=1 transmit shift register is empty, transmission completed)

URXISEL<1:0> - Receive interrupt mode selection bits
        0x - interrupt flag bit is set when a charatcer is received
        10 - interrupt flag bit is set when receive buffer is ¾ full (3 locations full)
        11 - interrupt flag bit is set when receive buffer is full (all 4 locations full)

ADDEN - Address character detect
(ADDEN=0 address detect mode disabled, ADDEN=1 address detect mode enabled)

RIDLE - Receiver IDLE bit
(RIDLE=0 UxRSR not empty, data is being received, RIDLE=1 receiver is  IDLE)

PERR - Parity error status bit
FERR - Framing error status bit
OERR - Recive buffer overrun error status bit
URXDA - Receive buffer data available bit
(URXDA=0 receive buffer is is empty, URXDA=1 receive buffer has data,
at least one more character can be read)
```

UART - Baud Rate Generator

- Dedicated 16-bit Baud Rate Generator
- Baud Rate controlled by UxBRG register
- (x = 1 or 2)
- **Baud Rate = Fcy / (16 * (UxBRG + 1))**
- where Fcy = Instruction Cycle Frequency
- Both transmitting and receiving devices
- must use same Baud Rate
- Bits are transmitted and/or received at
- the rate defined by the Baud Rate

Ví dụ

- If FCY=8MHz, and the desired baud rate is 9600,
  - UxBRG=(8•106/(16•9600))-1=51.083.
- UxBRG=51.
- The new baud rate for UxBRG=51 is 9615.384, i.e. the deviation is 0.16%

Ví dụ (tiếp)

BAUD RATE [KBPS]	FCY=30MHZ			FCY=25MHZ			FCY=20MHZ			FCY=16MHZ		
	KBAUD	Error [%]	BRG	KBAUD	Error [%]	BRG	KBAUD	Error [%]	BRG	KBAUD	Error [%]	BRG
0.3	0.3	0.0	6249	0.3	+0.01	5207	0.3	0.0	4166	0.3	+0.01	3332
1.2	1.1996	0.0	1562	1.2001	+0.01	1301	1.1996	0.0	1041	1.2005	+0.04	832
2.4	2.4008	0.0	780	2.4002	+0.01	650	2.3992	0.0	520	2.3981	-0.08	416
9.6	9.6154	+0.2	194	9.5859	-0.15	162	9.6154	+0.2	129	9.6154	+0.16	103
19.2	19.1327	-0.4	97	19.2901	+0.47	80	19.2308	+0.2	64	19.2308	+0.16	51
38.4	38.2653	-0.4	48	38.1098	-0.76	40	37.8788	-1.4	32	38.4615	+0.16	25
56	56.8182	+1.5	32	55.8036	-0.35	27	56.8182	+1.5	21	55.5556	-0.79	17
115	117.1875	+1.9	15	111.607	-2.95	13	113.6364	-1.2	10	111.1111	-3.38	8
250							250	0.0	4	250	0.0	3
500										500	0.0	1
MIN	0.0286	0.0	65535	0.0238	0.0	65535	0.019	0.0	65535	0.015	0.0	65535
MAX	1875	0.0	0	1562.5	0.0	0	1250	0.0	0	1000	0.0	0

UART - Transmission

- The first bit transmitted is a START bit
- Low level on UxTX pin for 1 bit time
- START bit followed by data and parity
- bits
- Data format (8 or 9 bits) and parity type
- (even, odd or no parity) are configured
- by PDSEL control bits in UxMODE
- register
- No parity for 9-bit data

UART - Transmission

- Last bit transmitted is a STOP bit
- □ High level on UxTX pin for 1 or 2 bit
- times
- □ Number of STOP bits are configured by
- STSEL control bit in UxMODE register
- □ During a transmission, TRMT status bit in the UxSTA register is clear

UART - Transmit Buffers

- 4-deep Transmit FIFO Buffer
- The characters in the buffer are shifted
- out of the buffer through UxTSR
- All 8 (or 9) data bits, are buffered
- Only the first character in the buffer is
- memory-mapped and thus useraccessible
- The UTXBF status bit in the UxSTA
- register indicates if the buffer is full

UART - Transmit Interrupts

- Transmit Interrupt indicated by UxTXIF bit and enabled by UxTXIE bit
- When UTXISEL bit in the UxSTA register = 1
- Interrupt occurs when buffer becomes empty
- Used for transmitting a block of 4 characters
- When UTXISEL bit = 0
- Interrupt occurs whenever a character is
- transferred to UxTSR
- Used for transmitting a single character

In this mode, an interrupt is generated as soon as the UTXEN bit is set

UART - Reception

- 4-deep Receive FIFO Buffer
- All 8 (or 9) data bits, as well as Error
- flags, are buffered
- Only the first character in the buffer is
- memory-mapped (UxRXREG)
- The error flags in the UxSTA register
- reflect the error states of the first
- character in the buffer
- URXDA status bit in the UxSTA register
- indicates if the buffer contains new data

UART - Receive Interrupts

- Receive Interrupt indicated by UxRXIF bit and enabled by UxRXIE bit
- When URXISEL bits in the UxSTA register = 11
  - Interrupt occurs when buffer becomes full
  - Used for receiving a block of 4 characters
- When URXISEL bits = 10
  - Interrupt when buffer has 3 characters
  - Used for receiving a block of 3 characters
- When URXISEL bits = 01 or 00
  - Interrupt whenever a character is received
  - Used for receiving a single character

UART - Error Detection

- Parity Error
  - When received parity does not match the parity calculated by module from received data
  - Indicated by PERR bit in the UxSTA register set
- Framing Error
  - When a STOP bit is expected on UxRX pin but a low logic level is detected
  - Indicated by FERR bit in the UxSTA register set
- Receive Overrun Error
  - When the Receive Buffer is full and a 5th character is received
  - Indicated by OERR bit in the UxSTA register set

UART - Address Detection

- When the UART is operating in 9-bit mode

(PDSEL = 11), and the ADDEN bit in the UxSTA register is set

  - The module will wait for an Address word, i.e., a 9-bit word with the 9th bit set
    - At this stage, the URXISEL bits in the UxSTA register must be set to 00 or 01
  - On receiving the Address word, the user inspects the lower byte to verify an address match
  - If an address match occurred, the user should clear the ADDEN bit, after which the module will wait for Data words (9-bit words with MSB clear)

UART - Features for LIN Support

- Transmission of Break Characters
- A Break character can be transmitted by setting the UTXBRK bit in the UxSTA register for at least 13 bit times
- Autobaud Detection
  - The UxRX pin is internally routed to an Input Capture pin (U1RX to IC1, U2RX to IC2)
  - Used for capturing both edges of START bit for determining baud rate
  - Enabled by setting the ABAUD bit in the UxMODE register

UART - Additional Features

- Alternate I/O
  - Some devices have an alternate pair of TX/RX pins
  - Enabled by setting ALTIO bit in UxMODE register
- Loopback Mode
  - UxTX pin internally connected to UxRX pin
  - Enabled by setting LPBACK bit in UxMODE Register
- Wake-up from SLEEP
  - Device can be woken up from SLEEP by START bit
  - Enabled by setting WAKE bit in UxMODE register

Ví dụ

```
unsigned rx1;

void main() {

  Uart1_Init(9600);                  // initialize USART module


  //--- un-comment the following lines to have Rx and Tx pins on their alternate
  //    locations. This is used to free the pins for other module, namely the SPI.
    U1MODEbits.ALTIO = 1;
    Delay_ms(10);                      // pause for usart lines stabilization
    rx1 = Uart1_Read_Char();           // perform dummy read to clear the register

  Uart1_Write_Char('s');               // signal start

  while(1) {
    if (Uart1_Data_Ready())  {       // check if there is data in the buffer
     rx1 = Uart1_Read_Char();        // receive data
     Uart1_Write_Char(rx1);          // send data back
    }
  }
}//~!
```