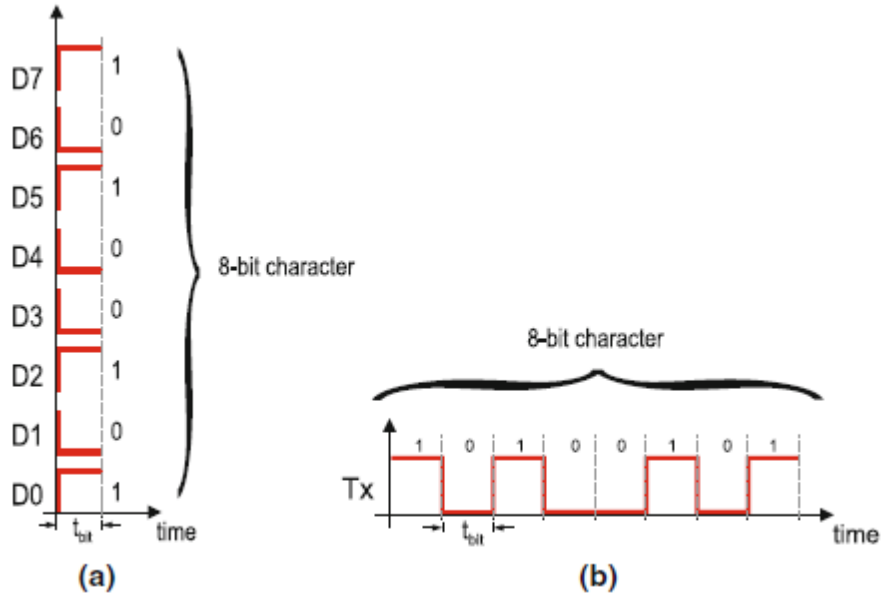
UNIT IV

SERIAL COMMUNICATIONS

Serial channels are the main form of communications used in digital systems nowadays. Diverse forms of serial communication formats and protocols can be found in applications ranging from short inter- and intra-chip interconnections, to the long range communication.

The Universal Serial Bus (USB), Blue-tooth, Local Area Networks, Wi-Fi, IrDA, the traditional RS-232, FireWire, I2C, SPI, PCI and many other protocols and formats are all based on serial links.

Each bit serially transmitted takes a pre-determined amount of time t_{bit}, requiring nt_{bit} seconds to transmit the entire character.



Parallel and Serial communication channels

Types of serial Channels

Serial channels are said to be ***simplex, half duplex, or full-duplex***, depending on their type of connectivity.

A simplex serial channel transmits permanently in only one direction over a dedicated link.

Examples: radio and TV broadcasts channels, printers and displays, or read-only devices.

A half-duplex serial channel features a single link that allows communication in either direction, but only in one direction at a time. It is a serial transceiver.

Eg: Walkie – talkie

Full Duplex: This type of data transmission allows data transfer in both directions simultaneously.

Eg: Telephone line

Transmission formats: The data in the serial communication can be sent in two formats i.e. **Asynchronous** and **Synchronous**.

Asynchronous transmission: In this the bits of a character or word can be sent at a constant rate. When no charaters are being sent a line stays at logic HIGH called MARK (STOP) and logic LOW called SPACE (START) .

The beginning of a character is indicated by a START bit which is always LOW. This is used to synchronize the transmitter and receiver. After the start bit, the data bits are sent with LSB first , followed by one or more stop bits. The combination of start bit, character and stop bits is called FRAME.

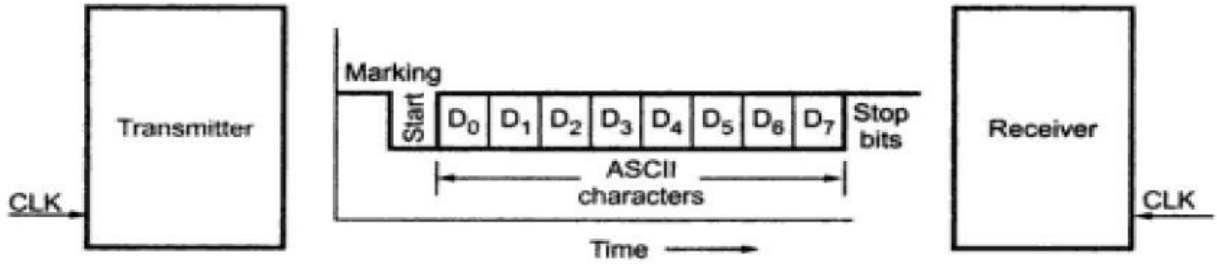


Figure 1: Transmission format for Asynchronous transmission

Synchronous transmission: The start and stop bits in each frame format represents overhead bytes that reduce the overall character rate. This overhead bits can be eliminated by synchronizing receiver and transmitter through common clock signal such a communication is called Synchronous serial communication.

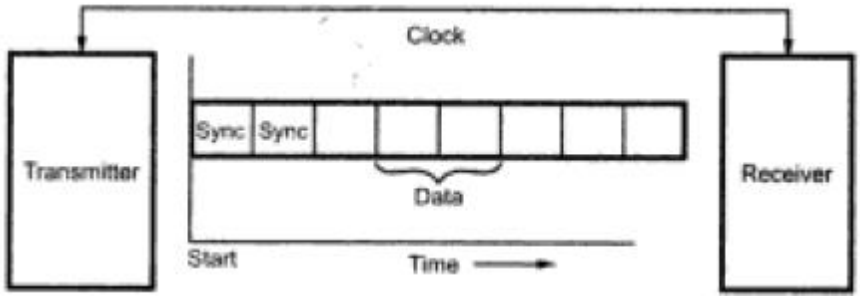


Figure 2: Synchronous serial transmission format

Synchronous bits are inserted instead of start and stop bits. Here a pair of SYNC bits are used at the start of data frame.

Comparison between Asynchronous and synchronous transmission formats

SNO	ASYNCHRONOUS SERIAL COMMUNICATION	SYNCHRONOUS SERIAL COMMUNICATION
1.	Transmitters and receivers are not synchronized by clock.	Transmitters and receivers are synchronized by clock.
2.	Bits of data are transmitted at constant rate.	Data bits are transmitted with synchronisation of clock.
3.	Characters may arrive at any rate at receiver.	Character is received at constant rate.
4.	Data transfer is character oriented.	Data transfer takes place in blocks.
5.	Start and stop bits are required to establish communication of each other.	Start and stop bits are not required to establish communication of each charcater. However, synchronisation bits are required to transfer the data block.
6.	Used in low-speed transmission at about speed less than 20kbps.	Used in high-speed transmissions.
7.	Example: UART, USB	Example: SPI, I2C

Consider transmitting the 8-bit character 0B7h with even parity enabled and one stop bit. The bit stream to be transmitted after adding the parity bit would be 111011010 (before attaching the start and stop bits). These leftmost eight bits correspond, from left to right and starting with the least significant bit, to the character to be transmitted; followed by the parity bit in the rightmost position.

UART Interface

In order to have a processor communicating over a serial channel, an interface module is needed. Serial interfaces, or adapters, fundamentally convert data from parallel to serial and vice versa, allowing the CPU to communicate through the serial channel. In asynchronous serial channels the most widely used interfaces are UARTs. UARTs are available as standalone peripherals or as embedded modules within MCUs.

A UART interface consists of the CPU interface, the clock interface, and the channel interface. The CPU interface becomes relevant when interfacing a UART chip to the CPU address, data, and control buses. The clock interface deals with the way the time base signals used to generate the transmission and reception baud rates are provided. The Channel-side interface connects to the actual serial channel, and becomes relevant in every UART application.

CPU – side interface AUART connects to the CPU buses through a set of bidirectional data lines (typically 8-bit) that transfer data into or out from the adapter, read/write control lines to specify the transfer direction, and selection lines, A*r*−1–A0 and CS, like any other I/O interface.

Clock interface feeds the baud rate generators inside the interface.

Channel Interface of a UART is TxD and RxD, which along with the signal ground (GND) carry the incoming and outgoing serial streams.

UART Configuration and Operation

Configuring the UART requires the following steps:

- Choosing the clock source for the baud rate generator(s): To establish the input clock frequency f_{clk} to the baud rate generator.
- Configuring the baud rate generator: Baud rate ***BR*** only takes dividing f_{clk} by a factor N determined as:

$$N = f_{clk}/BR$$

- Choosing the correct synchronization mode: To configure the corresponding bits in the control register to make the module operate as a UART.
- Choosing and configuring parity check (optional): Configure to use the same type of parity (even or odd), and configure the corresponding control bits.
- Configuring character and stop bit length: To configure different character lengths along with stop bit lengths.

Enabling: Enable whether operation will be handled via polling or via interrupts. The transmitter and/or receiver interrupt enable bits must be set.

Operation:

Polling:

When writing a new character for transmission in the data-out buffer, the TxReady flag must be polled to determine readiness. Otherwise data loss might occur.

When receiving by polling, the RxReady flag must be polled before retrieving incoming data from the receive buffer to avoid repeated retrieval of the same data.

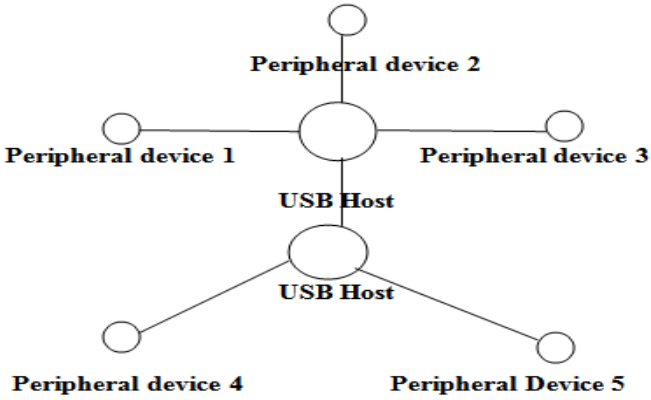
Interrupt:

In interrupt-based operation, a transmitter IRQ signals the readiness of the transmitter to accept new characters to be sent through the channel and the ISR can directly proceed to write into the transmission buffer.

The receiver interrupt means a new character was received and therefore the ISR can directly proceed with its retrieval, polling the error flags if they do not trigger interrupts by themselves.

Universal Serial Bus (USB)

- USB is a wired high speed serial bus for data communication. The USB communication system follows star topology with a USB host at centre and one or more peripherals connected to it.
- A USB host can support up to 127 slave peripheral devices and other USB hosts.
- USB transmits data in packet format. The USB communication is a host initiated one.
- The USB host controller is responsible for controlling the data communication, establishing connectivity, packetizing and formatting the data.
- The USB standard uses two different types of connector at the ends of the USB cable. **Type A connector** is used for **upstream connection** (connection with host, ex: PC, laptop) and **Type B connector** is used for **downstream connection** (connection with slave device).



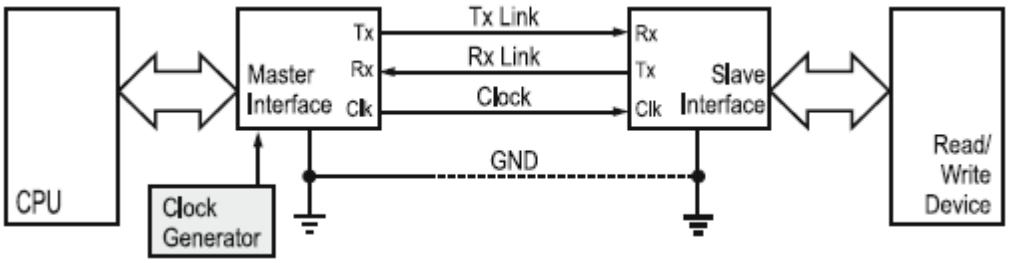
USB Device Connection

- USB interface has the ability to carry power to the connecting devices (GND and V_{BUS} pins).
- Each USB device contains a product ID (PID) which is embedded in to the USB chip by the manufacturer and vendor ID (VID) is supplied by the USB standards forum. These are essential for loading drivers to a USB device for communication.
- USB supports four different types of data transfers i.e. Control, Bulk, Isochronous and Interrupt.
    - **Control transfer** is used by USB system software to query, configure and issue commands to the USB device.
    - **Bulk transfer** is used for sending a block of data to the device. It supports error checking and correction. (Ex: transfer data to printer)
    - **Isochronous transfer, data** is transmitted in streams in real-time. It does not support error checking and re-transmission of data. (Ex: Audio devices)
    - **Interrupt transfer** is used for transferring small amounts of data. This mechanism uses polling technique to see whether USB device has any data to send. (Ex: Data from mouse or keyboard)

Synchronous Serial Communication

Synchronous serial channels are characterized by having both, transmitting and receiving devices synchronized with the same clock signal.

Synchronous serial channels usually operate in a master/slave mode where the master device initiates transfers and provides the clock signal driving the timing and synchronization in the channel. A slave device is controlled by the master to either receive or send information.



Synchronous Serial Communication

Serial Peripheral Interface (SPI) bus

The SPI bus is a synchronous bi-directional full duplex four-wire serial interface bus. SPI is a single master multi-slave system.

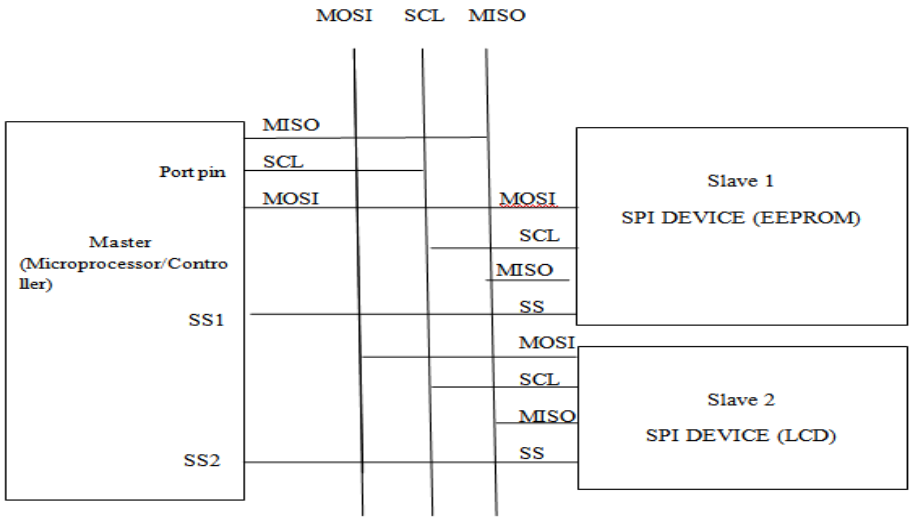
Signal lines for SPI bus

Master Out Slave In (MOSI) Signal line carrying the data from master to slave device.

Master In Slave Out (MISO) Signal line carrying the data from slave to master device.

Serial Clock (SCLK) Signal line carrying the clock signals.

Slave Select (SS) Signal line for slave device select.



SPI BUS

- The master device is responsible for generating the clock signals. It selects the required slave device by asserting corresponding slave device's slave select signal as "LOW" and data out line of all the slave devices in high impedance state.

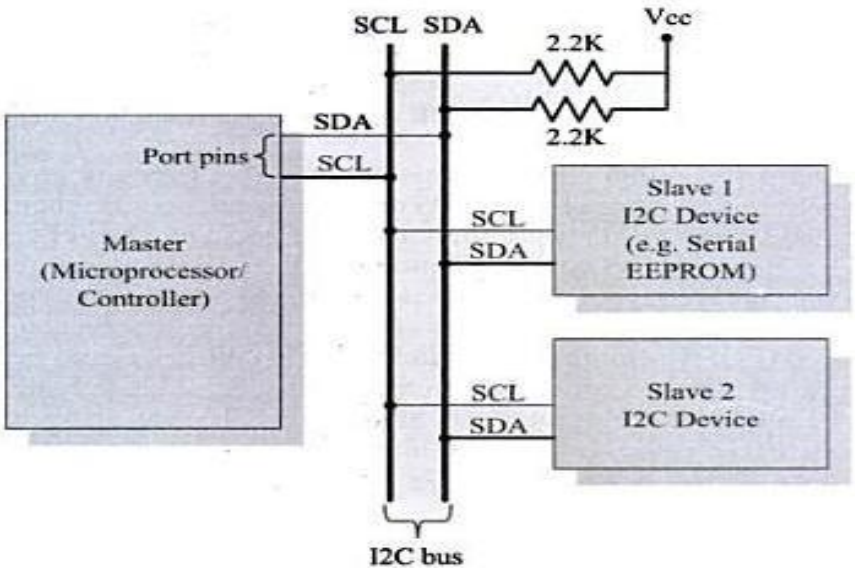
- SPI devices contain a certain set of registers. The serial peripheral control register holds various configuration parameters like master/slave selection for the device, baud rate selection for communication, clock signal control etc.
- The status register holds the status of various conditions for transmission and reception.
- SPI works on the principle of 'SHIFT REGISTER' for the data to transmit or receive.
- During transmission from master to slave, the data in master's shift register is shifted out to MOSI pin and it enters the shift register of slave device through MOSI pin of slave device.
- During transmission for slave to master, the shifted out data enters the shift register of master device through MISO pin.
- SPI bus is suitable for applications requiring transfer of data in *'streams'*.
- The limitation of SPI bus is that it does not support *Acknowledgement mechanism*.

Inter Integrated Circuit Bus (I^2C BUS)

- The *I^2C BUS* was designed to provide an easy way of connection between a microprocessor or microcontroller system and the peripheral chips.
- It is a synchronous bi-directional two-wire serial interface bus.

Features

- Two bus lines are required i.e. serial data line (SDA) and serial clock line (SCL).
- Each device is provided with unique address and simple master and slave relationships exist at all times.
- It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
- Serial 8-bit oriented data transfer can be made at *100 kbps in Standard mode, 400kbps in Fast mode and 3.4Mbps in High speed mode.*
- Number of IC's that can be connected to the same bus is limited by the maximum bus capacitance.



I^2C BUS

I^2C BUS Signals

SDA – SDA is responsible for transmitting the serial data across the devices.

SCL – SCL is responsible for generating synchronous clock pulses.

START – High-to-Low transition of the SDA line while SCL line is high.

ACK – Receiver pulls SDA line LOW transmitter allows it to float.

DATA – Transition take place while SCL is LOW.

STOP – Low-to-High transition of the SDA line while SCL line is high.

- Devices connected to the *I^2C BUS* can act as master or slave device.
- The master device is responsible for controlling the communication by initiating and terminating data transfer, sending data and generating necessary synchronization clock pulses.
- The address of a *I^2C* device is provided while hardwiring at the time of designing the embedded hardware.
- The slave device waits for the commands from the master and responds up on receiving the commands.

Sequence of Operation

- After initiating data transfer by START condition, the master sends the address of the slave device to which it wants to communicate over a SDA line. The master device sends Read/Write bit (RD=1, WR=0) along with slave address.
- Slave devices connected to the bus compares the address received to the address assigned to them. If any of the device's address matches with address sent by master on SDA line responds to master by sending ACK bit.
- Up on receiving ACK bit master device sends 8-bit data to the slave device over SDA line if the requested operation is write to device. If the operation is read, the slave device sends data to the master over SDA line.
- After completion of data transfer and successful reception of data slave device sends ACK bit to the master.
- The master device terminates the master by pulling SDA and SCL line to High to indicate STOP condition.

Advantages of I^2C

- Good for communication with on-board devices that are accessed occasionally.
- Easy to link multiple devices because of addressing scheme.
- Cost and complexity do not scale up with the number of devices.

Disadvantage of I^2C

- The complexity of supporting software components can be higher than that of competing schemes (for example, SPI).

Applications of I^2C

- Used as a control interface to signal processing devices those have separate data interfaces, e.g. RF tuners, video decoders and encoders, and audio processors.

Communications in MSP430

The communication modules available for the MSP430 family of microcontrollers are as follows:

- USART (Universal Synchronous/Asynchronous Receiver/Transmitter),
- USCI (Universal Serial Communication Interface) and
- USI (Universal Serial Interface)

Comparison

USART	USCI	USI
UART: - Only one modulator - n/a - n/a - n/a	**UART:** - Two modulators support n/16 timings - Auto baud rate detection - IrDA encoder & decoder - Simultaneous USCI_A and USCI_B (2 channels)	
SPI: - Only one SPI available - Master and Slave Modes - 3 and 4 Wire Modes	**SPI:** - Two SPI (one on each USCI_A and USCI_B) - Master and Slave Modes - 3 and 4 Wire Modes	**SPI:** - Only one SPI available - Master and Slave Modes
I^2C: *(on '15x/'16x only)* - Master and Slave Modes - up to 400kbps	**I^2C:** - Simplified interrupt usage - Master and Slave Modes - up to 400kbps	**I^2C:** - SW state machine needed - Master and Slave Modes

USART module

The USART (Universal Synchronous/Asynchronous Receiver/Transmitter) module is a base unit for serial communications, supporting both asynchronous communications (RS232) and synchronous communications (SPI).

The USART module is available in the 4xx series devices, particularly in the sub-series MSP430x42x and MSP430x43x.

USI module

The USI (Universal Serial Interface) module offers basic support for synchronous serial communications SPI and I2C. It is available in the MSP430x20xx devices family.

USCI module

USCI (Universal Serial Communication Interface) module is a communications interface designed to interconnect to industrial protocols:

- LIN (Local interconnect Network), used in cars (door modules, alarm, sunroof, etc.);
- IrDA (Infrared Data Association), used for remote controllers.

The USCI module is available in the following devices:

- MSP430F5xx;
- MSP430F4xx and MSP430FG461x;
- MSP430F2xx.

The USCI module supports:

- Low power operating modes (with auto-start);
- Two individual blocks:
- USCI_A:

o UART with Lin/IrDA support;

o SPI (Master/Slave, 3 and 4 wire modes).

- USCI_B:

o SPI (Master/Slave, 3 and 4 wire mode);

o I2C (Master/Slave, up to 400 kHz).

- Double buffered TX/RX
- Baud rate/Bit clock generator with:
- Auto-baud rate detect;
- Flexible clock source.
- RX glitch suppression;
- DMA enabled;
- Error detection.

Initialization sequence:

The recommended USCI initialization/re-configuration process is:

- Set UCSWRST (BIS.B #UCSWRST, &UCxCTL1);
- Initialize all USCI registers with UCSWRST = 1 (including

UCxCTL1);

- Configure ports;
- Clear UCSWRST via software (BIC.B #UCSWRST, &UCxCTL1);
- Enable interrupts (optional) via UCxRXIE and/or UCxTXIE.

Baud Rate:

For a specific clock source frequency, the divider value is given by:

$$N = BR_{CLK}/\text{Baud rate}$$

Typically, the value of *N* is not an integer value, so it is necessary to use a modulator.

Low-Frequency Baud Rate Generation

The "Low-Frequency Baud Rate Generation" mode is selected when UCOS16 = 0. The baud rate generation mode is useful for lowering power consumption, since it uses a low frequency clock source (32.768 kHz crystal).

$$UCBRx = \text{int}(N) * prescaler$$

Oversampling Baud Rate Generation

The "Oversampling Baud Rate Generation" mode is selected when UCOS16 = 1. This mode allows precise bit timing. It requires clock sources 16x higher than the desired baud rate.

$$UCBRx = \text{int}(N/16) * prescaler$$

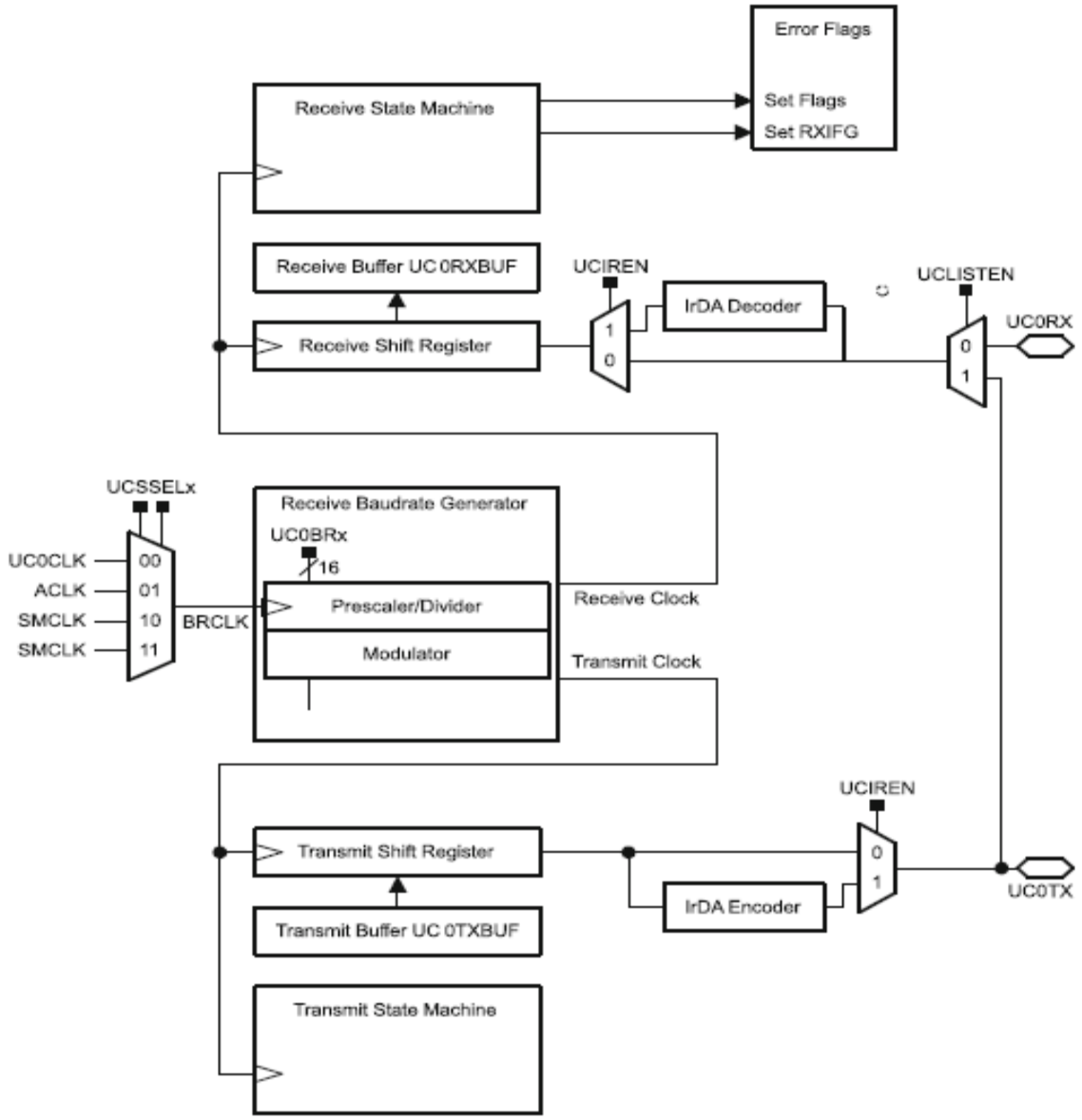
Serial communication modes operation

USCI operation: UART mode

The UART mode of the USCI, supported by USCI_A modules, provides all the capabilities of USART in UART mode, with added features to encode and decode IrDA bit streams in the UC0RX and UC0TX lines. Moreover, the USCI UART has auto baud detection capabilities, a feature that allows using it for LIN communications.

Configuration: The device needs to be placed in reset mode by setting the UCSWRST flag. The USCI is placed in its asynchronous mode by clearing UCSYNC bit in the UCAxCTL0 control register.

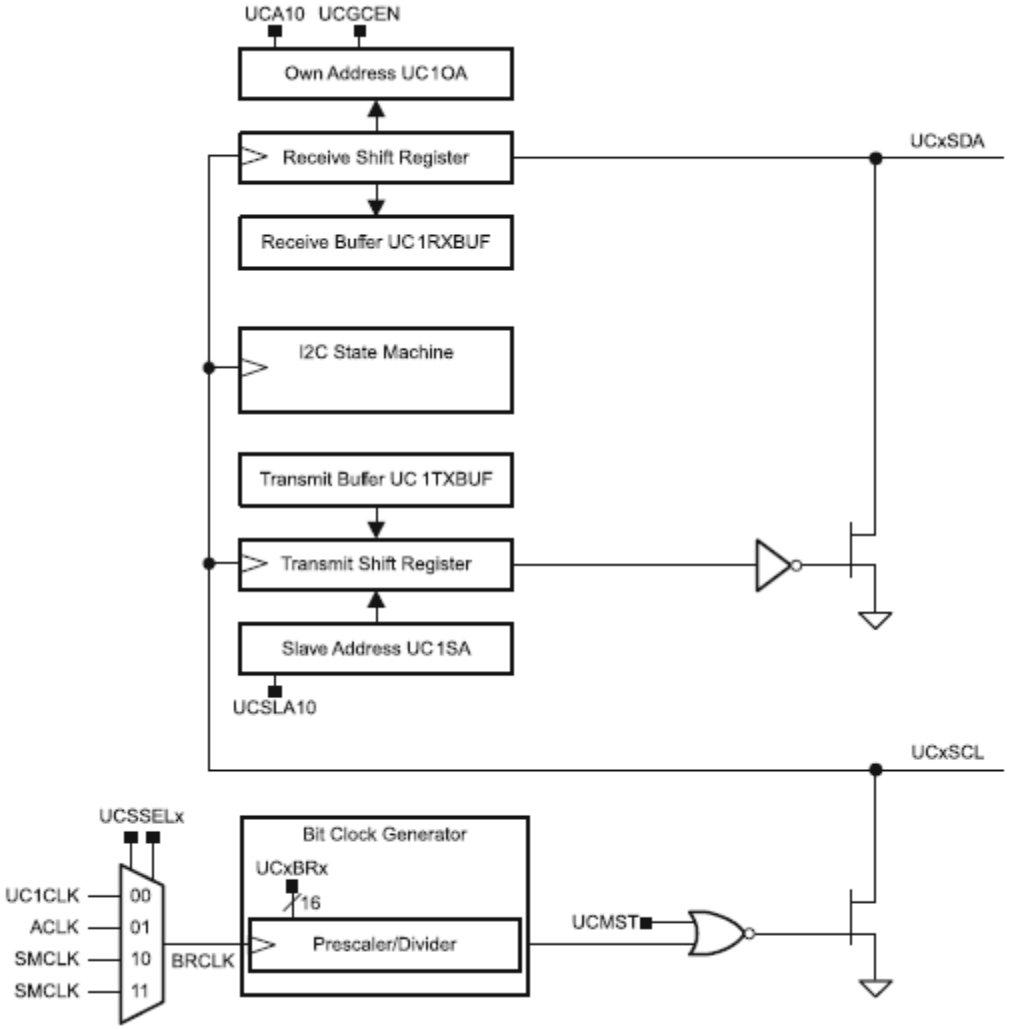
The overall sequence to initialize the USCI can be outlined as follows:
Step 1: Set UCSWRST to place the USCI in reset mode. UCSWRST is by default set by a PUC.
Step 2: Initialize all USCI registers, including UCAxCTL1, while holding UCSWRST = 1.
Step 3: Configure the USCI ports.
Step 4: Clear UCSWRST via software to enable the USCI.
Step 5: Enable interrupts via UCAxRXIE and/or UCAxTXIE.
In its asynchronous mode theUSCI can support point-to-point and multiprocessor transfers.
For common point-to-point connections, the USCIUART is configured in the idle line mode, with no multiprocessor capability enabled. This is achieved by setting bits UCMODEx = 00 in control register UCAxCTL0.



Block diagram of USCI_A configured in UART mode

MSP430 USCI in I2C Mode

Communication uses the bi-directional serial data (SDA) and serial clock (SCL) pins. A master initiates data transfers and generates the clock signal SCL. Any device addressed by a master is taken to be a slave.



Block Diagram for Operation of USCI in I^2C

- **I2C serial data:**
  - One clock pulse is generated by the master device for each data bit transferred;
  - Operates with byte data (MSB transferred first).
  - The first byte after a START condition consists of a 7-bit slave address and a R/W bit:

    R/W = 0: Master transmits data to a slave.

    R/W = 1: Master receives data from a slave.
  - The acknowledge (ACK) bit is sent from the receiver after each byte on the 9th SCL clock.

- I2C addressing modes (7-bit and 10-bit addressing modes).
- I2C module operating modes:
  - Master transmitter.
  - Master receiver.
  - Slave transmitter.
  - Slave receiver.

- **I2C transmit interrupt operation:**
  - UCBxTXIFG interrupt flag is set by the transmitter to indicate that UCBxTXBUF is ready to accept another character.
  - An interrupt request is also generated if UCBxTXIE and GIE are set.
  - UCBxTXIFG is automatically reset if a character is written to UCBxTXBUF or a NACK is received.

- **I2C receive interrupt operation:**
  - UCBxRXIFG interrupt flag is set each time a character is received and loaded into UCxRXBUF.
  - An interrupt request is also generated if UCBxRXIE and GIE are set.
  - UCBxRXIFG and UCBxRXIE are reset by a system reset PUC signal or when SWRST = 1.
  - UCxRXIFG is automatically reset when UCBxRXBUF is read.

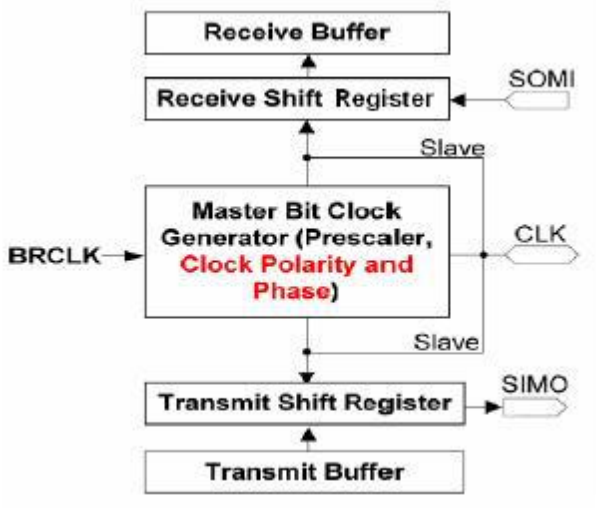
- Not-acknowledge interrupt, UCNACKIFG: Flag set when an acknowledge is but is not received.

- Start condition detected interrupt, UCSTTIFG: Flag set when the I2C module detects a START condition, together with its own address while in slave mode.

- Stop condition detected interrupt, UCSTPIFG: Flag set when the I2C module detects a STOP condition while in slave mode.

MSP430 USCI in SPI Mode

- Three or four signals interface are used for SPI data exchange:
  - UCxSIMO: Slave in, master out
  - UCxSOMI: Slave out, master in
  - UCxCLK: USCI SPI clock
  - UCxSTE: Slave transmit enable
- Data length is of 7 or 8 bits
- The serial transmit bit is start from LSB or MSB by UCMSB.



USCI operation: SPI mode

- LPMx operation.
- DMA enabled.
- Define mode: Master or Slave;
- Enable SPI transmit/receive by clearing the UCSWRST bit:
- Define serial clock control:
  - UCxCLK is provided by the master on the SPI bus.
  - Configure serial clock polarity and phase (UCCKPL and UCCKPH bits).
- SPI transmit interrupt operation:
  - UCxTXIFG interrupt flag is set by the transmitter to indicate that UCxTXBUF is ready to accept another character;
  - An interrupt request is also generated if UCxTXIE and GIE are set.
  - UCxTXIFG is automatically reset if the interrupt request is serviced or if a character is written to UCxTXBUF.
- SPI receive interrupt operation:
  - UCxRXIFG interrupt flag is set each time a character is received and loaded into UCxRXBUF.
  - An interrupt request is also generated if UCxRXIE and GIE are set.
  - UCxRXIFG and UCxRXIE are reset by a system reset PUC signal or when SWRST = 1;
  - UCxRXIFG is automatically reset if the pending interrupt is serviced (when UCSWRST = 1) or when UCxRXBUF is read.

Capture/Compare blocks

TIMER_A/TIMER_B contain independent capture and compare blocks, TACCRX/ TBCCRX) that may be used to capture the timer register contents, as they are at time of an event, or to generate an event when the timer register contents correspond to capture/compare register contents, e.g. to generate time intervals.

The mode is selected by the mode bit CAP in their individual Capture/Compare Control register, TACCTLx /TBCCTLx.

Capture mode

The capture mode is used to measure the period of timed events, with minimal CPU intervention. Capture mode configuration is achieved by carrying out the following steps:

- Set CAP bit to select the capture mode;
- Set SCS bit to synchronize the capture with the next timer clock to avoid race conditions.
- The input signal is sampled by the CCIxA (or CCIxB) input, selected by the CCISx bits in the Capture/Compare Control Register, TACCTLx (or TBCCTLx);
- The capture edge of the input signal (rising, falling, or both) is selected by the CMx bits;
- When the appropriate edge is detected on the selected input line, the value in the Timer register is latched into the TACCRx (or TBCCRx) register, providing a time mark for the event.
- The interrupt flag CCIFG is set.
- The bit COV (=1) controls an overflow event when a second capture is performed before the value from the first capture is read.

Compare mode

The compare mode is used for pulse generation or interrupts at specific time intervals. One of its common applications is to generate Pulse Width Modulation (PWM) output signals.

Compare mode operation is configured as follows:

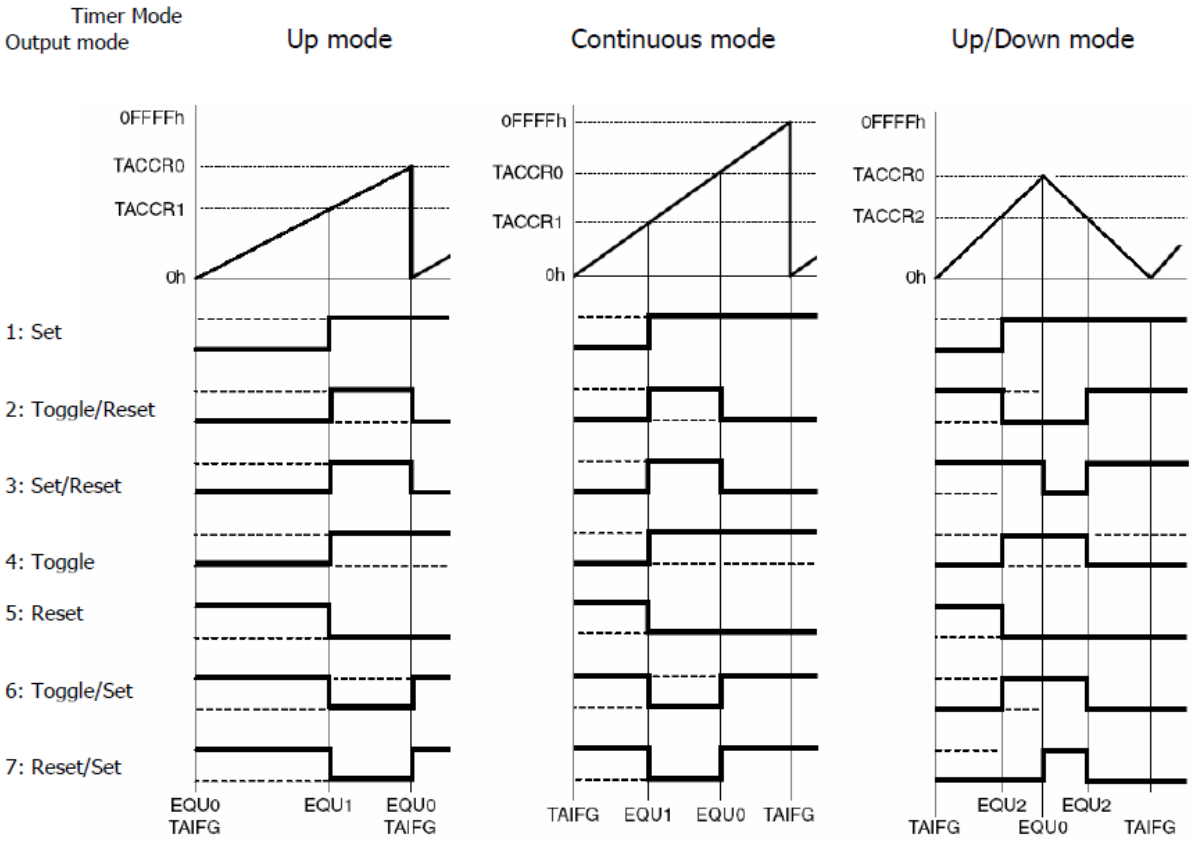
- Reset CAP bit to select compare mode.
- TxR counts to the value programmed in the TxCCRx register.
- When the timer value is equal to the value in the TxCCRx register, an interrupt is generated:

  Interrupt flag CCIFG is set.

  Internal signal EQUx = 1 (x is the number of the CCR channel).

Output operating modes:

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the bit OUTx
001	Set	OUTx = 1 ⇒ timer = TACCRx OUTx = 0 ⇒ timer = 0 or until another output mode is selected and affects the output
010	Toggle/Reset	OUTx = toggle ⇒ timer = TACCRx OUTx = 0 ⇒ timer = TACCR0
011	Set/Reset	OUTx = 1 ⇒ timer = TACCRx OUTx = 0 ⇒ timer = TACCR0
100	Toggle	OUTx = toggle ⇒ timer = TACCRx The output period is double the timer period
101	Reset	OUTx = 0 ⇒ timer = TACCRx OUTx = 1 ⇒ another output mode is selected and affects the output
110	Toggle/Set	OUTx = toggle ⇒ timer = TACCRx OUTx = 1 ⇒ timer = TACCR0
111	Reset/Set	OUTx = 0 ⇒ timer = TACCRx OUTx = 1 ⇒ timer = TACCR0



Output examples for different timer modes

Capture/Compare blocks registers

TACCTLx, Timer_A Capture/Compare Control Register

15	14	13	12	11	10	9	8
CM1	CM0	CCIS1	CCIS0	SCS	SCCI	Unused	CAP

7	6	5	4	3	2	1	0
OUTMOD2	OUTMOD1	OUTMOD0	CCIE	CCI	OUT	COV	CCIFG

Bit		Description
15-14	CMx	Capture mode: CM1 CM0 = 00 ⇒ No capture CM1 CM0 = 01 ⇒ Capture on rising edge CM1 CM0 = 10 ⇒ Capture on falling edge CM1 CM0 = 11 ⇒ Capture on both edges
13-12	CCISx	Capture/compare input select: CCIS1 CCIS0 = 00 ⇒ CCIxA CCIS1 CCIS0 = 01 ⇒ CCIxB CCIS1 CCIS0 = 10 ⇒ GND CCIS1 CCIS0 = 11 ⇒ V_{cc}
11	SCS	Synchronize capture input signal with timer clock: SCS = 0 ⇒ Asynchronous capture SCS = 1 ⇒ Synchronous capture
10	SCCI	Synchronized capture/compare input
8	CAP	Mode: Capture mode ⇒ CAP = 1 Compare mode ⇒ CAP = 0
7-5	OUTMODx	Output mode: OUTMOD2 OUTMOD1 OUTMOD0 = 000 ⇒ Bit OUT OUTMOD2 OUTMOD1 OUTMOD0 = 001 ⇒ Set OUTMOD2 OUTMOD1 OUTMOD0 = 010 ⇒ Toggle/Reset OUTMOD2 OUTMOD1 OUTMOD0 = 011 ⇒ Set / Reset OUTMOD2 OUTMOD1 OUTMOD0 = 100 ⇒ Toggle OUTMOD2 OUTMOD1 OUTMOD0 = 101 ⇒ Reset OUTMOD2 OUTMOD1 OUTMOD0 = 110 ⇒ Toggle / Set OUTMOD2 OUTMOD1 OUTMOD0 = 111 ⇒ Reset / Set
4	CCIE	Capture/compare interrupt enable when CCIE = 1.
3	CCI	Capture/compare input
2	OUT	Output state
1	COV	Capture overflow when COV = 1
0	CCIFG	Capture/compare interrupt flag CCIFG = 1 when interrupt pending