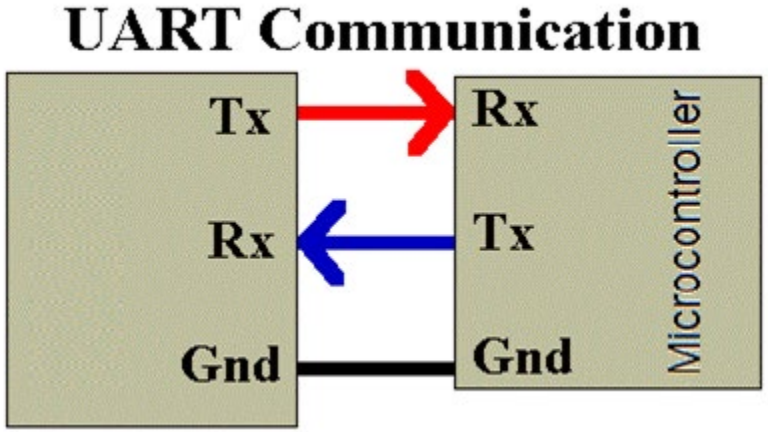
ESET 269

UART

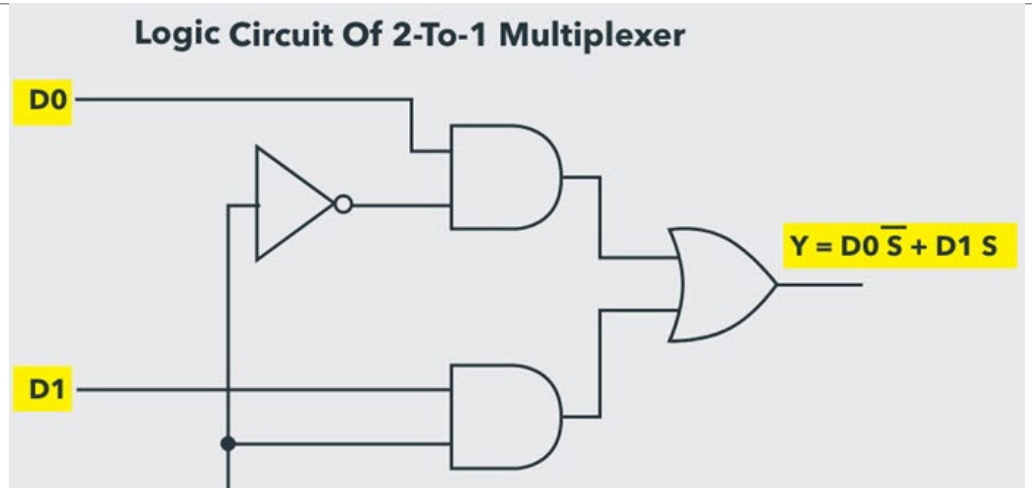
DR. GARTH V. CROSBY

UART

* Universal Asynchronous Receiver – Transmitter
* The MSP432 can have up to 4 UART ports: UCA0, UCA1, UCA2 an UCA3
* Circuitry in the microcontroller that translates parallel data to serial data and vice versa
  - Parallel bus data (inside microcontroller) is sent via serial (outside microcontroller)
* Requires only 3 connections. Tx, Rx, GND

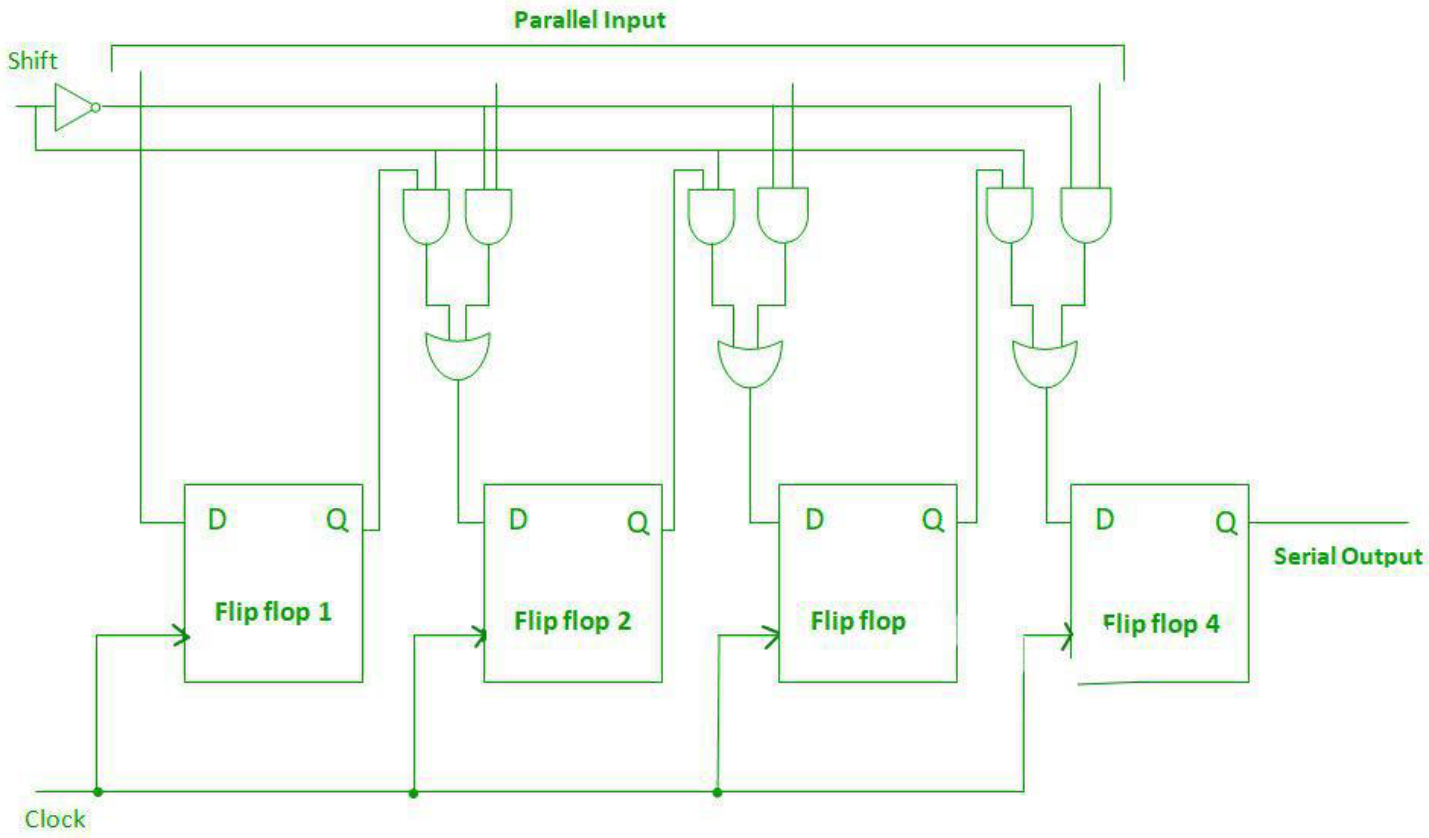


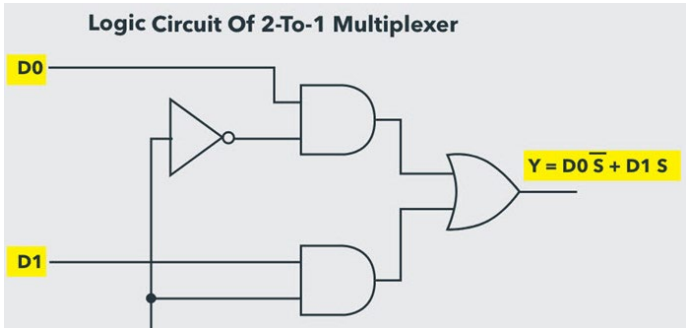
2x1 Multiplexer (Remember this?)



S	D0	D1	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

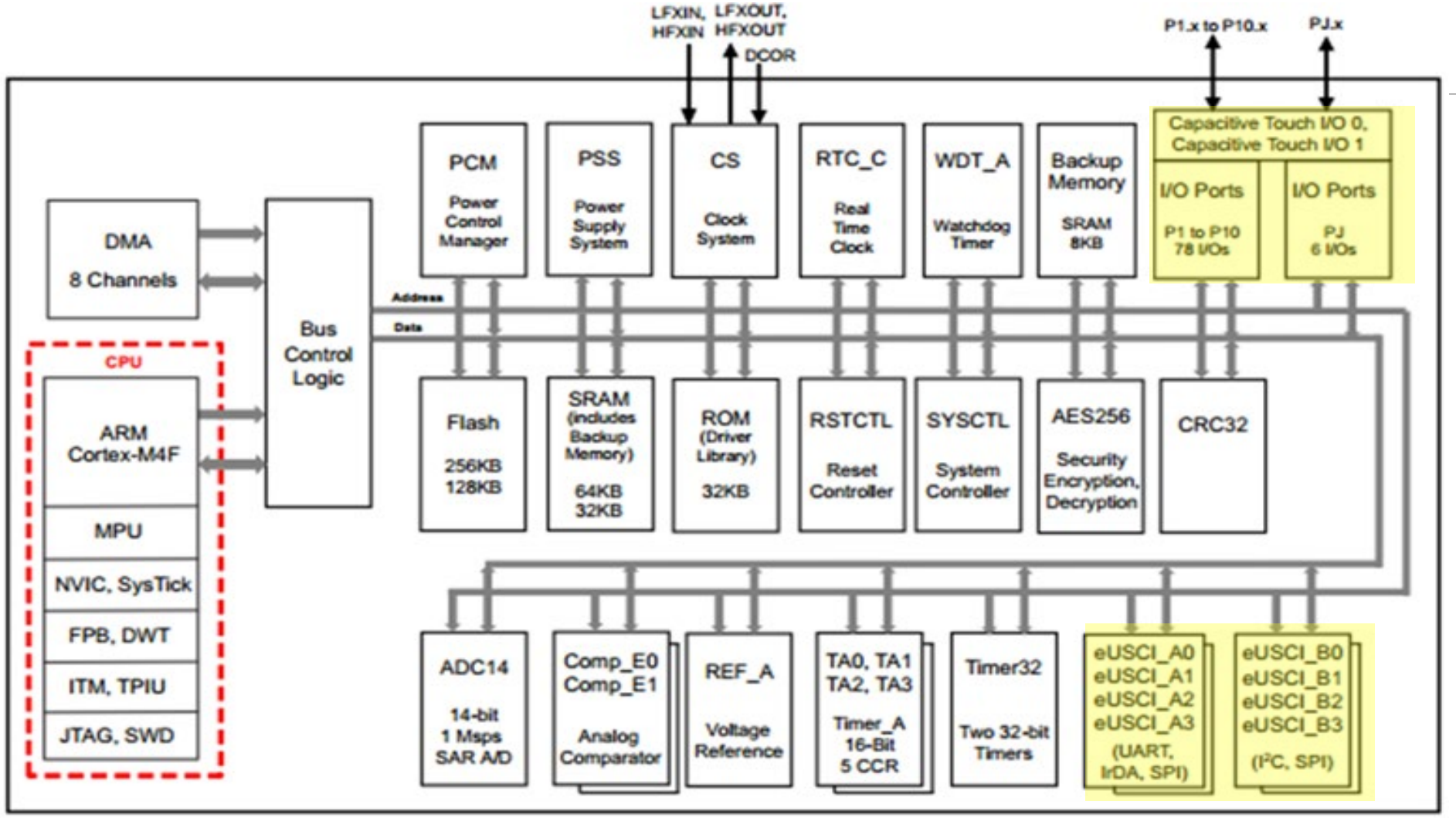
4-bit Parallel In Serial Out (PISO) Register





Select Input S

MSP432 Block Diagram



UART With MSP432

* MSP432 has 4 UART modules
  - One is tied to the circuitry going to PC on PORT 1

Pin Name	SEL = 00	SEL = 01	SEL = 10	SEL = 11
P1.0	Simple I/O	UCA0STE	-	-
P1.1	Simple I/O	UCA0CLK	-	-
P1.2	Simple I/O	UCA0RXD/UCA0SOMI	-	-
P1.3	Simple I/O	UCA0TXD/UCA0SIMO	-	-
P1.4	Simple I/O	UCB0STE	-	-
P1.5	Simple I/O	UCB0CLK	-	-
P1.6	Simple I/O	UCB0SIMO/UCB0SDA	-	-
P1.7	Simple I/O	UCB0SOMI/UCB0SCL	-	-

UART Registers

In all microcontrollers, there are 4 groups of registers in UART peripherals:

- **Configuration (Control) registers:** Before using the UART peripheral the configuration registers must be initialized. This sets some communication parameters such as:
  - Baud rate, word length, stop bit, start bit etc.
  - **UCAxCTLWO** and **UCAxBRW** are two of the configuration registers

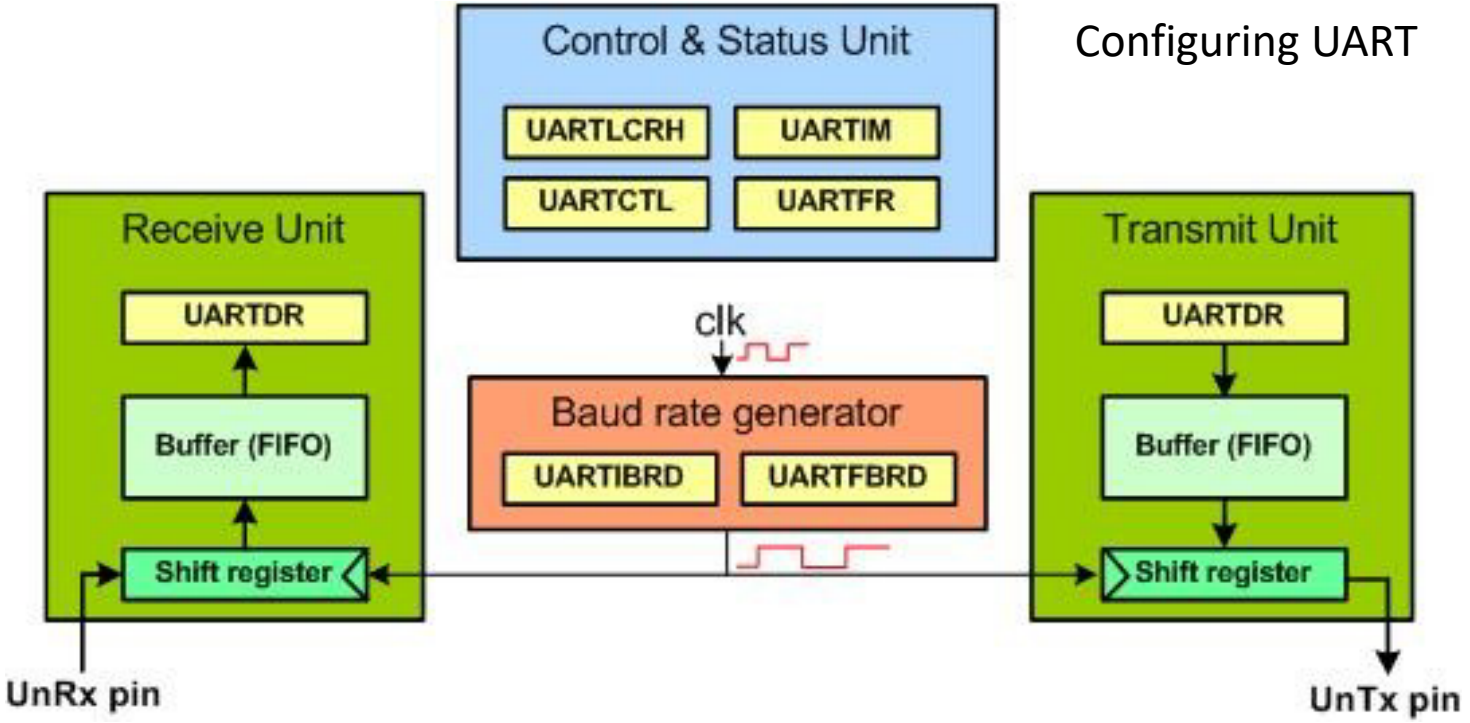
- **Transmit and Receive register:** To send data, write to the transmit register: **UCATXBUF**

  *The received data is stored in the receive register: **UCAxRXBUF**

- Status register (UCAxSTATW): contains some flags which sjows the error in sending and receiving data including: the framing error, the party, overrun errors and busy flag.

- **Flag register (UCAxIFG):** contains some flags which show the state of sending and receiving data including: the transmitter sent out entire byte, transmitter ready for another byte, the receiver received an entire byte of data.

UART Block Diagram



Configuring UART

UART-Baud Rate

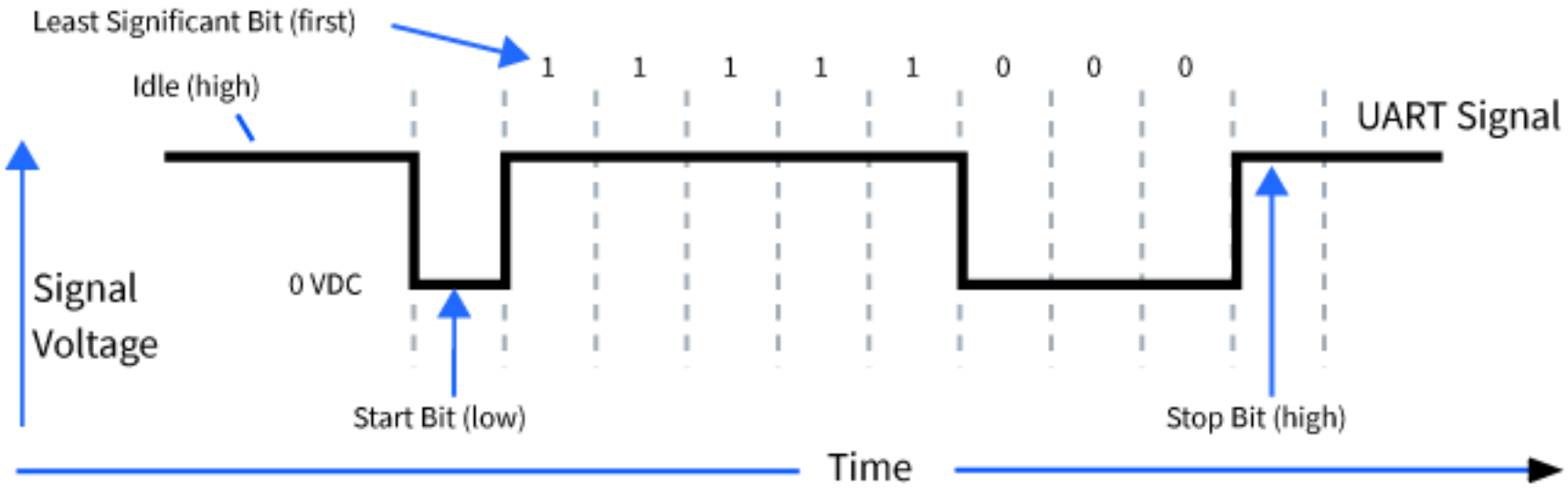
* UART does not use a shared clock between devices

* Both devices instead have a baud rate
  - How fast data is sent over serial line in bits per second (bps)
  - Inverse of baud rate is how long it takes for a single bit to be transmitted
  - Each device must operate at the same baud rate

* Most common baud rate is 9600. Other values include19200, 38400, 57600, and 115200

UART Frame

* UART signals when data is being transmitted with a start bit
* Data is typically 8-bits. 1 byte or ASCII character
* A stop bit is used to signal end of transmission



UART Frame

* Every byte of data requires 10 bits
  - 1 byte + 1 start bit + 1 stop bit
* If the baud is 9600 bits per second, or 960 bytes per second
* Why is it not 1,200 bytes per second?
  - The start and stop bit slow down transmission

Configuring UART

*The UART module must be set up before use.

* Enhanced Universal Serial Communication Interface(eUSCI)/modules supports serial communication in MSP432

* The MSP432 has two different types of serial interfaces/modules: **eUSCI_A** and **eUSCI_B**

  - eUSCI_Ax modules support UART and SPI protocols

  - eUSCI_Bx modules supports SPI and I2C protocols

*For UART we need to set multiple items in the UART Control Word 0 register

```
EUSCI_A0 ->CTLW0
```

* Baud rate is configured in the Baud Rate Word register

```
EUSCI_A0 ->BRW
```

EUSCI_A0 ->CTLW0

Typical UART setup for interface to PC

* No parity
* LSB first
* 8 – bit data
* one stop bit
* async. Mode
* Subsystem master clock
* D5 – D1 are 0

What is the CTLW0 register value for the above?

UCAxCTLW0:

D15	D14	D13	D12	D11	D10	D9	D8
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC

D7	D6	D5	D4	D3	D2	D1	D0	
UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST	0x0000

Figure 4-13: UART ControlWord0 (UCAxCTLW0) register

Field	Bit	Description
UCPEN	D15	0b = Parity disabled 1b = Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD).
UCPAR	D14	0b = Odd parity 1b = Even parity
UCMSB	D13	0b = LSB first 1b = MSB first
UC7BIT	D12	0b = 8-bit data 1b = 7-bit data
UCSPB	D11	0b = One stop bit 1b = Two stop bits
UCMODEx	D10:9	00b = UART mode 01b = Idle-line multiprocessor mode 10b = Address-bit multiprocessor mode 11b = UART mode with automatic baud-rate detection
UCSYNC	D8	0b = Asynchronous mode 1b = Synchronous mode
UCSSELx	D7:6	00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK
UCRXEIE	D5	0b = Erroneous characters rejected and UCRXIFG is not set. 1b = Erroneous characters received set UCRXIFG.
UCBRKIE	D4	0b = Received break characters do not set UCRXIFG. 1b = Received break characters set UCRXIFG.
UCDORM	D3	0b = Not dormant. All received characters set UCRXIFG. 1b = Dormant. Only characters that are preceded by an idle-line or with address bit set UCRXIFG. In UART mode with automatic baud-rate detection, only the combination of a break and synch field sets UCRXIFG.
UCTXADDR	D2	0b = Next frame transmitted is data. 1b = Next frame transmitted is an address.
UCTXBRK	D1	0b = Next frame transmitted is not a break. 1b = Next frame transmitted is a break or a break/synch.
UCSWRST	D0	0b = Disabled. eUSCI_A reset released for operation. 1b = Enabled. eUSCI_A logic held in reset state.

Table 4-3: UART Control 0 Word (UCAxCTLW0) register

Baud Rate

* Baud rate is set with the baud rate word register `EUSCI_A0 ->BRW`

$$Baud\ Rate = \frac{Clock}{BRW} \text{ (rounded down)}$$

* Above will lead to a small % error, but this can be tolerated

* The modulation control register must be set to 0 *before* setting baud rate
  - Not doing so will enable oversampling to correct % error; this can make the above equation invalid

```
EUSCI_A0->MCTLW = 0;
```

Steps to Setting Up UART

* Put UART in reset state by setting bit 0 of CTLW0 to 1
* Configure parity, LSB first, 8-bit data, etc.
* Set MCTWL to 0
* Set BRW value
* Set the port 1 pin 2 & 3 SEL1 = 0 and SEL0 = 1
* Take UART out of reset state

Example

```
EUSCI_A0 ->CTLW0 |=1; //put in reset state
EUSCI_A0 ->MCTLW = 0;
EUSCI_A0->CTLW0 |= 0x80;// 1 stop bit, no parity, SMCLK, 8-bit data
EUSCI_A0->BRW = 26; //baud rate
P1->SEL0 |= 0x0C;
P1->SEL1 &= ~0x0C;
EUSCI_A0 ->CTLW0 &=~0x01; //take out of reset state
```

Transmitting Data

* UART transmits one byte at a time
  - Typically send character data types

* Have to wait until current character is sent before sending a new character

1. Place character in transmit buffer

2. Wait until character transmitted

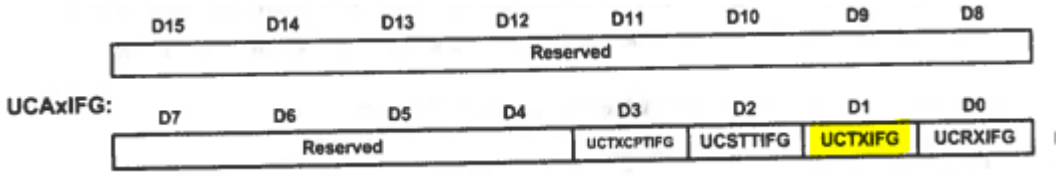
3. Place next character in transmit buffer

4. Repeat until all characters are sent

Transmit Buffer

* The buffer is 8-bits (size of a character data type) `EUSCI_A0->TXBUF`
* Once a value is placed in the buffer, it is automatically sent out the transmit pin
* The IFG register is used to monitor the transmit buffer
  - If the buffer is free, the TXIFG bit is set to 1, it is 0 if the transmit buffer has data

`EUSCI_A0->IFG`



Example

```
EUSCI_A0 ->TXBUF = 'Y';
while((EUSCI_A0 ->IFG & 2)==0)
{
  //wait
}
EUSCI_A0 ->TXBUF = 'e';
while((EUSCI_A0 ->IFG & 2)==0)
{
  //wait
}
EUSCI_A0 ->TXBUF = 's';
while((EUSCI_A0 ->IFG & 2)==0)
{
  //wait
}
```

* Can only place one character at a time in transmit buffer

String Transmit Example

```
char word[20] = "Hello World\n";
int i = 0;
while(word[i] != 0)
{
  EUSCI_A0 ->TXBUF = word[i];
  while((EUSCI_A0->IFG & 2)==0)
  {
    //wait
  }
  i++;
}
```

Sprintf

* Sprintf formats a string with conversion characters to place into a character array

```
Sprintf(char array, string, conversion character variables)
```

```
char word[20];
int x  = 4;
sprintf(word, "The %s is %d weeks\n\r","test",x);
printf("%s",word);
```

```
The test is 4 weeks
```

Example

```
char word[20];
float x = 4.556;
int y = -9009;
sprintf(word,"%.3f %d stuff\n\r",x,y);
int i = 0;
while(word[i] != 0)
{
  EUSCI_A0 ->TXBUF = word[i];
  while((EUSCI_A0->IFG & 2)==0)
  {
    //wait
  }
  i++;
}
```

```
4.556 -9009 stuff
```

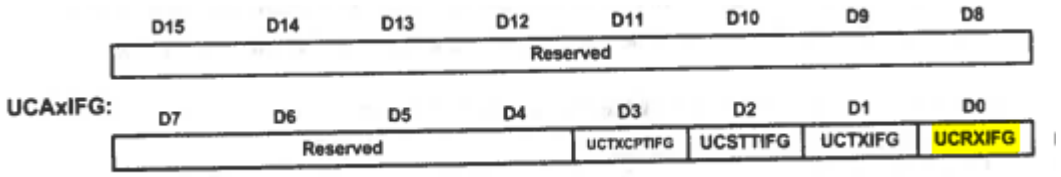
Receiving Data

* Any data sent is placed in a receive buffer
  - The characters are extracted one at time on a first in first out basis

* The RXIFG value in the IFG register is set to 1 when data is in the receive buffer, and 0 when nothing is present

* Program must poll for data periodically

```
char x;
if((EUSCI_A0 ->IFG & 1)!=0)
{
  x = EUSCI_A0->RXBUF;
}
```



Echoing

* The console window will not automatically display characters typed

* Echo back received input

```
while(1)
{
  char x;
  if((EUSCI_A0 ->IFG & 1)!=0)
  {
    x = EUSCI_A0->RXBUF;
    EUSCI_A0 ->TXBUF = x; //send what was typed
    while((EUSCI_A0->IFG & 2)==0)
    {
      //wait
    }
  }
}
```

Receiving String

* If expecting a user to type a string, each character received is stored as an element in an array
  - If character is not taken out of receive buffer, it is overwritten if a new one comes in

* Need a character to let program know end of string is reached
  - Enter key form keyboard is `\r`

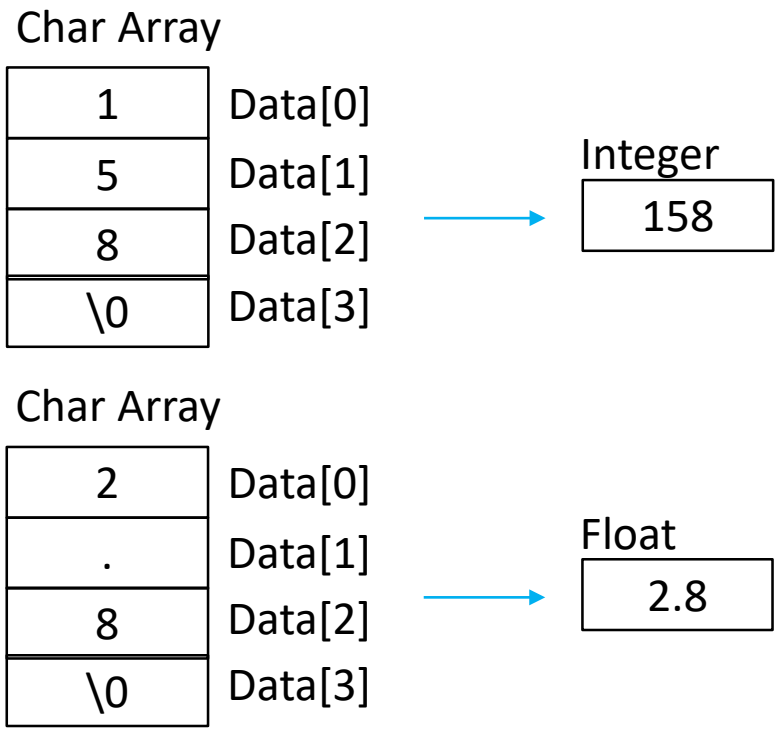
Example

```
char word[20];
while(1)
{
  if((EUSCI_A0 ->IFG & 1)!=0)
  {
    word[i]=EUSCI_A0->RXBUF; //place current character in array
      if(word[i] == '\r') //enter key pressed
      {
        word[i] ='\0'; //why change this?
        break; //break out of while loop
      }
      else
      {
        i++; //increment index
      }
  }
}
```

Handling Numbers

* Any number is treated as a string and have to be converted

* Easy method is to use the `atoi()` and `atof()` functions




```
char word[20];
while(1)
{
  if((EUSCI_A0 ->IFG & 1)!=0)
  {
    word[i]=EUSCI_A0->RXBUF; //place current character in array
      if(word[i] == '\r') //enter key pressed
      {
        word[i] ='\0'; //why change this?
        break; //break out of while loop
      }
      else
      {
        i++; //increment index
      }
  }
}
  ans = atoi(word); //convert string to integer
  ans = ans*2; //do math
  sprintf(word,"%d\r\n",ans); //format for printing
```

Functions For UART

* Read string – continuous while loop that checks the receive buffer until a termination character is received

* Write string – transmits a string using sprintf to format any conversion characters

* Helper functions to convert strings to numbers, convert data, etc.