UNIVERSITY OF ALASKA FAIRBANKS

EE 444/645: Embedded Systems Design

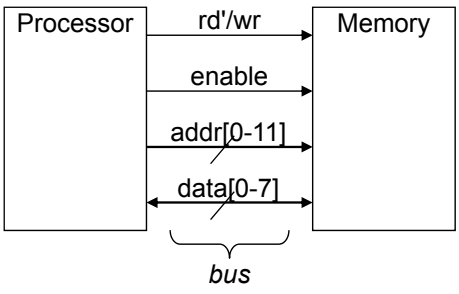
Microcontroller interfacing
MSP430 – USCI UART Mode

Spring 2017. Set: 5

Instructor: Dr. Dejan Raskovic

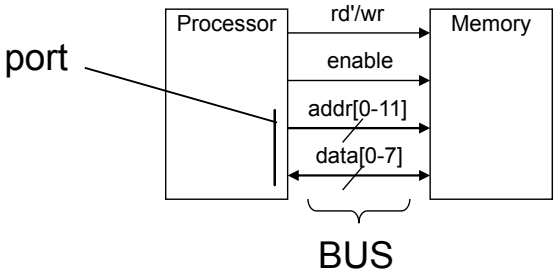
A simple bus

- Wires:
  - Uni-directional or bi-directional
  - One line may represent multiple wires
- Bus
  - Set of wires with a single function
    - Address bus, data bus
  - Or, entire collection of wires
    - Address, data and control
    - Associated protocol: rules for communication



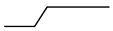
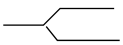
bus structure

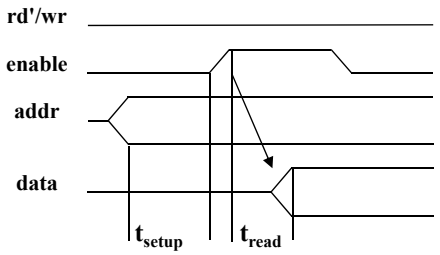
Ports



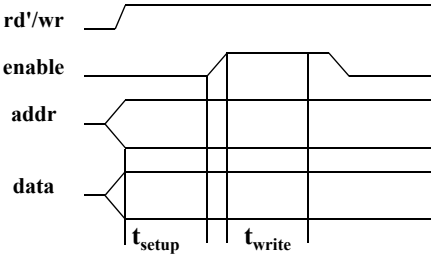
- Conducting device on periphery
- Connects bus to processor or memory
- Often referred to as a *pin*
  - Actual pins on periphery of IC package that plug into socket on printed-circuit board
  - Sometimes metallic balls instead of pins
  - Today, metal "pads" connecting processors and memories within single IC
- Single wire or set of wires with single function
  - E.g., 12-wire address port

Timing Diagrams

- Most common method for describing a communication protocol
- Time proceeds to the right on x-axis
- Control signal: low or high 
  - May be active low (e.g., go', /go, $\overline{go}$, or go_L)
  - Use terms *assert* (active) and *deassert*
  - Asserting go' means go=0
- Data signal: not valid or valid 
- Protocol may have subprotocols
  - Called bus cycle, e.g., read and write
  - Each may be several clock cycles



read protocol

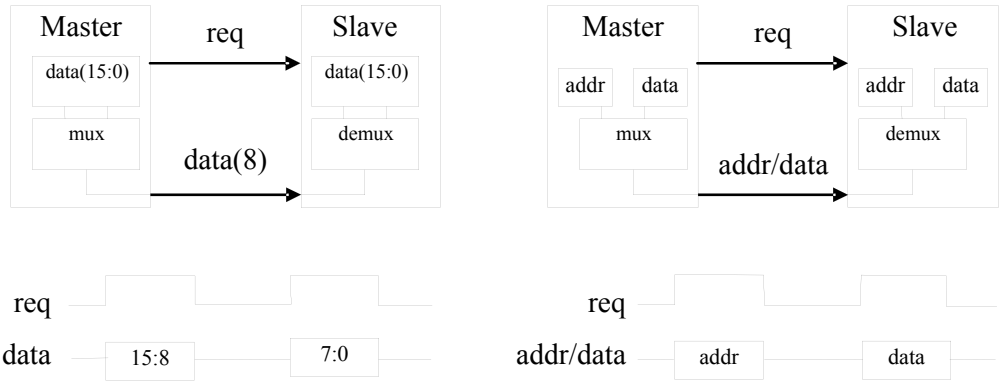


write protocol

Basic protocol concepts

- Actor: master initiates, slave (servant) responds
- Direction: sender, receiver
- Addresses: special kind of data
  - Specifies a location in memory, a peripheral, or a register within a peripheral
- Time multiplexing
  - Share a single set of wires for multiple pieces of data
  - Saves wires at expense of time

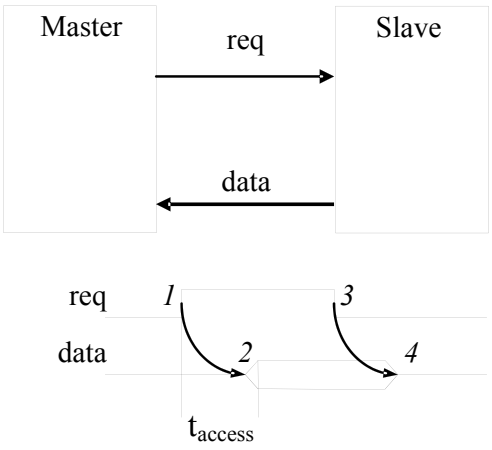
Time-multiplexed data transfer



data serializing

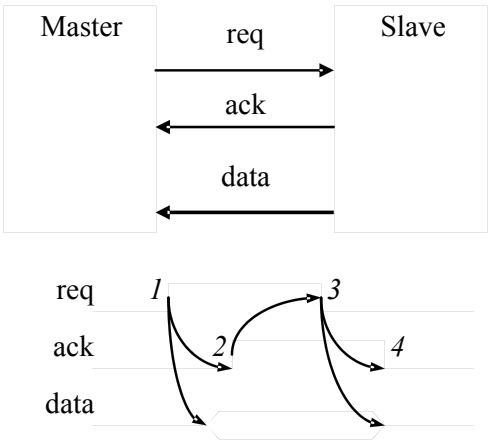
address/data muxing

Basic protocol concepts: control methods



1. Master asserts *req* to receive data

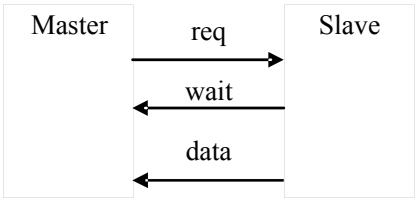
STROBE PROTOCOL

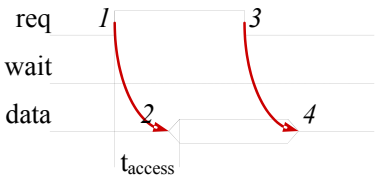


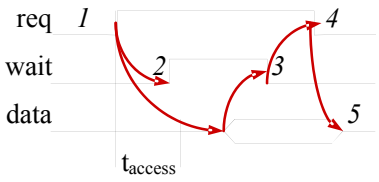
1. Master asserts *req* to receive data
2. Slave puts data on bus and asserts ***ack***
3. Master receives data and deasserts *req*
4. Slave ready for next request

HANDSHAKE PROTOCOL

A strobe/handshake compromise







1. Master asserts ***req*** to receive data
2. Slave puts data on bus **within time t_{access}** (wait line is unused)
3. Master receives data and **deasserts *req***
4. Slave ready for next request

FAST - RESPONSE CASE

1. Master asserts *req* to receive data
2. Slave can't put data within **t_{access}, asserts *wait ack***
3. Slave puts data on bus and **deasserts *wait***
4. Master receives data and **deasserts *req***
5. Slave ready for next request

SLOW - RESPONSE CASE

Types of bus-based I/O: memory-mapped I/O and standard I/O

Remember this?

- Processor talks to both memory and peripherals using same bus: two ways to talk to peripherals
  - **Memory-mapped I/O**
    - Peripheral registers occupy addresses in same address space as memory
    - e.g., Bus has 16-bit address
      - lower 32 K addresses may correspond to memory
      - upper 32 K addresses may correspond to peripherals
  - **Standard I/O (I/O-mapped I/O)**
    - Additional pin (*M/IO*) on bus indicates whether a memory or peripheral access is to be performed
    - e.g., Bus has 16-bit address
      - all 64 K addresses correspond to memory when *M/IO* set to 0
      - all 64 K addresses correspond to peripherals when *M/IO* set to 1

Memory-mapped I/O vs. Standard I/O

- Memory-mapped I/O
  - Requires no special instructions
    - Assembly instructions involving memory like MOV and ADD work with peripherals as well
    - Standard I/O requires special instructions (e.g., IN, OUT) to move data between peripheral registers and memory
- Standard I/O
  - No loss of memory addresses to peripherals
  - Simpler address decoding logic in peripherals possible
    - When number of peripherals much smaller than address space then high-order address bits can be ignored
      - smaller and/or faster comparators

Advanced communication principles

- Layering
  - Break complexity of communication protocol into pieces easier to design and understand
  - Lower levels provide services to higher level
    - Lower level might work with **bits** while higher level might work with **packets of data**
  - **Physical layer**
    - **Lowest** level in hierarchy
    - **Medium** to carry data from one *actor* (device or node) to another
- Parallel communication
  - Physical layer capable of transporting multiple bits of data
- Serial communication
  - Physical layer transports one bit of data at a time
- Wireless communication
  - No physical connection needed for transport at physical layer

Parallel communication

- Multiple data, control, and possibly power wires
  - One bit per wire
- High data throughput with short distances
- Typically used when connecting devices on same IC or same circuit board
  - Bus must be kept short
    - long parallel wires result in high capacitance values which requires more time to charge/discharge
    - Data misalignment between wires increases as length increases
- Higher cost, bulky

Serial communication

- Single data wire, possibly also control and power wires
- Words transmitted one bit at a time
- Higher data throughput with long distances
  - Less average capacitance, so more bits per unit of time
- Cheaper, less bulky
- More complex interfacing logic and communication protocol
  - Sender needs to decompose word into bits
  - Receiver needs to recompose bits into word
  - Control signals often sent on same wire as data increasing protocol complexity

Wireless communication

- Infrared (IR)
  - Electronic wave frequencies just below visible light spectrum
  - Diode emits infrared light to generate signal
  - Infrared transistor detects signal, conducts when exposed to infrared light
  - Cheap to build
  - Need line of sight, limited range
- Radio frequency (RF)
  - Electromagnetic wave frequencies in radio spectrum
  - Analog circuitry and antenna needed on both sides of transmission
  - Line of sight not needed, transmitter power determines range

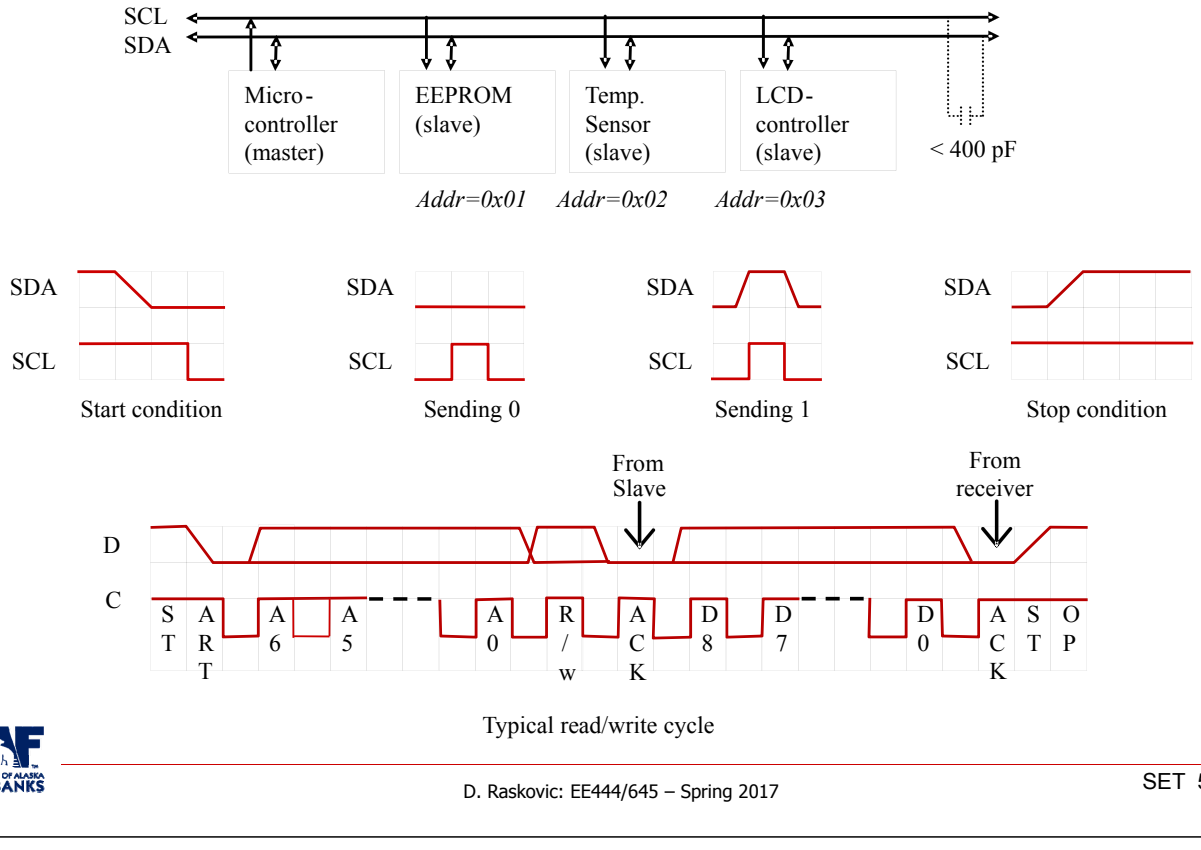
Error detection and correction

- Often part of bus protocol
- Error detection: ability of receiver to detect errors during transmission
- Error correction: ability of receiver and transmitter to cooperate to correct problem
  - Typically done by acknowledgement/retransmission protocol and/or
  - Use of error correcting codes
- Bit errors: a single bit is inverted
- **Burst errors**: consecutive bits received incorrectly
- Parity: extra bit sent with word used for error detection
  - Odd parity, even parity
  - Always detects single bit errors, but not all burst bit errors
- Checksum: extra word sent with data packet of multiple words
  - e.g., extra word contains XOR sum of all data words in packet

Serial protocols: I²C

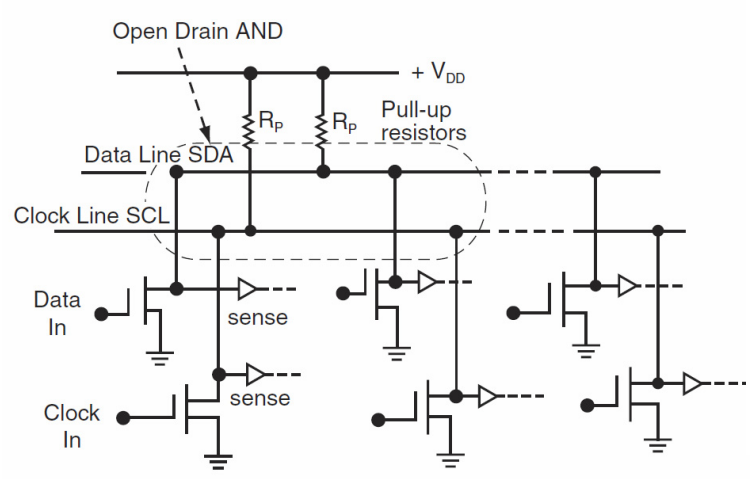
- I²C (Inter-IC)
  - Two-wire serial bus protocol developed by Philips Semiconductors
  - Enables peripheral ICs to communicate using simple communication hardware
  - Data transfer rates of up to 100 kbits/s and 7-bit addressing in normal mode.
  - 3.4 Mbits/s and 10-bit addressing in fast-mode
  - Common devices capable of interfacing to I²C bus:
    - EPROMS, Flash, and some RAM memory, real-time clocks, watchdog timers, many sensors, and microcontrollers

I²C bus structure



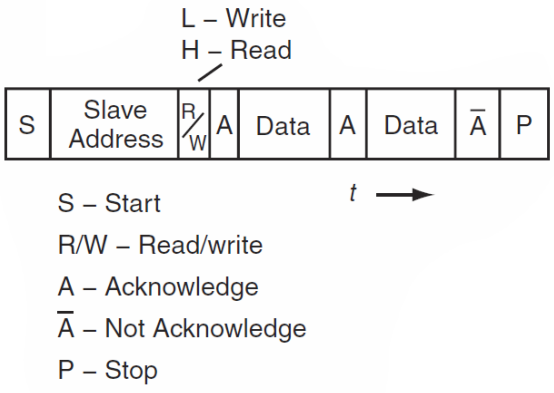
Typical read/write cycle

I²C Bus

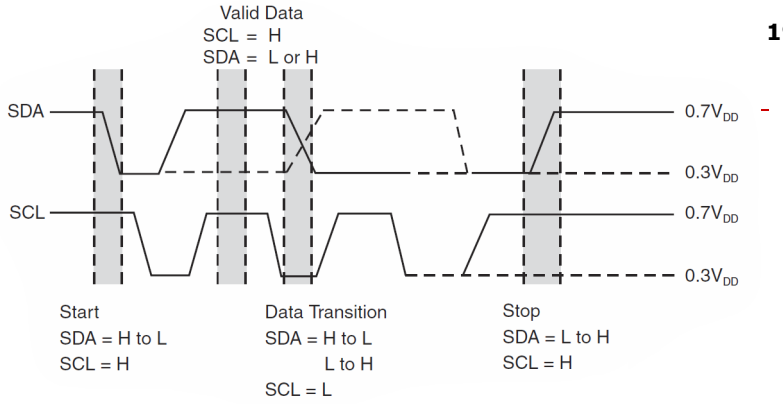


I²C Bus Schematic

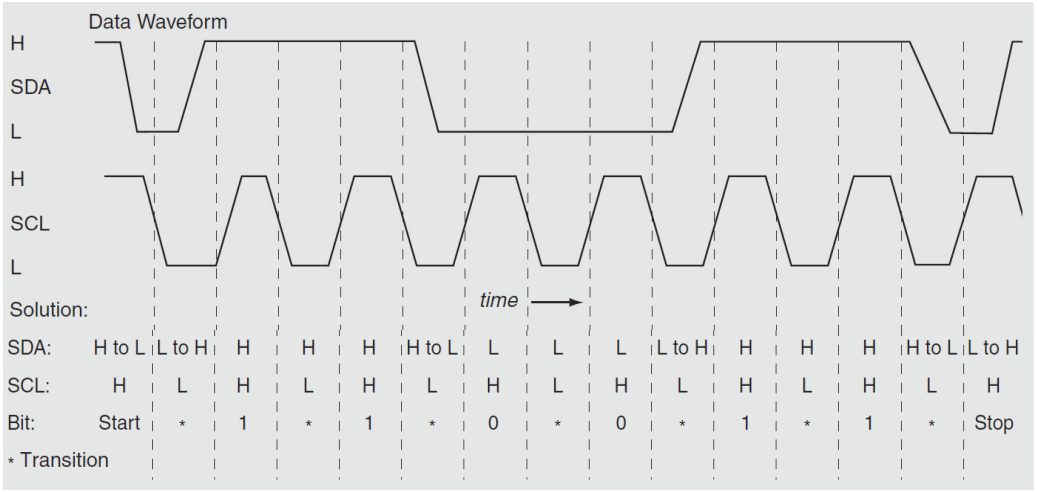
Typical master-transmitter slave-receiver format



I2C – An Example



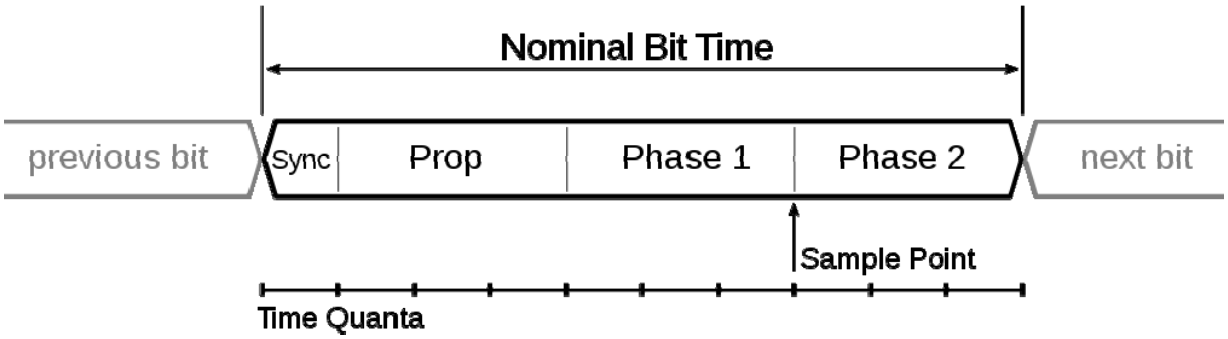
Determine the data bits in the I2C waveform shown:



Serial protocols: CAN

- CAN (Controller area network)
  - Protocol for real-time applications
  - Developed by Robert Bosch GmbH
  - Originally for communication among components of cars
  - Applications now using CAN include:
    - elevator controllers, copiers, telescopes, production-line control systems, and medical instruments
  - Data transfer rates up to 1 Mbit/s and 11-bit addressing
  - Actual physical design of CAN bus not specified in protocol
    - Requires devices to transmit/detect dominant and recessive signals to/from bus
    - e.g., '1' = dominant, '0' = recessive if single data wire used
    - Bus guarantees dominant signal prevails over recessive signal if asserted simultaneously
  - Bosh holds patents, manufacturers pay license fees to Bosh
    - Still relatively inexpensive

CAN bit timing



Serial protocols: LIN

- **L**ocal **I**nterconnection **N**etwork
- Started by 5 carmakers and Motorola as a low-cost companion to CAN
- Technical specifications:
  - Low cost single-wire implementation (enhanced ISO 9141)
  - Speed up to 20Kbit/s (limited for EMI-reasons)
  - Single Master / Multiple Slave Concept
    - No arbitration necessary
  - Low cost silicon implementation based on common UART interface hardware
    - Almost any microcontroller has necessary hardware on chip
  - Self synchronization without crystal or ceramics resonator in the slave nodes
    - Significant cost reduction of hardware platform
  - Guaranteed latency times for signal transmission (Predictability)

Serial protocols: FireWire

- FireWire (a.k.a. IEEE 1394)
  - High-performance serial bus developed by Apple Computer Inc.
  - Designed for interfacing independent electronic components
    - e.g., desktop (network), scanner, camera, external hard drive
  - Data transfer rates from 12.5 to 400 (800) Mbits/s, 64-bit addressing
  - Plug-and-play capabilities
  - Packet-based layered design structure
  - Capable of supporting a LAN similar to Ethernet
    - 64-bit address:
      - 10 bits for network ids, 1023 subnetworks
      - 6 bits for node ids, each subnetwork can have 63 nodes
      - 48 bits for memory address, each node can have 281 terabytes of distinct locations

Serial protocols: USB

- USB (Universal Serial Bus)
  - 2 data rates:
    - 12 Mbps for increased bandwidth devices
    - 1.5 Mbps for lower-speed devices (joysticks, game pads)
  - Tiered star topology can be used
    - One (or more) USB device (hub) connected to PC
      - hub can be embedded in devices like monitor, printer, or keyboard or can be standalone
      - Multiple USB devices can be connected to hub
      - Up to 127 devices can be connected like this
  - USB host controller
    - Manages and controls bandwidth and driver software required by each peripheral
    - Dynamically allocates power downstream according to devices connected/disconnected
  - USB2 – 480 Mbps (“Hi-Speed”)
  - USB3 – 3.2 Gbps after protocol overhead (“SuperSpeed”)

Serial Bus Standards - examples

Characteristic	FireWire (IEEE 1394, 1394b)	USB 2.0
Bus type	I/O	I/O
Basic data bus width	4	2
Clocking	Asynchronous	Asynchronous
Theoretical peak bandwidth	50 MB/sec (FW 400) 100 MB/sec (FW 800)	0.2 MB/sec (low speed) 1.5 MB/sec (full speed) 60 MB/sec (high speed)
Hot pluggable	Yes	Yes
Maximum number of devices	63	127
Maximum bus length	4.5 m	5 m

Parallel protocols - examples

- PCI Bus (Peripheral Component Interconnect)
  - High performance bus originated at Intel in the early 1990's
  - Standard adopted by industry and administered by PCISIG (PCI Special Interest Group)
  - Interconnects chips, expansion boards, processor memory subsystems
  - Data transfer rates of 127.2 to 508.6 Mbits/s and 32-bit addressing
    - Later extended to 64-bit while maintaining compatibility with 32-bit schemes
  - Synchronous bus architecture
  - Multiplexed data/address lines
- Many IC design companies have their own bus protocol
  - ARM Bus
    - Designed and used internally by ARM Corporation
    - Interfaces with ARM line of processors
    - Data transfer rate is a function of clock speed
    - 32-bit addressing
  - MSP430 variable width bus,
  - Etc.

Wireless protocols: IrDA

- IrDA
  - Protocol suite that supports short-range point-to-point infrared data transmission
  - Created and promoted by the Infrared Data Association (IrDA)
  - Data transfer rate of 9.6 kbps and more:
    - SIR, MIR, FIR, VFIR, UFIR, and Giga-IR
    - Giga-IR up to 1 Gbps
  - IrDA hardware deployed in notebook computers, printers, PDAs, digital cameras, public phones, cell phones since '90s
  - Lack of suitable drivers has slowed use by applications
  - Making a comeback:
    - IrSimple – transfer a cell phone image in less than 1 second
    - Less expensive than Bluetooth
    - No security issues
  - MSP430 supports it

Wireless protocols: Bluetooth

- Bluetooth
  - Global standard for wireless connectivity
  - Based on low-cost, short-range radio link
  - Connection established when within ~10 meters of each other
  - No line-of-sight required
    - e.g., Connect to printer in another room

Wireless protocols: Bluetooth

- IEEE 802.15.1
- 2.4 GHz radio spectrum (2402 – 2480 MHz), worldwide
- Spread spectrum frequency hopping, full-duplex signal at up to 1600 hops per second
- Intervals between hops are 1 MHz wide, 79 frequencies in total
- One master can manage up to seven slaves
- Different data rates in different versions of standards
  - 1 Mbps in v1.1 and v1.2 (in practice: ~720 kbps)
  - 3 Mbps with EDR in v2.0 (in practice: ~2.1 Mbps)
- v2.1 – lower power consumption in the sniff low-power mode
  - Added NFC pairing
- v3.0 (2009) – use Bluetooth link for negotiation and 802.11 for (optional) high-speed data connection (24 Mbps). It was planned to use UWB for speeds of up to 480 Mbps
- v4.0 Smart (2010) – includes *Classic, Low Energy, and High Speed*

Bluetooth Low Energy (BLE)

- Aimed at healthcare, fitness, security, and home entertainment
  - Many profiles defined (Glucose, Blood pressure, Location and navigation, Find me profile, etc.)
- Same 2.4 GHz ISM band, but simpler modulation
- Most phones have it (iPhone 4s+, Samsung Galaxy S3+, etc.)

Classic v. Low Energy

Technical Specification	Classic Bluetooth technology	BLE technology
Distance/Range	100 m (330 ft)	50 m (160 ft)
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.27 Mbit/s
Active slaves	7	Not defined; implementation dependent
Latency (from a non-connected state)	Typically 100 ms	6 ms
Total time to send data (det.battery life)	100 ms	3 ms, <3 ms
Voice capable	Yes	No
Network topology	Scatternet	Star-bus
Power consumption	1 as the reference	0.01 to 0.5 (depending on use case)
Peak current consumption	<30 mA	<15 mA
Service discovery	Yes	Yes
Profile concept	Yes	Yes

Wireless protocols: ZigBee

- IEEE 802.15.4
- Low-cost and very-low-power wireless connectivity at transmission distances between 10-75 meters
- Features
  - Data rates: 250 kbps, 40 kbps, or 20 kbps.
  - 255 devices per network.
  - Radio interface:
    - 2.4GHz ISM band (Global, 16 channels) – 250 kbps
    - 915MHz (Americas, 10 ch) – 40 kbps
    - 868MHz band (Europe, 1 ch) – 20 kbps
  - Support for critical latency devices, such as joysticks.
  - Automatic network establishment by the coordinator.
  - Fully handshaked protocol for transfer reliability.
  - Power management to ensure low power consumption.

ZigBee® Alliance

ZigBee v. classic Bluetooth

Parameter	Zigbee	Bluetooth
Frequency band	2.4 GHz	2.4 GHz
Modulation technique	Direct Sequence Spread Spectrum (DSSS)	Frequency Hopping Spread Spectrum (FHSS)
Protocol stack size	4- 32 KB	250 KB
Battery changes	Rare	Intended for frequent recharges
Max bandwidth	250 Kb/s	750 Kb/s
Max range	up to 70 meters	1 – 100 meters
Typical network join time	30 ms	3 sec
Network size	65536	8

- **ZigBee**
  - Smaller packets over large network
  - Mostly Static networks with many, infrequently used devices
  - Home automation, toys, remote controls, etc.
- **Bluetooth**
  - Larger packets over small network
  - Ad-hoc networks
  - File transfer
  - Screen graphics, pictures, hands-free audio, Mobile phones, headsets, PDAs, etc.

USART Peripheral Interface in MSP430

- Older chips: USART - Two modes of operation:
  - UART – Asynchronous, or
  - SPI – Synchronous
- Newer chips: USCI (Universal Serial Communication Interface)
  - UART/LIN/IrDA/SPI (Module(s) A)
  - I^2C/SPI (Module(s) B)

asynchronous transmission,

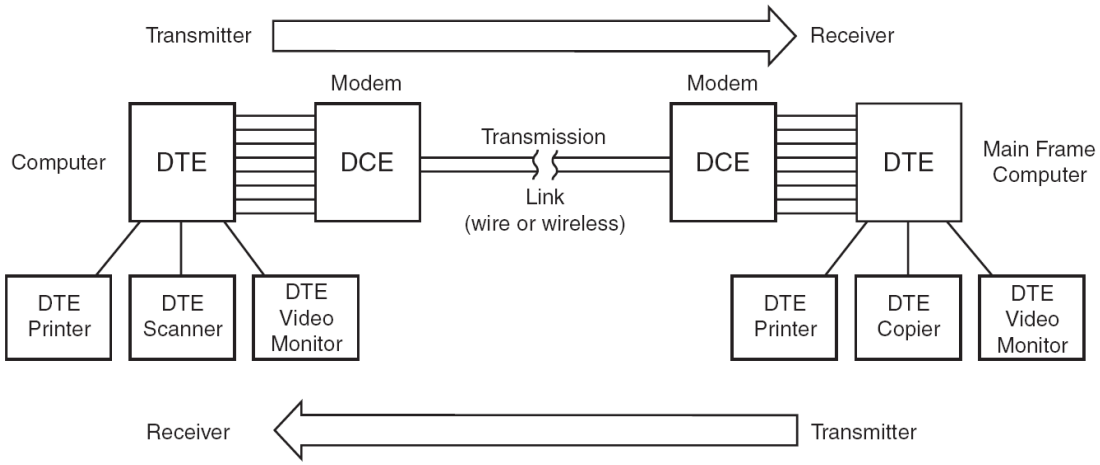
electronic communication between digital devices, as two separate computers that run at different speeds, that requires start and stop bits for each character transmitted.

- For example:
  - MSP430x44x – USART0 + USART1
  - MSP430FG4619 – USART + USCI

Serial Communication Background

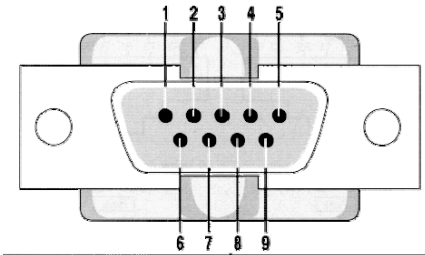
Danger! Ancient history! ☺

- The EIA standard RS-232-C in 1969 (EIA-232)
- Commonly known as RS-232, a standard for serial binary communications between a DTE (Data Terminal Equipment, i.e., computer, terminal) and a DCE (Data Communication Equipment, i.e., modem).
  - DTE, DCE use different pins to send/receive data
  - Problem today: what is a PDA? What is a GPS? DTE or DCE?
  - One solution: use the oscilloscope and check!



Serial Communication Background, Cont'd

- As a connector, the standard recommended a D-Subminiature 25-pin connector (one of the largest connectors you can find today ☺)
- D-Subminiature (or D-Sub):
  - Invented by Cannon
  - Part numbering: DsNg, where
    - s – shell size (A=15, B=25, C=37, D=50, E=9 pins)
    - N – the actual number of pins (see above)
    - g – gender (M for Male, F for Female)
    - For example, DB9M or DB9F, much more commonly used for RS232



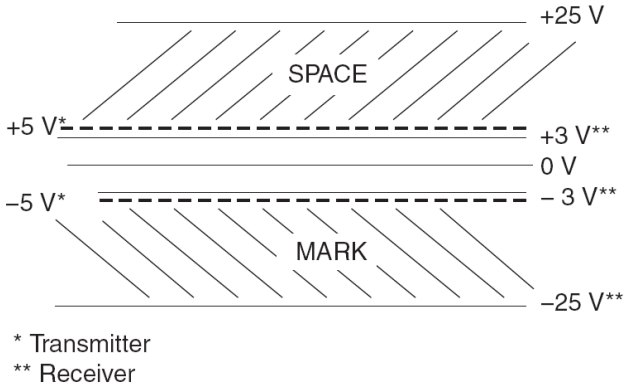
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Danger! Ancient history! ☺

Serial Communication Background

- **Originally,**
  - **Logic one** defined as a **negative voltage** (-3 V ÷ -25 V), **MARK**
  - **Logic zero** defined as a **positive voltage** (+3 V ÷ +25 V), **SPACE**

- More common voltage levels today: ±5 V,±10 V,±12 V, and ±15 V, depending on device's power supply
- Voltage levels
  - common ground very important
- Compare this with TTL and CMOS logic levels



- Originally, defined with (for today's standards):
  - Huge voltage swing for 0 and 1 representation (max ±25 V)
  - Short cables (50 ft)
  - Enormous connector (25-pin), etc

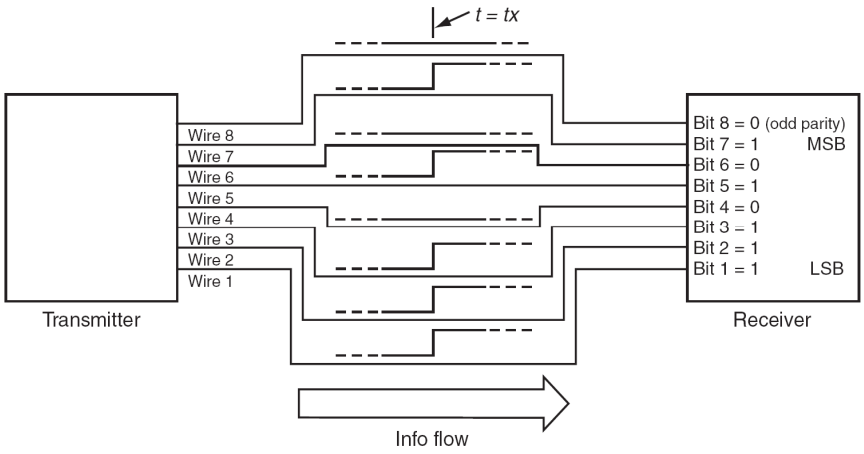
ASCII Table

Remember

		$b_6b_5b_4$ (column)							
$b_3b_2b_1b_0$	Row (hex)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	‘	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	’	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

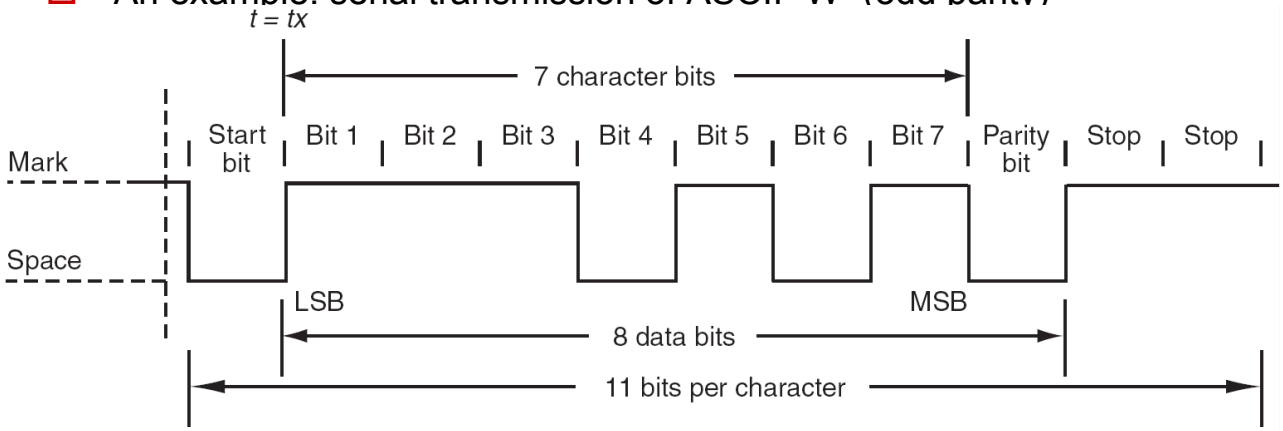
Parallel v. Serial

- An example: parallel transmission of ASCII "W" (odd parity)



Parallel v. Serial

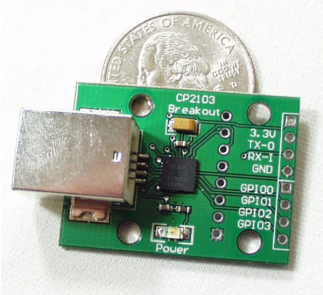
- An example: serial transmission of ASCII "W" (odd parity)

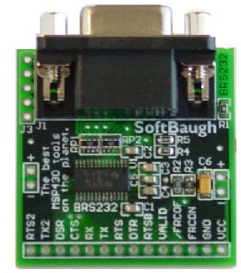


	PARALLEL	SERIAL
Lines Required	One line/bit	Single line
Bit Sequence	On all lines at same time	One bit following another
Speed	Faster	Slower ?
Transmission line length	Usually a short distance	Both long and short distances
Cost	More expensive	Less expensive
Critical Characteristic	Time relationship of bits	Needs start, stop bit

Serial Communication Background, Cont'd

- For a microcontroller with a Vcc of 3 V or less, you might need a "shifter" to adjust signal levels so that it can talk to other serial devices
- Typically implemented using one of the RS232 transceiver chips





PC, GPS, older equipment, etc.

Microcontroller interface

- Typical speeds: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, …

Universal Serial Communications Interface

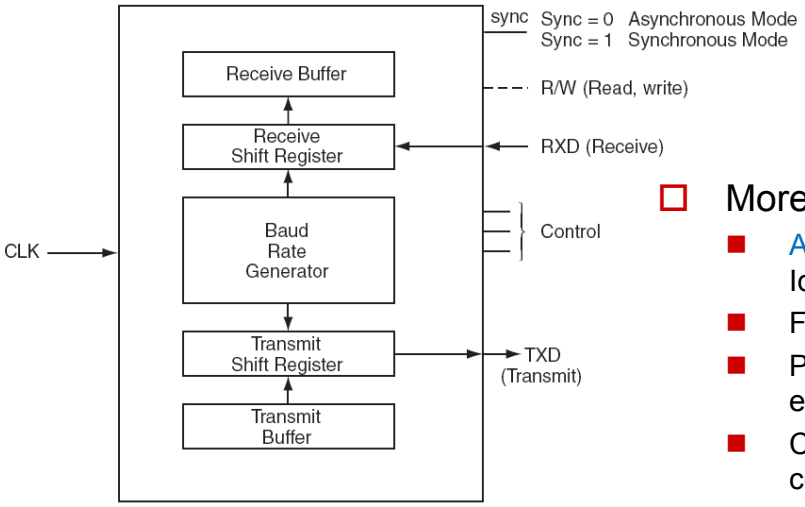
- USCI_Ax modules support:
  - UART mode
  - Pulse shaping for IrDA communications
  - Automatic baud-rate detection for LIN communications
  - SPI mode
- USCI_Bx modules support:
  - I2C mode
  - SPI mode

Device	Flash (KB)	SRAM (KB)	Timer_A(1)	Timer_B(2)	USCI	
					Channel A: UART/IrDA/ SPI	Channel B: SPI/I^2C
MSP430F5438A	256	16	5, 3	7	4	4

USCI UART Mode

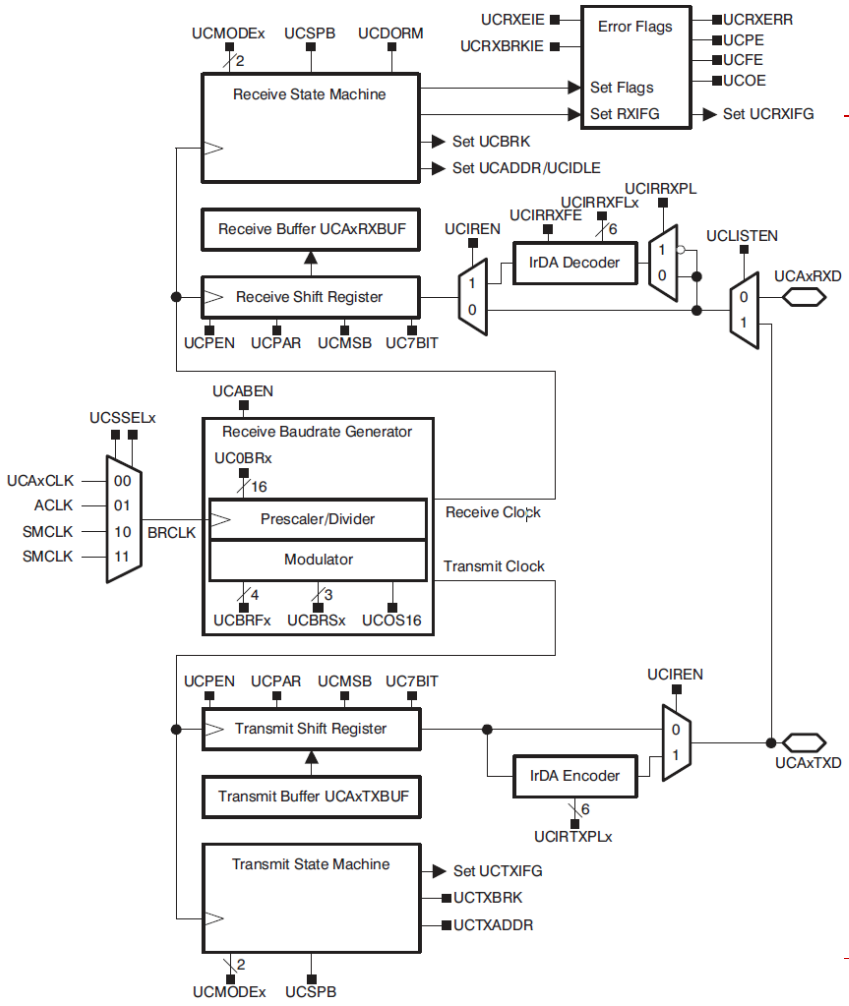
- UART mode features include:
  - 7- or 8-bit data with odd, even, or non-parity
  - Independent transmit and receive shift registers
  - Separate Tx and Rx buffer registers
  - LSbit-first data transmit and receive (MSbit also possible)
  - Built-in support for multiprocessor systems
  - Auto-wake up from LPMx modes
  - Programmable baud rate with modulation for fractional baud rate support
  - Status flags for error detection/suppression and address detection
  - Independent interrupt capability for receive and transmit
- Also can be configured for IrDA and LIN functionality

USCI UART Mode

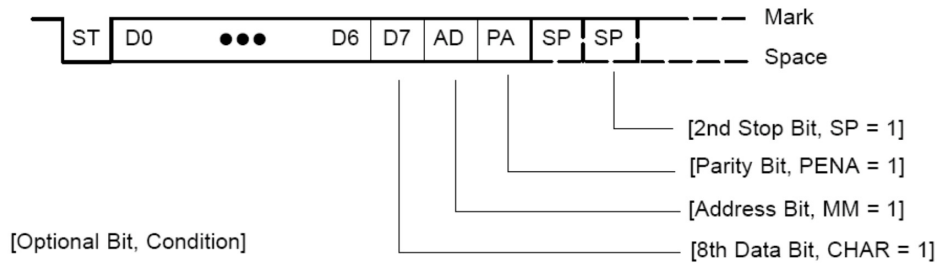


- More UART features:
  - Asynchronous modes, including Idle line/Address bit protocols
  - Full Duplex
  - Programmable internal or external baud generation
  - Clock polarity and clock phase control
  - De-glitch suppression
  - Parity Generation / Detection
  - Frame, Parity and Over-run error detection (suppression)

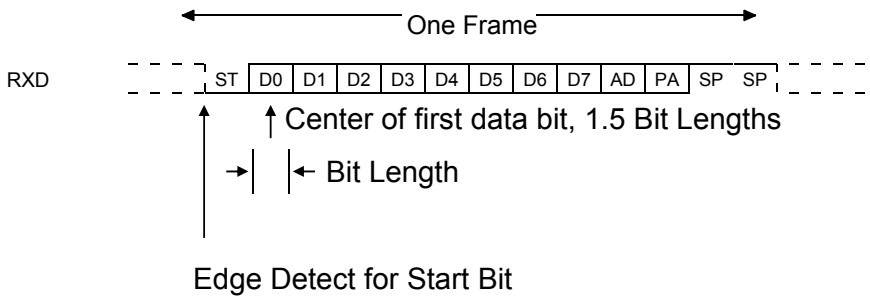
MSP430x5xx/6xx USCI



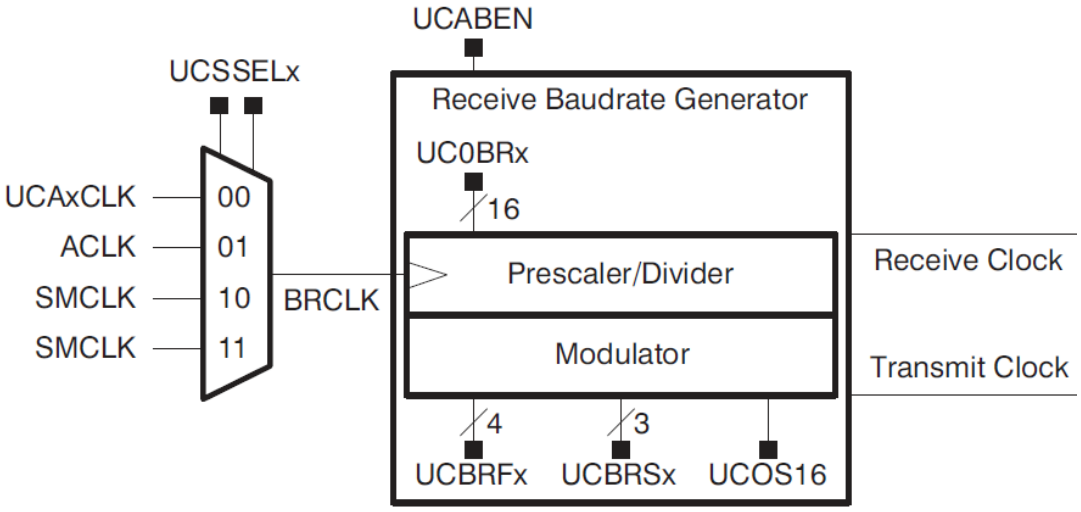
Character format



Considerations for Software / Hardware UART Solution.



UART Baud Rate Generation



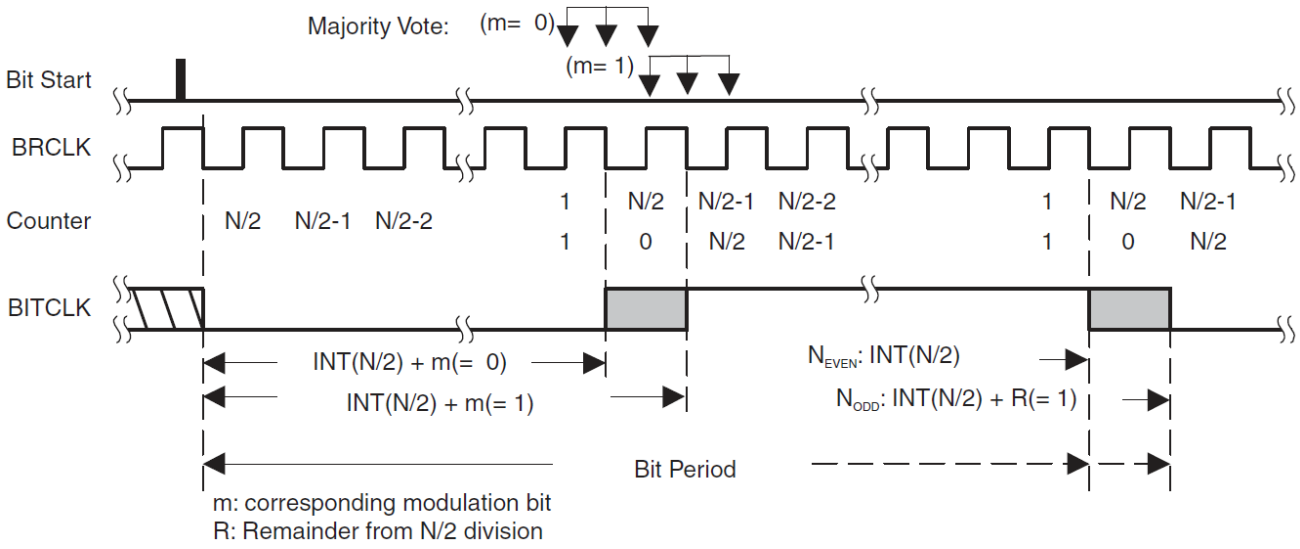
Baud Rate Timing – 2xx, 5xx/6xx

$$N = \frac{f_{BRCLK}}{Baudrate}$$ ← Clock source

- Two modes of operation:
  - Low-frequency baud-rate generation (UCOS16 in UCAxMCTL == 0)
  - Oversampling baud-rate generation (UCOS16 in UCAxMCTL == 1)
- Low-frequency mode
  - Use with low frequency sources
  - Set the integer portion of the divisor: UCBRx = INT(N)
  - Set the fractional portion: UCBRSx = round[(N – INT(N)) × 8]
  - Modulation pattern:

UCBRSx	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

Baud Rate Timing – 5xx/6xx



Baud Rate Timing – Oversampling Mode

- Uses one prescaler and one modulator to generate the clock that is 16x faster than the BITCLK (BITCLK16)
  - This stage can be bypassed
- Additional stage is used to generate BITCLK from BITCLK16
- The maximum USCI baud rate is 1/16 of the source clock
- Setting:
  - Set the prescaler to: UCBRx = INT(N/16)
  - Set the first stage modulator to: UCBRFx = round([(N/16) – INT(N/16)] × 16)
  - Use UCBRSx (values 0 – 7) if you need more precision

Oversampling bit timing

- Here's how to calculate the error:
  - The length of bit *i*:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}} \left((16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j] \right)$$

  - The end-of-bit time is then:

$$t_{bit,TX}[i] = \sum_{j=0}^{i} T_{bit,TX}[j]$$

  - The ideal end-of-bit time is:

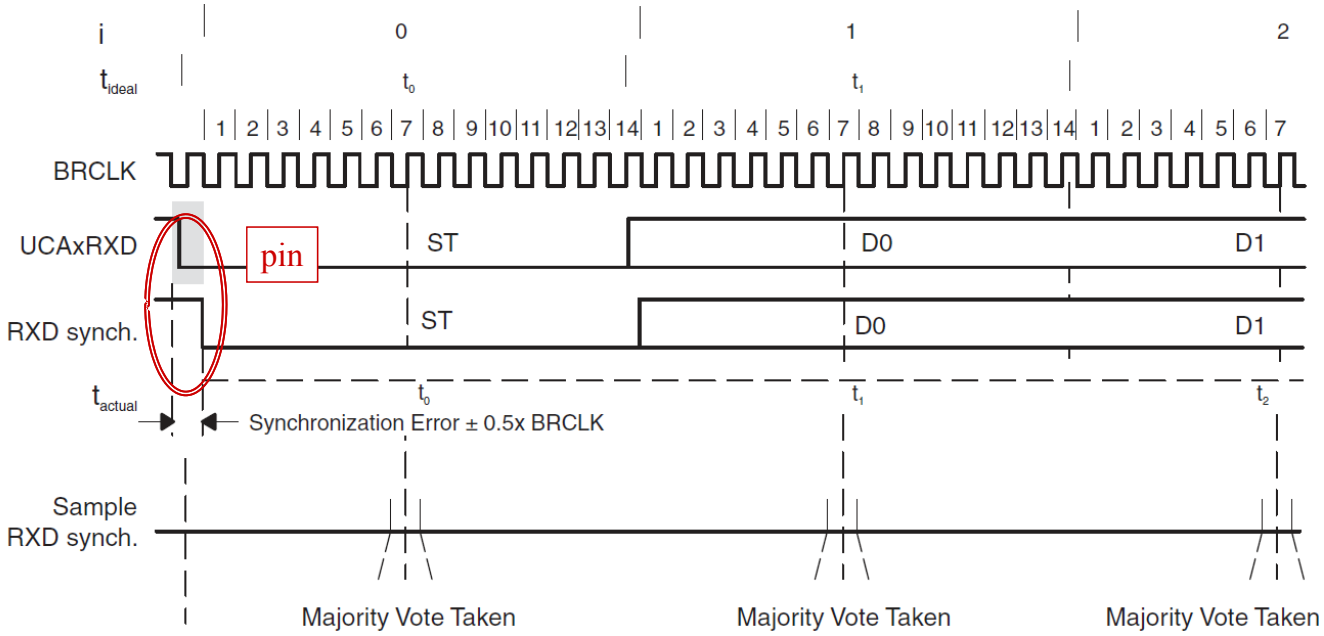
$$t_{bit,ideal,TX}[i] = (1/Baudrate)(i + 1)$$

  - An error, normalized to one ideal bit time (1/baudrate) is:

$$Error_{TX}[i] = (t_{bit,TX}[i] - t_{bit,ideal,TX}[i]) \times Baudrate \times 100\%$$

  - To reduce error, you might have to inc/dec UCBRFx

Baud Rate Timing – Receive Error



- Additional error on a receiving side:
  - between a start edge occurring and the start edge being accepted by the USART.

Table 34-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
32,768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	57600	17	3	0	-2.1	4.8	-6.8	5.8
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,048,576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1,048,576	57600	18	1	0	-4.6	3.3	-6.8	6.6
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
4,000,000	9600	416	6	0	-0.2	0.2	-0.2	0.4
4,000,000	19200	208	3	0	-0.2	0.5	-0.3	0.8
4,000,000	38400	104	1	0	-0.5	0.6	-0.9	1.2
4,000,000	57600	69	4	0	-0.6	0.8	-1.8	1.1
4,000,000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4,000,000	230400	17	3	0	-2.1	4.8	-6.8	5.8
4,194,304	9600	436	7	0	-0.3	0	-0.3	0.2

Table 34-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (continued)

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
4,194,304	19200	218	4	0	-0.2	0.2	-0.3	0.6
4,194,304	57600	72	7	0	-1.1	0.6	-1.3	1.9
4,194,304	115200	36	3	0	-1.9	1.5	-2.7	3.4
8,000,000	9600	833	2	0	-0.1	0	-0.2	0.1
8,000,000	19200	416	6	0	-0.2	0.2	-0.2	0.4
8,000,000	38400	208	3	0	-0.2	0.5	-0.3	0.8
8,000,000	57600	138	7	0	-0.7	0	-0.8	0.6
8,000,000	115200	69	4	0	-0.6	0.8	-1.8	1.1
8,000,000	230400	34	6	0	-2.1	0.6	-2.5	3.1
8,000,000	460800	17	3	0	-2.1	4.8	-6.8	5.8
8,388,608	9600	873	7	0	-0.1	0.06	-0.2	0,1
8,388,608	19200	436	7	0	-0.3	0	-0.3	0.2
8,388,608	57600	145	5	0	-0.5	0.3	-1.0	0.5
8,388,608	115200	72	7	0	-1.1	0.6	-1.3	1.9
12,000,000	9600	1250	0	0	0	0	-0.05	0.05
12,000,000	19200	625	0	0	0	0	-0.2	0
12,000,000	38400	312	4	0	-0.2	0	-0.2	0.2
12,000,000	57600	208	2	0	-0.5	0.2	-0.6	0.5
12,000,000	115200	104	1	0	-0.5	0.6	-0.9	1.2
12,000,000	230400	52	0	0	-1.8	0	-2.6	0.9
12,000,000	460800	26	0	0	-1.8	0	-3.6	1.8
16,000,000	9600	1666	6	0	-0.05	0.05	-0.05	0.1

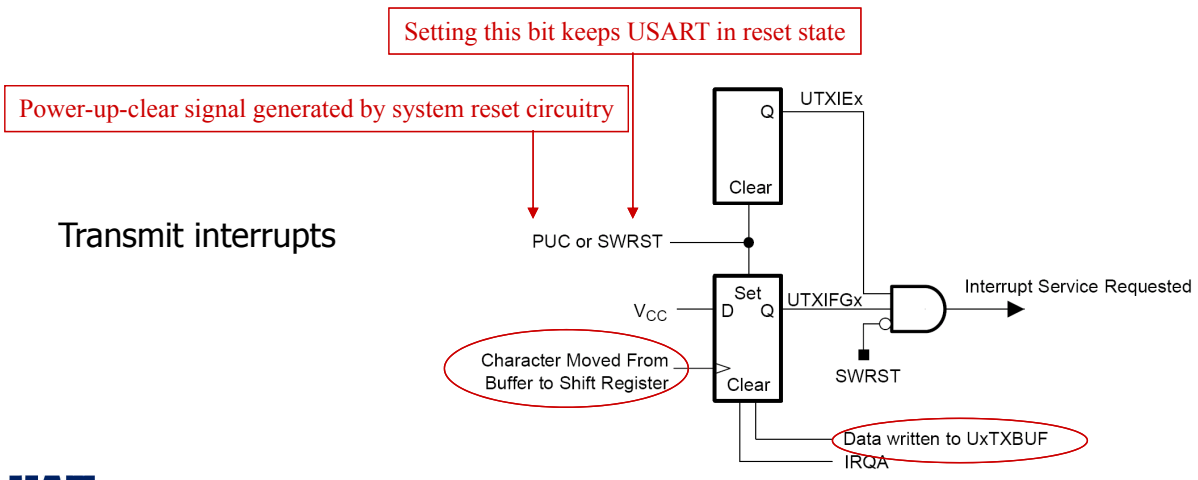
Table 34-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1

BRCLK Frequency (Hz)	Baud Rate (baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
1,000,000	9600	6	0	8	-1.8	0	-2.2	0.4
1,000,000	19200	3	0	4	-1.8	0	-2.6	0.9
1,048,576	9600	6	0	13	-2.3	0	-2.2	0.8
1,048,576	19200	3	1	6	-4.6	3.2	-5.0	4.7
4,000,000	9600	26	0	1	0	0.9	0	1.1
4,000,000	19200	13	0	0	-1.8	0	-1.9	0.2
4,000,000	38400	6	0	8	-1.8	0	-2.2	0.4
4,000,000	57600	4	5	3	-3.5	3.2	-1.8	6.4
4,000,000	115200	2	3	2	-2.1	4.8	-2.5	7.3
4,194,304	9600	27	0	5	0	0.2	0	0.5
4,194,304	19200	13	0	10	-2.3	0	-2.4	0.1
4,194,304	57600	4	4	7	-2.5	2.5	-1.3	5.1
4,194,304	115200	2	6	3	-3.9	2.0	-1.9	6.7
8,000,000	9600	52	0	1	-0.4	0	-0.4	0.1
8,000,000	19200	26	0	1	0	0.9	0	1.1
8,000,000	38400	13	0	0	-1.8	0	-1.9	0.2
8,000,000	57600	8	0	11	0	0.88	0	1.6
8,000,000	115200	4	5	3	-3.5	3.2	-1.8	6.4
8,000,000	230400	2	3	2	-2.1	4.8	-2.5	7.3
8,388,608	9600	54	0	10	0	0.2	-0.05	0.3
8,388,608	19200	27	0	5	0	0.2	0	0.5

USCI Interrupts

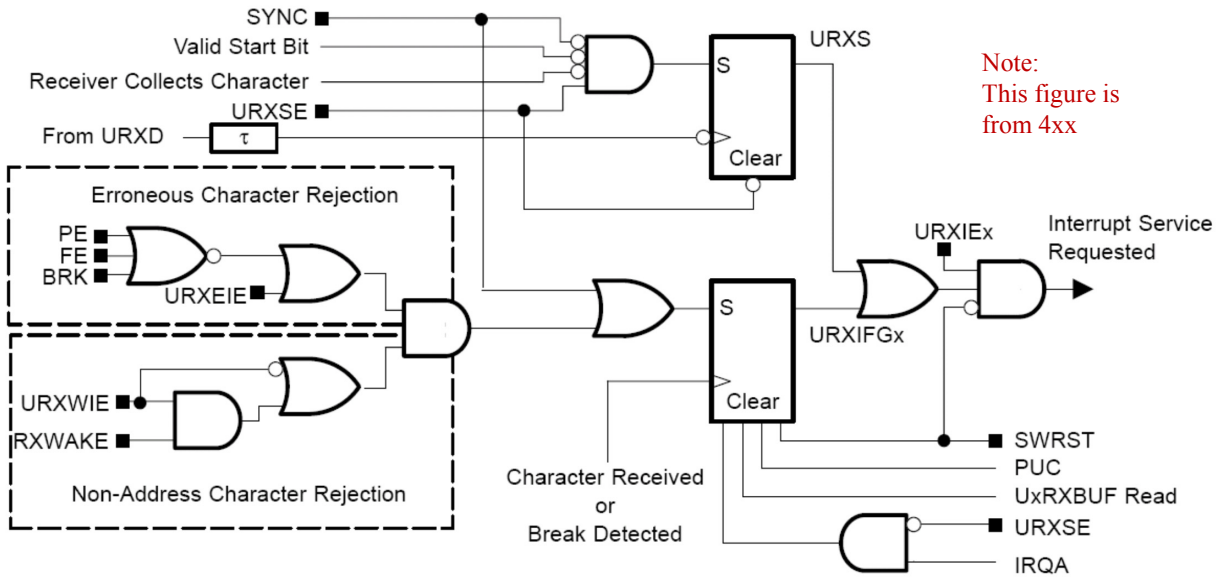
- Single interrupt vector, shared by Tx and Rx
- USCI_Ax and USCI_Bx do not share the same interrupt vector
- UCTXIFG is set to indicate that UCAxTXBUF is empty (ready to accept another character)
  - If the corresponding UCTXIE bit is set and global interrupts are enabled, an interrupt is generated
- UTXIFGx is reset if a character is written to UxTXBUF

Transmit interrupts



USCI Receive Interrupts

- UCRXIFG set to indicate that a character is received and loaded into UCAxRXBUF
  - If UCRXIE is set and global interrupts are enabled (GIE), an interrupt is generated
- UCRXIFG and UCRXIE are reset on PUC or when UCSWRST = 1
- UCRXIFG is reset when UCAxRXBUF is read

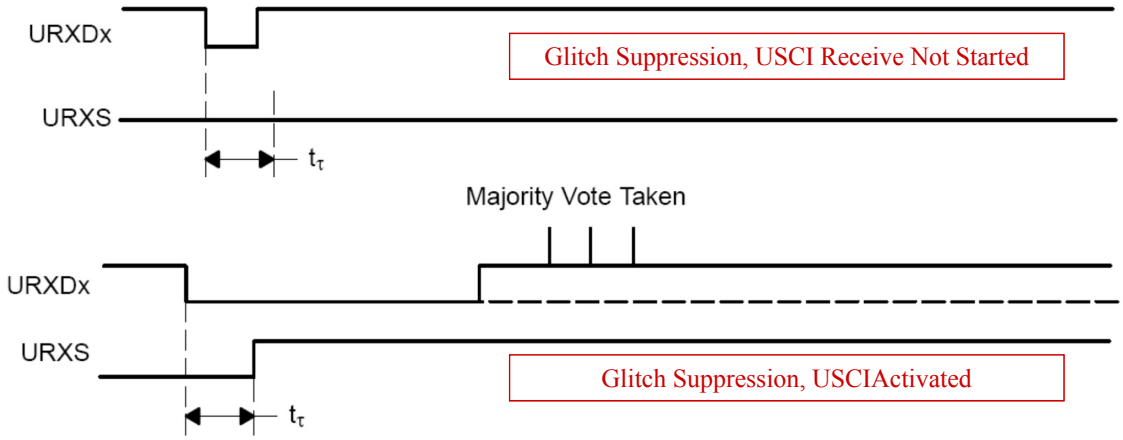


Automatic Error Detection

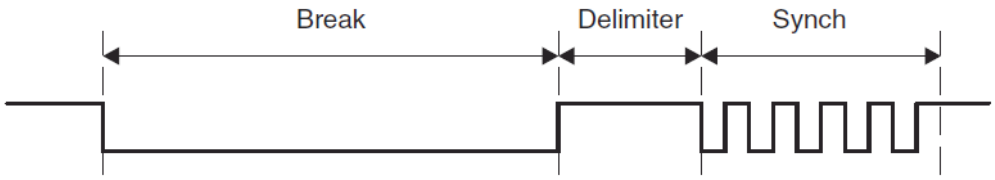
- Glitch supression
- Detecting following errors:
  - Framing – a low stop bit is detected – UCFE flag set
  - Parity – UCPE bit set
  - Break condition – all data, parity, and stop bits are low, and USCI is not in the automatic baud-rate mode – UCBRK bit set
  - Receive overrun – character in UCAxRXBUF overwritten – UCOE bit set

UCSI and Low-Power Modes

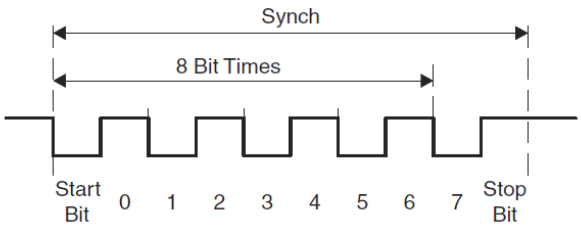
- If the MSP430 is in LPM, UCSI automatically activates the source clock when needed
- Glitch suppression prevents the accidental start (any low level on URXDx shorter than approximately 150 ns will be ignored)



Automatic Baud-Rate Detection

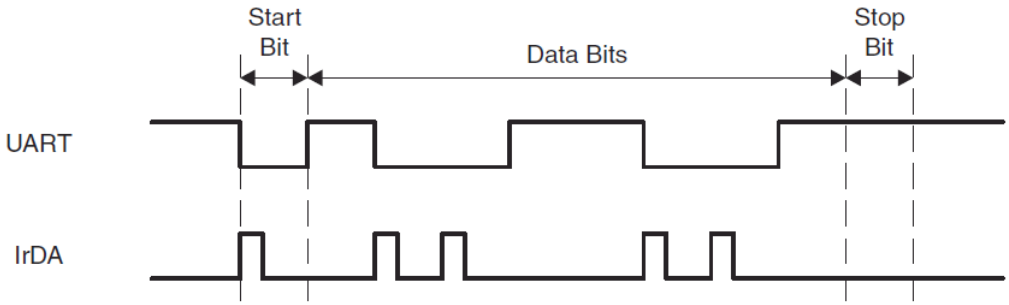


- Break must be more than 10 and less than 21 zeros
- Synch byte is 0x55



- When UCDORM = 1, characters are received but not transferred into the buffer, until a break/synch is detected. The following character is transferred to the buffer
  - To keep receiving data, the user must reset UCDORM
- For LIN conformance, the character format should be set to eight data bits, LSB first, no parity, and one stop bit.

IrDA



- The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART
- The standard requires 3/16 pulse duration
  - Use the BITCLK16 and set the length to 6 half-cycles
- For detection, MSP employs one analog deglitch filter and optionally, an additional programmable digital filter

USART Registers

- *Control and Status Registers (UART mode)*
  - USCI_Ax Control Word
  - USCI_Ax Baud Rate Control Word
  - USCI_Ax Modulation Control
  - USCI_Ax Status
  - USCI_Ax Receive Buffer
  - USCI_Ax Transmit Buffer
  - USCI_Ax Auto Baud Rate Control
  - USCI_Ax IrDA Control
  - USCI_Ax Interrupt Control
  - USCI_Ax Interrupt Vector
- One group for each USCI_A

USCI_Ax Control Register 0 (UCAxCTL0)

7	6	5	4	3	2	1	0
UCPEN	**UCPAR**	**UCMSB**	**UC7BIT**	**UCSPB**	**UCMODEx**		**UCSYNC=0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN Bit 7 Parity enable
- 0 Parity disabled
- 1 Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation.

UCPAR Bit 6 Parity select. UCPAR is not used when parity is disabled.
- 0 Odd parity
- 1 Even parity

UCMSB Bit 5 MSB first select. Controls the direction of the receive and transmit shift register.
- 0 LSB first
- 1 MSB first

UC7BIT Bit 4 Character length. Selects 7-bit or 8-bit character length.
- 0 8-bit data
- 1 7-bit data

UCSPB Bit 3 Stop bit select. Number of stop bits.
- 0 One stop bit
- 1 Two stop bits

UCMODEx Bits 2-1 USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.
- 00 UART mode
- 01 Idle-line multiprocessor mode
- 10 Address-bit multiprocessor mode
- 11 UART mode with automatic baud-rate detection

UCSYNC Bit 0 Synchronous mode enable
- 0 Asynchronous mode
- 1 Synchronous mode

USCI_Ax Control Register 1 (UCAxCTL1)

7	6	5	4	3	2	1	0
UCSSELx		**UCRXEIE**	**UCBRKIE**	**UCDORM**	**UCTXADDR**	**UCTXBRK**	**UCSWRST**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx Bits 7-6 USCI clock source select. These bits select the BRCLK source clock.
- 00 UCAxCLK (external USCI clock)
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCRXEIE Bit 5 Receive erroneous-character interrupt enable
- 0 Erroneous characters rejected and UCRXIFG is not set.
- 1 Erroneous characters received set UCRXIFG.

UCBRKIE Bit 4 Receive break character interrupt enable
- 0 Received break characters do not set UCRXIFG.
- 1 Received break characters set UCRXIFG.

UCDORM Bit 3 Dormant. Puts USCI into sleep mode.
- 0 Not dormant. All received characters set UCRXIFG.
- 1 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 Bit 2 Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode.
- 0 Next frame transmitted is data.
- 1 Next frame transmitted is an address.

UCTXBRK Bit 1 Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud-rate detection, 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise, 0h must be written into the transmit buffer.
- 0 Next frame transmitted is not a break.
- 1 Next frame transmitted is a break or a break/synch.

UCSWRST Bit 0 Software reset enable
- 0 Disabled. USCI reset released for operation.
- 1 Enabled. USCI logic held in reset state.

Baud rate and modulation registers

USCI_Ax Baud Rate Control Register 0 (UCAxBR0)

7	6	5	4	3	2	1	0
UCBRx - low byte							
rw	rw	rw	rw	rw	rw	rw	rw

USCI_Ax Baud Rate Control Register 1 (UCAxBR1)

7	6	5	4	3	2	1	0
UCBRx - high byte							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx Clock prescaler setting of the baud-rate generator. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value UCBRx.

USCI_Ax Modulation Control Register (UCAxMCTL)

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx Bits 7-4 First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 26-3 shows the modulation pattern.

UCBRSx Bits 3-1 Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 26-2 shows the modulation pattern.

UCOS16 Bit 0 Oversampling mode enabled

0 Disabled

1 Enabled

USCI_Ax Status Register (UCAxSTAT)

7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	UCPE	UCBRK	UCRXERR	UCADDR/ UCIDLE	UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

UCLISTEN Bit 7 Listen enable. The UCLISTEN bit selects loopback mode.

0 Disabled

1 Enabled. UCAxTXD is internally fed back to the receiver.

UCFE Bit 6 Framing error flag

0 No error

1 Character received with low stop bit

UCOE Bit 5 Overrun error flag. This bit is set when a character is transferred into UCAxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly.

0 No error

1 Overrun error occurred.

UCPE Bit 4 Parity error flag. When UCPEN = 0, UCPE is read as 0.

0 No error

1 Character received with parity error

UCBRK Bit 3 Break detect flag

0 No break condition

1 Break condition occurred.

UCRXERR Bit 2 Receive error flag. This bit indicates a character was received with error(s). When UCRXERR = 1, on or more error flags, UCFE, UCPE, or UCOE is also set. UCRXERR is cleared when UCAxRXBUF is read.

0 No receive errors detected

1 Receive error detected

UCADDR Bit 1 Address received in address-bit multiprocessor mode. UCADDR is cleared when UCAxRXBUF is read.

0 Received character is data.

1 Received character is an address.

UCIDLE Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAxRXBUF is read.

0 No idle line detected

1 Idle line detected

UCBUSY Bit 0 USCI busy. This bit indicates if a transmit or receive operation is in progress.

0 USCI inactive

1 USCI transmitting or receiving

Buffer Registers

USCI_Ax Receive Buffer Register (UCAxRXBUF)

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx Bits 7-0 The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset.

Reading resets the receive-interrupt flag

USCI_Ax Transmit Buffer Register (UCAxTXBUF)

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx Bits 7-0 The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UCAxTXD. Writing to the transmit data buffer clears UCTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset.

Transmitting sets the transmit-interrupt flag

IrDA Registers

USCI_Ax IrDA Transmit Control Register (UCAxIRTCTL)

7	6	5	4	3	2	1	0
UCIRTXPLx						UCIRTXCLK	UCIREN
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRTXPLx Bits 7-2 Transmit pulse length
Pulse length t_{PULSE} = (UCIRTXPLx + 1) / (2 × $f_{IRTXCLK}$)

UCIRTXCLK Bit 1 IrDA transmit pulse clock select
0 BRCLK
1 BITCLK16 when UCOS16 = 1. Otherwise, BRCLK.

UCIREN Bit 0 IrDA encoder/decoder enable
0 IrDA encoder/decoder disabled
1 IrDA encoder/decoder enabled

USCI_Ax IrDA Receive Control Register (UCAxIRRCTL)

7	6	5	4	3	2	1	0
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRRXFLx Bits 7-2 Receive filter length. The minimum pulse length for receive is given by:
t_{MIN} = (UCIRRXFLx + 4) / (2 × f_{BRCLK})

UCIRRXPL Bit 1 IrDA receive input UCAxRXD polarity
0 IrDA transceiver delivers a high pulse when a light pulse is seen.
1 IrDA transceiver delivers a low pulse when a light pulse is seen.

UCIRRXFE Bit 0 IrDA receive filter enabled
0 Receive filter disabled
1 Receive filter enabled

USCI_Ax Auto Baud Rate Control Register (UCAxABCTL)

7	6	5	4	3	2	1	0
Reserved		UCDELIMx		UCSTOE	UCBTOE	Reserved	UCABDEN
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

Reserved	Bits 7-6	Reserved
UCDELIMx	Bits 5-4	Break/synch delimiter length
		00 1 bit time
		01 2 bit times
		10 3 bit times
		11 4 bit times
UCSTOE	Bit 3	Synch field time out error
		0 No error
		1 Length of synch field exceeded measurable time.
UCBTOE	Bit 2	Break time out error
		0 No error
		1 Length of break field exceeded 22 bit times.
Reserved	Bit 1	Reserved
UCABDEN	Bit 0	Automatic baud-rate detect enable
		0 Baud-rate detection disabled. Length of break and synch field is not measured.
		1 Baud-rate detection enabled. Length of break and synch field is measured and baud-rate settings are changed accordingly.

USCI_Ax Interrupt Registers

USCI_Ax Interrupt Enable Register (UCAxIE)

7	6	5	4	3	2	1	0
Reserved						UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved	Bits 7-2	Reserved	**UCRXIE**	Bit 0	Receive interrupt enable
UCTXIE	Bit 1	Transmit interrupt enable			0 Interrupt disabled
		0 Interrupt disabled			1 Interrupt enabled
		1 Interrupt enabled			

USCI_Ax Interrupt Flag Register (UCAxIFG)

7	6	5	4	3	2	1	0
Reserved						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved	Bits 7-2	Reserved
UCTXIFG	Bit 1	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty.
		0 No interrupt pending
		1 Interrupt pending
UCRXIFG	Bit 0	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character.
		0 No interrupt pending
		1 Interrupt pending

USCI_Ax Interrupt Registers

USCI_Ax Interrupt Vector Register (UCAxIV)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**UCIVx**		**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCAxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending		
002h	Data received	UCRXIFG	Highest
004h	Transmit buffer empty	UCTXIFG	Lowest

USART Initialization

- The required USART initialization/re-configuration process is:
  - Set UCSWRST (Software reset bit in UCAxCTL1 control registers)
    - This is done automatically after PUC
  - Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1)
  - Configure ports
  - Clear UCSWRST via software
  - Enable interrupts (optional) via the UCRXIE and/or UCTXIE
- Failure to follow this process may result in unpredictable USART behavior.

UART example – 5xx

```
#include "msp430.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P3SEL = BIT4 + BIT5;                      // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 9;                              // 1MHz 115200 (see User's Guide)
  UCA0BR1 = 0;                              // 1MHz 115200
  UCA0MCTL |= UCBRS_1 + UCBRF_0;            // Modulation UCBRSx=1, UCBRFx=0
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

  _EINT();                                  // Enter LPM0, interrupts enabled
  LPM0;                                     // For debugger
}

// Echo back RXed character, confirm TX buffer is ready first
void USCI_A0_isr(void) __interrupt[USCI_A0_VECTOR]
{
  switch(UCA0IV)
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    while (!(UCA0IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;                  // TX -> RXed character
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}
```