

# Lesson 9


# Review: Function

- Functions are sequences of code that can be reused at any point in a program. Equivalent to Subroutines in assembler
  - Helps manage program complexity
  - Easier to design and debug
  - Functions can often be reused instead of starting over
  - Can use of “libraries” of functions developed by 3rd parties, instead of designing your own
- A function is “called” by another program to perform a task
  - The **function may** return a result to the caller
  - One or more arguments may be passed to the function/procedure
  - You must define (declare) before using the function

# Review: Function definition

Type of value to be  
returned to the caller\*

Parameters passed  
by the caller



```
int math_func (int n)
{
    int j;           //local variable
    j = n + 5;       //function body
    return(j);       //return the result
}
```

\* If no return value, specify “void”

## Review:

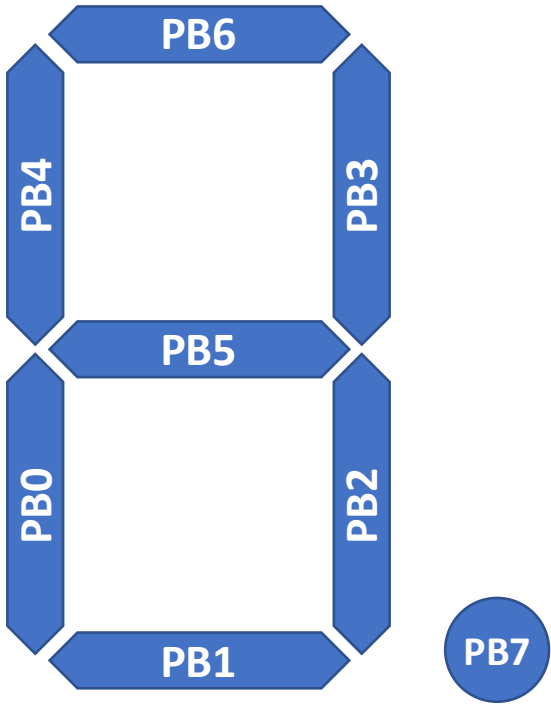
- Example 1:

Write a C code to read the switches value on PORTC and display it on the seven-segment using predesigned library file:

1. Configure PORTC as input and PORTB as output
2. Download these files from itslearning and add to your project :  
SevenSegment.c, SevenSegment.h
3. Modify SevenSegment.c file for all values
4. `#include "SevenSegment.h"` in top of main.c file
5. Use `SevenSegDisp(PINC);` function to display values

# Seven Segment

Number	Binary Value	Negative Binary
0	01011111	0b10100000
1	00001100	0b11110011
2		
3		
4		
5		
6		
7		
8		
9		



# Fuse Bits

## ATMEGA32A fuse bits

- There are few parameters in the AVR chip must be configured before it can be used for external environment.
- These parameters are set with the use of Fuse Bits.
- The fuse bit determines the behavior of the chip, what is the speed and voltage it runs at etc.
- You don't have to set the fuse bits every time you are using the controller till the time you want to use it under the same configuration.
- Fuse bits need to be changed only in case you want to change the initial configuration of the controller.

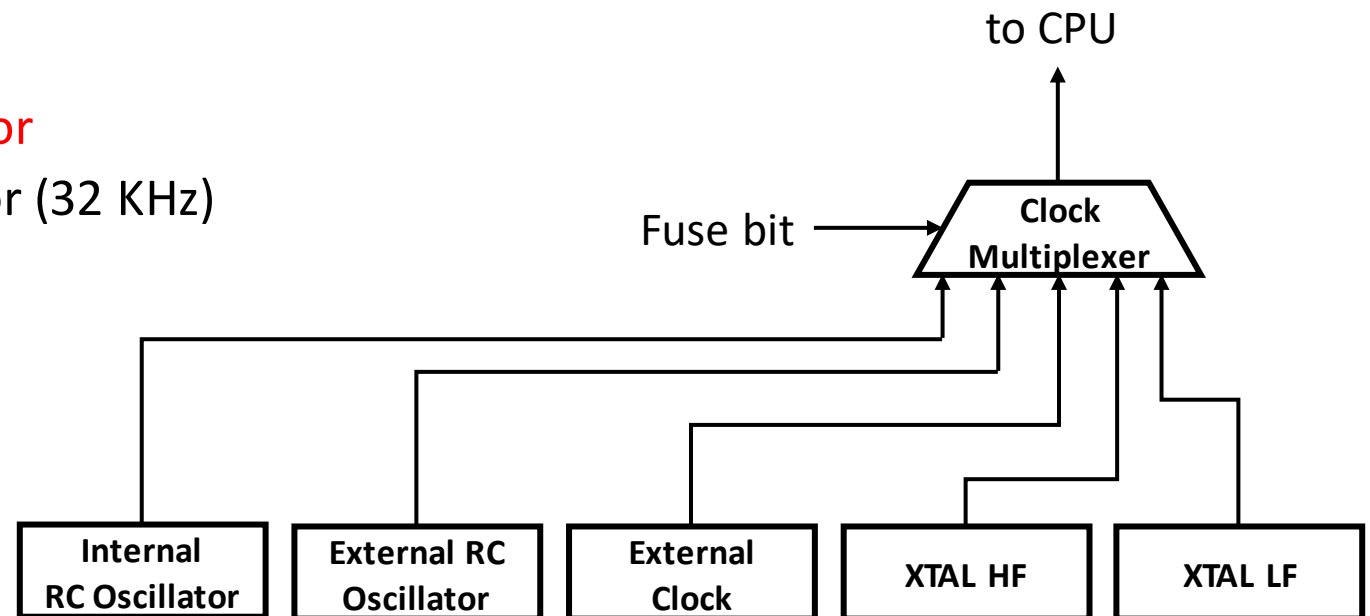
## ATMEGA32A fuse bits

- ATmega32A fuse bits are used for basic configuration of ATmega32A processor such as:
  - Start-up time
  - Clock source & frequency
  - Brown-out level
- The AVR microcontroller consists of sixteen fuse bits (two bytes) which are classified as low fuse and high fuse bytes.
- **Note! If fuse bits are incorrectly programmed, the microprocessor can become "dead."**



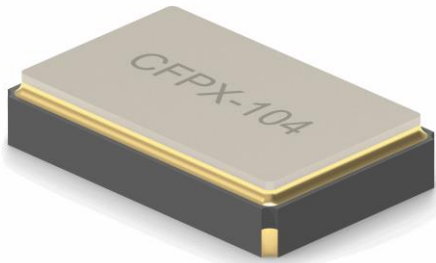
# AVR Clock Source

- There are two different clock sources:
  - Internal Calibrated RC Oscillator
  - External Sources:
    - RC Oscillator
    - External Clock
    - High Frequency XTAL Oscillator
    - Low Frequency XTAL Oscillator (32 KHz)

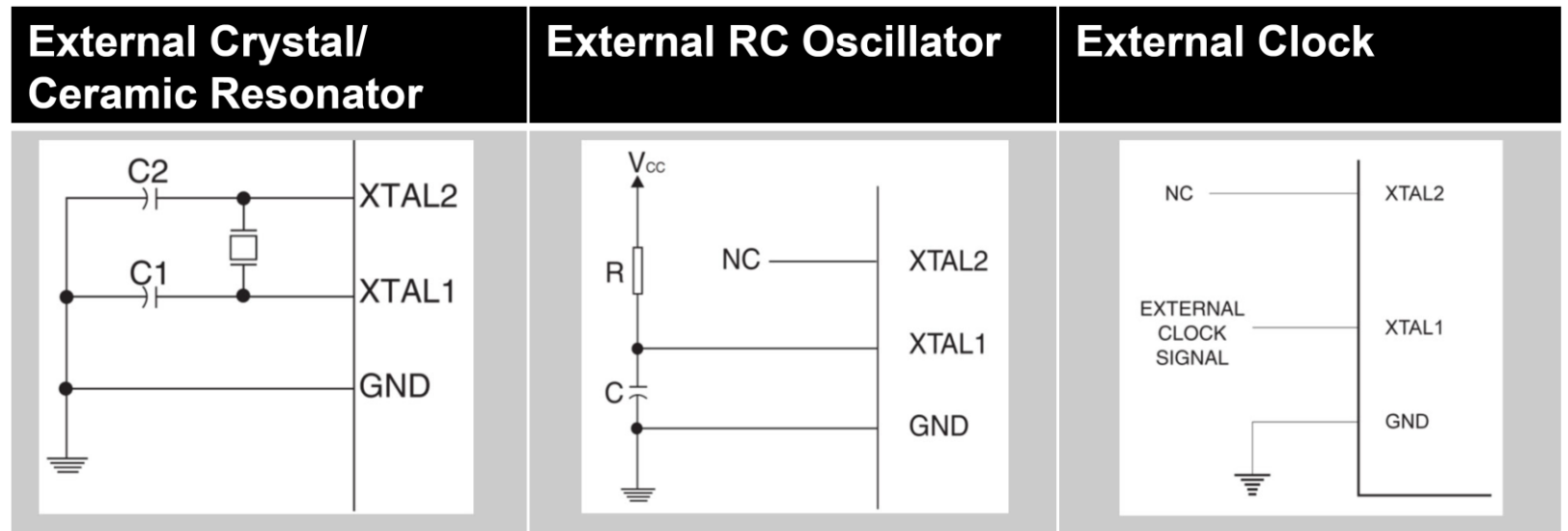


# AVR Clock Source connections

- Frequency of external crystals are extremely stable with respect to temperature changes: 0.003%. (30ppm)
- Internal Oscillator have a precision of about 3%. (30000ppm)
- [XTAL](#)



XTAL: CFPX-104  
Frequency tolerance: 20ppm  
Frequency stability: 30ppm

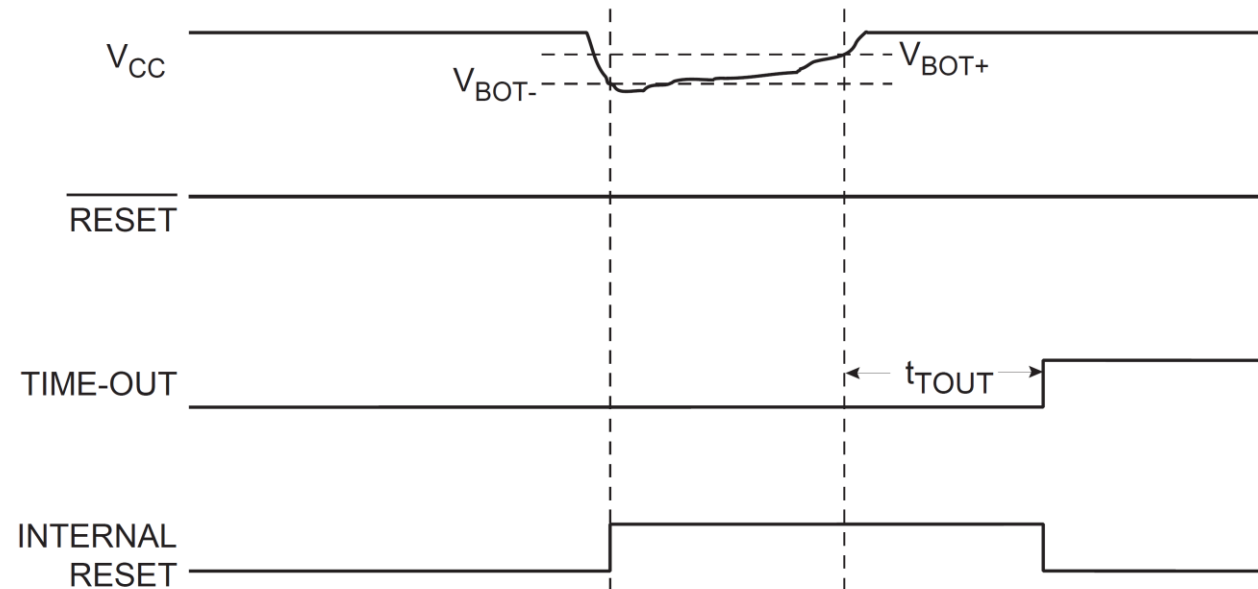


## Brown out detection

- A brownout for a chip means that the power supply voltage is too low for the MCU to run reliably at the speed of the clock.
- For example, the ATMEGA32 can run as fast as 16MHz but only if the power voltage is between 4.5V and 5.5V.
- If the voltage is lower than that, it may behave erratically, erasing or overwriting the RAM and EEPROM.
- It may also start running random piece of the code.
- To keep it from doing that, we can set the brownout voltage to 4V, then if the voltage drops, the chip will turn off until the voltage returns.
- It will then reset and start over.

# Brown out detection

- The trigger level for the BOD can be selected by the fuse BODLEVEL to be 2.7V or 4.0V
- The trigger level has a hysteresis to ensure spike free Brown-out Detection.
- The hysteresis on the detection level should be interpreted as:
  - $V_{BOT+} = V_{BOT} + 25 \text{ mV}$
  - $V_{BOT-} = V_{BOT} - 25 \text{ mV}$
- When VCC decreases to a value below the trigger level ( $V_{BOT-}$ ) and stay for more than 2  $\mu\text{s}$ , the Brown-out Reset is activated.
- When VCC increases above the trigger level ( $V_{BOT+}$ ), the delay counter starts the MCU after the Time-out period  $t_{TOUT}$  has expired.



# ATMEGA32A fuse bits

- The following slides will provide an overview of some important fuse bit settings
- Use the following [Link1](#), [Link2](#) for easy Fuse bit setting.

Table 27-3. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN <sup>(4)</sup>	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN <sup>(5)</sup>	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN <sup>(1)</sup>	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT <sup>(2)</sup>	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 26-6 for details)	0 (programmed) <sup>(3)</sup>
BOOTSZ0	1	Select Boot Size (see Table 26-6 for details)	0 (programmed) <sup>(3)</sup>
BOTRST	0	Select reset vector	1 (unprogrammed)

Table 27-4. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed) <sup>(1)</sup>
SUT0	4	Select start-up time	0 (programmed) <sup>(1)</sup>
CKSEL3	3	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL2	2	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL1	1	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL0	0	Select Clock source	1 (unprogrammed) <sup>(2)</sup>

## ATMEGA32A fuse bits

- Use the following [Link](#) for easy Fuse bit setting.
- Run Command Prompt
- Run:

```
avrdude -c usbasp -p m32 -U lfuse:w:0xf1:m -U hfuse:w:0x99:m
```

- **Red line** has been extracted from [Link](#)

## Example 2:

- Write a program in C to toggle LEDs every 1 second (using *\_delay\_ms()*)
  - A) Set ATMEGA32 to run with internal 1 MHz
  - B) Set ATMEGA32 to run with external High frequency XTAL
  - C) `#define F_CPU 8000000`

# Digital Data Communication

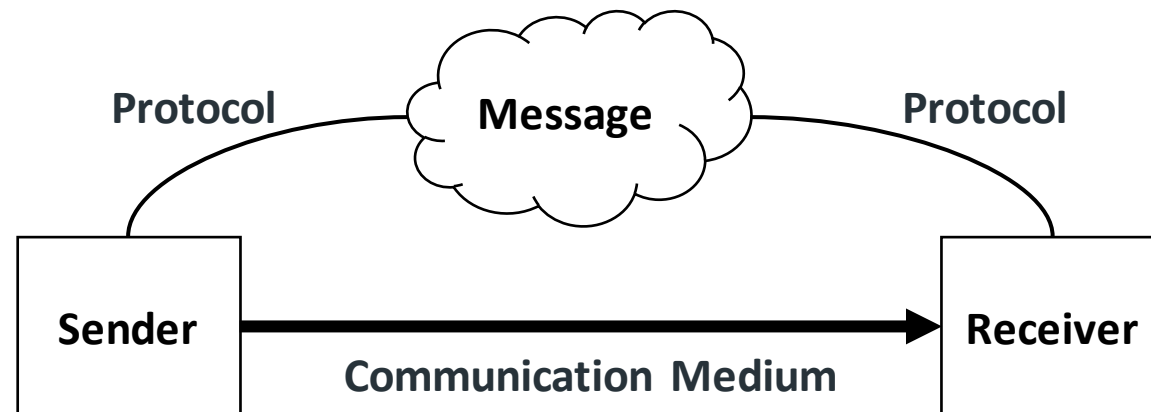


# Overview

- Serial communication medium
  - Synchronous / asynchronous communication
  - Simplex, half duplex or full duplex
- Universal Asynchronous Communication
  - UART - (Universal Asynchronous Receiver / Transmitter).
  - Data framing
  - Serial port programming

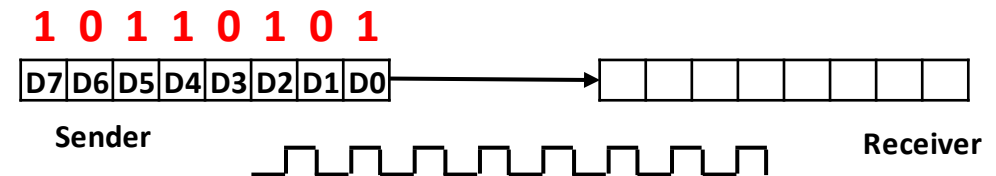
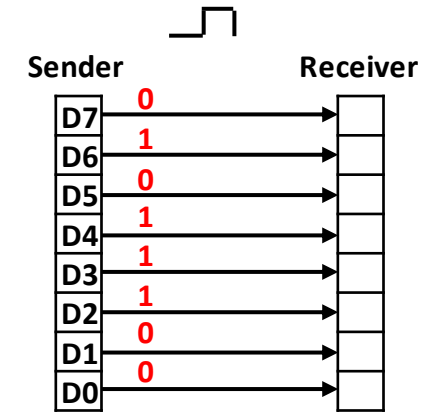
# Digital Data Communication

- Digital data communication is the transfer and reception of data in the form of a digital bitstream (0 and 1) and it is made up of:
  - **Message:** A piece of information that is to be transmitted
  - **Sender:** It is simply a device that sends data messages.
  - **Receiver:** It is a device that receives messages.
  - **Communication Medium:** Physical connection between two devices
  - **Protocol:** Set of rules to send and receive data



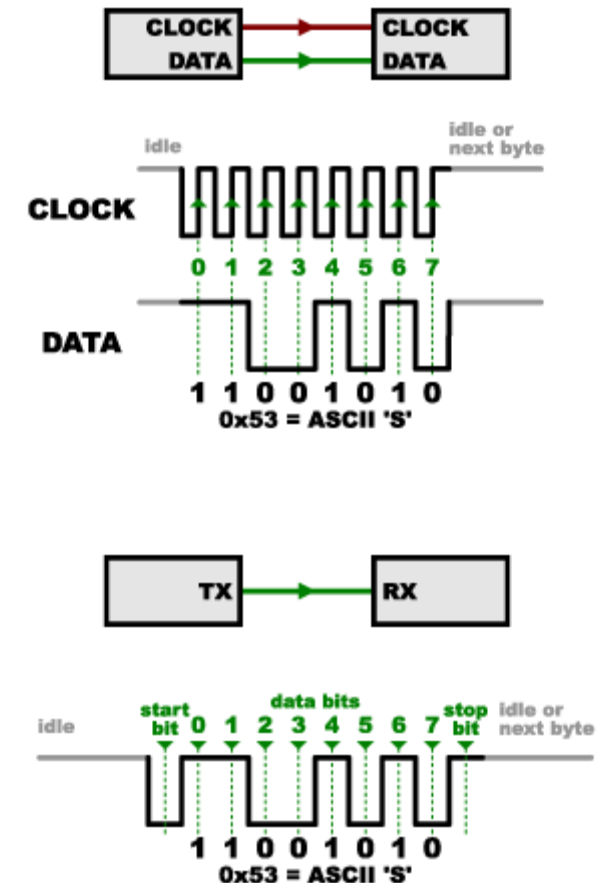
# Serial and Parallel Communication

- Data is transferred using two methods:
  - Parallel:
    - All the bits are transferred simultaneously
    - High speed
    - Need lot of wires: Impractical for long distances
    - Example: Old printers
  - Serial:
    - Bits are sent sequentially
    - Theoretically lower communication speed
    - Use common wire for all bits
    - Example: USB



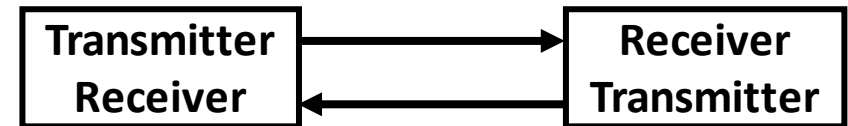
# Serial Data Transfer

- Two main methods in Serial Data transfer:
  - Synchronous
    - There is a common clock line between the Transmitter and Receiver
    - Data are captured at clock transition
  - Asynchronous
    - No common clock line between Transmitter and Receiver
    - Data are captured using internal clock based on predefined communication speed



# Serial Communication

- Two modes of Serial Communication:
  - Duplex: Data can be transmitted and Received
    - Full Duplex: Data can be sent and received simultaneously
      - Needs two wires for data



- Half Duplex: Data can **not** be sent and received simultaneously
    - Needs one **shared** wire for data



- Simplex
  - Data is only transmitted in one way



# UART Protocol

## UART: Universal Asynchronous Receiver and Transmitter

- UART is one the most used device-to-device communication protocols:
  - Between Microcontroller and Sensor/Microcontroller
- It is Full-Duplex Asynchronous Serial Protocol:
  - There is no clock between the transmitter and receiver
  - Data transmission is two-ways, same time



# UART Protocol

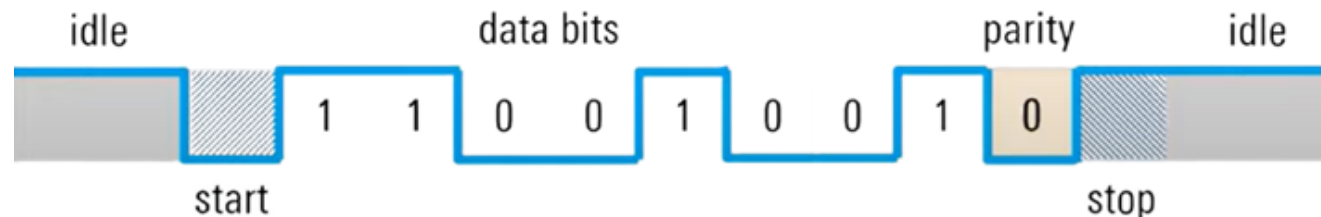
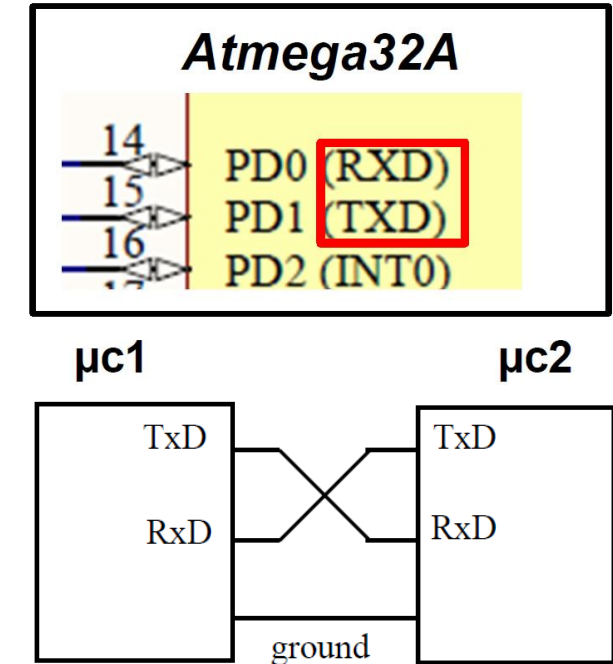
- In a data transfer, the data received at the receiving end is all 0s (Low Voltage) and 1s (High Voltage)
- For example, if the line is always in high logic level (logic **ONE**), how to distinguish the data transmission with idle mode without clock?
- **Protocol:** The sender and receiver must agree on how the data is packed, how many bits constitute a character, and when data begins and ends:
  - Transmit at the same (known) speed
    - There is no common clock line
  - Use the same frame structure



# UART frame format

- UART can transfer and receive data serially with different speed
- UART frame consists of:
  - Start and Stop bits
  - Data bits (5 to 9 usually 8 bits)
  - Parity bit (optional ) for error detection
- In the idle state (no data is transmitted), the line is held high
- Start bit: The first bit of a UART transmission (logic **0**).
- Stop bit: The last bit of a UART transmission (logic **1**).

Common UART Baud rates	
	4800
	9600
	19200
	57600
	115200
	1000000

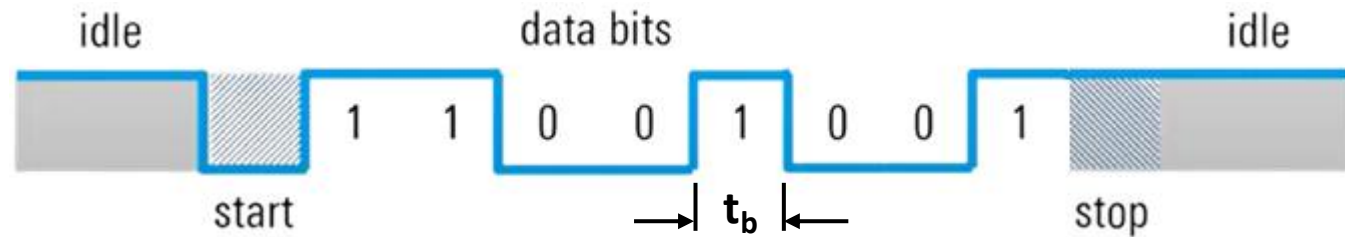


# UART frame format

- Data are typically sent with least significant bit (LSB) first

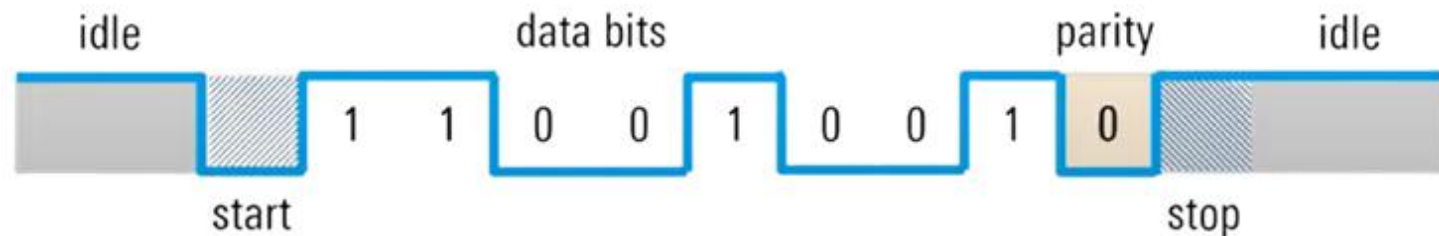
- Example:

- Data: 10010011
- Baud rate: 9600
- $t_b = 1/9600 = 104 \mu s$



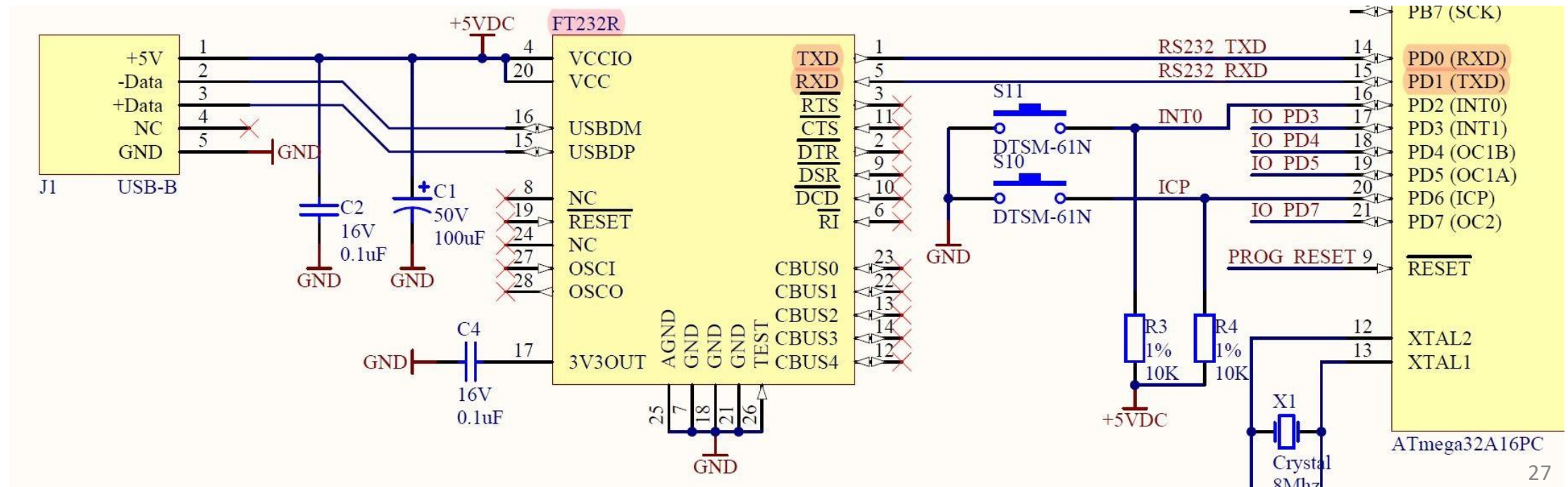
- Parity: used for error detection

- **Even Parity:** Number of 1s must be even
- Odd Parity: Number of 1s must be odd



# UART connection in AVR board

- FT232:
  - Single chip USB to UART interface/converter
  - Provides Virtual COM Port for PC



# UART Registers

- In the AVR microcontroller five registers are associated with the UART:
  - UBRR (UART Baud Rate Register)
  - UCSRA,UCSRB, *UCSRC* (UART Control Status Register)
  - UDR (UART Data Register)

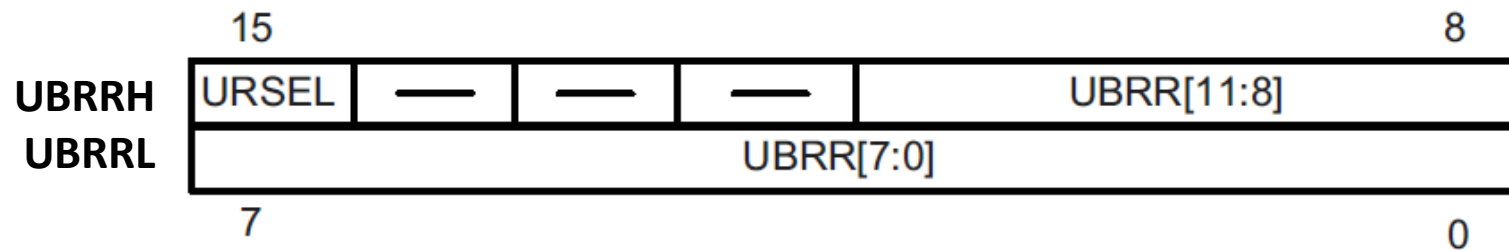
# UBRR register and baud rate in the AVR

- The AVR transfers and receives data serially at many different baud rates.
- There are some standard baud rates for UART communication.
- The baud rate in the AVR is programmable using **UBRR** register
- **BaudRate** =  $F_{osc} / (16(X + 1))$
- **X** is the value we load into the **UBRR** register
- **X** =  $F_{osc} / (16 \times \text{BaudRate}) - 1$
- Example:
  - Desired Baud-Rate=4800,  $F_{osc} = 1\text{MHz}$ ,  $X=12.02$
  - $X = 12 \rightarrow \text{Baud Rate}=4808$

Data Rate (bps)
1,200
2,400
4,800
9,600
19,200
38,400
57,600
115,200

# UBRR register and baud rate in the AVR

- UBRR register is 16-bit register



- It is accessible by **UBRRH** and **UBRRL**
- Only 12 bits of are used to set the USART baud rate
- $UBBR = F_{osc} / 16(\text{BaudRate}) - 1$

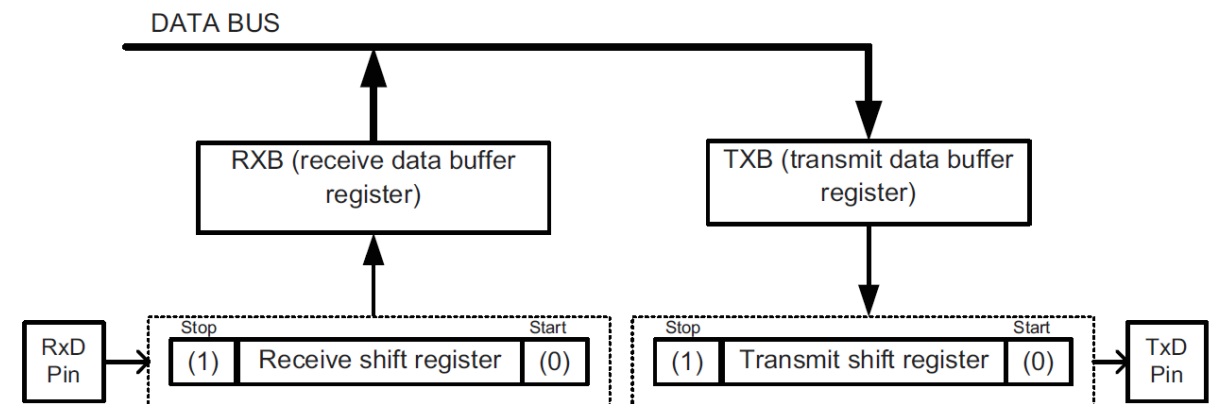
Data Rate (bps)
1,200
2,400
4,800
9,600
19,200
38,400
57,600
115,200

## UART: UDR

- Two registers as Transmit Register and Receive Register, they have same register address and name.
- When you write data to **UDR** register is written to **UDR (Write)**
- When you read data from **UDR** register is read from **UDR (Read)**

UDR – USART I/O Data Register

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



# UCSRA: UART Control and Status Register A

UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **B7:** RXC: UART Receive Complete Flag
- **B6:** TXC: UART Transmit Complete Flag
- **B5:** UDRE: UART Data Register Empty
- **B4:** FE: Frame Error
- **B3:** DOR: Data OverRun
- **B2:** PE: Parity Error
- **B1:** U2X: Double the UART Transmission Speed
- **B0:** MPCM: Multi-processor Communication Mode



# UCSRB: UART Control and Status Register B

UCSRB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **B7:** RXCIE: RX Complete Interrupt Enable
- **B6:** TXCIE: TX Complete Interrupt Enable
- **B5:** UDRIE: UART Data Register Empty Interrupt Enable
- **B4:** RXEN: Receiver Enable
- **B3:** TXEN: Transmitter Enable
- **B2:** UCSZ2: Character Size (1: 9 bit, 0: 8 bit)
- **B1:** RXB8: Receive Data Bit 8
- **B0:** TXB8: Transmit Data Bit 8

# UCSRC: UART Control and Status Register C

UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

- **B7:** URSEL: Register Select
- **B6:** UMSEL: USART Mode Select
- **B5,4:** UPM1:0: Parity Mode (Disabled, Enabled, 1, 2)
- **B3:** USBS: Stop Bit Select (1, 2)
- **B2,1:** UCSZ1:0: Character Size
- **B0:** UCPOL: Clock Polarity

Default: 8-bit data, rising edge, 1 Stop bit, Parity disabled

## UART: How to program UART to send and receive data in pooling mode

- Enable UART TX and RX in UCSRB
  - TXEN: Set Transmitter Enable bit
  - RXEN: Set Receiver Enable bit
- Calculate and configure Baud Rate: UBRR Registers
  - $UBRR = F_{osc} / 16(\text{BaudRate}) - 1$
- To send data:
  - Wait for UDRE: USART Data Register Empty Flag to send new data
  - Write to UDR Register
- To receive data:
  - Wait for RXC: UART Receive Complete Flag to receive new data
  - Read from UDR Register

## Example 3

- A) Write a C code to send numbers from 0 to 255 every 1 second, by UART protocol to your PC (UART Speed = 9600)
- B) Write a C code to Listen to UART port to receive data from the user  
When data is received, display the received data using LEDs on PORTB

## Example 3, A

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    UBRRH = ???; /* Config data rate */
    UBRRL = ???; /* Config data rate */
    UCSRB = ???; /* Enable transmitter */

    while (1)
    {
        for(int m=0;m<255;m++)
        {
            _delay_ms(1000);
            while ( !( UCSRA & (1<<UDRE)) ); /* Wait if UART Output Data Register is not empty */
            UDR = ???; /* send data out */
        }
    }
}
```

## Example 3, B

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    UBRRH = ???; /* Config data rate */
    UBRRL = ???; /* Config data rate */
    UCSRB = ???; /* Enable Receiver */
    DDRB = ???;

    while (1)
    {
        while (!(UCSRA & (1<<RXC))); /* Wait here if UART input Data Register is empty*/
        PORTB=~UDR;
    }
}
```

# ASCII Coding

# ASCII Coding

- ASCII stands for American Standard Code for Information Interchange
- ASCII code is the numerical representation of a character such as 'a' or '@' or '1' or an action such as new line
- It is mostly used to send and receive text data
- Every character or action on the keyboard has an 8 bit (1 Byte) ASCII code between 0 and 255



# ASCII Code table

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharhms.com

# Defining an ASCII character in C programming

- To define a characters in C language we can define:

```
uint8_t data='a';
```

- Same as writing:

```
uint8_t data=97;
```

dec	hex	oct	char
96	60	140	`
97	61	141	a
98	62	142	b
99	63	143	c
100	64	144	d
101	65	145	e

- To define a set of characters in C language, array of data can be used

- ```
uint8_t data[12]="AVR ATMEGA32";
```

- ```
uint8_t data[]="AVR ATMEGA32";
```

## Example 4

- Write a C code to send character from 'a' every 1 second using UART Protocol

## Example 4

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    uint8_t data=???;
    UBRRH = ???;      /* Config data rate */
    UBRRL = ???;      /* Config data rate */
    UCSRB = ???;      /* Enable Transmitter */

    while (1)
    {
        _delay_ms(1000);
        while ( ??? ); /* Wait if UART output Data Register is not empty*/
        UDR=data;
    }
}
```

# Assignment

1. Write a C code to send your name every 1 second using UART Protocol
2. Write a C code to wait for a command from the UART port. If the received command by UART is equal to 'n', then send your name by UART