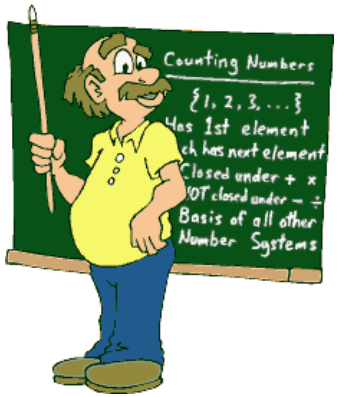
IECA

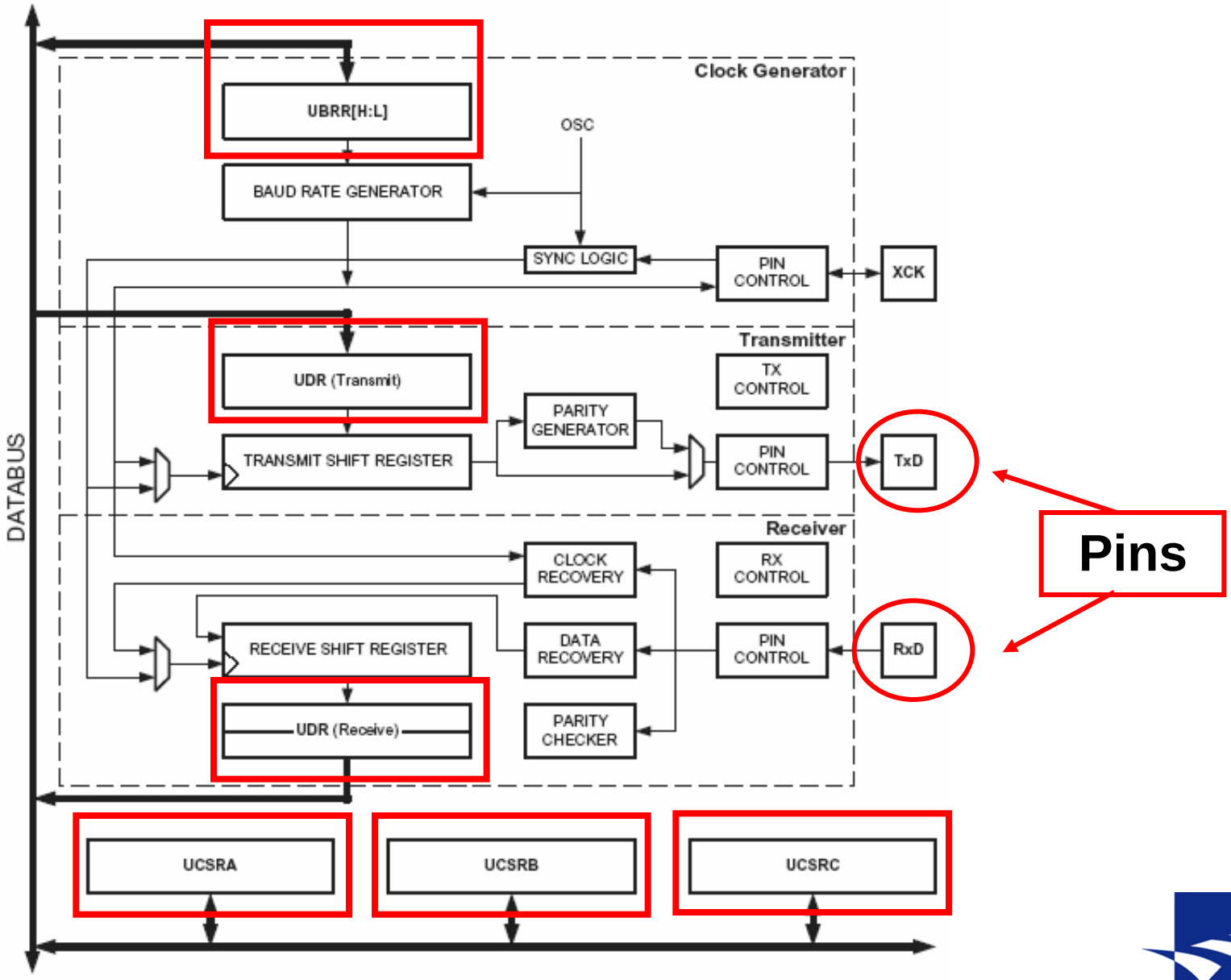
Embedded Computer Architecture

Lesson 18: UART interrupts etc.

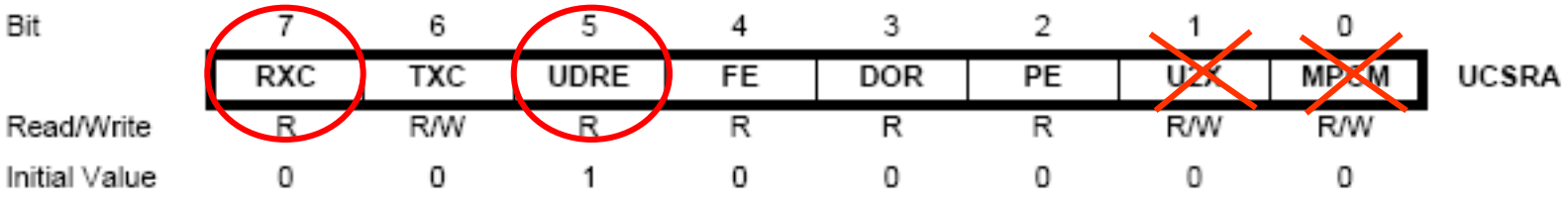


Version: 16-11-2015, Henning Hargaard

Mega32 USART



UCSRA: Control and Status Register A



RXC : New character received (”available in UDR”).

TXC : ”Transmitter ready” (ready for sending a new character and transmitter register empty).

UDRE : Ready to send a new character (”free to write UDR”).

FE : ”Framing Error” (received character has errors in stop bit).

DOR : ”Data overrun” (character received, before the prior received character has been read by SW).

PE : ”Parity Error” (received character has a parity error).

Test (”socrative.com”, Room = MSYS)

- What method is the right one for waiting for a new character received ?
- A:

```
if (UCSRA & 0b10000000) != 0)
return UDR;
```

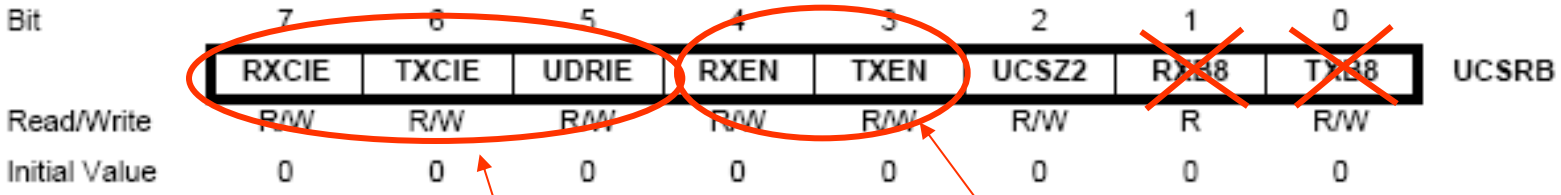
- B:

```
while (UCSRA & 0b10000000 == 0)
{ }
return UDR;
```

- C:

```
while (UCSRA & 0b10000000 != 0)
{ }
return UDR;
```

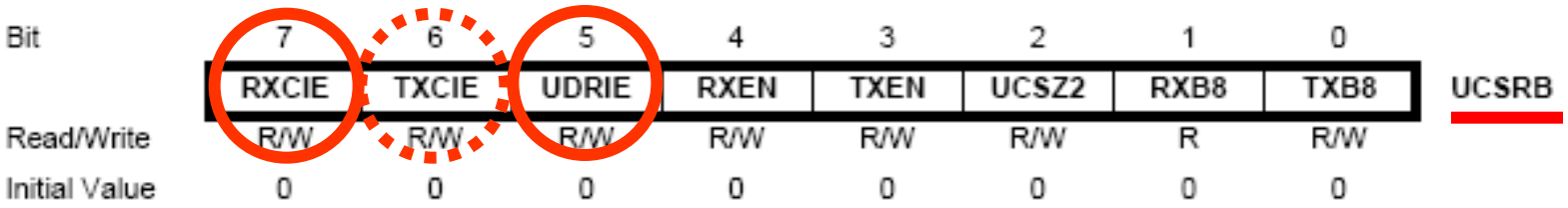
UCSRB: Control and Status Register B



Used by interrupt controlled USART

RXEN = 1: ”RX Enable” (switch on the receiver).
TXEN = 1: ”TX Enable” (switch on the transmitter).

USART interrupt enables



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	

RXCIE = 1: USART Rx Interrupt Enable.

Interrupt each time a new character is received.
The character will be ready in UDR, when interrupted.

(TXCIE = 1: USART Tx Interrupt Enable)

Interrupt each time the transmit register becomes empty and UDR is ready to send a new character.
Means that you are allowed to write the next character (to UDR).

UDRIE = 1: USART Data Register Empty Interrupt Enable.

Interrupt each time the UDR is ready for a new character (to be transmitted).

USART interrupt vectors

Table 10-1: Interrupt Vector Table for the Mega32

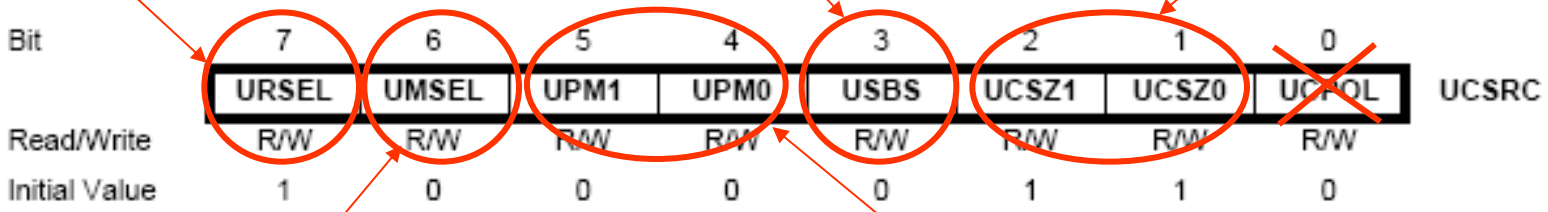
Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

UCSRC: Control and Status Register C

UCSZ1 and UCSZ0 : Number of data bits (see next page).

USBS : 0 = 1 stop bit. 1 = 2 stop bits.

Must be 1



UMSEL: 0 = Asynchronous mode.

UPM1 and UPM0: Selecting parity:

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Test (”socrative.com”, Room = MSYS)

- Assume we use 8 data bits and EVEN parity. What bit stream has parity error ?

A:

Data = 11101001 Paritetsbit = 1

B:

Data = 10001011 Paritetsbit = 0

C:

Data = 10001011 Paritetsbit = 1

Number of Data Bits

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	

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	

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

UBRRH + UBRRL :Baud Rate Registers

Must be 0 !

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Test (”socrative.com”, Room = MSYS)

- CPU clock frequency = 3,6864 MHz
  UBRRH = 2
  UBRRL = 255
  What is the BAUD rate ?

  A:
  300 bit/s
  B:
  1200 bit/s
  C:
  9600 bit/s
- D:
  115200 bit/s

LAB11: UART driver

```
/*****************************************
* "uart.h":                              *
* Header file for Mega32 UART driver.    *
* Henning Hargaard, 1/11 2011            *
*****************************************/
void InitUART(unsigned long BaudRate, unsigned char DataBit);
unsigned char CharReady();
char ReadChar();
void SendChar(char Ch);
void SendString(char* String);
void SendInteger(int Number);
/*****************************************/
```

The driver header file

InitUART()

void InitUART(unsigned long BaudRate, unsigned char DataBit)
Must initiate the UARTs to the desired baud rate (110 - 115200) and the desired number of data bits (5 - 8).
If the parameter BaudRate is less than 110 or greater than 115200, there must be no initialization of the UART.
If the parameter DataBit is less than 5 or greater than 8, there must be no initialization of the UART.

We assume that Mega32's clock frequency is 3.6864 MHz.
The value to be written to UBRRH and UBRRL, must in the function be calculated on the basis of the "BaudRate" parameter and Mega32's clock frequency (note that rounding may occur).

In addition, the UART should be initialized to:

- No parity.
- Asynchronous mode.
- Both the RX and TX enabled.
- 1 stop bit.
- All interrupts disabled.

Solution: InitUART()

```
void InitUART(unsigned long BaudRate, unsigned char DataBit)
{
unsigned int TempUBRR;

  if ((BaudRate >= 110) && (BaudRate <= 115200) && (DataBit >=5) && (DataBit <= 8))
  {
    // "Normal" clock, no multiprocessor mode (= default)
    UCSRA = 0b00100000;
    // No interrupts enabled
    // Receiver enabled
    // Transmitter enabled
    // No 9 bit operation
    UCSRB = 0b00011000;
    // Asynchronous operation, 1 stop bit, no parity
    // Bit7 always has to be 1
    // Bit 2 and bit 1 controls the number of databits
    UCSRC = 0b10000000 | (DataBit-5)<<1;
    // Set Baud Rate according to the parameter BaudRate:
    // Select Baud Rate (first store "UBRRH--UBRRL" in local 16-bit variable,
    //                   then write the two 8-bit registers separately):
    TempUBRR = XTAL/(16*BaudRate) - 1;
    // Write upper part of UBRR
    UBRRH = TempUBRR >> 8;
    // Write lower part of UBRR
    UBRRL = TempUBRR;
  }
}
```

Character functions

unsigned char CharReady( )
Checks the UART for a received character.
If a character is received, it returns a value different from 0 (= TRUE).
If no character is received, it returns the value 0 (= FALSE).
The function should **not** await receipt of a character, but simply return information on whether a character is received.

char ReadChar( )
Returns a received character from the UART's receive register (UDR).
The function **must first wait for a character to be received** (bit RXC the register UCSRA).

void SendChar(char Ch)
Sends one character via the UART. The character is the parameter.
Before the character is written to data register (UDR), the function must await "UART data register empty" (bit UDRE in register UCSRA).

CharReady(), ReadChar() and SendChar()

```
unsigned char CharReady()
{
   return UCSRA & (1<<7);
}

char ReadChar()
{
  // Wait for new character received
  while ( (UCSRA & (1<<7)) == 0 )
  {}
  // Then return it
  return UDR;
}

void SendChar(char Ch)
{
  // Wait for transmitter register empty (ready for new character)
  while ( (UCSRA & (1<<5)) == 0 )
  {}
  // Then send the character
  UDR = Ch;
}
```

SendString()

void SendString(char* String)
The function parameter is a pointer to the string that we want to send.
The pointer always points to the first character in the string, which is also assumed zero-terminated.
The user has a priori (i.e. before this function is called) created and stored the string in memory.

The following shows by pseudo code how the function can be implemented:

```
while ("What the pointer points at" is not 0)
{
   SendChar("What the pointer points at");
   Move the pointer one place forward;
}
```

SendString()

```
void SendString(char* String)
{
  // Repeat untill zero-termination
  while (*String != 0)
  {
    // Send the character pointed to by "String"
    SendChar(*String);
    // Advance the pointer one step
    String++;
  }
}
```

SendInteger()

void SendInteger(int Number)
This function should send the value of the integer "Number" received as the parameter.

If, for example, the function is called as follows:

SendInteger (147);

the following characters is send by the UART: '1 ', '4' and '7 '.

Hint:
First, create a local array of "appropriate" size.
Then use the standard function **itoa ( )** to convert the "Number" to a string stored in the array.
Remember # **include <stdlib.h>.**
itoa (number, array, 10) stores the string corresponding to "Number" in "array" (and zero-terminates it).
Then use the function **SendString( )** to send the string.

Text strings

	'V'	'i'	'g'	'g'	'o'	0
Index:	**0**	**1**	**2**	**3**	**4**	**5**

- A text string (string) is a char array, containing text.
  Each element hence contains one character.
- The last element must be a NULL-terminator (having the value 0). It defines the end of the string.
- In the compiler we can write a string directly by using qoutes: "…".
  The compiler automatically appends the NULL-terminator !

```
char str[6] = "Viggo";
```

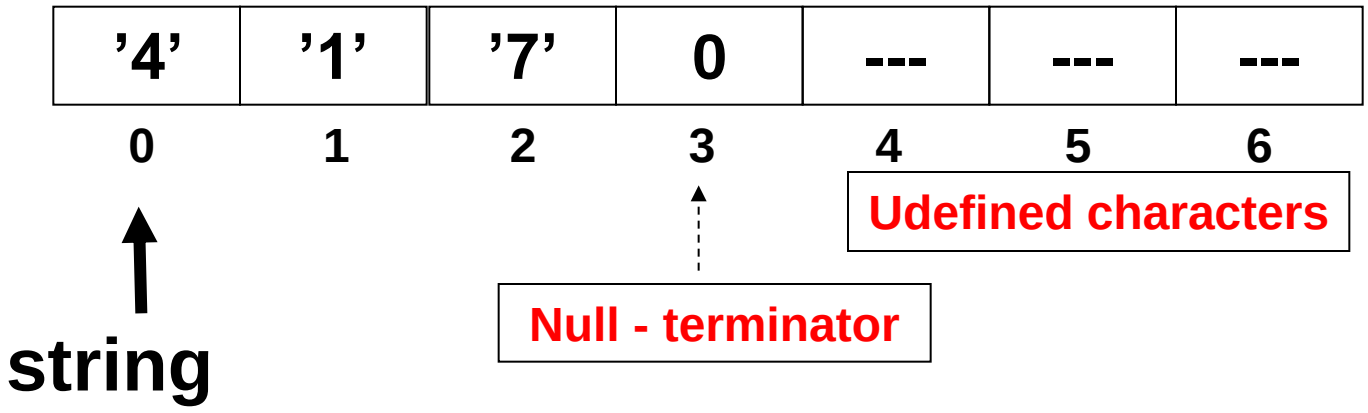
itoa(int n, char* str, int radix)

Having included <stdlib.h>, you have access to a wide range of functions, including itoa( ) being one of the most important.

itoa(int n, char* str, int radix) will format the value of n as a 0-terminated text string in an array, pointed to by str. Normally ”radix” is set to 10.

Example:

```
char string[7];
itoa( 417, string, 10 );   // *str is a pointer to a char array
```



SendInteger()

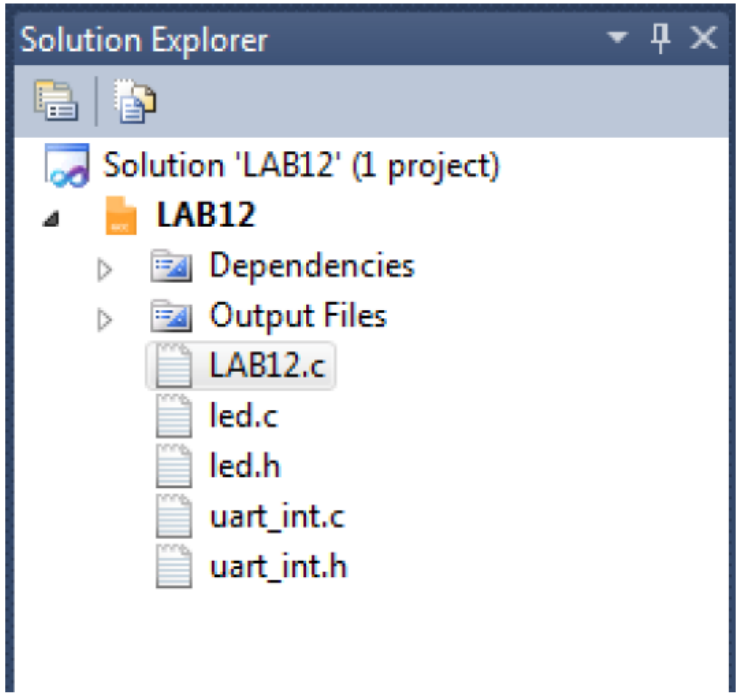
```
void SendInteger(int Number)
{
char array[7];
  // Convert the integer to an ASCII string (array), radix = 10
  itoa(Number, array, 10);
  // - then send the string
  SendString(array);
}
```

LAB12: Option for UART interrupt

```
/*****************************************
* "uart_int.h":                          *
* Header file for Mega32 UART driver.    *
* Henning Hargaard, 3/11 2011            *
*****************************************/
void InitUART(unsigned long BaudRate, unsigned char DataBit, unsigned char RX_int);
unsigned char CharReady();
char ReadChar();
void SendChar(char Ch);
void SendString(char* String);
void SendInteger(int Number);
/*****************************************/
```

If RX_int is 0, the UART receive interrupt is to be disabled (as in "the old driver").
If RX_int is different from 0, the UART receive interrupt is to be enabled.

LAB12



In the main() –function of ”LAB12.c” write code that:

- Initializes the UART, so that the UART receive interrupt is enabled.
- Performs global interrupt enable.
- Initializes the LED driver.
- Then go into an infinite loop (doing "nothing").

LAB12

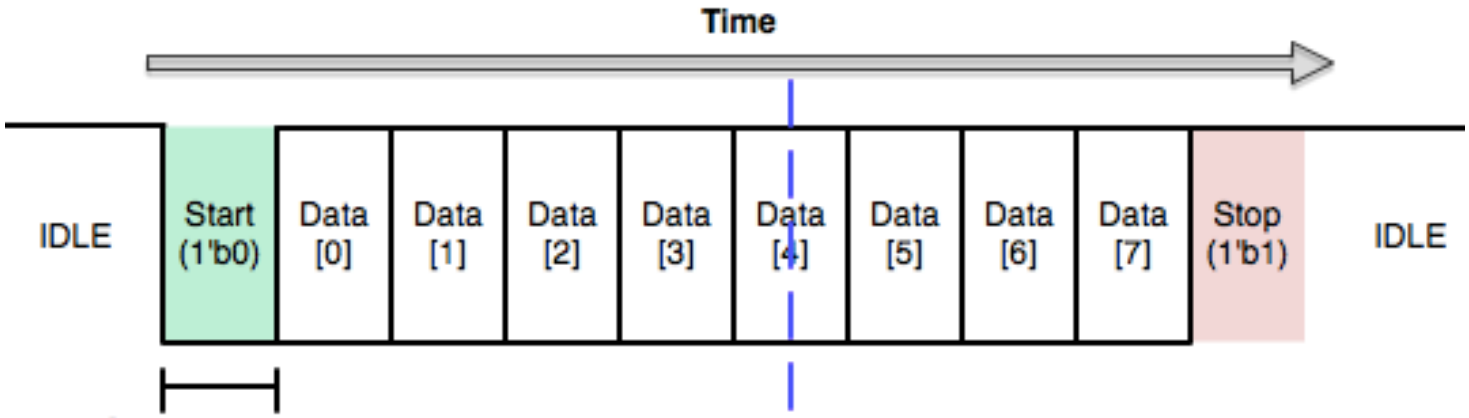
Also, write in "LAB12.c" an interrupt service routine (ISR) for UART receive interrupts.
The syntax is shown in Example 11-17, page 424 in the textbook.

In this ISR, place code to:

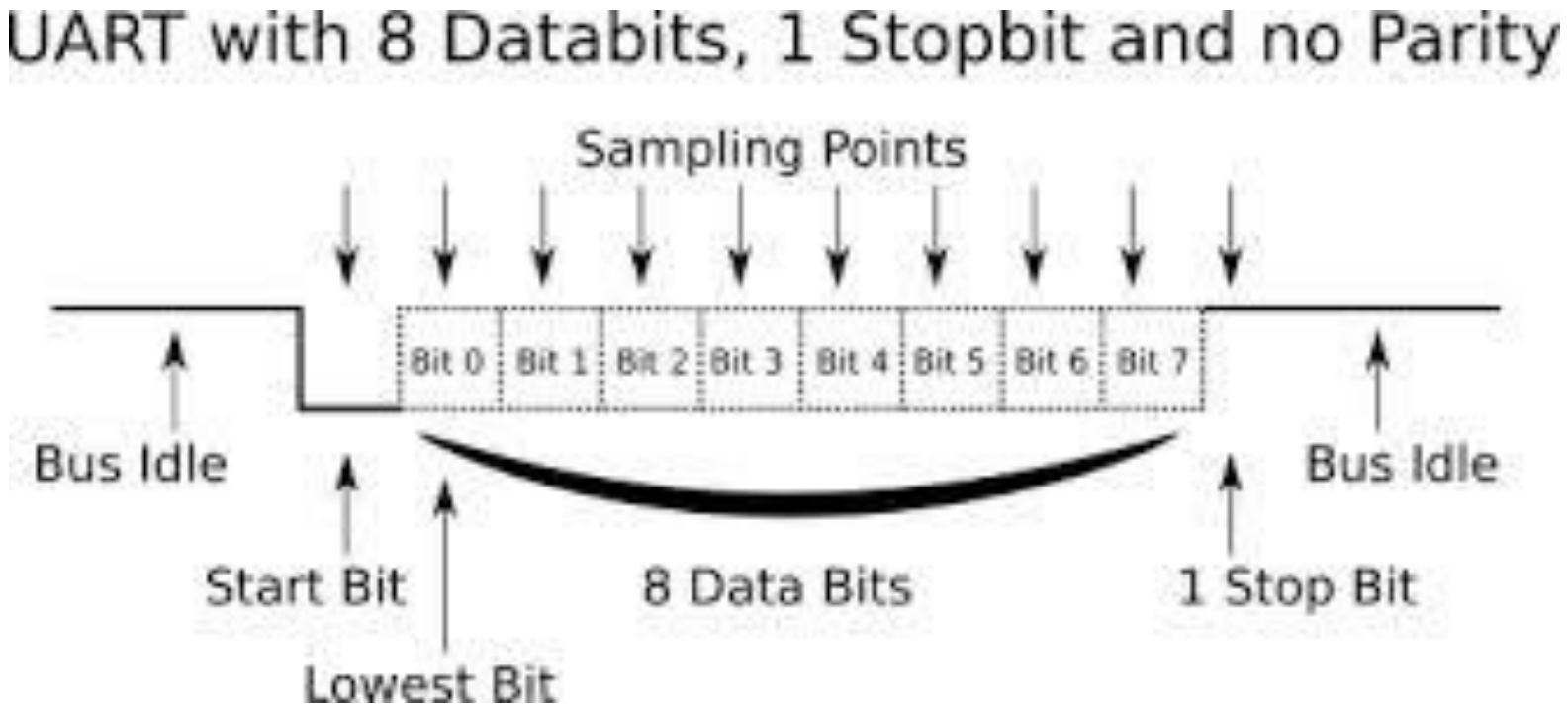
- Read the UART receive register UDR (= received character) to a local variable.
- If the received character is '0 ', '1', '2 ', '3', '4 ', '5', '6 'or '7', the corresponding LED must be toggled (if, for example the character is '3 ', LED number 3 must be toggled).
  Then send a text message back to the terminal: "LED number x is toggled." "x" is the number of the LED.

To receive characters do not use the **ReadChar( )** function, but only use receive interrupts.
To send characters one must use all relevant driver functions; e.g. **SendString( )** or **SendInteger( )**.

SW UART transmitter ?



SW UART receiver ?



End of lesson 18