1DT301, Computer Technology
Timers and USART
Tuesday, September 24, 2019

- Timers
- USART - The Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Serial Communication, RS232
- Introduction to lab 4
- Code examples for lab 4

PORTA on ATmega328p ?

Lab 4

- Square wave generator
- PWM
- Serial communication, RS 232

Lab 4

Task1: Square wave generator

Write a program in Assembly that creates a square wave. One LED should be connected and switch with the frequency 1 Hz. Duty cycle 50%. (On: 0.5 sec, Off: 0.5 sec.)
Use the timer function to create an interrupt with 2 Hz, which change between On and Off in the interrupt subroutine.

Task 2: Pulse Width Modulation (PWM)

Modify the program in Task 1 to obtain Pulse Width Modulation (PWM). The frequency should be fixed, but the duty cycle should be possible to change. Use two push buttons to change the duty cycle up and down. Use interrupt for each pushbutton. The duty cycle should be possible to change from 0 % up to 100 % in steps of 5 %. Connect the output to an oscilloscope, to visualize the change in duty cycle.

Linnéuniversitetet

Task 3: Serial communication

Write a program in Assembly that uses the serial communication port0 (RS232). Connect a computer to the serial port and use a terminal emulation program. (Ex. Hyper Terminal)
The program should receive characters that are sent from the computer, and show the code on the LEDs. For example, if you send character A, it has the hex code $65, the bit pattern is 0110 0101 and should be displayed with LEDs On for each 'one'. Use polled UART, which means that the UART should be checked regularly by the program.

Task 4: Serial communication with echo

Modify the program in task 3 to obtain an echo, which means that the received character should also be sent back to the terminal, This could be used as a confirmation in the terminal, to ensure that the character has been transferred correctly.

Task 5: Serial communication using Interrupt

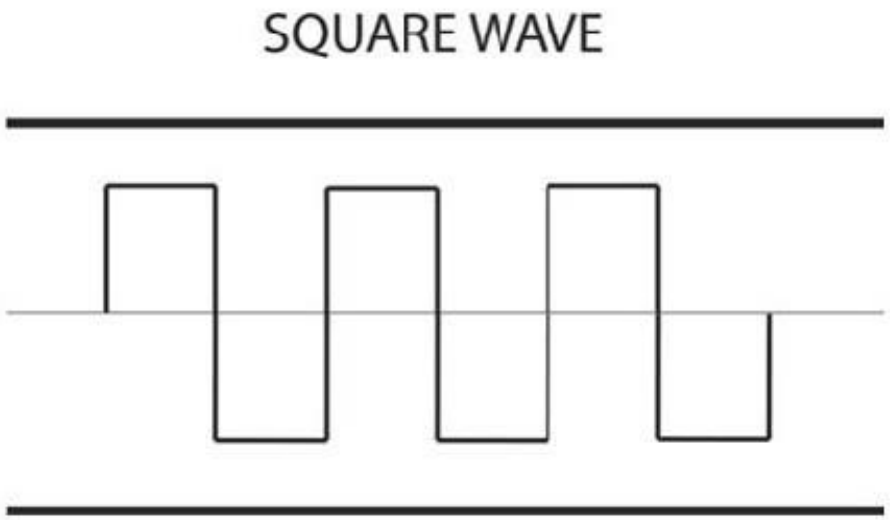
Do task 3 and 4, but use Interrupt instead of polled UART.
(USART, Rx Complete, USART Data Register Empty and USART, Tx Complete)

Lab 4, task 1:

Task1: Square wave generator

Write a program in Assembly that creates a square wave. One LED should be connected and switch with the frequency 1 Hz. Duty cycle 50%. (On: 0.5 sec, Off: 0.5 sec.)

Use the timer function to create an interrupt with 2 Hz, which change between On and Off in the interrupt subroutine.



Lab 4, task 1:

- Create an interrupt using a 8-bit timer, for example TIMER0.
- Select a prescaler value, that will divide the oscillator frequency.
  Example: osc. = 1 MHz, prescaler = 1024 => timer counts every ms. (1000 times / second)
- Let the timer give an interrupt with a delay that you choose, for example 10 ms.
  Maximum value is 255 = 255 ms.
- To get a delay of 500 ms, count 50 timerinterrupts, and toggle the LED.
- Alternatively, use a 16-bit timer, for example TIMER/COUNTER1, which can count up to 65 535 = 65 535 ms = 65,5 s.

TIMER0

interrupt vector table (ATMega2560)

```
; ***** INTERRUPT VECTORS ********************************************************
.equ    INT0addr        = 0x0002        ; External Interrupt Request 0
.equ    INT1addr        = 0x0004        ; External Interrupt Request 1
.equ    INT2addr        = 0x0006        ; External Interrupt Request 2
.equ    INT3addr        = 0x0008        ; External Interrupt Request 3
.equ    INT4addr        = 0x000a        ; External Interrupt Request 4
.equ    INT5addr        = 0x000c        ; External Interrupt Request 5
.equ    INT6addr        = 0x000e        ; External Interrupt Request 6
.equ    INT7addr        = 0x0010        ; External Interrupt Request 7
.equ    PCI0addr        = 0x0012        ; Pin Change Interrupt Request 0
.equ    PCI1addr        = 0x0014        ; Pin Change Interrupt Request 1
.equ    PCI2addr        = 0x0016        ; Pin Change Interrupt Request 2
.equ    WDTaddr = 0x0018        ; Watchdog Time-out Interrupt
.equ    OC2Aaddr        = 0x001a        ; Timer/Counter2 Compare Match A
.equ    OC2Baddr        = 0x001c        ; Timer/Counter2 Compare Match B
.equ    OVF2addr        = 0x001e        ; Timer/Counter2 Overflow
.equ    ICP1addr        = 0x0020        ; Timer/Counter1 Capture Event
.equ    OC1Aaddr        = 0x0022        ; Timer/Counter1 Compare Match A
.equ    OC1Baddr        = 0x0024        ; Timer/Counter1 Compare Match B
.equ    OC1Caddr        = 0x0026        ; Timer/Counter1 Compare Match C
.equ    OVF1addr        = 0x0028        ; Timer/Counter1 Overflow
.equ    OC0Aaddr        = 0x002a        ; Timer/Counter0 Compare Match A
.equ    OC0Baddr        = 0x002c        ; Timer/Counter0 Compare Match B
.equ    OVF0addr        = 0x002e        ; Timer/Counter0 Overflow
.equ    SPIaddr = 0x0030        ; SPI Serial Transfer Complete
.equ    URXC0addr       = 0x0032        ; USART0, Rx Complete
.equ    UDRE0addr       = 0x0034        ; USART0 Data register Empty
```

TIMER0
initialization

```
.org 0x72
restart:
        ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

        ldi temp, 0x01               ; initialize DDRB
        out DDRB, temp
```

In the file m2560def.inc, we can find:

```
.equ RAMEND = 0x21ff
```

TIMER0
initialization

```
ldi temp, 0x05          ; prescaler value to TCCR0
out TCCR0, temp         ; CS2 - CS2 = 101, osc.clock / 1024

ldi temp, (1<<TOIE0)    ; Timer 0 enable flag, TOIE0
out TIMSK, temp         ; to register TIMSK

ldi temp, 100           ; starting value for counter
out TCNT0, temp         ; counter register

sei                     ; enable global interrupt


start:

rjmp start              ; main loop
```

TIMER0
interrupt routine

```
timer0_int:
        push temp                       ; timer interrupt routine
        in temp, SREG                   ; save SREG on stack
        push temp

; additional code to create the square output

        ldi temp, 100                   ; starting value for counter
        out TCNT0, temp

        pop temp                        ; restore SREG
        out SREG, temp
        pop temp                        ; restore register
        reti                            ; return from interrupt
```

TIMER0 - TCCR0

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

TIMER0 - TIMSK

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, that is, when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, that is, when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

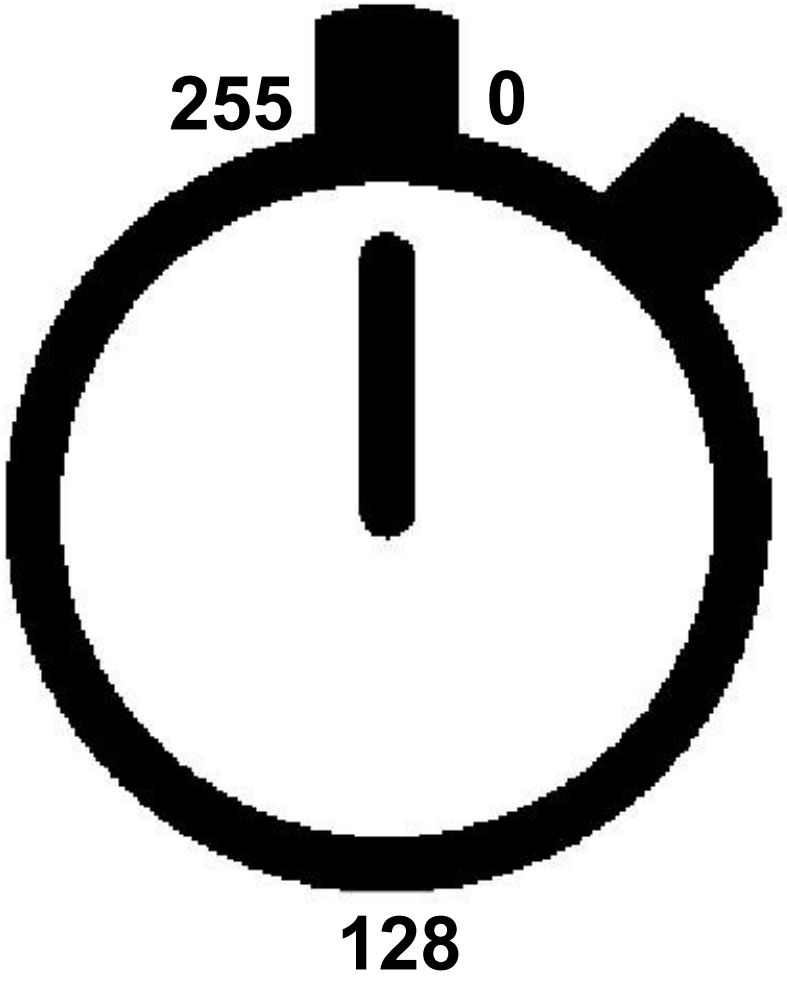
TIMER0 - TCNT0

Timer/Counter Register – TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.

TIMER0 - TCNT0



RJMP – Relative Jump

Description:

Relative jump to an address within PC - 2K +1 and PC + 2K (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

Operation:

(i) PC ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	-2K ≤ k < 2K	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

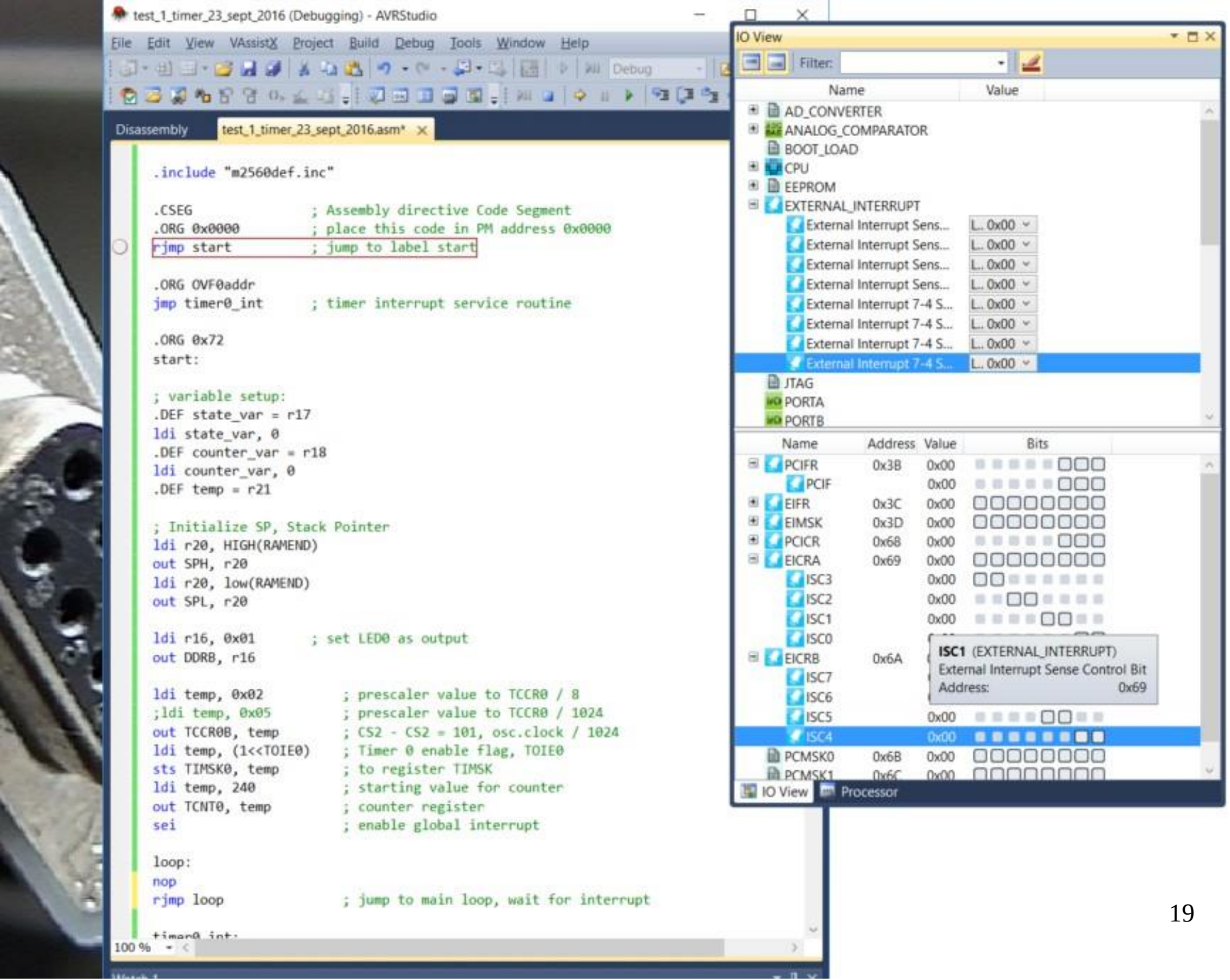
I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cpi     r16,$42   ; Compare r16 to $42
        brne    error     ; Branch if r16 <> $42
        rjmp    ok        ; Unconditional branch
error:  add     r16,r17   ; Add r17 to r16
        inc     r16       ; Increment r16
ok:     nop               ; Destination for rjmp (do nothing)
```

Words: 1 (2 bytes)

Cycles: 2

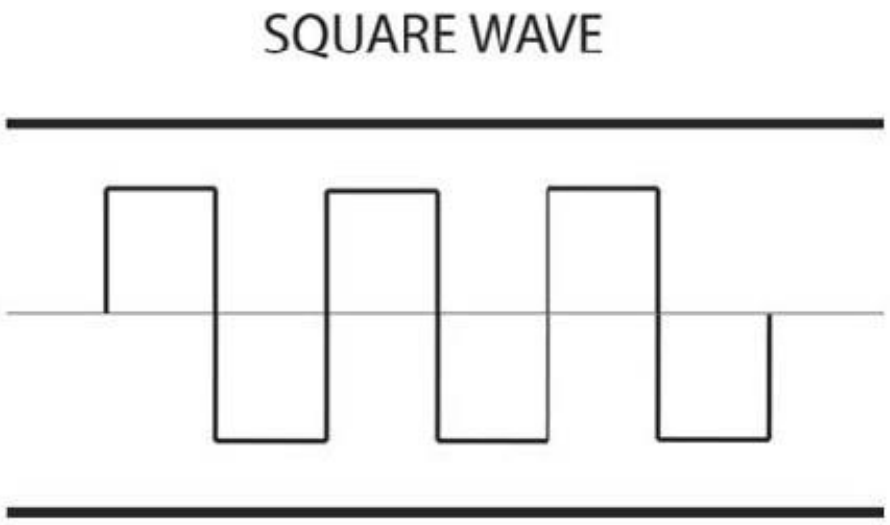

test_1_timer_23_sept_2016 (Debugging) - AVRStudio
File Edit View VAssistX Project Build Debug Tools Window Help
Disassembly
test_1_timer_23_sept_2016.asm*
.include "m2560def.inc"
.CSEG ; Assembly directive Code Segment
.ORG 0x0000 ; place this code in PM address 0x0000
rjmp start ; jump to label start
.ORG OVF0addr
jmp timer0_int ; timer interrupt service routine
.ORG 0x72
start:
; variable setup:
.DEF state_var = r17
ldi state_var, 0
.DEF counter_var = r18
ldi counter_var, 0
.DEF temp = r21
; Initialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)
out SPH, r20
ldi r20, low(RAMEND)
out SPL, r20
ldi r16, 0x01 ; set LED0 as output
out DDRB, r16
ldi temp, 0x02 ; prescaler value to TCCR0 / 8
;ldi temp, 0x05 ; prescaler value to TCCR0 / 1024
out TCCR0B, temp ; CS2 - CS2 = 101, osc.clock / 1024
ldi temp, (1<<TOIE0) ; Timer 0 enable flag, TOIE0
sts TIMSK0, temp ; to register TIMSK
ldi temp, 240 ; starting value for counter
out TCNT0, temp ; counter register
sei ; enable global interrupt
loop:
nop
rjmp loop ; jump to main loop, wait for interrupt
100 %
IO View
Filter:
Name
Value
AD_CONVERTER
ANALOG_COMPARATOR
BOOT_LOAD
CPU
EEPROM
EXTERNAL_INTERRUPT
External Interrupt Sens...
External Interrupt 7-4 S...
JTAG
PORTA
PORTB
Name
Address
Value
Bits
PCIFR
0x3B
PCIF
EIFR
0x3C
EIMSK
0x3D
PCICR
0x68
EICRA
0x69
ISC3
ISC2
ISC1
ISC0
EICRB
0x6A
ISC7
ISC6
ISC5
ISC4
PCMSK0
0x6B
ISC1 (EXTERNAL_INTERRUPT)
External Interrupt Sense Control Bit
Address: 0x69
IO View
Processor

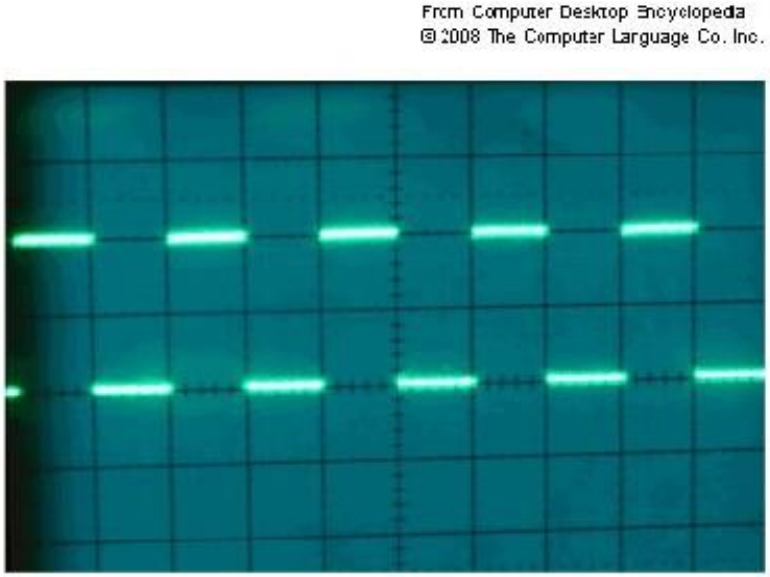
Lab 4, task 2:

Task 2: Pulse Width Modulation (PWM)

Modify the program in Task 1 to obtain Pulse Width Modulation (PWM). The frequency should be fixed, but the duty cycle should be possible to change. Use two push buttons to change the duty cycle up and down. Use interrupt for each pushbutton. The duty cycle should be possible to change from 0 % up to 100 % in steps of 5 %. Connect the output to an oscilloscope, to visualize the change in duty cycle.

Square Wave Generator





PWM, Pulse Width Modulation

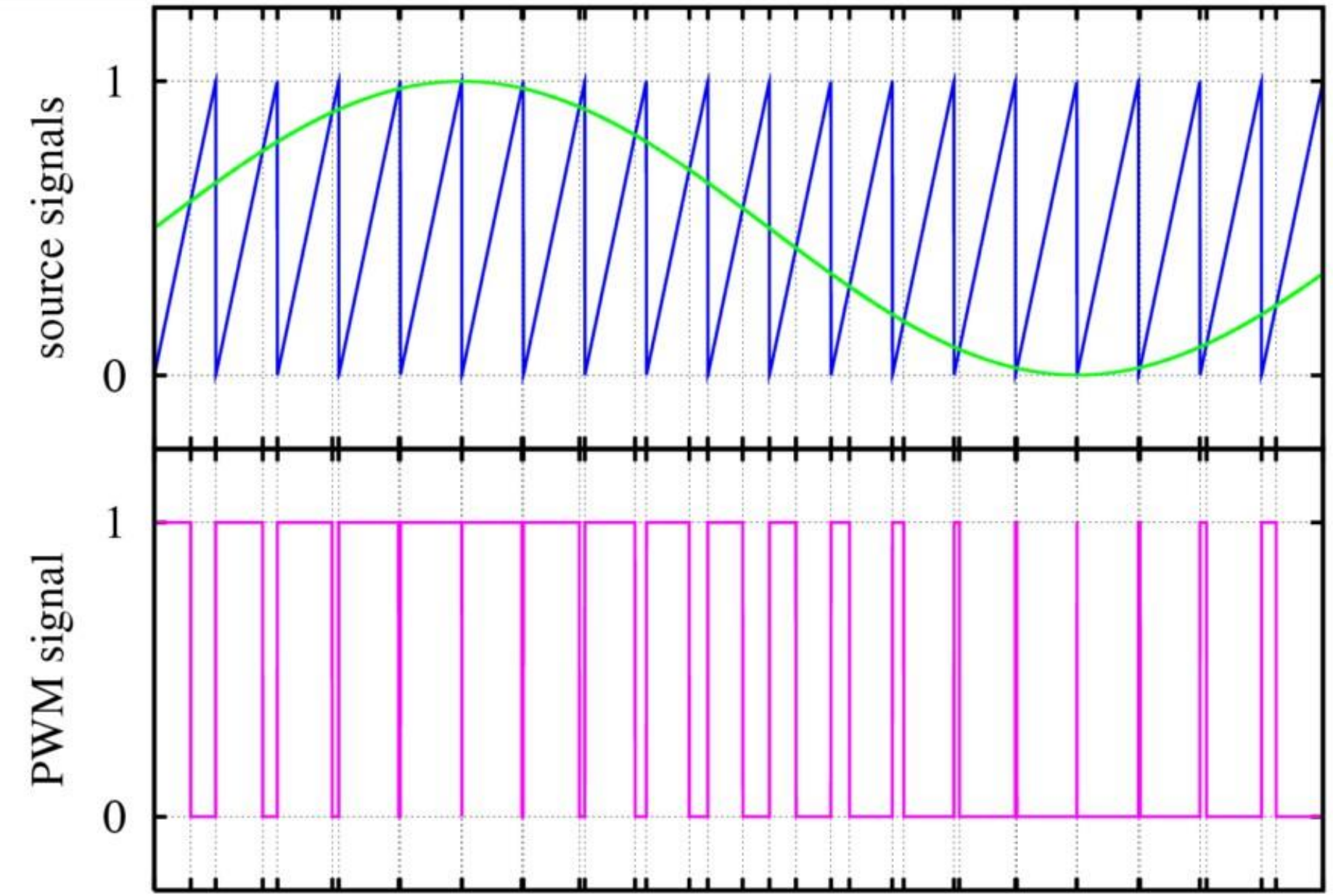
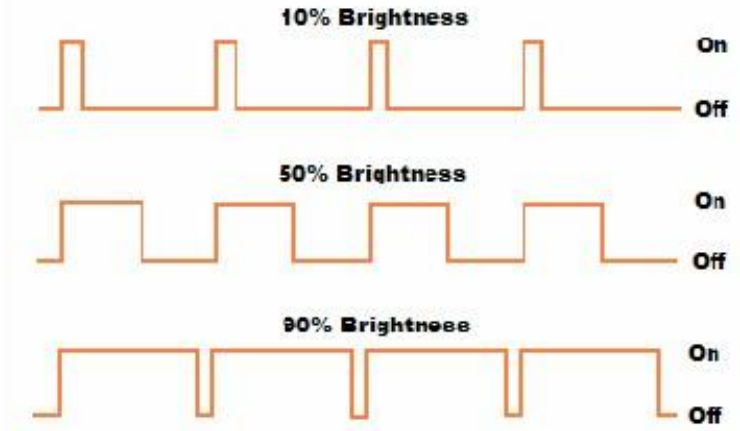
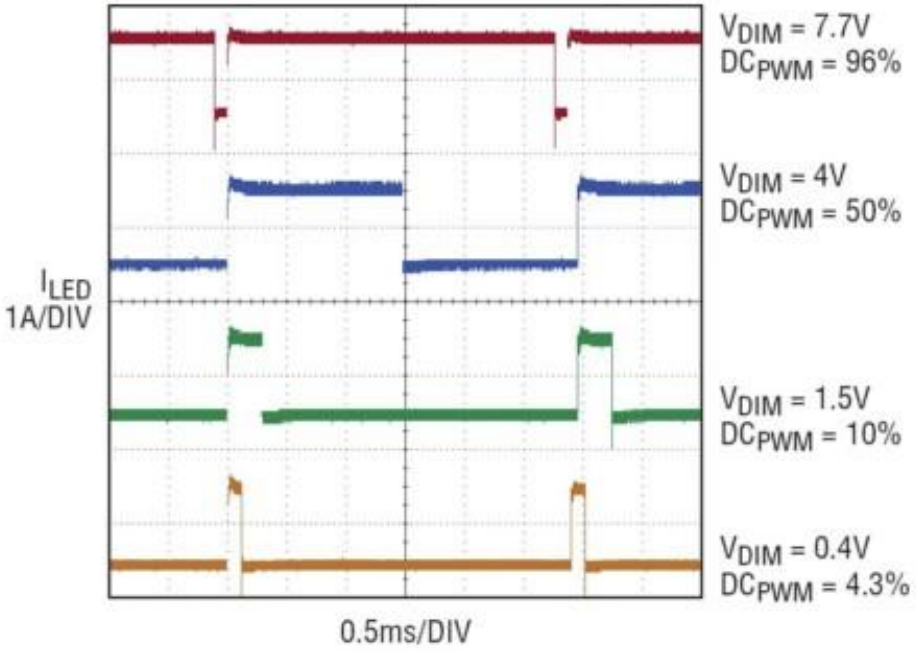


Fig. 3: Pulse Width Modulation (PWM)



PWM can also be used to control luminance and color uniformity.



Lab 4, task 3, 4:

Task 3: Serial communication

Write a program in Assembly that uses the serial communication port0 (RS232). Connect a computer to the serial port and use a terminal emulation program. (Ex. Hyper Terminal)
The program should receive characters that are sent from the computer, and show the code on the LEDs. For example, if you send character A, it has the hex code $65, the bit pattern is 0110 0101 and should be displayed with LEDs On for each 'one'. Use polled UART, which means that the UART should be checked regularly by the program.

Task 4: Serial communication with echo

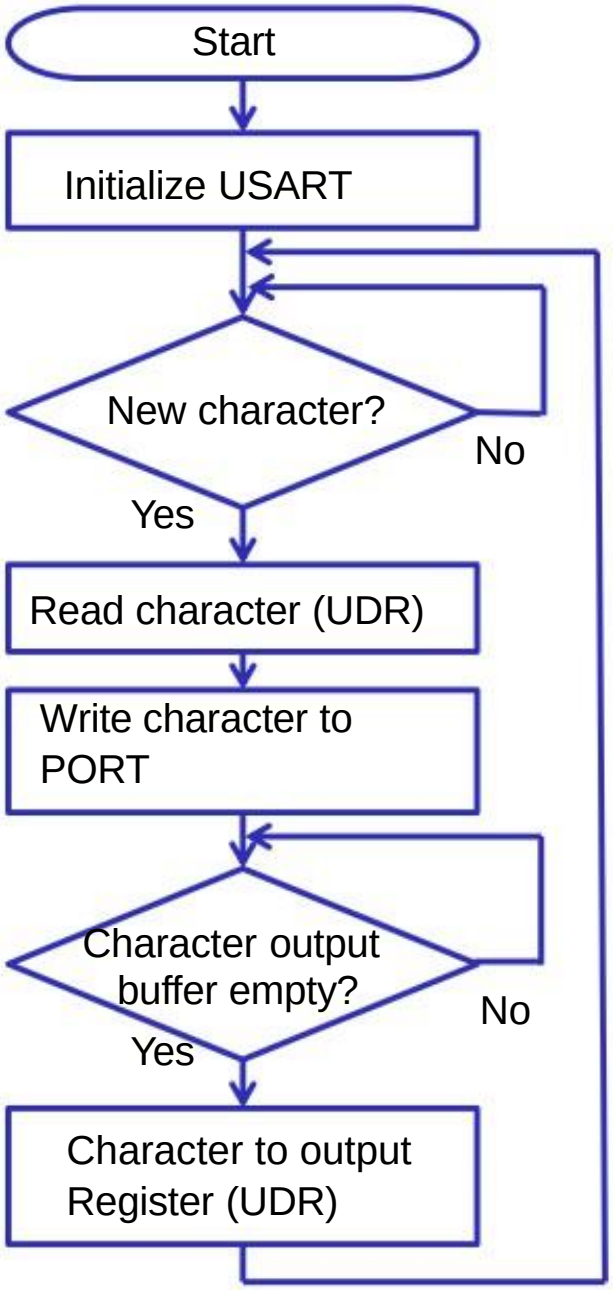
Modify the program in task 3 to obtain an echo, which means that the received character should also be sent back to the terminal, This could be used as a confirmation in the terminal, to ensure that the character has been transferred correctly.

Lab 4, task 5:

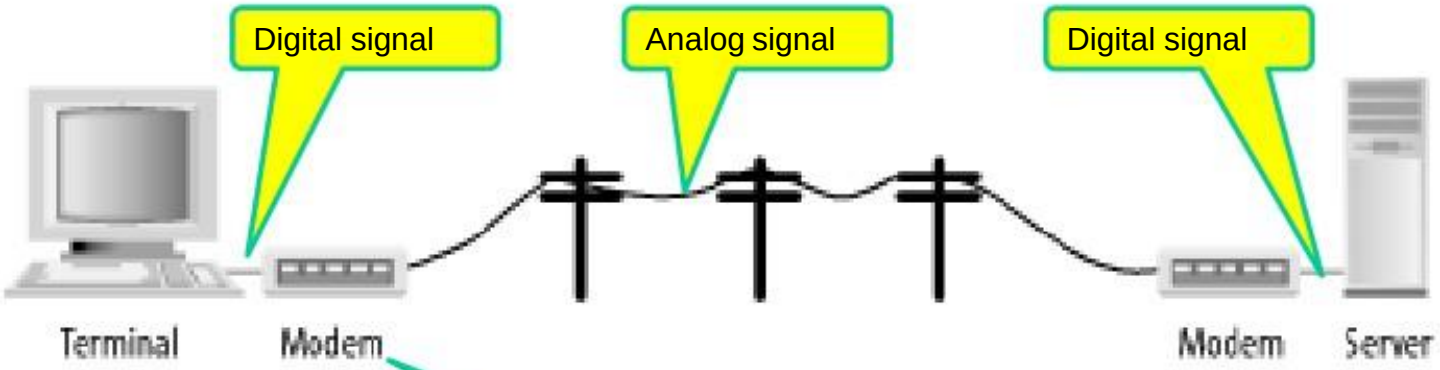
Task 5: Serial communication using Interrupt
Do task 3 and 4, but use Interrupt instead of polled UART.
(USART, Rx Complete, USART Data Register Empty and USART, Tx Complete)

Lab 4, task 3

Example,
polled reading of
serial port.



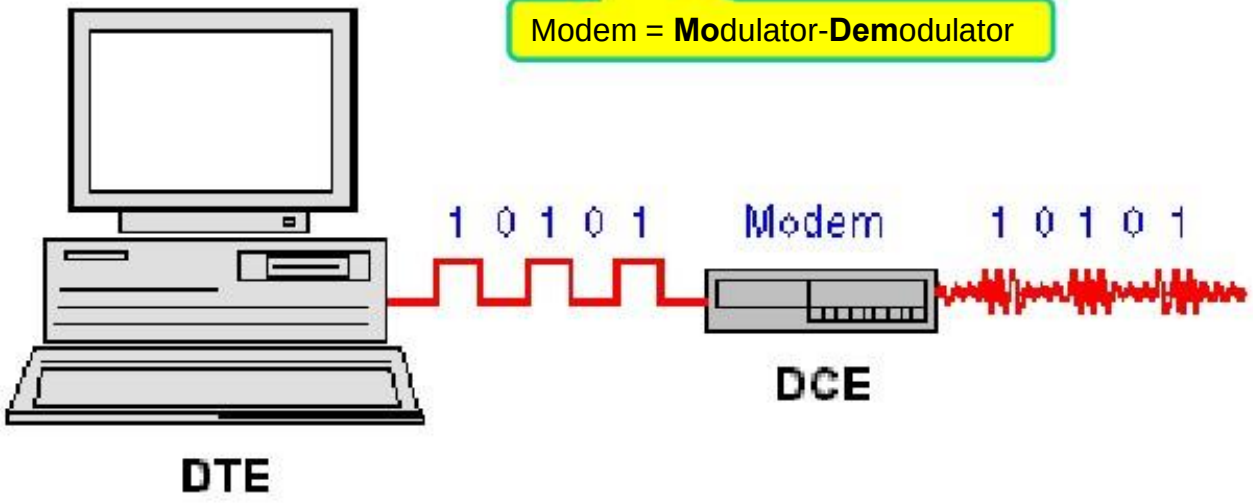
Serial Communication



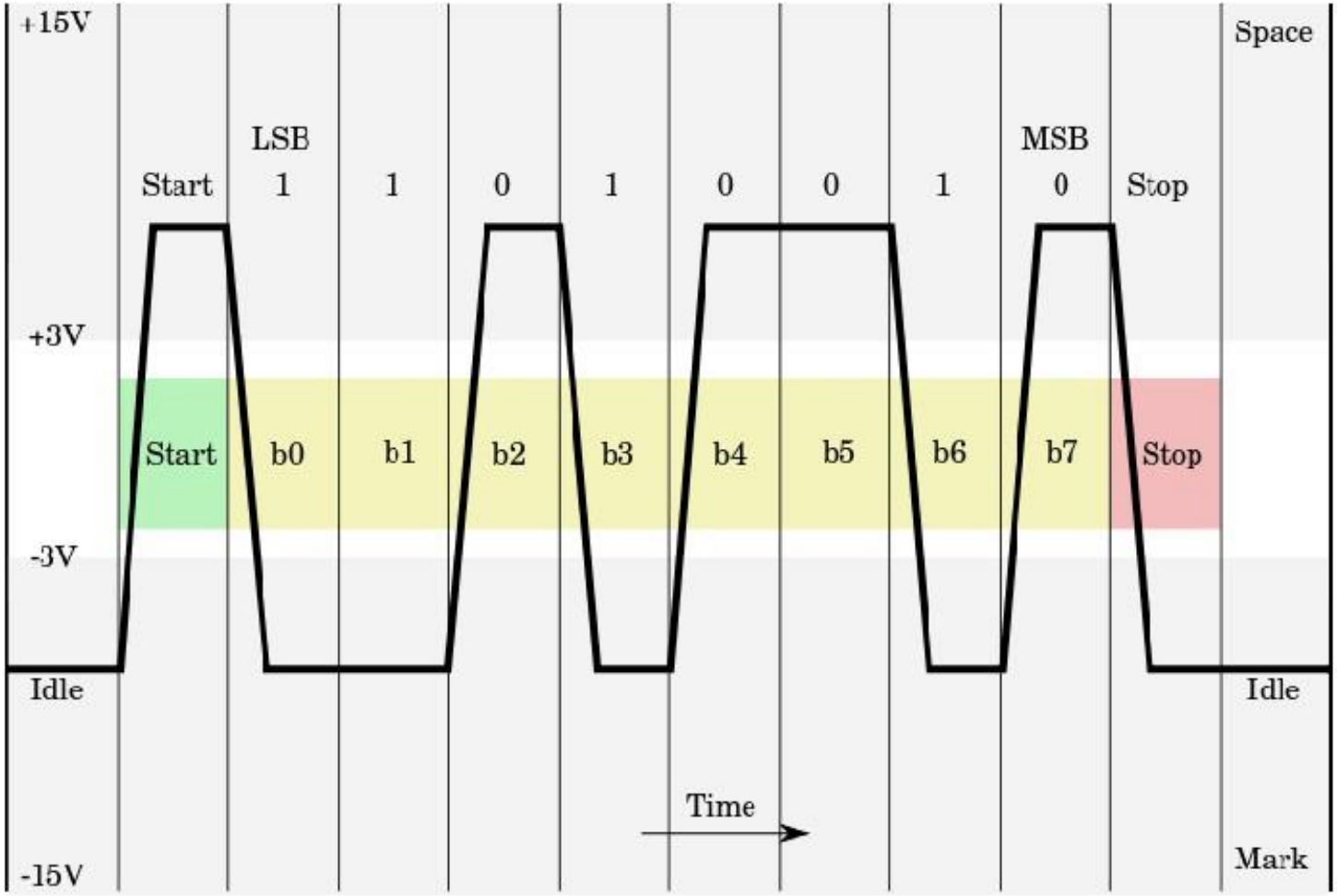
From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

Digital signal

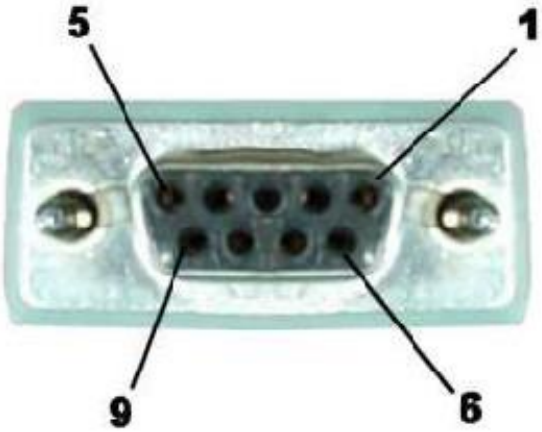
Modem = **Mo**dulator-**Dem**odulator



Serial communication, RS232

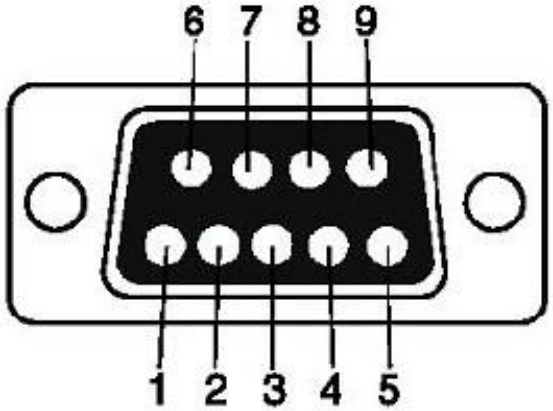


Serial communication, RS232



Note that Pin #9 is used for Contact Closure function (normally not used in RS-232).

Pin	Signal Description
1	DCD
2	TD
3	RD
4	DTR
5	Signal GND
6	DSR
7	RTS
8	CTS
9	contact closure



Acoustic modem

Acoustic modem

Old modem


POWER
Tx DATA
Rx DATA
CARRIER
ON LINE
VIEWTEXT
EXTERNAL CONTROL
ON LINE
LOCAL TEST
MIRACLE
WS2000
CCITT
ORIG
ANS
1200
1200 EQU
600
SMARTMODEM 2400
Hayes


US
NULL
LINE
FEED
FS


Copyright © 2006 Tommie Mademark

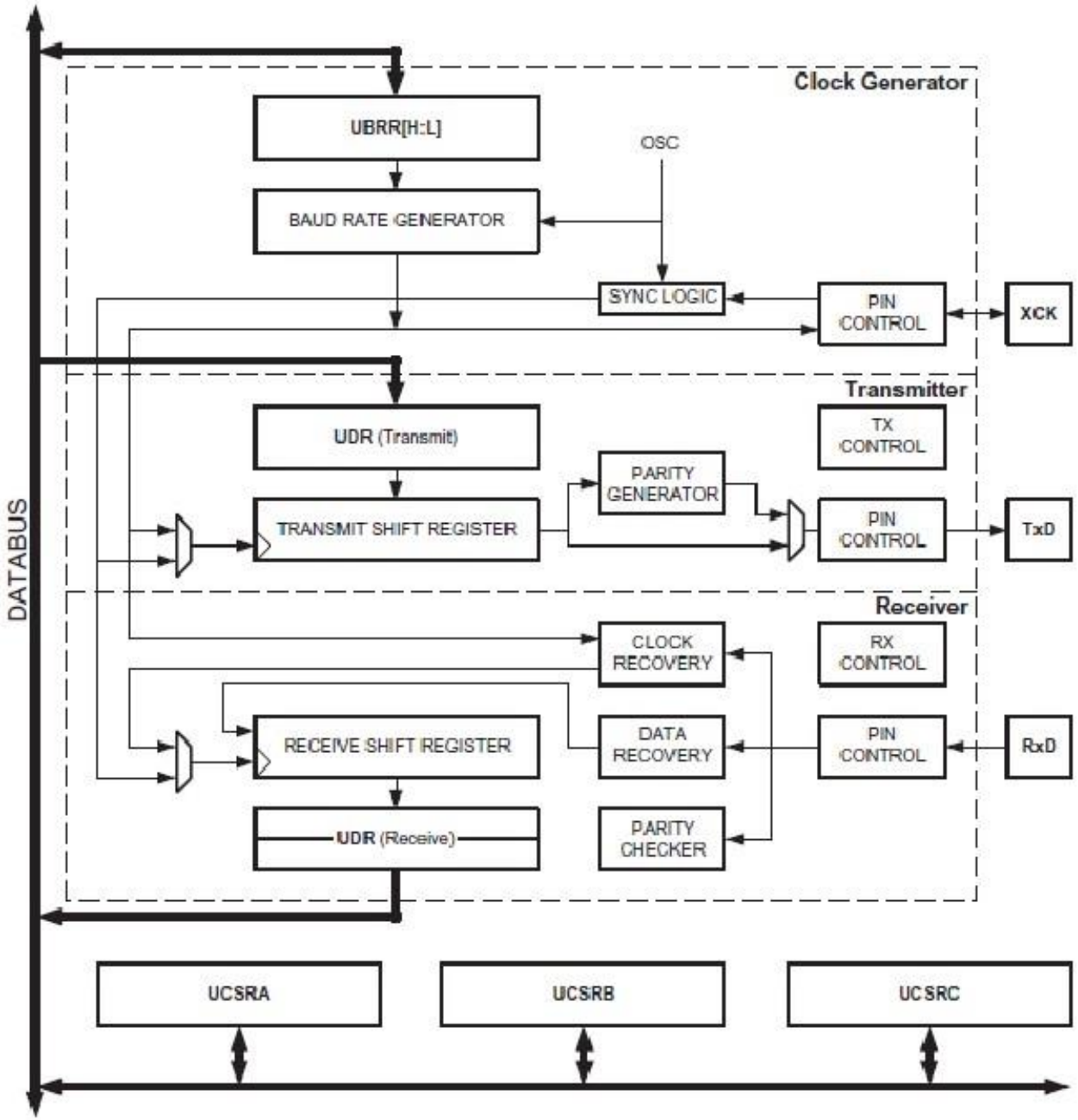
Original	RS232 on board baud rate
50	50
300	300
600	600
1200	1200
2400	2400
4800	4800
9600	9600
19.2K	19.2K
38.4K	38.4K
57.6K	57.6K
115.2K	115.2K

Table 22-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate [bps]	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4K	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2K	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8K	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4K	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6K	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8K	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2K	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4K	–	–	–	–	–	–	0	0.0%	–	–	–	–
250K	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max.(1)	62.5Kbps		125Kbps		115.2Kbps		230.4Kbps		125Kbps		250Kbps	

Note: 1. UBRR = 0, Error = 0.0%

Figure 69. USART Block Diagram[1]



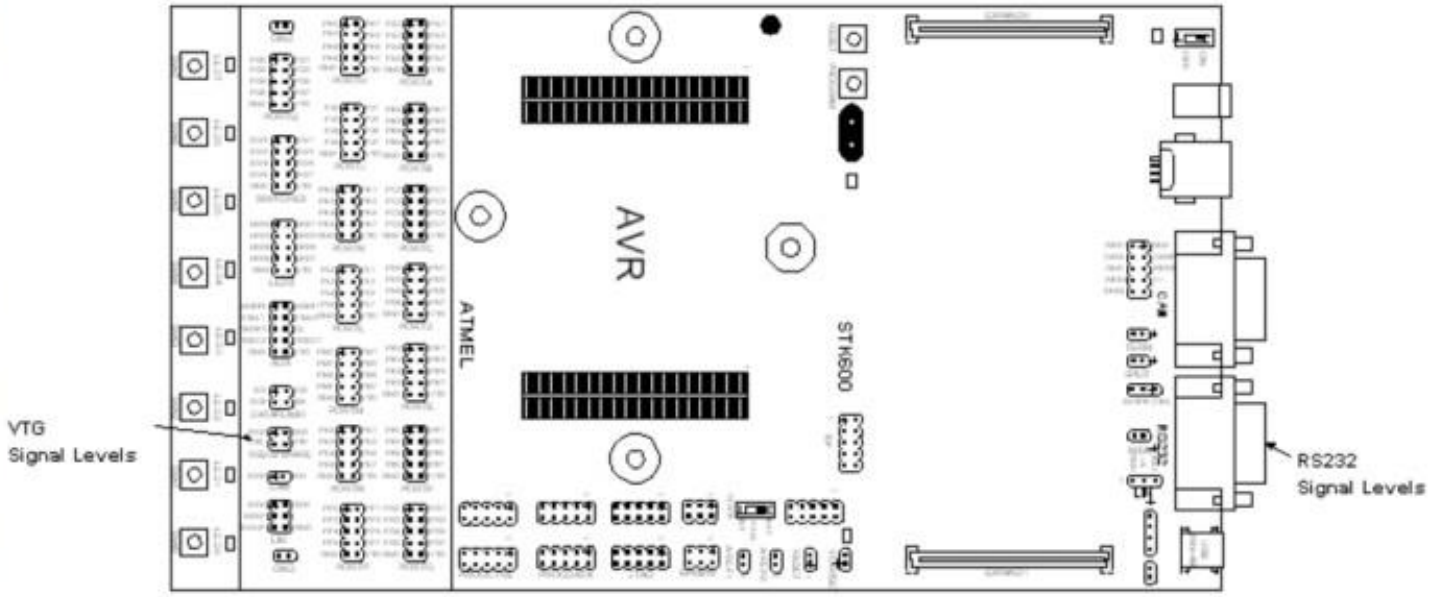
USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is highly flexible serial communication device. The main features are:

- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

User RS232 Interface

The STK600 includes a RS232 hardware that can be used for communication between the target AVR microcontroller in the socket and a PC serial port. STK600 has a 9-pin DSUB connector that can be connected to a PC with a straight serial cable (not a null modem cable).

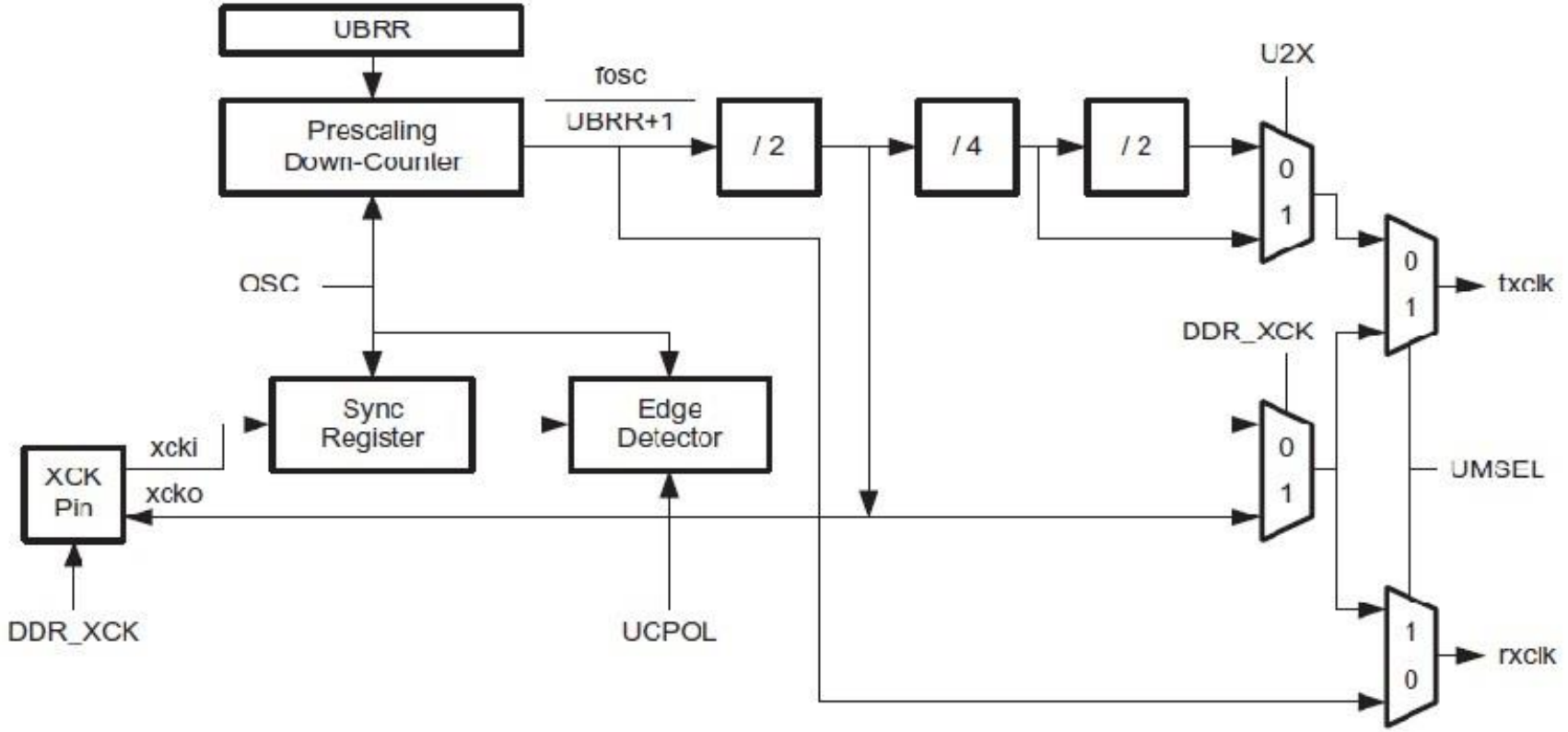


To use the RS232 interface, the AVR's UART pins must be connected to the appropriate pins on the "RS232 SPARE" pin header. Use a 2-wire cable to connect the AVR's RXD and TXD pins to the pin header. The "RS232 SPARE" pin header is found in the target header section, while the DSUB marked "RS232" is located on the other end of the card.

Optionally one can connect the RTS (Request To Send) and CTS (Clear To Send) signals to two free I/O ports. The RTS and CTS signals are used for flow control. The connection is shown below.

USART

Figure 70. Clock Generation Logic, Block Diagram



USART

Table 60. Equations for Calculating Baud Rate Register Setting

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)

Some examples of UBRR values for some system clock frequencies are found in Table 68 (see page 168).

USART

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 Kbps		125 Kbps		115.2 Kbps		230.4 Kbps		125 Kbps		250 Kbps	

1. UBRR = 0, Error = 0.0%

USART

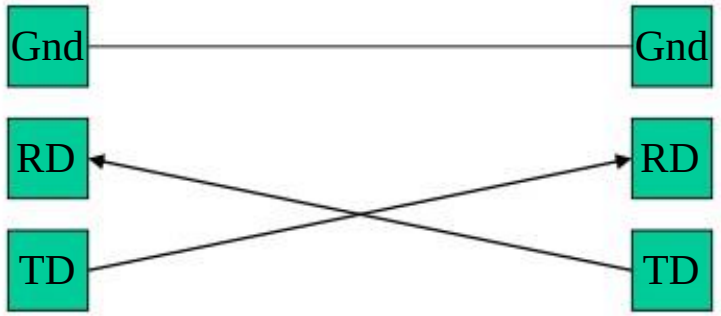
- Universal Synchronous and Asynchronous serial Receiver and Transmitter
- A standard I/O device that provides conversions between serial and parallel data
  - › Provides a basic protocol for serial communication
    - Speed, framing, error control

RS232

- This is a standard specification for the physical layer of a serial data interchange
  - › Specifies physical connections and voltage levels
  - › Data is transferred as a series of bits represented by voltage levels varying over time
- The encoding of data and signal interpretation is left to software

RS232 Basics

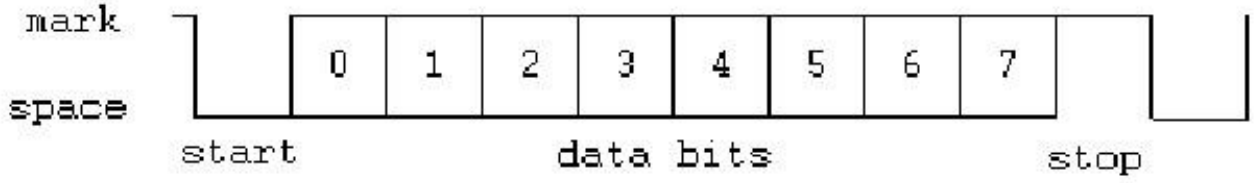
- For asynchronous communication, only three lines are needed
  - › Common ground
  - › Transmitted data (TD)
  - › Received data (RD)



    - Note that the transmit line on one end is the receive line on the other end
- Other lines are used for various signaling options

General Asynchronous Serial Communication

- The transmit line **idle** signal is 1 (**mark**)
- The transmitter signals the **start** of a **frame** by asserting a 0 (**space**) on TD
- At regular intervals (the baud rate), the transmitter asserts the bits of data, least significant first
- One or two **stop** bits (1) terminate the frame ending



AVR USART

- UART or USART is a standard I/O device
- The ATxmega128A1 has 8 independent USART integrated on-chip, producing logic signals for RS232 communication
  - › The microcontroller will only produce +/- Vcc, which is typically 5V; RS232 specifies +/-12V
  - › The Xplain connects to a compatible AT90USB1287 that converts the transmissions to USB format

USART Components

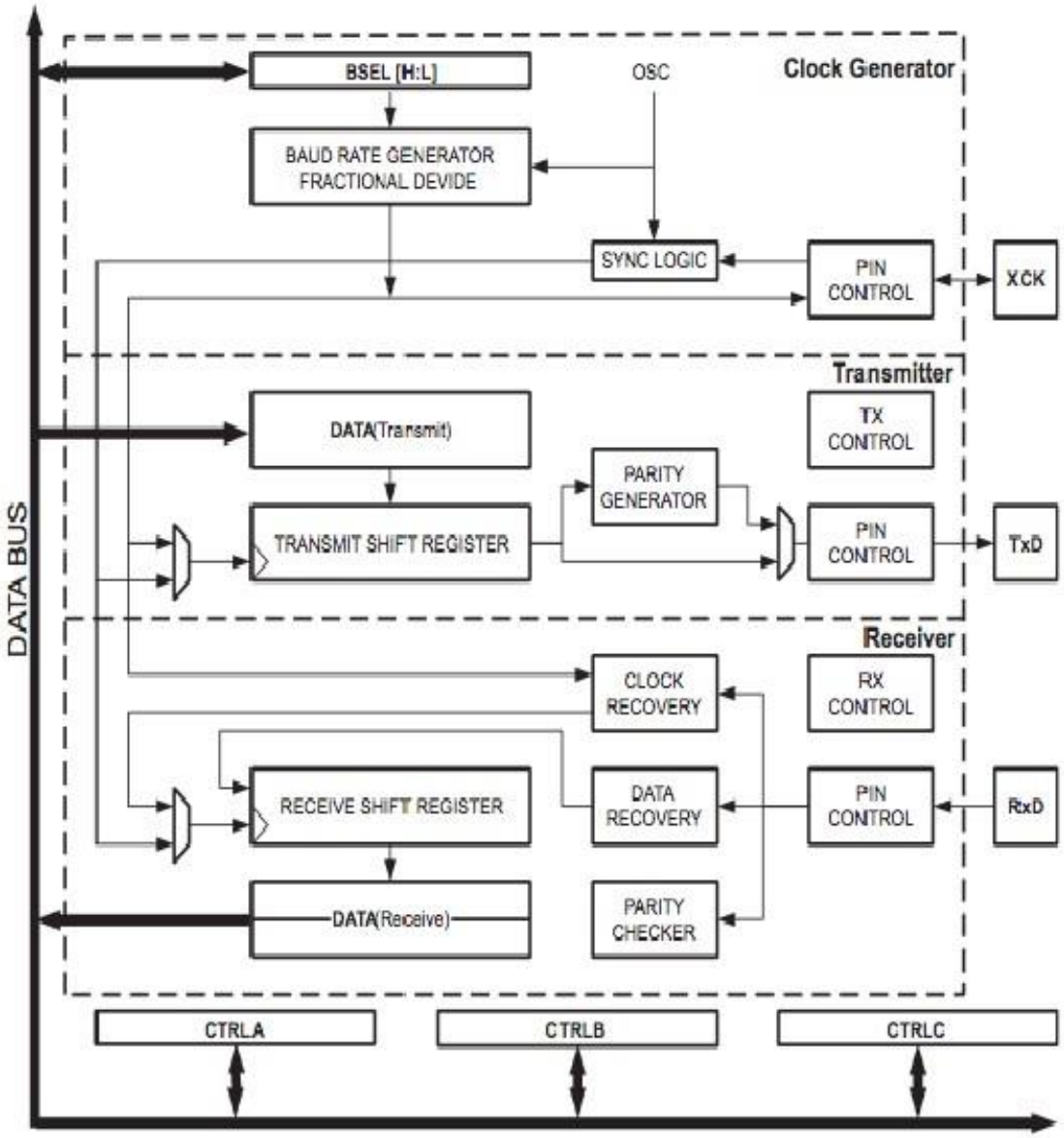
- Transmitter
  - › Manages stream of bits for each byte
- Receiver
  - › Manages receipt of bits and assembly into byte
- Clock Generator
  - › Allows the USART to operate in synchronous or asynchronous modes

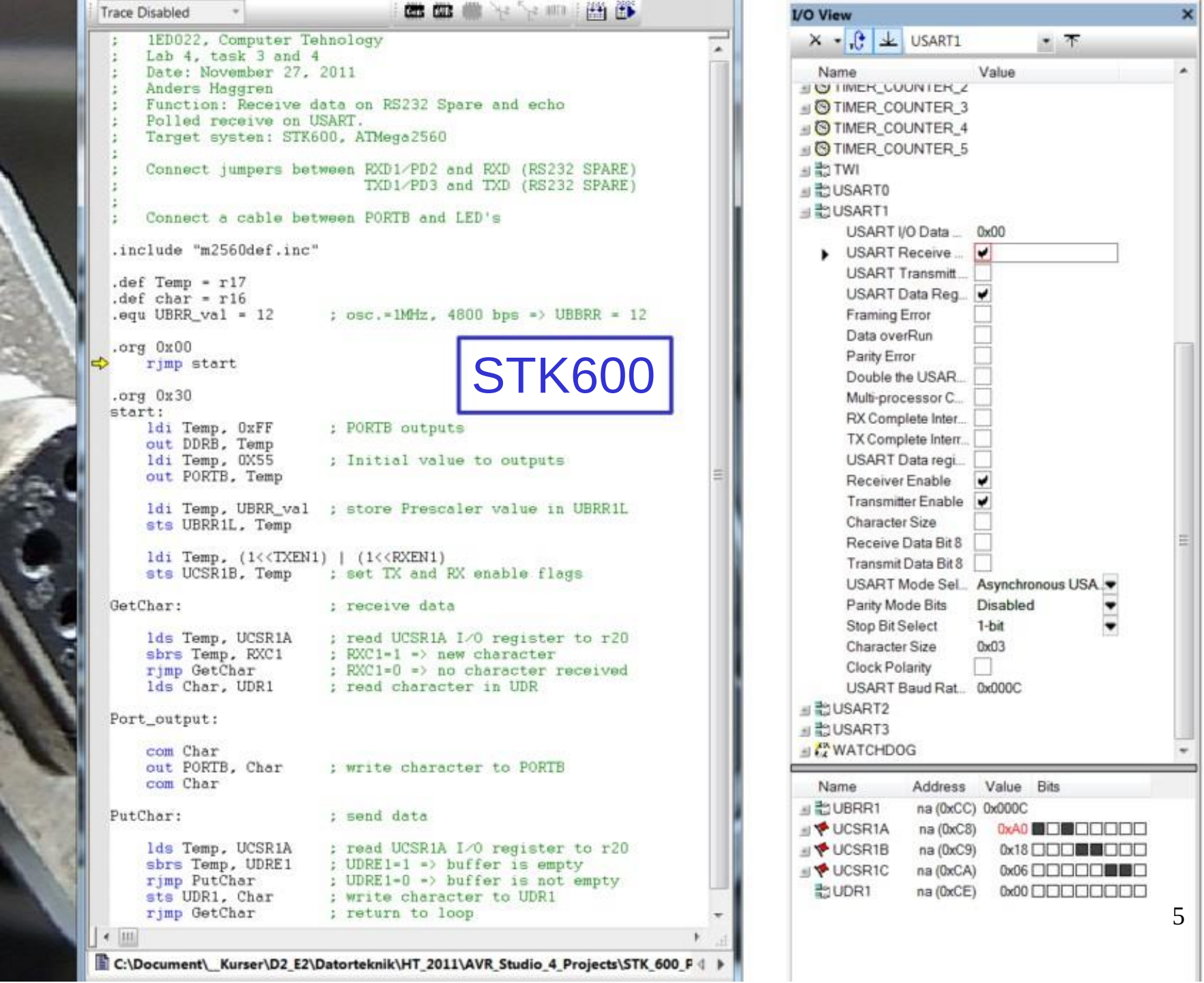
Physical Connections

- The USART utilizes two external pins on the microcontroller for transmit and receive
  - TXDC0 is PC2
  - RXDC0 is PC3
    - These bits of PORTC cannot be utilized for general digital I/O while serial communication is taking place

AVR USART

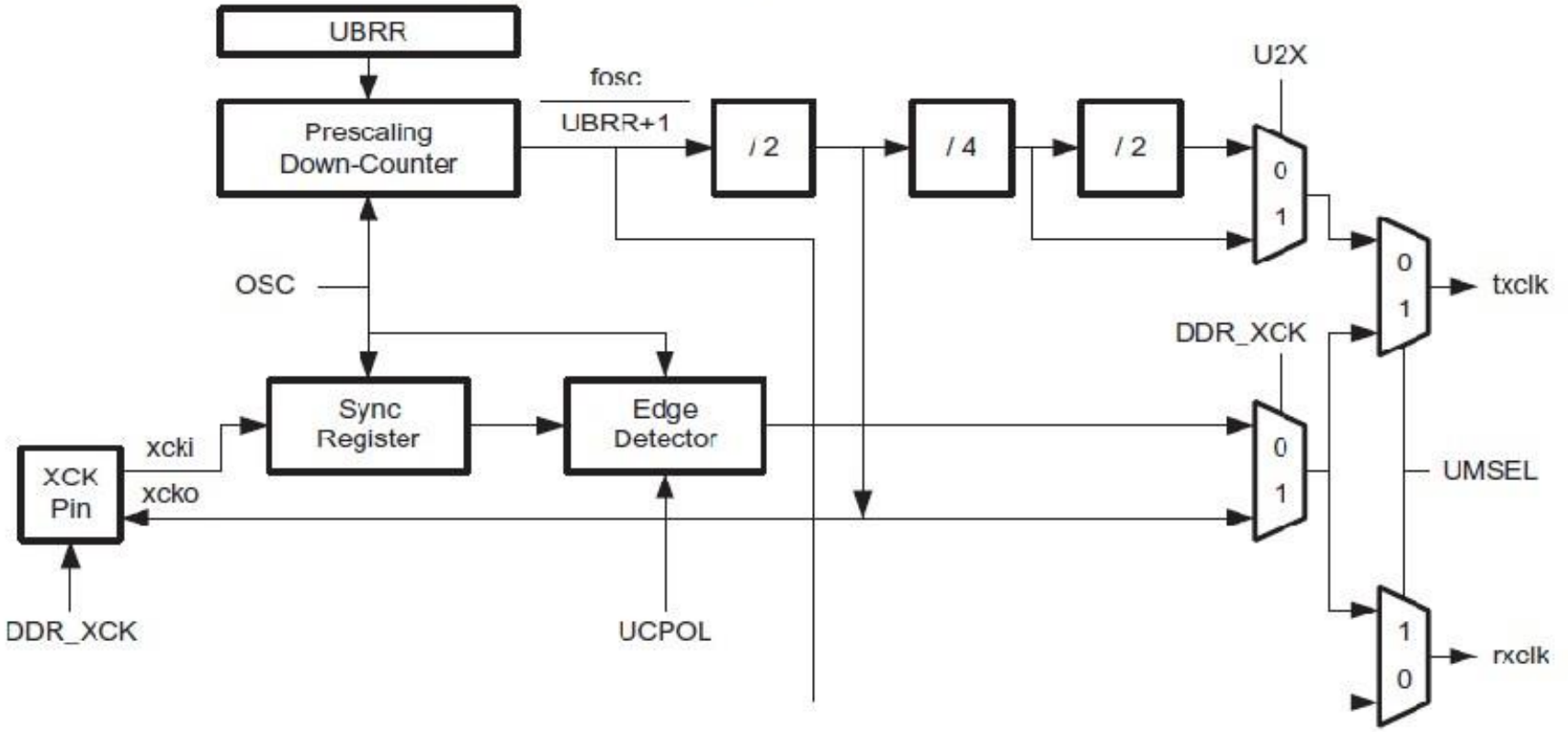
- Note the three
  control and
  Status
  registers
  and external
  pins




Trace Disabled
; 1ED022, Computer Tehnology
; Lab 4, task 3 and 4
; Date: November 27, 2011
; Anders Haggren
; Function: Receive data on RS232 Spare and echo
; Polled receive on USART.
; Target systen: STK600, ATMega2560
;
; Connect jumpers between RXD1/PD2 and RXD (RS232 SPARE)
; TXD1/PD3 and TXD (RS232 SPARE)
;
; Connect a cable between PORTB and LED's
.include "m2560def.inc"
.def Temp = r17
.def char = r16
.equ UBRR_val = 12 ; osc.=1MHz, 4800 bps => UBBRR = 12
.org 0x00
rjmp start
.org 0x30
start:
ldi Temp, 0xFF ; PORTB outputs
out DDRB, Temp
ldi Temp, 0X55 ; Initial value to outputs
out PORTB, Temp
ldi Temp, UBRR_val ; store Prescaler value in UBRR1L
sts UBRR1L, Temp
ldi Temp, (1<<TXEN1) | (1<<RXEN1)
sts UCSR1B, Temp ; set TX and RX enable flags
GetChar: ; receive data
lds Temp, UCSR1A ; read UCSR1A I/O register to r20
sbrs Temp, RXC1 ; RXC1=1 => new character
rjmp GetChar ; RXC1=0 => no character received
lds Char, UDR1 ; read character in UDR
Port_output:
com Char
out PORTB, Char ; write character to PORTB
com Char
PutChar: ; send data
lds Temp, UCSR1A ; read UCSR1A I/O register to r20
sbrs Temp, UDRE1 ; UDRE1=1 => buffer is empty
rjmp PutChar ; UDRE1=0 => buffer is not empty
sts UDR1, Char ; write character to UDR1
rjmp GetChar ; return to loop
STK600
C:\Document_Kurser\D2_E2\Datorteknik\HT_2011\AVR_Studio_4_Projects\STK_600_P
I/O View
USART1
Name
Value
TIMER_COUNTER_3
TIMER_COUNTER_4
TIMER_COUNTER_5
TWI
USART0
USART1
USART I/O Data ... 0x00
USART Receive ...
USART Transmitt...
USART Data Reg...
Framing Error
Data overRun
Parity Error
Double the USAR...
Multi-processor C...
RX Complete Inter...
TX Complete Interr...
USART Data regi...
Receiver Enable
Transmitter Enable
Character Size
Receive Data Bit 8
Transmit Data Bit 8
USART Mode Sel... Asynchronous USA...
Parity Mode Bits Disabled
Stop Bit Select 1-bit
Character Size 0x03
Clock Polarity
USART Baud Rat... 0x000C
USART2
USART3
WATCHDOG
Name Address Value Bits
UBRR1 na (0xCC) 0x000C
UCSR1A na (0xC8) 0xA0
UCSR1B na (0xC9) 0x18
UCSR1C na (0xCA) 0x06
UDR1 na (0xCE) 0x00

USART

Figure 70. Clock Generation Logic, Block Diagram



USART

Table 60. Equations for Calculating Baud Rate Register Setting

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)

Some examples of UBRR values for some system clock frequencies are found in Table 68 (see page 168).

USART

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 Kbps		125 Kbps		115.2 Kbps		230.4 Kbps		125 Kbps		250 Kbps	

1. UBRR = 0, Error = 0.0%

UBRR

- The prescaler value stored in UBRR is a value that depends on oscillator frequency and selected BAUD rate.
- If the value is lower than 255, it will fit in one register, UBRRL. (UBRR = UBRRL in ATmega16)
- Example:
  BAUD rate is 4800 bps,
  oscillator frequency is 1 MHz => UBRR is 12

UBRR

```
.equ UBRR_val = 12              ; osc. = 1MHz, BAUD rate 4800 bps

        ldi Temp, UBRR_val      ; store Prescaler value in UBBRRL
        out UBRRL, Temp

.def Temp = r17

        ldi Temp, (1<<TXEN) | (1<<RXEN)
        out UCR, Temp           ; set TX and RX Enable flags
```

USART

Assembly Code Example[(1)]

```
USART_Init:
  ; Set baud rate
  out   UBRRH, r17
  out   UBRRL, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ; Set frame format: 8data, 2stop bit
  ldi   r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

USART

USART Control and Status Register B – UCSRB

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 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete Interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete Interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty Interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

USART

USART Control and Status Register B – UCSRB

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 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, that is, when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

USART

USART Control and Status Register C – UCSRC

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	

The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/ UCSRC Registers" on page 162 section which describes how to access this register.

- **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between Asynchronous and Synchronous mode of operation.

Table 63. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

USART

```
.equ    UDR     =$0c
.equ    UCSRA   =$0b
.equ    USR     =$0b    ; For compatibility with S8535
.equ    UCSRB   =$0a
.equ    UCR     =$0a    ; For compatibility with S8535
.equ    UCSRC   =$20    ; Note! UCSRC equals UBRRH
.equ    UBRR    =$09
.equ    UBRRL   =$09    ; New name for UBRR
```

Dvs:

UBRR = UBRRL = $ 09
UCR = UCSRB = $ 0A
USR = UCSRA = $ 0B
UDR = $ 0C
UCRC = UBRRH = $ 20

UDR

USART Register Description

USART I/O Data Register – UDR

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	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

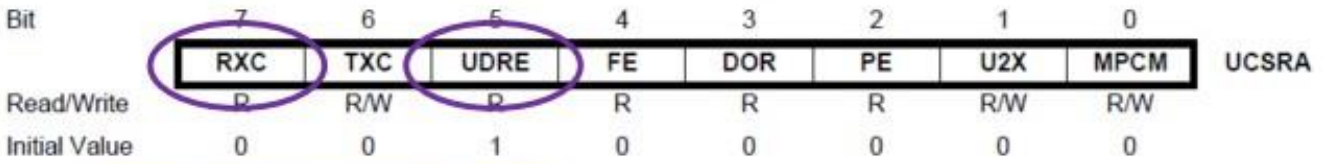
For 5-bit, 6-bit, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use read modify write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

UCSRA - USR

USART Control and Status Register A – UCSRA



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	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (that is, does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register empty Interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the transmitter is ready.

```
;   1ED022, Computer Tehnology
;   Lab 4, task 3 and 4
;   Date: November 27, 2011
;   Anders Haggren
;   Function: Receive data on RS232 Spare and echo
;   Polled receive on USART.
;   Target systen: STK600, ATMega2560

.include "m2560def.inc"

.def Temp = r17
.def char = r16
.equ UBRR_val = 12      ; osc.=1MHz, 4800 bps => UBBRR = 12

.org 0x00
    rjmp start

.org 0x30
start:
    ldi Temp, 0xFF      ; PORTB outputs
    out DDRB, Temp
    ldi Temp, 0X55      ; Initial value to outputs
    out PORTB, Temp

    ldi Temp, UBRR val  ; store Prescaler value in UBRR1L
    sts UBRR1L, Temp

    ldi Temp, (1<<TXEN1) | (1<<RXEN1)
    sts UCSR1B, Temp    ; set TX and RX enable flags

GetChar:                ; receive data
```

```
.org 0x30
start:
    ldi Temp, 0xFF      ; PORTB outputs
    out DDRB, Temp
    ldi Temp, 0X55      ; Initial value to outputs
    out PORTB, Temp

    ldi Temp, UBRR val  ; store Prescaler value in UBRR1L
    sts UBRR1L, Temp

    ldi Temp, (1<<TXEN1) | (1<<RXEN1)
    sts UCSR1B, Temp    ; set TX and RX enable flags

GetChar:                ; receive data

    lds Temp, UCSR1A    ; read UCSR1A I/O register to r20
    sbrs Temp, RXC1     ; RXC1=1 => new character
    rjmp GetChar        ; RXC1=0 => no character received
    lds Char, UDR1      ; read character in UDR

Port_output:

    com Char
    out PORTB, Char     ; write character to PORTB
    com Char

PutChar:                ; send data

    lds Temp, UCSR1A    ; read UCSR1A I/O register to r20
    sbrs Temp, UDRE1    ; UDRE1=1 => buffer is empty
    rjmp PutChar        ; UDRE1=0 => buffer is not empty
    sts UDR1, Char      ; write character to UDR1
    rjmp GetChar        ; return to loop
```

Initialize USART

Select BAUD rate, register UBBRL

STK500:

```
.equ UBRR_val = 12          ; osc. = 1MHz, BAUD rate 4800 bps

      ldi Temp, UBRR_val    ; store Prescaler value in UBBRRL
      out UBRRL, Temp
```

STK600:

```
.equ UBRR_val = 12          ; osc. = 1MHz, BAUD rate 4800 bps

      ldi Temp, UBRR_val    ; store Prescaler value in UBBRRL
      sts UBRR1L, Temp
```

Initialize USART
Set enable flags in UCR

STK500:

```
.def Temp = r17

        ldi Temp, (1<<TXEN) | (1<<RXEN)
        out UCR, Temp           ; set TX and RX Enable flags
```

STK600:

```
.def Temp = r17

        ldi Temp, (1<<TXEN1) | (1<<RXEN1)
        sts UCSR1B, Temp        ; set TX and RX Enable flags
```

Polled input,
check data received flag

STK500:

```
GetChar:                    ; receive data

        sbis USR, RXC       ; wait for character input
        rjmp GetChar        ; loop if no character input
        in r16, UDR         ; read character in UDR to r16
```

STK600:

```
GetChar:                    ; receive data
        lds r20, UCSR1A
        andi r20, 0x80      ; mask out bit 7 = RXC1
        cpi r20, 0x80       ; RXC1 = bit 7 in UCSR1A
        brne GetChar        ; RXC1 = 0, no character received
        lds r16, UDR1       ; read character in UDR to r16
```

Polled input / output,
send data

STK500:

```
PutChar:                    ; send data
        sbis USR, UDRE      ; wait if last character not yet sent
        rjmp PutChar        ; loop until UDRE flag  = 1, in USR
        out UDR, r16        ; write character in UDR
```

STK600:

```
PutChar:                    ; send data

        lds r20, UCSR1A
        andi r20, 0x20      ; mask out bit 5 = UDRE1
        cpi r20, 0x20       ; mask out bit 7 = RXC1
        brne PutChar        ; UDRE1 = 1, buffer is empty
        sts UDR1, r16       ; write character to UDR1
```

Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 13-12.

Table 13-12. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	T0 (Timer/Counter0 Clock Input)
PD6	T1 (Timer/Counter1 Clock Input)
PD5	XCK1 (USART1 External Clock Input/Output)
PD4	ICP1 (Timer/Counter1 Input Capture Trigger)
PD3	INT3/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)
PD2	INT2/RXD1 (External Interrupt2 Input or USART1 Receive Pin)
PD1	INT1/SDA (External Interrupt1 Input or TWI Serial DAta)
PD0	INT0/SCL (External Interrupt0 Input or TWI Serial CLock)

The alternate pin configuration is as follows:

- **INT3/TXD1 – Port D, Bit 3**

INT3, External Interrupt source 3: The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, Transmit Data (Data output pin for the USART1). When the USART1 Transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

- **INT2/RXD1 – Port D, Bit 2**

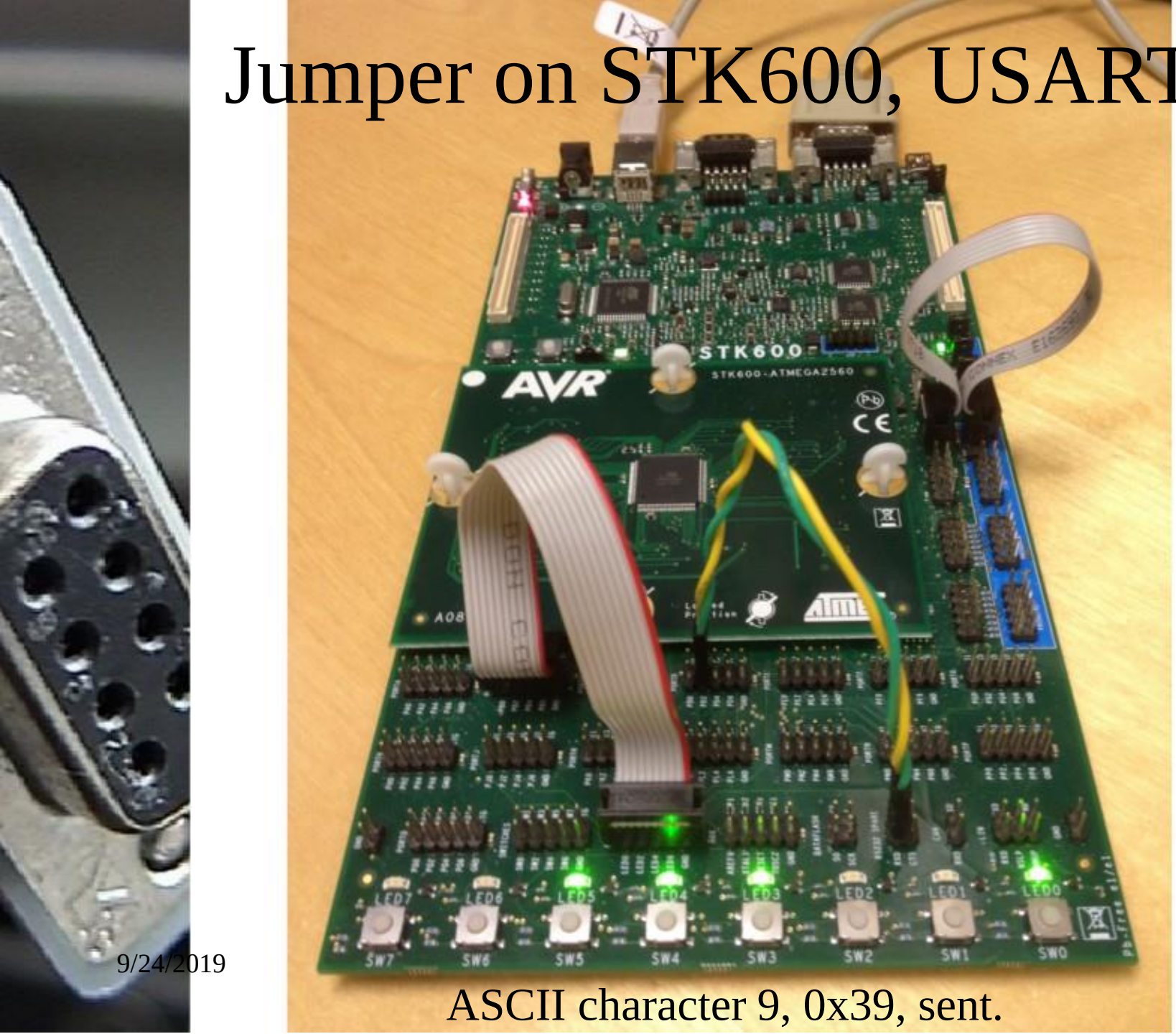
INT2, External Interrupt source 2. The PD2 pin can serve as an External Interrupt source to the MCU.

RXD1, Receive Data (Data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

TX Complete Interrupt Enable
USART Data register Empty Interrupt En...
Receiver Enable
Transmitter Enable
Character Size
Receive Data Bit 8
Transmit Data Bit 8
USART Mode Select — Asynchronous USART
Parity Mode Bits — Disabled

Name	Address	Value	Bits
UBRR1	na (0xCC)	0x000C	
UBRR1H	na (0xCD)	0x00	
UBRR1L	na (0xCC)	0x0C	
UCSR1A	na (0xC8)	0x20	
RXC1		0x00	
TXC1		0x00	
UDRE1		0x01	
FE1		0x00	
DOR1		0x00	
UPE1		0x00	
U2X1		0x00	
MPCM1		0x00	
UCSR1B	na (0xC9)	0x98	
RXCIE1		0x01	
TXCIE1		0x00	
UDRIE1		0x00	
RXEN1		0x01	
TXEN1		0x01	
UCSZ12		0x00	
RXB81		0x00	
TXB81		0x00	
UCSR1C	na (0xCA)	0x06	
UMSEL1		0x00	
UPM1		0x00	
USBS1		0x00	
UCSZ1		0x03	
UCPOL1		0x00	
UDR1	na (0xCE)	0x00	

Jumper on STK600, USART1



ASCII character 9, 0x39, sent.

1ED022, Computer Technology

- Disassembler
- Two’s complement for
  (for negative values stored binary)

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
.INCLUDE "m2560def.inc"
.DEF temp = r16
.DEF Output = r17
.DEF count = r18

.EQU timer_init = 100
.EQU timer_divide = 100

.ORG 0x00
    rjmp restart                ; reset interrupt
.ORG OVF0addr                   ; timer 0 interrupt
    rjmp timer0_int

    .org 0x72
restart:
    ldi temp, LOW(RAMEND)       ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF              ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF              ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b000000101       ; prescaler value to TCCR0
    out TCCR0B, temp            ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)        ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp            ; to register TIMSK
    ldi temp, timer_init        ; starting value for counter
    out TCNT0, temp             ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                   ; timer interrupt routine
    in temp, SREG               ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
restart:
    ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF               ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF               ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b000000101        ; prescaler value to TCCR0
    out TCCR0B, temp             ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)         ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp             ; to register TIMSK
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp              ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                    ; timer interrupt routine
    in temp, SREG                ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

cont:
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp
    pop temp                     ; restore SREG
    out SREG, temp
    pop temp                     ; restore register
;   com Output                   ; 1-complements (inverts) "Output"
    reti                         ; return from interrupt
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
restart:
    ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF               ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF               ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b000000101        ; prescaler value to TCCR0
    out TCCR0B, temp             ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)         ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp             ; to register TIMSK
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp              ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                    ; timer interrupt routine
    in temp, SREG                ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

cont:
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp
    pop temp                     ; restore SREG
    out SREG, temp
    pop temp                     ; restore register
;   com Output                   ; 1-complements (inverts) "Output"
    reti                         ; return from interrupt
```

C:\Document_Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
.INCLUDE "m2560def.inc"
.DEF temp = r16
.DEF Output = r17
.DEF count = r18

.EQU timer_init = 100
.EQU timer_divide = 100

.ORG 0x00
    rjmp restart                ; reset interrupt
.ORG OVF0addr                   ; timer 0 interrupt
    rjmp timer0_int

    .org 0x72
restart:
    ldi temp, LOW(RAMEND)       ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF              ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF              ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b000000101       ; prescaler value to TCCR0
    out TCCR0B, temp            ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)        ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp            ; to register TIMSK
    ldi temp, timer_init        ; starting value for counter
    out TCNT0, temp             ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                   ; timer interrupt routine
    in temp, SREG               ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
```

I/O View

ANALOG_COMPAR

Name	Value
EXTERNAL_INTERRUPT	
JTAG	
PORTA	
PORTB	
Port B Data Register	0x0F
Port B Data Direction Register	0xFF
Port B Input Pins	0x00
PORTC	
PORTD	
PORTE	
PORTF	
PORTG	
PORTH	
PORTJ	
PORTK	
PORTL	
SPI	
TIMER_COUNTER_0	
Timer/Counter0 Output Compare Regis...	0x00
Timer/Counter0 Output Compare Regis...	0x00
Timer/Counter0	0x64

Name	Address	Value	Bits
GTCCR	0x23 (0x43)	0x00	
OCR0A	0x27 (0x47)	0x00	
OCR0B	0x28 (0x48)	0x00	
TCCR0A	0x24 (0x44)	0x00	
TCCR0B	0x25 (0x45)	0x05	
TCNT0	0x26 (0x46)	0x64	
TIFR0	0x15 (0x35)	0x00	
TIMSK0	na (0x6E)	0x01	

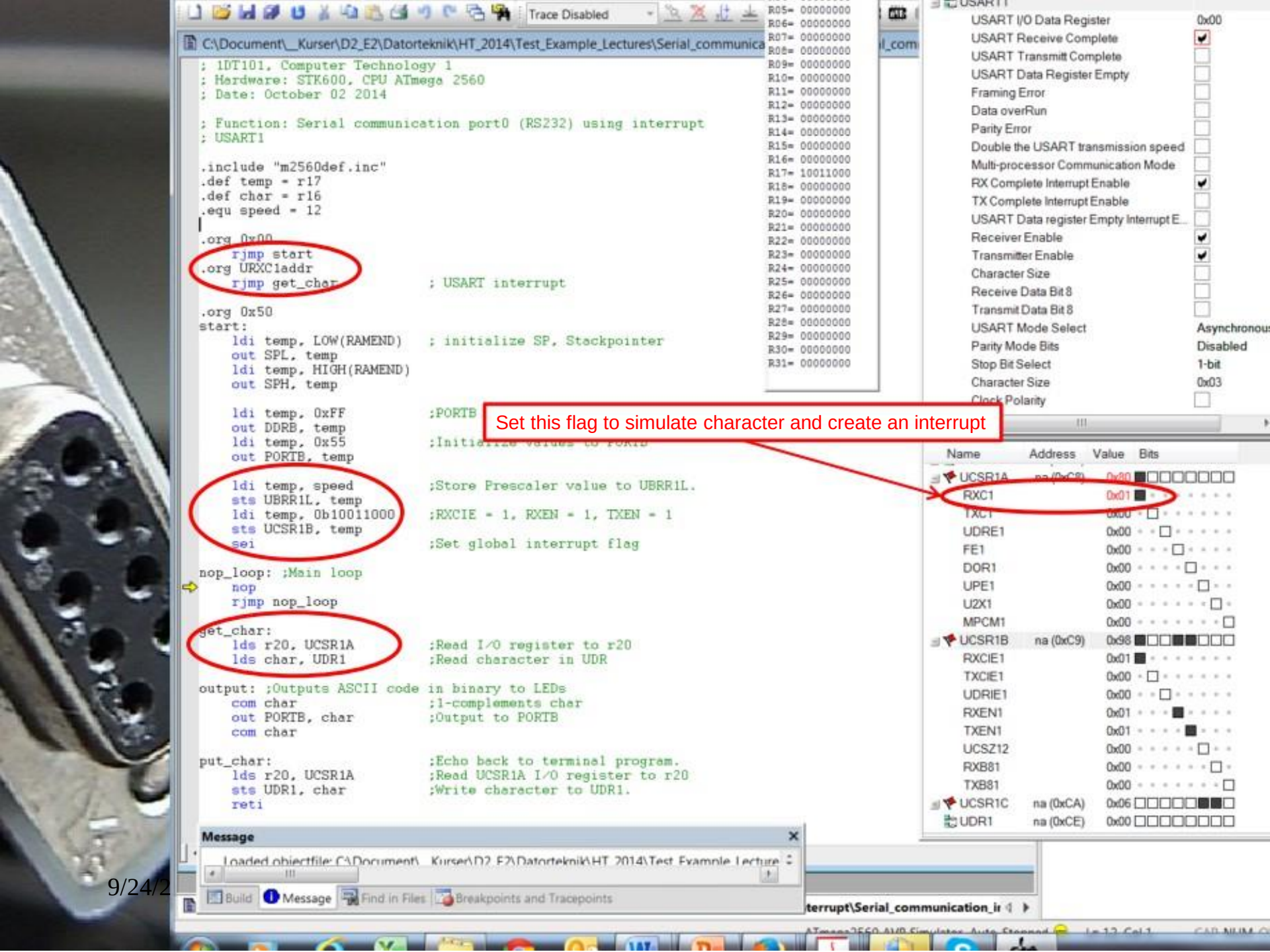
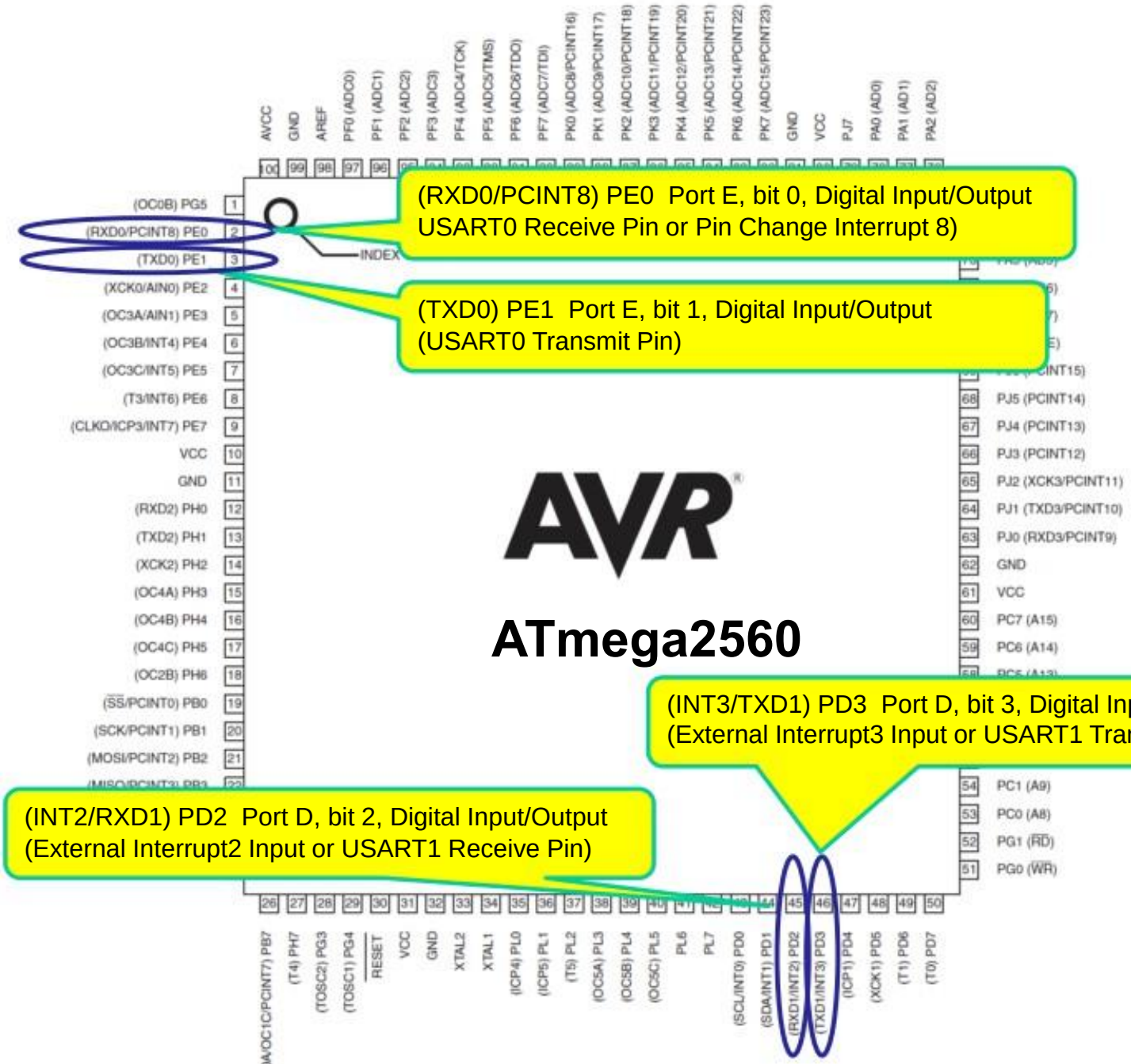

Trace Disabled
C:\Document_Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Serial_communica
; 1DT101, Computer Technology 1
; Hardware: STK600, CPU ATmega 2560
; Date: October 02 2014
; Function: Serial communication port0 (RS232) using interrupt
; USART1
.include "m2560def.inc"
.def temp = r17
.def char = r16
.equ speed = 12
.org 0x00
rjmp start
.org URXC1addr
rjmp get_char ; USART interrupt
.org 0x50
start:
ldi temp, LOW(RAMEND) ; initialize SP, Stackpointer
out SPL, temp
ldi temp, HIGH(RAMEND)
out SPH, temp
ldi temp, 0xFF ;PORTB
out DDRB, temp
ldi temp, 0x55 ;Initialize values to PORTB
out PORTB, temp
ldi temp, speed ;Store Prescaler value to UBRR1L.
sts UBRR1L, temp
ldi temp, 0b10011000 ;RXCIE = 1, RXEN = 1, TXEN = 1
sts UCSR1B, temp
sei ;Set global interrupt flag
nop_loop: ;Main loop
nop
rjmp nop_loop
get_char:
lds r20, UCSR1A ;Read I/O register to r20
lds char, UDR1 ;Read character in UDR
output: ;Outputs ASCII code in binary to LEDs
com char ;1-complements char
out PORTB, char ;Output to PORTB
com char
put_char: ;Echo back to terminal program.
lds r20, UCSR1A ;Read UCSR1A I/O register to r20
sts UDR1, char ;Write character to UDR1.
reti
R05= 00000000
R06= 00000000
R07= 00000000
R08= 00000000
R09= 00000000
R10= 00000000
R11= 00000000
R12= 00000000
R13= 00000000
R14= 00000000
R15= 00000000
R16= 00000000
R17= 10011000
R18= 00000000
R19= 00000000
R20= 00000000
R21= 00000000
R22= 00000000
R23= 00000000
R24= 00000000
R25= 00000000
R26= 00000000
R27= 00000000
R28= 00000000
R29= 00000000
R30= 00000000
R31= 00000000
USART1
USART I/O Data Register 0x00
USART Receive Complete
USART Transmit Complete
USART Data Register Empty
Framing Error
Data overRun
Parity Error
Double the USART transmission speed
Multi-processor Communication Mode
RX Complete Interrupt Enable
TX Complete Interrupt Enable
USART Data register Empty Interrupt E...
Receiver Enable
Transmitter Enable
Character Size
Receive Data Bit 8
Transmit Data Bit 8
USART Mode Select Asynchronous
Parity Mode Bits Disabled
Stop Bit Select 1-bit
Character Size 0x03
Clock Polarity
Set this flag to simulate character and create an interrupt
Name Address Value Bits
UCSR1A na (0xC8) 0x80
RXC1 0x01
TXC1 0x00
UDRE1 0x00
FE1 0x00
DOR1 0x00
UPE1 0x00
U2X1 0x00
MPCM1 0x00
UCSR1B na (0xC9) 0x98
RXCIE1 0x01
TXCIE1 0x00
UDRIE1 0x00
RXEN1 0x01
TXEN1 0x01
UCSZ12 0x00
RXB81 0x00
TXB81 0x00
UCSR1C na (0xCA) 0x06
UDR1 na (0xCE) 0x00
Message
Loaded objectfile: C:\Document\...Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lecture
Build
Message
Find in Files
Breakpoints and Tracepoints
terrupt\Serial_communication_ir
9/24/2

Figure 1-1. TQFP-pinout ATmega640/1280/2560



13.3.4 Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 13-12.

Table 13-12. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	T0 (Timer/Counter0 Clock Input)
PD6	T1 (Timer/Counter1 Clock Input)
PD5	XCK1 (USART1 External Clock Input/Output)
PD4	ICP1 (Timer/Counter1 Input Capture Trigger)
PD3	$\overline{\text{INT3}}$/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)
PD2	$\overline{\text{INT2}}$/RXD1 (External Interrupt2 Input or USART1 Receive Pin)
PD1	$\overline{\text{INT1}}$/SDA (External Interrupt1 Input or TWI Serial DAta)
PD0	$\overline{\text{INT0}}$/SCL (External Interrupt0 Input or TWI Serial CLock)

The alternate pin configuration is as follows:

- **$\overline{\text{INT3}}$/TXD1 – Port D, Bit 3**

INT3, External Interrupt source 3: The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, Transmit Data (Data output pin for the USART1). When the USART1 Transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

- **$\overline{\text{INT2}}$/RXD1 – Port D, Bit 2**

INT2, External Interrupt source 2. The PD2 pin can serve as an External Interrupt source to the MCU.

RXD1, Receive Data (Data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

The Port E pins with alternate functions are shown in Table 13-15.

Table 13-15. Port E Pins Alternate Functions

Port Pin	Alternate Function
PE7	INT7/ICP3/CLK0 (External Interrupt 7 Input, Timer/Counter3 Input Capture Trigger or Divided System Clock)
PE6	INT6/ T3 (External Interrupt 6 Input or Timer/Counter3 Clock Input)
PE5	INT5/OC3C (External Interrupt 5 Input or Output Compare and PWM Output C for Timer/Counter3)
PE4	INT4/OC3B (External Interrupt4 Input or Output Compare and PWM Output B for Timer/Counter3)
PE3	AIN1/OC3A (Analog Comparator Negative Input or Output Compare and PWM Output A for Timer/Counter3)
PE2	AIN0/XCK0 (Analog Comparator Positive Input or USART0 external clock input/output)
PE1	PDO(1)/TXD0 (Programming Data Output or USART0 Transmit Pin)
PE0	PDI(1)/RXD0/PCINT8 (Programming Data Input, USART0 Receive Pin or Pin Change Interrupt 8)

Note: 1. Only for ATmega1281/2561. For ATmega640/1280/2560 these functions are placed on MISO/MOSI pins.

- **PDO/TXD0 – Port E, Bit 1**

PDO, SPI Serial Programming Data Output. During Serial Program Downloading, this pin is used as data output line for the ATmega1281/2561. For ATmega640/1280/2560 this function is placed on MISO.

TXD0, USART0 Transmit pin.

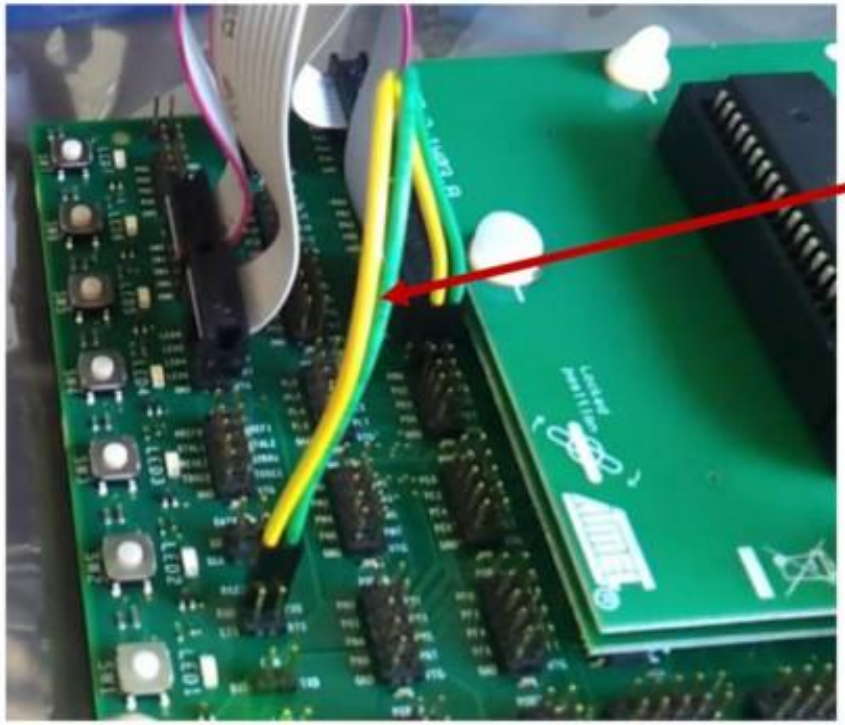
- **PDI/RXD0/PCINT8 – Port E, Bit 0**

PDI, SPI Serial Programming Data Input. During Serial Program Downloading, this pin is used as data input line for the ATmega1281/2561. For ATmega640/1280/2560 this function is placed on MOSI.

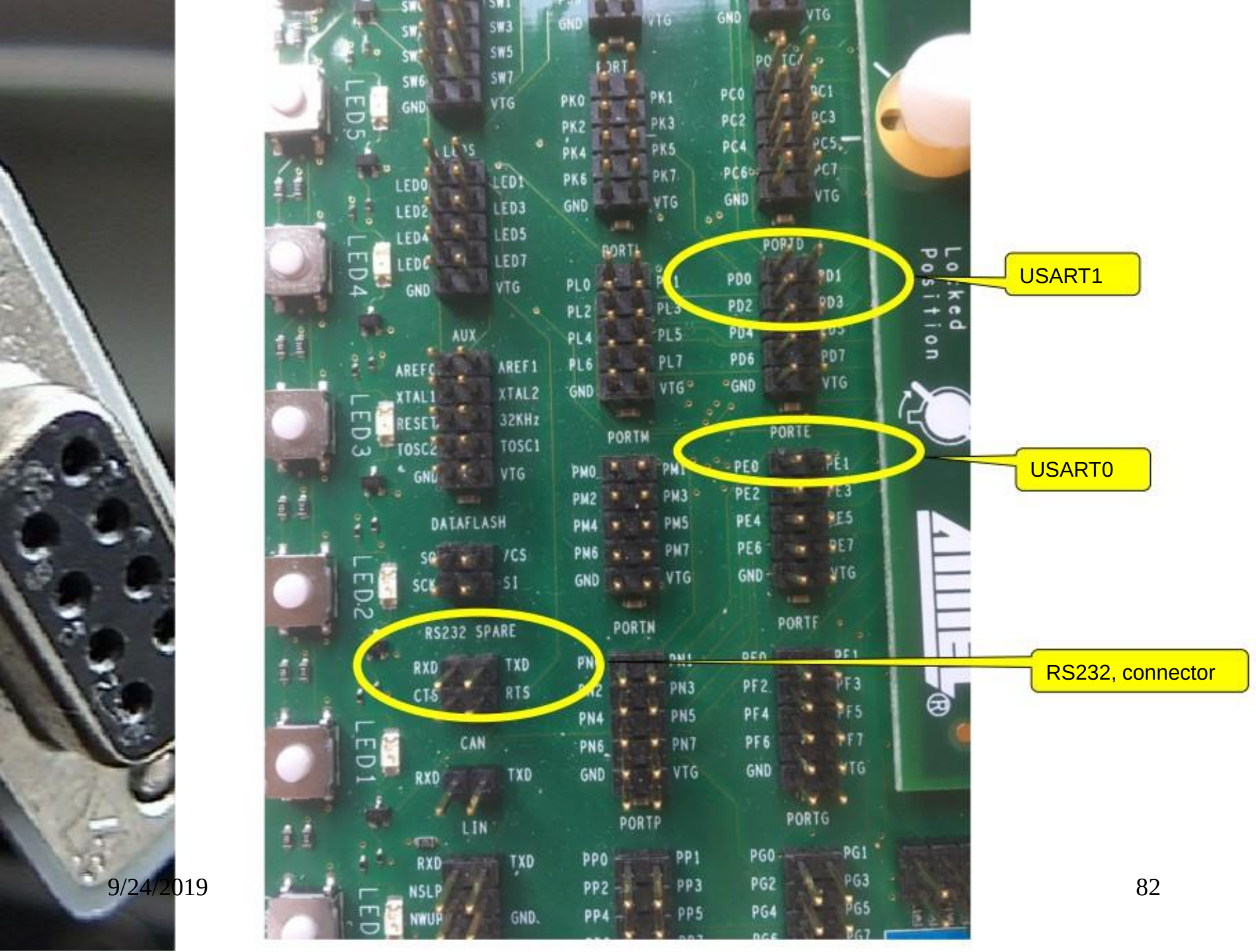
RXD0, USART0 Receive Pin. Receive Data (Data input pin for the USART0). When the USART0 receiver is enabled this pin is configured as an input regardless of the value of DDRE0. When the USART0 forces this pin to be an input, a logical one in PORTE0 will turn on the internal pull-up.

PCINT8, Pin Change Interrupt source 8: The PE0 pin can serve as an external interrupt source.

RS232 serial cable with DB9 connectors. Connect to RS232 SPARE on STK600 and to serial port on PC (COM1).



Connect jumper cable between PE0/PE1 and RXD/TXD – RS232 SPARE.


USART1
USART0
RS232, connector
RS232 SPARE
RXD
TXD
CTS
RTS
DATAFLASH
/CS
SCK
SI
CAN
LIN
PORTD
PORTE
PORTF
PORTG
PORTL
PORTM
PORTN
PORTP
PD0
PD1
PD2
PD3
PE0
PE1
PF2
PF3
PF4
PF5
PF6
PF7
PG0
PG1
PG2
PG3
PG4
PG5
LED0
LED1
LED2
LED3
LED4
LED5
LED7
AUX
AREF0
AREF1
XTAL1
XTAL2
RESET
32KHz
TOSC2
TOSC1
NSLP
NWUP
GND
VTG