ECE3411 – Spring 2019

Lecture 4

UART: Universal Asynchronous Receiver & Transmitter

Sung Yeul Park

Department of Electrical & Computer Engineering

University of Connecticut

Email: sung_yeul.park@uconn.edu

Copied from Lecture 2a, ECE3411 – Fall 2015, by Marten van Dijk and Syed Kamran Haider

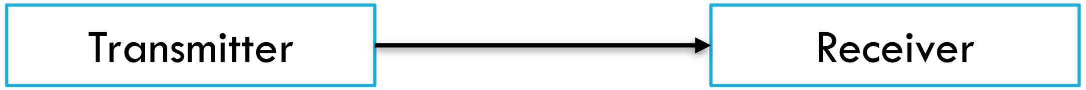
Based on the Atmega328P datasheet and material from Bruce Land's video lectures at Cornel

USART

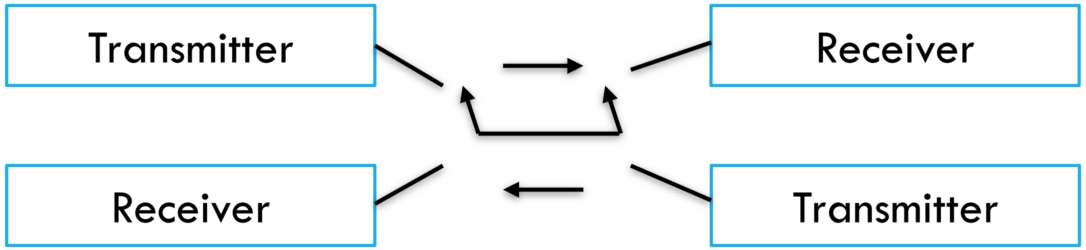
- USART communicates over a 3-wire cable: TX, RX, Gnd
- Designed for modems, a long time ago; protocol is slow
- HW allows full-duplex, i.e., HW can transmit and receive at exactly the same time
- Two Data registers: one for transmitting, the other for receiving data
- Baud rate in bits per second: 9600 Bd is approximately 0.1ms per bit
- The Baud rates of the receiving and transmitting devices need to match within 1.5%
- Each letter is sent one at a time.
- Each character has a unique ASCII value(8bit).
- RS232 protocol is designed to connect two devices together.

Simple, Half, Full-Duplex Data Transfers

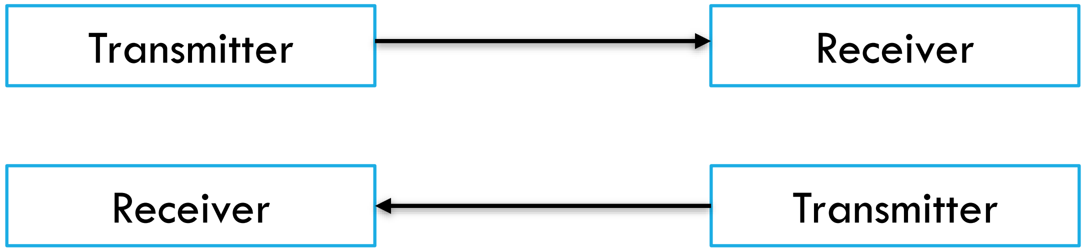
Simplex



Half-Duplex



Full-Duplex



ASCII Table

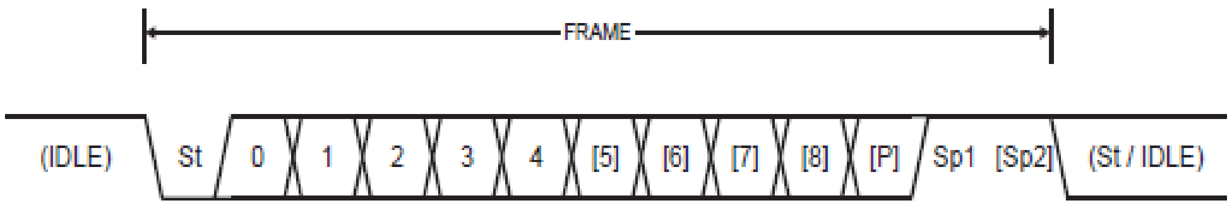
Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end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

Source: www.LookupTables.com

Frame Format

- To transmit a byte (i.e., one char) we need at least one start bit (receiving clock starts when falling edge is received), 8 data bits, and one stop bit: Total of 10 bits.

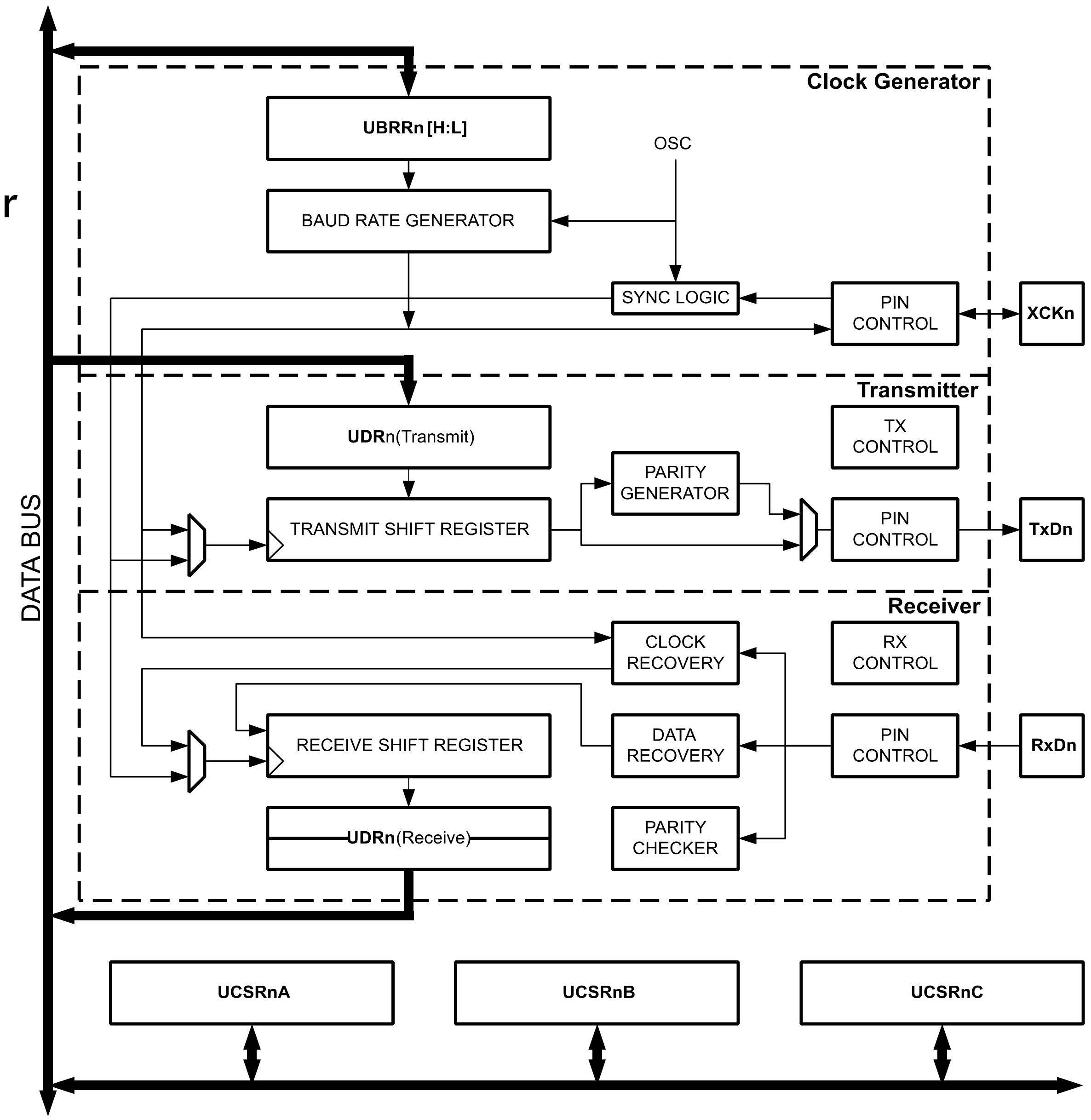
Figure 19-4. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

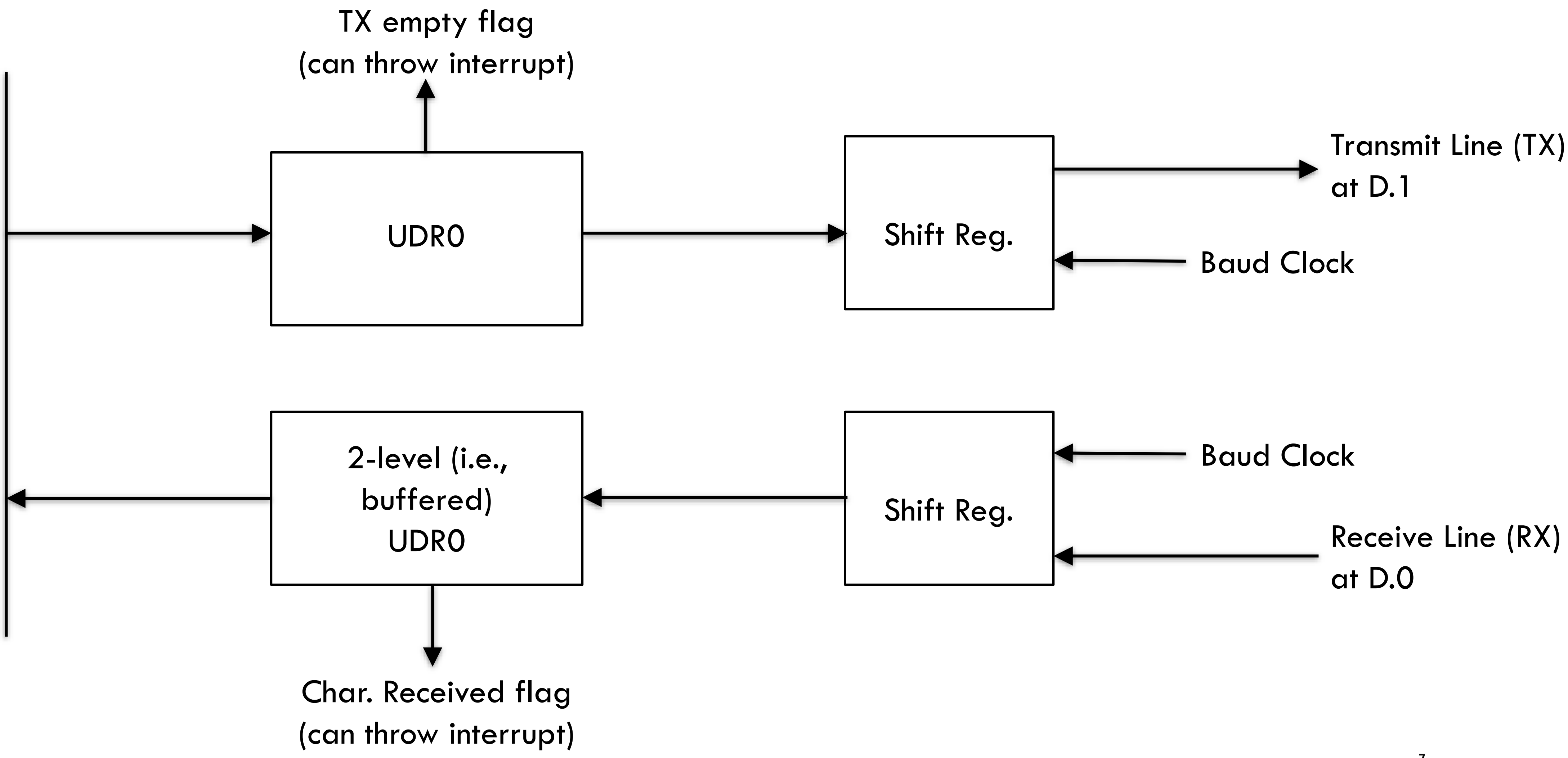
USART0 (Ch. 24 ATmega328PB Datasheet)

- USART = Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Clock generator, Transmitter, Receiver
- Bolted on to the MCU



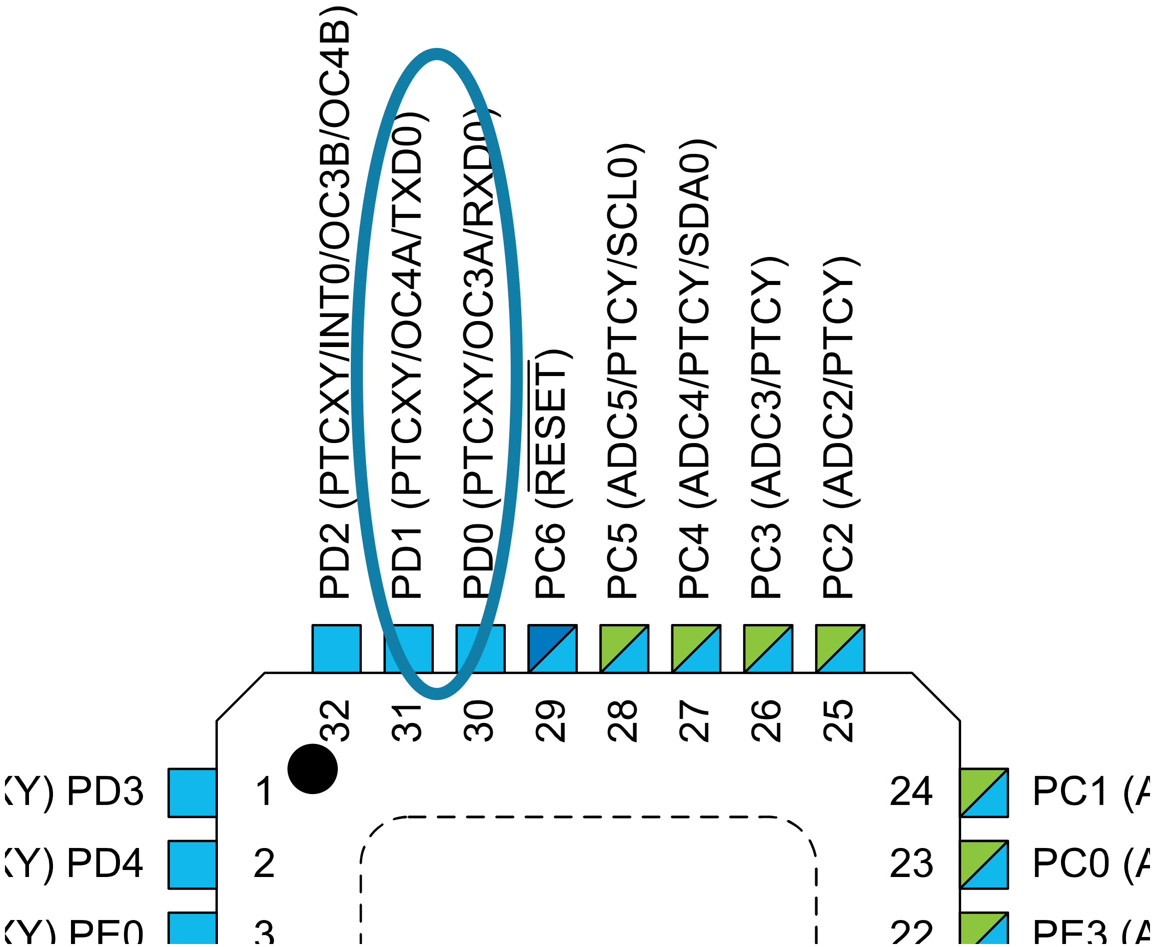
Control registers

USART



TX empty flag
(can throw interrupt)
UDR0
Shift Reg.
Transmit Line (TX)
at D.1
Baud Clock
2-level (i.e.,
buffered)
UDR0
Shift Reg.
Baud Clock
Receive Line (RX)
at D.0
Char. Received flag
(can throw interrupt)

TX and RX at PORTD



UBRR0H and UBRR0L

- Baud rate is translated relative to the system oscillator clock frequency f_OSC to two registers UBRR0H and UBRR0L, the high and low value of UBRR0 which is in the range [0,4095]

Table 24-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate(1)	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0) 16 samples per bit	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{16(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{16\mathrm{BAUD}} - 1$
Asynchronous Double Speed mode (U2Xn = 1) 8 samples per bit	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{8(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{8\mathrm{BAUD}} - 1$
Synchronous Master mode	$\mathrm{BAUD} = \frac{f_{\mathrm{OSC}}}{2(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\mathrm{OSC}}}{2\mathrm{BAUD}} - 1$

UBRR0H and UBRR0L

Baud Rate [bps]	f_{osc} = 16.0000 MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1 Mbps		2 Mbps	

Bit	15	14	13	12	11	10	9	8
					UBRRn[11:8]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	UBRRn[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

UDR0 for Transmission and Receiving

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – TXB / RXB[7:0] USART Transmit / Receive Data Buffer

(The receive and transmit buffers RXB and TXB are different in HW; in SW their names, i.e. I/O addresses, are the same. The shared name UDR0 in read mode means that RXB is read, and UDR0 in write mode means that TXB is written. Notice that reading and writing of bits in UDR0 can be done simultaneously since they affect different hardware buffers!)

Control register: UCSR0A

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

- RXC0: Receive character complete → There is something in the receive register worth reading
- TXC0: Transmit character compare → Is set when both entries in the Transmit Shift Register and Transmit Buffer (UDR0) are shifted out → Not very useful
- UDRE0: Transmit data empty → Goes high when 1 of the two buffers (see above) is empty → Time to refill
- FE0: Frame error if 4 samples of a bit do not match → Detects bad clock rate
- DOR0: Data overrun: If a new character is complete and RXC0 is still set, implies a lost char → SW did not read often enough
- UPE0: Parity error
- U2X0: Double speed (twice the baud rate) → reduces error checking (only 2 samples per bit)
- MPCM0: Multiple processor address mode (can connect more than 2 devices to the line)

Control register: UCSR0B

								Multiprocessor Stuff
Bit	7	6	5	4	3	2	1	0
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

- RXCIE0: Receive character complete interrupt enable → You can write an ISR for this
- TXCIE0: Enables interrupt for both members in TX queue being empty
- UDRIE0: Enables interrupt if the first of the output pipeline is empty
- RXEN0: RX enable → Disables D.0 for general I/O (completely overrides any other I/O)
- TXEN0: TX enable → Disables D.1 for general I/O (completely overrides any other I/O)
- UCSZ02: see next slides

Control register: UCSR0C

Bit	7	6	5	4	3	2	1	0
	UMSELn[1:0]		UPMn[1:0]		USBSn	UCSZn1 / UDORDn	UCSZn0 / UCPHAn	UCPOLn
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Table 24-10. USART Mode Selection

UMSEL[1:0]	Mode
00	Asynchronous USART
01	Synchronous USART
10	Reserved
11	Master SPI (MSPIM)(1)

Table 24-11. USART Mode Selection

UPM[1:0]	ParityMode
00	Disabled
01	Reserved
10	Enabled, Even Parity
11	Enabled, Odd Parity

Table 24-12. Stop Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Control register: UCSR0C

Table 24-13. Character Size Settings

UCSZ1[2:0]	Character Size
000	5-bit
001	6-bit
010	7-bit
011	8-bit
100	Reserved
101	Reserved
110	Reserved
111	9-bit

Initialization

```c
#define F_CPU 16000000UL
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
int main()
{
    …
    UART_Init(MYUBRR);
    …
}

/* Function Body */
void UART_Init(unsigned int ubrr)
{
    UBRR0H = (unsigned char) (ubrr>>8);
    UBRR0L = (unsigned char) ubrr;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
}
```

Transmission (24.6 datasheet)

```
int USART_Transmit(char c)
{
   while ( !(UCSR0A & (1<<UDRE0)) ) ;
   UDR0 = c;

   return 0;
}
```

Receiving

```
unsigned char USART_Receive(void)
{
    /* wait for data to be received */
    while ( !(UCSR0A & (1<<RXC0)) ) ;

    /* get and return received data from buffer */
    return UDR0;
}
```

stdio.h

- `FILE *stdout;`
- `FILE *stdin;`
- `printf(fmt_str, …);` - send string data
- `scanf("%s", buf);` - read string data
- `getchar();` - read one character
- `putchar(c);` - send one character

stdio.h

```
#include <stdio.h>
#include <string.h>
#include "uart.h"

int main()
{
	uart_init();
	while(1) {
		char buf[100];
		...
		scanf("%s", buf);
		if (strcmp(buf, "Hello") == 0){
			...
		}
	}
}
```

Transmission (24.6 datasheet & uart.c)

```c
int uart_putchar(char c, FILE *stream)
{
    /* Alarm (Beep, Bell) */
    if (c == '\a')
    {
      fputs("*ring*\n", stderr);
      return 0;
    }

   /* Newline is translated into a CR */
   if (c == '\n')
        uart_putchar('\r', stream);

   loop_until_bit_is_set(UCSR0A, UDRE0);
   UDR0 = c;

   return 0;
}
```

```c
/* avr/io.h implements useful macros besides
  * defining names for bit positions,
  * registers like DDx etc.
*/

#define _BV(bit) (1 << (bit))
#define bit_is_set(sfr, bit) \
        (_SFR_BYTE(sfr) & _BV(bit))
#define bit_is_clear(sfr, bit) \
        (!(_SFR_BYTE(sfr) & _BV(bit)))
#define loop_until_bit_is_set(sfr, bit) \
        do { } while (bit_is_clear(sfr, bit))
#define loop_until_bit_is_clear(sfr, bit) \
        do { } while (bit_is_set(sfr, bit))
```

Receiving

- int uart_getchar(FILE *stream) in uart.c is a simple line-editor that allows to delete and re-edit the characters entered, until either CR or NL is entered
- printable characters entered will be echoed using uart_putchar()
  - So you can see the character received by the MCU and you can verify whether the transmission was without error if you recognize the character as the transmitted one (as pressed by the keyboard)
- The core part in uart_getchar is

```
int uart_getchar(FILE *stream)
{
   …
   while ( !(UCSR0A & (1<<RXC0)) ) ;
   c = UDR0;
   …
   uart_putchar(c, stream);
   …
}
```

Connections

- Connect the board as in Fig.1. The configuration is same as Lab 3.
- You should disconnect LEDs connected to PD0 and PD1
- PD0 and PD1 are used as Tx and Rx pin for UART. So, you can’t use those for other purposes while using UART. If you don’t disconnect those LEDs, then those could blink/ be turned on while using UART.

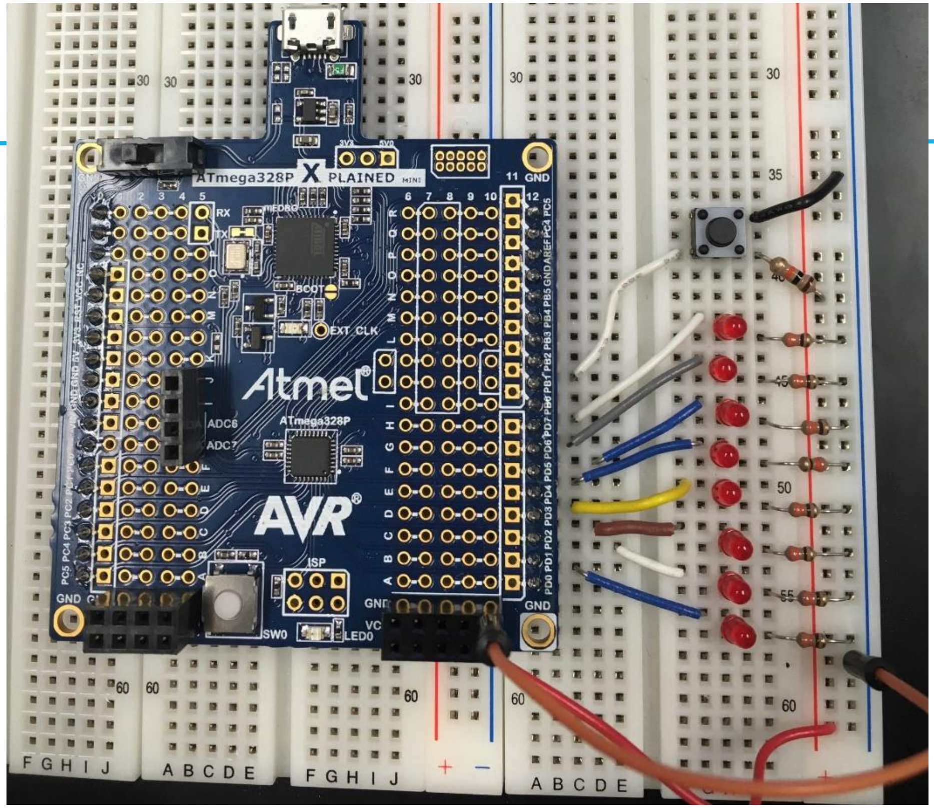


Fig1. Connections for GPIO and UART.

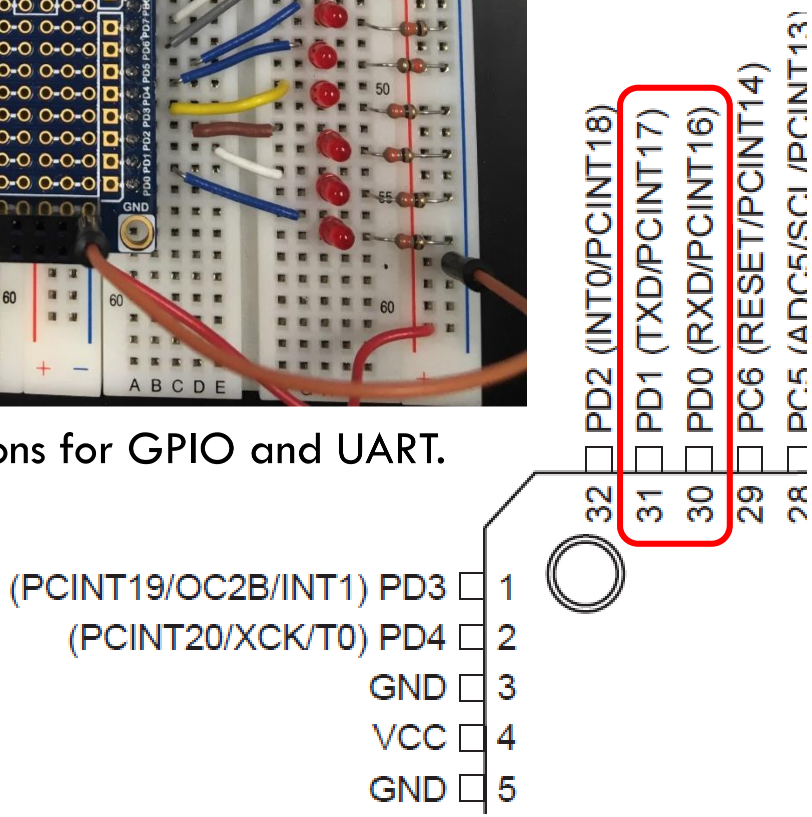
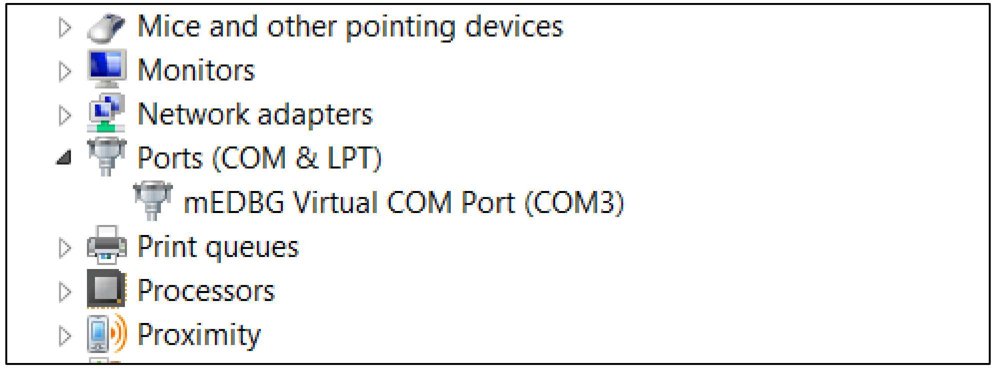


Fig2. UART Tx and Rx Shared with PD0 and PD1

UART Setup: COM Port Identification (1)

In order to setup UART communication between the Xplained mini and your PC, we first need to identify and setup the COM port used by Xplained Mini board

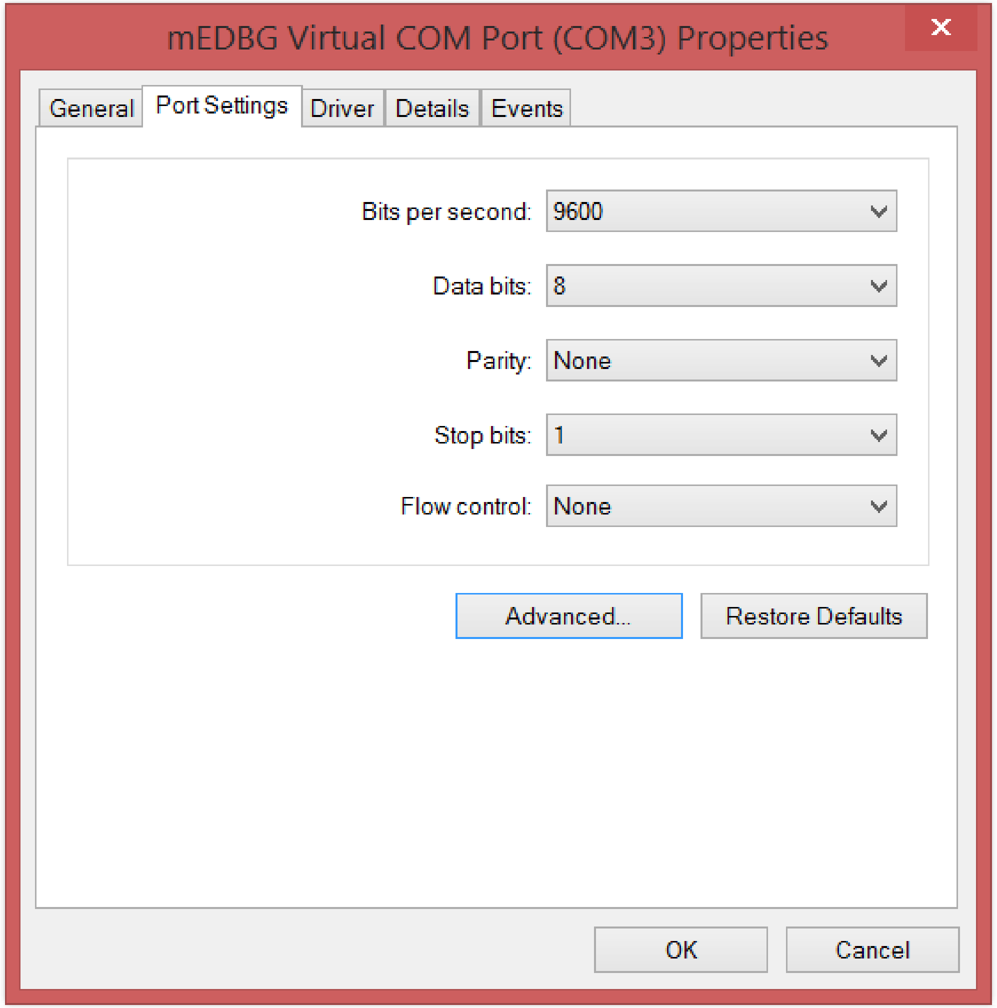
- Connect the Xplained Mini board to your computer via USB cable
- Go to: Control Panel → Device Manager
- Expand the Ports (COM & LPT) section as shown in the figure below.
- Note down the Port number shown against mEDBG Virtual COM Port, i.e. COM3 in the figure below.



Source: Presentation from Marten Van Dijk

UART Setup: COM Port Identification (2)

- Double Click to open the Properties window of mEDBG Virtual COM Port.
- Make sure the Port Settings are the same as shown below.
- If necessary, the COM Port number can be changed under Advanced tab. However, generally the default COM Port number works just fine.

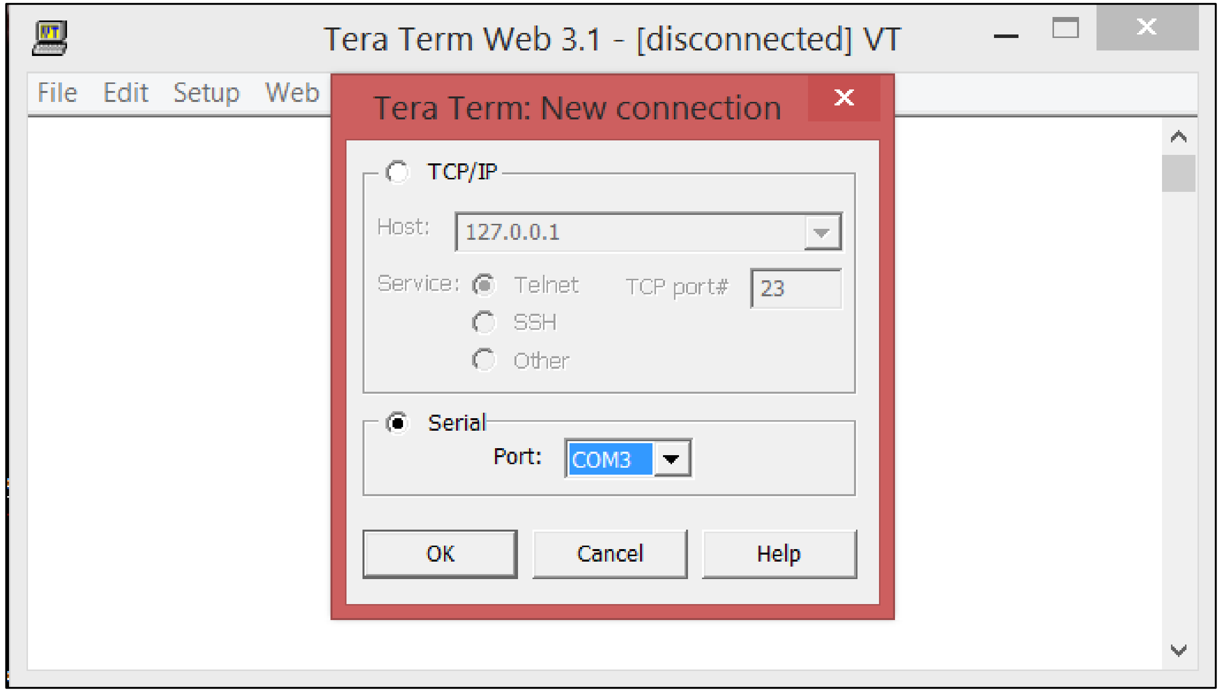


Source: Presentation from Marten Van Dijk

UART Setup: TeraTerm Pro

We will use TeraTerm Pro terminal to send/receive data to Xplained Mini over UART

1. Download ttpro313.zip file posted under Resources on Piazza
2. Unzip the file and run the application ttermpro.exe
3. In the new Connection window, select Serial and select your mEDBG COM Port number, e.g. COM3 (refer to the previous slide) and click OK.



Source: Presentation from Marten Van Dijk

UART Setup: Using uart.h Library

- In order to facilitate you, we provide a library file “uart.c” which defines some useful basic UART functions.
  - “uart.h” and “uart.c” can be downloaded from Piazza under Resources.
  - Files are also available in huskyCT in the folder supplementary files.
- The corresponding prototypes of the functions are declared in “uart.h” file which comes along with “uart.c” file.
- In order to use the function provided by “uart.c”, you need to:
  1. Add “uart.c” and “uart.h” files in your Atmel Studio project source files
  2. Include “uart.h” as a header file in your code, i.e. `#include "uart.h"`

Source: Presentation from Marten Van Dijk

Adding Header and C Files to a Project

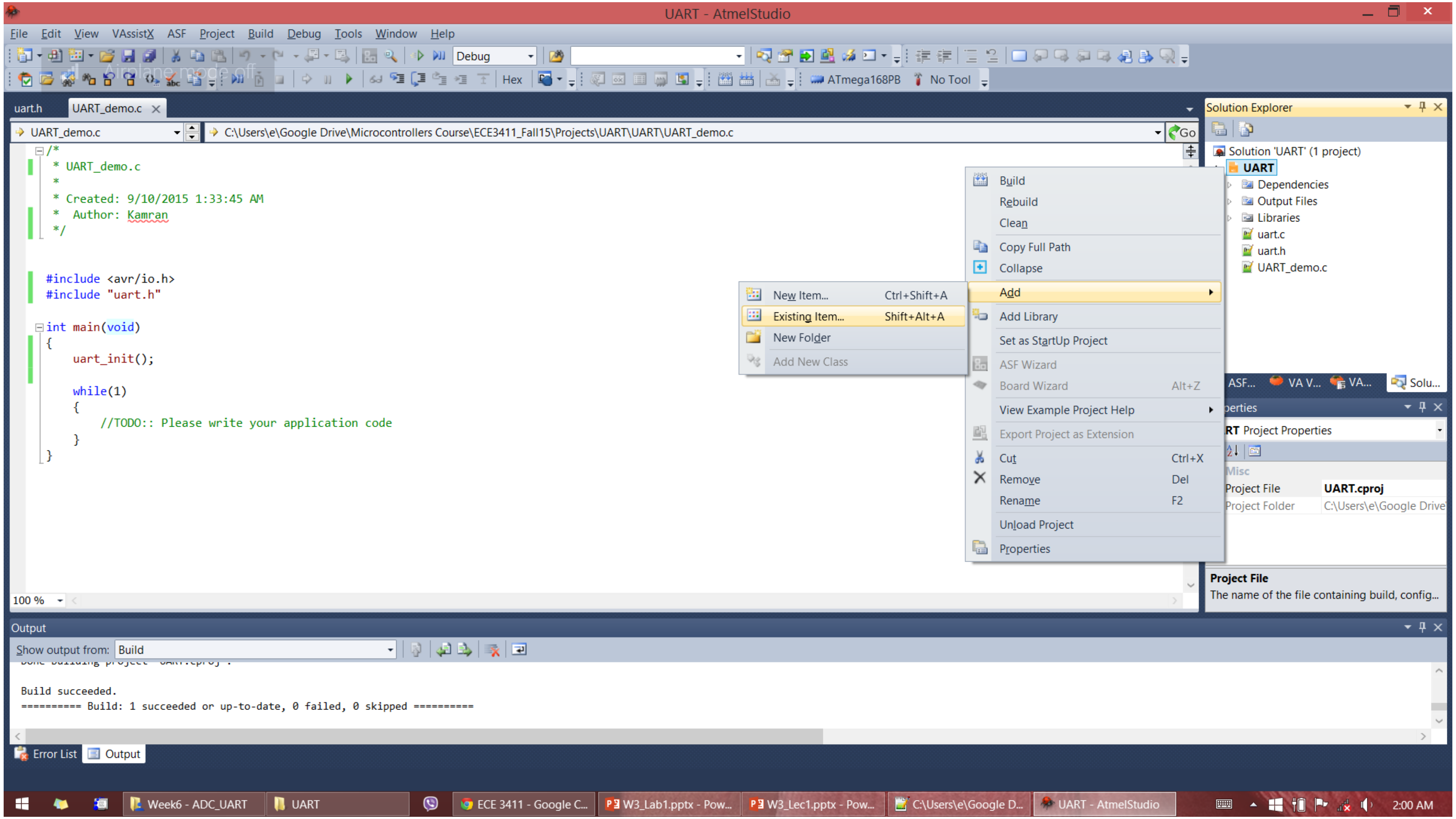
- Often, it is more convenient to include files within your project that contain definitions and functions that you will use frequently.
- This reduces the length of your main c file and eliminates the need for copying and pasting functions you've already written in the past.
- Suppose we want to add "uart.c" and "uart.h" to a project:
  1. Create a new project in Atmel Studio.
  2. Copy the files "uart.c" and "uart.h" into the project directory.
  3. In the 'Solution Explorer' window, right click on the project's name → Add → Existing Item …
  4. Select "uart.c" and "uart.h" and click "Add".
  5. Don't forget to declare/include the header file in you code by calling #include "uart.h"
- See the next few slides for illustration

Source: Presentation from Marten Van Dijk

Using uart.c

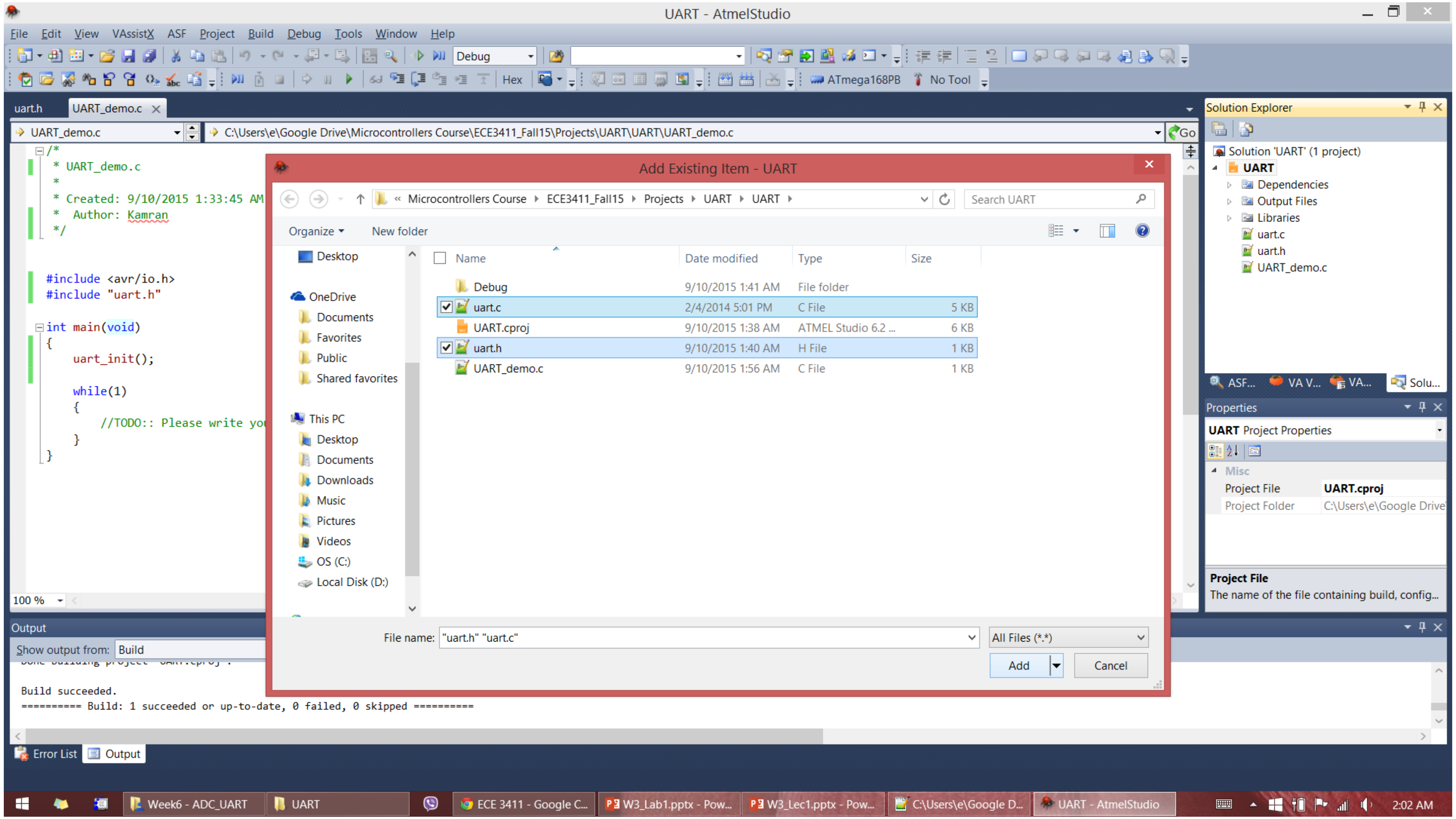
```
#include "uart.h“
...
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
...
int main(void)
{
   uart_init();                                  // Initialize UART
   stdout = stdin = stderr = &uart_str; // Set File outputs to point to UART stream
   ....
   // Can use fprintf and fscanf anywhere: here or in subroutines
   ...
   return 0;
}
```

Adding Header and C Files to a Project



Source: Presentation from Marten Van Dijk

Adding Header and C Files to a Project



Source: Presentation from Marten Van Dijk

Task 1: Blinking a single LED

- Blink a single LED at two different rates based on a push switch.
  - When the switch is not pressed, LED should blink at 2Hz frequency.
  - As long as the switch is pressed, LED should blink at 8Hz frequency.
- The blinking duty cycle should be 50%
  - E.g. for 2Hz frequency, the LED should be on for $1/4^{th}$ of a second, then off for next $1/4^{th}$ of a second and so on.
- You may use the on-board LED and push switch for this task.

Task 2: Changing LED Mode using UART→ Need to demo

Extend Task 1 such that the blinking frequency of the LED can be switched between 2Hz and 8Hz depending upon the string entered from the UART Terminal.

- The LED starts blinking at 2Hz
- After every 10 seconds, the program prints the message on terminal:
  "Do you want to change the LED mode? (Yes/No)"
- If the user enters "Yes", the LED blinking rate switches to the other frequency
  - E.g. if currently the frequency is 2Hz then it switches to 8Hz and vice versa
- If user enters "No" then the frequency stays the same.
- You may use the on-board LED for this task.