ECE3411 – Fall 2017
Lec1a.

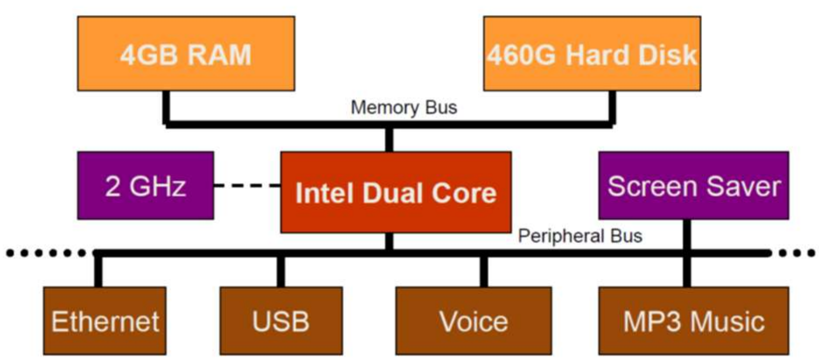
Introduction to Microcontrollers
General Purpose Digital Output

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

UCONN

Copied from Lecture 1b, ECE3411 – Fall 2015,
by Marten van Dijk and Syed Kamran Haider

A Personal Computer

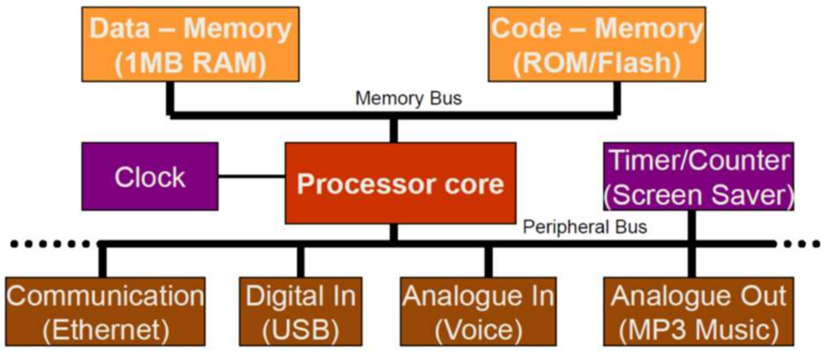


Slide from Sung Yeul Park

2

A Microcontroller

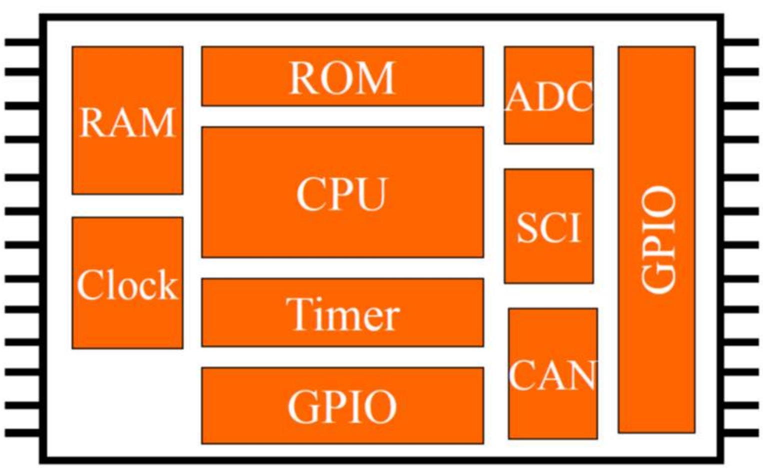
- A Microcontroller contains a processor core, memory and other peripherals on a single chip.



Slide from Sung Yeul Park

3

Microcontroller Structure

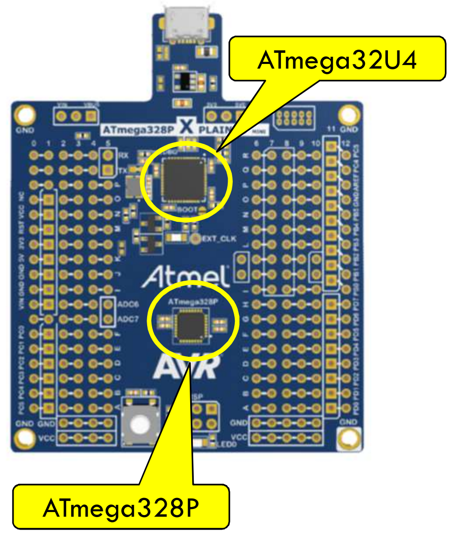


Slide from Sung Yeul Park

4

Atmega328P Xplained Mini Kit

- The ATmega328P Xplained Mini evaluation board provides a development platform for the Atmel ATmega328P Microcontroller.
- Target Microcontroller: ATmega328P
- On-board Programming & Debugging capability using Atmel Studio
  - Programmer Microcontroller: ATmega32U4
- USB connectivity
- Headers & Connectors for accessing target microcontroller's I/O pins



5

ATmega328P Features (1)

- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 32KBytes of In-System Self-Programmable Flash program memory
  - 1K Byte EEPROM
  - 2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security

6

ATmega328P Features (2)

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC with Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Phillips I2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change

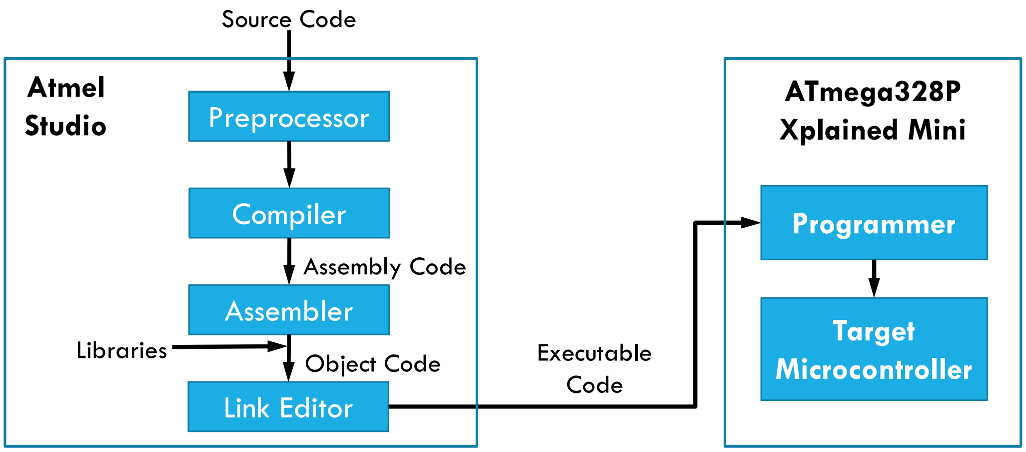
7

ATmega328P Features (3)

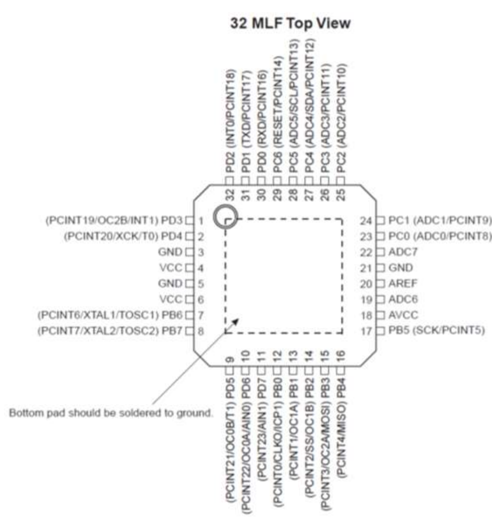
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  - Unique Device ID
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage: 1.8 - 5.5V
- Temperature Range: -40°C to 85°C
- Speed Grade: 0 - 20MHz @ 1.8 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

8

AVR Software Development Process



ATmega328P



Register & Port

- **Register**
  - A collection of flip-flops
  - Simultaneously loaded (written) in parallel or read
  - Interface between users and subsystems
  - Viewed as a software configurable switch

An 8-bit wide Register

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

- **Port**
  - A Port in AVR Microcontrollers represents a bank of pins.
  - A port provides an interface between the central processing unit and the internal and external hardware and software components.
  - E.g. PORTB, PORTC, PORTD etc.

Hardware Registers of a Port

Each Port on the Mega AVRs has three hardware registers associated to it:

- **DDRx** : *Data-Direction Register for Port x*
  - Controls whether each pin is configured for input or output.
  - By default, all pins are configured as inputs.
  - E.g. to enable a pin as output, a '1' is written to its slot in the DDRx.
- **PORTx** : *Port x Data Register*
  - When the DDRx bits are set to '1' (output) for a given pin, the PORT register controls whether that pin is set to logic high or low.
  - E.g. writing a '1' to a bit position in PORT register will produce VCC voltage at that pin & vice versa.
- **PINx** : *Port x Input Pins Address*
  - The PIN register addresses are used to read the digital voltage values for each pin that's configured as input.
  - E.g. a value '0' of a bit of PIN register indicates a low voltage at that pin & vice versa.

Examples of Predefined Registers

- AVR library has some predefined register names for each port.
  - E.g. for Port B, the registers are **DDRB**, **PORTB**, and **PINB**
- These registers can be thought of as regular variables
  - You can read their values in your code
  - You can write values to these registers (except PINx register)
- AVR library also has predefined keywords for each bit position of each port register
  - E.g. for 7th bit position of PINB register, the predefined keyword is **PINB7**
  - Similarly **PORTB5** represents 5th bit position of PORTB register
- Notice that the keywords for bit positions are constants
  - They simply define the bit number, not the bit value. E.g. PORTB5 = 5
  - These keywords are read-only, you cannot write any value to them.

13

Bit Masking Operations

- Bit masking operations allow us to modify a single bit in a register
- Let's say you want to modify bit i in a register called BYTE = 0b01100000
- To Set i^{th} bit → BYTE |= (1 << i);
  - E.g. if $i = 4$ then

    BYTE |= (1 << i) → BYTE = 0b01100000 | 0b00010000 = 0b01110000
- To Clear i^{th} bit → BYTE &= ~(1 << i);
  - E.g. if $i = 6$ then

    BYTE &= ~(1 << i) → BYTE = 0b01110000 & ~(0b01000000)

    BYTE = 0b01110000 & 0b10111111 = 0b00110000
- To Toggle i^{th} bit → BYTE ^= (1 << i);
  - E.g. if $i = 1$ then

    BYTE |= (1 << i) → BYTE = 0b00110000 ^ 0b00000010 = 0b00110010

14

The Structure of AVR C Code

```
[preamble & includes]
[possibly some function definitions]
int main(void){
    [chip initializations]
    while(1) {
        [do this stuff forever]
    }
    return(0);
}
```

- The preamble is where you include information from other files, define global variables, and define functions.
- main() is where the AVR starts executing the code when the power first goes on.
- Any configurations, e.g. configuring I/O pins etc., are done in main() before the **while(1)** loop.
- **while(1)** loop represents the core functionality of the program. It keeps on executing whatever is in the loop body forever (or as long as the AVR is powered).

15

A Simple Test Program

```
#include <avr/io.h>
int main(void)
{
    //configure LED pin as output
    DDRB |= 1<<DDRB5;
    while(1){
        /* check the button status (press - 0 , release - 1 ) */
        if( !( PINB & (1<<PINB7) ) ) {
            /* switch off (0) the LED until key is pressed */
            PORTB &= ~(1<<PORTB5);
        }
        else {
            /* switch on (1) the LED*/
            PORTB |= 1<<PORTB5;
        }
    }
}
```

On Xplained Mini kit,

- LED is connected to 5th pin of Port B
- Switch is connected to 7th pin of Port B

(!(EXPRESSION)) means (EXPRESSION == 0)

- Values for PINB & (1 <<PINB7) are 0=0b00000000 or 128=0b10000000

16

The Delay Library

- AVR supports a delay library to introduce delay between the execution of two code statements.
  - <util/delay.h> header file needs to be included in the code
- The delay library provides two functions
  - _delay_us(x) for introducing a delay of x microseconds
  - _delay_ms(x) for introducing a delay of x milliseconds
- <util/delay.h> library needs to know the Microcontroller's clock frequency for accurate time measurements
  - Clock frequency is defined by defining F_CPU in the code
- Xplained Mini kit runs the ATmega169PB on 16MHz frequency
  - #define F_CPU 16000000UL is included in the code to define the frequency for the delay library
- Only use delay functionality in order to define access functionality for e.g. LCD screen which requires precise timing sequences:
  - Never use delay functionality in your main program
  - We want to do other useful computation while waiting

Test Program to Blink LED

```
// ------- Preamble -------- //
#define F_CPU 16000000UL /* Tells the Clock Freq to the Compiler. */
#include <avr/io.h>          /* Defines pins, ports etc. */
#include <util/delay.h>      /* Functions to waste time */
int main(void) {
  // -------- Inits --------- //
  /* Data Direction Register B: writing a one to the bit enables output. */
        DDRB |= (1 << DDRB5);
  // ------ Event loop ------ //
  while (1) {
    PORTB = 0b00100000;  /* Turn on the LED bit/pin in PORTB */
    _delay_ms(1000);      /* wait for 1 second */
    PORTB = 0b00000000;  /* Turn off all B pins, including LED */
    _delay_ms(1000);      /* wait for 1 second */
  } /* End event loop */
  return (0); /* This line is never reached */
}
```

ECE3411 – Fall 2017
Lab1a.

AVR Board Setup
General Purpose Digital Output

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

UCONN

Adopted from Lab 2a slides "AVR Board Setup General Purpose Digital Output" by Marten van Dijk and Syed Kamran Haider, Fall 2015.

Development Board Setup

Development Board Setup has three steps

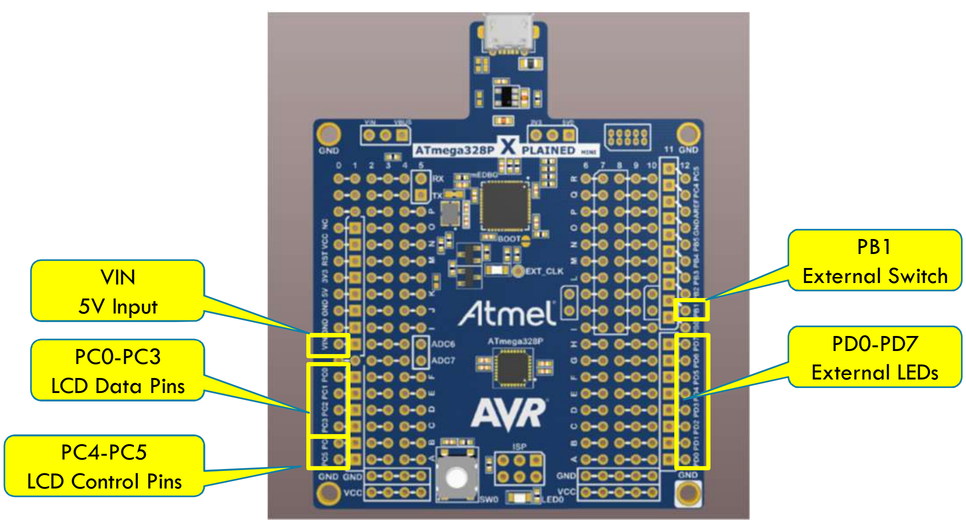
1. Soldering connectors for Xplained Mini kit
2. Soldering connectors for LCD
3. Putting everything together on the breadboard

2

Basics of Soldering

1. Heat the iron to 750F.
2. The LED will stop blinking once the iron has reached the desired temperature.
3. Heat the pad briefly.
4. With the iron sitting on the pad, push solder into the tip of the soldering iron.

ATmega328P Xplained Mini Pin Allocation



Initial board setup

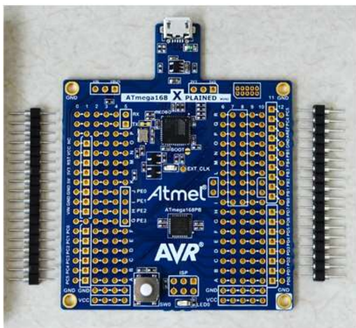
- Setup Atmel studio
  - Atmel Studio is available for download at the following link: http://www.atmel.com/tools/ATMELSTUDIO.aspx
  - You need to download "**Atmel Studio 6.2 sp2 (build 1563) Installer"** which is the first one in the list of available downloads
- As general guidelines for installation and getting familiar with Atmel Studio, please follow the Getting_Started_with_ATmega168PB_Application_Note.pdf document (from page 7 onward) posted under General Resources section.
  - Although this document targets ATmega168PB Xplained Mini kit, the exact same steps apply for ATmega328P Xplained Mini kit.
- Before you start soldering the board make sure the board is working fine.
  - Get the test code provided on the next slide working for your board.

Test code

```
#include <avr/io.h>
int main(void)
{
//configure LED pin as output
DDRB |= 1<<DDB5;
   while(1)
   {
      /* check the button status (press - 0 , release - 1 ) */
      if(!(PINB & (1<<PINB7))) {
            /* switch off (0) the LED until key is pressed */
            PORTB &= ~(1<<PORTB5);
      }
      else {
            /* switch on (1) the LED*/
            PORTB |= 1<<PORTB5;
      }
   } return 0;
}
```

Soldering connectors for Xplained Mini kit

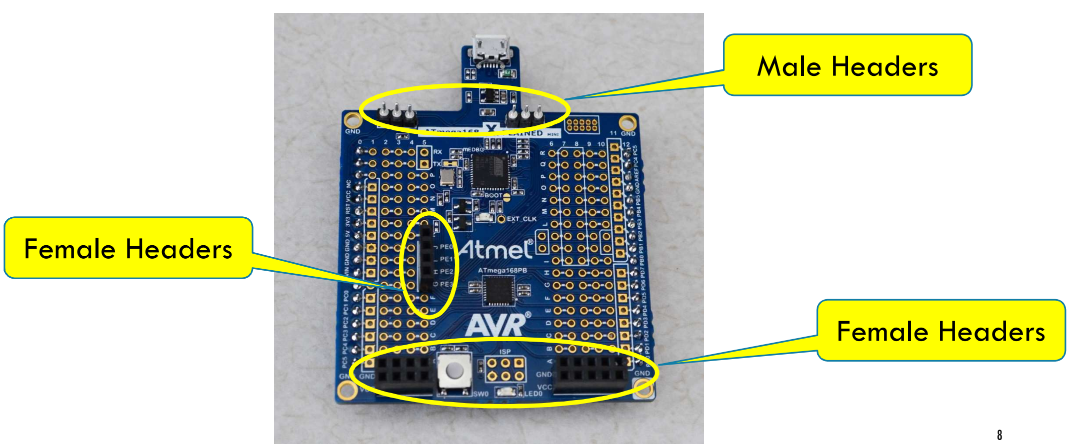
- Take 2 male headers each of 18-pins.
- Insert the thin side of the headers to outermost ports on both left and right side as shown in the bottom view of Xplained Mini.
- Solder the headers to the Xplained Mini pads from the top.



7

Soldering connectors for Xplained Mini kit

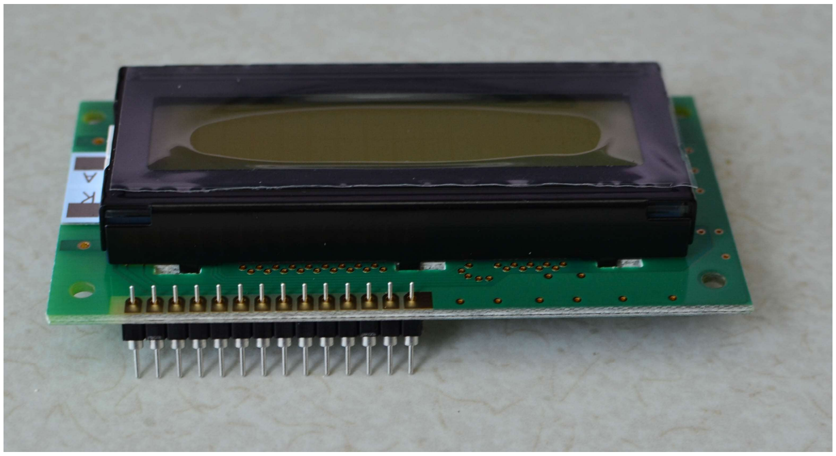
- Insert two 3-pin male headers from the top as shown, and solder from the bottom.
- Similarly Insert the three female headers from the top and solder from the bottom.



8

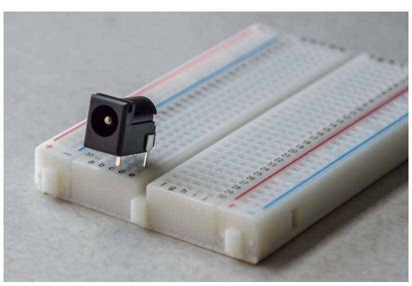
Soldering the Connectors for LCD

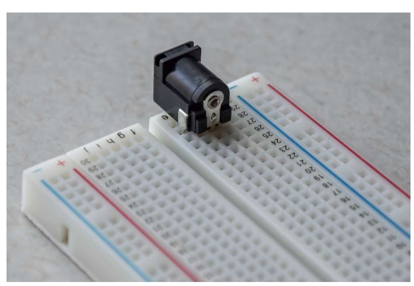
- Insert a 14-pin male header in LCD pads from the bottom and solder from the top.



Wiring the Breadboard (1)

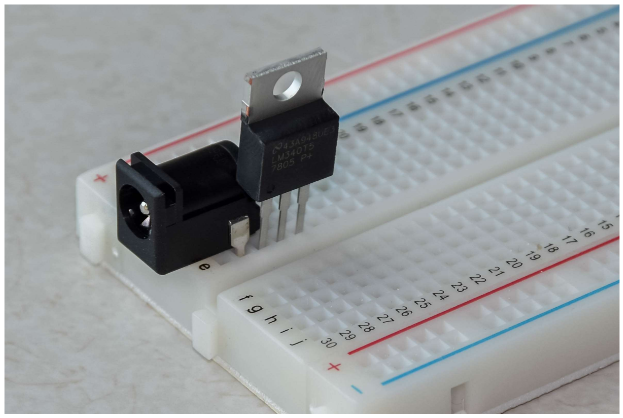
- Start with inserting the DC Power Jack pins into rows 28, 29, 30 and columns 'c' and 'e'.





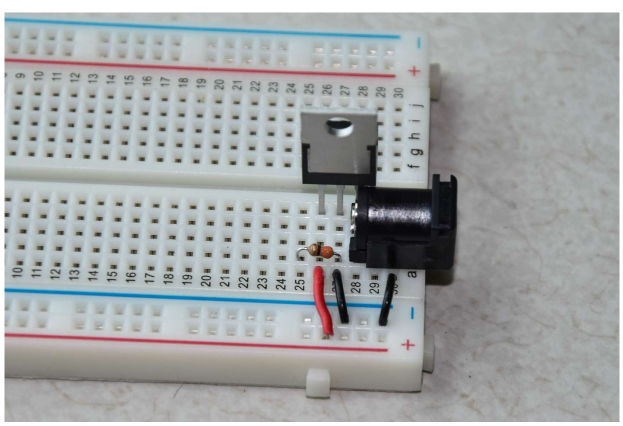
Wiring the Breadboard (2)

- Insert the 5V Regulator (7805) into rows 26, 27, 28 and column 'e' EXACTLY as shown in the figure.



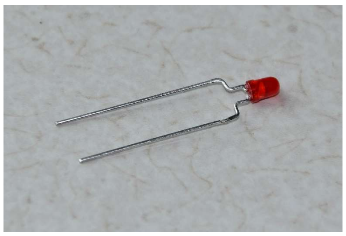
Wiring the Breadboard (3)

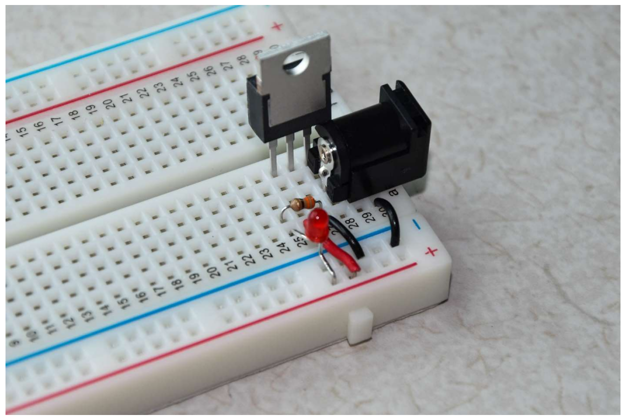
- Connect a 330 Ohm resistance and VCC (+) and GND (-) wires as shown in the figure.



Wiring the Breadboard (4)

- Insert long end of a LED in VCC (+) and short end in row 25.
- This LED is lit up whenever power is supplied to the board.

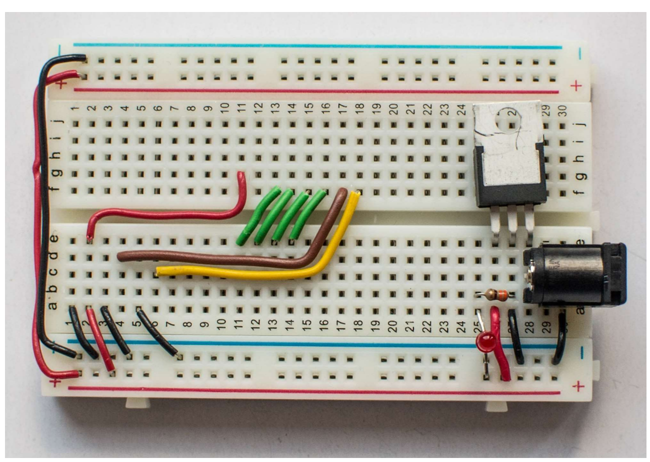




13

Wiring the Breadboard (5)

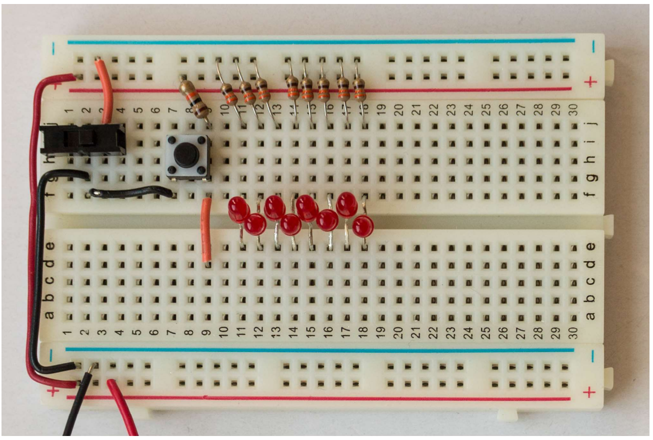
- Connect the rest of the wires as shown in the figure.



14

Wiring the Breadboard (6)

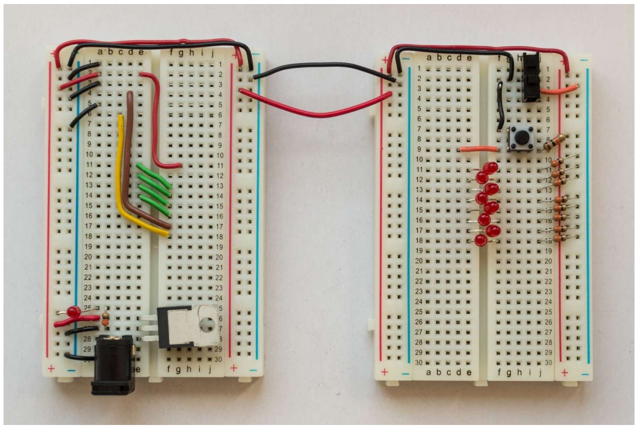
- On the second breadboard, connect eight LEDs, eight 330 Ohm resistors, 10 kOhm resistor, push switch, slider switch and other wires as shown in the figure.



15

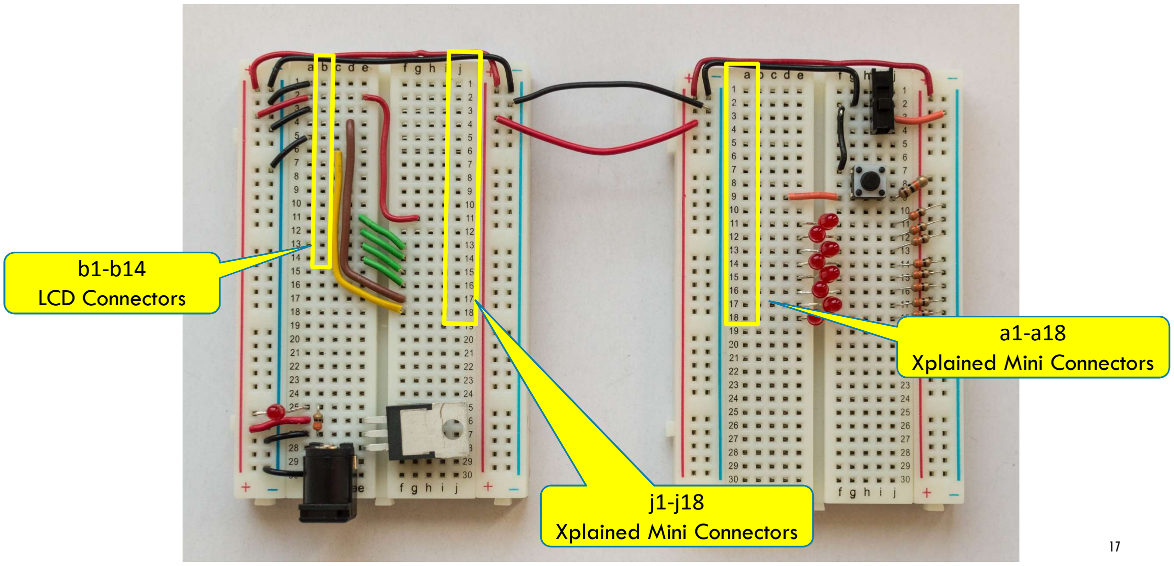
Wiring the Breadboard (7)

- Connect the two breadboards together by supplying VCC and GND from the left board to the right one.



16

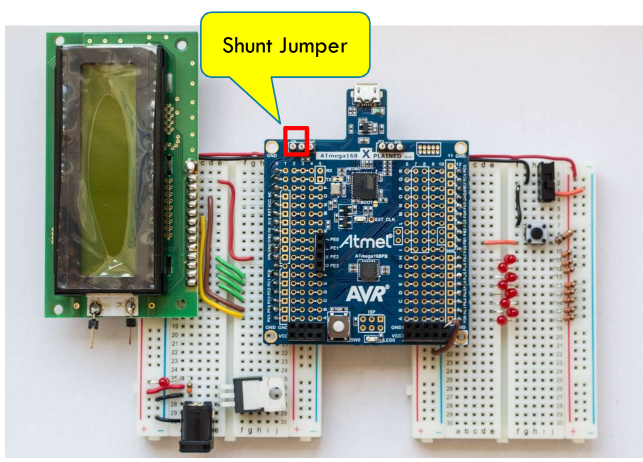
Wiring the Breadboard (8)



17

Putting everything together…

- Insert the left headers of Xplained Mini into column 'j' & rows 1-18 of left breadboard.
- Insert the right headers of Xplained Mini into column 'a' & rows 1-18 of right breadboard.
- Position the LCD outward and Insert its headers into column 'b' & rows 1-14 of left breadboard.
- Connect the right female GND header of Xplained Mini with the right breadboard's ground (-).
- Put a Shunt-Jumper to short the two pins indicated in order to power up the board using external adapter.



18

Test Code

```
// ------- Preamble -------- //
#define F_CPU 16000000UL /* Tells the Clock Freq to the Compiler. */
#include <avr/io.h>           /* Defines pins, ports etc. */
#include <util/delay.h>       /* Functions to waste time */
int main(void) {
  // -------- Inits --------- //
  /* Data Direction Register D: Setting Port D as output. */
        DDRD = 0b11111111;
  // ------ Event loop ------ //
  while (1) {
    PORTD = 0b01010101;  /* Turn on alternate LEDs in PORTD */
    _delay_ms(1000);         /* wait for 1 second */
    PORTD = 0b10101010;  /* Toggle the LEDs */
    _delay_ms(1000);         /* wait for 1 second */
  } /* End event loop */
  return (0); /* This line is never reached */
}
```


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 1/4th of a second, then off for next 1/4th of a second and so on.
- You may use the on-board LED and push switch for this task.

Task 2: Blinking 8 LEDs one after another

Extend the Task1 with another switch which activates the blinking to loop through all 8 LEDs one after another.

- When the system starts, LED 0 is active and blinks at 2Hz.
- As long as switch 1 is pressed, the currently active LED blinks at 8Hz. Otherwise it blinks at 2Hz.
- As long as switch 2 is pressed, the currently active LED keeps shifting towards left at the frequency depending upon the position of switch 1, and starts from 0 again.
  - E.g. if LED 0 is active currently, pressing switch 2 shifts the blinking to LED 1, 2, 3, … , 7 and then again LED 0 and so on.
- When switch 2 is released, the last active LED should keep blinking without anymore shifting.

ECE3411 – Fall 2017
Lec1b.

UART: Universal Asynchronous Receiver & Transmitter

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

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

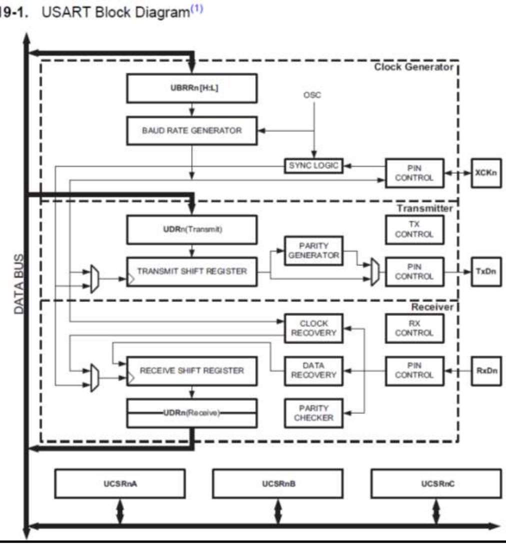
Based on the Atmega328P datasheet and material
from Bruce Land's video lectures at Cornel

UCONN

USART0 (Ch. 19 ATmega328P Datasheet)

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

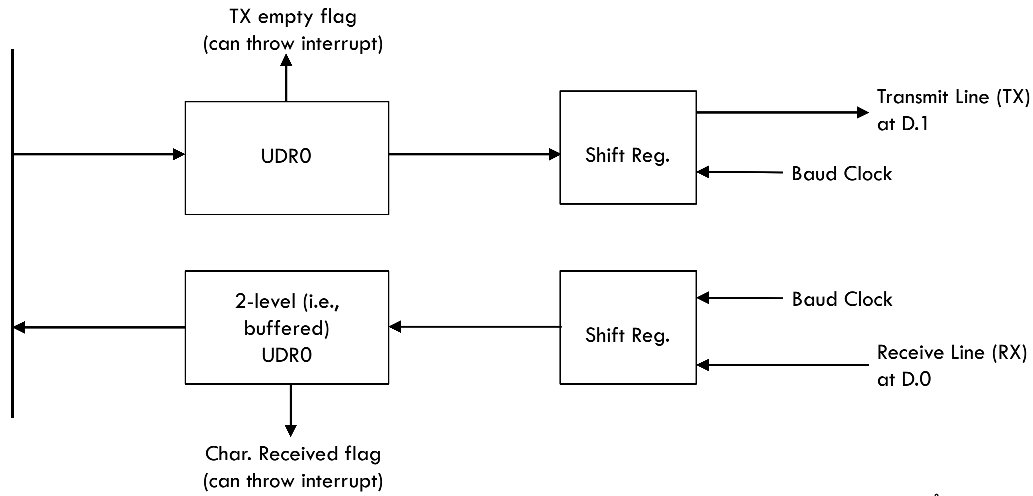
Figure 19-1. USART Block Diagram[1]



Control
registers

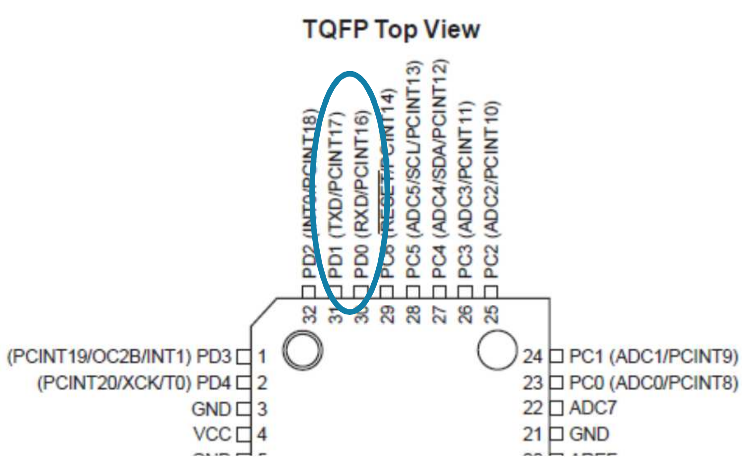
2

USART



3

TX and RX at PORTD



4

USART

- USART communicates over a 3-wire cable: TX, RX, Gnd
- Designed for a mechanical printer, a long time ago; protocol is slow
- HW allows full-duplex, i.e., HW can transmit and receive at exactly the same time
  - Need interrupt to utilize this in SW
- Baud rate in bits per second: 9600 Bd is approximately 0.1ms per bit
  - This is slow: Therefore, in SW start transmitting a character, then do something else!
  - In theory the Baud rate can be very large (1Mbit per second) but this can only be realized between MCUs
  - The used cable limits the maximum possible Baud rate
- Per bit the receiving clock makes 4 measurements and they all need to match: All, e.g. 10, bits within a frame give 40 measurements that all need to match
  - The Baud rates of the receiving and transmitting devices need to match within 1/40 = 2.5%

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 19-1. 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) 4 samples per bit	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1) 2 samples per bit	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

UBRR0H and UBRR0L

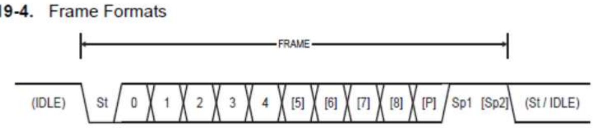
19.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	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	

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.

UDR0 for Transmission and Receiving

19.10.1 UDRn – USART I/O Data Register n

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (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 UDRn. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

(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!)

9

Control register: UCR0A

19.10.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	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

10

Control register: UCR0A

19.10.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- 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)

11

Control register: UCSR0B

Multiprocessor Stuff

19.10.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
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	

- 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

12

Control register: UCSR0C

19.10.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 19-4.

Table 19-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

13

Control register: UCSR0C

Table 19-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Table 19-6. USBS Bit Settings

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

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	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

Default: Frames of 10 bits.

Table 19-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

14

Initialization

```
#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);
}
```

15

Transmission (19.6.1 datasheet & uart.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 Carriage Return */
   if (c == '\n')  {uart_putchar('\r', stream); return 0;}

   /* In uart.c: loop_until_bit_is_set(UCSR0A, UDRE0); */
   while ( !(UCSR0A & (1<<UDRE0)) ) ;
   UDR0 = c;

   return 0;
}
```

```
/* avr/io.h implements useful macros besides defining
  * names for bit positions, registers like DDx  (or do we
  * use DDRx?) 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))
```

16

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);
   …
}
```

17

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

18

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;
}
```

ECE3411 – Fall 2017
Lab1b.

UART: Universal Asynchronous Receiver & Transmitter

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

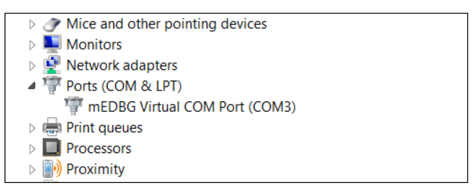
Copied from Lab 2b, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

UCONN

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

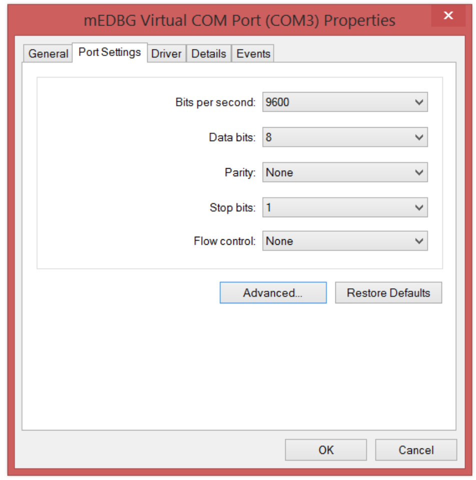
- Connect the Xplained Mini board to your computer via USB cable
- Go to: Control Panel → Device Manager
- Expand the Ports (COM & LPT) section as shown in the figure below.
- Note down the Port number shown against mEDBG Virtual COM Port, i.e. COM3 in the figure below.



2

UART Setup: COM Port Identification (1)

- 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.

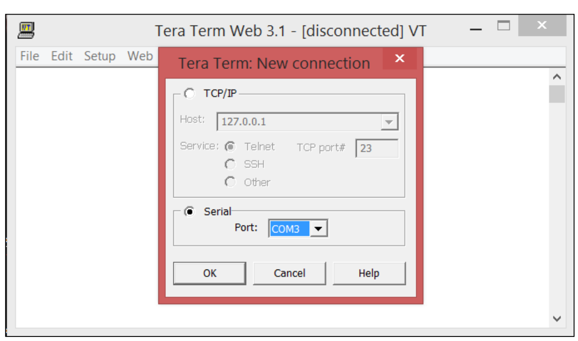


3

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.



4

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.
- 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"

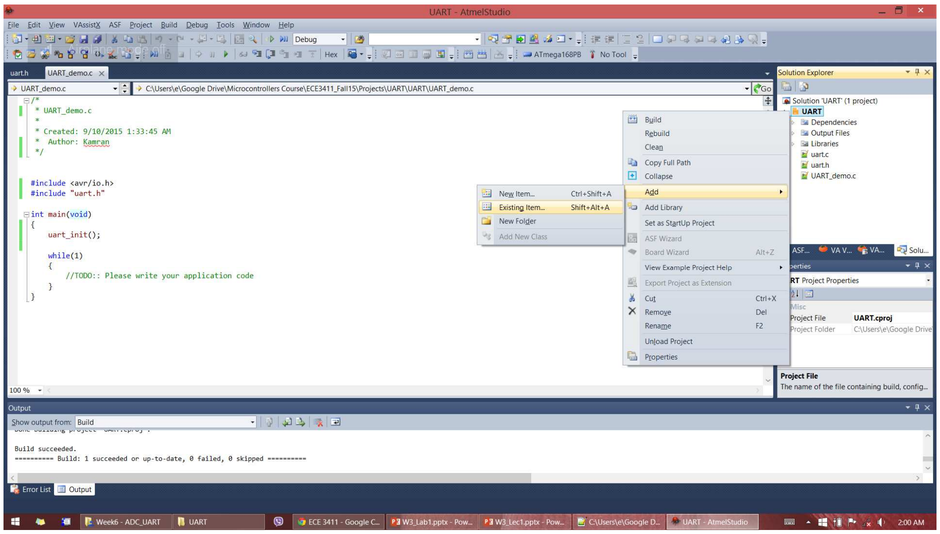
5

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

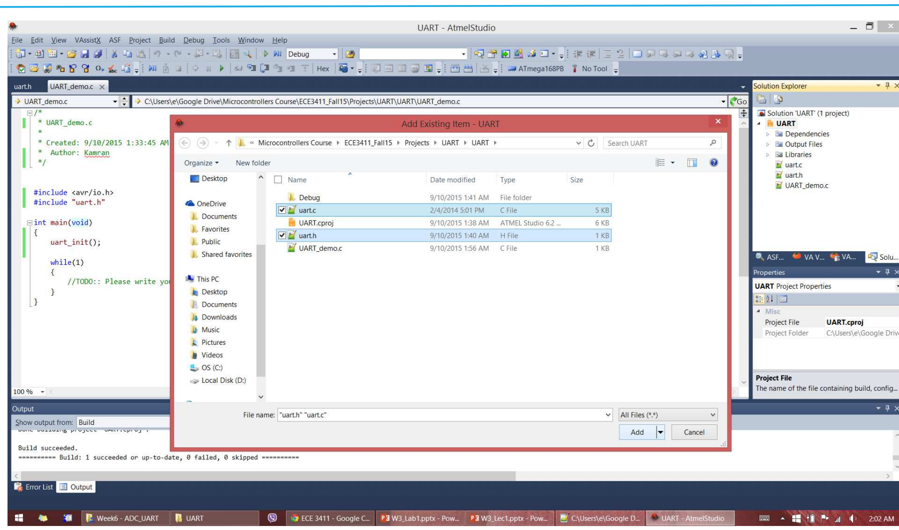
6

Adding Header and C Files to a Project



7

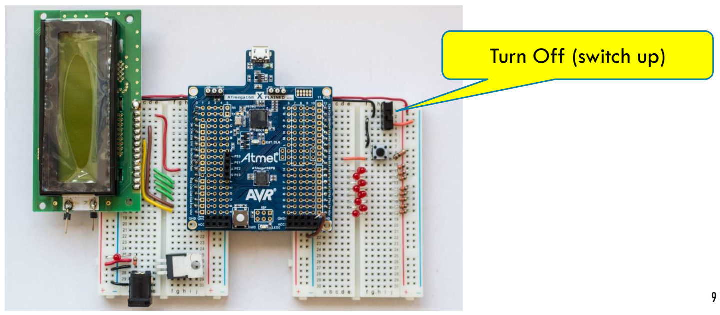
Adding Header and C Files to a Project



8

Running a UART Test Program (1)

- Connect the board to your computer and create a new project in Atmel Studio
- Include the "uart.h" and "uart.c" files in your project as described earlier
- Notice that PD0 and PD1 serve as UART RXD and TXD pins
  - **Hence these pins should not be used for any other purpose when using UART**
  - **Therefore make sure to turn off the switch to disconnect LEDs from Ground**



9

Running a UART Test Program (2)

- Open TeraTerm Pro terminal and create a connection as described earlier
- Copy the test program given on the next slide, compile it and run.
- You should see a "Hello!" message on TeraTerm window
  - Notice that you need to create the TeraTerm connection before running the program to see this greeting message.
- The test program simply sends back the string which it receives from the terminal
- Try writing some small strings, and you should see it printed out on the terminal once you hit Enter key.

10

UART Test Program

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include "uart.h"

// File stream for UART. Used for Transmission to demonstrate the fprintf function.
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

char rec[50];           // Declare a character buffer
int main(void)
{
        uart_init();                // Initialize UART
        stdout = stdin = stderr = &uart_str; // Set File outputs to point to UART stream
        fprintf(stdout, "Hello! \n");

        while(1){
                fscanf(stdin, "%s", rec);
                printf("Received: \n");
                fprintf(stdout, "%s \n", rec);

    }
}
```


Task: Changing LED Mode using UART

Extend Lab1a Task 1 such that the blinking frequency of the LED can be switched between 2Hz and 8Hz depending upon the string entered from the 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.

ECE3411 – Fall 2017
Lec1c.

General Purpose Digital Input
LCD Interfacing

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lecture 2b, 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

Ports and their control registers

- I/O ports are labelled B, C, D: special functions are set up for each
- Can set any bit of any port to be input or output within 1 cycle
- Let x be in {B,C,D}
  - DDRx takes an 8 bit value:
    - If a bit is 1, then the corresponding pin is an output
    - If a bit is 0, then the corresponding pin is an input
  - PORTx is an I/O register:
    - Write to a bit in PORTx sets the corresponding port/pin if the corresponding DDRx bit is set to 1
  - PINx contains inputs
- E.g.,
  1. DDRx says output
  2. Set PORT
  3. Read PIN is the value just set in the PORT
- The above registers control each I/O pin independently at a logical level

2

ATmega328P Header file snippet

#define PINB _SFR_IO8(0x03)	#define DDRB _SFR_IO8(0x04)	#define PORTB _SFR_IO8(0x05)
#define PINB0 0	#define DDB0 0	#define PORTB0 0
#define PINB1 1	#define DDB1 1	#define PORTB1 1
#define PINB2 2	#define DDB2 2	#define PORTB2 2
#define PINB3 3	#define DDB3 3	#define PORTB3 3
#define PINB4 4	#define DDB4 4	#define PORTB4 4
#define PINB5 5	#define DDB5 5	#define PORTB5 5
#define PINB6 6	#define DDB6 6	#define PORTB6 6
#define PINB7 7	#define DDB7 7	#define PORTB7 7

3

Reading a logic value from a Port

Suppose we want to read the logic value of 7th pin of Port B:

1. Read the register PINB in a character variable, i.e.
   `char reg = PINB`
2. Let PINB register has a value 0b10101010 then
   `reg = 0b10101010`
3. Create a mask to mask out all the bits in 'reg' except for 7th bit position, i.e.
   `0b10000000 = (1<<7) = (1<<PINB7)`
4. Use the mask to mask out all the bits except for the 7th bit, and decide based on the resultant value, i.e.

```
if( reg & (1<<PINB7) ) { /* 7th pin is logic 1 */ }
else                   { /* 7th pin is logic 0 */ }
```

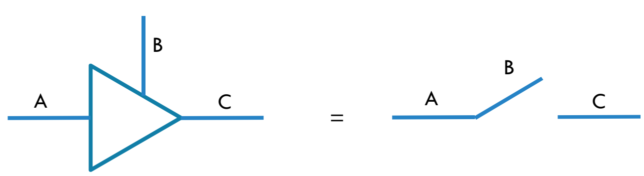
4

Tristate Buffer

- In a naïve button circuit, a closed button connects a pin to the MCU to Gnd:
  - When it opens, the MCU end of the button/switch (i.e. pin) dangles in the air
  - It acts as an antenna picking up high/low voltages depending on what frequency the local radio stations / "noisy" electrical appliances broadcast
  - Unreliable!
- Need a pull-up resistor (10kOhm) at the pin, so that if the switch is open, the voltage at the pin is pulled to high
  - If the switch is closed, the resistance to Gnd is much lower so that the voltage at the pin is close to zero
- The pull-up resistor is implicitly implemented by setting the output of the pin to high as a result of programming PORTx

5

Tristate Buffer



A (PORT)	B (DDR)	C (PIN)
0	1	Low impedance High out 0
1	1	Low impedance High out 1
0	0	High impedance
1	0	High impedance

- DDR (B) = 0 and PORT (A) = 1: Eliminates static effects/noise and allows to read port/pin in a coherent fashion → PORT (A) = 1 activates the pull-up resistor and makes reading PIN (C) reliable
- DDR (B) = 0 and PORT (A) = 0: Is good for creating high impedance if you do not want the PIN to have any current at all

6

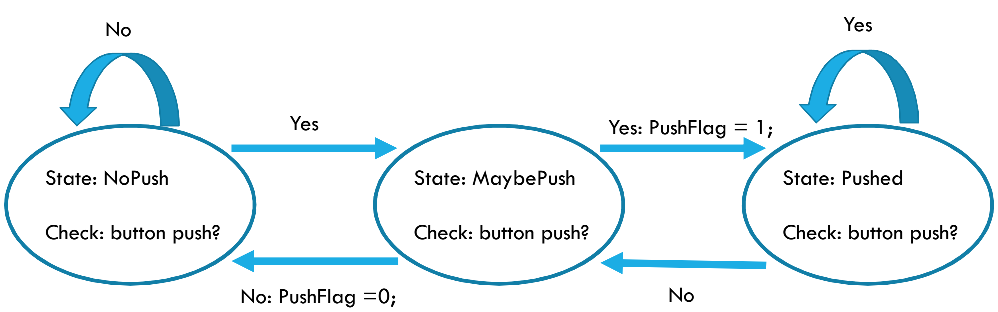
Debounce State Machine

- Capture a button push is a very fast process (compared to e.g setting a LED which is quite slow)
- When you press a switch closed, two surface are brought into contact with each other → no perfect match and electrical contact will be made and unmade a few times till the surfaces are firm enough together
  - The same is true when you release a button, but in reverse
  - Bouncing between high and low voltage is often at a timescale of a few us to a few ms → very often you do not see it
- No debouncing SW:

```
unsigned char PushFlag_NoDebounce; //message indicating a button push

void Task_PollingButton_NoDebounce(void)
{
  //button push of the switch connected to B.7
  if (~PINB & 0x80) PushFlag_NoDebounce = 1;
  else PushFlag_NoDebounce = 0;
}
```

7

Debounce State Machine



Checks happen every 30ms
- What happens if this time is increased?
- What happens if this time is decreased?

8

Debounce State Machine

```
unsigned char PushFlag_Debounce;

unsigned char PushState; //state machine
#define NoPush 1
#define Maybe 2
#define Pushed 3

void Task_PollingButton_Debounce(void)
{
  switch (PushState)
  {
     case NoPush:
        if (~PINB & 0x08) PushState=Maybe;
        else PushState=NoPush;
        break;
```

```
     case Maybe:
        if (~PINB & 0x08)
        {
           PushState=Pushed;
           PushFlag_Debounce=1;
        }
        else
        {
           PushState=NoPush;
           PushFlag_Debounce=0;
        }
        break;
     case Pushed:
        if (~PINB & 0x08) PushState=Pushed;
        else PushState=Maybe;
        break;
  }
}
```

9

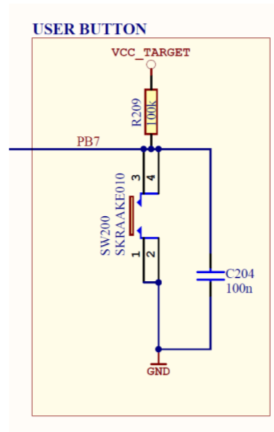
Debounce State Machine

- A SW debounce state machine can also be made for a keypad
- Depending on the application, you may want to add more actions to the Finite State Machine (FSM). In other words, you may want to synchronize your FSM with other tasks.
  - E.g., as soon as PushFlag =1 is set, I may want to increment a counter
  - E.g., during the state transition from NoPush to Maybe, the actual time (possibly as a translation from the HW timers hardcoded in the MCU) is recorded. As soon as NoPush changes into Pushed, this recorded time is considered to correspond to the moment of the most recent button push.

10

Hardware Debouncer

- HW debouncers are also possible:
  - Just by using a low pass filter (a capacitor across the two contacts of the switch)
  - However everyone debounces in SW, saving a few cents per capacitor
- Figure shows the schematic of the push button onboard ATmega328p Xplained Mini kit
  - This is Hardware Debounced switch (Notice the capacitor C204)
  - The switch is connected to PB7
- We will do software debouncing for this switch as well anyway.



11

LCD

- LCD has a command state machine:
  - Erase, Draw character, etc.
- Notice that (see http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf) the MCU is programmed through port B and C:
  - Cannot use PB3, PB4, PB5, PC6 to connect to LCD
  - If these would be connected to the databus for the LCD, then if a LCD read operation is interrupted, then the LCD is driving the bus → programmer cannot program the chip → program failure

2.2.4. Target Programming

The J204 header enable direct connection to the SPI bus with an external programmer for programming of the ATmega328P.

Table 2-5 SPI Header

J204 pin	ATmega328P pin	Function
1	PB4	MISO
2		VCC target
3	PB5	SCK
4	PB3	MOSI
5	PC6	RESET
6		GND

12

LCD

- LCD must be properly connected, otherwise the LCD does not acknowledge and the program hangs forever
- LCD library (lcd_lib.c and lcd_lib.h) uses #include <util/delay.h>
  - Allows using delay_ms() and delay_us()
  - We will use interrupts to program delay_ms() in next lectures (so that other computations can take place in the meantime)
  - Principle: Never use ms delays, but sometimes you may use a us delay because this is hard to get by using an interrupt
- Need to tell the LCD the clock rate of the MCU by setting
  #define F_CPU 16000000UL

13

LCD Example Display

Number = <u>Counter</u>
o ---------------------->
<----------------------

How do we store the constant string "Number=\0" ?

Many AVRs have limited amount of RAM in which to store data, but may have more Flash space available. The AVR is a Harvard architecture processor, where Flash is used for the program, RAM is used for data, and they each have separate address spaces.

- Let's use flash for storing data!

```
//For accessing program space:
#include <avr/pgmspace.h>

const   int8_t    LCD_number[]    PROGMEM="Number=\0";
```

- int8_t: Is the same as char
- LCD_number: Name
- []: [] tells C to look at the actual number of characters in the string and reserve and appropriate a chunk to hold it
- PROGMEM: Keyword tells C to store the string in program memory (flash)
- "Number=\0":
  - All strings in C are terminated by a \0 (i.e., the all-zero byte)
  - The string "Number=\0" is converted into ASCII integers, each integer is stored in 1 byte

14

LCD Example

```
/**
 * Written by Ruibing Wang (rw98@cornell.edu)
 * Mods for 644 by brl4@cornell.edu
 * Feb 2010
 *
 * Slightly modified for ECE-3411
 * by Marten van Dijk, Jan 2014
 */

#define F_CPU 16000000UL

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>
#include "lcd_lib.h"
```

```
const uint8_t LCD_initialize[] PROGMEM = "LCD Initialized\0";
const uint8_t LCD_number[] PROGMEM = "Number=\0";

// LCD display buffer: general purpose print buffer in RAM
// LCD can print 16 characters, 17th character holds \0
uint8_t lcd_buffer[17];

uint16_t count;          // a number to display on the LCD
uint8_t anipos, dir;     // move a character around
```

15

LCD Example (cont.)

```
// task  writes to LCD every 200 mSec
void task (void)
{
        // increment time counter and format string
        sprintf(lcd_buffer,"%-i",count++);
        LCDGotoXY(7, 0);
        // display the count
        LCDstring(lcd_buffer, strlen(lcd_buffer));

        // now move a char left and right
        LCDGotoXY(anipos,1);            //second line
        LcdDataWrite(' ');

        if (anipos>=7) dir=-1;   // check boundaries
        if (anipos<=0 ) dir=1;
        anipos =anipos+dir;
        LCDGotoXY(anipos,1);            //second line
        LcdDataWrite('o');
}
```

Prints to a string destination (not a file unit); C does internal transformation from integer to string format.

16

LCD Example (cont.)

```
int main(void)
{
        // Initializations:
        initialize_LCD();                    //initialize the display
        LCDcursorOFF();                      // Turn off the cursor
        CopyStringtoLCD(LCD_initialize, 0, 0);
        _delay_ms(2000);                     // Display message for 2 seconds

        LCDclr();    //clear the display
        // put some stuff on LCD starting at char=0 line=0
        CopyStringtoLCD(LCD_number, 0, 0);

        // Initialize animation state variables
        count=0;
        anipos = 0;
        LCDGotoXY(anipos,1);    //second line
        LcdDataWrite('o');

        while(1)    //main task scheduler loop
        {
                task();
                _delay_ms(200);
        }
}
```

This stalls any other computation … In next lectures we will use HW timer interrupts that can be used to wake up task() every 200ms. During task() idle time of 200ms other tasks can be completed.

17

LCD Pin Assignment

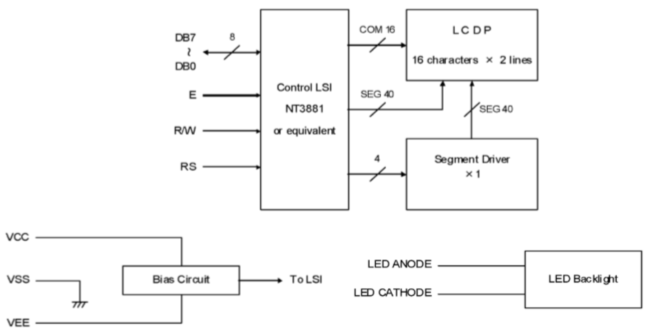
Taken from LCD Datasheet available here

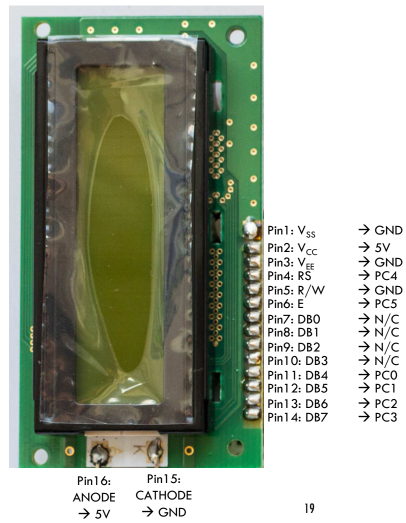
No.	Symbol	Level	Function
1	Vss	—	Power Supply (0V, GND)
2	Vcc	—	Power Supply for Logic
3	VEE	—	Power Supply for LCD Drive
4	RS	H / L	Register Select Signal
5	R/W	H / L	Read/Write Select Signal H : Read L : Write
6	E	H / L	Enable Signal (No pull-up Resister)
7	DB0	H / L	Data Bus Line / Non-connection at 4-bit operation
8	DB1	H / L	Data Bus Line / Non-connection at 4-bit operation
9	DB2	H / L	Data Bus Line / Non-connection at 4-bit operation
10	DB3	H / L	Data Bus Line / Non-connection at 4-bit operation
11	DB4	H / L	Data Bus Line
12	DB5	H / L	Data Bus Line
13	DB6	H / L	Data Bus Line
14	DB7	H / L	Data Bus Line
15	LED CATHODE	—	LED Cathode Terminal
16	LED ANODE	—	LED Anode Terminal

18

Block Diagram

- Because of limited number of I/O pins on Xplained Mini kit, we use LCD in 4-bit mode





Taken from LCD Datasheet available here

19

Write Operation Timing

```
void LcdCommandWrite_UpperNibble(uint8_t cm)
{
        // Give the higher half of 'cm' to DATA_PORT
        DATA_PORT = (DATA_PORT & 0xf0) | (cm >> 4);

        // Setting RS=0 to choose the instruction register
        // as we are writing a command
        CTRL_PORT &= ~(1<<RS);

        // Setting ENABLE=1
        CTRL_PORT |= (1<<ENABLE);

        // Allow the LCD controller to successfully read command in,
        // minimum 40 µs
        _delay_ms(1);

        // Setting ENABLE=0
        CTRL_PORT &= ~(1<<ENABLE);

        // Allow long enough delay for instruction writing
        _delay_ms(1);
}
```

See lcd.h for the definition of DATA_PORT and CTRL_PORT

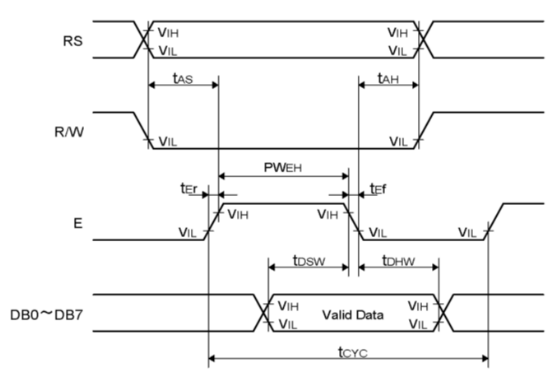


Fig.1 Write Operation Timing

Taken from LCD Datasheet available here

20

Read Operation Timing

- Read operation also follows similar timing as Write operation
  - Typically only a 'Busy Flag' is to be read
- We don't read 'Busy Flag', instead we provide the LCD controller long enough time to process the command
- Hence we only perform LCD writes
  - R/W signal is connected to GND, i.e. to always perform writes
  - This saves another I/O pin

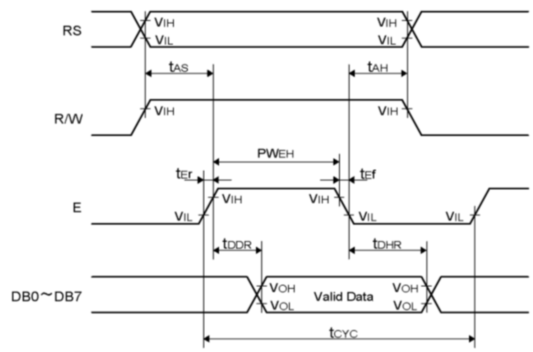


Fig.2 Read Operation Timing

Taken from LCD Datasheet available here

21

Timing Characteristics

Taken from LCD Datasheet available here

VCC=5.0V±10%

Parameter	Symbol	Conditions	Min.	Max.	Units
Enable Cycle Time	t_{CYC}	Fig.1, 2	500	—	ns
Enable Pulse Width	PW_{EH}	Fig.1, 2	300	—	ns
Enable Rise/Fall Time	t_{Er}, t_{Ef}	Fig.1, 2	—	25	ns
Address Setup Time	t_{AS}	Fig.1, 2	60	—	ns
Address Hold Time	t_{AH}	Fig.1, 2	10	—	ns
Write Data Setup Time	t_{DSW}	Fig.1	100	—	ns
Write Data Hold Time	t_{DHW}	Fig.1	10	—	ns
Read Data Delay Time	t_{DDR}	Fig.2	—	190	ns
Read Data Hold Time	t_{DHR}	Fig.2	20	—	ns

22

LCD Instruction Set

Taken from LCD Controller Datasheet available here

Instruction Set

Instruction	Code										Function	Execution time (max)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		(f_{OSC} = 250KHz)
Display Clear	0	0	0	0	0	0	0	0	0	1	Clear entire display area, restore display from shift, and load address counter with DD RAM address 00H.	1.64ms
Display/ Cursor Home	0	0	0	0	0	0	0	0	1	*	Restore display from shift and load address counter with DD RAM address 00H.	1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Specify direction of cursor movement and display shift mode. This operation takes place after each data transfer (read/write).	40μs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	Specify activation of display (D) cursor (C) and blinking of character at cursor position (B).	40μs
Display/ Cursor Shift	0	0	0	0	0	1	S/C	R/L	*	*	Shift display or move cursor.	40μs

LCD Instruction Set (cont.)

Instruction Set

Instruction	Code										Function	Execution time (max)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		(f_{OSC} = 250KHz)
Function Set	0	0	0	0	1	DL	N	F	*	*	Set interface data length (DL), number of display line (N), and character font (F).	40μs
RAM Address Set	0	0	0	1	ACG						Load the address counter with a CG RAM address. Subsequent data access is for CG RAM data.	40μs
DD RAM Address Set	0	0	1	ADD							Load the address counter with a DD RAM address. Subsequent data access is for DD RAM data.	40μs
Busy Flag/ Address Counter Read	0	1	BF	AC							Read Busy Flag (BF) and contents of Address Counter (AC).	0μs
CG RAM/ DD RAM Data Write	1	0	Write data								Write data to CG RAM or DD RAM.	40μs
CG RAM/ DD RAM Data Read	1	1	Read data								Read data from CG RAM or DD RAM.	40μs

Note 1: Symbol "*" signifies an insignificant bit (disregard).

Note 2: Correct input value for "N" is predetermined for each model.

LCD Instruction Set (cont.)

Instruction Set (continued)

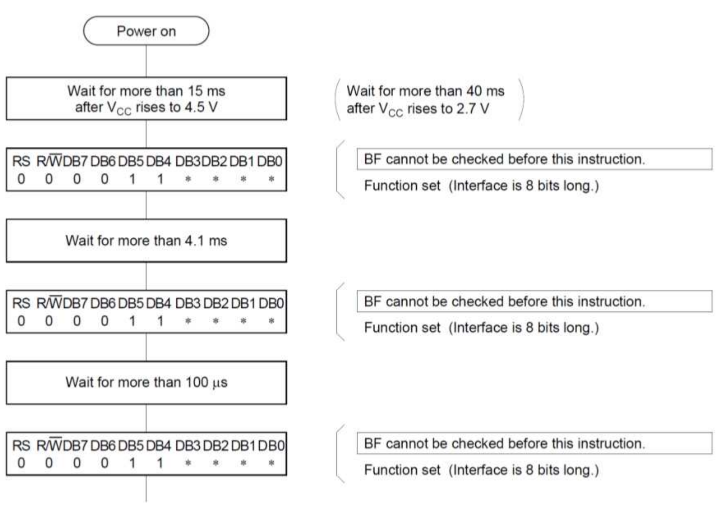
Instruction	Code										Function	Execution time (max)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		(f_{OSC} = 250KHz)
	I/D = 1 : Increment S = 1 : Display Shift On D = 1 : Display On C = 1 : Cursor Display On B = 1 : Cursor Blink On S/C = 1 : Shift Display R/L = 1 : Shift Right DL = 1 : 8-Bit N = 1 : Dual Line F = 1 : 5x10 dots BF = 1 : Internal Operation BF = 0 : Ready for Instruction	I/D = 0 : Decrement S/C = 0 : Move Cursor R/L = 0 : Shift Left DL = 0 : 4-Bit N = 0 : Signal Line F = 0 : 5x8 dots									DD RAM : Display Data RAM CG RAM : Character Generator RAM ACG : Character Generator RAM Address ADD : Display Data RAM Address AC : Address Counter	

Note 1: Symbol "*" signifies an insignificant bit (disregard).

Note 2: Correct input value for "N" is predetermined for each model.

25

LCD Initialization: 8-bit Mode



26

LCD Initialization: 8-bit Mode (cont.)

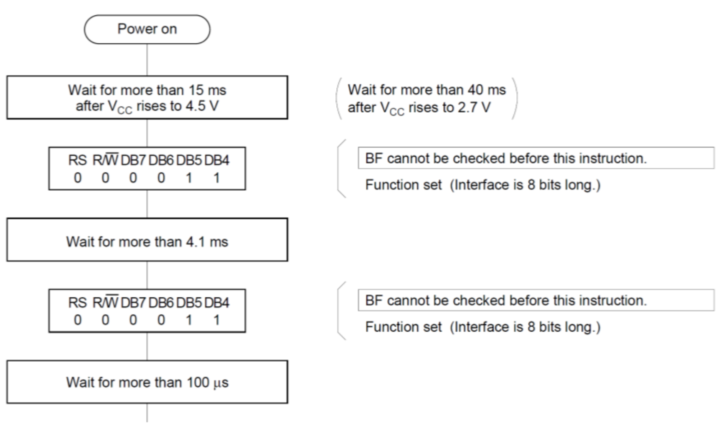
BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instuction time. (See Table 6.)

RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	1	1	N	F	*	*	Function set (Interface is 8 bits long. Specify the number of display lines and character font.) The number of display lines and character font cannot be changed after this point.
0	0	0	0	0	0	1	0	0	0	Display off
0	0	0	0	0	0	0	0	0	1	Display clear
0	0	0	0	0	0	0	1	I/D	S	Entry mode set

Initialization ends

27

LCD Initialization: 4-bit Mode




```
LcdCommandWrite_UpperNibble(0x30);
_delay_ms(4.1);
LcdCommandWrite_UpperNibble(0x30);
_delay_us(100);
```

28

LCD Initialization: 4-bit Mode (cont.)

```
LcdCommandWrite_UpperNibble(0x30);

// function set: 4-bit interface
LcdCommandWrite_UpperNibble(0x20);

// 4-bit interface, 2 lines, 5x8 font
LcdCommandWrite(0x28);

// turn display off, cursor off, no blinking
LcdCommandWrite(0x08);

// clear display, set address counter  to zero
LcdCommandWrite(0x01);

// entry mode set
LcdCommandWrite(0x06);
```

BF cannot be checked before this instruction.

RS	R/W̄	DB7	DB6	DB5	DB4	
0	0	0	0	1	1	Function set (Interface is 8 bits long.)

BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instuction time. (See Table 6.)

RS	R/W̄	DB7	DB6	DB5	DB4	
0	0	0	0	1	0	Function set (Set interface to be 4 bits long.) Interface is 8 bits in length.
0	0	0	0	1	0	Function set (Interface is 4 bits long. Specify the number of display lines and character font.) The number of display lines and character font cannot be changed after this point.
0	0	N	F	*	*	
0	0	0	0	0	0	Display off
0	0	1	0	0	0	
0	0	0	0	0	0	Display clear
0	0	0	0	0	1	
0	0	0	0	0	0	Entry mode set
0	0	0	1	I/D	S	

Initialization ends

LCD Command Write (4-bit Mode)

```
void LcdCommandWrite(uint8_t cm)
{
		// First send higher 4-bits
		DATA_PORT = (DATA_PORT & 0xf0) | (cm >> 4);		//give the higher half of cm to DATA_PORT
		CTRL_PORT &= ~(1<<RS);				//setting RS=0 to choose the instruction register
		CTRL_PORT |= (1<<ENABLE);			//setting ENABLE=1
		_delay_ms(1);					// allow the LCD controller to successfully read command in
		CTRL_PORT &= ~(1<<ENABLE);			// Setting ENABLE=0
		_delay_ms(1);					// allow long enough delay for instruction writing

		// Send lower 4-bits
		DATA_PORT = (DATA_PORT & 0xf0) | (cm & 0x0f);	//give the lower half of cm to DATA_PORT
		CTRL_PORT &= ~(1<<RS);				//setting RS=0 to choose the instruction register
		CTRL_PORT |= (1<<ENABLE);			//setting ENABLE=1
		_delay_ms(1);					// allow the LCD controller to successfully read command in
		CTRL_PORT &= ~(1<<ENABLE);			// Setting ENABLE=0
		_delay_ms(1);					// allow long enough delay for instruction writing
}
```

LCD Data Write (4-bit Mode)

```
void LcdDataWrite(uint8_t da)
{
        // First send higher 4-bits
        DATA_PORT = (DATA_PORT & 0xf0) | (da >> 4);      //give the higher half of cm to DATA_PORT
        CTRL_PORT |= (1<<RS);                             //setting RS=1 to choose the data register
        CTRL_PORT |= (1<<ENABLE);                         //setting ENABLE=1
        _delay_ms(1);                                     // allow the LCD controller to successfully read command in
        CTRL_PORT &= ~(1<<ENABLE);                        // Setting ENABLE=0
        _delay_ms(1);                                     // allow long enough delay

        // Send lower 4-bits
        DATA_PORT = (DATA_PORT & 0xf0) | (da & 0x0f);    //give the lower half of cm to DATA_PORT
        CTRL_PORT |= (1<<RS);                             //setting RS=1 to choose the data register
        CTRL_PORT |= (1<<ENABLE);                         //setting ENABLE=1
        _delay_ms(1);                                     // allow the LCD controller to successfully read command in
        CTRL_PORT &= ~(1<<ENABLE);                        // Setting ENABLE=0
        _delay_ms(1);                                     // allow long enough delay
}
```

31

ECE3411 – Fall 2017
Lab1c.

General Purpose Digital Input
LCD Interfacing

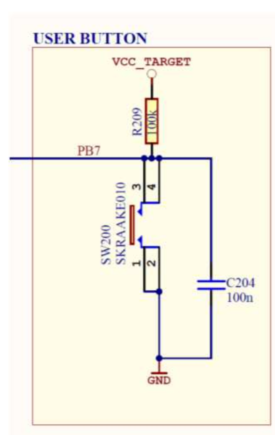
Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

UCONN

Adopted from Lab 2c slides "General Purpose Digital Input LCD Interfacing" by Marten van Dijk and Syed Kamran Haider, Fall 2015.

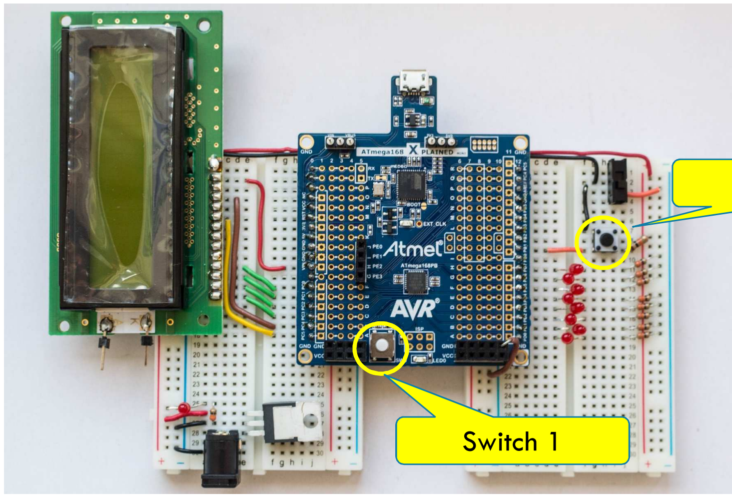
Push Switch Interface

- A push switch provides a logic HIGH or LOW value to the microcontroller pin to which it is connected
  - HIGH: When the switch is not pressed
  - LOW: When the switch is pressed
- Figure shows the schematic of the push button onboard ATmega328p Xplained Mini kit
  - The switch is connected to PB7
- We have another push switch on the bread board which is connected to PB1
- You should use the switch on the bread board (Switch 2) for debouncing tasks



2

Available Push Switches



3

LCD Interfacing

- We are going to use the LCD in 4-bit mode
  - Only 4 data wires are required instead of 8
- LCD pin assignment is as follows:

No.	Symbol	Connections with ATmega328P
1, 3	V_{SS}, V_{EE}	GND
2	V_{CC}	5V
4	RS	PC4
5	R/W	GND (Always Write to LCD)
6	E	PC5
7-10	DB0-DB3	Not Connected
11-14	DB4-DB7	PC0-PC3

Pin1: V_{SS} → GND
Pin2: V_{CC} → 5V
Pin3: V_{EE} → GND
Pin4: RS → PC4
Pin5: R/W → GND
Pin6: E → PC5
Pin7: DB0 → N/C
Pin8: DB1 → N/C
Pin9: DB2 → N/C
Pin10: DB3 → N/C
Pin11: DB4 → PC0
Pin12: DB5 → PC1
Pin13: DB6 → PC2
Pin14: DB7 → PC3

Pin16: ANODE → 5V
Pin15: CATHODE → GND

4

Using LCD Library

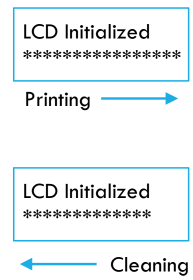
- In order to facilitate you, we provide a library file "lcd_lib.c" which defines some useful basic LCD functions.
  - "lcd_lib.h" and "lcd_lib.c" can be downloaded from Piazza under Resources.
- The corresponding prototypes of the functions are declared in "lcd_lib.h" file which comes along with "lcd_lib.c"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

LCD Test Program

```
// ------- Preamble -------- //
#define F_CPU 16000000UL      /* Tells the Clock Freq to the Compiler. */
#include <avr/io.h>           /* Defines pins, ports etc. */
#include <util/delay.h>       /* Functions to waste time */
#include "lcd_lib.h"          /* LCD Library */

int main(void) {
  // -------- Inits --------- //
   initialize_LCD();           /* Initialize LCD */

   LcdDataWrite('A');         /* Print a few characters for test */
   LcdDataWrite('B');
   LcdDataWrite('C');

  // ------ Event loop ------ //
  while (1) {
          /* Nothing to do */
  } /* End event loop */
  return (0);
}
```

Task: Reading a Non-Debounced & Debounced Switch

- Read the input of a push switch (PINB1) and print a character ‘ * ’ on the LCD for each button push
  - Whenever the button connected to PINB1 is pushed, one ‘ * ’ is printed on LCD. (So, no matter the duration, a single button push should result in printing only one ‘ * ’.)
- Once a row of LCD is filled with characters ‘ * ’, the subsequent button pushes should start clearing the LCD
  - Most recently printed character is cleared first, and so on until all ‘ * ’ are cleared.
- Implement this task with both non-debounced and debounced switch.



7

Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2017

Problem Set P1

There are 5 questions in this quiz. Answer each question according to the instructions given in at least 3 sentences on own words.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit! No handwritten solutions will be accepted.

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

SUBMIT YOUR ANSWERS IN PDF FORMAT

Do not write in the box below

1 (x/16)	2 (x/20)	3 (x/22)	4 (x/22)	5 (x/10)	Total (xx/100)

Name:

Student ID:

1. [16 points]: Assume initially $PORTC = 0b01011000$, $PORTB = 0b10100001$, $DDRB = 0xA5$ and $PINB = 100$

a. Give the bit representation of PORTC after computing $PORTC \quad \&= \quad \sim (1 << 4)$

b. What is the bit representation of PORTB: $PORTB \quad \hat{} = \quad ((1 << 5) \mid (1 << 1))$

c. What is the output of the register PINB : $PINB \quad |= \quad \sim ((12 >> 2)\&(16 >> 1))$

d. Give the bit representation of DDRB : $DDRB \quad |= \quad (19 >> 2)$

Initials:

2. [20 points]: Answer the following questions:

a. The compiler will generate an error while compiling the following line of C code. Write the correct version of this line in the space below.

```
const uint8_t my_string PROGMEM = "Hello!";
```

b. How many lines/wires do we need for a UART connection between a transmitter and receiver?

c. What is the minimum number of bits that must be transmitted to transmit one character in one UART frame?

d. Encircle one of the following options. The UDR0 register is used for:

(a) Receiving UART frames.

(b) Transmitting UART frames

(c) Both (a) and (b)

e. Consider the following push-switch circuit. When this switch is pushed, the logic value passed to AVR (i.e. voltage at node `'To AVR'`) is:

(a) Logic HIGH

(b) Logic LOW

(c) None of the above

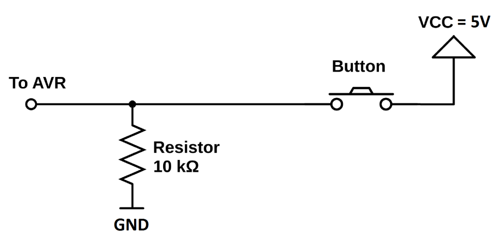


Figure 1: A push switch circuit.

Initials:

3. [22 points]: Using Table 1, calculate the required value of UART Baud Rate Register UBRR0 for a baud rate of 1000 in Asynchronous Normal mode, where the System Oscillator clock frequency of 16MHz. Also, write C code inside `Initialize_UBRR0(uint16_t Value)` function to store the value of argument `Value` into UBRR0 register.

Table 1: Equations for calculating UART Baud Rate Register setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRRn} + 1)}$	$\boldsymbol{UBRRn} = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRRn} + 1)}$	$\boldsymbol{UBRRn} = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRRn} + 1)}$	$\boldsymbol{UBRRn} = \frac{f_{OSC}}{2BAUD} - 1$

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

UBRRn Contents of the UBRRnH and UBRRnL Registers, (0-4095)

Calculated UBRR0 value =

```
/* Write the code for initializing 'UBRR0' here */
void Initialize_UBRR0(uint16_t Value)
{



}
```

Initials:

4. [22 points]: Use LCD Instruction Set table (Table 3) provided on page 6 to fill LCD Commands Table (Table 2) below with the correct bit values of **RS**, **R/W** and **DB7-DB0** signals to configure/control the LCD according the specified desired functionality.

Table 2: LCD Commands Table

No.	Desired Functionality	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	Set interface data length to 8-bit mode, number of display lines to 1, and character font to 5×10 dots.										
2	Turn the display OFF, cursor OFF, and no blinking.										
3	Set the direction of cursor movement towards right and turn the display shift mode ON.										
4	Turn the display ON, cursor ON, and no blinking.										
5	Move the cursor to position $(0, 5)$, i.e. first row and sixth column. **Hint:** The first row starts from DD RAM address $0\text{x}00$.										
6	Write the character 'A' to the LCD. The ASCII value of 'A' is $0\text{x}41$.										

Initials:

Table 3: LCD Instruction Set

Instruction	Code										Function	Execution time (max)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		(f_{OSC} = 250KHz)
Display Clear	0	0	0	0	0	0	0	0	0	1	Clear entire display area, restore display from shift, and load address counter with DD RAM address 00H.	1.64ms
Display/ Cursor Home	0	0	0	0	0	0	0	0	1	*	Restore display from shift and load address counter with DD RAM address 00H.	1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Specify direction of cursor movement and display shift mode. This operation takes place after each data transfer (read/write).	40μs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	Specify activation of display (D) cursor (C) and blinking of character at cursor position (B).	40μs
Display/ Cursor Shift	0	0	0	0	0	1	S/C	R/L	*	*	Shift display or move cursor.	40μs
Function Set	0	0	0	0	1	DL	N	F	*	*	Set interface data length (DL), number of display line (N), and character font (F).	40μs
RAM Address Set	0	0	0	1	ACG						Load the address counter with a CG RAM address. Subsequent data access is for CG RAM data.	40μs
DD RAM Address Set	0	0	1	ADD							Load the address counter with a DD RAM address. Subsequent data access is for DD RAM data.	40μs
Busy Flag/ Address Counter Read	0	1	BF	AC							Read Busy Flag (BF) and contents of Address Counter (AC).	0μs
CG RAM/ DD RAM Data Write	1	0	Write data								Write data to CG RAM or DD RAM.	40μs
CG RAM/ DD RAM Data Read	1	1	Read data								Read data from CG RAM or DD RAM.	40μs
	I/D = 1 : Increment S = 1 : Display Shift On D = 1 : Display On C = 1 : Cursor Display On B = 1 : Cursor Blink On S/C = 1 : Shift Display R/L = 1 : Shift Right DL = 1 : 8-Bit N = 1 : Dual Line F = 1 : 5x10 dots BF = 1 : Internal Operation BF = 0 : Ready for Instruction					I/D = 0 : Decrement S/C = 0 : Move Cursor R/L = 0 : Shift Left DL = 0 : 4-Bit N = 0 : Signal Line F = 0 : 5x8 dots					DD RAM : Display Data RAM CG RAM : Character Generator RAM ACG : Character Generator RAM Address ADD : Display Data RAM Address AC : Address Counter	

Note 1: Symbol "*" signifies an insignificant bit (disregard).

Note 2: Correct input value for "N" is predetermined for each model.

Initials:

5. [10 points]: Can you shortly describe what you have learned and feel confident about using in the future?

End of Problem Set

Initials:

Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2017

Advanced Problem Set A1

There are 4 questions in this quiz. Answer each question according to the instructions given in at least 3 sentences on own words.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit! No handwritten solutions will be accepted.

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

Do not write in the box below

1 (x/20)	2 (x/20)	3 (x/24)	4 (x/36)	Total (xx/100)

1. [20 points]: Let `Task1()` and `Task2()` be two functions from standard C library. We want to call `Task1()` once and only once every time a push button is pushed from released state, and we want to call `Task2()` once and only once every time the button is released from pushed state. The function `_button_pushed()` returns TRUE as long as the push button is pressed, and False otherwise.
Implement the above mentioned functionality by extending `Task_PollingButton_Debounce(void)` function given below.

```
/* Debouncing State Machine */
void Task_PollingButton_Debounce(void)
{
    switch (PushState)
    {
        case NoPush:
        if ( _button_pushed() ) PushState=Maybe;
        else PushState=NoPush;
        break;

        case Maybe:
        if ( _button_pushed() ){ PushState=Pushed; PushFlag_Debounce=1; }
        else { PushState=NoPush; PushFlag_Debounce=0; }
        break;

        case Pushed:
        if ( _button_pushed() ) PushState=Pushed;
        else PushState=Maybe;
        break;
    }
}


/* Write your code below */
```

Initials:

```
/* Your code continues here */
```

Initials:

2. [20 points]: Answer the following questions.

a. Software based debouncing performs *Read-Wait-Verify* sequence on the digital input signal to filter out the glitches. The figure below shows a push-switch circuit and the signal generated by it (i.e. the voltage at node 'To AVR') while going from 'Pushed' (Low) state to 'Released' (High) state. Each division on the horizontal axis of the graph represents $100\mu s$. What should be the minimum wait time for the *Read-Wait-Verify* sequence in order to filter out all the glitches shown in the graph? Please round your answer to the closest multiple of $100\mu s$.

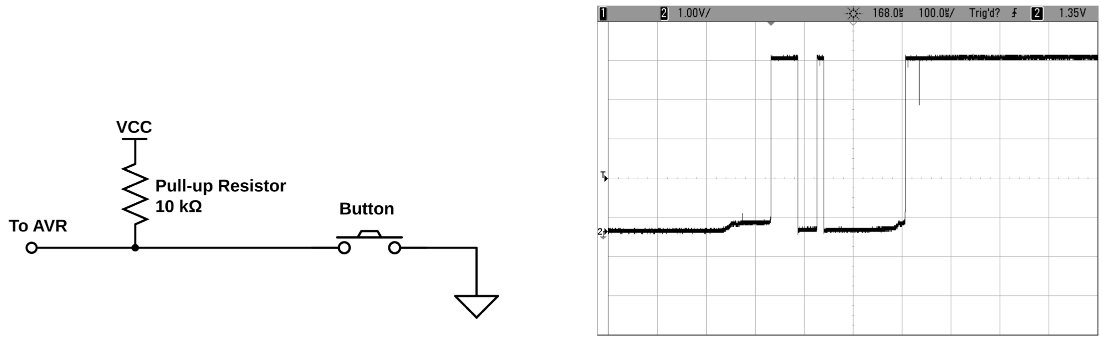


Figure 1: A push switch circuit and its generated signal.

b. The push switch circuit from the previous problem has been slightly modified as shown in the figure below. Please draw the waveform of the signal generated by this switch (i.e. the voltage at node 'To AVR') when the switch transitions from 'Pushed' state to 'Released' state. Compare this waveform with the one in the previous question and explain the difference between the two.

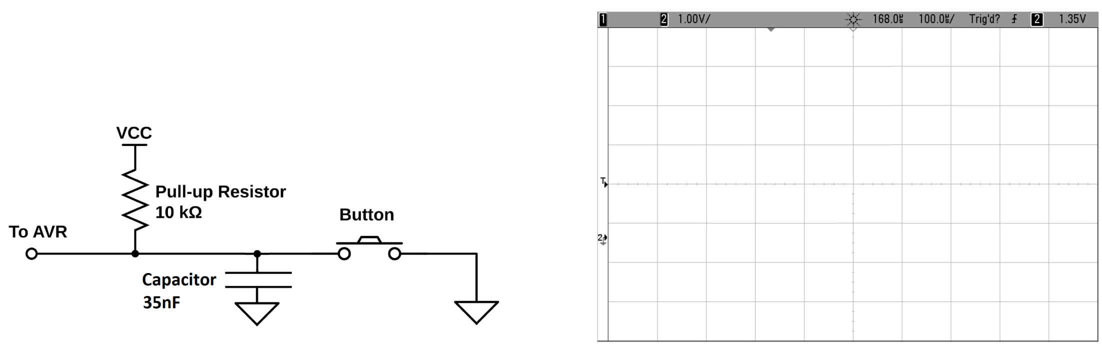


Figure 2: A modified push switch circuit.

Initials:

3. [24 points]: Suppose you are provided with an already initilaized LCD of 100×100 pixels along with the LCD libaray that contains two functions:

- *pixel_on(row, column)*, and
- *pixel_off(row, column)*.

If function *pixel_on(i, j)* is called, then the pixel residing at the i^{th} row and j^{th} column is switched "on". If function *pixel_off(i, j)* is called, then the pixel residing at the i^{th} row and j^{th} column is switched "off".

Using the above functions, you are required to control the glow of the LCD by switching on/off the pixels in a probabilistic manner.

a. Consider all the pixels are off, write a pseudo code to achieve 30% glow by controlling the switching of pixels in such a way that nearly 30% of the total pixels are "on" all the time with the following requirement: The distribution of these 30% pixels should be random across the LCD – in particular, approximately 30% granularity of turning on/off the pixels should be for each row/column of the LCD.

HINT: Use $RAND()$ function to generate numbers with a uniform distribution.

```
/* Declare any variables here */





/* Write your pseudo code below */





/* End of pseudo code */
```

Initials:

b. Each individual pixel should not either be always on or always off as this will over burden those pixels of the LCD that are always on. For this reason, you need to modify your pseudo code developed for part a. such that each individual pixel is on about 30% of the time in addition to the requirement that about 30% of the total number of pixels is on at any moment in time. For example, if the LCD is powered up for $T = 1000$ seconds, then each pixel is on for approximately 300 seconds randomly distributed over time and across the LCD.

```
/* Declare any variables here */
```

```
/* Write your pseudo code below */
```

```
/* End of pseudo code */
```

Initials:

4. [36 points]: UART (Universal Asynchronous Receiver Transmitter) is a kind of serial communication protocol which is commonly used for short-distance and low speed data exchange between computer and peripherals. It includes two main kernel modules, a receiver and a transmitter. The function of the transmit module is to convert the sending 8-bit parallel data into serial data.

For reliable transmission, it adds a start bit at the head of the data as well as a parity and stop bits at the end of the data. When the UART sets the START signal to 1, the transmit module immediately enters the START state to send the data, otherwise stays in the IDLE state. In this state, the 8-bit parallel data is read into a register BUFFER[7: 0]. The order follows 1 start bit, 8 data bits, 1 parity bit and 1 stop bit. The parity bit is determined according to the number of logic 1 values in the 8 data bits (1 for even number of 1's and 0 for odd number of 1's). Then the parity bit is output. When the data is ready to be transmitted, the system enters the WAIT state. In this state, the state machine realizes the parallel to serial conversion of outgoing data. Finally, logic 1 is output as the stop bit. Until the stop bit is received, the module stays in the WAIT state. When the data transmission is completed and stop bit is received, the state machine enters the STOP state. The state machine return to IDLE state after sending the stop bit, and waits for another data frame transmit command. Moreover, whenever the reset signal is set, the module goes to IDLE state.

a. Design a state machine diagram for the transmission module of UART.

b. Show a step by step transmission process when we need to transmit the message"**Hi!!**" (excluding the apostrophes) to the receiver. (HINT: Use the Hexadecimal form of letters and exclamation marks.)

End of Problem Set

Initials: