ECE3411 – Fall 2016

Lecture 1b.

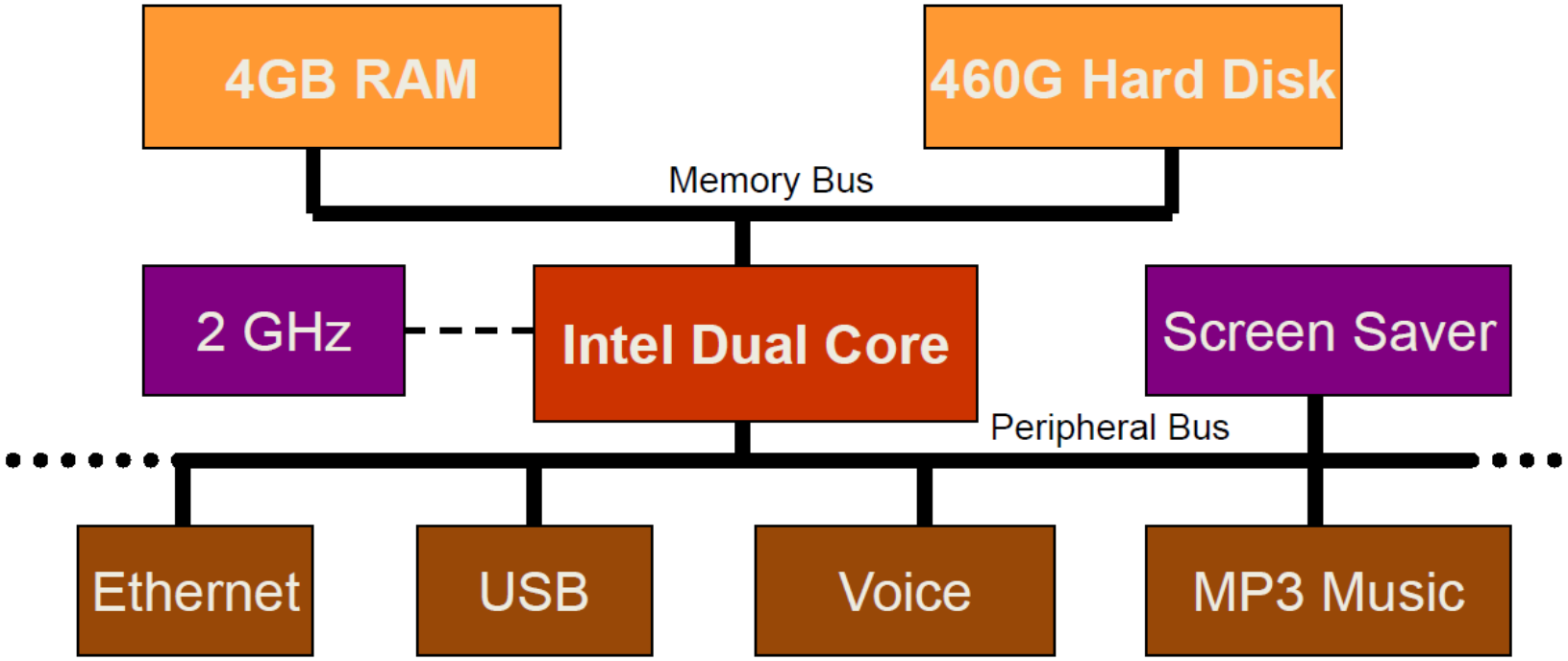
Introduction to Microcontrollers General Purpose Digital Output

Marten van Dijk & Syed Kamran Haider
Department of Electrical & Computer Engineering
University of Connecticut
Email: vandijk@engr.uconn.edu

UCONN

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

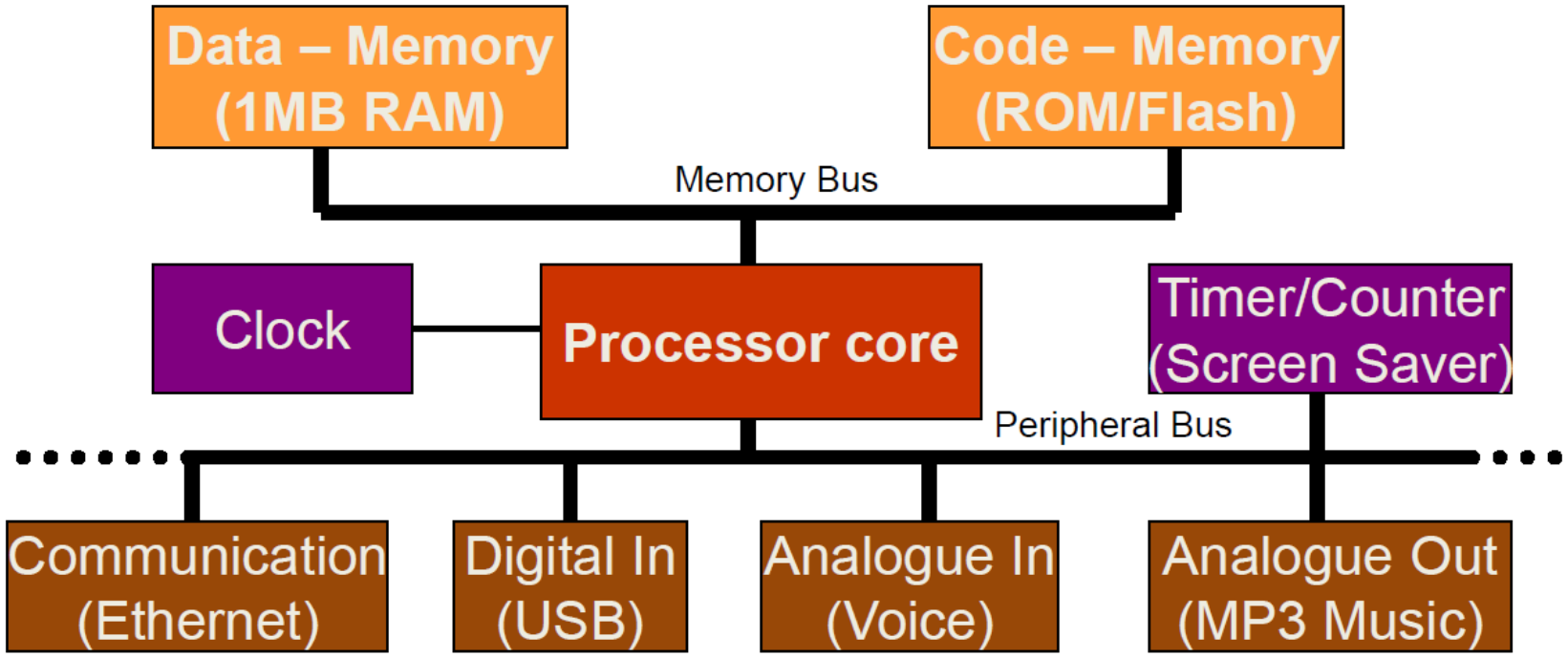
A Personal Computer



Slide from Sung Yeul Park

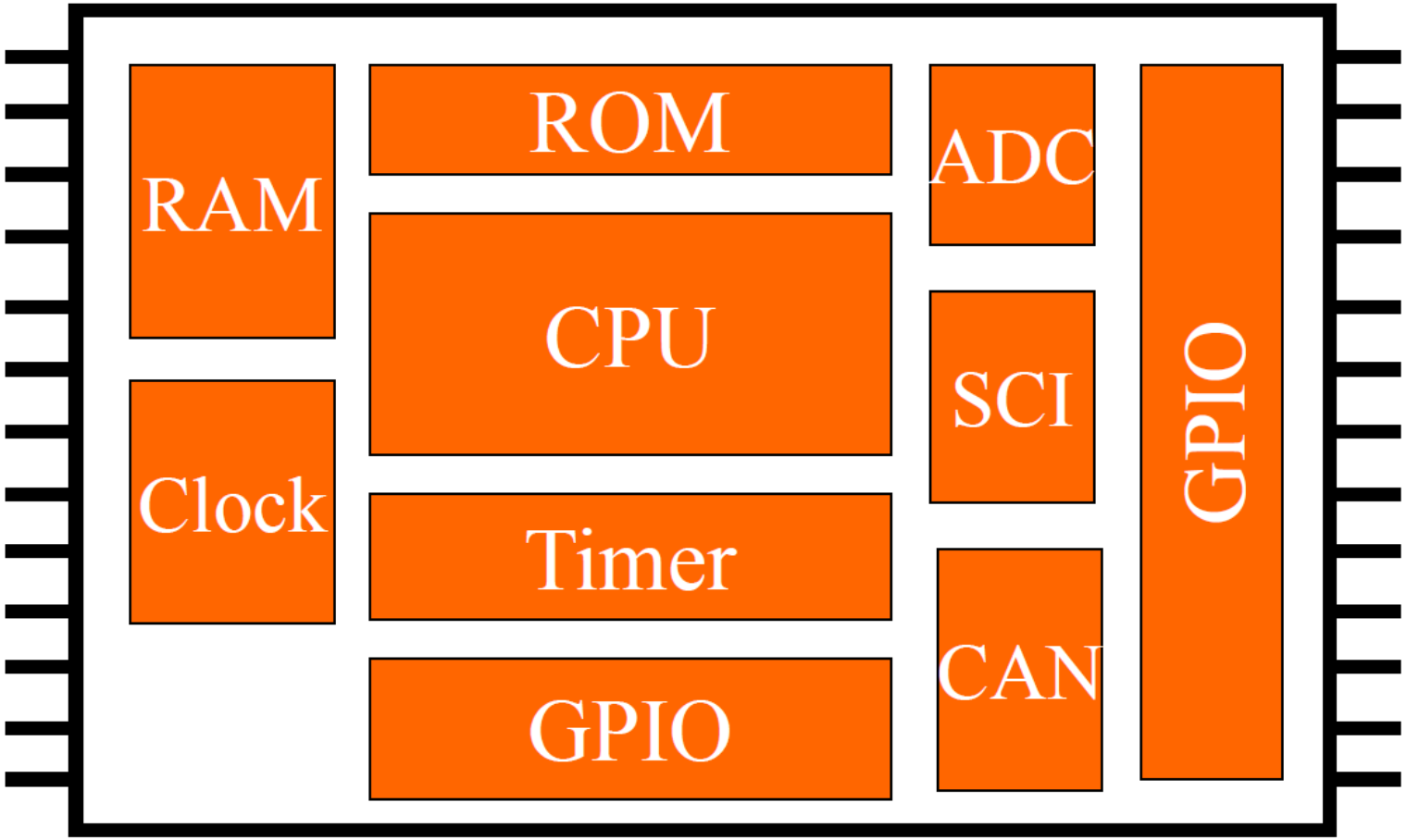
A Microcontroller

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



Slide from Sung Yeul Park

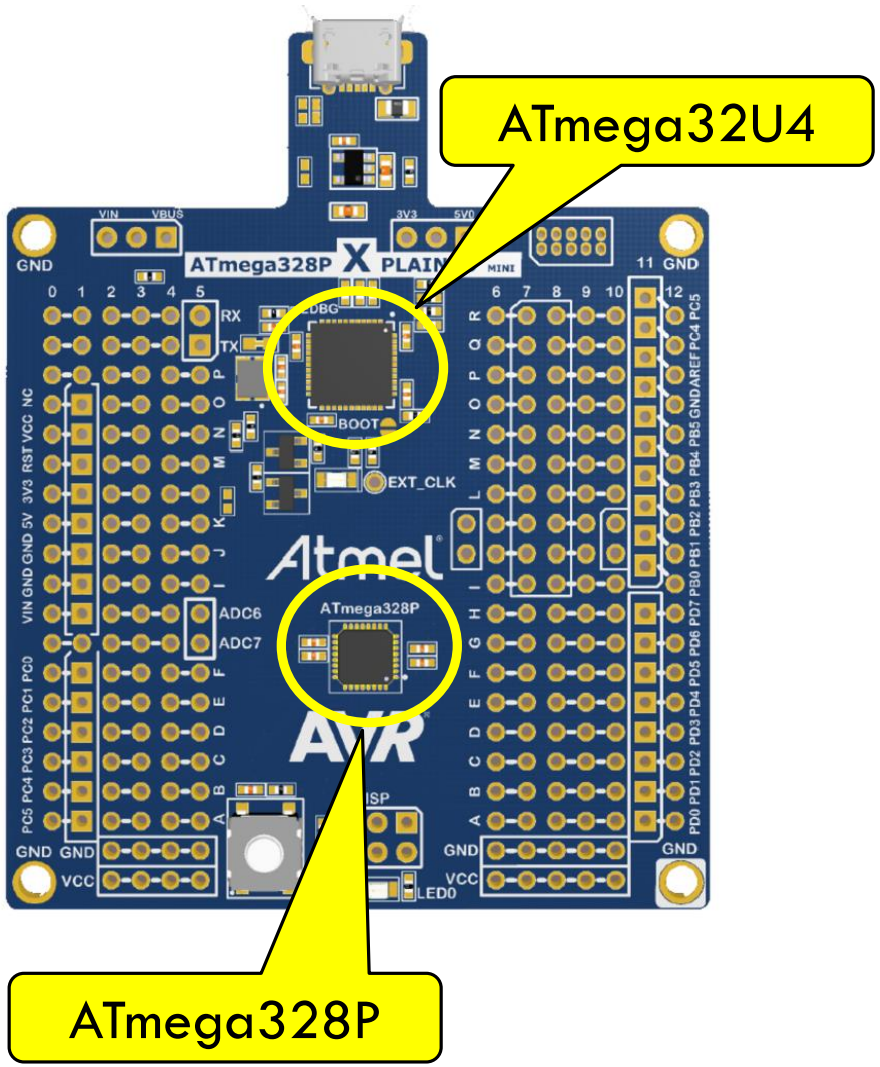
Microcontroller Structure



Slide from Sung Yeul Park

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



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

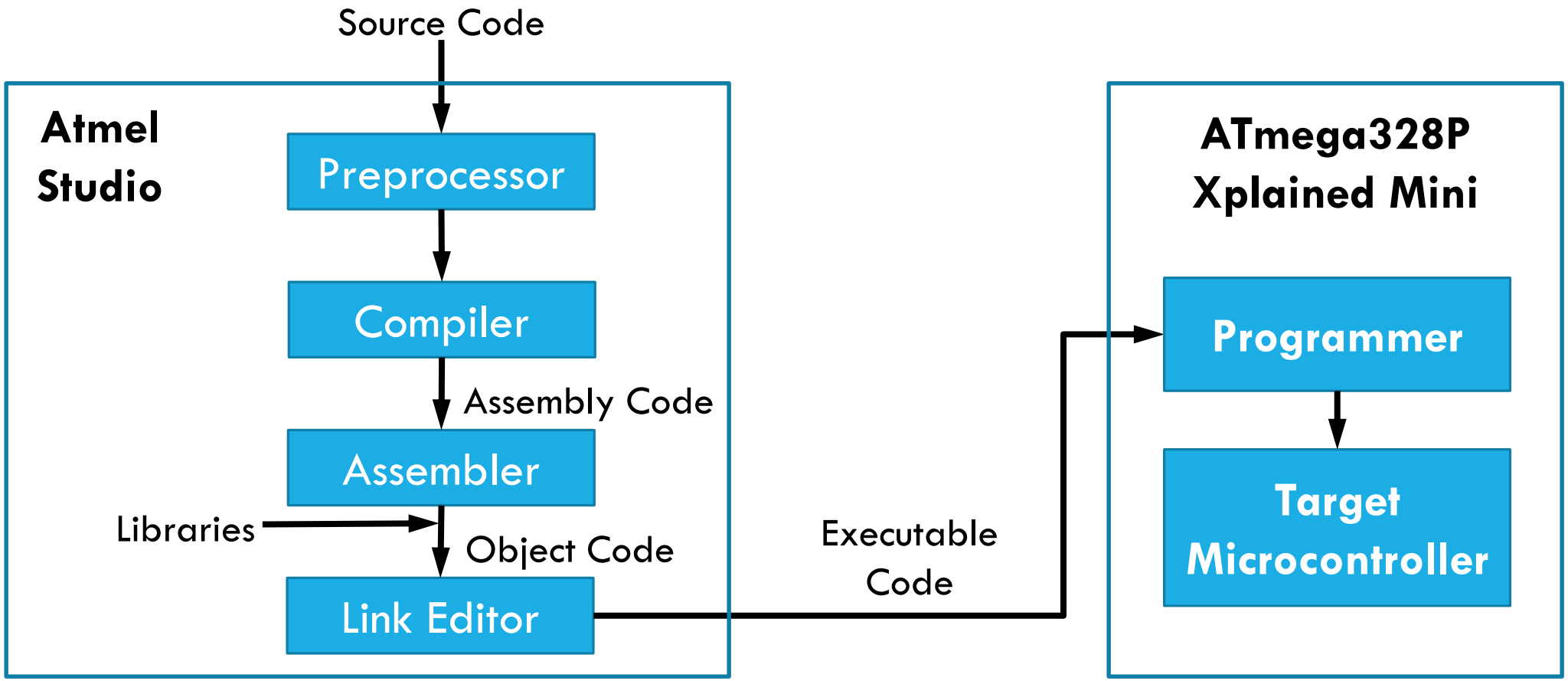
ATmega328P Features (2)

- 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

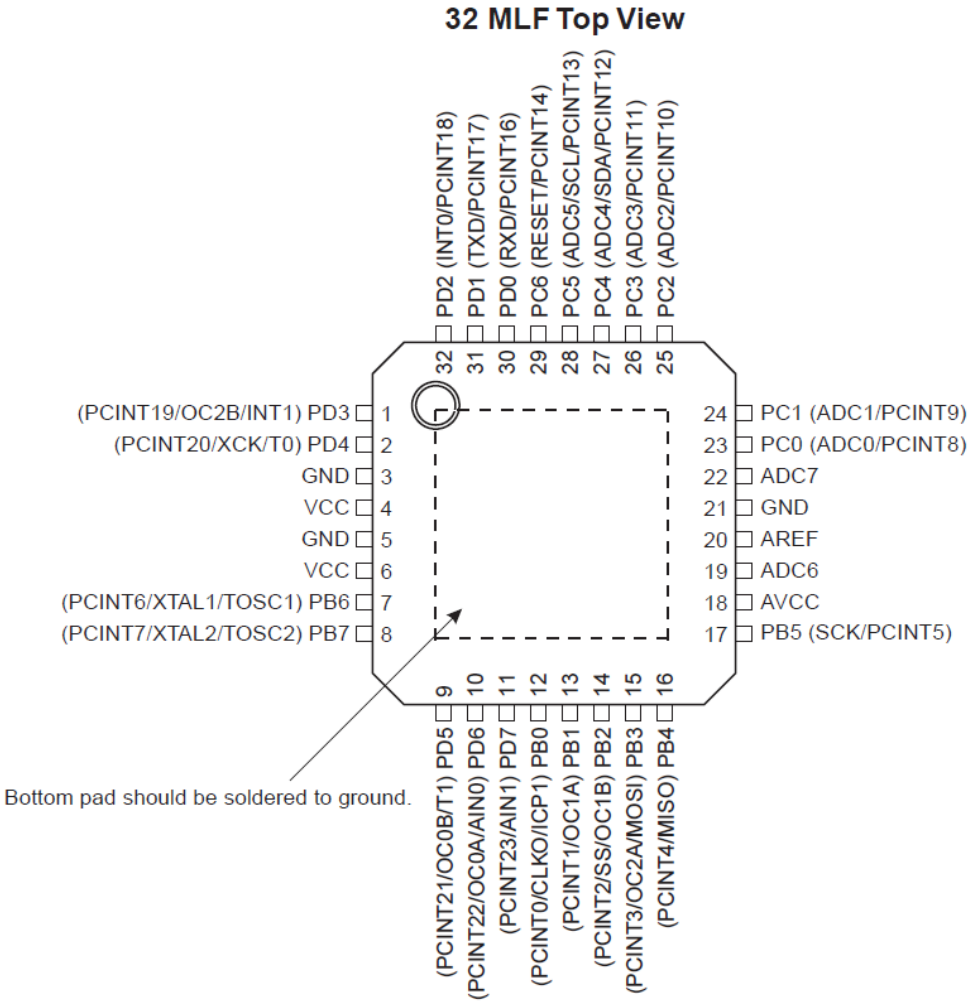
ATmega328P Features (3)

- 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)

AVR Software Development Process



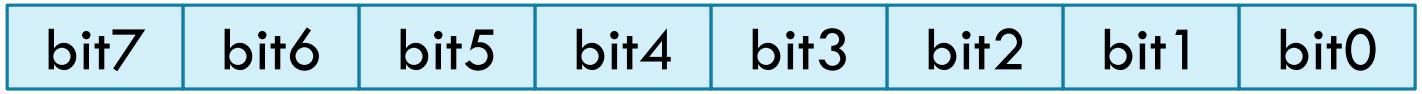
ATmega328P


32 MLF Top View
(PCINT19/OC2B/INT1) PD3 1
(PCINT20/XCK/T0) PD4 2
GND 3
VCC 4
GND 5
VCC 6
(PCINT6/XTAL1/TOSC1) PB6 7
(PCINT7/XTAL2/TOSC2) PB7 8
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 ADC7
21 GND
20 AREF
19 ADC6
18 AVCC
17 PB5 (SCK/PCINT5)
32 PD2 (INT0/PCINT18)
31 PD1 (TXD/PCINT17)
30 PD0 (RXD/PCINT16)
29 PC6 (RESET/PCINT14)
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
9 (PCINT21/OC0B/T1) PD5
10 (PCINT22/OC0A/AIN0) PD6
11 (PCINT23/AIN1) PD7
12 (PCINT0/CLKO/ICP1) PB0
13 (PCINT1/OC1A) PB1
14 (PCINT2/SS/OC1B) PB2
15 (PCINT3/OC2A/MOSI) PB3
16 (PCINT4/MISO) PB4
Bottom pad should be soldered to ground.

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



- **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 7^{th} bit position of PINB register, the predefined keyword is **PINB7**
  - Similarly **PORTB5** represents 5^{th} 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.

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

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).

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

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 */
}
```