

Smart Software Project

Lecture: Week 4
ATmega2560 MCU

Prof. HyungJune Lee
hyungjune.lee@ewha.ac.kr



이화여자대학교
EWHHA WOMANS UNIVERSITY

Today

- Review from the last lecture
 - Analog vs. Digital, PWM
 - ATmega 2560
- Midterm: 2pm - 4:30pm, Mon Apr 11
- ATmega2560 microcontroller (MCU) architecture
 - CPU, Memory, I/O ports
- Announcement



Class Schedule

Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	



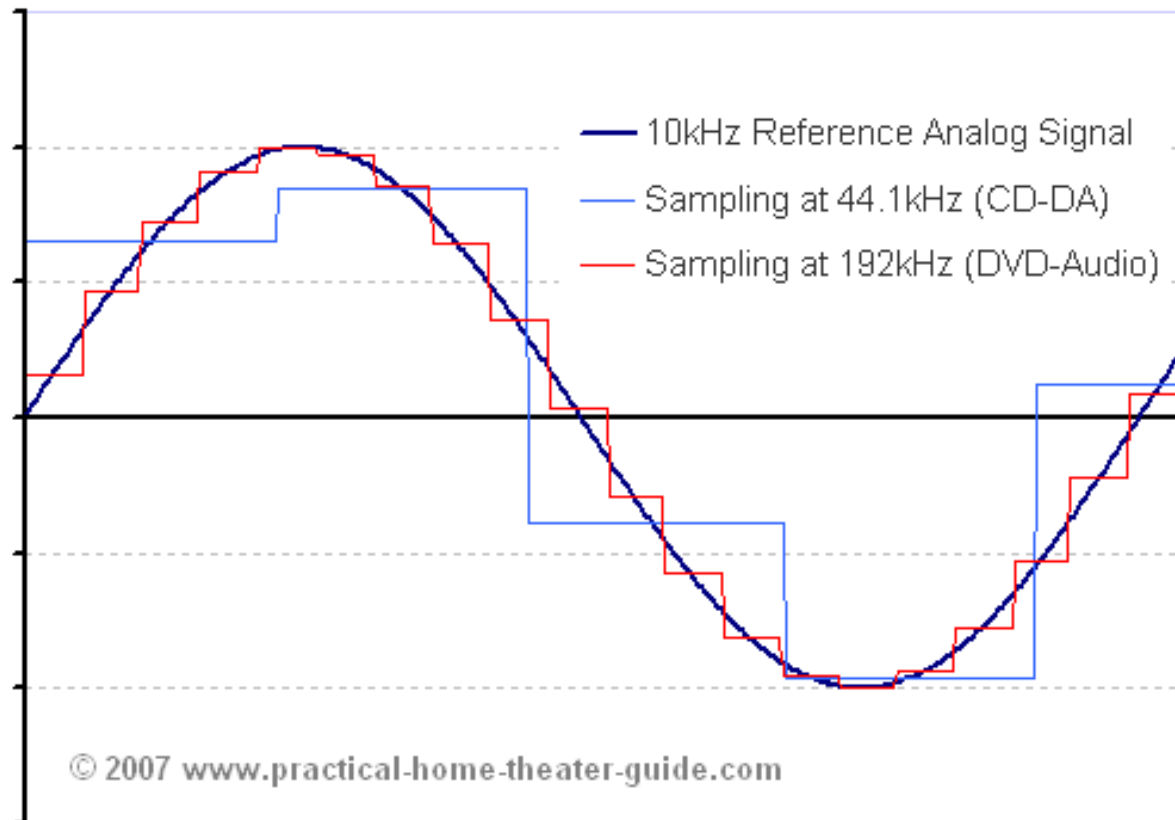
Today

- **Review from the last lecture**
 - Analog vs. Digital, PWM
 - ATmega 2560
- Midterm: 2pm - 4:30pm, Mon Apr 11
- ATmega2560 microcontroller (MCU) architecture
 - CPU, Memory, I/O ports
- Announcement

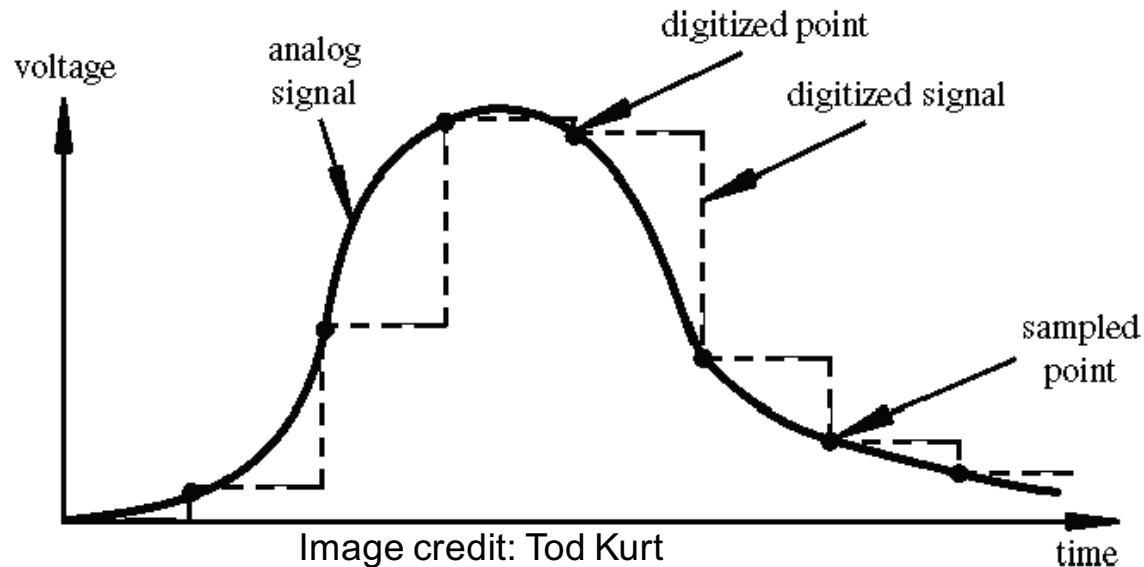


Analog vs. Digital

- Think about music stored on a CD: an analog signal captured on digital media
 - Sampling (with sampling rate)
 - Discretization (with discretization level)



Arduino Analog Input



- *Resolution*: the number of different voltage levels (i.e., *states*) used to discretize an input signal
- Resolution values range from 256 states (8 bits) to 4,294,967,296 states (32 bits)
- The Arduino uses 1024 states (10 bits)
- Smallest measurable voltage change is $5V/1024$ or 4.8 mV
- Maximum sample rate is 10,000 times a second

Analog Output

- Can a digital device produce analog output?

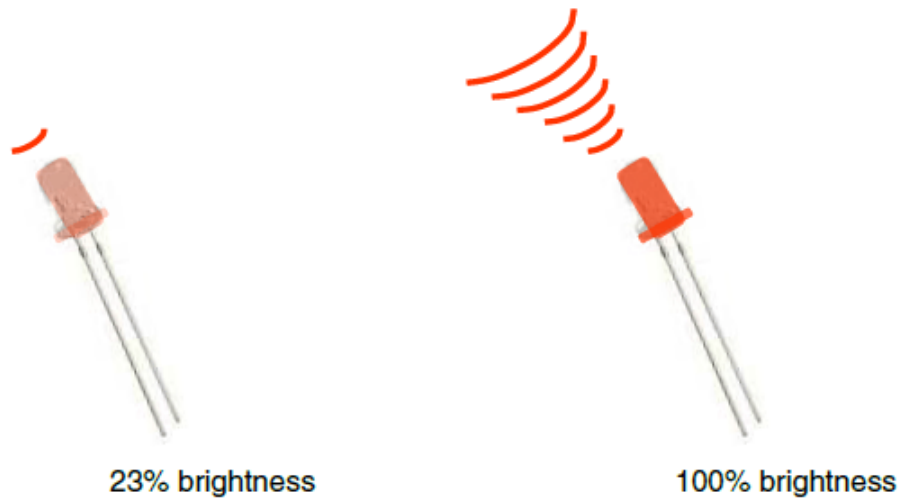
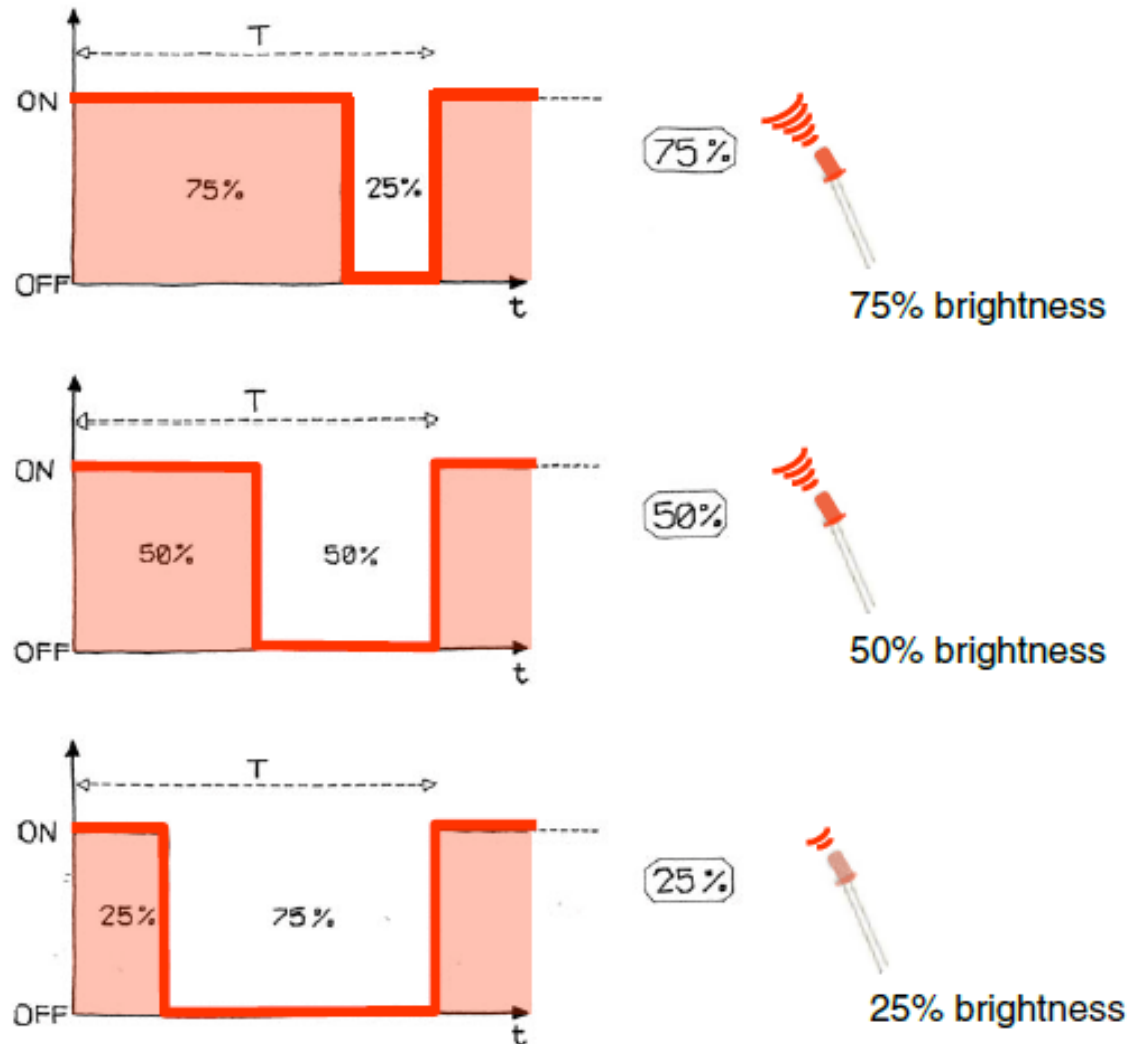


Image from *Theory and Practice of Tangible User Interfaces* at UC Berkley

- Analog output can be simulated using pulse width modulation (PWM)

Pulse Width Modulation

- Can't use digital pins to directly supply say 2.5V, but can pulse the output on and off really fast to produce the same effect
- On-off pulsing happens so quickly, the connected output device "sees" the result as a reduction in the voltage



PWM Duty Cycle

$$\text{Output voltage} = (\text{on_time} / \text{cycle_time}) * 5V$$

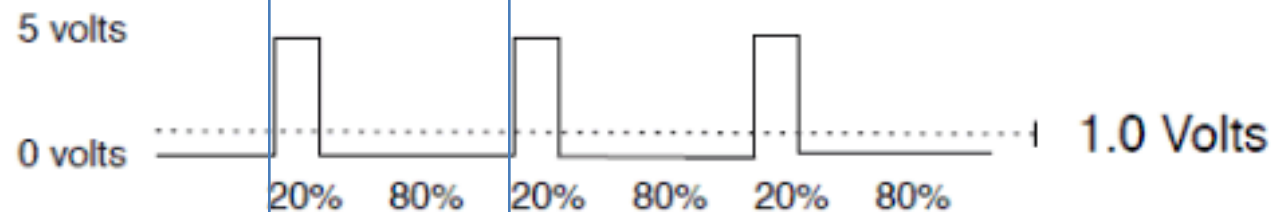
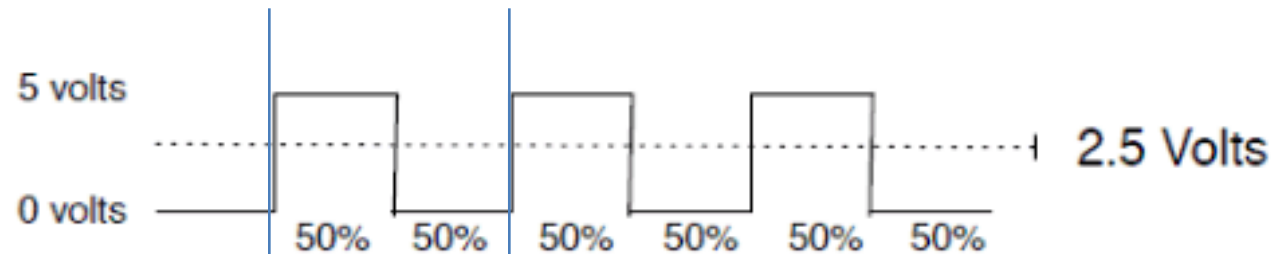
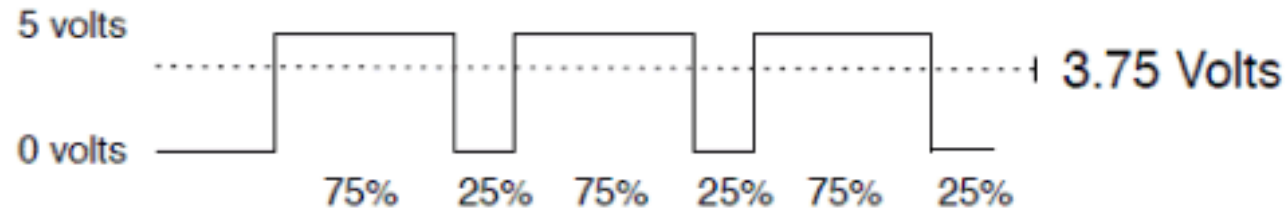


Image credit: Tod Kurt

Fixed cycle length; constant number of cycles/sec



Microprocessor vs. Microcontroller

- Microprocessor
 - Single chip semi-conductor device
 - But not a complete computer
 - Contains ALU, PC, timing & control units, registers
- Microcontroller
 - Functional computer **system-on-chip**
 - Contains microprocessor, memory, and program mable input/output peripherals
 - RAM, EEPROM
 - Timer, parallel I/O
 - ADC, DAC



What is ATmega2560?

- One of AVR 8-bit RISC **microcontrollers** by Atmel
- The acronym **AVR** has been reported to stand for
 - **A**dvanced **V**irtual **R**ISC and also for the chip's designers: **A**lf-Egil Bogen and **V**egard Wollan who designed the basic architecture at the Norwegian Institute of Technology
- RISC stands for **reduced instruction set computer**
: CPU design with a reduced instruction set as well as a simpler set of instructions (like for example PIC and AVR)



RISC vs. CISC

- RISC - Reduced Instruction Set Computer
 - The instruction set is small, and most instructions complete in one cycle (100 or less instruction types, smaller range of addressing modes).
 - Multiply & Divide performed using add/subtract & shift



RISC vs. CISC

- CISC - Complex Instruction Set Computer
 - The instruction set is large, and offers great variety of instructions (100 or more instruction types, many addressing modes).
 - Few instructions complete in one cycle
 - Typically includes multiply & divide operations that may take many cycles to complete.



Von Neumann vs. Harvard

- Von Neumann Architecture
 - The computer follows a step-by-step program that governs its operation.
 - The program is stored as data
 - No distinction between data and instructions.

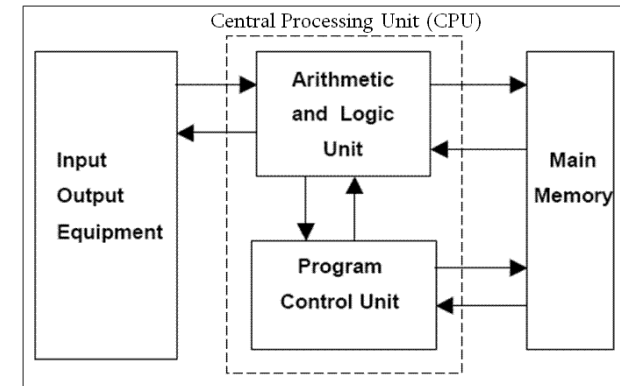
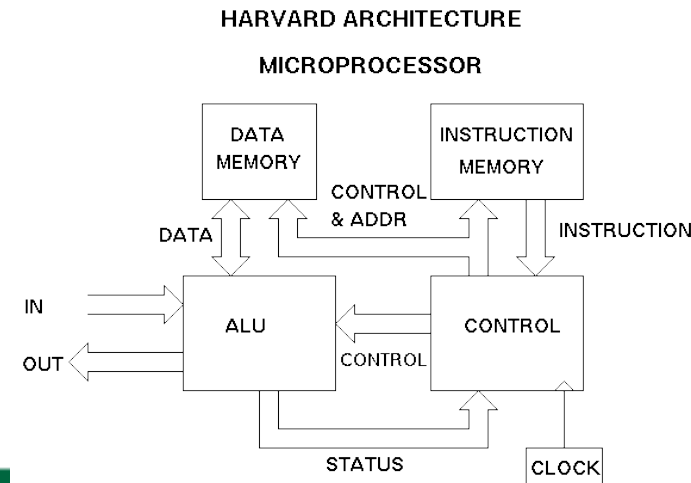


Figure : General structure of Von Neumann Architecture

- Harvard Architecture
 - Separate data and address spaces
 - The program is stored in its own address space



AVR 8-Bit RISC High Performance

- True single cycle execution
 - single-clock-cycle-per-instruction execution
 - PIC microcontrollers take 4 clock cycles per instruction
- One MIPS (mega instructions per second) per MHz
 - up to 20 MHz clock
- 32 x 8 bit general purpose registers
 - provide flexibility and performance when using high level languages
 - prevents access to RAM
- Harvard architecture
 - separate bus for program and data memory

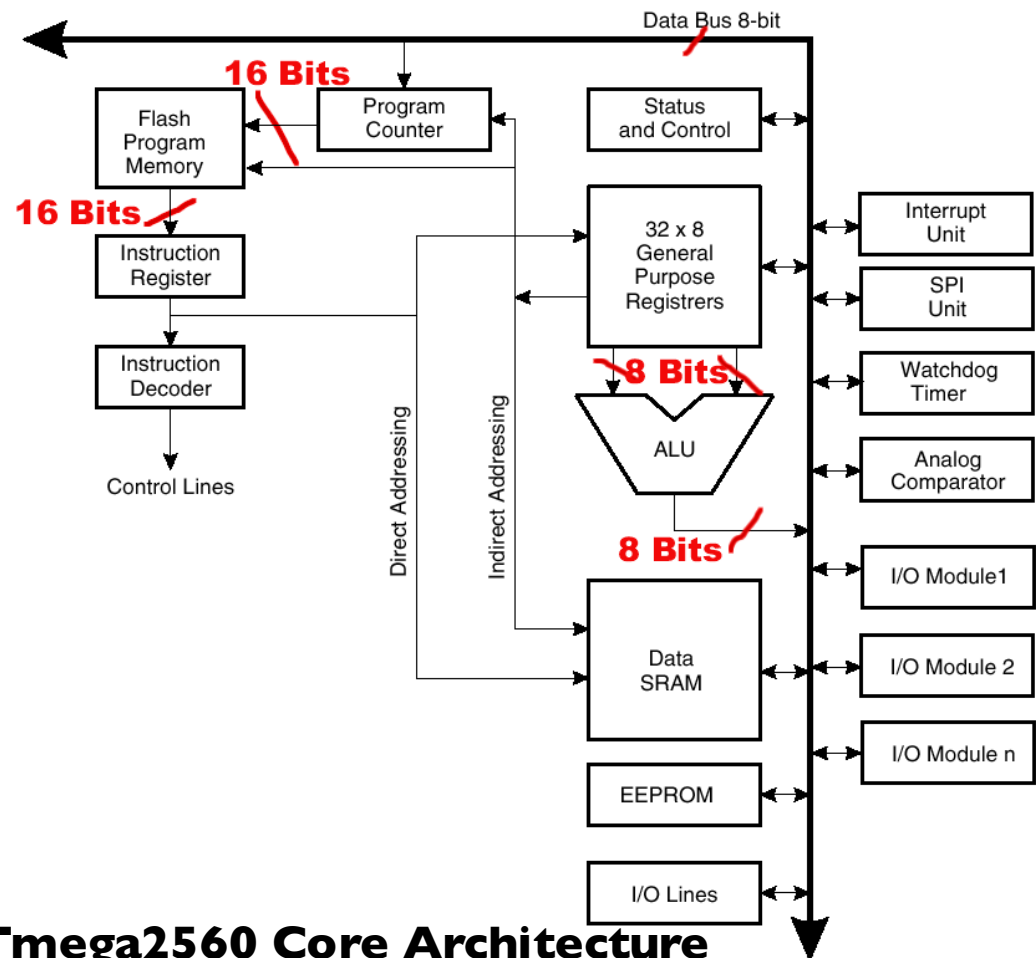


ATmega2560 CPU Architecture

Figure 3. Block Diagram of the AVR Architecture

Mega2560 CPU Core

- Seperate Instruction and Data Memories (Harvard)
- all 32 General Purpose Registers connected to ALU
- I/O Modules connected to Data Bus and accessible via Special Function Registers



ATmega2560 Core Architecture

Data Memory Map

Figure 8-2. Data Memory Map

Address (HEX)

0 - 1F

20 - 5F

60 - 1FF

200

21FF

2200

FFFF

32 Registers
64 I/O Registers
416 External I/O Registers
Internal SRAM (8192 × 8)
External SRAM (0 - 64K × 8)

AVR Register File

- The Register File
 - 32 8-bit registers

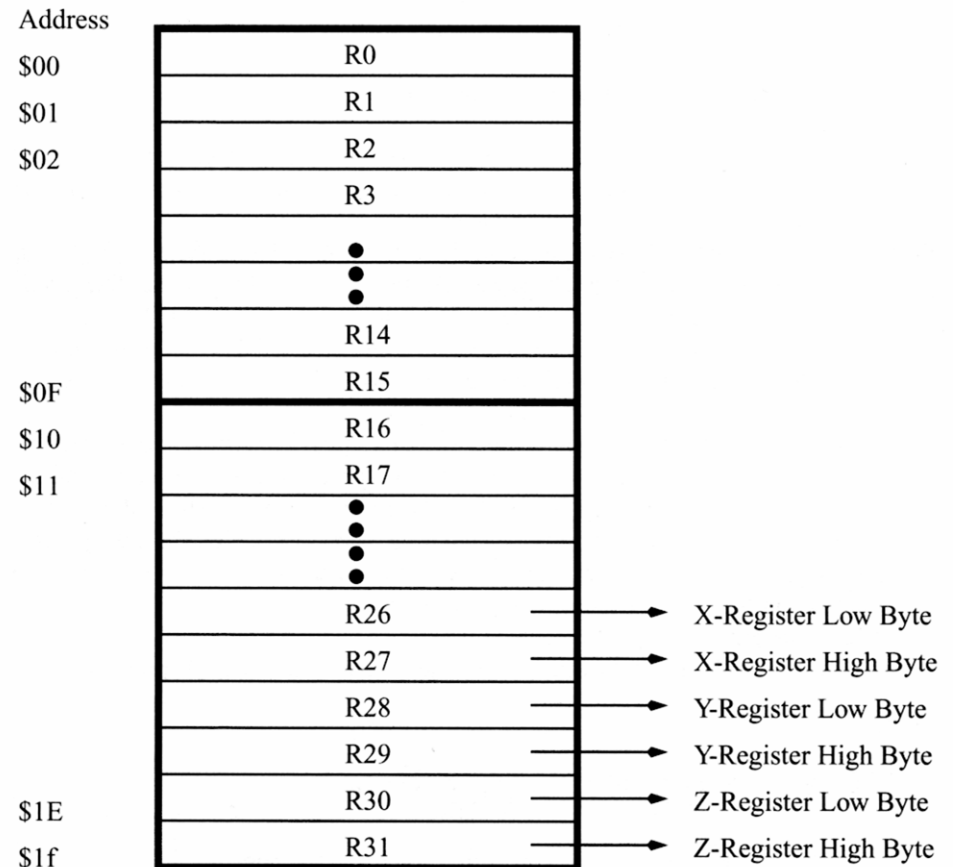


FIGURE 3.4 AVR register file.

Today

- Review from the last lecture
 - Analog vs. Digital, PWM
 - ATmega 2560
- Midterm: 2pm - 4:30pm, Mon Apr 11
- **ATmega2560 microcontroller (MCU) architecture**
 - CPU, Memory, I/O ports
- Announcement

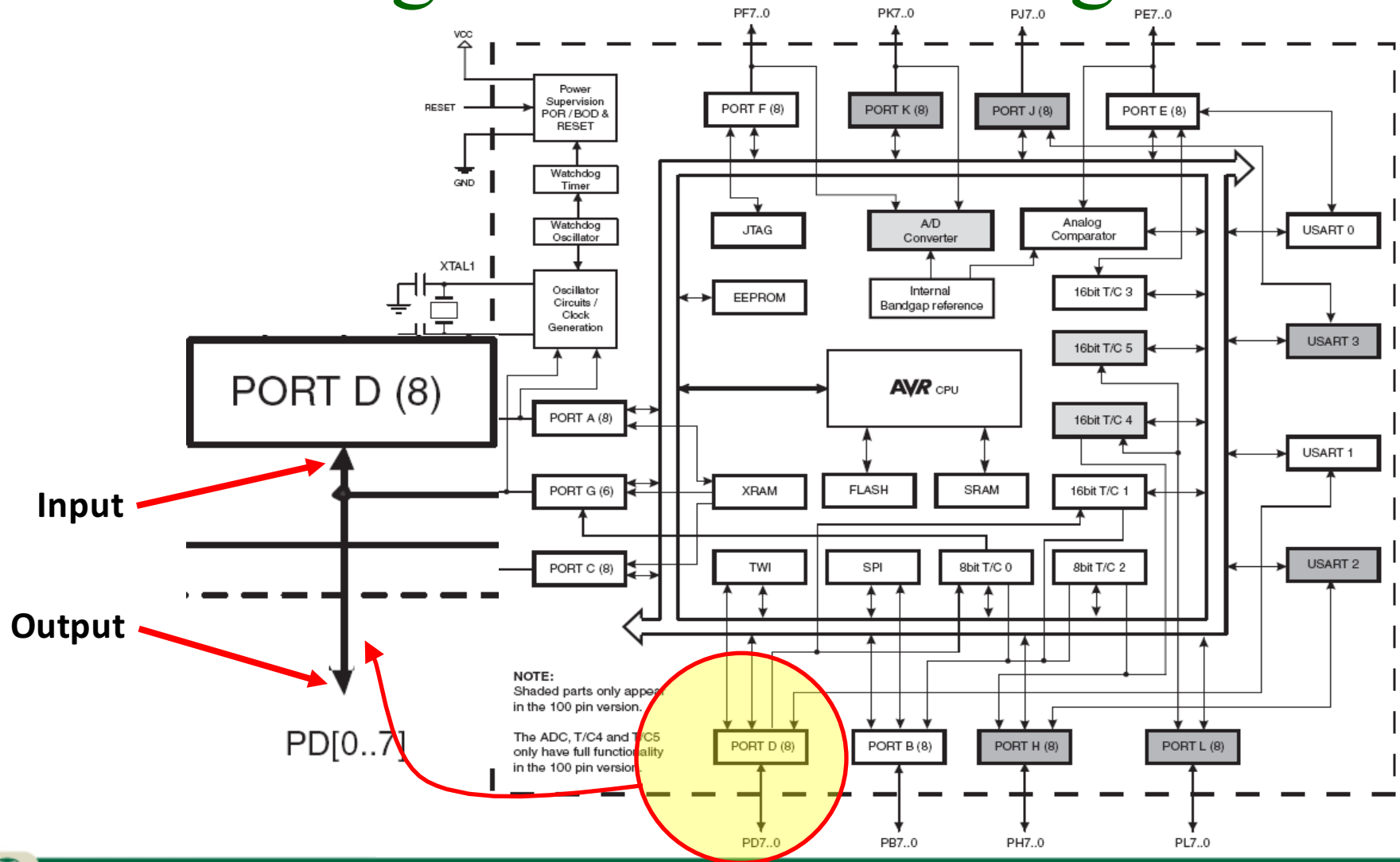


I/O Memory Registers

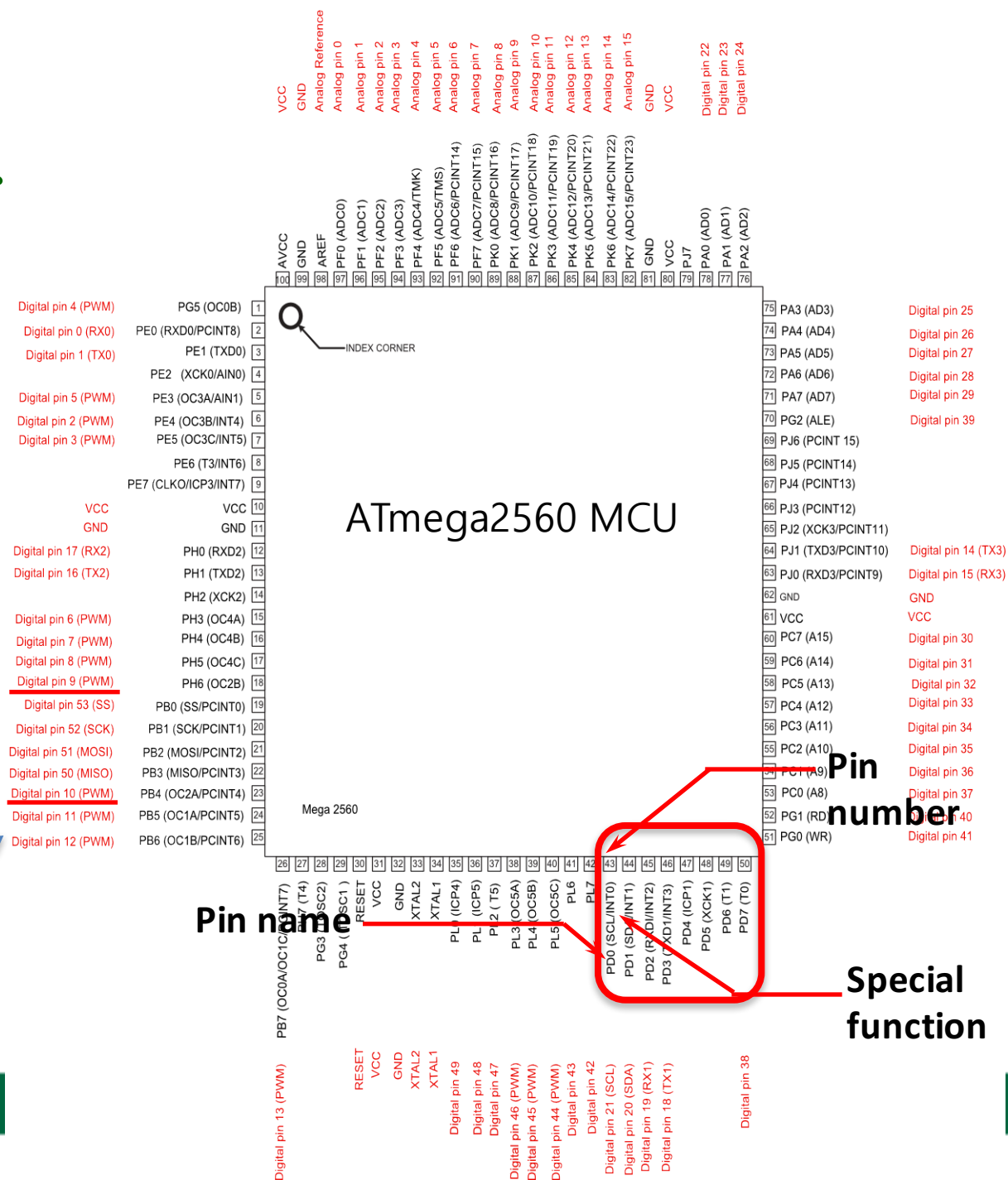
- | | |
|---|---|
| <ul style="list-style-type: none">• SREG: Status Register• SP: Stack Pointer Register• GIMSK: General Interrupt Mask Register• GIFR: General Interrupt Flag Register• MCUCR: MCU General Control Register• MCUSR: MCU Status Register• TCNTO: Timer/Counter 0 Register• TCCR0A: Timer/Counter 0 Control Register A• TCCR0B: Timer/Counter 0 Control Register B• OCR0A: Timer/Counter 0 Output Compare Register A• OCR0B: Timer/Counter 0 Output Compare Register B• TIMSK0: Timer/Counter 0 Interrupt Mask Register• TIFR0: Timer/Counter 0 Interrupt Flag Register• EEAR: EEPROM Address Register | <ul style="list-style-type: none">• EEDR: EEPROM Data Register• EECR: EEPROM Control Register• PORTB: PortB Data Register• DDRB: PortB Data Direction Register• PINB: Input Pins on PortB• PORTD: PortD Data Register• DDRD: PortD Data Direction Register• PIND: Input Pins on PortD• SPI I/O Data Register• SPI Status Register• SPI Control Register• UART I/O Data Register• UART Status Register• UART Control Register• UART Baud Rate Register• ACSR: Analog Comparator Control and Status Register |
|---|---|



ATmega2560 Block Diagram



Atmega2560 Microcontroller Pin Mapping to Arduino Board



Arduino ADK Board Pin Mapping

PORT Pin and Register Details

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	



Parallel I/O Ports

- Most general-purpose I/O devices
- Each I/O Port has 3 associated registers
 1. DDRx (where “x” is A, B, C...)
 - Data Direction Register Port x
 - Determines which bits of the port are input and which are output

```
DDRB = 0x02; /* sets the second lowest of port B to output */
```
 2. PORTx
 - Port Driver Register **for write**

```
PORTB = 0x02; /* sets the second bit of port B and clears the others */
```
 3. PINx
 - Port Pins Registers **for read**
 - Returns the status of all 8 port B pins.

```
unsigned int x;  
x = PINB; /* Places the status of port B into variable x */
```



Input/Output Ports

- All ports initially set to input
 - `DDRB = 0x00; /* by default */`
- Must declare all output pins using `DDRx` (Data Direction Registry Port x)
- The default for input port pins is *floating*
Can supply a pull-up resistor by writing logic 1 to the corresponding bit of the port driver register
 - `DDRA = 0xC0; /* upper 2 bits are output, lower 6 bits are input*/`
 - `PORTA = 0x03; /*enable internal pull-ups on lowest 2 bits*/`
- Port pins in output mode are typically capable of sinking 20 mA, but source much less



Data Direction Register (DDR)

- If the bit is zero -> pin will be an input
 - Making a bit to be zero == ‘**clearing** the bit’
- If the bit is one -> pin will be an output
 - Making a bit to be one == ‘**setting** the bit’
- To change the data direction for a set of pins belonging to PORTx at the same time:
 1. Determine which bits need to be set and cleared in DDRx
 2. Store the binary number or its equivalent (in an alternate base, such as hex) into DDRx



Bitwise Operations

- Treat the value as an array of bits
- Bitwise operations are performed on pairs of corresponding bits

X = 0b0011, Y = 0b0110

Z = X | Y = 0b0111

Z = X & Y = 0b0001

Z = X ^ Y = 0b0101

Z = ~X = 0b1100

Z = X << 1 = 0b0110

Z = x >> 1 = 0b0001



Bit Masks

- Need to access a subset of the bits in a variable
 - Write or read
- Masks are bit sequences which identify the important bits with a '1' value
- Ex. Set bits 3 and 5 or X, don't change other bits
 $X = 01010101$, $\text{mask} = 0010100$
 $X = X \mid \text{mask}$
- Ex. Clear bits 2 and 4
 $\text{mask} = 11101011$
 $X = X \& \text{mask}$



Bit Assignment Macros

```
#define SET_BIT(p,n) ((p) |= (1 << (n)))
```

```
#define CLR_BIT(p,n) ((p) &= ~(1 << (n)))
```

- $1 \ll n$ and $\sim(1 \ll n)$ create the mask
 - Single 1 (0) shifted n times
- Macro doesn't require memory access (on stack)



Example 1

- Make Arduino pin 10 (PB4) to be output
 - Arduino pin 10 connected to SmartCAR front LED

- Arduino approach

```
pinMode(10, OUTPUT);
```

- Alternative approach

```
DDRB = 0b00010000;
```

or

```
DDRB = 0x10;
```

or

```
DDRB |= (1 << PB4);
```



Example 2

- Make pins Arduino pins 0 and 1 (PE0 and PE1) inputs, and turn on initially (enabling pull-up R)

- Arduino approach

```
pinMode(0, INPUT);  
pinMode(1, INPUT);  
digitalWrite(0, HIGH);  
digitalWrite(1, HIGH);
```

- Alternative approach

```
DDRE = 0; // all PORTE pins inputs  
PORTE = 0b00000011;  
or  
PORTE = 0x03;
```

or better yet:

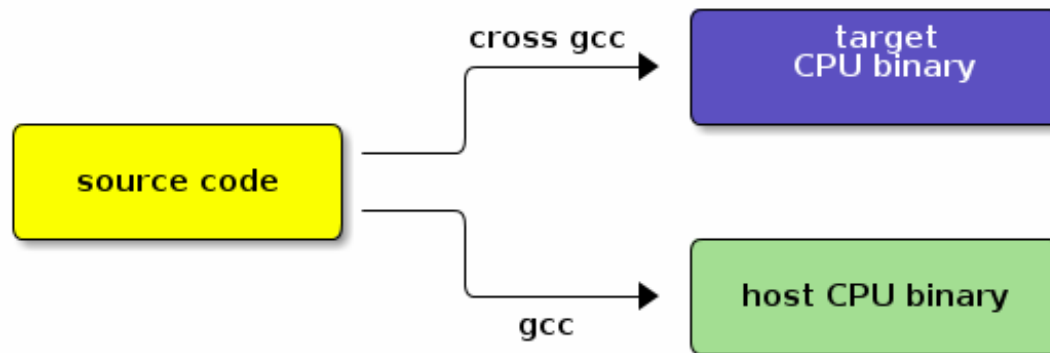
```
DDRE &= ~(1 << PE1 | 1 << PE0);  
PORTE |= (1 << PE1 | 1 << PE0);
```

QUESTION)
WHAT IS CROSS-COMPILER
IN EMBEDDED SYSTEM?



Cross-Compiler

- A compiler which generates code for a platform different from the one it executes on
 - Executes on host, generates code for target
- Generates an **object file (.o)**
- Contains machine instructions
- **References are virtual**
 - Absolute addresses are not yet available
 - Labels are used instead



Course Announcement

- For lab session, we will cover
 - SmartCAR Motor Control
- Next week, we will study on
 - ATmega2560 microcontroller (MCU) architecture
 - Timer & Interrupts

