



Introduction to the Ardunio

Microprocessor

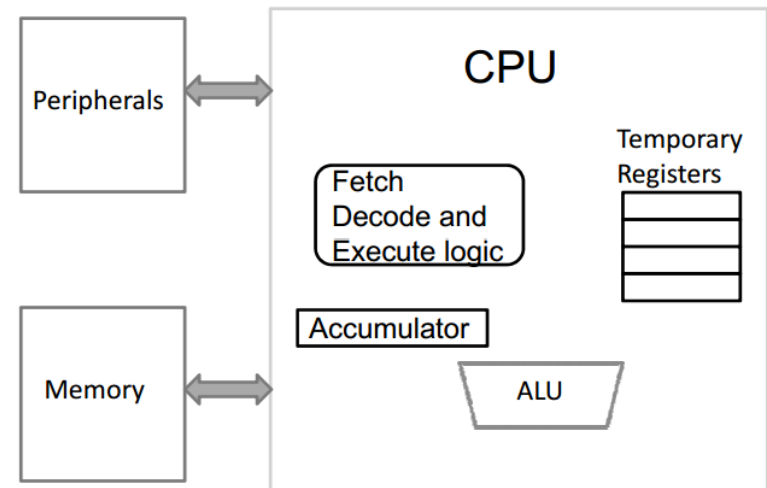
Microprocessor

an integrated circuit that contains all the functions of a central processing unit of a computer.

Microprocessor(uP)

Function of Microprocessor

- Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division.
- A microprocessor can move data from one [memory](#) location to another.
- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.



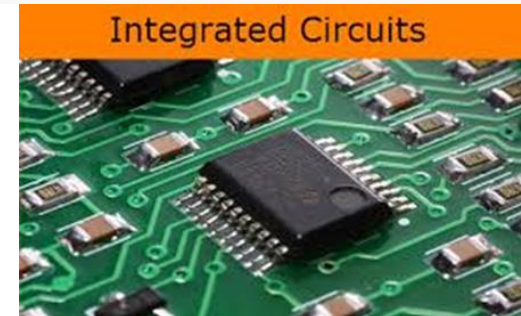
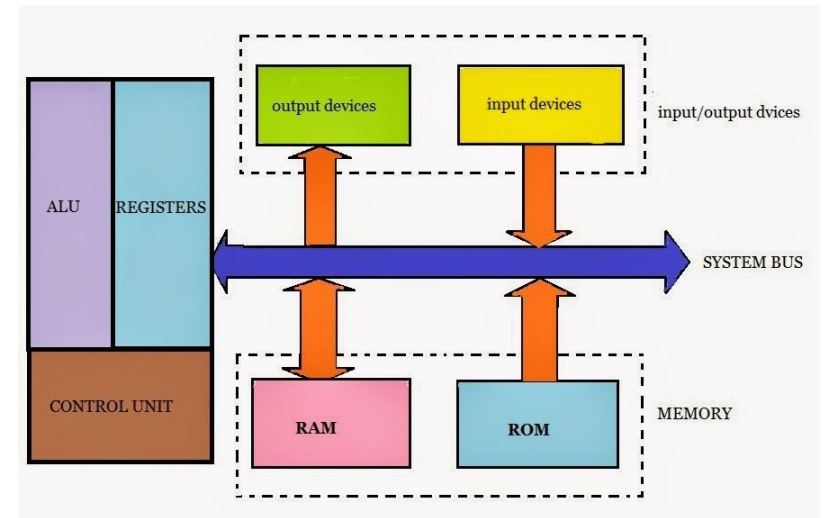
Microcontroller

Microcontroller

A Microcontroller is a **IC chip** that executes programs for controlling other devices or machines. It is a micro (small size as its a **Integrated Circuit** chip) device which is used for control of other devices and machines that's why it is called 'Microcontroller'. It is a **Microprocessor** having RAM,ROM and I/O ports.

Function of Microcontroller

- Same function as microprocessor.
- The majority of **microcontrollers** in use today are embedded in other machinery, such as automobiles, telephones, appliances, and peripherals for computer systems



Difference between Microprocessor and Microcontroller

- Key **difference in** both of them is presence of external peripheral, where **microcontrollers** have RAM, ROM, EEPROM embedded **in** it while we have to use external circuits **in** case of **microprocessors**.
- As all the peripheral of **microcontroller** are on single chip it is compact while **microprocessor** is bulky.

Advantage of Microcontroller compared to Computer

- Key **difference** in both of them is presence of external peripheral, where **microcontrollers** have RAM, ROM, EEPROM embedded **in** it while we have to use external circuits **in** case of **microprocessors**.
- As all the peripheral of **microcontroller** are on single chip it is compact while **microprocessor** is bulky.

Arduino

- What is Arduino?
 - Open-source platform used for building electronics projects
 - Consists of a physical programmable circuit board
 - IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board

Arduino

- Why is Arduino popular?
 - Does not need a separate piece of hardware (called a programmer) in order to load new code onto the board -- you can simply use a USB cable.
 - Arduino IDE uses a simplified version of C, making it easier to program (**we will see later**).
 - Several daughter boards are available to develop a project instantly

A large collection of electronic components and modules for an Arduino Uno project. The items include: various sensors (ultrasonic, temperature, light, distance, etc.), actuators (servo motors, solenoid, relay, etc.), cables (USB, power, jumper), a breadboard, a CD-ROM labeled 'Complete Starter Kit for UNO Project', and a variety of other electronic components like resistors, capacitors, and integrated circuits. The components are arranged in a grid-like fashion, showcasing the wide range of options available for the kit.

Arduino

- What is inside Arduino?

A microcontroller

Arduino

- Microcontrollers are basically a memory where you can store sequence of operations
 - Includes both logical and mathematical operations
 - Stored in 8 or 16 bit registers as memory

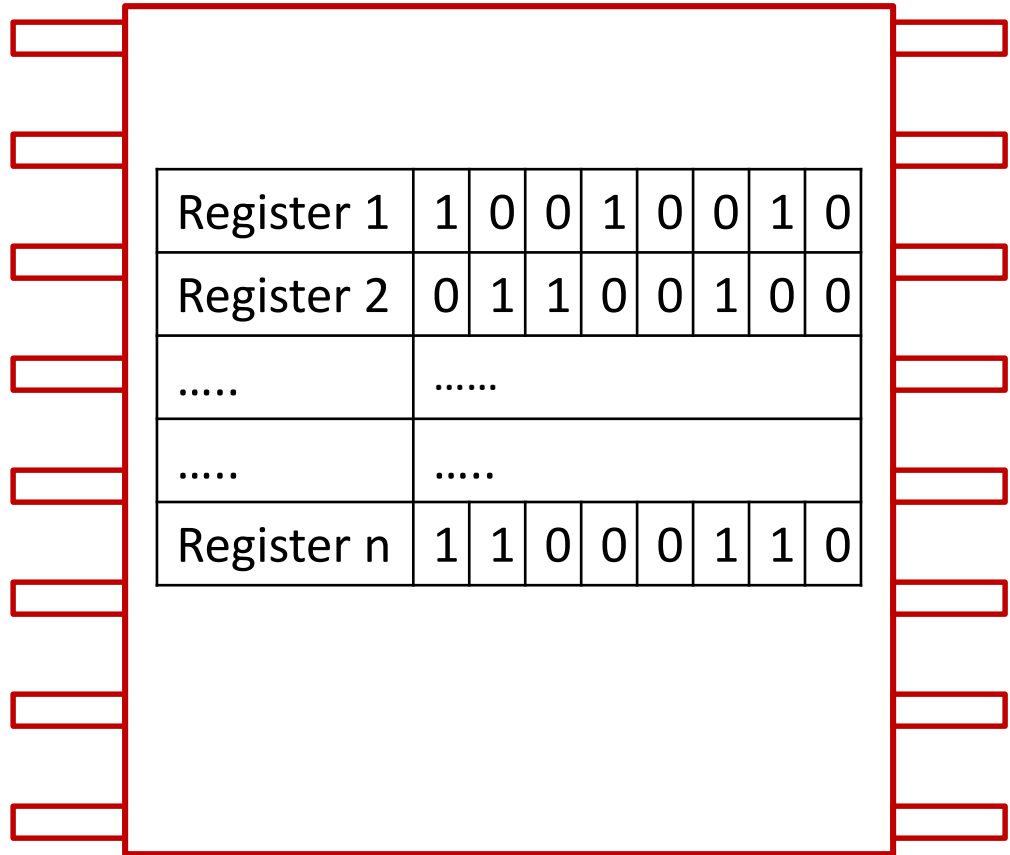
MSB							LSB
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	0	0	1	0	1	0	0

An 8 bit register

Arduino

A **register** is a place in a CPU that can store the data used for performing various operations such as addition and multiplication and loads the resulting data on main memory.

External Pins

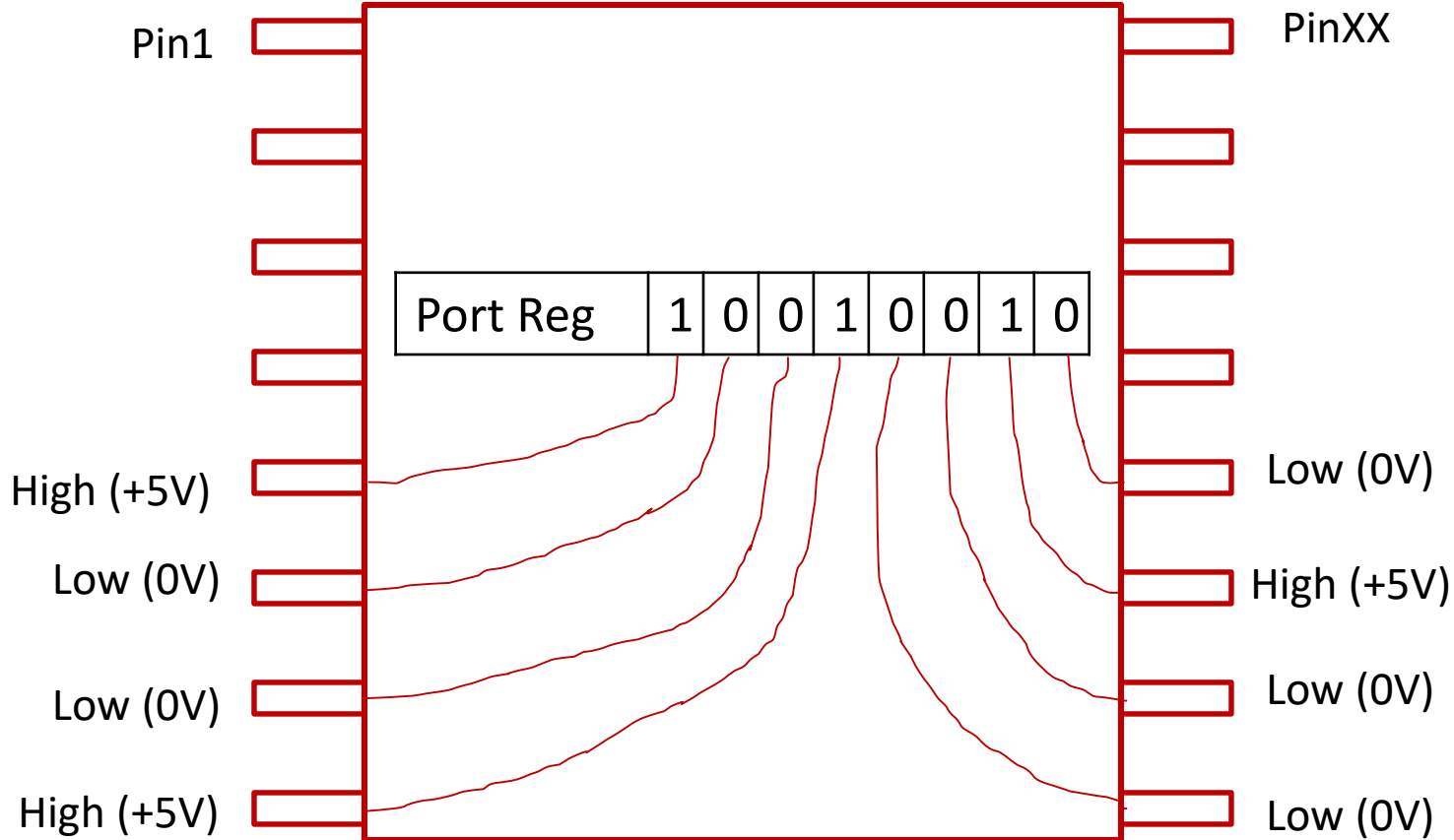


External Pins

Inside a Microcontroller Chip

Arduino

A **port** is a group of **pins** representing a standard interface. The pin can be configured as **1(High)** for **input** and **0 (Low)** for **output** as per the logic state. Input/output pins allow the **microcontroller** to be connected with the peripheral devices.



Microcontroller Chip - Ports

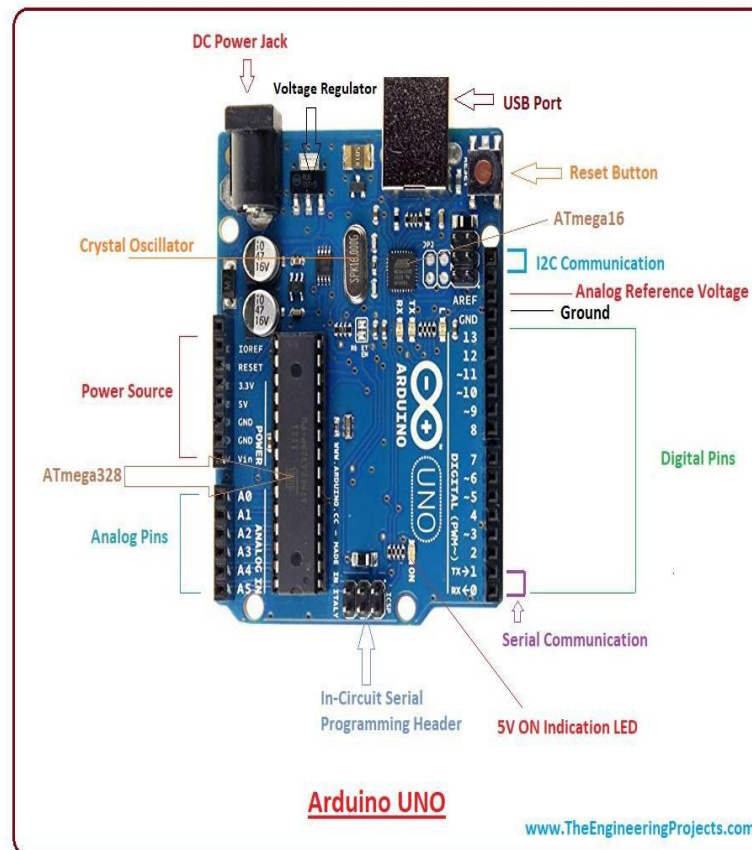
Arduino

- Which microcontroller is inside Arduino?
- -ATmega 328P



Why ATmega328 is used in Arduino?

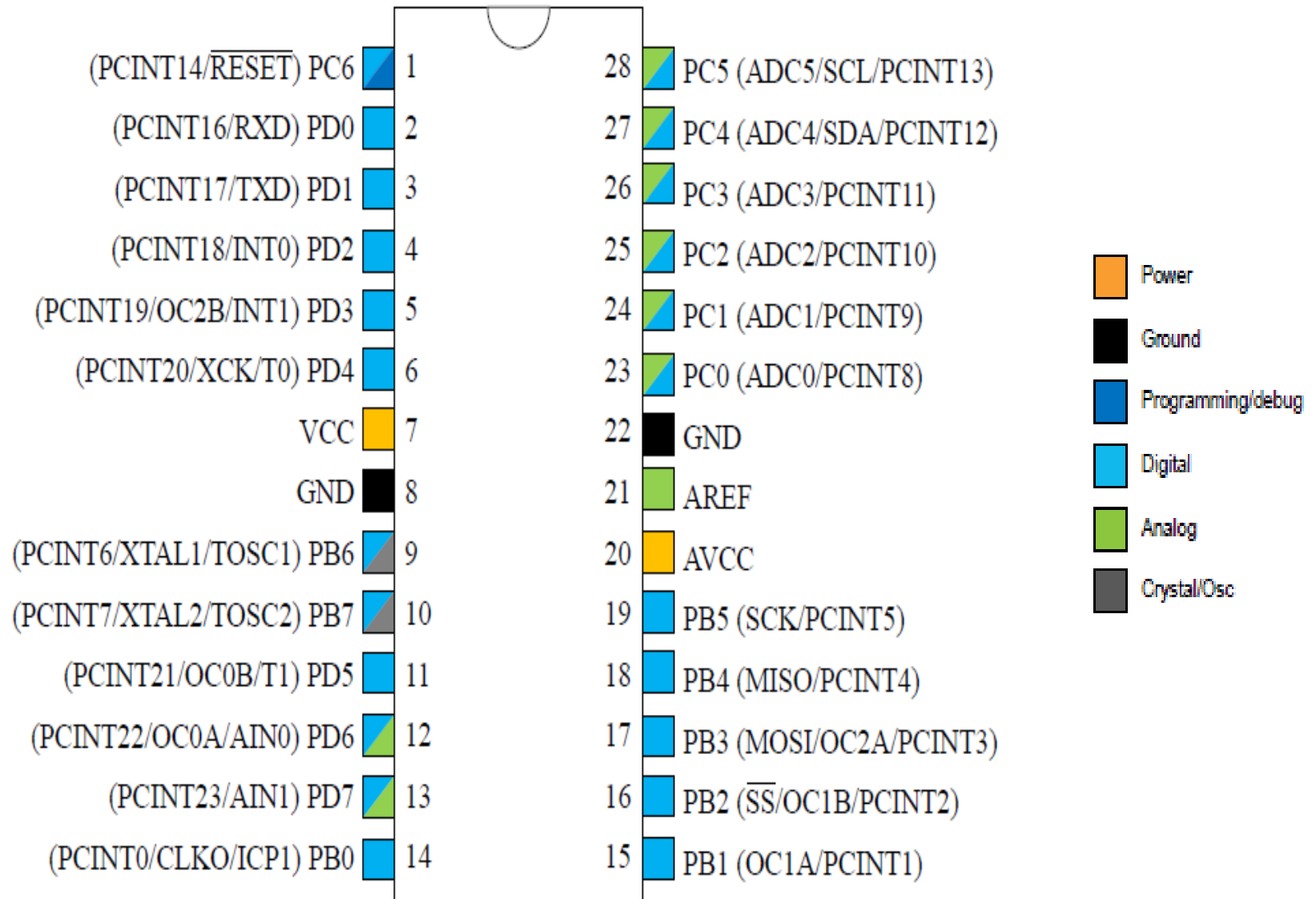
Arduino Uno is a microcontroller board based on the **ATmega328P** (datasheet). It has 14 digital input/output pins, 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an **ICSP header** and a reset button.



ATmega328

- Each pin of ATMEGA 328 performs several functions.
- Accordingly these pins are named
- Example: A pin with Label PC3 means it will act as PORTC3 I/O

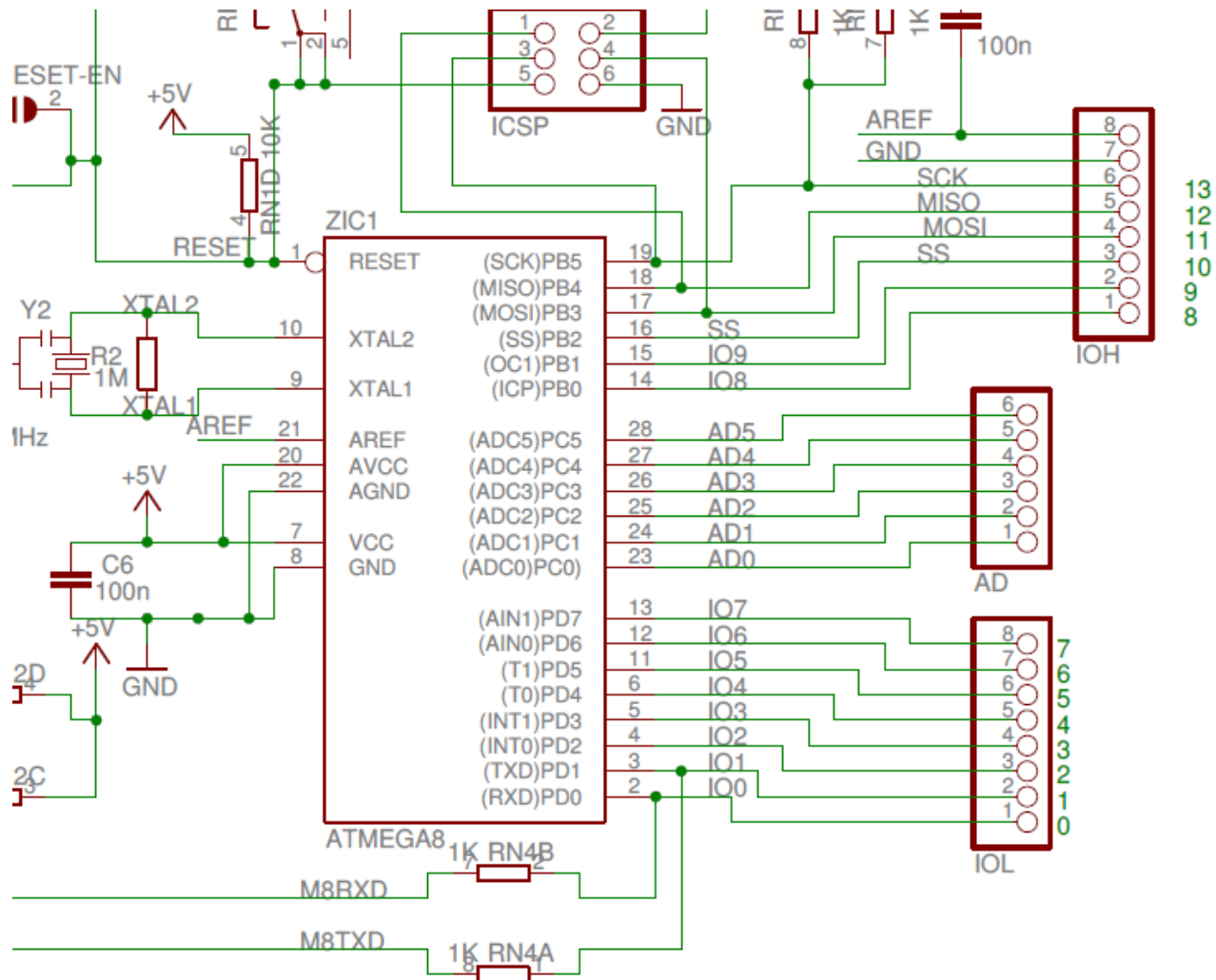
PIN Configurations Atmega328




ATMEGA Pins and Arduino

ATMEGA 328 Pin	Pin Name	Arduino board pin
23	PC0/ADC0	A0 (A means analog enabled pin)
24	PC1/ADC1	A1 (A means analog enabled pin)
15	PB1/OC1A	~9 (the ~ sign means PWM enabled pin)
19	PB5	13

Circuit Diagram



Program: To Glow LED



The screenshot shows the Arduino IDE interface. The title bar reads "sketch_may28a | Arduino 1.8.9". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The toolbar contains icons for checking, running, saving, and uploading. The code editor displays the following C++ code:

```
void setup()
{
  // To Glow LED:
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Below the code editor, there is a URL: <https://www.arduino.cc/en/Main/Software>. The status bar at the bottom indicates "15" on the left and "Arduino/Genuino Uno on COM3" on the right.

Arduino Approach

ARDUINO 1.8.9

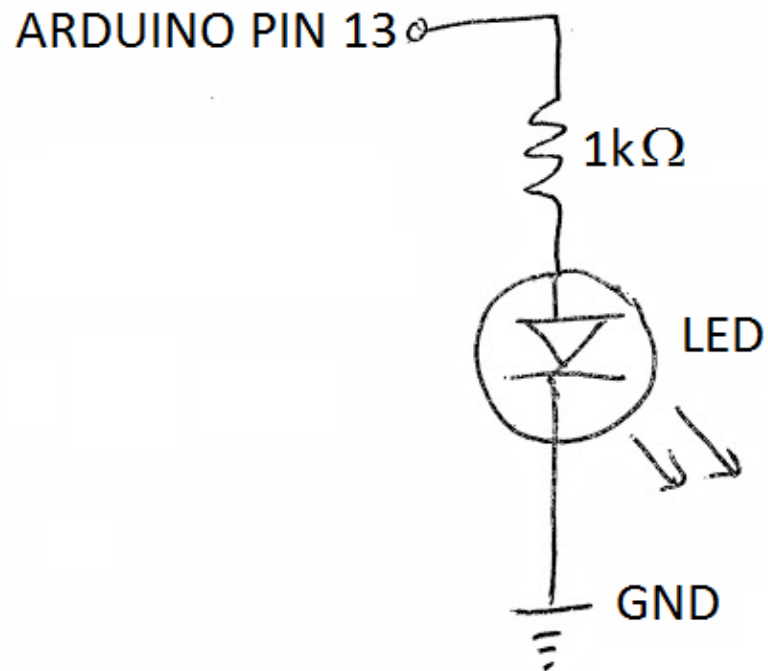
But you don't know what is happening inside

Program: To Glow LED

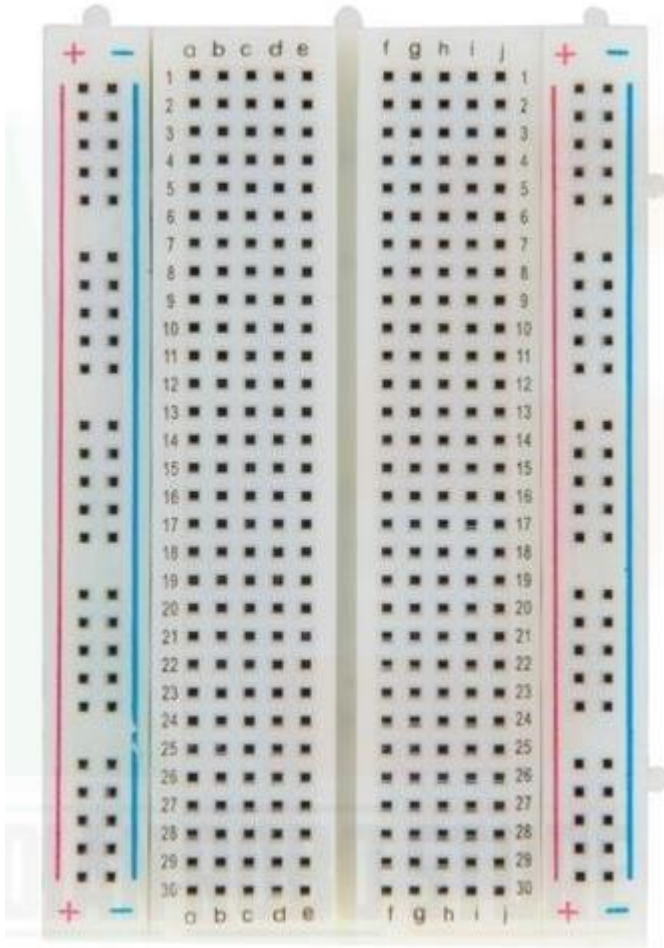
```
void setup()
{
  pinMode(LED_BUILTIN,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN,HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN,LOW);
  delay(1000);
}
```

Prepare Hardware



Components



Breadboard



Resistance

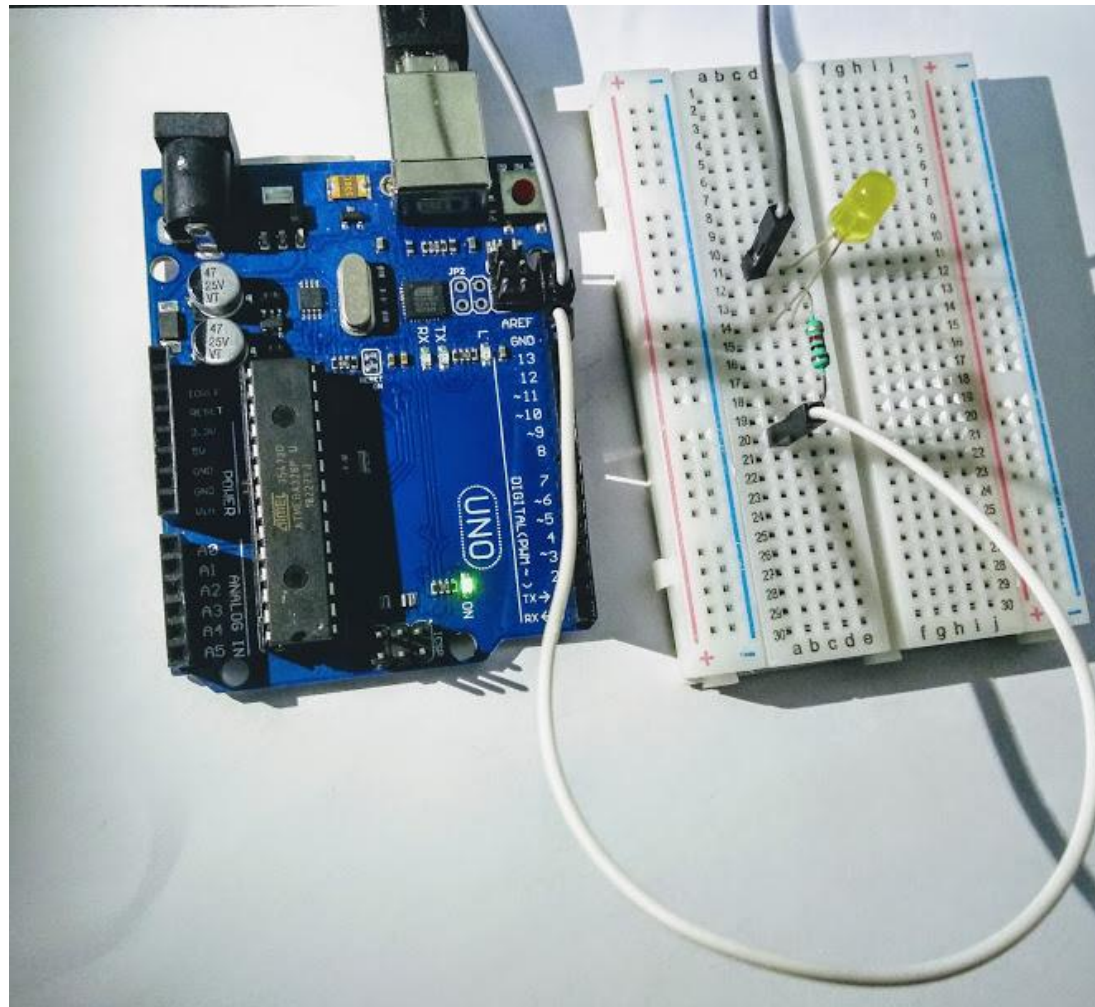


LED

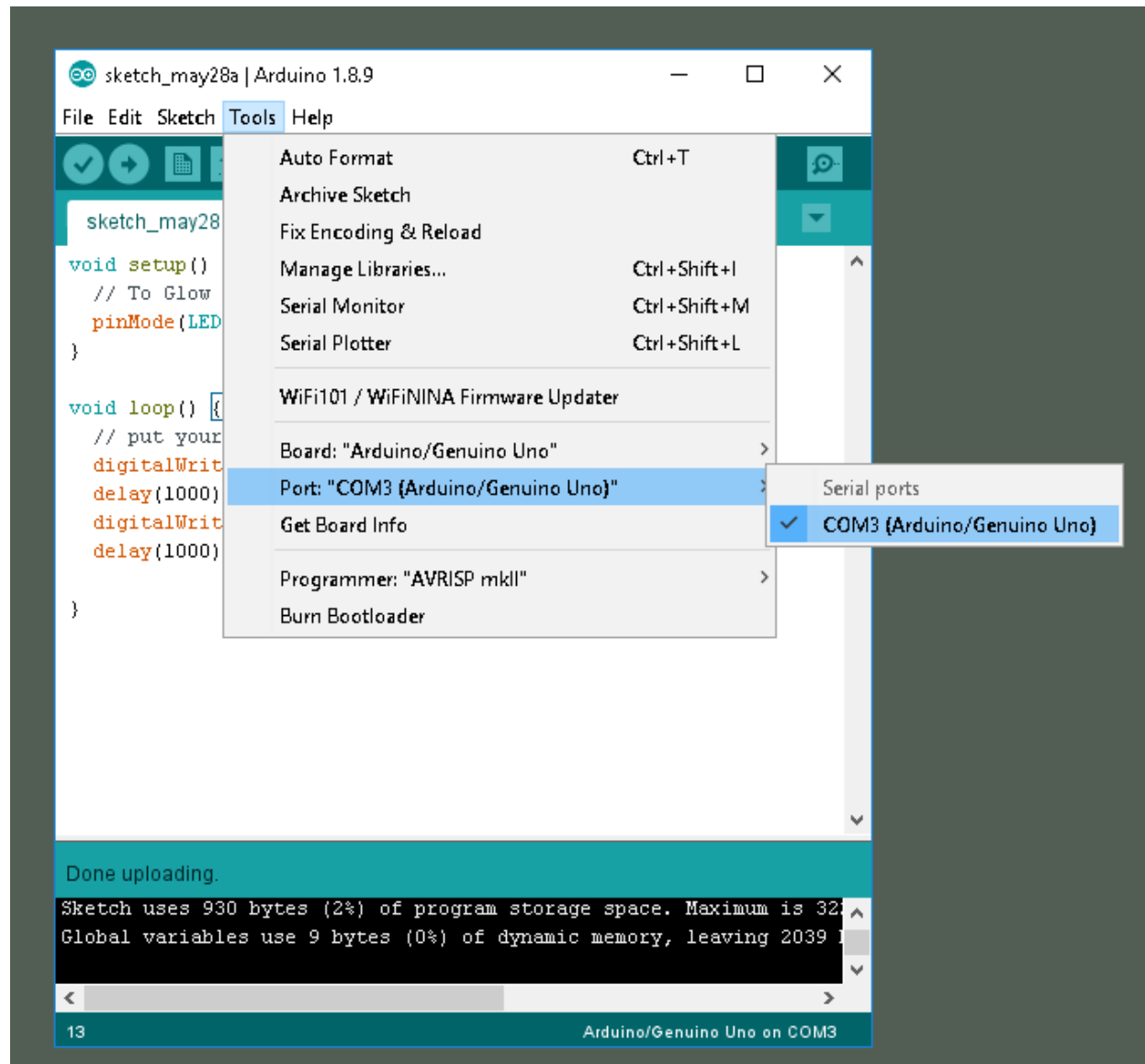


Jumper Wire

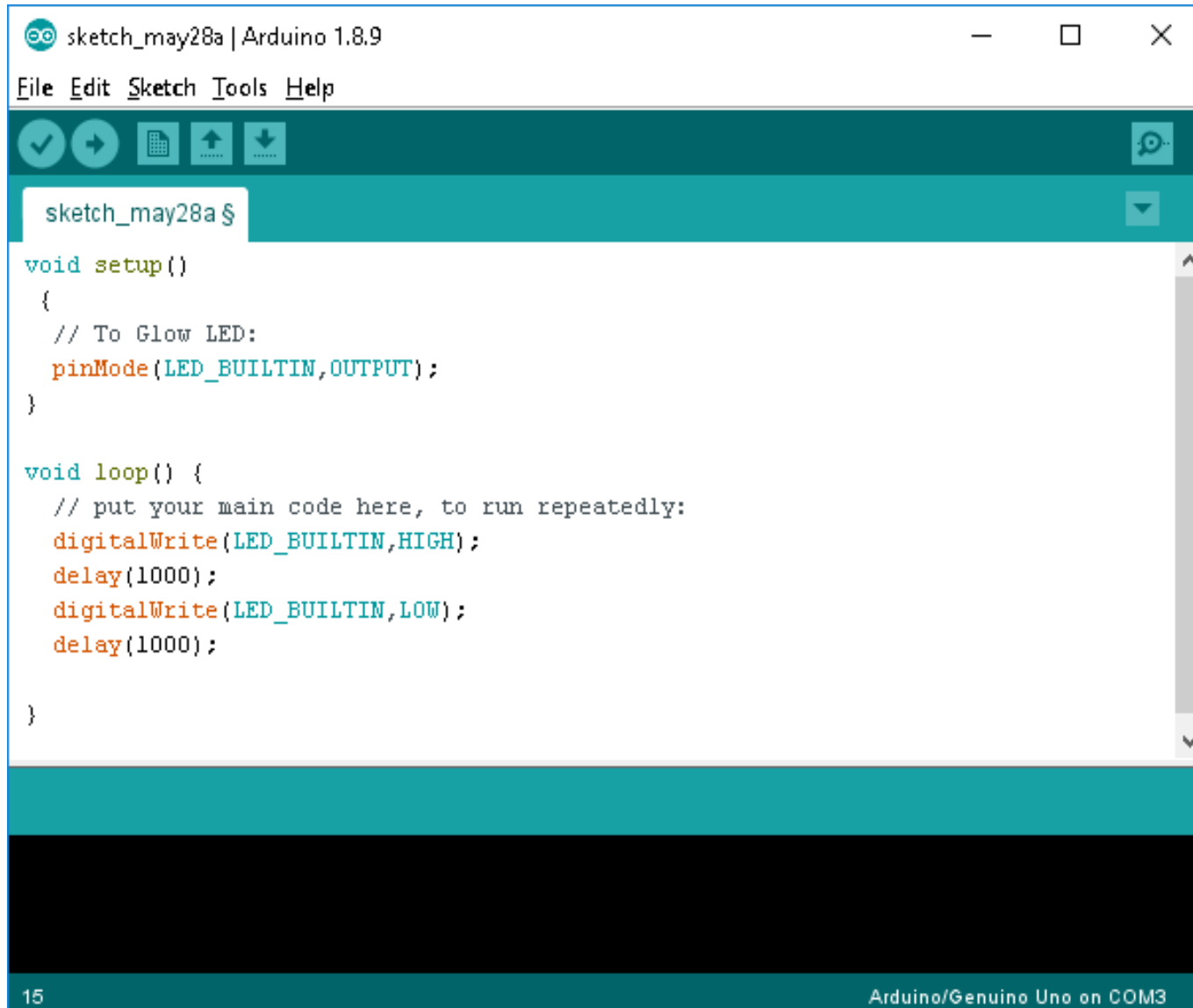
Hardware for LED



Computer Communication Setting



Compile and Run



The screenshot shows the Arduino IDE window titled "sketch_may28a | Arduino 1.8.9". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar contains icons for checking, running, uploading, and downloading. The sketch name "sketch_may28a" is displayed in the top left of the editor area. The code in the editor is as follows:

```
void setup()
{
  // To Glow LED:
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

The bottom status bar shows "15" on the left and "Arduino/Genuino Uno on COM3" on the right.

Register Level Programming

- For Arduino programming many things are hidden.
- We will focus on register level programming.
- For register level programming, some standard syntax needs to be known to put a value to a particular register, set or clear a bit of a register.

Register WRITE

- The simplest is to write a value to a register from datasheet

Example:

PORTB = 0x01

Hex

or, PORTB = 1

Decimal

or, PORTB = 0b00000001

Binary

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	0	0	1

How to Set a bit of PORT

- `bitSet(x, n)`

x: the numeric variable whose bit to set.

n: which bit to set, starting at 0 for the least-significant (rightmost) bit

Example:

`bitSet(PORTB, 5)`

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	1	0	0	0	0	0

How to Clear a bit of PORT

- `bitClear(x, n)`

x: the numeric variable whose bit to set.

n: which bit to clear, starting at 0 for the least-significant (rightmost) bit

Example:

`bitClear(PORTB, 5)`

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0	0	0	0	0	0	0	0

How to Read a bit of PORT

- `bitRead(x, n)`

x: the number from which to read.

n: which bit to read, starting at 0 for the least-significant (rightmost) bit.

Example:

`bitRead(PORTB, 5)`

PORT Pin Control

- Each port pin is controlled by data direction register: DDRx, where x is port letter, e. g. DDRB

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

- DDRx Register selects the direction of n-th pin of a PORT.
- If n-th bit of DDRx is written logic one, n-th pin of a PORT is **configured as an output pin**.
- If n-th bit of DDRx is written logic zero, n-th pin of a PORT is configured as an input pin.

PORT Pin Control

- Example:
bitSet(DDRB,5);
or, we can configure directly like `DDRB = 0b00100000;`

DDRB Register

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0

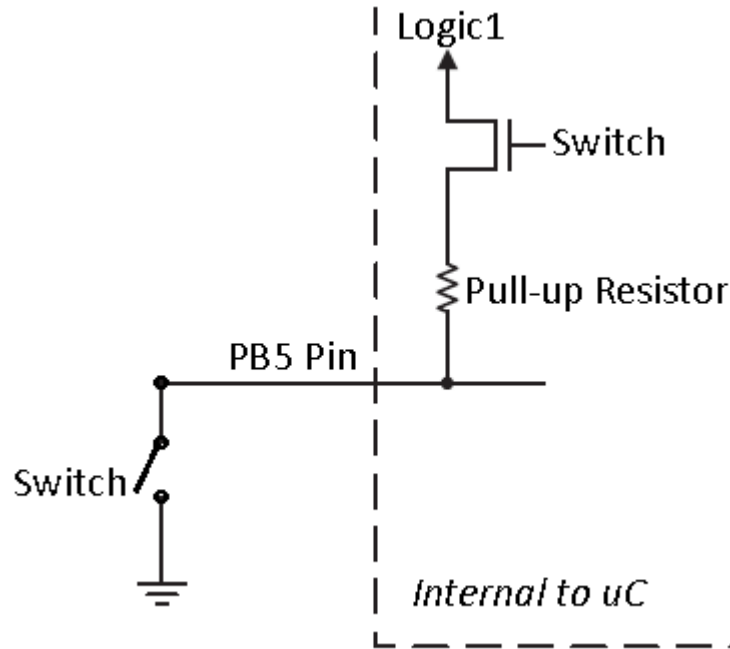
PORTB

Pin Direction

7	6	5	4	3	2	1	0
IN	IN	OUT	IN	IN	IN	IN	IN

PORT PIN Internal Circuit

- If n-th pin of PORTx is written logic one when the pin is configured as an input pin by DDRx register, the pull-up resistor is activated and vice-versa



PORT Pin Used as Input PIN

- Example:

`bitClear(DDRC, 3);` PC3 configured as input pin

`bitSet(PORTC, 3);` Pullup resistor of PC3 is now enabled

`bitClear(PORTC, 3);` Pullup resistor of PC3 is now disabled

PORT Pin Used as Output PIN

- If n-th pin of PORTx is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If n-th pin of PORTx is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

PORT Pin Used as Output PIN

- Example:

`bitSet(DDRC,3);` PC3 configured as output pin

`bitSet(PORTC,3);` PC3 is now high

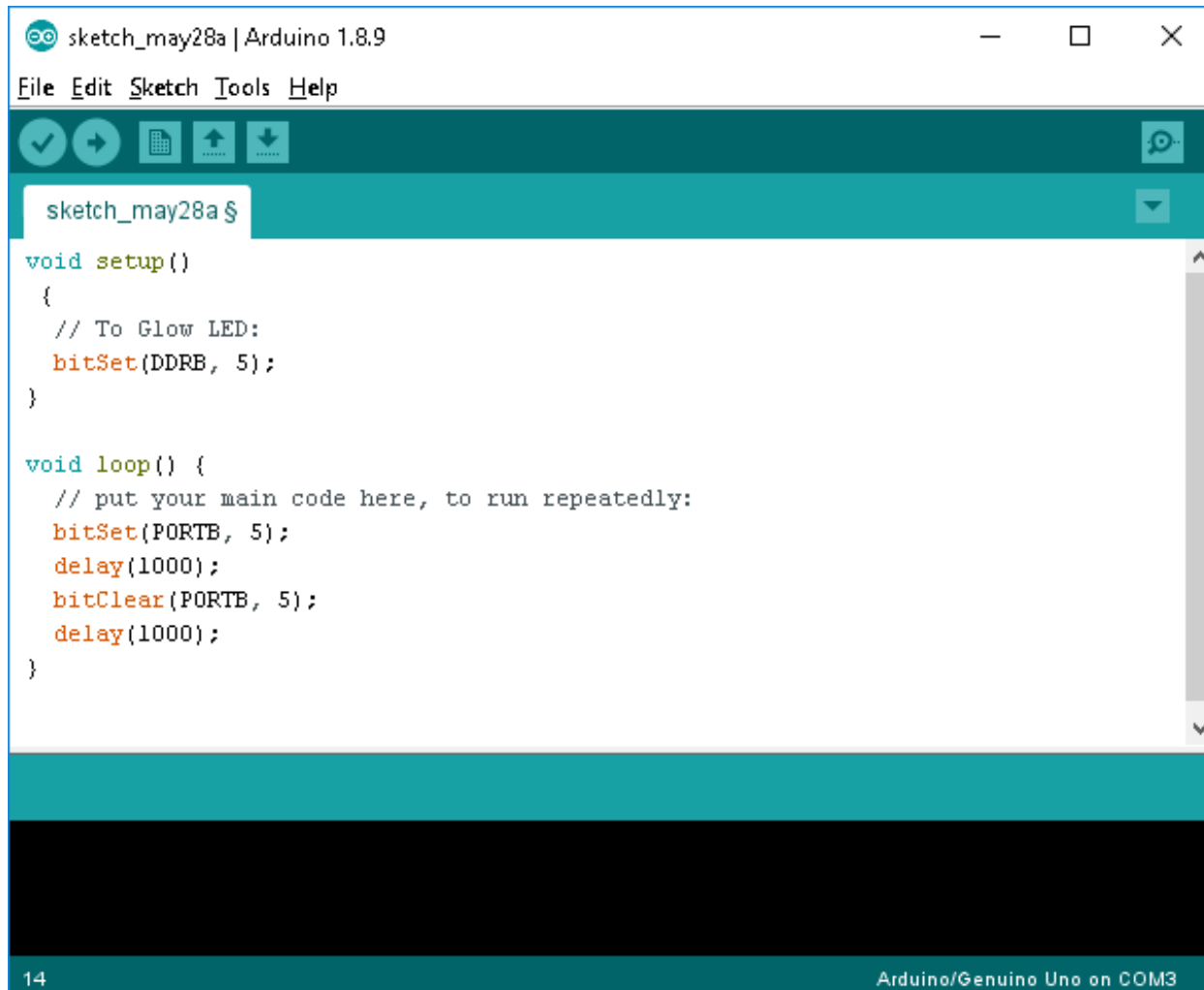
`bitClear(PORTC,3);` PC3 is now low

ATmega Program

```
void setup()
{
    // To Glow LED:
    bitSet(DDRB, 5); // Data direction Register, PORTB 5 as output
}

void loop() {
    // put your main code here, to run repeatedly:
    bitSet(PORTB, 5); // Make PORTB 5 pin high
    delay(1000); // wait for 1000ms
    bitClear(PORTB, 5); // Make PORTB 5 low
    delay(1000); // wait for 1000ms
}
```

ATmega Program



```
sketch_may28a | Arduino 1.8.9
File Edit Sketch Tools Help

sketch_may28a $

void setup()
{
  // To Glow LED:
  bitSet(DDRB, 5);
}

void loop() {
  // put your main code here, to run repeatedly:
  bitSet(PORTB, 5);
  delay(1000);
  bitClear(PORTB, 5);
  delay(1000);
}
```

14 Arduino/Genuino Uno on COM3

ATMEGA Program

- We are using delay in our program
- Otherwise the blinking of the LED will be too fast to be perceived
- As we have noticed, we are using internal delay routine of Arduino software
- Now, we will replace the internal delay routine of Arduino with internal hardware timer of ATMEGA328 chip

ATMEGA Timer

- Timers
- There are several 8-bit and 16-bit timers in ATMEGA328
- Naming: Timer0, Timer1, etc.
- We will learn Timer1 (16-bit)

Timer/Counter Control Register

TCCR1A – Timer/Counter1 Control Register A

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	

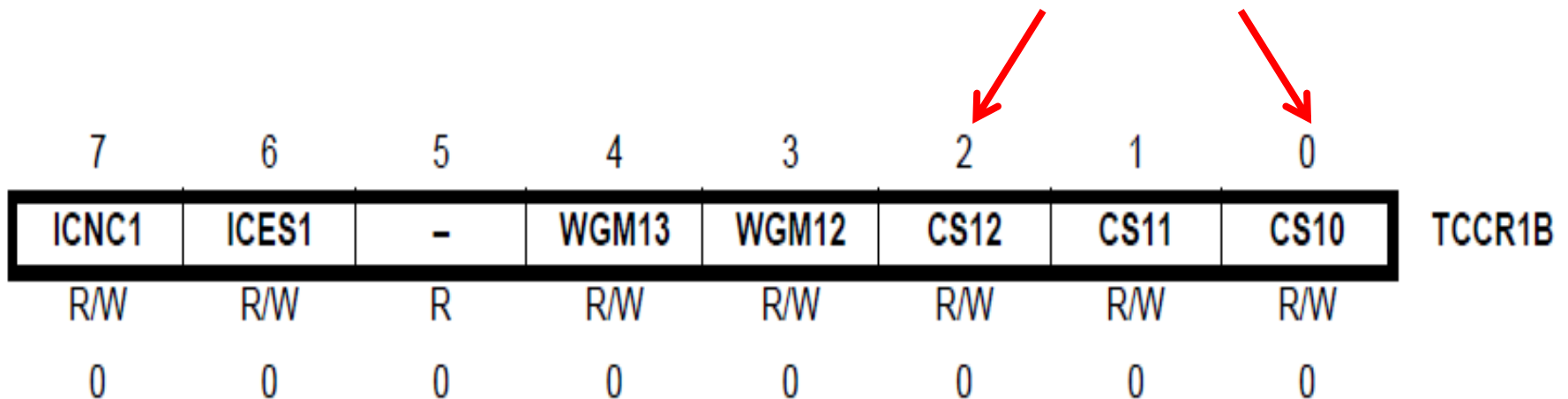
TCCR1B – Timer/Counter1 Control Register B

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Timer/Counter Control Register B

TCCR1B – Timer/Counter1 Control Register B

- Bit 2:0 – CS12:0: Clock Select



7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

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

Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{T2S}/(\text{No prescaling})$
0	1	0	$\text{clk}_{T2S}/8$ (From prescaler)
0	1	1	$\text{clk}_{T2S}/32$ (From prescaler)
1	0	0	$\text{clk}_{T2S}/64$ (From prescaler)
1	0	1	$\text{clk}_{T2S}/128$ (From prescaler)
1	1	0	$\text{clk}_{T2S}/256$ (From prescaler)
1	1	1	$\text{clk}_{T2S}/1024$ (From prescaler)

Prescaler Setting

- TCCR1B = 0b000000101;

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
0	0	0	0	0	1	0	1

- Thus, we are setting prescaler 1:1024

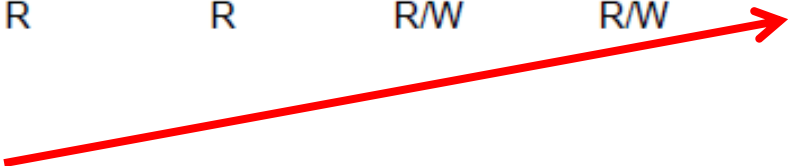
How to use this timer?

- Per 1024 clocks, (remember, we are using 1:1024 prescaler) Timer1 increments by 1
- Timer1 is 16 bit register; it overflows at FFFF (hex) or 65,535 (in decimal) and starts again from zero
- When Timer1 overflows, a flag is set
- This flag is read and further program is decided

TIFR1 – Timer/Counter1 Flag Register

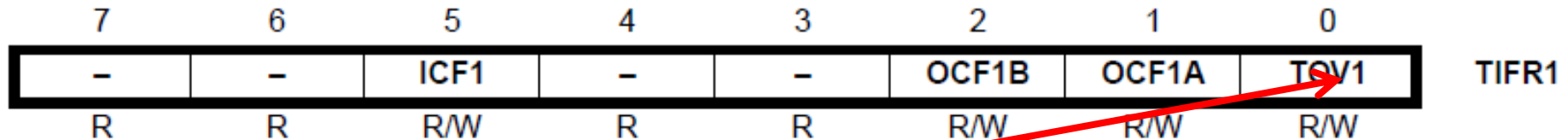
TIFR1 – Timer/Counter1 Interrupt Flag Register

7	6	5	4	3	2	1	0	
–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
R	R	R/W	R	R	R/W	R/W	R/W	



- Bit 0 – TOV1: Timer/Counter1, Overflow Flag
- TOV1 is set automatically on overflow, but is not cleared automatically
- We need to clear it every time

TIFR1 – Timer/Counter1 Flag Register



- Bit 0 – TOV1: Timer/Counter1, Overflow Flag

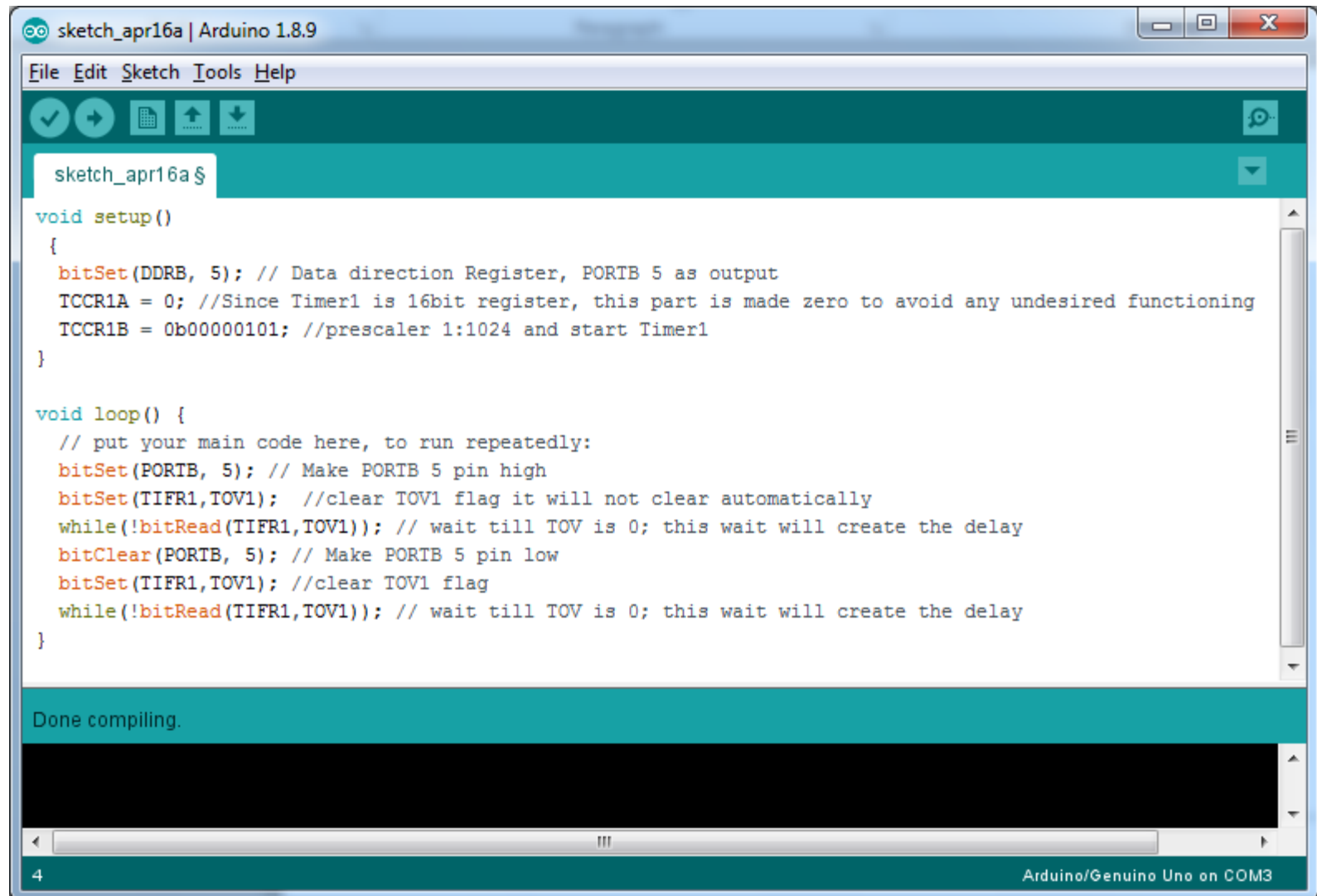
- Example:
- `bitSet(TIFR1,TOV1);` //set this bit to clear flag
- `while(!bitRead(TIFR1,TOV1));` // wait till TOV is 0
- Let us put this technique in our blinking LED program

Program configuring Timer as Delay

```
void setup()
{
    bitSet(DDRB, 5); // Data direction Register, PORTB 5 as output
    TCCR1A = 0; //Since Timer1 is 16bit register, this part is made zero to avoid any
                                                         undesired functioning
    TCCR1B = 0b00000101; //prescaler 1:1024 and start Timer1
}

void loop() {
    // put your main code here, to run repeatedly:
    bitSet(PORTB, 5); // Make PORTB 5 pin high
    bitSet(TIFR1,TOV1); //clear TOV1 flag it will not clear automatically
    while(!bitRead(TIFR1,TOV1)); // wait till TOV is 0; this wait will create the delay
    bitClear(PORTB, 5); // Make PORTB 5 pin low
    bitSet(TIFR1,TOV1); //clear TOV1 flag
    while(!bitRead(TIFR1,TOV1)); // wait till TOV is 0; this wait will create the delay
}
```

Program configuring Timer as Delay



The screenshot shows the Arduino IDE interface with a sketch named 'sketch_apr16a'. The code is written in C++ and configures Timer1 to create a delay by toggling a pin. The IDE status bar at the bottom indicates 'Done compiling.' and 'Arduino/Genuino Uno on COM3'.

```
sketch_apr16a | Arduino 1.8.9
File Edit Sketch Tools Help

sketch_apr16a $

void setup()
{
  bitSet(DDRB, 5); // Data direction Register, PORTB 5 as output
  TCCR1A = 0; //Since Timer1 is 16bit register, this part is made zero to avoid any undesired functioning
  TCCR1B = 0b000000101; //prescaler 1:1024 and start Timer1
}

void loop() {
  // put your main code here, to run repeatedly:
  bitSet(PORTB, 5); // Make PORTB 5 pin high
  bitSet(TIFR1,TOV1); //clear TOV1 flag it will not clear automatically
  while(!bitRead(TIFR1,TOV1)); // wait till TOV is 0; this wait will create the delay
  bitClear(PORTB, 5); // Make PORTB 5 pin low
  bitSet(TIFR1,TOV1); //clear TOV1 flag
  while(!bitRead(TIFR1,TOV1)); // wait till TOV is 0; this wait will create the delay
}

Done compiling.

4 Arduino/Genuino Uno on COM3
```

Delay Calculation from Timer Setting

- What will be the delay?
- Clock runs at 16MHz
- Due to prescaler, per 1024 clock, Timer1 increments by 1
- Thus, each increment of Timer1 takes

$$\frac{1 \times 1024}{16 \times 10^6} = 64 \mu s$$

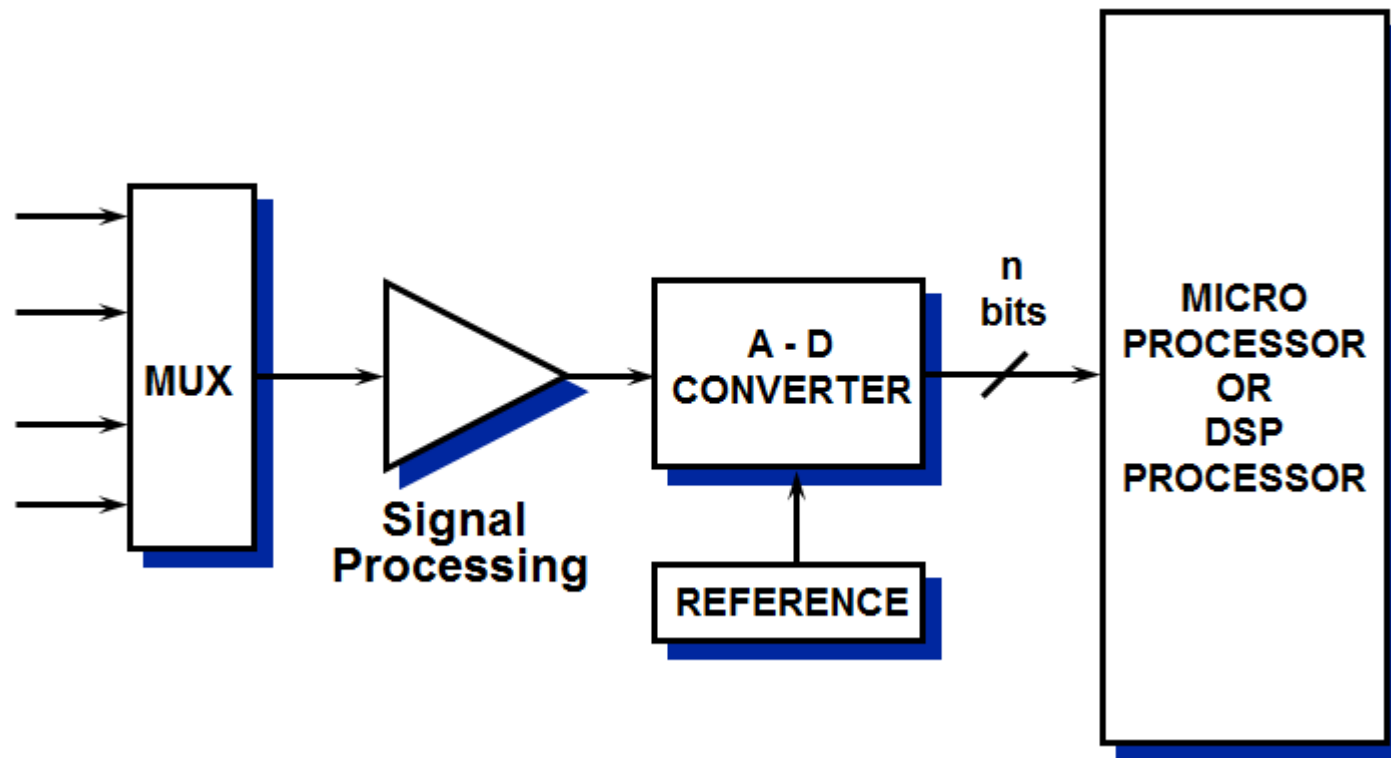
Delay Calculation from Timer Setting

- For Timer1 overflow, it will increment upto 65,535
- Then delay will be

$$65535 \times 64 \mu s = 4.19s$$

ADC

ADC (analog-to-digital converter) - Converts an analog signal to a digital signal



To operate ADC several register are to be configured

ADC Voltage Reference Selection

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 21-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

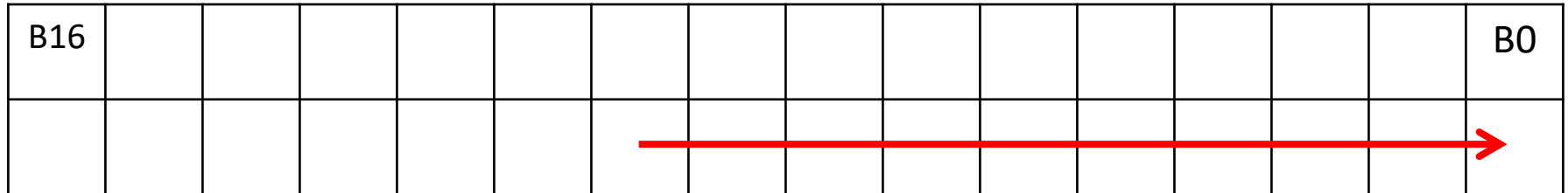
ADC Channel Selection

Input Channel Selections

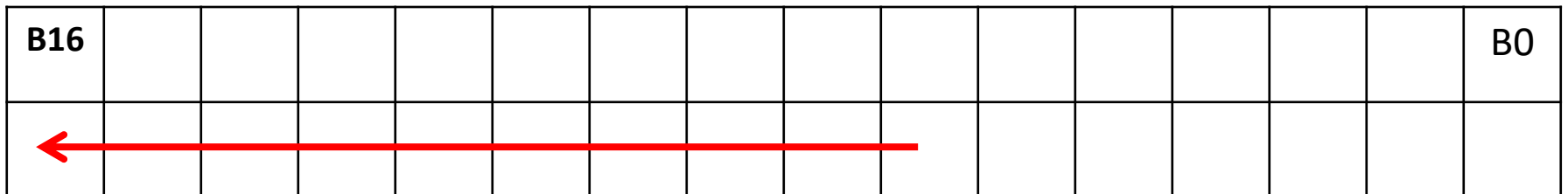
MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8 ⁽¹⁾
1001	(reserved)

ADC bit Alignment

The result of ADC is stored in a register named ADC. This register is 16 bit while the conversion result is 10 bit. So the result either need to be left justified or right justified.



10 bit ADC Right Justified



10 bit ADC Left Justified

ADC bit Alignment

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted.

ADC Control

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

ADC Conversion Rate

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

Table 21-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC Mode

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

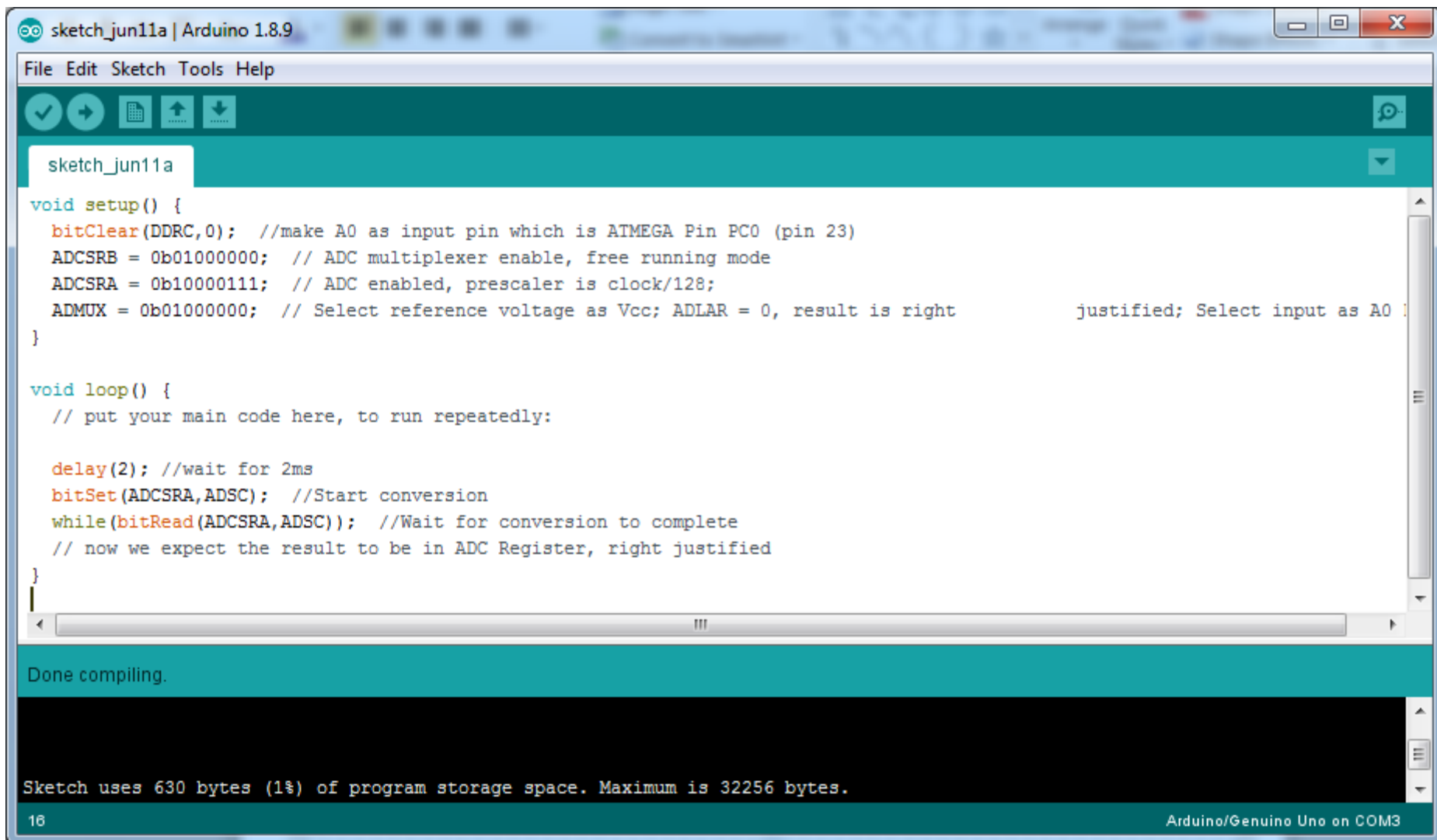
Table 21-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADC Programming

```
void setup() {  
    bitClear(DDRC,0); //make A0 as input pin which is ATMEGA Pin PC0 (pin 23)  
    ADCSRB = 0b01000000; // ADC multiplexer enable, free running mode  
    ADCSRA = 0b10000111; // ADC enabled, prescaler is clock/128;  
    ADMUX = 0b01000000; // Select reference voltage as Vcc; ADLAR = 0, result is right  
                           justified; Select input as A0 PIN  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
    delay(2); //wait for 2ms  
    bitSet(ADCSRA,ADSC); //Start conversion  
    while(bitRead(ADCSRA,ADSC)); //Wait for conversion to complete  
    // now we expect the result to be in ADC Register, right justified  
}
```

ADC Programming



The screenshot shows the Arduino IDE interface with a sketch named 'sketch_jun11a'. The code is for an ADC conversion on an ATMEGA pin. The setup function initializes the pin and registers. The loop function starts a conversion and waits for it to complete. The status bar at the bottom indicates the sketch is compiled and uses 630 bytes of program storage space.

```
sketch_jun11a | Arduino 1.8.9
File Edit Sketch Tools Help

sketch_jun11a

void setup() {
  bitClear(DDRC,0); //make A0 as input pin which is ATMEGA Pin PC0 (pin 23)
  ADCSRB = 0b01000000; // ADC multiplexer enable, free running mode
  ADCSRA = 0b10000111; // ADC enabled, prescaler is clock/128;
  ADMUX = 0b01000000; // Select reference voltage as Vcc; ADLAR = 0, result is right justified; Select input as A0
}

void loop() {
  // put your main code here, to run repeatedly:

  delay(2); //wait for 2ms
  bitSet(ADCSRA,ADSC); //Start conversion
  while(bitRead(ADCSRA,ADSC)); //Wait for conversion to complete
  // now we expect the result to be in ADC Register, right justified
}

Done compiling.

Sketch uses 630 bytes (1%) of program storage space. Maximum is 32256 bytes.

16 Arduino/Genuino Uno on COM3
```

How to know whether ADC is working?

- We have to put a serial monitor to view the result
- Two lines to be added to the above program

Serial.begin(9600);

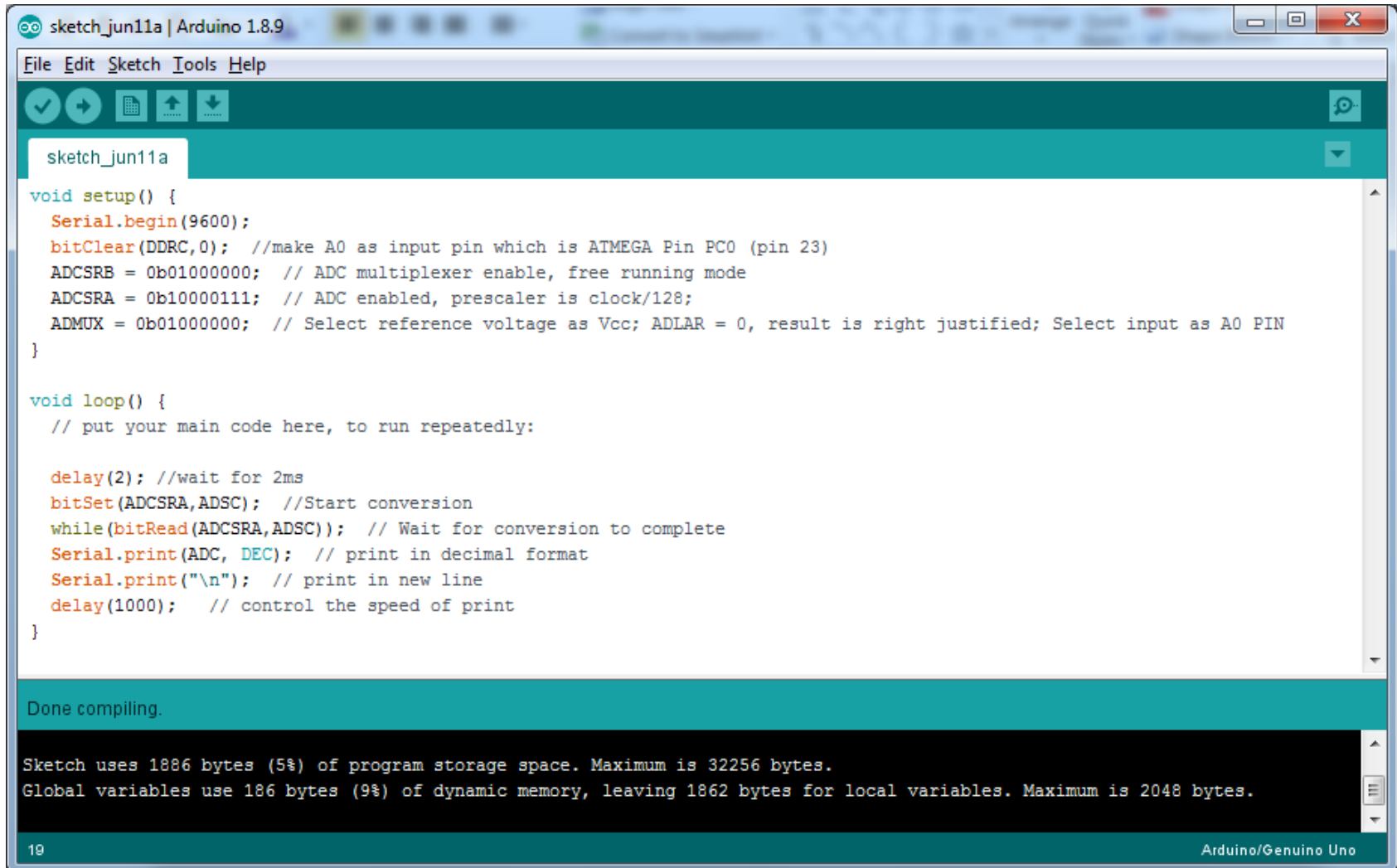
Serial.print(ADC, DEC);

These are inbuilt program, we are using only to verify whether ADC is working or not.

ADC Programming with Monitor

```
void setup() {  
  Serial.begin(9600);  
  bitClear(DDRC,0); //make A0 as input pin which is ATMEGA Pin PC0 (pin 23)  
  ADCSRB = 0b01000000; // ADC multiplexer enable, free running mode  
  ADCSRA = 0b10000111; // ADC enabled, prescaler is clock/128;  
  ADMUX = 0b01000000; // Select reference voltage as Vcc; ADLAR = 0, result is right  
                        justified; Select input as A0 PIN  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  
  delay(2); //wait for 2ms  
  bitSet(ADCSRA,ADSC); //Start conversion  
  while(bitRead(ADCSRA,ADSC)); // Wait for conversion to complete  
  Serial.print(ADC, DEC); // print in decimal format  
  Serial.print("\n"); // print in new line  
  delay(1000); // control the speed of print  
}
```

ADC Programming with Monitor

A screenshot of the Arduino IDE interface. The title bar shows 'sketch_jun11a | Arduino 1.8.9'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for checking, running, uploading, and downloading. The main text area shows a C++ sketch for an ADC program. The sketch includes a setup function that initializes the serial port and configures the ADC, and a loop function that performs a conversion and prints the result. The status bar at the bottom indicates 'Done compiling.' and provides memory usage statistics: 'Sketch uses 1886 bytes (5%) of program storage space. Maximum is 32256 bytes.' and 'Global variables use 186 bytes (9%) of dynamic memory, leaving 1862 bytes for local variables. Maximum is 2048 bytes.' The page number '19' is in the bottom left, and 'Arduino/Genuino Uno' is in the bottom right.

```
sketch_jun11a | Arduino 1.8.9
File Edit Sketch Tools Help

sketch_jun11a

void setup() {
  Serial.begin(9600);
  bitClear(DDRC,0); //make A0 as input pin which is ATMEGA Pin PC0 (pin 23)
  ADCSRB = 0b01000000; // ADC multiplexer enable, free running mode
  ADCSRA = 0b10000111; // ADC enabled, prescaler is clock/128;
  ADMUX = 0b01000000; // Select reference voltage as Vcc; ADLAR = 0, result is right justified; Select input as A0 PIN
}

void loop() {
  // put your main code here, to run repeatedly:

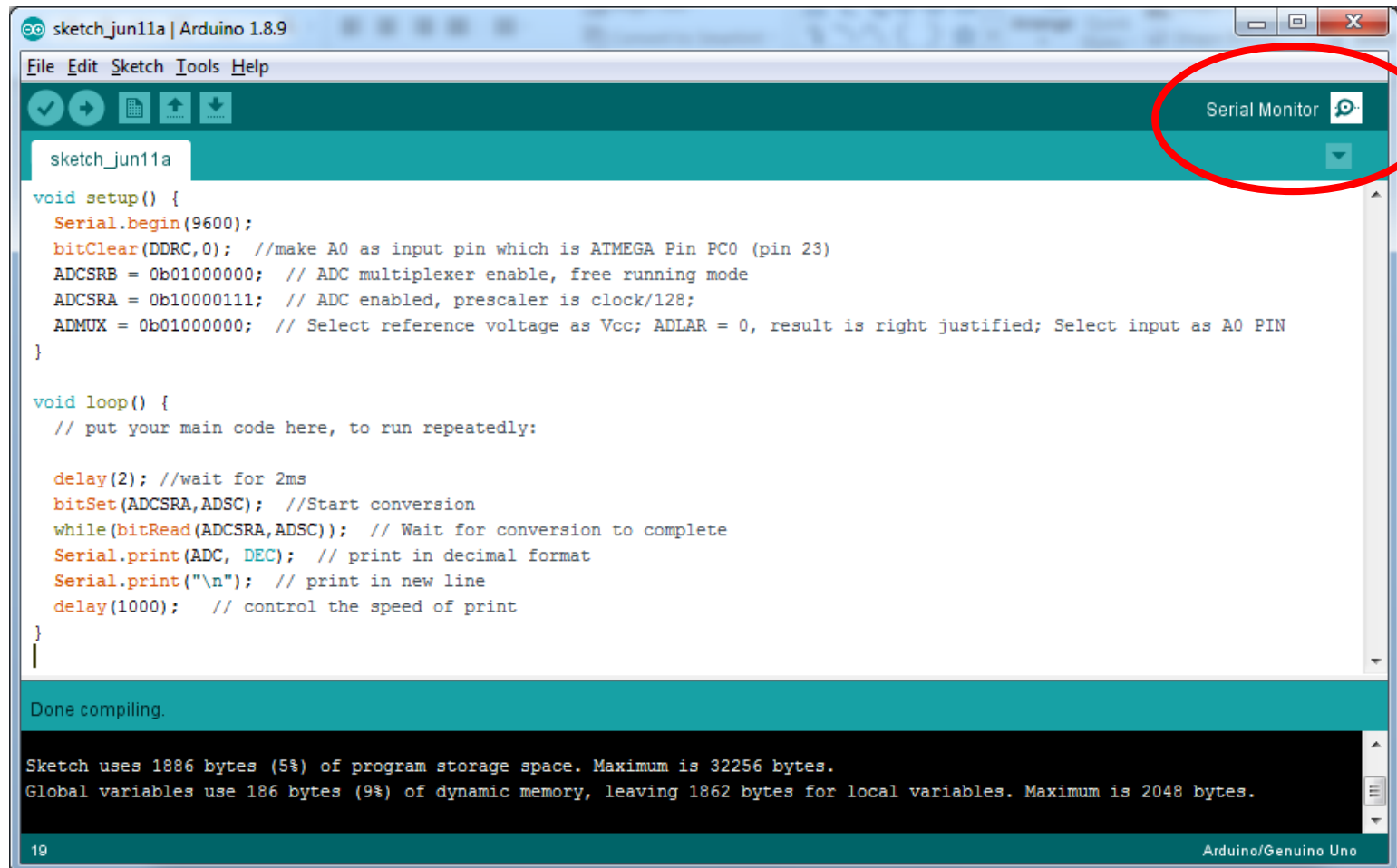
  delay(2); //wait for 2ms
  bitSet(ADCSRA,ADSC); //Start conversion
  while(bitRead(ADCSRA,ADSC)); // Wait for conversion to complete
  Serial.print(ADC, DEC); // print in decimal format
  Serial.print("\n"); // print in new line
  delay(1000); // control the speed of print
}
```

Done compiling.

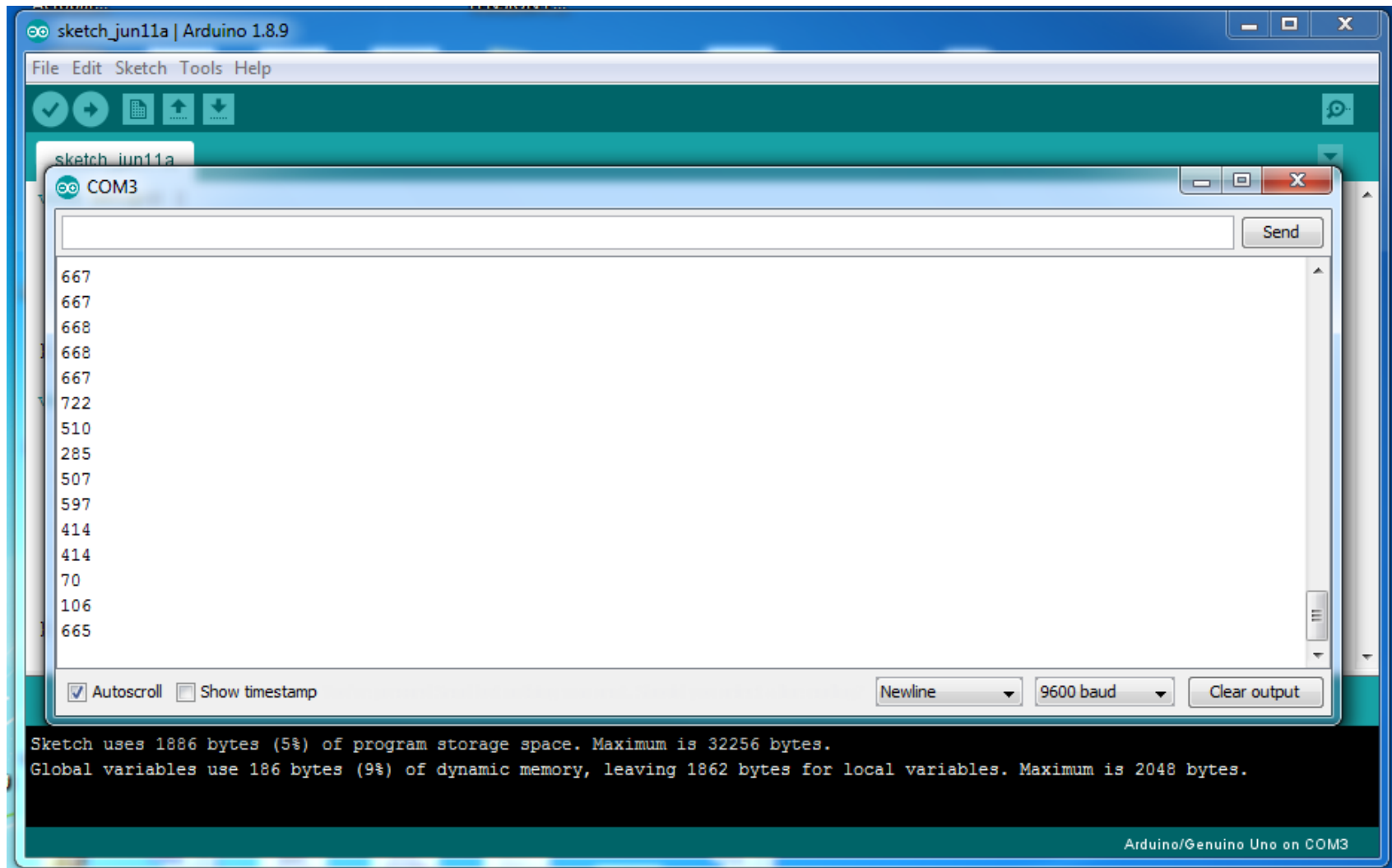
Sketch uses 1886 bytes (5%) of program storage space. Maximum is 32256 bytes.
Global variables use 186 bytes (9%) of dynamic memory, leaving 1862 bytes for local variables. Maximum is 2048 bytes.

19 Arduino/Genuino Uno

ADC Programming with Monitor



ADC Programming with Monitor



ADC Testing

- At first Connect A0 to GND
- Check Serial Monitor for 0
- Then Connect A0 to 5V
- Check Serial Monitor for 1023
- Then Connect A0 to 3.3V
- Check Serial Monitor for 667

ADC: Multiple I/P Pin

- What if we want to utilize two Pins?
- Example:
 - Select one Pin with the help of ADMUX Register
 - Wait
 - Convert and get result
 - Select another Pin with the help of ADMUX Register
 - Wait
 - Convert and get result **and so on . . .**

ADC: Programming Technique with Multiple Input Pin

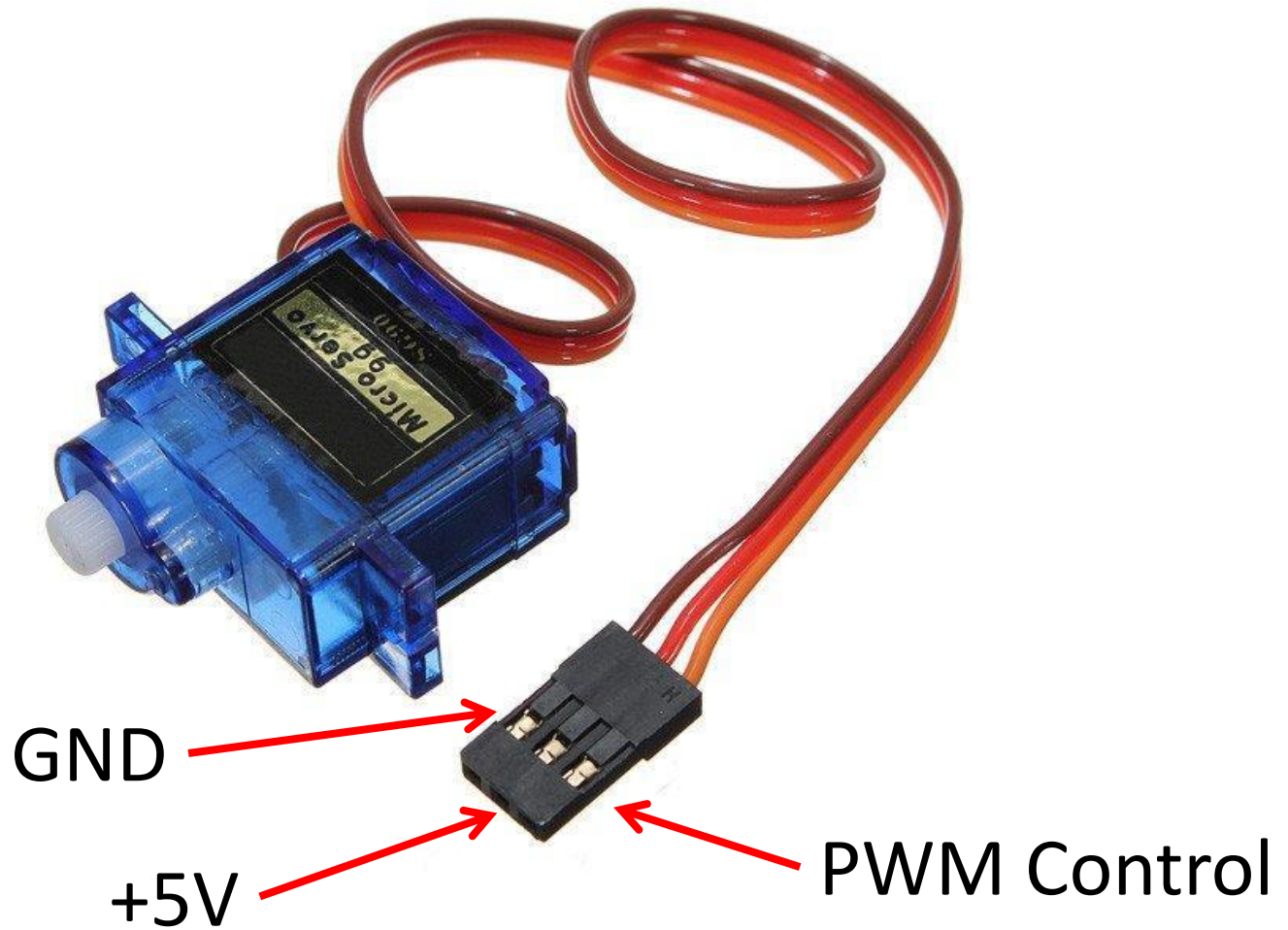
```
void setup() {  
    bitClear(DDRC,0); //make A0 as input pin which is ATMEGA Pin PC0 (pin 23)  
    ADCSRB = 0b01000000; // ADC multiplexer enable, free running mode  
    ADCSRA = 0b10000111; // ADC enabled, prescaler is clock/128;  
    ADMUX = 0b01000000; // Select reference voltage as Vcc; ADLAR = 0, result is rt justified  
}
```

```
void loop() {  
    // put your main code here, to run repeatedly:  
    ADMUX = 0b01000000; // Select input as A0  
    delay(2); //wait for 2ms  
    bitSet(ADCSRA,ADSC); //Start conversion  
    while(bitRead(ADCSRA,ADSC)); //Wait for conversion to complete  
    // do something with the result in ADC register for pin A0  
    ADMUX = 0b01000001; // Select input as A1  
    delay(2); //wait for 2ms  
    bitSet(ADCSRA,ADSC); //Start conversion  
    while(bitRead(ADCSRA,ADSC)); //Wait for conversion to complete  
    // do something with the result in ADC register for pin A1  
}
```

APPLICATION: SERVO CONTROL

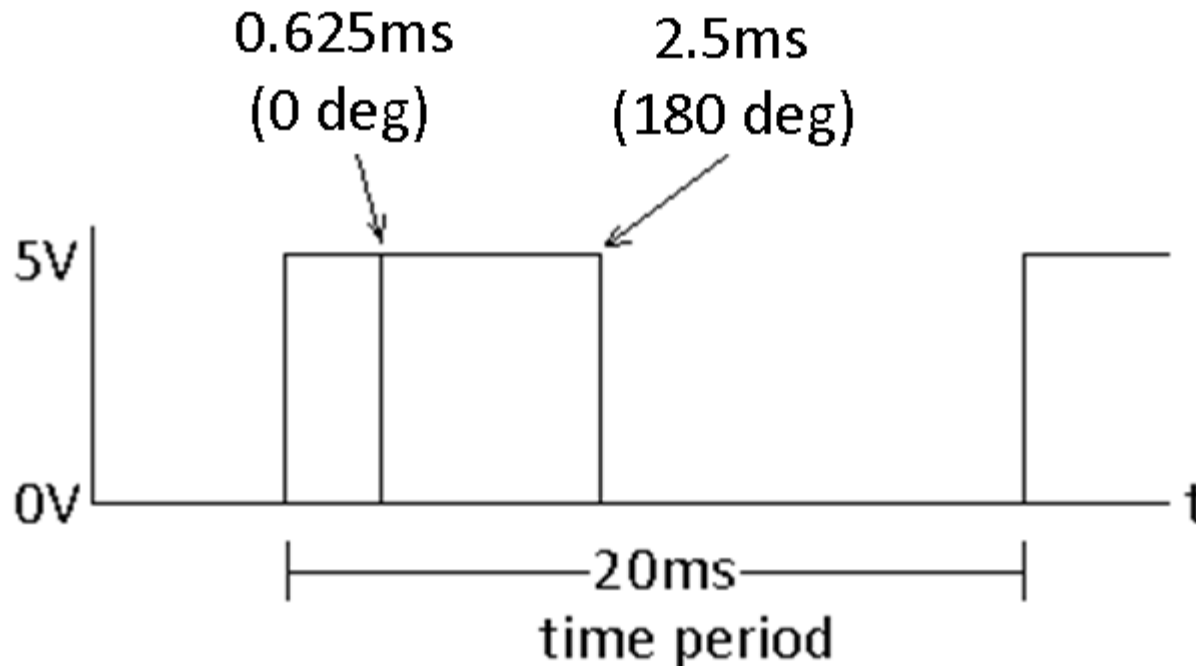
What is Servo?

SERVO 9g



Servo PWM Control Method

- Servo moves 0-180 degrees
- Uses pulse width modulation (PWM)



How to enable PWM in ATMEGA?

- Using Timer1 (we already know) to set the time period
- And some other registers to set pulse width

How to enable PWM in ATMEGA?

- Previously Timer1 allowed to overflow at FFFF(65535) and again start from zero.
- Now by controlling TCCR1 A and B , WGM 10 - 13 bits, Timer1 will overflow at a value given to another register called ICR1 and again starts at zero.
- Example: If ICR1 = 100, then Timer1 will overflow at 100 and start at zero again.

Recap of Timer1 Control Registers

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TCCR1B – Timer/Counter1 Control Register B

[illegible]

How to enable PWM in ATMEGA?

WGM13	WGM12	WGM11	WGM10
1	1	1	0

How to enable PWM in ATMEGA?

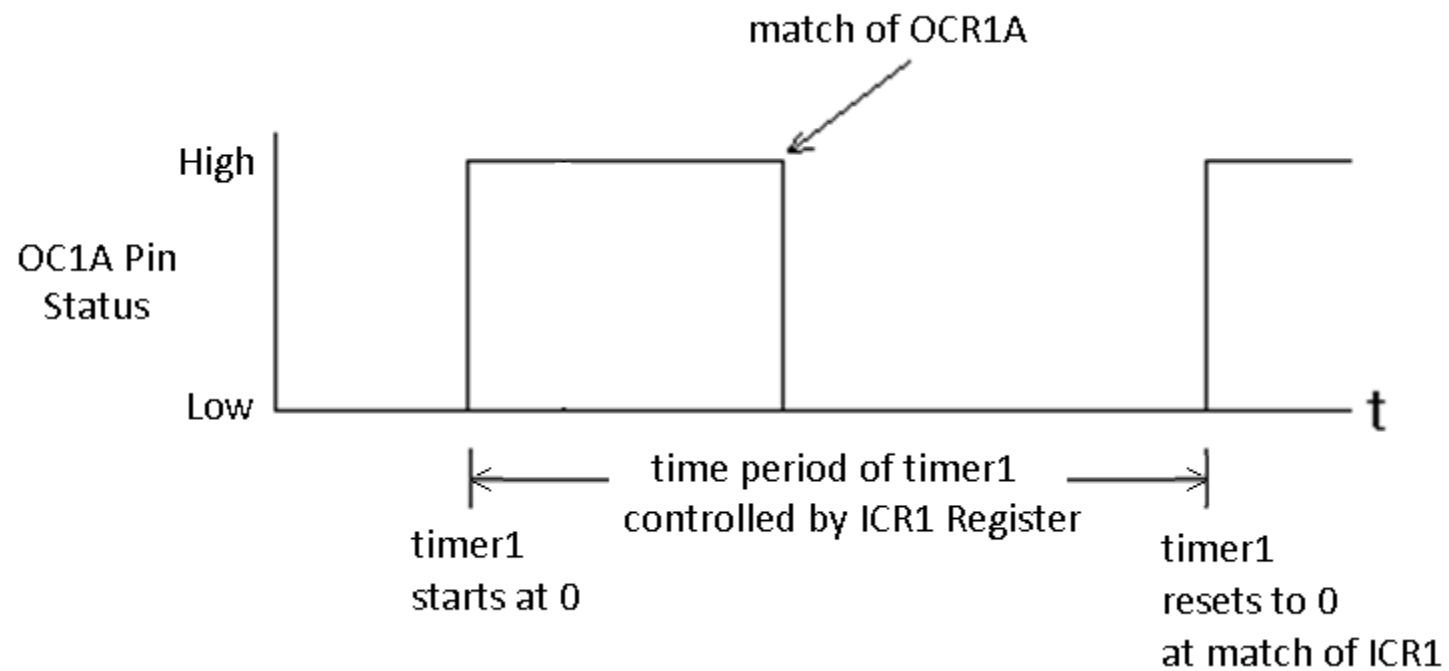
- Another register OCR1A controls the duration of pulse width of PIN OC1A of ATMEGA(PIN 15) or PIN9 of Arduino board
- When Timer1 is zero PIN OC1A gets high
- When Timer1 = value given in OCR1A, PIN OC1A goes low.
- Example: If OCR1A = 60, then PIN OC1A will get high at Timer1=0 and OC1A will get low at Timer1=60.

PWM Control

COM1A1	COM1A0	COM1B1	COM1B0
1	0	1	0

There are several other modes for configuration of PWM, we consider one of the mode

Timer1, ICR1, OC1A



How to adjust PWM time period?

- Timer1 is 16 bit register.
- With 16MHz clock and prescalar 1/8, to count from 0 to 65,535, the time period will be

$$\frac{65535 \times 8}{16 \times 1000000} = 32ms$$

- Since this is more than 20ms, ICR1 will have to be adjusted

Adjustment of ICR1

- Let ICR1 = x

$$\frac{x \times 8}{16 \times 1000000} = 20ms$$

- Which gives ICR1 = 40,000
- Code: ICR1 = 40000;

How to control the pulse width?

- Remember OCR1A register?
- We need a pulse width of 0.625ms for 0 degree
- We know, 40,000 is equivalent to 20ms
- So, 0.625ms is 1250
- Code: $\text{OCR1A}_{(\text{min})} = 1250;$

How to control the pulse width?

- We need a pulse width of 2.5 ms for 180 degree
- We know, 40,000 is equivalent to 20ms
- So, 2.5ms is 5000
- Code: $\text{OCR1A}_{(\text{max})} = 5000;$
- So, we will control the value of OCR1A register between 1250 and 5000.
- We can start with an intermediate value, say 2500

Main Code

- Input to two ADC PINS
- Start OCR1A with 2500 (servo initial position)
- If $ADC1 > ADC2$ increment OCR1A by 50
- OCR1A must not go higher than 5000
- If $ADC2 > ADC1$ decrement OCR1A by 50
- OCR1A must not go lower than 1250

Main Code

```
int i=0, j=0, k=0; // set some variables
```

```
void setup()
```

```
{
```

```
    // put your setup code here, to run once:
```

```
    bitSet(DDRB,PB1); //Set PB1 or OC1A as output
```

```
    bitClear(DDRC,PC0); //Set A0 as one analog input
```

```
    bitClear(DDRC,PC1); //Set A1 as one analog input
```

```
    TCCR1A = 0b10100010; // enable PWM mode as explained
```

```
    TCCR1B = 0b00011010; // enable PWM mode as explained
```

```
    ICR1=40000; // Time period set by this register
```

```
    ADMUX = 0b01000000; //Enable ADC as explained earlier
```

```
    ADCSRA = 0b10000111; //Enable ADC as explained earlier
```

```
    ADCSRB = 0b01000000; //Enable ADC as explained earlier
```

```
    OCR1A=2500; // initial pulse width set by this register
```

```
}
```

Main Code

```
void loop()
{
    // put your main code here, to run repeatedly:

    ADMUX = 0b01000000; // get input from A0 pin
    delay(2);
    bitSet(ADCSRA,ADSC);
    while(bitRead(ADCSRA,ADSC));
    j=ADC; // store result of A0

    ADMUX = 0b01000001; // get input from A1 pin
    delay(2);
    bitSet(ADCSRA,ADSC);
    while(bitRead(ADCSRA,ADSC));
    k=ADC; // store result of A1
```

Main Code

```
if ((j>k)&&(OCR1A<=4950)) // if j greater than k and if OCR1A less than
4950, so that it never exceeds 5000
OCR1A=OCR1A+50; // increment OCR1A by 50

if ((j<k)&&(OCR1A>=1300)) // if j less than k and if OCR1A greater than
1300, so that it never goes below 1250
OCR1A=OCR1A-50; // decrement OCR1A by 50

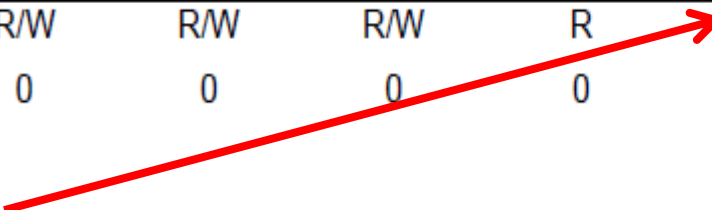
delay(50); // wait so that servo does not move rapidly
}
```

Thank You

How to enable Timer1?

Power Reduction Register

7	6	5	4	3	2	1	0	
PRTWI	PRTIM2	PRTIM0	-	PRTIM1	PRSPI	PRUSART0	PRADC	PRR
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	



- **Bit 3 - PRTIM1: Power Reduction Timer/Counter1**

Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

- `bitClear(PRR,PRTIM1);`

DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5:0 – ADC5D..ADC0D: ADC5..0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC5..0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.