ATmega32 Microcontroller Overview

- **Architecture: AVR, based on Advanced RISC**

- **Registers: 32 × 8-bit general-purpose registers**

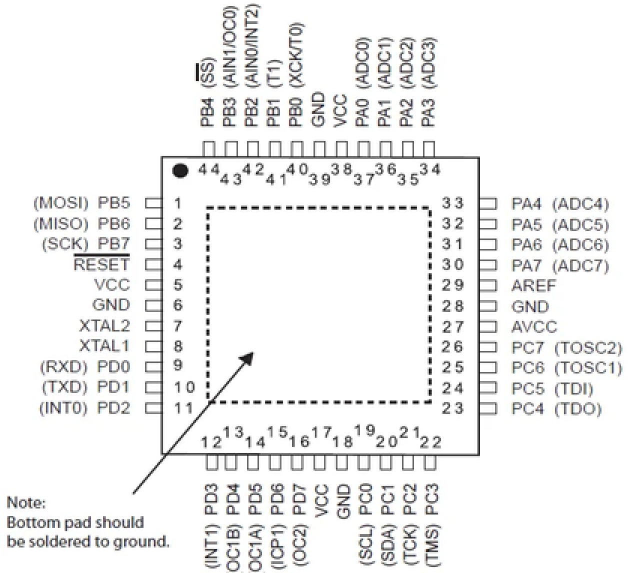
- **Peripheral Features:**

1. Two 8-bit timers
2. One 16-bit timer
3. Four PWM channels
4. 8-channel, 10-bit ADC
5. Interfaces: USART, SPI, TWI (I²C)
6. Watchdog timer for stuck loops
7. Analog comparator

- **Pin Configurations**

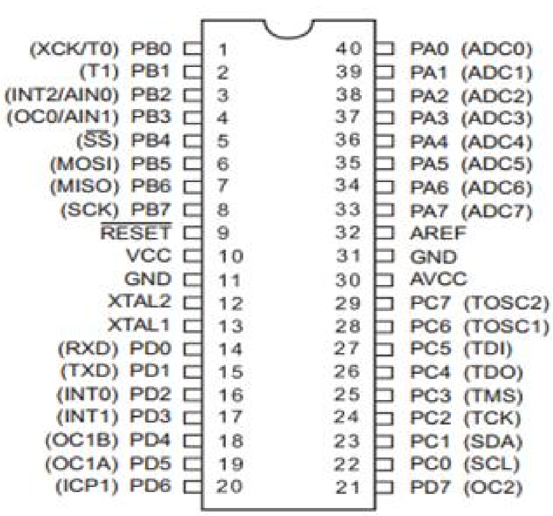
32 programmable I/O line

TQFP/MLF

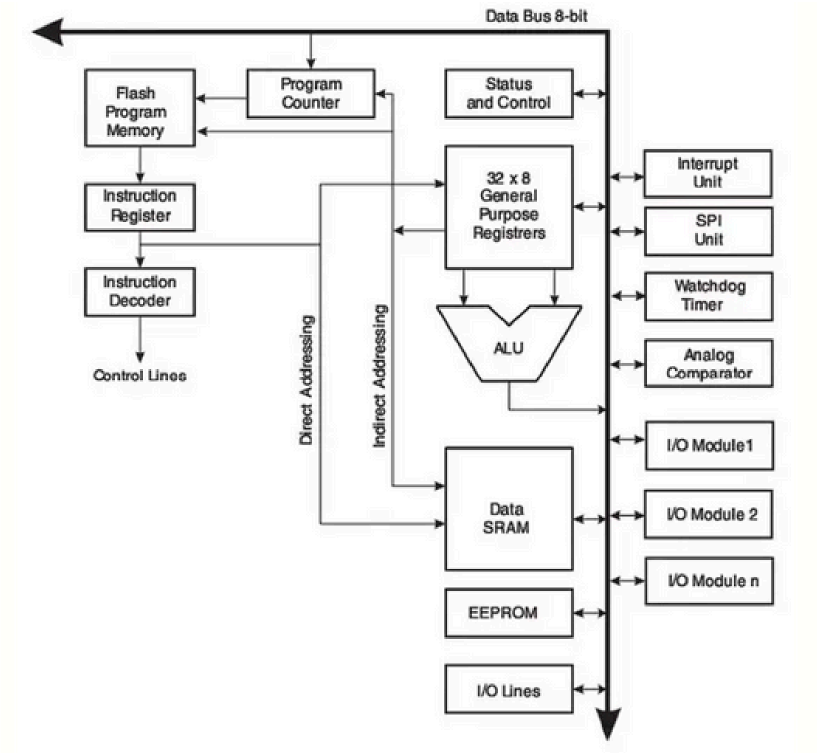


Note:
Bottom pad should
be soldered to ground.

PDIP



- **Block Diagram**



- **AVR CPU Core**

Harvard Architecture: separate program and data memory
Single-level pipeline: fetch & decode in parallel → faster execution
Fast ALU: supports efficient arithmetic and logic operations
Status Register (SREG): 8-bit, each bit is a flag (e.g., carry, zero, overflow, etc.)

- **System Clock**

Internal Clock: 1 MHz
External Clock: Up to 16 MHz
Clock type depends on the application (accuracy, speed, power).

- **Accessing I/O Registers**

Registers are used to control, read, and configure peripherals.

Steps:

1. Get the address

Example: 0x3B

2. Point to the address

volatile uint8_t *PORTA = (volatile uint8_t *)(0x3B);

   - volatile: prevents compiler optimizations (important for hardware registers).
   - uint8_t: 8-bit data type.
   - (0x3B): converts the hex address into a usable memory location.

3. Access directly

****((volatile uint8_t *)(0x3B)) = 0x10;***

DIO Programming (ATmega32)

1. General Information

- The ATmega32 microcontroller has 32 programmable I/O pins.
- These pins are divided into 4 ports (A, B, C, D), with 8 pins per port.
- Each port can be configured as general-purpose I/O.
- Each pin can serve a specific function depending on configuration.
- Pin behavior is controlled using I/O registers:
  - DDRx → Data Direction Register
  - PORTx → Port Register (Output Register)
  - PINx → Port Input Register

2. Data Direction Register (DDRx)

- Defines whether each pin is input or output.
- DDRx is an 8-bit register, each bit corresponds to one pin.

Examples:

- Configure pin 2 in Port A as input:

```
DDRA &= ~(1 << 2);   // clear bit → input
```

- Configure pin 6 in Port A as output:

```
DDRA |= (1 << 6);    // set bit → output
```

3. Port Register (PORTx)

- Used to set pin output values:
  - High (1) → logic HIGH
  - Low (0) → logic LOW

Examples:

- Set pin 3 in Port A to LOW:
  PORTA &= ~(1 << 3);
- Set pin 5 in Port A to HIGH:
  PORTA |= (1 << 5);

4. Port Input Register (PINx)

- Stores the current state of each pin (HIGH or LOW).
- Useful for reading inputs.

Example:

- Read the state of pin 7 in Port A:
  (PINA & (1 << 7))<<7   // Pin 7 is HIGH
- Read the state of pin 2 in Port A:
  (PINA & (1 << 2))<<2   // Pin 2 is HIGH