UNIT I Advance Processors

Teaching Hrs:10 **Marks 16**

Course Outcome-Suggest the relevant computing systems/processor for specific type of application

To attain above course outcome candidate must able to:

1a. Describe the given advancement in the processor architecture.
1b. Describe the given features of Arduino board.
1c. Describe the given function in Arduino IDE.
1d. Describe the given feature of the ARM7 processors.
1e. Compare the given salient features of ARM 7 and ARM 7TDMI processors.

Unit focus on following major points:

1.1 Advances in processor architecture: RISC, Pipelining and Superscalar concepts, advantages and Applications.
1.2 Arduino: Introduction, Compatible R2/R3 Uno board Features. Atmega 328: Introduction, pin description.
1.3 Arduino IDE: Features, Sketch: C,C++ functions setup(), loop(), pinMode(), digitalWrite(), digitalRead() and delay()
1.4 Arduino Interfacing: LED, Relay, DC motor.
1.5 ARM: Introduction, Features of ARM7 and ARM7TDMI, advantages, applications.
Versions of ARM processor only features

1.1 Introduction

Processors have undergone a tremendous evolution throughout their history. A key milestone in this evolution was the introduction of the microprocessor, term that refers to a processor that is implemented in a single chip. The first microprocessor was introduced by Intel under the name of Intel 4004 in 1971. It contained about 2,300 transistors, was clocked at 740 KHz and delivered 92,000 instructions per second while dissipating around 0.5 watts. Since then, practically every year we have witnessed the launch of a new microprocessor, delivering significant performance improvements over previous ones. Some studies have estimated this growth to be exponential, in the order of about 50% per year, which results in a cumulative growth of over three orders of magnitude in a time span of two decades. These improvements have been fueled by advances in the manufacturing process and innovations in processor architecture.

The complexity of an integrated circuit is bounded by physical limitations on the number of transistors that can be put onto one chip, the number of package terminations that can connect the processor to other parts of the system, the number of interconnections it is possible to make on the chip, and the heat that the chip can dissipate. Advancing technology makes more complex and powerful chips feasible to manufacture.

1.2Processor Selection Criteria

With numerous kinds of processors, various design philosophies are available for digital systems. Following considerations need to be factored during processor selection for a Digital Systems.

1. Performance Considerations
2. Power considerations
3. Peripheral Set
4. Operating Voltage
5. Specialized Processing Units

Performance: The first and foremost consideration in selecting the processor is its performance. The performance speed of a processor is dependent primarily on its architecture and its silicon design. Evolution of fabrication techniques helped packing more transistors in same area there by reducing the propagation delay. Also presence of cache reduces instruction/data fetch timing. Pipelining and super-scalar architectures further improves the performance of the processor. Branch prediction, speculative execution etc are some other techniques used for improving the execution rate. Multi-cores are the new direction in improving the performance.
Rather than simply stating the clock frequency of the processor which has limited significance to its processing power, it makes more sense to describe the capability in a standard notation. MIPS (Million Instructions Per Second) or MIPS/MHz was an earlier notation followed by Dhrystones and latest EEMBC's **CoreMark**. CoreMark is one of the best ways to compare the performance of various processors.
Processor architectures with support for extra instruction can help improving performance for specific applications. For example, SIMD (Single Instruction/Multiple Data) set and Jazelle – Java acceleration can help in improving multimedia and JVM execution speeds.
So size of cache, processor architecture, instruction set etc has to be taken in to account when comparing the performance.

Power: Increasing the logic density and clock speed has adverse impact on power requirement of the processor. A higher clock implies faster charge and discharge cycles leading to more power consumption. More logic leads to higher power density there by making the heat dissipation difficult. Further with more emphasis on greener technologies and many systems becoming battery operated, it is important the design is for optimal power usage.
Techniques like **frequency scaling** – reducing the clock frequency of the processor depending on the load, **voltage scaling** – varying the voltage based on load can help in achieving lower power usage. Further asymmetric multiprocessors, under near idle conditions, can effectively power off the more powerful core and load the less powerful core for performing the tasks. SoC comes with advanced power gating techniques that can shut down clocks and power to unused modules.

Peripheral Set: Every system design needs, apart from the processor, many other peripherals for input and output operations. Since in an embedded system, almost all the processors used are SoCs, it is better if the necessary peripherals are available in the chip itself. This offers various benefits compared to peripherals in external IC's such as optimal power architecture, effective data communication using DMA etc. So it is important to have peripheral set in consideration when selecting the processor.

Operating Voltages: Each and every processor will have its own operating voltage condition. The operating voltage maximum and minimum ratings will be provided in the respective data sheet or user manual.
While higher end processors typically operate with 2 to 5 voltages including 1.8V for Cores/Analogue domains, 3.3V for IO lines, needs specialized PMIC devices, it is a deciding factor in low end micro-controllers based on the input voltage. For example it is cheaper to work with a 5V micro-controller when the input supply is 5V and 3.3 micro-controllers when operated with Li-on batteries.

Specialized Processing Units: Apart from the core, presence of various co-processors and specialized processing units can help achieving necessary processing performance. Co-processors execute the instructions fetched by the primary processor thereby reducing the load on the primary. Some of the popular co-processors include

Floating Point Co-processor:
RISC cores supports primarily integer only instruction set. Hence presence of a FP co-processor can be very helpful in application involving complex mathematical operations including multimedia, imaging, codecs, signal processing etc.

Graphic Processing Unit:
GPU(Graphic Processing Unit) also called as Visual processing unit is responsible for drawing images on the frame buffer memory to be displayed. Since human visual perception needed at-least 16 Frames per second for a smooth viewing, drawing for HD displays involves a lot of data bandwidth. Also with increasing graphic requirements such as textures, lighting shadersetc, GPU's have become a mandatory requirements for mobile phones, gaming consoles etc.
Various GPU's like ARM's MALI, PowerVX, OpenGLetc are increasing available in higher end processors. Choosing the right co-processor can enable smooth design of the embedded application.

Digital Signal Processor:
DSP is a processor designed specifically for signal processing applications. Its architecture supports processing of multiple data in parallel. It can manipulate real time signal and convert to other domains for processing. DSP's are either available as the part of the SoC or separate in an external package. DSP's are very helpful in multimedia applications. It is possible to use a DSP along with a processor or use the DSP as the main processor itself.

Price:
Various considerations discussed above can be taken in to account when a processor is being selected for an embedded design. It is better to have some extra buffer in processing capacities to enable enhancements in functionality without going for a major change in the design. While engineers (especially software/firmware engineers) will want to have all the functionalities, price will be the determining factor when designing the system and choosing the right processor.
In the upcoming blog, we will discuss about various memory technologies and factors to be considered when selecting them.

1.3 Advances in Processor Architecture

RISC (Reduced Instruction Set Computer)**:**
A single-chip processor need not be the same as the optimal architecture for a multi-chip processor. Their argument was subsequently supported by the results of a processor design project undertaken by a postgraduate class at Berkeley which incorporated Reduced Instruction Set Computer (RISC) architecture. This design, the Berkeley RISC I, was much simpler than the commercial CISC processors. The RISC I instruction set differed from the minicomputer-like CISC instruction sets used on commercial microprocessors in a number of ways.

It had the following key features:

1. A fixed (32-bit) instruction size with few formats; CISC processors typically had variable length instruction sets with many formats.
2. A load-store architecture where instructions that process data operate only on registers and are separate from instructions that access memory; CISC processors typically allowed values in memory to be used as operands in data processing instructions.
3. A large register bank of thirty-two 32-bit registers, all of which could be used for any purpose, to allow the load-store architecture to operate efficiently; CISC register sets were getting larger, but none was this large and most had different registers for different purposes (for example, the data and address registers on the Motorola MC68000).
4. These differences greatly simplified the design of the processor and allowed the designers to implement the architecture using organizational features that contributed to the performance of the prototype devices:
5. RISC organization
6. Hard-wired instructions decode logic; CISC processors used large microcode ROMs to decode their instructions.
7. Pipelined execution; CISC processors allowed little, if any, overlap between consecutive instructions (though they do now).
8. Single-cycle execution; CISC processors typically took many clock cycles to complete a single instruction.

RISC advantages

1. A smaller die size. A simple processor should require fewer transistors and less silicon area.
2. A shorter development time. A simple processor should take less design effort and therefore have a lower design cost and be better matched to the process technology.
3. A higher performance.
2. Higher performance had been sought through ever-increasing complexity, this was a bit hard to swallow. The argument goes something like this: smaller things have higher natural frequencies (insects flap their wings faster than small birds, small birds faster than large birds, and so on)Clock rates So a simple processor ought to allow a high clock rate. So let's design our complex processor by starting with a simple one, then add complex instructions one at a time. When we add a complex instruction it will make some high-level function more efficient, but it will also slow the clock down a bit for all instructions. We can measure the overall benefit on typical programs, and when we do, all complex instructions make the program run slower. Hence we stick to the simple processor we started with.

RISC Pipelines

A RISC processor pipeline operates in much the same way, although the stages in the pipeline are different. While different processors have different numbers of steps, they are basically variations of these five, used in the MIPS R3000 processor:

1 fetch instructions from memory
2 read registers and decode the instruction
3 execute the instruction or calculate an address
4 access an operand in data memory
5 write the result into a register

RISC Disadvantages:

1. There is still considerable controversy among experts about the ultimate value of RISC architectures. Its proponents argue that RISC machines are both cheaper and faster, and are therefore the machines of the future.
2. However, by making the hardware simpler, RISC architectures put a greater burden on the software. Is this worth the trouble because conventional microprocessors are becoming increasingly fast and cheap anyway
3. As memory speed increased, and high-level languages displaced assembly language, the major reasons for CISC began to disappear, and computer designers began to look at ways computer performance could be optimized beyond just making faster hardware.
4. One of their key realizations was that a sequence of simple instructions produces the same results as a sequence of complex instructions, but can be implemented with a simpler (and faster) hardware design. (Assuming that memory can keep up.) RISC (Reduced Instruction Set Computers) processors were the result.
5. CISC and RISC implementations are becoming more and more alike. Many of today's RISC chips support as many instructions as yesterday's CISC chips. And today's CISC chips use many techniques formerly associated with RISC chips.
6. To some extent, the argument is becoming moot because CISC and RISC implementations are becoming more and more alike. Many of today's RISC chips support as many instructions as yesterday's CISC chips. And today's CISC chips use many techniques formerly associated with RISC chips.

RISC Applications:

1. Video processing.
2. Image processing.
3. Telecommunications.

CISC (Complex Instruction Set Computer**):**

CISC is an acronym for Complex Instruction Set Computer and are chips that are easy to program and which make efficient use of memory. Since the earliest machines were programmed in assembly language and memory was slow and expensive, the CISC philosophy made sense, and was commonly implemented in such large computers as the PDP-11 and the DECsystem 10 and 20 machines.

Most common microprocessor designs such as the Intel 80x86 and Motorola 68K series followed the CISC philosophy.

But recent changes in software and hardware technology have forced a re-examination of CISC and many modern CISC processors are hybrids, implementing many RISC principles.

CISC was developed to make compiler development simpler. It shifts most of the burden of generating machine instructions to the processor. For example, instead of having to make a compiler write long machine instructions to calculate a square-root, a CISC processor would have a built-in ability to do this.

CISC Attributes

The design constraints that led to the development of CISC (small amounts of slow memory and fact that most early machines were programmed in assembly language) give CISC instructions sets some common characteristics:

1. A 2-operand format, where instructions have a source and a destination. Register to register, register to memory, and memory to register commands. Multiple addressing modes for memory, including specialized modes for indexing through arrays
2. Variable length instructions where the length often varies according to the addressing mode
3. Instructions which require multiple clock cycles to execute.

E.g. Pentium is considered a modern CISC processor

Most CISC hardware architectures have several characteristics in common:

1. Complex instruction-decoding logic, driven by the need for a single instruction to support multiple addressing modes.
2. A small number of general purpose registers. This is the direct result of having instructions which can operate directly on memory and the limited amount of chip space not dedicated to instruction decoding, execution, and microcode storage.
3. Several special purposesregister. Many CTSC designs set aside special registers for the stack pointer, interrupt handling, and so on. This can simplify the hardware design somewhat, at the expense of making the instruction set more complex.
4. A 'Condition code" register which is set as a side-effect of most instructions. This register reflects whether the result of the last operation is less than, equal to, or greater than zero and records if certain error conditions occur.

Comparison RISC VS CISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

Instruction pipelining: instruction pipelining is a technique for implementing instruction-level parallelism within a single processor. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different processor units with different parts of instructions processed in parallel. It allows faster CPU throughput than would otherwise be possible at a given clock rate.Many designs include pipelines as long as 7, 10 and even 20 stages.

A processor executes an individual instruction in a sequence of steps. A typical sequence might be:
1. Fetch the instruction from memory (fetch).
2. Decode it to see what sort of instruction it is (dec).
3. Access any operands that may be required from the register bank (reg).
4. Combine the operands to form the result or a memory address (ALU).
5. Access memory for a data operand, if necessary (mem).
6. Write the result back to the register bank (res).

Not all instructions will require every step, but most instructions will require most of them. These steps tend to use different hardware functions, for instance the ALU is probably only used in step 4. Therefore, if an instruction does not start before its predecessor has finished, only a small proportion of the processor hardware will be in use in any step. An obvious way to improve the utilization of the hardware resources, and also the processor throughput, would be to start the next instruction before the current one has finished. This technique is called pipelining a very effective way of exploiting concurrency in a general-purpose processor. Taking the above sequence of operations, the processor is organized so that as soon as one instruction has completed step 1 and moved on to step 2, the next instruction begins step 1. This is illustrated in Figure 1.1 .In principle such a pipeline should deliver a six times speed-up compared with non-overlapped instruction execution; in practice things do not work out quite so well for reasons we will see below.
It is relatively frequent in typical computer programs that the result from one instruction is used as an operand by the next instruction. When this occurs the pipeline operation shown in Figure 1.1 breaks down, since the result of instruction 1 is not available at the time that instruction 2 collects its operands. Instruction 2 must therefore stall until the result is available

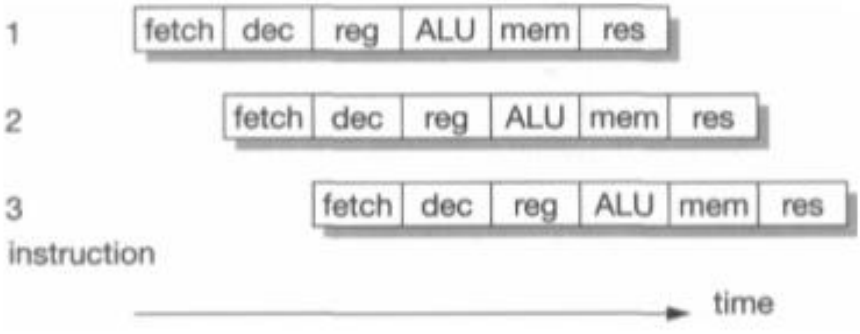

Fig 1.1 Pipelined instruction execution

Pipe line efficiency:
Though there are techniques which reduce the impact of these pipeline problems, they cannot remove the difficulties altogether. The deeper the pipeline (that is, the more pipeline stages there are), the worse the problems get. For reasonably simple processors, there are significant benefits in introducing pipelines from three to five stages long, but beyond this the law of diminishing returns begins to apply and the added costs and complexity outweigh the benefits.
Pipelines clearly benefit from all instructions going through a similar sequence of steps. Processors with very complex instructions where every instruction behaves differently from the next are hard to pipeline. In 1980 the complex instruction set microprocessor of the day was not pipelined due to the limited silicon resource, the limited design resource and the high complexity of designing a pipeline for a complex instruction set.
Branch instructions result in even worse pipeline behaviour since the fetch step of the following instruction is affected by the branch target computation and must therefore be deferred. Unfortunately, subsequent fetches will be taking place while the branch is being decoded and before it has been recognized as a branch, so the fetched instructions may have to be discarded.

Pipeline hazards
It is relatively frequent in typical computer programs that the result from one instruction is used as an operand by the next instruction. When this occurs the pipeline operation shown in breaks down, since the result of instruction 1 is not available at the time that instruction 2 collects its operands. Instruction 2 must therefore stall until the result is available, giving the behavior. This is a read-after-write pipeline hazard.

Advantages:
1. Increase in the number of pipeline stages increases the number of instructions executed simultaneously.
2. Faster ALU can be designed when pipelining is used.
3. Pipelined CPU's works at higher clock frequencies than the RAM. Pipelining increases the overall performance of the CPU.

Applications:
1. Digital signal processing (DSP) systems
2. Microprocessors.

Superscalar: Superscalar architecture is a method of parallel computing used in many processors. In a superscalar computer, the central processing unit (CPU) manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle.

Advantages:
The cycle time of the processor is reduced, thus increasing instruction issue-rate in most cases

Applications: Microprocessor

Application Specific System Processor (ASSP)
Application Specific System Processor (ASSP) ASSP is dedicated for faster processing and useful for applications like real-time video processing which incorporates lots of processing before transmitting. Assume embedded system for real time video processing. Real time video

arises for digital television, high definition TV decoder, set-top box, DVD players, web phones video conferencing etc. Processors which are dedicated to these types of specific tasks are the ASSP. These processors are configured and interfaced with the rest of the embedded system.

Assume an embedded system for specific protocol interconnects through some bus architecture to another system. In such applications some encryption and decryption is required. Also some RTOS features are necessary. If the software alone is used for this type of applications, then it will take a longer time. Therefore hardwired solutions are designed to meet the application specific system. Processor designed for these systems are the ASSP.

For example ASSP chip i2ehip fhttp://www, i2chip.com) has TCP, IP, and Ethernet 10/100 MAC(Media Access Control) hardwired logic included into it. In practice, an ASSP is used as an additional processing unit for running the application specific tasks in place of processing using embedded software.

The i2Chip W3100A is an LSI of hardware protocol stack that provides an easy, low-cost solution for high-speed Internet connectivity for digital devices by allowing simple installation of TCP/IP stack in the hardware. The W3100A offers system designers a quick, easy way to add Ethernet networking functionality to any product. Implementing this LSI into a system can completely offload Internet connectivity and processing standard protocols from the system, thereby significantly reducing the software development cost. The W3100A contains TCP/IP Protocol Stacks such as TCP, UDP, IP, ARP and ICMP protocols, as well as Ethernet protocols such as Data Link Control and MAC protocol. The W3100A offers a socket API (Application Programming Interface) that is similar to the windows socket API. The chip offers Intel and Motorola MCU (8051, i386, 6811 tested) bus interface and I 2 C for upper-layer and supports standard interface. The W3100A can be applied to handheld devices including Internet phones, VoIP SOC chips, Internet MP3 players, handheld medical devices, LAN cards for Web servers, cellular phones and many other nonportable electronic devices such as large consumer electronic products.

Application Specific Instruction Processor (ASIP)

For a number of applications GPP core may not be a suitable solution. For various security application, smart card, video game, mobile Internet, Gbps transceiver, Gbps LAN, missile system needs a special processing unit on a VLSI design circuit to function as a processor. These units are called Application Specific Instruction Processor (ASIP). Sometime for an application both configurable processor (FPGA or ASIP) and non - configurable processor (DSP or microprocessor or microcontroller) might be needed on a chip. Generally this type of applications are very important in some killer applications (application which is useful to millions of people) such as HDTV, cell-phone etc.

ASIP is a processor with instruction set designed for specific application area on VLSI chip or core. ASIP examples are microcontroller, I/O, DSP, Media, network or other domain-specific processors. ASIP can be designed with some VLSI design tools. ASIP are programmed with the instructions of the function related to Digital signal processing, control signal processing.etc.

Embedded processor / embedded microcontroller

Embedded processor is a processor with special features that allow it to embed multiple processes in to a system.

Real time applications and aerodynamics are two areas whereas fast, precise, and intensive calculations with fast content switching (from one program to another), are essential.

Embedded processor should have following capabilities.
1. Fast context switching.
2. 32 bit or 64 bit addition/multiplication with no share data problem in the operation.
3. 32 bit RISC core for fast, precise, intensive calculations

Embedded microcontroller is specially designed microcontroller and should have following capabilities.
1. Microcontroller has RAM, large flash, ROM, interrupt handlers, devices and peripherals and no external memory, or device or peripherals required.
2. Fast context switching and therefore lower latency of task in complex real operations. For example ARM and 68HC1x microcontroller saves all CPU registers very fast.

1.4 AVR Overview

The AVR is a modified Harvard architecture machine, where program and data are stored in separate physical memory systems that appear in different address spaces, but having the ability to read data items from program memory using special instructions.

Basic families

AVRs are generally classified into following:

tinyAVR — the ATtiny series

1 0.5–16 kB program memory
2 6–32-pin package
3 Limited peripheral set

megaAVR — the ATmega series

1 4–256 kB program memory
2 28–100-pin package
3 Extended instruction set (multiply instructions and instructions for handling larger program memories)
4 Extensive peripheral set

XMEGA — the ATxmega series

1 16–384 kB program memory
2 44–64–100-pin package (A4, A3, A1)
3 32-pin package : XMEGA-E (XMEGA8E5)

4 Extended performance features, such as DMA, "Event System", and cryptography support.
5 Extensive peripheral set with ADCs

Application-specific AVR

megaAVRs with special features not found on the other members of the AVR family, such as LCD controller, USB controller, advanced PWM, CAN, etc.

FPSLIC (AVR with FPGA)

1 FPGA 5K to 40K gates
2 SRAM for the AVR program code, unlike all other AVRs
3 AVR core can run at up to 50 MHz

32-bit AVRs

In 2006 Atmel released microcontrollers based on the 32-bit AVR32 architecture. They include SIMD and DSP instructions, along with other audio- and video-processing features. This 32-bit family of devices is intended to compete with the ARM-based processors. The instruction set is similar to other RISC cores, but it is not compatible with the original AVR or any of the various ARM cores.

Device architecture

Flash, EEPROM, and SRAM are all integrated onto a single chip, removing the need for external memory in most applications. Some devices have a parallel external bus option to allow adding additional data memory or memory-mapped devices. Almost all devices (except the smallest TinyAVR chips) have serial interfaces, which can be used to connect larger serial EEPROMs or flash chips.

Program memory

Program instructions are stored in non-volatile flash memory. Although the MCUs are 8-bit, each instruction takes one or two 16-bit words.

The size of the program memory is usually indicated in the naming of the device itself (e.g., the ATmega64x line has 64 kB of flash, while the ATmega32x line has 32 kB).

There is no provision for off-chip program memory; all code executed by the AVR core must reside in the on-chip flash. However, this limitation does not apply to the AT94 FPSLIC AVR/FPGA chips.

Internal data memory

The data address space consists of the register file, I/O registers, and SRAM.

Internal registers

The AVRs have 32 single-byte registers and are classified as 8-bit RISC devices.

In the tinyAVR and megaAVR variants of the AVR architecture, the working registers are mapped in as the first 32 memory addresses (0000_{16}–$001F_{16}$), followed by 64 I/O registers (0020_{16}–$005F_{16}$). In devices with many peripherals, these registers are followed by 160 "extended I/O" registers, only accessible as memory-mapped I/O (0060_{16}–$00FF_{16}$).

Actual SRAM starts after these register sections, at address 0060_{16} or, in devices with "extended I/O", at 0100_{16}.

Even though there are separate addressing schemes and optimized opcodes for accessing the register file and the first 64 I/O registers, all can still be addressed and manipulated as if they were in SRAM.

The very smallest of the tinyAVR variants use a reduced architecture with only 16 registers (r0 through r15 are omitted) which are not addressable as memory locations. I/O memory begins at address 0000_{16}, followed by SRAM. In addition, these devices have slight deviations from the standard AVR instruction set. Most notably, the direct load/store instructions (LDS/STS) have been reduced from 2 words (32 bits) to 1 word (16 bits), limiting the total direct addressable memory (the sum of both I/O and SRAM) to 128 bytes. Conversely, the indirect load instruction's (LD) 16-bit address space is expanded to also include non-volatile memory such as Flash and configuration bits; therefore, the LPM instruction is unnecessary and omitted.

In the XMEGA variant, the working register file is not mapped into the data address space; as such, it is not possible to treat any of the XMEGA's working registers as though they were SRAM. Instead, the I/O registers are mapped into the data address space starting at the very beginning of the address space. Additionally, the amount of data address space dedicated to I/O registers has grown substantially to 4096 bytes (0000_{16}–$0FFF_{16}$). As with previous generations,however, the fast I/O manipulation instructions can only reach the first 64 I/O register locations (the first 32 locations for bitwise instructions). Following the I/O registers, the XMEGA series sets aside a 4096 byte range of the data address space, which can be used optionally for mapping the internal EEPROM to the data address space (1000_{16}–$1FFF_{16}$). The actual SRAM is located after these ranges, starting at 2000_{16}.

GPIO ports

Each GPIO port on a tiny or mega AVR drives up to eight pins and is controlled by three 8-bit registers: DDR*x*, PORT*x* and PIN*x*, where *x* is the port identifier.

DDR*x*: Data Direction Register configures the pins as either inputs or outputs.

PORT*x*: Output port register. Sets the output value on pins configured as outputs. Enables or disables the pull-up resistor on pins configured as inputs.

PIN*x*: Input register, used to read an input signal. On some devices (but not all, check the datasheet), this register can be used for pin toggling: writing a logic one to a PIN*x*bit toggles the corresponding bit in PORT*x*, irrespective of the setting of the DDR*x* bit.[8]

xmegaAVR have additional registers for push/pull, totem-pole and pull-up configurations.

EEPROM

Almost all AVR microcontrollers have internal EEPROM for semi-permanent data storage. Like flash memory, EEPROM can maintain its contents when electrical power is removed.

In most variants of the AVR architecture, this internal EEPROM memory is not mapped into the MCU's addressable memory space. It can only be accessed the same way an external peripheral device is, using special pointer registers and read/write instructions, which makes EEPROM access much slower than other internal RAM.

However, some devices in the SecureAVR (AT90SC) family use a special EEPROM mapping to the data or program memory, depending on the configuration. The XMEGA family also allows the EEPROM to be mapped into the data address space.

Since the number of writes to EEPROM is not unlimited — Atmel specifies 100,000 write cycles in their datasheets — a well-designed EEPROM write routine should compare the contents of an EEPROM address with desired contents and only perform an actual write if the contents need to be changed.

Note that erase and write can be performed separately in many cases, byte-by-byte, which may also help prolong life when bits only need to be set to all 1s (erase) or selectively cleared to 0s (write).

Program execution

Atmel's AVRs have a two-stage, single-level pipeline design. This means the next machine instruction is fetched as the current one is executing. Most instructions take just one or two clock cycles, making AVRs relatively fast among eight-bit microcontrollers.

The AVR processors were designed with the efficient execution of compiled C code in mind and have several built-in pointers for the task.

MCU speed

The AVR line can normally support clock speeds from 0 to 20 MHz, with some devices reaching 32 MHz Lower-powered operation usually requires a reduced clock speed. All recent (Tiny, Mega, and Xmega, but not 90S) AVRs feature an on-chip oscillator, removing the need for external clocks or resonator circuitry. Some AVRs also have a system clock prescaler that can

divide down the system clock by up to 1024. This prescaler can be reconfigured by software during run-time, allowing the clock speed to be optimized.

Since all operations (excluding multiplication and 16-bit add/subtract) on registers R0–R31 are single-cycle, the AVR can achieve up to 1 MIPS per MHz, i.e. an 8 MHz processor can achieve up to 8 MIPS. Loads and stores to/from memory take two cycles, branching takes two cycles. Branches in the latest "3-byte PC" parts such as ATmega2560 are one cycle slower than on previous devices.

Development

AVRs have a large following due to the free and inexpensive development tools available, including reasonably priced development boards and free development software. The AVRs are sold under various names that share the same basic core, but with different peripheral and memory combinations. Compatibility between chips in each family is fairly good, although I/O controller features may vary.

See external links for sites relating to AVR development.

Features

Current AVRs offer a wide range of features:

1. Multifunction, bi-directional general-purpose I/O ports with configurable, built-in pull-up resistors
2. Multiple internal oscillators, including RC oscillator without external parts
3. Internal, self-programmable instruction flash memory up to 256 kB (384 kB on XMega)
4. In-system programmable using serial/parallel low-voltage proprietary interfaces or JTAG
5. Optional boot code section with independent lock bits for protection
6. On-chip debugging (OCD) support through JTAG or debugWIRE on most devices
7. The JTAG signals (TMS, TDI, TDO, and TCK) are multiplexed on GPIOs. These pins can be configured to function as JTAG or GPIO depending on the setting of a fuse bit, which can be programmed via ISP or HVSP. By default, AVRs with JTAG come with the JTAG interface enabled.
8. debugWIRE uses the /RESET pin as a bi-directional communication channel to access on-chip debug circuitry. It is present on devices with lower pin counts, as it only requires one pin.
9. Internal data EEPROM up to 4 kB
10. Internal SRAM up to 16 kB (32 kB on XMega)
11. External 64 kB little endian data space on certain models, including the Mega8515 and Mega162.

12. The external data space is overlaid with the internal data space, such that the full 64 kB address space does not appear on the external bus and accesses to e.g. address 0100_{16} will access internal RAM, not the external bus.
13. In certain members of the XMega series, the external data space has been enhanced to support both SRAM and SDRAM. As well, the data addressing modes have been expanded to allow up to 16 MB of data memory to be directly addressed.
14. AVRs generally do not support executing code from external memory. Some ASSPs using the AVR core do support external program memory.
15. 8-bit and 16-bit timers
16. PWM output (some devices have an enhanced PWM peripheral which includes a dead-time generator)
17. Input capture that record a time stamp triggered by a signal edge
18. Analog comparator
19. 10 or 12-bit A/D converters, with multiplex of up to 16 channels
20. 12-bit D/A converters
21. A variety of serial interfaces, including
22. I²C compatible Two-Wire Interface (TWI)
23. Synchronous/asynchronous serial peripherals (UART/USART) (used with RS-232, RS-485, and more)
24. Serial Peripheral Interface Bus (SPI)
25. Universal Serial Interface (USI): a multi-purpose hardware communication module that can be used to implement an SPI,[10] I^2C[11][12] or UART interface.
26. Brownout detection
27. Watchdog timer (WDT)
28. Multiple power-saving sleep modes
29. Lighting and motor control (PWM-specific) controller models
30. CAN controller support
31. USB controller support
32. Proper full-speed (12 Mbit/s) hardware & Hub controller with embedded AVR.
33. Also freely available low-speed (1.5 Mbit/s) (HID) bitbanging software emulations
34. Ethernet controller support
35. LCD controller support
36. Low-voltage devices operating down to 1.8 V (to 0.7 V for parts with built-in DC–DC upconverter)
37. picoPower devices
38. DMA controllers and "event system" peripheral communication.
39. Fast cryptography support for AES and DES

1.5 AVR ATmega328 Microcontroller High-Level Block Diagram

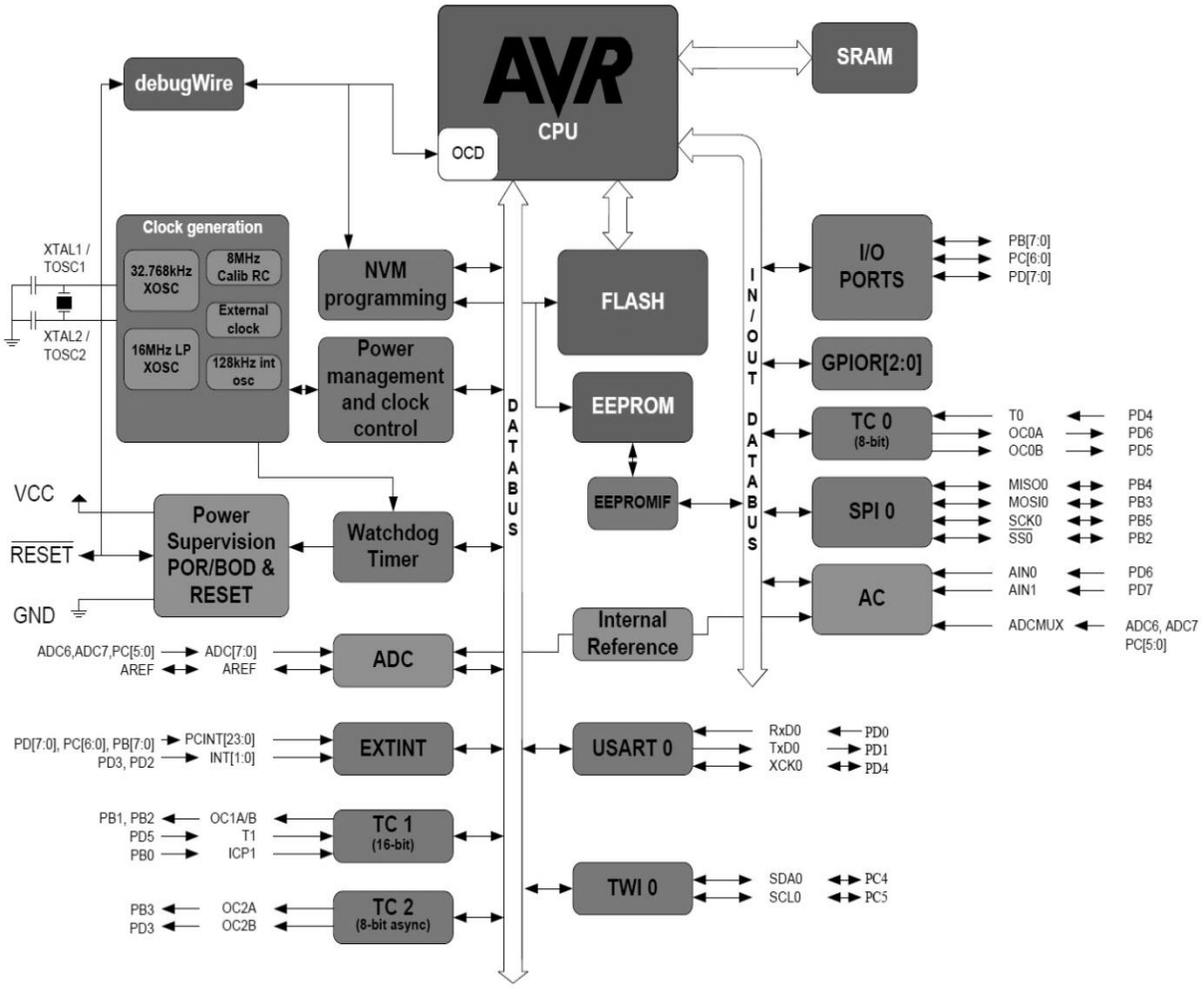

Fig 1.1ATmega 328 block diagram

1.6 ATmega328 Pin-out

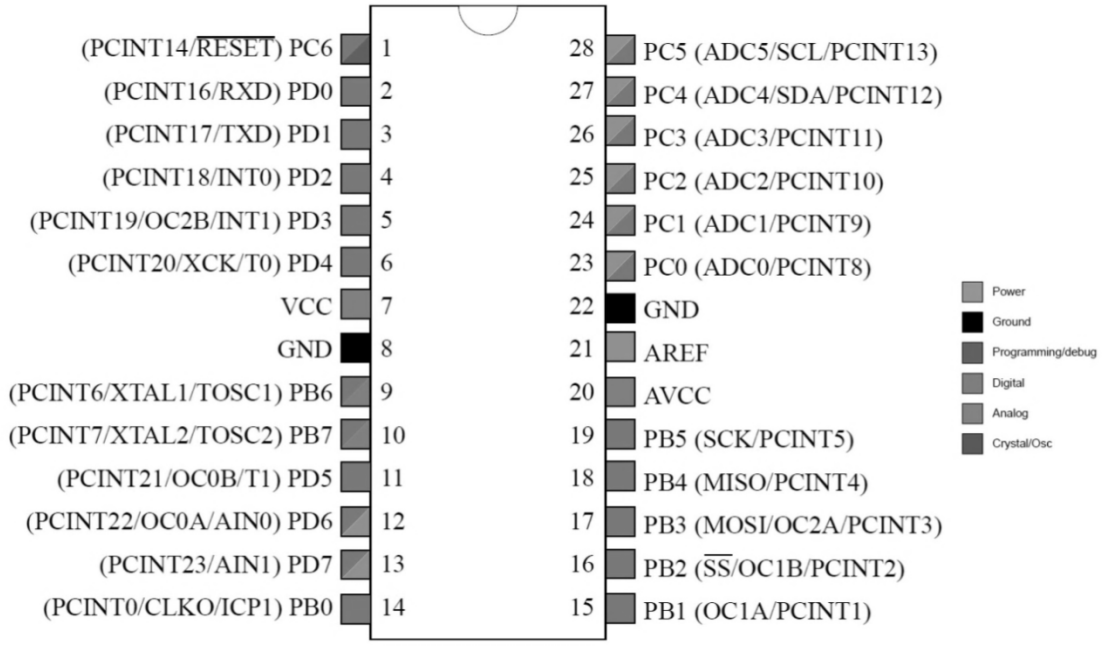

Fig 1.2 ATmega 328 Pin Diagram

1.7 Programming interfaces

There are many means to load program code into an AVR chip. The method to program AVR chips varies from AVR family to family. Most of the methods described below use the RESET line to enter programming mode. In order to avoid the chip accidentally entering such mode, it is advised to connect a pull-up resistor between the RESET pin and the positive power supply.

ISP

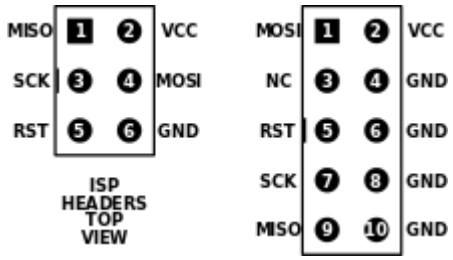

Fig 1.3ISP header 6 and 10-pin diagrams

The in-system programming (ISP) programming method is functionally performed through SPI, plus some twiddling of the Reset line. As long as the SPI pins of the AVR are not connected to anything disruptive, the AVR chip can stay soldered on a PCB while reprogramming. All that is needed is a 6-pin connector and programming adapter. This is the most common way to develop with an AVR.

The Atmel AVRISP mkII device connects to a computer's USB port and performs in-system programming using Atmel's software.

AVRDUDE (AVR Downloader/UploaDEr) runs on Linux, FreeBSD, Windows, and Mac OS X, and supports a variety of in-system programming hardware, including Atmel AVRISP mkII, Atmel JTAG ICE, older Atmel serial-port based programmers, and various third-party and "do-it-yourself" programmers.

Bootloader: Microcontrollers are usually programmed through a programmer unless you have a piece of firmware in your microcontroller that allows installing new firmware without the need of an external programmer. This is called a bootloader.

Most AVR models can reserve a bootloader region, 256 Byte to 4 KB, where re-programming code can reside. At reset, the bootloader runs first and does some user-programmed determination whether to re-program or to jump to the main application. The code can re-program through any interface available, or it could read an encrypted binary through an Ethernet adapter like PXE.

Optiboot Bootloader is a Small and Fast Bootloader used for Arduino and other Atmel AVR chips.

ROM

AVRs are available with a factory mask-ROM rather than flash for program memory. Because of the large up-front cost and minimum order quantity, a mask-ROM is only cost-effective for high-production runs.

Debugging interfaces

The AVR offers several options for debugging, mostly involving on-chip debugging while the chip is in the target system.

debugWIRE

debugWIRE™ is Atmel's solution for providing on-chip debug capabilities via a single microcontroller pin. It is particularly useful for lower pin count parts which cannot provide the four "spare" pins needed for JTAG. The JTAGICE mkII, mkIII and the AVR Dragon support debugWIRE. debugWIRE was developed after the original JTAGICE release, and now clones support it.

JTAG

The Joint Test Action Group (JTAG) feature provides access to on-chip debugging functionality while the chip is running in the target system. JTAG allows accessing internal

memory and registers, setting breakpoints on code, and single-stepping execution to observe system behaviour.

1.8 Arduino:

Introduction: Arduino is an open-source hardware and software platform, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices. Its products are licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the manufacture of Arduino boards and software distribution by anyone. Arduino boards are available commercially in preassembled form or as do-it-yourself (DIY) kits.

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers can be programmed using C and C++ programming languages. In addition to usingtraditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project.

Table 1.1 Atmel chips used in Arduino boards

Chip Number	On-Chip Flash	RAM	I/O pins	Pin numbers	Arduino Board
ATmega16	16K	1K	14	28	Nano or Uno
ATmega328	32K	2K	14	28	Nano or Uno
ATmega328p	(p) stands for low (Pico) Power consumption other features same as 328				
ATmega2560	256K	4K	54	100	Mega

Fig 1.5Arduino R3 UNO Board

1.10 Arduino IDE:

Arduino IDE Features:

1. **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
2. **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
3. **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

Sketch:

A *sketch* is a program written with the Arduino IDE. Sketches are saved on the development computer as text files with the file extension **.ino**. Arduino Software (IDE) pre-1.0 saved sketches with the extension **.pde**.

Arduino C/C++ program consists of two functions:

- `setup()`: This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to the function `main()`.
- `loop()`: After `setup()` function exits (ends), the `loop()` function is executed repeatedly in the main program. It controls the board until the board is powered off or is reset. It is analogous to the function `while(1)`

Table 1.2Arduino Functions Used for I/O:

Function	**Description Syntax**	
pinMode();	Designate the pin as OUTPUT or INPUT	pinMode(pin#, mode);
digitalWrite();	Write a LOW or HIGH to a pin	digitalWrite(pin#, value);
digitalRead();	Read the status of pin	digitalRead(pin#);
delay();	Create a delay in millisecond	delay(ms);

1.11Arduino Interfacing:

LED Interfacing:

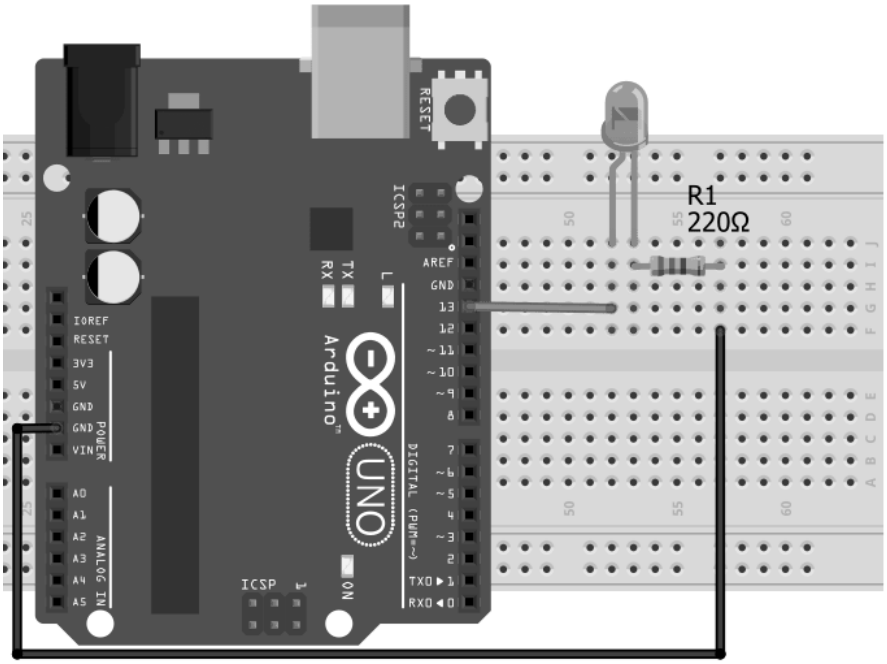

Fig 1.6Interfacing of LED to Arduino board

LED (Red) attached to pin 13 on an Arduino compatible board with ground.

Most Arduino boards contain a light-emitting diode (LED) and a current limiting resistor connected between pin 13 and ground, which is a convenient feature for many tests and program functions.

Functions pinMode(), digitalWrite(), and delay(), which are provided by the internal libraries included in the IDE environment.

The following program will blink LED with certain delay.

```
#define LED_PIN 13                    // Pin number attached to LED.

void setup(){
pinMode(LED_PIN,OUTPUT);// Configure pin 13 to be a digital output.
}

void loop(){
digitalWrite(LED_PIN,HIGH);// Turn on the LED.
delay(1000);// Wait 1 second (1000 milliseconds).
digitalWrite(LED_PIN,LOW);// Turn off the LED.
delay(1000);// Wait 1 second.
}
```

Relay Interfacing:

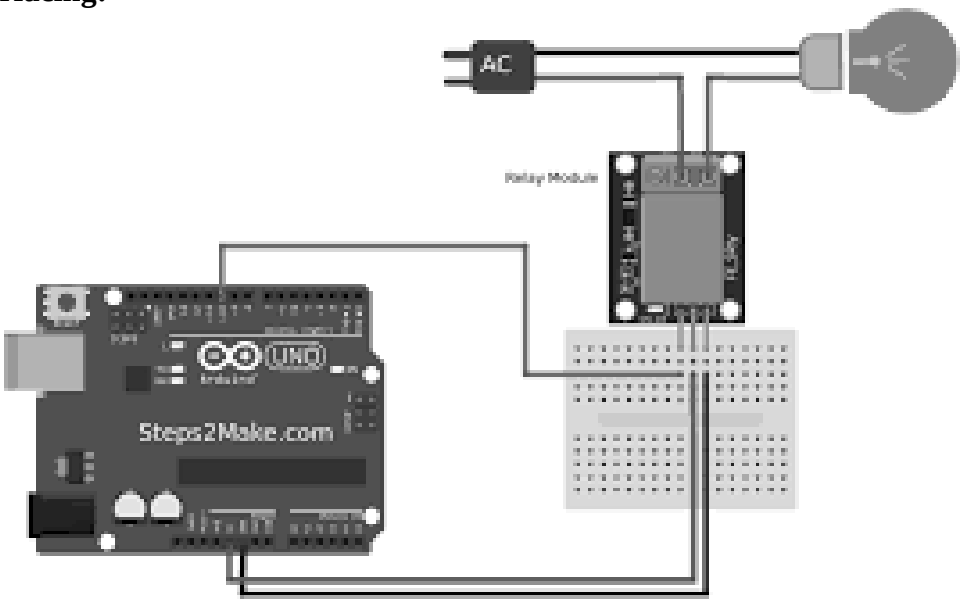

Fig 1.7Interfacing of Relay to Arduino board

```
//Relay is turned on for 5 seconds and then off.

void setup()
{
// Initialise the Arduino data pins for OUTPUT
```

```
  pinMode(RELAY, OUTPUT);
}

 void loop()
{
   digitalWrite(RELAY,LOW);           // Turns ON Relays
   delay(5000);                       // Wait 5 seconds
   digitalWrite(RELAY,HIGH);          // Turns Relay Off

 }
```

DC motor Interfacing:

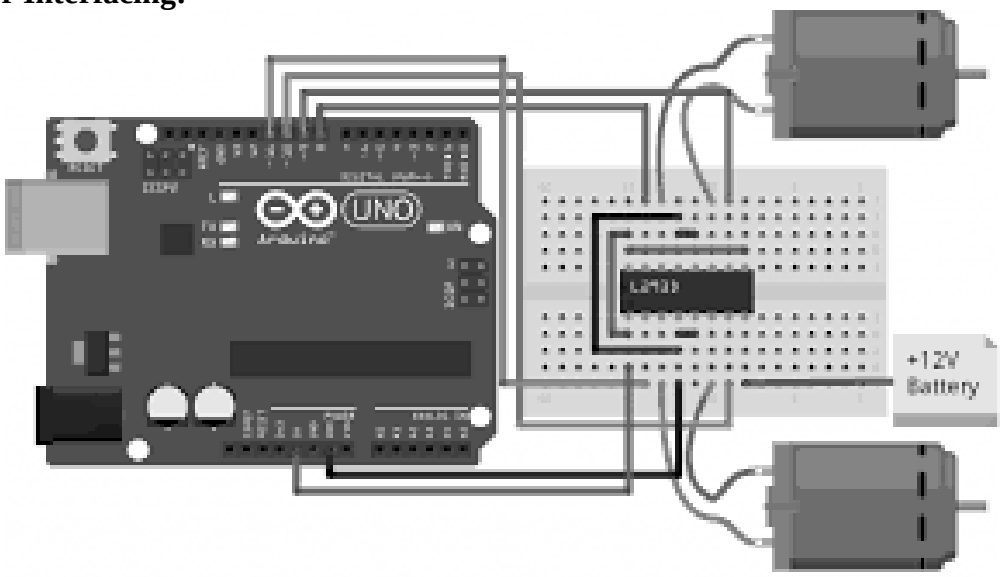

Fig 1.8Interfacing of DC motor to Arduino board

A direct current, or DC, motor is the most common type of motor. DC motors normally have just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

To control the direction of the spin of DC motor, without changing the way that the leads are connected, you can use a circuit called an H-Bridge. An H bridge is an electronic circuit that can drive the motor in both directions. H-bridges are used in many different applications, one of the most common being to control motors in robots. It is called an H-bridge because it uses four transistors connected in such a way that the schematic diagram looks like an "H."

Since we will be controlling only one motor in this tutorial, we will connect the Arduino to IN1 (pin 5), IN2 (pin 7), and Enable1 (pin 6) of the L298 IC. Pins 5 and 7 are digital, i.e. ON or OFF inputs, while pin 6 needs a pulse-width modulated (PWM) signal to control the motor speed. IN1 pin of the L298 IC is connected to pin 8 of the Arduino while IN2 is connected to pin 9. These

two digital pins of Arduino control the direction of the motor. The EN A pin of IC is connected to the PWM pin 2 of Arduino. This will control the speed of the motor.

1. Connect 5V and ground of the IC to 5V and ground of Arduino.
2. Connect the motor to pins 2 and 3 of the IC.
3. Connect IN1 of the IC to pin 8 of Arduino.
4. Connect IN2 of the IC to pin 9 of Arduino.
5. Connect EN1 of IC to pin 2 of Arduino.
6. Connect SENS A pin of IC to the ground.
7. Connect the Arduino using Arduino USB cable and upload the program to the Arduino using Arduino IDE software or Arduino Web Editor.
8. Provide power to the Arduino board using power supply, battery or USB cable.

The motor should now run first in the clockwise (CW) direction for 3 seconds and then counter-clockwise (CCW) for 3 seconds.

Code:

```
constintpwm = 2 ; //initializing pin 2 as pwm
constint in_1 = 8 ;
constint in_2 = 9 ;
 //For providing logic to L298 IC to choose the direction of the DC motor

void setup()
{
pinMode(pwm,OUTPUT) ; //we have to set PWM pin as output
pinMode(in_1,OUTPUT) ; //Logic pins are also set as output
pinMode(in_2,OUTPUT) ;
}

void loop()
 {
//For Clock wise motion , in_1 = High , in_2 = Low
digitalWrite(in_1,HIGH) ;
digitalWrite(in_2,LOW) ;
analogWrite(pwm,255) ;

 /*setting pwm of the motor to 255 we can change the speed of rotaion by
chaningpwm input but we are only using arduino so we are using higest value
to driver the motor */

//Clockwise for 3 secs
delay(3000) ;
```

```
//For brake
digitalWrite(in_1,HIGH) ;
digitalWrite(in_2,HIGH) ;
delay(1000) ;

//For Anti Clock-wise motion - IN_1 = LOW , IN_2 = HIGH
digitalWrite(in_1,LOW) ;
digitalWrite(in_2,HIGH) ;
delay(3000) ;

//For brake
digitalWrite(in_1,HIGH) ;
digitalWrite(in_2,HIGH) ;
delay(1000) ; }
```

1.12ARM Introduction:

ARM designs microprocessor technology that lies at the heart of advanced digital products, from mobile phones and digital cameras to games consoles and automotive systems, and is leading intellectual property (IP) provider of high-performance, low-cost, power-efficient RISC processors, peripherals, and system-on-chip (SoC) designs through involvement with organizations such as the Virtual Socket Interface Alliance (VSIA) and Virtual Component Exchange (VCX). ARM also offers design and software consulting services. ARM's architecture is compatible with all four major platform operating systems: Symbian OS, Palm OS, Windows CE, and Linux. As for software, ARM also works closely with with its partners to provide optimized solutions for existing market segments. These benefits are making the ARM company a complete solution provider.

With over forty partners licensed to use its architecture, ARM enables original equipment Manufacturers (OEM) to realize an accelerated time-to-market through complete product offerings, such as Prime Cell Peripherals, embedded software IP, development tools, training, and support The Company offers a complete solution that is essential to the manufacturing process. Although ARM does not manufacture processors itself, ARM licenses its cores to semi-conductor manufacturers to be integrated into ASIC standards and then the company in using test chips manufactured by its partners to measure and validate the functionality of the core.

ARM is able to accelerate OEM time-to-market by capitalizing on its architecture. By providing the IP and supporting services, customers can gain a jump on their design cycle and obtain a competitive edge in their targeted market segment. At that point, the architecture is portable to further product generations or applications as all code creation is directly compatible with any future architecture produced by ARM.

ARM's Global Technology Partner Network is the largest in the industry, spanning from Semiconductor manufacturers to distributors. ARM has worked diligently to ensure that the partnerships provide proven solutions in real-time operating systems (RTOS), EDA tools, development systems, applications software, and design consulting, all built around the ARM

Architecture. The ARM Company is working to establish standards, not just within the company, but across the industry by taking advantage of leadership opportunities in the creation of standards. ARM is the industry standard embedded microprocessor architecture, and is a leader in low-power high performance cores. ARM also has a large partner network supporting the entire design and development cycle. ARM is a full-solutions provider, supporting a broad range of applications.

1.13 Basic of ARM architecture:

ARM architecture is not synonymous with the single organization. But there is certain commonality across the different variants.

Basic ARM Organization.

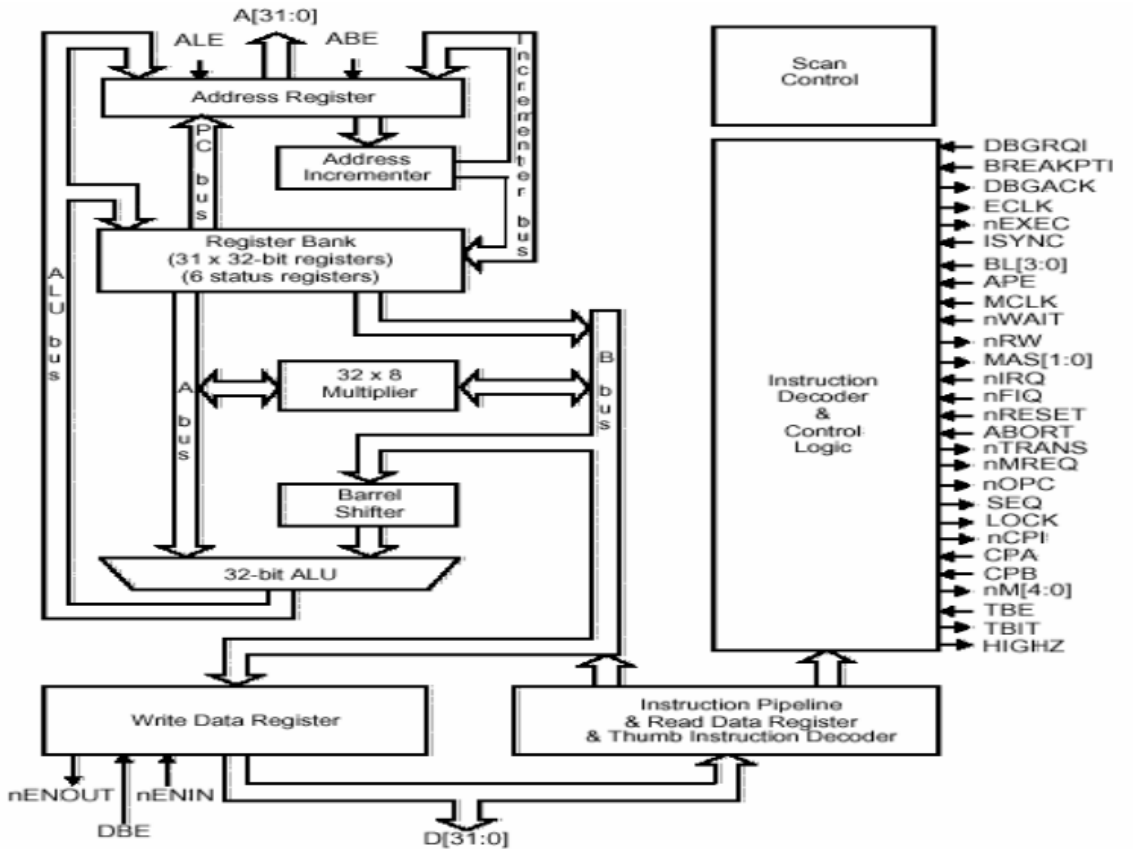

Fig 1.15ARM7 TDMI Core Diagram

It consists of register bank. It is connected to the ALU by two buses A and B. A is connected directly to ALU and B is connected through Barrel shifter. This Barrel shifter can actually preprocess the data which can come from one of this source registers; and the Barrel shifter can shift to the left, shift to the right or even rotate the data before it is fed to the ALU. Now, since all of these blocks i.e.ALU, Barrel shifter is also combinational circuit. So, the entire, all these operations that is operation that ALU carries out as well as operation that Barrel shifter carries out can take place in one cycle itself and that actually splits up to the operation execution speed.
Register bank can generate the address also. In fact the PC address is, PC also is part of the register bank and that can generate the address. As well as the other register banks, can be made

use of for generation for manipulation of address. Because registers are in a way symmetric they can have both address as well as the data and they can be operated in a symmetric way. The PC generates the address for the instruction.
Other operations can also be done using this registers. Instruction decodes and control provides a control signal. Address bus is 0 to 31 that means it is a 32 bit. Data buses are also 32 bits, so it is basically a 32 bit processor. It can operate on 32 bit operands and the addresses that it generates are also 32 bit. Register bank has a very prominent role.

All registers are 32 bits because data bus is 32 bit, operating at 32 bit operands as well as addresses are also 32 bits. There are 16 data registers in user mode and 2 data registers are visible.
User mode is a common operating mode. Used by user when running program on ARM.
Data registers are typically r0 to r15 and in fact in ARM, all registers are refer to by r followed by a number. So, here we are talking about data registers r0 to r15 which are visible in the user mode. Out of these registers, 3 registers perform special function they are r13, r14 and r15. r13 is a stack pointer, so this stack pointer refers to the entry point on the stack and this is critical for implementation of a stack in the memory. r14 is a link register. This link register is a register where return address is put whenever a subroutine is called. Here, we have got a single link register and in the link register the return address is put in. Then r15 is the program counter and obviously the current instruction what is being executed will be pointed to by the content of r15.

Now, depending on the context registers r13 and r14 can also be used as general purpose registers. In addition there are 2 status registers. CPSR, (current program status register) and SPSR s(saved program status register). These are basically the status registers which are not data registers. So, here in this registers effectively the status of the current execution is being captured. In fact this status can include status of your program as well as that of the processor.

And when it is operating in your 32 bit it is assume that all instructions are word aligned. That means all 32 bit instructions start at 32 bit boundary. And what does that imply, that implies that PC value is stored effectively in bits from 2 to 31, bit numbers 2 to 31, with bits 1, 0 effectively undefined or not really useful for referring to an instruction. Now, obviously this discussion refers to one fact that my 32 bit address in ARM refers to byte locations. Each byte with associated with a unique address so, if I am talking about 32 bit boundaries that means effectively I am talking about what blocks of 4 bytes. So, if I have one instruction starting at location 0 then that instruction will occupy location 0, 1, 2 as well as 3, fine. The next instruction would be located at 4 so, therefore these 2 bits, the least significant bits of PC that is r15 or in a way do not care for operations, okay. So, that is why we say that PC value is effectively stored in bits from 2 to 31.

Now, let us look at the status register CPSR. CPSR is- what is the current program status register; it has got a number of bits. Again it will be a 32 bit register; it is not that all bits are used at the same time. The condition code flags which occupy the higher that MSPs that is most significant bits in the status register; they are standard flags which reflect various arithmetic conditions. I have got negative flag if there is a negative result from ALU which is typically the most significant bit, it is associated the most significant bit. If it is one then it can be interpreted as a negative result when we are doing signed arithmetic set, Z indicates 0, C is the carry and V is overflow. There is this sticky overflow flag, this is with reference to saturation arithmetic and

there is interrupt, disable bits here which are for, there are two levels of interrupts. We shall come to that point later on.

So, we can enable or disable these two levels of interrupts by using these 2 bits. This T bit indicates whether you are in thump mode or not thumb mode because when we actually have an embedded 16 bit processor into the 32 bit architecture, we shall be making use of this T bit to know whether I am operating in the thumb mode or ordinary 32 bit mode. And rest are mode bits and these mode bits really defined what is called the mode of processors operation. The point I was discussing in terms of registers, we are telling that you can use about16 data registers, we can use 16 data registers in your program and normal operation and that is user mode, that mode specified in these bits. Now, before going into these processor modes in detail, let us briefly look at this sticky overflow flag. What is saturation? Saturation means when we reach the maximum value or the minimum value because of an arithmetic operation which may have overflow or underflow.

So, we call the processor modes either privileged or non-privileged mode. In a privileged mode you expect to have full read-write access to the CPSR. In a non-privileged mode only read access to the control field of CPSR but read-write access to the condition flags.

Now, try to understand this- what is the implication of these privileged and non-privileged modes. In a privileged mode what can happen actually, in a privileged modes as you can change the control bits that means you can have a full read as well as write access of the control bits. You can actually change the processor mode, you can enable, disable the interrupts. So, this is a privileged operation. In a non- privileged mode, these control fields can be simply read but cannot be changed, but the condition flags which can change because of an arithmetic operation would normally reflect the status of the arithmetic operation and that should be remain write enable even in non-privileged modes.

So, typically you will find that when we talk about these kind of operations, a typically user program is expected to run in a non- privileged mode because in user program is normally not expected to change the control bits.

And in a privileged mode typically you will expect the OS or the supervisory cell to run. Since we are targeting for ARM for more sophisticated applications, typically there would be an OS running in an ARM based system under which user programs are expected to execute. The OS is typically expected to be running in privileged mode and user applications running in non-privileged mode. In fact ARM has got 7 modes and these 7 modes can be now classified as privileged and non- privileged. In fact the privileged modes are abort, first interrupt request, supervisor system and another is undefined.

Now, privileged modes represent different scenarios. Abort is a mode when there is a failed attempt to access memory. This can happen for variety of reasons but this reasons we shall look at when we consider the memory architecture subsequently. But this is a particular mode in which the processor goes in when it detects that there is a failure to access the memory location.

The first interrupt request and interrupt request correspond to interrupt levels available on ARM. So, when a particular kind of interrupt occurs ARM processor goes into other first interrupt mode or interrupt request mode.

Supervisor mode is a state in which processor goes in after reset and generally it is a mode in which the OS kernel is supposed to operate because obviously when the processor is reset, the first thing that its excepted to execute is a operating system code and not user application of

program. So, this is a supervisor mode in which the processor goes in when the reset happens. The other two privileged modes are system mode and another mode is called undefined. In a system mode, is a special version of user mode that allows full read-write access of CPSR.

And it is also targeted for supervisory applications; many of the OS routines can be configured to run in the system mode. The undefined mode, processer enters this undefined mode when it encounters an undefined instruction that means when you are trying to use an illegal op-code for undefined instruction, the instruction undefined for particular processor, and then it goes into an undefined.

So, what you have found is that these privileged modes are primarily targeted for OS handling of special error conditions as well as that of interrupts and user mode is a mode intended for running user applications. Now, these modes have got associated with them a very interesting capability to manage the registers.

So, we go into the concept of what we call banked register in ARM architecture. ARM has got 37 registers in all and typically 20 registers are hidden from program at different times.

So, they are not visible registers and they are actually called banked registers and this banked registers becomes available only when processors is in a particular mode. In fact processors modes other than system mode have a set of associated banked registers that are subset of these 16 registers that we have talked about in the user mode, okay. And these banked registers have one-to-one mapping with the user mode registers. So, what happens, let us look at this, so I have use I am operating, let us say in a user mode, by some means I am in the user mode. So, in the user mode there are the 16 data registers which are available, okay and the current program which is getting executed, that status would get reflected in the CPSR register. Now, if the processor goes into some other mode, let us say that FIQ mode. FIQ is what- first interrupt; IRQ is interrupt request model. Now, in an FIQ mode, what we will find that I have got banked register r8, r9, r10, r11, r12 becoming available as well as r13 and r14.

What does that mean? These registers, I told you as got a one-to-one correspondence with the registers in the user mode that means effectively what I am getting in a first interrupt mode, I am getting a fresh copy of r8 to r14. I am getting a fresh copy of r8 to r14.

Now, what does that imply? It implies that if I am having an interrupt service routine which is operating in FIQ that is which is basically serving in the interrupt, in the first interrupt mode, it can use r8, r9 to r14 without bothering about what happens to the original content of these registers. See, we have told you that when you want to, you know otherwise also that when you go to the interrupt service routine and if I want to from there, if I want to come back to the original program and I have to do what, to come back to the correct state of computation, I need to store the registers in the stack. Now, storing the registers in the stack is R consisting of push a operation that will have the overhead and that is actually the software latency for interrupt processing. In the last class, we have looked that hardware latency, this becomes the software latency. The moment I have got a fresh copy of this registers; see if my ISR do not use any other register than this, I really do not meet to push this registers onto stack. So, I can minimize my software latency and in fact that is a reason why this mode is called first interrupt mode. In other one, the interrupt request mode what you have is simply r13 and r14 fresh copy is generated, okay. So, other registers the copy is not generated. So, other registers, you need to save if you are using them; so obviously it is not as fast as that of a first interrupt mode because here there will be some amount of software latency which will be involved. The similar thing is true for the other modes- the supervisory mode undefined, abort, all these modes have got a copy of r39, r14 but other register the fresh copy is not generated.

So, these original user mode registers have to be used; if they are to be retained they have to be safe in memory. So, what happens in case of CPSR? Now, corresponding to CPSR what you get in these modes what are called SPSR. So, what is the SPSR- save program state register. So, CPSR is copied into SPSR which becomes available in FIQ mode. So, when I return from this mode to say user mode, I have to take this content of SPSR back to CPSR because CPSR would be storing the current status; so when I am going back I should have the current status of the computation back with me. So, pictorially what we say here on the abort SPSR is each privileged mode except system mode has associated with it a save program status register or SPSR.

This SPSR is used to save the status of CPSR when the privileged mode is entered in order that the user state can be fully restored when the user process is resumed; this is particularly the job of SPSR. So, now if you summarize what we have saved that in the user mode I have got these 16 registers as well as CPSR.
In FIQ mode what I have got, I have got same r0 to r7 registers as that of user mode but a fresh copy of these registers, okay. Similar thing in IRG; IRG have got I can use the registers same as that of r0 to r12 that fresh copy of r13 and r1,4 okay and CPSR is showing here. So, this CPSR is copied into SPSR.

But, obviously in FIQ mode I got to have a CPSR which will reflect the state of the processors in that mode during execution of that program, okay. So, effectively what we are telling is that in these modes if you have writing onto this registers I shall overwrite the original data. See, if I have to save this data, if I have to get this data back, I need to save it in the stack. But in case of FIQ, I get a fresh copy of registers; so I need not save this register save on to the stack and thereby have reduced software latency.

The more changes can be made by directly writing onto CPSR or by hardware when the processor responds to exception or interrupt. And to return to user mode, a special return instruction is used that instructs the core to restore the original CPSR and banked registers. The ARM memory organization, in fact ARM can be configured either in little endian form or in big endian form.

And I have already told you that addresses are for each byte, okay. So, memory addresses decrease from top to bottom and left to right, this is what it has been shown and 32 bit word aligned is for 8 and 16 bit words also, so all these are aligned words, okay. So, and these configuration is has been shown for little endian. And this can be configured also as a big endian processor.
Now, we shall look at ARM instruction set because we shall be, we know now the data path of ARM; so we shall see through instructions how we can use this data path.

The instructions process data held in registers and access memory with load and store instructions. And that is typical of any RISC architecture and the classes of instructions are data processing branch, load-store software, and interrupt program status register instructions because these are the different roles. Today, we shall simply look at data processing instructions. Typically the ARM instruction set has got 3 address data processing instructions, okay- the two operands and one for destination

The other interesting feature of the instruction set is that you can do a conditional execution of each instruction. That means depending on certain condition, you can decide whether to execute an instruction or not. In fact this is a very special kind of branch instructions and I have told you, this is an enhancement in the RISC architecture to increase the computation throughput. The Barrel shifter enables shift and ALU together; so we shall have that enhancement of instruction set to do this operations and the other interesting thing we shall look at subsequently that you can actually increase the instruction set of the processor by adding on co-processors.
And how to do that, that will be a very interesting feature for ARM architecture. Now, before we look into data processing instructions, we need to know what kind of data types are supported in ARM. Obviously ARM is the 32 bit processor.

So, you will be having 32 bit word as a basic data type. But you can also look at each word as 8 bit, four 8 bit bytes and there can be operations on this bytes as well as 16 bit components as well. And you can configure the processor at power-up as little-endian or big-endian because that would define the definition of a value of a particular data whether it is being considered as a big-endian or small-endian. Now, data processing instructions obviously are concerned with manipulation of data within registers. We have MOVE instructions, arithmetic instructions; I have indicated multiply instructions because it implements multiplication and some variants of multiplication as well, then logical instructions, comparison instructions, this is what is standard in any instruction set of processors. And all these instructions can have a variant with a suffix S or not. When you add a suffix S to an instruction, it means that the condition flags will be set appropriately depending on the condition emerging out of the computation. If it is not, if you are not using a suffix, these condition flags will not be updated. These operands are typically 32 bit wide, can come from registers or specified as literal in the instruction itself.

That means I can have registers as well as immediate operands. The second operand can be sent via barrel shifter and 32 bit result is again placed in register only other cases that when you really do a 32 bit multiplication you generate 64 bit result which can go into multiple registers. So, this is an example of a move instruction. So move will obviously involve two operands, okay and I have shown you here, this is your destination register and N can be immediate value or source register. A simple example is r7 to r5; it means that I am moving content of r5 to r7. There is an interesting variant of move which is MVN the move negative. So, what you move? You move into Rd not of the 32 value from source, okay. So, this N can be a register, can also be an immediate value; so what you move in is not of that value into the destination.

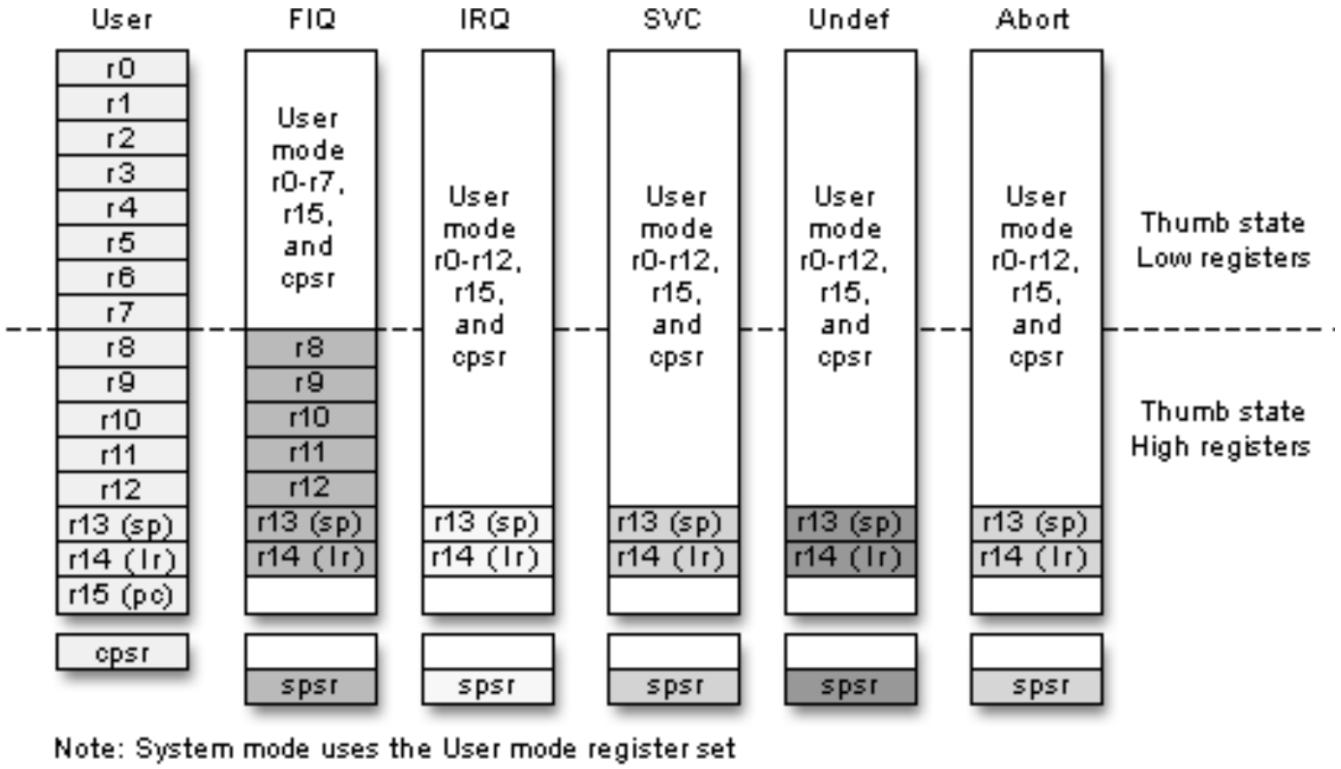

Fig 1.9Register organization

There are 37 total registers divided among seven different processor modes. Figure 1.9 shows the bank of registers visible in each mode. User mode, the only non-privileged mode, has the least number of total registers visible. It has no SPSR and limited access to the CPSR. FIQ and IRQ are the two interrupt modes of the CPU. Supervisor mode is the default mode of the processor on start up or reset. Undefined mode traps unknown or illegal instructions when they are passed through the pipeline. Abort mode traps illegal memory accesses as a result of fetching instructions or accessing data. Finally, system mode, which uses the user mode bank of registers, was introduced to provide an additional privileged mode when dealing with nested interrupts. Each additional mode offers unique registers that are available for use by exception handling routines. These additional registers are the minimum number of registers required to preserve the state of the processor, save the location in code, and switch between modes. FIQ mode, however, has an additional five banked registers to provide more flexibility and higher performance when handling critical interrupts. When the ARM core is in Thumb state, the registers banks are split into low and high register domains. The majority of instructions in Thumb state have a 3-bit register specifier. As a result, these instructions can only access the low registers in Thumb, R0 through R7. The high registers, R8 through R15, have more restricted use. Only a few instructions have access to these registers.

TDMI :T-D-M-I stands for
Thumb, which is a 16-bit instructionset extension to the 32-bit ARM architecture, referred as states of the processor.
"**D**" and "**I**" together comprise the on-chip debug facilities offered on all ARM cores.
These stand for the **D**ebug signals and Embedded**I**CE logic, respectively.
The M signifies the support for 64-bit results and an enhanced multiplier, resulting in higher performance. This multiplier is now standard on all ARMv4 architectures and above.

Thumb 16-bit Instructions

With growing code and data size, memory contributes to the system cost. The need to reduce memory cost leads to smaller code size and the use of narrower memory. Therefore ARM developed a modified instruction set to give market-leading code density for compiled standard C language. There is also the problem of performance loss due to using a narrow memory path, such as a 16-bit memory path with a 32-bit processor. The processor must take two memory access cycles to fetch an instruction or read and write data. To address this issue, ARM introduced another set of reduced 16-bit instructions labeled Thumb, based on the standard ARM 32-bit instruction set. For Thumb to be used, the processor must go through a change of state from ARM to Thumb in order to begin executing 16-bit code. This is because the default state of the core is ARM. Therefore, every application must have code at boot up that is written in ARM. If the application code is to be compiled entirely for Thumb, then the segment of ARM boot code must change the state of the processor. Once this is done, 16-bit instructions are fetched seamlessly into the pipeline without any result. It is important to note that the architecture remains the same. The instruction set is actually a reduced set of the ARM instruction set and only the instructions are 16-bit; everything else in the core still operates as 32-bit. An application code compiled in Thumb is 30% smaller on average than the same code compiled in ARM and normally 30% faster when using narrow 16-bit memory systems.

ARM7TDMI Processor Core

Architecture version 4T:

1 3-stage pipeline
2 Unified bus architecture
3 32-bit ARM ISA plus 16-bit Thumb extension
4 Forward compatible code
5 Embedded ICE on-chip debug
6 Hard Macrocell IP
7 Smallest Die Size: 0.53 mm2 on 0.18 μm process
8 Up to 110 MHz* on TSMC standard 0.18 μm
9 Industry leading 0.25 mW/MHz

The ARM7TDMI has a core based on the fourth version of the ARM architecture. This implementation uses a three stage pipeline - a standard fetch-decode-execute organization. It features a unified cache, as well as the Thumb extension permitting 32-bit and 16-bit operation. It is completely forward compatible, meaning that any code written for this core will be compatible with any new core releases, such as ARM9 or ARM10. This core also includes the on-chip debug extension discussed in the previous training module. The core is successful mainly because of the extremely small but high performance processor - slightly more than 70,000 transistors in all an with extremely low power consumption.

The ARM7TDMI family is popular with applications where small die size, high performance, and low power consumption help reduce system costs, especially when the system does not require cache. Applications include cellular phones, MP3 players, and mass storage.

ARM7TDMI applications

The standard ARM7TDMI processor core is a 'hard' macrocell, which is to say that it is delivered as a piece of physical layout, customized to the appropriate process technology. The ARM7TDMI-S is a synthesizable version of the ARM7TDMI, delivered as a high-level language module which can be synthesized using any suitable cell library in the target technology. It is therefore easier to port to a new process technology than is the hard macrocell.

The synthesis process supports a number of optional variations on the processor core functionality. These include: • omitting the EmbeddedlCE cell; • replacing the full 64-bit result multiplier with a smaller and simpler multiplier that supports only the ARM multiply instructions that produce a 32-bit result. Either of these options will result in a smaller synthesized macrocell with reduced functionality. The full version is 50% larger and 50% less power-efficient than the hard macrocell.

The ARM7TDMI processor core has found many applications in systems with simple memory configurations, usually including a few kilobytes of simple on-chip RAM. A typical example is a mobile telephone handset (where the same chip usually incorporates sophisticated digital signal processing hardware and associated

ARM9TDMI

The ARM9TDMI core takes the functionality of the ARM7TDMI up to a significantly higher performance level. Like the ARM7TDMI (and unlike the ARMS) it includes support for the Thumb instruction set and an EmbeddedlCE module for on-chip debug support. The performance improvement is achieved by adopting a 5-stage pipeline to increase the maximum clock rate and by using separate instruction and data memory ports to allow an improved CPI (Clocks Per Instruction - a measure of how much work a processor does in a clock cycle).

The rationale which leads from a requirement for higher performance to the need to increase the number of pipeline stages from three (as in the ARM7TDMI) to five, and to change the memory interface to employ separate instruction and data memories.

The 5-stage ARM9TDMI pipeline owes a lot to the StrongARM pipeline described in Section 12.3 on page 327. (The StrongARM has not been included as a core in this chapter as it has limited applicability as a stand-alone core.). The principal difference between the ARM9TDMI and the StrongARM core is that while StrongARM has a dedicated branch adder which operates in parallel with the register read stage, ARM9TDMI uses the main ALU for branch target calculations. This gives ARM9TDMI an additional clock cycle penalty for a taken branch, but results in a smaller and simpler core, and avoids a very critical timing path which is present in the StrongARM design. The StrongARM was designed for a particular process technology where this timing path could be carefully managed, whereas the ARM9TDMI is required to be readily portable to new processes where such a critical path could easily compromise the maximum usable clock rate.

where it is compared with the 3-stage ARM7TDMI pipeline. It how the major processing functions of the processor are redistributed across the additional pipeline stages in order to allow the clock frequency to be doubled (approximately) on the same process technology. Redistributing the execution functions (register read, shift, ALU, register write) is not all that is required to achieve this higher clock rate. The processor must also be able to access the

instruction memory in half the time that the ARM7TDMI takes and the instruction decode logic must also be restructured to allow the register read to take place concurrently with a substantial part of the decoding.

ARM7TDMI and ARM9TDMI pipeline comparison

The ARM7TDMI implements the Thumb instruction set by 'decompressing' Thumb instructions into ARM instructions using slack time in the ARM7 pipeline. The ARM9TDMI pipeline is much tighter and does not have sufficient slack time to allow Thumb instructions to be first translated into ARM instructions and then decoded; instead it has hardware to decode both ARM and Thumb instructions directly. The extra 'Memory' stage in the ARM9TDMI pipeline does not have any direct equivalent in the ARM7TDMI. Its function is performed by additional 'Execute' cycles that interrupt the pipeline flow. This interruption is an inevitable consequence of the single memory port used by the ARM7TDMI for both instruction and data accesses. During a data access an instruction fetch cannot take place. The ARM9TDMI avoids this pipeline interruption through the provision of separate instruction and data memories.

The ARM9TDMI has a coprocessor interface which allows on-chip coprocessors for floating-point, digital signal processing or other special-purpose hardware acceleration requirements to be supported. (At the clock speeds it supports there is little possibility of off-chip coprocessors being useful.)

The EmbeddedlCE functionality in the ARM9TDMI core gives the same system-level debug features as that on the ARM7TDMI core, with the following additional features:

• Hardware single-stepping is supported.

• Breakpoints can be set on exceptions in addition to the address/data/control conditions supported by ARM7TDMI.

ARM10TDMI

Increased clock rate

The ARM10TDMI is the current high-end ARM processor core and is still under development at the time of writing. Just as the ARM9TDMI delivers approximately twice the performance of the ARM7TDMI on the same process, the ARM10TDMI is positioned to operate at twice the performance of the ARM9TDMI. It is intended to deliver 400 Dhrystone 2.1 MIPS at 300 MHz on 0.25 urn CMOS technology. In order to achieve this level of performance, starting from the ARM9TDMI, two approaches have been combined:

1. The maximum clock rate has been increased.
2. The CPI (average number of Clocks per Instruction) has been reduced.

Since the ARM9TDMI pipeline is already fairly optimal, how can these improvements are achieved without resorting to a very complex organization, such as superscalar execution, which would compromise the low power and small core size that are the hallmarks of an ARM core?

The maximum clock rate that an ARM core can support is determined by the slowest logic path in any of the pipeline stages.

Resources

Sr. No.	Website used
1.	https://slideplayer.com/slide/8290583/
2.	https://www.arduino.cc/en/Guide/HomePage

Sr. No.	Website used
3.	https://en.wikipedia.org/wiki/ARM_architecture
4.	http://www.microdigitaled.com

Sample Question:

Sr. No.	Question	Level	Marks
1.	Main processor chip in computers is A. ASIC B. ASSP **C. CPU** D. CPLD	R	
2.	The CISC stands for _____ A. Computer Instruction Set Compliment B. Complete Instruction Set Compliment C. Computer Indexed Set Components **D. Complex Instruction set computer**	R	
3.	What does GPIO stand for? A. General Purpose Inner Outer Propeller **B. General Purpose Input Output Pins** C. General Purpose Interested Old People D. General Purpose Input Output Processor	R	
4.	What does IDE stand for? A. In Deep Environment **B. Integrated Development Environment** C. Internal Deep Escape D. IDE	R	
5.	A program written with the IDE for Arduino is called _____	R	

Sr. No.	Question	Level	Marks
	A. IDE source **B. Sketch** C. Cryptography D. Source code		
6.	Arduino IDE consists of 2 functions. What are they? A. Build() and loop() B. Setup() and build() **C. Setup() and loop()** D. Loop() and build and setup()	R	
7.			
8.	How many digital pins are there on the UNO board? **A. 14** B. 12 C. 16 D. 20		
9.	Most of processors designed by ARM are A. 16 bit **B. 32 bit** C. 64 bit D. 8 bit	R	
10.	The computer architecture aimed at reducing the time of execution of instructions is _____ A. CISC	U	

Sr. No.	Question	Level	Marks
	B. RISC C. ISA D. ANNA		
11.	How is the nature of instruction size in CISC processors? **A.** Fixed **B. Variable** **C.** Both a and b **D.** None of the above	U	
12.	If the three stages of execution in pipelining are overlapped, how would be the speed of execution? **A. Higher** B. Moderate C. Lower D. Unpredictable	U	
13.	What is/are the configuration status of control unit in RISC Processors? **A. Hardwired** **B.** Micro programmed **C.** Both a and b **D.** None of the above	U	
14.	A function is a series of programming statements that can be called by name. Which command is called once when the program starts: A. loop() **B. setup()** C. (output) D. (input)	U	
15.	What does p refer to in ATmega328p? A. Production	U	

Sr. No.	Question	Level	Marks
	B. Pico-Power C. Power-Pico D. Programmable on chip		
16.	The throughput of a super scalar processor is _____ A. less than 1 B. 1 **C. More than 1** D. Not Known	U	
17.	In super-scalar mode, all the similar instructions are grouped and executed together. **A. True** B. False	U	
18.	Each stage in pipelining should be completed within ____ cycle. **A. 1** B. 2 C. 3 D. 4	U	
19.	In pipelining the task which requires the least time is performed first. A. True **B. False**	U	
20.	It starts with a /* and continues until a */ what does this do? A. Loads a sketch **B. Makes comments** C. Compiles quicker	A	

Sr. No.	Question	Level	Marks
	D. Makes stars appear		
21.	A function is a series of programming statements that can be called by name. Which command is called once when the program starts: A. loop() **B. setup()** C. (input) D. (output)	A	
22.	What is the default bootloader of the Arduino UNO? **A. Optibootloader** B. AIR-boot C. Bare box D. GAG	A	
23.	What is the microcontroller used in Arduino UNO? **A. ATmega328p** B. ATmega2560 C. ATmega32114 D. AT91SAM3x8E	A	
24.			
25.			
26.			

Biblography

1
2
3
4