Outline

- **ARM EMBEDDED SYSTEMS**
  - The RISC Design Philosophy
  - The ARM Design Philosophy
  - Embedded System Hardware
  - Embedded System Software

- **ARM PROCESSOR FUNDAMENTALS**
  - Registers
  - Current Program Status Register
  - Pipeline
  - Exceptions
  - Interrupts and Vector Table
  - Core Extensions
  - Architecture Revisions
  - ARM Processor Families

- **LPC2148 MICROCONTROLLER**
  - Architectural features

EMBEDDED SYSTEM HARDWARE

- Embedded systems can control many different devices, from small sensors found on a production line, to the real-time control systems used on a NASA space probe. All these devices use a combination of software and hardware components.

- The following Figure shows a typical embedded device based on an ARM core. Each box represents a feature or function. The lines connecting the boxes are the buses carrying data.

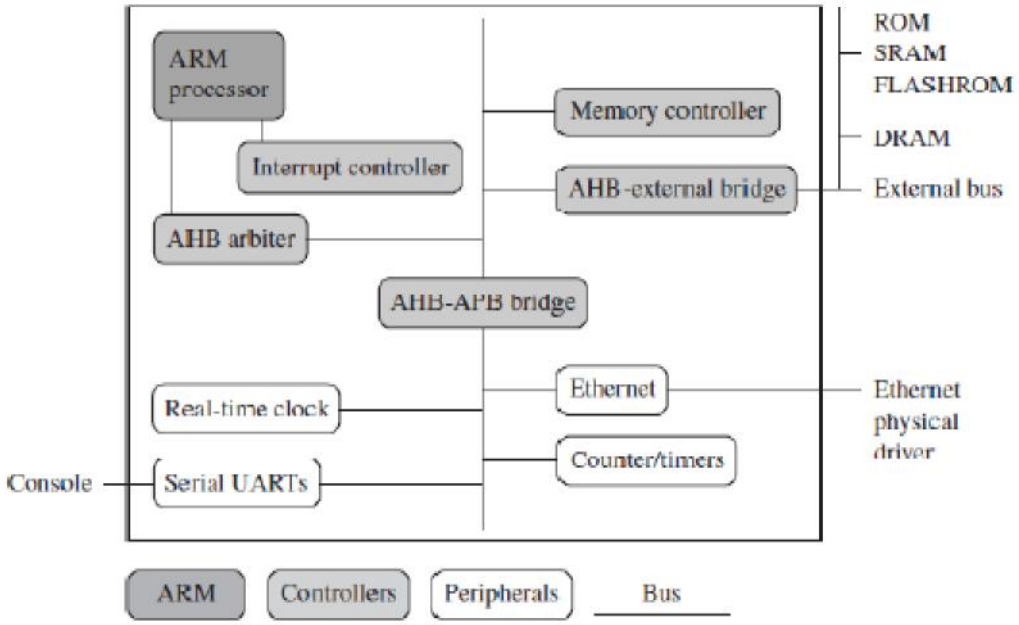

Figure: An ARM-based Embedded Device, a Microcontroller

We can separate the device into ***four main hardware components***:

1. The *ARM processor* controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics. An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) plus the surrounding components (memory and cache) that interface it with a bus.
2. *Controllers* coordinate important functional blocks of the system. Two commonly found controllers are interrupt and memory controllers.
3. The *peripherals* provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device.
4. A *bus* is used to communicate between different parts of the device.

ARM Bus Technology:

- Embedded devices use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core.

- There are ***two different classes of devices*** attached to the bus:
  1. The *ARM processor core* is a bus master—a logical device capable of initiating a data transfer with another device across the same bus.
  2. *Peripherals* tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.

- A bus has ***two architecture*** levels: A *physical level*—covers the electrical characteristics and bus width (16, 32, or 64 bits). The *protocol*—the logical rules that govern the communication between the processor and a peripheral.

AMBA Bus Protocol:

- The *Advanced Microcontroller Bus Architecture (AMBA)* was introduced in 1996 and has been widely adopted as the on-chip bus architecture used for ARM processors.
- The first AMBA buses introduced were the *ARM System Bus (ASB)* and the *ARM Peripheral Bus (APB)*. Later ARM introduced another bus design, called the *ARM High Performance Bus* (AHB).
- Using AMBA, peripheral designers can reuse the same design on multiple projects. A peripheral can simply be bolted onto the on-chip bus without having to redesign an interface for each different processor architecture. This plug-and-play interface for hardware developers improves availability and time to market.
- AHB provides higher data throughput than ASB because it is based on a centralized multiplexed bus scheme rather than the ASB bidirectional bus design. This change allows the AHB bus to run at higher clock speeds.

- ARM has introduced ***two variations*** on the AHB bus: *Multi-layer AHB* and *AHB-Lite*.
  - The Multi-layer AHB bus allows multiple active bus masters.
  - AHB-Lite is a subset of the AHB bus and it is limited to a single bus master.

- The example device shown in the Figure has three buses:
  - an *AHB bus* for the high- performance peripherals

- an *APB bus* for the slower peripherals
- a third *bus for external peripherals*, proprietary to this device.

Memory

An embedded system has to have some form of memory to store and execute code. You have to compare price, performance, and power consumption when deciding upon specific memory characteristics such as **hierarchy, width, and type**.

Hierarchy

All computer systems have memory arranged in some form of hierarchy. The following Figure shows the memory trade-offs: the fastest memory cache is physically located nearer the ARM processor core and the slowest secondary memory is set further away. Generally the closer memory is to the processor core, the more it costs and the smaller its capacity.

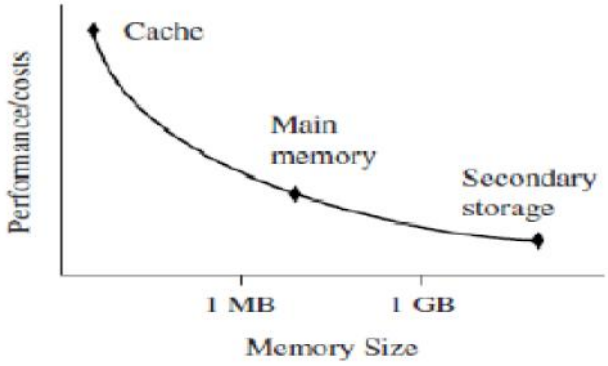

Figure: Memory Storage Trade-offs

The *cache* is placed between main memory and the core. It is used to speed up data transfer between the processor and main memory. A cache provides an overall increase in performance but with a loss of predictable execution time. Although the cache increases the general performance of the system, it does not help real-time system response.

- The *main memory* is large—around 256 KB to 256 MB (or even greater), depending on the application—and is generally stored in separate chips. Load and store instructions access the main memory unless the values have been stored in the cache for fast access.

- *Secondary storage* is the largest and slowest form of memory. Hard disk drives and CD-ROM drives are examples of secondary storage.

Types:

There are many *different types of memory*:

- *Read-only memory (ROM)* is the least flexible of all memory types because it contains an image that is permanently set at production time and cannot be reprogrammed.
  - ROMs are used in high-volume devices that require no updates or corrections. Many devices also use a ROM to hold boot code.

- *Flash ROM* can be written to as well as read, but it is slow to write so you shouldn't use it for holding dynamic data.
  - Its main use is for holding the device firmware or storing long-term data that needs to be preserved after power is off. The erasing and writing of flash ROM are completely software controlled with no additional hardware circuitry required, which reduces the manufacturing costs.

- *Dynamic random access memory (DRAM)* is the most commonly used RAM for devices. It has the lowest cost per megabyte compared with other types of RAM. DRAM is dynamic—it needs to have its storage cells refreshed and given a new electronic charge every few milliseconds, so you need to set up a DRAM controller before using the memory.

- *Static random access memory (SRAM)* is faster than the more traditional DRAM, but requires more silicon area. SRAM is static—the RAM does not require refreshing. The access time for SRAM is considerably shorter than the equivalent DRAM because SRAM does not require a pause between data accesses. But cost of SRAM is high.

- *Synchronous dynamic random access memory (SDRAM)* is one of many subcategories of DRAM. It can run at much higher clock speeds than conventional memory.

Peripherals

Embedded systems that interact with the outside world need some form of peripheral device. A ***peripheral device*** performs input and output functions for the chip by connecting to other devices or sensors that are off-chip.

- Each peripheral device usually performs a single function and may reside on-chip.

- Peripherals range from a simple serial communication device to a more complex 802.11 wireless device.
- All ARM peripherals are *memory mapped*
- *Controllers* are specialized peripherals that implement higher levels of functionality within an embedded system.
  - Two important types of controllers are memory controllers and interrupt controllers.

Memory Controllers: Memory controllers connect different types of memory to the processor bus.

- On power-up a memory controller is configured in hardware to allow certain memory devices to be active. These memory devices allow the initialization code to be executed.

Interrupt Controllers: When a peripheral or device requires attention, it raises an *interrupt* to the processor. An *interrupt controller* provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time by setting the appropriate bits in the interrupt controller registers.

There are *two types of interrupt controller* available for the ARM processor:

1. The *standard interrupt controller* sends an interrupt signal to the processor core when an external device requests servicing. It can be programmed to ignore or mask an individual device or set of devices.
   - The *interrupt handler* determines which device requires servicing by reading a device bitmap register in the interrupt controller.

2. The *vector interrupt controller (VIC)* is more powerful than the standard interrupt controller, because it prioritizes interrupts and simplifies the determination of which device caused the interrupt.

EMBEDDED SYSTEM SOFTWARE

An embedded system needs software to drive it. The following Figure shows four typical software components required to control an embedded device.

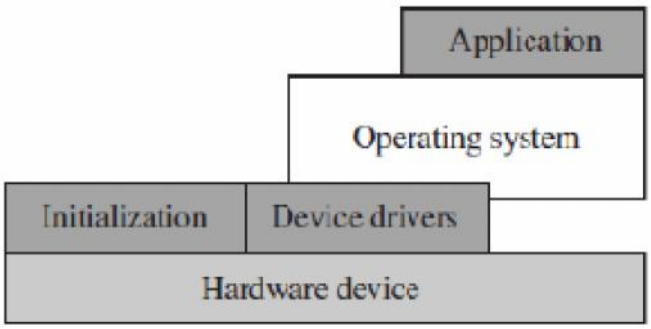

Figure: Software Abstraction Layers Executing on Hardware

- The *initialization code* is the first code executed on the board and is specific to a particular target or group of targets. It sets up the minimum parts of the board before handing control over to the operating system.
- The *operating system* provides an infrastructure to control applications and manage hardware system resources.
- The *device drivers* provide a consistent software interface to the peripherals on the hardware device.
- An *application* performs one of the tasks required for a device.
  - For example, a mobile phone might have a diary application.

  There may be multiple applications running on the same device, controlled by the operating system.

Initialization (Boot) Code:

- Initialization code (or boot code) takes the processor from the reset state to a state where the operating system can run. It usually configures the memory controller and processor caches and initializes some devices.
- The initialization code handles a number of administrative tasks prior to handing control over to an operating system image.
  - We can group these different tasks into *three phases*: **initial hardware configuration, diagnostics, and booting**.

1. ***Initial hardware configuration*** involves setting up the target platform, so that it can boot an image. The target platform comes up in a standard configuration; but, this configuration normally requires modification to satisfy the requirements of the booted image.

- For example, the memory system normally requires reorganization of the memory map, as shown in the following Example.

Example: *Initializing or organizing memory is an important part of the initialization code, because many operating systems expect a known memory layout before they can start.*

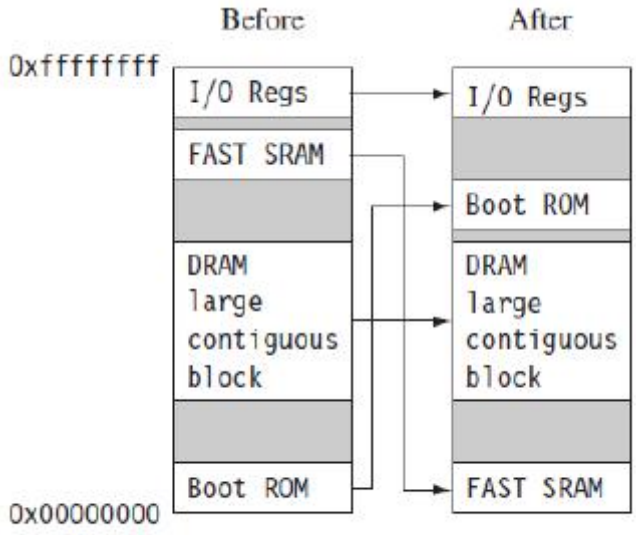

Figure: Memory Remapping

The above Figure shows memory before and after reorganization. It is common for ARM-based embedded systems to provide for memory remapping because it allows the system to start the initialization code from ROM at power-up. The initialization code then redefines or remaps the memory map to place RAM at address 0x00000000—an important step because then the exception vector table can be in RAM and thus can be reprogrammed.

2. ***Diagnostics*** are often embedded in the initialization code. Diagnostic code tests the system by exercising the hardware target to check if the target is in working order. It also tracks down standard system-related issues. The primary purpose of diagnostic code is fault identification and isolation.

3. ***Booting*** involves loading an image and handing control over to that image. The boot process itself can be complicated if the system must boot different operating systems or different versions of the same operating system.
   - Booting an image is the final phase, but first you must load the image. Loading an image involves anything from copying an

entire program including code and data into RAM, to just copying a data area containing volatile variables into RAM. Once booted, the system hands over control by modifying the program counter to point into the start of the image.

Operating System

- The initialization process prepares the hardware for an operating system to take control. An operating system organizes the system resources: the peripherals, memory, and processing time.

- ARM processors support over 50 operating systems. We can divide operating systems into *two main categories*: **real-time operating systems (RTOSs) and platform operating systems**.

1. ***RTOSs*** provide guaranteed response times to events. Different operating systems have different amounts of control over the system response time.
   - A *hard real-time* application requires a guaranteed response to work at all.
   - In contrast, a *soft real-time* application requires a good response time, but the performance degrades more gracefully if the response time overruns.

2. ***Platform operating systems*** require a memory management unit to manage large, non-real-time applications and tend to have secondary storage.
   - The Linux operating system is a typical example of a platform operating system.

Applications

- The operating system schedules *applications*—code dedicated to handle a particular task. An application implements a processing task; the operating system controls the environment.
  - An embedded system can have one active application or several applications running simultaneously.
- ARM processors are found in numerous market segments, including networking, auto-motive, mobile and consumer devices, mass storage, and imaging.
- ARM processor is found in networking applications.
- The mobile device segment is the largest application area for ARM processors, because of mobile phones.

- ARM processors are also found in mass storage devices such as hard drives and imaging products such as inkjet printers—applications that are cost sensitive and high volume.

In contrast, ARM processors are not found in applications that require leading-edge high performance. Because these applications tend to be low volume and high cost, ARM has decided not to focus designs on these types of applications.

CORE EXTENSIONS

- *Core extensions* are the standard hardware components placed next to the ARM core.

- They improve performance, manage resources, and provide extra functionality and are designed to provide flexibility in handling particular applications.

Each ARM family has different extensions available. There are *three hardware extensions*: **cache and tightly coupled memory, memory management, and the coprocessor interface**.

Cache and Tightly Coupled Memory:

- The cache is a block of fast memory placed between main memory and the core. It allows for more efficient fetches from some memory types. With a cache the processor core can run for the majority of the time without having to wait for data from slow external memory.

- Most ARM-based embedded systems use a single-level cache internal to the processor.

- ARM has *two forms of cache*. The first is found attached to the Von Neumann–style cores. It combines both data and instruction into a single unified cache, as shown in the following Figure.

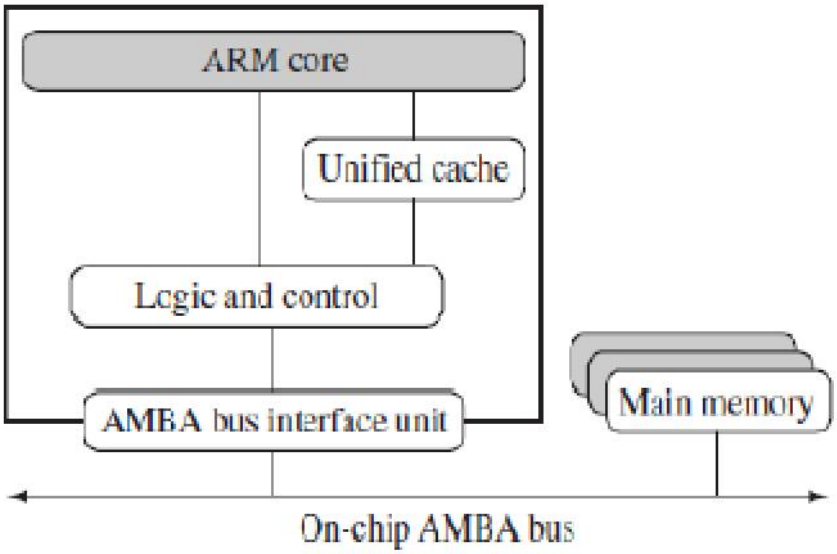

Figure: Von Neumann Architecture with Cache

- The second form, attached to the Harvard-style cores, has separate caches for data and instruction, as shown in the following Figure.

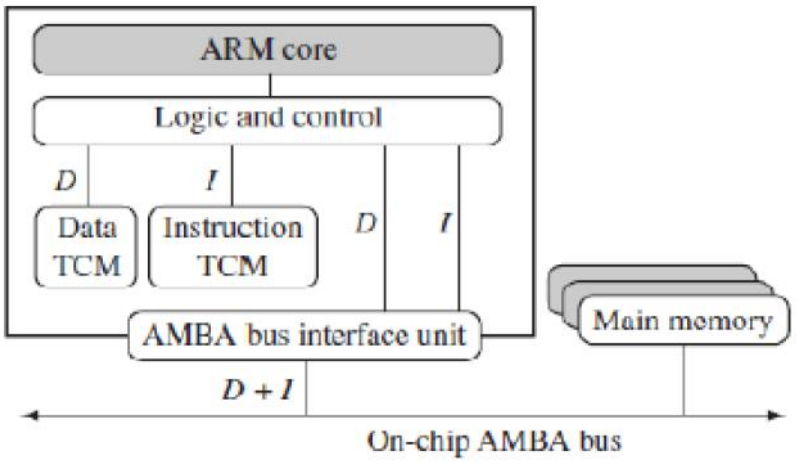

Figure: Harvard Architecture with TCMs

- A cache provides an overall increase in performance, but at the expense of predictable execution. But the real-time systems require the code execution to be deterministic— the time taken for loading and storing instructions or data must be predictable.

- This is achieved using a form of memory called *tightly coupled memory (TCM)*. TCM is fast SRAM located close to the core and guarantees the clock cycles required to fetch instructions or data.
- TCMs appear as memory in the address map and can be accessed as fast memory.

By combining both technologies, ARM processors can have both improved performance and predictable real-time response. The following Figure shows an example core with a combination of caches and TCMs.

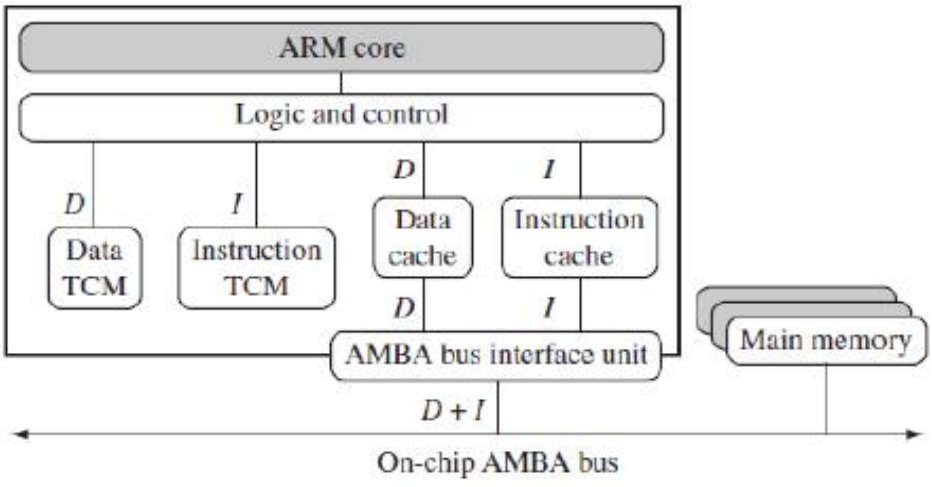

Figure: Harvard Architecture with Caches and TCMs

Memory Management:

- Embedded systems often use multiple memory devices. It is usually necessary to have a method to organize these devices and protect the system from applications trying to make inappropriate accesses to hardware. This is achieved with the assistance of memory management hardware.
- ARM cores have *three different types of memory management hardware:*
  - no extensions providing no protection
  - a memory protection unit (MPU) providing limited protection
  - a memory management unit (MMU) providing full protection

- ***Non protected memory*** is fixed and provides very little flexibility. It is normally used for small, simple embedded systems that require no protection from rogue applications.

- ***MPU***s employ a simple system that uses a limited number of memory regions. These regions are controlled with a set of special coprocessor registers, and each region is defined with specific access permissions. This type of memory management is used for systems that require memory protection but don't have a complex memory map.

- ***MMU***s are the most comprehensive memory management hardware available on the ARM. The MMU uses a set of translation tables to provide fine-grained control over memory. These tables are stored in main memory and provide a virtual-to-physical address map as well as access permissions. MMUs are designed for more sophisticated platform operating systems that support multitasking.

Coprocessors:

- Coprocessors can be attached to the ARM processor. A coprocessor extends the processing features of a core by extending the instruction set or by providing configuration registers. More than one coprocessor can be added to the ARM core via the coprocessor interface.
- The coprocessor can be accessed through a group of dedicated ARM instructions that provide a load-store type interface.
  - For example, coprocessor 15: The ARM processor uses coprocessor 15 registers to control the cache, TCMs, and memory management.

- The coprocessor can also extend the instruction set by providing a specialized group of new instructions.
  - For example, there are a set of specialized instructions that can be added to the standard ARM instruction set to process vector floating-point (VFP) operations.
- These new instructions are processed in the decode stage of the ARM pipeline.
  - If the decode stage sees a coprocessor instruction, then it offers it to the relevant coprocessor.
  - If the coprocessor is not present or doesn't recognize the instruction, then the ARM takes an undefined instruction exception, which allows you to emulate the behavior of the coprocessor in software.

Architecture Revisions

The nomenclature identifies individual processors and provides basic information about the feature set.

Nomenclature:

ARM uses the nomenclature shown in below figure to describe the processor implementations.

ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}

x—family
y—memory management/protection unit
z—cache
T—Thumb 16-bit decoder
D—JTAG debug
M—fast multiplier
I—EmbeddedICE macrocell
E—enhanced instructions (assumes TDMI)
J—Jazelle
F—vector floating-point unit
S—synthesizible version

Figure: ARM Nomenclature

- The letters and numbers after the word "ARM" indicate the features a processor may have. In the future the number and letter combinations may change as more features are added. Note the nomenclature does not include the architecture revision information.

- There are a few additional points to make about the ARM nomenclature:

  - All ARM cores after the ARM7TDMI include the *TDMI* features even though they may not include those letters after the "ARM" label.

  - The processor *family* is a group of processor implementations that share the same hardware characteristics. For example, the ARM7TDMI, ARM740T, and ARM720T all share the same family characteristics and belong to the ARM7 family.

  - *JTAG* is described by IEEE 1149.1 Standard Test Access Port and boundary scan architecture. It is a serial protocol used by ARM to send and receive debug information between the processor core and test equipment.

  - *EmbeddedICE macrocell* is the debug hardware built into the processor that allows breakpoints and watch points to be set.

  - *Synthesizable* means that the processor core is supplied as source code that can be compiled into a form easily used by EDA tools.

Below table shows the significant architecture enhancements from the original architecture version 1 to the current version 6 architecture. One of the most significant changes to the ISA was the introduction of the Thumb instruction set inARMv4T (the ARM7TDMI processor).

Revision	Example core implementation	ISA enhancement
ARMv1	ARM1	First ARM processor 26-bit addressing
ARMv2	ARM2	32-bit multiplier 32-bit coprocessor support
ARMv2a	ARM3	On-chip cache Atomic swap instruction Coprocessor 15 for cache management
ARMv3	ARM6 and ARM7DI	32-bit addressing Separate *cpsr* and *spsr* New modes—*undefined instruction* and *abort* MMU support—virtual memory
ARMv3M	ARM7M	Signed and unsigned long multiply instructions
ARMv4	StrongARM	Load-store instructions for signed and unsigned halfwords/bytes New mode—*system* Reserve SWI space for architecturally defined operations 26-bit addressing mode no longer supported
ARMv4T	ARM7TDMI and ARM9T	Thumb
ARMv5TE	ARM9E and ARM10E	Superset of the ARMv4T Extra instructions added for changing state between ARM and Thumb Enhanced multiply instructions Extra DSP-type instructions Faster multiply accumulate
ARMv5TEJ	ARM7EJ and ARM926EJ	Java acceleration
ARMv6	ARM11	Improved multiprocessor instructions Unaligned and mixed endian data handling New multimedia instructions

ARM Processor Families:

- ARM has designed a number of processors that are grouped into different families according to the core they use.
- The families are based on the ARM7, ARM9, ARM10, and ARM11 cores.
- The postfix numbers 7, 9, 10, and 11 indicate different core designs. The ascending number equates to an increase in performance and sophistication.
- Below table shows a rough comparison of attributes between the ARM7, ARM9, ARM10, and ARM11 cores.

Table: ARM family attributes comparison.

Attribute	ARM family			
	ARM7	**ARM9**	**ARM10**	**ARM11**
Pipeline depth	Three-stage	Five-stage	Six-stage	Eight-stage
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8x32	8x32	16x32	16x32

LPC2148 Microcontroller

The LPC2148 microcontroller is designed by Philips (NXP Semiconductor) with several in-built features & peripherals. Due to these reasons, it will make more reliable as well as the efficient option for an application developer.

Features of LPC2148

The main features of LPC2148 include the following.

- It is a 32-bit ARM7TDMI-S core (ARM7 family based microcontroller) available in a 64-pin package (Quad Flat package).
- It has 512 KB on-chip Flash ROM with In-System Programming (ISP) and In-Application Programming (IAP) via on-chip boot loader software, 32 KB SRAM.
- It has 1 PSR (Program Status Register) and 16 GPRs.
- It has 44 GPIO pins and 1 GPO pin and operating voltage of these I/O pins is 5V.
- It has 2 10-bit ADCs with 7 channels each for a total of 14 channels.
- It has 2 32-bit timers/counters with PWM unit. Each timer has four 32 bit capture channels which take the snapshot of timer value during the transition of any input signal.
- It contains the watch dog timer whose main purpose is to reset the microcontroller with in specific amount of time during erroneous state.
- It has Vectored Interrupt Controller.
- It has Low power Real-Time Clock (RTC) which is designed to set the counters for the measuring the whole time when the controller is in operating mode or idle mode.
- It has multiple serial interfaces: 2 UARTs (Universal Asynchronous Receiver Transmitter), 2 fast I2C (Inter-Integrated Circuit) buses with 400 kbps (kilo bits per second) speed, 1 SPI (Serial Peripheral Interface), 1 SSP (Serial Synchronous Port Controller).
- It contains an on-chip integrated oscillator which operates with an external crystal whose range is in between 1 MHz to 25 MHz.
- It takes 400 milliseconds time for erasing the data in full chip and 1 millisecond time for 256 bytes of programming.
- It has Embedded Trace interfaces and Embedded ICE RT offers real-time debugging with high-speed tracing of instruction execution and on-chip Real Monitor software.
- It offers a changeable output with one 10-bit DAC.
- It has power-saving modes like idle & power down.