

# ARM7 Processor Architecture

# ARM7 - Introduction

## Advanced (Acron) RISC Machines Processor Cores

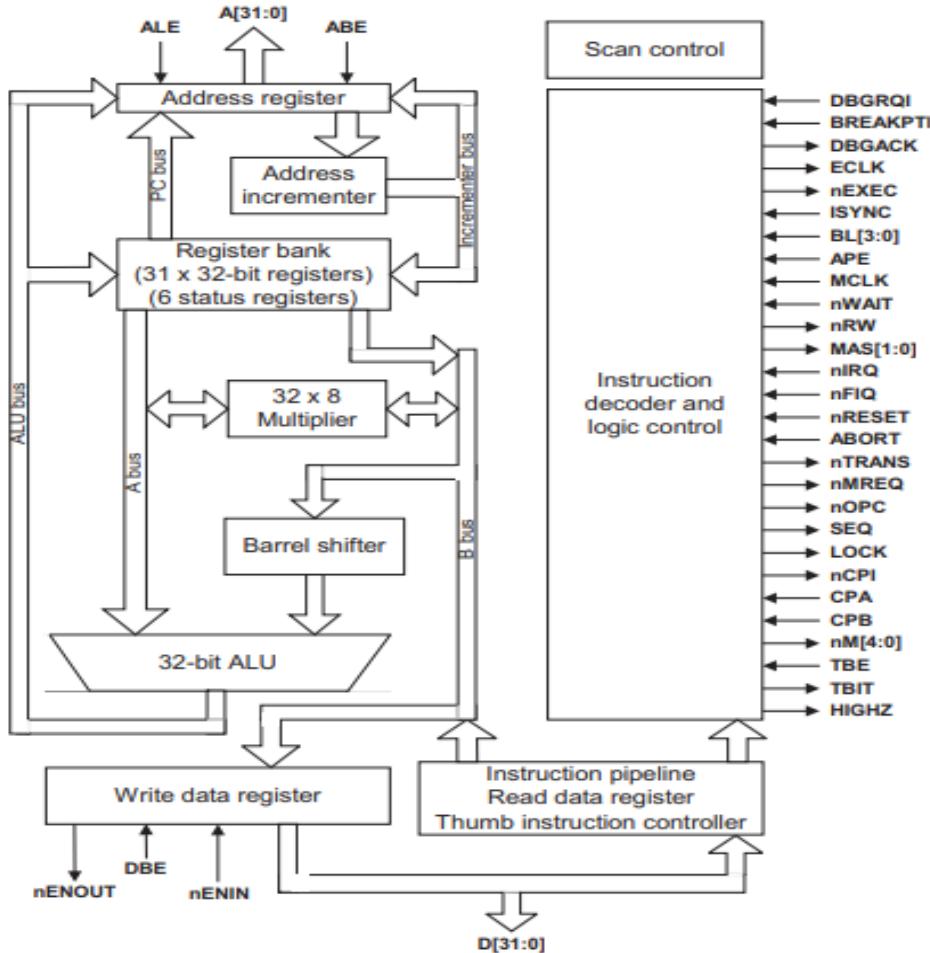
- ARM is a British semiconductor and software design company based in Cambridge, England,
- Originally known as Acorn RISC machine
- Now its **Advanced RISC Machines** is a family of reduced instruction set
- The primary business ARM is to design of 32 bit and 64 bit ARM High Performance RISC architecture processors (CPUs).
- ARM is the industry's leading provider of 32/64-bit embedded RISC microprocessor solutions
- Arm Ltd. develops the architecture and licenses it to other companies,
- who wants design their own Microcontrollers or systems-on-chips (SoC) and systems-on-modules (SoM)
- Ex; **ATMEL, Cirrus Logic, Freescale Semiconductor, NXP(Founded By Philips), OKI, Samsung, Sharp, STMicroelectronics, Texas Instruments**

- Due to their low costs, minimal power consumption, and lower heat generation
- ARM processors are desirable for light, portable, battery-powered devices
  - Including smart phones, laptops and tablet computers, as well as other embedded systems.
  - ARM processors are also used for desktops and servers, including the world's fastest supercomputer.

## Features of ARM7

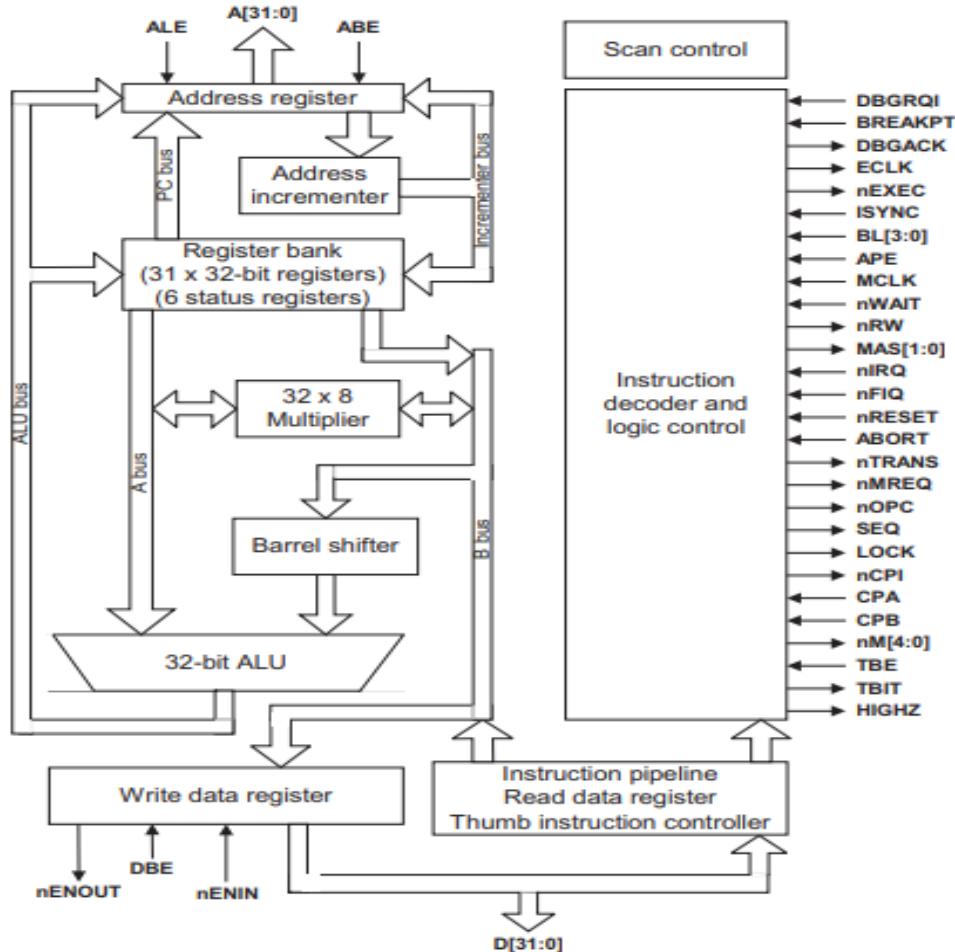
- 32-bit Advanced **RISC(Adv.) Processor** 32-bit ALU architecture (ARM v4T)
- Operates upto **60 to 100Mhz**
- **32-bit ARM instruction** set for maximum performance and flexibility
- **16-bit Thumb instruction** set for increased code density
- Unified bus interface, **32-bit data bus** carries both instructions and data
- **Three-stage pipeline**
  1. Fetch
  2. Decode
  3. Execute
- **JTAG interface** debug unit
- **32-Bit Memory Addressing Range**
- **USER mode, FIQ mode, IRQ mode, SVC mode**
- **UNDEFINED mode, ABORT mode, System Mode**

# ARM7 Processor Architecture

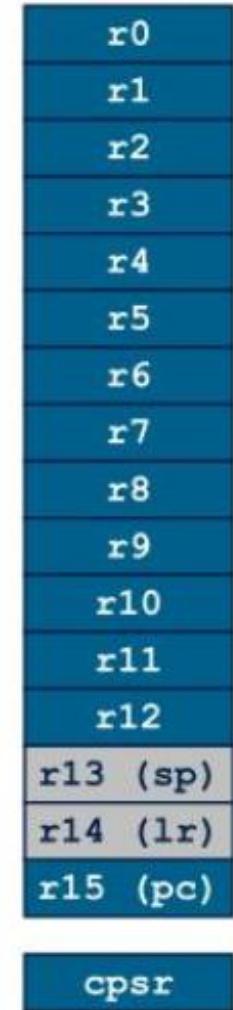
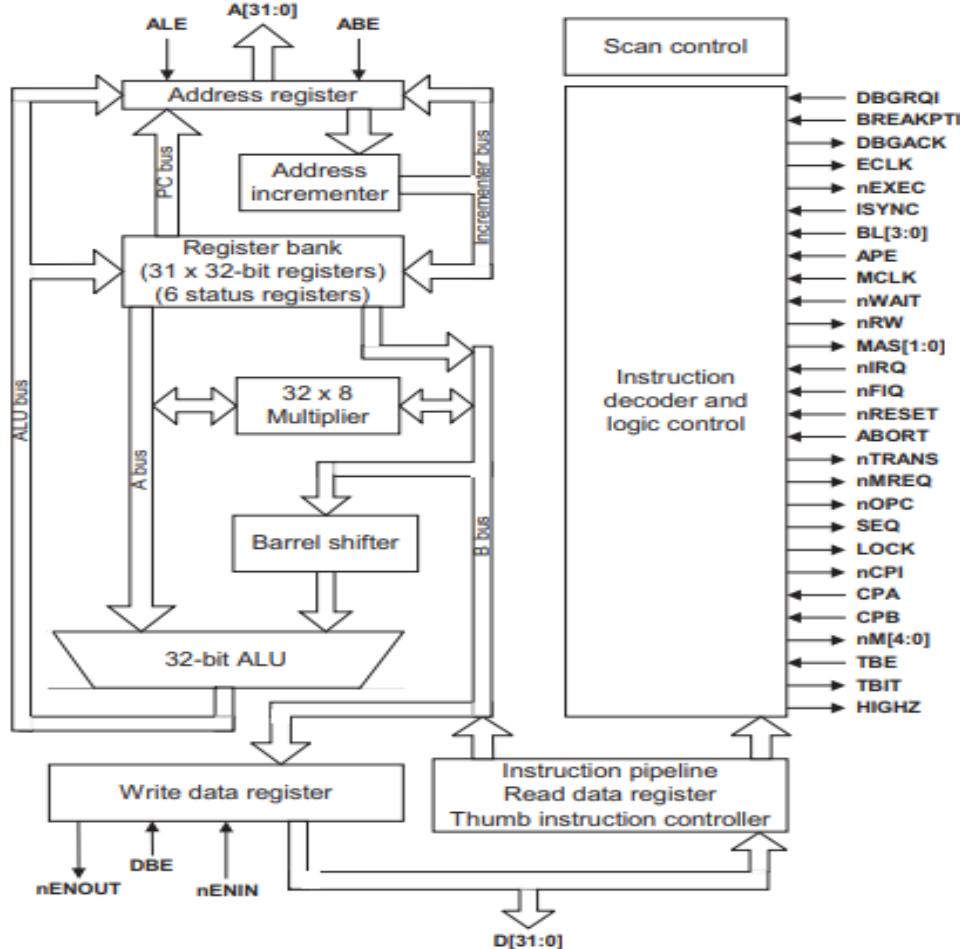


- 32 Bit ALU
- 32 Bit PC- Program Counter
- 32 bit MAC Unit
- 32 Bit Barrel Shifter
- 32 Bit Address Bus
- 32 Bit Data Bus
- Signed Extension
- Instruction Decoder

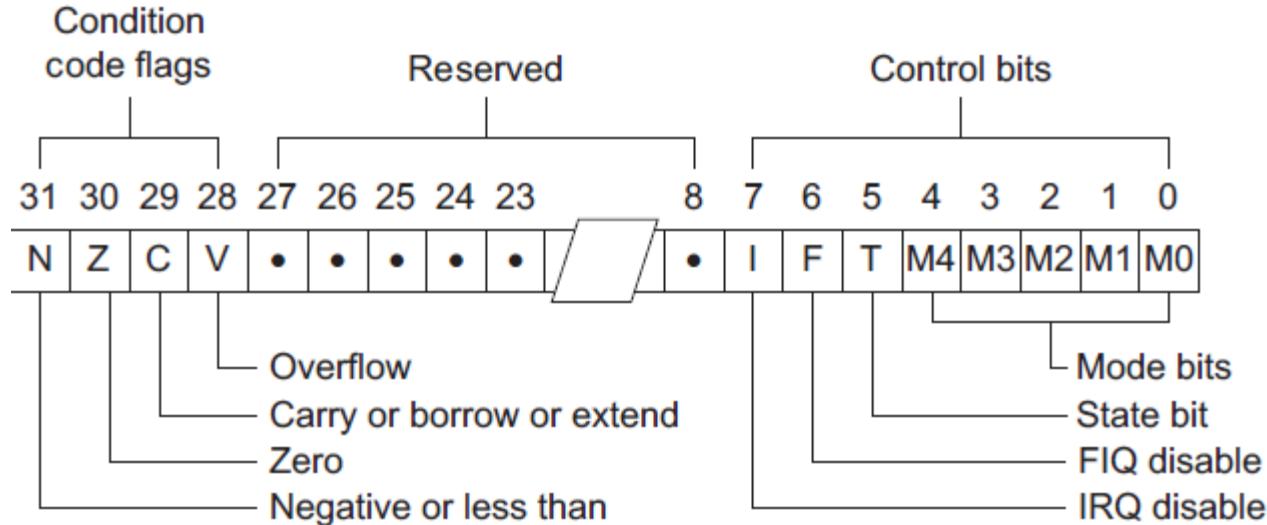
# ARM Processor Architecture



# ARM Processor Architecture



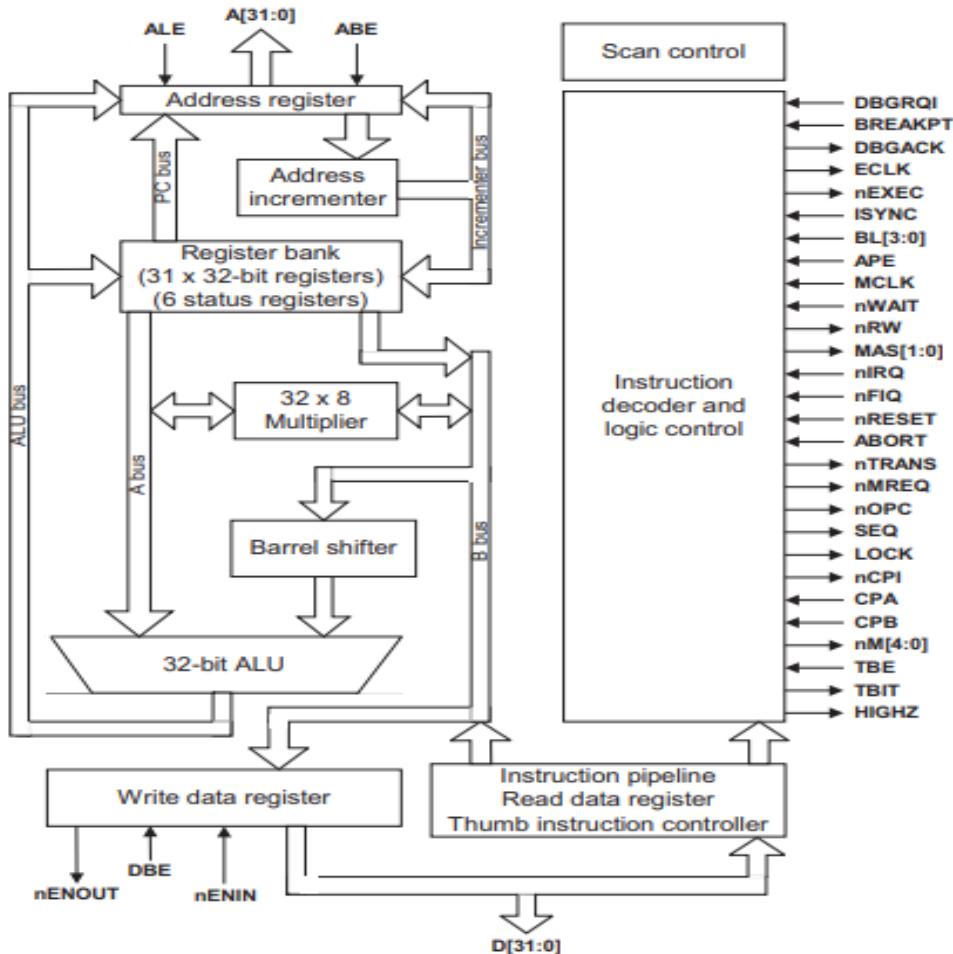
# The Current Program Status Registers-(CPSR)



## M[4:0] Mode

10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

# ARM Processor Architecture



## Inline barrel shifter

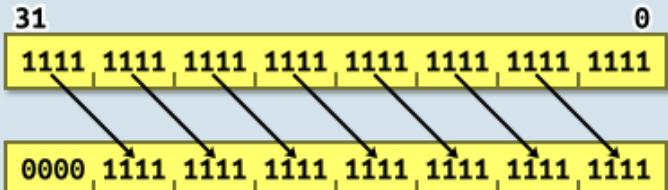
The ARM arithmetic logic unit has a 32-bit barrel shifter that is capable of shift and rotate operations.

The second operand of the instruction specifies the no. of bits of shift or rotation

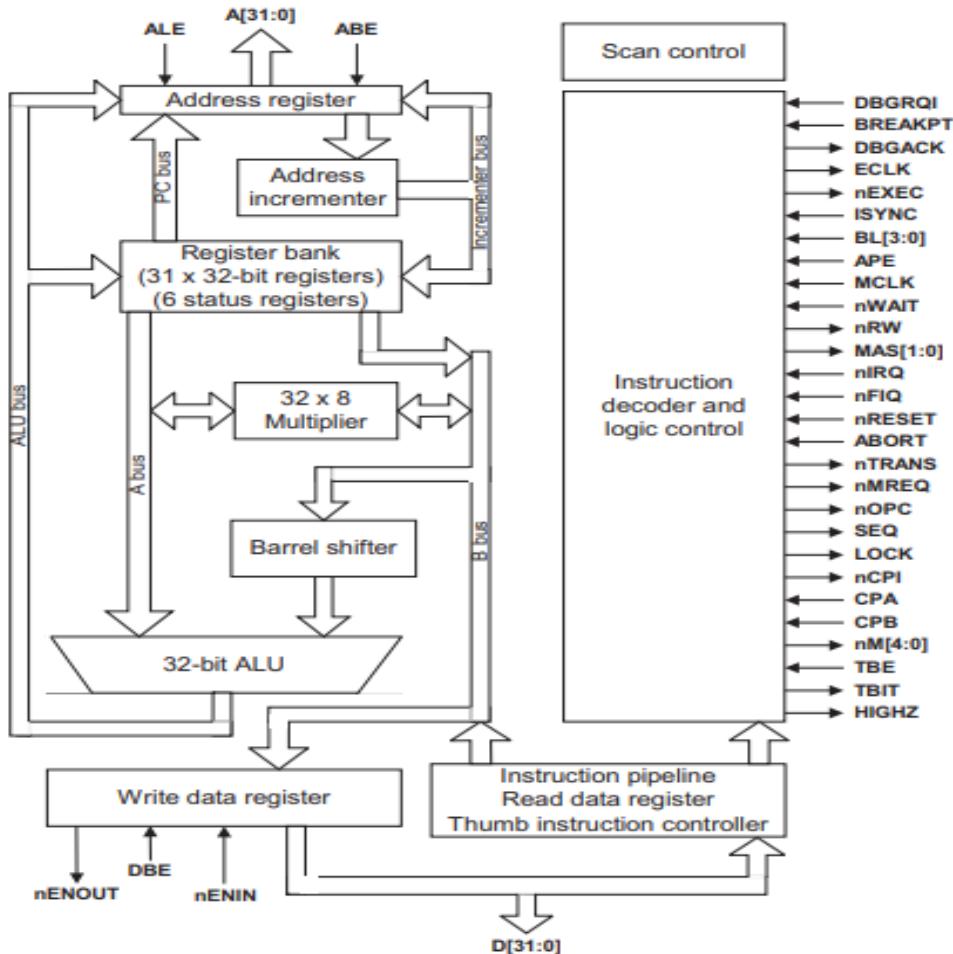
Ex: LSR R5,4

### LSR – Logical Shift Right

Example: Logical Shift Right by 4.



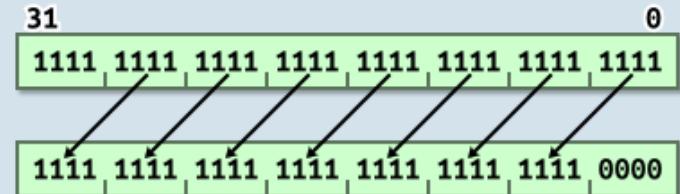
# ARM Processor Architecture



Ex: LSL R5,4

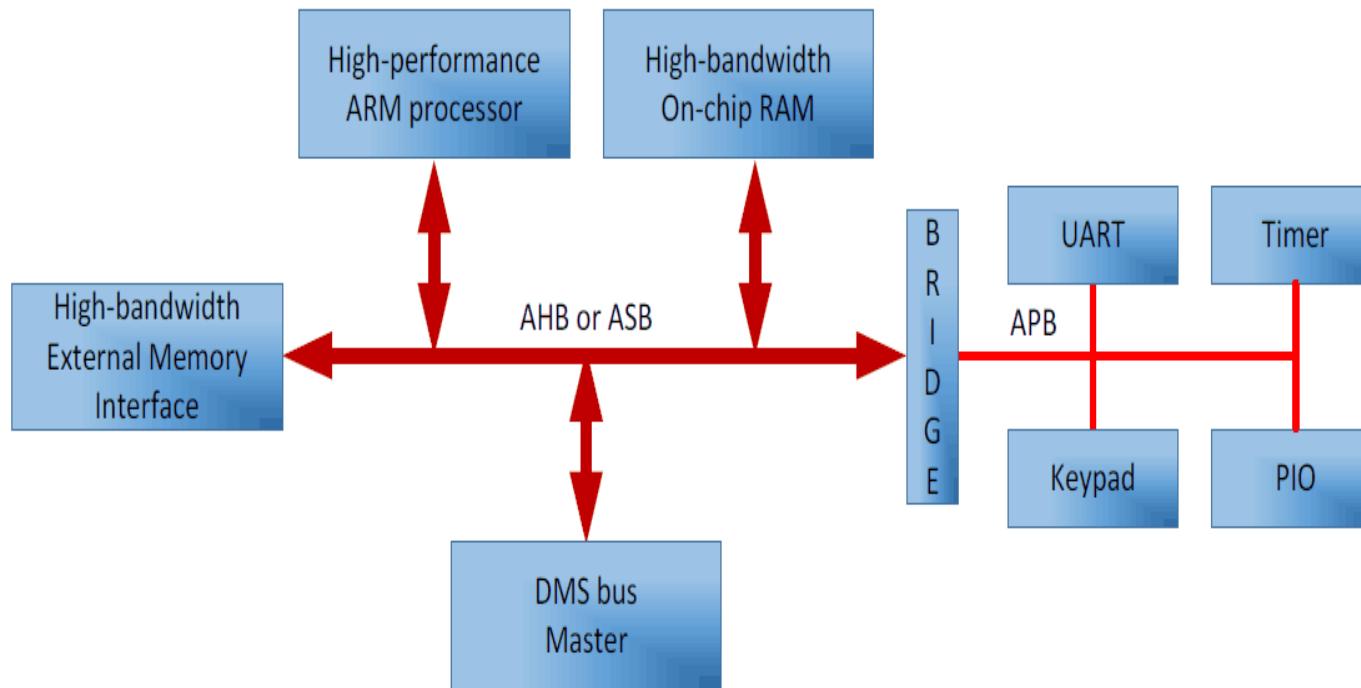
## LSL – Logical Shift Left

Example: Logical Shift Left by 4.



# AMBA Bus Protocol

## (Advanced Microcontroller Bus Architecture)

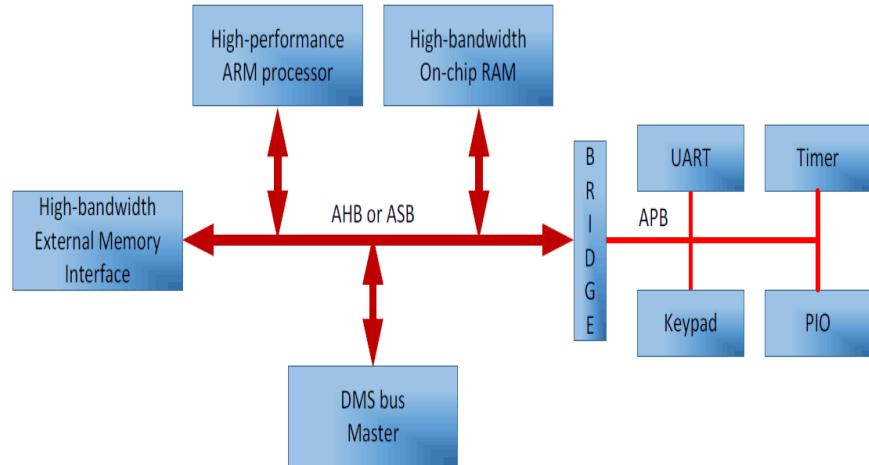


➤ **Adv.High Performance Bus (AHB)**

➤ **Adv. Peripheral Bus (APB)**

If AHB=60 Mhz  
Then APB=(1/4)AHB  
=15MHz

# AMBA Bus Architecture



The Advanced Micro controller Bus Architecture (**AMBA**) bus protocols is a set of interconnect specifications from ARM that standardizes on chip communication mechanisms between various functional blocks like USB, UART, I2C etc for building high performance SOC designs

# Thumb Instruction Set

- The ARM7TDMI processor has two instruction sets:
  - the 32-bit ARM instruction set
  - the 16-bit Thumb instruction set.

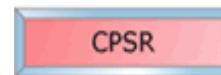
# **Thumb Instruction Set**

- The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions.
- Thumb instructions are 16 bits long,
- They have a corresponding 32-bit ARM instruction
- Thumb instructions operate with the standard ARM register configuration
- Interoperability between ARM and Thumb states.
- On execution, 16-bit Thumb instructions are transparently decompressed to full 32-bit ARM instructions in real time, without performance loss.

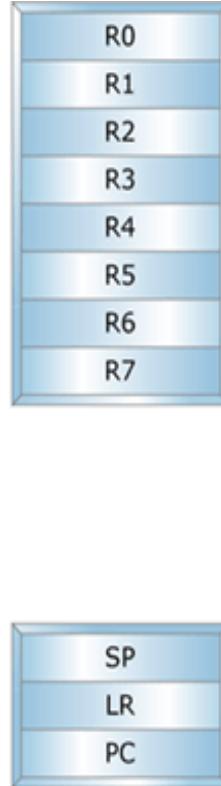
# **Advantages of Thumb Mode**

- Thumb code is typically **65% of the size of ARM code**,
- Provides **160% of the performance of ARM** code when running from a 16-bit memory system.
- The ARM7TDMI core ideally suited to embedded applications with **restricted memory bandwidth**.
- **Thumb has all the advantages of a 32-bit core:**
  - 32-bit address space
  - 32-bit registers
  - 32-bit shifter, and Arithmetic Logic Unit (ALU)
  - 32-bit memory transfer.

	<b>ARM (CPSR T=0)</b>	<b>Thumb (CPSR T=1)</b>
<b>Instruction size</b>	32 bit	16 bit
<b>Core Instructions</b>	58	30
<b>Conditional Execution</b>	Most	Only branch instructions
<b>Program Status Register</b>	R/W in privileged mode	No direct access
<b>Register Usage</b>	15 General Purpose registers (R0 – R15) +CPSR	8 GP Registers (R0-R7) + SP+LR PC



ARM



Thumb

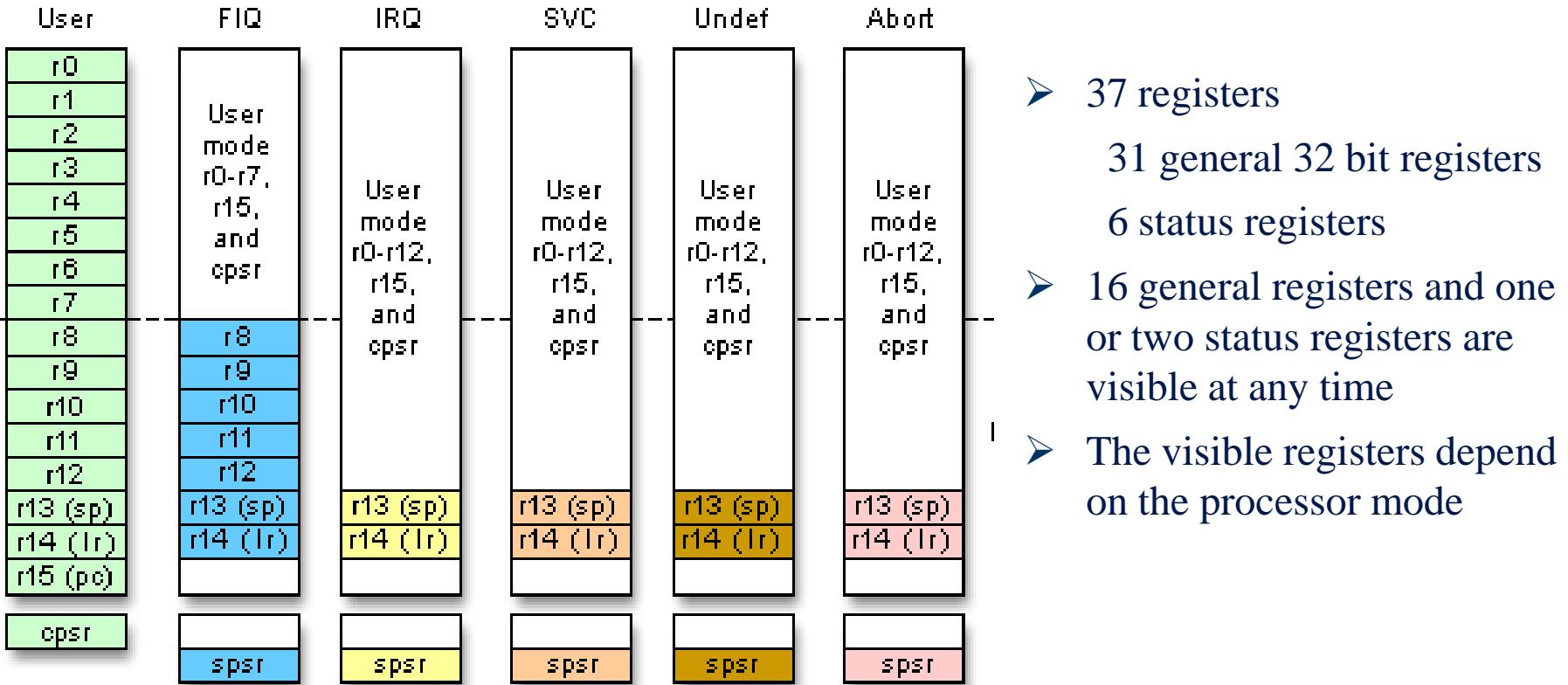
# ARM 7 – Operating Modes

The ARM has seven basic operating modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt Instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

- Exception modes
1. **User** : unprivileged mode under which most tasks run
  2. **System** : privileged mode using the same registers as user mode
  3. **Undef** : used to handle undefined instructions
  4. **Abort** : used to handle memory access violations
  5. **IRQ** : entered when a low priority (normal) interrupt is raised
  6. **FIQ** : entered when a high priority (fast) interrupt is raised
  7. **Supervisor** : entered on reset and when a Software Interrupt instruction is executed

# ARM 7 Register Set



Note: System mode uses the User mode register set

# Mode Switch

IRQ Mode  
LR,SP,SPSR

User Mode  
R0-R15

Main Pgm
Instruction 1
Instruction 2
INT x
Instruction 3
Instruction 4
Instruction 5
Instruction 6

PC  
To LR  
And CPSR to  
SPSR

Go to ISR  
PC ← Vector  
Address

Restore CPSR from  
SPSR

Restore PC from LR

Get original PC  
from LR  
SPSR to CPSR

ISR
Instruction 1
Instruction 2
Instruction 3
Clr int flag

# ARM Nomenclature

ARM *x*<sub>1</sub>*y*<sub>2</sub>*z*<sub>3</sub> T D M I E J F S (Example: ARM7-TDMI-S)

*x* - Series

*y* - MMU

*z* - Cache

T - Thumb

D - Debugger

M - Multiplier

I - Embedded In-Circuit Emulator (ICE) macrocell

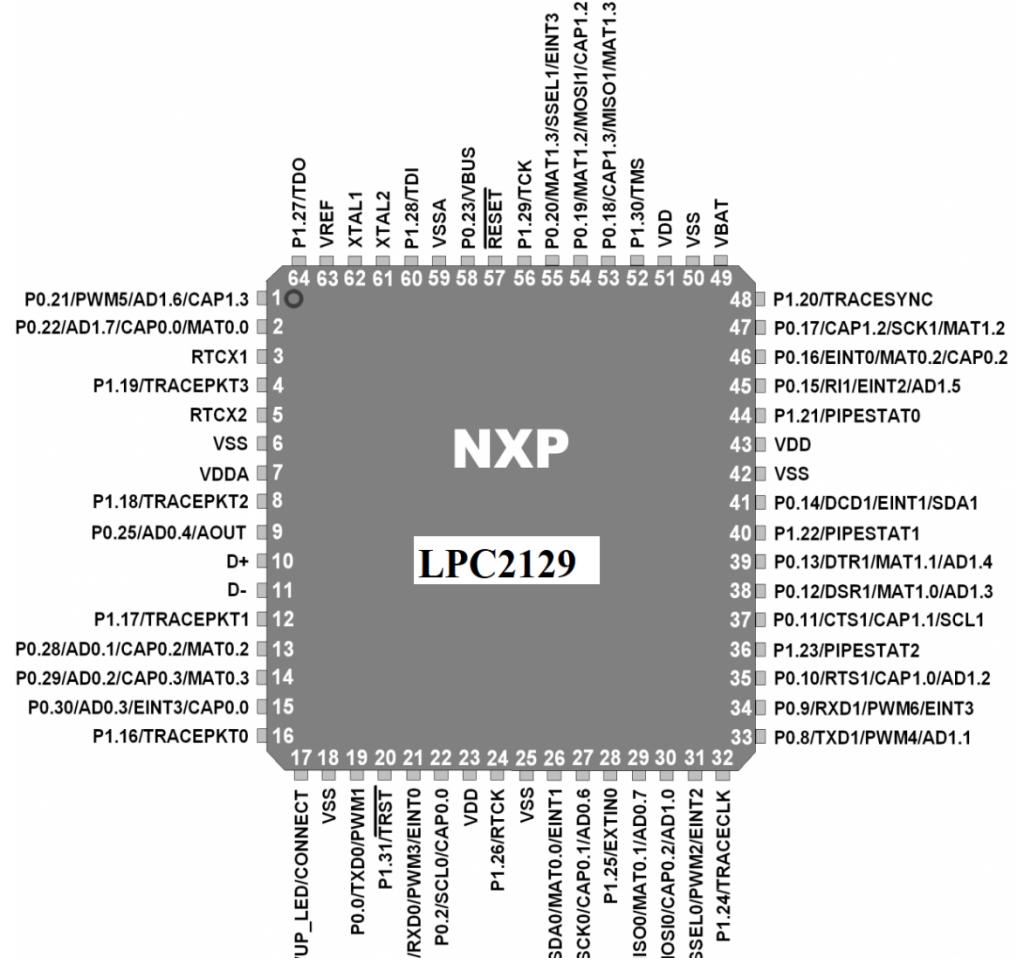
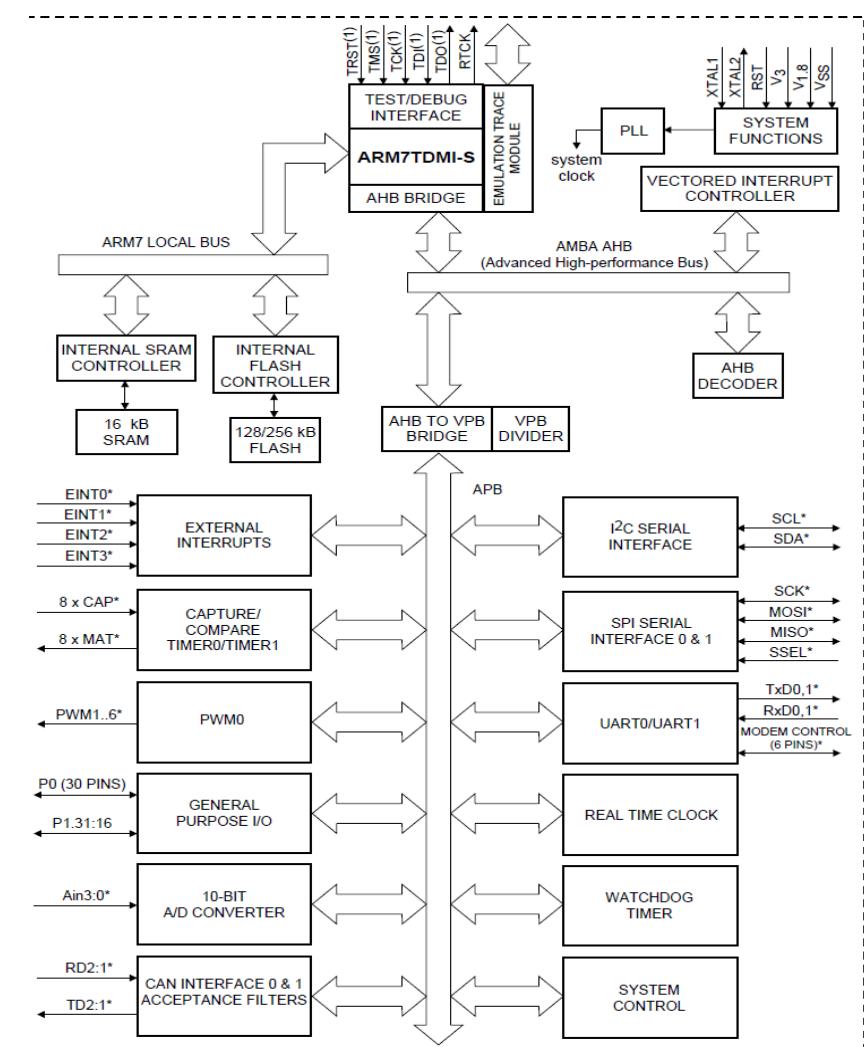
E - Enhanced Instructions for DSP

J - JAVA acceleration by Jazelle

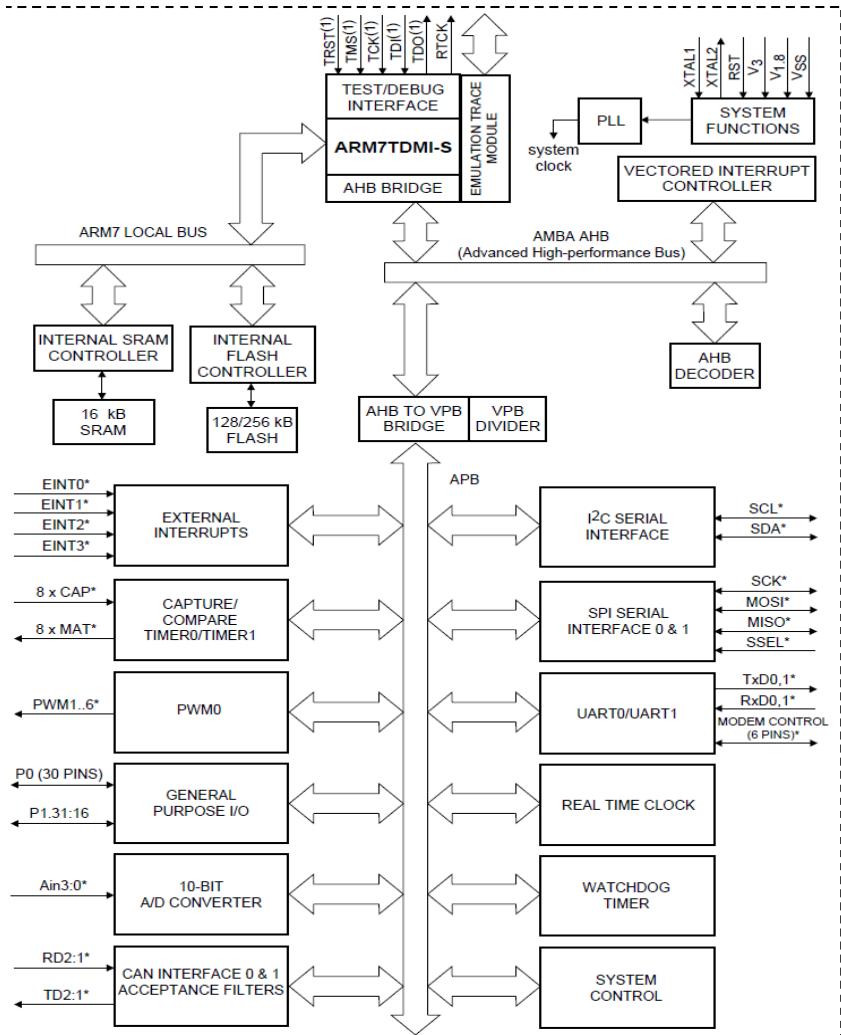
F - Floating-point

S - Synthesizable version

# **LPC2129 Microcontroller Internal Architecture (ARM7 Based Controller)**

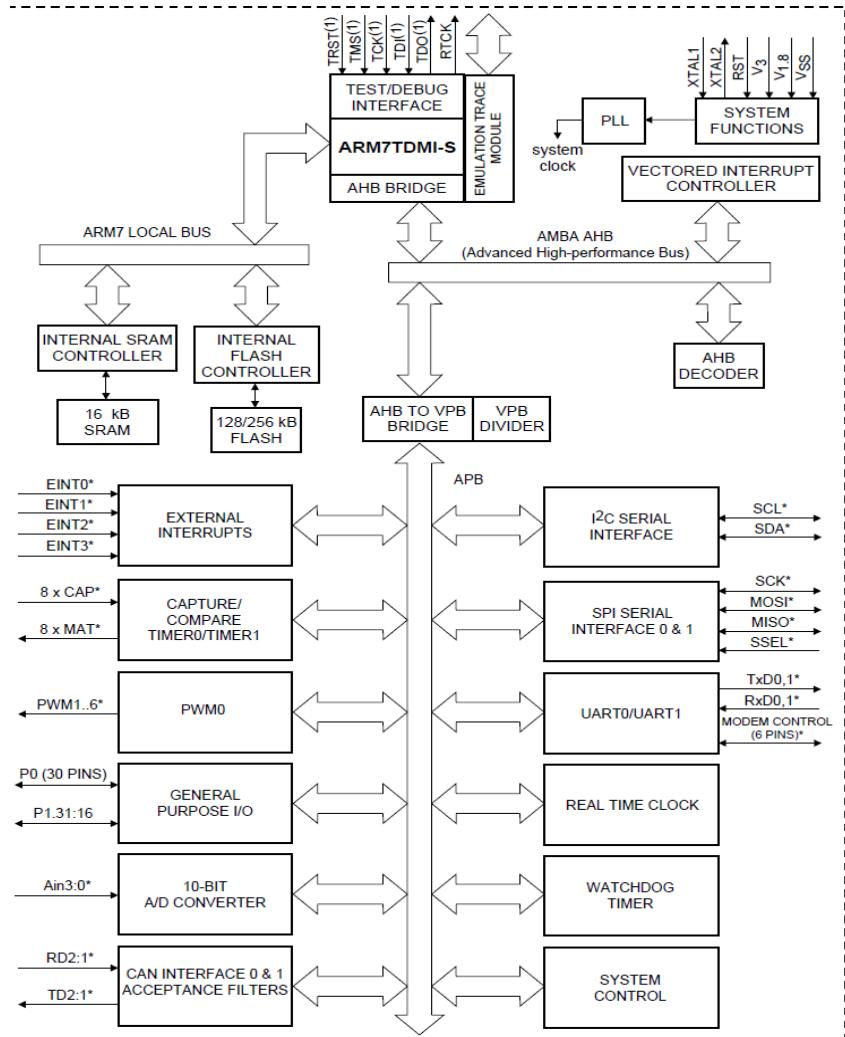


# LPC2129 Microcontroller Internal Architecture



- **LQFP 64 A Low-profile Quad Flat Package**
- **No of pins - 64**
- **ARM7TDMI @60 MHz maximum CPU clock**
- **CPU Clock can be programmable using on-chip PLL**
- **On chip SRAM - 16Kbytes**
- **On chip flash –256 Kb**
- **10 bit ADC channels – 4**
- **Two 32-bit timers/external**

# LPC2129 Microcontroller Internal Architecture



- **PWM unit (6 outputs),**
- **Real Time Clock (RTC),**
- **Watchdog Timer**
- **Vectored Interrupt Controller with configurable priorities.**
- **Two UARTs**
- **I2C-bus**
- **Two SPI interfaces.**
- **TWO interconnected CAN interfaces**

# Memory Map of LPC2129 Microcontroller

4.0 GB	AHB Peripherals	0xFFFF FFFF
3.75 GB	VPB Peripherals	0xF000 0000
3.5 GB		0xE000 0000
3.0 GB	Reserved for External Memory	0xC000 0000
2.0 GB	Boot Block (re-mapped from On-Chip Flash memory)	0x8000 0000
	Reserved for On-Chip Memory	
1.0 GB	16 kB On-Chip Static RAM	0x4000 3FFF 0x4000 0000
		0x0004 0000 0x0003 FFFF
	256 kB On-Chip Non-Volatile Memory (LPC2129/2194/2292/2294)	0x0002 0000
	128 kB On-Chip Non-Volatile Memory (LPC2119)	0x0001 FFFF 0x0000 0000
0.0 GB		

# General Purpose Input & Output (GPIO)

- LPC2129
  - Two 32 bit general purpose IO ports
  - **PORT 0 (32 pins)**
    - P0.0 to P0.31
  - **PORT 1 (32 pins)**
    - 16 pins available for GPIO
    - P1.16 to P1.31
  - Pins controlled using GPIO registers

# GPIO register set

- **IODIRx**
- Def → 0      0 → input port  
                  1 → output port
- **IOSET:** 1 → set the port pin
- **IOCLR:** 1 → clear the port pin
- **IOPIN** : to check the status of the port pin

**toggle the port pin P0.1**

**P0.1 ip or op**

**IODIRO**

**P0.1=1**

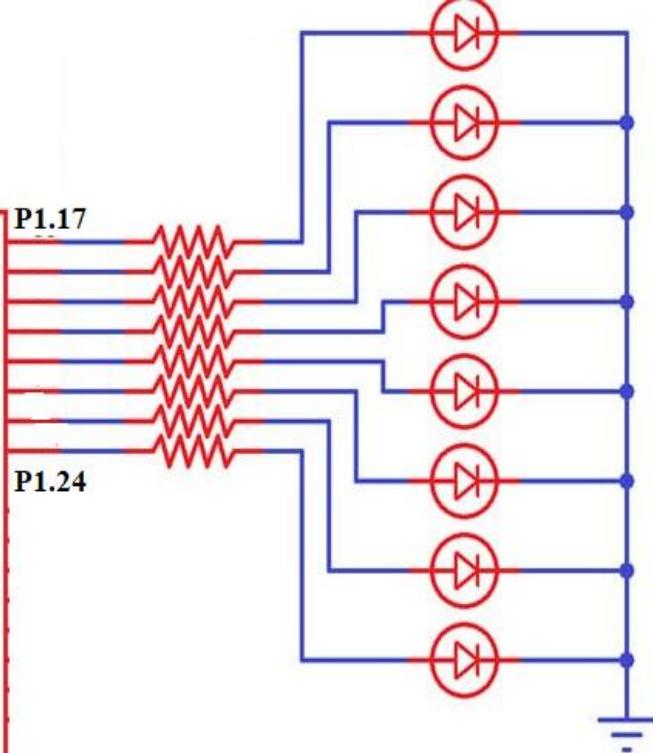
**DELAY**

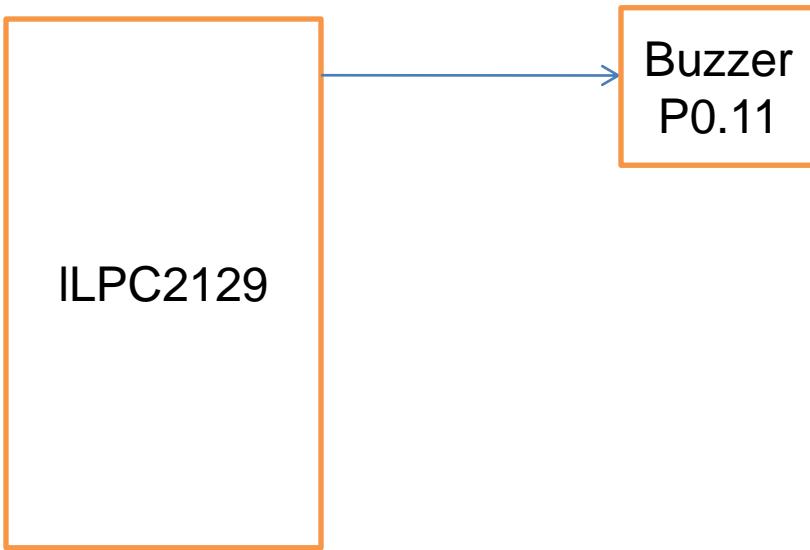
**P0.1=0**

**DELAY**

- Set a bit (OR with 1 in the bit pos)
- Clear a bit (AND with 0 in the bit pos)
  - $\sim$ (set flag)
- Toggle a bit (XOR with set flag)

L  
P  
C  
2  
1  
2  
9

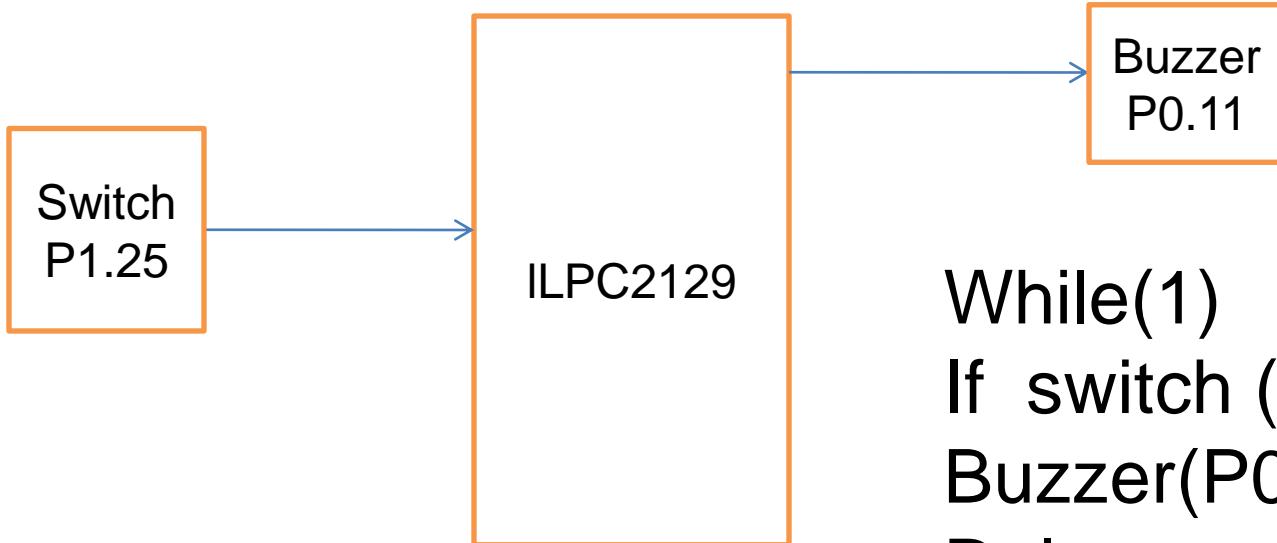




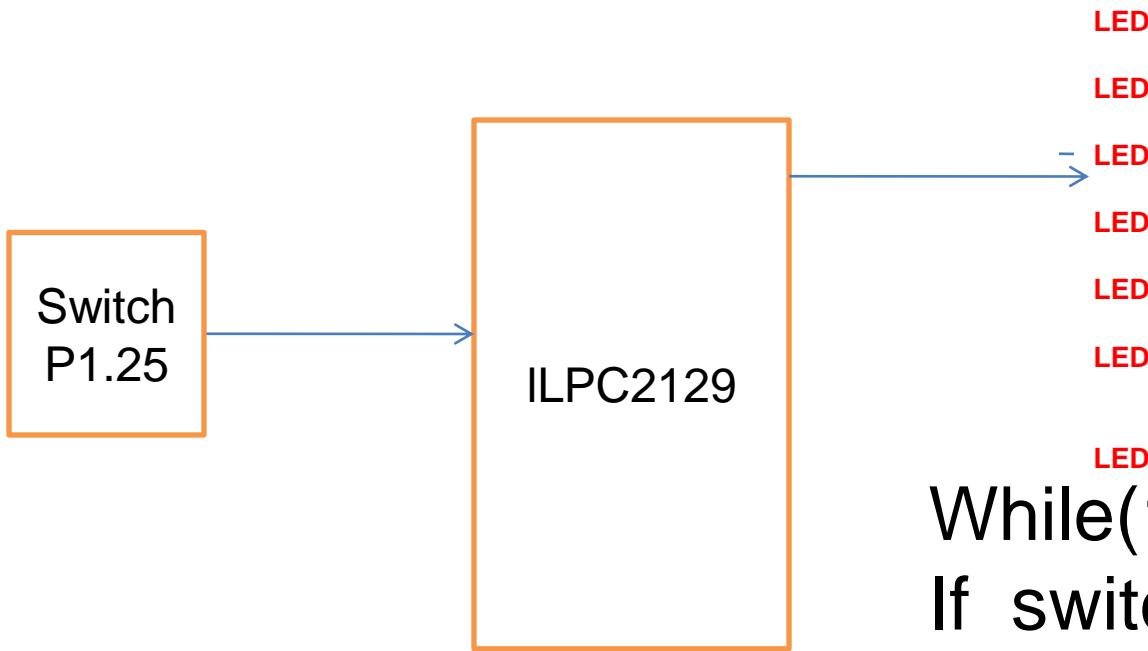
```
While(1)
{
    Buzzer(P0.11)=1
    Delay
    Bezzer (P0.11)=0
    delay
}
```

The code snippet is written in a pseudocode-like syntax, likely for a microcontroller. It defines an infinite loop ("While(1)"). Inside the loop, the state of pin P0.11 is set to 1 ("Buzzer(P0.11)=1"), followed by a delay ("Delay"). Then, the state of pin P0.11 is set to 0 ("Bezzer (P0.11)=0"), followed by another delay ("delay"). This creates a continuous alternating tone.

# IOPIN



```
While(1)
If switch (P1.25) ==1
Buzzer(P0.11)=1
Delay
Bezzer (P0.11)==0
```



```
While(1)
If switch (P1.25) ==1
LED= turn on
Delay
LED= turn off
```