

**Lecture 1****Architecture and Introduction**

Note: Much of these notes are taken from the book The Insider's Guide To The Philips ARM7-Based Microcontrollers by *Trevor Martin* [www.hitex.co.uk/arm](http://www.hitex.co.uk/arm)

The ARM (Advanced RISC Machine) is a 32-bit microcontroller created by a consortium of companies and manufactured in many different versions. It is widely used in modems, cell phones, personal audio, pagers, cameras, and many more embedded high end applications.

ARM is produced by many different companies and each adds its own variations. We will be using the ARM7 produced by Philips at the LPC 2138.

RISC – Reduced Instruction Set Computer.

RISC/CISC controversy and history

- In the 1970s as IC manufacturers were able to put more and more on a single chip, computer architects needed to decide what to do with the extra space.
- The prevailing idea was to create more complex instructions which are then executed in microcode. This reduces the use of program memory to fetch instructions and should run faster. This was called CISC for Complex Instruction Set Computer.
- A counter idea, created by IBM, was to use the extra space for things like more registers, cache memory, and numerical processing ability. This solution kept the instruction set very simple so that it took up little extra circuitry. This was called RISC and began about 1980.
- Many new RISC machines were built such as the SUN Sparc. The Intel x86 line represents an architecture that leans toward the CISC side.
- By the mid 1990s it was clear that the ideal machine was neither RISC nor CISC but something in between. Today most machines have features of both.

Generic RISC features:

- A large number of general purpose registers along with compiler technology to optimize register usage.
- A limited and simple instruction set
- An emphasis on optimizing the instruction pipeline
- Load and store architecture with simple addressing modes

Today, most CPU's have some RISC-like features but pure RISC machines are uncommon. The ARM processor has its origins in 1983 from Acorn Computers Ltd. In 1990 Acorn spun off a new company called Advanced RISC Machines Ltd. which licenses its ARM core to other companies for manufacture. Many other companies now make versions of the ARM processor including Intel, Atmel, Texas Instruments,

FreeScale, and many others. The ARM7 is one of the more successful version of the ARM processor and is the processor in the Nintendo and Gameboy computer games.

**Feature Summary ARM 7 core**

- 32-bit RISC processor (32-bit data & address bus)
- Big and Little Endian operating modes
- High performance RISC  
17 MIPS sustained @ 25 MHz (25 MIPS peak) @ 3V
- Low power consumption  
0.6mA/MHz @ 3V fabricated in .8 $\mu$ m CMOS
- Fully static operation  
ideal for power-sensitive applications
- Fast interrupt response  
for real-time applications
- Virtual Memory System Support
- Excellent high-level language support
- Simple but powerful instruction set

**The instruction set comprises eleven basic instruction types:**

- Two of these make use of the on-chip arithmetic logic unit, barrel shifter and multiplier to perform high-speed operations on the data in a bank of 31 registers, each 32 bits wide;
- Three classes of instruction control data transfer between memory and the registers, one optimized for flexibility of addressing, another for rapid context switching and the third for swapping data;
- Three instructions control the flow and privilege level of execution.
- Three types are dedicated to the control of external coprocessors which allow the functionality of the instruction set to be extended off-chip in an open and uniform way.

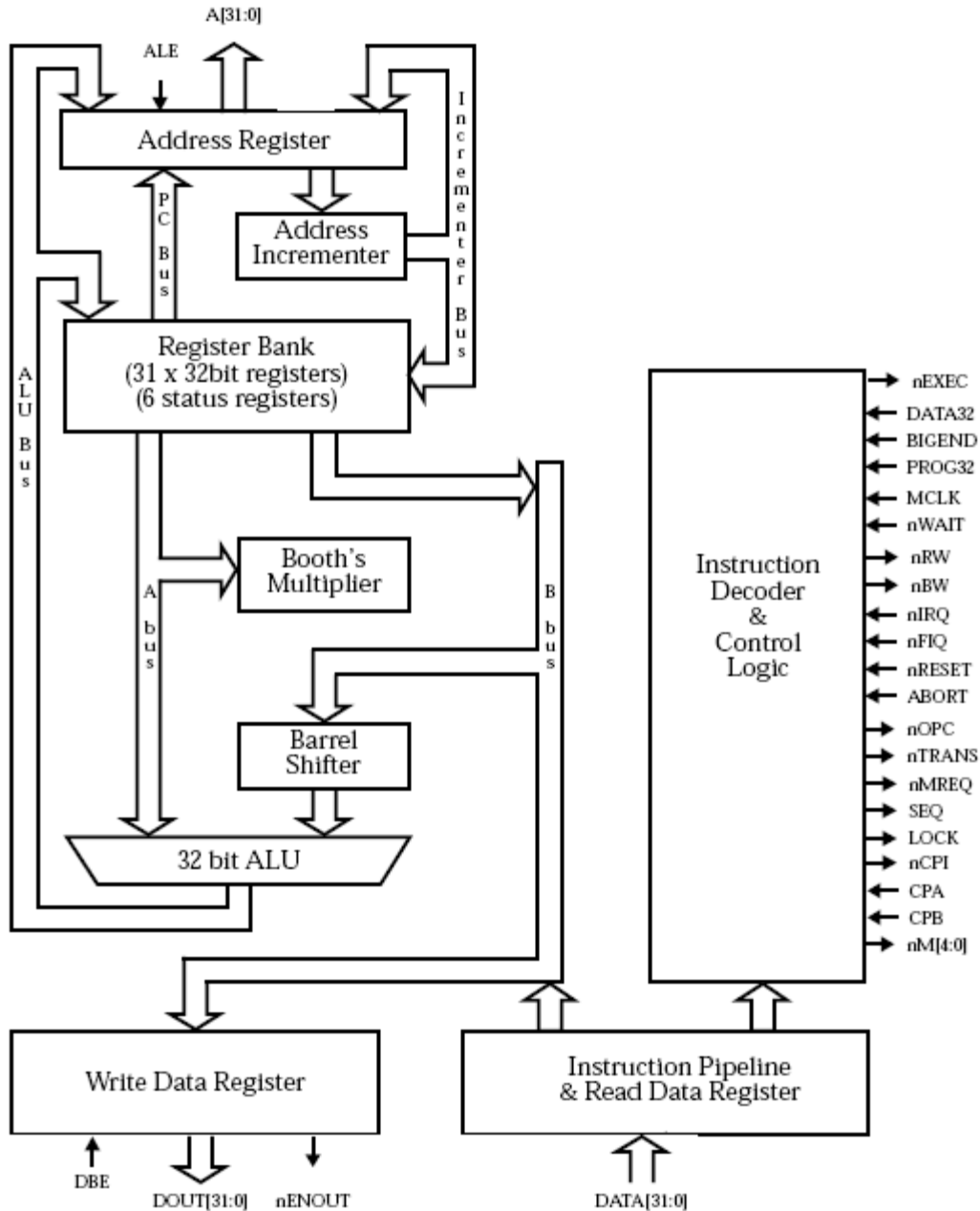
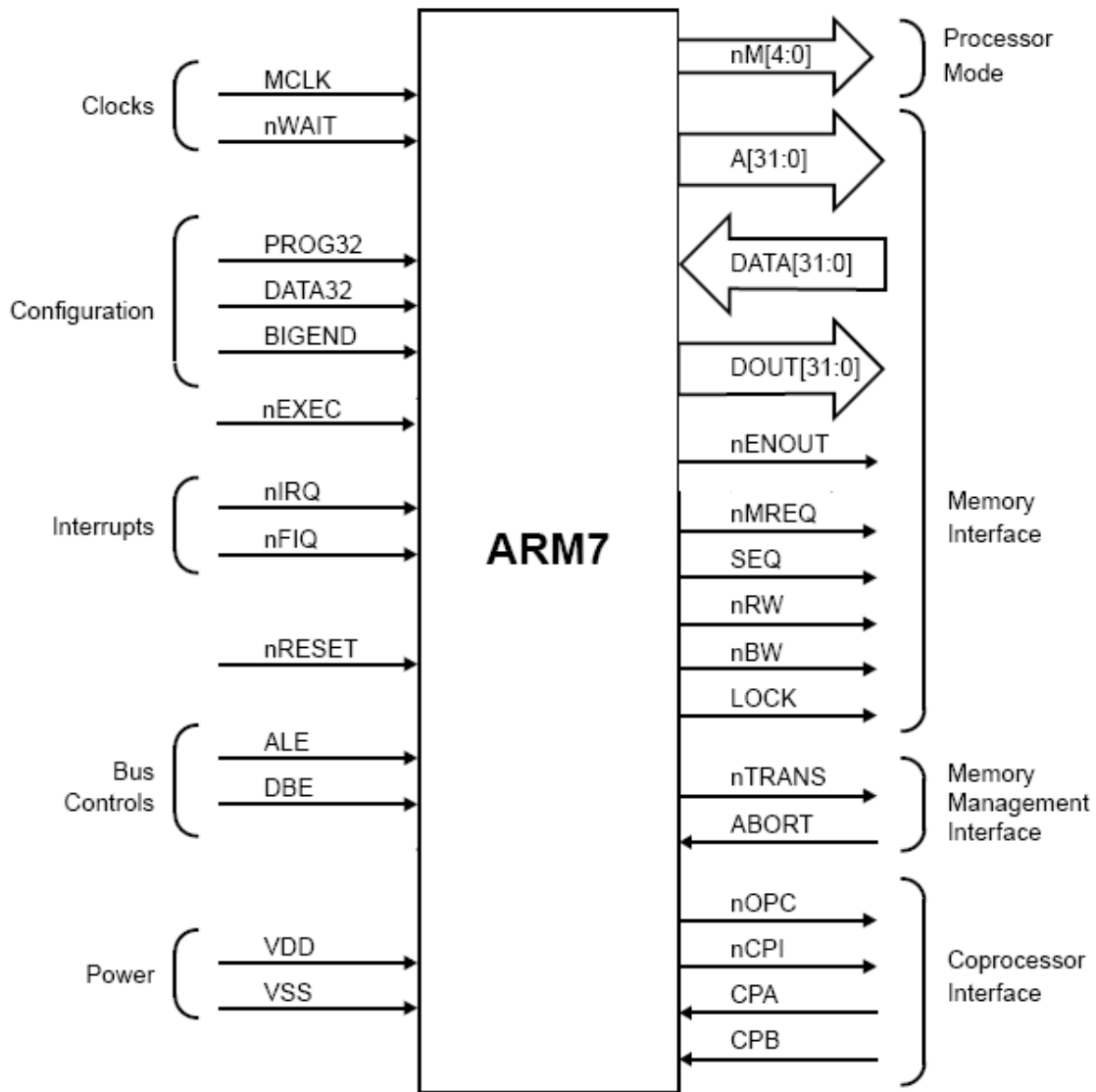


Figure 1: ARM7 Block Diagram

Notes:

1. Booth's multiplier is a hardware multiplier capable of multiplying 2's complement numbers.
2. Barrel shifter is a hardware shifter capable of shifting up to 32 positions in one clock period.
3. Not microprogrammed.

## 1.2 ARM7 Functional Diagram



### ARM7 supports six modes of operation:

- (1) User mode (usr): the normal program execution state
- (2) FIQ mode (fiq): designed to support a data transfer or channel process
- (3) IRQ mode (irq): used for general purpose interrupt handling
- (4) Supervisor mode (svc): a protected mode for the operating system
- (5) Abort mode (abt): entered after a data or instruction prefetch abort
- (6) Undefined mode (und): entered when an undefined instruction is executed

Mode changes may be made under software control or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The other modes, known as *privileged modes*, will be entered to service interrupts or exceptions or to access protected resources.

### General Registers and Program Counter Modes

| User32   | FIQ32    | Supervisor32 | Abort32  | IRQ32    | Undefined32 |
|----------|----------|--------------|----------|----------|-------------|
| R0       | R0       | R0           | R0       | R0       | R0          |
| R1       | R1       | R1           | R1       | R1       | R1          |
| R2       | R2       | R2           | R2       | R2       | R2          |
| R3       | R3       | R3           | R3       | R3       | R3          |
| R4       | R4       | R4           | R4       | R4       | R4          |
| R5       | R5       | R5           | R5       | R5       | R5          |
| R6       | R6       | R6           | R6       | R6       | R6          |
| R7       | R7       | R7           | R7       | R7       | R7          |
| R8       | R8_fiq   | R8           | R8       | R8       | R8          |
| R9       | R9_fiq   | R9           | R9       | R9       | R9          |
| R10      | R10_fiq  | R10          | R10      | R10      | R10         |
| R11      | R11_fiq  | R11          | R11      | R11      | R11         |
| R12      | R12_fiq  | R12          | R12      | R12      | R12         |
| R13      | R13_fiq  | R13_svc      | R13_abt  | R13_irq  | R13_und     |
| R14      | R14_fiq  | R14_svc      | R14_abt  | R14_irq  | R14_und     |
| R15 (PC) | R15 (PC) | R15 (PC)     | R15 (PC) | R15 (PC) | R15 (PC)    |

### Program Status Registers

|      |          |          |          |          |          |
|------|----------|----------|----------|----------|----------|
| CPSR | CPSR     | CPSR     | CPSR     | CPSR     | CPSR     |
|      | SPSR_fiq | SPSR_svc | SPSR_abt | SPSR_irq | SPSR_und |

Figure 5: Register Organisation

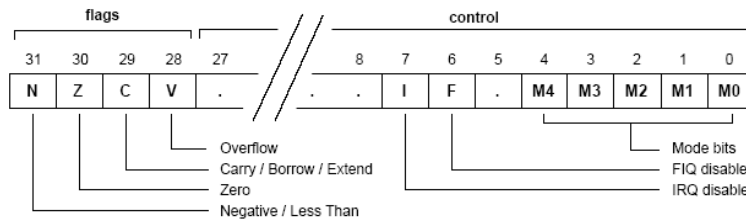


Figure 6: Format of the Program Status Registers (PSRs)

#### Notes:

1. There are 16 32-bit registers for general usage. R0 to R12 are called user registers. R13-R15 are special use registers. R13 is the stack pointer, R14 is the link register, and R15 is the program counter.
2. In addition to R0 to R15 there are other registers such as R8\_fiq which are banked and switched in depending on the exception mode. In total there are 31 registers.
3. The exception modes are FIQ (fast interrupt), Supervisor (Software interrupt), Abort (Data and prefetch abort), IRQ (interrupt), and Undefined (for undefined op codes).
4. Each mode has its own program status registers (PSR)
5. When an exception occurs the CPU changes modes and jumps to the appropriate interrupt. The register bank for that mode is instantly switched in.

| M[4:0] | Mode       | Accessible register set       |                |
|--------|------------|-------------------------------|----------------|
| 10000  | User       | PC, R14..R0                   | CPSR           |
| 10001  | FIQ        | PC, R14_fiq..R8_fiq, R7..R0   | CPSR, SPSR_fiq |
| 10010  | IRQ        | PC, R14_irq..R13_irq, R12..R0 | CPSR, SPSR_irq |
| 10011  | Supervisor | PC, R14_svc..R13_svc, R12..R0 | CPSR, SPSR_svc |
| 10111  | Abort      | PC, R14_abt..R13_abt, R12..R0 | CPSR, SPSR_abt |
| 11011  | Undefined  | PC, R14_und..R13_und, R12..R0 | CPSR, SPSR_und |

Table 2: The Mode Bits

The Current Program Status Register (CPSR) is a 32-bit register which holds the flags and the mode number.

Exceptions force an interrupt to a vector which is stored in lower memory.

### 3.4.6 Vector Summary

| Address    | Exception             | Mode on entry |
|------------|-----------------------|---------------|
| 0x00000000 | Reset                 | Supervisor    |
| 0x00000004 | Undefined instruction | Undefined     |
| 0x00000008 | Software interrupt    | Supervisor    |
| 0x0000000C | Abort (prefetch)      | Abort         |
| 0x00000010 | Abort (data)          | Abort         |
| 0x00000014 | -- reserved --        | --            |
| 0x00000018 | IRQ                   | IRQ           |
| 0x0000001C | FIQ                   | FIQ           |

Table 3: Vector Summary

### Instruction Set

All of the assembly language instructions are 32-bits wide. The figure below shows the format for the machine code.

|      | 31 | 28 | 27 | 26                   | 25                   | 24 | 23  | 22 | 21  | 20 | 19            | 16        | 15     | 12 | 11     | 8 | 7                        | 5 | 4 | 3  | 0                    |                    |                      |                  |
|------|----|----|----|----------------------|----------------------|----|-----|----|-----|----|---------------|-----------|--------|----|--------|---|--------------------------|---|---|----|----------------------|--------------------|----------------------|------------------|
| Cond | 0  | 0  | 1  | Opcode               |                      |    | S   | Rn |     | Rd |               | Operand 2 |        |    |        |   |                          |   |   |    |                      | Data Processing    |                      |                  |
| Cond | 0  | 0  | 0  | 0                    | 0                    | 0  | A   | S  | Rd  |    | Rn            |           | Rs     |    | 1      |   | 0                        | 0 | 1 | Rm |                      | Multiply           |                      |                  |
| Cond | 0  | 0  | 0  | 1                    | 0                    | B  | 00  |    | Rn  |    | Rd            |           | 0      |    | 0      | 0 | 0                        | 1 |   | 0  | 0                    | 1                  | Rm                   | Single Data Swap |
| Cond | 0  | 1  | 1  | P                    | U                    | B  | W   | L  | Rn  |    | Rd            |           | offset |    |        |   |                          |   |   |    |                      |                    | Single Data Transfer |                  |
| Cond | 0  | 1  | 1  | XXXXXXXXXXXXXXXXXXXX |                      |    |     |    |     |    |               |           |        |    |        |   |                          |   |   |    | 1                    | XXXX               | Undefined            |                  |
| Cond | 1  | 0  | 0  | P                    | U                    | S  | W   | L  | Rn  |    | Register List |           |        |    |        |   |                          |   |   |    | Block Data Transfer  |                    |                      |                  |
| Cond | 1  | 0  | 1  | L                    | offset               |    |     |    |     |    |               |           |        |    |        |   |                          |   |   |    |                      | Branch             |                      |                  |
| Cond | 1  | 1  | 0  | P                    | U                    | N  | W   | L  | Rn  |    | CRd           |           | CP#    |    | offset |   |                          |   |   |    | Coproc Data Transfer |                    |                      |                  |
| Cond | 1  | 1  | 1  | 0                    | CP Opc               |    | CRn |    | CRd |    | CP#           |           | CP     | 0  | CRm    |   | Coproc Data Operation    |   |   |    |                      |                    |                      |                  |
| Cond | 1  | 1  | 1  | 0                    | CP Opc               |    | CRn |    | Rd  |    | CP#           |           | CP     | 1  | CRm    |   | Coproc Register Transfer |   |   |    |                      |                    |                      |                  |
| Cond | 1  | 1  | 1  | 1                    | ignored by processor |    |     |    |     |    |               |           |        |    |        |   |                          |   |   |    |                      | Software Interrupt |                      |                  |

Figure 7: Instruction Set Summary

The condition field makes every instruction conditional. There are 4-bits allowed for the this field so that every ARM instruction has 16 variations. For example,  
EQMOV R1, #0x00800000  
will only move the immediate field #0x00800000 into register R1 if the Z flag (zero) is set.

## 4.2 The Condition Field

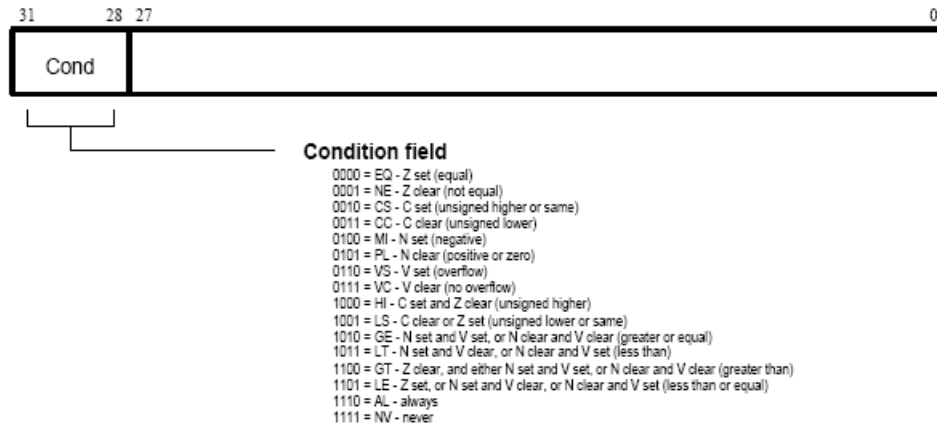


Figure 8: Condition Codes

Notes on the instruction set:

1. The branching instructions are all relative and allow branching of  $\pm 32$ MB of the current location.
2. A modified version of the branch is called a "branch link". It stores the current PC + 4 into the the link register (R14). This is used to bypass the stack when a subprogram is called which does not further call other subprograms.
3. Data processing instructions have a result register plus two operand registers. For example, `ADD R1, R1, #1` add 1 to register 1 and stores the result in register 1.
4. The ARM7 has a multiply instruction (signed or unsigned) as well as a MAC (multiply and accumulate) instruction for signal processing.
5. The ARM instruction set is a set of 32-bit instructions which can be very space inefficient on many applications. Hence many ARM processors also support a 16-bit instruction set called the THUMB instruction set. Thumb instructions are actually stored as 16-bit instructions but are expanded by the processor to become 32-bit ARM instructions for execution. This greatly improves space efficiency.
6. All ARM7 instructions are conditionally executed, which means that their execution may or may not take place depending on the values of the N, Z, C and V flags in the CPSR. If the *always* (AL) condition is specified, the instruction will be executed irrespective of the flags. The *never* (NV) class of condition codes shall not be used as they will be redefined in future variants of the ARM architecture. If a NOP is required it is suggested that `MOV R0,R0` be used. The assembler treats the absence of a condition code as though *always* had been specified.
7. The THUMB instruction set is a compressed version of the ARM instruction set. For example the add instruction in ARM is `ADD R0, R0, R1`. For the THUMB instruction set this instruction becomes `ADD R0, R1`.
8. The THUMB instructions also have access to only R0 to R7 plus R13, R14, and R15.

## LPC2138 from Philips

### FEATURES

- 16/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
- 8/16/32 kByte of on-chip static RAM and 32/64/512 kB of on-chip Flash program memory. 128 bit wide interface/accelerator enables high speed 60 MHz operation.
- In-System/In-Application Programming (ISP/IAP) via on-chip boot-loader software. Single Flash sector or full chip erase in 400 ms and programming of 256 bytes in 1 ms.
- Embedded ICE® RT and Embedded Trace interfaces offer real-time debugging with the on-chip RealMonitor™ software and high speed tracing of instruction execution.
- One (LPC2131/2132) or two (LPC2138) 8 channel 10-bit A/D converters provide a total of up to 16 analog inputs, with conversion times as low as 2.44 s per channel.
- Single 10-bit D/A converter provides variable analog output. (LPC2132/2138 only)
- Two 32-bit timers/counters (with four capture and four compare channels each), PWM unit (six outputs) and watchdog.
- Real-time clock equipped with independent power and clock supply permitting extremely low power consumption in powersave modes.
- Multiple serial interfaces including two UARTs (16C550), two Fast I2C (400 kbit/s), SPI™ and SSP with buffering and variable data length capabilities.
- Vectored interrupt controller with configurable priorities and vector addresses.
- Up to 47 of 5 V tolerant general purpose I/O pins in tiny LQFP64 package.
- Up to nine edge or level sensitive external interrupt pins available.
- 60 MHz maximum CPU clock available from programmable on-chip Phase-Locked Loop (PLL) with settling time of 100 microseconds.
- On-chip crystal oscillator with an operating range of 1 MHz to 30 MHz.
- Power saving modes include Idle and Power-down.
- Individual enable/disable of peripheral functions as well as peripheral clock scaling down for additional power optimization.
- Processor wake-up from Power-down mode via external interrupt.
- Single power supply chip with Power-On Reset (POR) and Brown-Out Detection (BOD) circuits:
  - CPU operating voltage range of 3.0 V to 3.6 V (3.3 V 10 %) with 5 V tolerant I/O pads.

Table 1: LPC2131/2132/2138 device information

| Device  | No. of pins | On-chip RAM | On-chip FLASH | No. of 10-bit AD Channels | No. of 10-bit DA Channels | Note |
|---------|-------------|-------------|---------------|---------------------------|---------------------------|------|
| LPC2131 | 64          | 8 kB        | 32            | 8                         | -                         | -    |
| LPC2132 | 64          | 16 kB       | 64            | 8                         | -                         | -    |
| LPC2138 | 64          | 32 kB       | 512           | 16                        | 1                         | -    |

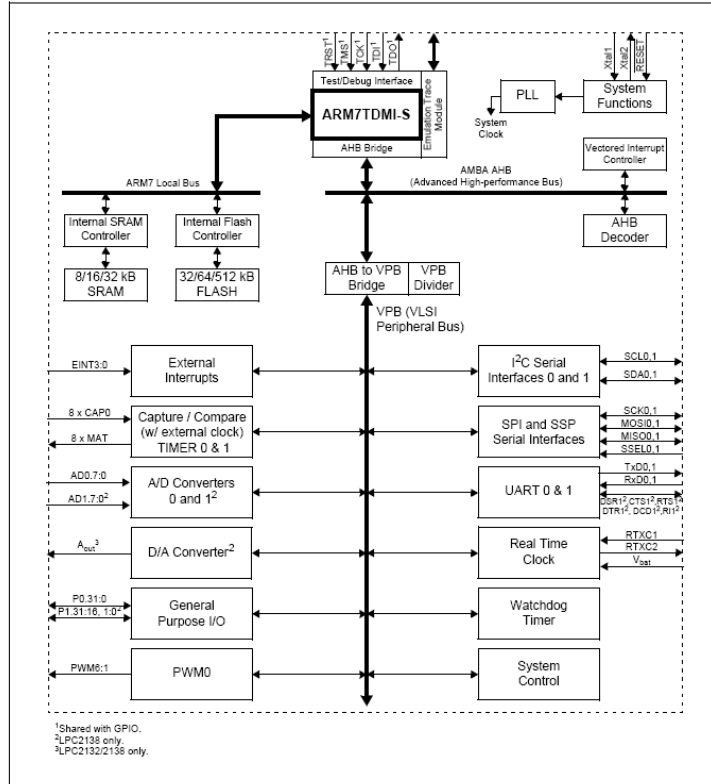


Figure 1: LPC2131/2132/2138 Block Diagram

### MEMORY MAPS

The LPC2131/2132/2138 incorporates several distinct memory regions, shown in the following figures. Figure 2 shows the overall map of the entire address space from the user program viewpoint following reset. The interrupt vector area supports address re-mapping, which is described later in this section.

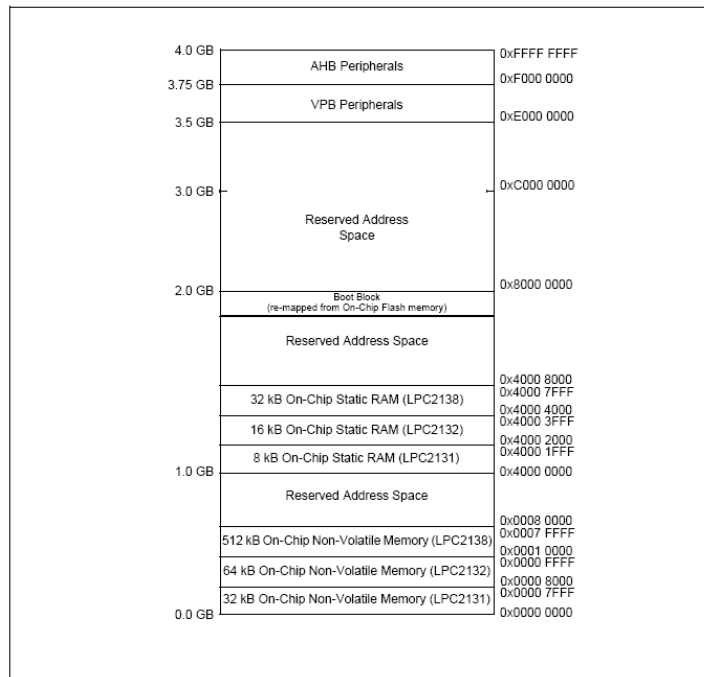


Figure 2: System Memory Map

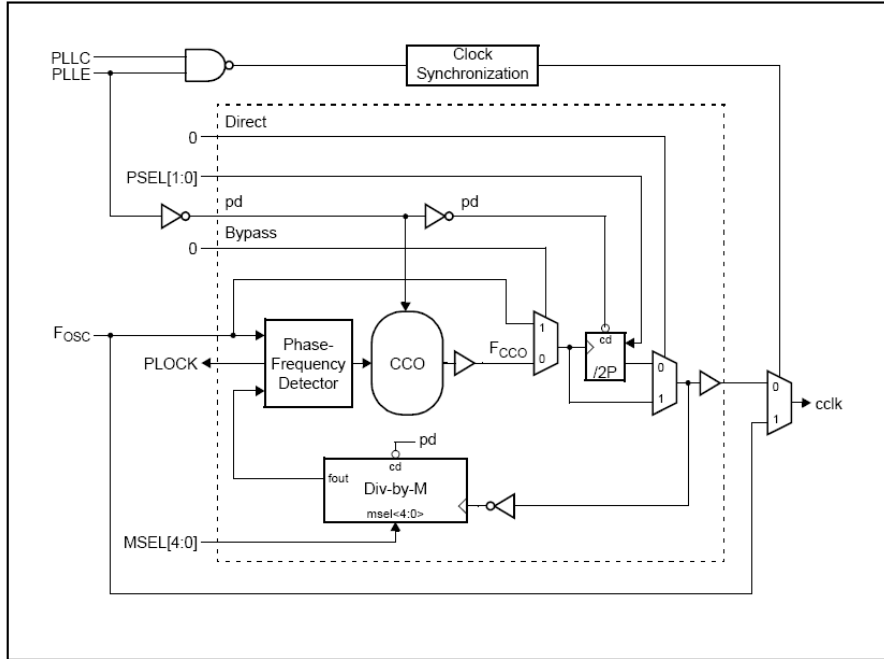


Figure 10: PLL Block Diagram

LPC2131/2132/2138 PINOUT

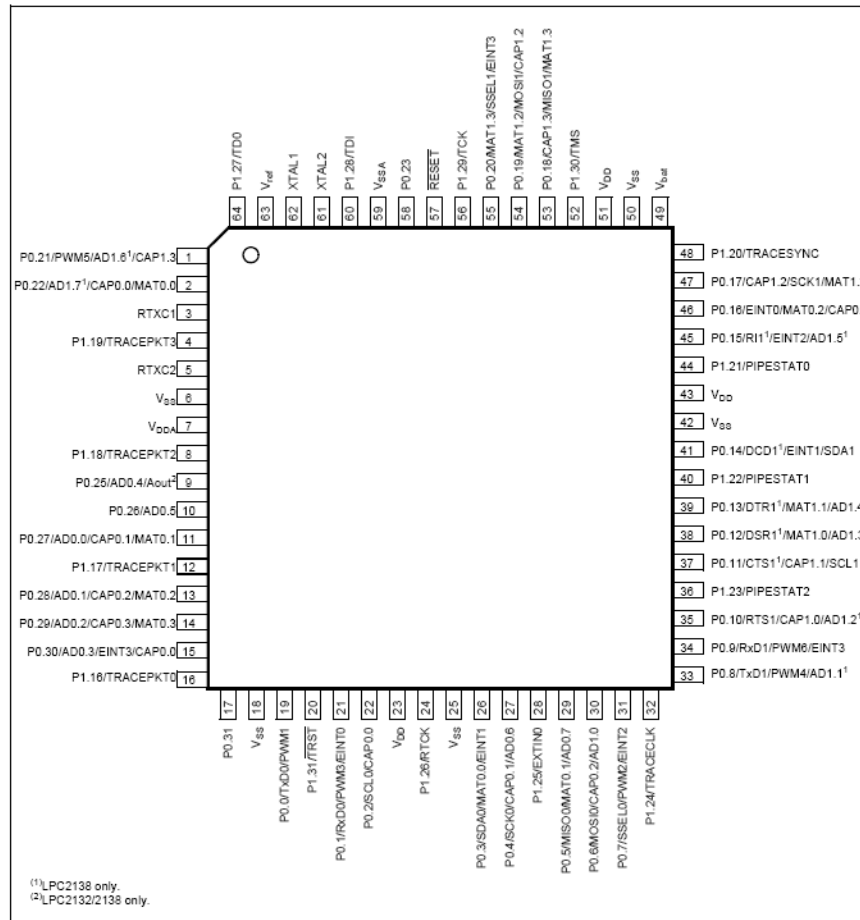
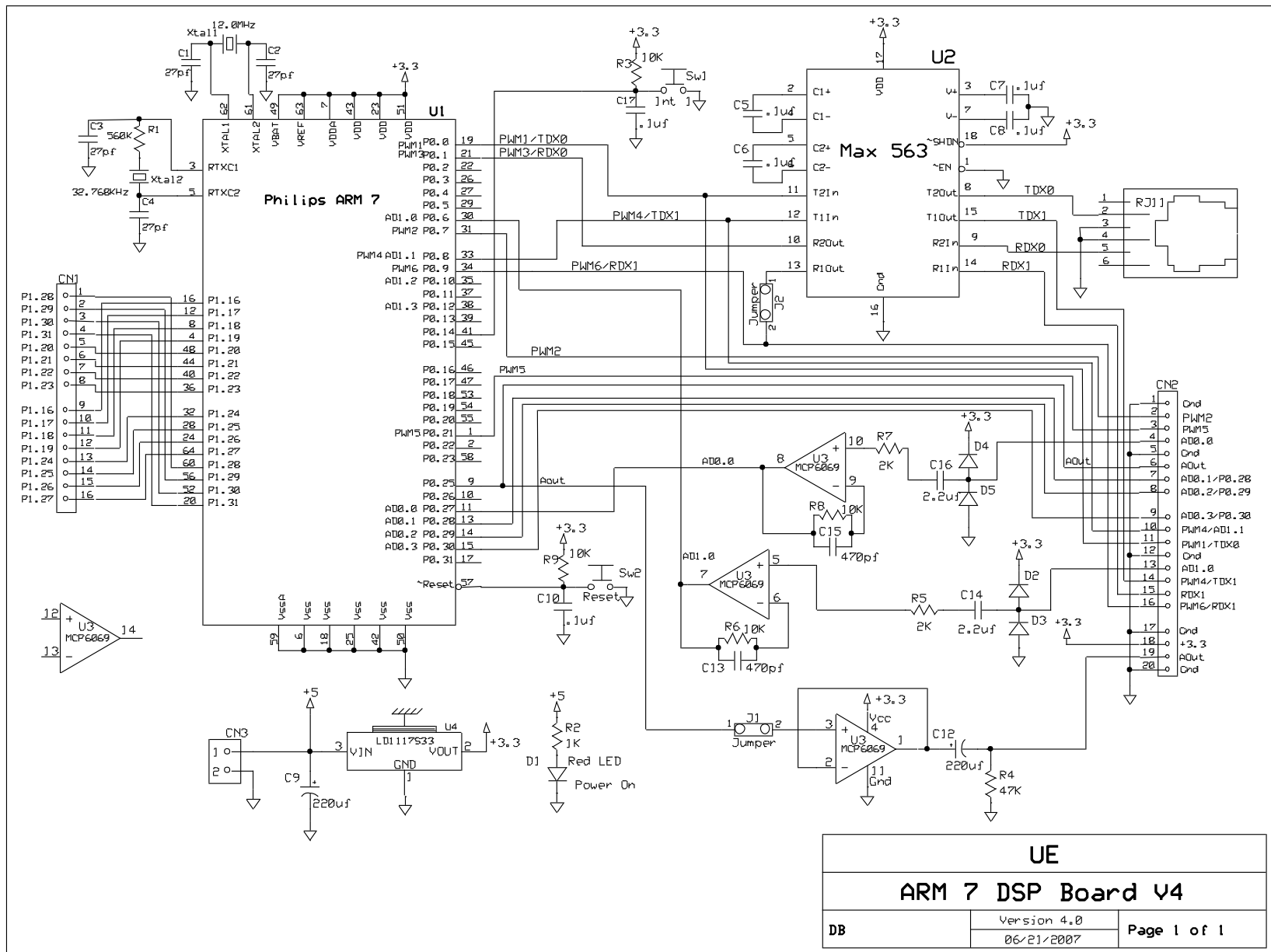


Figure 15: LPC2131/2132/2138 64-pin package



# EE 354 ARM 7 Board