UNIT II ARM PROCESSOR AND PERIPHERALS

ARM Architecture Versions – ARM Architecture – Instruction Set – Stacks and Subroutines – Features of the LPC 214X Family – Peripherals – The Timer Unit – Pulse Width Modulation Unit – UART – Block Diagram of ARM9 and ARM Cortex M3 MCU.

PART-A

1. **Draw and compare von-Neumann and Harvard architecture.**

Von Neumann Architecture

This kind of architecture consists of a single, shared memory for program and data. The computing system consist of a Central Processing Unit (CPU) and a memory. The CPU has several internal registers that store values used internally. One of these registers is the program counter (PC) which points to an instruction in memory. The memory holds data and instructions and can be read and written when given anaddress.

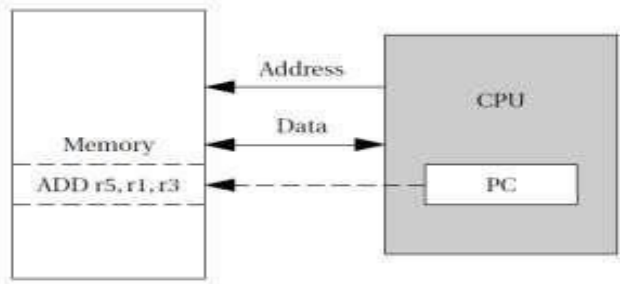


Fig. Von Neumann Architecture

Harvard Architecture

This Kind of Architecture consist of separate memory for data and Program. The program counter points to program memory not data memory. Harvard Architecture are widely used today for separation of program and data memories provides higher performance for digital signal processing.

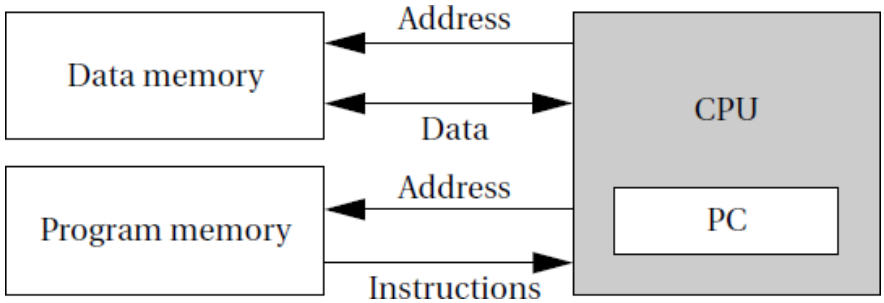


Fig. Harvard Architecture

2. In what ways CISC and RISC processors differ?

CISC	RISC
1. It provides number of addressing modes	It provides very few number of addressing modes
2. It has a micro programmed unit with a control memory	It has a hard wired unit without a control memory
3. An easy compiler design	Complex compiler design
4. Provides precise and intensive calculations slower than a RISC	Provides precise and intensive calculations faster than a RISC

3 Define Load-Store Architecture in ARM Programming. (Nov/Dec-2013)

ARM is a load-store architecture-data operands must first be loaded into the CPU and then stored back to main memory to save the results.

4. List some of the features of ARM processor.

- ARM processors have a good speed of execution to power consumption ratio
- They have a wide range of clock frequency ranging from 1 MHz to few GHz
- ARM processors have built in hardware for debugging
- Supports enhanced instructions for DSP operations

5. Define memory width.

The memory width is the number of bits the memory returns on each access typically 8,16,32 or 64 bits.
The memory width has a direct effort on the overall performance and cost ratio.

6. What is the use of Timer Control Register (TCR)?

- Timer Control Register is used to control the functions of timer/counter.It is used to enable/disable or reset the timer counter.Only the first two bits of the TCR must be used for these operations.
- When the first bit is 1, the counters are enabled.
- When the second bit is 1, both the counters are reset on the next positive edge of the peripheral clock

7. Write down the steps required for PWM generation.

- Reset and disable PWM counter using PWM TCR
- Load prescale value according to need of application in the PWMPR
- Load PWMMR0 with a value corresponding to the time period of your PWM wave
- Load any one of the remaining six match registers with the ON duration of the PWM cycle.

8. Why Cortex-M3 is needed in ARM processor?

- Delivering higher performance and richer features
- Performance and energy efficiency
- Full featured
- Rich connectivity

9. Difference between timer and counter.

Timer	Counter
The register incremented for every machine cycle	The register is incremented considering 1 to 0 transition at its corresponding to an external input pin
Maximum count rate is 1/12 of the oscillator frequency	Maximum count rate is 1/24 of the oscillator frequency
A timer uses the frequency of the internal clock and generates delay	A counter uses an external signal to count pulses.

10. List the functions of ARM processor in supervisor mode.
The supervisor mode has privileges that user modes do not control of memory management unit is reserved for supervisor mode to avoid the obvious problems that could occur when program bugs cause inadvertent changes in the memory management registers.

11. Define Subroutine.
A subroutine is a block of code that performs a task based on some arguments and optionally returns a result. By convention we use registers R0 to R3 to pass arguments to subroutines and R0 to pass a result back to the callers. A subroutine that requires more than four inputs uses the stack for the additional inputs.

12. Define UART.
UART (Universal Asynchronous Reciever/Transmitter) is one of the earliest mode of communication applied to computer. The information is transmitted one binary bit at a time as such it is a serial communication method.
UART is asynchronous because it doesn't require a transmitter provided clock to synchronize the transmission and receipt of data.

13. List the functions of PWM.
Pulse Width Modulation is a technique by which width of a pulse is varied while keeping the frequency constant. A period of a pulse consists of an ON cycle (HIGH) and an OFF cycle (LOW).The fraction for which the signal is ON over a period is known as duty cycle.

14.State any two data transfer and control flow instructions of ARM processor.
Data transfer instructions used to transfer data between registers and memory.
Memory to Register
Register to Memory
Control fow instructions are used to divert the flow of the program.
Branch instruction
Condition Codes

Conditional branch

15.Suggest the rules which apply to ARM data processing instructions.
Data processing instructions can process one of their operands using the barrel shifter
Data processing instructions are processed within the Arithmetic Logic Unit (ALU)
ARM Processor is the ability to shift the 32 bit binary pattern in one of the source registers let or right by a specific number of positions before it enters the ALU.

16.What is meant by conditional execution?

Conditional execution is an event used to controls whether or not the core will execute an instruction.Most instructions have a condition attribute that determines if the core will execute it based on the setting of the condition flags.

17.List the registers associated with timers in LPC2148.

- Prescale register(PR)
- Prescale counter Register (PC)
- Timer control Register (TCR)
- Timer counter Register (TC)
- Count control Register (CTCR)
- Match control Register (MCR)
- Interrupt Register(IR)

18.State the function of co processor .
Co processor is a computer processor used to supplement the functions of the primary processor.Operations performed by the co processor may be floating point arithmetic, graphics, signal processing, string processing.

19.Mention the Key features of LPC2148X

- 16 bit/32 bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
- 8KB to 40 KB of on-chip static RAM and 32 KB and 512 KB of on-chip flash memory.128-bit wide interface/accelerator enables high speed 60 MHZ operation.
- In system programming/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 B in 1 ms.
- Embedded ICE RT and Embedded Trace interfaces offer real time debugging with the on chip real monitor software and high speed tracing of instruction execution.

20.Difference between stack and stack pointer.
The difference between the stack and the stack pointer is that the stack is an area of memory, the stack pointer is the address of the last value pushed onto the stack.

21.Define peripheral devices.
Embedded systems that interact with the outside word means we 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.

22.Mention the instruction set in ARM processor

- Data processing instructions
- Branch instrunctions
- Load store instructios
- Software interrupt instructios and
- Program status register instructions

PART-B

1. Draw the architecture of an ARM 9 processor and explain its functional units.

ARM ARCHITECTURE

- ARM was an acronym for Advanced RISC Machine. ARM is a family of computer processor designed by Advanced RISC Machine (ARM) Limited company.
- The architectural simplicity of ARM processors has traditionally led to very small implementations, which allow devices with very low power consumption. Implementation size, performance, and very low power consumption remain key attributes in the development of the ARM architecture.
- ARM Processor are used for low-power and low-cost applications like Mobile phones, Communication modems, Automotive engine management systems and Hand-held digital systems.
- ARM architecture has been developed since 1980s and most widely used 32-bit instruction set architecture.

URES OF ARM

- ARM Processors are based on reduced instruction set computing (RISC) architecture.
- 32-bit Architecture but also supports 16 bits or 8 bits data types
- 32-bit processor register.
- 32-bit addresses.
- ARM Processors follow **Load and Store type architecture** where the data processing is performed only on the contents of the registers rather than directly on the memory. The instructions for data processing on registers are different from that access the memory.
- The **instruction set of ARM is uniform and fixed in length**. 32-bit ARM Processors have three instruction sets: general 32-bit ARM Instruction Set, 16- bit Thumb Instruction Set and Jazelle instruction set.
- ARM supports **multiple stages of pipeline** to speed up the flow of instructions. In a simple three stage pipeline, the instructions follow: fetch, decode and execute.
- **Memory is byte addressable.** Therefore, the word 0 in the ARM address space is at location 0, the word 1 is at location 4 and the word 2 is at the location 8 and so on, as a result the Program Counter (PC) is incremented by 4.

- **Both little-endian and big-endian memory addressing.** The ARM processor can be configured at power-up to address the bytes in a word in either **little-endian mode** (with the lowest-order byte residing in the lowest storage address) or **Big-endian mode** (with the lowest-order byte residing in the highest storage address).

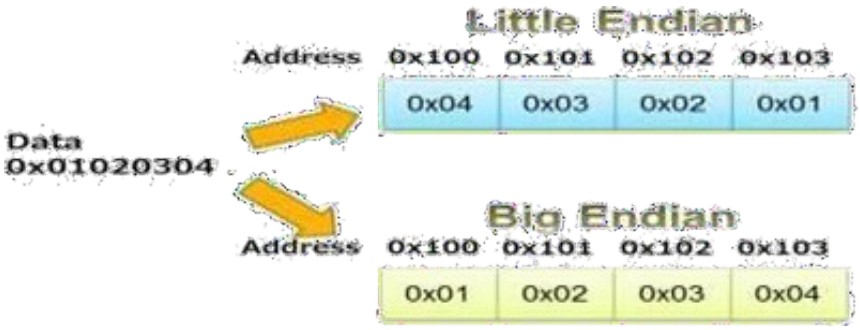


Figure 2.7 ARM Memory addressing

- ARM Processor used **AMBA (Advanced Microcontroller Bus Architecture) bus** interface. **AMBA** is an open source specification for on- chip interconnect specifications from ARM that Standardizes on chip communication mechanisms between various functional blocks for building high performance System on Chip (SOC) designs.
- These designs typically have one or more micro controllers or microprocessors along with several other components-internal memory or external memory bridge, DSP, DMA, accelerators and various other peripherals like USB, UART, PCIE, I2C etc-all integrated on a single chip.
- The primary motivation of AMBA protocols is to have a standard and efficient way to interconnecting these blocks with re-use across multiple designs.

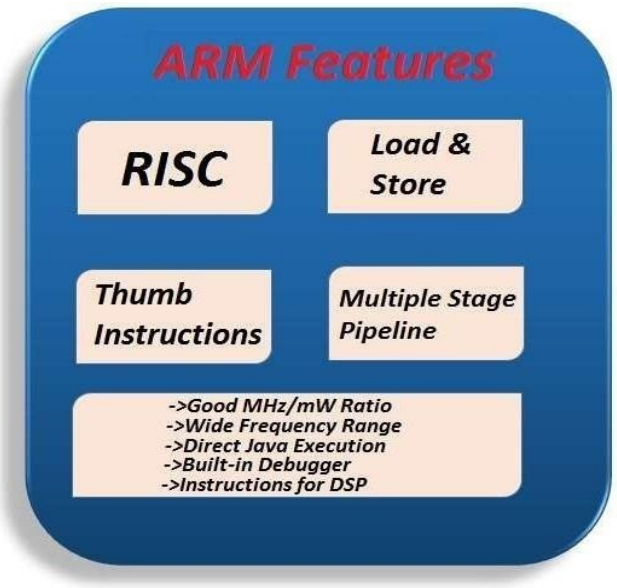


Figure 2.8 ARM Features

Some of the general features of ARM are listed here.

- ARM Processors have a good speed of execution to power consumption ratio.
- They have a wide range of clock frequency ranging from 1MHz to few GHz.
- They support direct execution of Java bytecodes using ARM's Java Jazelle DBX.
- ARM Processors have built in hardware for debugging.
- Supports enhanced instructions for DSP operations.

ARM ARCHITECTURE

ARM is a load-store reducing instruction set computer architecture; it means the core cannot directly operate with the memory. All data operations must be done by registers with the information which is located in the memory. Performing the operation of data and storing the value back to the memory. ARM consist of 37 register sets, 31 are general-purpose registers and 6 are status registers. The ARM uses seven processing modes that are used to run the user task.

THE ARM ARCHITECTURE PROFILES

The ARM architecture profiles are:

1. **Application profile (ARMv7-A e.g. Cortex-A8)**

- Application profiles implement a traditional ARM architecture with multiple modes and support a virtual memory system architecture based on an MMU. These profiles support both ARM and Thumb instruction sets.
- Features powerful processors found in high-end products like smartphones, tablets, or netbooks.

This includes the famous Cortex-A8 and Cortex-A9 (in your super phone) processors.

2. **Real-time profile (ARMv7-R e.g. Cortex-R4)**

- Real-time profiles im plement a traditional ARM architecture with multiple modes and support a protected memory system architecture based on an MPU.
- Can be found for example in control units for automotive systems or hard disk drive controllers.

They come with specific features suited to real-time environment constraints.

3. **Microcontroller profile (ARMv7-M e.g. Cortex-M3)**

- Microcontroller profiles implement a programmers' model designed for fast interrupt processing, with hardware stacking of registers and support for writing interrupt handlers in high-level languages. The processor is designed for integration into an FPGA and is ideal for use in very low power applications.

- They are smaller and used in numerous embedded systems like human interface devices, automotive control systems, power management systems, and others.

The ARM core is considered as a functional unit connected by data buses where,

- Arrow represents the flow of data
- Line represents the buses
- Boxes represents either an operation unit or storage area The functional

units of the ARM architecture are,

- Arithmetic and logic unit
- Booth multiplier
- Barrel shifter
- Control unit
- Register file

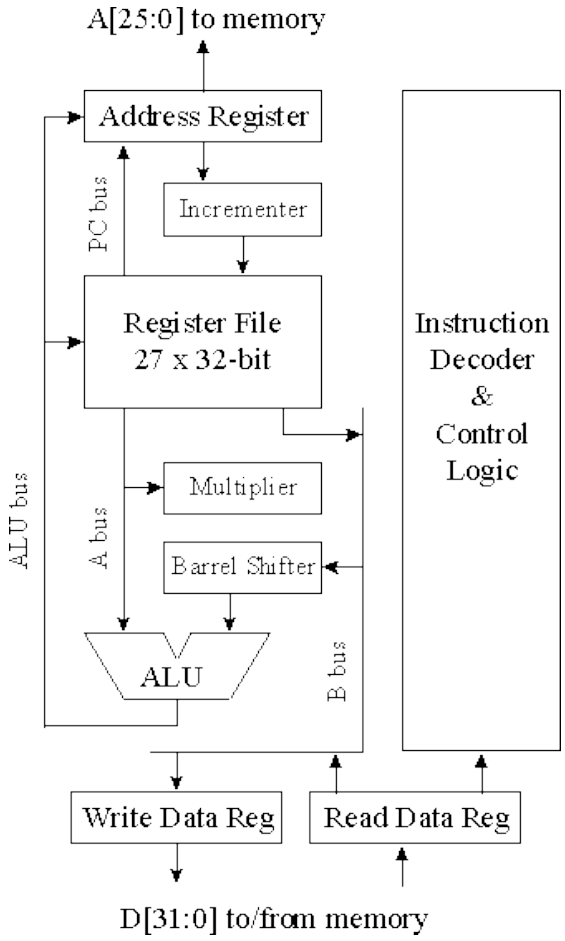


Figure 2.9 ARM Architecture

Priority encoder: The encoder is used in the multiple load and store instruction to point which register within the register file to be loaded or kept.

Multiplexers: Several multiplexers are accustomed to the management operation of the processor buses.

Arithmetic Logic Unit(ALU)

The ALU has two 32-bits inputs. The primary comes from the register file, whereas the other comes from the shifter. Status registers flags modified by the ALU outputs. The V-bit output goes to the V flag as well as the Count goes to the C flag. Whereas the foremost significant bit

really represents the S flag, the ALU output operation is done by NORed to get the Z flag. The ALU has a 4-bit function bus that perm its up to 16 opcode to be implemented.

Booth Algorithm

Booth algorithm is a noteworthy multiplication algorithm ic rule for 2's complement numbers. This treats positive and negative numbers uniform ly. Moreover, the runs of 0's or 1's within the multiplier factor are skipped over without any addition or subtraction being performed, thereby creating possible quicker multiplication.

Barrel Shifter

- The barrel shifter features a 32-bit input to be shifted. This input is coming back from the register file or it might be immediate data. The shifter has different control inputs coming back from the instruction register. The Shift field within the instruction controls the operation of the barrel shifter. This field indicates the kind of shift to be performed (logical left or right, arithmetic right or rotate right). The quantity by which the register ought to be shifted is contained in an immediate field within the instruction or it m ight be the lower 6 bits of a register within the register file.

The shift_val input bus is 6-bits, permitting up to 32bit shift. The shift type indicates the needed shift sort of 00, 01, 10, 11 are corresponding to shift left, shift right, an arithmetic shift right and rotate right, respectively. The barrel shifter is especially created with multiplexers.

Control Unit

For any microprocessor, control unit is the heart of the whole process and it is responsible for the system operation, so the control unit design is the most important part within the whole design. The control unit is sometimes a pure combinational circuit design. Here, the control unit is im plemented by easy state machine. The processor tim ing is additionally included within the control unit. Signals from the control unit are connected to each component within the processor to supervise its operation.

ARM REGISTERS

The amount of registers depends on the ARM version. ARM has 37 registers all of which are 32-bits long. 1 dedicated Program Counter (PC), 1 dedicated Current Program Status Register (CPSR), 5 dedicated Saved Program Status Registers (SPSR) and 30 General Purpose

Registers.

The first 16 registers are accessible in user-level mode, the additional registers are available in priv ileged software execution. These 16 registers can be split into two groups: general purpose and special purpose registers.

R0-R12: can be used during common operations to store temporary values, pointers (locations to memory), etc. R0, for example, can be referred as accumulator during the arithmetic operations or for storing the result of a previously called function. R7 becomes useful while working with syscalls as it stores the syscall number and R11 helps us to keep track of boundaries on the stack serv ing as the frame pointer. Moreover, the function calling convention on ARM specifies that the first four arguments of a function are stored in the registers r0-r3.

Register	Alias	Purpose
R0	–	General purpose
R1	–	General purpose
R2	–	General purpose
R3	–	General purpose
R4	–	General purpose
R5	–	General purpose
R6	–	General purpose
R7	–	Holds Syscall Number
R8	–	General purpose
R9	–	General purpose
R10	–	General purpose
R11	FP	Frame Pointer
Special Purpose Registers		
R12	IP	Intra Procedural Call
R13	SP	Stack Pointer
R14	LR	Link Register
R15	PC	Program Counter
CPSR	–	Current Program Status Register

Table 2.2
ARM
Registers

R13: SP (Stack Pointer): The Stack Pointer points to the top of the stack. The stack is an area of memory used for function-specific storage, which is reclaimed when the function returns.

R14: LR (Link Register): When a function call is made, the Link Register gets updated with a memory address referencing the next instruction where the function was initiated from. Doing this allows the program return to the "parent" function that initiated the "child" function call after the "child" function is finished.

R15: PC (Program Counter): The Program Counter is automatically incremented by the size of the instruction executed. This size is always 4 bytes in ARM state and 2 bytes in THUMB mode. When a branch instruction is being executed, the PC holds the destination address. During execution, PC stores the address of the current instruction plus 8 (two ARM instructions) in ARM state, and the current instruction plus 4 (two Thumb instructions) in Thumb(v1) state. This is different from x86 where PC always points to the next instruction to be executed.

Current Program Status Register (CPSR)

The Current Program Status Register (CPSR) holds:

- the APSR flags
- the current processor mode
- interrupt disable flags
- current processor state (ARM, Thumb, ThumbEE, orJazelle)
- endianness state (on ARMv4T andlater)
- execution state bits for the IT block (on ARMv6T2 and later).

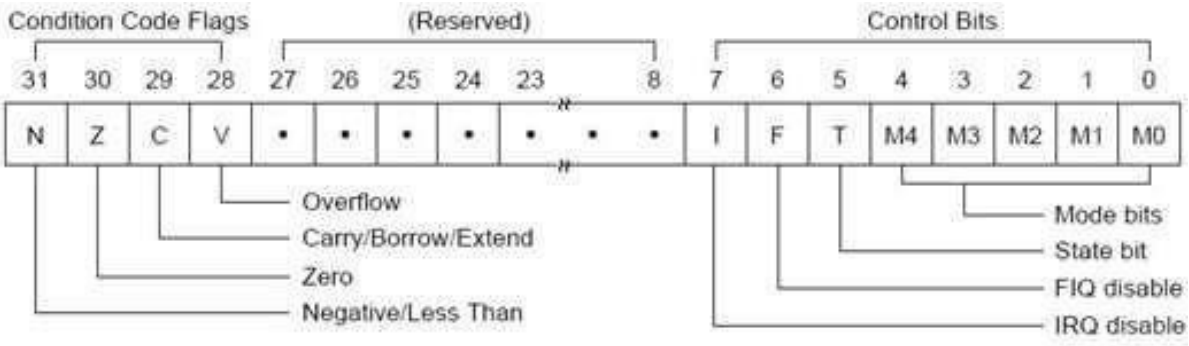


Figure 2.10 ARM CPSR

1. Condition Bits N

- If this result is regarded as a two's complement signed integer,
- then N = 1 If the result is negative and N = 0 if it is positive or zero.

Z

- Is set to 1 if the result of the instruction is zero and to 0 otherwise.
- This often indicates an equal result from a comparison.

C

Is set in one of four ways:

- For an addition, including the comparison instruction CMN, C is set to 1 if the addition produced a carry and to 0 otherwise.
- For a subtraction, including the comparison instruction CMP, C is set to 0 if the subtraction produced a borrow (that is, an unsigned underflow), and to 1 otherwise.
- For non-addition/subtractions that incorporate a shift operation, C is set to the last bit shifted out of the value by the shifter.
- For other non-addition/subtractions, C is normally left unchanged

V

Is set in one of two ways:

- For an addition or subtraction, V is set to 1 if signed overflow occurred, regarding the operands and result as two's complement signed integers.
- For non-addition/subtractions, V is normally left unchanged.

2. Interrupt bit

I - Disables IRQ interrupts when it is set. F - Disables FIQ interrupts when it is set.

3. Thumb Mode Bit

T - Thumb mode

4. Mode Bits

5 bits that control what mode the CPU is in.

M [4:0]	Mode
10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

Table 2.3 ARM Mode bits

Saved Program Status Register (SPSR)

- The SPSR is used to store the current value of the CPSR when an exception is taken so that it can be restored after handling the exception. Each exception handling mode can access its own SPSR. User mode and System mode do not have an SPSR because they are not exception handling modes.

- The execution state bits, endianness state and current processor state can be accessed from the SPSR in any exception mode, using the MSR and MRS instruction.

MODES OF OPERATION OF ARM PROCESSOR

The ARM uses seven processing modes that are used to run the user task.

- USER mode
- FIQ mode and IRQ mode
- SVC mode
- UNDEFINED mode
- ABORT mode
- THUMB mode

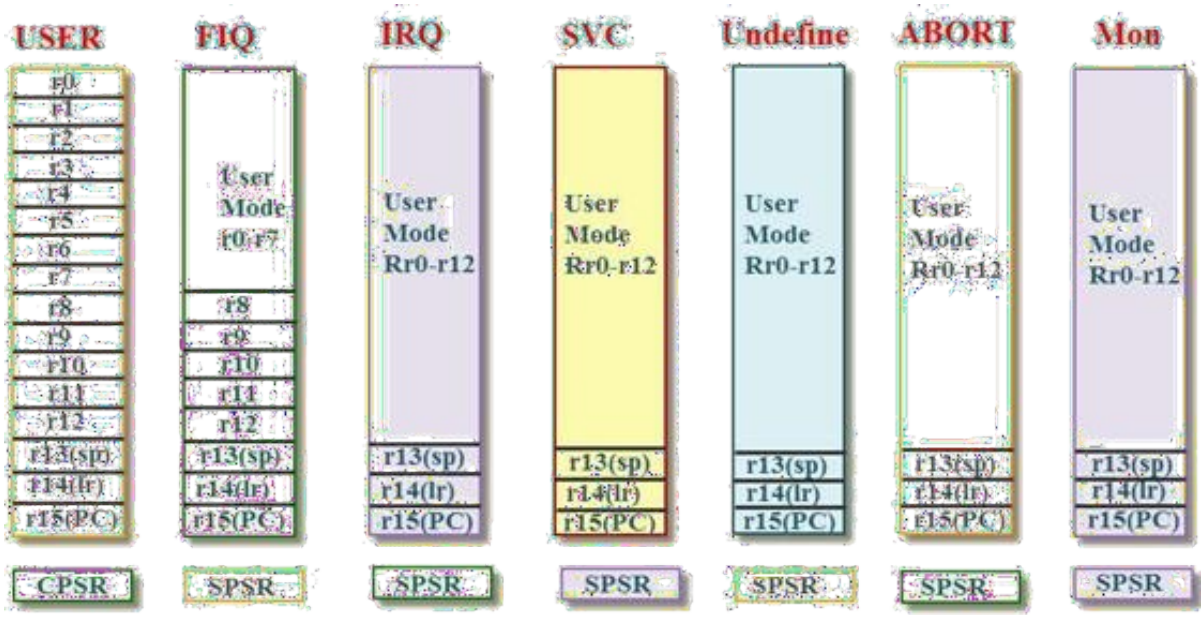


Figure 2.11 ARM Registers

1. **USER Mode:** The user mode is a normal mode, which has the least number of registers. It doesn't have SPSR and has limited access to the CPSR.

2. **FIQ and IRQ:** The FIQ and IRQ are the two interrupt caused modes of the CPU. The FIQ is processing interrupt and IRQ is standard interrupt. The FIQ mode has additional five banked registers to provide more flexibility and high performance when critical interrupts arehandled.

3. **SVC Mode:** The Superv isor mode is the software interrupt mode of the processor to start up or reset.

4. **Undefined Mode:** The Undefined mode traps when illegal instructions are executed. The ARM core consists of 32-bit data bus and faster data flow.

5. **THUMB Mode:** In THUMB mode 32-bit data is div ided into 16-bits and increases the processing speed.

6. **THUMB-2 Mode:** In THUMB-2 mode the instructions can be either 16-bit or 32-bit and it increases the performance of the ARM cortex –M3 microcontroller. The ARM cortex-m3 microcontroller uses only THUMB-2 instructions.

ARM INSTRUCTIONSET

- An Instruction Set Architecture (ISA) is part of the abstract model of a computer. It defines how software controls the CPU.

- The Arm ISA family allows developers to write software and firm ware that conforms to the Arm specifications, secure in the knowledge that any Arm- based processor will execute it in the same way. This is the foundation of the Arm portability and compatibility promise, underlying the Arm ecosystem.

2. Explain the operation of the BL instruction, including the state of ARM registers before and after its operation.

ARM INSTRUCTION SET ARCHITECTURE

- ARM processors have two main states they can operate in **ARM and Thumb**. The main difference between these two states is the instruction set, where instructions in ARM state are always 32-bit, and instructions in Thumb state are 16-bit (but can be 32-bit).
- There are different Thumb versions. The different nam ing is just for the sake of differentiating them from each other (the processor itself will always refer to it as Thumb).

  - **Thumb-1** (16-bit instructions): was used in ARMv6 and earlier architectures.
  - **Thumb-2** (16-bit and 32-bit instructions): extents Thumb-1 by adding more instructions and allowing them to be either 16-bit or 32-bit wide (ARMv6T2, ARMv7).
  - **ThumbEE:** includes some changes and additions aimed for dynamically generated code (code compiled on the device either shortly before or during execution).

ASSEMBLY LANGUAGE IN ARM

Assembly language is composed of instructions which are the main building blocks. ARM instructions are usually followed by one or two operands and generally use the following template:

MNEMONIC{S}{condition} {Rd}, Operand1, Operand2

Due to flex ibility of the ARM instruction set, not all instructions use all of the fields prov ided in the template. Nevertheless, the purpose of fields in the template are described as follows:

- MNEMONIC - Short name (mnemonic) of the instruction
- {S} - An optional suffix. If S is specified, the condition flags are updated on the result of the operation
- {condition}- Condition that is needed to be met in order for the instruction to be executed
- {Rd}- Register (destination) for storing the result of the

instruction. Operand1 - First operand, Either a register or an

immediate value

- Operand2- Second (flex ible) operand. Can be an immediate value (number) or a register with an optional shift

While the MNEMONIC, S, Rd and Operand1 fields are straight forward, the condition and Operand2 fields require a bit more clarification. The condition field is closely tied to the CPSR register's value, or to be precise, values of specific bits within the register. Operand2 is called a flex ible operand, because we can use it in various forms – as immediate value (with lim ited set of values), register or register with a shift.

1. DATA INSTRUCTION

The basic form of a data instruction is simple:

ADD r0,r1,r2

This instruction sets register r0 to the sum of the values stored in r1 and r2.

In addition to specify ing registers as sources for operands, instructions may also prov ide immediate operands, which encode a constant value directly in the instruction. For example,

ADD r0,r1,#2

sets r0 to r1 + 2.

2. ARITHMETIC INSTRUCTION

The arithmetic operations perform addition and subtraction; the with-carry versions include the current value of the carry bit in the computation.

RSB performs a subtraction with the order of the two operands reversed, so

that RSB r0,r1,r2 sets r0 to be r2 – r1.

ADD	Add
ADC	Add with carry
SUB	Subtract
SBC	Subtract with carry
RSB	Reverse subtract
RSC	Reverse subtract with carry
MUL	Multiply
MLA	Multiply and accumulate

Table 2.4 ARM arithmetic instruction

3. LOGICAL INSTRUCTION

The bit-wise logical operations perform logical AND, OR, and XOR operations (the exclusive or is called EOR).

The **BIC** instruction stands for bit clear: BIC r0,r1,r2 sets r0 to r1 and not r2. This instruction uses the second source operand as a mask: Where a bit in the mask is 1, the corresponding bit in the first source operand is cleared. The MUL instruction multiplies two values, but with some restrictions: No operand may be an immediate, and the two source operands must be different registers.

The **MLA** instruction performs a multiply-accumulate operation, particularly useful in matrix operations and signal processing.

The instruction MLA r0,r1,r2,r3 sets r0 to the value r1 # r2 + r3.

AND	Bit-wise and
ORR	Bit-wise or
EOR	Bit-wise exclusive-or
BIC	Bit clear

Table 2.5 ARM Logical instruction

4. SHIFT INSTRUCTIONS

The shift operations are not separate instructions rather, shifts can be applied to arithmetic and logical instructions. The shift modifier is always applied to the second source operand.

A left shift moves bits up toward the most-significant bits, while a right shift moves bits down to the least-significant bit in the word.

The **LSL and LSR** modifiers perform left and right logical shifts, filling the least-significant bits of the operand with zeroes.The arithmetic shift left is equivalent to an LSL, but the ASR copies the sign bit, if the sign is 0, a 0 is copied, while if the sign is 1, a 1 is copied.

LSL	Logical shift left (zero fill)
LSR	Logical shift right (zero fill)
ASL	Arithmetic shift left
ASR	Arithmetic shift right
ROR	Rotate right
RRX	Rotate right extended with C

Table 2.6 ARM shift instructions

5. ROTATE INSTRUCTIONS

The rotate modifiers always rotate right, moving the bits that fall off the least-significant bit up to the most-significant bit in the word.

The **RRX** modifier performs a 33-bit rotate, with the CPSR's C bit being inserted above the sign bit of the word; this allows the carry bit to be included in the rotation.

6. COMPARE INSTRUCTIONS

- Comparison operands do not modify general purpose registers but only set the values of the NZCV bits of the CPSR register.
- The compare instruction CMP r0, r1 computes r0 – r1, sets the status bits, and throws away the result of the subtraction.

- CMN uses an addition to set the status bits.
- TST performs a bit-wise AND on the operands, while TEQ performs an exclusive-or.

Table 2.7 ARM Compare instructions

CMP	Compare
CMN	Negated compare

TST	Bit-wise test
TEQ	Bit-wise negated test

7. MOVE INSTRUCTION

The instruction MOV r0,r1 sets the value of r0 to the current value of r1.

The MVN instruction complements the operand bits (one's complement) during the move.

MOV	Move
MVN	Move negated

Table 2.8 ARM Move instructions LOAD AND STOREINSTRUCTION

LDRB and STRB load and store bytes rather than whole words. LDRH and

SDRH operate on half-words.

LDRSH extends the sign bit on loading.

An ARM address may be 32 bits long. The ARM load and store instructions do not directly refer to main memory addresses, because a 32-bit address would not fit into an instruction that included an opcode and operands. Instead, the ARM uses register-indirect addressing. In register-indirect addressing, the value stored in the register is used as the address to be fetched from memory; the result of that fetch is the desired operand value.

LDR	Load
STR	Store
LDRH	Load half-word
STRH	Store half-word
LDRSH	Load half-word signed
LDRB	Load byte
STRB	Store byte
ADR	Set register to address

Table 2.9. ARM Load and Store instructions

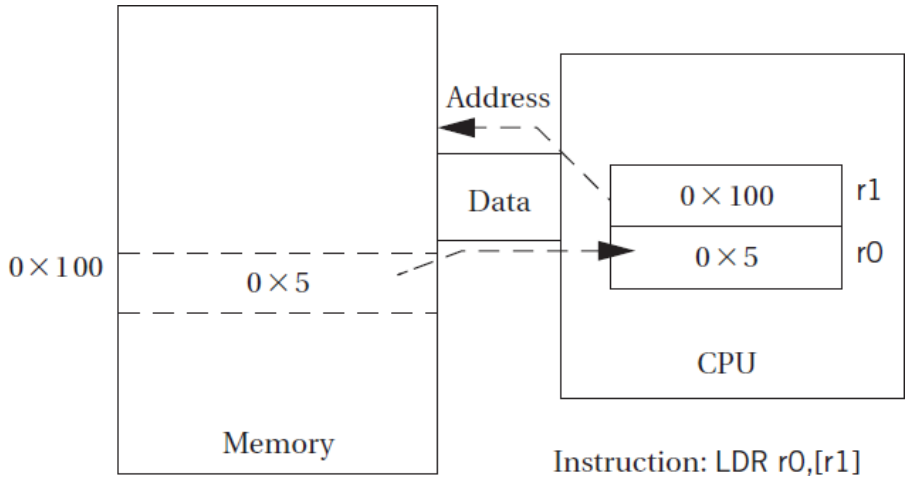


Figure 2.12 Register indirect addressing in the ARM

8. CONDITIONAL INSTRUCTION

The B (branch) instruction is the basic mechanism in ARM for changing the flow of control.

EQ	Equals zero Z=1
NE	Not equal to zero Z=0
CS	Carry set C=1
CC	Carry clear C=0
MI	Minus N=1
PL	Nonnegative (plus) N=0
VS	Overflow V=1
VC	No overflow V=0
HI	Unsigned higher C=1 and Z=0
LS	Unsigned lower or same C=0 or Z=1
GE	Signed greater than or equal N=V
LT	Signed less than N=V
GT	Signed greater than Z=0 and N=V
LE	Signed less than or equal Z=1 or N=V

Table 2.10 ARM Conditional instructions

Instructions are branched conditionally, based on the result of a given computation. The if statement is a common example. The ARM allows any instruction, including branches, to be executed conditionally. This allows branches to be conditional, as well as data operations.

3. With an example discuss about the stacks and subroutines used in ARM processor.

STACKS AND SUBROUTINES

1. SUBROUTINES

Large programs are hard to handle and so broken into smaller programs called as subroutines. A subroutine is a block of code that is called from different places from within a main program or other subroutines.

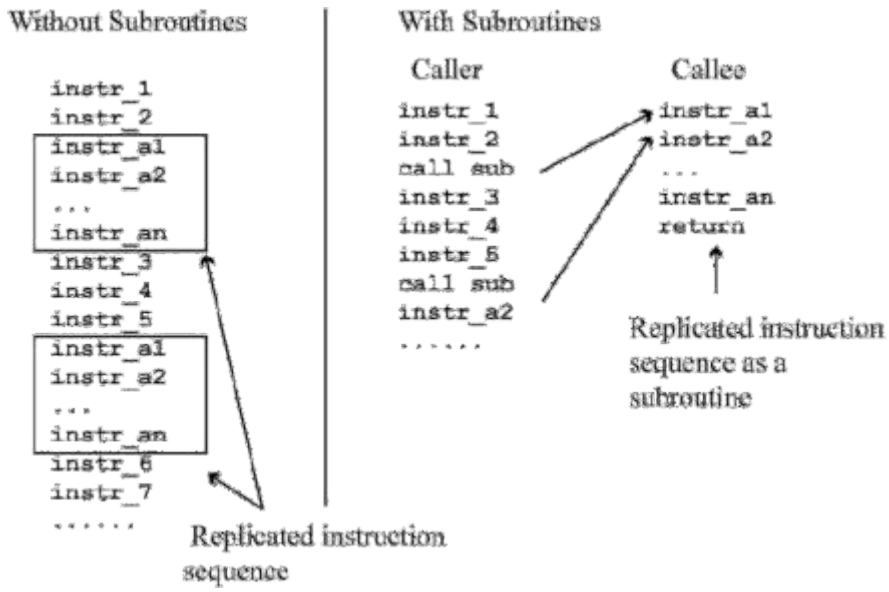


Figure 2.13 ARM subroutine.

A subroutine can have

- parameters that control its operation
- local variables for computation

Only the code for the subroutine call is repeated.

A subroutine may pass a return value back to the caller.

WRITING SUBROUTINES

When using subroutines, it is necessary to know the following:

- When should we jump? (use CALL)- A subroutine call can be implemented by pushing the return address on the stack and then jumping to the branch target address.
- Where do we return to? (use RETURN)- When the subroutine is done, remember to pop out the saved information so that it will be able to return to the next instruction immediately after the calling point.

Subroutines are based on MPU instructions and use STACK

A **Branch and Link** (BL) instruction is used to call a subroutine or a procedure in ARM. For instance,BL foo /*BL- Branch and link instruction, foo is a subroutine/procedure name*/ will perform a branch and link to the code starting at location.

The branch and link is much like a branch, except that before branching it stores the current PC value in r14. Thus, to return from a procedure, simply move the value of r14 (LR) to r15(PC)

MOV r15,r14

When subroutines are nested, the contents of the link register must be saved on a stack by the subroutine. Register R13, Stack Pointer is normally used as the pointer for this stack But this mechanism only lets us call procedures one level deep.

If, for example, we call a C function within another C function, the second function call will overwrite r14, destroying the return address for the first function call. The standard procedure for allowing nested procedure calls (including recursive procedure calls) is to build a stack, as illustrated in Figure 2.14. The C code shows a series of functions that call other functions: f1() calls f2(), which in turn calls f3(). The right side of the figure shows the state of the procedure call stack during the execution of f3(). The stack contains one activation record for each active procedure. When f3() finishes, it can pop the top of the stack to get its return address, leaving the return address for f2() waiting at the top of the stack for its return.

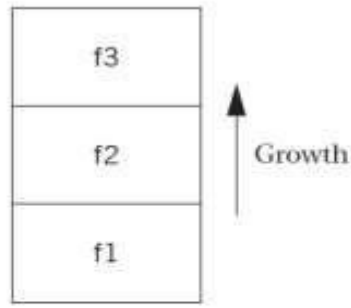
```
void f1(int a) {
    f2(a);
}

void f2(int r) {
    f3(r,5);
}

void f3(int x, int y) {
    g = x + y;
}

main() {
    f1(xyz);
}
```

C code



Function call stack

We can also use the procedure call stack to pass parameters. The conventions used to pass values into and out of procedures are known as **procedure linkage**.

To pass parameters into a procedure, the values can be pushed onto the stack just before the procedure call. Once the procedure returns, those values must be popped off the stack by the caller, because they may hide a return address or other useful information on the stack.

A procedure may also need to save register values for registers it modifies. The registers can be pushed onto the stack upon entry to the procedure and popped off the stack, restoring the previous values, before returning.

- Procedure stacks are typically built to grow down from high addresses.
- Assembly language programmers can use any means they want to pass parameters. Compilers use standard mechanisms to ensure that any function may call any other. The compiler passes parameters and return variables in a block of memory known as a **frame**. The frame is also used to allocate local variables. The stack elements are frames.
- A stack pointer (sp) defines the end of the current frame, while a frame pointer (fp) defines the end of the last frame. (The fp is technically necessary only if the stack frame can be grown by the procedure during execution).
- The procedure can refer to an element in the frame by addressing relative to sp.
- When a new procedure is called, the sp and fp are modified to push another frame onto the stack.
- The ARM Procedure Call Standard (APCS) is a good illustration of a typical procedure linkage mechanism. Although the stack frames are in main memory, understanding how registers are used is key to understanding the mechanism, as explained below.

- r0-r3 are used to pass the first four parameters into the procedure. r0 is also used to hold the return value. If more than four parameters are required, they are put on the stack frame.
- r4-r7 hold register variables.
- r11 is the frame pointer and r13 is the stack pointer.
- r10 holds the limiting address on stack size, which is used to check for stack overflows.

Other registers have additional uses in the protocol.

Example: Procedure Calls in ARM

Here is a simple example of two procedures, one of which calls another:

```
void f2(int x) {
int y;
y = x+1;
}
void f1(int a) { f2(a);
}
```

This function has only one parameter, so x will be passed in r0. The variable y is local to the procedure so it is put into the stack. The first part of the procedure sets up registers to manipulate the stack, then the procedure body is implemented.

2.4.2 STACK

Stack is a Temporary memory storage space by MPU during the execution of a program.

The stack is a data structure, known as last in first out (LIFO). In a stack, items entered at one end and leave in the reversedorder.

Stacks in m icroprocessors are implemented by using register called the stack pointer, sim ilar to the program counter (PC), to keep track of available stack locations. As items are added to the stack (pushed), the stack pointer is moving up, and as items are removed from the stack (pulled or popped),
the stack pointer is moved down.

Instructions to Store and Retrieve Information from the Stack

PUSH: Increment the memory address in the stack pointer (by one) and stores the contents of the Program counter on the top of the stack

POP: Discards the address of the top of the stack and decrement the stack pointer by one

STACK TYPES

ARM stacks are very flexible since the implementation is completely left to the software. Stack pointer is a register that points to the top of the stack. Normally, there are four different stack implementations depending on which way the stack grows.

1. Ascending stack

An Ascending stack grows upwards. It starts from a low memory address

and, as items are pushed onto it, progresses to higher memory addresses.

2. Descending stack

A Descending stack grows downwards. It starts from a high memory address, and as items are pushed

onto it, progresses to lower memory addresses. The previous examples have been of a Descending stack.

3. Empty stack

In an Empty stack, the stack pointers points to the next free (empty) location on the stack, i.e. the place where the next item to be pushed onto the stack will be stored.

4. Full stack

In a Full stack, the stack pointer points to the topmost item in the stack, i.e. the location of the last item to be pushed onto the stack.

4. Explain the peripherals for LPC214X family in ARM processor.

LPC2148 PROCESSOR

It is an ARM7 based processor with ARM7TDMI-S Processor core. It is based on ARMv4 architecture and the significant changes from its previous architecture is the introduction of the 16-bit Thumb instructions.

LPC2148 is manufactured by NXP Semiconductor (Phillips) and it is preloaded with many in-built features and peripherals. This makes it more efficient and reliable choice for an high-end application developer.

The ARM7 is a 32-bit general-purpose microprocessor, and it offers some of the features like little power utilization, and high performance. The architecture of an ARM is depended on the principles of RISC. The associated decode mechanism, as well as the RISC- instructions set are much easy when we compare with microprogrammed CISC-Complex Instruction Set Computers.

The Pipeline method is used for processing all the blocks in architecture. In general, a single

instruction set is being performed, then its descendant is being translated, & a 3rd-instruction is being obtained from the memory.

An exclusive architectural plan of ARM7 is called as Thumb, and it is perfectly suitable for high volume applications where the compactness of code is a matter The ARM7 also uses an exclusive architecture namely Thumb. It makes it perfectly suitable for different applications by memory limitations where the density of code is a matter.

FEATURES OF LPC2148

The main features of LPC2148 include the following.

- The LPC2148 is a 16 bit or 32 bit ARM7 family based microcontroller available in a small LQFP64 package.
- ISP (in system programming) or IAP (in application programming) using on-chip boot loader software.
- On-chip static RAM is 8 kB-40 kB, on-chip flash memory is 32 kB-512 kB, the wide interface is 128 bit, or accelerator allows 60 MHz high-speed operation.
- It takes 400 milli seconds time for erasing the data in fullchip and 1 millisecond time for 256 bytes of programming.
- 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 has 2 kB of endpoint RAM and USB 2.0 fullspeed device controller. Furthermore, this microcontroller offers 8kB on-chip RAM nearby to USB with DMA.
- One or two 10-bit ADCs offer 6 or 14 analog i/ps with low conversion time as 2.44 μs/ channel.
- Only 10 bit DAC offers changeable analog o/p.
- External event counter/32 bit timers-2, PWM unit, & watchdog.
- Low power RTC (real time clock) & 32 kHz clockinput.

- Several serial interfaces like two 16C550 UARTs, two I2C-buses with 400 kbit/s speed.
- 5 volts tolerant quick general purpose Input/output pins in a small LQFP64 package.
- Outside interrupt pins-21.
- 60 MHz of utmost CPU CLK-clock obtainable from the programmable-on-chip phase locked loop by resolving time is 100 µs.
- The incorporated oscillator on the chip will work by an exterior crystal that ranges from 1 MHz-25 MHz
- The modes for power-conserving mainly comprise idle & powerdown.
- For extra power optimization, there are individual enable or disable of peripheral functions and peripheral CLK scaling.

5. EXPLAIN THE ARCHITECTURE BLOCK DIAGRAM OF LPC2148

LPC 2148 microcontroller consist of three buses such as ARM7 Local bus, AHB (Advanced high performance bus) and VPB bus etc. these buses are used for performing different function and these are also consisting of different functioning parts such as,

EC

1. M

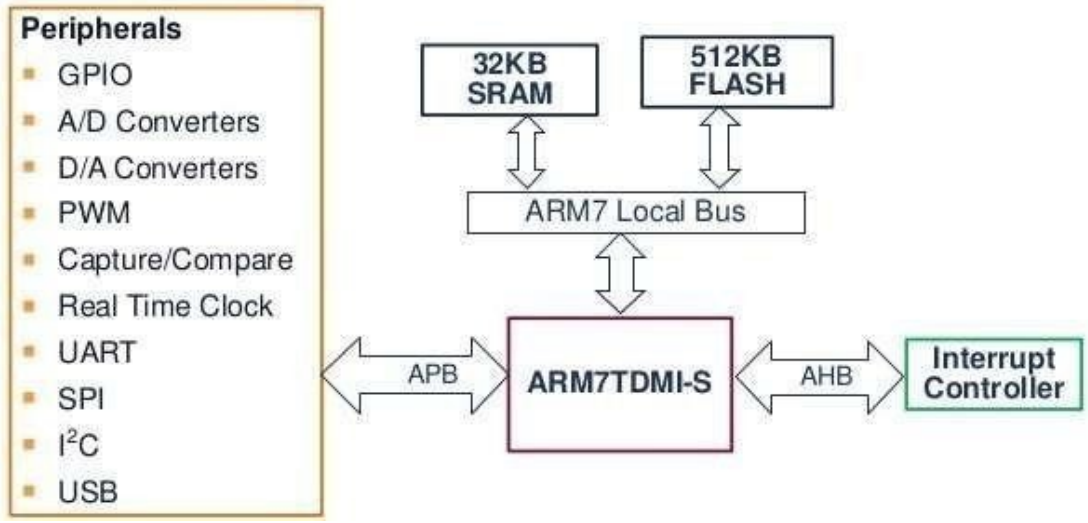
LPC ilt in

supp t al

appl

FL y be

use ays



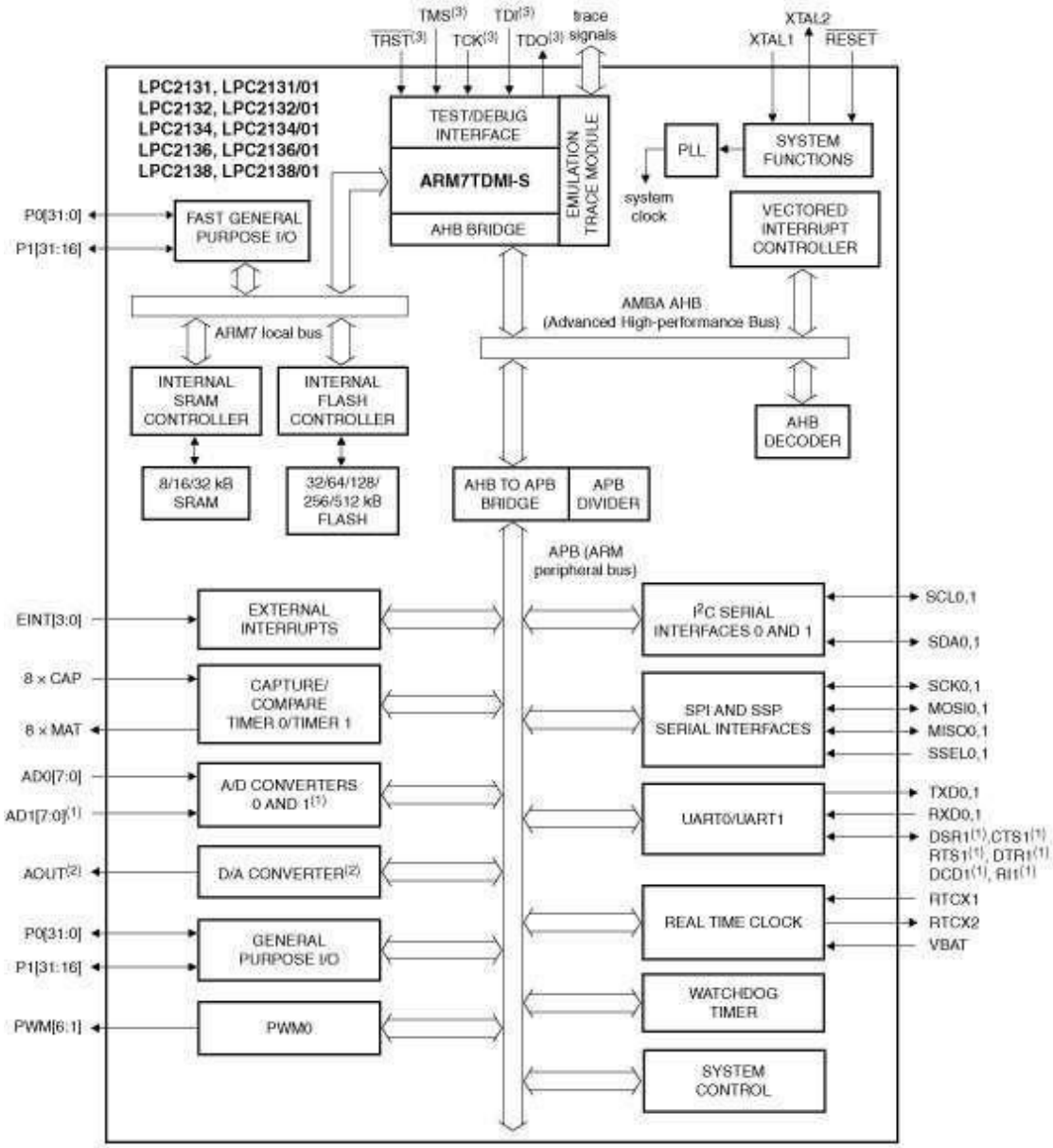
- Using serial built in JTAGInterface
- Using In-System Programming (ISP)
- By means of In-Application Programming (IAP) capabilities

The application program, using IAP functions may also erase and/or program the FLASH while the application is running. When the LPC2148 on chip bootloader is used, 500kB of flash memory is available for user code.

Static RAM Memory System: LPC2148 prov ides 32kB of static RAM which may be used for code and/or data storage. It may be accessed as 8-bit, 16-bit and 32-bits.

Interrupt sources

Every peripheral dev ice consists of a single interrupt line allied to the VIC (vector interrupt controller. All input requests are received by vectored interrupt controller (VIC) and it converts them into fast interrupt request (FIQ). So, fast interrupt request and non-fast interrupt requests are defined by programm ing setting in vectored interrupt contro ller.



Pin Connect Block

This block perm its chosen pins of the ARM7 based LPC2148 m icrocontroller for having several functions. The multiplexers can be controlled by the configuration registers for allowing the link between the pin as well as on-chip peripherals. Peripherals must be coupled with the suitable pins prev ious to being triggered, and prev ious to any connected interrupts being perm itted. The microcontroller functionality can be defined by the pin control module by its pin selection of registers in a given hardware env ironment. After rearranging all pins of ports (port 0 & port 1) are arranged as i/p by the given exceptions. If debug is allowed, the pins of the JTAG will guess the functionality of JTAG. If a trace is allowed, then the Trace pins will guess the functionality of trace. The pins connected to the I2C0 and I2C1 pins are open drain.

PERIPHERALS

GPIO (General Purpose Input Output)

ARM based LPC2148 m icrocontroller has 45 general purpose input output pins. The operating voltage of these input output pins is 5 volts.

GPIO registers control the dev ice pins which are not linked to a particular periphera l function. The device pins can be arranged as i/p or o/p. Indiv idual registers allow for clearing any number of o/p's concurrently. The output register value can be read back, & the present condition of the port pins.

LPC2148 has two IO ports each of 32-bit wide, prov ided by 64 IO pins. Ports are named as P0 and P1. Pins of each port labelled as Px.y where "x" stands for port number, 0 or 1. Where "y" stands for pin number usually between 0 to 31. Each pin can perform multiple functions. For example: Pin no.1 which is P0.21 serves as GPIO as well as PWM5, AD1.6 (A/D converter1, input 6), CAP1.3 (Capture input for Timer1, Channel 3).

Digital to analog Converter:

This LPC2148 m icrocontroller has one 10 bit digital to analog converter (DAC). This converter converts the digital input into analog output. The maximum DAC output voltages are called VREF voltages. Power down mode and buffered output is also available in this digital to analog converter.

10-bit ADC (Analog to Digital Converter)

The microcontrollers like LPC2144/46/48 include two ADC converters ADC0 and ADC1, and are only 10-bit straight approx imation ADC's. Although ADC0 includes 6- channels and ADC1 has 8- channels.

10-bit DAC (Digital to Analog Converter)

This LPC2148 m icrocontroller has one 10 bit digital to analog converter (DAC). This converter converts the digital input into analog output. The maximum DAC output voltages are called VREF voltages. Power down mode and buffered output is also available in this digital to analog converter.

Device Controller-USB 2.0

The universal serial bus consists of 4-wires, and that gives the support for communication between a number of peripherals and hosts. This controller allows the bandwidth of USB for connecting devices using a protocol based on thetoken.

The bus supports unplugging hot plugging and dynamic collection of the devices. Every communication is started through the host-controller. These microcontrollers are designed with a universal serial bus apparatus controller that allows 12 Mbit/sec data replaced by a host controller of USB.

UARTs

LPC2148 include two UARTs whose name are UART 0 and UART 01 for standard transmit & get data- lines. This LPC2148 m icrocontroller contains two UART whose name are UART 0 and UART 01. These UARTs are provided the full mode control handshake interface during transm itting or receiv ing the data lines. These are used 16 Byte data rate during transmitting or receiv ing the data. For covering wide range of baud rate, they also contain the built-in functional baud rate generator, therefore there is no need of any external crystal of specific value.

Serial I/O Controller of I2C-bus

LPC2148 includes two I2C bus controllers, and this is bidirectional. The inter-IC control can be done with the help of two wires namely an SCL and SDA. Here the SDA & SCL are serial clock line and the serial data line.

Every apparatus is identified by an indiv idual address. Here, transmitters and receivers can work in two modes like master mode/slave mode. This is a multi- master bus, and it can be managed by one or more bus masters linked to it. These microcontrollers support up to-400

kbit/s bit rates.

SPI Serial Input/Output Controller

These microcontrollers include a single SPI controller and intended to handle numerous masters & slaves associated with a specified bus.

Simply a master & a slave can converse over the interface throughout specified data transmission. During this, the master constantly transmits a byte-of-data toward the slave, as well as the slave constantly transmits data toward the master.

SSP Serial Input/Output Controller

These microcontrollers contain single SSP, and this controller is capable of process on an SPI, Microwire bus or 4-wire SSI. It can communicate with the bus of several masters as well as slaves

But, simply a particular master, as well as slave, can converse on the bus throughout a specified data transm it. This m icrocontroller supports full-duplex transfers, by 4-16 bits data frames used for the flow of data from the master- the slave as well as from the slave-the master.

Timers/Counters

Timers and counters are designed for counting the PCLK (peripheral clock) cycles & optionally produce interrupts based on 4-match registers.

This LPC2148 microcontroller has two timers or counters. These timers are 32 bit and are programmable with 32bit pre scaler value as well as it also has one externa l event counter. Each timer has four 32bit capture channels which take the snapshot of timer value during the transition of any input signal. W ith the help of this capture event the interruption could be also generate.

Watchdog Timer

LPC2148 microcontroller contains watchdog timer is used for resetting the microcontroller in a reasonable sum of time. When it is allowed then the timer will produce a reset of a system if the consumer program does not succeed to reload the timer in a fixed sum of time.

RTC-Real-time Clock

The RTC in LPC2148 is intended for providing counters to calculate the time when the idle

or normal operating method is chosen. The RTC uses a small amount of power and designed for appropriate battery power-driven arrangements where the central processing unit is not functioning constantly.

Power Control

These microcontrollers support two condensed power modes such as power-down mode and idle mode. In Idle mode, instructions execution is balanced until an interrupt or RST occurs. The functions of peripheral maintain operation throughout idle mode & can produce interrupts to cause the CPU to restart finishing. Idle mode removes the power utilized by the CPU, controllers, memory systems, and inner buses.

In power down mode, the oscillator is deactivated and the IC gets no inner clocks. The peripheral registers, processor condition with registers, inner SRAM values are conserved during Power-down mode & the chip logic levels output pins stayfixed.

This mode can be finished and the common process restarted by specific interrupts that are capable to work without clocks. Because the chip operation is balanced, Power-down mode decreases chip power utilization to almost zero.

5. Explain the concept of PWM -Pulse Width Modulator

The PWMs are based on the normal timer-block & also come into all the features, though simply the pulse width modulator function is fixed out on the microcontrollers like LPC2141/42/44/46/48.

The timer is intended to calculate PCLK (peripheral clock) cycles & optionally produce interrupts when particular timer values arise based on 7-match registers, and PWM function also depends on match register events.

The capability of indiv idually control increasing & decreasing boundary positions allows the pulse width modulation to be utilized for several applications. For example, the typical motor control with multi-phase uses 3-non-overlapping outputs of PWM by separate control of every pulse widths as well as
positions.

VPB Bus

The VPB div ider resolves the association between the CCLK (processor clock) and the PCLK (clock used by peripheral dev ices). This div ider is used for two purposes. The first use is to supply peripherals by the preferred PCLK using VPB bus so that they can work at the selected speed of the ARM processor. In order to accomplish this, this bus speed can be reduced the clock rate of the processor from 1⁄2 -1⁄4.

Because this bus must work accurately at power-up, and the default state at RST (reset) is for the bus to work at 1⁄4th of the processor clock rate. The second use of this is to perm it power savings whenever an application doesn't need any peripherals to work at the complete processor rate. Since the VPB-div ider is associated with the output of PLL, this remains active throughout an idle mode.

Emulation & Debugging

The microcontroller (LPC2141/42/44/46/48) holds emulation & debugging through serial port-JTAG. A trace-port perm it tracing the execution of the program. Trace functions & debugging concepts are multiplexed with port1 and GPIOs.

Code Security

The code security feature of these microcontrollers LPC2141/42/44/46/48 perm its a function to control whether it can be protected or debugged from inspection.

LPC 2148 PIN DESCRIPTION

LPC2148 m icrocontroller consists of two ports and registers. These ports could be used as input or output ports therefore the pins of these ports are called GPIO (general purposes input-output) pins.

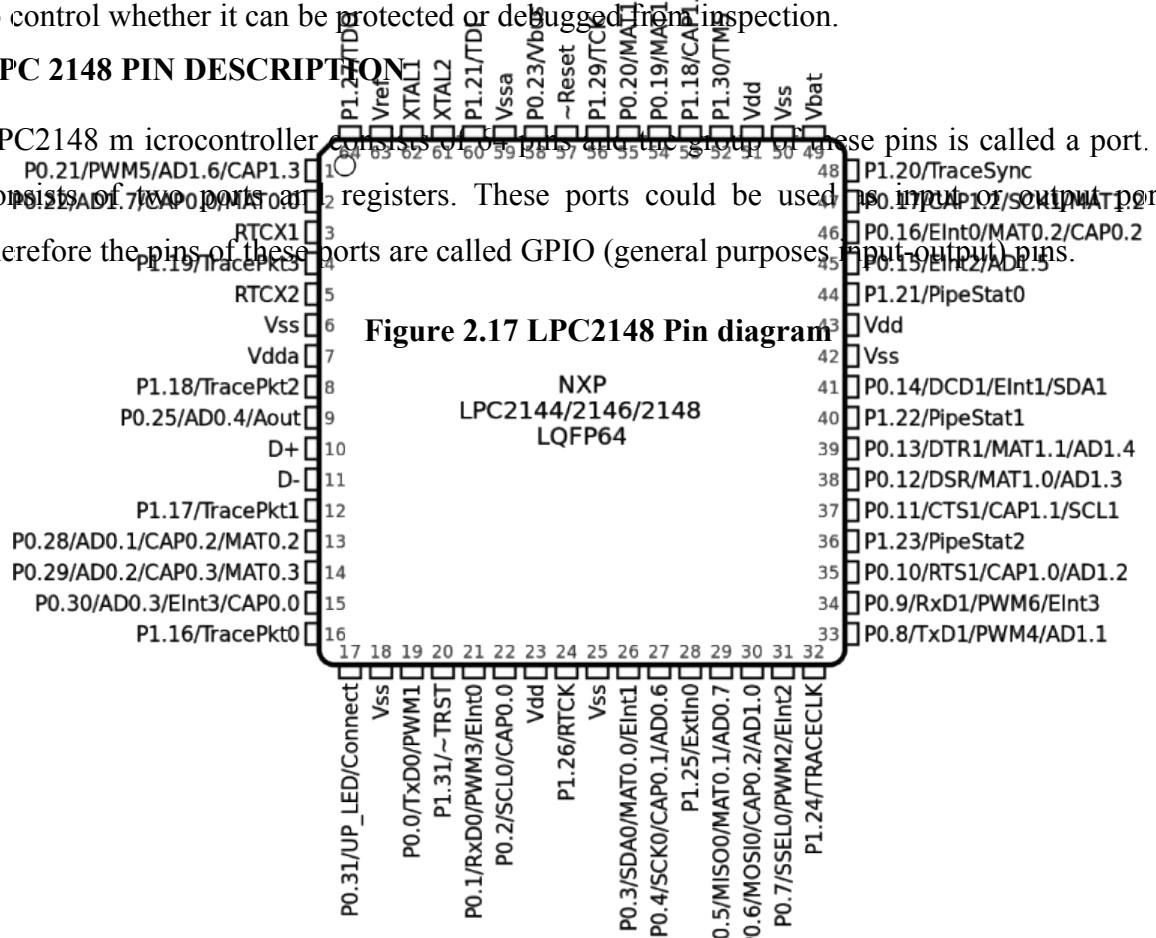


Figure 2.17 LPC2148 Pin diagram

Pin1-(P0.21/ PWM5CAP1.3/ AD1.6)

- P0.21 is a GPIO pin (general purpose I/Opin)
- AD1.6 is obtainable in LPC2144/46/48 microcontrollers only where an AD1.6 denotes ADC-1, i/p-6.
- PWM5 is a pulse width modulator output-5.
- CAP1.3 is a Capture i/p for Timer-1,channel-3

Pin2-(P0.22/ CAP0.0/AD1.7/ MAT0.02

- P0.22 is a GPIO digital pin
- AD1.7 pin is available in LPC2144/46/48 only where an AD1.7 denotes ADC-1, input-7
- CAP0.0 is a capture input pin for Timer-0,channel-0.
- MAT0.0 is a match o/p for Timer-0, channel-0

Pin3-RTXC1 3

It is an input to the RTC-oscillator circuit

Pin4- TRACEPKT3/ P1.19

- TRACEPKT3 is a trace packet, bit-3, standard input/output port by the inner pull- up.
- P1.19 is a GPIO digital pin

Pin5-RTXC2

This is an output pin from the RTC oscillator circuit

Pin6, Pin18, Pin25, Pin42,and

Pin50 These pins are a ground

reference **Pin7-VDDA**

This pin is an analog voltage power supply (3.3V), and this voltage is very useful for the on-chip analog to digital converters and digital to analog converters.

Pin8- P1.18/TRACEPKT2

- P1.18 is a GPIO digital pin
- TRACEPKT2 is a trace packet, bit-2, standard input/output port by the inner pull- up.

Pin9- P0.25/AOUT/AD0.4

- P0.25 is a GPIO digital pin I
- AD0.4 denotes ADC-0, input-4
- Aout- the output of DAC and that is accessible only in LPC2142/ LPC2144/ LPC2146/ LPC2148

Pin10- D+

This pin is a USB bidirectional D+ line

Pin11- D-

This pin is a USB bidirectional D-line

Pin12-P1.17/TRACEPKT1

- P1.17 is a GPIO digital pin
- TRACEPKT1 is a trace packet, bit-1, standard input/output port by the innerpull- up.

Pin13-P0.28/ CAP0.2/ AD0.1/MAT0.2

- P0.28 is a GPIO digital pin
- AD0.1 denotes ADC-0, input-1
- CAP0.2 is a capture i/p for Timer-0,channel-2
- MAT0.2 is a match o/p for Timer-0, channel-2

Pin14-P0.29/ CAP0.3/ AD0.2/MAT0.3

- P0.29 is a GPIO digital pin
- AD0.2 denotes ADC-0, input-2
- CAP0.3 is a capture i/p for Timer-0, channel-3
- MAT0.3 is a match o/p for Timer-0,channel-3

Pin15-P0.30/ EINT3/ AD0.3/CAP0.0

- P0.30 is a GPIO digital pin
- AD0.3 denotes ADC-0, input-3
- EINT3 is an external interrupt 3-input
- CAP0.3 is a capture i/p for Timer-0, channel-0

Pin16- P1.16/TRACEPKT0

- P1.16 is a GPIO digital pin
- TRACEPKT1 is a trace packet, bit-0, standard input/output port by inner pull-up

Pin17-P0.31/UP_LED/CONNECT

- P0.31 is a GPIO digital pin
- UP_LED is a USB good link LED indicator. When the device is arranged then it is low and when the device is not arranged, then it is high.CONNECT- This signal is used to control an exterior resistor (1.5 kΩ) under the control of a software control, and it is used by the feature of Soft Connect

Pin19- P0.0/PWM/TXD0

- P0.0 is a GPIO digital pin
- TXD0 is a transmitter o/p forUART0.
- PWM1 is a pulse width modulator o/p-1.

Pin20- P1.31/TRST

- P1.31 is a GPIO digital pin
- TRST is a test reset for JTAG interface.

Pin21-P0.1/ PWM3/ RXD0/EINT0

- P0.1 is a GPIO digital pin
- RXD0 is a receiver i/p forUART0.
- PWM3 is a pulse width modulator o/p-3.
- EINT0 is an external interrupt 0-input

Pin22- P0.2/ CAP0.0/SCL0

- P0.2 is a GPIO digital pin
- SCL0 is an I2C0 clock I/O, and open-drain o/p
- CAP0.0 is a capture i/p for Timer-0,channel-0.

Pin 23, 43, and 51- VDD

These pins are power supply voltage for the I/O ports as well as the core.

Pin24- P1.26/RTCK

- P1.26 is a GPIO digital pin
- RTCK is a returned test CLK o/p, an additional signal added to the JTAG-port. When the frequency of processor changes then it helps debuggersynchronization.

Pin26- P0.3/ SDA0/MAT0.0/EINT1

- P0.3 is a GPIO digital pin
- SDA0 is an I2C0 data I/O and open drain o/p for I2C bus observance.
- MAT0.0 is matched o/p for timer-0, channel-0.
- EINT1 is an external interrupt 1-i/p.

Pin27-P0.4/ CAP0.1/ SCK0/AD0.6

- P0.4 is a GPIO digital pin I/O
- SCK0 is a serial CLK for SPI0 and SPI CLK o/p from master/ i/p to slave.CAP0.1 is a capture i/p for timer-0, channel-0.
- IAD0.6 denotes ADC-0, input-6

Pin28-P1.25/EXTIN0

- P1.25 is a GPIO digital pin I/O
- EXTIN0 is an external trigger i/p, and standard input/output with inner pull-up

Pin29- P0.5/MAT0.1/MISO0/AD0.7

- P0.5 is a GPIO digital pin I/O
- MISO0 is a master in slave out for SPI0, data i/p to SPI-master/data o/p from SPI slave.
- MAT0.1 is a match o/p for timer-0,channel-1.
- AD0.7 denotes ADC-0, input-7.

Pin30-P0.6/MOSI0/CAP0.2/AD1.0

- P0.6 is a GPIO digital pin I/O
- MOSI0 is a master out slave in for SPI0, and data o/p from SPI master/ data i/p to SPI slave.
- CAP0.2 is a capture i/p for Timer-0, channel-2.

Pin31-P0.7/ PWM2/ SSEL0/EINT2

- P0.7 is a GPIO digital pin I/O
- SSEL0 is a slave select for SPI0 and chooses the SPI-interface as a slave.
- PWM2 is a pulse width modulator output-2.
- EINT2 is an external interrupt 2-input.

Pin32-P1.24/TRACECLK

- P1.24 is a GPIO digital pin I/O.
- TRACECLK is a trace CLK and standard input/output port with inner pull-up

Pin33-P0.8/TXD1/PWM4/AD1.1

- P0.8 is a GPIO digital pin I/O

- TXD1 is a transmitter o/p forUART1.

- PWM4 is a pulse width modulator o/p-4.
- AD1.1 denotes ADC-1, input-1, and it is obtainable only in LPC2144/46/48.

Pin34- P0.9/PWM6/RXD1/EINT3

- P0.9 is a GPIO digital pin I/O
- RXD1 is a receiver i/p forUART1.
- PWM6 is a pulse width modulator o/p-6.
- EINT3 is an external interrupt 3-input

Pin35-P0.10/RTS1/CAP1.0/AD1.2

- P0.10 is a GPIO digital pin I/O
- RTS1 is requesting to send o/p for UART1 and LPC2144/46/48.
- CAP1.0 is a capture i/p for timer-1, channel-0.
- AD1.2 denotes ADC-1, input-2, and it is obtainable only in LPC2144/46/48

Pin36-P1.23/PIPESTAT2

- P1.23 is a GPIO digital pin I/O
- PIPESTAT2 is a pipeline status, bit-2., and standard Input/Output port with inner pull-up

Pin37-P0.11/ CAP1.1/CTS1/ SCL1

- P0.11 is a GPIO digital pin I/O
- CTS1 is clear to send i/p for UART1, and these are accessible only in LPC2144/46/48
- CAP1.1 is a capture i/p for timer-1, channel-1.
- SCL1 — I2C1 CLK I/O, and open drain o/p for the I2C-bus observance

Pin38-P0.12/ MAT1.0/AD1.3/ DSR1

- P0.12 is a GPIO digital pin I/O
- DSR1 is a data set ready i/p for UART1, and these are accessible only in LPC2144/46/48.
- MAT1.0 is a match o/p for timer-1,channel-0.
- AD1.3 denotes ADC input-3, and it is accessible only in LPC2144/46/48.

Pin39-P0.13/DTR1/MAT1.1/AD1.4

- P0.13 is a GPIO digital pin I/O
- DTR1 is a data terminal ready o/p for UART1 and LPC2144/46/48 only.
- MAT1.1 is a match o/p for timer-1,channel-1.
- AD1.4 denotes ADC input-4, and these are accessible only in LPC2144/46/48.

Pin40-P1.22/PIPESTAT1

- P1.22 is a GPIO digital pin I/O
- PIPESTAT1 is a pipeline status, bit-1, and standard Input/Output port with inner pull-up Pin41- P0.14/DCD1/EINT1/SDA1
- P0.14 is a GPIO digital pin I/O
- DCD1 is a data carrier detect i/p for UART1, and also only for LPC2144/46/48 only.
- EINT1 is an exterior interrupt 1-input.
- SDA1 is an I2C1 data I/O and an open drain o/p for I2C bus observance

Pin44:P1.21/ PIPESTAT0 44

- I/O P1.21 is a GPIO digital pin I/O
- PIPESTAT0 is a Pipeline Status, bit 0, and standard Input/Output port by the inner pull-up.

Pin45: P0.15/ EINT2/ RI1/ AD1.5 45

- I/O P0.15 is a GPIO digital pin I/O
- RI1 is a ring pointer i/p for UART1 and it is accessible only in LPC2144/46/48.
- EINT2 is an external interrupt 2-input.
- AD1.5 indicates ADC 1, input-5, and also available only in LPC2144/46/48

Pin46: P0.16/ MAT0.2/ EINT0/ CAP0.2

- P0.16 is a GPIO digital pin I/O
- EINT0 is an external interrupt0- input.
- MAT0.2 is a match o/p for Timer-0, channel -2
- CAP0.2 is a capture i/p for Timer-0, channel-2.

Pin47: P0.17/ SCK1/ CAP1.2/ MAT1.2 47

- P0.17 is a GPIO digital pin I/O
- CAP1.2 is a capture i/p for Timer-1,channel-2.
- SCK1 is a serial CLK for SSP and CLK o/p from master to slave.
- MAT1.2 is a match o/p for Timer-1, channel-2.

Pin48: P1.20/ TRACESYNC

- P1.20 is a GPIO digital pin I/O
- TRACESYNC is tracesynchronization.

Pin49: VBAT

RTC power supply: This pin gives the supply to the RTC

Pin52: P1.30/TMS

- P1.30 is a GPIO digital pin I/O
- TMS is a test mode select for interfacing of JTAG.

Pin53: P0.18/CAP1.3/ MISO1/MAT1.3

- P0.18 is a GPIO digital pin I/O
- CAP1.3 is a capture i/p for Timer 1, channel 3.
- MISO1 is a master In Slave-out for SSP, and data i/p to SPI- master

Pin54: P0.19/ MOSI1/MAT1.2/CAP1.2

- P0.19 is a GPIO digital pin I/O.
- MAT1.2 denotes match o/p for Timer 1, channel 2.
- MOSI1 is a master out slave for SSPmaster.
- CAP1.2 is a capture i/p for Timer 1, channel 2.

Pin 55: P0.20/ SSEL1/ MAT1.3/ EINT3

- P0.20 is a GPIO digital pin I/O.
- MAT1.3 is a match o/p for Timer 1, channel 3. I
- SSEL1 is a Slave Select designed for SSP. Here, chooses the interface of SSP as a slave.
- EINT3 is an external interrupt 3-input.

Pin56: P1.29/TCK

- P1.29 is a GPIO digital pin I/O
- TCK is a test CLK for an interface of JTAG.

Pin57: External Reset Input

The device can be rearranged by a LOW on this pin, effecting Input/Output ports as well as peripherals for obtaining on their default conditions, & processor execution begins at address 0.

Pin58: P0.23/VBUS

- P0.23 is a GPIO digital pin I/O
- VBUS specifies the existence of USB-bus power

Pin59: VSSA

VSSA is an analog ground, and this must be the sim ilar voltage like VSS, although it should be separated to reduce error and noise

Pin60: P1.28/TDI 60

- P1.28 is a GPIO digital pin I/O
- TDI pin is a test data is used for interfacing JTAG

Pin61: XTAL2

XTAL2 is an o/p from the oscillator amplifier

Pin62: XTAL1

XTAL1 is an i/p to the internal CLK generator as well as oscillator circuits

Pin63: VREF-ADC Reference

This pin should be nom inally equal or less than to the voltage VDD although it should be separated for reducing error as well as noise.

Pin64: P1.27/TDO 64

- P1.27 is a GPIO digital pin I/O
- TDO is a test data out used for interfacing JTAG.

7. LPC214X FAMILY PERIPHERALS

TIMER/COUNTER TIMER

Timer is a specific type of clock which is used to measure the time intervals. It measures the time interval by counting the input clocks. Every timer needs a clock to work. We can measure any time interval if we know the time of one clock period.

e.g. Let's say we have 1 kHz input clock frequency for the timer unit, then, We can calculate time of one clock period as,

Time of one clock period = 1 / clock frequency

= 1 / 1000

= 1millisecond

i.e. 1000 clock counts prov ide a time interval of 1 second, and hence we can prov ide 1 second delay with these 1000 clock counts.

Now, once we have the time period of one clock, we can use this time period to generate delays that are integer multiples of it. We can also use the time period to measure the time interval between specific events of a received signal.

COUNTER

Counter is the unit which is sim ilar to Timers but works in a reverse manner to the timers. It counts the external events or we can say external clock ticks. It is mostly used to measure frequency from the counts of clock ticks.

e.g. Let's say Counter is measuring counts of external clock ticks, and frequently its count reaches 2000 in one second i.e. 2000 clock ticks/second.

Then, we can calculate external clock frequency as, External clock frequency = count of clocks / one second

= 2000 / 1

= 2 kHz

Hence, we can measure such external clock/event frequencies using counter.

There are many applications for which we can use these timers and counters in real world.

LPC TIMER/COUNTER

LPC2148 has two 32-bit timers/counters: Timer0/Counter0 & Timer1/Counter1.

- LPC2148 Timer has input of peripheral clock (PCLK) or an external clock. It counts the clock from either of these clock sources for its operation.
- LPC2148 T imer/Counter can generate an interrupt signal at specified time value.
- LPC2148 has match registers that contain count value which is continuously compared with the value of the Timer register. When the value in the Timer register matches the value in the match register, specific action (timer reset, or timer stop, or generate an interrupt) is taken.
- Also, LPC2148 has capture registers which can be used to capture the timer value on a specific external event on capture pins

TIMER 0 REGISTERS

1. T0IR (Timer0 Interrupt Register)

- It is an 8-bit read-write register.
- Consists of 4 bits for match register interrupts and 4 bits for compare register interrupts.
- If interrupt is generated, then the corresponding bit in this register will be high, otherwise it will be low.
- Writing a 1 to any bit of this register will reset that interrupt. Writing a 0 has no effect.

7	6	5	4	3	2	1	0
CR3	CR2	CR1	CR0	MR3	MR2	MR1	MR0

Figure 2.18 T0IR (Timer0 Interrupt Register)

2. T0TCR (Timer0 Timer Control Register)

- It is an 8-bit read-write register.

- It is used to control the operation of the timer counter.

- **Bit 0 – Counter Enable**

  0 = Counters are disabled

  1 = Timer counter and Prescale counter are enabled for counting

- **Bit 1 – Counter Reset**

  0 = Counter not reset

  1 = Timer counter and Prescale counter are synchronously reset on next positive edge of PCLK

Figure 2.19 T0TCR (Timer0 Timer Control Register)

3. T0CTCR (Timer0 Counter Control Register)

- It is an 8-bit read-write register.

- Used to select between timer counter mode.

- When in counter mode, it is used to select the pin and edges for counting.

Figure 2.20 T0CTCR (Timer0 Counter Control Register)

- **Bits 1:0 – Counter/Timer Mode**

  This field selects which rising edges of PCLK can increment Timer's Prescale Counter (PC), or clear PC and increment Timer Counter (TC).

  00 = Timer Mode: Every rising edge of PCLK

  01 = Counter Mode: TC is incremented on rising edge on the capture input selected by Bits 3:2.

  10 = Counter Mode: TC is incremented on falling edge on the capture input selected by Bits 3:2

  01 = Counter Mode: TC is incremented on both edges on the capture input selected by Bits 3:2

- **Bits 3:2 – Count Input Select**

When bits 1:0 in this register are not 00, these bits select which capture pin is sampled for clocking. 00 = CAP0.0

01 = CAP0.1

10 = CAP0.2

11 = CAP0.3

- **Note** : If counter mode is selected for a certain capture (CAP) input, then the corresponding 3 bits in the T0CCR register must be programmed as 000. Capture and/or interrupt can be selected for other CAP inputs.

4. T0TC (Timer0 Timer Counter)

- It is a 32-bit timercounter.
- It is incremented when the Prescale Counter (PC) reaches its maximum value held by Prescaler Register (PR).

Note: When TC overflow occurs, it does not generate any overflow interrupt. Alternatively, we can use match register to detect overflow event if needed.

T0PR (Timer0 Prescale Register)

- It is a 32-bit register.
- It holds the maximum value of the Prescale Counter.

5. T0PC (Timer0 Prescale Counter Register)

- It is a 32-bit register.
- It controls the div ision of PCLK by some constant value before it is applied to the Timer Counter.
- It is incremented on every PCLK.
- When it reaches the value in Prescale Register, the T imer Counter isincremented and Prescale Counter is reset on next PCLK.

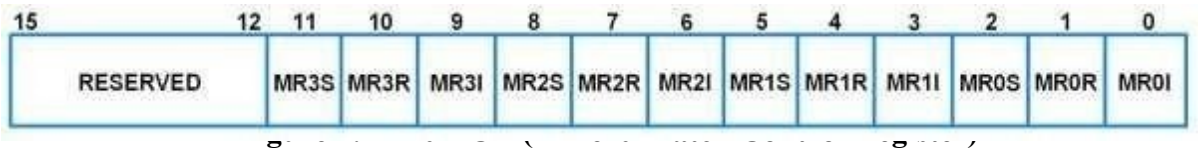
6. T0MR0-T0MR3 (Timer0 Match Registers)

- These are 32-bit registers.
- The values stored in these registers are continuously compared with the Timer Counter value.
- When the two values are equal, the timer can be reset or stop or an interrupt may be generated. The T0MCR controls what action should be taken

on a match.

8. T0MCR (Timer0 Match Control Register)

- It is a 16-bit register.
- It controls what action is to be taken on a match between the Match Registers and Timer Counter.



- **Bit 0 – MR0I (Match register 0 interrupt)**

  0 = This interrupt is disabled

1 = Interrupt on MR0. An interrupt is generated when MR0 matches the value in TC (Timer Counter)

- **Bit 1 – MR0R (Match register 0 reset)**

  0 = This feature is disabled

  1 = Reset on MR0. The TC (Timer Counter) will be reset if MR0 matches it

- **Bit 2 – MR0S (Match register 0 stop)**

  0 = This feature is disabled

  1 = Stop on MR0. The TC (Timer Counter) and PC (Prescale Counter) is stopped and Counter Enable bit in T0TCR is set to 0 if MR0 matches TC

- MR1, MR2 and MR3 bits function in the same manner as MR0 bits.

PULSE WIDTHMODULATION

Pulse W idth Modulation (PWM) is a technique by which width of a pulse is varied while keeping the frequency constant.

A period of a pulse consists of an ON cycle (HIGH) and an OFF cycle (LOW). The fraction for which the signal is ON over a period is known as duty cycle.

$$\text{Duty Cycle (In \%)} = \frac{Ton}{Ton+Toff} \times 100$$

E.g. Consider a pulse with a period of 10ms which remains ON (high) for 2ms.The duty cycle of this pulse will be

D = (2ms / 10ms) x 100 = 20%

Through PWM technique, we can control the power delivered to the load by using ON- OFF signal.

Pulse Width Modulated signals with different duty cycle are shown in figure 2.22.

LPC2148 has PWM peripheral through which we can generate multiple PWM signals on PWM pins. Also, LPC2148 supports two types of controlled PWM outputs as,

Single Edge Controlled PWM: All the rising (positive going) edges of the output waveform are positioned/fixed at the beginning of the PWM period. Only falling (negative going) edge position can be controlled to vary the pulse width of PWM.

Double Edge Controlled PWM: All the rising (positive going) and falling (negative going) edge positions can be controlled to vary the pulse width of PWM. Both the rising as well as the falling edges can be positioned anywhere in the PWM period.

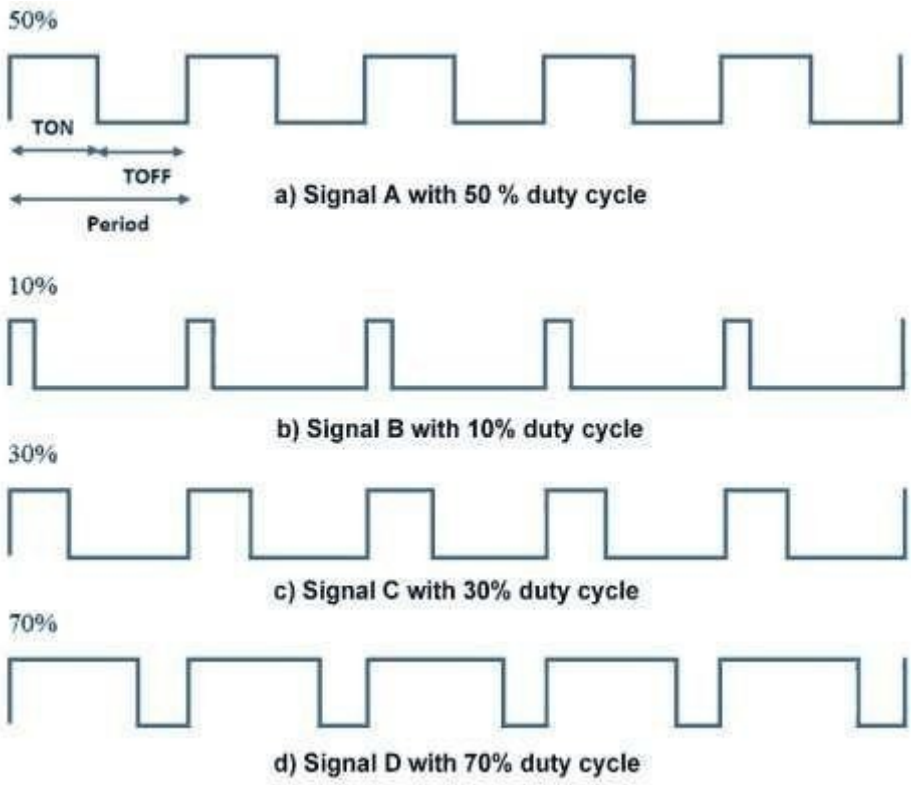


Figure 2.22 PWM signal with different Duty Cycle Waveforms

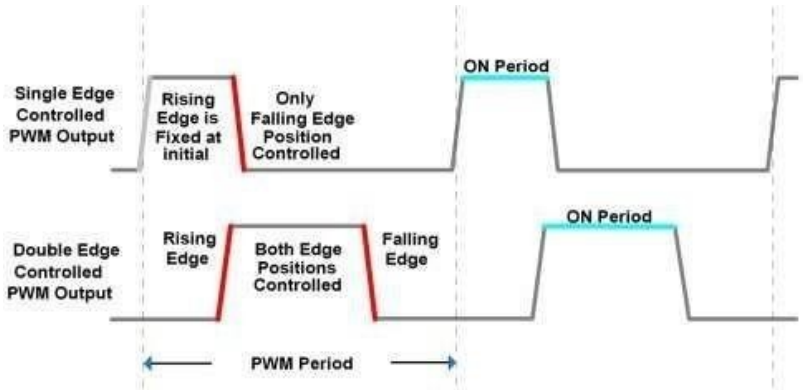


Figure 2.23 Types of PWM output supported by LPC2148

LPC2148 PWM

- The PWM in LPC2148 is based on standard 32-bit Timer Counter, i.e. PWMTC (PWM Timer Counter). This Timer Counter counts the cycles of peripheral clock (PCLK).
- Also, we can scale this timer clock counts using 32-bit PWM Prescale Register (PWMPR).
- LPC2148 has 7 PWM match registers (PWMMR0 – PWMMR06).
- One match register (PWMMR0) is used to set PWM frequency.
- Remaining 6 match registers are used to set PWM width for 6 different PWM signals in Single Edge Controlled PWM or 3 different PWM signals in Double Edge Controlled PWM.
- Whenever PWM Timer Counter (PWMTC) matches with these Match Registers then, PWM Timer Counter resets, or stops, or generates match interrupt, depending upon settings in PWM Match Control Register (PWMMCR).
- As shown in figure 2.24, PWMMR0 = 6 i.e. PWM period is 6 counts, after which PWM Timer Counter resets.
- PWM2 & PWM3 are configured as Single Edge Controlled PWM and PWM5 is configured as Double Edge Controlled PWM.
- Prescaler is set to increment PWM Timer Counter after every two Peripheral clocks (PCLK).
- Match registers (PWMMR2 & PWMMR3) are used to set falling edge position for PWM2 & PWM3.
- PWMMR4 & PWMMR5 are used to set rising & falling edge positions respectively for PWM5.

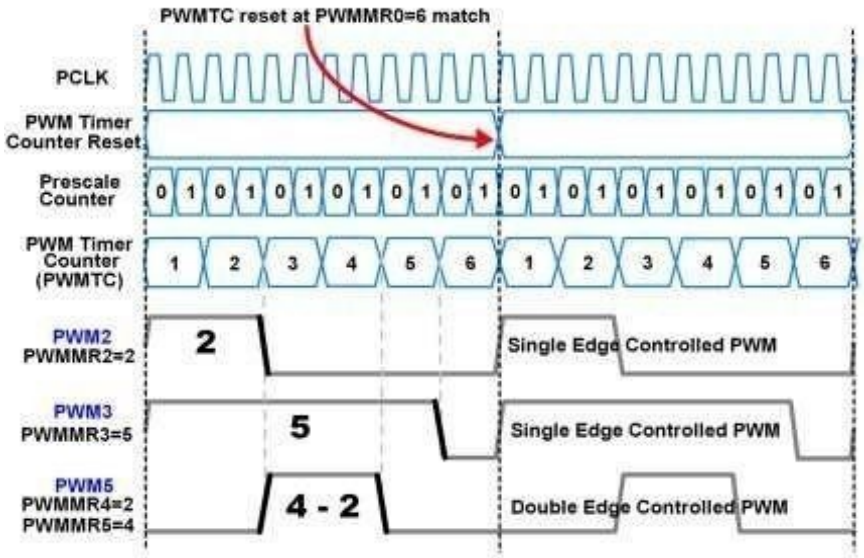


Figure 2.24 LPC2148PWM signal

DIFFERENT PWM THAT CAN BE GENERATED USING

LPC2148

The table 2.11 given below shows when the PWM is Set (Rising Edge) and Reset (Fal]ing Edge) for different PWM channels using 7 Match Register.

PWM Channel	Single Edge Controlled		Double Edge Controlled	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0	Match 1
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2	Match 3
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4	Match 5
6	Match 0	Match 6	Match 5	Match 6

Table 2.11 PWM set and reset for different PWM channels

LPC2148 PWM PINS

The pins that are used for PWM in LPC2148 are

P0.0/TXD0/**PWM1** P0.7/SSEL0/**PWM2**/ENT2

P0.1/RXD0/**PWM3**/ENT0

P0.8/TXD1/**PWM4**/AD1.1

P0.21/**PWM5**/AD1.6/CAP1.3

P0.9/RXD1/**PWM6**/EINT3

THE VARIOUS PWM REGISTERS THAT ARE USEFUL IN CONTROLLING AND GENERATING PWM.

1. PWMIR (PWM Interrupt Register)

- It is a 16-bit register.
- It has 7 interrupt bits corresponding to the 7 PWM match registers.
- If an interrupt is generated, then the corresponding bit in this register becomes HIGH.
- Otherwise the bit wil] beLOW.
- Writing a 1 to a bit in this register clears that interrupt.
- Writing a 0 has no effect.

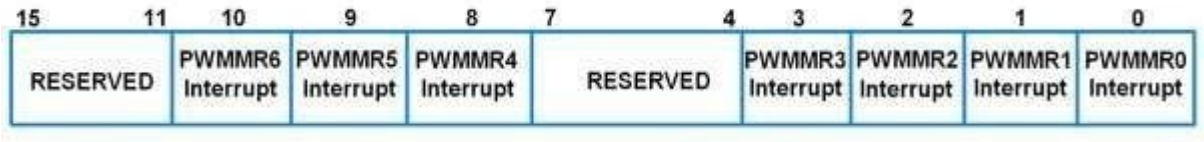


Figure 2.25 PWMIR (PWM Interrupt Register)

2. PWMTCR (PWM Timer Control Register)

- It is an 8-bit register.
- It is used to control the operation of the PWM Timer Counter.

Figure 2.26 PWMTCR (PWM Timer Control Register)

- **Bit 0 – Counter Enable**
  When 1, PWM Timer Counter and Prescale Counter are enabled.
  When 0, the counters are disabled.

- **Bit 1 – Counter Reset**
  When 1, the PWM Timer Counter and PWM Prescale Counter are synchronously reset on next positive edge of PCLK. Counter remains reset until this bit is returned to 0.

- **Bit 3 – PWM Enable**
  This bit always needs to be 1 for PWM operation. Otherwise PWM will operate as a normal timer.
  When 1, PWM mode is enabled and the shadow registers operate along with match registers. A write to a match register will have no effect as long as corresponding bit in PWMLER is not set. Note: PWMMR0 must always be set before PWM is enabled, otherwise match event will not occur to cause shadow register contents to become effective.

3. PWMTC (PWM Timer Counter)

- It is a 32-bit register.
- It is incremented when the PWM Prescale Counter (PWMPC) reaches its terminal count.

4. PWMPR (PWM Prescale Register)

- It is a 32-bit register.
- It holds the maximum value of the Prescale Counter.

5. PWMPC (PWM Prescale Counter)

- It is a 32-bit register.
- It controls the division of PCLK by some constant value before it is applied to the PWM Timer Counter.
- It is incremented on every PCLK.
- When it reaches the value in PWM Prescale Register, the PWM Timer Counter is incremented and PWM Prescale Counter is reset on next PCLK.

6. PWMMR0-PWMMR6 (PWM Match Registers)

- These are 32-bit registers.
- The values stored in these registers are continuously compared with thePWM Timer Counter value.
- When the two values are equal, the timer can be reset or stop or an interrupt may be generated.
- The PWMMCR controls what action should be taken on a match.

7. PWMMCR (PWM Match Control Register)

- It is a 32-bit register.
- It controls what action is to be taken on a match between the PWM Match Registers and PWM Timer Counter.

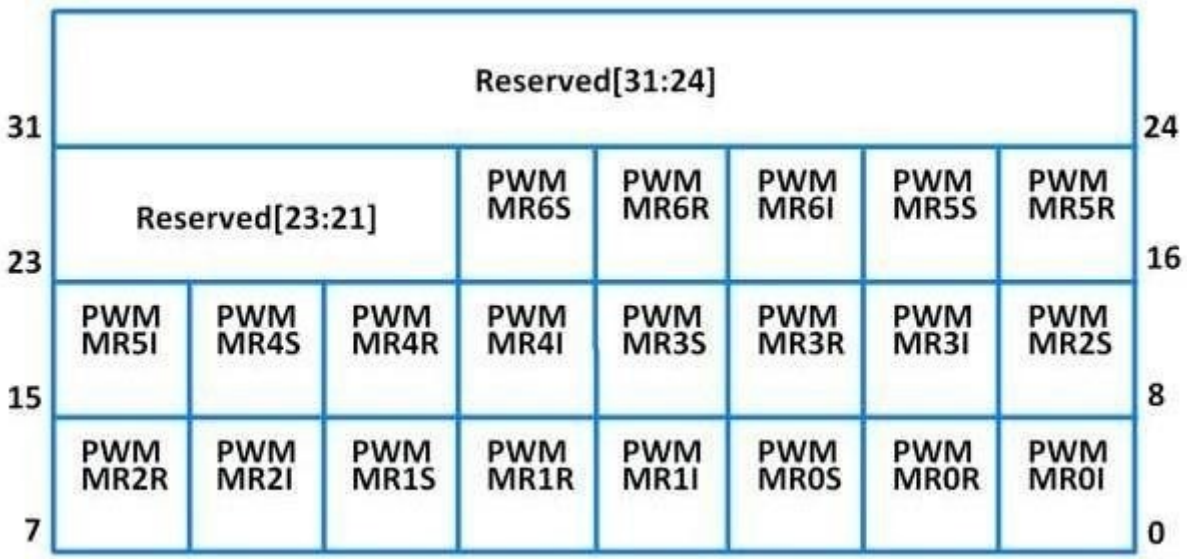


Figure 2.27 PWMMCR (PWM Match Control Register)

- **Bit 0** – PWMMR0I (PWM Match register 0 interrupt) 0 = This interrupt is disabled

  1 = Interrupt on PWMMR0. An interrupt is generated when PWMMR0 matches the value in PWMTC

- **Bit 1** – PWMMR0R (PWM Match register 0 reset) 0 = This feature is disabled 1 = Reset on PWMMR0. The PWMTC wi]be reset if PWMMR0 matches it

- **Bit 2** – PWMMR0S (PWM Match register 0 stop) 0 = This feature is disabled

  1 = Stop on PWMMR0. The PWMTC and PWMPC is stopped and Counter Enable bit in PWMTCR is set to 0 if PWMMR0 matches PWMTC

- PWMMR1, PWMMR2, PWMMR3, PWMMR4, PWMMR5 and PWMMR6 has same function bits (stop, reset, interrupt) as in PWMMR0.

8. PWMPCR (PWM Control Register)

- It is a 16-bit register.
- It is used to enable and select each type of PWM.

Figure 2.28 PWMPCR (PWM Control Register)

- **Bit 2** – PWMSEL2

  0 = Single edge controlled mode for PWM2 1 = Double edge controlled mode for PWM2 All other PWMSEL bits have similar operation as PWMSEL2 above.

- **Bit 10** – PWMENA2

  0 = PWM2 output disabled 1 = PWM2 output enabled

  All other PWMENA bits have similar operation as PWMENA2 above.

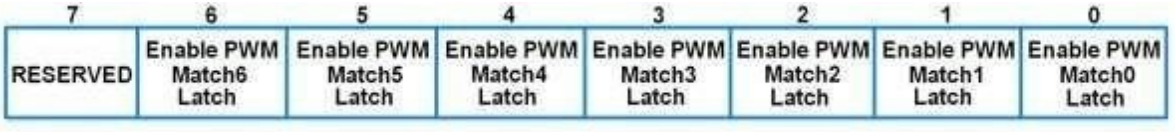
9. PWMLER (PWM Latch Enable Register)

- It is an 8-bit register.
- It is used to control the update of the PWM Match Registers when they are used for PWM generation.
    - When a value is written to a PWM Match Register while the timer is in PWM mode, the value is held in the shadow register. The contents of the shadow

register are transferred to the PWM Match Register when the timer resets (PWM Match 0 event occurs) and if the corresponding bit in PWMLER is set.

- Bit 6 – Enable PWM Match 6 Latch
  Writing a 1 to this bit allows the last written value to PWMMR6 to become effective when timer next is reset by the PWM match event.

- Similar description as that of Bit 6 for the remaining bits.



7	6	5	4	3	2	1	0
RESERVED	Enable PWM Match6 Latch	Enable PWM Match5 Latch	Enable PWM Match4 Latch	Enable PWM Match3 Latch	Enable PWM Match2 Latch	Enable PWM Match1 Latch	Enable PWM Match0 Latch

Figure 2.29 PWMLER (PWM Latch Enable Register) STEPS IN PWM GENERATION

- Reset and disable PWM counter using PWMTCR
- Load prescale value according to need of application in thePWMPR
- Load PWMMR0 with a value corresponding to the time period of your PWM wave
- Load any one of the remaining six match registers (two of the remaining six match registers for double edge controlled PWM) with the ON duration of the PWM cycle. (PWM will be generated on PWM pin corresponding to the match register you load the value with).

- Load PWMMCR with a value based on the action to be taken in the event of a match between match register and PWM timer counter.

- Enable PWM match latch for the match registers used with the help of PWMLER

- Select the type of PWM wave (single edge or double edge controlled) and which PWMs to be enabled using PWMPCR

- Enable PWM and PWM counter using PWMTCR

UART

INTRODUCTION

UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol in which data is transferred serially bit by bit at a time. Asynchronous seria l communication is widely used for byte-oriented transm ission. In Asynchronous seria l communication, a byte of data is transferred at a time.

UART serial communication protocol uses a defined frame structure for their data bytes. Frame structure in Asynchronous communication consists:

- **START bit:** It is a bit with which indicates that serial communication has started and it is always low.
- **Data bits packet**: Data bits can be packets of 5 to 9 bits. Normally we use 8- bit data packet, which is always sent after the START bit.
- **STOP bit**: This usually is one or two bits in length. It is sent after data bits packet to indicate the end of frame. Stop bit is always logichigh.

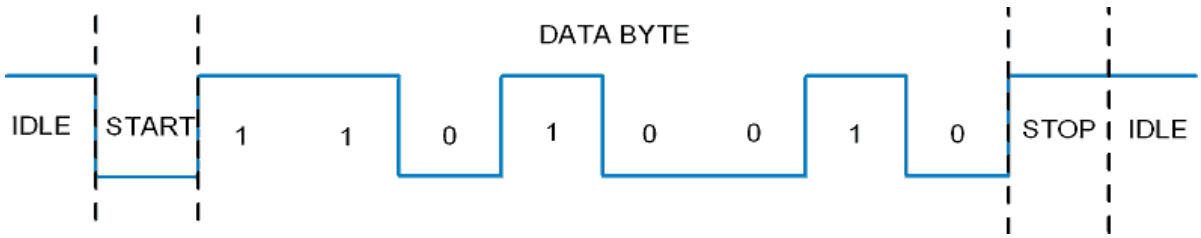


Figure 2.30 UART Frame structure

Usually, an asynchronous serial communication frame consists of a START bit (1 bit) followed by a data byte (8 bits) and then a STOP bit (1 bit), which forms a 10-bit frame as shown in the figure 2.30. The frame can also consist of 2 STOP bits instead of a single bit, and there can also be a PARITY bit after the STOP bit.

LPC2148 UART

LPC2148 has two inbuilt UARTs available i.e. UART0&UART1. So, we can connect two UART enabled dev ices (GSM module, GPS module, Bluetooth module etc.) with LPC2148 at

a time.

UART0 and UART1 are identical other than the fact that UART1 has modem interface included.

FEATURES OF UART0

- 16 byte Receive and Transmit FIFOs
- Built-in fractional baud rate generator with autobauding capabilities
- Software flow control through TXEN bit in Transmit Enable Register

FEATURES OF UART1

- 16 byte Receive and Transmit FIFOs
- Built-in fractional baud rate generator with autobauding capabilities
- Software and hardware flow control implementation possible
- Standard modem interface signals included with flow control (auto-CTS/RTS) fully supported in hardware

LPC2148 UART PINS

LPC2148 has 2 pins for UART0 and 8 pins for UART1.

UART0:

1. **TXD0** (Output pin): Serial Transmit datapin.
2. **RXD0** (Input pin): Serial Receive data pin.

UART1:

1. **TXD1** (Output pin): Serial Transmit datapin.
2. **RXD1** (Input pin): Serial Receive data pin.
3. **RTS1** (Output pin): Request To Send signal pin. Active low signal indicates that the UART1 would like to transmit data to the externalmodem.
4. **CTS1** (Input pin): Clear To Send signal pin. Active low signal indicates if the externa l modem is ready to accept transmitted data via TXD1 from the UART1.
5. **DSR1** (Input pin): Data Set Ready signal pin. Active low signal indicates if the external modem is ready to establish a communication link with the UART1.

6. **DTR1** (Output pin): Data Terminal Ready signal pin. Active low signal indicates that the UART1 is ready to establish connection with external modem.
7. **DCD1** (Input pin): Data Carrier Detect signal pin. Active low signal indicates ifthe external modem has established a communication link with the UART1 and data may be exchanged.
8. **RI1** (Input pin): Ring Indicator signal pin. Active low signal indicates that a telephone ringing signal has been detected by the modem.

UART0 REGISTERS

UART1 can be used in a similar way by using the corresponding registers for UART1.

1. **U0RBR (UART0 Receive Buffer Register)**

- It is an 8-bit read onlyregister.
- This register contains the received data.
- It contains the “oldest” received byte in the receive FIFO.
- If the character received is less than 8 bits, the unused MSBs are padded with zeroes.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. (DLAB = 0)

Figure 2.31 U0RBR (UART0 Receive Buffer Register)

2. **U0THR (UART0 Transmit Holding Register)**

- It is an 8-bit write onlyregister.
- Data to be transmitted is written to this register.
- It contains the “newest” received byte in the transmit FIFO.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access theU0THR. (DLAB = 0)

Figure 2.32 U0THR (UART0 Transmit Holding Register)

3. U0DLL and U0DLM (UART0 Divisor Latch Registers)

- U0DLL is the Divisor Latch LSB.
- U0DLM is the Divisor Latch MSB.
- These are 8-bit read-write registers.
- UART0 Div isor Latch holds the value by which the PCLK(Peripheral Clock) will be divided. This value must be 1/16 times the desired baud rate.
- A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART0 Divisor Latches. (DLAB = 1)

Figure 2.33 U0DLL

Figure 2.34 U0DLM

4. U0FDR (UART0 Fractional Divider Register)

- It is a 32-bit read write register.
- It decides the clock pre-scalar for baud rategeneration.
- If fractional div ider is active (i.e. DIVADDVAL>0) and DLM = 0, DLL must be greater than 3.

Figure 2.35 U0FDR (UART0 Fractional Divider Register)

- If DIVADDVAL is 0, the fractional baudrate generator will not impact the UART0 baudrate.
- Reset value of DIVADDVAL is0.
- MULVAL must be greater than or equal to 1 for UART0 to operate properly, regardless of whether the fractional baudrate generator is used or not.
- Reset value of MULVAL is1.
- The formula for UART0 baudrate is given below

$$\text{UART0 Baudrate} = \frac{Pclk}{DIVADDVAL}$$

$$16 * (256 * U0DLM + U0DLL * (1 + MULVAL)$$

- MULVAL and DIVADDVAL should have values in the range of 0 to 15. If this isnot ensured, the output of the fractional divider is undefined.
- The value of the U0FDR should not be modified while transm itting/receiv ing data. This may result in corruption of data.

5. U0IER (UART0 Interrupt Enable Register)

- It is a 32-bit read-write register.
  - It is used to enable UART0 interrupt sources.
- DLAB should be zero (DLAB = 0).

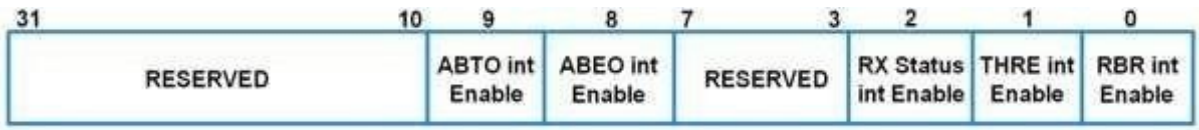


Figure 2.36 U0IER (UART0 Interrupt Enable Register)

- **Bit 0 - RBR Interrupt Enable. It also controls the Character Receive Time-Out interrupt.**

  0 = Disable Receive Data Available interrupt 1 = Enable Receive Data Available interrupt
- **Bit 1 - THRE Interrupt Enable**

  0 = Disable THRE interrupt 1 = Enable THREinterrupt
- **Bit 2 - RX Line Interrupt Enable**

  0 = Disable UART0 RX line status interrupts 1 = EnableUART0 RX line status interrupts
- **Bit 8 - ABEO Interrupt Enable**

  0 = Disable auto-baud time-out interrupt 1 = Enable auto-baud time-out interrupt
- **Bit 9 - ABTO Interrupt Enable**

  0 = Disable end of auto-baud interrupt

  1 = Enable the end of auto-baud interrupt

6. U0IIR (UART0 Interrupt Identification Register)

- It is a 32-bit read onlyregister.

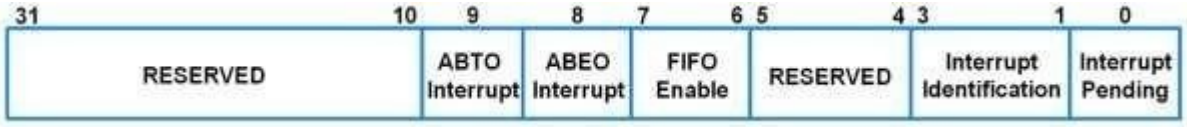


Figure 2.37 U0IIR (UART0 Interrupt Identification Register)

- It provides a status code that denotes the priority and source of a pending interrupt.
- It must be read before exiting the Interrupt Service Routine to clear the interrupt.
- **Bit 0 - Interrupt Pending**
  0 = At least one interrupt is pending 1 = No interrupts pending
- **Bit 3:1 - Interrupt Identification**
  Identifies an interrupt corresponding to theUART0 Rx FIFO. 011 = Receive Line Status (RLS) Interrupt

  010 = Receive Data Available (RDA) Interrupt
  110 = Character Time-out Indicator (CTI) Interrupt 001 = THRE Interrupt

- **Bit 7:6 - FIFO Enable.**
  These bits are equivalent to FIFO enable bit in FIFO Control Register, 0 = If FIFOs are disabled

  1 = FIFOs are enabled
- **Bit 8 - ABEO Interrupt**
  If interrupt is enabled, 0 = No ABEO interrupt

  1 = Auto-baud has finished successfully
- **Bit 9 - ABTO Interrupt**
  If interrupt is enabled, 0 = No ABTO interrupt 1 =

Auto- baud has timed out

7. U0LCR (UART0 Line Control Register)

- It is an 8-bit read-write register.
- It determines the format of the data character that is to be transmitted or received.

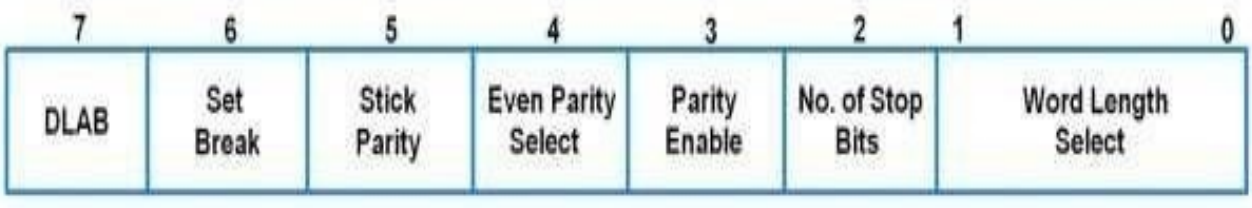


Figure 2.38 U0LCR (UART0 Line Control Register)

- **Bit 1:0 - Word Length Select**

  00 = 5-bit character length

  01 = 6-bit character length 10 = 7-bit character length 11 = 8-bit character length

- **Bit 2 - Number of Stop Bits**

  0 = 1 stop bit

  1 = 2 stop bits

- **Bit 3 - Parity Enable**

  0 = Disable parity generation and checking 1 = Enable parity generation and checking

- **Bit 5:4 - Parity Select**

  00 = Odd Parity 01 = Even Parity 10 = Forced "1" Stick Parity

  11 = Forced "0" Stick Parity

- **Bit 6 - Break Control**

  0= Disable break transmission 1 = Enable

break transmission

- **Bit 7 - Divisor Latch Access Bit (DLAB)** 0

  = Disable access to Divisor Latches 1 = Enable

  access to Divisor Latches

8. U0LSR (UART0 Line Status Register)

- It is an 8-bit read onlyregister.

Figure 2.39 U0LSR (UART0 Line Status Register)

- It provides status information on UART0 RX and TX blocks.
- **Bit 0 - Receiver Data Ready**

  0 = U0RBR is empty

  1 = U0RBR contains valid data
- **Bit 1 - Overrun Error**

  0 = Overrun error status inactive 1 = Overrun error status

  active This bit is cleared when U0LSR is read.
- **Bit 2 - Parity Error**

  0 = Parity error status inactive

1 = Parity error status active

This bit is cleared when U0LSR is read.

- **Bit 3 - Framing Error**

  0 = Framing error status inactive 1 = Framing error status active

  This bit is cleared when U0LSR is read.

- **Bit 4 - Break Interrupt**

  0 = Break interrupt status inactive 1 = Break interrupt status active

  This bit is cleared when U0LSR is read.

- **Bit 5 - Transmitter Holding Register Empty**

  0 = U0THR has valid data 1 = U0THR empty

- **Bit 6 - Transmitter Empty**

  0 = U0THR and/or U0TSR contains valid data 1 = U0THR and U0TSR empty

- **Bit 7 - Error in RX FIFO(RXFE)**

  0 = U0RBR contains no UART0 RX errors

  1 = U0RBR contains at least one UART0 RX error This bit is cleared when U0LSR is read.

9. U0TER (UART0 Transmit Enable Register)

- It is an 8-bit read-write register.

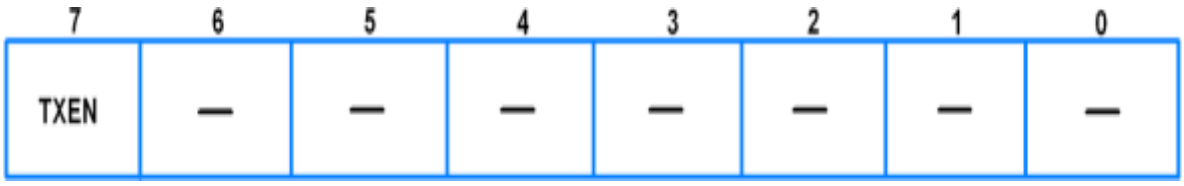


Figure2.40 U0TER (UART0 Transmit Enable Register)

- The U0TER enables implementation of software flow control. When TXEn=1, UART0 transmitter will keep sending data as long as they are available. As soon as TXEn becomes 0, UART0 transmission will stop.

- Software implementing software-handshaking can clear this bit when it receives an XOFF character (DC3). Software can set this bit again when itreceives an XON (DC1) character.
- **Bit 7 : TXEN**

  0 = Transmission disabled 1 = Transmission enabled
- If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit isset again.

PROGRAMMING OF UART0

1. **Initialization of UART0**

- Configure P0.0 and P0.1 as TXD0 and RXD0 by writing 01 to the corresponding bits in PINSEL0.
- Using U0LCR register, make DLAB = 1. Also, select 8-bit character length and 1 stop bit.
- Set appropriate values in U0DLL and U0DLM depending on the PCLK value and the baud rate desired. Fractional div ider can also be used to get different values of baudrate.
- **Example**, PCLK = 15MHz. For baud rate 9600, without using fractional divider register, from the baud rate formula, wehave,

$$9600 \quad \frac{15000000}{16 * (256 * U0DLM + U0DLL) * \left(1 + \frac{MulVal}{MulVal + DivAddVal}\right)}$$

On reset, MulVal = 1 and DivAddVal = 0 in the Fractional Divider Register.

- Hence, $(256 * U0DLM + U0DLL) = \frac{15000000}{16 * 9600} = 97.65$

  We can consider it to be 98 or 97. It will make the baud rate slightly less or more than 9600. This small change is tolerable. We will consider 97. Since 97 is less than 256 and register values cannot contain fractions, we will take U0DLM = 0. This will give U0DLM = 97.
- Make DLA = 0 using U0LCR register.

```
void UART0_init(void)
{
        PINSEL0 = PINSEL0 | 0x00000005;          /* Enable UART0 Rx0 and Tx0
pins of UART0 */
        U0LCR  =  0x83;     /* DLAB = 1, 1 stop bit, 8-bit character length */
        U0DLM  =  0x00;     /* For baud rate of 9600 with Pclk = 15MHz */
        U0DLL = 0x61;       /* We get these values of U0DLL and
U0DLM from formula */
        U0LCR = 0x03;       /* DLAB = 0 */
}
```

2. Receiving character

- Monitor the RDR bit in U0LSR register to see if valid data is available in U0RBR register.

```
unsigned char UART0_RxChar(void) /*A function to receive a byte on UART0 */
{
        While ((U0LSR & 0x01) == 0);     /*Wait till RDR bit becomes 1 which tells
that receiver contains valid data */
        return U0RBR;
}
```

3. Transmitting character

- Monitor the THRE bit in U0LSR register. When this bit becomes 1, it indicates that U0THR register is empty and the transmission is completed.

```
        void UART0_TxChar(char ch) /*A function to send a byte on UART0*/
{
U0THR = ch;
While ((U0LSR & 0x40) == 0); /* Wait till THRE bit becomes 1 which tells that
transmission is completed */
        }
```

PROGRAM

```
/*      UART0 in LPC2148(ARM7)*/
#include <lpc214x.h> #include <stdint.h>
#include "UART.h"
int main(void)
{
        char receive; UART0_init();  while(1)

        {
                receive = UART0_RxChar();
                UART0_SendString("Received:");
                UART0_TxChar(receive);
                UART0_SendString("\r\n");
        }
}
```

8. Explain the functional block diagram of ARM 9PROCESSOR

This family enables single processor solution for microcontroller, DSP & JAVA applications, offering sav ings in chip area & complex ity, power consumption & tim e to market.

- ARM9 – enhanced processors are wel l suited for applications requiring a mix of DSP+ Microcontrol ler performance

FEATURES OF ARM9

- Pipeline Depth: 5 stage (Fetch, Decode, Execute, Decode, Write)
- Operating frequency: 150 MHz
- Power Consumption: 0.19 mW/MHz
- MIPS/MHz: 1.1
- Architecture used: Harvard
- MMU/MPU: Present
- Cache Memory: Present (separate 16k/8k)
- ARM/ Thumb Instruction: Support both

- ISA (Instruction Set Architecture): V5T(ARM926EJ-S)
- 31 (32-Bit size) Registers
- 32-bit ALU & Barrel Shifter
- Enhanced 32- bit MAC block
- Memory Controller
  Memory operations are controlled by MMU orMPU
  - MMU:
    - Provides Virtual Memory Support
    - Fast Context Switching Extensions
  - MPU:
    - Enables memory protection & bounding
    - Sand – boxing of applications
- Flexible Cache Design (sizes can be 4KB to 128KB)
- Flexible Core Design
- DSP Enhancements: (very important)
- Single cycle 32x16 multiplier Implementation
- Speed up all the multiply instructions
- New 32x16 & 16x16 multiply instructions
- Allows independent access to 16 bit halves of registers
- ARM ISA supports 32x32 multiply instruction
- Saturating Arithmetic (QADD, QSUB)
- Count leading zero for factor Division

Applications of ARM9

1. Consumer type: Smart phones, PDA, Set-Top box, Electronics Toys, Digital Cameras, etc.
2. Networking type: Wireless LAN, 802.11, Bluetooth, etc.

3. Automatic: Power Train, ABS, Navigation, etc.
4. Embedded USB controllers, Bluetooth controllers, Medical scanners, etc.
5. Storage: HDD controllers, solid state drivers etc.

1. ARM920T PROCESSOR

The ARM920T processor is a member of the ARM9TDMI family of general-purpose microprocessors, which includes:

- ARM9TDMI (core)
- ARM940T (core plus cache and protection unit)
- ARM920T (core plus cache and MMU).

2. ARM9TDMI (CORE)

The ARM9TDMI processor core is a Harvard architecture dev ice implemented using a five-stage pipeline consisting of Fetch, Decode, Execute, Memory, and Write stages. It can be prov ided as a standalone core that can be embedded into more complex devices. The standalone core has a simple bus interface that allows you to design your own caches and memory systems around it.

The ARM9TDMI family of microprocessors supports both the 32-bit ARM and 16-bit Thumb instruction sets, allowing you to trade-off between high performance and high code density.

3. ARM920T (CORE PLUS CACHE AND MMU).

The ARM920T processor is a Harvard cache architecture processor that is targeted at multi programmer applications where full memory management, high performance, and low power are all- important. The separate instruction and data caches in this design are 16KB each in size, with an 8- word line length. The ARM920T processor implements an enhanced ARM architecture v4 MMU to provide translation and access permission checks for instruction and data addresses.

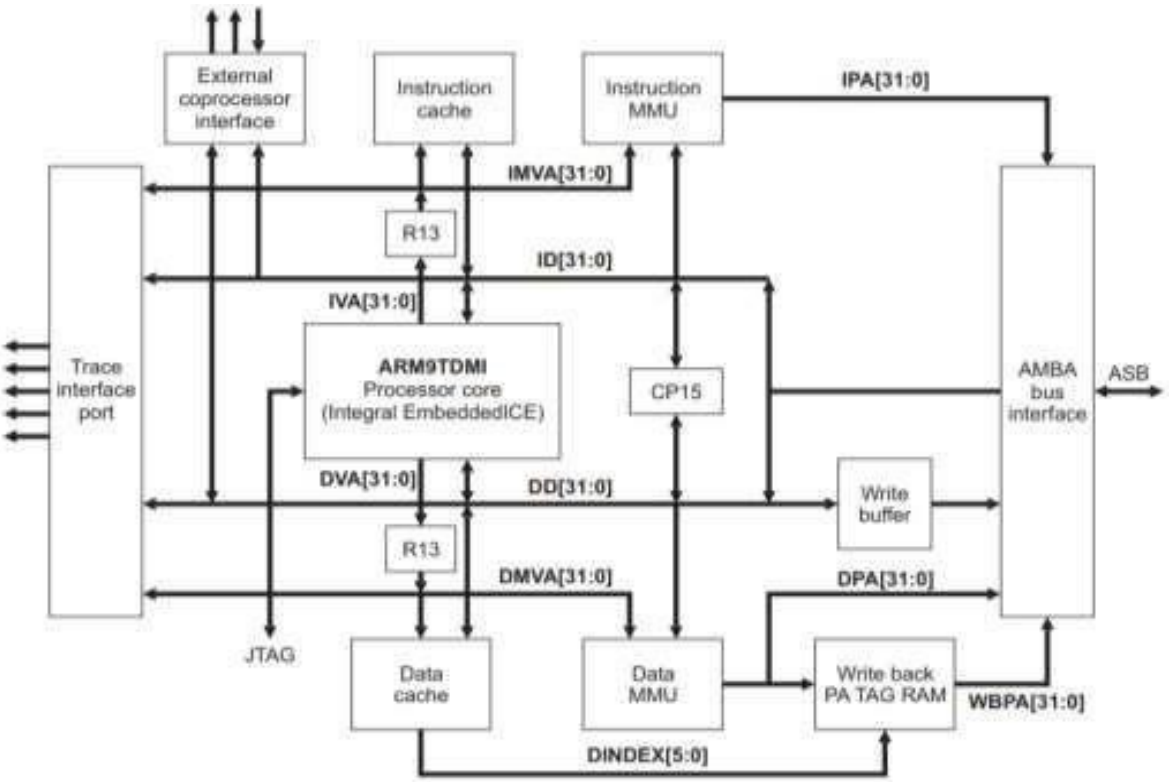
The ARM920T processor supports the ARM debug architecture and includes logic to assist in both hardware and software debug. The ARM920T processor also includes support for coprocessors, exporting the instruction and data buses along with simple handshaking signals.

The ARM920T interface to the rest of the system is over unified address and data buses. This

interface enables implementation of either an Advanced Microcontroller Bus Architecture (AMBA), Advanced System Bus (ASB) or Advanced High- performance Bus (AHB) bus scheme either as a fully - compliant AMBA bus master, or as a slave for production test. The ARM920T processor also has a Tracking ICE mode which allows an approach similar to a conventional ICE mode of operation.

The ARM920T processor supports the addition of an Embedded Trace Macrocell (ETM) for real-time tracing of instructions and data.

ARM920T FUNCTIONALBLOCKDIAGRAM



PRO(

The ARM920T processor incorporates the ARM9TDMI integer core, which implements the ARM architecture v4T. It executes the ARM and Thumb instruction sets, and includes Embedded ICE JTAG software debugfeatures.

The programmer's model of the ARM920T processor consists of the programmer's model of the ARM9TDMI core with the following additions and modifications:

The ARM920T processor incorporates two coprocessors:

- CP14, which allows software access to the debug communications channel. The registers defined in CP14 can be accessed using MCR and MRC instructions.
- The system control coprocessor, CP15, which prov ides additional registers that are used to configure and control the caches, MMU, protection system, the clocking mode, and other system options of the ARM920T, such as big or little-endian operation. The registers defined in CP15 can be accessed using MCR and MRC instructions.

The ARM920T processor also features an external coprocessor interface that allows the attachment of a closely-coupled coprocessor on the same chip, for example, a floating-point unit. Registers and operations prov ided by any coprocessors attached to the external coprocessor interface can be accessed using appropriate coprocessor instructions.

Memory accesses for instruction fetches and data loads and stores can be cached or buffered.

The MMU page tables that reside in main memory describe the v irtual to physica l address mapping, access permissions, and cache and write buffer configuration. These are created by the operating system software and accessed automatically by the ARM920T MMU hardware whenever an access causes a TLB miss.

The ARM920T has a Trace Interface Port that allows the use of Trace hardware and tools for real- time tracing of instructions and data.

2. ABOUT THE ARM9TDMI PROGRAMMER'S MODEL

The ARM9TDMI processor core implements ARM architecture v4T, and executes the ARM 32-bit instruction set and the compressed Thumb 16-bit instruction set.

ARMv4T specifies a sma ll number of implementation options. The options selected in the ARM9TDMI implementation are listed. For comparison, the options selected for the ARM7TDMI implementation are also shown in table 2.12.

Processor core	Architecture	Data Abort model	Value stored by direct STR, STRT, and STM of PC
ARM7TDMI	ARMv4T	Base updated	Address of instruction + 12
ARM9TDMI	ARMv4T	Base restored	Address of instruction + 12

Table 2.12 Comparison of ARM9TDMI and ARM7TDMI implementation

The ARM9TDMI is code-compatible with the ARM7TDMI, with two exceptions:

- The ARM9TDMI core implements the base restored Data Abort model. This significantly simplifies the software Data Abort handler.
- The ARM9TDMI fully implements the instruction set extension spaces added to the ARM (32-bit) instruction set in ARMv4 and ARMv4T.

These differences are explained in more detail in the following sections:

- Data Abort model
- Instruction set extension spaces.

Data Abort model

The base restored Data Abort model differs from the base updated Data Abort model implemented by ARM7TDMI.

The difference in the Data Abort models affects only a very small section of operating system code, the Data Abort handler. It does not affect user code. With the base restored Data Abort model, when a

Data Abort exception occurs during the execution of a memory access instruction, the base register is always restored by the processor hardware to the value the register contained before theinstruction

Y

was executed. This removes the requirement for the Data Abort handler to unwind any base register update that might have been specified by the aborted instruction.

INSTRUCTION SET EXTENSION SPACES.

All ARM processors implement the undefined instruction space as one of the entry mechanisms for the undefined instruction exception.

ARMv4 and ARMv4T also introduce a number of instruction set extension spaces to the ARM instruction set. These are:

- arithmetic instruction extension space
- control instruction extension space
- coprocessor instruction extension space
- load/store instruction extension space.

Instructions in these spaces are undefined, and cause an undefined instruction exception. The ARM9TDMI core fully implements all the instruction set extension spaces defined in ARMv4T as undefined instructions, allowing emulation of future instruction set additions.

ADDRESS IN ARM 920T

Three distinct types of address exist in an ARM920T system:

- Virtual Address (VA)
- Modified Virtual Address(MVA)
- Physical Address (PA).

ABOUT THE MMU

ARM920T processor implements an enhanced ARM architecture v4 MMU to provide translation and access permission checks for the instruction and data address ports of the ARM9TDMI core. The MMU is controlled from a single set of two-level page tables stored in main memory, that are
enabled by the M bit in CP15 register 1,

prov iding a single address translation and protection scheme. You can independently lock and flush the instruction and data TLBs in the MMU.

The MMU features are:

- standard ARMv4 MMU mapping sizes, domains, and access protection scheme
- mapping sizes are 1MB (sections), 64KB (large pages), 4KB (smal]pages), and 1KB (tiny pages)
- access permissions for sections
- access permissions for large pages and smal] pages can be specified separately for each quarter of the page (these quarters are cal]ed subpages)
- 16 domains implemented inhardware
- 64 entry instruction TLB and 64 entry data TLB
- hardware page table walks
- round-robin replacement algorithm (also cal]ed cyclic)
- invalidate whole TLB, using CP15 register 8
- invalidate TLB entry, selected by MVA, using CP15 register 8
- independent lockdown of instruction TLB and data TLB, using CP15 register 10.

ARM CORTEX M3

The ARM Cortex-M3 processor, the first of the Cortex generation of processors released by ARM in 2006, was primarily designed to target the 32-bit m icrocontroller market.

- The Cortex-M3 processor prov ides excellent performance at low gate count and comes with many new features previously available only in high-end processors.

FEATURES OF ARM CORTEX M3

Architecture	Armv7-M
Bus Interface	3x AMBA AHB-Lite interface (Harvard bus architecture) AMBA ATB interface for CoreSight debug components
ISA Support	Thumb/Thumb-2 subset
Pipeline	Three-stage
Memory Protection	Optional 8 region MPU with sub regions and background region
Bit Manipulation	Integrated Bit-field Processing Instructions and Bus Level Bit Banding
Interrupts	Non-maskable Interrupt (NMI) + 1 to 240 physical interrupts
Interrupt Priority Levels	8 to 256 priority levels
Wake-up Interrupt Controller	Optional
Enhanced Instructions	Hardware Divide (2-12 Cycles), Single-Cycle (32x32) Multiply, Saturated Adjustment Support
Sleep Modes	Integrated WFI and WFE Instructions and Sleep On Exit capability. Sleep and Deep Sleep Signals Optional Retention Mode with Arm Power Management Kit
Debug	Optional JTAG and Serial Wire Debug ports. Up to 8 Breakpoints and 4 Watchpoints
Trace	Optional Instruction (ETM), Data Trace (DWT), and Instrumentation Trace(ITM)

Table 2.13 ARM Cortex M3 features

Low Power

- 32-bit Cortex-M3 designed for low power operation, enabling longer battery life, especially critical in portable products including wireless networking applications
- High power efficiency with Thumb-2 instructionset
- Small core footprint with integrated power modesupport

High Performance

- Cortex-M3 delivering 1.25 DMIPS/MHz
- Separate data and instruction bus
- High code density and performance with Thumb-2 instruction set
- Excellent clock per instruction ratio
- Nested Vectored Interrupt Controller (NVIC) for outstanding interrupt handling
- Superior math capability

Thumb-2 Instruction Set Architecture (ISA)

Cortex-M3 supports 16- and 32-bit instructions available in the Thumb-2 instruction set. Both can be mixed without extra complex ity and without reducing the Cortex-M3 performance. Hardware div ide instructions and a number of multiply instructions give EFM32 users high data-crunching throughput.

3-stage Pipeline Core Based on Harvard Architecture

The ARM Cortex-M3 3-stage pipeline includes instruction fetch, instruction decode and instruction execution. Cortex-M3 also has separate buses for instructions and data. The Harvard architecture reduces bottlenecks common to shared data- and instruction buses. Quickly Serv icing Critical Tasks and Interrupts. From the low energy modes,EFM32's Cortex-M3 is active within 2 µs and delivers 1.25 DMIPS/MHz on the Dhrystone 2.1 Benchmark.

Nested vectored interrupt controller (NVIC)

- Low latency, low jitter interrupts response
- No need for assembly programming

The **NVIC** (Nested Vectored Interrupt Controller) is an integral part of the Cortex-M3 processor and ensures outstanding interrupt handling abilities. It is possible configure up to 240 physical interrupts with 1-256 levels of priority, and Non- Maskable Interrupts further increase interrupt handling. For embedded systems this enhanced determ inism makes it possible to handle critical tasks in a known number of cycles.

Reducing the 32-bit Footprint

The Cortex-M3 has a small footprint which reduces system cost. High 32-bit performance reduces an application's active periods, the periods where the CPU is handling data. Reducing the active periods increases the application's battery lifetime significantly, and the EFM32 can spend most of the time in the efficient low energy modes.

ARM CORTEX M3 ARCHITECTURE

The ARM Cortex-M3 processor has been designed 'from the ground up' to prov ide optimal performance and power consumption within a minimal memory system.

To achieve this the core executes only the Thumb-2 instruction set.

The design is based on a 3-stage pipeline Harvard architecture that maxim izes memory utilization through the support of unaligned date storage, and single cycle atomic bit manipulation.

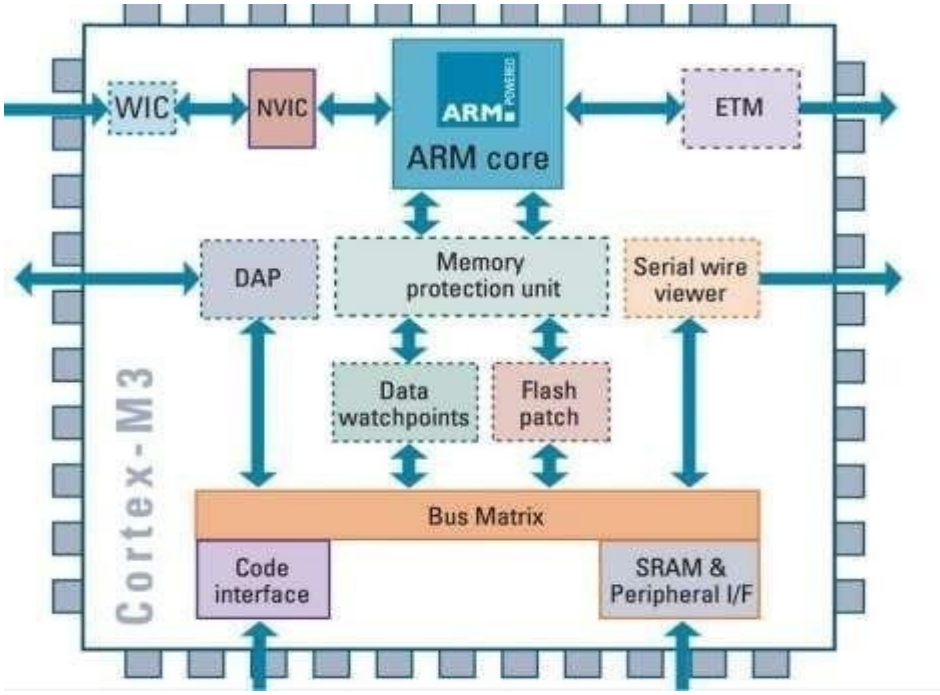


Figure 2.42 Cortex M3 architecture

The highly rev ised architecture implements hardware div ide and single-cycle multiply. The ARM Cortex-M3 uses 33k gates for the processing core and 60k gates total, including many closed system peripherals.

The ARM Cortex-M3 processor reduces the number of pins required for debug from five to one, by implementing a Single Wire Debug.

For system trace, the processor integrates an optional ETM alongside data watch points that can be configured to trigger on specific system events.

To enable simple and cost-effective profiling of these system events a SWV (Seria l Wire Viewer) can export streams of standard ACSII data through a single pin.

Flash Patch technology offers device and system developers the ability to patch errors in code from ROM to SRAM or Flash during both debug and run-time.

The Cortex-M3 processor integrates the core with a configurable interrupt controller to improve interrupt processing performance. In its standard im plementation the NVIC (Nested Vectored Interrupt Controller) supplies a NMI (Non-Maskable Interrupt) plus 32 general purpose physical interrupts with 8 levels of pre-emption priority, however through simple

synthesis choices the controller can be configured down to a single physical interrupt or up to 244.

The number of levels of preemptive priority can be configured at synthesis up to 255. Faster execution of ISR (Interrupt Serv ice Routines) is accomplished by using hardware stacking of registers and the ability to exit and restart load-store multiple executions.

This means that no assembler stubs are required to handle the movement of registers. Moving between active and pending interrupts has been simplified through the use of Tail-Chaining technology to replace serial stack Pop and Push actions that normally take over 30 clock cycles with a simple six cycle instruction fetch.

To enhance low power designs the NVIC integrates three sleep modes, including a Deep Sleep function that may be exported to other system components to enable the entire device to be rapidly powered down.

The ARM Cortex-M3 processor has two optional components, the MPU (Memory Protection Unit) and the ETM (Embedded Trace Macrocell). The fine grain MPU design enables applications to implement security priv ilege levels, separating code, data and stack on a task-by-task basis.

ARM CORTEX M3 MCU

The Cortex-M3 Processor versus Cortex-M3-Based MCUs

The Cortex-M3 processor is the central processing unit (CPU) of a m icrocontroller chip.

In addition, a number of other components are required for the whole Cortex-M3 processor-based m icrocontroller. After chip manufacturers license the Cortex-M3 processor, they can put the Cortex-M3 processor in their silicon designs, adding memory, peripherals, input/output (I/O), and other features. Cortex-M3 processor- based chips from different manufacturers will have different memory sizes, types, peripherals, and features.

ARM Cortex-M3 LPC1768 Microcontroller

The LPC1768 is m icrocontroller belongs to Cortex-M3 core whose architecture is shown in the figure 2.43. LPC1768 is m ixed signal processor from NXP Semiconductor. The Cortex-M3 offers many new features including **Thumb-2 Instruction** Set and **very low power** consumption,

low interrupt latency etc.

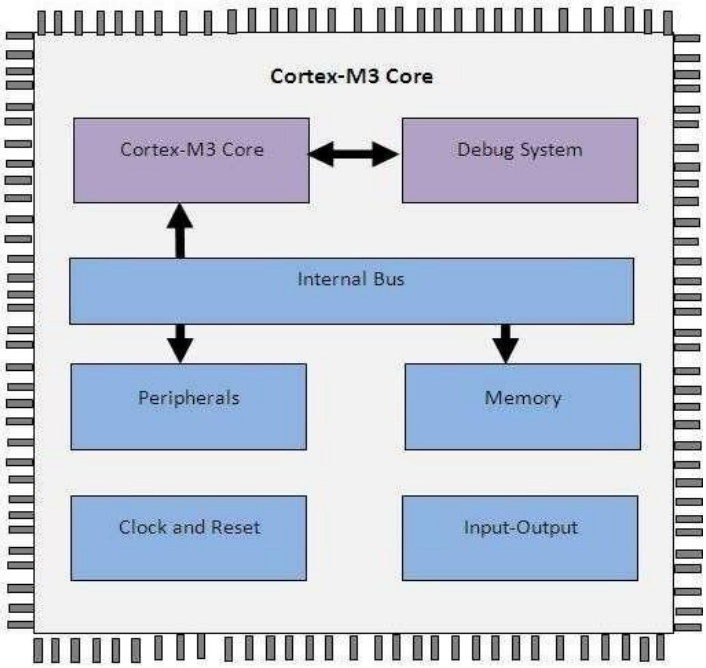


Figure 2.43 ARM Cortex-M3 LPC1768Microcontroller architecture