UNIT-5

ARM ARCHITECTURES AND PROCESSOR

Introduction:-

The ARM architecture Processor is an Advanced Reduced instruction set computing [RISC] machine & it's a 32-bit reduced instruction set computer RISC microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST microelectronics, Motorola and soon.

ARM Architecture :-

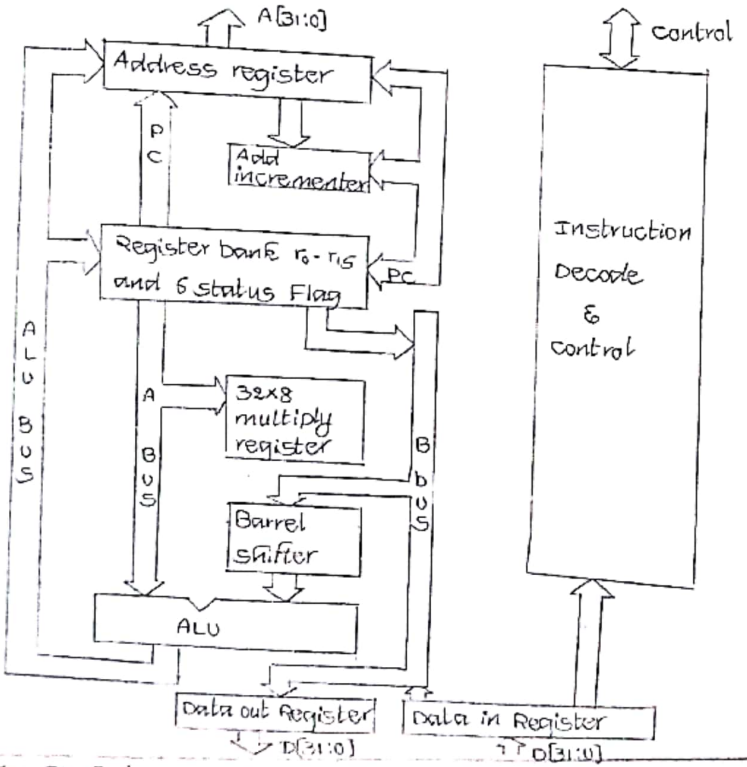


Fig:- Block diagram of ARM Architecture

→ The block diagram of ARM architecture is shown in above Figure.

→ The ARM architecture consists of following

1. Arithematic logic unit
2. multiplier
3. Barrel shifter
4. control unit
5. Register bank

multiplexer:- Several multiplexers are accustomed to the management operation of the Processor buses

Arithematic logic unit (ALU):- The ALU has two 32-bit inputs The Primary comes From the register bank. whereas the other comes From the shifter. Status register Flags modified by the ALU outputs The V-bit output goes to the V-Flag as well as the count goes to C Flag.

Barrel shifter:- The barrel shifter Features a 32-bit input to be shifted. This i/p is coming back From the register bank or it might be immediate data. The shifter has different control i/p's coming back From the instruction register.

incrementer:-

For load & store instructions, The incremented updates the contents of the address register before the Processor core reads or writes the next register value From or to the consecutive memory location.

Scanned with OKEN Scanner

Address Register: This fields the address generated by the load & store instructions & places it on the address bus

Instruction Decoder: It decodes the instruction opcode read from the memory & then the instruction is executed.

Register bank: This is a bank of 32-bit registers used for storing data items.

Features of ARM

1. Thumb set designed for 16-bit word lengths & instructions, which internally executes by same 32-bit core.
2. Most operations are executed over registers.
3. All instructions can be conditional.
4. It supports 25 different instruction.
5. ARM Provides no explicit return instruction.
6. The ARM architecture has a large variety of addressing modes.
7. many thumb data Processing instructions use a 2-address format
8. Jazelle instruction set : Introduces technological infrastructure for running Java code.

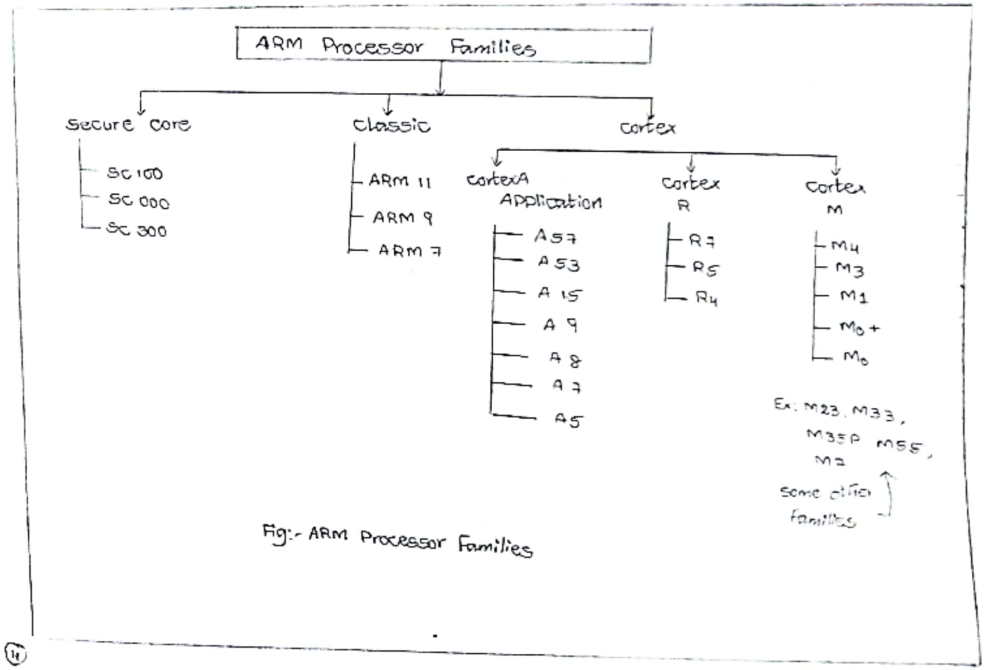


Fig:- ARM Processor Families

Scanned with OKEN Scanner

ARM Cortex-M series Family

ARM core	M0	M1	M3	M11
Year of establishment	2009	2007	2007	2010
ARM architecture	ARMV6-M	ARMV6-M	ARMV7-M	ARMV7E-M
memory architecture	Von-Neuman	Von-Neuman	Harvard	Harvard
No. of instruction Pipeline stages	3	3	3	3
Interrupts	1 to 3 [NMI]	1 to 32 [NMI]	1 to 240 [NMI]	1 to 240 [NMI]
Interrupt latency	16 cycles	23 for NMI, 26 for IRQ	12 cycles	12 cycles
microcontroller chips based on	Toshiba TX00, NXP LPC 1100, Cypress 4M, PSOC 4	xilinx spartan3 Altera cyclone-III, IGLOOle	NXPLPC 1300, Toshiba TX03, Actel Smart Fusan	Toshiba XH, NXPV3, Cypress, PSOC 6

ARM Cortex-M3 Processor Functional description

→The Cortex-M3 Processor Features:

* A low gate count Processor core, with low latency interrupt Processing that has:
  → A subset of the thumb instruction set defined in the ARMv7-M architecture Reference manual.
  → Banked stack Pointer [SP]

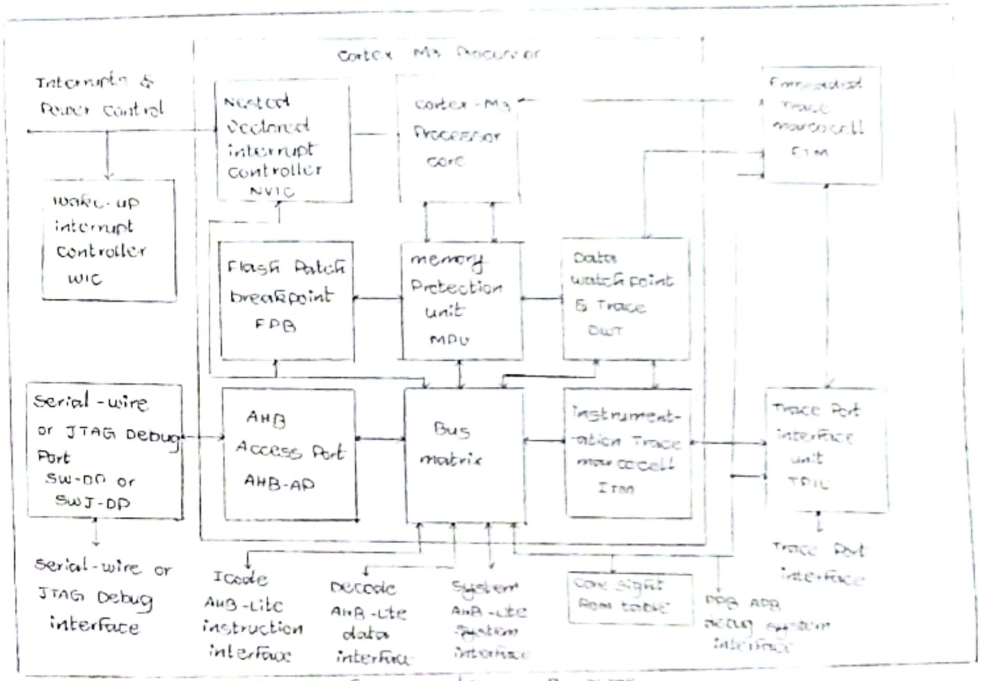


Fig:- Block diagram of ARM cortex M3 Processor

→ Hardware divide instructions. SDIV & UDIV.
→ Handler & Thread modes,
→ Thumb & debug states

Nested Vectored Interrupt Controller [NVIC]: closely integrated with the Processor core to achieve low latency interrupt Processing Features include:

→ External interrupts, configurable from 1 to 240
→ Bits of Priority, configurable from 3 to 8.
→ Dynamic reprioritization of interrupts.
→ Priority Grouping
→ Optional wake-up interrupt controller [WIC], Providing ultra-low Power sleep mode support

Memory Protection unit [MPU]:- An optional MPU for memory Protection, including

→ Eight memory regions.
→ Sub Region Disable [SRD], enabling efficient use of memory regions.
→ The ability to enable a background region that implements the default memory map attributes

Bus interfaces:

→ Three advanced high-Performance bus-lite [AHB-Lite], interfaces: Icode, Dcode & system bus interfaces.
→ Private Peripheral Bus [PPB] based on Advanced Peripheral Bus [APB] interface.
→ Bit band support that includes atomic bit band write & read operations,

→ Memory access alignment.
→ write buffer for buffering of write data

* low-cost debug solution that Features:

→ Debug access to all memory & registers in the system.
→ Serial wire debug Port [SWDP] (or) serial wire JTAG Debug Port [SWJ-DP] debug access. or both. Optional Flash Patch & Breakpoints [FPB]
→ optional Data watchpoint & Trace [DWT] unit for implementing watchpoints, data tracing & system Profiling.
→ Optional instrumentation Trace marcocell ITM for Support of Printf style debugging.
→ Optional Trace Port interface unit [TPIU] for bridging to a trace Port analyzer TPA, including single wire output [SWO] mode.
→ Optional Embedded Trace macrocell [ETM] for instrumentation trace.

ETM interface

→ AHB Trace marcocell interface
→ Debug Port AHB-AP interface
→ Bus interfaces

Bus interfaces: four external advanced high Performance bus AHB-lite bus interface

Icode memory interface:

Instruction fetches from code memory space 0x00000000

Scanned with OKEN Scanner

to 0xFFFFFFFF are performed over this 32-bit AHB-Lite bus.

→ The debugger cannot access their interface.

Dcode memory interface:

Data & debug access to code memory space 0x00000000 to 0x1FFFFFFF are performed over their 32 bit AHB-LITE Bus. Core & data accesses have a high priority than debug access on this bus.

Modes of operation & execution / Programming models

The Cortex-M3 Processor supports Thread and Handler operating modes, and may be run in Thumb or Debug operating states. In addition, the Processor can limit or exclude access to some resources by executing code in Priviledged or unprivileged mode.

Operating modes:

The conditions which cause the Processor to enter Thread or Handler mode are as follows:

Thread :- The Processor enters thread mode on Reset, or as a Result of an exception return. Privileged & unpriviledged code can run in thread mode.

Handler :- The Processor enters Handler mode as a result of an exception. All code is Priviledged in Handler mode.

Operating states:

The Processor can operate in thumb or debug state

Thumb :- This is normal execution running 16-bit & 32 bit halfword aligned Thumb instructions

Debug :- This is the state when the Processor in halting debug.

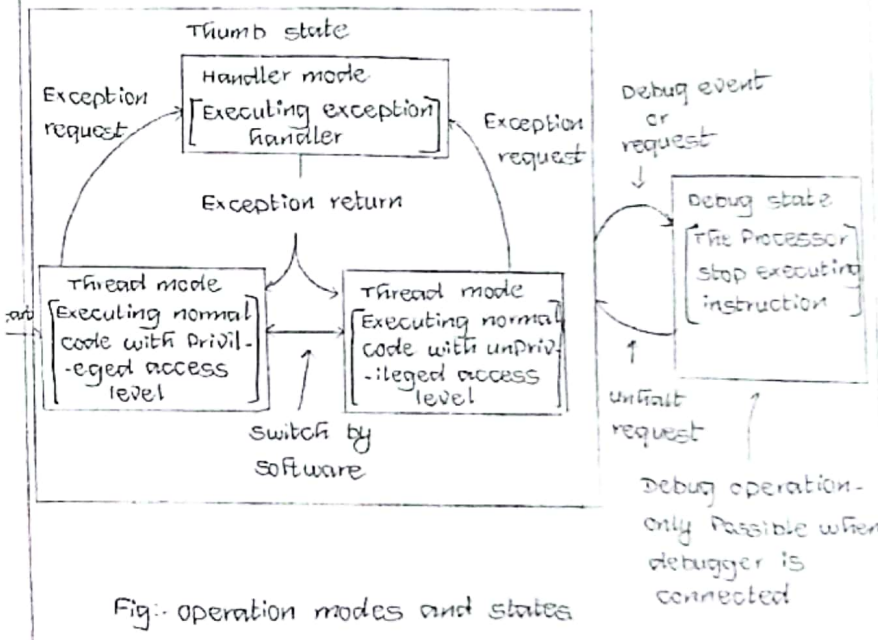


Fig:- operation modes and states

Registers:

Cortex-M3 Processor have a no of registers inside the Processor core to Perform data Processing & control. Most of the registers are grouped in a unit

Scanned with OKEN Scanner

a register bank. ARM architecture, if data is in memory is to be Processed. It has to be loaded From the memory to registers in register bank Processed inside the Processor & written back to memory if needed. This is commonly called 'load-store architecture.

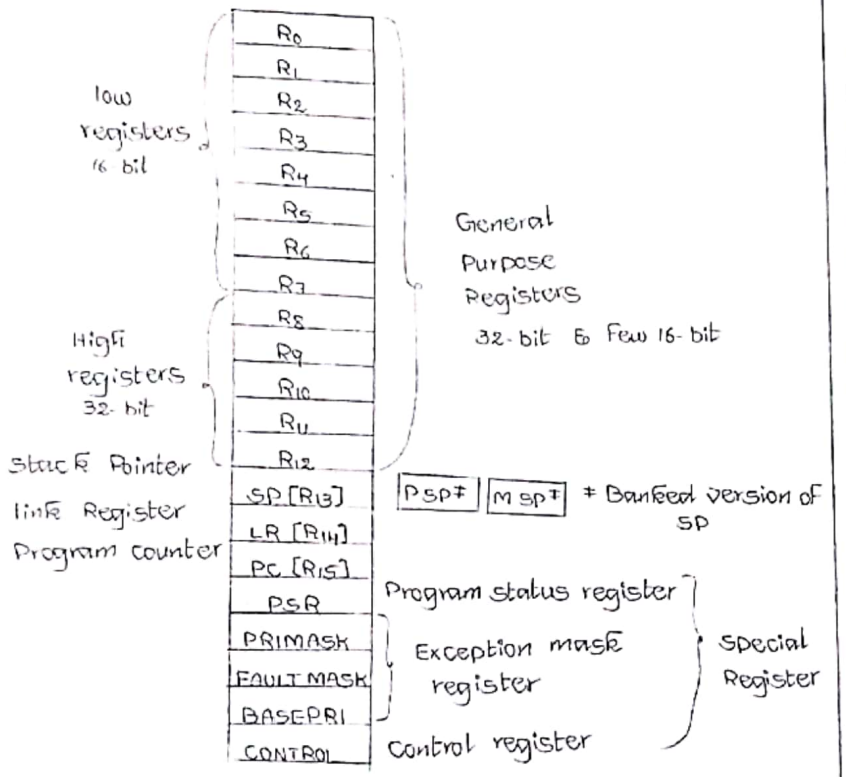


Fig:- Register organization

General Purpose Registers :-

R_0 - R_{12} are 32-bit general Purpose registers for data operations.

Stack Pointer :- The stack Pointer [SP] is a Register R_{13}. In Thread mode, bit[1] of the CONTROL register indicates the stack Pointer to use:

→ 0 = main stack Pointer MSP. This is the reset value

→ 1 = Process stack Pointer PSP.

On reset, the Processor loads the MSP with the value From address 0x00000000.

link Register :- The link register [LR] is register R_{14}. It stores the return information for subroutines, Function calls, & exceptions. On reset, the Processor sets the LR value to 0xFFFFFFFF.

Program Counter :- The Program counter PC is register R_{15}. It contains the current Program address. On reset, the Processor loads the PC with the value of the reset Vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset & must be 1.

Special Registers :- These registers contain the Processor status & define the operation states & interrupt/exception.

Special registers are most memory mapped & can be accessed using special registers access instructions such as MSR and MRS

MRS <reg>, <special_reg> : load special register into reg.

MSR <special_reg>, <reg>

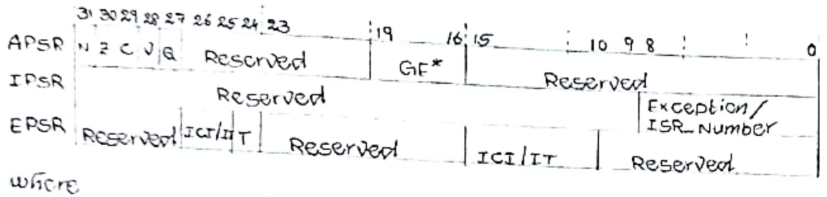
Scanned with OKEN Scanner

Program status Register:

The Program status Register PSR combines

1. Application Program status Register [APSR]
2. Interrupt Program status Register [IPSR]
3. Execution Program status Register [EPSR]

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:



where

N → Negative Flag

Z → zero Flag

C → carry Flag / Borrow Flag

V → overFlow Flag

Q → Stricky Saturation Flag

GE → Greater than or equal Flags for each byte lane.

ICI/IT → Interrupt continuable instruction/

T → Thumb state

Exception → indicate which exception the Processor is handling

Control Register:- The CONTROL register controls the stack used an Privilege level for software execution when the Processor is in thread mode. The bit assignments are:

Bits	Name	Function
[31:2]	-	Reserved
[1]	SPSEL	Defines the currently active stack pointer: In Handler mode this bit reads as zero & ignores write. The Cortex-M3 updates this bit automatically on execution return 0 = MSP is the current stack pointer 1 = PSP is the current stack pointer
[0]	nPRIV	Defines the thread mode Privilege level: 0 = Privileged 1 = unprivileged

Stack:-and stack Pointer:-

The Processor uses a Full descending stack. This means the stack Pointer holds the address of the last stacked item in memory. when the Processor pushes a new item onto the stack, it decrements the stackpointer & then writes the item to the new memory location. The Processor implements 2 stacks, the main stack & Process stack, with a Pointer for each field in independent registers.

In Thread mode, the CONTROL register controls whether the Processor uses the main stack or the Process stack. In Handler mode, the Processor always uses the main stack The options for Processor operations are:

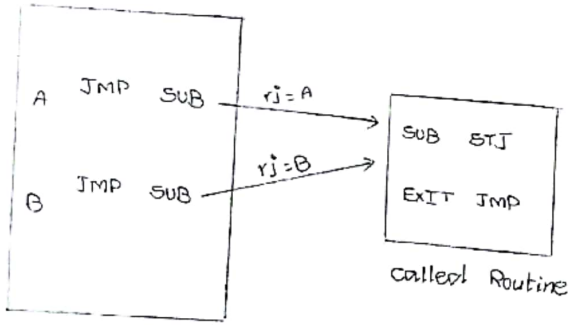
Scanned with OKEN Scanner

Processor mode	used to execute	Privilege level for Software execution	Stack used
Thread	Applications	Privileged or unprivileged	main stack or Process stack
Handler	Exception Handlers	Always Privileged	main stack

Subroutines and Parameter Passing

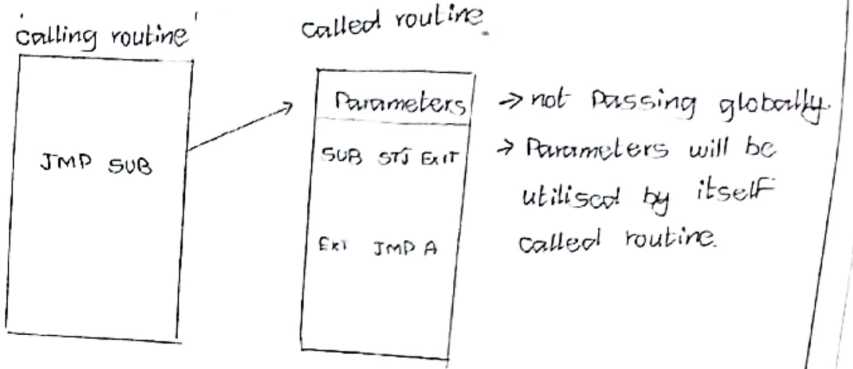
Every Program requires input data & a proper format for output data results. This code is given very similar from Program to Program.

So within the code same Program should be written in several places.



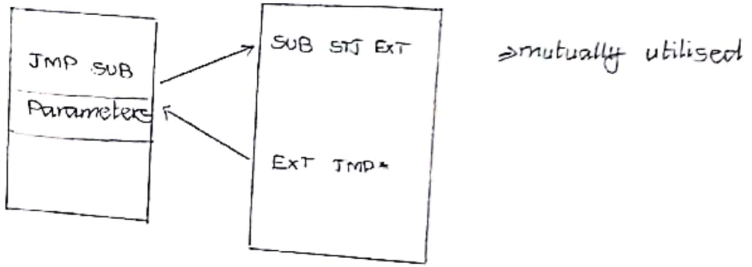

```
SUMMER   STJ  EXIT SUM     : Save Return address
         ENTA 0            : SUM = 0
         ENT1 9            : I1 = 9, ..... 0
LOOP     ADD  INCOME I1    : Add income(I1)
         DEC  I1
         J1NN LOOP         : check for end of loop
EXIT_SUM JMP  *            This instruction modified
```

→ Parameters Passed in Called routine.

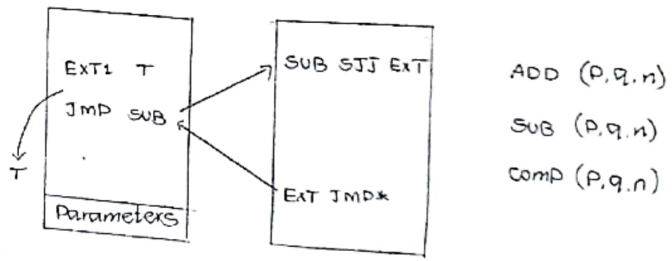


→ Parameter Passing is calling routine.

Forward Reference

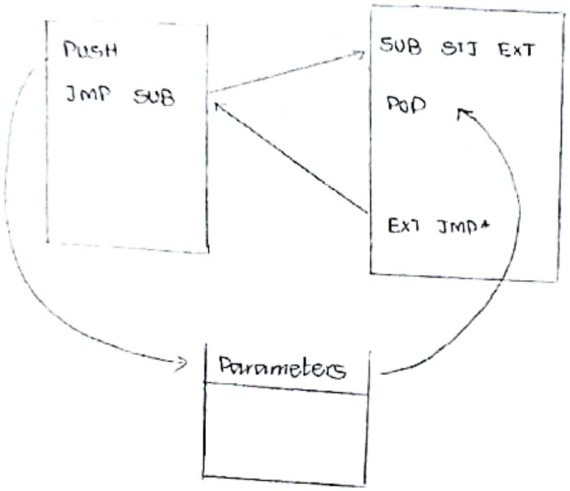


→ Passing Parameters in a table



Scanned with OKEN Scanner

Passing Parameter in stack -



Types of Parallel Ports.

1. Parallel Port one bit input :- complete revolution of a wheel.
2. Parallel one bit output :- Pwm output for a DAC
3. Parallel Port multi-bit input :- ADC input From liquid level measurement.
4. Parallel Port multi-bit output :- LCD display matrix unit in a cellular phone.

Nested Vectored interrupt Controller

This section describes the NVIC & the registers it uses. The NVIC supports:

→ An implementation-defined number of interrupts, in the range 1-240 interrupts

→ level and Pulse detection of interrupt signals
→ Dynamic reprioritization of interrupts
→ Grouping of Priority values into group Priority and subpriority Fields.
→ interrupt tail chaining
→ An external Non-maskable interrupt NMI
→ optional WIC, Providing ultra-low Power sleep mode Support.

The Processor automatically stacks its state on exception entry & unstacks this state on exception exit, with no instruction overhead This Provides low latency exception handling. The hardware implement -ation of the NVIC registers.

1. Interrupt Set-enable registers
2. Interrupt Clear-enable Registers
3. Interrupt Set-Pending Registers.
4. Interrupt clear-Pending Registers
5. Interrupt Active bit Registers.
6. Interrupt Priority Registers
7. Software trigger interrupt Registers.

Programming model

The ICTR characteristics are: Interrupt control type register

Purpose :- Shows the no. of interrupts lines that the NVIC supports

Usage constraints :- There are no usage constraints

Scanned with OKEN Scanner

Configurations :- This register is available in all Processor configurations

Attributes :-

Bits	Name	Function
[31:4]	-	Reserved
3:0	INTILINESNUM	Total number of interrupt lines in groups of 32

b_{0000} = 0 ... 32

b_{0001} = 33 64

b_{0010} = 65 96

b_{0011} = 97 ... 128

b_{0100} = 129 ... 160

b_{0101} = 161 ... 192

b_{0110} = 193 ... 224

b_{0111} = 225 ... 256

Functional description

The NVIC supports upto 240 interrupts each with upto 256 levels of Priority. You can change the Priority of an interrupt dynamically. The NVIC & the Processor core interface are closely coupled, to enable low latency interrupt processing & efficient Processing of late arriving interrupts. The NVIC maintains knowledge of the stacked, or nested, interrupts to enable tail-chaining of interrupts.

You can only fully access the NVIC from Privileged mode, but you can cause interrupts to enter a Pending state in user mode if you enable this capability using the Configuration control Register. Any other user mode access causes a bus Fault.

You can access all NVIC registers using byte, HalfWord, & word accesses unless otherwise stated. NVIC registers are located within the SCS.

Processor ~~Hand~~ exception Handling is described in exceptions.

low Power modes

Your implementation can include a wake-up interrupt controller WIC. This enables Processor & NVIC to be put into a very low Power sleep mode leaving the WIC to identify & Prioritize interrupts.

The Processor Fully implements the wait for interrupt [WFI], wait for event [WFE] & the send event [SEV] instructions. In addition, the Processor also supports the use of SLEEPONEXIT, that causes the Processor core to enter sleep mode when it returns from an exception Handler to Thread mode.

level vs Pulse interrupts

The Processor supports both level & Pulse interrupts. A level interrupt is held asserted until it is cleared by

Scanned with OKEN Scanner

the ISR accessing the device. A Pulse interrupt is a variant of an edge model.

For level interrupts, if the signal is not deasserted before the return from the interrupt routine, the interrupt again enters the pending state & reactivates. This is particularly useful for FIFO & buffer based devices bez it ensures that they drain either by a single ISR or by repeated invocations, with no extra work. This means that the device holds the signal in assert until the device is empty.

Pulse interrupts are mostly used for external signals & for rate or repeat signals.

Scanned with OKEN Scanner