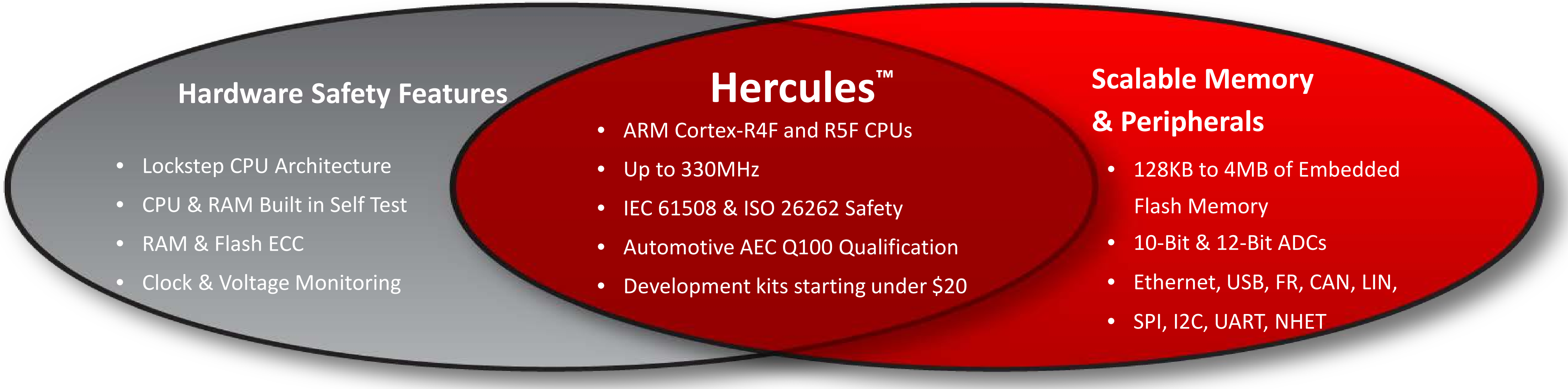
Hercules™ ARM® Cortex®-R4 System Architecture

Processor Overview

What is Hercules™?

TI's 32-bit ARM® Cortex™-R4/R5 MCU family for Industrial, Automotive, and Transportation Safety



Hardware Safety Features

- Lockstep CPU Architecture
- CPU & RAM Built in Self Test
- RAM & Flash ECC
- Clock & Voltage Monitoring

Hercules™

- ARM Cortex-R4F and R5F CPUs
- Up to 330MHz
- IEC 61508 & ISO 26262 Safety
- Automotive AEC Q100 Qualification
- Development kits starting under $20

Scalable Memory & Peripherals

- 128KB to 4MB of Embedded Flash Memory
- 10-Bit & 12-Bit ADCs
- Ethernet, USB, FR, CAN, LIN,
- SPI, I2C, UART, NHET

Broad Safety MCU Portfolio

- CPU performance from 80MHz to 330MHz
- Flash memory options ranging from 128KB to 4MB

High Reliability

- Proven hardware based on 20+ years of TI safety-critical system expertise
- Development flow refined for high quality & low DPPM

Reduce Systematic and Random Faults

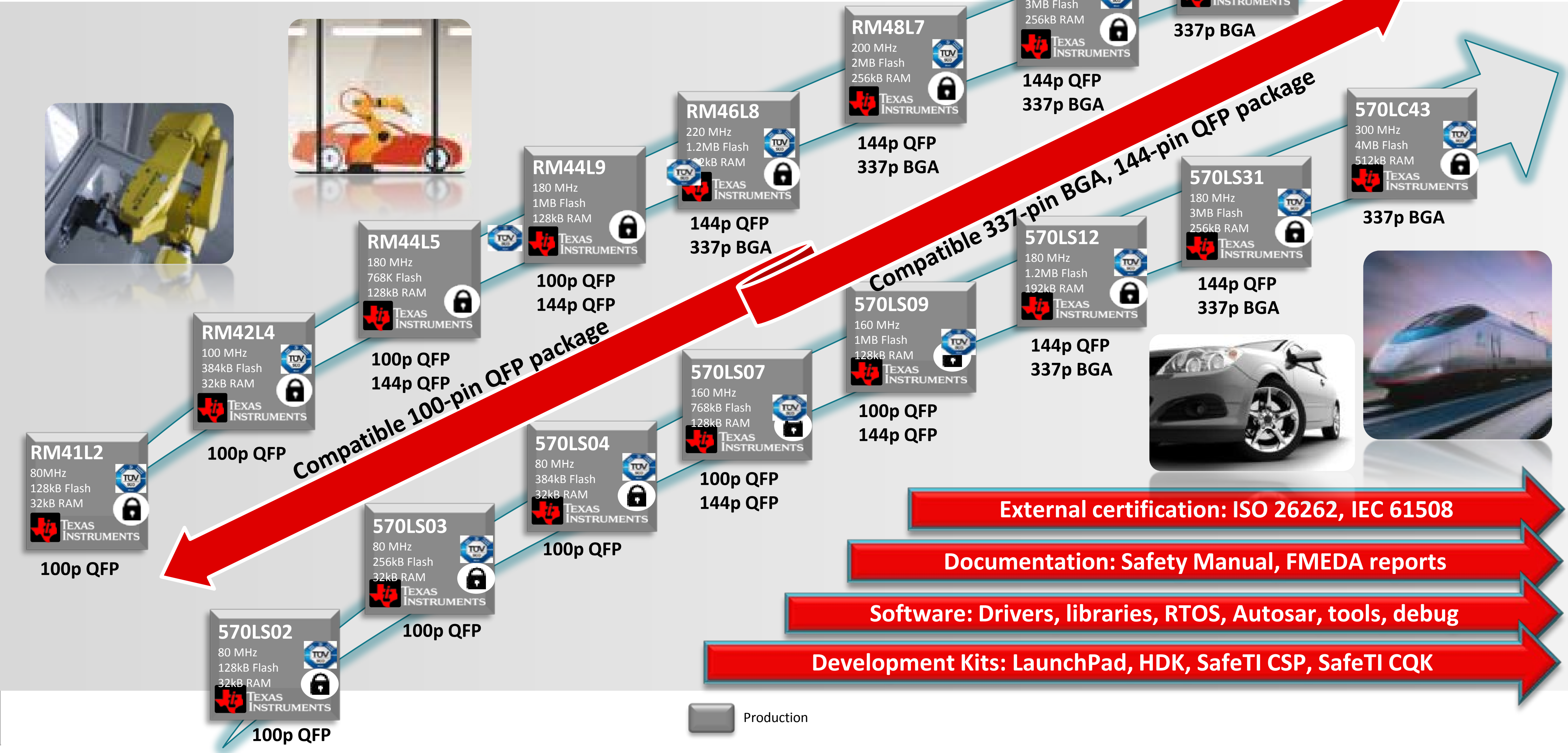
- Development flow certified to IEC61508 & ISO26262
- Integrated diagnostics protect against random faults

Time To Market

- Hardware safety features reduce software development
- Tools, software and safety documentation

Hercules™ MCUs

Scalable platform for functional safety applications



RM41L2
80MHz
128kB Flash
32kB RAM
100p QFP

RM42L4
100 MHz
384kB Flash
32kB RAM
100p QFP

RM44L5
180 MHz
768K Flash
128kB RAM
100p QFP
144p QFP

RM44L9
180 MHz
1MB Flash
128kB RAM
100p QFP
144p QFP

RM46L8
220 MHz
1.2MB Flash
192kB RAM
144p QFP
337p BGA

RM48L7
200 MHz
2MB Flash
256kB RAM
144p QFP
337p BGA

RM48L9
220 MHz
3MB Flash
256kB RAM
144p QFP
337p BGA

RM57L
330 MHz
4MB Flash
512kB RAM
337p BGA

570LS02
80 MHz
128kB Flash
32kB RAM
100p QFP

570LS03
80 MHz
256kB Flash
32kB RAM
100p QFP

570LS04
80 MHz
384kB Flash
32kB RAM
100p QFP

570LS07
160 MHz
768kB Flash
128kB RAM
100p QFP
144p QFP

570LS09
160 MHz
1MB Flash
128kB RAM
100p QFP
144p QFP

570LS12
180 MHz
1.2MB Flash
192kB RAM
144p QFP
337p BGA

570LS31
180 MHz
3MB Flash
256kB RAM
144p QFP
337p BGA

570LC43
300 MHz
4MB Flash
512kB RAM
337p BGA

Compatible 100-pin QFP package

Compatible 337-pin BGA, 144-pin QFP package

External certification: ISO 26262, IEC 61508

Documentation: Safety Manual, FMEDA reports

Software: Drivers, libraries, RTOS, Autosar, tools, debug

Development Kits: LaunchPad, HDK, SafeTI CSP, SafeTI CQK

▭ Production

Cortex-R: Ideal for safety-critical applications

Safety features

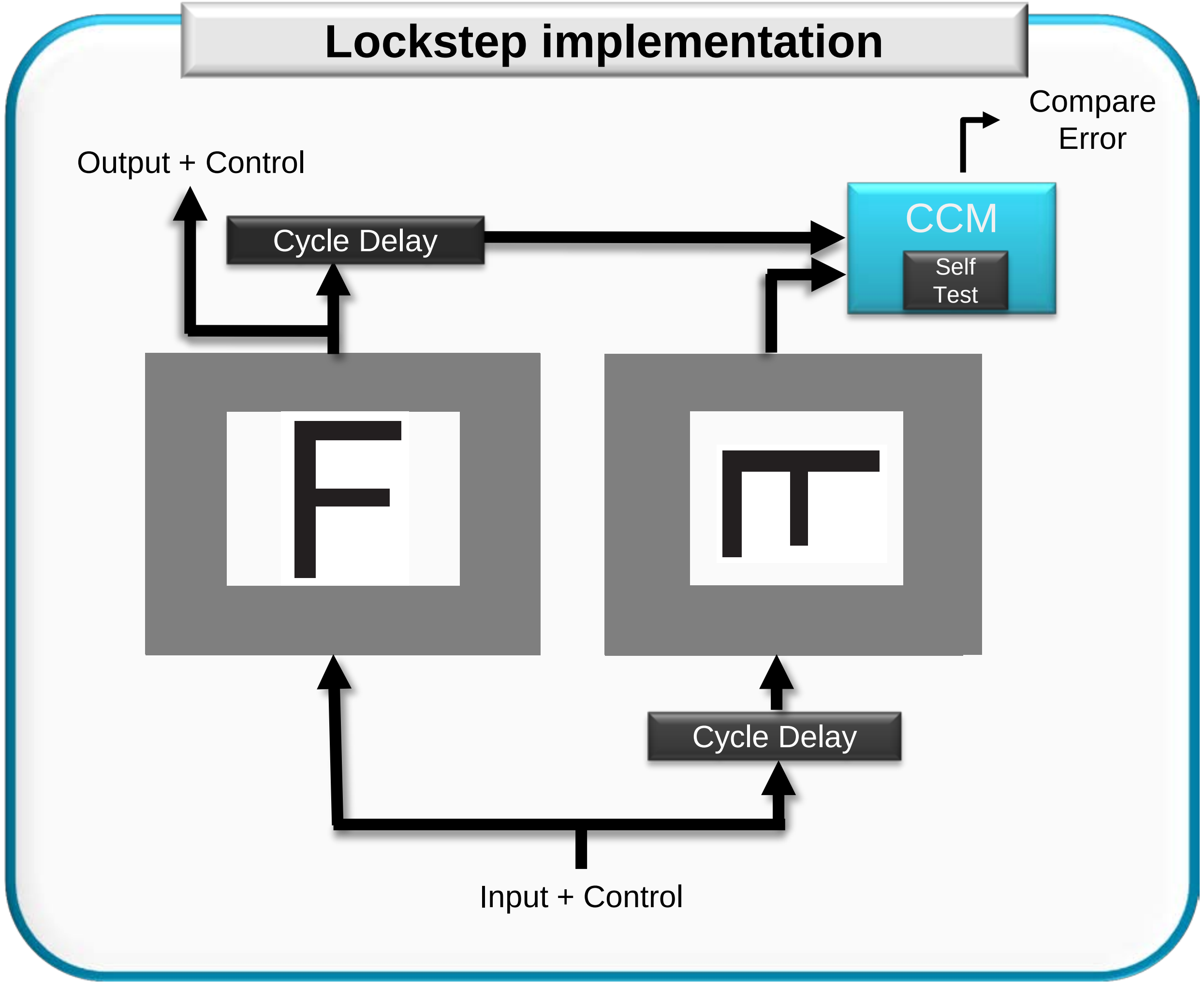
- Supports Lockstep
- Memory Protection Unit (MPU)
- Error-Correcting Code (ECC)

Higher performance

- 8-stage processor pipeline
- Dual issue – two instructions can execute in parallel
- Load store unit reduces stalling
- Pre-fetch and Branch Prediction Units
- Cached*

Real-time / determinism

- Tightly Coupled Memory (TCM)
- Fast interrupt response
- Deterministic interrupt response



Cortex-R4 features

- ARMv7-R architecture, supports ARM and Thumb2 instruction sets
- 8-stage processor pipeline
- Pre-fetch and Branch Prediction Units
- Floating-Point Unit
- Fast interrupt response
- Tightly Coupled Memory (TCM) with ECC
- Memory Protection Unit (MPU)
- Performance Monitoring Unit (PMU)

Cortex-R4 Hercules Processor

Prefetch unit

- Fetches instructions from the TCMs, or external memory
- Predicts the outcome of branches in the instruction stream

Data Processing Unit (DPU)

- Decodes and executes instructions
- Interfaces with LSU to transfer data to or from the memory system
- Holds general-purpose registers, status registers and control registers (CP15, CP14, etc.)

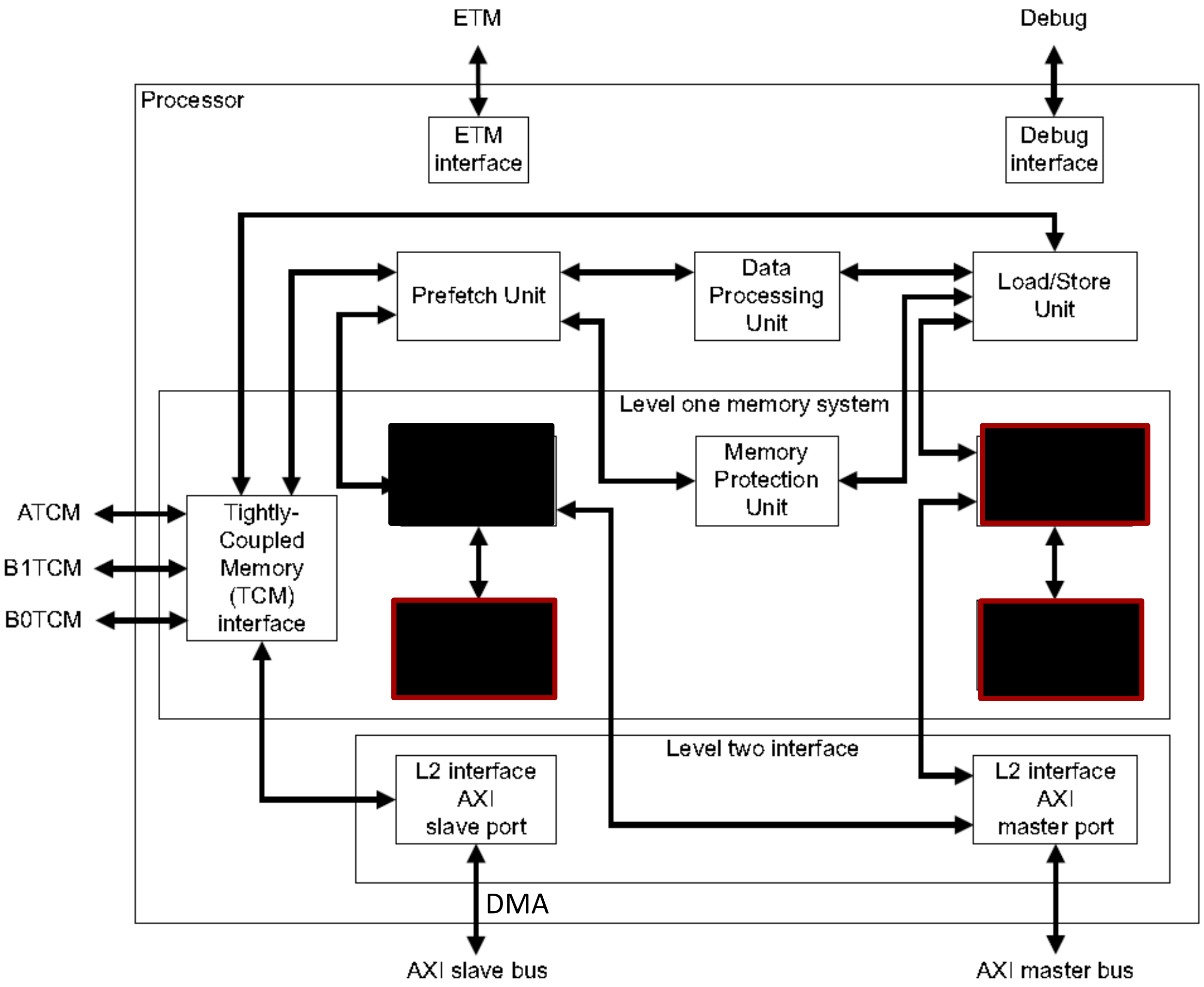
Load/store unit

- Manages all load & store operations, interfacing with the DPU, TCMs, and memory

AXI master interface

- Provides a high-bandwidth interface to on-chip RAM, peripherals, and interfaces to external memory
- Consists of a single AXI port with a 64-bit read/write channel for instruction & data fetches
- Can run at the same frequency as the processor

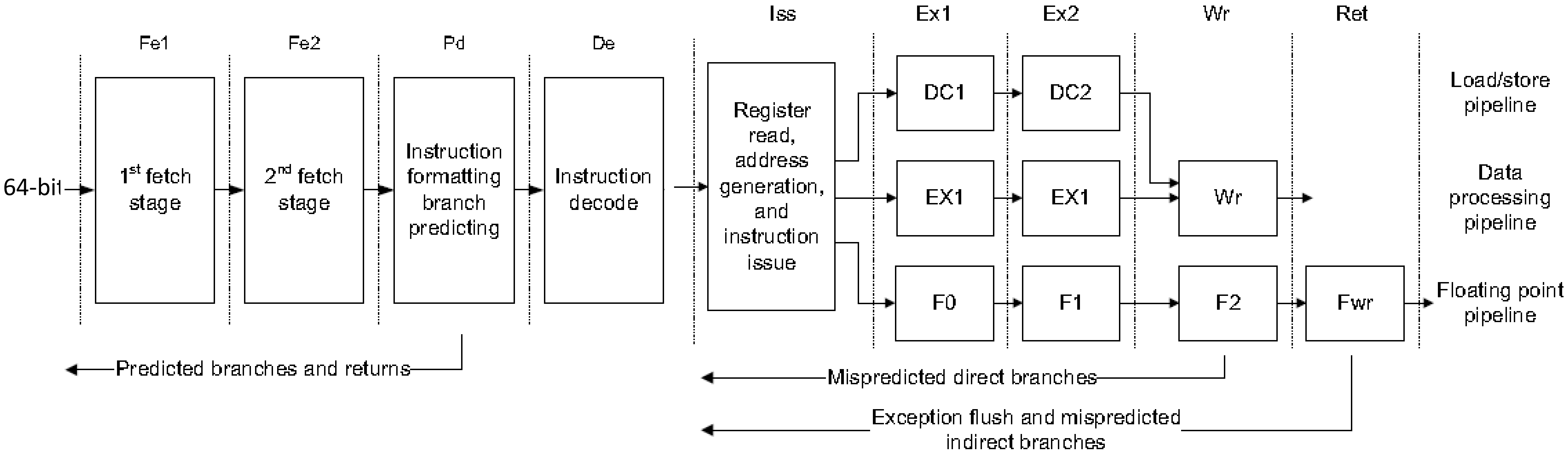
Cortex-R4 processor structure



Pipeline

The following stages make up the pipeline:

- Fetch stages (Fe1, Fe2)
- Pre-Decode and Decode stages (Pd, De)
- Issue stage (Iss)
- Execution stages (Ex1, Ex2, etc.)



Fe1
Fe2
Pd
De
Iss
Ex1
Ex2
Wr
Ret
64-bit
1st fetch stage
2nd fetch stage
Instruction formatting branch predicting
Instruction decode
Register read, address generation, and instruction issue
DC1
DC2
EX1
EX1
Wr
F0
F1
F2
Fwr
Load/store pipeline
Data processing pipeline
Floating point pipeline
Predicted branches and returns
Mispredicted direct branches
Exception flush and mispredicted indirect branches

Floating Point Unit (FPU)

- FPU is compliant to IEEE754
- 16 double-word (64 bits) registers
- 32 single-word (32 bits) registers
- Supports features:
  - Single-precision and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations
  - Conversions between fixed-point and floating-point data formats, etc
  - Comparisons
  - Underflow
  - Exceptions

Processor modes

Exception modes

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

- ARM has 7 basic operating modes.
- Modes other than user mode have privileged access rights.
- Usually the initial setup is done in SVC mode after reset, then switch to system or user mode afterwards.
- Privileged access rights are needed to access certain configuration registers in the processor and peripherals. For example, CP15, flash control registers.
- Cortex-R MCU has MPU, which can be used to set the memory access rights for certain regions.

Processor registers

	User/System mode	FIQ mode	IRQ mode	Supervisor mode	Abort Exception	Undefined Instruction
	R0	R0	R0	R0	R0	R0
	R1	R1	R1	R1	R1	R1
	R2	R2	R2	R2	R2	R2
	R3	R3	R3	R3	R3	R3
	R4	R4	R4	R4	R4	R4
	R5	R5	R5	R5	R5	R5
	R6	R6	R6	R6	R6	R6
	R7	R7	R7	R7	R7	R7
	R8	R8 FIQ	R8	R8	R8	R8
	R9	R9 FIQ	R9	R9	R9	R9
	R10	R10 FIQ	R10	R10	R10	R10
	R11	R11 FIQ	R11	R11	R11	R11
	R12	R12 FIQ	R12	R12	R12	R12
Stack Pointer (SP)	R13 SP	R13 FIQ	R13 IRQ	R13 SVC	R13 ABORT	R13 UNDEF
Link Register (LR)	R14 LR	R14 FIQ	R14 IRQ	R14 SVC	R14 ABORT	R14 UNDEF
Program Counter (PC)	R15 PC	R15 PC	R15 PC	R15 PC	R15 PC	R15 PC
Current Program Status Register (CPSR)	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
Saved Program Status Register (SPSR)		SPSR FIQ	SPSR IRQ	SPSR SVC	SPSR ABORT	SPSR UNDEF

Current Processor Status Register (CPSR)

31	30	29	28	27	26 25	24	23 20	19 16	15 10	9	8	7	6	5	4 0
N	Z	C	V	Q	IT	J	DNM	GE[3:0]	IT[7:2]	E	A	I	F	T	MODE[4:0]

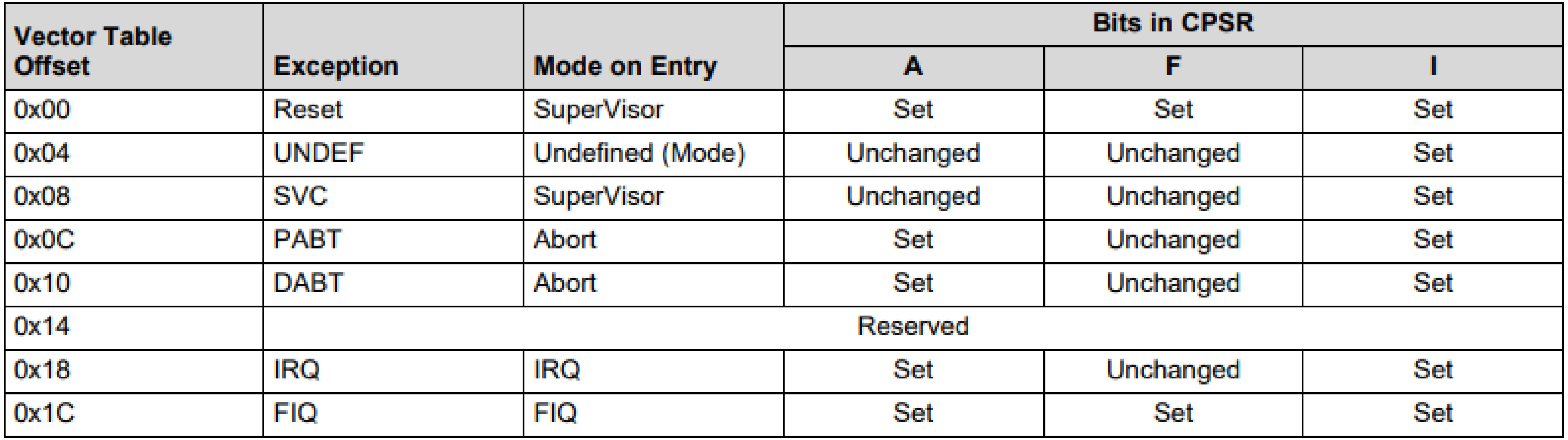
- **Condition Code Flags**
  - N = ALU Negative result
  - Z = ALU Zero result
  - C = ALU Carry out
  - V = ALU arithmetic overflow
  - Q = ALU sticky overflow
- J = Java state bit (always reads 0)
- IT[7:0] = If-Then
- DNM (Do Not Modify)
- GE[3:0] = Greater Than or Equal To
- E = Endianism of Data
- A = Imprecise Abort Disable
- **Interrupt Disable bits**
  - I = 1, disables the IRQ
  - F = 1, disables the FIQ
- State bit
  - T = 0, 32-bit instruction set
  - T = 1, 16-bit instruction set
- **Mode (defines processor mode)**

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

Supported data types

- The processor supports the following data types:
  - Double Word (64 bit)
  - Word (32 bit)
  - Half Word (16 bit)
  - Byte (8 bit)
- Although the processor supports unaligned accesses, TI does not recommend using unaligned accesses for bus performance.
  - Above data types should be aligned at their respective size boundary.
  - Most unaligned accesses are converted into multiple aligned accesses.
- The TMS570 devices are storing their data in big endian format (BE32), and RM4/5x stores data in little endian format.

Exception handling and the vector table



Vector Table Offset	Exception	Mode on Entry	Bits in CPSR		
			A	F	I
0x00	Reset	SuperVisor	Set	Set	Set
0x04	UNDEF	Undefined (Mode)	Unchanged	Unchanged	Set
0x08	SVC	SuperVisor	Unchanged	Unchanged	Set
0x0C	PABT	Abort	Set	Unchanged	Set
0x10	DABT	Abort	Set	Unchanged	Set
0x14	Reserved				
0x18	IRQ	IRQ	Set	Unchanged	Set
0x1C	FIQ	FIQ	Set	Set	Set

- Reset Highest Priority
- DABT
- FIQ
- IRQ
- UNDEF
- PABT
- SVC Lowest Priority

- The FIQ is implemented as a non-maskable interrupt in the Hercules MCU.
- nFIQ and nIRQ inputs are connected to VIM.

Exception handling

When an exception occurs, the CPU does the following:

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
  - Change to ARM state
  - Change to exception mode
  - When an IRQ interrupt is received, the CPU disables other IRQ interrupts
  - When an FIQ interrupt is received, the CPU disables both IRQ and FIQ interrupts
- Stores the return address in LR_<mode>
- Sets PC to vector address or ISR address
- Swaps banked registers

To return, the exception handler needs to:

- Restore CPSR from SPSR_<mode>
- Restore PC from LR_<mode>

Interrupt handling

The Hercules MCU supports three different modes to handle peripheral interrupts in hardware and software:

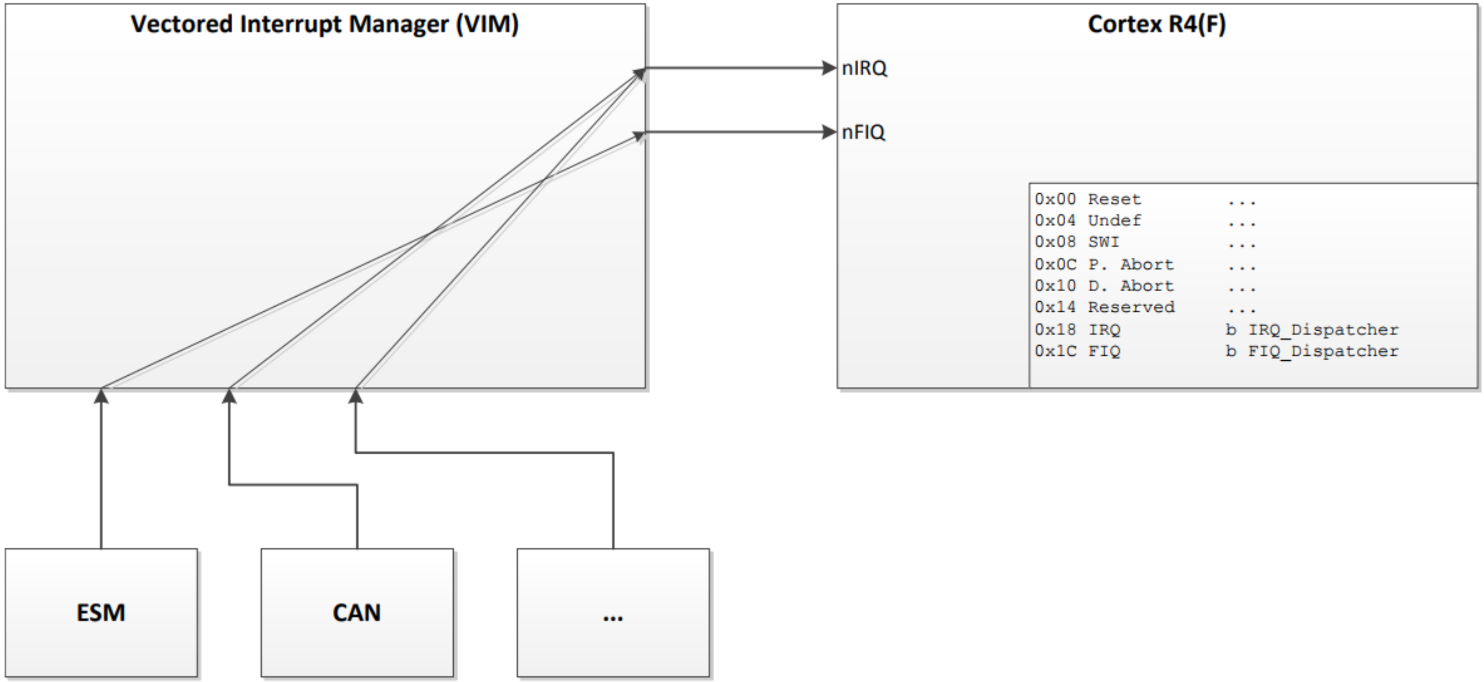
- **Index interrupts mode (legacy mode)**: The interrupt dispatching has to be done completely in software (software dispatcher).

- **Register-vectored interrupt mode**: This mode allows the interrupt dispatching to be done in hardware, the software has only to load the interrupt vector of the ISR from the VIM module and branch to the vector.

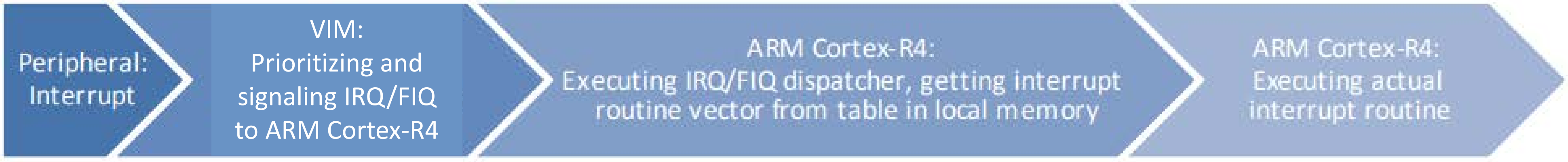
- **Hardware-vectored interrupt mode** (IRQ only): This mode has the advantage that the vector of the ISR has not been loaded by software. Instead, the vector is directly supplied to the MCU core via the VIC port, and saves some CPU cycles for lower interrupt latency compared to the second mode.

Index interrupts mode

1. Events occur within peripherals
2. Peripherals make FIQ/IRQ requests to the VIM
3. VIM prioritizes the requests & provides the highest ISR to CPU
4. CPU fetch from 0x18/0x1C.
5. Branch to ISR dispatcher.
6. Load Interrupt offset .(IRQINDEX, FIQINDEX)
7. Decide which ISR to execute.
8. Branch to ISR



```
0x00 Reset           ...
0x04 Undef           ...
0x08 SWI             ...
0x0C P. Abort        ...
0x10 D. Abort        ...
0x14 Reserved        ...
0x18 IRQ             b IRQ_Dispatcher
0x1C FIQ             b FIQ_Dispatcher
```



Peripheral: Interrupt

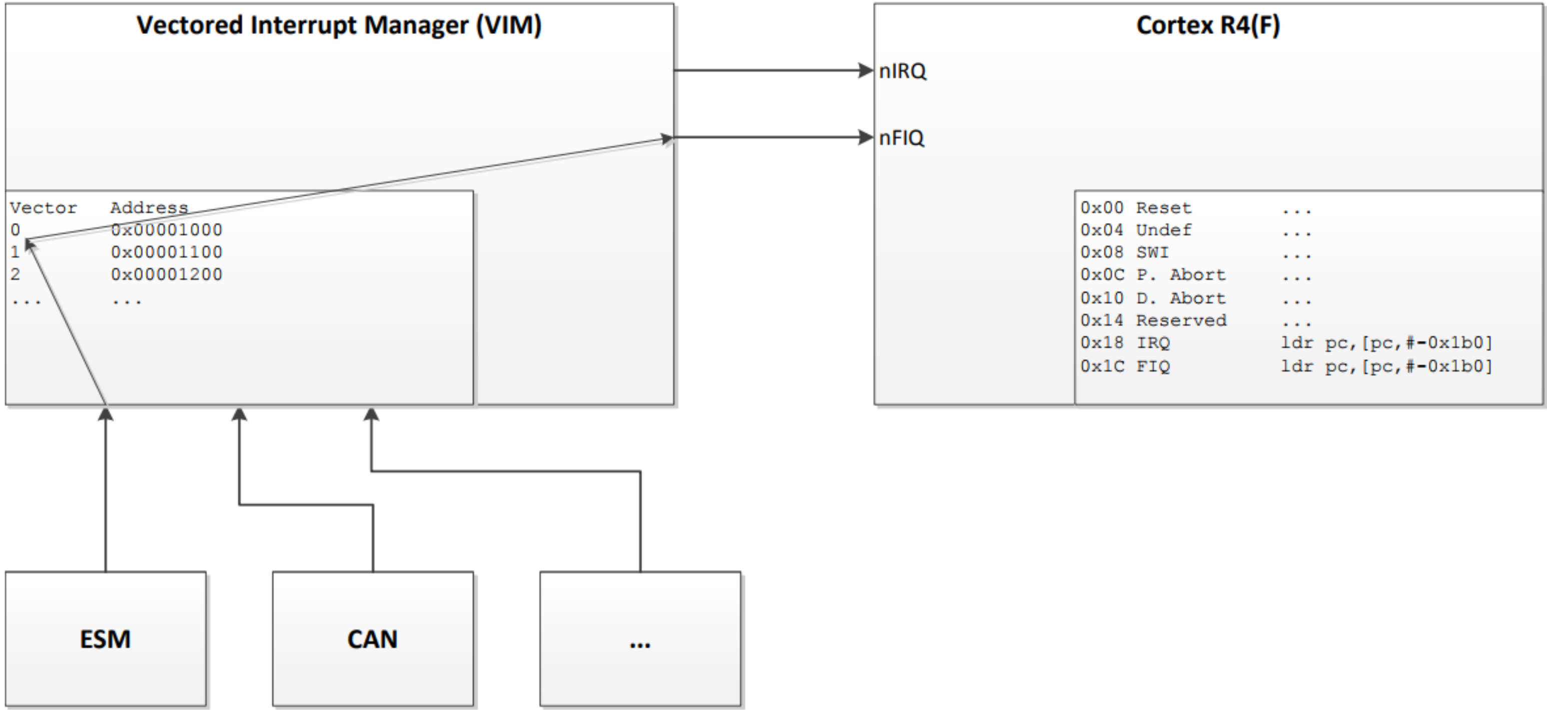
VIM: Prioritizing and signaling IRQ/FIQ to ARM Cortex-R4

ARM Cortex-R4: Executing IRQ/FIQ dispatcher, getting interrupt routine vector from table in local memory

ARM Cortex-R4: Executing actual interrupt routine

Register-vectored interrupts

1. Events occur within peripherals
2. Peripherals make FIQ/IRQ requests to the VIM
3. VIM prioritizes the requests & provides the addr of the highest ISR to CPU
4. CPU fetches from 0x18/0x1C
5. Branch to ISR (**LDR PC, [PC, #-0x1B0]**), (IRQVECREG/FIQVECREG.)
6. Branch to ISR



Vectored Interrupt Manager (VIM)

```
Vector   Address
0        0x00001000
1        0x00001100
2        0x00001200
...      ...
```

Cortex R4(F)

nIRQ

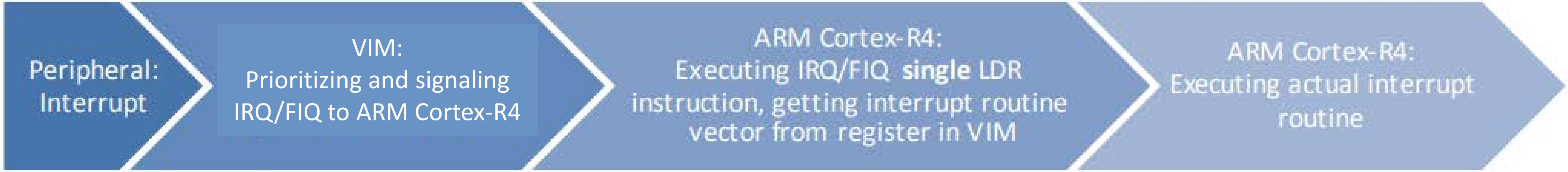
nFIQ

```
0x00 Reset        ...
0x04 Undef        ...
0x08 SWI          ...
0x0C P. Abort     ...
0x10 D. Abort     ...
0x14 Reserved     ...
0x18 IRQ          ldr pc,[pc,#-0x1b0]
0x1C FIQ          ldr pc,[pc,#-0x1b0]
```

ESM

CAN

...



Peripheral: Interrupt

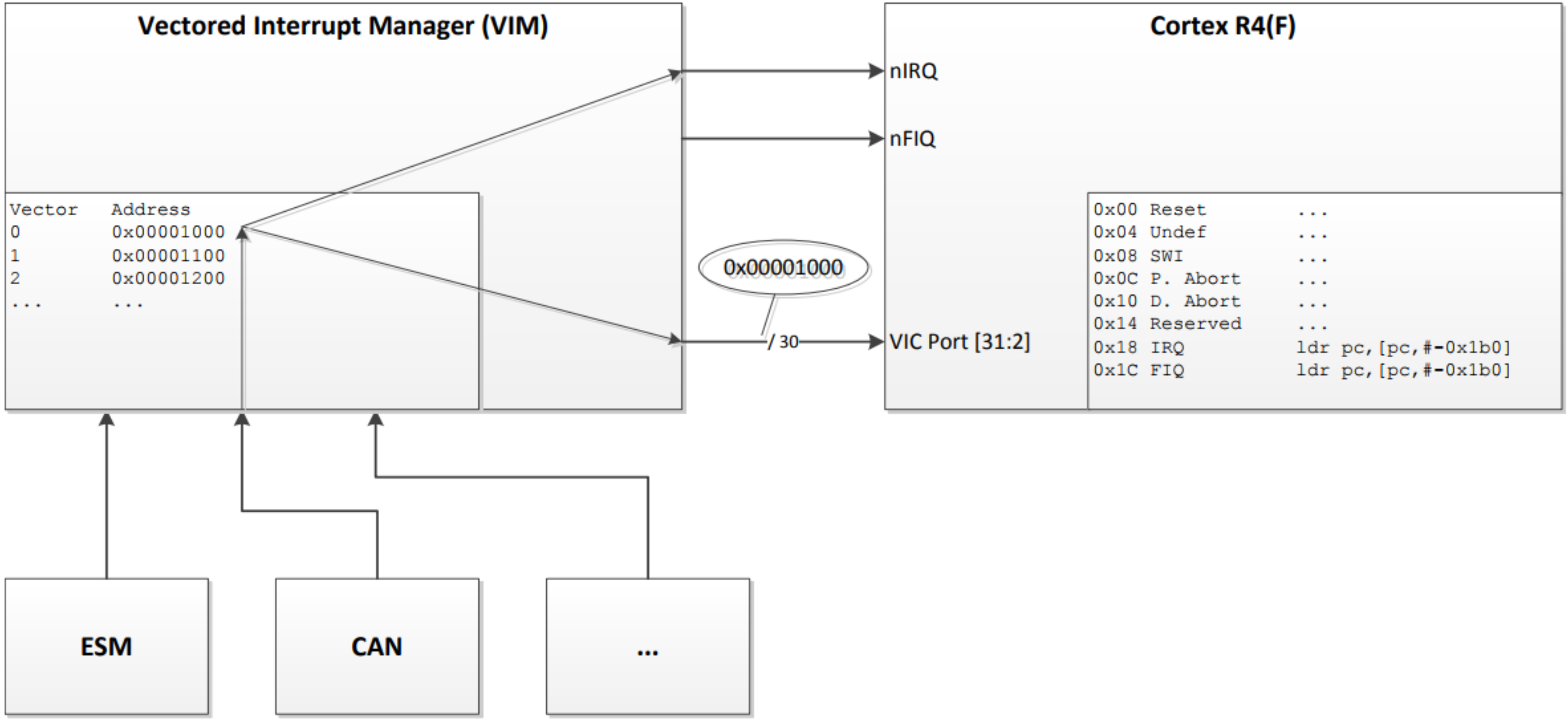
VIM: Prioritizing and signaling IRQ/FIQ to ARM Cortex-R4

ARM Cortex-R4: Executing IRQ/FIQ **single** LDR instruction, getting interrupt routine vector from register in VIM

ARM Cortex-R4: Executing actual interrupt routine

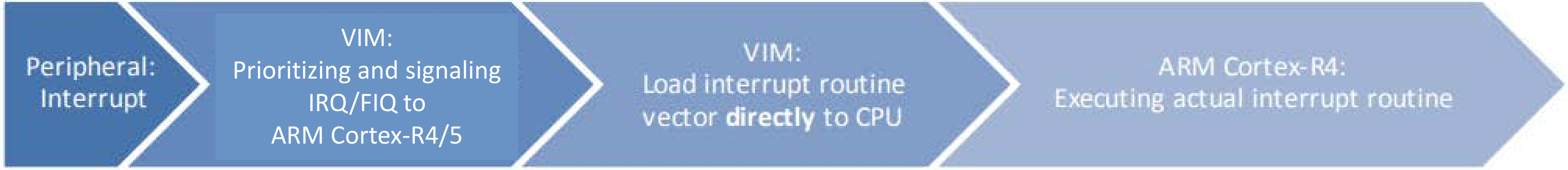
Hardware-vectored interrupts (only IRQ)

1. Events occur within peripherals
2. Peripherals make FIQ/IRQ requests to the VIM
3. VIM prioritizes the requests
4. VIM provides address of highest pending request directly to the processors VIC port.
5. CPU branches directly to ISR.



Vector	Address
0	0x00001000
1	0x00001100
2	0x00001200
...	...

```
0x00 Reset        ...
0x04 Undef        ...
0x08 SWI          ...
0x0C P. Abort     ...
0x10 D. Abort     ...
0x14 Reserved     ...
0x18 IRQ          ldr pc,[pc,#-0x1b0]
0x1C FIQ          ldr pc,[pc,#-0x1b0]
```


Peripheral: Interrupt

VIM: Prioritizing and signaling IRQ/FIQ to ARM Cortex-R4/5

VIM: Load interrupt routine vector **directly** to CPU

ARM Cortex-R4: Executing actual interrupt routine

Abort: Prefetch and Data

Prefetch Abort (PABT)

- CPU tries to execute an instruction from a protected or faulty memory location, such as:
  - The memory location is not implemented in the system.
  - The memory region is protected by the MPU.
  - An error is detected in the data by the ECC checking logic.
- All prefetch aborts are precise.

Data Abort (DABT)

- The CPU takes the data abort if data is read from or written to a protected or faulty memory location. This could be because of the following conditions:
  - The memory location is not implemented.
  - The memory location is read- or write-only in privileged mode (when processor is in User mode).
  - The memory location is read- or write-protected by the MPU.
  - If an error is detected in the data by the ECC checking logic.
- Data aborts can be precise or imprecise.

Abort type: Precise (synchronous), Imprecise (asynchronous)

Precise or Synchronous Aborts

- The abort is taken at the instruction that caused the exception.
- The abort handler could use the SPSR_abt and R14_abt (LR_abt) registers to determine the instruction that generated the abort and the CPU state when the abort occurred.
- Prefetch abort is always a precise abort.

Imprecise or Asynchronous Aborts

- If the exception is taken on an instruction later than the instruction that caused the exception.
- It is not possible to determine the exact instruction that caused the abort.
- This could be the case on writes to normal-type memory, where the write is stored in a buffer until the memory system is ready to perform it. In such cases, the exception will be generated and the abort will be taken after the appropriate store instruction was executed by the processor.

How to determine the cause of an abort

- The Cortex-R4/5 processor has a *system control coprocessor* implemented: The CP15. The CP15 offers the possibility to readout additional information about an abort.
- Four registers in the CP15 hold information about the cause of an abort:
  - Data Fault Status Register
  - Auxiliary Fault Status Registers
  - Data Fault Address Register
  - Instruction Fault Address Register

For more information

- **SafeTI Web Page: www.ti.com/safeti**
- **Hercules Web Page: www.ti.com/hercules**
  - Data sheets
  - Technical Reference Manual
  - Application notes
  - Software & tools downloads and updates
  - Order evaluation and development kits
- **Hercules Safety Microcontrollers Training Series training.ti.com/hercules**
  - Cortex-R Processor Architecture
  - Peripherals
  - Software
  - Functional Safety
- **For questions about this training, refer to the Engineer-2-Engineer Support Forum www.ti.com/hercules-support**
  - News and announcements
  - Ask technical questions
  - Search for technical content

TI E2E™ Community

Texas Instruments

This material is provided strictly "as-is," for informational purposes only, and without any warranty.
Use of this material is subject to TI's **Terms of Use**, viewable at TI.com