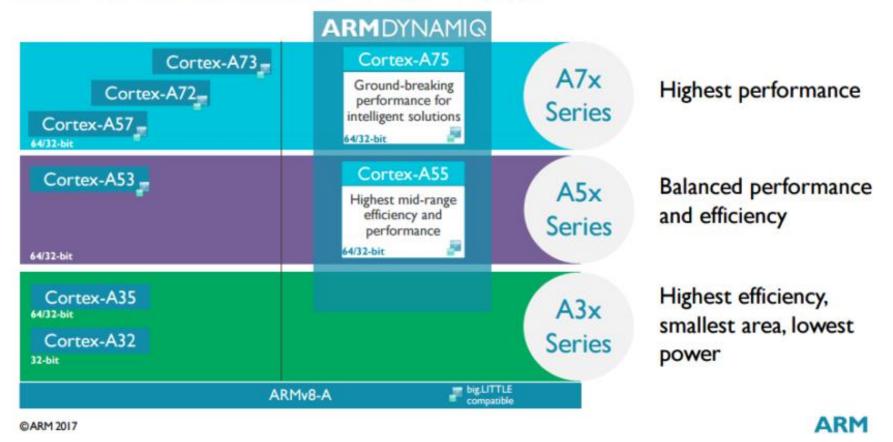
Toronto Metropolitan University

COE718: Embedded Systems Design

Latest Cortex A series CPUs Cortex-M3 Micro-architecture Features

ARMv8 64-bit Architecture

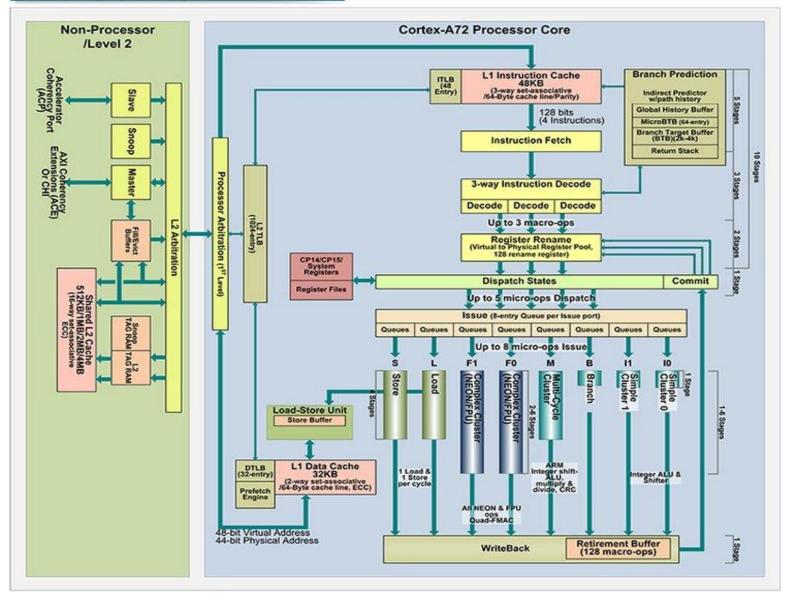
ARM Cortex-A ARMv8 portfolio



A72 has 3.5 times performance gain over A15. Pair with A53 to get big.Little Architecture. A73 introduced in 2016.

ARMv8 Cortex A72/A73 (2010-2015/2016)

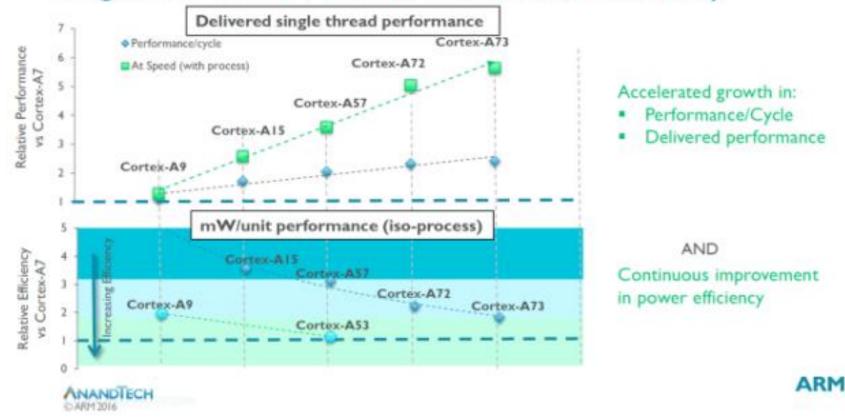
ARM Cortex-A72 Block Diagram



Copyright (c) 2015 Hiroshige Goto All rights reserved.

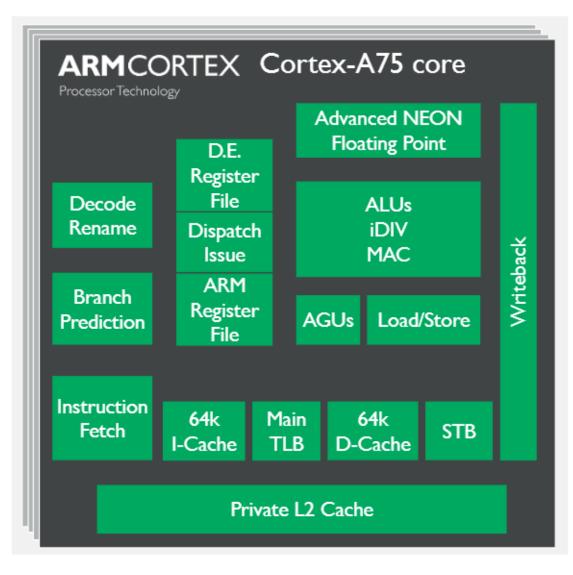
ARM Cortex A73 Performance

High End: More Performance, More Efficiency



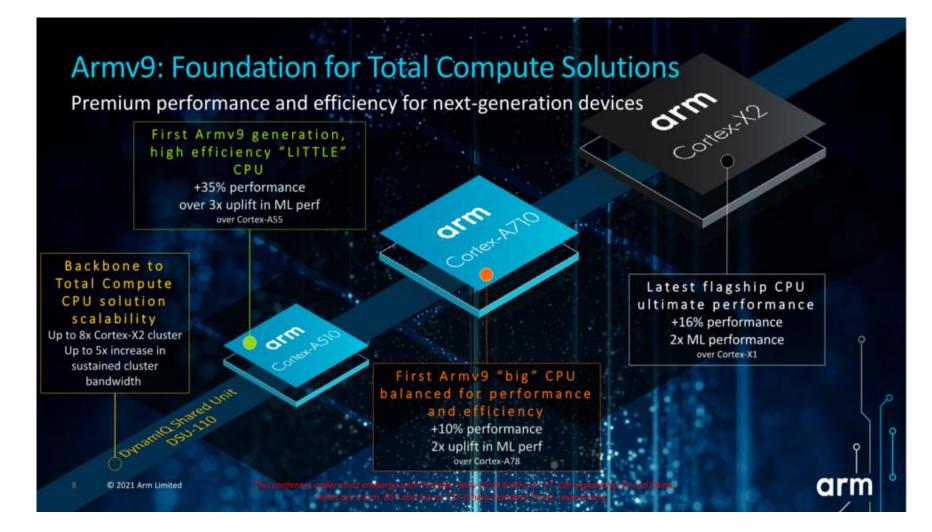
ARM Cortex-A73 claimed to be up to 10 percent faster than the A72.A73 uses 20% less power than A72 for same process/frequency.3GHz on a 10nm SoC & 2.8GHz on a 16nm SoC is achievable with A73.

Cortex A75



Cortex-A75 execute up to 3 instructions per clock cycle. A75 boasts 7 execution units, two load/stores, two NEON & FPU, a BPU and two integer cores.

Latest – Cortex A510, A710 and X2 Armv9 Generation



ARMv7 ISA – Data Processing

Instruction	Function
ADC	Add with carry
ADD	Add
ADR	Add PC and an immediate value and put the result in a register
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear (Logical AND one value with the logic inversion of another value)
CMN	Compare negative (compare one data with two's complement of another data and update flags)
CMP	Compare (compare two data and update flags)
CPY	Copy (available from architecture v6; move a value from one high or low register to another high or low register); synonym of MOV instruction
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MOV	Move (can be used for register-to-register transfers or loading immediate data)
MUL	Multiply
MVN	Move NOT (obtain logical inverted value)
NEG	Negate (obtain two's complement value), equivalent to RSB

ARMv7 ISA – Data Processing

Instruction	Function
ORR	Logical OR
RSB	Reverse subtract
ROR	Rotate right
SBC	Subtract with carry
SUB	Subtract
TST	Test (use as logical AND; Z flag is updated but AND result is not stored)
REV	Reverse the byte order in a 32-bit register (available from architecture v6)
REV16	Reverse the byte order in each 16-bit half word of a 32-bit register (available from architecture v6)
REVSH	Reverse the byte order in the lower 16-bit half word of a 32-bit register and sign extends the result to 32 bits (available from architecture v6)
SXTB	Signed extend byte (available from architecture v6)
SXTH	Signed extend half word (available from architecture v6)
UXTB	Unsigned extend byte (available from architecture v6)
UXTH	Unsigned extend half word (available from architecture v6)

ARMv7 ISA

ADC	ADD	ADR AND	ASR		CLZ
BFC	BFI	BIC CDP	CLREX	CBNZ CBZ	CMN
CMP			DBG	EOR	LDC
LDMIA	BKPT BLX	ADC ADD ADR	LDMDB	LDR	LDRB
LDRBT	BX CPS	(AND) (ASR) (B)	LDRD	LDREX	LDREXB
LDREXH	DMB	BL BIC	LDRH	LDRHT	LDRSB
LDRSBT	DSB	CMN CMP EOR	LDRSHT	LDRSH	LDRT
MCR	ISB	LDR LDRB LDM	LSL	LSR	MLS
MCRR	MRS	LDRH (LDRSB) (LDRSH)	MLA	MOV	MOVT
MRC	MSR	LSL LSR MOV	MRRC	MUL	MVN
NOP	NOP REV	MUL MVN ORR	ORN	ORR	PLD
PLDW	REV16 REVSH	(POP) (PUSH) (ROR)	PU	POP	PUSH
RBIT	SEV SXTB	(RSB) (SBC) (STM)	REV	REV16	REVSH
ROR	SXTH UXTB	STR STRB STRH	RRX	RSB	SBC
SBFX	UXTH WFE	(SUB) (SVC) (TST)	SDIV	SEV	SMLAL
SMULL	WFI YIELD	CORTEX-M0	SSAT	STC	STMIA
STMDB				STRB	STRBT
STRD	STREX	STREXB STREXH	STRH	STRHT	STRT
SUB	SXTB	SXTH TBB	ТВН	TEQ	TST
UBFX		UMLAL UMULL	USAT	UXTB	UXTH
WFE	WFI C	YIELD IT		C	ORTEX-M3

ARMv7 Suffixes

- Instructions do not update the PSR unless a suffix 'S' is appended to the instruction
 - i.e. ADD vs ADDS
- Exceptions: Compare (CMP) and Test (TST, TEQ etc)
- Write to the PSR directly
- 16b Thumb Instructions

ARMv7 Suffixes

 Table 4.16
 Examples of Preindexing Memory Access Instructions

Example		Description
LDRB.W Rd, LDRH.W Rd,	[Rn, #offset]! [Rn, #offset]! [Rn, #offset]! Rd2,[Rn, #offset]!	Preindexing load instructions for various sizes (word, byte, half word, and double word)
	[Rn, #offset]! [Rn, #offset]!	Preindexing load instructions for various sizes with sign extend (byte, half word)
STRB.W Rd, STRH.W Rd,	[Rn, #offset]! [Rn, #offset]! [Rn, #offset]! Rd2,[Rn, #offset]!	Preindexing store instructions for various sizes (word, byte, half word, and double word)

- For Memory accesses, also has suffixes appended to instruction to indicate size of the word to be loaded or stored
- i.e. LDR(size).W

- Can execute individual instructions conditionally based on the condition flags set by previous instruction(s).
- Cond Execution can be invoked by:
 - Using conditional branches
 - Adding condition code suffixes to instructions

Conditional Branches

Table 4.1 Suffixes in Instructions		
Suffix	Description	
S	Update Application Program Status register (APSR) (flags); for example: ADD <u>S</u> R0, R1 ; this will update APSR	
EQ, NE, LT, GT, and so on	Conditional execution; EQ = Equal, NE = Not Equal, LT = Less Than, GT = Greater Than, and so forth. For example: B <u>EQ</u> $\langle Label \rangle$; Branch if equal	

BNE, BLT, BGT etc

в	Branch
B <cond></cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR (this is actually a 32-bit instruction, but it is also available in Thumb in traditional ARM processors)
BLX	Branch with link and change state (BLX <reg> only)¹</reg>
BX <reg></reg>	Branch with exchange state
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

- IF-Then-Else structures (IT Blocks)
- Handle small conditional code
- Used to avoid branch penalties
- Maximum 4 conditionally executed instructions
- I = IF, T = Then, E = Else

Example of ITTEE block

I	if (R1 <r2) th="" then<=""></r2)>
Т	R2=R2-R1
Т	R2=R2/2
	else
Е	R1=R1-R2
Е	R1=R1/2

Example of ITTEE block

	СМР	R1, R2	; If R1 < R2 (less then)
if (R1 <r2) then<br="">R2=R2-R1</r2)>	ITTEE	LT	; then execute instruction 1 and 2 ; (indicated by T)
R2=R2/2			; else execute instruction 3 and 4 ; (indicated by E)
else	SUBLT.W	R2,R1	; 1 st instruction
R1=R1-R2	LSRLT.W	R2, # 1	; 2 nd instruction
R1=R1/2	SUBGE.W	R1,R2	; 3 rd instruction (notice the GE is
	LSRGE.W	R1, # 1	; opposite of LT) ; 4 th instruction

Symbol	Condition	Flag
EQ	Equal	Z set
NE	Not equal	Z clear
CS/HS	Carry set/unsigned higher or same	C set
CC/LO	Carry clear/unsigned lower	C clear
MI	Minus/negative	N set
PL	Plus/positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set
GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
LT	Signed less than	N set and V clear, or N clear and V set (N != V)
GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0, N == V)
LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
AL	Always (unconditional)	_

Instruction Suffixes

```
CMP RO, R1 ; Compare RO and R1

ITTEE GT ; If RO > R1 Then

; if true, first 2 statements execute,

; if false, other 2 statements execute

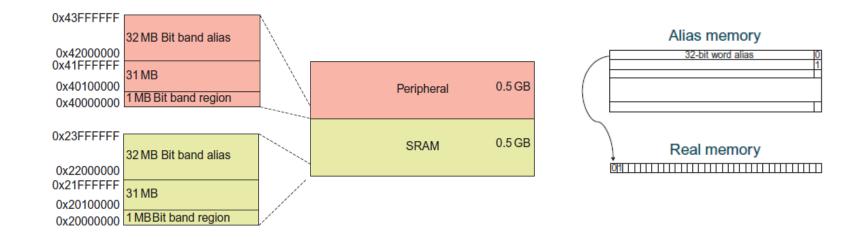
MOVGT R2, RO ; R2 = RO

MOVGT R3, R1 ; R3 = R1

MOVLE R2, RO ; Else R2 = R1

MOVLE R3, R1 ; R3 = RO
```

Bit-Banding



- Address 0x2000000 = SRAM
- 0x4000000 = Peripheral = external RAM, devices, vendor specific memory etc

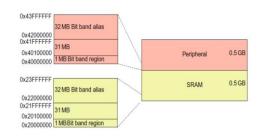
Bit-Banding - LPC 17xx

0x4010 0000	APB1 peripherals	4 GB	LPC1768 memory space	
0x400F C000	31 system control			0xFFFF FFFF
0x400C 0000	30 - 16 reserved		reserved	,
0x400B C000	15 QEI			0xE010 0000
0x400B 8000	14 motor control PWM		private peripheral bus	0xE000 0000
0x400B 4000	13 reserved		reserved	÷
0x400B 0000	12 repetitive interrupt timer			0x5020 0000
0x400A C000	11 reserved		AHB periherals	←' 0x5000 0000
0x400A 8000	10 I2S		reserved	0x3000 0000
0x400A 4000	9 reserved	ll í	reserved	0x4400 0000
0x400A 0000	8 I2C2	}pe	ripheral bit band alias addressing	0x4200 0000
0x4009 C000	7 UART3		reserved	
0x4009 8000	6 UART2		APB1 peripherals	0x4010 0000
0x4009 4000	5 Timer 3		APB0 peripherals	0x4008 0000
0x4009 0000	4 Timer 2	1 GB		0x4000 0000
0x4008 C000	3 DAC		reserved	0x2400 0000
0x4008 8000	2 SSP0	AHI	B SRAM bit band alias addressing	
0x4008 0000	1 - 0 reserved		reserved	0x2200 0000
		É É		0x200A 0000
			GPIO	0x2009 C000
			reserved	
		0.5 GB	AHB SRAM (2 blocks of 16 kB)	0x2008 4000
		0.5 GB		0x2007 C000
		Ĩ.	reserved	0x1FFF 2000
			8 kB boot ROM	0x1FFF 0000
			reserved	0x1000 8000
			32 kB local static RAM	
		I-code/D-code		0x1000 0000
		memory space		
		2	reserved	í
0x0000 04		+ 256 words		0x0008 0000
0x0000 00	000 active interrupt vectors		512 kB on-chip flash	
		0 GB (0x0000 0000

AHB peripherals			0x5020 0000
		127- 4 reserved	-
	3	USB controller	0x5000 C000
{	2	reserved	0x5000 8000
	1	GPDMA controller	0x5000 4000
	0	Ethernet controller	0x5000 0000

		APB0 peripherals	0x4008 0000
ſ		31 - 24 reserved	0x4006 0000
	23	I2C1	0x4005 C000
		22 - 19 reserved	0x4004 C000
	18	CAN2	0x4004 8000
	17	CAN1	0x4004 4000
	16	CAN common	0x4004 0000
	15	CAN AF registers	0x4003 C000
	14	CAN AF RAM	0x4003 8000
	13	ADC	0x4003 4000
	12	SSP1	0x4003 0000
- 1	11	pin connect	0x4002 C000
	10	GPIO interrupts	0x4002 8000
	9	RTC + backup registers	0x4002 4000
	8	SPI	0x4002 0000
	7	I2C0	0x4001 C000
	6	PWM1	0x4001 8000
	5	reserved	0x4001 4000
	4	UART1	0x4001 0000
	3	UART0	0x4000 C000
	2	TIMER1	0x4000 8000
	1	TIMER0	0x4000 4000
l	0	WDT	0x4000 0000

Bit-Banding



Bit Band Word Address =

Bit Band Alias Base Address + (Byte Offset * 32) + (Bit Number * 4) (1)

<u>Byte Offset</u> = Bit's Bit Band Base Address - Bit Band Base Address (2)

where:

Byte Offset

Bit's Bit Band Base Address - the base address for the targeted SRAM or peripheral register (i.e. the effective address of the port) (= real address)

Bit Band Base Address - for SRAM = 0x20000000, for Peripherals = 0x40000000

Bit Band Alias Base Address - for SRAM = 0x22000000, for Peripherals = 0x42000000

Bit Number - the bit position of the targeted register (i.e., pin of the port)

Benefits of Bit-Banding

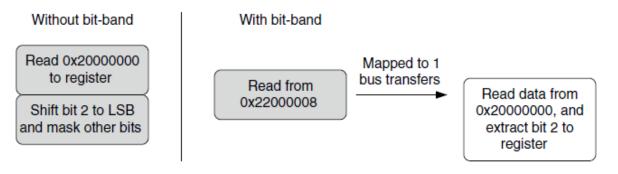


FIGURE 5.6

Read from the Bit-Band Alias.

Without bit-band	With bit-band
LDR R0,=0x20000000 ; Setup address LDR R1, [R0] ; Read UBFX.W R1, R1, #2, #1 ; Extract bit[2]	LDR RO,=0x22000008 ; Setup address LDR R1, [R0] ; Read