COE718: Embedded Systems Design

Lecture 2:
Cortex-M3 Microarchitecture Features

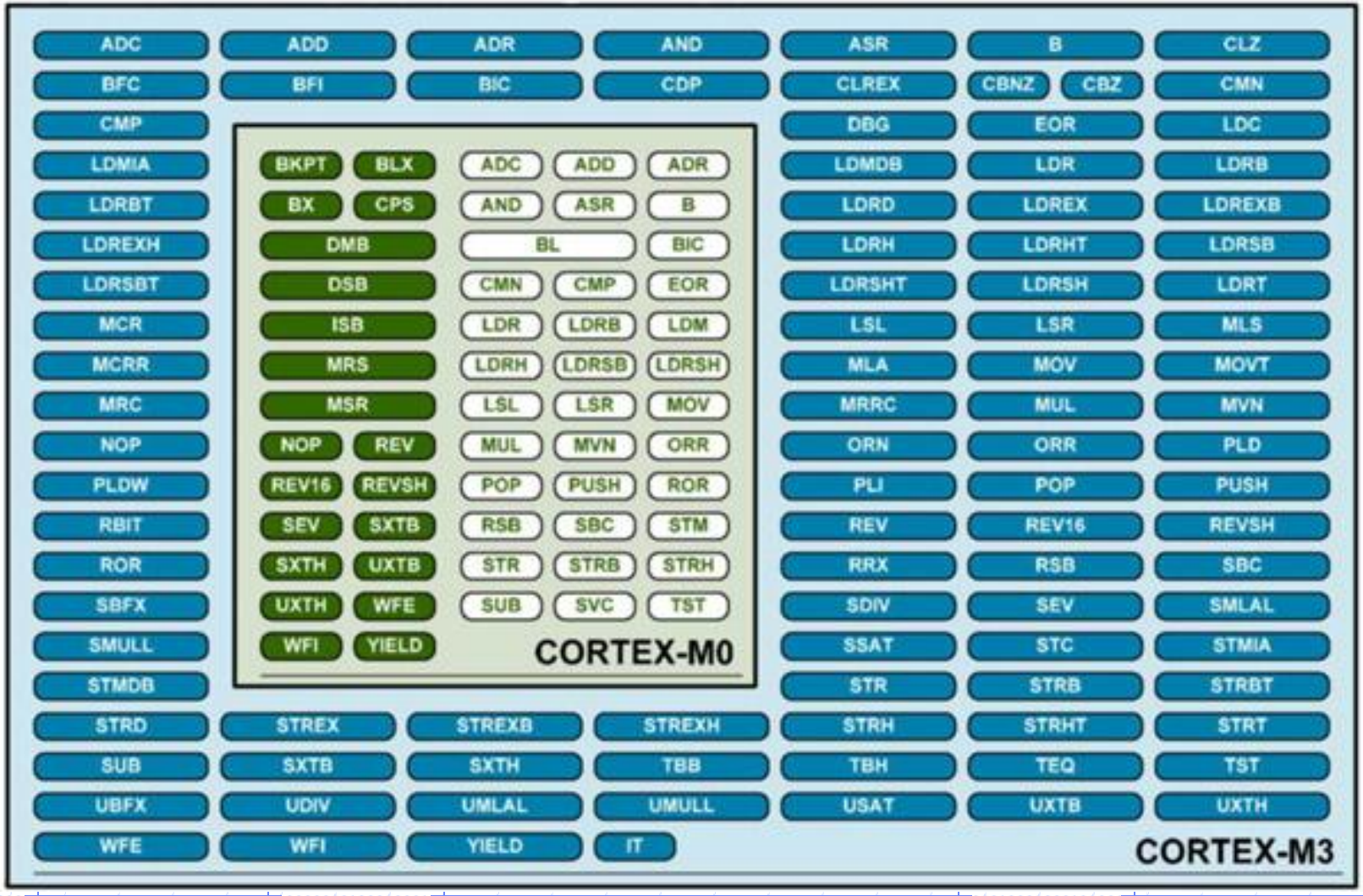
ARMv7 ISA – Data Processing

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

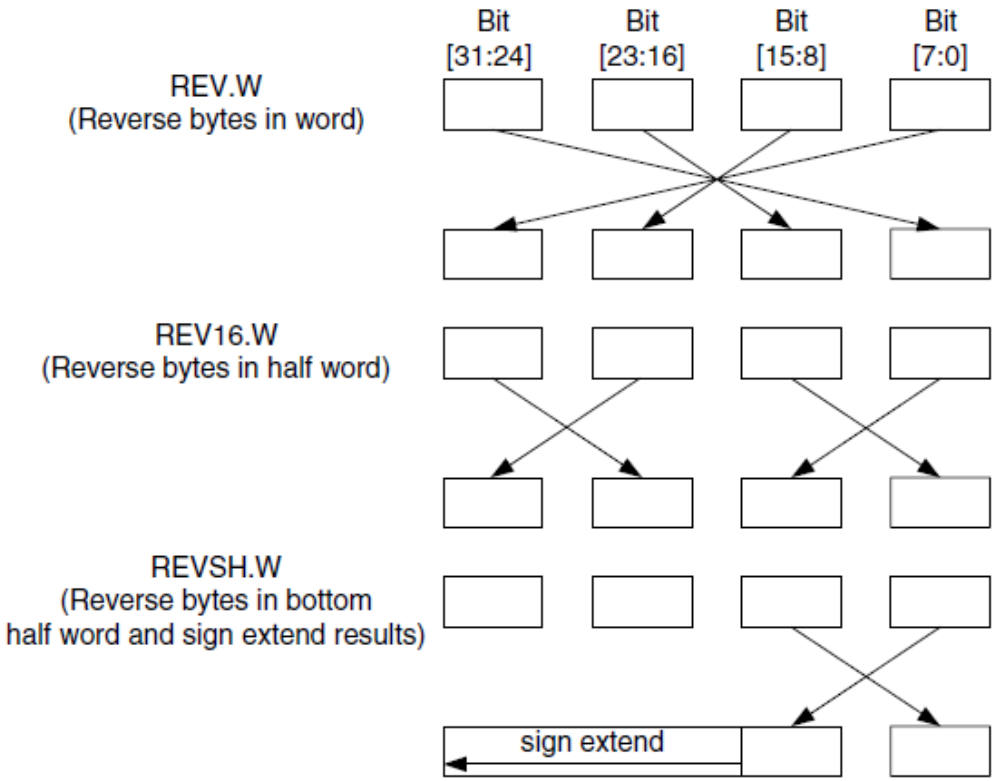
ARMv7 ISA – Data Processing

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

ARMv7 – Unique Instructions



ARMv7 Suffixes

- Instructions do not update the PSR unless a suffix ‘S’ is appended to the instruction
  - i.e. ADD vs ADD**S**
- 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

`LDR.W  Rd,  [Rn, #offset]!`
`LDRB.W Rd,  [Rn, #offset]!`
`LDRH.W Rd,  [Rn, #offset]!`
`LDRD.W Rd1, Rd2,[Rn, #offset]!`
	Preindexing load instructions for various sizes (word, byte, half word, and double word)

`LDRSB.W Rd, [Rn, #offset]!` `LDRSH.W Rd, [Rn, #offset]!`	Preindexing load instructions for various sizes with sign extend (byte, half word)

`STR.W  Rd,  [Rn, #offset]!`
`STRB.W Rd,  [Rn, #offset]!`
`STRH.W Rd,  [Rn, #offset]!`
`STRD.W Rd1, 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

ARMv7 Conditional Execution & Suffixes

- 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

ARMv7 Conditional Execution & Suffixes

- Conditional Branches

Table 4.1 Suffixes in Instructions

Suffix	Description
S	Update Application Program Status register (APSR) (flags); for example: `ADDS 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: `BEQ <Label> ; Branch if equal`

BNE, BLT, BGT etc

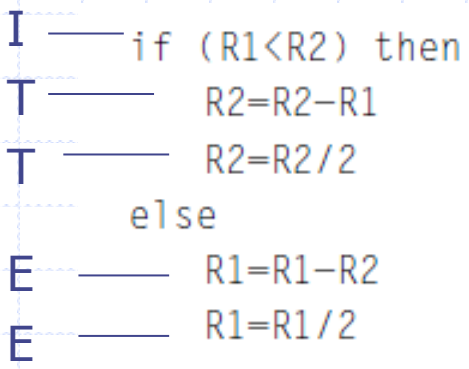
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR (this is actually a 32-bit instruction, but it is also available in Thumb in traditional ARM processors)
BLX	Branch with link and change state (BLX <reg> only)[1]
BX <reg>	Branch with exchange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

ARMv7 Conditional Execution & Suffixes

- 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

ARMv7 Conditional Execution & Suffixes

- Example of ITTEE block



ARMv7 Conditional Execution & Suffixes

- Example of ITTEE block

```
if (R1<R2) then
    R2=R2-R1
    R2=R2/2
else
    R1=R1-R2
    R1=R1/2
```

```
CMP     R1, R2 ; If R1 < R2 (less then)
ITTEE   LT     ; then execute instruction 1 and 2
               ; (indicated by T)
               ; else execute instruction 3 and 4
               ; (indicated by E)
SUBLT.W R2,R1  ; 1st instruction
LSRLT.W R2,#1  ; 2nd instruction
SUBGE.W R1,R2  ; 3rd instruction (notice the GE is
               ; opposite of LT)
LSRGE.W R1,#1  ; 4th instruction
```

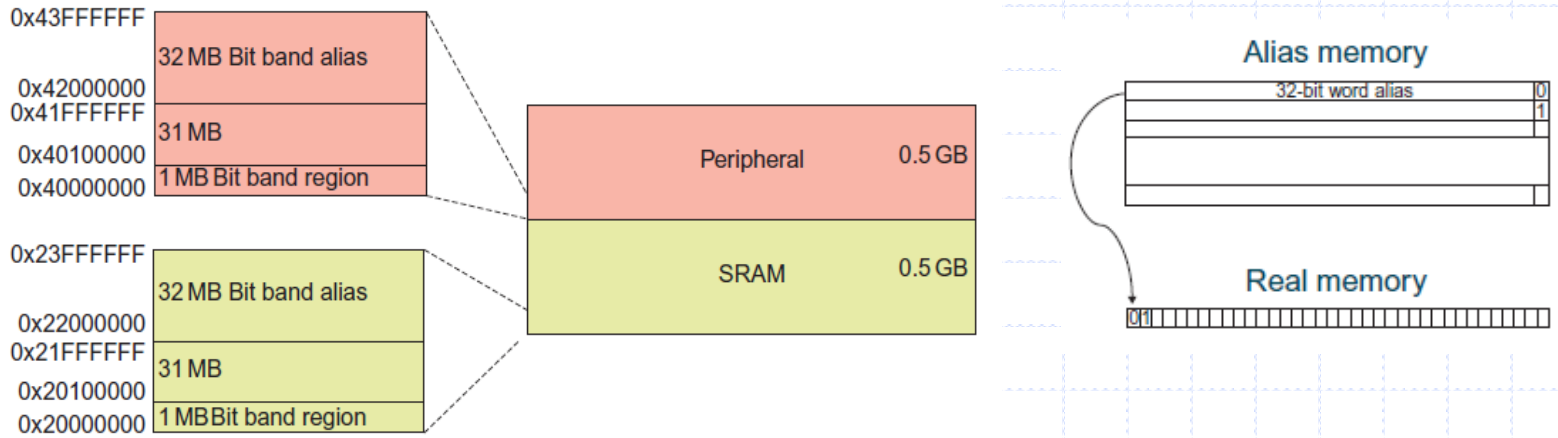
ARMv7 Conditional Execution & Suffixes

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

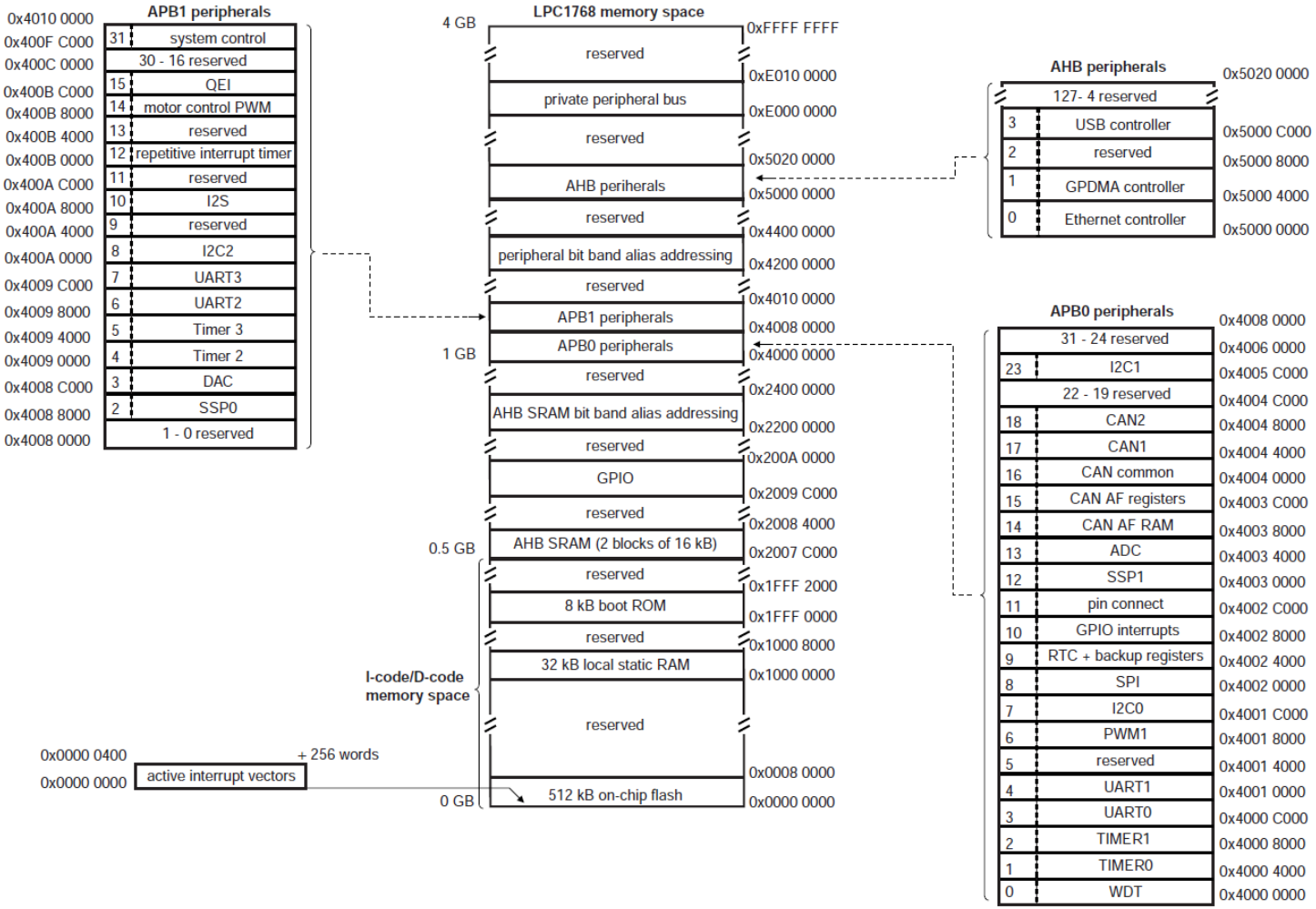
```
CMP R0, R1    ; Compare R0 and R1
ITTEE GT      ; If R0 > R1 Then
              ; if true, first 2 statements execute,
              ; if false, other 2 statements execute
MOVGT R2, R0 ;      R2 = R0
MOVGT R3, R1 ;      R3 = R1
MOVLE R2, R0 ; Else R2 = R1
MOVLE R3, R1 ;      R3 = R0
```

Bit-Banding

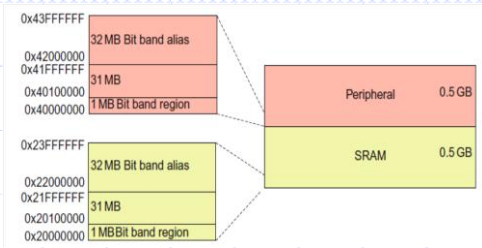


- Address 0x20000000 = SRAM
- 0x40000000 = Peripheral = external RAM, devices, vendor specific memory etc

Bit-Banding


APB1 peripherals
0x4010 0000
0x400F C000
0x400C 0000
0x400B C000
0x400B 8000
0x400B 4000
0x400B 0000
0x400A C000
0x400A 8000
0x400A 4000
0x400A 0000
0x4009 C000
0x4009 8000
0x4009 4000
0x4009 0000
0x4008 C000
0x4008 8000
0x4008 0000
31 system control
30 - 16 reserved
15 QEI
14 motor control PWM
13 reserved
12 repetitive interrupt timer
11 reserved
10 I2S
9 reserved
8 I2C2
7 UART3
6 UART2
5 Timer 3
4 Timer 2
3 DAC
2 SSP0
1 - 0 reserved
LPC1768 memory space
4 GB
0xFFFF FFFF
reserved
0xE010 0000
private peripheral bus
0xE000 0000
reserved
0x5020 0000
AHB periherals
0x5000 0000
reserved
0x4400 0000
peripheral bit band alias addressing
0x4200 0000
reserved
0x4010 0000
APB1 peripherals
0x4008 0000
APB0 peripherals
1 GB
0x4000 0000
reserved
0x2400 0000
AHB SRAM bit band alias addressing
0x2200 0000
reserved
0x200A 0000
GPIO
0x2009 C000
reserved
0x2008 4000
AHB SRAM (2 blocks of 16 kB)
0.5 GB
0x2007 C000
reserved
0x1FFF 2000
8 kB boot ROM
0x1FFF 0000
reserved
0x1000 8000
32 kB local static RAM
0x1000 0000
I-code/D-code memory space
reserved
0x0008 0000
512 kB on-chip flash
0 GB
0x0000 0000
0x0000 0400
+ 256 words
0x0000 0000
active interrupt vectors
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
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
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
0 WDT 0x4000 0000

Bit-Banding



Bit Band Word Address =

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

Byte Offset = 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

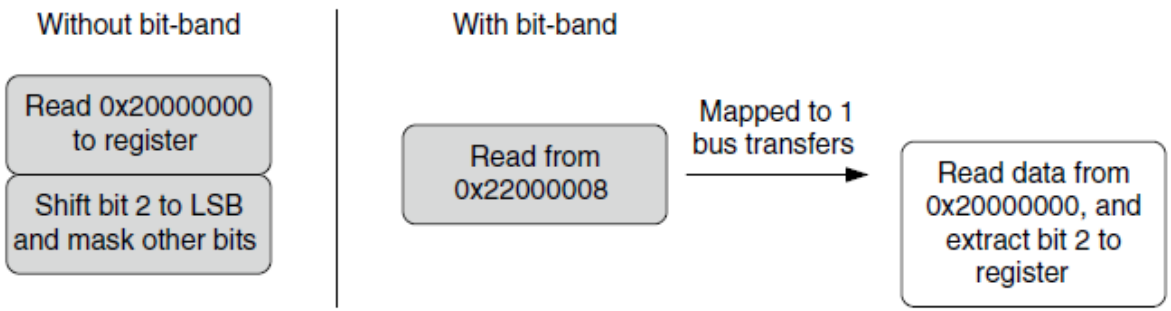


FIGURE 5.6

Read from the Bit-Band Alias.

Without bit-band

```
LDR    R0,=0x20000000 ; Setup address
LDR    R1, [R0]       ; Read
UBFX.W R1,  R1, #2, #1 ; Extract bit[2]
```

With bit-band

```
LDR   R0,=0x22000008 ; Setup address
LDR   R1, [R0]       ; Read
```

Embedded Systems

LETS PLAY.....

DESIGN THIS!

DESIGN THIS!

Intelligent Hanger
(Fashion & Engineering project)