HSR Hochschule für Technik Rapperswil

Name:

µ Computer Engineering 2

Quick-Reference / Summary

Version 2.4

Cortex Low-Power Leadership from ARM ARM

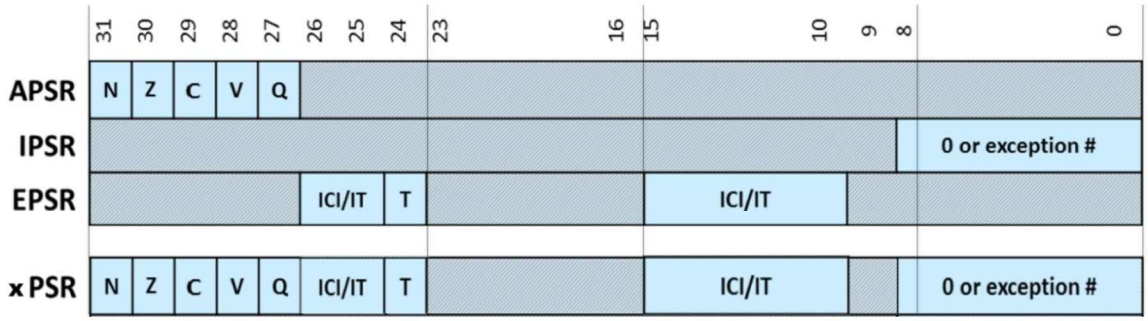
7-Bit ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Cortex-M3 Register-Set

Name		Functions (and banked registers)	
R0		General purpose register	Low registers
R1		General purpose register	
R2		General purpose register	
R3		General purpose register	
R4		General purpose register	
R5		General purpose register	
R6		General purpose register	
R7		General purpose register	
R8		General purpose register	High registers
R9		General purpose register	
R10		General purpose register	
R11		General purpose register	
R12		General purpose register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	
xPSR		Program status registers	Special registers
PRIMASK		Interrupt mask registers	
FAULTMASK			
BASEPRI			
CONTROL		Control register	

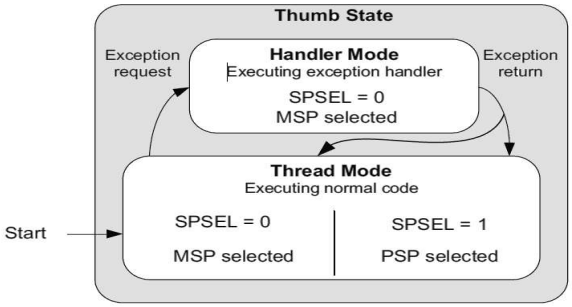
Cortex-M3 Program Status Register



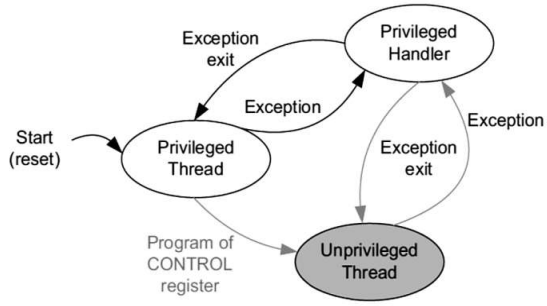
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception number	Indicates which exception the processor is handling

Document-Info:	CE2 Quick-Reference V2_4.docx						
NumPages	NumWords	NumChars	SaveDate	PrintDate			
12	1,583	7,787	11.08.20	11.08.20			

Cortex-M3 Operation-Modes und Stack-Pointer Auswahl



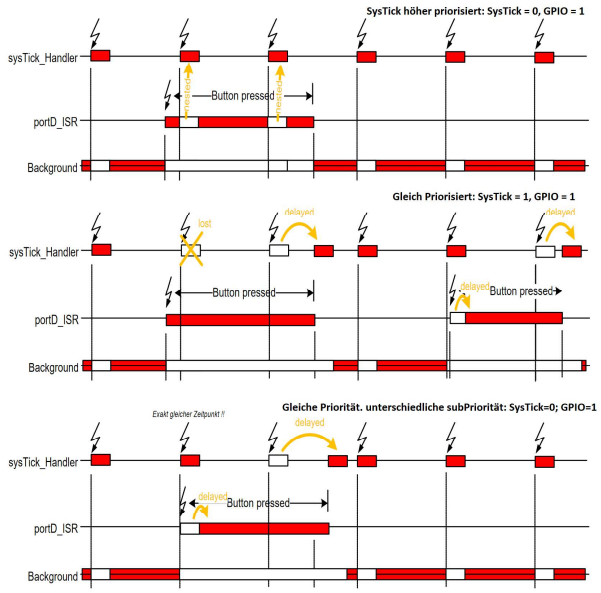
	31:3	2	1	0
CONTROL			SPSEL	nPRIV



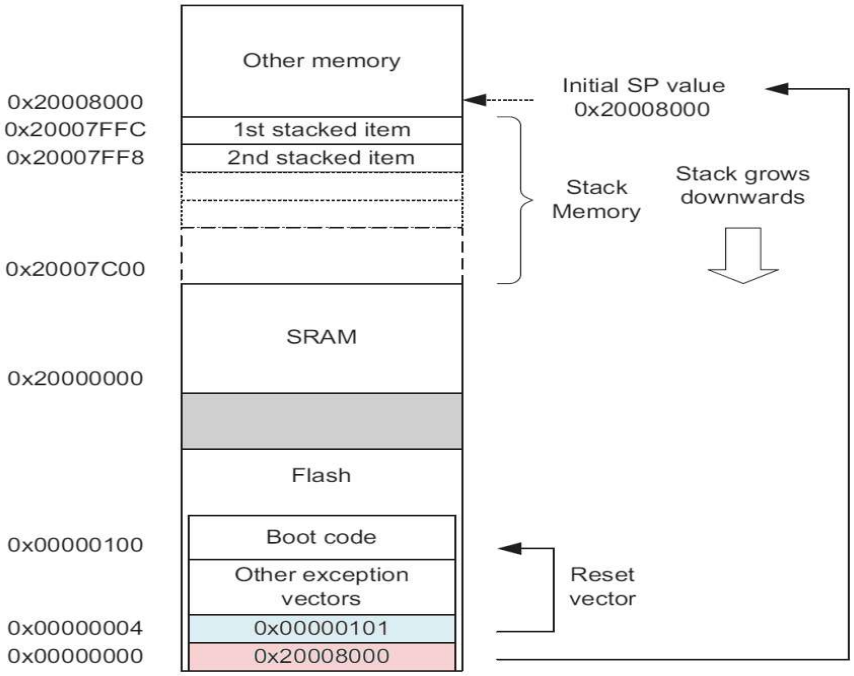
Cortex-M3 Exception-Vector-Table

Exception Type	CMSIS Interrupt Number	Address Offset	Vectors
18 - 255	2 - 239	0x48 – 0x3FF	IRQ #2 - #239
17	1	0x44	IRQ #1
16	0	0x40	IRQ #0
15	-1	0x3C	SysTick
14	-2	0x38	PendSV
NA	NA	0x34	Reserved
12	-4	0x30	Debug Monitor
11	-5	0x2C	SVC
NA	NA	0x28	Reserved
NA	NA	0x24	Reserved
NA	NA	0x20	Reserved
NA	NA	0x1C	Reserved
6	-10	0x18	Usage fault
4	-11	0x14	Bus Fault
4	-12	0x10	MemManage Fault
3	-13	0x0C	HardFault
2	-14	0x08	NMI
1	NA	0x04	Reset
NA	NA	0x00	Initial value of MSP

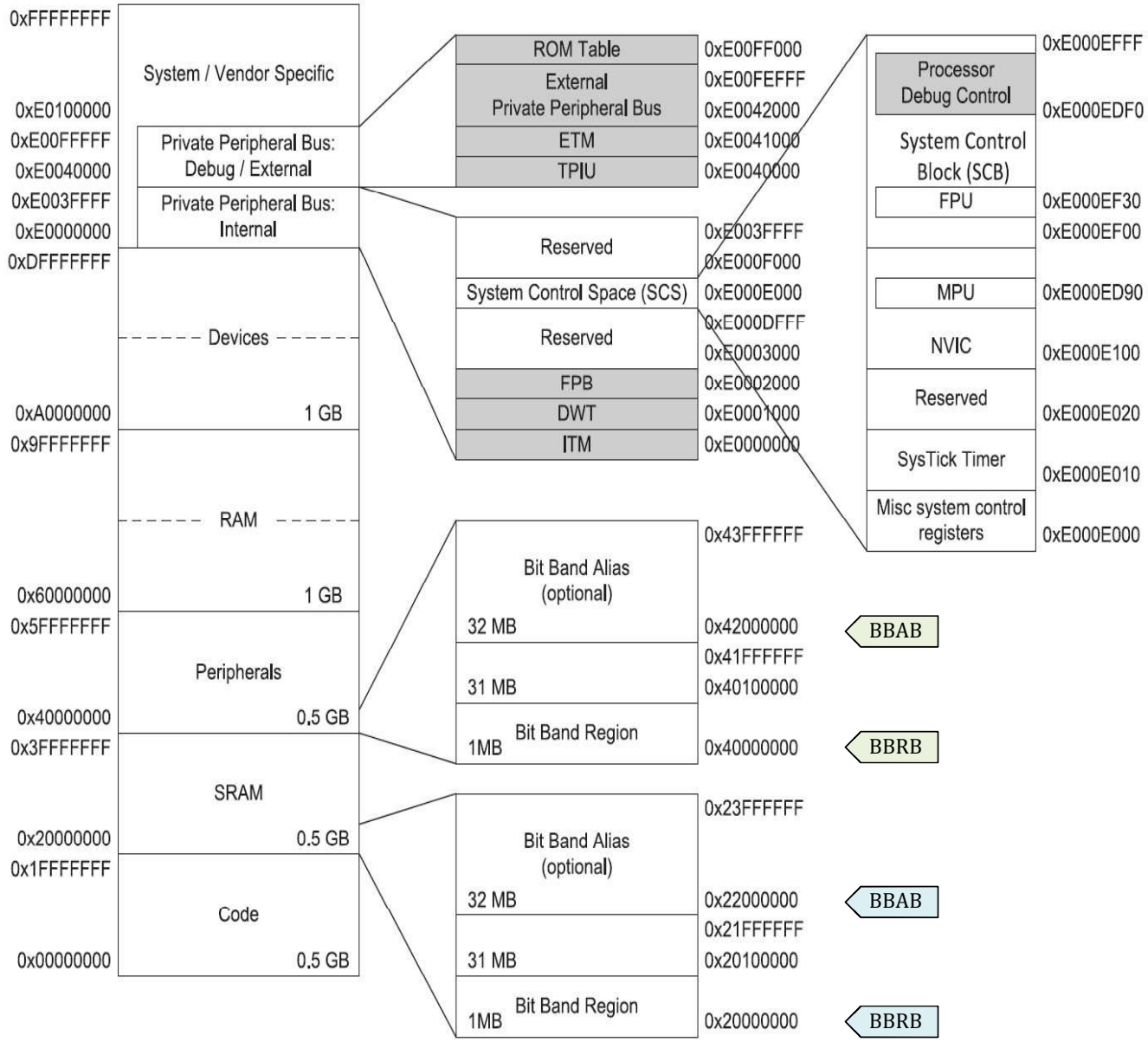
Je tiefer, deste höher die Priorität



Cortex-M3 Memory Organization und Reset Sequence



Cortex-M3 Memory-Map



Cortex-M3 Bit-Banding

$$BitBandAliasAddress = BitBandAliasBase + (MemoryAddress - BitBandRegionBase) * 32 + 4 * BitNumber$$

$$\boldsymbol{BBAA = BBAB + (MA - BBRB) * 32 + 4 * BNr}$$

Legende:

BBAA	= BitBandAliasAddress	= Word-Adresse des Bit-Band Alias
BBAB	= BitBandAliasBase	= Basisadresse der Bit-Band Alias (Speicherbereich)
MA	= MemoryAddress	= Memory-Adresse innerhalb der Bit-Band Region
BBRB	= BitBandRegionBase	= Basisadresse der Bit-Band Region (Speicherbereich)
BNr	= BitNumber	= Bit-Nummer in der Dateneinheit an der MemoryAddress

Assembler-Syntax *Keil vs. Code Composer Studio/GNU*

One of the difficulties in translating Keil to CCS is that the Keil syntax of **LDR R#,=Label** is not supported in CCS. So, to access variables and I/O ports we need to define a 32-bit pointer-constant using the .field pseudo-op. The actual machine code created by these two assemblers is virtually identical. The only difference is where in ROM the pointer-constant resides. In CCS you explicitly position the pointer-constants, and in Keil, the assembly automatically positions them.

```
;Keil
        THUMB
        AREA     DATA, ALIGN=2

        EXPORT   M
M       SPACE    4

    AREA |.text|,CODE,READONLY,ALIGN=2

PORTA   EQU 0x400043FC
BIT5    EQU 0x20
        EXPORT   InputPA5
InputPA5

        LDR R0,=PORTA    ;R0 = &PORTA
        LDR R1,[R0]      ;R1 = PORTA
        AND R1,R1,#BIT5  ;Mask
        LDR R2,=M        ;R2 = &M
        STR R1,[R2]      ;M = PA5
        BX  LR

        END
```

```
;CCS
        .thumb                    ;1)
        .data                     ;2)
        .align  4                 ;3)
        .global M                 ;4)
M       .field  32                ;5)
        .align  2                 ;6)
        .text                     ;7)
PtM     .field M,32               ;8)
PORTA   .field 0x400043FC,32      ;8)
BIT5    .equ    0x20              ;9)
        .global InputPA5          ;10)
        .thumbfunc InputPA5       ;11)
InputPA5: .asmfunc                ;12)
         LDR R0,PORTA             ;13)
         LDR R1,[R0]
         AND R1,R1,#BIT5
         LDR R2,PtM               ;13)
         STR R1,[R2]
         BX  LR
        .endasmfunc               ;12)
        .end                      ;14)
```

This illustrates the order and syntax of pseudo-ops in assembly files.

1) Use Thumb assembly language
2) This is a data section (variables typically go in RAM)
3) Align on 32-bit boundary
4) Declare the variable **M** globally visible to other files including to C programs
5) Define an uninitialized 32-bit object and call it M
6) Align on 16-bit boundary
7) This is a text section, which is executable code and callable from C (in ROM)
8) **.field** defines 32-bit objects and initialize them as pointers to M and to Port A
9) **.equ** defines a numerical constant
10) Declare it globally visible to other files including to C programs
11) There is a thumb function with this name
12) **.asmfunc** and **.endasmfunc** help with debugging, marking beginning and end
13) A pointer-constant is stored in ROM, and PC relative addressing is used
14) Marks the end of the file

Each compiler has its own syntax for handling inline assembly. The syntax for inline assembly in C is illustrated below. Both compilers follow the AAPCS convention for passing parameters and saving registers.

```
// Keil
__asm void Delay(unsigned long ulCount){
     subs r0, #1
     bne  Delay
     bx   lr
}
```

```
// CCS
void Delay(unsigned long ulCount){
__asm ( "     subs     r0, #1\n"
        "     bne      Delay\n"
        "     bx       lr\n");
}
```

This illustrates inline assembly in C programs.

The CCS code requires the quotation marks with a new line character at the end of each assembly line. This is a clever hack around to enable multiple lines to be written as one line. In essence Keil allows straight inline assembly, whereas in CCS you have to specify it as a string that will then be inserted. If you have to use assembly it is better to place it in a separate file, because inline assembly can be difficult to debug and makes the code less portable.

Cortex-M3 Instruction Set Details

Move / Add / Subtract	*Operation*	*{S}*	*<op>*	*Notes*
MOV R_d,<op>	R_d ← <op>	**NZ**	#imm16 or <op2>	
MOVT R_d,<op>	R_{dtop} ← <op>	**NZ**	#imm16	R_d[31:16]←imm16
MVN R_d,<op>	R_d ← ~<op>	**NZ**	<op2>	All bits are inverted first
ADD {R_d,}R_n,<op>	R_d ← R_n+ <op>	**NZCV**	#imm12 or <op2>	
ADC {R_d,}R_n,<op>	R_d ← R_n + <op> + C	**NZCV**		
SUB {R_d,}R_n,<op>	R_d ← R_n - <op>	**NZCV**		C=1: carry OR $\overline{\text{borrow}}$
SBC {R_d,}R_n,<op>	R_d ← R_n - <op> + C - 1	**NZCV**		C=1: carry OR $\overline{\text{borrow}}$
RSB {R_d,}R_n,<op>	R_d ← <op> - R_n	**NZCV**		C=1: carry OR $\overline{\text{borrow}}$

Multiply / Divide				
MUL {R_d,}R_n,R_m	R_d←R_n*R_m	**NZ**	32-bit product; C←undefined	
MLA R_d,R_n,R_m,R_a	R_d←(R_n*R_m)+R_a	-	32-bit product	
MLS R_d,R_n,R_m,R_a	R_d←R_a-(R_n*R_m)	-		
UMULL R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$←R_n*R_m	-	Unsigned 64-bit product	
UMLAL R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$←$R_{dhi}R_{dlo}$+R_n*R_m	-		
SMULL R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$←R_n*R_m	-	Signed 64-bit product	
SMLAL R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$←$R_{dhi}R_{dlo}$+R_n*R_m	-		
UDIV {R_d,}R_n,R_m	R_d←R_n/R_m	-	Unsigned 32-bit quotient; no remainder	
SDIV {R_d,}R_n,R_m	R_d←R_n/R_m	-	Signed 32-bit quotient; no remainder	

Compare / Test				
CMP R_n,<op>	R_n - <op>	*	C=1: carry OR $\overline{\text{borrow}}$	Always updates: NZCV
CMN R_n,<op>	R_n + <op>	*	C=1: carry OR $\overline{\text{borrow}}$	Always updates: NZCV
TST R_n,<op>	R_n & <op>	*	<op2>	Always updates: NZC
TEQ R_n,<op>	R_n ^ <op>	*		Always updates: NZC

Bitwise Logic				
AND {R_d,}R_n,<op>	R_d ← R_n & <op>	**NZC**	<op2>	
ORR {R_d,}R_n,<op>	R_d ← R_n | <op>	**NZC**		
EOR {R_d,}R_n,<op>	R_d ← R_n ^ <op>	**NZC**		
BIC {R_d,}R_n,<op>	R_d ← R_n & ~<op>	**NZC**		
ORN {R_d,}R_n,<op>	R_d ← R_n | ~<op>	**NZC**		

Shift / Rotate				
ASR R_d,R_m,<op>	R_d ← R_m >> <op>	**NZC**	Rs or #n	Sign extends
LSL R_d,R_m,<op>	R_d ← R_m << <op>	**NZC**		Zero fills
LSR R_d,R_m,<op>	R_d ← R_m >> <op>	**NZC**		Zero fills
ROR R_d,R_m,<op>	R_d ← R_m >> <op>	**NZC**		Right rotate
RRX R_d,R_m	R_d ← R_m >> 1	**NZC**	Right rotate, with extend through carry	

Bits / Bytes / Halfword / Words				
CLZ R_d,R_m	R_d ← CountZeroes(R_m)	-	Count leading zeroes (0-32)	
RBIT R_d,R_m	R_d ← RevBits(R_m)	-	Reverses bit order in word	
REV R_d,R_m	R_d ← RevByteOrder(R_m)	-	Reverses byte order in word	
REV16 R_d,R_m	R_d ← RevHalfWords(R_m)	-	Reverses byte order in each halfword independ	
REVSH R_d,R_m	R_d ← RevLoHalf(R_m)	-	Reverses byte order lower halfword, sign extend	
SXTB {R_d,}R_m{,ROR #num}	R_d ← SignedByte(R_m)	-	bits to rotate #num=<0|8|16|24>	Sign extends to word
SXTH {R_d,}R_m{,ROR #num}	R_d ← SiqnedHalf(R_m)	-		
UXTB {R_d,}R_m{,ROR #num}	R_d ← UnsignedByte(R_m)	-		Zero extends to word
UXTH {R_d,}R_m{,ROR #num}	R_d ← UnsiqnedHalf(R_m)	-		

Hinweis: *Dies ist eine nicht vollständige Zusammenstellung und zeigt lediglich die am häufigsten verwendeten Thumb-2 Instruktionen.*

Bitfield	*Operation*	*{S}*	*<op>*	*Notes*
BFC R_d,#lsb,#width	R_d<bits> ← 0	-	Bit field clear	
BFI R_d,R_n,#lsb,#width	R_d<bits> ← R_n<lsb's>	-	Bit field insert	
SBFX R_d,R_n,#lsb,#width	R_d←R_n<bits>	-	Signed bit field extract	
UBFX R_d,R_n,#lsb,#width	R_d←R_n<bits>	-	Unsigned bit field extract	

Branch / Call	PC –relative Addressing			
B label	PC←label	-	PC rel.offs. range=+/-16MB	unconditional branch
B{cond} label	if cond: then PC←label	-		{cond} is an optional condition
BL{cond} label	if cond: PC←label; LR←retAdr	-		Subroutine call
BX{cond} R_m	if cond: PC←R_m	-	any value in R_m	Subroutine return when R_m=LR
BLX{cond} R_m	if cond: PC←R_m;LR←retAdr	-		Subroutine call indirect via R_m
CBZ R_n,label	If R_n = 0: PC←label	-	PC rel.offs range=+4…+130	Compare and Branch on Zero
CBNZ R_n,label	If R_n ≠ 0: PC←label	-		Comp. and Branch on Non-Zero
IT$c_1c_2c_3$ cond	Each c; is one of T, E, or empty	-	Controls 1-4 instructions in "IT block"	

Literal Pool	PC –relative Addressing **R_t**			
ADR R_d,label	R_d←label	-	label = PC relative offset =+/-4095	
LDR R_t,label	R_t←mem_{32}[label]	-	PC rel.offs. range=+/-4095	
LDRB R_t,label	R_t←mem_8[label]	-		Zero fills bits 31..8 of R_t
LDRH R_t,label	R_t←mem_{16}[label]	-		Zero fills bits 31..16 of R_t
LDRSB R_t,label	R_t←mem_8[label]	-		Sign extends in bits 31..8 of R_t
LDRSH R_t,label	R_t←mem_{16}[label]	-		Sign extends in bits 31..16 of R_t
LDRD R_t,R_{t2},label	R_{t2},R_t←mem_{64}[label]	-	range=+/-1020	

Load/Store Memory	Memory Access **R_d**			
LDR R_d,<mem>	R_d←mem_{32}[EA]	-	<mem> see Memory Access Modes	
LDRB R_d,<mem>	R_d←mem_8[EA]	-		Zero fills bits 31..8 of R_d
LDRH R_d,<mem>	R_d←mem_{16}[EA]	-		Zero fills bits 31..16 of R_d
LDRSB R_d,<mem>	R_d←mem_8[EA]	-		Sign extends in bits 31..8 of R_d
LDRSH R_d,<mem>	R_d←mem_{16}[EA]	-		Sign extends in bits 31..16 of R_d
STR R_d,<mem>	R_d→mem_{32}[EA]	-		
STRB R_d,<mem>	R_d→mem_8[EA]	-		
STRH R_d,<mem>	R_d→mem_{16}[EA]	-		
LDRD R_d,R_{d2},<mem>	R_{d2},R_d←mem_{64}[EA]	-		may not use R_m in <mem>
STRD R_d,R_{d2},<mem>	R_{d2},R_d→mem_{64}[EA]	-		may not use R_m in <mem>

Stack, Multiple Load/Store			
POP {reglist}	regs←mem[SP++]	-	reglist: not SP; may include one of PC or LR
PUSH {reglist}	regs→mem[--SP]	-	reglist: may not include SP or PC
LDM{IA|DB} R_n{!},{reglist}	regs←mem[R_n]; (Notes)	-	optional ! will update R_n=R_n+/-(4*#regs)
STM{IA|DB} R_n{!},{reglist}	regs→mem[R_n]; (Notes)	-	IA = increment after, DB = decrement before

Special Functions			
CPSIE {I|F}	PRIMASK or FAULTMASK←0	-	Enable interrupts or faults
CPSID {I|F}	PRIMASK or FAULTMASK←1	-	Disable interrupts or faults
MRS R_d,spec_reg	R_d←spec_reg	-	Read special register
MSR spec_reg,R_m	spec_reg←R_m	**NZCV**	Write special register

<op2> *Examples of flexible operand <op2> creating the 32-bit number.* (e.g. Rd = Rn+op2)

`ADD Rd, Rn, Rm`	`;op2 = Rm`		Rm is signed or unsigned
`ADD Rd, Rn, Rm, ASR #n`	`;op2 = Rm>>n`	Arithmetic Shift Right	Rm is signed
`ADD Rd, Rn, Rm, LSL #n`	`;op2 = Rm<<n`	Logical Shift Left	Rm is signed or unsigned
`ADD Rd, Rn, Rm, LSR #n`	`;op2 = Rm>>n`	Logical Shift Right	Rm is unsigned
`ADD Rd, Rn, Rm, ROR #n`	`;op2 = Rm>>n`	Rotate Right	Rm is signed or unsigned
`ADD Rd, Rn, Rm, LSR #n`	`;op2 = Rm>>n`	Rotate Right through Carry	Rm is signed or unsigned
`ADD Rd, Rn, #constant`	`;op2 = constant`	where X and Y are hexadecimal digits: constant produced by shifting an 8-bit unsigned value left by any number of bits in the form: **0x00XY00XY** or **0xXY00XY00** or **0xXYXYXYXY**	

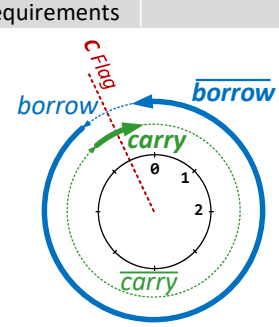
Symbols:

Symbols	Meaning
R_a R_d R_{d2} R_m R_n R_t R_{t2}	represent 32-bit registers
{R_d,}	if R_d is present R_d is destination, otherwise R_n
<op2>	the value generated by the flexible second operand op2
{S}	if S is present, instruction will set condition codes
{cond}	optional logical condition, condition code suffix
#imm12	any value in the range: 0...4095
#imm16	any value in the range: 0…65535
#offset	immediate: -255...+4095; pre/post-indexed: +/-255; LDRD/STRD: +/-1020
#n	any value in the range: 0… 31
value	any 32-bit value: signed, unsigned, or address
label	any address within the ROM of the microcontroller; offset range relative to PC for Branch/Call: +/-16MB; for ADR/LDR: +/-4095; for LDRD: +/-1020
R_m{,shift}	specifies an optional shift on R_m
R_n{,#offset}	specifies an optional offset to R_n
EA	Effective Address
{reglist}	List of registers, sequence is not relevant
.N or **.W**	Specify the use of 16-bit (narrow) instruction or 32-bit (wide) instruction.

{cond} Conditional Execution, Condition Code Suffix

Any one of these may be appended to any instruction mnemonic when used inside an If-Then-Else (IT) block.
(e.g., `IT` **NE** followed by ADD**NE** would add only if Z ≠ 0.) Exceptions: CBZ, CBNZ, CMP, CMN, NEG, TST, or TEQ

Condition Code	Meaning	Flags (APSR) / Requirements	unsigned	signed
EQ	Equal	Z = 1	X ==	X ==
NE	Not equal	Z = 0	X !=	X !=
CS or **HS**	Carry set, Unsigned ≥ carry OR $\overline{\text{borrow}}$	C = 1	X >=	
CC or **LO**	Carry clear,Unsigned < $\overline{\text{carry}}$ OR borrow	C = 0	X <	
MI	Min us/negative	N = 1		X -
PL	Plus/positive or zero (non-negative)	N = 0		X +
VS	Overflow	V = 1		X
VC	No overflow	V = 0		X
HI	Unsigned > ("Higher")	C = 1 && Z = 0	X >	
LS	Unsigned ≤ ("Lower or Same")	C = 0 || Z=1	X <=	
GE	Signed ≥ ("Greater than or Equal")	N = V		X >=
LT	Signed < ("Less Than")	N ≠ V		X <
GT	Signed > ("Greater Than")	Z = 0 && N = V		X >
LE	Signed ≤ ("Less than or Equal")	Z = 1 || N ≠ V		X <=
AL	Always (unconditional)	only used with IT instruction		



Rm{,shift} Shift Codes

Any of these may be applied to the register option of <op2> in Move/Add/Subtract, Compare, and Bitwise.

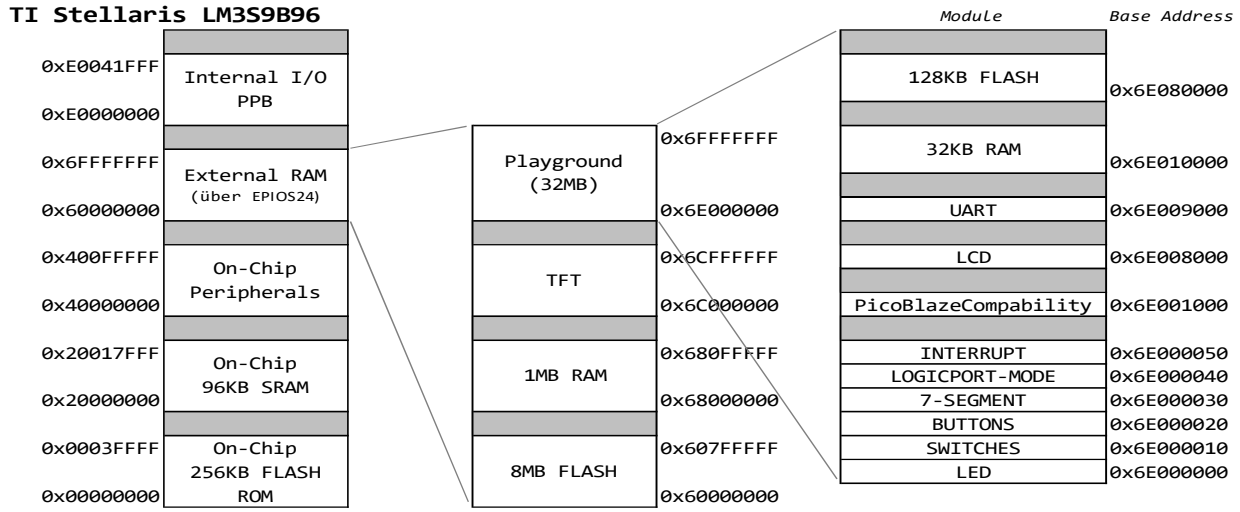
shift	Meaning	Notes
LSL #n	Logical shift left by n bits	Zero fills; 0 ≤ n ≤ 31
LSR #n	Logical shift right by n bits	Zero fills; 1 ≤ n ≤ 32
ASR #n	Arithmetic shift right by n bits	Sign extends; 1 ≤ n ≤ 32
ROR #n	Rotate right by n bits	1 ≤ n ≤ 32
RRX	Rotate right w/C by 1 bit	Through carry-flag

<mem> Memory Access Modes / Addressing Modes

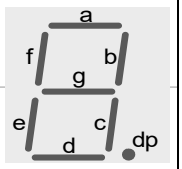
May be used with Load/Store Instructions LDR/STR. Exceptions: LDRD and STRD may not use R_m

Memory Access Mode	<mem>	EA=Effective Address	Example
Indirect addressing	**[R_n]**	EA ← R_n	[r2]
Immediate offset addressing	**[R_n,#offset]**	EA ← R_n+ offset	[r5,#100]
Register offset addressing{with shift}	**[R_n,R_m{,LSL #n}]**	EA ← R_n + (R_m << n)	[r4,r5,LSL #3]
Pre-indexed addressing	**[R_n,#offset]!**	R_n ← R_n+ offset; EA ← R_n	[r5,#100]!
Post-indexed addressing	**[R_n],#offset**	EA ← R_n; R_n ← R_n + offset	[rS],#100
Register	Rn		R2,R3
Register List	{Rn-Rm,PC}		R1-R3,PC
Register Immediate	Rn, #imm		R2,#100
Register Indirect with Index	[Rn,Rm]	EA<-- Rn+Rm	R0,[R1,R2]
PC-Relative	label		B

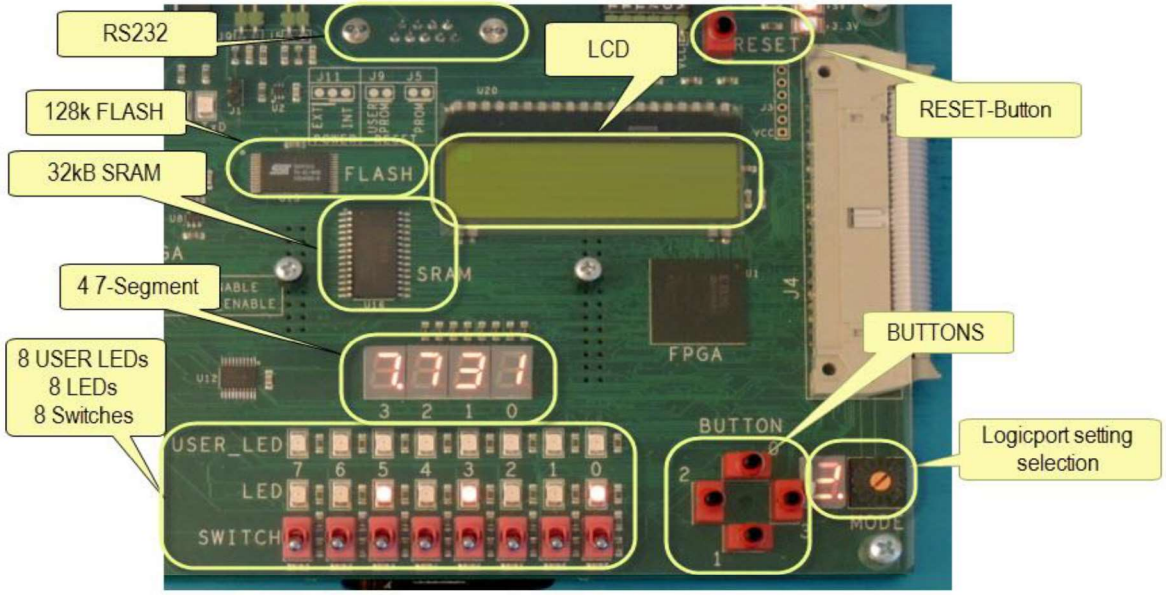
Hardware-Platform: Memory-Map



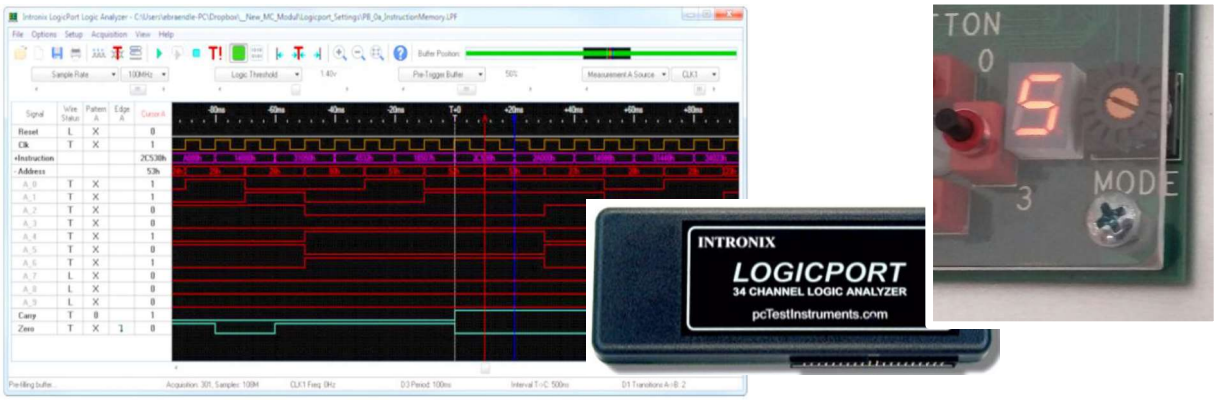
Hardware-Platform: Register Description

Module	Offset	Name	Type	Reset	Short Description <..> = BitNr.
LED	0x0	HW_LED_DATA	W	0x00	<7..0> Data ('1': on, '0': off)
SWITCHES	0x0	HW_SWITCH_DATA	R	0x??	<7..0> Data ('1': up, '0': down)
	0x1	HW_SWITCH_INTERRUPT_ENABLE	R/W	0x00	<7..0> Interrupt Enable ('1': enable, '0': disable)
	0x2	HW_SWITCH_INTERRUPT_BOTHEDGE	R/W	0x00	<7..0> Interrupt Both Edges ('1': both edge, '0': single edge)
	0x3	HW_SWITCH_INTERRUPT_EDGE	R/W	0x00	<7..0> Interrupt Edge ('1': rising, '0': falling)
	0x4	HW_SWITCH_INTERRUPT_STATUS	R/W	0x00	<7..0> R: Interrupt Status ('1': pending, '0': not pending) / W: Interrupt Clear ('1': clear, '0' no
BUTTONS	0x0	HW_BUTTON_DATA	R	0x??	<3..0> Data ('1': pressed, '0': released)
	0x1	HW_BUTTON_INTERRUPT_ENABLE	R/W	0x00	<3..0> Interrupt Enable ('1': enable, '0': disable)
	0x2	HW_BUTTON_INTERRUPT_BOTHEDGE	R/W	0x00	<3..0> Interrupt Both Edges ('1': both edge, '0': single edge)
	0x3	HW_BUTTON_INTERRUPT_EDGE	R/W	0x00	<3..0> Interrupt Edge ('1': rising, '0': falling)
	0x4	HW_BUTTON_INTERRUPT_STATUS	R/W	0x00	<3..0> R: Interrupt Status ('1': pending, '0': not pending / W: Interrupt Clear ('1': clear, '0' no action)
7-SEGMENT	0x0	HW_SEVENSEG0HEX_DATA	W	0x00	<7..4> reserved, <3..0> Value interpreted as hex digit (e.g. 0x0E displays hex digit 'E' on segments) Note: decimal point is not accessible, i.e. always pd = off 
	0x1	HW_SEVENSEG1HEX_DATA	W	0x00	
	0x2	HW_SEVENSEG2HEX_DATA	W	0x00	
	0x3	HW_SEVENSEG3HEX_DATA	W	0x00	
	0x4	HW_SEVENSEG0_DATA	W	0x00	Binary Data for each single-segment ('1': on, '0': off) <7> dp, <6> g, <5> f, <4> e, <3> d, <2> c, <1> b, <0> a
	0x5	HW_SEVENSEG1_DATA	W	0x00	
	0x6	HW_SEVENSEG2_DATA	W	0x00	
	0x7	HW_SEVENSEG3_DATA	W	0x00	
LOGICPORT-MODE	0x0	HW_LOGICPORTMODE_DATA	R/W	-	<3..0> Value of Logicportsetting (Dragwheel)
INTERRUPT	0x0	HW_INTERRRUPT_STATUS	R	0x00	<7> PicoBlazeCompatibility, <2> UART, <1> BUTTON, <0> SWITCH ('1': pending, '0': no)
PicoBlaze Compability	0x20	HW_PBC_ISTATUS	R/W	0x00	<4> BFLAG, <0> TFLAG
	0x21	HW_PBC_ICONFIG	R/W	0x00	<7> ICE, <6> reserved, <5> BSTEADY, <4> BIE, <3..2> TBASE, <1> TOFF, <0> TIE
	0x22	HW_PBC_TRELOAD	W	0xFF	<7..0> Reload Value
	0x23	HW_PBC_TVALUE	R/W	0x00	<7..0> Counter Value
	0x30	HW_PBC_LEDPORT	W	0x00	<7..0> Data ('1': on, '0': off)
	0x31	HW_PBC_BMPORT	R	0x??	<7> BUTTON0 ('1': pressed, '0': released), <3..0> Logicportsetting (Dragwheel)
	0x32	HW_PBC_SWITCHPORT	R	0x??	<7..0> Data ('1': up, '0': down)
	0x40	HW_PBC_SEVENSEG0	W	0x00	Binary Data for each single-segment ('1': on, '0': off) <7> dp, <6> g, <5> f, <4> e, <3> d, <2> c, <1> b, <0> a
	0x41	HW_PBC_SEVENSEG1	W	0x00	
	0x42	HW_PBC_SEVENSEG2	W	0x00	
	0x43	HW_PBC_SEVENSEG3	W	0x00	
LCD	0x0	HW_LCD_DATA	R/W	-	3x16 Character LCD (see Datasheet EA DOGM163L-A)
	0x1	HW_LCD_COMMAND	W	-	
UART	0x0	HW_UART_DATA	R/W	-	UART IC with 16B-FIFO (see Datasheet SC16C550BIB48)
	0x0	HW_UART_DIVISOR_LOW	-	-	
	0x1	HW_UART_DIVISOR_HIGH	-	-	
	0x1	HW_UART_INTERRUPT_ENABLE	-	-	
	0x2	HW_UART_FIFO_CONTROL	-	-	
	0x2	HW_UART_INTERRUPT_STATUS	-	-	
	0x3	HW_UART_LINE_CONTROL	-	-	
	0x4	HW_UART_MODEM_CONTROL	-	-	
	0x5	HW_UART_LINE_STATUŚ	-	-	
	0x6	HW_UART_MODEM_STATUS	-	-	
	0x7	HW_UART_SCRATCHPAD	-	-	.
RAM	-	HW_SRAM_DATA (BASE)	R/W	-	Data from RAM IC (see Datasheet BS62LV256SIP55)
FLASH	-	HW_FLASH_DATA	R/W	-	Data from FALSH IC (see Datasheet SST39VF010)
	0x5555	HW_FLASH_WP1	W	-	
	0x2AAA	HW_FLASH_WP2	W	-	

Hardware-Platform: Controls



Hardware-Platform: LogicPort Analyzer



Hardware-Platform: LogicPort Settings

Signal / Signalgruppe	Reset	Address Latch Enable	WRITE_n	READ_n	ADDRESS [27:0]	ADDRESS [19:0]	ADDRESS [3:0]	DATA [7:0]	INTERRUPT	INTERRUPT_CSn	LED_LATCH_ENABLE	LED_CSn	LED [7:0]	SEVENSEG_ENABLE [3:0]	SEVENSEG [7:0]	SEVENSEG_CSn	BUTTONS [3:0]	BUTTONS_CSn	BUTTONS_INTERRUPT	SWITCHES [7:0]	SWITCHES_CSn	SWITCHES_INTERRUPT	UART_CSn	UART_INTERRUPT	UART_Rx	UART_Tx	UART_RxD	UART_TxD	LCD_CSn	SRAM_CSn	FLASH_CSn	LOGICPORTMODE_CSn	USERLED_CSn	USERSWITCHES_CSn	USERBUTTONS_CSn	STUD_FPGA_INTERRUPT
0_Address.LPF		x	x	x	**x**																															
1_Address_Data.LPF		x	x	x		**x**		**x**																												
2_Switch_Button.LPF		x	x	x			**x**	**x**									**x**	x	x	**x**	x	x														
3_7Seg_Led.LPF		x	x	x			**x**	**x**			x	x		**x**	**x**	x																				
4_Interrupts.LPF		x	x	x			**x**	**x**	x	x									x			x		x												x
5_AddressMapping.LPF		x	x	x				**x**		x		x				x		x			x		x						x	x	x	x	x	x	x	
6_Interrupt_L_B_S.LPF									x				**x**				**x**		x	**x**	x															
7_Uart.LPF	x																							x	x	x	x	x								

Personal Notes

Personal Notes