sf16

16-bit microporcessors

Quick Reference Guide

V0.95
November 2013

Author: Martin Raubuch

Property of RACORS GmbH

info@racors.com

Introduction

The sf16 is a 16-bit microprocessor architecture for embedded control & computing applications with limited code size requirements. Main focus of the ISA (Instruction Set Architecture) definition is on high clock rates and small core implementations. Two ISA versions are available:
sf16b: base (b) ISA for general purpose control & computing
sf16d: dsp (d) extension: base ISA with extensions for DSP applications

The sf16 is a load/store architecture. All operands of computation instructions are either constants or contained in registers. Load/store instructions are used to transfer operands between registers and memory. The sf16d ISA slightly deviates from this concept. Some of the additional instructions have one memory source operand to improve performance.

The sf16 base ISA defines a generic and complete instruction set for efficient high level language compiler implementations.

sf16b (base ISA) features

- Harvard architecture with separate instruction and data interfaces
- 128kBytes instruction address space (can be extended up to 16Mbytes)
- 64kBytes data address space
- Fixed length 16-bit instruction coding
- 16 interrupts with programmable start addresses
- 8 x 16-bit general purpose registers and 8 special registers
- Native support for 8-bit and 16-bit signed and unsigned integer data types
- Higher precision integer and float data types supported by multi-instruction sequences
- Rich set of load/store addressing modes, including indirect with index and update addressing
- Little endian byte ordering
- Load/store multiple instructions for code efficient copying and function prologue/epilogue
- Bit manipulation & test instructions: set, clear, toggle & test
- 16*16 multiply with either 16-bit high word or 16-bit low word results
- Flexible debug concept with application specific debug modules

sf16d (DSP extension ISA) additional features

- multiply high instructions with optional left-shift and with one source operand read from memory
- multiply and accumulate instruction with optional left-shift and with one source operand read from memory
- multiply and subtract instruction with optional left-shift and with one source operand read from memory
- clip, clip with left-shift and clip to unsigned byte instructions
- Two registers with accumulation extension cache for sum-of-products calculations with 32-bit precision

General Purpose Registers

Register Bits				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rn	An	Name																	
0		R0	8 x 16-bit general purpose registers	R0															
1		R1		R1															
2		R2		R2															
3		R3		R3															
4		R4		R4															
5		R5		R5															
6	2	R6		R6															
7	3	R7		R7															

Registers R6 and R7 can also be used as indirect address An, the 3rd and 4th indirect address register are special registers TA and SP

Special Registers

Register Bits				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRn	An	Name	8 special registers																
0		CC	Condition Codes									RND				N	Z	O	C
1		CS	Control & Status	MS	AS	IS	IE	IR	IVTP										
2		LC	Loop Counter	LC															
3		AU	Address Update	AU															
4	0	SP	Stack Pointer	SP															
5	1	TA	Target Address (indirect jump)	TA															
6		SA	Subroutine (return) Address	SA															
7		ID	Core ID	REV				IMA				ISA				FML = 5			

Register Fields		
CC	C	Carry flag
	O	Overflow flag
	Z	Zero flag
	N	Negative flag
	RND	Round control, sf16d only,0: no round, 1-10 adds (1 << (RND+13)) to 32-bit multiply products
CS	IVTP	Interrupt Vectors Table Pointer, defines the 11 MSBs [15:5] of the interrupt vector table start address
	IR	Interrupt, 0: not in an interrupt, 1: interrupt processing
	IE	Interrupt Enable, 0: interrupts disabled, 1: interrupts enabled
	IS	Interrupt (enable) Shadow, used to save/restore IE when scie/rsie instructions are executed
	AS	Address select, if '1' accesses to the SA registers actually access the interrupt return address
	MS	Multiply Shift, enable a left-shift for the mlhnd, mlhsd, massd amd macsd instr., d ISA only
ID	FML	Core Family, 1: eco16, 2: eco32, 4: sf32, 5: sf16
	ISA	ISA = 1 = b (base), 2 = d (dsp)
	IMA	Implementation Architeture, 1: l = light, 4: u = ultralight
	REV	Revision, starting with 1

The MS bit of register CS and the RND field of register CC are availabe only in the d (DSP) ISA

Load/Store instructions

Operand Symbols		Operand Addressing													
DA8	8-bit absolut data address, scaled 0x0000-0x00FF (byte), 0x0000-0x01FE (short)	(DA8),Rd	($DO5_S$,An),Rd	(Rx,An),Rd	(An)+,Rd	-(An),Rd	(An)*,Rd	(An)+,RGS	Rs,(DA8)	Rs,($DO5_S$,An)	Rs,(Rx,An)	Rs,(An)+	Rs,-(An)	Rs,(An)*	RGS,-(An)
$DO5_S$	5-bit data address offset, signed, scaled, including zero -16,-14, ,… 15 (byte), -32, -30, … 30 (short)														
Rs	register Rn (R0-R7) used as source operand														
Rd	register Rn (R0-R7), used as destination operand														
Rx	register Rn (R0-R7), used as index operand														
An	registers An (SP, TA, R6, R7) used as indirect address														
RGS	register selection, any selection of R2, R3, R4, R5, R6, and R0, R1 (byte only) or TA, SA (short only)														
Mnemo	Description	Addressing Modes													
ldbt	load byte (8-bit) and zero-extend to 16 bits	*	*	*	*	*	*	*							
stbt	store byte (8-bit)								*	*	*	*	*	*	*
ldsh	load short (16-bit)	*	*	*	*	*	*	*							
stsh	store short (16-bit)								*	*	*	*	*	*	*

eda (16-bit effective data address) generation (Load/Store Addressing Modes)		
DA8	8-bit direct address	eda = size*DA8, size = 1 (byte), 2 (short)
($DO5_S$,An)	indirect with 5-bit signed offset	eda = An + size*$DO5_S$, size = 1 (byte), 2 (short)
(Rx,An)	indirect with scaled index	eda = An + size*Rx, size = 1 (byte), 2 (short)
(An)+	indirect with post-increment	eda = An, An += size, size = 1 (byte), 2 (short)
-(An)	indirect with pre-decrement	eda = An - size, size = 1 (byte), 2 (short), An = eda
(An)*	indirect with post-update	eda = An, An += AU (special register)

Flow Instructions

Operand Options		Oper. Addr.			
Implied	no operands or operands are implicitly defined	Implied	IA12	$IO8_S$	IAH
$IO8_S$	8-bit instruction address offset (16-bit word granularity), -128 to 127 instructions				
IA12	12-bit absolut instruction address in 16-bit word granularity, 0x0000 to 0x0FFF				
IAH	4-bit instruction address high (bits [15:12] of instr. Address), 0x1 - 0xF				

Mnemo	Description		Addr. Mode			
	jump, jump-to-subroutine, return					
jump	jump		*	*		
jpsr	jump to subroutine		*	*		
rtsr	return from subroutine		*			
rtir	return from interrupt		*			
	conditional branches					
mnemo	condition CND specified as logical equation of C = CC.C, O = CC.O, Z = CC.Z and N = CC.N					
brnc	branch if no carry	CND = ~C			*	
brcr	branch if carry	CND = C			*	
brno	branch if no overflow	CND = ~O			*	
brof	branch if overflow	CND = O			*	
brnz	branch if non zero	CND = ~Z			*	
brzr	branch if zero	CND = Z			*	
brps	branch if positive	CND = ~N			*	
brng	branch if negative	CND = N			*	
brls	branch if lower or same	CND = C | Z			*	
brhi	branch if higher	CND = ~C & ~Z			*	
brlo	branch if lower	CND = (N & ~O) | (~N & O)			*	
brge	branch if greater or equal	CND = (N & O) | (~N & ~O)			*	
brle	branch if lower or equal	CND = Z | (N & ~O) | (~N & O)			*	
brgt	branch if greater	CND = ~Z & ((N & O) | (~N & ~O))			*	
bral	branch always	CND = 1 (true)			*	
brlc	branch if loop counter is unequal zero	LC -= 1, CND = LC != 0			*	
	other					
siah	set instruction address high, sets the hidden 4-bit *IAH* register					*
stie	set interrupt enable, sets IE bit in CS		*			
clie	clear interrupt enable, clears IE bit in CS		*			
rsie	restore interrupt enable, transfers IS bit of CS to IE bit of CS		*			
scie	save and clear interrupt enable, transfers IE bit of CS to IS bit of CS, then clears IE		*			
stas	set address select, sets AS bit in CS		*			
clas	clear address select, clears AS bit in CS		*			
stop	stop, enter stopped (debug) state		*			
svpc	save program counter (write to debug port)		*			
rspc	restore program counter (read from debug port)		*			

The 16-bit target address of jump and jpsr instructions with the IA12 addressing mode is the concatenation of the hidden 4-bit *IAH* register and the 12-bit absolute address IA12. After the jump the *IAH* register is cleared to zero. Without a preceding siah instruction the address range is limited to 4k instructions from 0x0000-0x0FFF. With a preceding siah instruction the full instruction address range from 0x0000-0xFFFF can be reached.

Arithmetic Computation Instructions

Operand Field Elements		Operand Addressing								
$C7_{SN}$	7-bit constant (Signed, Not including zero), -64 to 64									
$C8_{A}$	8-bit constant (Asymmetric) -64 to -1 and 0 to 255									
$C8_{UN}$	8-bit constant (Unsigned, Not including zero) 1 to 256									
$C16_{U}$	16-bit constant (Unsigned), 8 LSBs are not coded and are all zeros 0x0000 to 0xFF00									
Rs,Rs0,Rs1	register Rn, used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand									
Rd,Rb	register Rn, used as destination (Rd) or as both source and destination (Rb) operand									
Ad	register An (SP, TA, R6, R7), used as destination operand									
Mnemo	Description	Addressing Modes								
		$C8_{UN}$,Rb	$C16_{U}$,Rb	$C7_{SN}$,Ad	$C8_{A}$,Rs1	Rs0,Rs1	Rs	Rs,Rd	Rs0,Rs1,Rd	Cond. Code Updated
addt	add to	*							*	*
addc	add with carry, destination = source0 + source1 + CC.C								*	*
adcf	add carry flag, destination = source + CC.C							*		*
addh	add to high word		*							
adsp	add to stack pointer			*						
subf	subtract from, source0 from source1	*							*	*
subc	subtract with carry, destination = source1 - source0 - CC.C								*	*
sbcf	subtract carry flag, destination = source - CC.C							*		*
comp	compare (subtract source 0 from source 1)				*	*				*
cmpc	compare with carry					*				*
cpcf	compare carry flag (subtract CC.C from source)						*			*
negt	negate							*		
absl	absolute value							*		
clzr	count leading zeros							*		
sxbt	sign extend byte							*		
sxsh	sign extend short							*		
mult	multiply unsigned, 16*16 -> 32, store low word of result in destination								*	
mlhu	multiply high unsigned, 16*16 -> 32, store high word of result in destination								*	
mlhs	multiply high signed, 16*16 -> 32, store high word of result in destination								*	

Miscellaneous Instructions

Operand Options		Operand Addressing																	
Implied	no operands or operands are implicitly defined																		
$BTI4_U$	4-bit bit index (Unsigned), 0 to 15																		
$SHC4_U$	4-bit shift count (Unsigned), 0 to 15																		
$C7_U$	7-bit constant (Unsigned) 0 to 127																		
$C7_{UN}$	7-bit constant (Unsigned, Not including zero) 1 to 128																		
$C7_{SN}$	7-bit constant (Signed, Not including zero) -64 to 64																		
$C8_U$	8-bit constant (Unsigned) 0 to 255																		
$C9_S$	9-bit constant (Signed), -256 to 255																		
Rs,Rs0,Rs1	register Rn, used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand																		
Rd, Rb	register Rn, used as destination operand (Rd) or both source and destination operand (Rb)																		
SRs, SRd	special register SRn, used as source (SRs) or destination (SRd) operand																		
Mnemo	Description	$BTI4_U$,Rs	$BTI4_U$,Rb	$SHC4_U$,Rb	$C7_U$,CC	$C7_U$,CS	$C7_{UN}$,LC	$C7_{SN}$,AU	$C9_S$,Rd	$C8_U$,Rs1,Rd	Rs0,Rs1	Rs,Rd	Rs,SRd	SRs,Rd	Rs0,Rs1,Rd	Rd	Rs	Condition Code Updated
		Addressing Modes																
	Logic																	
andb	logical and bit wise, also calculates parity of the result									*					*			*
iorb	logical inclusive or bit wise									*					*			
xorb	logical exclusive or bit wise														*			
invt	invert											*						
	Move																	
move	move								*			*						
mfsr	move from special register													*				
mtsr	move to special register				*	*	*	*					*					
mfdp	move from debug port, transfers the debug port input value to the destination operand															*		
mtdp	move to debug port, transfers the source operand to the core's debug port																*	
	Shift																	
shlz	shift left with zero fill			*											*			
shlf	shift left with feedback (from MSB)			*											*			
shru	shift right unsigned			*											*			
shrs	shift right signed			*											*			
	Bit Manipulation																	
btst	bit set		*												*			
btcl	bit clear		*												*			
bttg	bit toggle		*												*			
btts	bit test, does not update the destination register	*									*							*

DSP ISA Extension Instructions

Operand Options		Op. Adr.		
Implied	no operands or operands are implicitly defined	Rs,Rb	Rs,Rd	Rs0,(An)*,Ra
Rs,Rs0	register Rn, used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand			
Rb	register Rn used as destination or both source and destination			
Ra	accumulator register R4 or R5 used as destination or both source and destination			
Mnemo	Description			
addsd	left shift source operand 1 by one bit and add	*		
subsd	left shift source operand 1 by one bit and subtract	*		
clipd	clip signed to 0xC000 / 0x3FFF boundaries		*	
clshd	clip and shift, same as clip but with a 1-bit left shift after the clipping		*	
clubd	clip to unsigned byte, if < 0 clips to 0, if > 0xFF clips to 0xFF		*	
mlhsd	multiply signed, 16*16 -> 32, store high word of result in destination			*
mlnsd	multiply & negate signed, 16*16 -> 32, store high word of result in destination			*
macsd	multiply & accumulate signed, 16*16 -> 32, add high word of result to destination			*
massd	multiply & subtract signed, 16*16 -> 32, subtract high word of result from destination			*
mshsd	multiply & left-shift signed, 16*16 -> 32, store high word of result in destination			*
msnsd	multiply, left-shift & negate signed, 16*16 -> 32, store high word of result in destination			*
msasd	multiply, left-shift signed & accumulate, 16*16 -> 32, add high word of result to destination			*
msssd	multiply, left-shift signed & subtract, 16*16 -> 32, subtract high word of result from destination			*
m2hsd	multiply & 2-bit left-shift signed, 16*16 -> 32, store high word of result in destination			*
m2nsd	multiply, 2-bit left-shift & negate signed, 16*16 -> 32, store high word of result in destination			*
m2asd	multiply, 2-bit left-shift signed & accumulate, 16*16 -> 32, add high word of result from destination			*
m2ssd	multiply, 2-bit left-shift signed & subtract, 16*16 -> 32, subtract high word of result from destination			*