sf32

32-bit microprocessors

Quick Reference Guide

V1.0
February 2014

Author: Martin Raubuch

info@racors.com

Introduction

The sf32 family of 32-bit microprocessors is targeted at applications where high performance and small core sizes are most important. Fixed length 32-bit instruction coding enables low decoding complexity which results in high clock rates and small core foot prints. Multiple ISAs are available to address control & computing as well as DSP applications with optimized solutions.

- sf32b (base ISA) for general purpose control & computing, 32-bit data and instr. address spaces
- sf32d (dsp ISA) base ISA with 32-bit DSP extension for audio and 32-bit DSP applications

sf32b (base ISA)

The sf32b is a 32-bit microprocessor architecture for embedded control & computing applications. Main focus of the ISA definition is on high clock rates and small core implementations.

The sf32b is a load/store architectures. 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 sf32b defines a generic and complete instruction set for efficient high level language compiler implementations.

sf32b features

- Harvard architecture with separate instruction and data buses
- 4GBytes instruction address space
- 4GBytes data address space
- Fixed length 32-bit instruction coding
- 16 interrupts with programmable start addresses
- 24 x 32-bit general purpose registers and 7 special registers
- System (protected) and application operation modes
- Native support for 8-bit, 16-bit and 32-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
- 32*32 multiply with either 32-bit high word or 32-bit low word results
- Instructions for endianess conversion
- Flexible debug interface to connect application specific debug modules

sf32d (DSP ISA)

The sf32d is an extension of the sf32b base ISA and is fully backward compatible with the sf32b. Main target are 32-bit DSP and specifically audio applications. The DSP extension adds only a few special registers but no general purpose registers to the programming model of the sf32b ISA. Main additions are addressing modes with memory source operands and special add/subtract instructions that improve the performance of audio and general DSP algorithms. With the same pipeline architecture and computation resources implementations of sf32d processors are only slightly larger than base ISA implementations.

The sf32d deviates from the puristic load/store architecture of the sf32b. Some performance critical extension instructions have one source operand in memory.

The instructions with additional addressing modes and the additional, special add/subtract instructions can't be used easily from high level languages. The targeted use model suggests hand optimized assembler routines for performance critical DSP functions using the special addressing modes and instructions. The less performance critical higher layers and control code is written in C and compiled to the sf32b base ISA instruction set.

sf32d extension features

- Multiply-high and MAC (Multiply & Accumulate) instructions with one source operand in memory
- Optional 1-bit or 2-bit left-shift before accumulation
- Multiple indirect addressing modes for memory source operands with offset, index and auto-update
- Add/sub instructions with preceding left-shift of one source operand
- Clip to signed 16-bit, clip to signed 31-bit and clip to unsigned byte instructions
- 32-bit Iterative divide instruction
- Dual entries accumulation extension cache (patented) for sum-of-products calculation with 64-bit precision

Base ISA Registers

Register Bits				31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rn	An	Name		Physical Registers																															
0		R0	8 x 32-bit registers for data operands	R0																															
1		R1		R1																															
2		R2		R2																															
3		R3		R3																															
4		R4		R4																															
5		R5		R5																															
6		R6		R6																															
7		R7		R7																															
8	8	R8	16 x 32-bit registers for data operands or indirect addresses	R8																															
9	9	R9		R9																															
10	10	RA		RA																															
11	11	RB		RB																															
12	12	RC		RC																															
13	13	RD		RD																															
14	14	RE		RE																															
15	15	RF		RF																															
16	0	RP		RP																															
17	1	RQ		RQ																															
18	2	RU		RU																															
19	3	RV		RV																															
20	4	RW		RW																															
21	5	RX		RX																															
22	6	RY		RY																															
23	7	RZ		RZ																															
24		LC	Loop Counter	LC																															
25		CC	Condition Codes																												N	Z	O	C
26		CS	Control & Status	IVTP																												IS	IE	IR
27		EP	Extension ISA Parameters																																
28		TA	Target Address	TA																															
29		SA	Subroutine (return) Address	SA																															
30		IA	Interrupt (return) Address	IA																															
31		ID	Core ID																		REV			IMA			ISA			FML = 4				

Register Fields		
CC	C	Carry flag
	O	Overflow flag
	Z	Zero flag
	N	Negative flag
CS	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
	IVTP	Interrupt Vectors Table Pointer, defines the 26 MSBs [31:6] of the interrupt vector table start address in the data address space
ID	FML	Core Family, 1: eco16, 2: eco32, 4: sf32, 5: sf16
	ISA	ISA: 1: b = base, 2: d = dsp extension
	IMA	Implementation Architeture, 1: l = light, 4: u = ultralight
	REV	Revision, starting with 1

Load/Store instructions

Operand Symbols		Operand Addressing
$DA16_S$	16-bit absolut data address, signed, 0x00000000-0x00007FFF and 0xFFFF8000-0xFFFFFFFF	
$DO12_S$	12-bit data address offset, signed, -2048 to 2047	
$AU12_S$	12-bit address update, signed, -2048 to 2047	
Rs	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA,ID) used as source operand	
Rd	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA), used as destination operand	
Rx	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA), used as index operand	
Ru	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA), used as update operand	
An	registers An (R8-RZ) used as indirect address	
CND	condition, the instruction is executed only if the condition is true, 15 conditions	
RGS	register selection, any selection of R0-RB, RP-RV, LC, TA, SA	

Mnemo	Description	($DA16_S$),Rd,CND	($DO12_S$,An),Rd,CND	(An,$AU12_S$)*,Rd,CND	(An,Ru)*,Rd,CND	(Rx,An),Rd,CND	(An)+,RGS	-(An),RGS	Rs,($DA16_S$),CND	Rs,($DO12_S$,An),CND	Rs,(An,$AU12_S$)*,CND	Rs,(An,Ru)*,CND	Rs,(Rx,An),CND	RGS,(An)+	RGS,-(An)
		Addressing Modes													
ldbz	load byte (8-bit) and zero-extend	*	*	*	*	*	*	*							
ldbs	load byte (8-bit) and sign-extend	*	*	*	*	*	*	*							
stbt	store byte (8-bit)								*	*	*	*	*	*	*
ldsz	load short (16-bit) and zero extend	*	*	*	*	*	*	*							
ldss	load short (16-bit) and sign-extend	*	*	*	*	*	*	*							
stsh	store short (16-bit)								*	*	*	*	*	*	*
ldlg	load long (32-bit)	*	*	*	*	*	*	*							
stlg	store long (32-bit)								*	*	*	*	*	*	*

eda (32-bit effective data address for sf32 and 16-bit effective data address for sfr32) generation (Load/Store Addressing Modes)

$DA16_S$,CND	16-bit signed direct address	eda = $DA16_S$
($DO12_S$,An),CND	indirect with 12-bit signed offset	eda = An + $DO12_S$
(An,$AU12_S$)*,CND	indirect with 12-bit signed update	eda = An, An += $AU12_S$
(An,Ru)*,CND	indirect with indirect update	eda = An, An += Ru
(Rx,An),CND	indirect with scaled index	eda = An + size*Rx, size = 1 (byte), 2 (short), 4 (long)
(An)+	indirect with post-increment	eda = An, An += size, size = 1 (byte), 2 (short), 4 (long)
-(An)	indirect with pre-decrement	An -= size, eda = An, size = 1 (byte), 2 (short), 4 (long), An = eda

Flow Instructions

Operand Options		Oper. Addr.			
Implied	no operands or operands are implicitly defined	Implied	$IO16_S$	$IO16_S$,S	$IA29_U$
$IO16_S$	16-bit instruction address offset in bytes (32-bit word granularity), -32768 to 32764				
S	speculation, T (true) or F (false)				
$IA29_U$	29-bit absolut instruction address in 32-bit word granularity, 0x00000000 to 0x1FFFFFFC				

Mnemo	Description		Addr. Mode			
jump, jump-to-subroutine, return						
jump	jump (with the implied addressing mode the target address is in register TA)		*			*
jpsr	jump to subroutine (with the implied addressing mode the target address is in register TA)		*			*
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))			*	
brlc	branch if loop counter is unequal zero	LCT -= 1, CND = LCT != 0		*		
other						
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		*			
stop	stop, enter stopped (debug) state		*			
svpc	save program counter (write to debug port)		*			
rspc	restore program counter (read from debug port)		*			

Base ISA Arithmetic Computation Instructions

Operand Field Elements		Operand Addressing								
$C12_U$	12-bit constant (Unsigned) 0 to 4095	$C12_U$,Rs1,Rd,CND	$C16_U$,Rs1,Rd	$C32_U$,Rs1,Rd	$C16_S$,Rs1,Rd	$C17_S$,Rs1	Rs0,Rs1	Rs,Rd,CND	Rs0,Rs1,Rd,CND	Cond. Code Updated
$C16_U$	16-bit constant (Unsigned), 0 to 65535									
$C32_U$	32-bit constant (Unsigned), 0x00000000 to 0xFFFF0000, the 16 LSBs are always zero									
$C16_S$	16-bit constant (Signed), -32768 to 32767									
$C17_S$	17-bit constant (Signed), -65536 to 65535									
Rs,Rs0,Rs1	register Rn, used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand									
Rd	register Rn, used as destination operand									
CND	condition, the instruction is executed only if the condition is true, 15 conditions									
Mnemo	Description	Addressing Modes								
addt	add to	*	*						*	*
addc	add with carry, destination = source0 + source1 + CC.C	*							*	*
addh	add to high word			*						
subf	subtract from, source0 from source1	*	*						*	*
subc	subtract with carry, destination = source1 - source0 - CC.C	*							*	*
comp	compare (subtract source 0 from source 1)					*	*			*
cmpc	compare with carry (subtract source 0 + CC.C from source 1)					*	*			*
negt	negate							*		
absl	absolute value							*		
clzr	count leading zeros							*		
mult	multiply unsigned, 32*32 -> 64, store low word of result in destination								*	
mlhu	multiply high unsigned, 32*32 -> 64, store high word of result in destination								*	
mlhs	multiply high signed, 32*32 -> 64, store high word of result in destination								*	
mlcu	multiply constant, 16*32 -> 48, store lower 32 bits of result in destination		*							
mlcs	multiply constant, 16*32 -> 48, store lower 32 bits of result in destination				*					

Base ISA Miscellaneous Instructions

Operand Options													
Implied	no operands or operands are implicitly defined												
$BTI5_U$	5-bit bit index (Unsigned), 0 to 31												
$SHC5_U$	5-bit shift count (Unsigned), 0 to 31												
$C16_U$	16-bit constant (Unsigned), 0 to 65535												
$C17_S$	17-bit constant (Signed), -65536 to 65535												
Rs,Rs0,Rs1	register Rn, used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand												
Rd	register Rn, used as destination operand												
CND	condition, the instruction is executed only if the condition is true, 15 conditions												
Mnemo	Description	$BTI5_U$,Rs,CND	$BTI5_U$,Rs,Rd,CND	$SHC5_U$,Rs,Rd,CND	$C17_S$,Rd,CND	$C16_U$,Rs1,Rd	Rs0,Rs1,CND	Rs,Rd,CND	Rs0,Rs1,Rd,CND	Rd	Rs	Condition Code Updated
	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, zero-extend source operands to 32 bits				*			*				
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	*					*					*
	Endianess											
ibos	invert byte order short, swaps bytes 0 and 1, bits[31:16] remain unchanged							*				
ibol	invert byte order long, changes byte order from 3:2:1:0 to 0:1:2:3							*				

DSP Extension ISA Registers

Register Bits						31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rn	An	AHn	Dn	Name		Physical Registers Rn																																	
0				R0	8 x 32-bit registers for data operands	R0																																	
1				R1		R1																																	
2				R2		R2																																	
3				R3		R3																																	
4			4	R4		R4																																	
5			5	R5		R5																																	
6			6	R6		R6																																	
7			7	R7		R7																																	
8	8	0		R8	16 x 32-bit registers for data operands or indirect addresses	R8																																	
9	9	1		R9		R9																																	
10	10	2		RA		RA																																	
11	11	3		RB		RB																																	
12	12	4		RC		RC																																	
13	13	5		RD		RD																																	
14	14	6		RE		RE																																	
15	15	7		RF		RF																																	
16	0		0	RP		RP																																	
17	1		1	RQ		RQ																																	
18	2		2	RU		RU																																	
19	3		3	RV		RV																																	
20	4			RW		RW																																	
21	5			RX		RX																																	
22	6			RY		RY																																	
23	7			RZ		RZ																																	
24				LC	Loop Counter	LC																																	
25				CC	Condition Codes																														N	Z	O	C
26				CS	Control & Status	IVTP																														IS	IE	IR
27				EP	Extension ISA Parameters																					LU	ADRN1			ADRN0			RND					
28				TA	Target Address	TA																																	
29				SA	Subroutine (return) Address	SA																																	
30				IA	Interrupt (return) Address	IA																																	
31				ID	Core ID																					REV			IMA			ISA			TYP = 4		

Register Fields		
CC	C	Carry flag
	O	Overflow flag
	Z	Zero flag
	N	Negative flag
CS	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
	IVTP	Interrupt Vectors Table Pointer, defines the 26 MSBs [31:6] of the interrupt vector table start address in the data address space
EP	RND	Round, 0: no round, 1-19: adds 1 << (28+RND) to the product of mlnsd, mlhsd, msnsd, mshsd, m2nsd, m2hsd instructions
	ADRN0	Accumulation Extension Cache entry 0 Destination Register Number
	ADRN1	Accumulation Extension Cache entry 1 Destination Register Number
	LU	Accumulation Extension Cache Least Recently Used, 0: entry 0, 1: entry 1
ID	FML	Core Family, 1: eco16, 2: eco32, 4: sf32, 5: sf16
	ISA	ISA: 1: b = base, 2: d = dsp extension
	IMA	Implementation Architeture, 1: l = light, 4: u = ultralight
	REV	Revision, starting with 1

DSP Extension ISA instructions

Operand Symbols		Operand Addressing														
$DO12_S$	12-bit data address offset, signed, -2048 to 2047															
$AU12_S$	12-bit address update, signed, -2048 to 2047															
Rs,Rs0,Rs1	register Rn used as source (Rs), source 0 (Rs0) or source 1 (Rs1) operand															
Rd	register Rn used as destination operand															
Dd,Db	register Dn (R4-R7,RP-RV), used as destination (Dd) or both source and destination (Db) operand															
Es,Ed	register En (E0, E1, E2 or E3), used as source (Es) or destination (Ed) operand															
Rx	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA), used as index operand															
Ru	register Rn (R0-RZ,LC,CC,CS,ID,TA,SA,IA), used as update operand															
AHn	registers AHn (R8-RF) used as indirect address															
CND	condition, the instruction is executed only if the condition is true, 15 conditions	Rs	Rs,Ed	Es,Rd	Rs,Rd,CND	Rs0,Rs1,Rd,CND	Rs0,Rs1,Dd	($DO12_S$,An),Rs1,Dd	(An,$AU12_S$)*,Rs1,Dd	(An,Ru)*,Rs1,Dd	(Rx,An),Rs1,Dd	Rs0,Rs1,Db	($DO12_S$,An),Rs1,Db	(An,$AU12_S$)*,Rs1,Db	(An,Ru)*,Rs1,Db	(Rx,An),Rs1,Db
Mnemo	Description	Addressing Modes														
subsd	subtract with 1-bit left-shift of source operand 1					*										
addsd	add with 1-bit left-shift of source operand 1					*										
clssd	clip to signed short (low boundary 0x8000, high boundary 0x7FFF)				*											
clsmd	clip to signed maximum (low boundary 0xC0000000, high boundary 0x3FFFFFFF)				*											
clubd	clip to unsigned byte (low boundary 0, high boundary 0xFF)				*											
idivd	iterative divide, RX/RY -> E0 (quotient), E1 (remainder), RS = iteration count	*														
mtexd	move to extension register		*													
mfexd	move from extension register			*												
subfd	subtract from							*	*	*	*					
addtd	add to							*	*	*	*					
mlnsa	multiply & negate signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
mlhsa	multiply signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
macsa	multiply signed & accumulate, 32*32 -> 64, add high word of result to destination											*	*	*	*	*
macsa	multiply signed & subtract, 32*32 -> 64, subtract high word of result from destination											*	*	*	*	*
msnsa	multiply, left-shift & negate signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
mshsa	multiply & left-shift signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
msasa	multiply, left-shift signed & accumulate, 32*32 -> 64, add high word of result to destination											*	*	*	*	*
msssa	multiply, left-shift signed & subtract, 32*32 -> 64, subtract high word of result from destination											*	*	*	*	*
m2nsa	multiply, 2-bit left-shift & negate signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
m2hsa	multiply & 2-bit left-shift signed, 32*32 -> 64, store high word of result in destination						*	*	*	*	*					
m2asa	multiply, 2-bit left-shift signed & accumulate, 32*32 -> 64, add high word of result to destination											*	*	*	*	*
m2ssa	multiply, 2-bit left-shift signed & subtract, 32*32 -> 64, subtract high word of result from destination											*	*	*	*	*