ADSP 2106x

Rechenwerke
Auszüge aus
ADSP-2106x Sharc Users Manual
Analog Devices, Inc.

Überblick

ADSP-2106x with Super Harvard Architecture :

- 32-Bit IEEE Floating-Point Computation Units—Multiplier, ALU, and Shifter
- Data Register File
- Data Address Generators (DAG1, DAG2)
- Program Sequencer with Instruction Cache
- Interval Timer
- Dual-Ported SRAM
- External Port for Interfacing to Off-Chip Memory & Peripherals
- Host Port & Multiprocessor Interface
- DMA Controller, Serial Ports,Link Ports
- JTAG Test Access Port

Überblick

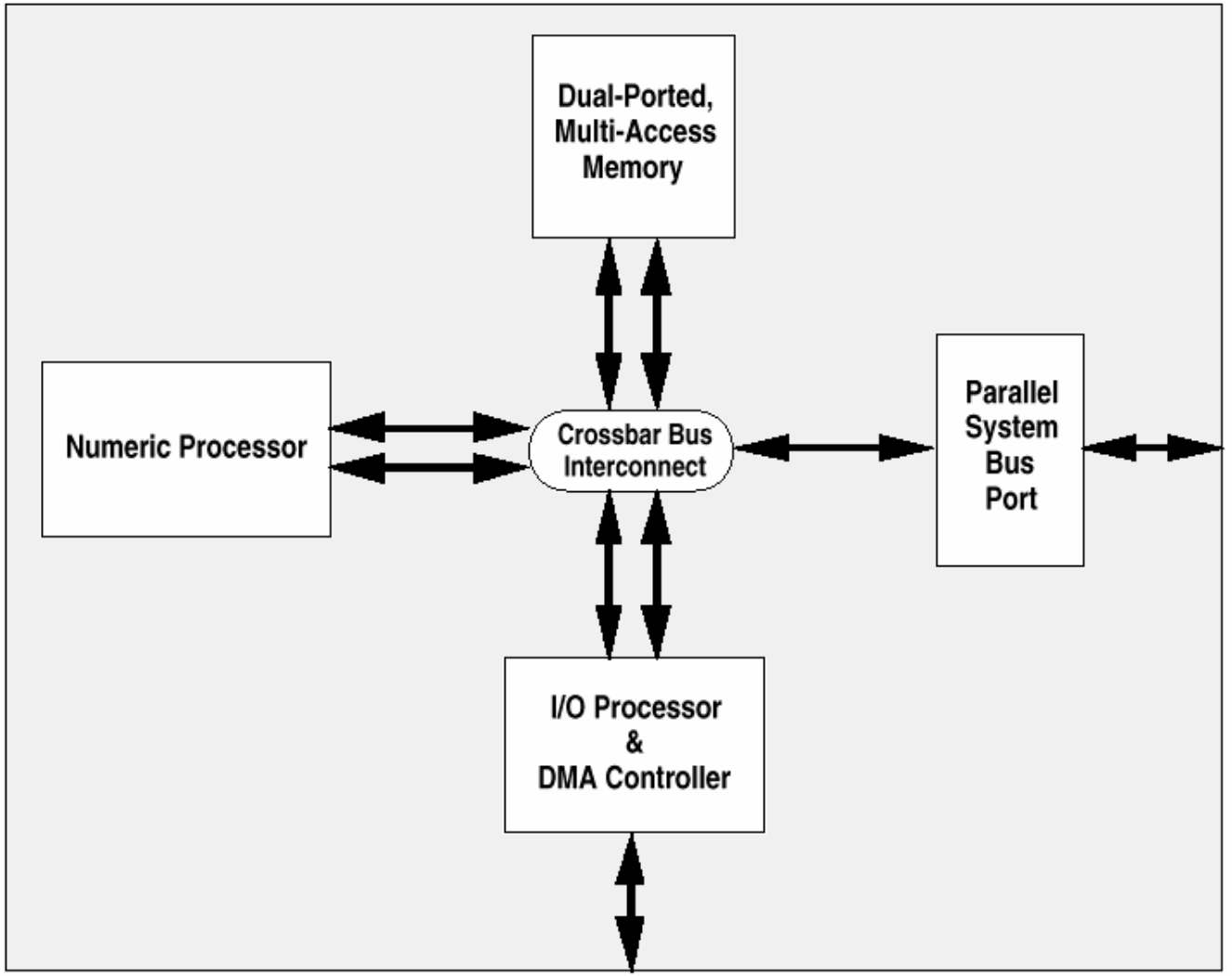
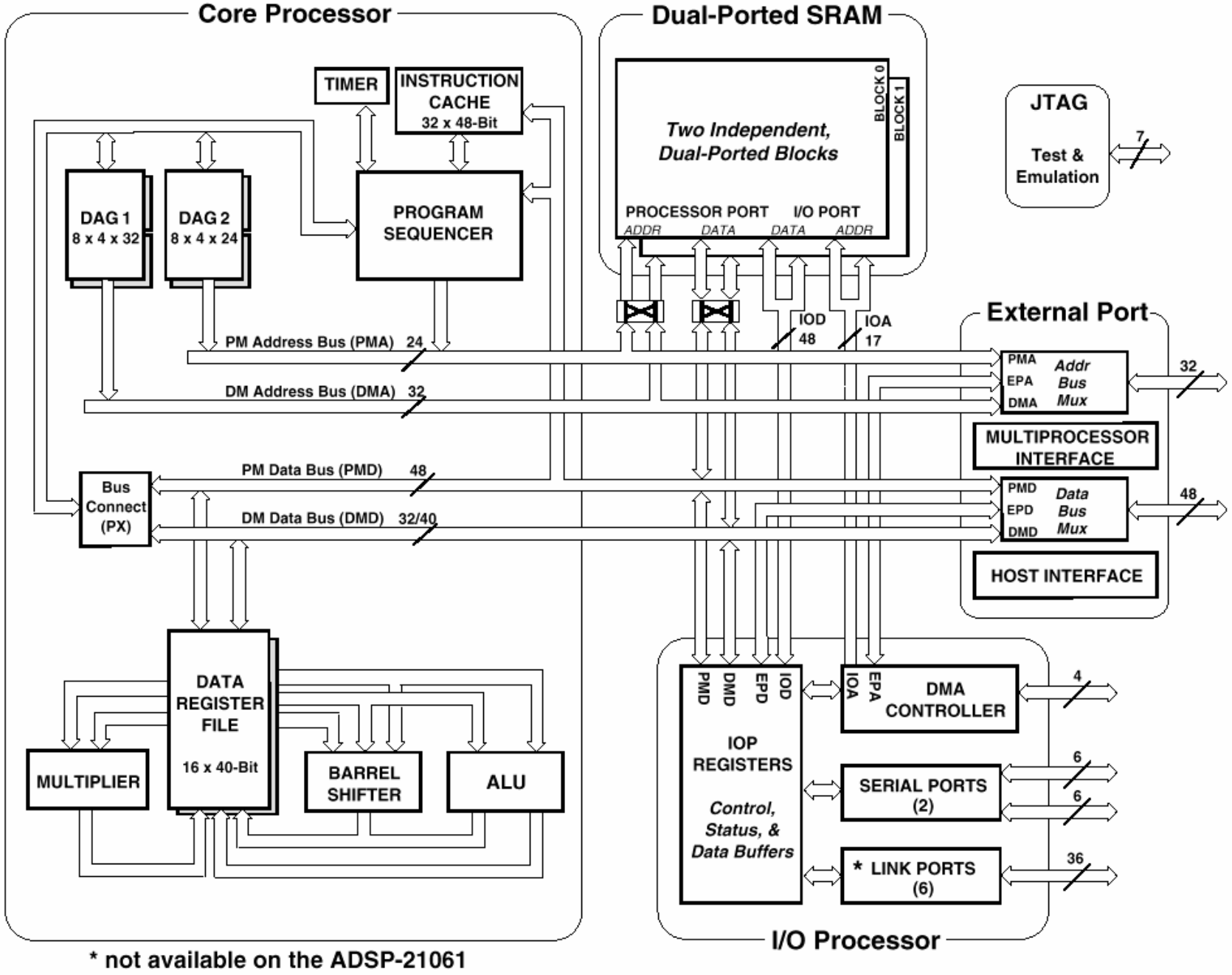
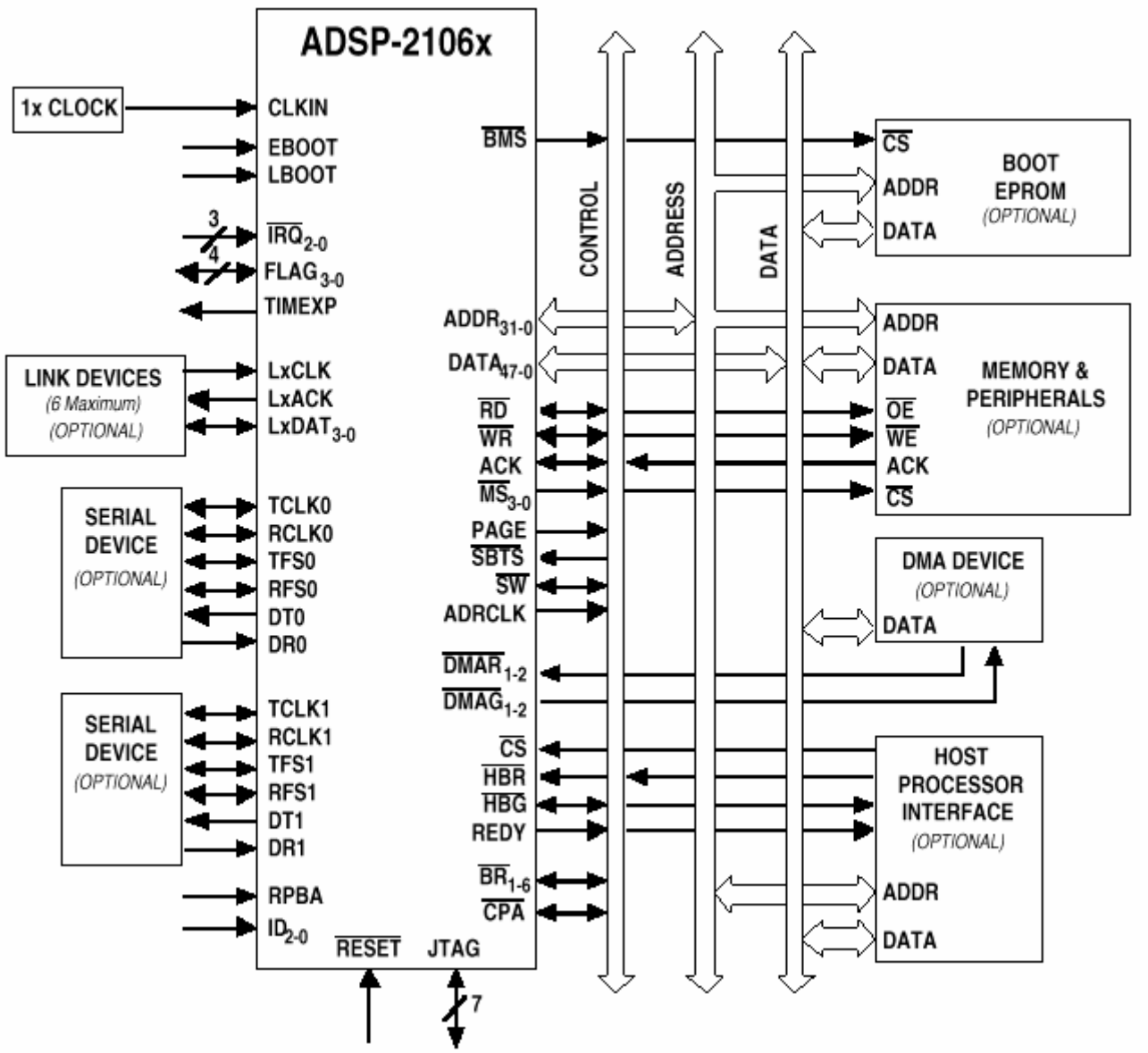
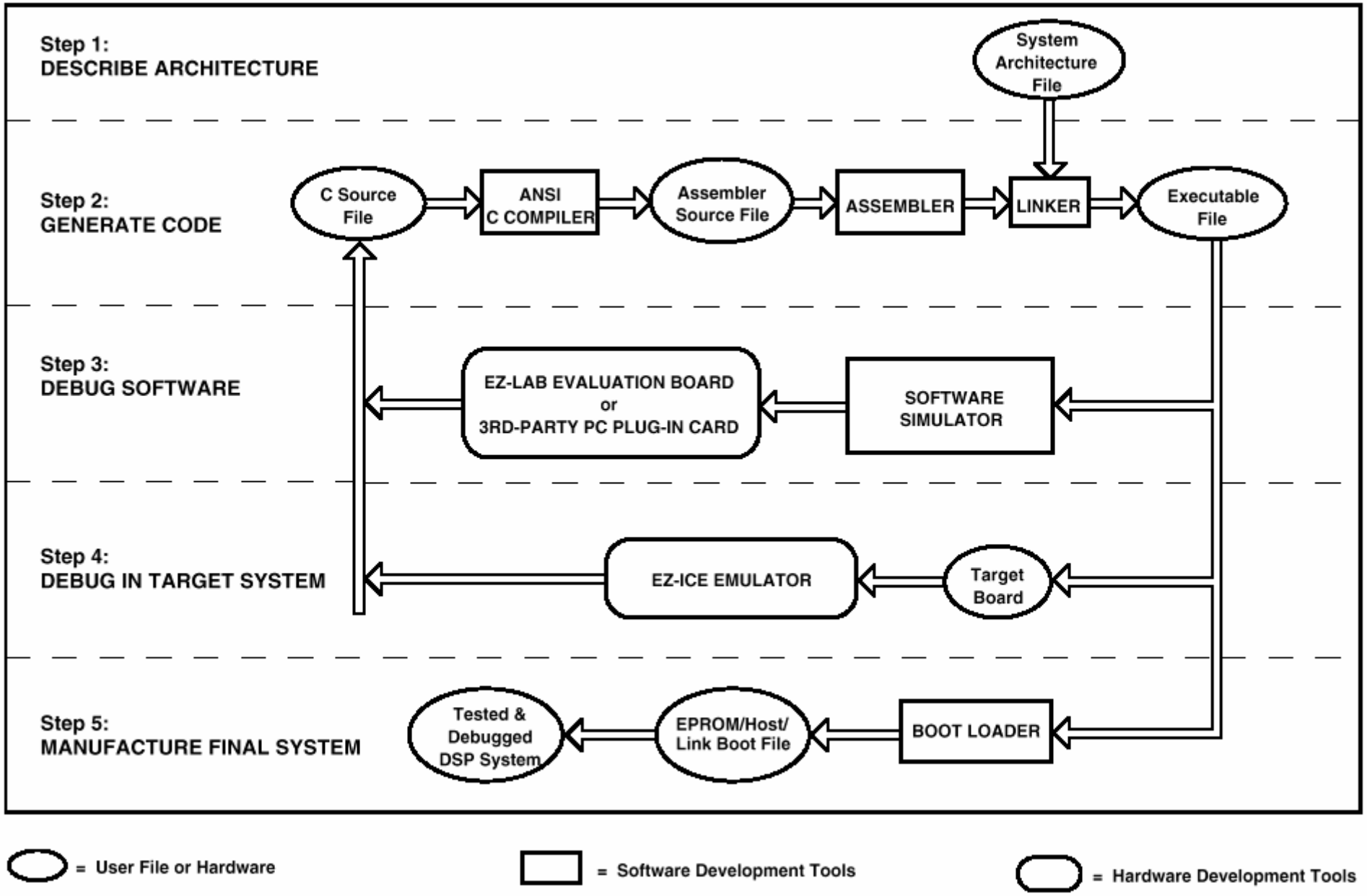


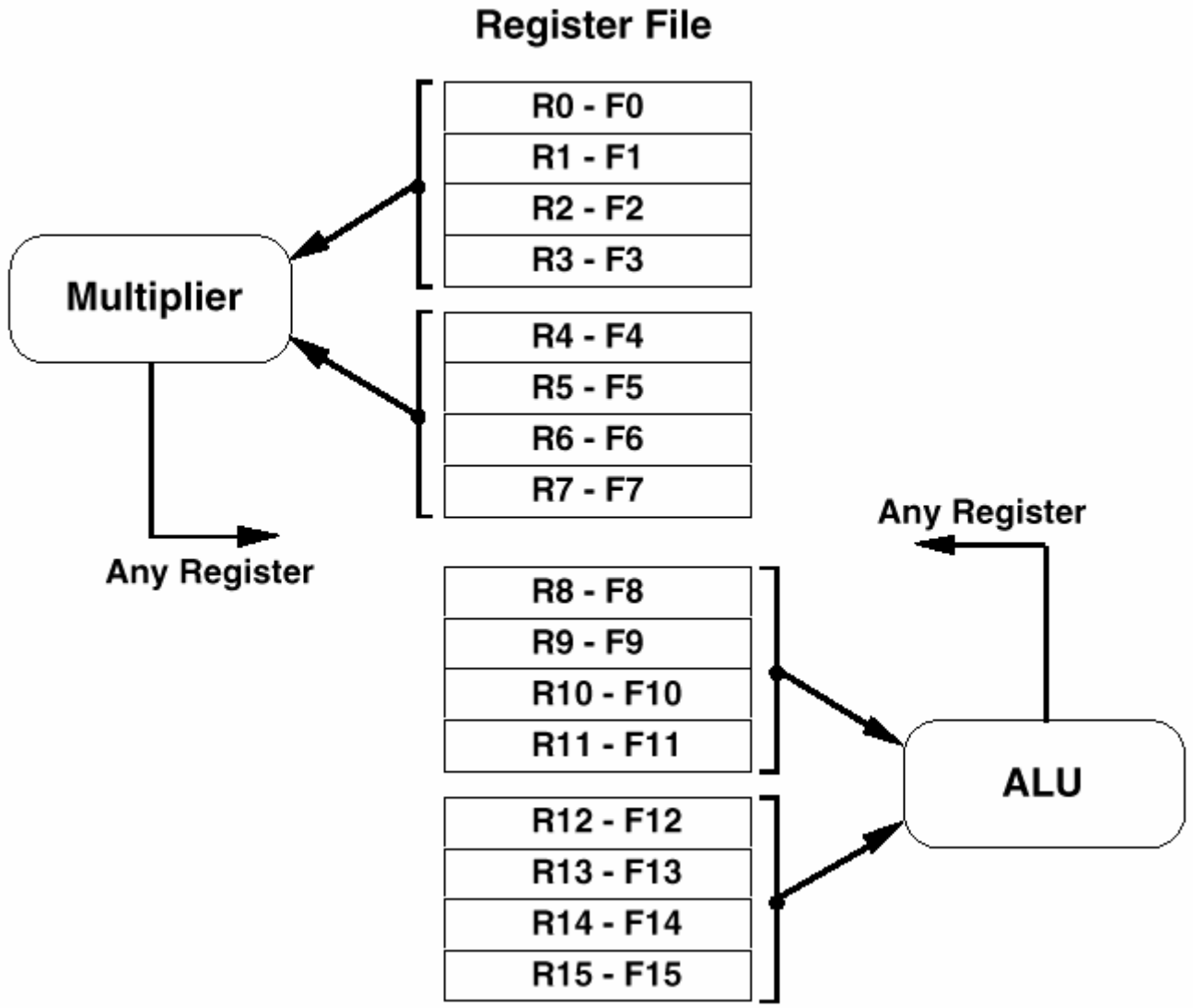
Figure 1.1 Super Harvard Architecture


Core Processor
TIMER
INSTRUCTION CACHE
32 x 48-Bit
DAG 1
8 x 4 x 32
DAG 2
8 x 4 x 24
PROGRAM SEQUENCER
Dual-Ported SRAM
Two Independent, Dual-Ported Blocks
BLOCK 0
BLOCK 1
PROCESSOR PORT
I/O PORT
ADDR
DATA
DATA
ADDR
JTAG
Test & Emulation
7
IOD
48
IOA
17
External Port
PM Address Bus (PMA) 24
DM Address Bus (DMA) 32
PMA
EPA
DMA
Addr Bus Mux
32
MULTIPROCESSOR INTERFACE
PM Data Bus (PMD) 48
DM Data Bus (DMD) 32/40
Bus Connect (PX)
PMD
EPD
DMD
Data Bus Mux
48
HOST INTERFACE
DATA REGISTER FILE
16 x 40-Bit
MULTIPLIER
BARREL SHIFTER
ALU
PMD
DMD
EPD
IOD
IOA
EPA
DMA CONTROLLER
4
IOP REGISTERS
Control, Status, & Data Buffers
SERIAL PORTS (2)
6
6
* LINK PORTS (6)
36
I/O Processor
* not available on the ADSP-21061

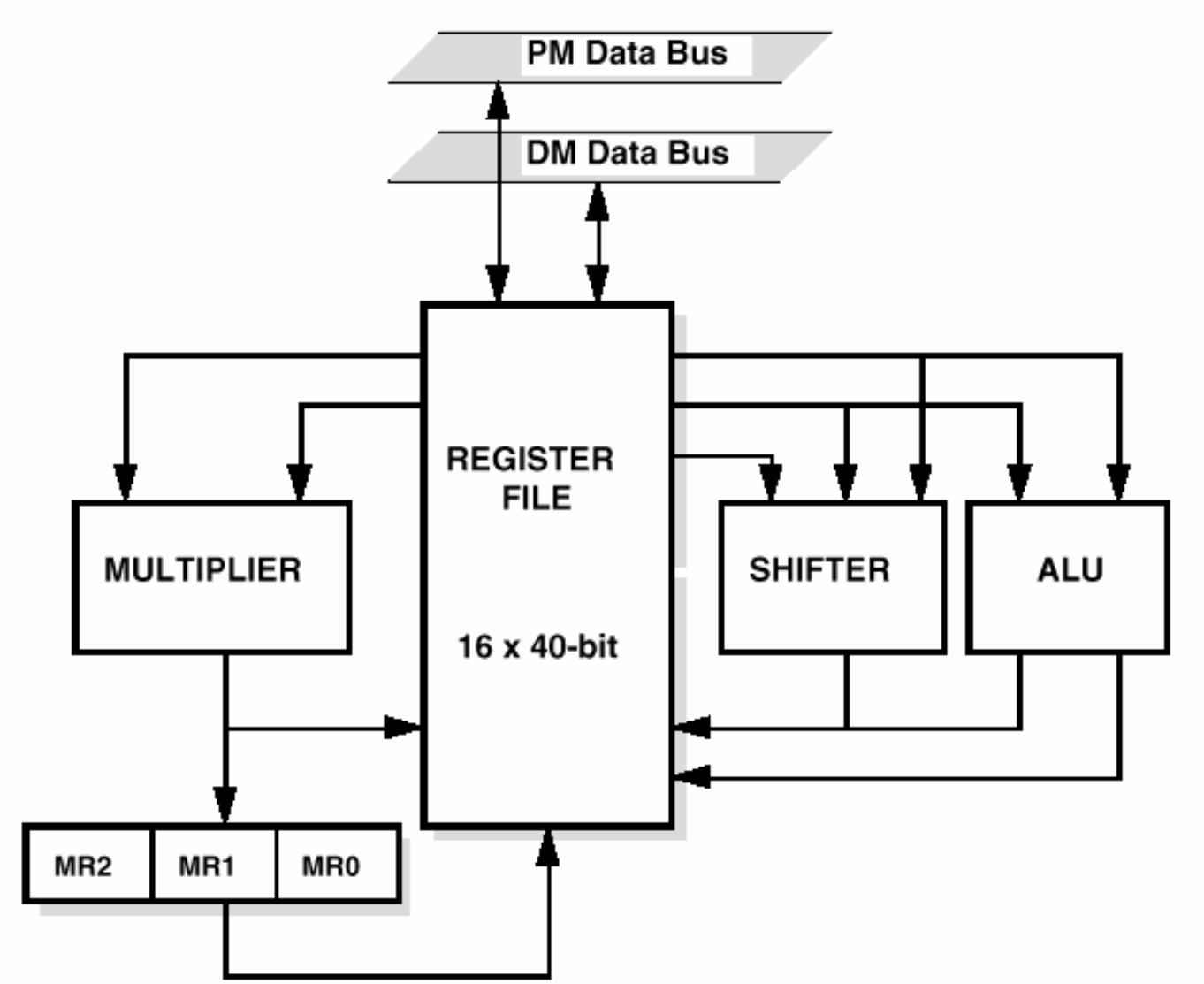

ADSP-2106x
1x CLOCK
CLKIN
EBOOT
LBOOT
3
IRQ2-0
4
FLAG3-0
TIMEXP
LINK DEVICES
(6 Maximum)
(OPTIONAL)
LxCLK
LxACK
LxDAT3-0
SERIAL DEVICE
(OPTIONAL)
TCLK0
RCLK0
TFS0
RFS0
DT0
DR0
SERIAL DEVICE
(OPTIONAL)
TCLK1
RCLK1
TFS1
RFS1
DT1
DR1
RPBA
ID2-0
RESET
JTAG
7
BMS
ADDR31-0
DATA47-0
RD
WR
ACK
MS3-0
PAGE
SBTS
SW
ADRCLK
DMAR1-2
DMAG1-2
CS
HBR
HBG
REDY
BR1-6
CPA
CONTROL
ADDRESS
DATA
CS
ADDR
DATA
BOOT EPROM
(OPTIONAL)
ADDR
DATA
OE
WE
ACK
CS
MEMORY & PERIPHERALS
(OPTIONAL)
DMA DEVICE
(OPTIONAL)
DATA
HOST PROCESSOR INTERFACE
(OPTIONAL)
ADDR
DATA


Step 1:
DESCRIBE ARCHITECTURE
System Architecture File
Step 2:
GENERATE CODE
C Source File
ANSI C COMPILER
Assembler Source File
ASSEMBLER
LINKER
Executable File
Step 3:
DEBUG SOFTWARE
EZ-LAB EVALUATION BOARD or 3RD-PARTY PC PLUG-IN CARD
SOFTWARE SIMULATOR
Step 4:
DEBUG IN TARGET SYSTEM
EZ-ICE EMULATOR
Target Board
Step 5:
MANUFACTURE FINAL SYSTEM
Tested & Debugged DSP System
EPROM/Host/ Link Boot File
BOOT LOADER
= User File or Hardware
= Software Development Tools
= Hardware Development Tools

Rechenwerke



Rechenwerke



Rechenwerke

The ADSP-2106x is

IEEE 754/854 compatible for single-precision floating-point operations

Rechenwerke - ALU

- **ALU**
- The ALU performs arithmetic operations on fixed-point or floating-point data
- and logical operations on fixed-point data.
- ALU fixed-point instructions operate on 32-bit fixed-point operands and output 32-bit fixed-point results.
- ALU floating-point instructions operate on 32-bit or 40-bit floating-point operands and output 32-bit or 40-bit floating-point results.

Rechenwerke - ALU

ALU instructions include:

- Floating-point addition, subtraction, add/subtract, average
- Fixed-point addition, subtraction, add/subtract, average
- Floating-point manipulation: binary log, scale, mantissa
- Fixed-point add with carry, subtract with borrow, increment, decrement
- Logical AND, OR, XOR, NOT
- Functions: Absolute value, pass, min, max, clip, compare
- Format conversion
- Reciprocal and reciprocal square root primitives

Rechenwerke - ALU

ALU instructions include:

- Floating-point addition, subtraction, add/subtract, average
- Fixed-point addition, subtraction, add/subtract, average
- Floating-point manipulation: binary log, scale, mantissa
- Fixed-point add with carry, subtract with borrow, increment, decrement
- Logical AND, OR, XOR, NOT
- Functions: Absolute value, pass, min, max, clip, compare
- Format conversion
- Reciprocal and reciprocal square root primitives

Rechenwerke - ALU

ALU Operating Modes

The ALU is affected by three bits in the MODE1 register:

- the ALU saturation bit affects ALU operations that yield fixed-point results
- the rounding mode and rounding boundary bits affect floating-point operations in both the ALU and multiplier.

Rechenwerke - ALU

ALU Operating Modes

MODE1 Register

Bit	Name	Function
13	ALUSAT	1=Enable ALU saturation (full scale in fixed-point)
		0=Disable ALU saturation
15	TRUNC	1=Truncation; 0=Round to nearest
16	RND32	1=Round to 32 bits;
		0=Round to 40 bits

Rechenwerke - ALU

Saturation mode

- **all positive fixed-point overflows cause the maximum positive fixed-point number (0x7FFFÊFFFF) to be returned, and all negative overflows cause the maximum negative number (0x8000Ê0000) to be returned.**
- **If the ALUSAT bit is set, fixed-point results that overflow are saturated.**
- **If the ALUSAT bit is cleared, fixed-point results that overflow are not saturated; the upper 32 bits of the result are returned unaltered.**
- **The ALU overflow flag reflects the ALU result before saturation.**

Rechenwerke - ALU

ALU Status Flags in ASTAT Register

Bit	Name	Definition
0	AZ	ALU result zero or floating-point underflow
1	AV	ALU overflow
2	AN	ALU result negative
3	AC	ALU fixed-point carry
4	AS	ALU X input sign (ABS, MANT operations)
5	AI	ALU floating-point invalid operation
10	AF	Last ALU operation was a floating-point operation

Rechenwerke - ALU

ALU also updates four “sticky” status flags in the STKY register.

- **Once set, a sticky flag remains high until explicitly cleared.**

Rechenwerke - ALU

STKY Register

Bit	Name	Definition
0	AUS	ALU floating-point underflow
1	AVS	ALU floating-point overflow
2	AOS	ALU fixed-point overflow
5	AIS	ALU floating-point invalid operation

Rechenwerke - ALU

Rn, Rx, Ry = Any register file location; treated as fixed-point

Fn, Fx, Fy = Any register file location; treated as floating-point

c = ADSP-21xx-compatible instruction

*** set or cleared, depending on results of instruction**

**** may be set (but not cleared), depending on results of instruction**

– no effect

ALU Instruction Summary

	Instruction	*ASTAT Status Flags*								*STKY Status Flags*			
	Fixed-point:	AZ	AV	AN	AC	AS	AI	AF	CACC	AUS	AVS	AOS	AIS
c	Rn = Rx + Ry	*	*	*	*	0	0	0	–	–	–	**	–
c	Rn = Rx – Ry	*	*	*	*	0	0	0	–	–	–	**	–
c	Rn = Rx + Ry + CI	*	*	*	*	0	0	0	–	–	–	**	–
c	Rn = Rx – Ry + CI – 1	*	*	*	*	0	0	0	–	–	–	**	–
	Rn = (Rx + Ry)/2	*	0	*	*	0	0	0	–	–	–	–	–
	COMP(Rx, Ry)	*	0	*	0	0	0	0	*	–	–	–	–
	Rn = Rx + CI	*	*	*	*	0	0	0	–	–	–	**	–
	Rn = Rx + CI – 1	*	*	*	*	0	0	0	–	–	–	**	–
	Rn = Rx + 1	*	*	*	*	0	0	0	–	–	–	**	–
	Rn = Rx – 1	*	*	*	*	0	0	0	–	–	–	**	–
c	Rn = –Rx	*	*	*	*	0	0	0	–	–	–	**	–
c	Rn = ABS Rx	*	*	0	0	*	0	0	–	–	–	**	–
	Rn = PASS Rx	*	0	*	0	0	0	0	–	–	–	–	–
c	Rn = Rx AND Ry	*	0	*	0	0	0	0	–	–	–	–	–
c	Rn = Rx OR Ry	*	0	*	0	0	0	0	–	–	–	–	–
c	Rn = Rx XOR Ry	*	0	*	0	0	0	0	–	–	–	–	–
c	Rn = NOT Rx	*	0	*	0	0	0	0	–	–	–	–	–
	Rn = MIN(Rx, Ry)	*	0	*	0	0	0	0	–	–	–	–	–
	Rn = MAX(Rx, Ry)	*	0	*	0	0	0	0	–	–	–	–	–
	Rn = CLIP Rx BY Ry	*	0	*	0	0	0	0	–	–	–	–	–

ALU Instruction Summary

Floating–point:													
Fn = Fx + Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn = Fx – Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn = ABS (Fx + Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn = ABS (Fx – Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn = (Fx + Fy)/2	*	0	*	0	0	*	1	–	**	–	–	**
COMP(Fx, Fy)	*	0	*	0	0	*	1	*	–	–	–	**
Fn = –Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = ABS Fx	*	*	0	0	*	*	1	–	–	**	–	**
Fn = PASS Fx	*	0	*	0	0	*	1	–	–	–	–	**
Fn = RND Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = SCALB Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Rn = MANT Fx	*	*	0	0	*	*	1	–	–	**	–	**
Rn = LOGB Fx	*	*	*	0	0	*	1	–	–	**	–	**
Rn = FIX Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Rn = FIX Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn = FLOAT Rx BY Ry	*	*	*	0	0	0	1	–	**	**	–	–
Fn = FLOAT Rx	*	0	*	0	0	0	1	–	–	–	–	–
Fn = RECIPS Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn = RSQRTS Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = Fx COPYSIGN Fy	*	0	*	0	0	*	1	–	–	–	–	**
Fn = MIN(Fx, Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn = MAX(Fx, Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn = CLIP Fx BY Fy	*	0	*	0	0	*	1	–	–	–	–	**

Rechenwerke - Multiplier

The multiplier performs

- **fixed-point or floating-point multiplication**
- **fixed-point multiply/accumulate operations.**
- **Floating-point multiply/accumulates can be accomplished through parallel operation of the ALU and multiplier**

Rechenwerke - Multiplier

- **Multiplier floating-point instructions operate on 32-bit or 40-bit floating-point operands and output 32-bit or 40-bit floating-point results.**
- **Multiplier fixed-point instructions operate on 32-bit fixed-point data and produce 80-bit results. Inputs are treated as fractional or integer, unsigned or signed.**

Rechenwerke - Multiplier

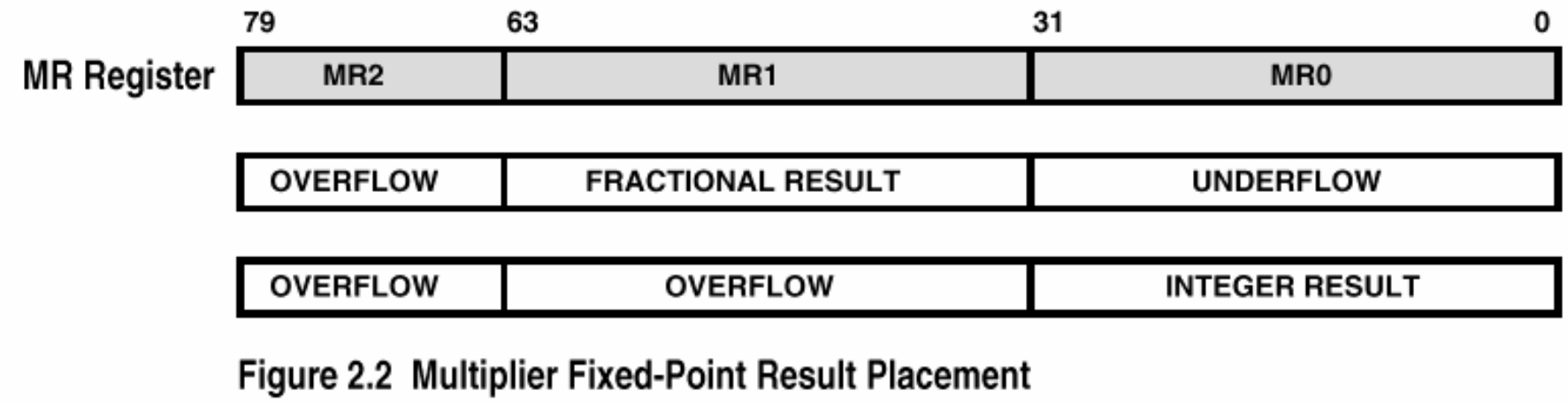


Figure 2.2 Multiplier Fixed-Point Result Placement

Rechenwerke - Multiplier

- **The multiplier updates four status flags at the end of each operation in Register ASTAT**

ASTAT

Bit	Name	Definition
6	MN	Multiplier result negative
7	MV	Multiplier overflow
8	MU	Multiplier underflow
9	MI	Multiplier floating-point invalid operation

Rechenwerke - Multiplier

The multiplier also updates four “sticky” status flags in the STKY register. Once set, a sticky flag remains high until explicitly cleared.

STKY

Bit	Name	Definition
6	MOS	Multiplier fixed-point overflow
7	MVS	Multiplier floating-point overflow
8	MUS	Multiplier underflow
9	MIS	Multiplier floating-point invalid operation

Instruction	*ASTAT Flags*				*STKY Flags*			
	MU	MN	MV	MI	MUS	MOS	MVS	MIS
Fixed-Point:								
Rn / MRF / MRB = Rx * Ry (S/U, S/U, F/I/FR)	*	*	*	0	–	**	–	–
Rn / MRF / MRB = MRF / MRB + Rx * Ry (S/U, S/U, F/I/FR)	*	*	*	0	–	**	–	–
Rn / MRF / MRB = MRF / MRB – Rx * Ry (S/U, S/U, F/I/FR)	*	*	*	0	–	**	–	–
Rn / MRF = SAT MRF / MRB (SI/UI/SF/UF)	*	*	*	0	–	**	–	–
Rn / MRF = RND MRF / MRB (SF/UF)	*	*	*	0	–	**	–	–
MRF / MRB = 0	0	0	0	0	–	–	–	–
MRxF / MRxB = Rn	0	0	0	0	–	–	–	–
Rn = MRxF / MRxB	0	0	0	0	–	–	–	–

Rechenwerke - Multiplier

Instruction	*ASTAT Flags*				*STKY Flags*			
	MU	MN	MV	MI	MUS	MOS	MVS	MIS
Floating-Point:								
Fn = Fx * Fy	*	*	*	*	**	_	**	**

Rechenwerke - Multiplier

Optional Modifiers for Fixed-Point:

(X-input ❑ | Y-input ❑ | Data format, rounding ❑)

S	Signed input
U	Unsigned input
I	Integer input(s)
F	Fractional input(s)
FR	Fractional inputs, Rounded output
(SF)	Default format for 1-input operations
(SSF)	Default format for 2-input operations

Rechenwerke Programmbeispiel

$$dmErgebnis = \sum_{k=1}^{5} pmVectori_k \cdot dmVectori_k$$

Rechenwerke - Multiplier

```
/*    Programm zur Demonstration der Festkommarechnung

      Programmentwurf:        H.W.W
      Datum:                        04.04.2008
      Tool:                         Visual DSP ++
*/

.SECTION /dm seg_dmda;
.VAR dmVectori[5]=10,-100,1000,-10000,100000;
.VAR dmErgebnis;

.SECTION /pm seg_pmda;
.VAR pmVectori[5]=2,-2,3,-3,4;
```

Rechenwerke - Multiplier

```
.SECTION /pm seg_rth;
        nop;
.align 0x4;
        nop;
        jump start;
```

Rechenwerke - Multiplier

```
/*  1.Adressierung der Vektopren durch
    die Adressgeneratoren    */
start:      B0=dmVectori;
            M0=1;
            L0=@dmVectori;
            B8=pmVectori;
            M8=1;
            L8=@pmVectori;
```

Rechenwerke - Multiplier

```
/*   2. Berechnung der Summe von Produkten in
     einer Schleife getrennte Multiplikation und
     Addition       */
                R9=0;        /*  Ergebnisregister    */
loopa1:         LCNTR=L0, DO loop1e UNTIL LCE;
                R0=DM(I0,M0), R4=PM(I8,M8);
                R8=R0*R4 (SSI);
loop1e:         R9=R9+R8;
                DM(dmErgebnis)=R9;
```

Rechenwerke - Multiplier

```
/*   3.Berechnung der Summe von Produkten in
     einer Schleife mit Multiplikation und
     Additionin einem Befehl                    */
                MRF=0;
loopa2:         LCNTR=L0, DO loop2e UNTIL LCE;
                R0=DM(I0,M0), R4=PM(I8,M8);
loop2e:         MRF=MRF+R0*R4 (SSI);
                R9=MR0F;
                DM(dmErgebnis)=R9;
```

Rechenwerke - Multiplier

```
/*     4.Berechnung der Summe von Produkten in einer Schleife
           mit Multiplikation und Addition in einem Befehl und
           parallelem Laden der Operanden        */
           MRF=0;
           R0=DM(I0,M0), R4=PM(I8,M8);
loopa3: LCNTR=L0, DO loop3e UNTIL LCE;
loop3e:MRF=MRF+R0*R4 (SSI),R0=DM(I0,M0), R4=PM(I8,M8);
           R9=MR0F;
           DM(dmErgebnis)=R9;
           idle;
```