6 Hardware Multiplier

The hardware multiplier is realized as each other 16 bit peripheral module, and not integrated into the CPU. The CPU is unchanged through all configurations, and the instruction set is not modified. It take no extra cycle for multiplication. Both operands are loaded into the multiplier's register and the result can be accessed immediately after loading the second operand.

Topic **Page**

6.1 Hardware Multiplier Operation 6-4
6.2 Hardware Multiplier Registers 6-9
6.3 Hardware Multiplier Special Function bits 6-10
6.4 Hardware Multiplier Software Restrictions 6-10

The Hardware Multiplier Module expands the capabilities of the MSP430 family without changing the basic architecture. Multiplication is possible for:

- 16 x 16 bit
- 16 x 8 bit
- 8 x 16 bit
- 8 x 8 bit

The hardware multiplier module supports three types of multiplication: unsigned multiplication (MPY), signed multiplication (MPYS) and unsigned multiplication and accumulation (MAC).

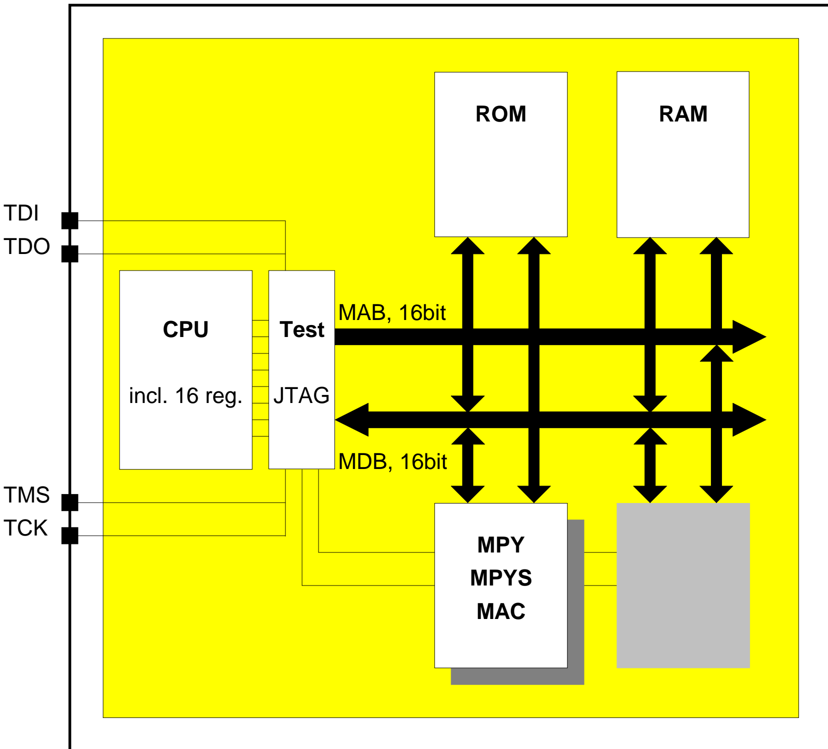

Figure 6.1: Connection of the Hardware Multiplier Module to the Bus System

6.1 Hardware Multiplier Operation

The hardware multiplier has two 16-bit registers for both operands, and three registers where the result of the multiplication is stored. The multiplication is executed correctly when the operand OP1 is written prior of the second operand OP2 to the operands' registers. The type of multiplication is selected when the first operand is written to the appropriate register. Writing the second operand to the appropriate register starts the multiplication. It is completed before the result registers are accessed using indexed address mode for the source operand. Another instruction is needed between the write of the second operand and the access to result registers, when indirect or indirect autoincrement address mode is used. Both operands - transferred to the hardware multiplier - have all seven address mode capabilities.
No instruction for the multiplication is added, which means that the real-time operation and the interrupt latency is unchanged.

Multiply unsigned, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
*******************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE     *
*      MULTIPLIER MODULE                                           *
*    USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA     *
*******************************************************************
OPERAND1  .EQU     0                ; 0:  OPERAND1 IS WORD (16BIT)
                                    ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU     0                ; 0:  OPERAND2 IS WORD (16BIT)
                                    ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU     0130H
MPYS      .EQU     0132H
MAC       .EQU     0134H
OP2       .EQU     0138H
RESLO     .EQU     013AH
RESHI     .EQU     013CH
SUMEXT    .EQU     013EH
          .BSS     OPER1,2,200H
          .BSS     OPER2,2
          .BSS     RAM,8

          .IF OPERAND1=8
          MOV.B    &OPER1,&MPY      ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
          .ELSE
          MOV      &OPER1,&MPY      ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
          .ENDIF

          .IF OPERAND1=8
          MOV.B    &OPER2,&OP2      ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
          .ELSE
          MOV      &OPER2,&OP2      ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
          .ENDIF

*******************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*      TO THE RAM DATA, 64BITS                                     *
*******************************************************************
          ADD      &RESLO,&RAM      ; ADD LOW RESULT TO RAM
          ADDC     &RESHI,&RAM+2    ; ADD HIGH RESULT RO RAM+2
          ADC      &RAM+4           ; ADD CARRY TO EXTENSION WORD
          ADC      &RAM+6           ; IF 64 BIT LENGHT IS USED
```

32 Bytes of program code, 32 execution cycles (16x16bit multiplication)

6

6

Multiply signed, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
********************************************************************
*       TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE    *
*       MULTIPLIER MODULE                                          *
*       IF ONE OF THE OPERANDS IS 8BIT, SIGN EXTENSION IS NEEDED   *
*     USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA    *
********************************************************************
OPERAND1  .EQU      0                ; 0:  OPERAND1 IS WORD (16BIT)
                                     ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU      0                ; 0:  OPERAND2 IS WORD (16BIT)
                                     ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU      0130H
MPYS      .EQU      0132H
MAC       .EQU      0134H
OP2       .EQU      0138H
RESLO     .EQU      013AH
RESHI     .EQU      013CH
SUMEXT    .EQU      013EH
          .BSS      OPER1,2,200H
          .BSS      OPER2,2
          .BSS      RAM,8

          .IF OPERAND1=0
          MOV       &OPER1,&MPYS     ; LOAD 1ST (WORD) OPERAND
                                     ; DEFINES ADD. SIGNED MULTIPLY
          .ELSE
          MOV.B     &OPER1,&MPYS     ; LOAD 1ST (BYTE) OPERAND,
                                     ; DEFINES ADD. SIGNED MULTIPLY
          SXT       &MPYS            ; EXPAND BYTE TO SIGNED WORD DATA
          .ENDIF
          .IF OPERAND2=0
          MOV       &OPER2,&OP2      ; LOAD 2ND (WORD) OPERAND  AND
                                     ; START SIGNED MULTIPLICATION
          .ELSE
          MOV.B     &OPER2,&OP2      ; LOAD 2ND (BYTE) OPERAND,
          SXT       &OP2             ; RE-LOAD 2ND OPERAND  AND START
                                     ; SIGNED ‘FINAL’ MULTIPLICATION
          .ENDIF

********************************************************************
*       EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION   *
*       TO THE RAM DATA, 64BITS                                    *
********************************************************************
          ADD       &RESLO,&RAM      ; ADD LOW RESULT TO RAM
          ADDC      &RESHI,&RAM+2    ; ADD HIGH RESULT RO RAM+2
          ADDC      &SUMEXT,&RAM+4   ; ADD SIGN WORD TO EXTENSION WORD
          ADDC      &SUMEXT,&RAM+6   ; IF 64 BIT LENGHT IS USED
```

36 Bytes program code, 36 execution cycles (16x16bit multiplication)

Multiply unsigned and accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE     *
*      MULTIPLIER MODULE                                           *
*      THE RESULT OF THE MULTIPLICATION IS ADDED TO THE CONTENT    *
*      OF BOTH RESULT REGISTERS, RESLO AND RESHI                   *
*      USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA   *
********************************************************************
OPERAND1  .EQU     0                ; 0:  OPERAND1 IS WORD (16BIT)
                                    ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2  .EQU     0                ; 0:  OPERAND2 IS WORD (16BIT)
                                    ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY       .EQU     0130H
MPYS      .EQU     0132H
MAC       .EQU     0134H
OP2       .EQU     0138H
RESLO     .EQU     013AH
RESHI     .EQU     013CH
SUMEXT    .EQU     013EH
          .BSS     OPER1,2,200H
          .BSS     OPER2,2
          .BSS     RAM,8

          .IF OPERAND1=8
          MOV.B    &OPER1,&MAC      ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
          .ELSE
          MOV      &OPER1,&MAC      ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
          .ENDIF

          .IF OPERAND1=8
          MOV.B    &OPER2,&OP2      ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
          .ELSE
          MOV      &OPER2,&OP2      ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
          .ENDIF
```

```
********************************************************************
*       EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION    *
*       TO THE RAM DATA, 64BITS                                     *
*       THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO AND       *
*       RESHI REGISTERS. THE UPPER TWO WORDS IN THE EXAMPLE ARE     *
*       FURTHER LOCATED IN THEIR RAM LOCATION                       *
********************************************************************
          ADDC      &SUMEXT,&RAM+4  ; ADD SUMEXTENSION TO RAM+4
          ADC       &RAM+6          ; IF 64 BIT LENGHT IS USED
```

32 Bytes program code, 32 execution cycles (16x16bit multiplication)

6

6.2 Hardware Multiplier Registers

The hardware multiplier module hardware is word structured, but it can be accessed by word or byte processing instructions.

Register	short form	Register type	Address	Initial state
• Multiply Unsigned/Op.1	MPY	Type of read/write	0130h	unchanged
• Multiply Signed/Operand1	MPYS	Type of read/write	0132h	unchanged
• Multiply+Accumulate/Op.1	MAC	Type of read/write	0134h	unchanged
• Reserved			0136h	unchanged
• Second Operand	OP2	Type of read/write	0138h	unchanged
• Result Low Word	ResLo	Type of read/write	013Ah	undefined
• Result High Word	ResHi	Type of read/write	013Ch	undefined
• Sum Extend register	SumExt	Type of read	013Eh	undefined

There are two registers implemented for both operands, operand OP1 and OP2. The operand 1 use three different addresses to address the same register. The different address information is decoded and defines the type of multiplication - unsigned, signed and unsigned+accumulate.

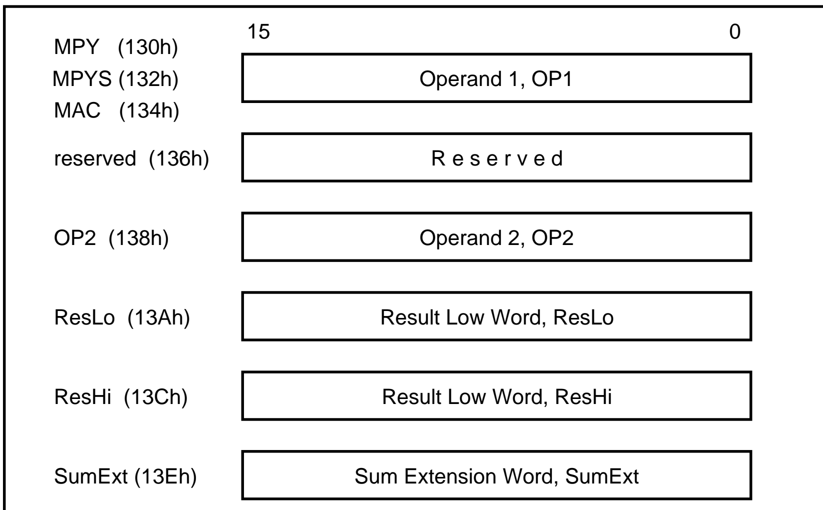

Figure 6.2: Registers of the Hardware Multiplier

The result is located in two word registers, the result high RESHI and result low RESLO register. The sum extend register SumExt holds the sign of the result of a signed 16x16bit multiplication, or holds the overflow of the multiply and accumulate (MAC) operation.
All registers have the LSB at bit0 and the MSB at bit7 (byte data) or bit15 (word data).

6.3 Hardware Multiplier Special Function bits

The hardware multiplier module completes all multiply operations fast without interrupt intervention, and therefore no special function bits are used.

6.4 Hardware Multiplier Software Restrictions

Two special cases need attention when the hardware multiplier is used:

- Use of indirect or indirect autoincrement address mode to process the result
- Use of the hardware multiplier in an interrupt routine

6.4.1 Hardware Multiplier Software Restrictions - Address mode

The access to the result of a multiplication works in indexed, indirect or indirect autoincrement mode. The access to the result registers can be done without any restrictions if indexed address mode is used - including symbolic and absolute address mode. Whenever the indirect and indirect autoincrement address mode is used to access the result registers, at least one instruction between the load of the second operand and access to one of the result registers is needed:

```
*******************************************************************
*    EXAMPLE: MULTIPLY OPERAND1 AND OPERAND 2                     *
*******************************************************************
RESLO     .SET      013AH          ; RESLO = ADDRESS OF RESLO
          PUSH      R5             ; R5 WILL HOLD THE ADDRESS OF
          MOV       #RESLO,R5      ; THE RESLO REGISTER

          MOV       &OPER1,&MPY    ; LOAD 1ST OPERAND,
                                   ; DEFINES ADD. UNSIGNED MULTIPLY
          MOV       &OPER2,&OP2    ; LOAD 2ND OPERAND AND START
                                   ; MULTIPLICATION

*******************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION   *
*      TO THE RAM DATA, 64BITS                                    *
*******************************************************************
          NOP                      ; MIN. ONE CYLES BETWEEN MOVING
                                   ; THE OPERAND2 TO HW-MULTIPIER
                                   ; AND PROCESSSING THE RESULT WITH
                                   ; INDIRECT ADDRESS MODE
          ADD       @R5+,&RAM      ; ADD LOW RESULT TO RAM
          ADDC      @R5,&RAM+2     ; ADD HIGH RESULT RO RAM+2
          ADC       &RAM+4         ; ADD CARRY TO EXTENSION WORD
          ADC       &RAM+6         ; IF 64 BIT LENGHT IS USED

          POP       R5
```

The example shows that the indirect or indirect address mode - used to transfer the result of a multiplication to the destination - needs more cycles and code than the absolute address mode. Obviously there is no special need to access the absolute hardware multiplier using indirect addressing mode.

6.4.2 Hardware Multiplier Software Restrictions - Interrupt Routines

The entire multiplication routine uses three major steps:

- move the operand OP1 to the hardware multiplier, the type of multiplication is defined
- move the operand OP2 to the hardware multiplier, the multiplication is started
- process the result of the multiplication in RESLO, RESHI SUMEXT registers

The following considerations are useful if the main routines use hardware multiplication. If no hardware multiplication is used in the main routines, the multiplication in an interrupt routine is protected from further interrupts, since the general interrupt enable bit is reset after entering the interrupt service routine. Normally a multiplication with the entire data processing should be done outside an interrupt routine following the rule: Keep interrupt routines as short as possible.

A multiplication in an interrupt routine has some feedback to the multiplication routine in the main routine:

Interrupt occurs after the first operand OP1 is transferred into hardware multiplier

The two LSBs of the first operand’s address defines the type of multiplication. This information can not be recovered by any later operation. The interrupt should not be able to be accepted between the first two steps - move operand OP1 and operand OP2 to the multiplier.

Interrupt occurs after the second operand OP2 is transferred into hardware multiplier

After the first two steps, the result is already in the corresponding registers RESLO, RESHI and SUMEXT and can be saved e.g. on the stack (*PUSH* ...) and restored after completing another multiplication (*POP* ...). But additional code and cycles in the interrupt routine are used. This can be avoided when the entire multiplication routine is protected by disabling any interrupt (*DINT*) before entering the multiplication routine and enabling interrupts (*EINT*) after the multiplication routine is completed. A negative impact on this method is that the critical interrupt latency is increased drastically for events which occur during this period.

General recommendation

In general a hardware multiplication within an interrupt routine should be avoided when a hardware multiplication is already used in main routines. The application specific

software, applied libraries or other included software should be taken into consideration. The different methods discussed show more negative implications than positive. Following the general recommendation to shorten interrupt routines is the best practice.