Chapter 12

Hardware Multiplier

The MSP430 hardware multiplier is a peripheral device and does not constitute part of the MSP430 CPU. It allows the multiplication of both signed and unsigned numbers to be carried out. The multiply and accumulate (MAC) operation is also supported, which is useful for implementing digital signal processing (DSP) tasks such as Finite Impulse Response (FIR) filters.

Topic **Page**

12.1 Introduction

The following devices in the MSP430 family contain a hardware multiplier peripheral module:

- 54xx;
- FG46xx;
- FE42x(A);
- F47xx;
- F44X;
- F42x(A);
- F261x;
- F24x(x);
- F16x(x).

The MSP430FG4618 device, as used on the Experimenter's board, supports multiply operations using the Hardware Multiplier, without affecting the CPU activities.

By writing operands to two registers (each one with 8 or 16 bits), the hardware multiplier supports:

- Unsigned multiply (MPY);
- Signed multiply (MPYS);
- Unsigned multiply and accumulate (MAC);
- Signed multiply and accumulate (MACS);
- Multiplications using 16×16 bits, 16×8 bits, 8×16 bits and 8×8 bits.

There are four different operand one registers (OP1), one for each multiplication type and a universal second operand two register (OP2).

The result of an operation can be accessed by reading two or three registers:

- Result low 16-bit word (bits 15 .. 0) in register RESLO;
- Result high 16-bit word (bits 31 .. 16) in register RESHI;
- When used MAC or MACS: bit 32 in register SUMEXT.

The result is available three MCLK cycles after the operands have been loaded into the peripheral registers.

TI have an Application Report (see Annex E) – The MSP430 Hardware Multiplier – Function and Applications <slaa042.pdf>, which contains detailed information concerning this peripheral module.

12.2 Hardware multiplier structure

The hardware multiplier structure contains (see *Figure 12-1*):

- ❑ Two 16-bit operand registers:
  - ■ The operand one register, OP1:
    - o Has four addresses, used to select the multiply mode (see *Table 12-1*);
  - ■ The operand two register, OP2:
    - o Writing to it initiates the multiply operation.
- ❑ Three result registers, RESLO, RESHI, and SUMEXT:
  - ■ RESLO stores the low word of the result;
  - ■ RESHI stores the high word of the result:
    - o The contents depend on the multiply operation (see *Table 12-2*).
  - ■ SUMEXT stores information about the result.
    - o The contents depend on the multiply operation (see *Table 12-3*).

Figure 12-1. Hardware multiplier block diagram.

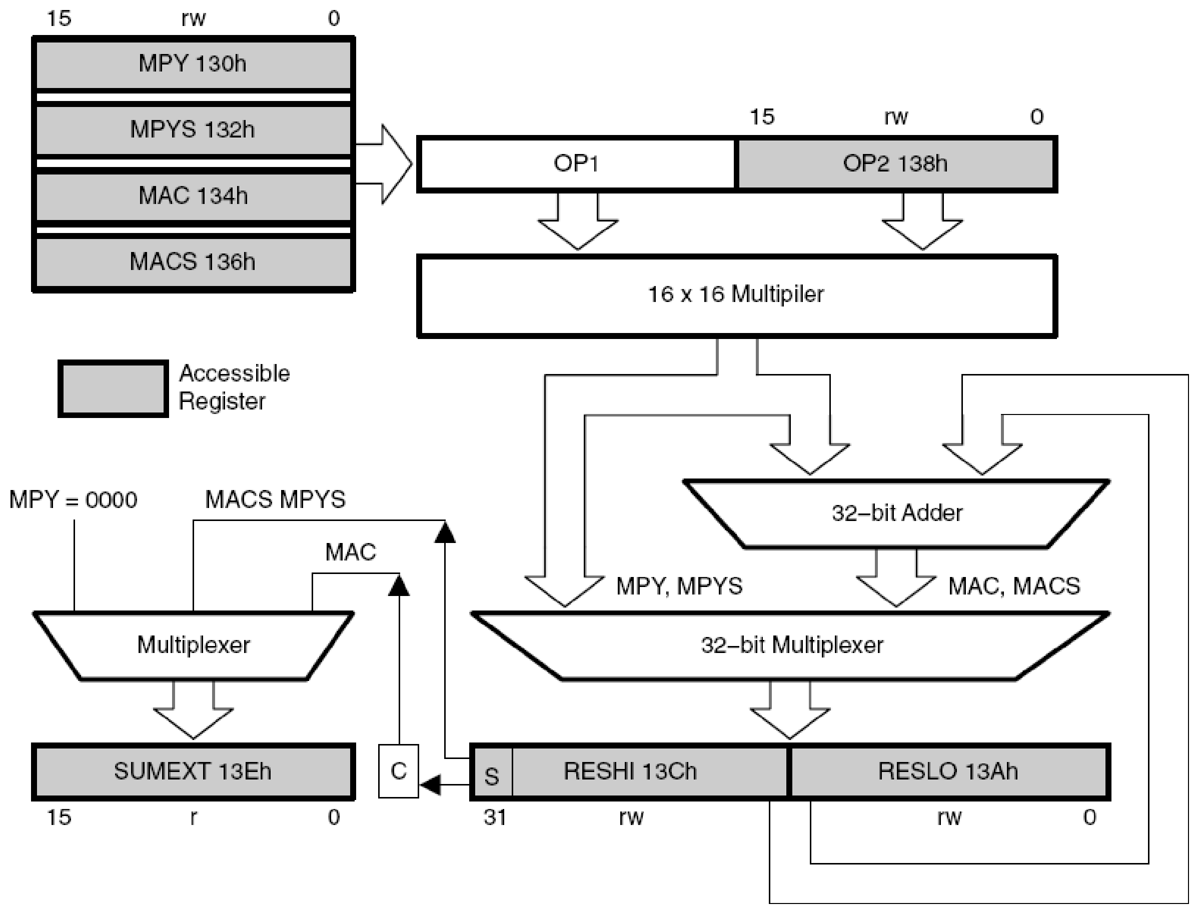

Table 12-1. OP1 addresses.

Register name	Multiplication operation	OP1 Address
MPY	Unsigned multiply	0130h
MPYS	Signed multiply	0132h
MAC	Unsigned multiply accumulate	0134h
MACS	Signed multiply accumulate	0136h

Table 12-2. RESHI contents.

Multiplication operation	RESHI content
Unsigned multiply (MPY)	Upper 16 bits of the result
Signed multiply (MPYS)	Bit 15 (MSB): sign Bits 14 - 0: upper 15 bits of the result Data format: Two’s complement
Unsigned multiply accumulate (MAC)	Upper 16 bits of the result
Signed multiply accumulate (MACS)	Upper 16 bits of the result Data format: Two’s complement

Table 12-3. SUMEXT contents.

Multiplication operation	SUMEXT content
Unsigned multiply (MPY)	SUMEXT = 0000h
Signed multiply (MPYS)	Extended sign of the result: SUMEXT = 00000h $\Rightarrow$ Result was positive or zero SUMEXT = 0FFFFh $\Rightarrow$ Result was negative
Unsigned multiply accumulate (MAC)	Carry of the result: SUMEXT = 0000h $\Rightarrow$ No carry for result SUMEXT = 0001h $\Rightarrow$ Result has a carry
Signed multiply accumulate (MACS)	Extended sign of the result: SUMEXT = 00000h $\Rightarrow$ Result was positive or zero SUMEXT = 0FFFFh $\Rightarrow$ Result was negative

Interrupts

The hardware multiplier should not be used in an interrupt service routine because the multiplier mode selection is lost and the results are unpredictable.

12.3 Signed and unsigned multiplication operation

See *Chapter 1 – Introductory Overview* for additional details.

Unsigned Multiply (MPY)

The two operands written to operand registers 1 and 2 are treated as unsigned numbers in the range 00000h (smallest number) to 0FFFFh (largest number).

The maximum possible result is obtained with input operands 0FFFFh and 0FFFFh:

```
0FFFFh x 0FFFFh = 0FFFE0001h
```

No carry is possible and the SUMEXT register always contains zero.

Signed Multiply (MPYS)

The two operands written to operand registers 1 and 2 are treated as signed Two's complement numbers, in the range 08000h (most negative number, –32768 in decimal) to 07FFFh (most positive number, +32767 in decimal).

The SUMEXT register contains the extended sign of the calculated result:

SUMEXT = 00000h: the result is positive;

SUMEXT = 0FFFFh: the result is negative.

Multiply-and-Accumulate (MAC)

The two operands written to operand registers 1 and 2 are treated as unsigned numbers (0h to 0FFFFh). The maximum possible result is obtained with input operands 0FFFFh and 0FFFFh:

```
0FFFFh x 0FFFFh = 0FFFE0001h
```

This result is added to the previous contents of the two sum registers (SUMLO and SUMHI). If a carry occurs during this operation, the SUMEXT register contains 1, otherwise it is cleared.

SUMEXT = 00000h: no carry occurred during the accumulation

SUMEXT = 00001h: a carry occurred during the accumulation

12.4 Hardware multiplier registers

The hardware multiplier registers are not intended to define the type of multiplication operation. Instead, they contain the operands and the data result. The registers used by the hardware multiplier are listed in *Table 12-4*.

Table 12-4. Hardware multiplier registers.

Register name	Description
MPY	Operand 1 - Unsigned multiply
MPYS	Operand 1 - Signed multiply
MAC	Operand 1 - Unsigned multiply accumulate
MACS	Operand 1 - Signed multiply accumulate
OP2	Operand 2
RESLO	Result (low word)
RESHI	Result (high word)
SUMEXT	Sum extension register

12.5 Laboratory 8: Multiplication operations analysis

This laboratory explores the hardware multiplier peripheral. It is composed of three different tasks, each of which evaluates a different characteristic of the hardware multiplier peripheral:

❑ Multiplication operation execution time, with and without the hardware multiplier.

❑ Differences between the use of the operator “*” and direct write to the hardware multiplier registers.

❑ Task operational analysis, in which the active power and the RMS value of an electrical system are calculated.

12.5.1 Lab8a: Multiplication without hardware multiplier

Project files

❑ C source files: **Chapter 12 > Lab8 > Lab8a_student.c**

Overview

This laboratory explores and analyses the MSP430's performance when it performs multiply operations without the hardware multiplier. The execution time is measured using an oscilloscope.

A. Resources

This laboratory only uses Port P2.1 connected to LED2 in order to measure the execution time of the multiply operation when it is performed by a software routine.

The default configuration of the FLL+ is used. All the clock signals required for the operation of the components of this device take their default values.

B. Software application organization

- ❑ The application starts by stopping the **Watchdog Timer**;
- ❑ Port P2.1 is configured as an output with the pin at a low level;
- ❑ The variables ***a*** and ***b*** to be multiplied are initialized;
- ❑ The multiplication of the two variables is performed between toggle P2.1 instructions;
- ❑ This application ends by putting the device into low power mode LPM4.

C. System configuration

Go to **Properties > TI Debug Settings** and select the **Target** tab. Uncheck the ***automatically step over functions without debug information when source stepping*** in order to allow stepping into the multiply routine;

Go to **Properties > C/C++ Build > Linker MSP430 Linker v3.0 > General options** and choose the option **None** at the ***Link in hardware version of RTS mpy routine***. With this linker option, the application will be built without the hardware multiplier and all multiplication operations will be performed by the software routine.

Rebuild the project and download it to the target.

D. Analysis of operation

- ❑ **Software multiplication routine analysis**
  - ■ Connect the oscilloscope probe to port P2.1 available on Header 4 pin 2;
  - ■ Put the cursor at line of code 51 {c = a*b} and **Run to line**;

- Go to **Disassembly** view and switch to **mixed disassembly view** in order to show both C and Assembly code;
- Observe that the variables ***a*** and ***b*** are passed by registers and the `#__mpyi` routine is called;
- Run the code step-by-step with the **Disassembly** view active. This action will lead to the software multiply routine;
- As the software multiply routine source code is not available, switch to **Assembly** view only;
- Run the application step-by-step until the RETA instruction;
- This multiplication is a time-consuming CPU operation.

❑ **Measurement of the multiply operation execution time**

- Restart the application. It will run from the beginning;
- Put the cursor on line of code 56 `{_BIS_SR(LPM4)}` and **Run to line**;
- Measure the time pulse time width using the oscilloscope;
- This software multiply operation takes around 54 μsec.

12.5.2 Lab8b: Multiplication with hardware multiplier

Project files

❑ C source files: **Chapter 12 > Lab8 > Lab8b_student.c**

Overview

This laboratory explores and analyses the MSP430's performance when it performs multiply operations using the hardware multiplier peripheral. Two different variants are analysed:

❑ Using the "*" operator;

❑ Accessing the hardware multiplier registers directly.

The execution times are measured with an oscilloscope.

A. Resources

This laboratory only uses Port P2.1 connected to LED2 in order to measure the execution time of the multiplication operation, when it is performed by the hardware multiplier.

The default configuration of the FLL+ is used. All the clock signals required for the operation of the components of the device take their default values.

B. Software application organization

- The application begins by stopping the **Watchdog Timer**;
- Port P2.1 is configured as an output with the pin at a low level;
- The code can be broken down into two parts:
  - In the first part of the code, the multiplication is performed with the "*****" operator. This task is performed between P2.1 toggles, in order to determine the time required to perform this operation;
  - The remaining part of the code is separated by some `_NOP()` operations. This coding allows analysis of the execution time using an oscilloscope. Here, the multiplication operation is performed by directly accessing the hardware multiplier registers. The multiplication of the variables is performed between toggle P2.1 instructions;
- This application ends with the device entering low power mode LPM4.

C. System configuration

Go to **Properties > TI Debug Settings** and select the **Target** tab. Uncheck the ***automatically step over functions without debug information when source stepping*** in order to allow stepping into the multiply routine.

Go to **Properties > C/C++ Build > Linker MSP430 Linker v3.0 > General options** and choose the option **16 (default)** at the ***Link in hardware version of RTS mpy routine***. With this linker option, the application will be built with the 16-bit hardware multiplier peripheral contained in the Experimenter's board.

Rebuild the project and download to the target.

D. Analysis of operation

❑ Analysis of hardware multiply routine with the "*" operator

- Connect the oscilloscope probe to port P2.1, which is connected to Header 4 pin 2;
- Put the cursor at line of code 55 {c = a*b} and **Run to line**;
- Go to **Disassembly** view and switch to **mixed disassembly view** in order to show both C and Assembly code;
- Observe that the variables ***a*** and ***b*** are passed to registers and `#__mpyi_hw` routine is called;
- Run the code step-by-step with the **Disassembly** view active. This action will lead to the multiply operation being performed by the hardware multiplier;

- As the hardware multiply routine source code is not available, switch to **Assembly** view only;
- The routine starts by pushing the **Status Register** onto the system stack (`PUSH` instruction) and disabling the interrupts (this always occurs when using the hardware multiplier peripheral);
- The next line of code exchanges data with the hardware multiplier;
- Then the SR is popped (`POP` instruction) from the system stack, restoring the system environment (data interrupt state restored);
- The routine finishes with a `RETA` instruction.

❑ Analysis of hardware multiply operation with direct registers access

- Switch to the **C** view;
- Put the cursor at line of code 72 {`MPY = a`} and **Run to line**;
- The routine call operation is avoided, as shown in the **Disassembly** view. This exemplifies an energy saving procedure because it shows how less CPU clock cycles can be used.

❑ Measurement of execution time of the multiply operation

- Restart the application. It will run from the beginning;
- Put the cursor at line of code 77 {`_BIS_SR(LPM4)`} and **Run to line**;
- Measure the pulse widths using the oscilloscope;
- The first time pulse corresponds to the hardware multiply routine with the operator "*", and has a width of 42 μsec;
- The second time pulse corresponds to the hardware multiply register operation and has a width of 19 μsec;
- Comparing both time pulses and the time pulse obtained in **Lab8a**, it can be seen that with the hardware multiplier there is a significant reduction of the time required to perform a multiply operation;
- The smaller time pulse corresponds to the hardware multiply operation writing directly to the hardware multiplier registers. This reduction in time means less power consumption, which is very useful for the design of low-power applications.

12.5.3 Lab8c: RMS and active power calculation

Project files

- ❑ C source files: **Chapter 12 > Lab8 > Lab8c_student.c**

Overview

This laboratory explores and analyses the MSP430 performance when it makes multiply operations using the hardware multiplier peripheral. In this laboratory, the active power and the RMS value of an electrical signal are calculated.

The execution times are measured using an oscilloscope.

A. Resources

This laboratory only uses Port P2.1 connected to LED2, in order to measure the execution time of the multiply operation when it is performed by the hardware multiplier.

The application uses the default configuration of the FLL+. All the clock signals required for the operation of the components of the device take their default values.

B. Software application organization

- The application starts by stopping the **Watchdog Timer**;
- Two _NOP() instructions are provided to associate breakpoints, in order to read current and voltage samples (N = 200) from files;
- Power is computed by applying the following formula:

$$P = \frac{1}{N}\sum_{k=1}^{N} u_k \, i_k$$

- A signed multiply operation is performed by writing the first sample of current to **MPYS** and the first sample of voltage to **OP2**;
- The result of the multiplication is stored in the **RESHI** and **RESLO** registers;
- A loop is performed with a signed multiply and accumulate (**MACS**) operation;
- The final result is transferred from the **RESHI** and **RESLO** registers to the long variable ***result***;

- The power is computed by dividing the variable ***result*** by the number of samples (*N*);
- Port P2.1 is active between **MACS** operations;
- The RMS current and voltage values are calculated from the following expressions:

$$I_{RMS} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} i_k \, i_k}$$

$$U_{RMS} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} u_k \, u_k}$$

- The two procedures are similar, with the exception of the square root (`sqrt`) operations;
- P2.1 is active during for all the RMS current calculation;
- The computation times of the `sqrt` and division operations are determined when the RMS voltage value is calculated;
- This application ends by putting the device into low power mode LPM4.

C. System configuration

Go to **Properties > TI Debug Settings** and select the **Target** tab. Uncheck the ***automatically step over functions without debug information when source stepping*** in order to allow stepping into the multiply routine.

Go to **Properties > C/C++ Build > Linker MSP430 Linker v3.0 > General options** and choose the option **16 (default)** at the ***Link in hardware version of RTS mpy routine***. With this linker option, the application will be built with the 16-bit hardware multiplier peripheral, contained in the Experimenter's board.

Rebuild the project and download the code to the target.

D. Analysis of operation

❑ **Loading samples from files**

- Insert a breakpoint at line of code 61 (first `_NOP()` operation);
- Edit **Breakpoint Properties** and choose the **Read Data from file** action;

- Configure the following data fields:
  - File: i.txt
  - Wrap around: True
  - Start address: &i
  - Length: 200

- Include a breakpoint at line of code 63 (second `_NOP()` operation);
- Edit **Breakpoint Properties** and choose the **Read Data from file** action;
- Configure the following data fields:
  - File: u.txt
  - Wrap around: True
  - Start address: &u
  - Length: 200

- Alternatively, import a breakpoint file from the project source code folder ***Lab8c_breakpoint.bkpt***;
- Put the cursor at line of code 67 and **Run to line**;
- In the **Variables** view, add the global variables ***i*** and ***u***;
- Check the data inside these arrays.

❑ **Computing active power**

- Connect the oscilloscope probe to port P2.1, which is available at Header 4 pin 2;
- Put the cursor at the line of code 88 and **Run to line**;
- In the **Variables** view, add the global variable ***P*** and format it to decimal;
- The active power is in the region of 1204 W;
- The pulse width, as viewed on the oscilloscope, corresponds to the time to perform the 200 signed multiply and accumulate operations and is 5.4 msec.

❑ **Compute RMS current value**

- Starting at the last step of the previous task, put the cursor at line of code 105 {`MPYS = u[0]`} and **Run to line**;
- Add the global variable ***I*** (RMS voltage);
- Set the value to 10;

- The pulse width, as viewed on the oscilloscope, corresponds to the time required to perform the 200 signed multiply and accumulate operations, 1 division operation and 1 square root operation, and is 12.6 msec;

❑ **Compute RMS voltage value**

- Starting at the last step of the previous task, put the cursor at line of code 121 {`_BIS_SR(LPM4)`} and **Run to line**;
- Add the global variable ***U*** (RMS voltage);
- Set the value to 240;
- The pulse width, as viewed on the oscilloscope, corresponds to the time to perform the 200 signed multiply and accumulate operations, and is 6.8 msec;

12.6 Quiz

1. The Hardware Multiplier peripheral module is implemented in the:

(a) eZ430-F2013 hardware development tool;

(b) eZ430-RF2500 hardware development tool;

(c) Experimenter’s board;

(d) None of above.

2. The Hardware Multiplier supports:

(a) Unsigned and signed multiply;

(b) Unsigned and signed multiply and accumulate;

(c) 16×16 bits, 16×8 bits, 8×16 bits, 8×8 bits;

(d) All of above.

3. The RESHI result register stores:

(a) The low word of the result;

(b) The high word of the result;

(c) The extended sign of the result;

(d) The carry of the result.

4. The MSB bit of RESHI register for the Signed multiply contains:

(a) The sign of the result;

(b) The MSB of the result;

(c) The carry of the result;

(d) None of above.

5. A SUMEXT = 0 in a MACS operation indicates that the result is:

(a) Positive;

(b) Zero;

(c) All of above;

(d) None of above.

6. The data format of the RESHI register for a signed multiplication operation is:

(a) Straight binary;

(b) Two's complement;

(c) One's complement;

(d) None of above.

Solution: 1. (c); 2. (d); 3. (b); 4. (a); 5. (c); 6. (b)

12.7 FAQs

1. Can I modify the operand registers during a multiplication operation?

Yes, but the result will be invalid.

2. Can I modify the result registers during a multiplication operation?

The actual result will not be valid if the result registers were modified after writing the second operand and before the multiplication operation is completed.