Divide Routines

Features

- **8- and 16-bit Implementations**
- **Signed and Unsigned Routines**
- **Speed and Code-size Optimized Routines**
- **Runable Example Programs**
- **Speed is Comparable with Hardware Dividers**
- **Extremely Compact Code**

Introduction

This application note lists subroutines for division of 8- and 16-bit signed and unsigned numbers for the AT94K Field Programmable System Level Integrated Circuit (FPSLIC™) and the AT94S Secure FPSLIC. A listing of all implementations with key performance specifications is given in Table 1.

Multiplication is not covered in this application note because the FPSLIC device includes a hardware multiplier. For information on using the hardware multiplier, refer to the “Using the FPSLIC Hardware Multiplier” application note, available at the Atmel web site (http://www.atmel.com).

Table 1. Performance Figures Summary

Application	Code Size (Words)	Execution Time (Cycles)
8/8 = 8 + 8-bit Unsigned (Code Optimized)	14	97
8/8 = 8 + 8-bit Unsigned (Speed Optimized)	66	58
8/8 = 8 + 8-bit Signed (Code Optimized)	22	103
16/16 = 16 + 16-bit Unsigned (Code Optimized)	19	243
16/16 = 16 + 16-bit Unsigned (Speed Optimized)	196	173
16/16 = 16 + 16-bit Signed (Code Optimized)	39	255

This application note listing consists of two files:

- **“div_a.asm”:** Code-size optimized divide routines.
- **“div_b.asm”:** Speed-optimized divide routines.

FPSLIC™

Programmable SLI AT94K/AT94S Series

Application Note

Rev. 1973B–11/01

8/8 = 8 + 8 Unsigned Division – “div8u”

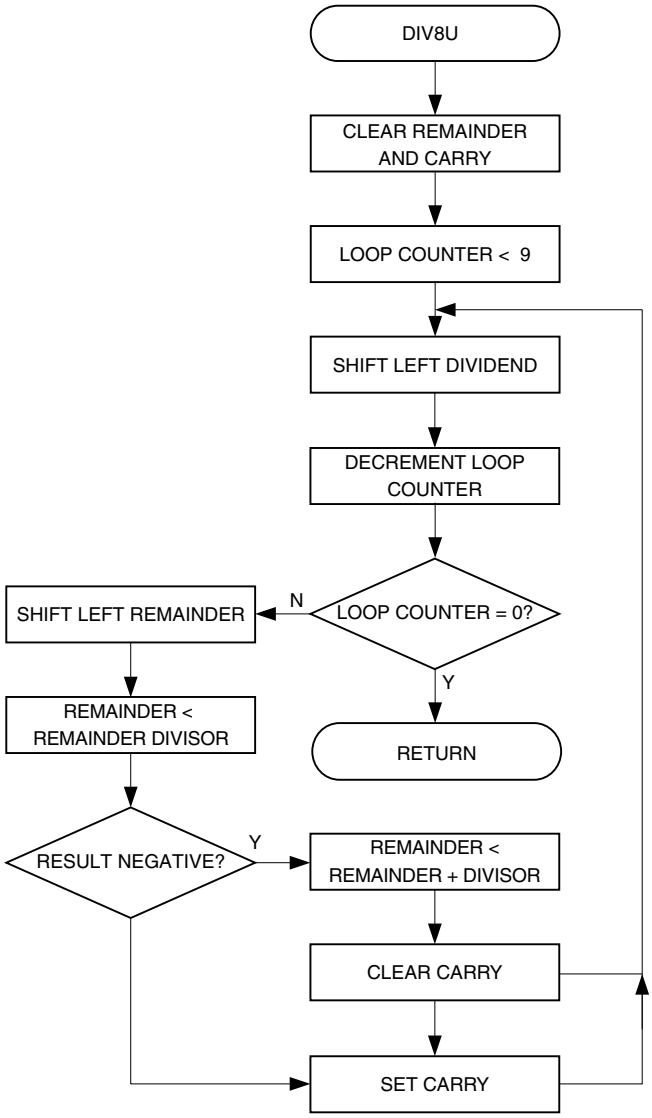
Both program files contain a routine called “div8u”, which performs unsigned 8-bit division. Both implementations are based on the same algorithm. The code-size optimized implementation uses looped code, whereas the speed-optimized code is a straight-line code implementation. Figure 1 shows the flowchart for the code-size optimized version.

Algorithm Description

The algorithm for unsigned 8/8 division (code-size optimized code) is as follows:

1. Clear remainder and carry.
2. Load loop counter with 9.
3. Shift left dividend into carry.
4. Decrement loop counter.
5. If loop counter = 0, return.
6. Shift left carry (from dividend/result) into remainder.
7. Subtract divisor from remainder.
8. If result negative, add back divisor, clear carry and go to step 3.
9. Set carry and go to step 3.

Figure 1. “div8u” Flowchart (Code-size Optimized Implementation)



Usage

The usage of “div8u” is the same for both implementations and is described in the following procedure:

1. Load register variable “dd8u” with the dividend (the number to be divided).
2. Load register variable “dv8u” with the divisor (the dividing number).
3. Call “div8u”.
4. The result is found in “dres8u” and the remainder in “drem8u”.

Performance

Table 2. “div8u” Register Usage (Code-size Optimized Version)

Register	Input	Internal	Output
R15		–	“drem8u” – Remainder
R16	“dd8u” – Dividend	–	“dres8u” – Result
R17	“dv8u” – Divisor	–	–
R18	–	“dcnt8u” – Lop Counter	–

Table 3. “div8u” Performance Figures (Code-size Optimized Version)

Parameter	Value
Code Size (Words)	14
Execution Time (Cycles)	97
Register Usage	Low Registers – 1 High Registers – 3 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

Table 4. “div8u” Register Usage (Speed Optimized Version)

Register	Input	Internal	Output
R15	–	–	“drem8u” – Remainder
R16	“dd8u” – Dividend	–	“dres8u” – Result
R17	“dv8u” – Divisor	–	–

Table 5. “div8u” Performance Figures (Speed Optimized Version)

Parameter	Value
Code Size (Words)	66
Execution Time (Cycles)	58
Register Usage	Low Registers – 1 High Registers – 2 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

8/8 = 8 + 8 Signed Division – “div8s”

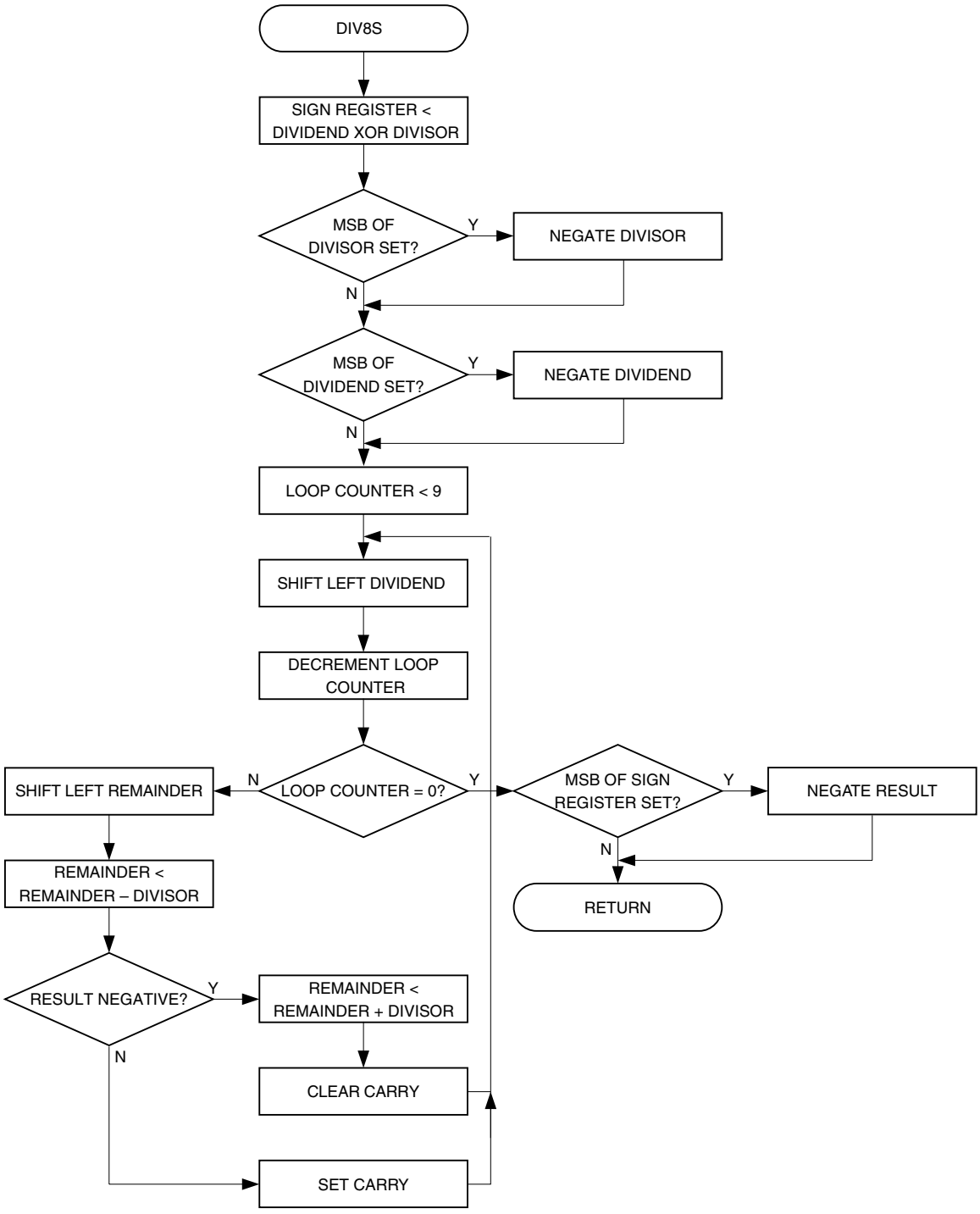
The subroutine “mpy8s” implements signed 8-bit division. The implementation is code-size optimized. If negative, the input values shall be represented on two’s complement’s form. Figure 2 shows the flowchart for the signed 8/8 division.

Algorithm Description

The algorithm for signed 8/8 division is as follows:

1. XOR dividend and divisor and store in a sign register.
2. If MSB of dividend set, negate dividend.
3. If MSB of divisor set, negate dividend.
4. Clear remainder and carry.
5. Load loop counter with 9.
6. Shift left dividend into carry.
7. Decrement loop counter.
8. If loop counter ≠ 0, go to step 11.
9. If MSB of sign register set, negate result.
10. Return.
11. Shift left carry (from dividend/result) into remainder.
12. Subtract divisor from remainder.
13. If result negative, add back divisor, clear carry and go to step 6.
14. Set carry and go to step 6.

Figure 2. “div8s” Flowchart (Signed 8/8 Division)



Usage

The usage of “div8s” follows the procedure below:

1. Load register variable “dd8s” with the dividend (the number to be divided).
2. Load register variable “dv8s” with the divisor (the dividing number).
3. Call “div8s”.
4. The result is found in “dres8s” and the remainder in “drem8s”.

Performance

Table 6. "div8s" Register Usage

Register	Input	Internal	Output
R14	–	"d8s" – Sign Register	–
R15	–	–	"drem8s" – Remainder
R16	"dd8s" – Dividend	–	"dres8s" – Result
R17	"dv8s" – Divisor	–	–
R18	–	"dcnt8s" – Lop Counter	–

Table 7. "div8s" Performance Figures

Parameter	Value
Code Size (Words)	22
Execution Time (Cycles)	103
Register Usage	Low Registers – 2 High Registers – 3 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

16/16 = 16 + 16 Unsigned Division – "div16u"

Both program files contain a routine called "div16u", which performs unsigned 16-bit division.

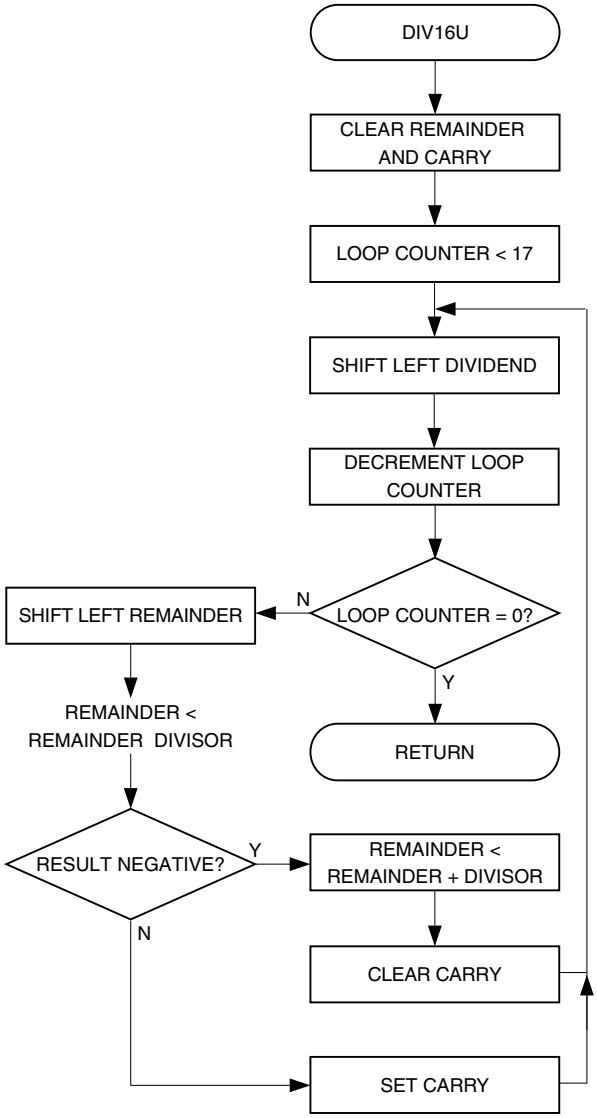
Both implementations are based on the same algorithm. The code-size optimized implementation uses looped code, whereas the speed optimized code is a straight-line code implementation. Figure 3 shows the flowchart for the code-size optimized version.

Algorithm Description

The algorithm for unsigned 16/16 division (code-size optimized code) is as follows:

1. Clear remainder and carry.
2. Load loop counter with 17.
3. Shift left dividend into carry
4. Decrement loop counter.
5. If loop counter = 0, return.
6. Shift left carry (from dividend/result) into remainder.
7. Subtract divisor from remainder.
8. If result negative, add back divisor, clear carry and go to step 3.
9. Set carry and go to step 3.

Figure 3. “div16u” Flowchart (Code-size Optimized Implementation)



Usage

The usage of “div16u” is the same for both implementations and is described in the following procedure:

1. Load the 16-bit register variable “dd16uH:dd16uL” with the dividend (the number to be divided).
2. Load the 16-bit register variable “dv16uH:dv16uL” with the divisor (the dividing number).
3. Call “div16u”.
4. The result is found in “dres16u” and the remainder in “drem16u”.

Performance

Table 8. "div16u" Register Usage (Code-size Optimized Version)

Register	Input	Internal	Output
R14	–	–	"drem16uL" – Remainder Low Byte
R15	–	–	"drem16uH – Remainder High Byte
R16	"dd16uL" – Dividend Low Byte	–	"dres16uL" – Result Low Byte
R17	"dd16uH" – Dividend High Byte	–	"dres16uH" – Result High Byte
R18	"dv16uL" – Divisor Low Byte	–	–
R19	"dv16uH" – Divisor High Byte	–	–
R20	–	"dcnt16u" – Lop Counter	–

Table 9. "div16u" Performance Figures (Code-size Optimized Version)

Parameter	Value
Code Size (Words)	19
Execution Time (Cycles)	243
Register Usage	Low Registers – 2 High Registers – 5 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

Table 10. "div16u" Register Usage (Speed Optimized Version)

Register	Input	Internal	Output
R14	–	–	"drem16uL" – Remainder Low Byte
R15	–	–	"drem16uH – Remainder High Byte
R16	"dd16uL" – Dividend Low Byte	–	"dres16uL" – Result Low Byte
R17	"dd16uH" – Dividend High Byte	–	"dres16uH" – Result High Byte
R18	"dv16uL" – Divisor Low Byte	–	–
R19	"dv16uH" – Divisor High Byte	–	–

Table 11. "div16u" Performance Figures (Speed Optimized Version)

Parameter	Value
Code Size (Words)	196 + return
Average Execution Time (Cycles)	173
Register Usage	Low Registers – 2 High Registers – 4 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

16/16 = 16 + 16 Signed Division – "div16s"

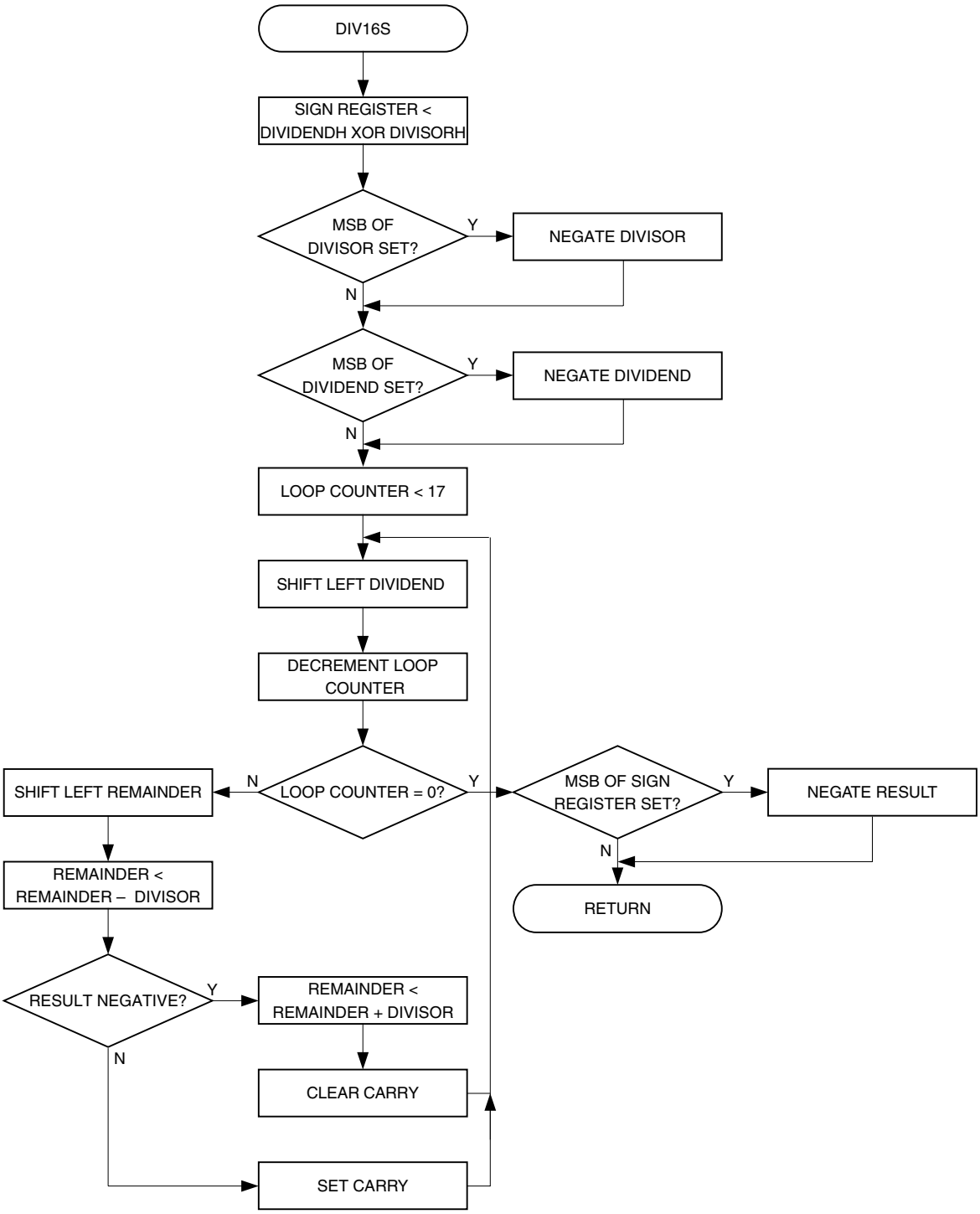
The subroutine "mpy16s" implements signed 16-bit division. The implementation is code-size optimized. If negative, the input values shall be represented on two's complement's form. Figure 4 shows the flowchart for the signed 16/16 division.

Algorithm Description

The algorithm for signed 16/16 division is as follows:

1. XOR dividend and divisor high bytes and store in a Sign register.
2. If MSB of dividend high byte set, negate dividend.
3. If MSB of divisor set high byte, negate dividend.
4. Clear remainder and carry.
5. Load loop counter with 17.
6. Shift left dividend into carry.
7. Decrement loop counter.
8. If loop counter $\neq 0$, go to step 11.
9. If MSB of sign register set, negate result.
10. Return.
11. Shift left carry (from dividend/result) into remainder.
12. Subtract divisor from remainder.
13. If result negative, add back divisor, clear carry and go to step 6.
14. Set carry and go to step 6.

Figure 4. “div16s” Flowchart (Signed 16/16 Division)



Usage

The usage of “div16s” is described in the following procedure:

1. Load the 16-bit register variable “dd16sH:dd16sL” with the dividend (the number to be divided).
2. Load the 16-bit register variable “dv16sH:dv16sL” with the divisor (the dividing number).
3. Call “div16s”.
4. The result is found in “dres16s” and the remainder in “drem16s”.

Performance

Table 12. “div16s” Register Usage

Register	Input	Internal	Output
R14	–	–	“drem16sL” – Remainder Low Byte
R15	–	–	“drem16sH” – Remainder High Byte
R16	“dd16sL” – Dividend Low Byte	–	“dres16sL” – Result Low Byte
R17	“dd16sH” – Dividend High Byte	–	“dres16sH” – Result High Byte
R18	“dv16sL” – Divisor Low Byte	–	–
R19	“dv16sH” – Divisor High Byte	–	–
R20	–	“dcnt16s” – Lop Counter	–

Table 13. “div16s” Performance Figures

Parameter	Value
Code Size (Words)	39
Execution Time (Cycles)	255
Register Usage	Low Registers – 2 High Registers – 5 Pointers – None
Interrupts Usage	None
Peripherals Usage	None

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Product Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Atmel Heilbronn
Theresienstrasse 2
POB 3535
D-74025 Heilbronn, Germany
TEL (49) 71 31 67 25 94
FAX (49) 71 31 67 24 23

Atmel Nantes
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 0 2 40 18 18 18
FAX (33) 0 2 40 18 19 60

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743

Atmel Programmable SLI Hotline
(408) 436-4119

Atmel Programmable SLI e-mail
fpslic@atmel.com

FAQ
Available on web site

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

1973B–11/01/xM