APPLICATION NOTE

AT6486: Using DIVAS on SAMC Microcontroller

SMART ARM-Based Microcontroller

Introduction

DIVAS stands for Division and Square Root Accelerator. DIVAS is a brand new peripheral introduced in SAMC family of microcontrollers. SAMC family is mainly targeted towards home appliance and industrial applications market. Such applications mostly require more involved mathematical operations for algorithms and automation calculations.

In most languages, including C, division is performed using intrinsic operators. Quotient and division modulus are represented by '/' and '%' operators respectively. For controllers without hardware divider (like Cortex® M0+), compiler replaces these operators with library code. This library code adds hundreds of bytes and consumes anywhere from 50 to 400 clock cycles depending on the size of operands. For square root, an additional library has to be included in the project (math.h). DIVAS addresses these issues by providing square root and division support in hardware and requires minimal code and clock cycles (2-16 cycles).

This application note discusses the APIs (application programming interfaces) available in Atmel Studio Framework (ASF) that make DIVAS easy to use. It also explains how compiler division operators ('%' and '/') can be overloaded to use DIVAS as backend for calculations.

Features

- Division and square root accelerator for Cortex M0+ systems
- Signed and unsigned division
- Unsigned 32-bit square root
- Result includes quotient/square root and remainder

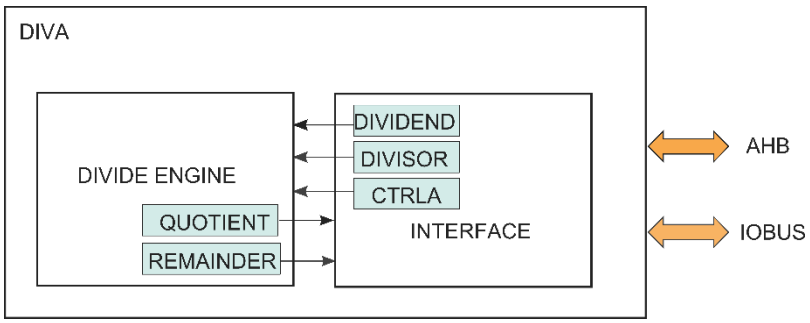
Table of Contents

1 Overview

1.1 Description

The Divide and Square Root Accelerator (DIVAS) is a programmable 32-bit signed or unsigned hardware divider and a 32-bit unsigned square root hardware engine. The DIVAS is connected to the high-speed bus matrix and may also be accessed using the low-latency CPU local bus (IOBUS; ARM® single-cycle I/O port). The DIVAS takes dividend and divisor values and returns the quotient and remainder when it is used as divider. The DIVAS takes unsigned input value and returns its square root and remainder when it is used as square root function.

Figure 1-1. DIVAS Block Diagram



1.2 Register Interface

DIVAS peripheral has seven registers. These registers are:

- CTRLA
  - Control register to disable leading zero optimization and enable signed division
- STATUS
  - Status register to detect divide by zero error and busy bit
  - When BUSY bit is set, CTRLA, DIVIDEND, DIVISOR, and SQRNUM registers are write protected
- DIVIDEND
  - Holds the 32-bit dividend for the divide operation
  - If CTRLA.SIGNED bit is set, DIVIDEND is signed 2's complement
- DIVISOR
  - Holds the 32-bit divisor for the divide operation
  - If CTRLA.SIGNED bit is set, DIVISOR is signed 2's complement
  - Writing DIVISOR register starts the divide function
- RESULT
  - Holds the 32-bit result. For division it is the quotient, for square root operation, it is the square root
  - In case of division, if CTRLA.SIGNED bit is set, quotient is signed 2's complement
- REMAINDER
  - Holds the 32-bit remainder for divide or square root operation
  - In case of division, if CTRLA.SIGNED bit is set, remainder is signed 2's complement
- SQRNUM
  - Holds the 32-bit unsinged input for the square root operation
  - Writing SQRNUM register starts the square root operation

2 Functional Description

The Divide and Square Root Accelerator (DIVAS) supports signed or unsigned hardware division of 32-bit values and unsigned square root of 32-bit value. It is accessible from the CPU via both the AHB bus and IOBUS. When the dividend and divide registers are programmed, the division starts and the result will be stored in the Result and Remainder registers. The Busy and Divide-by-zero status can be read from STATUS register.

2.1 Basic Operation

2.1.1 Initialization

THE DIVAS configuration cannot be modified while a divide operation is ongoing. The following bits must be written prior to starting a division.

- Sign selection bit in Control A register

Sign bit enables the signed division. Signed division is explained in Section 2.1.4 Signed Division.

- Disable leading zero optimization bit in Control A

Leading zero optimization is explained in Section 2.1.6 Leading Zero Optimization.

2.1.2 Clocks

The DIVAS bus clock (CLK_DIVAS_AHB) can be enabled or disabled in the power manager. This clock has to be enabled before using DIVAS.

2.1.3 Performing Division

First write the dividend to DIVIDEND register. Writing the divisor to DIVISOR register starts the division and sets the busy bit in the Status register (STATUS.BUSY). When the division has completed, the STATUS.BUSY bit is cleared and the result will be stored in RESULT and REMAINDER registers.

The RESULT and REMAINDER registers can be read directly via the high-speed bus without checking first STATUS.BUSY. Wait states will be inserted on the high-speed bus until the operation is complete. The IOBUS does not support wait states. For accesses via the IOBUS, the STATUS.BUSY bit must be polled before reading the result from the RESULT and REMAINDER registers.

2.1.4 Signed Division

When CTRLA.SIGNED is one, both the input and the result will be in 2’s complement format. The results of signed division are such that the remainder and dividend have the same sign and the quotient is negative if the dividend and divisor have opposite signs. 16-bit results are sign extended to 32-bits.

Note: When the maximum negative number is divided by the minimum negative number, the resulting quotient overflows the signed integer range and will return the maximum negative number with no indication of the overflow. This occurs for 0x80000000 / 0xFFFFFFFF in 32-bit operation and 0x8000 / 0xFFFF in 16-bit operation.

2.1.5 Divide by Zero

A divide by zero fault occurs if the DIVISOR is programmed to zero. QUOTIENT will be zero and the REMAINDER is equal to DIVIDEND. Divide by zero sets the Divide-by-zero bit in the Status register (STATUS.DBZ) to one. STATUS.DBZ must be cleared by writing a one to it.

2.1.6 Leading Zero Optimization

Leading zero optimization can reduce the time it takes to complete a division by skipping leading zeros in the DIVIDEND (or leading ones in signed mode). Leading zero optimization is enabled by default and can be disabled by the Disable Leading Zero bit in the Control A register (CTRLA.DLZ). When CTRLA.DLZ is zero, 16-bit division completes in 2-8 cycles and 32-bit division completes in 2-16 cycles, depending on the dividend value. If deterministic timing is required, setting CTRLA.DLZ to one forces 16-bit division to always take eight cycles and 32-bit division to always take 16 cycles.

2.1.7 Unsigned Square Root

When the square root input register (SQRNUM) is programmed, the square root function starts and the result will be stored in the Result and Remainder registers. The Busy status can be read from STATUS register.

2.2 Usage Summary

For division:

- Set sign bit in CTRLA register
- Set leading zero mode bit in CTRLA register
- Write to dividend register
- Write to divisor register (start division)
- Wait for STATUS.BUSY bit to be cleared
- Read the RESULT and REMAINDER registers through AHB or IOBUS

For square root:

- Write to SQRNUM register
- Wait for STATUS.BUSY bit to be cleared
- Read the RESULT and REMAINDER registers through AHB or IOBUS

3 Firmware Implementation

DIVAS drivers are a part of Atmel Studio Framework. Easy to use APIs have been provided to use DIVAS. There are two ways to use DIVAS in firmware; Peripheral APIs and operator overloading. Both these implementations are demonstrated in the application example accompanying this application note.

3.1 Peripheral APIs

DIVAS can be accessed by application program interfaces like any other peripheral. The ASF drivers provide APIs to set/reset each bit. The list of APIs provided by the driver are:

- `int32_t divas_idiv(int32_t numerator, int32_t denominator);`
- `uint32_t divas_uidiv(uint32_t numerator, uint32_t denominator);`
- `int32_t divas_idivmod(int32_t numerator, int32_t denominator);`
- `uint32_t divas_uidivmod(uint32_t numerator, uint32_t denominator);`
- `uint32_t divas_sqrt(uint32_t radicand);`
- `static inline void divas_enable_dlz(void);`
- `static inline void divas_disable_dlz(void);`
- `void _system_divas_init(void);`

3.2 Compiler Operator Overloading

A simpler way of using DIVAS is making it invisible to the programmer. This can be achieved by replacing the backend for compiler intrinsic operators '/' and '%'. When compiler compiles the statements containing these operators, it uses DIVAS APIs instead of library generated code automatically. Unfortunately, it is not possible for square root because square root does not have a C intrinsic operator.

DIVAS driver provided with ASF provides code for overloading division operators for both Atmel Studio and IAR™. The ASF example provided with this application note can be compiled to use overloaded operators by setting the following `#define` to true in project symbols. It is explained in Section 3.3.2 Description.

```
#define USE_OVERLOAD_OPERATORS true
```

3.3 DIVAS Example

This application note comes with an application example firmware. This application has been developed using Atmel Studio but the code is compatible with IAR as well.

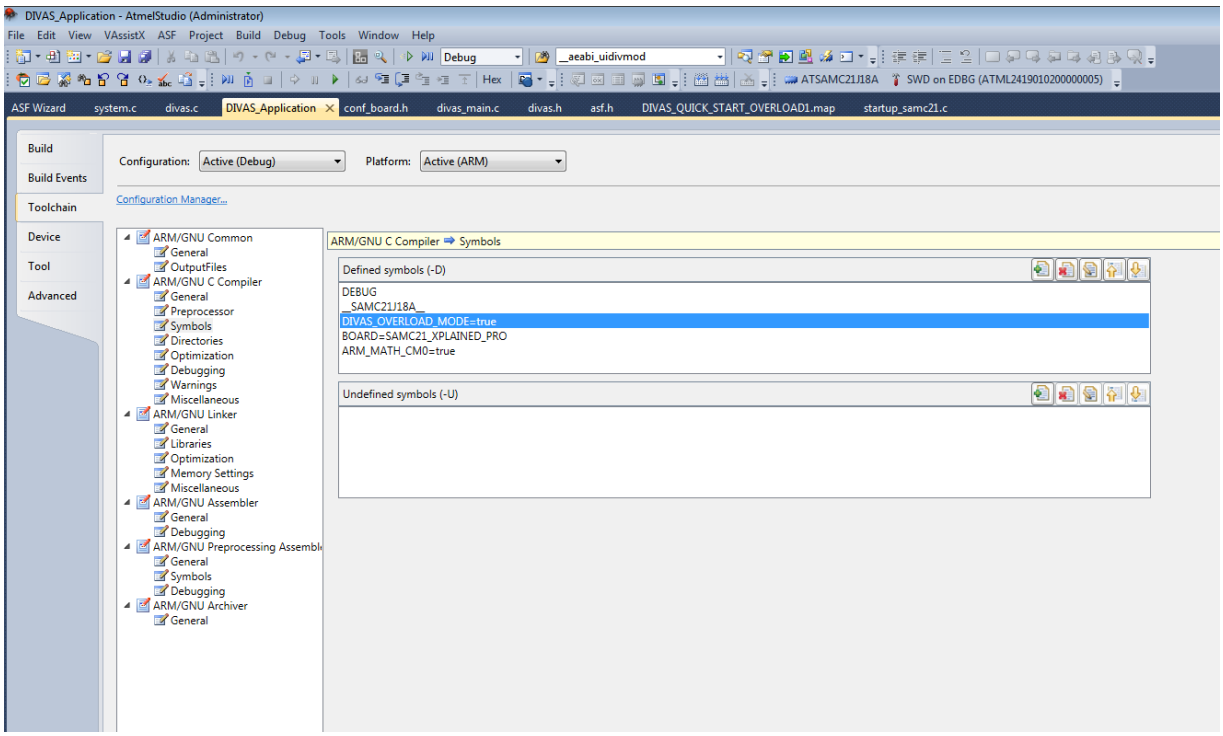
3.3.1 Requirements

- Atmel Studio with SAM C21 support
- SAM C21 Xplained Pro board

3.3.2 Description

The application should build without any errors on Atmel Studio. The application has test vectors in the form of arrays. If the firmware passes the tests, the LED will turn on. If the tests fail, the LED will start blinking. The application, by default is set to use DIVAS APIs. Overloaded operators can be enabled by defining `USE_OVERLOAD_OPERATORS` as true in the project symbols as shown in Figure 3-1. By default, it is set to false.

Figure 3-1. Project Properties



4 Revision History

Doc Rev.	Date	Comments
42465A	06/2015	Initial document release.

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**