APPLICATION NOTE

PAC52XX FLASH Memory Controller Configuration

Power Application Controller™

Marc Sousa
Senior Manager, Systems and Firmware

www.active-semi.com
Copyright © 2014 Active-Semi, Inc.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

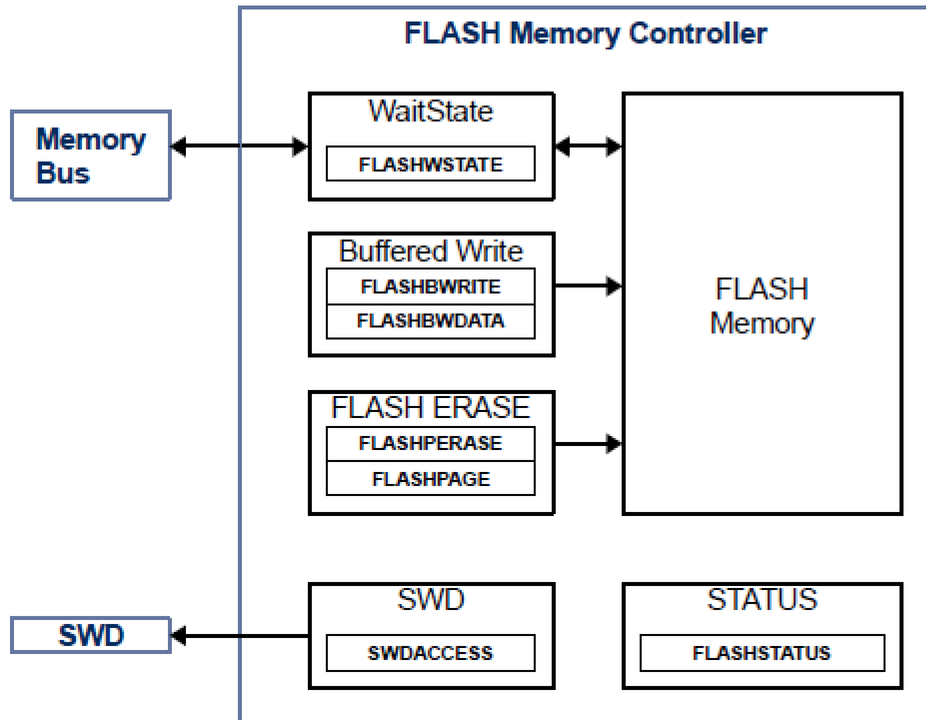
TABLE OF CONTENTS

APPLICATION NOTE 1
Table of Contents 2
Overview 3
Setting the FLASH Wait State 4
Configuration Using the PAC52XX SDK 5
Configuration Using the PAC52XX Peripheral Register Interface 5
Erasing and Writing FLASH Memory 6
Flash Configuration 6
Writing FLASH Memory 7
Reading Configuration Information from FLASH 9
FLASH Buffered Writes 11
Buffered Write using PAC52XX SDK 11
Buffered Write using PAC52XX Peripheral Register Interface 12
SWD Debug Access Protection 13
SWD Debug Access Protection Code 13
About Active-Semi 14

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

OVERVIEW

The PAC52XX family of devices contains a FLASH memory controller that manages access to FLASH memory. See the block diagram shown below:



The FLASH memory controller manages FLASH access from the AHB bus (not shown in the diagram above). The AHB bus makes requests to FLASH that can come from both SWD as well as the ARM Cortex-M0 MCU.

The FLASH memory controller allows the user to change the FLASH wait states, erase FLASH memory segments, performed buffered writes as well as disabling SWD access for secure code protection.

See the sections later in this document for a description of each of these techniques.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

SETTING THE FLASH WAIT STATE

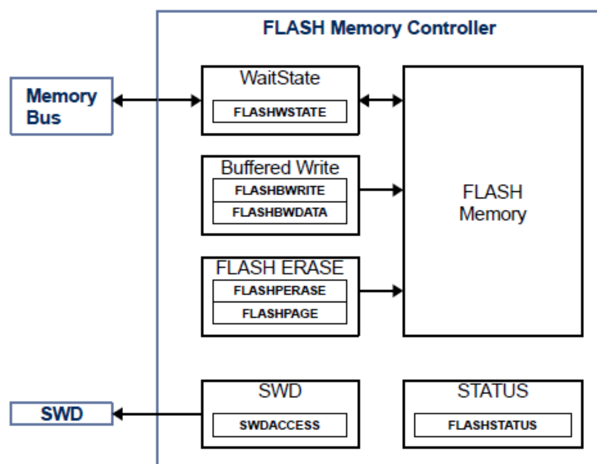
The PAC52XX family of controllers allows the user to have a wide range of clock configurations to support the ARM Cortex-M0 MCU as well as all the other system peripherals.

The FLASH memory on the PAC52XX devices has a maximum clock frequency that is typically lower than the HCLK, which clocks the ARM Cortex-M0. In order for FLASH memory to run correctly, it must not be clocked above the specified frequency.

Note that when code is executing from RAM, the same limitation does not exist: RAM can run as fast as HCLK may be configured.

For example, in the PAC5220 device, FLASH memory has an access time of 40ns (25MHz). If the user configures HCLK to be higher than 25MHz, then the clock to FLASH memory must be configured such that the maximum frequency is 25MHz.

See the block diagram below for reference.



The FLASH Memory controller is clocked using HCLK (not shown), and there is a block that inserts wait-states into the clock, so that the memory bus that clocks the FLASH memory is less than or equal to 25MHz.

The WaitState block inserts wait states in between HCLK pulses to achieve this goal. The user may insert between 0 and 3 wait states for each FLASH access. The default for the PAC52XX is 3 wait states.

For HCLK values of 25MHz or less, then wait state register may be configured for 0 wait states.

For HCLK values between 25MHz and 50MHz (the maximum HCLK frequency), the wait state register should be configured with at least 1 wait state.

The wait state register may be written at any time during program execution, but is typically configured at initialization. The user may adjust this register via the PAC52XX SDK, or by writing the peripheral registers directly.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

Below are examples that show both firmware techniques.

Configuration Using the PAC52XX SDK

To configure the FLASH memory controller for 1 wait state using the PAC52XX SDK, the following code may be used:

```
#include "pac5xxx_driver_memory.h"

...

pac5xxx_memctl_wait_state(FLASH_WSTATE_25MHZ_LT_HCLK_LTE_50MHZ);

...
```

Note that the definition for the wait states (FLASH_WSTATE_25MHZ_LT_HCLK_LTE_50MHZ) is in the file "pac5xxx_Memory.h".

Configuration Using the PAC52XX Peripheral Register Interface

When configuring the PAC52XX memory controller wait states, the device requires that the FLASHLOCK register be written with a special pattern (0x12345678) before the WAITSTATE register is written. This is to prevent accidental changing of the behavior of the FLASH memory controller.

To configure the FLASH memory controller for 0 wait states using the PAC52XX Peripheral Register Interface, the following code may be used:

```
#include "pac5xxx_Memory.h"

...

PAC5XXX_MEMCTL->FLASHLOCK = FLASH_LOCK_ALLOW_FLASHWSTATE_KEY; // 0x12345678
PAC5XXX_MEMCTL->FLASHWSTATE.VAL = FLASH_WSTATE_HCLK_LTE_25MHZ;

...
```

As soon as the wait state register is successfully written, then the FLASH memory controller will begin executing with the configured wait states when reading FLASH memory.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

ERASING AND WRITING FLASH MEMORY

Some applications need to be able to write FLASH memory during program execution for various tasks. One example would be to change persistent application configuration, which changes after the firmware has been created.

The PAC52XX allows users to change the contents of FLASH memory, and this application note describes a technique to perform this operation.

NOTE: *FLASH memory has a finite number of write cycles, before it becomes unusable. If the application writes FLASH too frequently, then FLASH memory will become unusable and will be unable to execute any firmware previously written to FLASH.*

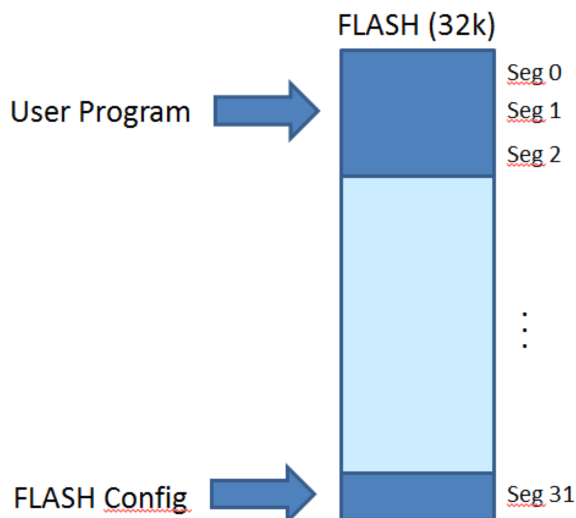
The application designer should take great care to make sure that this operation executes only for configuration changes, and not during normal operation. See the data sheet for the PAC52XX device for more information.

Flash Configuration

When a user builds a program for the PAC52XX device, FLASH memory is used to store the program and constant data. In the PAC52XX, FLASH memory begins at address 0. The user program will start at address 0.

The FLASH memory on the PAC52XX has an available FLASH size of 32k. There are 32 1k FLASH segments.

In this example, the user program will start at address 0, and the FLASH configuration data will be stored in segment 31 (the last segment).



In the PAC52XX, FLASH memory may be erased one segment at a time. So the minimum amount of memory to be erased will be 1k.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

For this example, the FLASH configuration information will be kept in segment 31. The user must take care to make sure the user FLASH program does not get any larger than 31 segments (so it does not over-write the FLASH configuration section).

Writing FLASH Memory

In order to write a segment of FLASH memory, the memory must first be erased. FLASH has to be erased each time before it is written. One segment at a time may be erased, and the segment size is 1k. So, the data to write into FLASH must have its segment erased on 1k boundaries before it may be written.

In order to erase FLASH segments, or to write data to erased FLASH memory, the CPU must be executing instructions out of RAM. You cannot be executing out of FLASH and erasing or writing FLASH memory at the same time.

To instruct the linker to place any function in RAM, you need to do the following.

CooCox:

- Define the symbol "GCC" in CoIde

IAR:

- Define the symbol "IAR" in IAR Embedded Workbench

After the tools have had these symbols properly defined, add the following symbol the beginning of the function signature in the source file: **PAC5XXX_RAMFUNC**. This will tell the linker to place this function in RAM.

For example, the following function would be placed in RAM:

```
PAC5XXX_RAMFUNC void run_this_from_ram(int i)
{
  // This function running in RAM
  ;
}
```

The API functions in the PAC52XX SDK that configure the memory controller must also let the linker know to put these functions in RAM. To do this, the pac5xxx_driver_config.h file must enable the PAC5XXX_DRIVER_MEMORY_RAM symbol, as shown below:

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

```
  @{
*/

#ifndef PAC5XXX_DRIVER_CONFIG_H
#define PAC5XXX_DRIVER_CONFIG_H

//#define ALL_RAM

#ifdef ALL_RAM
#define PAC5XXX_DRIVER_GPIO_RAM          /*!< Link GPIO driver functions in RAM
#define PAC5XXX_DRIVER_SPI_RAM           /*!< Link SPI driver functions in RAM
#define PAC5XXX_DRIVER_SOCBRIDGE_RAM         /*!< Link SOC Bridge driver functions i
#define PAC5XXX_DRIVER_TIMER_RAM           /*!< Link Timer driver functions in RAM
#define PAC5XXX_DRIVER_I2C_RAM           /*!< Link I2C driver functions in RAM
#define PAC5XXX_DRIVER_SYSTEM_RAM      /*!< Link System driver functions in RAM
#define PAC5XXX_DRIVER_UART_RAM        /*!< Link UART driver functions in RAM
#define PAC5XXX_DRIVER_ADC_RAM         /*!< Link ADC driver functions in RAM
#define PAC5XXX_DRIVER_MEMORY_RAM      /*!< Link Memory Controller driver functions
#define PAC5XXX_DRIVER_WATCHDOG_RAM    /*!< Link Watchdog Timer driver functions in
#define PAC5XXX_DRIVER_RTC_RAM         /*!< Link Real-time Clock driver functions ir
#define PAC5XXX_DRIVER_ARM_RAM         /*!< Link ARM driver functions into RAM
#define PAC5XXX_DRIVER_TILE_RAM      /*!< Link Tile driver functions into RAM
#else
//#define PAC5XXX_DRIVER_GPIO_RAM          /*!< Link GPIO driver functions in RAM
//#define PAC5XXX_DRIVER_SPI_RAM           /*!< Link SPI driver functions in RAM
//#define PAC5XXX_DRIVER_SOCBRIDGE_RAM     /*!< Link SOC Bridge driver functions i
//#define PAC5XXX_DRIVER_TIMER_RAM       /*!< Link Timer driver functions in RAM
//#define PAC5XXX_DRIVER_I2C_RAM           /*!< Link I2C driver functions in RAM
//#define PAC5XXX_DRIVER_SYSTEM_RAM    /*!< Link System driver functions in RAM
//#define PAC5XXX_DRIVER_UART_RAM      /*!< Link UART driver functions in RAM
//#define PAC5XXX_DRIVER_ADC_RAM       /*!< Link ADC driver functions in RAM
#define PAC5XXX_DRIVER_MEMORY_RAM      /*!< Link Memory Controller driver functions
//#define PAC5XXX_DRIVER_WATCHDOG_RAM  /*!< Link Watchdog Timer driver functions i
//#define PAC5XXX_DRIVER_RTC_RAM       /*!< Link Real-time Clock driver functions
//#define PAC5XXX_DRIVER_ARM_RAM       /*!< Link ARM driver functions into RAM
//#define PAC5XXX_DRIVER_TILE_RAM          /*!< Link tile driver functions into RA
#endif

/*@}*/ /* end of group PAC5XXX_Driver_Config */
```

To write the configuration information to FLASH, be sure to follow the example above in order to make sure the calling function resides in RAM. An example function that is called to update the configuration information could then look like below.

```
// Call this function when user wants to write new config data into FLASH segment 31
// Be sure to not call frequently. Performing too many writes to FLASH may burn out
FLASh
// and it will become unusable

// Also need to make sure that PAC5XXX_DRIVER_MEMORY_RAM symbol in
pac5xxx_driver_config.h is enabled, to
// compile the memory controller functions into RAM (needed to update FLASH)

// Note that this function *must* be linked into RAM or this operation will not work

PAC5XXX_RAMFUNC void write_config_area(uint16_t config_1, uint8_t config_2, uint8_t
config_3)
{
	FlashConfig* fc = (FlashConfig*)FLASH_CONFIG_ADDR;

	// First, erase segment 31

	pac5xxx_memctl_flash_page_erase(31);

	// Wait for erase to complete

	while (pac5xxx_memctl_page_erase_active() != 0);

	// Prepare to write to erased FLASH segment
```

© 2014 Copyright, Active-Semi International, Inc.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

```
	pac5xxx_memctl_flash_write_prep();

	// Write passed in data to FLASH segment, and write valid key so user knows
data is good
	// Hardware will stall each instruction, while write is taking place

	fc->config_1 = config_1;
	fc->config_2 = config_2;
	fc->config_3 = config_3;
	fc->valid_key = VALID_KEY;


}
```

In this function the first step is to erase the desired FLASH segment by calling the `pac5xxx_memctl_flash_page_erase(31)` function. This will erase FLASH segment 31.

After calling the function to erase a segment of FLASH, the CPU must wait until the erase is complete before beginning to write to FLASH memory. The user may do this by calling the `pac5xxx_memctl_page_erase_active()` function, to test if this operation has completed.

After the erase is complete, then you must prepare FLASH memory to be written, to try to prohibit unwanted writes to FLASH memory by calling the `pac5xxx_memctl_flash_write_prep()` function.

After this, you may write any FLASH in the segment you erased one time. You may not re-write any FLASH without first erasing the segment as shown above.

Reading Configuration Information from FLASH

Once FLASH has been written with configuration information as shown above, the firmware may read the contents of it at any time.

The program above writes a special key into FLASH, to indicate that the FLASH configuration has been written. The application firmware may read this key to determine if the contents of FLASH configuration are OK, as shown below.

```
// Set defaults

uint16_t c1 = 0xFACE;
uint8_t c2 = 0xAA;
uint8_t c3 = 0xBB;

int main(void)
{
	FlashConfig *fc = (FlashConfig*)FLASH_CONFIG_ADDR;

	// See if there is a valid key in FLASH, if so load data from FLASH.
	// If the key is not valid, just use default values

	if (fc->valid_key == VALID_KEY)
	{
		c1 = fc->config_1;
```


No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

```
        c2 = fc->config_2;
        c3 = fc->config_3;
    }


    while (1);

}
```

In this function, the user program (which may be in FLASH or RAM) checks to see if the valid key is present in configuration FLASH memory. If it is, it uses the configuration values from FLASH.

If the contents of configuration FLASH are not valid, then the default values can be used, as shown above.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

FLASH BUFFERED WRITES

The ARM Cortex-M0 may be configured to run at a faster clock frequency than the FLASH memory is capable of running at. If the ARM Cortex-M0 writes a location in FLASH memory, then the MCU will stall until the operation completes. This prevents the MCU from executing any other instructions until the FLASH write operation completes.

This stall may be unacceptably large, depending on the program requirements. For example, in the PAC5220 the FLASH write time is 20µs.

The PAC52XX family of controllers have a feature called "buffered writes" that allow the MCU to post a write operation, and regain control over the CPU to execute other instructions while waiting for the write operation to complete.

FLASH Buffered Writes allow the user to post a 16-bit write operation if using this feature.

To use this feature, the user will need to know the following information:

- FLASH page containing the data to write
- Relative memory address of data to write

Where:

FLASH page = (Target Address – Page Size [1024])

Address = (Target Address – (FLASH page * Page Size [1024])) / 2

For example, if the user wishes to write the data 0xFEED to the address 0x0000 0438 the values for FLASH page and Address would be as follows:

FLASH page = (0x0000 0438 – 0x400) = 0

Address = (0x0000 0438 – (0 * 0x400) / 2) = 0x0000 0438

To execute a buffered write to this address, the user may use either the PAC52XX SDK or PAC52XX Peripheral Register Interface. Both types of examples are shown below.

Buffered Write using PAC52XX SDK

To write the value 0xFEED to the address 0x000 0438 using Buffered Writes via the PAC52XX SDK, the user may use the code shown below.

Note that the user should first check to make sure there is not an active buffered write (in case this can happen), then write the data by calling the following function:

```
#include "pac5xxx_driver_memory.h"

...

while (pac5xxx_memctl_buffered_write_active() != 0);

pac5xxx_memctl_buffered_write(0, 0x0438, 0xFEED);
```


No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

Buffered Write using PAC52XX Peripheral Register Interface

To write the value 0xFEED to the address 0x000 0438 using Buffered Writes via the PAC52XX Peripheral Register Interface, the user may use the code shown below.

Note that the user should first check to make sure there is not an active buffered write (in case this can happen), then write the data by the following code:

```
#include "pac5xxx_Memory.h"

...

// Test to make sure no buffered write is active

while (PAC5XXX_MEMCTL->FLASHCTL.WRITE != 0);

// Write the FLASH buffered write key, page, address and data

PAC5XXX_MEMCTL->FLASHBWRITE = FLASH_LOCK_ALLOW_BWRITE_KEY;
PAC5XXX_MEMCTL->FLASHBWRITEDATA.PAGE = 0;
PAC5XXX_MEMCTL->FLASHBWRITEDATA.ADDRESS = 0x0438;
PAC5XXX_MEMCTL->FLASHBWRITEDATA.DATA = 0xFEED;
```

In this example, as soon as the DATA register is written with the value, the write will be posted and the MCU will be able to continue executing instructions.

Before the MCU performs another buffered write to FLASH, it should verify that no write operation is in progress by checking the state of the FLASHCTL.WRITE bit, as shown above.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

SWD Debug Access Protection

The PAC52XX Memory controller also contains a feature that allows the SWD peripheral to be disabled. This feature may be used by applications that wish to not allow access to FLASH memory via the SWD port, for code protection.

When this feature is enabled, it activates a FUSE which permanently disables any read or write access to the SWD port.

NOTE: BE VERY CAREFUL IMPLEMENTING THIS FEATURE, ONCE ENABLED IT IS NOT REVERSIBLE.

To activate this feature, the user must write a special pattern to the FLASHLOCK register (0xF983 62AB) and then write a special pattern to the SWDACCESS register to blow the fuse (0x6969).

Below are examples that show both firmware techniques.

SWD Debug Access Protection Code

The user may only activate this feature via the PAC52XX peripheral registers directly.

To activate SWD Debug Access Protection, the following code may be used:

```
#include "pac5xxx_Memory.h"

...

PAC5XXX_MEMCTL->FLASHLOCK = 0xF98362AB;
PAC5XXX_MEMCTL->FLASHWSTATE.VAL = 0x6969;

...
```

As soon as the FLASHWSTATE register is written as shown above, the SWD port will no longer be usable for reading or writing.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent

ABOUT ACTIVE-SEMI

Active-Semi, Inc. headquartered in Dallas, TX is a leading innovative semiconductor company with proven power management, analog and mixed-signal products for end-applications that require power conversion (AC/DC, DC/DC, DC/AC, PFC, etc.), motor drivers and control and LED drivers and control along with ARM microcontroller for system development.

Active-Semi's latest family of Power Application Controller (PAC)™ ICs offer high-level of integration with 32-bit ARM Cortex M0, along with configurable power management peripherals, configurable analog front-end with high-precision, high-speed data converters, single-ended and differential PGAs, integrated low-voltage and high-voltage gate drives. PAC IC offers unprecedented flexibility and ease in the systems design of various end-applications such as Wireless Power Transmitters, Motor drives, UPS, Solar Inverters and LED lighting, etc. that require a microcontroller, power conversion, analog sensing, high-voltage gate drives, open-drain outputs, analog & digital general purpose IO, as well as support for wired and wireless communication. More information and samples can be obtained from http://www.active-semi.com or by emailing marketing@active-semi.com

Active-Semi shipped its 1 Billionth IC in 2012, and has over 120 in patents awarded and pending approval.

LEGAL INFORMATION & DISCLAIMER

Copyright © 2012-2013 Active-Semi, Inc. All rights reserved. All information provided in this document is subject to legal disclaimers.

Active-Semi reserves the right to modify its products, circuitry or product specifications without notice. Active-Semi products are not intended, designed, warranted or authorized for use as critical components in life-support, life-critical or safety-critical devices, systems, or equipment, nor in applications where failure or malfunction of any Active-Semi product can reasonably be expected to result in personal injury, death or severe property or environmental damage. Active-Semi accepts no liability for inclusion and/or use of its products in such equipment or applications. Active-Semi does not assume any liability arising out of the use of any product, circuit, or any information described in this document. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of Active-Semi or others. Active-Semi assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein. Customers should evaluate each product to make sure that it is suitable for their applications. Customers are responsible for the design, testing, and operation of their applications and products using Active-Semi products. Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. All products are sold subject to Active-Semi's terms and conditions of sale supplied at the time of order acknowledgment. Exportation of any Active-Semi product may be subject to export control laws.

Active-Semi™, Active-Semi logo, Solutions for Sustainability™, Power Application Controller™, Micro Application Controller™, Multi-Mode Power Manager™, Configurable Analog Front End™, and Application Specific Power Drivers™ are trademarks of Active-Semi, I. ARM® is a registered trademark and Cortex™ is a trademark of ARM Limited. All referenced brands and trademarks are the property of their respective owners.

No portion of this document may be reproduced or reused in any form without Active-Semi's prior written consent