ambiq

USER'S GUIDE

Apollo3 GPIO Implementation

Ultra-Low Power Apollo MCU Family

A-MCUAP3-UGGA01EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
0.1	March 30, 2018	Initial version
0.2	April 3, 2018	Interrupts and r/w macros
1.0	March 6, 2022	Updated document template

Reference Documents

Document ID	Description

Table of Contents

1. Introduction 5

2. Overview of the GPIO Functions 6

2.1 am_hal_gpio_pinconfig() 6
2.2 am_hal_gpio_state_read() 8
2.3 am_hal_gpio_state_write() 8
2.4 Interrupt Functions 9
2.4.1 am_hal_gpio_interrupt_enable(uint64_t ui64InterruptMask) 9
2.4.2 am_hal_gpio_interrupt_disable(uint64_t ui64InterruptMask) 9
2.4.3 am_hal_gpio_interrupt_clear(uint64_t ui64InterruptMask) 9
2.4.4 am_hal_gpio_interrupt_status_get(bool bEnabledOnly, uint64_t *pui64IntStatus) 9
2.4.5 am_hal_gpio_interrupt_status_get(bool bEnabledOnly, uint64_t *pui64IntStatus) 9
2.5 GPIO Read and Write Macros 10
2.5.1 am_hal_gpio_input_read(n) 10
2.5.2 am_hal_gpio_output_read(n) 10
2.5.3 am_hal_gpio_enable_read(n) 10
2.5.4 am_hal_gpio_output_clear(n) 11
2.5.5 am_hal_gpio_output_set(n) 11
2.5.6 am_hal_gpio_output_toggle(n) 11
2.5.7 am_hal_gpio_output_tristate_disable(n) 11
2.5.8 am_hal_gpio_output_tristate_enable(n) 11
2.5.9 am_hal_gpio_output_tristate_toggle(n) 11
2.5.10 am_hal_gpio_interrupt_register(uint32_t ui32GPIONumber, am_hal_gpio_handler_t pdnHandler) 12
2.5.11 am_hal_gpio_interrupt_service(uint64_t ui64Status) 12

3. Creating a BSP Pin List 13

3.1 Create the bsp_pins.src file 13

SECTION

1

Introduction

The GPIO HAL implementation for Apollo3 has changed significantly from previous Apollo and Apollo2 implementations. The hardware design for Apollo3 GPIO remains very similar to the previous products with a few new features added. The new GPIO features, however, do add some complexity that the new GPIO HAL makes more manageable for the user.

The previous software implementation was heavily dependent on macros for configuration and usage, which caused confusion to end users, especially with pin configuration, and contributed to code size due to the inline coding. The new implementation abstracts most of the configuration into the HAL with the caller supplying a single word containing all pin configuration parameters in a single 32-bit word defined by a standard C bit field structure.

SECTION

2

Overview of the GPIO Functions

2.1 am_hal_gpio_pinconfig()

This function is called when configuring a given pad for its ultimate function. The specified parameters (**bfGpioCfg**) are checked for compatibility with the specified pin. Any configuration or parameter errors result in an error return.

The prototype is **`am_hal_gpio_pinconfig(uint32_t ui32Pin, am_hal_gpio_pincfg_t bfGpioCfg)`**.

The **ui32Pin** parameter is simply the pin number to be configured.

`am_hal_gpio_pincfg_t` is a bit field structure containing the following members:

Table 2-1: am_hal_gpio_pincfg_t Bit Field Structure

Field	Description
uFuncSel	This is a value from 0-7 which will usually come from **`am_hal_pin.h`**.
ePullup	Many pads can supply a pullup resistor. For those that do, this member defines the value of that pullup. It is one of the following enumerations: AM_HAL_GPIO_PIN_PULLUP_NONE AM_HAL_GPIO_PIN_PULLUP_1_5K AM_HAL_GPIO_PIN_PULLUP_6K
eGPOutcfg	This member is generally used when defining a pad as a GPIO output and defines the output type. It is one of the following enumerations: AM_HAL_GPIO_PIN_OUTCFG_DISABLE AM_HAL_GPIO_PIN_OUTCFG_PUSHPULL AM_HAL_GPIO_PIN_OUTCFG_OPENDRAIN AM_HAL_GPIO_PIN_OUTCFG_TRISTATE

Table 2-1: am_hal_gpio_pincfg_t Bit Field Structure *(Continued)*

Field	Description
eDriveStrength	For output configurations, many pads can be configured with various drive strengths. For those that do, this member defines that and will be one of the following enumerations: AM_HAL_GPIO_PIN_DRIVESTRENGTH_2MA AM_HAL_GPIO_PIN_DRIVESTRENGTH_4MA AM_HAL_GPIO_PIN_DRIVESTRENGTH_8MA AM_HAL_GPIO_PIN_DRIVESTRENGTH_12MA
eGPInput	This member is generally used when defining a pad as a GPIO input and defines the input type. It is one of the following enumerations: AM_HAL_GPIO_PIN_INPUT_NONE AM_HAL_GPIO_PIN_INPUT_ENABLE
eGPRdZero	This member is generally used when defining a pad as a GPIO input and defines whether the pin value can be read or if it always reads as zero. It is one of the following enumerations: AM_HAL_GPIO_PIN_RDZERO_READPIN AM_HAL_GPIO_PIN_RDZERO_ZERO
eIntDir	This member is used when interrupts are to be enabled for a pad. It is one of the following enumerations: AM_HAL_GPIO_PIN_INTDIR_LO2HI AM_HAL_GPIO_PIN_INTDIR_HI2LO AM_HAL_GPIO_PIN_INTDIR_NONE AM_HAL_GPIO_PIN_INTDIR_BOTH
ePowerSw	A select number of pins can be configured to source or sink current (see datasheet for which pins support these functions). For pins that support it, it is one of the following enumerations: AM_HAL_GPIO_PIN_POWERSW_NONE AM_HAL_GPIO_PIN_POWERSW_VDD AM_HAL_GPIO_PIN_POWERSW_VSS
uIOMnum	This member is used when a pad is defined to be a chip enable and designates the IO Master number (0-5) or MSPI (6) that the CE is to be used for. Most pads can be configured as a chip enable with each pad supporting 4 combinations of IOM/MSPI and channel numbers. See the datasheet for a table of these combinations. This member is always a value of 0-5 or 6.

Table 2-1: am_hal_gpio_pincfg_t Bit Field Structure *(Continued)*

Field	Description
uNCE	This member is used when a pad is defined to be a chip enable and is used in conjunction with uIOMnum to define the CE number for a particular SPI device. It is always a value of 0-3.
eCEpol	This member is used when a pad is defined to be a chip enable and specifies the polarity of the CE enable. It is one of the following enumerations: AM_HAL_GPIO_PIN_CEPOL_ACTIVELOW AM_HAL_GPIO_PIN_CEPOL_ACTIVEHIGH

2.2 am_hal_gpio_state_read()

The prototype is **`am_hal_gpio_state_read(uint32_t ui32Pin, am_hal_gpio_read_type_e eReadType, uint32_t *pui32ReadState).`**

ui32Pin is the pin number to be read.

eReadType is one of the following enumerations:

AM_HAL_GPIO_INPUT_READ
AM_HAL_GPIO_OUTPUT_READ
AM_HAL_GPIO_ENABLE_READ

pui32ReadState is a pointer to the variable to receive the read value of the pin.

2.3 am_hal_gpio_state_write()

This function is used for writing GPIO values.

The prototype is **`am_hal_gpio_state_write(uint32_t ui32Pin, am_hal_gpio_write_type_e eWriteType).`**

ui32Pin is the pin number to be read.

eWriteType is one of the following enumerations:

AM_HAL_GPIO_OUTPUT_SET
AM_HAL_GPIO_OUTPUT_CLEAR
AM_HAL_GPIO_OUTPUT_TOGGLE
AM_HAL_GPIO_OUTPUT_TRISTATE_ENABLE
AM_HAL_GPIO_OUTPUT_TRISTATE_DISABLE

2.4 Interrupt Functions

As with other peripherals, pins configured as GPIOs can be configured to provide interrupts. The HAL provides several functions to support this functionality.

2.4.1 am_hal_gpio_interrupt_enable(uint64_t ui64InterruptMask)

This function enables the given interrupt(s). Only bits 0-49 are valid in the mask.

2.4.2 am_hal_gpio_interrupt_disable(uint64_t ui64InterruptMask)

This function disables the given interrupt(s). Only bits 0-49 are valid in the mask.

2.4.3 am_hal_gpio_interrupt_clear(uint64_t ui64InterruptMask)

This function clears the given interrupt(s). Only bits 0-49 are valid in the mask. This function is often used in conjunction with **`am_hal_gpio_interrupt_status_get()`**, with the returned IntStatus used as the input to this function.

2.4.4 am_hal_gpio_interrupt_status_get(bool bEnabledOnly, uint64_t *pui64IntStatus)

This function returns the current interrupt status.

The API function returns **`AM_HAL_STATUS_SUCCESS`** if successful, otherwise it returns a fail code.

2.4.5 am_hal_gpio_interrupt_status_get(bool bEnabledOnly, uint64_t *pui64IntStatus)

This function returns the current interrupt status. It can return the status of every interrupt (**bEnabledOnly=false**) or the status of only those that are enabled (**bEnabledOnly=true**). The 64bit variable pointed to be pui64IntStatus contains the return status.

The API function returns **`AM_HAL_STATUS_SUCCESS`** if successful, otherwise it returns a fail code.

2.5 GPIO Read and Write Macros

While the primary read and write functions will suffice for virtually all applications, there may be situations where minimal response time is required. To support these situations a set of macros are provided which provide minimal inline code for accessing GPIOs.

Advantages to usage of these macros include faster GPIO read or write access times, no function call overhead, and simple read return values.

Drawbacks to usage of these macros include no error checking, larger resultant code size, no guaranteed atomicity, and risk to general safety.

The "_read" macros are counterparts to the enumerations used for the **`am_hal_gpio_state_read()`** function.

Likewise, the **`"_set, _clear, _toggle)`** macros are counterparts to the enumerations used for the **`am_hal_gpio_state_write()`** function.

The macros and their usage are outlined here.

2.5.1 am_hal_gpio_input_read(n)

Counterpart to **`am_hal_gpio_state_read(AM_HAL_GPIO_INPUT_READ)`**.

This macro is used for reading the value on a pin given by 'n' and returns the value as either a 0 or 1. It assumes the pin has been configured for reading. The macro effectively reads the appropriate value from the GPIO RDA or GPIO RDB registers.

2.5.2 am_hal_gpio_output_read(n)

Counterpart to **`am_hal_gpio_state_read(AM_HAL_GPIO_OUTPUT_READ)`**.

This macro is used for reading the value that was most recently written to the GPIO WTA or GPIO WTB register to be output to the pin (the pin given by 'n') and returns the value as either a 0 or 1.

2.5.3 am_hal_gpio_enable_read(n)

Counterpart to **`am_hal_gpio_state_read(AM_HAL_GPIO_ENABLE_READ)`**.

This macro is used for reading the value that was most recently written to the GPIO Enable (ENA or ENB) register. The enable is typically used when the pin is configured as GPIO and tri-state output and enables the output. The pin enable to be read is designated by 'n', and the macro returns the value as either a 0 or 1.

2.5.4 am_hal_gpio_output_clear(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_OUTPUT_CLEAR)`**.

This macro is used for writing a 0 to output to the pin designated by 'n'. There is no return value.

2.5.5 am_hal_gpio_output_set(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_OUTPUT_SET)`**.

This macro is used for writing a 1 to output to the pin designated by 'n'. There is no return value.

2.5.6 am_hal_gpio_output_toggle(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_OUTPUT_TOGGLE)`**.

This macro is used to toggle the current value being output to the pin designated by 'n'. There is no return value.

2.5.7 am_hal_gpio_output_tristate_disable(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_TRISTATE_DISABLE)`**.

This macro is used to disable the output enable on the pin designated by 'n'. There is no return value.

2.5.8 am_hal_gpio_output_tristate_enable(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_TRISTATE_ENABLE)`**.

This macro is used to enable the output enable on the pin designated by 'n'. There is no return value.

2.5.9 am_hal_gpio_output_tristate_toggle(n)

Counterpart to **`am_hal_gpio_state_write(AM_HAL_GPIO_TRISTATE_TOGGLE)`**.

This macro is used to toggle the current value of the output enable on the pin designated by 'n'. There is no return value.

2.5.10 am_hal_gpio_interrupt_register(uint32_t ui32GPIONumber, am_hal_gpio_handler_t pdnHandler)

This function is call by the application for registering specific handlers to specific GPIO interrupts.

The API function returns **`AM_HAL_STATUS_SUCCESS`** if successful, otherwise it returns a fail code.

2.5.11 am_hal_gpio_interrupt_service(uint64_t ui64Status)

This function is an overall service routine for GPIO interrupts. It is called by am_gpio_isr(), which also calls **`am_hal_gpio_interrupt_status_get()`** to use as an input parameter to this function.

The general usage is that the application calls **`am_hal_gpio_interrupt_register()`** to register a callback routine that this routine will call when the registered interrupt occurs. The application also supplies the main handler, **`am_gpio_isr()`**.

The API function returns **`AM_HAL_STATUS_SUCCESS`** if successful, otherwise it returns a fail code.

SECTION

3

Creating a BSP Pin List

3.1 Create the bsp_pins.src file

The file **bsp_pins.src** is a simple text file containing a list of keywords and values. It is subsequently read in by a Python script and generates two files: **am_bsp_pins.c** and **am_bsp_pins.h**. These two C files contain each of the pins bit field structures that are passed along to **am_hal_gpio_pinconfig()**.

> **NOTE:** The .src file should contain no tab characters (only spaces).

Also, indentation is important. A tab indentation of 4 spaces is recommended.

Each pin entry takes the form:

```
pin
    name            = UART_TX
    desc            = This pin is the COM_UART transmit pin.
    pinnum          = 35
    func_sel        = AM_HAL_PIN_35_UART1TX
    drvstrength     = 2
```

While there are about a dozen keywords (parameters) available, only the parameters required to define a pin need be included in any particular definition.

The keywords used in the file are shown in Table 3-1 on page 14.

Table 3-1: Keywords used in bsp_pins.src file

Keyword	Description
name	The name to be used for the pin. This name will be used as a base for generating defines. Each pin name must be unique.
desc	Optional: A description, if provided, will appear in the generated header file.
funcsel	A value 0-7, or the equivalent **AM_HAL_PIN_nn_xxxx** macro from am_hal_pin.h. Note that the **AM_HAL_PIN_nn_xxxx** nomenclature is preferred.
pinnum	The pin number for the pin being defined (0-49).
drvstrength	One of: 2, 4, 8, or 12. If not provided, 2 is default.
GPOutCfg	Typically used if the pin is being defined as GPIO (funcsel=3). One of: disable, pushpull, opendrain, tristate. * Also acceptable is a value 0-3, or a define.
GPinput	Only used if the pin is being defined as GPIO (funcsel=3). One of: true, false.
GPRdZero	One of readpin, zero (or true or false).
intdir	One of: none, lo2hi, hi2lo, either. Note: Does not enable any interrupt. Only configures the direction for when it is enabled.
pullup	One of: none, 1_5K, 6K, 12K, 24K. Also acceptable is a define (e.g., **AM_HAL_GPIO_PIN_PULLUP_1_5K**).
PowerSw	One of: VDD or VSS. Also acceptable is a define (e.g. **AM_HAL_GPIO_PIN_POWERSW_VDD**).

The following 3 parameters only apply when the pin is being defined as a chip enable (e.g., a CE for a SPI or MSPI device).

Table 3-2: Parameters when the pin is defined as a chip enable

Keyword	Description
IOMnum	The IOM number pertaining to the CE. 0-5 for SPI, 6 for MSPI.
CENum	A value from 0-3 representing the chip enable channel number. Results in a C define of the form: #define AM_BSP_<name>_CHNL <CEnum>
CEpol	Designates the chip enable polarity, active high or active low. One of: LOW (default) or HIGH.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 (512) 879-2850

A-MCUAP3-UGGA01EN v1.0
March 2022