GPIO LLD User Guide

TABLE OF CONTENTS

1 INTRODUCTION 1

2 KEY FEATURES 1

3 HARDWARE SUPPORT (EVM/SOCS) 1

4 DESIGN OVERVIEW 1

4.1 PERIPHERAL DEVICE DRIVER 2
4.2 DEVICE SPECIFIC MODULE LAYER 2
4.3 APPLICATION CODE 2
4.4 OSAL 2
4.5 CSL REGISTER LAYER 2

5 MODES OF OPERATION 3

5.1 INPUT OR OUTPUT 3
5.2 INTERRUPT SUPPORT 3

6 DRIVER CONFIGURATION 3

6.1 BOARD SPECIFIC CONFIGURATION 3
6.2 GPIO CONFIGURATION STRUCTURE 3
6.3 APIS 3
6.4 USAGE 4
6.5 API CALLING SEQUENCE 4
6.6 FLOW CHART 4

7 EXAMPLES 6

7.1 LED TOGGLING: 6
7.1.1 *Building the examples:* *6*
7.1.2 *Running the examples* *6*
7.1.2.1 AM572x IDK EVM 6
7.1.2.2 AM571x IDK EVM 6
7.1.2.3 AM572x GP EVM 6
7.1.3 *Supported platforms:* *7*

8 TEST 7

8.1.1 *Building the examples:* *7*
8.1.2 *Running the examples* *7*
8.1.2.1 AM572x IDK EVM 7
8.1.2.2 AM571x IDK EVM 7
8.1.2.3 AM572x GP EVM 8
8.1.3 *Supported platforms:* *8*

9 MIGRATION GUIDE 8

10 BENCHMARKING 8

1 Introduction

The GPIO module allows you to manage General Purpose I/O pins and ports via simple and portable APIs.
Because of its simplicity, the GPIO driver does not follow the model of other PDK drivers in which driver application interface has handle based approach. GPIO driver implementation will be based on pin based approach.

2 Key Features

- Pins can be configured as either input or Output.
- Interrupts can be generated on the each gpio pin.

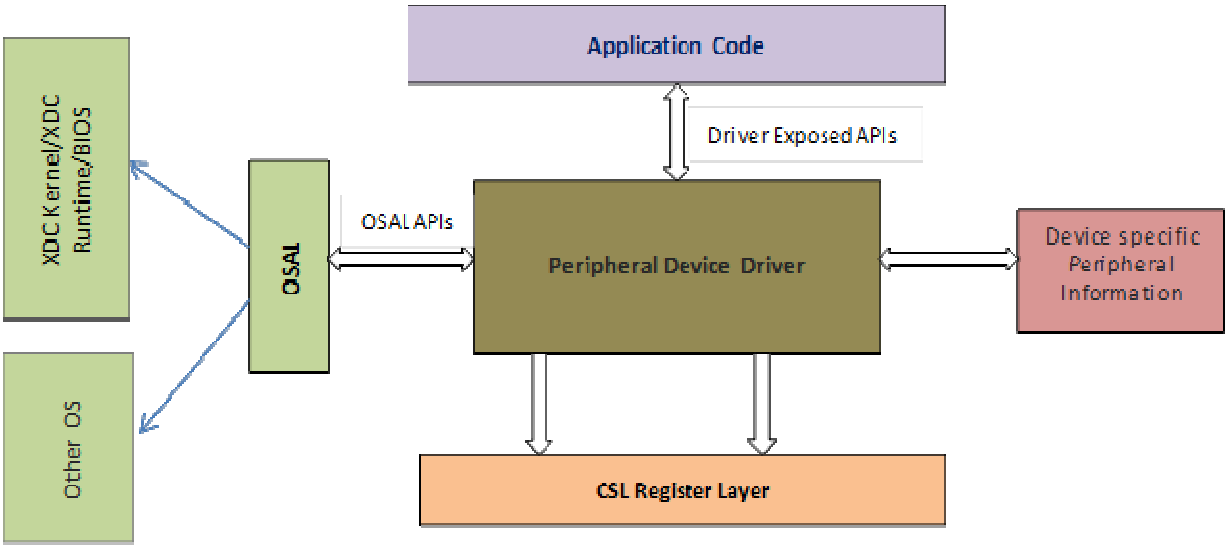
3 Hardware Support (EVM/SoCs)

Board	SoC	Cores
AM572x IDK EVM	AM572x	A15 & C66X
AM572x GP EVM	AM572x	A15 & C66X
AM571x IDK EVM	AM571x	A15 & C66X

4 Design Overview

The GPIO driver provides a well-defined API layer which allows applications to use the GPIO peripheral to control different GPIO Pins.

The below figure which shows the GPIO Driver architecture.



The figure illustrates the following key components:-

4.1 Peripheral device driver

This is the core GPIO device driver. The device driver exposes a set of well-defined APIs which are used by the application layer. The driver also exposes a set of well-defined OS abstraction APIs which will ensure that the driver is OS independent and portable. The driver uses the CSL register layer for MMR accesses.

4.2 Device specific module layer

This layer implements a well-defined interface which allows the core GPIO device driver to be ported to any device which has the same GPIO IP block. This layer may change for every device.

4.3 Application Code

This is the user of the driver and its interface through the well-defined APIs set. Application uses the driver APIs to send and receive data via the GPIO peripheral.

4.4 OSAL

The driver is OS independent and exposes all the operating system callouts via this OSAL layer.

4.5 CSL Register Layer

The GPIO driver uses the CSL GPIO functional layer to program the device IP by accessing the MMR (Memory Mapped Registers).

5 Modes of Operation

GPIO driver provides the following modes of operations.

5.1 Input or Output

Each gpio pin can be configured as either input or output. If it is configured as an output then the pin level can be written on the gpio pin.

5.2 Interrupt support

Each gpio pin can be configured to generate interrupts based on the interrupt event type configured. To generate the interrupt, gpio pin has to be of type input pin.

6 Driver Configuration

6.1 Board Specific Configuration

All the board specific configurations like enabling the clock and pin-mux before calling any of the driver APIs. The GPIO module allows you to manage General Purpose I/O pins via simple and portable APIs. GPIO pin behavior can be configured completely statically, or dynamically defined at runtime.

The application is required to supply a device specific GPIOXXX_Config structure to the module. This structure communicates to the GPIO module how to configure the pins that will be used by the application. The application is required to call GPIO_init (). This function will initialize all the GPIO pins defined in the GPIO_PinConfig table to the configurations specified. Once that is completed the other APIs can be used to access the pins.

6.2 GPIO Configuration Structure

The GPIO_soc.c file contains the declaration of the hardware attributes corresponding to the GPIO peripheral. These hardware attributes will include base address, interrupt number etc.

6.3 APIs

In order to use the GPIO module APIs, the GPIO.h header file should be included in an application as follows:

#include <ti/drv/gpio/GPIO.h>

The following are the GPIO APIs:

- **GPIO_init** The pins defined in the application-provided *GPIOXXX_config* structure are initialized accordingly
- **GPIO_read()** Reads the value of a GPIO pin.
- **GPIO_write ()** Writes the value to a GPIO pin

- **GPIO_toggle ()** Toggles the current state of a GPIO
- **GPIO_setConfig ()** Dynamically configure a gpio pin to a device specific setting. For many applications, the pin configurations provided in the static GPIO_PinConfig array is sufficient.
- **GPIO_enableInt()** Enable a GPIO pin interrupt
- **GPIO_disableInt()** Disable a GPIO pin interrupt
- **GPIO_clearInt()** Clear a GPIO pin interrupt flag

6.4 Usage

Once the GPIO_init() function has been called, the other GPIO APIs functions can be called. For example, LEDs can be switched on as follows:

```
GPIO_write(Board_LED0, Board_LED_ON);
GPIO_write(Board_LED1, Board_LED_ON);
GPIO_write(Board_LED2, Board_LED_ON);
```

For GPIO interrupts, once the GPIO_setupCallbacks() has been called for a port's GPIO_Callback structure, that port may be enabled for interrupts as follows:

```
GPIO_setupCallbacks(&Board_gpioCallbacks0)
GPIO_enableInt(Board_BUTTON0, GPIO_INT_FALLING);
GPIO_enableInt(Board_BUTTON1, GPIO_INT_RISING);
```

Interrupts may be configured to occur on rising edges, falling edges, both edges, a high level, or a low level.

6.5 API Calling Sequence

The below sequence indicates the calling sequence of GPIO driver APIs for a use case of toggling the LEDs.

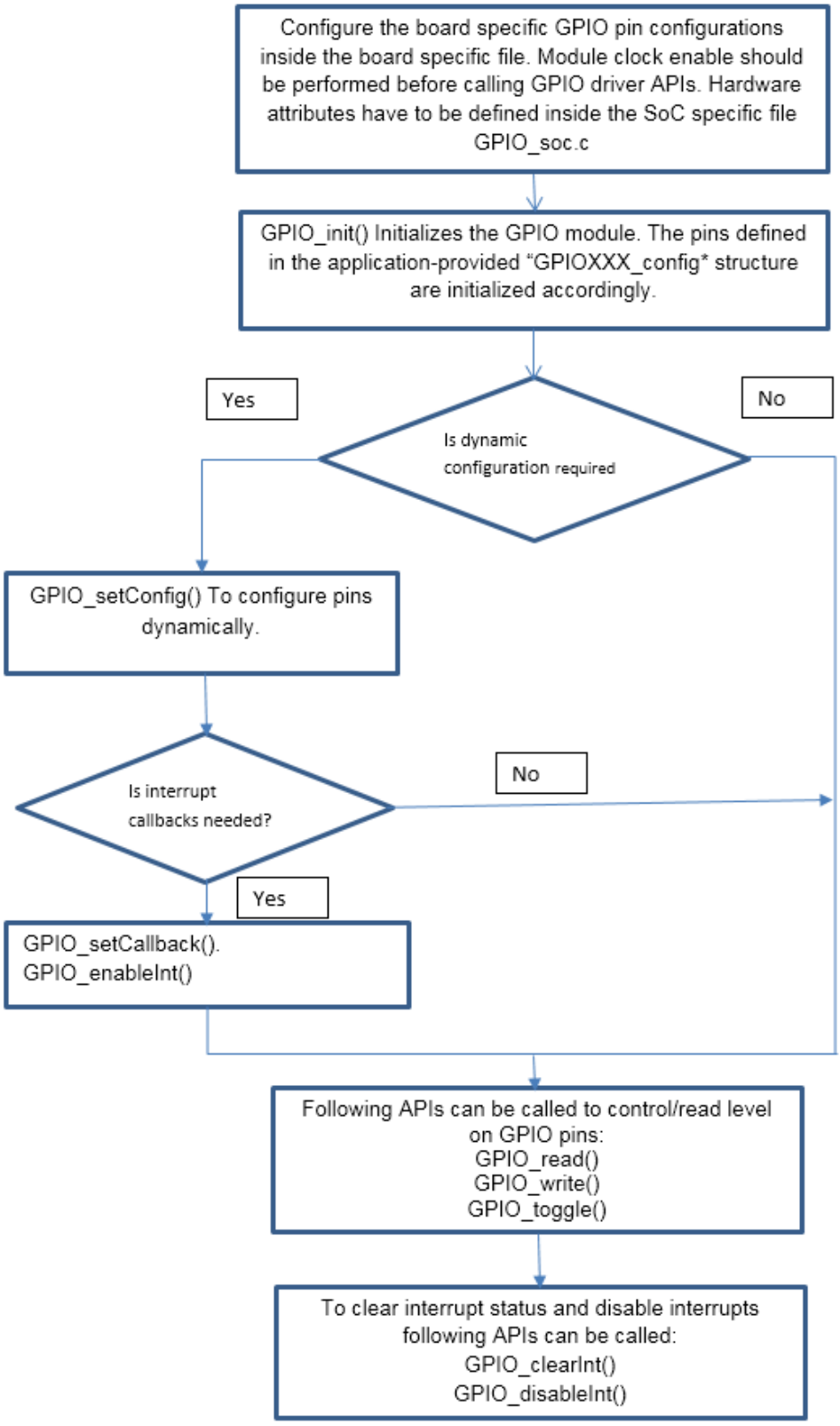
```
GPIO_init();

While(1)
{
   GPIO_write(Board_LED1, GPIO_PIN_VAL_HIGH);
   Delay();

   GPIO_write(Board_LED1, GPIO_PIN_VAL_LOW);
   Delay();
}
```

6.6 Flow chart

API flow path of GPIO driver is as shown in fig.


Configure the board specific GPIO pin configurations inside the board specific file. Module clock enable should be performed before calling GPIO driver APIs. Hardware attributes have to be defined inside the SoC specific file GPIO_soc.c
GPIO_init() Initializes the GPIO module. The pins defined in the application-provided "GPIOXXX_config* structure are initialized accordingly.
Is dynamic configuration required
Yes
No
GPIO_setConfig() To configure pins dynamically.
Is interrupt callbacks needed?
No
Yes
GPIO_setCallback().
GPIO_enableInt()
Following APIs can be called to control/read level on GPIO pins:
GPIO_read()
GPIO_write()
GPIO_toggle()
To clear interrupt status and disable interrupts following APIs can be called:
GPIO_clearInt()
GPIO_disableInt()

7 Examples

Following are the examples of supported for the GPIO Driver

7.1 LED toggling:

This application will toggle the led. The led toggling will be done inside a callback function, which will be called by Interrupt Service Routine. Interrupts are triggered manually and no external source is used to trigger interrupts.

7.1.1 Building the examples:

Following are list of GPIO projects which will reside the following location "packages/MyExampleProjects"

GPIO_LedBlink_AM571X_armExampleProject
GPIO_LedBlink_AM571X_c66xExampleProject
GPIO_LedBlink_AM572X_armExampleProject
GPIO_LedBlink_AM572X_c66xExampleProject
GPIO_LedBlink_AM572X_GpEvm_armExampleProject
GPIO_LedBlink_AM572X_GpEvm_c66xExampleProject

These projects have to be imported in CCS and have to be built. The ".out" files corresponding to each project will be generated after successfully compiling the projects.

7.1.2 Running the examples

The ".out" have to be loaded and executed. If the project is executed successfully, then one of the on board LED will blink.

7.1.2.1 AM572x IDK EVM

Status LED1 Yellow will blink.

7.1.2.2 AM571x IDK EVM

Industrial LED3 Red will blink

7.1.2.3 AM572x GP EVM

User1 LED will blink

7.1.3 Supported platforms:

AM572x GP EVM
AM572x IDK EVM
AM571x IDK EVM

8 Test

This application will toggle the led. The led toggling will be done inside a callback function, which will be called by Interrupt Service Routine. Interrupts are triggered manually and no external source is used to trigger interrupts.

8.1.1 Building the examples:

Following are list of GPIO projects which will reside the following location "packages/MyExampleProjects"

GPIO_LedBlink_AM571X_armTestProject
GPIO_LedBlink_AM571X_c66xTestProject
GPIO_LedBlink_AM572X_armTestProject
GPIO_LedBlink_AM572X_c66xTestProject
GPIO_LedBlink_AM572X_GpEvm_armTestProject
GPIO_LedBlink_AM572X_GpEvm_c66xTestProject

These projects have to be imported in CCS and have to be built. The ".out" files corresponding to each project will be generated after successfully compiling the projects.

8.1.2 Running the examples

The ".out" have to be loaded and executed. If the project is executed successfully, then one of the on board LED will blink.

8.1.2.1 AM572x IDK EVM

Status LED1 Yellow will blink.

8.1.2.2 AM571x IDK EVM

Industrial LED3 Red will blink

8.1.2.3 AM572x GP EVM

User1 LED will blink

8.1.3 Supported platforms:

AM572x GP EVM
AM572x IDK EVM
AM571x IDK EVM

9 Migration Guide

The driver supports multiple SoCs, Cores and different IP versions. High level APIs names will be same and internal implementation will differ for different versions of IPs. Separate source files will present for different versions of IPs and the files are which are not relevant to any particular IP version will be compiled out during library generation.

Users who are using the low level APIs (Device abstraction APIs: which perform hardware register read/write) have to use the high level APIs which are described in the section 6.3.

10 Benchmarking

Code size for library in bytes:

Initialized data : 21

Code: 5152