AXI GPIO

Zynq
Vivado 2015.2 Version

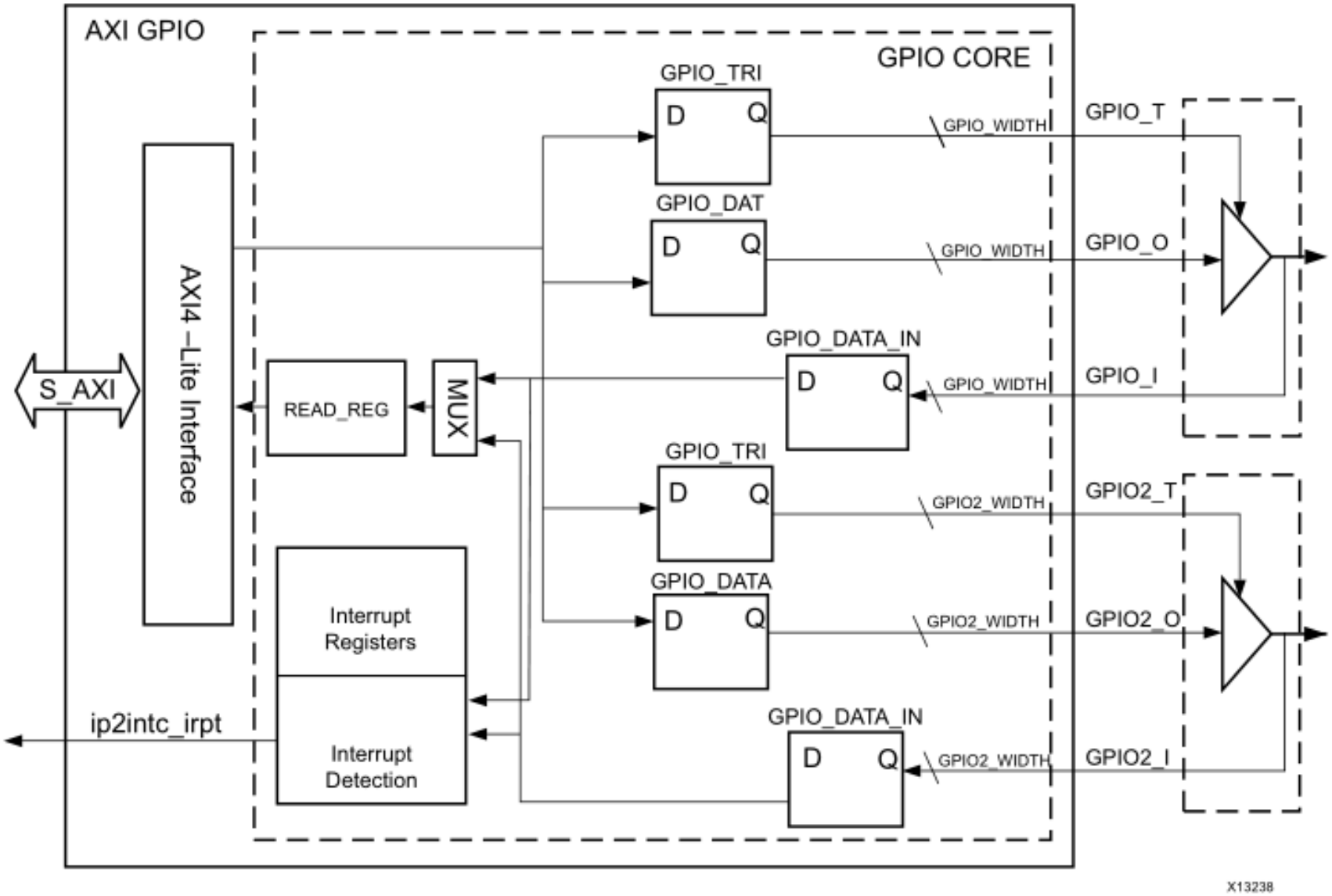
Outline

- Introducing AXI GPIO
- Configure GPIO IP Instance
- GPIO Programming Sequence
- SDK Application Development Flow
- Application Project Structure
- AXI GPIO Driver API
- References

AXI GPIO

- The AXI GPIO design provides **a general purpose** input/output interface to an **AXI4-Lite** interface.
- The AXI GPIO can be configured as either a **single** or a **dual**-channel device.
- The **width** of each channel is independently configurable: **32**-bit AXI4-Lite slave.
- The ports are configured dynamically for input or output by enabling or disabling the 3-state buffer.
- The channels can be configured to generate an interrupt when a transition on any of their inputs occurs.
- The GPIO core can be used to control the **internal properties** of the device as well as the **behavior of external devices**.

AXI GPIO Block Diagram



Clock and Reset

- The AXI GPIO operates on the s_axi_aclk.
- The AXI GPIO is reset when s_axi_aresetn is asserted. This is an active-Low reset synchronous to s_axi_aclk clock.

Configure GPIO IP Instance

- **GPIO Width**

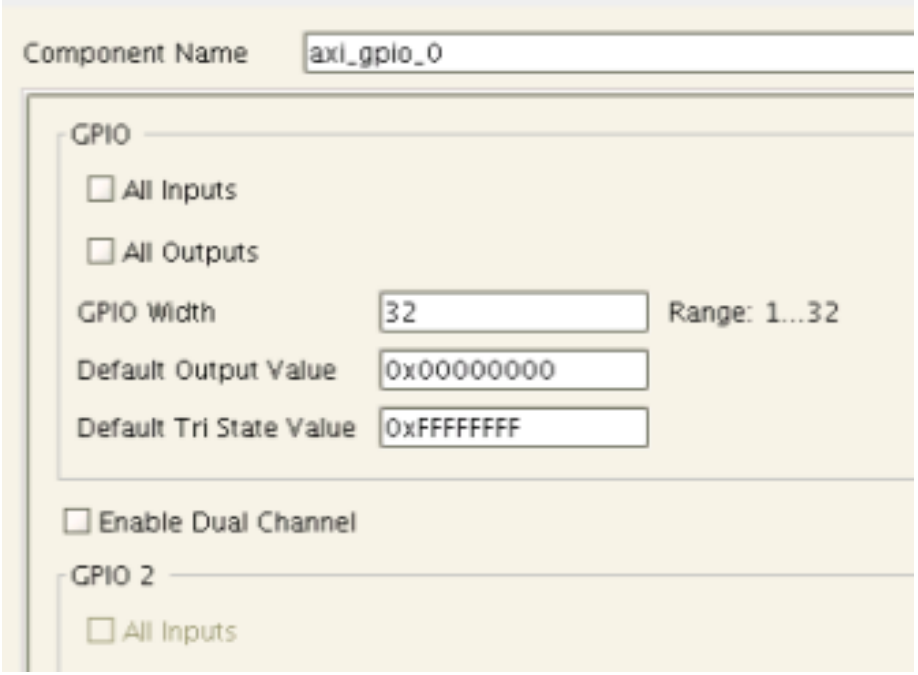
Can be from 1 to 32, and default value is **32.**

- **Default Output Value**

Sets the default value of all the enabled bits of this channel. By default, this parameter is set to **0x0**.

- **Default Tri State Value**

This value configures the input or output mode of each bit of GPIO channel. By default, this field has **0xFFFFFFFF**, configuring all GPIO bits in **input** mode.



Programming Sequence

- For input ports when the Interrupt is not enabled, use the following steps:
  1. Configure the port as input by writing the corresponding bit in GPIOx_TRI register with the value of 1.
  2. Read the corresponding bit in GPIOx_DATA register.

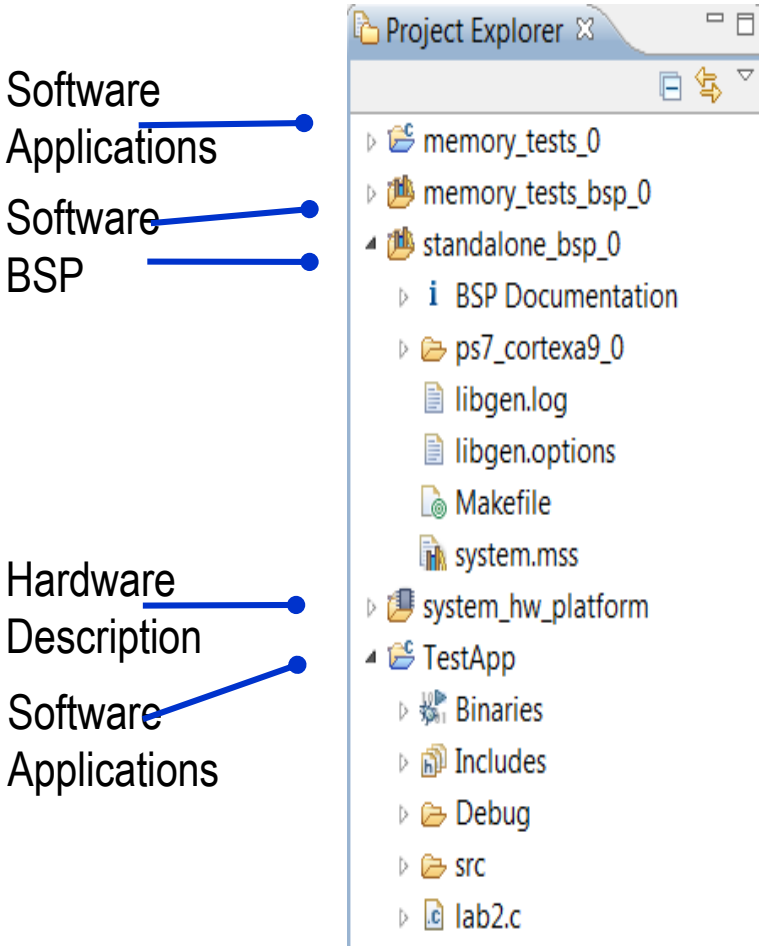
- For output ports, use the following steps:
  1. Configure the port as output by writing the corresponding bit in GPIOx_TRI register with a value of 0.
  2. Write the corresponding bit in GPIOx_DATA register.

Launching SDK From Vivado

- **Launch SDK**
  - In Vivado
    - **File> Export Hardware**
    - **File > Launch SDK**
  - Exporting
    - A Hardware Description File HDF file is first generated
    - A hardware platform specification project is then automatically created
      - The software application (and board support package) then can be created and associated with the hardware platform

C/C++ Project View

- Hierarchical list of the workspace projects in a hierarchical format
- Double-click to open a file
- Right-click the project to access its properties

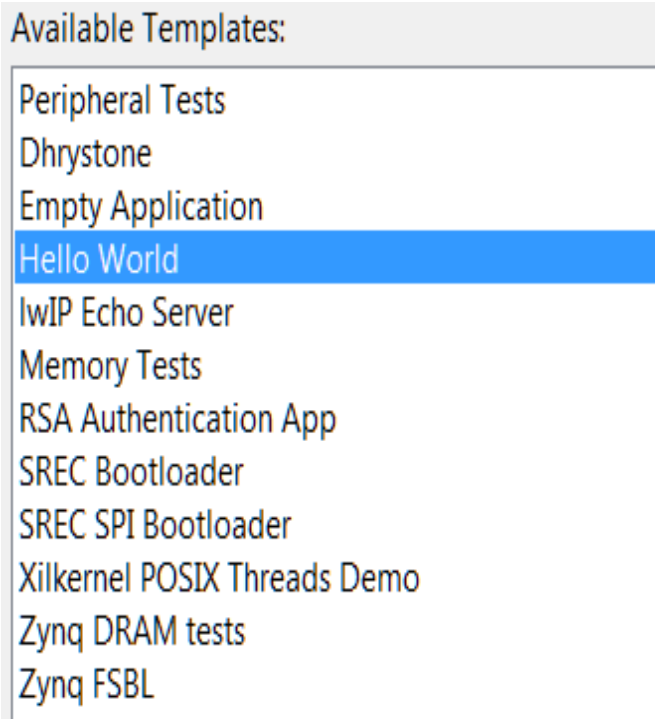


Creating a Board Support Package

- **The Board Support Package provides software services based on the processor and peripherals that make up the processor system**
- **Can be automatically created when creating Application project**
- **Can be created standalone**
- **Must be attached to a Hardware Platform**
  - **File > New > Board Support Package**
  - Select appropriate OS support
  - Third-party operating systems are supported with the appropriate BSP selection
  - Select required libraries support

Creating a Software Application Project

- **SDK supports multiple software application projects**
- **A software project is attached to a BSP project**
- **Sample applications are provided**
  - Great for quick test of hardware
    - Peripheral Tests
  - Starting point to base your own application on
- **Typically an Empty Application is opened to begin a non-standard project**



GPIO Driver API

- The General Purpose I/O driver resides in the gpio subdirectory.
- Details of the layer 1 high level driver can be found in the xgpio.h header file.
- Details of the layer 0 low level driver can be found in the xgpio_l.h header file.

gpio.h

- This file contains the software API definition of the Xilinx General Purpose I/O (XGpio) device driver component.
- The driver provides interrupt management functions. Implementation of interrupt handlers is left to the user.
- This driver is intended to be RTOS and processor independent.

Data Structure

- **There are two data structures used in gpio component:**
  - Xgpio
  - Xgpio_Config
- **Both are defined in gpio.h**

Data Structures - Xgpio

struct Xgpio

```
typedef struct {
    u32 BaseAddress;          /* Device base address */
    u32 IsReady;              /* Device is initialized and ready */
    int InterruptPresent;     /* Are interrupts supported in h/w */
    int IsDual;               /* Are 2 channels supported in h/w */
} XGpio;
```

- The XGpio driver instance data.
- The user is required to allocate a variable of this type for every GPIO device in the system.
- A pointer to a variable of this type is then passed to the driver API functions.

Data Structures – Xgpio_Config

struct Xgpio_Config

```
typedef struct {
        u16     DeviceId;           /* Unique ID  of device */
        u32     BaseAddress;        /* Device base address */
        int     InterruptPresent;   /* Are interrupts supported in h/w */
        int     IsDual;             /* Are 2 channels supported in h/w */
} XGpio_Config;
```

- **This typedef contains configuration information for the device.**

XGpio_Initialize

- **int XGpio_Initialize (XGpio **InstancePtr,* u16 *DeviceId*)**
  - Initialize the XGpio instance provided by the caller based on the given DeviceID.
- **Parameters:**
  - ***InstancePtr*** *is a pointer to an XGpio instance. The memory the pointer references must be* pre-allocated by the caller. Further calls to manipulate the component through the XGpio API must be made with this pointer.
  - ***DeviceId*** *is the unique id of the device controlled by this XGpio component. Passing in* a device id associates the generic XGpio instance to a specific device, as chosen by the caller or application developer.
- **Returns:**
  - **XST_SUCCESS** Initialization was successful.
  - **XST_DEVICE_NOT_FOUND** if the device configuration data was not found for a device with the supplied device ID.
- **Example:**

```
XGpio dip;
XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
```

XGpio_SetDataDirection

void XGpio_SetDataDirection (XGpio *InstancePtr,
unsigned Channel,
u32 DirectionMask);

- **Set the input/output direction of all discrete signals for the specified GPIO channel.**
- **Parameters:**
  - **InstancePtr** is a pointer to an XGpio instance to be worked on.
  - **Channel** contains the channel of the GPIO (1 or 2) to operate on.
  - **DirectionMask** is a bitmask specifying which discretes are input and which are output. Bits set to 0 are output and bits set to 1 are input.
- **Example:**

```
XGpio_SetDataDirection(&dip, 1, 0xffffffff); // set direction to input.
```

XGpio_DiscreteWrite

void XGpio_DiscreteWrite (XGpio *InstancePtr,
unsigned Channel,
u32 Data);

- **Write to discrete register for the specified GPIO channel.**
- **Parameters:**
  - **InstancePtr** is a pointer to an XGpio instance to be worked on.
  - **Channel** contains the channel of the GPIO (1 or 2) to operate on.
  - **Data** is the value to be written to the discrete register.
- **Example:**

```
XGpio led;
u32 data=1;
…
XGpio_DiscreteWrite(&led, 1, data);
```

XGpio_DiscreteRead

u32 XGpio_DiscreteRead (XGpio *InstancePtr,
unsigned Channel);

- **Read state of discretes for the specified GPIO channnel.**
- **Parameters:**
  - **InstancePtr** is a pointer to an XGpio instance to be worked on.
  - **Channel** contains the channel of the GPIO to operate on.
- **Returns:**
  - Current copy of the discretes register.
- **Example:**

```
XGpio push;
int psb_check;
…
psb_check = XGpio_DiscreteRead(&push, 1);
```

AXI GPIO Driver Calling Sequence

- XGpio_Initialize
- XGpio_SetDataDirection
- XGpio_DiscreteRead / XGpio_DiscreteWrite

References

- **AXI GPIO – pg144-axi-gpio.pdf:**
  To find this document, In Vivado **IP Integrator,** open **IP Catalog→**select **AXI GPIO→**right click **AXI GPIO,** select **Product Guide.**

- **Software API for AXI GPIO:**
  In SDK, after your project is created, open **system.mss** file, in **peripheral driver** session, find any instant of **GPIO,** click **documentation.** You will find the API document in the following directory:
  “VivadoDirectory”/data/embeddedsw/XilinxProcessorIPLib/drivers/gpio_v4_0/doc/html/api/index.html