International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

# Design and Implementation of General Purpose Input Output (GPIO) Protocol

**Bhavnil Patel[1] , Bhargav Tarpara[2]**
[1] *M.Tech. VLSI, U.V.Patel college of Engineering and Technology, Kherva, Mehsana, India*
[2]*Verification Technical Assistant*

***Abstract*— General purpose input/output (GPIO) is a generic pin on an integrated circuit (IC) whose behavior, including whether it is an i/o pin, that can be controlled by the user at run time. GPIO pins have no special purpose defined, and by default unused. The idea after that sometimes the system integrator building a full system that accustom the chip might find it useful to have a handful of immense digital control lines, and having these accessible from the chip can save the hassle of having to arrange additional circuitry to give them. In this paper i have tried to implement GPIO design using FPGA (Field Programmable Gate Array).That design check throw Directed Test case and apply that RTL(Register Transfer Level) in Physical Design (Backend side) so a complete ASIC (Application Specific Integrated Circuit) cycle.**

Keywords—General Purpose Input/output (GPIO), Integrated Circuit (IC), Field Programmable Gate Array (FPGA)**,** Register Transfer Level (RTL)**,**Application Specific Integrated Circuit (ASIC), Advanced Peripheral Bus (APB), Advanced Microcontroller Bus Architecture (AMBA), Universal Verification Methodology (UVM)

## I. INTRODUCTION

A General Purpose Input/output (GPIO) is an interface available on latest microcontrollers (MCU) to provide an ease of access to the devices internal properties. Generally there are multiple GPIO pins on a single MCU for the use of different interaction so concurrent application. The GPIO IP core is user-programmable general-purpose I/O controller. It is used to mechanism functions that are not implemented with the dedicated controllers in a system and necessary simple input and/or output software controlled signals [1].

The pins can be programmed as input, where data from several external source is being fed into the system to be flexible at a desired time and location. Output can also be execute on GPIOs, where formatted date can be transmitted efficiently to outside devices, this provides a simple implement to program and re transmit data depending on user desires through a single port interface. The pins are generally arranged into groups of 8 pins where signals can be sent or received to and from other sources [1].

In many applications, the GPIOs can be configured as interrupt lines for a CPU to signal immediate processing of input lines. In many designs, they also have the ability to control and utilize Direct Memory Access (DMA) to transfer blocks of data in a more effectively. Significantly all ports can be tailored to fit specific design aims and serve reusability within applications [1].

Every GPIO port can be configured for i/o or bypass mode. All output data can be set in one access Single or multiples bits can be set or cleared independently. Each GPIO port can provide an interrupt source and has its own configuration options:

1. Level sensitive, single edge triggered or level change.
2. Active high or low respectively rising edge or falling edge.
3. Individual interrupt enable register and status flags.[2]

The core provides several synthesis options to ease the system integration and minimize the gate count:

1.Selectable CPU bus width: default options are 8/16/32-bit.
2.Selectable number of GPIO ports.
3.CPU read back enable.[2]

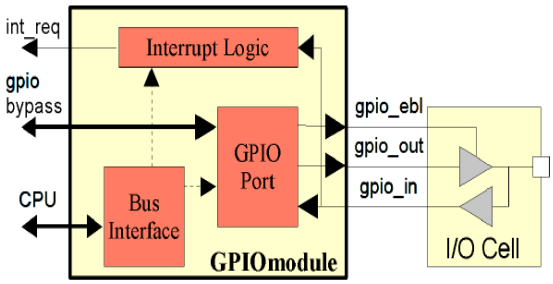


Figure 1: GPIO module Diagram [2]

**Objectives:**

General Purpose I/O (GPIO) pins are single necessary to give versatile to digital and analog signals for ADC conversions. To provide efficiency the signals should be nuclear controllable on a particular chip board. All GPIO should be able to define either an input mode or an output mode for individual pins on the chip. At last the pins must be extendable for a wide array of applications and functional uses that define its generality in use.

**Usage:**

1.Devices with pin scarcity integrated circuits such as system-on-a-chip, embedded and custom hardware, and programmable logic devices (for example, FPGAs).
2.Multi-function chips power managers, audio codecs, and video cards.
3.Embedded applications (for example, Arduino, BeagleBone, PSoC kits and Raspberry Pi) make heavy use of GPIO for reading from various environmental sensors (IR, video, temperature, 3-axis orientation, and acceleration), and for writing output to DC motors (via PWM), audio, LCD displays, or LEDs for status.

**Capability:**

1. GPIO pins can be configured to be input or output.
2. GPIO pins can be enabled/disabled.
3. Input values are like high=1, low=0.
4. Output values are writable/readable
5. Input values can often be used as IRQs (typically for wakeup events)
6. GPIO peripherals vary absolutely widely. In some of the cases, they are much simple, a group of pins which can be switched as a group to either i/p or o/p. In others case pins can be set up flexibly to accept or source different logic voltages like configurable drive strengths and pull ups/downs. The input and output voltages are not in all instances, that is limited to the supply voltage of the device with the GPIOs on and may be damaged by greater voltages.
7. A GPIO pin's state may be exposed to the software developer through one of a number of multiple interfaces, like a memory mapped peripheral, or by dedicated IO port instructions.
8. A few GPIOs have 5 V tolerant inputs: even when the device has a low supply voltage such as 2 V, the device can accept up to 5 V without damage.

**Ports:**

A GPIO port is a group of GPIO pins (typically 8 GPIO pins) manage in a group and controlled as a group.

The reminder of the paper is organized as follows section II descried GPIO Architecture & Block Diagram.

## II. GPIO Architecture & Block Diagram

**GPIO Architecture:**

General architecture of GPIO IP core. It consists of four main building blocks:

1.APB(Master & Slave)
2.GPIO registers
3.Auxiliary inputs
4.Interface to external I/O cells and pads [3]

1. APB:

The APB is part of the AMBA protocol logic family. It display a chipper interface that is optimized for minimal power consumption and reduced interface complexity.

The APB protocol is not pipe lined, APB use it to connect to lower bandwidth peripherals which do not wants the high performance of the AXI protocol.

The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow. Every transfer takes minimum 2 cycles.[4]

2. GPIO Registers:

The GPIO IP Core has multiple software accessible registers. Some of them registers have the same width as no. of general-purpose Input-Output signals and they are from 0– 31 bits. The Host through these registers programs type and operation of each general-purpose Input-Output Signal.[3]

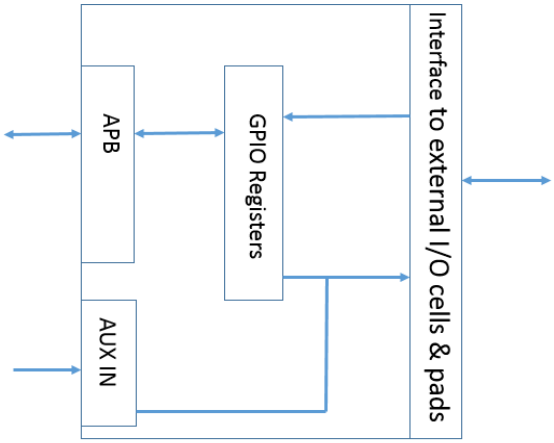


Figure 2: GPIO Core Architecture

3. Auxiliary Inputs:

The auxiliary inputs can bypass RGPIO_OUT outputs based on programming of RPGIO_AUX register. It can be used to multiplex other on chip peripheral devices on GPIO pins.[3]

4. Interface to External I/O Cells and Pads:

External interface connects GPIO core to external Input-output ring cells and pads. To assist open drain or 3 state outputs, suitable open-drain or three-state Input-output cells must be used. ECLK register is worked as a part of external interface. Usually register inputs based on External clock reference.[3]

**Features:**

1. Number of general-purpose I/O signals is user selectable and can be in range from 1 to 32. For more I/O multiple GPIO cores can be used in parallel.
2. Each general-purpose I/O signals can be bi-directional external bi-directional I/O Cells are required in this case.
3. Each general-purpose I/O signals can be three-stated or open-drain enabled (External 3 state or open-drain Input-Output cells need in this case).
4. General-purpose I/O signals programmed as inputs can cause an interrupt request to the CPU.
5. General-purpose I/O signals programmed as inputs can be registered at raising edge of system clock or at user programmed edge of external clock.
6. All general-purpose I/O signals are programmed as inputs at hardware reset.
7. Auxiliary inputs to GPIO core to bypass outputs from RGPIO_OUT register.
8. Alternative input reference clock signal from external interface.
9. Especially configurable (implementation of registers, external clock reverse versus needle flip-flops etc.)
10. APB interface

**GPIO Operation:**

This section explain the operation of the GPIO core. The GPIO core provides toggling of general-purpose outputs and sampling of general-purpose inputs under software control.[3]

General-purpose inputs can create interrupts so that software does not have to be in poll mode all the time when sampling inputs.[3]

Switching output drivers into open-drain or 3 state mode will disable general-purpose o/p. To lower number of pins of the chip, other on-

chip peripherals devices can be multiplexed each other with the GPIO pins. For this object, auxiliary inputs can be multiplexed on general-purpose outputs.

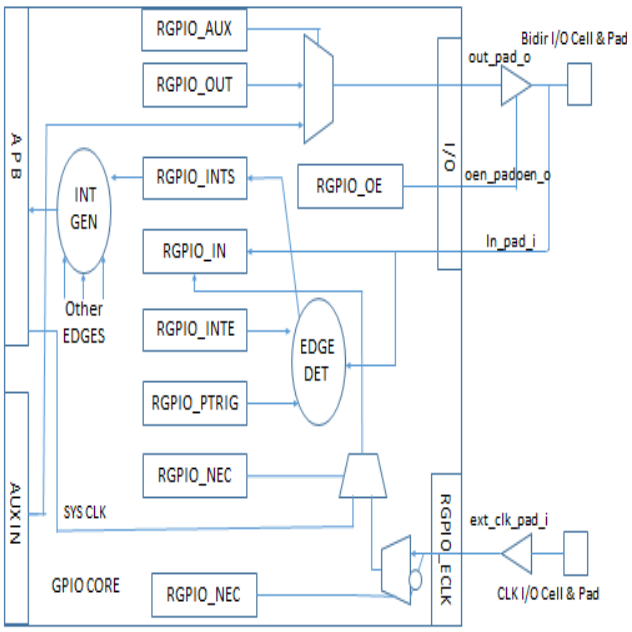


Figure 3: GPIO Block Diagram

## GPIO Operations:

- Hardware Reset
- General-Purpose I/O as Polled Input
- General-Purpose I/O as Input in Interrupt Mode
- General-Purpose I/O as Output
- General-Purpose I/O as Bi-Directional I/O
- General-Purpose I/O driven by Auxiliary Input.[3]

## GPIO Registers:

This section describes all control and status register inside the GPIO core.[3]

| Name | Width | Access | Description |
|---|---|---|---|
| RGPIO_IN | 0 – 31 | R | GPIO input data |
| RGPIO_OUT | 0 – 31 | R/W | GPIO output data |
| RGPIO_OE | 0 – 31 | R/W | GPIO output driver enable |
| RGPIO_INTE | 0 – 31 | R/W | Interrupt enable |
| RGPIO_PTRIG | 0 – 31 | R/W | Type of event that triggers an interrupt |
| RGPIO_AUX | 0 – 31 | R/W | Multiplex Auxiliary inputs to GPIO outputs |
| RGPIO_CTRL | 2 | R/W | Control register |
| RGPIO_INTS | 0 – 31 | R/W | Interrupt status |

Table 1. List of All Software Accessible Registers [3]

## I/O Ports:

GPIO IP core has three interfaces.

1.APB interface
2.Auxiliary inputs interface
3.Interface to external I/O cells and pads [3]

1.APB interface:

• Master Description:

APB is a single bus master so there is no need for an arbiter. The master carry the address and write buses and also express a conjugative decode of the address to decide which PSELx signal to trigger and it is also important for driving the PENABLE signal to time the transfer. It carry APB data onto the system bus during a read transfer.[4]

• Slave Description:

APB slaves have a much simple and flexible interface. The perfect implementation the interface will be dependent on the design style employed and many non-identical options are possible. In this two signals are mainly protect the loss data while transfer of data. They are PSLVERR and PREADY.[4]

2.Auxiliary inputs:

Auxiliary inputs descried above.

| Port | Width | Direction | Description |
|---|---|---|---|
| aux_i | 0-31 | Inputs | GPIO auxiliary inputs |

Table 2. Auxiliary input signals [3]

3.Interface to external I/O cells and pads

External interface connects GPIO core to external I/O ring cells and pads. To assist open drain or 3 state outputs, I/O cells with open drain or 3 state used.

Part of external interface is also ECLK signal. It can be used to register inputs based on external clock reference.[3]

| Port | Width | Direction | Description |
|---|---|---|---|
| in_pad_i | 0-31 | Inputs | GPIO inputs |
| out_pad_o | 0-31 | Outputs | GPIO outputs |
| oen_padoen_o | 0-31 | Outputs | GPIO output drivers enables (for three-state or open-drain drivers) |
| ext_clk_pad_i | 1 | Input | Alternative GPIO inputs' latch clock |

Table 3. External interface [3]

## III. GPIO Simulation Results

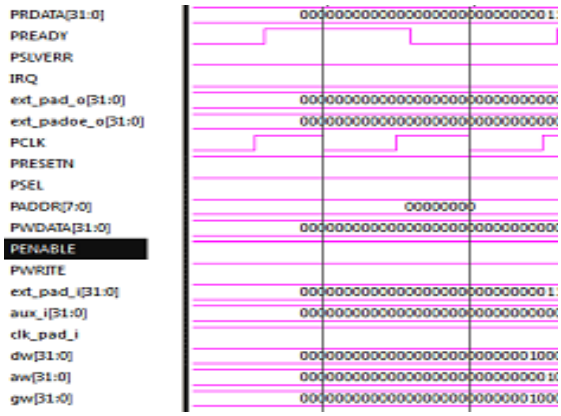


Figure 4. Simulation result of Input Register

Whenever all APB signal and ext_pad_i trigger at posedge of PCLK , Data which is contained by ext_pad_i has been stored to PRDATA .

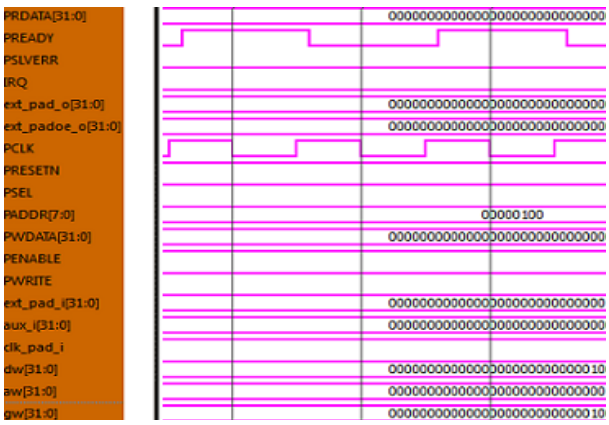


Figure 5. Simulation result of Output Register

When all APB signal trigger that time Data which contained in PWDATA has been stored to ext_pad_o and PRDATA.

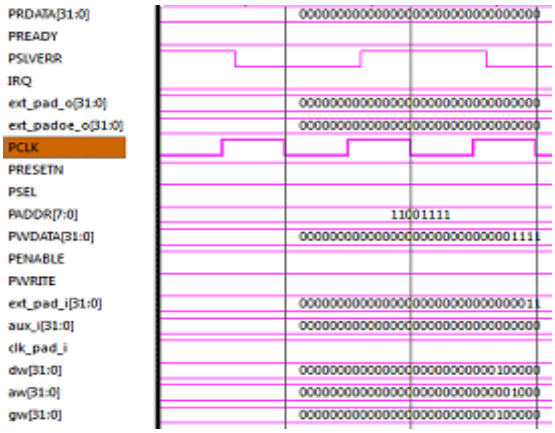


Figure 6. Simulation result of PSLVERR (Failure)

Over here address of PADDR given wrong then it shows transfer Failure so PSLVERR high(1) and PRDATA & ext_pad_o low(0).

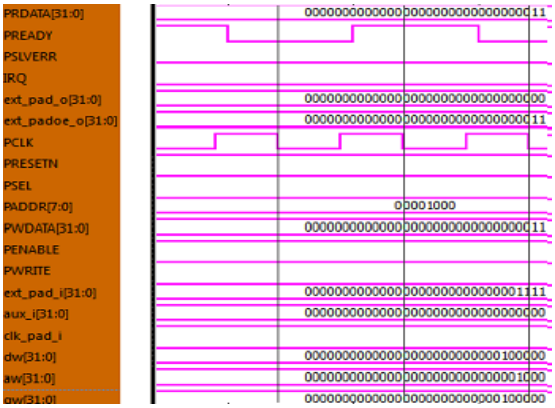


Figure 7. Simulation result of Output Enable Register

When output enable is trigger then PWDATA contained has been stored in ext_padoe_o & PRDATA.

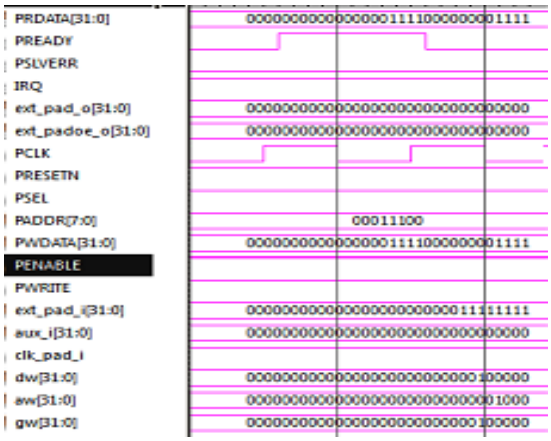


Figure 8. Simulation result of Interrupt status Register

In this interrupt status shows by IRQ high, data of PWDATA has been stored in PRDATA but ext_pad_o low because is show interrupt cause by input.

## CONCLUTION:

This paper gives an outline of the GPIO Protocol and explain the GPIO working in detail. The GPIO is designed using the Verilog HDL according to the specification and is verified using Xilinx. The simulation results show that the data write into register and read form registers, that value stored in a output side. Hence, the design is functionally correct. Xilinx also ensures the functional correctness of the design.

## MOTIVATIONAL WORK:

In this paper, Design of GPIO is verified though a Direct Test Case in Xilinx. But you can also verified in UVM by applying a Random or Direct Test Cases. After verified that design dump in to Design Complier (DC) and ICC. So, a full ASIC cycle complete and after that a design will Tap out.

## IV. REFERENCE

[1].Sasang Balachandran, “General Purpose Input/Output(GPIO)”: http://www.egr.msu.edu/classes/ece480/capstone/fall09/group03/AN_balachandran.pdf GPIO

[2].INICORE, http://inicore.com/pdf/act/mb_gpio_mod_act.pdf

[3]. OpenCores: http://opencores.org

[4]. AMBA Specification: http://www-micro.deis.unibo.it/~magagni/amba99.pdf

[5]. Chris Spear, ”System Verilog for Verification”

[6]. Samir Palnitkar ” Verilog HDL A Guide to Digital Design and Synthesis”