myGPIO

3.2

Generato da Doxygen 1.8.8

Ven 7 Lug 2017 12:40:56

Indice

Capitolo 1

Pagina Principale

Periferica AXI4 Lite che implementa una GPIO pilotabile da processing-system.

Registri interni del device

Il device possiede i registri indicati di seguito. Per oognuno di essi viene indicata la modalità di accesso (R sola lettura, W sola scrittura, R/W lettura scrittura), e l'offset, rispetto all'indirizzo base del device, col quale è possibile indirizzarli.

- MODE (R/W, offset +0x0): consente di impostare i singoli pin del device come ingressi o uscite; solo i GPI↩ O_width bit meno significativi del registro hanno significato, agire sui restanti bit non produce nessun effetto; Il valore che i singoli pin possono assumere è:
  - '1': il pin viene configurato come pin di uscita;
  - 'ò: il pin viene configurato come pin di ingresso;
- WRITE (R/W, offset +0x4): consente di imporre un valore ai pin del device, qualora essi siano configurati come uscite; solo i GPIO_width bit meno significativi del hanno significato, agire sui restanti bit non produce nessun effetto;
- READ (R, offset +0x8): consente di leggere il valore dei pin del device, sia quelli configurati come ingressi che quelli configurati come uscite (il cui valore coincide con quello settato nel registro WRITE); solo i GPIO_width bit meno significativi del registro hanno significato, gli altri vengono letti zero;
- GIES (Global Interrupt Enable/Status, R/W, offset 0xC): Consente di abilitare/disabilitare gli interrupt globali della periferica; solo due dei bit sono significativi:
  - IE (bit 0): interrupt enable, abilita gli interrupt, può essere scritto e letto; se posto ad '1' la periferica potrà generare interrupt quando uno dei pin impostati come ingresso assume valore '1' (ed il corrispondente bit in PIE è impostato ad '1'); se posto a '0' il device non genererà mai interruzioni;
  - IS (bit 1): interrupt status, settato internamente ad '1' nel caso in cui la periferica abbia generato interrupt; replica del segnale "interrupt" diretto verso il processing-system.
- PIE (Pin Interrupt Enable, R/W, offset 0x10): consente di abilitare/disabilitare gli interrupt per i singoli pin. Con GIES(0)='1' e MODE(n)='0' (cioè se gli interrupt globali sono abilitati e il pin n-esimo è configurato come input), se PIE(n)='1' allora il device genererà un interrupt verso il processing-system quando il pin n-esimo assumerà valore '1', mentre, se PIE(n)='0' non verrà generata una interruzione;
- IRQ (Interrupt Request, R, offset 0x14): IRQ(n)='1' indica che la sorgente di interruzione è il bit n-esimo; la or-reduce di tale registro costituisce il segnale "interrupt" diretto verso il processing system;
- IACK (Interrupt Ack, W, offset 0x18): imponento IACK(n)='1' è possibile segnalare al device che l'interruzione generata dal in n-esimo è stata servita; il bit IRQ(n) verrà resettato automaticamente.

Process di scrittura dei registri della periferica

Il process che implementa la logica di scrittura dei registri è stato modificato in modo da ottenere il seguente indirizzamento:

Indirizzo	Offset	Registro
b"00000"	0x00	MODE
b"00100"	0x04	WRITE
b"01000"	0x08	READ(∗)
b"01100"	0x0C	GIES(∗∗)
b"10000"	0x10	PIE
b"10100"	0x14	IRQ(∗∗∗)
b"11000"	0x18	IACK(∗∗∗∗)

(∗) Il registro READ è a sola lettura: le scritture su questo registro non producono effetti; la scrittura, infatti, avviene su slv_reg2, che è inutilizzato;
(∗∗) La scrittura ha effetto solo sul bit zero del registro;
(∗∗∗) Il registro IRQ è a sola lettura: le scritture su questo registro non producono effetti; la scrittura, infatti, avviene su slv_reg5, che è inutilizzato;
(∗∗∗∗) La scrittura su IACK è fittizzia, nel senso che appena si smette di indirizzare il registro, esso assume valore zero;

Process di lettura dei registri della periferica

Il process che implementa la logica di lettura dei registri è stato modificato in modo da ottenere il seguente indirizzamento:

Indirizzo	Offset	Registro
b"00000"	0x00	MODE
b"00100"	0x04	WRITE
b"01000"	0x08	READ(∗)
b"01100"	0x0C	GIES(∗∗)
b"10000"	0x10	PIE
b"10100"	0x14	IRQ
b"11000"	0x18	IACK(∗∗∗)

(∗) Il registro READ è direttamente connesso alla porta GPIO_inout
(∗∗) Il bit 2 di GIES è il flag "interrupt", che vale '1' nel caso in cui la periferica abbia generato interrupt ancora non gestiti.
(∗∗∗) Viene letto sempre zero, dal momento che la scrittura su tale registro è fittizzia.

Process di scrittura su IRQ

La logica di scrittura su IRQ è semplice (non viene scritto come un normale registro, ma pilotato internamente dalla periferica): se uno dei bit di GPIO_inout_masked è '1', (la or-reduce è 1) allora il valore del segnale GPIO_inout_↩ masked viene posto in bitwise-or con il valore attuale del registro IRQ, in modo da non resettare i bit di quest' ultimo che siano stati settati a seguito di una interruzione non ancora servita se uno dei bit di IACK è '1' (la or-reduce è '1'), allora il nuovo valore del registro IRQ viene ottenuto

- mascherando IACK con l'attuale valore di IRQ, in modo da non effettuare il set di bit resettati
- ponendo in XOR la maschera precedente con il valore attuale del registro

Capitolo 2

Indice dei moduli

2.1 Moduli

Questo è l'elenco di tutti i moduli:

Capitolo 3

Design Unit Index

3.1 Design Unit Hierarchy

Questo elenco di ereditarietà è ordinato approssimativamente, ma non completamente, in ordine alfabetico:

Capitolo 4

Design Unit Index

4.1 Design Unit List

Here is a list of all design unit members with links to the Entities they belong to:

Capitolo 5

Indice dei file

5.1 Elenco dei file

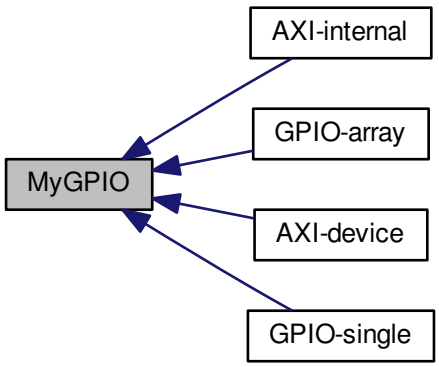
Questo è un elenco dei file documentati con una loro breve descrizione:

Capitolo 6

Documentazione dei moduli

6.1 MyGPIO

Diagramma di collaborazione per MyGPIO:



Moduli

- GPIO-array
- GPIO-single
- AXI-device
- AXI-internal

6.1.1 Descrizione dettagliata

6.2 GPIO-array

Diagramma di collaborazione per GPIO-array:

Entities

- GPIOarray entity

  Array di celle GPIO, pilotabili singolarmente.
- Structural architecture

Libraries

- ieee

Use Clauses

- std_logic_1164

Components

- GPIOsingle

Generics

- GPIO_width **natural:=** 4

  numero di istanze GPIO create, di default pari a 4 celle.

Ports

- GPIO_enable **in std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di abilitazione, permette di pilotare la linea "GPIO_inout". Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin.
- GPIO_write **in std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di input, diretto verso l'esterno del device. Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin.
- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device. Può essere usato per leggere/scrivere segnali digitali da/verso l'esterno del device.
- GPIO_read **out std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di output, diretto verso l'esterno del device. Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin, per cui questo segnale assume esattamente il valore con cui viene impostato il segnale GPIO_write(i). Se GPIO_enable(i)=0, il valore del segnale può essere forzato dall' esterno del device.

6.2.1 Descrizione dettagliata

6.3 GPIO-single

Diagramma di collaborazione per GPIO-single:

Entities

- GPIOsingle entity

  Cella base GPIO.
- Structural architecture

Libraries

- ieee

Use Clauses

- std_logic_1164

Ports

- GPIO_enable **in std_logic**

  segnale di abilitazione, permette di pilotare la linea "GPIO_inout". Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore del segnale GPIO_inout
- GPIO_write **in std_logic**

  segnale di input, diretto verso l'esterno del device GPIOsingle. Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore del pin GPIO_inout.
- GPIO_inout **inout std_logic**

  segnale bidirezionale diretto verso l'esterno del device. Può essere usato per leggere/scrivere segnali digitali da/verso l'esterno del device GPIOsingle
- GPIO_read **out std_logic**

  segnale di output, diretto verso l'esterno del device. Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore dei pin, per cui questo segnale assume esattamente il valore con cui viene impostato il segnale GPIO_write

6.3.1 Descrizione dettagliata

6.4 AXI-device

Diagramma di collaborazione per AXI-device:

Entities

- myGPIO entity

Libraries

- ieee

Use Clauses

- std_logic_1164
- numeric_std

Generics

- GPIO_width **natural:=** 4

  numero di GPIO offerti dalla periferica, di default pari a 4 celle.
- C_S00_AXI_DATA_WIDTH **integer:=** 32
- C_S00_AXI_ADDR_WIDTH **integer:=** 5

Ports

- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device.
- interrupt **out std_logic**

  segnale di interrupt a livelli diretto verso il processing - system. Se le interruzioni sono abilitate ed uno dei pin del device è settato come input ed è abilitato a generare interruzioni, diventa '1' appena tale pin assume valore '1', e mantiene tale valore fino a quando tutte le interruzioni non siano state servite.
- s00_axi_aclk **in std_logic**
- s00_axi_aresetn **in std_logic**
- s00_axi_awaddr **in std_logic_vector(C_S00_AXI_ADDR_WIDTH - 1 downto 0)**
- s00_axi_awprot **in std_logic_vector(2 downto 0)**
- s00_axi_awvalid **in std_logic**
- s00_axi_awready **out std_logic**

- s00_axi_wdata **in std_logic_vector(C_S00_AXI_DATA_WIDTH - 1 downto 0)**
- s00_axi_wstrb **in std_logic_vector((C_S00_AXI_DATA_WIDTH / 8)- 1 downto 0)**
- s00_axi_wvalid **in std_logic**
- s00_axi_wready **out std_logic**
- s00_axi_bresp **out std_logic_vector(1 downto 0)**
- s00_axi_bvalid **out std_logic**
- s00_axi_bready **in std_logic**
- s00_axi_araddr **in std_logic_vector(C_S00_AXI_ADDR_WIDTH - 1 downto 0)**
- s00_axi_arprot **in std_logic_vector(2 downto 0)**
- s00_axi_arvalid **in std_logic**
- s00_axi_arready **out std_logic**
- s00_axi_rdata **out std_logic_vector(C_S00_AXI_DATA_WIDTH - 1 downto 0)**
- s00_axi_rresp **out std_logic_vector(1 downto 0)**
- s00_axi_rvalid **out std_logic**
- s00_axi_rready **in std_logic**

6.4.1 Descrizione dettagliata

6.5 AXI-internal

Diagramma di collaborazione per AXI-internal:

Entities

- myGPIO_AXI entity

  Periferica AXI4 Lite che implementa una GPIO pilotabile da processing-system.
- arch_imp architecture

Processes

- PROCESS_0(**S_AXI_ACLK**)
- PROCESS_1(**S_AXI_ACLK**)
- PROCESS_2(**S_AXI_ACLK**)
- PROCESS_3(**S_AXI_ACLK**)
- PROCESS_4(**S_AXI_ACLK**)
- PROCESS_5(**S_AXI_ACLK**)
- PROCESS_6(**S_AXI_ACLK**)
- PROCESS_7(**MODE , WRITE , slv_reg2 , GIES , PIE , IRQ , IACK , slv_reg7 , axi_araddr , S_AXI_ARESETN , slv_reg_rden**)
- PROCESS_8(**S_AXI_ACLK**)

Libraries

- ieee

Use Clauses

- std_logic_1164
- numeric_std
- std_logic_misc

Components

- GPIOarray

Constants

- ADDR_LSB **integer:=(C_S_AXI_DATA_WIDTH/** 32 **)+** 1
- OPT_MEM_ADDR_BITS **integer:=** 2

Generics

- GPIO_width **natural:=** 4

  numero di GPIO offerti dalla periferica, di default pari a 4 celle.
- C_S_AXI_DATA_WIDTH **integer:=** 32
- C_S_AXI_ADDR_WIDTH **integer:=** 5

Ports

- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device.
- interrupt **out std_logic**

  segnale di interrupt a livelli diretto verso il processing - system. Se le interruzioni sono abilitate ed uno dei pin del device è settato come input ed è abilitato a generare interruzioni, diventa '1' appena tale pin assume valore '1', e mantiene tale valore fino a quando tutte le interruzioni non siano state servite.
- S_AXI_ACLK **in std_logic**
- S_AXI_ARESETN **in std_logic**
- S_AXI_AWADDR **in std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- S_AXI_AWPROT **in std_logic_vector(2 downto 0)**
- S_AXI_AWVALID **in std_logic**
- S_AXI_AWREADY **out std_logic**
- S_AXI_WDATA **in std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- S_AXI_WSTRB **in std_logic_vector((C_S_AXI_DATA_WIDTH/ 8)- 1 downto 0)**
- S_AXI_WVALID **in std_logic**
- S_AXI_WREADY **out std_logic**
- S_AXI_BRESP **out std_logic_vector(1 downto 0)**
- S_AXI_BVALID **out std_logic**
- S_AXI_BREADY **in std_logic**
- S_AXI_ARADDR **in std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- S_AXI_ARPROT **in std_logic_vector(2 downto 0)**
- S_AXI_ARVALID **in std_logic**
- S_AXI_ARREADY **out std_logic**
- S_AXI_RDATA **out std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- S_AXI_RRESP **out std_logic_vector(1 downto 0)**
- S_AXI_RVALID **out std_logic**
- S_AXI_RREADY **in std_logic**

Signals

- axi_awaddr **std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- axi_awready **std_logic**
- axi_wready **std_logic**
- axi_bresp **std_logic_vector(1 downto 0)**
- axi_bvalid **std_logic**
- axi_araddr **std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- axi_arready **std_logic**
- axi_rdata **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- axi_rresp **std_logic_vector(1 downto 0)**
- axi_rvalid **std_logic**
- MODE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- WRITE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- READ **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- GIES **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- PIE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**

- IRQ **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- IACK **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg2 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg5 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg6 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg7 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg_rden **std_logic**
- slv_reg_wren **std_logic**
- reg_data_out **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- byte_index **integer**
- GPIO_inout_masked **std_logic_vector(GPIO_width - 1 downto 0)**
- interrupt_tmp **std_logic:=' 0 '**

6.5.1 Descrizione dettagliata

Capitolo 7

Documentazione delle classi

7.1 arch_imp Architecture Reference

Processes

- PROCESS_0(S_AXI_ACLK)
- PROCESS_1(S_AXI_ACLK)
- PROCESS_2(S_AXI_ACLK)
- PROCESS_3(S_AXI_ACLK)
- PROCESS_4(S_AXI_ACLK)
- PROCESS_5(S_AXI_ACLK)
- PROCESS_6(S_AXI_ACLK)
- PROCESS_7(MODE , WRITE , slv_reg2 , GIES , PIE , IRQ , IACK , slv_reg7 , axi_araddr , S_AXI_AR↩ ESETN , slv_reg_rden)
- PROCESS_8(S_AXI_ACLK)
- PROCESS_9(S_AXI_ACLK , S_AXI_ARESETN , GPIO_inout_masked , IACK)

Components

- GPIOarray

Constants

- ADDR_LSB integer:=(C_S_AXI_DATA_WIDTH/ 32)+ 1
- OPT_MEM_ADDR_BITS integer:= 2

Signals

- axi_awaddr std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)
- axi_awready std_logic
- axi_wready std_logic
- axi_bresp std_logic_vector(1 downto 0)
- axi_bvalid std_logic
- axi_araddr std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)
- axi_arready std_logic
- axi_rdata std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)
- axi_rresp std_logic_vector(1 downto 0)
- axi_rvalid std_logic

- MODE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- WRITE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- READ **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- GIES **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- PIE **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- IRQ **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- IACK **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg2 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg5 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg6 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg7 **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- slv_reg_rden **std_logic**
- slv_reg_wren **std_logic**
- reg_data_out **std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- byte_index **integer**
- GPIO_inout_masked **std_logic_vector(GPIO_width - 1 downto 0)**
- interrupt_tmp **std_logic:=' 0 '**

Instantiations

- gpioarray_inst **GPIOarray**

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/myGPIO_AXI.vhd

7.2 GPIOarray Entity Reference

Array di celle GPIO, pilotabili singolarmente.

Diagramma delle classi per GPIOarray

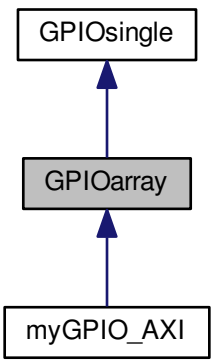
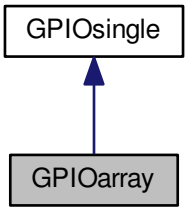


Diagramma di collaborazione per GPIOarray:



Entities

- Structural architecture

Libraries

- ieee

Use Clauses

- std_logic_1164

Generics

- GPIO_width **natural:=** 4

  numero di istanze GPIO create, di default pari a 4 celle.

Ports

- GPIO_enable **in std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di abilitazione, permette di pilotare la linea "GPIO_inout". Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin.
- GPIO_write **in std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di input, diretto verso l'esterno del device. Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin.
- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device. Può essere usato per leggere/scrivere segnali digitali da/verso l'esterno del device.
- GPIO_read **out std_logic_vector(GPIO_width - 1 downto 0)**

  segnale di output, diretto verso l'esterno del device. Quando GPIO_enable(i)=1, la linea GPIO_inout(i) e quella GPIO_write(i) sono connesse tra loro, consentendo la scrittura del valore del pin, per cui questo segnale assume esattamente il valore con cui viene impostato il segnale GPIO_write(i). Se GPIO_enable(i)=0, il valore del segnale può essere forzato dall' esterno del device.

7.2.1 Descrizione dettagliata

Array di celle GPIO, pilotabili singolarmente.

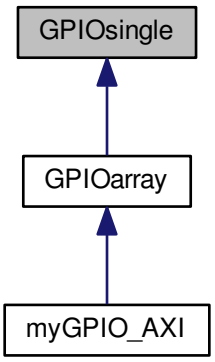
La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/GPIOarray.vhd

7.3 GPIOsingle Entity Reference

Cella base GPIO.

Diagramma delle classi per GPIOsingle



Entities

- Structural architecture

Libraries

- ieee

Use Clauses

- std_logic_1164

Ports

- GPIO_enable **in std_logic**

  segnale di abilitazione, permette di pilotare la linea "GPIO_inout". Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore del segnale GPIO_inout
- GPIO_write **in std_logic**

  segnale di input, diretto verso l'esterno del device GPIOsingle. Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore del pin GPIO_inout.

- GPIO_inout **inout std_logic**

  segnale bidirezionale diretto verso l'esterno del device. Può essere usato per leggere/scrivere segnali digitali da/verso l'esterno del device GPIOsingle
- GPIO_read **out std_logic**

  segnale di output, diretto verso l'esterno del device. Quando GPIO_enable=1, la linea GPIO_inout e quella GPIO_write sono connesse tra loro, consentendo la scrittura del valore dei pin, per cui questo segnale assume esattamente il valore con cui viene impostato il segnale GPIO_write

7.3.1 Descrizione dettagliata

Cella base GPIO.

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/GPIOsingle.vhd

7.4 myGPIO Entity Reference

Libraries

- ieee

Use Clauses

- std_logic_1164
- numeric_std

Generics

- GPIO_width **natural:=** 4

  numero di GPIO offerti dalla periferica, di default pari a 4 celle.
- C_S00_AXI_DATA_WIDTH **integer:=** 32

- C_S00_AXI_ADDR_WIDTH **integer:=** 5

Ports

- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device.
- interrupt **out std_logic**

  segnale di interrupt a livelli diretto verso il processing - system. Se le interruzioni sono abilitate ed uno dei pin del device è settato come input ed è abilitato a generare interruzioni, diventa '1' appena tale pin assume valore '1', e mantiene tale valore fino a quando tutte le interruzioni non siano state servite.
- s00_axi_aclk **in std_logic**

- s00_axi_aresetn **in std_logic**
- s00_axi_awaddr **in std_logic_vector(C_S00_AXI_ADDR_WIDTH - 1 downto 0)**
- s00_axi_awprot **in std_logic_vector(2 downto 0)**
- s00_axi_awvalid **in std_logic**
- s00_axi_awready **out std_logic**

- s00_axi_wdata **in std_logic_vector(C_S00_AXI_DATA_WIDTH - 1 downto 0)**
- s00_axi_wstrb **in std_logic_vector((C_S00_AXI_DATA_WIDTH / 8)- 1 downto 0)**
- s00_axi_wvalid **in std_logic**
- s00_axi_wready **out std_logic**
- s00_axi_bresp **out std_logic_vector(1 downto 0)**
- s00_axi_bvalid **out std_logic**
- s00_axi_bready **in std_logic**
- s00_axi_araddr **in std_logic_vector(C_S00_AXI_ADDR_WIDTH - 1 downto 0)**
- s00_axi_arprot **in std_logic_vector(2 downto 0)**
- s00_axi_arvalid **in std_logic**
- s00_axi_arready **out std_logic**
- s00_axi_rdata **out std_logic_vector(C_S00_AXI_DATA_WIDTH - 1 downto 0)**
- s00_axi_rresp **out std_logic_vector(1 downto 0)**
- s00_axi_rvalid **out std_logic**
- s00_axi_rready **in std_logic**

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/myGPIO.vhd

7.5 myGPIO_AXI Entity Reference

Periferica AXI4 Lite che implementa una GPIO pilotabile da processing-system.

Diagramma delle classi per myGPIO_AXI

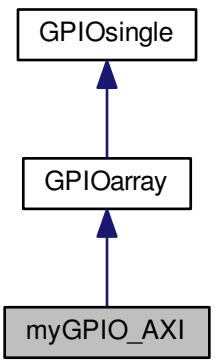
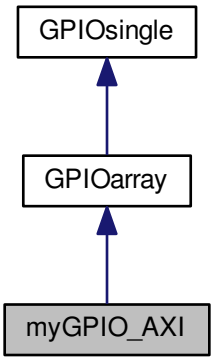


Diagramma di collaborazione per myGPIO_AXI:



Entities

- arch_imp architecture

Libraries

- ieee

Use Clauses

- std_logic_1164
- numeric_std
- std_logic_misc

Generics

- GPIO_width **natural:= 4**

  numero di GPIO offerti dalla periferica, di default pari a 4 celle.
- C_S_AXI_DATA_WIDTH **integer:= 32**
- C_S_AXI_ADDR_WIDTH **integer:= 5**

Ports

- GPIO_inout **inout std_logic_vector(GPIO_width - 1 downto 0)**

  segnale bidirezionale diretto verso l'esterno del device.
- interrupt **out std_logic**

  segnale di interrupt a livelli diretto verso il processing - system. Se le interruzioni sono abilitate ed uno dei pin del device è settato come input ed è abilitato a generare interruzioni, diventa '1' appena tale pin assume valore '1', e mantiene tale valore fino a quando tutte le interruzioni non siano state servite.
- S_AXI_ACLK **in std_logic**
- S_AXI_ARESETN **in std_logic**

- S_AXI_AWADDR **in std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- S_AXI_AWPROT **in std_logic_vector(2 downto 0)**
- S_AXI_AWVALID **in std_logic**
- S_AXI_AWREADY **out std_logic**
- S_AXI_WDATA **in std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- S_AXI_WSTRB **in std_logic_vector((C_S_AXI_DATA_WIDTH/ 8)- 1 downto 0)**
- S_AXI_WVALID **in std_logic**
- S_AXI_WREADY **out std_logic**
- S_AXI_BRESP **out std_logic_vector(1 downto 0)**
- S_AXI_BVALID **out std_logic**
- S_AXI_BREADY **in std_logic**
- S_AXI_ARADDR **in std_logic_vector(C_S_AXI_ADDR_WIDTH- 1 downto 0)**
- S_AXI_ARPROT **in std_logic_vector(2 downto 0)**
- S_AXI_ARVALID **in std_logic**
- S_AXI_ARREADY **out std_logic**
- S_AXI_RDATA **out std_logic_vector(C_S_AXI_DATA_WIDTH- 1 downto 0)**
- S_AXI_RRESP **out std_logic_vector(1 downto 0)**
- S_AXI_RVALID **out std_logic**
- S_AXI_RREADY **in std_logic**

7.5.1 Descrizione dettagliata

Periferica AXI4 Lite che implementa una GPIO pilotabile da processing-system.

Registri interni del device

Il device possiede i registri indicati di seguito. Per oognuno di essi viene indicata la modalità di accesso (R sola lettura, W sola scrittura, R/W lettura scrittura), e l'offset, rispetto all'indirizzo base del device, col quale è possibile indirizzarli.

- MODE (R/W, offset +0x0): consente di impostare i singoli pin del device come ingressi o uscite; solo i GPI↩ O_width bit meno significativi del registro hanno significato, agire sui restanti bit non produce nessun effetto; Il valore che i singoli pin possono assumere è:
  - '1': il pin viene configurato come pin di uscita;
  - 'ò: il pin viene configurato come pin di ingresso;
- WRITE (R/W, offset +0x4): consente di imporre un valore ai pin del device, qualora essi siano configurati come uscite; solo i GPIO_width bit meno significativi del hanno significato, agire sui restanti bit non produce nessun effetto;
- READ (R, offset +0x8): consente di leggere il valore dei pin del device, sia quelli configurati come ingressi che quelli configurati come uscite (il cui valore coincide con quello settato nel registro WRITE); solo i GPIO_width bit meno significativi del registro hanno significato, gli altri vengono letti zero;
- GIES (Global Interrupt Enable/Status, R/W, offset 0xC): Consente di abilitare/disabilitare gli interrupt globali della periferica; solo due dei bit sono significativi:
  - IE (bit 0): interrupt enable, abilita gli interrupt, può essere scritto e letto; se posto ad '1' la periferica potrà generare interrupt quando uno dei pin impostati come ingresso assume valore '1' (ed il corrispondente bit in PIE è impostato ad '1'); se posto a '0' il device non genererà mai interruzioni;
  - IS (bit 1): interrupt status, settato internamente ad '1' nel caso in cui la periferica abbia generato interrupt; replica del segnale "interrupt" diretto verso il processing-system.
- PIE (Pin Interrupt Enable, R/W, offset 0x10): consente di abilitare/disabilitare gli interrupt per i singoli pin. Con GIES(0)='1' e MODE(n)='0' (cioè se gli interrupt globali sono abilitati e il pin n-esimo è configurato come input), se PIE(n)='1' allora il device genererà un interrupt verso il processing-system quando il pin n-esimo assumerà valore '1', mentre, se PIE(n)='0' non verrà generata una interruzione;

- IRQ (Interrupt Request, R, offset 0x14): IRQ(n)='1' indica che la sorgente di interruzione è il bit n-esimo; la or-reduce di tale registro costituisce il flag "interrupt" (IS) di GIES, mentre lo stesso segnale, posto in AND con GIES(0) - interrupt enable - è diretto verso il processing system.
- IACK (Interrupt Ack, W, offset 0x18): imponento IACK(n)='1' è possibile segnalare al device che l'interruzione generata dal in n-esimo è stata servita; il bit IRQ(n) verrà resettato automaticamente.

Process di scrittura dei registri della periferica

Il process che implementa la logica di scrittura dei registri è stato modificato in modo da ottenere il seguente indirizzamento:

Indirizzo	Offset	Registro
b"00000"	0x00	MODE
b"00100"	0x04	WRITE
b"01000"	0x08	READ(∗)
b"01100"	0x0C	GIES(∗∗)
b"10000"	0x10	PIE
b"10100"	0x14	IRQ(∗∗∗)
b"11000"	0x18	IACK(∗∗∗∗)

(∗) Il registro READ è a sola lettura: le scritture su questo registro non producono effetti; la scrittura, infatti, avviene su slv_reg2, che è inutilizzato;
(∗∗) La scrittura ha effetto solo sul bit zero del registro;
(∗∗∗) Il registro IRQ è a sola lettura: le scritture su questo registro non producono effetti; la scrittura, infatti, avviene su slv_reg5, che è inutilizzato;
(∗∗∗∗) La scrittura su IACK è fittizzia, nel senso che appena si smette di indirizzare il registro, esso assume valore zero;

Process di lettura dei registri della periferica

Il process che implementa la logica di lettura dei registri è stato modificato in modo da ottenere il seguente indirizzamento:

Indirizzo	Offset	Registro
b"00000"	0x00	MODE
b"00100"	0x04	WRITE
b"01000"	0x08	READ(∗)
b"01100"	0x0C	GIES(∗∗)
b"10000"	0x10	PIE
b"10100"	0x14	IRQ
b"11000"	0x18	IACK(∗∗∗)

(∗) Il registro READ è direttamente connesso alla porta GPIO_inout
(∗∗) Il bit 2 di GIES è il flag "interrupt", che vale '1' nel caso in cui la periferica abbia generato interrupt ancora non gestiti.
(∗∗∗) Viene letto sempre zero, dal momento che la scrittura su tale registro è fittizzia.

Process di scrittura su IRQ

La logica di scrittura su IRQ è semplice (non viene scritto come un normale registro, ma pilotato internamente dalla periferica): se uno dei bit di GPIO_inout_masked è '1', (la or-reduce è 1) allora il valore del segnale GPIO_inout_↩ masked viene posto in bitwise-or con il valore attuale del registro IRQ, in modo da non resettare i bit di quest' ultimo che siano stati settati a seguito di una interruzione non ancora servita se uno dei bit di IACK è '1' (la or-reduce è '1'), allora il nuovo valore del registro IRQ viene ottenuto

- mascherando IACK con l'attuale valore di IRQ, in modo da non effettuare il set di bit resettati
- ponendo in XOR la maschera precedente con il valore attuale del registro

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/myGPIO_AXI.vhd

7.6 Structural Architecture Reference

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/GPIOsingle.vhd

7.7 Structural Architecture Reference

Components

- GPIOsingle

Instantiations

- single_gpio **GPIOsingle**

La documentazione per questa classe è stata generata a partire dal seguente file:

- Src/myGPIO/VHDL/GPIOarray.vhd

Capitolo 8

Documentazione dei file

8.1 Riferimenti per il file Src/myGPIO/VHDL/GPIOarray.vhd

Entities

- GPIOarray entity

  Array di celle GPIO, pilotabili singolarmente.
- Structural architecture

8.1.1 Descrizione dettagliata

Autore

Salvatore Barone salvator.barone@gmail.com

Data

07 04 2017

Copyright

8.2 Riferimenti per il file Src/myGPIO/VHDL/GPIOsingle.vhd

Entities

- GPIOsingle entity

  Cella base GPIO.
- Structural architecture

8.2.1 Descrizione dettagliata

Autore

Salvatore Barone salvator.barone@gmail.com

Data

07 04 2017

Copyright

8.3 Riferimenti per il file Src/myGPIO/VHDL/myGPIO.vhd

Entities

- myGPIO entity

8.3.1 Descrizione dettagliata

Autore

Salvatore Barone salvator.barone@gmail.com

Data

22 06 2017

Copyright

8.4 Riferimenti per il file Src/myGPIO/VHDL/myGPIO_AXI.vhd

Entities

- myGPIO_AXI entity

  Periferica AXI4 Lite che implementa una GPIO pilotabile da processing-system.
- arch_imp architecture

8.4.1 Descrizione dettagliata

Autore

Salvatore Barone salvator.barone@gmail.com

Data

22 06 2017

Copyright

Indice analitico