

# Tesina di Sistemi Embedded Gruppo IV

Colella Gianni - Mat. M63/670      Guida Ciro - Mat. M63/592  
Lombardi Daniele - Mat. M63/576

13 luglio 2017

# Indice

<b>1 Gpio custom su Zybo</b>	<b>1</b>
1.1 Traccia . . . . .	1
1.2 Procedimento . . . . .	2
1.2.1 Zynq + Gpio . . . . .	2
1.2.1.1 Creazione del progetto . . . . .	2
1.2.1.2 Creazione IP-Core . . . . .	3
1.2.1.3 Codice GPIO . . . . .	7
1.2.1.4 Inclusione GPIO nella periferica AXI 4 . . . . .	10
1.2.1.5 Block Design . . . . .	15
1.2.1.6 Sintesi e Implementazione . . . . .	18
1.2.1.7 SDK Tool . . . . .	19
1.2.2 Driver + Applicazione . . . . .	23
1.2.2.1 Libreria HAL driver . . . . .	23
1.2.2.2 Driver . . . . .	30
1.2.2.3 Versione object oriented . . . . .	32
1.3 Supporto per le interruzioni . . . . .	39
1.3.1 Modifica IP core . . . . .	39
1.3.2 Modifica libreria HAL driver . . . . .	41
<b>2 BSP custom per STM32 F4 Discovery</b>	<b>56</b>
2.1 Traccia . . . . .	56
2.2 Procedimento . . . . .	56
2.2.1 Creazione di un nuovo progetto . . . . .	56
2.2.2 Funzioni . . . . .	60
2.2.2.1 init_led_user . . . . .	61
2.2.2.2 init_button_user . . . . .	62
2.2.2.3 init_timer2 . . . . .	63
2.2.2.4 write_led . . . . .	63
2.2.2.5 read_button . . . . .	64
2.2.2.6 start_timer2 . . . . .	64
2.2.2.7 is_timer2_stopped . . . . .	64
2.2.3 Driver . . . . .	65
2.2.3.1 toggleAll . . . . .	66
2.2.3.2 counter . . . . .	66
2.2.3.3 supercar . . . . .	67
2.2.4 Applicazione . . . . .	68

---

<b>3 Esempi di utilizzo delle interruzioni su board STM</b>	<b>70</b>
3.1 Traccia . . . . .	70
3.2 Procedimento . . . . .	70
3.2.1 Soluzione bare metal . . . . .	70
3.2.2 Soluzione HAL . . . . .	72
3.2.3 Interrupt su EXTI1 . . . . .	74
<b>4 Bus Seriali</b>	<b>77</b>
4.1 Traccia . . . . .	77
4.2 Procedimento . . . . .	77
4.2.1 SPI . . . . .	77
4.2.2 I2C . . . . .	83
4.2.3 UART . . . . .	94
<b>5 Esempi di utilizzo dell'Universal Serial Bus</b>	<b>99</b>
5.1 Traccia . . . . .	99
5.2 Procedimento . . . . .	99
5.2.1 Invio dati seriali . . . . .	99
5.2.2 Device USB . . . . .	101
5.2.2.1 init_mouse . . . . .	102
5.2.2.2 x_position . . . . .	103
5.2.2.3 y_position . . . . .	104
5.2.2.4 position_on_led . . . . .	104
5.2.2.5 Driver . . . . .	105
<b>6 Kernel Linux</b>	<b>107</b>
6.1 Traccia . . . . .	107
6.2 Procedimento . . . . .	107
6.2.1 Requisiti . . . . .	107
6.2.2 Workspace setting . . . . .	108
6.2.3 Sintesi Hardware . . . . .	109
6.2.4 Compilazione U-boot . . . . .	113
6.2.5 Generazione FSBL . . . . .	115
6.2.6 Generazione BOOT.bin . . . . .	117
6.2.7 Generazione uImage . . . . .	118
6.2.8 Generazione del DTB . . . . .	119
6.2.9 Partizionamento SD card . . . . .	120
6.2.10 Settaggio SD card per il boot . . . . .	122
6.2.11 Test di boot . . . . .	125
<b>7 Driver Linux</b>	<b>127</b>
7.1 Traccia . . . . .	127
7.2 Procedimento: GPIO Xilinx . . . . .	127
7.2.1 Prima tipologia . . . . .	127
7.2.1.1 nodriver_header.h . . . . .	127
7.2.1.2 nodriver_function.h . . . . .	129

---

7.2.1.3	main.c . . . . .	135
7.2.1.4	makefile . . . . .	136
7.2.1.5	Esempio di esecuzione . . . . .	137
7.2.2	Seconda tipologia . . . . .	139
7.2.2.1	uiodriver_header.h . . . . .	143
7.2.2.2	uiodriver_function.h . . . . .	144
7.2.2.3	main.c . . . . .	150
7.2.2.4	makefile . . . . .	151
7.2.2.5	Esempio di esecuzione . . . . .	152
7.2.3	Terza tipologia . . . . .	153
7.2.3.1	uiointdriver_header.h . . . . .	156
7.2.3.2	uiointdriver_function.h . . . . .	158
7.2.3.3	main.c . . . . .	161
7.2.3.4	makefile . . . . .	161
7.2.3.5	Esempio di esecuzione . . . . .	162
7.3	Procedimento: GPIO custom . . . . .	162
7.3.1	Prima tipologia . . . . .	162
7.3.1.1	mygpio_nodriver.h . . . . .	163
7.3.1.2	mygpio_nodriver.h . . . . .	165
7.3.1.3	main.c . . . . .	169
7.3.1.4	makefile . . . . .	170
7.3.2	Seconda tipologia . . . . .	170
7.3.2.1	mygpio_uiodriver.h . . . . .	171
7.3.2.2	mygpio_uiodriver.c . . . . .	172
7.3.2.3	main.c . . . . .	176
7.3.3	Terza tipologia . . . . .	177
7.3.3.1	mygpio_uiointdriver.c . . . . .	178
<b>8</b>	<b>FreeRTOS</b>	<b>184</b>
8.1	Traccia . . . . .	184
8.2	Procedimento . . . . .	184
8.2.1	Esempi con 2 Task . . . . .	186
8.2.1.1	Esempio 1 . . . . .	186
8.2.1.2	Esempio 2 . . . . .	188
8.2.1.3	Esempio 3 . . . . .	189
8.2.2	Esempi con 3 Task . . . . .	190
8.2.2.1	Esempio 4 . . . . .	190
8.2.2.2	Esempio 5 . . . . .	191
8.2.2.3	Esempio 6 . . . . .	192
8.2.3	Esempi con 4 Task . . . . .	193
8.2.3.1	Esempio 7 . . . . .	193
8.2.3.2	Esempio 8 . . . . .	195
8.2.3.3	Esempio 9 . . . . .	196
8.2.3.4	Esempio 10 . . . . .	198
8.3	Considerazioni . . . . .	199

---

# Capitolo 1

## Gpio custom su Zybo

### 1.1 Traccia

Mostrare come utilizzare un General Purpose Input/Output per pilotare i led della board Digilent Zybo Zynq-7000. Per la realizzazione di tale obiettivo, si divida il lavoro in due parti. Nella prima parte, dopo aver definito manualmente un componente GPIO lo si impacchetti in un IP-Core e lo si utilizzi insieme al Processing System Zynq. Nella seconda parte, invece, in aggiunta all'architettura già creata, si introduca uno strato driver, utile al controllo dei led. Si segua, pertanto, lo schema di progetto in fig. 6.1.

Successivamente a partire dall'IP core creato, si aggiunga un supporto custom che permetta alla periferica di lanciare interruzioni verso il PS.

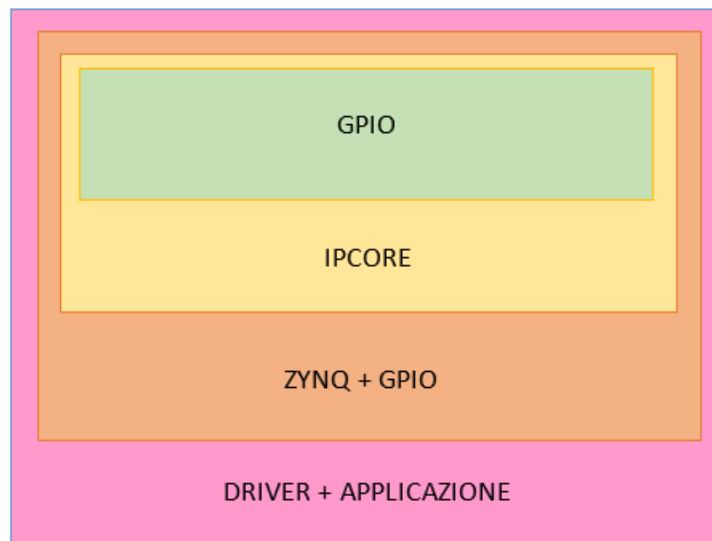


Figura 1.1: Schema di progetto

## 1.2 Procedimento

### 1.2.1 Zynq + Gpio

#### 1.2.1.1 Creazione del progetto

Aprire l'ambiente Vivado 2016.4. Prima di iniziare, assicurarsi che in **Tools → Options... → General** il Target Language sia impostato su VHDL, fig. 1.2.

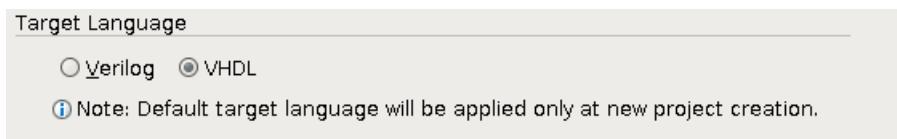


Figura 1.2: Target Language

Nella schermata principale, cliccare su **Create New Project**. Viene aperto un wizard in cui bisogna definire il nome del progetto, *gpio\_ptoject*. Andando avanti, viene chiesto di specificare il tipo di progetto da creare. Selezionare **RTL Project**, spuntando l'opzione *do not specify source at this time*, per indicare che, in questa fase, non si vogliono caricare i file sorgente, fig. 1.3.

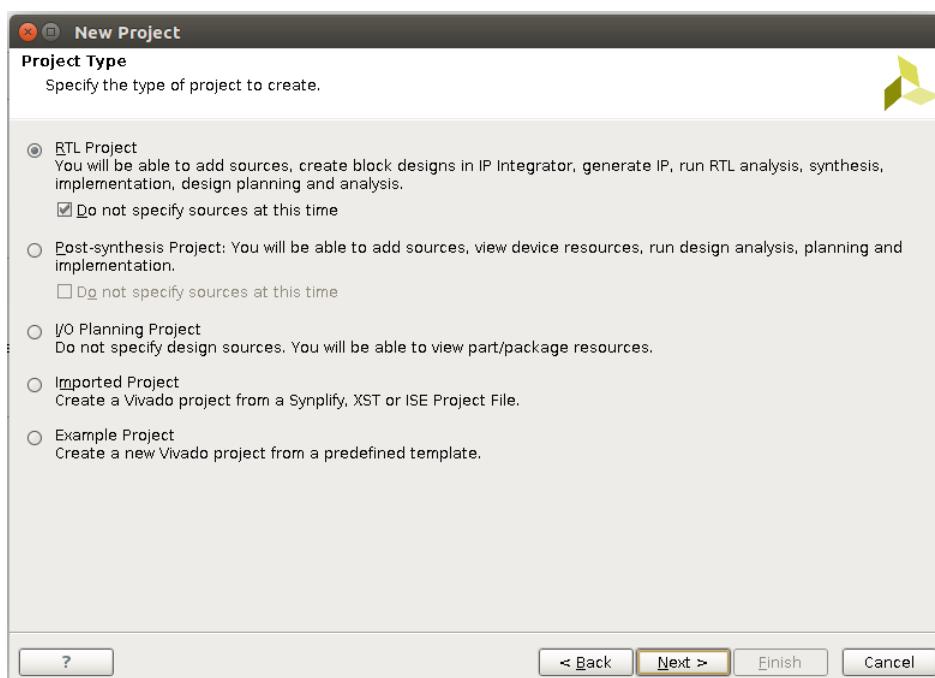


Figura 1.3: Project Type

Nella finestra successiva, viene chiesto di selezionare la board o le parti di default da associare al progetto che si sta creando. Cliccare su **Boards** e selezionare Zybo, fig. 1.4.

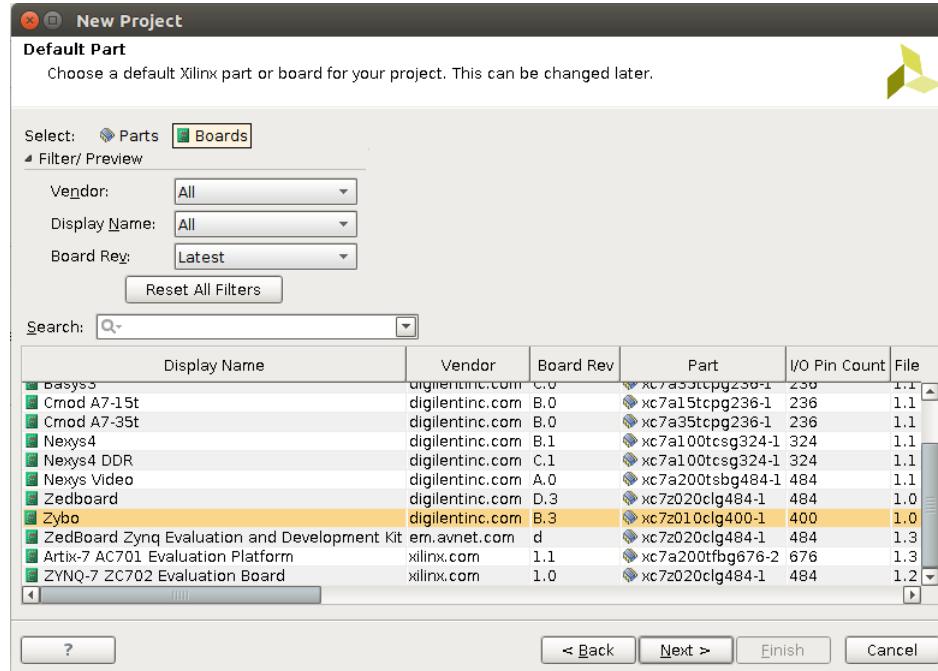


Figura 1.4: Default Part

### 1.2.1.2 Creazione IP-Core

Creato il progetto, bisogna istanziare una nuova periferica AXI4 al cui interno verrà definito successivamente il componente GPIO. A tal fine, si clicca su **Tools → Create and Package New IP** e si specifica che si sta creando una Periferica AXI4 spuntando *Create a new AXI4 peripheral*, come mostrato in fig. 1.5.

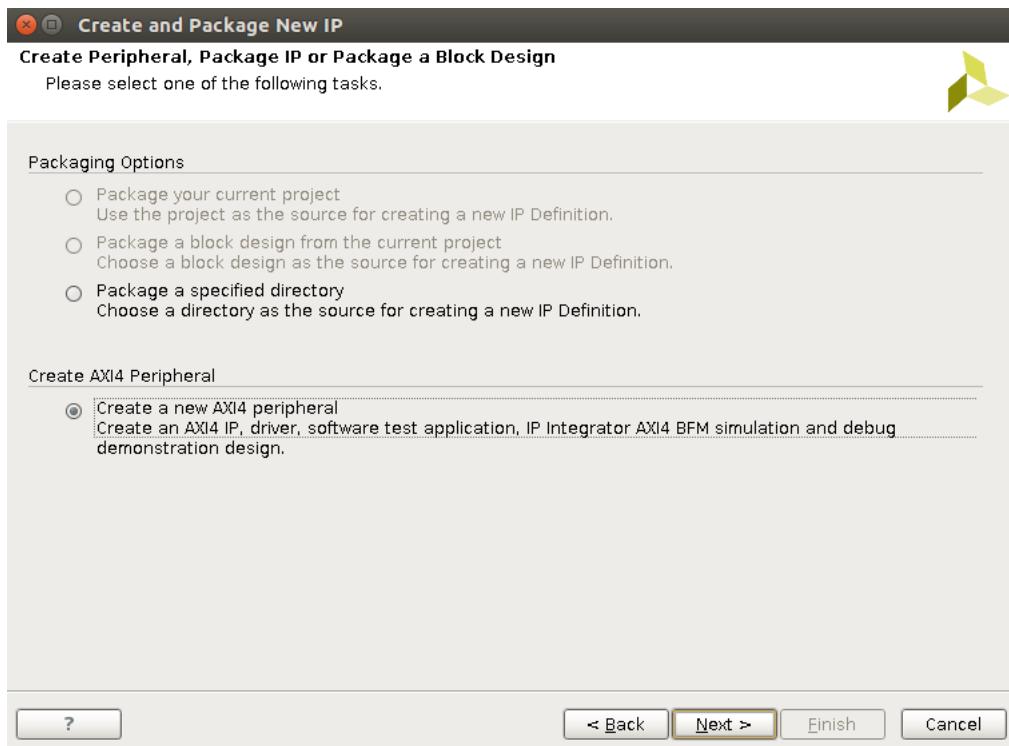


Figura 1.5: Create Peripheral, Package IP or Package a Block Design

Quindi, si settano: il nome dell'IP; versione; nome da visualizzare a display; descrizione; locazione in cui salvarlo.

In fig. 1.6. è possibile aggiungere più interfacce alla periferica che si sta creando. Nel caso preso in questione, ne occorre una soltanto. Si noti, inoltre, che bisogna specificare:

- *Name*, nome dell'interfaccia, *S00\_AXI*;
- *Interface Type*, tipo di interfaccia da utilizzare. Poichè il progetto non è eccessivamente complesso, si scegli *Lite*;
- *Interface mode*, specifica se la periferica che si sta creando deve agire da master o slave. Si sceglie *Slave*;
- *Data width (Bits)*, indica il parallelismo utilizzato dai registri della periferica. Si noti che, avendo utilizzato come target la Zybo della Digilent, tale parametro necessariamente dev'essere settato a *32*;
- *Number of Registers*, è il numero di registri richiesto alla periferica. Considerando che Vivado, quando istanzia una nuova periferica, riserva 16 byte come spazio di indirizzamento, e che, la board Zybo necessita obbligatoriamente di registri a 32 bit, allora tale numero deve essere almeno *4*.

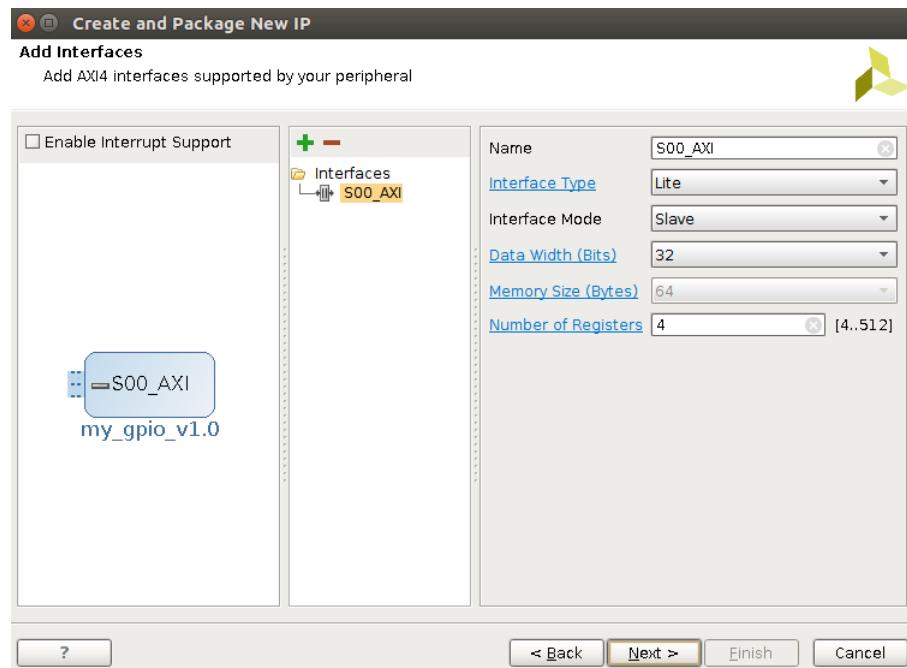


Figura 1.6: Add Interfaces

Dopo aver cliccato su **Next**, si va a selezionare **EditIP**. A questo punto, si aggiunge al progetto il file VHDL, in cui si va a creare il GPIO. Cliccare, quindi, **File → Add Source** e poi su **Add or Create design source**. Andare su **Create File**, specificare VHDL per il File Type, come nome *gpio* e come locazione in cui salvarlo *ip\_repo/my\_gpio\_1.0/hdl*, fig. 1.7.

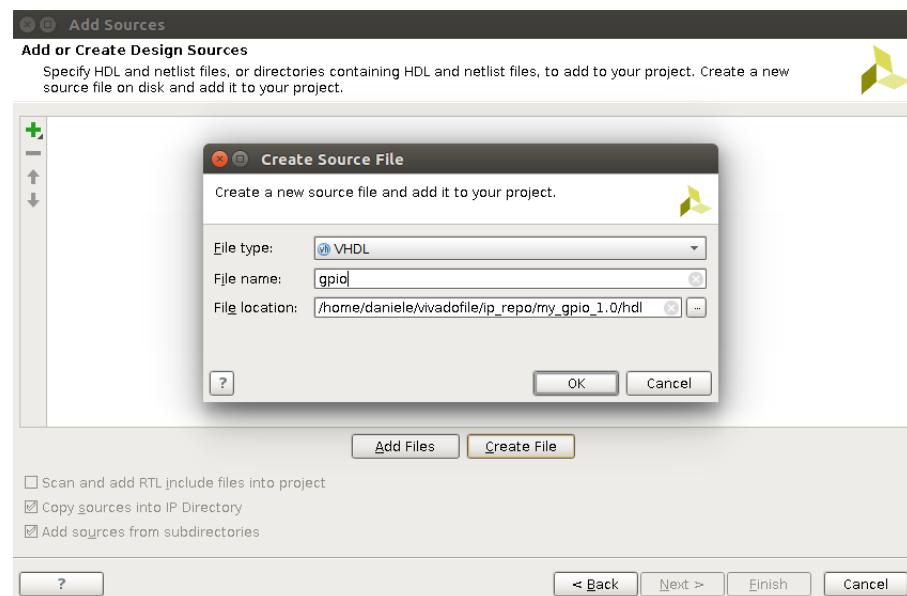


Figura 1.7: Add or Create Design Sources

Cliccando **Finish**, si apre un wizard per definire l'interfaccia del modulo. Con questa finestra, alla stessa maniera del tool Xilinx ISE, si può descrivere l'interfaccia del componente che si sta creando, in modo tale che il nuovo file .vhd che viene generato abbia già delle sezioni pre-configure, fig. 1.8.

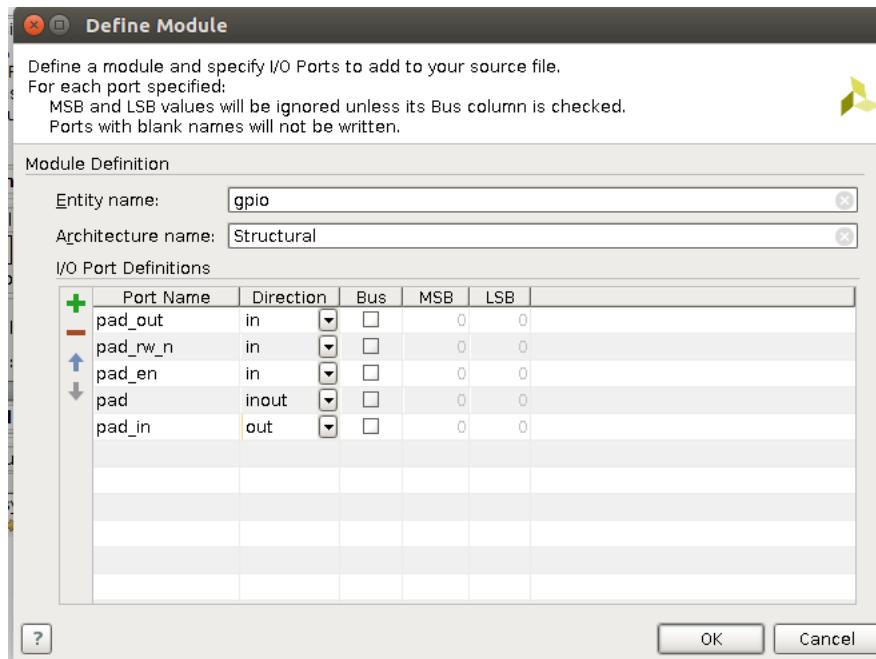


Figura 1.8: Define Module di gpio.vhd

Si noti che, nel Design Source, è comparso il file che si sta per creare, fig. 1.9.

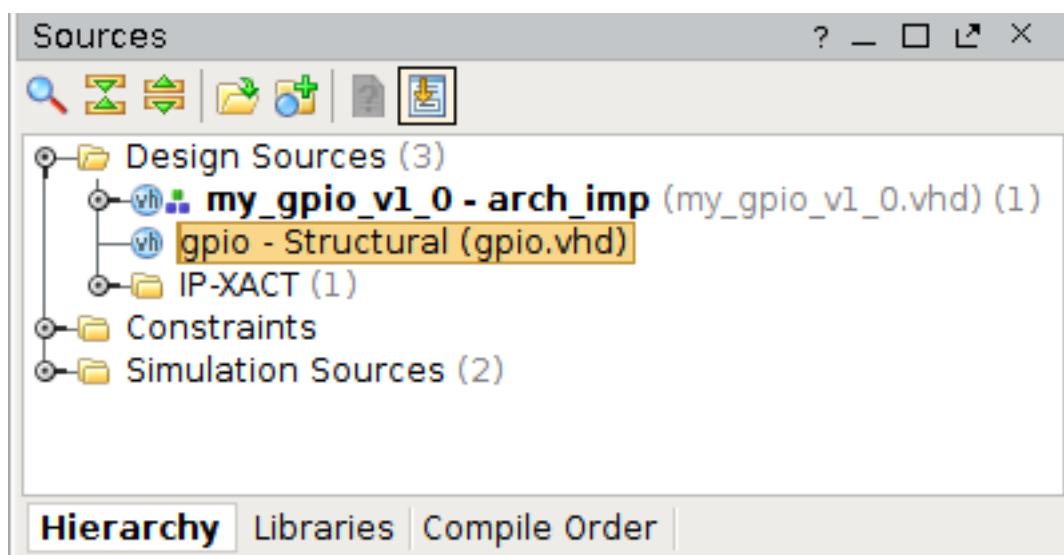


Figura 1.9: File gpio.vhd

### 1.2.1.3 Codice GPIO

Con riferimento allo schema semplificato di fig. 1.10, si ricorda che il componente GPIO presenta un segnale *pad* che può funzionare sia da input e sia da output. Quando *pad\_en* è attivo, il segnale *pad\_out* viene propagato su *pad*, che funge, quindi da segnale di output. Diversamente, quando *pad\_en* non è attivo, il buffer tri-state, comportandosi da interruttore, propaga alta impedenza. In entrambi i casi, il segnale di *pad* viene propagato su *pad\_in*.

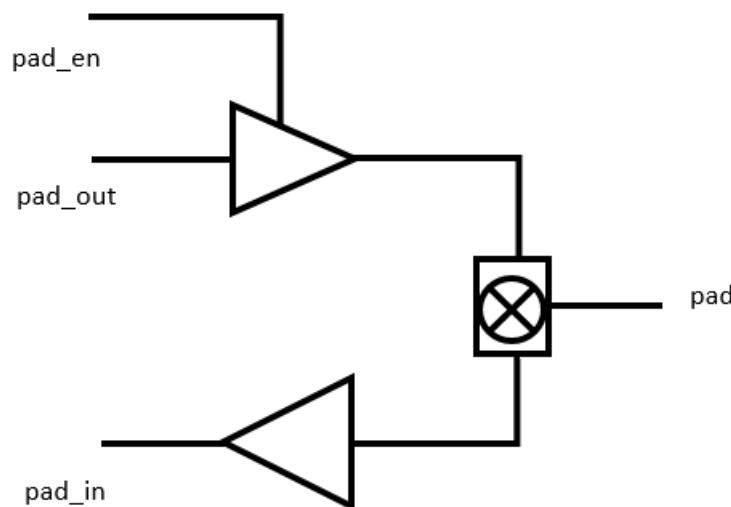


Figura 1.10: Schema semplificato componente GPIO

Partendo dallo schema sopra e aggiungendo un ulteriore segnale di controllo *pad\_rw\_n*, di seguito, si scrive il codice per implementare il componente GPIO, riportato di seguito.

```

1
2
3 -- Company: Gruppo IV - Sistemi Embedded 2016-17
4 -- Engineer: Colella Gianni, Guida Ciro, Lombardi Daniele
5
6 -- Create Date: 10.05.2017 12:24:39
7 -- Module Name: gpio - Dataflow
8 -- Target Devices: Zynq Z-7010
9 -- Tool Versions: Vivado 2016.4
10
11
12
13
14 library IEEE;

```

```

15 use IEEE.STD_LOGIC_1164.ALL;
16
17
18 entity gpio is
19   Port ( pad_out : in STD_LOGIC;
20         pad_rw_n : in STD_LOGIC;
21         pad_en : in STD_LOGIC;
22         pad_in : out STD_LOGIC;
23         pad : inout STD_LOGIC);
24 end gpio;
25
26 architecture DataFlow of gpio is
27
28 begin
29
30 pad_in <= pad when pad_en='1' else '0';
31 pad <= pad_out and pad_en when pad_rw_n = '0' else 'Z';
32
33 end DataFlow;

```

Codice 1.1: gpio.vhd

Dovendo gestire complessivamente 8 segnali (4 per i led, 4 per gli switch), si va ad implementare un componente generico che implementa tante istanze singole di GPIO quante ne sono specificate nel generic del codice. Si ripete lo stesso procedimento visto sopra, fig. 1.11.

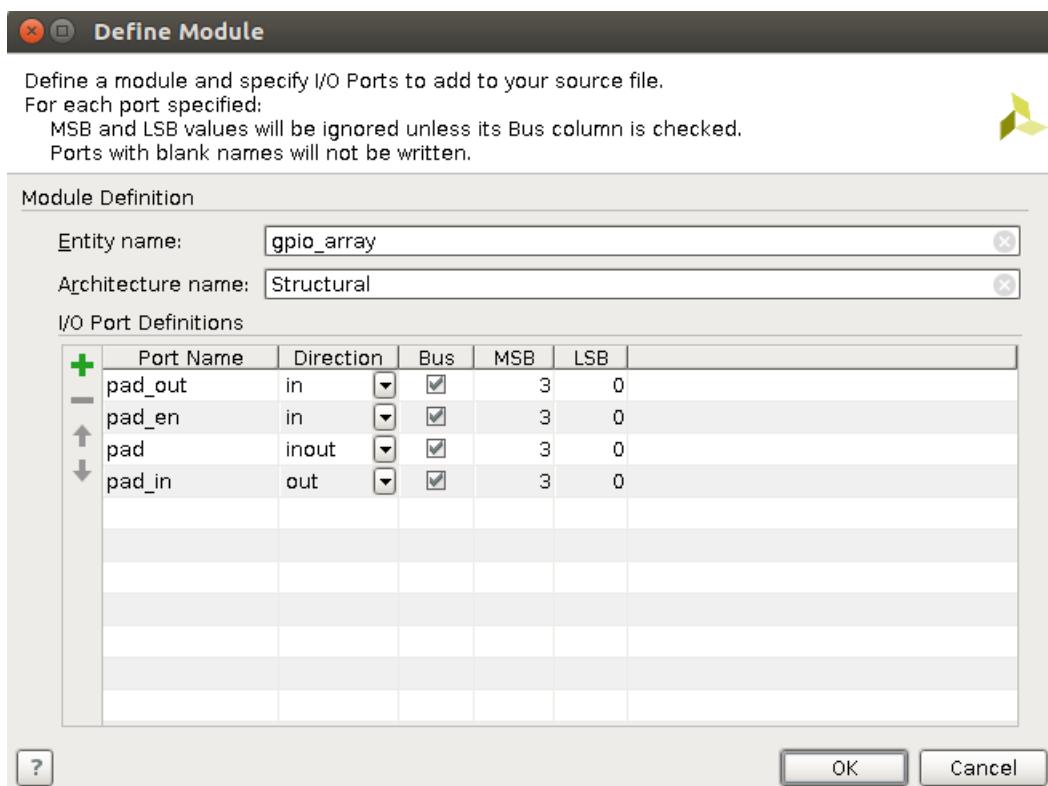


Figura 1.11: Define Module gpio\_array

Si riporta, per completezza, il codice utilizzato.

```

1-----+
2-- Company: Gruppo IV - Sistemi Embedded 2016-17
3-- Engineer: Colella Gianni, Guida Ciro, Lombardi Daniele
4--
5-- Create Date: 10.05.2017 12:34:37
6-- Module Name: gpio_array - Structural
7-- Target Devices: Zynq Z-7010
8-- Tool Versions: Vivado 2016.4
9--
10-----+
11
12
13 library IEEE;
14 use IEEE.STD_LOGIC_1164.ALL;
15
16
17 entity gpio_array is
18     Generic ( gpio_size : natural := 8);
19     Port ( pad_out : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
20             pad_rw_n : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
21             pad_en : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
22             pad_in : out STD_LOGIC_VECTOR (gpio_size-1 downto 0);
23             pad : inout STD_LOGIC_VECTOR (gpio_size-1 downto 0));
24 end gpio_array;
25
26 architecture Structural of gpio_array is
27
28 component gpio is
29     Port ( pad_out : in STD_LOGIC;
30             pad_rw_n : in STD_LOGIC;
31             pad_en : in STD_LOGIC;
32             pad_in : out STD_LOGIC;
33             pad : inout STD_LOGIC);
34 end component;
35
36 begin
37
38 MULTI_GPIO : for i in 0 to gpio_size-1 generate
39     SINGLE_GPIO : gpio port map (
40         pad_rw_n=>pad_rw_n(i),
41         pad_out => pad_out(i),
42         pad_en => pad_en(i),
43         pad => pad(i),
44         pad_in => pad_in(i));
45
46     end generate;
47 end Structural;

```

---

Codice 1.2: gpio array.vhd

### 1.2.1.4 Inclusione GPIO nella periferica AXI 4

Aprendo l'interfaccia della periferica AXI 4, *my\_gpio\_v1\_0\_S00\_AXI*, si nota che tutto il codice è già stato predisposto dal tool. Lo si va a modificare opportunamente per includere il componente GPIO appena creato nella periferica AXI 4. In particolare

- in *User to Add parameters here*, si aggiunge il codice di fig. 1.12;
- in *Users to Add ports here*, si aggiunge il codice di fig. 1.12;
- prima del *begin* si aggiunge il codice di fig. 1.13, che include il *component gpio\_array* e definisce il segnale *periph\_pad\_in*;
- nel process che gestisce *reg\_data\_out*, si aggiunge il codice di fig. 1.14;
- in *Add user logic here*, si aggiunge il codice di fig. 1.15.

```

my_gpio_v1_0_S00_AXI.vhd *
/home/gianni/vivado_ok/project_gpio/my_gpio_1.0/hdl/my_gpio_v1_0_S00_AXI.vhd

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity my_gpio_v1_0_S00_AXI is
6     generic (
7         -- Users to add parameters here
8         gpio_size: natural := 8;
9         -- User parameters ends
10        -- Do not modify the parameters beyond this line
11
12        -- Width of S_AXI data bus
13        C_S_AXI_DATA_WIDTH : integer := 32;
14        -- Width of S_AXI address bus
15        C_S_AXI_ADDR_WIDTH : integer := 4
16    );
17    port (
18        -- Users to add ports here
19        pad : inout STD_LOGIC_VECTOR (gpio_size-1 downto 0);
20
21        -- User ports ends
22        -- Do not modify the ports beyond this line
23
24        -- Global Clock Signal
25        S_AXI_ACLK : in std_logic;

```

Figura 1.12: Parte I codice interfaccia AXI 4

```

my_gpio_v1_0_S00_AXI.vhd*
/home/gianni/vivado_ok/project_gpio/my_gpio_1.0/hdl/my_gpio_v1_0_S00_AXI.vhd

121
122 component gpio_array is
123     Generic ( gpio_size : natural := 8);
124     Port ( pad_out : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
125             pad_rw_n : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
126             pad_en : in STD_LOGIC_VECTOR (gpio_size-1 downto 0);
127             pad_in : out STD_LOGIC_VECTOR (gpio_size-1 downto 0);
128             pad : inout STD_LOGIC_VECTOR (gpio_size-1 downto 0));
129 end component;
130
// 131 signal periph_pad_in: std_logic_vector(gpio_size-1 downto 0);
132

```

Figura 1.13: Parte II codice interfaccia AXI 4

```

355
356 process (slv_reg0, slv_reg1, slv_reg2, periph_pad_in, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
357 variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
358 begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
        when b"00" =>
            reg_data_out <= slv_reg0;
        when b"01" =>
            reg_data_out <= slv_reg1;
        when b"10" =>
            reg_data_out <= slv_reg2;
        when b"11" =>
            reg_data_out <= slv_reg3;
            reg_data_out <= (others => '0');
            reg_data_out(gpio_size-1 downto 0) <= periph_pad_in;
    end case;
end process;

```

Figura 1.14: Parte III codice interfaccia AXI 4

```

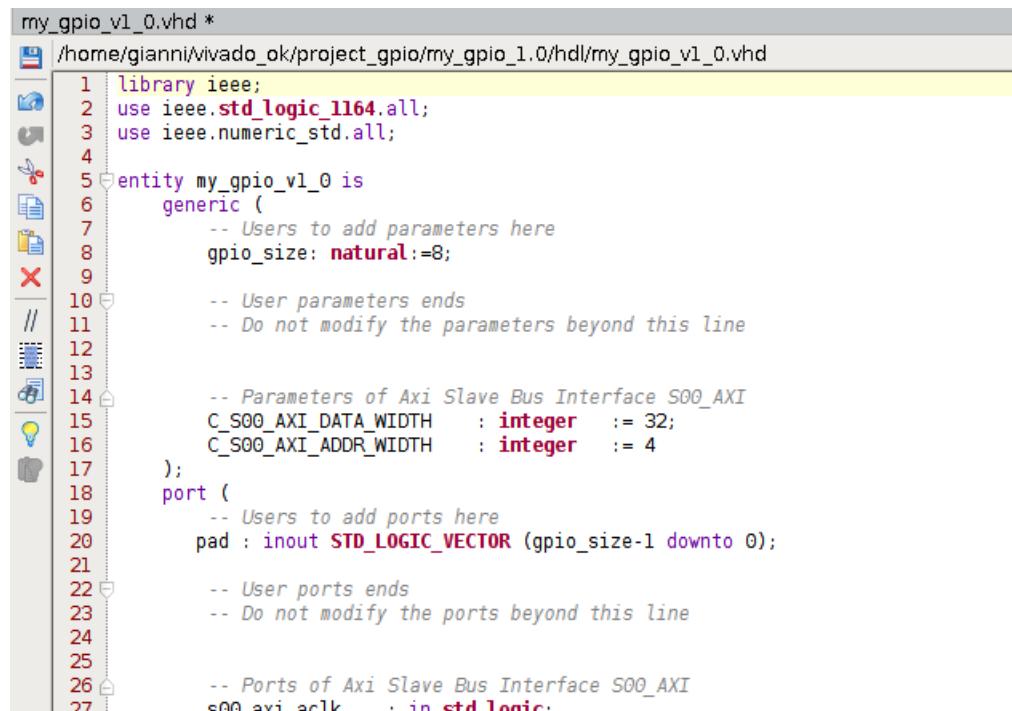
my_gpio_v1_0_S00_AXI.vhd*
/home/gianni/vivado_ok/project_gpio/my_gpio_1.0/hdl/rny_gpio_v1_0_S00_AXI.vhd

396
397     -- Add user logic here
398     GPIO_INST: gpio_array generic map(gpio_size)
399         port map(
400             pad_out=>slv_reg0(gpio_size-1 downto 0),
401             pad_rw_n=>slv_reg1(gpio_size-1 downto 0),
402             pad_en=>slv_reg2(gpio_size-1 downto 0),
403             pad_in=>periph_pad_in,
404             pad=>pad
405         );
406
407     -- User logic ends
408
409 end arch_imp;
410

```

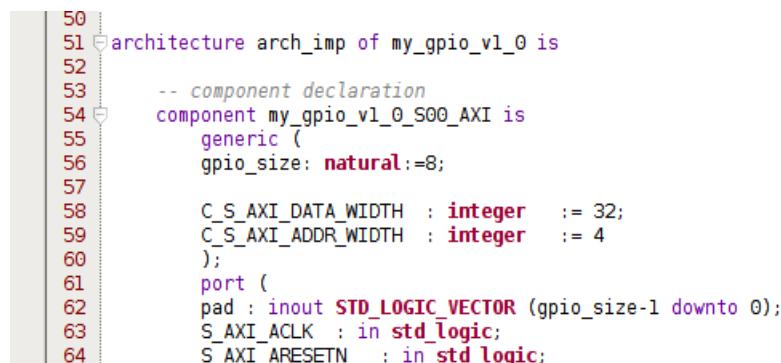
Figura 1.15: Parte IV codice interfaccia AXI 4

A questo punto, si modifica opportunamente anche la top level entity [my\\_gpio\\_v1\\_0](#), da fig. 1.16 a fig. 1.18.



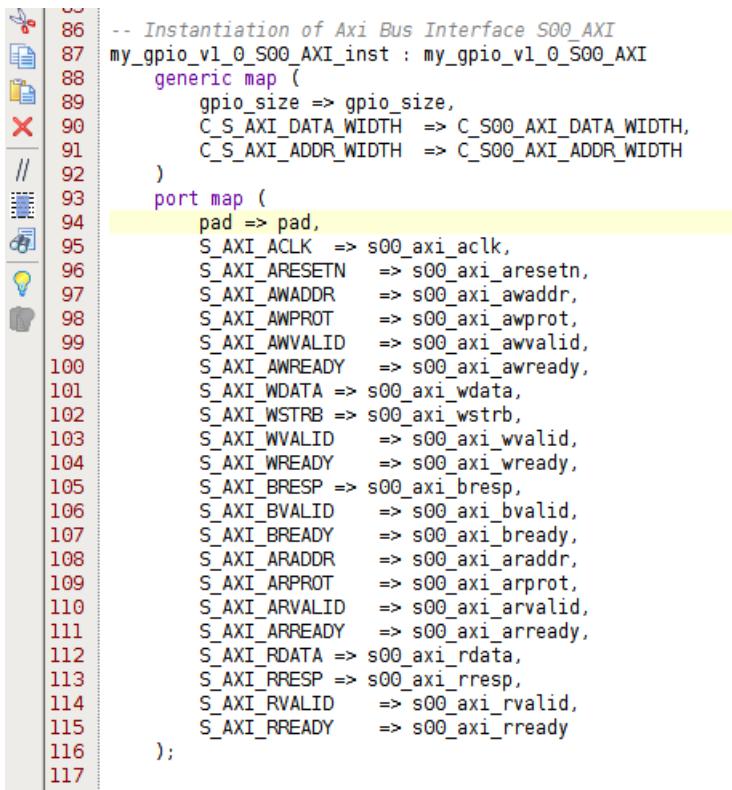
```
my_gpio_v1_0.vhd *
/home/gianni/vivado_ok/project_gpio/my_gpio_1.0/hdl/my_gpio_v1_0.vhd
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity my_gpio_v1_0 is
6 generic (
7     -- Users to add parameters here
8     gpio_size: natural:=8;
9
10    -- User parameters ends
11    -- Do not modify the parameters beyond this line
12
13
14    -- Parameters of Axi Slave Bus Interface S00_AXI
15    C_S00_AXI_DATA_WIDTH : integer := 32;
16    C_S00_AXI_ADDR_WIDTH : integer := 4
17 );
18 port (
19     -- Users to add ports here
20     pad : inout STD_LOGIC_VECTOR (gpio_size-1 downto 0);
21
22    -- User ports ends
23    -- Do not modify the ports beyond this line
24
25
26    -- Ports of Axi Slave Bus Interface S00_AXI
27    S00_AXI_ACLK : in std_logic;
```

Figura 1.16: Parte I codice Top level entity



```
50
51 architecture arch_imp of my_gpio_v1_0 is
52
53    -- component declaration
54    component my_gpio_v1_0_S00_AXI is
55        generic (
56            gpio_size: natural:=8;
57
58            C_S_AXI_DATA_WIDTH : integer := 32;
59            C_S_AXI_ADDR_WIDTH : integer := 4
60        );
61        port (
62            pad : inout STD_LOGIC_VECTOR (gpio_size-1 downto 0);
63            S_AXI_ACLK : in std_logic;
64            S_AXI_ARESETN : in std_logic;
```

Figura 1.17: Parte II codice Top level entity



```

86 -- Instantiation of Axi Bus Interface S00_AXI
87 my_gpio_v1_0_S00_AXI_inst : my_gpio_v1_0_S00_AXI
88   generic map (
89     gpio_size => gpio_size,
90     C_S_AXI_DATA_WIDTH => C_S00_AXI_DATA_WIDTH,
91     C_S_AXI_ADDR_WIDTH => C_S00_AXI_ADDR_WIDTH
92   )
93   port map (
94     pad => pad,
95     S_AXI_ACLK => s00_axi_aclk,
96     S_AXI_ARESETN => s00_axi_aresetn,
97     S_AXI_AWADDR => s00_axi_awaddr,
98     S_AXI_AWPROT => s00_axi_awprot,
99     S_AXI_AWVALID => s00_axi_awvalid,
100    S_AXI_AWREADY => s00_axi_awready,
101    S_AXI_WDATA => s00_axi_wdata,
102    S_AXI_WSTRB => s00_axi_wstrb,
103    S_AXI_WVALID => s00_axi_wvalid,
104    S_AXI_WREADY => s00_axi_wready,
105    S_AXI_BRESP => s00_axi_bresp,
106    S_AXI_BVALID => s00_axi_bvalid,
107    S_AXI_BREADY => s00_axi_bready,
108    S_AXI_ARADDR => s00_axi_araddr,
109    S_AXI_ARPROT => s00_axi_arprot,
110    S_AXI_ARVALID => s00_axi_arvalid,
111    S_AXI_ARREADY => s00_axi_arready,
112    S_AXI_RDATA => s00_axi_rdata,
113    S_AXI_RRESP => s00_axi_rrresp,
114    S_AXI_RVALID => s00_axi_rvalid,
115    S_AXI_RREADY => s00_axi_rready
116  );
117

```

Figura 1.18: Parte III codice Top level entity

Da **Flow Navigator** → **Package IP** andare su **File Groups** e cliccare su *Merge changes from File Groups Wizard*, fig. 1.19.

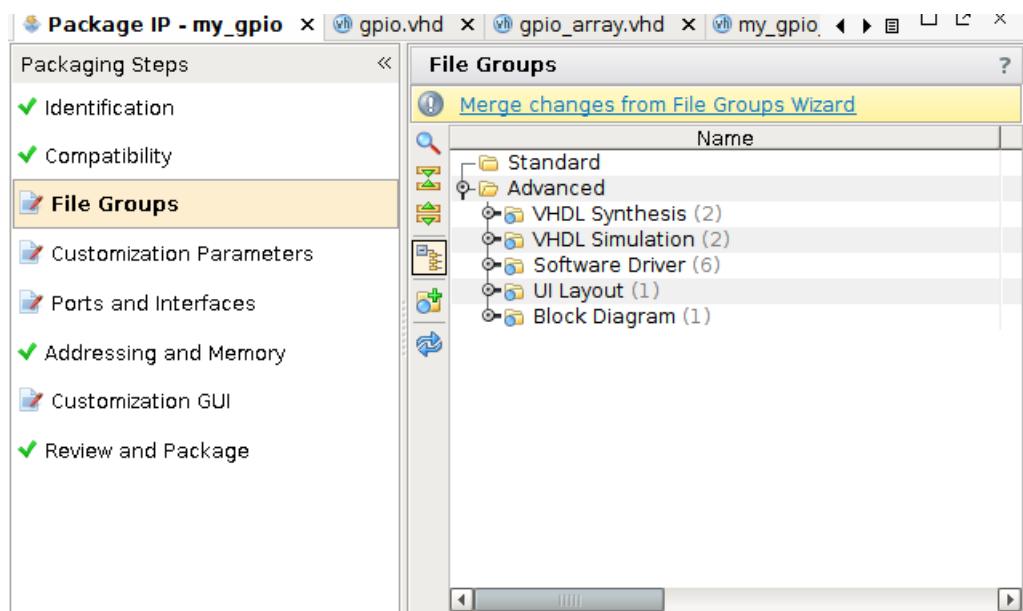


Figura 1.19: File Groups

Su **Customization Parameters**, si clicca su *Merge changes from Customization Parameters Wizard*, fig. 1.20.

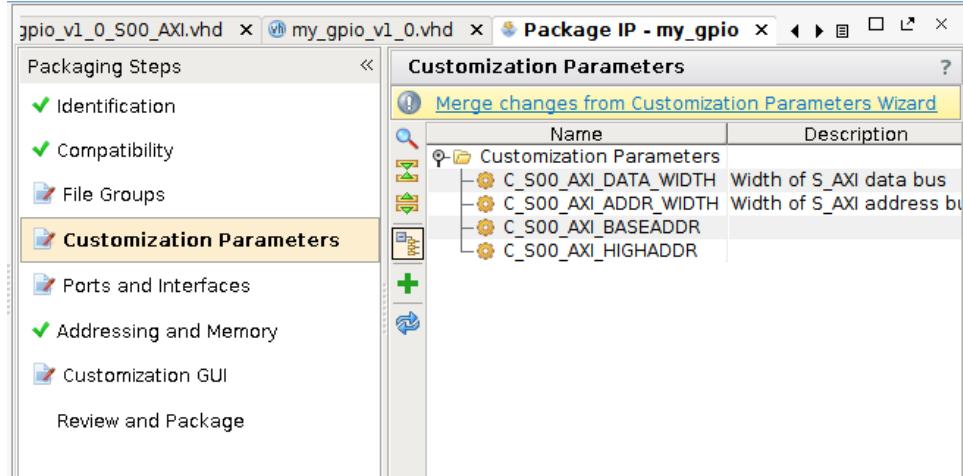


Figura 1.20: Customization Parameters

Adesso bisogna importare i parametri nascosti. Si clicca due volte sul parametro *gpio\_size* (comparso sotto Hidden Parameters) e si spunta l'opzione *Visible in Customization GUI*, fig. 1.21.

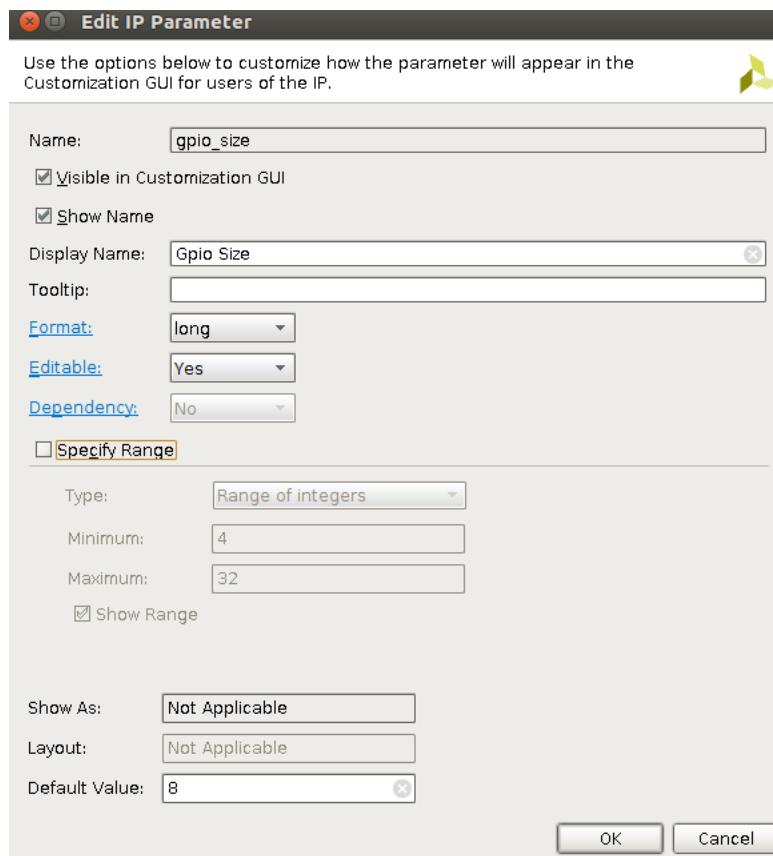


Figura 1.21: Edit IP Parameter

In **Customization GUI**, nella view Layout, si trascina *Gpio Size* tra gli altri parametri.

Per impacchettare il nuovo l'IP, nella sezione **Review and Package** cliccare su *Re-Package IP*. Si chiude, infine, l'istanza di Vivado.

### 1.2.1.5 Block Design

Da **Flow Navigator** → **IP Integrator** → **Create Block Design**, definire il nome *design\_for\_gpio*. Come mostrato in fig. 1.22, cliccare su **Add IP** per aggiungere nuovi IP-core al progetto.

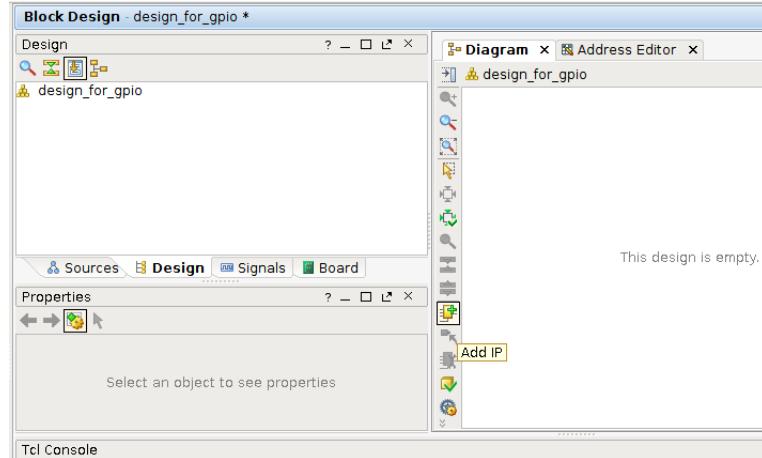


Figura 1.22: Add IP to Block Design

Dalla lista visualizzata, aggiungere *ZYNQ7 Processing System* e *my\_gpio\_v1.0*, vale a dire l'IP-core appena creato. Se non compare l'IP creato, allora andare su **Tools** → **Project Settings** → **IP** → **Repository Manager**, cliccare su **Add** ed inserire il path del repository in cui è stato definito l'IP, come in fig. 1.23.

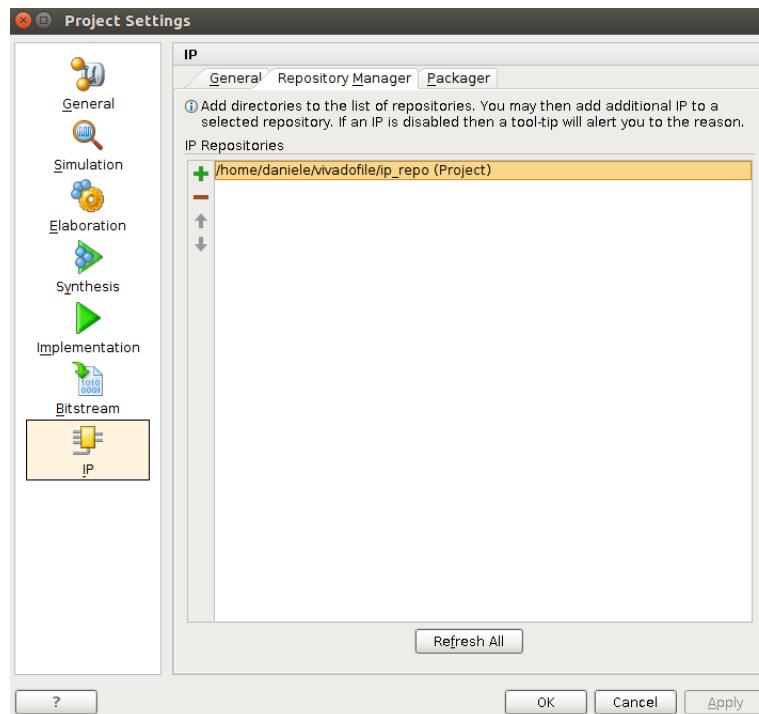


Figura 1.23: Select path repository

Si ottiene il Diagram di fig. 1.24.

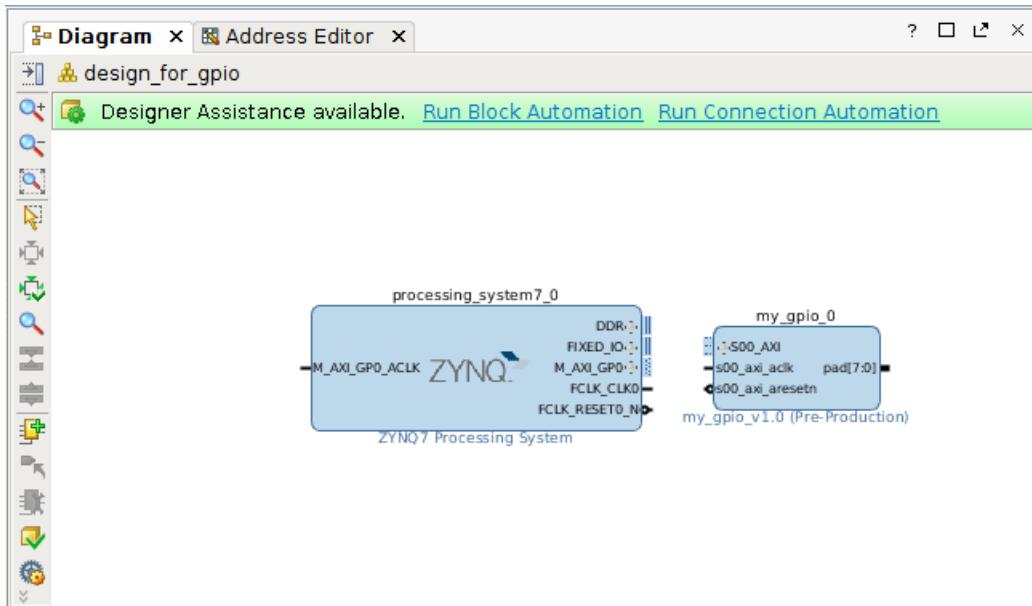


Figura 1.24: Diagram

Nella stessa finestra cliccare su *Run Block Automation* e successivamente su *Run Connection Automation*, che aggiunge al progetto l'*AXI Interconnect* e il *Processor System Reset*, opportunamente collegati dal tool Vivado. A questo punto, cliccando col tasto destro sul pad di uscita del gpio, andare su **Make External**, fig. 1.25.

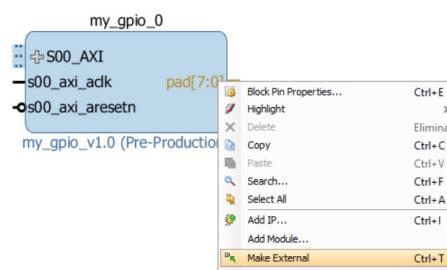


Figura 1.25: Make External

Si ottiene il Diagram di fig. 1.26.

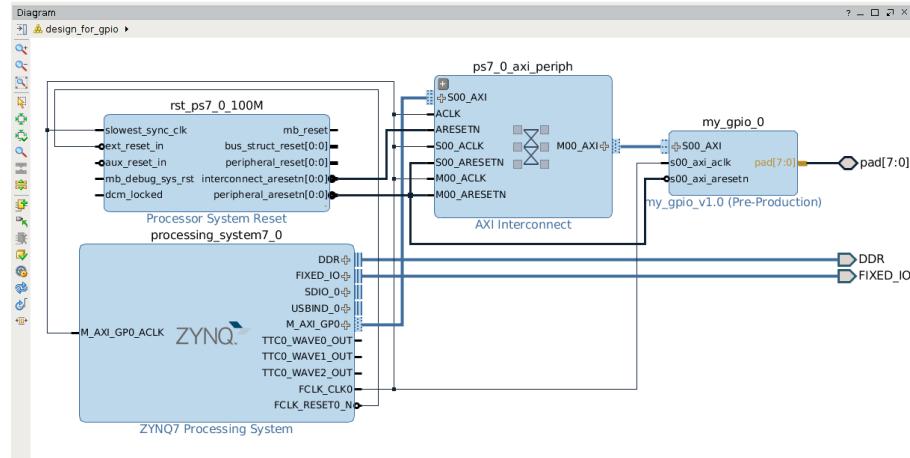


Figura 1.26: Block Diagram

Infine, cliccare su **Validate Design** (in alternativa, basta premere il tasto F6), per validare il design e accertarsi che non ci siano errori oppure critical warning.

### 1.2.1.6 Sintesi e Implementazione

Prima di procedere alla sintesi, cliccando con tasto destro su *design\_for\_gpio*, andare su **Create HDL Wrapper** e selezionare *Let Vivado manage wrapper and auto-update*. Si noti che Vivado crea automaticamente la gerarchia di file VHDL.

In **Flow Navigator → Synthesis** cliccare su **Run Synthesis**. A sintesi ultimata, spuntare *Run Implementation* per avviare l'implementazione. Ad implementazione ultimata, spuntare *Open Implemented Design*.

Aprire **Floorplanning** tra i Layout disponibili, come in fig. 1.27.

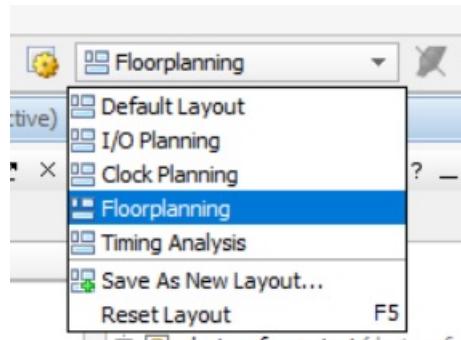
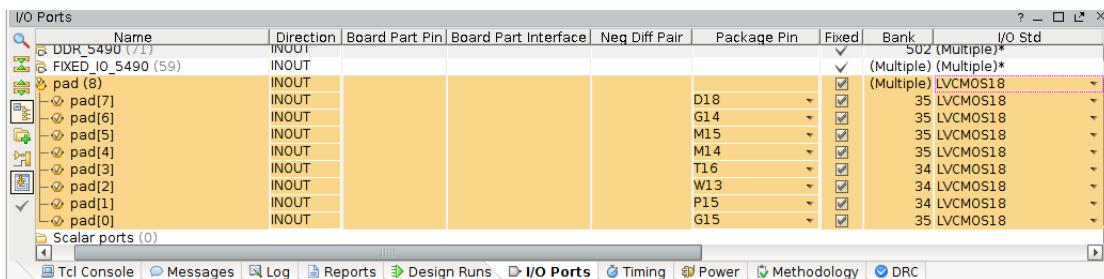


Figura 1.27: Floorplanning Layout

In I/O Ports mappare opportunamente le uscite del componente gpio con i pad della board, fig. 1.28.



The screenshot shows the Vivado I/O Ports configuration window. The table lists various pins and their assignments:

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
DDR_5490 (7)	INOUT				D18	<input checked="" type="checkbox"/>	502 (Multiple)*	(Multiple) LVCMS18
FIXED_IO_5490 (59)	INOUT				G14	<input checked="" type="checkbox"/>	(Multiple)	35 LVCMS18
pad (8)	INOUT				M15	<input checked="" type="checkbox"/>	(Multiple)	35 LVCMS18
pad[7]	INOUT				M14	<input checked="" type="checkbox"/>	(Multiple)	35 LVCMS18
pad[6]	INOUT				T16	<input checked="" type="checkbox"/>	(Multiple)	34 LVCMS18
pad[5]	INOUT				W13	<input checked="" type="checkbox"/>	(Multiple)	34 LVCMS18
pad[4]	INOUT				P15	<input checked="" type="checkbox"/>	(Multiple)	34 LVCMS18
pad[3]	INOUT				G15	<input checked="" type="checkbox"/>	(Multiple)	35 LVCMS18
pad[2]	INOUT							
pad[1]	INOUT							
pad[0]	INOUT							

Figura 1.28: Pad mapping

Si clicca su salva e si da il nome al file XDC creato, *pad\_location*.

In **Flow Navigator** → **Program and Debug** cliccare su **Generate Bitstream**.

A questo punto, andare su **File** → **Export** → **Export Hardware...** e spuntare l'opzione *Include bitstream*.

### 1.2.1.7 SDK Tool

Aprire SDK, direttamente dall'ambiente Vivado, cliccando su **File** → **Launch SDK**.

In SDK bisogna creare un nuovo progetto, andare, quindi, su **File** → **New** → **Application Project** a cui viene dato il nome *gpio*, fig. 1.29.

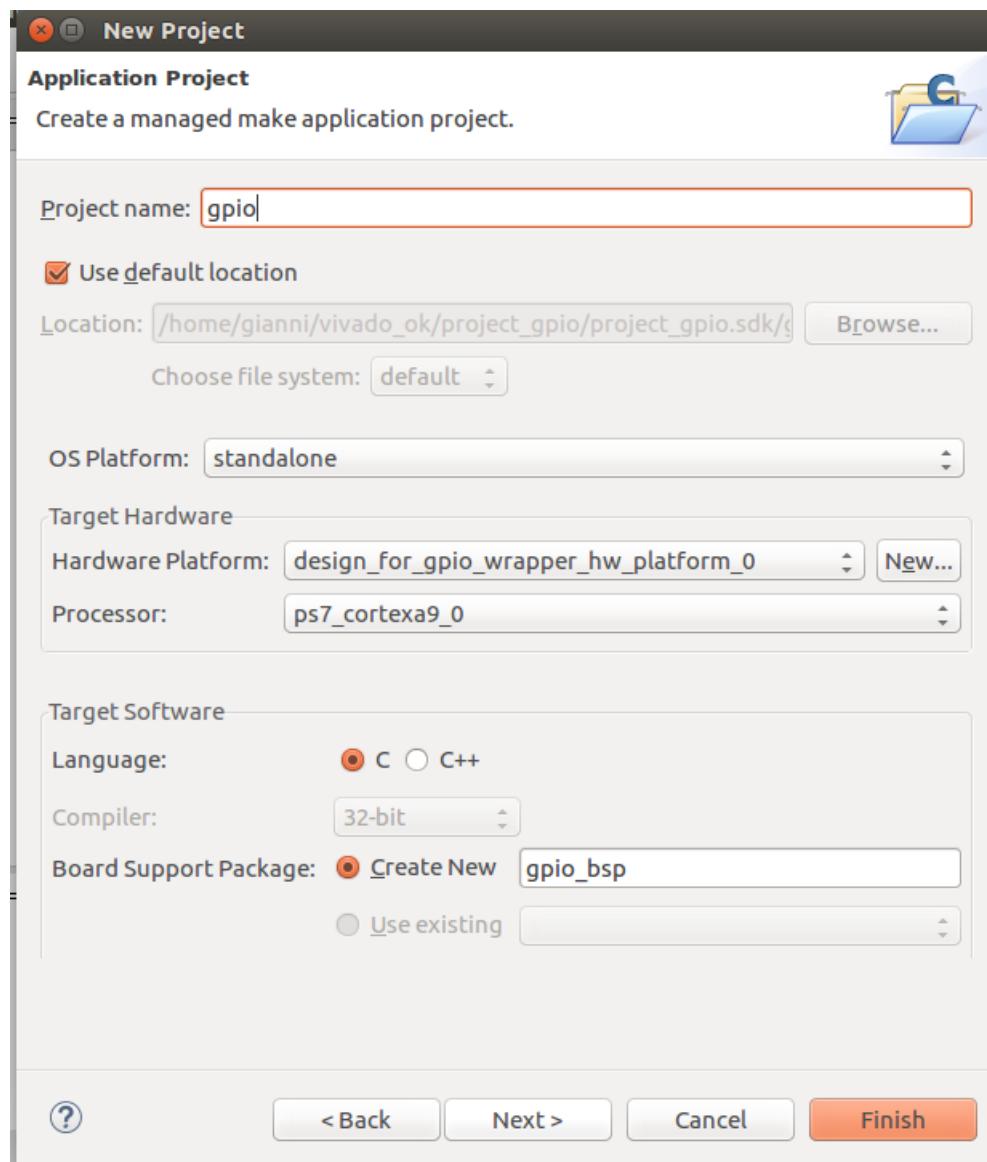


Figura 1.29: Gpio Application Project

Per programmare l'FPGA della board andare su **Xilinx Tools** → **Program FPGA**. Nella finestra che si apre, selezionare opportunamente il bitstream, fig. 1.30.

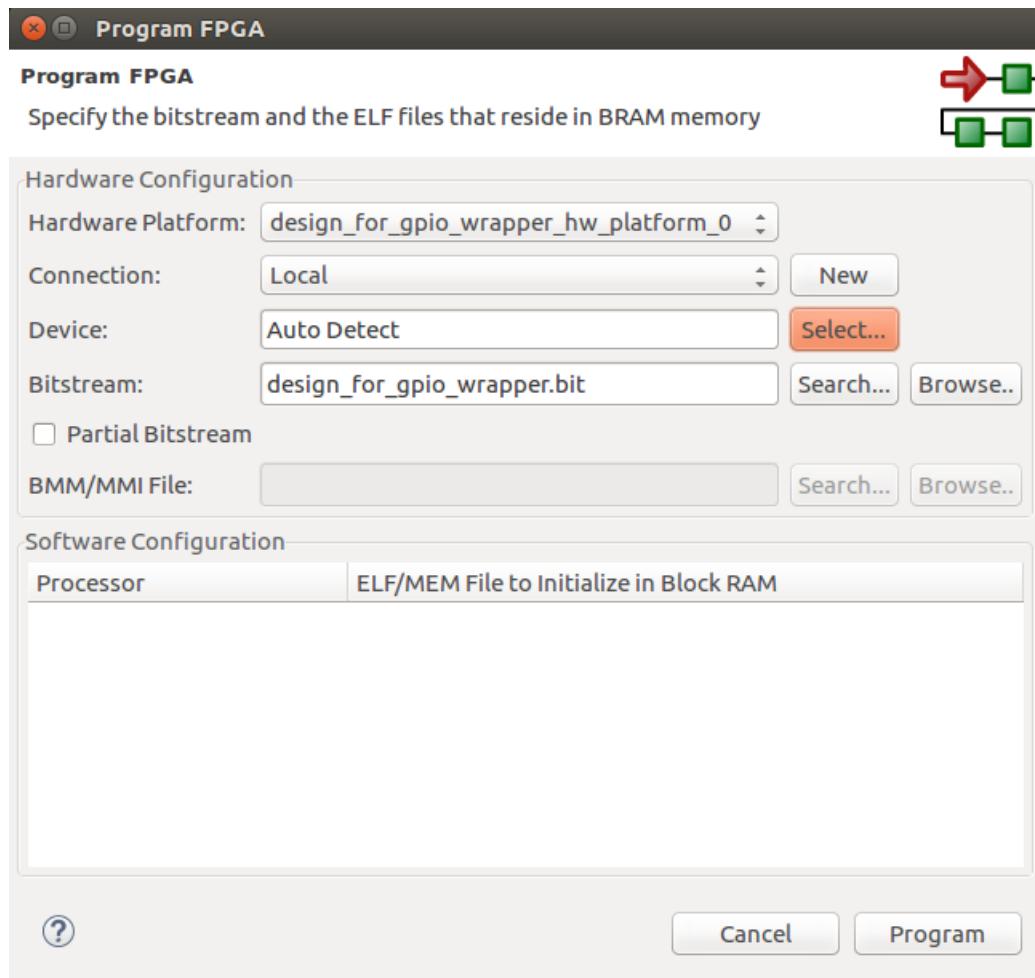


Figura 1.30: Program FPGA

Adesso è il momento di configurare il debug. Andare su **Run → Debug Configuration**. Nel menù a tendina a sinistra, selezionare **System Debugger**. Nella finestra **Target Setup**, in **Execute Script**, selezionare il file *ps7\_init.tcl* nella cartella *design\_for\_gpio\_wrapper\_hw\_platform\_0*, fig. 1.31. A causa di un bug della versione 2016.4, prima di caricare quest'ultimo bisogna modificarlo andando ad eliminare l'opzione *-force* ogni volta che si presenta.

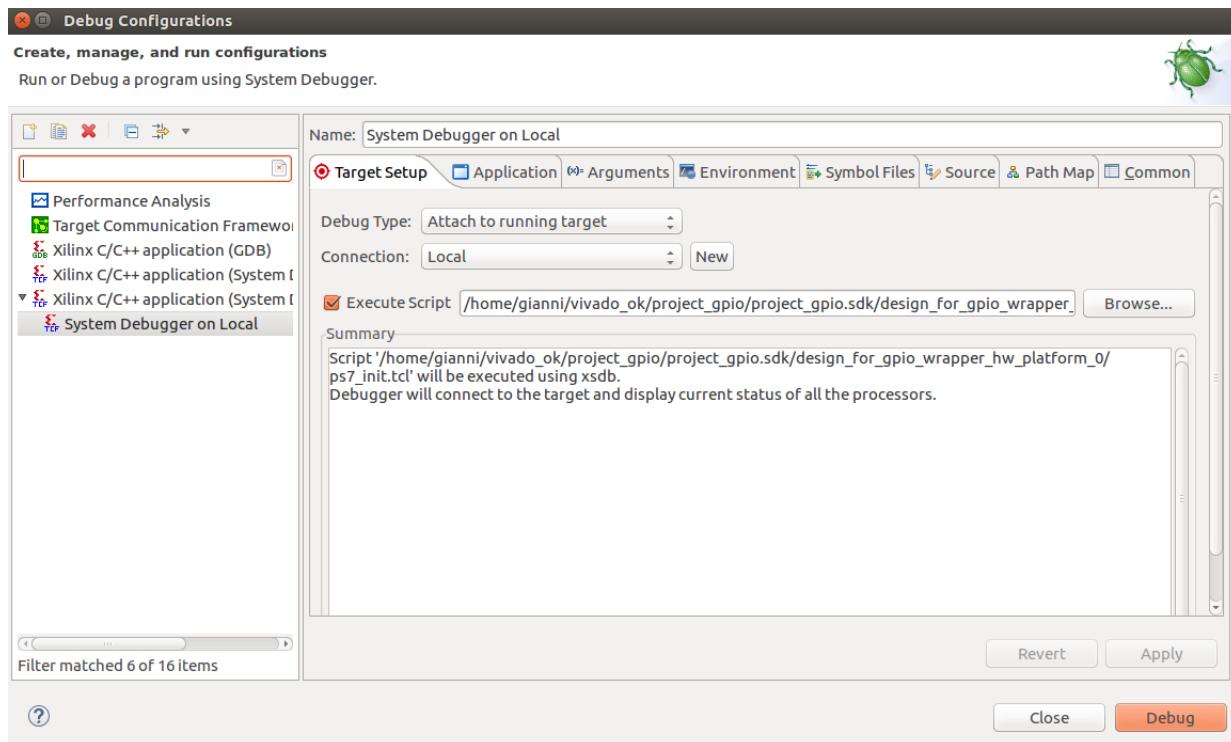


Figura 1.31: Debug Configuration

Premere **Apply** ed infine **Debug**.

In **Project Explorer**, aprire **gpio** → **Binaries**. Cliccando tasto destro su **gpio.elf**, selezionare **Run as** → **Launch on Hardware (System Debugger)**, fig. 1.32.

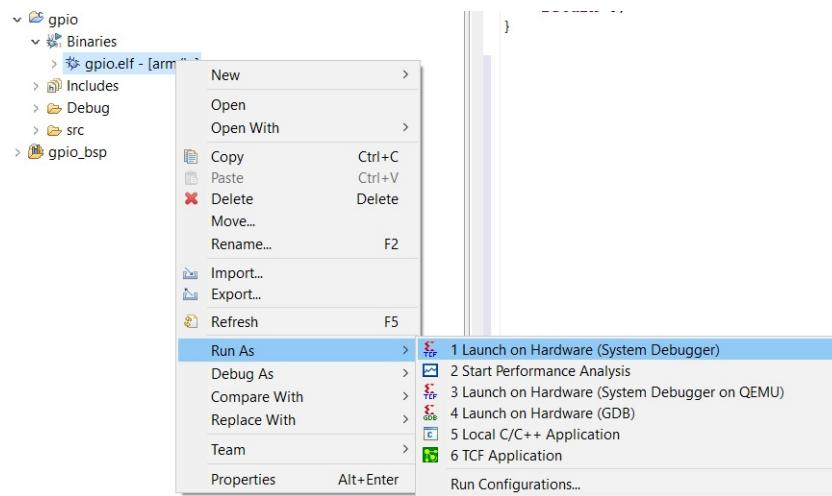


Figura 1.32: Launch on Hardware

Aprire la schermata **Debug**, in alto a destra. In **Windows** → **Show view**, selezionare **Memory**. Cliccare sul tasto + verde (*Add Memory Monitor*) e immettere il base address associato alla periferica gpio, **0x43c00000**. Nell'esempio considerato in fig. 1.33, per una maggiore leggibilità si è scelta come formattazione 1 byte per ogni colonna e 4 byte per ogni riga. Per modificarla basta cliccare tasto destro su **Address** e premere **Format**.

Address	0	1	2	3
43C00000	A0	00	00	00
43C00004	00	00	00	00
43C00008	F0	00	00	00

Figura 1.33: Register value

Come si nota, confrontando gli indirizzi col codice scritto precedentemente, *slv\_reg0* è mappato all’indirizzo *43c00000* e corrisponde al *pad\_out*, *slv\_reg1* è mappato all’indirizzo *43c00004* e corrisponde a *pad\_rw\_n*, infine, *slv\_reg2* è mappato all’indirizzo *43c00008* e corrisponde a *pad\_en*. Da notare che i byte sulle colonne sono in rappresentazione big endian, mentre i bit sono rappresentati da due valori esadecimale in rappresentazione little endian. In fig. 7.4 si osserva l’accensione dei led coorispondente al valore scritto nel registro *slv\_reg0*.

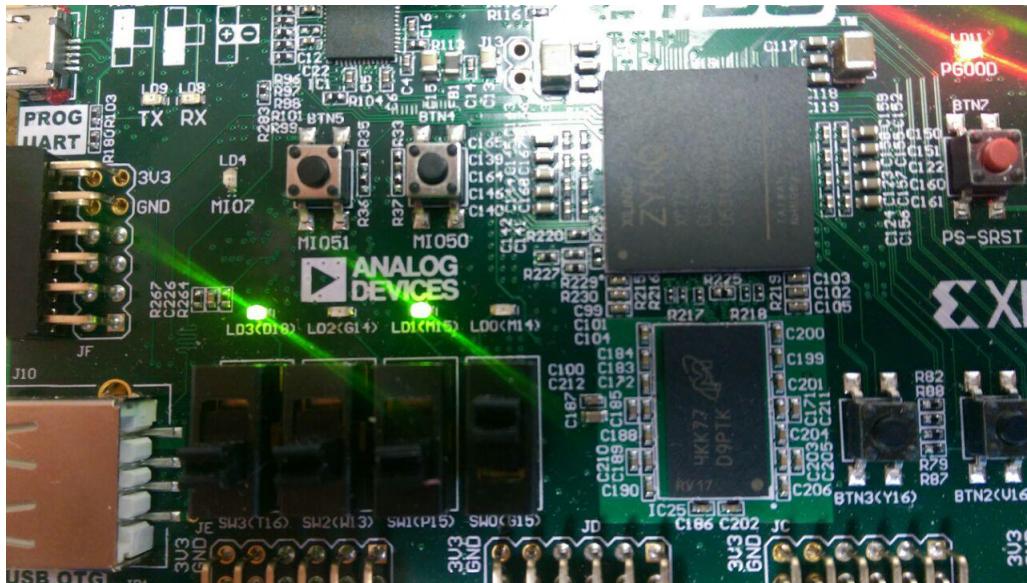


Figura 1.34: Accensione dei led sulla board

## 1.2.2 Driver + Applicazione

L’obiettivo di questa seconda parte dell’elaborato è scrivere un driver in C per pilotare la periferica di GPIO in maniera del tutto automatica.

A tal fine, bisogna innanzitutto creare una libreria in cui dichiarare tutte le funzioni che saranno utilizzate dal driver.

### 1.2.2.1 Libreria HAL driver

Nella view C/C++ di SDK, in **Project Explorer**, cliccare con tasto destro su **gpio** → **src** → **New**, cliccare prima su **Header File** e poi ripetere la stessa cosa con **Source File**, per creare rispettivamente *gpio\_header.h* e *gpio\_functions.c*. Infine, rinominare il file *helloworld.c* in *main.c*, fig. 1.35.

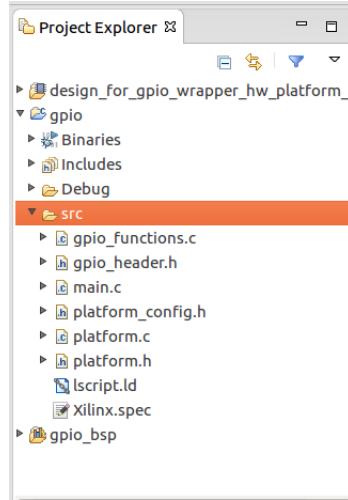


Figura 1.35: New Header File e Source File

### Dichiarazione macro

Per una corretta implementazione del driver è necessario individuare il base address e conoscere lo spiazzamento tra i vari registri della periferica che è stata implementata.

A tale scopo nell'esportazione dell'hardware da Vivado a SDK, viene creato un header file sotto il nome di *xparameters.h* in cui sono contenuti proprio i parametri suddetti sotto forma di macro (tale header può essere individuato in *gpio\_bsp/ps7\_cortexa9\_0/include*). Poiché le macro create automaticamente non sono di facile memorizzazione per il programmatore, in *gpio\_header.h*, dopo aver richiamato la libreria *xparameters.h* vengono create delle ulteriori macro che fungono sostanzialmente da alias. Di seguito, si riporta il codice.

```

19 /**
20  * @brief Macro indirizzo base GPIO
21  */
22 #define GPIO_ADDRESS (uint32_t*) XPAR_MY_GPIO_0_S00_AXI_BASEADDR
23
24 /**
25  * @brief Spiazzamenti registri periferica
26  */
27 #define PAD_OUT MY_GPIO_S00_AXI_SLV_REG0_OFFSET /*!< 0x0 */
28 #define PAD_RW_N MY_GPIO_S00_AXI_SLV_REG1_OFFSET /*!< 0x4 */
29 #define PAD_EN MY_GPIO_S00_AXI_SLV_REG2_OFFSET /*!< 0x8 */
30 #define PAD_IN MY_GPIO_S00_AXI_SLV_REG3_OFFSET /*!< 0xC */

```

Codice 1.3: "gpio header.h"

### Dichiarazione funzioni

Una corretta gestione della periferica di GPIO richiede l'utilizzo delle seguenti funzioni per la gestione dei registri:

- **gpio\_init** inizializza un puntatore all'indirizzo corretto della periferica, nonché una corretta configurazione dei relativi pin della periferica in lettura o scrittura;
- **set\_value\_reg** scrive all'interno di uno specifico registro certi valori, data una certa maschera fornita in input;

- **gpio\_write\_mask**, richiamando la *set\_value\_reg*, mediante una determinata maschera di bit, si occupa di scrivere un valore alto o basso in una determinata porzione del registro selezionata in input;
- **gpio\_write\_one**, data una certa posizione specificata in ingresso, scrive un valore alto o basso in una determinata porzione di un registro;
- **gpio\_read\_mask**, specificata una particolare maschera di bit, restituisce i valori presenti nel registro selezionato in ingresso;
- **gpio\_read\_one** restituisce il valore di un bit scritto in una specifica posizione di un registro;
- **gpio\_toggle\_mask**, data una certa maschera in ingresso, si occupa di invertire i valori binari all'interno di un registro il cui spiazzamento è fornito come input;
- **gpio\_toggle\_one**, allo stesso modo della precedente funzione, data una specifica posizione, inverte un valore da basso ad alto o viceversa, all'interno di un registro.

La seguente porzione di codice evidenzia i prototipi delle funzioni appena descritte.

```

55 void gpio_init (uint32_t* base_addr);
56 void set_value_reg(uint32_t reg, uint32_t mask,uint32_t mask_value);
57 void gpio_write_mask(uint32_t reg, uint8_t set, uint32_t mask, uint8_t value
58   );
59 void gpio_write_one(uint32_t reg, uint8_t set, uint32_t position, uint8_t
60   value);
61 uint32_t gpio_read_mask(uint32_t reg, uint8_t set, uint32_t mask);
62 uint32_t gpio_read_one(uint32_t reg, uint8_t set, uint32_t position);
63 void gpio_toggle_one(uint32_t reg,uint8_t set,uint8_t position);
64 void gpio_toggle_mask(uint32_t reg, uint8_t set,uint32_t mask);
```

Codice 1.4: "gpio header.h"

### Implementazione funzioni

Di seguito si riporta l'implementazione di tutte le funzioni.

```

1 /**
2 ****
3 * @file      gpio_functions.h
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *           Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     10-May-2017
8 * @brief    Questo file mette a disposizione un firmware per gestire il
9 *           comportamento della periferica
10 ****
11 #include "gpio_header.h"
12
13 /*Variabile globale per memorizzare l'indirizzo del base address*/
```

```

14 uint32_t* my_gpio_pointer;
15
16 /**
17 * @brief Questa funzione permette di inizializzare correttamente un GPIO
18 *
19 * @param base_addr: puntatore ad un intero di 32 bit per il corretto
20 *                   indirizzamento della periferica
21 *
22 * @retval none
23 */
24 void gpio_init (uint32_t* base_addr){
25     /*Verifica se l'indirizzo passato è allineato a 4 byte*/
26     assert((uint32_t)base_addr%4==0 && "Base address non allineato a 4 byte");
27
28     my_gpio_pointer=base_addr;
29
30     /*Configurazione pin di GPIO*/
31     *(my_gpio_pointer+PAD_EN/4)=INIT_CONFIG_EN;
32     *(my_gpio_pointer+PAD_RW_N/4)=INIT_CONFIG_RWN;
33 }
34
35 /**
36 * @brief Questa funzione permette di settare all'interno di uno specifico
37 *        registro, dei valori in base ad una determinata maschera di bit
38 *
39 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
40 *             indirizzo
41 *             base del registro in cui scrivere
42 * @param mask: intero a 32 bit che specifica una maschera di bit, in
43 *              particolare
44 *              se all'i-esima posizione vi è un valore pari a 1 allora si è
45 *              abilitati a
46 *              scrivere in quella posizione, se 0 allora il valore è mascherato
47 * @param mask_value: intero a 32 bit da scrivere all'interno del registro
48 * @retval none
49 */
50 void set_value_reg(uint32_t reg, uint32_t mask,uint32_t mask_value){
51     *(my_gpio_pointer+reg/4) = (*(my_gpio_pointer+reg/4) & ~mask) | (mask &
52         mask_value);
53 }
54
55 /**
56 * @brief Questa funzione permette di scrivere all'interno di uno
57 *        specifico
58 *        registro in base ad una determinata maschera di bit
59 *
60 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
61 *             indirizzo
62 *             base del registro in cui scrivere

```

```

57  * @param set: intero a 8 bit che indica lo spiazzamento interno al
58  *   registro
59  * @param mask: intero a 32 bit che specifica una maschera di bit, in
60  *   particolare
61  *       se all'i-esima posizione vi è un valore pari a 1 allora si è
62  *       abilitati a
63  *       scrivere in quella posizione, se 0 allora il valore è mascherato
64  * @param value: intero a 8 bit che indica il valore da scrivere
65  *
66 void gpio_write_mask(uint32_t reg, uint8_t set, uint32_t mask, uint8_t value
67  ) {
68     assert((reg==0 || reg%12!=0) && "Scrittura in una locazione di memoria non
69         consentita");
70     mask = mask<<set;
71     if(((0x0000000F<<set) & mask)==mask)
72         /*Se value è pari a 1 allora si andrà a scrivere tanti 1 in base alla
73             maschera */
74     value == HIGH ? set_value_reg(reg,mask,0xF<<set) : set_value_reg(reg,
75         mask,0x0);
76 }
77 /**
78  * @brief Questa funzione permette di scrivere un unico all'interno di uno
79  *       specifico registro in base ad una determinata posizione
80  *
81  * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
82  *       indirizzo
83  *       base del registro in cui scrivere
84  * @param set: intero a 8 bit che indica lo spiazzamento interno al
85  *       registro
86  * @param position: intero a 32 bit che specifica la posizione in cui
87  *       scrivere
88  * @param mask_value: intero a 8 bit che indica il valore da scrivere
89  *
90  * @retval none
91  */
92 void gpio_write_one(uint32_t reg, uint8_t set, uint32_t position, uint8_t
93     value){
94     gpio_write_mask(reg, set, 0x1<<position, value);
95 }

96 /**
97  * @brief Questa funzione permette di leggere all'interno di uno specifico
98  *       registro in base ad una determinata maschera di bit
99  *

```

```

95 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
96   indirizzo
97   base del registro in cui scrivere
98 * @param set: intero a 8 bit che indica lo spiazzamento interno al
99   registro
100 * @param mask: intero a 32 bit che specifica una maschera di bit, in
101   particolare
102   se all'i-esima posizione vi è un valore pari a 1 allora si è
103   abilitati a
104   leggere in quella posizione, viceversa se 0
105 *
106 * @retval intero a 32 bit
107 */
108
109 uint32_t gpio_read_mask(uint32_t reg, uint8_t set, uint32_t mask) {
110     mask = mask << set;
111     if(((0x000000F << set) & mask) == mask) {
112         return *(my_gpio_pointer + reg/4) & mask;
113     }
114     return -1;
115 }
116
117 /**
118 * @brief Questa funzione permette di leggere un unico valore all'interno
119   di uno
120   specifico registro in base ad una determinata posizione
121 *
122 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
123   indirizzo
124   base del registro in cui scrivere
125 * @param set: intero a 8 bit che indica lo spiazzamento interno al
126   registro
127 * @param position: intero a 32 bit che specifica la posizione in cui
128   leggere
129 *
130 * @retval intero a 32 bit
131 */
132
133 uint32_t gpio_read_one(uint32_t reg, uint8_t set, uint32_t position){
134     return gpio_read_mask(reg, set, 0x1 << position) == (0x1 << (position+set));
135 }
136
137 /**
138 * @brief Questa funzione permette il toggle di un singolo valore all'
139   interno di
140   uno specifico registro in base ad una determinata posizione
141 *
142 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
143   indirizzo

```

```

134     *      base del registro in cui effettuare il toggle
135     * @param set: intero a 8 bit che indica lo spiazzamento interno al
136     *      registro
137     * @param position: intero a 8 bit che specifica la posizione in cui
138     *      effettuare
139     *      il toggle
140     *
141     * @retval none
142     */
143
144 void gpio_toggle_one(uint32_t reg, uint8_t set,uint8_t position){
145     gpio_read_one(reg,set,position) == HIGH ? gpio_write_one(reg ,set,position
146     ,LOW) : gpio_write_one(reg,set,position,HIGH);
147 }
148
149 /**
150 * @brief Questa funzione permette il toggle di un gruppo di bit all'
151 *      interno di
152 *      uno specifico registro in base ad una determinata maschera
153 *      *
154 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
155 *      indirizzo
156 *      base del registro in cui effettuare il toggle
157 * @param set: intero a 8 bit che indica lo spiazzamento interno al
158 *      registro
159 * @param position: intero a 32 bit che specifica la maschera di bit, in
160 *      particolare se il valore i-esimo è pari a 1 allora in quella
161 *      posizione
162 *      si è abilitati ad effettuare il toggle, viceversa non si effettua
163 *      tale
164 *      operazione
165 *      *
166 * @retval none
167 */
168
169 void gpio_toggle_mask(uint32_t reg, uint8_t set,uint32_t mask){
170     int i;
171     for(i=0;i<4;i++) {
172         if((0x1<<(set*4+i)) == ((0x1<<(set*4+i)) & mask)){
173             gpio_toggle_one(reg, set,i+set*4);
174         }
175     }
176 }

```

Codice 1.5: "gpio functions.c"

### 1.2.2.2 Driver

In questa sezione si mostra un driver che utilizza alcune delle funzioni appena viste, in particolare la simulazione del lampeggiamento dei led posti sul muso anteriore di K.I.T.T., famosa auto del telefilm Anni '80 *Supercar*.

#### Codice

Per implementare il driver è necessaria una fase di inizializzazione, mediante la funzione `init`, in cui viene settato un puntatore che tiene traccia del base address della periferica e settati i relativi bit dei registri della GPIO. In particolare, si indica al registro delle abilitazioni di rendere disponibili ad essere modificati i bit relativi all'illuminazione dei led e successivamente si provvedere a resettare il registro dati in modo tale da cancellare eventuali valori presenti in precedenza. Successivamente viene richiamata la funzione `supercar` che implementa tutta la logica dello shifting di un singolo bit da destra verso sinistra e viceversa.

```

1 /**
2  ****
3 * @file      mainc
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *           Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     10-May-2017
8 * @brief    Programma principale che contiene al suo interno l'
9 *           implemetazione di
10 *           un particolare driver che pilota la periferica
11 ****
12 */
13
14 #include <stdio.h>
15 #include "platform.h"
16 #include "xil_printf.h"
17 #include "gpio_header.h"
18 #include <unistd.h>
19
20 void init();
21 void supercar();
22
23 int main()
24 {
25     init_platform();
26     /*!<Primo passo*/
27     /*Si inizializza correttamente la periferica*/
28     init();
29     /*!<Secondo passo*/
30     /*Si richiama il relativo driver*/
31     supercar();
32     cleanup_platform();

```

```

32     return 0;
33 }
34
35 /**
36 * @brief Questa funzione permette di inizializzare in maniera opportuna
37 * la
38 * periferica
39 * @note Si può notare come la periferica di GPIO è inizializzata in modo
40 * tale
41 * da utilizzare 8 pin: in accordo all'header file gpio_header.h sui
42 * primi 4 pin, a partire dai meno significativi vengono mappati degli
43 * switch,
44 * sui secondi 4 dei led
45 * @retval none
46 */
47
48 void init() {
49     gpio_init(GPIO_ADDRESS);
50     gpio_write_mask(PAD_EN, SET_LED, _F_, HIGH);
51     gpio_write_mask(PAD_OUT, SET_LED, _F_, LOW);
52 }
53
54 /**
55 * @brief Implementazione del driver per il pilotaggio della periferica
56 * @note Il driver permette la simulazione del lampeggiamento dei led
57 * posti sul muso anteriore di K.I.T.T., famosa auto del telefilm
58 * Anni '80 Supercar.
59 * @retval none
60 */
61
62 void supercar() {
63     int position=ZERO;
64     short int dx=0;
65     useconds_t usec = 500000;
66     while(1) {
67         usleep(usec);
68         /*Il ramo then permette di scorrere l'illuminazione dei led da destra
69          verso sinistra*/
70         if(!dx) {
71             gpio_toggle_one(PAD_OUT, SET_LED, position);
72             usleep(usec);
73             gpio_toggle_one(PAD_OUT, SET_LED, position);
74             /*Poiché il driver è specifico per la board Zynq-7000
75              * tale controllo prevede che all'illuminazione del quarto led da
76              * destra
77              * viene posta la variabile dx a 1 in modo tale che alla successiva
78              * iterazione lo scorrimento avvenga da sinistra a destra*/
79             if(position==THREE)
80                 dx=1;
81         }
82     }
83 }
```

```

76     else
77         position++;
78     }
79     /*Il ramo else permette di scorrere l'illuminazione dei led da sinistra
80      verso destra*/
81     else{
82         position--;
83         gpio_toggle_one(PAD_OUT, SET_LED, position);
84         usleep(usec);
85         gpio_toggle_one(PAD_OUT, SET_LED, position);
86         /*Se lo scorrimento ha raggiunto il led più a destra, allora viene
87          posta
88          * la variabile dx a 0 in modo tale che alla successiva iterazione,
89          * avvenga da destra verso sinistra*/
90         if(position==ZERO){
91             dx=0;
92             position++;
93         }
94     }
95 }
96 }
```

Codice 1.6: "main.c"

### 1.2.2.3 Versione object oriented

Di seguito viene mostrato lo stesso codice visto in precedenza rivisto mediante il paradigma della programmazione object oriented. La differenza con la precedente versione è sostanzialmente l'incapsulamento all'interno di una struttura dati delle informazioni riguardo un'istanza di GPIO, ossia del base address e dei relativi spiazzamenti tra i vari registri della periferica. Inoltre, è necessario che ad ogni funzione sia passato un riferimento a tale struttura dati affinché si agisca su una determinata istanza.

#### Header file

```

1 /**
2 ****
3 * @file      gpio_header.h
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     12-May-2017
8 * @brief    Header file della libreria HAL driver per gestione GPIO custom
9 ****
10 */
```

```

11
12 #ifndef SRC_GPIO_CUSTOM_H_
13 #define SRC_GPIO_CUSTOM_H_
14
15 #include "xparameters.h"
16 #include "my_gpio.h"
17 #include <assert.h>
18
19 /**
20  * @brief Definizione struttura dati per la gestione del GPIO
21  */
22 typedef struct{
23     uint32_t *base_address; /*!< GPIO Base address */
24     uint8_t pad_out_offset; /*!< GPIO offset registro di pad_out */
25     uint8_t pad_rw_n_offset; /*!< GPIO offset registro modalità read/write */
26     uint8_t pad_en_offset; /*!< GPIO offset registro abilitazione pin */
27     uint8_t pad_in_offset; /*!< GPIO offset registro pad_in*/
28 }gpio_custom_TypeDef;
29
30 /**
31  * @brief Configurazione dell'indirizzo base
32  */
33 #define GPIO_BASE XPAR_MY_GPIO_0_S00_AXI_BASEADDR
34
35 /**
36  * @brief Spiazzamenti registri periferica
37  */
38 #define gpio_custom_PAD_OUT MY_GPIO_S00_AXI_SLV_REG0_OFFSET /*!< 0x0 */
39 #define gpio_custom_PAD_RW_N MY_GPIO_S00_AXI_SLV_REG1_OFFSET /*!< 0x4 */
40 #define gpio_custom_PAD_EN MY_GPIO_S00_AXI_SLV_REG2_OFFSET /*!< 0x8 */
41 #define gpio_custom_PAD_IN MY_GPIO_S00_AXI_SLV_REG3_OFFSET /*!< 0xC */
42
43 /**
44  * @brief Definizione dei valori alto e basso
45  */
46 typedef enum{
47     LOW,
48     HIGH
49 }gpio_custom_ValueType;
50
51 /**
52  * @brief Posizionamento su GPIO di LED e Switch
53  */
54 typedef enum{
55     SET_SWITCH = 0x0,
56     SET_LED = 0x4
57 }gpio_custom_SetType;
58
59

```

```

60 /**
61 * @brief Impostazione della modalità su ogni pin di GPIO
62 */
63 typedef enum{
64     INIT_CONFIG_EN = 0x0,
65     INIT_CONFIG_RWN = 0xF
66 }gpio_custom_InitType;
67
68 /*****Funzioni utente per gestione registri*****/
69 void gpio_custom_Init(gpio_custom_TypeDef *gpio, uint32_t base_address);
70 void gpio_custom_Set_value_reg(gpio_custom_TypeDef *gpio, uint32_t reg,
71     uint32_t mask,uint32_t mask_value);
72 void gpio_custom_Write_mask(gpio_custom_TypeDef *gpio,uint32_t reg, uint8_t
73     set, uint32_t mask, uint8_t value);
74 void gpio_custom_Write_one(gpio_custom_TypeDef *gpio,uint32_t reg, uint8_t
75     set, uint32_t position, uint8_t value);
76 uint32_t gpio_custom_Read_mask(gpio_custom_TypeDef *gpio,uint32_t reg,
77     uint8_t set, uint32_t mask);
78 uint32_t gpio_custom_Read_one(gpio_custom_TypeDef *gpio, uint32_t reg,
79     uint8_t set, uint32_t position);
80 void gpio_custom_Toggle_one(gpio_custom_TypeDef *gpio, uint32_t reg,uint8_t
81     set,uint8_t position);
82 void gpio_custom_Toggle_mask(gpio_custom_TypeDef *gpio, uint32_t reg,
83     uint8_t set,uint32_t mask);
84
85 #endif /* SRC_GPIO_CUSTOM_H_ */

```

Codice 1.7: "gpio custom.h"

### Implementazione funzioni

```

1 /**
2 ****
3 * @file      gpio_custom.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     12-May-2017
8 * @brief    Questo file mette a disposizione un firmware per gestire il
9 *           comportamento della periferica
10 ****
11
12 */
13 /**
14 * @brief  Questa funzione permette di inizializzare correttamente un GPIO
15 *

```

```

16  * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
17  *      le informazioni
18  * @param base_addr: puntatore ad un intero di 32 bit per il corretto
19  *      indirizzamento della periferica
20  *
21  * @retval none
22  */
23
24 void gpio_custom_Init(gpio_custom_TypeDef *gpio, uint32_t base_address) {
25
26     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
27     assert(base_address%4==0 && "Base address non allineato a 4 byte");
28     gpio->base_address=(uint32_t*)base_address;
29     gpio->pad_en_offset=gpio_custom_PAD_EN;
30     gpio->pad_in_offset=gpio_custom_PAD_IN;
31     gpio->pad_out_offset=gpio_custom_PAD_OUT;
32     gpio->pad_rw_n_offset=gpio_custom_PAD_RW_N;
33     gpio->base_address[	gpio->pad_en_offset/4]=INIT_CONFIG_EN;
34     gpio->base_address[	gpio->pad_rw_n_offset/4]=INIT_CONFIG_RWN;
35 }
36 /**
37  * @brief Questa funzione permette di settare all'interno di uno specifico
38  *      registro, dei valori in base ad una determinata maschera di bit
39  *
40  * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
41  *      le informazioni
42  * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
43  *      indirizzo
44  *          base del registro in cui scrivere
45  * @param mask: intero a 32 bit che specifica una maschera di bit, in
46  *      particolare
47  *          se all'i-esima posizione vi è un valore pari a 1 allora si è
48  *          abilitati a
49  *              scrivere in quella posizione, se 0 allora il valore è mascherato
50  * @param mask_value: intero a 32 bit da scrivere all'interno del registro
51  *
52  * @retval none
53  */
54
55 void gpio_custom_Set_value_reg(gpio_custom_TypeDef *gpio, uint32_t reg,
56     uint32_t mask,uint32_t mask_value) {
57
58     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
59     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
60     ;
61     gpio->base_address[reg/4]=(gpio->base_address[reg/4] & ~mask) | (mask &

```

```

        mask_value);
}

59 /**
60 * @brief Questa funzione permette di scrivere all'interno di uno
61 * specifico
62 *      registro in base ad una determinata maschera di bit
63 *
64 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
65 *      le informazioni
66 *      di configurazione della specifica GPIO
67 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
68 *      indirizzo
69 *          base del registro in cui scrivere
70 * @param set: intero a 8 bit che indica lo spiazzamento interno al
71 *      registro
72 * @param mask: intero a 32 bit che specifica una maschera di bit, in
73 *      particolare
74 *          se all'i-esima posizione vi è un valore pari a 1 allora si è
75 *          abilitati a
76 *              scrivere in quella posizione, se 0 allora il valore è mascherato
77 * @param value: intero a 8 bit che indica il valore da scrivere
78 *
79 * @retval none
80 */
81
82 void gpio_custom_Write_mask(gpio_custom_TypeDef *gpio, uint32_t reg, uint8_t
83     set, uint32_t mask, uint8_t value) {
84
85     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
86     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
87     ;
88     assert((reg==0 || reg%12!=0) && "Scrittura in una locazione di memoria non
89         consentita");
90     mask = mask<<set;
91     if(((0x0000000F<<set) & mask)==mask)
92         value == HIGH ? gpio_custom_Set_value_reg(gpio, reg, mask, 0xF<<set) :
93             gpio_custom_Set_value_reg(gpio, reg, mask, 0x0);
94 }
95
96 /**
97 * @brief Questa funzione permette di scrivere un unico all'interno di uno
98 * specifico registro in base ad una determinata posizione
99 *
100 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
101 *      le informazioni
102 *      di configurazione della specifica GPIO
103 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
104 *      indirizzo

```

```

93     *      base del registro in cui scrivere
94     * @param set: intero a 8 bit che indica lo spiazzamento interno al
95     *      registro
96     * @param position: intero a 32 bit che specifica la posizione in cui
97     *      scrivere
98     * @param mask_value: intero a 8 bit che indica il valore da scrivere
99     *
100    */
101
101 void gpio_custom_Write_one(gpio_custom_TypeDef *gpio, uint32_t reg, uint8_t
102     set, uint32_t position, uint8_t value){
103     gpio_custom_Write_mask(gpio, reg, set, 0x1<<position, value);
104 }
105 /**
106 * @brief Questa funzione permette di leggere all'interno di uno specifico
107 *        registro in base ad una determinata maschera di bit
108 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
109 *        le informazioni
110 *        di configurazione della specifica GPIO
111 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
112 *        indirizzo
113 *        base del registro in cui scrivere
114 * @param set: intero a 8 bit che indica lo spiazzamento interno al
115 *        registro
116 * @param mask: intero a 32 bit che specifica una maschera di bit, in
117 *        particolare
118 *        se all'i-esima posizione vi è un valore pari a 1 allora si è
119 *        abilitati a
120 *        leggere in quella posizione, viceversa se 0
121 *
122 * @retval intero a 32 bit
123 */
124
124 uint32_t gpio_custom_Read_mask(gpio_custom_TypeDef *gpio, uint32_t reg,
125     uint8_t set, uint32_t mask){
126
127     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
128     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
129     ;
130     mask = mask<<set;
131     if(((0x0000000F<<set) & mask) == mask){
132         return (gpio->base_address[reg/4] & mask);
133     }
134     return -1;
135 }
136 /**

```

```

132 * @brief Questa funzione permette di leggere un unico valore all'interno
133     di uno
134 *      specifico registro in base ad una determinata posizione
135 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
136     le informazioni
137 *      di configurazione della specifica GPIO
138 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
139     indirizzo
140 *      base del registro in cui scrivere
141 * @param set: intero a 8 bit che indica lo spiazzamento interno al
142     registro
143 * @param position: intero a 32 bit che specifica la posizione in cui
144     leggere
145 *
146 * @retval intero a 32 bit
147 */
148
149 uint32_t gpio_custom_Read_one(gpio_custom_TypeDef *gpio, uint32_t reg,
150     uint8_t set, uint32_t position){
151     return gpio_custom_Read_mask(gpio,reg,set,0x1<<position) == (0x1<<(position+set));
152 }
153
154 /**
155 * @brief Questa funzione permette il toggle di un singolo valore all'
156     interno di
157 *      uno specifico registro in base ad una determinata posizione
158 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
159     le informazioni
160 *      di configurazione della specifica GPIO
161 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
162     indirizzo
163 *      base del registro in cui effettuare il toggle
164 * @param set: intero a 8 bit che indica lo spiazzamento interno al
165     registro
166 * @param position: intero a 8 bit che specifica la posizione in cui
167     effettuare
168 *      il toggle
169 *
170 * @retval none
171 */
172
173 void gpio_custom_Toggle_one(gpio_custom_TypeDef *gpio, uint32_t reg,uint8_t
174     set,uint8_t position){
175     gpio_custom_Read_one(gpio,reg,set,position) == HIGH ?
176         gpio_custom_Write_one(gpio,reg ,set,position,LOW) :
177         gpio_custom_Write_one(gpio,reg,set,position,HIGH);
178 }
179

```

```

166 /**
167 * @brief Questa funzione permette il toggle di un gruppo di bit all'
168 * interno di
169 * uno specifico registro in base ad una determinata maschera
170 *
171 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
172 * le informazioni
173 * di configurazione della specifica GPIO
174 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
175 * indirizzo
176 * base del registro in cui effettuare il toggle
177 * @param set: intero a 8 bit che indica lo spiazzamento interno al
178 * registro
179 * @param position: intero a 32 bit che specifica la maschera di bit, in
180 * particolare se il valore i-esimo è pari a 1 allora in quella
181 * posizione
182 * si è abilitati ad effettuare il toggle, viceversa non si effettua
183 * tale
184 * operazione
185 *
186 * @retval none
187 */
188
189 void gpio_custom_Toggle_mask(gpio_custom_TypeDef *gpio, uint32_t reg,
190     uint8_t set,uint32_t mask){
191     int i;
192     for(i=0;i<4;i++) {
193         if((0x1<<(set*4+i)) == ((0x1<<(set*4+i)) & mask)){
194             gpio_custom_Toggle_one(gpio,reg,set,i+set*4);
195         }
196     }
197 }

```

Codice 1.8: "gpio custom.h"

## 1.3 Supporto per le interruzioni

### 1.3.1 Modifica IP core

Per aggiungere il supporto alle interruzioni, come da titolo, è necessario apportare delle modifiche all'IP core creato nei paragrafi precedenti. Si aggiungono, dunque, dei registri nel modulo di interfacciamento col bus AXI in modo tale da estendere le funzionalità della periferica:

- **interrupt mask register** per mascherare le singole interruzioni;
- **interrupt status register** che tiene traccia delle interruzioni pendenti e non servite;
- **interrupt acknowledge** per la segnalazione da parte del processore e verso la periferica, di averla servita;

- **global interrupt enable** per l'abilitazione totale al supporto delle interruzioni.

Inoltre si aggiunge una seconda interfaccia di uscita che viene collegata direttamente alla PS per segnalare gli eventi: il segnale **irq**.

La parte più interessante è quella relativa alla logica implementativa relativa allo status register, nella fattispecie la scrittura di un bit pari a 1 ossia di un'interruzione pendente, avviene se e soltanto se la periferica è di input e la relativa maschera delle interruzioni permette di rilevare le interrupt sui singoli pin. Inoltre la segnalazione di aver servito un'interrupt avviene scrivendo un bit pari a 1 nel registro degli ack. Di conseguenza se almeno un bit del suddetto registro è pari a 1 vuol dire che il processore ha servito l'interrupt e di conseguenza è possibile resettare il registro di stato, nella posizione corrispondente all'interruzione che è stata servita. In altre parole, se sul registro **reg\_ack** è alto il 4° bit, allora nel registro **status\_register** sarà pulito soltanto il 4° bit. Tutti gli altri bit saranno lasciati inalterati, per segnalare eventualmente che ci sono ancora delle interruzioni pendenti.

```

457      -- Add user logic here
458
459      --istanza gpio
460      GPIO_ARRAY_INST: gpio_array port map(
461          pad_out => reg_pad_out(gpio_size-1 downto 0),
462          pad_rw_n => reg_pad_rw_n(gpio_size-1 downto 0),
463          pad_en => reg_pad_en(gpio_size-1 downto 0),
464          pad_in => periph_pad_in,
465
466          --Implementazione della logica di interrupt
467          --il segnale di interrupt deve essere ottenuto solo mediante una and
468          --bit a bit tra il segnale in ingresso (pad) e il registro delle
469          --interruption mascherate (reg_im),
470          --inoltre poiché un interrupt è desiderabile solo se esso avviene
471          --mediante una periferica di input è necessario eseguire un'altra
472          --and bit a bit col registro di
473          --read_write_n (reg_pad_rw_n)
474          pad_intr_temp <= (pad and reg_im(gpio_size-1 downto 0)) and
475              reg_pad_rw_n(gpio_size-1 downto 0);
476
477          --il segnale di output (irq) è alto se esiste almeno una interrupt
478          --pendente e la global interrupt è attiva (reg_gie)
479          irq <= or_reduce(status_register(gpio_size-1 downto 0)) and reg_gie(0)
480              ;
481
482          --logica di scrittura all'interno dello statut register:
483          --se vi è almeno un bit alto di ack (reg_ack) allora si pulisce tutto
484          --il registro di stato, se invece non è arrivato alcun ack ma la
485          --global interrupt è attiva,
486          --allora voglio salvare nello status ulteriori bit di pending
487          --altrimenti voglio tener memoria del valore precedente.
488      process( S_AXI_ACLK ) is
489          begin
490              if (rising_edge (S_AXI_ACLK)) then

```

```

481      if ( S_AXI_ARESETN = '0' ) then
482          status_register <= (others => '0');
483      else
484          if(or_reduce(reg_ack(gpio_size-1 downto 0))='1')then
485              status_register(gpio_size-1 downto 0) <=
486                  status_register(gpio_size-1 downto 0) xor (
487                      reg_ack(gpio_size-1 downto 0) and
488                      status_register(gpio_size-1 downto 0));
489          else
490              if(reg_gie(0)='1')then
491                  new_intr <= pad_intr_temp;
492              else
493                  new_intr <= (others => '0');
494              end if;
495              status_register(gpio_size-1 downto 0) <=
496                  status_register(gpio_size -1 downto 0) or
497                  new_intr;
498          end if;
499      end if;
500  end process;
501
502 -- User logic ends

```

Codice 1.9: "my gpio v1 0 S00 AXI.vhd"

### 1.3.2 Modifica libreria HAL driver

Avendo aggiunto un'altra funzionalità al GPIO, per la scrittura del relativo driver è necessario aggiungere delle ulteriori funzioni relative all'accesso dei registri sopra citati, nonchè estendere la struttura dati relativa alle istanze di GPIO. Inoltre, volendo svincolare ulteriormente il programmatore dalla conoscenza della periferica, sono implementate delle funzioni per l'accesso diretto a particolari registri.

Di seguito la libreria gpio\_custom.c.

```

1 /**
2 /**
3 ****
4 * @file      gpio_custom.h
5 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
6 *            Sistemi Embedded 2016-2017
7 * @version   V1.0
8 * @date     28-Jun-2017
9 * @brief    Header file della libreria HAL driver per gestione GPIO custom
10 ****
11 */

```

```

12 #ifndef SRC_GPIO_CUSTOM_H_
13 #define SRC_GPIO_CUSTOM_H_

14
15 #include <inttypes.h>
16 #include <assert.h>
17 #include <stdlib.h>
18 #include <stdio.h>

19
20 /**
21 * @brief Definizione struttura dati per la gestione del GPIO
22 */
23
24 typedef struct{
25     uint32_t *base_address; /*!< GPIO Base address*/
26     uint8_t pad_out_offset; /*!< GPIO offset registro di pad_out */
27     uint8_t pad_rw_n_offset; /*!< GPIO offset registro modalità read/write */
28     uint8_t pad_en_offset; /*!< GPIO offset registro abilitazione pin */
29     uint8_t pad_in_offset; /*!< GPIO offset registro pad_in*/
30     uint8_t imr_offset; /*!< GPIO offset registro interrupt mask*/
31     uint8_t sr_offset; /*!< GPIO offset registro di stato*/
32     uint8_t iack_offset; /*!< GPIO offset registro interrupt ack*/
33     uint8_t gie_offset; /*!< GPIO offset registro global interrupt enable*/
34 }gpio_custom_TypeDef;
35
36 /**
37 * @brief Configurazione dell'indirizzo base
38 */
39 #define GPIO_BASE XPAR_MY_GPIO_0_S00_AXI_BASEADDR
40
41 /**
42 * @brief Spiazzamenti registri periferica
43 */
44 #define gpio_custom_PAD_OUT    0x00      /*!< PAD_OUT Register, used to write
45                                     operation */
45 #define gpio_custom_PAD_RW_N   0x04      /*!< PAD_RW_N Register, used to set
46                                     Gpio operation mode (read=1, write=0) */
46 #define gpio_custom_PAD_EN     0x08      /*!< PAD_EN Register, used to enable
47                                     read and write operation */
47 #define gpio_custom_PAD_IN     0x0C      /*!< PAD_IN Register, used to read
48                                     operation */
48 #define gpio_custom_IMR        0x10      /*!< Interrupt Mask Register, used to mask
49                                     interrupt on GPIO single port */
49 #define gpio_custom_SR         0x14      /*!< Status Register, used to save pending
50                                     interrupts */
50 #define gpio_custom_IACK       0x18      /*!< Interrupt Ack Register, used to
51                                     receive interrupts ack */
51 #define gpio_custom_GIE        0x1C      /*!< Global Interrupt Enable, used to
52                                     enable/disable global interrupts of GPIO */

```

```

53 /**
54 * @brief Definizione dei valori alto e basso
55 */
56 typedef enum{
57     LOW,
58     HIGH
59 }gpio_custom_ValueType;
60
61
62
63 void gpio_custom_Init(gpio_custom_TypeDef *gpio, uint32_t base_address);
64
65 /*****Funzioni private per gestione registri generici*****
66 static void gpio_custom_Set_value_reg(gpio_custom_TypeDef *gpio, uint32_t
       reg, uint32_t mask,uint32_t mask_value);
67 static void gpio_custom_Write_mask(gpio_custom_TypeDef *gpio,uint32_t reg,
       uint32_t mask, gpio_custom_ValueType value);
68 static void gpio_custom_Write_one(gpio_custom_TypeDef *gpio,uint32_t reg,
       uint32_t position, gpio_custom_ValueType value);
69 static uint32_t gpio_custom_Read_mask(gpio_custom_TypeDef *gpio,uint32_t reg
       , uint32_t mask);
70 static uint32_t gpio_custom_Read_one(gpio_custom_TypeDef *gpio, uint32_t reg
       , uint32_t position);
71 static void gpio_custom_Toggle_one(gpio_custom_TypeDef *gpio, uint32_t reg,
       uint8_t position);
72 static void gpio_custom_Toggle_mask(gpio_custom_TypeDef *gpio, uint32_t reg,
       uint32_t mask);
73
74 /*****Funzioni utente per gestione specifici registri*****
75 */
76 void gpio_custom_SetEnable(gpio_custom_TypeDef *gpio,uint32_t mask,
    gpio_custom_ValueType value);
77 uint32_t gpio_custom_GetEnable(gpio_custom_TypeDef *gpio,uint32_t mask);
78
79 void gpio_custom_SetValue(gpio_custom_TypeDef *gpio, uint32_t mask,
    gpio_custom_ValueType value);
80 uint32_t gpio_custom_GetValue(gpio_custom_TypeDef *gpio, uint32_t mask);
81
82 void gpio_custom_SetMode(gpio_custom_TypeDef *gpio, uint32_t mask,
    gpio_custom_ValueType mvalue);
83 uint32_t gpio_custom_GetMode(gpio_custom_TypeDef *gpio, uint32_t mask);
84
85 void gpio_custom_SetInterruptMask(gpio_custom_TypeDef *gpio, uint32_t mask,
    gpio_custom_ValueType value);
86 uint32_t gpio_custom_GetInterruptMask(gpio_custom_TypeDef *gpio, uint32_t
    mask);
87

```

```

88 void gpio_custom_SetStatusInterrupt(gpio_custom_TypeDef *gpio, uint32_t mask
89   , gpio_custom_ValueType value);
90 uint32_t gpio_custom_GetStatusInterrupt(gpio_custom_TypeDef *gpio, uint32_t mask
91   , gpio_custom_ValueType value);
92
93 void gpio_custom_SetAckInterrupt(gpio_custom_TypeDef *gpio, uint32_t mask,
94   gpio_custom_ValueType value);
95 uint32_t gpio_custom_GetGlobalInterrupt(gpio_custom_TypeDef *gpio);
96 #endif /* SRC_GPIO_CUSTOM_H_ */

```

Codice 1.10: "gpio custom.h"

Qui, invece, viene riportato il file gpio\_custom.c

```

1 /**
2 ****
3 *
4 * @file      gpio_custom.c
5 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
6 *            Sistemi Embedded 2016-2017
7 * @version   V1.0
8 * @date     28-Jun-2017
9 * @brief    Questo file mette a disposizione un firmware per gestire il
10 *           comportamento della periferica
11 ****
12 */
13
14 /**
15 * @brief  Questa funzione permette di inizializzare correttamente un GPIO
16 *
17 * @param  gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
18 *           le informazioni
19 *           di configurazione della specifica GPIO
20 * @param  base_addr: puntatore ad un intero di 32 bit per il corretto
21 *           indirizzamento della periferica
22 *
23 * @retval none
24 */
25 void gpio_custom_Init(gpio_custom_TypeDef *gpio, uint32_t base_address) {
26
27     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
28     assert(base_address%4==0 && "Base address non allineato a 4 byte");
29     gpio->base_address=(uint32_t*)base_address;

```

```

28     gpio->pad_en_offset=gpio_custom_PAD_EN;
29     gpio->pad_in_offset=gpio_custom_PAD_IN;
30     gpio->pad_out_offset=gpio_custom_PAD_OUT;
31     gpio->pad_rw_n_offset=gpio_custom_PAD_RW_N;
32     gpio->gie_offset=gpio_custom_GIE;
33     gpio->iack_offset=gpio_custom_IACK;
34     gpio->imr_offset=gpio_custom_IMR;
35     gpio->sr_offset=gpio_custom_SR;
36
37 }
38
39 /**
40 * @brief Questa funzione permette di settare all'interno di uno specifico
41 *        registro, dei valori in base ad una determinata maschera di bit
42 *
43 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
44 *              le informazioni
45 *              di configurazione della specifica GPIO
46 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
47 *             indirizzo
48 *             base del registro in cui scrivere
49 * @param mask: intero a 32 bit che specifica una maschera di bit, in
50 *              particolare
51 *              se all'i-esima posizione vi è un valore pari a 1 allora si è
52 *              abilitati a
53 *              scrivere in quella posizione, se 0 allora il valore è mascherato
54 * @param mask_value: intero a 32 bit da scrivere all'interno del registro
55 * @retval none
56 */
57
58 void gpio_custom_Set_value_reg(gpio_custom_TypeDef *gpio, uint32_t reg,
59                               uint32_t mask,uint32_t mask_value) {
60
61     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
62     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
63     ;
64     gpio->base_address[reg/4]=(gpio->base_address[reg/4] & ~mask) | (mask &
65                           mask_value);
66 }
67
68 /**
69 * @brief Questa funzione permette di scrivere all'interno di uno
70 *        specifico
71 *        registro in base ad una determinata maschera di bit
72 *
73 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
74 *              le informazioni
75 *              di configurazione della specifica GPIO

```

```

68 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
69 *           indirizzo
70 *           base del registro in cui scrivere
71 * @param mask: intero a 32 bit che specifica una maschera di bit, in
72 *             particolare
73 *             se all'i-esima posizione vi è un valore pari a 1 allora si è
74 *             abilitati a
75 *             scrivere in quella posizione, se 0 allora il valore è mascherato
76 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
77 *               interno del registro
78 *
79 * @retval none
80 */
81
82 void gpio_custom_Write_mask(gpio_custom_TypeDef *gpio, uint32_t reg, uint32_t
83                             mask, gpio_custom_ValueType value) {
84
85     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
86     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
87     ;
88     if((0x0000000F & mask) == mask)
89         value == HIGH ? gpio_custom_Set_value_reg(gpio, reg, mask, 0xF) :
90                         gpio_custom_Set_value_reg(gpio, reg, mask, 0x0);
91 }
92
93 /**
94 * @brief Questa funzione permette di scrivere un unico valore all'interno
95 *       di uno
96 *       specifico registro in base ad una determinata posizione
97 *
98 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
99 *             le informazioni
100 *             di configurazione della specifica GPIO
101 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
102 *             indirizzo
103 *             base del registro in cui scrivere
104 * @param position: intero a 32 bit che specifica la posizione in cui
105 *                  scrivere
106 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
107 *               interno del registro
108 *
109 * @retval none
110 */
111
112 void gpio_custom_Write_one(gpio_custom_TypeDef *gpio, uint32_t reg, uint32_t
113                            position, gpio_custom_ValueType value){
114     gpio_custom_Write_mask(gpio, reg, 0x1<<position, value);
115 }

```

```

104 /**
105  * @brief Questa funzione permette di leggere all'interno di uno specifico
106  *        registro in base ad una determinata maschera di bit
107  * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
108  *              le informazioni
109  *          di configurazione della specifica GPIO
110  * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
111  *             indirizzo
112  *          base del registro in cui scrivere
113  * @param mask: intero a 32 bit che specifica una maschera di bit, in
114  *              particolare
115  *          se all'i-esima posizione vi è un valore pari a 1 allora si è
116  *          abilitati a
117  *          leggere in quella posizione, viceversa se 0
118  * @retval intero a 32 bit
119 */
120
121 uint32_t gpio_custom_Read_mask(gpio_custom_TypeDef *gpio,uint32_t reg,
122                                uint32_t mask){
123
124     assert(gpio!=NULL && "Errore allocazione di memoria per tipo GPIO");
125     assert(gpio->base_address!=NULL && "Errore allocazione base address GPIO")
126     ;
127
128     if((0x0000000F & mask) == mask){
129         return (gpio->base_address[reg/4] & mask);
130     }
131
132     return -1;
133 }
134
135 /**
136  * @brief Questa funzione permette di leggere un unico valore all'interno
137  *        di uno
138  *        specifico registro in base ad una determinata posizione
139  * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
140  *              le informazioni
141  *          di configurazione della specifica GPIO
142  * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
143  *             indirizzo
144  *          base del registro in cui scrivere
145  * @param position: intero a 32 bit che specifica la posizione in cui
146  *                  leggere
147  *
148  * @retval intero a 32 bit
149 */
150
151 uint32_t gpio_custom_Read_one(gpio_custom_TypeDef *gpio, uint32_t reg,
152                               uint32_t position){
153     return gpio_custom_Read_mask(gpio,reg,0x1<<position) == (0x1<<position);

```

```

142 }
143 /**
144 * @brief Questa funzione permette il toggle di un singolo valore all'
145 * interno di
146 * uno specifico registro in base ad una determinata posizione
147 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
148 * le informazioni
149 * di configurazione della specifica GPIO
150 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
151 * indirizzo
152 * base del registro in cui effettuare il toggle
153 * @param position: intero a 8 bit che specifica la posizione in cui
154 * effettuare
155 * il toggle
156 *
157 * @retval none
158 */
159
160 void gpio_custom_Toggle_one(gpio_custom_TypeDef *gpio, uint32_t reg,uint8_t
161 position) {
162 gpio_custom_Read_one(gpio,reg,position) == HIGH ? gpio_custom_Write_one(
163 gpio,reg ,position,LOW) : gpio_custom_Write_one(gpio,reg,position,HIGH
164 );
165 }
166 /**
167 * @brief Questa funzione permette il toggle di un gruppo di bit all'
168 * interno di
169 * uno specifico registro in base ad una determinata maschera
170 *
171 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
172 * le informazioni
173 * di configurazione della specifica GPIO
174 * @param reg: intero a 32 bit che indica lo spiazzamento rispetto all'
175 * indirizzo
176 * base del registro in cui effettuare il toggle
177 * @param position: intero a 32 bit che specifica la maschera di bit, in
178 * particolare se il valore i-esimo è pari a 1 allora in quella
179 * posizione
180 * si è abilitati ad effettuare il toggle, viceversa non si effettua
181 * tale
182 * operazione
183 *
184 * @retval none
185 */
186
187 void gpio_custom_Toggle_mask(gpio_custom_TypeDef *gpio, uint32_t reg,
188 uint32_t mask){
```

```

178     int i;
179     for(i=0;i<4;i++) {
180         if((0x1<<i) == ((0x1<<i) & mask)) {
181             gpio_custom_Toggle_one(gpio,reg,i);
182         }
183     }
184 }
185
186 /**
187 * @brief Questa funzione permette di settare il registro di abilitazione
188 *        in base ad una
189 *        maschera di bit
190 *
191 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
192 *        le informazioni
193 *        di configurazione della specifica GPIO
194 * @param mask: intero a 32 bit che specifica una maschera di bit, in
195 *        particolare
196 *        se all'i-esima posizione vi è un valore pari a 1 allora si è
197 *        abilitati a
198 *        scrivere in quella posizione, se 0 allora il valore è mascherato
199 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
200 *        interno del registro
201 *
202 * @retval none
203 */
204
205 void gpio_custom_SetEnable(gpio_custom_TypeDef *gpio,uint32_t mask,
206                             gpio_custom_ValueType value){
207     gpio_custom_Write_mask(gpio,gpio_custom_PAD_EN, mask, value);
208     printf("Settaggio registro abilitazione\n");
209 }
210
211 /**
212 * @brief Questa funzione permette di leggere il registro di abilitazione
213 *        in base ad una
214 *        maschera di bit
215 *
216 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
217 *        le informazioni
218 *        di configurazione della specifica GPIO
219 * @param mask: intero a 32 bit che specifica una maschera di bit, in
220 *        particolare
221 *        se all'i-esima posizione vi è un valore pari a 1 allora si è
222 *        abilitati a
223 *        scrivere in quella posizione, se 0 allora il valore è mascherato
224 * @retval intero a 32 bit
225 */

```

```

217 uint32_t gpio_custom_GetEnable(gpio_custom_TypeDef *gpio, uint32_t mask) {
218     printf("Lettura da registro abilitazione\n");
219     return gpio_custom_Read_mask(gpio, gpio_custom_PAD_EN, mask);
220 }
221
222 /**
223 * @brief Questa funzione permette di settare il registro pad_out in base
224 * ad una
225 *      maschera di bit
226 *
227 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
228 *      le informazioni
229 *      di configurazione della specifica GPIO
230 * @param mask: intero a 32 bit che specifica una maschera di bit, in
231 *      particolare
232 *          se all'i-esima posizione vi è un valore pari a 1 allora si è
233 *          abilitati a
234 *              scrivere in quella posizione, se 0 allora il valore è mascherato
235 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
236 *      interno del registro
237 *
238 * @retval none
239 */
240
241 void gpio_custom_SetValue(gpio_custom_TypeDef *gpio, uint32_t mask,
242     gpio_custom_ValueType value) {
243     gpio_custom_Write_mask(gpio, gpio_custom_PAD_OUT, mask, value);
244     printf("Scrittura valori su periferica\n");
245 }
246
247 /**
248 * @brief Questa funzione permette di leggere il registro pad_in in base
249 * ad una
250 *      maschera di bit
251 *
252 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
253 *      le informazioni
254 *      di configurazione della specifica GPIO
255 * @param mask: intero a 32 bit che specifica una maschera di bit, in
256 *      particolare
257 *          se all'i-esima posizione vi è un valore pari a 1 allora si è
258 *          abilitati a
259 *              scrivere in quella posizione, se 0 allora il valore è mascherato
260 * @retval intero a 32 bit
261 */
262
263 uint32_t gpio_custom_GetValue(gpio_custom_TypeDef *gpio, uint32_t mask) {
264     printf("Lettura valori da periferica\n");

```

```

256     return gpio_custom_Read_mask(gpio,gpio_custom_PAD_IN, mask);
257 }
258
259 /**
260 * @brief Questa funzione permette di settare il registro read/write_n in
261 *        base ad una
262 *        maschera di bit, ossia il modo di funzionamento del GPIO se di input
263 *        o di output
264 *
265 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
266 *        le informazioni
267 *        di configurazione della specifica GPIO
268 * @param mask: intero a 32 bit che specifica una maschera di bit, in
269 *        particolare
270 *        se all'i-esima posizione vi è un valore pari a 1 allora si è
271 *        abilitati a
272 *        scrivere in quella posizione, se 0 allora il valore è mascherato
273 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
274 *        interno del registro
275 *
276 * @retval none
277 */
278
279 void gpio_custom_SetMode(gpio_custom_TypeDef *gpio, uint32_t mask,
280                         gpio_custom_ValueType value){
281     gpio_custom_Write_mask(gpio,gpio_custom_PAD_RW_N, mask, value);
282     printf("Definita modalità periferica\n");
283 }
284
285 /**
286 * @brief Questa funzione permette di leggere il registro read/write_n in
287 *        base ad una
288 *        maschera di bit, ossia di capire la modalità di funzionamento del
289 *        GPIO
290 *
291 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
292 *        le informazioni
293 *        di configurazione della specifica GPIO
294 * @param mask: intero a 32 bit che specifica una maschera di bit, in
295 *        particolare
296 *        se all'i-esima posizione vi è un valore pari a 1 allora si è
297 *        abilitati a
298 *        scrivere in quella posizione, se 0 allora il valore è mascherato
299 * @retval intero a 32 bit
300 */
301
302 uint32_t gpio_custom_GetMode(gpio_custom_TypeDef *gpio, uint32_t mask) {
303     printf("Lettura tipologia periferica\n");
304     return gpio_custom_Read_mask(gpio,gpio_custom_PAD_RW_N, mask);
305 }
```

```

293 }
294
295 /**
296 * @brief Questa funzione permette di settare il registro interrupt
297 * mascherate in base ad una
298 * maschera di bit
299 *
300 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
301 * le informazioni
302 * di configurazione della specifica GPIO
303 * @param mask: intero a 32 bit che specifica una maschera di bit, in
304 * particolare
305 * se all'i-esima posizione vi è un valore pari a 1 allora si è
306 * abilitati a
307 * scrivere in quella posizione, se 0 allora il valore è mascherato
308 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
309 * interno del registro
310 *
311 * @retval none
312 */
313
314 /**
315 * @brief Questa funzione permette di leggere il registro delle interrupt
316 * mascherate in base ad una
317 * maschera di bit.
318 *
319 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
320 * le informazioni
321 * di configurazione della specifica GPIO
322 * @param mask: intero a 32 bit che specifica una maschera di bit, in
323 * particolare
324 * se all'i-esima posizione vi è un valore pari a 1 allora si è
325 * abilitati a
326 * scrivere in quella posizione, se 0 allora il valore è mascherato
327 * @retval intero a 32 bit
328 */
329
330 uint32_t gpio_custom_SetInterruptMask(gpio_custom_TypeDef *gpio, uint32_t mask,
331                                     gpio_custom_ValueType value) {
332     gpio_custom_Write_mask(gpio, gpio_custom_IMR, mask, value);
333     printf("Scrittura in registro delle interrupt mascherate\n");
334 }
335
336 /**
337 * @brief Questa funzione permette di leggere il registro delle interrupt
338 * mascherate in base ad una
339 * maschera di bit.
340 *
341 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
342 * le informazioni
343 * di configurazione della specifica GPIO
344 * @param mask: intero a 32 bit che specifica una maschera di bit, in
345 * particolare
346 * se all'i-esima posizione vi è un valore pari a 1 allora si è
347 * abilitati a
348 * scrivere in quella posizione, se 0 allora il valore è mascherato
349 * @retval intero a 32 bit
350 */
351
352 uint32_t gpio_custom_GetInterruptMask(gpio_custom_TypeDef *gpio, uint32_t
353                                       mask) {
354     printf("Lettura tipologia periferica\n");
355     return gpio_custom_Read_mask(gpio, gpio_custom_IMR, mask);
356 }
```

```

331 /**
332 * @brief Questa funzione permette di settare il registro di stato delle
333 * interrupt in base
334 * ad una maschera di bit
335 *
336 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
337 * le informazioni
338 * di configurazione della specifica GPIO
339 * @param mask: intero a 32 bit che specifica una maschera di bit, in
340 * particolare
341 * se all'i-esima posizione vi è un valore pari a 1 allora si è
342 * abilitati a
343 * scrivere in quella posizione, se 0 allora il valore è mascherato
344 * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
345 * interno del registro
346 *
347 * @retval none
348 */
349
350 void gpio_custom_SetStatusInterrupt(gpio_custom_TypeDef *gpio, uint32_t mask
351 , gpio_custom_ValueType value){
352     gpio_custom_Write_mask(gpio, gpio_custom_SR, mask, value);
353     printf("Scrittura in registro degli interrupt pendenti\n");
354 }
355
356 /**
357 * @brief Questa funzione permette di leggere il registro di stato delle
358 * interrupt in base ad una
359 * maschera di bit.
360 *
361 * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
362 * le informazioni
363 * di configurazione della specifica GPIO
364 * @param mask: intero a 32 bit che specifica una maschera di bit, in
365 * particolare
366 * se all'i-esima posizione vi è un valore pari a 1 allora si è
367 * abilitati a
368 * scrivere in quella posizione, se 0 allora il valore è mascherato
369 * @retval intero a 32 bit
370 */
371
372 uint32_t gpio_custom_GetStatusInterrupt(gpio_custom_TypeDef *gpio, uint32_t
373 mask) {
374     printf("Lettura tipologia periferica\n");
375     return gpio_custom_Read_mask(gpio, gpio_custom_SR, mask);
376 }
377
378 /**
379 * @brief Questa funzione permette di settare il registro di ack delle

```

```

369         interrupt in base
370             ad una maschera di bit
371
372     *
373     * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
374         le informazioni
375         di configurazione della specifica GPIO
376     * @param mask: intero a 32 bit che specifica una maschera di bit, in
377         particolare
378         se all'i-esima posizione vi è un valore pari a 1 allora si è
379         abilitati a
380         scrivere in quella posizione, se 0 allora il valore è mascherato
381     * @param value: valore di tipo gpio_custom_ValueType da scrivere all'
382         interno del registro
383
384     *
385     * @retval none
386 */
387
388 void gpio_custom_SetAckInterrupt(gpio_custom_TypeDef *gpio, uint32_t mask,
389     gpio_custom_ValueType value) {
390     gpio_custom_Write_mask(gpio, gpio_custom_IACK, mask, value);
391     printf("Invia interruto ack alla periferica\n");
392 }
393
394 /**
395     * @brief Questa funzione permette di settare il registro delle interrupt
396         globali
397
398     *
399     * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
400         le informazioni
401         di configurazione della specifica GPIO
402     * @retval none
403 */
404
405 void gpio_custom_SetGlobalInterrupt(gpio_custom_TypeDef *gpio,
406     gpio_custom_ValueType value) {
407     gpio_custom_Write_one(gpio, gpio_custom_GIE, 0x0, value);
408     printf("Scrittura in registro interruto globale\n");
409 }
410
411 /**
412     * @brief Questa funzione permette di leggere il registro delle interrupt
413         globali
414
415     *
416     * @param gpio: puntatore alla struttura gpio_custom_TypeDef che contiene
417         le informazioni
418         di configurazione della specifica GPIO
419     * @retval intero a 32 bit
420 */
421
422

```

```
407 uint32_t gpio_custom_GetGlobalInterrupt(gpio_custom_TypeDef *gpio) {  
408     printf("Lettura interrupt globale");  
409     return gpio_custom_Read_one(gpio, gpio_custom_GIE, 0x0);  
410 }  
411 }
```

Codice 1.11: "gpio custom.c"

# Capitolo 2

## BSP custom per STM32 F4 Discovery

### 2.1 Traccia

Si definisca e si implementi una BSP custom per il controllo dei led e del bottone utente per un microcontrollore STM32 F4 Discovery.

### 2.2 Procedimento

Trattandosi di un primo approccio alla programmazione su microcontrollore, i paragrafi presentati mettono in evidenza dapprima come creare un nuovo progetto in *System Workbench for STM32*, successivamente un'implementazione di una BSP (Board Support Package) custom .

#### 2.2.1 Creazione di un nuovo progetto

Nell'IDE di sviluppo dirigersi in **File** → **New** → **C Project**. Qui viene mostrato un wizard in cui è necessario definire il nome del progetto. Si sceglie di salvare il progetto nel workspace selezionato spuntando il flag *Use default location*, ed infine si specifica il tipo di progetto da creare. Selezionare **Executable** → **Empty Project**, e si sceglie come Toolchains *Ac6 STM32 MCU GCC*, come mostrato in fig. 2.1.

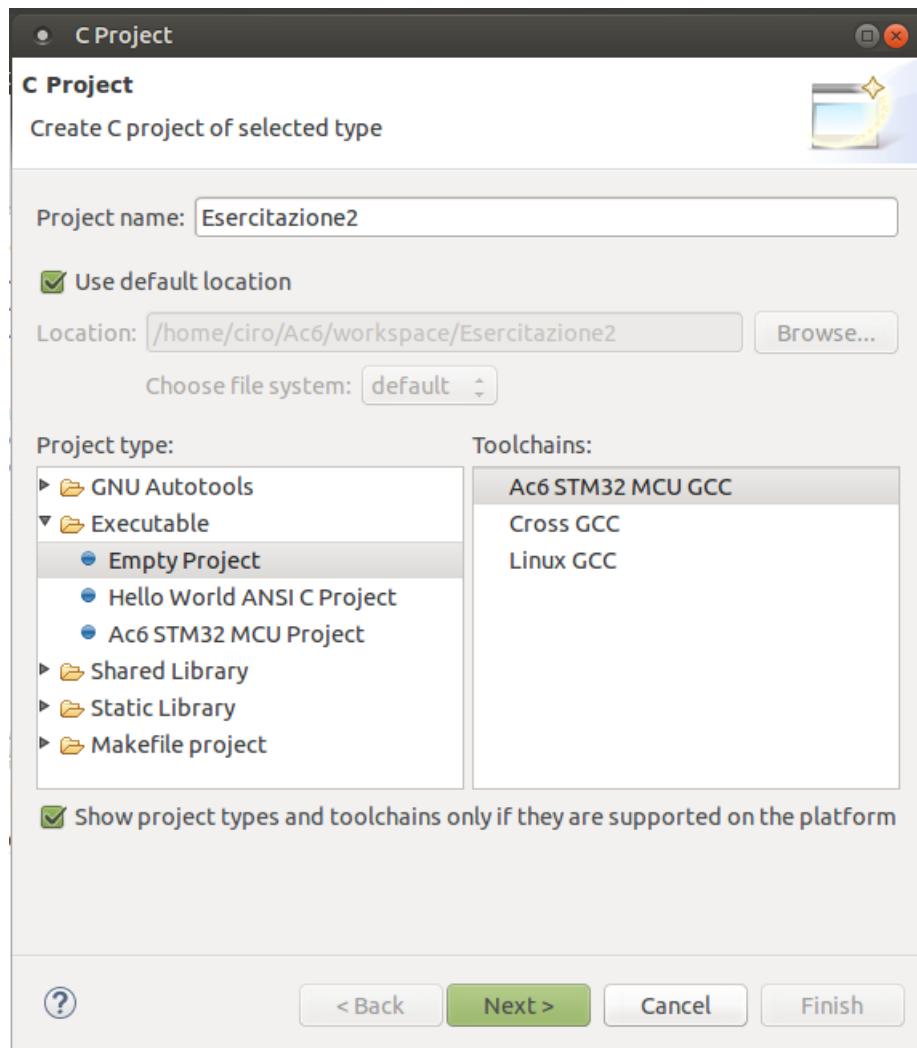


Figura 2.1: C Project

Cliccando su **Next** si sceglie la configurazione da usare e si spuntano sia *Debug* che *Release*, fig. 2.2.

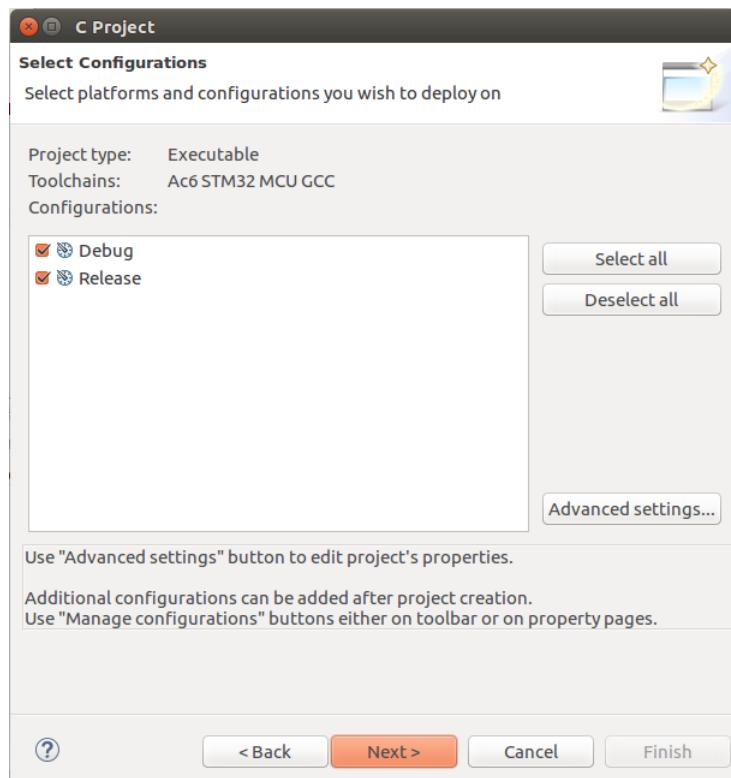


Figura 2.2: Select Configurations

Nella finestra successiva, viene chiesto di selezionare la board da associare al progetto che si sta creando. In **Series** si sceglie **STM32F4** e in **Boards** si seleziona **STM32F4DISCOVERY**, fig. 2.3.

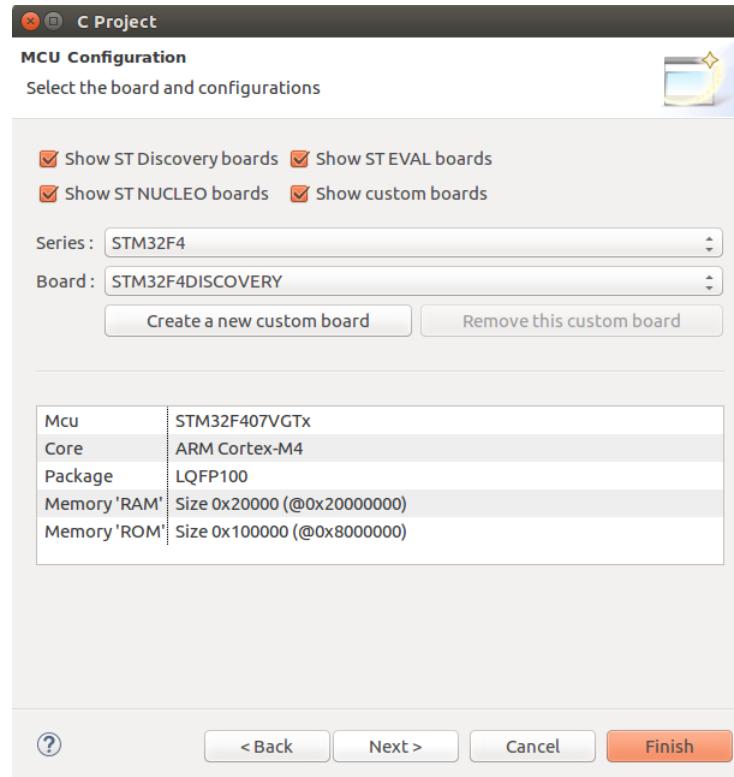


Figura 2.3: MCU Configuration

In **Project Firmware configuration** si sceglie Hardware Abstraction Layer (Cube HAL).

Se l'ambiente non riconosce che esiste già il firmware selezionato, permette di fare direttamente il download, fig. 2.4.

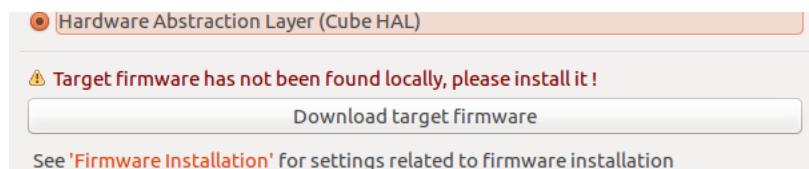


Figura 2.4: Download target firmware

Essendo lo scopo di questa documentazione definire una bsp custom, non è necessario aggiungere ulteriori driver o utilities di terze parti, fig. 2.5.

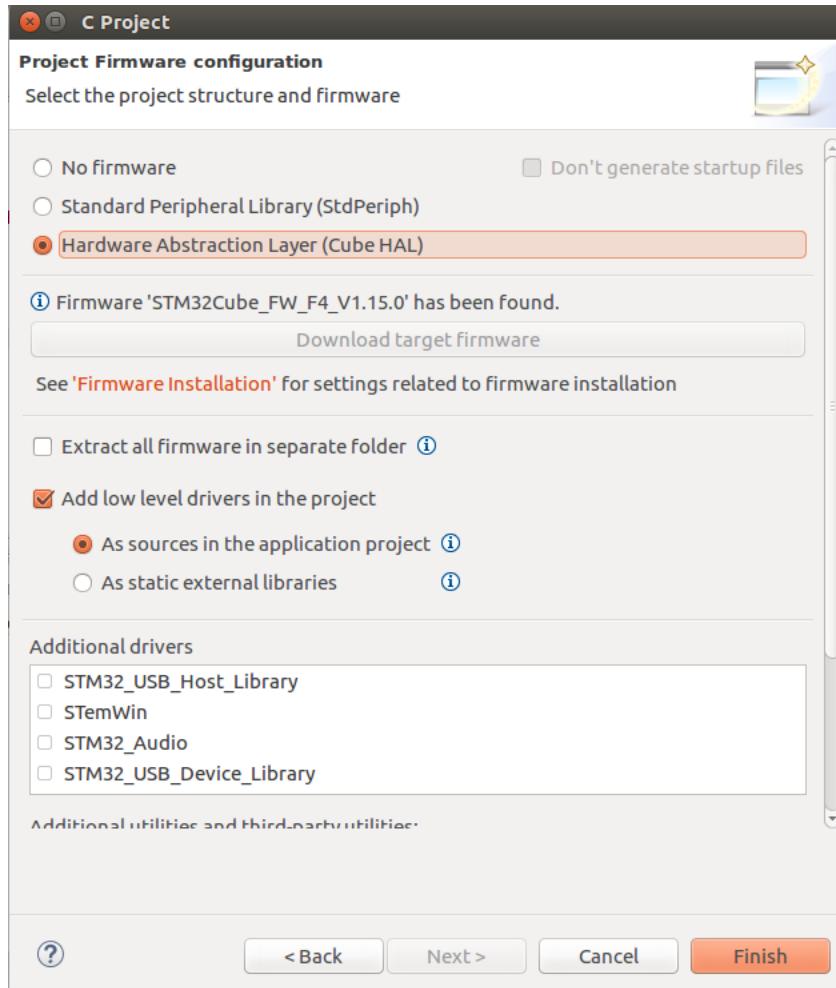


Figura 2.5: Project Firmware configuration

## 2.2.2 Funzioni

Passando ora alla parte operativa, è necessario creare due file: il primo è un header in cui viene definita la libreria per la bsp, il secondo invece per l'implementazione di essa. Dunque in **Project Explorer** da **Esercitazione2** → **src** si clicca con il tasto destro, **New** → **Source File** per il .c e **New** → **Header File** per il .h.

Di seguito, si elencano le funzioni implementate:

- init\_led\_user;
- init\_button\_user;
- write\_led\_user;
- read\_button\_user;
- init\_timer2;

- start\_timer2;
- is\_timer2\_stopped.

Il seguente codice evidenzia i prototipi delle funzioni appena elencate.

```

1  /**
2   ****
3   * @file      my_led_button_bsp.h
4   * @author    Colella Gianni - Guida Ciro - Lombardi Daniele
5   *           Group IV - Sistemi Embedded 2016-2017
6   * @version   V1.0
7   * @date      12-June-2017
8   * @brief     Libreria per BSP custom STM32F4 Discovery
9   ****
10 */
11
12 #ifndef MY_BSP_CUSTOM_H_
13 #define MY_BSP_CUSTOM_H_
14
15 /* Librerie Incluse
16  -----*/
16 #include "stm32f407xx.h"
17
18 /* Prototipi funzioni
19  -----*/
20 void init_led_user(); /*!< inizializza i led utente */
21 void init_button_user(); /*!< inizializza il bottone utente*/
22
23 void write_led_user(); /*!< scrittura valori su led utente */
24 uint8_t read_button_user(); /*!< lettura bottone utente */
25
26 void init_timer2(uint32_t delay); /*!< inizializzazione timer */
27 void start_timer2(); /*!< start conteggio*/
28 uint8_t is_timer2_stopped(); /*!< interroga se il timer ha terminato il
29                           conteggio*/
30
31#endif /* MY_BSP_CUSTOM_H_ */

```

Codice 2.1: my led button bsp.h

### 2.2.2.1 init\_led\_user

Tale funzione permette di settare i 4 led utente in modo tale da poter visualizzare i valori di un nibble. Si provvede a fornire l'abilitazione alla GPIOD. I pin da 12 a 15 vengono settati a funzionare come output (tali pin di GPIOD sono quelli direttamente collegati ai led) ed infine viene resettato

il relativo registro d'uscita in modo tale da pulirlo da eventuali valori scritti durante esecuzioni precedenti.

```

15 /**
16 * @brief Questa funzione inizializza i 4 led utente in modo tale da poter
17 *        essere
18 *        utilizzati come display di valori a 4 bit.
19 * @param None.
20 * @retval None.
21 */
22 void init_led_user(){
23
24     /*Abilitazione a GPIOD*/
25     RCC->AHB1ENR |= (1<<3);
26
27     /*Si indica ai pin 15-14-13-12 di PD di funzionare come output*/
28     GPIOD->MODER|= (0x55<<24);
29
30     /*Si spengono eventualmente i led accesi da un programma eseguito in
31      precedenza*/
32     GPIOD->ODR=0x0;
33 }
```

Codice 2.2: my led button bsp.c

### 2.2.2.2 init\_button\_user

Prevede il settaggio del bottone utente a funzionare come periferica di input, in particolare viene fornita l'abilitazione alla periferica GPIOA e viene settato il pin PA0 a funzionare da input, poiché quest'ultimo è quello direttamente collegato al bottone.

```

34 /**
35 * @brief Questa funzione inizializza il bottone utente.
36 * @param None.
37 * @retval None.
38 */
39
40 void init_button_user(){
41
42     /*Abilitazione GPIOA*/
43     RCC->AHB1ENR |= 1;
44
45     /*Valore di reset al moder di GPIOA, si indica a PA0 di funzionare da
46      input*/
47     GPIOA->MODER=0xA8000000;
48 }
```

Codice 2.3: my led button bsp.c

### 2.2.2.3 init\_timer2

Viene inizializzata la periferica TIM2 per funzionare da contatore alla rovescia. In particolare, viene fornita l'abilitazione, viene indicato che un evento di timeout genera un'interruzione, vengono settati il numero di colpi di clock da contare e, infine, viene scartato l'utilizzo del prescaler e resettato un'eventuale conteggio iniziato in precedenza.

```

49 void init_timer2(uint32_t clock) {
50
51     /*Abilitazione al Timer2*/
52     RCC->APB1ENR|=1;
53
54     /*Si indica al Timer2 che solo un overflow/underflow può generare 1'
55      interruzione*/
56     TIM2->CR1|=(1<<2);
57
58     /*Numero di colpi di clock da contare*/
59     TIM2->ARR=clock;
60
61     /*Non si utilizza il prescaler (divisione per 1)*/
62     TIM2->PSC=0;
63
64     /*Azzeramento conteggio*/
65     TIM2->CNT=0;
66 }
```

Codice 2.4: my led button bsp.c

### 2.2.2.4 write\_led

La funzione permette semplicemente di scrivere un valore direttamente sul led.

```

75 /**
76  * @brief Questa funzione permette di scrivere sul led utente.
77  * @param [in] value: intero a 8 bit che indica il valore da mostrare sul
78  *                   led
79  *
80  * @retval None.
81  */
82
83 void write_led(uint8_t value){
84     GPIOD->ODR|=value<<12;
85 }
```

Codice 2.5: my led button bsp.c

### 2.2.2.5 `read_button`

Tramite un'operazione di AND con un bit pari a 1, viene interrogato direttamente l'LSB del registro di output di GPIOA mappato fisicamente sul bottone utente. La funzione restituisce 1 se il bottone è premuto, 0 altrimenti.

```

86 /**
87  * @brief Questa funzione permette di leggere lo stato del bottone utente.
88  * @param [in] value: intero a 8 bit che indica il valore da mostrare sui
89  *                   led
90  *
91  * @retval Intero a 8 bit:
92  *         1 bottone premuto;
93  *         0 bottone non premuto.
94  */
95
96 uint8_t read_button() {
97     return GPIOA->IDR&1;
}

```

Codice 2.6: my led button bsp.c

### 2.2.2.6 `start_timer2`

Funzione per indicare alla periferica TIM2 di iniziare il conto alla rovescia: lo start viene indicato settando a 1 l'LSB del registro di controllo.

```

99 /**
100  * @brief Questa funzione indica al timer2 l'inizio di un countdown.
101  *
102  * @retval None.
103  */
104 void start_timer2() {
105     TIM2->CR1=1;
106 }

```

Codice 2.7: my led button bsp.c

### 2.2.2.7 `is_timer2_stopped`

Per capire se il conteggio del timer è esaurito, si può interrogare il registro di stato della periferica. A tal scopo è necessario leggere il valore dell'LSB con lo stesso metodo visto in par.2.2.2.5.

```

108 /**
109  * @brief Questa funzione permette di interrogare il timer2 se il
110  *        countdown è avvenuto.
111  *
112  * @retval Intero a 8 bit:
113  *         1 countdown avvenuto;
114  *         0 countdown non avvenuto.

```

```

114 */
115
116 uint8_t is_timer2_stopped() {
117
118     /*Se l'ultimo bit dello status register è pari a 1 allora il conteggio è
119      terminato*/
120     if(TIM2->SR&1) {
121         /*Si resetta l'ultimo bit per il prossimo conteggio*/
122         TIM2->SR=0;
123         return 1;
124     }
125     return 0;
}

```

Codice 2.8: my led button bsp.c

### 2.2.3 Driver

In questo paragrafo vengono mostrati 3 possibili esempi di driver che utilizzano le funzioni precedentemente descritte. Di seguito, l'elenco dei driver implementati:

- toggleAll;
- counter;
- supercar.

Per poter richiamare i driver in qualsiasi applicativo, viene proposto un header file adatto allo scopo.

```

1 /**
2 ****
3 * @file      driver.h
4 * @author Colella Gianni - Guida Ciro - Lombardi Daniele
5 *           Group IV - Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date      12-June-2017
8 * @brief     Libreria per pilotaggio periferiche
9 ****
10 */
11
12 #ifndef DRIVER_H_
13 #define DRIVER_H_
14
15 /* Librerie Incluse
16 -----------*/
16 #include "my_bsp_custom.h"
17

```

```

18 /* Prototipi funzioni
-----*/
19 void toggleAll(); /*! Toggle dei led utente*/
20 void supercar(); /*! Shift a destra e sinistra di un bit e mostrato sui led
 */
21 void counter(); /*! Conteggio con output sui led*/
22
23 #endif /* DRIVER_H_ */

```

Codice 2.9: driver.h

### 2.2.3.1 toggleAll

Come è intuibile dal nome tale driver permette il toggle dei 4 led utente; tale evento viene scatenato alla pressione e al successivo rilascio del bottone utente

```

21 /**
22 * @brief Questa funzione permette il toggle di tutti e 4 i led utente.
23 * @param None.
24 * @retval None.
25 */
26 void toggleAll(){
27
28     /*Si attende pressione bottone*/
29     if(read_button()){
30         /*Si attende rilascio bottone*/
31         while(read_button());
32         /*Se i led sono spenti allora si accendono*/
33         if(!on){
34             write_led(0xF);
35             on=1;
36         }
37         else{
38             /*Altrimenti si spengono*/
39             write_led(0x0);
40             on=0;
41         }
42     }
43 }

```

Codice 2.10: driver.c

### 2.2.3.2 counter

Il seguente driver permette di effettuare un conteggio modulo 16 premendo il bottone utente. Il contenuto della variabile di conteggio viene mostrato sui led tramite la funzione write\_led().

46

/\*\*

```

47 * @brief Questa funzione permette di implementare un contatore modulo 16 e
48     di visualizzare il
49     contenuto sui led.
50 * @note Il conteggio avviene premendo e successivamente rilasciando il
51     bottone utente
52 * @param None.
53 * @retval None.
54 */
55 void counter(){
56     /*Se viene premuto il bottone si computa il conteggio*/
57     if(read_button()){
58         /*Si attende che il bottone sia rilasciato*/
59         while(read_button());
60         count++;
61         /*Stampa conteggio sui led*/
62         write_led(count);
63         /*Se il conteggio è arrivato a 15 si resetta la variabile associata*/
64         if(count==15)
65             count=0;
66     }
67 }
```

Codice 2.11: driver.c

### 2.2.3.3 supercar

Terzo ed ultimo esempio di driver proposto mostra come shiftare da destra verso sinistra e viceversa un bit lungo i 4 led utente, simulando il lampeggiamento dei led sul muso anteriore di K.I.T.T. Per un risultato apprezzabile sono sfruttate le funzioni relative alla periferica TIM2 la quale lancia un segnale di countdown ogni 500 ms.

```

68 void supercar(){
69     /*Inizio conteggio*/
70     start_timer2();
71     /*Se il conteggio è terminato allora inizia lo shift*/
72     if(is_timer2_stopped()){
73         /*Se non è stato illuminato il led più a sinistra si shifta da destra a
74             sinistra*/
75         if(!dx){
76             val_shift=val_shift<<1;
77             write_led(val_shift);
78             /*Se il led illuminato è quello più a sinistra si setta una
79                 variabile pari a 1
80                 * per iniziare lo shift verso destra*/
81             if(val_shift==0x8)
82                 dx=1;
83         }
84     else{
```

```

83     /*Computazione dello shift verso destra*/
84     val_shift=val_shift>>1;
85     write_led(val_shift);
86     /*Se il led illuminato è quello più a destra si setta una variabile
87      pari a 0
88      * per iniziare lo shift verso sinistra*/
89     if(val_shift==0x1)
90         dx=0;
91     }
92 }
```

Codice 2.12: driver.c

## 2.2.4 Applicazione

Infine viene proposto il seguente applicativo per l'utilizzo dei driver precedentemente descritti.

```

1 /**
2 ****
3 * @file      my_led_button_bsp.c
4 * @author    Colella Gianni - Guida Ciro - Lombardi Daniele
5 *          Group IV - Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date      12-June-2017
8 * @brief     Main per pilotaggio periferiche
9 ****
10 */
11
12 /* Librerie Incluse
13  -----
14 #include "driver.h"
15
16 /* MACRO per selezione driver
17  -----
18 #define TOGGLEALL 0
19 #define COUNTER 1
20 #define SUPERCAR 2
21
22 /* Prototipi funzioni
23  -----
24 void loop(short int);
25 void init();
26
27 int main(void)
28 {
29
30 }
```

```

27     init();
28     for(;;) loop(SUPERCAR);
29     return 1;
30 }
31
32 /**
33 * @brief Questa funzione inizializza led, bottone utente e timer2.
34 * @param None.
35 * @retval None.
36 */
37
38 void init(){
39     init_led_user();
40     init_button_user();
41     init_timer2(8000000);
42 }
43
44 /**
45 * @brief Questa funzione permette di selezionare ed eseguire un driver
46 *        tra quelli implementati.
47 * @param None.
48 * @retval None.
49 */
50
51 void loop(short int f){
52
53     switch(f){
54
55         case TOGGLEALL:
56             toggleAll();
57             break;
58
59         case COUNTER:
60             counter();
61             break;
62
63         case SUPERCAR:
64             supercar();
65             break;
66
67         default:
68             break;
69     }
70 }
```

Codice 2.13: main.c

# Capitolo 3

## Esempi di utilizzo delle interruzioni su board STM

### 3.1 Traccia

Si mostrino degli esempi di utilizzo delle interruzioni su board STM.

### 3.2 Procedimento

Mediante la pressione del bottone utente si vuole dimostrare come è possibile incrementare il conteggio di una variabile e il contenuto mostrarlo sui led in notazione binaria. A tale scopo si propongono due soluzioni funzionalmente equivalenti, ma diverse nell'implementazione:

- interruzione a livello bare metal, in cui è necessario gestire il processo di comunicazione tra l'**NVIC** (Nested Vector Interrupt Controller) e la periferica di GPIO;
- interruzione a livello **HAL**, servendosi di una callback.

Entrambe le soluzioni prevedono una parte comune in cui vengono inizializzati i led e il bottone utente, in particolare, si indica al bottone che ad ogni pressione e successivo rilascio di esso deve essere generato un'interruzione hardware. Nel caso di utilizzo di un microcontrollore STM32 F4 Discovery, il relativo pin di GPIO del bottone è mappato sul generatore di interruzione EXTI0

#### 3.2.1 Soluzione bare metal

Tale tipo di soluzione prevede che sia gestito il meccanismo di acknowledge e pulizia del relativo registro di stato. In particolare, all'entrata della ISR viene richiamata la funzione **NVIC\_ClearPending**, la quale permette all'NVIC di segnalare alla periferica di GPIO che l'interruzione è stata servita; in uscita, invece, utilizzando la macro **HAL\_GPIO\_EXTI\_CLEAR\_IT**, viene pulito il registro di stato della periferica in modo tale che il processore possa effettuare il cambio di contesto, tornando all'esecuzione del programma principale.

<sup>1</sup> / \*\*

## CAPITOLO 3. ESEMPI DI UTILIZZO DELLE INTERRUZIONI SU BOARD STM

```
2 ****
3 * @file      main.c
4 * @author Colella Gianni - Guida Ciro - Lombardi Daniele
5 *   Group IV - Sistemi Embedded 2016-2017
6 * @version V1.0
7 * @date     16-June-2017
8 * @brief    Esempio di utilizzo delle interruzioni BARE metal su board
9 *           STM32
10 ****
11 */
12 /* Librerie Incluse
13 -----------*/
13 #include "stm32f4xx.h"
14 #include "stm32f4_discovery.h"
15 #include "stm32f4xx_hal_gpio.h"
16
17 /* variabili globali
18 -----------*/
18 short int count=0;
19
20 /* Prototipi funzioni
21 -----------*/
21 void setup();
22 void loop();
23
24 /**
25 * @brief Interrupt Service Routine lanciata da EXTI0.
26 * @note Una volta invocata la routine essa implementa una logica custom,
27 *       in particolare
28 *         una volta schicciato il bottone utente si provvede ad
29 *           incrementare di 1 una variabile
30 *             e mostrato il conteggio sui led.
31 * @param None.
32 * @retval None.
31 */
32
33 void EXTI0_IRQHandler(){
34     /*L'NVIC segnala alla periferica su EXTI0 che è stata servita */
35     NVIC_ClearPendingIRQ(EXTI0_IRQn);
36     while(GPIOA->IDR&1);
37         count++;
38         GPIOD->ODR=count<<12;
39         if(count==15)
40             count=0;
41     /*Una volta servita l'interrupt, viene segnala la linea di EXTI da pulire
42     * in modo tale da ritornare nel main*/
```

```

43     __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_0);
44 }
45
46 int main(void)
47 {
48     setup();
49     for(;;) loop();
50     return 1;
51 }
52 /**
53 * @brief Funzione per inizializzare le periferiche da utilizzare.
54 * @param None.
55 * @retval None.
56 */
57 void setup()
58 {
59     HAL_Init();
60     BSP_LED_Init(LED3);
61     BSP_LED_Init(LED4);
62     BSP_LED_Init(LED5);
63     BSP_LED_Init(LED6);
64     BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);
65 }
66
67 void loop()
68 }
```

### 3.2.2 Soluzione HAL

Qui, invece, è necessario solamente la reimplementazione di una callback: tutti i meccanismi di gestione dell'interruzione sono astratti al programmatore. Per tal motivo, è necessario solamente definire una logica custom da eseguire ogni qualvolta venga richiamata la ISR.

```

71 /**
72 ****
73 * @file      main.c
74 * @author    Colella Gianni - Guida Ciro - Lombardi Daniele
75 *           Group IV - Sistemi Embedded 2016-2017
76 * @version   V1.0
77 * @date      16-June-2017
78 * @brief     Esempio di utilizzo delle interruzioni mediante HAL su board
79 *           STM32
80 ****
81 */
```

```

82  /* Librerie Incluse
83  -----
84 #include "stm32f4xx.h"
85 #include "stm32f4_discovery.h"
86 #include "stm32f4xx_hal.h"
87
88 /* Variabili globali
89 -----
90 short int count=0;
91
92 void setup();
93 void loop();
94
95 /**
96 * @brief Interrupt Service Routine lanciata da EXTI0.
97 * @note Una volta invocata la routine essa implementa una logica custom,
98 *       in particolare
99 *           una volta schicciato il bottone utente si provvede ad
100 *           incrementare di 1 una variabile
101 *           e mostrato il conteggio sui led. Tale funzione reimplementa una
102 *           callback
103 * @param [in] GPIO_Pin: intero a 16 bit che indica il pin di GPIO da dove
104 *       è stata
105 *           lanciata l'interruzione.
106 * @retval None.
107 */
108 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
109     count++;
110     GPIOD->ODR=count<<12;
111     if(count==15)
112         count=0;
113 }
114
115 int main(void)
116 {
117     setup();
118     for(;;) loop();
119     return 1;
120 }
121
122 /**
123 * @brief Funzione per inizializzare le periferiche da utilizzare.
124 * @param None.
125 * @retval None.
126 */
127 void setup(){
128     HAL_Init();
129 }
```

```

125     BSP_LED_Init(LED3);
126     BSP_LED_Init(LED4);
127     BSP_LED_Init(LED5);
128     BSP_LED_Init(LED6);
129     BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);
130
131 }
132
133 void loop() {
134
135     HAL_PWR_EnterSLEEPMode (PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
136
137 }
```

### 3.2.3 Interrupt su EXTI1

Di seguito si riporta un esempio dell'utilizzo della **EXTI1** come interruzione esterna, in particolare si utilizza il pin PC1. Nella funzione **init\_pin1C()** si predispongono i registri **RCC\_AHB1ENR** e **SYSCFG\_EXTICR1** in modo tale che GPIOC venga abilitato e che dal pin PC1 venga lanciato ogni volta un'interruption. Si indica, inoltre, che l'interrupt è non mascherato e che esso è dovuto ad un fronte di salita. Infine grazie alle funzioni NVIC si setta il valore di priorità dell'interruzione a 15 e la si abilita.

```

140 /**
141 ****
142 * @file      main.c
143 * @author    Colella Gianni - Guida Ciro - Lombardi Daniele
144 *           Group IV - Sistemi Embedded 2016-2017
145 * @version   V1.0
146 * @date     16-June-2017
147 * @brief    Esempio di utilizzo delle interruzioni BARE metal su board
148 *           STM32 e periferica EXTI1
149 */
150
151 /* Librerie Incluse
152 -----
153 #include "stm32f4xx.h"
154 #include "stm32f4_discovery.h"
155
156 /* Prototipi funzione
157 -----
158 void setup();
159 void loop();
```

```

159 /* Variabili globali
-----*/
160 short int count=0;
161
162 /**
163 * @brief Interrupt Service Routine lanciata da EXTI1.
164 * @note Una volta invocata la routine essa implementa una logica custom,
165 *       in particolare
166 *           schicciato il bottone utente si provvede ad incrementare di 1
167 *           una variabile
168 *           e mostrato il conteggio sui led.
169 * @param None.
170 * @retval None.
171 */
172 void EXTI1_IRQHandler(){
173     count++;
174     GPIOD->ODR=count<<12;
175     if(count==15)
176         count=0;
177     NVIC_ClearPendingIRQ(EXTI1_IRQn);
178     __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_1);
179 }
180
181 int main(void)
182 {
183     setup();
184     for(;;)loop();
185 }
186
187 void setup(){
188     init_pin1C();
189     BSP_LED_Init(LED3);
190     BSP_LED_Init(LED4);
191     BSP_LED_Init(LED5);
192     BSP_LED_Init(LED6);
193 }
194
195 void loop(){}
196
197 /**
198 * @brief Inizializzazione del pin PC1.
199 * @note PC1 è settato in modo tale da lanciare interruzioni
200 * @param None.
201 * @retval None.
202 */
203 void init_pin1C(){

```

## CAPITOLO 3. ESEMPI DI UTILIZZO DELLE INTERRUZIONI SU BOARD STM

```
205
206     /*Abilitazione GPIOC*/
207     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
208
209     /*Si indica che da GPIOC, PC1, arriva un interrupt*/
210     SYSCFG->EXTICR[0] &= 0xFF0F;
211     SYSCFG->EXTICR[0] = SYSCFG_EXTICR1_EXTI1_PC;
212
213     /*Interupt da GPIOC non mascherato*/
214     EXTI->IMR = (1<<1);
215
216     /*Interrupt su fronte di salita*/
217     EXTI->RTSR=(1<<1);
218
219     /*Settaggio priorità su NVIC*/
220     NVIC_SetPriority(EXTI1 IRQn,15);
221
222     /*Abilitazione interrupt su NVIC*/
223     NVIC_EnableIRQ(EXTI1 IRQn);
224 }
```

# Capitolo 4

## Bus Seriali

### 4.1 Traccia

Si realizzino degli esempi di trasmissione e ricezione di messaggi tra 2 board STM32 Discovery, utilizzando i seguenti bus di comunicazione seriale:

1. SPI;
2. I2C;
3. UART.

### 4.2 Procedimento

#### 4.2.1 SPI

Si propone, di seguito, un esempio di comunicazione che utilizza il bus **Serial Peripheral Interface, SPI**. Inizialmente, con l'utilizzo del software **STM32CubeMX**, si configurano 2 board STM32F4Discovery: una, la si abilita a funzionare da Master; l'altra, da Slave, fig. 4.1.

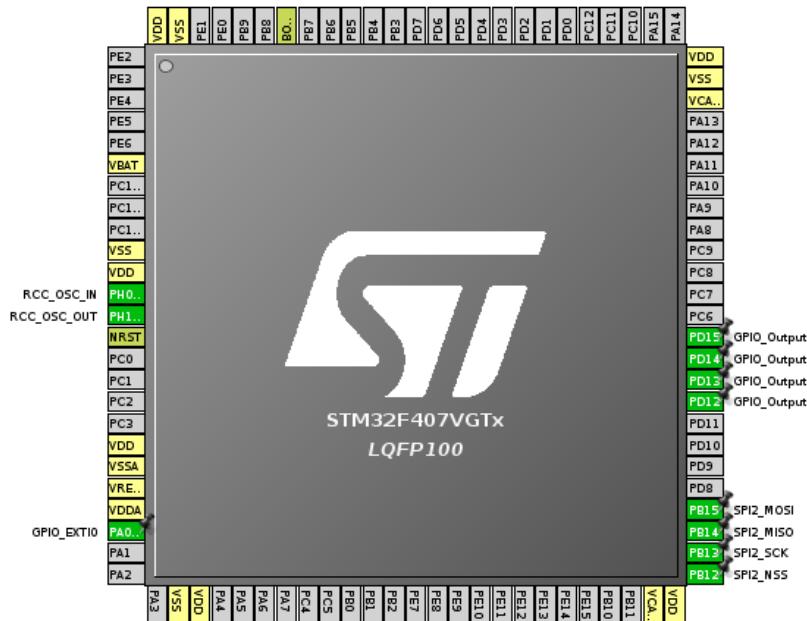


Figura 4.1: Configurazione della STM32F4Discovery per SPI

L'esercitazione, di seguito proposta, prevede che, quando viene premuto il push-button sul master, allora viene eseguita la trasmissione di un messaggio testuale dalla board Master a quella Slave. Se la trasmissione è andata a buon fine, si accende il LED 4 sulla board che funge da Master. Simultaneamente, se la ricezione è eseguita con successo, sullo Slave si accende il LED 6.

Si mette in evidenza che il push button sullo Slave è configurato con interrupt abilitata e con un valore di sub-priority pari a 2. Anche il bus SPI2 ha le interrupt abilitate, ma ad un valore di sub-priority pari ad 1. Quando si aziona l'interruzione su EXTI0, è richiama la callback corrispondente in cui viene effettuata la trasmissione in no-blocking mode dal Master allo Slave. In fig. 4.2 si rappresenta il funzionamento dell'esercitazione appena descritta.

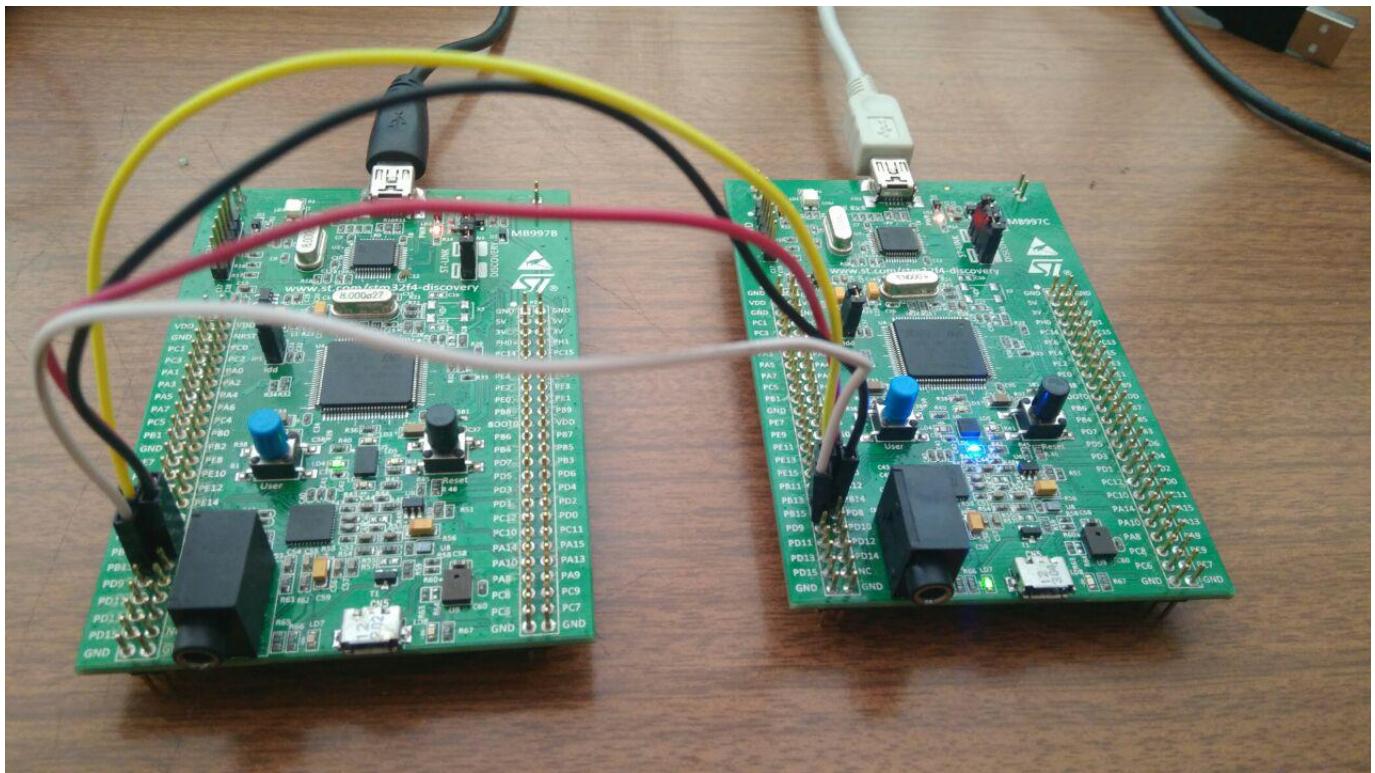


Figura 4.2: Esempio di funzionamento di trasmissione attraverso il bus SPI

## SPI Master

Di seguito, si riporta il codice implementato per la configurazione di SPI e la ridefinizione delle funzioni `_weak` callback.

```

75 /**
76 * @brief Funzione in cui si settano i parametri SPI
77 * @note
78 * @retval None
79 *      */
80 void setup_SPI(){
81     SpiHandle.Instance          = SPI2;
82     SpiHandle.Init.Mode        = SPI_MODE_MASTER;
83     SpiHandle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
84     SpiHandle.Init.Direction    = SPI_DIRECTION_2LINES;
85     SpiHandle.Init.CLKPhase    = SPI_PHASE_1EDGE;
86     SpiHandle.Init.CLKPolarity = SPI_POLARITY_HIGH;
87     SpiHandle.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
88     SpiHandle.Init.CRCPolynomial = 7;
89     SpiHandle.Init.DataSize    = SPI_DATASIZE_8BIT;
90     SpiHandle.Init.FirstBit    = SPI_FIRSTBIT_MSB;
91     SpiHandle.Init.NSS         = SPI_NSS_SOFT;
92     SpiHandle.Init.TIMode     = SPI_TIMODE_DISABLE;
93
94 }
```

```

95     if(HAL_SPI_Init(&SpiHandle) != HAL_OK)
96     {
97         /* Initialization Error */
98         Error_Handler();
99     }
100 }
```

Codice 4.1: "funzione setup SPI.c"

```

223 /**
224 * @brief Funzione Callback richiamata quando il trasferimento è
225 *        completato
226 * @param hspi: gestore spi
227 * @note Si esegue una operazione di Toggle sul PD12(LED4)
228 * @retval None
229 * */
230 void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi){
231     if(hspi->Instance==SpiHandle.Instance){
232         HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_12);
233         SPI2_NSS_DISABLE;
234     }
235 }
236 /**
237 * @brief Funzione Callback richiamata quando c'è stato un'errore durante
238 *        il trasferimento
239 * @param hspi: gestore spi
240 * @note Si esegue un'operazione di Toggle sul PD15(LED6)
241 * @retval None
242 * */
243 void HAL_SPI_ErrorCallback(SPI_HandleTypeDef *hspi)
244 {
245     /* Turn LED5 on: Transfer error in reception/transmission process */
246     HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_15);
247 }
248 /**
249 * @brief Questa funzione è eseguita quando scatta una interruzione di
250 *        EXTI0,
251 *        ovvero quando viene premuto il push button
252 * @param GPIO_Pin: si indica il pin associato al push button.
253 * @note si esegue la transmit.
254 * @retval None
255 * */
256 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
257     //si attende che si rilascia il bottone
258     while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0));
```

```

260 //si abilita il pin not slave select
261 SPI2_NSS_ENABLE;
262 while(HAL_SPI_Transmit_IT(&SpiHandle, (uint8_t *)aTxBuffer, sizeof(
263     aTxBuffer)) != HAL_OK)
264     while (HAL_SPI_GetState(&SpiHandle) != HAL_SPI_STATE_READY);
}

```

Codice 4.2: "Callback master"

## SPI Slave

Di seguito, invece, si riporta il codice per la ricezione del messaggio; la funzione per il settaggio dei parametri utili a configurare il protocollo SPI da Slave; l'implementazione della ridefinizione delle funzioni `_weak` callback.

```

41 int main(void)
42 {
43
44     HAL_Init();
45     setup_LED();
46     setup_SPI();
47
48     SystemClock_Config();
49     /* Infinite loop */
50     while (1)
51     {
52         if(HAL_SPI_Receive_IT(&SpiHandle, (uint8_t *)receive_data, SIZE) !=
53             HAL_OK)
54         {
55             Error_Handler();
56         }
57         while (HAL_SPI_GetState(&SpiHandle) != HAL_SPI_STATE_READY);
58     }
59 }

```

Codice 4.3: "main slave"

```

87 /**
88 * @brief Funzione in cui si settano i parametri SPI
89 * @note Si nota che la modalità è SLAVE
90 * @retval None
91 *
92 */
93 void setup_SPI()
94 {
95     SpiHandle.Instance          = SPI2;
96     SpiHandle.Init.Mode        = SPI_MODE_SLAVE;
97     SpiHandle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
}

```

```

98     SpiHandle.Init.Direction      = SPI_DIRECTION_2LINES;
99     SpiHandle.Init.CLKPhase       = SPI_PHASE_1EDGE;
100    SpiHandle.Init.CLKPolarity    = SPI_POLARITY_HIGH;
101    SpiHandle.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
102    SpiHandle.Init.CRCPolynomial   = 7;
103    SpiHandle.Init.DataSize        = SPI_DATASIZE_8BIT;
104    SpiHandle.Init.FirstBit         = SPI_FIRSTBIT_MSB;
105    SpiHandle.Init.NSS             = SPI_NSS_SOFT;
106    SpiHandle.Init.TIMode          = SPI_TIMODE_DISABLE;
107
108    if(HAL_SPI_Init(&SpiHandle) != HAL_OK)
109    {
110        /* Initialization Error */
111        Error_Handler();
112    }
113}
114
115 /**
116 * @brief Funzione Callback richiamata quando la ricezione è completata
117 * @param [in] hspi: gestore spi
118 * @note Si esegue un'operazione di Toggle sul PD15(LED6)
119 * @retval None
120 * */
121 void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi){
122     if(hspi->Instance==SpiHandle.Instance)
123         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_15);
124 }
125
126 /**
127 * @brief Funzione Callback richiamata quando c'è stato un'errore durante
128 * il trasferimento
129 * @param hspi: gestore spi
130 * @note Si esegue un'operazione di Toggle sul PD14(LED5)
131 * @retval None
132 * */
133 void HAL_SPI_ErrorCallback(SPI_HandleTypeDef *hspi){
134     if(hspi->Instance==SpiHandle.Instance)
135         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_15);
136 }
137
138 /**
139 * @brief Questa funzione è eseguita in caso di errore.
140 * @note In tal caso si accende il led rosso: LED5
141 * @retval None
142 * */
143 static void Error_Handler(void){
144     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
145 }

```

Codice 4.4: "funzioni slave"

#### 4.2.2 I2C

Si propone, di seguito, un esempio di trasmissione di un messaggio tra un Master e uno Slave attraverso il bus **Inter Integrated Circuit**, **I2C** o  $I^2C$ .

Inizialmente con l'utilizzo del software STM32CubeMX si configurano 2 board STM32F3Discovery: una, la si abilita a funzionare da Master; l'altra, da Slave, fig. 4.3.

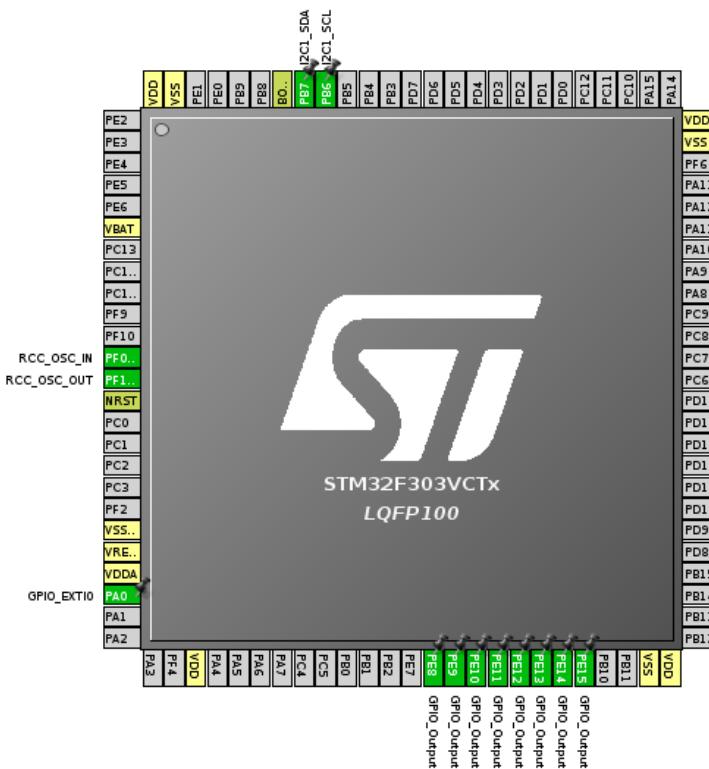


Figura 4.3: Configurazione della STM32F4Discovery per I2C

Si ripropone un esempio di funzionamento speculare a quello visto in 4.2.1, in cui è stato utilizzato il bus SPI. A differenza del caso precedente in cui il Master e lo Slave sono connessi attraverso 4 canali e il canale NSS è utilizzato per individuare lo Slave desiderato, in I2C ogni periferica è settata con un proprio indirizzo su 7 o 10 bit. Il Master, per poter comunicare con uno specifico Slave, deve inviare sul canale SDA prima l'indirizzo dello slave con cui vuole comunicare e successivamente il dato da trasmettere. Nell'esempio realizzato, si utilizza un'indirizzamento a 7 bit. In fig. 4.4 si rappresenta il funzionamento dell'esempio riportato.

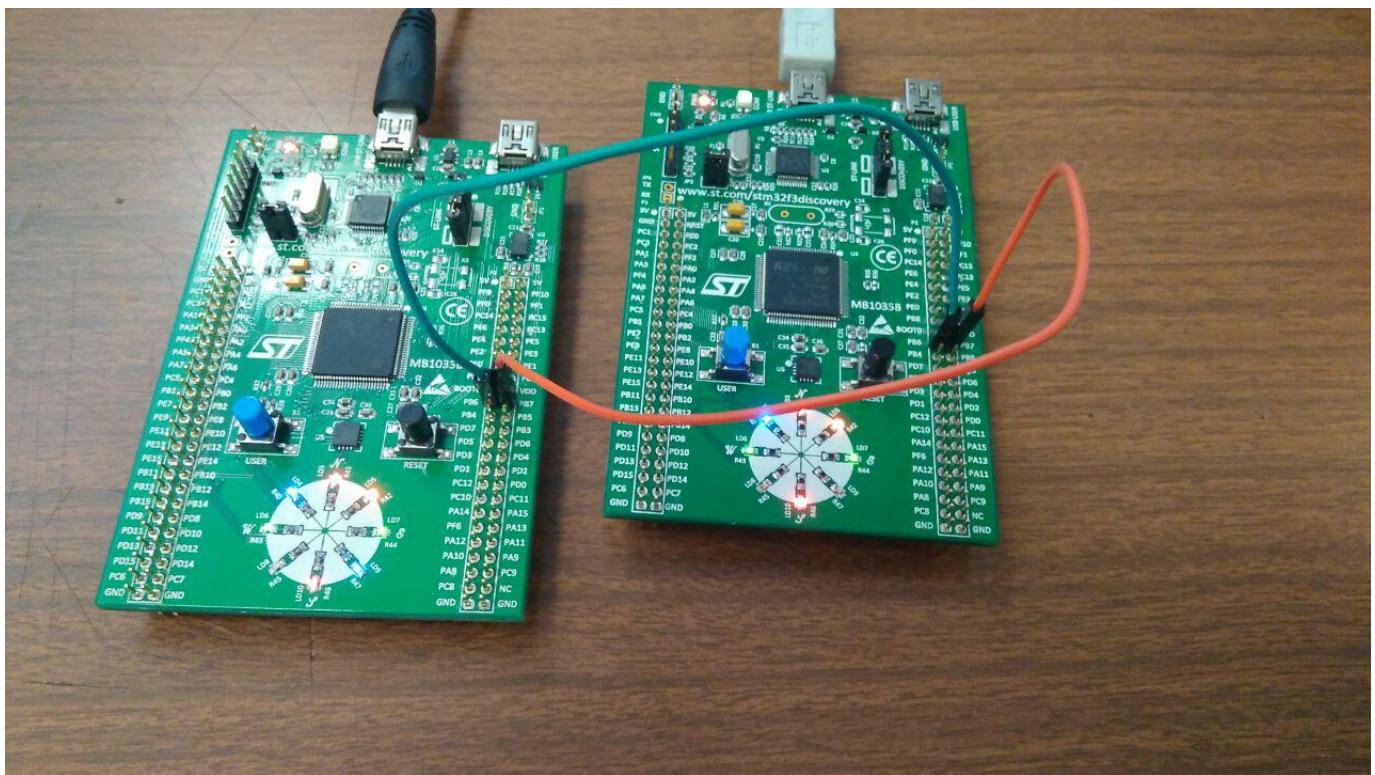


Figura 4.4: Esempio di funzionamento di trasmissione attraverso il bus I2C

## I2C Master

Si riportano le funzioni principali utilizzate per la trasmissione dal Master allo Slave.

```

146  * @brief Configurazione I2C
147  * @note utilizzando un valore di Timing pari a 004C4092A , partendo da un
148    clock a 72MHz, si ha un timing a 1MHz
149  * @param None
150  * @retval None
151  */
152 static void MX_I2C1_Init(void)
153 {
154     hi2c1.Instance          = I2C1;
155     hi2c1.Init.Timing       = 0x00C4092A;
156     hi2c1.Init.OwnAddress1  = ADDRESS_A;
157     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
158     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
159     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
160     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
161     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
162     {
163         _Error_Handler(__FILE__, __LINE__);
164     }
165     /*Configure Analogue filter */

```

```

166     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) !=  

167         HAL_OK)  

168     {  

169         _Error_Handler(__FILE__, __LINE__);  

170     }  

171     /***Configure Digital filter      */  

172     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)  

173     {  

174         _Error_Handler(__FILE__, __LINE__);  

175     }
176 }
```

Codice 4.5: "Funzione MX I2C1 master.c"

```

176  /*  

177   * @brief Funzione in cui si configurano i GPIO.  

178   * @param None  

179   * @retval None  

180 */  

181 static void MX_GPIO_Init(void)  

182 {  

183  

184     GPIO_InitTypeDef GPIO_InitStruct;  

185  

186     /* GPIO Ports Clock Enable */  

187     __HAL_RCC_GPIOF_CLK_ENABLE();  

188     __HAL_RCC_GPIOA_CLK_ENABLE();  

189     __HAL_RCC_GPIOE_CLK_ENABLE();  

190     __HAL_RCC_GPIOB_CLK_ENABLE();  

191  

192     /*Configure GPIO pin Output Level */  

193     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11  

194             |GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,  

195             GPIO_PIN_RESET);  

196  

197     /*Configure GPIO pin : PA0 */  

198     GPIO_InitStruct.Pin = GPIO_PIN_0;  

199     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;  

200     GPIO_InitStruct.Pull = GPIO_NOPULL;  

201     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);  

202  

203     /*Configure GPIO pins : PE8 PE9 PE10 PE11  

204             PE12 PE13 PE14 PE15 */  

205     GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11  

206             |GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;  

207     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;  

208     GPIO_InitStruct.Pull = GPIO_NOPULL;  

209     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```

209     HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
210
211     /* EXTI interrupt init*/
212     HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 3);
213     HAL_NVIC_EnableIRQ(EXTI0_IRQn);
214
215 }
216
217 /* USER CODE BEGIN 4 */
218 /**
219  * @brief Questa funzione è eseguita quando scatta l'interruzione di EXTI0,
220  * ovvero quando viene premuto il push button
221  * @param GPIO_Pin: si indica il pin associato al push button.
222  * @note si esegue la Master Transmit in no blocking mode verso lo slave
223  *      di indirizzo ADDRESS_B.
224  * @retval None
225  * */
226
227 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
228     //si attende che si rilascia il bottone
229     while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)!=0);
230     //si abilita il pin not slave select
231     HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|
232                     GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,GPIO_PIN_RESET);
233
234     while(HAL_I2C_Master_Transmit_IT(&hi2c1, (uint16_t)ADDRESS_B, (uint8_t*)send_data, TXSIZE) != HAL_OK)
235         while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
236
237 /**
238  * @brief Funzione Callback richiamata quando il Master completa il
239  * trasferimento verso lo slave
240  * @param hspi: gestore spi
241  * @note Si accendono tutti i LED della board
242  * @retval None
243  * */
244 void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef *hi2c) {
245     if(hi2c->Instance==hi2c1.Instance){
246         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8,GPIO_PIN_SET);
247         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_9,GPIO_PIN_SET);
248         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_10,GPIO_PIN_SET);
249         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_11,GPIO_PIN_SET);
250         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_12,GPIO_PIN_SET);
251         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_13,GPIO_PIN_SET);
252         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_14,GPIO_PIN_SET);
253         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_15,GPIO_PIN_SET);
254     }

```

```

254 }
255 }
256 }
257 /**
258 * @brief Funzione Callback richiamata quando c'è stato un'errore durante
259 * il trasferimento
260 * @param hspi: gestore spi
261 * @note Sulla board saranno accesi solo i LED 6 e 7, configurati nei pin
262 * PE15 e PE11, tutti gli altri saranno spenti
263 * @retval None
264 */
265 void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)
266 {
267     if(hi2c->Instance==hi2c1.Instance)
268     {
269         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|
270                         GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
271                         GPIO_PIN_RESET);
272         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_15,GPIO_PIN_SET);
273         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_11,GPIO_PIN_SET);
274         while(1);
275     }
276 }
277 /**
278 * @brief This function is executed in case of error occurrence.
279 * @param None
280 * @retval None
281 */
282 void _Error_Handler(char * file, int line)
283 {
284     /* USER CODE BEGIN Error_Handler_Debug */
285     /* User can add his own implementation to report the HAL error return
286      state */
287     while(1)
288     {
289         HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_11);
290         HAL_Delay(200);
291     }
292 }

```

Codice 4.6: "Funzione MX I2C1 master.c"

## I2C Slave

Di seguito, invece, si riportano le funzioni utilizzate dallo Slave.

```

1 /*
*****
```

```

2  * @name      : main.c
3  * @authors   : Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
4    Sistemi Embedded 2016-2017
5  * @date      : 13-June-2017
6  * @brief     : Main program body
7  ****
8
9  * In questo programma è configurato il main di una ricezione I2C.
10 * Prima di eseguire un loop infinito si attende di ricevere un messaggio
11 * attraverso il bus I2C
12 * dallo slave.
13 *
14 *      I2C2 SLAVE configuration
15 *      PB6 (I2C1_SDA) -----> (MASTER_SDA)
16 *      PB7 (I2C1_SCL) -----> (MASTER_SCL)
17 *
18 */
19
20
21 /*Definizione di due macro per stabilire gli indirizzi di master(a) e slave(b)*/
22 #define ADDRESS_A      0x18
23 #define ADDRESS_B      0x11
24
25 /*definizione del gestore del bus*/
26 I2C_HandleTypeDef hi2c1;
27
28 /*prototipi delle funzioni*/
29
30 void SystemClock_Config(void);
31 static void MX_GPIO_Init(void);
32 static void MX_I2C1_Init(void);
33 void loop(void);
34
35 /*ridefinizione dei prototipi per le callback di EXTI0 e I2CTxClp*/
36 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
37 void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef *hi2c);
38
39 /*definizione e inizializzazione di un baffer su cui ricevere il messaggio*/
40 uint8_t send_data[] = "****MESSAGGIO I2C****";
41 uint8_t receive_data[TXSIZE];
42
43 /**
44  * @brief Funzione main in cui vengono inizialmente effettuati i settaggi
45  *        di HALL, Clock, GPIO e bus

```

```

45     *      e successivamente si attende di ricevere un messaggio dal master
46     * @param  None
47     * @retval None
48     */
49 int main(void)
50 {
51
52     HAL_Init();
53
54     SystemClock_Config();
55
56     MX_GPIO_Init();
57     MX_I2C1_Init();
58     if(HAL_I2C_Slave_Receive_IT(&hi2c1, (uint8_t *)receive_data, sizeof(
59         receive_data)) != HAL_OK)
60     {
61         Error_Handler();
62     }
63     while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
64
65     while(1)
66     {
67         loop();
68     }
69 }
70 /**
71 * @brief Loop infinito richiamato dalla funzione main.c.
72 * @param  None
73 * @retval None
74 */
75 void loop(void){
76
77     HAL_Delay(150);
78     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_15);
79     HAL_Delay(150);
80     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_9);
81     HAL_Delay(150);
82     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_11);
83     HAL_Delay(150);
84     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_13);
85     HAL_Delay(150);
86     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_8);
87     HAL_Delay(150);
88     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_10);
89     HAL_Delay(150);
90     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_12);
91     HAL_Delay(150);
92     HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_14);

```

```

93     }
94 /**
95  * @brief System Clock Configuration
96  *        The system Clock is configured as follow :
97  *          System Clock source      = PLL (HSE)
98  *          SYSCLK(Hz)              = 72000000
99  *          HCLK(Hz)                = 72000000
100 *          AHB Prescaler          = 1
101 *          APB1 Prescaler        = 2
102 *          APB2 Prescaler        = 1
103 *          HSE Frequency(Hz)     = 8000000
104 *          HSE PREDIV            = 1
105 *          PLLMUL                = RCC_PLL_MUL9 (9)
106 *          Flash Latency(WS)     = 2
107 * @param None
108 * @retval None
109 */
110 void SystemClock_Config(void)
111 {
112
113     RCC_OscInitTypeDef RCC_OscInitStruct;
114     RCC_ClkInitTypeDef RCC_ClkInitStruct;
115     RCC_PeriphCLKInitTypeDef PeriphClkInit;
116
117     /**Initializes the CPU, AHB and APB busses clocks
118     */
119     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI |
120         RCC_OSCILLATORTYPE_HSE;
121     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
122     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
123     RCC_OscInitStruct.HSISState = RCC_HSI_ON;
124     RCC_OscInitStruct.HSICalibrationValue = 16;
125     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
126     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
127     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
128     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
129     {
130         _Error_Handler(__FILE__, __LINE__);
131     }
132
133     /**Initializes the CPU, AHB and APB busses clocks
134     */
135     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
136                         |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
137     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
138     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
139     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
140     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

```

```

141 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
142 {
143     _Error_Handler(__FILE__, __LINE__);
144 }
145
146 PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
147 PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
148 if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
149 {
150     _Error_Handler(__FILE__, __LINE__);
151 }
152
153     /**Configure the Systick interrupt time
154 */
155 HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
156
157     /**Configure the Systick
158 */
159 HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
160
161 /* SysTick_IRQn interrupt configuration */
162 HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
163 }

164
165 /* I2C1 init function */
166 /**
167 * @brief Configurazione I2C
168 * @note utilizzando un valore di Timing pari a 004C4092A , partendo da un
169 *       clock a 72MHz, si ha un timing a 1MHz
170 * @param None
171 * @retval None
172 */
173 static void MX_I2C1_Init(void)
174 {
175     hi2c1.Instance      = I2C1;
176     hi2c1.Init.Timing    = 0x00C4092A;
177     hi2c1.Init.OwnAddress1 = ADDRESS_B;
178     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
179     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
180     hi2c1.Init.OwnAddress2 = 0xFF;
181     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
182     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
183     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
184     {
185         _Error_Handler(__FILE__, __LINE__);
186     }
187
188     /**Configure Analogue filter
189 */

```

```

189     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) !=
190         HAL_OK)
191     {
192         _Error_Handler(__FILE__, __LINE__);
193     }
194
195     /* Configure Digital filter
196     */
197     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
198     {
199         _Error_Handler(__FILE__, __LINE__);
200     }
201
202 /**
203  * Analog
204  * Input
205  * Output
206  * EVENT_OUT
207  * EXTI
208 */
209 static void MX_GPIO_Init(void)
210 {
211
212     GPIO_InitTypeDef GPIO_InitStruct;
213
214     /* GPIO Ports Clock Enable */
215     __HAL_RCC_GPIOF_CLK_ENABLE();
216     __HAL_RCC_GPIOA_CLK_ENABLE();
217     __HAL_RCC_GPIOE_CLK_ENABLE();
218
219     /*Configure GPIO pin Output Level */
220     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
221                           |GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
222                           GPIO_PIN_RESET);
223
224     /*Configure GPIO pin : PA0 */
225     GPIO_InitStruct.Pin = GPIO_PIN_0;
226     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
227     GPIO_InitStruct.Pull = GPIO_NOPULL;
228     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
229
230     /*Configure GPIO pins : PE8 PE9 PE10 PE11
231                           PE12 PE13 PE14 PE15 */
232     GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
233                           |GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
234     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
235     GPIO_InitStruct.Pull = GPIO_NOPULL;
236     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

```

```

236     HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
237
238     /* EXTI interrupt init*/
239     HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 2);
240     HAL_NVIC_EnableIRQ(EXTI0_IRQn);
241
242 }
243
244 /* USER CODE BEGIN 4 */
245
246 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
247     //si attende che si rilascia il bottone
248     while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)!=0);
249     //si abilita il pin not slave select
250     HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|
251                     GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,GPIO_PIN_RESET);
252 }
253
254 /**
255 * @brief Funzione Callback richiamata quando lo slave completa il
256 *        trasferimento verso lo slave
257 * @param hspi: gestore spi
258 * @note Si accendono tutti i LED della board
259 * @retval None
260 */
261 void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef *hi2c) {
262     if(hi2c->Instance==hi2c1.Instance){
263         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|
264                     GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,GPIO_PIN_SET);
265     }
266 }
267
268 /**
269 * @brief Funzione Callback richiamata quando c'è stato un'errore
270 *        durante il trasferimento
271 * @param hspi: gestore spi
272 * @note Sulla board saranno accesi solo i LED 6 e 7, configurati nei pin
273 *       PE15 e PE11, tutti gli altri saranno spenti
274 * @retval None
275 */
276 void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c){
277     if(hi2c->Instance==hi2c1.Instance)
278     {
279         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|
280                         GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
281                         GPIO_PIN_RESET);
282         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_15,GPIO_PIN_SET);
283         HAL_GPIO_WritePin(GPIOE,GPIO_PIN_11,GPIO_PIN_SET);
284         while(1);
285     }

```

```

278 }
279 /**
280 * @brief This function is executed in case of error occurrence.
281 * @param None
282 * @retval None
283 */
284 void _Error_Handler(char * file, int line)
285 {
286     /* USER CODE BEGIN Error_Handler_Debug */
287     /* User can add his own implementation to report the HAL error return
288      state */
289     while(1)
290     {
291         HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_11);
292         HAL_Delay(200);
293     }
294     /* USER CODE END Error_Handler_Debug */
295 }
296
297 #ifdef USE_FULL_ASSERT
298
299 /**
300 * @brief Reports the name of the source file and the source line number
301 * where the assert_param error has occurred.
302 * @param file: pointer to the source file name
303 * @param line: assert_param error line source number
304 * @retval None
305 */
306 void assert_failed(uint8_t* file, uint32_t line)
307 {
308     /* USER CODE BEGIN 6 */
309     /* User can add his own implementation to report the file name and line
310      number,
311      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
312      */
313     /* USER CODE END 6 */
314 }
315 #endif

```

Codice 4.7: "main slave.c"

### 4.2.3 UART

Di seguito, si propone un esempio di funzionamento di una trasmissione attraverso l'utilizzo del bus **Universal Asynchronous Receiver-Transmitter**, **UART**. Inizialmente con l'utilizzo del soft-

ware STM32CubeMX si configurano 2 board STM32F4Discovery:una, la si abilita a funzionare da Master; l'altra, da Slave, fig. 4.5.

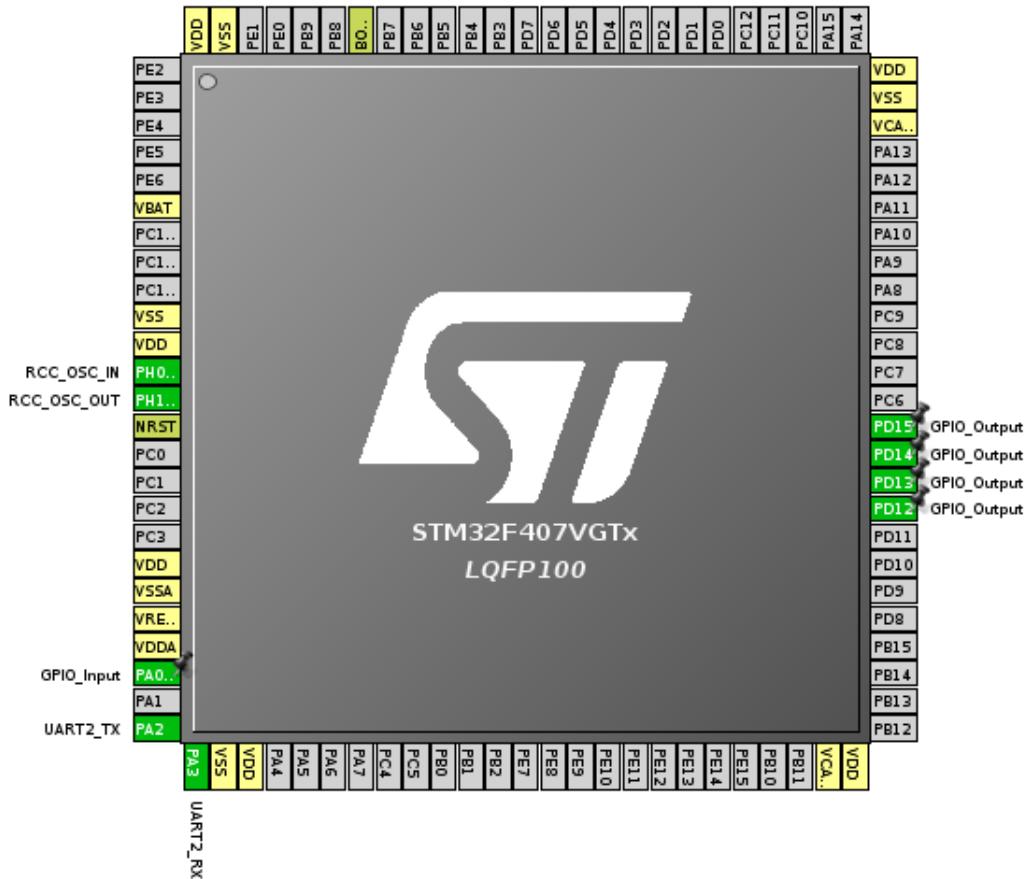


Figura 4.5: Configurazione della STM32F4Discovery per l'uso di UART

L'esercitazione prevede che, quando si preme il push-button sul Master, inizia la trasmissione di un messaggio in cui è presente un valore numerico, inizialmente 0. Se la ricezione va a buon fine, la board che funge da Slave accende il led corrispondente al valore numerico e prima di reinviarla al master ne incrementa il valore di 1 modulo 4. In questo modo il contenuto del messaggio appartiene constantemente all'intervallo [0,3]. Quando il Master riceve il messaggio dallo Slave, anch'esso accende il led corrispondente al valore ricevuto e si pone in attesa di una successiva trasmissione. In fig. 4.6 si rappresenta il funzionamento dell'esercitazione descritta.

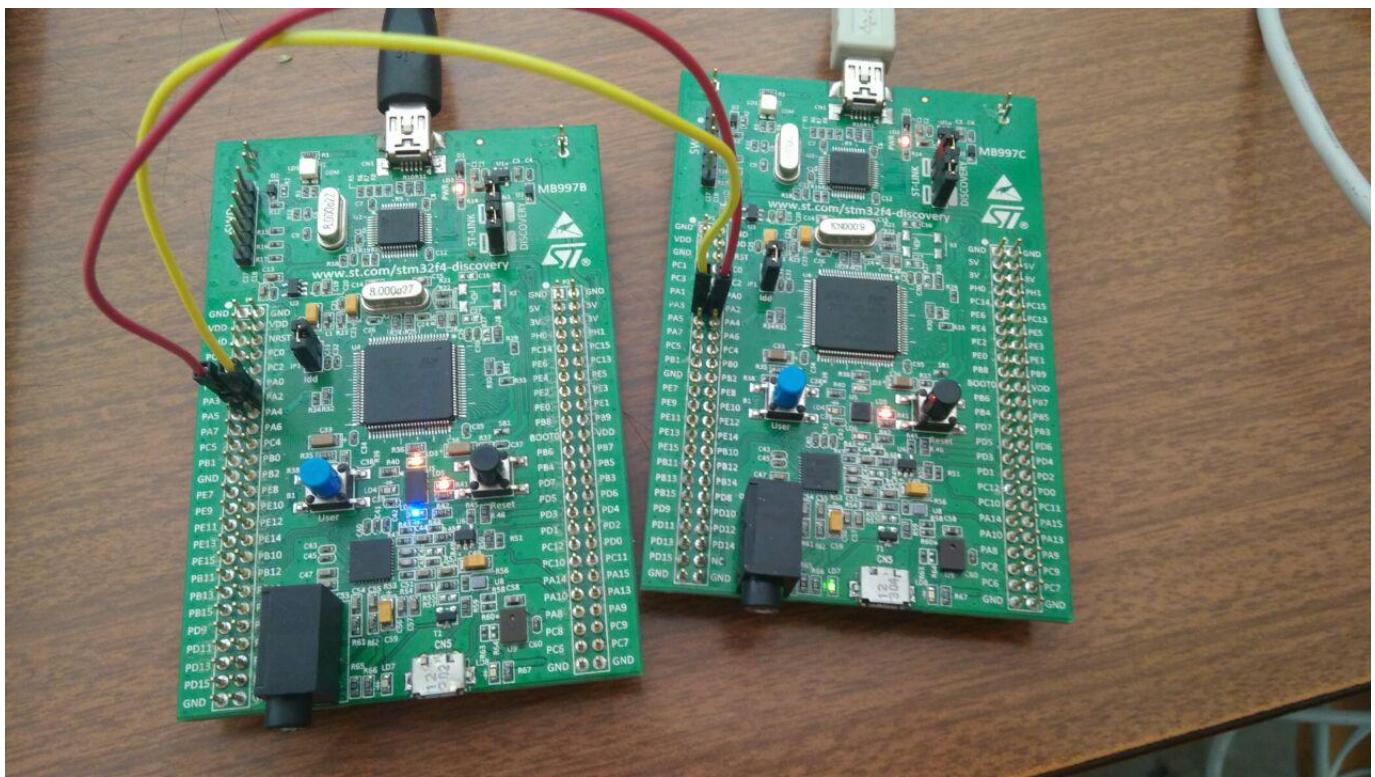


Figura 4.6: Esempio di funzionamento di trasmissione attraverso il bus UART

### UART Master

Di seguito, si riporta il codice della funzione **SetupUART** in cui si esegue il settaggio della configurazione UART e della funzione loop, eseguita in un ciclo infinito nel main del Master.

```

189 /**
190 * @brief Configurazione UART
191 * Si utilizza una USART Asincrona, nello specifico si sceglie
192 * USART2
193 *
194 * Si sceglie di settare i seguenti valori :
195 * - Lunghezza = 8 Bits
196 * - Stop Bit = 1 bit
197 * - Parity = None
198 * - BaudRate = 9600 baud
199 * - Hardware flow control disabled (RTS and CTS signals)
200 * @param None
201 * @retval None
202 */
203 static void SetupUART(void) {
204     huart2.Instance      = USART2;
205
206     huart2.Init.BaudRate    = 9600;
207     huart2.Init.WordLength  = UART_WORDLENGTH_8B;

```

```

208     huart2.Init.StopBits      = UART_STOPBITS_1;
209     huart2.Init.Parity       = UART_PARITY_NONE;
210     huart2.Init.HwFlowCtl   = UART_HWCONTROL_NONE;
211     huart2.Init.Mode        = UART_MODE_TX_RX;
212     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
213
214     if(HAL_UART_Init(&huart2) != HAL_OK)
215     {
216         Error_Handler();
217     }
218
219 }
220
221 /**
222 * @brief Loop per la trasmissione/ricezione di un messaggio attraverso il
223 *        bus UART.
224 * @param None
225 * @retval None
226 */
227 void loop(void){
228
229     /* Si attende che sia premuto il push button */
230     while (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) != GPIO_PIN_SET) {
231         HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
232         HAL_Delay(40);
233     }
234     /* Si attende che il push button sia rilasciato */
235     while (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) != GPIO_PIN_RESET);
236
237     /* Si spegne il led3 */
238     HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13,GPIO_PIN_RESET);
239
240     /* Si trasmette il messaggio da inviare (send_data) con un timeout di 5
241      secondi*/
242
243     if(HAL_UART_Transmit(&huart2, (uint8_t*)send_data, TXSIZE, 5000) != HAL_OK)
244     {
245         Error_Handler();
246     }
247
248     /* Si accende il LED6 quando il trasferimento è completato */
249     HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_SET);
250
251     /* Si pone il master in polling per una ricezione*/
252     if(HAL_UART_Receive(&huart2, (uint8_t *)receive_data, RXSIZE, 5000) != HAL_OK)
253     {
254         Error_Handler();
255     }
256     /* Si accende il led associato a arxBuffer quando la ricezione è stata

```

```

    completata */
253 HAL_GPIO_WritePin(GPIOD, LEDS[send_data[0]], GPIO_PIN_RESET);
254 send_data[0] = receive_data[0];
255 HAL_GPIO_WritePin(GPIOD, LEDS[send_data[0]], GPIO_PIN_SET);
256 }

```

Codice 4.8: "main master.c"

## UART Slave

```

/***
 * @brief Loop per la trasmissione/ricezione di un messaggio attraverso il
 *        bus UART.
 * @param None
 * @retval None
 */
void loop(void){
    /*Si è in polling di ricevere il messaggio dal master*/
    if(HAL_UART_Receive(&huart2, (uint8_t *)receive_data, RXSIZE, 5000) !=
       HAL_OK)
    {
        Error_Handler();
    }
    HAL_GPIO_WritePin(GPIOD, LEDS[send_data[0]], GPIO_PIN_RESET);
    /*Si incrementa di 1 modulo 4 il contenuto del messaggio ricevuto*/
    send_data[0] = (receive_data[0]+1)%4;
    HAL_GPIO_WritePin(GPIOD, LEDS[send_data[0]], GPIO_PIN_SET);

    /*Si invia al master il valore appena aggiornato*/
    if(HAL_UART_Transmit(&huart2, (uint8_t *)send_data, TXSIZE, 5000) !=
       HAL_OK)
    {
        Error_Handler();
    }
}

```

Codice 4.9: "main slave.c"

# Capitolo 5

## Esempi di utilizzo dell'Universal Serial Bus

### 5.1 Traccia

Si mostri come è possibile inviare dati serialmente, da una board STM Discovery ad un terminale virtuale, utilizzando un collegamento USB. Successivamente, si implementi un device o host USB.

### 5.2 Procedimento

#### 5.2.1 Invio dati seriali

Un semplice esempio di invio dati tramite USB può essere implementato mediante l'utilizzo dell'interfaccia **Communication Device Class, CDC**, simulando così un trasferimento **UART**.

Innanzitutto è necessario creare un progetto utilizzando, come già visto in 4, il tool STM32 CubeMX. La seguente figura mostra le impostazioni iniziali.

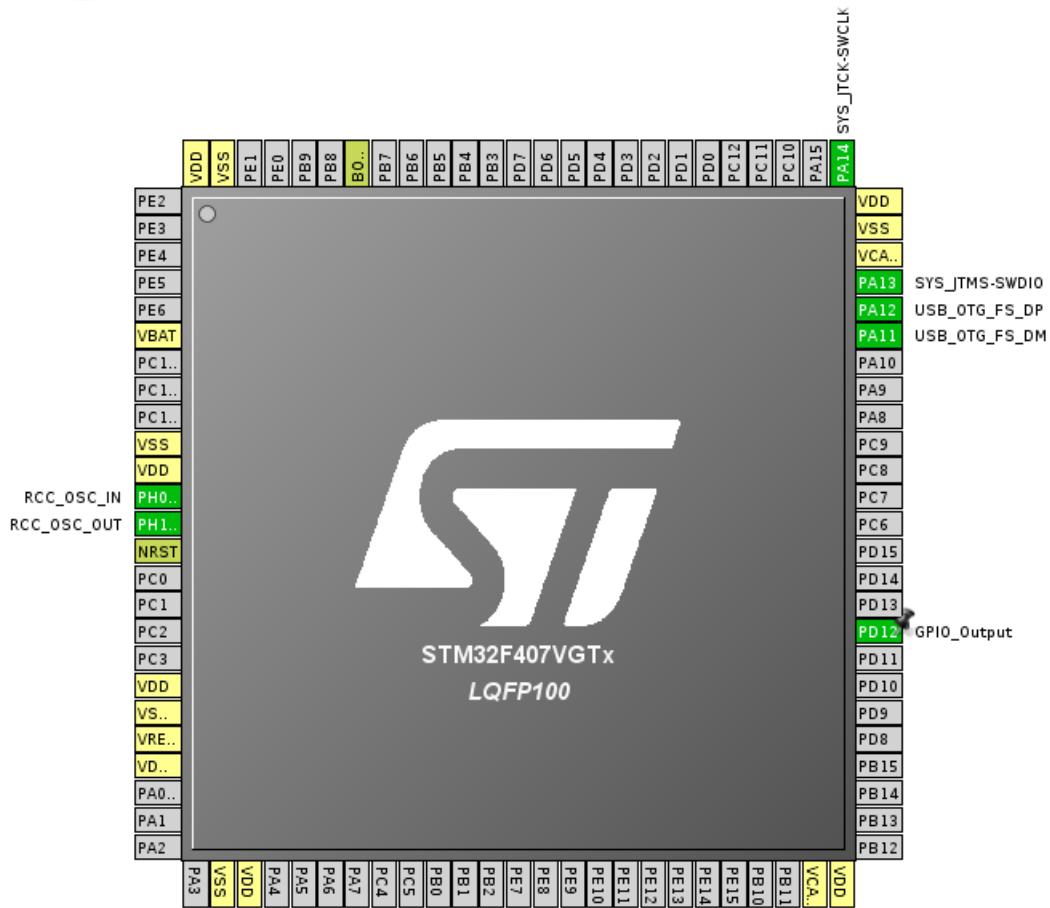


Figura 5.1: Impostazioni iniziali

La parte più interessante delle impostazioni è proprio quella riguardante USB. A tale scopo, nella scheda di *Pinout*, è necessario attivare la periferica USB come OTG ed impostarla come device, infine, bisogna indicare l'utilizzo del middleware USB CDC per l'implementazione di una porta COM virtuale.

Una volta generato il progetto con le relative configurazioni iniziali, si può passare all'implementazione di un semplice applicativo. Si importa, dunque, la libreria per la comunicazione tramite la classe CDC (`usbd_cdc_if.h`) e si aggiungono al main le seguenti righe di codice.

```

107
108     uint8_t car=0;
109     char space=' ';
110
111     while (1)
112     {
113         HAL_Delay(500);
114         CDC_Transmit_FS(&car, 1);
115         CDC_Transmit_FS((uint8_t*)space, sizeof(space));
116         car++;
117         if(car==127)
118             car=0;
    }

```

Codice 5.1: main.c

Mediante la funzione **CDC\_Transmit\_FS** si inviano serialmente 8 bit per volta, in particolare il frammento di codice appena visto è relativo alla stampa dei caratteri ASCII da 0 a 127.

Infine, tramite un terminale seriale, è possibile visualizzare a video la stampa di ogni singolo carattere.

```
• GtkTerm - /dev/ttyACM0 9600-8-N-1 (come superutente)
File Edit Log Configuration Controls Signals View
! "#$%& ' () *+, -./0123456789: ;<=>?@ABCDEFGHIJKLMNPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~[=][=][=][=]
```

Figura 5.2: Esempio di esecuzione

### 5.2.2 Device USB

Sulla falsa riga dell'esercizio precedente, per implementare un device USB tramite una board STM Discovery, è necessario seguire lo stesso procedimento, stavolta, tuttavia, il middleware da impostare è quello relativo alle periferiche **HID** (Human Interface Device). In particolare, di seguito si propone l'implementazione di un mouse, sfruttando il chip MEMS in dotazione su STM32F4 Discovery. Questo offre la possibilità di usare un accellerometro tramite bus SPI o I2C (nel caso in esame si è scelto di utilizzare SPI). Di seguito le funzioni implementate:

- init\_mouse;
- x\_position;
- y\_position;
- position\_on\_led.

Il seguente codice evidenzia i prototipi delle funzioni appena elencate.

```

1 /**
2 ****
3 * @file      mouse_position.h
4 * @author    Colella Gianni - Guida Ciro - Lombardi Daniele
5 *           Group IV - Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     05-July-2017
8 * @brief    Libreria per gestione puntatore di un mouse
9 ****
10 */
11
12 #ifndef MOUSE_POSITION_H_
13 #define MOUSE_POSITION_H_
14
```

```

15 /* Librerie Incluse
-----*/
16 #include "stm32f4xx_hal.h"
17
18 /**
19  * @brief Definizione tipo enumerativo per valori degli assi X e Y
20  */
21 typedef enum{
22     X,
23     Y
24 }mouse_position_AxisType;
25
26 /* Prototipi funzioni
-----*/
27 void init_mouse(SPI_HandleTypeDef); /*!< Inizializzazione dispositivo*/
28 uint8_t x_position(SPI_HandleTypeDef); /*!< Calcolo posizione su asse X*/
29 uint8_t y_position(SPI_HandleTypeDef); /*!< Calcolo posizione su asse Y*/
30 void position_on_led(uint8_t position,mouse_position_AxisType); /*!<
   Illuminazione led in base al movimento*/
31
32 #endif /* MOUSE_POSITION_H_ */

```

Codice 5.2: mouse position.h

### 5.2.2.1 init\_mouse

La funzione prevede di inizializzare correttamente l'accellerometro. In particolare, una volta selezionata la periferica, così come previsto dal relativo manuale di riferimento, è necessario inviare l'indirizzo del registro di inizializzazione per risvegliare la periferica. Successivamente è necessario indicare in tale registro che le posizioni costantemente da leggere sono quelle relative all'asse delle ascisse e delle ordinate, infine, così come previsto dal normale utilizzo di un bus SPI, si provvede a deselezionare la periferica.

```

15 /**
16  * @brief Questa funzione permette di inizializzare la periferica HID
17  *        mouse
18  *
19  * @note L'inizializzazione avviene mediante l'utilizzo di SPI
20  *
21  * @param hspil: struttura SPI_HandleTypeDef che contiene le informazioni
22  *               di configurazione
23  *               della specifica SPI
24  *
25  * @retval none
26 */
27
28 void init_mouse(SPI_HandleTypeDef hspil){
29
   /*Selezione via software della periferica accellerometro*/

```

```

29     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
30     /*Si invia l'indirizzo 0x20 su bus SPI per risvegliare il relativo
       registro */
31     uint8_t address_Acc=0x20;
32     HAL_SPI_Transmit(&hspil,&address_Acc,1,50);
33     /*Sul registro selezionato si indica che si vuole leggere il valore lungo
       gli assi X e Y*/
34     uint8_t value_Ctrl_Reg1=0x67; /*< Valore da scrivere in Ctrl_Reg1*/
35     HAL_SPI_Transmit(&hspil,&value_Ctrl_Reg1,1,50);
36     /*Deselezione via software della periferica accellerometro*/
37     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
38
39 }
```

Codice 5.3: "mouse position.c"

### 5.2.2.2 x\_position

La funzione restituisce il valore sull'asse X del mouse. In particolare, viene selezionata la periferica; inviato il valore 0x29 che funge da abilitazione al registro che contiene i valori d'interesse (aggiungendo 0x80, come riportato nel manuale di riferimento, si richiede al registro un numero di letture multiple); prelevato il valore dal suddetto registro tramite una funzione di ricezione e, infine, deselectonata la periferica.

```

41 /**
42  * @brief Questa funzione permette di calcolare la posizione sull'asse X.
43  *
44  * @param hspil: struttura SPI_HandleTypeDef che contiene le informazioni
45  *               di configurazione della specifica SPI.
46  *
47  * @retval Intero senza segno ad 8 bit.
48  */
49
50 uint8_t x_position(SPI_HandleTypeDef hspil){
51     uint8_t xval=0;
52     /*Selezione via software della periferica accellerometro*/
53     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
54     /*Si invia l'indirizzo 0x29+0x80 su bus SPI per risvegliare il relativo
       registro */
55     /*Si aggiunge 0x80 per indicare alla periferica che voglio più letture
       dall'asse X */
56     uint8_t address_Acc=0xA9;
57     HAL_SPI_Transmit(&hspil,&address_Acc,1,50);
58     /*Lettura valore sull'asse X*/
59     HAL_SPI_Receive(&hspil,&xval,1,50);
60     /*Deselezione via software della periferica accellerometro*/
61     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
62     return xval;
63 }
```

Codice 5.4: "mouse position.c"

### 5.2.2.3 y\_position

La funzione restituisce il valore sull'asse Y del mouse e la sua implementazione è identica a quella precedente, fatta eccezione ovviamente per il registro in cui leggere il valore desiderato.

```

65 /**
66 * @brief Questa funzione permette di calcolare la posizione sull'asse Y
67 *
68 * @param hspil: struttura SPI_HandleTypeDef che contiene le informazioni
69 *               di configurazione della specifica SPI
70 *
71 * @retval Intero ad 8 bit.
72 */
73
74 uint8_t y_position(SPI_HandleTypeDef hspil) {
75     uint8_t yval;
76     /*Selezione via software della periferica acellerometro*/
77     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
78     /*Si invia l'indirizzo 0x2B+0x80 su bus SPI per risvegliare il relativo
79      registro */
80     /*Si aggiunge 0x80 per indicare alla periferica che voglio più letture
81      dall'asse Y */
82     uint8_t address_Acc=0xAB;
83     HAL_SPI_Transmit(&hspil,&address_Acc,1,50);
84     /*Lettura valore sull'asse Y*/
85     HAL_SPI_Receive(&hspil,&yval,1,50);
86     /*Deselezione via software della periferica acellerometro*/
87     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
88     return yval;
89 }
```

Codice 5.5: "mouse position.c"

### 5.2.2.4 position\_on\_led

Infine, tramite questa funzione è possibile visualizzare la direzione del mouse sui 4 led utente, in particolare i led 4 e 5 sono quelli relativi al movimento lungo le ascisse, 3 e 6 lungo le ordinate.

```

89 /**
90 * @brief Questa funzione in base al movimento della board permette di
91 *        illuminare i led
92 *        utente.
93 * @param [in] position: posizione attuale
94 * @param [in] axis: X per illuminare i led relativi alle ascisse, Y per
95 *                   illuminare quelli
96 *                   relativi alle ordinate.
```

```

95 * @note I valori X e Y di tipo mouse_position_AxisType sono
96     rispettivamente pari a 0
97     e 1.
98 *
99 * @retval None.
100 */
101 void position_on_led(uint8_t position,mouse_position_AxisType axis){
102
103     /*Se il valore di position è relativo all'asse delle ascisse allora si
104         illuminano
105         i led orizzontali di una board STM32F4 discovey*/
106     if(axis==X){
107         /*Se il movimento è verso sinistra si illumina il led LD4 e si spegne
108             LD5*/
109         if(position>127){
110             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,GPIO_PIN_SET);
111             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,GPIO_PIN_RESET);
112         }
113         /*Altrimenti si illumina il led LD5 e si spegne LD4*/
114         else{
115             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,GPIO_PIN_SET);
116             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,GPIO_PIN_RESET);
117         }
118     }
119     /*Allo stesso modo di prima si illuminano i led LD3 o LD6 relativi all'
120         asse Y*/
121     else{
122         if(position>127){
123             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_SET);
124             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13,GPIO_PIN_RESET);
125         }
126         else{
127             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13,GPIO_PIN_SET);
128             HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_RESET);
129         }
130     }
131 }
```

Codice 5.6: "mouse position.c"

### 5.2.2.5 Driver

Infine, grazie alle funzioni precedentemente descritte, è possibile implementare il seguente driver necessario alla gestione del puntatore e del tasto sinistro. Per realizzarlo ci si serve della libreria usb device per HID, la quale offre un middleware pronto all'utilizzo, offrendo, pretanto, un livello di astrazione dell'hardware.

A tal proposito l'implementazione di un mouse necessita l'invio di 4 byte così come descritto dal manuale di riferimento per HID.

```

62  /*Inizializzazione device*/
63  init_mouse(hsp1);
64  uint8_t x_prev=0;
65  uint8_t y_prev=0;
66  /*Loop infinito per gestione posizione puntatore e tasto sinistro*/
67  while (1)
68  {
69      /*Si legge valore tasto sinistro*/
70      USB_TX_Buffer[0]=HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);
71
72      /*Lettura coordinata X*/
73      USB_TX_Buffer[1]=x_position(hsp1);
74
75      /*Lettura coordinata Y*/
76      USB_TX_Buffer[2]=y_position(hsp1);
77
78      /*Byte lasciato inutilizzato*/
79      USB_TX_Buffer[3]=0x00;
80
81      /*Se la soglia di spostamento è superata allora vengono inviati i 4 byte
       necessari allo spostamento del puntatore*/
82      if(USB_TX_Buffer[1]-x_prev>0.25 || USB_TX_Buffer[1]-x_prev<-0.25) {
83          if(USB_TX_Buffer[2]-y_prev>0.25 || USB_TX_Buffer[2]-y_prev<-0.25) {
84              USBD_HID_SendReport (&hUsbDeviceFS,USB_TX_Buffer,4);
85
86              /*Illuminazione led in base allo spostamento*/
87              position_on_led(USB_TX_Buffer[1],X);
88              position_on_led(USB_TX_Buffer[2],Y);
89          }
90      }
91      x_prev=USB_TX_Buffer[1];
92      y_prev=USB_TX_Buffer[2];
93      HAL_Delay(5);
94 }
```

Codice 5.7: "main.c"

Nella precedente implementazione è possibile notare come, per evitare che il puntatore si muova indefinitamente a causa delle continue letture della posizione, viene letto il valore attualmente rilevato e lo si confronta col valore precedente: se la soglia impostata viene superata allora vengono trasmessi i byte necessari, viceversa viene conservato il valore precedente.

# Capitolo 6

## Kernel Linux

### 6.1 Traccia

Si mostri come eseguire il booting su board Zybo Zynq 7000 attraverso una SD-card. In particolare, con riferimento alla fig. 6.1, a partire da un componente GPIO e il Processing System Zynq, si inserisca lo strato di Sistema Operativo per poi scrivere un driver, al fine di pilotare i led della board.



Figura 6.1: Schema di progetto

### 6.2 Procedimento

#### 6.2.1 Requisiti

Il tutorial è stato realizzato utilizzando, come sistema operativo, Ubuntu 16.04. Per eseguire correttamente la procedura è necessario installare:

- **Xilinx Vivado**, che fornisce un ambiente per la progettazione di componenti per FPGA, prodotti dall'azienda Xilinx. Di seguito è stata utilizzata la versione 2016.4, scaricabile al link <https://www.xilinx.com/support/download.html>;
- **Xilinx SDK**, che fornisce un ambiente per la creazione di piattaforme e applicazioni software, destinate a processori embedded di Xilinx. SDK è basato su Eclipse e lavora su progetti hardware realizzati con Vivado.

Inoltre, bisogna procurare i seguenti elementi da caricare sulla SD card:

- **Linux file system;**
- immagine di un **kernel Linux**;
- file **BOOT.bin**;
- **device tree** compilato.

Premesso che è necessario essere in possesso di:

- board **Zynq Zybo 7000**
- **micro SD card** di almeno 4 Gb

nei paragrafi successivi viene illustrato, passo passo, come generare gli elementi da caricare sulla card e le fasi successive utili al completamento dell'esercitazione.

### 6.2.2 Workspace setting

Prima di iniziare, se sulla macchina non è installato il CVS git, digitare il comando

```
sudo apt-get install git
```

e successivamente dare il comando

```
sudo apt-get install u-boot-tools
```

Creare, quindi, una cartella di lavoro *workspace\_master* in cui verranno salvati tutti i file utili all'esercitazione. Nel caso in esame la cartella ha il seguente path: */home/daniele/Scrivania/workspace\_master*. Portarsi all'interno di essa e, aperto un terminale, digitare i comandi

```
git clone -b master-next  
https://github.com/DigilentInc/u-boot-Digilent-Dev.git  
  
git clone -b master-next  
https://github.com/DigilentInc/Linux-Digilent-Dev.git
```

In questo modo, nella cartella vengono create 2 sottocartelle:

- *u-boot-Digilent-Dev*, contenente il codice U-boot della Digilent. L'U-boot è un boot loader per schede Embedded basate su processore ARM, MIPS, PowerPC e altri, che può essere installato su una boot ROM e usato per inizializzare e testare l'hardware oppure per scaricare e avviare codice applicativo;
- *Linux-Digilent-Dev*, contenente il kernel Linux 3.x.

A questo punto, sempre nella cartella di progetto, creare un file di script e chiamarlo *set\_ambiente.sh*. Al suo interno, si scrivono i seguenti comandi utili per configurare alcune variabili d'ambiente per la compilazione del Kernel Linux e del file U-boot:

```

1 export ARCH=arm
2 export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
3 export PATH=$PATH:/usr/Xilinx/SDK/2016.4/gnu/arm/lin/bin:/home/daniele/
   Scrivania/workspace_master/u-boot-Digilent-Dev/tools
4 source /usr/Xilinx/Vivado/2016.4/settings64.sh
5 source /usr/Xilinx/SDK/2016.4/settings64.sh

```

Chiaramente, bisogna adattare opportunamente i path dei comandi in base a dove sono stati installati gli ambienti di lavoro e in base a dove è stata creata la cartella *workspace\_master*. D'ora in avanti, nella documentazione, quando bisognerà riferirsi alla cartella di lavoro si utilizzerà il suo path relativo.

Creare, quindi, una cartella di nome *sd\_file* in cui saranno salvati tutti i file utili al settaggio dell'SD card. Infine, creare le seguenti sottocartelle *workspace\_master/zybo\_base\_system/source/hw* in cui verrà salvato il progetto hardware sviluppato con Vivado.

Alla fine della fase di settaggio, la situazione dovrebbe essere simile a quella in fig. 6.2.

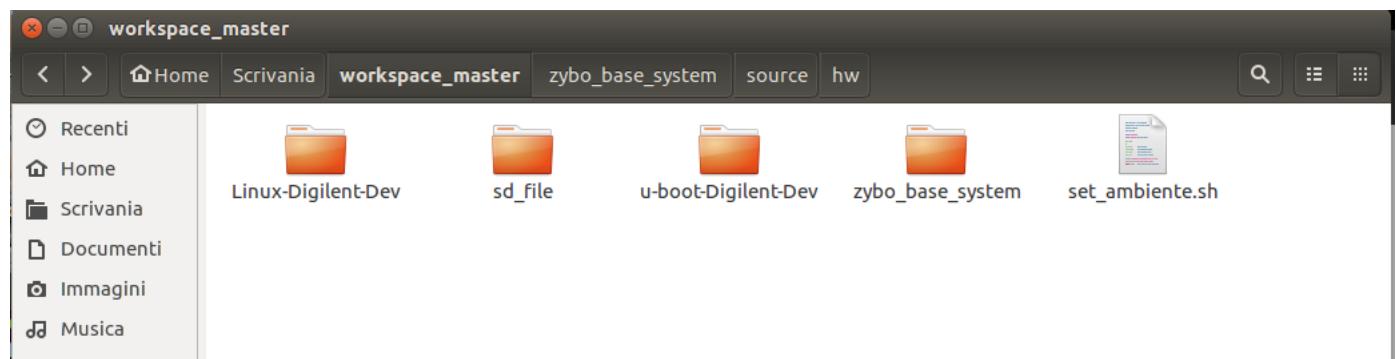


Figura 6.2: Cartella di progetto

### 6.2.3 Sintesi Hardware

L'obiettivo di questa prima fase è di sintetizzare, attraverso l'ambiente Vivado, un componente GPIO con le interruzioni abilitate, attraverso cui pilotare i led della board.

Aperto Vivado, dunque, realizzare un nuovo progetto di nome *project\_gpio\_interrupt* nel path creato appositamente nel paragrafo precedente, fig. 6.3.

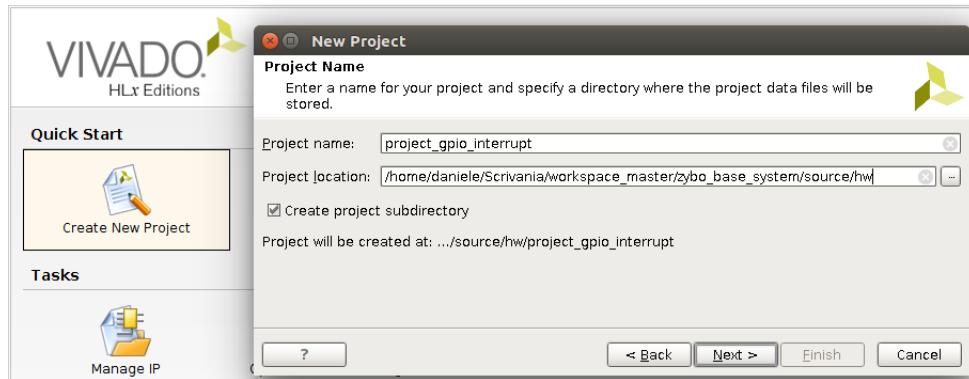


Figura 6.3: Creazione nuovo progetto in Vivado

A questo punto, seguendo i passi descritti nel Tutorial 1, si crea un nuovo Block Design a cui si da il nome *design\_for\_gpio\_interrupt*, nel quale si importano i seguenti IP-core:

- ZYNQ7 Processing System;
- AXI GPIO (fornita da Xilinx).

Facendo doppio click sul blocco AXI GPIO, si apre una finestra che consente di personalizzare l'IP-core importato. Selezionare, come interfaccia della board, quella dei led (*led 4bits*) e spuntare l'opzione *enable interrupts*, fig. 6.4.

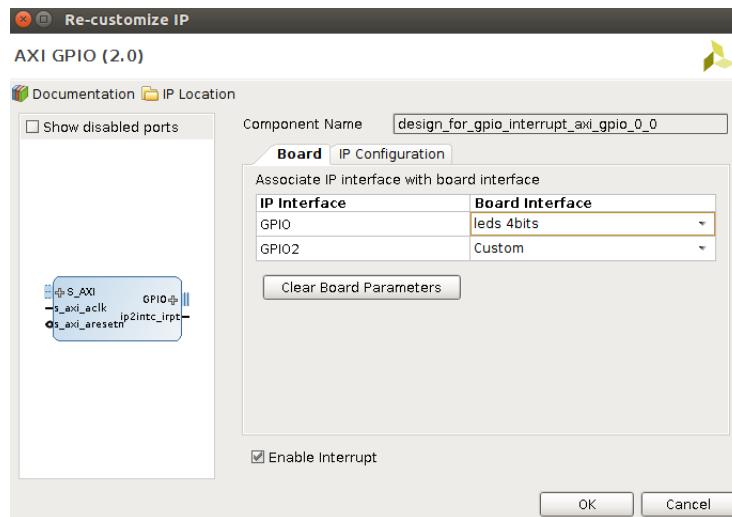


Figura 6.4: Re-customize IP AXI GPIO

Facendo doppio click sul blocco ZYNQ7 Processing System, si apre un'altra finestra Re-customize IP, fig. 6.5. Nel Zynq Block Design è possibile osservare tutta la parte architetturale della Programmable Logic.

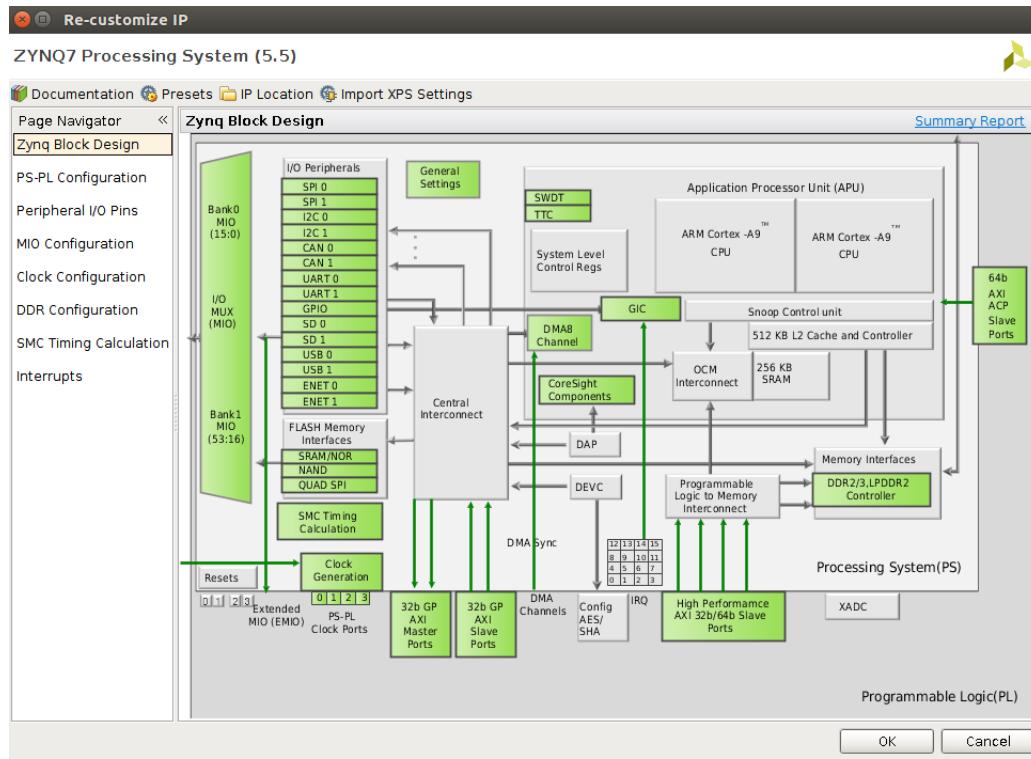


Figura 6.5: Re-customize IP ZYNQ7 Processing System

Nel Page Navigator a sinistra, selezionare Interrupts. Spuntare, quindi, l'opzione *Fabric Interrupts* e, dopo aver aperto il menù a tendina PL-PS Interrupt Ports, spuntare anche *IRQ\_F2P*, fig. 6.6.

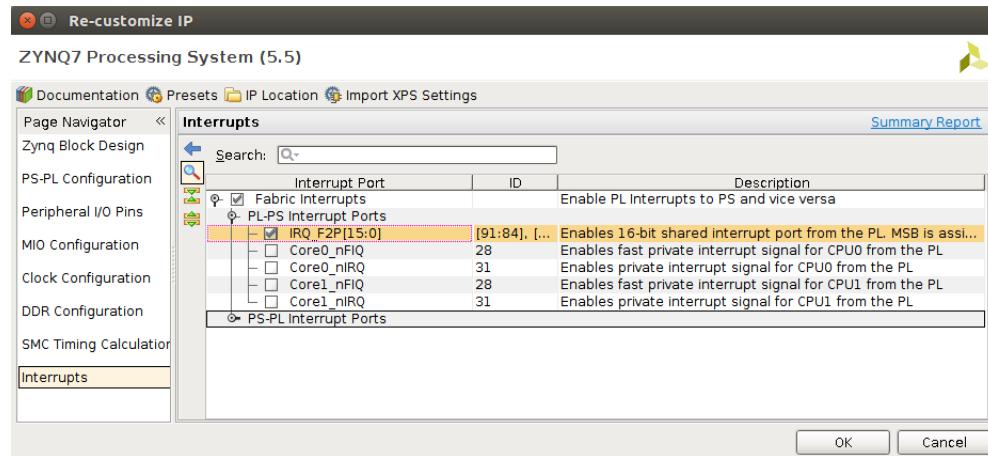


Figura 6.6: IRQ\_F2P Interrupt

A questo punto bisogna eseguire le operazioni di **Run Block Automation** e **Run Connection Automation**. Quando si esegue quest'ultima spuntare l'opzione All automation, presente nella porzione sinistra della finestra che si apre, fig. 6.7.

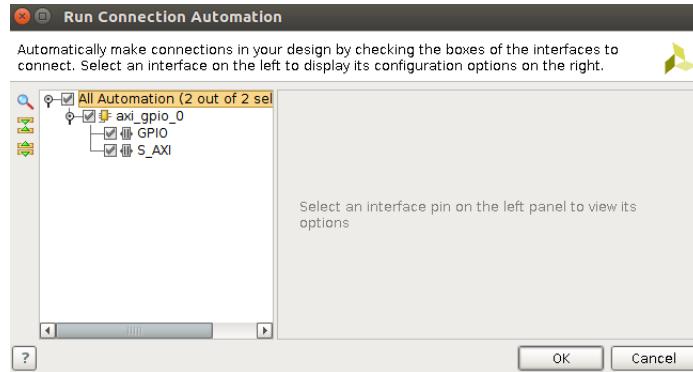


Figura 6.7: Run Connection Automation

A questo punto, bisogna collegare manualmente l’interfaccia *ip2intc\_irpt* dell’AXI GPIO con IRQ\_F2P di ZYNQ7 Processing System. Tale collegamento è evidenziato in giallo in fig. 6.8 .

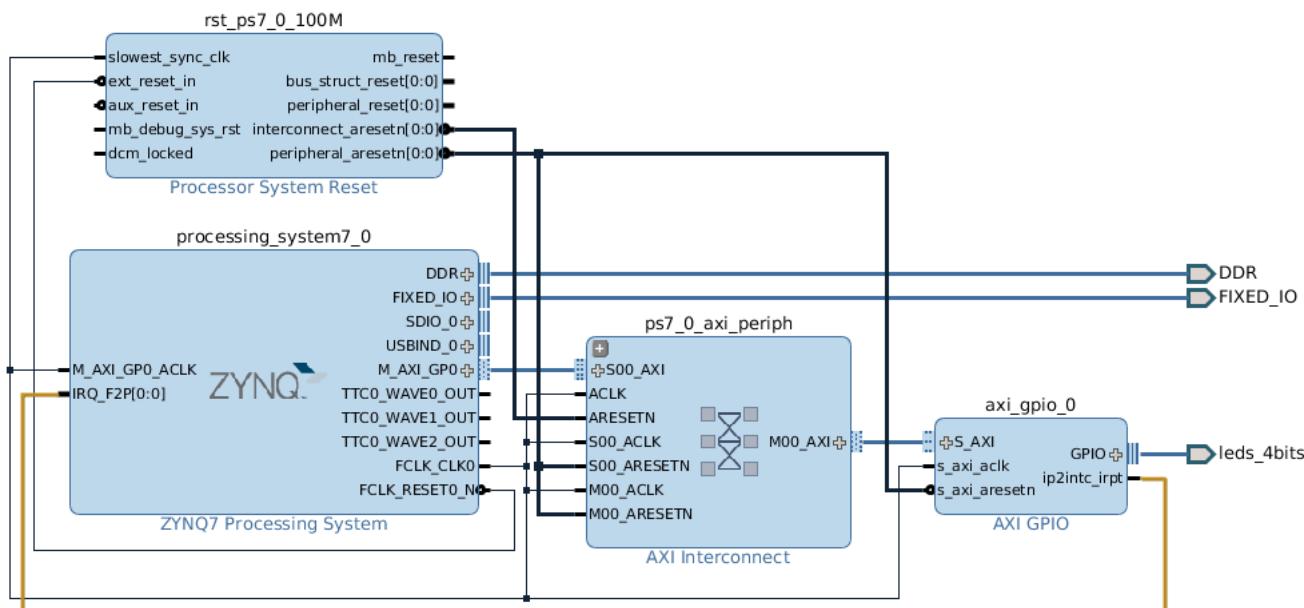


Figura 6.8: Interrupt Connection

Dopo aver validato il design (F6), si crea il wrapper del progetto.

A questo punto, in Flow Navigator, cliccare **Generate Bitstream**. Si noti che viene avviata automaticamente sia la fase di sintesi sia quella di implementazione, propedeutiche per la generazione del bitsream del progetto.

Come ci si può aspettare, se tra i Layout del progetto si seleziona **Floorplanning**, nella schermata I/O Ports è stato fatto automaticamente il mapping dei pad del GPIO con i led della board.

Andare su **File → Export → Export Hardware** e spuntare l’opzione *Include bitstream*. Infine cliccare su **File → Launch SDK** per aprire l’ambiente Xilinx SDK.

### 6.2.4 Compilazione U-boot

In questa fase si mostra come compilare l'**U-boot**, Universal Boot Loader. Questo viene lanciato dal **FSBL**, First Stage Boot Loader, e ha il compito di decomprimere il kernel Linux, caricarlo dall'SD-card alla memoria RAM e lanciarlo.

Preliminamente bisogna modificare il file **zynq\_zybo.h**, presente in *workspace\_master/u-boot-Digilent-Dev/include/configs*, per evitare che l'U-boot carichi i settaggi del ramdisk. In particolare, si eliminano tutte le occorrenze relative al ramdisk. Nelle fig. 6.9 e 6.10 viene mostrato un esempio in cui è stata eliminata la prima occorrenza evidenziata in arancione.



```
#define CONFIG_EXTRA_ENV_SETTINGS \
    "kernel_image=uImage\0" \
    "ramdisk_image=uramdisk.image.gz\0" \
    "devicetree_image=devicetree.dtb\0" \
    "bitstream_image=system.bit.bin\0" \
    "boot_image=BOOT.bin\0" \
    "loadbit_addr=0x100000\0" \
    "loadbootenv_addr=0x2000000\0" \
    "kernel_size=0x500000\0" \
    "devicetree_size=0x20000\0" \
    "ramdisk_size=0x5E0000\0" \
    "boot_size=0xF00000\0" \
    "fdt_high=0x20000000\0" \
    "initrd_high=0x20000000\0" \
    "bootenv=uEnv.txt\0" \
    "loadbootenv=fatload mmc 0 ${loadbootenv_addr} ${bootenv}\0" \
    "importbootenv=echo Importing environment from SD ...; " \
        "env import -t ${loadbootenv_addr} ${filesize}\0" \
    "mmc_loadbit_fat=echo Loading bitstream from SD/MMC/eMMC to RAM.. && " \
        "mmcinfo && " \
        "fatload mmc 0 ${loadbit_addr} ${bitstream_image} && " \
        "fpga load 0 ${loadbit_addr} ${filesize}\0" \
    "norboot=echo Copying Linux from NOR flash to RAM.. && " \
        "cp.b 0xE2100000 0x3000000 ${kernel_size} && " \
        "cp.b 0xE2600000 0x2A00000 ${devicetree_size} && " \
        "echo Copying ramdisk .." \

```

Figura 6.9: Occorrenze ramdisk in zynq\_zybo.h

```

#define CONFIG_EXTRA_ENV_SETTINGS \
"kernel_image=uImage\0" \
"devicetree_image=devicetree.dtb\0"      \
"bitstream_image=system.bit.bin\0"        \
"boot_image=BOOT.bin\0" \
"loadbit_addr=0x100000\0" \
"loadbootenv_addr=0x2000000\0" \
"kernel_size=0x500000\0" \
"devicetree_size=0x20000\0" \
"ramdisk_size=0x5E0000\0" \
"boot_size=0xF00000\0" \
"fdt_high=0x20000000\0" \
"initrd_high=0x20000000\0" \
"bootenv=uEnv.txt\0" \
"loadbootenv=fatload mmc 0 ${loadbootenv_addr} ${bootenv}\0" \
"importbootenv=echo Importing environment from SD ...; " \
"    env import -t ${loadbootenv_addr} ${filesize}\0" \
"mmc_loadbit_fat=echo Loading bitstream from SD/MMC/eMMC to RAM.. && \
" \
"mmcinfo && " \
"fatload mmc 0 ${loadbit_addr} ${bitstream_image} && " \
"fpga load 0 ${loadbit_addr} ${filesize}\0" \
"norboot=echo Copying Linux from NOR flash to RAM... && " \
"    cp.b 0xE2100000 0x3000000 ${kernel_size} && " \
"    cp.b 0xE2600000 0x2A00000 ${devicetree_size} && " \
"echo Copying ramdisk... && " \
"cp.b 0xE2620000 0x20000000 ${ramdisk_size} && " \
"echo Done copying ramdisk to RAM"

```

Figura 6.10: Eliminazione prima occorrenza ramdisk in zynq\_zybo.h

Dopo aver eliminato tutte le occorrenze, bisogna modificare opportunamente tutte le righe di codice che contengono bootm. Tale comando viene usato per fare il boot dell’immagine del kernel salvata in memoria. Il primo parametro passato è l’indirizzo dove si trova il kernel; il secondo parametro rappresenta l’indirizzo dell’immagine dell’initrd; il terzo parametro rappresenta, invece, l’indirizzo del device tree blob. Non dovendo utilizzare l’initrd, bisogna sostituire il suo indirizzo con il carattere – , come viene evidenziato in fig. 6.11.

```

ctrl_d_night=0x20000000\0
"bootenv=uEnv.txt\0" \
"loadbootenv=fatload mmc 0 ${loadbootenv_addr} ${bootenv}\0" \
"importbootenv=echo Importing environment from SD ...; " \
"    env import -t ${loadbootenv_addr} ${filesize}\0" \
"mmc_loadbit_fat=echo Loading bitstream from SD/MMC/eMMC to RAM.. &&
" \
"mmcinfo && " \
"fatload mmc 0 ${loadbit_addr} ${bitstream_image} && " \
"fpga load 0 ${loadbit_addr} ${filesize}\0" \
"norboot=echo Copying Linux from NOR flash to RAM... && " \
"cp.b 0xE2100000 0x3000000 ${kernel_size} && " \
"cp.b 0xE2600000 0x2A00000 ${devicetree_size} && " \
"bootm 0x3000000 - 0x2A00000\0" \
"qspiboot=echo Copying Linux from QSPI flash to RAM... && " \
"sf probe 0 0 0 && " \
"sf read 0x3000000 0x100000 ${kernel_size} && " \
"sf read 0x2A00000 0x600000 ${devicetree_size} && " \
"bootm 0x3000000 - 0x2A00000\0" \
"uenvboot=" \
"if run loadbootenv; then " \
"    echo Loaded environment from ${bootenv}; " \
"    run importbootenv; " \
"fi; " \
"if test -n $uenvcmd; then " \
"    echo Running uenvcmd ...; " \
"    run_uenvcmd: " \
"run_uenvcmd: " \
Header C/C++/ObjC ▾ Larg. tab.: 8 ▾ Rg 63, Col 50 ▾ INS

```

Figura 6.11: Modifica comando bootm

Portarsi ora nella cartella di progetto, aprire un terminale e lanciare il comando

```
source set_ambiente.sh
```

Entrare, quindi, nella cartella *u-boot-Digilent-Dev* con il comando

```
cd u-boot-Digilent-Dev/
```

ed eseguire i comandi

```
make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config
```

per selezionare la configurazione specifica della board Zybo e

```
make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
```

per compilare l'U-boot file.

A questo punto nella cartella in cui ci si trova è stato creato il file u-boot che bisogna copiare nella cartella *sd\_file* aggiungendo l'estensione .elf, usando il comando

```
cp u-boot ../sd_file/u-boot.elf
```

### 6.2.5 Generazione FSBL

L'FSBL è il responsabile del caricamento del bitstream e della configurazione del Processing System della Zynq a tempo di boot.

Per generarlo, in Xilinx SDK aperto in precedenza, andare su **File → New → Application Project**. e chiamare il progetto *fsbl*, fig. 6.12.

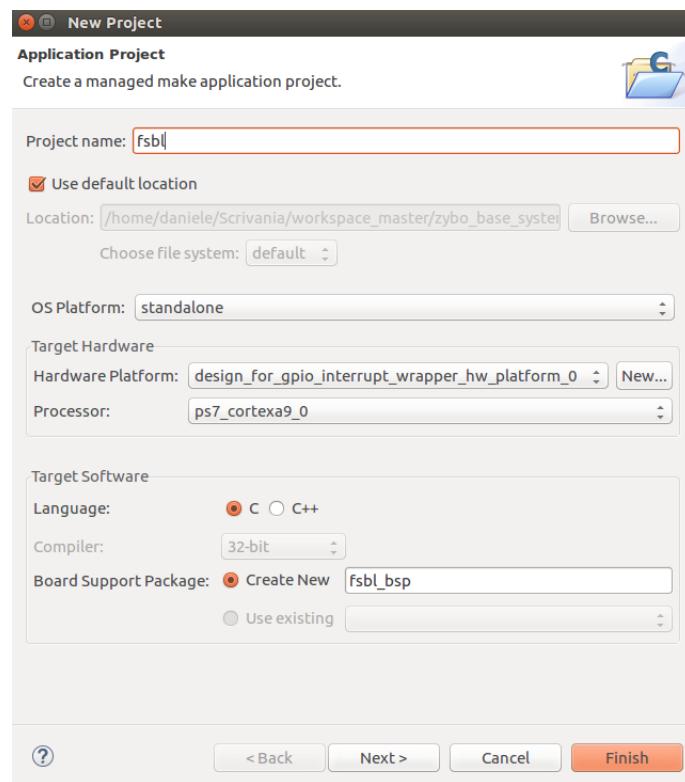


Figura 6.12: New Project

Come template si seleziona Zynq FSBL, fig. 6.13.

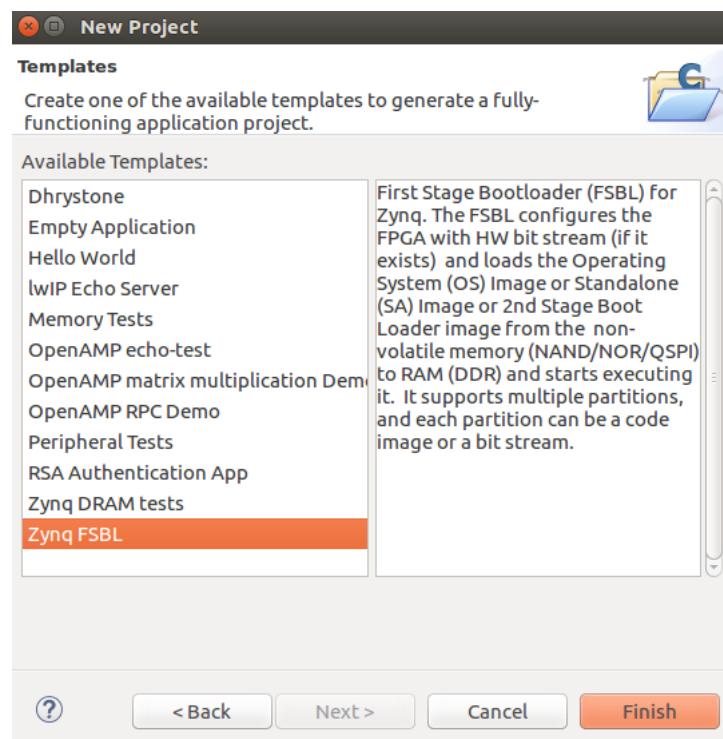


Figura 6.13: Zynq FSBL Template

A questo punto andare su **Project** → **Build All** per fare il build del progetto e generare il file **fsbl.elf** nella directory *workspace\_master/zybo\_base\_system/source/hw/project\_gpio\_interrupt/project\_gpio\_interrupt.sdk/fsbl/Debug*.

### 6.2.6 Generazione BOOT.bin

Arrivati a questa fase si ha tutto il necessario per generare il **BOOT.bin**. Quest'ultimo contiene al suo interno alcuni file specifici Xilinx utili a configurare le due parti della Zynq: la Programmable Logic e il Processing System. Al suo interno integra, in ordine, i seguenti file:

1. fsbl.elf;
2. design\_for\_gpio\_interrupt\_wrapper.bit;
3. u-boot.elf.

Sempre nell'ambiente Xilinx SDK, nel menu in alto, selezionare **Xilinx Tools** → **Create Boot Image**. Nel wizard che si apre, si sceglie di salvare i file output.bif e BOOT.bin nella sottocartella *sd\_file*, indicandolo rispettivamente in Output BIF file path e in Output path. Inoltre, nella sezione Image Boot Partitions, cliccando su Add si aggiungono i file sopra elencati, rispettando pedissequamente l'ordine indicato, fig. 6.14.

Per completezza, si riportano, di seguito, i path dei file da inserire.

- 1 workspace\_master/zybo\_base\_system/source/hw/project\_gpio\_interrupt/project\_gpio\_interrupt.sdk/fsbl/Debug/fsbl.elf
- 2 workspace\_master/zybo\_base\_system/source/hw/project\_gpio\_interrupt/project\_gpio\_interrupt.sdk/design\_for\_gpio\_interrupt\_wrapper\_hw\_platform\_0/design\_for\_gpio\_interrupt\_wrapper.bit
- 3 workspace\_master/sd\_file/u-boot.elf

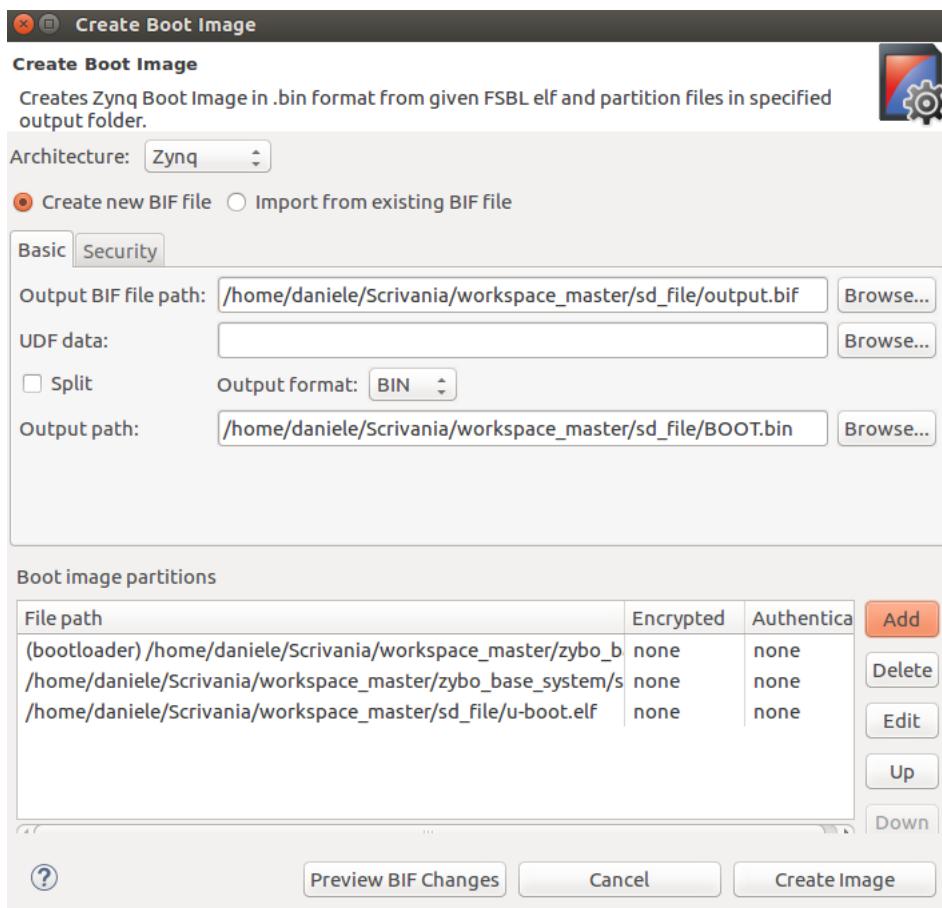


Figura 6.14: Create Boot Image

Cliccare, infine, su Create Image.

### 6.2.7 Generazione uImage

Per compilare il kernel Linux portarsi nella cartella *workspace\_master/Linux-Digilent-Dev* con il comando

```
cd ../../Linux-Digilent-Dev/
```

Digitando il comando

```
make config
```

è possibile creare una configurazione custom del kernel che si vuole compilare, mediante un'interfaccia testuale. La medesima operazione è possibile farla utilizzando il comando

```
make menuconfig
```

che, invece, mette a disposizione un vero e proprio menu di configurazione. Per maggiori approfondimenti si rimanda alla lettura del file README presente in *workspace\_master/Linux-Digilent-Dev*.

In alternativa, si può utilizzare il comando

```
make ${PLATFORM}_defconfig
```

che crea un file di configurazione utilizzando valori simbolo di default per l'architettura \${PLATFORM} specificata. Per semplicità, in questa documentazione viene scelta quest'ultima strada tra le tre proposte. Digitare, dunque, il comando

```
make xilinx_zynq_defconfig
```

A questo punto, con il comando

```
make
```

viene creato il file zImage, immagine compressa (*zipped*) del Kernel, nella directory *workspace\_master/Linux-Digilent-Dev/arch/arm/boot*. Per poter essere utilizzata dal bootloader, tale immagine deve essere decompressa e questo lo si fa attraverso il comando

```
make UIMAGE_LOADADDR=0x8000 uImage
```

che specifica anche a che locazione di memoria viene caricato il kernel Linux. Per completezza in fig. 6.15 si riporta l'output dello script eseguito.

```
daniele@daniele-Inspiron-7559:~/Scrivania/workspace_master/Linux-Digilent-Dev$ make UIMAGE_LOADADDR=0x8000 uImage
CHK include/config/kernel.release
CHK include/generated/uapi/linux/version.h
CHK include/generated/utsrelease.h
make[1]: "include/generated/mach-types.h" è aggiornato.
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
CHK kernel/config_data.h
Kernel: arch/arm/boot/Image is ready
Kernel: arch/arm/boot/zImage is ready
UIMAGE arch/arm/boot/uImage
Image Name: Linux-3.18.0-xilinx-46110-gd627f
Created: Sat Jun 10 12:48:46 2017
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 3447896 Bytes = 3367.09 kB = 3.29 MB
Load Address: 00008000
Entry Point: 00008000
  Image arch/arm/boot/uImage is ready
daniele@daniele-Inspiron-7559:~/Scrivania/workspace_master/Linux-Digilent-Dev$
```

Figura 6.15: Create uImage file

Come si evince dalla precedente figura, il file uImage (*unzipped*) viene creato nella stessa directory del file zImage. Lo si copia, quindi, nella cartella sd\_file

```
cp arch/arm/boot/uImage ../sd_file/uImage
```

### 6.2.8 Generazione del DTB

Il **DTB**, Device Tree Blob, è una sorta di database che rappresenta tutti i componenti hardware di una specifica board. In sostanza è il meccanismo scelto per passare informazioni hardware di basso livello dal bootloader al kernel. Il DTB viene generato a partire da un altro file chiamato **DTS**, Device Tree Source. Nel caso specifico, quest'ultimo lo si trova nella cartella *workspace\_master/Linux-Digilent-Dev/arch/arm/boot/dts*. Tra i file .dts disponibili, cercare, dunque, il file **zynq-zybo.dts** (specifico della Zybo) ed aprirlo con un editor di testo.

Apportare le modifiche alle righe 42, 51 e 60 presenti in fig. 6.16.

```

35     model = "Xilinx Zynq";
36     aliases {
37         ethernet0 = &ps7_ethernet_0;
38         serial0 = &ps7_uart_1;
39         spi0 = &ps7_qspi_0;
40     } ;
41     chosen {
42         bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait";
43         linux,stdout-path = "/amba@0/serial@e0001000";
44     } ;
45     cpus {
46         #address-cells = <1>;
47         #size-cells = <0>;
48         ps7_cortexa9_0: cpu@0 {
49             bus-handle = <&ps7_axi_interconnect_0>;
50             clock-latency = <1000>;
51             clocks = <&clkc 2>;
52             compatible = "arm,cortex-a9";
53             device_type = "cpu";
54             interrupt-handle = <&ps7_scugic_0>;
55             operating-points = <666667 1000000 333334 1000000 222223 1000000>;
56             reg = <0x0>;
57         } ;
58         ps7_cortexa9_1: cpu@1 {
59             bus-handle = <&ps7_axi_interconnect_0>;
60             clocks = <&clkc 2>;
61             compatible = "arm,cortex-a9";
62             device_type = "cpu";
63             interrupt-handle = <&ps7_scugic_0>;
64         } ;
65     } ;
66 
```

Figura 6.16: Modifiche a zynq-zybo.dts

A questo punto, sempre nella cartella *workspace/Linux-Digilent-Dev* digitare

```

./scripts/dtc/dtc -I dts -O dtb -o ../sd_file/devicetree.dtb
./arch/arm/boot/dts/zynq-zybo.dts

```

per generare **devicetree.dtb** nella cartella *sd\_file*.

### 6.2.9 Partizionamento SD card

Generati i file **BOOT.bin**, **devicetree.dtb** e **uImage** non resta che formattare la micro SD card. Nel caso in esame viene utilizzato il tool **gparted**. Se non lo si possiede, installarlo mediante il comando

```
sudo apt-get install gparted
```

Collegare opportunamente la micro SD card al computer. Nel caso in esame viene utilizzato un adattatore mini SD card - SD card. Avviato, quindi, gparted con

```
sudo gparted
```

creare le seguenti partizioni:

- **BOOT\_F**, di tipo FAT32 di almeno 1 Gb, fig. 6.17;
- **ROOT\_FS**, di tipo ext4 di almeno 3 Gb, fig. 6.18.

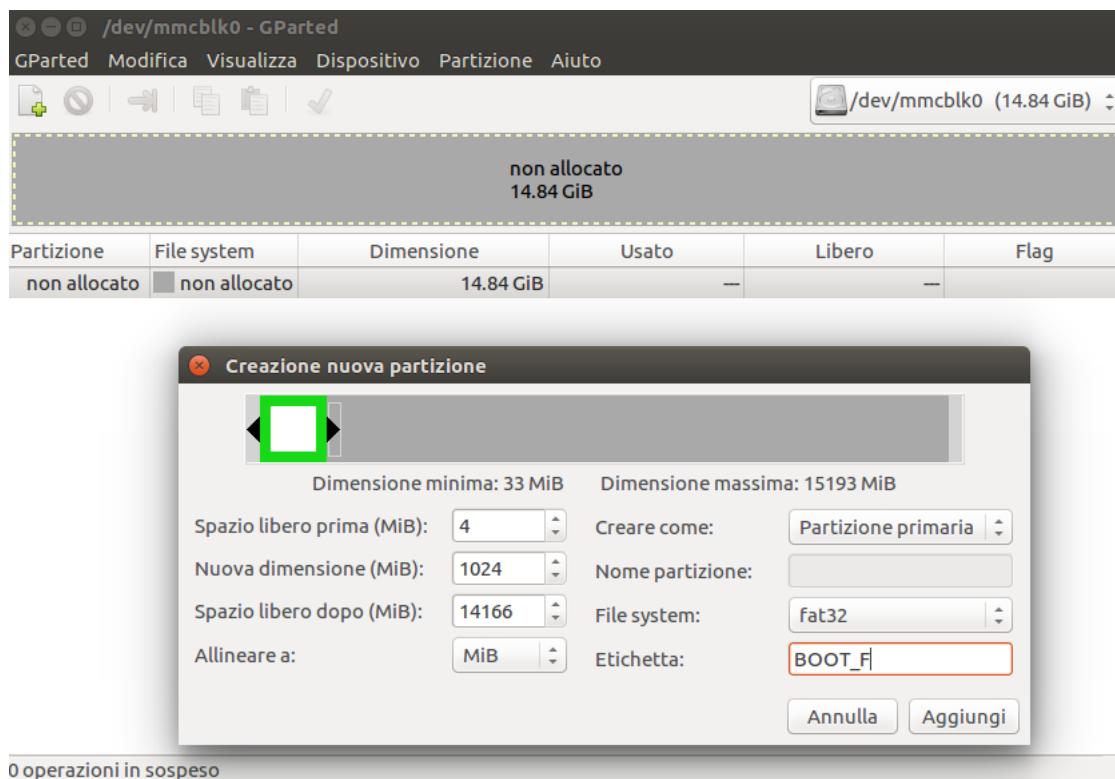


Figura 6.17: Prima partizione: BOOT\_F

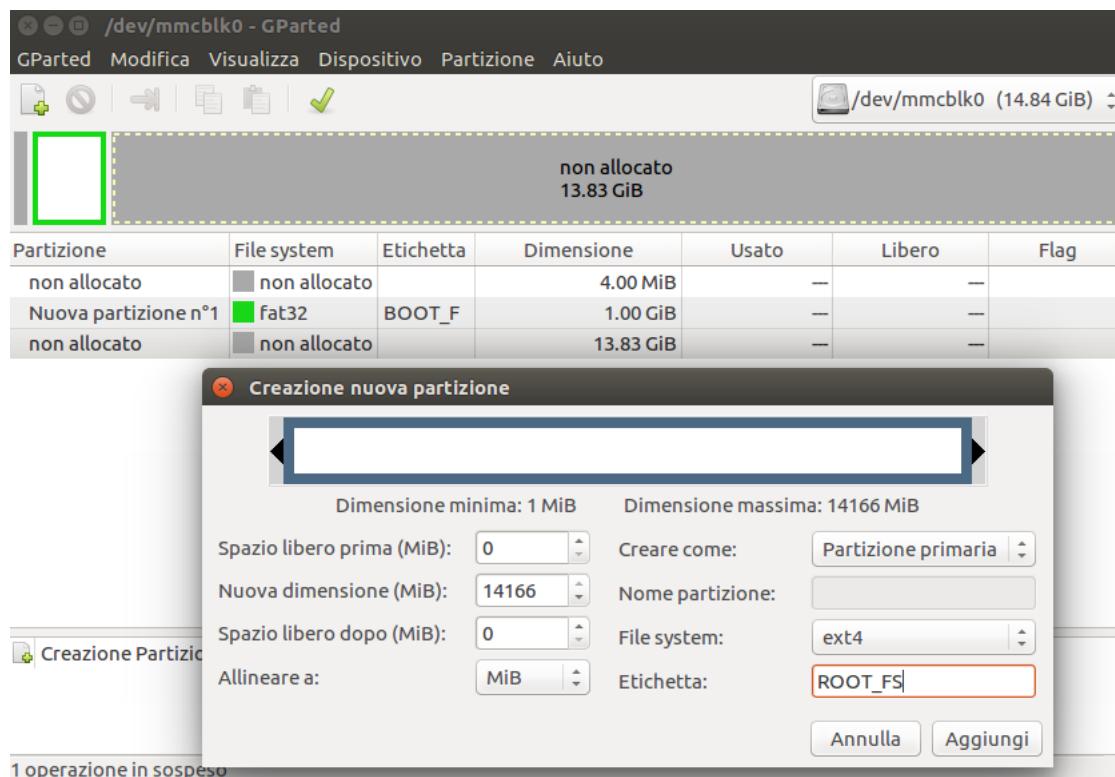


Figura 6.18: Seconda partizione: ROOT\_FS

Al termine delle operazioni di partizionamento la situazione dovrebbe essere quella in fig. 6.19.

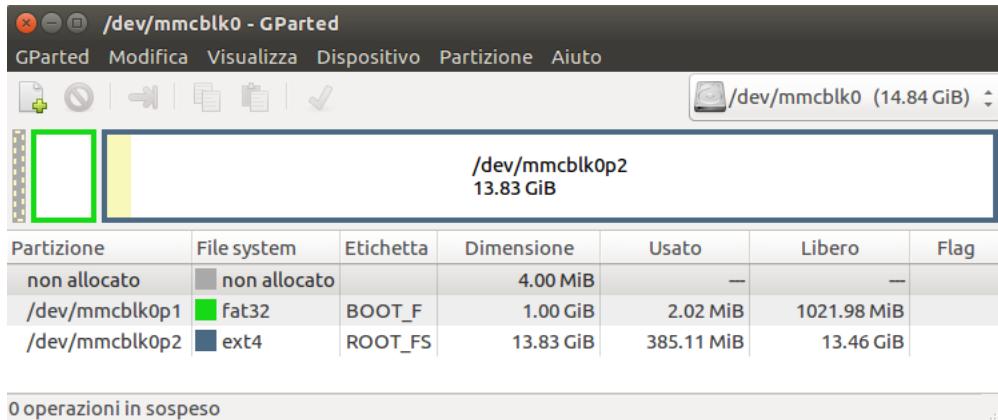


Figura 6.19: Partizionamento SD card

Si noti che bisogna lasciare 4 Mb di spazio non allocato.

### 6.2.10 Settaggio SD card per il boot

A questo punto nella partizione BOOT\_F della micro SD card bisogna copiare i file:

- BOOT.bin;
- devicetree.dtb;
- uImage.

Per far ciò, portarsi nella cartella di lavoro e digitare:

```
cp sd_file/BOOT.bin /media/daniele/BOOT_F
```

```
cp sd_file/devicetree.dtb /media/daniele/BOOT_F
```

```
cp sd_file/uImage /media/daniele/BOOT_F
```

Il risultato dovrebbe essere quello mostrato in fig. 6.20.

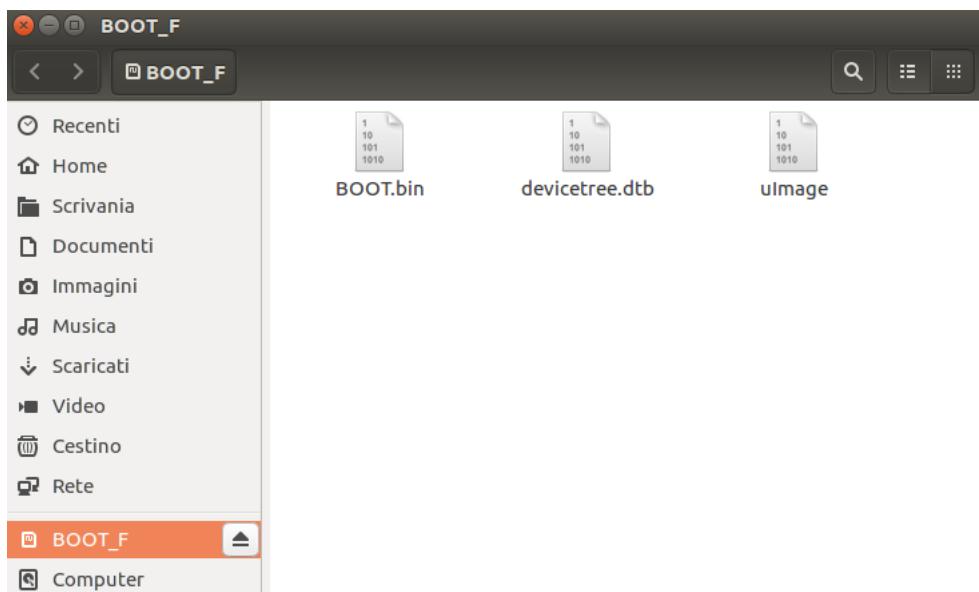


Figura 6.20: Copia file in BOOT\_F

Ora, bisogna caricare un filesystem nella seconda partizione predisposta appositamente, ROOT\_FS. Nel caso in esame viene scelto **Linaro file system**. Utilizzando il link <http://releases.linaro.org/> è possibile ottenere la versione che si preferisce. In questo tutorial viene scaricata <https://releases.linaro.org/archive/12.11/ubuntu/precise-images/ubuntu-desktop/linaro-precise-ubuntu-desktop-20121124-560.tar.gz>.

Portarsi nella cartella in cui è stato scaricato il file system. Creare una cartella di nome *linaro* sotto */temp* dove andare a copiare l'immagine Linaro zippata.

```
mkdir -p /tmp/linaro

sudo cp linaro-precise-ubuntu-desktop-20120923-436.tar.gz
      /tmp/linaro/fs.tar.gz
```

Per verificare che il pacchetto sia stato copiato

```
cd /tmp/linaro
```

```
ls
```

il risultato dovrebbe essere

```
fs.tar.gz
```

Adesso, spacchettare il file utilizzando

```
sudo tar zxf fs.tar.gz
```

Digitando

```
ls
```

questa volta il risultato dovrebbe essere

```
binary fs.tar.gz
```

A questo punto, verificare che la partizione su cui si dovrà mettere il file system non sia stata automaticamente montata dal sistema. Digitare

```
df
```

se il risultato è come quello mostrato in fig. 6.21

File system	1K-blocchi	Usati	Disponib.	Uso%	Montato su
udev	8126552	0	8126552	0%	/dev
tmpfs	1630196	9808	1620388	1%	/run
/dev/sda7	23765884	10776632	11758968	48%	/
/dev/sdb3	158471932	47431168	102967756	32%	/usr
tmpfs	8150968	3244	8147724	1%	/dev/shm
tmpfs	5120	4	5116	1%	/run/lock
tmpfs	8150968	0	8150968	0%	/sys/fs/cgroup
/dev/sda1	507904	37900	470004	8%	/boot/efi
/dev/sdb2	47953800	24116268	21378552	54%	/home
tmpfs	1630196	80	1630116	1%	/run/user/1000
/dev/mmcblk0p2	14146036	35424	13368980	1%	/media/daniele/ROOT_FS
/dev/mmcblk0p1	1046516	8324	1038192	1%	/media/daniele/BOOT_F

Figura 6.21: df result

smontare la partizione ROOT\_FS, utilizzando

```
sudo umount /media/daniele/ROOT_FS
```

Si crea, adesso, una cartella temporanea su cui andare a montare la micro SD

```
mkdir -p /tmp/sd_ext4
```

```
sudo mount /dev/mmcblk0p2 /tmp/sd_ext4
```

Usare il comando rsync per copiare la root del file system sulla micro SD

```
cd binary/boot/filesystem.dir
```

```
sudo rsync -a ./ /tmp/sd_ext4
```

Prima di rimuovere la scheda utilizzare il comando umount per smontare /tmp/sd\_ext4 ed assicurarsi che tutti i file siano stati sincronizzati

```
sudo umount /tmp/sd_ext4
```

Quest'operazione potrebbe richiedere diversi minuti.

### 6.2.11 Test di boot

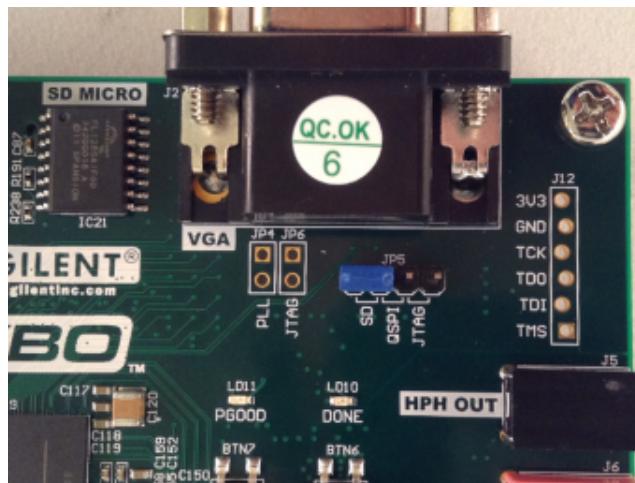
Per procedere alla fase di booting del kernel compilato tramite scheda SD sulla board Zybo, è necessario avere sulla propria macchina un emulatore di terminale di trasmissione seriale. In questa documentazione viene utilizzato **gtkterm**.

Per poterlo scaricare, digitare

```
sudo apt-get install gtkterm
```

A questo punto eseguire i seguenti passi:

1. Inserire la SD card appositamente predisposta nello slot della board
2. Assicurarsi che il jumper JP7 sia settato su SD, fig. 6.22
3. Da terminale, digitare `sudo gtkterm`
4. Nel menu di gtkterm andare su **Configuration → Port**
5. Settare le impostazioni mostrate in fig. 6.23
6. Premere il bottone PS-SRST e testare da terminale che effettivamente la board fa il boot del sistema che è stato configurato nei passi precedenti



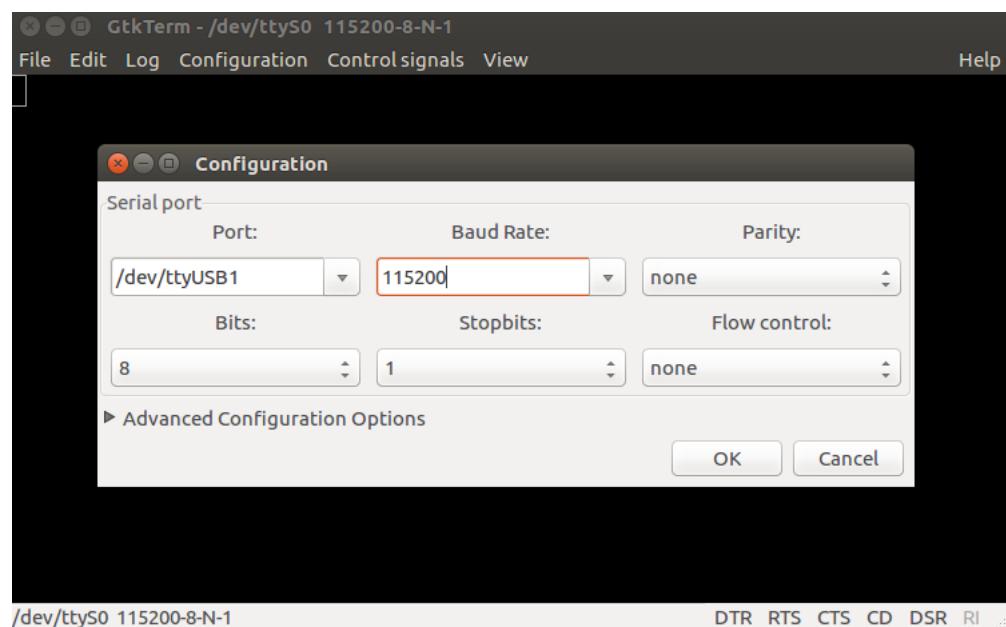


Figura 6.23: Gtkterm configuration

# Capitolo 7

## Driver Linux

### 7.1 Traccia

Si scriva un driver Linux in grado di pilotare una periferica GPIO di lettura o scrittura. Si esegua la traccia utilizzando le 3 diverse modalità di seguito elencate:

1. Accedere ai registri della periferica tramite indirizzo fisico della stessa;
2. Accedere alla GPIO come device uio, senza utilizzare il meccanismo delle interruzioni;
3. Accedere alla GPIO come device uio, utilizzando il meccanismo delle interruzioni.

In un primo momento, si utilizzi la periferica GPIO fornita dalla Xilinx e messa a disposizione come IP-core nell'ambiente Vivado; in un secondo momento, si ripeta l'esercizio utilizzando la GPIO custom sviluppata per il Capitolo 1.

### 7.2 Procedimento: GPIO Xilinx

I driver di seguito presentati fanno riferimento tutti unicamente alla periferica GPIO Xilinx.

#### 7.2.1 Prima tipologia

Questa prima modalità è quella, forse, più difficile da trovare nella realtà. Per fare uso di un driver del genere, infatti, non solo bisogna conoscere l'esatto indirizzo fisico della periferica a cui si vuole accedere, ma bisogna anche avere i privilegi di root nell'esecuzione del driver. Il perché è presto spiegato: tale meccanismo prevede di accedere direttamente alla memoria, che in Linux è vista come un file sotto /dev (/dev/mem). Più avanti nella trattazione si capirà ancora meglio tale concetto.

Di seguito si descrivono i file utilizzati per la scrittura del driver, andando ad evidenziare le principali peculiarità di ognuno di essi.

##### 7.2.1.1 nodriver\_header.h

Dopo aver richiamato alcune librerie utili per la definizione delle funzioni, si definiscono le macro RESET, GREEN e BOLDWHITE. Queste vengono utilizzate nella stampa di alcuni messaggi a

terminale, per rendere più fruibile la lettura. Le macro READ e WRITE sono invece utilizzate per definire se l'operazione che si vuole eseguire sulla periferica GPIO è di lettura o scrittura. La macro PAGE\_SIZE ha come valore la grandezza della pagina che il sistema operativo usa nel meccanismo della paginazione. MASK\_SIZE, invece, è una maschera di bit composta da tutti 1, eccetto che nella parte finale dove sono presenti tanti 0 quanti sono i bit che occorrono per definire l'offset di una pagina. A titolo d'esempio, si supponga che il SO utilizzi un sistema di paginazione con pagine grandi  $2^4$  bit = 0...010000 Tale valore viene restituito dalla system call `sysconf(_SC_PAGESIZE)` e associato alla macro PAGE\_SIZE. Dato un certo indirizzo di memoria, per estrarre il numero di pagina, escludendo, quindi, il suo offset, si utilizza, appunto, MASK\_SIZE. Nel caso preso d'esempio questa sarà pari a 1.....110000. Il calcolo è molto semplice, basta sottrarre 1 e negare tutto il risultato. Infine, vengono definiti i prototipi di tutte le funzioni che saranno richiamate all'interno del driver.

```

192 /**
193 ****
194 * @file      nodriver.h
195 * @author Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
196 *           Sistemi Embedded 2016-2017
197 * @version   V1.0
198 * @date     22-June-2017
199 * @brief    library nodriver gpio
200 ****
201 */
202 /* Includes
203 -----------*/
204 #include <stdio.h>
205 #include <stdlib.h>
206 #include <unistd.h>
207 #include <sys/mman.h>
208 #include <fcntl.h>
209 #include <errno.h>
210 /* Colour MACRO
211 -----------*/
212 #define RESET  "\033[0m"  /*!< Reset colour */
213 #define GREEN   "\033[32m"  /*!< Green colour */
214 #define BOLDWHITE "\033[1m\033[37m" /*!< Bold White colour */
215 /* GPIO operation MACRO
216 -----------*/
217 #define READ 0      /*!< Read operation is set */
218 #define WRITE 1     /*!< Write operation is set */
219 /* GPIO page MACRO
220 -----------*/
221 #define PAGE_SIZE sysconf(_SC_PAGESIZE)  /*!< Page size used by SO */

```

```

221 #define MASK_SIZE (~(PAGE_SIZE-1))      /*!< Bit mask to extract page address,
222     without offset */
223
224 /* Function prototypes
225 -----------*/
226 void usage(char *name); /*!< usage function is called to help the user */
227 int parse_command(int argc,char **argv,int* address,int* r_w,int* value); /*!
228     !< parse_command function is called to parse arguments passed to driver
229     */
230 int open_memory(int* fd,int phy_address, int* page_offset,void** virtual_address);/*!< open_memory function is called to open memory file
231     */
232 void read_gpio(int* value,void* virtual_address,int page_offset);/*!<
233     read_gpio function is called to do read operation from GPIO */
234 void write_gpio(int value,int phy_address,void* virtual_address, int page_offset);/*!< write_gpio function is called to do write operation on
235     GPIO */
236 void close_memory(int fd, void* virtual_address);/*!< close_memory function
237     is called to close memory file */

```

Codice 7.1: "nodriver header.h"

### 7.2.1.2 nodriver\_function.h

Di seguito la descrizione dettagliata di ogni funzione utilizzata dal driver.

#### usage

Tale funzione viene chiamata ogni volta l'utente sbaglia ad invocare correttamente il driver. Il suo scopo è, appunto, spiegare all'utente come passare correttamente ogni parametro al driver per eseguire correttamente l'operazione richiesta. La sintassi prevede che, dopo aver digitato il nome del driver, il primo parametro obbligatorio da passare è l'indirizzo fisico della periferica, preceduto dall'opzione -g. Subito dopo, si sceglie se effettuare una lettura (digitare l'opzione -i), oppure se effettuare una scrittura sulla GPIO (opzione -o seguita dal valore che si vuole scrivere). La funzione specifica anche che il valore passato per essere scritto può essere rappresentato in notazione decimale, ottale o esadecimale.

```

14 /**
15  * @brief Describes how 'nodriver' must be used, specifying all supported
16  *        options.
17  * @param [in] name: Specifies nodriver name.
18  * @retval None.
19  */
20 void usage(char *name) {
21     printf("The right way to use this driver is\n");
22     printf(BOLDWHITE"%s -g <GPIO_ADDRESS> -i|-o <VALUE>\n"RESET, name);
23     printf("Please, pay attention:\n");
24     printf(BOLDWHITE"\t-g"RESET GREEN"\t<GPIO_ADDR>"RESET"\n\t\tThe mandatory
25             parameter <GPIO_ADDR>\n\t\tspecifies the physical address of GPIO.\n\t
26             \t<GPIO_ADDR> can be expressed in\n\t\tdecimal representation (no

```

```

        prefix), \n\t\toctal representation (0 prefix)\n\t\tor hexadecimal
        representation\n\t\t(0x or 0X prefix)\n");
24 printf(BOLDWHITE"\n\t-i"RESET"\n\t\tSpecifies the input value of GPIO at\n
    \t\t<GPIO_ADDR> physical address \n");
25 printf(BOLDWHITE"\n\t-o"RESET GREEN"\t<VALUE>"RESET"\n\t\tThe mandatory
    parameter <VALUE> specify\n\t\tthe value must be write on GPIO output
    \n");
26 return;
27 }

```

Codice 7.2: "nodriver function.h"

### parse\_command

Questa funzione ha lo scopo di analizzare tutti gli argomenti che vengono passati da riga di comando. Il meccanismo di parsing ruota attorno al corretto utilizzo della funzione **getopt**, già fornita dal SO. Quest'ultima prende in ingresso *argc*, *argv* e *optstring*. *optstring* è una stringa che contiene tutti gli argomenti che si vogliono ricercare, scritti tutti attaccato, e seguiti dal carattere ':' se tali argomenti devono essere seguiti da un attributo obbligatorio. Quando una delle stringhe, presenti in *argv*, comincia per il carattere '-', la funzione *getopt* restituisce il carattere immediatamente successivo. Ai fini implementativi, vengono dichiarate le seguenti variabili locali:

- *index*, contiene il valore di ritorno della funzione *getopt*. Se un particolare argomento necessita di un argomento obbligatorio e questo non viene passato, la funzione ritornerà il carattere ':'. Se, invece, viene letto un carattere non presente in *optstring*, la funzione ritorna il carattere '?'.
- *mandatory\_opt*, utilizzata per tener traccia del fatto che sia stato analizzato o meno il parametro obbligatorio 'g'.
- *i\_or\_o*, utilizzata per tener traccia del fatto che sia stata richiesta almeno una delle due operazioni possibili, lettura o scrittura.
- *optstring*, contiene tutte gli argomenti che può riconoscere la funzione.

Se nel while viene riconosciuta l'opzione

- g, viene modificata la variabile *mandatory\_opt* e salvato l'indirizzo fisico della GPIO in *address*. Qui viene fatto anche un controllo se l'indirizzo fisico passato è diverso da 0x0.
- i, si effettua un controllo per capire se precedentemente è stata riconosciuta l'opzione obbligatoria 'g'. In caso affermativo si setta la variabile *r\_w* a READ e si indica che nel lanciare il driver è stata richiesta almeno una tra le opzioni di lettura o scrittura.
- o, si eseguono le stesse operazioni per l'argomento 'i', con la sola differenza del settaggio della variabile *r\_w* a WRITE.
- h, viene richiamata la funzione *usage*.
- :, vuol dire che non è stato passato un attributo ad un certo argomento. Viene, pertanto, richiamata la funzione *usage* per aiutare l'utente.

- ?, vuol dire che l'argomento passato non è riconosciuto dal programma. Pertanto, dopo un messaggio di errore, ancora una volta viene richiamata la funzione usage.

Alla fine del while, è stato inserito un ulteriore controllo per verificare se nel lancio del driver è stata richiesta almeno una delle 2 operazioni possibili su GPIO.

In sostanza, la funzione parse\_command ritorna come valore

- -1 se si è verificato un qualsiasi errore;
- 1 se è stata richiamata la funzione usage;
- 0 se il driver è stato lanciato rispettando la sintassi prevista dal programmatore.

```

29 /**
30  * @brief Parses nodriver arguments.
31  * @param [in] argc: number of parameters, passed to main function.
32  * @param [in] argv: parameters passed to main function
33  * @param [out] address: physical address of GPIO component
34  * @param [out] r_w: specifies if the operation on GPIO is a read or a
35  *                  write
36  * @param [out] value: contains the read value (if the operation on GPIO
37  *                   is a read)
38  *                   or the value to write (if the operation on GPIO is a write)
39  * @retval Integer status:
40  *         -1     An error occurred;
41  *         1     Help function is called;
42  *         0     Everything is ok.
43 */
44 int parse_command(int argc, char **argv, int* address, int* r_w, int* value) {
45     int index=0;
46     int mandatory_opt=-1; /* Keep track if the mandatory option 'g' is parsed
47     */
48     int i_or_o=0; /* Keep track if an i/o operation is requested */
49     static char *optstring = ":g:io:h";
50
51     while((index = getopt(argc, argv, optstring)) != -1) {
52         switch(index) {
53             case 'g':
54                 mandatory_opt=atoi(optarg); /* Change mandatory_opt value to
55                 memorize that the mandatory option is parsed */
56                 *address=strtoul(optarg, NULL, 0);
57                 /* Check if address passed is not 0x0 */
58                 if(*address==0){
59                     printf("Wrong physical address inserted!\n");
60                     return -1;
61                 }
62                 break;
63             case 'i':
64                 /* Check if before 'i' option was passed the mandatory option 'g'
65                 */
66         }
67     }
68 }
```

```

61     if (mandatory_opt!=-1) {
62         *r_w=READ; /* Set READ operation for GPIO */
63         i_or_o=1;
64     }
65     else{
66         printf("Missing mandatory parameter 'g'\n");
67         usage(argv[0]);
68         return -1;
69     }
70     break;
71 case 'o':
72     /* Check if before 'o' option was passed the mandatory option 'g'
73      */
74     if (mandatory_opt!=-1) {
75         *r_w=WRITE; /* Set WRITE operation for GPIO */
76         *value=strtoul(optarg, NULL, 0);
77         i_or_o=1;
78     }
79     else{
80         printf("Missing mandatory parameter 'g'\n");
81         usage(argv[0]);
82         return -1;
83     }
84     break;
85 case 'h':
86     usage(argv[0]);
87     return 1;
88     break;
89 case ':':
90     printf("Missing argument for '%c' option\n", optopt);
91     usage(argv[0]);
92     return -1;
93     break;
94 case '?':
95     printf("Option '%c' not recognized\n", optopt);
96     usage(argv[0]);
97     return -1;
98     break;
99 default:
100     usage(argv[0]);
101     return -1;
102     break;
103 }
104 }
105 /* Check if an i/o operation is requested */
106 if(i_or_o==0){
107     printf("Can't use \"BOLDWHITE\"nodriver \"RESET\"without specify i/o option\
n");

```

```

108     usage(argv[0]);
109     return -1;
110 }
111 return 0;
112 }
```

Codice 7.3: "nodriver function.h"

**open\_memory**

In questa funzione viene eseguita tutta la magia per poter accedere correttamente ai registri della GPIO. Poichè il SO utilizza il meccanismo della virtualizzazione se si provasse ad accedere alla periferica direttamente con l'indirizzo fisico passato in ingresso al driver come attributo, molto probabilmente il SO restituirebbe un segmentation fault. Quello che bisogna fare è accedere ai registri tramite indirizzo virtuale, visto che è l'unico modo con cui è possibile comunicare con qualsiasi periferica. Il SO astrae, infatti, tutta la memoria. La funzione che consente di avere un indirizzo virtuale, dato un certo indirizzo fisico, è la system call **mmap**.

La prima cosa che si fa è accedere alla memoria tramite la system call **open**, andando a specificare di voler accedere sia in lettura sia scrittura. La memoria sotto Linux, viene vista come un qualsiasi file, la si accede tramite */dev/mem*. Fatto ciò, attraverso un meccanismo di mascherazione, dall'indirizzo fisico viene estratto sia l'indirizzo fisico della pagina a cui è mappata la periferica, sia il suo offset (che poi è lo stesso anche per la pagina virtuale).

A questo punto, viene richiamata la **mmap**. Essendo il primo parametro NULL, allora è il Kernel a scegliere l'indirizzo a cui effettuare il mapping. Il secondo parametro, **PAGE\_SIZE**, specifica la grandezza delle pagine viste dal SO; **PROT\_READ|PROT\_WRITE** indica che le pagine può essere letta e scritta. Il terzo parametro, **MAP\_SHARED**, indica che le modifiche fatte al file sono condivise con gli altri processi mappati sulla stessa regione. *fd* è il descrittore che individua il file registri periferica; *page\_addr* è, invece, l'indirizzo fisico della pagina in cui si trova la periferica. Da qui, si capisce che la **mmap** accetta come indirizzo fisico solo un multiplo delle pagine utilizzate dal SO.

```

114 /**
115 * @brief Opens /dev/mem file in order to access GPIO address and
116 *        calculates page offset and virtual address of GPIO file.
117 * @param [out] fd: file descriptor name.
118 * @param [in] phy_address: GPIO's physical address.
119 * @param [out] page_offset: page offset which GPIO finds.
120 * @param [out] virtual_address: GPIO's virtual address.
121 * @retval Integer status:
122 *         -1      An error occurred.
123 */
124 int open_memory(int* fd, int phy_address, int* page_offset, void** virtual_address) {
125     if (*fd < 1) {
126         perror("Error to open /dev/mem file");
127         return -1;
128     }
129     int page_addr = phy_address & MASK_SIZE;
130     /* Calculates page offset */
```

```

131     *page_offset = phy_address - page_addr;
132     /* mmap system call returns virtual address of GPIO */
133     *virtual_address = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE,
134                             MAP_SHARED, *fd, page_addr);
135     return 0;
}

```

Codice 7.4: "nodriver function.h"

**read\_gpio**

Legge il valore contenuto nella GPIO e restituisce un messaggio di avvenuta lettura. Si noti, che per eseguire correttamente l'operazione è stato aggiunto l'offset all'indirizzo virtuale che si ha a disposizione.

```

137 /**
138  * @brief Reads the <value> in input to GPIO port.
139  * @param [out] value: read value.
140  * @param [in] virtual_address: GPIO's virtual address.
141  * @param [in] page_offset: page offset which GPIO finds.
142  * @retval None.
143 */
144 void read_gpio(int* value, void* virtual_address, int page_offset) {
145     *value = *((unsigned*)(virtual_address + page_offset));
146     printf("The value on GPIO input is: %08x\n", *value);
147 }

```

Codice 7.5: "nodriver function.h"

**write\_gpio**

Effettua la scrittura del valore passato al driver sulla periferica GPIO, stampando un messaggio di avvenuta scrittura.

```

149 /**
150  * @brief Writes <value> in output to GPIO port.
151  * @param [in] value: write value.
152  * @param [in] phy_address: GPIO's physical address.
153  * @param [in] virtual_address: GPIO's virtual address.
154  * @param [in] page_offset: page offset which GPIO finds.
155  * @retval None.
156 */
157 void write_gpio(int value, int phy_address, void* virtual_address, int
158                 page_offset) {
159     printf("Going to write onto %08x the value %08x\n", phy_address, value );
160     *((unsigned*)(virtual_address + page_offset)) = value;
}

```

Codice 7.6: "nodriver function.h"

**close\_memory**

Chiude la memoria, aperta precedentemente, invocando la system call munmap, alla quale viene passato l'indirizzo virtuale del file aperto e la grandezza delle pagine. Infine, si richiama la close per chiudere il descrittore di file.

```

162 /**
163 * @brief Closes /dev/mem file.
164 * @param [in] fd: file descriptor name.
165 * @param [in] virtual_address: GPIO's virtual address.
166 * @retval None.
167 */
168 void close_memory(int fd, void* virtual_address) {
169     munmap(virtual_address, PAGE_SIZE);
170     close(fd);
171 }

```

Codice 7.7: "nodriver function.h"

### 7.2.1.3 main.c

La funzione main è molto semplice. Dopo aver dichiarato una serie di variabili da passare alle funzioni, la prima operazione eseguita è verificare che la funzione adibita al parsing degli argomenti non restituisca nessun errore. Dopodichè si controlla che l'accesso in memoria non provochi errore. In funzione del valore della variabile *r\_or\_w*, quindi, viene richiamata la funzione di lettura o di scrittura. Infine, c'è la chiamata alla close\_memory.

```

1 /**
2 ****
3 * @file      main.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     22-June-2017
8 * @brief    driver "nodriver" to control gpio
9 ****
10 */
11 /* Includes
12 -----*/
13 #include "nodriver_header.h"
14
15 int main(int argc, char *argv[]){
16     /*!< First step */
17     /*! Variables Declaration */
18     int file_descriptor; /* Memory file descriptor */
19     int r_w=READ; /* Saves operation's type to do with GPIO */
20     int r_or_w_value = 0; /* Saves read/write value from/on GPIO */
21     int ret_parse; /* Returned value from parse_command function */
22
23     unsigned phy_addr, page_off; /* Respectively physical address of GPIO and
24                                 page offset */

```

```

23 void *virt_addr; /* Virtual address of GPIO */
24 /*!< Second step */
25 /*! parse_command function is called to parse all argument passed to
   driver nodriver */
26 /* Check if parse_command function returns error */
27 if((ret_parse=parse_command(argc,argv,&phy_addr,&r_w,&r_or_w_value))==1)
28   return 0;
29 else
30   if(ret_parse== -1)
31     return -1;
32 /*!< Third step */
33 /*! open_memory function is called to achieve gpio virtual address */
34 /* Check if open_memory function returns error */
35 if(open_memory(&file_descriptor,phy_addr,&page_off,&virt_addr)==-1){
36   printf("nodriver aborted!\n");
37   return -1;
38 }
39 /**
40 /*!< Fourth step*/
41 /*! read_gpio is called if r_w variable is READ, otherwise is called
   write_gpio function */
42 if (r_w == READ) read_gpio(&r_or_w_value,virt_addr,page_off);
43 else write_gpio(r_or_w_value,phy_addr,virt_addr,page_off);
44 /*!< First step */
45 /*! close_memory function is called to close memory file and delete file
   descriptor */
46 close_memory(file_descriptor,virt_addr);
47
48 return 0;
49 }
```

Codice 7.8: "main.c"

#### 7.2.1.4 makefile

Per completezza, si riporta anche un semplice makefile realizzato per agevolare la fase di compilazione del driver, utilizzando la primitiva make.

```

1 nodriver : main.o nodriver_function.o
2   cc -o nodriver main.o nodriver_function.o
3
4 main.o : main.c nodriver_function.c nodriver_header.h
5   cc -c main.c -o main.o
6
7 nodriver_function.o : nodriver_function.c nodriver_header.h
8   cc -c nodriver_function.c -o nodriver_function.o
9
10 clean :
11   rm -f nodriver
```

```

12    rm *.o
13    echo "Clean all file!"

```

Codice 7.9: "makefile"

### 7.2.1.5 Esempio di esecuzione

In questa sezione viene mostrato come eseguire il driver appena scritto per pilotare led, switch e button della board Zybo. Per la buona riuscita di questa parte esercitativa, si precisa che è necessario essere riusciti a completare il booting da SD card, descritto nel capitolo precedente.

Per prima cosa, all'interno della partizione BOOT\_F della SD card, si crea una cartella *driver* al cui interno si andranno a mettere le 3 tipologie di driver richieste dalla traccia. Collegare la scheda al computer, aprire un terminale ed eseguire

```
mkdir /media/daniele/BOOT_F	driver
```

digitare, infine, i comandi

```

mkdir /media/daniele/BOOT_F	driver/nodriver
mkdir /media/daniele/BOOT_F	driver/uio
mkdir /media/daniele/BOOT_F	driver/uio_int

```

per creare rispettivamente le cartelle in cui vanno inseriti il primo, secondo e terzo tipo di driver.

Si copiano, quindi, i file scritti all'interno della cartella corrispondente. Nel caso considerato il comando è

```
cp -a Sistemi_EMBEDDED/Esercitazione_7/Driver/0_noDriver/ .
      /media/daniele/BOOT_F	driver/nodriver/
```

Eseguire l'operazione di umount delle partizioni

```

umount /media/daniele/BOOT_F
umount /media/daniele/ROOT_FS

```

Seguendo le indicazioni del precedente capitolo, si setta la board in modo tale da essere avviata via SD. Si apre un emulatore di terminale e si fa il reboot del SO.

Per prima, cosa si monta la partizione BOOT\_F in */mnt*

```
mount /dev/mmcblk0p1 /mnt/
```

Con i comandi seguenti, si accede quindi alla cartella nella quale è stato caricato il driver

```
cd /mnt/Driver/nodriver
```

digitando *ls* il risultato dovrebbe essere

```
main.c makefile nodriver_function.c nodriver_header.h
```

Compiliamo il driver con la primitiva

```
make
```

Per passare i giusti parametri al driver, bisogna conoscere gli esatti indirizzi delle periferiche GPIO che si vogliono pilotare. Di seguito, si suppone di voler pilotare button, switch e led della board. Per la sintesi hardware, si rimanda alla descrizione fatta nel capitolo precedente, con l'unica eccezione di dover istanziare anzichè un'unica periferica, tre periferiche. Per conoscere l'esatto indirizzo a cui sono mappate, sempre in Vivado, aperto il block diagram, si apre la paletta **Address Editor**, fig. 7.1.

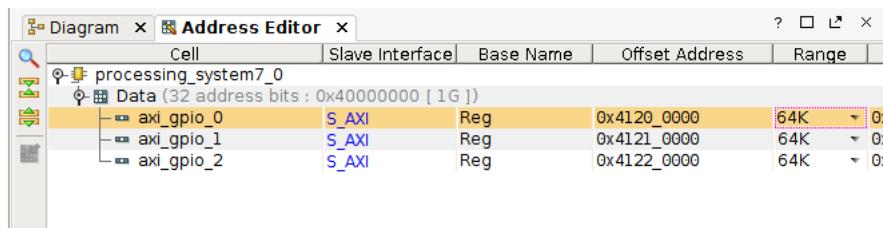


Figura 7.1: Address Editor

Dove

- `axi_gpio_0` è la GPIO associata ai button;
- `axi_gpio_1` è la GPIO associata agli switch;
- `axi_gpio_2` è la GPIO associata ai led.

Nelle figure che seguono si riportano alcuni esempi di funzionamento, in particolare si mostra:

- una lettura sui button, fig. 7.2;
- una lettura sugli switch, fig. 7.3;
- una scrittura sui led, fig. 7.4;
- il richiamo della funzione di help, fig. 7.5.

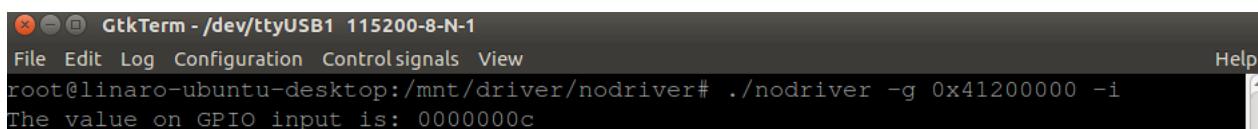


Figura 7.2: Messaggio restituito dopo la pressione di BTN2(V16) e BTN3(Y16)



Figura 7.3: Messaggio restituito dopo aver alzato SW0(G15) e SW2(W13)

```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@linaro-ubuntu-desktop:/mnt/driver/nodriver# ./nodriver -g 0x41220000 -o 0x7
Going to write onto 41220000 the value 00000007
```

Figura 7.4: Messaggio restituito dopo che sulla board si sono accesi LD0(M14), LD1(M15) e LD2(G14)

```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@linaro-ubuntu-desktop:/mnt/driver/nodriver# ./nodriver -h
The right way to use this driver is
./nodriver -g <GPIO_ADDRESS> -i|-o <VALUE>
Please, pay attention:
  -g      <GPIO_ADDR>
          The mandatory parameter <GPIO_ADDR>
          specifies the physical address of GPIO.
          <GPIO_ADDR> can be expressed in
          decimal representation (no prefix),
          octal representation (0 prefix)
          or hexadecimal representation
          (0x or 0X prefix)

  -i
          Specifies the input value of GPIO at
          <GPIO_ADDR> physical address

  -o      <VALUE>
          The mandatory parameter <VALUE> specify
          the value must be write on GPIO output
```

Figura 7.5: Usage function

### 7.2.2 Seconda tipologia

Questa seconda tipologia di driver prevede che l’utente non debba conoscere necessariamente l’indirizzo fisico della periferica GPIO a cui si vuole accedere. Infatti, poichè la periferica GPIO viene riconosciuta come un generico device uio, l’accesso ad essa è effettuato attraverso il file associato al device uio della GPIO, `/dev/uiox`. Si precisa che il driver di seguito sviluppato non fa utilizzo del meccanismo delle interruzioni.

Per garantire il funzionamento del driver è importante eseguire 2 cose:

1. Configurazione corretta del Kernel;
2. Modifica opportuna del device tree source.

Per ottemperare la prima delle due richieste, bisogna assicurarsi di aver abilitato l’utilizzo degli user space i/o driver all’interno della configurazione del Kernel. Per far ciò all’interno della cartella di lavoro, creata nel capitolo precedente, portarsi nella sottocartella contenente il kernel Linux

```
cd Linux-Digilent-Dev/
```

e digitare

```
make menuconfig
```

Nel menu di configurazione che si apre, accedere a **Device Drivers → Uerspace I/O drivers** e assicurarsi di settare il kernel così come mostrato in fig. 7.6.

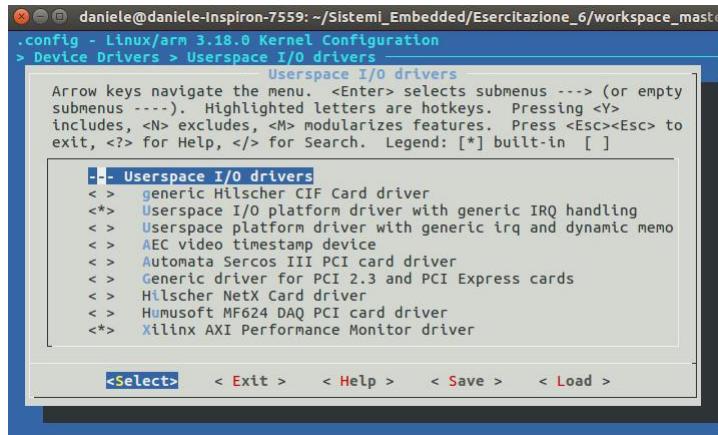


Figura 7.6: Userspace I/O drivers

Inoltre, per via di un bug nella versione 3.18 del kernel Linux, bisogna entrare nella cartella *Linux-Diligent-Dev/drivers/uio*

```
cd drivers/uio/
```

e dopo aver aperto il file **uio\_pdrv\_genirq.c**

```
gedit uio_pdrv_genirq.c
```

assicurarsi che alla riga 257 sia presente la stringa

```
1 { .compatible = "generic-uio", },
```

come evidenziato in fig. 7.7.

```
249 static const struct dev_pm_ops uio_pdrv_genirq_dev_pm_ops = {
250     .runtime_suspend = uio_pdrv_genirq_runtime_nop,
251     .runtime_resume = uio_pdrv_genirq_runtime_nop,
252 };
253
254 #ifdef CONFIG_OF
255 static struct of_device_id uio_of_genirq_match[] = {
256     { /* This is filled with module_param */ },
257     { .compatible = "generic-uio", },
258     { /* Sentinel */ },
259 };
260 MODULE_DEVICE_TABLE(of, uio_of_genirq_match);
261 module_param_string(of_id, uio_of_genirq_match[0].compatible, 128, 0);
262 MODULE_PARM_DESC(of_id, "Openfirmware id of the device to be handled by uio");
263 #endif
264
265 static struct platform_driver uio_pdrv_genirq = {
266     .probe = uio_pdrv_genirq_probe,
267     .remove = uio_pdrv_genirq_remove,
268     .driver = {
```

Figura 7.7: uio\_pdrv\_genirq.c

Per quanto riguarda invece la seconda richiesta, si fa utilizzo del tool SDK.

Prima di mostrare come operare, si fa una piccola precisazione. Nella prima tipologia di driver si utilizza direttamente l'indirizzo fisico della periferica, pertanto, se in una periferica GPIO Xilinx vengono sfruttati entrambi i canali, per accedervi basta fornire l'indirizzo del primo o secondo canale, in base a ciò che si è interessati. Poiché con questa seconda tipologia non si ha la possibilità di fornire un indirizzo fisico preciso, ma l'accesso alla periferica avviene solo tramite un indirizzo virtuale che punta alla prima locazione della periferica, l'accesso al primo o secondo canale deve essere fatto a livello driver. Il driver, che di seguito viene proposto, prevede, appunto, un meccanismo per garantire ciò. Per mostrare il completo funzionamento di esso, anziché utilizzare lo stesso progetto hardware del precedente driver (3 GPIO diverse: una per i bottoni, una per gli switch e l'altra per i led), si crea un altro progetto, implementando solo 2 GPIO Xilinx. Sulla prima vengono mappati bottoni e switch, rispettivamente canale 1 e 2 della periferica; sulla seconda, invece, vengono mappati i led della board. Per il procedimento, basta seguire le indicazioni fornite nel precedente capitolo e adattarle opportunamente alle specifiche di sopra.

Al termine della sintesi della parte hardware il block design dovrebbe essere come in fig. 7.8.

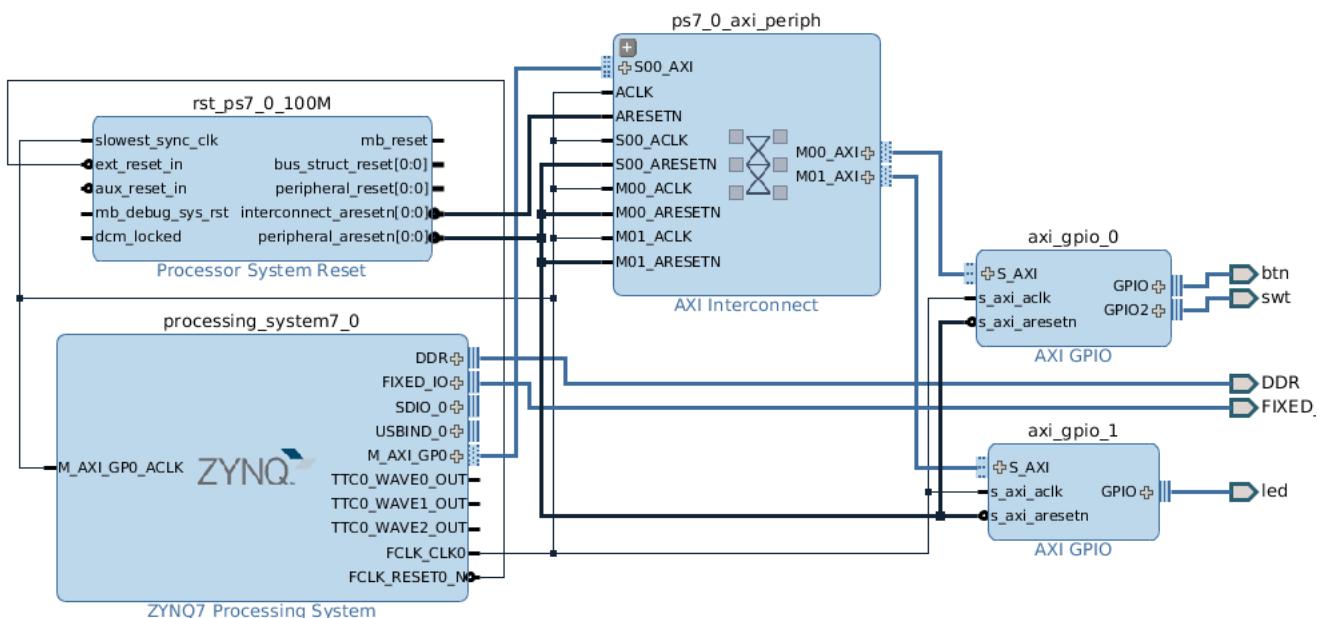


Figura 7.8: Block Design

Come da prassi, si effettua nell'ordine il wrapper del componente, la sintesi, l'implementazione, il mapping dei porti, la generazione del bit stream e l'export del progetto, includendo il bitstream. A questo punto si lancia finalmente SDK.

Seguendo le indicazioni del capitolo precedente si crea prima il FSBL, poi il BOOT.bin e, infine, si fa generare al tool il device tree source.

Aprire il file **system-top.dts** e modificarlo come in fig. 7.9. Si noti che, rispetto al capitolo precedente, bisogna aggiungere il parametro evidenziato a causa del bug del kernel precedentemente citato.

```

1 /*
2 * CAUTION: This file is automatically generated by Xilinx.
3 * Version:
4 * Today is: Sun Jul  2 18:25:27 2017
5 */
6
7
8 /dts-v1;
9 /include/ "zynq-7000.dtsi"
10 /include/ "pl.dtsi"
11 /include/ "pcw.dtsi"
12 {
13     chosen {
14         bootargs = console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk
15             rootfstype=ext4 rootwait uio_pdrv_genirq.of_id=generic-uio";
16     };
17     aliases {
18         ethernet0 = &gem0;
19         serial0 = &uart1;
20         spi0 = &qspi;
21     };
22     memory {
23         device_type = "memory";
24         reg = <0x0 0x20000000>;
25     };
26     cpus {};
27 };

```

Figura 7.9: system-top.dts

Modificare anche il file **pl.dts**, specificando nei campi *compatible* delle due periferiche, `axi_gpio_0` e `axi_gpio_1`, la stringa “generic-uio”, fig. 7.10.

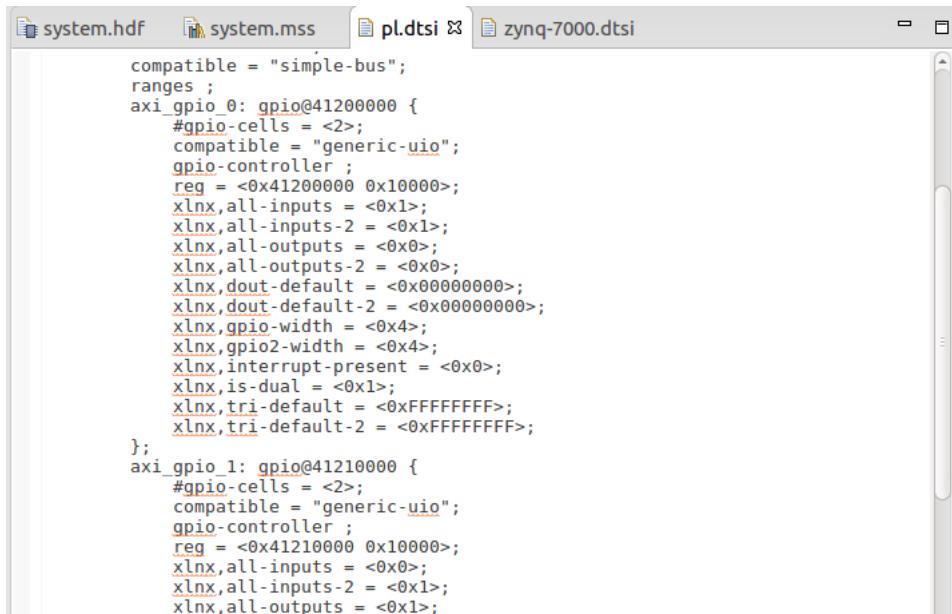


Figura 7.10: pl.dtsi

Arrivati a questo punto, si può finalmente generare il device tree blob da riga di comando, compilando il file `system-top.dts`.

Completata questa parte di configurazione, si descrivono adesso le principali differenze di questo driver rispetto a quello presentato precedentemente.

### 7.2.2.1 uiодriver\_header.h

Nell'header file si notano le macro relative agli offset dei registri GPIO\_DATA e GPIO\_TRI del primo canale e dei registri GPIO2\_DATA e GPIO2\_TRI del secondo canale della periferica AXI GPIO Xilinx. Inoltre, vengono definite 2 macro utilizzate proprio per tener traccia di quale dei due canali della periferica si vuole utilizzare. Per il resto, i prototipi delle funzioni sono gli stessi, eccetto qualche parametro aggiuntivo necessario per la gestione dei canali.

```

1  /**
2  ****
3
4  * @file      uiодriver_header.h
5  * @author    Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
6  *            Sistemi Embedded 2016-2017
7  * @version   V1.0
8  * @date     24-June-2017
9  * @brief    library 'uiодriver' for GPIO
10
11 */
12
13 /* Includes
14  -----
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <unistd.h>
18 #include <sys/mman.h>
19 #include <fcntl.h>
20 #include <errno.h>
21
22 /* Colour MACRO
23  -----
24 #define RESET    "\033[0m"      /*!< Reset colour */
25 #define GREEN    "\033[32m"      /*!< Green colour */
26 #define BOLDWHITE "\033[1m\033[37m" /*!< Bold White colour */
27
28 /* GPIO operation MACRO
29  -----
30 #define READ 0           /*!< Read operation is set */
31 #define WRITE 1          /*!< Write operation is set */
32
33 /* GPIO page MACRO
34  -----
35 #define GPIO_MAP_SIZE      0x10000  /*!< Bit mask to extract page address,
36                                     without offset */
37 #define GPIO_DATA_OFFSET    0x00    /*!< Offset of GPIO_DATA register */
38 #define GPIO_TRI_OFFSET     0x04    /*!< Offset of GPIO_TRI register */
39 #define GPIO2_DATA_OFFSET   0x08    /*!< Offset of GPIO2_DATA register */
40 #define GPIO2_TRI_OFFSET    0x0C    /*!< Offset of GPIO2_TRI register */

```

```

34  /*
35   * GPIO channel MACRO
36   *-----*/
36 #define GPIO_CHANNEL_1 1 /*!< GPIO channel 1 selected */
37 #define GPIO_CHANNEL_2 2 /*!< GPIO channel 2 selected */
38
39 /* Function prototypes
40  *-----*/
40 void usage(char *name); /*!< usage function is called to help the user */
41 int parse_command(int argc,char **argv,char** uiiod,int* channel,int* r_w,int
42 * value); /*!< parse_command function is called to parse arguments passed
43 to driver */
42 int open_device(int* fd, char* uiiod, void** virtual_address); /*!<
44 open_device function is called to open file associated to uio device */
43 void read_gpio(int* value,void* virtual_address, int channel); /*!< read_gpio
44 function is called to do read operation from GPIO */
44 void write_gpio(int value,char* uiiod,void* virtual_address, int channel); /*!
45 < write_gpio function is called to do write operation on GPIO */
45 void close_device(int fd, void* virtual_address); /*!< close_device function
46 is called to close uio file */

```

Codice 7.10: "uiodriver header.h"

### 7.2.2.2 uiodriver\_function.h

#### usage

La funzione resta pressappoco invariata, se non per il fatto che è stata aggiunta una nuova stringa da stampare a video per spiegare il corretto utilizzo del parametro 'c' per la selezione del canale da selezionare.

```

14 /**
15  * @brief Describes how 'uiodriver' must be used, specifying all supported
16  * options.
17  * @param [in] name: Specifies uiodriver name.
18  * @retval None.
19 */
20 void usage(char *name) {
21     printf("The right way to use this driver is\n");
22     printf(BOLDWHITE"%s -d <UIO_DEV_FILE> -c <CHANNEL> -i|-o <VALUE>\n"RESET,
23            name);
24     printf("Please, pay attention:\n");
25     printf(BOLDWHITE"\t-d"RESET GREEN"\t<UIO_DEV_FILE>"RESET"\n\t\tThe
26           mandatory parameter <UIO_DEV_FILE>\n\t\ttspecifies the uio device file
27           corrisponding\n\t\tto GPIO, for example /dev/uio0.\n");
28     printf(BOLDWHITE"\n\t-c"RESET GREEN"\t<CHANNEL>"RESET"\n\t\tThe mandatory
29           parameter <CHANNEL> specify\n\t\tthe GPIO channel to use \n\t\tThis
30           number must be 1 or 2");
31     printf(BOLDWHITE"\n\t-i"RESET"\n\t\tSpecifies the input value of GPIO \n\t
32           \tcorrisponding to <UIO_DEV_FILE> \n");

```

```

26     printf(BOLDWHITE"\n\t-o"RESET GREEN"\t<VALUE>"RESET"\n\t\tThe mandatory
27         parameter <VALUE> specify\n\t\tthe value must be write on GPIO output
28         \n");
      return;
}

```

Codice 7.11: "uiodriver function.h"

**parse\_command**

Questa volta, all'interno della funzione, vengono dichiarate 2 variabili locali **man\_opt\_d** e **man\_opt\_c**. Il loro scopo è di memorizzare se i parametri obbligatori 'd' e 'c' sono stati forniti in ingresso al driver. Ecco, allora, che se viene riconosciuto l'argomento 'c', viene controllato se precedentemente è stato fatto il parsing di 'd'. Meccanismo analogo viene replicato nei case 'i' e 'o' per controllare se precedentemente è stato fatto il parsing sia dell'argomento 'd', sia di quello 'c'. L'attributo che deve seguire l'argomento 'c' stabilisce quale dei due canali utilizzare, se il primo (1) o il secondo (2). Al termine della funzione oltre a controllare se è stata richiesta una delle due operazioni di lettura e scrittura, viene controllato anche se il numero fornito per selezionare il canale è corretto. Si noti, infine, che l'attributo di 'd' non è più un indirizzo fisico, ma il path dove si trova il file corrispondente al device uio.

```

30 /**
31 * @brief Parses uiodriver arguments.
32 * @param [in] argc: number of parameters, passed to main function.
33 * @param [in] argv: parameters passed to main function
34 * @param [out] uiiod: uio device file corresponding to GPIO
35 * @param [out] channel: number corresponding to GPIO channel
36 * @param [out] r_w: specifies if the operation on GPIO is a read or a
37     write
38 * @param [out] value: contains the read value (if the operation on GPIO
39     is a read)
40     or the value to write (if the operation on GPIO is a write)
41 * @retval Integer status:
42     -1 An error occurred;
43     1 Help function is called;
44     0 Everything is ok.
45 */
46 int parse_command(int argc,char **argv,char** uiiod,int* channel,int* r_w,int
47     * value){
48     int index=0;
49     int man_opt_d=-1; /* Keep track if the mandatory option 'd' is parsed */
50     int man_opt_c=-1; /* Keep track if the mandatory option 'c' is parsed */
51     int i_or_o=0; /* Keep track if an i/o operation is requested */
52     static char *optstring = ":d:c:io:h";
53
54     while((index = getopt(argc, argv, optstring)) != -1) {
55         switch(index) {
56             case 'd':
57                 man_opt_d=atoi(optarg); /* Change mand_opt_d value to memorize
58                     that the option 'd' is parsed */
59                 *uiiod=optarg;
60
61             case 'c':
62                 man_opt_c=atoi(optarg); /* Change mand_opt_c value to memorize
63                     that the option 'c' is parsed */
64                 *channel=optarg;
65
66             case 'i':
67                 i_or_o=1;
68
69             case 'o':
70                 i_or_o=2;
71
72             case 'r':
73                 r_w=0;
74
75             case 'w':
76                 r_w=1;
77
78             case 'h':
79                 help();
80                 exit(0);
81
82             default:
83                 fprintf(stderr,"Unknown option %c\n",optopt);
84                 exit(1);
85         }
86     }
87
88     if(man_opt_d==1 || man_opt_d==2) {
89         if(man_opt_c==1 || man_opt_c==2) {
90             if(i_or_o==1) {
91                 if(r_w==0) {
92                     uiiod=uiofd[0];
93                 } else {
94                     uiiod=uiofd[1];
95                 }
96             } else {
97                 if(r_w==0) {
98                     uiiod=uiofd[1];
99                 } else {
100                     uiiod=uiofd[0];
101                 }
102             }
103         } else {
104             if(i_or_o==1) {
105                 if(r_w==0) {
106                     uiiod=uiofd[0];
107                 } else {
108                     uiiod=uiofd[1];
109                 }
110             } else {
111                 if(r_w==0) {
112                     uiiod=uiofd[1];
113                 } else {
114                     uiiod=uiofd[0];
115                 }
116             }
117         }
118     }
119
120     if(uiiod==0) {
121         perror("Error opening uio device file");
122         exit(1);
123     }
124
125     if(channel==0) {
126         perror("Error opening channel");
127         exit(1);
128     }
129
130     if(r_w==0) {
131         if(fopen(uiiod,"r")==NULL) {
132             perror("Error opening file");
133             exit(1);
134         }
135     } else {
136         if(fopen(uiiod,"w")==NULL) {
137             perror("Error opening file");
138             exit(1);
139         }
140     }
141
142     if(value!=0) {
143         if(fread(value,1,4,stdin)==4) {
144             *value=*((int*)value);
145         } else {
146             perror("Error reading value");
147             exit(1);
148         }
149     }
150
151     if(r_w==1) {
152         if(fwrite(value,1,4,stdout)==4) {
153             *value=*((int*)value);
154         } else {
155             perror("Error writing value");
156             exit(1);
157         }
158     }
159
160     if(argc==1) {
161         if(help()==0) {
162             exit(1);
163         }
164     }
165
166     if(argc==2) {
167         if(help()==0) {
168             exit(1);
169         }
170     }
171
172     if(argc==3) {
173         if(help()==0) {
174             exit(1);
175         }
176     }
177
178     if(argc==4) {
179         if(help()==0) {
180             exit(1);
181         }
182     }
183
184     if(argc==5) {
185         if(help()==0) {
186             exit(1);
187         }
188     }
189
190     if(argc==6) {
191         if(help()==0) {
192             exit(1);
193         }
194     }
195
196     if(argc==7) {
197         if(help()==0) {
198             exit(1);
199         }
200     }
201
202     if(argc==8) {
203         if(help()==0) {
204             exit(1);
205         }
206     }
207
208     if(argc==9) {
209         if(help()==0) {
210             exit(1);
211         }
212     }
213
214     if(argc==10) {
215         if(help()==0) {
216             exit(1);
217         }
218     }
219
220     if(argc==11) {
221         if(help()==0) {
222             exit(1);
223         }
224     }
225
226     if(argc==12) {
227         if(help()==0) {
228             exit(1);
229         }
230     }
231
232     if(argc==13) {
233         if(help()==0) {
234             exit(1);
235         }
236     }
237
238     if(argc==14) {
239         if(help()==0) {
240             exit(1);
241         }
242     }
243
244     if(argc==15) {
245         if(help()==0) {
246             exit(1);
247         }
248     }
249
250     if(argc==16) {
251         if(help()==0) {
252             exit(1);
253         }
254     }
255
256     if(argc==17) {
257         if(help()==0) {
258             exit(1);
259         }
260     }
261
262     if(argc==18) {
263         if(help()==0) {
264             exit(1);
265         }
266     }
267
268     if(argc==19) {
269         if(help()==0) {
270             exit(1);
271         }
272     }
273
274     if(argc==20) {
275         if(help()==0) {
276             exit(1);
277         }
278     }
279
280     if(argc==21) {
281         if(help()==0) {
282             exit(1);
283         }
284     }
285
286     if(argc==22) {
287         if(help()==0) {
288             exit(1);
289         }
290     }
291
292     if(argc==23) {
293         if(help()==0) {
294             exit(1);
295         }
296     }
297
298     if(argc==24) {
299         if(help()==0) {
300             exit(1);
301         }
302     }
303
304     if(argc==25) {
305         if(help()==0) {
306             exit(1);
307         }
308     }
309
310     if(argc==26) {
311         if(help()==0) {
312             exit(1);
313         }
314     }
315
316     if(argc==27) {
317         if(help()==0) {
318             exit(1);
319         }
320     }
321
322     if(argc==28) {
323         if(help()==0) {
324             exit(1);
325         }
326     }
327
328     if(argc==29) {
329         if(help()==0) {
330             exit(1);
331         }
332     }
333
334     if(argc==30) {
335         if(help()==0) {
336             exit(1);
337         }
338     }
339
340     if(argc==31) {
341         if(help()==0) {
342             exit(1);
343         }
344     }
345
346     if(argc==32) {
347         if(help()==0) {
348             exit(1);
349         }
350     }
351
352     if(argc==33) {
353         if(help()==0) {
354             exit(1);
355         }
356     }
357
358     if(argc==34) {
359         if(help()==0) {
360             exit(1);
361         }
362     }
363
364     if(argc==35) {
365         if(help()==0) {
366             exit(1);
367         }
368     }
369
370     if(argc==36) {
371         if(help()==0) {
372             exit(1);
373         }
374     }
375
376     if(argc==37) {
377         if(help()==0) {
378             exit(1);
379         }
380     }
381
382     if(argc==38) {
383         if(help()==0) {
384             exit(1);
385         }
386     }
387
388     if(argc==39) {
389         if(help()==0) {
390             exit(1);
391         }
392     }
393
394     if(argc==40) {
395         if(help()==0) {
396             exit(1);
397         }
398     }
399
400     if(argc==41) {
401         if(help()==0) {
402             exit(1);
403         }
404     }
405
406     if(argc==42) {
407         if(help()==0) {
408             exit(1);
409         }
410     }
411
412     if(argc==43) {
413         if(help()==0) {
414             exit(1);
415         }
416     }
417
418     if(argc==44) {
419         if(help()==0) {
420             exit(1);
421         }
422     }
423
424     if(argc==45) {
425         if(help()==0) {
426             exit(1);
427         }
428     }
429
430     if(argc==46) {
431         if(help()==0) {
432             exit(1);
433         }
434     }
435
436     if(argc==47) {
437         if(help()==0) {
438             exit(1);
439         }
440     }
441
442     if(argc==48) {
443         if(help()==0) {
444             exit(1);
445         }
446     }
447
448     if(argc==49) {
449         if(help()==0) {
450             exit(1);
451         }
452     }
453
454     if(argc==50) {
455         if(help()==0) {
456             exit(1);
457         }
458     }
459
460     if(argc==51) {
461         if(help()==0) {
462             exit(1);
463         }
464     }
465
466     if(argc==52) {
467         if(help()==0) {
468             exit(1);
469         }
470     }
471
472     if(argc==53) {
473         if(help()==0) {
474             exit(1);
475         }
476     }
477
478     if(argc==54) {
479         if(help()==0) {
480             exit(1);
481         }
482     }
483
484     if(argc==55) {
485         if(help()==0) {
486             exit(1);
487         }
488     }
489
490     if(argc==56) {
491         if(help()==0) {
492             exit(1);
493         }
494     }
495
496     if(argc==57) {
497         if(help()==0) {
498             exit(1);
499         }
500     }
501
502     if(argc==58) {
503         if(help()==0) {
504             exit(1);
505         }
506     }
507
508     if(argc==59) {
509         if(help()==0) {
510             exit(1);
511         }
512     }
513
514     if(argc==60) {
515         if(help()==0) {
516             exit(1);
517         }
518     }
519
520     if(argc==61) {
521         if(help()==0) {
522             exit(1);
523         }
524     }
525
526     if(argc==62) {
527         if(help()==0) {
528             exit(1);
529         }
530     }
531
532     if(argc==63) {
533         if(help()==0) {
534             exit(1);
535         }
536     }
537
538     if(argc==64) {
539         if(help()==0) {
540             exit(1);
541         }
542     }
543
544     if(argc==65) {
545         if(help()==0) {
546             exit(1);
547         }
548     }
549
550     if(argc==66) {
551         if(help()==0) {
552             exit(1);
553         }
554     }
555
556     if(argc==67) {
557         if(help()==0) {
558             exit(1);
559         }
560     }
561
562     if(argc==68) {
563         if(help()==0) {
564             exit(1);
565         }
566     }
567
568     if(argc==69) {
569         if(help()==0) {
570             exit(1);
571         }
572     }
573
574     if(argc==70) {
575         if(help()==0) {
576             exit(1);
577         }
578     }
579
580     if(argc==71) {
581         if(help()==0) {
582             exit(1);
583         }
584     }
585
586     if(argc==72) {
587         if(help()==0) {
588             exit(1);
589         }
590     }
591
592     if(argc==73) {
593         if(help()==0) {
594             exit(1);
595         }
596     }
597
598     if(argc==74) {
599         if(help()==0) {
600             exit(1);
601         }
602     }
603
604     if(argc==75) {
605         if(help()==0) {
606             exit(1);
607         }
608     }
609
610     if(argc==76) {
611         if(help()==0) {
612             exit(1);
613         }
614     }
615
616     if(argc==77) {
617         if(help()==0) {
618             exit(1);
619         }
620     }
621
622     if(argc==78) {
623         if(help()==0) {
624             exit(1);
625         }
626     }
627
628     if(argc==79) {
629         if(help()==0) {
630             exit(1);
631         }
632     }
633
634     if(argc==80) {
635         if(help()==0) {
636             exit(1);
637         }
638     }
639
640     if(argc==81) {
641         if(help()==0) {
642             exit(1);
643         }
644     }
645
646     if(argc==82) {
647         if(help()==0) {
648             exit(1);
649         }
650     }
651
652     if(argc==83) {
653         if(help()==0) {
654             exit(1);
655         }
656     }
657
658     if(argc==84) {
659         if(help()==0) {
660             exit(1);
661         }
662     }
663
664     if(argc==85) {
665         if(help()==0) {
666             exit(1);
667         }
668     }
669
670     if(argc==86) {
671         if(help()==0) {
672             exit(1);
673         }
674     }
675
676     if(argc==87) {
677         if(help()==0) {
678             exit(1);
679         }
680     }
681
682     if(argc==88) {
683         if(help()==0) {
684             exit(1);
685         }
686     }
687
688     if(argc==89) {
689         if(help()==0) {
690             exit(1);
691         }
692     }
693
694     if(argc==90) {
695         if(help()==0) {
696             exit(1);
697         }
698     }
699
699
700     if(argc==91) {
701         if(help()==0) {
702             exit(1);
703         }
704     }
705
706     if(argc==92) {
707         if(help()==0) {
708             exit(1);
709         }
710     }
711
712     if(argc==93) {
713         if(help()==0) {
714             exit(1);
715         }
716     }
717
718     if(argc==94) {
719         if(help()==0) {
720             exit(1);
721         }
722     }
723
724     if(argc==95) {
725         if(help()==0) {
726             exit(1);
727         }
728     }
729
730     if(argc==96) {
731         if(help()==0) {
732             exit(1);
733         }
734     }
735
736     if(argc==97) {
737         if(help()==0) {
738             exit(1);
739         }
740     }
741
742     if(argc==98) {
743         if(help()==0) {
744             exit(1);
745         }
746     }
747
748     if(argc==99) {
749         if(help()==0) {
750             exit(1);
751         }
752     }
753
754     if(argc==100) {
755         if(help()==0) {
756             exit(1);
757         }
758     }
759
760     if(argc==101) {
761         if(help()==0) {
762             exit(1);
763         }
764     }
765
766     if(argc==102) {
767         if(help()==0) {
768             exit(1);
769         }
770     }
771
772     if(argc==103) {
773         if(help()==0) {
774             exit(1);
775         }
776     }
777
778     if(argc==104) {
779         if(help()==0) {
780             exit(1);
781         }
782     }
783
784     if(argc==105) {
785         if(help()==0) {
786             exit(1);
787         }
788     }
789
790     if(argc==106) {
791         if(help()==0) {
792             exit(1);
793         }
794     }
795
796     if(argc==107) {
797         if(help()==0) {
798             exit(1);
799         }
800     }
801
802     if(argc==108) {
803         if(help()==0) {
804             exit(1);
805         }
806     }
807
808     if(argc==109) {
809         if(help()==0) {
810             exit(1);
811         }
812     }
813
814     if(argc==110) {
815         if(help()==0) {
816             exit(1);
817         }
818     }
819
820     if(argc==111) {
821         if(help()==0) {
822             exit(1);
823         }
824     }
825
826     if(argc==112) {
827         if(help()==0) {
828             exit(1);
829         }
830     }
831
832     if(argc==113) {
833         if(help()==0) {
834             exit(1);
835         }
836     }
837
838     if(argc==114) {
839         if(help()==0) {
840             exit(1);
841         }
842     }
843
844     if(argc==115) {
845         if(help()==0) {
846             exit(1);
847         }
848     }
849
850     if(argc==116) {
851         if(help()==0) {
852             exit(1);
853         }
854     }
855
856     if(argc==117) {
857         if(help()==0) {
858             exit(1);
859         }
860     }
861
862     if(argc==118) {
863         if(help()==0) {
864             exit(1);
865         }
866     }
867
868     if(argc==119) {
869         if(help()==0) {
870             exit(1);
871         }
872     }
873
874     if(argc==120) {
875         if(help()==0) {
876             exit(1);
877         }
878     }
879
880     if(argc==121) {
881         if(help()==0) {
882             exit(1);
883         }
884     }
885
886     if(argc==122) {
887         if(help()==0) {
888             exit(1);
889         }
890     }
891
892     if(argc==123) {
893         if(help()==0) {
894             exit(1);
895         }
896     }
897
898     if(argc==124) {
899         if(help()==0) {
900             exit(1);
901         }
902     }
903
904     if(argc==125) {
905         if(help()==0) {
906             exit(1);
907         }
908     }
909
910     if(argc==126) {
911         if(help()==0) {
912             exit(1);
913         }
914     }
915
916     if(argc==127) {
917         if(help()==0) {
918             exit(1);
919         }
920     }
921
922     if(argc==128) {
923         if(help()==0) {
924             exit(1);
925         }
926     }
927
928     if(argc==129) {
929         if(help()==0) {
930             exit(1);
931         }
932     }
933
934     if(argc==130) {
935         if(help()==0) {
936             exit(1);
937         }
938     }
939
940     if(argc==131) {
941         if(help()==0) {
942             exit(1);
943         }
944     }
945
946     if(argc==132) {
947         if(help()==0) {
948             exit(1);
949         }
950     }
951
952     if(argc==133) {
953         if(help()==0) {
954             exit(1);
955         }
956     }
957
958     if(argc==134) {
959         if(help()==0) {
960             exit(1);
961         }
962     }
963
964     if(argc==135) {
965         if(help()==0) {
966             exit(1);
967         }
968     }
969
970     if(argc==136) {
971         if(help()==0) {
972             exit(1);
973         }
974     }
975
976     if(argc==137) {
977         if(help()==0) {
978             exit(1);
979         }
980     }
981
982     if(argc==138) {
983         if(help()==0) {
984             exit(1);
985         }
986     }
987
988     if(argc==139) {
989         if(help()==0) {
990             exit(1);
991         }
992     }
993
994     if(argc==140) {
995         if(help()==0) {
996             exit(1);
997         }
998     }
999
1000    if(argc==141) {
1001        if(help()==0) {
1002            exit(1);
1003        }
1004    }
1005
1006    if(argc==142) {
1007        if(help()==0) {
1008            exit(1);
1009        }
1010    }
1011
1012    if(argc==143) {
1013        if(help()==0) {
1014            exit(1);
1015        }
1016    }
1017
1018    if(argc==144) {
1019        if(help()==0) {
1020            exit(1);
1021        }
1022    }
1023
1024    if(argc==145) {
1025        if(help()==0) {
1026            exit(1);
1027        }
1028    }
1029
1030    if(argc==146) {
1031        if(help()==0) {
1032            exit(1);
1033        }
1034    }
1035
1036    if(argc==147) {
1037        if(help()==0) {
1038            exit(1);
1039        }
1040    }
1041
1042    if(argc==148) {
1043        if(help()==0) {
1044            exit(1);
1045        }
1046    }
1047
1048    if(argc==149) {
1049        if(help()==0) {
1050            exit(1);
1051        }
1052    }
1053
1054    if(argc==150) {
1055        if(help()==0) {
1056            exit(1);
1057        }
1058    }
1059
1060    if(argc==151) {
1061        if(help()==0) {
1062            exit(1);
1063        }
1064    }
1065
1066    if(argc==152) {
1067        if(help()==0) {
1068            exit(1);
1069        }
1070    }
1071
1072    if(argc==153) {
1073        if(help()==0) {
1074            exit(1);
1075        }
1076    }
1077
1078    if(argc==154) {
1079        if(help()==0) {
1080            exit(1);
1081        }
1082    }
1083
1084    if(argc==155) {
1085        if(help()==0) {
1086            exit(1);
1087        }
1088    }
1089
1090    if(argc==156) {
1091        if(help()==0) {
1092            exit(1);
1093        }
1094    }
1095
1096    if(argc==157) {
1097        if(help()==0) {
1098            exit(1);
1099        }
1100    }
1101
1102    if(argc==158) {
1103        if(help()==0) {
1104            exit(1);
1105        }
1106    }
1107
1108    if(argc==159) {
1109        if(help()==0) {
1110            exit(1);
1111        }
1112    }
1113
1114    if(argc==160) {
1115        if(help()==0) {
1116            exit(1);
1117        }
1118    }
1119
1120    if(argc==161) {
1121        if(help()==0) {
1122            exit(1);
1123        }
1124    }
1125
1126    if(argc==162) {
1127        if(help()==0) {
1128            exit(1);
1129        }
1130    }
1131
1132    if(argc==163) {
1133        if(help()==0) {
1134            exit(1);
1135        }
1136    }
1137
1138    if(argc==164) {
1139        if(help()==0) {
1140            exit(1);
1141        }
1142    }
1143
1144    if(argc==165) {
1145        if(help()==0) {
1146            exit(1);
1147        }
1148    }
1149
1150    if(argc==166) {
1151        if(help()==0) {
1152            exit(1);
1153        }
1154    }
1155
1156    if(argc==167) {
1157        if(help()==0) {
1158            exit(1);
1159        }
1160    }
1161
1162    if(argc==168) {
1163        if(help()==0) {
1164            exit(1);
1165        }
1166    }
1167
1168    if(argc==169) {
1169        if(help()==0) {
1170            exit(1);
1171        }
1172    }
1173
1174    if(argc==170) {
1175        if(help()==0) {
1176            exit(1);
1177        }
1178    }
1179
1180    if(argc==171) {
1181        if(help()==0) {
1182            exit(1);
1183        }
1184    }
1185
1186    if(argc==172) {
1187        if(help()==0) {
1188            exit(1);
1189        }
1190    }
1191
1192    if(argc==173) {
1193        if(help()==0) {
1194            exit(1);
1195        }
1196    }
1197
1198    if(argc==174) {
1199        if(help()==0) {
1200            exit(1);
1201        }
1202    }
1203
1204    if(argc==175) {
1205        if(help()==0) {
1206            exit(1);
1207        }
1208    }
1209
1210    if(argc==176) {
1211        if(help()==0) {
1212            exit(1);
1213        }
1214    }
1215
1216    if(argc==177) {
1217        if(help()==0) {
1218            exit(1);
1219        }
1220    }
1221
1222    if(argc==178) {
1223        if(help()==0) {
1224            exit(1);
1225        }
1226    }
1227
1228    if(argc==179) {
1229        if(help()==0) {
1230            exit(1);
1231        }
1232    }
1233
1234    if(argc==180) {
1235        if(help()==0) {
1236            exit(1);
1237        }
1238    }
1239
1240    if(argc==181) {
1241        if(help()==0) {
1242            exit(1);
1243        }
1244    }
1245
1246    if(argc==182) {
1247        if(help()==0) {
1248            exit(1);
1249        }
1250    }
1251
1252    if(argc==183) {
1253        if(help()==0) {
1254            exit(1);
1255        }
1256    }
1257
1258    if(argc==184) {
1259        if(help()==0) {
1260            exit(1);
1261        }
1262    }
1263
1264    if(argc==185) {
1265        if(help()==0) {
1266            exit(1);
1267        }
1268    }
1269
1270    if(argc==186) {
1271        if(help()==0) {
1272            exit(1);
1273        }
1274    }
1275
1276    if(argc==187) {
1277        if(help()==0) {
1278            exit(1);
1279        }
1280    }
1281
1282    if(argc==188) {
1283        if(help()==0) {
1284            exit(1);
1285        }
1286    }
1287
1288    if(argc==189) {
1289        if(help()==0) {
1290            exit(1);
1291        }
1292    }
1293
1294    if(argc==190) {
1295        if(help()==0) {
1296            exit(1);
1297        }
1298    }
1299
1300    if(argc==191) {
1301        if(help()==0) {
1302            exit(1);
1303        }
1304    }
1305
1306    if(argc==192) {
1307        if(help()==0) {
1308            exit(1);
1309        }
1310    }
1311
1312    if(argc==193) {
1313        if(help()==0) {
1314            exit(1);
1315        }
1316    }
1317
1318    if(argc==194) {
1319        if(help()==0) {
1320            exit(1);
1321        }
1322    }
1323
1324    if(argc==195) {
1325        if(help()==0) {
1326            exit(1);
1327        }
1328    }
1329
1330    if(argc==196) {
1331        if(help()==0) {
1332            exit(1);
1333        }
1334    }
1335
1336    if(argc==197) {
1337        if(help()==0) {
1338            exit(1);
1339        }
1340    }
1341
1342    if(argc==198) {
1343        if(help()==0) {
1344            exit(1);
1345        }
1346    }
1347
1348    if(argc==199) {
1349        if(help()==0) {
1350            exit(1);
1351        }
1352    }
1353
1354    if(argc==200) {
1355        if(help()==0) {
1356            exit(1);
1357        }
1358    }
1359
1360    if(argc==201) {
1361        if(help()==0) {
1362            exit(1);
1363        }
1364    }
1365
1366    if(argc==202) {
1367        if(help()==0) {
1368            exit(1);
1369        }
1370    }
1371
1372    if(argc==203) {
1373        if(help()==0) {
1374            exit(1);
1375        }
1376    }
1377
1378    if(argc==204) {
1379        if(help()==0) {
1380            exit(1);
1381        }
1382    }
1383
1384    if(argc==205) {
1385        if(help()==0) {
1386            exit(1);
1387        }
1388    }
1389
1390    if(argc==206) {
1391        if(help()==0) {
1392            exit(1);
1393        }
1394    }
1395
1396    if(argc==207) {
1397        if(help()==0) {
1398            exit(1);
1399        }
1400    }
1401
1402    if(argc==208) {
1403        if(help()==0) {
1404            exit(1);
1405        }
1406    }
1407
1408    if(argc==209) {
1409        if(help()==0) {
1410            exit(1);
1411        }
1412    }
1413
1414    if(argc==210) {
1415        if(help()==0) {
1416            exit(1);
1417        }
1418    }
1419
1420    if(argc==211) {
1421        if(help()==0) {
1422            exit(1);
1423        }
1424    }
1425
1426    if(argc==212) {
1427
```

```
56     break;
57 case 'c':
58     man_opt_c=atoi(optarg); /* Change man_opt_c value to memorize
59                     that the option 'c' is parsed */
60     /* Check if before 'c' option was passed the mandatory option 'd'
61                     */
62     if(man_opt_d!=-1){
63         *channel=atoi(optarg);
64     }
65     else{
66         printf("Missing mandatory parameter 'd'\n");
67         usage(argv[0]);
68         return -1;
69     }
70     break;
71 case 'i':
72     /* Check if before 'i' option was passed the mandatory option 'd'
73                     */
74     if(man_opt_d!=-1){
75         /* Check if before 'i' option was passed the mandatory option 'c'
76                     */
77         if(man_opt_c!=-1){
78             *r_w=READ; /* Set READ operation for GPIO */
79             i_or_o=1;
80         }
81         else{
82             printf("Missing mandatory parameter 'c'\n");
83             usage(argv[0]);
84             return -1;
85         }
86     }
87     else{
88         printf("Missing mandatory parameter 'd'\n");
89         usage(argv[0]);
90         return -1;
91     }
92     break;
93 case 'o':
94     /* Check if before 'o' option was passed the mandatory option 'd'
95                     */
96     if(man_opt_d!=-1){
97         /* Check if before 'o' option was passed the mandatory option 'c'
98                     */
99         if(man_opt_c!=-1){
100             *r_w=WRITE; /* Set WRITE operation for GPIO */
101             *value=strtoul(optarg, NULL, 0);
102             i_or_o=1;
103         }
104         else{
```

```

99         printf("Missing mandatory parameter 'c'\n");
100        usage(argv[0]);
101        return -1;
102    }
103}
104else{
105    printf("Missing mandatory parameter 'g'\n");
106    usage(argv[0]);
107    return -1;
108}
109break;
110case 'h':
111    usage(argv[0]);
112    return 1;
113    break;
114case ':':
115    printf("Missing argument for '%c' option\n", optopt);
116    usage(argv[0]);
117    return -1;
118    break;
119case '?':
120    printf("Option '%c' not recognized\n", optopt);
121    usage(argv[0]);
122    return -1;
123    break;
124default:
125    usage(argv[0]);
126    return -1;
127    break;
128}
129}
130}
131/* Check if channel number is right*/
132if(*channel!=GPIO_CHANNEL_1 && *channel!=GPIO_CHANNEL_2){
133    printf("Wrong channel number inserted\n");
134    usage(argv[0]);
135    return -1;
136}
137
138/* Check if an i/o operation is requested */
139if(i_or_o==0){
140    printf("Can't use "BOLDWHITE"uiodriver "RESET"without specify i/o option
141          \n");
142    usage(argv[0]);
143    return -1;
144}
145return 0;
}

```

Codice 7.12: "uiodriver function.h"

**open\_device**

Ha la stessa struttura della gemella `open_memory`. La sola differenza è che per generare l'indirizzo fisico non utilizza il descrittore di file associato al file dell'indirizzo di memoria, ma utilizza il descrittore associato al file del device `uio`.

```

147 /**
148 * @brief Opens uio device file in order to access to it by its virtual
149 * address.
150 * @param [out] fd: file descriptor name.
151 * @param [in] uiiod: uio device file corrisponding to GPIO.
152 * @param [out] virtual_address: uio device file's virtual address.
153 * @retval Integer status:
154 *         -1      An error occurred.
155 */
156 int open_device(int* fd, char* uiiod, void** virtual_address) {
157     *fd = open(uiiod, O_RDWR);
158     if (*fd < 1) {
159         perror("Error to open uio device file");
160         return -1;
161     }
162     /* mmap system call returns virtual address of <uiiod> file */
163     *virtual_address = mmap(NULL, GPIO_MAP_SIZE, PROT_READ|PROT_WRITE,
164     MAP_SHARED, *fd, 0);
165     return 0;
166 }
```

Codice 7.13: "uiodriver function.h"

**read\_gpio**

Nella funzione vengono dichiarate 2 variabili locali `gpio_tri_off` e `gpio_data_off`. A tali variabili vengono assegnati i valori appropriati (richiamando le macro dichiarate) in base al valore di `channel` che viene passato in ingresso alla funzione. Per il resto la lettura avviene così come nella prima tipologia di driver.

```

166 /**
167 * @brief Reads the <value> in input to GPIO port.
168 * @param [out] value: read value.
169 * @param [in] virtual_address: uio device file's virtual address.
170 * @param [in] channel: number corrisponding to GPIO channel
171 * @retval None.
172 */
173 void read_gpio(int* value, void* virtual_address, int channel){
174     /* Set the offsets to access to right gpio channel (1 or 2) */
175     int gpio_tri_off;
176     int gpio_data_off;
177     if(channel==GPIO_CHANNEL_1) {
178         gpio_tri_off=GPIO_TRI_OFFSET;
```

```

179     gpio_data_off=GPIO_DATA_OFFSET;
180 }
181 else{
182     gpio_tri_off=GPIO2_TRI_OFFSET;
183     gpio_data_off=GPIO2_DATA_OFFSET;
184 }
185 /* Set GPIO_TRI register values to configure the GPIO port as input */
186 *((unsigned *) (virtual_address + gpio_tri_off)) = 255;
187 /* Memorize the value from GPIO port in <value> */
188 *value = *((unsigned *) (virtual_address + gpio_data_off));
189 printf("The value on channel %d GPIO input is: %08x\n",channel,*value);
190 }
```

Codice 7.14: "uiodriver function.h"

**write\_gpio**

Il meccanismo di scrittura è simile al driver precedente. Anche qui viene usato lo stesso procedimento della funzione read\_gpio per poter accedere correttamente al canale desiderato della periferica.

```

192 /**
193 * @brief Writes <value> in output to GPIO port.
194 * @param [in] value: write value.
195 * @param [in] uiiod: uio device file corrisponding to GPIO
196 * @param [in] virtual_address: uio device file's virtual address.
197 * @param [in] channel: number corrisponding to GPIO channel
198 * @retval None.
199 */
200 void write_gpio(int value,char* uiiod,void* virtual_address, int channel){
201 /* Set the offsets to access to right gpio channel (1 or 2) */
202     int gpio_tri_off;
203     int gpio_data_off;
204     if(channel==GPIO_CHANNEL_1){
205         gpio_tri_off=GPIO_TRI_OFFSET;
206         gpio_data_off=GPIO_DATA_OFFSET;
207     }
208     else{
209         gpio_tri_off=GPIO2_TRI_OFFSET;
210         gpio_data_off=GPIO2_DATA_OFFSET;
211     }
212     /* Set GPIO_TRI register values to configure the GPIO port as output */
213     *((unsigned *) (virtual_address + gpio_tri_off)) = 0;
214     printf("Value %08x is going to write onto %s (channel %d)\n", value, uiiod,
215           channel);
216     /* Write <value> in GPIO register */
217     *((unsigned *) (virtual_address + gpio_data_off)) = value;
}
```

Codice 7.15: "uiodriver function.h"

**close\_device**

La funzione è identica alla close\_memory.

```

219 /**
220 * @brief Closes /dev/uiox file.
221 * @param [in] fd: file descriptor name of uio device
222 * @param [in] virtual_address: uio device's virtual address.
223 * @retval None.
224 */
225 void close_device(int fd, void* virtual_address) {
226     munmap(virtual_address, GPIO_MAP_SIZE);
227     close(fd);
228 }
```

Codice 7.16: "uiodriver function.h"

### 7.2.2.3 main.c

Anche il main è molto simile al precedente driver. In aggiunta, si ha la dichiarazione della variabile intera cha, in cui viene salvato il canale della periferica.

```

1 /**
2 ****
3 * @file      uiodriver.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     24-June-2017
8 * @brief    driver "uiodriver" to control gpio
9 ****
10 */
11 /* Includes
12 -----------*/
13 #include "uiodriver_header.h"
14
15 int main(int argc, char *argv[]){
16     /*!< First step */
17     /*! Variables Declaration */
18     int file_descriptor; /* uio file descriptor */
19     int r_w=READ; /* Saves operation's type to do with GPIO */
20     int r_or_w_value = 0; /* Saves read/write value from/on GPIO */
21     int ret_parse; /* Returned value from parse_command function */
22
23     char* uio_dev; /* uio device file name */
24     int cha; /* channel number of GPIO */
25     void *virt_addr; /* Virtual address of <uio_dev> */
26     /*!< Second step */
27 }
```

```

26  /*! parse_command function is called to parse all argument passed to
27   * driver uiodriver */
28  /* Check if parse_command function returns error */
29  if((ret_parse=parse_command(argc,argv,&uio_dev,&cha,&r_w,&r_or_w_value))  

30      ==1)
31      return 0;
32  else
33      if(ret_parse== -1)
34          return -1;
35  /*!< Third step */
36  /*! open_device function is called to achieve gpio virtual address */
37  /* Check if open_device function returns error */
38  if(open_device(&file_descriptor,uio_dev,&virt_addr)== -1) {
39      printf("nodriver aborted!\n");
40      return -1;
41  }
42  //!
43  /*!< Fourth */
44  /*! read_gpio is called if r_w variable is READ, otherwise is called
45   * write_gpio function */
46  /* Calls conveniently read_gpio or write_gpio function */
47  if (r_w == READ) read_gpio(&r_or_w_value,virt_addr,cha);
48  else write_gpio(r_or_w_value,uio_dev,virt_addr,cha);
49  /*!< First step */
50  /*! close_device function is called to close device file and delete file
51   * descriptor */
52  close_device(file_descriptor,virt_addr);
53
54  return 0;
55 }
```

Codice 7.17: "main.c"

#### 7.2.2.4 makefile

```

1 uiodriver : main.o uiodriver_function.o
2     cc -o uiodriver main.o uiodriver_function.o
3
4 main.o : main.c uiodriver_function.c uiodriver_header.h
5     cc -c main.c -o main.o
6
7 uiodriver_function.o : uiodriver_function.c uiodriver_header.h
8     cc -c uiodriver_function.c -o uiodriver_function.o
9
10 clean :
11     rm -f uiodriver
12     rm *.o
13     echo "Clean all file!"
```

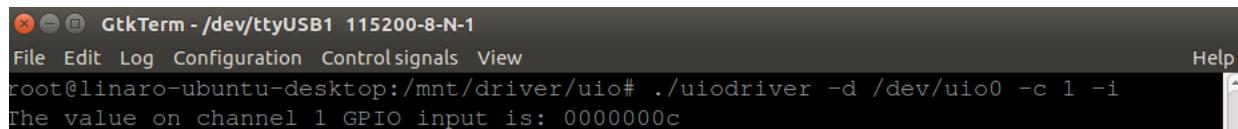
---

Codice 7.18: "makefile"

### 7.2.2.5 Esempio di esecuzione

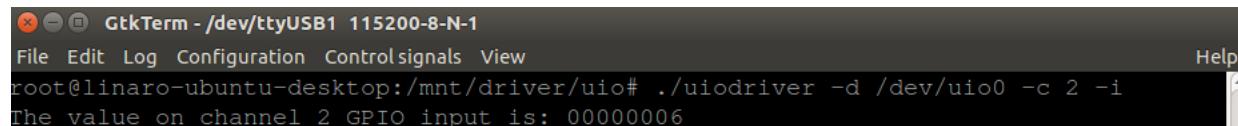
Come per il driver precedente, di seguito si riportano alcuni esempi di funzionamento, in particolare si mostra:

- una lettura sui button, fig. 7.11;
- una lettura sugli switch, fig. 7.12;
- una scrittura sui led, fig. 7.13;
- il richiamo della funzione di help, fig. 7.14.



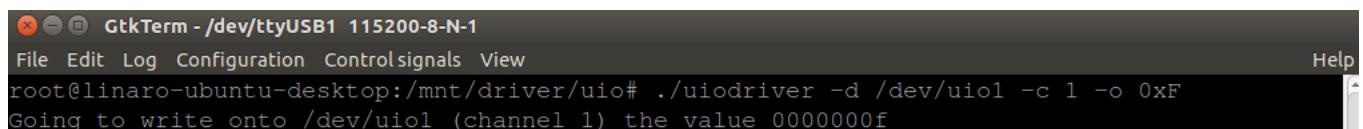
```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@linaro-ubuntu-desktop:/mnt/driver/uio# ./uiodriver -d /dev/uio0 -c 1 -i
The value on channel 1 GPIO input is: 0000000c
```

Figura 7.11: Messaggio restituito dopo la pressione di BTN(V16) e BTN3(Y16)



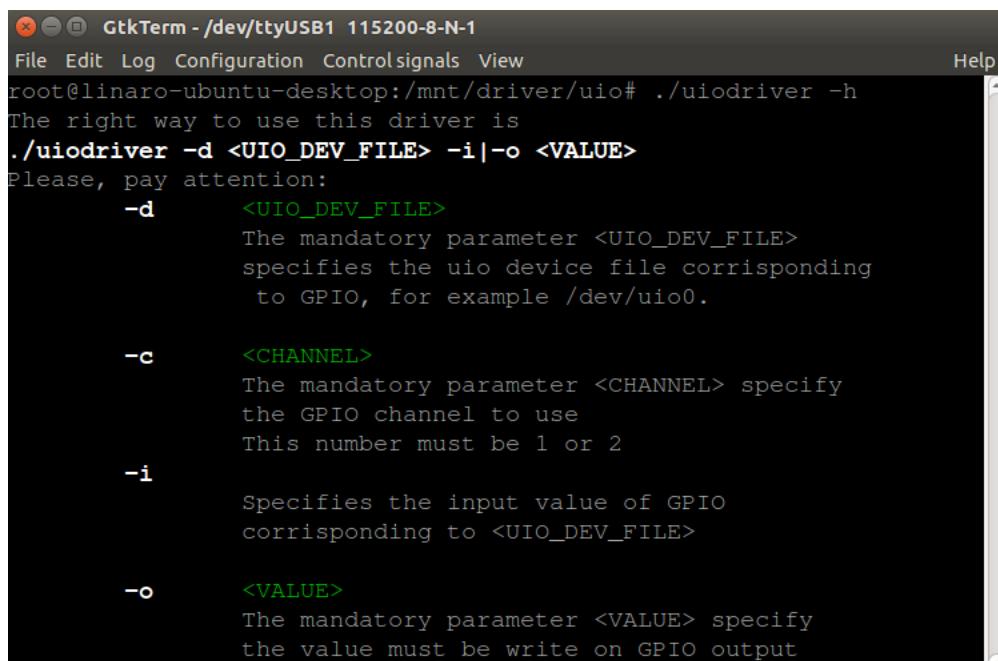
```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@linaro-ubuntu-desktop:/mnt/driver/uio# ./uiodriver -d /dev/uio0 -c 2 -i
The value on channel 2 GPIO input is: 00000006
```

Figura 7.12: Messaggio restituito dopo aver alzato SW1(P15) e SW2(W13)



```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@linaro-ubuntu-desktop:/mnt/driver/uio# ./uiodriver -d /dev/uio1 -c 1 -o 0xF
Going to write onto /dev/uio1 (channel 1) the value 0000000f
```

Figura 7.13: Messaggio restituito dopo che sulla board si sono accesi LD0(M14), LD1(M15), LD2(G14) e LD3(D18)



```

GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Control signals View Help
root@linaro-ubuntu-desktop:/mnt/driver/uio# ./uiodriver -h
The right way to use this driver is
./uiodriver -d <UIO_DEV_FILE> -i|-o <VALUE>
Please, pay attention:
-d      <UIO_DEV_FILE>
        The mandatory parameter <UIO_DEV_FILE>
        specifies the uio device file corresponding
        to GPIO, for example /dev/uio0.

-c      <CHANNEL>
        The mandatory parameter <CHANNEL> specify
        the GPIO channel to use
        This number must be 1 or 2

-i
        Specifies the input value of GPIO
        corrisponding to <UIO_DEV_FILE>

-o      <VALUE>
        The mandatory parameter <VALUE> specify
        the value must be write on GPIO output

```

Figura 7.14: Usage function

### 7.2.3 Terza tipologia

Questa terza tipologia di driver è simile alla seconda, il device, infatti, viene acceduto mediante driver uio. La differenza con il precedente sta nell'utilizzo del meccanismo delle interruzioni.

A partire dalla sintesi hardware del precedente driver, si aggiunge ad ogni periferica GPIO l'abilitazione a funzionare con le interruzioni. Per far ciò si rimanda alla guida nel precedente capitolo. Il block design risultante dovrebbe essere simile a quello descritto in fig. 7.15.

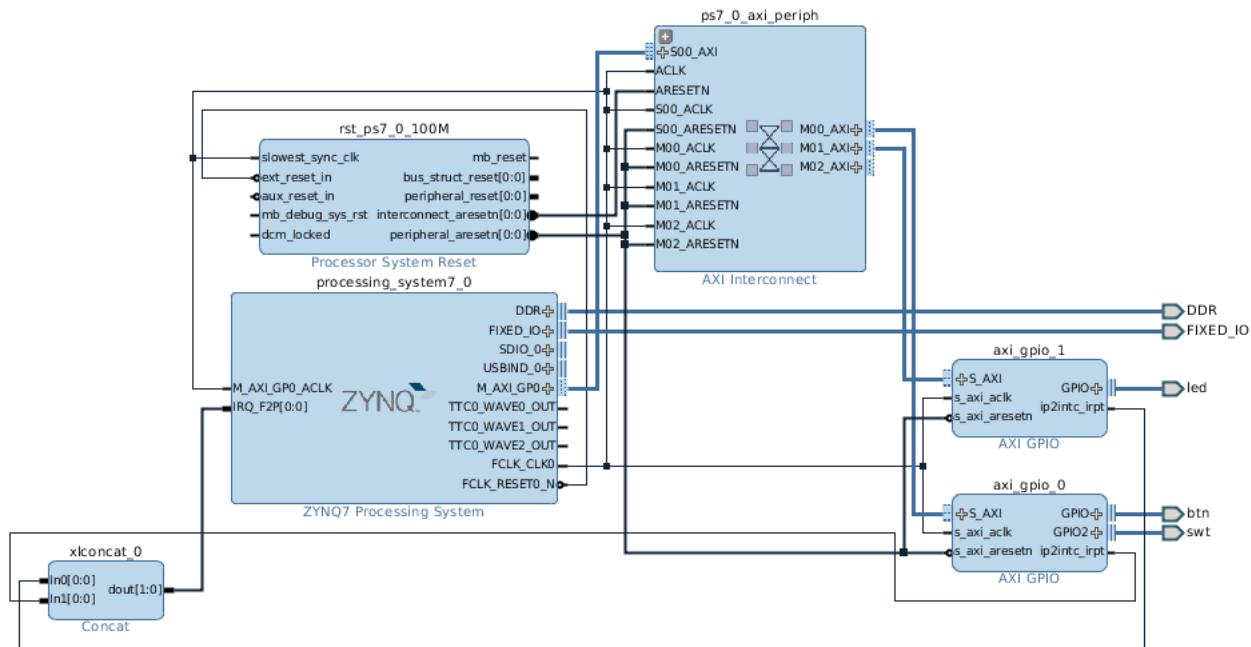


Figura 7.15: Block Design

Si nota che è stato aggiunto il componente **concat** per il corretto instradamento delle linee di interruzione. Facendo doppio click sul componente, si impostano il numero di porti da voler utilizzare, fig. 7.16.

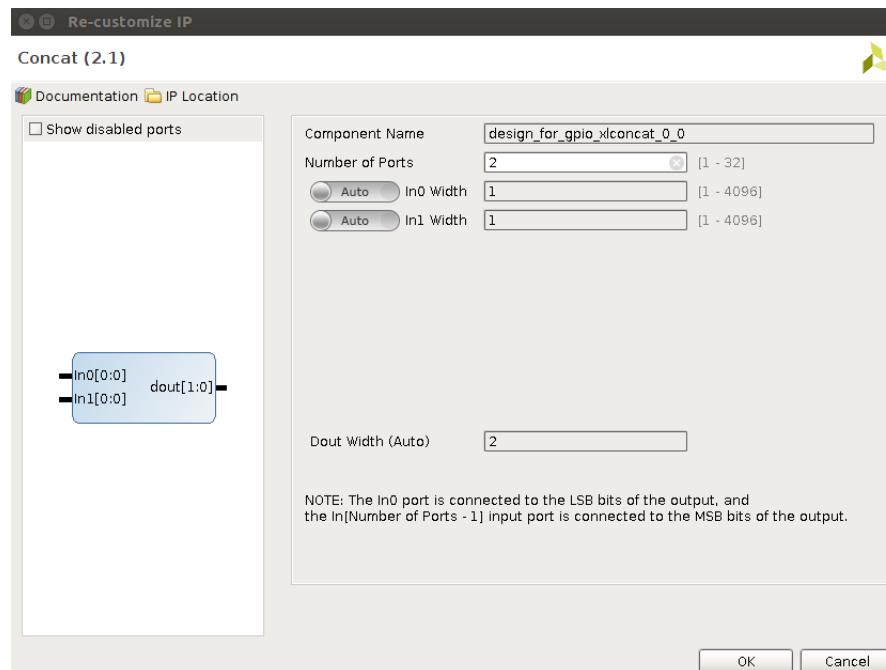


Figura 7.16: Concat customization

Dopo aver generato il bitstream e aver lanciato SDK, si eseguono le stesse procedure di modifica descritte precedentemente al fine di rendere la periferica compatibile con i driver uio. Questa volta, nel file pl.dts, sono comparsi altri campi che descrivono come verranno gestite le interruzioni del componente.

```

1  /*
2   * CAUTION: This file is automatically generated by Xilinx.
3   * Version:
4   * Today is: Sun Jul  9 20:39:39 2017
5   */
6
7
8  {
9      amba_pl: amba_pl {
10         #address-cells = <1>;
11         #size-cells = <1>;
12         compatible = "simple-bus";
13         ranges ;
14         axi_gpio_0: gpio@41200000 {
15             #gpio-cells = <2>;
16             #interrupt-cells = <2>;
17             compatible = "generic-uio";
18             gpio-controller ;
19             interrupt-controller ;
20             interrupt-parent = <&intc>;
21             interrupts = <0 30 4>;
22             reg = <0x41200000 0x10000>;
23             xlnx,all-inputs = <0x1>;
24             xlnx,all-inputs-2 = <0x1>;
25             xlnx,all-outputs = <0x0>;
26             xlnx,all-outputs-2 = <0x0>;
27             xlnx,dout-default = <0x00000000>;
28             xlnx,dout-default-2 = <0x00000000>;
29             xlnx, gpio-width = <0x4>;
30             xlnx, gpio2-width = <0x4>;
31             xlnx, interrupt-present = <0x1>;
32             xlnx, is-dual = <0x1>;
33             xlnx, tri-default = <0xFFFFFFFF>;
34             xlnx, tri-default-2 = <0xFFFFFFFF>;
35         };
36         axi_gpio_1: gpio@41210000 {
37             #gpio-cells = <2>;
38             #interrupt-cells = <2>;
39             compatible = "generic-uio";
40             gpio-controller ;
41             interrupt-controller ;
42             interrupt-parent = <&intc>;
43             interrupts = <0 29 4>;
44             reg = <0x41210000 0x10000>;

```

```

45     xlnx,all-inputs = <0x0>;
46     xlnx,all-inputs-2 = <0x0>;
47     xlnx,all-outputs = <0x1>;
48     xlnx,all-outputs-2 = <0x0>;
49     xlnx,dout-default = <0x00000000>;
50     xlnx,dout-default-2 = <0x00000000>;
51     xlnx,gpio-width = <0x4>;
52     xlnx,gpio2-width = <0x20>;
53     xlnx,interrupt-present = <0x1>;
54     xlnx,is-dual = <0x0>;
55     xlnx,tri-default = <0xFFFFFFFF>;
56     xlnx,tri-default-2 = <0xFFFFFFFF>;
57 };
58 };
59 };

```

Codice 7.19: "pl.dts"

In particolare, nel campo `interrupt-parent` viene definito il controller delle interruzioni. Prendendo in esempio l'`axi_gpio_1`, i valori `<0 29 4>` del campo `interrupts` significano:

- 0 - l'interruzione non è SPI, altrimenti ci sarebbe stato 1;
- 29 - è il numero dell'interruzione gestita dal GIC a cui bisogna sommare il valore 32. Se fosse stata un'interruzione SPI, si sarebbe sommato 16.
- 4 - l'interruzione viene presa in carico sui livelli alti del segnale. (1 per i rising edge; 2 falling edge; 8 livello basso).

### 7.2.3.1 uiointdriver\_header.h

In questo header file, rispetto a quello del driver precedente, si noti che è stata ampliata la parte riguardante le macro relative ai registri della GPIO. Sono state aggiunte `GPIO_GIER_OFFSET`, `GPIO_IP_ISR_OFFSET` e `GPIO_IP_IER_OFFSET` che individuano, rispettivamente, gli offset dei registri Global Interrupt Enable Register, Interrupt Status Register e Interrupt Enable Register. Sono state aggiunte anche delle maschere per l'abilitazione e la disabilitazione di tali registri, tenendo conto delle specifiche descritte nella documentazione della GPIO Xilinx, [https://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_gpio/v2\\_0/pg144-axi-gpio.pdf](https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf).

```

1 /**
2 ****
3 * @file      uiointdriver_header.h
4 * @author Colella Gianni - Guida Ciro - Lombardi Daniele
5 *           Group IV - Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date      2-July-2017
8 * @brief     library 'uiointdriver' for GPIO
9 ****

```

```

10  */
11
12 /* Includes
13  -----
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <stdint.h>
16 #include <unistd.h>
17 #include <sys/mman.h>
18 #include <fcntl.h>
19 #include <errno.h>
20 #include <sys/types.h>
21 #include <sys/stat.h>
22
23 /* Colour MACRO
24  -----
24 #define RESET "\033[0m" /*!< Reset colour */
25 #define GREEN "\033[32m" /*!< Green colour */
26 #define BOLDWHITE "\033[1m\033[37m" /*!< Bold White colour */
27
28 /* GPIO operation MACRO
29  -----
29 #define READ 0 /*!< Read operation is set */
30 #define WRITE 1 /*!< Write operation is set */
31
32 /* GPIO page MACRO
33  -----
33 #define GPIO_MAP_SIZE 0x10000 /*!< Bit mask to extract page address,
34   without offset */
34 #define GPIO_DATA_OFFSET 0x00 /*!< Offset of GPIO_DATA register */
35 #define GPIO_TRI_OFFSET 0x04 /*!< Offset of GPIO_TRI register */
36 #define GPIO2_DATA_OFFSET 0x08 /*!< Offset of GPIO2_DATA register */
37 #define GPIO2_TRI_OFFSET 0x0C /*!< Offset of GPIO2_TRI register */
38 #define GPIO_GIER_OFFSET 0x11C /*!< Offset of Global interrupt enable
39   register */
40 #define GPIO_IP_ISR_OFFSET 0x120 /*!< Offset of IP Interrupt status
41   register */
42 #define GPIO_IP_IER_OFFSET 0x128 /*!< Offset of IP Interrupt enable
43   register */
44
45 /* GPIO enable/disable mask
46  -----
46 #define GPIO_GIER_DISABLE 0x00000000 /*!< GPIO Global Interrupt disable
47   mask */
48 #define CHANNEL_1_IER_DISABLE 0x00000000 /*!< GPIO Channel 1 Interrupt
49   disable mask */
50 #define CHANNEL_2_IER_DISABLE 0x00000000 /*!< GPIO Channel 2 Interrupt
51   disable mask */
52 #define GPIO_GIER_ENABLE 0x80000000 /*!< GPIO Global Interrupt enable mask

```

```

        */
47 #define CHANNEL_1_IER_ENABLE 0x00000001 /*!< GPIO Channel 1 Interrupt
      enable mask */
48 #define CHANNEL_2_IER_ENABLE 0x00000002 /*!< GPIO Channel 2 Interrupt
      enable mask */
49 #define CHANNEL_1_ISR    0x00000001 /*!< GPIO Channel 1 ISR mask */
50 #define CHANNEL_2_ISR    0x00000002 /*!< GPIO Channel 2 ISR mask */

51
52 /* GPIO channel MACRO
-----*/
53 #define GPIO_CHANNEL_1 1 /*!< GPIO channel 1 selected */
54 #define GPIO_CHANNEL_2 2 /*!< GPIO channel 2 selected */

55
56 /* Function prototypes
-----*/
57 void usage(char *name); /*!< usage function is called to help the user */
58 int parse_command(int argc,char **argv,char** uiiod,int* channel,int* r_w,int
      * value);/*!< parse_command function is called to parse arguments passed
      to driver */
59 int open_device(int* fd, char* uiiod, void** virtual_address);/*!<
      open_device function is called to open file associated to uio device */
60 int read_gpio(int fd,int* value,char* uiiod, void* virtual_address, int
      channel);/*!< read_gpio function is called to do read operation from
      GPIO */
61 void write_gpio(int value,char* uiiod,void* virtual_address, int channel)
      ;/*!< write_gpio function is called to do write operation on GPIO */
62 void close_device(int fd,void* virtual_address);/*!< close_device function
      is called to close uio file */
63 int GPIO_Interrupt(long* base_add,int reg, unsigned long mask); /*!Set/reset
      Gpio registers in order to handle interrupts */

```

Codice 7.20: "uiointdriver header"

### 7.2.3.2 uiointdriver\_function.h

Di seguito si descrive l'unica funzione variata rispetto al driver precedente.

#### read\_gpio

Questa funzione è il cuore del driver uiointdriver. Innanzitutto, si osservi che sono state aggiunte una serie di variabili in più rispetto alla omonima funzione del precedente driver. Le variabili **num\_byte** e **test\_read** servono rispettivamente per tenere traccia del numero di byte letti dalla successiva operazione di read ed il contenuto di tale operazione. Servono unicamente come controllo. Alle variabili **gpio\_tri\_off** e **gpio\_data\_off**, si sono aggiunte **gpio\_cha\_interrupt\_en**, **gpio\_cha\_interrupt\_dis** e **gpio\_cha\_isr**. L'utilizzo è il medesimo, in base al valore del canale selezionato queste variabili assumeranno un certo valore che servirà per eseguire opportunamente le operazioni o sul canale 1 oppure sul 2.

Dopo aver settato la modalità di funzionamento del GPIO in lettura, semmai non fosse già stata predisposta, il driver entra in un ciclo infinito in cui legge il valore presente sul GPIO ogni volta che si scatena un'interruzione.

Il primo passo compiuto è quello di abilitare le interruzioni, pertanto, prima viene settato il bit 31 del registro GIER, poi, successivamente viene abilitato anche il bit del registro IPIER, rispettando il canale che si vuole utilizzare. Abilitate le periferiche, il processo si sospende sulla periferica, in attesa di essere svegliato dall'arrivo di un'interruzione. Una volta svegliato, viene effettuato un controllo per stabilire se l'operazione di read ha generato qualche problema, utilizzando le variabili di cui sopra. A questo punto, dopo aver disabilitato sia l'interruzione globale, sia quella del canale in oggetto, viene effettuata l'operazione di read sul registro data della GPIO. Se si decommenta il while al rigo 231, si induce il processo ad arrestarsi fintanto che il segnale che ha generato l'interruzione non si riporta ad un livello basso, che è quello di default. In altre parole, il processo non effettua più alcuna lettura finché lo switch o il bottone non vengono rilasciati. Successivamente, sempre previo controllo, viene fatto un clean dell'interruzione, in modo che il SO possa ripristinare la linea di interruzione della periferica. Infine, viene inviato un ack alla periferica per comunicarle di essere stata servita.

```

166 /**
167 * @brief Reads the <value> in input to GPIO port.
168 * @param [in] fd: file descriptor name.
169 * @param [out] value: read value.
170 * @param [in] uiiod: uio device file corrisponding to GPIO
171 * @param [in] virtual_address: uio device file's virtual address.
172 * @param [in] channel: number corrisponding to GPIO channel
173 * @retval None.
174 */
175 int read_gpio(int fd,int* value,char* uiiod, void* virtual_address, int
176   channel){
177   /* number of bytes read */
178   ssize_t num_byte;
179   /* to test if read operation is correct */
180   uint32_t test_read=1;
181
182   /* Set the offsets to access to right gpio channel (1 or 2) */
183   int gpio_tri_off;
184   int gpio_data_off;
185   unsigned long gpio_cha_interrupt_en;
186   unsigned long gpio_cha_interrupt_dis;
187   unsigned long gpio_cha_isr;
188   if(channel==GPIO_CHANNEL_1){
189     gpio_tri_off=GPIO_TRI_OFFSET;
190     gpio_data_off=GPIO_DATA_OFFSET;
191     gpio_cha_interrupt_en=CHANNEL_1_IER_ENABLE;
192     gpio_cha_interrupt_dis=CHANNEL_1_IER_DISABLE;
193     gpio_cha_isr=CHANNEL_1_ISR;
194   }
195   else{
196     gpio_tri_off=GPIO2_TRI_OFFSET;
197     gpio_data_off=GPIO2_DATA_OFFSET;
198     gpio_cha_interrupt_en=CHANNEL_2_IER_ENABLE;
199     gpio_cha_interrupt_dis=CHANNEL_2_IER_DISABLE;
200     gpio_cha_isr=CHANNEL_2_ISR;
201
202   }
203 }
```

```

200 }
201 /*-----*/
202
203 /* Set GPIO_TRI register values to configure the GPIO port as input */
204 *((unsigned *) (virtual_address + gpio_tri_off)) = 255;
205
206 while(1) {
207
208     /* Enable Global Interrupt of GPIO */
209     GPIO_Interrupt(virtual_address,GPIO_GIER_OFFSET,GPIO_GIER_ENABLE);
210     /* Enable Channel Interrut of GPIO */
211     GPIO_Interrupt(virtual_address,GPIO_IP_IER_OFFSET,gpio_cha_interrupt_en)
212         ;
213
214     /* to suspend the process on gpio peripheral */
215     num_byte=read(fd,&test_read,sizeof(test_read));
216     if(num_byte!=sizeof(test_read)){
217         printf("It occurs an error on read opeartion\n");
218         return -1;
219     }
220     else{
221         /* Disable Global Interrupt of GPIO */
222         GPIO_Interrupt(virtual_address,GPIO_GIER_OFFSET,GPIO_GIER_DISABLE);
223         /* Disable Channel Interrut of GPIO */
224         GPIO_Interrupt(virtual_address,GPIO_IP_IER_OFFSET,
225                         gpio_cha_interrupt_dis);
226
227         /* Memorize the value from GPIO port in <value> */
228         *value = *((unsigned *) (virtual_address + gpio_data_off));
229         printf("The value on channel %d of %s input is: %08x\n",channel,uiod,*value);
230     }
231
232     /* In this case the driver don't read the register until the input that
233      cause the interrupt is set to default low position */
234     // while(*((unsigned *) (virtual_address + gpio_data_off))!=0);
235
236     /* test reenable interrupt */
237     ssize_t re_enable = 1;
238     /* to clean interrupt */
239     re_enable = write(fd, &re_enable, sizeof(re_enable));
240     if (re_enable < sizeof(re_enable)) {
241         perror("write");
242         close(fd);
243         exit(EXIT_FAILURE);
244     }
245
246     /* Sent ack to GPIO peripheral */
247     GPIO_Interrupt(virtual_address, GPIO_IP_ISR_OFFSET, gpio_cha_isr);

```

```
245 }
246 }
247 }
```

Codice 7.21: "uiointdriver header.h"

### GPIO\_Interrupt

Questa funzione, in base al valore di reg passatole, viene utilizzata per modificare il contenuto dei registri della GPIO, adibiti al controllo delle interruzioni. In particolare, il nuovo valore da assumere viene passato attraverso **mask**, come unsigned long.

```
288 /**
289 * @brief According to the offset <reg>, modifies the values of enable
290 * registers of GPIO custom peripheral
291 * @param [in] base_add: virtual base address of GPIO peripheral
292 * @param [in] reg: offset value of selected register
293 * @param [in] mask: register value
294 * @retval 1.
295 */
296 int GPIO_Interrupt(long* base_add, int reg, unsigned long mask)
297 {
298     *((unsigned *) (base_add + (reg/4)))=mask;
299     return 1;
}
```

Codice 7.22: "uiointdriver function"

#### 7.2.3.3 main.c

Il main del driver è praticamente identico al precedente.

#### 7.2.3.4 makefile

```
1 uiodriver : main.o uiointdriver_function.o
2     cc -o uiointdriver main.o uiointdriver_function.o
3
4 main.o : main.c uiointdriver_function.c uiointdriver_header.h
5     cc -c main.c -o main.o
6
7 uiointdriver_function.o : uiointdriver_function.c uiointdriver_header.h
8     cc -c uiointdriver_function.c -o uiointdriver_function.o
9
10 clean :
11     rm -f uiointdriver
12     rm *.o
13     echo "Clean all file!"
```

Codice 7.23: "makefile"

### 7.2.3.5 Esempio di esecuzione

Di seguito, si mostra un esempio di esecuzione del driver in cui si sceglie di leggere dalla periferica GPIO mappata sugli switch della board Zybo, fig. 7.17.

```
GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Control signals View Help
root@linaro-ubuntu-desktop:/mnt/driver/Xilinx/uio_int# ./uiointdri
ver -d /dev/uio0
o0 -c 2 -i
The value on channel 2 of /dev/uio0 input is: 00000008
The value on channel 2 of /dev/uio0 input is: 0000000c
The value on channel 2 of /dev/uio0 input is: 0000000d
The value on channel 2 of /dev/uio0 input is: 0000000f
The value on channel 2 of /dev/uio0 input is: 0000000e
The value on channel 2 of /dev/uio0 input is: 0000000c
The value on channel 2 of /dev/uio0 input is: 00000008
The value on channel 2 of /dev/uio0 input is: 00000000
The value on channel 2 of /dev/uio0 input is: 00000002
The value on channel 2 of /dev/uio0 input is: 00000000
The value on channel 2 of /dev/uio0 input is: 00000004
The value on channel 2 of /dev/uio0 input is: 00000000
The value on channel 2 of /dev/uio0 input is: 00000008
The value on channel 2 of /dev/uio0 input is: 0000000a
The value on channel 2 of /dev/uio0 input is: 0000000b
The value on channel 2 of /dev/uio0 input is: 0000000a
The value on channel 2 of /dev/uio0 input is: 0000000e
The value on channel 2 of /dev/uio0 input is: 0000000c
/dev/ttyUSB1 115200-8-N-1 DTR RTS CTS CD DSR RI
```

Figura 7.17: Read operation

## 7.3 Procedimento: GPIO custom

Come si evince dalla traccia, in questa sezione si fa riferimento alla GPIO custom sviluppata nel Capitolo 1 di questa documentazione. In generale, le funzioni utilizzate dal driver sono praticamente analoghe a quelle viste nella precedente sezione, fatta eccezione per il fatto che si deve tener conto della diversa struttura interna della periferica, in particolar modo della locazione dei vari registri. Per non ripetersi nella trattazione, di seguito sono riportati semplicemente i codici utilizzati. In più, quando i codici non presentano grosse differenze tra una tipologia e l'altra del codice, sono omessi.

### 7.3.1 Prima tipologia

Di seguito si mostra il design project di riferimento in fig. 7.18, mentre in fig. 7.19 sono mostrati i base address delle periferiche utilizzate.

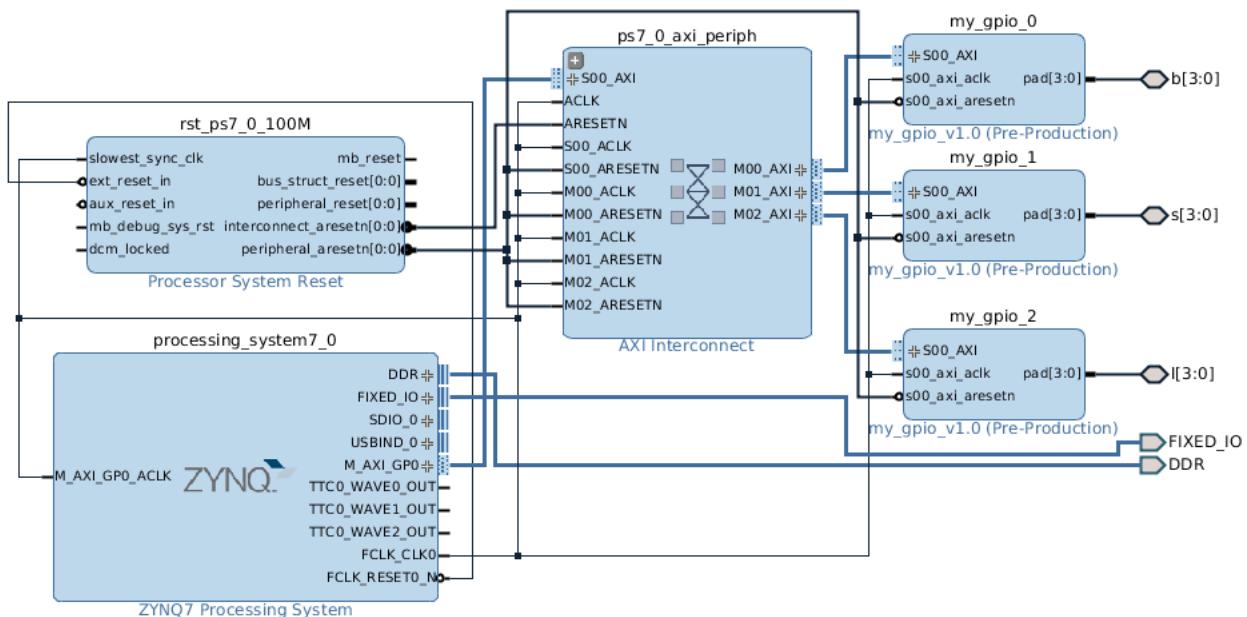


Figura 7.18: Design project GPIO custom

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [ 1G ])					
my_gpio_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
my_gpio_1	S00_AXI	S00_AXI_reg	0x43C1_0000	64K	0x43C1_FFFF
my_gpio_2	S00_AXI	S00_AXI_reg	0x43C2_0000	64K	0x43C2_FFFF

Figura 7.19: Base address periferiche

### 7.3.1.1 mygpio\_nodriver.h

```

1  /**
2   ****
3
4   * @file      mygpio_nodriver.h
5   * @author Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
6   *           Sistemi Embedded 2016-2017
7   * @version   V1.0
8   * @date     7-July-2017
9   * @brief    library mygpio_nodriver gpio
10  ****
11 */
12 /* Includes
----- */

```

```

12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <unistd.h>
15 #include <sys/mman.h>
16 #include <fcntl.h>
17 #include <errno.h>
18 #include "gpio_custom.h"

19
20 /**
21 * @brief Data Structure to manage data associated to GPIO custom
22 * peripheral
23 */
24 typedef struct{
25     gpio_custom_TypeDef* gpio_custom; /*!< Contains virtual address (page
26                                     offset included) and offset registers value of GPIO */
27     int gpio_phy_address;           /*!< Physical address of GPIO */
28     int fd;                      /*!< Memory file descriptor */
29     int r_w;                     /*!< Saves operation's type to do with GPIO */
30     uint32_t r_or_w_value;        /*!< Saves read/write value from/on GPIO */
31 }mygpio_TypeDef;

32 /* GPIO port MACRO
33 -----
34 #define GPIO_PORT 0xF             /*!< GPIO ports selected to read/write */

35 /* Colour MACRO
36 -----
37 #define RESET      "\033[0m"    /*!< Reset colour */
38 #define GREEN      "\033[32m"    /*!< Green colour */
39 #define BOLDWHITE  "\033[1m\033[37m" /*!< Bold White colour */

40 /* GPIO operation MACRO
41 -----
42 #define READ       1           /*!< Read operation is set */
43 #define WRITE      0           /*!< Write operation is set */

44 /* GPIO page MACRO
45 -----
46 #define PAGE_SIZE sysconf(_SC_PAGESIZE) /*!< Page size used by SO */
47 #define MASK_SIZE (~(PAGE_SIZE-1))    /*!< Bit mask to extract page address,
48                                     without offset */

49 /* Function prototypes
50 -----
51 void mygpio_usage(char *name); /*!< usage function is called to help the
52                               user */
53 int mygpio_parse_command(int argc,char **argv,mygpio_TypeDef* mygpio); /*!<
54                               parse_command function is called to parse arguments passed to driver */

```

```

51 int mygpio_open_memory(mygpio_TypeDef* mygpio); /*!< mygpio_open_memory
52     function is called to open memory file */
53 void mygpio_read_gpio(mygpio_TypeDef* mygpio); /*!< mygpio_read_gpio
54     function is called to do read operation from GPIO */
55 void mygpio_write_gpio(mygpio_TypeDef* mygpio); /*!< mygpio_write_gpio
56     function is called to do write operation on GPIO */
57 void mygpio_close_memory(mygpio_TypeDef* mygpio);/*!< mygpio_close_memory
58     function is called to close memory file */

```

Codice 7.24: "mygpio\_nodriver.h"

### 7.3.1.2 mygpio\_nodriver.h

```

1 /**
2 ****
3 * @file      mygpio_nodriver.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     7-July-2017
8 * @brief    Functions used for mygpio_nodriver gpio
9 ****
10 */
11 /* Includes
12 -----*/
13 #include "mygpio_nodriver.h"
14 /**
15 * @brief Describes how 'mygpio_nodriver' must be used, specifying all
16 *        supported options.
17 * @param [in] name: Specifies mygpio_nodriver name.
18 * @retval None.
19 */
20 void mygpio_usage(char *name) {
21     printf("The right way to use this driver is\n");
22     printf(BOLDWHITE"%s -g <GPIO_ADDRESS> -i|-o <VALUE>\n"RESET, name);
23     printf("Please, pay attention:\n");
24     printf(BOLDWHITE"\t-g"RESET GREEN"\t<GPIO_ADDR>"RESET"\n\t\tThe mandatory
          parameter <GPIO_ADDR>\n\t\tspecifies the physical address of GPIO.\n\t
          \t<GPIO_ADDR> can be expressed in\n\t\tdecimal representation (no
          prefix),\n\t\toctal representation (0 prefix)\n\t\tor hexadecimal
          representation\n\t\t(0x or 0X prefix)\n");
25     printf(BOLDWHITE"\n\t-i"RESET"\n\t\tSpecifies the input value of GPIO at\n
          \t\t<GPIO_ADDR> physical address \n");

```



```

69         /* Check if before 'o' option was passed the mandatory option 'g'
70         */
71     if(mandatory_opt!=-1){
72         mygpio->r_w=WRITE; /* Set WRITE operation for GPIO */
73         mygpio->r_or_w_value=strtoul(optarg, NULL, 0);
74         i_or_o=1;
75     }
76     else{
77         printf("Missing mandatory parameter 'g'\n");
78         mygpio_usage(argv[0]);
79         return -1;
80     }
81     break;
82 case 'h':
83     mygpio_usage(argv[0]);
84     return 1;
85     break;
86 case ':':
87     printf("Missing argument for '%c' option\n", optopt);
88     mygpio_usage(argv[0]);
89     return -1;
90     break;
91 case '?':
92     printf("Option '%c' not recognized\n", optopt);
93     mygpio_usage(argv[0]);
94     return -1;
95     break;
96 default:
97     mygpio_usage(argv[0]);
98     return -1;
99     break;
100 }
101 }
102 /* Check if an i/o operation is requested */
103 if(i_or_o==0){
104     printf("Can't use \"BOLDWHITE\"nodriver \"RESET\"without specify i/o option\
105           n");
106     mygpio_usage(argv[0]);
107     return -1;
108 }
109 return 0;
110 }
111 /**
112 * @brief Opens /dev/mem file in order to access custom GPIO address and
113 *        calculates page offset and virtual address of GPIO file.
114 * @param [inout] mygpio: data stucture that contains GPIO data
115 * @retval Integer status:

```

```

115     * -1      An error occurred.
116     */
117 int mygpio_open_memory(mygpio_TypeDef* mygpio) {
118
119     mygpio->fd = open ("/dev/mem", O_RDWR);
120     if (mygpio->fd < 1) {
121         perror("Error to open /dev/mem file");
122         return -1;
123     }
124     int page_phy_addr = (mygpio->gpio_phy_address) & MASK_SIZE;
125     printf("page_phy_addr %d\n",page_phy_addr); //debug
126     /* Calculates page offset */
127     off_t page_offset = (off_t)((mygpio->gpio_phy_address) - page_phy_addr);
128     printf("page_offset %lu\n",page_offset); //debug
129     /* mmap system call returns virtual page address of GPIO */
130     void* vrt_add = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED,
131                           mygpio->fd, page_phy_addr);
132     mygpio->gpio_custom->base_address=vrt_add + page_offset; // base address
133                           includes page offset
134
135     return 0;
136 }
137
138 /**
139  * @brief Reads the <mygpio->r_or_w_value> in input to GPIO port.
140  * @param [inout] mygpio: data stucture that contains GPIO data
141  * @retval None.
142  */
143 void mygpio_read_gpio(mygpio_TypeDef* mygpio) {
144     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
145     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, READ);
146     mygpio->r_or_w_value=gpio_custom_GetValue(mygpio->gpio_custom,GPIO_PORT);
147     printf("The value on GPIO %08x input is: %08x\n",mygpio->gpio_phy_address,
148           mygpio->r_or_w_value);
149 }
150
151 /**
152  * @brief Writes <mygpio->r_or_w_value> in output to GPIO port.
153  * @param [inout] mygpio: data stucture that contains GPIO data
154  * @retval None.
155  */
156 void mygpio_write_gpio(mygpio_TypeDef* mygpio) {
157     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
158     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, WRITE);
159     printf("Going to write onto GPIO %08x the value %08x\n", mygpio->
          gpio_phy_address, mygpio->r_or_w_value);
160     gpio_custom_SetValue(mygpio->gpio_custom, GPIO_PORT, LOW);
161     gpio_custom_SetValue(mygpio->gpio_custom, mygpio->r_or_w_value, HIGH);
162 }
```

```

160 /**
161 * @brief Closes /dev/mem file.
162 * @param [in] mygpio: data stucture that contains GPIO data
163 * @retval None.
164 */
165 void mygpio_close_memory(mygpio_TypeDef* mygpio){
166     munmap((void*) (mygpio->gpio_custom->base_address), PAGE_SIZE);
167     close(mygpio->fd);
168 }
```

Codice 7.25: "mygpio nodriver.c"

### 7.3.1.3 main.c

```

1 /**
2 ****
3 * @file      main.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *           Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     4-July-2017
8 * @brief    driver "mygpio_nodriver" to control gpio
9 ****
10 */
11 /* Includes
12 -----------*/
13 #include "mygpio_nodriver.h"
14
15 int main(int argc, char *argv[]){
16     /*!< First step */
17     /*! Structure and variable declaration */
18     mygpio_TypeDef mygpio; /* Structure associated to GPIO custom peripheral */
19     */
20     int ret_parse; /* Returned value from mygpio_parse_command function */
21     /*!< Second step */
22     /*! mygpio_parse_command function is called to parse all argument passed
23      to driver mygpio_nodriver */
24     /* Check if mygpio_parse_command function returns error */
25     if((ret_parse=mygpio_parse_command(argc, argv, &mygpio))==1)
26         return 0;
27     else
28         if(ret_parse== -1)
29             return -1;
30
31     /*!< Third step */
32 }
```

```

29  /*! open_memory function is called to achieve gpio virtual address */
30  /* Check if mygpio_open_memory function returns error */
31  if(mygpio_open_memory(&mygpio)==-1 {
32      printf("nodriver aborted!\n");
33      return -1;
34  }
35  //
36  /*!< Fourth step*/
37  /*! mygpio_read_gpio is called if r_w variable is READ, otherwise is
   called write_gpio function */
38  if ((mygpio.r_w) == READ) mygpio_read_gpio(&mygpio);
39  else mygpio_write_gpio(&mygpio);
40  /*!< First step */
41  /*! mygpio_close_memory function is called to close memory file and delete
   file descriptor */
42  mygpio_close_memory(&mygpio);
43
44  return 0;
45 }
```

Codice 7.26: "main.c"

### 7.3.1.4 makefile

```

1 mygpio_nodriver : main.o mygpio_nodriver.o gpio_custom.o
2     cc -o mygpio_nodriver main.o mygpio_nodriver.o gpio_custom.o
3
4 main.o : main.c mygpio_nodriver.c gpio_custom.c mygpio_nodriver.h
5     gpio_custom.h
6     cc -c main.c -o main.o
7
8 mygpio_nodriver.o : mygpio_nodriver.c mygpio_nodriver.h gpio_custom.h
9     cc -c mygpio_nodriver.c -o mygpio_nodriver.o
10
11 gpio_custom.o : gpio_custom.c gpio_custom.h
12     cc -c gpio_custom.c -o gpio_custom.o
13
14 clean :
15     rm -f mygpio_nodriver
16     rm *.o
17     echo "Clean all file!"
```

Codice 7.27: "makefile"

### 7.3.2 Seconda tipologia

Il design project e i base address sono i medesimi della tipologia precedente.

### 7.3.2.1 mygpio uiodriver.h

```

1  /**
2   ****
3   * @file      mygpio_uiodriver.h
4   * @author Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5   *           Sistemi Embedded 2016-2017
6   * @version   V1.0
7   * @date      7-July-2017
8   * @brief     library mygpio_uiodriver gpio
9   ****
10 */
11 /* Includes
12  -----
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <unistd.h>
16 #include <sys/mman.h>
17 #include <fcntl.h>
18 #include <errno.h>
19 #include "gpio_custom.h"
20 */
21 * @brief Data Structure to manage data associated to GPIO custom
22   peripheral
23 */
24 typedef struct{
25   gpio_custom_TypeDef* gpio_custom; /*!< Contains virtual address (page
26   offset included) and offset registers value of GPIO device uio*/
27   char* uiiod;                  /*!< uio device file corrisponding to GPIO */
28   int fd;                      /*!< Memory file descriptor */
29   int r_w;                     /*!< Saves operation's type to do with GPIO */
30   uint32_t r_or_w_value;        /*!< Saves read/write value from/on GPIO */
31 }mygpio_TypeDef;
32 */
33 /* GPIO port MACRO
34  -----
35 #define GPIO_PORT 0xF          /*!< GPIO ports selected to read/write */
36 */
37 /* Colour MACRO
38  -----
39 #define RESET    "\033[0m"    /*!< Reset colour */
40 #define GREEN    "\033[32m"    /*!< Green colour */
41 #define BOLDWHITE "\033[1m\033[37m" /*!< Bold White colour */

```

```

39  /* GPIO operation MACRO
40  -----*/
41 #define READ 1      /*!< Read operation is set */
42 #define WRITE 0     /*!< Write operation is set */
43
44  /* GPIO page MACRO
45  -----*/
46 #define PAGE_SIZE sysconf(_SC_PAGESIZE) /*!< Page size used by SO */
47
48  /* Function prototypes
49  -----*/
50 void mygpio_usage(char *name); /*!< usage function is called to help the
51 user */
52 int mygpio_parse_command(int argc,char **argv,mygpio_TypeDef* mygpio); /*!<
53 parse_command function is called to parse arguments passed to driver */
54 int mygpio_open_memory(mygpio_TypeDef* mygpio); /*!< mygpio_open_memory
55 function is called to open memory file */
56 void mygpio_read_gpio(mygpio_TypeDef* mygpio); /*!< mygpio_read_gpio
57 function is called to do read operation from GPIO */
58 void mygpio_write_gpio(mygpio_TypeDef* mygpio); /*!< mygpio_write_gpio
59 function is called to do write operation on GPIO */
60 void mygpio_close_memory(mygpio_TypeDef* mygpio);/*!< mygpio_close_memory
61 function is called to close memory file */

```

Codice 7.28: "mygpio uiodriver.h"

### 7.3.2.2 mygpio\_uiodriver.c

```

1 /**
2 ****
3 * @file      mygpio_uiodriver.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     7-July-2017
8 * @brief    Functions used for mygpio_uiodriver gpio
9 ****
10 */
11 /* Includes
12 -----*/
13 #include "mygpio_uiodriver.h"
14
15 /**
16 * @brief Describes how 'mygpio_uiodriver' must be used, specifying all
17 supported options.

```

```

16 * @param [in] name: Specifies mygpio_nodriver name.
17 * @retval None.
18 */
19 void mygpio_usage(char *name) {
20     printf("The right way to use this driver is\n");
21     printf(BOLDWHITE"%s -d <UIO_DEV_FILE> -i|-o <VALUE>\n"RESET, name);
22     printf("Please, pay attention:\n");
23     printf(BOLDWHITE"\t-d"RESET GREEN"\t<UIO_DEV_FILE>"RESET"\n\t\tThe
24         mandatory parameter <UIO_DEV_FILE>\n\t\tspecifies the uio device file
25         corrisponding\n\t\tto GPIO, for example /dev/uio0.\n");
26     printf(BOLDWHITE"\n\t-i"RESET"\n\t\tSpecifies the input value of GPIO \n\t
27         \tcorrisponding to <UIO_DEV_FILE> \n");
28     printf(BOLDWHITE"\n\t-o"RESET GREEN"\t<VALUE>"RESET"\n\t\tThe mandatory
29         parameter <VALUE> specify\n\t\tthe value must be write on GPIO output
30         \n");
31     return;
32     return;
33 }
34
35 /**
36 * @brief Parses mygpio_uiodriver arguments.
37 * @param [in] argc: number of parameters, passed to main function.
38 * @param [in] argv: parameters passed to main function
39 * @param [out] mygpio: data stucture that contains GPIO data
40 * @retval Integer status:
41 *      -1 An error occurred;
42 *      1 Help function is called;
43 *      0 Everything is ok.
44 */
45 int mygpio_parse_command(int argc, char **argv, mygpio_TypeDef* mygpio) {
46     int index=0;
47     int mandatory_opt=-1; /* Keep track if the mandatory option 'g' is
48                           parsed */
49     int i_or_o=0; /* Keep track if an i/o operation is requested */
50     static char *optstring = ":d:io:h"; /* Allowed parameters */
51
52     while((index = getopt(argc, argv, optstring)) != -1) {
53         switch(index) {
54             case 'd':
55                 mandatory_opt=atoi(optarg); /* Change mandatory_opt value to
56                                           memorize that the mandatory option is parsed */
57                 mygpio->uiod(optarg); /* Saves device uio path name */
58                 break;
59             case 'i':
60                 /* Check if before 'i' option was passed the mandatory option 'g'
61                  */
62                 if(mandatory_opt!=-1){
63                     mygpio->r_w=READ; /* Set READ operation for GPIO */
64                     i_or_o=1;
65                 }
66         }
67     }
68 }
```

```

57     }
58     else{
59         printf("Missing mandatory parameter 'g'\n");
60         mygpio_usage(argv[0]);
61         return -1;
62     }
63     break;
64 case 'o':
65     /* Check if before 'o' option was passed the mandatory option 'g'
66      */
67     if(mandatory_opt!=-1){
68         mygpio->r_w=WRITE; /* Set WRITE operation for GPIO */
69         mygpio->r_or_w_value=strtoul(optarg, NULL, 0);
70         i_or_o=1;
71     }
72     else{
73         printf("Missing mandatory parameter 'g'\n");
74         mygpio_usage(argv[0]);
75         return -1;
76     }
77     break;
78 case 'h':
79     mygpio_usage(argv[0]);
80     return 1;
81     break;
82 case ':':
83     printf("Missing argument for '%c' option\n", optopt);
84     mygpio_usage(argv[0]);
85     return -1;
86     break;
87 case '?':
88     printf("Option '%c' not recognized\n", optopt);
89     mygpio_usage(argv[0]);
90     return -1;
91     break;
92 default:
93     mygpio_usage(argv[0]);
94     return -1;
95     break;
96 }
97 }
98 /* Check if an i|o operation is requested */
99 if(i_or_o==0){
100     printf("Can't use \"BOLDWHITE\"nodriver \"RESET\"without specify i|o option\
101           ");
102     mygpio_usage(argv[0]);
103     return -1;
104 }
```

```

104     return 0;
105 }
106
107 /**
108 * @brief Opens /dev/mem file in order to access custom GPIO address and
109 * calculates page offset and virtual address of GPIO file.
110 * @param [inout] mygpio: data stucture that contains GPIO data
111 * @retval Integer status:
112 *          -1      An error occurred.
113 */
114 int mygpio_open_memory(mygpio_TypeDef* mygpio) {
115
115     mygpio->fd = open (mygpio->uiod, O_RDWR);
116     if (mygpio->fd < 1) {
117         perror("Error to open uio device file");
118         return -1;
119     }
120     /* mmap system call returns virtual page address of GPIO device uio */
121     void* vrt_add = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED,
122                         mygpio->fd, 0);
123     mygpio->gpio_custom->base_address=vrt_add; // virtual base address
124             extracted from mmap
125
126     return 0;
127 }
128
129 /**
130 * @brief Reads the <mygpio->r_or_w_value> in input to GPIO port.
131 * @param [inout] mygpio: data stucture that contains GPIO data
132 * @retval None.
133 */
134 void mygpio_read_gpio(mygpio_TypeDef* mygpio) {
135     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
136     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, READ);
137     mygpio->r_or_w_value= gpio_custom_GetValue(mygpio->gpio_custom,GPIO_PORT);
138     printf("The value on GPIO %s input is: %08x\n",mygpio->uiod,mygpio->
139           r_or_w_value);
140 }
141
142 /**
143 * @brief Writes <mygpio->r_or_w_value> in output to GPIO port.
144 * @param [inout] mygpio: data stucture that contains GPIO data
145 * @retval None.
146 */
147 void mygpio_write_gpio(mygpio_TypeDef* mygpio){
148     printf("Debug: do l'abilitazione al gpio\n");
149     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
150     printf("Debug: dico al gpio che è in scrittura\n");
151     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, WRITE);

```

```

148 printf("Going to write onto GPIO %s the value %08x\n", mygpio->uiod,
149     mygpio->r_or_w_value);
150 gpio_custom_SetValue(mygpio->gpio_custom, GPIO_PORT, LOW); // Cleans
151     previous values
152 gpio_custom_SetValue(mygpio->gpio_custom, mygpio->r_or_w_value, HIGH);
153 printf("Debug: ho finito di scrivere\n");
154 }
155 /**
156 * @brief Closes /dev/mem file.
157 * @param [in] mygpio: data stucture that contains GPIO data
158 * @retval None.
159 */
160 void mygpio_close_memory(mygpio_TypeDef* mygpio){
161     munmap((void*) (mygpio->gpio_custom->base_address), PAGE_SIZE);
162     close(mygpio->fd);
163 }
```

Codice 7.29: "mygpio uiodriver.c"

### 7.3.2.3 main.c

```

1 /**
2 ****
3 * @file      main.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     5-July-2017
8 * @brief    driver "mygpio_uiodriver" to control gpio
9 ****
10 */
11 /* Includes
12 -----------*/
12 #include "mygpio_uiodriver.h"
13
14 int main(int argc, char *argv[]){
15     printf("debug 1\n");
16     /*!< First step */
17     /*! Structure and variable declaration */
18     mygpio_TypeDef mygpio; /* Structure associated to GPIO custom peripheral
19     */
20     int ret_parse; /* Returned value from mygpio_parse_command function */
21     printf("debug 2\n");
22     /*!< Second step */
23 }
```

```

22  /*! mygpio_parse_command function is called to parse all argument passed
23   to driver mygpio_uiodriver */
24  /* Check if mygpio_parse_command function returns error */
25  if((ret_parse=mygpio_parse_command(argc,argv,&mygpio))==1)
26      return 0;
27  else
28      if(ret_parse== -1)
29          return -1;
30
31
32  printf("debug 3\n");
33  /*!< Third step */
34  /*! mygpio_open_memory function is called to achieve gpio virtual address
35   */
36  /* Check if mygpio_open_memory function returns error */
37  if(mygpio_open_memory(&mygpio)== -1){
38      printf("nodriver aborted!\n");
39      return -1;
40  }
41  //
42  /*!< Fourth step*/
43  /*! mygpio_read_gpio is called if r_w variable is READ, otherwise is
44   called write_gpio function */
45  if ((mygpio.r_w) == READ) mygpio_read_gpio(&mygpio);
46  else mygpio_write_gpio(&mygpio);
47  /*!< First step */
48  /*! mygpio_close_memory function is called to close memory file and delete
49   file descriptor */
50  mygpio_close_memory(&mygpio);

51
52  return 0;
53 }
```

Codice 7.30: "main.c"

### 7.3.3 Terza tipologia

In fig. 7.20 si osserva il design project di riferimento, mentre i base address sono identici alle altre due tipologie di driver.

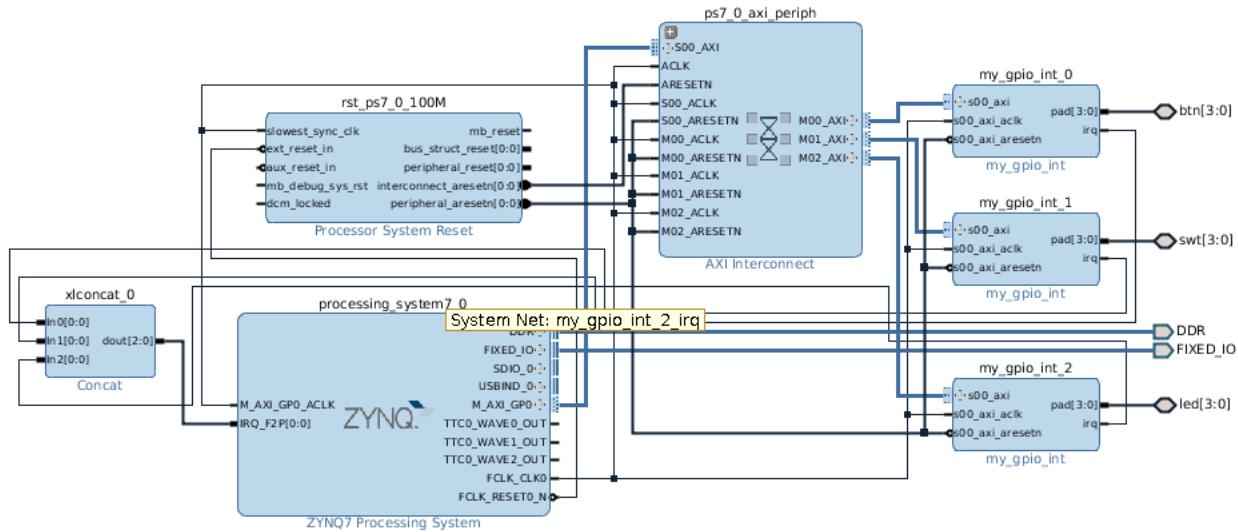


Figura 7.20: Design project GPIO custom con interrupt

### 7.3.3.1 mygpio\_uiointdriver.c

```

1 /**
2 ****
3 * @file      mygpio_uiointdriver.c
4 * @authors   Colella Gianni - Guida Ciro - Lombardi Daniele / Group IV -
5 *            Sistemi Embedded 2016-2017
6 * @version   V1.0
7 * @date     7-July-2017
8 * @brief    Functions used for mygpio_uiointdriver gpio
9 ****
10 */
11 /* Includes
12  -----
13 #include "mygpio_uiointdriver.h"
14 /**
15 * @brief Describes how 'mygpio_uiodriver' must be used, specifying all
16 *        supported options.
17 * @param [in] name: Specifies mygpio_nodriver name.
18 * @retval None.
19 */
20 void mygpio_usage(char *name) {
21     printf("The right way to use this driver is\n");
22     printf(BOLDWHITE"%s -d <UIO_DEV_FILE> -i|-o <VALUE>\n"RESET, name);
23     printf("Please, pay attention:\n");

```

```

23   printf(BOLDWHITE"\t-d"RESET GREEN"\t<UIO_DEV_FILE>"RESET"\n\t\tThe
24     mandatory parameter <UIO_DEV_FILE>\n\t\tspecifies the uio device file
25     corrisponding\n\t\tto GPIO, for example /dev/uio0.\n");
26   printf(BOLDWHITE"\n\t-i"RESET"\n\t\tSpecifies the input value of GPIO \n\t
27     \tcorrisponding to <UIO_DEV_FILE> \n");
28   printf(BOLDWHITE"\n\t-o"RESET GREEN"\t<VALUE>"RESET"\n\t\tThe mandatory
29     parameter <VALUE> specify\n\t\tthe value must be write on GPIO output
30     \n");
31   return;
32   return;
33 }
34
35 /**
36  * @brief Parses nodriver arguments.
37  * @param [in] argc: number of parameters, passed to main function.
38  * @param [in] argv: parameters passed to main function
39  * @param [out] mygpio: data stucture that contains GPIO data
40  * @retval Integer status:
41  *         -1      An error occurred;
42  *         1      Help function is called;
43  *         0      Everything is ok.
44 */
45 int mygpio_parse_command(int argc, char **argv, mygpio_TypeDef* mygpio) {
46   int index=0;
47   int mandatory_opt=-1;           /* Keep track if the mandatory option 'g' is
48                                parsed */
49   int i_or_o=0;                  /* Keep track if an i/o operation is requested */
50   static char *optstring = ":d:io:h"; /* Allowed parameters */
51
52   while((index = getopt(argc, argv, optstring)) != -1) {
53     switch(index) {
54     case 'd':
55       mandatory_opt=atoi(optarg); /* Change mandatory_opt value to
56                                memorize that the mandatory option is parsed */
57       mygpio->uiod(optarg);    /* Saves device uio path name */
58       break;
59     case 'i':
60       /* Check if before 'i' option was passed the mandatory option 'g'
61        */
62       if(mandatory_opt!=-1){
63         mygpio->r_w=READ; /* Set READ operation for GPIO */
64         i_or_o=1;
65       }
66       else{
67         printf("Missing mandatory parameter 'g'\n");
68         mygpio_usage(argv[0]);
69         return -1;
70       }
71       break;
72     }
73   }
74 }
```

```

64     case 'o':
65         /* Check if before 'o' option was passed the mandatory option 'g'
66          */
67         if(mandatory_opt!=-1){
68             mygpio->r_w=WRITE; /* Set WRITE operation for GPIO */
69             mygpio->r_or_w_value=strtoul(optarg, NULL, 0);
70             i_or_o=1;
71         }
72         else{
73             printf("Missing mandatory parameter 'g'\n");
74             mygpio_usage(argv[0]);
75             return -1;
76         }
77         break;
78     case 'h':
79         mygpio_usage(argv[0]);
80         return 1;
81         break;
82     case ':':
83         printf("Missing argument for '%c' option\n", optopt);
84         mygpio_usage(argv[0]);
85         return -1;
86         break;
87     case '?':
88         printf("Option '%c' not recognized\n", optopt);
89         mygpio_usage(argv[0]);
90         return -1;
91         break;
92     default:
93         mygpio_usage(argv[0]);
94         return -1;
95         break;
96     }
97
98     /* Check if an i|o operation is requested */
99     if(i_or_o==0){
100         printf("Can't use \"BOLDWHITE\"nodriver \"RESET\"without specify i|o option\
101           ");
102         mygpio_usage(argv[0]);
103         return -1;
104     }
105     return 0;
106 }
107 /**
108 * @brief Opens /dev/mem file in order to access custom GPIO address and
109 *        calculates page offset and virtual address of GPIO file.
110 * @param [inout] mygpio: data stucture that contains GPIO data

```

```

110 * @retval Integer status:
111 *          -1      An error occurred.
112 */
113 int mygpio_open_memory(mygpio_TypeDef* mygpio) {
114
115     mygpio->fd = open (mygpio->uiod, O_RDWR);
116     if (mygpio->fd < 1) {
117         perror("Error to open uio device file");
118         return -1;
119     }
120     /* mmap system call returns virtual page address of GPIO device uio */
121     void* vrt_add = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED,
122                          mygpio->fd, 0);
123     mygpio->gpio_custom->base_address=vrt_add; // virtual base address
124                          extracted from mmap
125
126     return 0;
127 }
128
129 /**
130 * @brief Reads the <mygpio->r_or_w_value> in input to GPIO port.
131 * @param [inout] mygpio: data stucture that contains GPIO data
132 * @retval None.
133 */
134 int mygpio_read_gpio(mygpio_TypeDef* mygpio) {
135
136     /* number of bytes read */
137     ssize_t num_byte;
138     /* to test if read operation is correct */
139     uint32_t test_read=1;
140
141     /* Set PAD_EN register to enable read operation from GPIO port */
142     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
143     /* Set PAD_RW_N register of GPIO to configure the GPIO port as input */
144     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, READ);
145
146     while(1) {
147         /* Enable Global Interrupt of GPIO */
148         gpio_custom_SetGlobalInterrupt(mygpio->gpio_custom, HIGH);
149
150         /* Enable Interrupt on all port of GPIO*/
151         gpio_custom_SetInterruptMask(mygpio->gpio_custom, GPIO_PORT, HIGH);
152
153         /* to suspend the process on GPIO peripheral */
154         num_byte=read(mygpio->fd, &test_read, sizeof(test_read));
155         if(num_byte!=sizeof(test_read)) {
156             printf("It occurs an error on read opeartion\n");
157             return -1;
158         }

```

```

157     else{
158         /* Disable Global Interrupt of GPIO */
159         gpio_custom_SetGlobalInterrupt(mygpio->gpio_custom, LOW);
160         /* Disable Interrupt on all port of GPIO*/
161         gpio_custom_SetInterruptMask(mygpio->gpio_custom, GPIO_PORT, LOW);
162
163         /* Memorize the value from GPIO port in <mygpio->r_or_w_value> */
164         mygpio->r_or_w_value = gpio_custom_GetValue(mygpio->gpio_custom,
165             GPIO_PORT);
166         printf("The value in input to GPIO %s is: %08x\n",mygpio->uiod,mygpio
167             ->r_or_w_value);
168     }
169
170     /* In this case the driver don't read the register until the input that
171      cause the interrupt is set to default low position */
172 //    while(*(unsigned*)gpio_custom_GetValue(mygpio->gpio_custom, GPIO_PORT)
173 //        !=0);
174
175     /* test reenable interrupt */
176     ssize_t re_enable = 1;
177     /* to clean interrupt */
178     re_enable = write(mygpio->fd, &re_enable, sizeof(re_enable));
179
180     if (re_enable < sizeof(re_enable)) {
181         perror("write");
182         close(mygpio->fd);
183         exit(EXIT_FAILURE);
184     }
185     /* Sent ack to GPIO peripheral */
186
187     gpio_custom_SetAckInterrupt(mygpio->gpio_custom,
188         gpio_custom_GetStatusInterrupt(mygpio->gpio_custom, GPIO_PORT), HIGH
189     );
190
191     return 1;
192 }
193
194 /**
195  * @brief Writes <mygpio->r_or_w_value> in output to GPIO port.
196  * @param [inout] mygpio: data stucture that contains GPIO data
197  * @retval None.
198 */
199 void mygpio_write_gpio(mygpio_TypeDef* mygpio){
200     gpio_custom_SetEnable(mygpio->gpio_custom, GPIO_PORT, HIGH);
201     gpio_custom_SetMode(mygpio->gpio_custom, GPIO_PORT, WRITE);
202     printf("Going to write onto GPIO %s the value %08x\n", mygpio->uiod,
203         mygpio->r_or_w_value);
204     gpio_custom_SetValue(mygpio->gpio_custom, GPIO_PORT, LOW); // Cleans
205     previous values

```

```
198     gpio_custom_SetValue(mygpio->gpio_custom, mygpio->r_or_w_value, HIGH);
199 }
200
201 /**
202 * @brief Closes /dev/mem file.
203 * @param [in] mygpio: data stucture that contains GPIO data
204 * @retval None.
205 */
206 void mygpio_close_memory(mygpio_TypeDef* mygpio) {
207     munmap((void*) (mygpio->gpio_custom->base_address), PAGE_SIZE);
208     close(mygpio->fd);
209 }
```

Codice 7.31: "mygpio uiointdriver.c"

# Capitolo 8

# FreeRTOS

## 8.1 Traccia

Si implementi una versione del Sistema Operativo RealTime **FreeRTOS** su board STMicroelettronico STM32F4Discovery e, implementando vari task con priorità diverse, si facciano valutazioni sul meccanismo di scheduling dei vari task.

## 8.2 Procedimento

Inizialmente con l'utilizzo del software STM32CubeMX si configura la board STM32F4Discovery come in fig. 8.1.

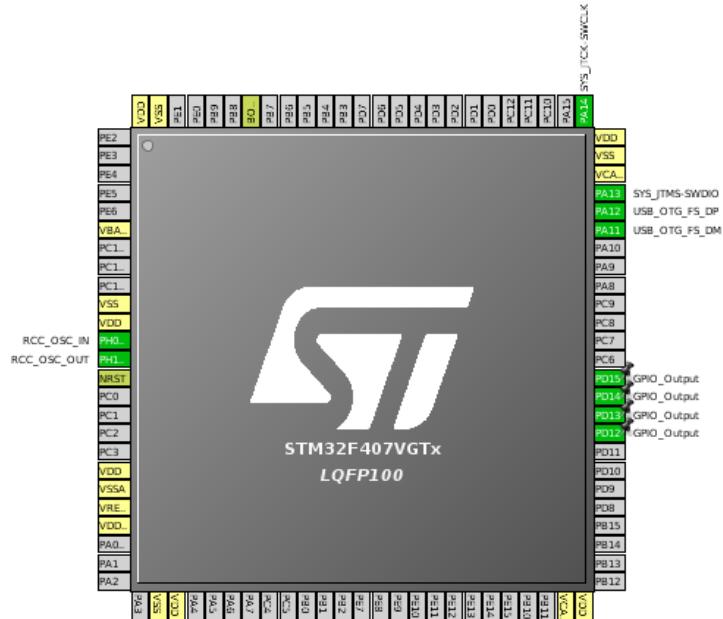


Figura 8.1: Configurazione della STM32F4Discovery per il SO FreRTOS

Si creano, dunque, 4 Task. Ciascuno ha il compito di effettuare il toggle di un LED: il Task 1 fa il toggle sul LED 5, il Task 2 sul LED 4, il Task 3 sul LED 5 e il Task 4 sul LED 6.

Per semplicità si posta il solo codice del Task 1 in quanto i 4 Task sono equivalenti.

```

126 /*Task 1 */
127 /**
128 * @brief Task 1, in cui si fa un Toggle del LED6 ogni DELAY1 ms
129 *       Ogni volta che è richiamato questo Task, si invia una CDC Transmit
130 *       "TASK ---1 che è utile per un debug da terminale
131 * @note pvParameters
132 * @retval None
133 *      */
134 static void Task1(void *pvParameters) {
135     char testo[]="Task---1\r\n";
136     for (;;) {
137         osDelay(DELAY1);
138         HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_15);
139         CDC_Transmit_FS((uint8_t*)testo,strlen(testo));
140     }
141 }
```

Codice 8.1: "funzione Task 1.c"

I Task vengono definiti all'interno del main, come mostrato nel seguente codice.

```

1  /* Includes
2   *-----*/
3 #include "main.h"
4 #include "stm32f4xx_hal.h"
5 #include "cmsis_os.h"
6 #include "usb_device.h"
7
8 osThreadId defaultTaskHandle;
9
10 void SystemClock_Config(void);
11 static void MX_GPIO_Init(void);
12
13 static void Task1(void *pvParameters) ;
14 static void Task2(void *pvParameters) ;
15 static void Task3(void *pvParameters) ;
16 static void Task4(void *pvParameters) ;
17
18 static const char *pcTextForTask1 ="Task 1 is running\r\n";
19 static const char *pcTextForTask2 ="Task 2 is running\r\n";
20 static const char *pcTextForTask3 ="Task 3 is running\r\n";
21 static const char *pcTextForTask4 ="Task 4 is running\r\n";
22
23 #define PRIORITY1 1
24 #define PRIORITY2 2
25 #define PRIORITY3 3
```

```

26 #define PRIORITY4 4
27 #define STACK_DIM 200
28
29 #define DELAY1 450
30 #define DELAY2 450
31 #define DELAY3 450
32 #define DELAY4 450
33
34 int main(void)
35 {
36     HAL_Init();
37     /* Configure the system clock */
38     SystemClock_Config();
39     /* Initialize all configured peripherals */
40     MX_GPIO_Init();
41     /*Initialize USB*/
42     MX_USB_DEVICE_Init();
43
44     /* Creazione dei 4 Task */
45     xTaskCreate(Task1, "Task1", STACK_DIM, (void*)pcTextForTask1, PRIORITY1, NULL);
46     xTaskCreate(Task2, "Task2", STACK_DIM, (void*)pcTextForTask2, PRIORITY2, NULL);
47     xTaskCreate(Task3, "Task3", STACK_DIM, (void*)pcTextForTask3, PRIORITY3, NULL);
48     xTaskCreate(Task4, "Task4", STACK_DIM, (void*)pcTextForTask4, PRIORITY4, NULL);
49
50     vTaskStartScheduler();
51
52     /* Start scheduler */
53     osKernelStart();
54
55     while (1) {
56     }
57
58 }
```

Codice 8.2: "main.c"

Allo scopo di analizzare l'algoritmo di scheduling di FreeRTOS, vengono implementati diversi esempi in cui può variare:

- il numero dei task;
- il loro valore di priorità;
- il delay di ogni task.

## 8.2.1 Esempi con 2 Task

### 8.2.1.1 Esempio 1

Nel primo esempio si implementano soltanto 2 task: Task1 e Task 2. A questi vengono assegnati due valori uguali sia di priorità, che di delay, come descritto in tab. 8.1. La fig. 8.2 generata con il

tool **OpenRTos Viewer** mostra informazioni aggiuntive sul task, in particolare si può osservare il Nome, la Priorità, e l'indirizzamento sullo stack.

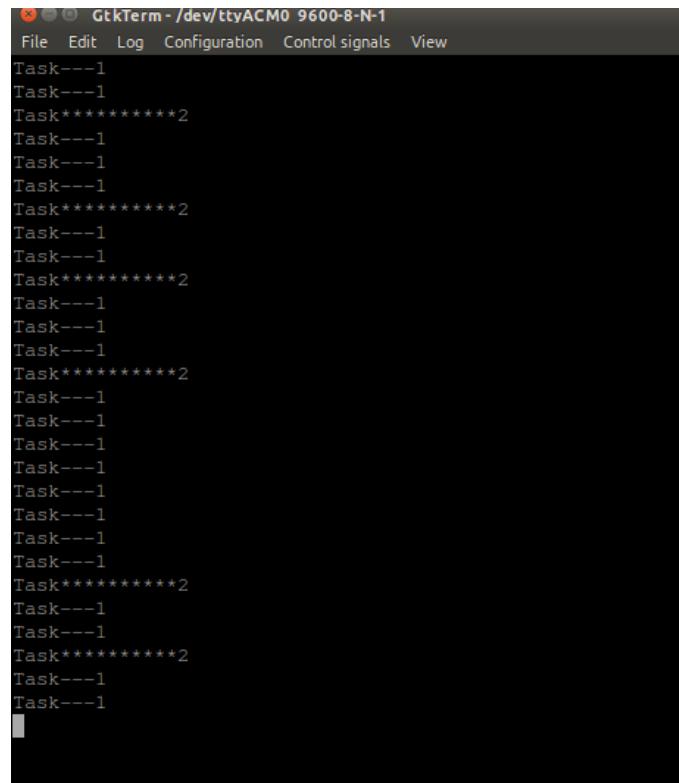
Task	Priorità	Delay
Task 1	1	450 ms
Task 2	1	450 ms

Tabella 8.1: Tabella riassuntiva dei task utilizzando nell'esempio n.1

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	1/1	0x200006f0	0x200009c4	RUNNING	None	Off	Unknown

Figura 8.2: Task-Table

In fig. 8.3 si mostra ciò che è stato prodotto utilizzando il terminale **gtkterm** sulla porta seriale ACM0.



The screenshot shows a terminal window titled "GtkTerm - /dev/ttyACM0 9600-8-N-1". The window has a menu bar with File, Edit, Log, Configuration, Control signals, and View. The main area of the terminal displays a continuous stream of text output. The output consists of several lines of "Task---1" followed by two lines of "Task\*\*\*\*\*2". This pattern repeats multiple times, indicating periodic task execution. The terminal window has a dark background and light-colored text.

Figura 8.3: Schermata del terminale per esempio n.1

### 8.2.1.2 Esempio 2

In questo secondo esempio si modifica la priorità al Task 2, aumentandola da 1 a 2. La tabella 8.2 riassume la configurazione utilizzata per il secondo esempio.

Task	Priorità	Delay
Task 1	1	450 ms
Task 2	2	450 ms

Tabella 8.2: Tabella riassuntiva dei task utilizzando nell'esempio n.2

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	2/2	0x200006f0	0x200009c4	RUNNING	None	Off	Unknown

Figura 8.4: Task-Table per l'esempio n.2

In fig. 8.5 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.

```
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task---1
Task*****2
Task---1
Task*****2
Task*****2
Task*****2
Task*****2
Task---1
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task---1
Task*****2
```

Figura 8.5: Schermata del terminale per esempio n.2

### 8.2.1.3 Esempio 3

Il terzo esempio presenta 2 Task. Al task 2 viene modificato il valore di Delay, che passa da 450 a 550. La tabella 8.3 riassume la configurazione utilizzata.

Task	Priorità	Delay
Task 1	1	450 ms
Task 2	1	550 ms

Tabella 8.3: Tabella riassuntiva dei task utilizzando nell'esempio n.3

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	1/1	0x200006f0	0x200009c4	RUNNING	None	Off	Unknown

Figura 8.6: Task-Table per l'esempio n.3

In fig. 8.7 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.

Figura 8.7: Schermata del terminale per esempio n.3

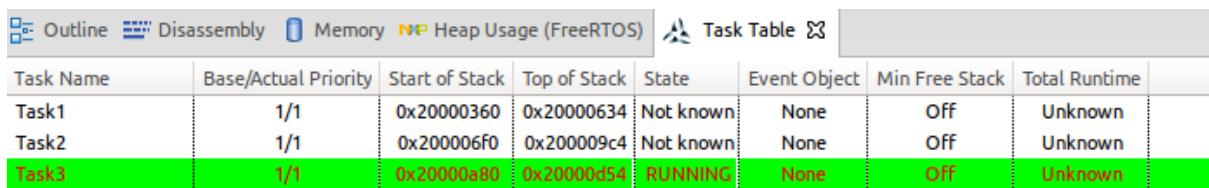
## 8.2.2 Esempi con 3 Task

### 8.2.2.1 Esempio 4

In questo esempio, si aggiunge un'ulteriore task e si riportano i risultati ottenuti da tale esperimento. La tabella 8.4 riassume la configurazione utilizzata.

Task	Priorità	Delay
Task 1	1	350 ms
Task 2	1	350 ms
Task 3	1	150 ms

Tabella 8.4: Tabella riassuntiva dei task utilizzando nell'esempio n.4



Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	1/1	0x200006f0	0x200009c4	Not known	None	Off	Unknown
Task3	1/1	0x20000a80	0x20000d54	RUNNING	None	Off	Unknown

Figura 8.8: Task-Table per l'esempio n.4

In fig. 8.9 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.

```

task---1
task---1
task*****2
task-----3
task---1
task*****2
task---1
task*****2
task---1
task-----3
task---1
task-----3
task---1
task-----3
task---1
task---1
task---1
task*****2
task---1
task*****2
task---1
task*****2
task---1
task-----3
task---1
task-----3
task---1
task---1
task---1
task---1

```

Figura 8.9: Schermata del terminale per esempio n.4

### 8.2.2.2 Esempio 5

Qui si modifica la priorità dei Task 2 e Task 3 e si riportano i risultati ottenuti. La tabella 8.5 riassume la configurazione utilizzata.

Task	Priorità	Delay
Task 1	1	350 ms
Task 2	3	350 ms
Task 3	6	350 ms

Tabella 8.5: Tabella riassuntiva dei task utilizzando nell'esempio n.5

Task Table								
Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime	
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown	
Task2	3/3	0x200006f0	0x200009c4	Not known	None	Off	Unknown	
Task3	6/6	0x20000a80	0x20000d54	RUNNING	None	Off	Unknown	

Figura 8.10: Task-Table per l'esempio n.5

In fig. 8.11 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.

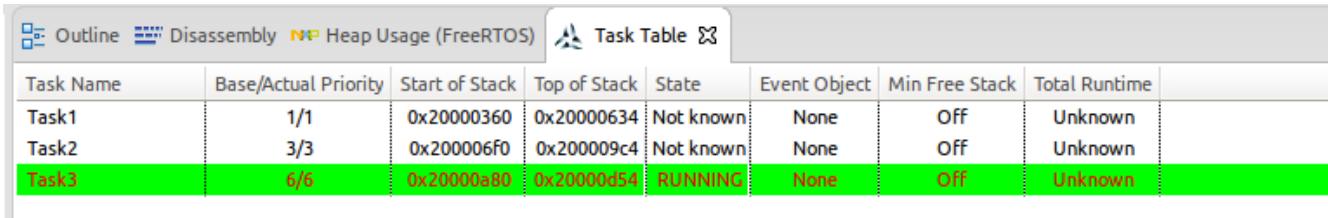
Figura 8.11: Schermata del terminale per esempio n.5

### 8.2.2.3 Esempio 6

Rispetto all'esempio precedente si modifica il Delay del task 2 e si riportano i risultati ottenuti. La tabella 8.6 riassume la configurazione utilizzata. Si nota che il comportamento è molto simile a quello dell'esempio 8.2.2.2.

Task	Priorità	Delay
Task 1	1	350 ms
Task 2	3	150 ms
Task 3	6	350 ms

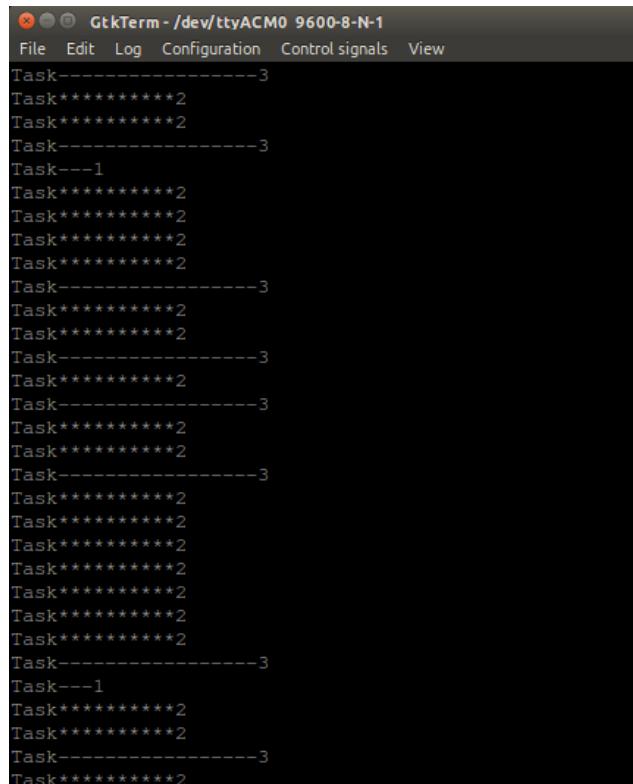
Tabella 8.6: Tabella riassuntiva dei task utilizzando nell'esempio n.6



Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	3/3	0x200006F0	0x200009c4	Not known	None	Off	Unknown
Task3	6/6	0x20000a80	0x20000d54	RUNNING	None	Off	Unknown

Figura 8.12: Task-Table per l'esempio n.6

In fig. 8.13 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.



```
GtkTerm - /dev/ttyACM0 9600-8-N-1
File Edit Log Configuration Control signals View
Task-----3
Task*****2
Task*****2
Task-----3
Task---1
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task-----3
Task*****2
Task*****2
Task*****2
Task-----3
Task*****2
Task-----3
Task*****2
Task*****2
Task-----3
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task-----3
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task*****2
Task-----3
Task---1
Task*****2
Task*****2
Task-----3
Task*****2
```

Figura 8.13: Schermata del terminale per esempio n.6

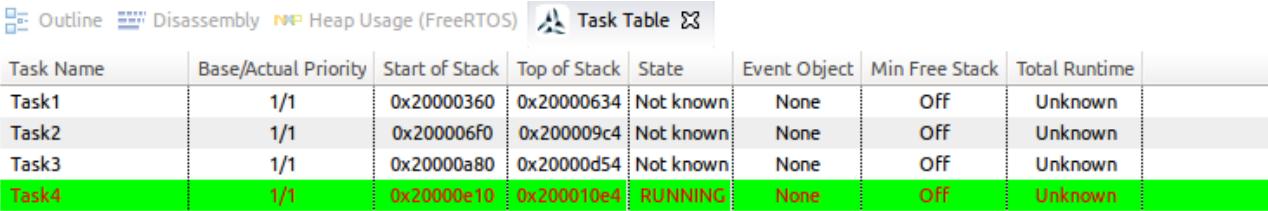
### 8.2.3 Esempi con 4 Task

#### 8.2.3.1 Esempio 7

Si aggiunge un ulteriore Task, in questo caso tutti i Task hanno lo stesso livello di priorità. La tabella 8.7 riassume la configurazione utilizzata.

Task	Priorità	Delay
Task 1	1	350 ms
Task 2	1	350 ms
Task 3	1	350 ms
Task 4	1	350 ms

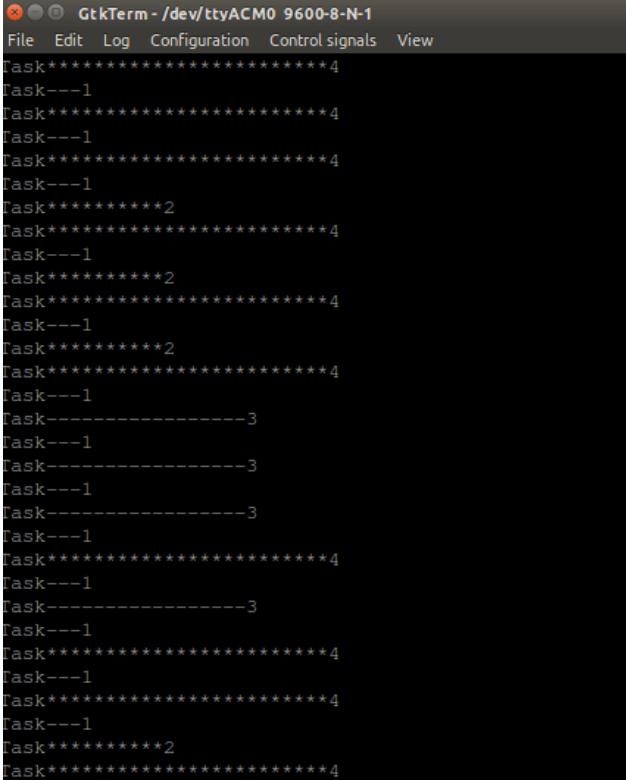
Tabella 8.7: Tabella riassuntiva dei task utilizzati nell'esempio n.7



Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	1/1	0x200006f0	0x200009c4	Not known	None	Off	Unknown
Task3	1/1	0x20000a80	0x20000d54	Not known	None	Off	Unknown
Task4	1/1	0x20000e10	0x200010e4	RUNNING	None	Off	Unknown

Figura 8.14: Task-Table per l'esempio n.7

In fig. 8.15 si mostra ciò che è stato prodotto utilizzando il terminale gtkterm sulla porta seriale ACM0.



```
Task*****4
Task--1
Task*****4
Task--1
Task*****4
Task--1
Task*****2
Task*****4
Task--1
Task*****2
Task*****4
Task--1
Task*****2
Task*****4
Task--1
Task-----3
Task--1
Task-----3
Task--1
Task-----3
Task--1
Task*****4
Task--1
Task-----3
Task--1
Task*****4
Task--1
Task*****4
Task--1
Task*****2
Task*****4
```

Figura 8.15: Schermata del terminale per esempio n.7

### 8.2.3.2 Esempio 8

La tabella 8.8 riassume la configurazione utilizzata per il primo esempio.

Così come accade per l'esempio 8.2.2.3, anche con 4 task si ripropone un comportamento equivalente.

Task	Priorità	Delay
Task 1	1	450 ms
Task 2	2	450 ms
Task 3	3	450 ms
Task 4	4	450 ms

Tabella 8.8: abella riassuntiva dei task utilizzati nell'esempio n.8

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime	Delta Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown	-
Task2	2/2	0x200006f0	0x200009c4	Not known	None	Off	Unknown	-
Task3	3/3	0x20000a80	0x20000d54	Not known	None	Off	Unknown	-
Task4	4/4	0x20000e10	0x200010e4	RUNNING	None	Off	Unknown	-

Figura 8.16: Task-Table per l'esempio n.8

In fig. 8.17 si mostra ciò che è stato prodotto.

```

GtkTerm - /dev/ttyACM0 9600-8-N-1
File Edit Log Configuration Control signals View
Task-----3
Task*****2
Task---1
Task*****4
Task-----3
Task*****2

```

Figura 8.17: Schermata del terminale per esempio n.8

### 8.2.3.3 Esempio 9

La tabella 8.9 riassume la configurazione utilizzata per il successivo esempio preso in considerazione. Si nota dalle Tabelle 8.18 che, quando è stato creato il Task 2 con Priorità 6, questo entra nello stato **running**. Successivamente viene creato il Task 3 attraverso la funzione

```
xTaskCreate(Task3, "Task3", STACK_DIM, (void*)pcTextForTask3, PRIORITY3, NULL)
```

Questo ha priorità pari a 3, viene definito, ma non passa nello stato di running.

In tab. 8.9 si mostra lo stato della Task Table quando viene eseguita la seguente linea di codice:

```
xTaskCreate(Task4, "Task4", STACK_DIM, (void*)pcTextForTask4, PRIORITY4, NULL);
```

In questo caso visto che il Task 4 ha una priorità pari alla priorità del Task in running, allora viene schedulato e il task 4 passa subito nello stato di running.

Task	Priorità	Delay
Task 1	1	350 ms
Task 2	6	450 ms
Task 3	3	250 ms
Task 4	6	150 ms

Tabella 8.9: tabella riassuntiva dei task utilizzati nell'esempio n.9

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	6/6	0x200006f0	0x200009c4	RUNNING	None	Off	Unknown
Task3	3/3	0x20000a80	0x20000d54	Not known	None	Off	Unknown

Figura 8.18: Task-Table 1 per l'esempio n.9

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	1/1	0x20000360	0x20000634	Not known	None	Off	Unknown
Task2	6/6	0x200006f0	0x200009c4	Not known	None	Off	Unknown
Task3	3/3	0x20000a80	0x20000d54	Not known	None	Off	Unknown
Task4	6/6	0x20000e10	0x200010e4	RUNNING	None	Off	Unknown

Figura 8.19: Task-Table per l'esempio n.9

In fig. 8.20 si mostra il risultato prodotto dai task.

Figura 8.20: Schermata del terminale per esempio n.9

#### 8.2.3.4 Esempio 10

La tabella 8.10 riassume la configurazione utilizzata per l'ultimo esempio. A differenza dei precedenti, si nota che ponendo il Task 1 a priorità maggiore rispetto a tutti gli altri, esso permane nello stato **running**, finché non arriva un nuovo task con priorità uguale o maggiore.

Task	Priorità	Delay
Task 1	6	350 ms
Task 2	5	450 ms
Task 3	3	250 ms
Task 4	1	150 ms

Tabella 8.10: abella riassuntiva dei task utilizzati nell'esempio n.10

Task Name	Base/Actual Priority	Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Total Runtime
Task1	6/6	0x20000360	0x20000634	RUNNING	None	Off	Unknown
Task2	5/5	0x200006f0	0x200009c4	Not known	None	Off	Unknown
Task3	3/3	0x20000a80	0x20000d54	Not known	None	Off	Unknown
Task4	1/1	0x20000e10	0x200010e4	Not known	None	Off	Unknown

Figura 8.21: Task-Table per l'esempio n.10

In fig. 8.22 si mostra il risultato ottenuto.

```

GtkTerm - /dev/ttyACM0 9600-8-N-1
File Edit Log Configuration Control signals View
Task*****4
Task-----3
Task*****4
Task-----3
Task*****2
Task*****2
Task--1
Task-----3
Task---1
Task*****2
Task-----3
Task*****4
Task*****4
Task*****2
Task-----3
Task-----3
Task---1
Task*****4
Task*****4
Task-----3
Task---1
Task*****4
Task-----3
Task*****4
Task---1
Task-----3
Task*****2
Task*****4
Task-----3
Task---1

```

Figura 8.22: Schermata del terminale per esempio n.10

### 8.3 Considerazioni

Lo scheduler è una delle componenti principali del Kernel di ogni Sistema Operativo, esso può sospendere e successivamente riprendere un'attività molte volte durante la vita del Task. Dagli esempi riportati si nota che FreeRtos schedula i Task in base al valore di priorità. Lo schedulatore è, quindi, classificabile come Priority based Preemptive Scheduler.

In fig. 8.23, si mostra dettagliatamente il comportamento dello scheduler quando 3 task vogliono accedere alla stessa periferica.

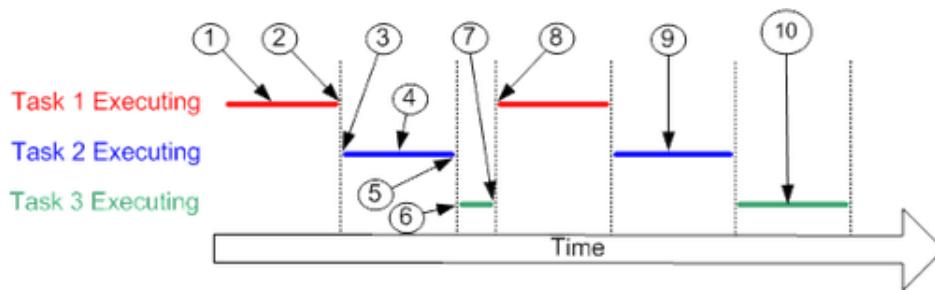


Figura 8.23: Schema di Scheduling di FreeRTOS

1. Il task 1 è in esecuzione;
2. Il kernel sospende il task 1 perché è arrivato il task 2 con priorità maggiore/uguale;
3. Si effettua il context switch tra il task 1 e il task 2;
4. Il task 2 è in esecuzione;
5. Il kernel sospende il task 2 perché è arrivato il task 3 con priorità maggiore/uguale;
6. Si effettua il context switch tra i task 2 e 3;
7. Il task 3 tenta di accedere alla stessa periferica del processore cui già aveva avuto accesso task 1 prima di essere bloccato;
8. Il task 1 viene risvegliato e completa la sua attività;
9. Viene effettuato un context switch tra i task 1 e 2, il quale completa la sua attività;
10. Successivamente il task 3 viene risvegliato e ha la periferica libera e quindi finché non viene sospeso di nuovo dal kernel;

Come si evince principalmente dagli esempi 8.2.3.3 e 8.2.3.4, viene eseguito un context switch o quando il task va in Delay oppure quando viene creato un task con un valore di priorità pari almeno al valore di priorità del task che in quel momento è in esecuzione.