

# Zynq-7000 Driver Pack

## 3.2

Generato da Doxygen 1.8.8

Mer 5 Lug 2017 11:48:31



# Indice

<b>1</b>	<b>Indice dei moduli</b>	<b>1</b>
1.1	Moduli . . . . .	1
<b>2</b>	<b>Indice delle strutture dati</b>	<b>3</b>
2.1	Strutture dati . . . . .	3
<b>3</b>	<b>Indice dei file</b>	<b>5</b>
3.1	Elenco dei file . . . . .	5
<b>4</b>	<b>Documentazione dei moduli</b>	<b>7</b>
4.1	Bare-metal . . . . .	7
4.1.1	Descrizione dettagliata . . . . .	9
4.1.2	Documentazione delle definizioni . . . . .	9
4.1.2.1	myGPIO_GIES_OFFSET . . . . .	9
4.1.2.2	myGPIO_IACK_OFFSET . . . . .	9
4.1.2.3	myGPIO_IRQ_OFFSET . . . . .	9
4.1.2.4	myGPIO_MODE_OFFSET . . . . .	9
4.1.2.5	myGPIO_PIE_OFFSET . . . . .	9
4.1.2.6	myGPIO_pin . . . . .	9
4.1.2.7	myGPIO_READ_OFFSET . . . . .	10
4.1.2.8	myGPIO_WRITE_OFFSET . . . . .	10
4.1.3	Documentazione dei tipi enumerati . . . . .	10
4.1.3.1	myGPIO_mask . . . . .	10
4.1.3.2	myGPIO_mode . . . . .	11
4.1.3.3	myGPIO_value . . . . .	11
4.1.4	Documentazione delle funzioni . . . . .	11
4.1.4.1	myGPIO_EnabledPinInterrupt . . . . .	11
4.1.4.2	myGPIO_GetRead . . . . .	11
4.1.4.3	myGPIO_GetValue . . . . .	12
4.1.4.4	myGPIO_GlobalInterruptDisable . . . . .	12
4.1.4.5	myGPIO_GlobalInterruptEnable . . . . .	12
4.1.4.6	myGPIO_Init . . . . .	13

4.1.4.7	myGPIO_IsGlobalInterruptEnabled	13
4.1.4.8	myGPIO_PendingInterrupt	13
4.1.4.9	myGPIO_PendingPinInterrupt	13
4.1.4.10	myGPIO_PinInterruptAck	14
4.1.4.11	myGPIO_PinInterruptDisable	14
4.1.4.12	myGPIO_PinInterruptEnable	14
4.1.4.13	myGPIO_SetMode	14
4.1.4.14	myGPIO_SetValue	15
4.1.4.15	myGPIO_Toggle	15
4.2	HD44780	16
4.2.1	Descrizione dettagliata	17
4.2.2	Documentazione dei tipi enumerati	18
4.2.2.1	HD44780_Direction_t	18
4.2.2.2	HD44780_InterfaceMode_t	18
4.2.3	Documentazione delle funzioni	18
4.2.3.1	HD44780_Clear	18
4.2.3.2	HD44780_CursorBlink	18
4.2.3.3	HD44780_CursorOff	19
4.2.3.4	HD44780_CursorOn	19
4.2.3.5	HD44780_DisplayOff	19
4.2.3.6	HD44780_Home	19
4.2.3.7	HD44780_Init4	19
4.2.3.8	HD44780_Init8	20
4.2.3.9	HD44780_MoveCursor	21
4.2.3.10	HD44780_MoveToRow1	21
4.2.3.11	HD44780_MoveToRow2	21
4.2.3.12	HD44780_Print	21
4.2.3.13	HD44780_printBinary32	22
4.2.3.14	HD44780_printBinary64	22
4.2.3.15	HD44780_printBinary8	22
4.2.3.16	HD44780_Printc	23
4.2.3.17	HD44780_printHex32	23
4.2.3.18	HD44780_printHex64	23
4.2.3.19	HD44780_printHex8	23
4.3	Zybo	24
4.3.1	Descrizione dettagliata	24
4.4	Button	25
4.4.1	Descrizione dettagliata	25
4.4.2	Documentazione delle definizioni	26
4.4.2.1	ZyboButton	26

4.4.2.2	ZyboButton_DebounceWait	26
4.4.3	Documentazione dei tipi enumerati	26
4.4.3.1	ZyboButton_mask_t	26
4.4.3.2	ZyboButton_status_t	26
4.4.4	Documentazione delle funzioni	26
4.4.4.1	ZyboButton_getStatus	26
4.4.4.2	ZyboButton_init	27
4.4.4.3	ZyboButton_waitWhileBusy	28
4.4.4.4	ZyboButton_waitWhileIdle	28
4.5	Led	29
4.5.1	Descrizione dettagliata	29
4.5.2	Documentazione delle definizioni	29
4.5.2.1	ZyboLed	29
4.5.3	Documentazione dei tipi enumerati	30
4.5.3.1	ZyboLed_mask_t	30
4.5.3.2	ZyboLed_status_t	30
4.5.4	Documentazione delle funzioni	30
4.5.4.1	ZyboLed_init	30
4.5.4.2	ZyboLed_setStatus	31
4.5.4.3	ZyboLed_toggle	31
4.6	Switch	33
4.6.1	Descrizione dettagliata	33
4.6.2	Documentazione delle definizioni	33
4.6.2.1	ZyboSwitch	33
4.6.3	Documentazione dei tipi enumerati	34
4.6.3.1	ZyboSwitch_mask_t	34
4.6.3.2	ZyboSwitch_status_t	34
4.6.4	Documentazione delle funzioni	34
4.6.4.1	ZyboSwitch_getStatus	34
4.6.4.2	ZyboSwitch_init	35
4.7	MyGPIO	36
4.7.1	Descrizione dettagliata	36
4.8	Linux-Driver	37
4.8.1	Descrizione dettagliata	39
4.8.2	Documentazione delle definizioni	41
4.8.2.1	DRIVER_FNAME	41
4.8.2.2	DRIVER_NAME	41
4.8.2.3	MAX_NUM_OF_DEVICES	41
4.8.2.4	myGPIOK_USED_INT	41
4.8.3	Documentazione delle funzioni	41

4.8.3.1	MODULE_ALIAS . . . . .	41
4.8.3.2	MODULE_AUTHOR . . . . .	41
4.8.3.3	MODULE_DESCRIPTION . . . . .	41
4.8.3.4	MODULE_DEVICE_TABLE . . . . .	41
4.8.3.5	MODULE_LICENSE . . . . .	41
4.8.3.6	module_platform_driver . . . . .	41
4.8.3.7	MODULE_VERSION . . . . .	42
4.8.3.8	myGPIOK_Destroy . . . . .	42
4.8.3.9	myGPIOK_GetDeviceAddress . . . . .	42
4.8.3.10	myGPIOK_GetPollMask . . . . .	42
4.8.3.11	myGPIOK_GlobalInterruptDisable . . . . .	42
4.8.3.12	myGPIOK_GlobalInterruptEnable . . . . .	42
4.8.3.13	myGPIOK_IncrementTotal . . . . .	43
4.8.3.14	myGPIOK_Init . . . . .	43
4.8.3.15	myGPIOK_irq_handler . . . . .	47
4.8.3.16	myGPIOK_llseek . . . . .	47
4.8.3.17	myGPIOK_open . . . . .	48
4.8.3.18	myGPIOK_PendingPinInterrupt . . . . .	49
4.8.3.19	myGPIOK_PinInterruptAck . . . . .	50
4.8.3.20	myGPIOK_PinInterruptDisable . . . . .	50
4.8.3.21	myGPIOK_PinInterruptEnable . . . . .	50
4.8.3.22	myGPIOK_poll . . . . .	50
4.8.3.23	myGPIOK_probe . . . . .	51
4.8.3.24	myGPIOK_read . . . . .	52
4.8.3.25	myGPIOK_release . . . . .	54
4.8.3.26	myGPIOK_remove . . . . .	55
4.8.3.27	myGPIOK_ResetCanRead . . . . .	55
4.8.3.28	myGPIOK_SetCanRead . . . . .	55
4.8.3.29	myGPIOK_TestCanReadAndSleep . . . . .	56
4.8.3.30	myGPIOK_WakeUp . . . . .	56
4.8.3.31	myGPIOK_write . . . . .	57
4.8.4	Documentazione delle variabili . . . . .	58
4.8.4.1	device_list . . . . .	58
4.8.4.2	myGPIOK_class . . . . .	58
4.8.4.3	myGPIOK_driver . . . . .	59
4.8.4.4	myGPIOK_fops . . . . .	59
4.8.4.5	myGPIOK_match . . . . .	60
4.9	DeviceList . . . . .	61
4.9.1	Descrizione dettagliata . . . . .	61
4.9.2	Documentazione delle funzioni . . . . .	61

4.9.2.1	myGPIOK_list_add . . . . .	61
4.9.2.2	myGPIOK_list_Destroy . . . . .	62
4.9.2.3	myGPIOK_list_device_count . . . . .	62
4.9.2.4	myGPIOK_list_find_by_minor . . . . .	62
4.9.2.5	myGPIOK_list_find_by_op . . . . .	62
4.9.2.6	myGPIOK_list_find_irq_line . . . . .	63
4.9.2.7	myGPIOK_list_Init . . . . .	63
4.10	MyGPIOK_t . . . . .	64
4.10.1	Descrizione dettagliata . . . . .	65
4.10.2	Documentazione delle definizioni . . . . .	65
4.10.2.1	myGPIOK_GIES_OFFSET . . . . .	65
4.10.2.2	myGPIOK_IACK_OFFSET . . . . .	65
4.10.2.3	myGPIOK_IRQ_OFFSET . . . . .	65
4.10.2.4	myGPIOK_PIE_OFFSET . . . . .	65
4.10.3	Documentazione delle funzioni . . . . .	65
4.10.3.1	myGPIOK_Destroy . . . . .	65
4.10.3.2	myGPIOK_GetDeviceAddress . . . . .	65
4.10.3.3	myGPIOK_GetPollMask . . . . .	66
4.10.3.4	myGPIOK_GlobalInterruptDisable . . . . .	66
4.10.3.5	myGPIOK_GlobalInterruptEnable . . . . .	66
4.10.3.6	myGPIOK_IncrementTotal . . . . .	66
4.10.3.7	myGPIOK_Init . . . . .	66
4.10.3.8	myGPIOK_PendingPinInterrupt . . . . .	70
4.10.3.9	myGPIOK_PinInterruptAck . . . . .	71
4.10.3.10	myGPIOK_PinInterruptDisable . . . . .	71
4.10.3.11	myGPIOK_PinInterruptEnable . . . . .	71
4.10.3.12	myGPIOK_ResetCanRead . . . . .	71
4.10.3.13	myGPIOK_SetCanRead . . . . .	72
4.10.3.14	myGPIOK_TestCanReadAndSleep . . . . .	72
4.10.3.15	myGPIOK_WakeUp . . . . .	73
<b>5</b>	<b>Documentazione delle classi</b>	<b>75</b>
5.1	Riferimenti per la struct HD44780_LCD_t . . . . .	75
5.1.1	Descrizione dettagliata . . . . .	76
5.1.2	Documentazione dei campi . . . . .	76
5.1.2.1	Data0 . . . . .	76
5.1.2.2	Data1 . . . . .	76
5.1.2.3	Data2 . . . . .	76
5.1.2.4	Data3 . . . . .	76
5.1.2.5	Data4 . . . . .	76

5.1.2.6	Data5	76
5.1.2.7	Data6	76
5.1.2.8	Data7	77
5.1.2.9	E	77
5.1.2.10	gpio	77
5.1.2.11	iface_mode	77
5.1.2.12	RS	77
5.1.2.13	RW	77
5.2	Riferimenti per la struct myGPIO_t	77
5.2.1	Descrizione dettagliata	77
5.2.2	Documentazione dei campi	78
5.2.2.1	base_address	78
5.2.2.2	gies_offset	78
5.2.2.3	iack_offset	78
5.2.2.4	irq_offset	78
5.2.2.5	mode_offset	78
5.2.2.6	pie_offset	78
5.2.2.7	read_offset	78
5.2.2.8	write_offset	78
5.3	Riferimenti per la struct myGPIOK_list_t	78
5.3.1	Descrizione dettagliata	79
5.3.2	Documentazione dei campi	79
5.3.2.1	device_count	79
5.3.2.2	device_list	79
5.3.2.3	list_size	79
5.4	Riferimenti per la struct myGPIOK_t	79
5.4.1	Descrizione dettagliata	80
5.4.2	Documentazione dei campi	80
5.4.2.1	can_read	80
5.4.2.2	cdev	80
5.4.2.3	class	80
5.4.2.4	dev	80
5.4.2.5	irq_mask	80
5.4.2.6	irqNumber	81
5.4.2.7	Mm	81
5.4.2.8	mreg	81
5.4.2.9	op	81
5.4.2.10	poll_queue	81
5.4.2.11	read_queue	81
5.4.2.12	rsrc	81



5.4.2.13	rsrc_size	81
5.4.2.14	sl_total_irq	81
5.4.2.15	slock_int	81
5.4.2.16	total_irq	82
5.4.2.17	vrtl_addr	82
5.5	Riferimenti per la struct param_t	82
5.5.1	Descrizione dettagliata	82
5.5.2	Documentazione dei campi	82
5.5.2.1	dev_descr	82
5.5.2.2	mode_value	83
5.5.2.3	op_mode	83
5.5.2.4	op_read	83
5.5.2.5	op_write	83
5.5.2.6	write_value	83
5.6	Riferimenti per la struct ZyboButton_t	84
5.6.1	Descrizione dettagliata	84
5.6.2	Documentazione dei campi	84
5.6.2.1	Button0_pin	84
5.6.2.2	Button1_pin	84
5.6.2.3	Button2_pin	84
5.6.2.4	Button3_pin	85
5.6.2.5	gpio	85
5.7	Riferimenti per la struct ZyboLed_t	85
5.7.1	Descrizione dettagliata	85
5.7.2	Documentazione dei campi	86
5.7.2.1	gpio	86
5.7.2.2	Led0_pin	86
5.7.2.3	Led1_pin	86
5.7.2.4	Led2_pin	86
5.7.2.5	Led3_pin	86
5.8	Riferimenti per la struct ZyboSwitch_t	86
5.8.1	Descrizione dettagliata	87
5.8.2	Documentazione dei campi	87
5.8.2.1	gpio	87
5.8.2.2	Switch0_pin	87
5.8.2.3	Switch1_pin	87
5.8.2.4	Switch2_pin	87
5.8.2.5	Switch3_pin	87

6.1	Riferimenti per il file Src/Examples/bsp_example.c	89
6.1.1	Documentazione delle funzioni	89
6.1.1.1	main	89
6.2	Riferimenti per il file Src/Examples/interrupt_bare.c	90
6.2.1	Documentazione delle definizioni	91
6.2.1.1	btn_base_addr	91
6.2.1.2	btn_irq_line	91
6.2.1.3	gic_id	91
6.2.1.4	led_base_addr	91
6.2.1.5	led_irq_line	91
6.2.1.6	swc_base_addr	91
6.2.1.7	swc_irq_line	91
6.2.2	Documentazione delle funzioni	91
6.2.2.1	btn_isr	91
6.2.2.2	gpio_init	91
6.2.2.3	int_config	92
6.2.2.4	main	92
6.2.2.5	swc_isr	92
6.2.3	Documentazione delle variabili	92
6.2.3.1	btn_gpio	92
6.2.3.2	GIC	92
6.2.3.3	led_gpio	92
6.2.3.4	swc_gpio	92
6.3	Riferimenti per il file Src/Examples/mygpiok.c	93
6.3.1	Documentazione delle definizioni	93
6.3.1.1	MODE_OFFSET	93
6.3.1.2	READ_OFFSET	94
6.3.1.3	WRITE_OFFSET	94
6.3.2	Documentazione delle funzioni	94
6.3.2.1	gpio_op	94
6.3.2.2	howto	95
6.3.2.3	main	95
6.3.2.4	parse_args	95
6.4	Riferimenti per il file Src/Examples/noDriver.c	96
6.4.1	Documentazione delle funzioni	96
6.4.1.1	gpio_op	96
6.4.1.2	howto	97
6.4.1.3	main	97
6.4.1.4	parse_args	99
6.5	Riferimenti per il file Src/Examples/readAll.c	100

6.5.1	Documentazione delle funzioni . . . . .	101
6.5.1.1	howto . . . . .	101
6.5.1.2	main . . . . .	101
6.5.1.3	parse_args . . . . .	101
6.6	Riferimenti per il file Src/Examples/uioint.c . . . . .	101
6.6.1	Documentazione delle funzioni . . . . .	102
6.6.1.1	gpio_op . . . . .	102
6.6.1.2	howto . . . . .	103
6.6.1.3	main . . . . .	104
6.6.1.4	parse_args . . . . .	105
6.7	Riferimenti per il file Src/Examples/uioc . . . . .	106
6.7.1	Documentazione delle funzioni . . . . .	107
6.7.1.1	gpio_op . . . . .	107
6.7.1.2	howto . . . . .	107
6.7.1.3	main . . . . .	108
6.7.1.4	parse_args . . . . .	109
6.8	Riferimenti per il file Src/Examples/xil_gpio.c . . . . .	110
6.8.1	Documentazione delle funzioni . . . . .	111
6.8.1.1	XilGpio_Ack_Interrupt . . . . .	111
6.8.1.2	XilGpio_Channel_Interrupt . . . . .	111
6.8.1.3	XilGpio_Global_Interrupt . . . . .	111
6.9	Riferimenti per il file Src/Examples/xil_gpio.h . . . . .	112
6.9.1	Documentazione delle definizioni . . . . .	113
6.9.1.1	CHANNEL1_ACK . . . . .	113
6.9.1.2	CHANNEL1_INTR_DISABLE . . . . .	113
6.9.1.3	CHANNEL1_INTR_ENABLE . . . . .	113
6.9.1.4	CHANNEL2_ACK . . . . .	113
6.9.1.5	CHANNEL2_INTR_DISABLE . . . . .	113
6.9.1.6	CHANNEL2_INTR_ENABLE . . . . .	113
6.9.1.7	GLOBAL_INTR_DISABLE . . . . .	113
6.9.1.8	GLOBAL_INTR_ENABLE . . . . .	113
6.9.1.9	GPIO_DATA_OFFSET . . . . .	113
6.9.1.10	GPIO_MAP_SIZE . . . . .	114
6.9.1.11	GPIO_READ_OFFSET . . . . .	114
6.9.1.12	GPIO_TRI_OFFSET . . . . .	114
6.9.1.13	XGPIO_GIE_GINTR_ENABLE_MASK . . . . .	114
6.9.1.14	XGPIO_GIE_OFFSET . . . . .	114
6.9.1.15	XGPIO_IER_OFFSET . . . . .	114
6.9.1.16	XGPIO_ISR_OFFSET . . . . .	114
6.9.2	Documentazione delle funzioni . . . . .	114

6.9.2.1	XilGpio_Ack_Interrupt . . . . .	114
6.9.2.2	XilGpio_Channel_Interrupt . . . . .	114
6.9.2.3	XilGpio_Global_Interrupt . . . . .	114
6.10	Riferimenti per il file Src/myGPIO/bare-metal/HD44780/hd44780.c . . . . .	114
6.10.1	Descrizione dettagliata . . . . .	117
6.10.2	Documentazione delle definizioni . . . . .	117
6.10.2.1	HD44780_clear . . . . .	117
6.10.2.2	HD44780_clear . . . . .	117
6.10.2.3	HD44780_cursor_blink . . . . .	117
6.10.2.4	HD44780_cursor_l . . . . .	117
6.10.2.5	HD44780_cursor_off . . . . .	117
6.10.2.6	HD44780_cursor_on . . . . .	117
6.10.2.7	HD44780_cursor_r . . . . .	117
6.10.2.8	HD44780_dec_no_shift . . . . .	117
6.10.2.9	HD44780_dec_shift . . . . .	117
6.10.2.10	HD44780_display_off . . . . .	117
6.10.2.11	HD44780_home . . . . .	117
6.10.2.12	HD44780_inc_no_shift . . . . .	117
6.10.2.13	HD44780_inc_shift . . . . .	117
6.10.2.14	HD44780_row1 . . . . .	117
6.10.2.15	HD44780_row2 . . . . .	117
6.10.2.16	lcd_command . . . . .	117
6.10.2.17	lcd_data . . . . .	117
6.10.2.18	lcd_enable . . . . .	118
6.10.2.19	lcd_read . . . . .	118
6.10.2.20	lcd_write . . . . .	118
6.10.2.21	timer_wait_ms . . . . .	118
6.10.2.22	timer_wait_us . . . . .	118
6.10.3	Documentazione delle funzioni . . . . .	118
6.10.3.1	HD44780_ConfigurePin . . . . .	118
6.10.3.2	HD44780_SetByte . . . . .	118
6.10.3.3	HD44780_ValidatePair . . . . .	118
6.10.3.4	HD44780_WriteCommand . . . . .	118
6.10.3.5	HD44780_WriteData . . . . .	118
6.11	Riferimenti per il file Src/myGPIO/bare-metal/HD44780/hd44780.h . . . . .	118
6.11.1	Descrizione dettagliata . . . . .	120
6.12	Riferimenti per il file Src/myGPIO/bare-metal/myGPIO.c . . . . .	121
6.12.1	Descrizione dettagliata . . . . .	122
6.13	Riferimenti per il file Src/myGPIO/bare-metal/myGPIO.h . . . . .	122
6.13.1	Descrizione dettagliata . . . . .	124

6.14	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/Zybo.h . . . . .	125
6.14.1	Descrizione dettagliata . . . . .	125
6.15	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboButton.c . . . . .	126
6.15.1	Descrizione dettagliata . . . . .	127
6.15.2	Documentazione delle definizioni . . . . .	127
6.15.2.1	timer_wait_ms . . . . .	127
6.15.3	Documentazione delle funzioni . . . . .	127
6.15.3.1	validatePair . . . . .	127
6.16	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboButton.h . . . . .	127
6.16.1	Descrizione dettagliata . . . . .	129
6.17	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboLed.c . . . . .	129
6.17.1	Descrizione dettagliata . . . . .	130
6.17.2	Documentazione delle funzioni . . . . .	131
6.17.2.1	validatePair . . . . .	131
6.18	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboLed.h . . . . .	131
6.18.1	Descrizione dettagliata . . . . .	132
6.19	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboSwitch.c . . . . .	132
6.19.1	Descrizione dettagliata . . . . .	133
6.19.2	Documentazione delle funzioni . . . . .	134
6.19.2.1	validatePair . . . . .	134
6.20	Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboSwitch.h . . . . .	134
6.20.1	Descrizione dettagliata . . . . .	135
6.21	Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK_list.c . . . . .	135
6.21.1	Descrizione dettagliata . . . . .	136
6.22	Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK_list.h . . . . .	136
6.22.1	Descrizione dettagliata . . . . .	137
6.23	Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK_main.c . . . . .	138
6.24	Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK_t.c . . . . .	140
6.24.1	Descrizione dettagliata . . . . .	141
6.25	Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK_t.h . . . . .	141
6.25.1	Descrizione dettagliata . . . . .	143
<b>7</b>	<b>Documentazione degli esempi</b>	<b>145</b>
7.1	bsp_example.c . . . . .	145
7.2	interrupt_bare.c . . . . .	146
7.3	mygpiok.c . . . . .	151
7.4	noDriver.c . . . . .	153
7.5	readAll.c . . . . .	155
7.6	uio-int.c . . . . .	157
7.7	uio.c . . . . .	159



# Capitolo 1

## Indice dei moduli

### 1.1 Moduli

Questo è l'elenco di tutti i moduli:

MyGPIO . . . . .	36
Bare-metal . . . . .	7
HD44780 . . . . .	16
Zybo . . . . .	24
Button . . . . .	25
Led . . . . .	29
Switch . . . . .	33
Linux-Driver . . . . .	37
DeviceList . . . . .	61
MyGPIOK_t . . . . .	64





## Capitolo 2

# Indice delle strutture dati

### 2.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

#### [HD44780\\_LCD\\_t](#)

Struttura opaca che astrae un device Display LCD con cntroller Hitachi HD44780, o compatibile. Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi. Il modulo permette di utilizzare sia l'interfacciamento ad otto bit che quello a quattro bit, inizializzando il device opportunamente, attraverso l'uso delle funzioni `H44780_Init8` e `H44780_Init4`. Il modulo fornisce anche semplici funzioni per la stampa di un carattere o di una stringa null-terminated di caratteri. Si veda la documentazione delle funzioni [HD44780\\_Printc\(\)](#) e [HD44780\\_Print\(\)](#). Inoltre sono presenti diverse funzioni di utilità generica, come quelle per la pulizia del display, per lo spostamento del cursore di un posto in avanti o indietro, alla riga in basso o in alto . . . . .

75

#### [myGPIO\\_t](#)

Struttura che astrae un device myGPIO . . . . .

77

#### [myGPIOK\\_list\\_t](#)

Struttura dati per la gestione degli oggetti [myGPIOK\\_t](#) gestiti dal driver . . . . .

78

#### [myGPIOK\\_t](#)

Struttura per l'astrazione di un device myGPIO in kernel-mode . . . . .

79

#### [param\\_t](#)

La struttura raccoglie tutti i parametri di esecuzione del programma . . . . .

82

#### [ZyboButton\\_t](#)

Struttura opaca che astrae l'insieme dei button presenti sulla board Digilent Zybo; . . . . .

84

#### [ZyboLed\\_t](#)

Struttura opaca che astrae l'insieme dei Led presenti sulla board Digilent Zybo; . . . . .

85

#### [ZyboSwitch\\_t](#)

Struttura opaca che astrae l'insieme degli switch presenti sulla board Digilent Zybo; . . . . .

86



## Capitolo 3

# Indice dei file

### 3.1 Elenco dei file

Questo è un elenco di tutti i file con una loro breve descrizione:

Src/Examples/ <a href="#">bsp_example.c</a>	89
Src/Examples/ <a href="#">interrupt_bare.c</a>	90
Src/Examples/ <a href="#">mygpiok.c</a>	93
Src/Examples/ <a href="#">noDriver.c</a>	96
Src/Examples/ <a href="#">readAll.c</a>	100
Src/Examples/ <a href="#">uio-int.c</a>	101
Src/Examples/ <a href="#">uio.c</a>	106
Src/Examples/ <a href="#">xil_gpio.c</a>	110
Src/Examples/ <a href="#">xil_gpio.h</a>	112
Src/myGPIO/bare-metal/ <a href="#">myGPIO.c</a>	121
Src/myGPIO/bare-metal/ <a href="#">myGPIO.h</a>	122
Src/myGPIO/bare-metal/HD44780/ <a href="#">hd44780.c</a>	114
Src/myGPIO/bare-metal/HD44780/ <a href="#">hd44780.h</a>	118
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">Zybo.h</a>	125
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboButton.c</a>	126
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboButton.h</a>	127
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboLed.c</a>	129
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboLed.h</a>	131
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboSwitch.c</a>	132
Src/myGPIO/bare-metal/ZyboBSP/ <a href="#">ZyboSwitch.h</a>	134
Src/myGPIO/linux-driver/ <a href="#">myGPIOK_list.c</a>	135
Src/myGPIO/linux-driver/ <a href="#">myGPIOK_list.h</a>	136
Src/myGPIO/linux-driver/ <a href="#">myGPIOK_main.c</a>	138
Src/myGPIO/linux-driver/ <a href="#">myGPIOK_t.c</a>	140
Src/myGPIO/linux-driver/ <a href="#">myGPIOK_t.h</a>	141



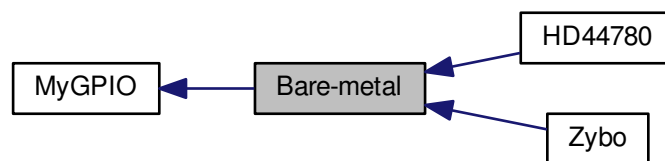
## Capitolo 4

# Documentazione dei moduli

### 4.1 Bare-metal

device-driver OO-like bare-metal per device myGPIO

Diagramma di collaborazione per Bare-metal:



### Moduli

- [HD44780](#)  
*driver per display Hitachi hd44780 basato su driver myGPIO bare-metal*
- [Zybo](#)

### Strutture dati

- struct [myGPIO\\_t](#)  
*Struttura che astrae un device myGPIO.*

### Definizioni

- #define [myGPIO\\_MODE\\_OFFSET](#) 0x00U  
*Offset, rispetto all'indirizzo base, del registro "mode" per il device myGPIO.*
- #define [myGPIO\\_WRITE\\_OFFSET](#) 0x04U  
*Offset, rispetto all'indirizzo base, del registro "write" per il device myGPIO.*
- #define [myGPIO\\_READ\\_OFFSET](#) 0x08U

- Offset, rispetto all'indirizzo base, del registro "read" per il device myGPIO.*
- #define `myGPIO_GIES_OFFSET` 0x0CU
- Offset, rispetto all'indirizzo base, del registro "gies" per il device myGPIO.*
- #define `myGPIO_PIE_OFFSET` 0x10U
- Offset, rispetto all'indirizzo base, del registro "pie" per il device myGPIO.*
- #define `myGPIO_IRQ_OFFSET` 0x14U
- Offset, rispetto all'indirizzo base, del registro "irq" per il device myGPIO.*
- #define `myGPIO_IACK_OFFSET` 0x18U
- Offset, rispetto all'indirizzo base, del registro "iack" per il device myGPIO.*
- #define `myGPIO_pin(i)` ((uint32\_t)(1 << (i)))
- Metodo alternativo per la specifica di uno dei pin di un device myGPIO.*

## Tipi enumerati (enum)

- enum `myGPIO_mask` {  
`myGPIO_pin0` = 0x1U, `myGPIO_pin1` = 0x2U, `myGPIO_pin2` = 0x4U, `myGPIO_pin3` = 0x8U,  
`myGPIO_pin4` = 0x10U, `myGPIO_pin5` = 0x20U, `myGPIO_pin6` = 0x40U, `myGPIO_pin7` = 0x80U,  
`myGPIO_pin8` = 0x100U, `myGPIO_pin9` = 0x200U, `myGPIO_pin10` = 0x400U, `myGPIO_pin11` = 0x800U,  
`myGPIO_pin12` = 0x1000U, `myGPIO_pin13` = 0x2000U, `myGPIO_pin14` = 0x4000U, `myGPIO_pin15` =  
0x8000U,  
`myGPIO_pin16` = 0x10000U, `myGPIO_pin17` = 0x20000U, `myGPIO_pin18` = 0x40000U, `myGPIO_pin19` =  
0x80000U,  
`myGPIO_pin20` = 0x100000U, `myGPIO_pin21` = 0x200000U, `myGPIO_pin22` = 0x400000U, `myGPIO_pin23`  
= 0x800000U,  
`myGPIO_pin24` = 0x1000000U, `myGPIO_pin25` = 0x2000000U, `myGPIO_pin26` = 0x4000000U, `myGPIO_↵`  
`pin27` = 0x8000000U,  
`myGPIO_pin28` = 0x10000000U, `myGPIO_pin29` = 0x20000000U, `myGPIO_pin30` = 0x40000000U, `myGP↵`  
`IO_pin31` = 0x80000000U,  
`myGPIO_byte0` = 0x000000ffU, `myGPIO_byte1` = 0x0000ff00U, `myGPIO_byte2` = 0x00ff0000U, `myGPIO_↵`  
`byte3` = 0xff000000U }  
*Maschere di selezione dei pin di un device myGPIO.*
- enum `myGPIO_mode` { `myGPIO_read`, `myGPIO_write` }  
*myGPIO\_mode, modalità di funzionamento (lettura/scrittura) di un device myGPIO*
- enum `myGPIO_value` { `myGPIO_reset`, `myGPIO_set` }  
*myGPIO\_value, valore di un myGPIO*

## Funzioni

- void `myGPIO_Init` (`myGPIO_t` \*gpio, uint32\_t base\_address)  
*Inizializza un device myGPIO.*
- void `myGPIO_SetMode` (`myGPIO_t` \*gpio, `myGPIO_mask` mask, `myGPIO_mode` mode)  
*Permette di settare la modalità lettura/scrittura dei pin di un device myGPIO;.*
- void `myGPIO_SetValue` (`myGPIO_t` \*gpio, `myGPIO_mask` mask, `myGPIO_value` value)  
*Permette di settare il valore dei pin di un device myGPIO, se configurati come output.*
- void `myGPIO_Toggle` (`myGPIO_t` \*gpio, `myGPIO_mask` mask)  
*Permette di invertire il valore dei pin di un device myGPIO, se configurati come output.*
- `myGPIO_value` `myGPIO_GetValue` (`myGPIO_t` \*gpio, `myGPIO_mask` mask)  
*Permette di leggere il valore dei pin di un device myGPIO;.*
- `myGPIO_mask` `myGPIO_GetRead` (`myGPIO_t` \*gpio)  
*Restituisce la maschera dei pin settati di un device myGPIO.*
- void `myGPIO_GlobalInterruptEnable` (`myGPIO_t` \*gpio)  
*Abilita gli interrupt globali;.*

- void `myGPIO_GlobalInterruptDisable` (`myGPIO_t *gpio`)  
*Disabilita gli interrupt globali;.*
- `myGPIO_value myGPIO_IsGlobalInterruptEnabled` (`myGPIO_t *gpio`)  
*Consente di verificare se gli interrupt globali siano abilitati.*
- `myGPIO_value myGPIO_PendingInterrupt` (`myGPIO_t *gpio`)  
*Consente di verificare se esistano interrupt non ancora serviti.*
- void `myGPIO_PinInterruptEnable` (`myGPIO_t *gpio`, `myGPIO_mask mask`)  
*Abilita gli interrupt per i singoli pin del device.*
- void `myGPIO_PinInterruptDisable` (`myGPIO_t *gpio`, `myGPIO_mask mask`)  
*Disabilita gli interrupt per i singoli pin del device.*
- `myGPIO_mask myGPIO_EnabledPinInterrupt` (`myGPIO_t *gpio`)  
*Consente di ottenere una maschera che indichi quali pin abbiano interrupt abilitati.*
- `myGPIO_mask myGPIO_PendingPinInterrupt` (`myGPIO_t *gpio`)  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- void `myGPIO_PinInterruptAck` (`myGPIO_t *gpio`, `myGPIO_mask mask`)  
*Invia al device notifica di servizio di un interrupt;.*

#### 4.1.1 Descrizione dettagliata

device-driver OO-like bare-metal per device myGPIO

#### 4.1.2 Documentazione delle definizioni

##### 4.1.2.1 #define myGPIO\_GIES\_OFFSET 0x0CU

Offset, rispetto all'indirizzo base, del registro "gies" per il device myGPIO.

##### 4.1.2.2 #define myGPIO\_IACK\_OFFSET 0x18U

Offset, rispetto all'indirizzo base, del registro "iack" per il device myGPIO.

##### 4.1.2.3 #define myGPIO\_IRQ\_OFFSET 0x14U

Offset, rispetto all'indirizzo base, del registro "irq" per il device myGPIO.

##### 4.1.2.4 #define myGPIO\_MODE\_OFFSET 0x00U

Offset, rispetto all'indirizzo base, del registro "mode" per il device myGPIO.

##### 4.1.2.5 #define myGPIO\_PIE\_OFFSET 0x10U

Offset, rispetto all'indirizzo base, del registro "pie" per il device myGPIO.

##### 4.1.2.6 #define myGPIO\_pin( i ) ((uint32\_t)(1<<(i)))

Metodo alternativo per la specifica di uno dei pin di un device myGPIO.

## Parametri

<i>in</i>	<i>i</i>	indice del bit da selezionare, da 0 (bit meno significativo) a 31 (bit più significativo)
-----------	----------	---

## Restituisce

maschera di selezione del pin i-esimo

## Esempi:

[interrupt\\_bare.c](#).

## 4.1.2.7 #define myGPIO\_READ\_OFFSET 0x08U

Offset, rispetto all'indirizzo base, del registro "read" per il device myGPIO.

## 4.1.2.8 #define myGPIO\_WRITE\_OFFSET 0x04U

Offset, rispetto all'indirizzo base, del registro "write" per il device myGPIO.

## 4.1.3 Documentazione dei tipi enumerati

## 4.1.3.1 enum myGPIO\_mask

Maschere di selezione dei pin di un device myGPIO.

## Valori del tipo enumerato

***myGPIO\_pin0*** myGPIO pin0 maschera di selezione del pin 0 di un device myGPIO  
***myGPIO\_pin1*** myGPIO pin1 maschera di selezione del pin 1 di un device myGPIO  
***myGPIO\_pin2*** myGPIO pin2 maschera di selezione del pin 2 di un device myGPIO  
***myGPIO\_pin3*** myGPIO pin3 maschera di selezione del pin 3 di un device myGPIO  
***myGPIO\_pin4*** myGPIO pin4 maschera di selezione del pin 4 di un device myGPIO  
***myGPIO\_pin5*** myGPIO pin5 maschera di selezione del pin 5 di un device myGPIO  
***myGPIO\_pin6*** myGPIO pin6 maschera di selezione del pin 6 di un device myGPIO  
***myGPIO\_pin7*** myGPIO pin7 maschera di selezione del pin 7 di un device myGPIO  
***myGPIO\_pin8*** myGPIO pin8 maschera di selezione del pin 8 di un device myGPIO  
***myGPIO\_pin9*** myGPIO pin9 maschera di selezione del pin 9 di un device myGPIO  
***myGPIO\_pin10*** myGPIO pin10 maschera di selezione del pin 10 di un device myGPIO  
***myGPIO\_pin11*** myGPIO pin11 maschera di selezione del pin 11 di un device myGPIO  
***myGPIO\_pin12*** myGPIO pin12 maschera di selezione del pin 12 di un device myGPIO  
***myGPIO\_pin13*** myGPIO pin13 maschera di selezione del pin 13 di un device myGPIO  
***myGPIO\_pin14*** myGPIO pin14 maschera di selezione del pin 14 di un device myGPIO  
***myGPIO\_pin15*** myGPIO pin15 maschera di selezione del pin 15 di un device myGPIO  
***myGPIO\_pin16*** myGPIO pin16 maschera di selezione del pin 16 di un device myGPIO  
***myGPIO\_pin17*** myGPIO pin17 maschera di selezione del pin 17 di un device myGPIO  
***myGPIO\_pin18*** myGPIO pin18 maschera di selezione del pin 18 di un device myGPIO  
***myGPIO\_pin19*** myGPIO pin19 maschera di selezione del pin 19 di un device myGPIO  
***myGPIO\_pin20*** myGPIO pin20 maschera di selezione del pin 20 di un device myGPIO



**myGPIO\_pin21** myGPIO pin21 maschera di selezione del pin 21 di un device myGPIO  
**myGPIO\_pin22** myGPIO pin22 maschera di selezione del pin 22 di un device myGPIO  
**myGPIO\_pin23** myGPIO pin23 maschera di selezione del pin 23 di un device myGPIO  
**myGPIO\_pin24** myGPIO pin24 maschera di selezione del pin 24 di un device myGPIO  
**myGPIO\_pin25** myGPIO pin25 maschera di selezione del pin 25 di un device myGPIO  
**myGPIO\_pin26** myGPIO pin26 maschera di selezione del pin 26 di un device myGPIO  
**myGPIO\_pin27** myGPIO pin27 maschera di selezione del pin 27 di un device myGPIO  
**myGPIO\_pin28** myGPIO pin28 maschera di selezione del pin 28 di un device myGPIO  
**myGPIO\_pin29** myGPIO pin29 maschera di selezione del pin 29 di un device myGPIO  
**myGPIO\_pin30** myGPIO pin30 maschera di selezione del pin 30 di un device myGPIO  
**myGPIO\_pin31** myGPIO pin31 maschera di selezione del pin 31 di un device myGPIO  
**myGPIO\_byte0** myGPIO byte0 maschera di selezione del pin 0-7 di un device myGPIO  
**myGPIO\_byte1** myGPIO byte1 maschera di selezione del pin 8-15 di un device myGPIO  
**myGPIO\_byte2** myGPIO byte2 maschera di selezione del pin 16-23 di un device myGPIO  
**myGPIO\_byte3** myGPIO byte3 maschera di selezione del pin 24-31 di un device myGPIO

#### 4.1.3.2 enum myGPIO\_mode

myGPIO\_mode, modalità di funzionamento (lettura/scrittura) di un device myGPIO

Valori del tipo enumerato

**myGPIO\_read** myGPIO\_read modalità lettura  
**myGPIO\_write** myGPIO\_write modalità scrittura

#### 4.1.3.3 enum myGPIO\_value

myGPIO\_value, valore di un myGPIO

Valori del tipo enumerato

**myGPIO\_reset** myGPIO\_reset, corrisponde al valore logico '0'  
**myGPIO\_set** myGPIO\_set, corrisponde al valore logico '1'

### 4.1.4 Documentazione delle funzioni

#### 4.1.4.1 myGPIO\_mask myGPIO\_EnabledPinInterrupt ( myGPIO\_t \* gpio )

Consente di ottenere una maschera che indichi quali pin abbiano interrupt abilitati.

Parametri

in	gpio	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
----	------	---

Restituisce

maschera che riporta i pin per i quali gli interrupt sono stati abilitati;

#### 4.1.4.2 myGPIO\_mask myGPIO\_GetRead ( myGPIO\_t \* gpio )

Restituisce la maschera dei pin settati di un device myGPIO.

**Parametri**

<i>in</i>	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
-----------	-------------	---

**Restituisce**

maschera dei pin settati di un device myGPIO

```

1 myGPIO_mask mask = myGPIO_read(&gpio);
2 if (mask & myGPIO_pin3) {
3     ...
4 }
5 else {
6     ...
7 }
```

**4.1.4.3 myGPIO\_value myGPIO\_GetValue ( myGPIO\_t \* gpio, myGPIO\_mask mask )**

Permette di leggere il valore dei pin di un device myGPIO;.

```

1 // legge il valore del pin 0 di un device myGPIO.
2 myGPIO_value value = gpio_getValue(gpio, myGPIO_pin0);
3
4 // legge il valore del pin 0, 3 e 5 di un device myGPIO.
5 // Verrà restituita la OR tra i valori dei pin
6 myGPIO_value value = gpio_getValue(gpio, myGPIO_pin0 | myGPIO_pin3 | myGPIO_pin5);
```

**Parametri**

<i>in</i>	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
<i>in</i>	<i>mask</i>	maschera dei pin su cui agire;

**Restituisce**

restituisce la OR dei pin letti

**Valori di ritorno**

<i>myGPIO_set</i>	se uno dei pin letti è myGPIO_set,
<i>myGPIO_reset</i>	se TUTTI i pin sono myGPIO_reset

**Avvertimento**

Usa la macro assert per verificare che gpio non sia un puntatore nullo

**4.1.4.4 void myGPIO\_GlobalInterruptDisable ( myGPIO\_t \* gpio )**

Disabilita gli interrupt globali;.

**Parametri**

<i>in</i>	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
-----------	-------------	---

**4.1.4.5 void myGPIO\_GlobalInterruptEnable ( myGPIO\_t \* gpio )**

Abilita gli interrupt globali;.

## Parametri

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
----	-------------	---

4.1.4.6 void myGPIO\_Init ( myGPIO\_t \* *gpio*, uint32\_t *base\_address* )

Inizializza un device myGPIO.

Inizializza una struttura di tipo [myGPIO\\_t](#), che astrae u device myGPIO, controllando che l'inizializzazione vada a buon fine, effettuando diversi test sui parametri di inizializzazione e restituendo un codice di errore.

## Parametri

in, out	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>base_address</i>	indirizzo di memoria a cui è mappato il device myGPIO;  1 myGPIO_t <i>gpio</i> ; 2 myGPIO_init(& <i>gpio</i> , BASE_ADDRESS);

## Avvertimento

Usa la macro assert per verificare che *gpio* e *base\_address* non siano nulli.

4.1.4.7 myGPIO\_value myGPIO\_IsGlobalInterruptEnabled ( myGPIO\_t \* *gpio* )

Consente di verificare se gli interrupt globali siano abilitati.

## Parametri

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
----	-------------	---

## Valori di ritorno

<i>myGPIO_set</i>	se il bit 0 del registro GIES è settato, ad indicare che gli interrupt sono abilitati
<i>myGPIO_reset</i>	se il bit 0 del registro GIES è resettato, ad indicare che gli interrupt non sono abilitati

4.1.4.8 myGPIO\_value myGPIO\_PendingInterrupt ( myGPIO\_t \* *gpio* )

Consente di verificare se esistano interrupt non ancora serviti.

## Parametri

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
----	-------------	---

## Valori di ritorno

<i>myGPIO_set</i>	se il bit 1 del registro GIES è settato, ad indicare che esistono interrupt pending
<i>myGPIO_reset</i>	se il bit 1 del registro GIES è resettato, ad indicare che non esistono interrupt pending

4.1.4.9 myGPIO\_mask myGPIO\_PendingPinInterrupt ( myGPIO\_t \* *gpio* )

Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
----	-------------	---

**Restituisce**

maschera che riporta i pin per i quali gli interrupt non sono stati ancora serviti;

**4.1.4.10 void myGPIO\_PinInterruptAck ( myGPIO\_t \* *gpio*, myGPIO\_mask *mask* )**

Invia al device notifica di servizio di un interrupt;.

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera di selezione dei bit;

**4.1.4.11 void myGPIO\_PinInterruptDisable ( myGPIO\_t \* *gpio*, myGPIO\_mask *mask* )**

Disabilita gli interrupt per i singoli pin del device.

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera di selezione degli interrupt da disabilitare, quelli non selezionati non vengono disabilitati;

**4.1.4.12 void myGPIO\_PinInterruptEnable ( myGPIO\_t \* *gpio*, myGPIO\_mask *mask* )**

Abilita gli interrupt per i singoli pin del device.

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera di selezione degli interrupt da abilitare;quelli non selezionati non vengono abilitati;

**4.1.4.13 void myGPIO\_SetMode ( myGPIO\_t \* *gpio*, myGPIO\_mask *mask*, myGPIO\_mode *mode* )**

Permette di settare la modalità lettura/scrittura dei pin di un device myGPIO;.

```
1 // setta i pin 0 ed 1 di un device myGPIO come pin di uscita, gli altri restano invariati
2 myGPIO_setMode(gpio, myGPIO_pin0 | myGPIO_pin1, myGPIO_write);
3
4 // setta i pin 19, 20 e 21 di un device myGPIO come pin di ingresso, gli altri restano invariati
5 myGPIO_setMode(gpio, myGPIO_pin19 | myGPIO_pin20 | myGPIO_pin21, myGPIO_read);
```

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera dei pin su cui agire;
in	<i>mode</i>	modalità di funzionamento dei pin;

**Avvertimento**

Usa la macro assert per verificare che gpio non sia un puntatore nullo

**4.1.4.14 void myGPIO\_SetValue ( myGPIO\_t \* gpio, myGPIO\_mask mask, myGPIO\_value value )**

Permette di settare il valore dei pin di un device myGPIO, se configurati come output.

```
1 // setta i pin 0 ed 1 di un device myGPIO a livello logico '1', gli altri restano invariati
2 myGPIO_setValue(gpio, myGPIO_pin0 | myGPIO_pin1, myGPIO_set);
3
4 // setta i pin 19, 20 e 21 di un device myGPIO a livello logico '0', gli altri restano invariati
5 myGPIO_setValue(gpio, myGPIO_pin19 | myGPIO_pin20 | myGPIO_pin21, myGPIO_reset);
```

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera dei pin su cui agire;
in	<i>value</i>	valore dei pin

**Avvertimento**

Usa la macro assert per verificare che gpio non sia un puntatore nullo

**4.1.4.15 void myGPIO\_Toggle ( myGPIO\_t \* gpio, myGPIO\_mask mask )**

Permette di invertire il valore dei pin di un device myGPIO, se configurati come output.

```
1 // inverte i pin 0 ed 1 di un device myGPIO a livello logico '1', gli altri restano invariati
2 myGPIO_toggle(gpio, myGPIO_pin0 | myGPIO_pin1);
```

**Parametri**

in	<i>gpio</i>	puntatore a <a href="#">myGPIO_t</a> , che astrae un device myGPIO;
in	<i>mask</i>	maschera dei pin su cui agire;

**Avvertimento**

Usa la macro assert per verificare che gpio non sia un puntatore nullo

## 4.2 HD44780

driver per display Hitachi hd44780 basato su driver myGPIO bare-metal

Diagramma di collaborazione per HD44780:



### Strutture dati

- struct [HD44780\\_LCD\\_t](#)

Struttura opaca che astrae un device Display LCD con controller Hitachi HD44780, o compatibile. Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi. Il modulo permette di utilizzare sia l'interfacciamento ad otto bit che quello a quattro bit, inizializzando il device opportunamente, attraverso l'uso delle funzioni [HD44780\\_Init8](#) e [HD44780\\_Init4](#). Il modulo fornisce anche semplici funzioni per la stampa di un carattere o di una stringa null-terminated di caratteri. Si veda la documentazione delle funzioni [HD44780\\_Printc\(\)](#) e [HD44780\\_Print\(\)](#). Inoltre sono presenti diverse funzioni di utilità generica, come quelle per la pulizia del display, per lo spostamento del cursore di un posto in avanti o indietro, alla riga in basso o in alto.

### Tipi enumerati (enum)

- enum [HD44780\\_InterfaceMode\\_t](#) { [HD44780\\_INTERFACE\\_4bit](#), [HD44780\\_INTERFACE\\_8bit](#) }

Modalità di interfacciamento. Il modulo supporta sia interfacciamento a 4 bit che ad 8 bit.

- enum [HD44780\\_Direction\\_t](#) { [HD44780\\_CursorLeft](#), [HD44780\\_CursorRight](#) }

Direzioni di spostamento del cursore, usata dalla funzione [HD44780\\_MoveCursor\(\)](#)

### Funzioni

- void [HD44780\\_Init8](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4, [myGPIO\\_mask](#) Data3, [myGPIO\\_mask](#) Data2, [myGPIO\\_mask](#) Data1, [myGPIO\\_mask](#) Data0)

Inizializza un display lcd HD44780 con interfacciamento ad 8 bit.

- void [HD44780\\_Init4](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4)

Inizializza un oggetto display lcd HD44780 affinché si utilizzi l'interfaccia a 4 bit.

- void [HD44780\\_Printc](#) ([HD44780\\_LCD\\_t](#) \*lcd, char c)

Stampa un carattere.

- void [HD44780\\_Print](#) ([HD44780\\_LCD\\_t](#) \*lcd, const char \*s)

Stampa una stringa null-terminated di caratteri.

- void [HD44780\\_printBinary8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)

Stampa un byte in binario. (bit più significativo a sinistra)

- void [HD44780\\_printBinary32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)

Stampa una word di 32 bit in binario. (bit più significativo a sinistra)

- void [HD44780\\_printBinary64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)

- Stampa un blocco di 64 bit in binario. (bit più significativo a sinistra)*
  - void [HD44780\\_printHex8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)
- Stampa un byte in esadecimale. (bit più significativo a sinistra)*
  - void [HD44780\\_printHex32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)
- Stampa una word di 32 bit in esadecimale. (bit più significativo a sinistra)*
  - void [HD44780\\_printHex64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)
- Stampa un blocco di 64 bit in esadecimale. (bit più significativo a sinistra)*
  - void [HD44780\\_Clear](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Pulisce il display e sposta il cursore all'inizio della prima riga.*
  - void [HD44780\\_Home](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della prima riga.*
  - void [HD44780\\_MoveToRow1](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della prima riga.*
  - void [HD44780\\_MoveToRow2](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della seconda riga.*
  - void [HD44780\\_MoveCursor](#) ([HD44780\\_LCD\\_t](#) \*lcd, [HD44780\\_Direction\\_t](#) dir)
- Sposta il cursore di una posizione a destra o sinistra.*
  - void [HD44780\\_DisplayOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Disattiva il display.*
  - void [HD44780\\_CursorOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Disattiva la visualizzazione del cursore.*
  - void [HD44780\\_CursorOn](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Attiva la visualizzazione del cursore.*
  - void [HD44780\\_CursorBlink](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Attiva il cursore lampeggiante.*

### 4.2.1 Descrizione dettagliata

driver per display Hitachi hd44780 basato su driver myGPIO bare-metal

Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi.

La struttura [HD44780\\_LCD\\_t](#) specifica quali siano i pin del microcontrollore che pilotano un determinato segnale del device. L'assegnazione segnale-coppia, quindi l'inizializzazione della struttura [HD44780\\_LCD\\_t](#) relativa ad un device lcd, DEVE essere effettuata tassativamente utilizzando le funzioni

- [HD44780\\_Init4\(\)](#)
- [HD44780\\_Init8\(\)](#)

le quali provvedono anche ad effettuare un test di connessione volto ad individuare eventuali segnali erroneamente associati.

Oltre alle funzioni di inizializzazione, il modulo fornisce anche funzioni basilari per la stampa su display lcd di

- caratteri, con la funzione [HD44780\\_Printc\(\)](#)
- stringhe null-terminated di caratteri, con la funzione [HD44780\\_Print\(\)](#)

Sono disponibili, inoltre, anche funzioni specifiche per inviare comandi al device:

- [HD44780\\_Clear\(\)](#)
- [HD44780\\_Home\(\)](#)
- [HD44780\\_MoveToRow1\(\)](#)
- [HD44780\\_MoveToRow2\(\)](#)
- [HD44780\\_MoveCursor\(\)](#)
- [HD44780\\_DisplayOff\(\)](#)
- [HD44780\\_CursorOff\(\)](#)
- [HD44780\\_CursorOn\(\)](#)
- [HD44780\\_CursorBlink\(\)](#)

## 4.2.2 Documentazione dei tipi enumerati

### 4.2.2.1 enum HD44780\_Direction\_t

Direzioni di spostamento del cursore, usata dalla funzione [HD44780\\_MoveCursor\(\)](#)

Valori del tipo enumerato

**HD44780\_CursorLeft** left sposta il cursore a sinistra

**HD44780\_CursorRight** right sposta il cursore a destra

### 4.2.2.2 enum HD44780\_InterfaceMode\_t

Modalità di interfacciamento. Il modulo supporta sia interfacciamento a 4 bit che ad 8 bit.

Valori del tipo enumerato

**HD44780\_INTERFACE\_4bit** Interfacciamento a quattro bit

**HD44780\_INTERFACE\_8bit** Interfacciamento ad otto bit

## 4.2.3 Documentazione delle funzioni

### 4.2.3.1 void HD44780\_Clear ( HD44780\_LCD\_t \* lcd )

Pulisce il display e sposta il cursore all'inizio della prima riga.

Parametri

in	lcd	display da pilotare;
----	-----	----------------------

### 4.2.3.2 void HD44780\_CursorBlink ( HD44780\_LCD\_t \* lcd )

Attiva il cursore lampeggiante.



## Parametri

<i>in</i>	<i>lcd</i>	display da pilotare;
-----------	------------	----------------------

4.2.3.3 void HD44780\_CursorOff ( HD44780\_LCD\_t \* *lcd* )

Disattiva la visualizzazione del cursore.

## Parametri

<i>in</i>	<i>lcd</i>	display da pilotare;
-----------	------------	----------------------

4.2.3.4 void HD44780\_CursorOn ( HD44780\_LCD\_t \* *lcd* )

Attiva la visualizzazione del cursore.

## Parametri

<i>in</i>	<i>lcd</i>	display da pilotare;
-----------	------------	----------------------

4.2.3.5 void HD44780\_DisplayOff ( HD44780\_LCD\_t \* *lcd* )

Disattiva il display.

## Parametri

<i>in</i>	<i>lcd</i>	display da pilotare;
-----------	------------	----------------------

4.2.3.6 void HD44780\_Home ( HD44780\_LCD\_t \* *lcd* )

Sposta il cursore all'inizio della prima riga.

## Parametri

<i>in</i>	<i>lcd</i>	display da pilotare;
-----------	------------	----------------------

4.2.3.7 void HD44780\_Init4 ( HD44780\_LCD\_t \* *lcd*, myGPIO\_t \* *gpio*, myGPIO\_mask *RS*, myGPIO\_mask *RW*, myGPIO\_mask *E*, myGPIO\_mask *Data7*, myGPIO\_mask *Data6*, myGPIO\_mask *Data5*, myGPIO\_mask *Data4* )

Inizializza un oggetto display lcd HD44780 affinché si utilizzi l'interfaccia a 4 bit.

Inizializza un oggetto [HD44780\\_LCD\\_t](#) verificando la validità delle coppie porta-pin per l' interfacciamento, configurando i pin myGPIO e iniziando il device.

## Avvertimento

Se i pin associati ai segnali di pilotaggio del device non sono correttamente configurati come pin di output, il dispositivo non funzionerà correttamente.

Non modificare i campi della struttura [HD44780\\_LCD\\_t](#) dopo che essa sia stata inizializzata.

La struttura [myGPIO\\_t](#), a cui fa riferimento il parametro *gpio*, va inizializzata a parte.

**Parametri**

in, out	<i>lcd</i>	puntatore a struttura di tipo <a href="#">HD44780_LCD_t</a> che descrive un display HD44780 da inizializzare;
in	<i>gpio</i>	puntatore alla struttura <a href="#">myGPIO_t</a> che astrae il device myGPIO a cui il display è connesso. Non viene inizializzato dalla funziona, sarà necessario inizializzarlo preventivamente;
in	<i>RS</i>	pin del device myGPIO a cui è associato il segnale RS (data/command) del display LCD;
in	<i>RW</i>	pin del device myGPIO a cui è associato il segnale RW (read/write) del display LCD;
in	<i>E</i>	pin del device myGPIO a cui è associato il segnale E (Enable) del display LCD;
in	<i>Data7</i>	pin del device myGPIO a cui è associato il segnale Data7 del display LCD;
in	<i>Data6</i>	pin del device myGPIO a cui è associato il segnale Data6 del display LCD;
in	<i>Data5</i>	pin del device myGPIO a cui è associato il segnale Data5 del display LCD;
in	<i>Data4</i>	pin del device myGPIO a cui è associato il segnale Data4 del display LCD;

```

1 myGPIO_t gpioDisplay;
2 GPIO_init(&gpioDisplay, XPAR_MYGPIO_3_S00_AXI_BASEADDR, 11, 0, 4, 8);
3 HD44780_LCD_t lcd;
4 HD44780_Init4(&lcd, &gpioDisplay,    GPIO_pin10, GPIO_pin9, GPIO_pin8,
5                                     GPIO_pin0, GPIO_pin1, GPIO_pin2, GPIO_pin3);
6 HD44780_Print(&lcd, "Ciao! Come va");
7 HD44780_MoveToRow2(&lcd);
8 HD44780_Print(&lcd, "lo studio?");

```

**4.2.3.8** void HD44780\_Init8 ( [HD44780\\_LCD\\_t](#) \* *lcd*, [myGPIO\\_t](#) \* *gpio*, [myGPIO\\_mask](#) *RS*, [myGPIO\\_mask](#) *RW*, [myGPIO\\_mask](#) *E*, [myGPIO\\_mask](#) *Data7*, [myGPIO\\_mask](#) *Data6*, [myGPIO\\_mask](#) *Data5*, [myGPIO\\_mask](#) *Data4*, [myGPIO\\_mask](#) *Data3*, [myGPIO\\_mask](#) *Data2*, [myGPIO\\_mask](#) *Data1*, [myGPIO\\_mask](#) *Data0* )

Inizializza un display lcd HD44780 con interfacciamento ad 8 bit.

**Avvertimento**

Se i pin associati ai segnali di pilotaggio del device non sono correttamente configurati come pin di output, il dispositivo non funzionerà correttamente.

Non modificare i campi della struttura dopo che essa sia stata inizializzata.

La struttura [myGPIO\\_t](#), a cui fa riferimento il parametro *gpio*, va inizializzata a parte.

**Parametri**

in, out	<i>lcd</i>	puntatore a struttura di tipo <a href="#">HD44780_LCD_t</a> che descrive un display HD44780 da inizializzare;
in	<i>gpio</i>	puntatore alla struttura <a href="#">myGPIO_t</a> che astrae il device myGPIO a cui il display è connesso. Non viene inizializzato dalla funziona, sarà necessario inizializzarlo preventivamente;
in	<i>RS</i>	pin del device myGPIO a cui è associato il segnale RS (data/command) del display LCD;
in	<i>RW</i>	pin del device myGPIO a cui è associato il segnale RW (read/write) del display LCD;
in	<i>E</i>	pin del device myGPIO a cui è associato il segnale E (Enable) del display LCD;
in	<i>Data7</i>	pin del device myGPIO a cui è associato il segnale Data7 del display LCD;
in	<i>Data6</i>	pin del device myGPIO a cui è associato il segnale Data6 del display LCD;
in	<i>Data5</i>	pin del device myGPIO a cui è associato il segnale Data5 del display LCD;

in	<i>Data4</i>	pin del device myGPIO a cui è associato il segnale Data4 del display LCD;
in	<i>Data3</i>	pin del device myGPIO a cui è associato il segnale Data3 del display LCD;
in	<i>Data2</i>	pin del device myGPIO a cui è associato il segnale Data2 del display LCD;
in	<i>Data1</i>	pin del device myGPIO a cui è associato il segnale Data1 del display LCD;
in	<i>Data0</i>	pin del device myGPIO a cui è associato il segnale Data0 del display LCD;

```

1 myGPIO_t gpioDisplay;
2 myGPIO_Init(&gpioDisplay, XPAR_MYGPIO_3_S00_AXI_BASEADDR);
3 HD44780_LCD_t lcd;
4 HD44780_Init8(&lcd, &gpioDisplay,    GPIO_pin10, GPIO_pin9, GPIO_pin8,
5                                     GPIO_pin0, GPIO_pin1, GPIO_pin2, GPIO_pin3,
6                                     GPIO_pin4, GPIO_pin5, GPIO_pin6, GPIO_pin7);
7 HD44780_Print(&lcd, "Ciao! Come va");
8 HD44780_MoveToRow2(&lcd);
9 HD44780_Print(&lcd, "lo studio?");

```

#### 4.2.3.9 void HD44780\_MoveCursor ( HD44780\_LCD\_t \* lcd, HD44780\_Direction\_t dir )

Sposta il cursore di una posizione a destra o sinistra.

##### Parametri

in	<i>lcd</i>	display da pilotare;
in	<i>dir</i>	direzione in cui spostare il cursore,

##### Si veda anche

direction\_t;

#### 4.2.3.10 void HD44780\_MoveToRow1 ( HD44780\_LCD\_t \* lcd )

Sposta il cursore all'inizio della prima riga.

##### Parametri

in	<i>lcd</i>	display da pilotare;
----	------------	----------------------

#### 4.2.3.11 void HD44780\_MoveToRow2 ( HD44780\_LCD\_t \* lcd )

Sposta il cursore all'inizio della seconda riga.

##### Parametri

in	<i>lcd</i>	display da pilotare;
----	------------	----------------------

#### 4.2.3.12 void HD44780\_Print ( HD44780\_LCD\_t \* lcd, const char \* s )

Stampa una stringa null-terminated di caratteri.

La funzione può essere utilizzata per stampare anche numeri interi e floating point. Si veda gli esempi di cui sotto.

##### Parametri

in	<i>lcd</i>	display da pilotare;
----	------------	----------------------

<i>in</i>	<i>s</i>	puntatore alla stringa null-terminated da stampare sul display;
-----------	----------	---

```

1 // stampa di un intero
2 #include <stdlib.h>
3 ...
4 char str[10]; // assicurarsi di allocare sufficiente spazio per la stampa del numero
5 sprintf(str,"%d", integer_number);
6 error = HD44780_Print(lcd, str);

1 // stampa di un intero
2 #include <stdlib.h>
3 ...
4 char str[10];
5 snprintf(str, 10,"%d", integer_number);
6 error = HD44780_Print(lcd, str);

1 // stampa di un float
2 #include <stdlib.h>
3 ...
4 char str[20]; // assicurarsi di allocare sufficiente spazio per la stampa del numero
5 sprintf(str,"%f", float_number);
6 error = HD44780_Print(lcd, str);

1 // stampa di un float, nel caso in cui la soluzione precedente dovesse non funzionare
2 #include <stdlib.h>
3 ...
4 char str[20];
5 int parte_intera, parte_decimale, moltiplicatore = 1000;
6 // se si desiderano più di tre cifre decimali basta aumentare la potenza del
7 // moltiplicatore
8 // es. cinque cifre decimali ==> moltiplicatore = 100000
9 // si sconsiglia di stampare più di quattro cifre decimali per non causare overflow
10 // nelle istruzioni che seguono
11 parte_intera = (int) float_number;
12 parte_decimale = (int)(float_number * moltiplicatore) - (parte_intera * moltiplicatore);
13 snprintf(str, 20,"%d.%d", parte_intera, parte_decimale);
14 error = HD44780_Print(lcd, str);

```

#### 4.2.3.13 void HD44780\_printBinary32 ( HD44780\_LCD\_t \* *lcd*, uint32\_t *w* )

Stampa una word di 32 bit in binario. (bit più significativo a sinistra)

##### Parametri

<i>in</i>	<i>lcd</i>	
<i>in</i>	<i>w</i>	word da stampare

#### 4.2.3.14 void HD44780\_printBinary64 ( HD44780\_LCD\_t \* *lcd*, uint64\_t *b* )

Stampa un blocco di 64 bit in binario. (bit più significativo a sinistra)

##### Parametri

<i>in</i>	<i>lcd</i>	
<i>in</i>	<i>b</i>	blocco da stampare

#### 4.2.3.15 void HD44780\_printBinary8 ( HD44780\_LCD\_t \* *lcd*, uint8\_t *b* )

Stampa un byte in binario. (bit più significativo a sinistra)

## Parametri

in	<i>lcd</i>	
in	<i>b</i>	byte da stampare

4.2.3.16 void HD44780\_Printc ( HD44780\_LCD\_t \* *lcd*, char *c* )

Stampa un carattere.

## Parametri

in	<i>lcd</i>	display da pilotare;
in	<i>c</i>	carattere da stampare sul display;

4.2.3.17 void HD44780\_printHex32 ( HD44780\_LCD\_t \* *lcd*, uint32\_t *w* )

Stampa una word di 32 bit in esadecimale. (bit più significativo a sinistra)

## Parametri

in	<i>lcd</i>	
in	<i>w</i>	word da stampare

4.2.3.18 void HD44780\_printHex64 ( HD44780\_LCD\_t \* *lcd*, uint64\_t *b* )

Stampa un blocco di 64 bit in esadecimale. (bit più significativo a sinistra)

## Parametri

in	<i>lcd</i>	
in	<i>b</i>	blocco da stampare

4.2.3.19 void HD44780\_printHex8 ( HD44780\_LCD\_t \* *lcd*, uint8\_t *b* )

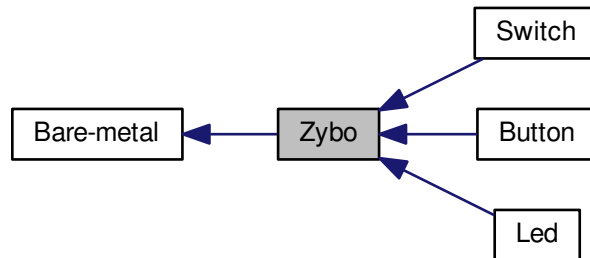
Stampa un byte in esadecimale. (bit più significativo a sinistra)

## Parametri

in	<i>lcd</i>	
in	<i>b</i>	byte da stampare

## 4.3 Zybo

Diagramma di collaborazione per Zybo:



### Moduli

- [Button](#)  
*supporto per gestione dei button su board Digilent Zybo*
- [Led](#)  
*supporto per gestione dei LED su board Digilent Zybo*
- [Switch](#)  
*supporto per gestione degli switch su board Digilent Zybo*

### 4.3.1 Descrizione dettagliata

## 4.4 Button

supporto per gestione dei button su board Digilent Zybo

Diagramma di collaborazione per Button:



### Strutture dati

- struct [ZyboButton\\_t](#)

*Struttura opaca che astrae l'insieme dei button presenti sulla board Digilent Zybo;.*

### Definizioni

- #define [ZyboButton\(i\)](#) ((uint32\_t)(1<<(i)))

*Metodo alternativo per la specifica di uno dei button presenti sulla board Digilent Zybo.*

- #define [ZyboButton\\_DebounceWait](#) 50

*Tempo di attesa (in millisecondi) usato per prevenire il fenomeno del bouncing. Il valore di default è 50, determinato empiricamente. Può essere modificato a piacimento cambiando il valore alla macro seguente.*

### Tipi enumerati (enum)

- enum [ZyboButton\\_mask\\_t](#) { [ZyboButton3](#) = 0x8U, [ZyboButton2](#) = 0x4U, [ZyboButton1](#) = 0x2U, [ZyboButton0](#) = 0x1U }

*Maschere di selezione dei PushButton.*

- enum [ZyboButton\\_status\\_t](#) { [ZyboButton\\_off](#), [ZyboButton\\_on](#) }

*Status di attivo/inattivo dei PushButton.*

### Funzioni

- void [ZyboButton\\_init](#) ([ZyboButton\\_t](#) \*buttons, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) Button3\_pin, [myGPIO\\_mask](#) Button2\_pin, [myGPIO\\_mask](#) Button1\_pin, [myGPIO\\_mask](#) Button0\_pin)

*Inizializza un oggetto di tipo [ZyboButton\\_t](#).*

- void [ZyboButton\\_waitWhileIdle](#) ([ZyboButton\\_t](#) \*buttons)

*Permettere di mettere il programma in attesa attiva finché i button restano inattivi;.*

- void [ZyboButton\\_waitWhileBusy](#) ([ZyboButton\\_t](#) \*buttons)

*Permettere di mettere il programma in attesa attiva finché i button restano attivi;.*

- [ZyboButton\\_status\\_t](#) [ZyboButton\\_getStatus](#) ([ZyboButton\\_t](#) \*buttons, [ZyboButton\\_mask\\_t](#) mask)

*Permette la lettura dello stato dei button presenti sulla board.*

#### 4.4.1 Descrizione dettagliata

supporto per gestione dei button su board Digilent Zybo

#### 4.4.2 Documentazione delle definizioni

##### 4.4.2.1 `#define ZyboButton( i ) ((uint32_t)(1<<(i)))`

Metodo alternativo per la specifica di uno dei button presenti sulla board Digilent Zybo.

Parametri

<i>i</i>	indice del button da selezionare, da 0 a 3
----------	--

Restituisce

maschera di selezione del button i-esimo

##### 4.4.2.2 `#define ZyboButton_DebounceWait 50`

Tempo di attesa (in millisecondi) usato per prevenire il fenomeno del bouncing. Il valore di default è 50, determinato empiricamente. Può essere modificato a piacimento cambiando il valore alla macro seguente.

#### 4.4.3 Documentazione dei tipi enumerati

##### 4.4.3.1 `enum ZyboButton_mask_t`

Maschere di selezione dei PushButton.

Valori del tipo enumerato

***ZyboButton3*** ZyboButton3, seleziona il button 3 sulla board Digilent Zybo;.

***ZyboButton2*** ZyboButton2, seleziona il button 2 sulla board Digilent Zybo;.

***ZyboButton1*** ZyboButton1, seleziona il button 1 sulla board Digilent Zybo;.

***ZyboButton0*** ZyboButton0, seleziona il button 0 sulla board Digilent Zybo;.

##### 4.4.3.2 `enum ZyboButton_status_t`

Status di attivo/inattivo dei PushButton.

Valori del tipo enumerato

***ZyboButton\_off*** ZyboButton\_off, corrisponde al valore logico '0', indica che un pushbutton è inattivo;.

***ZyboButton\_on*** ZyboButton\_on, corrisponde al valore logico '1', indica che un pushbutton è attivo;.

#### 4.4.4 Documentazione delle funzioni

##### 4.4.4.1 `ZyboButton_status_t ZyboButton_getStatus ( ZyboButton_t * buttons, ZyboButton_mask_t mask )`

Permette la lettura dello stato dei button presenti sulla board.

Parametri

<i>in</i>	<i>buttons</i>	puntatore a struttura <a href="#">ZyboButton_t</a> , che astrae l'insieme dei button presenti sulla board Digilent Zybo;
-----------	----------------	--



<i>in</i>	<i>mask</i>	maschera di selezione dei button, quelli non selezionati non vengono tenuti in considerazione
-----------	-------------	---

**Restituisce**

status status dei button

**Valori di ritorno**

<i>ZyboButton_on</i>	se uno dei button selezionati è attivo;
<i>ZyboButton_off</i>	altrimenti

```

1 ZyboButton_status_t button_status = ZyboButton_getStatus(&buttons, ZyboButton3);           // leggo lo
  stato ddi button 3
2 ZyboLed_status_t led_status = (button_status == ZyboButton_on ? ZyboLed_on : ZyboLed_off); // se lo stato
  è attivo accendo il led 3
3 ZyboLed_setStatus(&leds, ZyboLed3, led_status);                                         //
  accendo/spengo led 3

```

**Avvertimento**

Usa la macro assert per verificare che:

- buttons non sia un puntatore nullo;
- buttons->gpio non sia un puntatore nullo

**4.4.4.2** void ZyboButton\_init ( ZyboButton\_t \* buttons, myGPIO\_t \* gpio, myGPIO\_mask Button3\_pin, myGPIO\_mask Button2\_pin, myGPIO\_mask Button1\_pin, myGPIO\_mask Button0\_pin )

Inizializza un oggetto di tipo [ZyboButton\\_t](#).

**Parametri**

<i>in, out</i>	<i>buttons</i>	puntatore a struttura <a href="#">ZyboButton_t</a> , che astrae l'insieme dei button presenti sulla board Digilent Zybo;
<i>in</i>	<i>gpio</i>	puntatore a struttura <a href="#">myGPIO_t</a> , che astrae un device myGPIO; non viene inizializzato dalla funziona, sarà necessario inizializzarlo preventivamente; si faccia riferimento all'esempio riportato di seguito
<i>in</i>	<i>Button3_pin</i>	pin del device myGPIO a cui è associato il button 3 della board Digilent Zybo;
<i>in</i>	<i>Button2_pin</i>	pin del device myGPIO a cui è associato il button 2 della board Digilent Zybo;
<i>in</i>	<i>Button1_pin</i>	pin del device myGPIO a cui è associato il button 1 della board Digilent Zybo;
<i>in</i>	<i>Button0_pin</i>	pin del device myGPIO a cui è associato il button 0 della board Digilent Zybo;

```

1 myGPIO_t gpioButton;
2 GPIO_init(&gpioButton, XPAR_MYGPIO_1_S00_AXI_BASEADDR, 4, 0, 4, 8);
3 ZyboButton_t buttons;
4 ZyboButton_init(&buttons, &gpioButton, GPIO_pin3, GPIO_pin2, GPIO_pin1, GPIO_pin0);

```

**Avvertimento**

Usa la macro assert per verificare che:

- buttons non sia un puntatore nullo;
- gpio non sia un puntatore nullo
- ButtonN\_pin siano tutti pin differenti

#### 4.4.4.3 void ZyboButton\_waitWhileBusy ( ZyboButton\_t \* buttons )

Permettere di mettere il programma in attesa attiva finché i button restano attivi;.

##### Avvertimento

La funzione integra le funzionalità di debouncing. Il tempo di attesa è determinato sulla base del valore della macro ZyboButton\_DebounceWait. Per l'attesa viene usata la funzione usleep() di stdlib.

##### Parametri

in	buttons	puntatore a struttura <a href="#">ZyboButton_t</a> , che astrae l'insieme dei button presenti sulla board Digilent Zybo;
----	---------	--

```

1 ZyboButton_waitWhileIdle (&buttons);
2 ZyboButton_status_t button3_status = ZyboButton_getStatus (&buttons, ZyboButton3);           // leggo lo
   stato ddi button 3
3 ZyboButton_status_t button2_status = ZyboButton_getStatus (&buttons, ZyboButton2);           // leggo lo
   stato ddi button 2
4 ZyboButton_status_t button1_status = ZyboButton_getStatus (&buttons, ZyboButton1);           // leggo lo
   stato ddi button 1
5 ZyboButton_status_t button0_status = ZyboButton_getStatus (&buttons, ZyboButton0);           // leggo lo
   stato ddi button 0
6 ZyboButton_waitWhileBusy (&buttons);

```

##### Avvertimento

Usa la macro assert per verificare che:

- buttons non sia un puntatore nullo;
- buttons->gpio non sia un puntatore nullo

#### 4.4.4.4 void ZyboButton\_waitWhileIdle ( ZyboButton\_t \* buttons )

Permettere di mettere il programma in attesa attiva finché i button restano inattivi;.

##### Avvertimento

La funzione integra le funzionalità di debouncing. Il tempo di attesa è determinato sulla base del valore della macro ZyboButton\_DebounceWait. Per l'attesa viene usata la funzione usleep() di stdlib.

##### Parametri

in	buttons	puntatore a struttura <a href="#">ZyboButton_t</a> , che astrae l'insieme dei button presenti sulla board Digilent Zybo;
----	---------	--

```

1 ZyboButton_waitWhileIdle (&buttons);
2 ZyboButton_status_t button3_status = ZyboButton_getStatus (&buttons, ZyboButton3);           // leggo lo
   stato ddi button 3
3 ZyboButton_status_t button2_status = ZyboButton_getStatus (&buttons, ZyboButton2);           // leggo lo
   stato ddi button 2
4 ZyboButton_status_t button1_status = ZyboButton_getStatus (&buttons, ZyboButton1);           // leggo lo
   stato ddi button 1
5 ZyboButton_status_t button0_status = ZyboButton_getStatus (&buttons, ZyboButton0);           // leggo lo
   stato ddi button 0
6 ZyboButton_waitWhileBusy (&buttons);

```

##### Avvertimento

Usa la macro assert per verificare che:

- buttons non sia un puntatore nullo;
- buttons->gpio non sia un puntatore nullo

## 4.5 Led

supporto per gestione dei LED su board Digilent Zybo

Diagramma di collaborazione per Led:



### Strutture dati

- struct `ZyboLed_t`  
*Struttura opaca che astrae l'insieme dei Led presenti sulla board Digilent Zybo;*

### Definizioni

- #define `ZyboLed(i)` `((uint32_t)(1<<(i)))`  
*Metodo alternativo per la specifica di uno dei led presenti sulla board Digilent Zybo.*

### Tipi enumerati (enum)

- enum `ZyboLed_mask_t` { `ZyboLed3` = 0x8U, `ZyboLed2` = 0x4U, `ZyboLed1` = 0x2U, `ZyboLed0` = 0x1U }  
*Maschere di selezione dei led.*
- enum `ZyboLed_status_t` { `ZyboLed_off`, `ZyboLed_on` }  
*Status di accensione/spegnimento dei led.*

### Funzioni

- void `ZyboLed_init` (`ZyboLed_t` \*leds, `myGPIO_t` \*gpio, `myGPIO_mask` Led3\_pin, `myGPIO_mask` Led2\_pin, `myGPIO_mask` Led1\_pin, `myGPIO_mask` Led0\_pin)  
*Inizializza un oggetto di tipo `ZyboLed_t`.*
- void `ZyboLed_setStatus` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask, `ZyboLed_status_t` status)  
*Permette di accendere/spegnere i Led sulla board.*
- void `ZyboLed_toggle` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask)  
*Permette di accendere/spegnere i Led sulla board, invertendone il valore.*

#### 4.5.1 Descrizione dettagliata

supporto per gestione dei LED su board Digilent Zybo

#### 4.5.2 Documentazione delle definizioni

##### 4.5.2.1 #define `ZyboLed( i ) ((uint32_t)(1<<(i)))`

Metodo alternativo per la specifica di uno dei led presenti sulla board Digilent Zybo.

## Parametri

<i>in</i>	<i>i</i>	indice del led da selezionare, da 0 a 3
-----------	----------	---

## Restituisce

maschera di selezione del led i-esimo

## 4.5.3 Documentazione dei tipi enumerati

## 4.5.3.1 enum ZyboLed\_mask\_t

Maschere di selezione dei led.

## Valori del tipo enumerato

**ZyboLed3** ZyboLed3, seleziona il led 3 sulla board Digilent Zybo;.

**ZyboLed2** ZyboLed2, seleziona il led 2 sulla board Digilent Zybo;.

**ZyboLed1** ZyboLed1, seleziona il led 1 sulla board Digilent Zybo;.

**ZyboLed0** ZyboLed0, seleziona il led 0 sulla board Digilent Zybo;.

## 4.5.3.2 enum ZyboLed\_status\_t

Status di accensione/spegnimento dei led.

## Valori del tipo enumerato

**ZyboLed\_off** ZyboLed\_off, corrisponde al valore logico '0', per lo spegnimento dei Led.

**ZyboLed\_on** ZyboLed\_on, corrisponde al valore logico '1', per l'accensione dei Led.

## 4.5.4 Documentazione delle funzioni

4.5.4.1 void ZyboLed\_init ( ZyboLed\_t \* *leds*, myGPIO\_t \* *gpio*, myGPIO\_mask Led3\_pin, myGPIO\_mask Led2\_pin, myGPIO\_mask Led1\_pin, myGPIO\_mask Led0\_pin )

Inizializza un oggetto di tipo [ZyboLed\\_t](#).

Inizializza un oggetto di tipo [ZyboLed\\_t](#), che astrae e consente di pilotare i led presenti sulla board Digilent Zybo. Per il pilotaggio viene usato il modulo myGPIO ed un puntatore ad una struttura [myGPIO\\_t](#) che lo astrae. Tale struttura non viene inizializzata dalla funzione ZyboLed\_init, per cui sarà necessario inizializzarlo preventivamente. La funzione, però, si assume l'onere di configurare i pin del device myGPIO a cui i led sono connessi.

## Parametri

<i>in, out</i>	<i>leds</i>	puntatore a struttura <a href="#">ZyboLed_t</a> , che astrae l'insieme dei Led presenti sulla board Digilent Zybo;
<i>in</i>	<i>gpio</i>	puntatore a struttura <a href="#">myGPIO_t</a> , che astrae un device myGPIO; la struttura <a href="#">myGPIO_t</a> non viene inizializzata dalla funzione, per cui sarà necessario farlo preventivamente; si faccia riferimento all'esempio riportato di seguito.

in	<i>Led3_pin</i>	pin del device myGPIO a cui è associato il Led3 della board Digilent Zybo;
in	<i>Led2_pin</i>	pin del device myGPIO a cui è associato il Led2 della board Digilent Zybo;
in	<i>Led1_pin</i>	pin del device myGPIO a cui è associato il Led1 della board Digilent Zybo;
in	<i>Led0_pin</i>	pin del device myGPIO a cui è associato il Led0 della board Digilent Zybo;

```

1 myGPIO_t gpioLed;
2 GPIO_init(&gpioLed, XPAR_MYGPIO_0_S00_AXI_BASEADDR, 4, 0, 4, 8); // inizializzazione del
  device myGPIO
3 ZyboLed_t leds;
4 ZyboLed_init(&leds, &gpioLed, GPIO_pin3, GPIO_pin2, GPIO_pin1, GPIO_pin0); // inzializzazione della
  struttura ZyboLed_t

```

#### Avvertimento

Usa la macro assert per verificare che:

- leds non sia un puntatore nullo;
- gpio non sia un puntatore nullo
- LedN\_pin siano tutti pin differenti

#### 4.5.4.2 void ZyboLed\_setStatus ( ZyboLed\_t \* leds, ZyboLed\_mask\_t mask, ZyboLed\_status\_t status )

Permette di accendere/spegnere i Led sulla board.

##### Parametri

in	<i>leds</i>	puntatore a struttura <a href="#">ZyboLed_t</a> , che astrae l'insieme dei Led presenti sulla board Digilent Zybo;
in	<i>mask</i>	maschera di selezione dei led, quelli non selezionati vengono lasciati inalterati
in	<i>status</i>	status dei led, ZyboLed_on per accendere, ZyboLed_off per spegnere

```

1 ZyboLed_setStatus(&leds, ZyboLed3 | ZyboLed1, ZyboLed_on); // accensione dei Led 3 ed 1
2 ZyboLed_setStatus(&leds, ZyboLed3 | ZyboLed1, ZyboLed_off); // spegnimento dei Led 3 ed 1
3 ZyboLed_setStatus(&leds, ZyboLed2 | ZyboLed0, ZyboLed_on); // accensione dei Led 2 ed 0
4 ZyboLed_setStatus(&leds, ZyboLed0, ZyboLed_off); // spegnimento dei Led 2 ed 0
5 ZyboLed_setStatus(&leds, ZyboLed3 | ZyboLed1 | ZyboLed2 | ZyboLed0, ZyboLed_on); // accensione di
  tutti i led
6 ZyboLed_setStatus(&leds, ZyboLed3 | ZyboLed1 | ZyboLed2 | ZyboLed0, ZyboLed_off); // spegnimento di tutti
  i led

```

#### Avvertimento

Usa la macro assert per verificare che:

- leds non sia un puntatore nullo;
- leds->gpio non sia un puntatore nullo

#### 4.5.4.3 void ZyboLed\_toggle ( ZyboLed\_t \* leds, ZyboLed\_mask\_t mask )

Permette di accendere/spegnere i Led sulla board, invertendone il valore.

##### Parametri

in	<i>leds</i>	puntatore a struttura <a href="#">ZyboLed_t</a> , che astrae l'insieme dei Led presenti sulla board Digilent Zybo;
----	-------------	--

<code>in</code>	<code>mask</code>	maschera di selezione dei led, quelli non selezionati vengono lasciati inalterati
-----------------	-------------------	---

```
1 ZyboLed_toggle(&leds, ZyboLed3 | ZyboLed1); // accensione/spengimento dei Led 3 ed 1
```

#### Avvertimento

Usa la macro assert per verificare che:

- leds non sia un puntatore nullo;
- leds->gpio non sia un puntatore nullo

## 4.6 Switch

supporto per gestione degli switch su board Digilent Zybo

Diagramma di collaborazione per Switch:



### Strutture dati

- struct `ZyboSwitch_t`

*Struttura opaca che astrae l'insieme degli switch presenti sulla board Digilent Zybo;.*

### Definizioni

- #define `ZyboSwitch(i)` `((uint32_t)(1<<(i)))`

*Metodo alternativo per la specifica di uno degli switch presenti sulla board Digilent Zybo.*

### Tipi enumerati (enum)

- enum `ZyboSwitch_mask_t` { `ZyboSwitch3` = 0x8U, `ZyboSwitch2` = 0x4U, `ZyboSwitch1` = 0x2U, `ZyboSwitch0` = 0x1U }

*Maschere di selezione degli switch.*

- enum `ZyboSwitch_status_t` { `ZyboSwitch_off`, `ZyboSwitch_on` }

*Status di attivo/inattivo degli switch.*

### Funzioni

- void `ZyboSwitch_init` (`ZyboSwitch_t` \*switches, `myGPIO_t` \*gpio, `myGPIO_mask` Switch3\_pin, `myGPIO_mask` Switch2\_pin, `myGPIO_mask` Switch1\_pin, `myGPIO_mask` Switch0\_pin)

*Inizializza un oggetto di tipo `ZyboSwitch_t`.*

- `ZyboSwitch_status_t` `ZyboSwitch_getStatus` (`ZyboSwitch_t` \*switches, `ZyboSwitch_mask_t` mask)

*Permette la lettura dello stato degli switch presenti sulla board.*

#### 4.6.1 Descrizione dettagliata

supporto per gestione degli switch su board Digilent Zybo

#### 4.6.2 Documentazione delle definizioni

##### 4.6.2.1 #define `ZyboSwitch( i )` `((uint32_t)(1<<(i)))`

Metodo alternativo per la specifica di uno degli switch presenti sulla board Digilent Zybo.

## Parametri

<i>in</i>	<i>i</i>	indice dello switch da selezionare, da 0 a 3
-----------	----------	--

## Restituisce

maschera di selezione dello switch *i*-esimo

## 4.6.3 Documentazione dei tipi enumerati

## 4.6.3.1 enum ZyboSwitch\_mask\_t

Maschere di selezione degli switch.

## Valori del tipo enumerato

**ZyboSwitch3** ZyboSwitch3, seleziona lo switch 3 sulla board Digilent Zybo;.

**ZyboSwitch2** ZyboSwitch2, seleziona lo switch 2 sulla board Digilent Zybo;.

**ZyboSwitch1** ZyboSwitch1, seleziona lo switch 1 sulla board Digilent Zybo;.

**ZyboSwitch0** ZyboSwitch0, seleziona lo switch 0 sulla board Digilent Zybo;.

## 4.6.3.2 enum ZyboSwitch\_status\_t

Status di attivo/inattivo degli switch.

## Valori del tipo enumerato

**ZyboSwitch\_off** ZyboSwitch\_off, corrisponde al valore logico '0', indica che lo switch è inattivo;.

**ZyboSwitch\_on** ZyboSwitch\_on, corrisponde al valore logico '1', indica che lo switch è attivo;.

## 4.6.4 Documentazione delle funzioni

## 4.6.4.1 ZyboSwitch\_status\_t ZyboSwitch\_getStatus ( ZyboSwitch\_t \* switches, ZyboSwitch\_mask\_t mask )

Permette la lettura dello stato degli switch presenti sulla board.

## Parametri

<i>in</i>	<i>switches</i>	puntatore a struttura <a href="#">ZyboSwitch_t</a> , che astrae l'insieme degli switch presenti sulla board Digilent Zybo;
<i>in</i>	<i>mask</i>	maschera di selezione degli switch, quelli non selezionati non vengono tenuti in considerazione

## Restituisce

status status degli switch

## Valori di ritorno

<i>ZyboSwitch_on</i>	se uno degli switch selezionati è attivo;
<i>ZyboSwitch_off</i>	altrimenti

```
1 ZyboSwitch_status_t switch_status = ZyboSwitch_getStatus(&switches, ZyboSwitch3);           // leggo lo
  stato dello switch 3
2 ZyboLed_status_t led_status = (switch_status == ZyboSwitch_on ? ZyboLed_on : ZyboLed_off); // se lo switch
  3 è attivo accendo il led 3
3 ZyboLed_setStatus(&leds, ZyboLed3, led_status);                                           //
  accendo/spengo led 3
```



**Avvertimento**

Usa la macro assert per verificare che:

- switches non sia un puntatore nullo;
- switches->gpio non sia un puntatore nullo

**4.6.4.2** void ZyboSwitch\_init ( ZyboSwitch\_t \* switches, myGPIO\_t \* gpio, myGPIO\_mask Switch3\_pin, myGPIO\_mask Switch2\_pin, myGPIO\_mask Switch1\_pin, myGPIO\_mask Switch0\_pin )

Inizializza un oggetto di tipo [ZyboSwitch\\_t](#).

**Parametri**

in, out	<i>switches</i>	puntatore a struttura <a href="#">ZyboSwitch_t</a> , che astrae l'insieme degli switch presenti sulla board Digilent Zybo;
in	<i>gpio</i>	puntatore a struttura <a href="#">myGPIO_t</a> , che astrae un device myGPIO; non viene inizializzato dalla funziona, sarà necessario iniziarlo preventivamente; si faccia riferimento all'esempio riportato di seguito
in	<i>Switch3_pin</i>	pin del device myGPIO a cui è associato lo switch 3 della board Digilent Zybo;
in	<i>Switch2_pin</i>	pin del device myGPIO a cui è associato lo switch 2 della board Digilent Zybo;
in	<i>Switch1_pin</i>	pin del device myGPIO a cui è associato lo switch 1 della board Digilent Zybo;
in	<i>Switch0_pin</i>	pin del device myGPIO a cui è associato lo switch 0 della board Digilent Zybo;

```
1 myGPIO_t gpioSwitch;
2 GPIO_init(&gpioSwitch, XPAR_MYGPIO_1_S00_AXI_BASEADDR, 4, 0, 4, 8);
3 ZyboSwitch_t switches;
4 ZyboSwitch_init(&switches, &gpioSwitch, GPIO_pin3, GPIO_pin2, GPIO_pin1, GPIO_pin0);
```

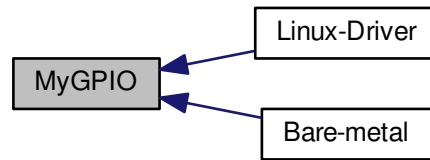
**Avvertimento**

Usa la macro assert per verificare che:

- switches non sia un puntatore nullo;
- gpio non sia un puntatore nullo
- SwitchN\_pin siano tutti pin differenti

## 4.7 MyGPIO

Diagramma di collaborazione per MyGPIO:



### Moduli

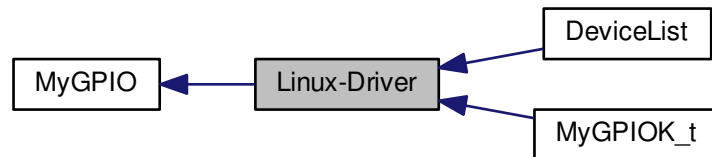
- [Bare-metal](#)  
*device-driver OO-like bare-metal per device myGPIO*
- [Linux-Driver](#)  
*Device-driver in kernel-mode per myGPIO.*

#### 4.7.1 Descrizione dettagliata

## 4.8 Linux-Driver

Device-driver in kernel-mode per myGPIO.

Diagramma di collaborazione per Linux-Driver:



### Moduli

- [DeviceList](#)

*Definisce la struttura dati [myGPIOK\\_list\\_t](#), la quale mantiene un riferimento agli oggetti [myGPIOK\\_t](#) gestiti dal driver.*

- [MyGPIOK\\_t](#)

*Definisce l'oggetto [myGPIOK\\_t](#), che rappresenta un device myGPIO a livello kernel.*

### Definizioni

- `#define DRIVER_NAME "myGPIOK"`

*Nome identificativo del device-driver. DEVE corrispondere al valore del campo "compatible" nel device tree source.*

- `#define DRIVER_FNAME "myGPIOK%d"`

*Nome del file creato in /dev/ per ciascuno dei device.*

- `#define MAX_NUM_OF_DEVICES 15`

*Numero massimo di device gestibili.*

- `#define myGPIOK_USED_INT 0xFFFFFFFFU`

*Maschera di abilitazione degli interrupt per i singoli pin.*

### Funzioni

- `MODULE_LICENSE ("GPL")`

- `MODULE_AUTHOR ("Salvatore Barone <salvator.barone@gmail.com>")`

- `MODULE_DESCRIPTION ("myGPIO device-driver in kernel mode")`

- `MODULE_VERSION ("3.2")`

- `MODULE_ALIAS (DRIVER_NAME)`

- `static int myGPIOK_probe (struct platform_device *op)`

*Viene chiamata quando il modulo viene inserito.*

- `static int myGPIOK_remove (struct platform_device *op)`

- `static int myGPIOK_open (struct inode *inode, struct file *file_ptr)`

*Invocata all'apertura del file corrispondente al device.*

- `static int myGPIOK_release (struct inode *inode, struct file *file_ptr)`

*Invocata alla chiusura del file corrispondente al device.*

- `static loff_t myGPIOK_llseek (struct file *file_ptr, loff_t off, int whence)`

- Implementa le system-call lseek() e llseek().*

  - static unsigned int **myGPIOK\_poll** (struct file \*file\_ptr, struct poll\_table\_struct \*wait)
 

*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*
  - static ssize\_t **myGPIOK\_read** (struct file \*file\_ptr, char \*buf, size\_t count, loff\_t \*off)
 

*Legge dati dal device.*
  - static ssize\_t **myGPIOK\_write** (struct file \*file\_ptr, const char \*buf, size\_t size, loff\_t \*off)
 

*Invia dati al device.*
  - static irqreturn\_t **myGPIOK\_irq\_handler** (int irq, struct pt\_regs \*regs)
 

*Interrupt-handler.*
  - **MODULE\_DEVICE\_TABLE** (of, **myGPIOK\_match**)
 

*Inserisce una nuova entry nella tabella delle corrispondenze device - driver.*
  - **module\_platform\_driver** (**myGPIOK\_driver**)
 

*la macro **module\_platform\_driver**() prende in input la struttura platform\_driver ed implementa le funzioni module\_init() e module\_close() standard, chiamate quando il modulo viene caricato o rimosso dal kernel.*
  - int **myGPIOK\_Init** (**myGPIOK\_t** \*myGPIOK\_device, struct module \*owner, struct platform\_device \*op, struct class \*class, const char \*driver\_name, const char \*device\_name, uint32\_t serial, struct file\_operations \*f\_ops, irq\_handler\_t irq\_handler, uint32\_t irq\_mask)
 

*Inizializza una struttura **myGPIOK\_t** e configura il device corrispondente.*
  - void **myGPIOK\_Destroy** (**myGPIOK\_t** \*device)
 

*Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.*
  - void **myGPIOK\_SetCanRead** (**myGPIOK\_t** \*device)
 

*Set del flag "interrupt occurred" (canRead)*
  - void **myGPIOK\_ResetCanRead** (**myGPIOK\_t** \*device)
 

*Reset del flag "interrupt occurred" (canRead)*
  - void **myGPIOK\_TestCanReadAndSleep** (**myGPIOK\_t** \*device)
 

*Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.*
  - unsigned **myGPIOK\_GetPollMask** (**myGPIOK\_t** \*device, struct file \*file\_ptr, struct poll\_table\_struct \*wait)
 

*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*
  - void **myGPIOK\_IncrementTotal** (**myGPIOK\_t** \*device)
 

*Incrementa il contatore degli interrupt per un particolare device.*
  - void **myGPIOK\_WakeUp** (**myGPIOK\_t** \*device)
 

*Risveglia i process in attesa sulle code di read e poll.*
  - void \* **myGPIOK\_GetDeviceAddress** (**myGPIOK\_t** \*device)
 

*Restituisce l'indirizzo virtuale di memoria cui è mappato un device.*
  - void **myGPIOK\_GlobalInterruptEnable** (**myGPIOK\_t** \*device)
 

*Abilita gli interrupt globali;.*
  - void **myGPIOK\_GlobalInterruptDisable** (**myGPIOK\_t** \*device)
 

*Disabilita gli interrupt globali;.*
  - void **myGPIOK\_PinInterruptEnable** (**myGPIOK\_t** \*device, unsigned mask)
 

*Abilita gli interrupt per i singoli pin del device.*
  - void **myGPIOK\_PinInterruptDisable** (**myGPIOK\_t** \*device, unsigned mask)
 

*Disabilita gli interrupt per i singoli pin del device.*
  - unsigned **myGPIOK\_PendingPinInterrupt** (**myGPIOK\_t** \*device)
 

*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
  - void **myGPIOK\_PinInterruptAck** (**myGPIOK\_t** \*device, unsigned mask)
 

*Invia al device notifica di servizio di un interrupt;.*

## Variabili

- static `myGPIOK_list_t * device_list = NULL`  
*Array di puntatori a struttura `myGPIOK_t`, contenente tutti i dati necessari al device driver.*
- static struct class \* `myGPIOK_class = NULL`  
*Classe del device. Ai device-drivers viene associata una classe ed un device-name. Per creare ed associare una classe ad un device driver si può usare la seguente.*
- static struct of\_device\_id `myGPIOK_match []`  
*Identifica il device all'interno del device tree.*
- static struct platform\_driver `myGPIOK_driver`  
*Definisce quali funzioni `probe()` e `remove()` chiamare quando viene caricato un driver.*
- static struct file\_operations `myGPIOK_fops`  
*mantiene puntatori a funzioni che definiscono il gli operatori che agiscono su un file/device.*

### 4.8.1 Descrizione dettagliata

Device-driver in kernel-mode per myGPIO.

#### Avvertimento

Se nel device tree source non viene indicato

```
compatible = "myGPIOK";
```

tra i driver compatibili con il device, il driver myGPIOK non viene correttamente istanziato ed il programma userspace non funzionerà.

#### Descrizione generale del driver

Il modulo driver implementa definisce il tipo `myGPIOK_t` ed implementa le seguenti funzioni:

- `myGPIOK_probe()`: richiamata quando il modulo, o un device compatibile col modulo, viene inserito;
- `myGPIOK_remove()`: richiamata quando il modulo, o un device compatibile, viene rimosso;
- `myGPIOK_open()`: implementa la system call `open()`;
- `myGPIOK_llseek()`: implementa la system call `seek()`;
- `myGPIOK_write()`: implementa la system call `write()`;
- `myGPIOK_irq_handler()`: implementa la ISR dedicata alla gestione delle interruzioni provenienti dal device;
- `myGPIOK_poll()`: implementa il back-end di tre diverse system-calls (`poll`, `epoll` e `select`);
- `myGPIOK_read()`: implementa la system call `read`.

Nel seguito viene presentato un breve excursus su tutto ciò che c'è da sapere per comprendere come funziona un device-driver e come poterne scrivere uno. Dopo aver letto il seguito, si consiglia, in ordine, di leggere, in ordine, la documentazione della struttura `myGPIOK_t`, poi quella delle funzioni

- `myGPIOK_probe()`;
- `myGPIOK_open()`;
- `myGPIOK_llseek()`;
- `myGPIOK_write()`;
- `myGPIOK_read()`;
- `myGPIOK_irq_handler()`;

## Platform-device

I device driver, anche se sono moduli kernel, non si scrivono come normali moduli Kernel. I "platform-device" sono device che non possono annunciarsi al software (non possono dire "Hey, sono qui!" al sistema operativo), quindi sono intrinsecamente "non-scopribili", nel senso che il sistema, al boot, deve sapere che ci sono, ma non è in grado di scoprirli. A differenza dei device PCI o USB, che non sono platform-device, un device I<sup>2</sup>C non viene enumerato a livello hardware, per cui è necessario che il sistema operativo sappia, a tempo di "compilazione", cioè prima del boot - quale device sia connesso al bus I<sup>2</sup>C. I non-discoverable devices stanno proliferando molto velocemente nel mondo embedded, per cui il Kernel Linux offre ancora la possibilità di specificare quale hardware sia presente nel sistema. Bisogna distinguere in:

- Platform Driver
- Platform Device

Per quanto riguarda la parte driver, il kernel Linux kernel definisce un insieme di operazioni standard che possono essere effettuate su un platform-device. Un riferimento può essere [http://lxr.free-electrons.com/source/include/linux/platform\\_device.h#L173](http://lxr.free-electrons.com/source/include/linux/platform_device.h#L173). Le callbacks probe() e remove() costituiscono l'insieme minimo di operazioni che devono essere implementate. Tali funzioni devono avere gli stessi parametri delle due seguenti, ma possono avere nome qualsiasi.

```
static int sample_drv_probe(struct platform_device *pdev) {
    //Empty Probe function.
}

static int sample_drv_remove(struct platform_device *pdev) {
    //Empty remove function.
}
```

La definizione di quali funzioni probe() e remove() chiamare quando viene caricato un driver viene effettuato attraverso la seguente struttura e la chiamata alla macro `module_platform_driver()`, la quale prende in input la struttura seguente ed implementa, al posto nostro, le funzioni module\_init() e module\_close() standard, chiamate quando il modulo viene caricato o rimosso dal kernel.

```
static struct platform_driver sample_pldriver = {
    .probe = sample_drv_probe,
    .remove = sample_drv_remove,
    .driver = {
        .name = DRIVER_NAME,
    },
};

module_platform_driver(sample_pldriver);
```

Si noti DRIVER\_NAME: deve essere identica alla stringa indicata, nel device-tree, al campo "compatible".

Affinché il driver possa essere caricato a caldo, è necessario aggiungere alla struttura di cui sopra qualche informazione in più. Tutti i device-driver devono esporre un ID. A tempo di compilazione, il processo di build estrae queste informazioni dai driver per la preparazione di una tabella. Quando si "inserisce" il device, la tabella viene riferita dal kernel e, se viene trovata una entry corrispondente al driver per quel device, il driver viene caricato ed inizializzato.

Usando la struttura

```
static struct of_device_id device_match[] = {
    {.compatible = DRIVER_NAME},
    {},
};

MODULE_DEVICE_TABLE(of, device_match);
```

si identifica un particolare device. La macro `MODULE_DEVICE_TABLE()` viene usata per inserire una nuova entry nella tabella accennata precedentemente. Alla struttura platform\_driver possono essere aggiunte anche queste informazioni, per cui essa si presenterà come riportato di seguito.

```
static struct platform_driver sample_pldriver = {
    .probe = sample_drv_probe,
```

```

.remove = sample_drv_remove,
.driver = {
    .name = DRIVER_NAME,
    .owner = THIS_MODULE,
    .of_match_table = device_match,
},
};

```

## 4.8.2 Documentazione delle definizioni

### 4.8.2.1 #define DRIVER\_FNAME "myGPIOK%d"

Nome del file creato in /dev/ per ciascuno dei device.

### 4.8.2.2 #define DRIVER\_NAME "myGPIOK"

Nome identificativo del device-driver. DEVE corrispondere al valore del campo "compatible" nel device tree source.

### 4.8.2.3 #define MAX\_NUM\_OF\_DEVICES 15

Numero massimo di device gestibili.

In realtà, il numero di device gestibili è virtualmente illimitato. Il fattore limitante è la memoria disponibile ed in numero di Major-minor number disponibili. MAX\_NUM\_OF\_DEVICES incide sulla dimensione della struttura dati myGPIOK\_list\_t che mantiene i riferimenti alle strutture di controllo dei diversi device.

### 4.8.2.4 #define myGPIOK\_USED\_INT 0xFFFFFFFFU

Maschea di abilitazione degli interrupt per i singoli pin.

## 4.8.3 Documentazione delle funzioni

### 4.8.3.1 MODULE\_ALIAS ( DRIVER\_NAME )

### 4.8.3.2 MODULE\_AUTHOR ( "Salvatore Barone <salvator.barone@gmail.com>" )

### 4.8.3.3 MODULE\_DESCRIPTION ( "myGPIO device-driver in kernel mode" )

### 4.8.3.4 MODULE\_DEVICE\_TABLE ( of , myGPIOK\_match )

Inserisce una nuova entry nella tabella delle corrispondenze device - driver.

Parametri

in, out	of	riferimento alla tabella
in	myGPIOK_↔ match	struttura of_device_id

### 4.8.3.5 MODULE\_LICENSE ( "GPL" )

### 4.8.3.6 module\_platform\_driver ( myGPIOK\_driver )

la macro `module_platform_driver()` prende in input la struttura `platform_driver` ed implementa le funzioni `module_↔init()` e `module_close()` standard, chiamate quando il modulo viene caricato o rimosso dal kernel.

## Parametri

in	<i>myGPIOK_driver</i>	struttura platform_driver associata al driver
----	-----------------------	---

## 4.8.3.7 MODULE\_VERSION ( "3.2" )

## 4.8.3.8 void myGPIOK\_Destroy ( myGPIOK\_t \* device )

Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , specifica il particolare device su cui agire
----	---------------	--

## 4.8.3.9 void\* myGPIOK\_GetDeviceAddress ( myGPIOK\_t \* device )

Restituisce l'indirizzo virtuale di memoria cui è mappato un device.

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	---------------	---

## 4.8.3.10 unsigned myGPIOK\_GetPollMask ( myGPIOK\_t \* device, struct file \* file\_ptr, struct poll\_table\_struct \* wait )

Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
in, out	<i>file</i>	
in, out	<i>wait</i>	

## Restituisce

restituisce una maschera di bit che indica se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili.

Questo metodo è il back-end di tre diverse system-calls: poll, epoll e select, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno.

## 4.8.3.11 void myGPIOK\_GlobalInterruptDisable ( myGPIOK\_t \* device )

Disabilita gli interrupt globali;.

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	---------------	---

## 4.8.3.12 void myGPIOK\_GlobalInterruptEnable ( myGPIOK\_t \* device )

Abilita gli interrupt globali;.



## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	---------------	---

4.8.3.13 void myGPIOK\_IncrementTotal ( [myGPIOK\\_t](#) \* *device* )

Incrementa il contatore degli interrupt per un particolare device.

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	---------------	---

## Incremento del numero totale di interrupt

Dopo aver settato il flag, viene incrementato il valore degli interrupt totali. Anche questa operazione viene effettuata in mutua esclusione.

4.8.3.14 int myGPIOK\_Init ( [myGPIOK\\_t](#) \* *myGPIOK\_device*, struct module \* *owner*, struct platform\_device \* *op*, struct class \* *class*, const char \* *driver\_name*, const char \* *device\_name*, uint32\_t *serial*, struct file\_operations \* *f\_ops*, irq\_handler\_t *irq\_handler*, uint32\_t *irq\_mask* )

Inizializza una struttura [myGPIOK\\_t](#) e configura il device corrispondente.

## Parametri

in	<i>myGPIOK_device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
in	<i>owner</i>	puntatore a struttura struct module, proprietario del device (THIS_MODULE)
in	<i>op</i>	puntatore a struct platform_device, costituito dal parametro "op" con cui viene invocata probe() o la remove()
in	<i>class</i>	puntatore a struct class, classe del device, deve essere stata precedentemente creata con class_create()
in	<i>driver_name</i>	nome del driver
in	<i>device_name</i>	nome del device
in	<i>serial</i>	numero seriale del device
in	<i>f_ops</i>	puntatore a struttura struct file_operations, specifica le funzioni che agiscono sul device
in	<i>irq_handler</i>	puntatore irq_handler_t alla funzione che gestirà gli interrupt generati dal device
in	<i>irq_mask</i>	maschera delle interruzioni del device

## Valori di ritorno

0	se non si è verificato nessun errore
---	--------------------------------------

## Major-number e Minor-number

Ai device drivers sono associati un major-number ed un minor-number. Il major-number viene usato dal kernel per identificare il driver corretto corrispondente ad uno specifico device, quando si effettuano operazioni su di esso. Il ruolo del minor number dipende dal device e viene gestito internamente dal driver. Questo driver, così come molti altri, usa il Major ed il minor number per distinguere le diverse istanze di device myGPIO che usano il device-driver myGPIOK. La registrazione di un device driver può essere effettuata chiamando **alloc\_chrdev\_region()**, la quale alloca un char-device numbers. Il major number viene scelto dinamicamente e restituito dalla funzione attraverso il parametro dev. La funzione restituisce un valore negativo nel caso in cui si verificano errori, 0 altrimenti.

```
1 int alloc_chrdev_region (dev_t * dev, unsigned baseminor, unsigned count, const char *name);
```

- dev: major e minor number

- baseminor: primo dei minor number richiesti
- count: numero di minornumber richiesti
- name: nome del device

## Operatori

Essendo un device "visto" come un file, ogni device driver deve implementare tutte le system-call previste per l'interfacciamento con un file. La corrispondenza tra la system-call e la funzione fornita dal driver viene stabilita attraverso la struttura `file_operations`. La struttura dati `file_operations`, definita in `<linux/fs.h>` mantiene puntatori a funzioni definite dal driver che consentono di definire il comportamento degli operatori che agiscono su un file.

```
1 static struct file_operations myGPIO_fops = {
2     .owner      = THIS_MODULE,
3     .llseek     = driver_seek,
4     .read       = driver_read,
5     .write      = driver_write,
6     .poll       = driver_poll,
7     .open       = driver_open,
8     .release    = driver_release
9 };
```

Ogni campo della struttura deve puntare ad una funzione del driver che implementa uno specifico "operatore" su file, oppure impostata a NULL se l'operatore non è supportato. L'esatto comportamento del kernel, quando uno dei puntatori è NULL, varia da funzione a funzione. La lista seguente introduce tutti gli operatori che un'applicazione può invocare su un device. La lista è stata mantenuta snella, includendo solo i campi strettamente necessari.

- *struct module \*owner* :  
il primo campo della struttura non è un operatore, ma un puntatore al modulo che "possiede" la struttura. Il campo ha lo scopo di evitare che il modulo venga rimosso dal kernel quando uno degli operatori è in uso. Viene inizializzato usando la macro `THIS_MODULE`, definita in `<linux/module.h>`.
- *loff\_t (\*llseek) (struct file \*, loff\_t, int)* : il campo `llseek` è usato per cambiare la posizione della "testina" di lettura/ scrittura in un file. La funzione restituisce la nuova posizione della testina. `loff_t` è un intero a 64 bit (anche su architetture a 32 bit). Eventuali errori vengono segnalati con un valore di ritorno negativo. Se questo campo è posto a NULL, eventuali chiamate a `seek` modificheranno la posizione della testina in un modo imprevedibile.
- *ssize\_t (\*read) (struct file \*, char \_\_user \*, size\_t, loff\_t \*)* :  
usata per leggere dati dal device. Se lasciato a NULL, ogni chiamata a `read` fallirà e non sarà possibile leggere dal device. La funzione restituisce il numero di byte letti con successo ma, nel caso si verifichi un errore, restituisce un numero intero negativo.
- *ssize\_t (\*write) (struct file \*, const char \_\_user \*, size\_t, loff\_t \*)* :  
invia dati al device. Se NULL ogni chiamata alla system-call `write` causerà un errore. Il valore di ritorno, se non negativo, rappresenta il numero di byte correttamente scritti.
- *unsigned int (\*poll) (struct file \*, struct poll\_table\_struct \*)* :  
questo metodo è il back-end di tre diverse system-calls: `poll`, `epoll` e `select`, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno. La funzione dovrebbe restituire una maschera che indichi se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili. Se viene lasciata NULL si intende che le operazioni di lettura/scrittura sul device siano sempre non-bloccanti.
- *int (\*open) (struct inode \*, struct file \*)* :  
Anche se, di solito, è la prima operazione che si effettua su un file, non è strettamente necessaria la sua implementazione. Se lasciata NULL, l'apertura del device andrà comunque a buon fine, ma al driver non verrà inviata alcuna notifica.
- *int (\*release) (struct inode \*, struct file \*)* :  
questo operatore viene invocato quando il file viene rilasciato. Come `open`, può essere lasciato NULL.

L'inizializzazione di un device a caratteri passa anche attraverso la definizione di questo tipo di operatori. Essi possono essere impostati attraverso l'uso della funzione

```
1 void cdev_init (struct cdev *cdev, const struct file_operations *fops);
```

la quale prende, come parametri

- cdev: puntatore a struttura cdev da inizializzare;
- fops: puntatore a struttura file\_operation con cui inizializzare il device.

#### Creazione del device

Il passo successivo è la registrazione del device e la sua aggiunta al filesystem. Tale operazione può essere effettuata chiamando

```
1 struct device * device_create( struct class *class, struct device *parent, dev_t devt, const char *fmt,
    ...)
```

- class: puntatore alla struttura class alla quale il device deve essere registrato
- parent: puntatore ad eventuale device parent
- devt: tmajor number
- fmt: nome del device.

La funzione può essere usata solo sulla classe dei device a caratteri. Crea un device all'interno del filesystem, associandogli il major number preventivamente inizializzato. La funzione restituisce il puntatore alla struttura device creata all'interno del filesystem. Si noti che il puntatore alla struttura classes DEVE essere stato precedentemente creato attraverso una chiamata alla funzione *class\_create()*.

#### Aggiunta del device

Il driver, a questo punto, è pronto per essere aggiunto. È possibile aggiungere il driver usando

```
1 int cdev_add (struct cdev *p, dev_t dev, unsigned count);
```

La quale accetta come parametri

- p: puntatore a struttura cdev structure per il device
- dev: device number (precedentemente inizializzato usando la funzione *alloc\_chrdev\_region()*)
- count: numero di minor-numbers richiesti per il device

La funzione restituisce un numero negativo in caso di errore.

#### Accedere al segmento di memoria a cui la periferica è mappata

Un driver, tipicamente, prende possesso del segmento di memoria cui è mappato il device con la funzione di probe. Il problema è che il device è mappato ad un indirizzo di memoria fisico ed il Kernel, così come qualsiasi altro programma, lavora su indirizzi di memoria virtuali. La funzione

```
1 int of_address_to_resource(struct device_node *node, int index, struct resource *r);
```

popola una struttura resource con l'indirizzo di memoria cui è mappato il device usando le informazioni contenute all'interno del device tree. Ad esempio, se il device tree contiene

```
1 reg = <0x41200000 0x10000>;
```

significa che l'indirizzo fisico associato al device è l'indirizzo 0x41200000, che al device sono riservati 0x10000 bytes. *of\_address\_to\_resource()* setterà *res.start* = 0x41200000 e *res.end* = 0x4120ffff.

### Allocazione della memoria del device

Le regioni di memoria per di I/O vanno allocate prima di poter essere usate.

```
1 struct resource *request_mem_region(unsigned long start, unsigned long len, char *name);
```

Questa funzione alloca una regione di memoria di len byte a partire da start restituendone l'indirizzo, mentre nel caso in cui si verifichi un errore viene restituito NULL. La funzione viene chiamata per ottenere l'accesso esclusivo della regione di memoria, per evitare che driver diversi tentino di accedere allo stesso spazio di memoria.

### Remapping

L'allocazione dello spazio di memoria non è l'unico step da eseguire prima che tale memoria possa essere usata. È necessario fare in modo che sia resa accessibile al kernel attraverso un mapping, usando la funzione.

```
1 void *ioremap(unsigned long phys_addr, unsigned long size);
```

### Registrazione di un interrupt-handler

Il modulo deve registrare un handler per gli interrupt. L'handler deve essere compatibile con il tipo puntatore a funzione irq\_handler\_t, così definito.

```
1 struct irqreturn_t (*irq_handler_t)(int irq, struct pt_regs * regs);
```

Il modulo definisce la funzione [myGPIOK\\_irq\\_handler\(\)](#). L'handler può essere registrato usando

```
1 int request_irq(    unsigned int irqNumber,
2                    irqreturn_t (*handler)(int, void *, struct pt_regs *),
3                    unsigned long irqflags,
4                    const char *devname,
5                    void *dev_id);
```

IL parametro irqNumber può essere determinato automaticamente usando la funzione

```
1 unsigned int irq_of_parse_and_map(struct device_node *node, int index);
```

La funzione irq\_of\_parse\_and\_map() effettua un looks-up nella specifica degli interrupt all'interno del device tree e restituisce un irq number così come lo aspetta request\_irq() (cioè compaci con l'enumerazione in /proc/interrupts). Il secondo argomento della funzione è, tipicamente, zero, ad indicare che, all'interno del device tree, verrà preso in considerazione soltanto il primo degli interrupt specificate. Il device tree, nella sezione dedicata al gpio, reca

```
1 interrupts = <0 29 4>;
```

Il primo numero (0) è un flag che indica se l'interrupt sia connesso ad una line SPI (shared peripheral interrupt). Un valore diverso da zero indica che la linea è SPI. Il secondo numero si riferisce all'interrupt number. Per farla breve, quando si definisce la parte hardware, in questo specifico esempio il device GPIO è connesso alla linea 61 del GIC. Sottraendo 32 si ottiene 29. Il terzo numero si riferisce alla tipologia dell'interrupt. Sono possibili tre valori:

- 0 : power-up default
- 1 : rising-edge
- 4 : a livelli, active alto

### Inizializzazione della wait-queue per la system-call read() e poll()

In linux una wait queue viene implementata da una struttura dati wait\_queue\_head\_t, definita in <linux/wait.h>. Il driver in questione prevede due wait-queue differenti: una per la system-call read() ed una per la system-call poll(). Entrambe le code vengono inizializzate dalla funzione [myGPIOK\\_probe\(\)](#).

```
1 init_waitqueue_head(&my_queue);
```

Si veda la documentazione della funzione [myGPIOK\\_read\(\)](#) per dettagli ulteriori.

### Inizializzazione degli spinlock

I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spinlock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`. L'inizializzazione di uno spinlock avviene usando la funzione

```
1 void spin_lock_init(spinlock_t *lock);
```

### Abilitazione degli interrupt del device

A seconda del valore `CFLAGS_myGPIOK.o` (si veda il Makefile a corredo), vengono abilitati gli interrupt della periferica. Se si tratta del GPIO Xilinx vengono abilitati gli interrupt globali e gli interrupt sul canale due. Se si tratta del device GPIO custom, essendo esso parecchio più semplice, è necessario abilitare solo gli interrupt globali.

**4.8.3.15** `static irqreturn_t myGPIOK_irq_handler ( int irq, struct pt_regs * regs )` `[static]`

Interrupt-handler.

Parametri

<i>irq</i>	
<i>regs</i>	

Valori di ritorno

<i>IRQ_HANDLED</i>	dopo aver servito l'interruzione
--------------------	----------------------------------

Gestisce il manifestarsi di un evento interrompente proveniente dalla periferica. Viene registrata dalla funzione [myGPIOK\\_probe\(\)](#) affinché venga richiamata al manifestarsi di un interrupt sulla linea cui è connesso il device

### Disabilitazione delle interruzioni della periferica

Prima di servire l'interruzione, gli interrupt della periferica vengono disabilitati. Se si tratta di un GPIO Xilinx, vengono disabilitati sia gli interrupt globali che quelli generati dal secondo canale. Se, invece, si tratta di un device GPIO custom myGPIO, vengono disabilitati solo gli interrupt globali.

### Setting del valore del flag "interrupt occurred"

Dopo aver disabilitato gli interrupt della periferica, occorre settare in modo appropriato il flag "interrupt occurred", in modo che i processi in attesa possano essere risvegliati in modo sicuro. Per prevenire race condition, tale operazione viene effettuata mutua esclusione.

### Incremento del numero totale di interrupt

Dopo aver settato il flag, viene incrementato il valore degli interrupt totali. Anche questa operazione viene effettuata in mutua esclusione.

### Wakeup dei processi sleeping

La ISR deve chiamare esplicitamente `wakeup()` per risvegliare i processi messi in sleeping in attesa che un particolare evento si manifestasse. Se due processi vengono risvegliati contemporaneamente potrebbero originarsi race-condition.

**4.8.3.16** `static loff_t myGPIOK_llseek ( struct file * file_ptr, loff_t off, int whence )` `[static]`

Implementa le system-call `lseek()` e `llseek()`.

**Avvertimento**

L'implementazione di read() e write() non sposta la testina di lettura/scrittura!

**Parametri**

in, out	file_ptr	
in	off	
in	whence	

**Restituisce**

Nuova posizione della "testina" di lettura/scrittura

**4.8.3.17 static int myGPIOK\_open ( struct inode \* inode, struct file \* file\_ptr ) [static]**

Invocata all'apertura del file corrispondente al device.

**Parametri**

in	inode	
in, out	file	

**Valori di ritorno**

0	se non si verifica nessun errore
---	----------------------------------

**Il metodo open()**

Il metodo open di un device driver viene fornito per effettuare ogni inizializzazione necessaria ad operazioni successive. Effettua le seguenti operazioni:

- verifica che non si siano manifestati errori;
- inizializza il device
- aggiorna il puntatore f\_op, se necessario;
- alloca e popola ogni struttura dati necessaria, ponendola successivamente nel campo private\_data della struttura dati file.

In primo luogo è necessario identificare il device che sta per essere aperto. Tenendo presente che il prototipo di qualunque metodo open è

```
1 int (*open)(struct inode *inode, struct file *filp);
```

**Identificare il particolare device associato al file**

Il parametro inode contiene tutte le informazioni necessarie all'interno del campo i\_cdev, il quale contiene la struttura cdev inizializzata precedentemente dalla funzione di probe(). Il problema è che non abbiamo bisogno della sola struttura cdev, ma della struttura che la contiene, in questo caso della struttura [myGPIOK\\_t](#). Fortunatamente i programmatori del kernel hanno reso la vita semplice agli altri, predisponendo la macro container\_if() definita in <linux/kernel.h>.

```
1 container_of(pointer, container_type, container_field);
```

La macro prende in ingresso un puntatore ad un campo di tipo container\_field, di una struttura container\_type, restituendo il puntatore alla struttura che la contiene.

Disponendo, in questo caso, della struttura [myGPIOK\\_list\\_t](#), si fa uso di quest'ultima, e delle sue funzioni membro, per risalire al device a partire dal suo major e minor number. Il puntatore al particolare device [myGPIOK\\_t](#) sarà conservato all'interno del campo private\_data della struttura file.

#### 4.8.3.18 unsigned myGPIOK\_PendingPinInterrupt ( myGPIOK\_t \* *device* )

Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

**Restituisce**

maschera che riporta i pin per i quali gli interrupt non sono stati ancora serviti;

**4.8.3.19 void myGPIOK\_PinInterruptAck ( myGPIOK\_t \* device, unsigned mask )**

Invia al device notifica di servizio di un interrupt;.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	mask maschera di selezione degli interrupt da notificare; quelli non selezionati non vengono notificati;

**4.8.3.20 void myGPIOK\_PinInterruptDisable ( myGPIOK\_t \* device, unsigned mask )**

Disabilita gli interrupt per i singoli pin del device.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	maschera di selezione degli interrupt da disabilitare; quelli non selezionati non vengono disabilitati;

**4.8.3.21 void myGPIOK\_PinInterruptEnable ( myGPIOK\_t \* device, unsigned mask )**

Abilita gli interrupt per i singoli pin del device.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	maschera di selezione degli interrupt da abilitare; quelli non selezionati non vengono abilitati;

**4.8.3.22 static unsigned int myGPIOK\_poll ( struct file \* file\_ptr, struct poll\_table\_struct \* wait ) [static]**

Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.

**Parametri**

<i>in, out</i>	<i>file</i>	
<i>in, out</i>	<i>wait</i>	

**Restituisce**

restituisce una maschera di bit che indica se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili.

Questo metodo è il back-end di tre diverse system-calls: poll, epoll e select, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno.



4.8.3.23 `static int myGPIOK_probe ( struct platform_device * op ) [static]`

Viene chiamata quando il modulo viene inserito.

**Parametri**

<i>in, out</i>	<i>op</i>	
----------------	-----------	--

**Valori di ritorno**

<i>0</i>	nel caso in cui non si sia verificato nessun errore;
<i>-ENOMEM</i>	nel caso in cui non sia possibile allocare memoria;
<i>&lt;0</i>	per altri errori

**Inizializzazione del driver**

Il device myGPIO viene gestito come un character-device, ossia un device su cui è possibile leggere e/o scrivere byte. Il kernel usa, internamente, una struttura `cdev` per rappresentare i device a caratteri. Prima che il kernel invochi le funzioni definite dal driver per il device, bisogna allocare e registrare uno, o più, oggetti `cdev`. Per farlo è necessario includere `<linux/cdev.h>`, che definisce tale struttura e le relative funzioni.

**Device Class**

Ai device-drivers viene associata una classe ed un device-name. Per creare ed associare una classe ad un device driver si può usare la seguente.

```
1 struct class * class_create(struct module * owner, const char * name);
```

- `owner`: puntatore al modulo che "possiede" la classe, `THIS_MODULE`
- `name`: puntatore alla stringa identificativa (il nome) del device driver, `DRIVER_NAME`

**4.8.3.24** `static ssize_t myGPIOK_read ( struct file * file_ptr, char * buf, size_t count, loff_t * off )` `[static]`

Legge dati dal device.

**Parametri**

<i>in</i>	<i>file</i>	
<i>out</i>	<i>buf</i>	
<i>in</i>	<i>count</i>	
<i>in</i>	<i>off</i>	

**Avvertimento**

L'offset viene diviso per quattro prima di essere aggiunto all'indirizzo base del device.

**Restituisce**

restituisce un valore negativo nel caso in cui si sia verificato un errore. Un valore maggiore o uguale a zero indica il numero di byte scritti con successo.

**Operazioni di lettura e scrittura**

I metodi `read()` e `write()` effettuano operazioni simili, ossia copiare dati da/verso il device. Il loro prototipo è molto simile.

```
1 ssize_t read(struct file *filp, char __user *buff, size_t count, loff_t *offp);
2 ssize_t write(struct file *filp, const char __user *buff, size_t count, loff_t *offp);
```

Per entrambi i metodi `filep` è il puntatore al file che rappresenta il device, `count` è la dimensione dei dati da trasferire, `buff` è il puntatore al buffer contenente i dati (da scrivere per la `write()` o letti per la `read()`). Infine `offp` è il puntatore ad un oggetto "long offset type" che indica la posizione alla quale si sta effettuando l'accesso.

### I/O bloccante

Nel paragrafo precedente è stato ignorato un problema importante: come comportarsi quando il driver non è in grado di servire immediatamente una richiesta? Una chiamata a `read()` potrebbe arrivare quando i dati non sono disponibili, ma potrebbero esserlo in futuro, oppure, una chiamata a `write()` potrebbe avvenire quando il device non è in grado di accettare altri dati (perché il suo buffer di ingresso potrebbe essere pieno). Il processo chiamante è totalmente all'oscuro di queste dinamiche, anzi potrebbe non avere la minima conoscenza delle dinamiche interne del device: chiama le funzioni `read()` o `write()` e si aspetta che facciano ciò che devono fare, per cui, nell'impossibilità di servire la richiesta, il driver bloccare il processo e metterlo tra i processi "sleeping", fin quando la richiesta non può essere servita. Il codice (la ISR) che dovrà risvegliare il processo quando potrà servire la sua richiesta, deve essere a conoscenza dell'evento "risvegliante" e deve essere in grado di "trovare" i processi in attesa per quel particolare evento. Per questo motivo, tutti i processi in attesa di un particolare evento vengono posti all'interno della stessa wait queue. Il codice della ISR deve effettuare una chiamata a `wakeup()` per risvegliare i processi in attesa di un evento quando questo si è manifestato. Si veda la documentazione della funzione `myGPIOK_irq_handler()` per dettagli ulteriori.

### I/O non-bloccante

Esistono casi in cui il processo chiamante non vuole essere bloccato in attesa di un evento. Questa evenienza viene esplicitamente indicata attraverso il flag `O_NONBLOCK` flag in `filp->f_flags`. Il flag viene definito in `<linux/fcntl.h>` il quale è incluso in `<linux/fs.h>`.

#### Porre un processo nello stato sleeping

Quando un processo viene messo nello stato sleep, lo si fa aspettandosi che una condizione diventi vera in futuro. Al risveglio, però, non c'è nessuna garanzia che quella particolare condizione sia ancora vera, per cui essa va nuovamente testata. Il modo più semplice per porre un processo nello stato sleeping è chiamare la macro `wait_event()`, o una delle sue varianti: essa combina la gestione della messa in sleeping del processo ed il check della condizione che il processo si aspetta diventi vera.

```
1 wait_event_interruptible(queue, condition);
```

Il parametro `queue` è la coda di attesa mentre `condition` è la condizione che, valutata true, causa la messa in sleep del processo. La condizione viene valutata sia prima che il processo sia messo in sleeping che al suo risveglio. Lo sleep in cui il processo viene messo chiamando `wait_event_interruptible()` può essere interrotto anche da un segnale, per cui la macro restituisce un intero che, se diverso da zero, indica che il processo è stato risvegliato da un segnale.

La condizione sulla quale i processi vengono bloccati riguarda il flag "interrupt occurred". Fin quando questo flag, posto in and con la maschera `MYGPIOK_SREAD`, è zero, il processo deve restare bloccato, per cui i processi che effettuano `read()` bloccante restano bloccati finché `int_occurred & MYGPIOK_SREAD == 0`. Quando tale uguaglianza non sarà più valida, perché il valore di `int_occurred` viene settato dalla funzione `myGPIOK_irq_handler()`, allora il processo verrà risvegliato.

#### Reset del flag "interrupt occurred" per read() bloccanti

Nel momento in cui il processo viene risvegliato e la condizione della quale era in attesa è tale che esso può continuare la sua esecuzione, è necessario resettare tale flag. Questa operazione va effettuata per prevenire race-condition dovute al risveglio di più processi in attesa del manifestarsi dello stesso evento. Il reset del flag va, pertanto, effettuato in mutua esclusione.

#### Accesso ai registri del device

Si potrebbe sentire la tentazione di usare il puntatore restituito da `ioremap()` dereferenziandolo per accedere alla memoria. Questo modo di procedere non è portabile ed è prone ad errori. Il modo corretto di accedere alla memoria è attraverso l'uso delle funzioni per il memory-mapped I/O, definite in `<asm/io.h>`. Per leggere dalla memoria vengono usate le seguenti:

```
1 unsigned int ioread8(void *addr);
2 unsigned int ioread16(void *addr);
3 unsigned int ioread32(void *addr);
```

`addr` è l'indirizzo di memoria virtuale del device, ottenuto mediante chiamata a `ioremap()`, a cui viene, eventualmente, aggiunto un offset. Il valore restituito dalle funzioni è quello letto dalla particolare locazione di memoria a cui viene effettuato accesso. Per scrivere nella memoria vengono usate le seguenti:

```
1 void iowrite8(u8 value, void *addr);
2 void iowrite16(u16 value, void *addr);
3 void iowrite32(u32 value, void *addr);
```

### Accesso alla memoria userspace

`Buff` è un puntatore appartenente allo spazio di indirizzamento del programma user-space che utilizza il modulo kernel. Il modulo, quindi, non può accedere direttamente ad esso, dereferenziandolo, per diverse ragioni, tra le quali:

- a seconda dell'architettura sulla quale il driver è in esecuzione e di come il kernel è stato configurato, il puntatore userspace potrebbe non essere valido mentre il modulo kernel viene eseguito;
- la memoria user-space è paginata e potrebbe non essere presente in RAM quando la system-call viene effettuata, per cui dereferenziando il puntatore potrebbe originarsi un page-fault con conseguente terminazione del processo che ha effettuato la system-call;
- il puntatore in questione potrebbe essere stato fornito da un programma user-space buggato o malizioso, motivo per cui dereferenziandolo verrebbe a crearsi un punto di accesso attraverso il quale il programma userspace può modificare la memoria senza costrizioni.

Ovviamente il driver deve essere in grado di poter accedere al buffer userspace, per cui tale accesso va fatto solo ed esclusivamente attraverso delle funzioni fornite dal kernel stesso, e definite in `<asm/uaccess.h>`

```
1 unsigned long copy_to_user(void __user *to, const void *from, unsigned long count);
2 unsigned long copy_from_user(void *to, const void __user *from, unsigned long count);
```

Queste due funzioni non si limitano a copiare dati da/verso userspace: verificano, infatti, anche che il puntatore al buffer userspace sia valido. Se il puntatore non risultasse valido la copia non viene effettuata. Sia il metodo `read()` che il metodo `write()` restituiscono un valore negativo nel caso in cui si sia verificato un errore. Un valore maggiore o uguale a zero indica il numero di byte trasferiti con successo.

### Piccola nota sull'endianess

Il processore Zynq è little endian. Per questo motivo è possibile convertire `char*` in `uint32_t*` mediante un semplice casting, senza invertire manualmente l'ordine dei byte.

### Debouncing

Sebbene normalmente non necessario, in questo caso si è preferito inserire un hot-loop, in modo da attendere che il device di input venga riportato allo stato di riposo prima di continuare l'esecuzione della funzione. Questo piccolo espediente serve a fare in modo che, nel nostro caso, non vengano generate interruzioni spurie. Il ciclo va rimosso in qualsiasi applicazione che non riguardi la pressione di un tasto.

### Ack degli interrupt della periferica

Viene inviato l'Ack alla periferica, per segnalargli che l'interrupt è stato servito, solo dopo che la lettura sia stata effettuata.

### Abilitazione degli interrupt della periferica

Dopo aver inviato notifica di servizio dell'interruzione al device, vengono nuovamente abilitati gli interrupt.

**4.8.3.25** `static int myGPIOK_release ( struct inode * inode, struct file * file_ptr ) [static]`

Invocata alla chiusura del file corrispondente al device.

## Parametri

in	<i>inode</i>	
in	<i>file</i>	

## Valori di ritorno

0	se non si verifica nessun errore
---	----------------------------------

4.8.3.26 static int myGPIOK\_remove ( struct platform\_device \* *op* ) [static]

Viene chiamata automaticamente alla rimozione del mosulo.

## Parametri

in, out	<i>op</i>	
---------	-----------	--

## Valori di ritorno

0	se non si verifica nessun errore
---	----------------------------------

Dealloca tutta la memoria utilizzata dal driver, de-inizializzando il device e disattivando gli interrupt per il device, effettuando tutte le operazioni inverse della funzione [myGPIOK\\_probe\(\)](#).

4.8.3.27 void myGPIOK\_ResetCanRead ( myGPIOK\_t \* *device* )

Reset del flag "interrupt occurred" (canRead)

## Parametri

in	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	---------------	---

## Reset del flag "interrupt occurred" per read() bloccanti

Nel momento in cui il processo viene risvegliato e la condizione della quale era in attesa è tale che esso può continuare la sua esecuzione, è necessario resettare tale flag. Questa operazione va effettuata per prevenire race-condition dovute al risveglio di più processi in attesa del manifestarsi dello stesso evento. Il reset del flag va, pertanto, effettuato in mutua esclusione.

I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spinlock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`. Esistono diversi modi per acquisire uno spinlock. Nel seguito viene usata la funzione

```
1 void spin_lock(spinlock_t *lock);
```

per rilasciare uno spinlock, invece, verrà usata

```
1 void spin_unlock(spinlock_t *lock);
```

4.8.3.28 void myGPIOK\_SetCanRead ( myGPIOK\_t \* *device* )

Set del flag "interrupt occurred" (canRead)

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

**Setting del valore del flag "interrupt occurred"**

Dopo aver disabilitato gli interrupt della periferica, occorre settare in modo appropriato il flag "interrupt occurred", in modo che i processi in attesa possano essere risvegliati in modo sicuro. Per prevenire race condition, tale operazione viene effettuata mutua esclusione. I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spinlock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in <linux/spinlock.h>. Esistono diversi modi per acquisire uno spinlock. Nel seguito viene usata la funzione

```
1 void spin_lock_irqsave(spinlock_t *lock, unsigned long flags);
```

la quale disabilita gli interrupt sul processore locale prima di acquisire lo spinlock, per poi ripristinarlo quando lo spinlock viene rilasciato, usando

```
1 void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags);
```

**4.8.3.29 void myGPIOK\_TestCanReadAndSleep ( myGPIOK\_t \* device )**

Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

**Porre un processo nello stato sleeping**

Quando un processo viene messo nello stato sleep, lo si fa aspettandosi che una condizione diventi vera in futuro. Al risveglio, però, non c'è nessuna garanzia che quella particolare condizione sia ancora vera, per cui essa va nuovamente testata. Il modo più semplice per porre un processo nello stato sleeping è chiamare la macro `wait_event()`, o una delle sue varianti: essa combina la gestione della messa in sleeping del processo ed il check della condizione che il processo si aspetta diventi vera.

```
1 wait_event_interruptible(queue, condition);
```

Il parametro `queue` è la coda di attesa mentre `condition` è la condizione che, valutata true, causa la messa in sleep del processo. La condizione viene valutata sia prima che il processo sia messo in sleeping che al suo risveglio. Lo sleep in cui il processo viene messo chiamando `wait_event_interruptible()` può essere interrotto anche da un segnale, per cui la macro restituisce un intero che, se diverso da zero, indica che il processo è stato risvegliato da un segnale.

La condizione sulla quale i processi vengono bloccati riguarda il flag "interrupt occurred". Fin quando questo flag, posto in and con la maschera `MYGPIOK_SREAD`, è zero, il processo deve restare bloccato, per cui i processi che effettuano `read()` bloccante restano bloccati finché `int_occurred & MYGPIO_SREAD == 0`. Quando tale uguaglianza non sarà più valida, perché il valore di `int_occurred` viene settato dalla funzione [myGPIOK\\_irq\\_handler\(\)](#), allora il processo verrà risvegliato.

**4.8.3.30 void myGPIOK\_WakeUp ( myGPIOK\_t \* device )**

Risveglia i process in attesa sulle code di read e poll.

**Parametri**

<i>in</i>	<i>device</i>	puntatore a struttura <code>myGPIOK_t</code> , che si riferisce al device su cui operare
-----------	---------------	--

**Wakeup dei processi sleeping**

La ISR deve chiamare esplicitamente `wakeup()` per risvegliare i processi messi in sleeping in attesa che un particolare evento si manifestasse. La funzione

```
1 void wake_up_interruptible(wait_queue_head_t *queue);
```

risveglia tutti i processi posti in una determinata coda (risvegliando solo quelli che, in precedenza, hanno effettuato una chiamata a `wait_event_interruptible()`). Se due processi vengono risvegliati contemporaneamente potrebbero originarsi race-condition.

**4.8.3.31** `static ssize_t myGPIOK_write ( struct file * file_ptr, const char * buf, size_t size, loff_t * off ) [static]`

Invia dati al device.

**Parametri**

<i>in</i>	<i>file</i>	
<i>in</i>	<i>buf</i>	
<i>in</i>	<i>size</i>	
<i>in</i>	<i>off</i>	

**Avvertimento**

L'offset viene diviso per quattro prima di essere aggiunto all'indirizzo base del device.

**Restituisce**

restituisce un valore negativo nel caso in cui si sia verificato un errore. Un valore maggiore o uguale a zero indica il numero di byte scritti con successo.

**Operazioni di lettura e scrittura**

I metodi `read()` e `write()` effettuano operazioni simili, ossia copiare dati da/verso il device. Il loro prototipo è molto simile.

```
1 ssize_t read(struct file *filp, char __user *buff, size_t count, loff_t *offp);
2 ssize_t write(struct file *filp, const char __user *buff, size_t count, loff_t *offp);
```

Per entrambi i metodi `filep` è il puntatore al file che rappresenta il device, `count` è la dimensione dei dati da trasferire, `buff` è il puntatore al buffer contenente i dati (da scrivere per la `write()` o letti per la `read()`). Infine `offp` è il puntatore ad un oggetto "long offset type" che indica la posizione alla quale si sta effettuando l'accesso.

**Accesso alla memoria userspace**

`Buff` è un puntatore appartenente allo spazio di indirizzamento del programma user-space che utilizza il modulo kernel. Il modulo, quindi, non può accedere direttamente ad esso, dereferenziandolo, per diverse ragioni, tra le quali:

- a seconda dell'architettura sulla quale il driver è in esecuzione e di come il kernel è stato configurato, il puntatore userspace potrebbe non essere valido mentre il modulo kernel viene eseguito;
- la memoria user-space è paginata e potrebbe non essere presente in RAM quando la system-call viene effettuata, per cui dereferenziando il puntatore potrebbe originarsi un page-fault con conseguente terminazione del processo che ha effettuato la system-call;

- il puntatore in questione potrebbe essere stato fornito da un programma user-space buggato o malizioso, motivo per cui dereferenziandolo verrebbe a crearsi un punto di accesso attraverso il quale il programma userspace può modificare la memoria senza costrizioni.

Ovviamente il driver deve essere in grado di poter accedere al buffer userspace, per cui tale accesso va fatto solo ed esclusivamente attraverso delle funzioni fornite dal kernel stesso, e definite in `<asm/uaccess.h>`

```
1 unsigned long copy_to_user(void __user *to, const void *from, unsigned long count);
2 unsigned long copy_from_user(void *to, const void __user *from, unsigned long count);
```

Queste due funzioni non si limitano a copiare dati da/verso userspace: verificano, infatti, anche che il puntatore al buffer userspace sia valido. Se il puntatore non risultasse valido la copia non viene effettuata. Sia il metodo `read()` che il metodo `write()` restituiscono un valore negativo nel caso in cui si sia verificato un errore. Un valore maggiore o uguale a zero indica il numero di byte trasferiti con successo.

#### Piccola nota sull'endianess

Il processore Zynq è little endian. Per questo motivo è possibile convertire `char*` in `uint32_t*` mediante un semplice casting, senza invertire manualmente l'ordine dei byte.

#### Accesso ai registri del device

Si potrebbe sentire la tentazione di usare il puntatore restituito da `ioremap()` dereferenziandolo per accedere alla memoria. Questo modo di procedere non è portabile ed è prone ad errori. Il modo corretto di accedere alla memoria è attraverso l'uso delle funzioni per il memory-mapped I/O, definite in `<asm/io.h>`.

Per leggere dalla memoria vengono usate le seguenti:

```
1 unsigned int ioread8(void *addr);
2 unsigned int ioread16(void *addr);
3 unsigned int ioread32(void *addr);
```

`addr` è l'indirizzo di memoria virtuale del device, ottenuto mediante chiamata a `ioremap()`, a cui viene, eventualmente, aggiunto un offset. Il valore restituito dalle funzioni è quello letto dalla particolare locazione di memoria a cui viene effettuato accesso.

Per scrivere nella memoria vengono usate le seguenti:

```
1 void iowrite8(u8 value, void *addr);
2 void iowrite16(u16 value, void *addr);
3 void iowrite32(u32 value, void *addr);
```

## 4.8.4 Documentazione delle variabili

### 4.8.4.1 `myGPIOK_list_t* device_list = NULL` [static]

Array di puntatori a struttura `myGPIOK_t`, contenente tutti i dati necessari al device driver.

### 4.8.4.2 `struct class* myGPIOK_class = NULL` [static]

Classe del device Ai device-drivers viene associata una classe ed un device-name. Per creare ed associare una classe ad un device driver si può usare la seguente.

```
1 struct class * class_create(struct module * owner, const char * name);
```

Parametri:

- `owner`: puntatore al modulo che "possiede" la classe, `THIS_MODULE`
- `name`: puntatore alla stringa identificativa (il nome) del device driver, `DRIVER_NAME`



## 4.8.4.3 struct platform\_driver myGPIOK\_driver [static]

## Valore iniziale:

```
= {
    .probe = myGPIOK_probe,
    .remove = myGPIOK_remove,
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = myGPIOK_match,
    },
}
```

Definisce quali funzioni probe() e remove() chiamare quando viene caricato un driver.

## 4.8.4.4 struct file\_operations myGPIOK\_fops [static]

## Valore iniziale:

```
= {
    .owner      = THIS_MODULE,
    .llseek     = myGPIOK_llseek,
    .read       = myGPIOK_read,
    .write      = myGPIOK_write,
    .poll       = myGPIOK_poll,
    .open       = myGPIOK_open,
    .release    = myGPIOK_release
}
```

mantiene puntatori a funzioni che definiscono il gli operatori che agiscono su un file/device.

Essendo un device "visto" come un file, ogni device driver deve implementare tutte le system-call previste per l'interfacciamento con un file. La corrispondenza tra la system-call e la funzione fornita dal driver viene stabilita attraverso tale struttura. La struttura dati file\_operations, definita in <linux/fs.h> mantiene puntatori a funzioni definite dal driver che consentono di definire il comportamento degli operatori che agiscono su un file.

La struttura dati file\_operations, definita in <linux/fs.h> mantiene puntatori a funzioni definite dal driver che consentono di definire il comportamento degli operatori che agiscono su un file.

```
1 static struct file_operations myGPIO_fops = {
2     .owner      = THIS_MODULE,
3     .llseek     = driver_seek,
4     .read       = driver_read,
5     .write      = driver_write,
6     .poll       = driver_poll,
7     .open       = driver_open,
8     .release    = driver_release
9 };
```

Ogni campo della struttura deve puntare ad una funzione del driver che implementa uno specifico "operatore" su file, oppure impostata a NULL se l'operatore non è supportato. L'esatto comportamento del kernel, quando uno dei puntatori è NULL, varia da funzione a funzione. La lista seguente introduce tutti gli operatori che un'applicazione può invocare su un device. La lista è stata mantenuta snella, includendo solo i campi strettamente necessari.

- *struct module \*owner* :  
il primo campo della struttura non è un operatore, ma un puntatore al modulo che "possiede" la struttura. Il campo ha lo scopo di evitare che il modulo venga rimosso dal kernel quando uno degli operatori è in uso. Viene inizializzato usando la macro THIS\_MODULE, definita in <linux/module.h>.
- *loff\_t (\*llseek) (struct file \*, loff\_t, int)* : il campo llseek è usato per cambiare la posizione della "testina" di lettura/ scrittura in un file. La funzione restituisce la nuova posizione della testina. loff\_t è un intero a 64 bit (anche su architetture a 32 bit). Eventuali errori vengono segnalati con un valore di ritorno negativo. Se questo campo è posto a NULL, eventuali chiamate a seek modificheranno la posizione della testina in un modo imprevedibile.

- *ssize\_t (\*read) (struct file \*, char \_\_user \*, size\_t, loff\_t \*)* :  
usata per leggere dati dal device. Se lasciato a NULL, ogni chiamata a read fallirà e non sarà possibile leggere dal device. La funzione restituisce il numero di byte letti con successo ma, nel caso si verifichi un errore, restituisce un numero intero negativo.
- *ssize\_t (\*write) (struct file \*, const char \_\_user \*, size\_t, loff\_t \*)* :  
invia dati al device. Se NULL ogni chiamata alla system-call write causerà un errore. Il valore di ritorno, se non negativo, rappresenta il numero di byte correttamente scritti.
- *unsigned int (\*poll) (struct file \*, struct poll\_table\_struct \*)* :  
questo metodo è il back-end di tre diverse system-calls: poll, epoll e select, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno. La funzione dovrebbe restituire una maschera che indichi se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili. Se viene lasciata NULL si intende che le operazioni di lettura/scrittura sul device siano sempre non-bloccanti.
- *int (\*open) (struct inode \*, struct file \*)* :  
Anche se, di solito, è la prima operazione che si effettua su un file, non è strettamente necessaria la sua implementazione. Se lasciata NULL, l'apertura del device andrà comunque a buon fine, ma al driver non verrà inviata alcuna notifica.
- *int (\*release) (struct inode \*, struct file \*)* :  
questo operatore viene invocato quando il file viene rilasciato. Come open, può essere lasciato NULL.

#### 4.8.4.5 struct of\_device\_id myGPIOK\_match[] [static]

##### Valore iniziale:

```
= {
    { .compatible = DRIVER_NAME },
    {} ,
}
```

Identifica il device all'interno del device tree.

Tutti i device-driver devono esporre un ID. A tempo di compilazione, il processo di build estrae queste informazioni dai driver per la preparazione di una tabella. Quando si "inserisce" il device, la tabella viene riferita dal kernel e, se viene trovata una entry corrispondente al driver per quel device, il driver viene caricato ed inizializzato.

## 4.9 DeviceList

Definisce la struttura dati `myGPIOK_list_t`, la quale mantiene un riferimento agli oggetti `myGPIOK_t` gestiti dal driver.

Diagramma di collaborazione per DeviceList:



### Strutture dati

- struct `myGPIOK_list_t`

*Struttura dati per la gestione degli oggetti `myGPIOK_t` gestiti dal driver.*

### Funzioni

- int `myGPIOK_list_Init` (`myGPIOK_list_t` \*list, uint32\_t list\_size)  
*Inizializza una struttura dati `myGPIOK_list_t`.*
- void `myGPIOK_list_Destroy` (`myGPIOK_list_t` \*list)  
*Dealloca gli oggetti internamente gestiti da un oggetto `myGPIOK_list_t`, liberando la memoria.*
- int `myGPIOK_list_add` (`myGPIOK_list_t` \*list, `myGPIOK_t` \*device)  
*Aggiunge un riferimento ad un oggetto `myGPIOK_t` alla lista.*
- `myGPIOK_t` \* `myGPIOK_list_find_by_op` (`myGPIOK_list_t` \*list, struct platform\_device \*op)  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- `myGPIOK_t` \* `myGPIOK_list_find_by_minor` (`myGPIOK_list_t` \*list, dev\_t dev)  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- `myGPIOK_t` \* `myGPIOK_list_find_irq_line` (`myGPIOK_list_t` \*list, int irq\_line)  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- uint32\_t `myGPIOK_list_device_count` (`myGPIOK_list_t` \*list)  
*Restituisce il numero di device correntemente inseriti nella lista.*

#### 4.9.1 Descrizione dettagliata

Definisce la struttura dati `myGPIOK_list_t`, la quale mantiene un riferimento agli oggetti `myGPIOK_t` gestiti dal driver.

#### 4.9.2 Documentazione delle funzioni

##### 4.9.2.1 int myGPIOK\_list\_add ( myGPIOK\_list\_t \* list, myGPIOK\_t \* device )

Aggiunge un riferimento ad un oggetto `myGPIOK_t` alla lista.

**Parametri**

<i>in</i>	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista a cui aggiungere l'oggetto
<i>in</i>	<i>device</i>	puntatore a <a href="#">myGPIOK_t</a> , oggetto da aggiungere alla lista

**Valori di ritorno**

<i>-1</i>	se lo spazio si è esaurito
<i>0</i>	se non si manifesta nessun errore

**Avvertimento**

La funzione si limita ad aggiungere l'oggetto [myGPIOK\\_t](#) alla lista, senza effettuare alcun tipo di controllo su di esso. Non viene verificato, ad esempio, se il device che si intende aggiungere sia effettivamente già presente in lista o se si tratti di un puntatore nullo.

**4.9.2.2 void myGPIOK\_list\_Destroy ( myGPIOK\_list\_t \* list )**

Dealloca gli oggetti internamente gestiti da un oggetto [myGPIOK\\_list\\_t](#), liberando la memoria.

**Parametri**

<i>in</i>	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista da distruggere
-----------	-------------	---

**4.9.2.3 uint32\_t myGPIOK\_list\_device\_count ( myGPIOK\_list\_t \* list )**

Restituisce il numero di device correntemente inseriti nella lista.

**Parametri**

<i>in</i>	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista di cui si intende conoscere il numero di oggetti <a href="#">myGPIOK_t</a> contenuti
-----------	-------------	---

**Restituisce**

numero di device correntemente inseriti nella lista

**4.9.2.4 myGPIOK\_t \* myGPIOK\_list\_find\_by\_minor ( myGPIOK\_list\_t \* list, dev\_t dev )**

Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.

**Parametri**

<i>in</i>	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista in cui effettuare la ricerca
<i>in</i>	<i>dev</i>	major/minor number associato al device, parametro con cui viene invocata la open() o la release()

**Restituisce**

indirizzo dell'oggetto [myGPIOK\\_t](#), se è presente nella lista, NULL altrimenti

**4.9.2.5 myGPIOK\_t \* myGPIOK\_list\_find\_by\_op ( myGPIOK\_list\_t \* list, struct platform\_device \* op )**

Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.

## Parametri

in	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista in cui effettuare la ricerca
in	<i>op</i>	puntatore a struct platform_device, argomento con cui viene invocata probe() o remove(), associato ad un oggetto <a href="#">myGPIOK_t</a>

## Restituisce

indirizzo dell'oggetto [myGPIOK\\_t](#), se è presente nella lista, NULL altrimenti

4.9.2.6 [myGPIOK\\_t](#) \* [myGPIOK\\_list\\_find\\_irq\\_line](#) ( [myGPIOK\\_list\\_t](#) \* *list*, int *irq\_line* )

Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.

## Parametri

in	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista in cui effettuare la ricerca
	<i>[in]</i>	irq_line linea di interruzione alla quale il device è connesso, parametro di invocazione della ISR

## Restituisce

indirizzo dell'oggetto [myGPIOK\\_t](#), se è presente nella lista, NULL altrimenti

4.9.2.7 int [myGPIOK\\_list\\_init](#) ( [myGPIOK\\_list\\_t](#) \* *list*, uint32\_t *list\_size* )

Inizializza una struttura dati [myGPIOK\\_list\\_t](#).

## Parametri

in	<i>list</i>	puntatore a <a href="#">myGPIOK_list_t</a> , lista da inizializzare
in	<i>list_size</i>	numero massimo di device che la struttura dati potrà gestire

## Valori di ritorno

<i>-ENOMEM</i>	nel caso in cui la struttura non possa essere allocata in memoria
<i>0</i>	se non si manifestano errori

## 4.10 MyGPIOK\_t

Definisce l'oggetto `myGPIOK_t`, che rappresenta un device myGPIO a livello kernel.

Diagramma di collaborazione per `MyGPIOK_t`:



### Strutture dati

- struct `myGPIOK_t`  
*Struttura per l'astrazione di un device myGPIO in kernel-mode.*

### Definizioni

- #define `myGPIOK_GIES_OFFSET` 0x0CU  
*Offset, rispetto all'indirizzo base, del registro "GIES".*
- #define `myGPIOK_PIE_OFFSET` 0x10U  
*Offset, rispetto all'indirizzo base, del registro "PIE".*
- #define `myGPIOK_IRQ_OFFSET` 0x14U  
*Offset, rispetto all'indirizzo base, del registro "IRQ".*
- #define `myGPIOK_IACK_OFFSET` 0x18U  
*Offset, rispetto all'indirizzo base, del registro "IACK".*

### Funzioni

- int `myGPIOK_Init` (`myGPIOK_t` \*myGPIOK\_device, struct module \*owner, struct platform\_device \*op, struct class \*class, const char \*driver\_name, const char \*device\_name, uint32\_t serial, struct file\_operations \*f\_ops, irq\_handler\_t irq\_handler, uint32\_t irq\_mask)  
*Inizializza una struttura `myGPIOK_t` e configura il device corrispondente.*
- void `myGPIOK_Destroy` (`myGPIOK_t` \*device)  
*Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.*
- void `myGPIOK_SetCanRead` (`myGPIOK_t` \*device)  
*Set del flag "interrupt occurred" (canRead)*
- void `myGPIOK_ResetCanRead` (`myGPIOK_t` \*device)  
*Reset del flag "interrupt occurred" (canRead)*
- void `myGPIOK_TestCanReadAndSleep` (`myGPIOK_t` \*device)  
*Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.*
- unsigned `myGPIOK_GetPollMask` (`myGPIOK_t` \*device, struct file \*file\_ptr, struct poll\_table\_struct \*wait)  
*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*
- void `myGPIOK_IncrementTotal` (`myGPIOK_t` \*device)  
*Incrementa il contatore degli interrupt per un particolare device.*
- void `myGPIOK_WakeUp` (`myGPIOK_t` \*device)  
*Risveglia i process in attesa sulle code di read e poll.*

- void \* [myGPIOK\\_GetDeviceAddress](#) ([myGPIOK\\_t](#) \*device)  
*Restituisce l'indirizzo virtuale di memoria cui è mappato un device.*
- void [myGPIOK\\_GlobalInterruptEnable](#) ([myGPIOK\\_t](#) \*myGPIOK\_device)  
*Abilita gli interrupt globali;.*
- void [myGPIOK\\_GlobalInterruptDisable](#) ([myGPIOK\\_t](#) \*myGPIOK\_device)  
*Disabilita gli interrupt globali;.*
- void [myGPIOK\\_PinInterruptEnable](#) ([myGPIOK\\_t](#) \*myGPIOK\_device, unsigned mask)  
*Abilita gli interrupt per i singoli pin del device.*
- void [myGPIOK\\_PinInterruptDisable](#) ([myGPIOK\\_t](#) \*myGPIOK\_device, unsigned mask)  
*Disabilita gli interrupt per i singoli pin del device.*
- unsigned [myGPIOK\\_PendingPinInterrupt](#) ([myGPIOK\\_t](#) \*myGPIOK\_device)  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- void [myGPIOK\\_PinInterruptAck](#) ([myGPIOK\\_t](#) \*myGPIOK\_device, unsigned mask)  
*Invia al device notifica di servizio di un interrupt;.*

#### 4.10.1 Descrizione dettagliata

Definisce l'oggetto [myGPIOK\\_t](#), che rappresenta un device myGPIO a livello kernel.

#### 4.10.2 Documentazione delle definizioni

##### 4.10.2.1 #define myGPIOK\_GIES\_OFFSET 0x0CU

Offset, rispetto all'indirizzo base, del registro "GIES".

##### 4.10.2.2 #define myGPIOK\_IACK\_OFFSET 0x18U

Offset, rispetto all'indirizzo base, del registro "IACK".

##### 4.10.2.3 #define myGPIOK\_IRQ\_OFFSET 0x14U

Offset, rispetto all'indirizzo base, del registro "IRQ".

##### 4.10.2.4 #define myGPIOK\_PIE\_OFFSET 0x10U

Offset, rispetto all'indirizzo base, del registro "PIE".

#### 4.10.3 Documentazione delle funzioni

##### 4.10.3.1 void myGPIOK\_Destroy ( [myGPIOK\\_t](#) \* device )

Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.

Parametri

in	device	puntatore a struttura <a href="#">myGPIOK_t</a> , specifica il particolare device su cui agire
----	--------	--

##### 4.10.3.2 void\* myGPIOK\_GetDeviceAddress ( [myGPIOK\\_t](#) \* device )

Restituisce l'indirizzo virtuale di memoria cui è mappato un device.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

4.10.3.3 unsigned myGPIOK\_GetPollMask ( myGPIOK\_t \* *device*, struct file \* *file\_ptr*, struct poll\_table\_struct \* *wait* )

Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in, out</i>	<i>file</i>	
<i>in, out</i>	<i>wait</i>	

## Restituisce

restituisce una maschera di bit che indica se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili.

Questo metodo è il back-end di tre diverse system-calls: poll, epoll e select, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno.

4.10.3.4 void myGPIOK\_GlobalInterruptDisable ( myGPIOK\_t \* *device* )

Disabilita gli interrupt globali;.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

4.10.3.5 void myGPIOK\_GlobalInterruptEnable ( myGPIOK\_t \* *device* )

Abilita gli interrupt globali;.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

4.10.3.6 void myGPIOK\_IncrementTotal ( myGPIOK\_t \* *device* )

Incrementa il contatore degli interrupt per un particolare device.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

## Incremento del numero totale di interrupt

Dopo aver settato il flag, viene incrementato il valore degli interrupt totali. Anche questa operazione viene effettuata in mutua esclusione.

4.10.3.7 int myGPIOK\_Init ( myGPIOK\_t \* *myGPIOK\_device*, struct module \* *owner*, struct platform\_device \* *op*, struct class \* *class*, const char \* *driver\_name*, const char \* *device\_name*, uint32\_t *serial*, struct file\_operations \* *f\_ops*, irq\_handler\_t *irq\_handler*, uint32\_t *irq\_mask* )

Inizializza una struttura [myGPIOK\\_t](#) e configura il device corrispondente.



## Parametri

in	<i>myGPIOK_↔ device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
in	<i>owner</i>	puntatore a struttura struct module, proprietario del device (THIS_MODULE)
in	<i>op</i>	puntatore a struct platform_device, costituito dal parametro "op" con cui viene invocata probe() o la remove()
in	<i>class</i>	puntatore a struct class, classe del device, deve essere stata precedentemente creata con class_create()
in	<i>driver_name</i>	nome del driver
in	<i>device_name</i>	nome del device
in	<i>serial</i>	numero seriale del device
in	<i>f_ops</i>	puntatore a struttura struct file_operations, specifica le funzioni che agiscono sul device
in	<i>irq_handler</i>	puntatore irq_handler_t alla funzione che gestirà gli interrupt generati dal device
in	<i>irq_mask</i>	maschera delle interruzioni del device

## Valori di ritorno

0	se non si è verificato nessun errore
---	--------------------------------------

## Major-number e Minor-number

Ai device drivers sono associati un major-number ed un minor-number. Il major-number viene usato dal kernel per identificare il driver corretto corrispondente ad uno specifico device, quando si effettuano operazioni su di esso. Il ruolo del minor number dipende dal device e viene gestito internamente dal driver. Questo driver, così come molti altri, usa il Major ed il minor number per distinguere le diverse istanze di device myGPIO che usano il device-driver myGPIOK. La registrazione di un device driver può essere effettuata chiamando **alloc\_chrdev\_region()**, la quale alloca un char-device numbers. Il major number viene scelto dinamicamente e restituito dalla funzione attraverso il parametro dev. La funzione restituisce un valore negativo nel caso in cui si verificano errori, 0 altrimenti.

```
1 int alloc_chrdev_region (dev_t * dev, unsigned baseminor, unsigned count, const char *name);
```

- dev: major e minor number
- baseminor: primo dei minor number richiesti
- count: numero di minornumber richiesti
- name: nome del device

## Operatori

Essendo un device "visto" come un file, ogni device driver deve implementare tutte le system-call previste per l'interfacciamento con un file. La corrispondenza tra la system-call e la funzione fornita dal driver viene stabilita attraverso la struttura file\_operations. La struttura dati file\_operations, definita in <linux/fs.h> mantiene puntatori a funzioni definite dal driver che consentono di definire il comportamento degli operatori che agiscono su un file.

```
1 static struct file_operations myGPIO_fops = {
2     .owner      = THIS_MODULE,
3     .llseek     = driver_seek,
4     .read       = driver_read,
5     .write      = driver_write,
6     .poll       = driver_poll,
7     .open       = driver_open,
8     .release    = driver_release
9 };
```

Ogni campo della struttura deve puntare ad una funzione del driver che implementa uno specifico "operatore" su file, oppure impostata a NULL se l'operatore non è supportato. L'esatto comportamento del kernel, quando uno dei puntatori è NULL, varia da funzione a funzione. La lista seguente introduce tutti gli operatori che un'applicazione può invocare su un device. La lista è stata mantenuta snella, includendo solo i campi strettamente necessari.

- *struct module \*owner* :  
il primo campo della struttura non è un operatore, ma un puntatore al modulo che "possiede" la struttura. Il campo ha lo scopo di evitare che il modulo venga rimosso dal kernel quando uno degli operatori è in uso. Viene inizializzato usando la macro `THIS_MODULE`, definita in `<linux/module.h>`.
- *loff\_t (\*llseek) (struct file \*, loff\_t, int)* : il campo `llseek` è usato per cambiare la posizione della "testina" di lettura/ scrittura in un file. La funzione restituisce la nuova posizione della testina. `loff_t` è un intero a 64 bit (anche su architetture a 32 bit). Eventuali errori vengono segnalati con un valore di ritorno negativo. Se questo campo è posto a `NULL`, eventuali chiamate a `seek` modificheranno la posizione della testina in un modo imprevedibile.
- *ssize\_t (\*read) (struct file \*, char \_\_user \*, size\_t, loff\_t \*)* :  
usata per leggere dati dal device. Se lasciato a `NULL`, ogni chiamata a `read` fallirà e non sarà possibile leggere dal device. La funzione restituisce il numero di byte letti con successo ma, nel caso si verifichi un errore, restituisce un numero intero negativo.
- *ssize\_t (\*write) (struct file \*, const char \_\_user \*, size\_t, loff\_t \*)* :  
invia dati al device. Se `NULL` ogni chiamata alla system-call `write` causerà un errore. Il valore di ritorno, se non negativo, rappresenta il numero di byte correttamente scritti.
- *unsigned int (\*poll) (struct file \*, struct poll\_table\_struct \*)* :  
questo metodo è il back-end di tre diverse system-calls: `poll`, `epoll` e `select`, le quali sono usate per capire se una operazione di lettura/scrittura da un device possano risultare bloccanti o meno. La funzione dovrebbe restituire una maschera che indichi se sia possibile effettuare operazioni di lettura/scrittura non bloccanti, in modo che il kernel possa bloccare il processo e risvegliarlo solo quando tali operazioni diventino possibili. Se viene lasciata `NULL` si intende che le operazioni di lettura/scrittura sul device siano sempre non-bloccanti.
- *int (\*open) (struct inode \*, struct file \*)* :  
Anche se, di solito, è la prima operazione che si effettua su un file, non è strettamente necessaria la sua implementazione. Se lasciata `NULL`, l'apertura del device andrà comunque a buon fine, ma al driver non verrà inviata alcuna notifica.
- *int (\*release) (struct inode \*, struct file \*)* :  
questo operatore viene invocato quando il file viene rilasciato. Come `open`, può essere lasciato `NULL`.

L'inizializzazione di un device a caratteri passa anche attraverso la definizione di questo tipo di operatori. Essi possono essere impostati attraverso l'uso della funzione

```
1 void cdev_init (struct cdev *cdev, const struct file_operations *fops);
```

la quale prende, come parametri

- `cdev`: puntatore a struttura `cdev` da inizializzare;
- `fops`: puntatore a struttura `file_operation` con cui inizializzare il device.

#### Creazione del device

Il passo successivo è la registrazione del device e la sua aggiunta al filesystem. Tale operazione può essere effettuata chiamando

```
1 struct device * device_create( struct class *class, struct device *parent, dev_t devt, const char *fmt,
    ...)
```

- `class`: puntatore alla struttura `class` alla quale il device deve essere registrato
- `parent`: puntatore ad eventuale device parent
- `devt`: `tmajor number`
- `fmt`: nome del device.

La funzione può essere usata solo sulla classe dei device a caratteri. Crea un device all'interno del filesystem, associandogli il major number preventivamente inizializzato. La funzione restituisce il puntatore alla struttura device creata all'interno del filesystem. Si noti che il puntatore alla struttura classes DEVE essere stato precedentemente creato attraverso una chiamata alla funzione `class_create()`.

#### Aggiunta del device

Il driver, a questo punto, è pronto per essere aggiunto. È possibile aggiungere il driver usando

```
1 int cdev_add (struct cdev *p, dev_t dev, unsigned count);
```

La quale accetta come parametri

- p: puntatore a struttura cdev structure per il device
- dev: device number (precedentemente inizializzato usando la funzione `alloc_chrdev_region()`)
- count: numero di minor-numbers richiesti per il device

La funzione restituisce un numero negativo in caso di errore.

#### Accedere al segmento di memoria a cui la periferica è mappata

Un driver, tipicamente, prende possesso del segmento di memoria cui è mappato il device con la funzione di probe. Il problema è che il device è mappato ad un indirizzo di memoria fisico ed il Kernel, così come qualsiasi altro programma, lavora su indirizzi di memoria virtuali. La funzione

```
1 int of_address_to_resource(struct device_node *node, int index, struct resource *r);
```

popola una struttura resource con l'indirizzo di memoria cui è mappato il device usando le informazioni contenute all'interno del device tree. Ad esempio, se il device tree contiene

```
1 reg = <0x41200000 0x10000>;
```

significa che l'indirizzo fisico associato al device è l'indirizzo 0x41200000, che al device sono riservati 0x10000 bytes. `of_address_to_resource()` setterà `res.start = 0x41200000` e `res.end = 0x4120ffff`.

#### Allocazione della memoria del device

Le regioni di memoria per di I/O vanno allocate prima di poter essere usate.

```
1 struct resource *request_mem_region(unsigned long start, unsigned long len, char *name);
```

Questa funzione alloca una regione di memoria di len byte a partire da start restituendone l'indirizzo, mentre nel caso in cui si verifichi un errore viene restituito NULL. La funzione viene chiamata per ottenere l'accesso esclusivo della regione di memoria, per evitare che driver diversi tentino di accedere allo stesso spazio di memoria.

#### Remapping

L'allocazione dello spazio di memoria non è l'unico step da eseguire prima che tale memoria possa essere usata. È necessario fare in modo che sia resa accessibile al kernel attraverso un mapping, usando la funzione.

```
1 void *ioremap(unsigned long phys_addr, unsigned long size);
```

#### Registrazione di un interrupt-handler

Il modulo deve registrare un handler per gli interrupt. L'handler deve essere compatibile con il tipo puntatore a funzione `irq_handler_t`, così definito.

```
1 struct irqreturn_t (*irq_handler_t)(int irq, struct pt_regs * regs);
```

Il modulo definisce la funzione `myGPIOK_irq_handler()`. L'handler può essere registrato usando

```
1 int request_irq(    unsigned int irqNumber,
2                    irqreturn_t (*handler)(int, void *, struct pt_regs *),
3                    unsigned long irqflags,
4                    const char *devname,
5                    void *dev_id);
```

IL parametro `irqNumber` può essere determinato automaticamente usando la funzione

```
1 unsigned int irq_of_parse_and_map(struct device_node *node, int index);
```

La funzione `irq_of_parse_and_map()` effettua un look-up nella specifica degli interrupt all'interno del device tree e restituisce un irq number così come de lo aspetta `request_irq()` (cioè compaci con l'enumerazione in `/proc/interrupts`). Il secondo argomento della funzione è, tipicamente, zero, ad indicare che, all'interno del device tree, verrà preso in considerazione soltanto il primo degli interrupt specificate. Il device tree, nella sezione dedicata al gpio, reca

```
1 interrupts = <0 29 4>;
```

Il primo numero (0) è un flag che indica se l'interrupt sia connesso ad una line SPI (shared peripheral interrupt). Un valore diverso da zero indica che la linea è SPI. Il secondo numero si riferisce all'interrupt number. Per farla breve, quando si definisce la parte hardware, in questo specifico esempio il device GPIO è connesso alla linea 61 del GIC. Sottraendo 32 si orriene 29. Il terzo numero si riferisce alla tipologia dell'interrupt. Sono possibili tre valori:

- 0 : power-up default
- 1 : rising-edge
- 4 : a livelli, active alto

#### Inizializzazione della wait-queue per la system-call `read()` e `poll()`

In linux una wait queue viene implementata da una struttura dati `wait_queue_head_t`, definita in `<linux/wait.h>`. Il driver in questione prevede due wait-queue differenti: una per la system-call `read()` ed una per la system-call `poll()`. Entrambe le code vengono inizializzate dalla funzione `myGPIOK_probe()`.

```
1 init_waitqueue_head(&my_queue);
```

Si veda la documentazione della funzione `myGPIOK_read()` per dettagli ulteriori.

#### Inizializzazione degli spinlock

I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spilock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`. L'inizializzazione di uno spinlock avviene usando la funzione

```
1 void spin_lock_init(spinlock_t *lock);
```

#### Abilitazione degli interrupt del device

A seconda del valore `CFLAGS_myGPIOK.o` (si veda il Makefile a corredo), vengono abilitati gli interrupt della periferica. Se si tratta del GPIO Xilinx vengono abilitati gli interrupt globali e gli interrupt sul canale due. Se si tratta del device GPIO custom, essendo esso parecchio più semplice, è necessario abilitare solo gli interrupt globali.

#### 4.10.3.8 unsigned myGPIOK\_PendingPinInterrupt ( myGPIOK\_t \* device )

Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

## Restituisce

maschera che riporta i pin per i quali gli interrupt non sono stati ancora serviti;

## 4.10.3.9 void myGPIOK\_PinInterruptAck ( myGPIOK\_t \* device, unsigned mask )

Invia al device notifica di servizio di un interrupt;.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	mask maschera di selezione degli interrupt da notificare; quelli non selezionati non vengono notificati;

## 4.10.3.10 void myGPIOK\_PinInterruptDisable ( myGPIOK\_t \* device, unsigned mask )

Disabilita gli interrupt per i singoli pin del device.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	maschera di selezione degli interrupt da disabilitare; quelli non selezionati non vengono disabilitati;

## 4.10.3.11 void myGPIOK\_PinInterruptEnable ( myGPIOK\_t \* device, unsigned mask )

Abilita gli interrupt per i singoli pin del device.

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
<i>in</i>	<i>mask</i>	maschera di selezione degli interrupt da abilitare; quelli non selezionati non vengono abilitati;

## 4.10.3.12 void myGPIOK\_ResetCanRead ( myGPIOK\_t \* device )

Reset del flag "interrupt occurred" (canRead)

## Parametri

<i>in</i>	<i>device</i>	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
-----------	---------------	---

## Reset del flag "interrupt occurred" per read() bloccanti

Nel momento in cui il processo viene risvegliato e la condizione della quale era in attesa è tale che esso può continuare la sua esecuzione, è necessario resettare tale flag. Questa operazione va effettuata per prevenire race-condition dovute al risveglio di più processi in attesa del manifestarsi dello stesso evento. Il reset del flag va, pertanto, effettuato in mutua esclusione.

I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spinlock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi

di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`. Esistono diversi modi per acquisire uno spinlock. Nel seguito viene usata la funzione

```
1 void spin_lock(spinlock_t *lock);
```

per rilasciare uno spinlock, invece, verrà usata

```
1 void spin_unlock(spinlock_t *lock);
```

#### 4.10.3.13 void myGPIOK\_SetCanRead ( myGPIOK\_t \* device )

Set del flag "interrupt occurred" (canRead)

Parametri

in	device	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	--------	---

Setting del valore del flag "interrupt occurred"

Dopo aver disabilitato gli interrupt della periferica, occorre settare in modo appropriato il flag "interrupt occurred", in modo che i processi in attesa possano essere risvegliati in modo sicuro. Per prevenire race condition, tale operazione viene effettuata mutua esclusione. I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono essere usati in codice non interrompibile. Gli spinlock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`. Esistono diversi modi per acquisire uno spinlock. Nel seguito viene usata la funzione

```
1 void spin_lock_irqsave(spinlock_t *lock, unsigned long flags);
```

la quale disabilita gli interrupt sul processore locale prima di acquisire lo spinlock, per poi ripristinarlo quando lo spinlock viene rilasciato, usando

```
1 void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags);
```

#### 4.10.3.14 void myGPIOK\_TestCanReadAndSleep ( myGPIOK\_t \* device )

Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.

Parametri

in	device	puntatore a struttura <a href="#">myGPIOK_t</a> , che si riferisce al device su cui operare
----	--------	---

Porre un processo nello stato sleeping

Quando un processo viene messo nello stato sleep, lo si fa aspettandosi che una condizione diventi vera in futuro. Al risveglio, però, non c'è nessuna garanzia che quella particolare condizione sia ancora vera, per cui essa va nuovamente testata. Il modo più semplice per porre un processo nello stato sleeping è chiamare la macro `wait_event()`, o una delle sue varianti: essa combina la gestione della messa in sleeping del processo ed il check della condizione che il processo si aspetta diventi vera.

```
1 wait_event_interruptible(queue, condition);
```

Il parametro `queue` è la coda di attesa mentre `condition` è la condizione che, valutata `true`, causa la messa in `sleep` del processo. La condizione viene valutata sia prima che il processo sia messo in `sleeping` che al suo risveglio. Lo `sleep` in cui il processo viene messo chiamando `wait_event_interruptible()` può essere interrotto anche da un segnale, per cui la macro restituisce un intero che, se diverso da zero, indica che il processo è stato risvegliato da un segnale.

La condizione sulla quale i processi vengono bloccati riguarda il flag "interrupt occurred". Fin quando questo flag, posto in `and` con la maschera `MYGPIOK_SREAD`, è zero, il processo deve restare bloccato, per cui i processi che effettuano `read()` bloccante restano bloccati finché `int_occurred & MYGPIO_SREAD == 0`. Quando tale uguaglianza non sarà più valida, perché il valore di `int_occurred` viene settato dalla funzione `myGPIOK_irq_handler()`, allora il processo verrà risvegliato.

#### 4.10.3.15 void myGPIOK\_WakeUp ( myGPIOK\_t \* device )

Risveglia i process in attesa sulle code di `read` e `poll`.

##### Parametri

<code>in</code>	<code>device</code>	puntatore a struttura <code>myGPIOK_t</code> , che si riferisce al device su cui operare
-----------------	---------------------	--

##### Wakeup dei processi sleeping

La ISR deve chiamare esplicitamente `wakeup()` per risvegliare i processi messi in `sleeping` in attesa che un particolare evento si manifestasse. La funzione

```
1 void wake_up_interruptible(wait_queue_head_t *queue);
```

risveglia tutti i processi posti in una determinata coda (risvegliando solo quelli che, in precedenza, hanno effettuato una chiamata a `wait_event_interruptible()`). Se due processi vengono risvegliati contemporaneamente potrebbero originarsi `race-condition`.





## Capitolo 5

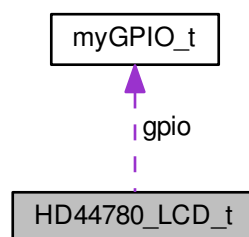
# Documentazione delle classi

### 5.1 Riferimenti per la struct HD44780\_LCD\_t

Struttura opaca che astrae un device Display LCD con cntroller Hitachi HD44780, o compatibile. Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi. Il modulo permette di utilizzare sia l'interfacciamento ad otto bit che quello a quattro bit, inizializzando il device opportunamente, attraverso l'uso delle funzioni `HD44780_Init8` e `HD44780_Init4`. Il modulo fornisce anche semplici funzioni per la stampa di un carattere o di una stringa null-terminated di caratteri. Si veda la documentazione delle funzioni [HD44780\\_Printc\(\)](#) e [HD44780\\_Print\(\)](#). Inoltre sono presenti diverse funzioni di utilità generica, come quelle per la pulizia del display, per lo spostamento del cursore di un posto in avanti o indietro, alla riga in basso o in alto.

```
#include <hd44780.h>
```

Diagramma di collaborazione per HD44780\_LCD\_t:



### Campi

- [myGPIO\\_t \\* gpio](#)
- [myGPIO\\_mask RS](#)
- [myGPIO\\_mask RW](#)
- [myGPIO\\_mask E](#)
- [myGPIO\\_mask Data7](#)
- [myGPIO\\_mask Data6](#)
- [myGPIO\\_mask Data5](#)
- [myGPIO\\_mask Data4](#)

- [myGPIO\\_mask Data3](#)
- [myGPIO\\_mask Data2](#)
- [myGPIO\\_mask Data1](#)
- [myGPIO\\_mask Data0](#)
- [HD44780\\_InterfaceMode\\_t iface\\_mode](#)

### 5.1.1 Descrizione dettagliata

Struttura opaca che astrae un device Display LCD con cntroller Hitachi HD44780, o compatibile. Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi. Il modulo permette di utilizzare sia l'interfacciamento ad otto bit che quello a quattro bit, inizializzando il device opportunamente, attraverso l'uso delle funzioni [HD44780\\_Init8](#) e [HD44780\\_Init4](#). Il modulo fornisce anche semplici funzioni per la stampa di un carattere o di una stringa null-terminated di caratteri. Si veda la documentazione delle funzioni [HD44780\\_Printc\(\)](#) e [HD44780\\_Print\(\)](#). Inoltre sono presenti diverse funzioni di utilità generica, come quelle per la pulizia del display, per lo spostamento del cursore di un posto in avanti o indietro, alla riga in basso o in alto.

Esempi:

[bsp\\_example.c](#).

### 5.1.2 Documentazione dei campi

#### 5.1.2.1 myGPIO\_mask Data0

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D0

#### 5.1.2.2 myGPIO\_mask Data1

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D1

#### 5.1.2.3 myGPIO\_mask Data2

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D2

#### 5.1.2.4 myGPIO\_mask Data3

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D3

#### 5.1.2.5 myGPIO\_mask Data4

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D4

#### 5.1.2.6 myGPIO\_mask Data5

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D5

#### 5.1.2.7 myGPIO\_mask Data6

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D6

#### 5.1.2.8 myGPIO\_mask Data7

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale D7

#### 5.1.2.9 myGPIO\_mask E

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale E

#### 5.1.2.10 myGPIO\_t\* gpio

puntatore a struttura [myGPIO\\_t](#), che astrae il particolare myGPIO usato per il pilotaggio del display

#### 5.1.2.11 HD44780\_InterfaceMode\_t iface\_mode

modalità di funzionamento dell'interfaccia verso il display (4 oppure 8 bit)

#### 5.1.2.12 myGPIO\_mask RS

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale RS

#### 5.1.2.13 myGPIO\_mask RW

maschera di selezione per il pin del device myGPIO usato per il pilotaggio del segnale RW

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

- [Src/myGPIO/bare-metal/HD44780/hd44780.h](#)

## 5.2 Riferimenti per la struct myGPIO\_t

Struttura che astrae un device myGPIO.

```
#include <myGPIO.h>
```

### Campi

- uint32\_t \* [base\\_address](#)
- uint8\_t [mode\\_offset](#)
- uint8\_t [write\\_offset](#)
- uint8\_t [read\\_offset](#)
- uint8\_t [gies\\_offset](#)
- uint8\_t [pie\\_offset](#)
- uint8\_t [irq\\_offset](#)
- uint8\_t [iack\\_offset](#)

### 5.2.1 Descrizione dettagliata

Struttura che astrae un device myGPIO.

Comprende l'indirizzo di memoria a cui il device è mappato e gli offset dei registri attraverso i quali è possibile interagire con il device stesso. La struttura è pensata, com'è ovvio, per consentire l'uso di più device GPIO nello stesso programma, identificando ciascuno di essi attraverso l'indirizzo di memoria al quale sono mappati. La struttura di

cui sopra, nel resto dell'implementazione del driver, è totalmente trasparente a chi la utilizza, nel senso che non è strettamente necessario conoscerne i dettagli per poter utilizzare il driver.

Esempi:

[bsp\\_example.c](#), [interrupt\\_bare.c](#), [noDriver.c](#), [uio-int.c](#), e [uio.c](#).

## 5.2.2 Documentazione dei campi

### 5.2.2.1 uint32\_t\* base\_address

indirizzo base

### 5.2.2.2 uint8\_t gies\_offset

offset, rispetto all'indirizzo base, del registro "gies"

### 5.2.2.3 uint8\_t iack\_offset

offset, rispetto all'indirizzo base, del registro "iack"

### 5.2.2.4 uint8\_t irq\_offset

offset, rispetto all'indirizzo base, del registro "irq"

### 5.2.2.5 uint8\_t mode\_offset

offset, rispetto all'indirizzo base, del registro "mode"

### 5.2.2.6 uint8\_t pie\_offset

offset, rispetto all'indirizzo base, del registro "pie"

### 5.2.2.7 uint8\_t read\_offset

offset, rispetto all'indirizzo base, del registro "read"

### 5.2.2.8 uint8\_t write\_offset

offset, rispetto all'indirizzo base, del registro "write"

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

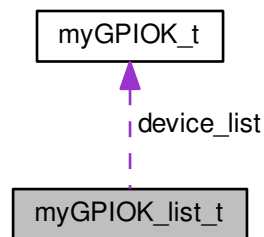
- [Src/myGPIO/bare-metal/myGPIO.h](#)

## 5.3 Riferimenti per la struct myGPIOK\_list\_t

Struttura dati per la gestione degli oggetti [myGPIOK\\_t](#) gestiti dal driver.

```
#include <myGPIOK_list.h>
```

Diagramma di collaborazione per myGPIOK\_list\_t:



## Campi

- [myGPIOK\\_t](#) \*\* [device\\_list](#)
- [uint32\\_t](#) [list\\_size](#)
- [uint32\\_t](#) [device\\_count](#)

### 5.3.1 Descrizione dettagliata

Struttura dati per la gestione degli oggetti [myGPIOK\\_t](#) gestiti dal driver.

La struttura dati, sebbene non strettamente necessaria alla gestione dei diversi oggetti [myGPIOK\\_t](#), ciascuno dei quali corrispondente ad un diverso device gestito dal driver myGPIOK, è pensata per semplificare l'accesso a questi ultimi, tenendo un riferimento a tutti gli oggetti e le strutture dati coinvolte nel funzionamento del modulo in un unico "posto", accessibile attraverso questa struttura dati.

### 5.3.2 Documentazione dei campi

#### 5.3.2.1 [uint32\\_t](#) [device\\_count](#)

numero di device correntemente attivi e gestiti dal driver

#### 5.3.2.2 [myGPIOK\\_t](#)\*\* [device\\_list](#)

array di puntatori a struttura [myGPIOK\\_t](#), ciascuno dei quali si riferisce ad un device differente

#### 5.3.2.3 [uint32\\_t](#) [list\\_size](#)

dimensione dell'array, corrisponde al numero massimo di device gestibili, definito in fase di inizializzazione

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

- [Src/myGPIO/linux-driver/myGPIOK\\_list.h](#)

## 5.4 Riferimenti per la struct myGPIOK\_t

Stuttura per l'astrazione di un device myGPIO in kernel-mode.

```
#include <myGPIOK_t.h>
```

## Campi

- dev\_t [Mm](#)
- struct platform\_device \* [op](#)
- struct cdev [cdev](#)
- struct device \* [dev](#)
- struct class \* [class](#)
- uint32\_t [irqNumber](#)
- uint32\_t [irq\\_mask](#)
- struct resource [rsrc](#)
- struct resource \* [mreg](#)
- uint32\_t [rsrc\\_size](#)
- void \* [vrtl\\_addr](#)
- wait\_queue\_head\_t [read\\_queue](#)
- wait\_queue\_head\_t [poll\\_queue](#)
- uint32\_t [can\\_read](#)
- spinlock\_t [slock\\_int](#)
- uint32\_t [total\\_irq](#)
- spinlock\_t [sl\\_total\\_irq](#)

### 5.4.1 Descrizione dettagliata

Stuttura per l'astrazione di un device myGPIO in kernel-mode.

è buona abitudine, se non quasi indispensabile, definire una struttura dati nella quale contenere tutto ciò che è legato al device o al driver. In questo modulo viene usata la struttura [myGPIOK\\_t](#) per contenere tutto ciò che è necessario al funzionamento del driver.

### 5.4.2 Documentazione dei campi

#### 5.4.2.1 uint32\_t can\_read

Flag "puoi leggere" Il valore viene settato dalla funzione [myGPIOK\\_irq\\_handler\(\)](#) al manifestarsi di un interrupt, prima di risvegliare i processi in attesa di un interrupt. I processi che effettuano read() bloccante restano bloccati finché `int_occurred = 0`

#### 5.4.2.2 struct cdev cdev

Stuttura per l'astrazione di un device a caratteri Il kernel usa, internamente, una struttura cdev per rappresentare i device a caratteri. Prima che il kernel invochi le funzioni definite dal driver per il device, bisogna allocare e registrare uno, o più, oggetti cdev. In questo caso è sufficiente allocare uno solo di questi oggetti.

#### 5.4.2.3 struct class\* class

#### 5.4.2.4 struct device\* dev

#### 5.4.2.5 uint32\_t irq\_mask

maschera delle interruzioni interne per il device

#### 5.4.2.6 `uint32_t irqNumber`

interrupt-number a cui il device è connesso. Restituito dalla chiamata alla funzione `irq_of_parse_and_map()`

#### 5.4.2.7 `dev_t Mm`

Major e minor number associati al device

#### 5.4.2.8 `struct resource* mreg`

puntatore alla regione di memoria cui il device è mappato

#### 5.4.2.9 `struct platform_device* op`

Puntatore a struttura `platform_device` cui l'oggetto `myGPIOK_t` si riferisce

#### 5.4.2.10 `wait_queue_head_t poll_queue`

wait queue per la system-call `poll()`

#### 5.4.2.11 `wait_queue_head_t read_queue`

wait queue per la system-call `read()` Una chiamata a `read()` potrebbe arrivare quando i dati non sono disponibili, ma potrebbero esserlo in futuro, oppure, una chiamata a `write()` potrebbe avvenire quando il device non è in grado di accettare altri dati (perché il suo buffer di ingresso potrebbe essere pieno). Il processo chiamante non ha la minima conoscenza delle dinamiche interne del device, per cui, nell'impossibilità di servire la richiesta, il driver deve bloccare il processo e metterlo tra i processi "sleeping", fin quando la richiesta non può essere servita. Tutti i processi in attesa di un particolare evento vengono posti all'interno della stessa wait queue. In linux una wait queue viene implementata da una struttura dati `wait_queue_head_t`, definita in `<linux/wait.h>`.

#### 5.4.2.12 `struct resource rsrc`

Struttura che astrae una risorsa device, dal punto di vista della memoria alla quale la risorsa è mappata. In particolare i campi "start" ed "end" contengono, rispettivamente, il primo e l'ultimo indirizzo fisico a cui il device è mappato.

#### 5.4.2.13 `uint32_t rsrc_size`

`rsrc.end - rsrc.start` numero di indirizzi associati alla periferica. occorre per effettuare il mapping indirizzo fisico - indirizzo virtuale

#### 5.4.2.14 `spinlock_t sl_total_irq`

Spinlock usato per garantire l'accesso in mutua esclusione alla variabile `total_irq` da parte delle funzioni del modulo

#### 5.4.2.15 `spinlock_t slock_int`

Spinlock usato per garantire l'accesso in mutua esclusione alla variabile `int_occurred` da parte delle funzioni del modulo. I semafori sono uno strumento potentissimo per per l'implementazione di sezioni critiche, ma non possono

essere usati in codice non interrompibile. Gli spillock sono come i semafori, ma possono essere usati anche in codice non interrompibile, come può esserlo un modulo kernel. Sostanzialmente se uno spinlock è già stato acquisito da qualcun altro, si entra in un hot-loop dal quale si esce solo quando chi possiede lo spinlock lo rilascia. Trattandosi di moduli kernel, è di vitale importanza che la sezione critica sia quanto più piccola possibile. Ovviamente l'implementazione è "un pò" più complessa di come è stata descritta, ma il concetto è questo. Gli spinlock sono definiti in `<linux/spinlock.h>`.

#### 5.4.2.16 `uint32_t total_irq`

numero totale di interrupt manifestatesi

#### 5.4.2.17 `void* vrtl_addr`

indirizzo virtuale della periferica

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

- [Src/myGPIO/linux-driver/myGPIOK\\_t.h](#)

## 5.5 Riferimenti per la struct `param_t`

La struttura raccoglie tutti i parametri di esecuzione del programma.

### Campi

- `int dev_descr`  
*device descriptor*
- `uint8_t op_mode`  
*impostato ad 1 se l'utente intende effettuare scrittura su mode*
- `uint32_t mode_value`  
*valore che l'utente intende scrivere nel registro mode*
- `uint8_t op_write`  
*impostato ad 1 se l'utente intende effettuare scrittura su write*
- `uint32_t write_value`  
*valore che l'utente intende scrivere nel registro write*
- `uint8_t op_read`  
*impostato ad 1 se l'utente intende effettuare lettura da read*

### 5.5.1 Descrizione dettagliata

La struttura raccoglie tutti i parametri di esecuzione del programma.

Esempi:

[mygpiok.c](#).

### 5.5.2 Documentazione dei campi

#### 5.5.2.1 `int dev_descr`

device descriptor



Esempi:

[mygpiok.c](#).

#### 5.5.2.2 uint32\_t mode\_value

valore che l'utente intende scrivere nel registro mode

Esempi:

[mygpiok.c](#).

#### 5.5.2.3 uint8\_t op\_mode

impostato ad 1 se l'utente intende effettuare scrittura su mode

Esempi:

[mygpiok.c](#).

#### 5.5.2.4 uint8\_t op\_read

impostato ad 1 se l'utente intende effettuare lettura da read

Esempi:

[mygpiok.c](#).

#### 5.5.2.5 uint8\_t op\_write

impostato ad 1 se l'utente intende effettuare scrittura su write

Esempi:

[mygpiok.c](#).

#### 5.5.2.6 uint32\_t write\_value

valore che l'utente intende scrivere nel registro write

Esempi:

[mygpiok.c](#).

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

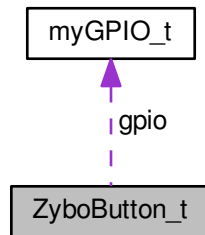
- Src/Examples/[mygpiok.c](#)

## 5.6 Riferimenti per la struct ZyboButton\_t

Struttura opaca che astrae l'insieme dei button presenti sulla board Digilent Zybo;.

```
#include <ZyboButton.h>
```

Diagramma di collaborazione per ZyboButton\_t:



### Campi

- [myGPIO\\_t \\* gpio](#)
- [myGPIO\\_mask Button3\\_pin](#)
- [myGPIO\\_mask Button2\\_pin](#)
- [myGPIO\\_mask Button1\\_pin](#)
- [myGPIO\\_mask Button0\\_pin](#)

### 5.6.1 Descrizione dettagliata

Struttura opaca che astrae l'insieme dei button presenti sulla board Digilent Zybo;.

Esempi:

[bsp\\_example.c](#).

### 5.6.2 Documentazione dei campi

#### 5.6.2.1 myGPIO\_mask Button0\_pin

maschera di selezione per il particolare bit del device myGPIO connesso al button numero 0 della board Zybo

#### 5.6.2.2 myGPIO\_mask Button1\_pin

maschera di selezione per il particolare bit del device myGPIO connesso al button numero 1 della board Zybo

#### 5.6.2.3 myGPIO\_mask Button2\_pin

maschera di selezione per il particolare bit del device myGPIO connesso al button numero 2 della board Zybo

#### 5.6.2.4 myGPIO\_mask Button3\_pin

maschera di selezione per il particolare bit del device myGPIO connesso al button numero 3 della board Zybo

#### 5.6.2.5 myGPIO\_t\* gpio

puntatore a struttura [myGPIO\\_t](#), che astrae il particolare myGPIO usato per la lettura dello stato dei button presenti sulla board

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

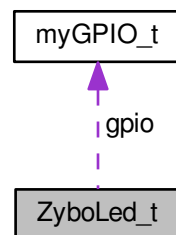
- [Src/myGPIO/bare-metal/ZyboBSP/ZyboButton.h](#)

## 5.7 Riferimenti per la struct ZyboLed\_t

Struttura opaca che astrae l'insieme dei Led presenti sulla board Digilent Zybo;.

```
#include <ZyboLed.h>
```

Diagramma di collaborazione per ZyboLed\_t:



### Campi

- [myGPIO\\_t \\* gpio](#)
- [myGPIO\\_mask Led3\\_pin](#)
- [myGPIO\\_mask Led2\\_pin](#)
- [myGPIO\\_mask Led1\\_pin](#)
- [myGPIO\\_mask Led0\\_pin](#)

#### 5.7.1 Descrizione dettagliata

Struttura opaca che astrae l'insieme dei Led presenti sulla board Digilent Zybo;.

Esempi:

[bsp\\_example.c](#).

## 5.7.2 Documentazione dei campi

### 5.7.2.1 myGPIO\_t\* gpio

puntatore a struttura [myGPIO\\_t](#), che astrae il particolare myGPIO usato per il pilotaggio dei led presenti sulla board

### 5.7.2.2 myGPIO\_mask Led0\_pin

maschera di selezione per il particolare bit del device myGPIO usato per il pilotaggio del led numero 0 della board Zybo

### 5.7.2.3 myGPIO\_mask Led1\_pin

maschera di selezione per il particolare bit del device myGPIO usato per il pilotaggio del led numero 1 della board Zybo

### 5.7.2.4 myGPIO\_mask Led2\_pin

maschera di selezione per il particolare bit del device myGPIO usato per il pilotaggio del led numero 2 della board Zybo

### 5.7.2.5 myGPIO\_mask Led3\_pin

maschera di selezione per il particolare bit del device myGPIO usato per il pilotaggio del led numero 3 della board Zybo

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

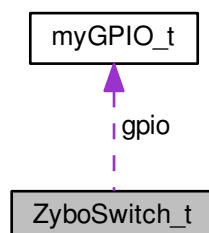
- [Src/myGPIO/bare-metal/ZyboBSP/ZyboLed.h](#)

## 5.8 Riferimenti per la struct ZyboSwitch\_t

Struttura opaca che astrae l'insieme degli switch presenti sulla board Digilent Zybo;.

```
#include <ZyboSwitch.h>
```

Diagramma di collaborazione per ZyboSwitch\_t:



## Campi

- [myGPIO\\_t \\* gpio](#)
- [myGPIO\\_mask Switch3\\_pin](#)
- [myGPIO\\_mask Switch2\\_pin](#)
- [myGPIO\\_mask Switch1\\_pin](#)
- [myGPIO\\_mask Switch0\\_pin](#)

### 5.8.1 Descrizione dettagliata

Struttura opaca che astrae l'insieme degli switch presenti sulla board Digilent Zybo;.

Esempi:

[bsp\\_example.c](#).

### 5.8.2 Documentazione dei campi

#### 5.8.2.1 [myGPIO\\_t\\* gpio](#)

puntatore a struttura [myGPIO\\_t](#), che astrae il particolare myGPIO usato per la lettura dello stato degli switch presenti sulla board

#### 5.8.2.2 [myGPIO\\_mask Switch0\\_pin](#)

maschera di selezione per il particolare bit del device myGPIO connesso allo switch numero 0 della board Zybo

#### 5.8.2.3 [myGPIO\\_mask Switch1\\_pin](#)

maschera di selezione per il particolare bit del device myGPIO connesso allo switch numero 1 della board Zybo

#### 5.8.2.4 [myGPIO\\_mask Switch2\\_pin](#)

maschera di selezione per il particolare bit del device myGPIO connesso allo switch numero 2 della board Zybo

#### 5.8.2.5 [myGPIO\\_mask Switch3\\_pin](#)

maschera di selezione per il particolare bit del device myGPIO connesso allo switch numero 3 della board Zybo

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

- [Src/myGPIO/bare-metal/ZyboBSP/ZyboSwitch.h](#)



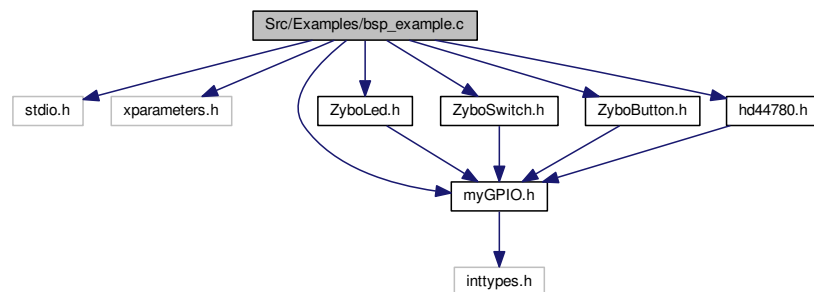
## Capitolo 6

# Documentazione dei file

### 6.1 Riferimenti per il file Src/Examples/bsp\_example.c

```
#include <stdio.h>
#include "xparameters.h"
#include "myGPIO.h"
#include "ZyboLed.h"
#include "ZyboSwitch.h"
#include "ZyboButton.h"
#include "hd44780.h"
```

Grafo delle dipendenze di inclusione per bsp\_example.c:



### Funzioni

- int `main` ()

#### 6.1.1 Documentazione delle funzioni

##### 6.1.1.1 int main ( )

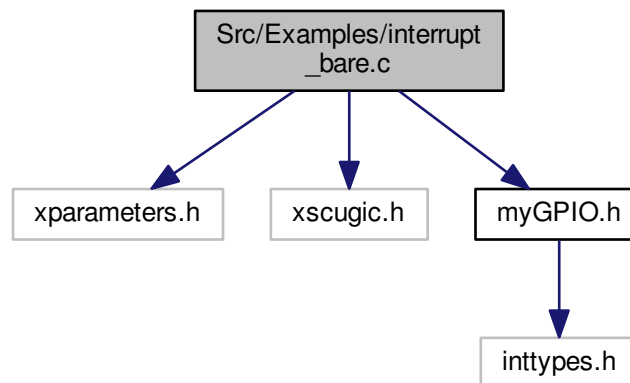
Esempi:

`bsp_example.c`.

## 6.2 Riferimenti per il file Src/Examples/interrupt\_bare.c

```
#include "xparameters.h"
#include "xscugic.h"
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per interrupt\_bare.c:



### Definizioni

- #define [led\\_base\\_addr](#) XPAR\_MYGPIO\_0\_S00\_AXI\_BASEADDR
- #define [btn\\_base\\_addr](#) XPAR\_MYGPIO\_1\_S00\_AXI\_BASEADDR
- #define [swc\\_base\\_addr](#) XPAR\_MYGPIO\_2\_S00\_AXI\_BASEADDR
- #define [led\\_irq\\_line](#) XPAR\_FABRIC\_MYGPIO\_0\_INTERRUPT\_INTR
- #define [btn\\_irq\\_line](#) XPAR\_FABRIC\_MYGPIO\_1\_INTERRUPT\_INTR
- #define [swc\\_irq\\_line](#) XPAR\_FABRIC\_MYGPIO\_2\_INTERRUPT\_INTR
- #define [gic\\_id](#) XPAR\_SCUGIC\_0\_DEVICE\_ID

### Funzioni

- void [gpio\\_init](#) (void)
- void [btn\\_isr](#) (void \*)
- void [swc\\_isr](#) (void \*)
- int [int\\_config](#) (void)
- int [main](#) ()

### Variabili

- [myGPIO\\_t](#) [led\\_gpio](#)
- [myGPIO\\_t](#) [btn\\_gpio](#)
- [myGPIO\\_t](#) [swc\\_gpio](#)
- XScuGic [GIC](#)



### 6.2.1 Documentazione delle definizioni

6.2.1.1 `#define btn_base_addr XPAR_MYGPI0_1_S00_AXI_BASEADDR`

Esempi:

[interrupt\\_bare.c.](#)

6.2.1.2 `#define btn_irq_line XPAR_FABRIC_MYGPI0_1_INTERRUPT_INTR`

Esempi:

[interrupt\\_bare.c.](#)

6.2.1.3 `#define gic_id XPAR_SCUGIC_0_DEVICE_ID`

Esempi:

[interrupt\\_bare.c.](#)

6.2.1.4 `#define led_base_addr XPAR_MYGPI0_0_S00_AXI_BASEADDR`

Esempi:

[interrupt\\_bare.c.](#)

6.2.1.5 `#define led_irq_line XPAR_FABRIC_MYGPI0_0_INTERRUPT_INTR`

6.2.1.6 `#define swc_base_addr XPAR_MYGPI0_2_S00_AXI_BASEADDR`

Esempi:

[interrupt\\_bare.c.](#)

6.2.1.7 `#define swc_irq_line XPAR_FABRIC_MYGPI0_2_INTERRUPT_INTR`

Esempi:

[interrupt\\_bare.c.](#)

### 6.2.2 Documentazione delle funzioni

6.2.2.1 `void btn_isr ( void * data )`

Esempi:

[interrupt\\_bare.c.](#)

6.2.2.2 `void gpio_init ( void )`

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.2.3 int int\_config ( void )

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.2.4 int main ( )

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.2.5 void swc\_isr ( void \* *data* )

Esempi:

[interrupt\\_bare.c.](#)

### 6.2.3 Documentazione delle variabili

#### 6.2.3.1 myGPIO\_t btn\_gpio

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.3.2 XScuGic GIC

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.3.3 myGPIO\_t led\_gpio

Esempi:

[interrupt\\_bare.c.](#)

#### 6.2.3.4 myGPIO\_t swc\_gpio

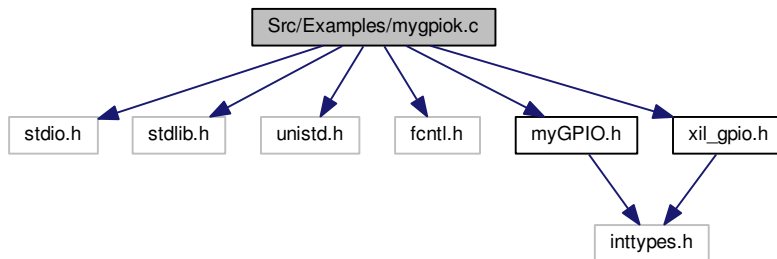
Esempi:

[interrupt\\_bare.c.](#)

## 6.3 Riferimenti per il file Src/Examples/mygpiok.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include "myGPIO.h"
#include "xil_gpio.h"
```

Grafo delle dipendenze di inclusione per mygpiok.c:



### Strutture dati

- struct [param\\_t](#)

*La struttura raccoglie tutti i parametri di esecuzione del programma.*

### Definizioni

- #define [MODE\\_OFFSET](#) [myGPIO\\_MODE\\_OFFSET](#)
- #define [WRITE\\_OFFSET](#) [myGPIO\\_WRITE\\_OFFSET](#)
- #define [READ\\_OFFSET](#) [myGPIO\\_READ\\_OFFSET](#)

### Funzioni

- void [howto](#) (void)  
*Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.*
- int [parse\\_args](#) (int argc, char \*\*argv, [param\\_t](#) \*param)  
*Effettua il parsing dei parametri passati al programma.*
- void [gpio\\_op](#) ([param\\_t](#) \*param)  
*Effettua operazioni su un device.*
- int [main](#) (int argc, char \*\*argv)

#### 6.3.1 Documentazione delle definizioni

##### 6.3.1.1 #define MODE\_OFFSET myGPIO\_MODE\_OFFSET

Esempi:

[mygpiok.c](#), [noDriver.c](#), e [uio.c](#).

### 6.3.1.2 #define READ\_OFFSET myGPIO\_READ\_OFFSET

Esempi:

[mygpiok.c](#), [noDriver.c](#), e [uio.c](#).

### 6.3.1.3 #define WRITE\_OFFSET myGPIO\_WRITE\_OFFSET

Esempi:

[mygpiok.c](#), [noDriver.c](#), e [uio.c](#).

## 6.3.2 Documentazione delle funzioni

### 6.3.2.1 void gpio\_op ( param\_t \* param )

Effettua operazioni su un device.

Parametri

in	param	puntatore a struttura <a href="#">param_t</a> , contiene i vari parametri di esecuzione del programma.
----	-------	--

La funzione viene invocata dopo che sia stato eseguito il parsing dei parametri passati al programma quando esso viene invocato. È stata scritta per funzionare sia con il GPIO Xilinx che con il GPIO custom myGPIO. È possibile utilizzare il primo definendo la macro **XIL\_GPIO**. Effettua, sul device, le operazioni impostate, in accordo con i parametri passati al programma alla sua invocazione.

Nel caso in cui si usi un driver ad-hoc, è possibile usare le funzioni `read()` e `write()` per interagire con il device, leggendo il valore dei registri o scrivendolo. Il driver myGPIOK mette a disposizione anche la funzione `seek()` che permette di scegliere quale registro leggere o scrivere.

#### Impostazione della modalità di funzionamento

Per impostare la modalità di funzionamento è necessario scrivere sul registro **MODE**. L'offset di tale registro è determinato in base al particolare device che si sta utilizzando. Dopo aver spostato la "testina di scrittura" sul registro **MODE** usando la funzione `seek()`, viene effettuata la scrittura su di esso usando la funzione `write()`. È possibile, definendo la macro **USE\_PWRITE**, usare la funzione `pwrite()`, che combina le due operazioni.

#### Operazione di scrittura

Per impostare il valore dei pin del device è necessario scrivere sul registro **WRITE**. L'offset di tale registro è determinato in base al particolare device che si sta utilizzando. Dopo aver spostato la "testina di scrittura" sul registro **WRITE** usando la funzione `seek()`, viene effettuata la scrittura su di esso usando la funzione `write()`. È possibile, definendo la macro **USE\_PWRITE**, usare la funzione `pwrite()`, che combina le due operazioni.

#### Operazione di lettura con interrupt

La lettura dei pin del device avviene mediante la chiamata alla funzione `read()`, dopo aver spostato la "testina di lettura" sul registro **READ**. L'offset di tale registro, come nei due casi precedenti, viene determinato in base al particolare device che si sta usando. È possibile, definendo la macro **USE\_PREAD**, usare la funzione `pread()`, che combina le operazioni di `seek()` e `read()`. Il driver myGPIOK implementa un meccanismo di lettura bloccante: qualora non ci siano dati disponibili, il processo che chiama `read()` viene sospeso e messo in attesa che i dati siano disponibili. Quando arriva una interruzione dal device, il driver myGPIOK lo gestisce e risveglia i processi che erano stati messi precedentemente in attesa. Si legga la documentazione del driver myGPIOK per i dettagli.

Esempi:

[mygpiok.c](#).

## 6.3.2.2 void howto ( void )

Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.

Esempi:

[mygpiok.c](#).

## 6.3.2.3 int main ( int argc, char \*\* argv )

Esempi:

[mygpiok.c](#).

## 6.3.2.4 int parse\_args ( int argc, char \*\* argv, param\_t \* param )

Effettua il parsing dei parametri passati al programma.

Parametri

in	argc	
in	argv	
out	param	puntatore a struttura <a href="#">param_t</a> , conterrà i vari parametri di esecuzione del programma.

Valori di ritorno

0	se il parsing ha successo
-1	se si verifica un errore

Parsing dei parametri del programma.

Il parsing viene effettuato usando la funzione getopt().

```
1 #include <unistd.h>
2 int getopt(int argc, char * const argv[], const char *optstring);
```

Essa prende in input i parametri argc ed argv passati alla funzione [main\(\)](#) quando il programma viene invocato. Quando una delle stringhe che compongono argv comincia con il carattere '-', getopt() la considera una opzione. Il carattere immediatamente successivo il '-' identifica la particolare opzione. La funzione può essere chiamata ripetutamente, fino a quando non restituisce -1, ad indicare che sono stati analizzati tutti i parametri passati al programma. Quando getopt() trova un'opzione, restituisce quel carattere ed aggiorna la variabile globale optind, che punta al prossimo parametro contenuto in argv. La stringa optstring indica quali sono le opzioni considerate. Se una opzione è seguita da ':' vuol dire che essa è seguita da un argomento. Tale argomento può essere ottenuto mediante la variabile globale optarg.

Parametri riconosciuti

La funzione riconosce i parametri:

- 'd' : seguito dal percorso del device /dev/myGPIOK col quale interagire
- 'w' : operazione di scrittura, seguito dal valore che si intende scrivere, in esadecimale; la scrittura verrà effettuata sul registro WRITE;
- 'm' : impostazione modalità, seguito dalla modalità col quale impostare il device; la scrittura verrà effettuata sul registro MODE;

- 'r' : operazione di lettura, primo di argomento; la lettura viene effettuata dal registro READ ed è non bloccante, nel senso che viene semplicemente letto il contenuto del registro.

Se non viene specificato il device myGPIOK col quale interagire è impossibile continuare. Per questo motivo, in questo caso, la funzione restituisce -1, per cui il programma viene terminato.

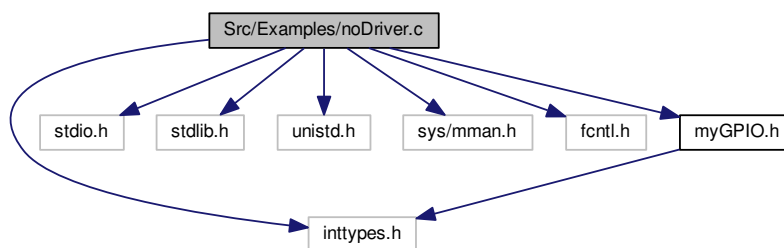
Esempi:

[mygpio.c](#).

## 6.4 Riferimenti per il file Src/Examples/noDriver.c

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per noDriver.c:



## Funzioni

- void [howto](#) (void)  
*Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.*
- int [parse\\_args](#) (int argc, char \*\*argv, uint32\_t \*gpio\_address, uint8\_t \*op\_mode, uint32\_t \*mode\_value, uint8\_t \*op\_write, uint32\_t \*write\_value, uint8\_t \*op\_read)  
*Effettua il parsing dei parametri passati al programma.*
- void [gpio\\_op](#) (void \*vrt\_gpio\_addr, uint8\_t op\_mode, uint32\_t mode\_value, uint8\_t op\_write, uint32\_t write\_value, uint8\_t op\_read)  
*Effettua operazioni su un device.*
- int [main](#) (int argc, char \*\*argv)  
*funzione [main\(\)](#).*

### 6.4.1 Documentazione delle funzioni

6.4.1.1 void [gpio\\_op](#) ( void \* vrt\_gpio\_addr, uint8\_t op\_mode, uint32\_t mode\_value, uint8\_t op\_write, uint32\_t write\_value, uint8\_t op\_read )

Effettua operazioni su un device.

**Parametri**

in	<i>vrt_gpio_addr</i>	indirizzo di memoria del device gpio
in	<i>op_mode</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
in	<i>mode_value</i>	conterrà il valore che l'utente intende scrivere nel registro mode
in	<i>op_write</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
in	<i>write_value</i>	conterrà il valore che l'utente intende scrivere nel registro write
in	<i>op_read</i>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

La funzione viene invocata dopo che sia stato eseguito il parsing dei parametri passati al programma quando esso viene invocato. È stata scritta per funzionare sia con il GPIO Xilinx che con il GPIO custom myGPIO. È possibile utilizzare il primo definendo la macro **XIL\_GPIO**. Effettua, sul device, le operazioni impostate, in accordo con i parametri passati al programma alla sua invocazione.

**Impostazione della modalità di funzionamento**

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di impostazione della modalità di funzionamento del GPIO vengono effettuate scrivendo direttamente sul registro MODE del device. In caso contrario si è preferito utilizzare la funzione `myGPIO_setMode()` (Si veda il modulo myGPIO). Funzionalmente non c'è differenza.

**Operazione di scrittura**

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di scrittura del valore dei pin del device GPIO vengono effettuate scrivendo direttamente sul registro WRITE del device. In caso contrario si è preferito utilizzare la funzione `myGPIO_setValue()` (Si veda il modulo myGPIO). Funzionalmente non c'è differenza.

**Operazione di lettura**

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di lettura del valore dei pin del device GPIO vengono effettuate leggendo direttamente dal registro READ del device. In caso contrario si è preferito utilizzare la funzione `myGPIO_getRead()` (Si veda il modulo myGPIO). Funzionalmente non c'è differenza. La lettura è non bloccante: viene semplicemente letto il valore contenuto nel registro perché tale modalità di interazione non permette l'implementazione di un meccanismo di lettura basato su interruzioni.

**Esempi:**

[noDriver.c](#).

**6.4.1.2 void howto ( void )**

Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.

**Esempi:**

[noDriver.c](#).

**6.4.1.3 int main ( int argc, char \*\* argv )**

funzione [main\(\)](#).

**Parsing dei parametri di invocazione**

Il parsing dei parametri passati al programma all'atto della sua invocazione viene effettuato dalla funzione [parse\\_args\(\)](#). Si rimanda alla sua documentazione per i dettagli sui parametri riconosciuti.

Se non viene specificato l'indirizzo fisico del device al quale accedere è impossibile continuare. Per questo motivo, in questo caso, il programma viene terminato.

### Apertura di /dev/mem

In questo specifico esempio l'interfacciamento avviene da user-space, agendo direttamente sui registri di memoria, senza mediazione di altri driver, usando il device /dev/mem. Questo presuppone che si sia nelle condizioni di poter calcolare dell'indirizzo di memoria virtuale del device.

L'accesso al device /dev/mem viene ottenuto mediante la system-call `open()`:

```
1 #include <sys/stat.h>
2 #include <fcntl.h>
3 int open(const char *path, int oflag, ... );
```

la quale restituisce il descrittore del file /dev/mem, usato nel seguito per effettuare le operazioni di I/O. I valori del parametro `oflag` specificano il modo in cui il file /dev/mem viene aperto. In questo caso viene usato `O_RDWR`, il quale garantisce accesso in lettura ed in scrittura. Altri valori sono `O_RDONLY`, il quale garantisce accesso in sola lettura, ed `O_WRONLY`, che, invece, garantisce accesso in sola scrittura.

### Calcolo dell'indirizzo virtuale del device.

Linux implementa la segregazione della memoria. Vale a dire che un processo può accedere solo agli indirizzi di memoria (virtuali) appartenenti al suo address-space. Se è necessario effettuare un accesso ad un indirizzo specifico, bisogna effettuare il mapping di quell'indirizzo nell'address space del processo. Linux implementa la paginazione della memoria, quindi l'indirizzo del quale si desidera effettuare il mapping, apparterrà ad una specifica pagina di memoria. Per sapere a quale pagina appartenga l'indirizzo, è necessario conoscere quale sia la dimensione delle pagine di memoria. Tipicamente la dimensione delle pagine è una potenza del due. Si supponga che l'indirizzo di cui si vuole fare il mapping è **0x43C002F0** e che la dimensione delle pagine sia 16KB. Scrivendo la dimensione delle pagine in esadecimale

**0x00002000**

sottraendo 1

**0x00001FFF**

negando

**0xFFFFE000**

si ottiene una maschera che, posta in `and` con un indirizzo, restituisce l'indirizzo della pagina di memoria a cui l'indirizzo appartiene. In questo caso

**0x43C002F0 & 0xFFFFE000 = 0x43C00000**

L'indirizzo della pagina potrà essere usato per il mapping, ma per accedere allo specifico indirizzo è necessario calcolarne l'offset, sottraendogli l'indirizzo della pagina. In questo modo, dopo aver effettuato il mapping, si potrà accedere allo stesso a partire dall'indirizzo virtuale della pagina stessa.

### Conversione dell'indirizzo fisico in indirizzo virtuale

La "conversione" dell'indirizzo fisico del device in indirizzo virtuale appartenente allo spazio di indirizzamento del processo viene effettuato tramite la chiamata alla funzione `mmap()`, la quale stabilisce un mapping tra lo spazio di indirizzamento di un processo ed un file, una porzione di memoria condivisa o un qualsiasi altro memory-object, restituendo un indirizzo virtuale valido, attraverso il quale è possibile accedere al blocco di memoria fisico.



```
1 #include <sys/mman.h>
2 void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
```

Per semplicità supponiamo che la chiamata alla funzione sia la seguente:

```
pa=mmap(addr, len, prot, flags, fildes, off);
```

la semantica dei diversi parametri è:

- pa: indirizzo virtuale dell'address-space locale del processo, a cui viene eseguito il map; se il mapping ha successo viene restituito qualcosa di diverso da MAP\_FAILED;
- addr:
- len: lunghezza, in byte, del blocco mappato; in questo caso viene usato il valore restituito da sysconf(\_SC\_PAGESIZE);
- prot: specifica i permessi di accesso al blocco di memoria del quale si sta facendo il mapping;
  - PROT\_READ indica che il blocco può essere letto;
  - PROT\_WRITE indica che il blocco può essere scritto;
  - PROT\_NONE sta ad indicare che il blocco non può essere acceduto;
- flags: fornisce informazioni aggiuntive circa la gestione del blocco di dati di cui si sta facendo il mapping; il valore del flag può essere uno dei seguenti:
  - MAP\_SHARED: modifiche al blocco sono condivise con chiunque altri lo stia usando;
  - MAP\_PRIVATE: le modifiche sono private;
- fildes: descrittore del file /dev/mem
- off: indirizzo fisico del blocco che si intende mappare; è necessario che sia allineato alla dimensione della pagina di memoria, così come restituito dalla funzione sysconf(\_SC\_PAGESIZE);

In questo caso la chiamata a mmap avviene con i seguenti parametri:

```
1 uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
2 uint32_t page_mask = ~(page_size-1); // maschera di conversione indirizzo -> indirizzo pagina
3 uint32_t page_addr = gpio_addr & page_mask; // indirizzo della "pagina fisica" a cui è mappato il
    device
4 uint32_t offset = gpio_addr - page_addr; // offset del device rispetto all'indirizzo della pagina
5 void* vrt_page_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, page_addr) +
    offset;
```

### Operazioni sul device

Una volta effettuato il mapping, le operazioni preventivate con l'invocazione del programma vengono effettuate dalla funzione `gpio_op()`. Si rimanda alla sua documentazione per i dettagli sulle operazioni effettuate().

Esempi:

`noDriver.c`.

```
6.4.1.4 int parse_args ( int argc, char ** argv, uint32_t * gpio_address, uint8_t * op_mode, uint32_t * mode_value, uint8_t *
    op_write, uint32_t * write_value, uint8_t * op_read )
```

Effettua il parsing dei parametri passati al programma.

**Parametri**

in	<i>argc</i>	
in	<i>argv</i>	
out	<i>gpio_address</i>	conterrà l'indirizzo di memoria del device gpio
out	<i>op_mode</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
out	<i>mode_value</i>	conterrà il valore che l'utente intende scrivere nel registro mode
out	<i>op_write</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
out	<i>write_value</i>	conterrà il valore che l'utente intende scrivere nel registro write
out	<i>op_read</i>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

**Valori di ritorno**

0	se il parsing ha successo
-1	se si verifica un errore

**Parsing dei parametri del programma.**

Il parsing viene effettuato usando la funzione getopt().

```
1 #include <unistd.h>
2 int getopt(int argc, char * const argv[], const char *optstring);
```

Essa prende in input i parametri argc ed argv passati alla funzione [main\(\)](#) quando il programma viene invocato. Quando una delle stringhe che compongono argv comincia con il carattere '-', getopt() la considera una opzione. Il carattere immediatamente successivo il '-' identifica la particolare opzione. La funzione può essere chiamata ripetutamente, fino a quando non restituisce -1, ad indicare che sono stati analizzati tutti i parametri passati al programma. Quando getopt() trova un'opzione, restituisce quel carattere ed aggiorna la variabile globale optind, che punta al prossimo parametro contenuto in argv. La stringa optstring indica quali sono le opzioni considerate. Se una opzione è seguita da ':' vuol dire che essa è seguita da un argomento. Tale argomento può essere ottenuto mediante la variabile globale optarg.

**Parametri riconosciuti**

La funzione riconosce i parametri:

- 'à' : seguito dall'indirizzo fisico della periferica con la quale interagire, il quale può essere indicato in esadecimale;
- 'w' : operazione di scrittura, seguito dal valore che si intende scrivere, in esadecimale; la scrittura verrà effettuata sul registro WRITE;
- 'm' : impostazione modalità, seguito dalla modalità col quale impostare il device; la scrittura verrà effettuata sul registro MODE;
- 'r' : operazione di lettura, primo di argomento; la lettura viene effettuata dal registro READ ed è non bloccante, nel senso che viene semplicemente letto il contenuto del registro.

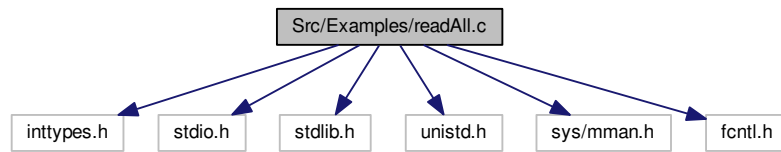
**Esempi:**

[noDriver.c](#).

**6.5 Riferimenti per il file Src/Examples/readAll.c**

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
```

Grafo delle dipendenze di inclusione per readAll.c:



## Funzioni

- void [howto](#) (void)
- int [parse\\_args](#) (int argc, char \*\*argv, uint32\_t \*gpio\_address, uint32\_t \*max\_offset)
- int [main](#) (int argc, char \*\*argv)

### 6.5.1 Documentazione delle funzioni

#### 6.5.1.1 void howto ( void )

Esempi:

[readAll.c](#).

#### 6.5.1.2 int main ( int argc, char \*\* argv )

Esempi:

[readAll.c](#).

#### 6.5.1.3 int parse\_args ( int argc, char \*\* argv, uint32\_t \* gpio\_address, uint32\_t \* max\_offset )

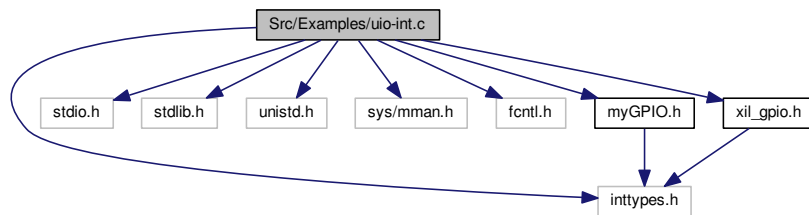
Esempi:

[readAll.c](#).

## 6.6 Riferimenti per il file Src/Examples/uioint.c

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "myGPIO.h"
#include "xil_gpio.h"
```

Grafo delle dipendenze di inclusione per uio-int.c:



## Funzioni

- void **howto** (void)  
*Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.*
- int **parse\_args** (int argc, char \*\*argv, char \*\*uio, uint8\_t \*op\_mode, uint32\_t \*mode\_value, uint8\_t \*op\_write, uint32\_t \*write\_value, uint8\_t \*op\_read)  
*Effettua il parsing dei parametri passati al programma.*
- void **gpio\_op** (void \*vrt\_gpio\_addr, int uio\_descriptor, uint8\_t op\_mode, uint32\_t mode\_value, uint8\_t op\_write, uint32\_t write\_value, uint8\_t op\_read)  
*Effettua operazioni su un device.*
- int **main** (int argc, char \*\*argv)  
*funzione **main()**.*

### 6.6.1 Documentazione delle funzioni

**6.6.1.1 void gpio\_op ( void \* vrt\_gpio\_addr, int uio\_descriptor, uint8\_t op\_mode, uint32\_t mode\_value, uint8\_t op\_write, uint32\_t write\_value, uint8\_t op\_read )**

Effettua operazioni su un device.

#### Parametri

in	<b>vrt_gpio_addr</b>	indirizzo di memoria del device gpio
in	<b>uio_descriptor</b>	descrittore del file /dev/uioX usato
in	<b>op_mode</b>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
in	<b>mode_value</b>	conterrà il valore che l'utente intende scrivere nel registro mode
in	<b>op_write</b>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
in	<b>write_value</b>	conterrà il valore che l'utente intende scrivere nel registro write
in	<b>op_read</b>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

La funzione viene invocata dopo che sia stato eseguito il parsing dei parametri passati al programma quando esso viene invocato. È stata scritta per funzionare sia con il GPIO Xilinx che con il GPIO custom myGPIO. È possibile utilizzare il primo definendo la macro **XIL\_GPIO**. Effettua, sul device, le operazioni impostate, in accordo con i parametri passati al programma alla sua invocazione.

#### Impostazione della modalità di funzionamento

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di impostazione della modalità di funzionamento del GPIO vengono effettuate scrivendo direttamente sul registro MODE del device. In caso contrario si è preferito utilizzare la funzione myGPIO\_setMode() (Si veda il modulo myGPIO). Funzionalmente non c'è differenza.

### Operazione di scrittura

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di scrittura del valore dei pin del device GPIO vengono effettuate scrivendo direttamente sul registro WRITE del device. In caso contrario si è preferito utilizzare la funzione `myGPIO_setValue()` (Si veda il modulo `myGPIO`). Funzionalmente non c'è differenza.

### Operazione di lettura con interrupt

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di lettura del valore dei pin del device GPIO vengono effettuate leggendo direttamente dal registro READ del device. In caso contrario si è preferito utilizzare la funzione `myGPIO_getRead()` (Si veda il modulo `myGPIO`). Funzionalmente non c'è differenza. **La lettura avviene usando il meccanismo delle interruzioni**

NOTA: la parte di codice per il GPIO Xilinx è stata scritta per hardware configurato in modo che il channel 1 del gpio fosse connessi esclusivamente ai led, mentre switch e button fossero connessi al channel 2 dello stesso GPIO. Il channel 1 ha dimensione 4 bit, mentre il channel 2 è da 8 bit.

### Attesa dell'arrivo di una interruzione

Gli interrupt sono gestiti effettuando una lettura bloccante su `/dev/uioX`. Una `read()` su `/dev/uioX` fa in modo che il processo venga sospeso ed inserito nella coda dei processi in attesa di un evento su quel file. Appena l'interrupt si manifesta, il processo viene posto nella coda dei processi pronti. La funzione `read()` consente di ottenere anche il numero totale di interrupt manifestatisi su quella particolare periferica. La `read()` restituisce il numero di interrupt che si sono manifestati. Quando un device possiede più di una sorgente di interrupt interna, ma non possiede maschere IRQ differenti o registri di stato differenti, potrebbe essere impossibile, per un programma in userspace, determinare quale sia la sorgente di interrupt se l'handler implementato nel kernel le disabilita scrivendo nei registri.

In questo caso è stato ritenuto opportuno, a titolo di esempio, mostrare come sia possibile bloccare il programma, dopo aver "servito" l'interruzione scatenata alla pressione di un tasto, fino a quando il tasto (o i tasti) premuti non siano riportati alla posizione di riposo. Lo stato del registro READ della periferica viene ripetutamente letto all'interno di un hot-loop, fino a quando non assume valore nullo. In tal caso si ha la certezza che i button o gli switch, in questo caso, siano stati riportati alla posizione di riposo. Si tenga presente che il device GPIO Xilinx genera una interruzione sia alla pressione che al rilascio di uno dei button o di uno degli switch

Dopo che button e switch siano stati riportati alla posizione di riposo, viene inviato l'ack al device, per segnalargli che l'interrupt è stato servito.

### Riabilitare gli interrupt UIO

Per lasciare inalterati i registri della periferica il kernel deve disabilitare completamente le interruzioni per la linea di interrupt cui la periferica è connessa, in modo che il programma userspace possa determinare la causa scatenante l'interruzione. Una volta terminate le operazioni, però, il programma userspace non può riabilitare le interruzioni, motivo per cui il driver implementa anche una funzione `write()`. La funzione `write()`, chiamata su `/dev/uioX`, consente di riabilitare le interruzioni per quella specifica periferica, scrivendo 1.

### Esempi:

[uio-int.c](#).

#### 6.6.1.2 void howto ( void )

Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.

### Esempi:

[uio-int.c](#).

### 6.6.1.3 int main ( int argc, char \*\* argv )

funzione [main\(\)](#).

#### Parsing dei parametri di invocazione

Il parsing dei parametri passati al programma all'atto della sua invocazione viene effettuato dalla funzione [parse\\_↵args\(\)](#). Si rimanda alla sua documentazione per i dettagli sui parametri riconosciuti.

Se non viene specificato il device UIO col quale interagire è impossibile continuare. Per questo motivo, in questo caso, il programma viene terminato.

#### Accesso ad un device /dev/uioX

Il driver generic-UIO è il driver generico per eccellenza. Ad ogni periferica compatibile con UIO è associato un file diverso in /dev/uioX attraverso il quale è possibile raggiungere il device. Tale file sarà /dev/uio0 per il primo device, /dev/uio1 per il secondo, /dev/uio2 per il terzo e così via. on for subsequent devices. Tale file può essere usato per accedere allo spazio degli indirizzi del device usando mmap().

In questo caso, rispetto all'esempio noDriver, accedere al device è estremamente più semplice. Se il device è compatibile con il driver UIO, è possibile "aprire" un file in /dev/uioX, effettuare il mapping, connettendo il device allo spazio di indirizzamento del processo, senza la necessità di conoscere l'indirizzo fisico della periferica col quale di intende comunicare.

L'accesso al device /dev/uioX viene ottenuto mediante la system-call open():

```
1 #include <sys/stat.h>
2 #include <fcntl.h>
3 int open(const char *path, int oflag, ... );
```

la quale restituisce il descrittore del file /dev/uioX, usato nel seguito per effettuare le operazioni di I/O. I valori del parametro oflag specificano il modo in cui il file /dev/uioX viene aperto. In questo caso viene usato O\_RDWR, il quale garantisce accesso in lettura ed in scrittura. Altri valori sono O\_RDONLY, il quale garantisce accesso in sola lettura, ed O\_WRONLY, che, invece, garantisce accesso in sola scrittura.

#### Mapping un device /dev/uioX

La "conversione" dell'indirizzo fisico del device in indirizzo virtuale appartenente allo spazio di indirizzamento del processo viene effettuato tramite la chiamata alla funzione mmap(), la quale stabilisce un mapping tra lo spazio di indirizzamento di un processo ed un file, una porzione di memoria condivisa o un qualsiasi altro memory-object, restituendo un indirizzo virtuale valido, attraverso il quale è possibile accedere al blocco di memoria fisico.

```
1 #include <sys/mman.h>
2 void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
```

Per semplicità supponiamo che la chiamata alla funzione sia la seguente:

```
pa=mmap(addr, len, prot, flags, fildes, off);
```

la semantica dei diversi parametri è:

- pa: indirizzo virtuale dell'address-space locale del processo, a cui viene eseguito il map; se il mapping ha successo viene restituito qualcosa di diverso da MAP\_FAILED;
- addr:
- len: lunghezza, in byte, del blocco mappato; in questo caso viene usato il valore restituito da sysconf(\_SC↵\_PAGESIZE);
- prot: specifica i permessi di accesso al blocco di memoria del quale si sta facendo il mapping;
  - PROT\_READ indica che il blocco può essere letto;

- PROT\_WRITE indica che il blocco può essere scritto;
- PROT\_NONE sta ad indicare che il blocco non può essere acceduto;
- flags: fornisce informazioni aggiuntive circa la gestione del blocco di dati di cui si sta facendo il mapping; il valore del flag può essere uno dei seguenti:
  - MAP\_SHARED: modifiche al blocco sono condivise con chiunque altri lo stia usando;
  - MAP\_PRIVATE: le modifiche sono private;
- fildes: descrittore del file /dev/mem
- off: indirizzo fisico del blocco che si intende mappare; è necessario che sia allineato alla dimensione della pagina di memoria, così come restituito dalla funzione sysconf(\_SC\_PAGESIZE);

In questo caso la chiamata a mmap avviene con i seguenti parametri:

```
1 uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
2 void* vrt_gpio_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, 0);
```

Rispetto al "driver" nodriver, la chiamata differisce per un solo particolare: essendo descriptor il descrittore di uioX, e l'offset specificato nullo, la funzione restituisce direttamente l'indirizzo virtuale del device nello spazio di indirizzamento del processo.

#### Operazioni sul device

Una volta effettuato il mapping, le operazioni preventivate con l'invocazione del programma vengono effettuate dalla funzione `gpio_op()`. Si rimanda alla sua documentazione per i dettagli sulle operazioni effettuate().

Esempi:

[uio-int.c](#).

**6.6.1.4** `int parse_args ( int argc, char ** argv, char ** uio, uint8_t * op_mode, uint32_t * mode_value, uint8_t * op_write, uint32_t * write_value, uint8_t * op_read )`

Effettua il parsing dei parametri passati al programma.

#### Parametri

in	<i>argc</i>	
in	<i>argv</i>	
out	<i>uio_file</i>	file uio da usare
out	<i>op_mode</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
out	<i>mode_value</i>	conterrà il valore che l'utente intende scrivere nel registro mode
out	<i>op_write</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
out	<i>write_value</i>	conterrà il valore che l'utente intende scrivere nel registro write
out	<i>op_read</i>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

#### Valori di ritorno

0	se il parsing ha successo
-1	se si verifica un errore

Parsing dei parametri del programma.

Il parsing viene effettuato usando la funzione `getopt()`.

```
1 #include <unistd.h>
2 int getopt(int argc, char * const argv[], const char *optstring);
```

Essa prende in input i parametri `argc` ed `argv` passati alla funzione `main()` quando il programma viene invocato. Quando una delle stringhe che compongono `argv` comincia con il carattere '-', `getopt()` la considera una opzione. Il carattere immediatamente successivo il '-' identifica la particolare opzione. La funzione può essere chiamata ripetutamente, fino a quando non restituisce -1, ad indicare che sono stati analizzati tutti i parametri passati al programma. Quando `getopt()` trova un'opzione, restituisce quel carattere ed aggiorna la variabile globale `optind`, che punta al prossimo parametro contenuto in `argv`. La stringa `optstring` indica quali sono le opzioni considerate. Se una opzione è seguita da ':' vuol dire che essa è seguita da un argomento. Tale argomento può essere ottenuto mediante la variabile globale `optarg`.

#### Parametri riconosciuti

La funzione riconosce i parametri:

- 'd' : seguito dal percorso del device /dev/uioX col quale interagire
- 'w' : operazione di scrittura, seguito dal valore che si intende scrivere, in esadecimale; la scrittura verrà effettuata sul registro WRITE;
- 'm' : impostazione modalità, seguito dalla modalità col quale impostare il device; la scrittura verrà effettuata sul registro MODE;
- 'r' : operazione di lettura, primo di argomento; la lettura viene effettuata dal registro READ ed è non bloccante, nel senso che viene semplicemente letto il contenuto del registro.

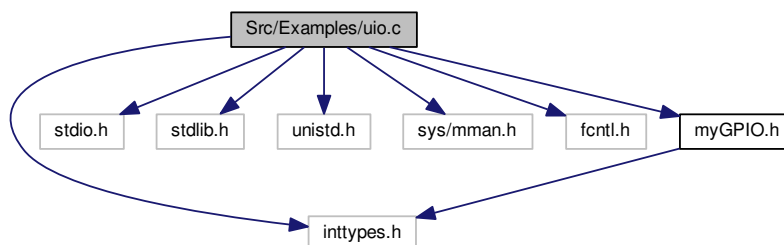
Esempi:

[uio-int.c](#).

## 6.7 Riferimenti per il file Src/Examples/uio.c

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per `uio.c`:



## Funzioni

- void [howto](#) (void)

*Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.*



- int [parse\\_args](#) (int argc, char \*\*argv, char \*\*uio, uint8\_t \*op\_mode, uint32\_t \*mode\_value, uint8\_t \*op\_write, uint32\_t \*write\_value, uint8\_t \*op\_read)  
*Effettua il parsing dei parametri passati al programma.*
- void [gpio\\_op](#) (void \*vrt\_gpio\_addr, uint8\_t op\_mode, uint32\_t mode\_value, uint8\_t op\_write, uint32\_t write\_value, uint8\_t op\_read)  
*Effettua operazioni su un device.*
- int [main](#) (int argc, char \*\*argv)  
*funzione [main\(\)](#).*

### 6.7.1 Documentazione delle funzioni

6.7.1.1 void [gpio\\_op](#) ( void \* [vrt\\_gpio\\_addr](#), uint8\_t [op\\_mode](#), uint32\_t [mode\\_value](#), uint8\_t [op\\_write](#), uint32\_t [write\\_value](#), uint8\_t [op\\_read](#) )

Effettua operazioni su un device.

#### Parametri

in	<a href="#">vrt_gpio_addr</a>	indirizzo di memoria del device gpio
in	<a href="#">op_mode</a>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
in	<a href="#">mode_value</a>	conterrà il valore che l'utente intende scrivere nel registro mode
in	<a href="#">op_write</a>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
in	<a href="#">write_value</a>	conterrà il valore che l'utente intende scrivere nel registro write
in	<a href="#">op_read</a>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

La funzione viene invocata dopo che sia stato eseguito il parsing dei parametri passati al programma quando esso viene invocato. È stata scritta per funzionare sia con il GPIO Xilinx che con il GPIO custom myGPIO. È possibile utilizzare il primo definendo la macro **XIL\_GPIO**. Effettua, sul device, le operazioni impostate, in accordo con i parametri passati al programma alla sua invocazione.

#### Impostazione della modalità di funzionamento

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di impostazione della modalità di funzionamento del GPIO vengono effettuate scrivendo direttamente sul registro MODE del device. In caso contrario si è preferito utilizzare la funzione [myGPIO\\_setMode\(\)](#) (Si veda il modulo myGPIO). Funzionalmente non c'è differenza.

#### Operazione di scrittura

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di scrittura del valore dei pin del device GPIO vengono effettuate scrivendo direttamente sul registro WRITE del device. In caso contrario si è preferito utilizzare la funzione [myGPIO\\_setValue\(\)](#) (Si veda il modulo myGPIO). Funzionalmente non c'è differenza.

#### Operazione di lettura

Nel caso in cui si stia operando su un device GPIO Xilinx, le operazioni di lettura del valore dei pin del device GPIO vengono effettuate leggendo direttamente dal registro READ del device. In caso contrario si è preferito utilizzare la funzione [myGPIO\\_getRead\(\)](#) (Si veda il modulo myGPIO). Funzionalmente non c'è differenza. La lettura è non bloccante: viene semplicemente letto il valore contenuto nel registro. UIO permette l'implementazione di meccanismi di lettura basati su interruzione, ma in questo caso, per semplicità, tale meccanismo è stato omesso. Si veda il modulo UIO-interrupt.

#### Esempi:

[uio.c](#).

6.7.1.2 void [howto](#) ( void )

Stampa un messaggio che fornisce indicazioni sull'utilizzo del programma.

Esempi:

[uio.c](#).

6.7.1.3 `int main ( int argc, char ** argv )`

funzione [main\(\)](#).

#### Parsing dei parametri di invocazione

Il parsing dei parametri passati al programma all'atto della sua invocazione viene effettuato dalla funzione [parse\\_↔args\(\)](#). Si rimanda alla sua documentazione per i dettagli sui parametri riconosciuti.

Se non viene specificato il device UIO col quale interagire è impossibile continuare. Per questo motivo, in questo caso, il programma viene terminato.

#### Accesso ad un device /dev/uioX

Il driver generic-UIO è il driver generico per eccellenza. Ad ogni periferica compatibile con UIO è associato un file diverso in /dev/uioX attraverso il quale è possibile raggiungere il device. Tale file sarà /dev/uio0 per il primo device, /dev/uio1 per il secondo, /dev/uio2 per il terzo e così via. on for subsequent devices. Tale file può essere usato per accedere allo spazio degli indirizzi del device usando mmap().

In questo caso, rispetto all'esempio noDriver, accedere al device è estremamente più semplice. Se il device è compatibile con il driver UIO, è possibile "aprire" un file in /dev/uioX, effettuare il mapping, connettendo il device allo spazio di indirizzamento del processo, senza la necessità di conoscere l'indirizzo fisico della periferica col quale di intende comunicare.

L'accesso al device /dev/uioX viene ottenuto mediante la system-call open():

```
1 #include <sys/stat.h>
2 #include <fcntl.h>
3 int open(const char *path, int oflag, ... );
```

la quale restituisce il descrittore del file /dev/uioX, usato nel seguito per effettuare le operazioni di I/O. I valori del parametro oflag specificano il modo in cui il file /dev/uioX viene aperto. In questo caso viene usato O\_RDWR, il quale garantisce accesso in lettura ed in scrittura. Altri valori sono O\_RDONLY, il quale garantisce accesso in sola lettura, ed O\_WRONLY, che, invece, garantisce accesso in sola scrittura.

#### Mapping un device /dev/uioX

La "conversione" dell'indirizzo fisico del device in indirizzo virtuale appartenente allo spazio di indirizzamento del processo viene effettuato tramite la chiamata alla funzione mmap(), la quale stabilisce un mapping tra lo spazio di indirizzamento di un processo ed un file, una porzione di memoria condivisa o un qualsiasi altro memory-object, restituendo un indirizzo virtuale valido, attraverso il quale è possibile accedere al blocco di memoria fisico.

```
1 #include <sys/mman.h>
2 void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
```

Per semplicità supponiamo che la chiamata alla funzione sia la seguente:

```
pa=mmap(addr, len, prot, flags, fildes, off);
```

la semantica dei diversi parametri è:

- pa: indirizzo virtuale dell'address-space locale del processo, a cui viene eseguito il map; se il mapping ha successo viene restituito qualcosa di diverso da MAP\_FAILED;
- addr:
- len: lunghezza, in byte, del blocco mappato; in questo caso viene usato il valore restituito da sysconf(\_SC↔\_PAGESIZE);

- **prot:** specifica i permessi di accesso al blocco di memoria del quale si sta facendo il mapping;
  - PROT\_READ indica che il blocco può essere letto;
  - PROT\_WRITE indica che il blocco può essere scritto;
  - PROT\_NONE sta ad indicare che il blocco non può essere acceduto;
- **flags:** fornisce informazioni aggiuntive circa la gestione del blocco di dati di cui si sta facendo il mapping; il valore del flag può essere uno dei seguenti:
  - MAP\_SHARED: modifiche al blocco sono condivise con chiunque altri lo stia usando;
  - MAP\_PRIVATE: le modifiche sono private;
- **filedes:** descrittore del file /dev/mem
- **off:** indirizzo fisico del blocco che si intende mappare; è necessario che sia allineato alla dimensione della pagina di memoria, così come restituito dalla funzione sysconf(\_SC\_PAGESIZE);

In questo caso la chiamata a mmap avviene con i seguenti parametri:

```
1 uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
2 void* vrt_gpio_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, 0);
```

Rispetto al "driver" nodriver, la chiamata differisce per un solo particolare: essendo descriptor il descrittore di uioX, e l'offset specificato nullo, la funzione restituisce direttamente l'indirizzo virtuale del device nello spazio di indirizzamento del processo.

#### Operazioni sul device

Una volta effettuato il mapping, le operazioni preventivate con l'invocazione del programma vengono effettuate dalla funzione `gpio_op()`. Si rimanda alla sua documentazione per i dettagli sulle operazioni effettuate().

#### Esempi:

[uio.c](#).

**6.7.1.4** `int parse_args ( int argc, char ** argv, char ** uio, uint8_t * op_mode, uint32_t * mode_value, uint8_t * op_write, uint32_t * write_value, uint8_t * op_read )`

Effettua il parsing dei parametri passati al programma.

#### Parametri

in	<i>argc</i>	
in	<i>argv</i>	
out	<i>uio_file</i>	file uio da usare
out	<i>op_mode</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su mode
out	<i>mode_value</i>	conterrà il valore che l'utente intende scrivere nel registro mode
out	<i>op_write</i>	sarà impostato ad 1 se l'utente intende effettuare scrittura su write
out	<i>write_value</i>	conterrà il valore che l'utente intende scrivere nel registro write
out	<i>op_read</i>	sarà impostato ad 1 se l'utente intende effettuare lettura da read

#### Valori di ritorno

<i>0</i>	se il parsing ha successo
----------	---------------------------

-1	se si verifica un errore
----	--------------------------

### Parsing dei parametri del programma.

Il parsing viene effettuato usando la funzione `getopt()`.

```
1 #include <unistd.h>
2 int getopt(int argc, char * const argv[], const char *optstring);
```

Essa prende in input i parametri `argc` ed `argv` passati alla funzione `main()` quando il programma viene invocato. Quando una delle stringhe che compongono `argv` comincia con il carattere '-', `getopt()` la considera una opzione. Il carattere immediatamente successivo il '-' identifica la particolare opzione. La funzione può essere chiamata ripetutamente, fino a quando non restituisce -1, ad indicare che sono stati analizzati tutti i parametri passati al programma. Quando `getopt()` trova un'opzione, restituisce quel carattere ed aggiorna la variabile globale `optind`, che punta al prossimo parametro contenuto in `argv`. La stringa `optstring` indica quali sono le opzioni considerate. Se una opzione è seguita da ':' vuol dire che essa è seguita da un argomento. Tale argomento può essere ottenuto mediante la variabile globale `optarg`.

### Parametri riconosciuti

La funzione riconosce i parametri:

- 'd' : seguito dal percorso del device /dev/uioX col quale interagire
- 'w' : operazione di scrittura, seguito dal valore che si intende scrivere, in esadecimale; la scrittura verrà effettuata sul registro WRITE;
- 'm' : impostazione modalità, seguito dalla modalità col quale impostare il device; la scrittura verrà effettuata sul registro MODE;
- 'r' : operazione di lettura, primo di argomento; la lettura viene effettuata dal registro READ ed è non bloccante, nel senso che viene semplicemente letto il contenuto del registro.

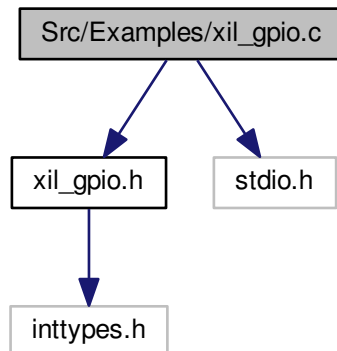
### Esempi:

[uio.c](#).

## 6.8 Riferimenti per il file Src/Examples/xil\_gpio.c

```
#include "xil_gpio.h"
#include <stdio.h>
```

Grafo delle dipendenze di inclusione per xil\_gpio.c:



## Funzioni

- int [XilGpio\\_Global\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t mask)
- int [XilGpio\\_Channel\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t mask)
- int [XilGpio\\_Ack\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t channel)

### 6.8.1 Documentazione delle funzioni

6.8.1.1 int [XilGpio\\_Ack\\_Interrupt](#) ( uint32\_t \* *baseAddress*, uint32\_t *channel* )

Esempi:

[uio-int.c](#).

6.8.1.2 int [XilGpio\\_Channel\\_Interrupt](#) ( uint32\_t \* *baseAddress*, uint32\_t *mask* )

Esempi:

[uio-int.c](#).

6.8.1.3 int [XilGpio\\_Global\\_Interrupt](#) ( uint32\_t \* *baseAddress*, uint32\_t *mask* )

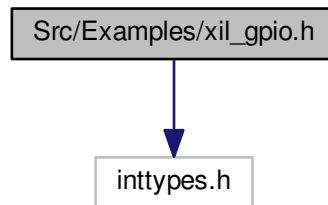
Esempi:

[uio-int.c](#).

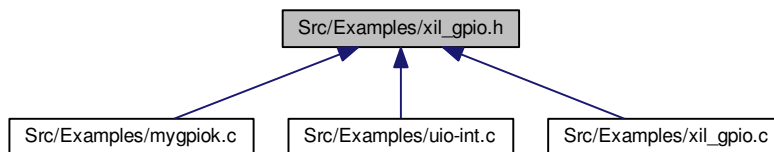
## 6.9 Riferimenti per il file Src/Examples/xil\_gpio.h

```
#include <inttypes.h>
```

Grafo delle dipendenze di inclusione per xil\_gpio.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



### Definizioni

- `#define XGPIO_GIE_GINTR_ENABLE_MASK` 0x80000000
- `#define GPIO_DATA_OFFSET` 0x00
- `#define GPIO_TRI_OFFSET` 0x04
- `#define GPIO_READ_OFFSET` 0x08
- `#define XGPIO_GIE_OFFSET` 0x11C
- `#define XGPIO_ISR_OFFSET` 0x120
- `#define XGPIO_IER_OFFSET` 0x128
- `#define GLOBAL_INTR_ENABLE XGPIO_GIE_GINTR_ENABLE_MASK`
- `#define CHANNEL1_INTR_ENABLE` 0x00000001
- `#define CHANNEL2_INTR_ENABLE` 0x00000002
- `#define GLOBAL_INTR_DISABLE` 0x00000000
- `#define CHANNEL1_INTR_DISABLE` 0x00000000
- `#define CHANNEL2_INTR_DISABLE` 0x00000000
- `#define CHANNEL1_ACK` 0x01
- `#define CHANNEL2_ACK` 0x02
- `#define GPIO_MAP_SIZE` 0x10000

## Funzioni

- int [XilGpio\\_Global\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t mask)
- int [XilGpio\\_Channel\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t mask)
- int [XilGpio\\_Ack\\_Interrupt](#) (uint32\_t \*baseAddress, uint32\_t channel)

### 6.9.1 Documentazione delle definizioni

6.9.1.1 `#define CHANNEL1_ACK 0x01`

6.9.1.2 `#define CHANNEL1_INTR_DISABLE 0x00000000`

6.9.1.3 `#define CHANNEL1_INTR_ENABLE 0x00000001`

6.9.1.4 `#define CHANNEL2_ACK 0x02`

Esempi:

[uio-int.c](#).

6.9.1.5 `#define CHANNEL2_INTR_DISABLE 0x00000000`

Esempi:

[uio-int.c](#).

6.9.1.6 `#define CHANNEL2_INTR_ENABLE 0x00000002`

Esempi:

[uio-int.c](#).

6.9.1.7 `#define GLOBAL_INTR_DISABLE 0x00000000`

Esempi:

[uio-int.c](#).

6.9.1.8 `#define GLOBAL_INTR_ENABLE XGPIO_GIE_GINTR_ENABLE_MASK`

Esempi:

[uio-int.c](#).

6.9.1.9 `#define GPIO_DATA_OFFSET 0x00`

Esempi:

[uio-int.c](#).

6.9.1.10 `#define GPIO_MAP_SIZE 0x10000`

6.9.1.11 `#define GPIO_READ_OFFSET 0x08`

Esempi:

[uio-int.c](#).

6.9.1.12 `#define GPIO_TRI_OFFSET 0x04`

Esempi:

[uio-int.c](#).

6.9.1.13 `#define XGPIO_GIE_GINTR_ENABLE_MASK 0x80000000`

6.9.1.14 `#define XGPIO_GIE_OFFSET 0x11C`

Global interrupt enable register

6.9.1.15 `#define XGPIO_IER_OFFSET 0x128`

Interrupt enable register

6.9.1.16 `#define XGPIO_ISR_OFFSET 0x120`

Interrupt status register

## 6.9.2 Documentazione delle funzioni

6.9.2.1 `int XilGpio_Ack_Interrupt ( uint32_t * baseAddress, uint32_t channel )`

6.9.2.2 `int XilGpio_Channel_Interrupt ( uint32_t * baseAddress, uint32_t mask )`

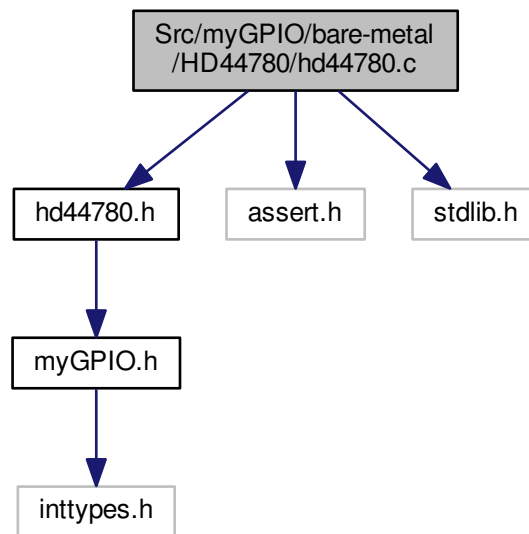
6.9.2.3 `int XilGpio_Global_Interrupt ( uint32_t * baseAddress, uint32_t mask )`

## 6.10 Riferimenti per il file Src/myGPIO/bare-metal/HD44780/hd44780.c

```
#include "hd44780.h"
#include <assert.h>
#include <stdlib.h>
```



Grafo delle dipendenze di inclusione per hd44780.c:



## Definizioni

- #define HD44780\_clear 0x01
- #define HD44780\_home 0x02
- #define HD44780\_row1 0x80
- #define HD44780\_row2 0xC0
- #define HD44780\_cursor\_r 0x14
- #define HD44780\_cursor\_l 0x10
- #define HD44780\_display\_off 0x08
- #define HD44780\_cursor\_off 0x0C
- #define HD44780\_cursor\_on 0x0E
- #define HD44780\_cursor\_blink 0x0F
- #define HD44780\_clear 0x01
- #define HD44780\_dec\_no\_shift 0x04
- #define HD44780\_dec\_shift 0x05
- #define HD44780\_inc\_no\_shift 0x06
- #define HD44780\_inc\_shift 0x07
- #define timer\_wait\_ms(ms) usleep(ms<<10)
- #define timer\_wait\_us(us) usleep(us)
- #define lcd\_command(lcd) myGPIO\_setValue(lcd->gpio, lcd->RS, myGPIO\_reset)
- #define lcd\_data(lcd) myGPIO\_setValue(lcd->gpio, lcd->RS, myGPIO\_set)
- #define lcd\_write(lcd) myGPIO\_setValue(lcd->gpio, lcd->RW, myGPIO\_reset)
- #define lcd\_read(lcd) myGPIO\_setValue(lcd->gpio, lcd->RW, myGPIO\_set)
- #define lcd\_enable(lcd)

## Funzioni

- void [HD44780\\_SetByte](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t byte)
- void [HD44780\\_WriteCommand](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t command)
- void [HD44780\\_WriteData](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t data)
- int [HD44780\\_ValidatePair](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- void [HD44780\\_ConfigurePin](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- void [HD44780\\_Init8](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4, [myGPIO\\_mask](#) Data3, [myGPIO\\_mask](#) Data2, [myGPIO\\_mask](#) Data1, [myGPIO\\_mask](#) Data0)
- Inizializza un display lcd HD44780 con interfacciamento ad 8 bit.*
- void [HD44780\\_Init4](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4)
- Inizializza un oggetto display lcd HD44780 affinché si utilizzi l'interfaccia a 4 bit.*
- void [HD44780\\_Printc](#) ([HD44780\\_LCD\\_t](#) \*lcd, char c)
- Stampa un carattere.*
- void [HD44780\\_Print](#) ([HD44780\\_LCD\\_t](#) \*lcd, const char \*s)
- Stampa una stringa null-terminated di caratteri.*
- void [HD44780\\_printBinary8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)
- Stampa un byte in binario. (bit più significativo a sinistra)*
- void [HD44780\\_printBinary32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)
- Stampa una word di 32 bit in binario. (bit più significativo a sinistra)*
- void [HD44780\\_printBinary64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)
- Stampa un blocco di 64 bit in binario. (bit più significativo a sinistra)*
- void [HD44780\\_printHex8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)
- Stampa un byte in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_printHex32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)
- Stampa una word di 32 bit in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_printHex64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)
- Stampa un blocco di 64 bit in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_Clear](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Pulisce il display e sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_Home](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_MoveToRow1](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_MoveToRow2](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Sposta il cursore all'inizio della seconda riga.*
- void [HD44780\\_MoveCursor](#) ([HD44780\\_LCD\\_t](#) \*lcd, [HD44780\\_Direction\\_t](#) dir)
- Sposta il cursore di una posizione a destra o sinistra.*
- void [HD44780\\_DisplayOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Disattiva il display.*
- void [HD44780\\_CursorOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Disattiva la visualizzazione del cursore.*
- void [HD44780\\_CursorOn](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Attiva la visualizzazione del cursore.*
- void [HD44780\\_CursorBlink](#) ([HD44780\\_LCD\\_t](#) \*lcd)
- Attiva il cursore lampeggiante.*

### 6.10.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

### 6.10.2 Documentazione delle definizioni

6.10.2.1 `#define HD44780_clear 0x01`

6.10.2.2 `#define HD44780_clear 0x01`

6.10.2.3 `#define HD44780_cursor_blink 0x0F`

6.10.2.4 `#define HD44780_cursor_l 0x10`

6.10.2.5 `#define HD44780_cursor_off 0x0C`

6.10.2.6 `#define HD44780_cursor_on 0x0E`

6.10.2.7 `#define HD44780_cursor_r 0x14`

6.10.2.8 `#define HD44780_dec_no_shift 0x04`

6.10.2.9 `#define HD44780_dec_shift 0x05`

6.10.2.10 `#define HD44780_display_off 0x08`

6.10.2.11 `#define HD44780_home 0x02`

6.10.2.12 `#define HD44780_inc_no_shift 0x06`

6.10.2.13 `#define HD44780_inc_shift 0x07`

6.10.2.14 `#define HD44780_row1 0x80`

6.10.2.15 `#define HD44780_row2 0xC0`

6.10.2.16 `#define lcd_command( lcd ) myGPIO_setValue(lcd->gpio, lcd->RS, myGPIO_reset)`

6.10.2.17 `#define lcd_data( lcd ) myGPIO_setValue(lcd->gpio, lcd->RS, myGPIO_set)`

#### 6.10.2.18 `#define lcd_enable( lcd )`

**Valore:**

```
myGPIO_setValue(lcd->gpio, lcd->E, myGPIO_set); \
    timer_wait_us(100); \
    myGPIO_setValue(lcd->gpio, lcd->E, myGPIO_reset)
```

#### 6.10.2.19 `#define lcd_read( lcd ) myGPIO_setValue(lcd->gpio, lcd->RW, myGPIO_set)`

#### 6.10.2.20 `#define lcd_write( lcd ) myGPIO_setValue(lcd->gpio, lcd->RW, myGPIO_reset)`

#### 6.10.2.21 `#define timer_wait_ms( ms ) usleep(ms<<10)`

#### 6.10.2.22 `#define timer_wait_us( us ) usleep(us)`

### 6.10.3 Documentazione delle funzioni

#### 6.10.3.1 `void HD44780_ConfigurePin ( HD44780_LCD_t * lcd )`

#### 6.10.3.2 `void HD44780_SetByte ( HD44780_LCD_t * lcd, uint8_t byte )`

#### 6.10.3.3 `int HD44780_ValidatePair ( HD44780_LCD_t * lcd )`

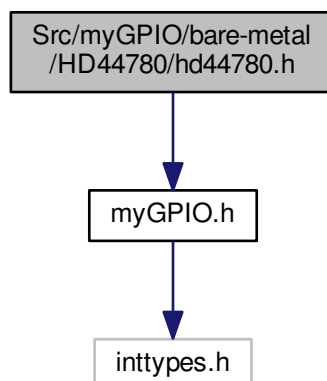
#### 6.10.3.4 `void HD44780_WriteCommand ( HD44780_LCD_t * lcd, uint8_t command )`

#### 6.10.3.5 `void HD44780_WriteData ( HD44780_LCD_t * lcd, uint8_t data )`

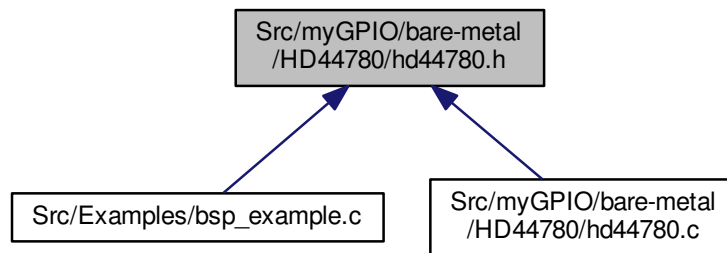
## 6.11 Riferimenti per il file `Src/myGPIO/bare-metal/HD44780/hd44780.h`

```
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per `hd44780.h`:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct [HD44780\\_LCD\\_t](#)

Struttura opaca che astrae un device Display LCD con controller Hitachi HD44780, o compatibile. Un oggetto di tipo [HD44780\\_LCD\\_t](#) rappresenta un device lcd HD44780. Il modulo è pensato per permettere la gestione di più display da parte dello stesso processore, agendo su oggetti [HD44780\\_LCD\\_t](#) diversi. Il modulo permette di utilizzare sia l'interfacciamento ad otto bit che quello a quattro bit, inizializzando il device opportunamente, attraverso l'uso delle funzioni [HD44780\\_Init8](#) e [HD44780\\_Init4](#). Il modulo fornisce anche semplici funzioni per la stampa di un carattere o di una stringa null-terminated di caratteri. Si veda la documentazione delle funzioni [HD44780\\_Printc\(\)](#) e [HD44780\\_Print\(\)](#). Inoltre sono presenti diverse funzioni di utilità generica, come quelle per la pulizia del display, per lo spostamento del cursore di un posto in avanti o indietro, alla riga in basso o in alto.

## Tipi enumerati (enum)

- enum [HD44780\\_InterfaceMode\\_t](#) { [HD44780\\_INTERFACE\\_4bit](#), [HD44780\\_INTERFACE\\_8bit](#) }

Modalità di interfacciamento. Il modulo supporta sia interfacciamento a 4 bit che ad 8 bit.

- enum [HD44780\\_Direction\\_t](#) { [HD44780\\_CursorLeft](#), [HD44780\\_CursorRight](#) }

Direzioni di spostamento del cursore, usata dalla funzione [HD44780\\_MoveCursor\(\)](#)

## Funzioni

- void [HD44780\\_Init8](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4, [myGPIO\\_mask](#) Data3, [myGPIO\\_mask](#) Data2, [myGPIO\\_mask](#) Data1, [myGPIO\\_mask](#) Data0)

Inizializza un display lcd HD44780 con interfacciamento ad 8 bit.

- void [HD44780\\_Init4](#) ([HD44780\\_LCD\\_t](#) \*lcd, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) RS, [myGPIO\\_mask](#) RW, [myGPIO\\_mask](#) E, [myGPIO\\_mask](#) Data7, [myGPIO\\_mask](#) Data6, [myGPIO\\_mask](#) Data5, [myGPIO\\_mask](#) Data4)

Inizializza un oggetto display lcd HD44780 affinché si utilizzi l'interfaccia a 4 bit.

- void [HD44780\\_Printc](#) ([HD44780\\_LCD\\_t](#) \*lcd, char c)

Stampa un carattere.

- void [HD44780\\_Print](#) ([HD44780\\_LCD\\_t](#) \*lcd, const char \*s)

Stampa una stringa null-terminated di caratteri.

- void [HD44780\\_printBinary8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)

Stampa un byte in binario. (bit più significativo a sinistra)

- void [HD44780\\_printBinary32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)

Stampa una word di 32 bit in binario. (bit più significativo a sinistra)

- void [HD44780\\_printBinary64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)  
*Stampa un blocco di 64 bit in binario. (bit più significativo a sinistra)*
- void [HD44780\\_printHex8](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint8\_t b)  
*Stampa un byte in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_printHex32](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint32\_t w)  
*Stampa una word di 32 bit in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_printHex64](#) ([HD44780\\_LCD\\_t](#) \*lcd, uint64\_t b)  
*Stampa un blocco di 64 bit in esadecimale. (bit più significativo a sinistra)*
- void [HD44780\\_Clear](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Pulisce il display e sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_Home](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_MoveToRow1](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Sposta il cursore all'inizio della prima riga.*
- void [HD44780\\_MoveToRow2](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Sposta il cursore all'inizio della seconda riga.*
- void [HD44780\\_MoveCursor](#) ([HD44780\\_LCD\\_t](#) \*lcd, [HD44780\\_Direction\\_t](#) dir)  
*Sposta il cursore di una posizione a destra o sinistra.*
- void [HD44780\\_DisplayOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Disattiva il display.*
- void [HD44780\\_CursorOff](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Disattiva la visualizzazione del cursore.*
- void [HD44780\\_CursorOn](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Attiva la visualizzazione del cursore.*
- void [HD44780\\_CursorBlink](#) ([HD44780\\_LCD\\_t](#) \*lcd)  
*Attiva il cursore lampeggiante.*

### 6.11.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

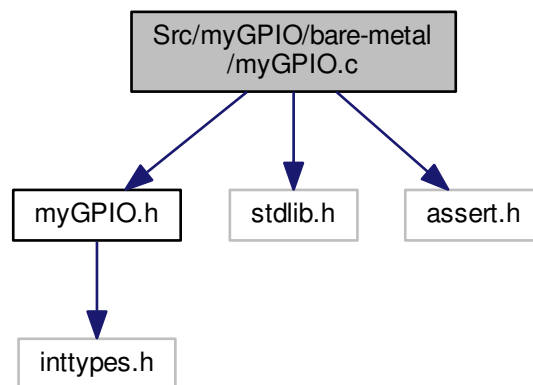
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.12 Riferimenti per il file Src/myGPIO/bare-metal/myGPIO.c

```
#include "myGPIO.h"
#include <stdlib.h>
#include <assert.h>
```

Grafo delle dipendenze di inclusione per myGPIO.c:



### Funzioni

- void **myGPIO\_Init** (myGPIO\_t \*gpio, uint32\_t base\_address)  
*Inizializza un device myGPIO.*
- void **myGPIO\_SetMode** (myGPIO\_t \*gpio, myGPIO\_mask mask, myGPIO\_mode mode)  
*Permette di settare la modalità lettura/scrittura dei pin di un device myGPIO;.*
- void **myGPIO\_SetValue** (myGPIO\_t \*gpio, myGPIO\_mask mask, myGPIO\_value value)  
*Permette di settare il valore dei pin di un device myGPIO, se configurati come output.*
- void **myGPIO\_Toggle** (myGPIO\_t \*gpio, myGPIO\_mask mask)  
*Permette di invertire il valore dei pin di un device myGPIO, se configurati come output.*
- myGPIO\_value myGPIO\_GetValue (myGPIO\_t \*gpio, myGPIO\_mask mask)  
*Permette di leggere il valore dei pin di un device myGPIO;.*
- myGPIO\_mask myGPIO\_GetRead (myGPIO\_t \*gpio)  
*Restituisce la maschera dei pin settati di un device myGPIO.*
- void **myGPIO\_GlobalInterruptEnable** (myGPIO\_t \*gpio)  
*Abilita gli interrupt globali;.*
- void **myGPIO\_GlobalInterruptDisable** (myGPIO\_t \*gpio)  
*Disabilita gli interrupt globali;.*
- myGPIO\_value myGPIO\_IsGlobalInterruptEnabled (myGPIO\_t \*gpio)  
*Consente di verificare se gli interrupt globali siano abilitati.*
- myGPIO\_value myGPIO\_PendingInterrupt (myGPIO\_t \*gpio)  
*Consente di verificare se esistano interrupt non ancora serviti.*
- void **myGPIO\_PinInterruptEnable** (myGPIO\_t \*gpio, myGPIO\_mask mask)  
*Abilita gli interrupt per i singoli pin del device.*
- void **myGPIO\_PinInterruptDisable** (myGPIO\_t \*gpio, myGPIO\_mask mask)  
*Disabilita gli interrupt per i singoli pin del device.*

- `myGPIO_mask myGPIO_EnabledPinInterrupt (myGPIO_t *gpio)`  
*Consente di ottenere una maschera che indichi quali pin abbiano interrupt abilitati.*
- `myGPIO_mask myGPIO_PendingPinInterrupt (myGPIO_t *gpio)`  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- `void myGPIO_PinInterruptAck (myGPIO_t *gpio, myGPIO_mask mask)`  
*Invia al device notifica di servizio di un interrupt;.*

### 6.12.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

12 05 2017

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

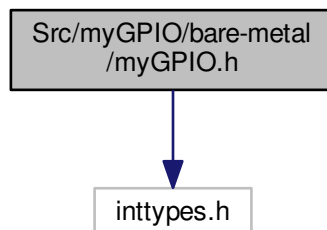
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.13 Riferimenti per il file Src/myGPIO/bare-metal/myGPIO.h

```
#include <inttypes.h>
```

Grafo delle dipendenze di inclusione per myGPIO.h:







*myGPIO\_mode*, modalità di funzionamento (lettura/scrittura) di un device *myGPIO*

- enum *myGPIO\_value* { *myGPIO\_reset*, *myGPIO\_set* }

*myGPIO\_value*, valore di un *myGPIO*

## Funzioni

- void *myGPIO\_Init* (*myGPIO\_t* \*gpio, uint32\_t base\_address)  
*Inizializza un device myGPIO.*
- void *myGPIO\_SetMode* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask, *myGPIO\_mode* mode)  
*Permette di settare la modalità lettura/scrittura dei pin di un device myGPIO;.*
- void *myGPIO\_SetValue* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask, *myGPIO\_value* value)  
*Permette di settare il valore dei pin di un device myGPIO, se configurati come output.*
- void *myGPIO\_Toggle* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask)  
*Permette di invertire il valore dei pin di un device myGPIO, se configurati come output.*
- *myGPIO\_value* *myGPIO\_GetValue* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask)  
*Permette di leggere il valore dei pin di un device myGPIO;.*
- *myGPIO\_mask* *myGPIO\_GetRead* (*myGPIO\_t* \*gpio)  
*Restituisce la maschera dei pin settati di un device myGPIO.*
- void *myGPIO\_GlobalInterruptEnable* (*myGPIO\_t* \*gpio)  
*Abilita gli interrupt globali;.*
- void *myGPIO\_GlobalInterruptDisable* (*myGPIO\_t* \*gpio)  
*Disabilita gli interrupt globali;.*
- *myGPIO\_value* *myGPIO\_IsGlobalInterruptEnabled* (*myGPIO\_t* \*gpio)  
*Consente di verificare se gli interrupt globali siano abilitati.*
- *myGPIO\_value* *myGPIO\_PendingInterrupt* (*myGPIO\_t* \*gpio)  
*Consente di verificare se esistano interrupt non ancora serviti.*
- void *myGPIO\_PinInterruptEnable* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask)  
*Abilita gli interrupt per i singoli pin del device.*
- void *myGPIO\_PinInterruptDisable* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask)  
*Disabilita gli interrupt per i singoli pin del device.*
- *myGPIO\_mask* *myGPIO\_EnabledPinInterrupt* (*myGPIO\_t* \*gpio)  
*Consente di ottenere una maschera che indichi quali pin abbiano interrupt abilitati.*
- *myGPIO\_mask* *myGPIO\_PendingPinInterrupt* (*myGPIO\_t* \*gpio)  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- void *myGPIO\_PinInterruptAck* (*myGPIO\_t* \*gpio, *myGPIO\_mask* mask)  
*Invia al device notifica di servizio di un interrupt;.*

### 6.13.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

12 05 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

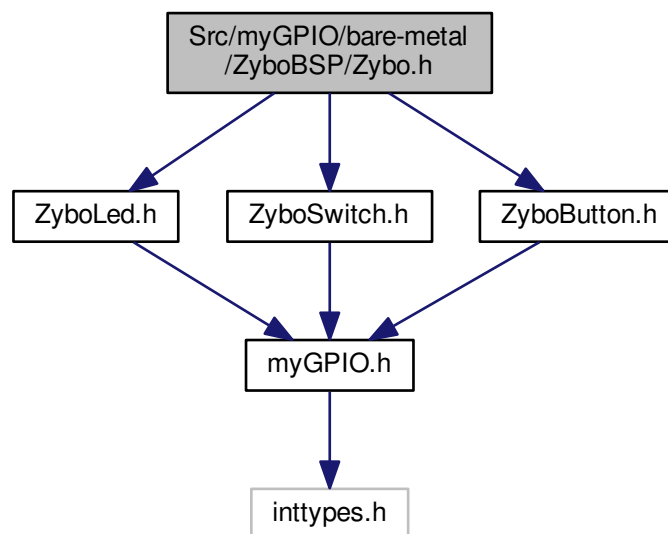
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.14 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/Zybo.h

```
#include "ZyboLed.h"  
#include "ZyboSwitch.h"  
#include "ZyboButton.h"
```

Grafo delle dipendenze di inclusione per Zybo.h:



### 6.14.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

## Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

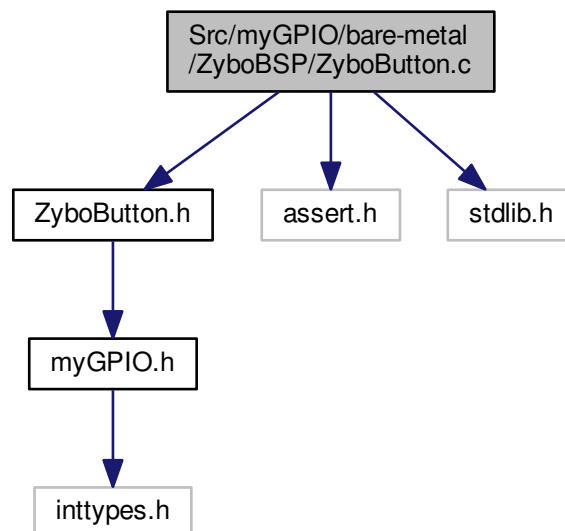
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.15 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboButton.c

```
#include "ZyboButton.h"
#include <assert.h>
#include <stdlib.h>
```

Grafo delle dipendenze di inclusione per ZyboButton.c:



## Definizioni

- #define [timer\\_wait\\_ms](#)(ms) usleep(ms<<10)

## Funzioni

- static int [validatePair](#) ([ZyboButton\\_t](#) \*buttons)
- void [ZyboButton\\_init](#) ([ZyboButton\\_t](#) \*buttons, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) Button3\_pin, [myGPIO\\_mask](#) Button2\_pin, [myGPIO\\_mask](#) Button1\_pin, [myGPIO\\_mask](#) Button0\_pin)

Inizializza un oggetto di tipo `ZyboButton_t`.

- void `ZyboButton_waitWhileIdle` (`ZyboButton_t` \*buttons)

Permettere di mettere il programma in attesa attiva finché i button restano inattivi;.

- void `ZyboButton_waitWhileBusy` (`ZyboButton_t` \*buttons)

Permettere di mettere il programma in attesa attiva finché i button restano attivi;.

- `ZyboButton_status_t` `ZyboButton_getStatus` (`ZyboButton_t` \*buttons, `ZyboButton_mask_t` mask)

Permette la lettura dello stato dei button presenti sulla board.

### 6.15.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

### 6.15.2 Documentazione delle definizioni

6.15.2.1 `#define timer_wait_ms( ms ) usleep(ms<<10)`

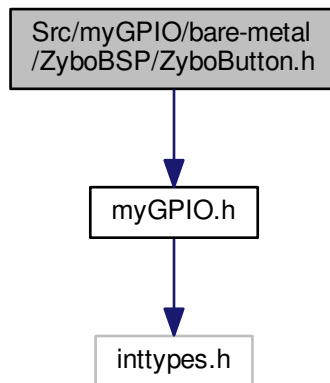
### 6.15.3 Documentazione delle funzioni

6.15.3.1 `static int validatePair ( ZyboButton_t * buttons ) [static]`

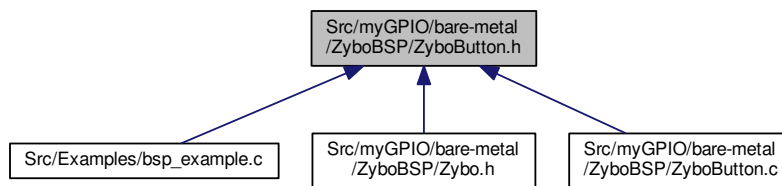
## 6.16 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboButton.h

```
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per ZyboButton.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct [ZyboButton\\_t](#)

*Struttura opaca che astrae l'insieme dei button presenti sulla board Diligent Zybo.*

## Definizioni

- #define [ZyboButton\(i\)](#) ((uint32\_t)(1<<(i)))

*Metodo alternativo per la specifica di uno dei button presenti sulla board Diligent Zybo.*

- #define [ZyboButton\\_DebounceWait](#) 50

*Tempo di attesa (in millisecondi) usato per prevenire il fenomeno del bouncing. Il valore di default è 50, determinato empiricamente. Può essere modificato a piacimento cambiando il valore alla macro seguente.*

## Tipi enumerati (enum)

- enum [ZyboButton\\_mask\\_t](#) { [ZyboButton3](#) = 0x8U, [ZyboButton2](#) = 0x4U, [ZyboButton1](#) = 0x2U, [ZyboButton0](#) = 0x1U }

*Maschere di selezione dei PushButton.*

- enum `ZyboButton_status_t` { `ZyboButton_off`, `ZyboButton_on` }

*Status di attivo/inattivo dei PushButton.*

## Funzioni

- void `ZyboButton_init` (`ZyboButton_t` \*buttons, `myGPIO_t` \*gpio, `myGPIO_mask` Button3\_pin, `myGPIO_mask` Button2\_pin, `myGPIO_mask` Button1\_pin, `myGPIO_mask` Button0\_pin)

*Inizializza un oggetto di tipo `ZyboButton_t`.*

- void `ZyboButton_waitWhileIdle` (`ZyboButton_t` \*buttons)

*Permettere di mettere il programma in attesa attiva finché i button restano inattivi;.*

- void `ZyboButton_waitWhileBusy` (`ZyboButton_t` \*buttons)

*Permettere di mettere il programma in attesa attiva finché i button restano attivi;.*

- `ZyboButton_status_t` `ZyboButton_getStatus` (`ZyboButton_t` \*buttons, `ZyboButton_mask_t` mask)

*Permette la lettura dello stato dei button presenti sulla board.*

### 6.16.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

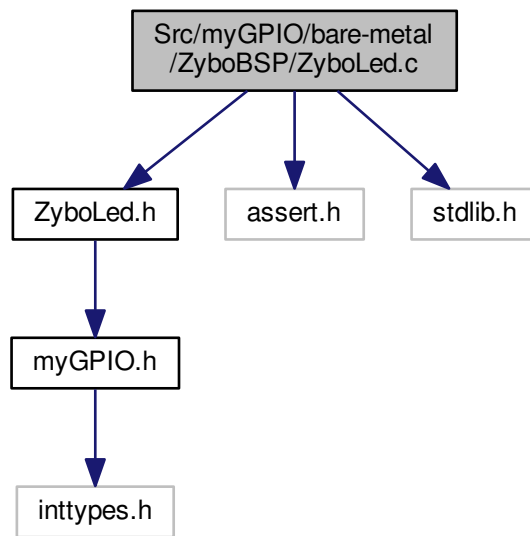
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.17 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboLed.c

```
#include "ZyboLed.h"
#include <assert.h>
#include <stdlib.h>
```

Grafo delle dipendenze di inclusione per ZyboLed.c:



## Funzioni

- static int `validatePair` (`ZyboLed_t` \*leds)
- void `ZyboLed_init` (`ZyboLed_t` \*leds, `myGPIO_t` \*gpio, `myGPIO_mask` Led3\_pin, `myGPIO_mask` Led2\_pin, `myGPIO_mask` Led1\_pin, `myGPIO_mask` Led0\_pin)  
*Inizializza un oggetto di tipo `ZyboLed_t`.*
- void `ZyboLed_setStatus` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask, `ZyboLed_status_t` status)  
*Permette di accendere/spegnere i Led sulla board.*
- void `ZyboLed_toggle` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask)  
*Permette di accendere/spegnere i Led sulla board, invertendone il valore.*

### 6.17.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.



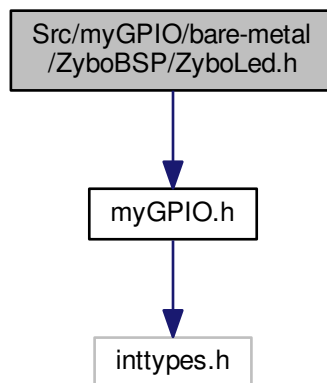
## 6.17.2 Documentazione delle funzioni

6.17.2.1 `static int validatePair ( ZyboLed_t * leds ) [static]`

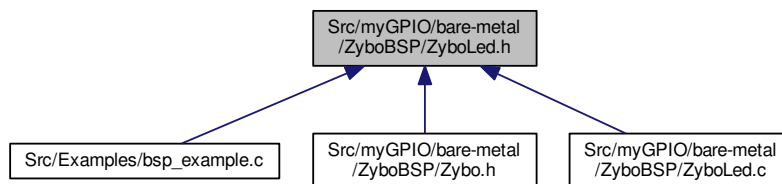
## 6.18 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboLed.h

```
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per ZyboLed.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct [ZyboLed\\_t](#)

*Struttura opaca che astrae l'insieme dei Led presenti sulla board Digilent Zybo;*

## Definizioni

- `#define ZyboLed(i) ((uint32_t)(1<<(i)))`

*Metodo alternativo per la specifica di uno dei led presenti sulla board Digilent Zybo.*

## Tipi enumerati (enum)

- enum `ZyboLed_mask_t` { `ZyboLed3` = 0x8U, `ZyboLed2` = 0x4U, `ZyboLed1` = 0x2U, `ZyboLed0` = 0x1U }

*Maschere di selezione dei led.*

- enum `ZyboLed_status_t` { `ZyboLed_off`, `ZyboLed_on` }

*Status di accensione/spegnimento dei led.*

## Funzioni

- void `ZyboLed_init` (`ZyboLed_t` \*leds, `myGPIO_t` \*gpio, `myGPIO_mask` Led3\_pin, `myGPIO_mask` Led2\_pin, `myGPIO_mask` Led1\_pin, `myGPIO_mask` Led0\_pin)

*Inizializza un oggetto di tipo `ZyboLed_t`.*

- void `ZyboLed_setStatus` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask, `ZyboLed_status_t` status)

*Permette di accendere/spegnere i Led sulla board.*

- void `ZyboLed_toggle` (`ZyboLed_t` \*leds, `ZyboLed_mask_t` mask)

*Permette di accendere/spegnere i Led sulla board, invertendone il valore.*

### 6.18.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

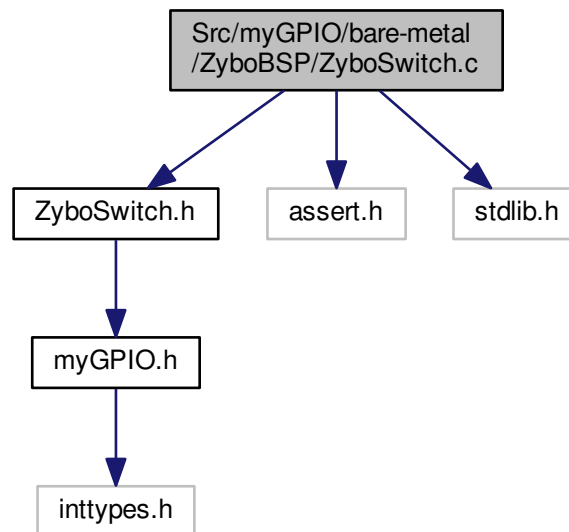
Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.19 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboSwitch.c

```
#include "ZyboSwitch.h"
#include <assert.h>
#include <stdlib.h>
```

Grafo delle dipendenze di inclusione per ZyboSwitch.c:



## Funzioni

- static int `validatePair` (`ZyboSwitch_t` \*switches)
- void `ZyboSwitch_init` (`ZyboSwitch_t` \*switches, `myGPIO_t` \*gpio, `myGPIO_mask` Switch3\_pin, `myGPIO_mask` Switch2\_pin, `myGPIO_mask` Switch1\_pin, `myGPIO_mask` Switch0\_pin)  
*Inizializza un oggetto di tipo `ZyboSwitch_t`.*
- `ZyboSwitch_status_t` `ZyboSwitch_getStatus` (`ZyboSwitch_t` \*switches, `ZyboSwitch_mask_t` mask)  
*Permette la lettura dello stato degli switch presenti sulla board.*

### 6.19.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

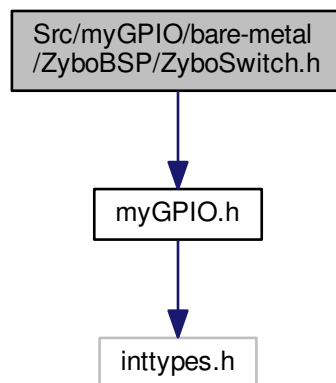
## 6.19.2 Documentazione delle funzioni

6.19.2.1 `static int validatePair ( ZyboSwitch_t * switches ) [static]`

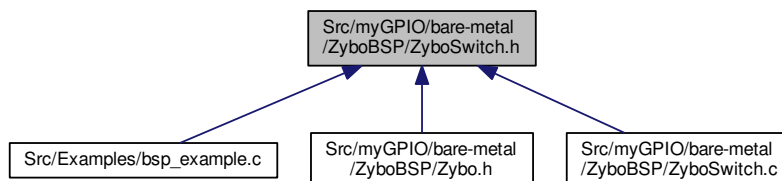
## 6.20 Riferimenti per il file Src/myGPIO/bare-metal/ZyboBSP/ZyboSwitch.h

```
#include "myGPIO.h"
```

Grafo delle dipendenze di inclusione per ZyboSwitch.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct [ZyboSwitch\\_t](#)

*Struttura opaca che astrae l'insieme degli switch presenti sulla board Digilent Zybo.*

## Definizioni

- #define [ZyboSwitch\(i\)](#) ((uint32\_t)(1<<(i)))

*Metodo alternativo per la specifica di uno degli switch presenti sulla board Digilent Zybo.*

## Tipi enumerati (enum)

- enum [ZyboSwitch\\_mask\\_t](#) { [ZyboSwitch3](#) = 0x8U, [ZyboSwitch2](#) = 0x4U, [ZyboSwitch1](#) = 0x2U, [ZyboSwitch0](#) = 0x1U }

*Maschere di selezione degli switch.*

- enum [ZyboSwitch\\_status\\_t](#) { [ZyboSwitch\\_off](#), [ZyboSwitch\\_on](#) }

*Status di attivo/inattivo degli switch.*

## Funzioni

- void [ZyboSwitch\\_init](#) ([ZyboSwitch\\_t](#) \*switches, [myGPIO\\_t](#) \*gpio, [myGPIO\\_mask](#) Switch3\_pin, [myGPIO\\_mask](#) Switch2\_pin, [myGPIO\\_mask](#) Switch1\_pin, [myGPIO\\_mask](#) Switch0\_pin)

*Inizializza un oggetto di tipo [ZyboSwitch\\_t](#).*

- [ZyboSwitch\\_status\\_t](#) [ZyboSwitch\\_getStatus](#) ([ZyboSwitch\\_t](#) \*switches, [ZyboSwitch\\_mask\\_t](#) mask)

*Permette la lettura dello stato degli switch presenti sulla board.*

### 6.20.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

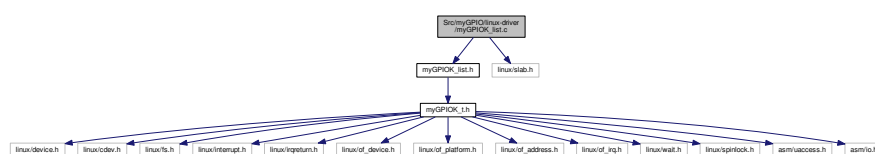
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.21 Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK\_list.c

```
#include "myGPIOK_list.h"
```

```
#include <linux/slab.h>
```

Grafo delle dipendenze di inclusione per myGPIOK\_list.c:



## Funzioni

- `int myGPIOK_list_Init (myGPIOK_list_t *list, uint32_t list_size)`  
*Inizializza una struttura dati `myGPIOK_list_t`.*
- `void myGPIOK_list_Destroy (myGPIOK_list_t *list)`  
*Dealloca gli oggetti internamente gestiti da un oggetto `myGPIOK_list_t`, liberando la memoria.*
- `int myGPIOK_list_add (myGPIOK_list_t *list, myGPIOK_t *device)`  
*Aggiunge un riferimento ad un oggetto `myGPIOK_t` alla lista.*
- `myGPIOK_t * myGPIOK_list_find_by_op (myGPIOK_list_t *list, struct platform_device *op)`  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- `myGPIOK_t * myGPIOK_list_find_by_minor (myGPIOK_list_t *list, dev_t dev)`  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- `myGPIOK_t * myGPIOK_list_find_irq_line (myGPIOK_list_t *list, int irq_line)`  
*Ricerca un oggetto `myGPIOK_t` all'interno della lista.*
- `uint32_t myGPIOK_list_device_count (myGPIOK_list_t *list)`  
*Restituisce il numero di device correntemente inseriti nella lista.*

### 6.21.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

24 06 2017

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

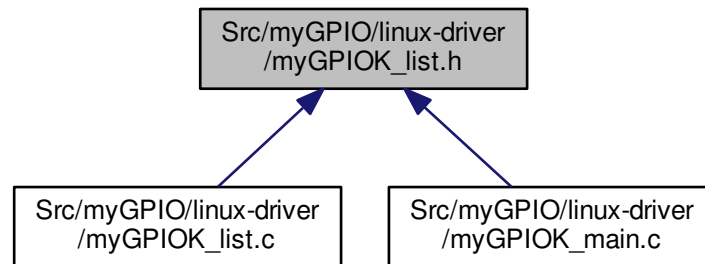
## 6.22 Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK\_list.h

```
#include "myGPIOK_t.h"
```

Grafo delle dipendenze di inclusione per myGPIOK\_list.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct [myGPIOK\\_list\\_t](#)

*Struttura dati per la gestione degli oggetti [myGPIOK\\_t](#) gestiti dal driver.*

## Funzioni

- int [myGPIOK\\_list\\_Init](#) ([myGPIOK\\_list\\_t](#) \*list, uint32\_t list\_size)  
*Inizializza una struttura dati [myGPIOK\\_list\\_t](#).*
- void [myGPIOK\\_list\\_Destroy](#) ([myGPIOK\\_list\\_t](#) \*list)  
*Dealloca gli oggetti internamente gestiti da un oggetto [myGPIOK\\_list\\_t](#), liberando la memoria.*
- int [myGPIOK\\_list\\_add](#) ([myGPIOK\\_list\\_t](#) \*list, [myGPIOK\\_t](#) \*device)  
*Aggiunge un riferimento ad un oggetto [myGPIOK\\_t](#) alla lista.*
- [myGPIOK\\_t](#) \* [myGPIOK\\_list\\_find\\_by\\_op](#) ([myGPIOK\\_list\\_t](#) \*list, struct platform\_device \*op)  
*Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.*
- [myGPIOK\\_t](#) \* [myGPIOK\\_list\\_find\\_by\\_minor](#) ([myGPIOK\\_list\\_t](#) \*list, dev\_t dev)  
*Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.*
- [myGPIOK\\_t](#) \* [myGPIOK\\_list\\_find\\_irq\\_line](#) ([myGPIOK\\_list\\_t](#) \*list, int irq\_line)  
*Ricerca un oggetto [myGPIOK\\_t](#) all'interno della lista.*
- uint32\_t [myGPIOK\\_list\\_device\\_count](#) ([myGPIOK\\_list\\_t](#) \*list)  
*Restituisce il numero di device correntemente inseriti nella lista.*

### 6.22.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

24 06 2017

## Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## 6.23 Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK\_main.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/proc_fs.h>
#include <linux/fcntl.h>
#include <linux/unistd.h>
#include <linux/platform_device.h>
#include <linux/of_device.h>
#include <linux/of_platform.h>
#include <linux/of_address.h>
#include <linux/of_irq.h>
#include <linux/interrupt.h>
#include <linux/irqreturn.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/sched.h>
#include <linux/poll.h>
#include "myGPIOK_t.h"
#include "myGPIOK_list.h"
```

Grafo delle dipendenze di inclusione per myGPIOK\_main.c:



## Definizioni

- `#define DRIVER_NAME "myGPIOK"`  
Nome identificativo del device-driver. DEVE corrispondere al valore del campo "compatible" nel device tree source.
- `#define DRIVER_FNAME "myGPIOK%d"`  
Nome del file creato in /dev/ per ciascuno dei device.
- `#define MAX_NUM_OF_DEVICES 15`



*Numero massimo di device gestibili.*

- #define `myGPIOK_USED_INT` 0xFFFFFFFFU

*Maschera di abilitazione degli interrupt per i singoli pin.*

## Funzioni

- `MODULE_LICENSE` ("GPL")
- `MODULE_AUTHOR` ("Salvatore Barone <salvator.barone@gmail.com>")
- `MODULE_DESCRIPTION` ("myGPIO device-driver in kernel mode")
- `MODULE_VERSION` ("3.2")
- `MODULE_ALIAS` (`DRIVER_NAME`)
- static int `myGPIOK_probe` (struct platform\_device \*op)  
*Viene chiamata quando il modulo viene inserito.*
- static int `myGPIOK_remove` (struct platform\_device \*op)
- static int `myGPIOK_open` (struct inode \*inode, struct file \*file\_ptr)  
*Invocata all'apertura del file corrispondente al device.*
- static int `myGPIOK_release` (struct inode \*inode, struct file \*file\_ptr)  
*Invocata alla chiusura del file corrispondente al device.*
- static loff\_t `myGPIOK_llseek` (struct file \*file\_ptr, loff\_t off, int whence)  
*Implementa le system-call lseek() e llseek().*
- static unsigned int `myGPIOK_poll` (struct file \*file\_ptr, struct poll\_table\_struct \*wait)  
*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*
- static ssize\_t `myGPIOK_read` (struct file \*file\_ptr, char \*buf, size\_t count, loff\_t \*off)  
*Legge dati dal device.*
- static ssize\_t `myGPIOK_write` (struct file \*file\_ptr, const char \*buf, size\_t size, loff\_t \*off)  
*Invia dati al device.*
- static irqreturn\_t `myGPIOK_irq_handler` (int irq, struct pt\_regs \*regs)  
*Interrupt-handler.*
- `MODULE_DEVICE_TABLE` (of, `myGPIOK_match`)  
*Inserisce una nuova entry nella tabella delle corrispondenze device - driver.*
- `module_platform_driver` (`myGPIOK_driver`)  
*la macro `module_platform_driver()` prende in input la struttura `platform_driver` ed implementa le funzioni `module_init()` e `module_close()` standard, chiamate quando il modulo viene caricato o rimosso dal kernel.*

## Variabili

- static `myGPIOK_list_t` \* `device_list` = NULL  
*Array di puntatori a struttura `myGPIOK_t`, contenente tutti i dati necessari al device driver.*
- static struct class \* `myGPIOK_class` = NULL  
*Classe del device Ai device-drivers viene associata una classe ed un device-name. Per creare ed associare una classe ad un device driver si può usare la seguente.*
- static struct of\_device\_id `myGPIOK_match` []  
*Identifica il device all'interno del device tree.*
- static struct platform\_driver `myGPIOK_driver`  
*Definisce quali funzioni `probe()` e `remove()` chiamare quando viene caricato un driver.*
- static struct file\_operations `myGPIOK_fops`  
*mantiene puntatori a funzioni che definiscono il gli operatori che agiscono su un file/device.*

## 6.24 Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK\_t.c

```
#include "myGPIOK_t.h"
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/poll.h>
```

Grafo delle dipendenze di inclusione per myGPIOK\_t.c:



### Funzioni

- int **myGPIOK\_Init** (**myGPIOK\_t** \*myGPIOK\_device, struct module \*owner, struct platform\_device \*op, struct class \*class, const char \*driver\_name, const char \*device\_name, uint32\_t serial, struct file\_operations \*f\_ops, irq\_handler\_t irq\_handler, uint32\_t irq\_mask)  
*Inizializza una struttura **myGPIOK\_t** e configura il device corrispondente.*
- void **myGPIOK\_Destroy** (**myGPIOK\_t** \*device)  
*Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.*
- void **myGPIOK\_SetCanRead** (**myGPIOK\_t** \*device)  
*Set del flag "interrupt occurred" (canRead)*
- void **myGPIOK\_ResetCanRead** (**myGPIOK\_t** \*device)  
*Reset del flag "interrupt occurred" (canRead)*
- void **myGPIOK\_TestCanReadAndSleep** (**myGPIOK\_t** \*device)  
*Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.*
- unsigned **myGPIOK\_GetPollMask** (**myGPIOK\_t** \*device, struct file \*file\_ptr, struct poll\_table\_struct \*wait)  
*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*
- void **myGPIOK\_IncrementTotal** (**myGPIOK\_t** \*device)  
*Incrementa il contatore degli interrupt per un particolare device.*
- void **myGPIOK\_WakeUp** (**myGPIOK\_t** \*device)  
*Risveglia i process in attesa sulle code di read e poll.*
- void \* **myGPIOK\_GetDeviceAddress** (**myGPIOK\_t** \*device)  
*Restituisce l'indirizzo virtuale di memoria cui è mappato un device.*
- void **myGPIOK\_GlobalInterruptEnable** (**myGPIOK\_t** \*device)  
*Abilita gli interrupt globali;.*
- void **myGPIOK\_GlobalInterruptDisable** (**myGPIOK\_t** \*device)  
*Disabilita gli interrupt globali;.*
- void **myGPIOK\_PinInterruptEnable** (**myGPIOK\_t** \*device, unsigned mask)  
*Abilita gli interrupt per i singoli pin del device.*
- void **myGPIOK\_PinInterruptDisable** (**myGPIOK\_t** \*device, unsigned mask)  
*Disabilita gli interrupt per i singoli pin del device.*
- unsigned **myGPIOK\_PendingPinInterrupt** (**myGPIOK\_t** \*device)  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- void **myGPIOK\_PinInterruptAck** (**myGPIOK\_t** \*device, unsigned mask)  
*Invia al device notifica di servizio di un interrupt;.*

### 6.24.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

24 06 2017

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

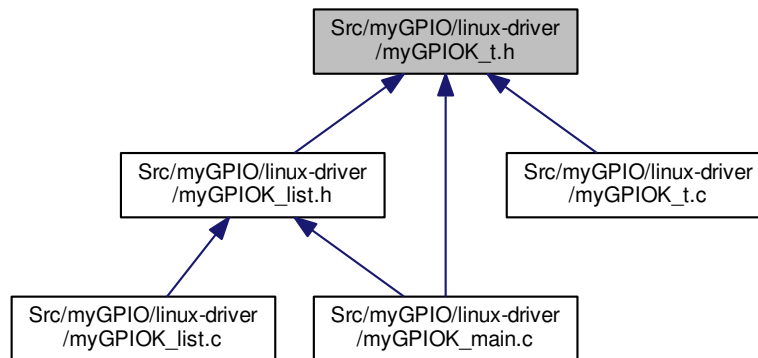
## 6.25 Riferimenti per il file Src/myGPIO/linux-driver/myGPIOK\_t.h

```
#include <linux/device.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/interrupt.h>
#include <linux/irqreturn.h>
#include <linux/of_device.h>
#include <linux/of_platform.h>
#include <linux/of_address.h>
#include <linux/of_irq.h>
#include <linux/wait.h>
#include <linux/spinlock.h>
#include <asm/uaccess.h>
#include <asm/io.h>
```

Grafo delle dipendenze di inclusione per myGPIOK\_t.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



## Strutture dati

- struct `myGPIO_t`  
*Struttura per l'astrazione di un device myGPIO in kernel-mode.*

## Definizioni

- #define `myGPIOK_GIES_OFFSET` 0x0CU  
*Offset, rispetto all'indirizzo base, del registro "GIES".*
- #define `myGPIOK_PIE_OFFSET` 0x10U  
*Offset, rispetto all'indirizzo base, del registro "PIE".*
- #define `myGPIOK_IRQ_OFFSET` 0x14U  
*Offset, rispetto all'indirizzo base, del registro "IRQ".*
- #define `myGPIOK_IACK_OFFSET` 0x18U  
*Offset, rispetto all'indirizzo base, del registro "IACK".*

## Funzioni

- int `myGPIOK_Init` (`myGPIO_t` \*myGPIO\_device, struct module \*owner, struct platform\_device \*op, struct class \*class, const char \*driver\_name, const char \*device\_name, uint32\_t serial, struct file\_operations \*f\_ops, irq\_handler\_t irq\_handler, uint32\_t irq\_mask)  
*Inizializza una struttura `myGPIO_t` e configura il device corrispondente.*
- void `myGPIOK_Destroy` (`myGPIO_t` \*device)  
*Deinizializza un device, rimuovendo le strutture kernel allocate per il suo funzionamento.*
- void `myGPIOK_SetCanRead` (`myGPIO_t` \*device)  
*Set del flag "interrupt occurred" (canRead)*
- void `myGPIOK_ResetCanRead` (`myGPIO_t` \*device)  
*Reset del flag "interrupt occurred" (canRead)*
- void `myGPIOK_TestCanReadAndSleep` (`myGPIO_t` \*device)  
*Testa la condizione "interrupt occurred", mettendo in attesa il processo, se necessario.*
- unsigned `myGPIOK_GetPollMask` (`myGPIO_t` \*device, struct file \*file\_ptr, struct poll\_table\_struct \*wait)  
*Verifica che le operazioni di lettura/scrittura risultino non-bloccanti.*

- void `myGPIOK_IncrementTotal` (`myGPIOK_t *device`)  
*Incrementa il contatore degli interrupt per un particolare device.*
- void `myGPIOK_WakeUp` (`myGPIOK_t *device`)  
*Risveglia i process in attesa sulle code di read e poll.*
- void \* `myGPIOK_GetDeviceAddress` (`myGPIOK_t *device`)  
*Restituisce l'indirizzo virtuale di memoria cui è mappato un device.*
- void `myGPIOK_GlobalInterruptEnable` (`myGPIOK_t *myGPIOK_device`)  
*Abilita gli interrupt globali;.*
- void `myGPIOK_GlobalInterruptDisable` (`myGPIOK_t *myGPIOK_device`)  
*Disabilita gli interrupt globali;.*
- void `myGPIOK_PinInterruptEnable` (`myGPIOK_t *myGPIOK_device`, unsigned mask)  
*Abilita gli interrupt per i singoli pin del device.*
- void `myGPIOK_PinInterruptDisable` (`myGPIOK_t *myGPIOK_device`, unsigned mask)  
*Disabilita gli interrupt per i singoli pin del device.*
- unsigned `myGPIOK_PendingPinInterrupt` (`myGPIOK_t *myGPIOK_device`)  
*Consente di ottenere una maschera che indichi quali interrupt non siano stati ancora serviti;.*
- void `myGPIOK_PinInterruptAck` (`myGPIOK_t *myGPIOK_device`, unsigned mask)  
*Invia al device notifica di servizio di un interrupt;.*

### 6.25.1 Descrizione dettagliata

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

24 06 2017

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.



## Capitolo 7

# Documentazione degli esempi

### 7.1 bsp\_example.c

Uso del driver myGPIO bare-metal per il pilotaggio dei Led, lettura di switch e button, pilotaggio di un display Hitachihd44780.

#### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

#### Data

12 05 2017

#### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

```
#include <stdio.h>
#include "xparameters.h"
#include "myGPIO.h"
#include "ZyboLed.h"
#include "ZyboSwitch.h"
#include "ZyboButton.h"
#include "hd44780.h"

int main()
{
    myGPIO_t gpioLed;
    myGPIO_Init(&gpioLed, XPAR_MYGPIO_0_S00_AXI_BASEADDR);
    ZyboLed_t boardLed;
    ZyboLed_init(&boardLed, &gpioLed, myGPIO_pin3,
                myGPIO_pin2, myGPIO_pin1, myGPIO_pin0);

    myGPIO_t gpioSwitch;
    myGPIO_Init(&gpioSwitch, XPAR_MYGPIO_1_S00_AXI_BASEADDR);
    ZyboSwitch_t boardSwitch;
```

```

ZyboSwitch_init(&boardSwitch, &gpioSwitch, myGPIO_pin3,
myGPIO_pin2, myGPIO_pin1, myGPIO_pin0);

myGPIO_t gpioButton;
myGPIO_Init(&gpioButton, XPAR_MYGPIIO_2_S00_AXI_BASEADDR);
ZyboButton_t boardButton;
ZyboButton_init(&boardButton, &gpioButton, myGPIO_pin3,
myGPIO_pin2, myGPIO_pin1, myGPIO_pin0);

myGPIO_t gpioDisplay;
myGPIO_Init(&gpioDisplay, XPAR_MYGPIIO_3_S00_AXI_BASEADDR);

HD44780_LCD_t lcd;
// Si decommenti le seguenti linee se si desidera interfacciarsi in modalità 8-bit
// HD44780_Init8(&lcd, &gpioDisplay, myGPIO_pin10, myGPIO_pin9, myGPIO_pin8
// myGPIO_pin0, myGPIO_pin1, myGPIO_pin2, myGPIO_pin3,
// myGPIO_pin4, myGPIO_pin5, myGPIO_pin6, myGPIO_pin7);

HD44780_Init4(&lcd, &gpioDisplay, myGPIO_pin10,
myGPIO_pin9, myGPIO_pin8,
myGPIO_pin0, myGPIO_pin1,
myGPIO_pin2, myGPIO_pin3);
HD44780_Clear(&lcd);
HD44780_Print(&lcd, "Hello world!");

ZyboLed_mask_t led_mask[] = {ZyboLed0, ZyboLed1,
ZyboLed2, ZyboLed3};
ZyboSwitch_mask_t switch_mask[] = {ZyboSwitch0,
ZyboSwitch1, ZyboSwitch2, ZyboSwitch3};
ZyboButton_mask_t button_mask[] = {ZyboButton0,
ZyboButton1, ZyboButton2, ZyboButton3};
for (;;) {
    int i;
    for (i=0; i<4; i++) {
        ZyboButton_status_t button_status =
ZyboButton_getStatus(&boardButton, button_mask[i]);
        ZyboSwitch_status_t switch_status =
ZyboSwitch_getStatus(&boardSwitch, switch_mask[i]);
        ZyboLed_status_t led_status = (button_status ==
ZyboButton_on || switch_status == ZyboSwitch_on ?
ZyboLed_on : ZyboLed_off);
        ZyboLed_setStatus(&boardLed, led_mask[i], led_status);
    }
}

return 0;
}

```

## 7.2 interrupt\_bare.c

Uso del driver myGPIO con interruzioni bare-metal su Zynq-7000

### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

### Data

23 06 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.



### Configurazione hardware

L'esempio fa riferimento ad una configurazione hardware in cui, oltre alla ip-core Zynq7000 processing system, sono presenti tre diversi device myGPIO, uno connesso ai led (base address 0x43c00000), uno connesso ai button (base address 0x43c10000) ed uno connesso agli switch (base address 0x43c20000). Lo schema viene riportato di seguito:

#### ISR per la gestione di interrupt provenienti dal gpio connesso agli switch

```
void swc_isr(void* data) {
    myGPIO_GlobalInterruptDisable(&swc_gpio);
    myGPIO_mask enabledInterrupt = myGPIO_EnabledPinInterrupt (&swc_gpio
    );
    myGPIO_PinInterruptDisable(&swc_gpio, enabledInterrupt);

    myGPIO_mask pendingInterrupt = myGPIO_PendingPinInterrupt (&
    swc_gpio);
    myGPIO_PinInterruptAck(&swc_gpio, pendingInterrupt);

    myGPIO_mask value = myGPIO_GetRead(&swc_gpio);
    myGPIO_SetValue(&led_gpio, myGPIO_pin0 |
    myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3,
    myGPIO_reset);
    myGPIO_SetValue(&led_gpio, value, myGPIO_set);

    myGPIO_PinInterruptEnable(&swc_gpio, enabledInterrupt);
    myGPIO_GlobalInterruptEnable(&swc_gpio);
}
```

La funzione di cui sopra non fa altro che disabilitare momentaneamente le interruzioni della periferica, leggere lo stato del registro "read", resettare i led, per poi accendere solo quello corrispondente allo switch arrivo e riabilitare l'interrupt della periferica.

#### ISR per la gestione di interrupt provenienti dal gpio connesso ai button

```
void btn_isr(void* data) {
    myGPIO_GlobalInterruptDisable(&btn_gpio);
    myGPIO_mask enabledInterrupt = myGPIO_EnabledPinInterrupt (&btn_gpio
    );
    myGPIO_PinInterruptDisable(&btn_gpio, enabledInterrupt);

    myGPIO_mask pendingInterrupt = myGPIO_PendingPinInterrupt (&
    btn_gpio);
    myGPIO_PinInterruptAck(&btn_gpio, pendingInterrupt);

    myGPIO_mask value = myGPIO_GetRead(&btn_gpio);
    myGPIO_SetValue(&led_gpio, myGPIO_pin0 |
    myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3,
    myGPIO_reset);
    myGPIO_SetValue(&led_gpio, value, myGPIO_set);

    myGPIO_PinInterruptEnable(&btn_gpio, enabledInterrupt);
    myGPIO_GlobalInterruptEnable(&btn_gpio);
}
```

La funzione di cui sopra non fa altro che disabilitare momentaneamente le interruzioni della periferica, leggere lo stato del registro "read", resettare i led, per poi accendere solo quello corrispondente al button premuto e riabilitare l'interrupt della periferica.

### Configurazione del GIC e registrazione degli interrupt handler

```
int int_config(void) {
    // inizializza il driver del GIC
    Xil_ExceptionInit();

    // ottiene i parametri di configurazione del GIC, lo configura ed inizializza
    // sintassi : XScuGic_LookupConfig(GIC_id)
    // sintassi : XScuGic_CfgInitialize(GIC_ptr, config, cpu_address)
    XScuGic_Config *IntcConfig = XScuGic_LookupConfig(gic_id);
    if (IntcConfig == NULL)
        return -1;
    if (XScuGic_CfgInitialize(&GIC, IntcConfig, IntcConfig->CpuBaseAddress) != XST_SUCCESS)
        return -1;

    // registra l'interrupt handler del GIC alla logica di gestione del processing-system
    // sintassi : Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, handler, gic_ptr)
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, (Xil_ExceptionHandler)XScuGic_InterruptHandler, &
```

```

    GIC);

// registrazione degli handler
// le righe seguenti stabiliscono quale sia l'handler da chiamare e quali dati bisogna passargli
// qualora si manifesti una interruzione su una linea di irq.
// sintassi : XScuGic_Connect(GIC, irq_line, handler, data)
if (XScuGic_Connect(&GIC, btn_irq_line, (Xil_InterruptHandler)btn_isr, (void*)NULL) !=
    XST_SUCCESS)
    return -1;
if (XScuGic_Connect(&GIC, swc_irq_line, (Xil_InterruptHandler)swc_isr, (void*)NULL) !=
    XST_SUCCESS)
    return -1;

// abilitazione degli interrupt sulle linee connesse alle periferiche
// sintassi: XScuGic_Enable(GIC,irq_line);
XScuGic_Enable(&GIC, btn_irq_line);
XScuGic_Enable(&GIC, swc_irq_line);

// abilitazione degli interrupt delle periferiche
myGPIO_GlobalInterruptEnable(&btn_gpio);
myGPIO_PinInterruptEnable(&btn_gpio, myGPIO_pin0 |
    myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3);
myGPIO_GlobalInterruptEnable(&swc_gpio);
myGPIO_PinInterruptEnable(&swc_gpio, myGPIO_pin0 |
    myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3);

// abilitazione degli interrupt del processing-system
Xil_ExceptionEnable();
return 0;
}

```

La funzione di configurazione delle interrupt e del GIC fa uso di alcune funzioni di libreria definite nell'header file "xscugic.h", il quale implementa il driver della periferica GIC, e di alcune macro definite nel file "xparameters.h". Solo per comodità le macro definite in xparameters.h sono state ridefinite come segue.

```

#define led_base_addr XPAR_MYGPIO_0_S00_AXI_BASEADDR
#define btn_base_addr XPAR_MYGPIO_1_S00_AXI_BASEADDR
#define swc_base_addr XPAR_MYGPIO_2_S00_AXI_BASEADDR
#define led_irq_line XPAR_FABRIC_MYGPIO_0_INTERRUPT_INTR
#define btn_irq_line XPAR_FABRIC_MYGPIO_1_INTERRUPT_INTR
#define swc_irq_line XPAR_FABRIC_MYGPIO_2_INTERRUPT_INTR
#define gic_id XPAR_SCUGIC_0_DEVICE_ID

```

Le funzioni di libreria usate sono riportate di seguito, assieme ad una breve descrizione tratta dalla documentazione interna.

- **Xil\_ExceptionInit()**: inizializza gli exception-handlers di tutti i processori. Per ARM Cortex A53, R5 ed A9 gli exception-handlers sono inizializzati staticamente, per cui questa funzione non fa niente. Viene mantenuta per prevenire errori in fase di compilazione e per garantire backward-compatibility.
- **XScuGic\_LookupConfig()**: lookup della configurazione di un device, basandosi sull'identificativo univoco dello stesso, dalla tabella contenente le configurazioni di tutti i device. Prende in ingresso un parametro DeviceId e restituisce un puntatore a XScuGic, contenente la configurazione, o NULL se il device non viene trovato.
- **XScuGic\_CfgInitialize()**: inizializza e configura un interrupt-controller instance/driver. La procedura di inizializzazione prevede:
  - l'inizializzazione dei campi di una struttura XScuGic;
  - la configurazione della Initial vector-table, con funzioni stub;
  - disabilitazione di tutte le sorgenti di interruzione

Parametri:

- InstancePtr: puntatore a struttura XScuGic;
  - ConfigPtr: puntatore alla configurazione del device, restituito dalla funzione XScuGic\_LookupConfig();
  - EffectiveAddr: indirizzo base del device; Restituisce XST\_SUCCESS se l'inizializzazione viene completata con successo.
- **Xil\_ExceptionRegisterHandler()**: crea una connessione tra l'identificativo di una sorgente di eccezioni e l'handler associato, in modo che l'handler venga eseguito qualora si manifestasse una eccezione. Prende i seguenti parametri:

- exception\_id: ID della sorgente di eccezioni;
  - Handler: puntatore alla funzione di servizio;
  - Data: puntatore ai dati da passare all'handler;
- XScuGic\_Connect(): crea una connessione tra l'identificativo di una sorgente di interruzioni e l'handler associato, in modo che l'handler venga eseguito qualora si manifestasse una interruzione. Prende i seguenti parametri:
    - InstancePtr: puntatore ad una istanza XScuGic;
    - Int\_Id: ID della sorgente di interruzioni;
    - Handler: puntatore alla funzione di servizio;
    - CallbackRef: puntatore ai dati da passare alla isr;
- Restituisce XST\_SUCCESS se l'handler è stato connesso correttamente.
- XScuGic\_Enable(): abilita la sorgente di interruzioni Int\_Id. Se ci sono pending interrupt per tale linea, scateneranno una interruzione dopo la chiamata a questa funzione. Parametri:
    - InstancePtr: puntatore ad una istanza XScuGic;
    - Int\_Id: ID della sorgente di interruzioni;
  - Xil\_ExceptionEnable(): abilita le interruzioni.

```
#include "xparameters.h"
#include "xscugic.h"
#include "myGPIO.h"

myGPIO_t led_gpio;
myGPIO_t btn_gpio;
myGPIO_t swc_gpio;
XScuGic GIC;

#define led_base_addr XPAR_MYGPIO_0_S00_AXI_BASEADDR
#define btn_base_addr XPAR_MYGPIO_1_S00_AXI_BASEADDR
#define swc_base_addr XPAR_MYGPIO_2_S00_AXI_BASEADDR

#define led_irq_line XPAR_FABRIC_MYGPIO_0_INTERRUPT_INTR
#define btn_irq_line XPAR_FABRIC_MYGPIO_1_INTERRUPT_INTR
#define swc_irq_line XPAR_FABRIC_MYGPIO_2_INTERRUPT_INTR

#define gic_id XPAR_SCUGIC_0_DEVICE_ID

// funzione di inizializzazione dei device gpio
void gpio_init(void);

// isr per button e switch
// devono necessariamente avere questa firma: restituire void e possedere un solo parametro puntatore
// a void. In questo caso non viene utilizzato (tutte le variabili sono globali), ma tale puntatore
// può essere usato per scambiare dati di ingresso/uscita alle isr
void btn_isr(void*); // isr per i button
void swc_isr(void*); // isr per gli switch

// funzione di configurazione del device gic e delle interruzioni
int int_config(void);

int main() {
    gpio_init();
    int_config();
    for (;;)
        return 0;
}

void gpio_init(void) {
    uint8_t i;

    myGPIO_Init(&led_gpio, led_base_addr);
    myGPIO_Init(&btn_gpio, btn_base_addr);
    myGPIO_Init(&swc_gpio, swc_base_addr);
    for (i=0; i<4; i++) {
        myGPIO_SetMode(&led_gpio, myGPIO_pin(i),
            myGPIO_write);
        myGPIO_SetValue(&led_gpio, myGPIO_pin(i),
            myGPIO_reset);
        myGPIO_SetMode(&btn_gpio, myGPIO_pin(i),
            myGPIO_read);
    }
}
```

```

        myGPIO_SetValue(&btn_gpio, myGPIO_pin(i),
myGPIO_reset);
        myGPIO_SetMode(&swc_gpio, myGPIO_pin(i),
myGPIO_read);
        myGPIO_SetValue(&swc_gpio, myGPIO_pin(i),
myGPIO_reset);
    }
}

void btn_isr(void* data) {
myGPIO_GlobalInterruptDisable(&btn_gpio);
myGPIO_mask enabledInterrut = myGPIO_EnabledPinInterrupt(&btn_gpio
);
myGPIO_PinInterruptDisable(&btn_gpio, enabledInterrut);

myGPIO_mask pendingInterrupt = myGPIO_PendingPinInterrupt(&
btn_gpio);
myGPIO_PinInterruptAck(&btn_gpio, pendingInterrupt);

myGPIO_mask value = myGPIO_GetRead(&btn_gpio);
myGPIO_SetValue(&led_gpio, myGPIO_pin0 |
myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3,
myGPIO_reset);
myGPIO_SetValue(&led_gpio, value, myGPIO_set);

myGPIO_PinInterruptEnable(&btn_gpio, enabledInterrut);
myGPIO_GlobalInterruptEnable(&btn_gpio);
}

void swc_isr(void* data) {
myGPIO_GlobalInterruptDisable(&swc_gpio);
myGPIO_mask enabledInterrut = myGPIO_EnabledPinInterrupt(&swc_gpio
);
myGPIO_PinInterruptDisable(&swc_gpio, enabledInterrut);

myGPIO_mask pendingInterrupt = myGPIO_PendingPinInterrupt(&
swc_gpio);
myGPIO_PinInterruptAck(&swc_gpio, pendingInterrupt);

myGPIO_mask value = myGPIO_GetRead(&swc_gpio);
myGPIO_SetValue(&led_gpio, myGPIO_pin0 |
myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3,
myGPIO_reset);
myGPIO_SetValue(&led_gpio, value, myGPIO_set);

myGPIO_PinInterruptEnable(&swc_gpio, enabledInterrut);
myGPIO_GlobalInterruptEnable(&swc_gpio);
}

int int_config(void) {
// inizializza il driver del GIC
Xil_ExceptionInit();

// ottiene i parametri di configurazione del GIC, lo configura ed inizializza
// sintassi : XScuGic_LookupConfig(GIC_id)
// sintassi : XScuGic_CfgInitialize(GIC_ptr, config, cpu_address)
XScuGic_Config *IntcConfig = XScuGic_LookupConfig(gic_id);
if (IntcConfig == NULL)
return -1;
if (XScuGic_CfgInitialize(&GIC, IntcConfig, IntcConfig->CpuBaseAddress) != XST_SUCCESS)
return -1;

// registra l'interrupt handler del GIC alla logica di gestione del processing-system
// sintassi : Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, handler, gic_ptr)
Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, (Xil_ExceptionHandler)XScuGic_InterruptHandler, &GIC)
;

// registrazione degli handler
// le righe seguenti stabiliscono quale sia l'handler da chiamare e quali dati bisogna passargli
// qualora si manifesti una interruzione su una line di irq.
// sintassi : XScuGic_Connect(GIC, irq_line, handler, data)
if (XScuGic_Connect(&GIC, btn_irq_line, (Xil_InterruptHandler)btn_isr, (void*)NULL) !=
XST_SUCCESS)
return -1;
if (XScuGic_Connect(&GIC, swc_irq_line, (Xil_InterruptHandler)swc_isr, (void*)NULL) !=
XST_SUCCESS)
return -1;

// abilitazione degli interrupt sulle linee connesse alle periferiche
// sintassi: XScuGic_Enable(GIC,irq_line);
XScuGic_Enable(&GIC, btn_irq_line);
XScuGic_Enable(&GIC, swc_irq_line);

// abilitazione degli interrupt delle periferiche
myGPIO_GlobalInterruptEnable(&btn_gpio);
myGPIO_PinInterruptEnable(&btn_gpio, myGPIO_pin0 |
myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3);

```

```

myGPIO_GlobalInterruptEnable(&swc_gpio);
myGPIO_PinInterruptEnable(&swc_gpio, myGPIO_pin0 |
    myGPIO_pin1 | myGPIO_pin2 | myGPIO_pin3);

// abilitazione degli interrupt del processing-system
Xil_ExceptionEnable();
return 0;
}

```

## 7.3 mygpiok.c

Programma di esempio per l'interfacciamento con una periferica myGPIO attraverso un driver kernel.

### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

### Data

16 06 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

In questo specifico esempio l'interfacciamento avviene da user-space, interagendo attraverso il driver myGPIOK.

### Avvertimento

Se nel device tree source non viene indicato

```
compatible = "myGPIOK";
```

tra i driver compatibili con il device, il driver myGPIOK non viene correttamente istanziato ed il programma userspace non funzionerà.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#include "myGPIO.h"
#include "xil_gpio.h"

#ifdef __XIL_GPIO__
#define MODE_OFFSET      GPIO_TRI_OFFSET
#define WRITE_OFFSET     GPIO_DATA_OFFSET
#define READ_OFFSET      GPIO_READ_OFFSET
#else
#define MODE_OFFSET      myGPIO_MODE_OFFSET
#define WRITE_OFFSET     myGPIO_WRITE_OFFSET
#define READ_OFFSET      myGPIO_READ_OFFSET
#endif

```

```

void howto(void) {
    printf("Uso:\n");
    printf("gpio -d /dev/device -w|m <hex-value> -r\n");
    printf("\t-m <hex-value>: scrive nel registro \"mode\"\n");
    printf("\t-w <hex-value>: scrive nel registro \"write\"\n");
    printf("\t-r: legge il valore del registro \"read\"\n");
    printf("I parametri possono anche essere usati assieme.\n");
}

typedef struct {
    int dev_descr;
    uint8_t op_mode;
    uint32_t mode_value;
    uint8_t op_write;
    uint32_t write_value;
    uint8_t op_read;
} param_t;

int parse_args( int argc, char **argv, param_t *param) {
    int par;
    char* devfile = NULL;
    while((par = getopt(argc, argv, "d:w:m:r")) != -1) {
        switch (par) {
            case 'd' :
                devfile = optarg;
                break;
            case 'w' :
                param->write_value = strtoul(optarg, NULL, 0);
                param->op_write = 1;
                break;
            case 'm' :
                param->mode_value = strtoul(optarg, NULL, 0);
                param->op_mode = 1;
                break;
            case 'r' :
                param->op_read = 1;
                break;
            default :
                printf("%c: parametro sconosciuto.\n", par);
                howto();
                return -1;
        }
    }
    if (devfile == NULL) {
        printf ("è necessario specificare il device col quale interagire!\n");
        howto();
        return -1;
    }
    param->dev_descr = open(devfile, O_RDWR);
    if (param->dev_descr < 1) {
        perror(devfile);
        return -1;
    }
    return 0;
}

void gpio_op (param_t *param) {
    if (param->op_mode == 1) {
        printf("Scrittura sul registro mode: %08x\n", param->mode_value);
#ifdef __USE_PWRITE__
        lseek(param->dev_descr, MODE_OFFSET, SEEK_SET);
        write(param->dev_descr, &(param->mode_value), sizeof(uint32_t));
#else
        pwrite(param->dev_descr, &(param->mode_value), sizeof(uint32_t),
            MODE_OFFSET);
#endif
    }
    if (param->op_write == 1) {
        printf("Scrittura sul registro write: %08x\n", param->write_value);
#ifdef __USE_PWRITE__
        lseek(param->dev_descr, WRITE_OFFSET, SEEK_SET);
        write(param->dev_descr, &(param->write_value), sizeof(uint32_t));
#else
        pwrite(param->dev_descr, &(param->mode_value), sizeof(uint32_t),
            WRITE_OFFSET);
#endif
    }
    if (param->op_read == 1) {
        uint32_t read_value = 0;
#ifdef __USE_PREAD__
        lseek(param->dev_descr, READ_OFFSET, SEEK_SET);
        read(param->dev_descr, &read_value, sizeof(uint32_t));
#else
        pread(param->dev_descr, &read_value, sizeof(uint32_t),
            READ_OFFSET);
#endif
    }
}

```

```

        printf("Lettura dal registro read: %08x\n", read_value);
    }
}

int main (int argc, char **argv) {
    param_t param;

    printf("%s build %d\n", argv[0], BUILD);

    if (parse_args(argc, argv, &param) == -1)
        return -1;

    gpio_op(&param);

    close(param.dev_descr);

    return 0;
}

```

## 7.4 noDriver.c

Questo è un programma di esempio per l'interfacciamento con una periferica myGPIO.

### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

### Data

12 06 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

In questo specifico esempio l'interfacciamento avviene da user-space, agendo direttamente sui registri di memoria, senza mediazione di altri driver, usando il device-file /dev/mem.

```

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#include "myGPIO.h"

void howto(void) {
    printf("Uso:\n");
    printf("noDriver -a gpio_phisycal_address -w|m <hex-value> -r\n");
    printf("\t-m <hex-value>: scrive nel registro \\"mode\\"");
    printf("\t-w <hex-value>: scrive nel registro \\"write\\"");
    printf("\t-r: legge il valore del registro \\"read\\"");
    printf("I parametri possono anche essere usati assieme.\n");
}

```

```

int parse_args(    int      argc,
                  char      **argv,
                  uint32_t   *gpio_address,
                  uint8_t     *op_mode,
                  uint32_t     *mode_value,
                  uint8_t     *op_write,
                  uint32_t     *write_value,
                  uint8_t     *op_read)
{
    int par;
    while((par = getopt(argc, argv, "a:w:m:r")) != -1) {
        switch (par) {
            case 'a' :
                *gpio_address = strtoul(optarg, NULL, 0);
                break;
            case 'w' :
                *write_value = strtoul(optarg, NULL, 0);
                *op_write = 1;
                break;
            case 'm' :
                *mode_value = strtoul(optarg, NULL, 0);
                *op_mode = 1;
                break;
            case 'r' :
                *op_read = 1;
                break;
            default :
                printf("%c: parametro sconosciuto.\n", par);
                howto();
                return -1;
        }
    }
    return 0;
}

void gpio_op (    void*      vrt_gpio_addr,
                uint8_t      op_mode,
                uint32_t      mode_value,
                uint8_t      op_write,
                uint32_t      write_value,
                uint8_t      op_read)
{
    printf("Indirizzo gpio: %08x\n", (uint32_t)vrt_gpio_addr);
#ifdef __XIL_GPIO__
#define MODE_OFFSET      4U
#define WRITE_OFFSET     0U
#define READ_OFFSET     8U
#else
    myGPIO_t gpio;
    myGPIO_Init(&gpio, (uint32_t)vrt_gpio_addr);
#endif

    if (op_mode == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+MODE_OFFSET)) = mode_value;
        mode_value = *((uint32_t*)(vrt_gpio_addr+MODE_OFFSET));
#else
        myGPIO_SetMode(&gpio, mode_value, myGPIO_write);
        myGPIO_SetMode(&gpio, ~mode_value, myGPIO_read);
#endif
        printf("Scrittura sul registro mode: %08x\n", mode_value);
    }
    if (op_write == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+WRITE_OFFSET)) = write_value;
        write_value = *((uint32_t*)(vrt_gpio_addr+WRITE_OFFSET));
#else
        myGPIO_SetValue(&gpio, write_value, myGPIO_set);
        myGPIO_SetValue(&gpio, ~write_value, myGPIO_reset);
#endif
        printf("Scrittura sul registro write: %08x\n", write_value);
    }
    if (op_read == 1) {
        uint32_t read_value = 0;
#ifdef __XIL_GPIO__
        read_value = *((uint32_t*)(vrt_gpio_addr+READ_OFFSET));
#else
        read_value = myGPIO_GetRead(&gpio);
#endif
        printf("Lettura dal registro read: %08x\n", read_value);
    }
}

```



```

int main(int argc, char** argv) {
    uint32_t gpio_addr = 0;          // indirizzo di memoria del device gpio
    uint8_t op_mode = 0;             // impostato ad 1 se l'utente intende effettuare scrittura su mode
    uint32_t mode_value;              // valore che l'utente intende scrivere nel registro mode
    uint8_t op_write = 0;             // impostato ad 1 se l'utente intende effettuare scrittura su write
    uint32_t write_value;            // valore che l'utente intende scrivere nel registro write
    uint8_t op_read = 0;             // impostato ad 1 se l'utente intende effettuare lettura da read

    printf("%s build %d\n", argv[0], BUILD); // BUILD viene definita in compilazione

    if (parse_args(argc, argv, &gpio_addr, &op_mode, &mode_value, &op_write, &write_value, &
        op_read) == -1)
        return -1;
    if (gpio_addr == 0) {
        printf("è necessario specificare l'indirizzo di memoria del device.\n");
        howto();
        return -1;
    }

    int descriptor = open ("/dev/mem", O_RDWR);
    if (descriptor < 1) {
        perror(argv[0]);
        return -1;
    }

    uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
    uint32_t page_mask = ~(page_size-1);       // maschera di conversione indirizzo -> indirizzo
    pagina
    uint32_t page_addr = gpio_addr & page_mask; // indirizzo della "pagina fisica" a cui è mappato il
    device
    uint32_t offset = gpio_addr - page_addr;    // offset del device rispetto all'indirizzo della
    pagina
    void* vrt_page_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, page_addr);
    if (vrt_page_addr == MAP_FAILED) {
        printf("Mapping indirizzo fisico - indirizzo virtuale FALLITO!\n");
        return -1;
    }
    void* vrt_gpio_addr = vrt_page_addr + offset; // indirizzo virtuale del device gpio
    gpio_op(vrt_gpio_addr, op_mode, mode_value, op_write, write_value, op_read);

    munmap(vrt_page_addr, page_size);
    close(descriptor);

    return 0;
}

```

## 7.5 readAll.c

Programma di test/debug. Legge tutti i registri della periferica direttamente dal file /dev/mem

### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

### Data

19 06 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

```

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

void howto(void) {
    printf("Uso:\n");
    printf("noDriver -a <gpio_phisycal_address> -o <max-offset>\n");
    printf("-a <gpio_phisycal_address>: indirizzo fisico del device GPIO\n");
    printf("\t-o <max-offset>: offset dell'ultimo registro letto\n");
}

int parse_args(int argc, char **argv, uint32_t *gpio_address, uint32_t *max_offset) {
    int par;

    while((par = getopt(argc, argv, "a:o:")) != -1) {
        switch (par) {
            case 'a' :
                *gpio_address = strtoul(optarg, NULL, 0);
                break;
            case 'o' :
                *max_offset = strtoul(optarg, NULL, 0);
                break;
            default :
                printf("%c: parametro sconosciuto.\n", par);
                howto();
                return -1;
        }
    }
    return 0;
}

int main(int argc, char** argv) {
    uint32_t gpio_addr = 0;
    uint32_t max_offset = 16;

    if (parse_args(argc, argv, &gpio_addr, &max_offset) == -1)
        return -1;

    if (gpio_addr == 0) {
        printf("è necessario specificare l'indirizzo di memoria del device.\n");
        howto();
        return -1;
    }

    int descriptor = open ("/dev/mem", O_RDWR);
    if (descriptor < 1) {
        perror(argv[0]);
        return -1;
    }

    uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
    uint32_t page_mask = ~(page_size-1); // maschera di conversione indirizzo -> indirizzo
    pagina
    uint32_t page_addr = gpio_addr & page_mask; // indirizzo della "pagina fisica" a cui è mappato il
    device
    uint32_t offset = gpio_addr - page_addr; // offset del device rispetto all'indirizzo della
    pagina
    // conversione dell'indirizzo fisico in indirizzo virtuale
    void* vrt_page_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, page_addr);

    if (vrt_page_addr == MAP_FAILED) {
        printf("Mapping indirizzo fisico - indirizzo virtuale FALLITO!\n");
        return -1;
    }

    void* vrt_gpio_addr = vrt_page_addr + offset; // indirizzo virtuale del device gpio

    printf("base address : %08X\n", (uint32_t) vrt_gpio_addr);
    uint32_t read_value = 0;
    uint32_t i;
    for (i=0; i<=max_offset; i+=4) {
        read_value = *((uint32_t*)(vrt_gpio_addr+i));
        printf("\toffset : %08X => %08X\n", i, read_value);
    }

    munmap(vrt_page_addr, page_size);
    close(descriptor);

    return 0;
}

```

## 7.6 uio-int.c

Questo è un programma di esempio per l'interfacciamento con una periferica myGPIO.

### Autore

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

### Data

14 06 2017

### Copyright

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

In questo specifico esempio l'interfacciamento avviene da user-space, interagendo attraverso il driver uio. **Utilizza gli interrupt per la lettura.**

### Avvertimento

Se nel device tree source non viene indicato

```
compatible = "generic-uio";
```

tra i driver compatibili con il device, il driver UIO non viene correttamente istanziato ed il programma non funzionerà.

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "myGPIO.h"
#include "xil_gpio.h"

void howto(void) {
    printf("Uso:\n");
    printf("uio -d /dev/uioX -w|m <hex-value> -r\n");
    printf("\t-m <hex-value>: scrive nel registro \"mode\"\n");
    printf("\t-w <hex-value>: scrive nel registro \"write\"\n");
    printf("\t-r: legge il valore del registro \"read\"\n");
    printf("I parametri possono anche essere usati assieme.\n");
}

int parse_args(    int      argc,
                  char      **argv,
                  char      **uio,
                  uint8_t    *op_mode,
                  uint32_t    *mode_value,
                  uint8_t    *op_write,
                  uint32_t    *write_value,
                  uint8_t    *op_read)
{
    int par;
    while ((par = getopt(argc, argv, "d:w:m:r")) != -1) {
```

```

switch (par) {
case 'd' :
    *uio = optarg;
    break;
case 'w' :
    *write_value = strtoul(optarg, NULL, 0);
    *op_write = 1;
    break;
case 'm' :
    *mode_value = strtoul(optarg, NULL, 0);
    *op_mode = 1;
    break;
case 'r' :
    *op_read = 1;
    break;
default :
    printf("%c: parametro sconosciuto.\n", par);
    howto();
    return -1;
}
}
return 0;
}

void gpio_op (    void*        vrt_gpio_addr,
                 int          uio_descriptor,
                 uint8_t      op_mode,
                 uint32_t      mode_value,
                 uint8_t      op_write,
                 uint32_t      write_value,
                 uint8_t      op_read)
{
    printf("Indirizzo gpio: %08x\n", (uint32_t)vrt_gpio_addr);
#ifdef __XIL_GPIO__
    myGPIO_t gpio;
    myGPIO_Init(&gpio, vrt_gpio_addr);
#endif

    if (op_mode == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+GPIO_TRI_OFFSET)) = mode_value;
        mode_value = *((uint32_t*)(vrt_gpio_addr+GPIO_TRI_OFFSET));
#else
        myGPIO_SetMode(&gpio, mode_value, myGPIO_write);
        myGPIO_SetMode(&gpio, ~mode_value, myGPIO_reset);
#endif
        printf("Scrittura sul registro mode: %08x\n", mode_value);
    }
    if (op_write == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+GPIO_DATA_OFFSET)) = write_value;
        write_value = *((uint32_t*)(vrt_gpio_addr+GPIO_DATA_OFFSET));
#else
        myGPIO_SetValue(&gpio, write_value, myGPIO_set);
        myGPIO_SetValue(&gpio, ~write_value, myGPIO_reset);
#endif
        printf("Scrittura sul registro write: %08x\n", write_value);
    }
    if (op_read == 1) {
        uint32_t read_value = 0;
        // interrupt enable (interni alla periferica)
#ifdef __XIL_GPIO__
        // (globale + canale 2)
        XilGpio_GlobalInterrupt((uint32_t)vrt_gpio_addr,
        GLOBAL_INTR_ENABLE);
        XilGpio_ChannelInterrupt((uint32_t)vrt_gpio_addr,
        CHANNEL2_INTR_ENABLE);
#else
        myGPIO_GlobalInterruptEnable(&gpio);
        myGPIO_PinInterruptEnable(&gpio, myGPIO_pin0|
        myGPIO_pin1|myGPIO_pin2|myGPIO_pin3);
#endif

        printf("Attesa dell'interruzione\n");
        uint32_t interrupt_count = 1;
        if (read(uio_descriptor, &interrupt_count, sizeof(uint32_t)) != sizeof(uint32_t)) {
            printf("Read error!\n");
            return;
        }
        printf("Interrupt count: %08x\n", interrupt_count);
        // disabilitazione interrupt (interni alla periferica)
#ifdef __XIL_GPIO__
        XilGpio_GlobalInterrupt((uint32_t)vrt_gpio_addr,
        GLOBAL_INTR_DISABLE);
        XilGpio_ChannelInterrupt((uint32_t)vrt_gpio_addr,
        CHANNEL2_INTR_DISABLE);

```

```

        #else
        myGPIO_GlobalInterruptDisable(&gpio);
        myGPIO_PinInterruptDisable(&gpio, myGPIO_pin0|
myGPIO_pin1|myGPIO_pin2|myGPIO_pin3);
        #endif

        // "servizio" dell'interruzione.
        // lettura del registro
        #ifdef __XIL_GPIO__
        read_value = *((uint32_t*)(vrt_gpio_addr+GPIO_READ_OFFSET));
        #else
        read_value = myGPIO_GetRead(&gpio);
        #endif
        printf("Lettura dal registro read: %08x\n", read_value);
        #ifdef __XIL_GPIO__
        while (*((uint32_t*)(vrt_gpio_addr+GPIO_READ_OFFSET)) != 0);
        #else
        while(myGPIO_GetRead(&gpio) != 0);
        #endif

        // invio dell'ack alla periferica
        #ifdef __XIL_GPIO__
        XilGpio_Ack_Interrupt((uint32_t*)vrt_gpio_addr,
CHANNEL2_ACK);
        #else
        myGPIO_PinInterruptAck(&gpio,
myGPIO_PendingPinInterrupt(&gpio));
        #endif

        uint32_t reenable = 1;
        if (write(uio_descriptor, (void*)&reenable, sizeof(uint32_t)) != sizeof(uint32_t)) {
            printf("Write error!\n");
            return;
        }
    }
}

int main(int argc, char** argv) {
    char* uio_file = 0;           // nome del file uio
    uint8_t op_mode = 0;          // impostato ad 1 se l'utente intende effettuare scrittura su mode
    uint32_t mode_value;          // valore che l'utente intende scrivere nel registro mode
    uint8_t op_write = 0;         // impostato ad 1 se l'utente intende effettuare scrittura su write
    uint32_t write_value;         // valore che l'utente intende scrivere nel registro write
    uint8_t op_read = 0;          // impostato ad 1 se l'utente intende effettuare lettura da read

    printf("%s build %d\n", argv[0], BUILD); // BUILD viene definita in compilazione
    if (parse_args(argc, argv, &uio_file, &op_mode, &mode_value, &op_write, &write_value, &
op_read) == -1)
        return -1;
    if (uio_file == 0) {
        printf("è necessario specificare l'indirizzo di memoria del device.\n");
        howto();
        return -1;
    }
    int descriptor = open (uio_file, O_RDWR);
    if (descriptor < 1) {
        perror(argv[0]);
        return -1;
    }
    uint32_t page_size = sysconf(_SC_PAGESIZE); // dimensione della pagina
    void* vrt_gpio_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, 0);
    if (vrt_gpio_addr == MAP_FAILED) {
        printf("Mapping indirizzo fisico - indirizzo virtuale FALLITO!\n");
        return -1;
    }
    gpio_op(vrt_gpio_addr, descriptor, op_mode, mode_value, op_write, write_value, op_read);

    munmap(vrt_gpio_addr, page_size);
    close(descriptor);

    return 0;
}

```

## 7.7 uio.c

Questo è un programma di esempio per l'interfacciamento con una periferica myGPIO.

**Autore**

Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

**Data**

13 06 2017

**Copyright**

Copyright 2017 Salvatore Barone [salvator.barone@gmail.com](mailto:salvator.barone@gmail.com)

This file is part of Zynq7000DriverPack

Zynq7000DriverPack is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

Zynq7000DriverPack is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

In questo specifico esempio l'interfacciamento avviene da user-space, interagendo attraverso il driver uio.

**Avvertimento**

Se nel device tree source non viene indicato

```
compatible = "generic-uio";
```

tra i driver compatibili con il device, il driver UIO non viene correttamente istanziato ed il programma non funzionerà.

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "myGPIO.h"

void howto(void) {
    printf("Uso:\n");
    printf("uio -d /dev/uioX -w|m <hex-value> -r\n");
    printf("\t-m <hex-value>: scrive nel registro \\"mode\\"\\n");
    printf("\t-w <hex-value>: scrive nel registro \\"write\\"\\n");
    printf("\t-r: legge il valore del registro \\"read\\"\\n");
    printf("I parametri possono anche essere usati assieme.\\n");
}

int parse_args(    int      argc,
                  char      **argv,
                  char      *uio,           // file uio da usare
                  uint8_t    *op_mode,      // impostato ad 1 se l'utente intende effettuare scrittura su
                  mode
                  uint32_t    *mode_value,  // valore che l'utente intende scrivere nel registro mode
                  uint8_t    *op_write,     // impostato ad 1 se l'utente intende effettuare scrittura su
                  write
                  uint32_t    *write_value, // valore che l'utente intende scrivere nel registro write
                  uint8_t    *op_read)     // impostato ad 1 se l'utente intende effettuare lettura da
                  read
{
    int par;
    while((par = getopt(argc, argv, "d:w:m:r")) != -1) {
        switch (par) {
            case 'd' :
                *uio = optarg;
                break;
            case 'w' :
                *write_value = strtoul(optarg, NULL, 0);
                *op_write = 1;
                break;
        }
    }
}
```

```

        case 'm' :
            *mode_value = strtoul(optarg, NULL, 0);
            *op_mode = 1;
            break;
        case 'r' :
            *op_read = 1;
            break;
        default :
            printf("%c: parametro sconosciuto.\n", par);
            howto();
            return -1;
    }
}
return 0;
}

void gpio_op (    void*        vrt_gpio_addr,
                 uint8_t      op_mode,
                 uint32_t      mode_value,
                 uint8_t      op_write,
                 uint32_t      write_value,
                 uint8_t      op_read)
{
    printf("Indirizzo gpio: %08x\n", (uint32_t)vrt_gpio_addr);

#ifdef __XIL_GPIO__
#define MODE_OFFSET      4U
#define WRITE_OFFSET     0U
#define READ_OFFSET      8U
#else
    myGPIO_t gpio;
    myGPIO_Init(&gpio, (uint32_t)vrt_gpio_addr);
#endif

    if (op_mode == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+MODE_OFFSET)) = mode_value;
        mode_value = *((uint32_t*)(vrt_gpio_addr+MODE_OFFSET));
#else
        myGPIO_SetMode(&gpio, mode_value, myGPIO_write);
        myGPIO_SetMode(&gpio, ~mode_value, myGPIO_reset);
#endif
        printf("Scrittura sul registro mode: %08x\n", mode_value);
    }
    if (op_write == 1) {
#ifdef __XIL_GPIO__
        *((uint32_t*)(vrt_gpio_addr+WRITE_OFFSET)) = write_value;
        write_value = *((uint32_t*)(vrt_gpio_addr+WRITE_OFFSET));
#else
        myGPIO_SetValue(&gpio, write_value, myGPIO_set);
        myGPIO_SetValue(&gpio, ~write_value, myGPIO_reset);
#endif
        printf("Scrittura sul registro write: %08x\n", write_value);
    }
    if (op_read == 1) {
        uint32_t read_value = 0;
#ifdef __XIL_GPIO__
        read_value = *((uint32_t*)(vrt_gpio_addr+READ_OFFSET));
#else
        read_value = myGPIO_GetRead(&gpio);
#endif
        printf("Lettura dal registro read: %08x\n", read_value);
    }
}

int main(int argc, char** argv) {
    char* uio_file = 0;           // nome del file uio
    uint8_t op_mode = 0;          // impostato ad 1 se l'utente intende effettuare scrittura su mode
    uint32_t mode_value;          // valore che l'utente intende scrivere nel registro mode
    uint8_t op_write = 0;         // impostato ad 1 se l'utente intende effettuare scrittura su write
    uint32_t write_value;         // valore che l'utente intende scrivere nel registro write
    uint8_t op_read = 0;         // impostato ad 1 se l'utente intende effettuare lettura da read

    printf("%s build %d\n", argv[0], BUILD); // BUILD viene definita in compilazione

    if (parse_args(argc, argv, &uio_file, &op_mode, &mode_value, &op_write, &write_value, &op_read) == -1)
        return -1;
    if (uio_file == 0) {
        printf("È necessario specificare il device UIO col quale interagire.\n");
        howto();
        return -1;
    }
    int descriptor = open (uio_file, O_RDWR);
    if (descriptor < 1) {
        perror(argv[0]);
    }
}

```

```
        return -1;
    }
    uint32_t page_size = sysconf(_SC_PAGESIZE);    // dimensione della pagina
    void* vrt_gpio_addr = mmap(NULL, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, descriptor, 0);
    if (vrt_gpio_addr == MAP_FAILED) {
        printf("Mapping indirizzo fisico - indirizzo virtuale FALLITO!\n");
        return -1;
    }
    gpio_op(vrt_gpio_addr, op_mode, mode_value, op_write, write_value, op_read);

    munmap(vrt_gpio_addr, page_size);
    close(descriptor);

    return 0;
}
```



# Indice analitico

## Bare-metal, [7](#)

- [myGPIO\\_byte0](#), [11](#)
- [myGPIO\\_byte1](#), [11](#)
- [myGPIO\\_byte2](#), [11](#)
- [myGPIO\\_byte3](#), [11](#)
- [myGPIO\\_pin0](#), [10](#)
- [myGPIO\\_pin1](#), [10](#)
- [myGPIO\\_pin10](#), [10](#)
- [myGPIO\\_pin11](#), [10](#)
- [myGPIO\\_pin12](#), [10](#)
- [myGPIO\\_pin13](#), [10](#)
- [myGPIO\\_pin14](#), [10](#)
- [myGPIO\\_pin15](#), [10](#)
- [myGPIO\\_pin16](#), [10](#)
- [myGPIO\\_pin17](#), [10](#)
- [myGPIO\\_pin18](#), [10](#)
- [myGPIO\\_pin19](#), [10](#)
- [myGPIO\\_pin2](#), [10](#)
- [myGPIO\\_pin20](#), [10](#)
- [myGPIO\\_pin21](#), [10](#)
- [myGPIO\\_pin22](#), [11](#)
- [myGPIO\\_pin23](#), [11](#)
- [myGPIO\\_pin24](#), [11](#)
- [myGPIO\\_pin25](#), [11](#)
- [myGPIO\\_pin26](#), [11](#)
- [myGPIO\\_pin27](#), [11](#)
- [myGPIO\\_pin28](#), [11](#)
- [myGPIO\\_pin29](#), [11](#)
- [myGPIO\\_pin3](#), [10](#)
- [myGPIO\\_pin30](#), [11](#)
- [myGPIO\\_pin31](#), [11](#)
- [myGPIO\\_pin4](#), [10](#)
- [myGPIO\\_pin5](#), [10](#)
- [myGPIO\\_pin6](#), [10](#)
- [myGPIO\\_pin7](#), [10](#)
- [myGPIO\\_pin8](#), [10](#)
- [myGPIO\\_pin9](#), [10](#)
- [myGPIO\\_read](#), [11](#)
- [myGPIO\\_reset](#), [11](#)
- [myGPIO\\_set](#), [11](#)
- [myGPIO\\_write](#), [11](#)

## Button, [25](#)

- [ZyboButton0](#), [26](#)
- [ZyboButton1](#), [26](#)
- [ZyboButton2](#), [26](#)
- [ZyboButton3](#), [26](#)
- [ZyboButton\\_off](#), [26](#)
- [ZyboButton\\_on](#), [26](#)

## HD44780

- [HD44780\\_CursorLeft](#), [18](#)
- [HD44780\\_CursorRight](#), [18](#)
- [HD44780\\_INTERFACE\\_4bit](#), [18](#)
- [HD44780\\_INTERFACE\\_8bit](#), [18](#)
- [HD44780\\_CursorLeft](#)
  - [HD44780](#), [18](#)
- [HD44780\\_CursorRight](#)
  - [HD44780](#), [18](#)
- [HD44780\\_INTERFACE\\_4bit](#)
  - [HD44780](#), [18](#)
- [HD44780\\_INTERFACE\\_8bit](#)
  - [HD44780](#), [18](#)

## Led, [29](#)

- [ZyboLed0](#), [30](#)
- [ZyboLed1](#), [30](#)
- [ZyboLed2](#), [30](#)
- [ZyboLed3](#), [30](#)
- [ZyboLed\\_off](#), [30](#)
- [ZyboLed\\_on](#), [30](#)

- [myGPIO\\_byte0](#)
  - [Bare-metal](#), [11](#)
- [myGPIO\\_byte1](#)
  - [Bare-metal](#), [11](#)
- [myGPIO\\_byte2](#)
  - [Bare-metal](#), [11](#)
- [myGPIO\\_byte3](#)
  - [Bare-metal](#), [11](#)
- [myGPIO\\_pin0](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin1](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin10](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin11](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin12](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin13](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin14](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin15](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin16](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin17](#)
  - [Bare-metal](#), [10](#)
- [myGPIO\\_pin18](#)

- Bare-metal, [10](#)
- myGPIO\_pin19
  - Bare-metal, [10](#)
- myGPIO\_pin2
  - Bare-metal, [10](#)
- myGPIO\_pin20
  - Bare-metal, [10](#)
- myGPIO\_pin21
  - Bare-metal, [10](#)
- myGPIO\_pin22
  - Bare-metal, [11](#)
- myGPIO\_pin23
  - Bare-metal, [11](#)
- myGPIO\_pin24
  - Bare-metal, [11](#)
- myGPIO\_pin25
  - Bare-metal, [11](#)
- myGPIO\_pin26
  - Bare-metal, [11](#)
- myGPIO\_pin27
  - Bare-metal, [11](#)
- myGPIO\_pin28
  - Bare-metal, [11](#)
- myGPIO\_pin29
  - Bare-metal, [11](#)
- myGPIO\_pin3
  - Bare-metal, [10](#)
- myGPIO\_pin30
  - Bare-metal, [11](#)
- myGPIO\_pin31
  - Bare-metal, [11](#)
- myGPIO\_pin4
  - Bare-metal, [10](#)
- myGPIO\_pin5
  - Bare-metal, [10](#)
- myGPIO\_pin6
  - Bare-metal, [10](#)
- myGPIO\_pin7
  - Bare-metal, [10](#)
- myGPIO\_pin8
  - Bare-metal, [10](#)
- myGPIO\_pin9
  - Bare-metal, [10](#)
- myGPIO\_read
  - Bare-metal, [11](#)
- myGPIO\_reset
  - Bare-metal, [11](#)
- myGPIO\_set
  - Bare-metal, [11](#)
- myGPIO\_write
  - Bare-metal, [11](#)
- Switch, [33](#)
  - ZyboSwitch0, [34](#)
  - ZyboSwitch1, [34](#)
  - ZyboSwitch2, [34](#)
  - ZyboSwitch3, [34](#)
  - ZyboSwitch\_off, [34](#)
  - ZyboSwitch\_on, [34](#)
- Zybo, [24](#)
- ZyboButton0
  - Button, [26](#)
- ZyboButton1
  - Button, [26](#)
- ZyboButton2
  - Button, [26](#)
- ZyboButton3
  - Button, [26](#)
- ZyboButton\_off
  - Button, [26](#)
- ZyboButton\_on
  - Button, [26](#)
- ZyboLed0
  - Led, [30](#)
- ZyboLed1
  - Led, [30](#)
- ZyboLed2
  - Led, [30](#)
- ZyboLed3
  - Led, [30](#)
- ZyboLed\_off
  - Led, [30](#)
- ZyboLed\_on
  - Led, [30](#)
- ZyboSwitch0
  - Switch, [34](#)
- ZyboSwitch1
  - Switch, [34](#)
- ZyboSwitch2
  - Switch, [34](#)
- ZyboSwitch3
  - Switch, [34](#)
- ZyboSwitch\_off
  - Switch, [34](#)
- ZyboSwitch\_on
  - Switch, [34](#)