Università degli Studi di Padova

Corso di Laurea in Ingegneria Elettronica

Sviluppo di un sistema anti-intrusione su scheda FPGA

Laureando: Loris Pretto

Relatore: Prof. Daniele Vogrig

Sommario

Scopo della tesi era lo sviluppo di un sistema anti-intrusione. È stato inizialmente realizzato il modello VHDL per la descrizione del comportamento del sistema e successivamente il progetto è stato implementato su architettura FPGA. Il tutto è stato prodotto e testato sulla scheda Spartan 3 Starter Kit Board, prodotta e commercializzata da Digilent, con l'aiuto della suite di sviluppo ISE 12.4 nella sua versione gratuita concessa da Xilinx.

Indice

Introduzione

Ogni sistema di allarme prevede l'utilizzo di sensori al fine di monitorare la situazione circostante e decidere di conseguenza se e come attivare le procedure previste per i vari casi distinti in sede di progetto.
Per quel che riguarda i sensori utilizzati nei sistemi anti-intrusione, esistono due tipologie principali, che sono i quelli che generano impulsi di allarme (ad esempio dei sensori di movimento) e quelli che sono sensibili allo stato (ad esempio se una porta o una finestra sono aperte o chiuse).
Come è logico, in ogni sistema anti-intrusione è prevista la possibilità di accensione, cioè di renderlo pronto a segnalare eventuali condizioni di errore, e quindi anche quella di spegnimento, disattivandone le funzioni (ad esempio quando si entra in casa propria). Tuttavia queste operazioni sono molto delicate, perché possono consentire a chiunque (e quindi anche a potenziali intrusi) il controllo totale sul funzionamento dell'impianto e devono quindi essere adeguatamente protette, tramite chiave di abilitazione o per mezzo di un codice di protezione numerico o addirittura alfanumerico. In caso di codice di sicurezza, è ragionevole fornire all'utente una procedura per consentire la modifica di tale combinazione segreta. Come scelta di buonsenso tale funzione di modifica dovrà essere preceduta dallo spegnimento del sistema (che risulta protetto dalla password da modificare), in modo da impedire la variazione del codice in un sistema attivo. In impianti più raffinati potrebbe inoltre essere chiesta l'immissione del codice precedente, in modo da garantire la massima sicurezza.
Devono inoltre essere realizzate delle funzionalità per consentire all'utente di interfacciarsi con il sistema, che deve comunicare attraverso un display informazioni riguardanti ad esempio l'ultimo allarme verificatosi o se il sistema è acceso oppure spento.
Infine si devono scegliere le modalità di intervento nel caso in cui si verifichino errori, nei casi più semplici si ricorre a segnali acustici e luci.
Nel progetto seguente sono stati usati degli interruttori per la simulazione di sensori di cui rilevare lo stato, dei pulsanti per i sensori che generano impulsi d'allarme, un modulo con quattro display a sette segmenti per la visualizzazione dello stato in cui si trova il sistema, dei LED (uniti al display) per la segnalazione degli allarmi.
Un'opzione spesso utile nei sistemi elettronici digitali è la funzione di reset. Con questo comando non si fa altro che portare il circuito in una condizione di funzionamento certa e stabilita a priori, in un impianto anti-intrusione si può pensare che tale processo equivalga ad annullare gli allarmi in atto e a spegnere l'apparato. Inserendo questa possibilità tuttavia si rende il sistema d'allarme molto vulnerabile: se malintenzionati accedessero alla procedura per il reset, l'intera struttura verrebbe disattivata, cosa inaccettabile. Pertanto tale comando va debitamente protetto, mettendolo in un luogo nascosto oppure rendendo la procedura attiva previa inserimento di una combinazione segreta.

Scheda Spartan 3 Starter Kit Board

Il progetto è stato sviluppato sulla scheda Spartan 3 Starter Kit Board, sulla quale sono montati un FPGA Spartan-3 XC3S200 (descritto in maggior dettaglio in seguito), una memoria Flash Xilinx da 2Mbit (XCF02S) che contiene la programmazione, 1Mbit di memoria non-volatile per dati o codice rimanente dalla programmazione. È presente inoltre un jumper per caricare la configurazione da altre fonti.
Tra i dispositivi di comunicazione sono da menzionare una porta VGA a 3 bit (8 colori); una porta RS232 con 9 pin e un adattatore di livello dei segnali, utilizzabile per comunicazioni seriali con un PC, esiste anche la possibilità di avere un secondo canale; una porta PS2 che consente la connessione con mouse o tastiera (i pacchetti inviati e ricevuti dipendono dal dispositivo effettivamente collegato).
Sono inoltre previste forme di I/O più semplici, in particolare la scheda è corredata di otto interruttori; quattro pulsanti (il segnale è '1' se premuti); otto LED rossi; quattro display a sette segmenti, i catodi sono comuni tra i moduli (il segnale deve essere '0' se si vogliono accendere i segmenti), la selezione del display da illuminare si effettua portando a zero l'anodo comune (quindi per scrivere i display andranno abilitati e disabilitati continuamente in sequenza).
La scheda è munita anche di una sorgente di clock a cristallo con frequenza di 50 MHz (sul lato inferiore della scheda), è possibile inoltre aggiungere un altro clock nell'apposito alloggiamento.
Il trasferimento del codice VHDL prodotto nel FPGA si effettua per mezzo di un cavo JTAG, da connettere alla scheda in un apposito pettine, mentre la comunicazione con il PC avviene attraverso la porta parallela di quest'ultimo.
Sono previsti inoltre dei connettori per consentire di espandere le funzionalità della scheda con altri moduli per la realizzazione di particolari applicazioni.

Vista superiore della scheda Spartan 3 Starter Kit Board

Specifiche del progetto da realizzare

Introduzione

Si vuole sfruttare la scheda Spartan 3 Starter Kit Board per sviluppare un allarme anti-intrusione, mediante una descrizione VHDL.

Descrizione

Il sistema di allarme deve prevedere il seguente funzionamento:

• I pulsanti **BTN3** e **BTN2** rappresentano due sensori di intrusione quali, ad esempio, i sensori di movimento a infrarossi. Se il sistema è allarmato e si genera un segnale (tasto premuto) su uno di questi due, si deve attivare il segnale di allarme.

• I deviatori **SW7** e **SW6** sono due sensori di stato (indicano ad esempio se una finestra o una porta sono aperti). Dovrà essere generato un allarme solo quando lo stato di questi sensori differisce da quello che si ha nel momento di attivazione del sistema (cioè solo se una porta chiusa viene aperta o una finestra aperta viene chiusa).

• Il display a sette segmenti deve comunicare informazioni sullo stato del sistema. Quando è attivato devono essere accesi tutti e quattro i punti decimali, mentre devono essere spenti quando è disattivato. Appena attivato, il display dovrà riportare la scritta "**On**". Se il sistema viene poi disattivato senza che siano stati generati allarmi il display riporterà la scritta "**OFF**". Quando si verifica un allarme invece il display indicherà la scritta "**Er**" nelle prime due cifre e nelle due seguenti un numero che identifica il numero di allarme; tale scritta rimarrà attiva anche dopo la disattivazione dell'allarme.

• Quando viene generato un allarme, si accende il LED **LD0** per circa 5 secondi, dopo i quali il sistema rimane comunque attivo per rilevare altre infrazioni.

• L'attivazione del sistema avviene tramite chiave numerica: con i deviatori **SW3-SW0** imposto il numero e poi, premendo il tasto **BTN0**, si attiva / disattiva il sistema.

• Per impostare il codice di attivazione / disattivazione, il sistema deve essere disattivato e, con lo switch **SW4** in posizione 'HI', alla pressione di **BTN0** viene memorizzato il codice presente sugli **SW3-SW0**. **SW4** deve poi essere riportato a livello 'LO' per permettere al sistema di funzionare in maniera corretta. Se **SW4** viene portato a '1' mentre il sistema è attivato, il suo contributo deve essere ignorato.

• Il codice può essere sbagliato solo per tre volte consecutive, sia in fase di attivazione che in fase di disattivazione. A ogni errore dovranno accendersi in sequenza i LED **LD7**, **LD6** e **LD5**. Al quarto sbaglio deve attivarsi l'allarme. Quando invece il codice è corretto deve riazzerarsi il contatore degli errori e spegnersi i LED.

• Il tasto **BTN1** invece è un reset globale. Alla sua pressione il sistema deve disattivarsi (se attivato), un eventuale allarme in atto deve essere interrotto, il codice deve essere portato a "0000" e il display deve indicare "**OFF**".

• L'intero progetto deve essere scalabile. Si definiscano quindi nella entity TOP del progetto 3 numeri naturali (come constant o come generic) che indicano:

N Numero dei sensori che generano impulsi di allarme (default = 2)
M Numero dei sensori dei quali rilevare il cambio di stato (default = 2)
L Dimensione del codice di attivazione (default = 4)

Tutto il codice deve essere sviluppato in modo che cambiando questi numeri il progetto rimanga sintetizzabile.

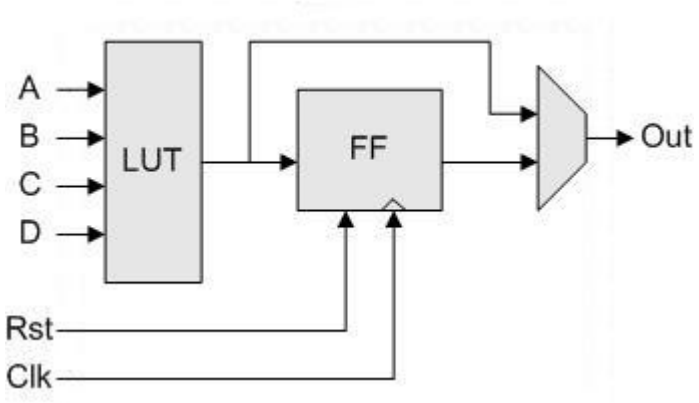
Scelte progettuali per i comportamenti non descritti

Eventuali condizioni non specificate erano a discrezione, per alcune era esplicitamente dichiarato che la scelta spettava al progettista.
La prima scelta è stata la numerazione dei sensori di allarme: è stato scelto di usare numeri da 1 a N per i sensori di intrusione e da N + 1 a N + M per i sensori di cui rilevare cambi di stato, questa scelta non è molto rilevante ai fini del funzionamento del sistema, è solamente una semplice convenzione per associare un numero ai vari sensori.
Inoltre le specifiche lasciavano libertà su come segnalare che il codice è stato inserito per quattro volte consecutive sbagliato. È stato scelto di non far accendere sequenzialmente i LED **LD7**, **LD6** e **LD5** in sequenza al quarto errore di inserimento, ma di far visualizzare al display la scritta "**Er C**", con i punti decimali accesi.
Inoltre era da gestire correttamente il caso in cui ci si venisse a trovare nella situazione appena descritta: consentire l'accensione o lo spegnimento del sistema sarebbe stato insensato, quindi si è deciso che l'unico modo per sbloccare il sistema da questa condizione è eseguire il reset tramite il tasto **BTN1**.
È stato scelto di far visualizzare la scritta "**COdE**" con i punti decimali spenti mentre il sistema è nella fase di modifica del codice di protezione, per far capire all'utilizzatore che l'impianto si trova in una fase di funzionamento particolare. Se poi l'apparato fosse spento e visualizzasse l'ultimo allarme generato, portando il sistema nella fase di inserimento della combinazione e impostandola, una volta riportato **SW4** nello stato 'LO' (cioè a livello logico '0'), il sistema ricorda ancora tale errore, cosa logica in quanto la sola modifica del codice non comporta la risoluzione dell'infrazione rilevata.
Quando il sistema è spento ma visualizza l'ultimo allarme verificatosi, alla successiva riaccensione questa informazione viene cancellata e il display visualizza "**On**" con i punti decimali accesi. Ciò è motivato dal fatto che quando si spegne un sistema con un allarme generato, è naturale andare a verificare la causa d'allarme e se è possibile la si risolve, quindi alla successiva abilitazione non è più necessario tenere traccia dell'inconveniente accaduto nella precedente fase di funzionamento. Inoltre, in sistemi più evoluti, può essere pensato di scrivere ogni causa di errore in modo automatico in un file, che tiene traccia della storia passata dell'impianto.
Come ultima assunzione è stato scelto di far visualizzare sempre l'ultimo allarme, e quindi se il sistema rileva ulteriori infrazioni rispetto a quella visualizzata, il display viene aggiornato con la più recente. Ciò è logico se si pensa al fatto che andando a verificare sempre l'ultimo errore riscontrato si va più vicini alla causa che l'ha generato; ad esempio un ladro che dapprima entra da una finestra, percorre un corridoio e tenta di aprire una cassaforte potrebbe attivare più sensori, ma quelli da tenere sotto più stretta osservazione sono senz'altro quelli nella stanza dove sono custoditi gli oggetti di valore.

FPGA

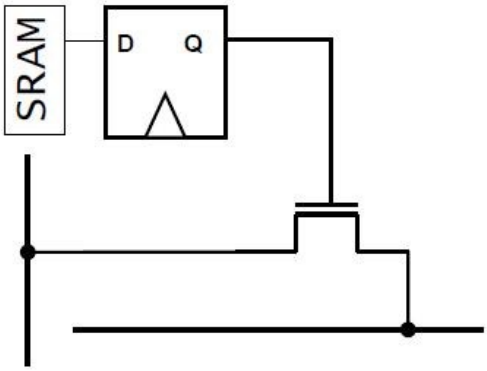
Un FPGA (Field Programmable Logic Array) è un componente elettronico programmabile dall'utente, si presenta come un normale integrato e può essere programmato più e più volte, consentendo un certa flessibilità. I primi esemplari furono proposti da Xilinx, con architetture simili ai CPLD e realizzati in tecnologia CMOS. Essenzialmente sono composti da molti blocchi programmabili semplici, collegati da una griglia di interconnessioni. Ogni blocco è composto da una LUT (Look-up Table), che è in grado di generare una qualsiasi funzione logica a 3-5 ingressi, un registro per salvare un risultato e un multiplexer per consentire di combinare funzioni logiche più complicate. Alcuni FPGA inoltre possiedono moduli specifici, come sommatori, moltiplicatori e memorie RAM.

Esempio di slice contenente una LUT a quattro ingressi



La struttura di base è simile a quella di un CPLD, con leggere differenze: gli FPGA hanno complessità maggiori, possiedono un maggior numero di blocchi programmabili semplici, i tempi di propagazione sono determinati per buona parte dalle interconnessioni e sono fortemente influenzati dal piazzamento dei blocchi logici, infine la programmazione avviene per mezzo di SRAM o più raramente attraverso l'utilizzo di antifusibili.
La programmazione di questi dispositivi è solitamente svolta con il supporto di linguaggi HDL e strumenti CAD che svolgono le fasi di simulazione e sintesi del circuito descritto dal codice HDL.
Da notare è il fatto che la produzione di FPGA è di fatto dominata da Xilinx e Altera, che da sole detengono circa i tre quarti del mercato mondiale, oltre a questi colossi ci sono poi produttori più piccoli che detengono la rimanente fetta del mercato. Normalmente ognuna di queste case produttrici offre una versione gratuita di un sistema di sviluppo CAD.
Questi componenti non raggiungono tuttavia le prestazioni degli ASIC (Application Specific Integratred Circuit) in termini di tempi di propagazione e consumo di potenza. Ciò deriva dal fatto che il circuito non è ottimizzato essendo già piazzato su silicio, potendo solo variare i collegamenti elettrici tra le strutture. Inoltre anche l'area di semiconduttore è spesso sovradimensionata rispetto al circuito effettivamente ottenuto, causando uno spreco di materiale. Tuttavia vi sono anche dei vantaggi nell'uso di questa tecnologia: in primo luogo si compra un componente finito e standard, per renderlo effettivamente funzionante è necessaria la fase di programmazione.

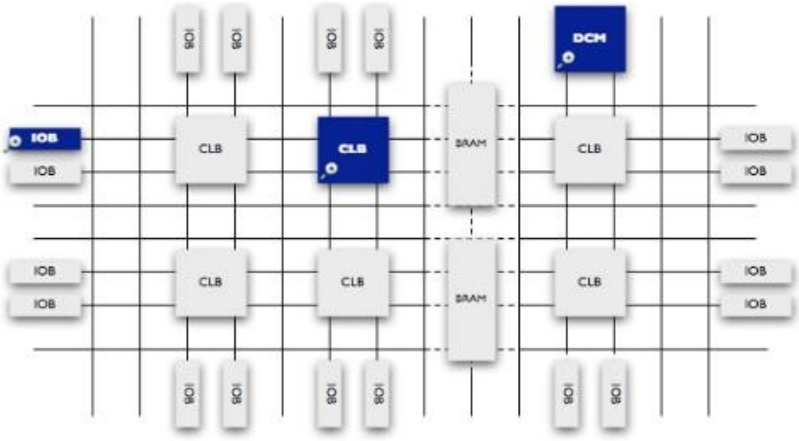
La fase di progettazione è assistita da strumenti CAD, che supportano anche fasi importanti e delicate come la simulazione del circuito sintetizzato. Inoltre il componente è riconfigurabile varie volte, a meno che vengano utilizzati interruttori a fusibile.
La figura a lato mostra un esempio di come si possa ottenere una connessione elettrica programmabile: quando il contenuto della SRAM è '1', il transistor MOS diventa un circuito chiuso e la connessione è stabilita. Viceversa, se la memoria contiene '0' il transistor si comporta come un circuito aperto e non vi è collegamento elettrico tra le piste.



Xilinx FPGA

Gli FPGA sviluppati da Xilinx hanno tutti la stessa architettura di base, sono cioè composti da molte slices, raggruppate a loro volta in CLB (Configurable Logic Block); IOB (Input/Output Block) e moduli DCM (Digital Clock Manager).
Ogni CLB è composto da una matrice di commutazione a 4-6 ingressi, circuiti di selezione (multiplexer, …) e flip-flop. Ogni slice, come già detto, consiste in una LUT, un registro e logica per gestire risultati più complicati provenienti da altre slices. La matrice dei contatti è molto flessibile e permette di gestire logica combinatoria, registri o RAM.
I blocchi IOB sono delle interfacce tra il dispositivo e l'esterno, realizzano svariati protocolli esistenti garantendo la massima flessibilità, questi moduli sono organizzati in banchi.
L'apparato DCM è utile per garantire che il segnale di clock arrivi alla logica sequenziale con un ritardo minore possibile e senza distorsione.
Oltre a queste caratteristiche di base, quasi tutti gli FPGA sono ormai dotati di un blocco di RAM integrato, sommatori e moltiplicatori.

Schema di un FPGA Xilinx



FPGA Spartan-3 XC3S200 (XC3S200FT256)

In questo componente ogni CLB consta di 4 slices, contenenti 2 LUT a 4 ingressi, 2 elementi di memoria (configurabili sia come flip-flop che come latch), multiplexer per creare operazioni logiche a più di quattro variabili, un percorso veloce per il calcolo del riporto e logica dedicata a funzioni aritmetiche. Ogni CLB è direttamente collegato ai 9 CLB vicini e ad una matrice di commutazione per la gestione delle interconnessioni interne al componente.
Lo Spartan-3 XC3S200 è inoltre dotato di 12 moltiplicatori hardware 18x18 bit, che consentono di eseguire moltiplicazioni con segno fornendo un risultato a 36 bit, sono inoltre ottimizzati per lo svolgimento di operazioni MAC (multiply and accumulate).
I moduli DCM (ne sono presenti 4) generano, a partire da una sorgente di clock di ingresso, un clock a frequenza multipla intera o frazionaria e con diverse fasi. Inoltre sono utili per compensare lo skew (cioè il fenomeno per il quale il segnale arriva a componenti diversi in tempi diversi) in reti di distribuzione del clock lunghe.
I blocchi di I/O (IOB) risultano divisi in tre parti, un percorso di ingresso, uno di uscita e uno three-state. Ogni percorso possiede elementi di memoria e può invertire i segnali. Questi blocchi sono organizzati in otto banchi, sono supportati diversi standard per segnali unipolari o differenziali, con tensioni comprese tra 1.2 e 3.3 V.

Tabella riassuntiva sulle caratteristiche degli FPGA della famiglia Spartan 3

Device	System Gates	Equivalent Logic Cells[1]	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000[2]	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	633	300
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	633	300

Notes:
1. Logic Cell = 4-input Look-Up Table (LUT) plus a 'D' flip-flop. "Equivalent Logic Cells" equals "Total CLBs" x 8 Logic Cells/CLB x 1.125 effectiveness.
2. These devices are available in Xilinx Automotive versions as described in DS314: Spartan-3 Automotive XA FPGA Family.

Linguaggi di descrizione hardware (HDL)

I linguaggi HDL (Hardware Description Language) sono nati con lo scopo di descrivere in modo preciso e accurato sistemi digitali, anche molto complessi, infatti permettono diversi livelli di astrazione, uso di librerie già costituite e strumenti per la simulazione. In parallelo all'introduzione di questi linguaggi viene messo a punto il concetto di sintesi, cioè come dalla descrizione della funzione di un circuito elettronico si può ricavare il circuito stesso.
Il linguaggio più utilizzato in Europa è il VHDL (Very High Speed Integrated Circuit HDL), sviluppato per conto dell'esercito americano e standardizzato dall'IEEE; ma esistono molti altri linguaggi, come Verilog e System-C.
Normalmente essi hanno l'aspetto di un normale linguaggio di programmazione, pertanto si possono usare vari tipi di dati (variabili), operazioni logico-aritmetiche e istruzioni di selezione (if-else) o iterative (while, for); inoltre essi consentono di istanziare e definire blocchi hardware precostituiti e gestirli in una struttura gerarchica. Sono inoltre in grado di descrivere il sistema a diversi livelli, i più usati sono:

- comportamentale (behavioral), che descrive la funzione dell'apparato senza pensare all'effettiva implementazione a livello di celle elementari;
- RTL (Register Transfer Level o Data-Flow), la quale descrive il sistema pensandolo composto da registri, bus, logica combinatoria;
- strutturale (structural), il sistema viene immaginato come un'insieme di componenti elementari connessi tra di loro.

Essi supportano l'intera fase di progettazione: sviluppo, simulazione, sintesi, testing, documentazione.
Nello sviluppo del progetto assegnato è stato usato il linguaggio VHDL, con la suite di sviluppo (concessa da Xilinx) ISE 12.4.
La descrizione di un modulo VHDL si avvale delle seguenti parti:

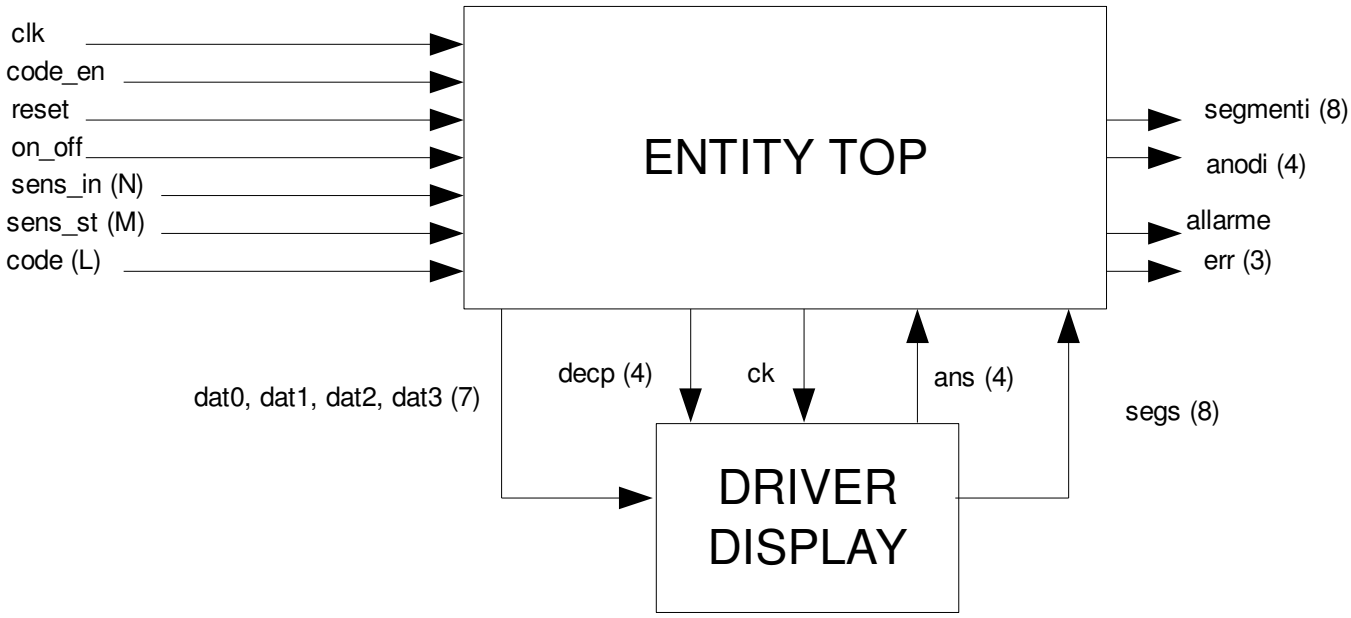
- un'interfaccia (entity) che descrive ingressi e uscite del circuito e il suo nome,
- una o più implementazioni (architecture) che sono la vera e propria descrizione del comportamento del circuito.

Altri elementi fondamentali nei modelli VHDL sono segnali e processi. Un processo è un blocco di codice che descrive il funzionamento di un modulo di logica sequenziale o combinatoria . È possibile far comunicare tra loro dei processi attraverso dei segnali. I processi sono inoltre caratterizzati dall'avere la stessa priorità, l'ordine in cui compaiono in una architecture non è importante ai fini del risultato. Un determinato processo si attiva quando uno dei segnali a cui è sensibile (riportati nella sensitivity list) cambia il suo valore (cioè si verifica un evento su quel segnale). I segnali sono strutture dati a cui è associato un tipo e a volte un valore iniziale, possono rappresentare forme d'onda nel tempo.
IEEE ha creato dei package particolari che estendono le funzionalità del linguaggio; tali pacchetti estendono il tipo bit aggiungendo nuovi valori (ad esempio il valore 'Z' per indicare lo stato di alta impedenza) nel pacchetto STD_LOGIC_1164 che definisce quindi il nuovo tipo di dato STD_LOGIC e le operazioni logiche per la sua elaborazione, e definendo anche operazioni aritmetiche su questi dati nei pacchetti NUMERIC_STD e STD_LOGIC_ARITH, che definiscono i tipi signed e unsigned e le relative operazioni matematiche.

Descrizione del codice prodotto

L'intero progetto è diviso in due entity, top e driver7seg, le quali comunicano attraverso appositi segnali. Nella figura sottostante è riportato uno schema dell'intero sistema.

Schema del sistema anti-intrusione



Entity top

Come da specifica, all'inizio della entity top sono stati definiti come generic tre parametri riguardanti il numero di allarmi e la lunghezza del codice, per poi elencare ingressi e uscite della entity.

Definizione della entity top

```
entity top is
      generic ( N : integer := 2;    -- numero allarmi di intrusione
                M : integer := 2; -- numero allarmi di stato
                L : integer := 4  -- dimensione codice attivazione
            );
      Port (    -- clock
                clk : in  STD_LOGIC;
                -- interruttore che abilita l'inserimento del codice
                code_en : in  STD_LOGIC;
                -- reset del sistema
                reset : in  STD_LOGIC;
```

```
                -- pulsante che abilita o disabilita il sistema / setta
                -- il codice
                on_off : in  STD_LOGIC;
                -- sensori di intrusione (pulsanti)
                sens_in : in STD_LOGIC_VECTOR (N - 1 downto 0);
                -- sensori di stato (switch)
                sens_st : in STD_LOGIC_VECTOR (M - 1 downto 0);
                -- codice di attivazione
                code : in STD_LOGIC_VECTOR (L - 1 downto 0);
                -- segmenti e punto display
                segmenti : out  STD_LOGIC_VECTOR (7 downto 0);
                -- anodi display
                anodi : out  STD_LOGIC_VECTOR (3 downto 0);
                -- led di avvenuto allarme
                allarme : out  STD_LOGIC;
                -- led segnalazione codice errato
                err : out  STD_LOGIC_VECTOR (2 downto 0));
    end top;
```

Si è scelto di spezzettare il codice in molti processi, ognuno per ogni segnale target, al fine di garantire semplicità e comprensibilità all'intero progetto.

Come prima cosa all'interno dell'architecture di questa entity si istanzia un componente di tipo driver7seg, definendone i collegamenti ai segnali di supporto per farlo comunicare con gli altri processi.
Inizialmente c'è un processo che a partire dalla rilevazione di un allarme ne determina il numero secondo quanto riportato nelle scelte per la realizzazione del codice. Tramite due cicli for si individua quale sia la causa di allarme e si aggiorna un segnale (chiamato i), che viene utilizzato nei due processi successivi, che servono per ricavare i dati da inviare al driver per il display sette segmenti per la visualizzazione del numero associato al relativo sensore.
Successivamente è stato inserito un processo per l'aggiornamento della scritta visualizzata dal display, tramite istruzioni else-if-elsif sono stati inviati i dati al display in base alla situazione in cui si trovava il sistema (acceso, spento, ...). In fase di sintesi, in questo pezzo di codice venivano generati molti warning (relativi alla sintesi di latch), per la risoluzione dei quali è bastato scrivere in modo completo le varie istruzioni di selezione, in modo da non avere casi incompleti, che determinano l'inserimento di latch.
Per quanto riguarda i processi che accendono / spengono il sistema, la scelta fatta è stata di utilizzare un bit per segnalare la condizione di on / off, uno per gestire un sistema in cui si sono verificati allarmi o no e uno per la verifica della situazione di inserimento di un nuovo codice di protezione. In particolare, il processo per l'accensione del sistema consiste praticamente in un flip-flop con reset asincrono, il cui segnale di clock è il pulsante stesso di abilitazione (c'è però da controllare che ci siano tentativi sufficienti a procedere e che non si stia impostando un nuovo codice).

Codice riportante il processo di accensione o spegnimento

```
process (activate, code, reset, tent)
     begin
          if reset = '1' then
               acceso <= '0';
          elsif activate'event and activate = '1' then
               if code = code_mem and codice = '0' and tent >= 0 then
                    acceso <= not acceso;
               end if;
          end if;
     end process;
```

Quindi ci sono due processi, rispettivamente per il reset della condizione di errore da spento ad acceso e per rilevare la condizione di errore, espressa da un apposito bit (chiamato errore).
Il processo seguente (molto semplice), verifica che il sistema sia spento prima di procedere a portare il sistema nello stato di inserimento di un nuovo codice di protezione, determinato dalla posizione dello switch **SW4**.
Successivamente c'è il segmento di codice per l'acquisizione dello stato iniziale all'accensione del sistema (questo proprio nella forma di un semplice flip-flop), seguito da quello per l'aggiornamento del codice di protezione, entrambi constano di poche righe e sono molto semplici.

Processi che rilevano lo stato iniziale degli interruttori e aggiornano il codice

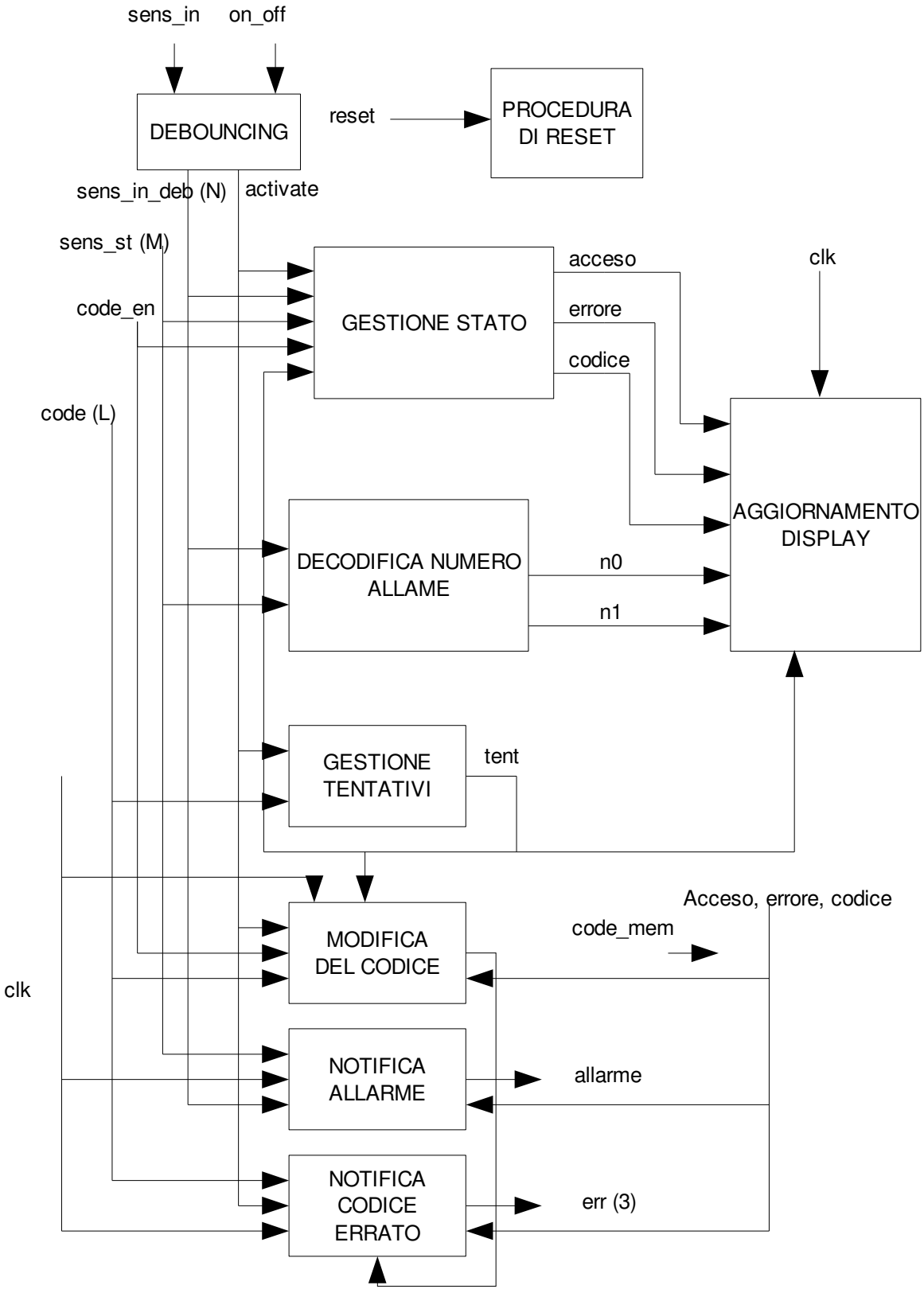
```
process (activate)
     begin
          if activate'event and activate = '1' then
               stato_in <= sens_st;
          end if;
     end process;

     process (activate, reset)
     begin
          if reset = '1' then -- sono all'inizio, reset
               code_mem <= (others => '0');
          elsif (activate'event and activate = '1') and codice = '1'
then
               code_mem <= code;
          end if;
     end process;
```

A questo punto sono stati inseriti una serie di processi che gestiscono la visualizzazione di condizioni di errore tramite i LED e variano il numero di tentativi per l'abilitazione del sistema. Il primo comanda l'accensione del LED **LD0** per 5 secondi circa, mentre il

secondo aggiorna il numero di tentativi a disposizione, decrementandoli se il codice è sbagliato, portandoli nuovamente al massimo se invece il codice è corretto o se è stato

Schema della entity top



eseguito il reset globale del sistema attraverso **BTN1**. I due successivi si occupano di gestire l'accensione sequenziale dei LED **LD7**, **LD6** e **LD5** in seguito al tentativo di modificare lo stato del sistema con un codice non corretto; ciò viene eseguito per mezzo di un contatore e di un segnale che viene aggiornato dopo determinati valori di conteggio.
È stato necessario inoltre utilizzare diversi processi per realizzare le funzioni di anti-rimbalzo dei pulsanti, in modo da evitare problemi legati a comportamenti oscillatori di questi ultimi. In particolare si è scelto di eseguire tali procedure su tutti i pulsanti tranne per **BTN1** che realizza la funzione di reset. Infatti anche se per effetto di eventuali rimbalzi il sistema viene resettato più volte non si verifica nessun inconveniente. I processi per la generazione di pulsanti con anti-rimbalzo sono tre: il primo crea un clock rallentato a partire da un segnale di clock di ingresso (fornito dall'oscillatore montato sulla scheda), il secondo e il terzo realizzano il vero e proprio debounce dei pulsanti, rispettivamente del pulsante di accensione e spegnimento e di quelli adibiti a simulare i sensori. Tali processi non fanno altro che "campionare" i segnali provenienti dai pulsanti e nel caso in cui il segnale e la sua versione campionata siano uguali, si assegna all'uscita il valore campionato.

*Processo che esegue la procedura di debouncing per il pulsante di accensione (**BTN0**)*

```
process(clk)
     begin
          if clk'event and clk = '1' then
               if clk_100Hz ='1' then
                    if on_off = ac_sampled then
                         activate <= on_off;
                    end if;
                    ac_sampled <= on_off;
               end if;
          end if;
     end process;
```

Entity driver7seg

È stato utile definire e realizzare un driver per il display sette segmenti, in modo da alleggerire il codice riportato nella entity top, dove ne è riportata un'istanza.

Dichiarazione e istanza del componente che realizza il driver per il diplay

```
Dichiarazione
...
component driver7seg is Port (
           dat0 : in  STD_LOGIC_VECTOR (6 downto 0);
           dat1 : in  STD_LOGIC_VECTOR (6 downto 0);
           dat2 : in  STD_LOGIC_VECTOR (6 downto 0);
           dat3 : in  STD_LOGIC_VECTOR (6 downto 0);
           decp : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
            ck : in  STD_LOGIC;
            segs : out  STD_LOGIC_VECTOR (7 downto 0);
            ans : out  STD_LOGIC_VECTOR (3 downto 0));
      end component;
      ...

      Istanza

      ...
      WR : driver7seg
      port map (
      dat0 => d0,
      dat1 => d1,
      dat2 => d2,
      dat3 => d3,
      decp => dp,
      ck => clk,
      segs => segmenti,
      ans => anodi
      );
      ...
```

Nel modulo driver7seg.vhd è stato definito quindi il componente per la corretta gestione del display: i segnali di ingresso sono i segnali per l'accensione dei segmenti dei quattro componenti, quelli per i punti decimali e una sorgente di clock. Il clock viene rallentato, e questa sua versione a frequenza ridotta viene usata per accendere alternativamente in sequenza i quattro moduli sette segmenti abilitandone o disabilitandone gli anodi comuni. La selezione viene effettuata per mezzo di una variabile (co) incrementata ad ogni fronte di salita del clock rallentato.

Estratto del processo per l'accensione sequenziale dei moduli sette segmenti

```
...
      if c(12)'event and c(12) = '1' then
            co <= co + 1;
            if co = "00" then  -- primo display
                  ans <= "1110";
                  segs (6 downto 0) <= dat0;
                  segs(7) <= decp(0);
            elsif co = "01" then -- secondo display
                  ans <= "1101";
                  segs (6 downto 0) <= dat1;
                  segs(7) <= decp(1);
            elsif co = "10" then -- terzo display
                  ans <= "1011";
                  segs (6 downto 0) <= dat2;
                  segs(7) <= decp(2);
      ...
```

Risultati

Il codice prodotto descritto in precedenza è stato trasferito e testato sulla board, tutte le specifiche sono state soddisfatte, anche quelle indicate come comportamenti non previsti.
Merita un'osservazione la quantità di risorse del componente FPGA effettivamente utilizzate dal codice prodotto. Pur non essendo ottimizzato il consumo di celle, si vede dalla figura che la gran parte dell'hardware non viene utilizzato, ed è a disposizione per eventuali aggiunte al progetto originario, ma anche per l'aggiunta di funzioni completamente diverse.

Consumo di risorse da parte del codice prodotto

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	162	3,840	4%	
Number used as Flip Flops	157			
Number used as Latches	5			
Number of 4 input LUTs	253	3,840	6%	
Number of occupied Slices	183	1,920	9%	
Number of Slices containing only related logic	183	183	100%	
Number of Slices containing unrelated logic	0	183	0%	
Total Number of 4 input LUTs	326	3,840	8%	
Number used as logic	253			
Number used as a route-thru	73			
Number of bonded IOBs	28	173	16%	
IOB Latches	1			
Number of BUFGMUXs	2	8	25%	
Average Fanout of Non-Clock Nets	3.08			

In particolare si vede che le risorse libere sono all'incirca il 90%, sia per quel che riguarda i registri che per quel che riguarda le LUT.
Nel codice finale inoltre sono presenti pochi warning (solo sette), in gran parte dovuti al piazzamento di latch anziché di flip-flop. Anche le specifiche sulla temporizzazione impostate nel progetto risultano tutte soddisfatte, per fare ciò è stato necessario mettere le giuste condizioni sulla gestione del clock (contenute nel file top.ucf insieme alle associazioni tra ingressi-uscite e pin del componente), per specificare correttamente quando i valori in uscita dai registri fossero ormai stabilizzati.

Conclusioni

Le maggiori difficoltà incontrate nello sviluppo del progetto sono legate alla realizzazione dello stesso tramite la suite di sviluppo. Per prima cosa è stato svolto del lavoro per acquisire familiarità con l'ambiente di sviluppo, la scrittura di codice in VHDL e la scheda Spartan 3. A tale scopo sono stati svolti alcuni semplici esercizi, tratti dal vecchio corso di Laboratorio di Elettronica Digitale, che sono risultati molto utili per apprendere i fondamenti per poter proseguire con il progetto vero e proprio. Tuttavia, anche nello sviluppo del sistema di allarme sono state incontrate delle difficoltà; infatti, prima di arrivare alla definitiva versione funzionante sono stati necessari parecchi tentativi per scrivere codice che risultasse corretto. In particolare inizialmente si era pensato di realizzare l'aggiornamento della logica che sovrintende l'accensione, spegnimento e generazione degli allarmi in un unico processo, con il risultato che la fase di sintesi segnalava svariati errori. Da qui la scelta di utilizzare tre bit per controllare il sistema: uno segnala se il sistema è spento o acceso, il secondo divide i casi in cui si è verificato un allarme oppure no, il terzo indica se si è nella fase di inserimento di un nuovo codice. Questa scelta ha permesso di dividere il codice in processi contenenti poche istruzioni e quindi molto più semplici e comprensibili.

Come ultima cosa si sottolinea uno dei vantaggi dello sviluppo di sistemi digitali tramite listati VHDL, la portabilità. Il codice realizzato può essere trasferito in svariate architetture, dovendo ripetere i passi di sintesi e di piazzamento dei componenti (svolti automaticamente dall'IDE). Quindi si sarebbe potuto anche comprare un FPGA, creare una propria board e testare il sistema su quest'ultima senza particolari problemi.