0.1 Implementazione

L'interfaccia del componente è la seguente:

```
entity scan_chain is
  generic(
    width : integer := 8;
    shift_direction : std_logic := '1'
  );  Port (
    clock : in  STD_LOGIC;
    en : in  STD_LOGIC;
    reset_n : in STD_LOGIC;
    scan_en : in  STD_LOGIC;
    d_reg : in  STD_LOGIC_VECTOR (width-1 downto 0);
    scan_in : in  STD_LOGIC;
    q_reg : out  STD_LOGIC_VECTOR (width-1 downto 0);
    scan_out : out  STD_LOGIC
  );
end scan_chain;
```

Codice Componente 1: Implementazione data-flow dell'anodes manager.

In ingresso, oltre ai segnali di *clock*, *enable* e *reset*, il componente avrà i seguenti segnali:

- *scan_ en*: bit di selezione della modalità di funzionamento (1 per modalità normale, 1 per modalità controllo);
- *d_ reg*: valori in ingresso dei flip-flop nel registro;
- *scan_ in*: valore in ingresso da inserire nel registro in caso di shift;

In uscita, invece, *q_ reg* sarà l'uscita del registro, mentre *scan_ out* sarà il bit tirato fuori dal registro in caso di shift.

Per quanto riguarda l'implementazione, è stata utilizzata una descrizione di tipo *structural*. In particolare, è stato generato un numero di flip-flop multiplexati pari al valore generico *width*, che rappresenta il parallelismo del registro. Tale componente è un tipo particolare di flip-flop D il cui ingresso viene prima selezionato tramite multiplexer. Per generare un registro che effettui shift a destra, si fissa come *scan_ in* di ogni flip-flop il valore in uscita del flip-flop precedente (alla sua sinistra), dopodiché si utilizza il multiplexer per decidere quale ingresso portare nel flip-flop. In particolare, *scan_ en* sarà il segnale di selezione: se 0 (modalità normale), in ingresso avremo il corrispettivo valore in ingresso di *d_ reg*, se 1 (modalità controllo) l'ingresso del flip-flop sarà *scan_ in*, e dunque l'uscita di quello precedente. Il discorso è analogo nel caso di shift a sinistra: in base al valore generico *shift_ direction* si stabilisce quale tipologia di shift register si vuole generare e dunque come collegare tra loro i flip-flop. L'implementazione completa è consultabile qui: scan_chain.vhd.

0.2 Simulazione

Per testing, si è effettuata una simulazione behavioural del componente: durante tale simulazione, si è utilizzato il componente in entrambe le sue modalità di funzionamento con la stringa '00000111'. In particolare, dapprima si è utilizzata la modalità normale (*scan_ en*=0) per caricare i valore nel registro. Dopodiché, quando *scan_ en*=1, il registro si comporta correttamente come uno shift register, shiftando i bit a sinistra ad ogni fronte di salita del clock. Una volta posto di nuovo *scan_ en*=0, il registro smette di shiftare le cifre e, al successivo fronte di salita del clock, carica di nuovo il valore in ingresso nel registro. I risultati sono consultabili in fig. 2.

Per completezza, in fig. 3 è stato riportato il risultato della simulazione con shift a destra (*shift_ direction*=1), perfettamente coerente con il comportamento atteso.

0.3 Approfondimento: Scan Chain On Board

Oltre alla classica simulazione, per testare il corretto funzionamento della scan chain abbiamo deciso di provare tale componente sulla board.

In particolare si è realizzata una top level entity che utilizza la scan chain per sostenere il valore in ingresso alla batteria di display a 7 segmenti presenti sulla board.

L'interfaccia della top level entity è la seguente :

```
1  [...]
2
3  entity scan_chain_on_board is
4        Port (  clock     : in  STD_LOGIC;
5                scan_in   : in  STD_LOGIC;
6             scan_clk  : in  STD_LOGIC;
7             scan_en   : in  STD_LOGIC;
8             scan_out    : out STD_LOGIC;
9             anodes   : out STD_LOGIC_VECTOR (7 downto 0);
10            cathodes  : out STD_LOGIC_VECTOR (7 downto 0)
11 );
12 end scan_chain_on_board;
13
14 [...]
```

Codice Componente 2: Interfaccia top level entity .

In ingresso, oltre ai segnali di *clock, scan_ in e scan_ en abbiamo anche scan_ clk. Quest'ultimo segnale è il clock per la scan chain, ma essendo quello della board un segnale a frequenza troppo elevata, che ci avrebbe impedito di apprezzare il funzionamento della scan chain* [1]*. Pertanto abbiamo deciso, anche se sbagliato, di dare manualmente un impulso "per simulare un clock personalizzato" premendo un pulsante presente sulla board. Quest'ultima scelta errata ci è stata segnalata anche*

[1] anche con un clock negli ordini dei khz premendo il pulsante associato a scan_en venivano eseguite troppe operazioni di shif

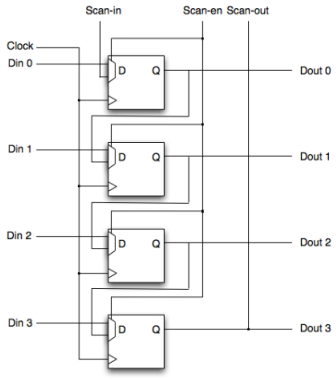

Figura 1: Architettura del componente scan_chain.

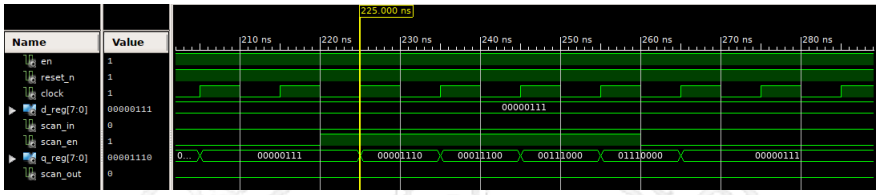

Figura 2: Simulazione del componente scan chain (shift a sinistra).

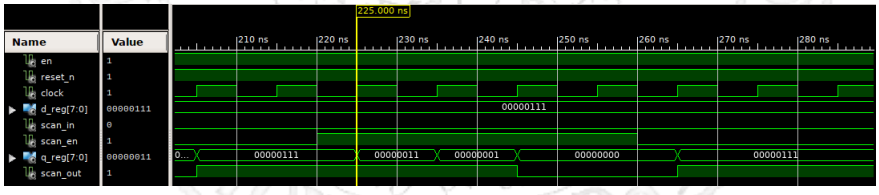

Figura 3: Simulazione del componente scan chain (shift a sinistra).

dal tool di sintesi, infatti mappando scan_ clk su un pulsante viene generato un errore, ma lo stesso tool suggerisce l'opportuna modifica da effettuare al file ucf per poter bypassare tale errore[2].

Di seguito il mapping di scan_en e di scan_clk sull'ucf:

```
[...]
#bottone centrale
  NET "scan_en"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14
#bottone inferiore
  NET "scan_clk"          CLOCK_DEDICATED_ROUTE = FALSE | LOC=P18 |
      IOSTANDARD=LVCMOS33; #IO_L9N_T1_DQS_D13_14
[...]
```

Codice Componente 3: Interfaccia top level entity .

Per testare il funzionamento del componente, bisogna settare il bit in ingresso alla scan chain spostando il primo switch e poi premere il pulsante centrale per abilitare l'operazione di shift e simulare i colpi di clock premedo il pulsante in basso. Se il bit in uscita alla scan chain è alto allora il primo led della batteria si illuminerà.

L'implementazione completa della top level_entity è consultabile qui: scan_chain_on_board.vhd.

[2] *Aggiungere al file ucf* "CLOCK_DEDICATED_ROUTE = FALSE" alla riga dove si effettua il mapping del bottone.