UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Ingegneria
Corso di Laurea in Ingegneria delle Telecomunicazioni

Relazione Progetto del Corso di Elettronica Dei Sistemi Digitali

Decodificatore Morse

Professore:
Prof. R. Passerone.

Studenti:
Rigamonti Michele
Zeni Mattia

Anno Accademico 2009-2010

Codice Morse

Il codice morse Morse è un sistema di trasmissione di lettere, numeri e segni di punteggiatura per mezzo di un segnale in codice ad intermittenza. Possiamo vederlo come una forma primitiva di trasmissione digitale.
Comunemente viene visto come un codice "binario", costituito da 2 soli stati (Codice Morse Internazionale), il punto e la linea, ma in realtà sono 5 (Codice Morse Americano): punto, linea, intervallo breve (tra lettere), intervallo medio (tra parole) e intervallo lungo (tra frasi).
Per quel che concerne il progetto ci siamo limitati ad usare la forma semplificata del codice Morse, quella conosciuta da tutti con soli 2 simboli, il Codice Morse Internazionale.

Storia del codice Morse:

Il nome del codice deriva dal suo creatore, Samuel Morse, che lo presentò pubblicamente nel 1837 e ne depositò il brevetto nel 1840. In realtà fu aiutato dal tecnico Alfred Vail che lo perfezionò. L'applicazione più immediata del codice fu il **telegrafo elettrico**, un apparecchio in grado di mandare un messaggio in portante tramite un semplice contatto. La prima dimostrazione fu un collegamento telegrafico negli Stati Uniti, tra Baltimora e Washington nel 1844.

La versione primitiva del codice era differente da quella attuale, ogni combinazione di punti e linee corrispondeva ad un numero, che doveva essere ricercato in un manuale per trovare la parola corrispondente. Con l'aiuto di Vail si arrivò al codice come lo conosciamo oggi, quindi dove la combinazione di punti e linee corrisponde ad una lettera o ad un numero:

CARATTERE	CODICE	DIGIT	CARATTERE	CODICE	DIGIT
A	● —		N	— ●	
B	— ● ● ●		O	— — —	
C	— ● — ●		P	● — — ●	
D	— ● ●		Q	— — ● —	
E	●		R	● — ●	
F	● ● — ●		S	● ● ●	
G	— — ●		T	—	
H	● ● ● ●		U	● ● —	
I	● ●		V	● ● ● —	
J	● — — —		X	— ● ● —	
K	— ● —		Y	— ● — —	
L	● — ● ●		Z	— — ● ●	

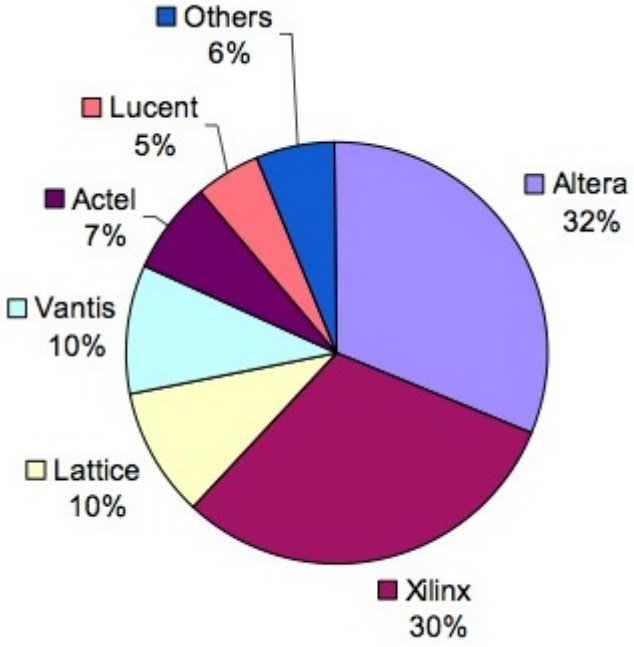
M	— —				

Tale codice è stato (e lo è ancora a livello radioamatoriale) il sistema di comunicazione che ha segnato la storia delle comunicazioni a distanza. Il telegrafo è stato utilizzato fino a pochi anni fa dalle maggiori Istituzioni dello Stato, enti pubblici e privati ed è stato lo standard internazionale per le comunicazioni marittime fino al 1999.

FPGA

Le FPGA sono dispositivi digitali programmabili, costituiti da un array di componenti logici (porte logiche, Flip-Flop, Buffer) ognuno dei quali può effettuare un'operazione molto semplice. Questi singoli componenti possono essere collegati tra loro con delle interconnessioni stabilite dal programmatore tramite uno speciale linguaggio di definizione hardware per eseguire compiti anche molto complessi.
Vengono molto utilizzate poiché rappresentano un buon compromesso tra costo e prestazioni rispetto ai molto più costosi e prestazionali ASIC, che per essere prodotti a prezzo vantaggioso richiedono un gran volume di pezzi. Inoltre una stessa FPGA può svolgere compiti differenti in momenti differenti a seconda del software con cui vengono scritte.

Vediamo un grafico del mercato delle FPGA:



Vediamo che la maggior parte del mercato è conteso dalla Xilinx e dalla Altera. Il progetto infatti è stato eseguito sulla scheda XC3S200 della Xilinx.
Solitamente le FPGA vengono programmate come il VHDL.

IL PROGETTO

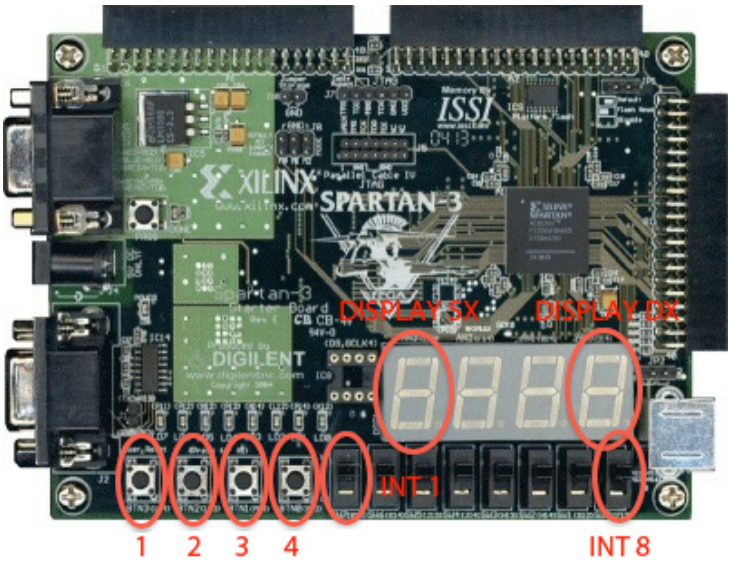
Il progetto consiste in un decodificatore Morse. L'utente ha la possibilità di inserire punto o linea, **a seconda di quanto mantiene premuto il pulsante apposito** (Bottone 3). Questo è reso possibile grazie ad un preciso lavoro di tempi che è stato svolto con l'ausilio di più contatori.
Era necessario infatti distinguere tra i **rimbalzi del tasto, i punti e le linee**. I rimbalzi, soprattutto, hanno creato molti problemi poichè il programma capiva che l'utente avesse digitato più punti, mentre ne era stato digitato uno solo. Ci sono volute molte ore in laboratorio prima di capire che era questo il problema. È stato necessario introdurre quindi un contatore che filtrasse gli sbalzi indesiderati.

Il software si presenta all'utente con 2 modalità:

- La prima è quella chiamata **Telegrafo**, dove l'utente digita con l'apposito pulsante (Bottone 3) i punti e le linee che vanno a comporre le lettere dell'alfabeto Morse e queste vengono visualizzate sui display a 7 segmenti. Il tutto avviene facendo scorrere una lettera alla volta per liberare un digit che servirà a visualizzare la lettera successiva.
- La seconda modalità è chiamata **Apprendimento** e serve per imparare il codice Morse. Sul display più a sinistra viene visualizzata una lettera generata in modo casuale e l'utente dovrà introdurre la corretta sequenza di punti e linee che corrisponde a quella lettera. L'introduzione avviene tramite lo stesso pulsante della modalità precedente (Bottone 3). La lettera corrispondente all'inserimento, verrà visualizzata nel digit più a destra. Se le due coincidono, lo schermo lampeggia e poi cambia la lettera random in modo da poter andare avanti con l'apprendimento. Se invece si è commesso un errore viene visualizzata lo stesso la lettera nel display più a destra per qualche secondo e poi, invece che cambiare lettera random, questa rimane inalterata finchè non la si digita giusta.
  Abbiamo inoltre deciso di introdurre un **aiuto** per gli utenti che si trovano in difficoltà: se non si conosce la corrispondenza per una lettera, si può utilizzare il primo interruttore sulla sinistra che, se alzato, abilita i led visualizzando i corrispondenti punti e linee che formano la lettera stessa. Quindi, dopo aver verificato, l'utente può riportare l'interruttore in posizione abbassata per ritornare alla modalità di inserimento.

Il qualsiasi momento si può uscire dalla modalità in cui si trova per tornare al menù, premendo il Bottone 4 o resettare il dispositivo tramite il **reset** (Interruttore 8).

Accesa la scheda l'utente visualizza sui display a 7 segmenti il menù scorrevole. Gli viene indicato che per entrare nella modalità Apprendimento deve premere il Bottone 1 mentre se vuole la modalità Telegrafo deve scegliere il Bottone 2.



DESCRIZIONE DEI COMPONENTI

MACCHINE A STATI FINITI: *Msf_menu, Msf_app e Msf_tel*

Il progetto si compone principalmente di tre diverse macchine a stati finiti, in quanto è stato necessario separarle sia per utilità pratica, sia per alleggerire il carico computazionale ad ogni colpo di clock.

La principale è ***MSF_MENU*** nella quale troviamo quattro stati: reset, menu, appendice e telegraph.

Il primo di questi è abilitato dall'interruttore *reset*, in qualunque stato ci troviamo se questo selettore viene attivato la macchina si posiziona nella condizione di reset, nella quale vengono azzerati tutti i registri e i selettori. Quando l'interruttore di reset viene abbassato, la macchina si porta in automatico nello stato *menu*. A questo punto sui digit viene visualizzato un menu scorrevole che mostra all'utente le due principali modalità di utilizzo, ovvero la modalità apprendimento, per imparare le associazioni corrette del codice morse; e la modalità telegrafo, nella quale viene visualizzato ciò che l'utente immette in codice morse.

Per passare alle modalità apprendimento o telegrafo macchina rimane in attesa, rispettivamente, dei bottoni *B1* o *B2*. Al fine di coordinare le tre macchine vi sono dei segnali di controllo, infatti mentre la macchina principale si trova negli stati di reset o menu, le altre due sono in una situazione di attesa dei segnali di attivazione, ovvero *Telegraph_EN* e *Apprendice_EN.* Quando vengono attivati le altre macchine cominciano le loro operazioni, e la *MSF_MENU* rimane in attesa dei segnali in ingresso, attivi alti, *Apprendice_STOP* e *Telegraph_STOP* per poter tornare allo stato di *menu*.

La ***MSF_APP*** attende che il segnale di enable si alzi, in seguito resetta i contatori e i registri, poi abilita la scrittura di una lettera casuale nel registro collegato al primo digit (quello sulla sinistra), in seguito entra nello stato di *appr_wait*. In questo stato un contatore attende per un limite di tempo fissato, in seguito controlla se sono stati immessi input dall'utente: in caso negativo ricomincia a contare, se invece sono stati salvati dei punti e delle linee, visualizza sul quarto digit la lettera corrispondente agli input presenti nei registri. Se le lettere presenti sul primo e sul quarto digit sono uguali torna ad *appr_1_pre* e dopo un reset genera un'altra lettera; se le lettere sono diverse segnala l'errore e resetta solamente il registro collegato al quarto digit, lasciando la possibilità di ritentare. Nel caso l'utente volesse conoscere la corretta sequenza morse collegata alla lettera visualizzata, è necessario solamente che abiliti l'interruttore *help* e i led mostreranno la sequenza corretta. Anche durante il normale inserimento i led aiuteranno l'utente mostrando i dati in memoria. Se l'utente desidera tornare al menù principale potrà farlo con il bottone *B4*, così facendo la *MSF_APP* alzerà il segnale di stop riposizionandosi nello stato di *idle*, ed al contempo la *MSF_MENU* cambierà il proprio stato passando a *menu.*

Per la normale modalità, ovvero la visualizzazione continua del codice morse immesso, si abilita la ***MSF_TEL***, passando alla modalità *telegraph*; anche in questo caso la macchina principale si mette in attesa del segnale di stop, e le operazioni vengono svolte dalla *MSF_TEL* che per prima cosa resetta tutti i registri e i selettori ed in seguito si pone nello stato di *tel_wait*. Dopo che lo stato viene mantenuto per un periodo di tempo impostato la macchina passa allo stato *tel_assign_pre* dove nei la macchina assegna nel quinto registro il simbolo corrispettivo agli imput dell'utente, qui ci sono tre possibili alternative:

- Se l'utente non ha inserito alcun punto o linea la macchina assegna al quinto registro un contenuto nullo, che equivale ad uno spazio, se la lettera precedente (contenuta nel quarto registro) non è a sua volta uno spazio, allora la macchina effettua uno shift passando il contenuto del secondo registro nel primo, del terzo nel secondo, e così via, resettando infine il contenuto dell'ultimo registro. Il risultato finale consiste nel visualizzare uno spazio sul display, che può esser utilizzato per separare due parole diverse.
- Se l'utente non ha inserito alcun punto o linea, ed anche il contenuto del quarto registro è uno spazio, la macchina non esegue nessuno shift e si limita solamente a resettare il contenuto del quinto registro, questo per evitare di visualizzare più di uno spazio, che risulterebbe inutile.
- Se l'utente ha inserito dei punti o delle linee invece, verrà visualizzata la lettera corrispondente nel digit più a sinistra.

Mentre si trova nello stato di *tel_wait*, la macchina controlla anche che non venga premuto *B4*, in quel caso alza il segnale di *Telegraph_STOP* e si pone nello stato di *idle*, mentre il controllo passa nuovamente alla *MSF_MENU*.

CONTATORI: *Cont*

Nel progetto i tempi risultano fondamentali, per esempio è proprio grazie alla diversa durata del segnale che si identifica la differenza fra linee e punti. I contatori vengono utilizzati proprio per poter identificare se un evento ha una durata maggiore o minore rispetto ad un limite, oppure per mantenere la macchina in uno stato per una durata di tempo fissata. Come riferimento si utilizza la frequenza interna e si incrementa una variabile finché questa non raggiunge un limite di tempo stabilito, in termini di numero di cicli.

```
architecture Behavioral of Cont is
begin
        process(CLK, CONT_RST)
        variable cnt : integer :=0;
        begin
            if rising_edge(clk) then
                    if (CONT_RST='1') then
                                    cnt:=0;
                                    CONT <= '0';
                    end if;
                    if (CONT_EN='1') then
                                    cnt:=cnt+1;
                                    if (cnt=timer) then
                                                    CONT <= '1';
                                    end if;
                    end if;
            end if;
            end process;
end Behavioral;
```

Per controllare ogni contatore si ha un segnale di enable ed uno di reset. Sono di questo tipo *CONTP* che viene utilizzato per stabilire il tempo di attesa prima di procedere all'assegnazione della lettera relativa alle informazioni nei registri; *CONT2* che stabilisce la differenza fra un punto e una linea (quando il contatore finisce viene anche segnalato visivamente in quanto si accendono i punti nei digit) e *CONT_stato* che permette alla macchina di rimanere in attesa in uno stato per un periodo di tempo preciso. Quando il contatore raggiunge la fine lo segnala assegnando un valore "alto" all'uscita.

REGISTRI: *Reg_Symbol e Reg_Digit*

I quattro ***Reg_Symbol*** sono dei semplici registri con un ingresso costituito da un vettore a due bit, un uscita di ugual tipo ed un segnale ad un singolo bit che funge da *enable* del registro. Questi registri sono stati predisposti per contenere i punti e le linee inserite dall'utente. Sono quattro in quanto ogni lettera è composta da un massimo di quattro elementi. È stata scelto di assegnare il valore *"00"* qualora il registro risulti vuoto, *"01"* se contiene un punto e *"11"* se contiene una linea. In questo modo alla macchina è possibile verificare se un registro contiene un valore o se è vuoto (e quindi è possibile scriverci), solo guardando il primo bit a destra.

```
architecture Behavioral of Reg_Symbol is
begin
process (REG_symbol_EN)
variable tmp_1 : std_logic_vector (1 downto 0);
begin
        if (REG_symbol_EN'event and REG_symbol_EN='1') then
            tmp_1 := WR;
        end if;
        Reg_Symbol_OUT <= tmp_1;
end process;
end Behavioral;
```

I ***Reg_Digit*** sono dei registri che contengono l'informazione in modo da poter accendere i digit e visualizzare l'informazione voluta in modo corretto. In ingresso vi è un selettore a due bit, un segnale singolo che funge da *enable*, tre ingressi ed un uscita un uscita a sette bit, uno per ogni segmento dei digit.

```
architecture Behavioral of Reg_Digit is
signal tmp_1 : std_logic_vector (6 downto 0);
begin
process (REG_DIGIT_EN, SEL, IN1_random, IN2_new,tmp_1)
begin
        if (RE_DIGIT_EN'event and REG_DIGIT_EN='1') then
        if (SEL="00") then tmp_1<="1111111"; end if;                    -- reset
        if (SEL="01") then tmp_1<=IN1_random; end if;                   -- random
        if (SEL="11") then tmp_1<=IN2_new; end if;
        if (SEL="10") then tmp_1<=IN3_new; end if;
        end if;
        Reg_DIGIT_OUT <= tmp_ 1;
end process;
end Behavioral;
```

Quando l'enable si abilita viene scritto nel registro il valore corrispondente all'ingresso che il selettore indica. Sono stati impostati i seguenti riferimenti: *"00"* per resettare il registro, *"01"* per scrivere all'interno del registro una lettera casuale, prendendo l'uscita del *generatore random; "11"* e *"10"* per leggere l'informazione da un altro registro, per esempio nel momento in cui si vuole effettuare uno shift delle lettere visualizzate.

SISTEMA ANTIBOUNCE: *Reg_Button e Contbutton*

Un problema riscontrato consisteva nella non corretta lettura dell'input effettuato per mezzo di bottoni e interruttori, infatti nel momento in cui si preme uno di questi è possibile che la tensione compia delle fluttuazioni prima di stabilizzarsi al valore alto (o basso), per questo una singola pigiatura del bottone può esser interpretata come una sequenza di due o più premute. Per risolvere il problema è stato implementato un sistema di antirimbalzo, composto da un registro (*Reg_Butto*n) che opera in questo modo:

```
architecture Behavioral of Reg_Button is
begin
process(clock, reset)
begin
 if(reset='1') then output<='0';
elsif rising_edge(clock) then output <= input;
end if;
end process;
end Behavioral;
```

in pratica sincronizza con il clock l'aggiornamento del segnale corrispondente al bottone, in questo modo si tagliano le fluttuazioni di frequenza maggiore. Inoltre associato a questo registro si trova un contatore*, CONTBUTTON*: se il segnale associato al bottone non rimane alto finché questo contatore non arriva alla fine, il segnale viene ignorato. Questo è un'ulteriore controllo, per evitare le fluttuazioni che possono aver durata maggiore di un colpo di clock.

COMPARATORI: *Comp_1, Comp_2 e Comp_space*

Il principale blocco combinatorio asincrono è ***COMP_1,*** un comparatore che riceve in ingresso le uscite dei *Reg_Digit*, ed in base a queste assegna la lettera opportuna. In questo modo ogni volta che uno degli ingressi cambia, in quanto è stato inserito un punto o una linea, il comparatore pone in uscita la sequenza di sette bit adatta per visualizzare la lettera sul digit.

```
architecture Behavioral of Comparator_1 is
begin
Confronto: process(COMP_1_IN_1,COMP_1_IN_2,COMP_1_IN_3,COMP_1_IN_4)
begin
if (COMP_1_IN_4(0) = '1') then -- 4 BIT
        if (COMP_1_IN_1(1) = '1') then
                if (COMP_1_IN_2(1) = '1') then
                        if (COMP_1_IN_3(1) = '1') then
                                if (COMP_1_IN_4(1) = '1') then
                                COMP_1_OUT <= "1110111";
                                -- quattro linee non esiste
                                else
                                COMP_1_OUT <= "1110111";
                                -- tre linee e un punto non esiste
                                end if;
                        else
                                if (COMP_1_IN_4(1) = '1') then
                            COMP_1_OUT <= "1010000"; -- Stampa Q
                                else
                                COMP_1_OUT <= "0100010"; -- Stampa Z
                                end if;
                        end if;
                else
                        if (COMP_1_IN_3(1) = '1') then
                                if (COMP_1_IN_4(1) = '1') then
                            COMP_1_OUT <= "0010001"; -- Stampa Y
                        else
                                COMP_1_OUT <= "0101100"; -- Stampa C
                                end if;
                        else
                                if (COMP_1_IN_4(1) = '1') then
                            COMP_1_OUT <= "0000001"; -- Stampa X
                else
                                COMP_1_OUT <= "0000101"; -- Stampa B
                                end if;
                        end if;
                end if;
        else
                if (COMP_1_IN_2(1) = '1') then
                        if (COMP_1_IN_3(1) = '1') then
                                if (COMP_1_IN_4(1) = '1') then
                            COMP_1_OUT <= "0011011"; -- Stampa J
                                else
                                COMP_1_OUT <= "1100000"; -- Stampa P
                                end if;
                        else
                                if (COMP_1_IN_4(1) = '1') then
                                COMP_1_OUT <= "1110111";
                                -- punto linea punto linea non esiste
                                else
                                COMP_1_OUT <= "0101101"; -- Stampa L
                                end if;
                        end if;
                else
                        if (COMP_1_IN_3(1) = '1') then
                                if (COMP_1_IN_4(1) = '1') then
                                COMP_1_OUT <= "1110111";
                                -- punto punto linea linea non esiste
                                else
                                COMP_1_OUT <= "1100100"; -- Stampa F
                                end if;
                        else
```

```
                                                if (COMP_1_IN_4(1) = '1') then
                                        COMP_1_OUT <= "0001111"; -- Stampa V
                                                else
                                                COMP_1_OUT <= "1000101"; -- Stampa H
                                                end if;
                                        end if;
                                end if;
        end if;
else
        if (COMP_1_IN_3(0) = '1') then -- 3 BIT
                                if (COMP_1_IN_1(1) = '1') then
                                        if (COMP_1_IN_2(1) = '1') then
                                                if (COMP_1_IN_3(1) = '1') then
                                        COMP_1_OUT <= "0000111"; -- Stampa O
                                                else
                                                COMP_1_OUT <= "0000100"; -- Stampa G
                                                end if;
                                        else
                                                if (COMP_1_IN_3(1) = '1') then
                                        COMP_1_OUT <= "1000001"; -- Stampa K
                                                else
                                                COMP_1_OUT <= "0000011"; -- Stampa D
                                                end if;
                                        end if;
                                else
                                        if (COMP_1_IN_2(1) = '1') then
                                                if (COMP_1_IN_3(1) = '1') then
                                                COMP_1_OUT <= "1110111";      -- Stampa W
                                                else
                                                COMP_1_OUT <= "1100111"; -- Stampa R
                                                end if;
                                        else
                                                if (COMP_1_IN_3(1) = '1') then
                                        COMP_1_OUT <= "0001001"; -- Stampa U
                                                else
                                                COMP_1_OUT <= "0010100"; -- Stampa S
                                                end if;
                                        end if;
                                end if;
                else
                                if (COMP_1_IN_2(0) = '1') then -- 2 BIT
                                        if (COMP_1_IN_1(1) = '0') then
                                                if (COMP_1_IN_2(1) = '1') then
                                                COMP_1_OUT <= "1000000"; -- Stampa A
                                                else
                                                COMP_1_OUT <= "1101101"; -- Stampa I
                                                end if;
                                        else
                                                if (COMP_1_IN_2(1) = '1') then
                                                COMP_1_OUT <= "1001000"; -- Stampa M
                                                else
                                                COMP_1_OUT <= "1000111"; -- Stampa N
                                                end if;
                                        end if;
                                else
                                        if (COMP_1_IN_1(0) = '1') then -- 1 BIT
                                                if (COMP_1_IN_1(1) = '1') then
                                                COMP_1_OUT <= "0100101"; -- Stampa T
                                                else
                                                COMP_1_OUT <= "0100100"; -- Stampa E
                                                end if;
                                        else
                                                COMP_1_OUT <= "1111111"; -- Stampa Spazio
```

```
                                        end if;
                                end if;
        end if;
end if;
end process;
end Behavioral;
```

come si nota l'intera logica si appoggia al fondamento che la macchina a stati finiti comincia a riempire il primo digit, per questo il controllo comincia da quello opposto in modo da scartare tutte le lettere che hanno una lunghezza maggiore rispetto al numero di simboli inseriti. In seguito analizza se i simboli sono punti o linee e associa l'uscita opportuna.

Un altro tipo di comparatore è ***COMP_2***, questo riceve in ingresso due stringe di sette bit e le confronta, se sono uguali alza l'uscita, in caso contrario la mantiene a zero.

```
architecture Behavioral of Comparator_2 is
    begin
process (COMP_2_IN_1, COMP_2_IN_2)
begin
        if (COMP_2_IN_1 = COMP_2_IN_2) then
                COMP_2_RESULT <= '1';
        else
                COMP_2_RESULT <= '0';
        end if;
end process;
end Behavioral;
```

questo tipo di comparatore è utilizzato per confrontare il contenuto di due registri collegati ai digit, per esempio nella modalità apprendimento permette alla macchina a stati finite di capire se l'utente ha inserito correttamente la lettera.

Simile a questo comparatore è ***COMP_SPACE***, che viene utilizzato dalla *MSF_TEL* per evitare di stampare sul display troppi spazi, infatti confronta se il contenuto del quarto e quinto registro è uguale, ed allo stesso tempo contengono entrambi una stringa *"1111111"* (corrisponde al digit spento in quanto la logica è inversa).

```
architecture Behavioral of Comparator_Space is
begin
process (COMP_S_IN_1, COMP_S_IN_2)
begin
if (COMP_S_IN_1 = COMP_S_IN_2) then
  if (COMP_S_IN_1 = "1111111") then
        COMP_S_RESULT <= '1';
        else
        COMP_S_RESULT <= '0';
        end if;
else
         COMP_S_RESULT <= '0';
end if;
end process;
end Behavioral;
```

RANDOM: *Cont_random, RND e Selec_Generator_Random*

Nella modalità di apprendimento è utile che venga generata in modo casuale una lettera, in modo tale che l'utente possa esercitarsi nel digitare correttamente il codice morse. Per render possibile questa cosa si utilizzano tre diversi blocchi. Un blocco che genera un clock rallentato, ovvero ***Cont_Random***

```
architecture Behavioral of Cont_Random is
signal count_sig_Random : std_logic_vector((counter_width_Random-1) downto 0);
begin
 process(CLK)
```

```
begin
 if (rising_edge(CLK)) then
            if (count_sig_Random = (frequency_ratio_Random-1)) then
   count_sig_Random <= (others => '0');
  else
   count_sig_Random <= count_sig_Random + 1;
  end if;

  if (count_sig_Random <= (frequency_ratio_Random/100*Duty_cycle_Random-1)) then
   Random_EN <= '1';
  else
   Random_EN <= '0';
  end if;
 end if;
end process;
end Behavioral;
```

questo segnale abilita l'incremento di un selettore a sei bit, in modo da poter scorrere tutte le lettere, il blocco in questione è ***Select_Random_Generator*** e si compone di questo processo

```
architecture Behavioral of SELECT_GENERATOR_RANDOM is
signal x: STD_LOGIC_VECTOR(5 downto 0);
begin
    process(Random_EN)
        begin
            if rising_edge(Random_EN) then
              x<=x+"000001";
              if (x = (32)) then
                    x <= (others => '0');
              end if;
            end if;
    end process;
    SEL_RND<=x;
end Behavioral;
```

il selettore è posto in ingresso ad un blocco combinatorio che assegna all'uscita un vettore di sette bit, uno per ogni lettera già codificata in modo opportuno per poter esser visualizzata, questa parte è svolta dal blocco ***RND***:

```
architecture Behavioral of RND is
begin
with SEL_SRM select
        COMP_1_OUT_RND <= "1000000" when "000000",
          "0000101" when "000001",
          "0101100" when "000010",
          "0000011" when "000011",
          "0100100" when "000100",
          "1100100" when "000101",
          …
          "1000000" when others;
end Behavioral;
```

facendo così scorrere velocemente il segnale di enable generato dal *Cont_Random* si ottiene un uscita da *RND* che cambia velocemente, perciò quando si attiva la modalità apprendimento, viene scelta in modo casuale l'uscita.

MENU SCORREVOLE: Select_Generator e *Shift_Register_Menu*

Anche nel caso del menu scorrevole la struttura è molto simile alla precedente, si ha un selettore che si resetta con un apposito segnale, in modo da far partire il menu dall'inizio ogni volta che si accede allo stato *menu*. Un

segnale di *enable* che abilita l'incremento del selettore e un blocco, ***Shift_Register_Menu***, nel quale è salvato l'intero menù da visualizzare e in base al selettore le quattro uscite cambiano e ognuna porta l'informazione ad un digit.

MULTIPLEXER: *Multiplexer, Mux_aiuto e Multiplexer_Mode*

I ***multiplexer*** son anch'essi componenti combinatori e asincroni, ricevono in ingresso un selettore di due bit, due ingressi e un uscita formati da sette bit. Sono dei multiplatori per il segnale che si collega direttamente ai digit. Prelevano il segnale dai *reg_digit* oppure dallo *shift_register_menu*, infatti solitamente viene sempre visualizzato il contenuto dei registri, ad eccezione della condizione in cui *MSF_MENU* si trovi nello stato *menu* e quindi sul display scorre il menu che illustra all'utente le modalità disponibili.

```
architecture Dataflow of Multiplexer is
begin
with SEL select
        SIG_O <= IN1_menu when '0',      -- Menu
        IN2_new when '1',                -- New
        "1111111" when others;
end Dataflow;
```

I ***multiplexer_mode*** sono dei componenti necessari vista la presenza di più macchine a stati finiti. Infatti se si intende controllare per esempio l'*enable* di un registro da più macchine, è necessario che giunga al registro solo il segnale corretto, della macchina che in quel momento è attiva. Per questo vi è un selettore a due bit secondo il quale all'uscita viene assegnato uno dei tre segnali provenienti dalle macchine a stati finiti.

```
architecture Behavioral of Multiplexer_Mode is
signal app : std_logic;
begin
process(CLK,Sig_APP,Sig_TEL,Sig_MEN,SEL)
begin
if (rising_edge(CLK)) then
            if SEL="00" then
            app<=Sig_MEN;
            end if;
            if SEL="01" then
            app<=Sig_TEL;
            end if;
            if SEL="10" then
            app<=Sig_APP;
            end if;
            if SEL="11" then
            app<=Sig_MEN;
            end if;
end if;
end process;
O<=app;
end Behavioral;
```

il selettore è l'unione dei due segnali che abilitano rispettivamente le due macchine APP e TEL

```
SEL_MODE(0) <=TEL_EN_s;
SEL_MODE(1) <=APP_EN_s;
```

È presente anche il segnale di clock al quale vengono sincronizzati i cambiamenti dell'uscita altrimenti si verificavano problemi di glitch e di sincronismo che portavano a degli errori nella scrittura dei registri.

DISPLAY LAMPEGGIANTE: *Digit_Blinker*

Questo blocco è stato creato per far lampeggiare il display, per esempio per segnalare all'utilizzatore, in modalità apprendimento, che la lettera inserita corrisponde a quella creata in modo random. Si ha un segnale in ingresso *Blink_MSF* che viene attivato dalla macchina a stati finiti, e funge da *enable*. In ingresso e in uscita abbiamo due vettori di quattro bit che corrispondono al segnale che porta l'alimentazione ai quattro digit, in questo modo controllando questo segnale sarà possibile il lampeggio. L'ultimo segnale in ingresso è *Blink_EN_s*, generato da un divisore di frequenza, che scandirà gli intervalli di lampeggio, in quanto il clock era inutilizzabile per questo scopo poiché troppo veloce.

```
architecture Behavioral of Digit_Blinker is
signal x: STD_LOGIC_VECTOR(3 downto 0);
signal y: STD_LOGIC_VECTOR(1 downto 0);
begin
  process(Blink_MSF,enDIGIT_S,Blink_EN_s,y)
                        begin
                        if Blink_MSF='0' then
                        x <= enDIGIT_S;
                        else
                        if rising_edge(Blink_EN_s) then
    y<=y+"01";
    end if;
              x(0) <= enDIGIT_S(0) OR y(0);
              x(1) <= enDIGIT_S(1) OR y(0);
              x(2) <= enDIGIT_S(2) OR y(0);
              x(3) <= enDIGIT_S(3) OR y(0);
                        end if;
            end process;
            enDIGIT_out<=x;
end Behavioral;
```

poiché i digit lavorano a logica inversa, l'or che troviamo equivale ad un and, i display saranno accesi solo se i due segnali risultano "bassi", mentre se uno dei due, o entrambi sono alti il display è spento.

AIUTO: *Mux_Aiuto e Aiuto_Help*

All'interno della modalità apprendimento è possibile fornire un aiuto all'utilizzatore nel caso in cui questo lo richiedesse. Per far ciò è sufficiente azionare il primo selettore a sinistra e la macchina si porta ad uno stato apposito definito di *help*, qui rimane per tutto il tempo che il selettore è attivo. Il blocco all'esterno riceve in ingresso questo segnale e agisce da multiplatore. Quando il segnale è basso collega i led direttamente al contenuto dei registri che contengono i punti e le linee che l'utente inserisce, quando il segnale è alto invece assegna all'uscita il vettore di otto bit proveniente dal blocco di aiuto.

```
architecture Dataflow of Mux_aiuto is
begin
with SEL select
     SIG_O <= IN1_menu when '0',     -- Menu
              IN2_app when '1',      -- App
              "00000000" when others;
end Dataflow;
```

Il blocco ***Aiuto_Help*** riceve in ingresso il contenuto del registro collegato al primo digit, ovvero della lettera da indovinare, e associa l'uscita corrispondente in modo da visualizzare sui led i punti e le linee associati alla lettera visualizzata.

```
architecture Behavioral of AIUTO_HELP is
```

```
begin
  with SIG_DIGIT_1 select
    LED_D <="10110000" when "1000000", --A
                "11101010" when "0000101", --B
                "11101110" when "0101100", --C
                "11101000" when "0000011", --D
                …
                 "00000000" when others;
end Behavioral;
```

UTILIZZO DEI DIGIT: Counter4, *Decoder, FrequencyDivider e Multiplexer_Dig*

Vista la particolare struttura della scheda è necessario adottare degli accorgimenti per visualizzare lettere diverse sui quattro digit. Infatti la scheda è dotata di un'unica linea di otto bit che porta le informazioni ai digit (in modo da accendere determinati segmenti) e quattro linee separate che giungono rispettivamente ad ogni digit e portano l'alimentazione. Se si accendessero tutti i digit e si alzerebbe opportunamente uno o più segnali degli otto bit comuni, si vedrebbe la stessa cosa su tutti i digit. Per evitare ciò si accende uno alla volta ogni digit ed al contempo si setta la linea comune in modo da far visualizzare l'opportuna informazione su ogni digit, se si velocizza il tutto con un adeguato tempo di refresh i digit sembreranno sempre accesi e con diverse informazioni ogniuno.
Questo blocco si compone di un divisore di frequenza che fornisce un tempo di refresh atto allo scopo

```
architecture Behavioral of FrequencyDivider is
 signal counter_sig : std_logic_vector((counter_width-1) downto 0);
begin
 process(reset, clock_in)
 begin
            if (reset = '1') then
   counter_sig <= (others => '0');
   clock_out <= '0';
  elsif (rising_edge(clock_in)) then
                    if (counter_sig = (frequency_ratio-1)) then
    counter_sig <= (others => '0');
   else
    counter_sig <= counter_sig + 1;
   end if;
   if (counter_sig <= (frequency_ratio/100*Duty_cycle-1)) then
    clock_out <= '1';
   else
    clock_out <= '0';
   end if;
  end if;
 end process;
end Behavioral;
```

questo "clock rallentato" incrementa allo stesso tempo un selettore che si resetta automaticamente una volta arrivato alla fine

```
architecture Behavioral of Counter4 is
signal x: STD_LOGIC_VECTOR(1 downto 0);
begin
        process(clock_div,reset)
                begin
                        if(reset='1') then x<=(others=>'0');
                        elsif rising_edge(clock_div) then
                                x<=x+"01";
                        end if;
        end process;
        outCONT<=x;
end Behavioral;
```

il selettore viene decodificato e fornisce in uscita un vettore di quattro bit al quale corrisponde l'alimentazione fornita ad ogni singolo digit

```
architecture Behavioral of Decoder1 is
begin
        with I select
                O <= "0111" when "00",
                        "1011" when "01",
                        "1101" when "10",
                        "1110" when "11",
                        "1111" when others;
end Behavioral;
```

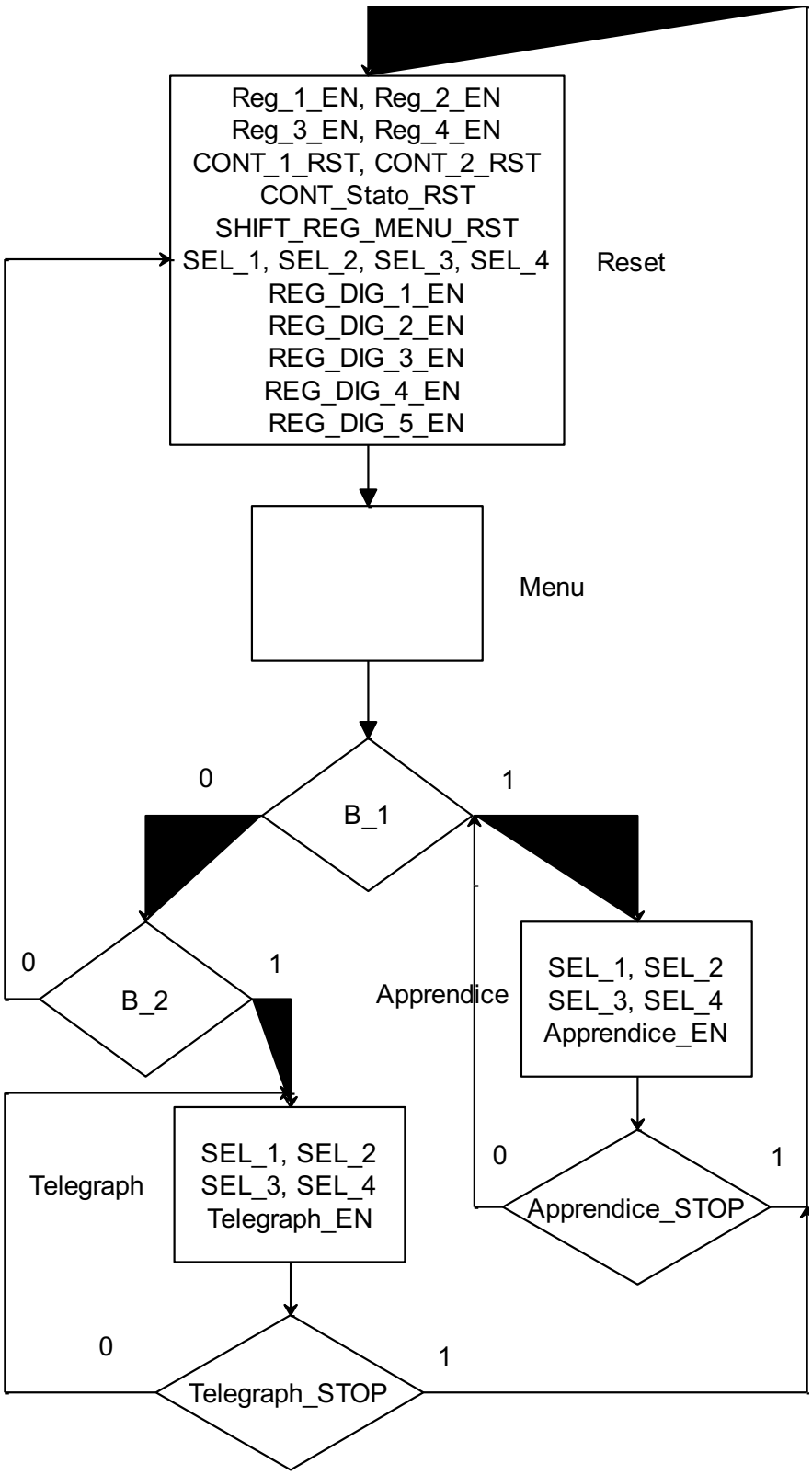
coordinato a questo il multiplexer riceve in ingresso il selettore e pone in uscita l'informazione che si intende visualizzare sul singolo digit che in quell'istante viene attivato

```
architecture Dataflow of Multiplexer_dig is
begin
        with SEL select
                O<=Sig_1 when "00",
                  Sig_2 when "01",
                  Sig_3 when "10",
                          Sig_4 when "11",
                  (others=>'0') when others;
end Dataflow;
```

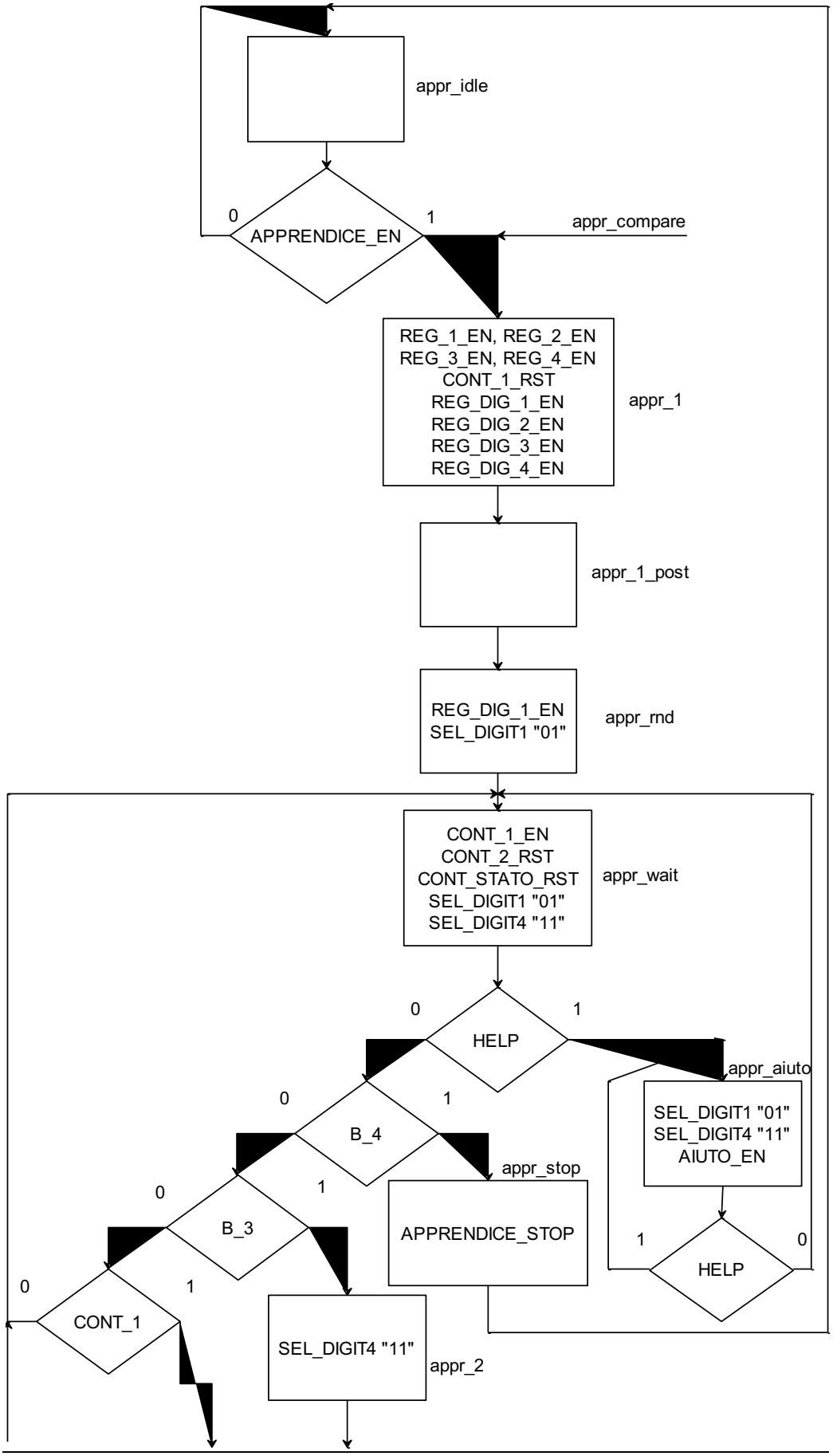
DIAGRAMMI ASM

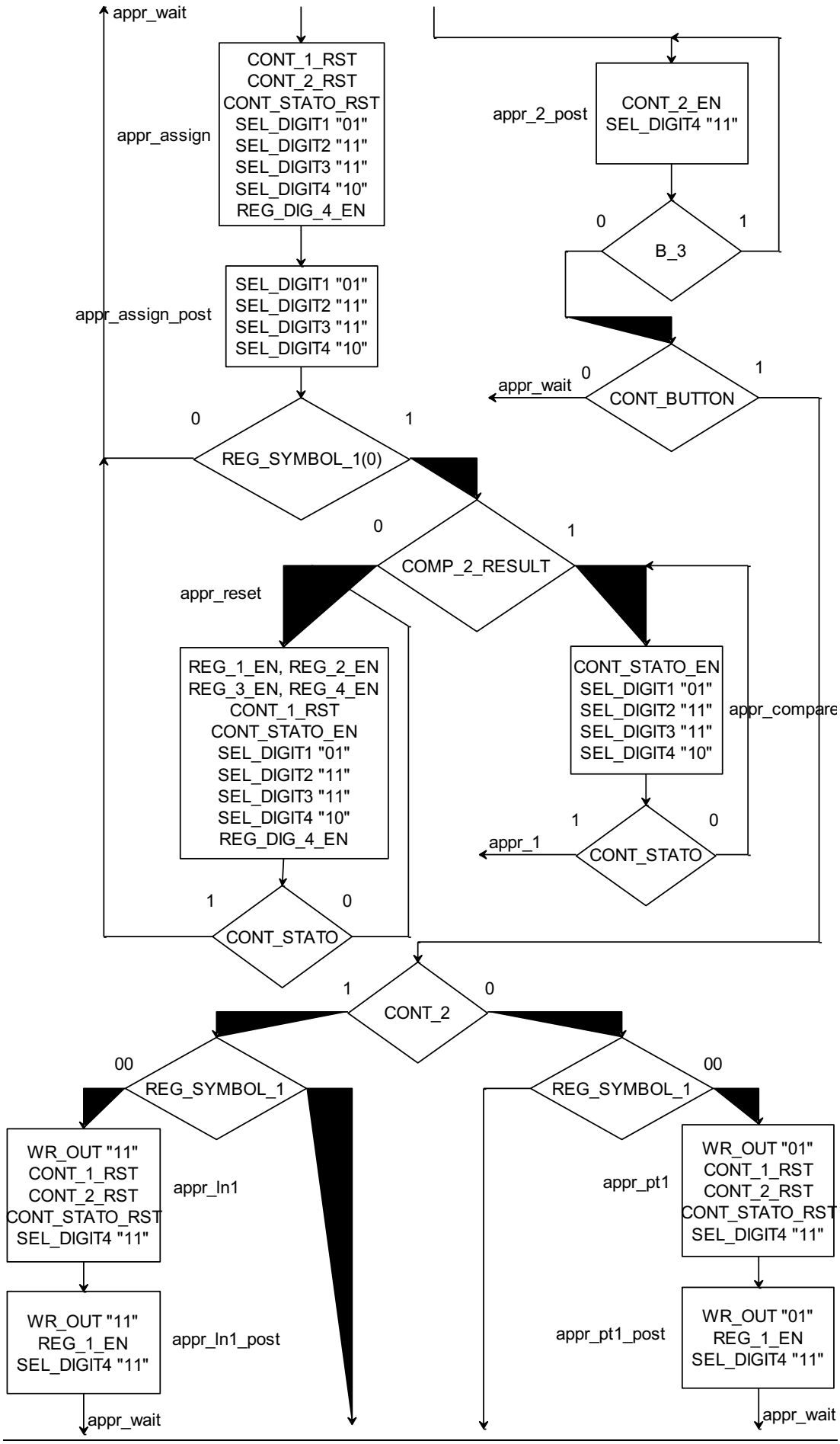
Di seguito sono elencati i diagrammi ASM per le 3 macchine a stati finiti che abbiamo utilizzato nel progetto:

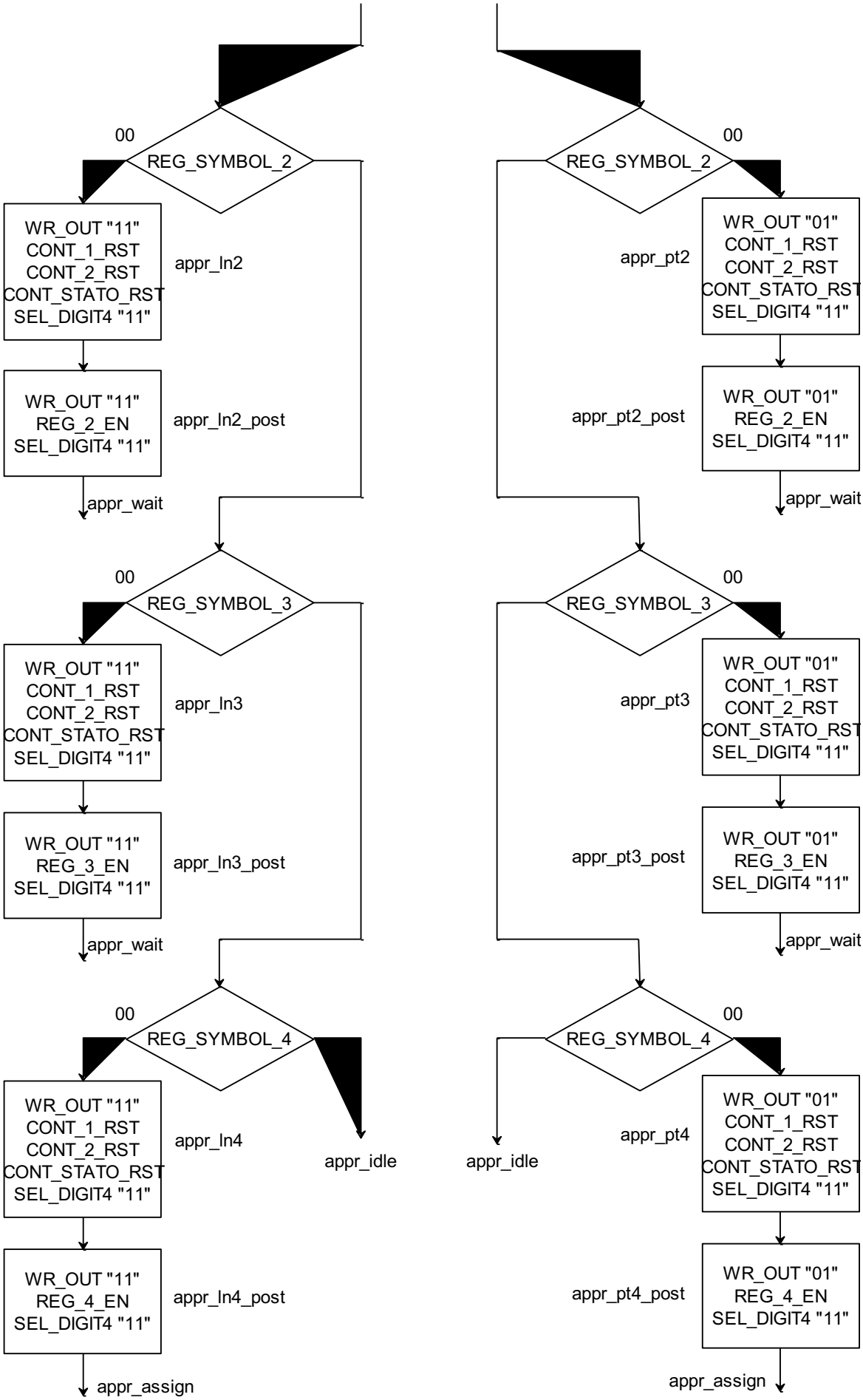
MSF Menu:



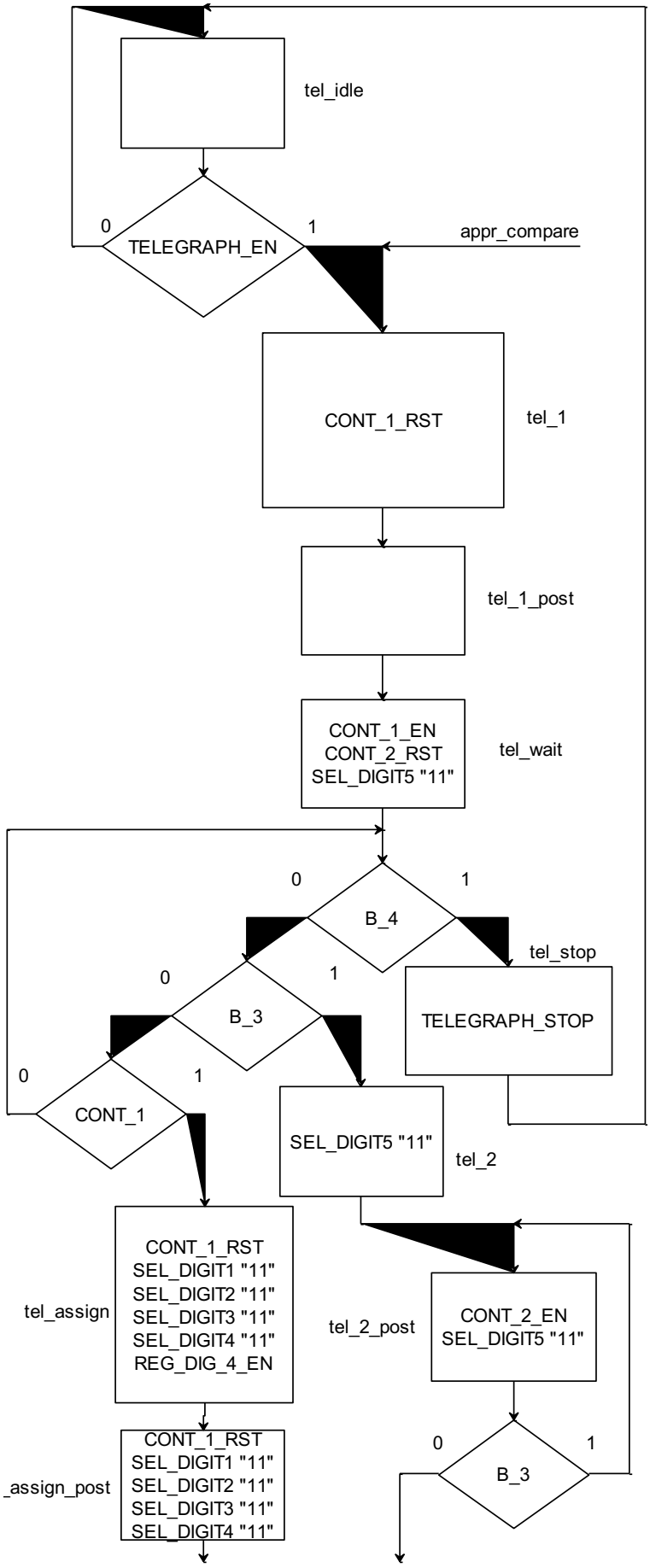
MSF Apprendimento:

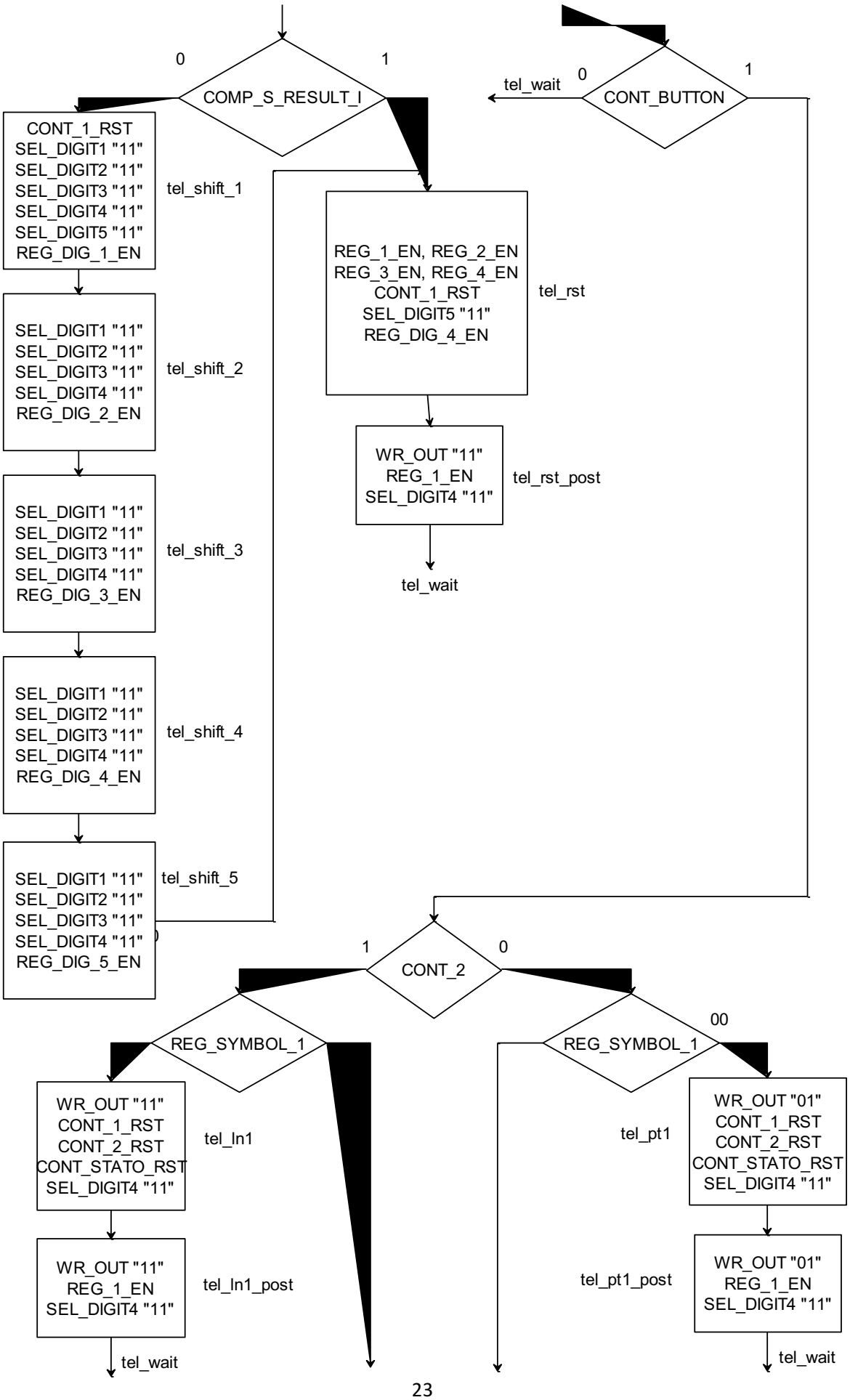


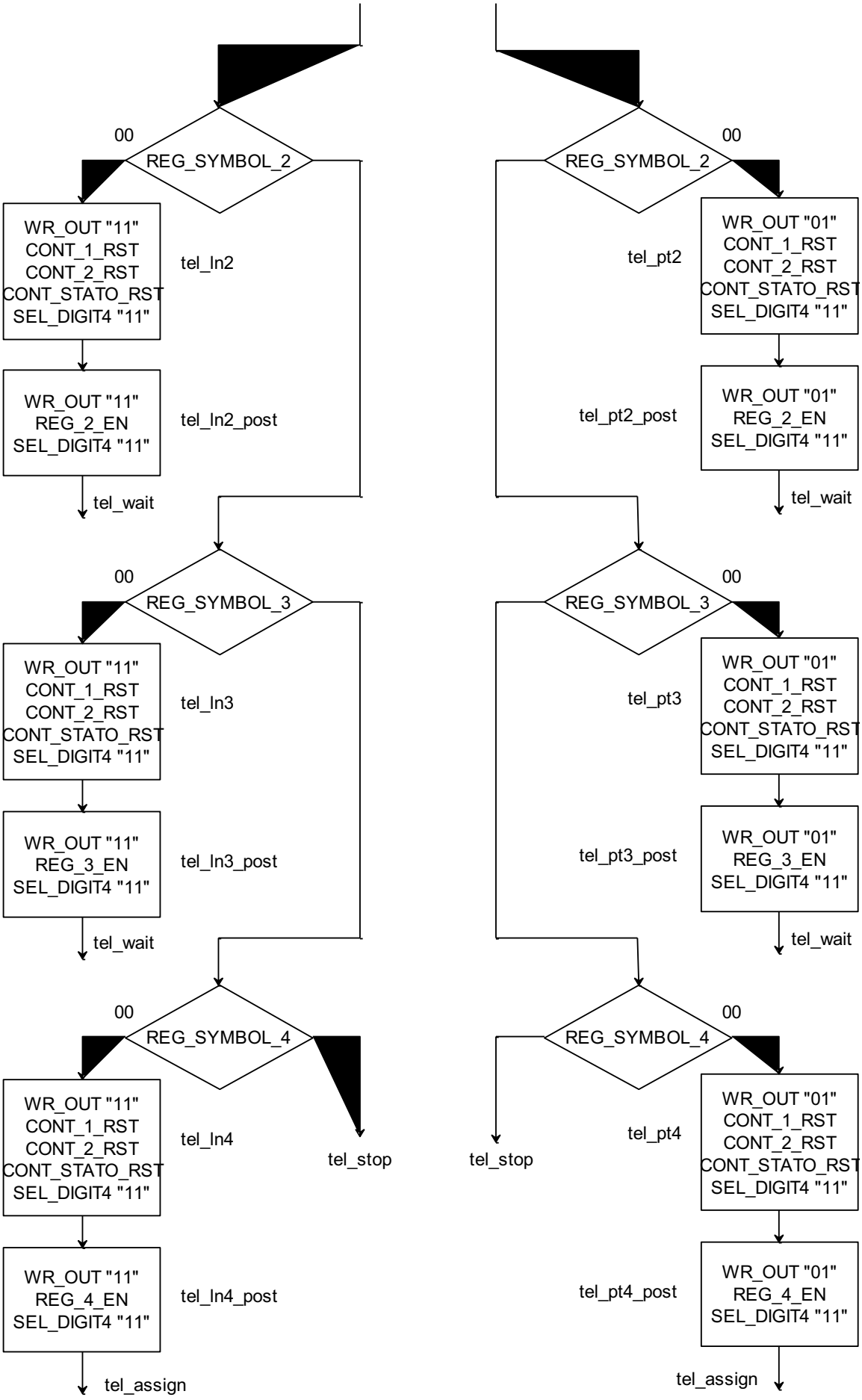

appr_wait
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT1 "01"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "10"
REG_DIG_4_EN
appr_assign
SEL_DIGIT1 "01"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "10"
appr_assign_post
REG_SYMBOL_1(0)
0
1
CONT_2_EN
SEL_DIGIT4 "11"
appr_2_post
B_3
CONT_BUTTON
appr_wait
COMP_2_RESULT
appr_reset
REG_1_EN, REG_2_EN
REG_3_EN, REG_4_EN
CONT_1_RST
CONT_STATO_EN
SEL_DIGIT1 "01"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "10"
REG_DIG_4_EN
CONT_STATO
CONT_STATO_EN
SEL_DIGIT1 "01"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "10"
appr_compare
appr_1
CONT_2
00
REG_SYMBOL_1
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_ln1
WR_OUT "11"
REG_1_EN
SEL_DIGIT4 "11"
appr_ln1_post
appr_wait
appr_pt1
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_pt1_post
WR_OUT "01"
REG_1_EN
SEL_DIGIT4 "11"


00
REG_SYMBOL_2
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_ln2
WR_OUT "11"
REG_2_EN
SEL_DIGIT4 "11"
appr_ln2_post
appr_wait
00
REG_SYMBOL_3
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_ln3
WR_OUT "11"
REG_3_EN
SEL_DIGIT4 "11"
appr_ln3_post
appr_wait
00
REG_SYMBOL_4
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_ln4
appr_idle
WR_OUT "11"
REG_4_EN
SEL_DIGIT4 "11"
appr_ln4_post
appr_assign
00
REG_SYMBOL_2
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_pt2
WR_OUT "01"
REG_2_EN
SEL_DIGIT4 "11"
appr_pt2_post
appr_wait
00
REG_SYMBOL_3
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_pt3
WR_OUT "01"
REG_3_EN
SEL_DIGIT4 "11"
appr_pt3_post
appr_wait
00
REG_SYMBOL_4
appr_idle
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
appr_pt4
WR_OUT "01"
REG_4_EN
SEL_DIGIT4 "11"
appr_pt4_post
appr_assign

MSF Telegrafo:




COMP_S_RESULT_I
0
1
CONT_1_RST
SEL_DIGIT1 "11"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "11"
SEL_DIGIT5 "11"
REG_DIG_1_EN
tel_shift_1
SEL_DIGIT1 "11"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "11"
REG_DIG_2_EN
tel_shift_2
SEL_DIGIT1 "11"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "11"
REG_DIG_3_EN
tel_shift_3
SEL_DIGIT1 "11"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "11"
REG_DIG_4_EN
tel_shift_4
SEL_DIGIT1 "11"
SEL_DIGIT2 "11"
SEL_DIGIT3 "11"
SEL_DIGIT4 "11"
REG_DIG_5_EN
tel_shift_5
REG_1_EN, REG_2_EN
REG_3_EN, REG_4_EN
CONT_1_RST
SEL_DIGIT5 "11"
REG_DIG_4_EN
tel_rst
WR_OUT "11"
REG_1_EN
SEL_DIGIT4 "11"
tel_rst_post
tel_wait
CONT_BUTTON
tel_wait
0
1
CONT_2
1
0
REG_SYMBOL_1
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_ln1
WR_OUT "11"
REG_1_EN
SEL_DIGIT4 "11"
tel_ln1_post
tel_wait
REG_SYMBOL_1
00
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_pt1
WR_OUT "01"
REG_1_EN
SEL_DIGIT4 "11"
tel_pt1_post
tel_wait


00
REG_SYMBOL_2
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_ln2
WR_OUT "11"
REG_2_EN
SEL_DIGIT4 "11"
tel_ln2_post
tel_wait
00
REG_SYMBOL_3
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_ln3
WR_OUT "11"
REG_3_EN
SEL_DIGIT4 "11"
tel_ln3_post
tel_wait
00
REG_SYMBOL_4
WR_OUT "11"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_ln4
tel_stop
WR_OUT "11"
REG_4_EN
SEL_DIGIT4 "11"
tel_ln4_post
tel_assign
00
REG_SYMBOL_2
tel_pt2
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_pt2_post
WR_OUT "01"
REG_2_EN
SEL_DIGIT4 "11"
tel_wait
00
REG_SYMBOL_3
tel_pt3
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_pt3_post
WR_OUT "01"
REG_3_EN
SEL_DIGIT4 "11"
tel_wait
00
REG_SYMBOL_4
tel_stop
tel_pt4
WR_OUT "01"
CONT_1_RST
CONT_2_RST
CONT_STATO_RST
SEL_DIGIT4 "11"
tel_pt4_post
WR_OUT "01"
REG_4_EN
SEL_DIGIT4 "11"
tel_assign

DESIGN SUMMARY

Top_Level_Entity Project Status (08/25/2010 - 11:44:14)			
Project File:	ESD.xise	**Parser Errors:**	No Errors
Module Name:	Top_Level_Entity	**Implementation State:**	Programming File Generated
Target Device:	xc3s200-4ft256	**Errors:**	No Errors
Product Version:	ISE 12.1	**Warnings:**	11 Warnings (11 new, 0 filtered)
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	362	3,840	9%	
Number of 4 input LUTs	871	3,840	22%	
Number of occupied Slices	547	1,920	28%	
Number of Slices containing only related logic	547	547	100%	
Number of Slices containing unrelated logic	0	547	0%	
Total Number of 4 input LUTs	967	3,840	25%	
Number used as logic	871			
Number used as a route-thru	96			
Number of bonded IOBs	27	173	15%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	3.38			

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Wed Aug 25 11:42:39	0	1 Warning (1 new, 0	2 Infos (2 new, 0

		2010		filtered)	filtered)
Translation Report	Current	Wed Aug 25 11:42:55 2010	0	0	0
Map Report	Current	Wed Aug 25 11:43:11 2010	0	0	2 Infos (2 new, 0 filtered)
Place and Route Report	Current	Wed Aug 25 11:43:47 2010	0	10 Warnings (10 new, 0 filtered)	3 Infos (3 new, 0 filtered)
Power Report					
Post-PAR Static Timing Report	Current	Wed Aug 25 11:43:55 2010	0	0	5 Infos (5 new, 0 filtered)
Bitgen Report	Current	Wed Aug 25 11:44:08 2010	0	0	1 Info (1 new, 0 filtered)

Secondary Reports		[-]
Report Name	**Status**	**Generated**
WebTalk Report	Current	Wed Aug 25 11:44:09 2010
WebTalk Log File	Current	Wed Aug 25 11:44:14 2010

Notiamo che nel report “Place and Route” sono presenti 10 Warnings che però, come riportato sul sito della Xilinx stessa, sono da ignorare poiché **falsi warning**:

Description

Keywords*: clock*

PAR successfully routed my design, but the PAR report file (.par) contains warning messages about clock skew. What is the problem?

"WARNING:Route:447 - CLK Net:XYZ may have excessive skew because
CLK pins failed to route using a CLK template."

Solution
*In some cases prior to ISE revision 9.1i, this clock skew warning was issued as a **false warning**. You can safely ignore the warning message if the Clock Report in the .par file does not report any skew problems for the clock net listed in the message.*