Laboratorio di Sistemi Digitali Integrati

Realizzazione di un analizzatore di riflessi

Studenti:

Leonardo Bellettini 260347
Edoardo Bollea 262968
Melissa Valloni 261799
Alberto Vaudagna 265418

Professore:

Massimo Ruo Roch

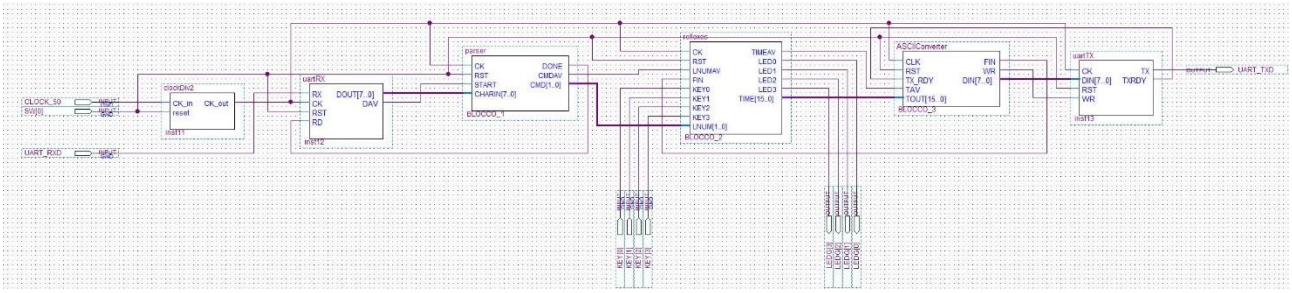
Sistemi Digitali Integrati 2018/2019

Sommario

L’analizzatore di riflessi

Il progetto consiste nella creazione, su un’ FPGA, di un dispositivo in grado di analizzare i riflessi di un individuo. Un utente, da terminale esterno, invia la stringa “LX” con 1≤X≤4. Il circuito, sintetizzato sull’ FPGA collegata al terminale con una connessione seriale RS232, accende il led corrispondente al numero ricevuto dopo un tempo casuale compreso tra 10000 e 20000 millisecondi (in modo da rendere il tutto più imprevedibile e testare la reazione in maniera più veritiera). A questo punto un timer interno cronometra il tempo che intercorre tra l’accensione del led e la pressione del pulsante corrispondente (fino ad un massimo di 65535 millisecondi) e invia il tempo al terminale iniziale sempre sfruttando il protocollo RS232 e nel formato “Thhhh” dove hhhh è un numero in esadecimale compreso tra 0 e 65535.

Datapath



Lo schema generale qui riportato è quello usato per testare sulla DE2 il circuito: compare pertanto un dispositivo clockDiv2 non necessario ai fini dell’analizzatore di riflessi, ma introdotto poiché il clock interno della scheda ha una frequenza di 50Mhz, contro i 25Mhz richiesti al dispositivo progettato.

La progettazione del circuito è partita dai blocchetti di trasmissione e ricezione, come richiesto. Saranno infatti i primi ad essere descritti singolarmente.

Successivamente si è pensato a come strutturare il resto del sistema: si è diviso il circuito in più sottoblocchi cercando un approccio modulare che rispettasse la semantica di ogni blocco rispetto alla sua funzione.

Vi è quindi un blocco che decodifica i comandi ricevuti da seriale, uno che gestisce le operazioni della macchina (misure di tempo e gestione dei LED) e uno che converte i risultati prodotti dal circuito in un formato adatto alla specifica.

Per ogni blocco sono stati definiti i segnali di interfacciamento con l’esterno e quali fossero i loro timing prima di passare allo sviluppo di ciascuno.

UART RX

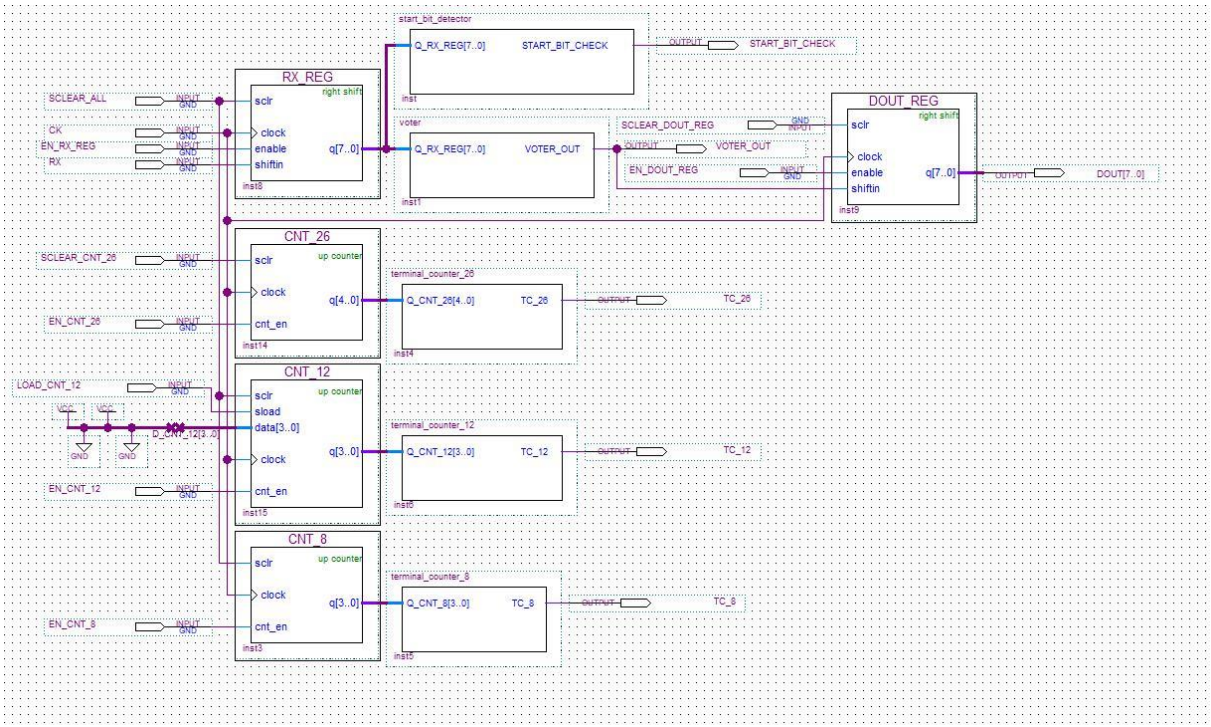
La progettazione del blocchetto di ricezione UART, basato sullo standard RS232, è partita dalle specifiche sull'interfaccia e sul timing fornite, che non verranno pertanto riportate.

L'unica variazione apportata riguarda il segnale di reset, non presente inizialmente, aggiunto poiché fondamentale per il corretto funzionamento di ogni blocco sequenziale.

Dopo aver compreso quale fosse il funzionamento globale del sistema in cui questo blocco avrebbe dovuto lavorare sono state apportate alcune modifiche in un secondo momento, motivo per cui sono presenti alcune differenze rispetto alla prima versione consegnata.

UART RX: Datapath

Questo è lo schematico del datapath che è stato ricavato:



Gli elementi che compongono il datapath saranno ora spiegati, funzionalmente, in ordine di uso temporale, cioè dal primo che viene usato durante l'operazione del dispositivo all'ultimo.

CNT_26

Il primo oggetto a entrare in funzione è un contatore a modulo 32. La sua funzione è quella di dividere il clock di ingresso (25MHz) affinché la sua frequenza sia 8 volte maggiore di quella di trasmissione del protocollo RS232 scelto, per poter sovracampionare i segnali in arrivo al ricevitore. Poiché è sconsigliabile usare segnali diversi come clock, la gestione di questo contatore è affidata alla control unit, in modo tale che i segnali che devono cambiare quando si raggiunge il valore di fine conta mutino, ma sempre in corrispondenza del fronte di salita del clock principale. Per ottenere questo è stata introdotta una rete combinatoria che segnala quando il contatore sta per finire di contare: avendo presupposto in fase iniziale di mandare tutti i segnali del contatore all'unità di controllo, si è ricavato quali ingressi della porta AND che genera il terminal count a partire dal time diagram. Si ottiene che il segnale

di fine conta deve essere alto nel momento in cui il valore di uscita del contatore raggiunge quello desiderato: dovendo dividere per 27 il clock e partendo a contare da 0, il terminal count sale quando il segnale vale 25. Questo stesso ragionamento verrà applicato a tutti gli altri contatori e non verrà più riportato.

RX_REG

Il registro di ingresso, che dovrà contenere i campioni del segnale ricevuto. Ogni campione, ricevuto su un ingresso seriale di tipo right-shift, viene salvato in corrispondenza del segnale di fine conta di CNT_26: questo processo non si ferma mai, per tanto in qualsiasi situazione si trovi il sistema il contatore continuerà ad avanzare e il registro a campionare i dati. Inizialmente questa funzione non era stata correttamente implementata: il registro veniva resettato una volta reso disponibile un dato in uscita. Tuttavia, in questo modo non era possibile riconoscere una transizione STOP-START bit.

START_BIT_DETECTOR

Questo blocco, descritto in vhdl, è una semplice porta AND a 8 ingressi che controlla quando le uscite del registro di ingresso siano 4 a '1' e 4 a '0': ipotizzando di essere in idle, si riconosce in questo modo la condizione di inizio trasmissione, e il sistema potrà evolvere.

CNT_12

Subito dopo aver individuato una transizione di inizio trasmissione, questo contatore inizia ad essere incrementato. Il suo scopo è quello di traslare la finestra temporale di campioni centrandola su un intero bit: 4 bit dello start bit sono già stati campionati quando questo contatore entra in funzione, mancano quindi 4 campioni dello start bit e gli 8 del primo bit ricevuto. L'abilitazione di questo contatore è gestita insieme al registro di ingresso, quindi al salire del terminal count di CNT_26. Anche in questo caso il valore del terminal count è stato ricavato a partire dal time diagram, visibile nella prossima sezione.
Dopo aver assolto al compito di inquadrare i campioni, CNT_12 diventa responsabile di contare quanti campionamenti sono stati eseguiti per ciascun bit ricevuto: un'opzione di load è stata aggiunta per poter caricare il valore costante 5 ("0101"), in questo modo il segnale di fine conta comunica quando controllare il contenuto del registro di ingresso e quando salvare un bit nel registro di uscita.

VOTER

Il controllo sopra citato viene eseguito dal voter. I 5 campioni centrali del bit ricevuto vengono analizzati in modo combinatorio per decidere se vi siano più '1' o più '0'. La presenza del voter serve a ridurre le possibilità che un campione errato incida sul risultato finale. L'uscita del voter, oltre che essere l'ingresso seriale del registro di uscita, è anche mandata alla control unit per permetterle di verificare che il decimo bit ricevuto sia effettivamente un '1' (stop bit) e che quindi la sequenza sia valida.

CNT_8

L'ultimo contatore salva il numero di bit ricevuti e salvati nel registro di uscita. Esso viene incrementato ogni volta che CNT_12 finisce di contare. Anche in questo caso si rimanda al time diagram per capire quale valore è stato scelto per il circuito di terminal count.

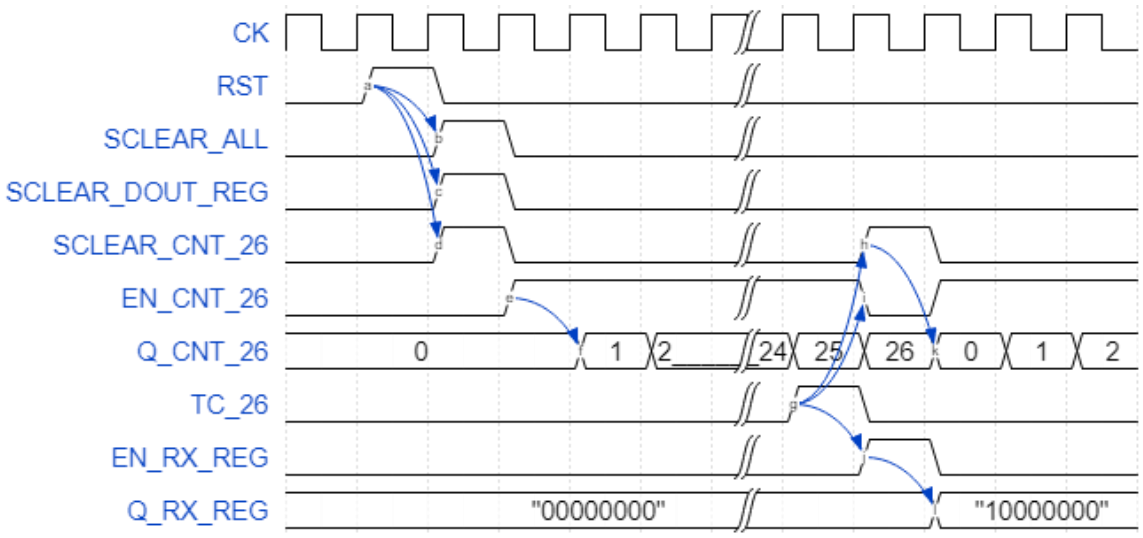
DOUT_REG

Infine, si trova il registro di uscita: anch'esso è uno right-shift-register e viene abilitato a shiftare al suo interno il segnale di uscita del voter ogni volta che terminal count del contatore modulo 12 viene asserito. Quando il dato viene letto dall'esterno, il registro viene resettato a tutti 0.

UART RX: Time Diagram

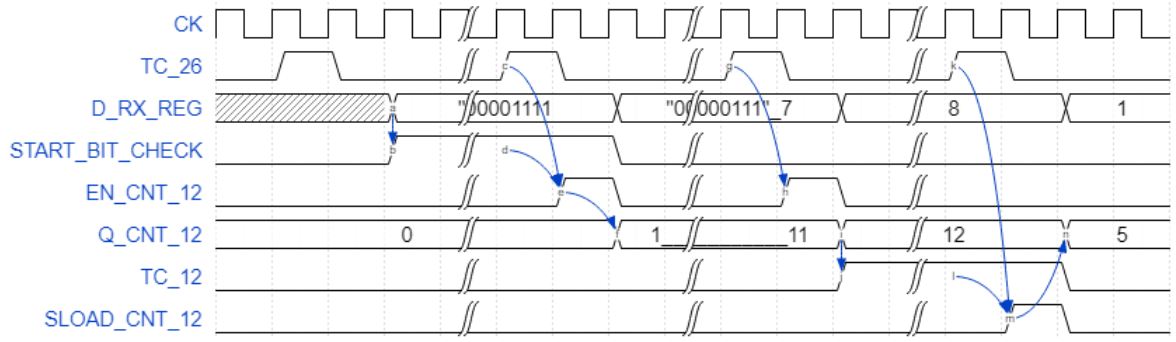
Il diagramma temporale di questo blocco è stato suddiviso in più parti per semplificarne la comprensione. Ciascuna verrà commentata singolarmente. Non sono stati rappresentati tutti i possibili casi in cui il sistema può trovarsi, ma solo alcuni considerati significativi ai fini del funzionamento.
Il programma utilizzato per la scrittura dei time diagram consente di disegnare frecce solo a partire dai fronti, non permette infatti di farle partire nel punto in cui il clock effettivamente campiona il segnale: pertanto il punto di partenza di ogni freccia dovrebbe in realtà essere al fronte di salita del clock successivo rispetto a quella in cui è disegnato.

Gestione CNT_26



Si può qui osservare il momento in cui viene asserito il reset (RST), attivo alto, che abilita tutti i segnali di clear interni: il segnale SCLEAR_ALL controlla CNT_12 e RX_REG. Per quest'ultimo viene asserito anche il clock enable, presente nel blocchetto generato dal megawizard di quartus.
Subito dopo il reset il sistema avvia CNT_26 e inizia a campionare: quando il contenuto del contatore è 25, il terminal count sale e abilita il registro di ingresso, oltre che resettare il contatore stesso.

Inizio ricezione e gestione di CNT_12

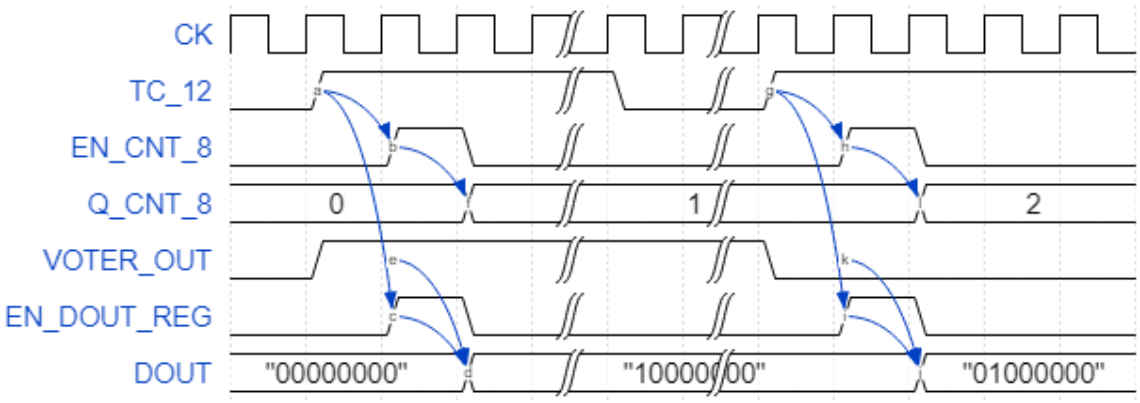


L’evoluzione del contenuto del registro segue a quella del time diagram sopra descritto. La dicitura “7”, ”8”, ”1” come contenuto del registro indica quanti campioni del bit ricevuto sono salvati all’interno del registro.

Si può osservare come in modo combinatorio si attivi il segnale SBC (START_BIT_CHECK): quando TC_26 viene campionato a ‘1’ mentre SBC è anch’esso a ‘1’, il contatore modulo 12 viene abilitato per la prima volta.

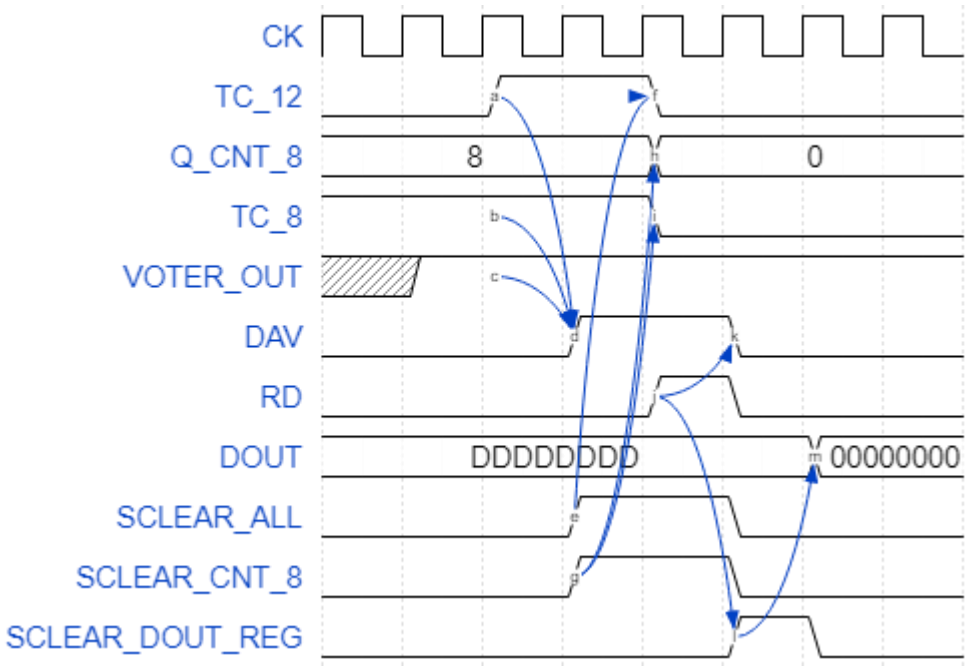
Infine, se TC_26 sale quando anche TC_12 è alto, CNT_12 verrà caricato al valore costante di 5: in questo modo il contatore può contare gli 8 campioni di ciascun bit ricevuto.

Salvataggio del bit ricevuto nel registro di uscita



Quando CNT_12 finisce di contare vengono abilitati contemporaneamente sia CNT_8 che il registro di uscita.
A seconda del valore del voter viene aggiunto un bit diverso come MSB del registro di uscita: dal time diagram si possono vedere entrambi i casi, prima quando l’uscita del voter vale “1”, poi quando vale “0”.

Fine ricezione e arrivo del segnale RD

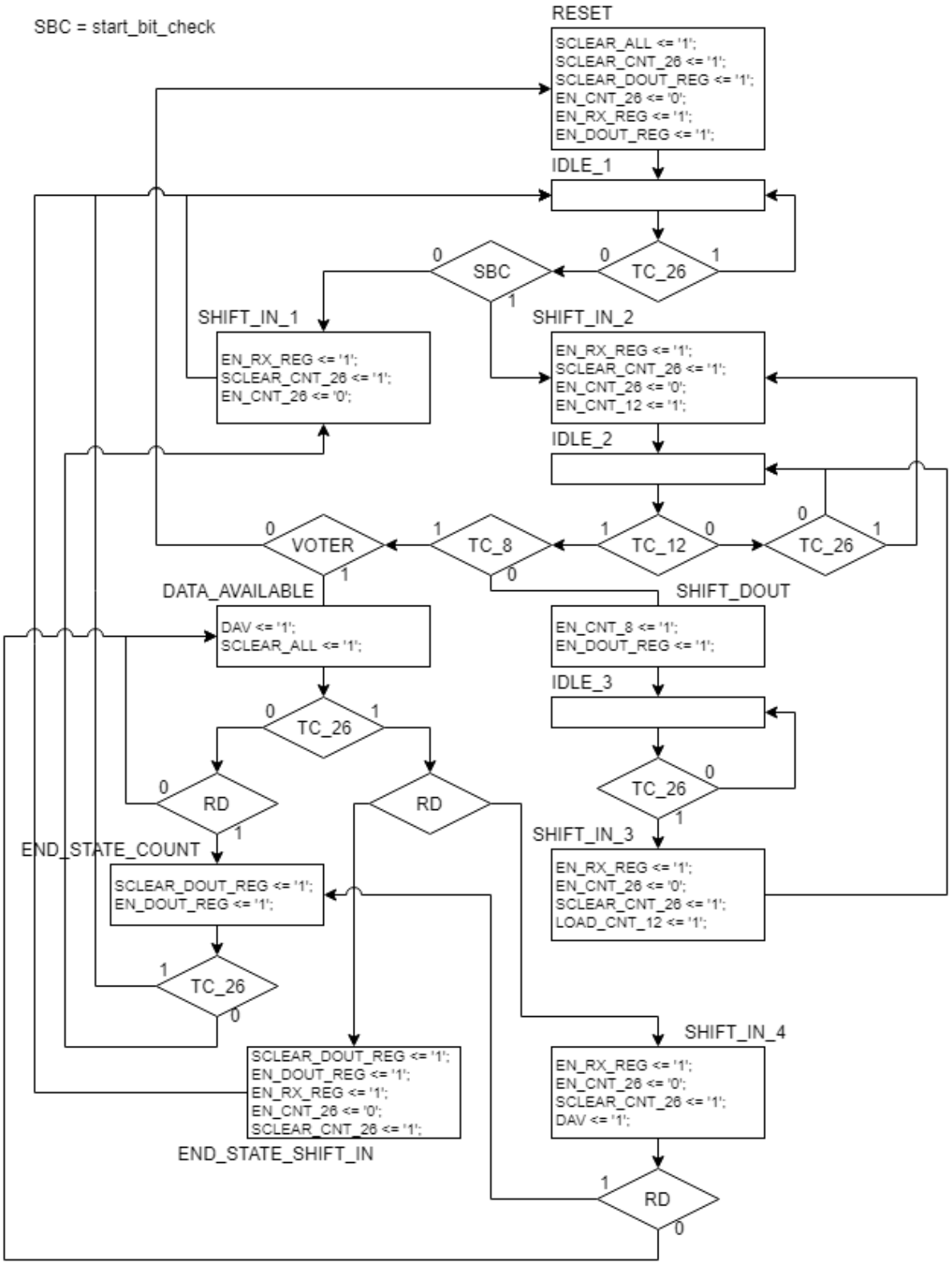


Dopo che l'ultimo bit è stato caricato nel registro di uscita, il sistema continua a campionare per controllare che anche il bit di stop sia stato ricevuto correttamente: se quando TC_12 sale e contemporaneamente anche il terminal count di CNT_8 è a "1" e l'uscita del voter è a "1", il dato è stato ricevuto correttamente. Questo causa la salita del segnale di DAV. Contestualmente vengono asseriti anche i segnali di clear del datapath. In questo caso SCLEAR_ALL resetta solo CNT_12, in quanto non viene asserito il clock enable di RX_REG: questa imprecisione è dovuta a una modifica successiva. (Ci si è infatti resi conto che resettando in questo punto il registro di ingresso sarebbe stato necessario aspettare un intero bit di idle dopo aver asserito RD affinché una nuova ricezione potesse avere successo. Tuttavia, questo avrebbe limitato le prestazioni impedendo di riconoscere sequenze di tipo stop-start bit. Vista la possibilità di non modificare l'interfaccia del datapath si è solo cambiato un segnale nella control unit (EN_RX_REG) per risolvere il problema.)

Si è quindi considerato il caso limite in cui appena dopo aver campionato il DAV esternamente, un altro dispositivo asserisce subito il segnale RD. Come da specifica il segnale DAV scende immediatamente dopo aver campionato RD, inoltre il registro di uscita viene resettato.

UART RX: Control Unit

Si è quindi derivata un'ASM chart del controllo a partire dal timing per strutturare successivamente la control unit in vhdl.



I segnali di default sono tutti a “0” tranne l’enable di CNT_26 che deve continuare a contare tranne quando viene resettato.

Gli stati sono stati inseriti in funzione del posizionamento temporale dei segnali che contengono.

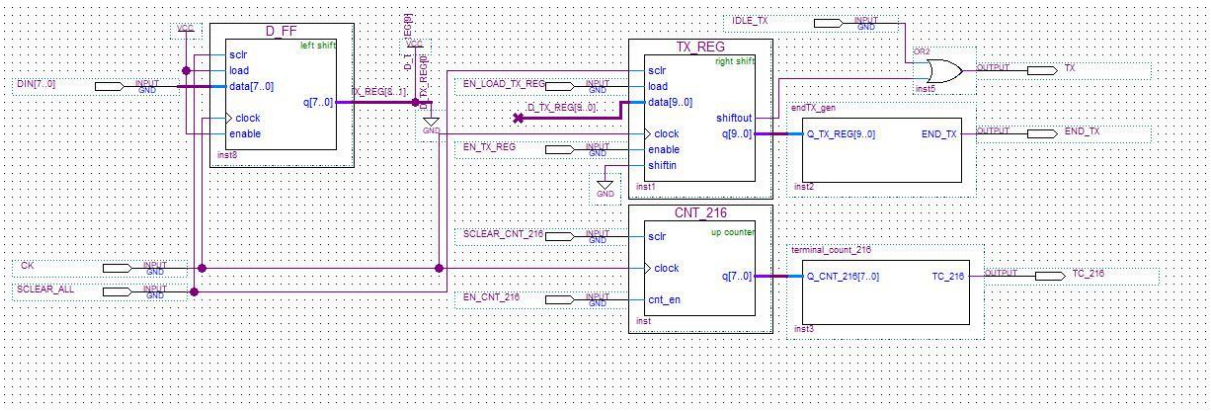
L’unica parte non spiegata nei time diagram riguarda la lettura del segnale RD: avendo deciso di continuare a campionare la linea in ingresso anche quando il dato viene fornito in uscita si sono aggiunti tre stati per la gestione di questa parte. L’idea è che, dovendo sempre contare, RD può arrivare o mentre si sta contando o quando si deve resettare il contatore e campionare con il registro di ingresso. Il primo dei 3 stati viene raggiunto quando RD sale a “1” e non si deve resettare, si incrementa quindi il contatore e si torna in IDLE_1 ad aspettare il prossimo campione. Il secondo copre il caso in cui RD è a “1” quando il contatore deve essere resettato e un bit shiftato dentro al registro di ingresso, per poi tornare in IDLE_1. Il sistema evolve nell’ultimo stato quando RD vale “0” e il contatore ha raggiunto la fine: il nuovo campione viene caricato nel registro e un nuovo controllo su RD viene svolto: se positivo il sistema conta ancora una volta e torna in IDLE_1, altrimenti torna ad aspettare RD nello stato di DATA_AVAILABLE.

UART TX

La progettazione del blocchetto di trasmissione UART è avvenuta in modo analogo al blocco di ricezione. Valgono pertanto le stesse considerazioni fatte sul layout del timing e sulle scelte iniziali di progetto.

UART TX: Datapath

Questo è lo schematico del datapath che è stato ricavato:



Gli elementi che compongono il datapath saranno ora spiegati, funzionalmente, in ordine di uso temporale, cioè dal primo che viene usato durante l'operazione del dispositivo all'ultimo.

D_FF

Un layer di flipflop è stato inserito tra l'ingresso del dispositivo e il registro TX_REG: questo si è reso necessario poiché il dato da trasmettere viene fornito assieme al segnale WR. Essendo il sistema controllato da una macchina stati, non era possibile asserire contemporaneamente a WR il segnale di load del registro senza introdurre un rischioso stato di mealy. Si è quindi scelto di ritardare il segnale entrante usando dei flipflop.

TX_REG

Il registro principale che svolge le operazioni all'interno del dispositivo è un right-shift-register la cui uscita seriale è collegata ad una porta OR e il cui ingresso parallelo ha 10 bit: il più significativo fisso a '1', quello meno significativo fisso a '0', saranno stop e start bit del segnale trasmesso, e i restanti 8 collegati all'uscita dei flipflop.

L'ingresso seriale è collegato fisso a 0: questo è necessario per capire quando la trasmissione deve essere conclusa e la linea riportata in idle.

CNT_216

Il contatore permette di dividere il clock(25Mhz) per 217, ottenendo così il baud rate desiderato. Come per i contatori dell'RX anche qui si è usato il timing per decidere quali ingressi della porta AND andavano negati per un corretto funzionamento del sistema.

La gestione del contatore è affidata quindi alla control unit: ogni qual volta il terminal count sale a '1', il contatore viene resettato e il registro shifta via un bit.

endTX_gen

Blocchetto di logica che controlla quando il contenuto del registro è “0000000001”: quando questo accade la trasmissione deve terminare e la linea deve essere riportata all’uno logico, riasserendo il segnale TXRDY.

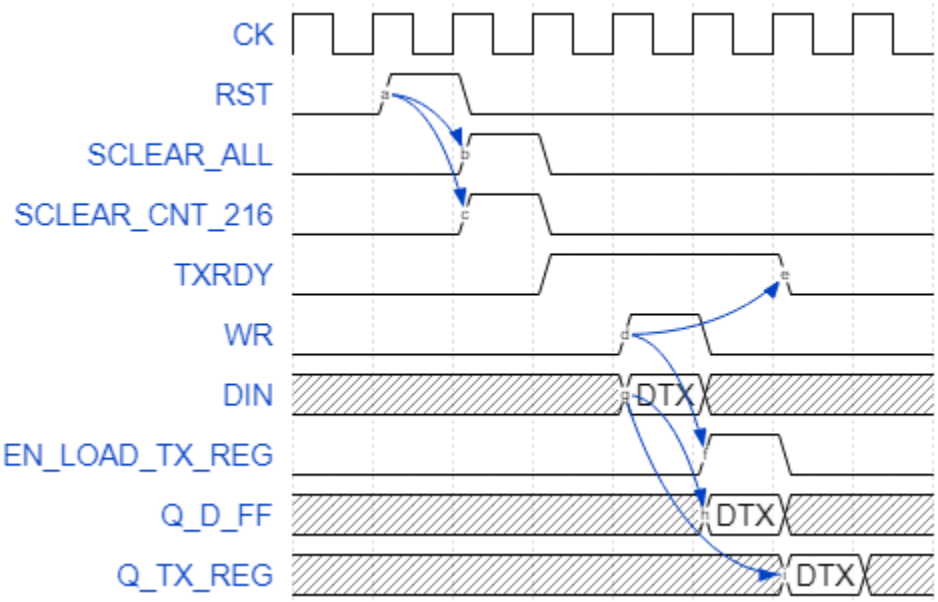
Porta OR

Una singola porta OR gestisce l’uscita di trasmissione: un segnale controllato dalla macchina a stati è in grado di rendere trasparente o meno la porta, il cui altro ingresso è collegato all’uscita del registro.

UART TX: Time Diagram:

Il diagramma temporale di questo blocco è stato anch’esso suddiviso in più parti per semplificarne la comprensione. Ciascuna verrà commentata singolarmente.

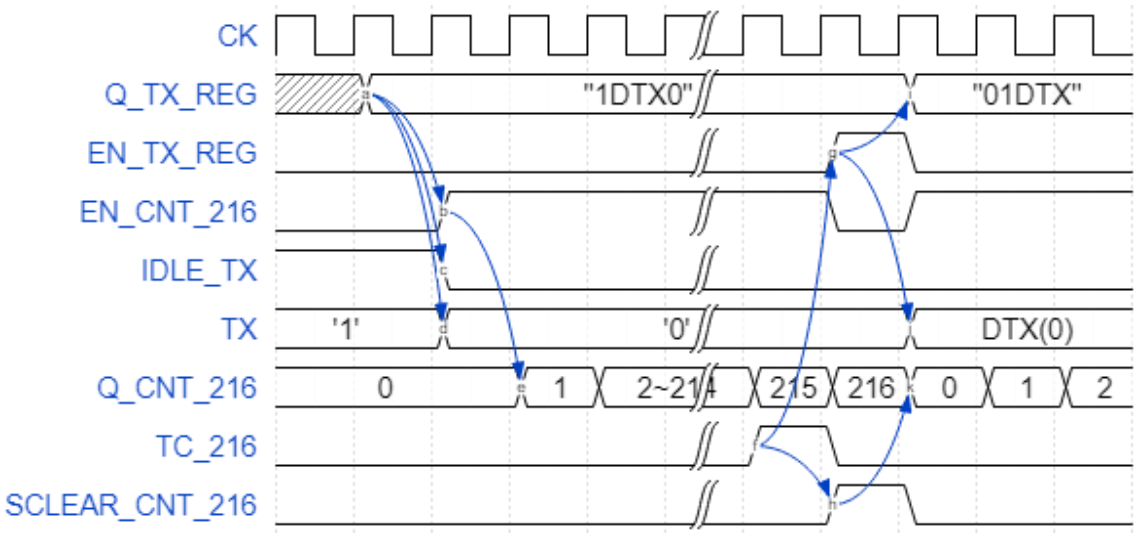
Salvataggio del dato da trasmettere



Si può qui osservare il momento in cui viene asserito il reset (RST), attivo alto, che abilita i segnali di clear interni: il segnale SCLEAR_ALL resetta il registro e i flipflop. Anche in questo caso vengono asseriti i segnali di clock enable per permettere il corretto reset.

Subito dopo il reset il segnale di TXRDY viene portato attivo alto: quando viene ricevuto il segnale WR, il dato da trasmettere viene salvato prima nei flipflop e subito dopo nel registro.

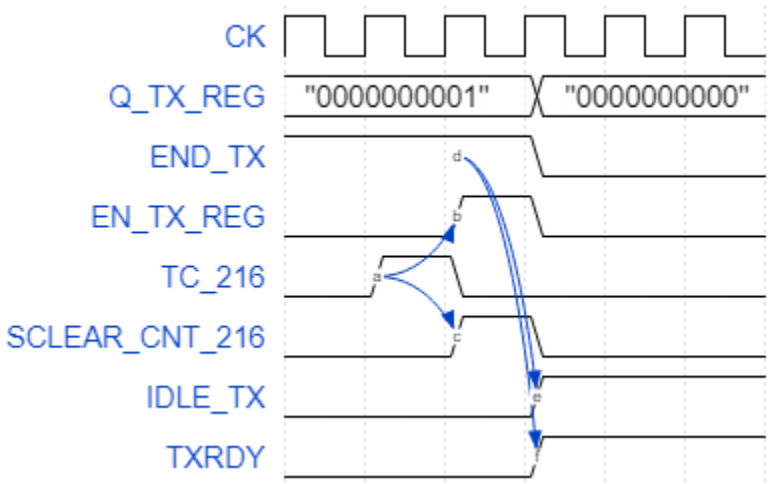
Inizio trasmissione e shift del registro



Dopo aver caricato il dato nel registro, viene abilitato il contatore e la linea viene subito portata a “0” disabilitando il segnale IDLE_TX.

Quando il contatore termina la conta viene resettato e contemporaneamente viene shiftato di un bit il contenuto del registro. (La scrittura “1DTX0” indica che il contenuto del registro è composto da un bit noto a “1”, 8 bit ignoti ma che contengono il dato da inviare e un bit noto a “0”.

Fine trasmissione

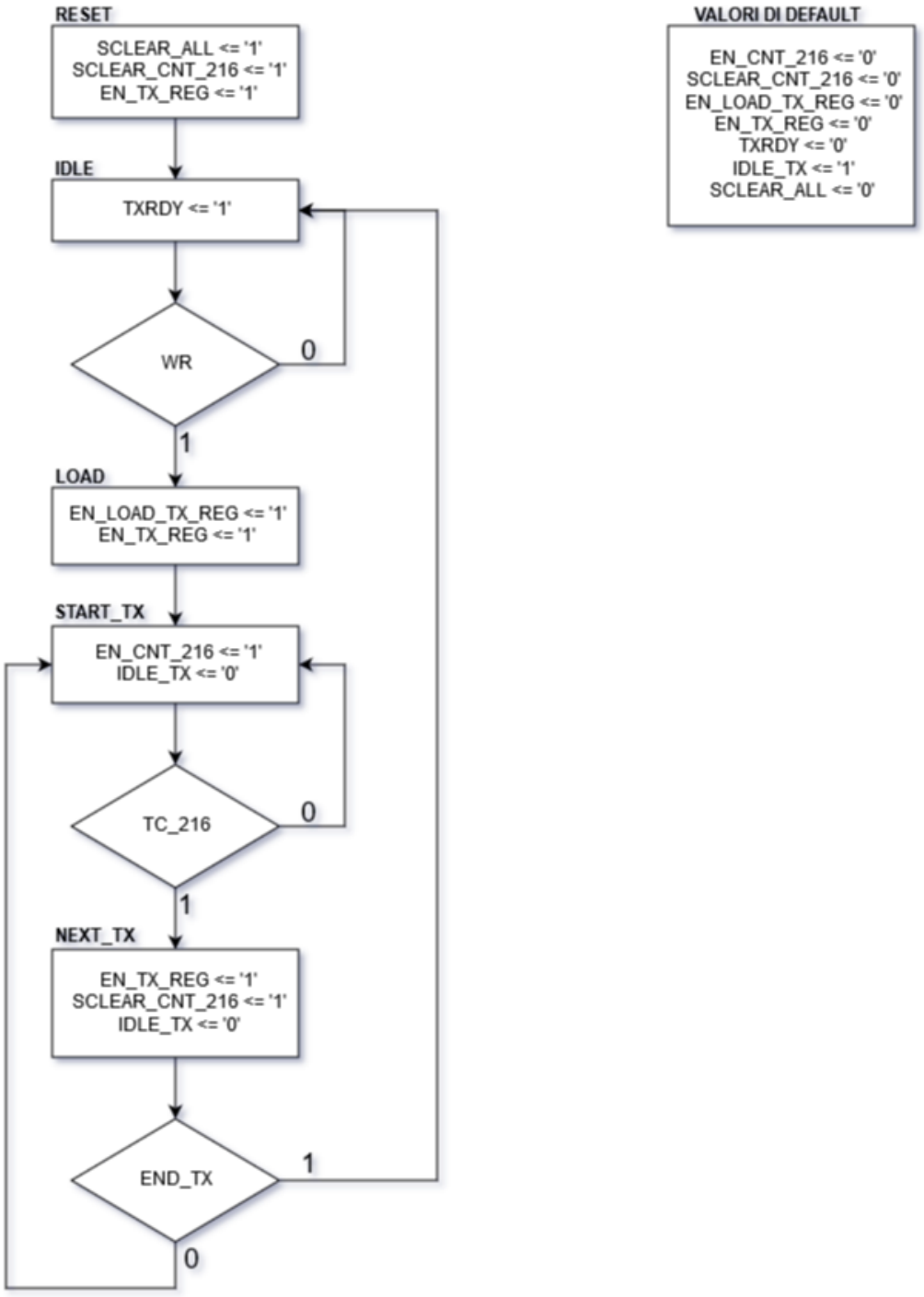


Quando il contenuto del registro diventa “0000000001”, il segnale END_TX va all’uno logico: durante questo periodo di tempo viene trasmesso lo stop bit. Quando il contatore finisce di contare il registro viene shiftato, il contatore resettato e la linea ritorna a ‘1’, insieme a TXRDY. C’è un ritardo di un colpo di clock tra TC_216 e il cambio di IDLE_TX: questo perchè

nel colpo di clock in cui sale il clear del contatore il contatore arriva a 216. Se si cambiasse prima il valore dell'uscita lo stop bit sarebbe più breve di 40 ns. Per cui la decisione su END_TX viene prese con un colpo di clock di ritardo.

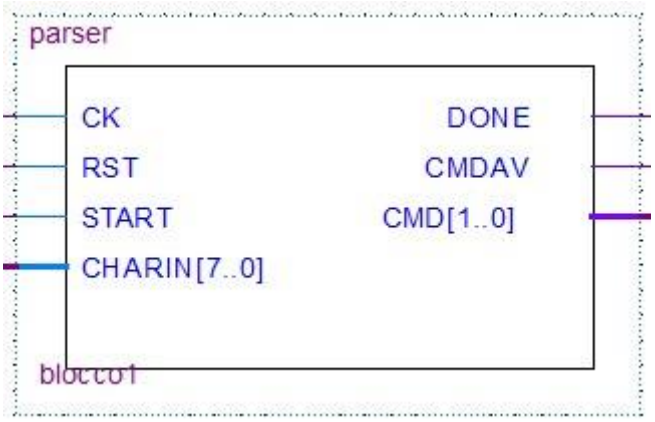
UART TX: Control Unit

Si è quindi derivata un'ASM chart del controllo a partire dal timing per strutturare successivamente la control unit in vhdl.



Blocco 1: Parser

Il primo blocco del circuito, non atto a comunicare con l'esterno, è il parser. Come da definizione il parser deve capire se e quali comandi vengono ricevuti. Nel nostro circuito questo significa identificare e codificare per il resto del circuito le sequenze di caratteri "L1", "L2", "L3", "L4", codificati rispettivamente in "00", "01", "10", "11". L'interfaccia del dispositivo è la seguente:



CK: il clock del dispositivo, come da specifica deve avere una frequenza di 25MHz

RST: segnale di reset che porta ad uno stato noto la macchina stati interna e resetta i registri. Il segnale è considerato attivo all'uno logico.

START: segnale che comunica al dispositivo che un nuovo carattere sarà disponibile in ingresso. Si veda il timing per comprendere quando il dato in ingresso è richiesto valido affinché venga campionato correttamente.

CHARIN: ingresso parallelo di 8 bit che riceve i caratteri che compongono la sequenza di comando, uno alla volta.

DONE: segnale che comunica all'esterno che il carattere ricevuto al precedente START è stato analizzato e che è quindi possibile asserire un nuovo START e fornire un nuovo carattere.

CMDAV: (command available), se è stata ricevuta una sequenza di comando corretta, questo segnale viene portato a '1'. Resta a '1' fino a che START non viene asserito nuovamente.

CMD: (command), segnale che contiene la codifica del comando. Questo segnale è da considerarsi valido solo quando CMDAV è attivo alto.

Blocco 1: Datapath

Durante la fase di derivazione del datapath si è cercato un approccio il più modulare possibile. Si è pertanto suddiviso in tre blocchi principali:

ASCII_L_match: una semplice porta AND a 8 ingressi, opportunamente negati, che controlla che il segnale ricevuto in ingresso corrisponda al carattere “L”.

ASCII_1_2_3_4_match: 5 porte AND e una porta OR che controllano che il segnale ricevuto in ingresso corrisponda al carattere “1”, “2”, “3”, “4”.

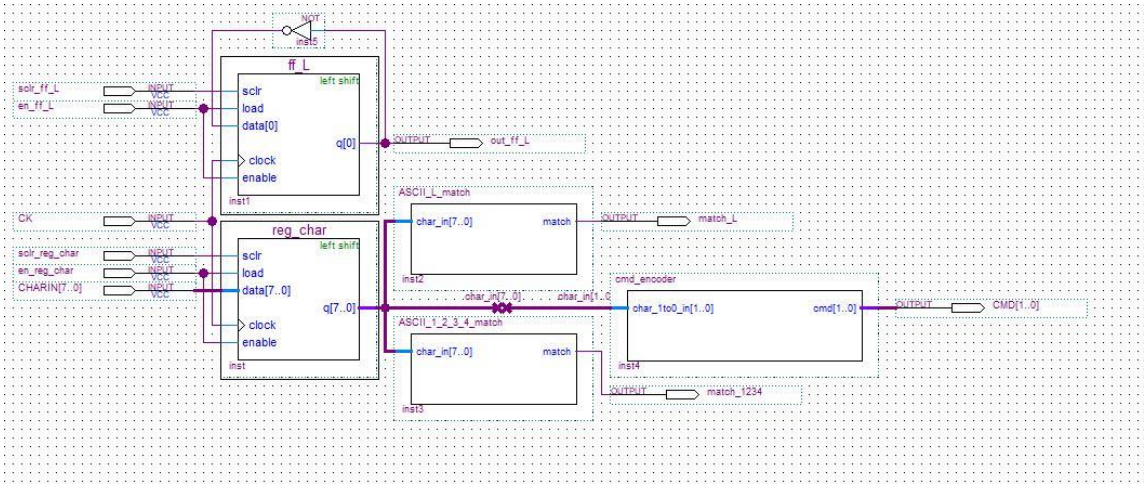
cmd_encoder: un encoder che riceve in ingresso i due bit meno significativi dell’ingresso e vi sottrae 1, tramite l’uso di due inverter e una porta XOR. In questo modo l’uscita di questo blocchetto è quella richiesta dalla specifica decisa in fase di progettazione del sistema.

Oltre a questi blocchi sono stati inseriti un registro di ingresso e un T-flipflop:

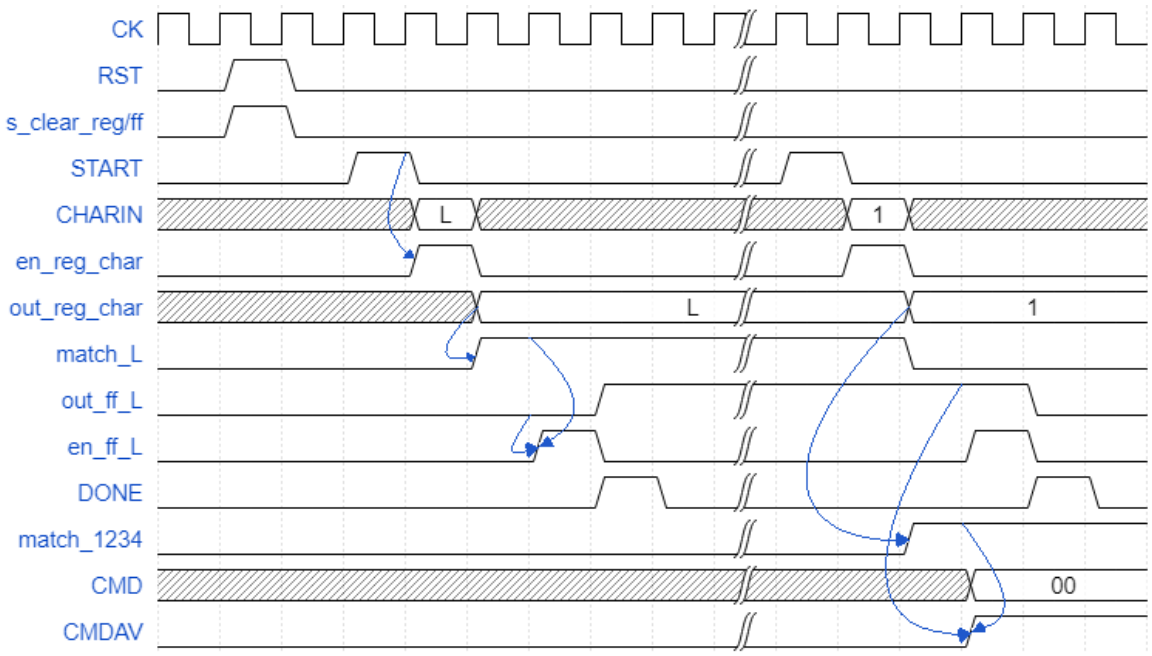
il primo salva il dato ricevuto in ingresso per permettere alla macchina a stati di gestire le operazioni interne e temporizzare il sistema, che risulterebbe altrimenti totalmente combinatorio.

il secondo salva un flag che permette al dispositivo di sapere se al ciclo precedente è stato ricevuto o meno un carattere “L”.

Questo è il datapath risultante:



Blocco 1: Time Diagram



Il timing di ingressi e uscite è stato deciso in fase di definizione dell'interfaccia del blocco. In particolare, il carattere in ingresso deve essere valido al colpo di clock successivo rispetto a quello in cui viene campionato lo START. Il comando in uscita viene asserito contemporaneamente al CMDAV e restano entrambi validi fino a un nuovo segnale di START. Il segnale di DONE viene asserito ogni volta che il controllo sul valore del carattere di ingresso viene terminato e il flip flop contiene il valore corretto.

La fase cruciale del timing riguarda il momento della decisione: esistono quattro possibili casi (di cui 2 riportati nel time diagram):

-match_L = "0" e out_ff_L = "0": il sistema non fa nulla, comunica all'esterno di aver finito con il segnale di DONE e torna ad aspettare di ricevere una "L".

-match_L = "1" e out_ ff_L = "0": in questo caso il sistema cambia il valore del flipflop, aspettandosi al prossimo carattere uno tra "1", "2", "3", "4".

-match_1_2_3_4 = "0" e out_ ff_L = "1": il secondo carattere ricevuto è scorretto. Il sistema cambia valore al flipflop e torna ad aspettare una "L".

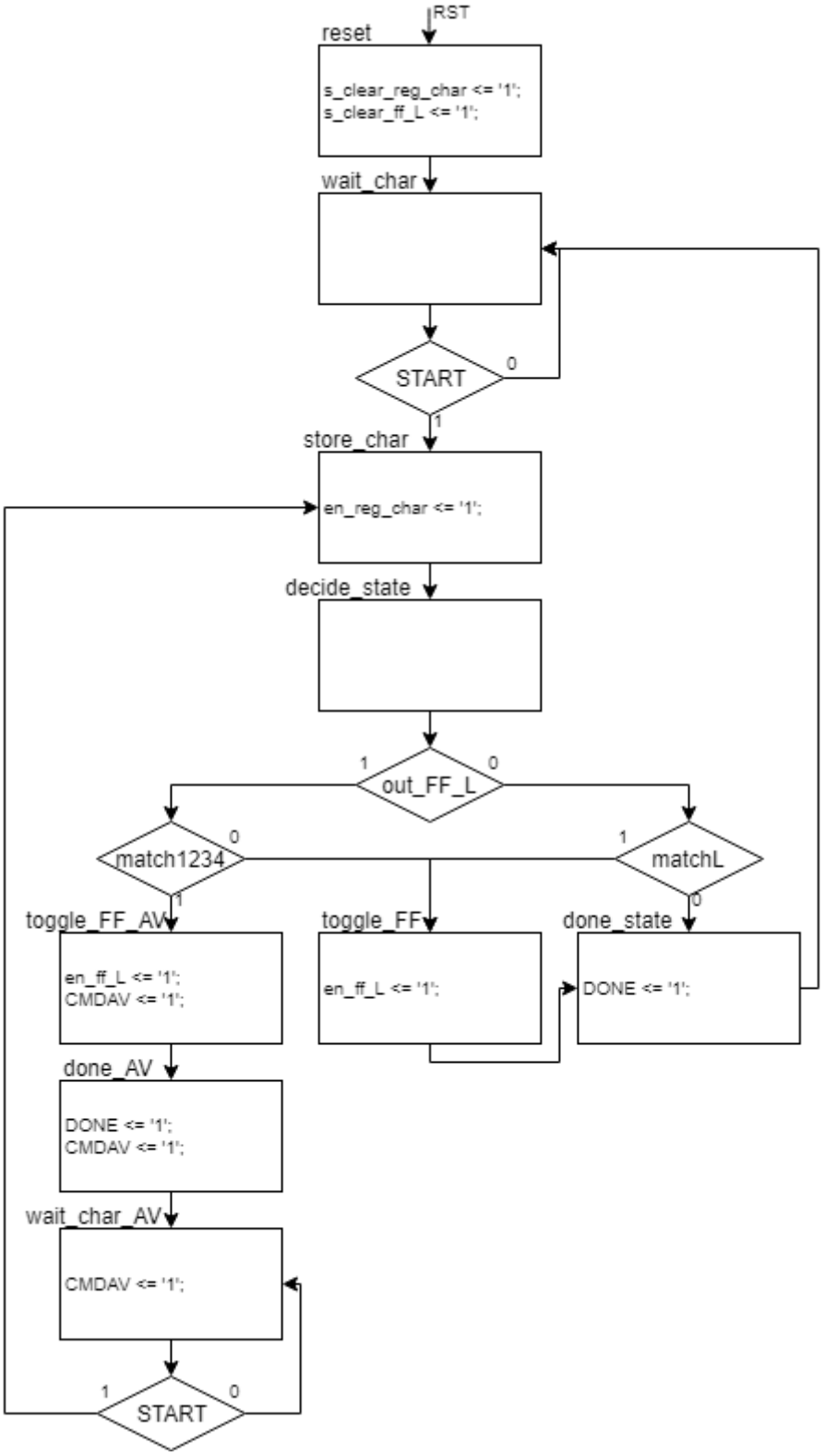
-match_1_2_3_4 = "1" e out_ ff_L = "1": un comando è stato ricevuto, viene quindi subito asserito il CMDAV. L'uscita è combinatoria a partire dal registro di ingresso e non deve quindi essere controllata. Al colpo di clock dopo il CMDAV viene asserito il DONE.

Esisterebbero teoricamente altri 4 casi, tuttavia si suppone che il dispositivo sia trasparente a match_L quando out_ ff_L è uguale a "1", e viceversa con match_1_2_3_4.

Si deduce da queste considerazioni che se il parser riceve la sequenza "LL", non sarà in grado di considerare la seconda "L" come un nuovo inizio di sequenza, ma se ne aspetterà subito dopo un'altra.

Blocco 1: Control Unit

Partendo dal timing si è derivata la seguente control unit, descritta attraverso una ASM chart del controllo:

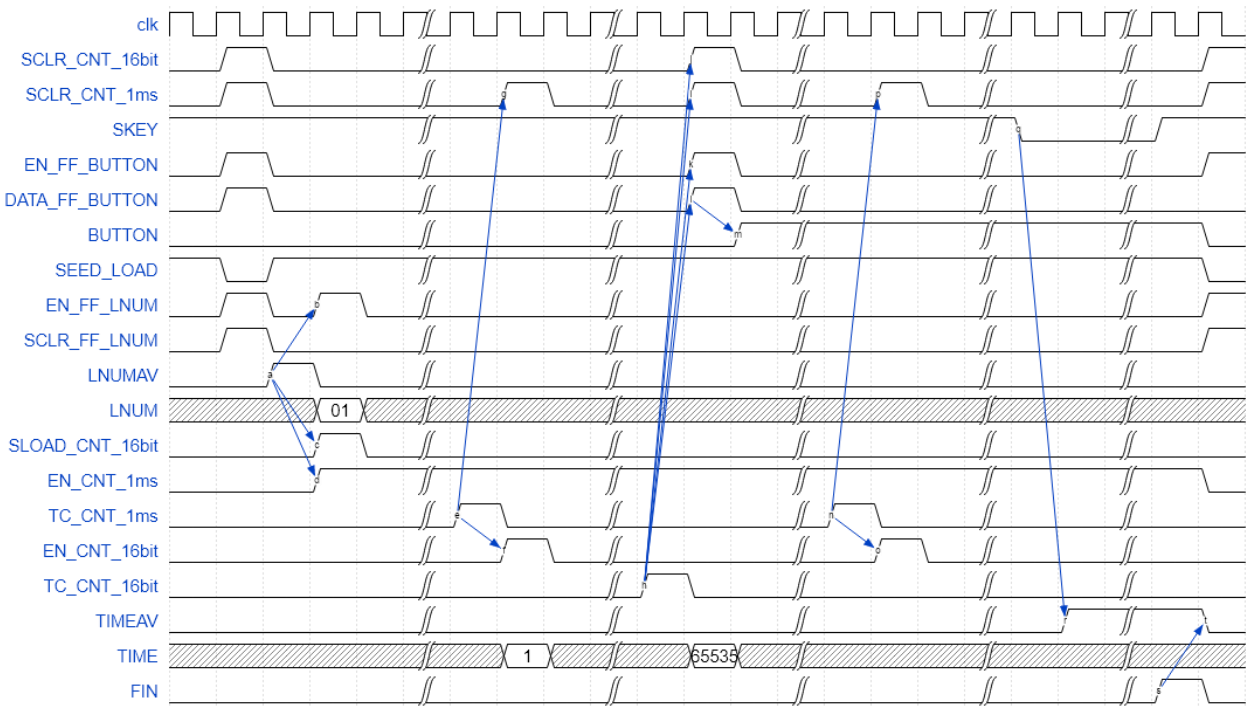


Dallo schema si può vedere come la “trasparenza” dei segnali di match rispetto al segnale out_FF sia stata implementata.

Vi è inoltre uno stato vuoto, introdotto per rendere l’ASM coerente con il time diagram precedentemente disegnato.

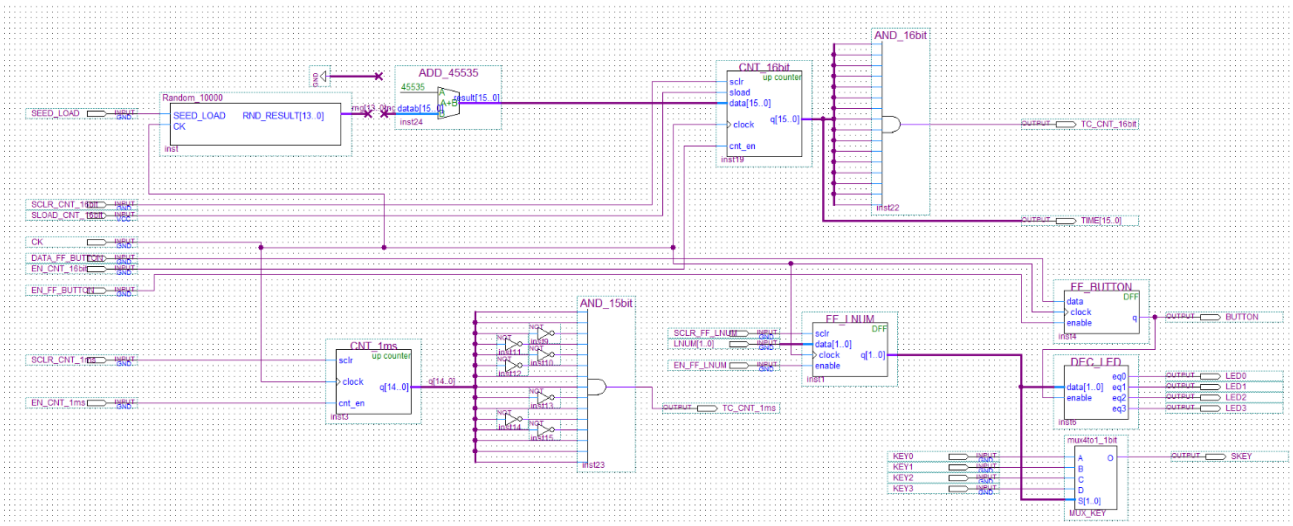
Gli stati contrassegnati da “AV” sono quelli in cui il sistema asserisce una codifica del comando in uscita, accompagnata dal CMDAV.

Blocco 2 : Generatore di ritardo casuale e cronometro



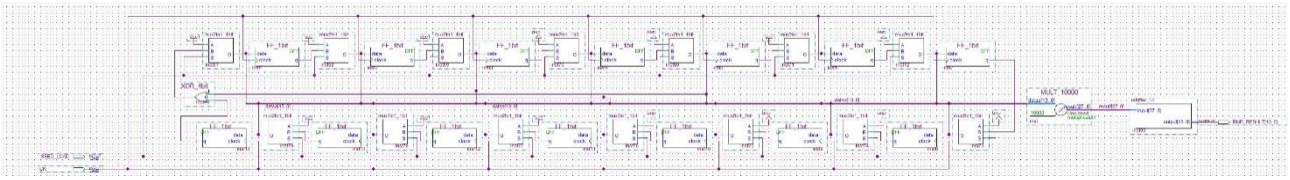
Il ruolo di questo blocco è quello di ricevere il numero del led/pulsante dal blocco precedente, aspettare un tempo casuale compreso tra 10 e 20 secondi, accendere il led precedentemente scelto, attendere fino ad un massimo di 65536 millisecondi che un utente prema il pulsante corrispondente al led e inviare il tempo trascorso al blocco successivo.

Blocco 2: Datapath



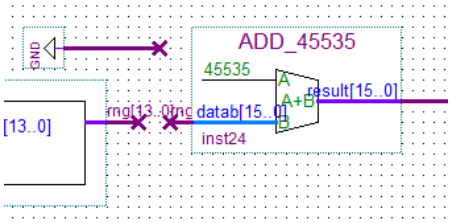
L’analizzatore di riflessi si compone di 7 blocchetti fondamentali.

Blocchetto 2.1: Random Generator



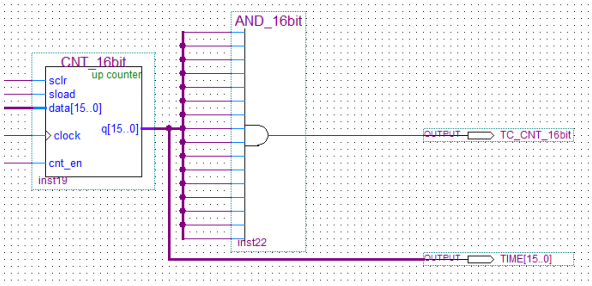
Per la generazione di un numero casuale tra 10000 e 20000, si è ritenuto opportuno sfruttare l'architettura di un'LFSR: uno shift register composto da 14 registri dove l'ingresso del primo registro è dato dall'or logico tra le uscite di 4 registri. Gli output di tutti e 14 i registri sono collegati ad un moltiplicatore e uno shifter che si occupano di eseguire l'operazione (output*10000/65536) per rendere l'uscita compresa tra 0 e 10000. Grazie ai multiplexer all'ingresso di ogni registro, è possibile impostare un seed (un valore iniziale da cui partire all'accensione).

Blocchetto 2.2: Adder



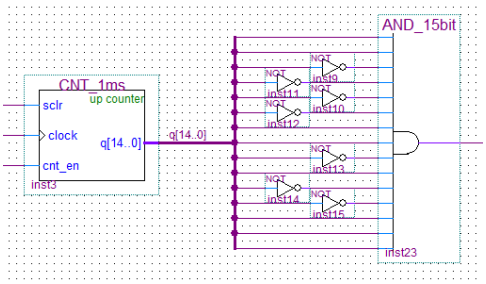
Grazie ad un adder è stato possibile sommare 45535 all'uscita del random generator in modo da avere, all'ingresso del blocchetto successivo, un numero casuale compreso tra 45535 e 55535. La sua utilità sarà compresa meglio in seguito.

Blocchetto 2.3: Counter 16 bit



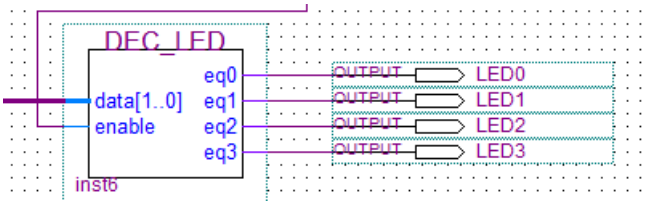
Per contare i millisecondi necessari all'accensione del led e quelli trascorsi alla pressione del pulsante, si è pensato di utilizzare un unico contatore a 16 bit. La doppia funzionalità di questo counter è possibile grazie al caricamento (tramite data[] e sload) di valori iniziali diversi. In particolare, verrà caricato un valore compreso tra 45535 e 55535 in modo da contare tra 10000 e 20000 millisecondi quando si dovrà attendere un tempo casuale per l'accensione del led e verrà resettato il valore iniziale a 0 quando dovrà contare fino a 65535 per aspettare che venga premuto il pulsante.

Blocchetto 2.4: Counter 15 bit



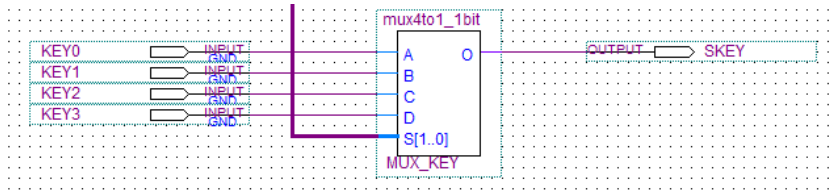
L'utilità di questo blocchetto è semplicemente quella di contare il trascorrere dei millisecondi (1 millisecondo ≈ 25000 colpi di clock da 25MHz) abilitando (previo passaggio attraverso CU) il counter 16 bit ogni millisecondo.

Blocchetto 2.5: Decoder



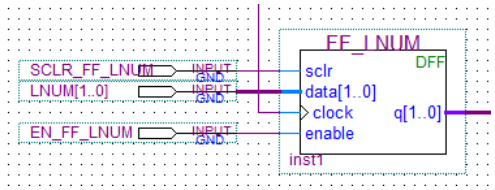
Questo blocchetto è un semplice decoder utile per accendere, dopo un tempo casuale, il led scelto dall'utente.

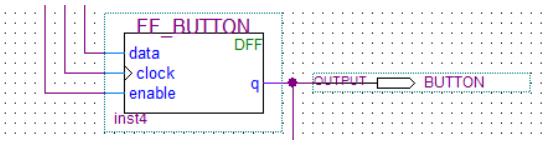
Blocchetto 2.6: Multiplexer



Grazie a questo blocchetto è possibile rendere la coltrol unit sensibile esclusivamente al pulsante corrispondente al led acceso.

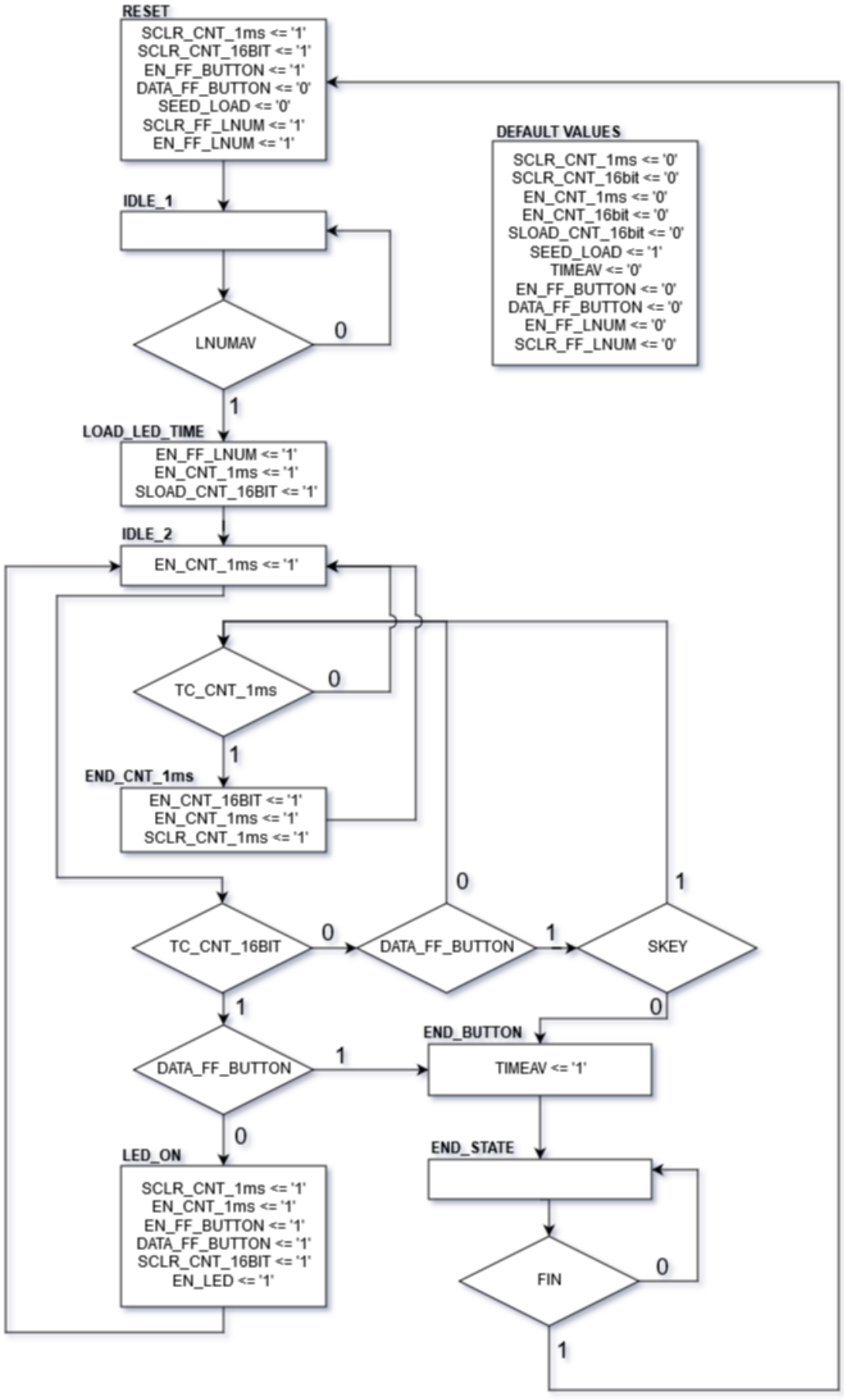
Blocchetto 2.7: Flip flop





Mediante due flip flop dotati di "enable", è possibile salvare le variabili di stato "BUTTON" e "LNUM" usate dalla control unit.

Blocco 2: Control Unit



Per generare i segnali di controllo del datapath precedentemente illustrato, si è deciso di progettare una control unit di cui è possibile vedere una rappresentazione qui sopra. Come si può notare, sono stati utilizzati 7 stati.

Il primo stato è il classico stato di reset inserito come stato iniziale in tutte le macchine a stati e usato per ripristinare la macchina in caso di errore; qui viene resettato il random generator (caricando il seed) e i vari flip flop e contatori.

Il secondo stato è uno stato di idle in cui la macchina rimane in attesa che, dal blocco precedente, arrivi il segnale “LNUMAV” indicante la presenza di un dato in “LNUM” (il numero del led che dovrà accendere il blocco 2).

Un colpo di clock dopo, nello stato “LOAD_LED_TIME”, il dato “LNUM” viene caricato nel flip flop “FF_LNUM” abilitando il suo enable, viene caricato il valore casuale generato nel contatore da 16 bit (“CNT_16BIT”) e viene abilitato il contatore da 15 bit (“CNT_1ms”) che, ricordiamo, serve per contare lo scorrere dei millisecondi.

Nel secondo stato di idle (“IDLE_2”) la control unit continua ad incrementare “CNT_1ms” in attesa che il suo terminal count (“TC_CNT_1ms”) o quello di “CNT_16BIT” (“TC_CNT_16BIT”) assumano un valore logico alto ad indicare che 1 ms è trascorso o che il tempo da attendere per accendere il led è finito. Di “DATA_FF_BUTTON” e “SKEY” si parlerà in seguito.

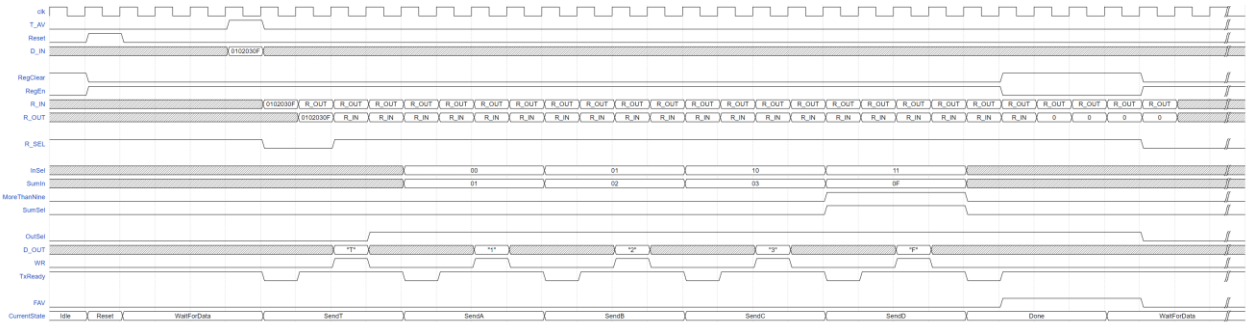
Nel caso in cui “TC_CNT_1ms” vada a “1”, si entra nello stato “END_CNT_1ms” in cui, semplicemente, viene abilitato “CNT_16BIT” per un clock (facendolo avanzare di 1) e resettato “CNT_1ms”. A questo punto si torna in “IDLE_2”.

Una volta che è trascorso il tempo casuale tra 10 e 20 secondi (“TC_CNT_16BIT”=”1”), si può entrare nello stato “LED_ON” in cui viene acceso il led corrispondente al valore “LNUM”, vengono resettati i contatori e, cosa più importante, viene salvato un “1” nel flip flop FF_BUTTON. L’utilità di quest’ultimo componente è quella memorizzare se il led è già stato acceso e, quindi, si è nella fase in cui bisogna attendere che l’utente prema il pulsante. Questa soluzione ha permesso una riduzione notevole dell’ASM permettendo il riutilizzo degli stati “IDLE_2” e “END_CNT_1ms” inserendo, semplicemente, i branch “DATA_FF_BUTTON” (due percorsi che permettono l’uscita dai due stati anche appena l’utente preme il pulsante).

Non appena “SKEY” va a valore logico basso (pressione del bottone giusto) o è trascorso il tempo limite di 65536 millisecondi, infatti, l’uscita del blocco ”TIMEAV” va ad “1” per un colpo di clock per indicare al blocco successivo che è disponibile il dato “TIME”.

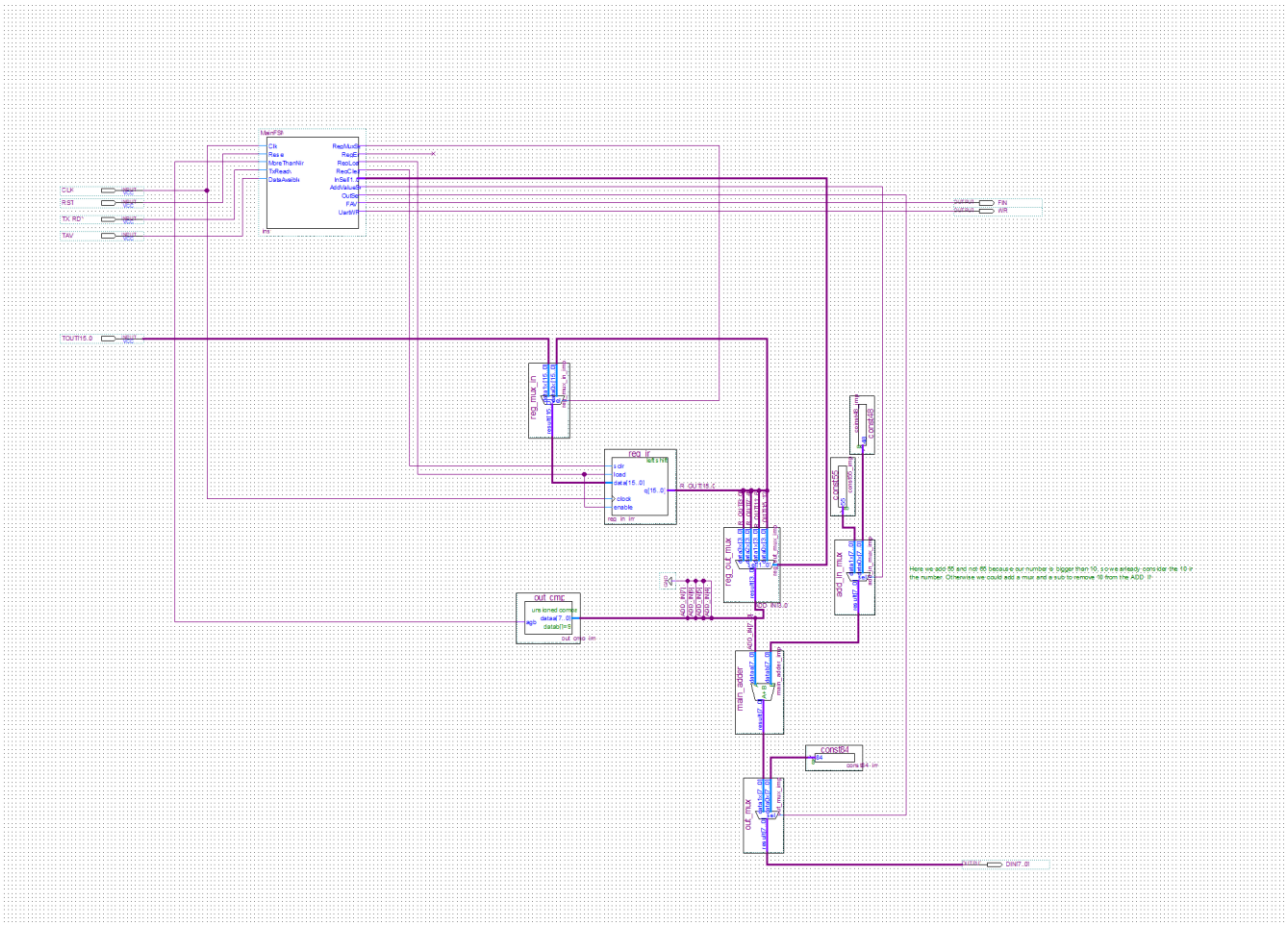
L’ultimo stato (“END_STATE”) è un altro stato di idle in cui la macchina rimane in attesa che gli “FIN” assuma valore “1” ad indicare che il blocco successivo è pronto a ricevere un nuovo dato e il tutto può ripartire.

Blocco 3: ASCII Converter



Il ruolo di questo blocco è quello di convertire il dato “TIME” in esadecimale e, quindi, in ASCII, prima di darlo in uscita alla Uart TX.

Blocco 3: Datapath

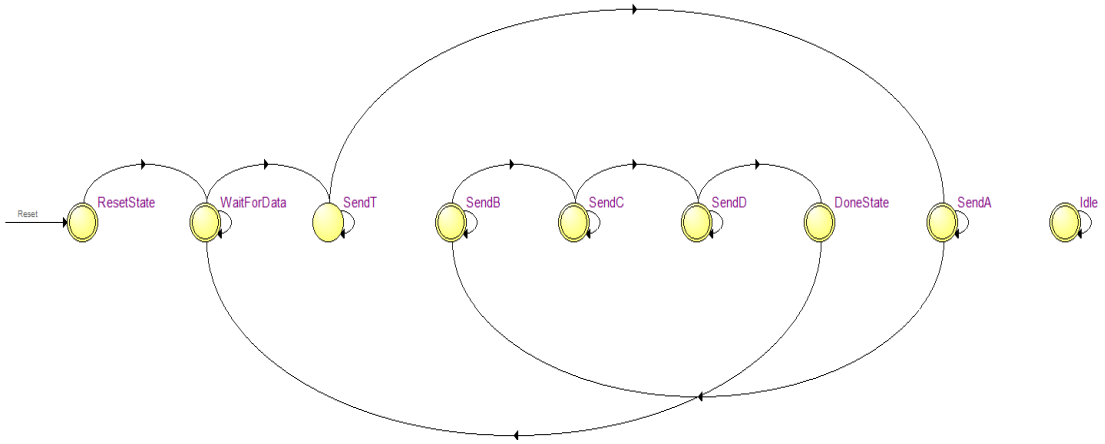


I componenti principali sono:

- 1 registro che mantiene i dati proveniente dal modulo precedente
- 1 sommatore che permette di selezionare quale valore aggiungere al dato in ingresso per ottenere o i numeri in ASCII o le lettere
- 1 mux che permetti di selezionare 4 dei 16 bit del dato in ingresso a seconda di quale valore si sta mandando nella UART
- 1 comparatore che determina se il dato corrente è maggiore o uguale a 9
- 1 macchina a stati che gestisce il datapath e manda il segnale di WR per il modulo UART

Blocco 3 : Control Unit

La FSM di questo blocco è formata dai seguenti stati: Reset, WaitForData, SendT, SendA, SendB, SendC, SendD, DoneState e Idle e presenta il seguente pallogramma:



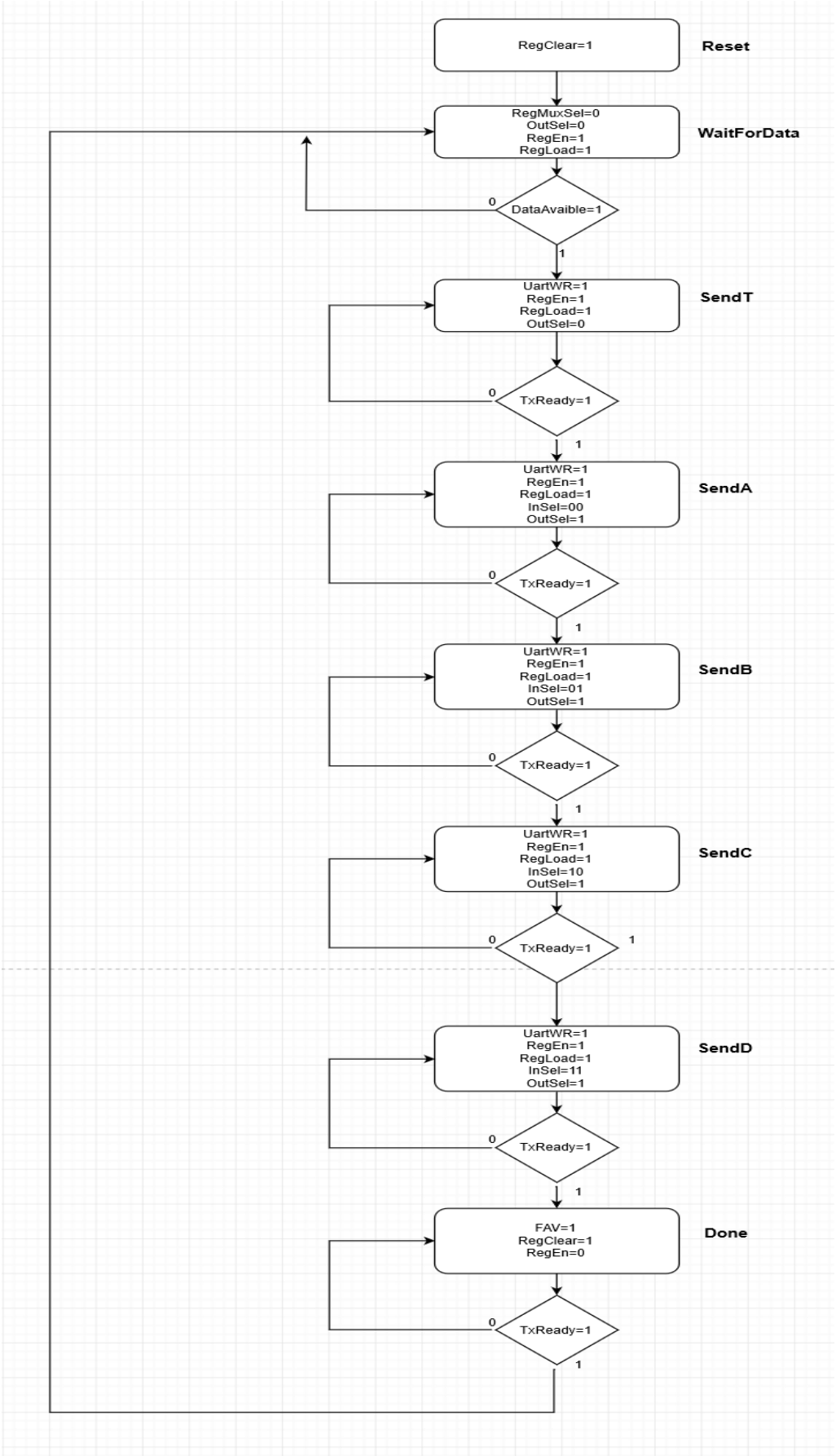
La macchina parte dallo stato “ResetState” e, a quel punto, entra in “WaitForData”.

Quando TAV va a valore logico alto, il dato in ingresso viene salvato nel registro reg_in e la macchina a stati entra nello stato “SendT” nel quale manda in uscita la lettera T codificata in ASCII.

Nei seguenti stati “SendA”, “SendB”, “SendC” e “SendD” si compone progressivamente il numero esadecimale in ASCII che verrà dato in uscita alla UART TX. Prima di ognuno dei 5 stati, è necessario attendere che TXReady sia a “1” per indicarci che il modulo TX è pronto alla trasmissione. Per informarlo della presenza di un dato da trasmettere, dopo ognuno degli stati viene portato WR a valore “1”. La conversione in ASCII avviene sommando al numero in esadecimale la costante “48” se è al più 9 o la costante “55” se è maggiore di 9.

Dopo aver inviato tutti e 5 i caratteri, viene portato a valore alto il segnale “FIN” che informa tutti gli altri blocchi dell’avvenuto invio e, quindi, del fatto che il blocco è pronto per un nuovo “TIME”.

Si riporta in seguito l’ASM della macchina a stati:



Conclusioni

Il progetto, così come è stato detto nell'introduzione, è stato fatto per essere modulare, come se ogni blocco fosse una "scatola nera" a sé. Per verificare l'atomicità di ognuno di essi, si sono effettuati test singoli sfruttando il software "Modelsim" e usando opportuni test case il più possibile realistici e vari in modo da verificare gli scenari più comuni e limitare l'eventualità di un errore. Per completezza i file usati per le simulazioni sono presenti nella cartella del progetto. Ogni blocco presenta uno stato di reset accessibile premendo un apposito pulsante sulla scheda proprio per evitare un blocco totale della macchina da cui non si riuscisse ad uscire. La simulazione, oltre che virtuale, è stata fatta su un dispositivo fisico, la scheda Altera DE2 dotata di FPGA Cyclone II EP2C35F672C6, collegata ad un computer sul quale si inviavano e ricevevano i dati sfruttando il programma "Putty".