Lo scopo delle memorie in questo design è duplice: separare due regimi con velocità differenti e memorizzare i dati e i programmi per lo Z80X.  
Per sfruttare al meglio le potenzialità dell’FPGA, ho usato gli IP messi a disposizione.

Block RAMs, RAM e ROM

Figura X – Schema BRAM generica

Il design presenta due BRAM identiche ma con due scopi differenti. La prima rappresenta una ROM, per cui da parte dello Z80X non è scrivibile, mentre l’altra è una RAM completamente usabile dal microprocessore. Entrambe permettono l’uso completo da parte di due port, funzionalità real dual port. Il primo port, che ha priorità maggiore, è sempre collegato al controllore così da poter caricare il programma e leggere il valore delle memorie indipendentemente dalle operazioni che sta compiendo lo Z80X.

I port delle BRAM presentano la stessa struttura. Per prima cosa c’è il segnale di temporizzazione che in questo caso è il medesimo. Dopodiché vi sono i pin di controllo: EN, WE e REGCE. Con il primo si abilita la BRAM che altrimenti non risponde ai comandi, con il secondo invece si abilita la scrittura. Il terzo pin abilita i registri d’uscita che servono a mantenere l’uscita della BRAM constante quando non è utilizzata. Permettono anche di non avere un valore fluttuante sul bus dati durante la lettura.

Le BRAM presentano un tempo di accesso pari a 2 cicli di clock e un tempo di latenza di un ciclo.[Buso p.146] Per permettere la giusta temporizzazione, il controllore fa uso di due entity dedicate, BRAM\_READER e BRAM\_WRITER, che implementano i cicli di lettura e scrittura. Queste entity permettono di avere un segnale di DONE nel momento in cui il dato è sicuramente stato elaborato correttamente a costo di un ciclo di clock in più.

Dopo questi segnali ci sono il bus indirizzi ADDR assieme ai due bus dati DIN e DOUT. La dimensione di questi può essere molto variabile in base al numero di parole contenute e della loro lunghezza. Inoltre si può modificare la lunghezza delle parole in ingresso ed uscita permettendo l’uso della memoria come una cache. Per permette un uso più semplice in questa configurazione, WE può diventare un vettore con cui selezionare il byte della parola da scrivere.

Distribuited RAM, CHAR\_RAM

L’unica RAM distribuita è la CHAR\_RAM che serve ad immagazzinare i caratteri che il driver 7 segmenti mostrerà sul display. Ho scelto di implementarla come RAM distribuita poiché è di piccole dimensioni, 256 bit, e se avessi usato una BRAM avrei dovuto sprecare interamente una pagina di 9kb poiché è il taglio minimo.

CHAR\_RAM è di tipo simple dual port poiché presenta due port indipendenti ma da uno si può sia leggere che scrivere mentre dall’altro è permessa la sola lettura. Il primo port è connesso all’interfaccia con lo Z80 mentre il secondo al driver.

FIFOs

Schema X^2 – Schema FIFO generica

Le FIFOs messe a disposizione come IP presentano due port di controllo e sono implementabili sia con BRAM che con RAM distribuite. In comune sono presenti oltre al segnale di temporizzazione anche un segnale di reset dell’intera FIFO, RST, e un segnale che informa della quantità di dati presenti nella FIFO, DATACNT.  
Il port di scrittura presenta i due segnali per il dato in ingresso, DIN, e l’abilitazione alla scrittura, WR\_EN. Oltre a questi ci sono dei segnali che danno informazioni sullo stato della FIFO:  
FULL, informa che la FIFO è piena e non accetterà più dati;  
OVERFLOW, informa che nell’ultima operazione di scrittura la FIFO era piena ed il dato appena scritto è stato ignorato;  
WR\_ACK, è la conferma della scrittura del dato sulla FIFO.  
Il port di lettura presenta il bus per il dato in uscita, DOUT, che può avere dimensione diversa da DIN, e il segnale per leggere il dato successivo, RD\_EN. Oltre a questi vi sono dei segnali di controllo per gestire il flusso di dati:  
EMPTY, comunica che la FIFO non ha più dati disponibili;  
VALID, comunica che il dato su DOUT può essere letto;  
UNDERFLOW, comunica che nella precedente lettura la FIFO era vuota e che in uscita non sta presentando un dato valido.  
La presenza del segnale VALID assieme a EMPTY è particolarmente utile nel caso di FIFO non First-Word Fall-Through (FWFT). Nelle FIFO FWFT, appena un dato è stato scritto è già disponibile sull’uscita e non richiede di scorrere tutta la FIFO prima di averlo disponibile. Nel mio caso ho usato solo FIFO FWFT per la semplicità di utilizzo.

Ho usato le FIFOs principalmente per due scopi: separare i regimi con velocità differenti e per immagazzinare i dati mantenendo il loro ordine.

Al primo scopo ci sono le tre FIFOs per la comunicazione UART: FIFO\_RX, FIFO\_TX e FIFO\_TX\_SLAVE.  
FIFO\_RX serve ad immagazzinare un’intera riga di comando proveniente dall’interfaccia UART mantenendo l’ordine con cui è stata inviata. In questo modo se il controllore sta eseguendo una scrittura sull’UART e nel frattempo arriva un altro comando, la FIFO lo immagazzina. Così il controllore può leggere i caratteri del nuovo comando nel momento in cui l’invio è finito.  
FIFO\_TX\_SLAVE viene scritta man mano che il controllore legge i caratteri da FIFO\_RX. Questa FIFO serve ad immagazzinare il comando che il controllore sta eseguendo per poi renderlo disponibile sull’UART quando la scrittura del report è terminata. È utile poiché può succedere che la risposta del controllore sia abbastanza lunga da costringere lo steso controllore, al comando successivo, ad attendere che la precedente scrittura termini prima di iniziare il nuovo report.  
FIFO\_TX è la memoria che contiene i caratteri che il controllore vuole inviare sull’UART. Il controllore ha necessità di questa FIFO per essere libero di svolgere altre funzioni mentre è in corso la scrittura su UART, che dura decisamente di più di un singolo ciclo del controllore.  
Il collegamento delle due FIFO\_TX e FIFO\_TX\_SLAVE è fatto per mezzo dell’entity FIFO ARBITER che ha il compito di direzionare i dati dal controllore o dalla FIFO slave sulla FIFO master. Principalmente l’arbiter da priorità alla scrittura da parte del controllore, cioè se DN\_CHWR è disattivato. Altrimenti se DN\_CHWR è attivo scarica la FIFO slave nell’altra. Nel caso in cui durante questo scarico, DN\_CHWR si attiva, l’arbiter finisce l’operazione in corso non trasmettendo il segnale WR\_ACK, che il controllore si attende dalla FIFO.

Le FIFOs SNAP\_FIFO e REP\_FIFO servono alla porzione di sistema che si occupa di fare le istantanee dello stato dello Z80X.  
SNAP\_FIFO contiene le istantanee, chiamate anche snaps, da parte di Z80\_SNAPPER. Quest’ultima entity tramuta lo snap in una parola di 64 bit che viene immagazzinata nella FIFO. La FIFO è di dimensioni elevate, 2048 parole, perché lo snapper funziona a burst cioè se attivato fa andare lo Z80X sino al riempimento della memoria e poi blocca il clock verso il microprocessore fino a che la FIFO non viene svuotata.  
REP\_FIFO invece contiene i caratteri derivanti dalla conversione della parola dello snap memorizzato nella SNAP\_FIFO. Per cui tramuta in una serie di caratteri direttamente stampabili sulla UART la parola immagazzinata e la rende disponibile sulla REP\_FIFO. Così il controllore può semplicemente leggere un dato dalla REP\_FIFO e metterlo sulla FIFO\_TX in caso di lettura dagli snaps. Anche in questo caso la dimensione della FIFO è considerevole poiché ogni snap occupa 28 caratteri e per mantenere un buon flusso di dati serve che vi siano molti snap convertiti, già pronti per essere scritti.

Controllori FIFO, CHAR\_FEEDER\_FIFO e CHAR\_WRITER\_FIFO

Per gestire correttamente la lettura e la scrittura sulle FIFOs ho usato due entity dedicate.

CHAR\_FEEDER\_FIFO

Figura X^3 – Schema dell’entity CHAR\_FEEDER\_FIFO

Per leggere i dati dalla FIFO ho usato un’entity che si collega ai soli pin di controllo del port d’uscita della FIFO. Si occupa di segnalare all’entity a valle quando il dato è pronto e di avanzare con le parole quando necessario.

Per far ciò il port per l’entity a valle presenta tre segnali: RDY, GOT e VLD.  
RDY, abbrev. di Ready, è attivato dall’entity che legge quando è pronta per il valore successivo. Viene disattivato durante l’elaborazione del dato.  
Quando il dato presentato è stato letto ma non ancora processato, viene attivato il segnale GOT.  
CHAR\_FEEDER\_FIFO attiva la linea VLD per segnalare che il dato presente sull’uscita della FIFO è valido alla lettura.

Per cui CHAR\_FEEDER\_FIFO prima attende che la FIFO non sia più vuota, WTEMPTY, e poi che il dato presente in uscita sia valido, WTVLD. Dopodiché attiva il segnale VLD in uscita, VON. A questo punto disattiva VLD se la entity a valle a recepito, VOFF, e quando questa è pronta, legge il valore successivo, RDEN. A questo punto, se si verifica un underflow, la macchina torna all’attesa che la FIFO non sia vuota, altrimenti attende solo che il dato sia valido.

Lo scopo di questa macchina, oltre a svolgere il semplice ciclo di lettura della FIFO, è anche quello di permette la lettura della stessa FIFO da parte di più entity in contemporanea e di gestire cicli di lettura di durata non costante. Specialmente nel controllore, dove sono presenti più riconoscitori di sequenze, quest’entity temporizza le fasi di lettura sulla durata dell’esecuzione più lunga, in corso tra i riconoscitori di sequenze.

Figura X^4 – Diagramma di stato di CHAR\_FEEDER\_FIFO

CHAR\_WRITER\_FIFO

Figura X^5 - Schema dell’entity CHAR\_WRITER\_FIFO

Per scrivere su una FIFO, l’entity usa l’intero bus di controllo e fornisce da lei il dato in uscita. Però presume di leggere i dati da una ROM.

Per cui ad ogni scrittura di un blocco di dati, l’entity dev’essere resettata e in quel caso campiona il valore presente all’ingresso MAX\_ADDR che segnala quale sia la dimensione del blocco di dati. Il valore viene campionato per essere sicuri che non vari nel corso dell’elaborazione.  
Dopodiché entra nel ciclo di trasferimento. Per prima cosa attende che il valore fornito dalla ROM sia valido attivando il segnale VALID e campiona il valore presente su DIN. Poi attende che la FIFO su cui sta scrivendo abbia spazio libero. In quel caso fornisce sull’uscita il valore di DIN campionato ed emette un impulso su WR\_EN per poi attendere che la scrittura sia andata a buon fine. Altrimenti ripete le fasi di attesa e scrittura.  
La mancata attivazione di WR\_ACK è considerata un fallimento della scrittura se scatta un timer di attesa o si attiva il segnale OVERFLOW. Se la scrittura va a buon fine si incrementa l’indirizzo con cui si legge dalla ROM. L’indirizzo è fornito da un contatore variabile che conta sino al valore campionato di MAX\_ADDR e quando lo raggiunge attiva la linea RCO, abbrev. di Ripple-Carry Out. L’entity interpreta questo segnale come la fine del processo e l’ingresso nello stato DN che comunica all’esterno col segnale DONE.

Figura X^6 – Diagramma di stato di CHAR\_WRITER\_FIFO