Lo Z80 venne rilasciato sul mercato nel 1976. È formato da circa 10 000 transistor*[silicio]* e quando venne progettato fu disegnato interamente a mano*[silicio p.141].* Confrontando questo numero di transistor con quello dei 20 miliardi del recente Apple M2*[https://www.apple.com/newsroom/2022/06/apple-unveils-m2-w<ith-breakthrough-performance-and-capabilities/]*, risulta impensabile che non ci sia stato un cambiamento nella tecnologia di progettazione.

Con la crescita esponenziale della complessità dei sistemi e l’avvento della VLSI (Very Large Scale Integration) divenne indispensabile l’uso di programmi che assistano il progettista detti CAD (Computer-Aided Design tools). Questi programmi svolgono tre fasi principali: l’acquisizione di un design (capture), la sintesi (synthesis) e la conseguente verifica (verification)*[Digital design p.32]*. Però per fare tutto questo serve un mezzo per poter comunicare al programma l’organizzazione del design. A questo scopo sono stati creati i linguaggi di descrizione hardware, abbreviato HDLs (Hardware Description Languages). Con questi linguaggi si descrive per mezzo di una sintassi codificata il comportamento di un circuito logico possibilmente a tre diversi livelli di astrazione: strutturale (Structural), che descrive esplicitamente i collegamenti con gli elementi hardware base; register-transfer level RTL, che nasconde i dettagli tecnologici rispetto allo strutturale; comportamentale (Behavioural), in cui si descrive il comportamento ai pin del circuito e si lascia al tool di sintesi il compito di definire l’hardware necessario.

Figura 1.1 - Livelli di astrazione relativi agli HDL *[L.H. Crockett et al. The Zynq Book: Embedded Processing Withe ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC. Strathclyde Academic Media, 2014. ISBN: 9780992978709. URL: https://books.google.it/books?id=9dfvoAEACAAJ]*

Il VHDL è assieme al Verilog il linguaggio HDL più usato che permette di descrivere un circuito digitale. Questi circuiti vengono chiamati design entities, o più brevemente entities. Questa è composta da due parti l’entity declaration, in cui si definisce l’interfaccia dell’entity con il mondo esterno, e l’architecture body, che definisce le operazioni interne dell’entity e cioè l’implementazione della stessa, come mostrato nella Figura 1.2 *[Digital design p.14]*L’entity declaration è anche detta port poiché l’elenco di segnali di connessione è preceduta da questa parola chiave. Nell’entity declaration si può aggiungere anche un altro campo detto generic. In questo campo si possono definire delle variabili che generalizzano l’entity così da slegare il design da particolarità modificabili come la dimensione. Per esempio si può voler descrivere un sommatore e lasciare generico il numero di bit. Usando il campo generic, si definisce una variabile che rappresenta il numero di bit che verrà usata per definire i segnali del port e l’implementazione. Però questa rimane non definita sino all’istanziazione del componente, detta instantiation, in un entity di ordine superiore. Con questo vantaggio del VHDL si può riutilizzare la stesa entity in condizioni diverse.All’interno dell’architecture body, detta anche architecture, possono essere definiti dei segnali, signals, utilizzati per collegare più parti interne. Queste possono essere a loro volta istanze di altre entities. I segnali possono rappresentare tipi elementari, come singoli bit, vettori di bit ordinati che rappresentano i bus, numeri interi oppure tipi più strutturati come array o record.

Figura 1.2 – Esempio di design entity VHDL*[Digital design p.130]*

L’assegnazione di valori ai segnali avviene in maniera simultanea per mezzo di assegnazione diretta, concurrent assignement, o condizionale, conditional assignment. Insieme definiscono l’elemento base dell’architecture, il concurrent statement. Per cui nel VHDL, a differenza di altri linguaggi di programmazione procedurale, tutte le entity e le assegnazioni simultanee sono attive ed eseguite allo stesso tempo.

Un tipo particolare di concurrent statement è il costrutto process. Questo costrutto può racchiudere delle assegnazioni o altri costrutti tipici dei linguaggi di programmazione con if-then-else, while loop e for loop per svolgere operazioni più complesse. Un process viene attivato quando uno dei segnali definiti nella lista di segnali che segue la parola chiave, detta sensitivity list, cambia di stato cioè avviene un evento.

Il VHDL può essere usato sia per descrivere entities sintetizzabili, che si traducono poi in un vero e proprio circuito, o entities che fanno da banchi di prova per testare il funzionamento di altre entities. In questo caso, ci si riferisce all’entity che attua il test come testbench e alla testata come DUT (Device Under Test) per mantenere un’analogia con il mondo fisico. Nel testbench vengono definiti dei process con particolari funzioni che sfruttano l’esecuzione sequenziale e ciclica del codice presente all’interno svolto comunque in parallelo agli altri concurrent statements. Per cui ci sono solitamente dei process per generare dei segnali di temporizzazione per mezzo di direttive wait for ed altri process che generano gli stimoli da dare al DUT per verificarne il funzionamento. Per cui il VHDL e i linguaggi HDL più in generale permettono sia di sintetizzare che creare dei sistemi per simulare il comportamento di altri dispositivi.

Avendo la possibilità di descrivere un sistema per mezzo di un CAD adatto, si può sviluppare interamente un sistema. Si può seguire un processo iterativo, come mostrato nella Figura 1.3, che partendo dalle specifiche si arriva alla descrizione del comportamento dell’entity, partizionarla in blocchi elementari più piccoli se necessario. Dopodiché si possono simulare le entities per mezzo di testbech dedicati e verificarne il funzionamento. Inoltre si può verificare il rispetto delle specifiche elencate e la conseguente ottimizzazione o approvazione. In questo modo si può progettare velocemente un sistema arrivando all’implementazione fisica che è la parte più onerosa anche rispetto al tempo solo alla fine. *[Digital Design p.23]*

Figura 1.3 – Processo iterativo di progettazione *[Digital Design p.23]*

Per questo progetto ho adottato esattamente questo metodo. Partendo dall’esterno della scheda verso l’interno ho definito le specifiche dell’entity dove necessario. Nel caso dello Z80, le specifiche erano già definite nel datasheet. Dopodiché ho ricorsivamente diviso le entities in altre più piccole e semplici. Di seguito ho fatto l’implementazione del design in VHDL e la conseguente simulazione di verifica seguendo la metodologia iterativa. Solo nel momento in cui arrivavo a un sistema funzionante sulla simulazione lo sintetizzavo sull’FPGA e ne facevo una verifica sulla scheda poiché quest’ultimo passaggio richiedeva la maggior parte del tempo.

Un altro grande vantaggio dell’uso dei linguaggi HDL è la portabilità. Un’entity definita per un determinato ambiente può essere trasportata senza troppe difficoltà in un altro. Il tool di sintesi si occuperà di adattare la tecnologia al comportamento richiesto. Per far ciò serve uno standard di riferimento del linguaggio e di conseguenza ne vennero pubblicati vari. Tra questi il primo è IEEE Std 1076 del 1987, che definisce il linguaggio, e rinnovato secondo le nuove versioni, la cui più recente è VHDL-2008. Un altro standard importante è IEEE 1164 del 1993 in cui vengono standardizzati i valori logici usati e le conseguenti operazioni logiche per permettere una sintesi e simulazione più dettagliata.

Per poter usare l’FPGA su cui ho implementato la scheda di sviluppo, ho usato il CAD fornito dal produttore dell’FPGA, Xilinx. L’azienda mette a disposizione varie versioni dell’ambiente di sviluppo in base all’FPGA usata. Nel mio caso utilizzavo una Spartan 6 che permetteva solo l’uso di ISE 14.7 con VHDL-93. Quest’ambiente mette a disposizione oltre ai tool di sintesi anche un simulatore, un programmatore per l’FPGA e un analizzatore logico integrato detto ILA. Quest’ultimo se istanziato nel progetto, invia al PC lo stato dei segnali collegati all’ILA attraverso lo stesso protocollo di programmazione.

Lo stesso tool mette a disposizione un’interfaccia dedicata per inserire nel progetto degli IPs, chiamati IP cores. Gli IPs, abbreviazione di Intellectual Properties, sono entity che come mattoncini possono essere uniti al progetto principale per svolgere delle funzioni specifiche. Gli IPs coprono un vasto numero di funzioni: possono descrivere memorie, interfacce verso protocolli di comunicazione o svolgere compiti particolari come algoritmi di compressione. Gli IPs vengono usati come black boxes facendo riferimento solo al loro port e al datasheet allegato. Un esempio noto di IP sono i core ARM che vengono autorizzati per essere usati nei vari design.*[Digital design p.195]*

Nel progetto ho fatto uso di alcuni IPs, messi a disposizione dal tool, per usare delle special features dell’FPGA quali Block RAMs e FIFOs e di un IP open-source per la gestione dell’interfaccia UART.