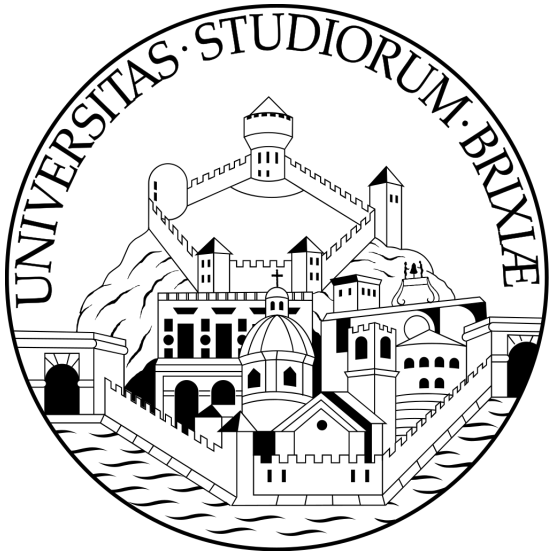
Università degli Studi di Brescia

Progetto di sistemi elettronici digitali

Implementazione della periferica UART per un processore MIPS su FPGA



MAINETTI GAMBERA EUGENIO ENRICO 86097

MARELLA SILVIA 710047

Indice

1 Obiettivo

L'architettura **MIPS** (Microprocessor without Interlocked Pipeline Stages) è un'architettura informatica per microprocessori **RISC** ed è particolarmente diffusa per quanto riguarda i sistemi embedded. Questo progetto ha come obiettivo l'aggiunta di un protocollo di comunicazione su un processore MIPS già implementato. In particolare si richiede di modificare l'architettura del MIPS in modo da aggiungere l'interfaccia di comunicazione **UART** al fine di far comunicare il MIPS, implementato su un FPGA, con un altro dispositivo.

2 MIPS

Come è stato detto in precedenza, il **MIPS (Microprocessor without Interlocked Pipeline Stages)** è un tipo di architettura utilizzata per microprocessori di tipo **RISC (Reduced Instruction Set Computer)**. Microprocessori di questo tipo sono caratterizzati da un *instruction set* ridotto rispetto ai microprocessori di tipo **CISC (Complex Instruction Set Computer)** e sono molto utilizzati per quanto riguarda i sistemi embedded.
Il MIPS utilizzato è un processore a 32 bit implementato in VHDL. Il codice che implementa il MIPS utilizzato in questo progetto è stato sviluppato da Corrado Santoro ed è organizzato come in figura 1.

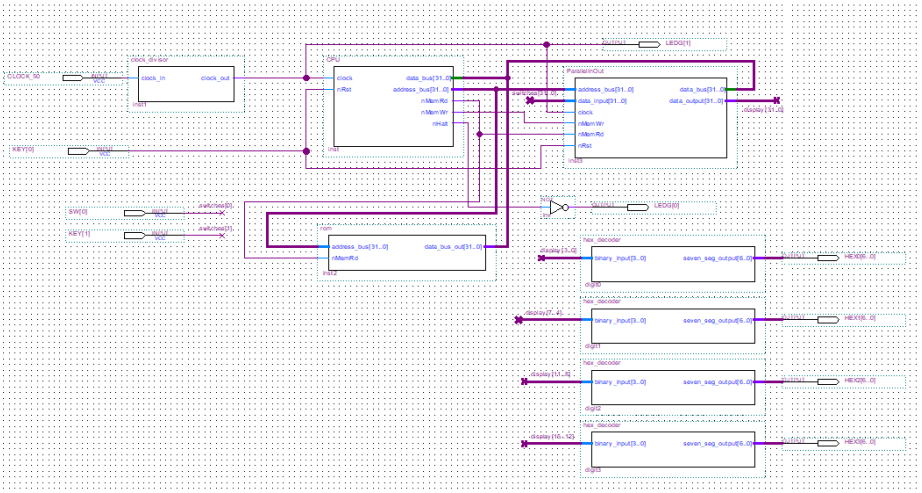


Figura 1: Schema a blocchi del codice che implementa il MIPS utilizzato

I componenti principali di cui è costituito lo schema sono:

- **La CPU**: implementa la gestione dei registri e delle operazioni elementari da svolgere. La CPU ha come ingressi il segnale di clock e il segnale di reset asincrono e come uscite il data_bus, l'address_bus, il bit di read, il

bit di write e il bit di halt. La CPU è la parte principale del MIPS, infatti gestisce i registri, implementa la funzione di *fetch* come una macchina a stati e tutte le operazioni base, che sono definite dai diversi opcode in un file omonimo incluso nel file che implementa la CPU.

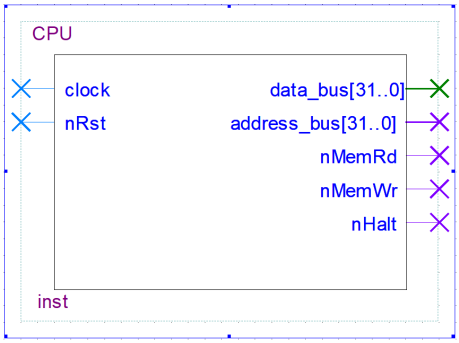


Figura 2: Schema a blocchi della CPU

- **La ROM**: La ROM ha come ingressi l'address_bus e il bit di read e come unica uscita il data_bus_out. Nella ROM sono implementati gli step da eseguire in base all'ingresso dell'address_bus. Infatti in base al valore di address_bus viene modificata l'uscita di data_bus_out. Il valore di data_bus_out sarà l'ingresso data_bus della CPU e del ParallelInOut.

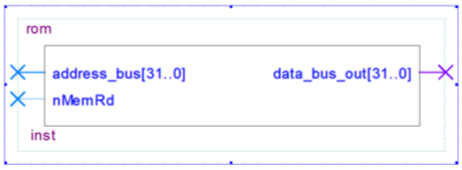


Figura 3: Schema a blocchi della ROM

- **Il ParallelInOut**: Il ParallelInOut ha come ingressi l'address_bus, il data_input, il segnale di clock, il bit di lettura e scrittura e il reset; come uscita ha il data_output e come inout ha il data_bus. Il ParallelInOut ha la funzione di controllare il valore dell'address_bus e dei bit di lettura e scrittura in modo da modificare data_bus e data_output. Modificando il data_bus, la CPU ritornerà un valore diverso di address_bus che, a sua volta, influirà sul funzionamento della ROM; mentre il data_output sarà codificato nell'hex_decoder per essere portato in uscita.

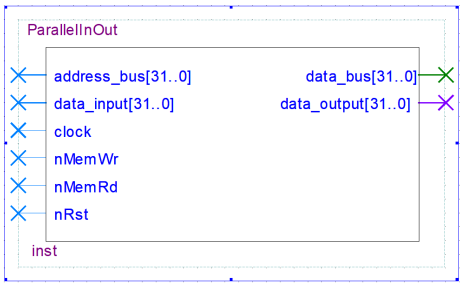


Figura 4: Schema a blocchi del ParallelInOut

Inoltre sono stati anche implementati il clock divisor, che permette di far funzionare il sistema alla frequenza di $f_{clk} = 100Hz$, e gli hex_decoder, che servono per visualizzare il numero in notazione esadecimale sul display. Il processore è implementato per mostrare sul display un numero in formato esadecimale e incrementarlo o decrementarlo.

3 L'interfaccia UART

L'interfaccia **UART** è uno standard di comunicazione di tipo **seriale asincrono** a tre fili (Tx, Rx e GND). Questo tipo di trasmissione richiede l'utilizzo di 10 bit:

- 1 start bit
- 8 data bit
- 1 stop bit

L'asincronia del sistema di comunicazione è possibile grazie al fatto che sia ricevente che trasmettente lavorano allo stesso **baudrate** (in particolare in questo sistema viene usato un baudrate di 115200) e quindi non è necessaria una linea dedicata al clock. Il protocollo di comunicazione prevede una linea alta in caso di *idle* del sistema, lo start bit viene generato abbassando la linea per un tempo pari almeno al *bitrate*, eventuali errori di lettura vengono prevenuti da un *oversampling* eseguito tipicamente ad una velocità 16 volte maggiore a quella del baudrate. Una volta identificato lo start bit il ricevitore si prepara a leggere 8 bit in sequenza che compongono il byte che viene trasmesso e attende il bit finale che consiste in un segnale alto.

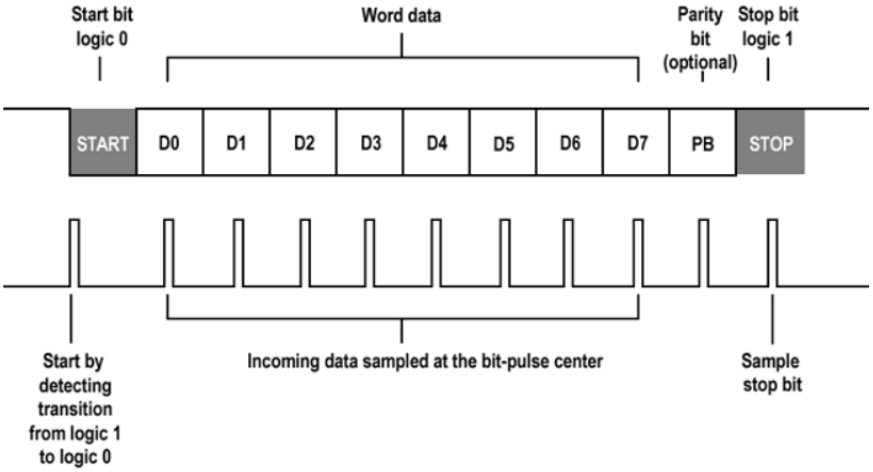


Figura 5: Frame UART

Opzionalmente tra l'ottavo data bit e lo stop bit può essere implementato un bit per i controlli di parità che tuttavia in questo progetto sono stati trascurati a beneficio di una maggiore velocità di trasmissione e semplicità del codice. Il codice VHDL che implementa questo standard è stato fornito da Jakub Cabal ed è costituito da un top level file in cui sono definite le specifiche principali come la frequenza di clock e il baude rate della trasmissione oltre che l'over-sampling. Vengono definiti anche due moduli: il modulo Tx, responsabile della trasmissione dei dati, e il modulo Rx, deputato alla ricezione. Inoltre poiché nella maggior parte dei computer moderni la periferica seriale per ususfruire direttamente dell'UART non è presente, è stato utilizzato un convertitore UART to USB analogo a quello mostrato in figura 6.

Figura 6: convertitore UART to USB

4 Codice implementato

Il codice implementato quindi ha il compito di **collegare l'interfaccia di trasmissione seriale con il processore MIPS** in modo che l'FPGA su cui è caricato il processore possa scambiare dati con un altro dispositivo, che in questo caso è un PC. Per lo scambio dei dati sono stati utilizzati tre pin del GPIO dell'FPGA (ricezione, trasmissione e ground), inoltre è stato utilizzato il display 7-segment per visualizzare i caratteri in esadecimale ricevuti dal PC. Partendo quindi dal codice del MIPS, è stato necessario modificare in particolare la sezione che implementa il ParallelInOut e quella che implementa la ROM. In particolare la sezione dell'UART è stata inclusa nella sezione del ParallelInOut. Uno schema semplificato dello schema a blocchi completo con le relative sezioni da modificare è rappresentato in figura 7.

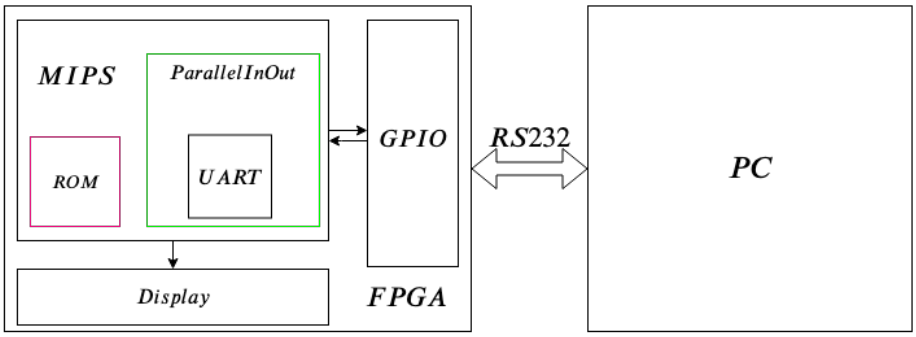


Figura 7: Schema a blocchi del progetto: le parti evidenziate sono le sezioni a cui sono state apportate le modifiche

Le modifiche che sono state apportate al ParallelInOut riguardano l'aggiunta di un ingresso e un'uscita: in particolare è stato aggiunto come ingresso il segnale **UART_RXD** e come uscita il segnale **UART_TXD**, entrambi connessi a due pin del GPIO a cui è stato connesso il cavo per la trasmissione seriale. Il blocco del ParallelInOut modificato è mostrato in figura 8.

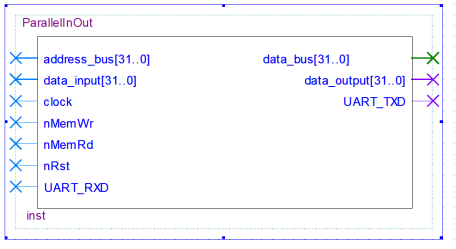


Figura 8: Schema del ParallelInOut a cui sono state apportate le modifiche per la trasmissione seriale.

Il codice che implementa il ParallelInOut è costituito da un process principale in cui si controlla il valore dell'address_bus in ingresso al blocco, in particolare se l'address_bus ha valore esadecimale pari a 00008000, vengono modificati i valori di data_bus e data_output con il valore dell'address_bus in modo da leggere il contenuto del registro stesso e mostrarlo in uscita sul pannello a sette segmenti. In alternativa, se l'address_bus ha valore esadecimale pari a 00009000, porta il valore di data_send a 1, in modo da abilitare la trasmissione seriale. Questa porzione di codice è stata rappresentata in figura 9.

```
process(clock,nRst)
begin

   if nRst = '0' then
      data_output  <= (others => '0');
      data_bus <= (others => 'Z');
      data_send <= '0';
   elsif (clock'event and clock = '0') then

      if (address_bus = x"00008000" and busy = '0' ) then -- address is 0x8000
          data_send <= '0';

         if nMemWr = '0' then

            data_output <= data_bus ;

         elsif nMemRd = '0' then
            data_bus <= x"000000" & data_out;
            data_output <= x"000000" & data_out;
         else
            data_bus <= (others => 'Z');
         end if;

      elsif (address_bus =x"00009000" and busy = '0') then
      data_in <= data_bus(7 downto 0);
      data_send <= '0';
      data_output <= x"00000010";
      data_send <= '1';
      data_output <= x"00000001";
         end if;
   end if;
end process;
```

Figura 9: Codice contenuto nella sezione del ParallelInOut

Per quanto riguarda invece la ROM, il codice modificato è rappresentato in figura 10. In particolare, è stato riportato solo il process dell'architecture della ROM, in cui è presente un case principale determinato dal valore dell'address_bus. La ROM comprende diverse operazioni al variare dell'address_bus, in particolare legge il valore contenuto nel registro 8000 (quindi, legge il dato ricevuto dalla trasmissione seriale) e, se viene inviato un particolare numero (che in questo caso risulta essere 09) inizia a trasmettere quel numero in eco.

```
   process(nMemRd)
   begin
         if nMemRd = '0' then
            case address_bus is

when x"00000000" =>      out_byte <= NOP; -- non esegue nulla
when x"00000001" =>      out_byte <= LW  & R0 & R1  & x"8000";
when x"00000002" =>      out_byte <= ADD_I & R0 & R2 & x"0009";
when x"00000003" =>      out_byte <= BEQ & R1 & R2 & x"0005";
when x"00000004" =>      out_byte <= JUMP  & "00" & x"000001";
when x"00000005" =>      out_byte <= SW    & "00000" & R1 & x"9000";
when x"00000006" =>      out_byte <= JUMP  & "00" & x"000001";

               when others =>       out_byte <= (others => 'Z');
            end case;
         else
            out_byte <= (others => 'Z');
         end if;
   end process;
```

Figura 10: Codice contenuto nella sezione della ROM

In riferimento quindi allo schema mostrato in figura 7 e alle modifiche elencate, il codice modificato è rappresentato in figura 11.

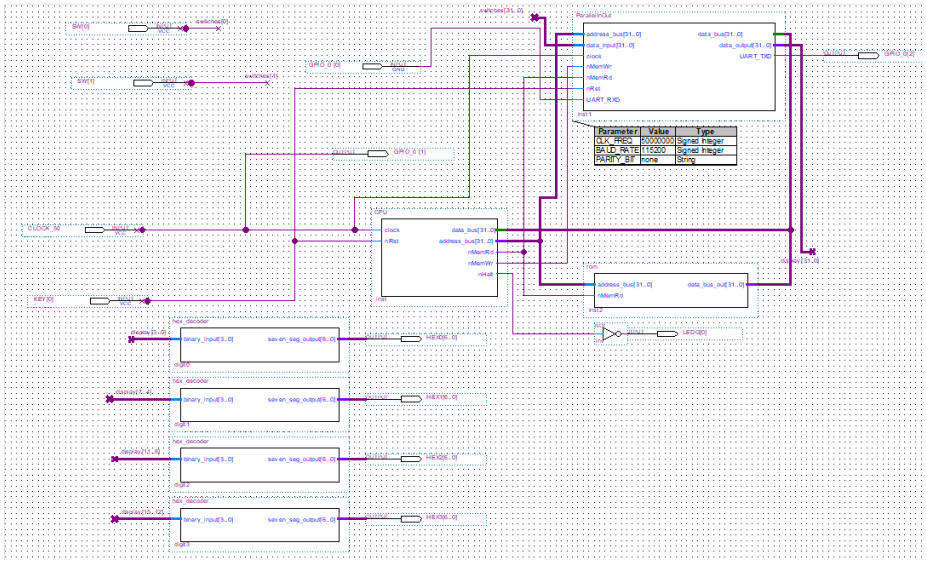


Figura 11: Schema a blocchi del codice che implementa il MIPS connesso all'interfaccia UART

Il programma VHDL quindi esegue le seguenti operazioni:

1. Salva i dati ricevuti dalla seriale che vengono salvati nel registro 8000
2. Legge i dati del registro e li porta in output sul display dell'FPGA
3. Se il dato ricevuto risulta essere uguale a 09, scrive il valore sul registro 9000
4. Legge il dato sul registro 9000 e lo porta in uscita alla trasmissione seriale, trasmettendolo.

5 Conclusioni

In conclusione, è stato implementato un protocollo di comunicazione preesistente su un processore MIPS già implementato in VHDL. In particolare, è stata modificata l'architettura del MIPS in modo da aggiungere l'interfaccia di comunicazione UART al fine di far comunicare il MIPS con un altro dispositivo, che in questo caso è un PC. Il codice implementato permette al PC di inviare numeri esadecimali, che vengono ricevuti e visualizzati sul display dell'FPGA. All'invio di un particolare numero viene innescata la trasmissione dati da FPGA a PC, in cui l'FPGA trasmette in eco un carattere.