ELABORATI DI PROGETTAZIONE SISTEMI EMBEDDED

Enrico Giordano VR386687

Progetto 1: implementazione moltiplicatore in SystemC RTL

# Descrizione del progetto

Il progetto consisteva nella realizzazione di un moltiplicatore in virgola mobile, doppia precisione, tra due numeri con virgola.

Il risultato finale è formato da due moduli SystemC: un moltiplicatore implementato come FSMD (controllore + datapath) e un testbench per stimolare e verificare il progetto; tutto questo è stato diviso in più file:

* main.cc : descrive la modalità di collegamento tra il modulo mul\_RTL e il testbench;
* mul\_RTL.cc: descrive il modulo che esegue la moltiplicazione tra due numeri e genera in uscita il risultato;
* mul\_RTL\_testbench.cc : descrive il modulo testbench.

# Interfaccia

L’interfaccia è descritta nel file main.cc, che si occupa di collegare il modulo che esegue la moltiplicazione e il testbench per testarlo.

L’interfaccia si occupa di :

* creare le istanze del modulo del moltiplicatore e del modulo di testing;
* dichiarare le porte di ingresso e uscita del sistema e i segnali che permettono la comunicazione dei moduli precedentemente istanziati;
* eseguire il binding delle porte, ovvero di associare le porte ai rispettivi moduli garantendone la comunicazione.

I segnali istanziati nell’interfaccia sono i seguenti:

* clock: <sc\_logic> rappresenta il segnale sincrono di clock che gestisce lo scorrere degli istanti di tempo del sistema;
* reset\_signal: <bool> viene utilizzato per resettare il sistema.
* p\_In\_data1: <sc\_lv<64>> rappresenta un numero da moltiplicare;
* p\_In\_data2: <sc\_lv<64>> rappresenta un numero da moltiplicare;
* p\_In\_enable: <sc\_uint<1>> indica se può iniziare la computazione;
* p\_Out\_enable: <sc\_uint<1>> indica se è pronto un dato in uscita;
* p\_result: <sc\_lv<64>> rappresenta il risultato della moltiplicazione;

p\_In\_enable e p\_Out\_enable sono segnali che determinano rispettivamente l’inizio e la fine dell’esecuzione dell’algoritmo; di conseguenza vengono utilizzati come flag di collegamento e sincronizzazione tra il modulo di esecuzione e quello di test.

Per effettuare il collegamento, è stato necessario istanziare nel main i due moduli (mul\_RTL\_testbench e mul\_RTL) e collegare ogni loro segnale, input e output con i segnali sopra descritti. Dopo averli collegati, per far partire la simulazione si esegue la funzione sc\_start().

# Descrizione EFSM

L’EFSM, descritta nel file mul\_RTL.cc, è il modulo di elaborazione del sistema e rappresenta l’unione di datapath e macchina a stati finiti dell’algoritmo della moltiplicazione.

L’algoritmo che rappresenta questa macchina a stati è il seguente:

1. SR: reset di tutti i segnali e uscite;
2. S0: preparazione del sistema e inizializzazione delle porte di uscita e di ingresso (si scrive 0 in queste porte in modo da eliminare ogni segnale “sporco”);
3. SoI: lettura dei numeri ed estrapolazione delle singole parti di questi (segno, esponente e mantissa in variabili interne per facilitare i calcoli);
4. S1: somma degli esponenti per ottenere l’esponente del risultato;
5. S2: polarizzazione dell’esponente risultante (togliere 1023 da esso);
6. S3: prodotto tra mantisse in interazione (parte di controllo), ovvero controllo se si devono fare altri passi per la moltiplicazione (in tutto si fanno 52 controlli perché si deve controllare ogni bit della mantissa del secondo numero);
7. S31: prodotto tra mantisse in interazione (parte di esecuzione), ovvero se la mantissa del secondo numero all’indice i-esimo vale ‘1’, si somma alla mantissa risultante la mantissa del primo numero shiftandola di i volte;
8. S4: normalizzazione del risultato in interazione (parte di esecuzione), ovvero si controlla che non ci siano numeri dopo il 104-esimo bit, se ce ne sono si shifta la mantissa risultante a destra di 1 e si incrementa di 1 l’esponente risultante, altrimenti si shifta di 1 a sinistra la mantissa e si decrementa l’esponente;
9. S5: controllo di overflow, ovvero se l’esponente vale 0 o se la mantissa vale 0;
10. S7: normalizzazione del risultato in interazione (parte di controllo), ovvero si controlla se bisogna normalizzare ancora;
11. S8: calcolo del segno;
12. S9: preparazione del risultato da mandare in output.

Il file che contiene l’ESFSM contiene 2 metodi:

* void mul\_RTL :: elaborate\_mul\_FSM(void), che contiene la parte datapath (di calcolo);
* void mul\_RTL :: elaborate\_mul(void), che contiene la parte di fsm (cambio di stato).

# Descrizione dei processi

La procedura di esecuzione del sistema prevede l’istanziazione di due metodi descritti attraverso il comando “SC\_METHOD”, che sono rispettivamente i due processi sopradescritti. Questi processi sono dichiarati di tipo method poiché la loro esecuzione viene eseguita totalmente e può essere richiamata dal sistema più volte, senza mai essere interrotta.

I metodi sono sensibili ai cambiamenti di valore di una serie di segnali che compongono, per ogni metodo, una sensitivity list; al variare di valore di ogni segnale appartenente alla sensitivity list di un metodo, il metodo viene eseguito nuovamente. Un ulteriore procedura di istanziazione dei processi è quella che fa uso delle thread attraverso il comando “SC\_THREAD”. Il processo di testing è istanziato come thread, questo vuol dire che può essere lanciato una volta sola e interrotto attraverso il comando wait(), questo perché il testbench deve chiamare ed attendere il termine dell’esecuzione del modulo mul\_RTL. Tutti i processi sono sensibili al variare del clock.

mul\_RTL ha la sensitivity list descritta in questo modo:

sensitive << STATUS << isready << number\_a << number\_b;

mentre elaborate\_mul\_FSM ha questa sensitivity list:

sensitive << reset.neg();

sensitive << clk.pos();