

Esempio di progetto di un microprocessore in Gezel e VHDL

Esercitazione 07 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2019-20

Indice

1. Esempio di progetto di un microprocessore in Gezel e VHDL
2. argomenti dell'esercitazione
3. architettura del microprocessore e tecniche di descrizione
4. codifica di campi della microistruzione: Output, SBUS, ALU
5. codifica di campi della microistruzione: Shifter, Dest
6. codifica del campo Next e della microistruzione
7. microprogramma e datapath del modulo di controllo (1)
8. datapath del modulo di controllo (2)
9. datapath del banco dei registri
10. datapath dell'ALU
11. datapath dello Shifter
12. datapath del processore microprogrammato
13. riprogrammazione della microarchitettura
14. esperienza di laboratorio
15. riferimenti

in questa esercitazione si trattano:

- architettura di un processore microprogrammato
- tecniche di descrizione della microarchitettura
- codifica dei campi della microistruzione
- microprogramma nella memoria di controllo
- datapath dei moduli componenti
- datapath del processore microprogrammato
- datapath di un testbench e simulazione
- esperienza di laboratorio:
 1. riprogrammazione del microprocessore per una diversa applicazione: calcolo del delay di traiettorie di Collatz
 2. estensione del modello con una FSM per realizzare le funzioni considerate nell'esperienza di laboratorio precedente e sua implementazione su FPGA

architettura del microprocessore e tecniche di descrizione

consideriamo la microarchitettura schematizzata in figura 6.7, dotata di un microprogramma per il calcolo del GCD con l'algoritmo di Euclide e con la piccola estensione di una porta di output per il risultato del calcolo

il bit più a sinistra, non usato, nel formato della microistruzione mostrato nella stessa figura sarà qui usato per indicare la validità del dato in uscita

la descrizione formale della microarchitettura di questo processore dedicato sfrutta le seguenti tecniche:

- direttive di preprocessore C, in particolare la direttiva `#define`, per la definizione della codifica delle microistruzioni e dei loro campi
- descrizione in Gezel dell'unità di controllo, che include il microprogramma nella memoria di controllo, e del datapath quale collezione di moduli componenti: banco di registri, ALU e shifter
- il modulo Gezel in cima alla gerarchia interconnette l'unità di controllo e gli altri moduli componenti, e genera segnali di controllo di I/O (strobe)
- la descrizione comprende un semplice testbench che fornisce al processore una sequenza di coppie in input e stampa i risultati che ne raccoglie

codifica di campi della microistruzione: Output, SBUS, ALU

```
// wordlength in the datapath
#define WLEN 16

/* encoding for data output */
#define O_NIL 0 /* OT <- 0 */
#define O_WR 1 /* OT <- SBUS */

/* encoding for SBUS multiplexer */
#define SBUS_R0 0 /* SBUS <- R0 */
#define SBUS_R1 1 /* SBUS <- R1 */
#define SBUS_R2 2 /* SBUS <- R2 */
#define SBUS_R3 3 /* SBUS <- R3 */
#define SBUS_R4 4 /* SBUS <- R4 */
#define SBUS_R5 5 /* SBUS <- R5 */
#define SBUS_R6 6 /* SBUS <- R6 */
#define SBUS_R7 7 /* SBUS <- R7 */
#define SBUS_IN 8 /* SBUS <- IN */
#define SBUS_X SBUS_R0 /* don't care */

/* encoding for ALU */
#define ALU_ACC 0 /* ALU <- ACC */
#define ALU_PASS 1 /* ALU <- SBUS */
#define ALU_ADD 2 /* ALU <- ACC + SBUS */
#define ALU_SUBA 3 /* ALU <- ACC - SBUS */
#define ALU_SUBS 4 /* ALU <- SBUS - ACC */
#define ALU_AND 5 /* ALU <- ACC and SBUS */
#define ALU_OR 6 /* ALU <- ACC or SBUS */
#define ALU_NOT 7 /* ALU <- not SBUS */
#define ALU_INCS 8 /* ALU <- SBUS + 1 */
#define ALU_INCA 9 /* ALU <- ACC + 1 */
#define ALU_CLR 10 /* ALU <- 0 */
#define ALU_SET 11 /* ALU <- 1 */
#define ALU_X ALU_ACC /* don't care */
```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 1)

codifica di campi della microistruzione: Shifter, Dest

```
/* encoding for shifter */
#define SHFT_SHL 1 /* Shifter <- shiftright(alu) */
#define SHFT_SHR 2 /* Shifter <- shiftright(alu) */
#define SHFT_ROL 3 /* Shifter <- rotateleft(alu) */
#define SHFT_ROR 4 /* Shifter <- rotateright(alu) */
#define SHFT_SLA 5 /* Shifter <- shiftrightarithmetical(alu) */
#define SHFT_SRA 6 /* Shifter <- shiftrightarithmetical(alu) */
#define SHFT_NIL 7 /* Shifter <- ALU */
#define SHFT_X SHFT_NIL /* don't care */

/* encoding for result destination */
#define DST_R0 0 /* R0 <- Shifter */
#define DST_R1 1 /* R1 <- Shifter */
#define DST_R2 2 /* R2 <- Shifter */
#define DST_R3 3 /* R3 <- Shifter */
#define DST_R4 4 /* R4 <- Shifter */
#define DST_R5 5 /* R5 <- Shifter */
#define DST_R6 6 /* R6 <- Shifter */
#define DST_R7 7 /* R7 <- Shifter */
#define DST_ACC 8 /* ACC <- Shifter */
#define DST_NIL 15 /* not connected <- shifter */
#define DST_X DST_NIL /* don't care instruction */
```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 2)

codifica del campo Next e della microistruzione

```

/* encoding for jump field */
#define NXT_NXT 0      /* CSAR <- CSAR + 1 */
#define NXT_JMP 1      /* CSAR <- Address */
#define NXT_JC 2       /* CSAR <- (carry==1)? Address : CSAR + 1 */
#define NXT_JNC 10     /* CSAR <- (carry==0)? Address : CSAR + 1 */
#define NXT_JZ 4        /* CSAR <- (zero==1)? Address : CSAR + 1 */
#define NXT_JNZ 12     /* CSAR <- (zero==0)? Address : CSAR + 1 */
#define NXT_X NXT_NXT

/* encoding for the micro-instruction word */
#define MI(OUT, SBUS, ALU, SHFT, DEST, NXT, ADR) \
    (OUT << 31) | \
    (SBUS << 27) | \
    (ALU << 23) | \
    (SHFT << 20) | \
    (DEST << 16) | \
    (NXT << 12) | \
    (ADR)

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 3)

microprogramma e datapath del modulo di controllo (1)

```

dp control(    in  carry, zero      : ns(1);
               out ctl_ot           : ns(1);
               out ctl_sbusr        : ns(4);
               out ctl_alu          : ns(4);
               out ctl_shft         : ns(3);
               out ctl_dest         : ns(4)) {

    lookup cstore : ns(32) = {
        // 0 Lstart:      IN -> R0
        MI(O_NIL, SBUS_IN, ALU_PASS, SHFT_NIL, DST_R0, NXT_NXT, 0),
        // 1              IN -> ACC
        MI(O_NIL, SBUS_IN, ALU_PASS, SHFT_NIL, DST_ACC, NXT_NXT, 0),
        // 2 Lcheck:      (R0 - ACC)      || JUMP IF Z Ldone
        MI(O_NIL, SBUS_R0, ALU_SUBS, SHFT_NIL, DST_NIL, NXT_JZ, 6),
        // 3              (R0 - ACC) << 1 || JUMP IF C Lsmall
        MI(O_NIL, SBUS_R0, ALU_SUBS, SHFT_SHL, DST_NIL, NXT_JC, 5),
        // 4              R0 - ACC -> R0   || JUMP Lcheck
        MI(O_NIL, SBUS_R0, ALU_SUBS, SHFT_NIL, DST_R0, NXT_JMP, 2),
        // 5 Lsmall:      ACC - R0 -> ACC  || JUMP Lcheck
        MI(O_NIL, SBUS_R0, ALU_SUBA, SHFT_NIL, DST_ACC, NXT_JMP, 2),
        // 6 Ldone:       R0 -> OUT        || JUMP Lstart
        MI(O_WR, SBUS_R0, ALU_X, SHFT_X, DST_X, NXT_JMP, 0)
    };
}

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 4)

datapath del modulo di controllo (2)

```

reg csar          : ns(12);
sig mir           : ns(32);
sig ctl_nxt       : ns(4);
sig csar_nxt      : ns(12);
sig ctl_address   : ns(12);

always {
    mir = cstore(csar);
    ctl_ot      = mir[31];
    ctl_sbus    = mir[27:30];
    ctl_alu     = mir[23:26];
    ctl_shft    = mir[20:22];
    ctl_dest    = mir[16:19];
    ctl_nxt     = mir[12:15];
    ctl_address = mir[ 0:11];
    csar_nxt = csar + 1;
    csar =
        (ctl_nxt == NXT_NXT) ? csar_nxt :
        (ctl_nxt == NXT_JMP) ? ctl_address :
        (ctl_nxt == NXT_JC) ? ((carry==1) ? ctl_address : csar_nxt) :
        (ctl_nxt == NXT_JZ) ? ((zero==1) ? ctl_address : csar_nxt) :
        (ctl_nxt == NXT_JNC) ? ((carry==0) ? ctl_address : csar_nxt) :
        (ctl_nxt == NXT_JNZ) ? ((zero==0) ? ctl_address : csar_nxt) :
        csar;
}

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 5)

datapath del banco dei registri

```

dp regfile ( in  ctl_dest : ns(4);
               in  ctl_sbus  : ns(4);
               in  data_in  : ns(WLEN);
               out data_out  : ns(WLEN)) {
    reg r0 : ns(WLEN);
    reg r1 : ns(WLEN);
    reg r2 : ns(WLEN);
    reg r3 : ns(WLEN);
    reg r4 : ns(WLEN);
    reg r5 : ns(WLEN);
    reg r6 : ns(WLEN);
    reg r7 : ns(WLEN);
    always {
        r0 = (ctl_dest == DST_R0) ? data_in : r0;
        r1 = (ctl_dest == DST_R1) ? data_in : r1;
        r2 = (ctl_dest == DST_R2) ? data_in : r2;
        r3 = (ctl_dest == DST_R3) ? data_in : r3;
        r4 = (ctl_dest == DST_R4) ? data_in : r4;
        r5 = (ctl_dest == DST_R5) ? data_in : r5;
        r6 = (ctl_dest == DST_R6) ? data_in : r6;
        r7 = (ctl_dest == DST_R7) ? data_in : r7;
        data_out =
            (ctl_sbus == SBUS_R0) ? r0 :
            (ctl_sbus == SBUS_R1) ? r1 :
            (ctl_sbus == SBUS_R2) ? r2 :
            (ctl_sbus == SBUS_R3) ? r3 :
            (ctl_sbus == SBUS_R4) ? r4 :
            (ctl_sbus == SBUS_R5) ? r5 :
            (ctl_sbus == SBUS_R6) ? r6 :
            (ctl_sbus == SBUS_R7) ? r7 :
            r0;
    }
}

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 6)

datapath dell'ALU

```

dp alu (   in  ctl_dest : ns(4);
           in  ctl_alu  : ns(4);
           in  sbus     : ns(WLEN);
           in  shift    : ns(WLEN);
           out q        : ns(WLEN);

           reg acc : ns(WLEN);
           always {
               q = (ctl_alu == ALU_ACC) ? acc :
                   (ctl_alu == ALU_PASS) ? sbus :
                   (ctl_alu == ALU_ADD) ? acc + sbus :
                   (ctl_alu == ALU_SUBA) ? acc - sbus :
                   (ctl_alu == ALU_SUBS) ? sbus - acc :
                   (ctl_alu == ALU_AND) ? acc & sbus :
                   (ctl_alu == ALU_OR) ? acc | sbus :
                   (ctl_alu == ALU_NOT) ? ~ sbus :
                   (ctl_alu == ALU_INCS) ? sbus + 1 :
                   (ctl_alu == ALU_INCA) ? acc + 1 :
                   (ctl_alu == ALU_CLR) ? 0 :
                   (ctl_alu == ALU_SET) ? 1 :
                   0;
               acc = (ctl_dest == DST_ACC) ? shift : acc;
           }
       }

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 7)

datapath dello Shifter

```

dp shifter ( in  ctl      : ns(3);
              out zero    : ns(1);
              out cy      : ns(1);
              in  shft_in : ns(WLEN);
              out so      : ns(WLEN)) {
           always {
               so = (ctl == SHFT_NIL) ? shft_in :
                   (ctl == SHFT_SHL) ? (ns(WLEN)) (shft_in << 1) :
                   (ctl == SHFT_SHR) ? (ns(WLEN)) (shft_in >> 1) :
                   (ctl == SHFT_ROL) ? (ns(WLEN)) (shft_in # shft_in[WLEN-1]) :
                   (ctl == SHFT_ROR) ? shft_in[0] # ((ns(WLEN-1)) (shft_in >> 1)) :
                   0;
               (ctl == SHFT_SLA) ? (ns(WLEN)) (shft_in << 1) :
               (ctl == SHFT_SRA) ? (ns(WLEN)) (((tc(WLEN)) shft_in) >> 1) :
               0;
               zero = (shft_in == 0);
               cy = (ctl == SHFT_NIL) ? 0 :
                   (ctl == SHFT_SHL) ? shft_in[WLEN-1] :
                   (ctl == SHFT_SHR) ? 0 :
                   (ctl == SHFT_ROL) ? shft_in[WLEN-1] :
                   (ctl == SHFT_ROR) ? shft_in[0] :
                   (ctl == SHFT_SLA) ? shft_in[WLEN-1] :
                   (ctl == SHFT_SRA) ? 0 :
                   0;
           }
       }

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 8)

datapath del processore microprogrammato

```

dp_hmm( in   din   : ns(WLEN); out din_strb : ns(1);
        out  dout  : ns(WLEN); out dout_strb : ns(1)) {
    sig carry, zero : ns(1);
    sig ctl_ot      : ns(1);
    sig ctl_sbus    : ns(4);
    sig ctl_alu     : ns(4);
    sig ctl_shft    : ns(3);
    sig ctl_dest    : ns(4);
    sig rf_out, rf_in : ns(WLEN);
    sig sbus        : ns(WLEN);
    sig alu_in      : ns(WLEN);
    sig alu_out     : ns(WLEN);
    sig shft_in     : ns(WLEN);
    sig shft_out    : ns(WLEN);
    use control(carry, zero, ctl_ot, ctl_sbus, ctl_alu, ctl_shft, ctl_dest);
    use regfile(ctl_dest, ctl_sbus, rf_in, rf_out);
    use alu(ctl_dest, ctl_alu, sbus, alu_in, alu_out);
    use shifter(ctl_shft, zero, carry, shft_in, shft_out);
    always {
        sbus        = (ctl_sbus == SBUS_IN) ? din : rf_out;
        din_strb    = (ctl_sbus == SBUS_IN) ? 1 : 0;
        dout        = sbus;
        dout_strb    = (ctl_ot == O_WR) ? 1 : 0;
        rf_in       = shft_out;
        alu_in       = shft_out;
        shft_in      = alu_out;
    }
}

```

Schaumont, Listing 6.2 - Micro-programmed controller in GEZEL (part 9)

riprogrammazione della microarchitettura

problema: progettare e realizzare un dispositivo di calcolo del delay delle traiettorie di Collatz quale quello della precedente esperienza di laboratorio, però usando un'architettura microprogrammata invece che un circuito ad-hoc

è possibile utilizzare il processore microprogrammato visto sopra, modificando solo il microprogramma?

quasi... occorrono solo due piccole modifiche:

- con dati di I/O a 16 bit, occorrono registri a 32 bit
- per conseguenza, va inserita l'estensione a 32 bit fra la porta di ingresso e il banco dei registri

sono allegate due descrizioni:

- `cpp_hmm_gcd.fdl`: quella presentata nelle pagine precedenti per il calcolo del GCD, arricchita di un modulo di test per la simulazione con *fdlsim*, fornito nella distribuzione Gezel

- `cpp_hmm_delay_collatz.fdl`: la stessa descrizione, eccetto per le suddette modifiche, riprogrammata per il calcolo del delay di traiettorie di Collatz, priva del modulo di test

la seconda descrizione è la base di lancio della proposta di lavoro che segue

come già indicato, questa esperienza ha lo stesso duplice obiettivo della precedente esperienza di laboratorio, però usando un'architettura microprogrammata invece che un circuito ad-hoc; in particolare, occorre:

1. estendere la descrizione `cpp_hmm_delay_collatz.fdl` con un modello FSMD che realizzi l'interfaccia di controllo dell'esecuzione del microprogramma da parte dell'utente, con la stessa specifica funzionale dell'I/O di utente data per il primo obiettivo della precedente esperienza
 anche in questo caso, per il primo obiettivo si richiede la traduzione in VHDL e la simulazione per collaudarne la correttezza; supponendo che il modello esteso sia denominato `cpp_test_hmm_delay_collatz.fdl`, la traduzione in VHDL si effettua con la seguente procedura da linea di comando:


```
cpp -P cpp_test_hmm_delay_collatz.fdl >test_hmm_delay_collatz.fdl
fdlvhd test_hmm_delay_collatz.fdl
```
2. realizzare il modello sulla FPGA della DE1-SoC, come nel caso della precedente esperienza di laboratorio, riusando gli stessi moduli aggiuntivi `bcd` e `rn6appA` e lo stesso modulo top-level per la stessa associazione delle porte di I/O del modello FSMD, tradotto in VHDL, ai pin della FPGA
 si raccomanda di prestare attenzione alla sincronizzazione fra FSM di controllo e microprocessore: con il microprogramma dato, lo strobe di output dura un solo ciclo, mentre invece lo strobe di input sta alto durante la lettura di 0 sulla porta di ingresso, con un fronte di discesa alla lettura di un input non nullo per risalire ad alto subito dopo lo strobe di output

letture raccomandate:

Schaumont, Ch. 6, Sect. 6.5

per ulteriore consultazione:

Zwoliński, Ch. 7, Sect. 7.3-5