

Microprocessor design example in Gezel and VHDL

Tutorial 07 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2017-18

Table of Contents

1. Microprocessor design example in Gezel and VHDL
2. tutorial outline
3. microprocessor architecture and description techniques
4. encoding of microinstruction fields: output, SBUS, ALU
5. encoding of microinstruction fields: Shifter, Dest
6. Next field and microinstruction encoding
7. microprogram and controller datapath (1)
8. controller datapath (2)
9. register file datapath
10. ALU datapath
11. Shifter datapath
12. microprogrammed processor datapath
13. reprogramming the microarchitecture
14. lab experience
15. references

this tutorial deals with:

- architecture of a microprogrammed processor
- microarchitecture description techniques
- encoding of microinstruction fields
- microprogram in the control store
- datapaths of component modules
- datapath of the microprogrammed processor
- testbench datapath and simulation
- lab experience:
 1. reprogramming the microprocessor for a different application: computation of the delay of Collatz trajectories
 2. extending the model with an FSMD to implement the functions considered in the previous lab experience, with FPGA implementation of the extended model

microprocessor architecture and description techniques

let's consider the microarchitecture drawn in figure 6.7, endowed with a microprogram for the GCD computation with Euclid's algorithm and with the small extension of an output port for the computation result

the leftmost, unused bit in the microinstruction format shown in the same figure will be used here to indicate validity of output data

the formal description of the microarchitecture of this dedicated processor exploits the following techniques:

- C preprocessor directives, in particular the `#define` directive, to define the encoding of microinstructions and of their fields
- Gezel description of the control unit, including the microprogram in the control store, and of the datapath as a collection of component modules: register file, ALU, and shifter
- the top-level Gezel module interconnects the control unit and the other component modules, and generates the I/O (strobe) control signals
- the description includes a simple testbench that feeds the processor with a sequence of input pairs and prints the collected results

encoding of microinstruction fields: 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)

encoding of microinstruction fields: 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)

Next field and microinstruction encoding

```

/* 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)

microprogram and controller datapath (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)

controller datapath (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_sbust   = 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)

register file datapath

```

dp regfile ( in  ctl_dest : ns(4);
              in  ctl_sbust : 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_sbust == SBUS_R0) ? r0 :
            (ctl_sbust == SBUS_R1) ? r1 :
            (ctl_sbust == SBUS_R2) ? r2 :
            (ctl_sbust == SBUS_R3) ? r3 :
            (ctl_sbust == SBUS_R4) ? r4 :
            (ctl_sbust == SBUS_R5) ? r5 :
            (ctl_sbust == SBUS_R6) ? r6 :
            (ctl_sbust == SBUS_R7) ? r7 :
            r0;
    }
}

```

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

ALU datapath

```

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)

Shifter datapath

```

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)

microprogrammed processor datapath

```

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)

reprogramming the microarchitecture

problem: design and implement a computational device for the delay of Collatz trajectories like that out of the previous lab experience, but using a microprogrammed architecture rather than an ad-hoc circuit

is it possible to utilize the microprogrammed processor seen above, by only changing the microprogram?

almost... only two minor changes are required:

- with 16-bit I/O data, 32-bit registers are needed
- as a consequence, data extension is needed from the input port to the register file

two descriptions are enclosed:

- `cpp_hmmtest.fdl`: the one presented in the previous pages for GCD computation, endowed with a test module for simulation with `fdlsim`, available from the Gezel distribution
- `cpp_hmm_delay_collatz.fdl`: the same description, except for the aforementioned changes, reprogrammed for the computation of the delay of Collatz trajectories, with no test modules

the second description is the launch base for the forthcoming work proposal

lab experience

as already mentioned, this experience has the same twofold target as the previous lab experience, but using a microprogrammed architecture rather than an ad-hoc circuit; more precisely, the task assignment is:

1. to extend the `cpp_hmm_delay_collatz.fdl` description with an FSM model of the user interface to control the microprogram execution, with the same functional specification of the user I/O as given for the first objective of the previous experience

also in the present case, for the first target, translation to VHDL and simulation are required, the latter as a correctness check; assuming the extended model is named `cpp_test_hmm_delay_collatz.fdl`, its translation to VHDL is obtained by the following command-line procedure:

```
cpp -P cpp_test_hmm_delay_collatz.fdl >test_hmm_delay_collatz.fdl
fdlvhd test_hmm_delay_collatz.fdl
```

2. to implement the model on the DE1-SoC FPGA, just as in the case of the previous lab experience, while reusing the same additional modules `bcd` and `rn6appA`, and the same top-level module for the same port map of I/O ports of the FSM model, translated to VHDL, to I/O pins of the FPGA

it is recommended to pay attention to the synchronization between control FSM and microprocessor; with the given microprogram, the output strobe lasts one cycle only, whereas the input strobe stays high while the processor is reading 0 from the input port, with a falling edge upon reading a nonzero input and a rising edge just after the output strobe

references

recommended readings:

Schaumont, Ch. 6, Sect. 6.5

for further consultation:

Zwolinski (2004) Sect. 7.3-5