Design-ul unui înmulțitor secvențial pentru numere reprezentate în Semn-Marime

Opriţoiu Flavius
flavius.opritoiu@cs.upt.ro

9 decembrie 2023

Introducere

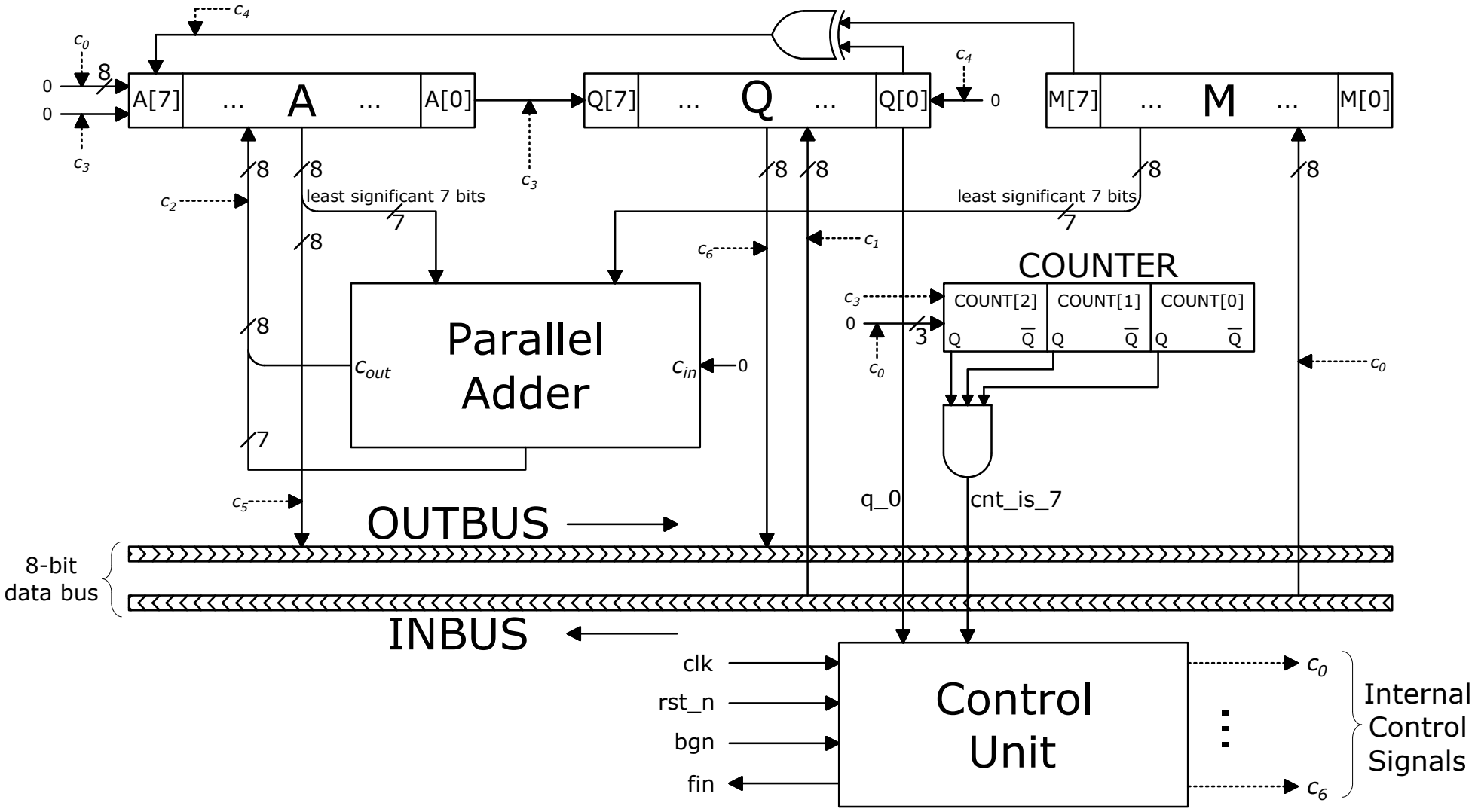
Obiective:

- Construcția unei arhitecturi pentru înmulțirea secvențială a numerelor reprezentate în Semn-Mărime

Caracteristicile arhitecturii:

- Operarea numerelor fracționare pe 8 biți
- Proiectarea componentelor înmulțitorului
- Proiectarea căii de control a arhitecturii

Arhitectura secvențială de înmulțitor Semn-Mărime



Notă: Exceptând `rst_b`, toate celelalte semnale sunt *sincrone*.

Registrul M (rezolvat)

Stochează deînmulțitul, având următoarea interfață:

- `ld_ibus`: încarcă valoarea deînmulțitului din INBUS
  - `ibus`: magistrala INBUS
- `q`: conținutul registrului

```
1  module reg_m(
2    input clk, rst_b, ld_ibus, [7:0] ibus,
3    output reg [7:0] q
4  );
5    always @ (posedge clk, negedge rst_b)
6      if (!rst_b)                    q <= 0;
7      else if (ld_ibus)              q <= ibus;
8  endmodule
```

Registrul Q

Stochează înmulțitorul, având interfața:

- `clr_lsb`: șterge LSB-ul registrului
- `ld_ibus`: încarcă valoarea înmulțitorului din INBUS
  - `ibus`: magistrala INBUS
- `sh_r`: deplasare la dreapta a conținutului registrului
  - `sh_i`: bitul de încărcat în MSB la deplasarea la dreapta
- `ld_obus`: livrează conținutul registrului la OUTBUS
  - `obus`: magistrala OUTBUS
- `q`: conținutul registrului

```
1  module reg_q(
2    input clk, rst_b, clr_lsb, ld_ibus, ld_obus, sh_r,
3    input sh_i, [7:0] ibus,
4    output reg [7:0] obus, [7:0] q
5  );
6    always @(posedge clk, negedge rst_b)
7      //treat inputs rst_b, clr_lsb, ld_ibus, sh_r here

9    always @(*)  //write content to obus when ld_obus==1
10     obus = (ld_obus) ? q : 8'bz;
11 endmodule
```

Registrul A

Acumulator, având interfața:

- `clr`: reset sincron, activ la 1 (șterge conținutul)
- `ld_sum`: încarcă rezultatul sumatorului în acumulator
  - `sum`: rezultatul sumatorului
- `ld_sgn`: încarcă semnul acumulatorului (MSB-ul)
  - `sgn`: semnul acumulatorului
- `ld_obus`: livrează conținutul acumulatorului la OUTBUS
  - `obus`: magistrala OUTBUS
- `sh_r`: deplasare la dreapta a conținutului registrului
  - `sh_i`: bitul de încărcat în MSB la deplasarea la dreapta
- `q`: conținutul acumulatorului

```
1  module reg_a(
2    input clk, rst_b, clr, sh_r, ld_sgn, ld_obus, ld_sum,
3    input sh_i, sgn, [7:0] sum,
4    output reg [7:0] obus, [7:0] q
5  );
6    //implementation here

8    //write content to obus
9  endmodule
```

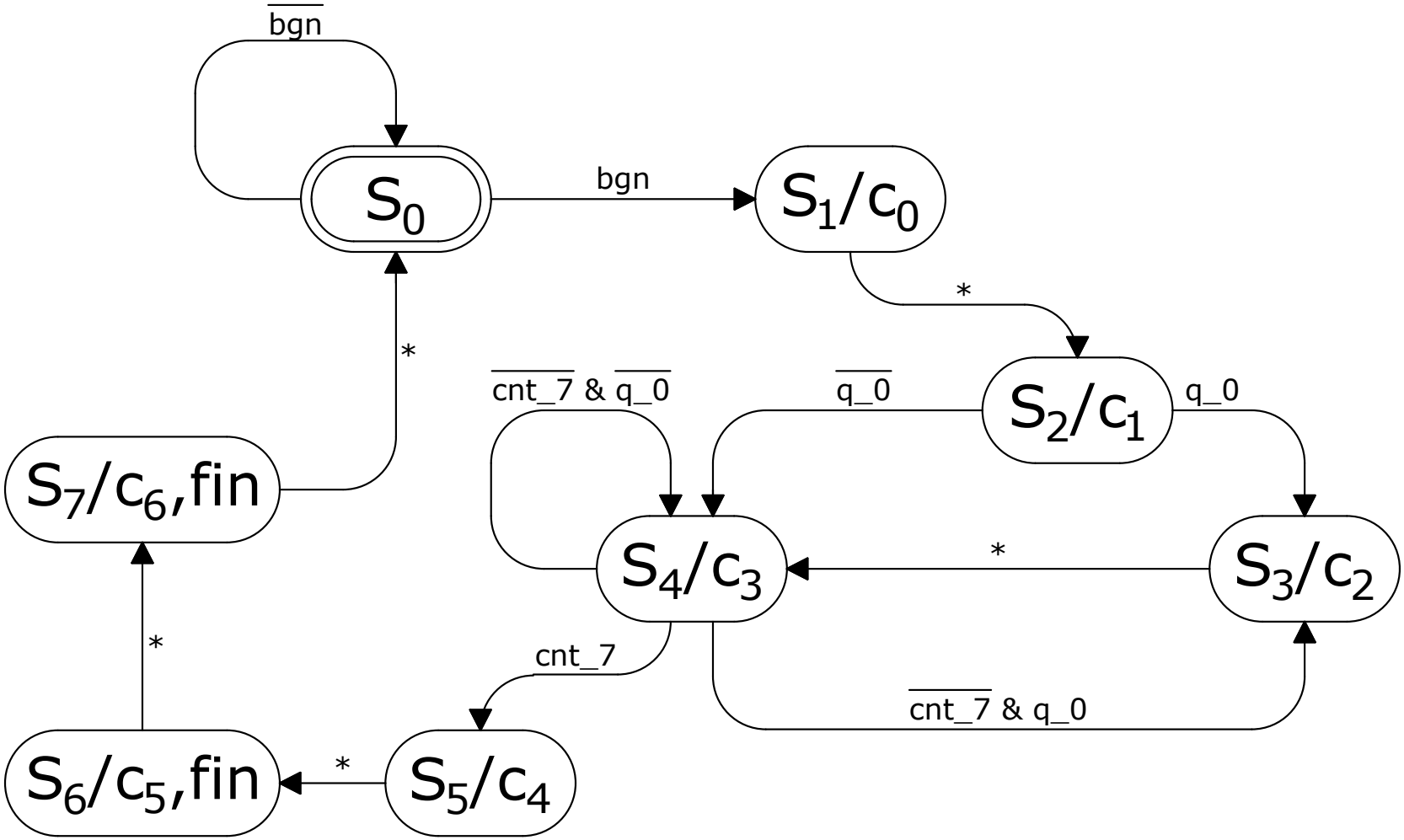
Unitate de Control

Unitatea de control are următoarea interfață:

- `bgn`: demarează algoritmul de înmulțire
- `q_0`: LSB-ul registrului Q
- `cnt_is_7`: valoarea COUNTER este 7
- `c0`: șterge registrul A și COUNTER, încarcă M din INBUS
- `c1`: încarcă Q din INBUS
- `c2`: încarcă rezultatul sumatorului înapoi în A
- `c3`: deplasează A concatenat cu Q la dreapta (încarcă 0 în MSB-ul lui A), incrementează COUNTER
- `c4`: setează semnul lui A, șterge LSB-ul lui Q
- `c5`/`c6`: livrează registrul A/Q la OUTBUS
- `fin`: marchează finalul operației, activat împreună cu c5, c6

```
1 module ctrl_u(
2   input clk, rst_b, bgn, q_0, cnt_is_7,
3   output c0, c1, c2, c3, c4, c5, c6, fin
4 );
5   //implementation here
6 endmodule
```

Unitate de Control (contin.)



Important: Unitatea de control va avea frontul declanșator complementar față de toate celelalte componente secvențiale sincrone din arhitetură!

Integrarea tuturor componentelor

Arhitectura secvențială de înmulțire Semn-Mărime:

```
1   module sm_unit (
2     input clk, rst_b, bgn, [7:0] ibus,
3     output fin, [7:0] obus
4   );
5     //implementation here
6   endmodule

8   module sm_unit_tb;
9       reg clk, rst_b, bgn; reg [7:0] ibus; wire fin; wire [7:0] obus;

11      sm_unit test (.clk(clk), .rst_b(rst_b), .bgn(bgn), .ibus(ibus),
12          .fin(fin), .obus(obus));
13      localparam CLK_PERIOD=100, CLK_CYCLES=17, RST_PULSE=25;
14      localparam X=8'b10010111/*=-23*2^(-7)*/, Y=8'b10000011;/*=-3*2^(-7)*/
15      initial begin clk=1'd0; repeat (CLK_CYCLES*2) #(CLK_PERIOD/2) clk=~clk; end
16      initial begin rst_b=1'd0; #(RST_PULSE); rst_b=1'd1; end
17      initial begin bgn=1'd1; #200; bgn=1'd0; end
18      initial begin ibus=0; #100 ibus=X; #100 ibus=Y; end
19  endmodule
```

Testbench-ul păstrează OUTBUS în impedanță ridicată până la momentul 1300, când, pentru 1 ciclu de tact, OUTBUS devine 8'b00000000, după care, pentru un alt ciclu de clock, OUTBUS devine 8'b10001010 revenind, ulterior, în impedanță ridicată.