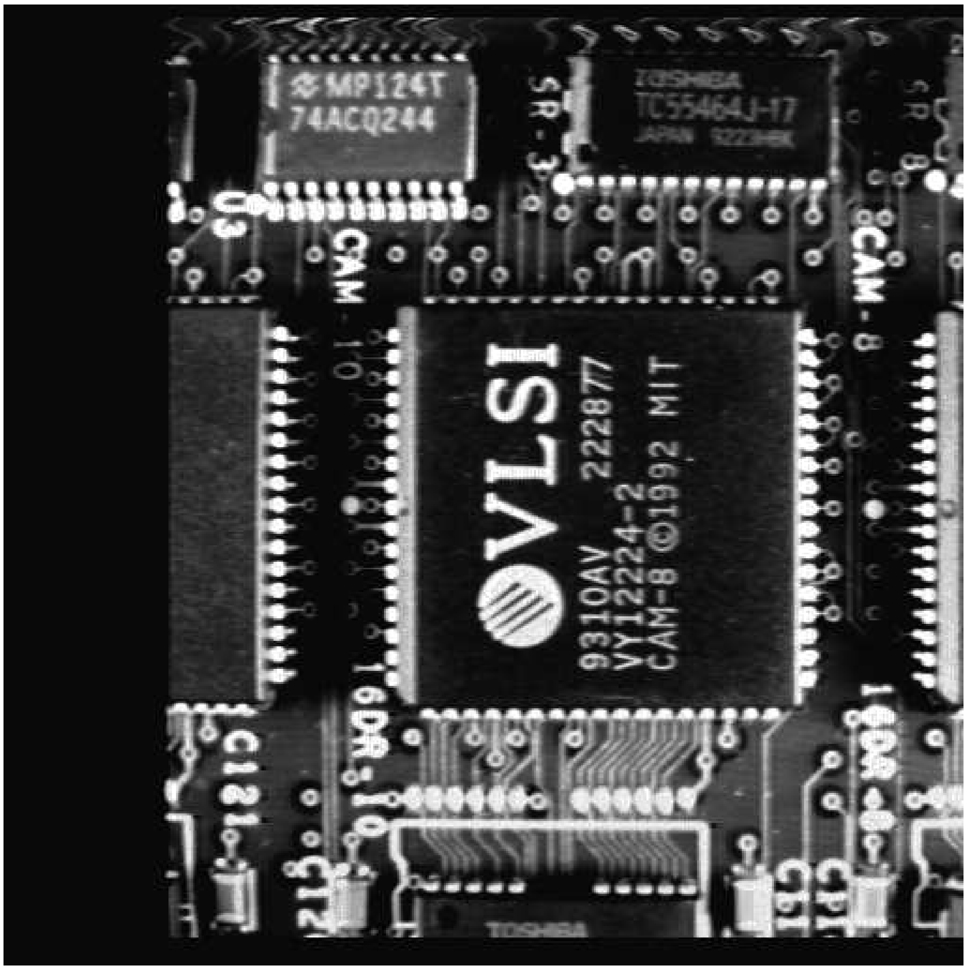
Norman Margolus
Tommaso Toffoli

STEP: a SpaceTime Event Processor

The STEPchip

(version 2.1.12)



MIT Laboratory for Computer Science
Cambridge, MA 02139

Contents

1 The STEPchip

As explained in the *Hardware Reference*, each module contains sixteen copies of the STEPchip—a custom VLSI chip that one may view as embodying the STEP architecture.[1]

[1] In the present version of the STEP system, the STEPchip is but one of the four personalities of a die called the CAM die. The other three personalities of this die are chips used by the STEP controller (documented in *The STEP Controller*). A specific die personality is selected by hardwiring two chip-select lines.

The CAM die is realized on a 1.2-micron gate array (made by VLSI Technology) with two levels of metallization. The STEPchip personality uses approximately 16K gates; the other three personalities, a total of about 9K gates.

A Pinout

The STEPchip is packaged in a 68-pin PLCC with the following pinout

```
pin I/O CAM     PAD TYPE

01 I    CLK     TTL Input
02 I    VSS
03 IO   MPB     CMOS 8ma PU
04 IO   MPA     CMOS 8ma PU
05 IO   DISP    TTL 4ma NP
06 O    DA10    TTL 8ma slew PU
07 O    DOE*    TTL 8ma slew PU
08 O    DCAS*   8ma slew PU
09 IO   DD2     TTL 2ma PU
10 IO   DD3     TTL 2ma PU
11 IO   DD1     TTL 2ma PU
12 I    VDD
13 I    VSS
14 IO   DD0     TTL 2ma PU
15 O    DWE*    TTL 8ma slew PU
16 O    DRAS*   TTL 8ma slew PU
17 O    DA9     TTL 8ma slew PU
18 I    VSS
19 O    DA0     TTL 8ma slew PU
20 O    DA1     TTL 8ma slew PU
21 O    DA2     TTL 8ma slew PU
22 O    DA3     TTL 8ma slew PU
23 I    VSS
24 I    VDD
25 O    DA4     TTL 8ma slew PU
26 O    DA5     TTL 8ma slew PU
27 O    DA6     TTL 8ma slew PU
28 O    DA7     TTL 8ma slew PU
29 O    DA8     TTL 8ma slew PU
30 I    GlueIn5 CMOS 2ma PU     \
31 IO   Glue5   CMOS 4ma PU     / Z+
32 I    GlueIn4 CMOS 2ma PU     \
33 IO   Glue4   CMOS 4ma PU     / Z-
34 I    VSS
35 I    CLR*    TTL Input
36 I    VDD
37 IO   SAD     TTL 4ma PU
38 IO   Glue3   CMOS 4ma PU     \
39 I    GlueIn3 CMOS 2ma PU     / Y+
40 IO   Glue2   CMOS 4ma PU     \
41 I    GlueIn2 CMOS 2ma PU     / Y-
42 IO   Glue1   CMOS 4ma PU     \
43 I    GlueIn1 CMOS 2ma PU     / X+
44 IO   Glue0   CMOS 4ma PU     \
45 I    GlueIn0 CMOS 2ma PU     / X-
46 I    VSS
47 I    VDD
48 I    N_DL_In CMOS 16ma NP
49 O    DA11    TTL 8ma slew PU
50 I    S0      TTL Input PU
51 I    S1      TTL Input PU
52 I    N_IOSEL*CMOS Input NP
53 I    VDD
54 I    VSS
55 I    N_RSIn  CMOS Input NP
56 I    N_LSIn  CMOS 2ma NP
57 I    SAA     TTL 8ma PU
58 I    INT*    TTL 4ma NP Schmitt
59 I    SBA     TTL 8ma PU
60 I    SBD     TTL 4ma PU
61 O    IN_LSOUTTTL 2ma PU
62 I    FlyIn   TTL Input
63 I    N_DR_In CMOS 8ma slew PD
64 O    IOSEL   TTL 8ma PU
65 I    ModSel  TTL 4ma? NP
66 I    Sync    CMOS Input NP
67 O    FlyOut  8ma slew PU
68 I    VDD
```


Figure A.1: External CBus host cable.

B STEPchip schematics

This appendix contains the complete schematics for the STEPchip. The nodes of the schematics hierarchy are called **modules**.[1] Two modules, namely **`weight`** and **`mff`**, define very low-level primitives and have been omitted.

Most modules are in the form of *circuit* schematics; the other modules, having tags '`_p`' or '`_p_mksc`' at the end of their names, are *state-machine* tables.

The hierarchy tree, headed by the `cam` node, expands as follows

```
cam
    busif
        hw_env_reg
        interrupt_reg
            ser_latch
        mpc_reg_a
            serial_in
        mpc_reg_b
            serial_in
        opc_dcd
            bex29
        opc_reg
        reg_len_dcd
            cntr7_dn
            reg_len_p_mksc
        reg_out_mux
            mux29_1
        vwe_reg
    data_path
        dram_cnt_reg
            cntr8_ud_sr
        evc_control
            cntr5_dn
        event_ctr
            cntr8_src
            par_ser
        glue_decode
            glue_cmp
            glue_output_mux
            glue_reg
            glue_xin_mux
            glue_yin_mux
            glue_zin_mux
        glue_pipe
        src_delay
        ssr_decode
            src_mxr
    dram_if
        addr_pipe
        dram_ctrl
            cntr5_dn
            cntr8_dn
            dram_ctrl1_p_mksc
        nibble_fifo
        nibble_pipe
        scan
            scan1_p_mksc
        scan_dly
            est_dly
    node
        nodeio
            node_data
                mfct_sync
            node_iosel
            node_rwfsm_t
                node_rwfsm_p
            node_status
                mfct_sync
                mff_sync_init
            nodefsm_p
    scan_gen
        dim_reg
            mxr24_1b
        kick_reg
            bex30
            kick_cy_h
            kick_cy_l
        offset_reg
            par_ser
        run_reg
            cntr6_dn
            sc_latch
            sync_dcd
        scan_reg
            bex35
            mux24_1
        site_gen
            cntr8_sr
            mxr24_1
            mxr24_1a
    sram_if
        lutmac_reg
            cntr8_sr
```

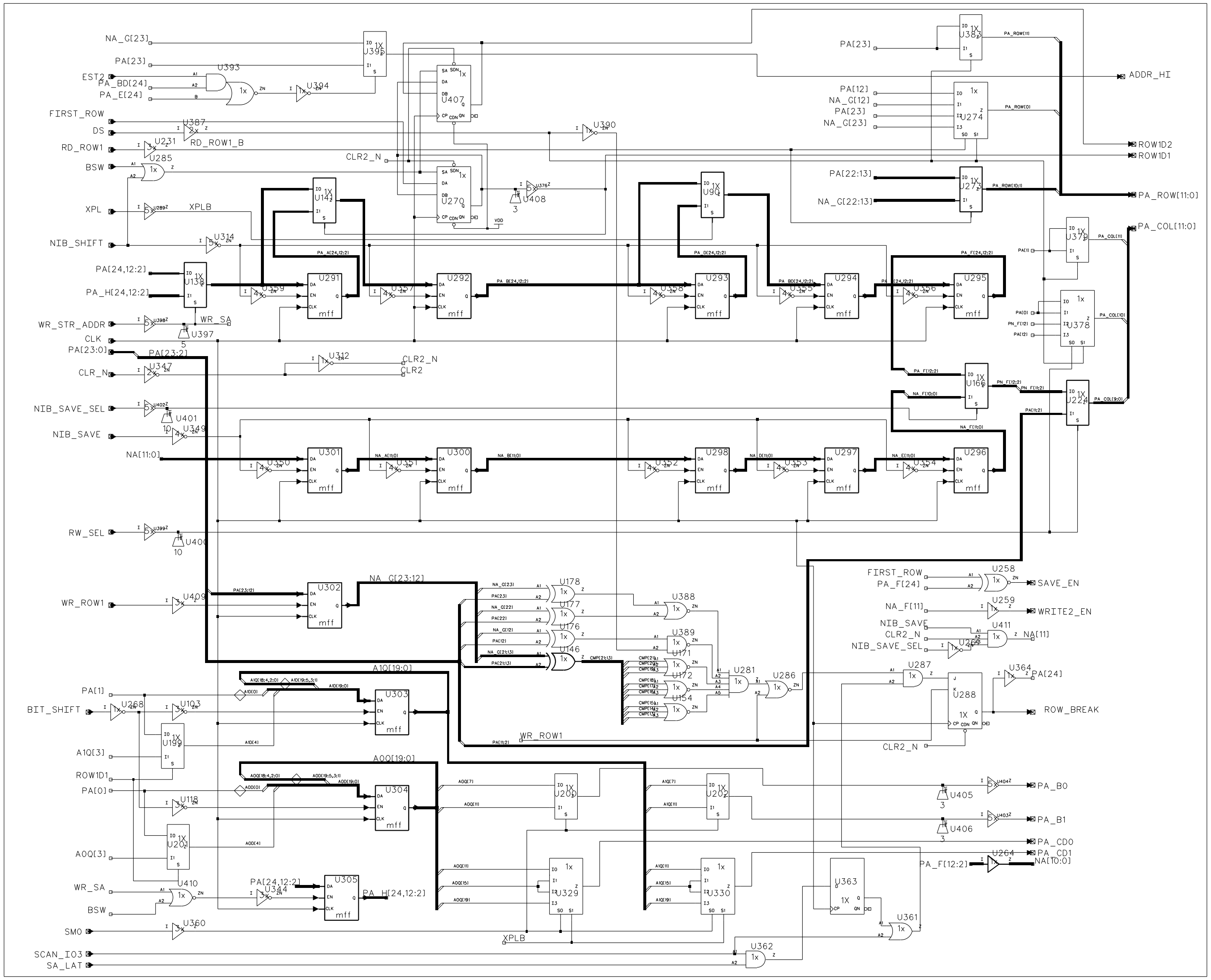
[1] These are Valid design modules, not to be confused with the physical modules that make up a STEP system.

B.1 List of STEPchip schematics

The modules documented in the following pages appear in the order listed below. Circuit schematics are given as PostScript files with extension `ps`. State-machine tables are given as text files with extension `sm`.

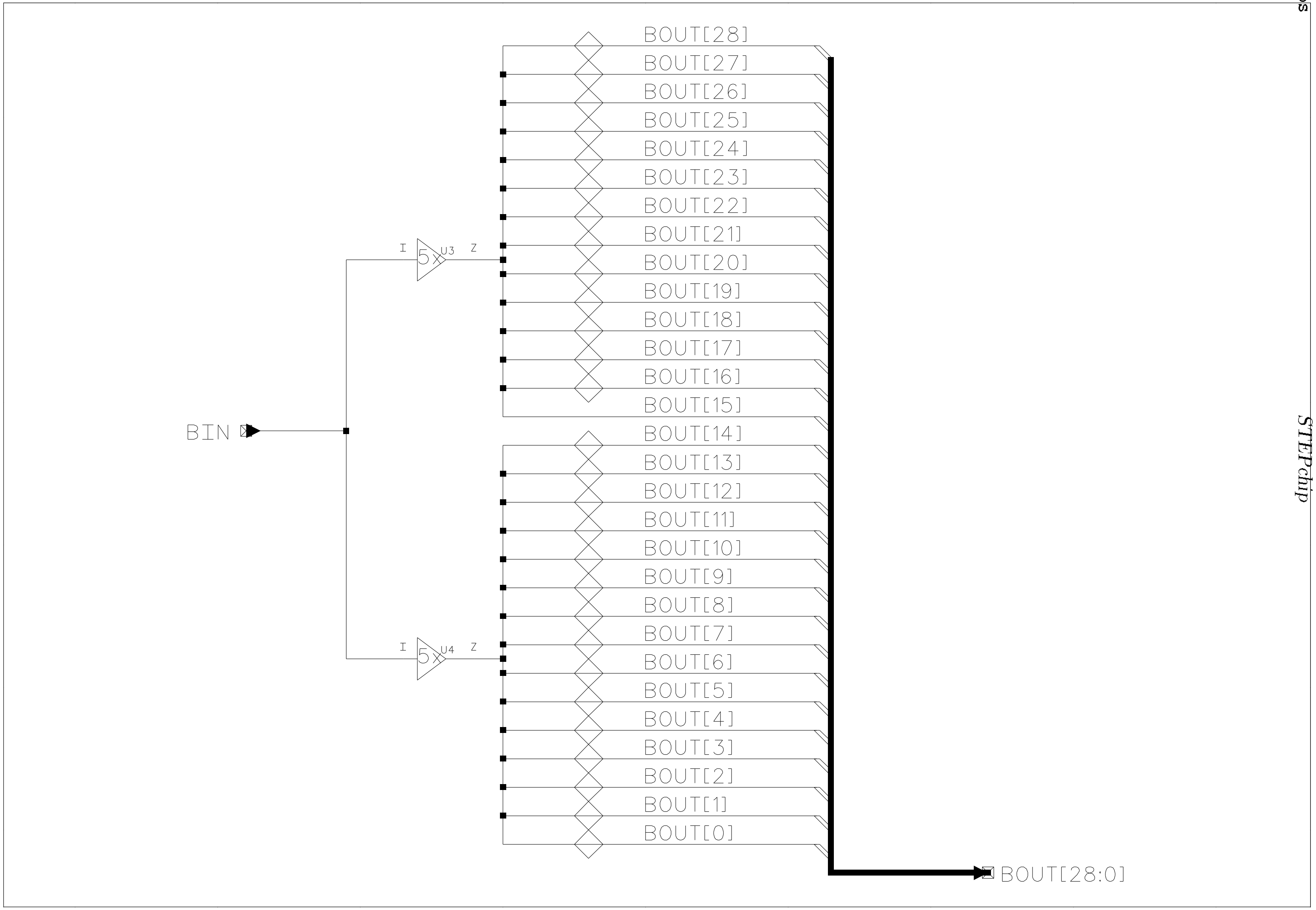
Thus, for example, module `cam` is documented by the schematics `cam.ps`, while module `nodefsm_p` is documented by the state machine `nodefsm.sm` (the tag '`_p`' appears in the module name but not in the file name).

```
addr_pipe.ps
bex29.ps
bex30.ps
bex35.ps
busif.ps
cam.ps
cntr5_dn.ps
cntr6_dn.ps
cntr7_dn.ps
cntr8_dn.ps
cntr8_sr.ps
cntr8_src.ps
cntr8_ud_sr.ps
data_path.ps
dim_reg.ps
dram_cnt_reg.ps
dram_ctrl.ps
dram_ctrl1.sm
dram_if.ps
est_dly.ps
evc_control.ps
event_ctr.ps
glue_cmp.ps
glue_decode.ps
glue_output_mux.ps
glue_pipe.ps
glue_reg.ps
glue_xin_mux.ps
glue_yin_mux.ps
glue_zin_mux.ps
hw_env_reg.ps
interrupt_reg.ps
kick_cy_h.ps
kick_cy_l.ps
kick_reg.ps
lutmac_reg.ps
mfct_sync.ps
mff_sync_init.ps
mpc_reg_a.ps
mpc_reg_b.ps
mux24_1.ps
mux29_1.ps
mxr24_1.ps
mxr24_1a.ps
mxr24_1b.ps
nibble_fifo.ps
nibble_pipe.ps
node.ps
node_data.ps
node_iosel.ps
node_rwfsm.sm
node_rwfsm_t.ps
node_status.ps
nodefsm.sm
nodeio.ps
offset_reg.ps
opc_dcd.ps
opc_reg.ps
par_ser.ps
reg_len_dcd.ps
reg_len.sm
reg_out_mux.ps
run_reg.ps
sc_latch.ps
scan.ps
scan1.sm
scan_dly.ps
scan_gen.ps
scan_reg.ps
ser_latch.ps
serial_in.ps
site_gen.ps
sram_if.ps
src_delay.ps
src_mxr.ps
ssr_decode.ps
sync_dcd.ps
vwe_reg.ps
```



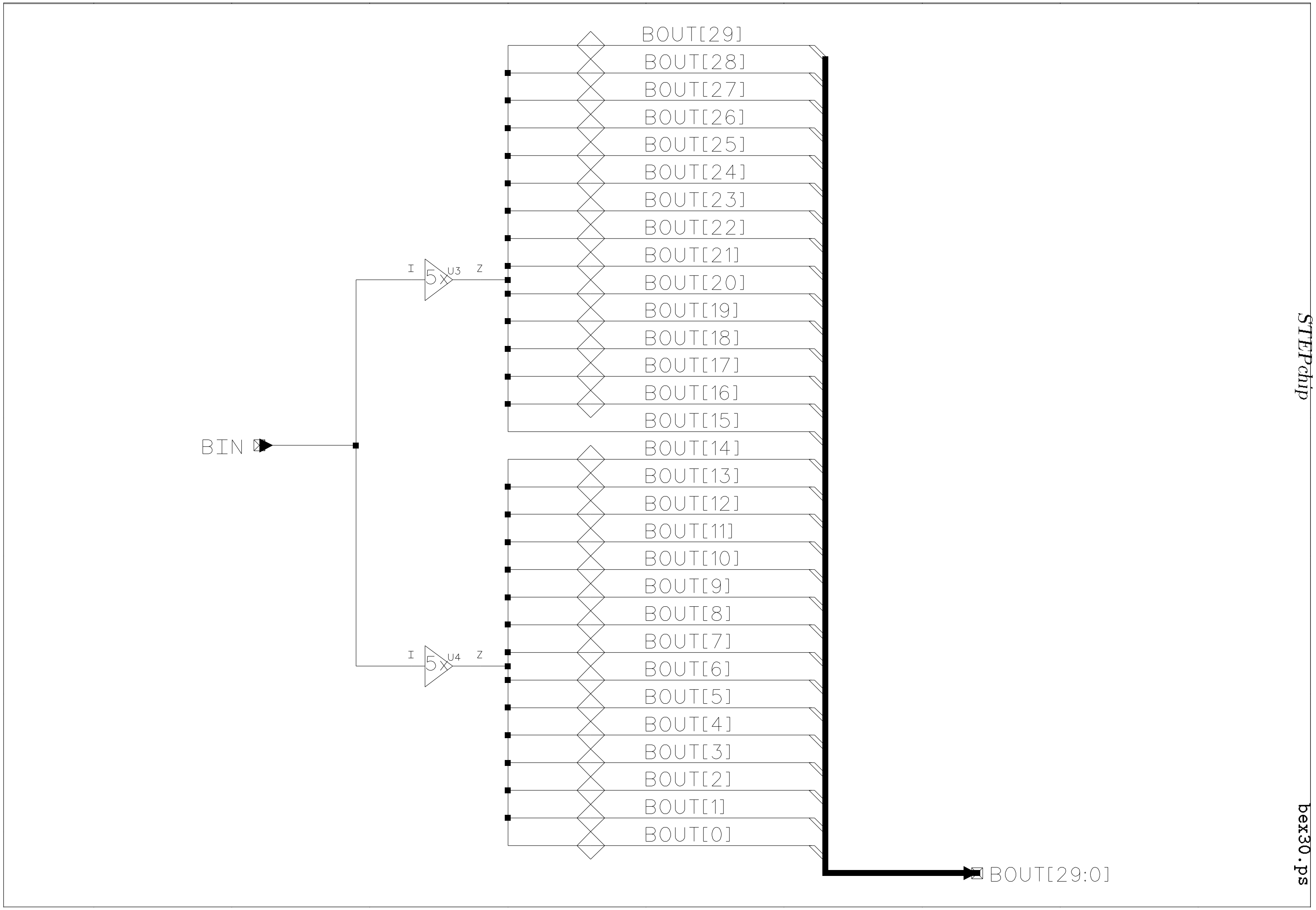
STEPchip

addr_pipe.ps

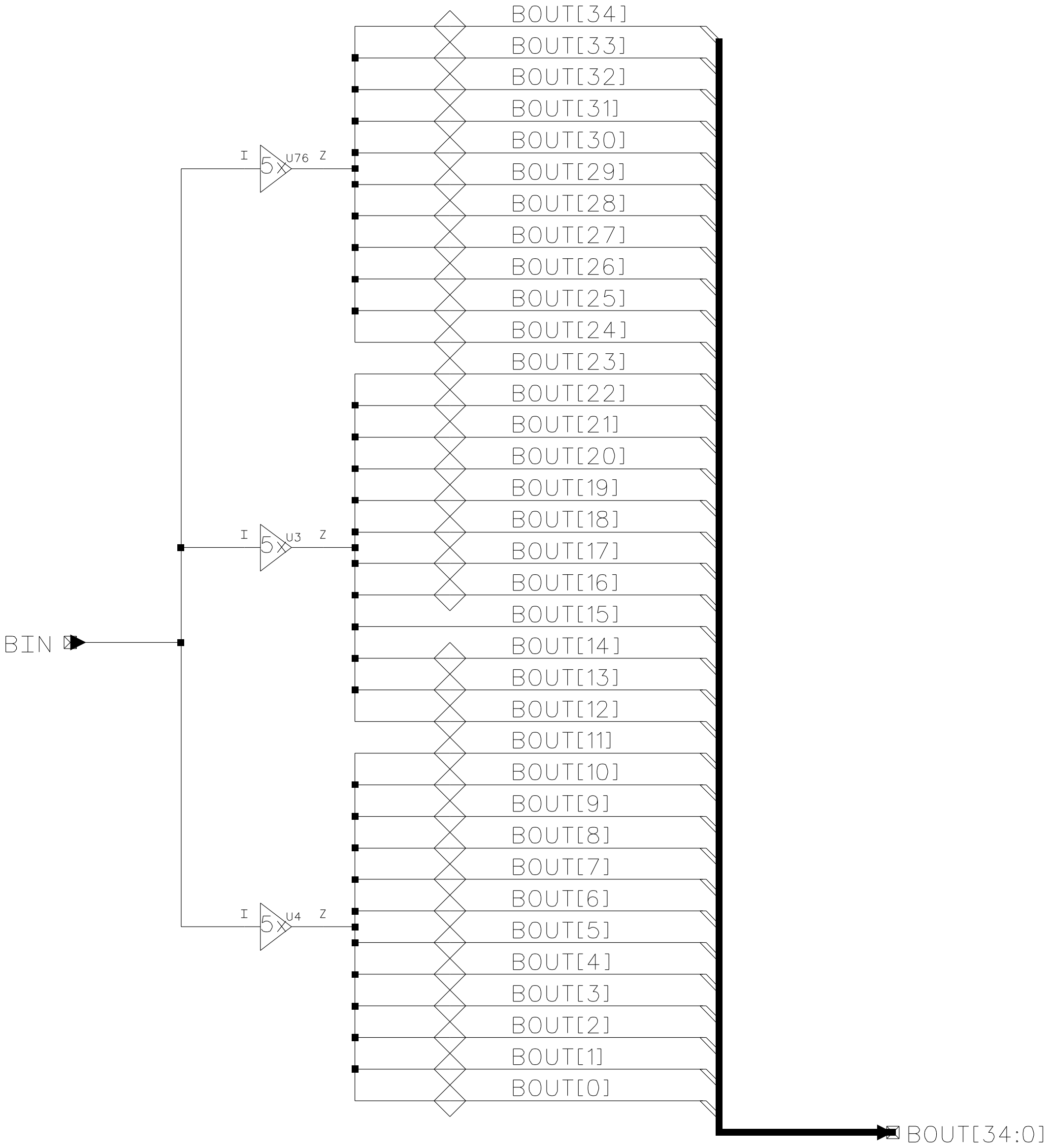

BIN
I
U3
5x
Z
I
U4
5x
Z
BOUT[28]
BOUT[27]
BOUT[26]
BOUT[25]
BOUT[24]
BOUT[23]
BOUT[22]
BOUT[21]
BOUT[20]
BOUT[19]
BOUT[18]
BOUT[17]
BOUT[16]
BOUT[15]
BOUT[14]
BOUT[13]
BOUT[12]
BOUT[11]
BOUT[10]
BOUT[9]
BOUT[8]
BOUT[7]
BOUT[6]
BOUT[5]
BOUT[4]
BOUT[3]
BOUT[2]
BOUT[1]
BOUT[0]
BOUT[28:0]

STEPchip

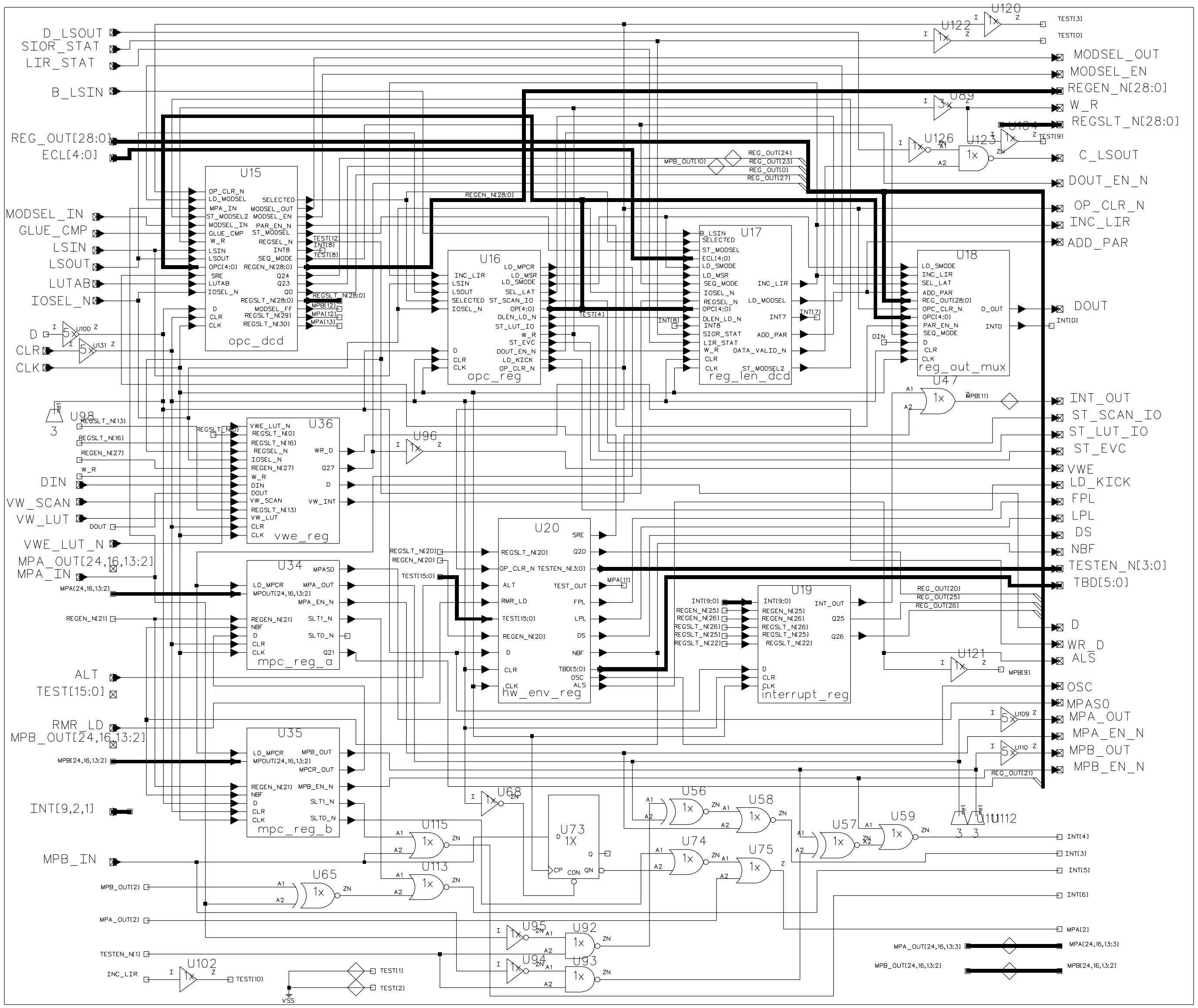
[la]bex30 by nhm on 15-Feb-93 at 4:19 P.M.



[la]bex35 by nhm on 15-Feb-93 at 4:19 P.M.

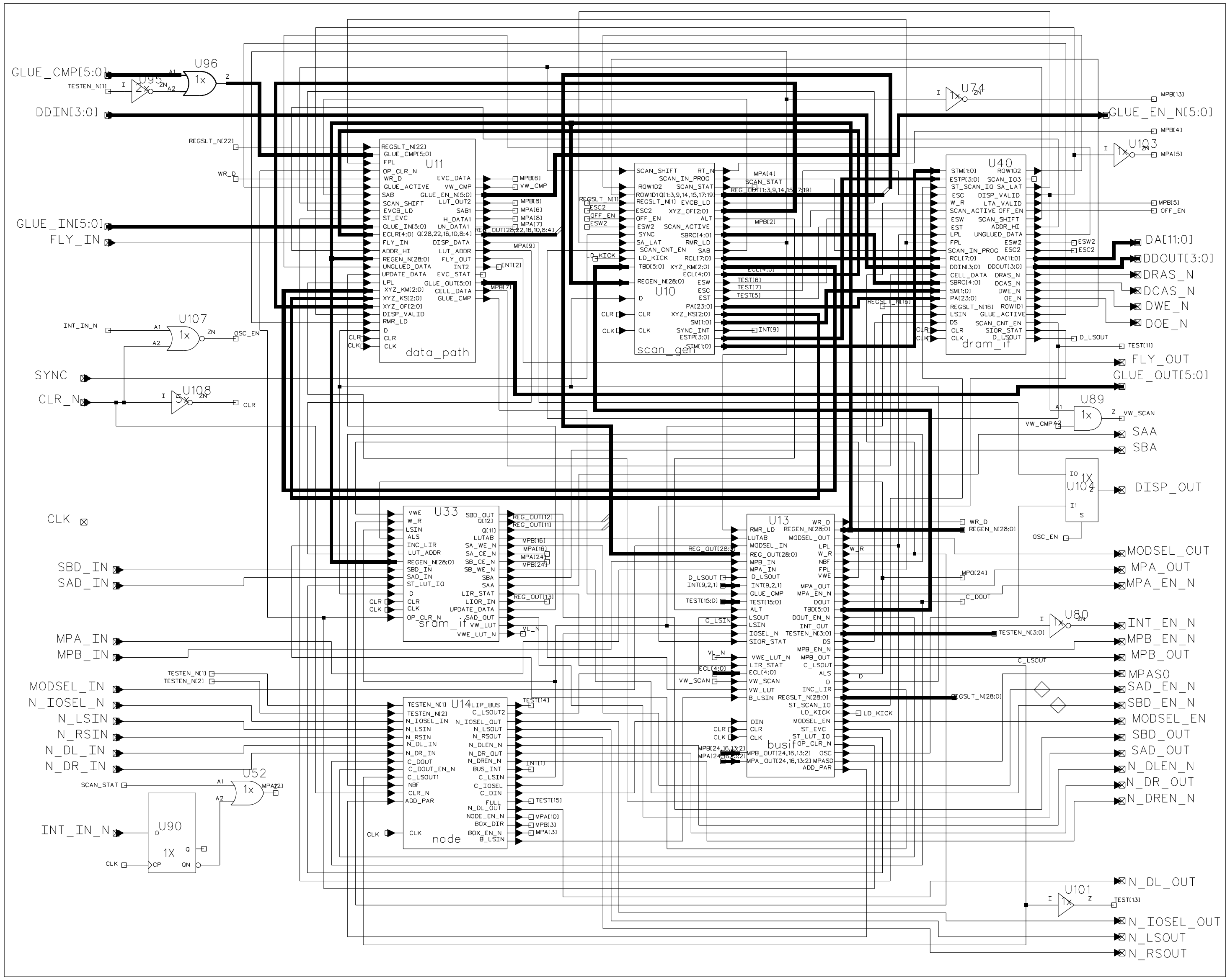


STEPchip

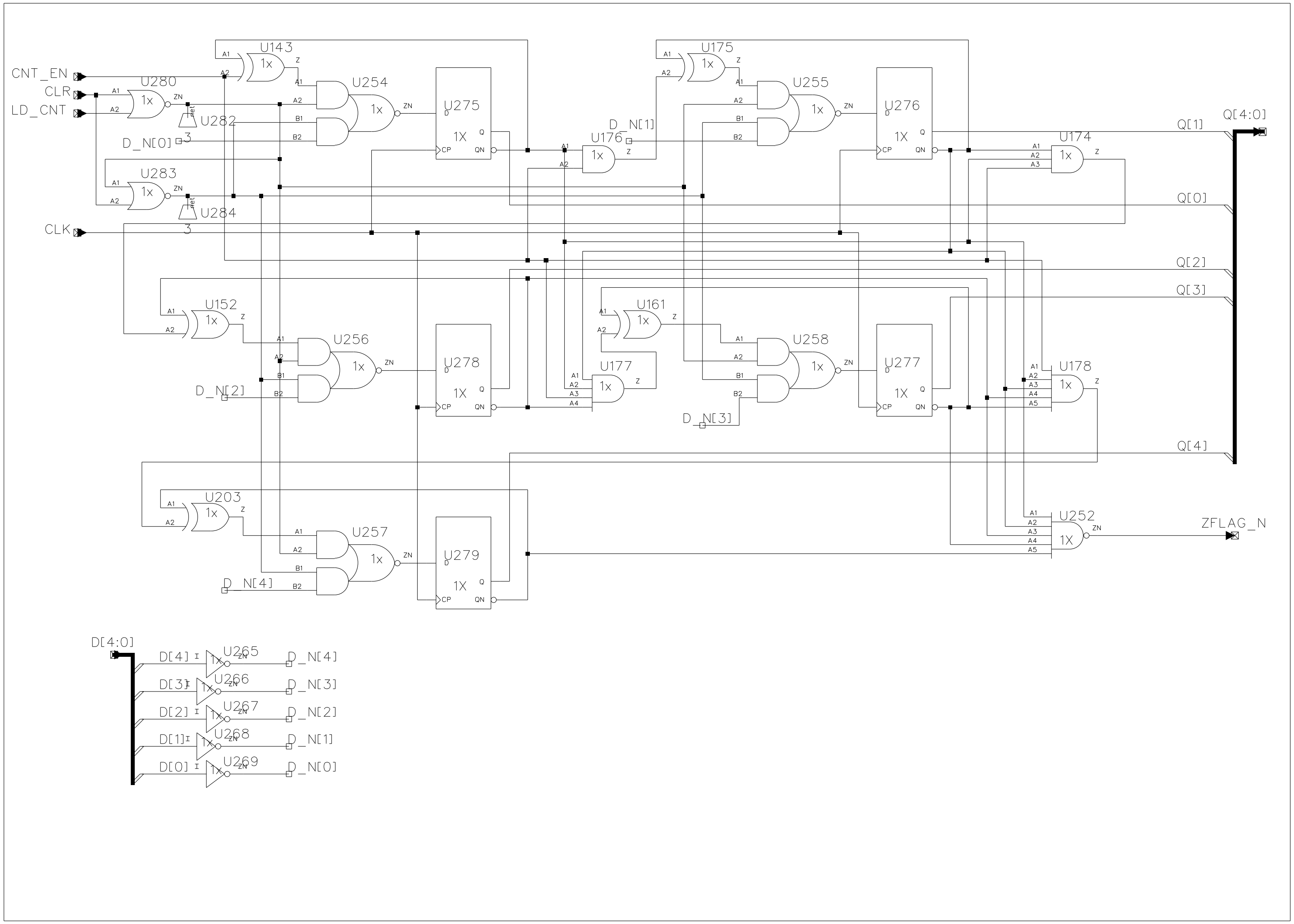


STEPchip

busif.ps


GLUE_CMP[5:0]
DDIN[3:0]
GLUE_IN[5:0]
FLY_IN
SYNC
CLR_N
CLK
SBD_IN
SAD_IN
MPA_IN
MPB_IN
MODSEL_IN
N_IOSEL_N
N_LSIN
N_RSIN
N_DL_IN
N_DR_IN
INT_IN_N
U96
U95
U107
U108
U11
data_path
U10
scan_gen
U40
dram_if
U33
sram_if
U14
node
U13
busif
U52
U90
U74
U103
U89
U104
U80
U101
GLUE_EN_N[5:0]
DA[11:0]
DDOUT[3:0]
DRAS_N
DCAS_N
DWE_N
DOE_N
FLY_OUT
GLUE_OUT[5:0]
SAA
SBA
DISP_OUT
MODSEL_OUT
MPA_OUT
MPA_EN_N
INT_EN_N
MPB_EN_N
MPB_OUT
MPASO
SAD_EN_N
SBD_EN_N
MODSEL_EN
SBD_OUT
SAD_OUT
N_DLEN_N
N_DR_OUT
N_DREN_N
N_DL_OUT
N_IOSEL_OUT
N_LSOUT
N_RSOUT

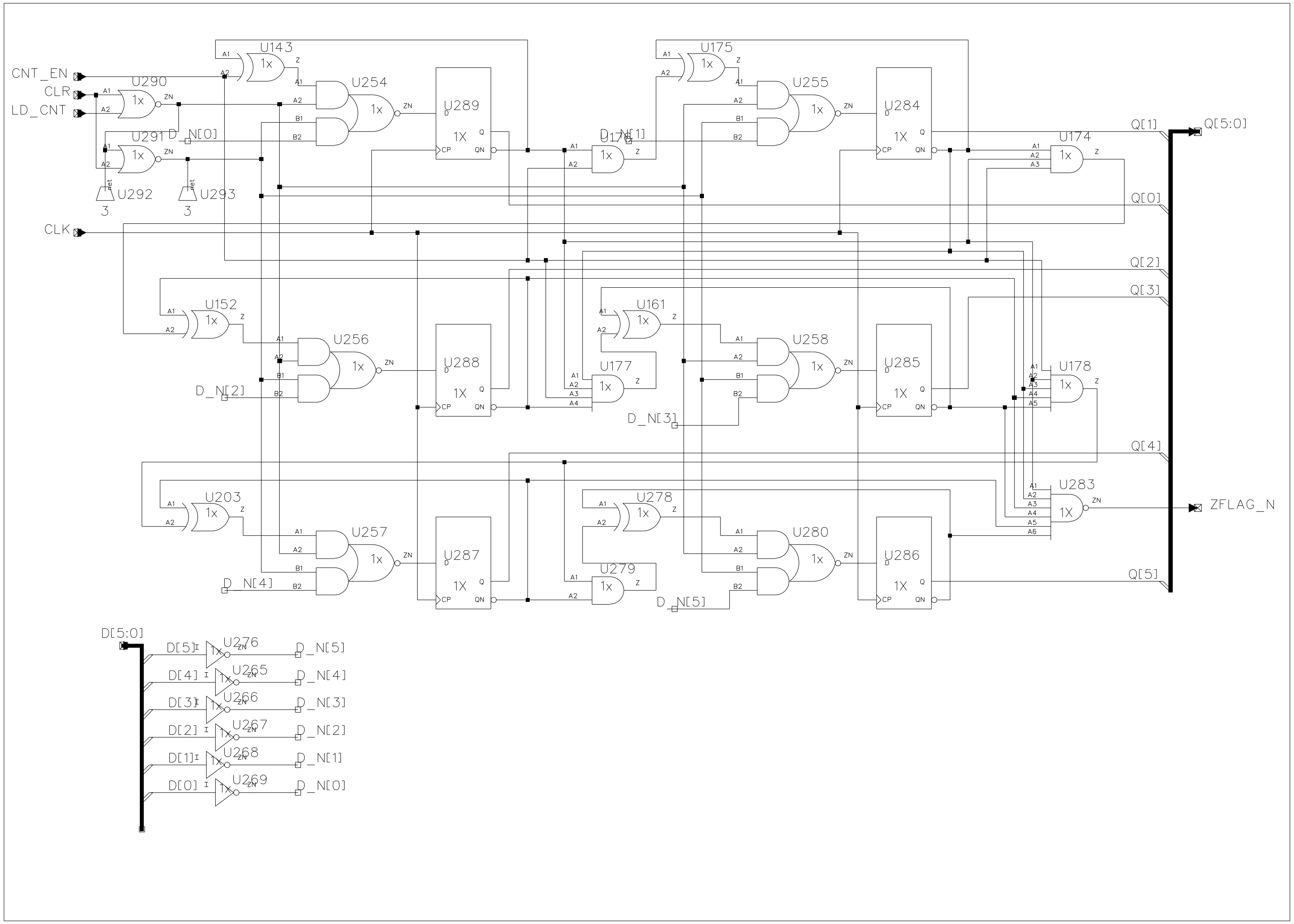
STEPchip



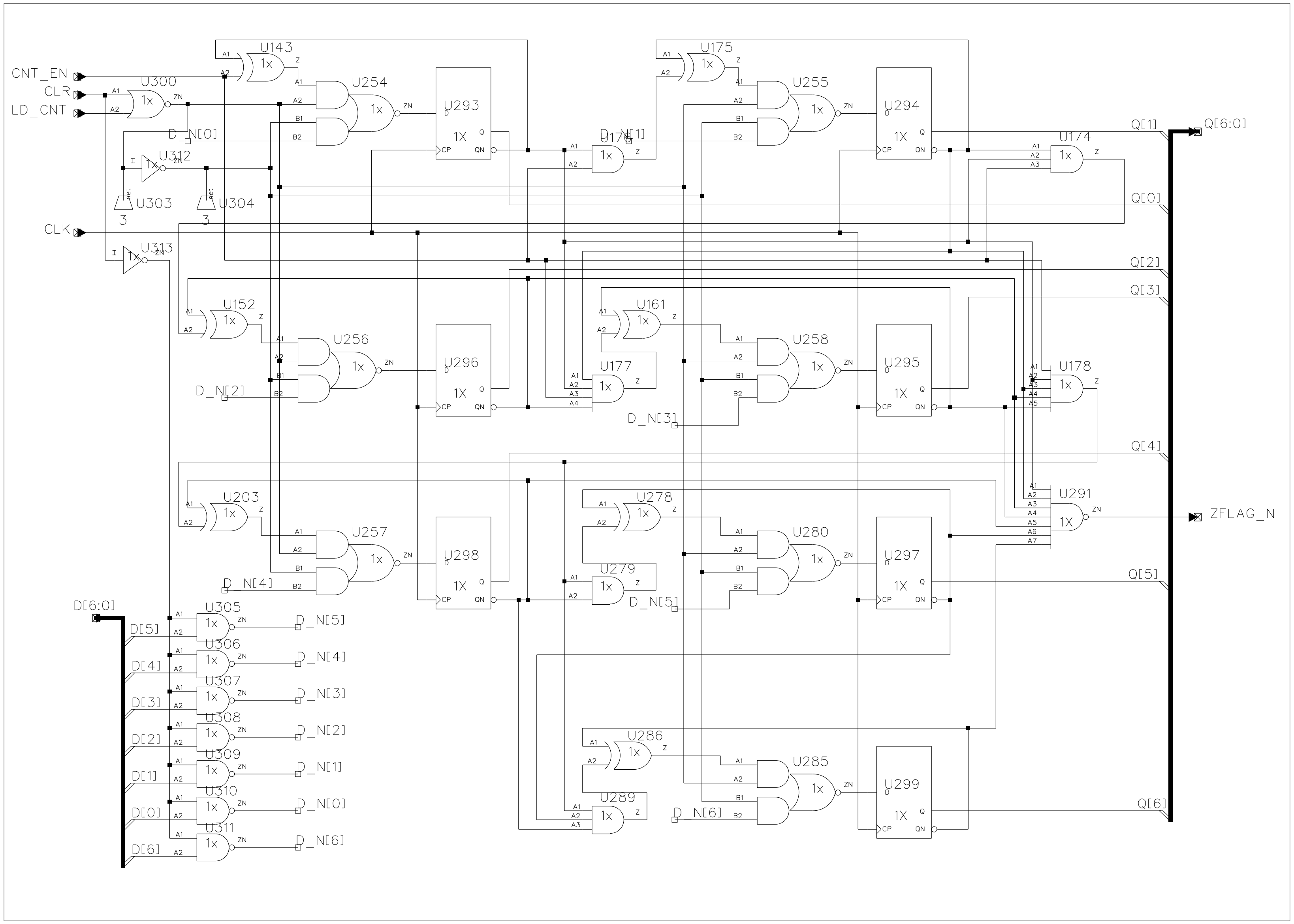
STEPchip

cntr5_dn.ps

[la]cntr6_dn by nhm on 15-Feb-93 at 4:23 P.M.



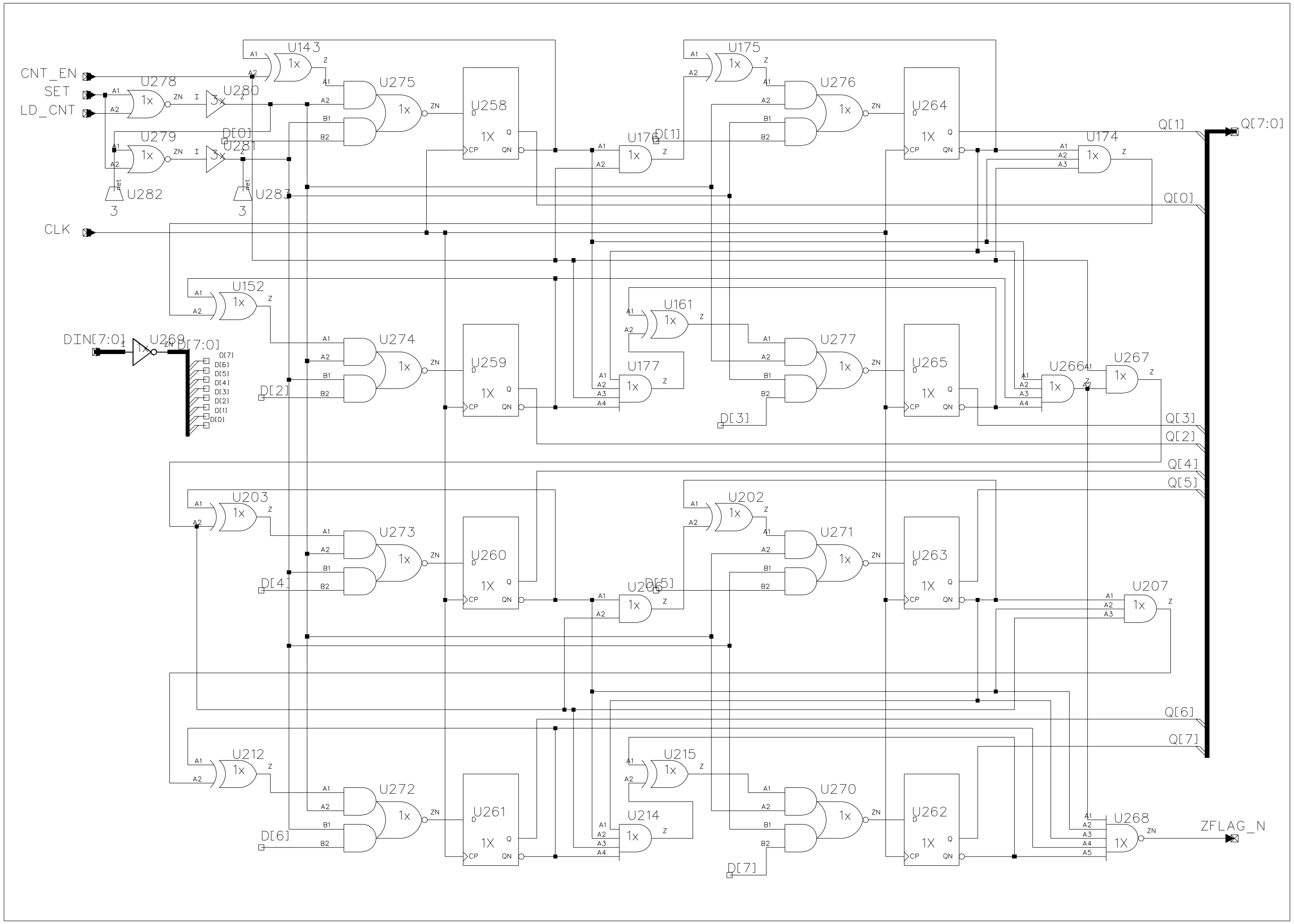
[la]cntr7_dn by nhm on 15-Feb-93 at 4:23 P.M.



STEPchip

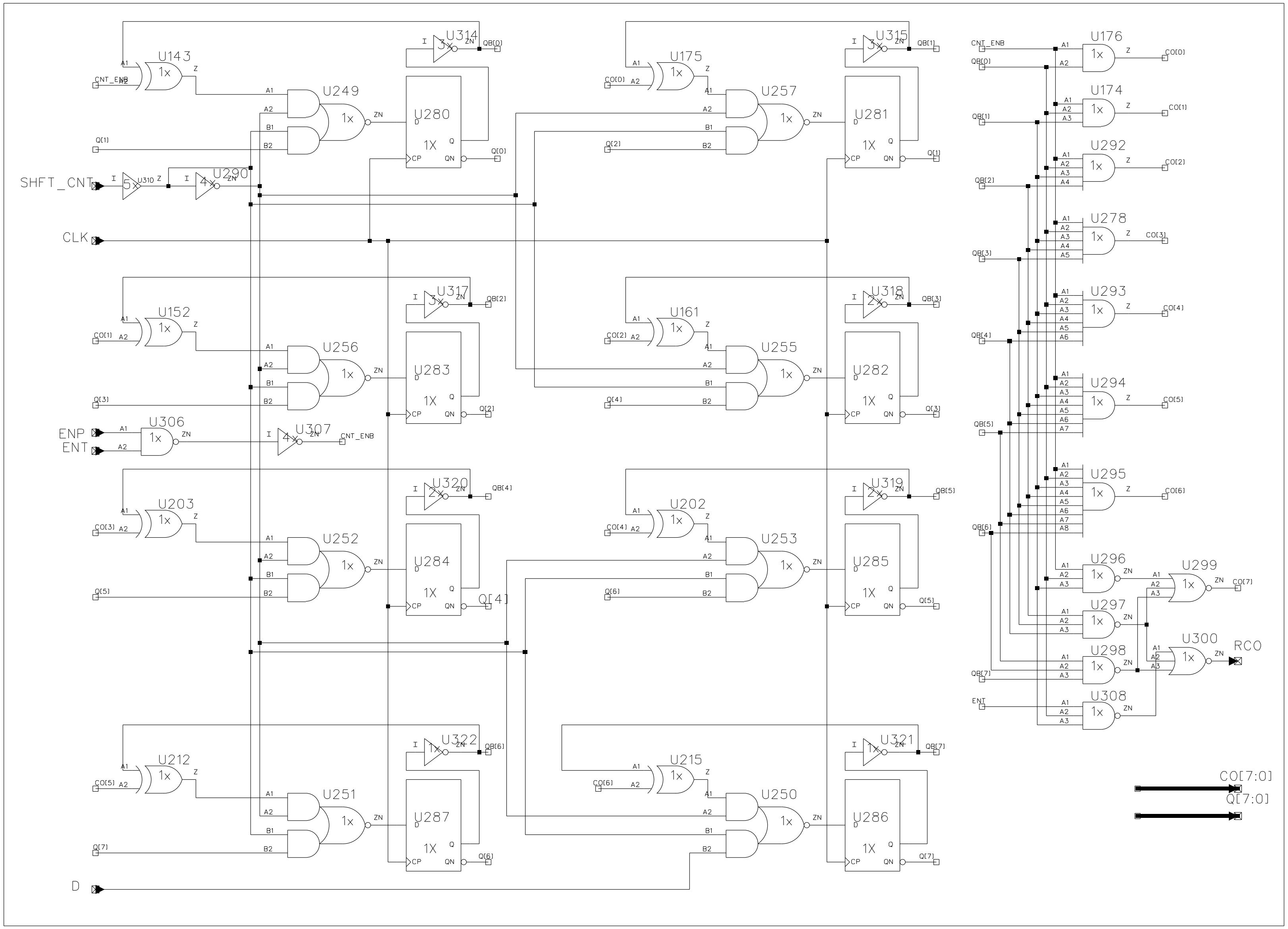
cntr7_dn.ps

[la]cntr8_dn by nhm on 15-Feb-93 at 4:23 P.M.



STEPchip

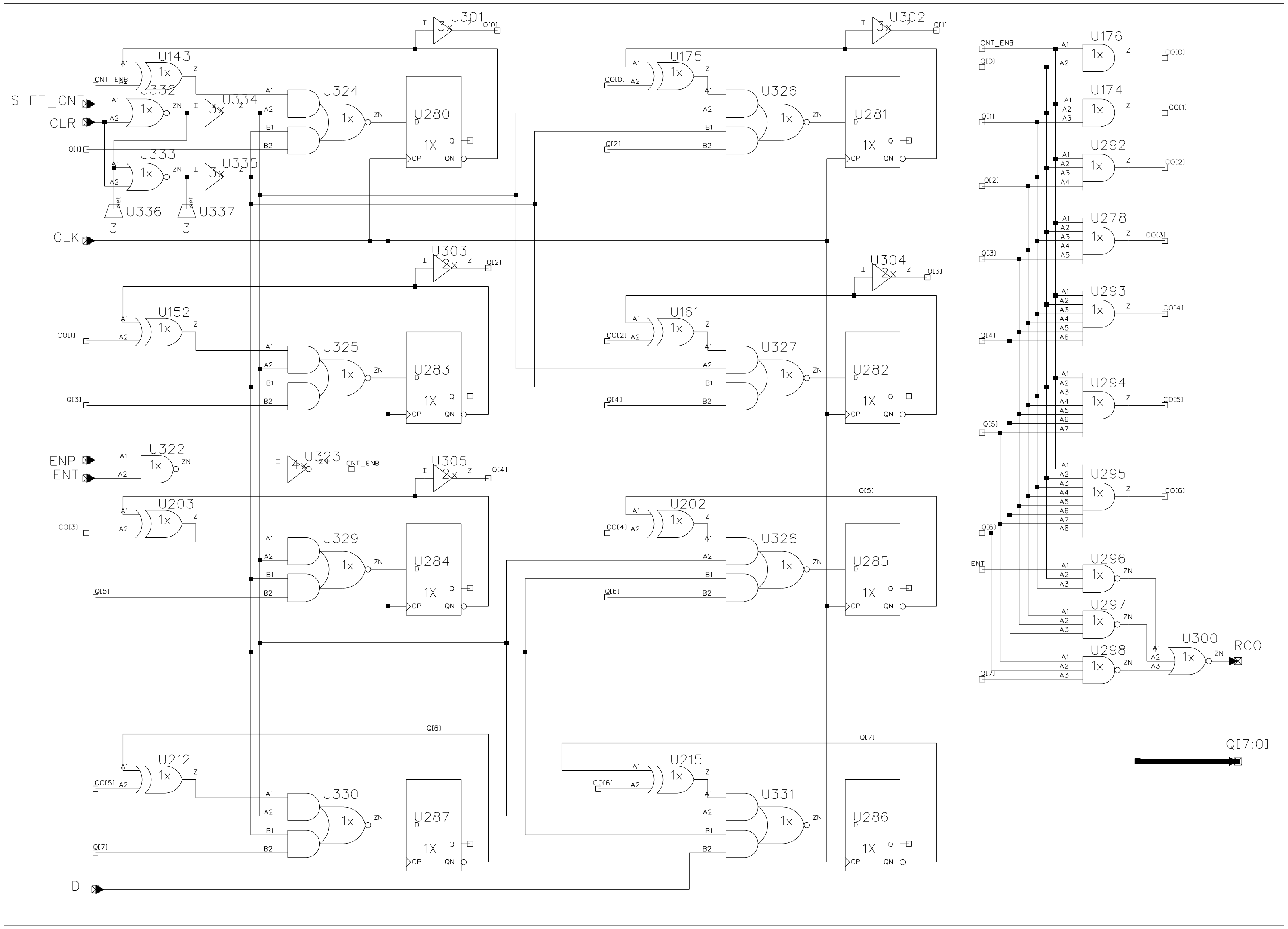
[la]cntr8_sr by nhm on 15-Feb-93 at 4:23 P.M.



STEPchip

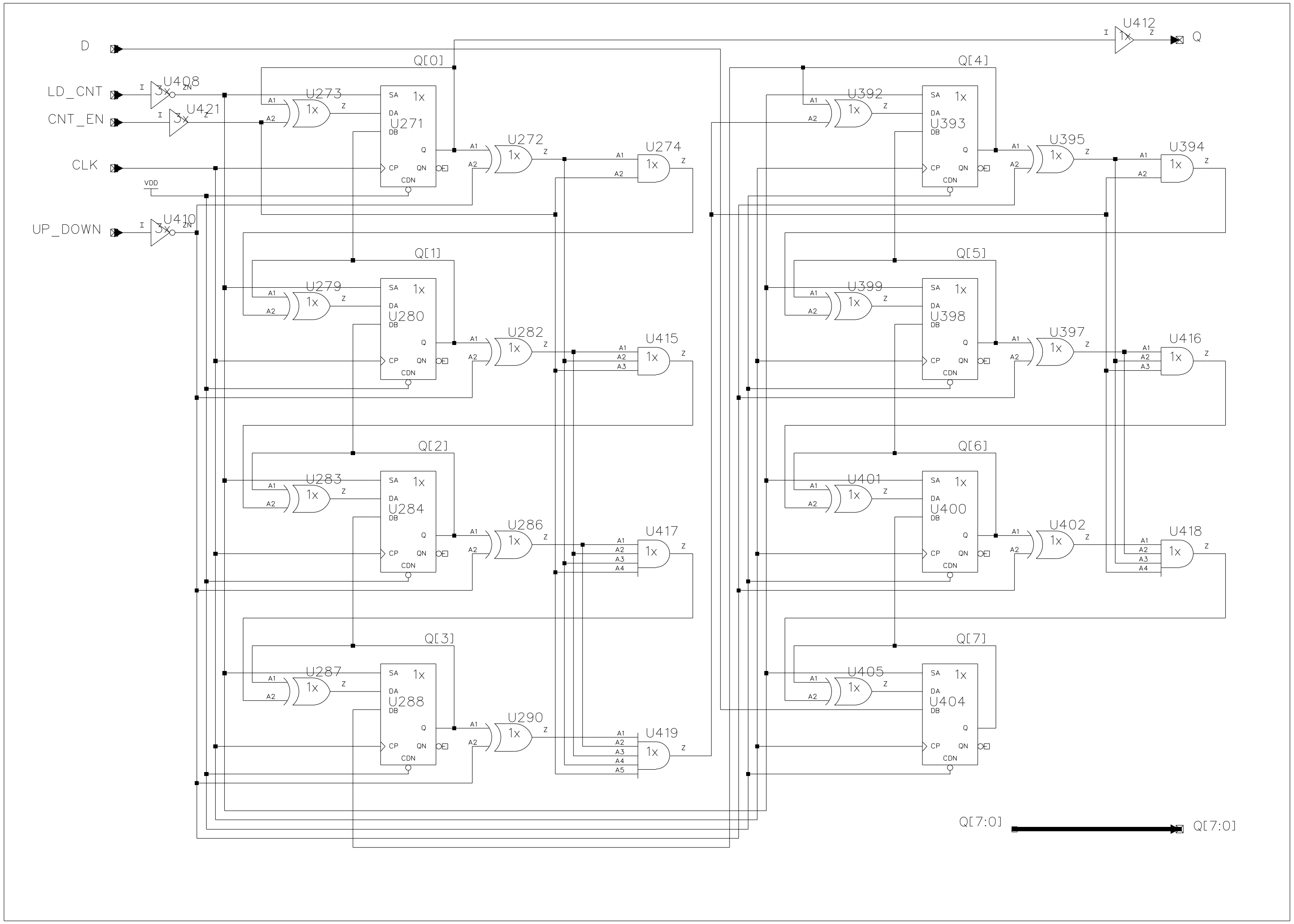
cntr8_sr.ps

[la]cntr8_src by nhm on 15-Feb-93 at 4:23 P.M.



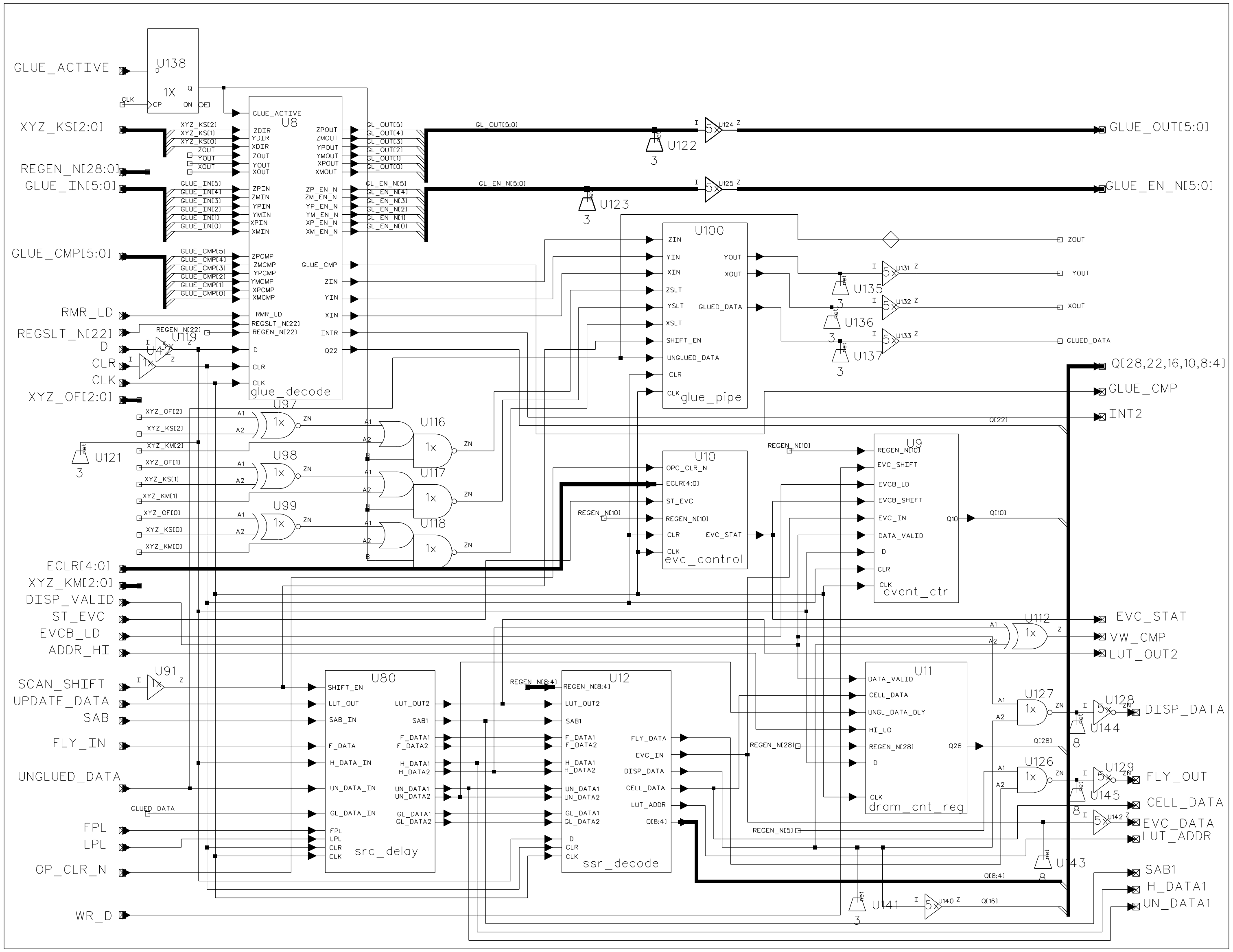
STEPchip

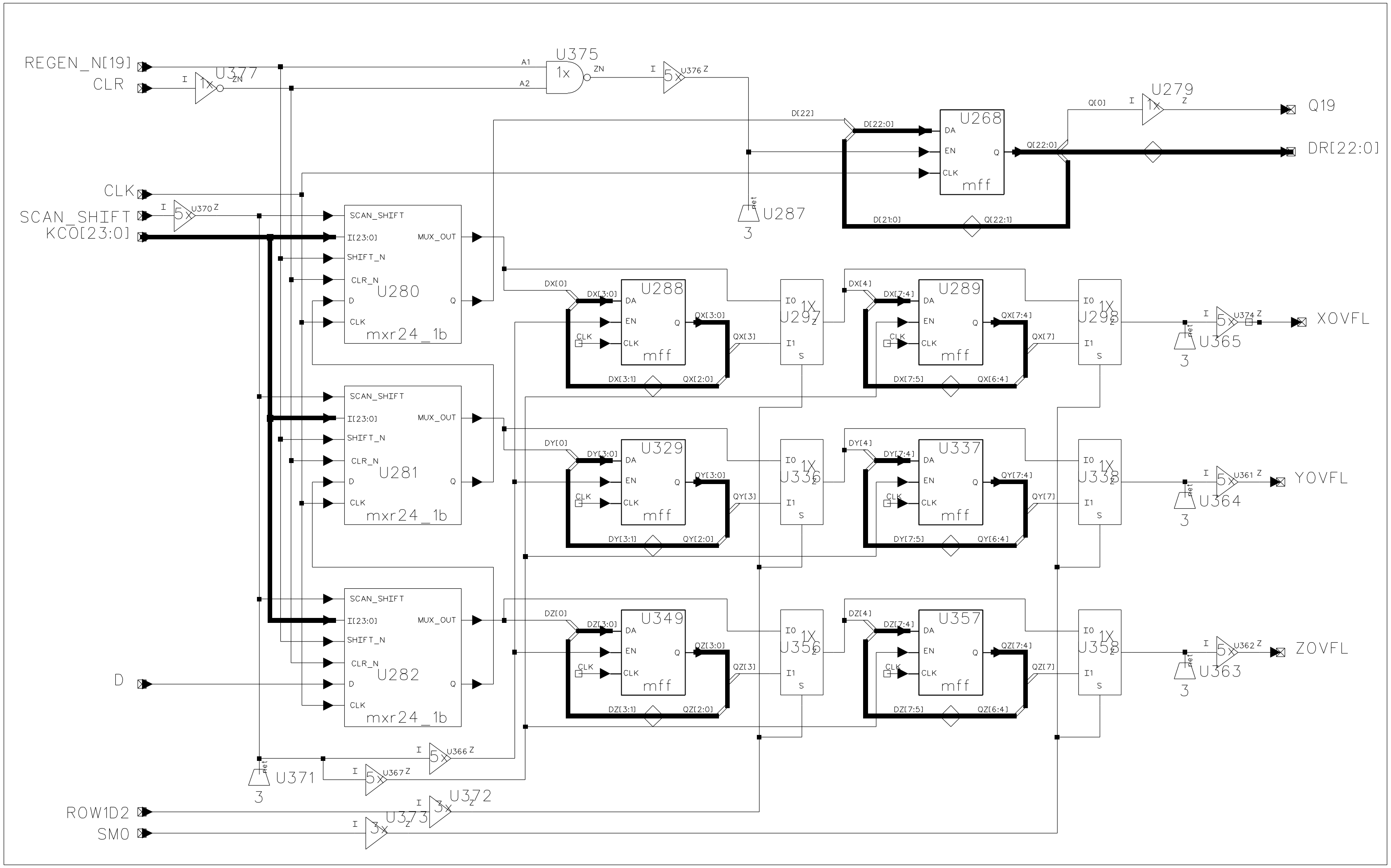
[la]cntr8_ud_sr by nhm on 15-Feb-93 at 4:23 P.M.



STEPchip

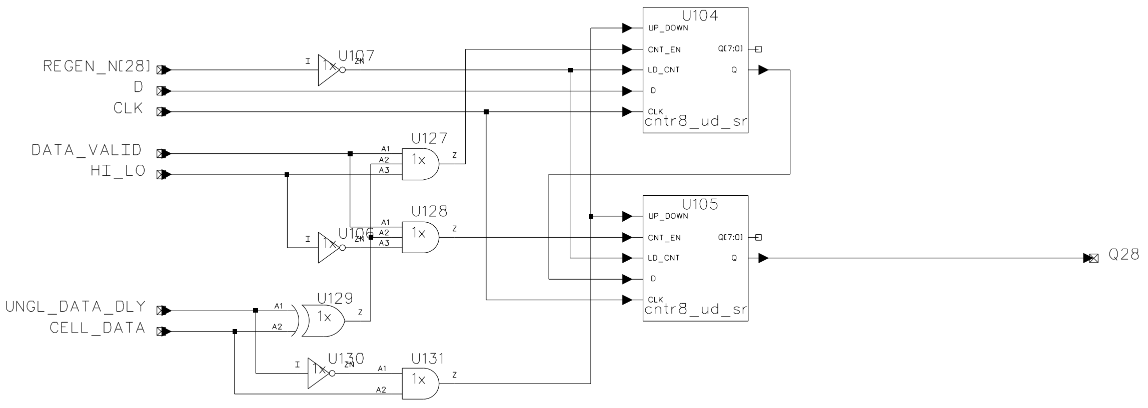
cntr8_ud_sr.ps

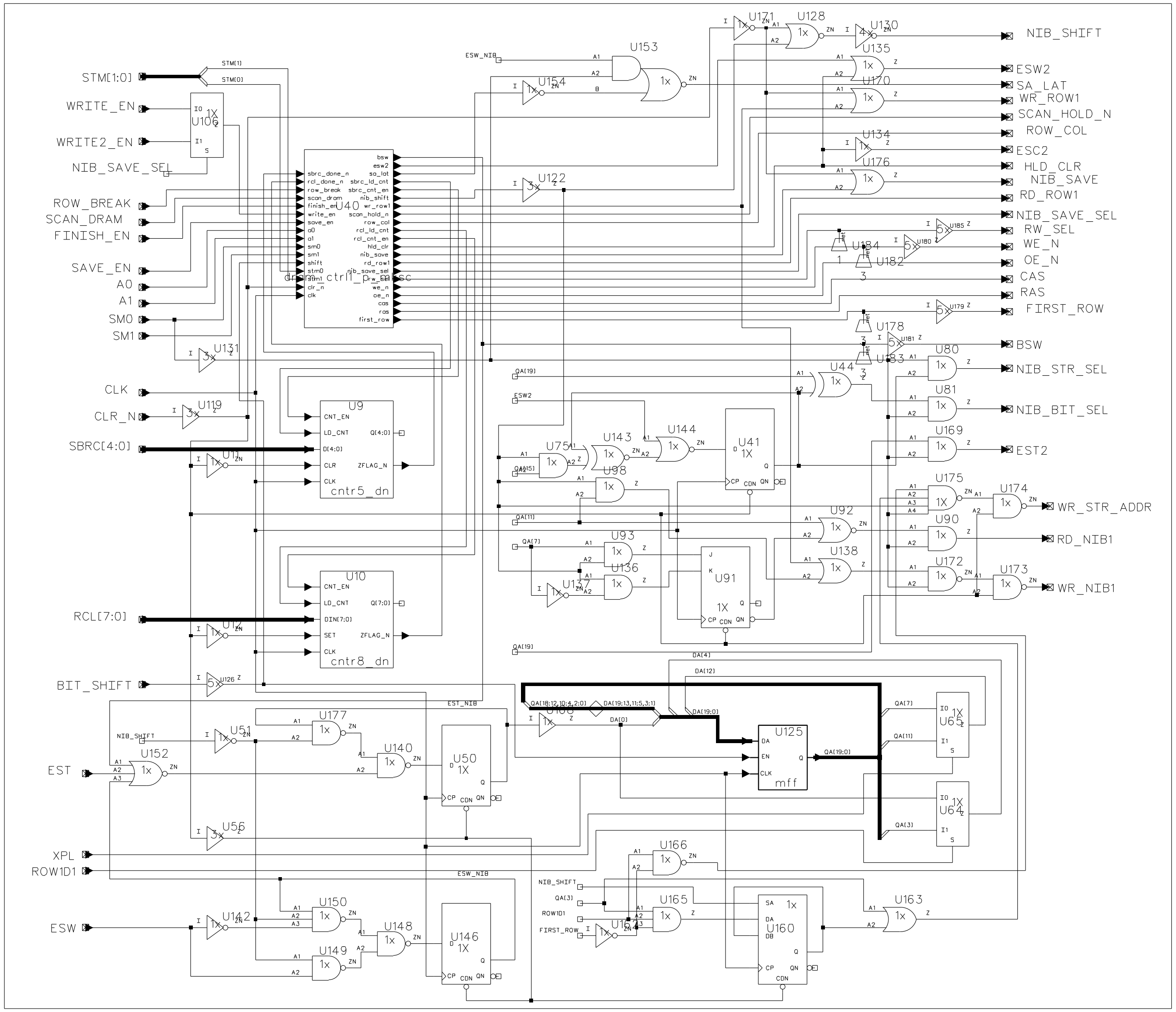

GLUE_ACTIVE
XYZ_KS[2:0]
REGEN_N[28:0]
GLUE_IN[5:0]
GLUE_CMP[5:0]
RMR_LD
REGSLT_N[22]
D
CLR
CLK
XYZ_OF[2:0]
ECLR[4:0]
XYZ_KM[2:0]
DISP_VALID
ST_EVC
EVCB_LD
ADDR_HI
SCAN_SHIFT
UPDATE_DATA
SAB
FLY_IN
UNGLUED_DATA
FPL
LPL
OP_CLR_N
WR_D
U138
glue_decode
U8
glue_pipe
U100
evc_control
U10
event_ctr
U9
dram_cnt_reg
U11
src_delay
U80
ssr_decode
U12
GLUE_OUT[5:0]
GLUE_EN_N[5:0]
Q[28,22,16,10,8:4]
GLUE_CMP
INT2
EVC_STAT
VW_CMP
LUT_OUT2
DISP_DATA
FLY_OUT
CELL_DATA
EVC_DATA
LUT_ADDR
SAB1
H_DATA1
UN_DATA1



STEPchip

dim_reg.ps


REGEN_N[28]
D
CLK
DATA_VALID
HI_LO
UNGL_DATA_DLY
CELL_DATA
U107
U127
U128
U106
U129
U130
U131
U104
U105
UP_DOWN
CNT_EN
LD_CNT
D
CLK
Q[7:0]
Q
cntr8_ud_sr
Q28



STEPchip

dram_ctrl.ps

```
#cell2 * dram_ctrl1 sm * 1 any 0 v8r1.5
# "16-Oct-90 GMT" "22:00:23 GMT" "16-Oct-90 GMT" "22:00:23 GMT" td * .
# DRAM timing FSM for CAM8
# Written by T. Durgavich Aug 23, 1990  REV 9
# filename = "dram_ctrl.sm"
# Last modified November 27, 1991

# {sm1,sm0} = 0  open stretch, no row-breaks no deferred writes
# {sm1,sm0} = 1  closed stretch, no row-breaks no deferred writes
# {sm1,sm0} = 2  open stretch, row-breaks allowed
# {sm1,sm0} = 3  closed stretch, row-breaks allowwed

# Moved scan_cnt_en to start 2 clocks earlier so we may latch
# site address on this date, 10/17/91

sm dramsm;

clock clk 25;

reset !clr_n --> s0;

inputs
       stm1
       stm0
       shift
       sm1
       sm0
       a1
       a0
       save_en
       write_en
       finish_en
       scan_dram
       row_break
       rcl_done_n
       sbrc_done_n;

rs=1 outputs
       first_row          #Set low on start of second row
       ras                #Must be RS so no glitches
       cas                #Must be RS so no glitches
       oe_n               #Must be RS so no glitches
       we_n               #Must be RS so no glitches
       rw_sel;            #Must be RS so no glitches

rs=0 outputs
       nib_save_sel
       rd_row1
       nib_save;           #Must be RS so no glitches

outputs
       hld_clr = 0
       rcl_cnt_en = 0
       rcl_ld_cnt = 0
       row_col = 0
       scan_hold_n = 0
       wr_row1 = 0
       nib_shift = 0
       sbrc_cnt_en = 0
```

```
       sbrc_ld_cnt = 0
       sa_lat = 0
       esw2 = 0
       bsw = 0;

# ROW_COL must be reclocked on the falling edge for
# a half clock delay
# WE_N must be reclocked on the falling edge

#   States 0 thru 3 perform a refresh
# ROW_COL is sm1 for test purposes we can view high and low bits
# of the address

state s0        #  ras=1 cas=1
{
let   row_col = sm1;
let   rcl_ld_cnt = 1;
let   sbrc_ld_cnt = 1;
let   sa_lat = 1;
}
      scan_dram   --> s4  cas = 1 ,# RS values on next clock edge
        --> s1 cas = 0;

state s1       #   ras=1 cas=0
{
let   row_col = sm1;
let   sa_lat = 1;
}
      scan_dram  --> s4 ras = 1 cas = 1 ,
      --> s2 ras = 0 cas = 0;

state s2        #  ras=0 cas=0
{
let   row_col = sm1;
let   cas = 1;
let   sa_lat = 1;
}
      scan_dram  --> s4 ras = 1 ,
      --> s3 ras = 0;

state s3        #  ras=0 cas=1
{
let   ras = 1;
let   row_col = sm1;
let   sa_lat = 1;
}
      scan_dram  --> s4 ,
      --> s0 ;

#   States 4 thru 7 perform a RAS/CAS read
#   for the first row

state s4        #  ras=1 cas=1
{
let   row_col =1;
let   first_row = 1;   #Set to a 1 in the next state
let   rw_sel = 1;
let   we_n = 1;
}
```

```
      --> s5;

state s5        #   ras=1 cas=1
{
let   ras = 0;
let   row_col =1;
}
      --> s6;

state s6        #   ras=0 cas=1
{
let   row_col = 0;
let   cas = 0;
let   oe_n = 0; #Ignore read if mode 3
}
      --> s7;

state s7         #  ras=0 cas=0 oe_n=0  DRAM read
{
let   row_col =0;
let   wr_row1 = 1;#Latch row and first nibble (mode 3)
let   oe_n = 1;
let   cas = 1;
let   sa_lat =  sm0; #High if mode 1,3
}
      --> s8;

state s8         #  oe_n=1
{
let   bsw = 1;
let   sa_lat = shift & !sm0 ; #High if mode 0,2
}
      (!a1 & !a0 & shift & !stm1 & !stm0) --> s9  rw_sel = 0,
      (!a1 &  a0 & shift & !stm1 & !stm0) --> s10 rw_sel =0, #No write
      ( a1 & !a0 & shift & !stm1 & !stm0) --> s11,
      ( a1 &  a0 & shift & !stm1 & !stm0) --> s12 oe_n = 0 cas = 0,
      (      !a0 & shift & !stm1 &  stm0) --> s9  rw_sel = 0,
      (       a0 & shift & !stm1 &  stm0) --> s11,
      (            shift &  stm1        ) --> s9  rw_sel = 0,
      --> s8;

state s9        #  oe_n=1 rw_sel=0
{
let    scan_hold_n = 1;
let    we_n = !write_en;
let    sa_lat = shift & !sm0; #High if mode 0,2
}
      shift & finish_en & sm1 -->
         s21 ras = 1 cas = 1,
      shift & finish_en & !sm1 -->
         s34 ras = 1 cas = 1 sbrc_cnt_en = 1 rcl_ld_cnt = 1,
      shift  --> s10,
       --> s9;

# Writes allowed in this state
state s10       # oe_n=1 rw_sel=0
{
let    scan_hold_n = 1;
let    we_n = 1;
```

```
}
      shift & finish_en & sm1 -->
         s21 ras = 1 cas = 1,
      shift & finish_en & !sm1 -->
         s34 ras = 1 cas = 1 sbrc_cnt_en = 1 rcl_ld_cnt = 1,
      shift  --> s11 rw_sel = 1 cas = 1,
      --> s10;

state s11       #   oe_n=1 rw_sel=1
{
let   scan_hold_n = 1;
}
      shift & finish_en & sm1 -->
         s21 ras = 1 cas = 1,
      shift & finish_en & !sm1 -->
         s34 ras = 1 cas = 1 sbrc_cnt_en = 1 rcl_ld_cnt = 1,
      shift  --> s12 oe_n = 0 cas = 0 rw_sel = 1,
      --> s11;

state s12       #  oe_n=0  rw_sel=1 DRAM read or shift when finished
{
let   oe_n = shift;
let   scan_hold_n = !row_break ;
let   sa_lat = shift &  sm0 & !row_break; #High if mode 1,3
let   nib_shift = shift;
}
      shift & finish_en & sm1 -->
         s21 ras = 1 cas = 1,
      shift & finish_en & !sm1 -->
         s34 ras = 1 cas = 1 sbrc_cnt_en = 1 rcl_ld_cnt = 1,
      shift & row_break  -->
         s13 ras = 1 cas = 1 first_row = 0,
      shift & !row_break  -->
         s9 rw_sel = 0 cas = 1,
      --> s12;

#   States 13 thru 21 perform a RAS/CAS read
#   for the second row

state s13        # ras=1 cas=1 rw_sel=1  Row break occurred first_row=0
{
let   row_col = 1;
let   nib_save = save_en & write_en & shift;
}
      shift --> s14 ras = 1 cas = 1,
      --> s13;

state s14        # ras=1 cas=1 rw_sel=1 nib_save=1
{
let   ras = 0;
let   row_col =1;
let   nib_save = 0;
}
      shift --> s15 rw_sel = 1,
      --> s14;

state s15        # ras=0 cas=1 rw_sel=1
{
let   row_col = 0;
```

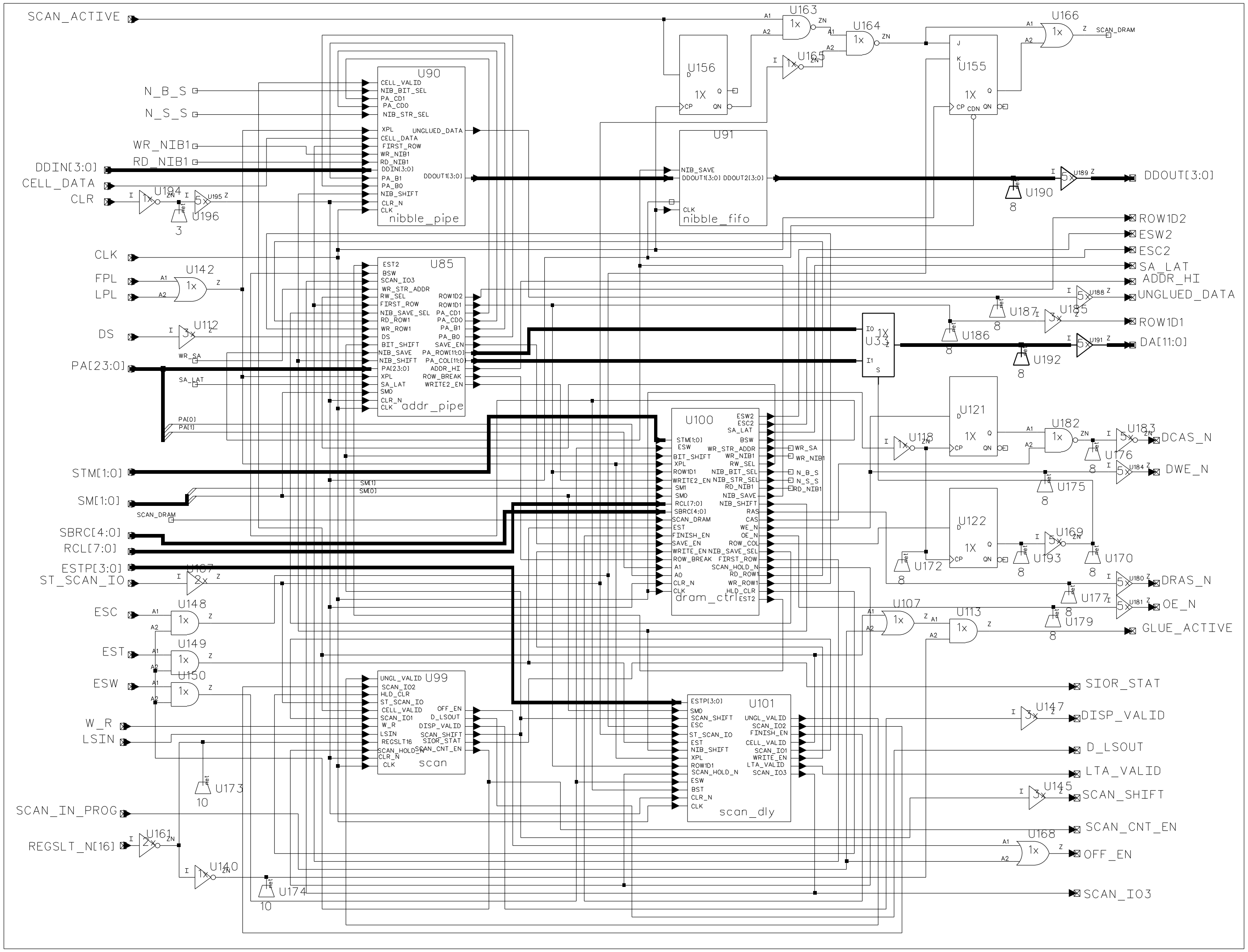
```
let   oe_n = 0;
}
      shift --> s16 cas = 0,
      --> s15;

state s16    #  ras=0 cas=0 oe_n=0 rw_sel=1
{
let   row_col =0;
let   nib_shift = shift;
let   oe_n = shift;
let   scan_hold_n = 1;
let   sa_lat = shift & sm0; #High if mode 1,3
}
      shift --> s17 rw_sel = 0 cas = 1,
      --> s16;

# States 17-20 Read/Write
state s17        #  ras=0 cas=0 oe_n=1 rw_sel=0
{
let   scan_hold_n = 1;
let   oe_n = 1;
let   we_n = !(write_en & !save_en);
let   nib_save = write_en &  save_en & shift;
let   sa_lat = shift & !sm0; #High if mode 0,2
}
      shift & finish_en --> s21 ras = 1 cas = 1,
      shift  --> s18,
      --> s17;

state s18        #  oe_n=1 rw_sel=0
{

let   scan_hold_n = 1;
let   we_n = 1;
let   nib_save = 0;
}
      shift & finish_en --> s21 ras = 1 cas = 1,
      shift  --> s19 rw_sel = 1 cas = 1,
      --> s18;

state s19       #  oe_n=1 rw_sel=1
{
let   scan_hold_n = 1;
}
      shift & finish_en --> s21 ras = 1 cas = 1,
      shift  --> s20 oe_n = 0 cas = 0,
      --> s19;

state s20       #  oe_n=0 rw_sel=1
{
let   oe_n = shift;
let   scan_hold_n = 1;
let   nib_shift = shift;
let   sa_lat = shift & sm0; #High if mode 1,3
}
      shift & finish_en --> s21 ras = 1 cas = 1 ,
      shift --> s17 rw_sel = 0 cas = 1,
      --> s20;
```

```
#
# The next 14 states simply write the stored data from the row change
# The data to be written is 0 - 4 nibbles.
# If no row change occurred these 12 states must still be done to
# keep in synchronism with the chips that may have had a row change


state s21      #  ras=1 cas=1 rw_sel=0/1
{
let   rd_row1 = 1;
let   row_col = 1;
let   nib_save_sel = 1;
let   rw_sel = 0;
let   we_n = 1;
let   oe_n = 1;
}
      --> s22;

state s22      #  ras=1 cas=1 rw_sel=0
{
let   rd_row1 = 1;
let   ras = 0;
let   row_col = 1;
}
      --> s23;

state s23     #  ras=0 cas=1 rw_sel=0
{
let   rd_row1 = 0;
let   row_col = 0;
}
      --> s24;

# CAS is externally generated by reclocking WE_N (falling edge)

state s24     #  ras=0 rw_sel=0
{
let   we_n = !write_en;
let   nib_save = 1;
let   row_col = 0;
}
      --> s25;

state s25    #  rw_sel=0
{
let   we_n = 1;
let   nib_save = 0;
}
      --> s26;

state s26    #   rw_sel=0
{
let   we_n = !write_en;
let   nib_save = 1;
}
      --> s27;

state s27    # rw_sel=0
{
```

```
let   we_n = 1;
let   nib_save = 0;
}
      --> s28;

state s28    # rw_sel=0
{
let   we_n = !write_en;
let   nib_save = 1;
}
      --> s29;

state s29    #  rw_sel=0
{
let   we_n = 1;
let   nib_save = 0;
}
      --> s30;

state s30    #    rw_sel=0
{
let   we_n = !write_en;
let   nib_save = 1;
}
      --> s31;

state s31   #   rw_sel=0
{
let   we_n = 1;
let   nib_save = 0;
}
      --> s32;

state s32   #   rw_sel=0
{
let   we_n = !write_en;
let   nib_save = 1;
}
      --> s33;

state s33   #  rw_sel=0
{
let   sbrc_cnt_en = 1;
let   rcl_ld_cnt = 1;
let   we_n = 1;
let   nib_save = 0;
}
      --> s34 ras = 1 cas = 1 nib_save_sel = 0;

# Start of end-of-sweep refreshes with RCL and SBRC
state s34    # ras=1 cas=1
{
let   row_col = 1;
let   first_row = 1;                    # RS flip-flop
let   oe_n = 1;
let   we_n = 1;
let   rw_sel = 1;
}
      !scan_dram & !rcl_done_n --> s0              # Finished scan, go home
```

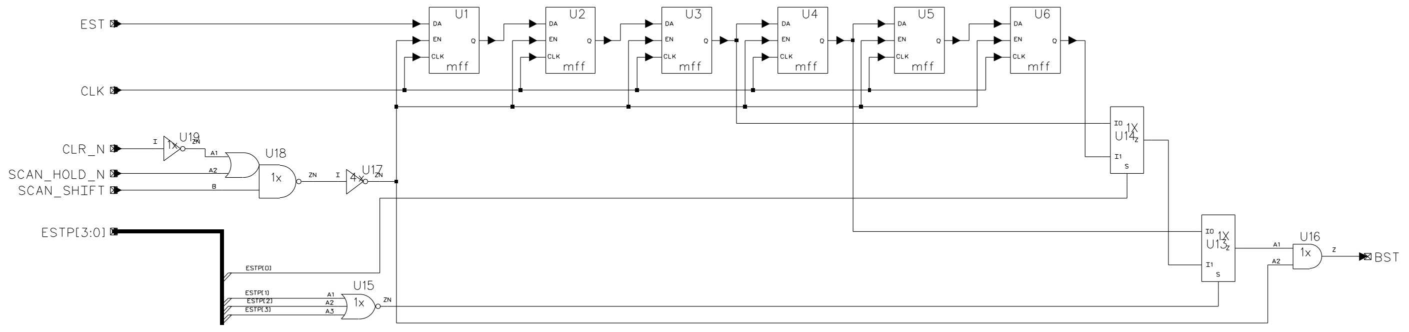
```
        sbrc_ld_cnt = 1 rcl_ld_cnt = 1 hld_clr = 1 ,
      !scan_dram & sbrc_done_n & rcl_done_n --> s35
         cas = 0,
       scan_dram & !sbrc_done_n & !rcl_done_n --> s4
         sbrc_ld_cnt = 1 esw2 = 1,                # Next sweep,reload SBRC
       scan_dram & sbrc_done_n --> s4
         esw2 =1,                                 # Next sweep
       !sbrc_done_n & rcl_done_n --> s35
         cas = 0;                                 # Next refresh

state s35   #  ras=1 cas=0
{
let   ras = 0;
let   row_col = 1;
}
      --> s36;

state s36  #  ras=0 cas=0
{
let   row_col = 1;
let   cas = 1;
}
      --> s37;

state s37  #  ras=0 cas=1
{
let   ras = 1;
let   row_col = 1;
let   rcl_cnt_en = 1;
}
      --> s34;

end
```

[la]dram_if by nhm on 15-Feb-93 at 4:24 P.M.



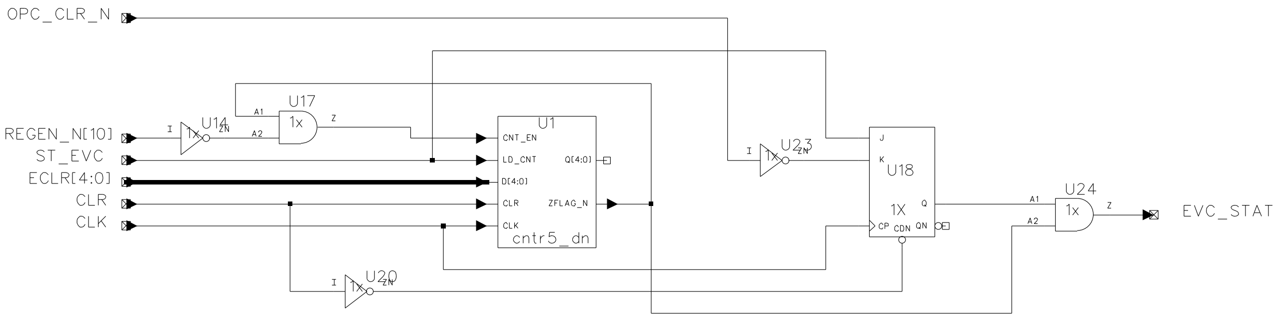
STEPchip

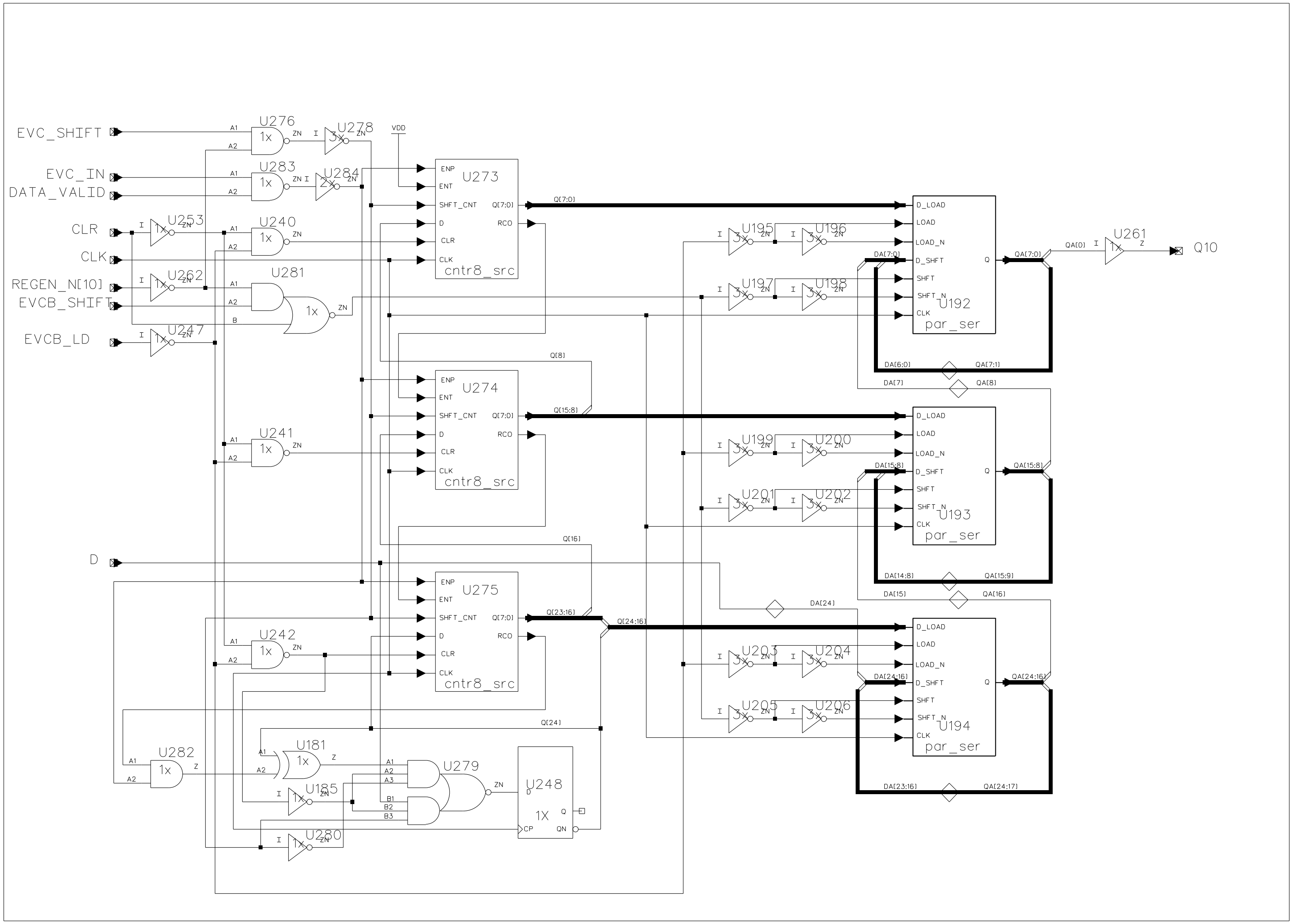
[la]est_dly by nhm on 15-Feb-93 at 4:25 P.M.



STEPchip

est_dly.ps

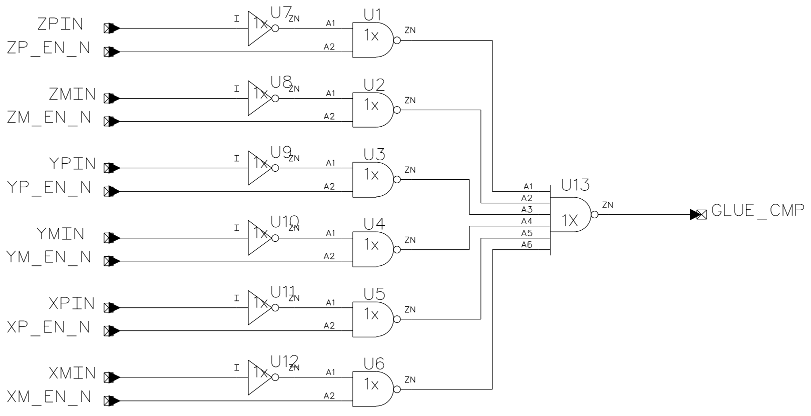

OPC_CLR_N
REGEN_N[10]
ST_EVC
ECLR[4:0]
CLR
CLK
U14
U17
U1
CNT_EN
LD_CNT
D[4:0]
CLR
CLK
Q[4:0]
ZFLAG_N
cntr5_dn
U20
U23
U18
J
K
Q
QN
CP
CDN
U24
EVC_STAT



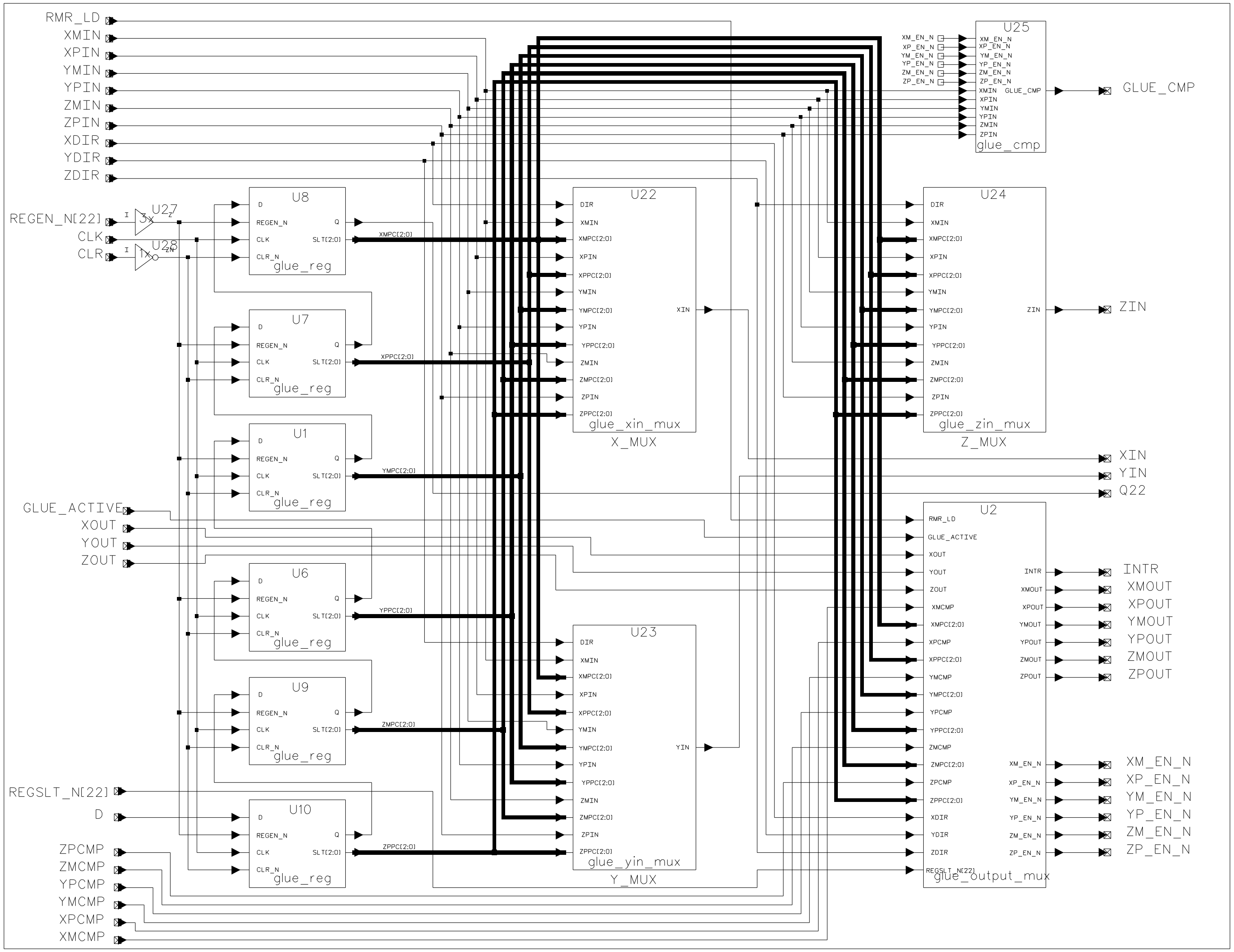
STEPchip

event_ctr.ps

[la]glue_cmp by nhm on 15-Feb-93 at 4:25 P.M.



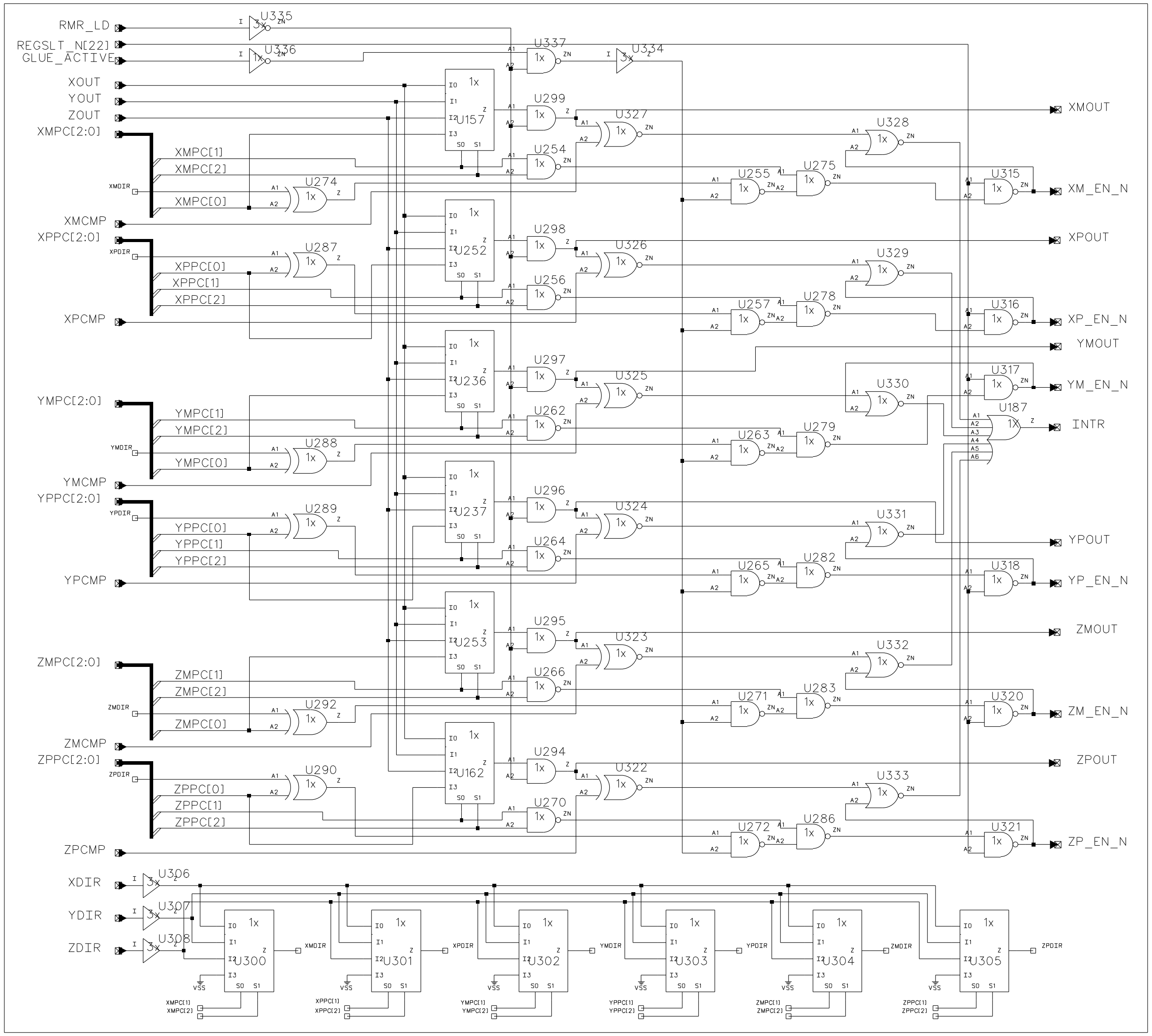
[la]glue_decode by nhm on 15-Feb-93 at 4:25 P.M.



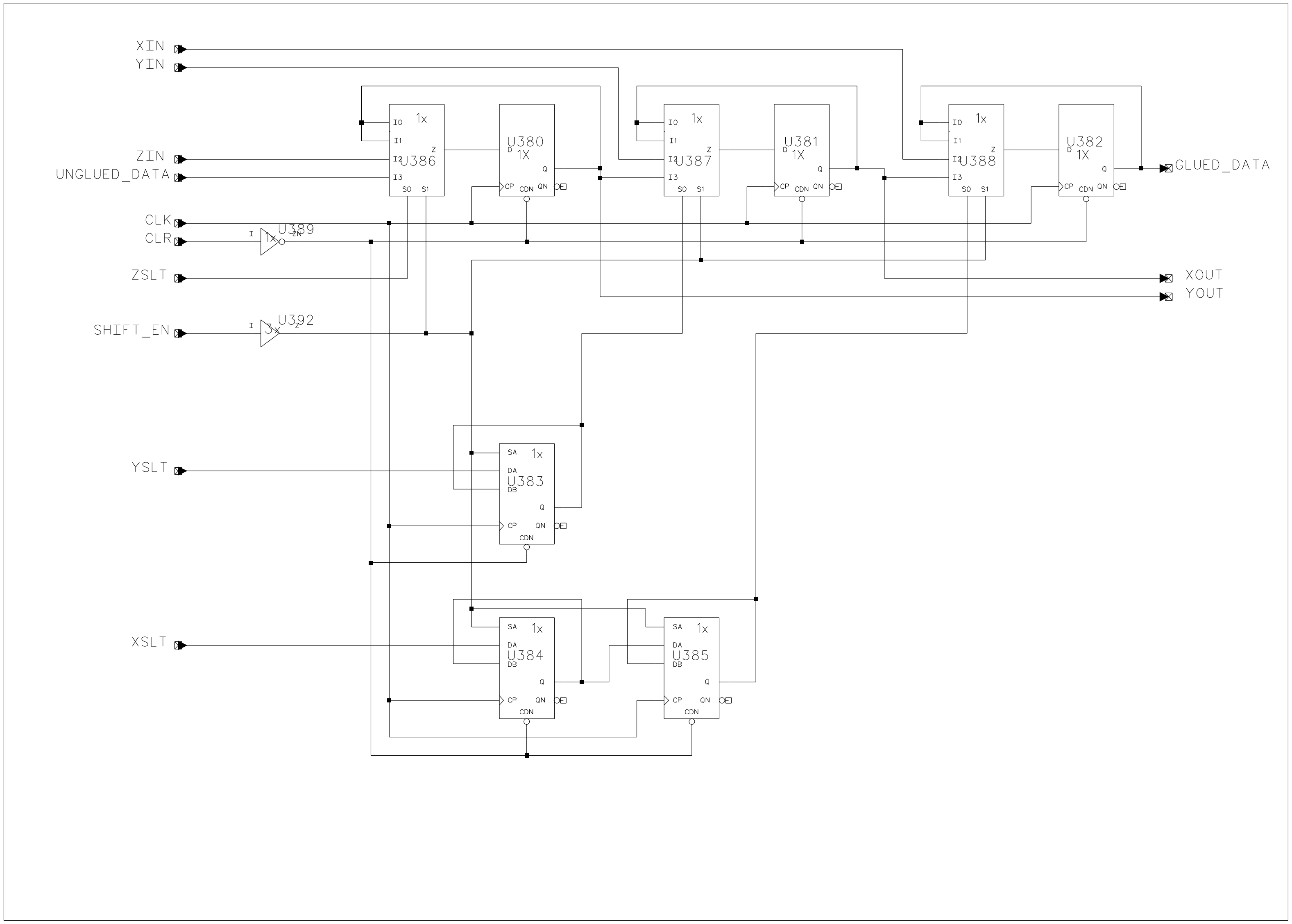
STEPchip

glue_decode.ps

[la]glue_output_mux by nhm on 15-Feb-93 at 4:25 P.M.

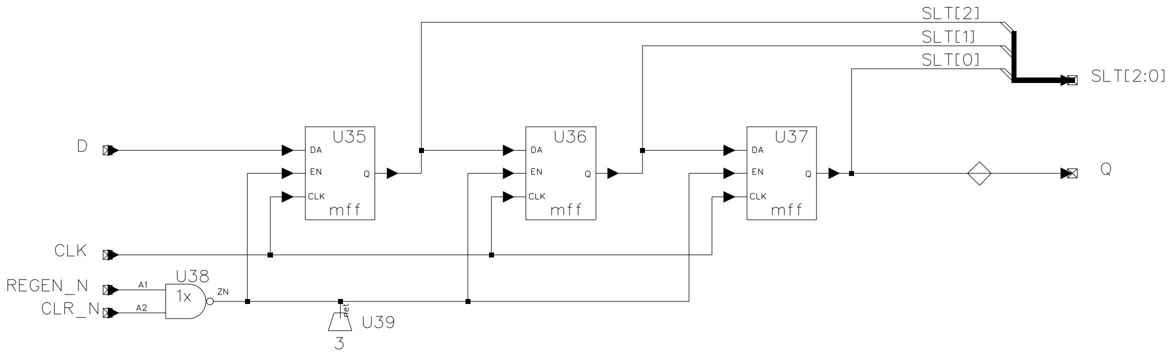


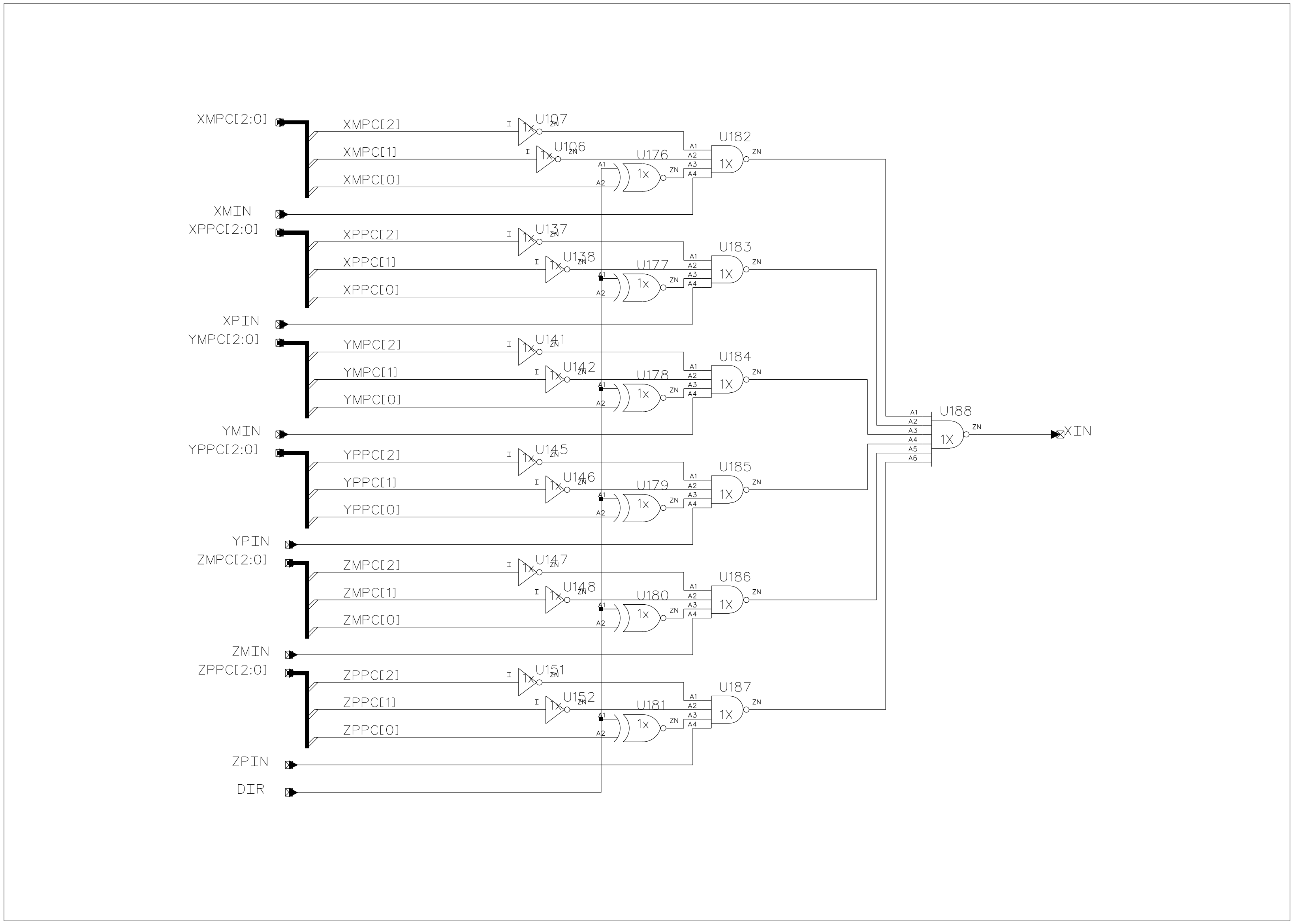
STEPchip



STEPchip

glue_pipe.ps

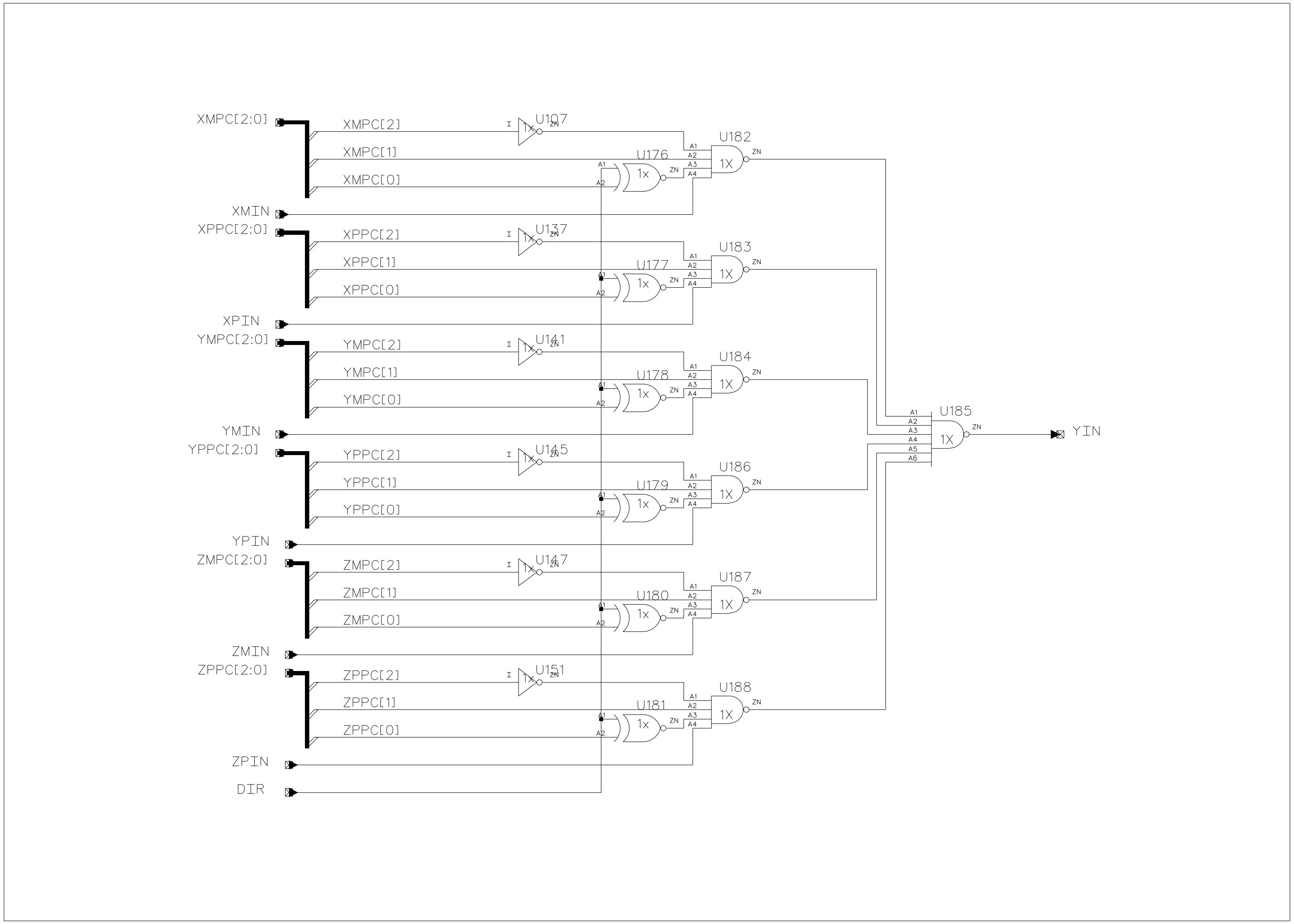

SLT[2]
SLT[1]
SLT[0]
SLT[2:0]
D
U35
DA
EN
CLK
Q
mff
U36
U37
Q
CLK
REGEN_N
CLR_N
U38
A1
A2
1x
ZN
U39
3

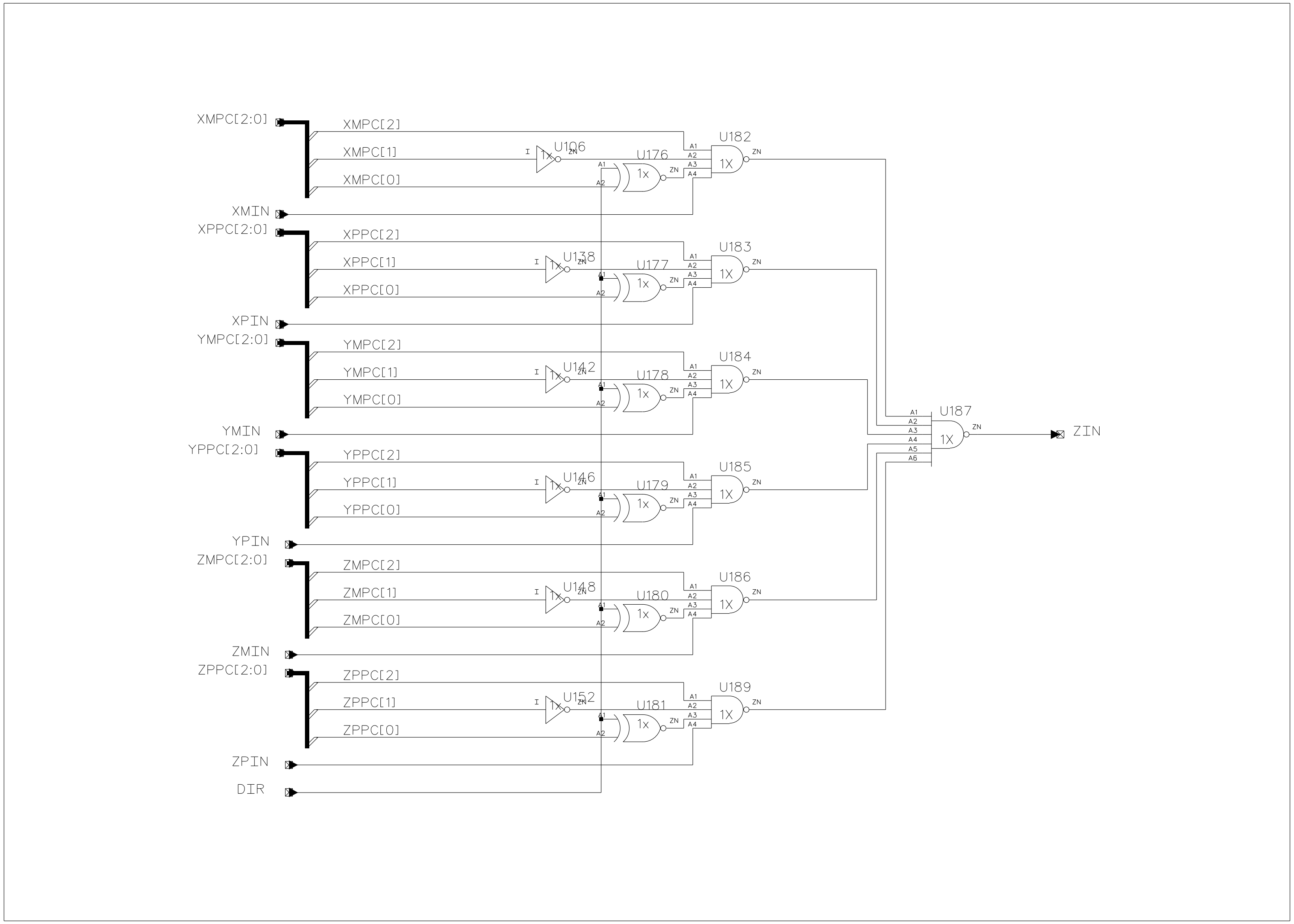


STEPchip

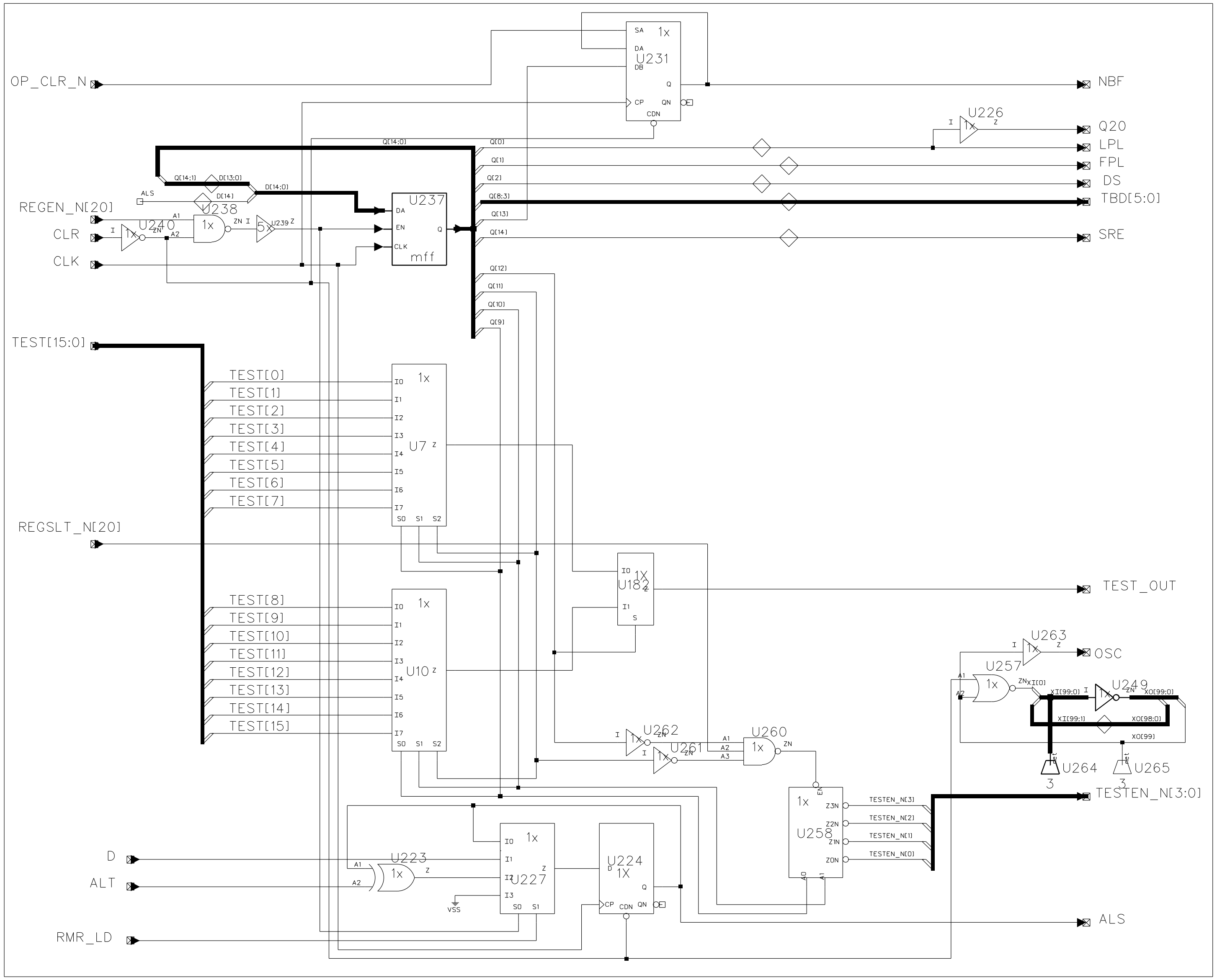
glue_xin_mux.ps

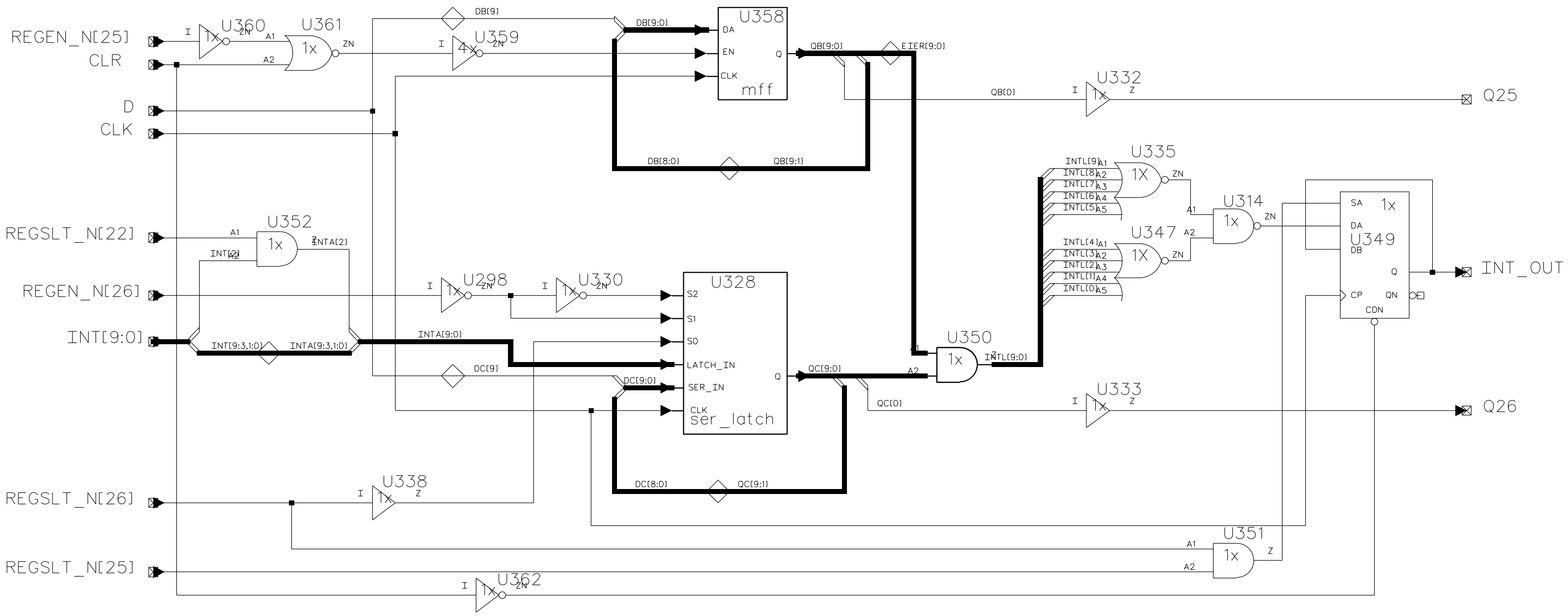
[la]glue_yin_mux by nhm on 15-Feb-93 at 4:27 P.M.




XMPC[2:0]
XMPC[2]
XMPC[1]
XMPC[0]
U106
U176
U182
XMIN
XPPC[2:0]
XPPC[2]
XPPC[1]
XPPC[0]
U138
U177
U183
XPIN
YMPC[2:0]
YMPC[2]
YMPC[1]
YMPC[0]
U142
U178
U184
YMIN
YPPC[2:0]
YPPC[2]
YPPC[1]
YPPC[0]
U146
U179
U185
U187
ZIN
YPIN
ZMPC[2:0]
ZMPC[2]
ZMPC[1]
ZMPC[0]
U148
U180
U186
ZMIN
ZPPC[2:0]
ZPPC[2]
ZPPC[1]
ZPPC[0]
U152
U181
U189
ZPIN
DIR

[la]hw_env_reg by nhm on 15-Feb-93 at 4:27 P.M.

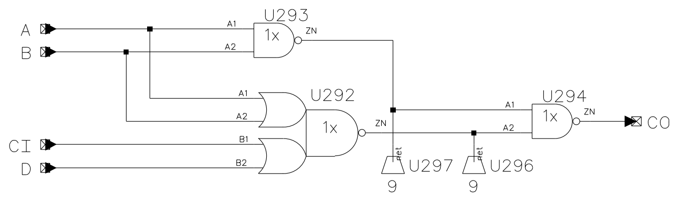




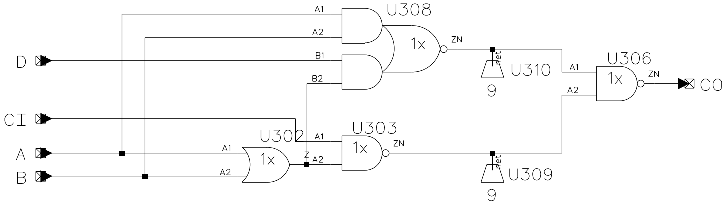
STEPchip

interrupt_reg.ps

[la]kick_cy_h by nhm on 15-Feb-93 at 4:27 P.M.

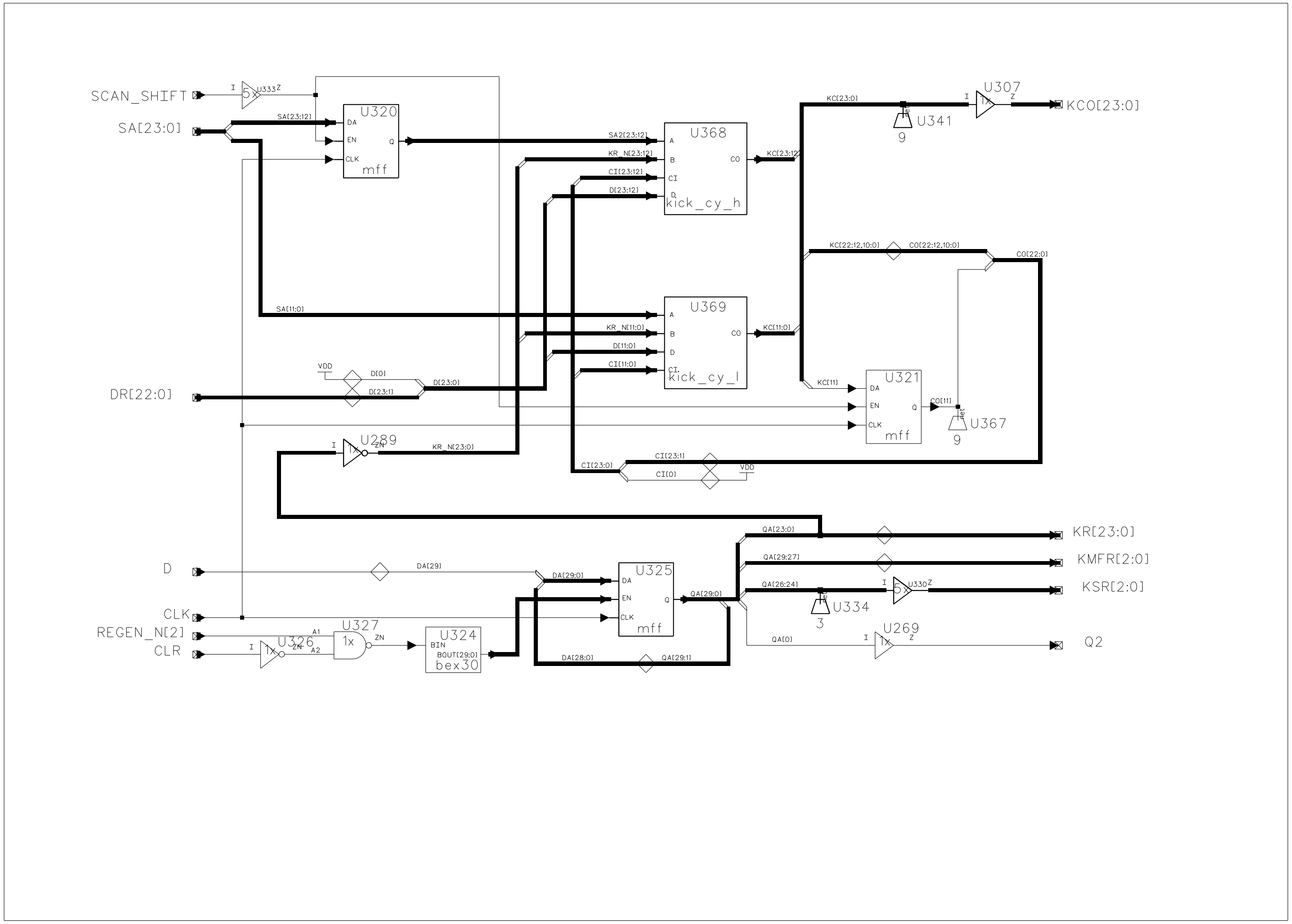


[la]kick_cy_l by nhm on 15-Feb-93 at 4:27 P.M.



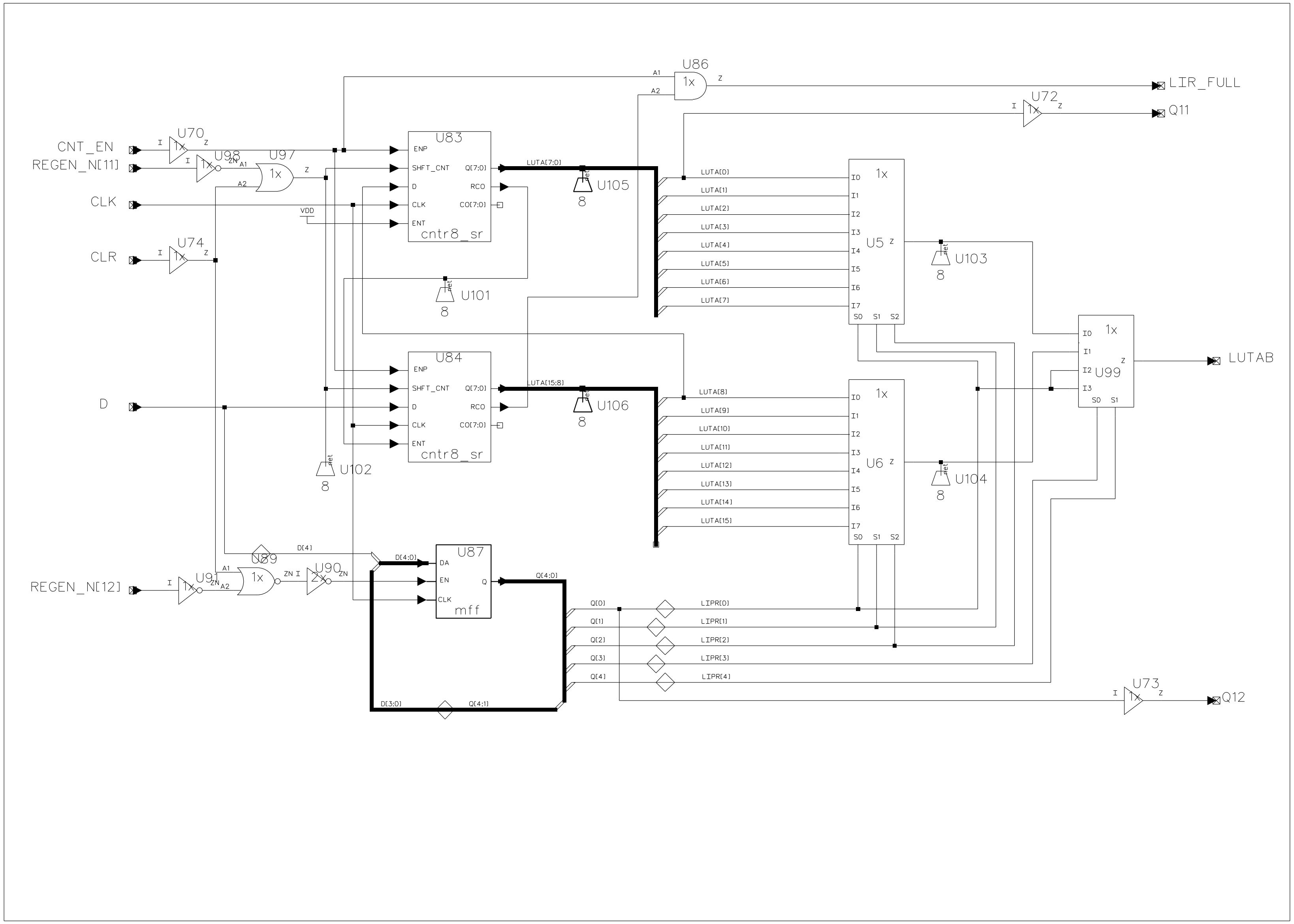
STEPchip

kick_cy_l.ps



SCAN_SHIFT
SA[23:0]
DR[22:0]
D
CLK
REGEN_N[2]
CLR
U333
U320
mff
U368
kick_cy_h
U369
kick_cy_l
U321
mff
U367
U307
U341
KCO[23:0]
U289
U325
mff
U327
U326
U324
bex30
U334
U330
U269
KR[23:0]
KMFR[2:0]
KSR[2:0]
Q2

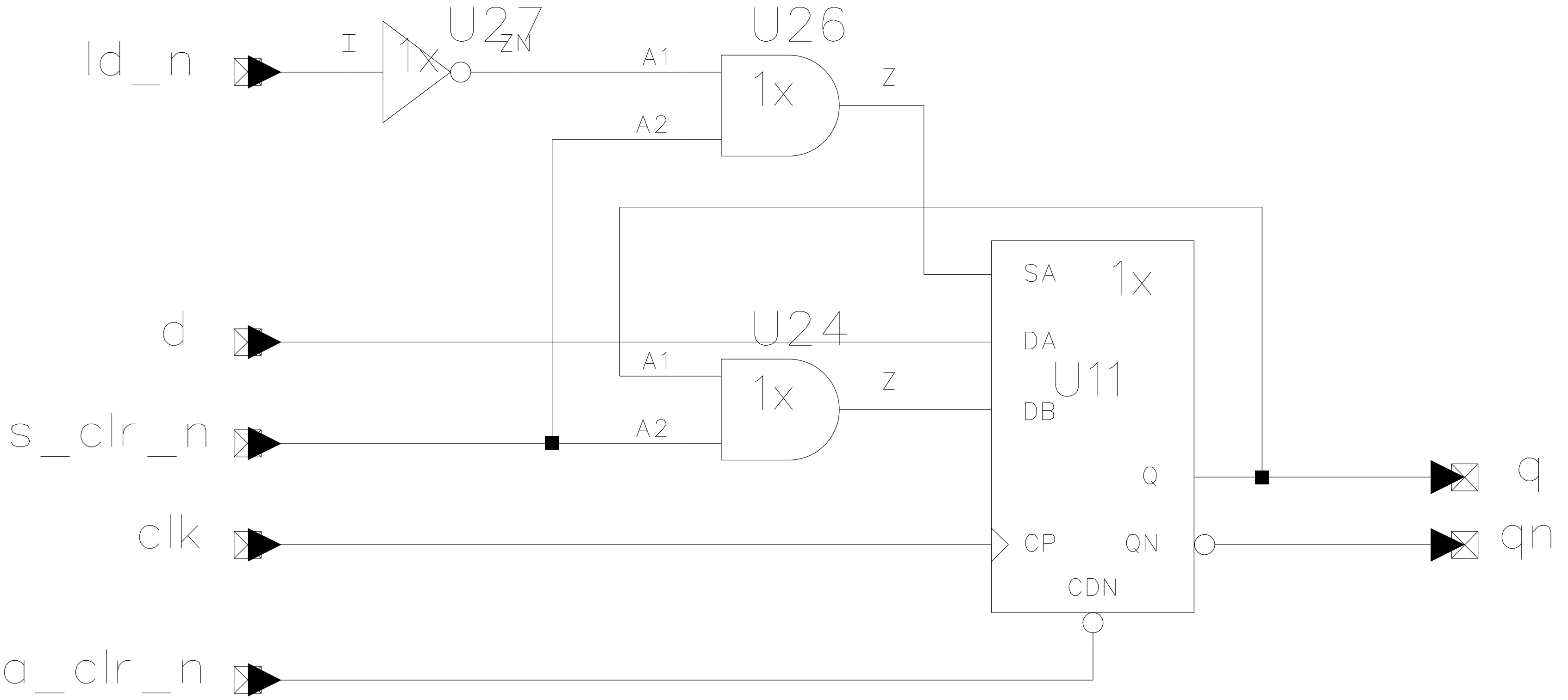
STEPchip



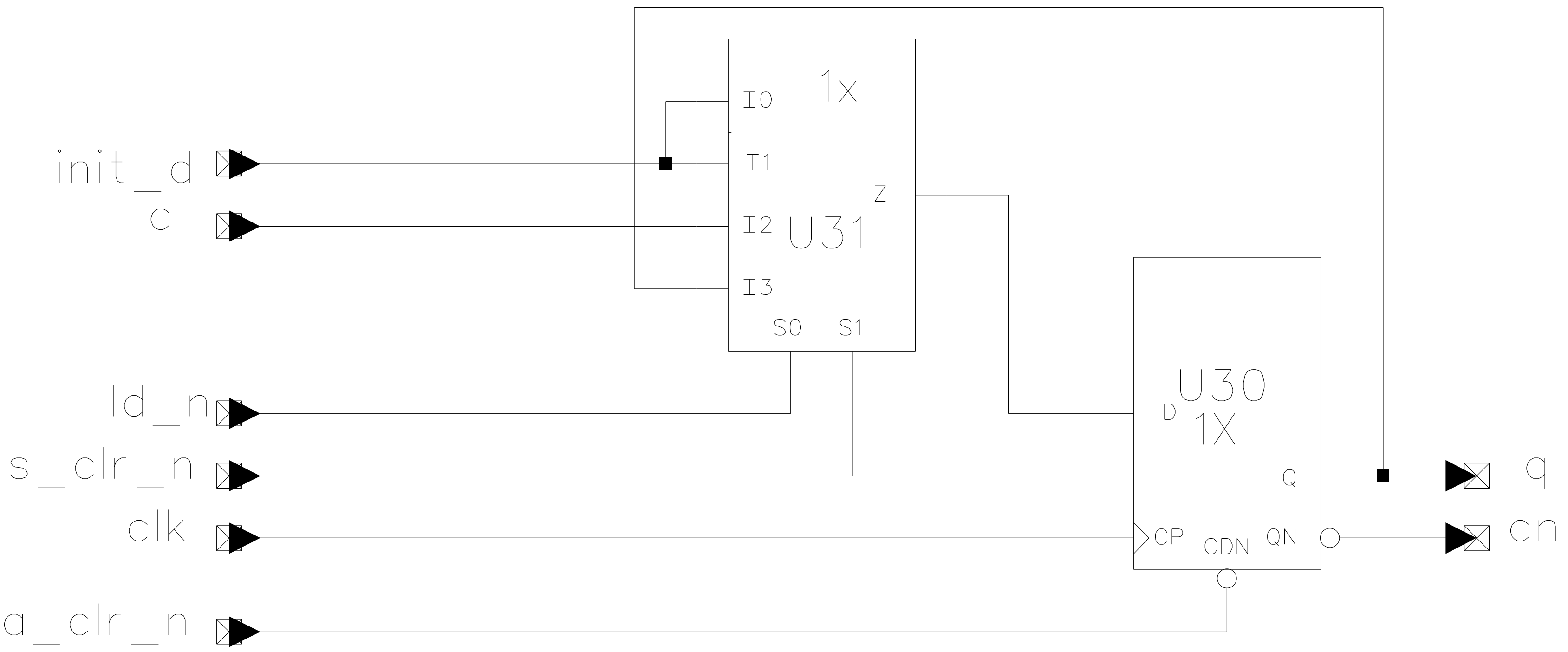
STEPchip

lutmac_reg.ps

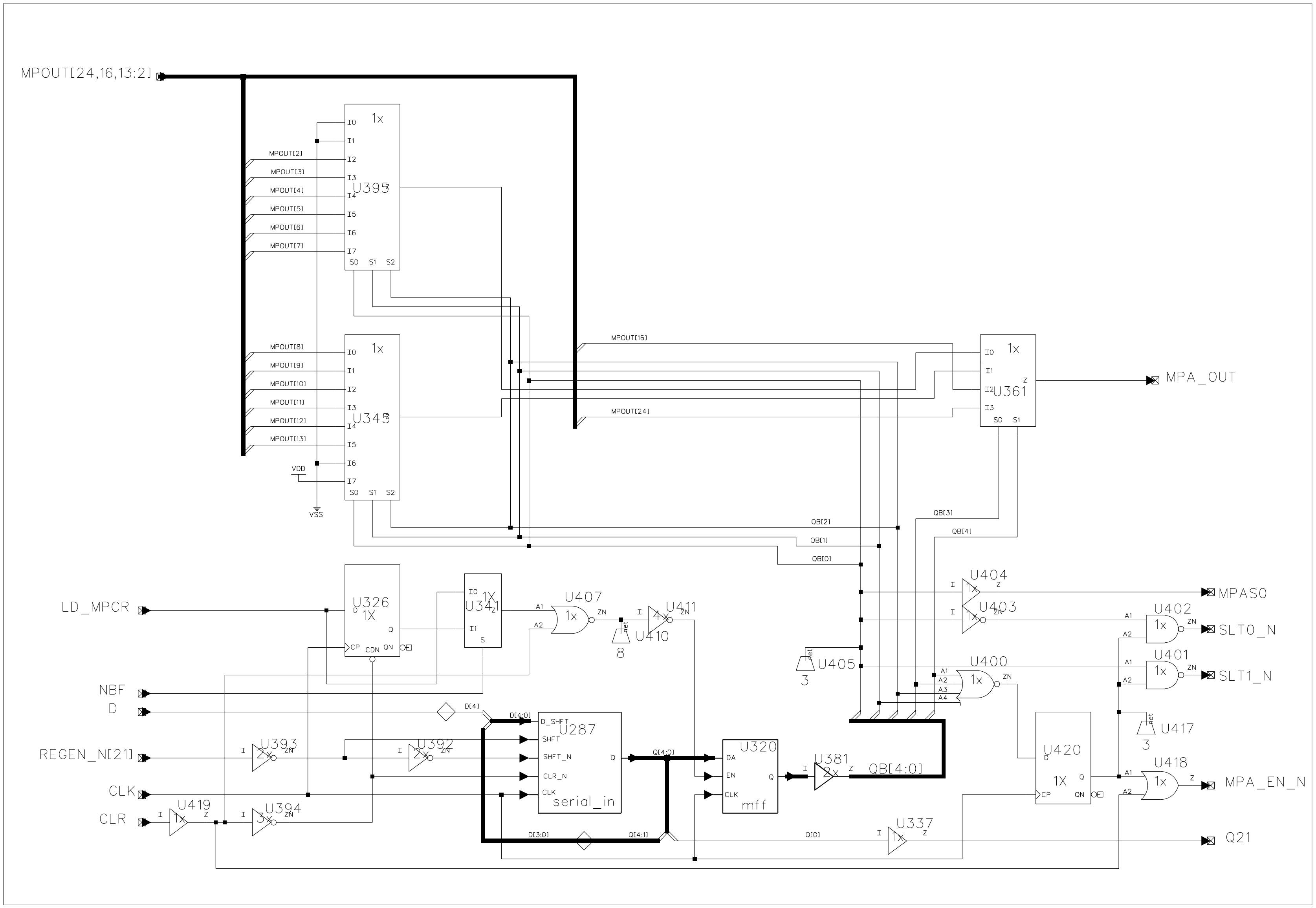
[la]mfct_sync by nhm on 15-Feb-93 at 4:28 P.M.



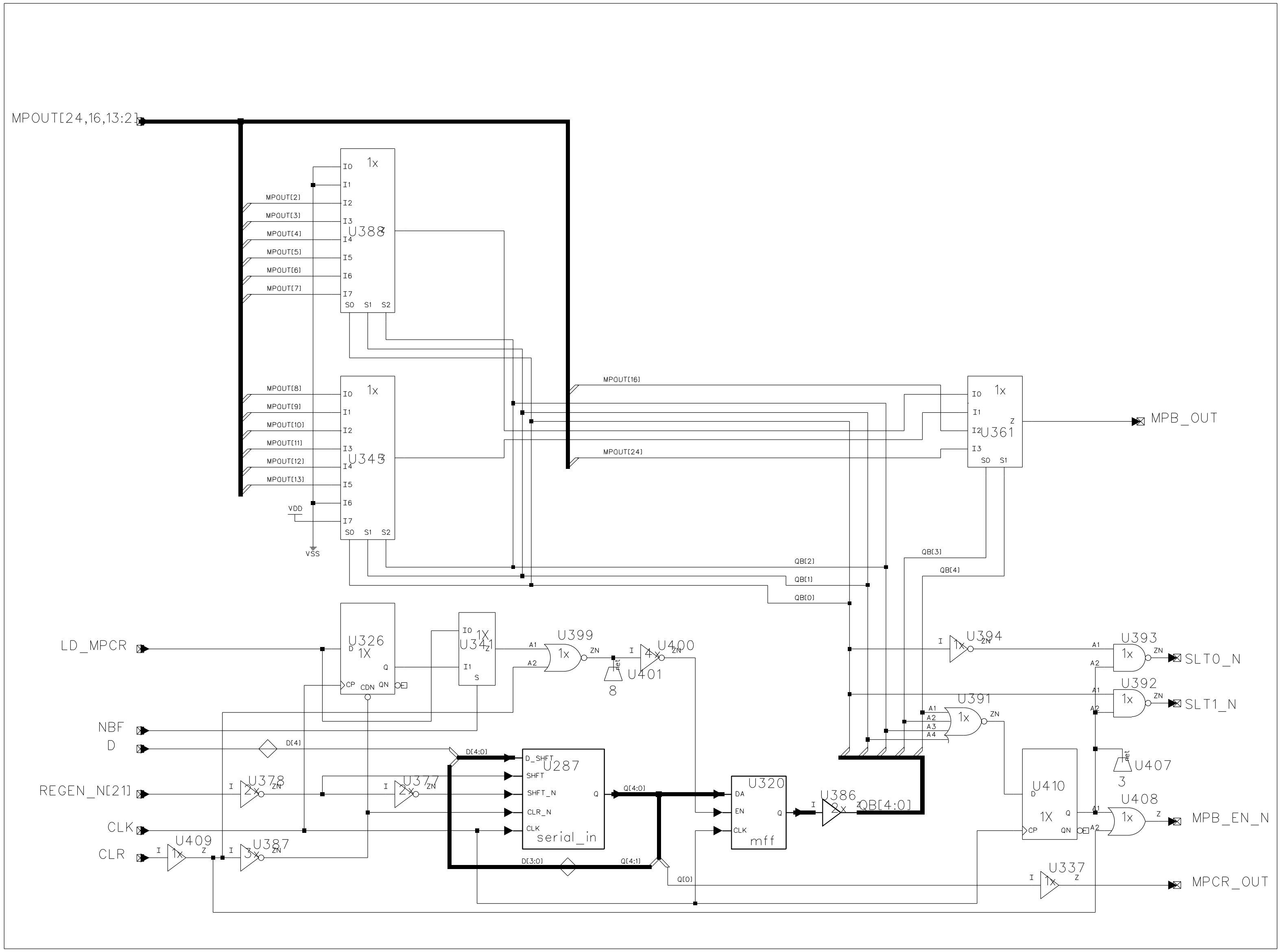
ff with enable, async clear, and sync clear



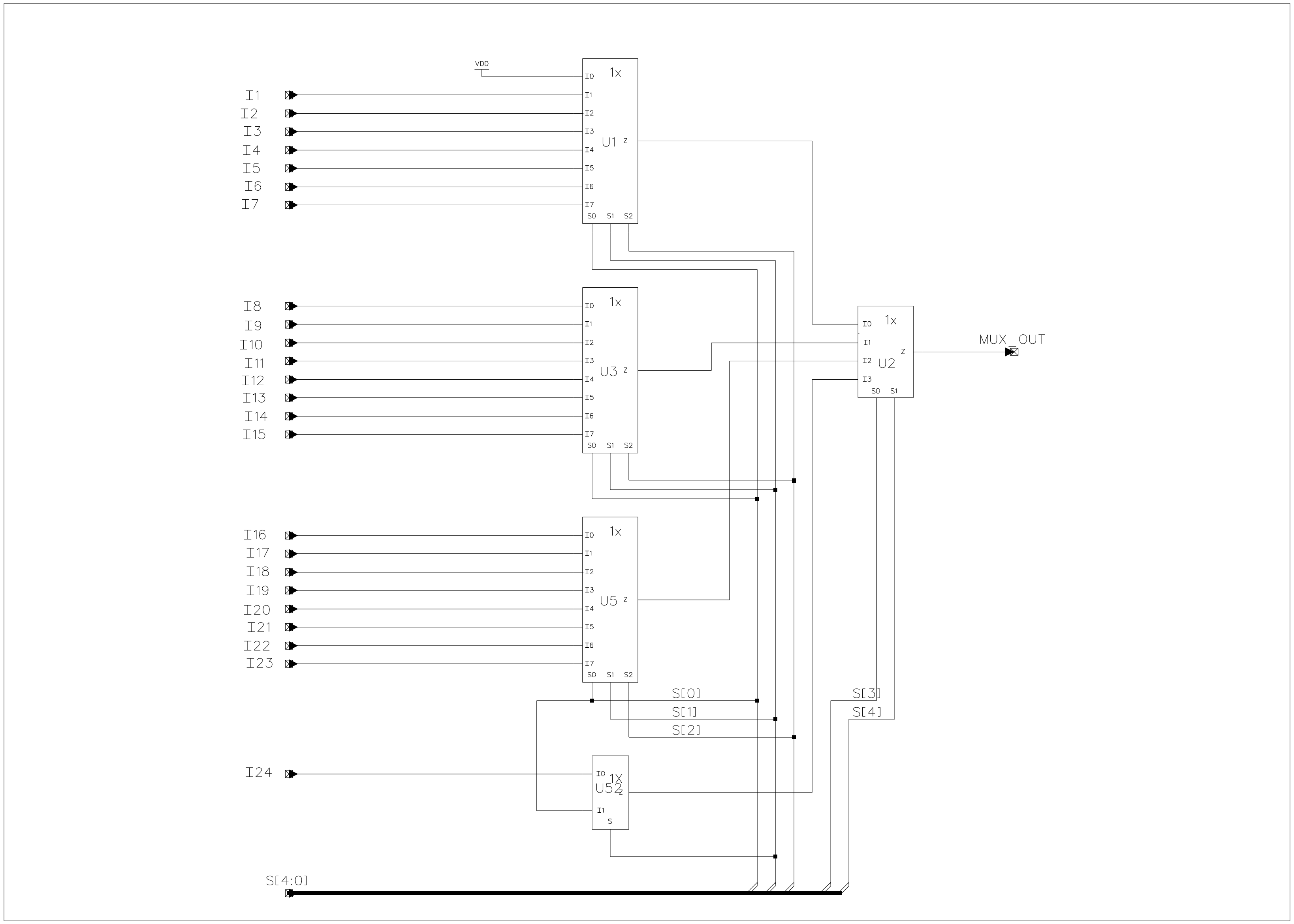
ff with enable, async clear, and sync initialization


MPOUT[24,16,13:2]
U395
U345
U361
MPA_OUT
MPOUT[16]
MPOUT[24]
QB[0]
QB[1]
QB[2]
QB[3]
QB[4]
VDD
VSS
LD_MPCR
U326
U341
U407
U411
U410
U404
U403
U405
U400
U402
U401
MPAS0
SLT0_N
SLT1_N
NBF
D
REGEN_N[21]
CLK
CLR
U419
U393
U394
U392
U287
serial_in
U320
mff
U381
QB[4:0]
U337
U420
U417
U418
MPA_EN_N
Q21
D[4]
D[4:0]
D[3:0]
Q[4:0]
Q[4:1]
Q[0]

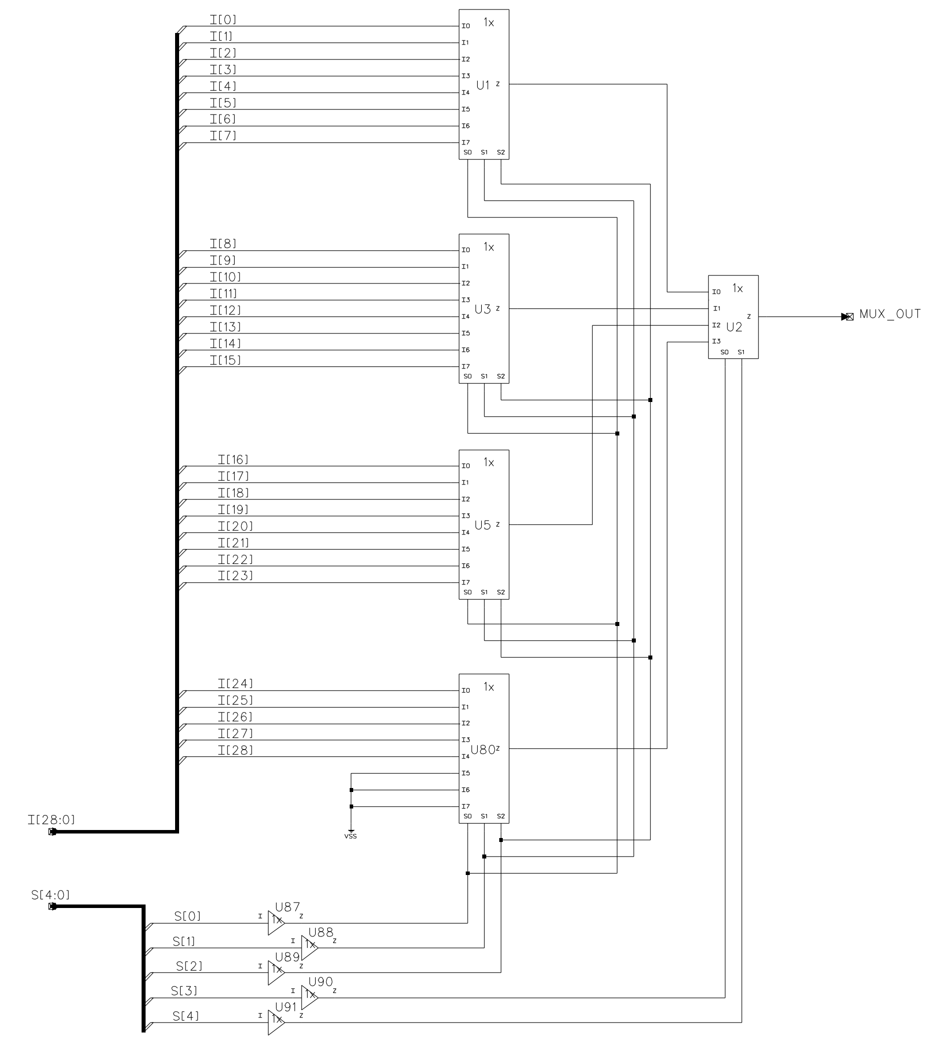
STEPchip


MPOUT[24,16,13:2]
U388
U343
U361
MPOUT[16]
MPOUT[24]
MPB_OUT
VDD
VSS
QB[0]
QB[1]
QB[2]
QB[3]
QB[4]
LD_MPCR
NBF
D
REGEN_N[21]
CLK
CLR
U326
U341
U399
U400
U401
U394
U393
U392
U391
SLT0_N
SLT1_N
U287
serial_in
U320
mff
U386
QB[4:0]
U378
U377
U409
U387
U410
U407
U408
MPB_EN_N
U337
MPCR_OUT

[la]mux24_1 by nhm on 15-Feb-93 at 4:28 P.M.



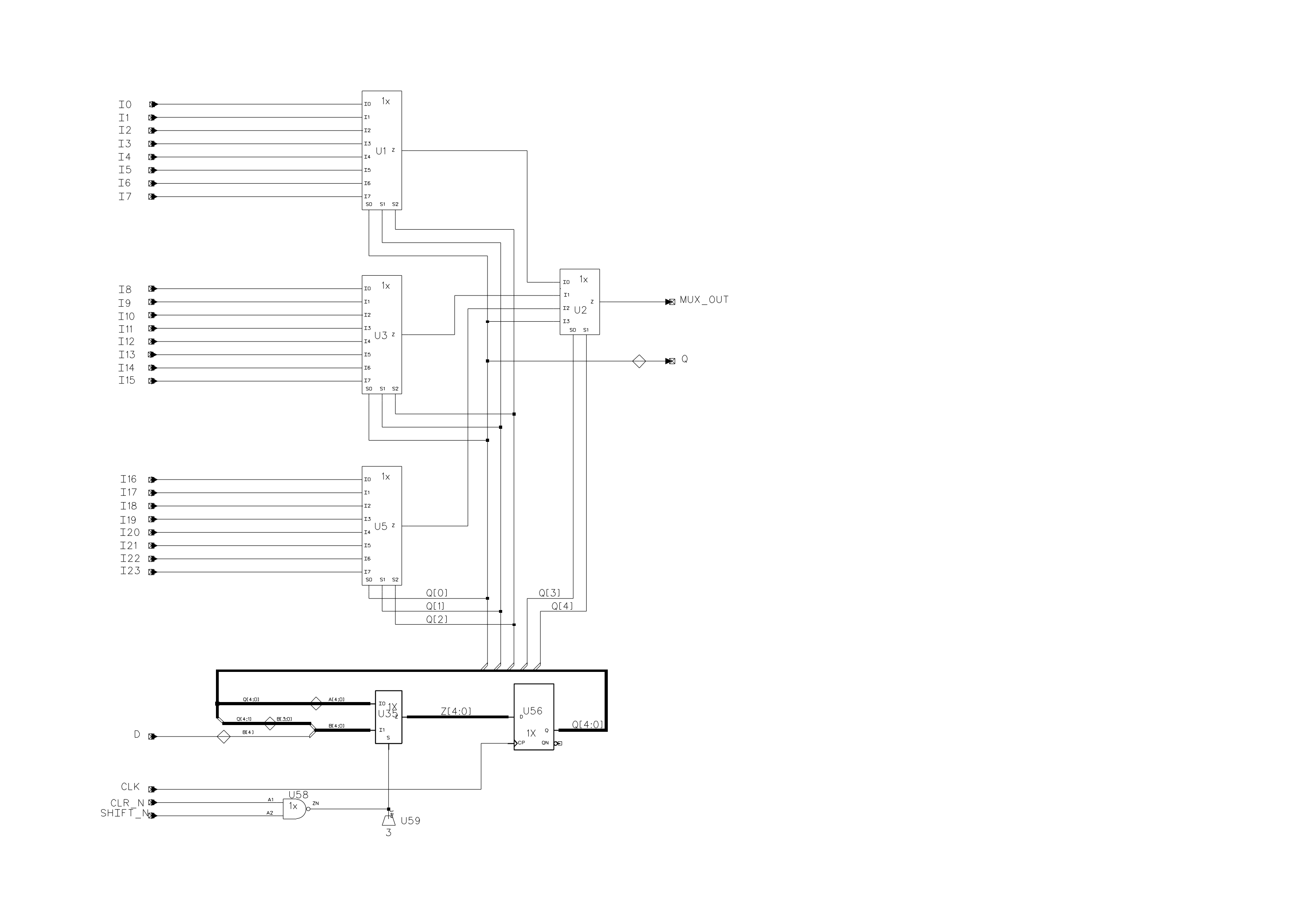
[la]mux29_1 by nhm on 15-Feb-93 at 4:28 P.M.



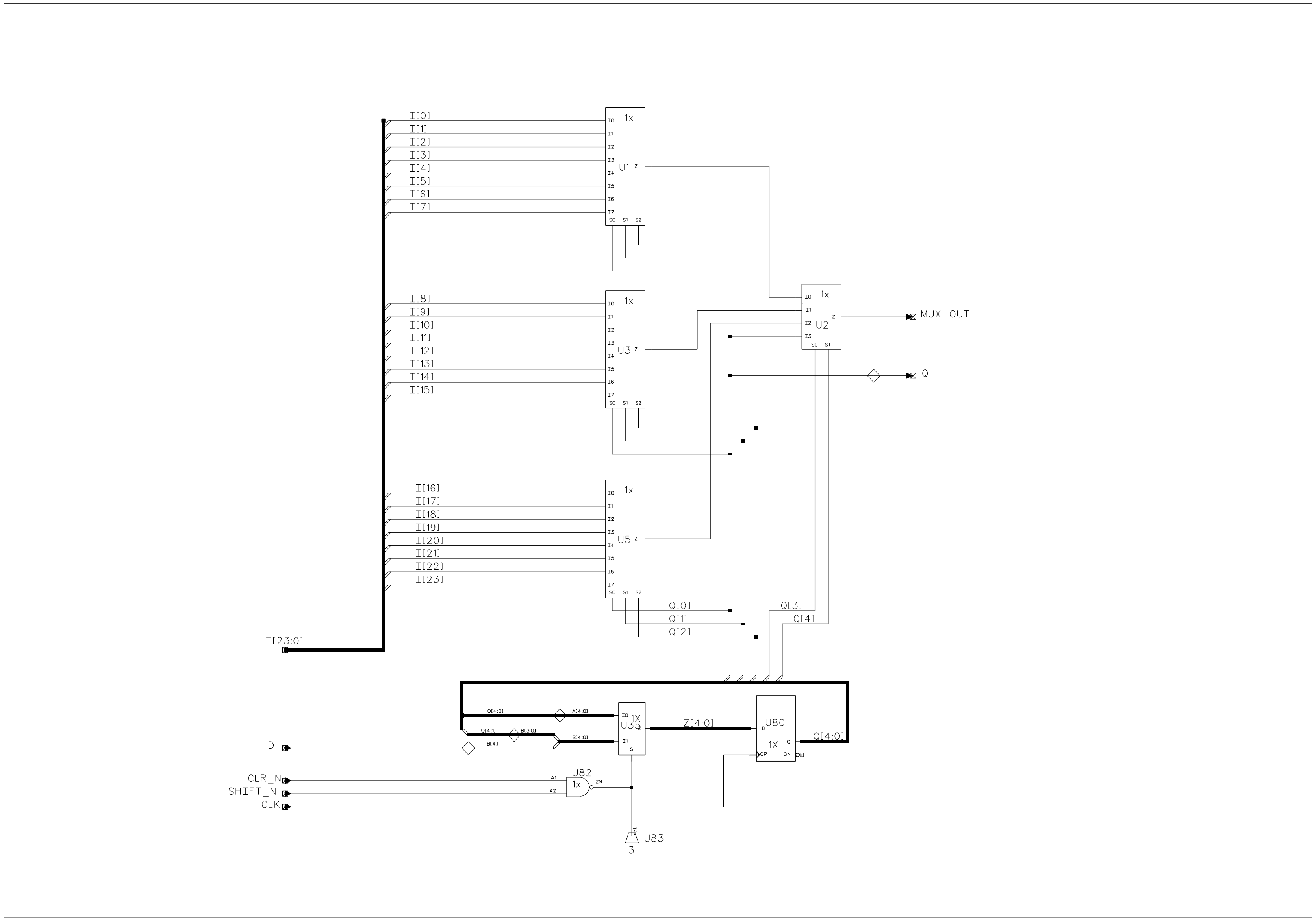
STEPchip

mux29_1.ps

[la]mxr24_1 by nhm on 15-Feb-93 at 4:28 P.M.

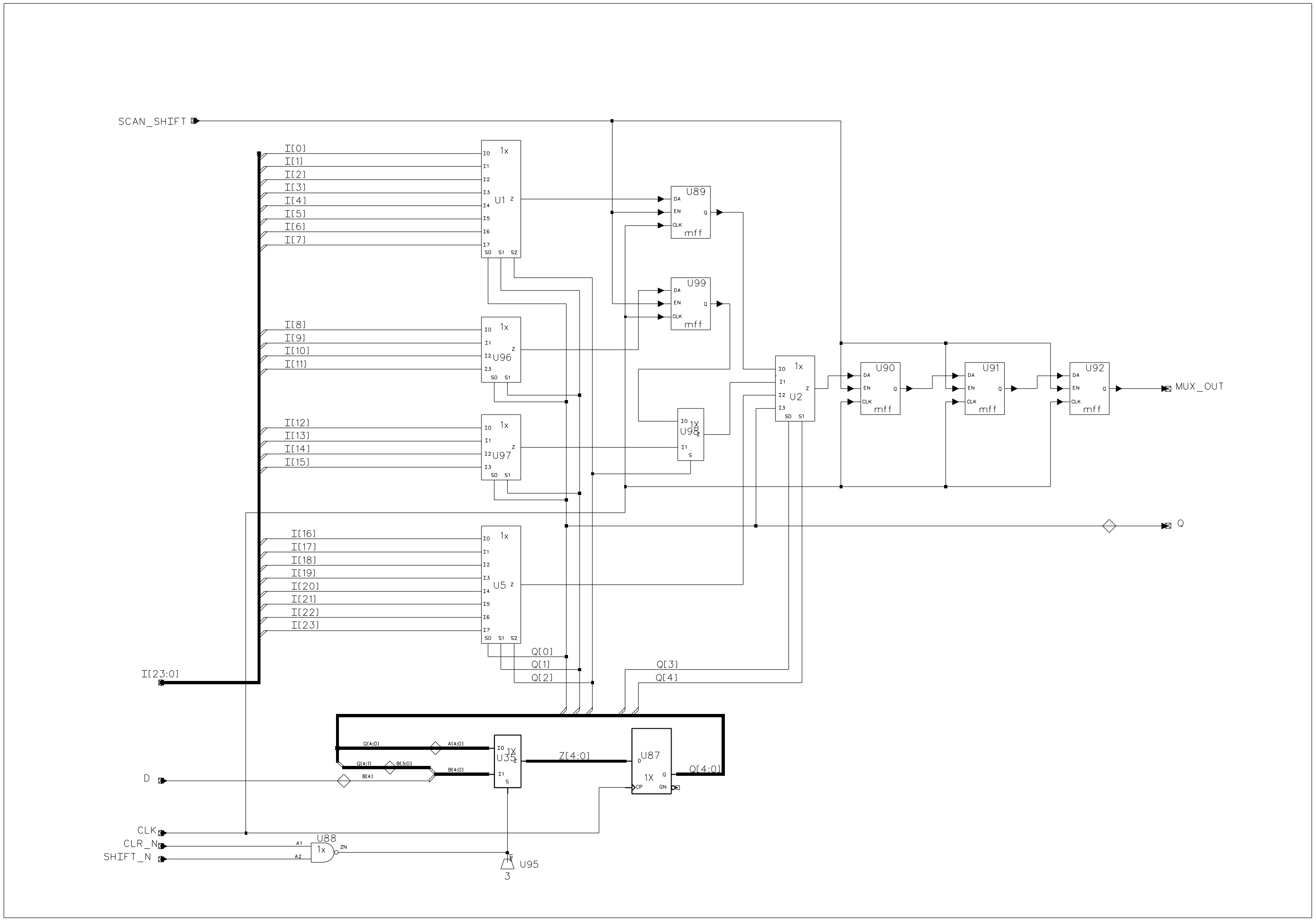


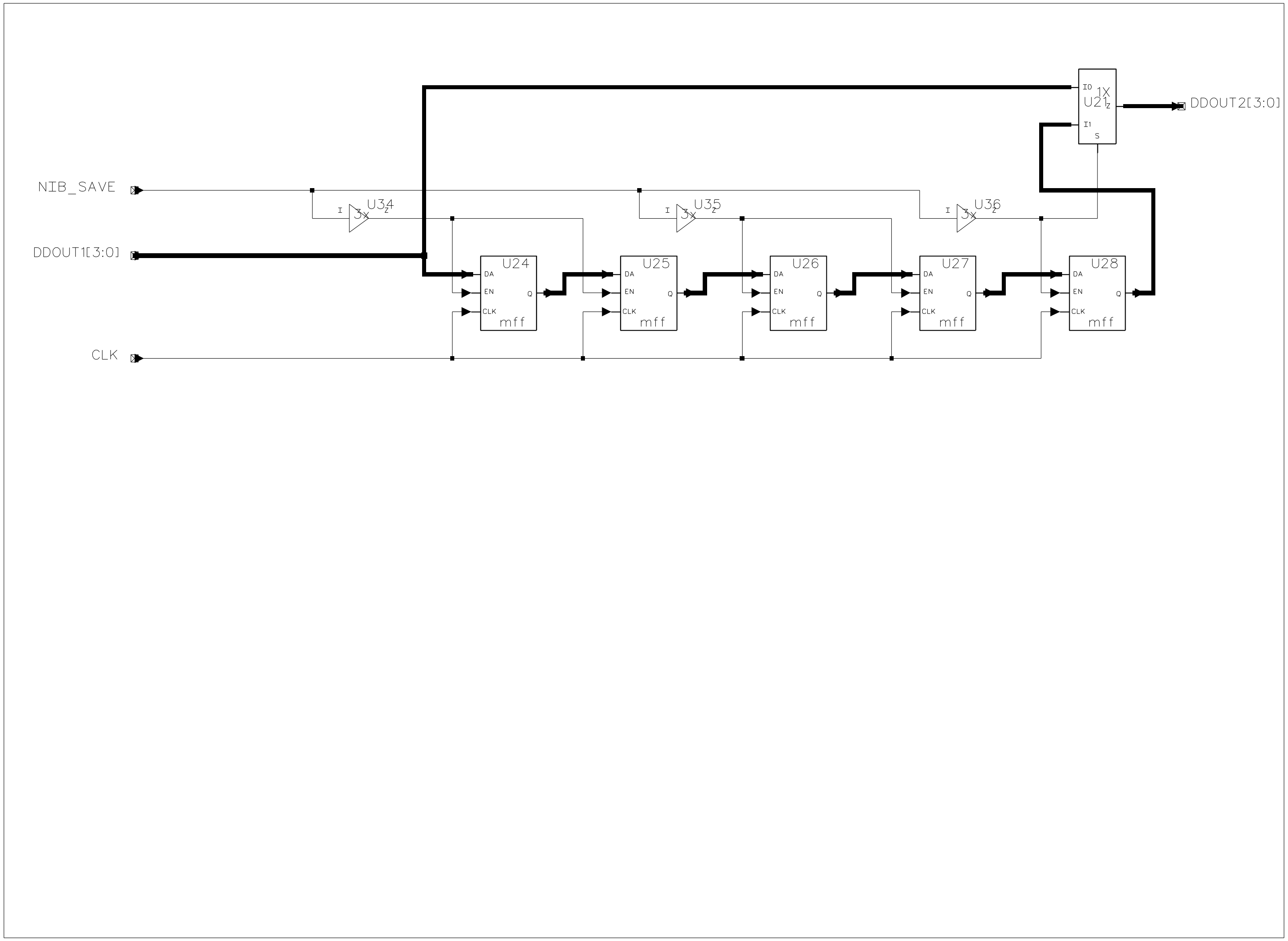
[la]mxr24_1a by nhm on 15-Feb-93 at 4:28 P.M.

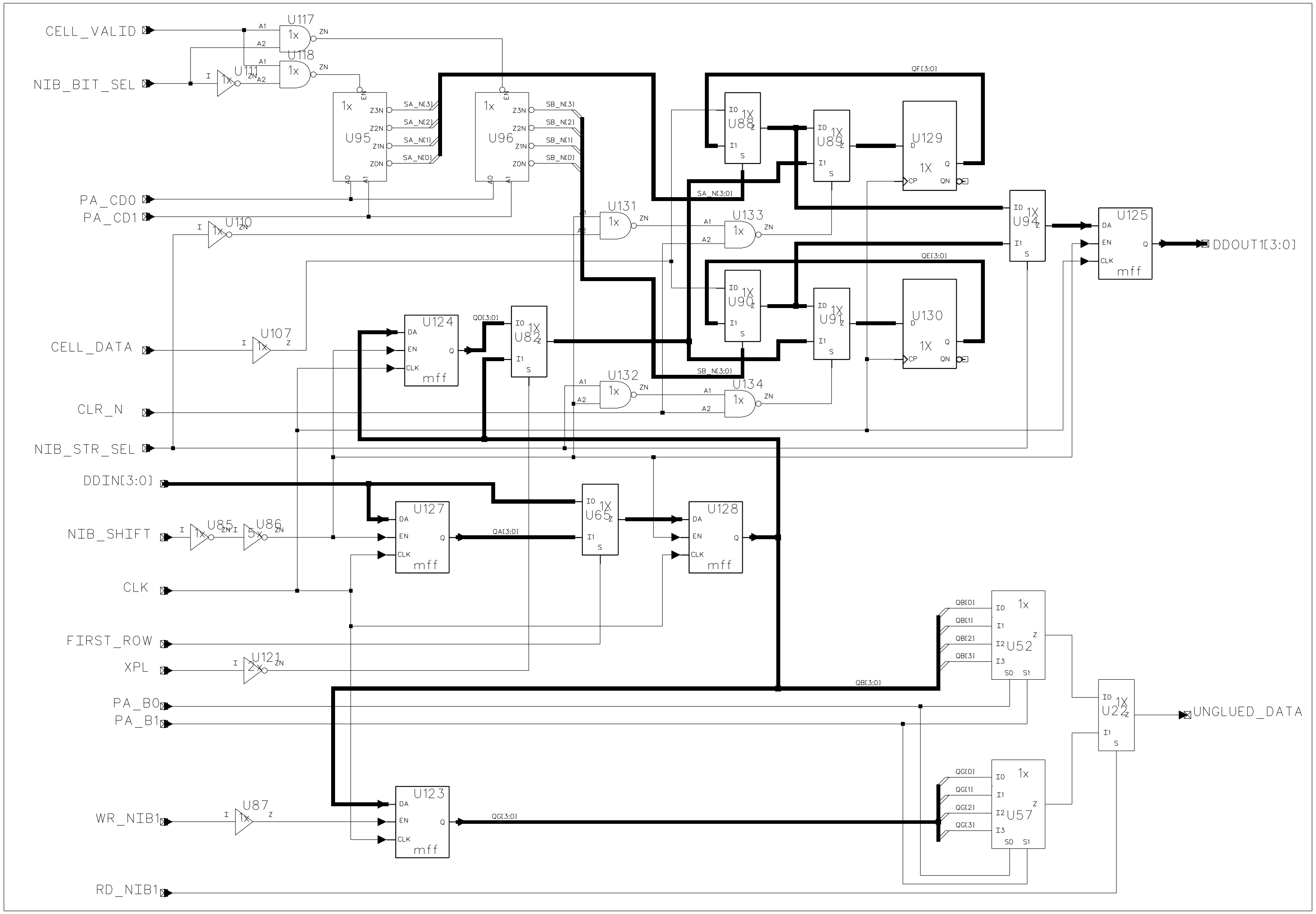


STEPchip

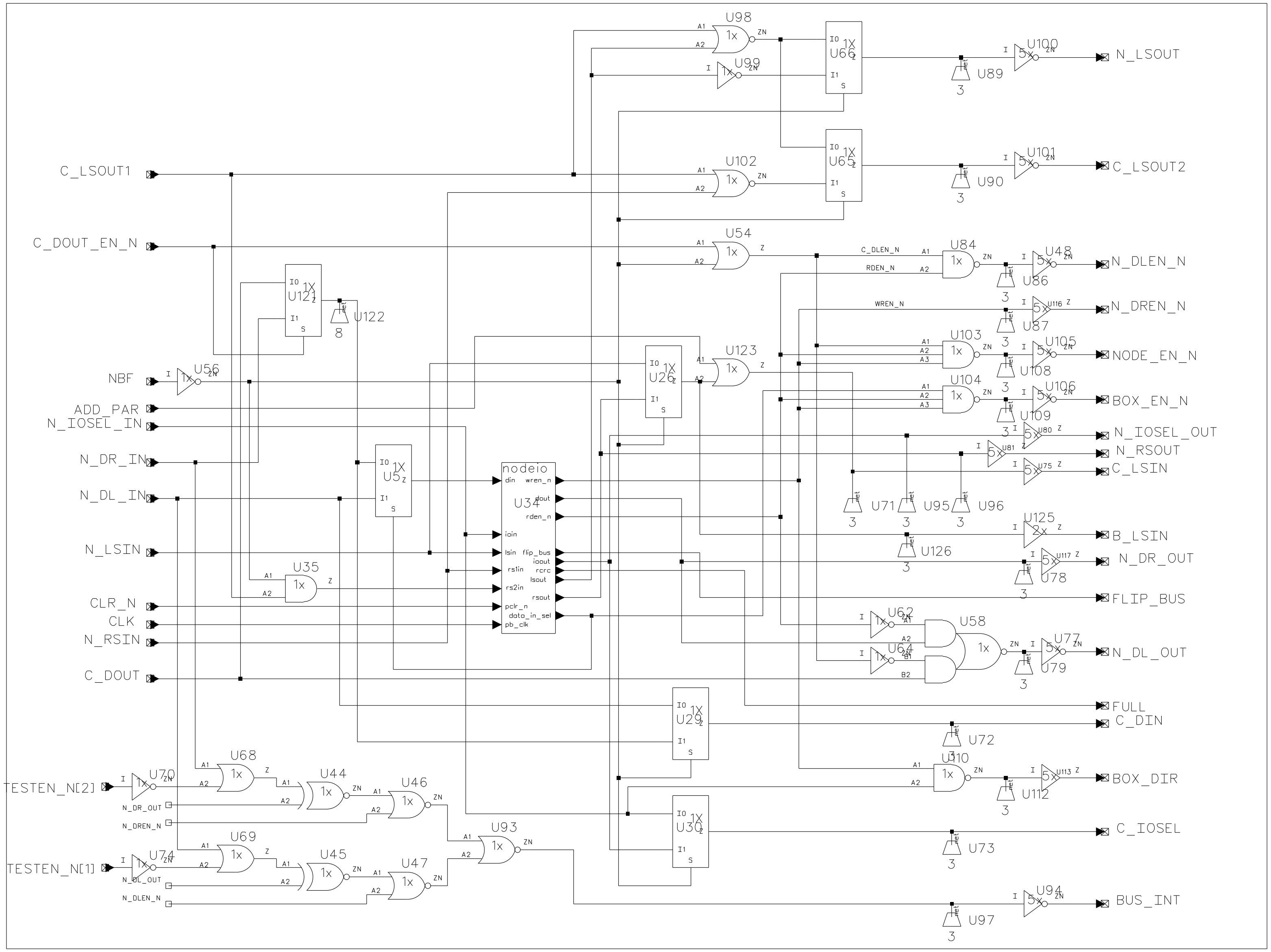
mxr24_1a.ps


SCAN_SHIFT
I[0]
I[1]
I[2]
I[3]
I[4]
I[5]
I[6]
I[7]
U1
I[8]
I[9]
I[10]
I[11]
U96
I[12]
I[13]
I[14]
I[15]
U97
I[16]
I[17]
I[18]
I[19]
I[20]
I[21]
I[22]
I[23]
U5
U89
mff
U99
mff
U98
U2
U90
mff
U91
mff
U92
mff
MUX_OUT
Q
Q[0]
Q[1]
Q[2]
Q[3]
Q[4]
I[23:0]
U35
Z[4:0]
U87
Q[4:0]
D
CLK
CLR_N
SHIFT_N
U88
U95
3


NIB_SAVE
DDOUT1[3:0]
CLK
DDOUT2[3:0]
U21
1X
I0
I1
S
Z
U34
U35
U36
3x
I
Z
U24
U25
U26
U27
U28
DA
EN
CLK
Q
mff



CELL_VALID
NIB_BIT_SEL
PA_CD0
PA_CD1
CELL_DATA
CLR_N
NIB_STR_SEL
DDIN[3:0]
NIB_SHIFT
CLK
FIRST_ROW
XPL
PA_B0
PA_B1
WR_NIB1
RD_NIB1
DDOUT1[3:0]
UNGLUED_DATA
U117
U118
U111
U110
U95
U96
U131
U132
U133
U134
U107
U124
U82
U88
U89
U90
U91
U129
U130
U94
U125
U127
U65
U128
U85
U86
U121
U52
U57
U22
U87
U123
SA_N[3:0]
SB_N[3:0]
QF[3:0]
QE[3:0]
QD[3:0]
QA[3:0]
QB[3:0]
QC[3:0]
mff

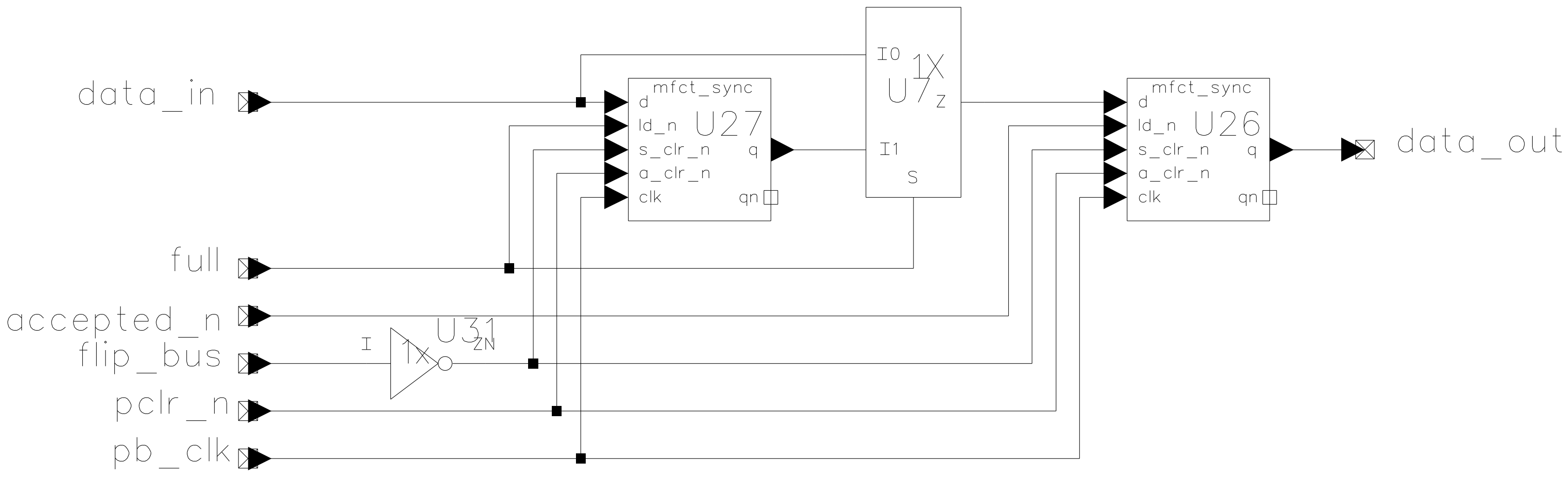


STEPchip

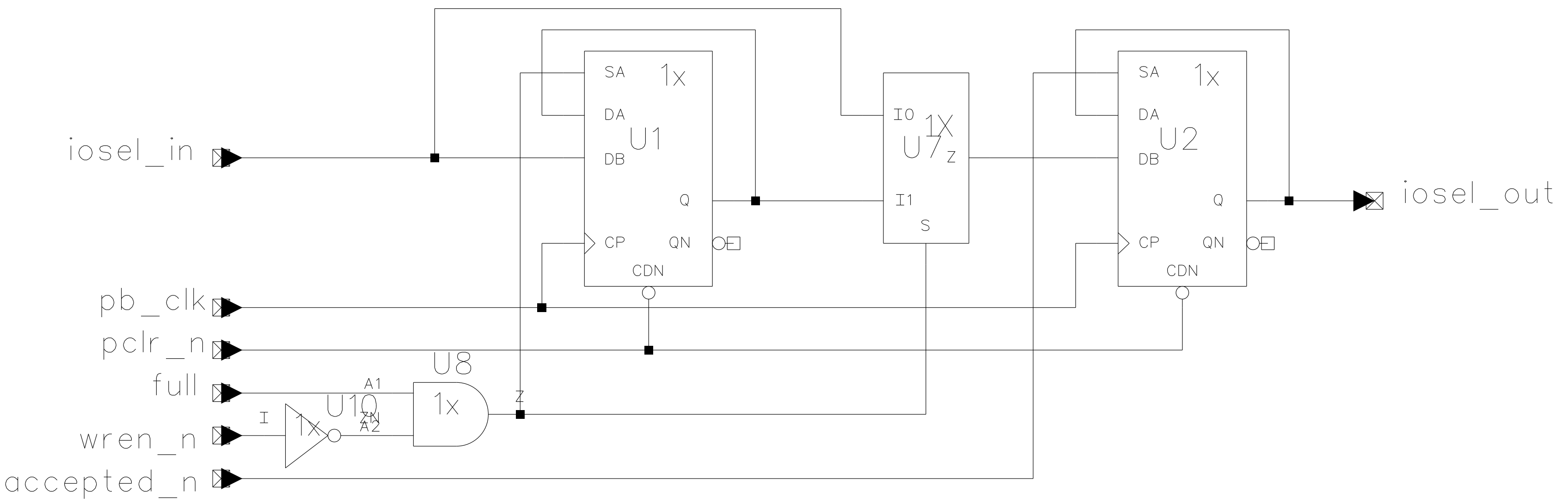
node.ps

[la]node_data by nhm on 15-Feb-93 at 4:29 P.M.

node_data.ps



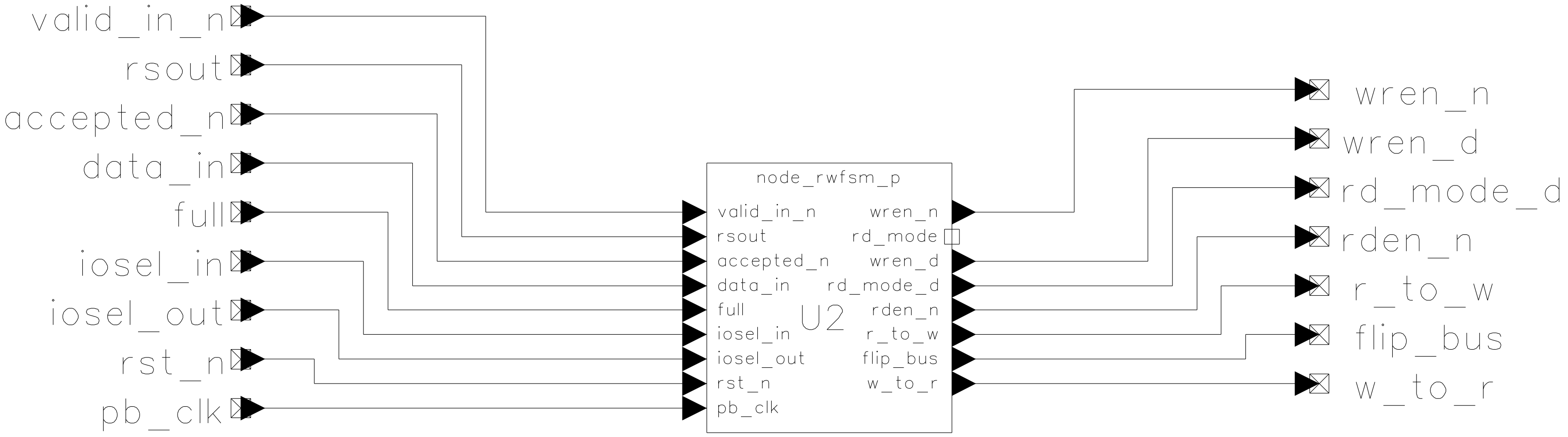
STEPchip



The gate prevents iosel_out from hanging up high
in read mode if no data is available.

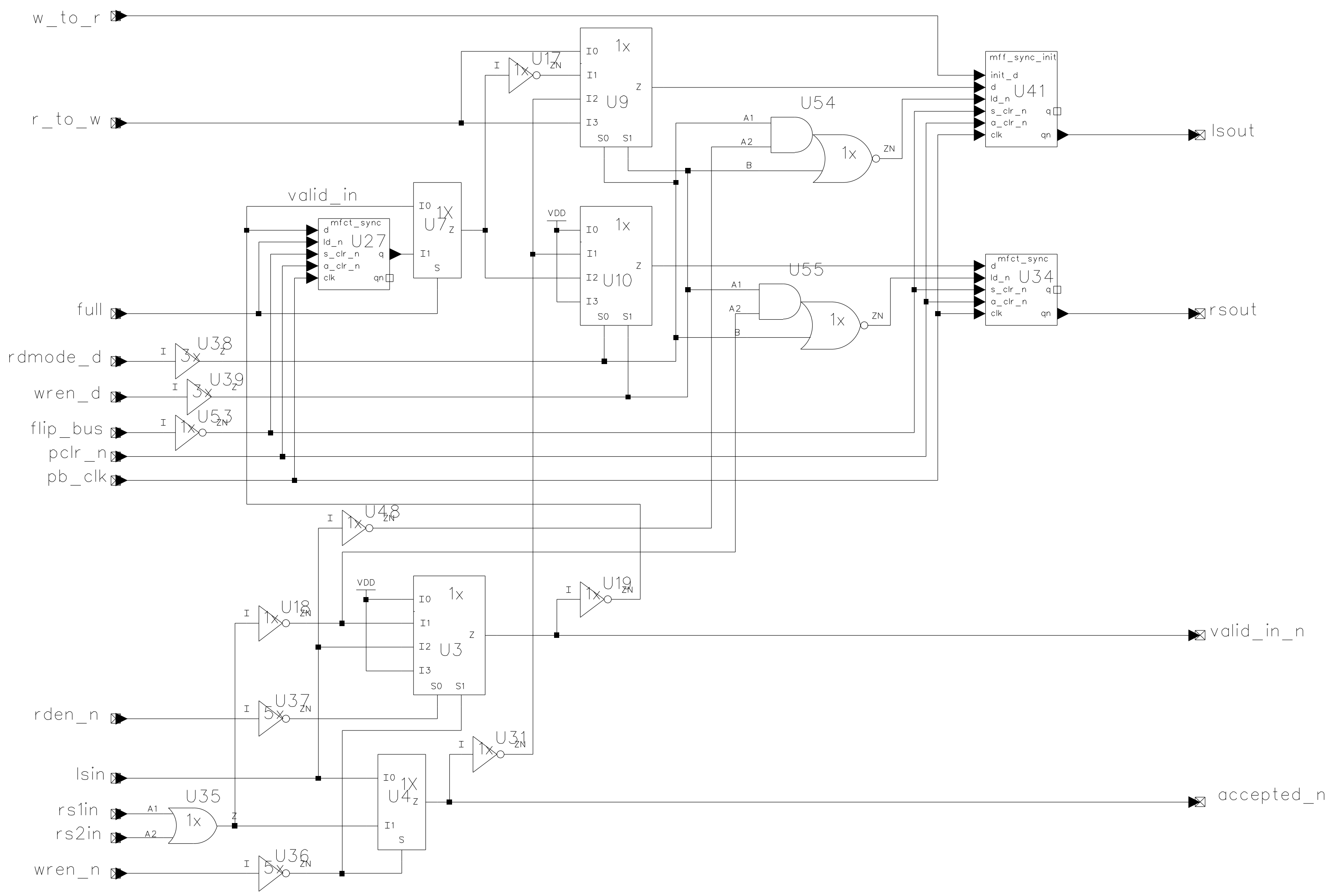
```
#cell2 * node_rwfsm sm * 6 any 1024 v8r2.2
# "5-Feb-92 GMT" "19:52:28 GMT" "5-Feb-92 GMT" "19:52:28 GMT" faust * .
sm node_rwfsm;

# Finite State Machine file for the Bus direction logic of a PBus node
# AUTHOR : David Harnanan
# FILENAME : /im/cam8/interface/node_rwfsm.sm
# DATE : 11 September 1990       REVISED : 2/5/92

# 12/11/90 faust
#    Modified to make read and write enables low true.
#    Also changed signal names:
#      rst    to rst_n
#      /chg   to chg_n
#      /rstnd to rstnd_n

# 1/8/91 faust
#    fixed bad equation for rstnd_n output

# 1/21/91 faust
#    made rd_mode wake up default to 0

# 1/30/92 faust
#    Added outputs rd_mode_d and wren_d; these are just the
#    d inputs to rd_mode and wren.

# 2/5/92 faust
#    Changed name to from rwfsm to node_rwfsm.
#    Changed signal names:
#       inv to rsout
#       invin to valid_in_n
#       fr to accepted_n
#       f to full
#       ioin to iosel_in
#       iod to iosel_out
#       chg_n to w_to_r
#       rstnd_n to flip_bus

clock pb_clk 25;

reset !rst_n --> wr0chk;

inputs
        iosel_out       # iodata
        iosel_in        # iosel input
        full            # node contains 2 data bits
        data_in         # read/write bit
        accepted_n      # receiving node ready (active low)
        rsout           # handshake signal to next node
        valid_in_n;        # valid input data (active low)

outputs
        w_to_r          # change from wr -> rd
        flip_bus        # turn bus around
        r_to_w          # change direction rd -> wr
        rden_n          # read tristate enable (active low)
        rd_mode_d       # read mode d input
        wren_d;         # write enable d input
```

```
rs=0 outputs
        rd_mode;        # read mode

rs=1 outputs
        wren_n;         # write tristate enable,

let iolh = iosel_in & !iosel_out;       # low to high iosel edge
let iohl = !iosel_in & iosel_out;       # high to low iosel edge

# gate read enable directly with iosel_in to disable bus faster
let rden_n = !(rd_mode & iosel_in);

# look for opcode bit or rising edge of iosel; we should never
#   see a parity bit while in this state.
state wr0chk
      {let wren_n=0;
        let wren_d=1;}
      !iosel_in & !valid_in_n & !full --> write,
      iosel_in & !valid_in_n --> wait_parity,      # only parity bit xmted ...
             --> wr0chk;

# got at least one opcode bit, now stay in write mode until we see
#   a read bit (i.e., the last data bit before rising edge of iosel is 0)
state write
      !full & !data_in & iolh & accepted_n & !rsout --> wtfull wren_d=1,
      !full & !data_in & iolh & !accepted_n & !rsout --> wr_rd wren_n=1 rd_mode_d=1 w_to_r=1 flip_bus=1,
             --> write wren_d=1;

# Writing to register 0, now wait for iosel to drop, then return to wr0chk
state wait_parity
      !full & !iosel_in & !valid_in_n --> wr0chk wren_d=1,
             --> wait_parity wren_d=1;

# Wait for the right node to accept the read bit
state wtfull
      !accepted_n --> wr_rd wren_n=1 rd_mode_d=1 w_to_r=1 flip_bus=1,
          --> wtfull wren_d=1;

# Transition from write to read
state wr_rd
      --> read rd_mode=1 rd_mode_d=1 w_to_r=1 flip_bus=1;

# In read mode, wait for falling edge of iosel
state read
      iohl  --> rd_wr rd_mode=0 flip_bus=1,
            --> read rd_mode_d=1;

# Transition from read to write
state rd_wr
        {let r_to_w = 1;}
       --> wait_parity wren_n=0 wren_d=1;

end
```


valid_in_n
rsout
accepted_n
data_in
full
iosel_in
iosel_out
rst_n
pb_clk
node_rwfsm_p
U2
valid_in_n
rsout
accepted_n
data_in
full
iosel_in
iosel_out
rst_n
pb_clk
wren_n
rd_mode
wren_d
rd_mode_d
rden_n
r_to_w
flip_bus
w_to_r
wren_n
wren_d
rd_mode_d
rden_n
r_to_w
flip_bus
w_to_r

STEPchip

[la]node_status by nhm on 15-Feb-93 at 4:29 P.M.

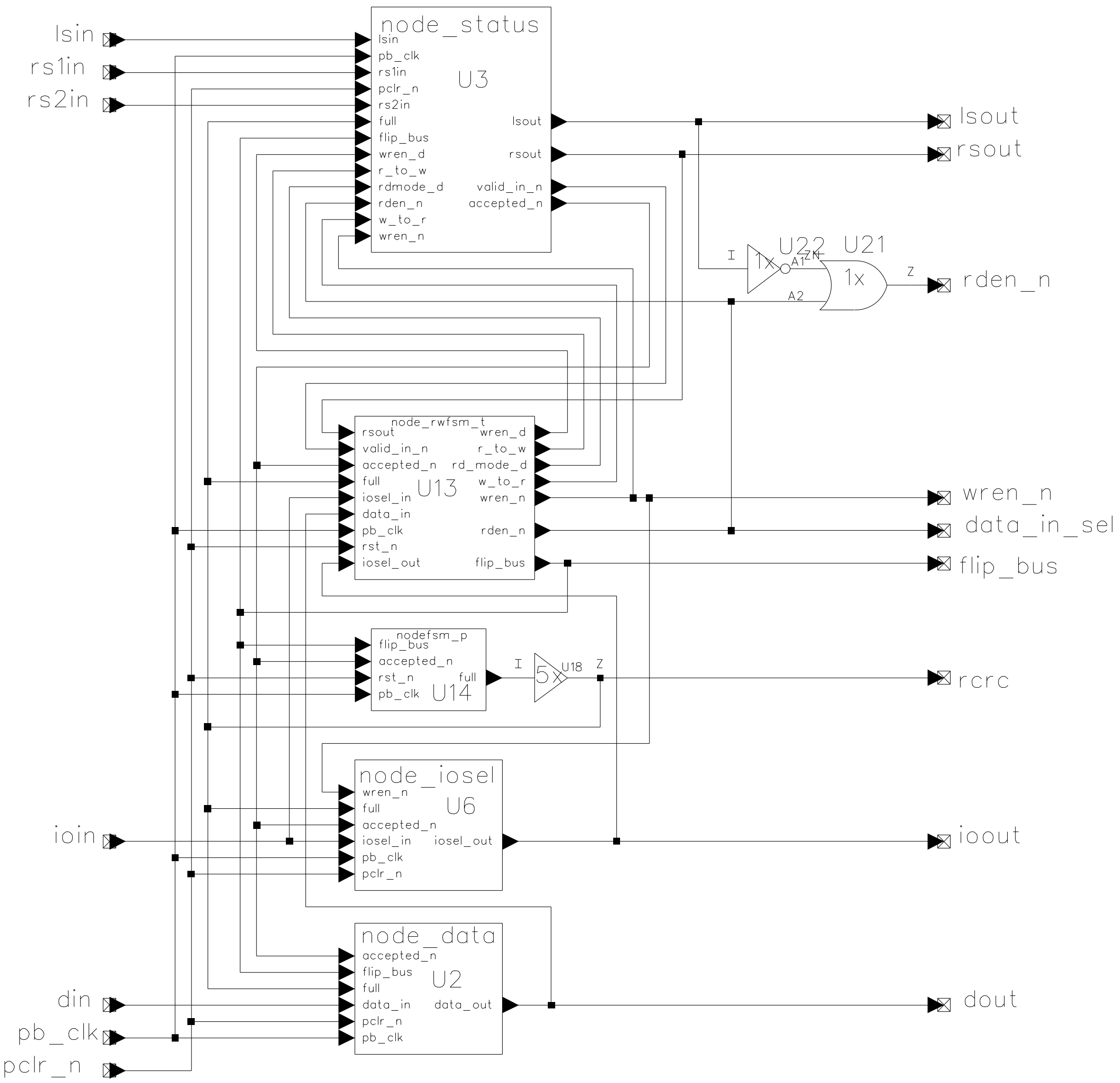


STEPchip

node_status.ps

```
#cell2 * nodefsm sm * 19 any 0 v8r2.2
# "5-Feb-92 GMT" "19:31:01 GMT" "5-Feb-92 GMT" "19:31:01 GMT" faust * .
sm nodefsm;

# Finite State Machine file for each PBus node
# AUTHOR : David Harnanan
# FILENAME : /im/cam8/interface/nodefsm.sm
# DATE : 10 September 1990      REVISED : 2/5/92

# revised 1/21/91 faust
#   added rstnd_n input--it performs a synchronous
#   reset on the fsm (low true)

# revised 1/30/92 faust
#   changed to not generate full output during rstnd_n low

# revised 2/5/92 faust
#   changed signal names:
#       rst to rst_n
#       rstnd_n to flip_bus
#       rsin to accepted_n
#   changed state names:
#       two to hold
#       one to load

clock pb_clk 25;

reset !rst_n --> hold;

inputs  accepted_n      # right status in
        flip_bus;       # change from rd to wr, or wr to rd.

outputs full;           # recirculate data within node.

state hold              # Two data bits in node
      {let full = 1;}
      flip_bus --> hold,
      !accepted_n --> load,
            --> hold;

state load              # One data bit in node
      flip_bus --> hold,
      accepted_n --> hold,
           --> load;

end
```

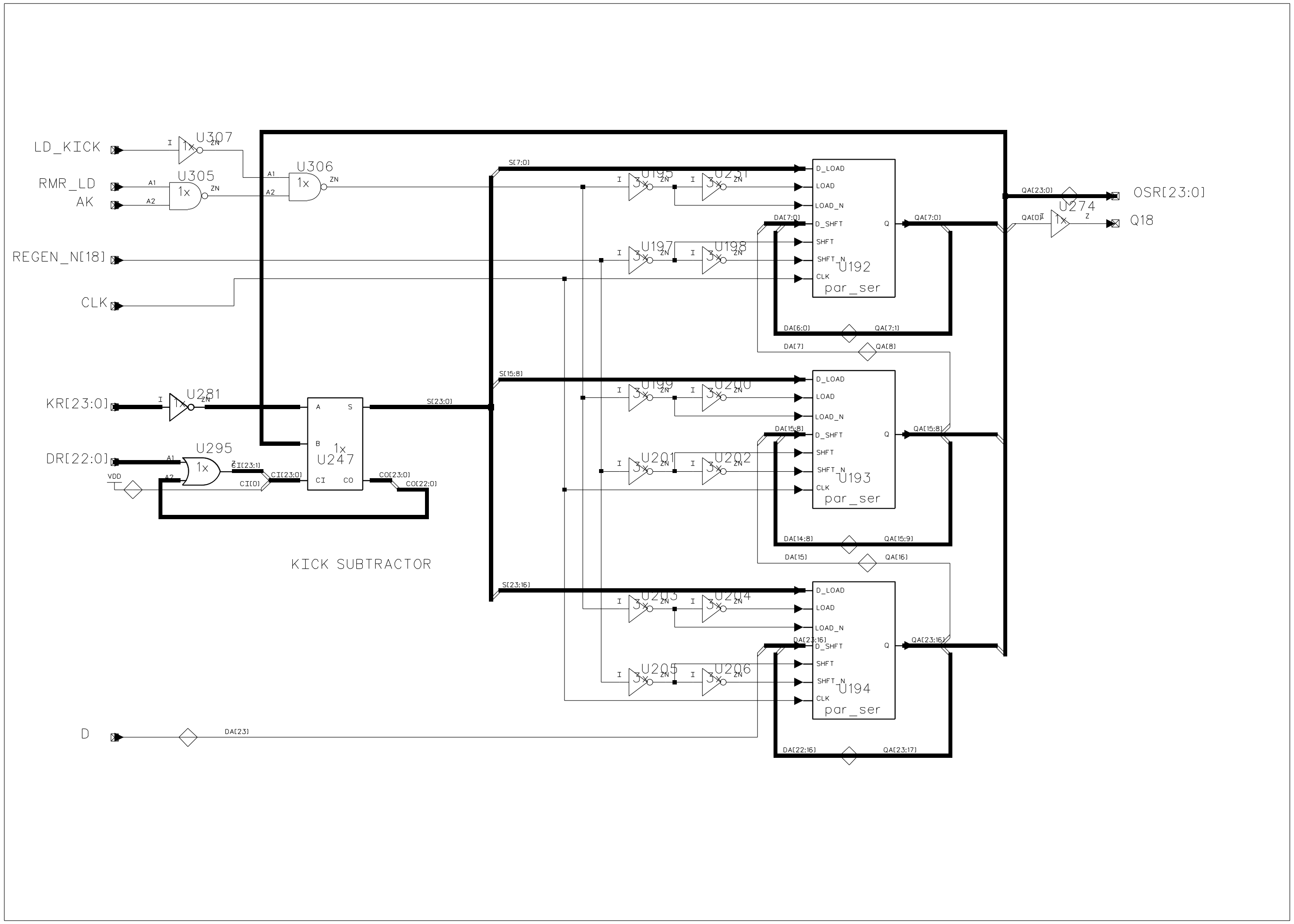
[la]nodeio by nhm on 15-Feb-93 at 4:29 P.M.

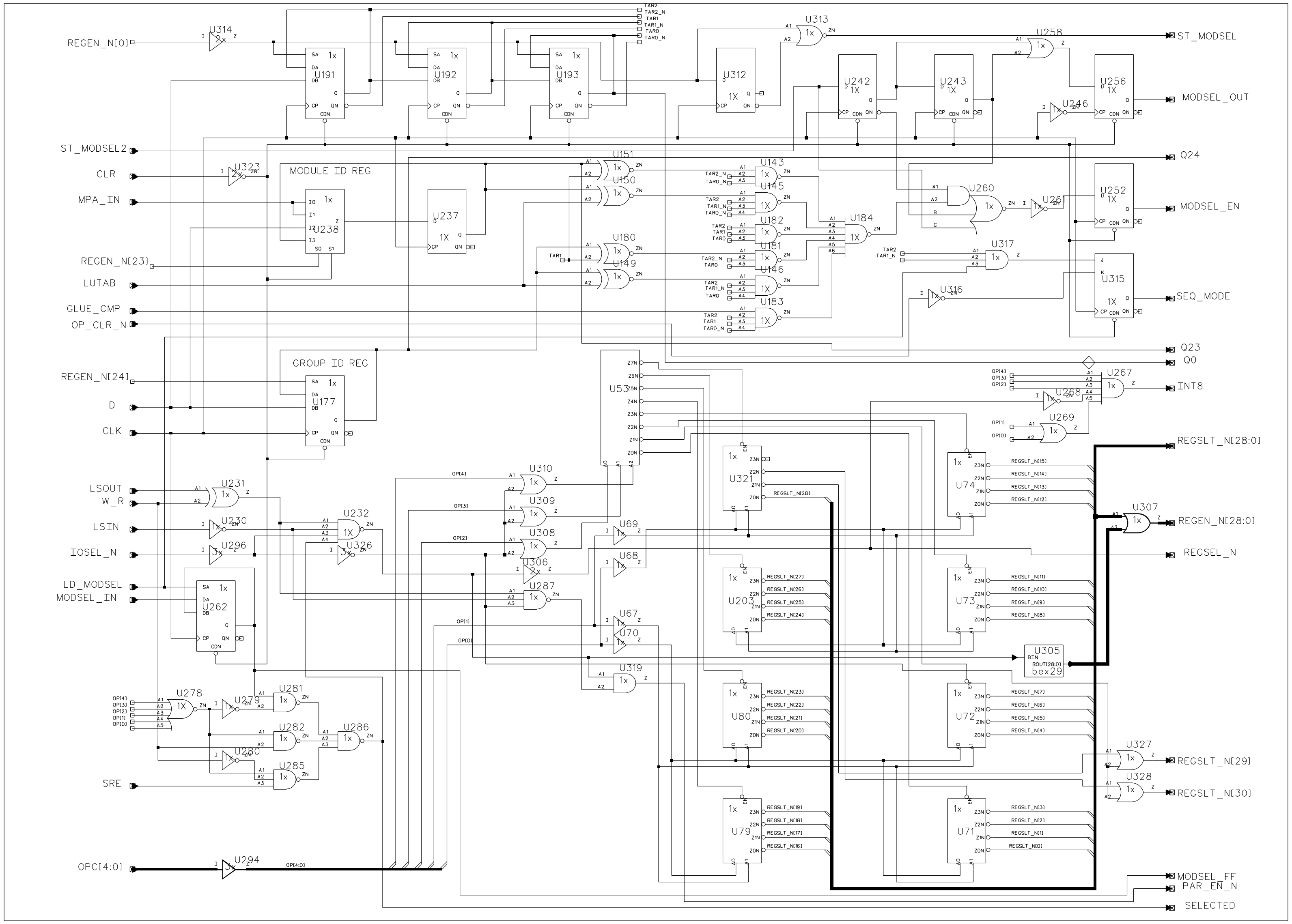


B-65

STEPchip

nodeio.ps

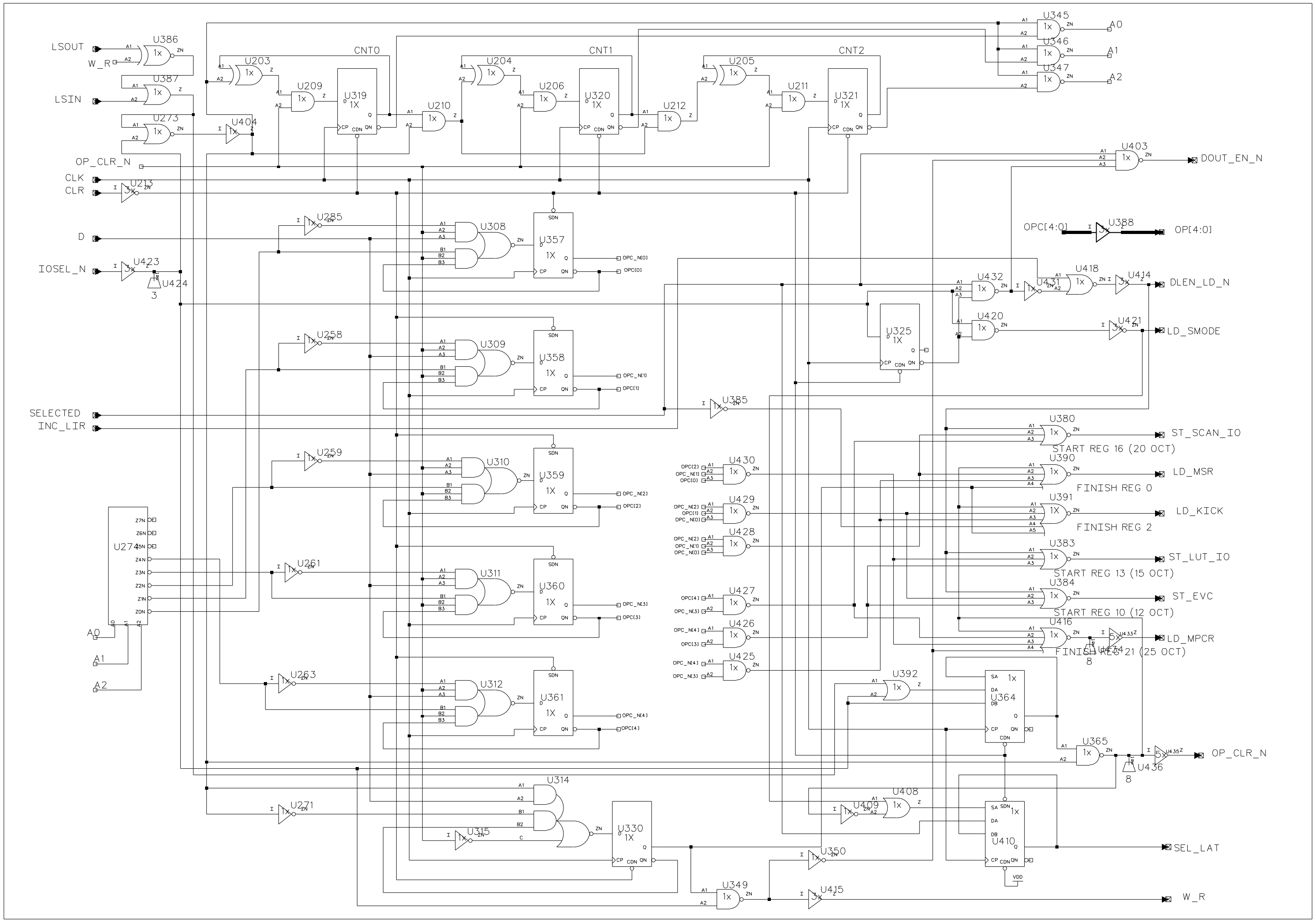

LD_KICK
RMR_LD
AK
REGEN_N[18]
CLK
KR[23:0]
DR[22:0]
D
U307
U305
U306
U281
U295
U247
U192
U193
U194
par_ser
U274
OSR[23:0]
Q18
KICK SUBTRACTOR



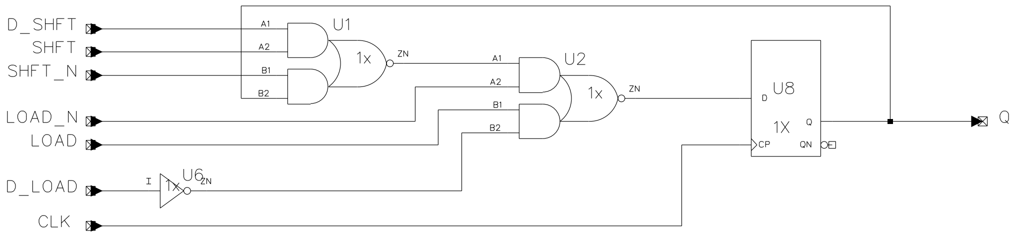
STEPchip

opc_dcd.ps

[la]opc_reg by nhm on 15-Feb-93 at 4:30 P.M.

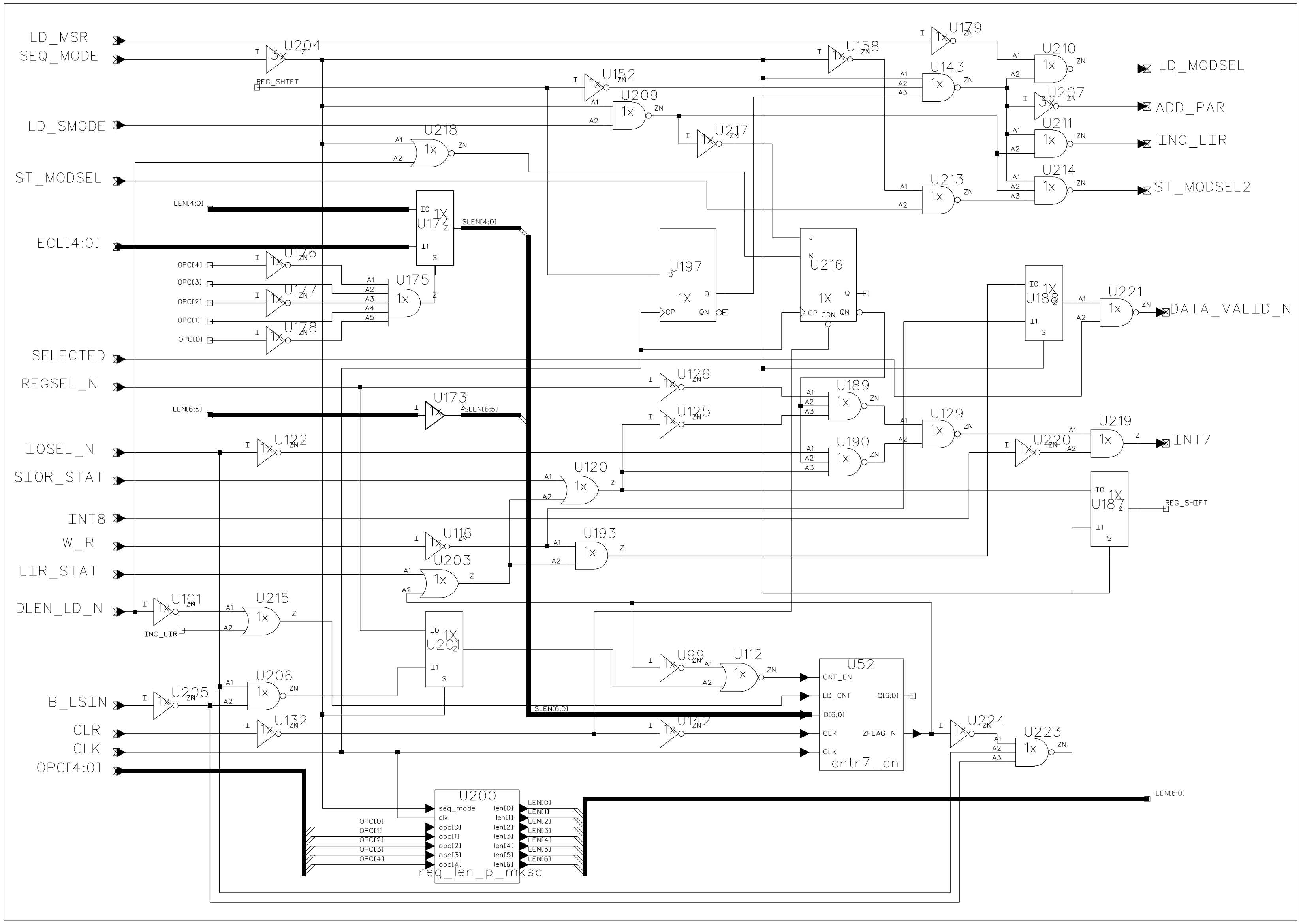


[la]par_ser by nhm on 15-Feb-93 at 4:30 P.M.



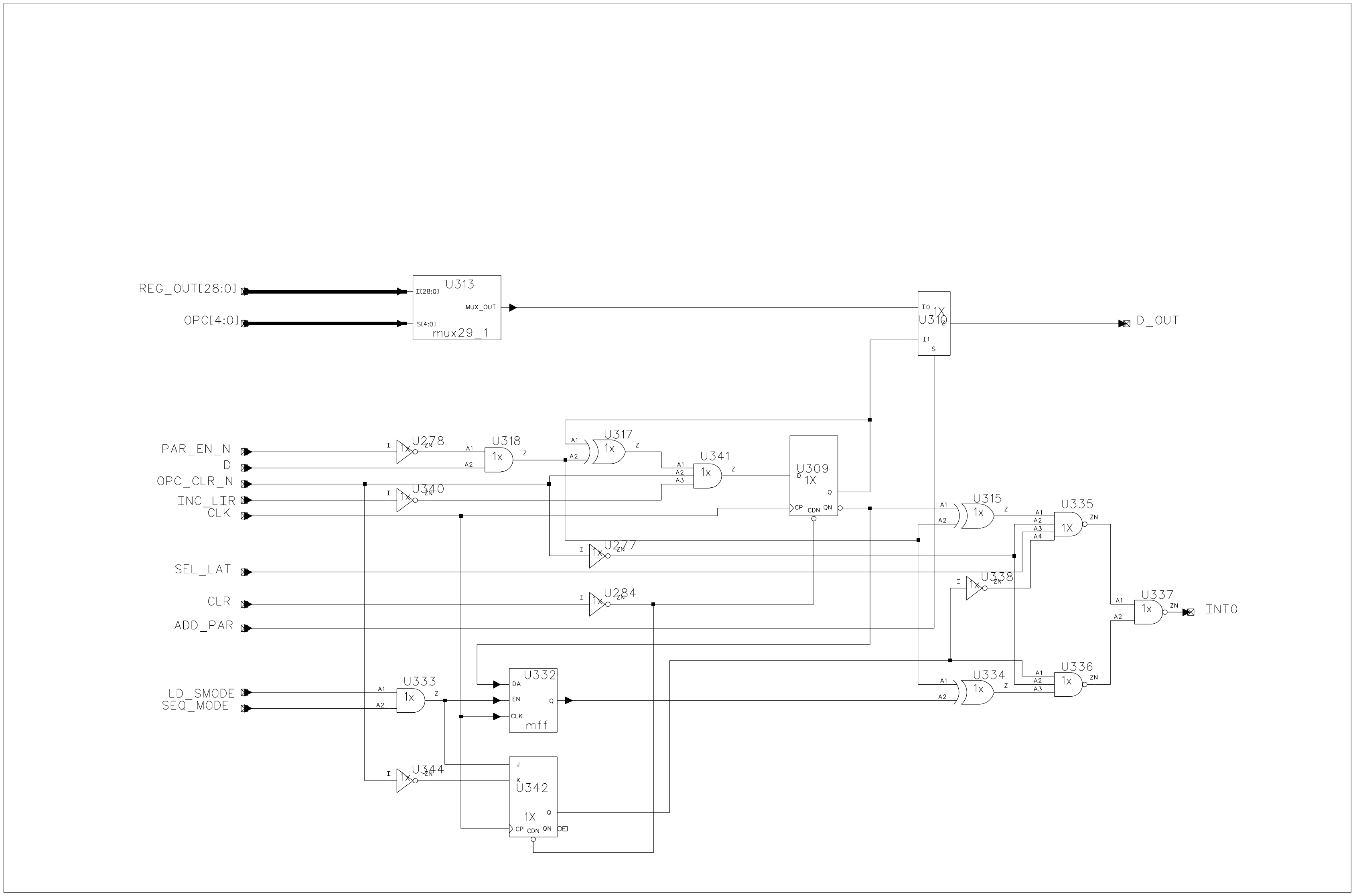
STEPchip

par_ser.ps


LD_MSR
SEQ_MODE
LD_SMODE
ST_MODSEL
ECL[4:0]
SELECTED
REGSEL_N
IOSEL_N
SIOR_STAT
INT8
W_R
LIR_STAT
DLEN_LD_N
B_LSIN
CLR
CLK
OPC[4:0]
LD_MODSEL
ADD_PAR
INC_LIR
ST_MODSEL2
DATA_VALID_N
INT7
REG_SHIFT
INC_LIR
LEN[4:0]
LEN[6:5]
LEN[6:0]
SLEN[4:0]
SLEN[6:5]
SLEN[6:0]
U204
U179
U158
U143
U210
U152
U209
U207
U211
U217
U218
U214
U213
U174
U176
U175
U177
U178
U197
U216
U188
U221
U126
U173
U189
U125
U129
U219
U122
U190
U220
U120
U187
U116
U193
U203
U101
U215
U201
U99
U112
U52
U206
U205
U132
U142
U224
U223
U200
cntr7_dn
reg_len_p_mksc

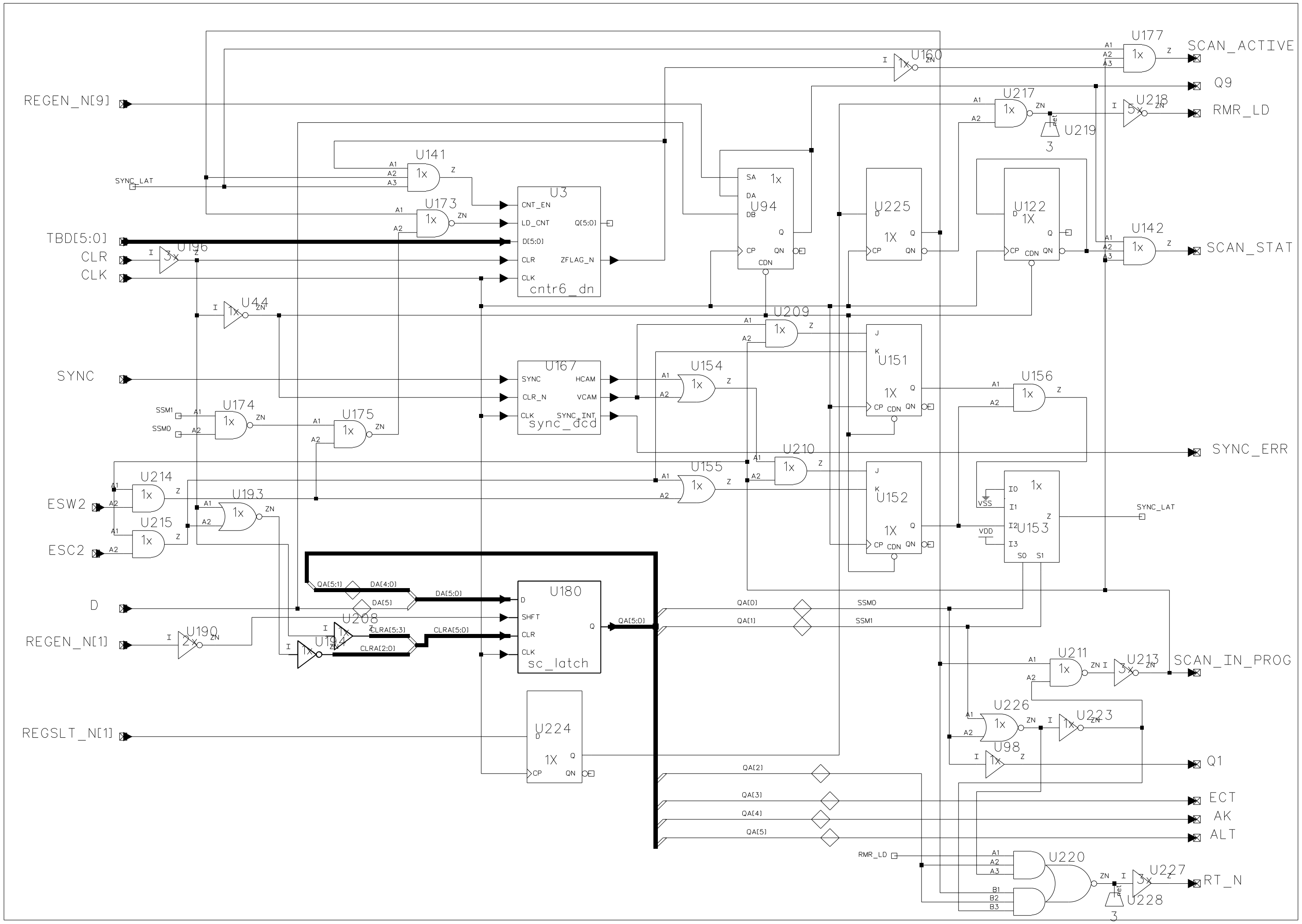
STEPchip

```
#cell2 * reg_len sm * 4 any 0 v8r1.8
# "12-Apr-91 GMT" "19:19:52 GMT" "12-Apr-91 GMT" "19:19:52 GMT" td * .
sm reg_len1;

clock clk 25;

inputs seq_mode opc[4:0];

outputs len[6:0];

state conditions

 opc = 'd0  --> len = 'd3,
 opc = 'd1  --> len = 'd6,
 opc = 'd2  --> len = 'd30,
 opc = 'd3  --> len = 'd5,
 opc = 'd4  --> len = 'd6,
 opc = 'd5  --> len = 'd6,
 opc = 'd6  --> len = 'd6,
 opc = 'd7  --> len = 'd6,
 opc = 'd8  --> len = 'd6,
 !seq_mode & (opc = 'd9)  --> len = 'd1,
  seq_mode & (opc = 'd9)  --> len = 'd4,
 opc = 'd10 --> len = 'd0,
 opc = 'd11 --> len = 'd16,
 opc = 'd12 --> len = 'd5,
 opc = 'd13 --> len = 'd0,
 opc = 'd14 --> len = 'd24,
 opc = 'd15 --> len = 'd120,
 opc = 'd16 --> len = 'd0,
 opc = 'd17 --> len = 'd35,
 opc = 'd18 --> len = 'd24,
 opc = 'd19 --> len = 'd38,
 opc = 'd20 --> len = 'd16,
 opc = 'd21 --> len = 'd10,
 opc = 'd22 --> len = 'd18,
 !seq_mode & (opc = 'd23) --> len = 'd1,
  seq_mode & (opc = 'd23) --> len = 'd4,
 !seq_mode & (opc = 'd24) --> len = 'd1,
  seq_mode & (opc = 'd24) --> len = 'd4,
 opc = 'd25 --> len = 'd10,
 opc = 'd26 --> len = 'd10,
 !seq_mode & (opc = 'd27) --> len = 'd3,
  seq_mode & (opc = 'd27) --> len = 'd6,
 opc = 'd28 --> len = 'd16,
 opc = 'd29 --> len = 'd0,   # Unused register
 opc = 'd30 --> len = 'd0,   # Unused register
 opc = 'd31 --> len = 'd0;   # Unused register



end
```


REG_OUT[28:0]
OPC[4:0]
U313
I[28:0]
S[4:0]
MUX_OUT
mux29_1
U310
D_OUT
PAR_EN_N
D
OPC_CLR_N
INC_LIR
CLK
SEL_LAT
CLR
ADD_PAR
LD_SMODE
SEQ_MODE
U278
U318
U317
U341
U309
U340
U315
U335
U277
U338
U284
U337
INT0
U333
U332
mff
U334
U336
U344
U342

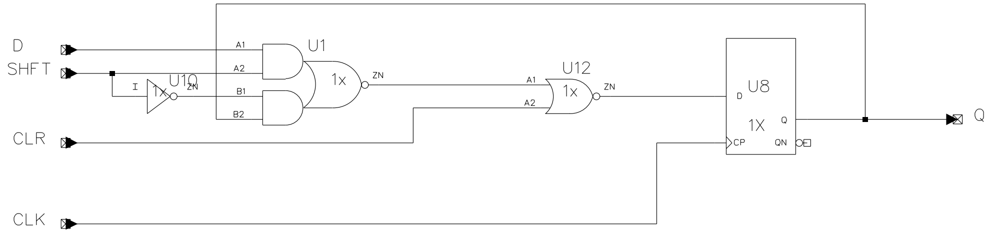
STEPchip

[la]run_reg by nhm on 15-Feb-93 at 4:31 P.M.

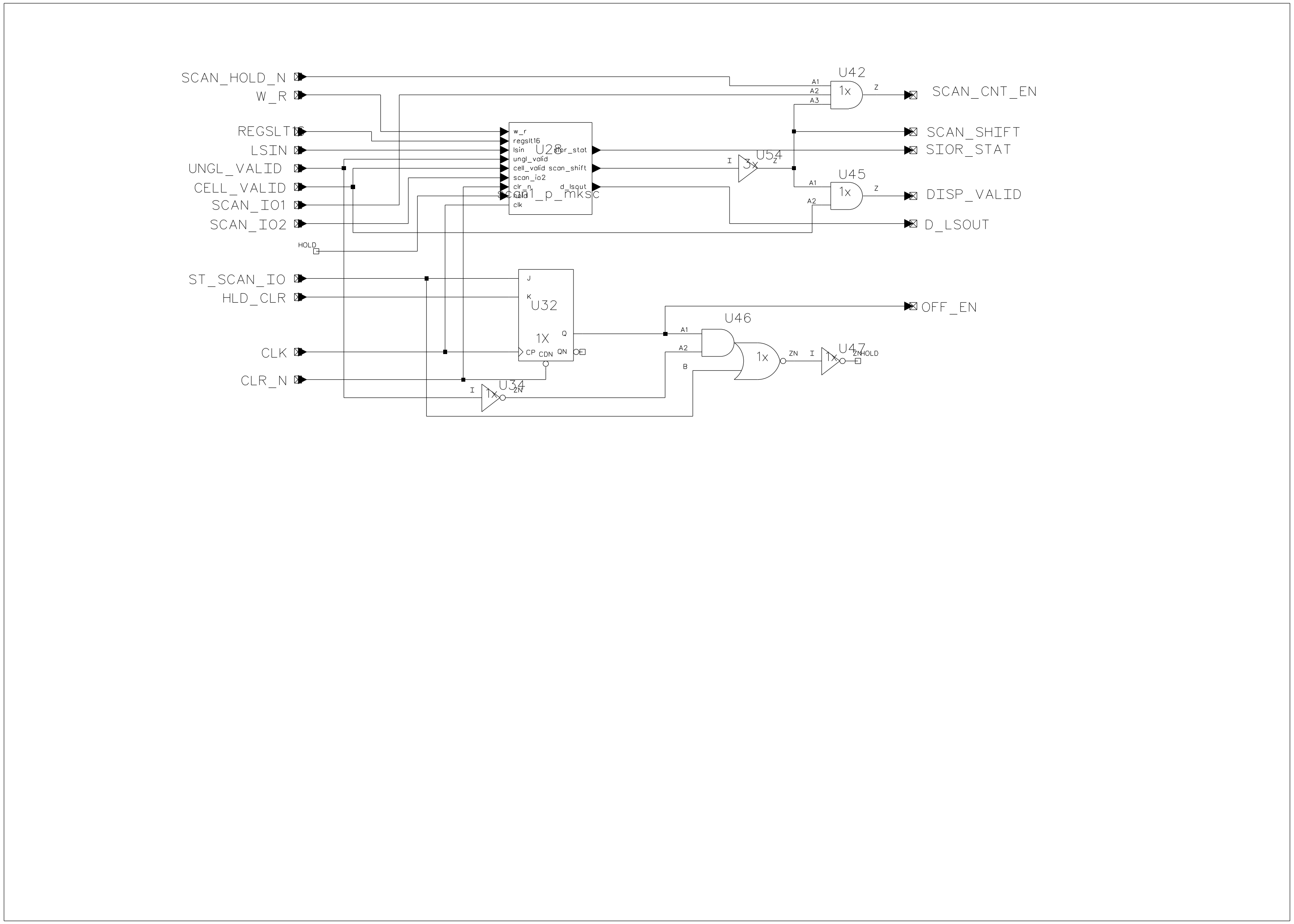


run_reg.ps

[la]sc_latch by nhm on 15-Feb-93 at 4:31 P.M.



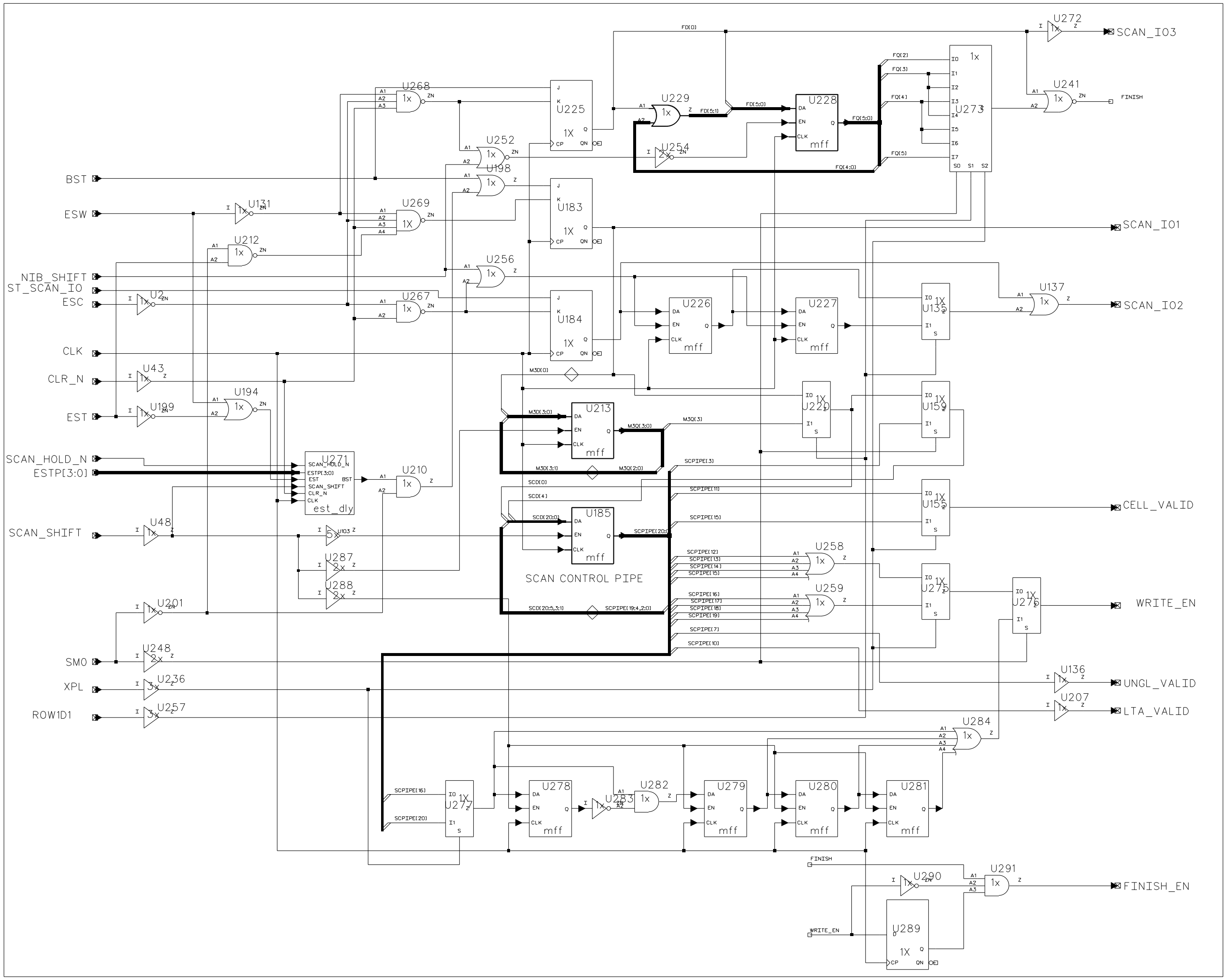
[la]scan by nhm on 15-Feb-93 at 4:31 P.M.



STEPchip

scan.ps

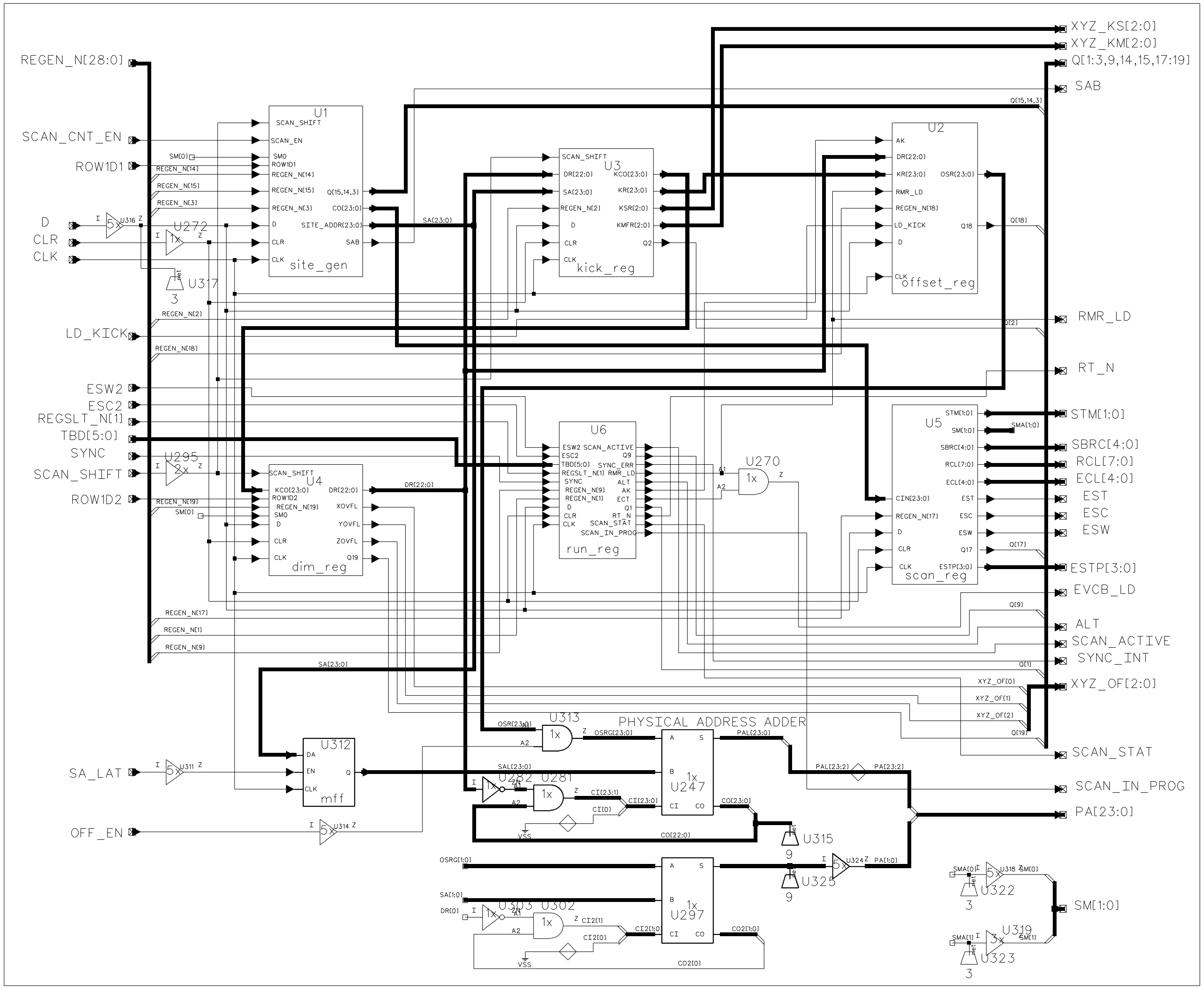
```
#cell2 * scan1 sm * 9 any 1024 v8r1.8
# "14-May-91 GMT" "18:48:02 GMT" "14-May-91 GMT" "18:48:02 GMT" td * .

sm scan1;

clock clk 25;

inputs
    hold
    clr_n
    scan_io2
    cell_valid
    ungl_valid
    lsin
    regslt16
    w_r;

outputs
    d_lsout
    scan_shift
    sior_stat;


# d_lsout = 1 means ready to accept on reads and not ready on writes
# d_lsout = 0 means ready to accept on writes and not ready on reads

let d_lsout = ( w_r & hold )
            | (regslt16 & !w_r & cell_valid );


#Don't gate (w_r & !lsin) with scan_io1 so pipe flushes when scan_io1=0

let scan_shift  = (!regslt16 )
           | (!clr_n)
           |(regslt16 & w_r & !ungl_valid)
           |(regslt16 & w_r & ungl_valid & !lsin)
           |(regslt16 & !w_r & !cell_valid )
           |(regslt16 & !w_r & cell_valid & !lsin);

let sior_stat = (scan_io2)
           | (regslt16);


end
```



STEPchip

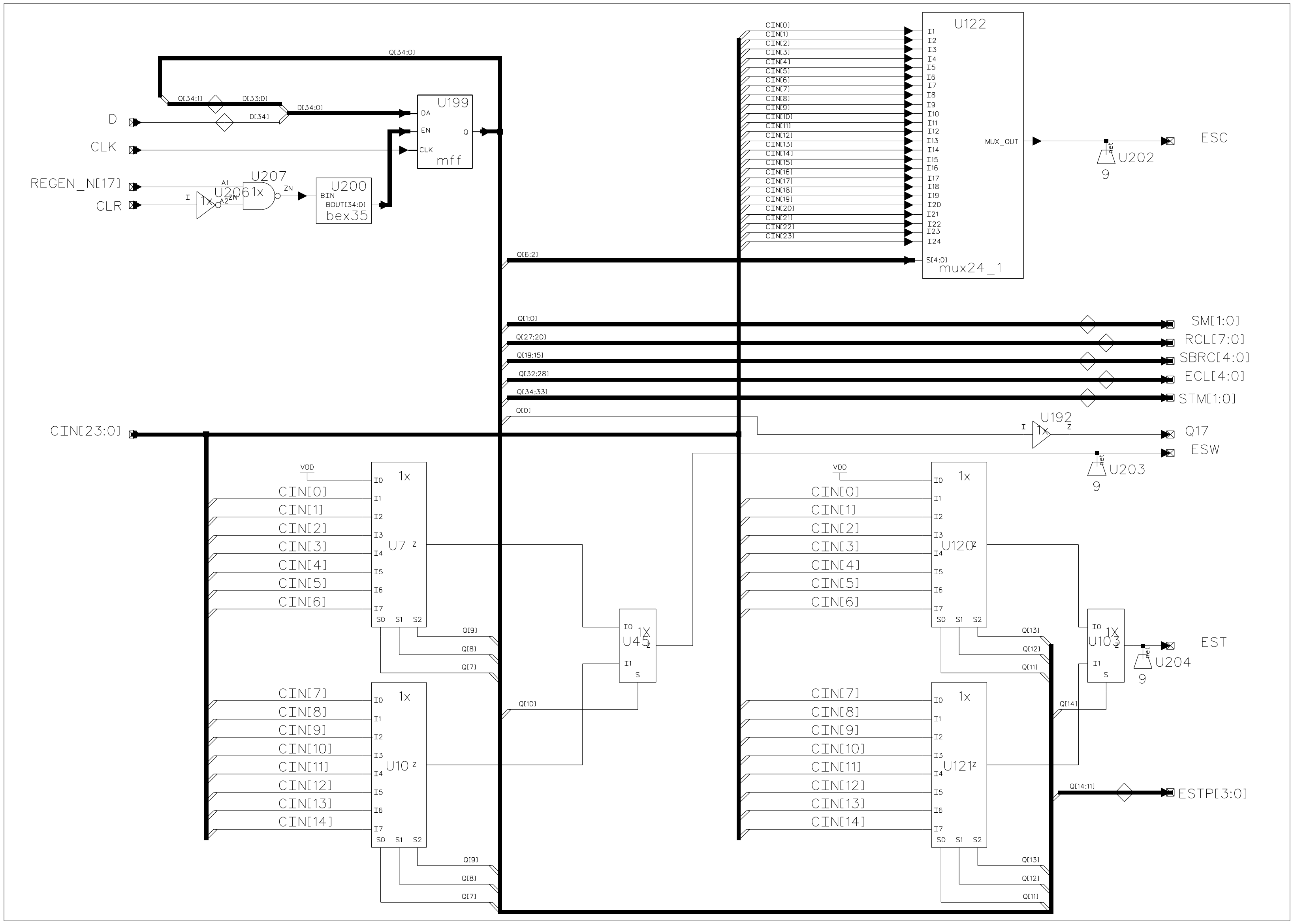
scan_dly.ps

[la]scan_gen by nhm on 15-Feb-93 at 4:32 P.M.



STEPchip

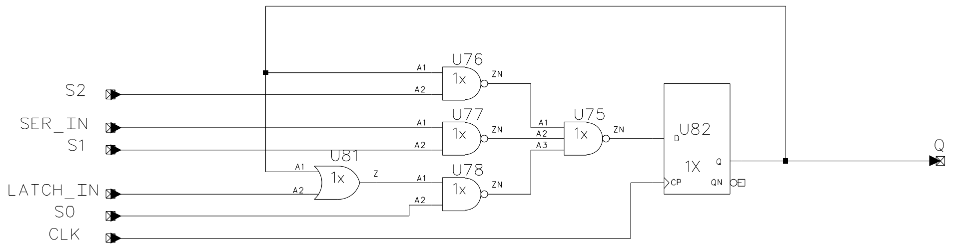
[la]scan_reg by nhm on 15-Feb-93 at 4:32 P.M.

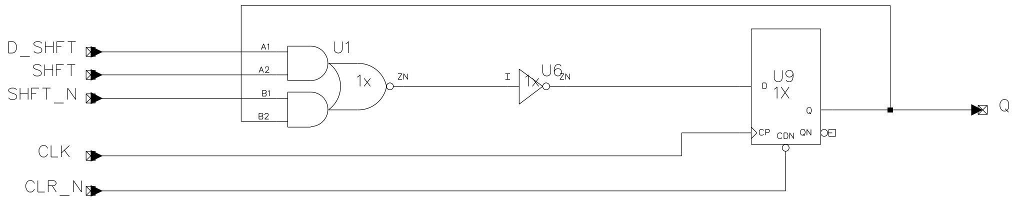


STEPchip

scan_reg.ps

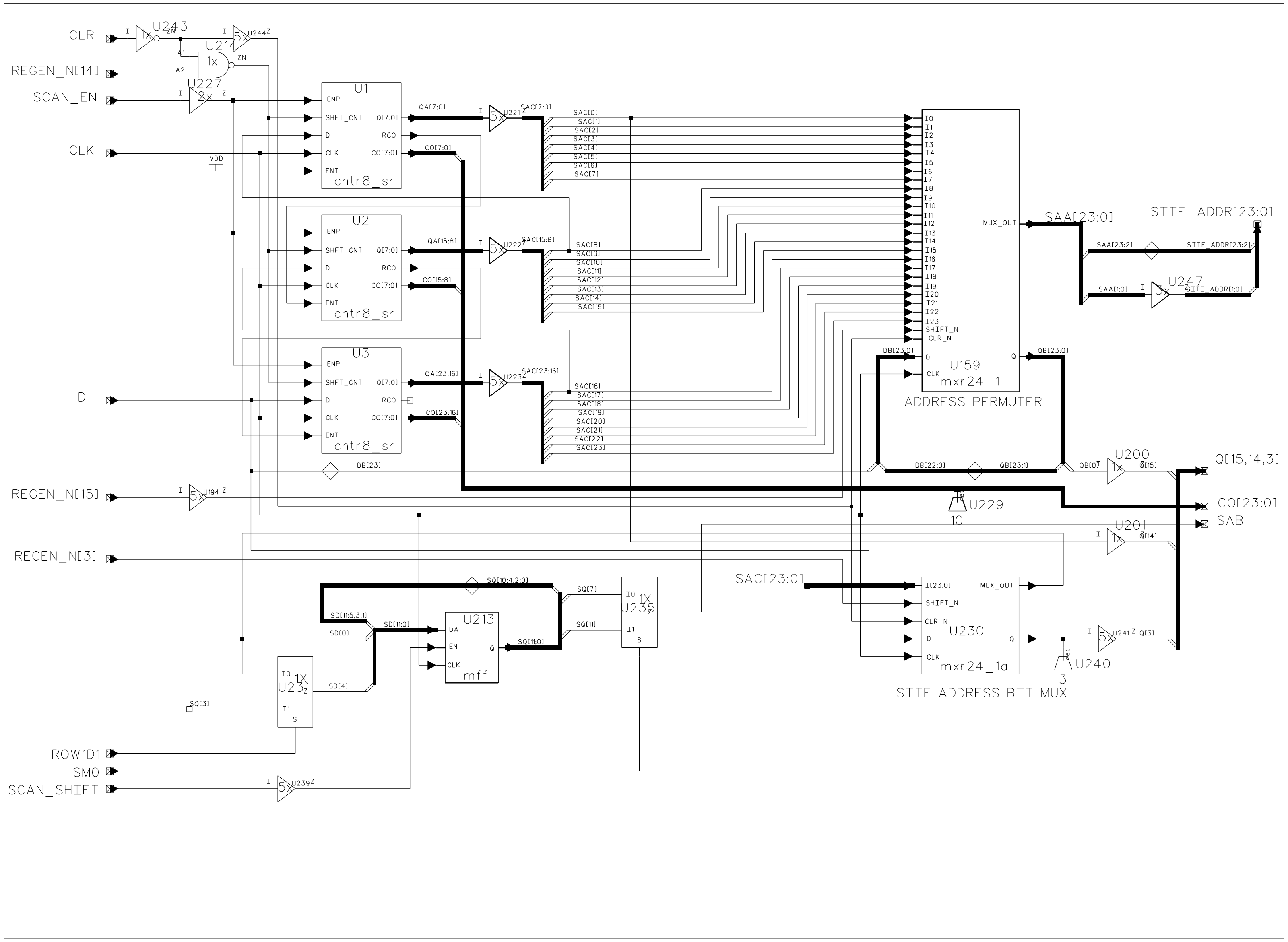
[la]ser_latch by nhm on 15-Feb-93 at 4:32 P.M.



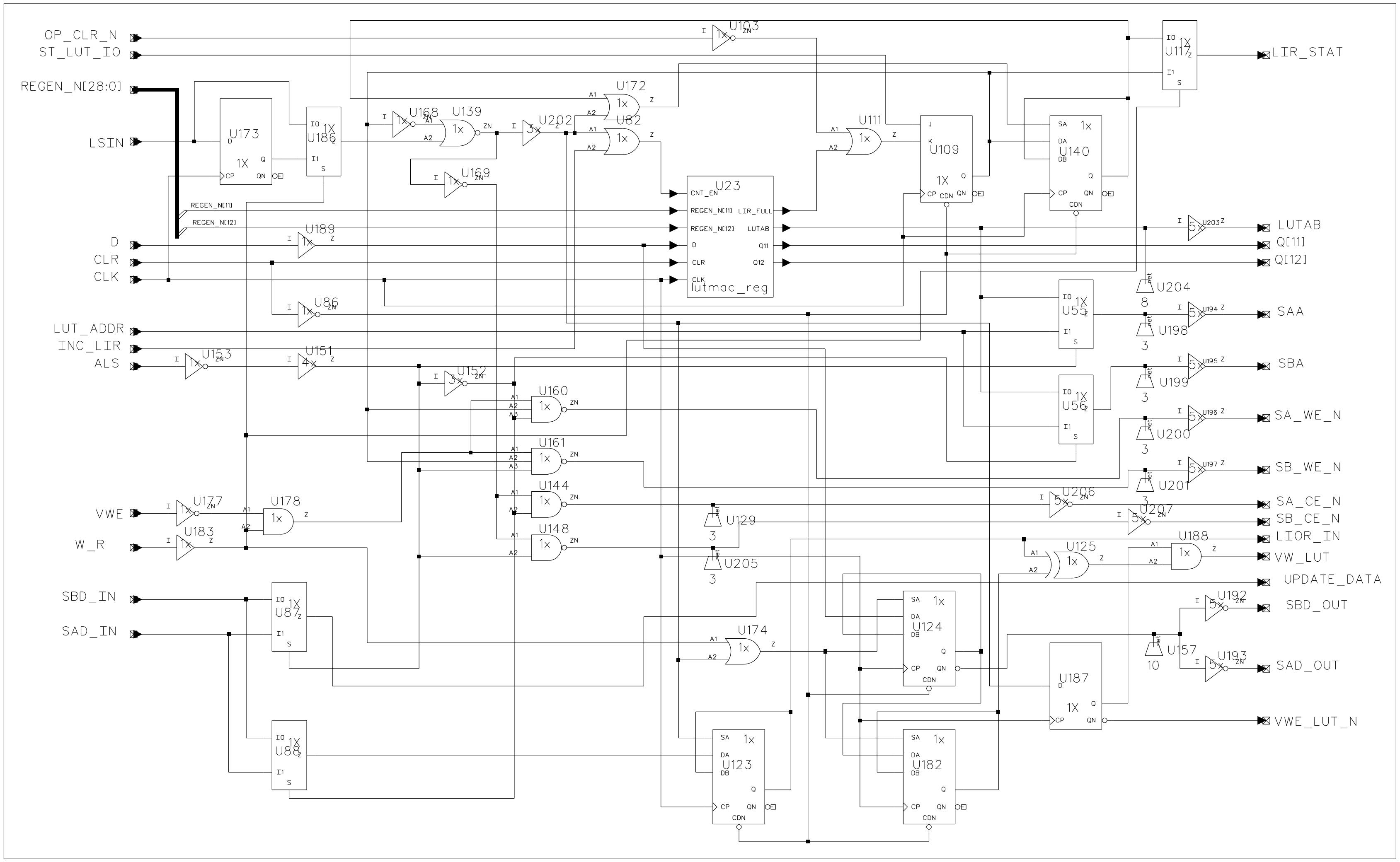


STEPchip

serial_in.ps


CLR
REGEN_N[14]
SCAN_EN
CLK
D
REGEN_N[15]
REGEN_N[3]
ROW1D1
SM0
SCAN_SHIFT
U243
U214
U227
U1
cntr8_sr
U2
cntr8_sr
U3
cntr8_sr
U159
mxr24_1
ADDRESS PERMUTER
SAA[23:0]
SITE_ADDR[23:0]
U247
U200
Q[15,14,3]
CO[23:0]
SAB
U201
U229
SAC[23:0]
U230
mxr24_1a
SITE ADDRESS BIT MUX
U240
U213
mff
U235
U231

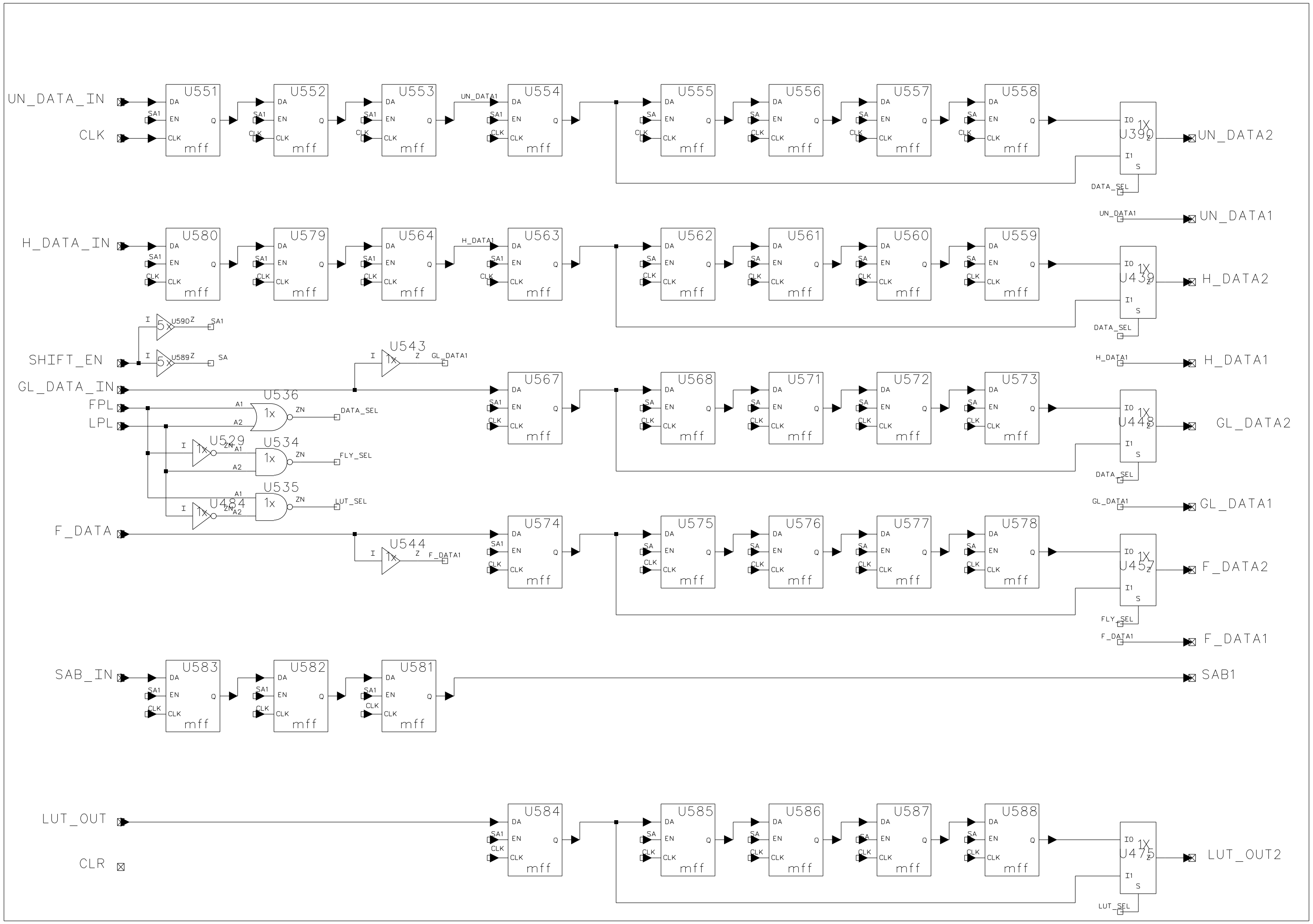
[la]sram_if by nhm on 15-Feb-93 at 4:33 P.M.



STEPchip

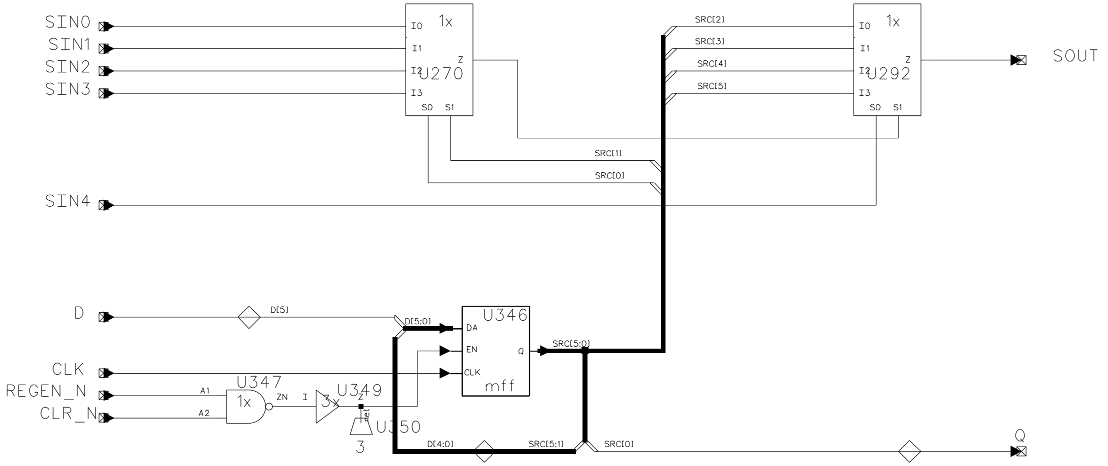
sram_if.ps

[la]src_delay by nhm on 15-Feb-93 at 4:33 P.M.



STEPchip

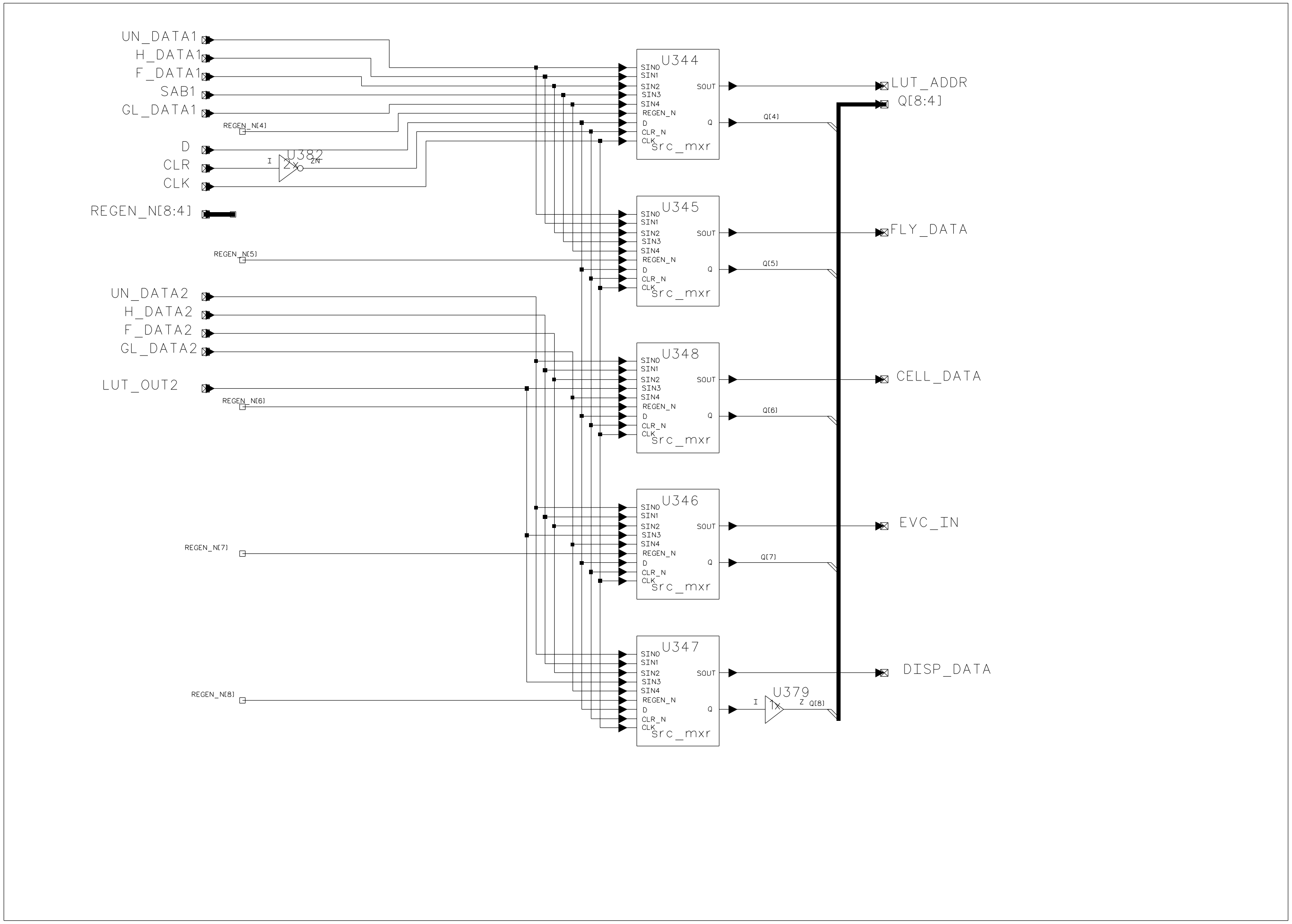
[la]src_mxr by nhm on 15-Feb-93 at 4:33 P.M.



STEPchip

src_mxr.ps

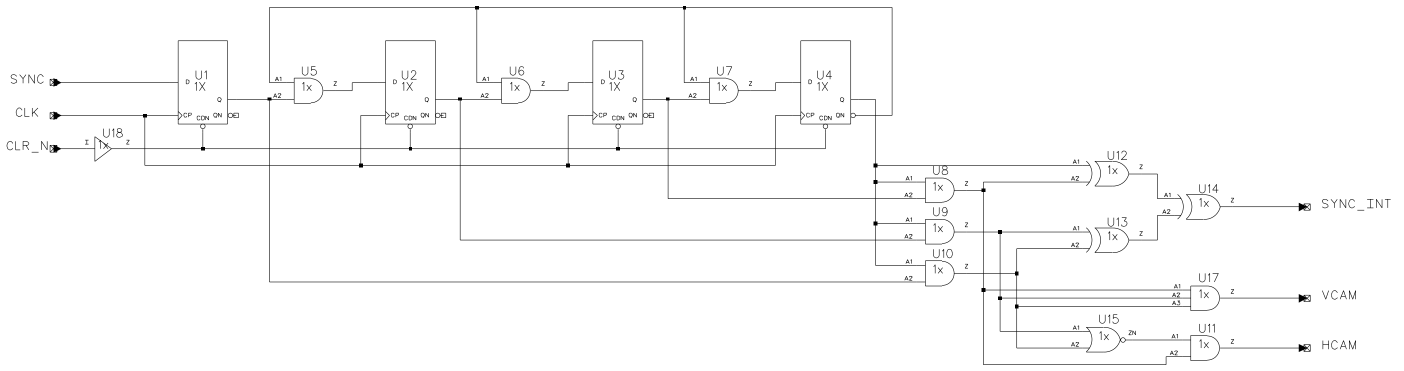
[la]ssr_decode by nhm on 15-Feb-93 at 4:33 P.M.



ssr_decode.ps

STEPchip

[la]sync_dcd by nhm on 15-Feb-93 at 4:33 P.M.



STEPchip

sync_dcd.ps

[la]vwe_reg by nhm on 15-Feb-93 at 4:34 P.M.