State Assignment and Minimization using Meg **(***Mealy Machine***)**

Meg (Mealy Equation Generator) is a state machine generator. It accepts a high level language description of a state machine and translates it into an output format suitable for implementation. The output of *Meg* is a truth table which can be minimized using *Espresso* and fed into the PLA generation tool *MPLA* to generate a PLA layout of the Mealy machine. *Please refer to the Meg, Espresso and MPLA man pages for details.* In this tutorial we will demonstrate the use of *Meg* using two different examples.

Example 1

In this simple example we will demonstrate the use *Meg* to create a Mealy implementation of a sequence detector with one input and one output. The output should become "1" when the detector receives the sequence "0101" on the input. The state diagram for this detector is shown in Fig.1.

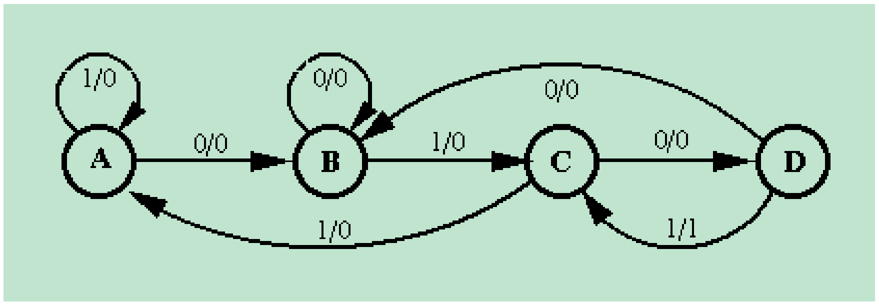

Figure 1: State diagram of the 0101 sequence detector

- ***Describe the sequential circuit using a Meg input program:*** Create an input program (***seqdet.meg***) as follows: Describe the ordering of inputs and outputs using using the commands **INPUT** and **OUTPUT.** Next, describe each state in the state machine by providing input condition and next state information using **IF-THEN-ELSE** statements. Output assertions are provided using parenthesis as shown below. The sequence detector input program, ***seqdet.meg*** is given below.

```
--A 0101 sequence detector
--Mealy machine implementation for meg

INPUTS:       x;
OUTPUTS:   z;
A:      IF x THEN A ELSE B;
B:      IF NOT x THEN B ELSE C;
C:      IF x THEN A ELSE D;
D:      IF x THEN C (z) ELSE B;
```

- ***Generate a truth table for the Mealy circuit:*** Use *Meg* as follows to generate an output truth table for the sequence detector.

```
meg -T seqdet.meg
```

The output truth table file (***meg.tt***) generated by *Meg* is shown below.

```
.i 3
.o 3
.ilb x StBit0* StBit1*
.ob StBit1* StBit0* z
.p -1
000100
100000
001100
101010
010110
110000
011100
111011
.e
```

Note: The statements **`.ilb'** and **`.olb'** give the labeling of inputs and outputs in the PLA layout that will be finally generated.

- ***Generate a PLA layout:*** Use *Espresso* and *MPLA* to generate a PLA layout using the ***meg.tt*** file as follows:

```
espresso meg.tt  |  mpla -I -O -s SCS3cis -o seqdet
```

In the above command line, **- I** and **- O** are used to provide clocked inputs and outputs in the PLA layout. The above command will generate the *Magic* layout ***seqdet.mag.*** The sequence detector PLA layout is shown in Fig 2.

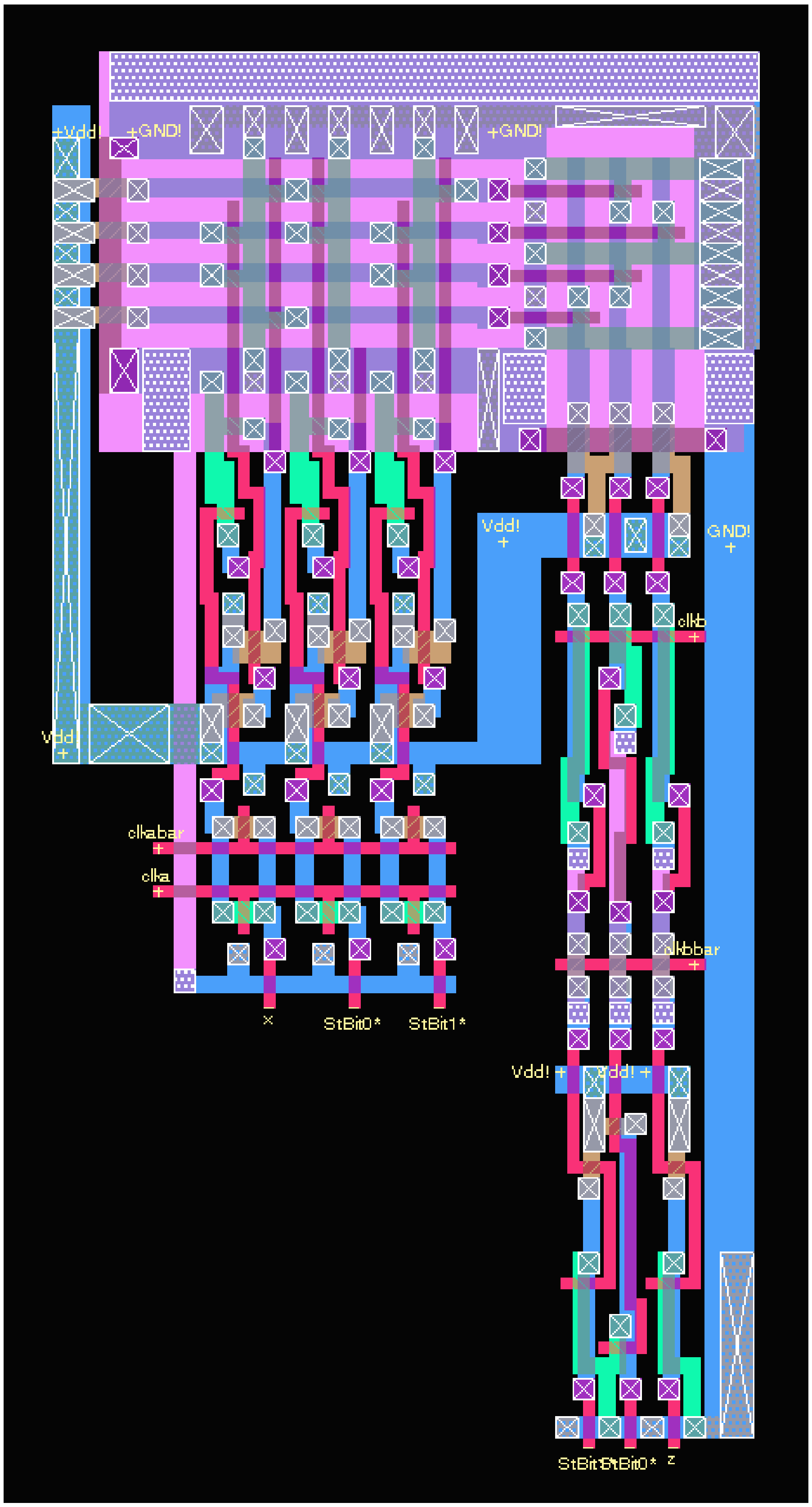

Figure 2: 0101 sequence detector PLA layout (without supply and feedback connection)

● ***Create the final layout:*** From the layout in Fig. 2 you can see that, *(i)* the ***Vdd*** connections for the output registers are not attached to the main PLA ***Vdd***, and *(ii)* the state bits feedback connections from output to input are missing. Make these connections in *Magic* using metal1layer. The final sequence detector layout is shown in Fig. 3.

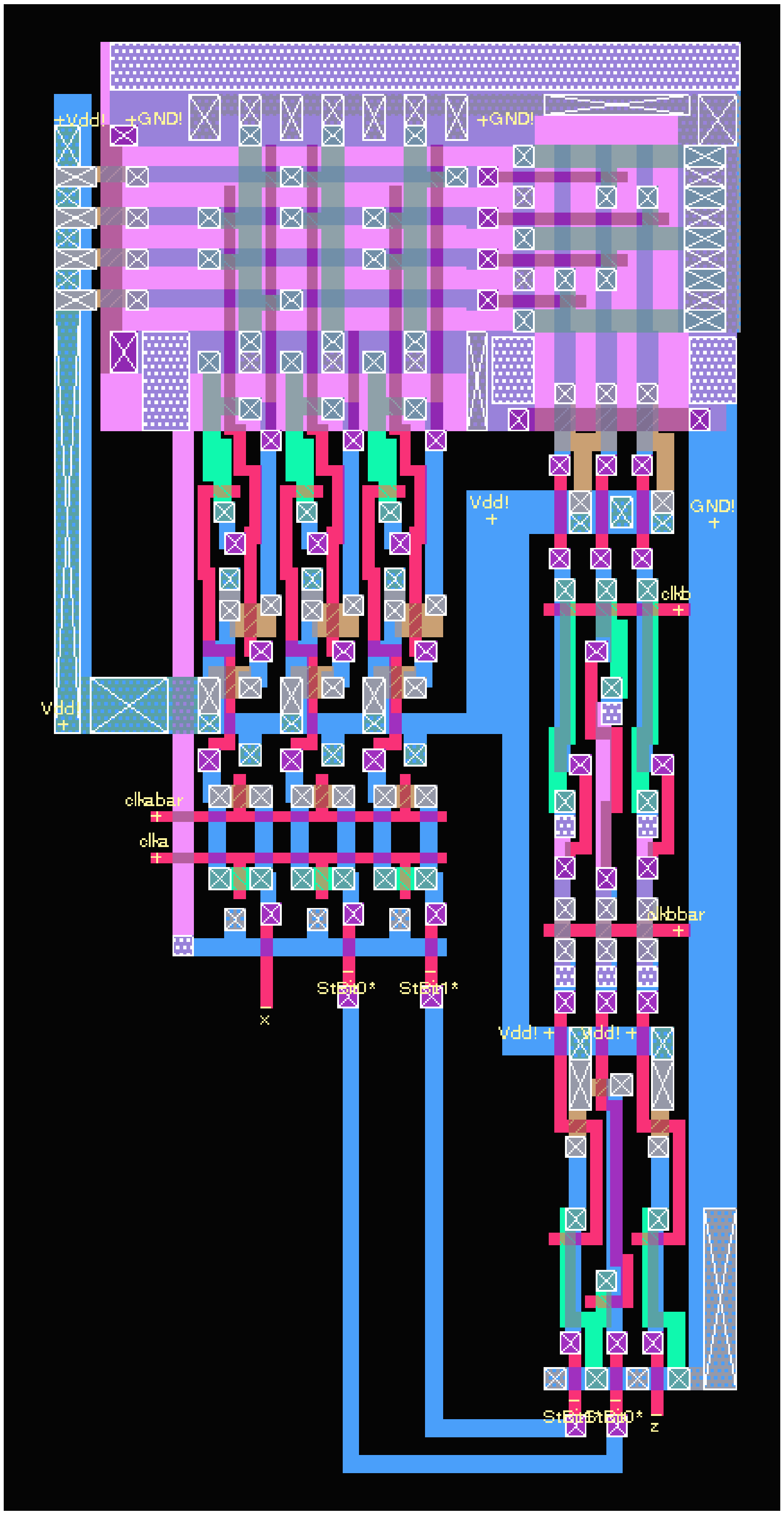

Figure 3: 0101 sequence detector PLA layout with supply and feedback connections

Example 2

In the next example we will explore some other features of *Meg* using a ***2-bit Up/Down Counter.*** The *Meg* input program for this counter is given below.

```
--   2-BIT UP/DOWN COUNTER

INPUTS:  restart  up  down;
OUTPUTS:  b1  b0;

RESET ON restart TO St0;

St0  :  case  (up down)
          0  0    =>  St0 (b1=0 b0=0);
          1  0    =>  St1 (b1=0 b0=1);
          0  1    =>  St3 (b1=1 b0=1);
     endcase => ANY;

St1  :  case  (up down)
          0  0    =>  St1 (b1=0 b0=1);
          1  0    =>  St2 (b1=1 b0=0);
          0  1    =>  St0 (b1=0 b0=0);
     endcase => ANY;

St2  :  case  (up down)
          0  0    =>  St2 (b1=1 b0=0);
          1  0    =>  St3 (b1=1 b0=1);
          0  1    =>  St1 (b1=0 b0=1);
     endcase => ANY;

St3  :  case  (up down)
          0  0    =>  St3 (b1=1 b0=1);
          1  0    =>  St0 (b1=0 b0=0);
          0  1    =>  St2 (b1=1 b0=0);
     endcase => ANY;
```

In the program, note the use of "**RESET**" statement. It is a good idea to include a reset signal in a sequential circuit. It improves testability of your circuit since you can always use this signal to initialize your circuit to a known "reset" state (St0 in the above example). Fig 4 shows a PLA layout for the counter.

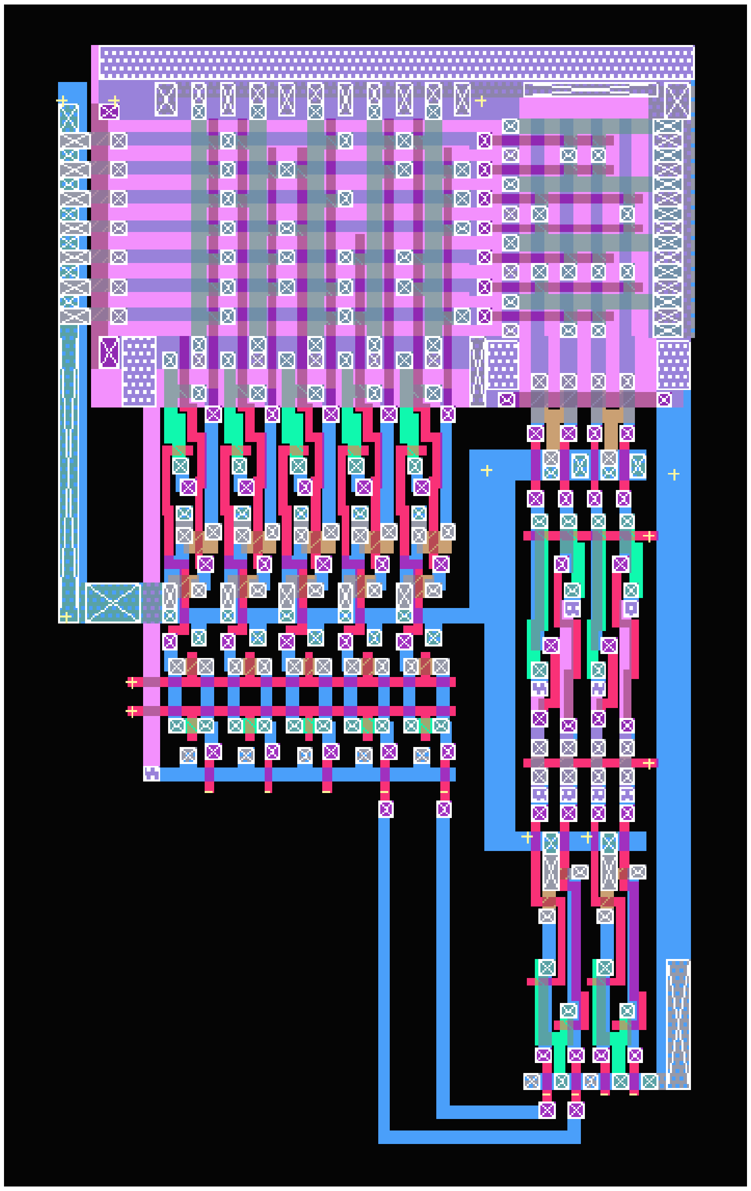

Figure 4: PLA layout of the 2-bit up/down counter

Details of the *IRSIM* simulation of the above counter are presented in the *IRSIM tutorials.* The results of this simulation are depicted in Fig. 5 below.

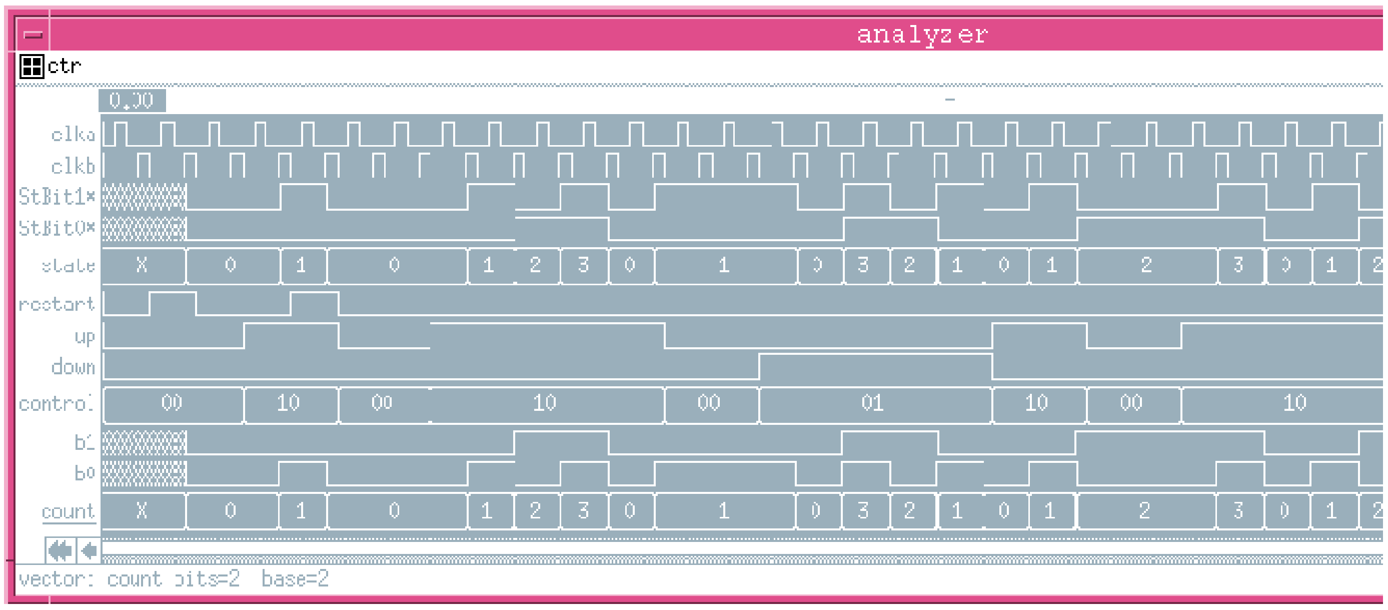

Figure 5: IRSIM simulation of the 2-bit up/down counter

Return to EE6325 homepage