State Assignment and Minimization using Peg(Moore Machine)

Peg (PLA Equation Generator) is a state assignment and minimization tool for generating *Moore sequential machines.* The output of *Peg* is a set of state and output equations. These equations can be fed to *Eqntott* to generate a truth table which can be minimized using *Espresso* and fed into the PLA generation tool *MPLA* to generate a PLA layout of the Moore machine. *Please refer to the Meg, Eqntott, Espresso and MPLA man pages for details.* In this tutorial we will demonstrate the use of *Peg* using two different examples.

Example 1

In this simple example we will demonstrate the use *Peg* to create a Moore 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.

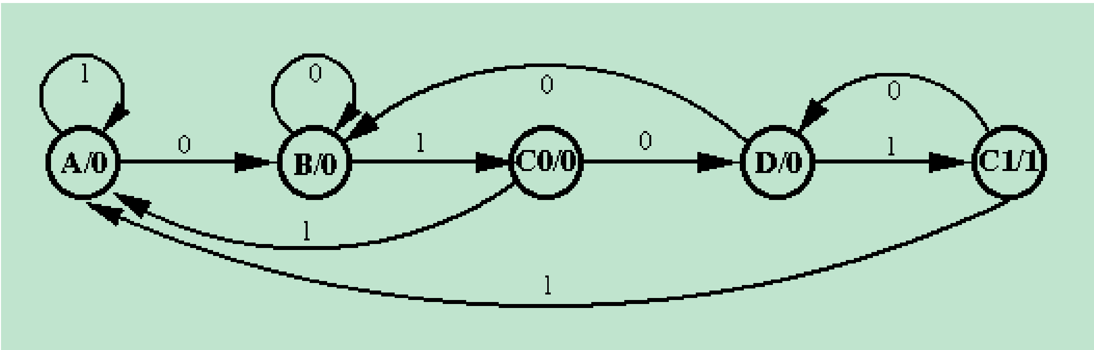

Figure 1: State diagram of the 0101 sequence detector

- ***Describe the sequential circuit using a Peg input program:*** Create an input program (***seqdet.peg***) 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 the **ASSERT** statement as shown below. The sequence detector input program, ***seqdet.peg*** is given below.

```
--A 0101 sequence detector
--Moore machine implementation for peg

INPUTS:      x;
OUTPUTS:  z;
A:       IF x THEN A ELSE B;
B:       IF NOT x THEN B ELSE C0;
C0:     IF x THEN A ELSE D;
C1:     ASSERT z;
           IF x THEN A ELSE D;
D:       IF x THEN C1 ELSE B;
```

- ***Generate equations for the Moore circuit:*** Use *Peg* as follows to generate an output truth table for the sequence detector.

```
peg seqdet.peg  >  peg.eqn
```

The output truth table file (***peg.eqn***) generated by *Peg* is shown below.

```
INORDER      =   x  InSt0*  InSt1*  InSt2*;
OUTORDER  =   OutSt2*  OutSt1*  OutSt0*  z;
OutSt2*          =   ( InSt0*&!InSt1*&!InSt2* ) | ( !x&!InSt0*&!InSt1* );
OutSt1*          =   ( x& InSt0*&!InSt1*&!InSt2 ) | (  x&!InSt0*&!InSt1*&InSt2* );
OutSt0*          =   ( !x&!InSt0*& InSt1* );
z                      =   ( !InSt0*& InSt1*& InSt2* );
```

Note: The statements `**INORDER'** and `**OUTORDER'** give the ordering of inputs and outputs in the PLA layout that will be finally generated.

- ***Generate a PLA layout:*** Use *Eqntott, Espresso* and *MPLA* to generate a PLA layout using the ***peg.eqn*** file as follows:

```
eqntott -l  peg.eqn  |  espresso  |  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. All the above commands can be combined on one command line as follows:

```
peg seqdet.peg  |  eqntott -I  |  espresso  |  mpla -I -O -s
SCS3cis -o seqdet
```

The above command will generate the *Magic* layout ***seqdet.mag.*** (*Note: At present Eqntott only runs on an old machine called tulia*). The sequence detector PLA layout is shown in Fig 2.

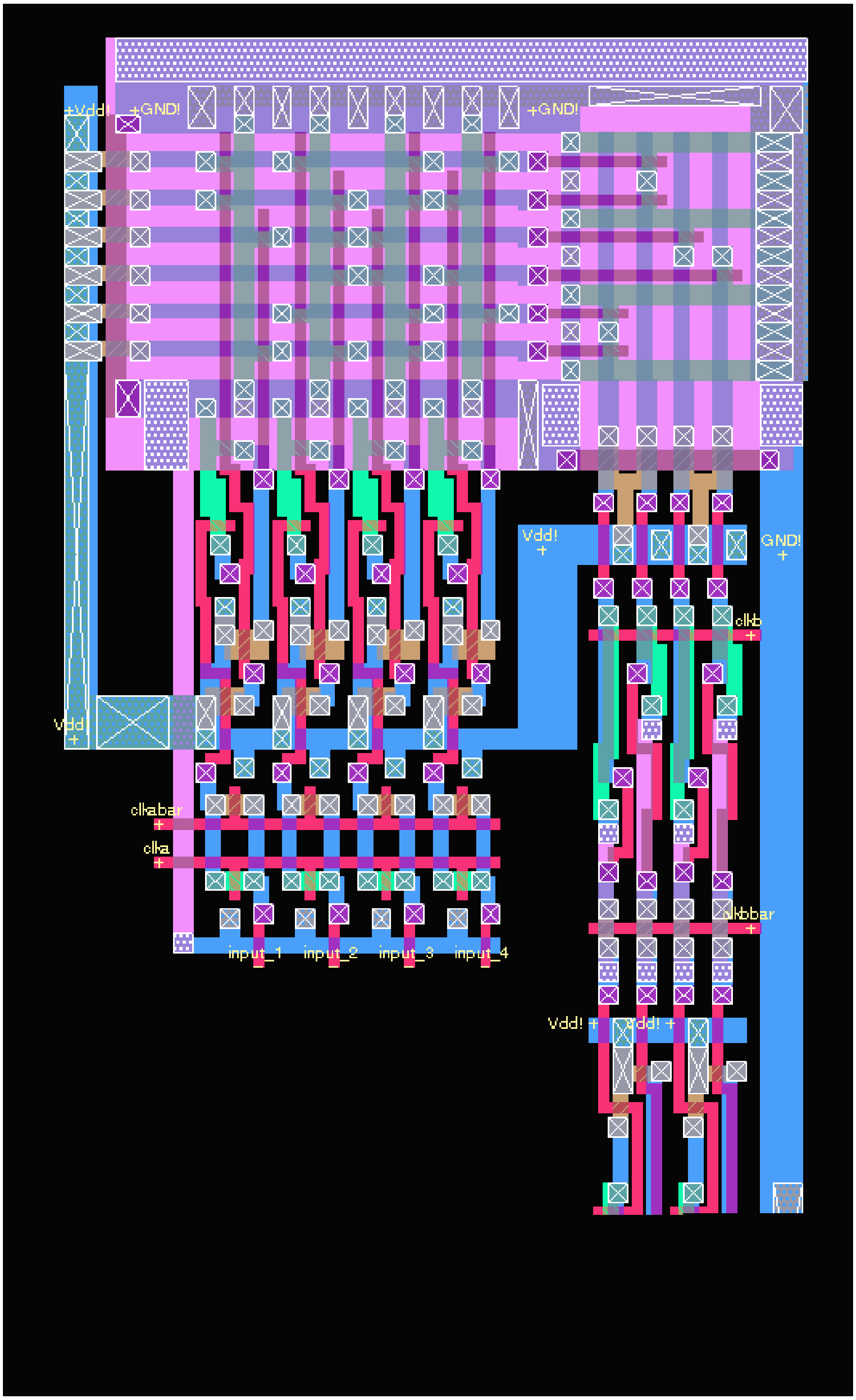

Figure 2: 0101 sequence detector PLA layout (without supply and feedback connection)

● ***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 metal1 layer. The final sequence detector layout is shown in Fig. 3.

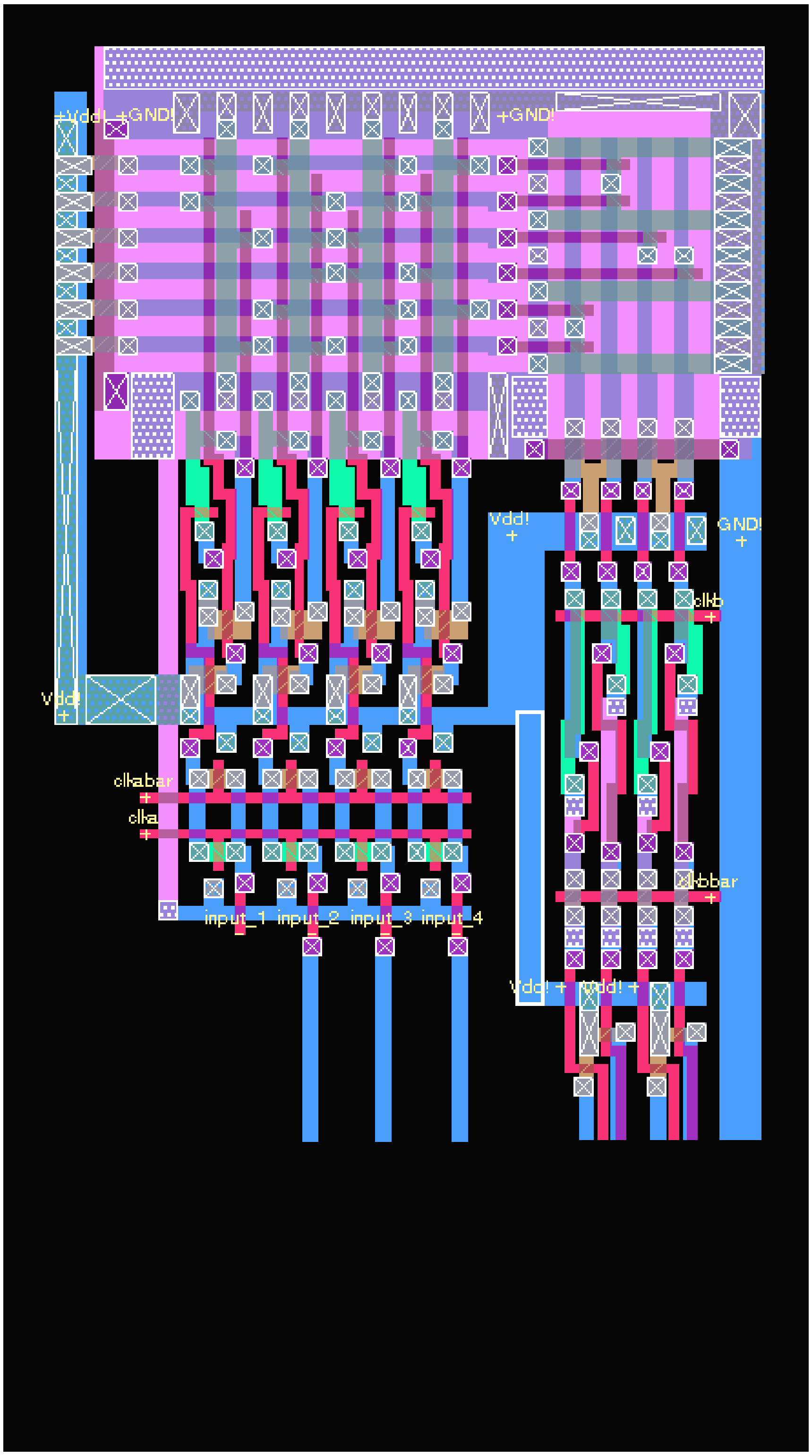

Figure 3: 0101 sequence detector PLA layout with supply and feedback connections

Example 2

In the next example we will explore some other features of *Peg* using a ***Bit Serial Adder.*** The *Peg* input program for this counter is given below.

```
-- 1-bit Serial Adder using Moore Machine

INPUTS   :   RESET a b;
OUTPUTS  :   sum;

W  :  CASE (a b)
         00 => W;
         01 => X;
         10 => X;
         11 => Y;
      ENDCASE;

X  :  ASSERT sum;
      CASE (a b)
         00 => W;
         01 => X;
         10 => X;
         11 => Y;
      ENDCASE;

Y  :  CASE (a b)
         00 => X;
         01 => Y;
         10 => Y;
         11 => Z;
      ENDCASE;

Z  :  ASSERT sum;
      CASE (a b)
         00 => X;
         01 => Y;
         10 => Y;
      ENDCASE => Z;
```

In the program, note the use of "**RESET"** input. It is a good idea to include a reset signal in a sequential circuit. It im proves testability of your circuit since you can always use this signal to initialize your circuit to a known "reset" state (state "**W**" in n the above example). In the above reset is implemented by including the keyword **"RESET"** as one of the input signals. In this implementation the state machine will jump to the first state on the state list when the signal **"RESET"** is asserted high (i.e., state "**W"** in the above example). Alternatively, you may force a jump to the first state on the state list by adding logic to the PLA state outputs to pull all of the state output lines low when a reset is desired. Fig 4 shows a PLA layout for the counter. *Note: This layout was modified by adding labels based on the notation used in the input **.peg** file.*

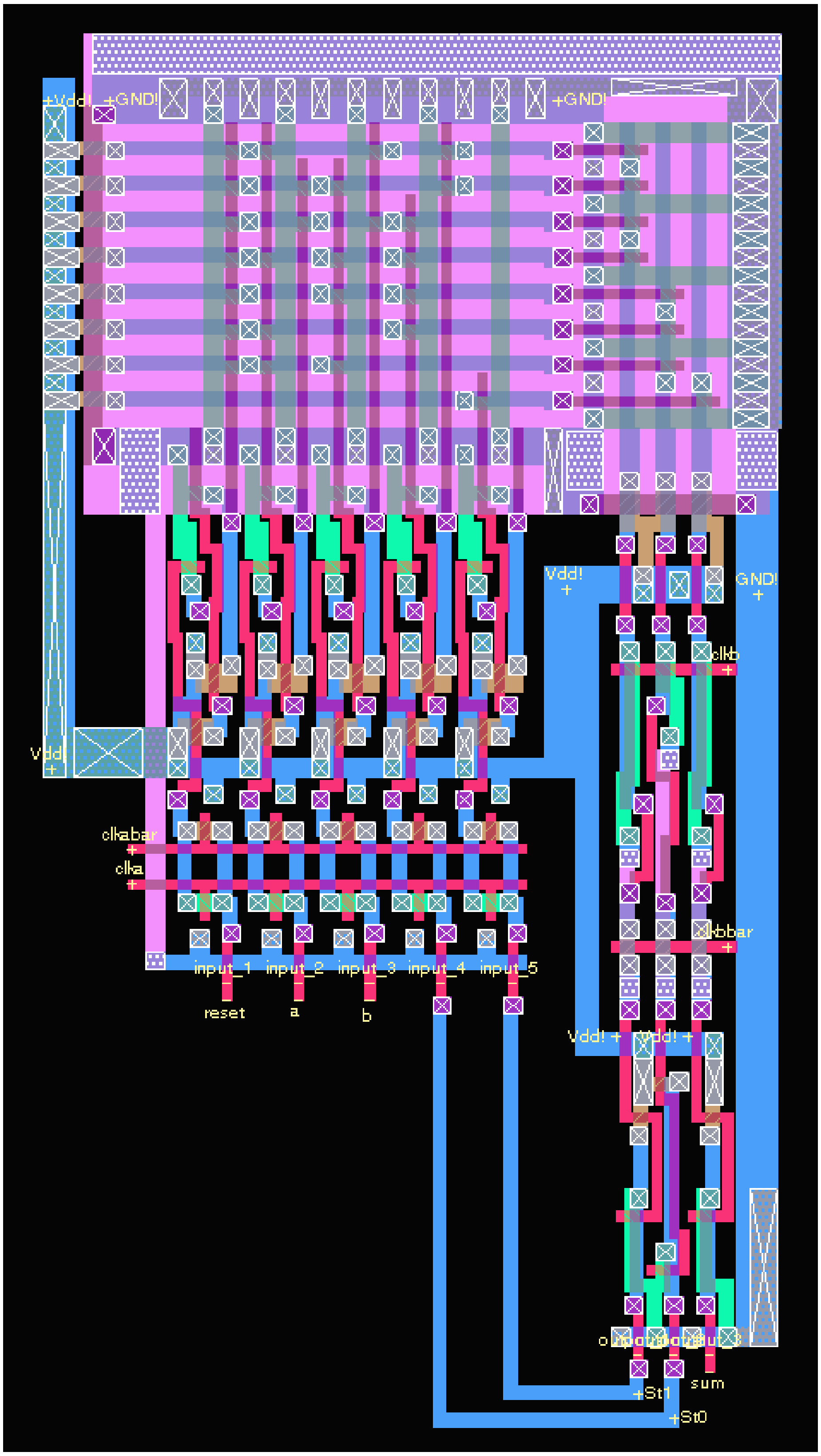

Figure 4: PLA layout of a Bit Serial Adder

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

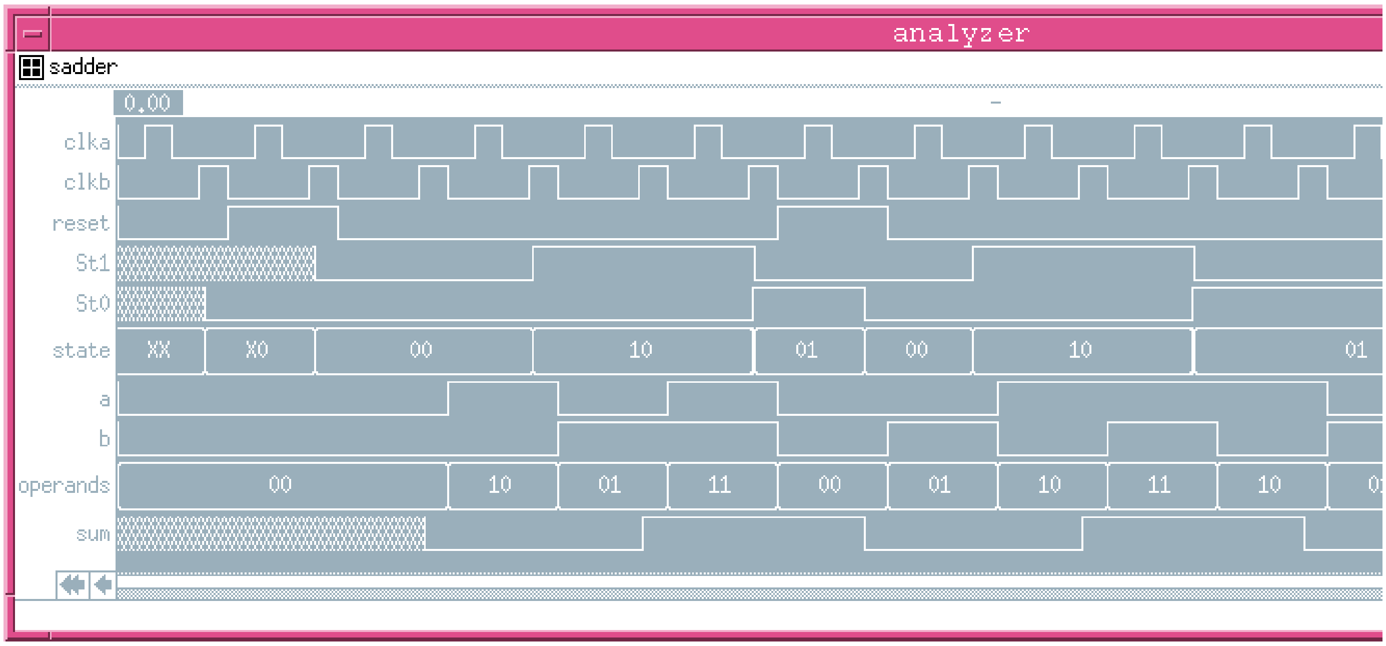

Figure 5: IRSIM simulation of the Bit Serial Adder

From the above results it can be seen that the output ***sum*** which is associated with the state (since it is a Moore machine), changes one cycle after the state change. This because the output is also clocked due to the use of the -O option for *mpla.*

Return to EE6325 homepage