

**EE681**  
**Final Project**

**Automated Clock Gating**

**Sheng Ye**  
**Woojoo Lee**

**University of Southern California**

# EE 681 Final Project

## 1. Introduction

Clock gating is a popular technique to save power consumption in a sequential circuit by decreasing the switching probability of gates and wires. The drawback of adding clock gating is that the area of the circuit increases. Thus there is an optimal solution for a circuit in terms of power saving/area overhead. Our goal in this project is to find this solution automatically for any sequential circuit using Boolean switching theories.

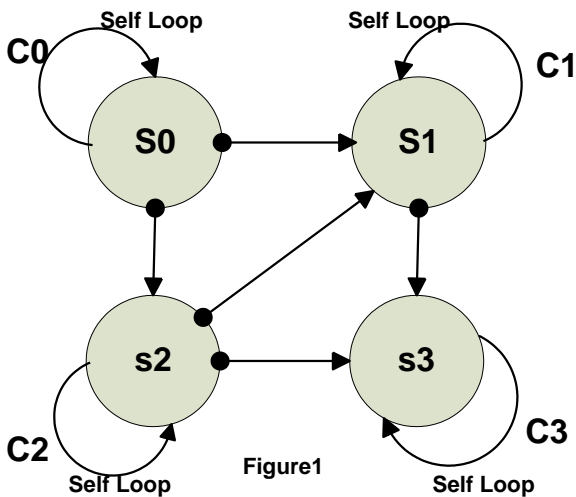
Our approaches are based on clustering latches whose clock signal is gated by some logic. Finding common subsets of clock gating conditions is our first part of project, and implementing them in SIS is the second part. Then verifying them using our programs is the last part of project. In our project, we use two different approaches, self-loop based and CODC based clock gating, and we can reduce the power consumption without much area overhead.

## 2. Background

### 2-1) Self-loop based clock gating

According to Luca Benini's paper (L. Benini and G. De Micheli, Dynamic Power Management, Boston: Springer, 1998), if a state has a self-loop in the state transition graph (STG), then clock can be stopped whenever a self-loop is to be executed. We first used this idea, a self-loop in FSM.

In the figure 1, for instance, if the previous state is S0 and input is C0, (C0·S0), then all



states including C0 should be the same with previous states, and the clocks for all states can be not switched. If we can know all the self-loop conditions in the STG, we can cluster them to apply clock gating. In the figure1 case, we know that only the condition  $(C0 \cdot S0 + C1 \cdot S1 + C2 \cdot S2 + C3 \cdot S3)$  can propagate data, which means clock should be switched. And this can be designed like figure 2. In this design, note that we can use only one AND gate to make a control signal to clock gating, avoid to use XOR gate which is

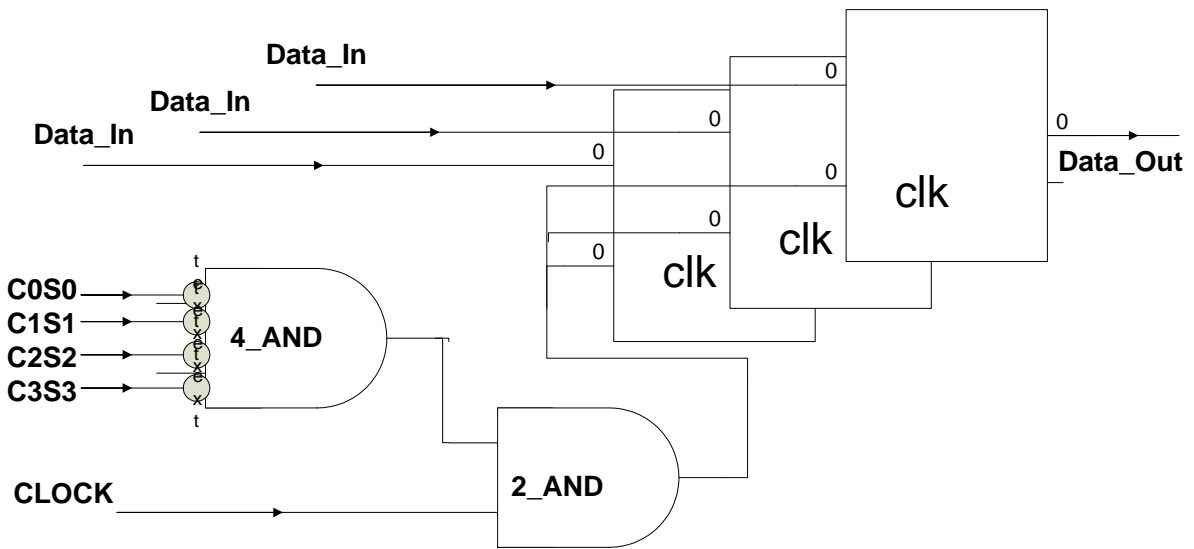


Figure 2

commonly used in automated clock gating but requests more area than AND gate.

### 2-2) CODC based clock gating

Using Compatible Observability Don't Cares (CODC) to simplify a Boolean network is a common technique in logic synthesis. The use of CODC can be extended to clock gating by stopping the clock to a flip-flop when the CODC condition satisfies for its output to all primary outputs (including inputs to flip-flops) in its transitive fanout cone. This technique not only saves the switching power for the flip-flops that are gated, but some or all the unnecessary switching power of the nodes that are in their transitive fanout cone. Figure3 is a simple example demonstrates this.

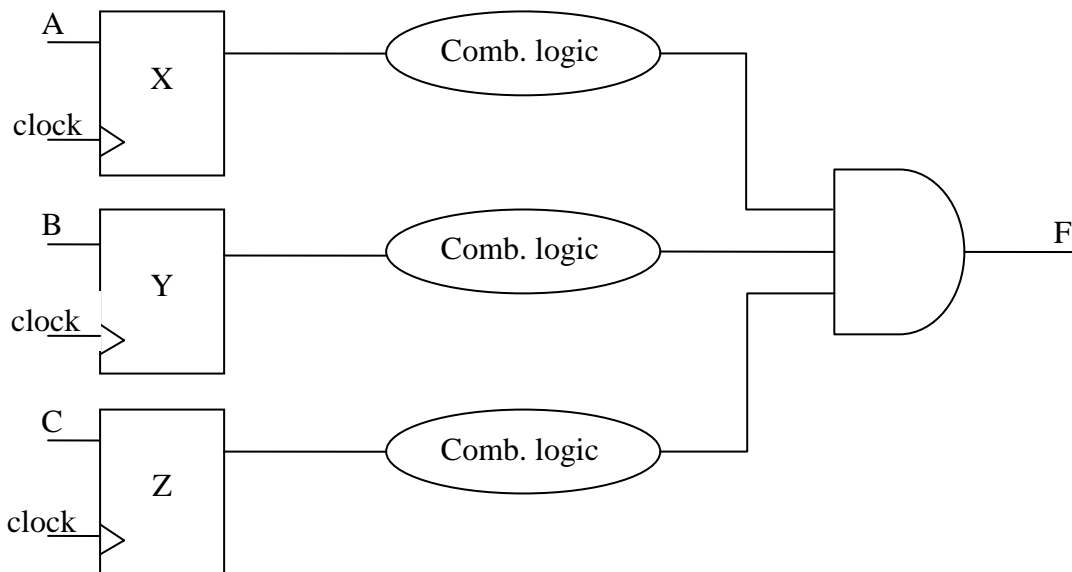


Figure 3

The CODC condition for B to F is  $A = 0$  or  $C = 0$ , and the CODC condition for C to F is  $A = 0$ . So we can use an AND with input A to gate the clock to flip-flops Y and Z.

### 3. Implementation

#### 3-1) Self-loop based clock gating

We focused on finding self-looping condition in a network, and connect them to one clock control gate, AND gate. First we tried to convert the network to the STG in order to find each self-loops of given network. We could successfully convert the blif-type network to the kiss-type STG and find self-loops. But we realized that it is hard to recover the original network from the STG in SIS. Then we rather limited the network to the one used by one-hot encoding, because one-hot encoding can guarantee that all latches can be directly matched to the all states. In other words, we can just find all latches' self-loop condition, and use them as control nodes. Here is a simple explanation about the part of the code related with these procedures.

```
Detect_loop(network,option){}
```

```
Check_One_hot_encoding(network): Compared extracted STG and duplicated Network to determine it is a one-hot encoding network. If it is true, go to the next.
```

```
Find_selfloop_input(list,node): Store all fanin nodes of all latches to the arrays. Store all transitive fanout nodes through the each fanout path started from all latch output nodes. Compare the two kinds of arrays to find a self-loop.
```

```
Node_function(matching_node)==NODE_OR//NODE_NAND: if we can find nodes from the previous step, we can determine each of them whether it is self-loop node or a other input node, though the idea that self-loop node should be the fanout of OR or NAND gate in one-hot encoding network.
```

Then we connected all inverted self-loop nodes to the AND gates. And the fanout node of AND gate and clock nodes are connected to another AND gate. This design is the same with figure2.

```
Node_func_t: define the new structure type to make a logic gate and add it to the network. It includes logic gate onset function and the number of fanin.
```

```
Node_Create_funct(function) : Make a new logic gate node
```

```
Network_add_node: Use this built-in function to add the new node to the network.
```

The next step was about the implementation of clock chain. We checked that there is no proper given benchmark in SIS, which has to be one-hot encoding, we used other benchmark, fsm.blif, 4 stages FSM from DaChang. But this benchmark has no buffer in between clock signals and latches, it is hard to test the effect of clock chain. Clock chain, or clock tree is directly relative with clock's power consumption, because it causes big loads when distributing clock signals to the whole circuit. Without considering it, in our case, we would just be able to reduce the latches' switching power consumption induced by clock. So we designed a module that makes us to control how many clock buffers will be generated in the network.

*Define NUMBER\_OF\_CLOCK\_BUFFERS:* we can change this variable to control the clock loads in the network.

*COMMANDs :* "myproj -h" generates clock buffers, "myproj -s" added clock gating module without clock buffers, and "myproj -t" do both of them.

### **3-2) CODC based clock gating**

While SIS already has functions that utilize ODCs for logic simplification, we found that it was difficult to incorporate them into the aforementioned clock gating scheme. The ODCs generated by the functions for the test cases we conducted were incorrect. So while implementing our own ODC calculation functions was impossible with such a limited amount of time, a compromise was made so the concept of clock gating using CODC was still exploited without using any ODC calculation function. We limited the configuration of circuits for our solution to handle so the ODC calculation is simple and obvious. The following section describes this configuration of circuits in detail.

#### **CODC Based Clock Gating for a Set of Circuits with Certain Configuration**

Figure 4 describes the circuits our solution can handle. It has the following requirements for the circuit:

1. The flip-flops in a group on the left can only fanout to a same gate that has a controlling vale (except for X);
2. One of the flip-flops (X) in this group has to connect to the gate directly;
3. Node F has to be either an input to a latch, or a primary output;
4. All flip-flops are connected to the same clock;

Figure 5 shows the clock gated version of the circuit in Figure 4 using the CODC based clock gating algorithm.

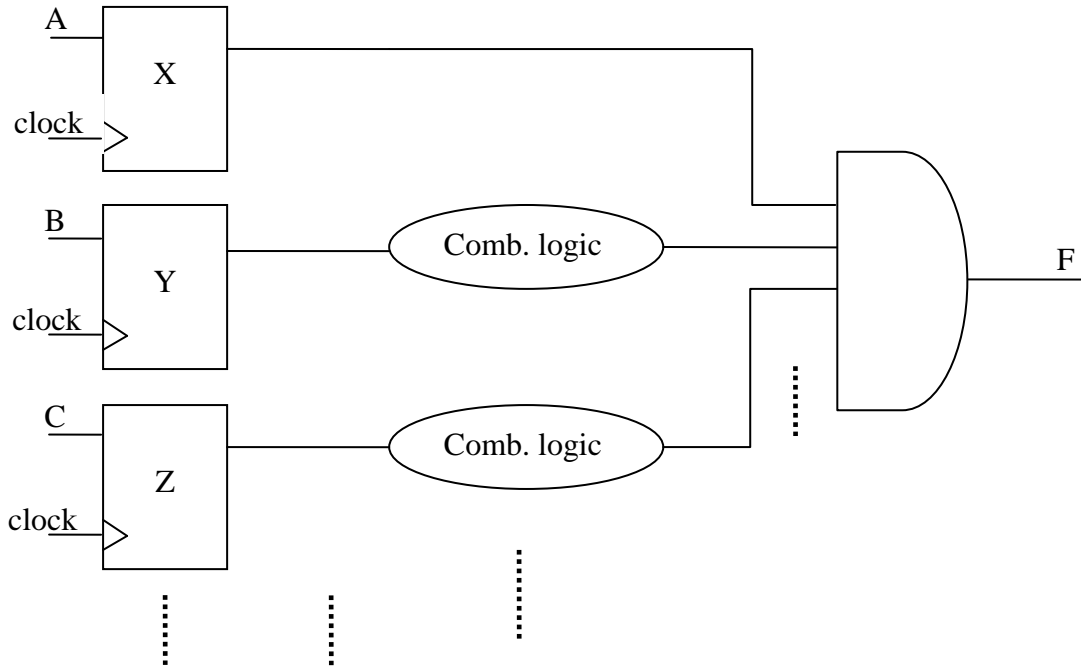


Figure 4

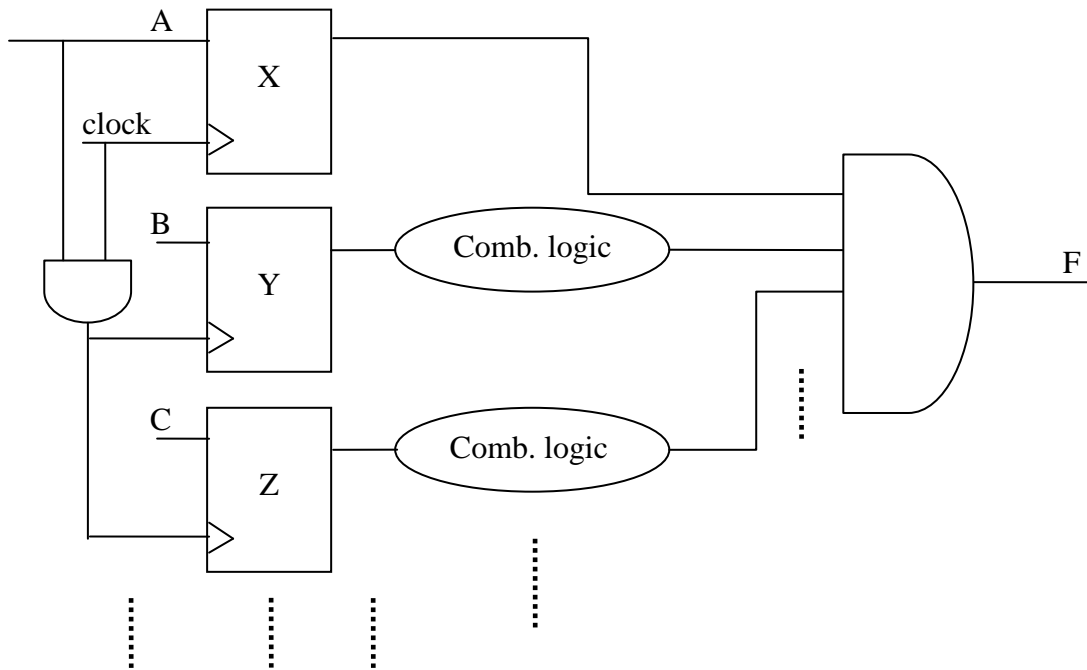


Figure 5

### **CODC Based Clock Gating Algorithm**

For each latch  $L_i$ :

    Finds the first Type OR or Type AND gate,  $G_i$ , at its input;

    Finds a latch,  $J_i$ , whose output is directly connected to  $G_i$ ;

    For every latch in  $G_i$ 's transitive fanin cone except  $J_i$ :

        Checks if the latch only fanouts to  $G_i$ . If no, break the loop;

    Creates a new node,  $A_i$ , with type AND and adds it to the network;

    Connects the input of  $J_i$  to an input of  $A_i$  and connects clock to the other input;

    If  $G_i$  is Type OR and the unateness between  $J_i$ 's output to  $G_i$  is positive:

        Sets  $G_i$ 's unateness from the input of  $J_i$  to negative;

    If  $G_i$  is Type AND and the unateness between  $J_i$ 's output to  $G_i$  is negative:

        Sets  $G_i$ 's unateness from the input of  $J_i$  to negative;

    For every latch in  $G_i$ 's transitive fanin cone except  $J_i$ :

        Set its control to the output of  $A_i$ ;

## **4. Result**

### **4-1) Self-loop based clock gating**

To the experiment, we had to use "power\_estimate" function which is a built in SIS. But this function seemed not working to clock gated network. Then we modified it, specially "power\_seq.c" code for sequential logic network.

When using original "power\_estimate -t sequential" function :

Original circuit : 26.3 uw,

Gated Clock without clk\_bufs : 30.0uw,

Origin + 3 clk\_bufs for each latch: 41.3 uw, Gated Clock+ 3 clk\_bufs for each latch: 45 uw

When we modified it, we used our test result above that one clock buffer causes 1.25uw power increase, and the amount of increased power consumption from Original circuit to Gated Clock without clk\_bufs, 3.7uw is just from adding clock gating module. And we made assumptions that the power consumption of latch before adding clock gating module would be in the 2uw to 5uw range. Then we used power\_estimate function with option 'sampling' to make a reasonable assumption about *changed switching probability factor* after adding clock gating module. Though 22400 times random input vector generation in 'Gated Clock without clk\_bufs mode', we know that there is a 4.6uw difference between minimum power and maximum power. If we guess

the default latch power is about 2.5uW, then  $(\text{num}_{\text{latches}} \cdot 2.5\text{uW}(1 - x)) = 4.4\text{uW}$ , so the *changed switching probability factor*,  $x$ , would be 0.56.

And we tested our program with changing switching probability of enable nodes as 0.28. We also added 3 clock buffer to each latch, the total power consumption for clock chain became almost 50% of the circuit total power.

When using modified "power\_estimate" with Latch\_power: 2.5uW, and *changed switching probability* of Clock enable signal :  $0.56 * 0.5 = 0.28$   
 Origin+3clk\_bufs for each latch: **41.3uW**, Gated Clock+ 3 clk\_bufs for each latch: **38uW**  
**7% Power consumption reduction**

And we also could get other result from small modification of values:

When using modified Latch\_power: 4uW, Num of clk\_bufs for each latch: 3  
 Origin **41.3uW**, Gated Clock: **33.8uW**  
**18% Power consumption reduction**

When using modified Latch\_power: 2.5uW, Num of clk\_bufs for each latch: 4  
 Origin **46.3uW**, Gated Clock: **39.2uW**  
**15% Power consumption reduction**

#### 4-2) CODC based clock gating

Effectiveness of ODC based Clock Gating\*

	No. of Latches gated	No. of gates in combinational logic	Power w.o clock gating (uW)	Power w. clock gating (uW)	Power saved
Test Case 1	1	10	94	81	13.8%
Test Case 2	2	10	127	99	22%
Test Case 3	3	10	213	157	26.3%

\*Test cases are created by Yanzhi Wang and the power estimation function for circuits with gated clocks is from Hsun Wei and Da Cheng

## 5. Discussion and Future Work

Since the built in power estimate function in SIS does not support clock gating, we had to modify the power estimate function and needed to make reasonable assumptions to apply them. We also tried to use a function created by another group to estimate the power of gated clock circuits. However, all the way we tried has not been perfected to estimate the power accurate. So the power saving in the above table does not fully reflect the actual case but shows our approach should be effective to save power consumption and to reduce the area overhead.

The limitation of self-loop based clock gating comes from 'STG to network' recovery problem. So far we can apply it to one-hot encoding network and can expect its power reduction. If we can make an idea about converting STG and network, this approach would be more complete.

Since the algorithm used in CODC part does not rely on the built in ODC function in SIS, a restriction on the input circuits has to be applied. However, it is possible to implement a suitable CODC function for clock gating if time permits. This will allow us to handle any circuits and fully utilize the power of CODC to produce optimal results.

This clock gating solution cannot be evaluated by the method Mehrdad Najibi proposed. Because the method only measures the power saving on the clock tree, but our solution saves power primarily by reducing unnecessary switching activities along the data path.