EE677 Course Project

FSM Minimizer and Synthesizer using ESPRESSO Heuristic

K Akhilesh Rao (213079018)
Dhruva S H (213079022)

Introduction and Motivation

All digital systems are described in RTL design style, which consists of a data-path and a controller.

- The data-path consists of combinational logic and storage elements.
- The controller is a Finite State Machine that controls the data-path.

The data-path is usually the dominating part with respect to both area and resource utilization. Hence, it is essential to minimize the combinational circuits present in the data-path.

The sequential part of the circuit (i.e FSM) consists of some combinational logic (next state logic and output logic), along with flip-flops to store the state.

1. The number of flip-flops in an FSM can be reduced by minimizing the number of states.
2. The combinational logic in an FSM as well as in the data-path can be reduced using logic minimization techniques.

FSM State Minimization using Equivalence Checking

FSM state minimization is done using equivalence checking method. This method checks the state transition table for distinguishable states iteratively.

Two states S_i and S_j are said to be distinguishable states if and only if there exists a finite input sequence for which the FSM produces distinguishable output sequence when it starts from S_i or S_j.

State minimization can be done by forming classes of equivalent states.
Before the first input is applied, it is assumed that all states are equivalent. Hence, there is only 1 class. After the first input is applied, the states can be distinguished. After the second input is applied, the states can be further distinguished, and so on. Eventually, on application of any further number of inputs (which will be $n-1$, where n is the number of distinguishable states), the classes can't be further distinguished, which is the terminating condition.

This process has been implemented in Python, using State Transition Table as the input. The program iteratively finds distinguishable states and comes up with a minimized State Transition Table after combining indistinguishable (or equivalent) states.

Once the number of states have been minimized, the next state logic and output logic of the FSM have to be minimized in order to obtain the best realization of the FSM.

Logic Minimization using ESPRESSO Heuristic

For large scale logic minimization, exact methods such as Q-M Algorithm are not practical. Hence there is a need for heuristic based logic minimization techniques. ESPRESSO is one such heuristic, which is widely used.

The idea of ESPRESSO algorithm is to find a minimal set of min-terms that covers the ON set (i.e all terms that evaluate to 1). It uses positional cube representation.

Operations in ESPRESSO:

- **Expand** : Combine adjacent smaller cubes to form a larger cube, hence reducing the expression (by eliminating 1 or more variables).
- **Irredundant** : Remove redundant cubes i.e cubes that contain only points that are already contained by other cubes.
- **Reduce** : Separate a larger cube to form smaller cubes, hence expanding the expression (by adding 1 or more variables).

  Reduce operation does not directly help in minimizing the function, but it helps in exploring new expansion opportunities that can potentially be better than existing (discovered) expansions.

Functions to implement each of these operations have been implemented. The high level algorithm is described-

```
choice_exp = calc_weights (on_set)
on_set = expand(on_set, choice_exp)
on_set = irredundant(on_set)
while (cost > target and count < iter):
    choice_red = calc_weights (on_set)
    on_set = reduce (on_set, choice_red)
    choice_exp = calc_weights (on_set)
    on_set = expand (on_set, choice_exp)
    on_set = irredundant (on_set)
    cost = calc_cost (on_set)
    store on_set with least cost
return on_set, cost
```

Some details regarding the algorithm are elaborated below.

- Since any term can in the ON-set can be expanded, a weight vector is calculated to choose which of the terms should be expanded. The term which has the least weight is most likely to be combined with many other terms, hence it is chosen for expansion.
- Expansion is always done as deep as possible i.e, whatever terms can be expanded while not intersecting with the OFF-set are expanded.
- Similarly, a weight vector is is calculated to choose which of the terms should be reduced as well. The term which has the highest weight is most likely to be combined with many other terms, hence it is chosen for expansion.
- Unlike expansion, reduction process can have multiple options in terms of the depth of reduction i.e how far to reduce. For this, a decaying function is used. In the initial iterations, reduction is carried out fully but as the number of iterations increases, the depth of reduction is reduced. This is a heuristic used to add versatility in the algorithm.
- A target cost is taken as user input. If the target is reached, the algorithm stops iterating, but if the target is not reached, it will stop after the specified number of iterations and return the best solution found.
- Critical path delay i.e the maximum cost among all the next state functions and output functions is displayed.

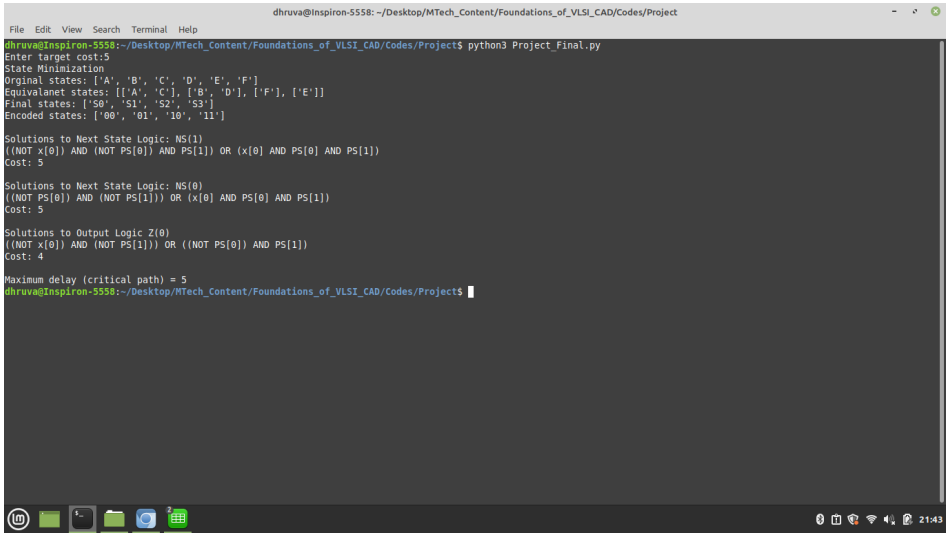
Example Illustration

An example FSM (state transition table) is shown. This is provided as input to the minimizer.

x	PS	NS	z
0	A	E	0
0	B	F	0
0	C	E	0
0	D	F	0
0	E	C	0
0	F	B	0
1	A	D	1
1	B	D	0
1	C	B	1
1	D	B	0
1	E	F	1
1	F	C	0

The target cost is specified to be 5.

Output of the minimizer is shown below.



Therefore, the tool will minimize both the number of states (i.e the number of flip-flops) as well as the combinational logic required to implement an FSM.