Amit Levi - amitlevi@campus.technion.a[redacted] Dan Shahak - [redacted]

236237 - Practical assignment 2

Part 1:

1. We've used the graph_algorithm plugin to find all the strongly connected components. The FSM cannot be too large, neither can it be of size 1. Therefor we drop all the components of size 1 and proceeded to pick the smallest component left.

2. We create a module from all the gates found in the previous step.

3. We filter the gates according to the GateTypeProperty which is either combinational or sequential. Thus, we're left with two groups of corresponding gates which we use to create the two requested submodules.

4. For each flip flop, we use the get_subgraph_function on the fanin net of the datapin and all the logic gates in front of the flip flops. The Boolean functions are as follows:

   ((! (((! net_321) & (! net_323)) | ((! net_325) | (! net_326)))) | (! ((! net_324) | (! ((! net_322) & (! net_323))))))

   ((! ((! (! net_324)) | ((! net_323) | (! (! net_322))))) | ((! net_323) & (! (! net_322))))

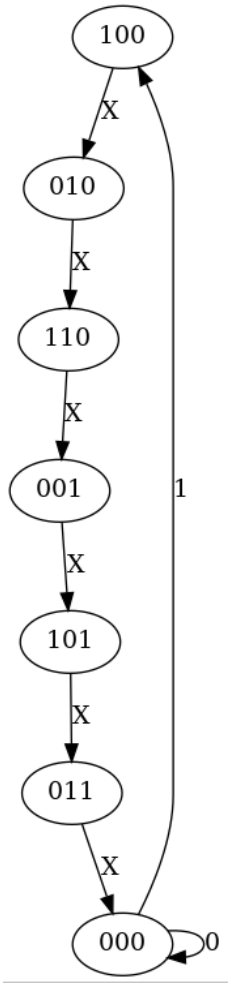
   ((! ((! net_323) | ((! net_322) | (! net_326)))) | ((! net_323) & (! (! net_324))))

   Where net_231 is the input signal and the rest are either the Qs or the QNs of the flip flops.

5. We brute-force all reachable states starting from the initial state of '000'. The following is a transition table first as the raw print (source, destination, signal) and then as a table:
   {('010', '110', '0'), ('110', '001', '1'), ('100', '010', '1'), ('001', '101', '1'), ('101', '011', '0'), ('000', '000', '0'), ('000', '100', '1'), ('001', '101', '0'), ('011', '000', '0'), ('101', '011', '1'), ('010', '110', '1'), ('011', '000', '1'), ('110', '001', '0'), ('100', '010', '0')}

SOURCE	SIGNAL	DESTINATION
010	0	110
110	1	001
100	1	010
001	1	101
101	0	011
000	0	000
000	1	100
001	0	101
011	0	000
101	1	011
010	1	110
011	1	000
110	0	001
100	0	010

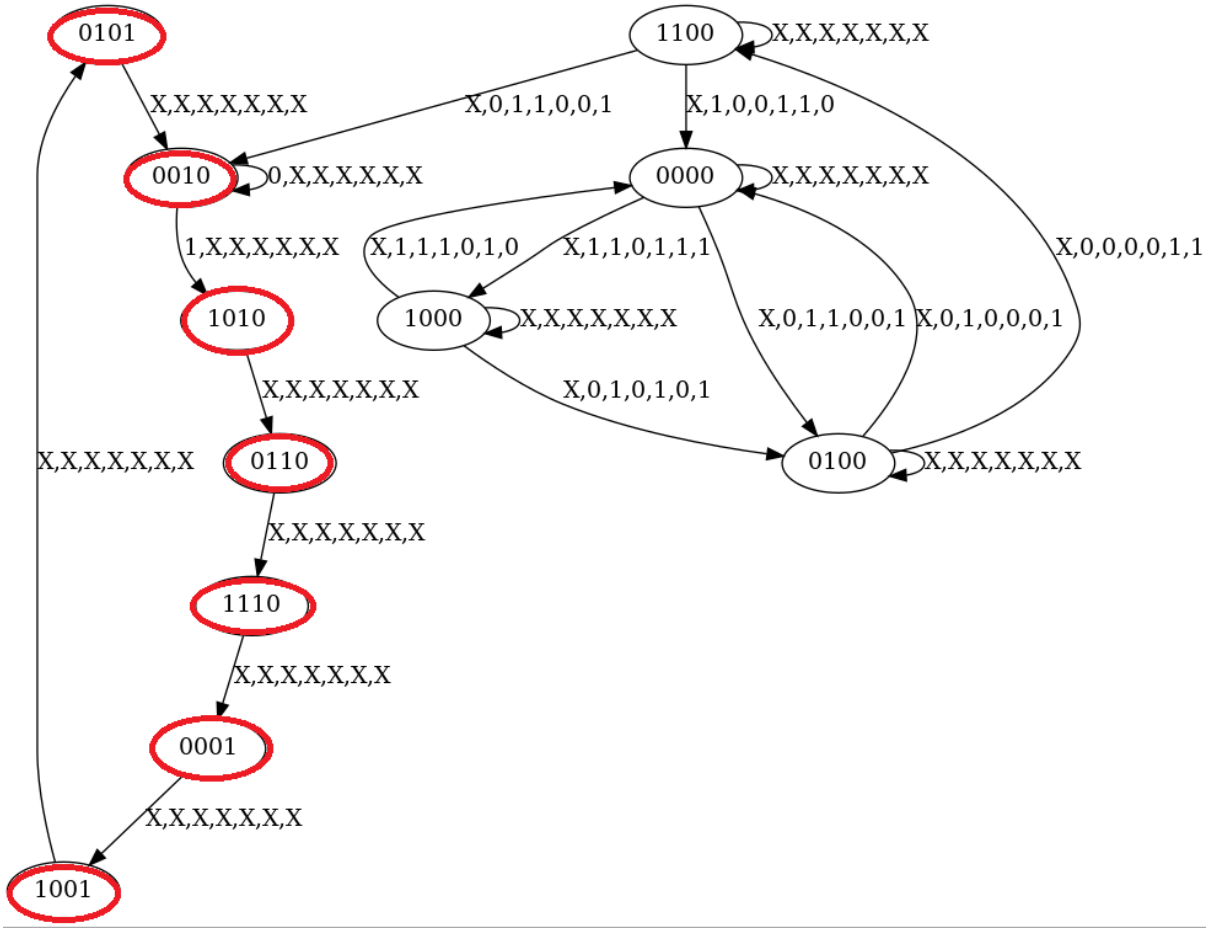
6. The visualized .dot FSM graph is:



7. The FSM controls the sequential steps of the ToyCipher's operation. The initial state (000) awaits a ONE signal from net_231. Once received, it runs through six sequential operations and then resets to its initial state, awaiting a ONE once more.

Part 2:

1. The input signals (according to their order in the graph) are:
   net_378, net_307, net_306, net_308, net_305, net_309, net_304.
   X denotes that the signal can be either 0 or 1 for the edge. Edges with explicit values are prioritized over plausible edges with X values. This was done for visualization purposes only due to the very large number of edges. (0000) is the initial state.



   The FSM has a total of 7 + 4 = 11 states. All the 7 states that have been marked in red belong to the original FSM whereas all the 4 non-marked states are states that belong to the obfuscated part of the FSM.

2. Oddly enough we got two paths from the initial state (0000) to our original FSM. Therefor there are two possible enabling keys.
   One is of length 3: (0, 1, 1, 0, 0, 1), (0, 0, 0, 0, 1, 1), (0, 1, 1, 0, 0, 1)
   The other of length 4: (1, 1, 0, 1, 1, 1), (0, 1, 0, 1, 0, 1), (0, 0, 0, 0, 1, 1), (0, 1, 1, 0, 0, 1)

   We believe that the second is more likely to be correct, as the skipping edge from 0000 to 0100 could be explained by an abundance of signals in our analysis.

   Where each step describes the signals for:
   (net_307, net_306, net_308, net_305, net_309, net_304)

   As net_378 is the net that controls the original FSM.