Algorithm Final Project Report

Li-Wei Chen

B04901014 EE3

1 Introduction

Given the input/output bit string $b_1 b_2 \cdots b_n$, where b_i is the i^{th} input/output pair and a the FSM $(\Sigma, S, s_0, \delta, F)$, we now needs to embed a specified watermark sequence, utilizing the unspecified transitions. Details are covered in [1].

We present an greedy algorithm, starting from computing the maximum length one may go from each $s \in S$ with the input and output relation specified by $b_i b_{i+1} \cdots b_n$, we greedily choose the max of them.

The maximum length is defined as following. Not taking the unspecified transition into consideration, the maximum length is recorded if there is no further path satisfing the input bit string. We also add a constraint, if the one candidate stops at b_j, the terminate state for the it must have a unspecified transition for b_{j+1} to be the maximum length path, expect for when the last input is b_n. If their are multiple states that holds the same length, we choose one randomly.

Figure 1 shows adding a sequence watermark to a existing FSM. the input sequence of the watermarks is 00, 11, 00, 10, 00, and we expect the output to be 0, 0, 0, 1, 1. Hence the red dashed transitions is inserted to the FSM.

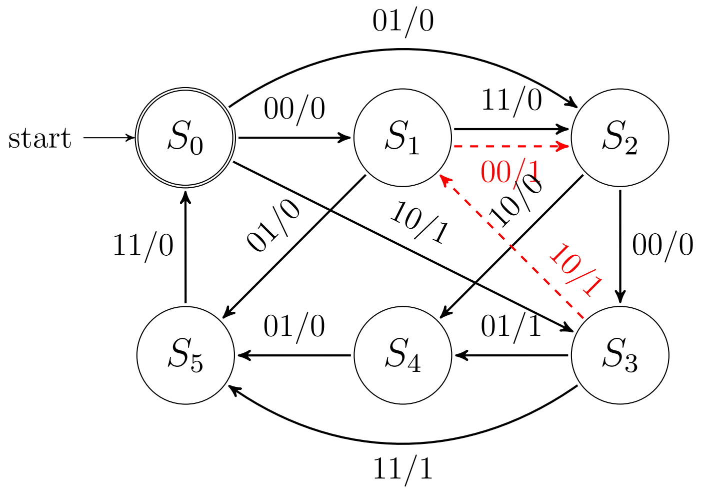

Figure 1: Adding watermark to a FSM

2 Algorithm

2.1 Greedy Algorithm

Starting from b_0, we first choose the maximum length state as the first state, then if the maximum path stops at b_i, we then choose the maximum length state for b_{i+2} as the second state. Using b_j as the augmented trasition from the first state to the second state.

A new state is inserted if all the states has zero maximum length and the required input is already specified for all states. Suppose the next input/output pair is b_i, then we use b_i as transition to the new state, and b_{i+1} as the new input/output pair(transition) and continue the algorithm. Once a new transition or state is add to the graph, they have no difference from the predefined ones. That is, the algorithm will also take them into consideration.

To take the added transition into consideration, we first add the transition into the graph then run the find max length algorithm. Then we delete it right after the algorithm is done to avoid interference to the next state to run. The transition to the max length state will be added afterwards. The pseudocode is on Figure 3.

2.2 Greedy Choice Property

In this section we'll prove the greedy choice property in the loop invariance. First note that once we choose a max length path, we cannot make transitions during the path without reaching the end. This is simply because the next input is occupied to transit to the next state on the path. Hence we can only discuss the situation of choosing to transit to different states when we are at the end.

Let the maximum length state s with max length m, Suppose there is a minimum cost path is choose to transit to $s'_1 \neq s_1$ with max length $m'_1 < m_1$, we know for the first $m_1 + 1$ inputs, $Cost(s_1) = 1$ and $Cost(s'_1) \geq 1$, thus $Cost(m) \leq Cost(m')$. Namely, for the $m_1 + 2$ to the $2m_1 + 2$ inputs, $Cost(s'_2) \geq 1$, otherwise we'll choose s'_2 instead of s_2. Iteratively, same for $\sum\limits_{i=1}^{n-1} m_i + i + 1$ to $\sum\limits_{i=1}^{n-1} 2m_i + i + 1$ and $Cost(s_i) \leq Cost(s'_i)$.

Since the minimum of all cost can be broken into sum of minimum of cost of each fraction, we can conclude that the maximum length choise s_i is the minimum cost. Also note that adding new state is equivalent to $Cost(s'_i) = 2$ and $m'_i = 0$, since the new state have no possible transition to have a path.

2.3 Initial State

Since we can run any latent state we want before we start the watermark sequence, we have to deal with how to choose the first state. We first again choose the max length state starting from the first input, and then the greedy choice property mentioned in the previous section can be applied.

Nevertheless, this time we have no gaurantee that we can find the state satisfying the max length property. Thus, if there is no qualified state, we first search for the state that the transition for the first input is free. If it exists, use the free transition to add a new state. Otherwise, search a random free transition and add a new state on it. The pseudocode is on Figure 2.

Algorithm 1 FindInitialState

```
1: $G$: Input FSM, $(\Sigma, S, s_0, \delta, F)$
2: $b^i$: Input bit string
3: $b^o$: Output bit string
4: $j = 0$, $start = resetstate$
5: $maxlen = 0, dest = start$, $ns = null$
6: for each s in G.S do
7:     $m, d =$ FindMaxLength($s$, $b^i$, $b^o$, 0)
8:     if $m \geq maxlen$ and $d \neq null$ then
9:         $maxlen = m$
10:        $ns = d$
11:        $dest = s$
12:    end if
13: end for
14: if $nst \neq null$ then
15:     $start = ns$
16: else
17:     $start =$ FreeTransition($b^i_0$, $b^o_0$)
18:                                 ▷ Search for free transition with the first I/O
19:     if $start = null$ then
20:         $freestate, trans =$ GetFreeTransition($G$)
21:                                 ▷ Search an arbitrary free transition
22:         $start =$ AddNewState($freestate$, $trans$, $G$)
23:                                 ▷ Add a new state with the free transition
24:     end if
25: end if
26: return $start, maxlen$
```

Figure 2: Psedocode for finding the initial state

Algorithm 2 Watermark

```
1: $G$: Input FSM, $(\Sigma, S, s_0, \delta, F)$
2: $b^i$: Input bit string
3: $b^o$: Output bit string
4: $j = 0$, $start = resetstate$
5: $start, j$ = FindInitialState()                ▷ Find the Optimal Initial State
6: while $j \neq b^i.length - 1$ do
7:     $maxlen = 0, dest = start$, $ns = null$
8:     for each s in G.S do
9:         AddTransition($start$, $dest$, $b^i_j$, $b^o_j$)
10:        $m, d$ = FindMaxLength($s$, $b^i$, $b^o$, $j + 1$)
11:        DeleteTransition($start$, $b^i_j$)
12:        if $m \geq maxlen$ and $d \neq null$ then
13:            $maxlen = m$
14:            $ns = d$
15:            $dest = s$
16:        end if
17:    end for
18:    if $nst \neq null$ then
19:        AddTransition($start$, $dest$, $b^i_j$, $b^o_j$)
20:    else
21:        $ns$ = AddNewState($dest$, $b^i_j$, $G$)         ▷ Add state and transition
22:    end if
23:    $j = j + maxlen + 1$
24:    $start = ns$
25: end while
26: if $j = b^i.length - 1$ then                          ▷ Last one to add
27:     AddTransition($start$, $start$, $b^i_j$, $b^o_j$)               ▷ Self loop
28: end if
```

Figure 3: Psedocode for the watermark algorithm

2.4 CSFSM Detection

The detection of CSFSM is after the data is loaded and the graph is constructed, we run over each state the check if the out transitions span the whole possible inputs. If yes, the program will report that a CSFSM is detected.

2.5 Time Complexity Analysis

Let the number of bits of the input be n_i, that of the outputs be n_o, the number of states S, and the total number of transitions we have to make is $N = \lceil \frac{128}{n_i+n_o} \rceil$. The comlexity of $FindMaxLength$ is $O(maxlen)$, however, experiments shows that the max length is often close to 0. Thus $O(NS) \approx O(S)$. Since search for free transitions requires running all possible bits of inputs, which is $O(2^{n_i})$, thus the complexity of the $FindInitialState$ is $O(2^{n_i}+S)$. Thus the total algorithm takes $O(2^{n_i}+(2S+N+1)) \approx O(2^{n_i}+S)$ for $S \gg N$.

Since $n_i+n_o \geq 2$, $N \leq 64$, and $n_i \leq 127$, we can get rid of the N and n_i terms, making our algorithm $O(S)$, a linear time algorithm. Nonetheless, still, as n_i grows, the complexity will grow exponentially by the $FindInitialState$. There may be a better implementation that exchanges memory for complexity(i.e. store the free transitions seperately and maintain them) we may try in the future.

3 Don't-cares Optimization

We find that the optimization of merging transitions into don't-cares is really important. The importance lies in the transition cost is defined by the number of transitions including don't-cares. That is, the cost of 1--- is one instead of eight.

Since the transitions are given in the input kiss file, we can make the best use of the don't-cares already optimized by the officials. Upon reading in the kiss file, the don't care strings are stored in a STL map(implemented in a RB tree), mapping transitions to a vector of strings.

We implement the map key of transitions by the output value and the next state index, after the main algorithm is finished, the transitions will be push into the map one by one. Once a transition is pushed into the vector of its key, it will search for the possible merging of the previous bits. We perform inplace merging, since cannot afford recursive adding new bitstrings to the vector, which consumes at most 3^{2n_i} memory.

We are not confirmed to find the optimal solution, since the inplace property will let us loss the imformation to check for merging. For instance, 010-1, 0-000, -1010 can merge to produce 010--, however, we cannot find a non-heuristic algorithm to find such pairs.

Nonetheless, by including the original transition in the input kiss file, we are guaranteed not to have extra transitions for the original circuit. We can get benefits if the added transitions can be merged into the original circuit to produce don't-cares, or the original kiss file is not optimal and our merging algorithm find it out.

4 Experiments

Sadly, we didn't manage to attend the contest, thus we can only use the four kiss file, three md5 file to perform experiments. For the space concern, we will only show the results on `t2.kiss` with `md5_1.dat, md5_2.dat, md5_3.dat` added to each FSM. The result is as followings.

```
          INPUT                          OUTPUT
.i 9                           .i 9
.o 10                          .o 10
.p 161                         .p 180
.s 30                          .s 31
.r S0                          .r S0
1-0000--- S0 S0 0000000000     000100101 S0 S0 0100100000
1--11---- S0 S0 0000000000     1-0000--- S0 S0 0000000000
1-0001--- S0 S1 1010100110     1--11---- S0 S0 0000000000
1-0010--- S0 S10 1000000010    1-0001--- S0 S1 1010100110
1-0011--- S0 S10 1001001101    1-0010--- S0 S10 1000000010
1-010---- S0 S10 1000000001    1-0011--- S0 S10 1001001101
```

```
1-1-0---- S0 S12 0000000000   1-010---- S0 S10 1000000001
1-10----- S0 S12 0000000000   1-1-0---- S0 S12 0000000000
------0-- S1 S2 0001100110    1-10----- S0 S12 0000000000
------1-- S1 S8 0010101000    010011001 S0 S27 0110101000
00------- S2 S0 0111111110    000110101 S0 S30 1000110100
01------- S2 S0 0110101000    001111001 S0 S30 1111111111
10-11---- S2 S0 0000000000    ------0-- S1 S2 0001100110
11-11---- S2 S0 0010111000    ------1-- S1 S8 0010101000
1-0001--- S2 S1 0110000100    00------- S2 S0 0111111110
1-00000-- S2 S2 0100000000    01------- S2 S0 0110101000
1-0011--- S2 S2 0101000000    10-11---- S2 S0 0000000000
1-010---- S2 S2 0100000000    11-11---- S2 S0 0010111000
1-1-0---- S2 S2 0100000000    1-0001--- S2 S1 0110000100
1-10----- S2 S2 0100000000    1-00000-- S2 S2 0100000000
1-0010--- S2 S3 0100000000    1-0011--- S2 S2 0101000000
1-00001-- S2 S7 0100000000    1-010---- S2 S2 0100000000
0-------- S3 S0 1010010111    1-1-0---- S2 S2 0100000000
1--11---- S3 S0 0000011001    1-10----- S2 S2 0100000000
1-0000--- S3 S3 0100000000    1-0010--- S2 S3 0100000000
1-0011--- S3 S3 0101000000    1-00001-- S2 S7 0100000000
1-010---- S3 S3 0100000000    0-------- S3 S0 1010010111
1-1-0---- S3 S3 0100000000    1--11---- S3 S0 0000011001
1-10----- S3 S3 0100000000    1-0000--- S3 S3 0100000000
1-0010--- S3 S4 0110001000    1-0011--- S3 S3 0101000000
1-0001--- S3 S5 0110001100    1-010---- S3 S3 0100000000
--------- S4 S6 0110010000    1-1-0---- S3 S3 0100000000
--------- S5 S1 0110010100    1-10----- S3 S3 0100000000
0-------- S6 S0 0000000000    1-0010--- S3 S4 0110001000
1--11---- S6 S0 0000000000    1-0001--- S3 S5 0110001100
1-0001--- S6 S1 0110000100    --------- S4 S6 0110010000
1-0010--- S6 S4 0110001000    --------- S5 S1 0110010100
1-00000-- S6 S6 0100000000    0-------- S6 S0 0000000000
1-0011--- S6 S6 0101000000    1--11---- S6 S0 0000000000
1-010---- S6 S6 0100000000    1-0001--- S6 S1 0110000100
1-1-0---- S6 S6 0100000000    1-0010--- S6 S4 0110001000
1-10----- S6 S6 0100000000    1-00000-- S6 S6 0100000000
1-00001-- S6 S29 1100000000   1-0011--- S6 S6 0101000000
00------- S7 S0 0000000000    1-010---- S6 S6 0100000000
```

```
01------- S7 S0 0010100000    1-1-0---- S6 S6 0100000000
10-11---- S7 S0 0010000000    1-10----- S6 S6 0100000000
11-11---- S7 S0 0010100000    1-00001-- S6 S29 1100000000
1-0000--- S7 S7 1100000000    00------- S7 S0 0000000000
1-0011--- S7 S7 1101000000    01------- S7 S0 0010100000
1-010---- S7 S7 1100000000    10-11---- S7 S0 0010000000
1-1-0---- S7 S7 1100000000    11-11---- S7 S0 0010100000
1-10----- S7 S7 1100000000    1-0000--- S7 S7 1100000000
1-0001--- S7 S8 0110100100    1-0011--- S7 S7 1101000000
1-0010--- S7 S29 1100000000   1-010---- S7 S7 1100000000
--------- S8 S9 0110010000    1-1-0---- S7 S7 1100000000
0-------- S9 S0 0010100000    1-10----- S7 S7 1100000000
1--11---- S9 S0 0010100000    1-0001--- S7 S8 0110100100
1-0001--- S9 S8 0110000100    1-0010--- S7 S29 1100000000
1-00000-- S9 S9 0100000000    --------- S8 S9 0110010000
1-010---- S9 S9 0100000000    0-------- S9 S0 0010100000
1-1-0---- S9 S9 0100000000    1--11---- S9 S0 0010100000
1-10----- S9 S9 0100000000    1-0001--- S9 S8 0110000100
1-00001-- S9 S28 0100100000   1-00000-- S9 S9 0100000000
0-------- S10 S0 0000000000   1-010---- S9 S9 0100000000
1---0--10 S10 S0 0000000000   1-1-0---- S9 S9 0100000000
1--0---10 S10 S0 0000000000   1-10----- S9 S9 0100000000
1--11---- S10 S0 0000000000   1-00001-- S9 S28 0100100000
1---0--00 S10 S10 0000000000  0-------- S10 S0 0000000000
1--0---00 S10 S10 0000000000  1---0--10 S10 S0 0000000000
1---0---1 S10 S11 0000000000  1--0---10 S10 S0 0000000000
1--0----1 S10 S11 0000000000  1--11---- S10 S0 0000000000
0-------- S11 S0 0000000000   1---0--00 S10 S10 0000000000
1---0--1- S11 S0 0000000000   1--0---00 S10 S10 0000000000
1--0---1- S11 S0 0000000000   1---0---1 S10 S11 0000000000
1--11---- S11 S0 0000000000   1--0----1 S10 S11 0000000000
1--010-0- S11 S10 0100000000  0-------- S11 S0 0000000000
1---0--0- S11 S11 0000000000  1---0--1- S11 S0 0000000000
1--011-0- S11 S11 0000000000  1--0---1- S11 S0 0000000000
0-------- S12 S0 0000000000   1--11---- S11 S0 0000000000
1--11---- S12 S0 0000000000   1--010-0- S11 S10 0100000000
1-0001--- S12 S10 1000000001  1---0--0- S11 S11 0000000000
1--000--- S12 S12 0000000000  1--011-0- S11 S11 0000000000
```

```
1--011--- S12 S12 0000000000  0-------- S12 S0 0000000000
1--10---- S12 S12 0000000000  1--11---- S12 S0 0000000000
1--010--- S12 S13 0000000000  1-0001--- S12 S10 1000000001
0-------- S13 S0 0000000000   1--000--- S12 S12 0000000000
1--11---- S13 S0 0000000000   1--011--- S12 S12 0000000000
1--000--- S13 S13 0000000000  1--10---- S12 S12 0000000000
1--01---- S13 S13 0000000000  1--010--- S12 S13 0000000000
1--001--- S13 S14 0000000001  0-------- S13 S0 0000000000
1--10---- S13 S14 0000000001  1--11---- S13 S0 0000000000
0-------- S14 S0 1000110100   1--000--- S13 S13 0000000000
1--11---- S14 S0 0000000000   1--01---- S13 S13 0000000000
1--000--- S14 S14 0000000000  1--001--- S13 S14 0000000001
1--01---- S14 S14 0000100100  1--10---- S13 S14 0000000001
1--10---- S14 S14 0000000000  0-------- S14 S0 1000110100
1--001--- S14 S15 0010100100  1--11---- S14 S0 0000000000
--1------ S15 S16 0010010000  1--000--- S14 S14 0000000000
--0------ S15 S22 0010010000  1--01---- S14 S14 0000100100
0-------- S16 S0 0111011001   1--10---- S14 S14 0000000000
1--000--- S16 S16 0000000000  1--001--- S14 S15 0010100100
1--011--- S16 S16 0000000000  --1------ S15 S16 0010010000
1--1----- S16 S16 0000000000  --0------ S15 S22 0010010000
1--010--- S16 S17 0000000000  0-------- S16 S0 0111011001
1--001--- S16 S19 0010000100  1--000--- S16 S16 0000000000
0-------- S17 S0 0000000000   1--011--- S16 S16 0000000000
1--000--- S17 S17 0000000000  1--1----- S16 S16 0000000000
1--011--- S17 S17 0000000000  1--010--- S16 S17 0000000000
1--1----- S17 S17 0000000000  1--001--- S16 S19 0010000100
1--001--- S17 S18 0010001100  0-------- S17 S0 0000000000
1--010--- S17 S20 0010001000  1--000--- S17 S17 0000000000
--------- S18 S19 0010010100  1--011--- S17 S17 0000000000
--------- S19 S16 0010010000  1--1----- S17 S17 0000000000
--------- S20 S21 0010010000  1--001--- S17 S18 0010001100
0-------- S21 S0 0000100100   1--010--- S17 S20 0010001000
1--001--- S21 S19 0010000100  --------- S18 S19 0010010100
1--010--- S21 S20 0010001000  --------- S19 S16 0010010000
1--000--- S21 S21 0000000000  --------- S20 S21 0010010000
1--011--- S21 S21 0000000000  0-------- S21 S0 0000100100
1--1----- S21 S21 0000000000  1--001--- S21 S19 0010000100
```

```
0-------- S22 S0 0000000000   1--010--- S21 S20 0010001000
1-011---- S22 S0 0000000000   1--000--- S21 S21 0000000000
1-0011--- S22 S10 1000000011  1--011--- S21 S21 0000000000
1-1------ S22 S12 0000000000  1--1----- S21 S21 0000000000
1-0000--- S22 S22 0011100111  0-------- S22 S0 0000000000
1-0010--- S22 S23 0000000000  100100101 S22 S0 1001010100
1-0001--- S22 S25 0010000100  1-011---- S22 S0 0000000000
0-------- S23 S0 0000000000   1-0011--- S22 S10 1000000011
1-011---- S23 S0 0000000000   1-1------ S22 S12 0000000000
1-0011--- S23 S10 1000000011  1-0000--- S22 S22 0011100111
1-1------ S23 S12 1001000000  1-0010--- S22 S23 0000000000
1-0000--- S23 S23 0000000000  1-0001--- S22 S25 0010000100
1-010---- S23 S23 0000000000  0-------- S23 S0 0000000000
1-0001--- S23 S24 0010001100  1-011---- S23 S0 0000000000
1-0010--- S23 S26 0010001000  1-0011--- S23 S10 1000000011
--------- S24 S25 0010010100  1-1------ S23 S12 1001000000
--------- S25 S22 0010010000  1-0000--- S23 S23 0000000000
--------- S26 S27 0010010000  1-010---- S23 S23 0000000000
0-------- S27 S0 0000000000   1-0001--- S23 S24 0010001100
1-011---- S27 S0 0000000000   1-0010--- S23 S26 0010001000
1-1------ S27 S12 0000000000  --------- S24 S25 0010010100
1-0001--- S27 S25 0010000100  --------- S25 S22 0010010000
1-0010--- S27 S26 0010001000  --------- S26 S27 0010010000
1-0000--- S27 S27 0000000000  0-------- S27 S0 0000000000
1-010---- S27 S27 0000000000  1-011---- S27 S0 0000000000
0-------- S28 S0 0010100000   1-1------ S27 S12 0000000000
1--11---- S28 S0 0010100000   100011100 S27 S22 1101101010
1-0001--- S28 S8 0110000100   1-0001--- S27 S25 0010000100
1-010---- S28 S10 1100000000  1-0010--- S27 S26 0010001000
1-0000--- S28 S28 1100000000  1-0000--- S27 S27 0000000000
1-001---- S28 S28 1100000000  1-010---- S27 S27 0000000000
1-1-0---- S28 S28 1100000000  0-------- S28 S0 0010100000
1-10----- S28 S28 1100000000  1--11---- S28 S0 0010100000
0-------- S29 S0 1111111111   1-0001--- S28 S8 0110000100
1--11---- S29 S0 0000000000   1-010---- S28 S10 1100000000
1-0001--- S29 S8 0110100100   1-0000--- S28 S28 1100000000
1-0000--- S29 S29 1100000000  1-001---- S28 S28 1100000000
1-0010--- S29 S29 1100000000  1-1-0---- S28 S28 1100000000
```

```
1-0011--- S29 S29 1101000000
1-010---- S29 S29 1001010100
1-10----- S29 S29 1100000000
.e
```

```
1-10----- S28 S28 1100000000
0-------- S29 S0 1111111111
1--11---- S29 S0 0000000000
1-0001--- S29 S8 0110100100
1-0000--- S29 S29 1100000000
1-0010--- S29 S29 1100000000
1-0011--- S29 S29 1101000000
1-010---- S29 S29 1001010100
1-10----- S29 S29 1100000000
110010010 S30 S16 0000100100
001001101 S30 S30 0100011001
010001111 S30 S30 0000100100
011000101 S30 S30 1000010101
011011110 S30 S30 1010011100
100100000 S30 S30 0000111111
100111010 S30 S30 1100000000
110000011 S30 S30 0011100111
110010111 S30 S30 0010101001
110110110 S30 S30 0101000000
111101001 S30 S30 0110000000
111101111 S30 S30 0111101010
111110111 S30 S30 1001000000
.e
```

The debug message of running `t2.kiss`

```
000001000 1010010111
000100101 0100100000
010011001 0110101000
100011100 1101101010
100100101 1001010100
001111001 1111111111
110110110 0101000000
Start State : s0
Maxlen : 1 Destination : s3 Next State : s0
Maxlen : 0 Destination : s0 Next State : s0
Add Transition from s0 to s0 with 000100101 0100100000
Maxlen : 0 Destination : s27 Next State : s27
Add Transition from s0 to s27 with 010011001 0110101000
```

```
Maxlen : 0 Destination : s22 Next State : s22
Add Transition from s27 to s22 with 100011100 1101101010
Maxlen : 0 Destination : s0 Next State : s0
Add Transition from s22 to s0 with 100100101 1001010100
Maxlen : 0 Destination : s0 Next State None
Adding state s30
Add Transition from s0 to s30 with 001111001 1111111111
Add Transition from s30 to s30 with 110110110 0101000000

110010111 0010101001
011011110 1010011100
100100000 0000111111
111101111 0111101010
001001101 0100011001
011000101 1000010101
100111010 1100000000
Start State : s30
Maxlen : 0 Destination : s30 Next State : s30
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 110010111 0010101001
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 011011110 1010011100
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 100100000 0000111111
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 111101111 0111101010
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 001001101 0100011001
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 011000101 1000010101
Add Transition from s30 to s30 with 100111010 1100000000

010001111 0000100100
111110111 1001000000
110000011 0011100111
110010010 0000100100
011001111 0111011001
000110101 1000110100
```

```
111101001 0110000000
Start State : s30
Maxlen : 0 Destination : s30 Next State : s30
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 010001111 0000100100
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 111110111 1001000000
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s30 to s30 with 110000011 0011100111
Maxlen : 1 Destination : s16 Next State : s0
Add Transition from s30 to s16 with 110010010 0000100100
Maxlen : 0 Destination : s30 Next State : s30
Add Transition from s0 to s30 with 000110101 1000110100
Add Transition from s30 to s30 with 111101001 0110000000
```

We can see that the gain from *maxlen* is only 2. This is because as the number of output bits gets bigger, we have a smaller chance to get the right output specified by the md5 file. We can see the cost is adding $(31-30) = 1$ state and adding $(180-161) = 19$ transitions, making the total cost $1+19 = 20$.

For further experience results, please visit our github and play with our code. It's easy to use and you can use `make dbg` to get the debug messages for seeing how the algorithm works.

5 Future Works

There can be done more for the Don't-cares Optimization, and the key lies in the unspecified transitions. If we can make the most use of them, specifying the optimal output for them, then they can serve as the glue to merge some transitions into a single don't care transition. Nevertheless, this part is not that easy, and we can't find any existing algorithm for solving this.

We are currently in the phase of testing the correctness of the code, finding either there are bugs in the code or in the algorithm. We may further use verilog as the official recommanded, however, since we didn't participate the contest, we have to find the software on our own.

Also, the input size is constraint in 64 bits, while the input provided from the contest may go to 127 bits. This is because the maximum number we can use in the computer is $2^{64} - 1$, which is the unsigned long to be the array index. We can use a new defined data structure(i.e. concat two arrays) to relax the constraint, but we haven't do so yet.

We also need to change to output the latent code if their are any, it's trivial compared to the main algorithm. We'll done it before the final demo since it's necessary for the verilog to verify the correctness.

6 Contributions

All our codes are in `https://github.com/b04901014/AlgFinalProject`, there may be some adjustments after we upload the source code to ceiba, thus the code in the url is the newest.

The teamwork is organized as following, Guan-Yu Chen is responsible for the presentation on 19th June, 2018, while all the other works are done by Li-Wei Chen(coding, reports, etc.). Guan-Yu Chen also give a few advice to form the algorithm. Shan-Yuan Zheng did nothing. More detailed teamwork contributions can be seen in the github commitment record.

7 References

[1] Chip Implementation Center,*"Watermarking based Intellectual Property Core Protection Scheme"*,ICCAD Contest Problem D,June 2018.