6 State Encoding

- Till now some state assignment with no discussion of alternatives has been assumed. The number of possible binary state representations for a problem is quite large. For example the number of state assignments for a four-state machine is 24. It is quite obvious that the different assignments may yield different hardware.

- The appropriate choice of state assignments can reduce the required number of logic gates needed to implement the excitation and output switching functions. The number of D-FFs \( n \) is related to the number of states \( m \) in the circuit:

\[
2^{n-1} < m \leq 2^n
\]

There will be

\[
N_{SA} = \frac{2^n}{(2^n - m)!}
\]

ways of assigning \( 2^n \) combinations of binary state representations to \( m \) states.
• It can be shown that not all these assignments are unique with respect to resulting logic equations. The size of sum of products representation is invariant under permutation and complementation of encoding bits. Hence the number of possible unique assignments $N_{UA}$ is given by (comp. De Micheli):

$$N_{UA} = (2^n - 1)!/(2^n - m)!n!.$$ 

• For more than four states a complete enumeration is impractical so techniques for choosing a good assignment are necessary. Criteria will be minimal gate count and/or small propagation delay.

• Three different guidelines for state encoding are presented:
  - Minimum bit change (i.e. Gray code, O'Brien code, Johnson counter)
  - Prioritised adjacency (neighbourhood of states in a state assignment K-map)
  - One-hot method (one D flip-flop for each state)
6.1 Popular Strategies

6.1.1 Minimum Bit Change

- Binary values will be assigned to the states in a sequence that the number of all bit changes during state transitions becomes a minimum. This strategy supports two level logic with sum of products representation (CPLD). For a bit change with a reset of a D-FF at least an AND-gate and an extra OR-gate input is necessary in order to perform the state feed back. These gates will be saved if a bit change is avoided.

Sequential-Encoding

- Total weight of bit changes: 6

Minimum-Bit Change

- Total weight of bit changes: ....
6.1.2 Prioritised Adjacency

- There are two ways to arrange the ones on a K-map (based on z-rule minterm numbers) for next state forming logic minimisation:
  - Vertically by making the ones combine within a given column.
  - Horizontally by making the ones combine within a given row.
  In both cases the ones are placed in adjacent K-map cells which differ only in one coordinate.

- This grouping of ones can be obtained by adopting the following state assignment rules:
  - **Rule 1**: States that have the same next state for a given input should be given logically adjacent assignments.
  - **Rule 2**: States that are the next states of a single present state under logically adjacent inputs should be given logically adjacent assignments.
  - **Rule 3**: States that generate the same output for a given input should be given a logically adjacent assignment.
Example for a state encoding with rules 1 and 2 for prioritised adjacency

Given state diagram

State diagram with assignments

Adjacencies!

Rule 1: (A, B), ...
Rule 2: ...
Rule 3: ...

State assignment KV-map:

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
• **Rule 1:** States A and B should be adjacent because they both go to state C under input 0. The same is valid for state A and C.

• **Rule 2:** Yields A adj B, A adj C, B adj D and C adj D.

• The final assignments will by found by plotting the states on state assignment K-maps, with each cell representing the state bit combination assigned to one state of the FSM. Adjacent states are identified on these K-maps in the same manner as adjacent product terms.

• In general it is not always possible to satisfy all adjacencies suggested by rules 1 and 2. In case of conflicts assignments according **rule 1 should be** preferred in order to achieve a larger number of ones which can be grouped.
6.1.3 One-Hot Encoding

- A one-hot state assignment uses **one state bit and therefore one flip-flop per state**. All the state bits except one are equal to zero at any given time. The single state bit that is equal to 1 is called the hot state.

- Therefore a n-state sequential circuit requires n state bits. The resulting circuit contains **more flip-flops** than other encoding methods and hence one-hot encoding is favoured for FSM implementations with FPGAs which offer a larger number of flip-flops than CPLDs.

- Also this may appear to be a waste of flip-flops this method provides for a **more simple next state forming logic and output circuit** of FSMs. Less stages of logic are necessary for control of a D-type flip-flop input which represents the next state bit. With fewer concatenated logic gates a **higher clock frequency** will be realisable. In general the total amount of logic gates won't be less in comparison to the other encoding method results.
Less logic noise is possible in the FSMs output signals. Since the output functions only involve two coupled state bits internally initiated static hazards are less possible.

- In the case of a FSM controller which has to perform a device selection in the kind of "1 out of n" a one-hot encoding will be favourable because not output logic is needed. With each state which becomes hot a different device can be enabled directly with the state bit (namely the D-FF output).

- One important draw back of one-hot encoding is caused by the large number of unused states. For mission critical FSM controllers the designer has to ensure that a transition from all so called pseudo states to a determined reset state will take place.

- The impact of one-hot encoding on a simple FSM logic can be demonstrated with the one-to-one transformation from ASM chart elements to digital hardware elements:
A control signal output within a state box can be transformed into an OR-gate whose inputs are bits of states in which the output is asserted.

Multiple inputs to a state can be transformed to an OR-gate at the next state D-FF input.

A condition box can be transformed into pair of AND-gates.
6.2 Safe FSM with no Lock Up

- It is possible in VHDL to code a FSM with unused states. If \( n \) flip-flops are implemented and the FSM works with \( m \) states and \( m \) will be less than \( 2^n \) then a number of unused states will exist.

- In order to support a safe FSM with no lock up in undefined states the case-statement has to be closed with a when others clause.

- The coding style determines whether unreachable states will be recognized in order to achieve quality of results:
  - **Minimal risk with safest FSM**: All undefined states are identified and the derived next state forming logic will perform a transition to a reset state which is assigned with when others.
  - **Minimal cost and fastest FSM**: It will be assumed that no unused state will ever be reached. The next state forming logic will then be minimised with no safe transition from pseudo states.
To be independent from a synthesis tool's properties and performance it is suggested to overcome the problem of unpredictable states with following recommendations:

- Define the number of enumeration for states of the FSM to be a power of two.
- In the case statement of the FSM use the when others statement to define recovery for unexpected states.
- For mission critical designs use a kind of binary encoding instead of one-hot if there are any doubts about recovery from unpredicted states.
6.3 State Assignment in VHDL

- With the definition of an enumeration type for state signals synthesis tools will perform a binary state encoding with a rising sequence of bit combinations from left to right:
  
  State encoding: 0000 0001 0010 0011 ... ...1000
  
  ```
  type STATE_TYPE is (IDLE, GNT0, GNT1, GNT2, GNT3, GNT4, GNT5, GNT6, GNT7);
  ```

- If one-hot encoding is selected with the synthesis options then a single state bit will be equal to one beginning with the LSB of the state assignment (from left to right):
  
  State encoding: 000000001 000000010 ... 100000000
  
  ```
  type STATE_TYPE is (IDLE, GNT0, ... GNT7);
  ```

- Explicite state encodings within the VHDL code can be described with an user-defined attribute `ENUM_ENCODING` which is supported by synthesis tools. The first step is to declare the attribute as a string and in the second step the string has to be defined in order to perform the assignment:
  
  ```
  type STATE_TYPE is (IDLE, GNT0, GN1, GNT2, GNT3, GNT4, GNT5, GNT6, GNT7);
  attribute ENUM_ENCODING: string; -- property declaration
  attribute ENUM_ENCODING of STATE_TYPE: type is -- property definition
  "0000 1000 1100 1101 1001 1010 1110 1111 1011";
  signal STATE, NEXT_STATE: STATE_TYPE;
  ```
6.4 Application of a Synthesis Tool (Xilinx ISE 6.2)

Compact
Minimizes the number of state variables and flip-flops. This technique is based on hypercube immersion. Compact encoding is appropriate when trying to optimize area.

Sequential
Identifies long paths and applies successive radix two codes to the states on these paths. Next state equations are minimized.

Gray
Guarantees that only one state variable switches between two consecutive states. It is appropriate for controllers exhibiting long paths without branching. In addition, this coding technique minimizes hazards and glitches. Very good results can be obtained when implementing the state register with T or JK flip-flops.

Johnson
Similar to the Gray option, shows benefits with state machines containing long paths with no branching.
Synthesis of a Polling Circuit with 8 Clients

entity ARBITER_8 is
port(   CLK, RESET : in bit;
    R: in bit_vector(7 downto 0);  -- device requests
    G: out bit_vector(7 downto 0));  -- device grants
end ARBITER_8;
architecture BEHAVIOUR of ARBITER_8 is
type STATE_TYPE is (IDLE, GNT0, GNT1, GNT2, GNT3, GNT4, GNT5, GNT6, GNT7);
attribute ENUM_ENCODING: string;  -- property declaration
attribute ENUM_ENCODING of STATE_TYPE: type is  -- property assignment
"0000 1000 1100 1101 1001 1010 1110 1111 1011";
begin
    // Define signal variables
    STATE, NEXT_STATE: STATE_TYPE;

    // Define process
    REG: process(CLK, RESET)
    begin
        if RESET = '1' then
            STATE <= IDLE after 10 ns;
        elsif CLK='1' and CLK'event then
            STATE <= NEXT_STATE after 10 ns;  -- present state update
        end if;
    end process REG;
COMB_LOGIC:  

process (STATE, R) 
begin 
  G <= (others=>'0') after 10 ns; -- default output signals 
  NEXT_STATE <= IDLE after 10 ns; 
  case STATE is 
    when IDLE => 
      if R(0) = '1' then NEXT_STATE <= GNT0 after 10 ns; 
      elsif R(1) = '1' then NEXT_STATE <= GNT1 after 10 ns; 
      elsif R(2) = '1' then NEXT_STATE <= GNT2 after 10 ns; 
      elsif R(3) = '1' then NEXT_STATE <= GNT3 after 10 ns; 
      elsif R(4) = '1' then NEXT_STATE <= GNT4 after 10 ns; 
      elsif R(5) = '1' then NEXT_STATE <= GNT5 after 10 ns; 
      elsif R(6) = '1' then NEXT_STATE <= GNT6 after 10 ns; 
      elsif R(7) = '1' then NEXT_STATE <= GNT7 after 10 ns; 
      end if; 
    when GNT0 => 
      G(0) <= '1' after 10 ns; 
      if R(0) = '1' then NEXT_STATE <= GNT0 after 10 ns; 
      end if; 
    .... 
    when others => NEXT_STATE <= IDLE after 10 ns; 
  end case; 
end process; 
end BEHAVIOUR;
Comparison of CPLD- and FPGA-Implementation Results (ISE 6.2)

<table>
<thead>
<tr>
<th>Target hardware</th>
<th>State encoding</th>
<th>Max. frequency / MHz</th>
<th>Used HW resources</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPLD-</td>
<td>Implicit sequential without ENUM_ENCODING</td>
<td>76.9 MHz</td>
<td>12 MC, 31 PT, 4 D-FF, 8 outputs</td>
<td>Transitions 25</td>
</tr>
<tr>
<td>XC95108-PC84-20</td>
<td>Explicit prioritised adjacency with ENUM_ENCODING, user</td>
<td>76.9 MHz</td>
<td>12 MC, 44 PT, 4 D-FF</td>
<td></td>
</tr>
<tr>
<td>Implicit Gray without ENUM_ENCODING</td>
<td>76.9 MHz</td>
<td>12 MC, 35 PT 4 D-FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGA Spartan 3</td>
<td>Implicit sequential without ENUM_ENCODING</td>
<td>172.47 MHz</td>
<td>28 LUT, 4 D-FF, 14 slices, 209 gc</td>
<td>Static hazards</td>
</tr>
<tr>
<td>XC3S400-4 PQ208CES</td>
<td>Explicit prioritised adjacency with ENUM_ENCODING</td>
<td>181.58 MHz</td>
<td>31 LUT, 4 D-FF, 16 slices, 230 gc</td>
<td>Static hazards</td>
</tr>
<tr>
<td>Implicit One-Hot without ENUM_ENCODING</td>
<td>117.45 MHz</td>
<td>17 LUT, 9 D-FF, 9 slices, 177 gc</td>
<td>No output hazards</td>
<td></td>
</tr>
<tr>
<td>Implicit Gray without ENUM_ENCODING</td>
<td>136.89 MHz</td>
<td>32 LUT, 4 D-FF, 16 slices, 227 gc</td>
<td>Static hazards</td>
<td></td>
</tr>
</tbody>
</table>
Polling Circuit with Explicit Gray Encoding: FPGA Implementation (XC3S400-4PQ208CES)

Static hazards in output signals are caused by multiple state bit transitions.
Polling Circuit with One-Hot Encoding: FPGA Implementation (XC3S400-4PQ208CES)