Low-Power Design

Task:
Experiment with various low-power design techniques (different state encoding algorithms and decomposition) in order to find the best implementation in terms of power consumption. Since the FSM for this task is taken from a verification benchmark set, ensure its correctness and make adjustments to the description if necessary. Use Stochastic FSM State Encoder applet to analyze the stochastic behavior of the FSM and to calculate defect of user-defined encoding. Verify that the decomposed network realization is functionally identical to the original FSM. Report the resultant dynamic power consumption and resource utilization. Use FPGA of Nexys-3 board as target device. Although no hardware verification of the design is required, be sure to provide the pin assignment as well.

Low-Power Design Approaches
The hardware implementation of the FSM generally consists of a register, where binary state codes are held, and combinational logic, which computes the next state and outputs. Both parts serve as power dissipation sources, whereas power is consumed during charging and discharging of load capacitances. The dynamic power dissipation in the combinational part of the circuit is very difficult to estimate. Therefore, reduction of switching activity in the state register is considered the primary optimization goal.

The two most common approaches, which address this issue, are FSM state assignment and FSM decomposition. Both methods require probabilistic (stochastic) analysis of the FSM. Given the FSM description and the input probability distribution, the stochastic behavior can be studied by simulating the FSM in the context of its environment. As the observation time increases, the probability that FSM is in each of its states converges to a set of time-independent constant values called steady state probabilities. These stochastic estimations can then be successfully applied for solving state assignment and decomposition problems targeting low-power logic synthesis.
State Assignment

In a high-level specification states of the FSM are represented with variables in symbolic form. As current digital circuits employ bistable storage elements, which can hold one of only two possible values, transformation of these abstract variables to physical implementation requires binary encoding. In other words, each symbolic variable should be replaced with a binary vector. The resultant circuit is dependent on the selected encoding, which may affect area, performance, testability and power consumption among others. In order to achieve reduction of switching activity in the state register, the Hamming distance (number of bits by which two codes differ) between adjacent states with higher transition probability should be minimized.

The optimization is measured in terms of a cost function, which considers Hamming distance between adjacent state codes, as well as the probability of making that particular transition. It can be assumed, that there is no difference between transitions from logic '1' to logic '0' or vice versa. Therefore, state transitions may be considered undirected. Note, that loop transitions do not alter the state code and should not be considered. The cost function can then be defined with (1), where $p_{ij}$ is the total transition probability of undirected transition for neighbor states $i$ and $j$, and $H_{ij}$ is the Hamming distance between codes of these states.

\[
C = \sum_{ij} p_{ij} \cdot H_{ij}
\]  

(1)

The quality of user-defined encoding can be measured with encoding defect. Encoding defect is a ratio of cost function values for the received encoding to the perfect encoding. In case of perfect encoding (although it is not necessarily attainable), the Hamming distance of codes for each pair of neighbor states should equal one.

Xilinx Integrated Software Environment (ISE) allows to employ different state assignment algorithms (along with user-defined encoding), which can be specified in the properties of the Synthesize process under HDL Options tab. The user-defined encoding can be applied
to the VHDL code of the FSM as follows:

```vhdl
package FSM is

type STATE_TYPE is (State_0, State_1, State_2, State_3, State_4);
attribute enum_encoding: string;
attribute enum_encoding of STATE_TYPE: type is "110 100 101 001 000";

Note, that the order of state codes should exactly match the order of symbolic variables in
the declaration of the enumerated type “STATE_TYPE”.

Computational Kernel Extraction
Sequential circuits may have an extremely large number of reachable states, but
probabilistic analysis show that during normal operation only a relatively small subset is
actually being visited (computational kernel). After computational kernel is identified (by
means of stochastic analysis), it can be separated from the rest of the circuit using the
notion of additive decomposition.

Decomposition problem is a task of substituting prototype FSM with a network of
interconnected sub-FSMs, which has the same terminal behavior. The main idea of additive
decomposition is the inclusion of a special “idle” state to each component sub-machine.
When not in use, the sub-FSM stays in that state. Thus, there is no redundant switching
activity present for non-working block (dynamic power management). Only one
sub-machine in the decomposed network can be active, while others component machines
remain suspended (stay in “idle” state). As the likelihood of computational kernel being
selected is very high, it can replace the original circuit for a large fraction of the operation
time, thus greatly reducing the switching activity of the circuit.

Table 1: Steady State Probability Distribution of the Example FSM

<table>
<thead>
<tr>
<th>State</th>
<th>Steady State Probability</th>
<th>State</th>
<th>Steady State Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>init0</td>
<td>0.5000001408</td>
<td>read0</td>
<td>0.0006720432</td>
</tr>
<tr>
<td>init1</td>
<td>0.3346775136</td>
<td>write0</td>
<td>0.0006720432</td>
</tr>
<tr>
<td>init2</td>
<td>0.0877016376</td>
<td>RMACK</td>
<td>0.0006720432</td>
</tr>
<tr>
<td>init4</td>
<td>0.0584677584</td>
<td>WMACK</td>
<td>0.0006720432</td>
</tr>
<tr>
<td>IOwait</td>
<td>0.0161290368</td>
<td>read1</td>
<td>0.0003360216</td>
</tr>
</tbody>
</table>
```
Consider an example FSM with steady state probability distribution as summarized in Table 1. It is easily seen that FSM spends 83% of its operation time in states “init0” and “init1”. Therefore, these states should form the computational kernel. The resultant computational kernel is a rather simple sub-FSM with three states (including “idle” state). Undoubtedly, it should consume considerably less power than original FSM does. Therefore, as for the most part computational kernel is going to remain an active component, a significant reduction in power consumption can be expected.

In order to ensure a proper transfer of control, components have to exchange information using so-called internal variables (Z-s). Whenever one sub-FSM is about to make a transition to “idle” state, it should pass information about its current state to other components in the network via additional output. This information is necessary to determine which sub-FSM gains the control and to which state it should move to.

Decomposed network for example FSM is presented on Figure 1. Component B0 represents
the computational kernel and works alternatively with component B1, which provides functionality not covered by B0. It can be seen that computational kernel can pass control only to one state of the component B1. Using output $Z_{B01}$, component B0 signals that it is about to enter “Idle” state and requests the transfer of control. Upon request, component B1 should make transition to the corresponding state. The same procedure of control transfer is valid for component B1. However, it can pass control to two states of the computational kernel. Therefore, component B1 requires two outputs ($Z_{B11}$ and $Z_{B12}$) to explicitly specify the state, which computational kernel should transit to.

Additional information on additive decomposition can be obtained at:
http://www.pld.ttu.ee/applets/decS/

Stochastic FSM State Encoder
Stochastic FSM Encoder is a set of tools, which perform FSM stochastic analysis. The software uses FSM description in KISS2 format as input data. Functionality, which is provided by the application, is divided into tabs with common information console. Each tab performs certain class of operations on the FSM.

Switch to “FSM Manager” tab. Place FSM description in KISS2 format into special text area and press “Load FSM” button to read the description. In case KISS2 description contains errors, application reports type and line number for each encountered error in information console.

Next, switch to “Probability Manager” tab. Set number of steps to 1000 and press “Perform Stochastic Analysis” button. The stochastic analysis provides steady state probability distribution for the loaded FSM description, which is then used to calculate the total transition probabilities. The resultant stochastic data can be viewed in information console. To print steady state probability distribution click “Print Steady State” button. To print total transition probabilities click “Print Total Transition” button. If “Print Non-directional Transitions” check box is set, then probabilities for nondirectional transitions are printed.
Next, switch to “Encoding Defect Calculation” tab. Provide user-defined encoding in the special text area. Each state code should be provided on a separate line. Every entry contains state label and the code itself separated with space. To print encoding defect click “Print Encoding Defect” button.

Stochastic FSM State Encoder applet can be found at:
http://www.pld.ttu.ee/applets/state/

**XPower Analyzer**

In order to estimate the impact of the proposed low-power design approaches on power consumption, XPower Analyzer tool can be used. Highlight top-level source file in Sources for window. Unfold Implementation process, then Place & Route process and double-click XPower Analyzer. The default settings for the switching rates should be applied when program starts (12.5%). Set the frequency of clock signal to 50MHz. Apply changes by clicking Update Power Analysis button. Only dynamic power component should be considered, as it is the target of minimization.

**KISS2 File Format**

File lines may be of two types: header lines and data lines. Header lines describe the common FSM properties and start with ASCII “.” character. Data line describes the input pattern that produces a transition from given current state to next state and output pattern.

*Table 2: KISS2 File Format Property Specifiers*

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Value type</th>
<th>Value description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Integer</td>
<td>number of input lines</td>
</tr>
<tr>
<td>o</td>
<td>Integer</td>
<td>number of output lines</td>
</tr>
<tr>
<td>s</td>
<td>Integer</td>
<td>number of states used</td>
</tr>
<tr>
<td>p</td>
<td>Integer</td>
<td>number of products</td>
</tr>
<tr>
<td>r</td>
<td>String</td>
<td>reset state (not mandatory)</td>
</tr>
</tbody>
</table>

In header line, the next character after the “.” is treated as property specifier. This character must always be followed by a whitespace character and a property value. The property specifiers and corresponding permitted values are summarized in Table 2.
Data lines consist of four entries delimited with a space character: input pattern, current state, next state, output pattern. Input and output patterns may contain only symbols "1", "0", and "-" (don't care). An example of FSM description is presented in Listing 1.

Listing 1: Example of FSM Description in KISS2 File Format

```
.i 3
.o 4
.p 11
.s 7
--- s0 s1 0000
--0 s1 s2 1110
-01 s1 s3 1110
 011 s1 s4 1110
 111 s1 s5 1110
 0-- s2 s1 0010
 1-- s2 s5 0010
--- s3 s4 0001
--- s4 s6 0101
--- s5 s4 1100
--- s6 s0 0111
```