

Synchronous FSM Design Methodology for Low Power Smart Sensors and RFID Devices

Matěj Machalec¹, Jakub Šťastný¹

¹ ASICentrum s.r.o., Novodvorská 994, Praha 4, 142 00
 Email: {matej.machalec,jakub.stastny}@asicentrum.cz

Abstract – This contribution focuses on integration of low power design approaches for synchronous Finite State Machines (FSMs) designs into a standard RTL digital design flow. Text summarizes current techniques and presents a methodology for design of FSMs with reduced power consumption targeted to low-power smart sensors and RFID devices. The advantage of the described approach is a seamless integration of an FSM encoding algorithm into a standard RTL ASIC design flow. Described methodology allows to reduce device power consumption and is compatible with all design tools. The methodology is also independent on the designed device and fully reusable between projects. The addition of low-power customized FSM encoding needs only one additional design step requiring only small amount of coding; the effort overhead is nearly zero. The contribution to the digital logic design is evaluated by integration of the methodology into our ASIC design flow during design of a real ASIC smart sensor platform; a reduction of power consumption of up to 70% in the FSMs is demonstrated while the power optimization process itself took less than one manday.

1 Introduction

We have been dealing with research in the field of low power logic design styles applicable to smart sensor and RFID devices digital logic design. As any design contains a few Finite State Machines (FSMs) of various complexity, we also focused our effort to synchronous FSMs designs and related low power design approaches.

The first step of our work was to evaluate existing approaches from the practical usability point of view. The state encoding, state register clock gating, and FSM decomposition approaches were evaluated. Encoding of FSM states is the most interesting one; a lot of papers on encoding algorithms exist, but all of them neglect the basic engineer's problem: how to integrate these algorithms into a real ASIC RTL (Register Transfer Level) level-based flow. Surprisingly, this integration is not straightforward as it must be simple, fast, and easy to learn. The FSM state encoding process shall not require any large change of already debugged RTL code to avoid introduction of new bugs into the design. In addition to this, standard RTL synthesis tools do not support automatic low power encoding of FSM states.

The article presents a simple methodology for integration of customized FSM encoding into the VHDL RTL level desing flow. The proposed methodology is independent on the designed devices, used tools, and fully reusable between projects. The customized FSM encoding is done in only one addi-

tional design step – generation and integration of the state package into the design; the effort overhead is nearly zero.

This contribution is organized as follows: the second chapter deals with theoretic background of synchronous low-power FSM design. The third chapter presents the selected approach and its integration into a design flow. The fourth chapter summarizes results achieved with the selected approach followed by some design rules in the fifth chapter. The last section of the article is devoted to conclusions.

2 CMOS Logic Consumption

The CMOS digital circuit total power consumption is the combination of the following components [1]:

Switching power is the power dissipated by toggling of wires in the design:

$$P_{sw} = p_{sw} \times C_l \times V \times V_{dd} \times f_{clk}, \quad (1)$$

where p_{sw} is the probability that a transition on a wire occurs, C_l is the wire output loading capacitance, $V=V_{dd}$ is the output voltage swing, V_{dd} is the supply voltage, and f_{clk} is the toggling frequency. Switching power can be reduced by reduction of C_l (selection of cells, fan-out reduction), V_{dd} and V reduction, lowering of f_{clk} , or by reduction of p_{sw} .

Short-circuit(crowbar) power is the power dissipated as a result of short circuit current arising when both NMOS and PMOS transistors in the CMOS gate are simultaneously open connecting the supply to the ground.

Finally, leakage power originates from substrate injection and sub-threshold effect.

Leakage power is negligible from smart sensors and RFID devices point of view because it is low in used CMOS technology libraries. Switching and short-circuit power are of most interest for us as switching activity reduction results not only in an overall power consumption reduction, but is also beneficial from the signal integrity point of view and reduces noise spreading over the device via power supply wires (important in mixed-mode devices as smart sensors).

3 Low-Power Design Approaches

While the power consumption optimization can be made on various design abstraction levels (system, architectural, algorithmic, circuit design/RTL, and technological), we are interested only in RTL level, here. Only switching probability p_{sw} and to lesser extent capacitance C_l can be effectively controlled on the RTL level. Regarding the FSM design we have to consider [2]

- **encoding of states:** internal FSM switching activity as well as output glitching can be reduced if we suitably encode the states of FSMs to reduce Hamming distances (number of differing bits) between any codes assigned to adjacent FSM states.
- **clock gating:** clock trees usually consume up to 50% of the dynamic power of the chip and 45-50% of the clock tree power comes from the internal switching power of the registers [3]. Reduction of clock switching by clock gating is thus desirable. Clock gating can be used to prevent FSM state register clock from toggling if the state does not change.

3.1 Design Examples

A simple FSM design will be used through the text to demonstrate all the concepts. The presented FSM is a simplified control FSM of the I2C controller used in a real world smart sensor ASIC device, see Fig. 1 for its transition diagram. Statistics of state transitions reported by the digital simulator using an extensive verification suite are in Table 1. There are two ways to implement an FSM – as Moore or Mealy one, see Figure 2 for the simplified diagrams of both, in this case the FSM is a Mealy one. Further, note the Test Observability Point (TOP) output of the FSM. Ad hoc test logic was used in the designed system, FSM state was made visible from the external world to increase device fault coverage and allow simple testing or debugging of the device.

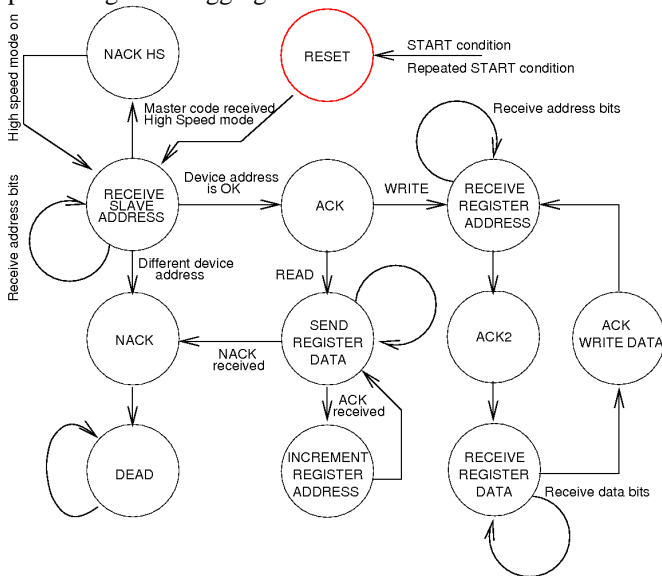


Figure 1: I2C line control FSM

An internal proprietary smart-sensor reusable platform was used for evaluation of methodology from digital design flow point of view, see Figure 3. The platform has the following properties:

- SPI and I2C interfaces to external world,
- integrated NVM memory controller for on-chip storage of calibration and configuration data,
- integrated proprietary RISC programmable CPU with hardware support for digital signal processing operations (three address generators, MAC unit),
- interface for on-chip AD converter,

- possibility to process up to four analogue channels, one of which can be temperature, support for temperature compensation and interfaces to on-chip DA converters trimming analogue blocks.

The device contains seven large FSMs which were optimized using the presented methodology, see Table 3. An example of the finite state machine implementation in VHDL using three processes is in Listing 1.

Table 1: FSM transition statistics.

From State	To State	Num of Transitions
RESET	REC_SL_ADDR	4948
REC_SL_ADDR	REC_SL_ADDR	29682
	ACK	4242
	NACK	56
	NACK_HS	648
ACK	REC_RE_ADDR	2470
	SEN_REG_DATA	1772
REC_RE_ADDR	REC_RE_ADDR	19446
	ACK2	2778
ACK2	REC_RE_DATA	2778
REC_RE_DATA	REC_RE_DATA	6973
	ACK_WR_DATA	995
ACK_WR_DATA	REC_RE_ADDR	995
NACK	DEAD	56
DEAD	DEAD	489
SEND_REG_DATA	NACK	1772
	SEN_REG_DATA	16402
	INC_RE_ADDR	318
INC_RE_ADDR	SEN_REG_DATA	318
NACK_HS	REC_SL_ADDR	648

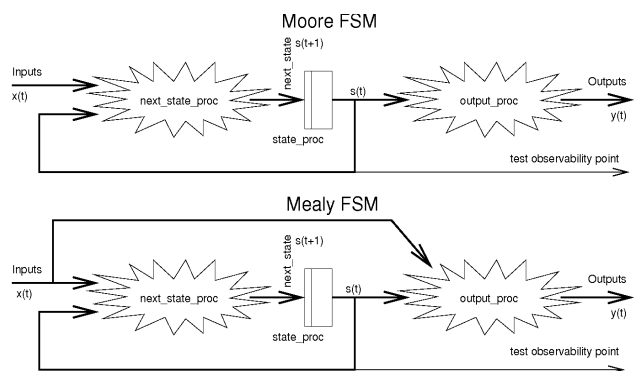


Figure 2: Mealy and Moore FSMs – basic structures. The names of the blocks correspond to the FSM VHDL code example below.

Listing 1 An example of VHDL Code Implementing finite state machine.

```

ENTITY i2c_fsm IS
  PORT (
    --reset and clock signals:
    i2c_res      : IN  std_logic;
    scl          : IN  std_logic;
    sda         : IN  std_logic;
  );
END ENTITY i2c_fsm;

```


ratic device behaviour or to device lock-up. Some of the codes allow detection of an invalid device state.

- finally, with real ASIC design it is sometimes necessary to do an Engineering Change Order (ECO, [5]) correction if a bug is found in the device; then there is a need to change the FSM logic by hand. The easiness or even sheer possibility of such a change depends strongly on used FSM encoding.

The descriptions of the most popular codes follow.

3.2.1 Binary code

Binary code is well known by all designers. Advantages of this code are quick coding and minimal number of flip-flops ($\log_2(N)$, where N is the number of states) needed to capture the FSM state.

Disadvantage is an excessive HD_{ave} between codewords. Further, next state and output signals depend on all bits of the internal FSM state registers, thus combinational logic blocks are large [6] and complex which also limits possible ECO change. Interpretation of STA results is complicated due to possibility of false paths in the next state logic design [6]. Detection of invalid state is impossible without usage of error detecting codes.

3.2.2 Gray Code

Advantage of Gray code is that codes assigned to two adjacent states have minimal possible Hamming distance equal to 1. Thanks to this, glitching at the combinational FSM outputs is reduced. The number of state register flip-flops is also minimal as in case of binary code.

The disadvantage is that both the next state logic as well as output logic depend on all values of all state registers. Possibility of ECO change of such an FSM is again limited. SEU detection can be partially based on the knowledge of maximal Hamming distance between two adjacent codewords; anyway, an error detection code is still necessary.

Gray code sequence is usually a sequence of $N=2^k$ length. However, a Gray code can be generated always if the number of codewords is an even number, see [7].

For complex FSM with a lot of branches and loops in the transition diagram is Gray encoding sometimes not possible. Then a suboptimal encoding can be used attempting to encode only states in a frequently visited loop with Gray code to minimize HD_{ave} , sacrificing the Hamming distance for less probable transitions.

3.2.3 One-hot/Zero-hot Code

Each state is encoded as a binary string with all zeros but only one logical one and length of code word is equal to number of FSM states. Due to this, Hamming distance between any different two states is equal to 2.

The one-hot code has many advantages; first, the coding allows to simply detect an SEU. The combinational logic for next state and output is usually smaller than in case of binary code allowing an FSM to operate at higher system clock frequency. Glitching at the FSM output signals should be reduced compared to binary coding. Interpretation of STA results is

simple as there are no false paths in the FSM. Last, but not least, ECO change of a one-hot encoded FSM is very simple and it is possible to add new states without a need for recoding the whole FSM. It is also possible to simply derive the FSM electrical scheme from the transition diagram [8].

The clear disadvantage is the number of flip-flops needed to realize this coding; we need N registers. This can lead to increased switching power consumption in the registers as well as in the clock tree. However, this problem is not critical with FPGA devices which are register-rich architectures.

Sometimes it can be advantageous to encode initial states with all zeros which can save one flip-flop and simplify the reset logic. Similarly we can get zero-hot code where each state is encoded as all ones but only one logical zero.

Designer shall take care of the real implementation of the one-hot encoded FSM after synthesis. The next state shall always depend only on one distinct register set to 1, not on the other ones at 0 [6].

3.2.4 Johnson code

Johnson code is built as follows: the first state is encoded with all zeros; for the next states the state vector is shifted left with a MSB set to logical one afterwards until all registers in the state register are set to ones. The process is repeated for the rest of states with a zero given from the right during left shift.

Disadvantage is that number of flip-flops in the state register is equal to $N/2$. Advantage of this code is that adjacent states have the Hamming distance of their codes equal to one. This code is suitable for clock generators since glitching at combinational logic outputs is suppressed. An FSM state encoding by Johnson code may not exist for larger FSMs with many transitions between different states.

3.2.5 Random Coding

This method uses a simple random generation of codewords assigned to FSM states. The random generation is usually repeated and for each run the cost function (1) is calculated. The encoding with minimal cost is taken as result giving a solution which is almost optimal one (for smaller FSMs) and this method is very quick.

3.2.6 Customized FSM Encoding

The selection of the state assignment depends on several parameters such as the number of states, the number of paths and their lengths, the number of forks in the FSM transition diagram and the complexity of the predicates on transitions between states. It is recommended [9] to use "1 of N" coding for smaller FSMs (up to approx. 10 states) due to extremely simple decoding logic. A good general practice is also to group states generating the same outputs and assign them codes with minimum mutual Hamming distances. For bigger FSMs, Gray encoding can do a good job in power consumption reduction, however output logic activity should be also taken in the account.

Often any of presented codes does not reach a minimal possible HD_{ave} and there is a need to use a customized encoding. A lot of algorithms for customized encoding already exist

based on different principles (simulated annealing, genetic algorithms, simple greedy approach, etc). A cost function similar to (1) is usually used. Encoding algorithm finds the minimum of the cost function – a (sub)optimal solution minimizing switching activity by searching through the state space of possible state encodings.

Example of greedy search algorithm could be a POW3 encoding algorithm [10]. Simple and easy to use algorithm can be found in [11]. Slightly more complicated algorithm called "WEAKLY CROSSED EDGE CUTS" (WCEC) is presented in [12]. From experimental measured results can be seen that for most of the benchmarks, WCEC algorithm produces circuits with less switching activity than POW3. Paper [13] presents a Gray code encoding algorithm. The basic principle is to assign states to Gray code tree (GCT) nodes. As our work was primarily targeted to the design of a simple and easy to use methodology, we needed an efficient algorithm which is easy to implement so as we may quickly evaluate the feasibility of the whole RTL design flow extension. Owing to this we decided to use algorithm described in [14], we call it ALG1 in the text below, its description follows.

The FSM is first to be represented by Probabilistic State Transition Graph (PSTG). PSTG is a directed graph consisting of a set of nodes and a set of edges. Each node represents a state of the FSM. Directed edge S_i-S_j represents a transition from state S_i to state S_j . Each edge is also associated with a number, which denotes a probability of transitions from state S_i to state S_j . The encoding algorithm is then as follows:

1. Select a node n with maximum sum of all N_n incident edge weights. N_n is set of adjacent nodes (to states) from state n .
2. Sort adjacent nodes of node n by incident edge weights in decreasing order.
3. Assign some code to n and minimally distant codes to the N_n adjacent nodes. Allows all N_b nodes to be at distance 1 from n . N_b is the sub-set of first b nodes from sorted set N_n (b is the number of code length). If $|N_n| > |N_b|$ then assign any free codes for rest of nodes.
4. Remove node n (and all incident edges) from graph.
5. Go to Step 1 with smaller graph if there is any node which has not been assigned a code to.

3.2.7 Examples of Encodings

Table 2 contains examples of described encoding. States of the demonstrational FSM are encoded using all the presented algorithms and HD_{ave} is computed for all of the codes. The best results (lowest HD_{ave}) are reached by the Gray and Alg1 encodings.

3.3 FSM State Register Clock Gating

A FSM usually stays in some of states for a longer time (see eg REC_RE_ADDR state in Figure 1 and Table 1) with outputs stable. Under such circumstances the state register clock signal can be gated to prevent it from toggling to reduce switching activity in the circuit.

A typical FSM with state clock gating support is in Figure 5. Block f_a is an activation function generating gating signal for clock; f_a shall be simple enough not to further increase

device power consumption. Gating signal is then latched to prevent propagation of glitches to the AND gate and gated clock line.

Experimental results show that the size of the logic slightly increases but for most of benchmarks FSM power savings vary between 10% and 30% [15]. In some cases the power reduction can be close to 100% [15]. In large FSM benchmarks the power reduction is about 10% [15].

Table 2: I2C Line Control FSM – examples of state encodings

State name	Bin	Gray	One-hot	John-son	Alg1
RESET	0000	0000	0000000000	000000	0000
REC_SL_ADDR	0001	0001	1000000000	000001	1000
ACK	0010	0011	0100000000	000011	1100
REC_RE_ADDR	0011	0010	0010000000	000111	0100
ACK2	0100	0110	0001000000	001111	0110
REC_RE_DATA	0101	0111	0000100000	011111	0010
ACK_WR_DATA	0110	1111	0000010000	111111	0011
NACK	0111	1110	0000001000	111110	1001
DEAD	1000	1010	0000000100	111100	0001
SEN_REG_DATA	1001	1011	0000000010	111000	1110
INC_RE_ADDR	1010	1001	0000000001	110000	1111
NACK_HS	1011	1000	0000000000	100000	1010
HD_{ave}	0,47	0,31	0,46	0,38	0,31

3.4 Partition (decomposition) of FSM

Controllers are often critical from power consumption point of view because they are continuously running while some parts of device datapath can be turn off. Delay through the FSM can constrain delay through the data path if a Mealy FSM is used, too. Due to this it is advantageous to decompose FSM into more mutually interacting sub-FSMs having the same input-output behaviour as the original FSM. Only one sub-FSM is active controlling all FSMs outputs while other sub-FSMs are idle. When active sub-FSM ends with its execution, it sends activation signal to another idle sub-FSM and deactivates itself. Clock lines as well as power lines driving idle sub-FSMs can be gated off to reduce dissipated power.

Decomposition algorithms (see eg [16, 17, 10]) attempt to find the optimal solution for partition of FSM. At the beginning there is some random solution and then algorithm tries to find better partition. Again, a cost function is used giving information about the current solution (partition) from switching activity point of view.

Based on our experience, purely low-power driven FSM decomposition is not practical during RTL level design as it complicates debugging. The interplay of decomposed FSMs adds a new degree of freedom to the RTL design, complicates verification, and increases probability of some fatal system design bug (eg FSM deadlock due to bad decomposition). Finally, none of the tools we use supports an automatic FSM

decomposition, so that hand-made solution would be inevitable. Due to all these disadvantages we will not deal with FSM decomposition any further in this paper; functionally driven FSM decomposition early in the system level design is a better solution.

4 Methods

The proposed methodology for an automatic FSM encoding supporting also FSM state register clock gating was designed with respect to the the following criteria:

- **simplicity and speed of the application:** the application shall be simple enough not to force an ASIC designer to learn a completely new tool. Further, it shall not require much effort not to negatively impact *time to market* of the design.
- **compatibility with used tools:** the adopted method shall be compatible with our tools (DC compiler for synthesis, NCSim for digital simulation), shall not require any new tools, and shall not require a significant change in the RTL digital logic design flow.
- **reduction of device power consumption:** there shall be a significant positive impact of the chosen method on the device power consumption.

4.1 State Encoding

Customized state encoding can be used even despite of the fact that standard RTL synthesis tools usually do not support an automatic optimization of state encoding for low power designs. To use customized encodings, we first need to get statistics on FSM transitions which can be done in the following ways:

- **using additional VHDL code:** a dedicated process tracking the number of state to state transitions can be added to the FSM RTL code. This solution is compatible with any digital simulator but the additional effort needed to be spent on coding of the monitoring block would not be negligible.
- **PSL coverage points:** a dedicated PSL language [18] constructs can be added to the FSM design. The code would again monitor state transitions and then report the statistics using coverage points. This solution is compatible with any digital simulator supporting PSL constructs, however, there is again a need for an additional effort spent on PSL assertions writing.
- **FSM coverage automated tool support:** all modern digital simulators have the code coverage option. The code coverage is a verification feature; the simulator keeps track of executed RTL code lines and reports which lines of RTL code were not executed during device verification and thus are prone to errors. The same works for FSM state coverage; the simulator keeps track of which transitions were executed and how many times and is able to generate a report on this. This report can be further used as a source of statistics to compute state transition probabilities. Advantage of this approach is a minimal effort needed to obtain the transition probabilities. The dis-

advantage is that the digital simulator must support this feature; however, all modern simulators implement it. Then the format of the FSM coverage report is tool-dependent requiring a unique parser for any digital simulator used. An example of such a FSM coverage report follows:

All FSM Detail Report, Instance-Based

```

=====
Instance name:
design.top_dig(struct):i_i2c_top:i_i2c_fsm
Module/Entity name: design.i2c_fsm(rtl)
State Register: curr_state
Number of covered states: 12 of 12
Number of uncovered states: 0 of 12
Number of covered transitions: 20 of 20
Number of uncovered transitions: 0 of 20
Number of covered arcs: 20 of 20
Number of uncovered arcs: 0 of 20

```

State Coverage :

State	Encoding	Num of Visits
RESET	0000	5084
REC_SL_ADDR	0001	35278
--other states		
NACK_HS	1011	648

Reset State Num of Resets

Reset State	Num of Resets
RESET	5084

```

Inputs = ( (I2C_DET_BIT_7='1'), (I2C_DEV_ADDR_EQ='1'),
--other inputs

```

Arc Coverage :

P-State	N-State	Inputs	Num of Visits
RESET	REC_SL_ADDR	-----	4948
REC_SL_ADDR	REC_SL_ADDR	0-----	29682
	ACK	11----	4242
--other states			
NACK_HS	REC_SL_ADDR	-----	648

The second question is when to integrate the customized FSM encoding to the RTL design. We can do it

1. **in the RTL coding phase:** the enumerated definition of states would be simply replaced by constants of *std_logic_vector* type. An advantage of this approach is a simpler implementation of TOP, see Figure 2, and improved RTL-level Power Compiler device power estimation (as enumerated type is replaced by real state codes). A disadvantage is the slight loss of abstraction during device debugging; instead of abstract enumerated type values, the real state codes will be shown in the simulator „waves“ window.
2. **during synthesis:** The synthesis process can be controlled with appropriate commands and forced to encode the FSM states in a predefined way. This approach has an advantage in higher level of abstraction and no need of RTL code modification; however, the TOP is harder to implement and used commands are dependent on the synthesis tool.

We decided for the RTL level application. First, the whole digital device is designed and debugged in a standard way with FSMs using enumerated data type. Then, in the final stage of the design the FSM encoding flow is applied; the following steps are to be done:

1. digital simulation is run with FSM coverage option, the coverage report is generated. A synthesis is rec-

- recommended to be run to get a first design size estimation.
- generated report is analyzed by a proprietary PERL script. The script reads the FSM transitions statistics and generates a VHDL package with constants with the same name as members of the enumerated type, but of *std_logic_vector* type. These constants encode FSM states, see below an example of such a package. Along with the package, HD_{ave} is reported for the selected encodings. The script is to be run for all the supported encodings and the one reaching minimal HD_{ave} is to be chosen for the final implementation. This is actually a greedy heuristic approach to the FSM encoding selection.
- original FSM enumerated type state definition is replaced by the generated package which introduces new encoding of the FSM and a new synthesis is done. The sizes prior and after encoding applications are to be compared to check if the new encoding has not hampered design size.

Note that state encoding is applicable for FPGA as well as ASIC designs; with both platforms power savings can be obtained. See Figure 4 for the flow flowchart and an example of an encoding package listed below.

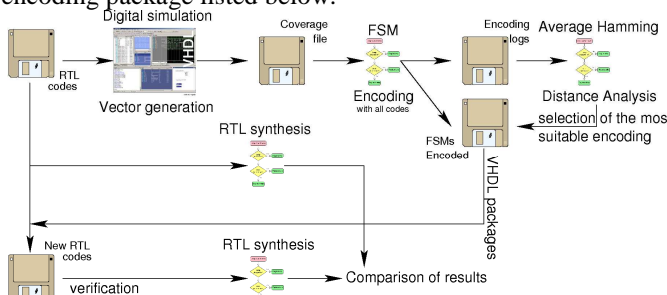


Figure 4 The FSM encoding flow.

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

PACKAGE i2c_fsm_pkg IS

CONSTANT i2c_fsm_code_width : natural := 4;
SUBTYPE t_i2c_fsm_state IS std_logic_vector(i2c_fsm_code_width-1
DOWNT0 0);

CONSTANT RESET
-- other states
....
CONSTANT NACK_HS
: t_i2c_fsm_state := "1010";

END i2c_fsm_pkg;

PACKAGE BODY i2c_fsm_pkg IS
END i2c_fsm_pkg;
    
```

4.2 State Register Clock Gating

State clock gating can be easily implemented on RTL level. However, the complicated part is to determine the suitable states for clock gating and an appropriate activation function F_a . Results which are obtained from simulation and FSM coverage tool can be also used to determine states which are suitable for state clock gating.

The following steps are to be done to introduce clock gating to the FSM state registers:

- digital simulation is run with FSM coverage option, the coverage report is generated.
- designer analyses the report and finds states in which the FSM spends significant time by waiting.
- clock gating is implemented on RTL level based on this information.

Implementation of clock gating can be further facilitated using DC compiler/Power Compiler automatic clock gating feature. DC compiler is able to convert clock enables to clock gating which would allow to simply transfer the ASIC RTL design to FPGA device prototyping stage. The ASIC codes will be synthesized with clock gating then, while FPGA prototype will use clock enables without any additional design effort.

Clock gating is applicable only with ASIC designs. This method is not applicable with FPGA designs as there is no possibility to balance clock trees.

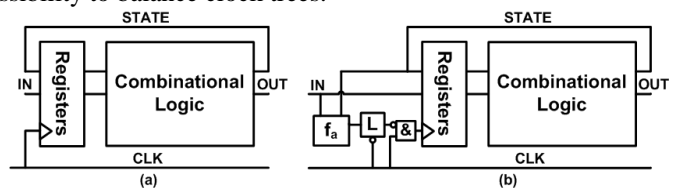


Figure 5:a) Single clock flip-flop based FSM. b) Gated clock version; drawing based on [19].

5 Results

Proposed methodology was used to encode all the FSMs in the demonstrational ASIC smart sensor digital design platform.

5.1 Power Consumption

FSM state clock gating is introduced to the following FSMs: *ee_w_flag_ctrl*, *ee_slave_fsm*, *ee_ctrl_fsm* and *fsm_control* (FSMs are marked with '+'). The power consumption estimation of the original design without and with state clock gating (columns *orig* and *orig+* respectively) are presented in tables below. The last column (%) contains effectiveness of the best coding marked in green compared to *orig* or *orig+* in percents.

All power estimation results in Tables 3 and 4 are in mW; Synopsys Power Compiler was used to estimate the device power consumption. The best result is marked in green for all of the presented FSMs. The results show that

- a significant power saving can be achieved (up to 70%) depending on the FSM complexity which is in compliance with statements in [14] and others.
- HD_{ave} based encoding selection leads to an optimal solution in 4 out of 7 cases (compare red and green field in Table 5 with corresponding power consumptions in Table 4). The heuristic selection is wrong for the following FSMs, see also red fields in Table 5:
 - mcu_ctrl_fsm*: the chosen encoding still lowers the FSM power consumption by 9% compared to the original state. In addition to this, the amount of glitching at the FSM output is significantly reduced by both Gray and ALG1 encodings and leads to a significant further power savings in the device (the FSM is an instruction decoder of the

RISC CPU with large capacitive loads on its outputs). So the ALG1 selection is not optimal, but still leads to power savings.

- *lock_ee_img_fsm*: a small FSM with just four states toggling very rarely in the design. It does not make sense to apply the customized encoding on such an FSM at all.
- *i2c_fsm*: no power savings would be obtained if the Gray code is chosen instead of ALG1; here the heuristic approach fails.

It is necessary to emphasise here, that the simulation used for FSM coverage generation shall really activate the FSM in a way as close to the real application as possible not to get skewed statistics.

Table 3: RTL power estimations

FSM	Bin	Gray	ALG1	orig+	%
				orig	%
mcu_ctrl_fsm	3.77E-03	3.66E-03	3.47E-03	NA	
				5.42E-03	64
lock_ee_img_fsm	3.70E-06	1.29E-05	7.65E-06	NA	
				3.61E-06	102
ee_w_flag_ctrl (+)	2.47E-05	2.42E-05	2.40E-05	2.42E-05	99
				6.82E-05	35
ee_slave_fsm (+)	1.08E-05	1.06E-05	1.04E-05	1.10E-05	95
				2.03E-05	51
ee_ctrl_fsm (+)	1.23E-04	9.27E-05	1.17E-04	1.70E-04	55
				3.04E-04	30
fsm_control (+)	1.18E-05	1.16E-05	1.09E-05	1.22E-05	89
				2.24E-05	49
i2c_fsm	3.92E-05	4.16E-05	3.36E-05	NA	
				7.98E-05	42

Table 4: Post-place&route power estimations

FSM	Bin	Gray	ALG1	orig+	%
				orig	%
mcu_ctrl_fsm	5.73E-03	4.94E-03	5.55E-03	NA	
				6.08E-03	81
lock_ee_img_fsm	3.52E-06	4.69E-06	5.55E-06	NA	
				3.49E-06	101
ee_w_flag_ctrl (+)	2.56E-05	2.55E-05	2.42E-05	2.59E-05	93
				2.78E-05	87
ee_slave_fsm (+)	1.31E-05	1.24E-05	1.31E-05	1.31E-05	95
				1.62E-05	77
ee_ctrl_fsm (+)	1.11E-04	9.87E-05	1.44E-04	1.39E-04	71
				1.54E-04	64
fsm_control (+)	1.25E-05	1.23E-05	1.17E-05	1.31E-05	89
				1.61E-05	73
i2c_fsm	5.33E-05	4.62E-05	3.51E-05	NA	
				4.53E-05	77

Table 5: HD_{ave} for encoded FSMs. Green indicates that the HD_{ave} prediction is in compliance with real power consumption of the FSM estimated using post-place&routed netlist, red indicates that the HD_{ave} mispredicted the power consumption.

FSM	HD_{ave}		
	Bin	Gray	ALG1
mcu_ctrl_fsm	0.1934	0.3293	0.1584
lock_ee_img_fsm	0.0488	0.0316	0.0316
ee_w_flag_ctrl	0.0010	0.0010	0.0007
ee_slave_fsm	0.0660	0.0356	0.0356
ee_ctrl_fsm	0.3763	0.1940	0.1981
fsm_control	0.0178	0.0116	0.0116
i2c_fsm	0.4686	0.3070	0.3102

5.2 Real Design Effort

The real design effort needed to run FSM coverage, to compute HD_{ave} for all the encodings, and to encode all the FSMs was approximately half a day with the demonstrational ASIC design. The whole procedure was smooth and proven as useful. The methodology was also tested by another designer on a different project with similar results.

6 Conclusions, Next Steps

The methodology allows integration of all the theoretical results and algorithms presented in scientific papers into a standard RTL ASIC design flow. Our contribution lies in development of an extension to the standard flow using FSM coverage to gather information on state transitions, a dedicated tool for FSM encoding, and the VHDL RTL coding methodology for fast FSM encoding without any additional effort. The ease of application was tested in the frame of one pilot project and it was shown that the whole process is fast and takes less than half a day of work. Additional testing with another design was also done. The only proprietary part of the whole flow is the encoding and FSM coverage log parsing script which can be easily implemented eg in Perl.

Further, the chosen approach simplifies TOP for FSM state design and gives enough information to find out in which states it is advantageous to implement FSM state register clock gating.

As the heart of the process is a greedy heuristic, it is recommended to avoid encoding of small/simple FSMs; in case of doubt, a verification with tool estimating device power consumption can be done (eg Power Compiler).

We already plan the following extensions of the implemented tool:

1. integration of an automatic decision process based on HD_{ave} ; the tool will do all the analysis by itself and generate only the optimal encoding,
2. integration of an Error Correction Code (ECC) generator to allow a simple design of an FSM with state register protected against SEUs.
3. extension of supported encoding algorithms with some other heuristic approaches.

Acknowledgment

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100029.

References

- [1] HOROWITZ Mark, INDERMAUR Thomas, and GONZALEZ Ricardo. Low-power digital design, IEEE Symposium on Low Power Electronics, Digest of Technical Papers, 1994, pp 8-11.
- [2] ARSLAN T., ERDOGAN A. T., HORROCKS D. H. Low power design for DSP: methodologies and techniques, Microelectronic Journal 27, 1996, pp. 713-744.
- [3] SHABEL Jeff. Analyzing Clock Trees, SNUG Boston 2005.
- [4] Single event upset
http://en.wikipedia.org/wiki/Single_event_upset
- [5] Engineering Change Order
http://en.wikipedia.org/wiki/Engineering_Change_Order
- [7] MAXFIELD Clive. Yet another Gray code conundrum [online], July 19, 2007 [cit. 2009-11-19]
<http://www.pldesignline.com/201002340;jsessionid=SEEHOQLT04WKZQE1GHPSKHWATMY32JVN?printableArticle=true>
- [8] GOLSON Steve. One-hot state machine design for FPGAs, 3rd PLD Design conference, Santa Clara, 1993.
- [6] GOLSON Steve. State machine design techniques for Verilog and VHDL, 1994
- [9] PIGUET Christian. Low-Power Electronics Design, CRC Press.
- [10] BENINI Luca, DE MICHELI Giovanni. State Assignment for Low Power Dissipation, IEEE Journal of Solid-states Circuits, vol. 30, no. 8, March 1995, pp 258-268
- [11] PANDA P. R. Low Power FSM Encoding. I.I.T Delhi, 2008.
- [12] FOMINA Elena. State encoding techniques for low power FSM, International conference on computer systems and technologies, CompSysTech 2005.
- [13] WANG Chua-Chin, DER JENG Ming. Power Estimation of Internal Nodes for Finite State Machine Using Gray Code Encoding in State Assignment, Department of Electrical Engineering, National Sun Yat-Sen University Kaohsiung, Taiwan 80424
- [14] ROY, Kaushik, PRASAD, Sharat C. Circuit activity based logic synthesis for low power reliable operations. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 1, no 4, December 1993, pp. 503 – 513.
- [15] BENINI Luca, DE MICHELI Giovanni. Optimal synthesis of gated clocks for low-power Finite-State Machines, IEEE transactions on computer-aided design of integrated circuits and systems ISSN 0278-0070 CODEN ITCSDI 1996, vol.15, n6, pp.630-643.
- [16] BENINI Luca, DE MICHELI Giovanni, VERMEULEN Frederik. Finite State Machine Partitioning for Low Power Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium, 31 May-3 Jun 1998, Volume: 2, 5-8 vol.2, ISBN: 0-7803-4455-3
- [17] LEE MingHung, HWANG TingTing, HUANG Shi-Yu. Decomposition of Extended Finite State Machine for Low Power Design, Proceedings of the conference on Design, Automation and Test in Europe - Volume 1 - 11152, 2003. ISBN:1530-1591 , 0-7695-1870-2
- [18] The Designer's Guide To PSL
<http://www.doulos.com/knowhow/psl/>
- [19] FAVALLI M., BENINI Luca, DE MICHELI Giovanni. Design for testability of gated-clock FSMs, Proceedings of the 1996 European conference on Design and Test, 1996, pp. 58.