FINITE STATE MACHINE OPTIMISATION THROUGH RESOURCE SWITCHING

Yassen Gorbounov

University of Mining and Geology “St. Ivan Rilski”, 1700 Sofia; E-mail: y.gorbounov@mgu.bg

ABSTRACT. The article presents an approach for optimising the use of finite state machines in terms of the hardware resource used in physical implementation. This is done by using a shared context, which allows the repeated use of one and the same computational structure to perform different tasks. In this way, complex control algorithms can be implemented using a reduced number of components. The proposed approach makes it possible to achieve a high degree of integration and preserve the integrity of the signal by achieving shorter paths for its propagation. Ultimately, this leads to a reduction in energy consumption and a lower cost.

Key words: finite state machines, context switching, programmable logic.

ОПТИМИЗАЦИЯ НА КРАЙНИ АВТОМАТИ ЧРЕЗ ПРЕВКЛЮЧВАНЕ НА РЕСУРСИ

Ясен Горбунов

Минно-геоложки университет „Св. Иван Рилски“, 1700 София

РЕЗЮМЕ. Статията представя подход за оптимизация на използването на крайни автомати по отношение на заемания при физическа реализация хардуерен ресурс. Това е направено чрез използване на споделен контекст, което позволява многократната употреба на един и същ изчислителен апарат за изпълнение на различни задачи. По този начин могат да бъдат реализирани сложни управляващи алгоритми, използващи редуциран брой компоненти. Предложеният подход дава възможност за постигане на висока степен на интеграция и запазване на интегритета на сигнала чрез постигане на по-къси пътища за неговото разпространение. В крайна сметка това води до намаляване на консумацията на енергия и понижена цена.

Ключови думи: крайни автомати, контекстуално превключване, програмируема логика.

Introduction

With the burst development of computing technology, the demands on digital circuits increase in terms of their speed, the reduction of energy consumption, and the increase in the degree of integration. For this reason, programmable logic devices, which can be distinguished by their high operating frequencies, the possibility of multiple reprogramming, and their ability to execute parallel algorithms (Pavlitov 2007, Harris 2013), are increasingly used nowadays. This feature is inherent in their architecture and is their undeniable advantage over conventional microprocessors and microcontrollers.

The contemporary user-programmable logic devices can be categorised generally into two major groups – FLASH-based Complex Programmable Logic Devices (CPLD) and RAM-based Field Programmable Gate Arrays (FPGA) with the later being most promising. These integrated circuits abound in logic gates which connections can be configured by the user and allow for the flexible synthesis of complex circuits. In the devices of the first type, the main building block is the functional block that comprises a Programmable Logic Array (PLA) or Programmable Array Logic (PAL) matrix and a bunch of Macrocells (MC) whose core is the D-type flip-flop. The devices of the second type contain Look-up Tables (LUT), an adder, a D-type flip-flop and several multiplexers, all combined in a Configurable Logic Block (CLB) which is repeated in a field programmable structure. This organization is depicted in Fig. 1.

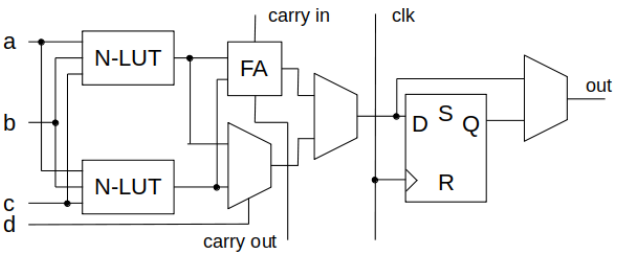

Fig. 1. Typical configurable Logic Block (CLB) cell in a FPGA

Besides this FPGAs integrate several single or dual-port Block RAMs of few kilobytes each. The above shows that the number of registers in an FPGA is limited, while at the same time memory cells prevail.

Finite automata are often used to solve various tasks and build complex algorithms to control the execution of processes in automation and computer technology in general. Their principle of operation will be discussed later. Here it is important to mention only that the central place in their structure is occupied by the state register, which is essentially a memory. To solve a specific task, structures composed of several finite state machines (FSM) can be used. Very often

they can be of the same type but perform different and sequential or time-multiplexed tasks. In essence, this principle is similar to the multitasking mechanism introduced long ago in computer architectures (Hennessy 2012, Muhammad 2020). The aim of the present article is to propose a unified approach for the repeated use of a computational structure (finite automaton) on which the state register is replaced at different moments of time, i.e. the contents of the memory. This allows reducing the number of flip-flops at the expense of flexible use of the built-in Block RAM memory without affecting the performance of the algorithm.

FSM architectural overview

Finite automata represent an abstract mathematical behavioral model of a machine with finite number of states and limited memory. There exist three major types of finite state automata, namely Medvedev, Moore, and Mealy state machine. Their organisation is summarised in Fig. 2.

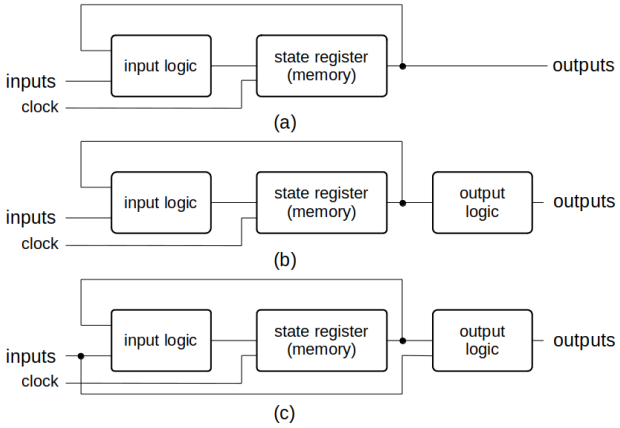

Fig. 2. The three types of finite state automata: (a) Medvedev, (b) Moore, and (c) Mealy

It can be seen that the Medvedev FSM (a) is included in all other types. The combinational logic that forms the inputs of the state register is one and the same for each of the three FSM types. Only the logic that forms the outputs differ.

Mathematically the FSM can be defined by its input alphabet *X*, output alphabet *Z*, and internal states *A* (1).

$$\begin{aligned} X&=\{x_0,x_1,x_2,\ldots,x_n\}\\ Z&=\{z_0,z_1,z_2,\ldots,z_m\}\\ A&=\{a_0,a_1,a_2,\ldots,a_w\} \end{aligned} \quad (1)$$

One of the states is set as the initial state and it is entered on reset. The working algorithm of the FSM is defined by its transitions function (2).

$$A_{new}=\lambda(A_{old},X) \quad (2)$$

Finally the differing function of the outputs is given by (3) for the Moore state machine and by (4) for the Mealy state machine.

$$Z=\delta_1(A) \quad (3)$$

$$Z=\delta_2(A,X) \quad (4)$$

Of practical interest is the design of deterministic finite state automata (DFA) where each of their transitions is determined by the state and a letter of the input alphabet; the reading of an input character is mandatory for each transition; each state has at most one transition at a given input.

From the given description, which is quite trivial, two deductions can be made. First, in some cases it is possible to keep the organisational structure of the FSM but only change the contents of the memory. Second, it is possible to mix the FSM styles (Moore and Mealy) by using two independent output functions.

Example of the problem

A simplified traffic light controller model can serve as a basic example of the problem of using several finite state machines that share one and the same functionality. A drawing of this model is shown in Fig. 3.

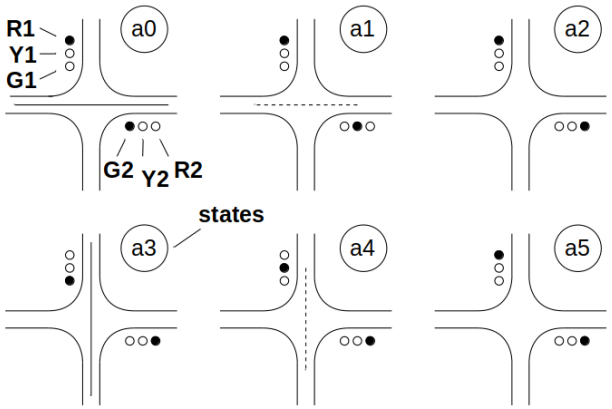

Fig. 3. Simplified traffic light controller model

This model represents two traffic lights that control the traffic in North-South (NS) and East-West (EW) directions on a crossroad. It has 6 states but for safety reasons, two of them are repeated. In state a0 the movement in the NS direction is forbidden while the EW direction allows passage. In state a0 the NS direction is still forbidden while the yellow light is lit for the EW direction allowing some time for those who entered the crossroad to leave it. In state a2 both NS and EW directions are blocked. In state a3 the passage is allowed in the NS direction and forbidden in the EW direction. In a4 the yellow light is lit for the NS direction allowing those who entered the crossroad in this direction to leave it. In the a5 state, both NS and EW directions are blocked. Next, the sequence repeats.

The directed graph of the described algorithm is shown in Fig. 4. The main loop is represented with the nodes a0 to a5. To make the crossroad even safer a display that shows the remaining time is added and it can be different for the different traffic light colors. In the graph the timer is represented with the hexagonal nodes that contain the number of the remaining seconds. This algorithm can be implemented using three counters. The first one is responsible for the main loop (the states in Fig. 3) and it is counting up. The second one is responsible for the time of 59 seconds. This larger time is

needed for the red/green lights as this is the longest wait time. In this mode the counter is counting down.

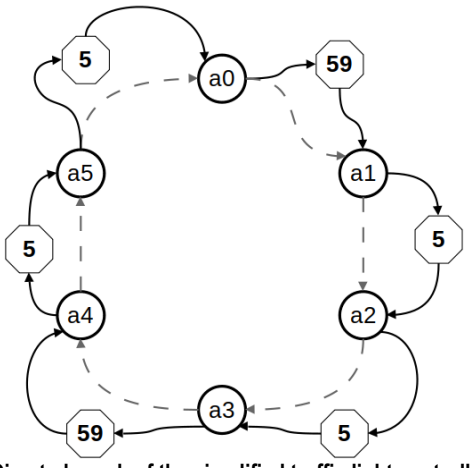

Fig. 4. Directed graph of the simplified traffic light controller model

The third counter is also counting in the down direction but it is initialized with a smaller time of 5 seconds.

Although normally the traffic light systems are not designed exactly this way, the example clearly demonstrates how similar finite state machine models can operate together. Since in this case, they work consecutively it becomes obvious that a single FSM is sufficient for the task.

Context switching approach

The context switching technique is commonly used in the computer architecture domain in which the context or states of a process are stored so that they can be restored at a time when the processor resumes the execution of the process. Context switching is a feature of multitasking, which enables a single processor to be shared by multiple processes (Hennessy 2012, Muhammad 2020). The same concept can be applied at the low level when designing FSM models such as the one that was described in the previous chapter.

The generalised concept of context switching is depicted in Fig. 5.

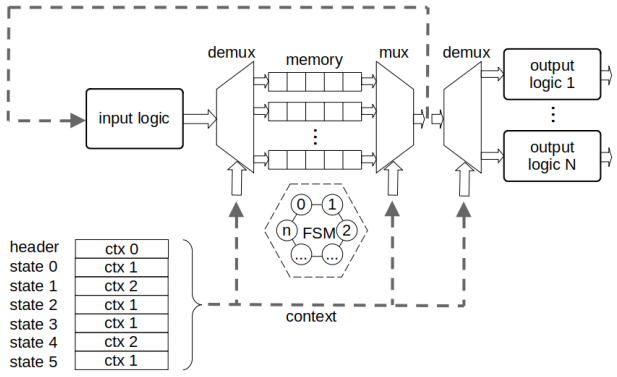

Fig. 5. Context switching technique applicable to finite state machine modeling

The example in this figure is based on the traffic light model but it is valid in the more general case. A comparison with Fig. 2 shows that the input logic and the state transitions table (equation (2)) remain the same while the contents of the state register are changing as it corresponds to distinct counting numbers. If replacing the latter functionality with a memory an array of values can be created. The exact value (memory row) can be stored and fed back to the input logic with the aid of a synchronously working pair of input demultiplexer and an output multiplexer. Their selector inputs are controlled by the switching matrix – a table that contains the number of the context. Using the same demultiplexing mechanism the output logic functions can also differ between each other depending on the context. Despite the fact that these functions are purely combinatorial, i.e. they do not possess memory, they will never lose their outputs because they are connected with the memory array that forms the state register.

For the example of the traffic light controller the up/down counter shown in Fig. 6 can be used. It is of a synchronous type which guarantees that no cumulative delay in the clock chain will be generated. Also this counter supports asynchronous reset and load.

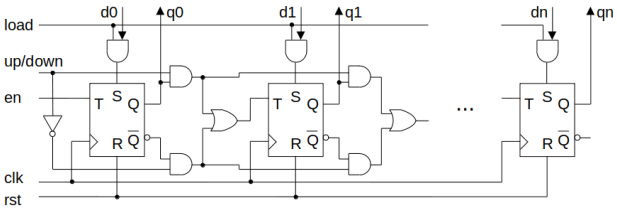

Fig. 6. An N-bit synchronous up/down counter built using T flip-flops

Instead of implementing the same circuit at the gate level, the behavioural modeling style could be used as it is much more versatile. The parameterised Verilog description of the N-bit synchronous behavioural up-down counter is given in Table 1. This code allows to easily reconfigure the bit-width of the counter without having to redesign it.

Table 1. *Parameterised Verilog code of the N-bit synchronous behavioural up-down counter*

```
module cntNbitUpDown #( parameter N = 4 ) (
   input                 i_clk,
   input                 i_rst,
   input                 i_dir,
   input                 i_load,
   input [N-1 : 0]       i_data,
   output reg [N-1 : 0]  o_cnt);

   always @(posedge i_clk or negedge i_rst or posedge i_load)
      if (~i_rst)
         o_cnt = 0;
      else
         if (i_load)
            o_cnt = i_data;
         else
            if (i_dir)
               o_cnt = o_cnt + 1;
            else
               o_cnt = o_cnt - 1;
endmodule
```

The parameterised Verilog description of the synchronous memory array is given in Table 2. This code allows to change the number of address bits and the word size by configuring parameters A_SIZE and W_SIZE.

Table 2. *Parameterized Verilog code of the memory module*

```
module mem
   #( parameter A_SIZE = 4,
      parameter W_SIZE = 8 ) (
   input                      i_clk,
   input                      i_rst,
   input [A_SIZE-1 : 0]    addr,
   input [W_SIZE-1 : 0]   i_data,
   input                      i_cs,
   output [W_SIZE-1 : 0] o_data );

   reg [W_SIZE-1 : 0]    mem    [0 : A_SIZE-1];

   assign o_data = mem[addr];

   integer i;
   always @(posedge i_clk or negedge i_rst) begin
      if (~i_rst | i_cs)
         if (~i_rst)
            for (i = 0; i < (A_SIZE-1); i = i + 1)
               mem[i] <= 0;
         else
            mem[addr] <= i_data;
   end
endmodule
```

If using the Xilinx (now AMD) Spartan-6 FPGA device, compiling this code will infer a properly configured Block RAM module (UG383 2011). Its general (unconfigured) overview is given in Fig. 7.

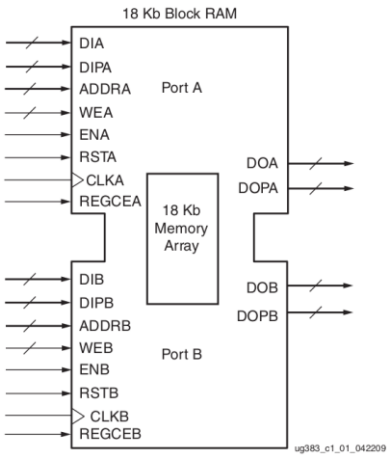

Fig. 7. Spartan-6 Block RAM memory can be configured as a single-port or dual-port RAM with up to 18 Kb

The Spartan-6 family has a total of 268 blocks of Block RAM that can store up to 18K bits of data. It can be used in either single-port or dual-port mode having independent bit-widths in two-port configuration. It is possible to store more than one look-up table into a single Block RAM. This can be done by using some simple arithmetic for calculating the separate memory spaces allocated for each LUT.

The simulation results for the counter using the described context switching approach are given in Fig. 8.

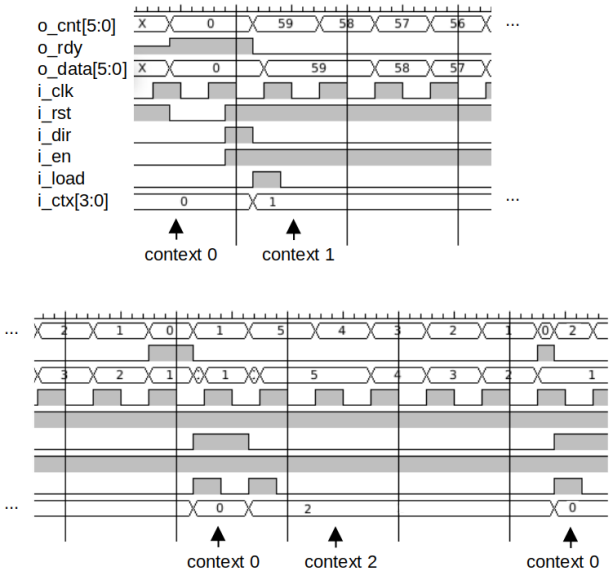

Fig. 8. Simulation output demonstrating the context switching

A single counter is used in the proposed algorithm. It can be seen that this counter is loaded with different values depending on the context. Context 0 is the main loop (nodes a0-a5 connected with dashed lines in Fig. 4) in which the counter counts up. The current state is stored in the memory and after switching the context it is restored back so the counter continues from where it stopped. Context 1 corresponds to the 59-seconds timer and context 2 corresponds to the 5-seconds timer. The simulation of the output functions is not shown in the figure.

The same approach can be successfully used for other purposes. As a brief example in computer architectures the interrupt servicing mechanism is using the stack memory which is in fact a LIFO (last in, first out) buffer. When servicing an interrupt the important register values are put on top of the stack. When exiting the interrupt service routine (which is simply a sub-program) those values are restored. If using the context switching approach a versatile version of a stackless interrupt servicing can be implemented. It would allow to jump between different subroutines without using the stack, i.e. with using priority concurrency. An idea for this is given in Fig. 9.

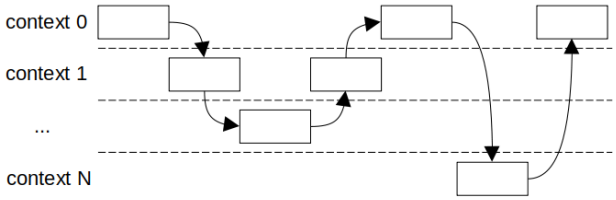

Fig. 9. Stackless interrupt servicing

In (Gorbounov 2022) another example is given. It thoroughly discusses an approach for building a high-performance neural network structure with only a single layer of neurons that is based on the switching context method. It allows to reduce the total amount of neurons in a neural

network thus significantly reducing the amount of logic resources occupied on the silicon die. In this case the contexts are switching between the different weight matrices and the matrices of the activation functions. As a result the number of physical neural network layers is significantly reduced.

Conclusion

Finite automata apparatus is a powerful tool for solving different control problems. The rush for increasing the speed of control algorithms and reducing the energy consumption at the same time poses important optimization challenge. This article suggests a possible solution towards solving this contemporary task. It is done by using a resource sharing mechanism called context switching. It allows to reuse some automata models by changing the state values held by the state register which in this case is replaced with a memory array. A simplified example of a traffic light controller is shown which demonstrates the viability of the proposed method. This opens a large field for future improvements.

References

Gorbounov Y., H. Chen. 2022. Context-switching neural node for constrained-space hardware, EAI CSECS 2022 - *EAI Conference on Computer Science and Education in Computer Science* (in print)

Harris, D., S. Harris. 2013. *Digital Design and Computer Architecture,* 2ed. Morgan Kaufmann Elsevier, ISBN 978-0-12-394424-5

Hennessy, J., D. Patterson. 2012. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Elsevier, ISBN 978-8178672663

Mano M., M. Ciletti. 2018. *Digital Design,* 6th ed. Pearson, ISBN 978-1-292-23116-7

Muhammad A., X. Jiao, W. Liu, I. Moerman. 2020. CMCVT: A Concurrent Multi-Channel Virtual Transceiver. - *International Journal of Electronics and Communications, 120*, doi.org/10.1016/j.aeue.2020.153230, ISSN 1434-8411

Palnitkar S. 2003. *Verilog HDL,* 2nd ed.. Prentice Hall, ISBN 978-0132599702

Pavlitov, C., Y. Gorbounov. 2007. Programmable logic in electromechanics. Technical University of Sofia, ISBN 978-954-438-645-0, pp.79-118 (in Bulgarian)

UG383 v1.5. 2011. Spartan-6 FPGA Block RAM Resources User Guide, www.xilinx.com, last accessed 2022/03/01