Article

ASMD-FSMD technique for designing digital devices on FPGA

Valery Salauyou [1]

[1] Faculty of Computer Science, Bialystok University of Technology; valsol@mail.ru; Tel.: +48 516 038 349

Abstract: Recently, there has been, on the one hand, an increase in the complexity of digital device designs and, on the other hand, an increase in the requirements for the development time and the reliability of the designs. One of the directions of solving this problem is developing new techniques for designing digital devices.This paper proposes a new technique for designing digital devices based on finite state machines with datapath (FSMD), when the functioning of the device is described in the form of an algorithm state machine with datapath (ASMD) charts. The new technique is called ASMD-FSMD. Different digital device design techniques are compared to each other using design examples of a synchronous multiplier on field programmable gate array (FPGA). The efficiency of the ASMD-FSMD technique compared to the traditional approach in terms of area and performance was investigated. The ASMD-FSMD technique, compared to the traditional one, reduces the area from 28.6% to 39.7% and increases the speed for some designs to 17.6%. In addition, using the ASMD-FSMD technique significantly reduces design time and increases design reliability. In conclusion, recommendations for using the ASMD-FSMD technique are made.

Keywords: digital device, finite state machines with datapath, algorithm state machine with datapath, field programmable gate array, design technique, development time, reliability, area, performance.

1. Introduction

Traditionally, the designed digital device is usually represented in the form of a datapath and a control unit (controller), which are designed separately. The datapath is in the form of a set of standard functional blocks (registers, buses, multiplexers, etc.), and the control unit is in the form of a finite state machine (FSM).

In [1], the control unit and the datapath are proposed to be combined together and presented as a finite state machine with datapath (FSMD). The FSMD model has quickly become popular. In [2], the FSMDs for synchronous and asynchronous designs are presented. The FSMD model proved to be very convenient for testing the equivalence of the two designs obtained from synthesis or various design transformations [3,4]. In [5], the digital system is proposed to be presented as an FSMD network, which leads to the implementation of racing-free hardware. The overall FSMD model is not always convenient when designing specific applications. In [6], formal FSMD models are presented for both the processor architecture and the ASIC (application-specific integrated circuit) architecture. In [7], the FSMD model with synchronous memory accesses is offered. In [8], the FSMD model for array-handling is discussed. The decomposition of FSMD to reduce the power of digital systems is presented in [9,10]. A comparison of the efficiency of FSMs and FSMDs is considered in [11]. It was shown that the Moore FSMD has the highest efficiency in relation to the canonical FSM.

Due to its visualization, the algorithm state machine (ASM) charts have become widespread to represent FSM behavior. The ASMs were first proposed in [12] as an alternative to state diagrams. In [13], PROM-, FPLA- and multiplexer-based implementations of the ASMs is considered. The minimization method of the number of ASM vertices is presented in [14]. In [15], the ABELITE tool for ASM-based controller synthesis is described.

Traditionally, the ASM charts were used to represent an operation algorithm of the control unit, i.e. FSM. In [16], it was proposed that the ASM is used to describe both the behavior of the control unit and the register operations that are performed in the datapath. For this purpose, the Moore ASM chart is used to represent the states of the control unit and ovals are inserted in the ASM transitions. The operations that are performed in the datapath on this transition are recorded in ovals. Such an ASM is called an algorithmic state machine with datapath (ASMD).

Recently, ASMD charts have been increasingly used in FPGA designs: when implementing industrial control systems [17]; to implement the asin function using the CORDIC algorithm [18]; during hardware implementation of cryptographic algorithm AES [19]; when designing a universal asynchronous receiver transmitter (UART) [20], etc.

This paper proposes the technique called ASMD-FSMD for designing digital devices on FPGA, based on the Verilog hardware description language (HDL). The ASMD-FSMD technique is illustrated by the design example of a synchronous multiplier.

The article is organized as follows. In section 2, an implemented multiplication algorithm is described. The traditional approach to designing the synchronous multiplier in the form of the datapath and the control unit is considered in 3. In 4, the ASMs are presented. Definitions of ASMD and FSMD is provided in 5. In 6, an ASMD modification to improve design performance is considered. A description of the FSMD in Verilog HDL is presented in 7. In 8, a formal representation of the ASMD-FSMD technique as an algorithm is provided. Experimental studies comparing the traditional approach and the ASMD-FSMD technique are presented in 9. The results of experimental studies are discussed in 10. The Conclusion provides recommendations for the effective use of the ASMD-FSMD technique.

2. Multiplication algorithm

A simple school multiplication algorithm performs an arithmetic operation of multiplying two binary unsigned numbers P = A*B, where A is a multiplicand, B is a multiplier, and P is a product. Let the width of the binary words A and B be the same and equal to N bits, then the product P will have a width of 2N bits. At the beginning of algorithm execution, P is zeroed out. In each multiplication cycle, the least significant bit of the multiplier B is checked. If B[0]=1, then the multiplicand A is added to the product P. If B[0]=0, then zero or nothing is added to the product P. Then the value of the multiplier B is shifted one bit to the right and the value of the product P is shifted one bit to the left. The multiplication algorithm ends after considering all bits of the multiplier B. The considered multiplication algorithm can be described in Verilog as follows.

```
module algorithm_mult #(parameter N=4)
    (input [N-1:0] a,b,                 // input words
    output [2*N-1:0] p);                // product
    reg [N-1:0] rb;                     // intermediate variables
    reg [2*N-1:0] ra,rp;
    integer i;                          // loop variable
    always @(*)                         // procedural block
    begin
        ra = a;    rb = b;    rp = 0;   // initialization of variables
        for (i=0; i<N; i=i+1)           // multiplication cycle
        begin
            // adding of partial products
            if (rb[0])  rp = rp + ra;
            else        rp = rp + {2*N{1'b0}};
            rb = rb >> 1;               // shift of variable values
            ra = ra << 1;
        end
    end
    assign p = rp;                      // forming of result
endmodule
```

In fact, the above code describes a combinational circuit rather than a synchronous multiplier. To obtain the synchronous multiplier, the registers must be set on the input (a and b) and output (p) buses, which are clocked by the clock. For such a multiplier, the multiplication operation will always be performed in one clock cycle.

The algorithm_mult design has a fairly high speed. However, the implementation cost (area) of the algorithm_mult design increases rapidly with an increase in the width N of the input words A and B. Therefore, algorithmic description of digital devices is rarely used for hardware implementation. This paper discusses a synchronous multiplier when each multiplication cycle is performed in one or more clock cycles.

3. Traditional approach for synchronous multiplier implementation

In the case of the traditional approach, the hardware implementation of the synchronous multiplier is a sequential circuit, the inputs of which are the words A and B, and a value of the product P is generated at output. The circuit is controlled by the clk and reset signals. After setting the multiplied word values at inputs A and B, the multiplication process is started by setting the Run signal to one. The end of the multiplication process is indicated by a value one at the output Done, after which the product value can be read from the output P.

Figure 1 shows a block diagram of the synchronous multiplier in the form of the datapath and a control unit, which is implemented as the FSM. The values of the multiplied words A and B are the inputs of the datapath. The P product and the Done signal are generated at the outputs of the datapath. In addition, the datapath generates the roll signal, which is the output of the modulo counter and indicates the end of the multiplication process.

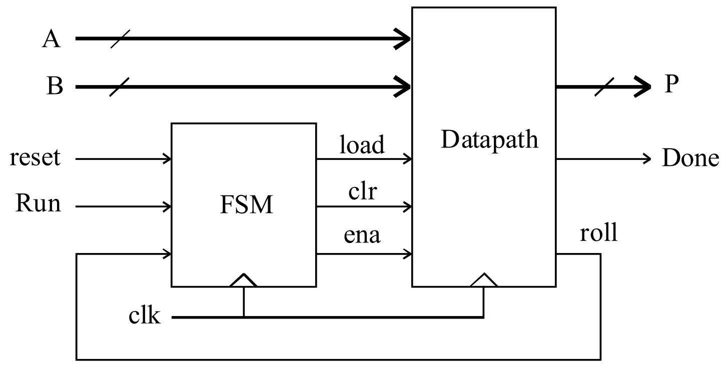

Figure 1. Block diagram of synchronous multiplier in the form of datapath and FSM

The external reset and Run signals, as well as the internal roll signal, are inputs of the FSM. The FSM generates the following control signals: *clr* - to reset the registers of the datapath; *load* - to load values of the multiplied words *A* and *B* into the datapath registers; *ena* - to allow shift operations. The *clk* clock synchronizes both the datapath and the FSM.

A diagram of the datapath that implements the multiplication algorithm is shown in Figure 2.

doi:10.20944/preprints202010.0043.v1

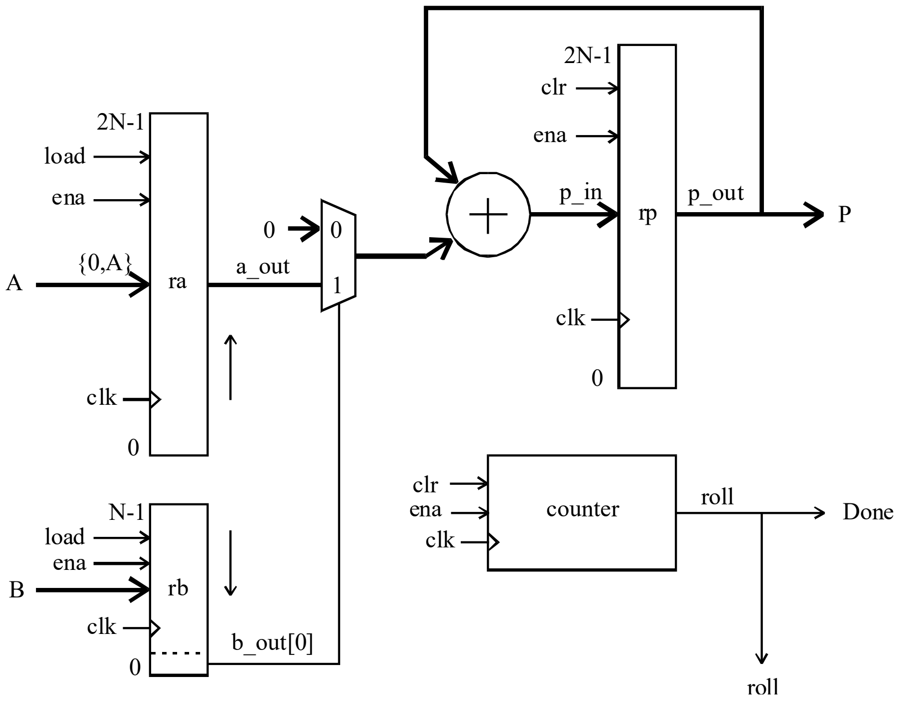

Figure 2. Diagram of the datapath for the multiplication algorithm

The datapath includes the shift register ra to the left by 2N bits for storing the multiplied A, the shift register rb to the right by N bits for storing the multiplier B, the bus multiplexer 2-1 by 2N bits, the adder by 2N bits, and the counter that generates the *roll* signal indicating the end of the multiplication process. The *roll* signal is the same as the external *Done* signal.

The control unit is an FSM, and both a Mealy type machine (Figure 3,a) and a Moore type machine (Figure 3,b) can be used.

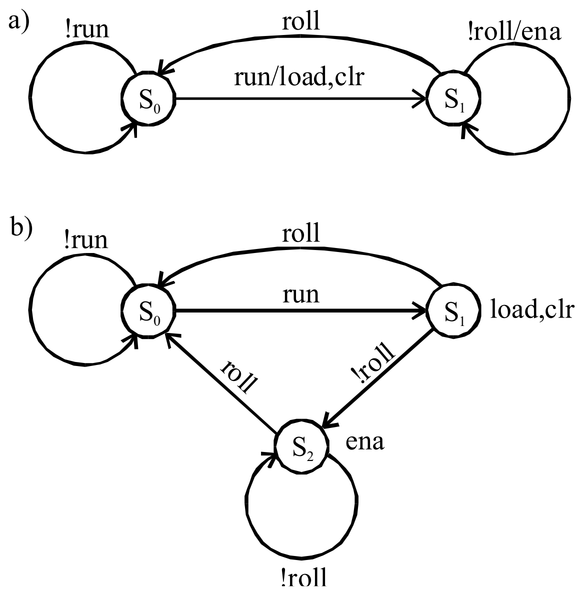

Figure 3. Control unit representation of the synchronous multiplier as a state diagram: Mealy FSM (a), Moore FSM (b)

Note that a Mealy FSM contains two states: S_0 and S_1, while a Moore FSM contains three states: S_0, S_1, and S_2.

To implement an FPGA multiplier, all components of the multiplier must be represented in one of the HDL. In our case, the Verilog HDL is used. A detailed description of datapath components (Figure 2) and Mealy and Moore FSMs (Figure 3) for the implementation of a synchronous multiplier in the case of the traditional approach is given in [21].

4. Algorithmic state machine charts

The algorithmic state machine (ASM) chart is intended to visually describe the behavior of the state machine and it is an oriented graph [12] containing vertices of three types: a state box (a rectangle); a decision box (a rhombus); a conditional output box (an oval).

The state box (a rectangle) determines the state of the FSM. A state name (such as S0, START, INITIAL, etc.) can be written near the state box, as well as a binary state code. In the case of FSM Moore, outputs that are equal to one in this state are written within the state box. By default, all other outputs in this state are equal to zero. Note that ASM does not allow you to describe FSMs with undefined output values.

The decision box (a rhombus) is the branching point of the FSM algorithm. The decision box records the conditions to be checked. The outputs of the decision box are denoted by values 0 and 1, which correspond to transitions in the case of a zero (false) or a unit (true) value of the condition check result. The condition can be an input variable of an FSM, a logical expression, or a single bit of a bit vector.

In the conditional output boxes (ovals), the Mealy FSM outputs are written, which are equal to one on the given transition.

In ASM charts for Moore FSMs, there are no conditional output boxes (ovals), and in ASM charts for Mealy FSMs nothing is written into the state box.

The main building element of the ASM chart is the ASM block (Figure 4). An ASM block consists of one state box and an optional network of decision boxes (rhombuses) and conditional output boxes (ovals). Note that rhombuses can precede ovals and follow ovals.

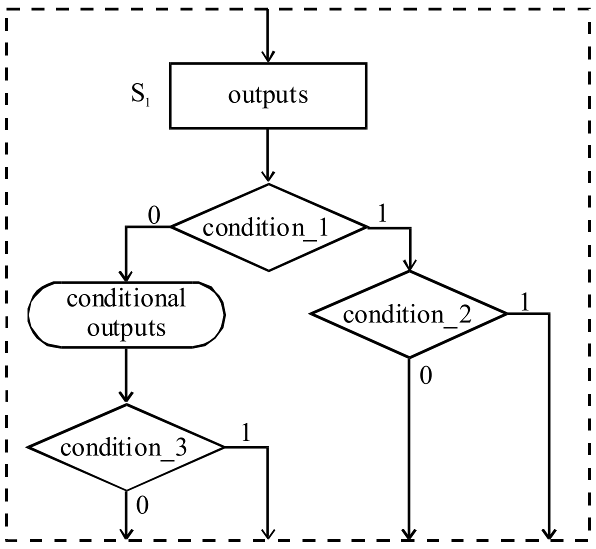

Figure 4. ASM block

The ASM chart is a composition of interconnected ASM blocks. The following ASM building rules must be observed: each output of any ASM vertex can be connected to only one input of another vertex, that is, branching of the FSM algorithm is possible only in decision boxes; feedbacks are not allowed inside the ASM blocks.

Note some characteristic features embedded in the nature of the ASM charts. The ASM charts always explicitly identify FSM states using the ASM blocks. The transition from one ASM block to another is always performed in one clock cycle, that is, the FSM behavior is directly tied to automata time. Using the ASM chart, the developer can always track how many clock cycles some fragment of the FSM algorithm will be performed. In addition, ASM initially determines the type of FSM: Mealy or Moore.

The following is the algorithm for building the ASM chart.

Algorithm 1. ASM chart construction technique

1. The FSM states are determined.
2. The ASM block is constructed for each state. For each state, all transitions from that state are defined for all possible values of input variables. If some input variable does not affect transitions from this state, it does not occur inside the ASM block.
3. For the Moore FSM, the output variables are written inside the state boxes (rectangles) in which they take the value of one in that state.
4. For the Mealy FSM, the output variables are written inside the conditional output boxes (ovals) in which they take the value of one on that transition.
5. The ASM blocks are connected to each other in accordance with the FSM algorithm. Each output of the ASM block can be connected to only one input of this or another ASM block.
6. End.

Note that the construction of the ASM chart begins with the determination of the FSM states, that is, from the very beginning, the developer uniquely determines the FSM states. Then, transitions from each state are determined. Next, the values of the output variables are determined. The last step connects the ASM blocks to each other, which allows the developer to once again check the correctness of the ASM diagram.

5. ASMD charts and FSM with datapath (FSMD)

The ASM chart with datapath (ASMD) is an ASM chart in which any register operations that are valid in Verilog can be written in rectangles and ovals and any Verilog logical expressions can be checked in the decision box. The ASMD chart, as well as the ASM chart, consists of ASMD blocks. The actions described within the ASMD block are performed during one clock cycle.

The FSM implemented based on ASMD will be called a *finite state machine with datapath* (*FSMD*), and the FSMD design technique based on ASMD, an *ASMD-FSMD technique*.

Note that in our case, unlike [1], FSMD is not separated into the control unit (FSM) and the datapath, but more closely resembles a behavioral (algorithmic) description of device functioning. However, in contrast to the algorithmic description (design algorithm_mult), the FSMD states are explicitly defined in ASMD.

Our approach to building the ASMD chart is also different from the ASMD chart defined in [16]. Our ASMD chart can describe any type of machine, both Mealy and Moore (in [16] Moore only), and operations performed in a datapath can be written in both ovals and rectangles (in [16] ovals only).

Figure 5 shows the ASMD that corresponds to the Moore FSMD for implementing our multiplication algorithm.

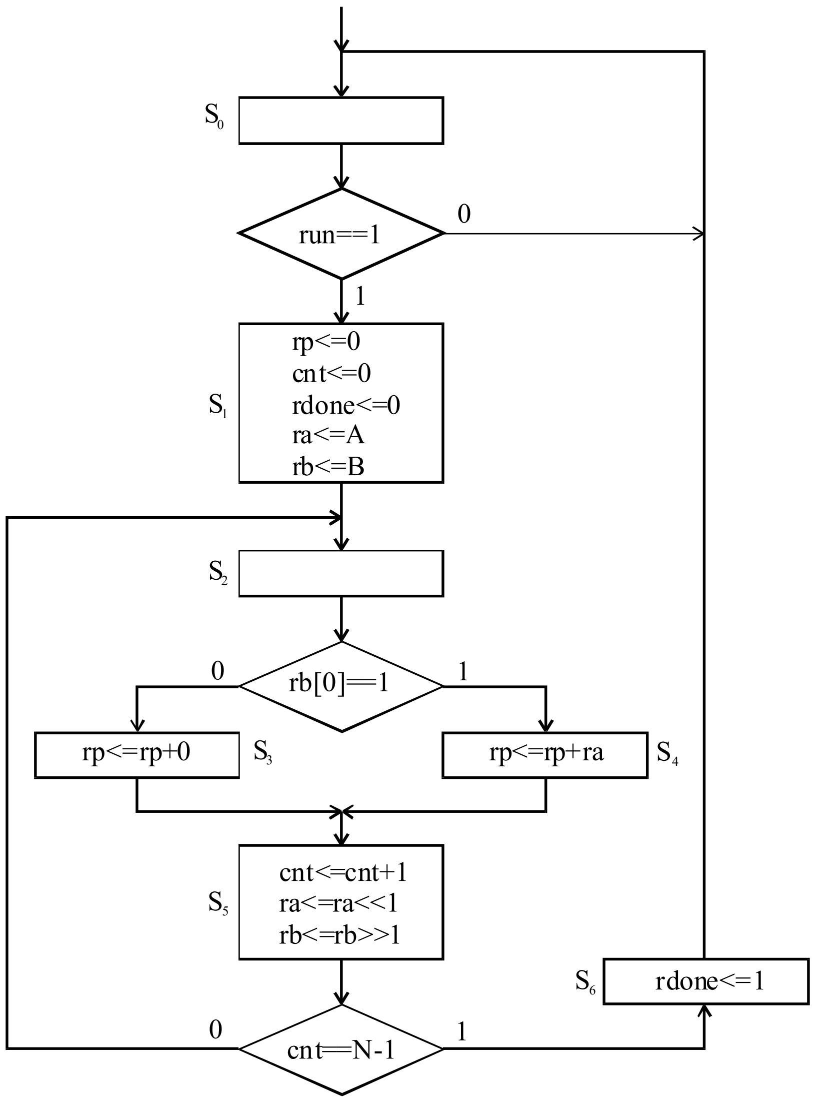

Figure 5. ASMD chart describing the multiplication algorithm as the Moore FSMD

In the S_0 state, a *run* signal is expected to be set, which triggers the multiplication process. In the S_1 state, the product register rp, the counter cnt, and the flip-flop rdone are reset. The output of the rdone flip-flop generates the *done* signal indicating that the multiplication process is complete. In addition, the initial values of registers ra and rb are loaded in the S_1 state.

The next S_2 state is the entry point to the multiplication cycle. One multiplication cycle corresponds to forming a partial product, adding it to the product and performing the necessary register shifts, as well as increasing the counter value. In the S_2 state, depending on the rb[0] value, you must add either a null value or the ra register value to the rp register. However, since ASMD describes a Moore machine, these actions can only be performed in separate states S_3 and S_4.

The state S_3 or S_4 is followed by the state S_5, in which the value of the counter cnt is increased and the ra and rb registers are shifted. The cnt counter value is also checked in the S_5 state. If the value of cnt is N-1, the FSM state is set to S_6, otherwise the FSM state is set to S_2 to perform a new iteration of the multiplication process. In the S_6 state, the rdone flip-flop is set to one and the transition is executed to the initial state S_0.

Directly by ASMD in Figure 5, you can determine the number of n_{Moore} clock cycles required to multiply N-bit binary numbers. The initial S_0 state for Moore FSMs always requires one clock cycle. In the S_1 state, the registers are initialized and loaded, which requires another clock cycle. Performing one multiplication cycle associated with processing one bit of multiplier B requires three synchronization cycles (states S_2, S_3 or S_4 and S_5). Generation of the *rdone* signal also requires one clock cycle (state S_6). Thus, the ASMD for the Moore machine multiplies the N-bit binary numbers by n_{Moore} clock cycles, where

$$n_{Moore} = 3N + 3. \quad (1)$$

Note that fairness (1) is also confirmed by the results of time modeling.

Figure 6 shows the ASMD that corresponds to the Mealy FSMD to implement our multiplication algorithm.

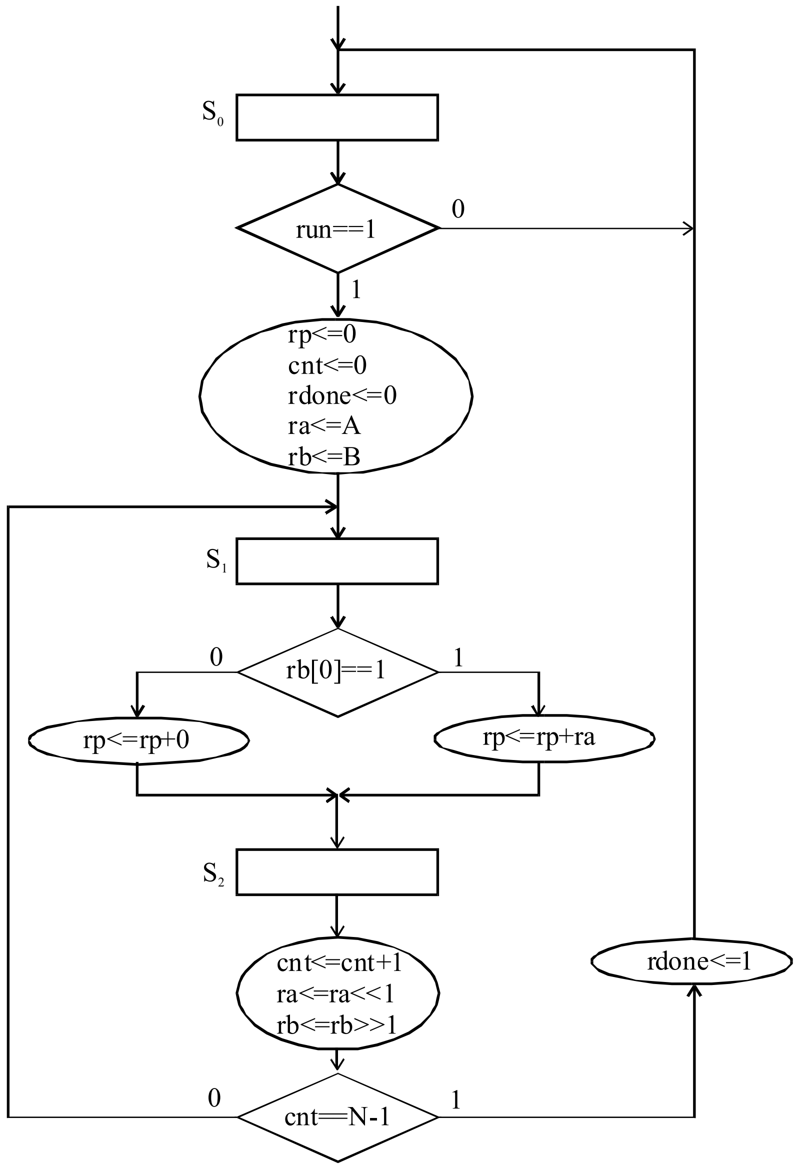

Figure 6. ASMD chart describing the multiplication algorithm as the Mealy FSMD

In the initial state S0, the machine waits for the *run* signal to be set to one. After setting the *run* signal to one, the multiplier registers are initialized.

The next S_1 state is the entry point to the multiplication cycle. In the S_1 state, the value of the rb[0] bit is checked and, depending on its value, either zero or ra is added to the contents of the rp register. Note that two separate states are not required here, as in Moore ASMD (the S_3 and S_4 states in Figure 5).

In the S_2 state, the necessary shifts of ra and rb registers are performed, as well as the cnt counter is incremented. Then, in state S_2, the previous counter value is checked, if it is equal to N-1, then the multiplication algorithm ends and the machine goes into state S_0 with the setting of the rdone register. Otherwise, the cycle of the multiplication process is repeated.

Figure 6 shows that one multiplication cycle is performed in two clock cycles, so the ASMD for the Mealy FSM multiplies N-bit binary numbers by the n_{Mealy} clock cycles, where

$$n_{Mealy} = 2N + 1. \tag{2}$$

A comparison of the Moore FSMD and the Mealy FSMD from our example shows that the Mealy FSMD has significantly fewer states (3 and 7, respectively) and also operates significantly faster, since each cycle of the multiplication process is performed in two clock cycles (in the Moore FSMD, each cycle of the multiplication process is performed in three clock cycles).

6. Performance increase of the synchronous multiplier

The fact that one iteration of the multiplication process is performed in several clock cycles makes the practical use of the above mentioned multiplier designs extremely ineffective, since it significantly increases the multiplication time. How do you develop a FSMD that performs each multiplication cycle in one clock cycle? To do this, you need to build an ASMD chart in which one multiplication cycle corresponds to one ASMD block. The ASMD chart for the Mealy machine that meets this requirement is shown in Figure 7.

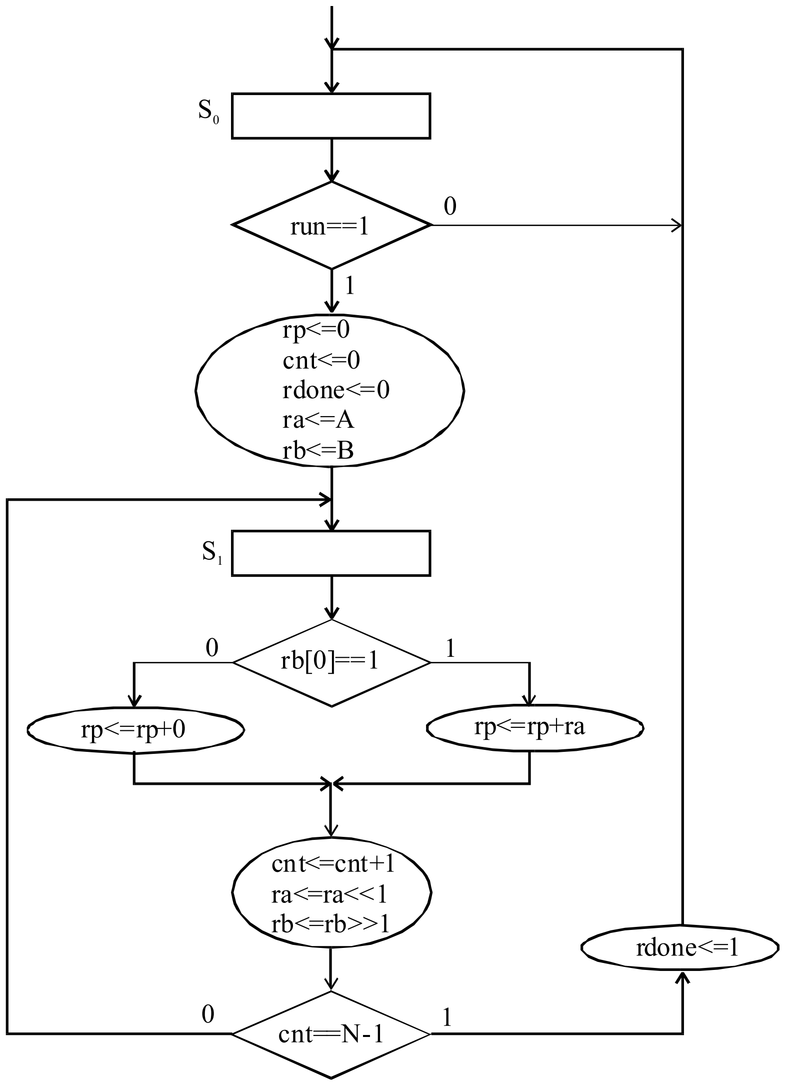

Figure 7. ASMD chart for the Mealy machine, providing the highest performance of the synchronous multiplier

A feature of the ASMD chart in Figure 7 is that there are only two ASMD blocks: in the S_0 state, registers are initialized, and the S_1 state corresponds to the full multiplication cycle.

Figure 7 shows that one multiplication cycle is performed in one clock cycle, so the multiplication of N-bit binary numbers according to ASMD in Figure 7 is carried out in n_{Mealy_1} clock cycles, where

$$n_{Mealy\_1} = N + 1. \tag{3}$$

To increase the performance of the Moore FSMD, we convert ASMD in Figure 5 to reduce the number of states in the multiplication cycle. The converted ASMD is shown in Figure 8.

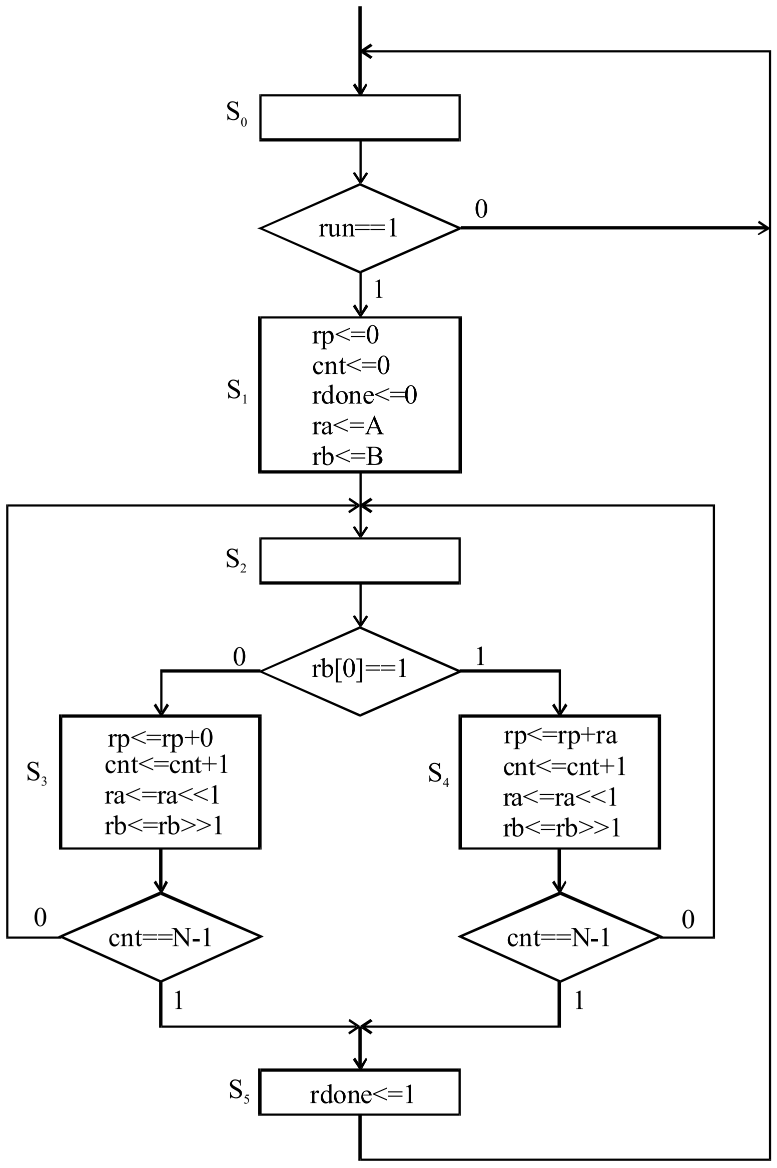

Figure 8. ASMD chart for the Moore machine, providing increased performance of the synchronous multiplier

In Figure 8, the counter increment and register shift operations are performed in states S_3 and S_4. This eliminates the S_5 state from the ASMD in Figure 5. Unfortunately, in Moore ASMD in Figure 8, it is not possible to describe a multiplication cycle using one state, as for the Mealy ASMD. The S_2 state is required to enter in the multiplication cycle. Then the rb[0] bit is checked. Taking actions, depending on the value of rb[0], also requires at least one more state (the S_3 or S_4 state).

Thus, the multiplication cycle passes through two Moore FSMD states, i.e., is performed in two clock cycles. As a result, the multiplication of the N-bit binary numbers according to ASMD in Figure 8 is carried out in n_{Moore_1} clock cycles, where

$$n_{Moore\_1} = 2N + 1. \tag{4}$$

7. Description of FSMD in Verilog

To implement the synchronous multiplier on the FPGA in the form of the Mealy FSM, the ASMD chart in Figure 7 is used. The design code of the Mealy FSMD in Verilog is given below.

```
module mult_FSMD_Mealy #(parameter N=4)
        (input clk, reset, run,              // control signals
        input [N-1:0] a,b,                   // input words
        output [2*N-1:0] p,                  // product
        output done);                        // signal of the multiplication end
        reg [2*N-1:0] ra,rp;                 // declaration of the registers
        reg [N-1:0] rb;
        reg rdone;
        reg [N_cnt-1:0] cnt;                 // counter
        localparam N_cnt=clogb2(N-1);        // determination of the counter size
        function integer clogb2(input [N-1:0] v);    // constant function
                for (clogb2 = 0; v > 0; clogb2 = clogb2 + 1)
                        v = v >> 1;
        endfunction
        localparam [0:0] s0=0,s1=1;          // declaration of the FSM states
        reg [0:0] state;                     // state variable
        always @(posedge clk)                // beginning of the multiplication cycle
                if(reset)   state <= s0;
                else
                        case (state)
                        s0:     if(run)
                                        begin
                                                rp <= 0; cnt <= 0; rdone <= 0;
                                                ra <= {{N{1'b0}},a};
                                                rb <= b;
                                                state <= s1;
                                        end
                                else  state <= s0;
                        s1:             begin
                                                if(rb[0])   rp <= rp + ra;
                                                else        rp <= rp + {2*N{1'b0}};
                                                cnt <= cnt + 1'b1;
                                                rb <= rb >> 1;
                                                ra <= ra << 1;
                                                if (cnt == N-1)
                                                begin
                                                        rdone <= 1'b1;
                                                        state <= s0;
                                                end
                                                else  state <= s1;
                                        end
                        default: state <= s0;
                endcase
        assign p = rp;                      // to form of the output values
        assign done = rdone;
endmodule
```

In the mult_FSMD_Mealy module, an FSM is described in the description style with a single process [21]. This is permissible, since the *p* product and the *done* signal in the multiplier are generated on the output of the registers. The above code uses a constant function clogb2 [21] to determine the size of the counter, which calculates the smallest integer greater than or equal to the binary logarithm of value v.

The Moore FSMD design code in Verilog, which corresponds to the ASMD chart in Figure 8, is as follows.

```
module mult_FSMD_Moore #(parameter N=4)
        (input clk, reset, run,             // control signals
        input [N-1:0] a,b,                  // input words
        output [2*N-1:0] p,                 // the product
        output done);                       // signal of the multiplication end
        reg [2*N-1:0] ra,rp;                // declaration of the registers
        reg [N-1:0] rb;
        reg rdone;
        reg [N_cnt-1:0] cnt;                // the counter
        localparam N_cnt=clogb2(N-1);   // determination of the counter size
        function integer clogb2(input [N-1:0] v);     // constant function
                for (clogb2 = 0; v > 0; clogb2 = clogb2 + 1)
                        v = v >> 1;
        endfunction
        localparam [2:0] s0=0,s1=1,s2=2,s3=3,s4=4,s5=5;  // declaration of the FSM states
        reg [2:0] state;                    // state variable
        always @(posedge clk)               // beginning of the multiplication cycle
                if(reset)   state <= s0;
                else
                        case (state)
                        s0:  if(run)          state <= s1;
                             else       state <= s0;
                        s1:  begin
                                        rp <= 0; cnt <= 0; rdone <= 0;
                                        ra <= {{N{1'b0}},a};
                                        rb <= b;
                                        state <= s2;
                             end
                        s2:  if(rb[0])  state <= s4;
                             else       state <= s3;
                        s3:  begin
                                        rp <= rp + {2*N{1'b0}};
                                        cnt <= cnt + 1'b1;
                                        rb <= rb >> 1;
                                        ra <= ra << 1;
                                        if (cnt == N-1)       state <= s5;
                                        else                  state <= s2;
                             end
                        s4:  begin
```

```
                                  rp <= rp + ra;
                                  cnt <= cnt + 1'b1;
                                  rb <= rb >> 1;
                                  ra <= ra << 1;
                                  if (cnt == N-1)       state <= s5;
                                  else                  state <= s2;
                         end
                 s5:     begin
                                  rdone <= 1'b1;
                                  state <= s0;
                         end
                 endcase
        assign p = rp;                        // to form of the output values
        assign done = rdone;
    endmodule
```

In the mult_FSMD_Moore module, an FSM is described also with a single process [21]. In this case, the next state is determined in each state, as well as all actions performed on the registers according to the ASMD chart in Figure 8 are described.

8. The ASMD-FSMD technique of the digital device design

The formal description of the discussed ASMD-FSMD technique can be presented as follows.

Algorithm 2. The ASMD-FSMD technique of the digital device design**.**

1. The FSMD states are determined.
2. The ASMD block is constructed for each FSMD state.
   2.1. In the ASMD decision boxes, the logical functions are written, the values of which are checked in this state.
   2.2. For the Moore FSMD in the state boxes (rectangles), the operations that are performed on the content of the registers in this state are written.
   2.3. For the Mealy FSMD in the conditional output boxes (ovals), the operations that are performed on the content of the registers on this transition are written.
3. The ASMD blocks are connected to each other in accordance with the algorithm of the device operation. Each output of the ASMD block can be connected to only one input of this or other ASMD block.
4. If necessary, the ASMD is modified to increase the performance of the designed device. To do this, the algorithm cycles are analyzed and the ASMD is changed in such a way as to minimize the number of states in the cycle path.
5. The Verilog-code of the FSMD is built directly by ASMD. In Verilog code, the variables correspond to the device registers. The logical expressions in the **if** statements correspond to the logical functions checked in the ASMD decision boxes. The actions performed in ASMD blocks are described as procedural blocks **begin...end**. In each block for the Moore FSMD, the operations performed in this ASMD state are described, and the next states according to the ASMD transitions are determined. In each block for the Mealy FSMD, all the actions performed in the corresponding ASMD block are described in the style of the algorithmic description.
6. End.

Note some of the features of the ASMD-FSMD technique. The main design stage is building an ASMD chart based on the behavior (the operating algorithm) of a digital device. There is no strict separation of the digital device into the datapath and the control unit (FSM). At the same time, the ASMD chart determines the FSM states that correspond to the states of the control unit. This allows you to bind the algorithm of the functioning of the designed device to synchronization clocks. Therefore, according to the ASMD chart, the developer can track how many clock cycles each branch of the algorithm performs. In addition, the ASMD chart does not explicitly define the structure of the datapath.

The ASMD-FSMD technique differs from those known in that the ASMD chart can describe both the Moore FSM and the Mealy FSM (in [16] only the Moore FSM). The actions performed in the datapath can be written in both rectangles and ovals (in [16] only in ovals).

Note also that the same operating algorithm of the device can be described by different ASMD charts, this affects the performance and the implementation cost (area) of the design. To increase performance, the algorithm loops should be described with a minimum number of states to minimize the number of states in the loop path. Mealy FSMs are better suited for this purpose.

9. Experimental research

Using the example of the simple synchronous multiplier, three techniques for designing digital devices were considered: an algorithmic implementation in the Verilog HDL, a traditional technique (in the form of a composition of the datapath and the control unit), and a technique using an FSMD (the ASMD-FSMD technique). To check the effectiveness of these techniques, the following designs of synchronous multipliers were studied:

- algorithm_mult – the algorithmic implementation in the Verilog HDL;
- mult_FSM_Mealy – the multiplier built in the form of the datapath and the control unit when a Mealy FSM is used as the control unit;
- mult_FSM_Moore – the multiplier built in the form of the datapath and the control unit when a Moore FSM is used as the control unit;
- mult_FSMD_Mealy – the multiplier built according to ASMD for Mealy FSMD in Figure 7;
- mult_FSMD_Moore – the multiplier built according to ASMD for Moore FSMD in Figure 8.

The results of the multiplier designs study in the Quartus Prime tool version 18.1 for the FPGA Cyclone IV E family are shown in Tables 1 - 5, where N is the width of the input words; L is the number of the FPGA logical elements used (the implementation cost or the area); F is the maximum frequency of the design operation in megahertz (speed); T is the duration of one clock cycle in nanoseconds, T = 1/F; n is the number of clock cycles required to calculate the result; t is the time (in nanoseconds) required to calculate the result, t = T*n. For design algorithm_mult n is 1, for design mult_FSM_Mealy n is (N + 1), for design mult_FSM_Moore n is (N + 2), for design mult_FSMD_Mealy n is defined as (3), and for design mult_FSMD_Moore n is defined as (4).

Table 1. Results of design study algorithm_mult

N	L	F	T	n	t
4	34	130.23	7.679	1	7.679
8	129	75.00	13.333	1	13.333
16	513	30.39	32.916	1	32.916
32	2043	11.28	88.652	1	88.652
64	8187	4.47	223.714	1	223.714
128	32854	0.99	1010.101	1	1010.101

Table 2. Results of design study mult_FSM_Mealy

N	L	F	T	n	t
4	36	245.28	4.077	5	20.39
8	66	212.95	4.696	9	42.26
16	123	224.77	4.449	17	75.63
32	236	155.23	6.442	33	212.59
64	464	110.29	9.067	65	589.36
128	919	31.43	31.817	129	4104.39

Table 3. Results of design study mult_FSM_Moore

N	L	F	T	n	t
4	35	282.84	3.536	6	21.22
8	64	159.26	6.279	10	62.79
16	124	177.65	5.659	18	101.86
32	237	157.01	6.369	34	216.55
64	462	110.19	9.075	66	598.95
128	916	31.57	31.676	130	4117.88

Table 4. Results of design study mult_FSMD_Mealy

N	L	F	T	n	t
4	28	285.88	3.498	5	17.49
8	49	228.89	4.369	9	39.32
16	90	202.18	4.946	17	84.08
32	172	172.06	5.812	33	191.80
64	335	111.64	8.957	65	582.21
128	657	32.50	30.769	129	3969.20

Table 5. Results of design study mult_FSMD_Moore

N	L	F	T	n	t
4	48	192.98	5.182	11	57.00
8	102	174.19	5.741	19	109.08
16	192	155.13	6.446	35	225.61
32	304	152.65	6.551	67	438.92
64	578	104.98	9.526	131	1247.91
128	1145	29.64	33.738	259	8738.14

To more clearly compare the designs in question, Tables 6 and 7 show the implementation cost L and the time t of performing the multiplication operation in different designs for different values of N.

Table 6. Implementation cost L of the multipliers (the number of FPGA logical elements)

Designs	N=4	N=8	N=16	N=32	N=64	N=128
algorithm_mult	34	129	513	2043	8187	32854
mult_FSM_Moore	35	64	124	237	462	916
mult_FSM_Mealy	36	66	123	236	464	919
mult_FSMD_Moore	47	102	192	309	583	1139
mult_FSMD_Mealy	28	49	90	172	335	657

Table 7. Multiplication time t (in nanoseconds)

Designs	N=4	N=8	N=16	N=32	N=64	N=128
algorithm_mult	8	13	33	89	224	1010
mult_FSM_Moore	21	63	102	217	599	4118
mult_FSM_Mealy	20	42	76	213	589	4104
mult_FSMD_Moore	76	156	345	663	1831	12232
mult_FSMD_Mealy	17	39	84	192	582	3969

10. Discussion of results

From Tables 6 and 7, it can be seen that when using the traditional digital device design technique (designs mult_FSM_Moore and mult_FSM_Mealy), the type of FSM (Mealy or Moore) does not play a special role, since the implementation cost and the speed for designs mult_FSM_Moore and mult_FSM_Mealy are approximately the same.

In the case of using the ASMD-FSMD technique, the mult_FSMD_Mealy design has a clear advantage over the mult_FSMD_Moore design in terms of both area and performance. The advantage of the mult_FSMD_Mealy design in terms of area is that the ASMD multiplication cycle for the Moore FSM in Figure 8 duplicates a number of operations (counter increase, register shift), and the performance advantage is that one multiplication cycle in the ASMD chart in Figure 8 is performed in two synchronization cycles. Thus, when using the ASMD-FSMD technique, preference should be given to Mealy FSM.

Comparison of the ASMD-FSMD technique (design mult_FSMD_Mealy) with the traditional technique (designs mult_FSM_Moore and mult_FSM_Mealy) shows that the ASMD-FSMD technique exceeds the traditional technique both in area (from 28.6% to 39.7%) and speed (maximum 17.6% for the design mult_FSM_Moore at N = 4). Figure 9 shows the area advantage of the ASMD-FSMD technology over the traditional technology.

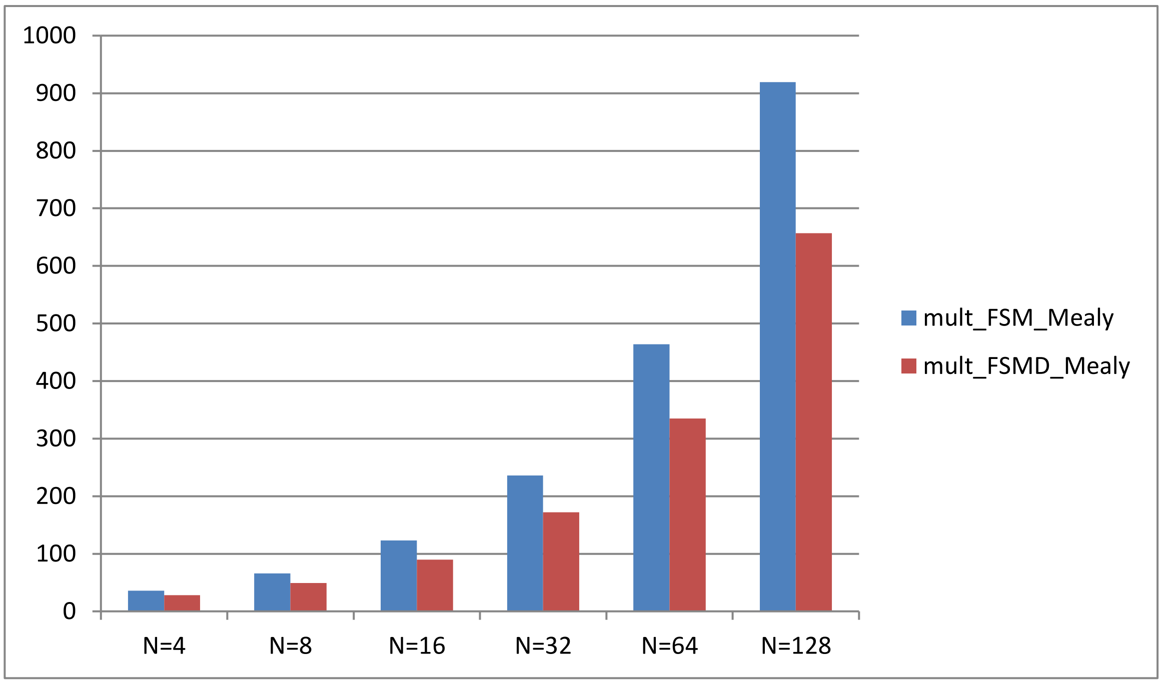

Figure 9. Advantage of the ASMD-FSMD technique (design mult_FSM_Mealy) over the traditional technique (design mult_FSMD_Mealy) in the implementation cost

Note that the ASMD-FSMD technique allows to describe the design code in the Verilog HDL directly according to the ASMD chart. Therefore, using the ASMD-FSMD technique allows you to significantly reduce the design time compared to the traditional approach, since there is no need to design the datapath and all its components, as well as the control unit and the top-level module.

The ASMD-FSMD technique also improves the reliability of the designs. The fact is that many stages of traditional design are performed by the developer manually and are the source of difficult to detect errors. In the ASMD-FSMD technique, after the ASMD chart is built, the Verilog description of the design is performed directly by the ASMD chart without any intermediate descriptions.

In order to quantify the design time and reliability of the synchronous multiplier designs in the case of using the traditional approach and the ASMD-FSMD technique, we compare the code length (the number of rows) of the mult_FSM_Mealy design (the traditional approach) and the mult_FSMD_Mealy design (the ASMD-FSMD technique). The design mult_FSM_Mealy consists of the top-level module (mult_a_Mealy), the control device (fsm_Mealy) and the datapath (datapath_mult). In turn, the datapath contains the following components: the adder, the modulo counter (counter_modulo_M), the bus multiplexer (mux_2), the left shift register (shl_load), and the right shift register (shr_load), as well as the regular register (register). The number of rows for each module is shown in Table 8.

Table 8. Number of design code lines mult_FSMD_Mealy and mult_FSM_Mealy

Module name	Number of code lines
mult_a_Mealy	17
fsm_Mealy	28
datapath_mult	26
adder	8
counter_modulo_M	23
mux_2	9
shl_load	11
shr_load	11
register	11
∑ (mult_FSMD_Mealy)	144
mult_FSMD_Mealy	58

Table 8 shows that using the ASMD-FSMD technique, compared to the traditional approach, reduces the design code length by a factor 144/58 = 2.483, i.e. about by a factor 2.5. Let the design development time be directly proportional to the design code length. It can be assumed that for our example of the synchronous multiplier, using the ASMD-FSMD technique reduces the design time by a factor 2.5.

It is more difficult to quantify the increase in design reliability as a result of the application of the ASMD-FSMD technique. It can be assumed that the reliability of the design is also proportional to the design code length. However, the number of links between modules and a number of other factors should be taken into account when evaluating reliability. If reliability is estimated only by the number of the design code length, then the use of the ASMD-FSMD technique in our case allows you to increase the reliability of the design at least by a factor 2.5.

The ASMD-FSMD technique has special requirements for building the ASMD chart. Experience with ASMD-FSMD has shown that certain rules must be followed for its effective use. First of all, you should strive to build ASMD charts for Mealy FSMs. If the operating algorithm of the device contains loops, then each cycle in the ASMD chart should be described by one state. If this is not possible, use the minimum number of states.

The algorithmic implementation of the multiplier (design algorithm_mult) significantly exceeds all the designs considered in terms of speed (3-5 times), but is inferior in terms of an area, which rapidly increases with an increase in the number of N bits of input words. A technique for designing digital devices based on an algorithmic implementation can be recommended for building small devices, as well as for modeling, in order to check the algorithm of the device.

11. Conclusions

Using the example of synchronous multiplier design, three techniques for designing digital devices were considered: the algorithmic implementation in the Verilog language, the traditional technique (in the form of a datapath and an FSM), and the ASMD-FSMD technique based on FSMDs. It was shown that the application of the ASMD-FSMD technique allows to reduce the area and increase the speed of the designs compared to the traditional technique. However, the main advantage of the ASMD-FSMD technique is a significant reduction in design time and an increase in design reliability compared to the traditional approach.

The ASMD-FSMD technique can be used in the design of digital devices not only on FPGA, but also on the basis of ASIC. Any hardware description language such as VHDL or SystemVerilog can be used as a design description language. The use of the ASMD-FSMD technique for high-level design of complex designs is also seen as a promising direction.

Acknowledgments: The present study was supported by a grant WZ/WI-IIT/4/2020 from Bialystok University of Technology and founded from the resources for research by Ministry of Science and Higher Education.

References

1. Gajski, D.D.; Dutt, N.D.; Wu A.C.; Lin S.Y. *High—Level Synthesis: Introduction to Chip and System Design*; Kluwer: Boston, USA, 1992; pp. 1-359.
2. Auletta, R.; Reese, B.; Traver, C.A comparison of synchronous and asynchronous FSMD designs. Proceedings of the IEEE International Conference on Computer Design (ICCD'93), Cambridge, MA, USA, October 3-6, 1993; IEEE: Cambridge, USA, 1993; pp. 178-182.
3. Karfa, C.; Sarkar, D.; Mandal, C. Verification of datapath and controller generation phase in high-level synthesis of digital circuits. *IEEE Transactions on CAD* **2010**, 29 (3), pp. 479-492.
4. Hu, J.; Wang, G.; Chen, G.; Wei, X. Equivalence Checking of Scheduling in High-Level Synthesis Using Deep State Sequences. *IEEE Access* **2019**, 7, pp. 183435-183443.
5. Schaumont, P.; Shukla, S.; Verbauwhede, I. Design with race-free hardware semantics. Proceedings of the Design Automation & Test in Europe Conference, Munich, Germany, 6-10 March 2006; IEEE: 2007, Vol. 1, pp. 6-pp.
6. Zhu, J.; Gajski, D.D. A unified formal model of ISA and FSMD. Proceedings of the Seventh International Workshop on Hardware/Software Codesign (CODES'99), Rome, Italy, 3 March 1999; IEEE, 1999; (IEEE Cat. No. 99TH8450), pp. 121-125.
7. Kavvadias, N.; Masselos, K. Automated synthesis of FSMD-based accelerators for hardware compilation. Proceedings of the 23rd International Conference on Application-Specific Systems, Architectures and Processors, Delft, Netherlands, 9-11 July 2012; IEEE, 2012; – pp. 157-160.
8. Banerjee, K.; Sarkar, D.; Mandal, C. Extending the FSMD framework for validating code motions of array-handling programs. *IEEE Transactions on CAD* **2014**. Vol. 33(12), pp. 2015-2019.
9. Hwang, E.; Vahid, F.; Hsu, Y.C. FSMD functional partitioning for low power. Proceedings of the Conference on Design, automation and test in Europe, Munich, Germany, 9-12 March 1999; IEEE, 1999; pp. 7-es.
10. Abdullah, A.C.; Ooi, C.Y.; Ismail, N.B.; Mohammad, N.B. Power-aware through-silicon-via minimization by partitioning finite state machine with datapath. Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC, Canada, 22-25 May 2016; IEEE, 2016; pp. 1942-1945.
11. Babakov, R.; Barkalov, A.; Titarenko, L. Research of efficiency of microprogram final-state machine with datapath of transitions. Proceedings of the 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), Lviv, Ukraine, 21-25 Feb. 2017; IEEE, 2017; pp. 203-206.
12. Clare, C.R. *Designing logic systems using state machines*, McGraw-Hill Book Company: New York, USA, 1973; pp. 1-121.
13. Green, D.H.; Chughtai, M.A. Use of multiplexers in direct synthesis of ASM-based designs, *IEE Proceedings E-Computers and Digital Techniques* **1986**. Vol. 133(4), pp. 194-200.
14. Baranov, S. Minimization of algorithmic state machines. Proceedings of the 24th EUROMICRO Conference (Cat. No. 98EX204), Vasteras, Sweden, 27-27 Aug. 1998; IEEE, 1998; Vol. 1, pp. 176-179.
15. Jenihhin, M.; Baranov, S.; Raik, J.; Tihhomirov, V. PSL assertion checkers synthesis with ASM based HLS tool ABELITE. Proceedings of the 13th Latin American Test Workshop (LATW), Quito, Ecuador, 10-13 April 2012; IEEE, 2012; pp. 1-6.
16. Ciletti, M.D. *Advanced digital design with the Verilog HDL*. Prentice Hall of India: New Delhi, 2005, pp. 1-1012.
17. Martin, P.; Bueno, E.; Rodriguez, F.J.; Saez, V. A methodology for optimizing the FPGA implementation of industrial control systems. Proceedings of the 35th Annual Conference of IEEE Industrial Electronics, Porto, Portugal, 3-5 Nov. 2009; IEEE, 2009; pp. 2811-2816.

18. Saha, A.; Ghosh, A.; Kumar, K.G. FPGA Implementation of Arcsine Function Using CORDIC Algorithm. *AMSE JOURNALS-AMSE IIETA publication-2017-Series: Advances A* **2017**, Vol. 54(2), pp 197-202.
19. Burciu, P. An Efficient (Low Resources) Modular Hardware Implementation of the AES Algorithm. *Journal of Electrical Engineering, Electronics, Control and Computer Science* **2019**, Vol. 5(3), pp. 1-10.
20. Sowmya, K.B.; Shreyans, G.; Vishnusai, R.T. Design of UART Module using ASMD Technique. Proceedings of the Fifth International Conference on Communication and Electronics Systems (ICCES 2020), Coimbatore, India, 10-12 June 2020; IEEE, 2020; pp. 176-181.
21. Salauyou, V.V. *Verilog language in embedded systems design on FPGA*, Hotline – Telecom: Moscow, Russia, 2020; pp. 1-440. (in Russian)