DOI: 10.24427/978-83-66391-96-3_5

Rozdział 5
Metodyka ASMD-FSMD projektowania układów cyfrowych na FPGA z wykorzystaniem języka Verilog

Valery Salauyou
Wydział Informatyki, Politechnika Białostocka

Streszczenie: W ostatnim czasie obserwuje się wzrost złożoności projektów układów cyfrowych przy jednoczesnym zaostrzeniu wymagań dotyczących czasu opracowywania i zwiększenia niezawodności projektów. Jednym z kierunków rozwiązania tego problemu jest wprowadzanie nowych metod projektowania układów cyfrowych.

W artykule omówiono metodykę projektowania układów cyfrowych oparta na *automatcie skończonym ze ścieżką przetwarzania danych* (finite state machine with datapath – FSMD), w którym działanie układu opisano w postaci diagramu blokowego automatu ze ścieżką przetwarzania danych (algorithmic state machine with datapath – ASMD). Metodyka ta jest nazywana *metodyką ASMD-FSMD*.

Porównanie różnych metod projektowania układów cyfrowych zostanie przedstawione na przykładzie projektowania mnożnika synchronicznego na bazie *programowalnych układów logicznych* (FPGA – field programmable gate array).

Wykazano, że metodyka ASMD-FSMD, w porównaniu z tradycyjną, pozwala na obniżenie kosztów realizacji z 28,6% do 39,7% oraz zwiększenie wydajności poszczególnych projektów do 17,6%. Ponadto, zastosowanie metodyki ASMD-FSMD może znacznie skrócić czas projektowania i zwiększyć niezawodność projektów. W podsumowaniu przedstawiono zalecenia dotyczącymi stosowania metodyki ASMD-FSMD.

Słowa kluczowe: układ wykonawczy, układ sterujący, automat skończony, automat skończony ze ścieżką przetwarzania danych, diagram blokowy automatu, diagram blokowy automatu ze ścieżką przetwarzania danych, projektowanie układów cyfrowych, programowalne układy logiczne, język Verilog.

Wprowadzenie

Urządzenie cyfrowe projektowane w sposób tradycyjny jest zwykle przedstawiane w postaci *układu wykonawczego* (datapath) i *układu sterującego* (controller lub control unit), które są zwykle projektowane oddzielnie: układ wykonawczy (operacyjny)

ma postać zbioru standardowych jednostek funkcjonalnych (rejestrów, magistrali, multiplekserów itp.), a układ sterujący ma postać *automatu skończonego* (finite state machine – FSM).

W [1] zaproponowano połączenie układów wykonawczych i sterujących oraz przedstawienie ich jako *automatu skończonego ze ścieżką przetwarzania danych* (FSMD). Model FSMD szybko stał się popularny i w [2] zastosowano FSMD do projektów układów synchronicznych i asynchronicznych. Model FSMD okazał się bardzo wygodny do sprawdzania równoważności dwóch układów uzyskanych w wyniku syntezy lub różnych przekształceń projektowych [3,4]. W [5] zaproponowano przedstawienie systemu cyfrowego w postaci sieci FSMD, co prowadzi do realizacji sprzętu pozbawionego zjawiska wyścigów.

Ogólny model FSMD nie jest zawsze wygodny podczas projektowania układów do pewnych określonych zastosowań. Dlatego w szeregu prac proponuje się rozszerzenia FSMD: w [6] – reprezentujące architekturę procesora i ASIC (application-specific integrated circuit); w [7] – dla układu synchronicznego dostępu do pamięci; w [8] – dla programów do przetwarzania tablic.

Dekompozycja FSMD mająca na celu zmniejszenie poboru mocy jest rozważana w [9], a dekompozycja z wykorzystaniem bramkowania w [10].

Porównanie skuteczności FSM i FSMD omówiono w [11]. Wykazano tam, że model FSMD automatu typu Moore'a jest bardziej efektywny pod względem kosztów realizacji w porównaniu ze strukturą kanoniczną FSM.

Ze względu na swoją przejrzystość, *diagramy blokowe maszyn stanów* (algorithmic state machine – ASM) są szeroko stosowane do reprezentacji algorytmu działania FSM. ASM zostały po raz pierwszy zaproponowane w [12] jako alternatywa dla grafów automatów skończonych. W [13] rozważono implementację ASM na PROM, FPLA i multiplekserach. W [14] zaproponowano metody minimalizacji liczby wierzchołków ASM. Praca [15] opisuje narzędzie ABELITE do syntezy sterowników opartych na ASM.

Tradycyjnie, ASM reprezentuje algorytm działania układu sterującego, tj. FSM. W [16] proponuje się wykorzystanie ASM zarówno do opisu działania układu sterującego, jak i do opisu operacji rejestrowych wykonywanych w układzie wykonawczym. W tym celu, do reprezentacji stanów układu sterującego służy ASM automatu Moore'a, na którego przejściach wstawiane są owale (elementy ASM automatu Mealy'ego). W owalach zapisywane są operacje wykonywane w układzie wykonawczym w danym przejściu. Taki ASM jest nazywany *diagram blokowy automatu ze ścieżką przetwarzania danych* (algorithmic state machine with datapath – ASMD).

Diagramy ASMD są ostatnio coraz częściej wykorzystywane w projektach FPGA: przy realizacji przemysłowych systemów sterowania [17]; do implementacji funkcji asin za pomocą algorytmu CORDIC [18]; do sprzętowej implementacji algorytmu kryptograficznego AES [19]; przy projektowaniu uniwersalnego asynchronicznego odbiornika-nadajnika UART [20] i innych.

W tym artykule zaproponowano metodykę zwaną ASMD-FSMD do projektowania układów cyfrowych na układach FPGA z wykorzystaniem języka Verilog. Opisowi metodyki zostanie zilustrowany przykładem projektowania mnożnika synchronicznego.

Artykuł jest zorganizowany w następujący sposób. W rozdziale 1 opisano zaimplementowany algorytm mnożenia. W rozdziale 2 omawiono tradycyjne podejście do projektowania mnożnika w postaci układów wykonawczego i sterującego. W rozdziale 3 opisano ASM. W rozdziale 4 zdefiniowano ASMD i FSMD. W rozdziale 5 omawiono możliwość konwersji ASMD w celu poprawy wydajności projektu. W rozdziale 6 przedstawiono opis FSMD w języku Verilog. W rozdziale 7 zawierto formalną reprezentację metodyki ASMD-FSMD jako algorytmu. W rozdziale 8 opisano badania eksperymentalne porównujące podejście tradycyjne i metodyki ASMD-FSMD. Wyniki badań eksperymentalnych omówiono w rozdziale 9. Podsumowanie zawiera zalecenia dotyczące efektywnego stosowania metodyki ASMD-FSMD.

5.1. Algorytm mnożenia

Rozważmy najprostszy szkolny algorytm mnożenia, który wykonuje operację mnożenia arytmetycznego $P = A \cdot B$ dwóch liczb binarnych bez znaku, gdzie A i B są mnożnikami (czynnikami), a P jest iloczynem (produktem). Niech szerokość słów binarnych A i B będzie taka sama i równa N bitów, wtedy iloczyn P będzie miał szerokość $2N$ bitów. Początkowo P jest wyzerowane. W każdym cyklu mnożenia sprawdzana jest wartość najmniej znaczącego bitu mnożnika $B[0]$. Jeśli $B[0] = 1$, to mnożnik A jest dodawany do iloczynu P. Jeśli $B[0] = 0$, to do iloczynu P jest dodawane zero lub nic nie jest wykonywane. Następnie zawartość czynnika B zostaje przesunięta o jeden bit w prawo, a wartość iloczynu P – w lewo. Algorytm kończy się po uwzględnieniu wszystkich bitów czynnika B.

Rozważany algorytm mnożenia w języku Verilog można opisać w następujący sposób.

```
module algorithm_mult #(parameter N = 4)
        (input [N-1:0] a,b,                    // słowa wejściowe
        output [2*N-1:0] p);                   // słowo wyjściowe
        reg [N-1:0] rb;                        // zmienne pośrednie
        reg [2*N-1:0] ra,rp;
        integer i;                             // iterator
        always @(*)                            // blok proceduralny
        begin
                ra = a;   rb = b;   rp = 0;    // inicjalizacja zmiennych
                for (i = 0; i<N; i = i + 1)    // cykl mnożenia
                begin
```

```
                    // dodawanie wyników cząstkowych
                    if (rb[0])          rp = rp + ra;
                    else                rp = rp + {2*N{1'b0}};
                    rb = rb >> 1;               // przesunięcie zawartości zmiennych
                    ra = ra << 1;
                end
        end
        assign p = rp;                          // formowanie wyniku
endmodule
```

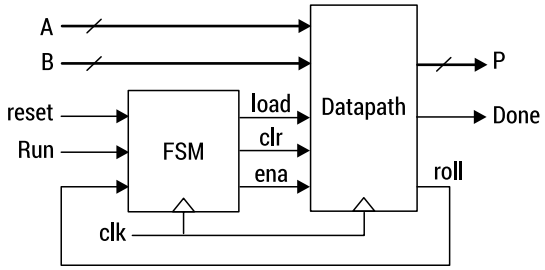
W rzeczywistości powyższy kod opisuje układ kombinacyjny, a nie mnożnik synchroniczny. Aby uzyskać mnożnik synchroniczny, na szynach wejściowych a i b, a także na szynie wyjściowej p, należy ustawić rejestry taktowane sygnałem zegarowym clk. Operacja mnożenia dla takiego układu zawsze będzie wykonywana w jednym cyklu zegara.

Projekt *algorytm_mult* ma dość wysoką wydajność. Jednak koszt realizacji projektu *algorytm_mult* gwałtownie rośnie wraz ze wzrostem liczby bitów *N* słów wejściowych *A* i *B*. Dlatego algorytmiczny opis układów cyfrowych jest rzadko stosowany do implementacji sprzętowej. W tym artykule jest rozważany mnożnik synchroniczny, w którym każdy cykl mnożenia jest wykonywany w jednym lub większej liczbie cykli zegara.

5.2. Tradycyjne podejście przy implementacji mnożnika synchronicznego

W przypadku tradycyjnego podejścia, sprzętową implementacją mnożnika synchronicznego jest obwód synchroniczny, do którego wejścia trafiają wartości mnożonych słów: mnożnika *A* i mnożnika *B*, a na wyjściu formuje się wartość iloczynu *P*. Obwód jest sterowany sygnałem zegarowym clk i sygnałem resetowania reset. Po ustawieniu wartości mnożonych słów na wejściach A i B rozpoczyna się proces mnożenia przez ustawienie sygnału Run na wartość „1". Koniec procesu mnożenia jest sygnalizowany przez wartość „1" na wyjściu Done, po czym wartość iloczynu można odczytać z wyjścia P.

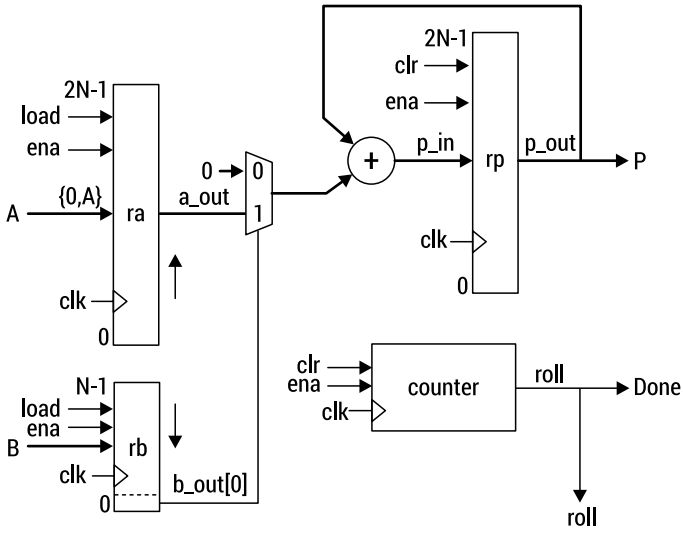
Na rysunku 5.1 przedstawiono schemat blokowy szeregowego mnożnika w postaci *układu wykonawczego* (datapath) i *układu sterującego* (controler), który jest realizowany w postaci automatu skończonego FSM. Wartości mnożonych słów A i B są podawane na wejście układu sterującego. Produkt P i sygnał Done są tworzone na wyjściach układu wykonawczego. Ponadto układ wykonawczy generuje sygnał roll, który jest wyjściem *licznika modulo* i sygnalizuje koniec procesu mnożenia.



RYSUNEK 5.1. Schemat blokowy mnożnika synchronicznego w postaci układu wykonawczego i sterującego

Wejścia FSM to zewnętrzne sygnały reset i Run oraz wewnętrzny sygnał roll. Automat skończony generuje następujące sygnały sterujące: clr – resetowanie rejestrów układu wykonawczego; load – ładowanie wartości mnożonych słów A i B do rejestrów układu wykonawczego; ena – w celu umożliwienia przesunięcia zawartości rejestrów przesuwających. Układ wykonawczy i układ sterujący są synchronizowane tym samym zegarem clk.

Schemat układu realizującego algorytm mnożenia przedstawiono na rysunku 5.2.

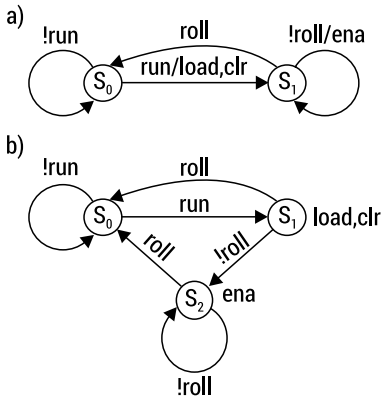


RYSUNEK 5.2. Schemat układu realizującego algorytm mnożenia

Układ wykonawczy zawiera 2*N*-bitowy rejestr przesuwający ra (w lewo) do przechowywania mnożnika *A*, *N*-bitowy rejestr przesuwający rb (w prawo) do przechowywania mnożnika *B*, 2*N*-bitowy multiplekser (2-1) magistrali, 2*N*-bitowy sumator i licznik modulo, który generuje sygnał roll, sygnalizujący koniec procesu mnożenia. Sygnał roll jest taki sam, jak zewnętrzny sygnał Done.

Należy zwrócić uwagę, że w układzie na rysunku 5.2, sygnał b_out[0], który steruje wyborem wejść multipleksera, jest generowany nie przez układ sterujący, ale jest wartością bitu zerowego rejestru rb. W takim przypadku układ wykonawczy działa jako urządzenie sterujące. Zasadniczo nie ma ścisłej granicy między funkcjami układu wykonawczego i układu sterującego. Niektóre funkcje układu sterującego mogą być realizowane przez układ wykonawczy (jak w pokazanym przykładzie) i odwrotnie.

Działanie synchronicznego układu sterującego mnożnikiem można przedstawić jako automat skończony. W tym przypadku można użyć zarówno automatu Mealy'ego (rysunek 5.3, a), jak i automatu typu Moore'a (rysunek 5.3, b).



RYSUNEK 5.3. **Reprezentacja synchronicznego układu sterującego mnożnikiem w postaci grafu automatu skończonego: a – typu Mealy'ego, b – typu Moore'a**

Można zauważyć, że automat Mealy'ego zawiera dwa stany: S_0 i S_1, podczas gdy automat typu Moore'a zawiera trzy stany: S_0, S_1 i S_2.

Aby zaimplementować układ mnożący na FPGA, konieczne jest przedstawienie wszystkich części układu w jednym z języków opisu sprzętu. W omawianym przypadku używany jest do tego język Verilog. Szczegóły opisu elementów składowych układu wykonawczego (rysunek 5.2) oraz automatów skończonych Mealy'ego i Moore'a (rysunek 5.3) do realizacji mnożnika synchronicznego w przypadku podejścia tradycyjnego, podano w [21].

5.3. Diagramy blokowe automatów ASM

Diagram blokowy automatu (algorithmic state machine – ASM) służy do wizualnego opisu algorytmu działania automatu skończonego i jest skierowanym grafem sprzężonym [12] zawierającym trzy typy wierzchołków:

- prostokąty – *wierzchołki stanów* (state box);
- romby – *wierzchołki warunkowe* (decision box);
- owale – *wierzchołki wyjść warunkowych* (conditional output box).

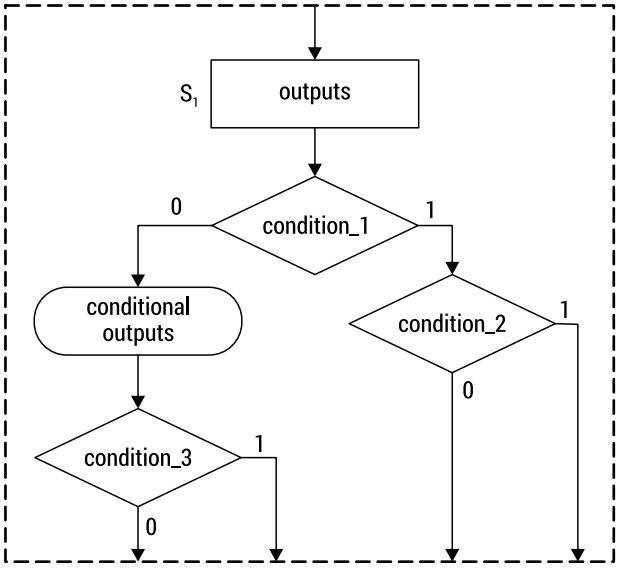
Wierzchołek stanu ASM (prostokąt) określa stan automatu. W górnej części wierzchołka można zapisać nazwę stanu (na przykład S0, START, INITIAL itp.), a także kod binarny stanu. W przypadku automatu Moore'a sygnały wyjściowe zapisywane są wewnątrz wierzchołka stanu, który w tym stanie przyjmuje wartość „1". Domyślnie przyjmuje się, że wszystkie inne wyjścia w tym stanie mają wartość równą zero. Należy zauważyć, że ASM nie pozwala na opisywanie automatów o skończonych stanach z wyjściami nieokreślonymi.

Warunki do sprawdzenia zapisane są w wierzchołkach warunkowych ASM (rombach), czyli wierzchołek warunkowy ASM jest punktem rozgałęzienia algorytmu. Wyjścia wierzchołka warunkowego są oznaczone wartościami 0 i 1, które odpowiadają przejściom w przypadku zerowej (fałszywej) lub jedynkowej (prawdziwej) wartości wyniku sprawdzenia warunku. Warunkiem może być zmienna wejściowa automatu skończonego, wyrażenie logiczne, pojedynczy bit wektora bitowego itp.

W wierzchołkach wyjść warunkowych (owali) zapisywane są sygnały wyjściowe automatu Mealy'ego, które przyjmują wartość jeden przy pewnym przejściu. Sygnały wyjściowe zapisane w owalach nazywane są *wyjściami warunkowymi* (conditional output).

W ASM dla automatów Moore'a nie ma wierzchołków wyjść warunkowych (owale), a w ASM dla automatów Mealy'ego nie jest nic zapisane w wierzchołkach stanów.

Głównym blokiem konstrukcyjnym diagramu blokowego ASM jest *blok ASM* (ASM block), którego przykład pokazano na rysunku 5.4.



RYSUNEK 5.4. **Blok ASM**

Blok ASM zawiera tylko jeden wierzchołek stanu (prostokąt) i może mieć kilka wierzchołków warunkowych (rombów) i wierzchołków wyjść warunkowych (owale), a romby mogą poprzedzać owale lub następować po nich. Wejścia i wyjścia wierzchołków są połączone za pomocą krawędzi. Blok ASM ma tylko jedno wejście, które jest wejściem do początku stanu i może mieć jedno lub więcej wyjść. Blok ASM opisuje zachowanie automatu w jednym stanie podczas jednego cyklu synchronizacji.

Diagram blokowy ASM (zwany dalej po prostu *diagramem ASM*) jest kompozycją połączonych ze sobą bloków ASM. W takim przypadku konieczne jest przestrzeganie następujących zasad konstruowania ASM: każde wyjście dowolnego wierzchołka ASM można podłączyć tylko do jednego wejścia innego wierzchołka, tj. rozgałęzienie algorytmu jest możliwe tylko w wierzchołkach warunkowych; sprzężenia zwrotne są zabronione wewnątrz bloku ASM.

Należy zwrócić uwagę na niektóre charakterystyczne cechy ASM. Za pomocą bloków ASM w ASM jawnie wyznaczają się wewnętrzne stany automatu skończonego. Przejście z jednego bloku ASM do drugiego odbywa się zawsze w jednym cyklu zegarowym, tj. algorytm działania automatu jest bezpośrednio powiązany z czasem automatu. Dzięki temu, zawsze można prześledzić, ile cykli zegara zajmie wykonanie danego fragmentu algorytmu. Jednocześnie ASM zachowuje przejrzystość diagramów blokowych. Ponadto ASM wstępnie określa typ automatu: Mealy'ego lub Moore'a.

Przedstawmy sposób konstruowania ASM automatu skończonego w postaci następującego algorytmu.

Algorytm 1. Metodologia konstruowania ASM.

1. Określa się stany automatu skończonego.
2. Dla każdego stanu budowany jest blok ASM. W tym przypadku wszystkie przejścia od tego stanu są określane dla wszystkich możliwych wartości zmiennych wejściowych. Jeśli jakaś zmienna wejściowa nie wpływa na przejścia z danego stanu, to nie występuje wewnątrz bloku ASM.
3. W przypadku automatu Moore'a zmienne wyjściowe są zapisywane w wierzchołkach stanów (prostokątach), które w tym stanie przyjmują wartość jeden.
4. W przypadku automatu Mealy'ego zmienne wyjściowe są zapisywane w wierzchołkach wyjść warunkowych (owale), które przyjmują wartości „1" w danym przejściu.
5. Bloki ASM są ze sobą połączone zgodnie z algorytmem pracy układu. W takim przypadku każde wyjście bloku ASM można podłączyć tylko do jednego wejścia tego samego lub innego bloku ASM.
6. Koniec.

Można zauważyć, że konstrukcja ASM zaczyna się od określenia stanów automatu skończonego, tj. od samego początku projektant jednoznacznie określa stany automatu. Następnie określane są przejścia z każdego stanu, tj. definiuje się przejścia między stanami. Następnie określane są wartości zmiennych wyjściowych: osobno dla automatu Mealy'ego i osobno dla automatu Moore'a. Ostatni etap, czyli połączenie ze sobą bloków ASM, ma raczej charakter formalny, co pozwala jeszcze raz sprawdzić poprawność konstrukcji ASM.

5.4. Diagramy blokowe ASMD i automaty skończone ze ścieżką przetwarzania danych FSMD

Diagram blokowy ASM ze ścieżką przetwarzania danych (ASM with datapath – ASMD) to ASM, w którym, w prostokątach i owalach, można zapisywać dowolne operacje na rejestrach, które są dozwolone w języku Verilog, a wszelkie wyrażenia logiczne języka Verilog można sprawdzać w wierzchołkach warunkowych. Diagram ASMD, podobnie jak ASM, składa się z bloków. Działania opisane w bloku ASMD wykonywane są w jednym cyklu zegarowym.

Automat skończony realizowany zgodnie z ASMD będzie nazywany *automatem skończonym ze ścieżką przetwarzania danych* (FSM with datapath – FSMD), a metodyka projektowania FSMD według ASMD nazywana jest *metodyką ASMD-FSMD*.

Należy zauważyć, że w naszym przypadku, w przeciwieństwie do [1], FSMD nie jest podzielony na *układ sterujący* (FSM) i *układ wykonawczy* (datapath), ale raczej przypomina behawioralny (algorytmiczny) opis działania urządzenia. Jednak w przeciwieństwie do opisu algorytmicznego (projekt algorithm_mult), ASMD jawnie definiuje stany FSMD.

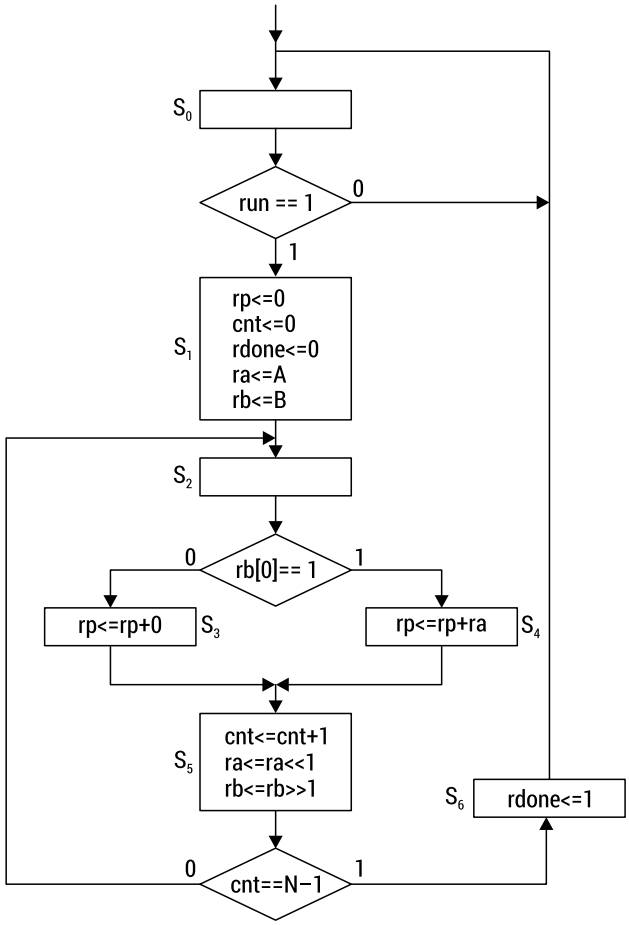
Przedstawione tu podejście do konstruowania ASMD różni się również od podejścia, które jest zdefiniowane w [16]: pokazany tu ASMD może opisywać dowolny typ automatu (Mealy'ego lub Moore'a), a operacje wykonywane w układzie wykonawczym mogą być zapisywane zarówno w owalach, jak i w prostokątach ASMD.

Na rysunku 5.5 pokazano ASMD, który odpowiada FSMD Moore'a w celu zaimplementowania rozważanego algorytmu mnożenia.

Stan S_0 jest stanem oczekiwania na sygnał uruchomienia run, który rozpoczyna proces mnożenia. W stanie S_1, rejestr iloczynu rp, licznik cnt i przerzutnik przechowujący wartość sygnału Done są zerowane. Dodatkowo w stanie S_1 ładowane są początkowe wartości rejestrów ra i rb.

Następny stan S_2 jest punktem wejścia w cykl procesu mnożenia. Jeden cykl mnożenia odpowiada utworzeniu iloczynu cząstkowego, dodaniu go do wyniku i wykonaniu niezbędnych przesunięć zawartości rejestrów, a także zwiększeniu wartości licznika. W stanie S_2, w zależności od wartości rb[0], konieczne jest dodanie albo wartości zerowej, albo wartości rejestru ra do rejestru rp. Ponieważ jednak ASMD opisuje automat typu Moore'a, czynności te mogą być wykonywane tylko w oddzielnych stanach S_3 i S_4.

Po stanie S_3 lub S_4 następuje stan S_5, w którym licznik cnt jest zwiększany, a rejestry ra i rb są przesuwane. Dodatkowo w stanie S_5 sprawdzana jest wartość licznika cnt. Jeśli wartość cnt jest równa *N*-1, następuje przejście do stanu S_6, w przeciwnym razie do stanu S_2 w celu wykonania nowej iteracji procesu mnożenia. W stanie S_6, przerzutnik rdone jest ustawiany na jeden i następuje przejście do stanu początkowego S_0.



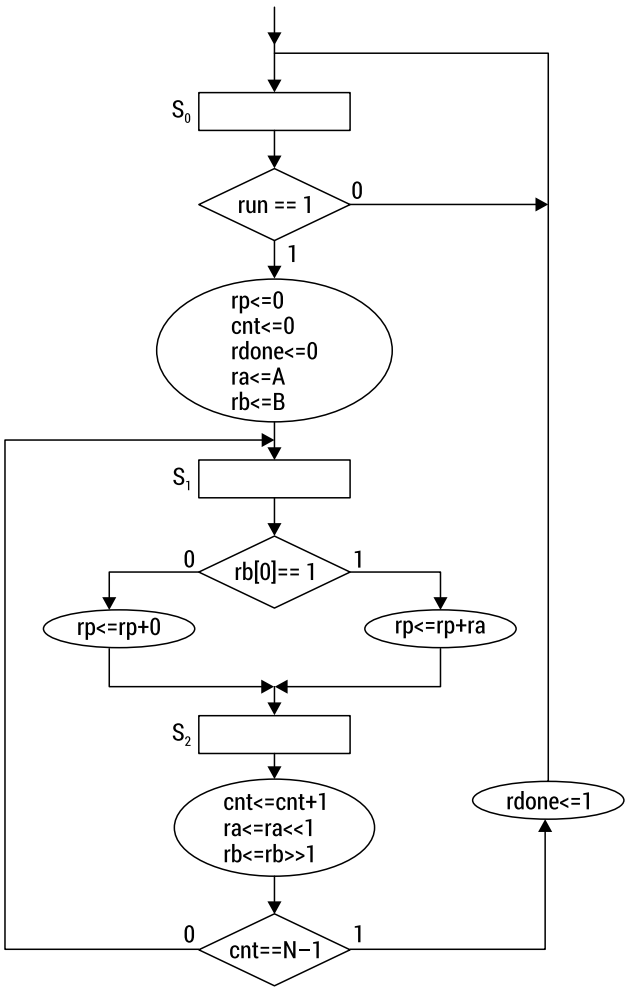
RYSUNEK 5.5. **Diagram ASMD opisujący algorytm mnożenia a w postaci FSMD Moore'a**

Bezpośrednio z ASMD pokazanego na rysunku 5.5, można określić liczbę taktów zegara n_{Moore} wymaganych do pomnożenia *N*-bitowych liczb binarnych. Stan początkowy S_0 dla automatów typu Moore'a zawsze wymaga jednego taktu zegara. W stanie S_1 rejestry są inicjalizowane i ładowane, co wymaga jeszcze jednego cyklu taktowania. Wykonanie jednej iteracji mnożenia (pętli) związanej z przetwarzaniem jednego bitu mnożnika B wymaga trzech cykli zegarowych (stany S_2, S_3 lub S_4 i S_5). Utworzenie wyjścia rdone wymaga również jednego cyklu zegara (stan S_6). Zatem FSMD dla automatu Moore'a wykonuje mnożenie *N*-bitowych liczb binarnych w n_{Moore} cyklach zegara, gdzie:

$$n_{Moore} = 3N + 3 \tag{5.1}$$

Poprawność zależności (1) potwierdzają również wyniki symulacji czasowej.

Na rysunku 5.6 przedstawiono ASMD, który odpowiada FSMD typu Mealy'ego służącego do implementacji rozważanego algorytmu mnożenia.



RYSUNEK 5.6. **Diagram ASMD służący do realizacji algorytmu mnożenia a przy użyciu automatu FSMD typu Mealy'ego**

W stanie początkowym S_0 automat oczekuje na nadejście wartości sygnału run równej „1". Po ustawieniu sygnału run na „1", wszystkie rejestry mnożnika są inicjalizowane zgodnie z wymaganiami.

Następny stan S_1 jest punktem wejścia w cykl procesu mnożenia. W stanie S_1 sprawdzana jest wartość bitu rb[0] i zależnie od jego wartości, do zawartości rejestru rp dodawana jest albo wartość zerowa, albo zawartość rejestru ra. Można zauważyć, że dwa oddzielne stany nie są tutaj wymagane, jak w FSMD Moore'a (S_3 i S_4 na rysunku 5.5).

W stanie S_2 wykonywane są niezbędne przesunięcia rejestrów ra i rb oraz inkrementacja licznika. Następnie w stanie S2 sprawdzana jest poprzednia wartość licznika, jeśli jest równa *N*-1, to kończy się algorytm mnożenia i automat przechodzi do stanu S_0 z ustawieniem przyrzutnika rdone. W przeciwnym razie cykl procesu mnożenia jest powtarzany.

Zgodnie z rysunkiem 5.6 można zauważyć, że jeden cykl mnożenia jest wykonywany w dwóch cyklach zegara, dlatego FSMD dla automatu Mealy'ego wykonuje mnożenie *N*-bitowych liczb binarnych w n_{Mealy} taktach zegara, gdzie:

$$n_{Mealy} = 2N + 1 \tag{5.2}$$

Porównanie FSMD Moore'a i FSMD Mealy'ego z rozważanego przykładu pokazuje, że FSMD Mealy'ego ma znacznie mniej stanów (odpowiednio 3 i 7), a także działa znacznie szybciej, ponieważ każdy cykl procesu mnożenia jest wykonywany w dwóch taktach zegara (w FSMD Moore'a w trzech taktach zegara).

5.5. Zwiększenie prędkości mnożnika synchronicznego

Fakt, że jeden cykl procesu mnożenia jest wykonywany w kilku cyklach zegarowych, sprawia, że praktyczne zastosowanie omówionych powyżej konstrukcji mnożników jest niezwykle nieefektywne, ponieważ znacznie wydłuża czas mnożenia.

Pytanie brzmi: jak zaprojektować FSMD, który wykonuje każdy cykl mnożenia w jednym takcie zegara? Aby to zrobić, konieczne jest zbudowanie diagramu ASMD, w którym jeden blok ASMD odpowiada cyklowi mnożenia. Diagram ASMD dla automatu Mealy'ego spełniającego postawione wymagania pokazano na rysunku 5.7.

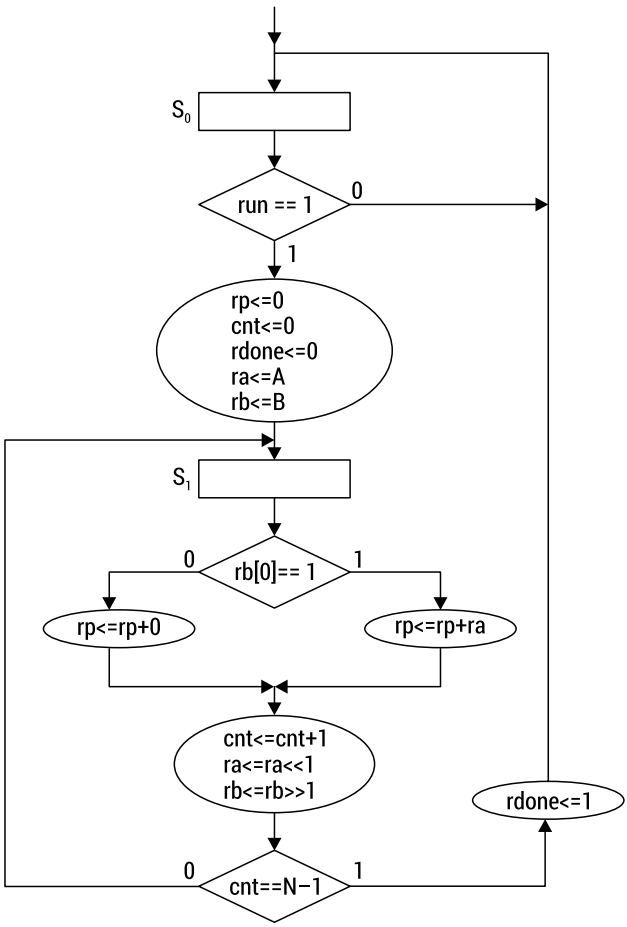
Cechą szczególną diagramu ASMD z rysunkiem 5.7 jest to, że istnieją tu tylko dwa bloki ASMD: w stanie S_0 wykonywana jest inicjalizacja rejestrów, a stan S_1 odpowiada pełnemu cyklowi mnożenia.

W ASMD z rysunku 5.7 stany S_1 i S_2 ASMD z rysunku 5.6 zostały połączone w jeden stan. Stało się to możliwe dzięki temu, że operacje wykonywane w stanach S_1 i S_2 zmieniają zawartość różnych rejestrów (w stanie S_1 zmienia się zawartość rejestru rp, a w stanie S_2 zawartość rejestrów ra i rb).

Należy zauważyć, że wszystkie metody optymalizacji syntezy behawioralnej mogą być zastosowane w ASMD [22], np. *eliminacja martwego kodu* (dead code elimination), *propagacja stałych* (constant propagation), *ruch kodu* (code motion), *ekspansja w linii* (in-line expansion), *rozwijanie pętli* (loop unrolling) i *redukcja wysokości drzewa* (tree height reduction). Ponadto do ASMD można zastosować inne techniki optymalizacji specyficzne dla ASMD, na przykład w celu zmniejszenia liczby stanów w pętli.

Na rysunku 5.7 można zauważyć, że jeden cykl mnożenia jest wykonywany w jednym cyklu zegara, dlatego mnożenie *N*-bitowych liczb binarnych zgodnie z ASMD z rysunku 5.7 odbywa się w n_{Mealy_1} cyklach zegarowych, gdzie:

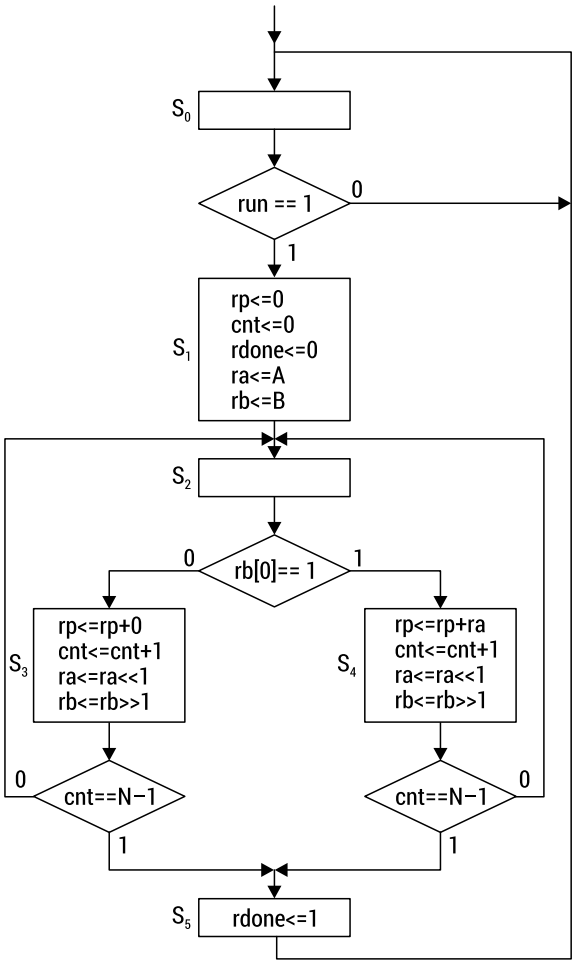
$$n_{Mealy\_1} = N + 1 \tag{5.3}$$



RYSUNEK 5.7. **Diagram ASMD automatu Mealy'ego, który zapewnia największą szybkość mnożnika synchronicznego**

Aby zwiększyć wydajność FSMD typu Moore'a, należy przekształcić ASMD z rysunku 5.5 w celu zmniejszenia liczby stanów w cyklu. Przekształcony diagram ASMD pokazano na rysunku 5.8.

W tym przypadku operacje zwiększania licznika i przesuwania rejestrów są wykonywane w stanach S_3 i S_4. Pozwala to na usunięcie stanu S_5 z ASMD z rysunek 5.5. Ta metoda transformacji ASMD odnosi się do metod *redukcji wysokości drzew* (tree height reduction) [22].



RYSUNEK 5.8. **Diagram ASMD automatu Moore'a, zapewniający wzrost szybkości działania mnożnika synchronicznego**

Niestety, ASMD Moore'a z rysunku 5.8 nie opisuje cyklu mnożenia za pomocą jednego stanu, jak ASMD Mealy'ego. Stan S_2 jest wymagany do wejścia w cyklu obliczania iloczynu cząstkowego. Następnie sprawdzany jest najmniej znaczący bit mnożnika (rb[0]). Wykonywanie działań zależnych od wartości rb[0] wymaga również co najmniej jednego dodatkowego stanu. Tak więc cykl operacji mnożenia przechodzi przez dwa stany FSMD Moore'a, czyli jest wykonywany w dwóch cyklach zegarowych. W rezultacie pomnożenie *N*-bitowych liczb binarnych za pomocą ASMD z rysunku 5.8 odbywa się w n_{Moore_1} cyklach zegara, gdzie:

$$n_{Moore\_1} = 2N + 3 \tag{5.4}$$

5.6. Opis FSMD w języku Verilog

Aby zaimplementować synchroniczny mnożnik na FPGA w postaci FSM Mealy'ego, należy wybrać diagram ASMD z rysunku 5.7. Poniżej znajduje się kod projektu FSMD Mealy w języku Verilog.

```
module mult_FSMD_Mealy #(parameter N = 4)
        (input clk, reset, run,                 // sygnały sterujące
        input [N-1:0] a,b,                      // słowa wejściowe
        output [2*N-1:0] p,                     // wynik
        output done);                           // sygnał zakończenia mnożenia

        reg [2*N-1:0] ra,rp;                    // deklaracja rejestrów
        reg [N-1:0] rb;
        reg rdone;
        reg [N_cnt-1:0] cnt;                    // licznik

        localparam N_cnt=clogb2(N-1); // określenie rozmiaru licznika

        function integer clogb2(input [N-1:0] v); // funkcja stała
                for (clogb2 = 0; v > 0; clogb2 = clogb2 + 1)
                        v = v >> 1;
        endfunction

        localparam [0:0] s0 = 0, s1 = 1;        // dklaracja stanów
        reg [0:0] state;                        // zmienna stanu

        always @(posedge clk)           // rozpoczęcie procesu mnożenia
                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;                  // określenie wartości wyjściowych
        assign done = rdone;
endmodule
```

Moduł mult_FSMD_Mealy jest opisany za pomocą stylu opisu automatów skończonych z jednym procesem [21]. W naszym przypadku jest to dopuszczalne, ponieważ na wyjściu rejestrów generowane są iloczyn p i sygnał done. Powyższy kod używa funkcji stałej clogb2 do określenia rozmiaru licznika. Funkcja ta oblicza najmniejszą liczbę całkowitą większą lub równą logarytmowi przy podstawie 2 z liczby v.

Kod projektu FSMD Moore'a w języku Verilog, który odpowiada ASMD z rysunku 5.8 przedstawia się następująco.

```
module mult_FSMD_Moore #(parameter N = 4)
        (input clk, reset, run,                 // sygnały sterujące
        input [N-1:0] a,b,                      // słowa wejściowe
        output [2*N-1:0] p,                     // wynik
        output done);                           // sygnał zakończenia mnożenia

        reg [2*N-1:0] ra,rp;                    // deklaracja rejestrów
        reg [N-1:0] rb;
        reg rdone;
        reg [N_cnt-1:0] cnt;                    // licznik

        localparam N_cnt=clogb2(N-1); // określenie rozmiaru licznika

        function integer clogb2(input [N-1:0] v); // funkcja stała clogb2 –
                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;  // dklaracja stanów
        reg [2:0] state;                                // zmienna stanu

        always @(posedge clk)               // rozpoczęcie procesu mnożenia
                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;                          // określenie wartości wyjściowych
        assign done = rdone;
endmodule
```

Do opisu automatu Moore'a został również użyty styl opisu z jednym procesem [21]. W tym przypadku stan następny jest określany w każdym stanie obecnym, a wszystkie czynności wykonywane na rejestrach opisane są zgodnie z diagramem ASMD z rysunku 5.8.

5.7. Metodologia ASMD-FSMD projektowania układów cyfrowych

W tej części zostanie pokazany formalny opis rozważanej metodyki ASMD-FSMD w postaci następującego algorytmu.

Algorytm 2. Metodyka ASMD-FSMD projektowania układów cyfrowych.

1. Określane są stany FSMD.
2. Dla każdego stanu budowany jest blok ASMD.
   2.1. W wierzchołkach warunkowych ASMD zapisywane są funkcje logiczne, których wartość jest sprawdzana w danym stanie.
   2.2. W przypadku FSMD Moore'a wierzchołki stanów (prostokąty) zapisuje się operacje wykonywane na zawartości rejestrów w danym stanie.
   2.3. W przypadku FSMD Mealy'ego operacje wykonywane na zawartości rejestrów w danym przejściu są zapisywane w wierzchołkach wyjść warunkowych (owale).
3. Bloki ASMD są ze sobą połączone zgodnie z algorytmem pracy układu. W takim przypadku każde wyjście bloku ASMD można podłączyć tylko do jednego wejścia tego samego lub innego bloku ASMD.
4. W razie potrzeby diagram ASMD jest modyfikowany w celu zwiększenia prędkości projektowanego urządzenia. W tym celu analizowane są pętle algorytmu, a ASMD przekształca się w taki sposób, aby zminimalizować liczbę stanów w obrębie pętli.
5. Kod FSMD w Verilog jest budowany bezpośrednio z ASMD. Zmiennym algorytmu w kodzie odpowiadają rejestry. Funkcje logiczne testowane w wierzchołkach warunkowych ASMD odpowiadają wyrażeniom logicznym w instrukcjach if. Akcje wykonywane w blokach ASMD są opisywane jako bloki proceduralne begin… end. Każdy blok dla FSMD Moore'a opisuje operacje wykonywane w danym stanie diagramu ASMD i definiuje stany następne (stany przejść) zgodnie z przejściami diagramu ASMD. W każdym bloku dla FSMD Mealy'ego tworzony jest opis algorytmiczny wszystkich czynności wykonywanych w odpowiednim bloku ASMD.
6. Koniec.

Należy zwrócić uwagę na niektóre cechy metody ASMD-FSMD. Głównym etapem projektowania jest budowa diagramu ASMD w oparciu o algorytm działania (funkcjonowania) układu. Nie ma ścisłego oddzielenia układu wykonawczego od układu sterującego. Jednocześnie ASMD definiuje stany automatu skończonego, które odpowiadają

stanom układu sterującego. Pozwala to powiązać algorytm działania projektowanego urządzenia z taktami zegarowymi. Dlatego, korzystając z ASMD, można prześledzić, ile cykli zegara zajmuje każda gałąź algorytmu. ASMD nie definiuje również jawnie struktury układu wykonawczego.

Różnica między techniką ASMD-FSMD a innymi znanymi metodami polega na tym, że w ASMD można używać zarówno automatu typu Moore'a, jak i automatu typu Mealy'ego (w [16] tylko automatu Moore'a); czynności wykonywane w układzie wykonawczym na zawartości rejestrów mogą być zapisywane zarówno w prostokątach ASMD, jak i w owalach (w [16] tylko w owalach).

Należy również zauważyć, że jeden i ten sam algorytm działania układu można opisać różnymi diagramami ASMD, co wpływa na szybkość i koszt realizacji projektu. Aby zwiększyć wydajność, pętle algorytmu należy opisać jak najmniejszą liczbą stanów w celu zminimalizowania liczby stanów w obrębie pętli. Do tego celu lepiej nadaje się automat typu Mealy'ego.

5.8. Badania eksperymentalne

Na przykładzie prostego mnożnika synchronicznego rozważono trzy techniki projektowania układów cyfrowych: implementację algorytmiczną w języku Verilog, technikę tradycyjną (w postaci złożenia układu wykonawczego i sterującego) oraz technikę wykorzystującą automaty FSMD ze ścieżką przetwarzania danych (metodyka ASMD-FSMD). Aby przetestować skuteczność tych technik, zbadano następujące projekty mnożników synchronicznych:

- algorytm_mult – algorytmiczna implementacja mnożnika w języku Verilog;
- mult_FSM_Mealy – mnożnik zbudowany w postaci układów wykonawczego i sterującego, gdzie rolę układu sterującego pełni automat Mealy'ego;
- mult_FSM_Moore – mnożnik zbudowany w postaci układów wykonawczego i sterującego, gdzie rolę układu sterującego pełni automat Moore'a;
- mult_FSMD_Mealy – mnożnik zbudowany zgodnie z diagramem ASMD automatu Mealy'ego z rysunek 5.7;
- mult_FSMD_Moore – mnożnik zbudowany zgodnie z diagramem ASMD automatu Moore'a z rysunek 5.8.

Wyniki przprowadzonych badan w systemie Quartus Prime w wersji 18.1 z domyślnymi parametrami syntezy układów FPGA z rodziny Cyclone IV E przedstawiono w tabelach 5.1–5.5, gdzie N jest liczbą bitów słów wejściowych mnożnika; L to liczba użytych elementów logicznych FPGA (koszt realizacji); F – maksymalna częstotliwość pracy układu w MHz (szybkość); T – czas trwania jednego taktu zegara w nanosekundach, $T = 1/F$; n – liczba taktów zegara potrzebnych do obliczenia wyniku; t – czas (w nanosekundach) wymagany do obliczenia wyniku, $t = T \cdot n$.

Należy zauważyć, że przy realizacji tych projektów na układach FPGA nie zostały wykorzystane wbudowane bloki DSP i wbudowane bloki mnożnika.

Dla projektu algorytm_mult $n = 1$, dla projektu mult_FSM_Mealy wartość n wynosi $(N + 1)$, dla projektu mult_FSM_Moore wartość n wynosi $(N + 2)$, dla projektu mult_FSMD_Mealy wartość n jest określana zgodnie z (3), a dla projektu mult_FSMD_Moore – zgodnie z (4).

TABELA 5.1. **Wyniki badań projektu algorytm_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

TABELA 5.2. **Wyniki badań projektu 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

TABELA 5.3. **Wyniki badań projektu 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

TABELA 5.4. Wyniki badań projektu 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

TABELA 5.5. Wyniki badań projektu 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

W celu lepszego porównania rozpatrywanych projektów, w tabeli 5.6 i tabeli 5.7 przedstawiono wartości kosztu realizacji L i czasu t wykonania operacji mnożenia w tych projektach dla różnych wartości N.

TABELA 5.6. Koszt L realizacji mnożników (liczba elementów logicznych)

Project	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

TABELA 5.7. Czas t operacji mnożenia (w nanosekundach)

Project	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

5.9. Dyskusja wyników

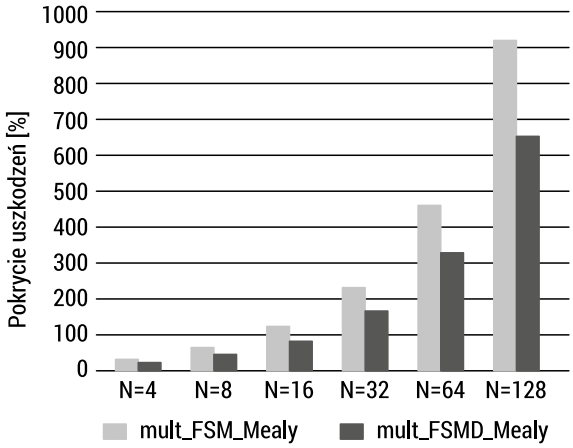
Analizując tabele 5.6 i 5.7 można zauważyć, że przy zastosowaniu tradycyjnej metodologii projektowania układów cyfrowych (projekty mult_FSM_Moore i mult_FSM_Mealy) typ automatu (Mealy lub Moore) nie odgrywa szczególnej roli, ponieważ koszt realizacji i szybkość działania projektów mult_FSM_Moore i mult_FSM_Mealy są w przybliżeniu takie same.

W przypadku zastosowania metodyki ASMD-FSMD projekt mult_FSMD_Mealy ma wyraźną przewagę nad projektem mult_FSMD_Moore, zarówno pod względem kosztów realizacji, jak i wydajności. Przewagę projektu mult_FSMD_Mealy pod względem kosztów realizacji dowodzi fakt, że w cyklu mnożenia ASMD dla automatu Moore'a z rysunku 5.8, wiele operacji jest dublowanych (inkrementacja licznika, przesuwanie rejestrów), a przewaga szybkości wynika z faktu, że jeden cykl mnożenia w ASMD z rysunku 5.8 jest wykonywany w dwóch taktach zegarowych. Dlatego przy stosowaniu metodyki ASMD-FSMD należy preferować automaty typu Mealy'ego.

Porównanie metodyki ASMD-FSMD (projekt mult_FSMD_Mealy) z metodologią tradycyjną (projekty mult_FSM_Moore i mult_FSM_Mealy) pokazuje, że metodyka ASMD-FSMD przewyższa tradycyjną metodologię zarówno pod względem kosztów realizacji (od 28,6% do 39,7%), jak i szybkości działania układów (maksymalnie o 17,6% dla projektu mult_FSM_Moore przy N = 4).

Przewaga metodyki ASMD-FSMD nad metodą tradycyjną pod względem kosztów implementacji, pokazana na rysunku 5.9.

Należy zwrócić szczególną uwagę, że projekt mnożnika mult_FSMD_Mealy został opisany bezpośrednio w języku Verilog zgodnie z ASMD z rysunku 5.7, bez opracowywania układów wykonawczego i sterującego, w porównaniu do projektów mult_FSM_Moore i mult_FSM_Mealy. Z tego powodu metodyka ASMD-FSMD znacznie skraca czas projektowania, w porównaniu z podejściem tradycyjnym, ponieważ nie ma potrzeby projektowania układu wykonawczego i wszystkich jego elementów składowych, a także układu sterującego i modułu nadrzędnego.



RYSUNEK 5.9. Porównanie kosztów realizacji mnożników synchronicznych w przypadku zastosowania tradycyjnej metodyki projektowania (projekt mult_FSM_Mealy) oraz metodyki ASMD-FSMD (projekt mult_FSMD_Mealy)

Należy również zauważyć, że metodyka projektowania ASMD-FSMD poprawia niezawodność projektów. Faktem jest, że wiele etapów tradycyjnego projektowania jest wykonywanych ręcznie przez projektanta i jest źródłem trudnych do znalezienia błędów. W metodyki ASMD-FSMD po zbudowaniu diagramu ASMD, opis projektu w języku Verilog wykonuje się bezpośrednio na bazie diagramu ASMD, bez żadnych opisów pośrednich.

Poniżej podjęto próbę oszacowania czasu projektowania i niezawodności projektów mnożników synchronicznych realizowanych przy użyciu tradycyjnego podejścia i metodyki ASMD-FSMD. Aby to zrobić, zostanie porównana długość kodu (liczba wierszy) projektów mult_FSM_Mealy (podejście tradycyjne) i mult_FSMD_Mealy (metodyka ASMD-FSMD). Projekt mult_FSM_Mealy składa się z modułu najwyższego poziomu (mult_a_Mealy), układu sterującego (fsm_Mealy) i układu wykonawczego (datapath_mult). Z kolei układ wykonawczy zawiera następujące komponenty: sumator (adder), licznik modulo (counter_modulo_M), multiplekser magistrali (mux_2), rejestr przesuwający w lewo (shl_load) i rejestr przesuwający w prawo (shr_load), a także zwykły rejestr (rejestr). Liczbę wierszy dla każdego modułu podano w tabela 5.8.

Analizując tabelę 5.8 widać, że w przypadku zastosowania metodyki ASMD-FSMD, w porównaniu z podejściem tradycyjnym, długość kodu skraca się 144/58 = 2,483 razy, tj. około 2,5 razy. Dlatego możemy założyć, że w rozważanym przykładzie mnożnika synchronicznego, użycie metodyki ASMD-FSMD może skrócić czas projektowania około 2,5 krotnie.

TABELA 5.8. Liczba linii kodu dla elementów projektu mult_FSM_Mealy i projektu mult_FSMD_Mealy

Nazwa modułu	Liczba linii kodu
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_FSM_Mealy)	144
mult_FSMD_Mealy	58

Trudniej jest określić ilościowo wzrost niezawodności projektu w wyniku zastosowania metodyki ASMD-FSMD. Można założyć, że niezawodność projektu jest również proporcjonalna do rozmiaru kodu projektu. Jednak ocena niezawodności powinna uwzględniać liczbę powiązań między modułami oraz szereg innych czynników. Jeżeli niezawodność oceniana jest tylko na podstawie liczby wierszy kodu, to zastosowanie w naszym przypadku metodyki ASMD-FSMD pozwala co najmniej 2,5-krotnie zwiększyć niezawodność projektu.

W metodyki ASMD-FSMD, na konstrukcję diagramu ASMD nakładane są specjalne wymogi. Doświadczenie ze stosowania metodyki ASMD-FSMD pokazało, że do jej efektywnego wykorzystania konieczne jest przestrzeganie pewnych zasad. Przede wszystkim należy dążyć do zbudowania diagramów blokowych ASMD dla automatów typu Mealy'ego. Jeżeli algorytm pracy układu zawiera cykle, to cały cykl w ASMD należy opisać jednym stanem. Jeśli nie jest to możliwe, należy użyć minimalnej liczby stanów.

Algorytmiczna implementacja mnożnika (projekt algorytm_mult) znacznie przewyższa wszystkie rozważane projekty pod względem szybkości (3–5 razy), jednak ma gorszy koszt realizacji, który szybko rośnie wraz ze wzrostem długości słowa mnożnika wejściowego N. Metoda projektowania układów cyfrowych oparta na implementacji algorytmicznej może być zalecana do budowy niewielkich układów, a także do celów symulacyjnych w celu sprawdzenia poprawności algorytmu działania układu.

Podsumowanie

Na przykładzie projektowania mnożnika synchronicznego rozważono trzy metody projektowania układów cyfrowych: implementację algorytmiczną w języku Verilog, tradycyjną (gdy projekt przedstawiany jest jako kompozycja układów wykonawczego i sterującego) oraz opartą na automatach skończonych ze ścieżką przetwarzania danych (metodyka ASMD-FSMD).

Porównanie proponowanej metodyki ASMD-FSMD z podejściem tradycyjnym pokazuje, że zastosowanie nowej metodyki może obniżyć koszt realizacji od 28,6% do 39,7%, a także w niektórych przypadkach zwiększyć szybkość działania projektów o 17,6%.

Jednak główną zaletą metodyki ASMD-FSMD jest to, że pozwala ona na skrócenie czasu projektowania i zwiększenie niezawodności projektów co najmniej 2,5-krotnie.

Metodyka ASMD-FSMD może być wykorzystana przy projektowaniu układów cyfrowych opartej na dowolnej bazie technologicznej (niekoniecznie FPGA), na przykład z wykorzystaniem układów ASIC. Jako język opisu dla automatów skończonych ze ścieżką przetwarzania danych FSMD może być użyty dowolny język opisu sprzętu, na przykład VHDL lub SystemVerilog. Obiecującym kierunkiem rozwoju jest również wykorzystanie metodyki ASMD-FSMD do wysokopoziomowego projektowania złożonych układów, a także opracowanie specjalnych algorytmów do optymalizacji ASMD, na przykład w celu zmniejszenia liczby stanów w cyklach.

Praca została wykonana przy częściowym wsparciu finansowym Politechniki Białostockiej (Polska, grant WZ / WI-IIT / 4/2020).

Bibliografia

[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

[2] Auletta R., Reese B., Traver, C., *A comparison of synchronous and asynchronous FSMD designs*, Proc. of the IEEE International Conference on Computer Design (ICCD'93), Cambridge, MA, USA, October 3–6, 1993, IEEE: Cambridge, USA 1993,178–182

[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), 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, nr 7, s. 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, 1, 6

[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), 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, 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, 33(12), 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, 7

[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, 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, 203–206

[12] Clare C.R., *Designing logic systems using state machines*, McGraw-Hill Book Company, New York USA 1973

[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, 133(4), 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, 1, 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, 1–6

[16] Ciletti M.D., *Advanced digital design with the Verilog HDL*, Prentice Hall of India, New Delhi, India 2005

[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, 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, 54(2), 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, 5(3), 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, 176–181

[21] Salauyou V.V., *Verilog language in embedded systems design on FPGA*, Hotline – Telecom, Moscow, Russia 2020

[22] Bergamaschi R.A., *Behavioral Synthesis: an Overview* [w:]. R. Reis, M. Lubaszewski, J.A. Jess (red.), *Design of Systems on a Chip: Design and Test*. Springer, Boston, USA 2006, 103–131

ASMD-FSMD for designing digital devices on FPGA, based on the Verilog hardware description language

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.

In the technique ASMD-FSMD, the device 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, the FSMD states are explicitly defined in ASMD. This paper provides a formal description of ASMD-FSMD technology. 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. Our ASMD chart can describe any type of machine, both Mealy and Moore. 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.

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.