WYDZIAŁ ELEKTRONIKI
POLITECHNIKA KOSZALIŃSKA

Projekt zespołowy

Rok akademicki 2008/2009	**Część1: Projekt potokowej jednostki przetwarzającej przeznaczonej do realizacji algorytmu FFT**		
Kierunek studiów:	Informatyka	Wykonawca:	Popielewski Robert
Semestr:	VII		
Grupa:	PKiSI 2		

Numer zadania 2 (zgodnie z plikiem zadaniaprojzesp.pdf):

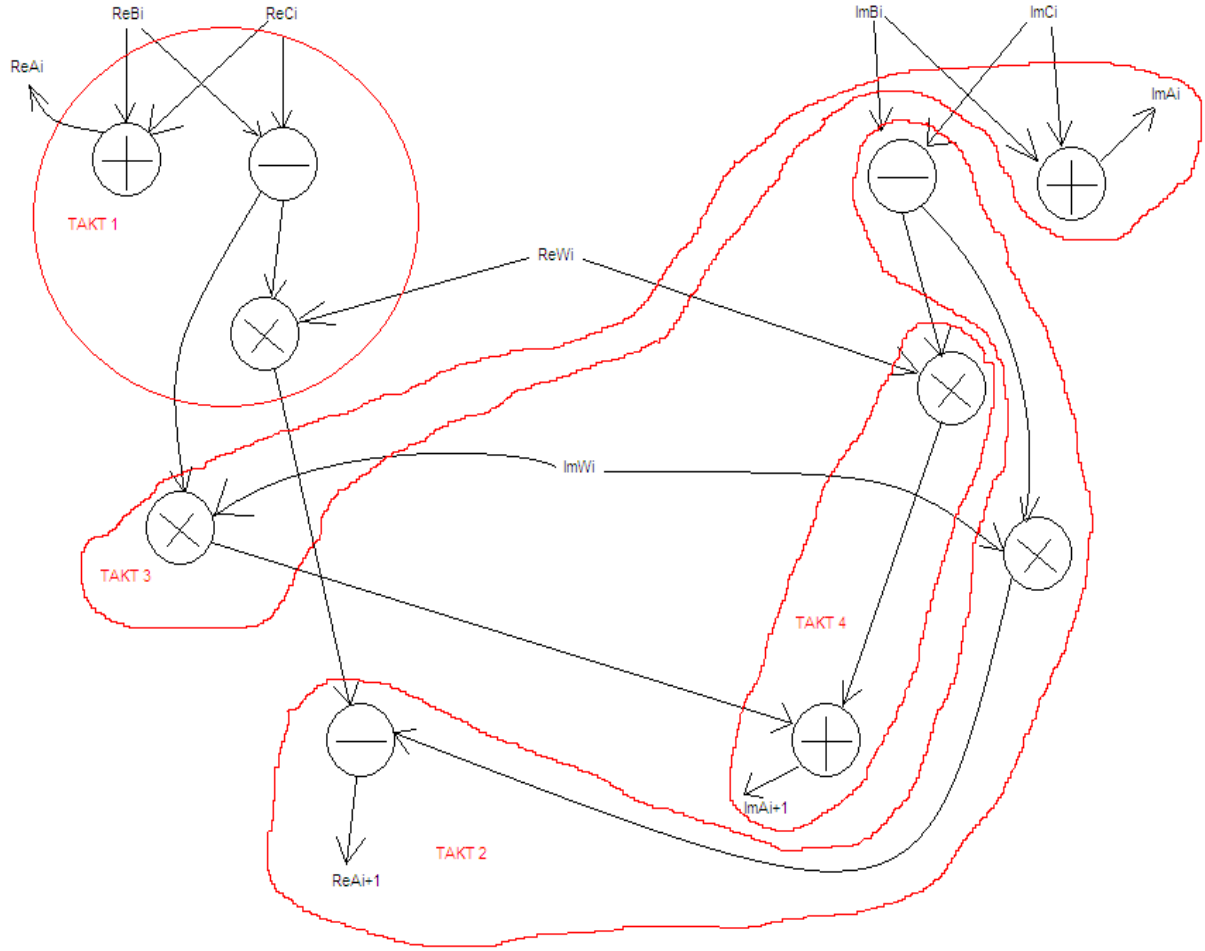
- FFT z podziałem w dziedzinie częstotliwości (F)
- Liczba bloków mnożących: 1
- Liczba bloków sumujących: 2
- Maksymalna długość cyklu obliczeniowego: 4 takty
- Odwrócona bitowo kolejność danych na wejściu

Projekt rozpocząłem od zaprojektowania jednostki ALU. Jednostka ta musiała zrealizować w ciągu 4 taktów:

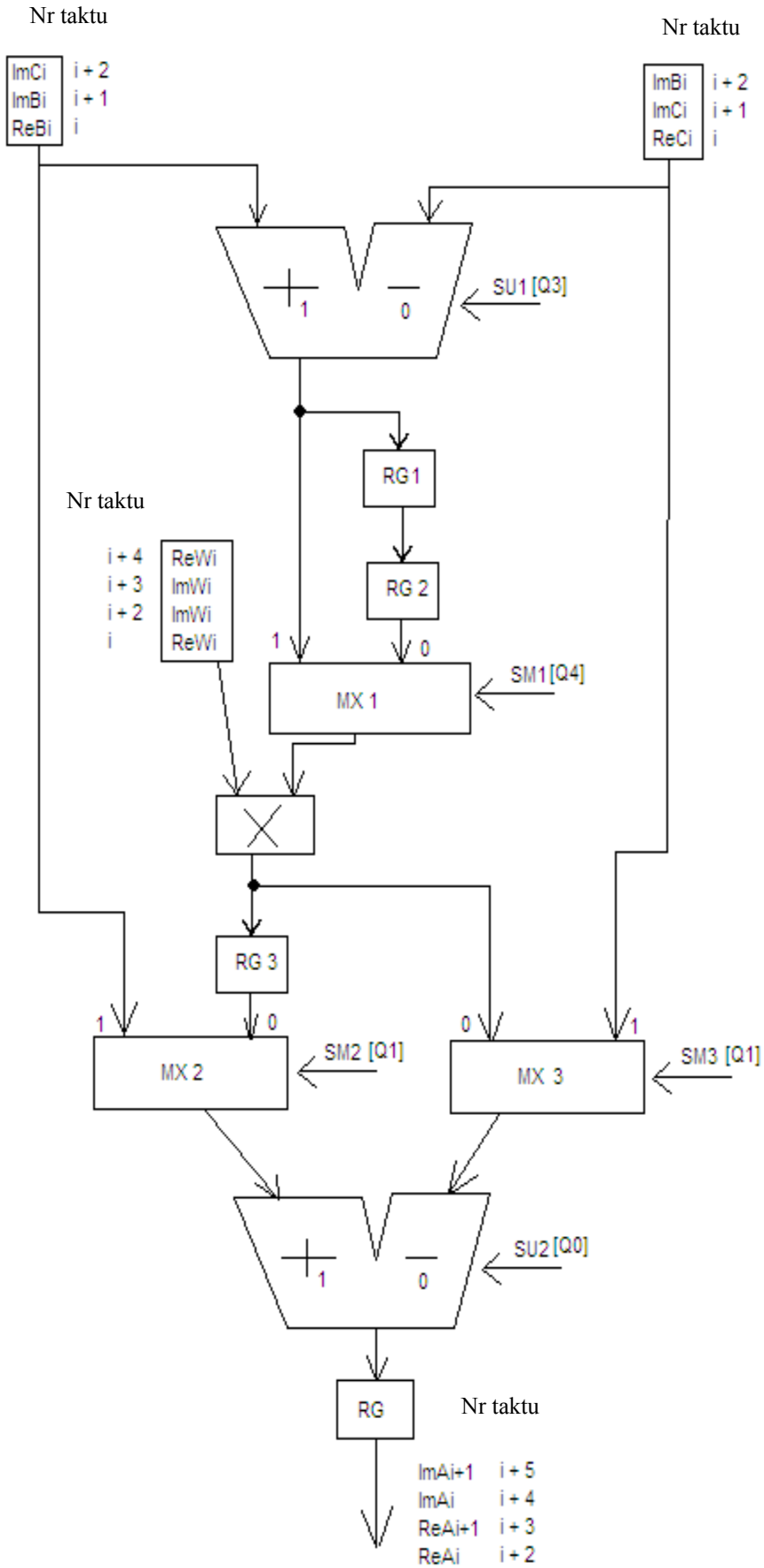
- 4 operacje mnożenia
- 6 operacji sumowania

Operacje te to:
ReAi = ReBi + ReCi
ImAi = ImBi + ImCi
ReAi+1 = (ReBi – ReCi) * ReWi – (ImBi – ImCi) * ImWi
ImAi+1 = (ReBi – ReCi) * ImWi + (ImBi – ImCi) * ReWi

Poniższy graf prezentuje te operacje:



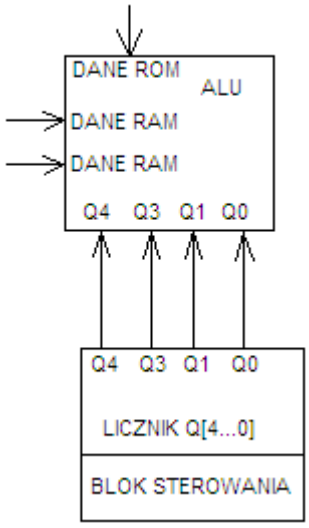
Ponieważ dysponuję niewielką ilością sumatorów, toteż każdy z wyników (tj. ReAi, ImAi, ReAi+1, ImAi+1) wyprowadziłem szeregowo. W oparciu o powyższy graf zaprojektowałem schemat ALU:


Nr taktu
ImCi
ImBi
ReBi
i + 2
i + 1
i
Nr taktu
ImBi
ImCi
ReCi
i + 2
i + 1
i
1
0
SU1 [Q3]
RG 1
Nr taktu
i + 4
i + 3
i + 2
i
ReWi
ImWi
ImWi
ReWi
RG 2
1
0
MX 1
SM1 [Q4]
RG 3
1
0
MX 2
SM2 [Q1]
0
1
MX 3
SM3 [Q1]
1
0
SU2 [Q0]
RG
Nr taktu
ImAi+1 i + 5
ImAi i + 4
ReAi+1 i + 3
ReAi i + 2

Rejestr RG na samym dole schematu jest, w praktyce często stosowanym elementem opóźniającym, zapewniającym stabilną pracę układu.
Blok sterowania (który jest celem tego projektu) oddziaływać będzie na elementy układu:

- SU1[Q3]
- SU2 [Q0]
- SM 1[Q4]
- SM 2 [Q1]
- SM 3 [Q1]

Wartości w nawiasach kwadratowych [] oznaczają poszczególne sygnały wychodzące z bloku sterowania. W ten sposób można narysować pierwszy schemat oddziaływań między blokiem sterowania a układem ALU:



Aby sprecyzować, co w który takcie ma miejsce, można skonstruować poniższą tabelę:

Nr taktu	Operacja	SU1	SU2	SM1	SM2 SM3	Odczyt z RAM	Odczyt z ROM
i	ReBi – ReCi => RG1 (ReBi – ReCi)ReWi => RG3 ReBi+ReCi => ReAi	0-	1+	1	1	ReBi ReCi	ReWi
i+1	RG1 => RG2 ImBi – ImCi => RG1 RG3 – (ImBi – ImCi)*ImWi => ReAi+1	0-	0-	1	0	ImBi ImCi	ImWi
i+2	(RG2)*ImWi => RG3 ImBi + ImCi => ImAi	dc(1)	1+	0	1	#!RG[ImCi ImBi]	ImWi
i+3	RG1 => RG2 RG2 * ReWi + RG3 => ImAi+1	dc(1)	1+	0	0		ReWi

W każdym z taktów do dyspozycji miałem 1 mnożnik i 2 sumatory, czyli w 4 taktach miałem do dyspozycji tak jakby 4 mnożniki i 8 sumatorów co może bez największego problemu zrealizować 4 operacje mnożenia i 6 operacji (dodawania/odejmowania). Po wykonaniu wszystkich potrzebnych mi operacji, powinny mi zostać 2 operacje sumowania. Na tabeli w miejscu tych 2-óch niepotrzebnych sumowań (takt i+2 oraz takt i+3) wstawiłem wartość dc (ang. don't care). W miejsca te można wstawić dowolny bit. Ja wstawiłem „1" i „1" ponieważ taki układ uprości mi tabelę w następujący sposób:

- SU1 = |SM1
- SM2 = SM3

Uproszczenie to spowoduje mniejszą liczbę sygnałów wychodzących z bloku sterowania. Zapis #!RG[] opiszę w dalszej części sprawozdania.

Aby powstał kompletny blok sterowania, należy opracować schemat odczytu z pamięci RAM i ROM oraz zapisu do pamięci RAM, bo być może tutaj również pojawi się multiplekser, którym trzeba będzie sterować sygnałami z bloku sterowania.
W jednostce RAM znajdują się dane:

- ReBi, ImBi
- ReCi, ImCi

Odczyt z pamięci RAM danych B i C przebiega w następujący sposób:

TAKTY	1	2	3	4
	B	C		

Ponieważ w 1 i 2 takcie odczytywane są argumenty B i C toteż 3-ci i 4-ty takt mogą posłużyć do zapisu do tej samej pamięci RAM:

TAKTY	1	2	3	4
	B	C	Ai	Ai+1

Ze schematu ALU odczytujemy kolejność odczytywanych i zapisywanych danych:
Odczyt danych RAM:

1. ReBi + ReCi
2. ImBi + ImCi
3. ImCi + ImBi

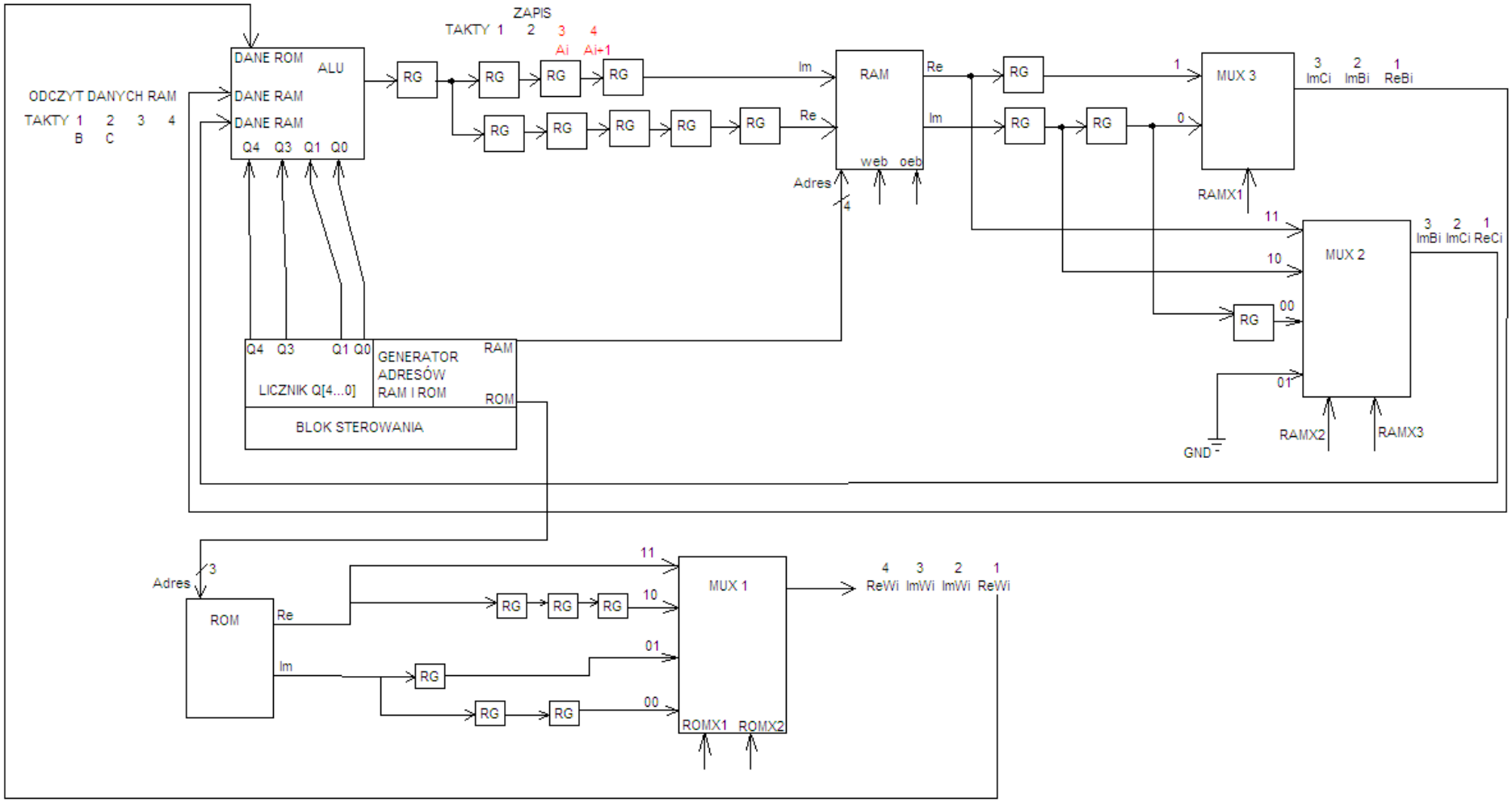
Odczyt danych ROM

1. ReWi
2. ImWi
3. ImWi
4. ReWi

Zapis danych do RAM:

3. ReAi + ImAi
4. ReAi+1 + ImAi+1

W ten sposób układ odczytu i zapisu do pamięci ulegnie znacznej rozbudowie, ale z całego układu odpadnie dodatkowa kość RAM'u. Poniższy układ został zaprojektowany w taki sposób aby spełniał wymagania co do odczytu z układów RAM i ROM oraz zapisu do RAM. Należy pamiętać iż układ RAM i ROM dysponują wejściem do wskazywania na aktualnie czytaną komórkę pamięci. Układ RAM dodatkowo dysponuję wejściami odczytu (oeb) i zapisu(web). Jednoczesny zapis i odczyt jest nie możliwy.


ZAPIS
TAKTY 1 2 3 4
Ai Ai+1
ODCZYT DANYCH RAM
TAKTY 1 2 3 4
B C
DANE ROM
ALU
DANE RAM
DANE RAM
Q4 Q3 Q1 Q0
RG
Im
Re
RAM
web oeb
Adres
4
MUX 3
3 2 1
ImCi ImBi ReBi
RAMX1
MUX 2
3 2 1
ImBi ImCi ReCi
RAMX2
RAMX3
GND
Q4 Q3 Q1 Q0
LICZNIK Q[4...0]
GENERATOR
ADRESÓW
RAM I ROM
BLOK STEROWANIA
Adres
3
ROM
MUX 1
4 3 2 1
ReWi ImWi ImWi ReWi
ROMX1 ROMX2

W schemacie dochodzą dodatkowo 3 multipleksery, które mogą być sterowane za pomocą bloku sterowania. Wszystkie operacje opisałem na poniższej tabeli:

Nr taktu	Operacja	SU1	SU2	SM1	SM2 SM3	Odczyt z RAM	Odczyt z ROM	Zapis do RAM
i	ReBi – ReCi => RG1 (ReBi – ReCi)ReWi => RG3 ReBi+ReCi => ReAi	0-	1+	1	1	ReBi ReCi	ReWi	
i+1	RG1 => RG2 ImBi – ImCi => RG1 RG3 – (ImBi – ImCi)*ImWi => ReAi+1	0-	0-	1	0	ImBi ImCi	ImWi	
i+2	(RG2)*ImWi => RG3 ImBi + ImCi => ImAi	dc(1)	1+	0	1	#!RG[ImCi ImBi]	ImWi	
i+3	RG1 => RG2 RG2 * ReWi + RG3 => ImAi+1	dc(1)	1+	0	0		ReWi	
i+4	ReBi – ReCi => RG1 (ReBi – ReCi)ReWi => RG3 ReBi+ReCi => ReAi	0-	1+	1	1	ReBi ReCi	ReWi	
i+5	RG1 => RG2 ImBi – ImCi => RG1 RG3 – (ImBi – ImCi)*ImWi => ReAi+1	0-	0-	1	0	ImBi ImCi	ImWi	
i+6	(RG2)*ImWi => RG3 ImBi + ImCi => ImAi	dc(1)	1+	0	1	#!RG[ImCi ImBi]	ImWi	ReAi ImAi
i+7	RG1 => RG2 RG2 * ReWi + RG3 => ImAi+1	dc(1)	1+	0	0		ReWi	ReAi+1 ImAi+1

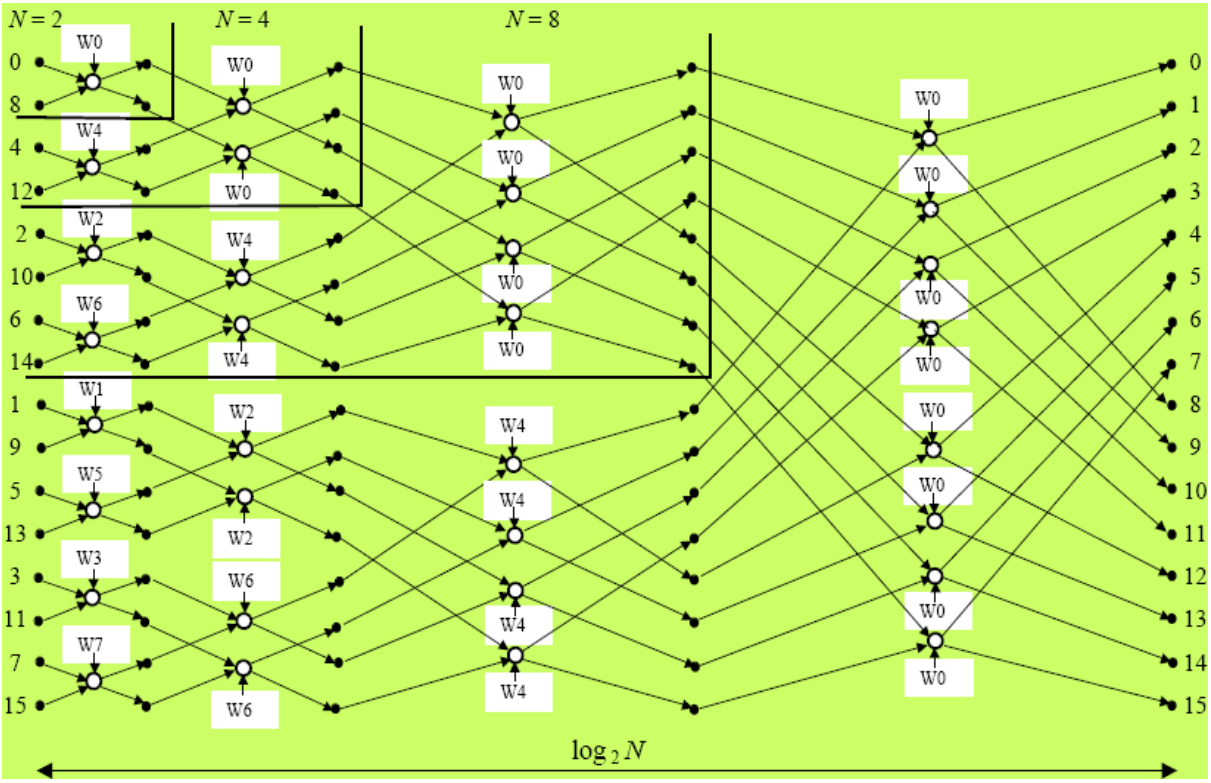
Zielone grube linie w tabeli zaznaczają pełny cykl wraz zapisem do pamięci RAM.

Wnioski jakie płyną z powyższej tabeli są następujące:

Realizacja operacji w ALU odbywa się w czterech taktach, ale ponieważ zdecydowałem się na użycie jednej kości RAM do odczytu danych i ich zapisu, toteż zapis danych musi przedłużyć się o jedną operację bazową. Jest to efekt tego iż w trzecim i czwartym takcie pierwszej operacji bazowej, mając do dyspozycji 1 mnożnik oraz 2 sumatory w ALU, nie mogę wyprowadzić jednocześnie wyników odpowiednio Ai oraz Ai+1.
W taktach i+2 oraz i+6 w kolumnie Odczyt z RAM widnieje następujący zapis:
#!RG[ImCi ImBi]
Oznacza on , iż tak naprawdę dane ImCi oraz ImBi nie są odczytywane z pamięci RAM ponieważ ta operacja przebiegła w taktach poprzednich. Dane te pochodzą z rejestrów, które przechowały informacje z wcześniej odczytanej pamięci RAM. Oznacza to iż w taktach i+2 oraz i+3 mogę dokonać zapisu danych w układzie RAM.
Ostatnim wnioskiem będzie to iż zapis do pamięci RAM będzie opóźniony o 4 takty.

Teraz należy przeanalizować układ generatora adresów RAM/ROM.

Odwrócona kolejność bitowa na wejściu, podział w dziedzinie częstotliwości oraz sposób zapisu wyników do pamięci RAM wymusiły odpowiednie zaprojektowanie generatora adresów:



Korzystając z powyższego rysunku zaprojektowałem poniższą tabelę (kolejność danych jest przestawiona dla kroków N=4 i N=8 ale to w żaden sposób nie wpływa na działanie układu):

N = 2		N = 4		N = 8		N = 16	
RAM	**ROM**	**RAM**	**ROM**	**RAM**	**ROM**	**RAM**	**ROM**
0000 1000	000	0000 0100	000	0000 0010	000	0000 0001	000
0100 1100	100	0010 0110	100	0001 0011	100	1000 1001	000
0010 1010	010	0001 0101	010	1000 1010	000	0100 0101	000
0110 1110	110	0011 0111	110	1001 1011	100	1100 1101	000
0001 1001	001	1000 1100	000	0100 0110	000	0010 0011	000
0101 1101	101	1010 1110	100	0101 0111	100	1010 1011	000
0011 1011	011	1001 1101	010	1100 1110	000	0110 0111	000
0111 1111	111	1011 1111	110	1101 1111	100	1110 1111	000

Z powyższej tabeli można określić pewne zależności między powstawaniem adresów RAM i ROM.

N = 2		N = 4		N = 8		N = 16	
RAM	ROM	RAM	ROM	RAM	ROM	RAM	ROM
0000 1000	000	0000 0100	000	0000 0010	000	0000 0001	000
0100 1100	100	0010 0110	100	0001 0011	100	1000 1001	000
0010 1010	010	0001 0101	010	1000 1010	000	0100 0101	000
0110 1110	110	0011 0111	110	1001 1011	100	1100 1101	000
0001 1001	001	1000 1100	000	0100 0110	000	0010 0011	000
0101 1101	101	1010 1110	100	0101 0111	100	1010 1011	000
0011 1011	011	1001 1101	010	1100 1110	000	0110 0111	000
0111 1111	111	1011 1111	110	1101 1111	100	1110 1111	000

Załóżmy, że pojedyncze bity RAM nazywają się następująco:
RA3 RA2 RA1 RA0
przy czym A0 to bit najmłodszy.
Natomiast bity ROM:
RO2 RO1 RO0

Na powyższej tabeli zaznaczyłem następujące zależności:
Dla kroku N = 2

- RO2 = RA2
- RO1 = RA1
- RO0 = RA0

Dla kroku N = 4

- RO2 = RA1
- RO1 = RA0
- RO0 = 0

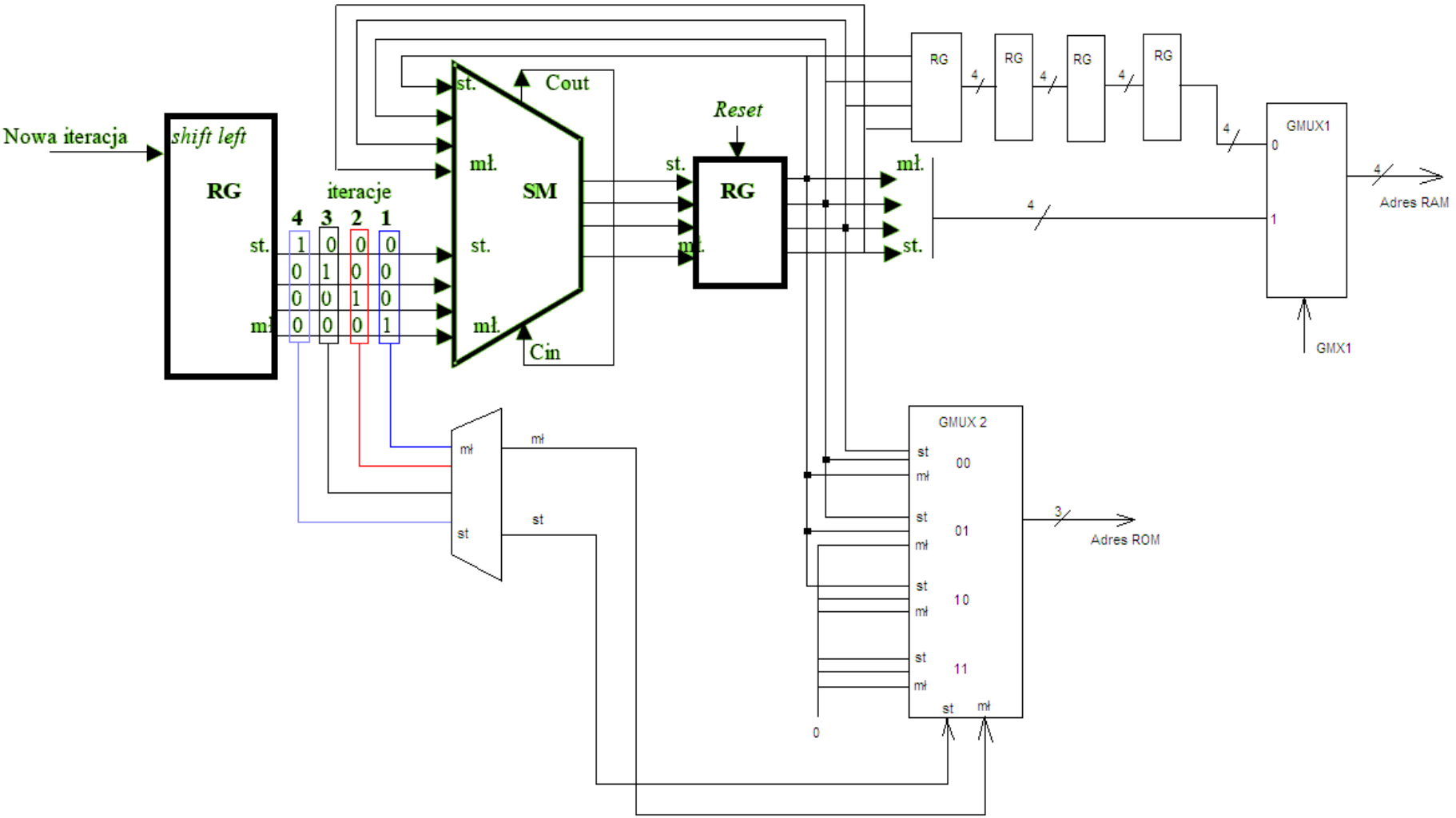
Dla kroku N = 8

- RO2 = RA0
- RO1, RO0 = 0

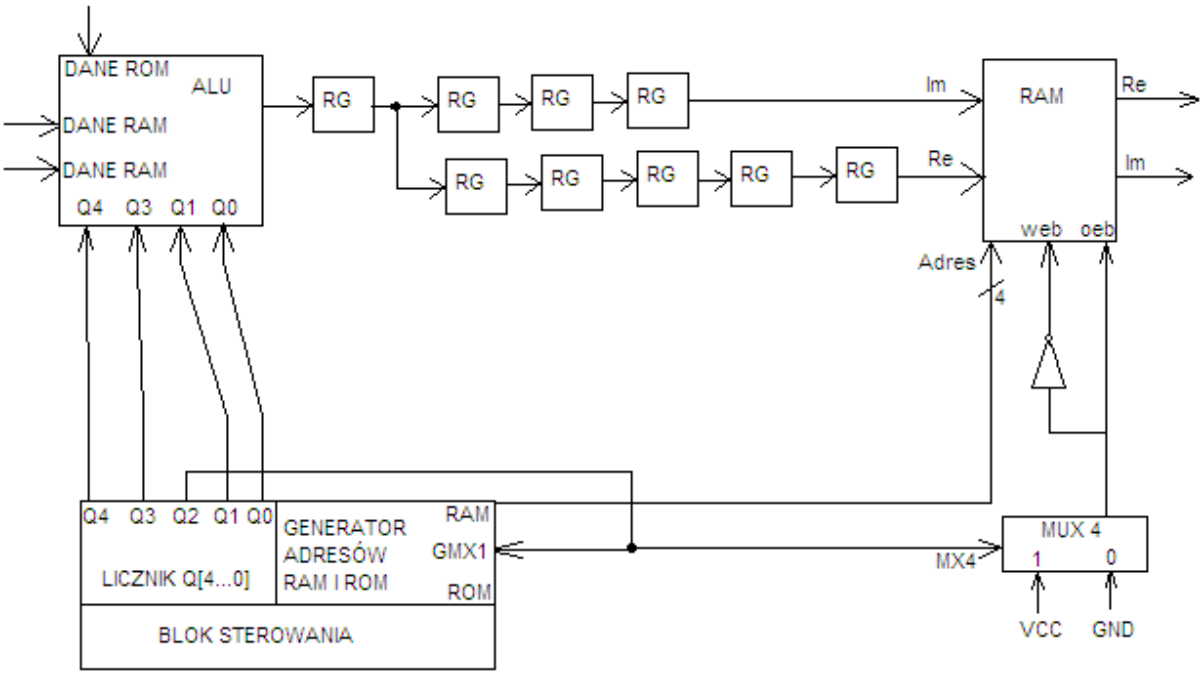
Dla kroku N = 16

- RO2, RO1, RO0 = 0

Zależności te należało wykorzystać do stworzenia generatora adresów RAM/ROM:


Nowa iteracja
shift left
RG
st.
mł.
iteracje
4 3 2 1
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
SM
Cout
Cin
Reset
RG
RG
RG
RG
RG
4
GMUX1
0
1
Adres RAM
GMX1
GMUX 2
00
01
10
11
3
Adres ROM
0

Adresy ROM tworzone są na podstawie Iteracji(Kroku). Dla kroku N = 2 będą generowane te adresy ROM, które zostały opisane wyżej w tabelach. Sterowanie wychodzących adresów ROM zapewnia dekoder, którego wyjście połączone jest z wejściami adresowymi multipleksera GMUX2.
Z powodu opóźnienia w zapisie w układzie RAM, generator adresów musiał zostać wyposażony w dodatkowe rejestry przechowujące stary adres do zapisu danych. Zostało to zrealizowane na 4 rejestrach. Adres do układu RAM jest generowany w zależności od tego czy mamy do czynienia z zapisem na układ RAM (0 na wejście GMX1), czy odczytem z pamięci RAM (1 na wejście GMX1). Bit sterujący wejściem adresowym GMX1, będzie również sterował sygnałami web(zapis) i oeb(odczyt) pamięci RAM. Bit ten będzie generowany z bloku sterowania. Na schemacie zaznaczony jest on jako Q2:



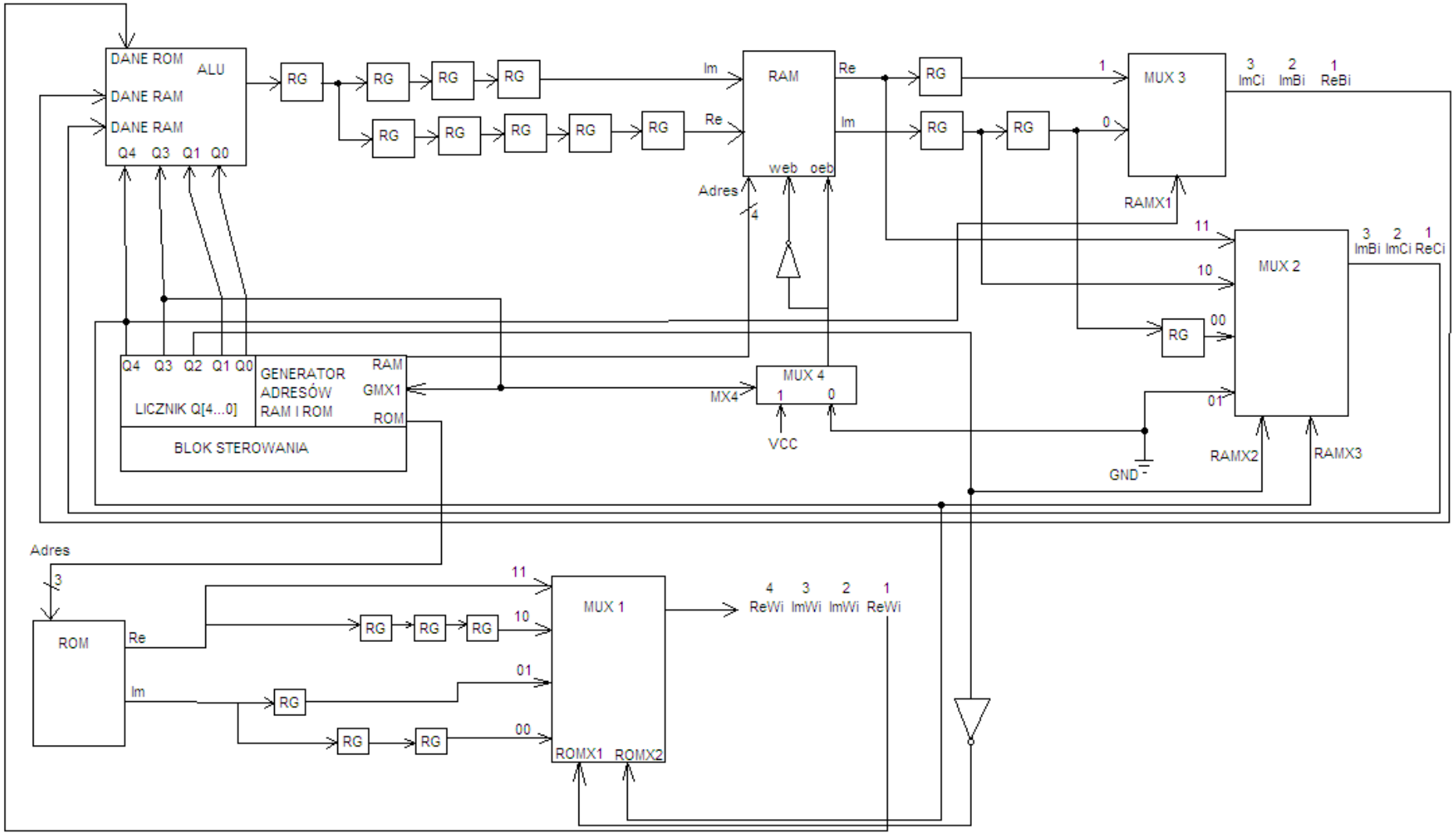
Sterowanie multiplekserami oraz sumatorami może zostać zastąpione prostym licznikiem który cyklicznie będzie generował impulsy do odpowiednich układów. Licznik zaprojektowałem na przerzutnikach typu D. Poniżej zaprezentuję uproszczoną tabelę odpowiadającą za sterowanie wszystkimi multiplekserami oraz sumatorami:

	SM1, RAMX3, ROMX2, RAMX1 [Q4]	SU1, MX4, GMX1 [Q3]	RAMX2, |ROMX1 [Q2]	SM2, SM 3 [Q1]	SU2[Q0]
S	**0**	**0**	**0**	**0**	**0**
i	1	0	0	1	1
i+1	1	0	1	0	0
i+2	0	1	1	1	1
i+3	0	1	0	0	1

~ROMX1 = zanegowany sygnał ROMX1

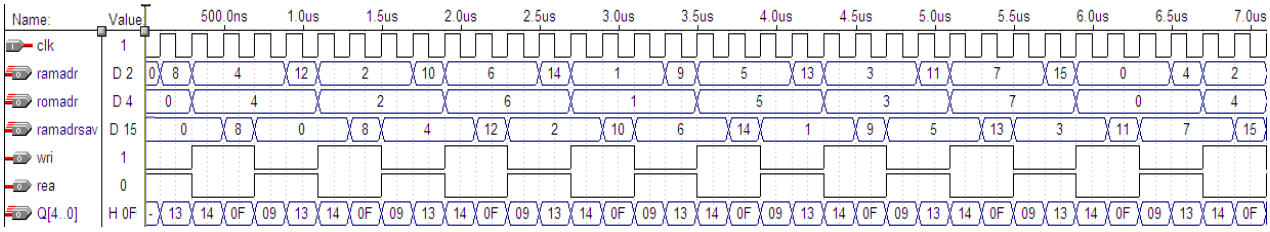
D4 = ~q2 *(~q4 *~q3* ~q1 *~q0 + q4* ~q3 *q1 *q0 + ~q4 *q3 *~q1 *q0)
D3 = q2 *(q4 *~q3* ~q1 *~q0 + ~q4 *q3 *q1 *q0)
D2 = q4 *~q3(~q2 *q1 *q0 + q2 *~q1 *~q0)
D1 = ~q1 *(~q4 *q3 *~q2* q0 + q4 *~q3 *q2 *~q0 +~q4 *~q3* ~q2 *~q0)
D0 = ~q4 *(~q3*~q2*~q1*~q0 + q3*q2*q1*q0 + q3*~q2*~q1*q0) + q4*~q3*q2*~q1*~q0

Takt S jest początkową pozycją w której znajduje się licznik przed uruchomieniem. Wraz z pierwszym taktem przerzutników, następuje prawidłowe generowanie impulsów na sumatory i multipleksery.
Mając opisany cały blok sterowania, sporządziłem kompletny schemat całego układu:

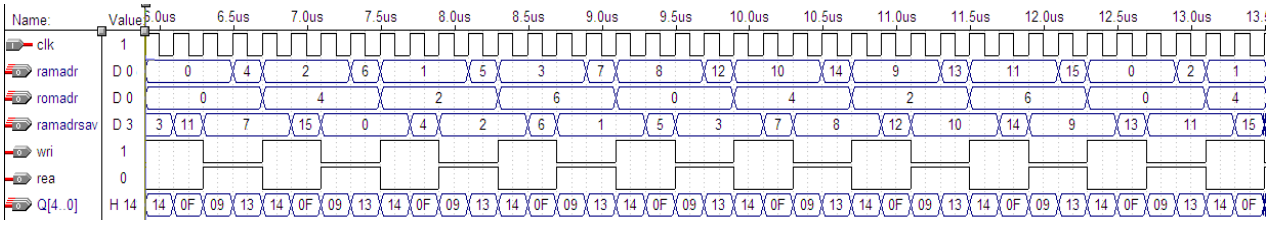

DANE ROM
ALU
DANE RAM
DANE RAM
Q4 Q3 Q1 Q0
RG
Im
RAM
Re
web oeb
Adres
4
1
MUX 3
0
3 2 1
ImCi ImBi ReBi
RAMX1
11
10
MUX 2
00
01
3 2 1
ImBi ImCi ReCi
RAMX2
RAMX3
Q4 Q3 Q2 Q1 Q0
LICZNIK Q[4...0]
GENERATOR
ADRESÓW
RAM I ROM
RAM
GMX1
ROM
BLOK STEROWANIA
MUX 4
MX4
VCC
GND
Adres
3
ROM
Re
Im
MUX 1
ROMX1 ROMX2
4 3 2 1
ReWi ImWi ImWi ReWi

Wyniki:

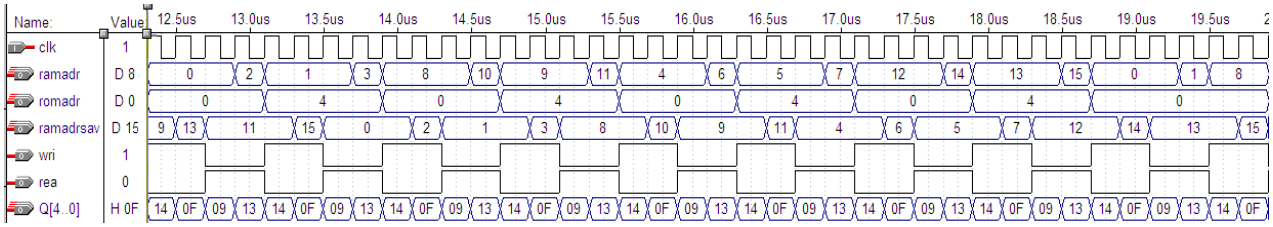
Krok 1:



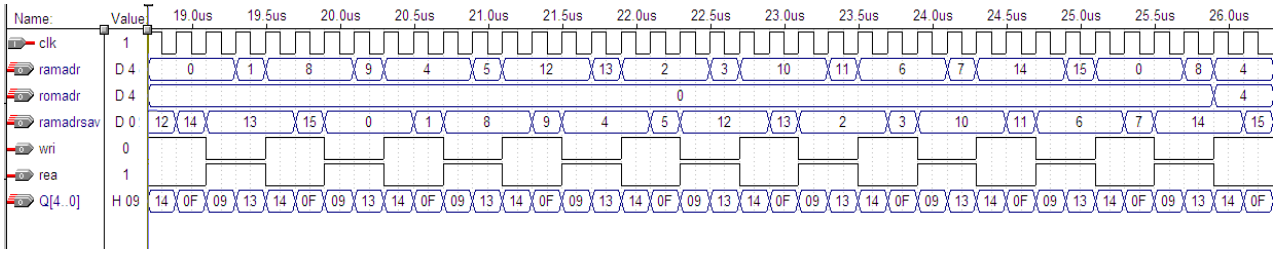
Krok 2:



Krok 3:



Krok 4:



clk – zegar doprowadzony do wszystkich układów mających w swojej strukturze przerzutniki.
ramadr – adres RAM z którego odczytywana jest wartość.
romadr – adres ROM
ramadrsav – adres RAM do którego jest zapisywany jest wynik z ALU
wri – sygnał sterujący zapisem. (web)
rea – sygnał sterujący odczytem (oeb)
Q[4..0] – sygnały sterujące multiplekserami i sumatorami