POLITECHNIKA WROCŁAWSKA
Wydział Elektroniki

ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW

Projekt i implementacja mikrosterownika PIC16C5X wzbogacona o przetwarzanie potokowe i przewidywanie skoków.

Sprawozdanie z realizacji projektu.

Prowadzący:
Dr inż. Tadeusz Tomczak

Michał Salamaga, 226191
Kamil Paczos 218377

Grupa: I
Temat: 2
Termin: CZ/TP/11:15

Wrocław, 16 maja 2019

Spis treści:

1. Wstęp

1.1 Cele i założenia projektu

Celem projektu było stworzenie mikrosterownika PIC16C5X. Miał on zawierać pamięć, możliwość wykonywania instrukcji i układy wejścia/wyjścia.
Do realizacji projektu użyliśmy oprogramowania Tkgate umożliwiającego zbudowanie układu oraz jego testy.

2. Implementacja

Implementacja projektu została utworzona w narzędziu przeznaczonym do projektowania i symulowania układów elektronicznych – Tkgate.

2.1 Ogólna architektura układu

Do stworzenia układu użyliśmy:
- standardowych bramek logicznych
- pamięci ROM i RAM
- multiplekserów
- wejść zegarowych
- przełączników
- sumatorów
- registrów
- przełączników DIP
- przerzutników typu D

2.1.1 Opis architektury

Do prawidłowego działania mikrokontrolera utworzono pamięć pod nazwą *Program memory*. Przechowuje ona kolejne instrukcje dla układu. Instrukcje te są przesyłane do rejestru instrukcji (*Instruction register*), który na jeden cykl zegarowy je zapisuje. Następnie bity od 6 do 11 trafiają do pamięci ROM nazwanej *ALU Control*. Pamięć ta w zależności od podanego adresu wybiera odpowiednią linię w multiplekserze znajdującym się w jednostce arytmetyczno-logicznej (*ALU*). W zależności od wybranej linii przesyłana jest wybrana liczba lub stałe zero lub zawartość rejestru roboczego powiększona o zawartość pamięci RAM (adres pamięci, spod której jest pobierana wartość to 5 najmłodszych bitów instrukcji). Dane te mogą trafić do rejestru roboczego w zależności od instrukcji.
W układzie został zastosowany blok (*Program counter*), służy on do wywoływania kolejnych adresów w pamięci zawierającej instrukcje (*Program memory*).

2.1.2 Wykaz sygnałów

Na wykresach czasowych znajdują się oznaczenia sygnałów zgodne z tabelą poniżej.

Tabela 1: Wykaz symboli sygnałów w układzie

Kod sygnału	Opis
CLK	Sygnał zegara
IR_out	Wyjście rejestru instrukcji
IR_mx_IN	Wejście multipleksera licznika programu zawierające adres z rejestru instrukcji
PC_1	Wyjście z inkrementera licznika programu
PC_Input	Wejście rejestru licznika programu
pcout	Wyjście rejestru licznika programu
acc_in	Wejście rejestru roboczego W (akumulatora)
acc_out / acc_out0	Wyjście rejestru roboczego W (akumulatora)
ALU_Ctrl_out	Wyjście pamięci ROM kontrolującej ALU
ram_in	Wejście adresowe pamięci RAM

RAM_out	Wyjście danych z pamięci RAM
Program	Aktualna instrukcja (zamiennie z IR_out)
Z_flag	Flaga Z (zero)
xor_ins_in	Wejście bramki XOR wprowadzające dane z aktualnej instrukcji
xor_out	Wyjście z bramki XOR
const_mux_ctrl	Sygnał sterujący multiplekserem wybierającym stałą 1 / 2 w module licznika rozkazów

2.2 Zaimplementowane instrukcje

Wszystkie instrukcje, które zostały zaimplementowane w projekcie, działają i są obsługiwanie zgodnie z definicją zawartą w *Instruction Set Summary* procesora PIC16C5X.

2.2.1 GOTO

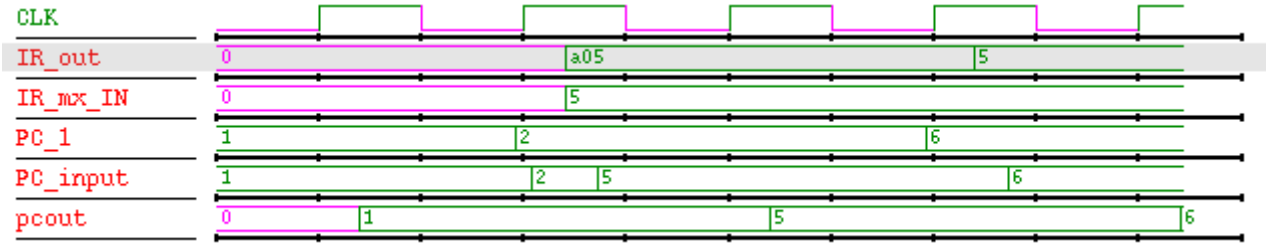
W mikrokontrolerze PIC16CX5 instrukcja GOTO jest instrukcją skoku bezwarunkowego. Jej 12-bitowy kod składa się z kodu instrukcji ‘*101’* i 9-bitowego adresu skoku. W implementacji przyjęto, że wartość wpisana jako adres jest bezpośrednio wskaźnikiem na miejsce w pamięci programu.

Za przetworzenie instrukcji skoku odpowiada dwu wejściowy multiplekser. Wejście 0 połączone jest z wyjściem inkrementera adresu. Wejście 1 natomiast zawiera 9-bitowy fragment wyjścia z rejestru instrukcji. Za sterowanie multiplekserem odpowiada bramka AND z zanegowanym jednym wejściem. Połączona jest ona kolejno z bitami 11, 10 (zanegowany) i 9 wyjścia z rejestru instrukcji. Układ ten pozwala wytworzyć na wyjściu stan wysoki w momencie pojawienia się instrukcji skoku. W tym momencie multiplekser przełączany jest na kanał 1, a na wejściu licznika programu pojawia się adres pobrany z instrukcji skoku.

Działanie instrukcji skoku zostało zademonstrowane na prostym programie przedstawionym poniżej:

0
A05
2
3
4
5
6
7

Jak widać kod zawiera instrukcje, które ze względu na brak implementacji zostaną potraktowane jako NOP. Na drugim miejscu w kodzie programu zakodowano instrukcję o kodzie ‘*1010 0000 0101’*, czyli instrukcję skoku do adresu 5. Z przeprowadzonego testu wynika, że instrukcja została zaimplementowana prawidłowo, i po adresie 1 licznik programu przyjął wartość 5.



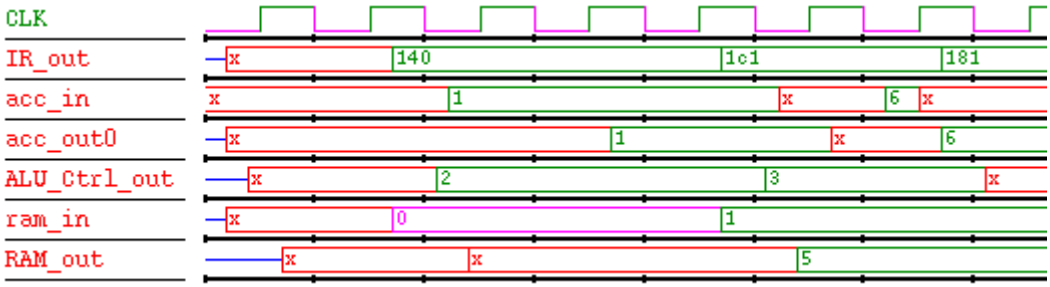
Rysunek 1: Wykres czasowy testu instrukcji GOTO

2.2.2 ADDWF

Instrukcja dodawania ADDWF służy do dodania liczb znajdujących się w rejestrze roboczym W oraz wskazanym rejestrze f. W celu jej implementacji niezbędne było wykonanie rejestru opartego na pamięci RAM zdolnego do przechowywania danych i pozwalającego na dostęp do nich po podaniu odpowiedniego adresu. Dodatkowo zaimplementowany został rejestr roboczy, nazywany zamiennie akumulatorem. Jego wejściem jest multiplekser sterowany przez jednostkę kontrolną wykonaną na pamięci ROM. W zależności od kodu instrukcji steruje ona wejściem akumulatora ładując go żądanymi wartościami.

Sama instrukcja dodawania składa się 6-bitowego kodu instrukcji i 5-bitowego adresu rejestru, z którego ma być pobrana wartość. Za wykonanie działania odpowiada moduł 8-bitowego sumatora, którego wyjście trafia n multiplekser. Nie została zaimplementowana obsługa przeniesień.

W przebiegu testowym akumulator został załadowany wartością 1, a w pamięci RAM wskazany adres zawierający wartość 5. Wynik, który po wykonaniu operacji pojawił się w akumulatorze wyniósł więc 6.



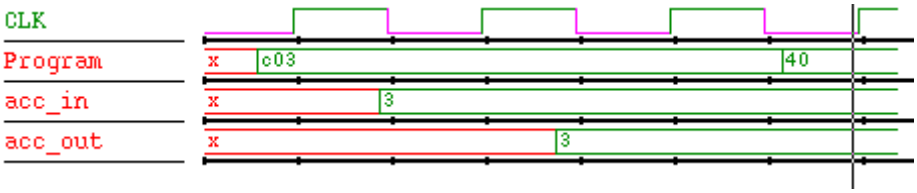
Rysunek 2: Wykres czasowy testu instrukcji ADDWF

Jak widać na załączonym wyżej przebiegu testowym, problemem występującym w układzie jest stabilne utrzymanie wartości akumulatora. Pomimo wielu prób problem ten nie został rozwiązany do końca, a jedynie została zmniejszona jego skala.

2.2.3 MOVLW

Instrukcja ta wykorzystywana jest do załadowania rejestru roboczego 8-bitową liczbą podaną jako jej parametr. Jej wykonanie wymagało dodania kolejnego multipleksera na wejściu akumulatora. Sterowany jest on poprzez bramkę NAND, która przy bitach 8-11 aktualnej instrukcji równych kodowi instrukcji MOVLW aktywuje kanał multipleksera zawierający bity 0-7 bieżącej instrukcji. W przeciwnym przypadku aktywny jest kanał 0, zawierający wartość przekazaną z multipleksera ALU.

Test instrukcji obejmował załadowanie rejestru roboczego wartością 3.



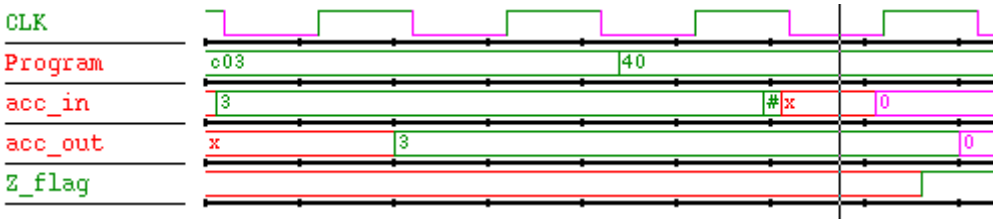
Rysunek 3: Wykres czasowy testu instrukcji MOVLW

2.2.4 CLRW

Instrukcja CLRW jest odpowiedzialna za czyszczenie rejestru roboczego z jego poprzedniej wartości i wpisanie do niego zera. Implementacja tej instrukcji była bardzo prosta i ograniczyła się do zmiany zawartości pamięci ROM odpowiedzialnej za sterowanie multiplekserem jednostki ALU. Jeżeli w instrukcji pojawi się kod CLRW, wówczas multiplekser sterowany jest na kanał zawierający stałe 0. Wartość ta jest następnie wpisywana do akumulatora.

Dodatkowo zaimplementowany został szczątkowy rejestr flag, zawierający flagę Z odpowiedzialną za sygnalizację zera w akumulatorze. Flaga ta ustawiana jest w momencie wykrycia włączenia kanału zwierające stałe 0 w multiplekserze.

Test obejmował wypełnienie rejestru instrukcją MOVLW oraz następnie wyzerowanie go omawianą instrukcją CLRW.

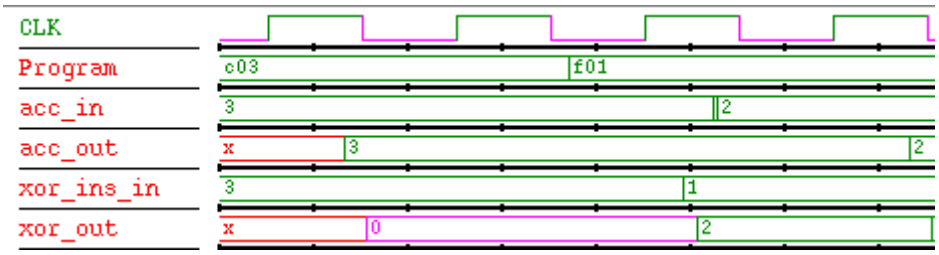


Rysunek 4: Wykres czasowy testu instrukcji CLRW

2.2.5 XORLW

Instrukcja ta służy do wykonania operacji logicznej XOR na rejestrze roboczym oraz 6-bitowej liczbie podanej w kodzie instrukcji. Jako, że liczba ALU, a co za tym idzie cały układ operują na liczbach 8-bitowych, wymagane było uzupełnienie liczby z instrukcji o dwa zera na początku jej zapisu. Otrzymana 8-bitowa liczba była przekazywana na bramkę XOR, której drugim wejściem była zawartość rejestru roboczego. Wyjście z tej bramki skierowane zostało na dodatkowy multiplekser, zarządzany przez bramkę AND wykrywającą kod instrukcji XORLW na linii wychodzącej z rejestru instrukcji. Wyjście multipleksera wpisuje odpowiednią wartość do akumulatora.

Test działania instrukcji obejmował wykonanie operacji XOR na liczbie 1 przekazanej w instrukcji i liczbie 3 przechowywanej w akumulatorze. Wytworzony został poprawny wynik – 2, który następnie został wpisany do akumulatora.

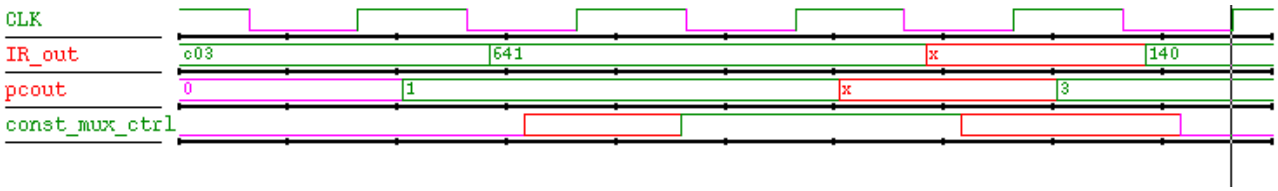


Rysunek 5: Wykres czasowy testu instrukcji XORLW

2.2.6 BTFSC

Ze zbioru instrukcji operujących na bitach
zaimplementowana została instrukcja odpowiadająca za sprawdzenie bitu i pominięcie kolejnej instrukcji, jeżeli bit ten równy jest 0. Kod instrukcji tej zawiera informacje o tym, który bit powinien zostać sprawdzony oraz z którego rejestru powinien zostać pobrany. W celu jej zaimplementowania wyjście z pamięci operacyjnej rozdzielone zostało na pojedyncze bity i skierowane na multiplekser 8-kanałowy, sterowany przez 3 bitowy numer z rejestru instrukcji. Wyjście multipleksera zostało zanegowane i połączone operacją AND z bramką wykrywającą kod instrukcji BTFSC w szynie danych rejestru instrukcji. W wyniku tego, gdy wykryta zostaje instrukcja pominięcia i bit jest ustawiony na 0 wytworzona zostaje logiczna jedynka. Wykorzystana zostaje ona do sterowania multiplekserem umieszczonym przy inkrementerze licznika programu. Gdy wykonana ma być instrukcja pominięcia, licznik programu zamiast o 1, zwiększany jest o 2.

Test działania instrukcji obejmował obserwację licznika rozkazów w momencie, gdy wywołana została instrukcja BTFSC. W czasie wywołania instrukcji testowany był drugi bit liczby zapisanej w pamięci RAM, która w tym wypadku wynosiła 1. Jak widać na wykresie czasowym instrukcja została poprawnie wykonana a licznik programu zwiększony o 2.



Rysunek 6: Wykres czasowy testu instrukcji BTFSC

Jak widać w układzie występują stany nieustalone. Wszelkie próby ich zniwelowania nie powiodły się.

3. Wnioski

Implementacja mikrokontrolera jest zadaniem bardzo złożonym. Wymaga wykonania analizy wielu aspektów, a także dokładnego rozplanowania i nieustannego testowania tworzonych rozwiązań. Pomocny na pewno nie był fakt, że Tkgate jest narzędziem bardzo niestabilnym. W trakcie pracy często zdarzają się momenty, w których program samoczynnie zamyka się lub nie jest w stanie połączyć właściwych sygnałów. Problematyczne okazało się również wykonanie podziału wykonania programu na potoki. Istnieją bardzo ograniczone źródła poruszające ten temat, w związku z czym w skończonym czasie nie było możliwe zaimplementowanie tego rozwiązania.

Wykonany układ pomimo tego, że w znacznej większości działa prawidłowo, ma problem ze stanami nieustalonymi i czasami przepływu sygnału. Jest to zdecydowanie obszar, w którym jest możliwość dalszego rozwoju układu i poprawy przygotowanej implementacji.

Projekt znajduje się w repozytorium Github: https://github.com/lelu0/oiak_projekt_2019.

4. Bibliografia

[1] Mlb.co.jp. (n.d.). *TKGate User Documention (Editor)*. [online] Available at: http://www.mlb.co.jp/linux/science/tkgate/doc/gateEdit.html [Accessed 16 May 2019].

[2] Pageperso.lif.univ-mrs.fr. (n.d.). *TKGate User Documentation (Simulation)*. [online] Available at: http://pageperso.lif.univ-mrs.fr/~peter.niebert/archi/tkgate-doc/gateSim.html [Accessed 16 May 2019].

[3] What-when-how.com. (n.d.). *Stored Program Processing Part 1 (PIC Microcontroller)*. [online] Available at: http://what-when-how.com/pic-microcontroller/stored-program-processing-part-1-pic-microcontroller/ [Accessed 16 May 2019].

[4] ww1.microchip.com. (2002). *PIC16C5X Data Sheet*. [online] Available at: http://ww1.microchip.com/downloads/en/devicedoc/30453d.pdf [Accessed 16 May 2019].

5. Spis ilustracji

6. Spis załączników

1. Schemat wykonanego układu
2. Instruction Set Summary mikrokontrolera wzorcowego