Luca Giordano, fresk w Palazzo Medici-Riccardi

Hades
Uniwersalny symulator układów

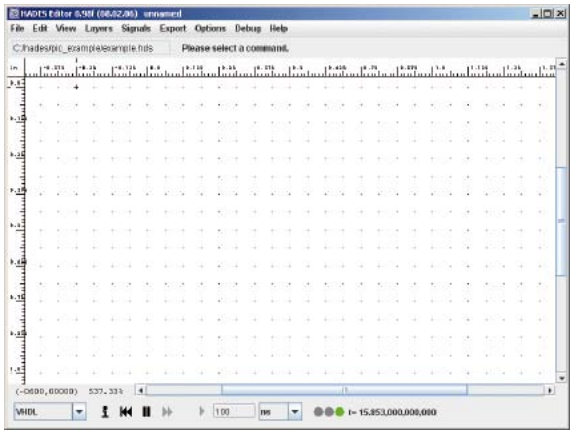
Doświadczenie programisty i jakość użytych narzędzi w czasie pisania nowego lub modyfikacji istniejącego programu głównie decyduje o efekcie końcowym. O ile doświadczenie zdobywa się wraz z ilością napisanych linii kodu, to o jakości narzędzi głównie decyduje zasobność kieszeni. Czasami jednak pro publico bono powstają darmowe narzędzia, które naprawdę warte są zauważenia. Do nich należy między innymi symulator prezentowany w artykule.

W firmach żyjących z tworzenia oprogramowania zazwyczaj się nie oszczędza na kompilatorach, symulatorach, emulatorach i modułach ewaluacyjnych, bo od tego czy szybko i dobrze zostanie napisany program zależy zysk właściciela. W nieco trudniejszej sytuacji mogą być piszący programy w firmach, których główna działalność nie opiera się na tworzeniu oprogramowania, lub mniej lub bardziej zaawansowani amatorzy. Do niedawna właśnie oni mieli dużo trudniej, bo narzędzia były bardzo drogie. Dzisiaj można legalnie dostać bezpłatne kompilatory z niewielkimi ograniczeniami funkcjonalnymi, oraz bezpłatne środowiska projektowe z wbudowanymi programowanymi symulatorami.

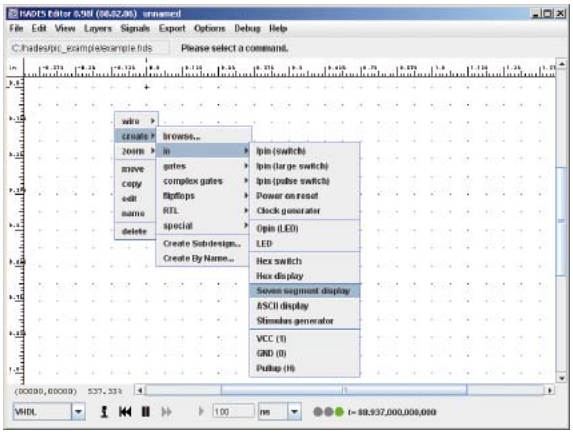
Symulowanie wykonywanego kodu jest jedną z głównych czynności wykonywanych w trakcie pisania programu. Umiejętne wykorzystanie programowanego symulatora potrafi znacznie skrócić czas usuwania błędów z programu. Jednak programowe symulatory są obarczone zasadniczym ograniczeniem. Nie potrafią w pełni symulować działania mikrokontrolera w połączeniu z układami peryferyjnymi.

Ograniczeń symulatorów programowych nie mają symulatory sprzętowe. Potrafią emulować działanie mikrokontrolera w działającym układzie z rzeczywistą prędkością taktowania. Przykładem takiego rozwiązania może być MPLAB ICE 2000 Mimo niezaprzeczalnych zalet, ich stosowanie jest ograniczone dość wysoka ceną. Producenci chcąc obniżyć ceny sprzętowej symulacji wbudowują w mikrokontrolery z pamięcią programu Flash układy symulatorów sprzętowych współpracujących ze stosunkowo prostymi przystawkami- emulatorami. Takie emulatory (np. ICD2 firmy Microchip) nie są już aż tak bardzo drogie, ale dla mniej zamożnych entuzjastów wydatek rzędu 500...600 złotych może być nie do zaakceptowania. Pozostaje szukanie nietypowych bezpłatnych narzędzi. Jednym z takich rozwiązań jest programowy symulator Hades stworzony przez Normana Hendricha z uniwersytetu z Hamburga.

Hades jest z jednej strony ograniczony do symulowania jednego dość starego mikrokontrolera PIC16F84, ale z drugiej strony ma wiele cech nie spotykanych w innych symulatorach. Ma on przede wszystkim możliwość edytowania otoczenia mikrokontrolera. Autor zdefiniował bibliotekę wielu elementów zewnętrznych: bramek logicznych, przerzutników, generatorów przebiegów cyfrowych, generatorów pojedynczych impulsów, 7-segmentowego wyświetlacza



Rys. 1. Okno symulatora



Rys. 2. Dodawanie nowych elementów do symulacji

LED, wyświetlacza alfanumerycznego LED, diod LED, przycisków (microswitch). Można też zdefiniować zewnętrzne pamięci RAM i ROM, rejestry, multipleksery, sumatory, liczniki. Najbardziej rozbudowane elementy biblioteki to oprócz mikrokontrolera PIC16F84 odbiornik i nadajnik systemu DCF77 (radiowy zegar cyfrowy) i terminal RS232. Każdy z elementów ma swoje wejścia i wyjścia dowolnie łączone w trakcie edycji.

Projekt symulatora jest umieszczony na stronie autora pod adresem *http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/index.html*. Plik symulatora *Hades software archives* można pobrać po otwarciu zakładki *Download*. Na tej stronie warto tez od razu pobrać plik pomocy *Hades tutorial* w formacie .pdf.

Symulator jest napisany w języku Java i wymaga zainstalowanego interpretera. Wszystkie zalecenia dotyczące interpretera Java są dokładnie opisane w pliku pomocy. Autor przewidział możliwość używania symulatora w systemach operacyjnych Windows, Linux i Mac OS. Po ściągnięciu pliku hades.jar trzeba go zapisać w docelowym folderze. Plik z rozszerzeniem .jar jest jednocześnie plikiem spakowanym i wykonywalnym. Można go rozpakować, ale autor wyraźnie zaleca aby tego nie robić, przynajmniej na początku pracy z programem.

Edytor uruchamiany jest po podwójnym kliknięciu lewym klawiszem myszki na pliku *hades.jar* (**rys. 1**).

Większość okna programu zajmuje obszar edytora. Tutaj wykonywana jest właściwa edycja: umieszczanie elementów i połączeń miedzy nimi. Powyżej jest umieszczone okno z nazwą ścieżką dostępu aktywnego projektu, oraz pasek narzędziowy. Ikony sterujące pracą symulatora umieszczono w dolnej części okna symulatora:

- i — Przycisk Info. Po jego przyciśnięciu wyświetlany jest status symulatora
- << — Zatrzymuje i zeruje symulator
- || — Pauza symulacji
- >| — Pojedynczy krok symulacji
- >> — Start lub restart symulacji. Symulacja jest wykonywana do zatrzymywania polecenim stop lub pauza
- > — Wykonywanie symulacji przez zdefiniowany czas

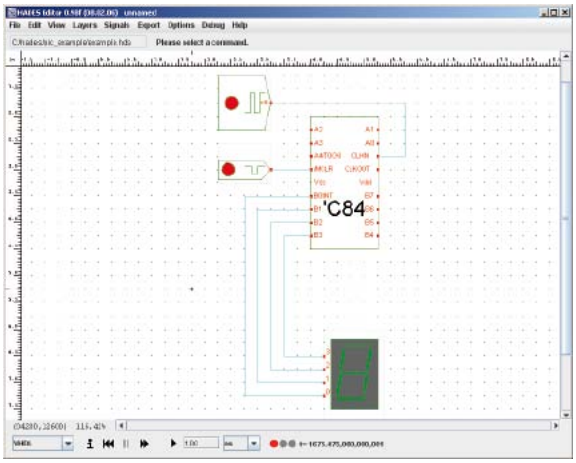
Trzy kolorowe elementy z prawej strony określają status wykonywania programu:

- Zielony – symulacja w toku,
- Żółty – pauza,
- Czerwony – symulacja zatrzymana.

Prędkość wykonywania programu jest definiowana w okienkach pomiędzy ikonami sterującymi, a elementami statusu symulacji.

Edycja środowiska symulacyjnego nie jest trudna. Nowe elementu dodaje się po kliknięciu prawym klawiszem myszki na puste pole edytora. Pojawia się wtedy menu kontekstowe, z którego wybieramy zakładkę Create. Elementy są funkcjonalnie pogrupowane. Wybranie na przykład 7 segmentowego wyświetlacza LED następuje po dwukrotnym kliknięciu na zakładkę Seven segment display (**rys. 2**).

W ten sposób dodawane są wszystkie elementy poza mikrokontrolerem. Chociaż jest zakładka create–>special–>PIC16C84 microcontroller, to nie udało mi się dodać mikrokontrolera w ten sposób.



Rys. 3. Okno symulatora w trakcie edycji

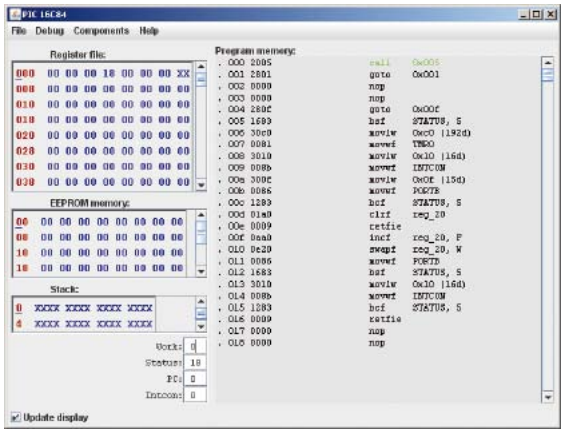
Po skontaktowaniu się z autorem programu okazało się, ze mikrokontroler dodaje się przez zakładkę create –> create by class name: hades.models.pic.Pic16F84. Przygotowane sa tez inne modele mikrokontrolera na przykład hades.models.pic.FastPic16F84 nie wymagający zewnętrznego zegara. Dokładniejsze dane są zawarte w pliku pomocy.

Na **rys. 3** pokazano okno edytora w trakcie edycji z elementami:

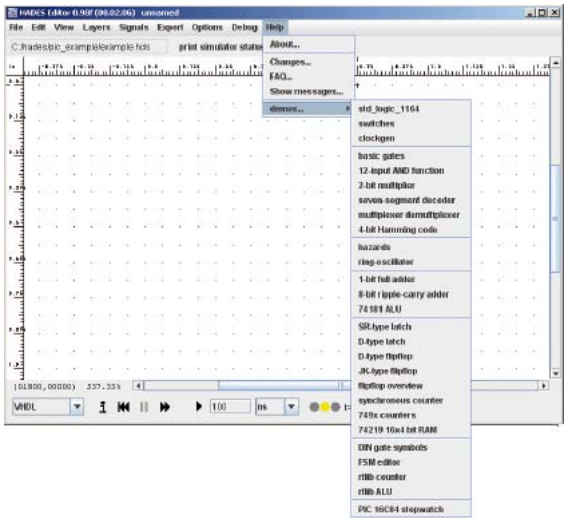
- Mikrokontrolerem PIC16F84,
- Wyświetlaczem 7 segmentowym LED zintegrowanym z enkoderem (element Hex display),
- Generatorem impulsu zerującego mikrokontroler Power On Reset,
- Generatorem sygnału zegarowego clock generator.

Każdy z elementów można edytować, przesuwać lub usuwać po kliknięciu na nim prawym klawiszem myszki i wybraniu z menu kontekstowego odpowiedniej zakładki. Edytowanie elementu zależy od jego typu. Inaczej jest edytowany na przykład generator przebiegu zegarowego, gdzie jest tylko ustawiany głównie okres (częstotliwość), a inaczej mikrokontroler. Połączenia pomiędzy „węzłami" symulowanych układów są wykonywane z zakładki Wire –> Connect. Edytor ma dość spore możliwości i w trakcie tworzenia nowych projektów można będzie je dokładniej poznawać. Na początek wystarczy umiejętność dodawania nowych elementów, ich edycja i wykonywanie połączeń.

Jak można się domyśleć, jednym z najbardziej rozbudowanych elementów jest mikrokontroler. Częstotliwość taktowania można wybrać edytując generator sygnału zegarowego, ale symulowany program musi być wczytany w trakcie edycji samego mikrokontrolera. Program musi być wcześniej napisany w asemblerze i skompilowany na przykład przez bezpłatny MPASM w środowisku MPLAB. Do symulacji będzie nam potrzebny plik wynikowy z rozszerzeniem .hex. Okno edycji mikrokontrolera pokazano na **rys. 4**. Otwiera się je po kliknięciu prawym klawiszem na symbolu mikrokontrolera i wybra-



Rys. 4. Okno edycji mikrokontrolera



Rys. 5. Przykładowe projekty symulacji

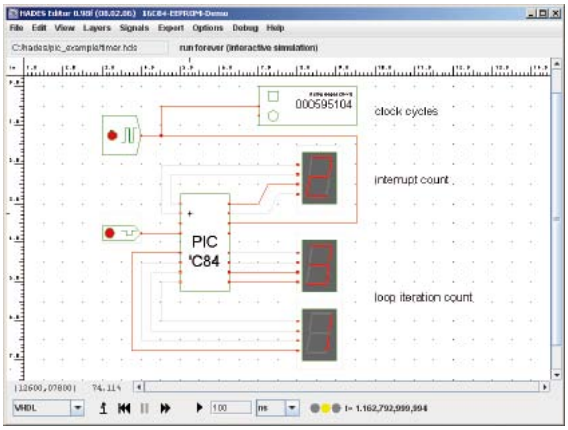
niu Edit. Skompilowany plik otwiera się standardowo z menu *File–>Open*. Po wczytaniu programu z prawej strony okna pojawia się zdisasemblowany program. Z lewej strony umieszczone zostały okna z zawartością pamięci RAM, EEPROM, stosu i rejestrów W, statusowego, licznika rozkazów i Intcon.

Możliwości symulatora można poznać otwierając przykładowe projekty przygotowane przez autora programu. Można otworzyć sobie jeden z przykładowych projektów z zakładki Help->demos (**rys. 5**). Przykłady też można ściągnąć ze strony edytora z zakładki *Downloads –> Hades design examples*. Programy przykładowe dla mikrokontrolera PIC można testować na stronie autora otwierając zakładkę *PIC Cosimulation*. Są tam umieszczone działające aplety Javy i krótki ale wyczerpujący opis każdego z projektów. Istnieje możliwość uruchomienia każdego z tych projektów na symulatorze Hades przez kliknięcie na dole strony na link *Run this demo in the Hades editor (via Java WebStart)*.

Spróbujmy bliżej przyjrzeć się symulacji działania prostego programu zliczającego przerwania od przepełnienia wbudowanego Timera 0. Projekt symulacji jest umieszczony pod adresem *http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/72-pic/10-demo84/timer.html*

Mikrokontroler jest taktowany generatorem sygnału zegarowego o częstotliwości 1 MHz (element *Clock generator*). Sygnał zerowania zapewnia element *Power on reset*. Do portów PORTA i PORTB dołączono trzy 7-segmentowe wyświetlacze zintegrowane z enkoderem (element *Hex display*). Do zliczania cykli zegarowych wprowadzono dodatkowy licznik opisany jako „*clock cycles*" (**rys. 6**).

Wyświetlacz opisany jako „*interrupt count*" pokazuje zawartość licznika zliczającego przerwania od przepełnienia licznika Timer0.



Rys. 6. Okno symulacji przerwania od Timer0

List. 1. Początek programu

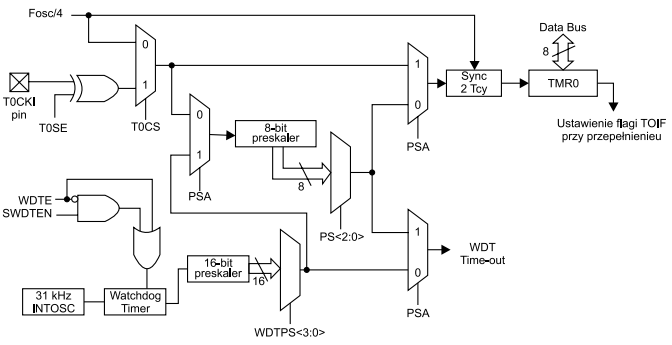
```
org 0          ;wektor po zerowaniu
goto Start     ;początek programu

org 4             ;wektor przerwania
goto InterruptHandler     ;procedura obsługi przerwania

org 10
Start:
call InitPorts     ;inicjalizacja portów
call InitTimer     ;inicjalizacja licznika
```

Program mikrokontrolera jest dość prosty. Po zerowaniu wykonywane są procedury inicjalizacyjne portów i licznika Timer0 (**list. 1**). W czasie inicjalizacji portów rejestry PORTA i PORTB są zerowane, a wszystkie linie portów A i B ustawiane jako wyjściowe przez wyzerowanie rejestrów TRISA i TRISB. Linie A3...A0 sterują wyświetlaczem licznika przerwań, a linie B7...B0 dwoma wyświetlaczami 8-bitowego pomocniczego licznika zliczającego iteracje pomiędzy przerwaniami (*loop iteration count*, **list. 2**).

List. 2. Procedura inicjalizacji portów

```
InitPorts:
  movlw 0x00
  movwf PORTB       ; zerowanie rejestru PORTB
  movwf PORTA       ;zerowanie rejestru PORTA
  bsf STATUS,RP0    ; bank 1
  clrf TRISA       ; wszystkie linie portu A wyjściowe
  clrf TRISB       ; wszystkie linie portu B wyjściowe
  bcf STATUS,RP0   ;bank 0
  return
```

Schemat blokowy licznika Timer0 pokazano na **rys. 7**. Aktualna wartość licznika/timera znajduje się w rejestrze specjalnym TMR0 – rejestr ten można odczytywać i zapisywać. Tryb pracy układu timera/licznika jest ustalany bitem T0CS z rejestru OPTION_REG (**rys. 8**). Gdy T0CS=0, to układ pracuje jako timer. Zliczane są wtedy impulsy z wewnętrznego przebiegu zegarowego mikrokontrolera (Fosc/4). Zawartość rejestru TMR0 można programowo zmieniać, należy jednak zwrócić uwagę, że po każdym wpisaniu nowej wartości do rejestru TMR0, zliczanie impulsów jest przerywane na dwa kolejne cykle rozkazowe. Powoduje to rozbieżności odmierzanego czasu, jednak



Rys. 7. Schemat blokowy licznika Timer0

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

bit 7: !RBPU=1 – rezystory podciągające do napięcia zasilającego linii portu PORTB odłączone lub =0 – dołączone
bit 6: INTEDG=1 – przerwanie zewnętrzne INT zgłaszane przy narastającym zboczu sygnału na wyprowadzeniu RB0/INT lub =0 – przy opadającym zboczu
bit 5: T0CS=1 – TIMER0 zlicza impulsy sygnału z wyprowadzenia T0CKI, =0 – TIMER0 zlicza impulsy sygnału o częstotliwości Fosc/4
bit 4: T0SE=1 – zwiększanie wartości timera TMR0 przy opadającym zboczu na wyprowadzeniu T0CKI lub =0 – przy zboczu narastającym
bit 3: PSA=1 – preskaler podłączony do licznika WDT, =0 preskaler podłączony do timera TMR0
bity 2...0: PS2...PS0 – bity wyboru wartości współczynnika podziału preskalera

PS2	PS1	PS0	Wartość współczynnika podziału preskalera dla timera TMR0	Wartość współczynnika podziału preskalera dla licznika WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Rys. 8. Rejestr OPTION_REG

można tego uniknąć wpisując odpowiednio skorygowane wartości do rejestru TMR0.

Gdy układ Timer0 pracuje jako licznik (T0CS=1), to zliczane są impulsy z wyprowadzenia T0CKI. W takim przypadku zwiększanie wartości licznika (rejestru TMR0) następuje przy każdym narastającym (gdy T0SE=0 w rejestrze specjalnym OPTION_REG) lub opadającym (gdy T0SE=1) zboczu sygnału podawanego na wyprowadzenie T0CKI.

Rozszerzenie zakresu zliczania modułu Timer0 jest możliwe dzięki wbudowaniu w mikrokontroler programowanego preskalera. Stopień podziału częstotliwości przez preskaler jest ustawiany za pomocą bitów PS2...PS0 z rejestru OPTION_REG (wartość podziału 1:1...1:256). Układ preskalera jest wspólny dla modułu Timer0 i *watchdoga* . O tym gdzie dołączony jest preskaler decyduje stan bitu PSA rejestru OPTION_REG. Gdy PSA z rejestru jest wyzerowany, to preskaler jest dołączony do układu Timer0. Wtedy każde zapisanie do rejestru licznika TMR0 powoduje wyzerowanie licznika preskalera.

Procedura inicjalizacji licznika jest pokazano na **list. 3**. Licznik pracuje w trybie czasomierza (timer) i zlicza impulsy o częstotliwości Fclk/4.

Procedura InitTimer odblokowuje tez przerwania od przepełnienia licznika i ustawia bit globalnego zezwolenia na przerwania GIE.

Preskaler dzieli wejściowe impulsy przez 256. Przerwania od przepełnienia będą zgłaszane po zliczeniu 256 impulsów na wejściu licznika czyli co 262144 cykle zegarowe (256×4×256). W trakcie symulacji można zobaczyć, że wartość na wyświetlaczu *interrupt count* zmienia się przy wielokrotności wartości 262144. Żeby pomiar był prawidłowy, to w edycji licznika trzeba ustawić *rising edge* lub *faling edge*.

Przy każdym przepełnieniu licznika jest ustawiany bit flagi przerwania T0IF, do licznika rozkazów jest ładowany wektor przerwania – adres 0004 i rozpoczyna się wykonywanie procedury obsługi przerwania pokazanej na **list. 4**. Każde przerwanie inkrementuje zmienną N_INTERRUPTS. Po inkrementacji 4 starsze bity tej zmiennej są zerowane, a 4 młodsze przesyłane do rejestru PORTA sterującego wyświetlaczem interrupt count. Procedura przerwania musi się Kończyc wyzerowaniem bitu T0IF i rozkazem powrotu retfie ustawiającym bit GIE, który jest automatycznie zerowany po przyjęciu przerwania.

List. 3. Inicjalizacja licznika Timer0
```
InitTimer:
  bsf STATUS, RP0   ; bank 1
  movlw B'10000111' ; rtcc inc = tcylc/256 = tclk/(4*256)
  movwf OPTION_REG
  bcf STATUS, RP0   ; bank 0
  clrf TMR0         ;zerowanie TMR0 i preskalera
  movlw B'10100000' ;odblokowanie przerwania
  movwf INTCON
  return
```

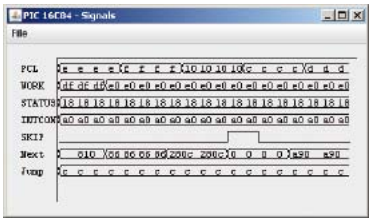
List. 4. Procedura obsługi przerwania
```
InterruptHandler:
  incf N_INTERRUPTS    ;inkrementacja licznika przerwań
  morf N_INTERRUPTS,0
  andlw 0x0f    ;zerowanie 4 starszych bitów
  movwf PORTA   ;przesłanie 4 młodszych bitów licznika
przerwań do PORTA
  clrf INTCON   ;zerowanie T0IF
  bsf INTCON,T0IE  ;odblokowanie T0IE
  retfie        ;powrót z ustawieniem GIE
```

W czasie pomiędzy przerwaniami program w pętli inkrementuje zmienną N_ITERATIONS. Wartość tej zmiennej jest przesyłana do PORTB i wyświetlana na wyświetlaczach loop iteration count (**list. 5**).

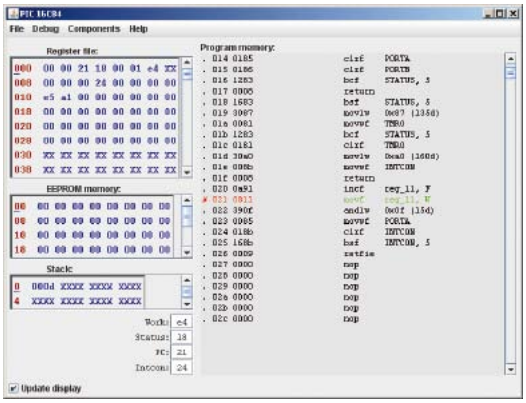
Ponieważ Hades jest symulatorem, to oprócz edycji układu symulowanego jego zasadniczym zadaniem jest symulacja działania projektu. W układach z mikrokontrolerem naturalną symulacją jest krokowe wykonywania programu. W tym celu trzeba otworzyć okno edycji mikrokontrolera, gdzie jest wyświetlany wykonywany program w asemblerze. Przed krokowym wykonywaniem programu trzeba tak ustawić taki czas wykonywanej symulacji żeby był równy jednemu cyklowi maszynowemu mikrokontrolera. Dla 1 MHz cykl zegarowy trwa 1/250 kHz=4 μs. Po tych ustawieniach każde przyciśnięcie ikony „>" w dolnej części ekranu symulatora spowoduje wykonanie jednego rozkazu mikrokontrolera. Aktualnie wykonywany rozkaz jest podświetlany na zielono. Oczywiście wynik działania na przykład zmiany stanów na liniach portów jest uwzględniany w topologii symulowanego układu. W czasie symulacji można na bieżąco podglądać zawartość rejestrów licznika rozkazów (PCL), rejestru roboczego W (Work), rejestrów statusu i INTCON w oknie Signals (**rys. 9**). otwieranym z w oknie edycji PIC16C84

Zawartość okna Signals jest również odświeżana przy ciągłym wykonywaniu programu. Hades ma też możliwość ustawiania pułapek programowych w wykonywanym programie. Żeby ustawić pu-

List. 5. nieskończona pętla programu
```
Loop:
  incf N_ITERATIONS    ;inkrementacja licznika N_
ITERATIONS
  movf N_ITERATIONS,0  ;przesłania zawartości N_
ITERATIONS do PORTB
  movwf PORTB
  goto Loop
```



Rys. 9. Okno Signals



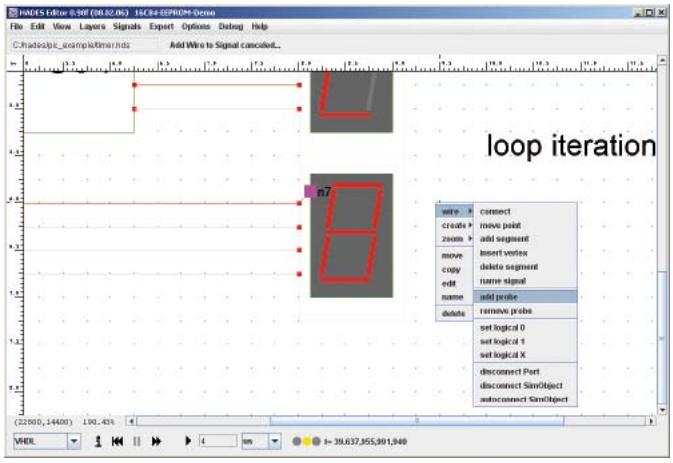
Rys. 10. Ustawianie programowej pułapki

łapkę trzeba kliknąć na pierwsza kolumnę wyświetlanego kodu (kropkę). Kropka jest zamieniana na znak „#" i wiersz wyświetlany jest na czerwono (**rys. 10**). Kiedy licznik rozkazów jest równy adresowi rozkazu z pułapką symulacja jest zatrzymywana i symulator przechodzi w stan pauzy. Można ustawić kilka pułapek programowych.

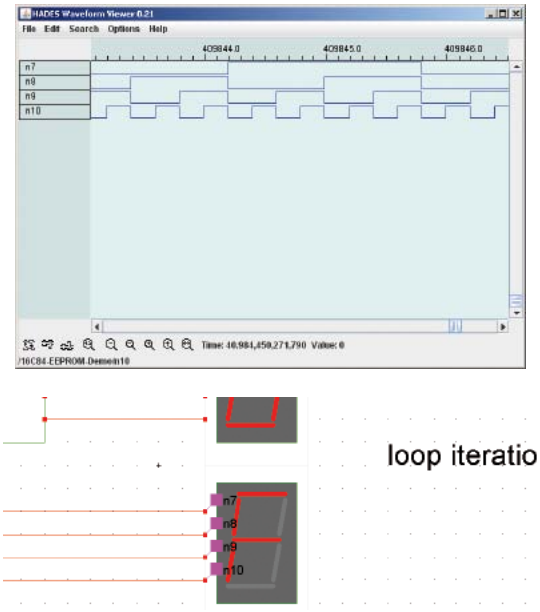
Bardzo interesującą funkcją przydatna przy testowaniu układu jest programowy analizator stanów logicznych. W dowolnym połączeniu pomiędzy układami można dołączyć wirtualną sondę logiczną. Dołączenie sondy powoduje automatyczne zdefiniowanie kanału pomiarowego przypisanemu tej sondzie. Sondę dodaje się z interaktywnego menu z zakładki *wire–>add probe* (**rys. 11**). Każdej z sond program automatycznie nadaje nazwę, którą potem można zmienić poleceniem *name*.

Okno analizatora otwiera się z menu *Signals–>Show Waves*. Na **rys. 12** pokazano okno analizatora z przebiegiem z czterech sond dołączonych do linii sterujących dolnego wyświetlacza *loop iteration count* oraz fragment schematu z miejscem umieszczenia sond.

Prezentowane tu narzędzie ma dość spore możliwości. Z oczywistych względów w artykule zostały opisane tylko wybrane elementy. Czytelnicy zainteresowani programem mogą znaleźć wszystkie niezbędne informacje na stronie autora. Hades jest narzędziem, które może pomóc w pracy nad własnymi projektami. Według mojej opinii może być



Rys. 11. Dodawanie wirtualnych sond



Rys. 12. Okno analizatora i fragment schematu projektu z zaznaczonymi sondami

bardzo przydatny na przykład przy analizowaniu działania mikrokontrolera z zewnętrznymi układami kombinacyjnymi (bramki, przerzutniki, rejestry, sumatory itp.). Doskonała dokumentacja, duża ilość przykładów sprawiają, że warto zainteresować się tym programem.

Tomasz Jabłoński, EP
tomasz.jablonski@ep.com.pl