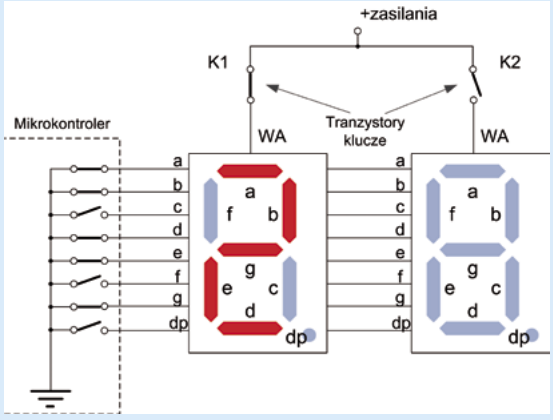
FlowCode i E-blocks (4)

Multipleksowany wyświetlacz LED, używanie przerwań Timera

Flow Code ma specjalne makra przeznaczone do obsługi wyświetlaczy LED. Co ważne, dzięki parametryzacji są one przygotowane do obsługi wyświetlaczy multiplekso-wanych. Można to robić na dwa sposoby – albo układając pojedyncze wyświetlacze 7-segmentowe, albo posługując się komponentem o nazwie „led7seg4". Jest to makro przeznaczone do obsługi wyświetlacza 4-cyfrowego, np. modułu E-blocks o symbolu EB008-00-1. W którymś z kolejnych artykułów pokażemy, jak zastosować taki wyświe-tlacz do zbudowania zegara, jednak teraz nauczymy się elementarza, to jest jak dołą-czyć taki wyświetlacz oraz jak obsłużyć go korzystając z przerwań.

Multipleksowanie wyświetlania to technika znana od bardzo dawna i należy ona do elementarza umiejęt-ności programisty urządzeń embedded. Zasada działa-nia wykorzystuje fizyczną właściwość ludzkiego wzroku – większość ludzi nie jest w stanie dostrzec migotania o częstotliwości wyższej niż 50 Hz. Konstruując wyświe-tlacz multipleksowany trzeba zapewnić takie sterowanie, aby w ciągu cyfr była możliwość włączenia i wyłączenia każdej z nich. Najczęściej wykonuje się to używając wy-świetlaczy ze wspólną anodą, włączając od strony anody i dodatniego bieguna zasilania tranzystory – klucze, na-tomiast katody poszczególnych segmentów wszystkich cyfr zwiera się ze sobą (A z A, B z B itd.). W ten sposób tworzy się wspólną dla wszystkich cyfr, 8-bitową magi-stralę. 8-bitową, ponieważ oprócz 7 segmentów tworzą-cych cyfrę trzeba też uwzględnić kropkę dziesiętną.

Poglądowy **rysunek 1**, na którym linie portów mi-krokontrolera oraz tranzystory – klucze zastąpiono me-chanicznymi włącznikami, ilustruje zasadę działania sterowania multipleksowanego. W danym momencie powinien być zamknięty tylko jeden klucz doprowadza-jący zasilanie do wspólnej anody danej cyfry. Na rys. 1 jest zamknięty klucz K1, natomiast klucz K2 jest otwarty. W ten sposób, dzięki odpowiedniej kombinacji sygnałów na wyjściach mikrokontrolera, na wyświetlaczu – nazwij-my go umownie pierwszym – świeci się cyfra „2", a wy-świetlacz drugi jest zgaszony. Oczywiście, w praktyce za załączanie zasilania anod i zwieranie katod najczęściej odpowiadają mikrokontroler i program sterujący, a wy-łączniki należy zastąpić odpowiednimi driverami, o ile sterowanie wyświetlaczami bezpośrednio z wyjść mikro-kontrolera nie jest możliwe.

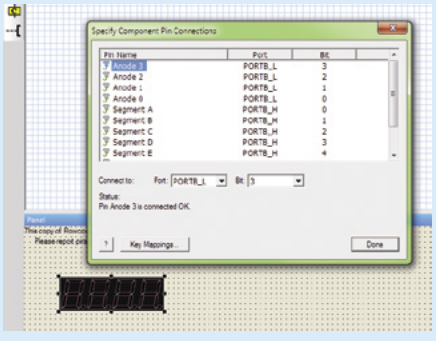


Rysunek 1. Schemat multipleksowania dwóch wy-świetlaczy LED.

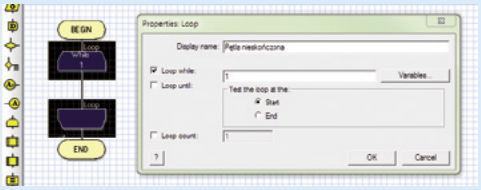
Ćwiczenie 2: Sterowanie wyświetlaczem 4-cyfrowym

Do wykonania tego ćwiczenia użyłem płytek EB0064-00-2 (bazowa z mikrokontrolerem) oraz EB0008-00-1 (wyświe-tlacz 4-cyfrowy, 7-segmentowy LED). Na płytce bazowej zainstalowałem mikrokontroler dsPIC33FJ32GP202. Ca-łość zasiliłem z zasilacza stabilizowanego dostarczające-go napięcie 12 V przy obciążeniu do 1 A. Płytkę z wy-świetlaczami dołączyłem do złącz PORT_BL i PORT_BH. Oprócz tego połączenia, należy za pomocą odrębnego przewodu dołączyć zasilanie od terminatora J1 na płytce bazowej (styku oznaczonego +14 V) do terminatora J3 na płytce wyświetlacza (do styku oznaczonego jako +V). Po tej czynności możemy zająć się wykonaniem programu sterującego.

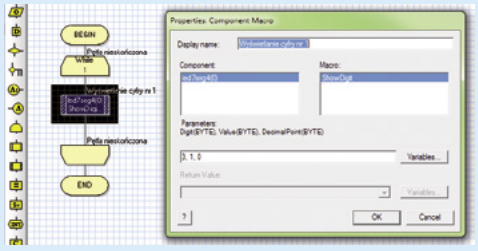
Tworzenie programu rozpoczynamy od uruchomie-nia Flow Code i wybrania z okienka dialogowego opcji *Create a new FlowCode chart*. Następnie określamy typ mikrokontrolera i klikamy na OK. Moduł 4-cyfrowego, 7-segmentowego wyświetlacza LED znajdziemy wśród komponentów *Outputs*. Nosi on nazwę *led7seg4*. Klika-my na jego symbolu, co powoduje ułożenie rysunku wy-świetlacza w obszarze okna *Panel*. Teraz należy dołączyć wyświetlacz do odpowiednich wyprowadzeń mikrokon-trolera. W tym celu klikamy na nim prawym klawiszem myszki i z menu kontekstowego wybieramy *Connections*.



Rysunek 2. Sposób dołączenia modułu wyświetlacza 4-cyfrowego



Rysunek 3. Konfigurowanie pętli *While*



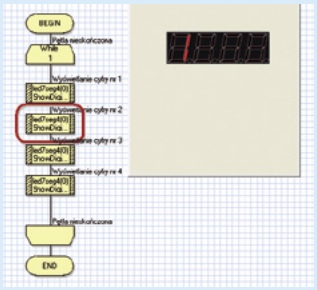
Rysunek 4. Konfigurowanie makra wyświetlającego cyfrę – tu jest to najstarsza cyfra

W tym przykładzie anody wyświetlaczy są dołączone do młodszych bitów portu RB, natomiast segmenty do starszych:

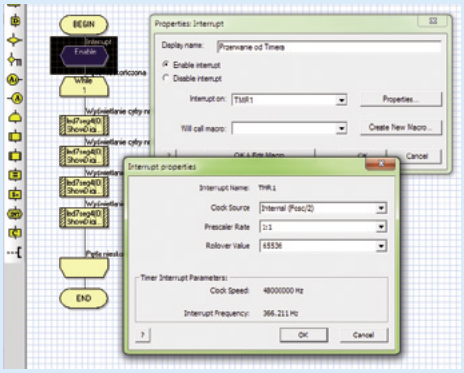
- Anode 0 – PORTB_L(0), … Anode 3 – PORTB_L(3).
- Segment A – PORTB_H(0), … Segment G – PORTB_H(6), Segment DP – PORTB_H(7).

Część definicji połączeń pokazano na **rysunku 2**.

Zgodnie z zasadą multipleksowania, musimy „przemiatać" poszczególne cyfry LED. Jest to operacja cykliczna, powtarzana bez przerwy, najlepiej w ustalonych, stałych odcinkach czasu. Na początek do sterowania wyświetlaniem użyjemy pętli nieskończonej *while(1)*. Jak pamiętamy, aby ułożyć pętlę na diagramie przepływu przenosimy jej symbol z paska ikon w odpowiednie miejsce. Pętla powinna być skonfigurowana w taki sposób, jak na **rysunku 3**. Teraz pomiędzy symbolami początku i końca pętli układamy blok makra obsługi komponentu (*Component Macro*). Dwukrotnie klikamy na jego symbolu, wskazujemy komponent *led-7seg4(0)* oraz makro *ShowDigit* (**rysunek 4**). W pustej linii obok przycisku *Variables* wpisujemy następujące argumenty: *„3, 1, 0"* (ciąg liczb rozdzielonych przecinkiem, bez znaku cudzysłowu). W normalnej sytuacji, gdy samodzielnie tworzymy program do obsługi takiego wyświetlacza, pomiędzy zaświeceniami poszczególnych cyfr trzeba zadbać o wyłączenie poprzednio obsługiwanej. We Flow Code, jeśli używamy makra *ShowDigit*, to nie ma takiej potrzeby. Producent sam zadbał o wyłączenie pozostałych cyfr.



Rysunek 5. Symulowanie pracy programu wyświetlającego stałe w pętli nieskończonej



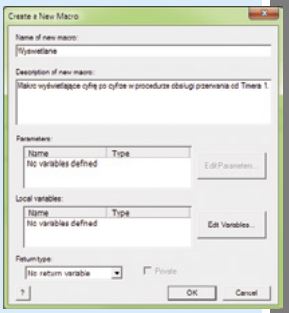
Rysunek 6. Okno właściwości przerwania od Timera 1 przed ustawieniem częstotliwości taktowania

Pierwszy parametr w ciągu określa numer cyfry, która zostanie zaświecona. W naszym wypadku będzie to pierwsza z lewej – to ona nosi numer 0. Drugi to wartość do wyświetlenia (tu „1"). Odpowiednia konwersja liczby dziesiętnej na kod wyświetlacza jest wykonywana przez makro i nie trzeba się nią zajmować. Ostatni parametr określa czy przy wyświetlaniu cyfry ma być załączona kropka dziesiętna. „1" oznacza, że tak, natomiast „0" – nie.

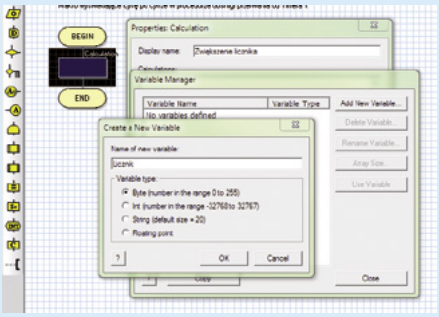
W związku z tym, że mamy 4 cyfry do wyświetlania niezbędne jest użycie 4 makr. Dodajemy je na arkuszu, a jako kolejne parametry wpisujemy: „2, 2, 0", „1, 3, 0", „0, 4, 0". Zapamiętujemy program na dysku. Jego ostateczną wersję pokazano na **rysunku 5**, na którym obok diagramu przepływu zamieszczono również wygląd wyświetlacza symulowanego programowo. Jednak tak wykonany program, o ile dobrze ilustruje zasadę obsługi wyświetlacza, o tyle jest bezużyteczny w zastosowaniach praktycznych. Po pierwsze, posługuje się stałymi wyświetlając liczbę „1234". Do wyświetlenia takiego komunikatu jest szkoda mikrokontrolera – równie dobrze można to zrobić zwierając odpowiednie wyprowadzenia. Po drugie, gdybyśmy pomiędzy makra wyświetlające poszczególne cyfry wstawili jakieś operacje, to w efekcie różnego czasu ich wykonania poszczególne cyfry na wyświetlaczu miałyby różną jasność świecenia. Umieszczenie np. procedury odczytu czujnika DS18B20 tuż za lub tuż przed ciągiem makr wyświetlających wynik pomiaru skończyłoby się migotaniem całego pola. Dlatego wyświetlanie multipleksowane najczęściej jest realizowane za pomocą procedury obsługi przerwania od timera, co gwarantuje ten sam czas wyświetlania poszczególnych cyfr oraz możliwość modyfikowania wskazań „w locie" poprzez podmianę zmiennych do wyświetlenia.

Wykonania nowego programu nie rozpoczniemy jak poprzednio od utworzenia nowego projektu, ale zmodyfikujemy „stary" program. Na początek zajmiemy się przerwaniami. Z paska ikon wybieramy *INT* i umieszczamy tuż przed pętlą *while*. Dwukrotnie klikamy na niej o otwieramy okienko zatytułowane *Properties: Interrupt*. Z rozwijanej listy *Interrupt on:* wybieramy *TMR1*, a następnie klikamy na klawiszu *Properties…*, którego kolor etykiety zmienił się z szarego na czarny. Otworzy się okienko pokazane na **rysunku 6**. Jak można zauważyć, służy ono do ustawienia częstotliwości, z którą będzie uruchamiane przerwanie od Timera 1. Na razie jednak nie ustawiliśmy parametru *Clock Speed*, więc podaną częstotliwość należy traktować jedynie jako ciekawostkę. Klikając na *OK* zamykamy okno i naciskamy klawisz *Create New Macro…* Otworzy się okno pokazane na **rysunku 7**. W pierwszej linii wpisujemy nazwę makro (np. *Wyswietlanie*), możemy przy tym dodać komentarz. Gdyby była potrzeba przekazania parametrów, utworzenia zmiennych lokalnych, czy zwrócenia wartości, to również można to zrobić za pomocą odpowiednich definicji.

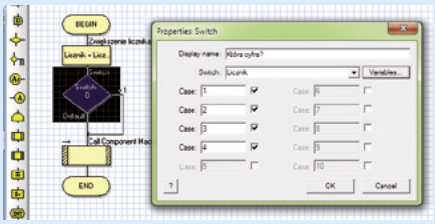
Tu należy się kilka słów na temat procedury obsługi przerwania. Oczywiście, moglibyśmy zbudować ją tak samo, jak „stary" program główny – po odebraniu przerwania CPU mógłby wyświetlić wszystkie cyfry i na tym zakończyć jego obsługę. Jednak w praktyce ten sposób obsługi powoduje, że jest trudno zapanować nad jasnością świecenia segmentów



Rysunek 7. Okno służące do utworzenia makra obsługi przerwania



Rysunek 8. Tworzenie nowej zmiennej licznikowej



Rysunek 9. Konfigurowanie klauzuli *Switch*

w sytuacjach, gdy chcemy zaimplementować regulację intensywności ich świecenia. Ponadto, jak na mój gust, procesor spędza nieco za dużo czasu w procedurze obsługi przerwania. Dlatego każde wejście do procedury obsługi będzie odpowiadało wyświetleniu pojedynczej cyfry. O tym, która to będzie cyfra, zadecyduje wartość zmiennej o nazwie *Licznik*. I tak, wyświetlaniu cyfry nr 1 będzie odpowiadał *Licznik = 4*, cyfry nr 2 *Licznik = 3* itd. Zmienna licznikowa musi być zmienną globalną, ponieważ musi być zachowywana po wyjściu z procedury obsługi przerwania.

Przechodząc od rozważań do czynów, naciskamy klawisz *OK* i zamykamy w ten sposób okienko *Create a new macro*. Teraz klikamy na *OK & Edit Macro*, co powoduje utworzenie nowej zakładki – nowego arkusza przeznaczonego do umieszczenia instrukcji procedury obsługi przerwania. Procedura obsługi wygląda jak zwykły program i w ten sam sposób tworzy się ją. Na początku diagramu, pomiędzy polami *Start – Stop*, umieszczamy pole o nazwie *Calculation*. Klikamy na nim dwukrotnie, otwierając w ten sposób okienko zatytułowane *Properties: Calculation*. Za jego pomocą założymy i zmodyfikujemy zmienną o nazwie *Licznik*. Klikamy na przycisk *Variables*, a następnie *Add New Variable…* i w pustej linijce okienka *Create a New Variable* wpisujemy nazwę *Licznik* (jak na **rysunku 8**). Typ zmiennej *Byte* jest ustawiony domyślnie – po wpisaniu nazwy wystarczy kliknąć na *OK*. Wskazujemy nazwę zmiennej na liście i klikamy na *Use Variable*. Wpisujemy działanie *Licznik = Licznik + 1* i naciskamy OK.

Jako kolejne pole na diagramie umieszczamy klauzulę *Switch*. Posłużyłem się nią dla lepszej czytelności programu. Oczywiście, jeśli ktoś ma na to ochotę, może w tym miejscu zastosować *If*. Jeszcze bardziej eleganckie byłoby użycie licznika jako zmiennej indeksującej tablicę lub wskaźnika, ale w tym momencie odłóżmy tę „akrobatykę" programowania na rzecz czytelności programu.

Wygląd poprawnie uzupełnionego okienka *Properties: Switch* pokazano na **rysunku 9**. Za pomocą Ctrl+X i Ctrl+V przenosimy makra wyświetlające poszczególne cyfry z arkusza *Main* na arkusz *Wyswietlanie* i układamy w odpowiednich miejscach na diagramie. W gałęzi *Default* umieszczamy pole *Calculation*, którego zadaniem jest wyzerowanie zmiennej *Licznik*, jeśli ta nie ma wartości z zakresu 1…4. W ostatnim warunku (*Licznik = 4*) również wyzerujemy zmienną *Licznik* (**rysunek 10**) co oszczędzi nam jeden cykl obsługi przerwania. Gotową procedurę obsługi przerwania pokazano na **rysunku 11**. Niestety, to jeszcze nie wszystko. Makra wyświetlające poszczególne cyfry zostały przez nas przeniesione za pomocą Wytnij – Wklej, a więc nadal zawierają stałe do wyświetlenia. Trzeba je zastąpić zmiennymi, dla uproszczenia nazwijmy zmienne odpowiadające poszczególnym cyfrom: *Cyfra1000, Cyfra100, Cyfra10* i *Cyfra1*, gdzie *Cyfra1000* to najstarsza waga dziesiętna, tu jednostki tysięcy.

Aby dodać zmienne klikamy na „pierwszym z brzegu" makrze obsługi wyświetlania. Pojawi się okno właściwości, w którym klikamy na przycisk *Variables*. Teraz za pomocą *Add New Variable…* dodajmy 4 zmienne typu *Byte*, jak na **rysunku 12**. Nadszedł czas na zmodyfikowanie makr wyświetlających – zmieniamy je, okienko po okienku (pole po polu), wpisując w poszczególne pola parametrów:

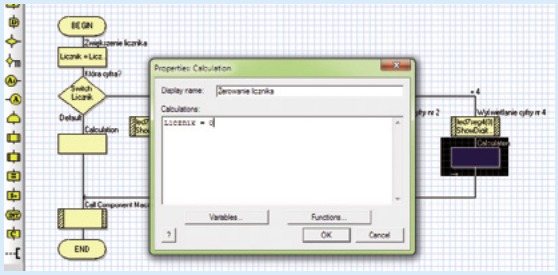
- Dla *Licznik = 1* wpisujemy *„3, Cyfra1000, 0"*.
- Dla *Licznik = 2* wpisujemy *„2, Cyfra100, 0"*.
- Dla *Licznik = 3* wpisujemy *„1, Cyfra10, 0"*.
- Dla *Licznik = 4* wpisujemy *„0, Cyfra1, 0"*.

Po zmodyfikowaniu wszystkich pól makr możemy uznać procedurę obsługi przerwania za zakończoną i wrócić do programu głównego tzn. do zakładki *Main*.

Teraz doprowadzimy do porządku częstotliwość zegarową. Zdecydowałem się na taktowanie mikrokontrolera za pomocą rezonatora kwarcowego umieszczonego na płytce bazowej. Ma on częstotliwość 12 MHz. Jego użycie wymaga ustawienia trybu pracy mikrokontrolera oraz parametrów dla IDE. Parametry mikrokontrolera ustawia się za pomocą menu *Chip -> Configure*. W oknie zatytułowanym *PICmicro Configuration – Slot 2* wybieramy:

- Oscillator mode: Primary Oscillator (XT, HS, EC).
- Two-speed Oscillator Start-Up Enable: Start up with FRC, then switch.
- Primary Oscillator Source: HS Oscillator Mode.
- OSCO/OSC2 Pin Function: OSCO pin Has Digital I/O function.
- Watchdog Timer Enable: Disable.
- JTAG Port Enable: Disabled.

Pozostałe parametry można pozostawić domyślne. Teraz ustawimy częstotliwość oscylatora. Można to zrobić wybierając z menu *View* opcję *Project Options*. Tu z listy *Clock speed (Hz)* wybieramy *12000000* i naciskamy klawisz *OK*. Po zamknięciu okienka ponownie klikamy dwukrotnie na polu zezwolenia na przerwanie (blok *INT*), które teraz nosi nazwę *Enable TMR1* oraz na klawisz *Properties*. Jak można zauważyć, teraz program podaje poprawną częstotliwość, z którą będzie wywo-



Rysunek 10. Zerowanie licznika po wyświetleniu czwartej, ostatniej cyfry

ływane przerwanie. Pamiętajmy, że poszczególne cyfry muszą być odświeżane z częstotliwością co najmniej 50 Hz, a w związku z przyjętą filozofią obsługi przerwania częstotliwość jego wywoływania jest dzielona przez 4. Dlatego też nie powinna ona być mniejsza od 4×50 Hz = 200 Hz. Z drugiej strony, nie może też być zbyt wysoka, tak aby CPU miało czas na realizację innych zadań. W praktyce powinno się wybierać najniższą możliwą częstotliwość zapewniającą kompromis pomiędzy jakością wyświetlania a wydajnością – tu zdecydowałem się na 366 Hz, co powoduje przerwanie wykonywania programu głównego co około 2,7 ms i powoduje odświeżanie cyfr z częstotliwością 91,5 Hz (**rysunek 13**).

Po zamknięciu okna właściwości przerwania pozostało podstawienie zmiennych do wyświetlenia. W tym przykładzie zrobimy to za pomocą pola *Calculation*, jak pokazano na **rysunku 14**. Następnie można przejść do testowania nowoutworzonego programu. W tym celu przesyłamy go do mikrokontrolera, ponieważ symulator kiepsko radzi sobie z emulowaniem przerwań.

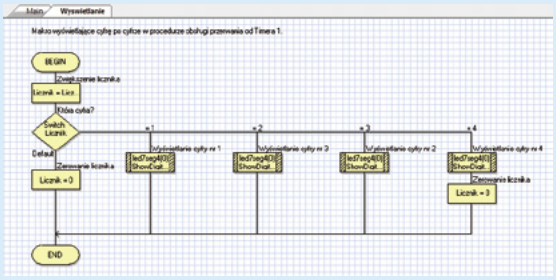
Zauważone błędy

Przez dłuższy czas nie umiałem zmienić typu mikrokontrolera korzystając z menu. Program wykonany np. dla dsPIC30F2011 za nic nie dawał się przenieść np. na dsPIC33FJ32GP202, a przecież to powinna być oczywista funkcja kompilatora. Po pewnym czasie użytkowania Flow Code wreszcie znalazłem na czym polegał mój błąd. Typu mikrokontrolera nie wolno zmieniać za pomocą menu *Chip -> Configure* (chociaż wydaje się to oczywiste), ale należy w tym celu posłużyć się menu *View -> Project Options* i tu, korzystając z listy *Target* , wybrać właściwy typ mikrokontrolera. Wtedy zostaje on zmieniony również w oknie dostępnym w menu *Chip*, a kompilator podmienia odpowiednie pliki nagłówkowe. Przy próbie zmiany typu mikrokontrolera w inny sposób, kompilator „buntuje się”, generuje całą litanię błędów i nie pozwala na zaprogramowanie mikrokontrolera.

Uważny Czytelnik mógł zauważyć, że na początku artykułu napisałem o wyborze mikrokontrolera GP202, a tymczasem kursor myszy osunął się o dwa stopnie w dół i przypadkowo, przez nieuwagę, wybrałem GP302, co można zauważyć na rysunkach. To zmusiło mnie do rozwiązania problemu ze zmianą mikrokontrolera, co szczęśliwie udało się bez przepisywania całego programu.

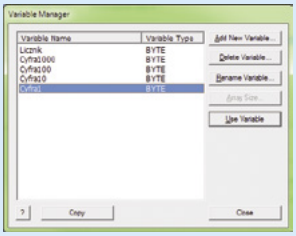
Podczas eksperymentowania z różnymi mikrokontrolerami dsPIC zauważyłem, że po zmianie typu mikrokontrolera interfejs JTAG jest za każdym razem domyślnie załączony. W wypadku konfliktów wyprowadzeń I/O używanych do sterowania dołączonymi peryferiami oraz interfejsu JTAG powoduje to, że program uruchomiony w symulatorze działa normalnie, natomiast po zaprogramowaniu mikrokontrolera powoduje „dziwne” zachowanie urządzenia docelowego. W wypadku wyświetlacza LED były wyświetlane niepełne liczby i to tak zupełnie jakby bez powodu. Moim zdaniem, jeśli natrafi się na jakieś problemy, po pierwsze należy podejrzewać siebie, po drugie – opcje mikrokontrolera (a w nich załączony JTAG lub aktywny, nieobsługiwany *Watchdog*), a dopiero na końcu – kompilator Flow Code.

Wykonując zmiany i często używając symulatora, warto przed uruchomieniem programu zapisać go na dysku. W czasie powstawania tego artykułu co najmniej

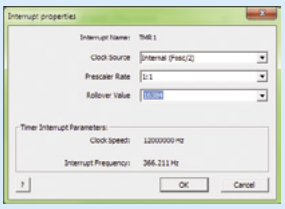


Rysunek 11. Wygląd procedury (makra) obsługi przerwania

ze 2 razy uchroniło to mnie przed koniecznością tworzenia programu od nowa. Symulator (o czym wspomniałem) kiepsko radzi sobie z obsługą przerwań. Na dodatek, o ile przed każdym zaprogramowaniem mikrokontrolera IDE proponuje zapisanie programu źródłowego na dysku, o tyle nie robi tego przed uruchomieniem symulatora. Jeśli symulator spowoduje zawieszenie się IDE, a nie zapiszemy programu na dysku, to tracimy go bezpowrotnie. Ciekaw jestem czy jest to cecha mojego komputera lub systemu operacyjnego, czy też błędu w IDE i czy poprawiono to w wersji 5.



Rysunek 12. Zmienne używane w programie

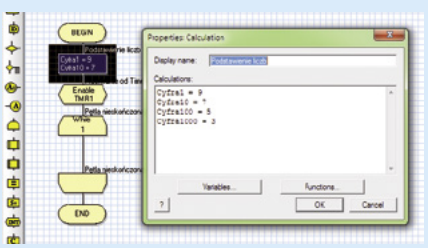


Rysunek 13. Okno właściwości przerwania od Timera 1 po ustawieniu częstotliwości taktowania

Podsumowanie

Po tak zachęcających rezultatach w kolejnym odcinku kursu być może pokusimy się o wykonanie przykładowego programu zegarka. Obsługa przerwań za pomocą Flow Code jest bardzo łatwa i co ważne – obrazowa, nietrudno zrozumieć o co chodzi. Moim zdaniem jest to świetne narzędzie dydaktyczne, które umożliwia przełożenie rysowanych w wielu szkołach średnich oraz na uczelniach wyższych algorytmów na rzeczywisty program dla mikrokontrolera i zobaczenie efektów swojego „rysowania”.

Jacek Bogusz, EP



Rysunek 14. Podstawienie cyfr do wyświetlanie (w efekcie na wyświetlaczu „3579”)

Redakcja Elektroniki Praktycznej dziękuje firmie TME z Łodzi, dystrybutorowi firmy Matrix Multimedia, za udostępnienie zestawu E-blocks oraz mikrokontrolerów używanych w kursie programowania FlowCode.