Visual Micro Lab

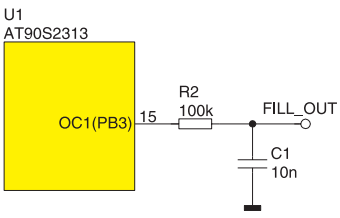
Program Visual Micro Lab przedstawialiśmy na łamach EP już kilka razy. Stworzyła go grupa inżynierów z kilku krajów, zrzeszonych w wirtualną grupę Advanced Micro Tools z siedzibą w Hiszpanii. Mimo że od ukazania się ostatniego artykułu nie minęło dużo czasu, twórcom programu udało się wypuścić do dziś już dwie kolejne wersje programu. Aktualna nosi numer 3.6. Przyjrzymy się jej dokładniej, gdyż z pewnością Visual Micro Lab zainteresuje wielu konstruktorów wykorzystujących w swojej pracy mikrokontrolery ST62 i AVR.

Program wspomagający realizację projektów na mikrokontrolerach AVR i ST62

Przypomnijmy pokrótce najważniejsze własności programu. Visual Micro Lab to zintegrowane środowisko projektowe (*Integrated Development Environment - IDE*), służące do uruchamiania systemów wykorzystujących mikrokontrolery ST62 lub AVR. Aktualnie obsługiwane są modele:

- w rodzinie ST6: ST6200, 01, 03, 08, 09, 10, 15, 20, 25,
- w rodzinie AVR: ATiny11, ATiny12, ATiny15, ATiny22, AT90S2343, AT90S2323, AT90S1200, AT90S2313, AT90S4433, AT90S4414, AT90S8515, AT90S4434, AT90S8535, ATmega8, ATmega16, ATmega161, ATmega162, ATmega32, ATmega64, ATmega128.

Najważniejszą - i trzeba przyznać - niezbyt często spotykaną w innych programach cechą VMLAB-a jest połączenie funkcji klasycznego środowiska IDE oraz symulatora najczęściej wykorzystywanych w systemach mikroprocesorowych układów peryferyjnych, w tym analogowych. Do dyspozycji mamy m.in.: rezystor, kondensator, diodę LED, klawiaturę 4x4, wzmacniacz operacyjny, komparator, przetworniki C/A i A/C, funktory logiczne, generatory różnych przebiegów, potencjometry, a także monitor I^2C, alfanumeryczny wyświetlacz LCD oraz szeregowy port komunikacyjny TTY. W symulacji można również używać wszystkich komponentów mikrokontrolera, jak na przykład: timery/liczniki, UART, komparator analogowy, watchdog itd. Mogą one pracować w systemie przerwań, jeśli tylko jest to możliwe w „prawdziwym" mikrokontrolerze.



Rys. 1. Uproszczony schemat układu eksperymentalnego

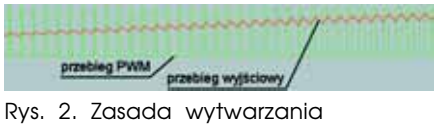
Atuty VMLAB-a

Najważniejszą cechą VMLAB-a jest połączenie funkcji klasycznego środowiska IDE oraz symulatora najczęściej wykorzystywanych w systemach mikroprocesorowych układów peryferyjnych, w tym analogowych.

Uruchamianie wirtualnego sprzętu ułatwia również wirtualny oscyloskop wielokanałowy. Przyjęcie takiej koncepcji pozwoliło stworzyć narzędzie, które powinno uprościć etap uruchamiania aplikacji, a przede wszystkim obniżyć jego koszty. Konstruktorzy nie tracą czasu na mozolne lutowanie układu, nie muszą także korzystać z drogich emulatorów sprzętowych ICE (*In Circuit Emulator*). Uniknięcie fizycznego montowania prototypu pozwala na rozbicie projektu na wiele zadań, które mogą być wykonywane jednocześnie przez cały zespół inżynierów. Do zera zmniejsza się ryzyko fizycznego uszkodzenia układu. Nie bez znaczenia jest fakt, że za pomocą jednego narzędzia można uruchamiać urządzenia bazujące na dwóch odmiennych rodzinach mikrokontrolerów. W dalszym opisie skoncentrujemy się na mikrokontrolerach AVR.

Rozpoznanie bojem

Najlepszą metodą zapoznania się z programem będzie zrealizowanie konkretnego projektu. Posłużymy się do tego przykładem, zawartym w pakiecie, z tym że dla lepszego zilustrowania współpracy VMLAB-a z kompilatorem C asemblerowa wersja oryginalna została przetłumaczona na ten właśnie język. Program przedstawiono na **list. 1**. Zanim przystąpimy do pracy, musimy zainstalować środowisko Visual Micro Lab. Można to zrobić, pobierając odpowiednie pliki ze strony *http://www.amtools.net*, wersja ewaluacyjna 3.6 jest także dostępna na naszym CD-ROM-ie (CD-EP5/2003B). Zainstalowanie wersji pełnej wymaga wypełnienia odpowiedniego formularza i dokonania opłaty. Po wykonaniu tych czynności, drogą elektroniczną dostarczane są kody dla wersji pełnej. Przed dokonaniem zakupu warto wypróbować VMLAB-a, instalując 21-dniową wersję ewaluacyjną. Jedynym jej ograniczeniem jest określony czas działania, po przekroczeniu którego nie będzie możliwe skompilowanie programu do postaci wykonywalnej. Program można rów-



Rys. 2. Zasada wytwarzania przebiegu sinusoidalnego z przebiegu PWM

List. 1

```
#define __AVR_AT90S2313__
#include <io.h>              // Most basic include files
#include <interrupt.h>       // Add the necessary ones
#include <signal.h>          // here
#include <progmem.h>

unsigned char angle;

unsigned char sine_tbl[128] __attribute__((progmem)) =
    {
     0,0,1,1,2,3,4,6,7,9,10,12,14,16,18,21,23,25,28,31,
     33,36,39,42,45,48,51,54,57,60,64,67,70,73,76,79,82,
     85,88,91,94,96,99,102,104,106,109,111,113,115,117,118,
     120,121,123,124,125,126,126,127,127,127,127,127,127,
     127,126,126,125,124,123,121,120,118,117,115,113,111,
     109,106,104,102,99,96,94,91,88,85,82,79,76,73,70,67,
     64,60,57,54,51,48,45,42,39,36,33,31,28,25,23,21,18,16,
     14,12,10,9,7,6,4,3,2,1,1,0,0,0,0,0
    };

SIGNAL(SIG_OVERFLOW1)
{
 unsigned char temp;

 angle=(angle)&0x7f;
 temp=PRG_RDB(&sine_tbl[angle++]);
 OCR1H=0;
 OCR1L=temp;
}

int main(void)
{
 DDRB=0x08;
 sbi(TIMSK,TOIE1);
 TCCR1A=1<<PWM10 | 1<<COM1A1;
 TCCR1B=1<<CS10;
 angle=0;
 sei();
 while(1);
}
```

nież zobaczyć „w akcji" na filmie *avi* zamieszczonym na CD-EP5/2003B.

Instalator zakłada na dysku domyślny katalog *c:\VMLAB*, w którym umieszcza wszystkie niezbędne pliki. Visual Micro Lab zawiera w sobie asembler obsługiwanych mikrokontrolerów, umożliwia również korzystanie z kompilatora GNU C (GCC) (tylko dla AVR-ów) lub dowolnego innego kompilatora generującego pliki COFF. Kompilatory muszą być jednak zainstalowane na komputerze niezależnie od VMLAB-a. Po wykonaniu wszystkich czynności niezbędnych do przygotowania sobie środowiska możemy już przystąpić do prawdziwej pracy inżynierskiej. Wracamy zatem do projektu, który będziemy wspólnie tworzyć. Będzie to generator przebiegu sinusoidalnego, wykorzystujący *Timer 1* pracujący w trybie modulatora PWM.

Rys. 3. Okno kreacji projektu

Na **rys. 1** przedstawiono uproszczony schemat ideowy układu eksperymentalnego. Zasada działania jest następująca. Timer 1 generuje przebieg PWM o zmienianym w dość specyficzny sposób współczyniku wypełnienia (**rys. 2**). Do wyjścia OC1 (PB3) dołączono układ całkujący składający się z rezystora R2 i kondensatora C1. W wyniku działania tego układu na wyjściu *FIL_OUT*, przy odpowiednim doborze wartości elementów R2 i C1, uzyskuje się przebieg sinusoidalny. Wartości kolejnych próbek sinusoidy są zawarte w tablicy *sine_tbl*. Zmieniając jej poszczególne elementy w niemal natychmiastowy sposób, bez konieczności programowania rzeczywistego mikrokontrolera, można zaobserwować skutek na wyjściu układu. Podobnie zresztą będziemy badać przebiegi wyjściowe dla różnych wartości elementów RC. Wartość próbki jest pobierana w przerwaniu *Timera 1*, generowanego po jego przepełnieniu.

Mając podstawy teoretyczne, możemy przystąpić do dzieła. Otwieramy nowy projekt, wybierając polecenie *Project->New project*. Pojawia się okno kreatora projektu (**rys. 3**), w którym

Rys. 4. Plik *makefile* generowany przez VMLAB

należy wypełnić odpowiednie pola. Po pierwsze: nadajemy nazwę np. *sinwavec.prj*, po drugie: wybieramy typ mikrokontrolera, np. AT90S2313, po trzecie określamy, czy będzie to program pisany w asemblerze, czy w języku C. Przyjmujemy wersję GNU C i zgadzamy się na automatyczne wygenerowanie pliku *makefile*, co z pewnością ucieszy wielu z nas. Na koniec wreszcie decydujemy o tym, jakie pliki źródłowe będą wchodziły do projektu (przycisk *Add this to list*). W naszym przykładzie będzie to tylko jeden plik. Akceptujemy proponowaną nazwę *sinewave.c*, na co program odpowiada, że plikiem wynikowym będzie *sinewave.hex*. Ustawienia akceptujemy naciśnięciem OK.

Teraz ukazują się liczne okna programu, z czego dwa są najważniejsze na tym etapie. Pierwsze z nich zawiera treść pliku *makefile*. Można go ręcznie wyedytować, jeśli proponowane ustawienia są nieodpowiednie. W szczególności, często będą zapewne zmieniane opcje optymalizacji kompilatora (pozycja *#compiler flags*). Jak widać na **rys. 4**, okno ma jeszcze zakładkę *sinwavec.c*, w której będziemy tworzyć wersję źródłową, ale to za chwilę. Wcześniej zajmiemy się naszym wirtualnym sprzętem. W tym celu odszukujemy tzw. okno projektu, np. wybierając polecenia *View->Project File*. Są już tam pewne dane przeniesione z pliku *makefile* oraz dodatkowe

Rys. 5. Okno projektu

Rys. 6. Pasek narzędziowy programu Visual Micro Lab

informacje o montowanym układzie. Proponuję zmienić częstotliwość rezonatora kwarcowego na 8 MHz - do weryfikacji układu dysponowałem akurat taką wersją. W polu CLOCK należy więc wpisać parametr *8meg*.

Teraz pozostało jeszcze dołączenie układu całkującego. Nie sposób tu omówić wszystkich możliwości, jakie daje nam w tym względzie program. Są one świetnie opisane wraz z przykładami w pomocy, niestety tylko w wersji angielskiej. Nasz układ całkujący będzie miał postać:

```
R2 PB3 fil_out 100K
;R2 dołączony do PB3
;i do wyjścia fil_out
C1 fil_out vss 10n
;C1 dołączony do wyjścia
;fil_out i masy
```

Ponadto warto dołączyć oscyloskop do końcówki PB3 (przebieg PWM) i do wyjścia fil_out:

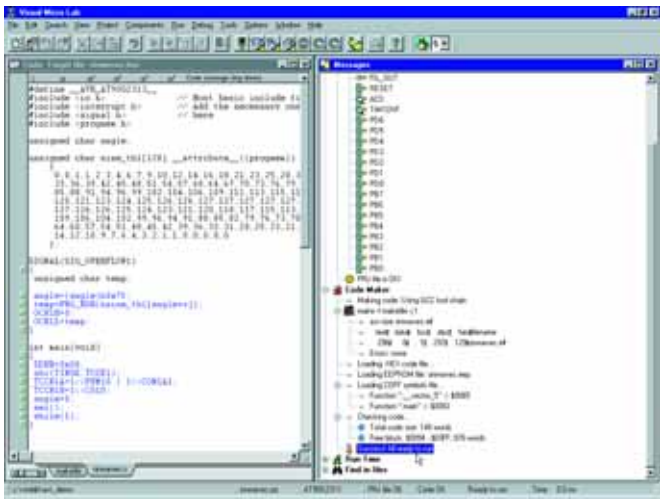
```
.plot v(PB3)
v(fil_out);podłączenie
oscyloskopu
```

Zawartość całego pliku projektu przedstawiono na **rys. 5**. Po wykonaniu wszystkich czynności edycyjnych okno to można zamknąć. Teraz możemy przystąpić do pisania wersji źródłowej naszego programu. Zaletą VMLAB-a jest możliwość zdefiniowania pięciu różnych organizacji ekranu. Jest to bardzo przydatna właściwość, ponieważ rozmieszczenie wszystkich okien programu nawet na monitorze 17-calowym, przy rozdzielczości 1024x768 jest prawie niemożliwe. Zapamiętywanie widoków pozwoli szybko przechodzić pomiędzy ekranami zawierającymi istotne w danym momencie informacje. Widoki zmienia się, wybierając odpowiednią pozycję z rozwijanej listy umieszczonej na końcu paska narzędziowego (**rys. 6**). Pierwsza faza programu najczęściej będzie polegała na sięganiu do źródła, kompilowaniu programu, przeglądaniu komunikatów kompilatora i ponownej edycji. Warto więc w jednym z widoków podzielić ekran tylko na dwa okna: edycyjne i komunikatów. Zapewni nam to dużą czytelność i komfort podczas wstępnych prac z programem. Po doprowadzeniu tekstu programu do stanu umożliwiającego kompilację, uruchamiamy ją, wybierając polecenie *Project->Build*. Można też użyć klawisza F9 lub ikony *Build* (środkowa ikona na pasku narzędziowym). W trakcie kompilacji w oknie *Messages* pojawiają się wszystkie komunikaty generowane przez kompilator. Jeśli wystąpi choćby jeden błąd, kursor edycyjny zostanie ustawiony w podejrzanym o to miejscu. Czasami jednak faktyczna nieprawidłowość może mieć miejsce gdzie indziej. Na przykład brak średnika kończącego linię programu C spowoduje wystąpienie błędu dopiero w dalszej części. Wszystkie miejsca, na które trzeba zwrócić uwagę, są sygnalizowane wykrzyknikami umieszczonymi przed treścią komunikatu. Kolor żółty oznacza, że jest to tylko ostrzeżenie, program będzie działał. Przykładowo VMLAB ostrzega w ten sposób przed zadeklarowaniem zmiennej, która nie jest używana dalej w żadnym miejscu programu. Gorzej, gdy wykrzyknik ma kolor czerwony. W takiej sytuacji kompilator nie tworzy pliku wykonywalnego, a więc uruchomienie programu nie będzie możliwe. Tego typu błędy trzeba bezwzględnie poprawić. Czasami zdarza się, że jedna nieprawidłowość w źródle powoduje wygenerowanie całej listy błędów. Ich eliminacja jest tym, co programiści lubią najbardziej. Niestety wszystko co dobre szybko się kończy i wreszcie pojawia się oczekiwany mimo wszystko komunikat *Success! All ready to run* (**rys. 7**).

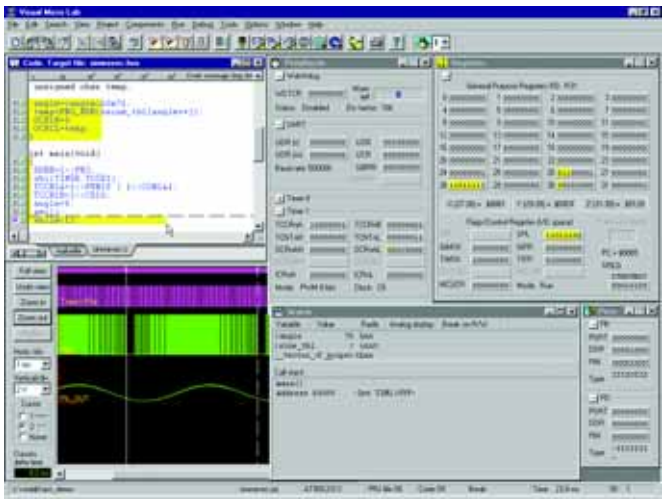
Od tego momentu rozpoczyna się kolejna faza uruchamiania aplikacji - symulacja. Teraz możemy stworzyć sobie kolejny ekran roboczy, który będzie zawierał przydatne do symulacji składniki - przykład pokazano na **rys. 8**. Na pewno warto umieścić na nim choćby niewielkie okno źródła programu. Wykorzystując krokowy tryb pracy, będzie w nim zaznaczana aktualnie wykonywana instrukcja na poziomie odpowiadającym wersji źródłowej (język C lub asembler). Jeśli program jest napisany w C, a chcemy wykonywać go na poziomie rozkazów mikrokontrolera, konieczne będzie dołożenie okna *Program memory* i uczynienie go aktywnym. Po lewej stronie każdej instrukcji widnieją malutkie przyciski ekranowe. Ich naciśnięcie powoduje wstawienie w danym miejscu programu pułapki (*breakpoint*). Aktywnych pułapek może być wiele.

Interesujący wybór

Twórcy VMLAB-a skupili się na dwóch rodzinach mikrokontrolerów:
- ST62: ST6200, 01, 03, 08, 09, 10, 15, 20, 25,
- AVR: ATiny11, ATiny12, ATiny15, ATiny22, AT90S2343, AT90S2323, AT90S1200, AT90S2313, AT90S4433, AT90S4414, AT90S8515, AT90S4434, AT90S8535, ATmega8, ATmega16, ATmega161, ATmega162, ATmega32, ATmega64, Atmega128.
Wybór jest - jak na Polskę - nietypowy, ale dowodzi dużej popularności ST62 w innych krajach europejskich.



Rys. 7. Przykładowy widok okna roboczego



Rys. 8. Program "Visual Micro Lab" podczas pracy

Rys. 9. Okno panelu sterującego

Podczas prac uruchomieniowych często zachodzi konieczność pomiaru czasu wykonywania się np. określonej procedury. Zadanie to nie jest najmocniejszą stroną VMLAB-a, lepiej chyba robi to AVR Studio, jednak VMLAB ma w tym względzie pewną dość atrakcyjną cechę. Wyświetla on mianowicie w oknie programu graficzną interpretację „czasu przebywania" jednostki centralnej w określonym miejscu programu. Są to żółte paski nałożone na poszczególne instrukcje. Trzeba jednak pamiętać, że skala czasu jest logarytmiczna! Trudno mówić o precyzji takiego pomiaru, ale w niektórych sytuacjach informacja podana w ten sposób może okazać się bardzo przydatna.

Nie sposób wyobrazić sobie uruchamiania programu bez możliwości podglądania zasobów mikrokontrolera. W Visual Micro Lab oczywiście taką możliwość mamy i to w dość atrakcyjnej formie (**rys. 8**). Poszczególne komponenty mikrokontrolera są zgrupowane w kilku niezależnych oknach, np.: rejestry, pamięć programu, pamięć danych SRAM, pamięć EEPROM, porty oraz UART, timery, watchdog. Zawartość każdego z nich może być wyświetlana w postaci binarnej, hexadecymalnej, całkowitoliczbowej, a nawet w kodach ASCII. Na wszystkie pola jest nałożony półprzezroczysty pasek graficzny, który w postaci analogowej pokazuje stan danego komponentu. Każdy z rejestrów może być w dowolnej chwili zmieniony. Niestety nie dotyczy to licznika programu, co oznacza, że program może być wykonany tylko w takiej kolejności, jak w układzie rzeczywistym. Użytkownik ma też możliwość podglądania zmiennych programu. Bywają z tym niestety problemy, zmienne te nie zawsze są dostępne. Nie można też modyfikować w prosty sposób ich wartości. Szkoda. Pewnym wyjściem z sytuacji jest jedynie podejrzenie miejsca przechowywania (rejestry, pamięć RAM) i edycja na tym poziomie. Specyficznym oknem VMLAB-a jest panel tzw. sterujący. Zawiera on zewnętrzne urządzenia peryferyjne wykorzystywane w aplikacji (**rys. 9**). Może to być klawiatura 4x4, alfanumeryczny wyświetlacz LCD, wirtualny terminal TTY, diody świecące, potencjometry. Ich parametry konfiguruje się w oknie projektu.

Popatrzmy teraz, jak to wszystko działa w praktyce. Program jest już wyczyszczony ze wszystkich błędów, możemy go uruchomić. Przygotowujemy sobie ekran np. tak, jak rys. 8 i naciskamy ikonkę z semaforem lub klawisz F5. Nasz wirtualny system zaczyna działać. Na oscyloskopie pojawiają się przebiegi generowane przez mikrokontroler. Aby przyjrzeć się nieco dokładniej, jak to robi, zatrzymujemy program, naciskając ikonkę ze znakiem Stop i kolejne rozkazy wykonujemy krokowo. Tu okazuje się, że właściwa część programu to martwa pętla, w której mikrokontroler właściwie przebywa przez większość czasu. Jest to bardzo wyraźnie widoczne na wspomnianych wcześniej paskach pod instrukcjami programu. CPU jedynie na krótkie okresy przechodzi do procedury obsługi przerwania, w której następuje modyfikacja parametrów generowanego przebiegu PWM. Krokowe wykonywanie programu do momentu, aż zostanie wygenerowane przerwanie mogłoby trwać bardzo długo. Dlatego wygodniej będzie założyć pułapkę w procedurze przerwania i puścić program w trybie ciągłym. Po zatrzymaniu się na pułapce, kontynuujemy pracę krokowo. Wykonując program, warto przyjrzeć się, jak są mieniane zawartości rejestrów np. OCR1nAAH i OCR-nAL (okno *Peripherals->Timer 1*). Przebieg na oscyloskopie może być dość swobodnie skalowany, umożliwiając ogarnięcie całości w większym czasie, a także przyjrzenie się detalom z rozdzielczością 100 ns/działkę. Ustawiając na oscylogramie dwa kursory, można dokonać prostego pomiaru czasu. Ciekawym doświadczeniem może być np. zbadanie wpływu wartości elementów R2, C1 na kształt generowanego przebiegu. Odpowiednie modyfikacje przeprowadzamy w oknie projektu.

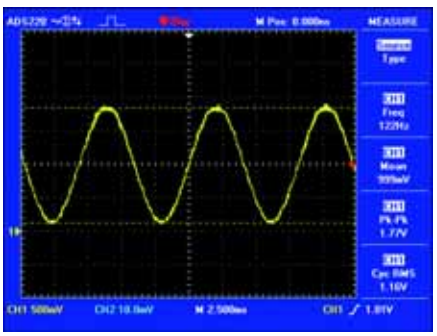
Także dla fanów AVR-GCC

Naturalnym "partnerem" VMLAB-a jest kompilator AVR-GCC. Dzięki temu połączeniu fani tego kompilatora uzyskują łatwy dostęp do symulacji pisanych programów.

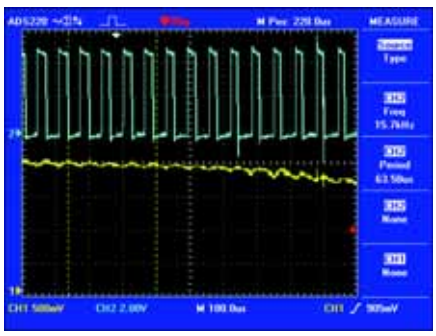
To jest Twoja chwila prawdy

Można powiedzieć, że wszystko, co robiliśmy do tej pory, to była tylko zabawa. To dość przyjemne wrażenie, gdy opracowując poważne projekty, wykorzystując profesjonalne narzędzie, czerpiemy taką radość z pracy. Teraz jednak przechodzimy do końcowego etapu projektu. Odkładamy program, bierzemy lutownicę do ręki i... z drżeniem serca oczekujemy końcowego rezultatu. Nasz projekt nie był zbyt skomplikowany, trzeba to uczciwie przyznać. Ale zdarzało się wielokrotnie z innymi narzędziami, że nawet w takich przypadkach wyniki symulacji bywały rażąco odmienne od rzeczywistych. Mając akurat pod ręką zestaw, na którym mógłbym zweryfikować działanie opisywanego wyżej projektu, nie mogłem sobie tego odmówić. Oscylogramy układu rzeczywistego zdjęte oscyloskopem cyfrowym są przedstawione na **rys. 10** i **11**. Rezultat porównania jest niemal szokujący. Okazało się, że symulacja dała bardzo wierne wyniki zarówno pod względem jakościowym, jak i ilościowym. Oczywiście taki test nie może być podstawą do wydania całkowicie wiarygodnej opinii o programie. Mimo to oceniam program bardzo wysoko, a wykorzystuję go już od dość dawna. Nie jest on pozbawiony pewnych błędów, ale jak widać rozwija się w sposób ciągły, dając nadzieję na stopniowe eliminowanie błędów. Bardzo dużą zaletą jest natomiast dość dobra integracja z kompilatorem GCC oraz dużo lepsze symulowanie wewnętrznych bloków funkcjonalnych mikrokontrolera, niż robi to AVR Studio. Zachęcam do wypróbowania.

Jarosław Doliński, AVT
jaroslaw.dolinski@ep.com.pl



Rys. 10. Przebieg generowany w układzie rzeczywistym



Rys. 11. Przebieg rzeczywisty w powiększeniu