ATmega w środowisku Microchip

Programowanie 8 bitowych mikrokontrolerów AVR

Przejęcie Atmela przez Microchip spowodowało, że pojawiło się wiele obaw o rozwój mikrokontrolerów i narzędzi programowych, na przykład – Atmel Studio. Te obawy przynajmniej na razie się nie potwierdziły, ale Microchip powoli zaczyna wprowadzać zmiany w mikrokontrolerach AVR. Jednym z takich działań było wbudowywanie bloków funkcjonalnych CIP, działających niezależnie od rdzenia. To rozwiązanie stosowane z powodzeniem w PIC16F jest na pewno dobrym posunięciem, zwiększającym możliwości prostych jednostek. W parze z modyfikacjami sprzętowymi jednak musi iść ewolucja narzędzi programowych. Mimo tego, że Atmel Studio jest nadal oferowane i rozwijane, to chyba w dłuższej perspektywie trudno będzie wymagać od Microchipa, aby wspierał dwie niezależne platformy projektowe. Pierwszym oznaką jest możliwość programowania i debugowania mikrokontrolerów z rodziny Atmega 0+ w środowisku MPLAB X IDE i z użyciem kompilatora MPLAB XC8 w wersji od V2.05 Ponadto, MPLAB XC8 od wersji V2.05 używa kompilatora AVR GCC w projektach dla mikrokontrolerów AVR, więc za pomocą XC8 można kompilować projekty dla każdego z mikrokontrolerów AVR. Środowisko projektowe IDE MPLAB X ma też możliwość importowania projektów tworzonych w Atmel Studio. A wszystko po to, aby jak najszybciej i przy minimalnym nakładzie pracy można było pracować w MPLAB X, co raczej nie jest przypadkowe.

Przez wiele lat rozwoju mikrokontrolerów jednostki 8-bitowe były najbardziej popularne i najchętniej stosowane w układach sterowania. Wynikało to z możliwości technologicznych, dostępności narzędzi programistycznych i w konsekwencji ceny. Rdzenie mikrokontrolerów 8-bitowych z czasów największej świetności dzisiaj są archaiczne i trudno w nich znaleźć rozwiązania pozwalające na optymalne programowanie z użyciem języka C. Trzeba jednak pamiętać, że przez bardzo długi czas do programowania używano kompilatora asemblera, a programista musiał dokładnie znać budowę rdzenia i listę rozkazów.

W miarę upływu czasu sytuacja ulegała zmianie. Na rynku było dostępnych coraz więcej kompilatorów języka C za umiarkowaną cenę lub wręcz za darmo. Ostatecznie to dostęp do narzędzi, takich jak kompilator C, programator/debugger i środowisko projektowe mogą w mniejszym lub większym stopniu decydować o wyborze tej czy innej rodziny mikrokontrolerów.

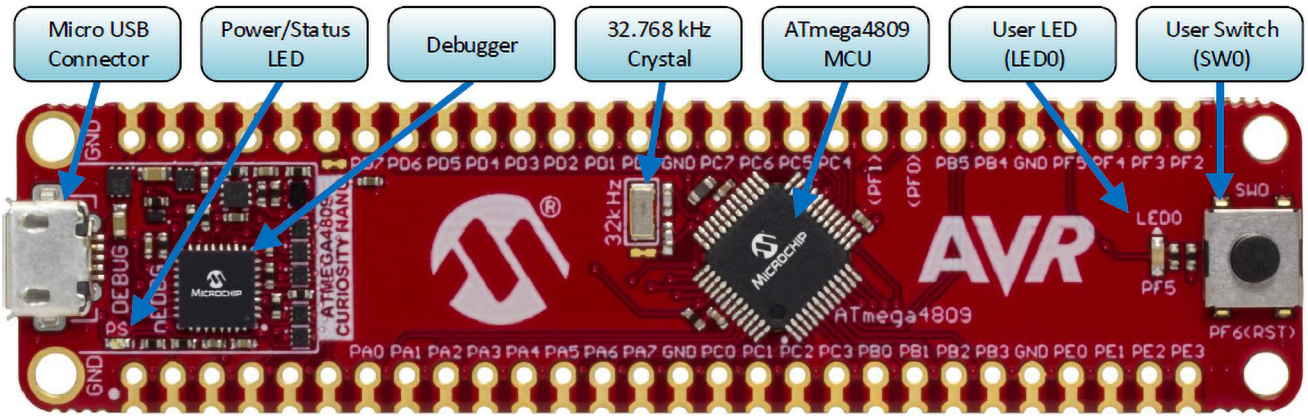
Co dalej z AVR i Atmel Studio?

Przejęcie firmy Atmel przez Microchipa w styczniu 2016 roku to jedno z ważniejszych wydarzeń w światku użytkowników mikrokontrolerów. W naszym kraju i nie tylko w nim, trwały spory o to, co jest lepsze – Atmel AVR czy Microchip PIC16F lub PIC18F. Za Atmelem przemawiały: dostępność tanich programatorów i bezpłatny kompilator AVR GCC. Dużą pomocą dla producenta w pozyskiwaniu nowych fanów był wykorzystywany przez entuzjastów interpreter Bascom i oczywiście Ardiuno oparte o mikrokontrolery AVR. Jednak kiedy przegląda się różne fora zagraniczne i czyta te czasopisma elektroniczne, które jeszcze pozostały na rynku, to okazuje się, że „atmele" nie wszędzie są najbardziej popularnymi 8-bitowcami i sporo uwagi poświęca się produktom Microchipa. Trudno bowiem nie zauważyć, że PICe mają sporo zalet: relatywnie dobre i bardzo dobre układy peryferyjne, dobre moduły ewaluacyjne, bardzo dobre środowisko projektowe MPLAB X i bezpłatne kompilatory dla wszystkich rodzin, w tym MPALB XC8 dla PIC16F i PIC18F. Co prawda, dotychczas w porównaniu z AVR GCC kompilator XC8 w wersji bezpłatnej dla mikrokontrolerów PIC ma wyłączoną lub ograniczoną optymalizację kodu, ale dla wielu zastosowań nie ma to większego znaczenia. Listę zalet i wad można by długo przytaczać, ale mimo tego, że przy dzisiejszym stanie techniki dostępne są w zbliżonej cenie dużo nowocześniejsze, 32-bitowe jednostki, to 8-bitowce nadal są popularne i chętnie stosowane do realizacji mniej złożonych zadań.

Wspomniane przejęcie Atmela spowodowało, że pojawiło się wiele obaw o rozwój samych mikrokontrolerów i narzędzi firmowych, na przykład środowiska programistycznego Atmel Studio. Te obawy przynajmniej na razie się nie potwierdziły, ale Microchip zaczyna powoli wywierać wpływ na budowę samych mikrokontrolerów AVR. Jednym z takich działań jest wbudowywanie peryferii działających niezależnie od rdzenia (CIP). To rozwiązanie znane i stosowane z powodzeniem w PIC16F jest na pewno dobrym posunięciem, zwiększającym możliwości układów. W parze z modyfikacjami sprzętowymi musi jednak iść ewolucja narzędzi programowych. Mimo tego, że Atmel Studio jest nadal oferowane i chyba stale rozwijane, to chyba w dłuższej perspektywie trudno będzie wymagać od Microchipa, aby wspierał dwie niezależne platformy projektowe. Pierwszym zwiastunem tego trendu jest możliwość programowania i debugowania mikrokontrolerów z rodziny ATmega 0+ w środowisku MPLAB X IDE i z użyciem kompilatora MPLAB XC8 w wersji od V2.05

Jedna **z ważniejszych wiadomości dla użytkowników mikrokontrolerów AVR jest t**aka, że **środowisko** MPLAB XC8 od wersji V2.05 używa kompilatora AVR GCC w projektach dla mikrokontrolerów AVR. Dzięki temu, za pomocą XC8 można kompilować projekty dla każdego z mikrokontrolerów AVR, które były wspierane przez AVR GCC. Środowisko projektowe IDE MPLAB X ma **również** możliwość importowania projektów tworzonych w Atmel Studio z poziomu paska narzędziowego (File → Import → Atmel Studio Project). Wszystko po to, aby jak najszybciej i przy minimalnym nakładzie pracy można było pracować w MPLAB X.

Ponieważ jak na razie XC8 to w praktyce dwa oddzielne kompilatory, to projekty tworzone dla PIC16F i PIC18F będą się w niektórych aspektach różniły od projektów dla mikrokontrolerów AVR. Dlatego dla każdej z rodzin przygotowano oddzielne instrukcje użytkownika dla tego kompilatora.



Sygnał debugger'a	sygnał docelowy UPDI	opis
ID		linie ID – dla rozszerzeń
CDC TX	UART TX	linia USB CDC TX
CDC RX	UARTRX	linia USB CDC RX
DBG0	UPDI	linia danych debuggowania
DBG1	GPIO1	linia zegara debuggowania/DGI GPIO
DBG2	GPIO0	linia DGI GPIO
DBG3	RESET	linia zerowania
NC		nie połączone
VBUS		napięcie VBUS dla urzadzeń zewnętrznych
VOFF		wejście wyłaczenia zasilania
VTG		napięcie zasilania
GND		Masa

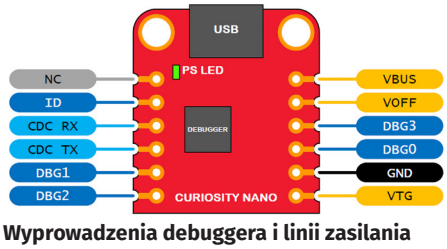
Rysunek 1. Moduł Curiosity Nano ATmega4809

Zmiana przyzwyczajeń nie jest tym, co lubimy robić. Przechodzenie z Atmel Studio na IDE MPLAB X mimo sporych ułatwień na pewno nie każdemu się spodoba, Jednak system wsparcia projektanta przez Microchip ma sporo zalet. Po pierwsze, programator/debuger. Jest dostępny niedrogi, ale bardzo dobry programator/debuger MPLAB SNAP i również tani, ale już w obudowie – PICkit-4. MPLAB SNAP można kupić za około 80 złotych brutto. Jest on w stanie zaprogramować i debuggować wszystkie rodziny mikrokontrolerów PIC i sporo AVR. PICKit4 – programator/debugger w obudowie kosztuje około 230 złotych brutto. Należy się spodziewać, że w przyszłości te programatory będą mogły programować i debugować wszystkie oferowane przez Microchipa AVR bez ponoszenia dodatkowych kosztów (firmware programatora jest wgrywane automatycznie z poziomu MPLAB X IDE). Inną możliwością może być wykorzystanie wbudowanego w tanie moduły ewaluacyjne programatora/debuggera. Jest jeszcze jedna ważna zaleta przemawiająca na korzyść IDE MPLAB X. Jest to, jak zobaczymy dalej, świetne narzędzie konfiguratora peryferii w postaci wtyczki MCC. Co prawda, MCC ma odpowiednika w postaci Atmel START, ale za to jest zintegrowane z IDE w postaci tzw. wtyczki (plug-in).

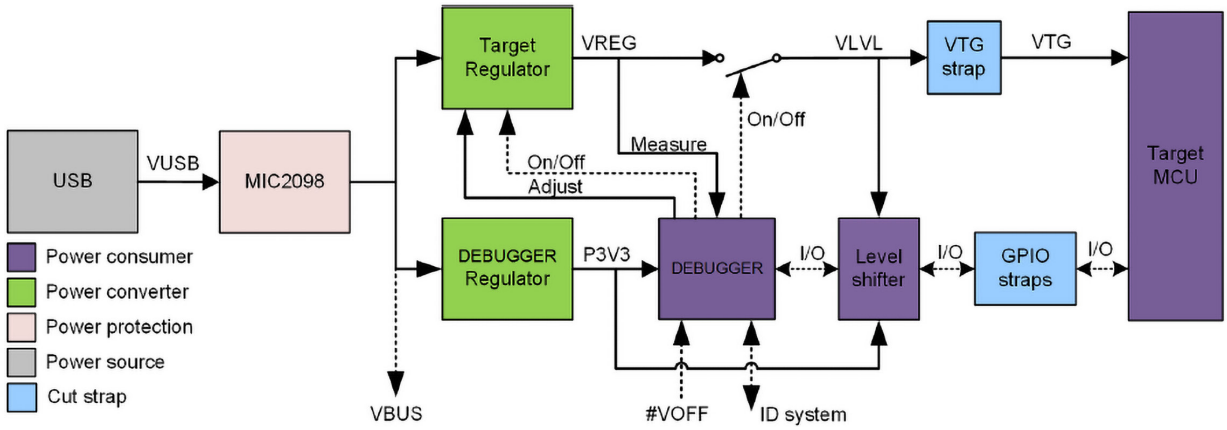
AVR i MPLAB

O tym, że AVR można programować i debugować z poziomu MPLAB już wiemy. Popatrzmy teraz, jak się do tego zabrać. Pierwszym krokiem może być zaopatrzenie się w tani (koszt zakupu to około 9 euro w sklepie internetowym Microchip) moduł ewaluacyjny Curiosity Nano ATmega4809 (**rysunek 1**). Moduł zmontowano na wąskiej płytce drukowanej z umieszczonymi po obu stronach punktami lutowniczymi o rastrze 2,54 mm, do których doprowadzono wyprowadzenia mikrokontrolera. Płytkę modułu można zamontować na płycie bazowej/ewaluacyjnej techniką montażu powierzchniowego, lub po wlutowaniu listew goldpinów metodą montażu przewlekanego. Na płytce zostały zamontowane: mikrokontroler ATMega4809 MFR, jeden żółty LED, przycisk, oscylator 32768 Hz, układ pełniący funkcję programatora i debuggera, wirtualnego portu COM i dwukanałowego analizatora logicznego (debugger może współpracować z MPLAB X IDE i z ATmel Studio 7), złącze mikro USB, stabilizator LDO MIC5353 regulowany przez układ debuggera w zakresie od 1,8 V do 5,1 V (górna granica jest wyznaczana przez napięcie portu USB).

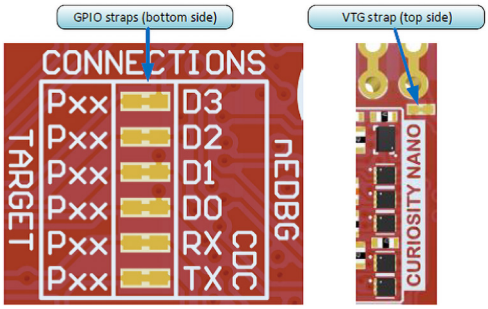
Cały układ: debugger i mikrokontroler są zasilane z portu USB. Z punktu widzenia magistrali USB debugger jest urządzeniem typu composite i zawiera kilka interfejsów: debuggera, pamięci masowej, Data Gateway i portu wirtualnego COM (CDC). Port wirtualny COM jest połączony z interfejsem UART mikrokontrolera ATmega4809. Daje to możliwość łatwego komunikowania się uruchamianej aplikacji z terminalami znakowymi przez port USB. VCOM ma jednak ograniczenia: prędkość transmisji od 1200 baud do 500 kbaud, długość słowa tylko 8 bitów i brak wsparcia dla sprzętowej kontroli przepływu. Na **rysunku 2** pokazano wyprowadzenia związane z USB Composite układu debuggera. Cały układ jest zasilany z portu USB napięciem z zakresu 4,4…5,25 V . Do zasilania mikrokontrolera pełniącego



Rysunek 2. Wyprowadzenia debuggera i linii zasilania



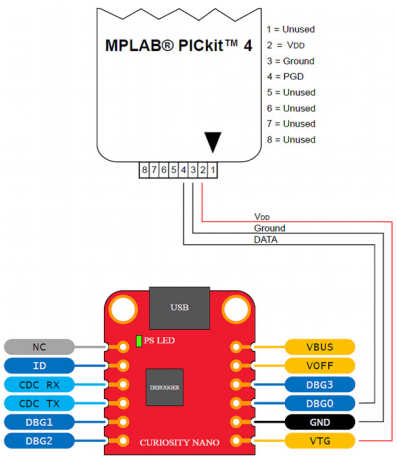
Rysunek 3. Schemat blokowy układu zasilania



Rysunek 4. Odłączenie wbudowanego debuggera od mikrokontrolera

wszystkie funkcje debuggera jest przeznaczony stabilizator LDO o napięciu wyjściowym +3,3 V. ATmega4809 jest zasilany kolejnym stabilizatorem LDO, dostarczającym napięcia programowanego przez układ debuggera – od +1,8 V do VBUS. Linie mikrokontrolera są połączone z liniami debuggera, więc jest konieczne stosowanie konwertera poziomów logicznych. Schemat blokowy układu zasilania pokazano na **rysunku 3**. Układ debuggera można odłączyć od zamontowanego na płytce mikrokontrolera przez przecięcie połączeń na płytce modułu (**rysunek 4**) i wykorzystać go do programowania/debuggowania innego mikrokontrolera. Odłączenie (przecięcie ścieżek) pozwala też na użycie zewnętrznego programatora, na przykład, wspomnianego już PICkit-4 (**rysunek 5**).

Na **rysunku 6** pokazano wszystkie wyprowadzenia modułu z opisami funkcji: portów, wyjść alternatywnych układów peryferyjnych i zasilania. Moduł łączy się z komputerem za pomocą kabla USB ze złączem USB micro. System Windows powinien automatycznie wykryć urządzenie klasy „USB composite", wyszukać i zainstalować odpowiednie sterowniki. Po ich zainstalowaniu moduł jest widoczny jako dysk pamięci masowej o nazwie „CURIOSITY". Jego zawartość to kilka plików, w tym dwa tekstowe KIT-INFO i STATUS. W pliku



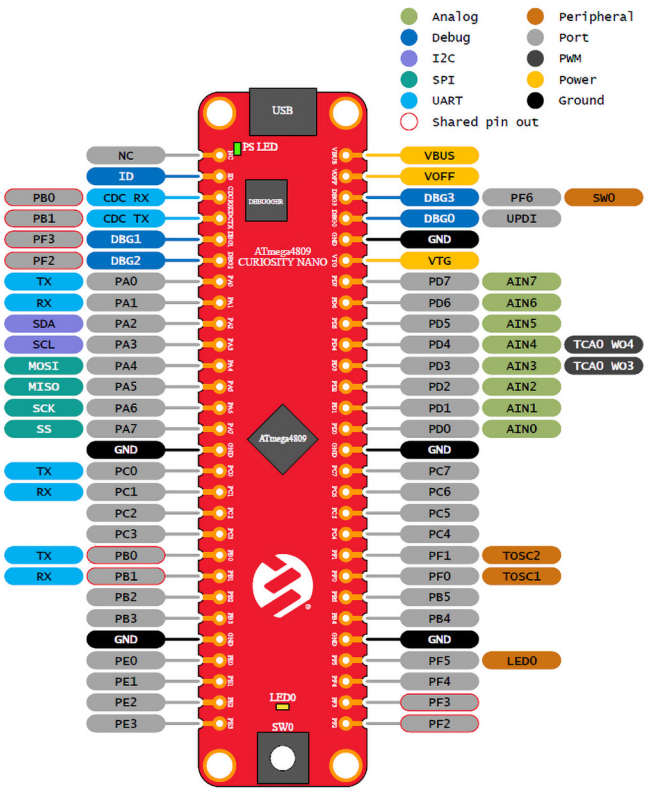
Rysunek 5. Programowanie za pomocą PICkit-4

KIT-IFO są umieszczone informacje o dacie firmware, typie płytki, numerze seryjnym i typie mikrokontrolera (**rysunek 7**).

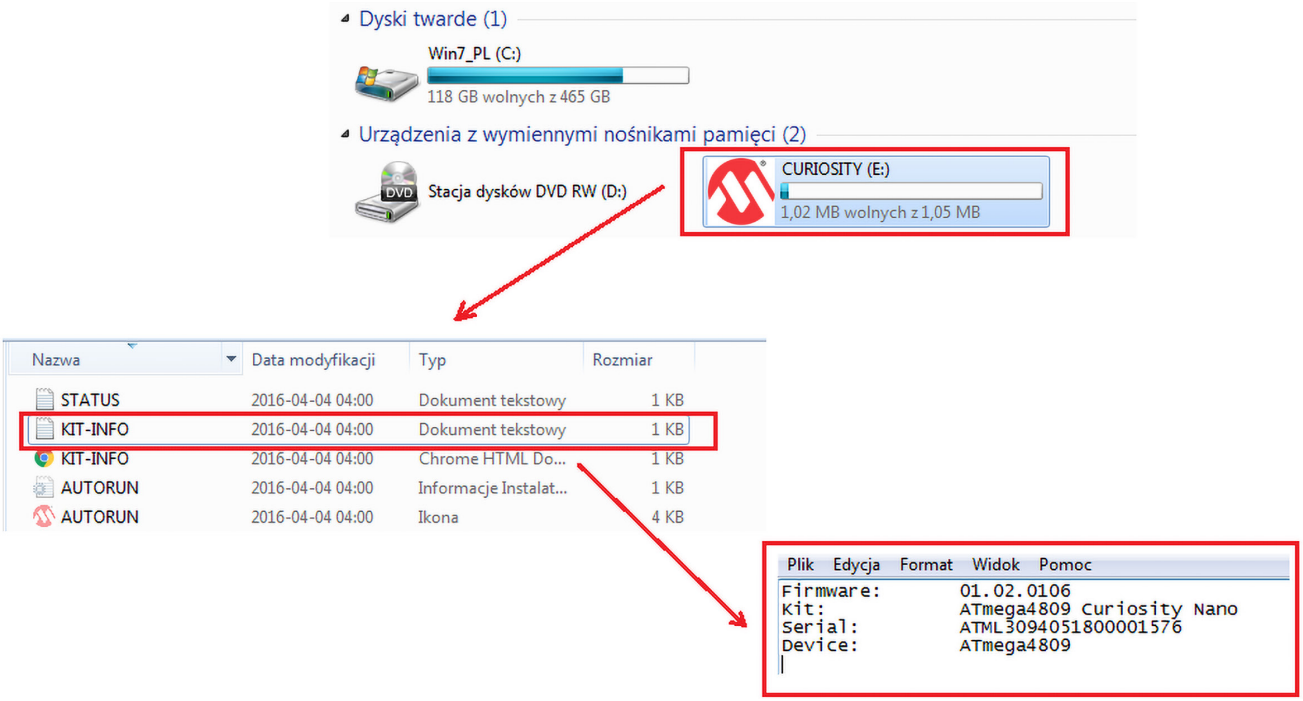
Teraz możemy przystąpić do tworzenia projektu w środowisku IDE MPLAB X. Będzie nam potrzebny MPLAB X w wersji V5.05 lub nowszej i kompilator w wersji od V2.05. Zaczynamy standardowo – od wyboru projektu Microchip Embeded i StandAlone Project, jak na **rysunku 8**. W kolejnym kroku wybieramy rodzinę mikrokontrolerów i konkretny typ ATmega4809 (**rysunek 9**).

W czasie pisania artykułu wszystkie te narzędzia nie były jeszcze dokładnie przetestowane i wsparcie dla projektów z mikrokontrolerami AVR było dostępne w wersji przedprodukcyjnej Beta.

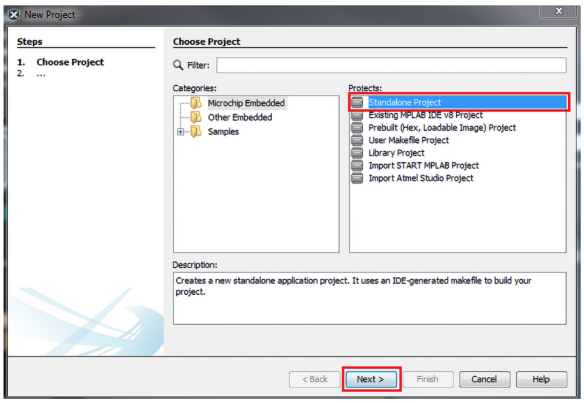
W oknie wyboru narzędzi programowych na liście *Alternate Tools* wybieramy zgłoszony debugger modułu nEDBG (**rysunek 10**). Ten debugger, podobnie jak PICKit-4 i SNAP, jest oznaczony żółtym markerem (kółeczkiem). Oznacza to, że ten komponent nie został jeszcze w pełni przetestowany, ale może być używany z ograniczonym zaufaniem. Po przetestowaniu i zatwierdzeniu testów marker powinien się zmienić na zielony. Narzędzie, które nie współpracuje z wybranym mikrokontrolerem jest oznaczane na czerwono. Następnie (**rysunek 11**) wybieramy w kolejnym oknie kompilator XC8 w wersji z żółtym markerem (v2.05). Na sam koniec kreator projektu prosi



Rysunek 6. Wyprowadzenia modułu Curiosity Nano ATmega4809



Rysunek 7. Debugger jako dysk pamięci masowej w systemie Windows

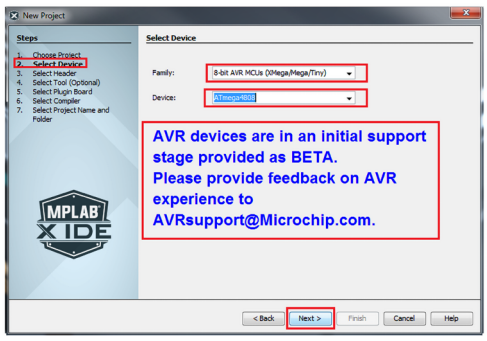


Rysunek 8. Wybór typu projektu

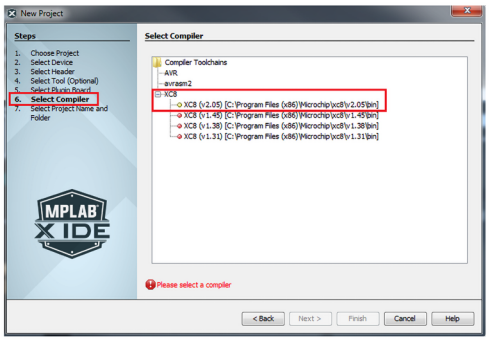
o podanie jego nazwy i ścieżki dostępu do katalogu, w którym będzie zapisany (**rysunek 12**).

Kreator projektu w MPLAB X nie generuje automatycznie szkieletu projektu z podstawowymi plikami, w tym z plikiem *main.c*. Programista musi albo sam sobie te pliki utworzyć w katalogu projektu, albo skorzystać z wtyczki konfiguratora MCC.

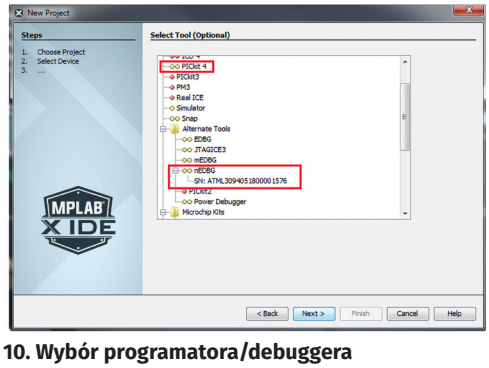
MCC to narzędzie pozwalające na konfigurowanie układów skojarzonych z rdzeniem (układ taktowania i inne), kontroler przerwań i bloki peryferyjne. Konfigurator wykorzystuje, na ile to, możliwe interfejs graficzny i dlatego konfiguracja jest o wiele łatwiejsza i mniej podatna na błędy. Ja go bardzo chętnie używam do konfigurowania mikrokontrolerów PIC. W czasie mojego testu z ATmega4809, MCC działał poprawnie, w zakresie, w jakim miałem okazję wykorzystać go do bardzo projektu demonstracyjnego. Konfiguratory są szeroko



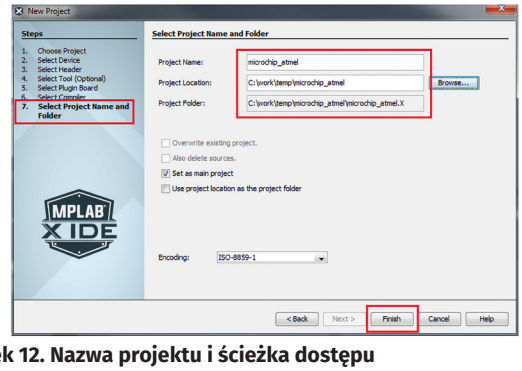
Rysunek 9. Wybór mikrokontrolera do projektu



Rysunek 11. Wybór kompilatora XC8



Rysunek 10. Wybór programatora/debuggera



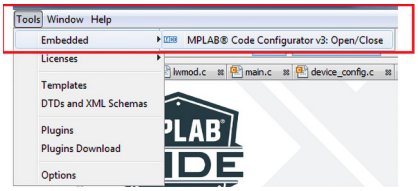
Rysunek 12. Nazwa projektu i ścieżka dostępu

stosowane dla bardziej rozbudowanych mikrokontrolerów i warto tu wspomnieć o STM32CubeMX dla STM32, czy konfiguratorze wbudowanym w IDE e2studio Renesasa. MCC jest uruchamiamy z poziomu paska narzędziowego *Tools* → *Embedded* (**rysunek 13**).

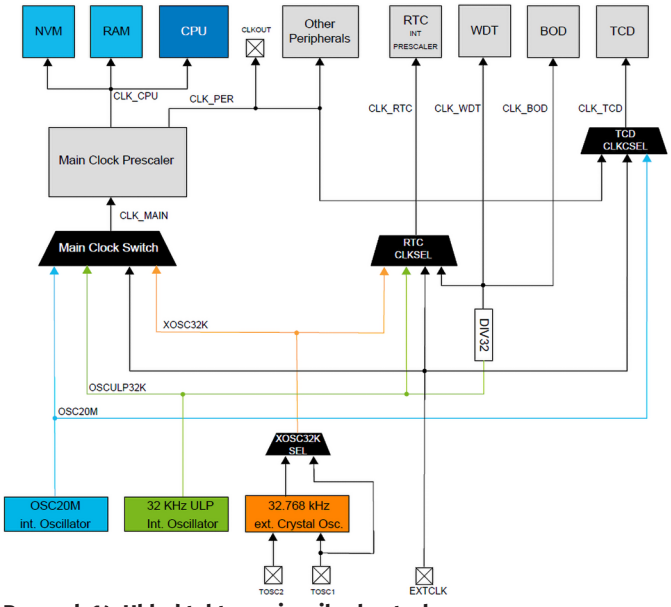
Okno główne składa się z dwóch okien: Project Resources i Device Resources. Zaczynamy od okna Project Resources. Są tam umieszczone trzy elementy:

1. **Interrupt Manager** przeznaczony do konfigurowania układu przerwań.
2. **Pin Module** przeznaczony do konfigurowania wyprowadzeń mikrokontrolera.
3. **System Module** przeznaczony do konfigurowania układów związanych z pracą mikrokontrolera.

obecnie produkowane mikrokontrolery, w tym również te relatywnie mało skomplikowane, z 8-bitowym rdzeniem, wymagają konfigurowania układów taktowania. Poza tym, trzeba włączyć (lub wyłączyć) i skonfigurować kilka ważnych bloków: detektor spadku napięcia zasilającego, watchdog, układ nadzoru taktowania itp. Zależnie od rozwiązania te konfiguracje są umieszczane w obszarze

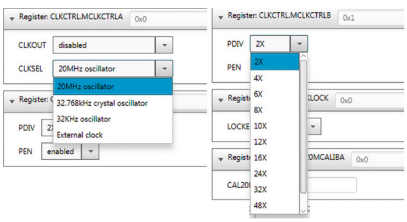


Rysunek 13. Uruchamianie wtyczki MCC



Rysunek 14. Układ taktowania mikrokontrolera

Rysunek 15. Ustawienia rejestru OSCCFG



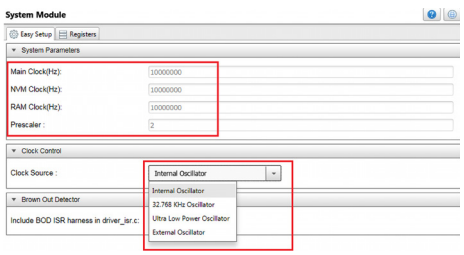
Rysunek 16. Ustawienie taktowania

bitów konfiguracyjnych (FUSE), i/lub są wykonywane programowo po zerowaniu mikrokontrolera.

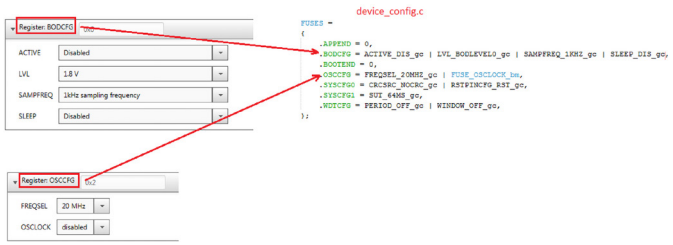
Mikrokontrolery AVR były znane z problemu zablokowania działania mikrokontrolera, kiedy użytkownik nieopatrznie źle ustawił bity konfiguracyjne. Mogło to być, na przykład, zaprogramowanie źródła zegara na zewnętrzny oscylator kwarcowy zamiast wewnętrznego RC. Przywrócenie kontroli nad mikrokontrolerem wymagało dołączenia zewnętrznego kwarcu lub przeprogramowania za pomocą programatora „wysokonapięciowego". Problem był tak znany, ze pojawiły się specjalizowane urządzenia typu „AVR Fusebit Doctor" pozwalające na przeprogramowanie obszaru pamięci konfiguracji. Przyznam, że zanim rozpocząłem jakiekolwiek testy próbowałem znaleźć opisy tego typu zagrożeń. Zablokowanie w taki sposób mikrokontrolera (jeżeli było możliwe) praktycznie dyskwalifikowało moduł do dalszego użycia. Do programowania/debuggowania naszego mikrokontrolera używane jest wejście UPDI. UPDI jest wejściem dedykowanym i nie jest współdzielone z inną funkcją mikrokontrolera. Według zapewnień kart katalogowych, nie możliwości zablokowania programatora/debuggera przez nieuważne zaprogramowanie jakiegoś bezpiecznika FUSE. Podobnie jest w wypadku taktowania. Projektanci mikrokontrolera wzięli sobie do serca problemy użytkowników. Układ taktowania ma do wyboru cztery źródła zegara:

1. Generator RC o częstotliwości 20 MHz.
2. Generator RC o częstotliwości 32 kHz.
3. Generator stabilizowany oscylatorem kwarcowym o częstotliwości 32,768 kHz.
4. Sygnał zegarowy z zewnętrznego generatora (wejście EXTCLK).

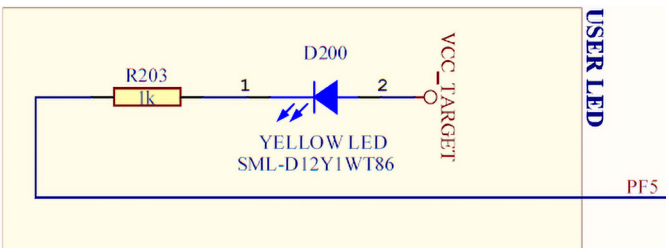
Po zerowaniu mikrokontrolera źródło zegara ustawia się na takie, które jest zaprogramowane w nieulotnej pamięci konfiguracyjnej (FUSEBIT). Może to być oscylator RC o częstotliwości 20 MHz lub 16 MHz. Nasz mikrokontroler ma wbudowany oscylator OSC20M o częstotliwości 20 MHz i po włączeniu zasilania jest źródłem taktowania z domyślnym preskalerem o wartości 6 (**rysunek 15**). Tu również nie da się zablokować mikrokontrolera ustawiając, na przykład, taktowanie z zewnętrznego źródła. Przełączenie na inne źródło



Rysunek 17. Zakładka Easy Setup okna System Module



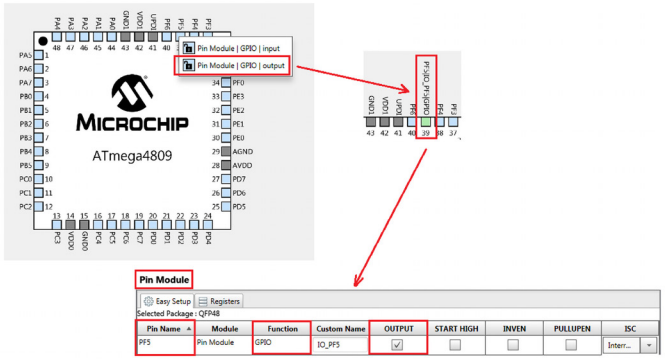
Rysunek 18. Ustawienia bitów konfiguracyjnych za pomocą MCC



Rysunek 19. Dołączenie diody LED na module

taktowania i zmiana podziału preskalera odbywa się programowo przez zapis rejestru CLKCTRL. Zapobiega to skutecznie zablokowaniu mikrokontrolera, bo po restarcie można go zmienić programowo. Wybór taktowania z wejścia sygnału zewnętrznego jest dodatkowo zabezpieczony. Jeżeli na wejściu EXTCLK nie pojawi się dostateczna liczba zboczy zegara, to nawet mimo zaprogramowania go jako źródła taktowania wewnętrzny układ się nie przełączy na sygnał z EXTCLK.

Zobaczmy jak można zegar skonfigurować w MCC. Zaczynamy od ustawienia bitów rejestrów pamięci konfiguracji (FUSE). Skupimy się tylko nad programowaniem zegara. Wszystkie inne konfiguracje dotyczące detektora spadku napięcia, watchdoga czy BOR zostały domyślnie wyłączone, bo do pierwszych testów nie będą potrzebne. W oknie system module w zakładce *Register* odnajdujemy rejestr OSCCFG i ustawiany tam FREQSEL=20 MHz i OSCLOCK wyłączony. W kolejnym kroku ustawiamy w rejestrze CLKCTRL docelowe źródło taktowania mikrokontrolera. Dla przypomnienia, może to być oscylator RC 20 MHz (OSC20M), oscylator RC 32 kHz, oscylator kwarcowy 32768 Hz lub zegar zewnętrzny. W tym samym rejestrze ustawiamy wartość preskalera PDIV od 2 do 48 z możliwością całkowitego wyłączenia (PEN=DISABLE). Zostało to pokazane na **rysunku 16**. Oscylator można wybrać w zakładce Easy Setup. Jest w niej również wyświetlana rzeczywista, ustawiona częstotliwość taktowania wynikająca z częstotliwości generatora zegarowego i ustawienia preskalera. Ja ustawiłem jako źródło zegara OSC20M i preskaler na wartość 2, aby mikrokontroler w czasie testów był taktowany częstotliwością 10 MHz. Zakładkę *Easy Setup* elementu *System Module* pokazano na **rysunku 17**. Na **rysunku 18** pokazano przykłady ustawień rejestrów konfiguracyjnych i fragment kodu źródłowego definiujący rejestry konfiguracyjne mikrokontrolera pochodzący z pliku *device_config.c* wygenerowanego przez MCC na podstawie tych ustawień. Na **listingu 1** zamieszczono definicję bitów konfiguracyjnych, a na **listingu 2** funkcję zapisującą rejestr CLKCTRL.



Rysunek 20. Konfigurowanie wyprowadzenia PF5 jako wyjściowego

Listing 1. Definicja rejestrów konfiguracyjnych

```
#include <avr/io.h>
FUSES =
{
    .APPEND = 0,
    .BODCFG = ACTIVE_DIS_gc|LVL_BODLEVEL0_gc|SAMPFREQ_1KHZ_gc|SLEEP_DIS_gc,
    .BOOTEND = 0,
    .OSCCFG = FREQSEL_20MHZ_gc|FUSE_OSCLOCK_bm,
    .SYSCFG0 = CRCSRC_NOCRC_gc|RSTPINCFG_RST_gc,
    .SYSCFG1 = SUT_64MS_gc,
    .WDTCFG = PERIOD_OFF_gc|WINDOW_OFF_gc,
};
```

Listing 2. Inicjalizowanie układu taktowania (rejestru CLKCTRL)

```
#include „../include/clkctrl.h”
//brief Initialize clkctrl interface
 int8_t CLKCTRL_init()
{
    //  ccp_write_io((void*)&CLKCTRL.OSC32KCTRLA,0 << CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */);
    //  ccp_write_io((void*)&(CLKCTRL.XOSC32KCTRLA),CLKCTRL_CSUT_1K_gc /* Crystal startup time: 1k cycles */
    //  | 0 << CLKCTRL_ENABLE_bp /* Enable: disabled */
    //  | 0 << CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */
    //  | 0 << CLKCTRL_SEL_bp /* Select: disabled */);
    //  ccp_write_io((void*)&(CLKCTRL.OSC20MCTRLA),0 << CLKCTRL_RUNSTDBY_bp /* Run standby: disabled */);
    ccp_write_io((void*)&(CLKCTRL.MCLKCTRLB),CLKCTRL_PDIV_2X_gc /* Prescaler division: 2X */
                | 1 << CLKCTRL_PEN_bp /* Prescaler enable: enabled */);
    ccp_write_io((void*)&(CLKCTRL.MCLKCTRLA),1 << CLKCTRL_CLKOUT_bp /* System clock out: enabled */
                | CLKCTRL_CLKSEL_OSC20M_gc /* clock select: 20MHz oscillator */);
    //ccp_write_io((void*)&(CLKCTRL.MCLKLOCK),0 << CLKCTRL_LOCKEN_bp /* lock ebable: disabled */);
    return 0;
}
```

Listing 3. Konfigurowanie wyprowadzenia PF5

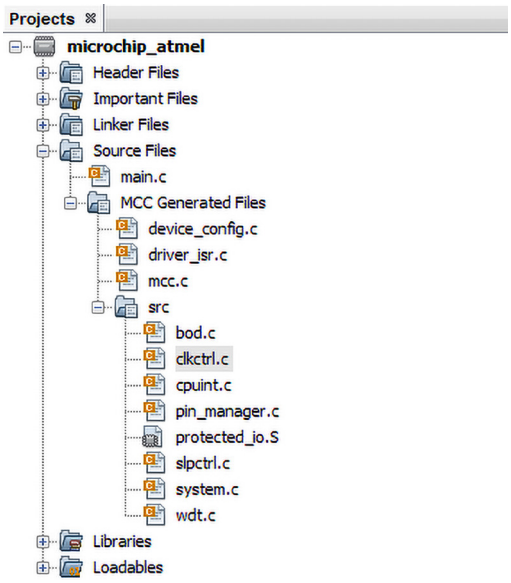
```
#include „../include/pin_manager.h”
void PIN_MANAGER_Initialize()
{
    IO_PF5_set_dir(
        // <y> Pin direction
        // <id> pad_dir
        // <PORT_DIR_OFF”> Off
        // <PORT_DIR_IN”> In
        // <PORT_DIR_OUT”> Out
        PORT_DIR_OUT);

    IO_PF5_set_level(
        // <y> Initial level
        // <id> pad_initial_level
        // <false”> Low
        // <true”> High
        false);

    IO_PF5_set_pull_mode(
        // <y> Pull configuration
        // <id> pad_pull_config
        // <PORT_PULL_OFF”> Off
        // <PORT_PULL_UP”> Pull-up
        PORT_PULL_OFF);

    IO_PF5_set_inverted(
        // <y> Invert I/O on pin
        // <id> pad_invert
        // <false”> Not inverted
        // <true”> Inverted
        false);

    IO_PF5_set_isc(
        // <y> Pin Input/Sense Configuration
        // <id> pad_isc
        // <PORT_ISC_INTDISABLE_gc”> Interrupt disabled but input buffer enabled
        // <PORT_ISC_BOTHEDGES_gc”> Sense Both Edges
        // <PORT_ISC_RISING_gc”> Sense Rising Edge
        // <PORT_ISC_FALLING_gc”> Sense Falling Edge
        // <PORT_ISC_INPUT_DISABLE_gc”> Digital Input Buffer disabled
        // <PORT_ISC_LEVEL_gc”> Sense low Level
        PORT_ISC_INTDISABLE_gc);
}
```

Mamy skonfigurowane taktowanie i przechodzimy teraz do napisania „nieśmiertelnego” programu migania diodą LED umieszczoną na płytce modułu. Dioda jest dołączona przez rezystor 1 kΩ ograniczający prąd katodą do linii portu PF5. Anoda diody jest połączona z napięciem zasilającym mikrokontroler (**rysunek 19**). Aby sterować diodą trzeba linię portu PF5 ustawić jako wyjściową. Do tego celu użyjemy konfiguratora wyprowadzeń wtyczki MCC. Na **rysunku 20** interfejs użytkownika, za pomocą którego można wykonać zmianę. Wystarczy jedynie kliknąć na wyprowadzenie i z listy wybrać dostępną funkcję wyprowadzenia. W przypadku PF5 może to być wejście lub wyjście GPIO. Jeżeli wcześniej wybierzemy konfigurację jakiegoś modułu na przykład komunikacyjnego to w liście funkcji wyprowadzenia mogą się pojawić alternatywne funkcje wyprowadzeń tego modułu.

Ustawienie wyprowadzenia jako wyjściowego GPIO powoduje zmianę koloru na zielony i pojawienie się opisu funkcji wyprowadzenia. Jednocześnie w oknie *Pin Module* zostanie pokazany wpis, w którym można konfigurować dodatkowe ustawienie, na przykład, podciąganie do plusa zasilania lub sposób wyzwalania przerwania zewnętrznego skojarzonego z danym wyprowadzeniem. Kod wygenerowany na podstawie konfiguracji wykonanej za pomocą MCC pokazano na **listingu 3**.

Po skonfigurowaniu mikrokontrolera generujemy kod wyjściowy klikając na przycisk *Generate* w oknie *Project Resources*. Konfigurator MCC wygeneruje wszystkie potrzebne pliki i katalogi, jak to zostało pokazane na **rysunku 21**. Pozostaje teraz tylko napisać program, który w pętli będzie zaświecał i gasił diodę LED. Na przykład taki, jak pokazany na **listingu 4**.

Program można skompilować lub skompilować i od razu wysłać do pamięci Flash mikrokontrolera. Po pierwszym przyłączeniu modułu do portu USB i zainstalowaniu sterowników IDE MPLAB X wysłał do pamięci programatora/debuggera umieszczonego na płytce odpowiednie firmware i po tej operacji mogłem zaprogramować mikrokontroler ATmega4089. Do manipulacji stanami wyprowadzeń użyłem funkcji bibliotecznej *PORTF_set_pin_level*, a do odliczania opóźnień również bibliotecznej funkcji *_delay_ms()*. Ta druga funkcja wymagała korekty wartości częstotliwości zegara taktującego mikrokontroler z domyślnej 1 MHz, na nasze 10 MHz. Nie wiedzieć czemu, żeby to zrobić trzeba było zmienić atrybuty pliku *delay.h*, ponieważ były ustawione na „tylko do odczytu". Tak napisany program zadziałał od razu, ale postanowiłem jeszcze sprawdzić, jak działa debugger. Po kliknięciu na ikonę debuggera na pasku narzędziowym program jest kompilowany i przesyłany do pamięci mikrokontrolera. Debugger powinien mieć możliwość wykonywania podstawowych operacji: pracy krokowej, uruchomienie ciągłego wykonywania programu (run), ustawiania (breakpoint), czy podglądanie wartości zmiennych globalnych i lokalnych. Po pobieżnym sprawdzeniu działania debugger'a okazało się, że jest jakiś problem z ustawianiem pułapek. W jednych miejscach da się je ustawić, w innych pojawia się problem. Na przykład, nie dało się ustawić żadnej pułapki w pętli nieskończonej.



Rysunek 21. Struktura plików wygenerowanych przez MCC

Listing 4. Pętla diody LED

```
/* Initializes MCU, drivers and middleware */
SYSTEM_Initialize(); //inicjalizacja mikrokontrolera
while (1)
{
    PORTF_set_pin_level(5,true);  //wystawienie na PF5 stanu wysokiego (dioda zgaszona)
    _delay_ms(1000);//opóźnienie 1sek
    PORTF_set_pin_level(5,false); //wystawienie na PF5 stanu niskiego (dioda zapalona)
    _delay_ms(1000);
}
```

Bardzo podobnie jak układ taktowania i linie portów, można konfigurować układy peryferyjne i często powiązany z nim układ przerwań.

Na koniec

Użytkownicy mikrokontrolerów AVR mają możliwość programowania w różnych środowiskach projektowych w tym w natywnym Atmel Studio. Przez wiele lat rozwoju na pewno jest to narzędzie dojrzałe i doskonale zintegrowane z kompilatorem AVR GCC. Nie można tego powiedzieć w chwili obecnej o zestawie MPLAB X z kompilatorem XC8 przeznaczonym dla mikrokontrolerów AVR, ale to tylko kwestia czasu i MPLAB X będzie również działał w wersji stabilnej. Czy zatem programiści projektujący układy z mikrokontrolerami powinni brać pod uwagę środowisko Microchipa? Według mnie zdecydowanie tak i to przynajmniej z dwóch powodów. Po pierwsze, środowiska projektowe i kompilatory ewoluują wraz z nowo wprowadzanymi do produkcji mikrokontrolerami. Mimo zapewnień, że Microchip nie porzuci Atmel Studio może się tak zdarzyć, że od pewnego momentu nowe elementy nie będą wspierane. Czy rzeczywiście tak będzie? Nie wiadomo, ale pierwszy krok ku temu być może już został zrobiony. Drugi powód, to wieloletnia strategia Microchip polegająca na rozwijaniu swoich bardzo dobrych układów peryferyjnych. Ta strategia doprowadziła do powstania nowoczesnych układów peryferyjnych pracujących niezależnie od rdzenia mikrokontrolera i wręcz nazwanych „core independent". Pomysł polegał na tym, by dość stary i lekko zmodyfikowany rdzeń PIC16F otoczyć układami peryferyjnymi, których praca prawie zupełnie nie obciąża samego rdzenia. Układy Core independent już zaczęły migrować do mikrokontrolerów AVR. Oczywiście, konfigurację i uruchamianie peryferii wykonuje rdzeń, ale kluczowa jest właśnie ta konfiguracja. Chociaż jest wykonywana przez zapisywanie rejestrów SFR i można ją wykonywać ręcznie w programie pisanym w języku C, to jest to dość uciążliwe. W IDE MPLAB X można to zrobić za pomocą wspomnianej wtyczki konfiguratora MCC. Nie sądzę, żeby Microchip wyposażył Atmel Studio w podobne narzędzie. Kiedy pojawiły się mikrokontrolery PIC16F z „core independent" próbowałem skonfigurować bardzo przydatny układ prostej logiki programowanej CLC. Zajęło to mnóstwo czasu zanim się udało. Po jakimś czasie trzeba było zmienić konfigurację i pracę trzeba było wykonać od nowa. MCC wykonuje to szybko i bez błędów.

Trzeba tez pamiętać, że wspólna platforma projektowa daje możliwość prawie natychmiastowego przejścia do pracy ze wszystkimi rodzinami mikrokontrolerów PIC, w tym według mnie bardzo udanej, 16-bitowej PIC24, czy dsPIC33 z jednostką przeznaczona do wykonywania operacji DSP. Dostępne są również wersje 32-bitowe z rdzeniem MIPS. Firma nie próżnuje i zaczyna wprowadzać mikrokontrolery dwurdzeniowe – na razie tylko w rodzinie dsPIC33CH. Dla konstruktora, który potrzebuje czegoś więcej niż 8-bitowej wydajności, może to być sposób na szybkie przejście na wyższy poziom. Dla wszystkich tych rodzin można pobrać pełne bezpłatne wersje kompilatorów z ograniczoną optymalizacją kodu.

Tomasz Jabłoński, EP