Nowe mikrokontrolery Microchip DSC z rodziny dsPIC33CH

Mikrokontrolery z rodziny dsPIC33 są przeznaczone głównie do stosowania w układach automatyki i sterowania, wymagających wykonywania złożonych algorytmów. Wydajny, 16-bitowy rdzeń RISC jest zintegrowany z jednostką DSP zoptymalizowaną do szybkiego wykonywania algorytmów przetwarzania cyfrowego. Takiemu połączeniu producent nadał nazwę Digital Signal Controllers – DSC.

Jednymi z bardziej wymagających zastosowań mikrokontrolerów dsPIC33 jest sterowanie silnikami BLDC oraz obwodami mocy zasilacza impulsowego. Bardzo ważnym rynkiem, który chłonie dużą liczbę elementów elektronicznych, również mikrokontrolerów, jest rynek elektroniki motoryzacyjnej. To zastosowanie jest bardzo wymagające dla elementów elektronicznych. Jest to spowodowane pracą w bardzo trudnych warunkach środowiskowych: narażeniem na skrajne temperatury otoczenia, zmiany wilgotności czy przepięcia elektryczne. Podobnie jest w przypadku urządzeń medycznych. Praca w trudnych warunkach i coraz większe oczekiwania co do niezawodności i funkcjonalności wymagają od producentów stałej modernizacji rodzin mikrokontrolerów. Przykładem takiej modernizacji jest nowa rodzina dsPIC33CH.

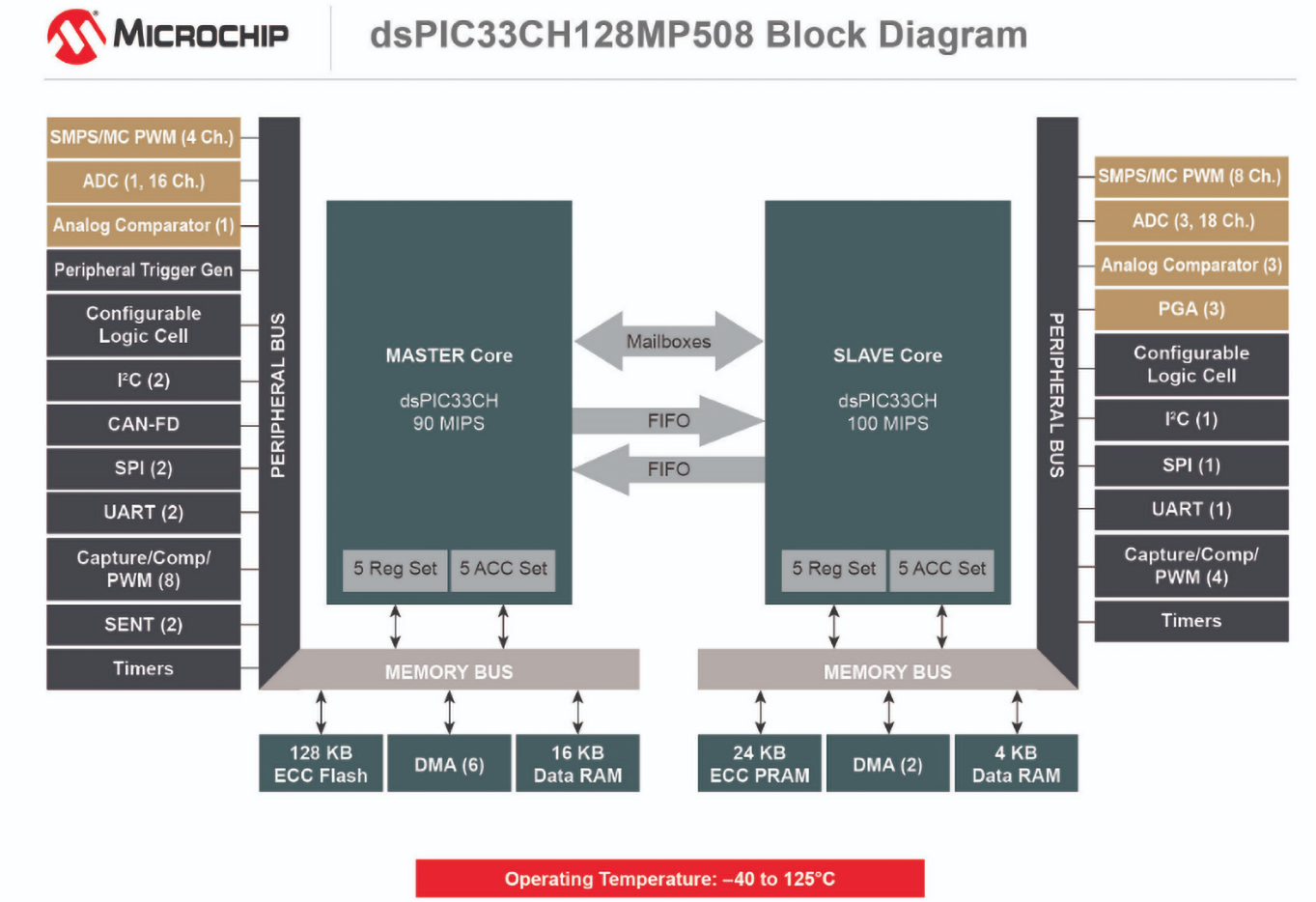
Wiele zaawansowanych układów sterowania wymaga szybkiej reakcji na przychodzące zdarzenia. Może to być, na przykład, sygnał z czujnika zamontowanego na wale silnika i informującego o tym, że jego obroty mają zbyt małą wartość, co może oznaczać przeciążenie. Układ sterowania powinien zareagować na ten sygnał tak szybko, jak to możliwe, aby zapobiec uszkodzeniu silnika, obwodów sterowania, zasilania i w końcu uszkodzeniu maszyny napędzanej przez silnik. W typowych aplikacjach najszybsza reakcja na zdarzenie jest możliwa przy użyciu przerwań. Mikrokontrolery z rodziny dsPIC33CH mają ciekawe rozwiązania, znacznie przyspieszające przyjęcie i obsługę przerwań, ale główną modyfikacją jest umieszczenie w strukturze mikrokontrolera dwóch niezależnych rdzeni, działających w konfiguracji master-slave, jak pokazano na **rysunku 1**.

Mikroprocesory wielordzeniowe to już dość stara idea, jak na standardy rozwoju elektroniki. Współcześnie nikogo nie dziwią mikroprocesory 4-rdzeniowe, montowane w smartfonach, niekoniecznie nawet tych „flagowych". Tu jednak mamy do czynienia z mikrokontrolerami przeznaczonymi do układów sterowania, a takie rozwiązania do niedawna były niedostępne. Jak pokazano na rysunku 1, każdy z rdzeń ma swoją magistralę dla układów peryferyjnych i magistralę danych. Każdy z rdzeni ma też niezależny zestaw bloków funkcjonalnych i swoją pamięć RAM. Taka budowa pozwala im na jednoczesne wykonywanie dwóch zadań, zupełnie niezależnie od siebie. Trzeba tylko pamiętać, że wyprowadzenia mikrokontrolera są współdzielone pomiędzy obie jednostki. Jeżeli blok peryferyjny rdzenia przejmie kontrolę nad wyprowadzeniem mikrokontrolera, to drugi rdzeń nie ma możliwości używania tego wyprowadzenia.

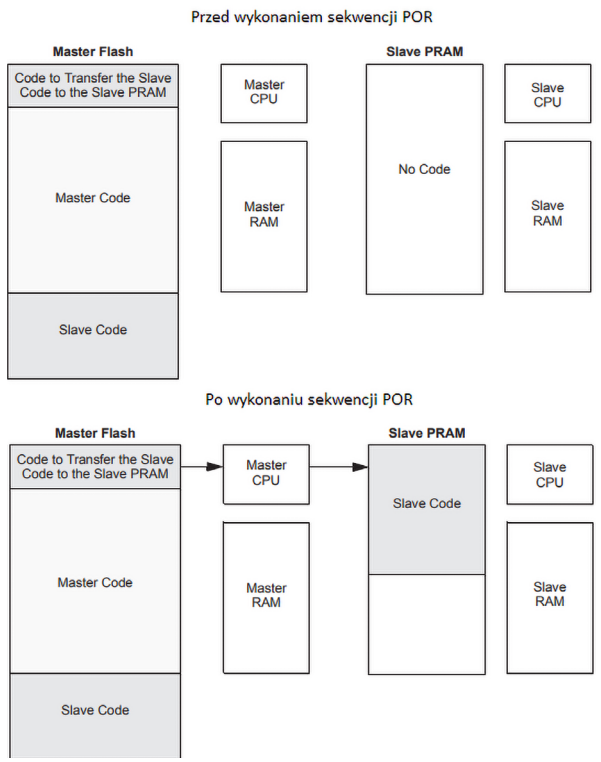
W praktyce oba rdzenie można programować i debugować oddzielnie. Mają własne, niezależne kontrolery przerwań, generatory przebiegu zegarowego i multiplekser dołączający wyprowadzenia do linii bloków funkcjonalnych PPS. Upraszczając, można powiedzieć, że każdy mikrokontroler dsPI33CH to w praktyce dwie kompletne, nowoczesne jednostki dsPIC33.

Wykorzystanie potencjału jednostki 2-rdzeniowej wymaga odpowiedniego oprogramowania. Po wygenerowaniu kodu programu dla obu jednostek, pamięć Flash rdzenia master jest zapisywana w trakcie programowania mikrokontrolera kodem dla jednostki master i kodem dla jednostki slave. Kiedy sekwencja zerowania mikrokontrolera (POR) zostanie zakończona, to kod przeznaczony dla rdzenia slave jest automatycznie przepisywany do pamięci PRAM (Program RAM) rdzenia slave. Następnie rozkazy rdzenia slave są pobierane z pamięci PRAM i obie jednostki pracują niezależnie. Pokazano to na **rysunku 2**. Oba rdzenie korzystają z tych samych źródeł przebiegów zegarowych, ale mają niezależne, programowalne moduły taktowania z układami powielania częstotliwości PLL i dzielnikami częstotliwości DIV (**rysunek 3**).

Dwa rdzenie pracujące niezależnie będą najczęściej wykonywały podzielone zadania jednego układu sterowania i dlatego muszą mieć możliwość komunikacji i wymiany danych. Do tego celu



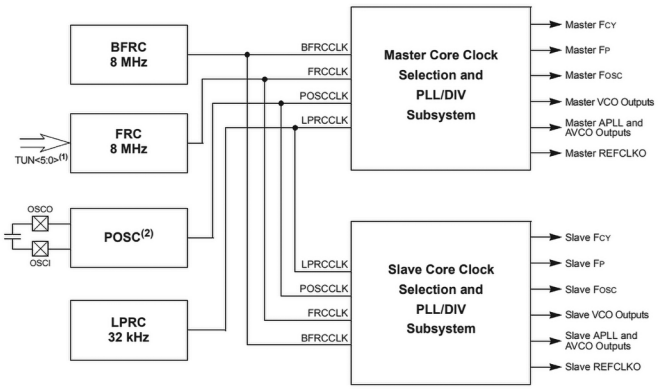
Rysunek 1. Schemat blokowy mikrokontrolera z rodziny dsPIC33CH



Rysunek 2. Transfer kodu do pamięci PRAM rdzenia slave

został zaprojektowany specjalny interfejs MSI (Master Slave Interface). Jest on czymś w rodzaju bramy do przesyłania danych pomiędzy rdzeniami i z założenia główną rolę odgrywa w nim procesor master.

Jak już wiemy, oba rdzenie pracują taktowane niezależnymi przebiegami zegarowymi i musimy przyjąć, że mogą one pracować przy



Rysunek 3. Układy taktowania rdzeni ze współdzielonymi źródłami sygnałów zegarowych

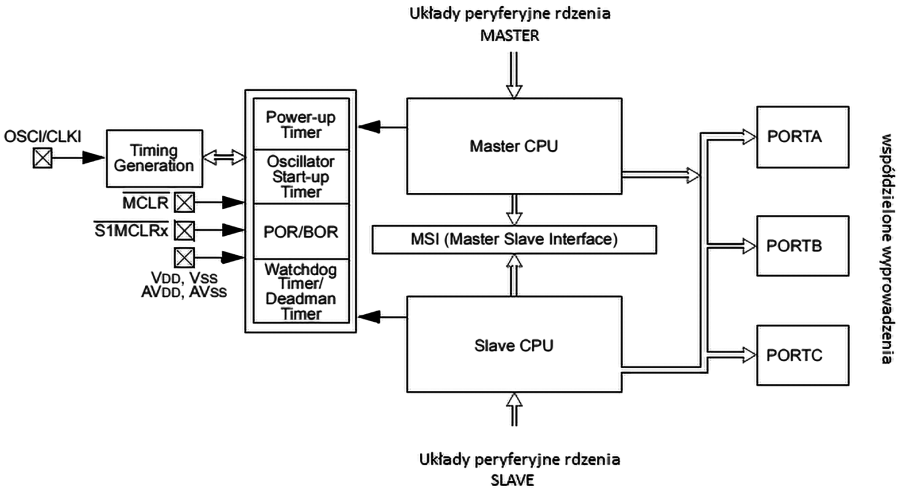
znacznie różniących się częstotliwościach taktowania. Interfejs MSI powinien zapewnić synchronizację przesyłania danych. Rejestry sterujące interfejsu są umieszczane w obszarze SFR każdego z rdzeni. MSI jest zbudowany z:

- 16 jednokierunkowych rejestrów danych Data Mailbox Registers. Kierunek przesyłania danych każdego z rejestrów jest określony przez bity konfiguracyjne fuse bit w trakcie programowania pamięci mikrokontrolera.
- 8 bloków kontroli przepływu danych Mailbox Data Flow Control Protocol Blocks indywidualnie włączanych przez fuse bits.

Port używany do zapisywania danych jest portem aktywnym, a port odczytywania danych jest portem pasywnym. Przepływ danych może być kontrolowany za pomocą automatycznie zgłaszanych przerwań lub testowany metodą poolingu. Do transferu danych można zamiennie używać mechanizmu DMA. Transmisja danych może się odbywać w trybie **Mailbox-based transfer** i **FIFO based transfer**.

Szczegółowa budowa interfejsu MSI i zasada jego działania jest dokładnie opisana w dokumencie „Master Slave Interface MSI Module DS70005278B", dostępnym na stronie internetowej firmy Microchip.

Podział zadania sterowania na dwa niezależne rdzenie i poprawnie napisane oprogramowanie na pewno pozwolą na wykonanie wydajnej aplikacji. Dla wielu programistów będzie to na początku trudne zadanie. Tworzenie projektu dla dwóch pracujących równolegle aplikacji nie jest też standardowym zadaniem programistów systemów wbudowanych, nawet tych, którzy programują, posługując się systemami RTOS. Pierwszym poważnym problemem będzie skonfigurowanie niezależnie dla każdego z rdzeni: taktowania, układu watchdog, układu ICD (debugowanie kodu), współdzielenia wyprowadzeń mikrokontrolera i oczywiście interfejsu MSI. Trzeba również mieć możliwość tworzenia kodu dla dwóch rdzeni, debugowania tego kodu, a potem zapisania go w pamięci Flash w takiej lokalizacji, żeby był automatycznie przepisywany do pamięci PRAM rdzenia slave. Tu wielką pomocą będzie wtyczka MCC współpracująca ze środowiskiem MPLAB X IDE oraz przewodnik programisty Microchip Developer z zamieszczonym dokumentem MPLAB Code Configurator Support for Dual-Core Devices, dostępny on-line po adresem ***http://bit.ly/2DFYs9s***.



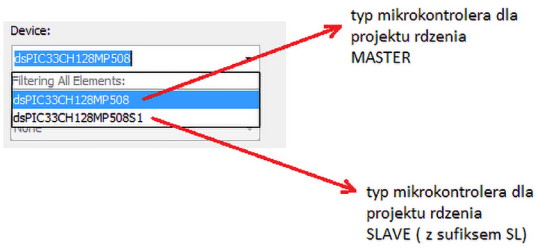
Rysunek 4. Rdzenie połączone interfejsem MSI

Proces konfigurowania obu rdzeni jest dość złożony i przebiega wieloetapowo. Konfiguracja jest zapisywana w nieulotnej pamięci konfiguracyjnej (fuse bit). W trakcie pracy nad projektem trzeba importować te ustawienia z rdzenia master do rdzenia slave. Wspomniany wyżej przewodnik powinien rozwiać wszelkie wątpliwości odnośnie do sposobu konfigurowania rdzeni, tworzenia dwóch niezależnych aplikacji i współpracy pomiędzy nimi. Opisany jest tam też „prosty" program przykładowy wymieniający informację pomiędzy rdzeniami i sygnalizujący za pomocą LED sterowanych przez rdzenie master i slave. Kod aplikacji wykorzystującej dwurdzeniowość jest z oczywistych względów dzielony na dwie prawie niezależne części, powiązane wspólnymi ustawieniami bitów konfiguracyjnych.

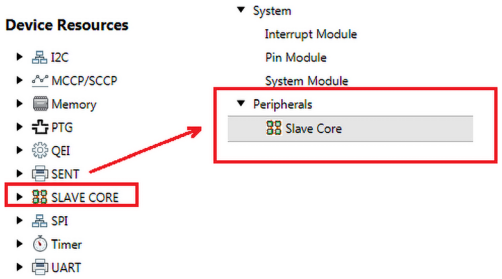
Najpierw tworzymy projekt dla rdzenia master, wybierając nazwę mikrokontrolera bez sufiksu „Sl", jak pokazano na **rysunku 5**. Do tego celu jest wykorzystane środowisko programowe IDE MPLAB X w wersji 5.10 lub nowszej z zainstalowana wtyczką MCC (od wersji 3.66). Używając wtyczki MCC, w tym projekcie trzeba z listy Device Resources dodać element SLAVE CORE (**rysunek 6**). Pozwoli to na zdefiniowanie wspólnych ustawień dla rdzeni master i slave, które później będą importowane do projektu tworzonego dla rdzenia slave.

Jak łatwo się domyślić, jedną z ważniejszych definicji z punktu widzenia pracy z dwoma rdzeniami będzie konfigurowanie interfejsu MSI. Ustawienia MSI są zapamiętywane w pamięci konfiguracyjnej. Każdy rdzeń ma swoją pamięć konfiguracyjną i po skonfigurowaniu MSI i innych ustawień dotyczących rdzenia slave (w projekcie dla rdzenia master) trzeba je potem będzie zaimportować do pamięci konfiguracyjnej projektu dla rdzenia slave. Na **rysunku 7** pokazano przykładową konfigurację MSI wykonaną za pomocą MCC. Zdefiniowano tu 2 protokoły: „A" i „B". Protokół „A" przesyła 4 bajty danych z rdzenia slave do rdzenia master, a protokół „B" 4 bajty danych w kierunku odwrotnym. Każda z tych transmisji może zgłaszać przerwanie. Priorytet i kontekst tych przerwań jest definiowany w oknie Interrupt Manager.

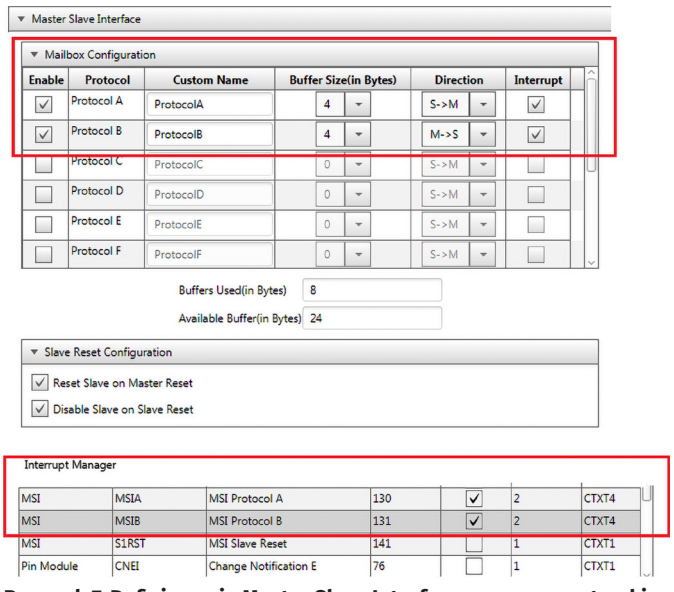
W projekcie dla rdzenia master jest generowany kod umieszczany w pamięci Flash, przeznaczony dla tego rdzenia. Ale jak już wiemy, projekt musi również umieścić w pamięci Flash kod dla rdzenia slave, który po zakończeniu zerowania jest przepisywany do pamięci PRAM (rys. 2). Żeby ten projekt „wiedział", jaki kod ma pobrać, musi mieć zdefiniowaną nazwę projektu dla rdzenia slave. Nadanie nazwy projektu programu dla rdzenia slave jest wykonywane w oknie Slave Project Name (**rysunek 8**). Potem musimy utworzyć projekt dla



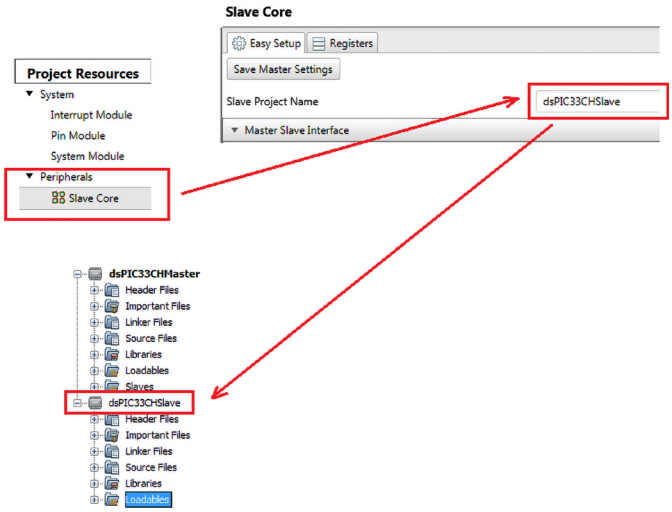
Rysunek 5. Typ mikrokontrolera z wyborem projektu master/slave



Rysunek 6. MCC – dodanie definicji Slave Core



Rysunek 7. Definiowanie Master Slave Interface za pomocą wtyczki MCC

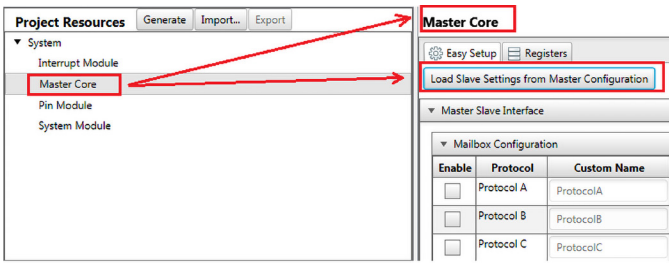


Rysunek 8. Nadanie nazwy projektu dla rdzenia slave

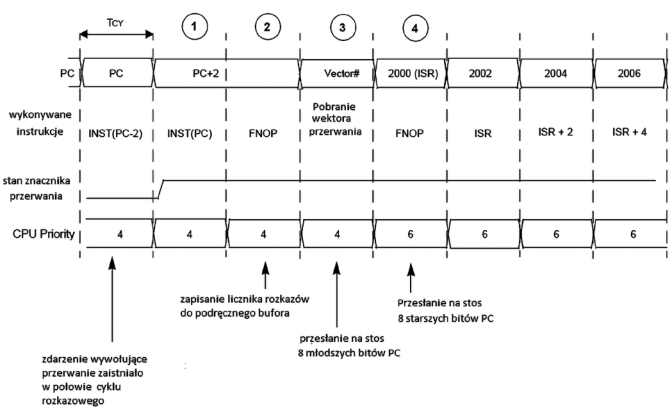
rdzenia slave z mikrokontrolerem o nazwie właściwiej dla projektu slave (rysunek 5). Ten projekt musi mieć nazwę dokładnie taką samą, jaką wpisujemy w oknie Slave Project Name.

Po wykonaniu konfiguracji trzeba w bloku peryferyjnym Slave Core kliknąć na Save Master Settings, aby zapamiętać ustawienia w pliku MyConfig.mc3. Następnie otwieramy projekt dla rdzenia slave o nazwie dsPIC33CHSlave i uruchamiamy wtyczkę konfiguratora MCC. Tam w zakładce System pojawi się element Master Core. W tym elemencie klikamy na przycisk LoadSlave Settings from Master Configuration – rysunek 9. Odszukujemy w katalogu projektu dsPIC33CHMaster plik MyConfig.mc3 i importujemy ustawienia dla rdzenia slave wykonane w projekcie dla rdzenia master. W pierwszym momencie może się to wydawać trochę zagmatwane, ale w rzeczywistości jest to działanie logiczne.

Wbudowanie dwóch pełnoprawnych rdzeni (z punktu widzenia wydajności i zasobów) daje programistom duże możliwości projektowania wydajnych układów sterowania i nadzoru. Jednak to nie wszystko, co oferują nowe jednostki Microchipa w tej dziedzinie.



Rysunek 9. Element Master Core i importowanie ustawień rdzenia master w projekcie o nazwie dsPIC33CHSlave



Rysunek 10. Sekwencja czynności wykonywanych w trakcie przyjmowania i wykonywania przerwania

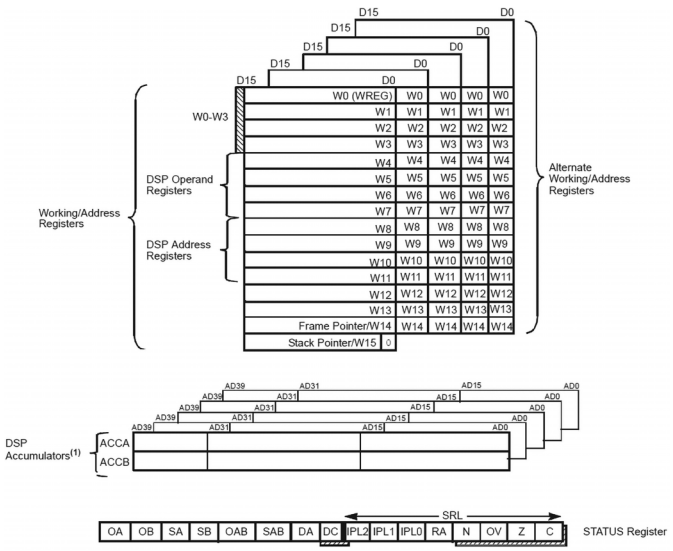
Kolejnym, może nie tak spektakularnym jak dwa rdzenie, ale również istotnym ulepszeniem jest modyfikacja pracy kontrolera przerwań.

Kontrolery przerwań w różnych mikrokontrolerach działają w podobny sposób. Źródło przerwania musi być odblokowane i jeżeli jest taka możliwość, to mieć zaprogramowany priorytet. Przyjęcie przerwania powoduje „zawieszenie" wykonywania programu głównego i rozpoczęcie wykonywania procedur obsługi przerwania. W wielu układach sterowania kluczowy jest czas upływający od momentu zaistnienia zdarzenia wywołującego przerwanie do chwili rozpoczęcia wykonywania kodu obsługi przerwania. Niestety, nigdy nie następuje to natychmiast, ponieważ przedtem mikrokontroler ma do wykonania kilka czynności. Załóżmy, że zdarzenie jest asynchroniczne, czyli może wystąpić w momencie niezsynchronizowanym z zegarem taktującym rdzeń. Zanim stanie się cokolwiek innego, mikrokontroler musi dokończyć wykonywanie bieżącego rozkazu. Potem jest inicjowana sekwencja kolejnych czynności:

- Na stos lub do odpowiednich buforów w pamięci RAM jest zapisywany licznik rozkazów (PC) po to, aby po zakończeniu obsługi przerwania program mógł wrócić w to samo miejsce.
- Układy logiczne sterownika przerwań ładują do licznika rozkazów wektor przerwania, czyli adres, pod którym jest zapisany adres skoku do procedury obsługi przerwania.
- Wykonywany jest skok do obsługi przerwania.
- Procedura obsługi przerwania musi w pierwszej kolejności zachować za zwartość rejestrów, które są współdzielone z programem głównym, na przykład: rejestrów roboczych, akumulatora jednostki DSP i statusu.

Dopiero od tego momentu można wykonać sekwencję rozkazów reagujących na zaistniałe zdarzenie. Pokazano to schematycznie dla przerwania przychodzącego dla instrukcji wykonywanej w jednym cyklu na **rysunku 10**.

Pierwszym intuicyjnym sposobem na skrócenie czasu reakcji jest zastosowanie szybszego mikrokontrolera. Może to być jednostka taktowana przebiegiem o większej częstotliwości i/lub jednostką z bardziej optymalnym rdzeniem (rozkazy wykonywane w mniejszej liczbie cykli, architektura RISC). Według danych producenta w rodzinie mikrokontrolerów dsPIC33CH dwukrotnie zwiększono wydajność w porównaniu do mikrokontrolerów dsPIC33 starszej generacji. Na przykład starsza rodzina dsPICFJ64GS ma katalogową wydajność na poziomie 50 MPIS, a dsPIC33CH: dla rdzenia master około 90 MIPS, a dla rdzenia slave około 100 MIPS. Przypuszczalnie różnica wynika z tego, że rdzeń slave używa szybszej pamięci PRAM. Niestety, nie znalazłem metodologii pomiaru wydajności rdzenia i nie wiem, jak ten wzrost ma się do czasu wykonania pojedynczej



Rysunek 11. Alternatywne rejestry W, akumulatora DSP i rejestr statusu

instrukcji. Jednak należy spodziewać się, że w nowych mikrokontrolerach rozkazy będą wykonywane szybciej, a co za tym idzie, czas reakcji na zdarzenie będzie krótszy.

Kolejnym wąskim gardłem jest konieczność zachowania zawartości rejestrów współużytkowanych przez program główny i procedurę obsługi przerwania. Ta procedura musi na samym początku zapamiętać zawartość – czyli w praktyce przesłać na stos – zawartość rejestrów roboczych W, akumulatora DSP (jeżeli używany) i rejestru statusowego. Ostatnimi czynnościami kończącymi obsługę przerwania jest odtworzenia zawartości tych rejestrów (pobranie ze stosu). Każdy z rdzeni dsPIC33CH ma 16 rejestrów ogólnego przeznaczenia W0…W15. Zależnie od tego, co ma robić procedura obsługi przerwania, trzeba część tych rejestrów zachowywać na stosie, a po zakończeniu odtworzyć ich zawartość, pobierając ze stosu. Aby tego nie robić programowo, ponieważ zabiera to czas procesora, w CPU wbudowano cztery zestawy rejestrów alternatywnych W0…W14. Jednostka DSP ma również 4 zestawy alternatywnych rejestrów akumulatora DSP. Producent nazywa zestaw rejestrów alternatywnych **Interrupt Context Saving Registers**. Zestaw rejestrów alternatywnych jest powiązany z poziomem priorytetu przerwania. Jest to zapisywane w rejestrze konfiguracyjnym FALTREG w momencie programowania pamięci mikrokontrolera. Jeżeli programista chce to przyporządkowanie zmienić w trakcie pracy programu, to może użyć dedykowanego rozkazu CTXTSWP.

Po zgłoszeniu przerwania o zadanym priorytecie układy logiczne automatycznie przełączają CPU na używanie przez procedurę obsługi zaprogramowanego zestawu rejestrów alternatywnych, przypisanych do tego priorytetu. Program obsługi przerwania nie niszczy zawartości zestawu rejestrów podstawowych i nie trzeba ich zachowywać na stosie. Kilka zestawów rejestrów alternatywnych umożliwia obsługę przerwań zagnieżdżonych.

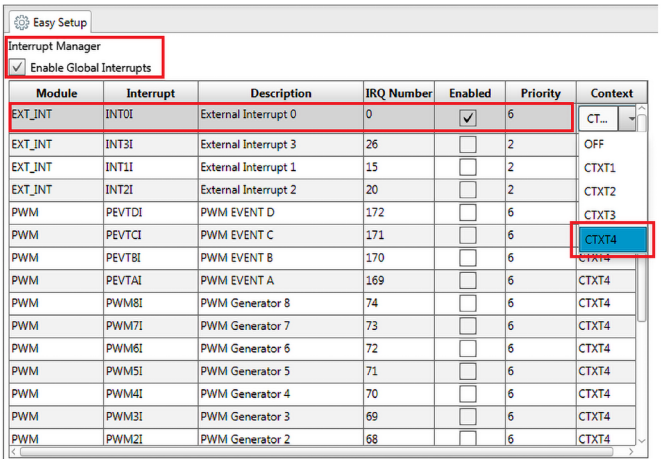
Na **rysunku 12** pokazano okno konfiguracji układu przerwań z wykorzystaniem wtyczki MCC. Dla każdego ze źródeł przerwań można indywidulanie przypisać priorytet (kolumna Priority) i zestaw rejestrów alternatywnych (Context). W naszym przykładzie jest konfigurowane przerwanie zewnętrzne EX_INT INT0 z priorytetem 6 i zestawem rejestrów CTXT4.

Szybka reakcja na przerwanie to bardzo duża zaleta w układach sterowania z krytycznym czasowo wymaganiem reakcji na zdarzenia, ale rodzina mikrokontrolerów sdP33CH ma również wysokiej jakości bloki peryferyjne. W czasach, kiedy wiele urządzeń pracuje w sieci LAN i jest dołączonych do Internetu, dużego znaczenia nabiera problem szeroko rozumianego bezpieczeństwa funkcjonalnego. W mikrokontroler wbudowano funkcjonalności mające na celu wykonanie bezpiecznie działających aplikacji. Jest to test pamięci RAM (BIST) do sprawdzenia poprawności pamięci RAM, zabezpieczenie przed dostępem do zawartości pamięci Flash CodeGuard, Deadman Timer DMT, moduł korekcji błędów ECC wbudowany w kontroler pamięci Flash, moduł generujący wielomian kontrolny CRC.

Automatyczne sprawdzenie pamięci RAM może być uruchamiane po zerowaniu mikrokontrolera lub na żądanie, w trakcie normalnej pracy. Test sprawdza poprawność działania komórek we wszystkich lokacjach pamięci RAM i w wypadku wykrycia uszkodzenia komórki RAM sygnalizuje to odpowiednimi bitami statusowymi.

System ochrony zawartości pamięci nieulotnej Flash (nazwany CodeGuard) jest podzielony na 3 segmenty: **Boot Segment**, **General Segment** i **Configuration Segment**. Najwyższy stopień ochrony ma segment Boot, a segment General, w którym jest umieszczony kod użytkownika, najniższy.

Główną funkcją Deadman Timer DMT jest monitorowanie prawidłowości wykonywania programu użytkownika. DMT jest licznikiem synchronicznym, zliczającym liczbę cykli pobrania rozkazów (fetch). Jeżeli program użytkownika nie wyzeruje licznika DMT po ustalonej liczbie pobrań, to jest zgłaszane wewnętrzne przerwanie i program może zareagować na takie zdarzenie. Użytkownik programuje limit czasu zliczania lub okno określające zakres zliczanych pobrań.



Module	Interrupt	Description	IRQ Number	Enabled	Priority	Context
EXT_INT	INT0I	External Interrupt 0	0	☑	6	CT...
EXT_INT	INT3I	External Interrupt 3	26	☐	2	OFF
EXT_INT	INT1I	External Interrupt 1	15	☐	2	CTXT1
EXT_INT	INT2I	External Interrupt 2	20	☐	2	CTXT2
PWM	PEVTDI	PWM EVENT D	172	☐	6	CTXT3
PWM	PEVTCI	PWM EVENT C	171	☐	6	CTXT4
PWM	PEVTBI	PWM EVENT B	170	☐	6	CTXT4
PWM	PEVTAI	PWM EVENT A	169	☐	6	CTXT4
PWM	PWM8I	PWM Generator 8	74	☐	6	CTXT4
PWM	PWM7I	PWM Generator 7	73	☐	6	CTXT4
PWM	PWM6I	PWM Generator 6	72	☐	6	CTXT4
PWM	PWM5I	PWM Generator 5	71	☐	6	CTXT4
PWM	PWM4I	PWM Generator 4	70	☐	6	CTXT4
PWM	PWM3I	PWM Generator 3	69	☐	6	CTXT4
PWM	PWM2I	PWM Generator 2	68	☐	6	CTXT4

Rysunek 12. Konfigurowanie priorytetów przerwań i przełączania kontekstu za pomocą MCC

Moduł korekcji błędów pamięci Flash ECC wykrywa błędy na jednej pozycji w słowie (błędy 1-bitowe) i automatycznie je koryguje. W czasie zapisywania danych do pamięci (programowanie mikrokontrolera) ECC generuje 7-bitowy wielomian kontrolny dla każdych z dwóch słów instrukcji. Te wielomiany są zapisywane w specjalnym obszarze pamięci, niedostępnym dla programu użytkownika (do odczytu i zapisu). W czasie odczytywania zawartości pamięci Flash (na przykład pobieranie rozkazów przez CPU) wielomian jest ponownie obliczany i porównywany z tym wygenerowanym w czasie zapisu. Brak zgodności obu wielomianów wywołuje jedną z dwóch reakcji modułu ECC:

- Dla błędu 1-bitowego następuje korekcja błędu i ponowne zapisanie komórki pamięci.
- Dla błędu na więcej niż jednej pozycji dane w pamięci nie są ani korygowane, ani zapisywane. Mikrokontroler wykonuje wtedy restart i odczytując odpowiedni rejestr, można zidentyfikować jego przyczynę.

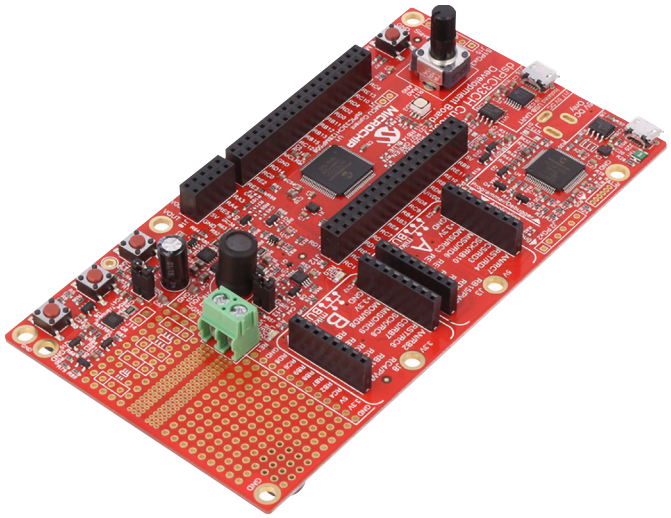
Zadaniem modułu CRC jest sprzętowe wyliczanie sumy kontrolnej głównie w celu zabezpieczenia przed skutkami przekłamania bloku danych transmitowanych lub odbieranych. Wyliczanie CRC jest zawsze oczekiwane przy przesyłaniu danych przez łącza radiowe. Bardziej skomplikowane algorytmy kontroli transmisji potrafią za pomocą wyliczanych sum CRC korygować błędy w przesyłanym bloku danych.

Oprócz bardziej zaawansowanych układów chroniących wykonywany kod przed błędami, znajdziemy tu typowe rozwiązania: licznik watchdog WDT, układ nadzoru układu taktowania Clock Monitor with Backup Ocillator i FCSN (Fail Safe Clock Monitoring).

Rodzina mikrokontrolerów dsPIC33 jest często kojarzona z układami sterowania silnikami elektrycznymi oraz ze sterownikami układów mocy. Do tego celu jest przeznaczony blok peryferyjny HSPWM. Ma on 8 niezależnych, szybkich generatorów PWM o rozdzielczości 250 ps. Wszystkie kanały są wyposażone w programowane układy generowania czasu martwego (dead time). Według producenta, HSPWM można zastosować w aplikacjach: konwerterów DC/DC i AC/DC, systemach oświetleniowych, sterowania silnikami BLDC, PMSM, ACIM i SRM.

Poważnie potraktowano też część analogową. Dwa przetworniki o rozdzielczości 12 bitów pracują z częstotliwością 3,5 Ms/s. Każdy z 24 kanałów analogowych ma bufor do zapisywania wyniku konwersji. Przetworniki są wyzwalane niezależnie, przy czym to wyzwalanie można dość elastycznie konfigurować.

Układy elektroniczne stosowane w motoryzacji wymagają przesyłania danych przez łącza szeregowe. Stosuje się tu interfejsy LIN i CAN. Interfejs LIN jest używany do komunikacji z nieskomplikowanymi czujnikami i umożliwia tworzenie małych sieci (podsystemów). Te podsystemy mogą się potem komunikować z innymi systemami za pomocą magistrali CAN. Moduł magistrali szeregowej UART wspiera protokoły LIN2.2, ale też DMX oraz IrDA. Podstawowym interfejsem



Fotografia 13. dsPIC33CH Curiosity Development Board

przeznaczonym dla aplikacji automotive jest interfejs CAN Flexible Data (FD) pracujący w trybach FD i CAN2.0B.

Tradycyjnie Microchip wspiera konstruktorów, oferując im możliwość zakupu modułów ewaluacyjnych. Takim modułem jest na przykład dsPIC33CH Curiosity Development Board (**fotografia 13**). Pochodzi ze znanej serii modułów „Curiosity", charakteryzujących się niewygórowaną ceną i przeznaczonych do testowania mikrokontrolerów bez ponoszenia dodatkowych kosztów. Te moduły mają wbudowany programator/debugger, kilka elementów, takich jak przyciski, diody LED, konwerter USB/UART i złącza dla modułów rozszerzających oferowanych przez serbską firmę Mikroelektronika. Oprócz modułu Curiosity można kupić moduły procesora dsPIC33CH128MP508 Motor Control i dsPIC33CH128MP508 General Purpose przeznaczone do płyt bazowych z bogatym wyposażeniem. Ale to już jest droższa alternatywa.

Rodzina mikrokontrolerów dsPIC33CH jest zorientowana na aplikacje wymagające krótkiego czasu odpowiedzi na zdarzenia. Zastosowane tu niestandardowe rozwiązania pozwalają na krótki czas reakcji, który przy standardowym ujęciu problemu jest możliwy do uzyskania dla o wiele szybszych mikrokontrolerów. Wyposażenie w dwa rdzenie z jednostką DSP, bardzo dobre bloki peryferyjne, zmodyfikowany układ przerwań, układy kontroli pamięci i zabezpieczenia predestynują rodzinę dsPI33CH do wielu zaawansowanych zastosowań w różnych wymagających gałęziach techniki. Tradycyjnie dobre wsparcie programowe: środowisko MPLAB IDE, wtyczka MCC i bezpłatny kompilator języka C, uzupełnione przez wspomniany przewodnik po programowaniu i dostępny na stronie producenta program testowy, pozwalają wszystkim chętnym na łagodne wejście w świat programowania aplikacji dla mikrokontrolerów wielordzeniowych.

Tomasz Jabłoński, EP