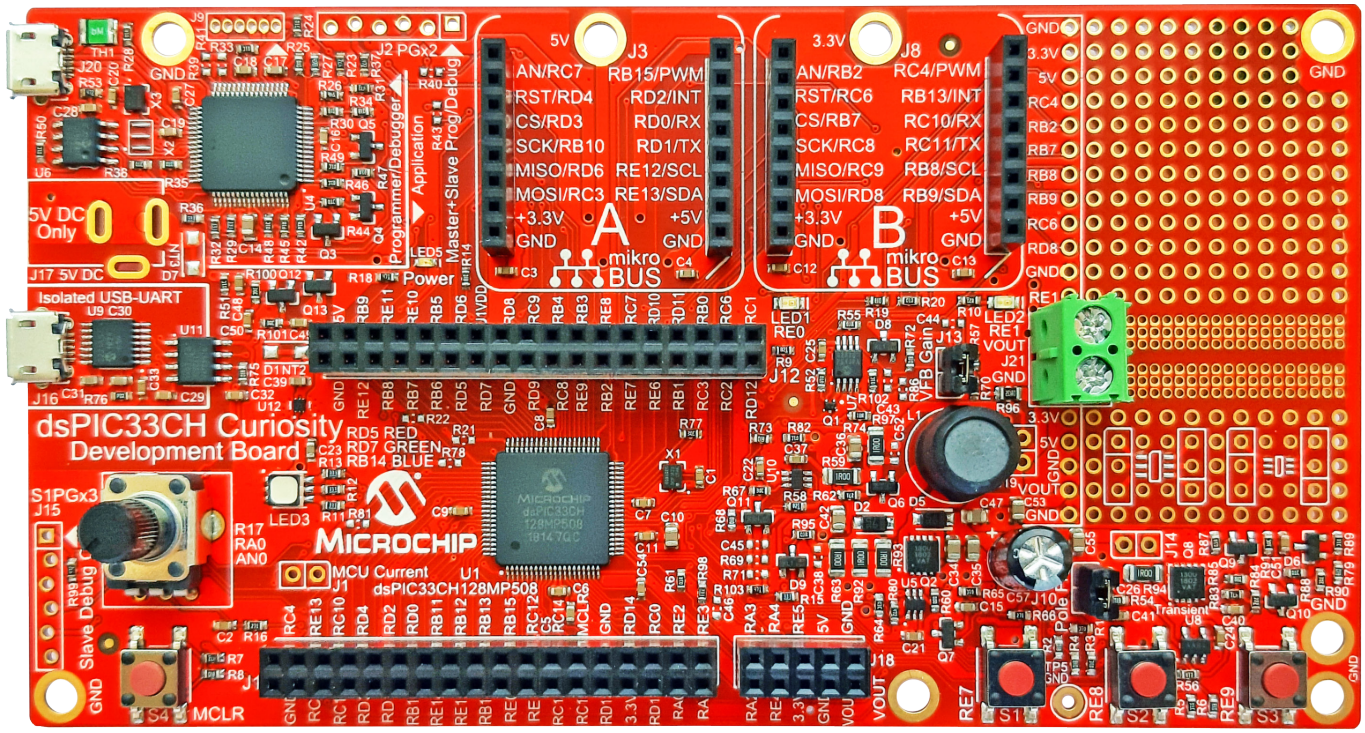


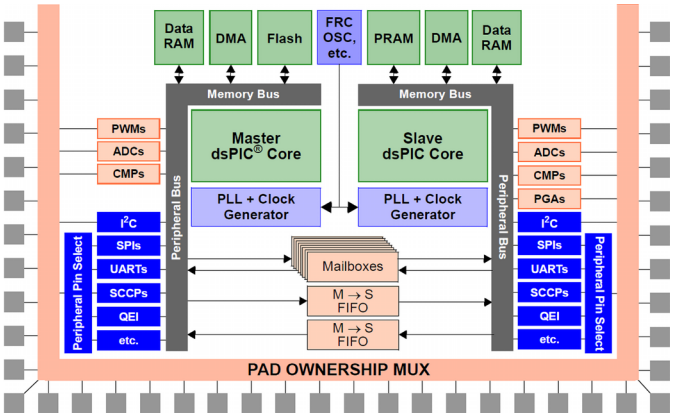
Fotografia 1. Moduł dsPIC33CH Curiosity Development Board

Dwurdzeniowe dsPIC33CH w praktyce

*Dzięki uprzejmości firmy Microchip stałem się posiadaczem zestawu dsPI33CH Curiosity Development Board (**fotografia 1**). Jest to testowa platforma mająca na pokładzie opisywany już w „Elektronice Praktycznej" dwurdzeniowy mikrokontroler dsPIC33CH128MP508. Wielordzeniowe mikroprocesory nie są czymś niezwykłym, ale w przypadku mikrokontrolerów takie rozwiązania dopiero zaczynają się pojawiać.*

Architektura mikrokontrolera dwurdzeniowego z rodziny sdPIC33CH została pokazana na **rysunku 2**. W jednej strukturze umieszczono dwa praktycznie niezależne od siebie mikrokontrolery. Każdy z nich ma swoje magistrale pamięci i układów peryferyjnych, swój zestaw układów peryferyjnych i układ programowania taktowania. Wspólne elementy to oscylator przebiegu taktującego oraz dzielone między siebie wyprowadzenia. Oba rdzenie łączy też możliwość wzajemnej wymiany danych poprzez moduł MSI – Master Slave Interface oraz kanały DMA.

Mikrokontroler dsPIC33CH128MP508 jest bogato wyposażony w pamięć i układy peryferyjne. Do dyspozycji jest 128 kB pamięci programu Flash i 16 kB pamięci danych RAM. Trzeba jednak pamiętać, że pamięć Flash jest współdzielona i musi pomieścić kod rdzenia Master i kod rdzenia Slave. Na kod programu rdzenia Slave przewidziano 24 kB pamięci PRAM. W przypadku, kiedy cała pamięć przeznaczona dla rdzenia Slave jest wykorzystana, na kod rdzenia Master można wykorzystać maksymalnie 104 kB pamięci Flash. Do pamięci PRAM jest na żądanie rdzenia MASTER przepisywany kod z wydzielonego obszaru pamięci Flash.



Rysunek 2. Architektura mikrokontrolera rodziny dsPIC33CH

Podział układów peryferyjnych pomiędzy rdzenie został pokazany na **rysunku 3**.

Projekt w środowisku MPLAB X IDE

Projekt w środowisku MPLAB X IDE wykorzystujący dwa rdzenie dla mikrokontrolerów dwurdzeniowych to w praktyce dwa standardowe projekty tworzone osobno dla rdzenia Master i rdzenia Slave i tak skonfigurowane, żeby ze sobą współpracowały.

Prace nad projektem dla rdzenia Master rozpoczynamy standardowo od polecenia New Project (File → New Project) – **rysunek 4**. Z rozwijanej listy wybieramy mikrokontroler dsPIC33CH128MP508 zamontowany w module dsPIC33CH Curiosity Development Board.

Rdzeń	Flash	RAM	12-bit ADC	Timery	SCCP	CAN FD	SENT	UART	SPI/I²S	I²C
Master	128 kB	16 kB	1	1	8	1	2	2		2
Slave	24 kB (PRAM)	4 kB	3	1	4	–	–	1		1

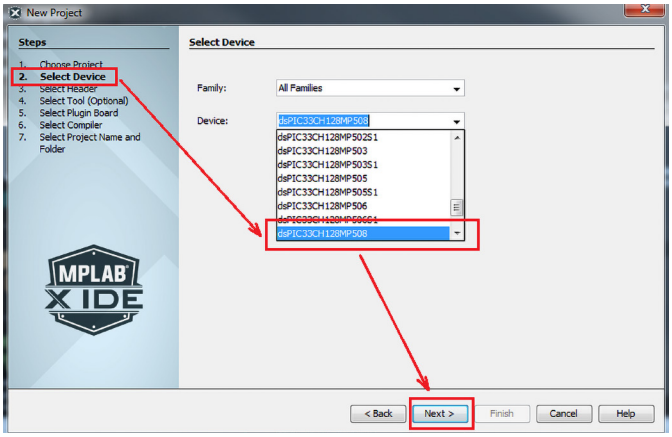
Rdzeń	QEI	CLC	PTG	CRC	PWM	komparator	PGA	Current Bias	REFO
Master	1	4	1	1	4	1	–	1	1
Slave	1	4	–	–	8	3	3	0	1

Rysunek 3. Zestawienie układów peryferyjnych dsPIC33CH128MP508

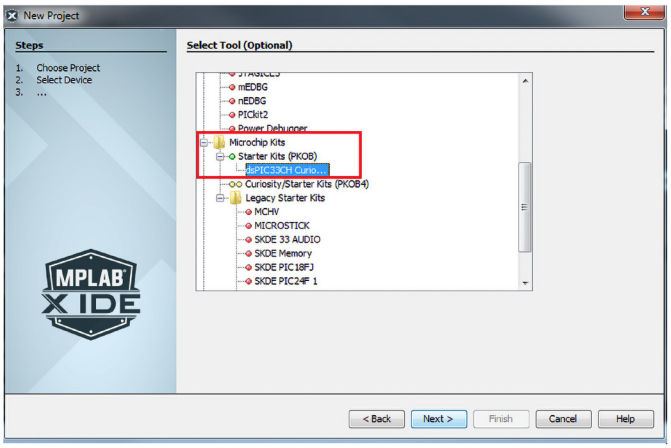
Dla każdego mikrokontrolera z rodziny dwurdzeniowych mikrokontrolerów dsPIC33CH umieszczono na liście dwa elementy. Dla projektu rdzenia Master nazwa mikrokontrolera jest taka jak jego nazwa katalogowa – dla naszego przypadku jest to dsPIC33CH128MP508. Poza tym na liście jest umieszczony dodatkowy element z sufiksem S1 przeznaczony dla projektów rdzenia Slave. Dokładniej zostanie to opisane później przy okazji tworzenia projektu dla rdzenia Slave.

Kolejnym krokiem wykonywanym w kreatorze projektu jest wybór programatora /debuggera. Na płytce modułu jest zabudowany debugger wyposażony w złącze USB micro. Złącze to jest wykorzystywane do połączenia z komputerem i równolegle do zasilania modułu. Jeżeli w trakcie tworzenia projektu moduł jest połączony z komputerem przez USB, to na liście Select Tools pojawia się do wyboru opcja Starter Kits (PKOB) i należy ją wybrać – **rysunek 5**

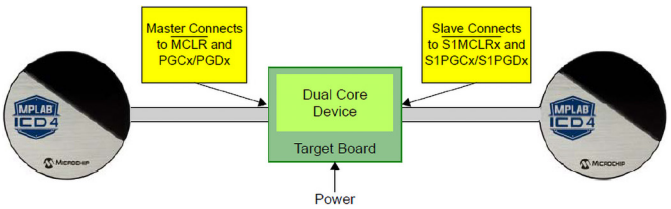
Debugger pozwala na programowanie pamięci Flash i debugowanie jednego rdzenia. Jest możliwe jednoczesne użycie dwu sprzętowych debuggerów podłączonych do dwu portów USB i debugowanie dwu rdzeni jednocześnie. Schematycznie zostało to pokazane na **rysunku 6**. Porty komunikacji debugger → mikrokontroler (linie PGC i PGD) można wybrać w trakcie pracy nad projektem. Na płytce dsPIC33CH Curiosity Development Board umieszczono złącze J15 S1PGx3 (slave debug only) przeznaczone



Rysunek 4. Wybór mikrokontrolera



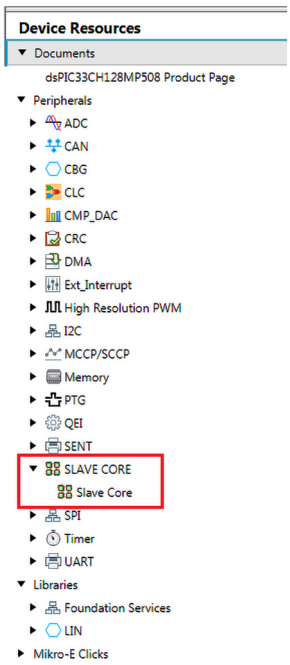
Rysunek 5. Wybór programatora/debuggera



Rysunek 6. Debugowanie kodu dla dwu rdzeni jednocześnie

do podłączenia debuggera przeznaczonego do debugowania kodu rdzenia Slave.

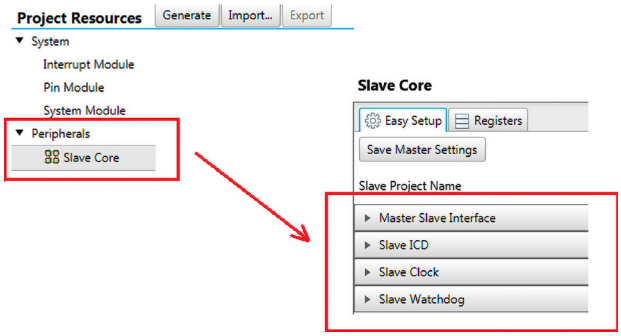
W kolejnych krokach kreatora wybieramy wersję kompilatora XC16. W trakcie pisania tego artykułu dla sdPIC33CH wymagana była wersja co najmniej v1.36. Wybór nazwy projektu i jego katalogu kończy pracę kreatora. Od jakiegoś czasu kreator projektu w MPLAB IDE nie tworzy szkieletu programu użytkownika i nie ma w nim żadnych plików źródłowych. Użytkownik może oczywiście samodzielnie umieszczać własne pliki źródłowe w strukturze projektu, ale lepszym pomysłem jest wykorzystanie do tego celu konfiguratora projektu dostępnego w MPLAB X IDE w postaci wtyczki MCC.



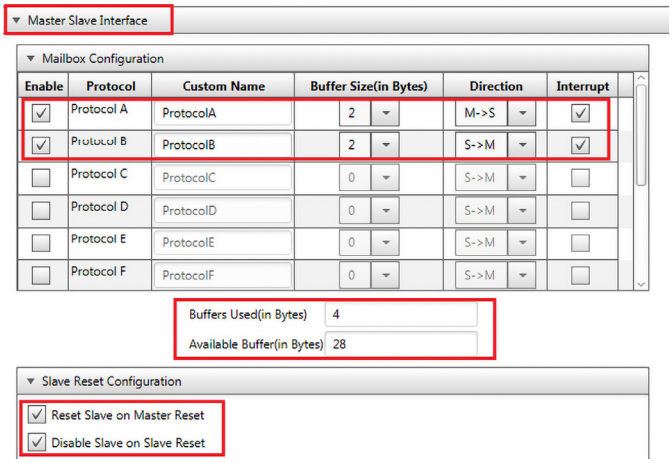
Rysunek 7. Lista układów peryferyjnych z elementem Slave Core

Konfigurowanie ustawień dla rdzenia Slave – element Slave Core

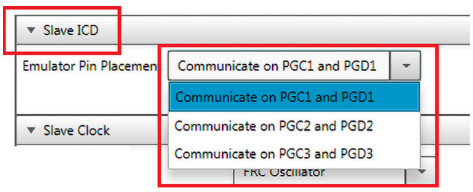
Wybranie mikrokontrolera z rodziny dsPIC33CH powoduje, że MCC „wie" o potrzebie konfigurowania projektu dla dwu rdzeni. Po uruchomieniu w oknie Device Resources → Peripherials na liście oprócz układów peryferyjnych dostępnych dla rdzenia Master jest umieszczony element Slave Core pozwalający skonfigurować część ustawień rdzenia Slave dla pracy dwuprocesorowej – **rysunek 7**.



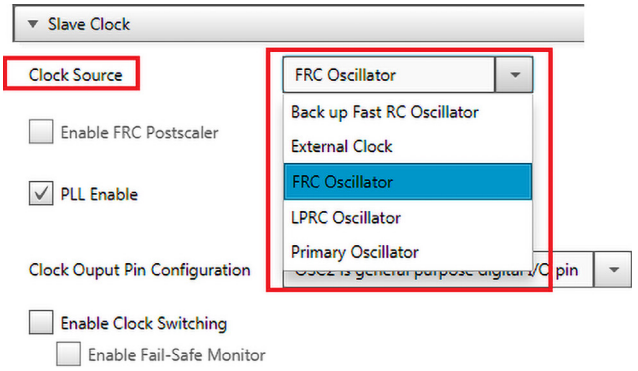
Rysunek 8. Konfigurowanie rdzenia Slave



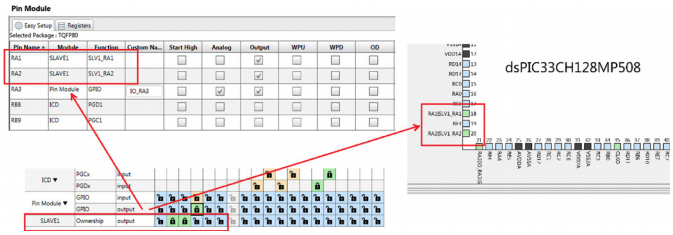
Rysunek 9. Konfigurowanie MSI i zerowania



Rysunek 10. Konfiguracja podłączenia debuggera rdzenia Slave



Rysunek 11. Konfiguracja taktowania



Rysunek 12. Konfigurowanie wyjść dla rdzenia Slave

Po dodaniu Slave Core do projektu MCC możemy konfigurować pracę rdzenia Slave za pomocą zakładek:

- Master Slave Interface – konfiguracja pracy interfejsu MSI,
- Slave ICD – wybór wyprowadzeń sygnałów debuggera ICD rdzenia Slave,
- Slave Clock – konfiguracja taktowania rdzenia Slave,
- Slave Watchdog – konfiguracja watchdoga.

Konfigurowanie MSI (**rysunek 9**) polega na wyborze kanału (kanałów), długości bufora do transmisji i kierunku przesyłania danych (Master → Slave lub Slave → Master). Zaznaczenie opcji Interrupt powoduje, że MSI będzie zgłaszał przerwania w trakcie pracy. Ważne jest też konfigurowanie zachowania się rdzenia Master po wykonaniu zerowania mikrokontrolera. Na rysunku 9 zerowanie zostało tak skonfigurowane, że zerowanie rdzenia Master powoduje również zerowanie rdzenia Slave i po zerowaniu rdzeń Slave jest blokowany – musi zostać odblokowany przez program wykonywany przez rdzeń Master.

Wspominana już możliwość niezależnego debugowania rdzenia Slave i podłączenia niezależnego debuggera jest konfigurowana w oknie Slave ICD – **rysunek 10**. Jeżeli chcemy debuggować rdzeń Slave, na płytce powinniśmy wybrać PGC3 i PGD3.

Układ taktowania rdzenia Slave ma swój własny preskaler i układ PLL. Źródło sygnału dla układu taktowania wybiera się w oknie Slave Clock. Można tu wybrać wewnętrzny szybki oscylator FRC, oscylator LPRC, sygnał z zewnętrznego generatora, wbudowanego oscylatora kwarcowego. Można również wyprowadzić sygnał taktujący na wyprowadzenie OSC2 mikrokontrolera (Clock Pin Configuration) i włączyć opcję Enable Clock Switching. Ta ostatnia po wykryciu braku sygnału zegarowego, na przykład z generatora zewnętrznego, lub oscylatora kwarcowego pozwala układowi taktowania na automatyczne przełączenie się na wewnętrzny oscylator FRC.

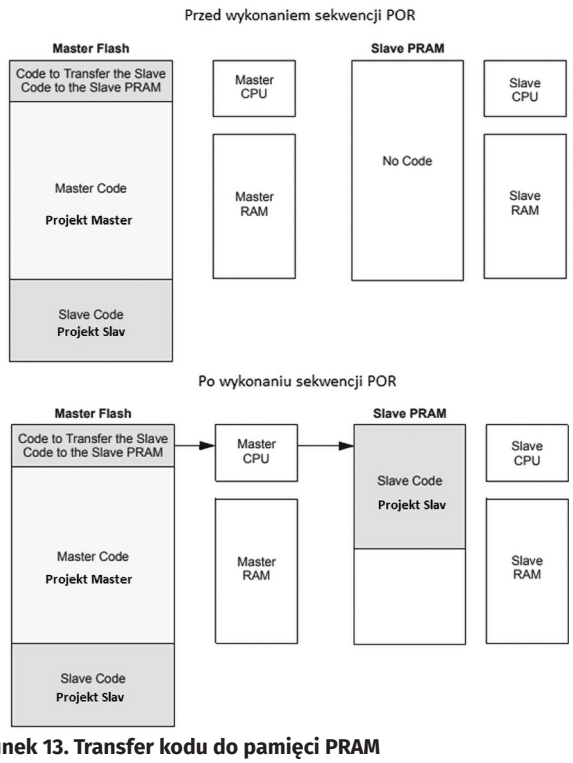
Ostatnią czynnością konfiguracyjną modułu Slave Core jest ustawienie działania układu watchdoga dla rdzenia Slave.

Wróćmy na chwilę do rysunku 2. Oba rdzenie mikrokontrolera są dość konsekwentnie izolowane od siebie. Wspólnym elementem jest interfejs komunikacyjny MSI (Master Slave Interface) oraz współdzielone wyprowadzenia mikrokontrolera. Współdzielenie jakichkolwiek zasobów jest zawsze źródłem potencjalnych konfliktów przy próbie uzyskania do nich dostępu. Jeżeli wyprowadzenie jest skonfigurowane jako wejściowe, to konflikty nie występują, bo oba rdzenie mogą bez zakłóceń jednocześnie odczytywać jego stan. W przypadku wyprowadzenia skonfigurowanego jako wyjściowe trzeba wykonać konfigurację przypisania wyprowadzenia wyjściowego dla rdzenia Slave, wykorzystując do tego celu element Pin Module konfiguratora MCC (**rysunek 12**).

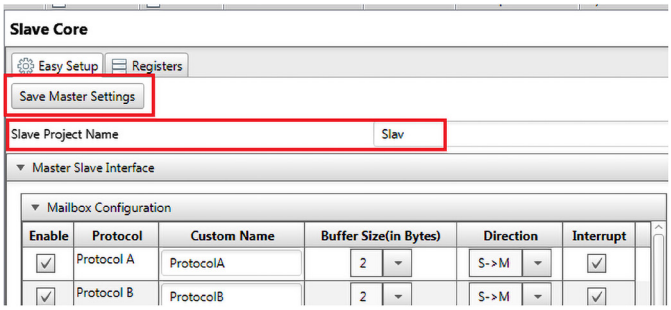
Wszystkie opisane konfiguracje rdzenia Slave reprezentowane w MCC za pomocą elementu Slave Core są zapisywane do bitów konfiguracyjnych (bezpieczników) i muszą być potem eksportowane do pamięci odpowiadającej za konfigurację rdzenia Slave.

Projekt dla rdzenia Slave

Dwa niezależne rdzenie wymagają projektu złożonego z dwu różnych, niezależnych od siebie projektów dla każdego z rdzeni. Możemy sobie wyobrazić przestrzeń adresową pamięci programu Flash mikrokontrolera z dwoma obszarami: pierwszy zawiera kod dla rdzenia



Rysunek 13. Transfer kodu do pamięci PRAM



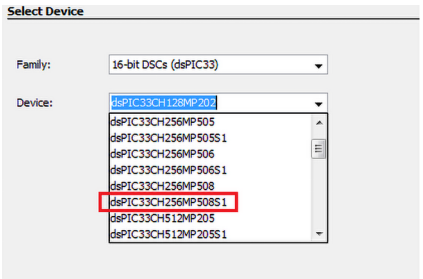
Rysunek 14. Zapisanie ustawień i nadanie nazwy projektowi dla rdzenia Slave

Master wygenerowany przez projekt o nazwie Master, drugi kod dla rdzenia Slave wygenerowany przez projekt o nazwie Slav. Zostało to pokazane na **rysunku 13**.

Przy odpowiedniej konfiguracji dwu połączonych projektów, projekt dla rdzenia Master umieszcza w pamięci Flash mikrokontrolera swój kod na początku przestrzeni adresowej, a za nim kod dla rdzenia Slave. Jeżeli korzystamy z MPLAB X IDE z wtyczką MCC i wykonamy konfigurację według przedstawionego przepisu, to wszystko powinno się wykonać automatycznie.

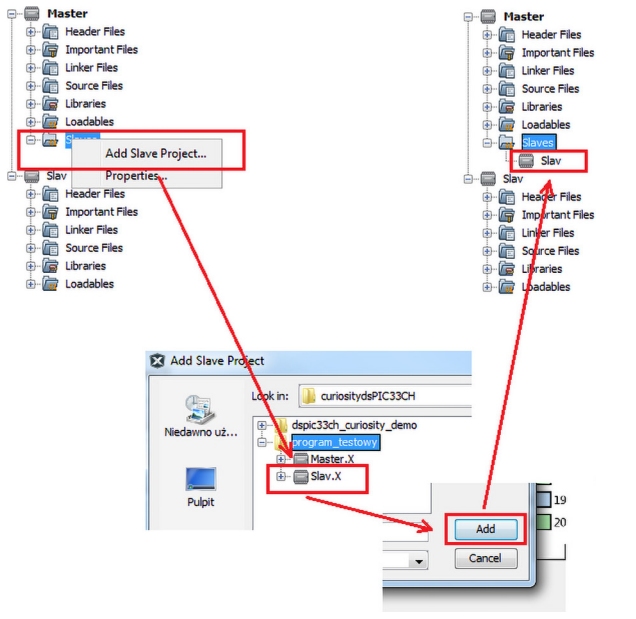
Procedurę łączenia dwu projektów zaczynamy od nadania nazwy projektu Slave w oknie Slave Project Name elementu Slave Core – w naszym przypadku będzie to nazwa Slav. Kiedy wszystkie ustawienia dotyczące rdzenia Slave są gotowe, to je zapisujemy, klikając na Save Master Settings – **rysunek 14**.

Teraz możemy się przygotować do dodania projektu dla rdzenia Slave. Najpierw musimy utworzyć projekt o nazwie takiej jaką nadaliśmy w oknie Slave Project Name – w naszym przypadku „Slav" – **rysunek 14**. Jak pamiętamy, dla takiego projektu musimy wybrać z listy mikrokontrolerów taki sam mikrokontroler, ale z sufiksem S1. W naszym przypadku będzie to dsPIC33CH256MP508S1 – **rysunek 15.**

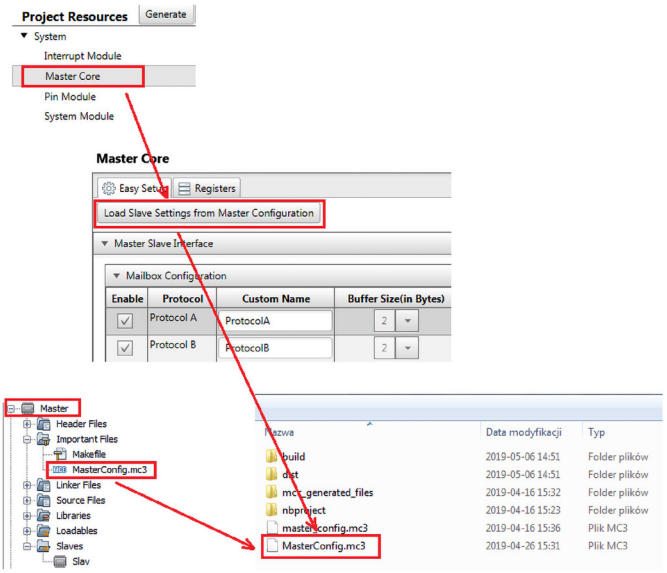


Rysunek 15. Wybór mikrokontrolera dla projektu dla rdzenia Slave

Jest to niezbędne, żeby MCC wiedział, że projekt ma być konfigurowany dla rdzenia Slave i udostępnił



Rysunek 16. Dodanie projektu Slav do projektu dla rdzenia Master



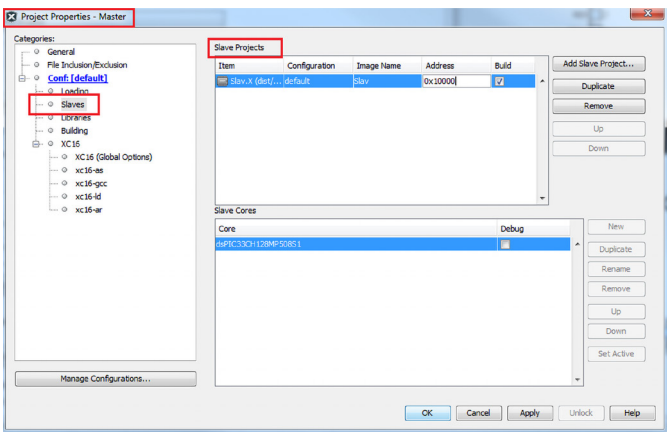
Rysunek 17. Wczytywanie konfiguracji rdzenia Slave ustawianej w projekcie dla rdzenia Master

mechanizmy wymiany konfiguracji pomiędzy oboma projektami. Kiedy projekt dla rdzenia Slave jest już utworzony (na tym etapie nie musi być konfigurowany), to można go dodać do projektu dla rdzenia Master. W projekcie Master MCC tworzy katalog Slave. Po kliknięciu prawym klawiszem myszy można dodać do niego wcześniej utworzony projekt o wymaganej nazwie Slav, tak jak to zostało pokazane na **rysunku 16**.

W tym momencie mamy dwa ściśle ze sobą powiązane projekty, które będą jednocześnie kompilowane z poziomu projektu Master. Jak wiemy, nowo utworzony projekt nie zawiera żadnych plików źródłowych. Przechodzimy teraz do projektu Slav i uruchamiamy MCC. Konfigurowanie układów peryferyjnych i taktowania odbywa się standardowo i jego opis pominę. Będzie nas za to interesowała konfiguracja modułu Master Core umieszczonego w oknie Project Recources. Jedyną czynnością, jaką możemy i powinniśmy tu wykonać, to wczytanie konfiguracji wykonywanej dla rdzenia Slave w wykonywaną w projekcie dla rdzenia Master. Dla przypomnienia ta konfiguracja obejmowała moduł Master Slave Interface, wyprowadzeń ICD, źródła sygnału taktowania i układu watchdoga. Na **rysunku 17** zostało pokazane wczytanie tej konfiguracji.

W oknie Master Core nie możemy niczego zmienić, możemy tylko obejrzeć to, co ustawiliśmy w projekcie Master. Żeby wykonać zmiany konfiguracji, trzeba to zrobić w projekcie dla Master i tam zapisać. Potem trzeba ponownie wczytać tę konfigurację do projektu Slave, tak jak to zostało pokazane powyżej.

Kiedy konfiguracja rdzenia Slave (projekt Slav) jest wykonana to uruchamiamy generowanie kodu przez MCC i teoretycznie oba projekty



Rysunek 18. kategoria Slaves w oknie właściwości projektu Master

są gotowe do kompilacji. Przed tym jeszcze trzeba przejść do właściwości projektu Master i wybrać kategorię Slaves. W oknie Slave Projects powinien się wyświetlić nasz dodany projekt Slav. Zaznaczamy tu opcję Build i w trakcie kompilacji najpierw zostanie skompilowany projekt Master, a po nim projekt Slav. W polu Address możemy podać adres, do którego linker umieści kod dla rdzenia Slave w pamięci Flash.

Przy ustawieniach z **rysunku 18** uruchomienie kompilacji projektu Master spowoduje skompilowanie obu projektów: Master i Slav i umieszczenie kodu dla procesora Slave w pamięci Flash od adresu 0x100000. Jeżeli chcemy, żeby kod dla Slave mógł być debugowany, to zaznaczamy opcję Debug w oknie Slave Cores.

Przykładowy program

Konfiguracja układów peryferyjnych, taktowania i watchdoga przez wtyczkę MCC to pierwszy i ważny krok przy pracy nad aplikacją. Pokażę teraz, jak wykorzystać skonfigurowane projekty do napisania prostej aplikacji wykorzystującej oba rdzenie i komunikację pomiędzy nimi. Komunikacja będzie się opierała na module MSI i mechanizmie wymiany danych przez mailbox.

Wróćmy na chwilę do rysunku 9 i okna Slave Reset Configuration. Ustawiliśmy zerowanie procesora Slave razem z zerowaniem procesora Master i blokowanie pracy procesora Slave po jego zerowaniu. Oznacza to, że wykonanie zerowania procesora Master automatycznie zeruje procesor Slave i jest on po tym zerowaniu zatrzymany. Wiemy też, że po procedurze zerowania procesora Master powinno być wykonane przepisanie kodu dla procesora Slave. Na rysunku 1 jest pokazany fragment programu procesora Master wykonujący się po zerowaniu. Po inicjalizacji układów peryferyjnych, układu przerwań i taktowania (SYSTEM_Initialize()) są wykonywane funkcje SLAVE1_Program() i SLAVE1_Start() – **listing 1**.

Obie te funkcje wywołują skompilowane funkcje biblioteczne, tak jak to zostało pokazane na **listingu 2**.

W tym momencie kod rdzenia Slave jest w jego pamięci programu PRAM i rdzeń pracuje. Rozpoczynamy programowanie transmisji przez Mailbox z Master do Slave.

Listing 1. Inicjalizacja mikrokontrolera, przepisanie kodu do PRAM i uruchomienie rdzenia Slave

```
int main(void)
{
    // inicjalizacja mikrokontrolera
    SYSTEM_Initialize();
//przepisanie kodu rdzenia Slave do PRAM
    SLAVE1_Program();
//uruchomienie kodu rdzenia Slave
    SLAVE1_Start();
```

Listing 2. Funkcje SLAVE1_Start() i Slave1_Program()

```
void SLAVE1_Start()
{
    _start_slave();
}

void SLAVE1_Program()
{
    _program_slave(SLAVE_NUMBER,0,SLAVE_IMAGE);
}
```

Listing 3. Deklaracje i inicjalizacja buforów mailbox

```
ProtocolA_DATA dataSend;        //deklaracja struktury bufora nadawania MSI
ProtocolB_DATA dataReceive;     //deklaracja struktury bufora odbiou MSI

dataSend.ProtocolA[0] = 0xaaaa;
dataReceive.ProtocolB[0] = 0;
```

Listing 4. Funkcja zgłaszania przerwania przez Master

```
void SLAVE1_InterruptRequestGenerate()
{
    MSI1CONbits.MSTIRQ = 1;
}
```

Listing 5. Ustawienie MTSIACK

```
void MASTER_InterruptRequestAcknowledge()
{
    SI1CONbits.MTSIACK = 1;
}
```

Listing 6. Sekwencja wysyłania danych przez Master

```
SLAVE1_ProtocolAWrite((ProtocolA_DATA*)&dataSend);          //wysłanie danej przez Mailbox

    SLAVE1_InterruptRequestGenerate();                      //zgłoszenie przerwania
    while(!SLAVE1_IsInterruptRequestAcknowledged());        //czekanie na potwierdzenia
    SLAVE1_InterruptRequestComplete();                      //zakończenie sekwencji zgłoszenia przerw.
    while(SLAVE1_IsInterruptRequestAcknowledged());         //czekanie na potwierdzenie
```

Na **listingu 3** są pokazane deklaracje struktur bufora nadawania i odbioru, inicjalizacja bufora nadawania i zerowanie bufora odbioru.

Teraz musimy wykonać sekwencję wymiany danych pomiędzy rdzeniami. Wykorzystamy protokół wymiany danych oparty na mechanizmie przerwań. Pierwszym krokiem sekwencji jest wysłanie danych do bufora Mailbox przez protokół A skonfigurowany do transmisji Master → Slave. Wykonuje to funkcja SLAVE1_protocolAWrite ().

Procesor Master może zgłaszać przerwanie bezpośrednio do procesora Slave przez ustawienie bitu MTSIRQ. Również procesor Slave może zgłaszać bezpośrednio przerwanie do procesora Master przez ustawienie bitu STMIRQ.

Po zapisaniu bufora Master zgłasza przerwanie przez ustawienie bitu MTSIRQ – **listing 4**.

Po ustawieniu MTSIRQ procesor Master synchronizuje zegar taktujący transmisją z zegarem procesora Slave. Procesor Slave generuje przerwanie przez ustawienie bitu MSIMIF. Teraz Slave musi potwierdzić rdzeniowi Master przyjęcie przerwania przez ustawienie bitu MTSIACK. Na **listingu 5** jest pokazana funkcja void MASTER_InterruptRequestAcknowledge() ustawiająca MTSIACK.

Listing 7. Sekwencja odbierania danych przez Slave

```
    while(!MASTER_IsInterruptRequested()); //czekaj na zgłoszenie przerwania
    MASTER_InterruptRequestAcknowledge();//wyślij potwierdzenie
    while(MASTER_IsInterruptRequested());//czekaj na zakończenie sekwencji
    MASTER_InterruptRequestAcknowledgeComplete();//wyślij potwierdzenie

    //odczytanie bufora mailbox
    MASTER_ProtocolARead((ProtocolA_DATA*)&dataReceive);
```

Listing 8. Przesłanie danych przez rdzeń Slave do rdzenia Master

```
  //zapisanie bufora mailbox ProtocolB
    MASTER_ProtocolBWrite((ProtocolB_DATA*)&dataSend);

    MASTER_InterruptRequestGenerate();                      //zgłoszenie przerwania
    while(!MASTER_IsInterruptRequestAcknowledged());        //czekanie na potwierdzenie
    MASTER_InterruptRequestComplete();                      //kończenie sekwencji
    while(MASTER_IsInterruptRequestAcknowledged());         //oczekiwanie na potwierdzenie
```

Listing 9. Odczytanie danych z rdzenia Slave

```
    while(!SLAVE1_IsInterruptRequested());        //czekaj na przerwanie od Slave
    SLAVE1_InterruptRequestAcknowledge();         //wyślij potwierdzenie
    while(SLAVE1_IsInterruptRequested());         //czekaj na zakończenie sekwencji
    SLAVE1_InterruptRequestAcknowledgeComplete(); //wyślij potwierdzenie

    //odczytaj bufor ProtocolB
    SLAVE1_ProtocolBRead((ProtocolB_DATA*)&dataReceive);
```

Na **listingu 6** przedstawiono fragment programu realizujący przesłanie danych z rdzenia Master do rdzenia Slave wykorzystujący protokół oparty na zgłaszaniu przerwań i przesyłaniu potwierdzeń.

Po stronie rdzenia Slave musimy czekać na zgłoszenie przerwania, wysłać potwierdzenie, czekać na zakończenie sekwencji i ponownie wysłać potwierdzenie. Jeżeli wszystko pójdzie dobrze, to można przeczytać dane z bufora ProtocolA Mailbox – **listing 7**.

W trakcie konfiguracji MSI został zdefiniowany drugi kanał transmisji ProtocolB od Slave do Master. Możemy teraz wysłać dane ze Slave do Master. Sekwencja jest praktycznie taka sama, różni się tylko bitami, które trzeba ustawić i testować. Wysyłanie danych przez Slave rozpoczyna się od zapisania bufora danych kanału ProtocolB, a potem jest zgłaszane przerwanie do Master, oczekiwanie na potwierdzenie i zakończenie sekwencji – **listing 8**.

Te dane rdzeń Master odbiera według takiej samej zasady. Najpierw czeka na przerwanie, potem wysyła potwierdzenie, czeka na kończenie sekwencji i wysyła ponowne potwierdzenie – **listing 9**.

Kompletny program wykonuje następujące czynności:

- Master przesyła do rdzenia Slave 16-bitową wartość,
- Slave odsyła do rdzenia Master 16-bitową wartość,
- Slave sprawdza, czy odebrana wartość jest zgodna z oczekiwaną – jeżeli tak to zapala diodę D2,
- Master sprawdza, czy odebrana wartość jest zgodna z oczekiwaniem i jeżeli tak, to zapala diodę D1.

Praktyczne próby przeprowadzone z modułem dsPIC33CH Curiosity Development Board zakończyły się sukcesem. Opisane powyżej tworzenie projektów dla rdzenia Master i rdzenia Slave i konfigurowanie obu rdzeni za pomocą MCC pozwoliły na napisanie prostej aplikacji „dwurdzeniowej" wykorzystującej komunikację w protokole wykorzystującym przerwania i moduł Master Slave Interface pracującego jako Mailbox. Wszystko to daje solidne podstawy do pisania własnych, bardziej lub mniej rozbudowanych aplikacji wykorzystujących dwa wydajne rdzenie i możliwość wymiany danych pomiędzy nimi. Oczywiście program testowy miał za zadanie sprawdzić możliwość działania mikrokontrolera i w praktyce może być trudny do zastosowania w bardziej wymagających aplikacjach. Jego wadą jest blokowanie programu w pętlach nieskończonych przy testowaniu oczekiwania na przerwania lub na potwierdzenia. To bardzo prosty sposób na zablokowanie działania aplikacji, jak coś pójdzie nie tak jak powinno. Najlepiej gdyby transmisja pomiędzy rdzeniami całkowicie opierała się na przerwaniach i pracowała w tle programów głównych.

Jest to moje pierwsze praktyczne zetknięcie się z mikrokontrolerem dwurdzeniowym. Jestem pod wrażeniem pracy, jaką Microchip wykonał, żeby można było w miarę szybko skonfigurować projekty dla obu rdzeni i uruchomić między nimi komunikację. Co równie ważne, zmiany w konfiguracjach są łatwe i szybkie do wykonania. Tradycyjnie oprócz sprzętu (modułu z wbudowanym lub zewnętrznym programatorem/debuggerem) wszystko jest dostępne i nieodpłatne. dsPIC33CH jest specyficzną rodziną, nastawiona na wykonywanie algorytmów sterowania, w tym sterowania silnikami DC i układami przetwornic DC/DC. Mam nadzieję, że w miarę szybko rozwiązanie to zacznie migrować do uniwersalnych 16-bitowych jednostek PIC24 i stanie się bardziej popularne i coraz tańsze.

Tomasz Jabłoński