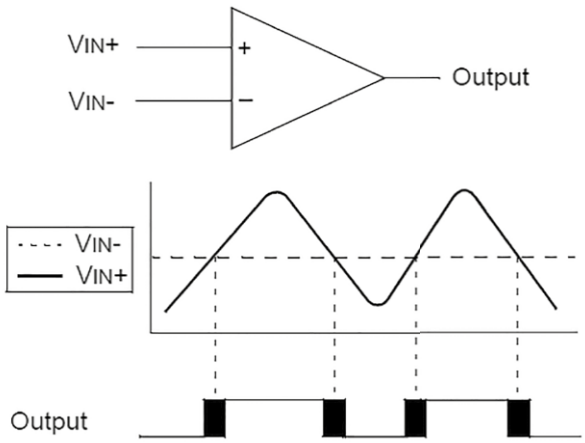
8-bitowa kontrofensywa (2)

Komparator analogowy i przetwornik A/C

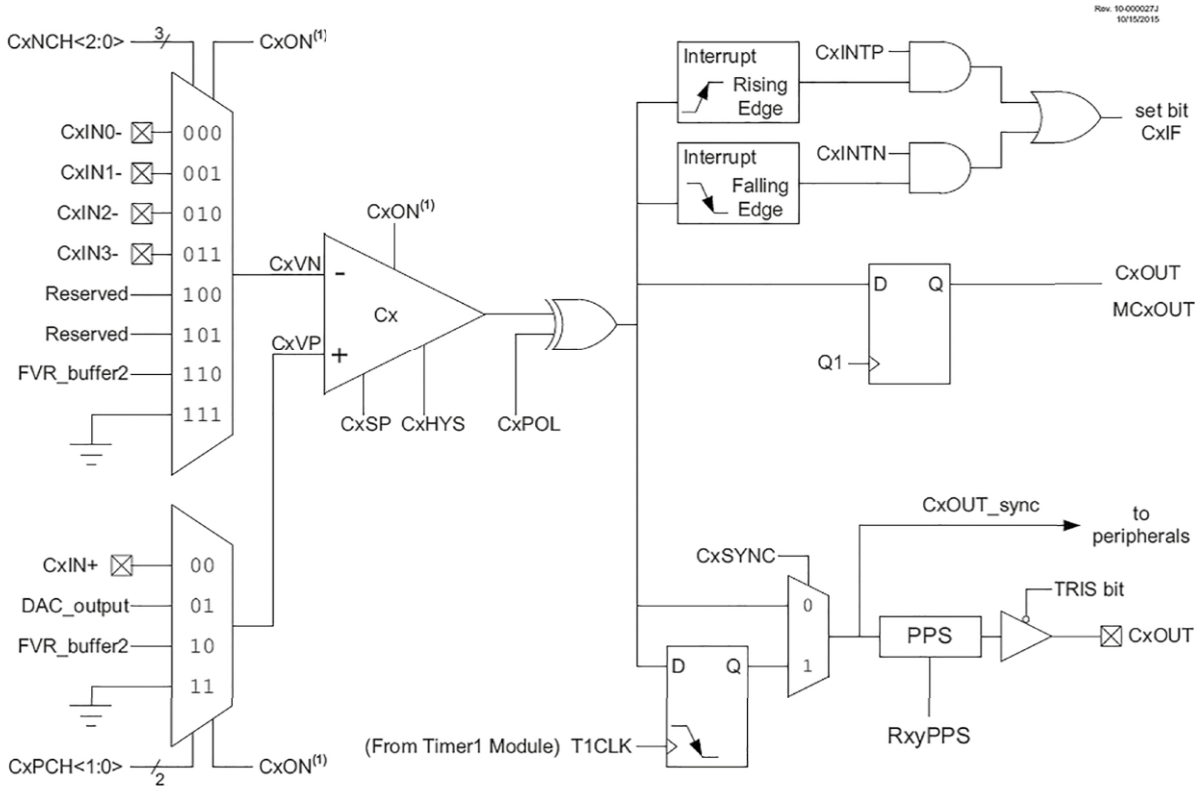
Firma Microchip, jeden z największych na świecie graczy w obszarze projektowania i produkcji mikrokontrolerów, wymyślił strategię, dzięki której użytkownicy poszukujący nieskomplikowanych mikrokontrolerów chętnie sięgną po 8-bitowe, sprawdzone mikrokontrolery PUC. Zastosowano pomysłowe połączenie starego, ale sprawdzonego i bardzo popularnego rdzenia PIC16 z nietypowymi, nowatorskimi peryferiami, pracującymi niezależnie od rdzenia (core independent). Ta „niezależność" polega na tym, że działanie układów peryferyjnych nie zależy od częstotliwości taktowania mikrokontrolera i ich praca nie obciążą CPU. Mogą zatem wykonywać pewne specyficzne czynności dużo szybciej niż rdzeń, a jedyne obszary wspólne, to rejestry konfiguracyjne i rejestry danych.



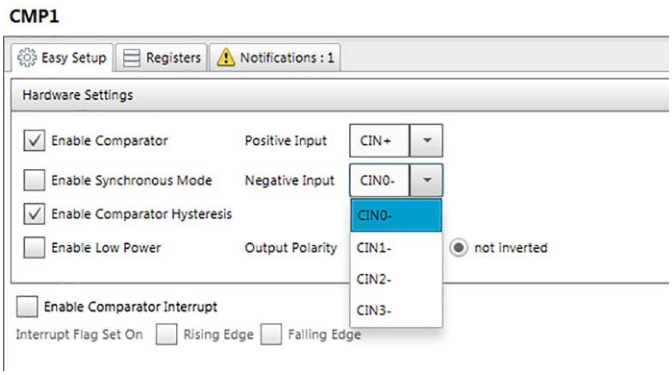
Rysunek 1. Działanie komparatora

Komparator analogowy jest blokiem przeznaczonym do porównania dwóch napięć, a wynik porównania jest obrazowany za pomocą poziomu logicznego (**rysunek 1**). Komparator nie zastąpi przetwornika A/C, ale świetnie spełni swoje zadanie w aplikacjach, w których jest konieczne tylko wykrywanie progu przekroczenia napięcia. Komparator wykona tę czynność bardzo szybko a jego działanie jest niezależne od działania rdzenia. Typowe zastosowania to:

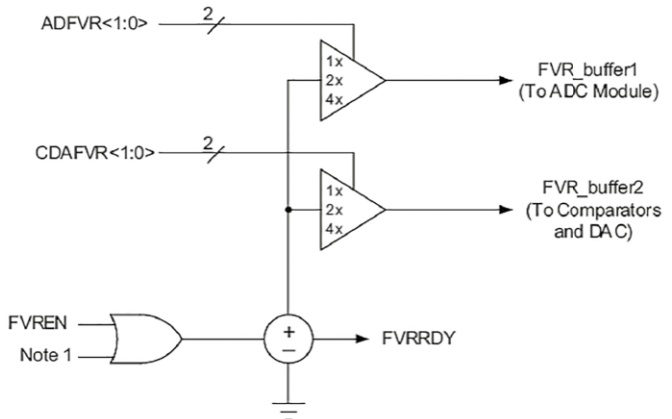
- Wykrywanie zbyt wysokiego lub zbyt niskiego napięcia w układach zasilania.
- Wykrywanie przeciążenia w układach sterowania silnikami elektrycznymi.
- Wykrywanie niskiego napięcia baterii.
- Zasilacze impulsowe (wykrywanie impulsów prądowych).



Rysunek 2. Moduł komparatora mikrokontrolerów PIC16F1xxx



Rysunek 3. Okno MCC Easy Setup konfigurowania komparatora



Rysunek 4. moduł napięcia referencyjnego FVR

- Komparator analogowy w układach analogowych kontrolowanych za pomocą mikrokontrolera.

Typowy moduł komparatora zaimplementowany w mikrokontrolerach z rodziny PIC16F1xxx jest wyposażony w dwa multipleksery wejściowe (po jednym dla każdego wejścia), jak pokazano na **rysunku 2**. Napięcie wejściowe jest pobierane z wyprowadzeń mikrokontrolera, wyjścia modułu wewnętrznego napięcia referencyjnego FVR lub z wyjścia przetwornika C/A. Ten przetwornik, zależnie od typu mikrokontrolera, może być 5-, 8- lub 10-bitowy. Wynik porównania może być:

- Przesłany do wyprowadzenia mikrokontrolera na wyjście CxOUT.
- Dostępny jako sygnał wewnętrzny CxOUT. Ten sygnał można łączyć wewnętrznie z innymi modułami peryferyjnymi.
- Dostępny jako flaga generowania przerwania CxIF. Przerwanie jest wyzwalane narastającym zboczem, opadającym zboczem (lub jednym i drugim). Ustawienie flagi CxIF może tez wybudzać mikrokontroler ze stanu uśpienia.

Nawet tak prosty w działaniu moduł wymaga skonfigurowania: trzeba wybrać sygnały wejściowe, ustalić polaryzację sygnału wyjściowego (wprost/zanegowany), skonfigurować polaryzację sygnału wyzwalania przerwania, włączyć lub wyłączyć histerezę przełączania, zaprogramować przypisanie wyjść sygnału CxOUT (moduł PPS) i w końcu włączyć moduł.

Na **rysunku 3** pokazano okno Easy Setup wtyczki MCC dające możliwość użycia następujących opcji:

- Enable Comparator – włącza i wyłącza moduł komparatora.
- Enable Synchronous Mode – umożliwia pracę synchronicznie z taktowaniem rdzenia.
- Enable Comparator Hysteresis – włączenie histerezy progu komparatora. Histereza ma stałe napięcie 25 mV.
- Enable Low Power – włączenie trybu oszczędzania energii.
- Output Polarity – zanegowany/niezanegowany.
- Wejście dodatnie Positive Input – wybór z rozwijanej listy sygnału wejściowego na wejściu dodatnim komparatora.
- Wejście ujemne Negative Input – wybór z rozwijanej listy sygnału wejściowego na wejściu dodatnim komparatora.
- Enable Comparator Interrupt – odblokowanie zgłaszania przerwań i wybór zbocza.

MCC generuje dwie funkcje:

1. *CMP1_Initialize()* inicjalizującą moduł komparatora zgodnie z wykonanymi ustawieniami (**listing 1**).
2. *CMP1_GetOutputStstus()* odczytującą stan wyjścia komparatora (**listing 2**).

Moduł Curiosity jest wyposażony w potencjometr i za jego pomocą można przetestować działanie komparatora. Napięcie z wyjścia komparatora będzie porównywane z napięciem referencyjnym *Fixed Voltage Reference* (FVR).

Moduł napięcia referencyjnego FVR

Napięcie referencyjne jest używane przez moduły komparatora i przetworników A/C i C/A. W układach niewymagających dużej dokładności pomiaru napięciem referencyjnym dla przetworników jest napięcie zasilania Vdd. Jeżeli jest potrzebna większa dokładność, to precyzyjne napięcie referencyjne może być podawane z zewnątrz przez odpowiednie wejście.

W mikrokontrolerach PIC16F1xxx wbudowano moduł napięcia referencyjnego FVR. Jest on zbudowany z dwóch bloków: źródła dokładnego i stabilnego napięcia o wartości nominalnej 1,024 V i dwóch programowanych wzmacniaczy napięcia stałego o wzmocnieniach x1, x2 i x4 (**rysunek 4**). Te wzmacniacze spełniają dwie funkcje. Pierwsza, to obniżenie wysokiej impedancji źródła (band-gap). Druga, to umożliwienie uzyskania 3 różnych napięć: 1,024 V, 2048 V i 4,096 V dzięki

Listing 1. Inicjalizacja komparatora z przykładowymi ustawieniami

```
void CMP1_Initialize(void)
{
    // set the CMP to the options selected in MPLAB(c) Code Configurator
    // C1HYS enabled; C1SP hi_speed; C1ON enabled; C1POL not inverted; C1SYNC asynchronous;
    CM1CON0 = 0x86;
    // C1INTN no_intFlag; C1INTP no_intFlag; C1PCH Vss; C1NCH CIN0-;
    CM1CON1 = 0x30;
}
```

Listing 2. Funkcja zwracająca stan wyjścia komparatora

```
bool CMP1_GetOutputStatus(void)
{
    return (CMOUTbits.MC1OUT);
}
```

Listing 3. Funkcje konfiguracji i testowania gotowości modułu FVR

```
void FVR_Initialize(void)
{
    // CDAFVR 2x; FVREN enabled; TSRNG Lo_range; ADFVR off; TSEN disabled;
    FVRCON = 0x88;
}

bool FVR_IsOutputReady(void)
{
    return (FVRCONbits.FVRRDY);
}
```

wzmocnieniu napięcia. Każdy ze wzmacniaczy może być włączany i programowany indywidualnie. Pierwszy z nich jest źródłem napięcia odniesienia dla przetwornika A/C, a drugi dla komparatorów lub przetwornika C/A. Konfiguracja FVR jest zapisywana w rejestrze FVRCON. Tu skonfigurujemy moduł za pomocą MCC. Ponieważ w tym momencie potrzebne nam będzie napięcie referencyjne dla komparatora, to zaprogramujemy napięcie 2,048 V w FVR_buffer2 (wzmocnienie2) oraz wyłączymy FVR_buffer1 (**rysunek 5**).

MCC generuje dwie funkcje: konfiguracyjną *FVR_initialize()* i zwracająca status *FVR_IsOutputReady()*. Pokazano je na **listingu 2**.

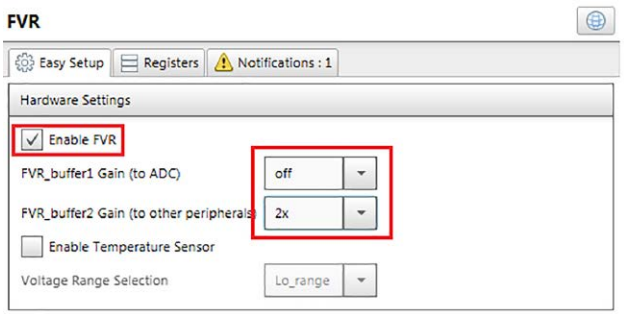
Moduł napięcia referencyjnego jest bardzo wygodny w użyciu, ale trzeba pamiętać o ograniczeniach. Dokładność napięcia referencyjnego zależy od temperatury otoczenia i jakości napięcia zasilającego. To wszystko powoduje, że występują fluktuacje napięcia na poziomie 1…2%. Należy to wziąć pod uwagę używając FVR jako źródła napięcia referencyjnego dla 10-bitowego przetwornika A/C. Niedokładność tego napięcia będzie wyższa niż niedoskonałości samego przetwornika (offset, nieliniowość przetwarzania). Może się więc zdarzyć, że w pewnych aplikacjach wymagających dużej precyzji pomiarów trzeba będzie użyć zewnętrznego źródła napięcia referencyjnego.

Komparator + FVR – testy praktyczne

Mamy skonfigurowany komparator z napięciem FVR (bufor2) dołączonym do wejścia dodatniego. Napięcie FVR jest równe 2,048 V. Wejście ujemne CIN1- przypisałem do wyprowadzenia RC1 (nóżka 15). Trzeba teraz suwak potencjometru połączyć z CIN1- i testować stan wyjścia komparatora.

W module Curiosity suwak potencjometru jest domyślnie połączony poprzez rezystor 0 Ω (R33) z wyprowadzeniem RC0, ponieważ może ono być skonfigurowane w roli wejścia przetwornika A/C. W naszym układzie testowym trzeba usunąć to połączenie przez wylutowanie R33 i połączyć suwak potencjometru z portem RC1.

Program główny uzupełniamy o pętle nieskończoną pokazaną na **listingu 3**. Funkcja *CMP1_GetOutputStatus()* odczytuje i zwraca stan wewnętrznego sygnału MC1OUT (rys. 2). W module Curiosity zamontowano 4 diody świecące połączone przez rezystory 1 kΩ z liniami portów RA5, RA2, RC5 i sygnałem PGEC (zegar interfejsu programującego ICSP). Ustawienie linii sterującej diodą powoduje jej zaświecenie, więc jeśli na RA2 wystąpi poziom wysoki i ta linia jest skonfigurowana jako wyjście, to LED D6 się zaświeci. Jeśli funkcja *CMP1_GetOutputStatus* zwróci prawdę, czyli napięcie na wejściu ujemnym połączonym z suwakiem potencjometru będzie mniejsze od napięcia z modułu FVR równego 2,048 V, to dioda LED pozostanie zaświecona.



Rysunek 5. Konfigurowanie FVR

Można zdefiniować negowanie wyjścia komparatora i dioda będzie się zaświecała przy napięciu wyższym od napięcia VFR. Mikrokontroler w module jest zasilany napięciem +3,3 V. Wykonałem 2 próby: dla napięcia referencyjnego 1,024 V i napięcia 2,048 V. W obu wypadkach komparator działał prawidłowo.

Przetwornik A/C

Parametrem przetwornika A/C, na który przede wszystkim zwracamy uwagę, jest jego rozdzielczość bitowa. Teoretycznie – im większa rozdzielczość, tym lepiej, ale w praktyce duża rozdzielczość może być degradowana przez błędy przetwornika wynikające z, na przykład, nieliniowości przetwarzania lub z offsetu. Mikrokontrolery z rodziny PIC16F1xxx mają wbudowany przetwornik o rozdzielczości 10 bitów i małym błędzie przetwarzania wynoszącym typowo ±1 LSB, nieprzekraczającym ±2,5 LSB dla maksymalnego błędu wzmocnienia.

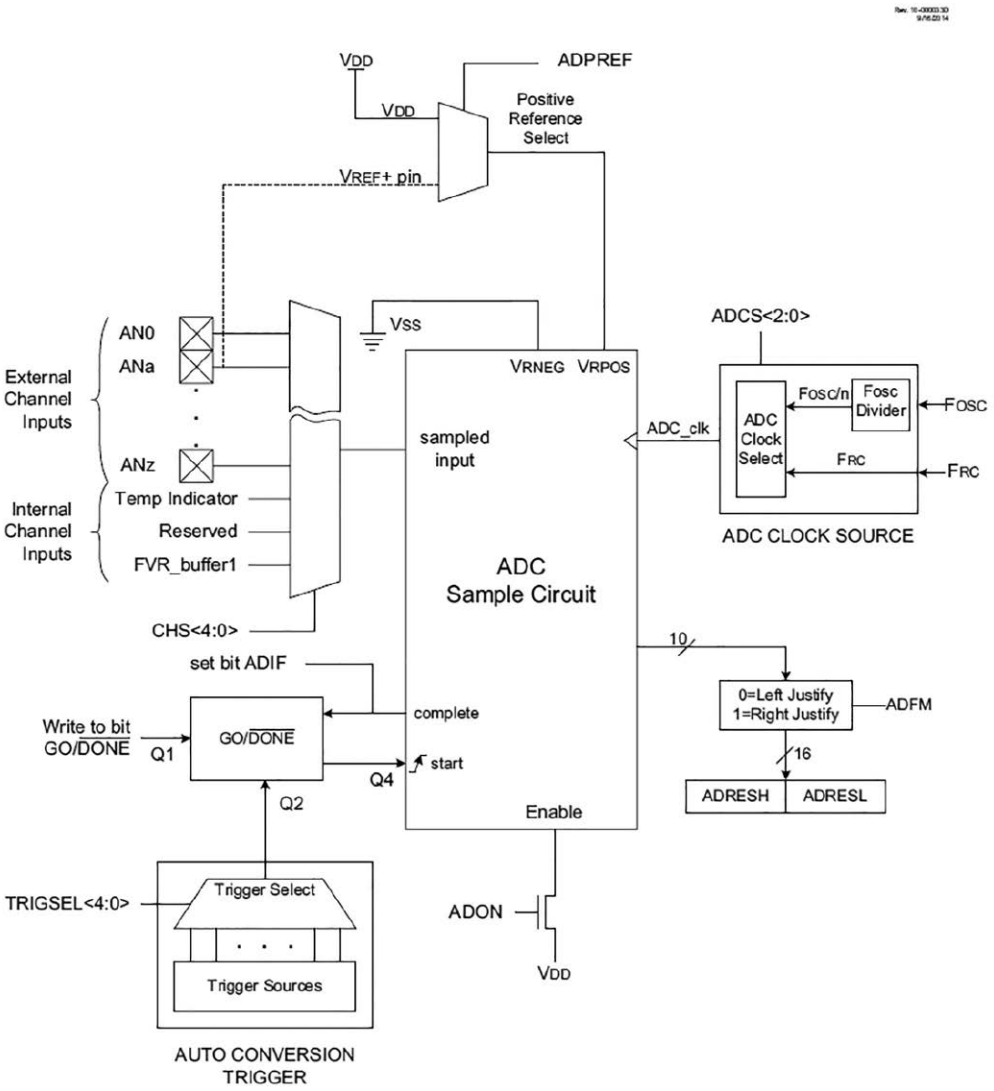
Kolejnym ważnym parametrem jest liczba wejść analogowych W naszym mikrokontrolerze może ich być 12. Poza tymi wejściami przetwornik może mierzyć napięcie z wyjścia AN1 przetwornika cyfrowo analogowego C/A, modułu pomiaru temperatury i modułu napięcia referencyjnego *FVR_buffer_1*.

Przetwornik używa wejścia napięcia referencyjnego dodatniego Vrpos i ujemnego Vrneg. Napięciem dodatnim może być napięcie podane na jedno z wybranych wejść analogowych, napięcie zasilania Vdd lub napięcie z modułu FVR_buffer_1. W przetworniku zastosowano konwerter SAR. Prędkość konwersji (maksymalnie 100 kSa/s) jest dostosowana do wydajności 8-bitowego rdzenia. Próbkowane napięcie jest dołączane przez multiplekser do wejściowego kondensatora o małej pojemności. Po ustawieniu bitu GO w rejestrze ADCON lub wystąpieniu zdarzenia wyzwalającego konwersję, kondensator jest odłączany od wejścia (mierzonego źródła napięcia) i jest wykonywana konwersja analogowo-cyfrowa. Wynik konwersji jest zapisywany w 16-bitowym rejestrze ADRES. Źródłem taktowania przetwornika może być wewnętrzny oscylator RC (Frc) o stałej częstotliwości 500 kHz. Taki tryb taktowania pozwala na pracę przetwornika w trybie ograniczonego poboru mocy (sleep), gdy wszystkie inne źródła taktowania są wyłączane. Taktowanie może też być wykonywane z użyciem podzielonego sygnału zegarowego taktującego rdzeń. Trzeba jednak pamiętać, aby częstotliwość nie była wyższa niż maksymalna częstotliwość taktowania przetwornika (okres Tad≥1 μs).

Chociaż programowe wyzwolenie przetwarzania wymaga działania rdzenia (ustawienie bitu GO), to można alternatywnie użyć wyzwalania sprzętowego. Uzyskuje się wtedy niezależną synchronizację pomiędzy konwersją, a innymi peryferiami. Dzięki tej właściwości jest możliwe zbudowanie kompletnego rozbudowanego połączenia pomiędzy peryferiami analogowymi i cyfrowymi pracującego niezależnie od rdzenia mikrokontrolera. Przy projektowaniu układu pomiarowego należy pamiętać, że przetwornik ma niską impedancję wejściową. Rekomendowana przez producenta

impedancja źródła mierzonego napięcia nie powinna być większa niż 10 kΩ.

Konfigurowanie przetwornika poprzez MCC można podzielić na dwa etapy. Pierwszy z nich to konfiguracja modułu przetwornika, a drugi konfiguracja wejścia analogowego. Na **rysunku 7** pokazano okno **ADC Easy Setup**. Wybieramy tu: włączenie modułu przetwornika (enable ADC), wybór zegara taktującego (clock source), sposób zapisania 10-bitowego wyniku konwersji w rejestrze 16-bitowym (dosunięty do lewej lub dosunięty do prawej) oraz sposób wyzwalania konwersji automatycznej (tu wyjście CMP1). Dodatkowo, można odblokować zgłaszanie przerwań od zdarzenia zakończenia konwersji. W zakładce *ADC registers* są powtórzone ustawienia z Easy Setup, ale można tu dodatkowo wybrać aktywne wejście analogowe. Oczywiście, kiedy nasza aplikacja tego wymaga, to program użytkownika może wybierać inne dostępne wejścia według potrzeb.



Rysunek 6. Schemat blokowy przetwornika analogowo – cyfrowego

Wybór aktywnego wejścia analogowego musi być połączony ze skonfigurowaniem linii portu w roli wejścia analogowego. Najlepiej jest to zrobić korzystając z okna konfiguracji linii portów. Po dodaniu modułu ADC, MCC „wie", że określone linie portów mogą być wejściami analogowymi i daje możliwość zaprogramowania ich do tej funkcji. Po kliknięciu na symbol niebieskiej, otwartej kłódki w pozycji wiersza **ADC –> ANx** zmienia się symbol na kłódkę zamkniętą w kolorze zielonym. Linia portu zostaje przeprogramowana na wejście analogowe (**rysunek 9**).

Listing 4. Pętla testowania pracy komparatora

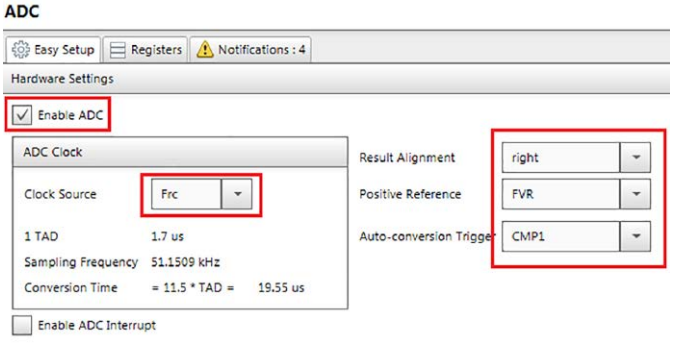
```
while (1)
{
    LATAbits.LATA2=CMP1_GetOutputStatus();
}
```

Listing 5. Inicjalizacja modułu przetwornika analogowo – cyfrowego

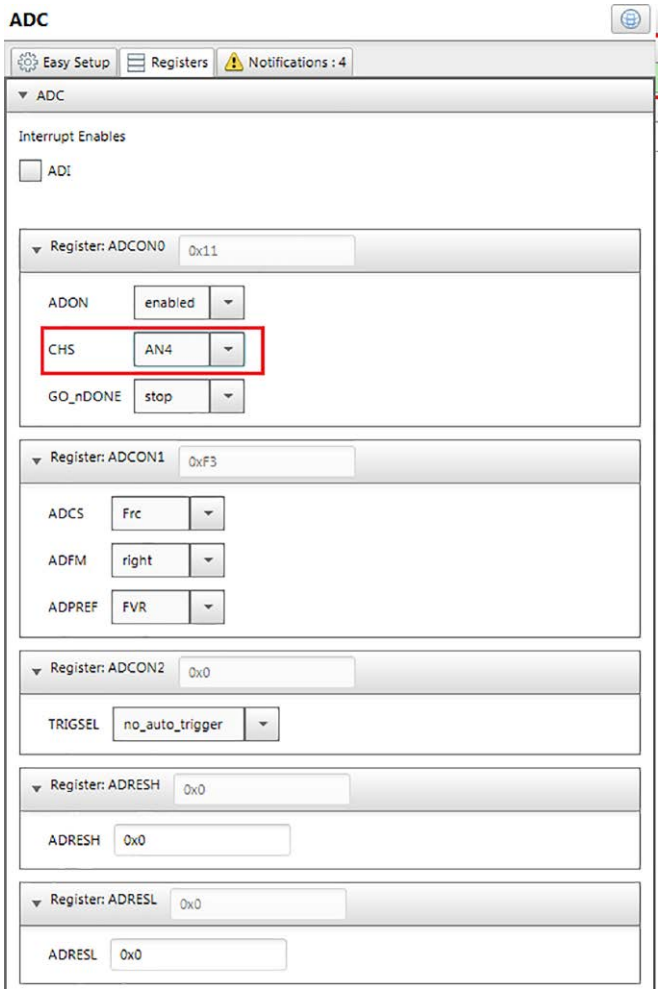
```
void ADC_Initialize(void)
{
    // set the ADC to the options selected in the User Interface
    // GO_nDONE stop; ADON enabled; CHS AN4;
    ADCON0 = 0x11;
    // ADFM right; ADPREF FVR; ADCS Frc;
    ADCON1 = 0xF3;
    // TRIGSEL no_auto_trigger;
    ADCON2 = 0x00;
    // ADRESL 0;
    ADRESL = 0x00;//zerowanie rejestru wyniku konwersji
    // ADRESH 0;
    ADRESH = 0x00;

}
```

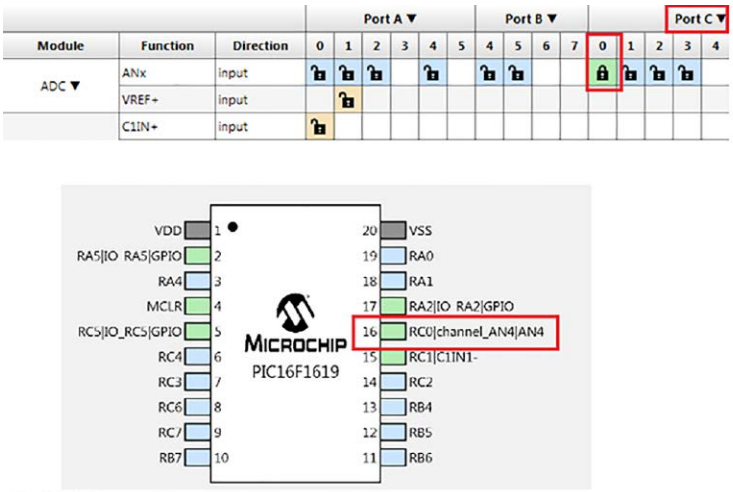
Po skonfigurowaniu modułu MCC generuje kilka przydatnych funkcji. Pierwszą z nich jest inicjalizacja modułu według wprowadzonych ustawień – **listing 4**. Pokazana na **listingu 5** funkcja *ADC_StartConversion (adc_channel_t channel)* wybiera aktywne wejście analogowe i programowo wyzwala konwersję. Kolejna funkcja – *adc_result_t ADC_GetConversion(adc_channel_t channel)* – wybiera aktywne wejście, wyzwala programowo konwersję, czeka na jej zakończenie i zwraca wynik w postaci liczby 16 bitowej (**listing 7**).



Rysunek 7. Konfiguracja modułu przetwornika analogowo – cyfrowego



Rysunek 8. Konfigurowanie przetwornika i wybór wejścia analogowego



Pin Module

Easy Setup | Registers | Notifications : 2

Selected Package : PDIP20

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD
RA2	Pin Module	GPIO	IO_RA2	☐	☐	☑	☑	
RA5	Pin Module	GPIO	IO_RA5	☐		☑	☑	
RC0	ADC	AN4	channel_AN4	☐	☑	☐	☑	
RC1	CMP1	C1IN1-		☐	☑	☐	☑	
RC5	Pin Module	GPIO	IO_RC5	☐		☑	☑	

Rysunek 9. Zaprogramowanie RC0 jako wejście analogowe AN4

Listing 6. Wybranie wejścia i start konwersji

```
#define ACQ_US_DELAY 5
void ADC_StartConversion(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;
    // Turn on the ADC module
    ADCON0bits.ADON = 1;
    // Acquisition time delay
    __delay_us(ACQ_US_DELAY);
    // Start the conversion
    ADCON0bits.GO_nDONE = 1;
}
```

Listing 7. Wybranie wejścia, konwersja i zwrócenie wyniku konwersji

```
adc_result_t ADC_GetConversion(adc_channel_t channel)
{
    // select the A/D channel
    ADCON0bits.CHS = channel;
    // Turn on the ADC module
    ADCON0bits.ADON = 1;
    // Acquisition time delay
    __delay_us(ACQ_US_DELAY);
    // Start the conversion
    ADCON0bits.GO_nDONE = 1;
    // Wait for the conversion to finish
    while (ADCON0bits.GO_nDONE)
    {
        ...
    }
    // Conversion finished, return the result
    return ((ADRESH << 8) + ADRESL);
}
```

Listing 8. Pomiar napięcia z potencjometru i przeliczenia na wolty

```
//deklaracje zmiennych
float volt;
uint32_t res=0;
char buf_res[6];
//fragment programu
res=ADC_GetConversion(channel_AN4);//konwersja i odczytanie wartości
volt=(float)res*3.22;//przeliczenie na wartość w woltach dla Vref=3,3V.
volt=volt/1000;
sprintf(buf_res,"V=%1.3f",volt);//konwersja na znaki ASCII
```

Listing 9. Fragment programu do mierzenia napięcia z DAC

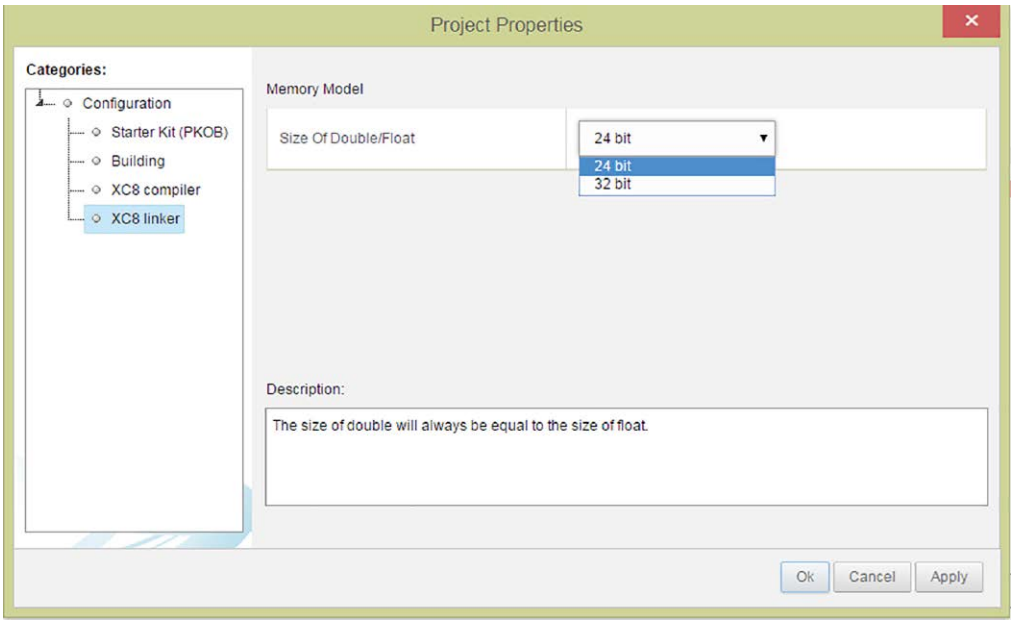
```
DAC1_SetOutput(77);
res=ADC_GetConversion(channel_DAC);
volt=(float)res*3.22;
volt=volt/1000;
sprintf(buf_res,"V=%1.3f",volt);
```

Testy praktyczne

Wygenerowane przez MCC funkcje konfiguracji i wyzwalania konwersji możemy użyć do praktycznych testów działania przetwornika. Ze schematu ideowego modułu Curiosity wynika, że suwak potencjometru modułu można poprzez zworkę J3 połączyć z linią portu RC0 będącą jednocześnie wejściem analogowym AN4. Napięcie na suwaku będzie się zmieniało od 0 V do +3,3 V (Vdd), więc ustawimy w konfiguracji przetwornika Positive reference VDD (rys. 7). Zakres pomiarowy napięcia wejściowego jest równy 3,3 V, a rozdzielczość 10 bitów. Zmiana wartości na ostatnim bicie odpowiada zmianie napięcia równej 3,3 V/1024=3,22 mV. Przeliczanie wartości z wyjścia przetwornika na napięcie wyrażone w woltach będzie polegało na pomnożeniu przez 3,22 i podzieleniu przez 1000. Dokładność wyniku zależy też od długości słowa zmiennoprzecinkowego. W ustawieniach linkera XC8 można wybrać długość 24- lub 32-bitową (**rysunek 10**). Deklaracje *float*, *double* i *long double* mają taką samą liczbę bitów. Wersja bezpłatna kompilatora obsługuje tylko słowa mieszczące 24 bity,

więc używając jej nie mamy większego wyboru.

W programie testowym, za pomocą funkcji ADC_GetConversion(4) przeprowadzimy konwersję napięcia na wejściu analogowym AN4. Wynik konwersji zostanie przeliczony według zależności podanej powyżej. Dodatkowo, za pomocą funkcji *sprintf* wartość napięcia umieszczona zmiennej typu *float* zostanie przekonwertowana na znaki ASCII, aby wynik był gotowy do pokazania na wyświetlaczu lub do przesłania do innego urządzenia. Pomiar napięcia za pomocą wbudowanego przetwornika A/C pokazano na **listingu 8**.



Rysunek 10. Długość słowa double/float

Ponieważ zestaw nie ma wyświetlacza, to posłużymy się sprzętowym symulatorem wbudowanym w moduł Curiosity i obsługiwanym przez MPLAB. Niestety, próby użycia debugera z poziomu MPALB Xpress się nie powiodły. Co prawda, debuger się uruchamiał, program się zatrzymywał na pułapce, ale nie można było podejrzeć wartości zmiennych, a to było głównym zadaniem tego testu. Po kilku niepowodzeniach skonfigurowałem przetwornik za pomocą „stacjonarnego" MPLAB X IDE i wszystko zadziałało.

Rysunek 11. Podgląd zmiennych debugera w czasie pomiaru napięcia

Debugowanie programu jest bardzo łatwe:

- Uruchamiamy debuger za pomocą ikony z paska narzędzi debugera.
- Ustawiamy punkt zatrzymania/pułapkę na pierwszej instrukcji po *sprintf*.
- Uruchamiamy wykonywanie programu za pomocą ikony .
- Kiedy program się zatrzyma na pułapce otwieramy okno *Variables*, w którym można zobaczyć wartość zmiennych programu (**rysunek 11**).

Listing 10. Test przetwornika DAC dla Vref=2,048V

```
DAC1_SetOutput(125);
res=ADC_GetConversion(channel_DAC);
volt=(float)res*2;
volt=volt/1000;
sprintf(buf_res,"V=%1.3f",volt);
```

Listing 11. Zapis liczby 10 bitowej do przetwornika DAC1

```
{
    //DAC input reference range should be 10bit.
    //Input data right justified.
    DAC1CON0bits.DAC1FM = 0;
    //Loading 10bit data to DAC1
    DAC1REFL  = (uint8_t) input10BitData;
    DAC1REFH  = (uint8_t)(input10BitData >> 8);
    //Loading DAC1 double buffer data to latch.
    DACLDbits.DAC1LD = 0x01;
}
```

Poprawność pomiaru najłatwiej sprawdzić mierząc multimetrem napięcie na suwaku potencjometru i porównując je z wynikami wyliczonym przez program. W moim wypadku pomiar zgadzał się z dokładnością do 1 mV.

TOMASZ JABŁOŃSKI, EP