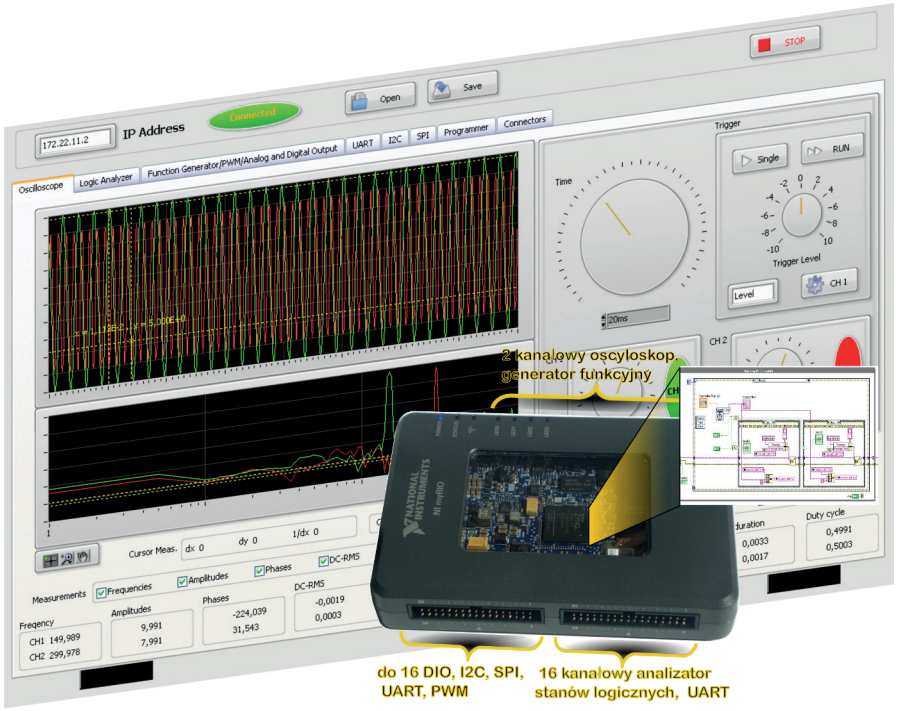
Założenia projektowe:

1. Oscyloskop:
   - Liczba kanałów: 2.
   - Częstotliwość próbkowania: 250 kHz/kanał.
   - Podstawa czasu 100 µs/dz…200 s/dz.
   - Czułość: 2 V/dz.
   - Możliwość sterowania modułem wzmacniaczy wejściowych.
   - Pomiar wielu parametrów sygnału.
2. Generator funkcyjny:
   - Liczba kanałów: 2.
   - Przebiegi: sinus, prostokąt.
   - Zakres częstotliwości: 0,01 Hz…100 kHz.
3. Analizator stanów logicznych:
   - Liczba kanałów: 16.
   - Częstotliwość próbkowania: do 40 MHz.
   - Wyzwalanie: za pomocą dowolnego wzorca.
4. Generator PWM:
   - Liczba kanałów: 1.
   - Zakres częstotliwości: 40 Hz…40 kHz.
5. Interfejsy komunikacyjne:
   - 2×UART.
   - SPI.
   - I²C.
   - Komunikacja z przyrządem za pomocą USB lub Wi-Fi.



myRIO – platforma edukacyjna od National Instruments (3)

Uniwersalny przyrząd laboratoryjny

Firma National Instruments zaproponowała kolejny system przeznaczony dla studentów. Tym razem jest on oparty na technologii wykorzystanej w popularnej platformie CompactRIO. Pozwala na przećwiczenie zagadnień związanych z automatyką, robotyką, systemami kontrolno-pomiarowymi, a szczególnie z wbudowanymi systemami czasu rzeczywistego.

W artykule skupię się na przygotowaniu przedstawionego w poprzedniej części programu. Praca z systemami wbudowanymi różni się od przygotowywania aplikacji do akwizycji danych za pomocą kart pomiarowych czy sterowania urządzeniami kontrolno-pomiarowymi poprzez różne interfejsy komunikacyjne. W projekcie należy zadbać o aplikację pracującą po stronie komputera, program działający w myRIO i konfigurację struktury FPGA. Trzeba również wybrać sposób wymiany danych pomiędzy układem i komputerem, właściwe sterowanie i analizę danych. Na szczęście, wszystkie te platformy można programować za pomocą jednego środowiska i zarządzać w jednym oknie projektu. Wszystko może być obsłużone z poziomu LabVIEW przy wykorzystaniu graficznego języka programowania i bibliotek, które znacznie skracają czas przygotowania projektu.

Projekt startowy

Przygotowanie aplikacji rozpoczynamy od generowania nowego projektu. W tym celu wybieramy w LabVIEW *Create Project*. W nowo otwartym oknie wybieramy (jak na **rysunku 1**) *Templates* → *myRIO* i tu pojawia się do wyboru kilka możliwości.

Wybierając *Blank Project*, musimy o wszystko zatroszczyć się sami. Wybierając *myRIO Project*, dostajemy zestaw standardowych funkcji zawartych w FPGA. Możemy korzystać z palety funkcji myRIO (**rysunek 2**), zapewniającej dostęp do interfejsów komunikacyjnych, generatorów PWM, akcelerometru, wszystkich wejść/wyjść cyfrowych oraz przetworników A/C i C/A, ale nie mamy możliwości ingerencji w FPGA.

W naszym wypadku najlepszym rozwiązaniem jest wybór *myRIO Custom FPGA Project*. Wówczas zostanie wygenerowany projekt z dostępem do wszystkich funkcji zawartych w palecie myRIO zawierający plik *FPGA Main Default*. Zapamiętany w nim kod pozwala na skonfigurowanie struktury FPGA, zapewniając dostęp do palety myRIO. W tym wypadku jest to najlepszy wybór, ponieważ zamierzam zachować część standardowo dostępnych funkcji. W projekcie usuniemy niektóre fragmenty kodu i zmienne wygenerowane podczas jego tworzenia, dlatego korzystając z palety myRIO, należy używać funkcji niskiego poziomu, przekazując do nich jako parametry wejściowe tylko pozostawione w projekcie zmienne. **Rysunek 3** przedstawia drzewo projektu. Możemy na nim wyróżnić wyraźnie pogrupowane elementy związane

z każdą z dostępnych platform. Nasz projekt będzie składał się z aplikacji uruchomionej na komputerze PC, zapewniającej interfejs użytkownika i analizę danych. Po stronie myRIO będzie uruchomiony program, zapewniający komunikację z aplikacją na PC i sterujący pracą układu. Trzecią platformą wyróżnioną w projekcie jest struktura FPGA. To od niej właśnie rozpoczniemy pracę. Na początku w oknie projektu musimy dodać dwa kanały DMA, pozwalające na bezpośredni zapis danych z przetworników pomiarowych i analizatora stanów logicznych do pomięci w trybie FIFO. Można to zrobić, klikając prawym klawiszem myszy na *FPGA Target* i wybierając *New → FIFO*. W otwartym oknie należy ustawić odpowiednie parametry. W zakładce *General* w polu *Name* wpisujemy *L.Analyzer DMA* (nazwa kanału DMA), w polu *Type* wybieramy *Target to Host – DMA* (określa kierunek przepływu danych z układu FPGA do systemu), w polu *Requested Number of Elements* wpisujemy *1023* (określa rozmiar bufora). Przechodzimy do zakładki *Data Type* i w polu *Data Type* wybieramy typ danych U16. Resztę parametrów pozostawiamy bez zmian. Tak samo przygotowujemy kanał DMA dla oscyloskopu, ja nazwałem go *SCOPE.DMA*, w tym wypadku typ danych to *I16*.

Przygotowanie struktury FPGA

Rozpoczynamy od modyfikacji funkcji zawartych w pliku *FPGA Main Default*. Są one dobrze opisane i łatwo się zorientować, który fragment za co odpowiada. Po otworzeniu diagramu widzimy kod składający się z wielu pętli. Każda z pętli odpowiada za konfigurację fragmentu struktury logicznej. Pętle nie muszą być ze sobą w żaden sposób powiązane. Zawartość każdej z nich można traktować jak niezależny „układ scalony" realizujący odpowiednią funkcję logiczną. Dostęp do struktury realizowany jest najczęściej za pomocą rejestrów, poprzez zapis czy odczyt odpowiednich wartości.

Wejścia analogowe

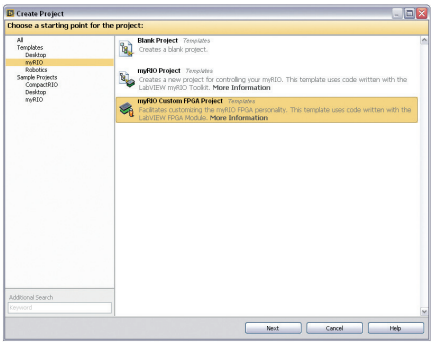
Jako pierwszą zmodyfikujemy pętlę *Analog Input* widoczną na **rysunku 4**. Jak pamiętamy z pierwszej części, dysponujemy tylko jednym przetwornikiem pomiarowym. Jego czas musi zostać podzielony pomiędzy wszystkie skanowane kanały. Pozostawienie tylko dwóch kanałów pozwala na osiągnięcie częstotliwości próbkowania 250 kS/s na każdy kanał. Dodanie kolejnych spowodowałoby znaczne jej zmniejszenie. Zaciski z lewej strony realizują funkcję multipleksera, przełączając kolejno wszystkie mierzone kanały. Ich liczbę należy ograniczyć do dwóch, pozostawiając tylko te związane z wejściami *ConnectorC/AI0* i *ConnectorC/AI1*. To one będą pracowały jako wejścia oscyloskopu. Poza tym, z pętli usuwamy niemal wszystkie rejestry znajdujące się po lewej stronie, pozostawiając tylko *AI.C_0.VAL* i *AI.C_1.VAL*. Teraz diagram musimy uzupełnić o elementy realizujące funkcje związane z oscyloskopem.

Rysunek 5 przedstawia zmodyfikowaną pętlę. Zawarty w niej kod wykonuje się sekwencyjnie, w pierwszej kolejności odmierzany jest czas pomiędzy próbkami, następnie pobierane są dane z przetwornika pomiarowego i sprawdzany jest warunek wyzwolenia oscyloskopu. Wynik konwersji przetwornika jest porównywany z poprzednią próbką dla określenia zbocza sygnału. Sterowanie pracą oscyloskopu odbywa się za pomocą maszyny stanu zrealizowanej w oparciu o strukturę *Case*. Wyróżnionych zostało 5 stanów:

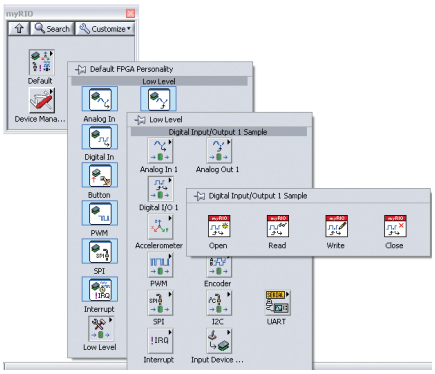
1. **Idle** – stan domyślny, oscyloskop nie pracuje.
2. **Wait for Trigger** – oczekuje na spełnienie warunku wyzwolenia akwizycji próbek.
3. **Start Acquire** – prowadzona jest akwizycja, dane z przetwornika za pomocą kanału DMA zapisywane są do bufora FIFO.
4. **Stop Acquire** – zakończenie akwizycji po zebraniu odpowiedniej liczby próbek. Ustawione zostają odpowiednie stany „diod" sygnalizujących rozpoczęcie i zakończenie akwizycji, przechodzimy do stanu Idle.
5. **Break Acquire** – przerwanie akwizycji i pozostawienie informacji o liczbie zebranych próbek.
6. **Clear Acquie Counter** – zerowanie licznika danych.

Rysunek 6 przedstawia fragment panelu czołowego z widocznymi rejestrami sterującymi oscyloskopem.

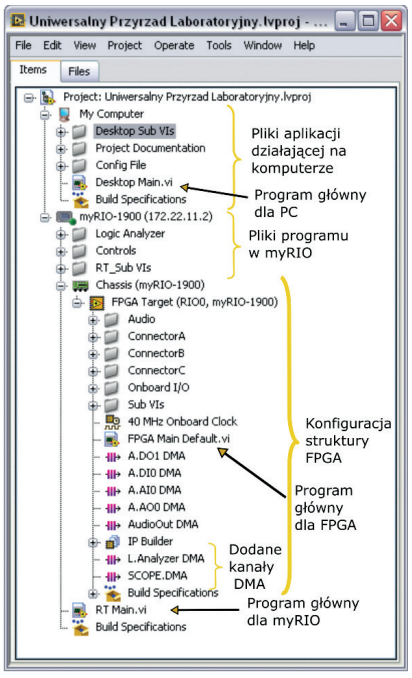
- SCOPE.S.Frequency – wartość licznika odmierzającego czas pomiędzy próbkami, liczymy ją według wzoru *SCOPE.S.Frequency =40 M/fpr* (40 MHz zegar taktujący FPGA, fpr – pożądana częstotliwość próbkowania).
- SCOPE.TO.AQUIRE.DATA – określa liczbę danych podczas jednego cyklu wyzwolenia.
- SCOPE.TRIGGER.VAL – poziom wyzwolenia, w kodzie przetwornika.
- SCOPE.TRIGGER.SOURCE – źródło wyzwolenia.
- SCOPE. A.TRIGGER.TYPE – rodzaj wyzwolenia.
- SCOPE.STATE – pozwala wymusić przejście maszyny stanów do określonego stanu.
- SCOPE.AQUIRE.DATA – liczba aktualnie zgromadzonych próbek.
- SCOPE.END.AQUIRE – informuje o zakończeniu akwizycji.
- SCOPE.TRIGGER – informuje o rozpoczęciu akwizycji.
- AI.C_0.VAL – aktualny wynik konwersji przetwornika dla kanału 0, jest



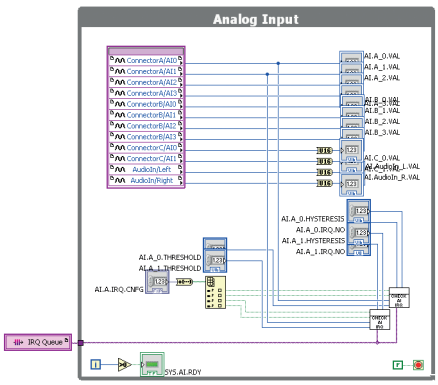
Rysunek 1. Okno generowania projektu



Rysunek 2. Paleta funkcji MyRIO



Rysunek 3. Drzewo projektu



Rysunek 4. Diagram realizujący funkcję wejść analogowych

aktualizowana na bieżąco niezależnie od tego, czy oscyloskop pracuje.
- AI.C_1.VAL – aktualny wynik konwersji przetwornika dla kanału 1.
- SCOPE.CURRENT.STATE – aktualny stan maszyny stanów.

Wyjścia analogowe

Zajmiemy się teraz obsługą przetworników C/A. Z pierwszej części wiemy, że każde wyjście analogowe jest wyposażone we własny przetwornik. Dlatego bez problemu możemy korzystać z wszystkich jednocześnie.

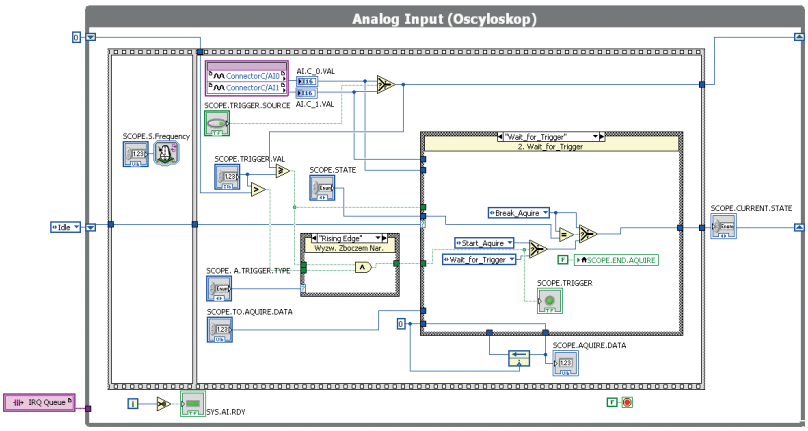
Niewielkiej modyfikacji musimy poddać pętlę *Analog Output* pokazaną na **rysunku 7**. Po lewej stronie widzimy zaciski przekazujące wartości do przetworników. Ponieważ dwa kanały będą pracować jako wyjścia generatorów sygnałowych, musimy usunąć zaciski *ConnectorC/AO0* i *ConnectorC/AO1* związane z wyjściami analogowymi w porcie C. Należy również usunąć rejestry *AO.C_0.VAL* i *AO.C_1.VAL*. Pozostała część diagramu pozostaje bez zmian, dzięki temu funkcje z palety myRIO pozwolą na dostęp do pozostawionych przetworników.

Generator funkcyjny

Na **rysunku 8** zamieszczono diagram odpowiedzialny za generowanie sygnałów analogowych. Do realizacji generatora wykorzystałem funkcje zawarte w palecie *FPGA Math & Analysis → Generation*. Są to generatory DDS. Każdy z kanałów zawiera oddzielną pętlę a więc może pracować niezależnie z różnymi częstotliwościami. Przewidziałem tylko dwa przebiegi, ale można to łatwo rozbudować. Generowane próbki przebiegu są sprzętowo mnożone przez liczbę z przedziału –1…1, pozwalając na jego skalowanie.

Składowa stała została zrealizowana poprzez dodanie do każdej z próbek liczby z przedziału –2048…2048. Te wartości wynikają z faktu, że próbki trafiają bezpośrednio do przetwornika. Wyłączenie generatora powoduje ustawienie napięcia wyjściowego przetwornika równej zero.

Na **rysunku 9** przedstawiono rejestry sterujące generatorami DDS. Każdy z generatorów dysponuje dokładnie takim samym zestawem rejestrów:
- F.GEN.CH0.On – włącza/wyłącza generator.
- F.GEN.CH0.Frequency (periods/tick) – określa częstotliwość sygnału.
- F.GEN.CH0.Phase offset (periods) – przesunięcie fazowe 0…360 stopni.
- F.GEN.CH0.Waveform – rodzaj przebiegu: 0 – sinusoidalny, 1 – prostokątny.
- F.GEN.CH0.Scale – liczba z przedziału –1…1 jest mnożona przez każdą próbkę generowanego sygnału.
- F.GEN.CH0.Offset – wartość składowej stałej (–2048…2048).
- F.GEN.CH0.Reset – zeruje generator.

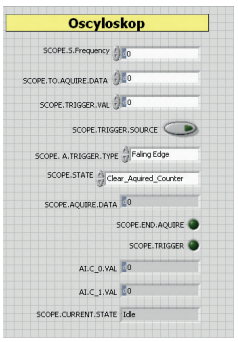


Rysunek 5. Zmodyfikowany diagram z rys. 4 realizujący podstawowe funkcje oscyloskopu

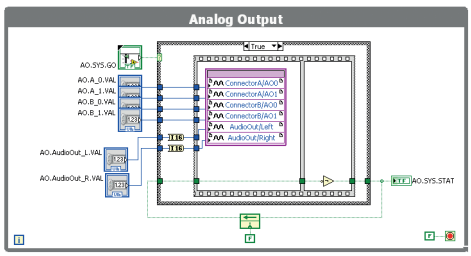
Cyfrowe funkcje portu A

Postanowiłem niemal w całości pozostawić funkcje cyfrowe zrealizowane w porcie A. Ograniczone zasoby FPGA zmusiły mnie do zrezygnowania z dwóch kanałów generatora PWM. **Rysunek 10** przedstawia diagram odpowiedzialny za ten port. Należy z niego usunąć funkcje *DIO_PWM* zaznaczone kolorem czerwonym, odpowiedzialne za kanały 1 i 2 generatora PWM oraz wszystkie rejestry do nich podłączone. W to miejsce wstawiamy funkcje *DIO 1-bit*, dołączając niezbędne sygnały.

Na **rysunku 11** pokazano zmodyfikowany diagram z zaznaczonymi dodatkowymi funkcjami obsługującymi bity 9 i 10 portu A. Taka modyfikacja pozwala na dostęp do wszystkich linii portu poprzez funkcje palety myRIO – wyjątkiem są usunięte kanały PWM. Linie portu wykorzystywane jako wyjścia usuniętych generatorów mogą teraz pracować jako zwykłe wejścia/wyjścia cyfrowe.



Rysunek 6. Rejestry sterujące oscyloskopem

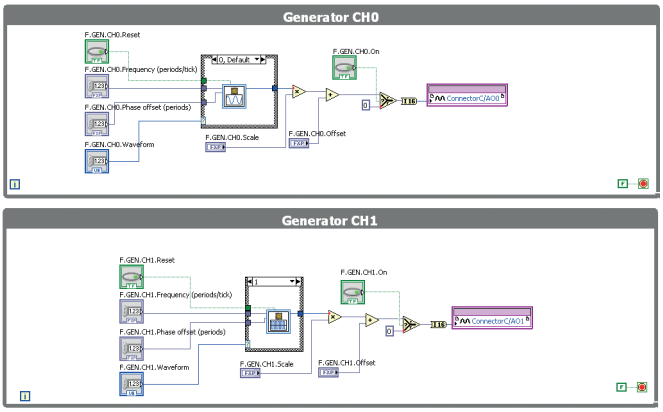


Rysunek 7. Diagram realizujący funkcję wyjść analogowych po usunięciu zacisków związanych z przetwornikami dołączonymi do portu C

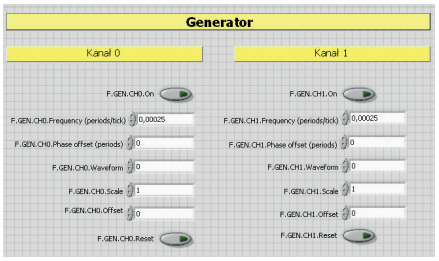
Analizator stanów logicznych

Na potrzeby analizatora stanów logicznych przewidziałem cały port B. Aby zrealizować funkcje analizatora, niezbędne jest usunięcie całego diagramu przedstawionego na **rysunku 12**. Na **rysunku 13** pokazano diagram realizujący podstawowe funkcje analizatora. Kod zawarty w pętli wykonuje się sekwencyjnie, pierwszą funkcją jest licznik odmierzający zadany odcinek czasu wynikający z częstotliwości próbkowania, następnie odczytywana jest zawartość wszystkich pinów portu. Później jest wywoływana struktura *case*. Na jej bazie została zbudowana maszyna stanów. To ona steruje kolejnymi etapami pracy analizatora. Wyróżnione zostało pięć stanów:

1. **Idle** – stan domyślny, analizator nie pracuje.
2. **Wait for Trigger** – oczekuje na spełnienie warunku wyzwolenia, w tym stanie porównywane są wartości odczytane z portu z zadanym wzorcem,



Rysunek 8. Diagram generatorów DDS



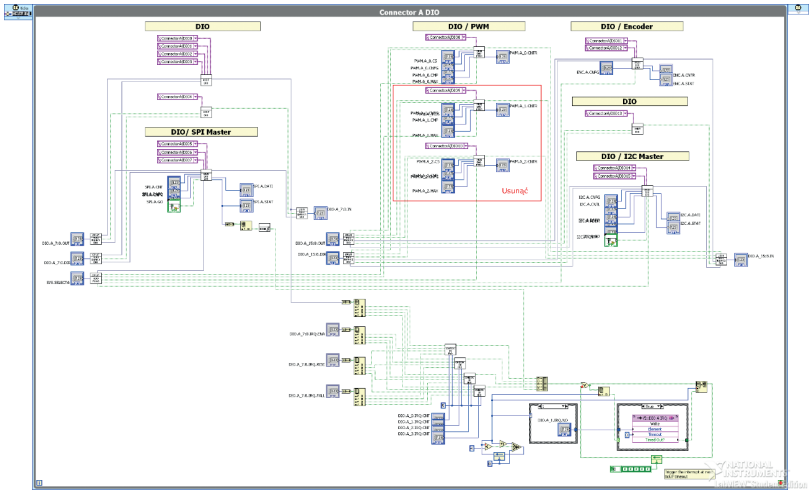
Rysunek 9. Rejestry sterujące generatorami sygnałowymi

po wcześniejszym zamaskowaniu nieistotnych bitów.

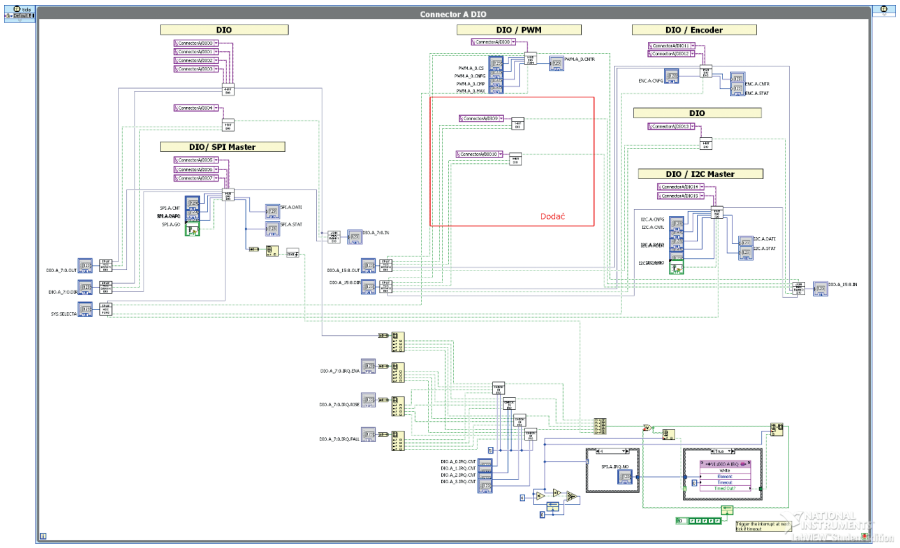
3. **Start Acquire** – prowadzona jest akwizycja, odczytane z portu wartości za pomocą kanału DMA zapisywane są do bufora FIFO.
4. **Stop Acquire** – zakończenie akwizycji po zebraniu odpowiedniej liczby próbek, ustawione zostają odpowiednie stany „diod" sygnalizujących rozpoczęcie i zakończenie akwizycji, przechodzimy do stanu Idle.
5. **Break Acquire** – przerwanie akwizycji i pozostawienie informacji o liczbie zebranych próbek.
6. **Clear Acquire Counter** – zerowanie licznika danych.

Na **rysunku 14** zaprezentowano rejestry związane z analizatorem stanów logicznych. Funkcje poszczególnych rejestrów:

- LOG. A.STATE – steruje maszyną stanów.
- LOG.A.DATA.TO.AQUIRE – liczba próbek do zebrania.
- LOG.A.Freqency(Ticks) – odstępy czasowy pomiędzy kolejnymi próbkami.
- LOG.A.PATTERN – wzorzec bitowy.
- LOG.A.MASK – maska bitowa.
- LOG.A .TRIGGER – sygnalizuje wyzwolenie pomiaru.
- LOG.A .END.AQUIRE – sygnalizuje zakończenie pomiaru.
- LOG. A.CURRENT.STATE – aktualny stan maszyny stanów.



Rysunek 10. Diagram realizujący cyfrowe funkcje portu A

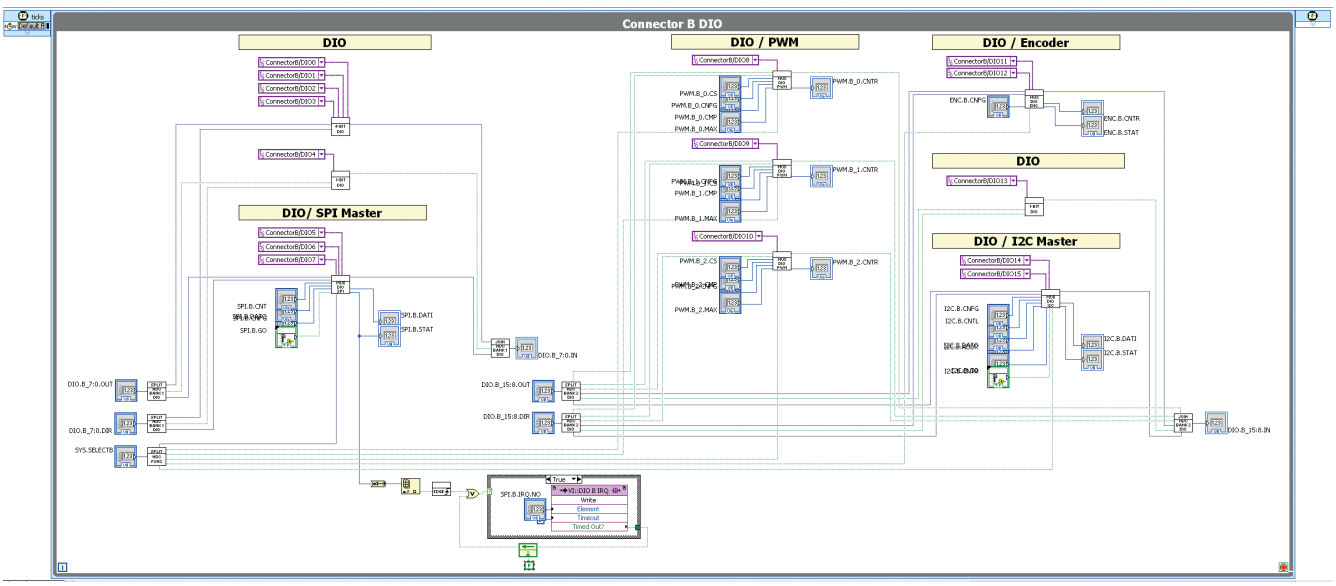


Rysunek 11. Zmodyfikowany diagram realizujący cyfrowe funkcje portu A

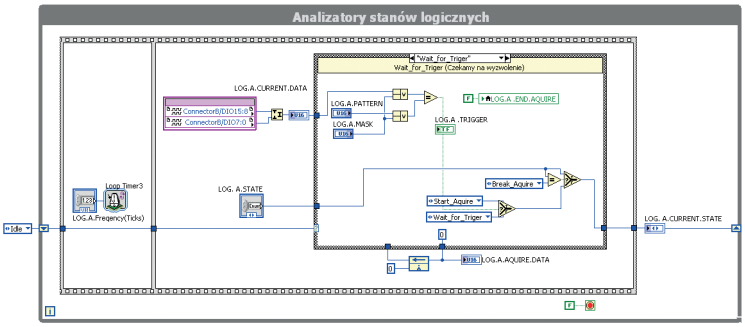
- LOG.A.AQUIRE.DATA – określa, ile danych zostało do tej pory zgromadzonych w pamięci FIFO.
- LOG.A.CURRENT.DATA – aktualny stan bitów w porcje, jest aktualizowany niezależnie od stanu pracy analizatora.

Cyfrowe funkcje portu C

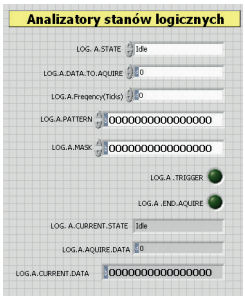
Część analogowa portu jest najbardziej rozbudowana, dlatego w naszym układzie pełni rolę związaną głównie z przetwornikami A/C i CA. **Rysunek 15** prezentuje diagram odpowiedzialny za cyfrowe funkcje portu C,



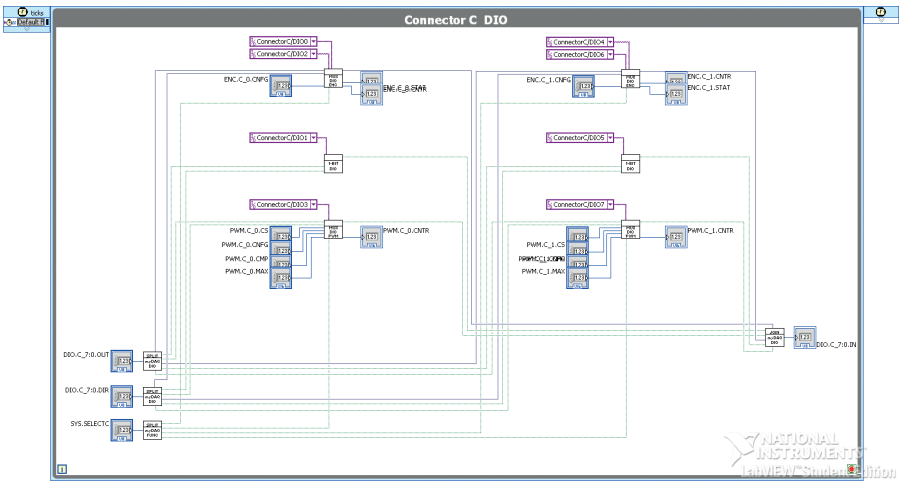
Rysunek 12. Diagram realizujący cyfrowe funkcje portu B



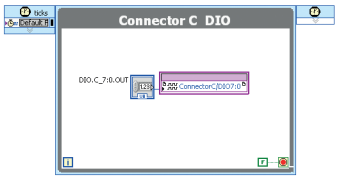
Rysunek 13. Diagram analizatora stanów logicznych



Rysunek 14. Rejestry analizatora stanów logicznych



Rysunek 15. Diagram realizujący cyfrowe funkcje portu C



Rysunek 16. Zmodyfikowany diagram dla portu C

są one znacznie skromniejsze w porównaniu z portami A i B. W naszym urządzeniu funkcje te zostaną zredukowane do ośmiobitowego portu wyjściowego. Jego zadaniem będzie sterowanie wzmocnieniami układów wejściowych oscyloskopu. Dlatego prawie całą zawartość pętli *Connector C DIO* należy usunąć. Pozostawiamy tylko rejestr związany z danymi zapisywanymi do portu, dodając zacisk *ConnectorC/DIO7:0*. Zmodyfikowany diagram jest przedstawiony na **rysunku 16**.

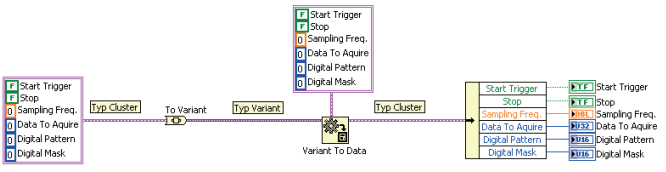
Tak zmodyfikowany diagram należy skompilować, wciskając przycisk *Run*. Do kompilacji można wykorzystać lokalny kompilator lub skorzystać z sieciowego, jeśli mamy do niego dostęp. W zależności od wydajności komputera, kompilacja może potrwać kilkanaście minut. Po jej pomyślnym zakończeniu plik konfiguracyjny przesyłany jest do FPGA i uruchamiany. Od tej pory na panelu czołowym można śledzić zawartości wszystkich rejestrów i przetworników.

Program pracujący po stronie myRIO

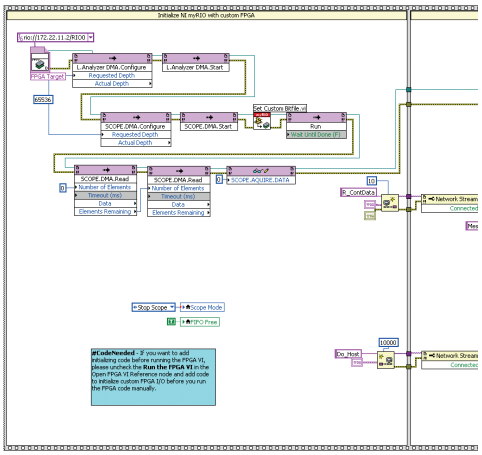
Zadania wymagające największej szybkości i najkrótszej zwłoki czasowej zrealizowane zostały wewnątrz struktury FPGA, program pracujący na platformie myRIO będzie czytał dane z kanałów DMA i przesyłał je do komputera. Drugim jego zadaniem jest odbiór komunikatów od aplikacji pracującej na komputerze, ich interpretacja i wykonanie.

W tym miejscu należy zastanowić się nad sposobem komunikacji pomiędzy myRIO a komputerem. Jest ona kluczowym elementem niezawodnej pracy układu. Powinna być szybka i niezawodna, a jednocześnie nieskomplikowana. Środowisko zapewnia nam kilka technik. Mamy do dyspozycji *TCP/IP*, *UDP*, *Shared Variables*, *Network Streams* i inne. Zastanawiając się nad jej wyborem, pomyślałem o wykorzystaniu zmiennych współdzielonych. Są one bardzo wygodne w użyciu, ale transmisja odbywa się bez potwierdzeń. Najrozsądniejszym wyborem wydaje się zastosowanie funkcji *Network Streams*. Transmisja jest prowadzona pomiędzy dwoma punktami, początkowym (*Writer Endpoint*) i punktem końcowym (*Reader Endpoint*). Protokół zapewnia dostarczenie danych do punktu końcowego, każda prawidłowa transmisja jest potwierdzana. Dane zarówno po stronie nadawczej, jak i odbiorczej gromadzone są w buforach FIFO. Wydajność transmisji jest również duża.

Kolejnym problemem do rozwiązania jest budowa komunikatów, najprościej zbudować komunikat w postaci struktury zawierającej kod komunikatu i dane do niego dołączone. W naszym przypadku kod komunikatu będzie krótką tekstową informacją nazwaną *Message*, natomiast dane będą przesyłane w polu *Message Data*. Teraz powstaje nowy problem – pole *Message Data* powinno przesyłać dane różnych typów. Mogą nimi być zmienne tekstowe tablice z danymi i innego rodzaju zmienne. Na tym etapie nie jesteśmy w stanie określić, jakie dane będziemy potrzebowali przesłać. Środowisko oferuje rozwiązanie tego problemu dając funkcje do konwersji danych na typ *Variant*, pozwalający przesyłać dane niemal dowolnego typu. **Rysunek 17** przedstawia sposób konwersji danych typu *Cluster* na *Variant* i odwrotnie. Konwersją na *Variant* zajmuje się funkcja *To Variant*, konwersja odwrotna jest wykonywana funkcją *Variant To Data*. Jako



Rysunek 17. Przykład konwersji do typu Variant i odwrotnej

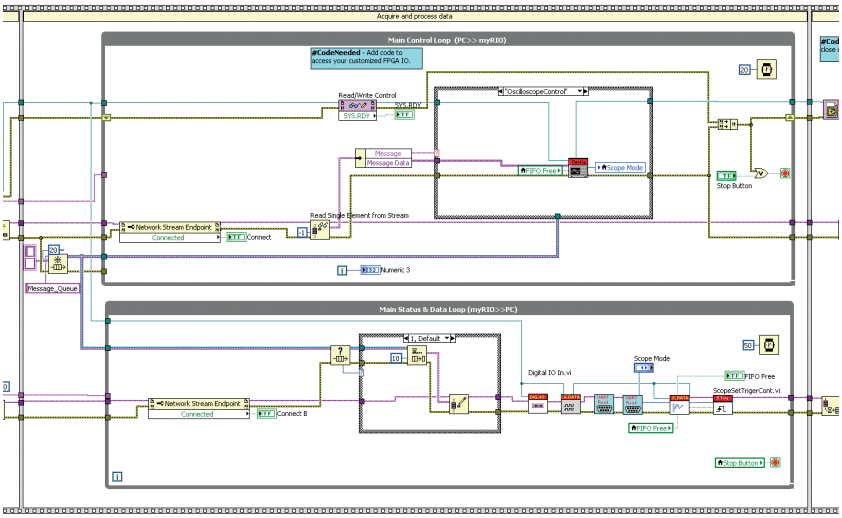


Rysunek 18. Kod inicjujący myRIO

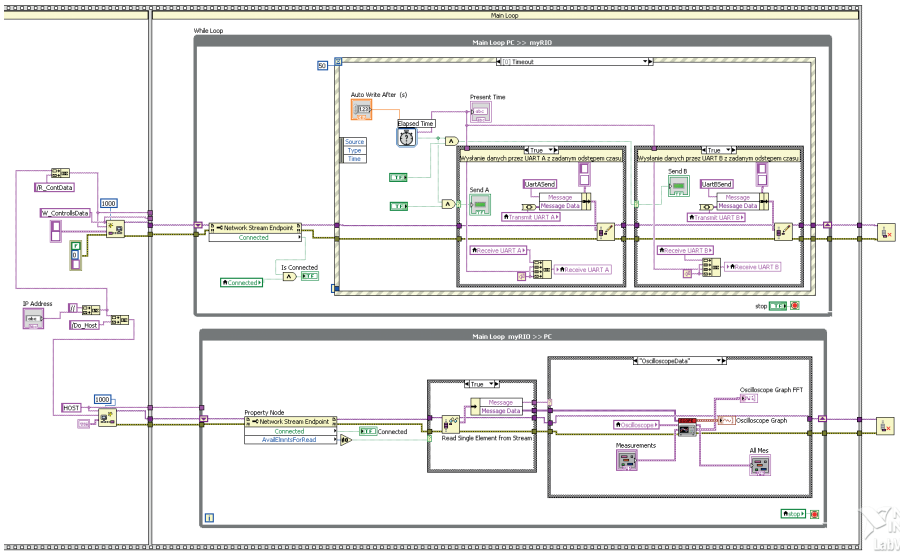
argument, do wejścia *type* musimy dołączyć stałą o dokładnie takiej samej strukturze, jak nasz typ wejściowy. Pozwoli to na poprawną interpretację danych.

Mając rozwiązanie najważniejszych na tym etapie problemów, możemy przystąpić do pisania kodu. Program rozpoczynamy od inicjalizacji myRIO. Na **rysunku 18** pokazano odpowiedzialny za to kod. Inicjalizacja obejmuje:

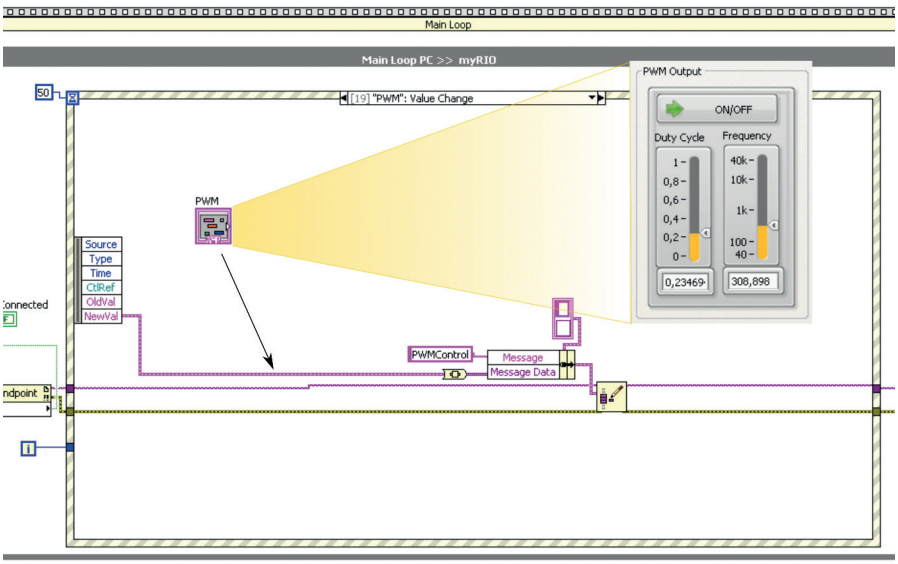
- Konfigurowanie FPGA.
- Konfigurowanie kanałów DMA.
- Inicjalizację zmiennych.
- Nawiązanie połączenia z komputerem.



Rysunek 19. Pętle główne programu w myRIO



Rysunek 20. Diagram aplikacji pracującej na komputerze



Rysunek 21. Ramka struktóry zdarzeń odpowiedzialnej za sterowanie generatorem PWM

Po nawiązaniu połączenia program przechodzi do pętli głównych widocznych na **rysunku 19**. Obsługa transmisji od komputera do myRIO jest realizowana w pętli (*Main Control Loop – PC ➞ myRIO*). Jedną z pierwszych funkcji, która zostaje wywołana, jest *Read Single Element from Stream*. Oczekuje ona na dane z aplikacji uruchomionej na komputerze. W takiej konfiguracji jak obecnie do końcówki *timeout* przyłączono wartość *–1*. Oznacza to, że czas oczekiwania na dane jest nieskończenie długi, a więc wszystkie funkcje znajdujące się dalej zostaną wykonane tylko w wypadku nadejścia komunikatu z komputera. W przeciwnym razie pętla zatrzyma się, czekając na nadejście danych z komputera.

Odebrany komunikat jest rozdzielany na dwie części: *Message* i *Data*. Zawartość pola *Message* jest przekazywana bezpośrednio do struktury *Case* kierującej program do właściwej funkcji. Dane z pola *Data* różnią się dla każdego komunikatu, nie tylko wartościami, ale również typem. Dlatego każda funkcja interpretuje i rozgrupowuje je oddzielnie, i nie ma możliwości zrobienia tego przed ich wywołaniem.

Obsługa nowego komunikatu ogranicza się tylko do dodania kolejnej zakładki w strukturze *Case* i umieszczenia wewnątrz funkcji odpowiedzialnej za realizację zadań z nim związanych. Oczywiście, w niektórych wypadkach może okazać się, że zadania związane z nowym komunikatem będą wykonywane w drugiej pętli. Wówczas do wymiany danych pomiędzy pętlami można wykorzystać zmienne lokalne lub inną technikę synchronizacji zadań wykonywanych równolegle. W programie zaszła konieczność przesyłania danych odczytywanych za pomocą interfejsów I²C i SPI z pętli „górnej" do „dolnej". Wykorzystałem do tego celu funkcje z palety *Synchronization → Queue Operations*. Są to bufory FIFO, dane w „górnej" pętli zapisywane funkcją *Enqueue Element* są pobierane w „dolnej" pętli przez *Dequeue Element*. Funkcja *Get Queue Status* sprawdza, czy w kolejce znajdują się jakieś elementy. Komunikaty przekazywane poprzez kolejkę mają dokładnie taką samą konstrukcję, jak wysyłane do PC, dlatego przesłanie ich do komputera nie wymaga żadnych dodatkowych operacji. Obecnie dane odczytane z kolejki za pomocą *Dequeue Element* są przesyłane do *Write Single Element to Stream* i transmitowane do komputera w niezmienionej formie. Takie rozwiązanie znacznie upraszcza nasz program i daje możliwość łatwej modyfikacji.

Wysyłając dane z „górnej” pętli, zapisujemy je do kolejki FIFO, a „dolna” pętla prześle je dalej kanałem zwrotnym.

W pętli „dolnej”, oprócz retransmisji komunikatów z kolejki FIFO, transmitowane są dane pomiarowe z oscyloskopu, analizatora stanów logicznych oraz dane odczytane z portów szeregowych.

Aplikacja sterująca dla PC

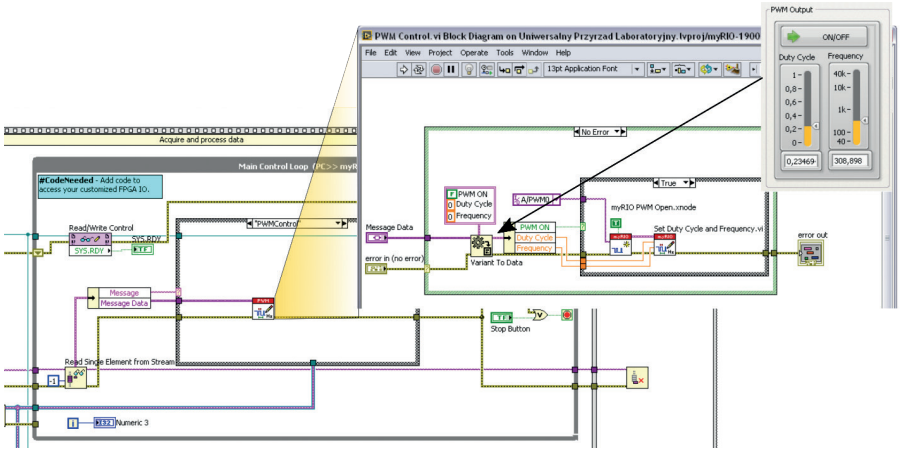
Aplikacja na komputerze PC zapewnia analizę danych pomiarowych i ich wizualizację. Steruje wszystkimi funkcjami przyrządu poprzez komunikaty wysyłane do myRIO. Program główny jest zbudowany z dwóch równolegle wykonujących się pętli *While* pokazanych na **rysunku 20**. Pętla *Main Loop PC ➞ myRIO* obsługuje panel użytkownika i wysyłanie poleceń do *myRIO*. Naciśnięcie lub zmiana wartości w wybranych kontrolkach panelu czołowego powoduje wywołanie odpowiedniej ramki struktury zdarzeń. Część zdarzeń jest związana z obsługą aplikacji, a pozostałe wysyłają komunikaty sterujące myRIO z odpowiednimi parametrami. Pętla *Main Loop myRIO ➞ PC* odbiera komunikaty przysyłane z myRIO i analizuje je, wywołując odpowiednią funkcje.

Przykład wymiany komunikatów

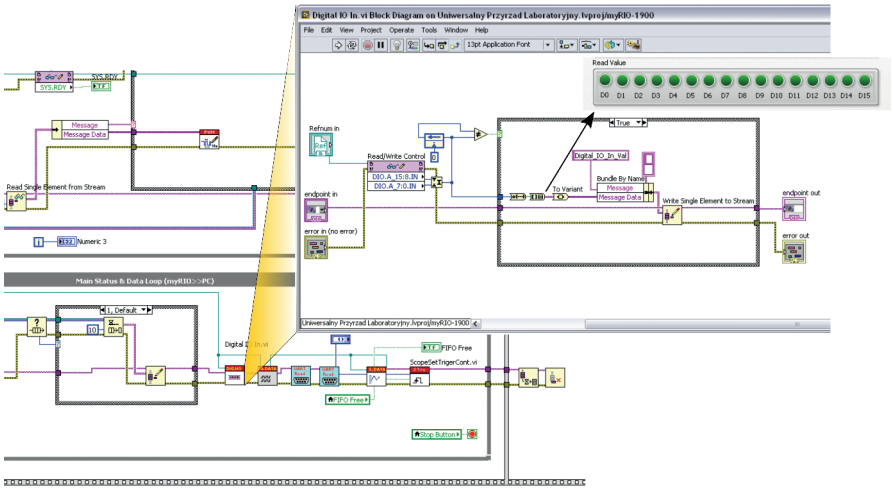
Kluczowym elementem przydatnym przy rozbudowie programu jest znajomość sposobu komunikowania się aplikacji pracującej po stronie komputera z myRIO. Dlatego prześledzimy na dwóch najprostszych przykładach wymianę komunikatów w obu kierunkach.

Jako przykład komunikatu wysyłanego do myRIO posłuży polecenie programujące generator PWM. Na panelu czołowym generator steruje kontrolką *PWM*, widoczną w prawym górnym rogu na **rysunku 21**. Jest to zmienna typu *cluster* (odpowiednik struktury w języku C) składająca się z trzech elementów: przycisku i dwóch suwaków odpowiadających za częstotliwość oraz współczynnik wypełnienia przebiegu. W pętli *Main Loop PC ➞ myRIO* zostało zdefiniowane zdarzenie związane ze zmianą wartości dowolnego elementu tej kontrolki. Naciśnięcie na przycisk lub zmiana położenia suwaka spowoduje wywołanie funkcji zawartej w tej ramce. W efekcie zostanie wysłany komunikat zawierający w polu *Message* tekst *PWMControl*, natomiast w polu *Message Data* zawartość wszystkich pól kontrolki *PWM* (po konwersji na typ *Variant*), a więc położenie przycisku wartości współczynnika wypełnienia i częstotliwości. W ten sposób zmiana jednego z parametrów powoduje uaktualnienie wszystkich trzech.

Przejdźmy teraz do programu po stronie myRIO, czyli do pliku *RT Main*. Tutaj, w pętli *Main Control Loop* (*PC ➞ myRIO*), funkcja *Read Single Element from Stream* zwraca zawartość odebranego komunikatu. Po rozgrupowaniu wartość pola *Message* jest przekazywana bezpośrednio do struktury *Case* wywołującej odpowiednią funkcję. W tym wypadku funkcja została zapisana jako oddzielny pliku (*Subvi*), a jej diagram widzimy w prawym górnym rogu **rysunku 22**. Zawartość pola *Message Data* jest konwertowana do pierwotnej postaci za pomocą funkcji *Wariant To Data* dzięki dołączeniu do zacisku *type* stałej o dokładnie takiej strukturze, jak nasza zmienna po stronie komputera. Strzałki na obu rysunkach wskazują miejsca, w których zmienne są tego samego typu. Następnie dane są rozgrupowywane i przekazywane do funkcji konfigurujących. Skorzystałem ze standardowych funkcji niskiego poziomu dostępnych w palecie myRIO. Na **rysunku 22** zamieszczono fragment kodu odpowiedzialny za uruchomienie i zmianę parametrów generatora PWM. Należy pamiętać, że wartość przekazana do zacisku *Chanel Name* funkcji *myRIO PWM Open* może być równa tylko *A/PWM0*, ponieważ pozostałe układy PWM i wszystkie zmienne z nimi związane zostały usunięte z projektu. Konieczne jest również przekazanie wartości *True* do zacisku *Allow multiple opens?* Pozostawienie wartości domyślnej *False* będzie skutkowało zgłaszaniem błędu przy powtórnym wywołaniu funkcji.



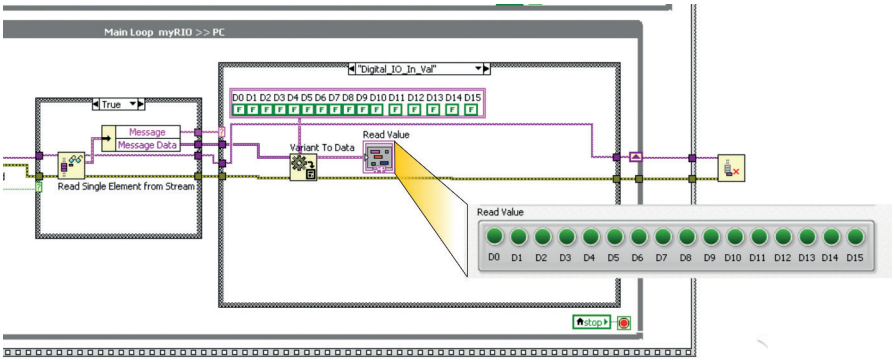
Rysunek 22. Fragment programu odpowiadającego za konfiguracje generatora PWM po stronie myRIO



Rysunek 23. Fragment programu czytającego i wysyłającego wartość bitów portu A, pracującego po stronie myRIO

Odczyt danych z myRIO

Przesyłanie danych z myRIO do komputera odbywa się w pętli *Main Status & Data Loop* (*myRIO ➞ PC*). Pierwsze funkcje odpowiadają za sprawdzenie stanu kolejki FIFO przekazującej dane z „górnej” pętli i ich przesłanie do komputera. Następnie wywoływane są funkcje odpowiedzialne za odczyt stanu bitów w porcie A, przesyłanie danych analizatora stanów logicznych, portów UART i oscyloskopu. Sposób wysyłania komunikatów prześledzimy, analizując przesyłanie wartości bitów portu A. Na **rysunku 23** pokazano pętlę główną wraz z funkcją *Digital IO In.vi* odczytującą port A – jej diagram jest widoczny z prawej strony rysunku. Funkcja *Read/Write Control* odczytuje zawartość portu wprost z rejestrów FPGA. Odczytana wartość jest porównywana z poprzednią dla uniknięcia niepotrzebnych transmisji – dane przesyłane są tylko w wypadku wykrycia ich zmiany. Odczytane wartości grupowane są w zmienną 16-bitową, konwertowane do tablicy bitowej i klastra danych, który



Rysunek 24. Fragment programu czytającego komunikaty od myRIO i wyświetlającego wartość bitów portu A

jest zamieniany na typ *Variant*. Używając tak przygotowanej wartości, tworzy się komunikat o takiej samej strukturze, jak poprzednio. Pole *Message* zawiera tekst *Digital_IO_In_Val*, natomiast *Message Data* wartości bitów portu. Komunikat jest wysyłany poprzez *Write Single Element to Stream*.

Przejdźmy teraz do aplikacji działającej na komputerze, czyli pliku *Desktop Main*. Odczyt komunikatów z myRIO odbywa się w pętli *Main Loop myRIO → PC* pokazanej na **rysunku 24**. Korzystając z *Property Node*, sprawdzamy, czy w kolejce czeka jakiś komunikat i jeśli tak, to *Read Single Element from Stream* odczytuje go, a następnie zostaje on rozgrupowany, natomiast zawartość pola *Message* trafia do struktury *Case* wywołującej odpowiednią funkcję. Zawartość pola *Message Data* jest poddana konwersji z typu *variant* do *cluster*, jak poprzednio, za pomocą *Variant To Data*. Tym razem do końcówki *Type* przyłączamy stałą o strukturze takiej, jak *Read Value*. Wynik konwersji może być przekazany wprost do *Read Value* i wyświetlony na panelu czołowym.

Podsumowanie

Aplikacja jest dość rozbudowana, mająca spore możliwości, wykorzystująca dostępne zasoby w znacznym stopniu. Najciekawszym elementem wydają mi się panele związane z interfejsami I²C oraz SPI pozwalające na modyfikowanie zawartości rejestrów. Przygotowane algorytmy były testowane głównie na pamięciach. Różnorodność dostępnych układów może spowodować konieczność ich rozbudowy lub dodanie nowych. Jak pisałem w poprzedniej części, w kontrolce *SPI Device Info* pole *Address Code* wykorzystałem do rozróżnienia sposobu kodowania adresu komórki pamięci. Myślę, że należy wykorzystać je również do wyboru właściwego algorytmu dla konkretnego układu. Zawartość tego pola jest przechowywana w pliku konfiguracyjnym. Dlatego raz wybrany algorytm będzie zapamiętany i przypisany do układu. Po stronie myRIO program, sprawdzając jego zawartość, powinien wybrać właściwy algorytm.

W artykule skupiłem się głównie na strukturze FPGA i zagadnieniach związanych z komunikacją pomiędzy myRIO a programem sterującym. Pokazałem sposób przepływu danych w obu kierunkach. Zawarte informacje powinny ułatwić dalszą rozbudowę i udoskonalenie programu. Szczegółowy opis wszystkich funkcji byłby zbyt obszerny, osoby pracujące w tym środowisku bez trudu przeanalizują interesujące ich fragmenty kodu i dostosują je do własnych potrzeb i upodobań.

Wiesław Szaj
wszaj@prz.edu.pl