# Eksperymenty z FPGA (8)

**W tej części kursu uruchomimy przetwornik analogowo-cyfrowy wbudowany w układ MAX10. Przygotujemy projekt odczytujący napięcie z dwóch potencjometrów i wyświetlający ich wartość na diodach LED. Nauczymy się także, jak wygenerować bloki IP, pozwalające na skonfigurowanie peryferiów FPGA. Przed przystąpieniem do wykonywania eksperymentów, zachęcam do aktualizacji repozytorium z przykładami (np. poleceniem git pull).**



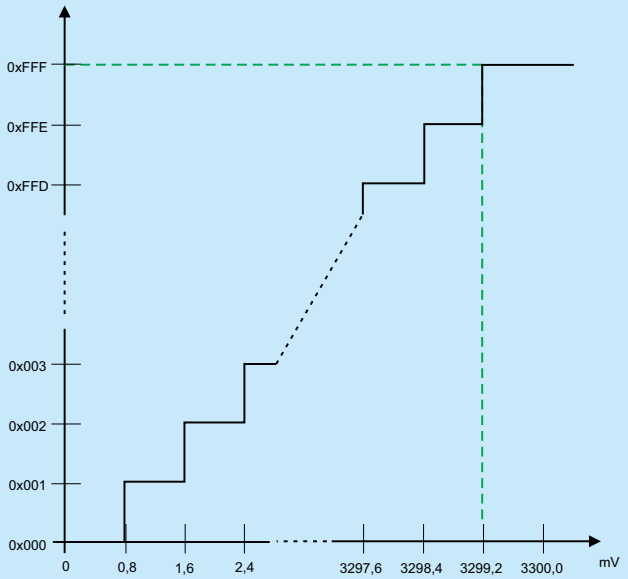
## Przetwornik ADC w MAX10

Informację na temat przetwornika analogowo-cyfrowego znajdziemy w dokumentacji producenta [1]. W układzie 10M04, znajdującym się na płytce Rysino, dostępny jest jeden przetwornik ADC. Może wykonywać pomiar napięcia na jednym z dziesięciu kanałów, do których należą:

- jedno dedykowane wejście analogowe (ANAIN1),
- osiem wejść, które mogą pracować zarówno jako wejście analogowe, jak i wejście-wyjście cyfrowe (ADC1IN1–ADC1IN8),
- sygnał z czujnika temperatury układu.

Na płytce Rysino, podobnie jak w klasycznej płytce Arduino, wyprowadzonych jest sześć z nich. Piny od A0 do A5 są połączone kolejno do wejść ADC1IN1...ADC1IN6. Pamiętajmy, że jeśli wykorzystamy któreś z tych wejść analogowych, to nie można skonfigurować żadnego pinu z banku 1A jako wejścia lub wyjścia cyfrowego. Informację o funkcji poszczególnych pinów znajdziemy w dokumentacji technicznej [2]. Znajduje się tam duża tabela, w której opisano funkcję każdego ze 144 dostępnych pinów. Fragment dotyczący przetwornika ADC widzimy na **rysunku 1**.

Uproszczony schemat przetwornika pokazuje **rysunek 2**. Ma on formę tak zwanego *hard IP block*, czyli „twardego" bloku własności intelektualnej, który jest na stałe „wytrawiony" w strukturze układu. Pierwszym jego elementem jest analogowy multiplekser, gdzie wybieramy wejście analogowe, na którym ma być wykonany pomiar. Dalej znajdziemy moduł S&H (*sample and hold*), czyli układ próbkująco-podtrzymujący. Dzięki niemu mierzone napięcie jest stałe przez cały czas pomiaru. Kolejnym elementem jest przetwornik analogowo-cyfrowy typu SAR (*successive approximation ADC*, czyli przetwornik z sukcesywną aproksymacją).



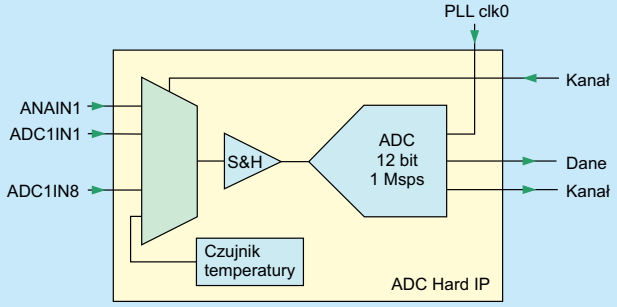

**Rysunek 3. Charakterystyka przetwornika ADC dla napięcia odniesienia 3,3 V**

Można wybrać pomiędzy dwoma wewnętrznymi napięciami odniesienia: 3 V oraz 3,3 V. Pomiar jest wykonywany z rozdzielczością 12 bitów. Napięciu 0 V odpowiada wartość 0x000, a maksymalnemu napięciu, wartość 0xFFF (4095). **Rysunek 3** pokazuje wartości zwracane względem różnych napięć wejściowych, dla wartości napięcia odniesienia równego 3,3 V. Zwróćmy uwagę, że 0xFFF nie oznacza wartości maksymalnej, ale jest zwracana

intel FPGA — Pin Information for the Intel® MAX®10 10M04SA Device
Version 2017.12.15
Note (1)

| Bank Number | VREF | Pin Name/Function | Optional Function(s) | Configuration Function | Dedicated Tx/Rx Channel | Emulated LVDS Output Channel | IO Performance | E144 (2) |
|---|---|---|---|---|---|---|---|---|
| 1A | VREFB1N0 | IO | ADC1IN1 | | DIFFIO_RX_L1n | DIFFOUT_L1n | Low_Speed | 6 |
| 1A | VREFB1N0 | IO | ADC1IN2 | | DIFFIO_RX_L1p | DIFFOUT_L1p | Low_Speed | 7 |
| 1A | VREFB1N0 | IO | ADC1IN3 | | DIFFIO_RX_L3n | DIFFOUT_L3n | Low_Speed | 8 |
| 1A | VREFB1N0 | IO | ADC1IN4 | | DIFFIO_RX_L3p | DIFFOUT_L3p | Low_Speed | 10 |
| 1A | VREFB1N0 | IO | ADC1IN5 | | DIFFIO_RX_L5n | DIFFOUT_L5n | Low_Speed | 11 |
| 1A | VREFB1N0 | IO | ADC1IN6 | | DIFFIO_RX_L5p | DIFFOUT_L5p | Low_Speed | 12 |
| 1A | VREFB1N0 | IO | ADC1IN7 | | DIFFIO_RX_L7n | DIFFOUT_L7n | Low_Speed | 13 |
| 1A | VREFB1N0 | IO | ADC1IN8 | | DIFFIO_RX_L7p | DIFFOUT_L7p | Low_Speed | 14 |
| 1B | VREFB1N0 | IO | | JTAGEN | | | | 15 |
| 1B | VREFB1N0 | IO | | TMS | DIFFIO_RX_L11n | DIFFOUT_L11n | Low_Speed | 16 |

**Rysunek 1. Fragment opisu pinów układu 10M4[2]**




**Rysunek 2. Schemat bloku ADC**

dla wartości mniejszej o jeden krok (dla 3,3 V jest to napięcie 3,292 V). Maksymalna częstotliwość próbkowania to milion próbek na sekundę (z wyjątkiem termometru, dla którego jest ograniczona do 50 tysięcy).

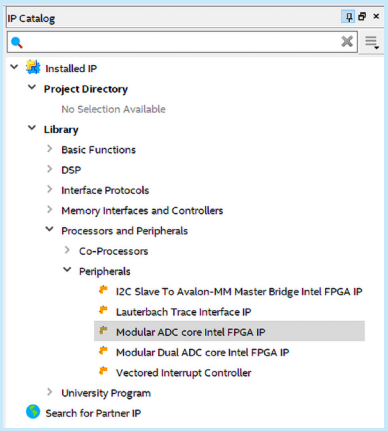
Na rysunku 2 widoczne są dwie magistrale, pozwalające na obsługę przetwornika. Pierwsza z nich, wejściowa, pozwala na wybór kanału. Kiedy zakończy się konwersja, na magistrali wyjściowej pojawi się numer zmierzonego kanału oraz uzyskana wartość. Ostatnim elementem jest zegar, który taktuje przetwornik. Na stałe jest to zerowe wyjście z bloku PLL, na cztery dostępne (*phase lock loop*, czyli pętli synchronizacji fazy). Dzięki temu można zmieniać częstotliwość

próbkowania w szerokim zakresie. Dokładny opis bloku znajduje się w dokumentacji technicznej [3] zegara.

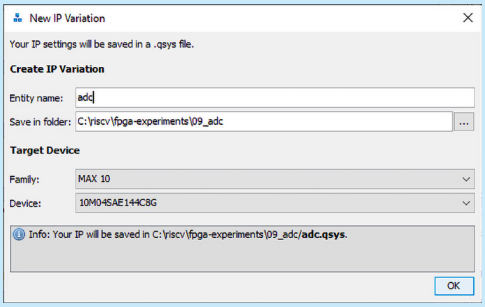
## Generowanie bloków IP

Aby skorzystać z modułu ADC i PLL, musimy wygenerować odpowiednie bloki IP w pakiecie Quartus. Proces ten został pokazany krok po kroku na filmie instruktażowym [4]. Artykuł opisuje jedynie wybrane momenty.

Na początek trzeba stworzyć nowy projekt i wybrać używany przez nas model układu FPGA, tak jak było to wykonywane we wcześniejszych częściach kursu. Kolejnym krokiem jest wygenerowe bloku obsługującego ADC. Po prawej stronie głównego okna środowiska Quartus znajduje się panel IP Catalog (**rysunek 4**). Rozwijamy po kolei *Library* → *Processors and Peripherals* → *Peripherals* i dwa razy klikamy opcję *Modular ADC core Intel FPGA IP*. Pojawi się okno tworzenia nowego IP, pokazane na **rysunku 5**.




**Rysunek 4. Panel IP Catalog w środowisku Quartus**




**Rysunek 5. Okno New IP Variation**

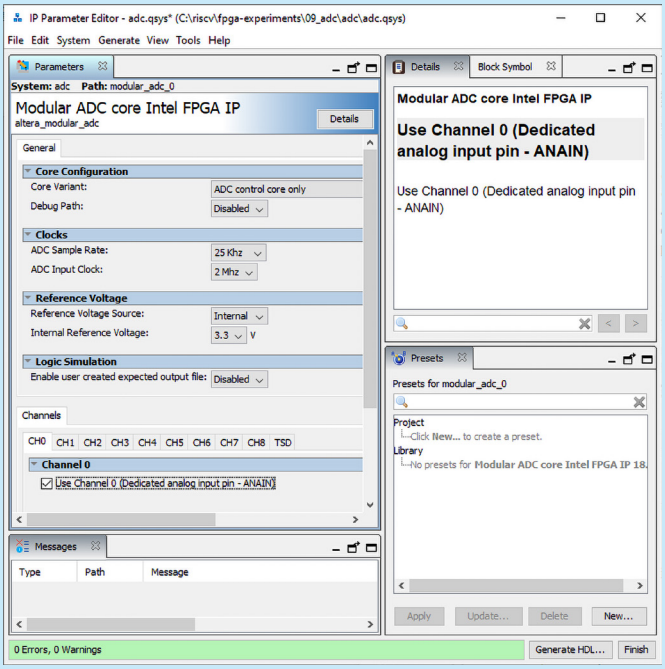
W polu *Entity name* (*nazwa instancji*) wpisujemy nazwę wygenerowanego modułu (wpisałem *adc*). W kolejnym polu *Save in folder* (*zapisz w folderze*) podajemy miejsce w którym będą zapisane pliki. Można zostawić opcję domyślną, czyli główny folder projektu. Aby utrzymać większy porządek w plikach, można utworzyć nowy folder, dopisując go w ścieżce (utworzyłem tu folder *adc*). W dolnej części okna widzimy rodzinę układów scalonych (tutaj MAX10) oraz dokładny model układu. Podane informacje zostały pobrane z konfiguracji projektu. Zatwierdzamy naciskając OK.

Przechodzimy do okna konfiguracji widocznego na **rysunku 6**. Z pierwszego menu rozwijanego wybieramy wariant modułu. Opcja *ADC control core only* oznacza tryb minimalny. Wyborem kanału oraz odczytem wyjścia zajmiemy się manualnie. W zakładce *Clocks* (*zegary*) wybieramy częstotliwość próbkowania oraz częstotliwość zegara – 25 tysięcy próbek na sekundę. Częstotliwość zegara jest ustalona na 2 MHz, a na płytce znajduje się generator o częstotliwości 8 MHz. Nie jest to problem, ponieważ wygenerowaniem dodatkowego przebiegu zegarowego, zajmie się pętla PLL. Przechodzimy teraz do kolejnej części, gdzie wybieramy źródło napięcia odniesienia (*reference voltage*). Wybrałem tu opcję *Internal* (wewnętrzne) i wartość 3,3 V.

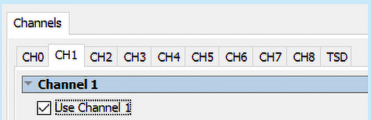
Ostatnim krokiem jest wybór kanałów w panelu *Channels*. Najpierw odznaczamy domyślnie włączony kanał CH0. Następnie z zakładek wybieramy CH1 i zaznaczamy „ptaszkiem" jego wybór, co pokazano na **rysunku 7**. To samo wykonujemy dla kanału CH2, ponieważ użyjemy go w projekcie. Po zakończonej konfiguracji klikamy znajdujący się w prawym dolnym rogu przycisk *Generate HDL*.

Przechodzimy do okna wyboru parametrów i generowania kodu RTL. Okno dzieli się na dwie części: konfigurację kodu HDL (wykorzystywanego do syntezy) oraz implementację. W obydwu częściach wybieramy język *Verilog*. Na samym dole mamy możliwość wybrania ścieżki (*Path*). Pliki będą umieszczone w dodatkowym folderze, o takiej nazwie jak nazwa IP. Klikamy *Generate* (generuj). Pojawi się okienko (**rysunek 9**), w którym zostaniemy poinformowani o wykonywanych czynnościach, a gdy się zakończą, klikamy przycisk *Close*. Ostatnim krokiem jest dodanie do projektu pliku *adc.qip*, z katalogu *synthesis*.

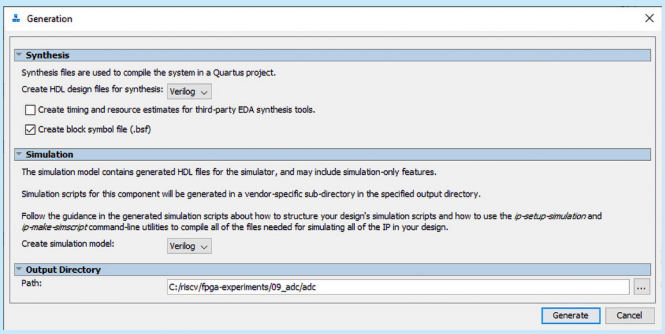
Kiedy znamy już wymaganą częstotliwość zegara, przechodzimy do konfiguracji pętli PLL. Znowu korzystamy z panelu *IP Catalog*. Tym razem, co ilustruje **rysunek 10**, wybieramy *Library* → *Basic Functions* → *Clocks; PLLs and Resets* → *PLL* → *ALTPLL*. Pojawi się okno (**rysunek 11**), w którym możemy wybrać lokalizację plików dla nowego komponentu oraz język. Tworzymy nowy folder *pll* i wybieramy *Verilog*.




**Rysunek 6. Konfiguracja ADC**



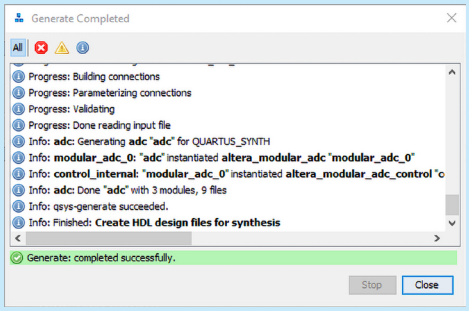

**Rysunek 7. Wybór kanału**



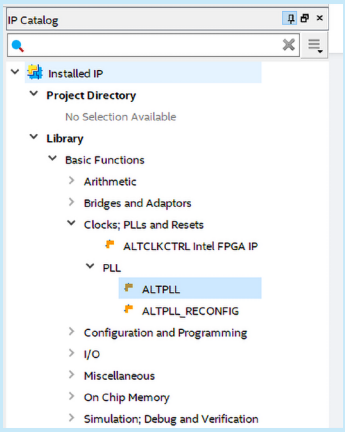

**Rysunek 8. Parametry kodu RTL**

Po kliknięciu *OK* przejdziemy do kreatora, w którym wybierzemy konfigurację. Pierwszy ekran pokazuje **rysunek 12**. Konfigurujemy w nim częstotliwość wejściowego sygnału zegarowego. W naszym przypadku jest to 8,000 MHz.

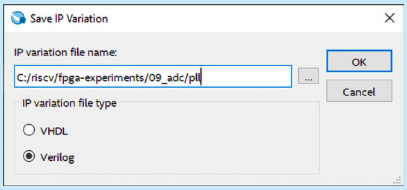
Po kliknięciu Next przejdziemy do okna (**rysunek 13**), w którym wybieramy wejścia i wyjścia tworzonego modułu. Tutaj możemy odznaczyć opcję Create an 'areset' input, ponieważ nie będziemy korzystać z tego wejścia.



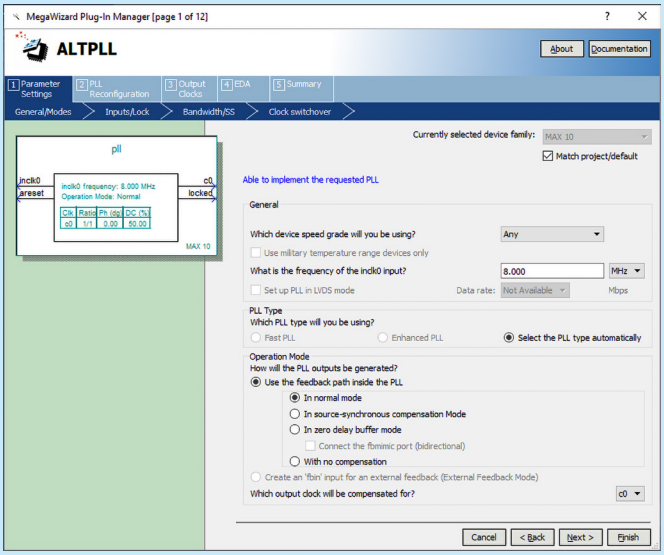

**Rysunek 9. Okno z informacjami o generowaniu IP core**




**Rysunek 10. Konfiguracja bloku PLL**




**Rysunek 11. Okno tworzenia IP dla PLL**




**Rysunek 12. Konfiguracja parametrów wejściowego sygnału zegarowego**

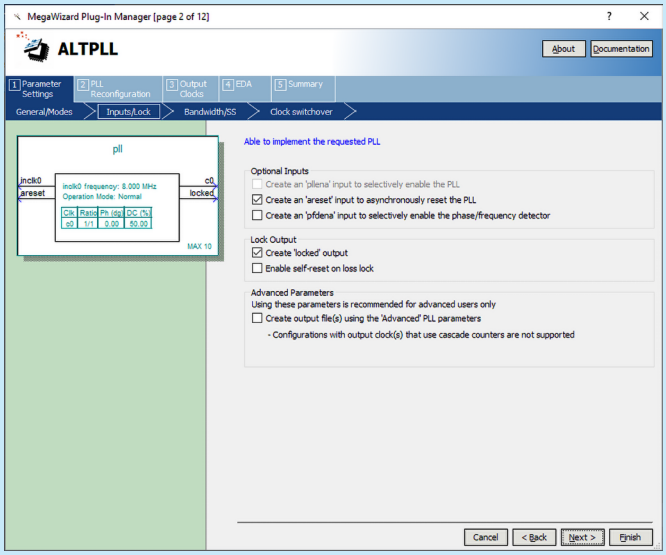
Klikamy kilkukrotnie Next, aż dojdziemy do sekcji Output Clocks (zegary wyjściowe). Pierwszy ekran (**rysunek 14**) odpowiada za wyjście clk c0. Zaznaczamy opcję Enter output clock frequency (podaj częstotliwość wyjściowego sygnału zegarowego). W polu tekstowym, zgodnie z ustawieniami podczas konfiguracji ADC, wpisujemy 2 MHz. Ustawienia przesunięcia fazy (*clock phase shift*) i wypełnienia (*clock duty cycle*) zostawiamy bez zmian. To już ostatni ekran, w którym dokonujemy zmian. Warto zapoznać się z kolejnymi wyjściami za pomocą przycisku Next ale niecierpliwi mogą od razu skorzystać z opcji Finish.

Na koniec zobaczymy okno (**rysunek 15**) z pytaniem – czy chcemy dodać nowo stworzony blok IP do projektu? Wybieramy *Yes*.

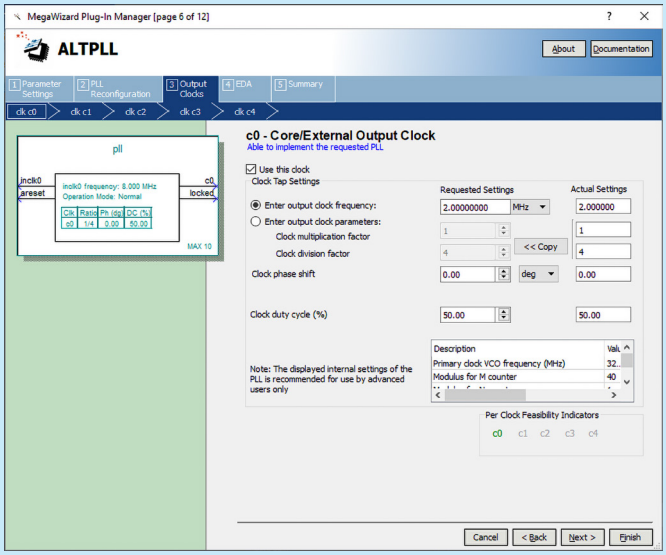
## Łączymy bloki

Wygenerowaliśmy moduły obsługujące „twarde" podzespoły, umieszczone w strukturze układu FPGA. Zaprojektujemy teraz logikę, która połączy je w całość. Projekt pokazany jest na **rysunku 16**. Przyłączenie bloku PLL jest stosunkowo proste. Do wejścia dostarczamy sygnał zegarowy, a oba wyjścia łączymy z analogicznymi wejściami ADC. Obsługa przetwornika jest bardziej rozbudowana – jest obsługiwana poprzez dwie magistrale. Pierwsza magistrala, nazwana command, pozwala wybrać kanał oraz rozpocząć wykonywanie pomiaru. Druga, czyli response, służy do odczytywania wyniku.

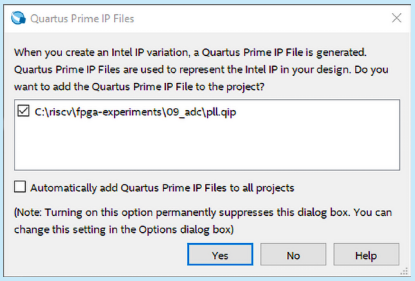
Do wyboru kanału posłuży licznik1. Zlicza on modulo 2, czyli po kolei 0,1,0,1.... Inkrementacja następuje, gdy ADC przyjmie



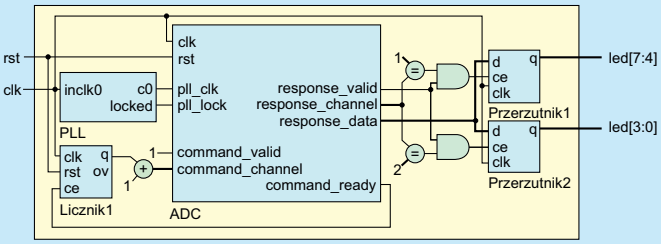

**Rysunek 13. Konfiguracja wejść/wyjść**




**Rysunek 14. Konfiguracja wyjścia clk 0**




**Rysunek 15. Potwierdzenie dodania bloku IP do projektu**




**Rysunek 16. Schemat logiki obsługującej przetwornik ADC**

komendę, zlecającą wykonanie poprzedniego pomiaru – uzyskamy to poprzez połączenie wyjścia ready z wejściem CE. Chcemy aby próbkowanie odbyło się z maksymalną częstotliwością, dlatego sygnał valid na stałe ustawiamy na 1. Korzystamy z kanałów 1 i 2, więc do wyjścia z licznika należy dodać jeden. Miejmy na uwadze, że zwiększyła się szerokość sygnału – z jednego do dwóch bitów.

Magistrala wyjściowa składa się z trzech sygnałów: informacji o poprawności danych (valid), numeru kanału (channel) oraz samych danych (data). Tym razem nie mamy wejścia ready. Oznacza to, że nie można zablokować pracy przetwornika, a nieprzetworzone próbki zostaną utracone.

Do wyświetlania wyników każdego z kanałów posłużą po cztery diody LED. Będziemy prezentować tylko cztery najstarsze bity wyniku (z dwunastu bitów dostępnych). Dla każdego kanału przygotujemy osobny bufor. Rolę buforów pełnić będą odpowiednio przerzutniki1 i przerzutniki2. Ich sygnały CE są logiczną sumą informacji o kanale oraz sygnału valid. Piny układu FPGA zostaną podłączone do wyjść przerzutników.

Implementację projektu z rysunku 16 widzimy na **listingu 1**. Zgodnie z projektem, mamy dwa wejścia rst i clk oraz jedno (wektorowe) wyjście, odpowiedzialne za sterowanie diodami świecącymi. Poniżej znajduje się definicja pomocniczych sygnałów, które posłużą do połączenia ze sobą składowych naszego projektu.

Pierwszą składową jest instancja pętli PLL, zawarta w liniach 24…27. Mamy tu jedno wejście inclk0, gdzie podłączamy nasz wejściowy zegar oraz dwa wyjścia c0 i locked. Wyjścia podłączone są bezpośrednio do odpowiadających im wejść bloku ADC.

Linie 29…36 odpowiadają za blok wyboru kanału. Znajduje się tu licznik, tym razem liczący modulo 2. Jego wyjście jest doprowadzone do modułu adc, najważniejszego w naszym projekcie. Pierwsze cztery wejścia odpowiadają za zegary oraz resetowanie. Kolejne pięć sygnałów, zaczynających się od przedrostka command, to magistrala pozwalająca na wybór kanału. Poza sygnałami valid, channel i ready widzimy dwa dodatkowe: startofpacket i endtofpacket, wykorzystywane przy bardziej złożonych funkcjach magistrali. Dodatkowe sygnały nie mają istotnego znaczenia w tym projekcie, dlatego na stałe przypisaliśmy im wartość 0.

Ostatnie pięć sygnałów to magistrala wyjściowa. Rozpoznamy ją po nazwach rozpoczynających się od response. Tutaj także widzimy sygnały startofpacket i endtofpacket, których nie używamy i zostawiamy obok nich puste nawiasy: (). Pozostałe trzy sygnały służą do odczytania wyników. Blok always_ff z linii 54…60 zapisuje ostatni wynik, uzyskany dla danego kanału w podprowadzonym do niego rejestrze. Jeżeli sygnał valid jest ustawiony, to za pomocą bloku case dokonujemy wyboru odpowiedniej części rejestru led.

Przed uruchomieniem programu, przeprowadzimy jego symulację. W skład projektu wchodzą teraz nie tylko moduły, lecz także dostarczone przez producenta bloki IP, wraz z odpowiednimi bibliotekami, które musimy odpowiednio załadować w programie ModelSim. Najpierw jednak przygotujemy test bench, który znajdziemy w pliku *09_ADC/adc_top_tb.sv* (**listing 2**). Listing jest bardzo krótki, dlatego że moduł ma tylko dwa wejścia. W pierwszym bloku initial (linie 14…17), generujemy sygnał zegarowy o okresie 125 ns, czyli częstotliwości 8 MHz. W kolejnym bloku (linie 19…22) znajdziemy sygnał resetu. Na samym końcu (linie 24…27) znajdziemy instancję naszego modułu.

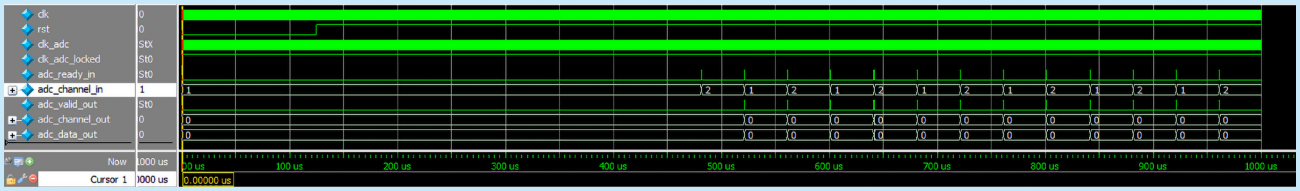
Musimy teraz napisać skrypt, który przygotuje i uruchomi symulację. Jego fragment widzimy na **listingu 3**. Zaczynamy od zdefiniowania stałej QUARTUS_DIR. Przechowuje ona ścieżkę pod którą znajdziemy środowisko Quartus. Jeśli lokalizacja znajduje się w innym miejscu, stała musi zostać zmodyfikowana. Linia 12 odpowiada za załadowanie bibliotek dostarczonych przez producenta układu. Widzimy, że nazwa nie została jeszcze uaktualniona z „Alter-u”

**Listing 1. Implementacja obsługi ADC w języku SystemVerilog (09_ADC/adc_top.sv)**

```
10 module adc_top #(
11  input wire clk,
12  input wire rst,
13  output logic [7:0]led
14 );
15  wire clk_adc;
16  wire clk_adc_locked;
17  wire adc_ready_in;
18  wire adc_counter;
19  wire [4:0]adc_channel_in;
20  wire [4:0]adc_channel_out;
21  wire adc_valid_out;
22  wire [11:0]adc_data_out;
23
24  pll pll_inst (
25      .inclk0(clk),
26      .c0(clk_adc),
27      .locked(clk_adc_locked));
28
29  counter #(.N(2)) ctx_channel (
30      .clk(clk),
31      .rst(rst),
32      .ce(adc_ready_in),
33      .q(adc_counter),
34      .ov());
35
36  assign adc_channel_in = 5'b1 + adc_counter;
37
38  adc adc_inst (
39      .clock_clk(clk),
40      .reset_sink_reset_n(rst),
41      .adc_pll_clock_clk(clk_adc),
42      .adc_pll_locked_export(clk_adc_locked),
43      .command_valid(1'b1),
44      .command_channel(adc_channel_in),
45      .command_startofpacket(1'b0),
46      .command_endofpacket(1'b0),
47      .command_ready(adc_ready_in),
48      .response_valid(adc_valid_out),
49      .response_channel(adc_channel_out),
50      .response_data(adc_data_out),
51      .response_startofpacket(),
52      .response_endofpacket());
53
54  always_ff @(posedge clk)
55  if(adc_valid_out)
56  case (adc_channel_out)
57      5'd1: led[7:4] <= adc_data_out[11-:4];
58      5'd2: led[3:0] <= adc_data_out[11-:4];
59      default: ;
60  endcase
61
62 endmodule
```

**Listing 2. Test bench dla modułu adc_top (09_ADC/adc_top_tb.sv)**

```
11 module adc_top_tb;
12  logic clk, rst;
13
14  initial begin
15      clk <= '0;
16      forever #62.5ns clk <= ~clk;
17  end
18
19  initial begin
20      rst <= 1'b0;
21      #1250 rst <= 1'b1;
22  end
23
24  adc_top dut(
25      .clk(clk),
26      .rst(rst),
27      .led());
28
29 endmodule
```



**Rysunek 17. Wynik symulacji modułu adc_top**

na „Intel”. Poniżej znajduje się lista plików. Dla pętli PLL kompilujemy jeden wygenerowany plik *pll/pll.v*. Następny jest plik z projektem licznika, a poniżej kolejnych osiem opisujących blok IP przetwornika ADC. Na końcu, za pomocą polecenia vsim, uruchamiamy symulację. Opcją -L dołączamy dodatkową bibliotekę. W pozostałej, niezamieszczonej tu, części skryptu, następuje dodanie sygnałów i uruchomienie symulacji na 1 ms. Aby zobaczyć rezultat, uruchamiamy program ModelSim, przechodzimy do katalogu *09_ADC i* wpisujemy polecenie: do adc_top_sim.do

**Listing 3. Kompilowanie modułów w symulacji (09_ADC/adc_top_sim.do)**

```
08 set QUARTUS_DIR "C:/intelFPGA_lite/18.1/quartus/eda/sim_lib/"
09
10 vlib work
11
12 vlog $QUARTUS_DIR/altera_mf.v
13
14 vlog pll/pll.v
15 vlog ../02_counter/counter.sv
16 vlog adc/adc/simulation/submodules/altera_modular_adc_control_avrg_fifo.v
17 vlog adc/adc/simulation/submodules/fiftyfivenm_adcblock_primitive_wrapper.v
18 vlog adc/adc/simulation/submodules/chsel_code_converter_sw_to_hw.v
19 vlog adc/adc/simulation/submodules/fiftyfivenm_adcblock_top_wrapper.v
20 vlog adc/adc/simulation/submodules/altera_modular_adc_control_fsm.v
21 vlog adc/adc/simulation/submodules/altera_modular_adc_control.v
22 vlog adc/adc/simulation/submodules/adc_modular_adc_0.v
23 vlog adc/adc/simulation/adc.v
24 vlog adc_top.sv
25 vlog adc_top_tb.sv
26
27 vsim -L fiftyfivenm_ver -novopt work.adc_top_tb
```

Rezultat symulacji pokazany jest na **rysunku 17**. W pierwszym i trzecim wierszu znajdziemy sygnały zegarowe: wejściowy clk oraz wygenerowany przez PLL clk_adc. Na **rysunku 18** widzimy sygnał w powiększeniu. Możemy zobaczyć, że konfiguracja jest poprawna: wyjściowy sygnał ma częstotliwość cztery razy mniejszą niż wejściowy, czyli w naszym przypadku 2 MHz.

**Listing 4. Konfiguracja sygnałów zegarowych (09_adc/09_adc.sdc)**

```
05 create_clock -name {clk} -period 125.000 -waveform { 0.000 62.500 } [get_ports { clk }]
06 derive_pll_clocks
```

Wróćmy jednak do rysunku 17. Po zresetowaniu widzimy, że moduł ADC potrzebuje około 450 µs na rozpoczęcie działania. Gotowość do pracy jest sygnalizowana pojawieniem się wartości 1 na wyjściu ready. Cały czas na wejściu channel ustawiona była wartość 1. W momencie przyjścia sygnału ready nastąpiło zwolnienie sygnału CE licznika. Dzięki temu od razu (czyli w kolejnym takcie zegara) nastąpi wybór kanału 2. Pozostanie w takim stanie, aż do jego obsłużenia, zasygnalizowanego pojawieniem się kolejnej jedynki na wyjściu ready.

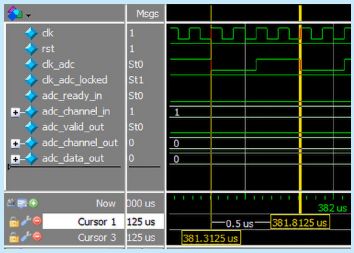
Trzy ostatnie wiersze obrazują magistralę wyjściową. Jeżeli sygnał valid jest w stanie niskim, na pozostałych dwóch sygnałach także ustawione są zera. Wartość zmienia się tylko na jeden takt zegara – moment ten pokazuje **rysunek 19**. Zwróćmy uwagę, że gdy dostępne jest wyjście, blok ADC jest gotowy do rozpoczęcia kolejnego przetwarzania. Za pomocą kursorów możemy zmierzyć, że pomiędzy sygnałem ready i valid mija 40 µs, co odpowiada ustawionej częstotliwości próbkowania, czyli 25 kHz.

Zweryfikowaliśmy projekt w symulacji, czas uruchomić go w sprzęcie. Wracamy do projektu w środowisku Quartus. Musimy przyporządkować piny układu FPGA dla wejść i wyjść naszego modułu. Wejścia analogowe są przypisane na stałe i nie musimy ich ustawiać.

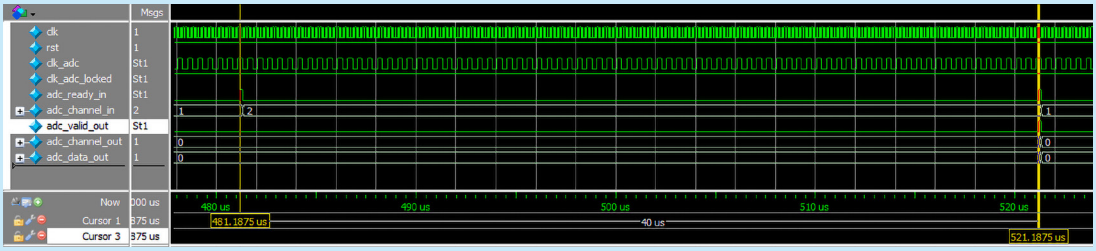
Powinniśmy także skonfigurować zegary. Ponieważ tym razem mamy dwa sygnały zegarowe, należy o tym poinformować narzędzie odpowiedzialne za syntezę, czyli plik *09_adc.sdc* pokazany na **listingu 4**. Tak jak w poprzednich projektach, najpierw konfigurujemy zewnętrzny sygnał. Tym razem w linii 6, znajdziemy informację o dodatkowych zegarach pochodzących z pętli PLL. Nie musimy dodawać ich parametrów, ponieważ zostaną pobrane z modułu IP.

Po zakończeniu budowy projektu, w informacji o zegarach znajdziemy podsumowanie takie jak pokazane na **rysunku 20**. Warto przyjrzeć się rozłożeniu elementów w narzędziu *Chip Planner* (**rysunek 21**). Kolorem czerwonym zaznaczono lokalizację przetwornika ADC, a zielonym, lokalizację pętli PLL. Polecam obejrzeć je szczegółowo i przyjrzeć się ich opisowi.

Mając zbudowany projekt, możemy przystąpić do testowania działania na sprzęcie. W tym celu, zgodnie ze schematem z **rysunku 22**, do wejść A0 i A1 podłączamy potencjometry. Przykładową realizację na płytce stykowej możemy zobaczyć na **fotografii 1**. Po zaprogramowaniu układu FPGA, będziemy mogli zmieniać stan diod poprzez obrót gałek. Wynik końcowy został zaprezentowany na filmie [4].



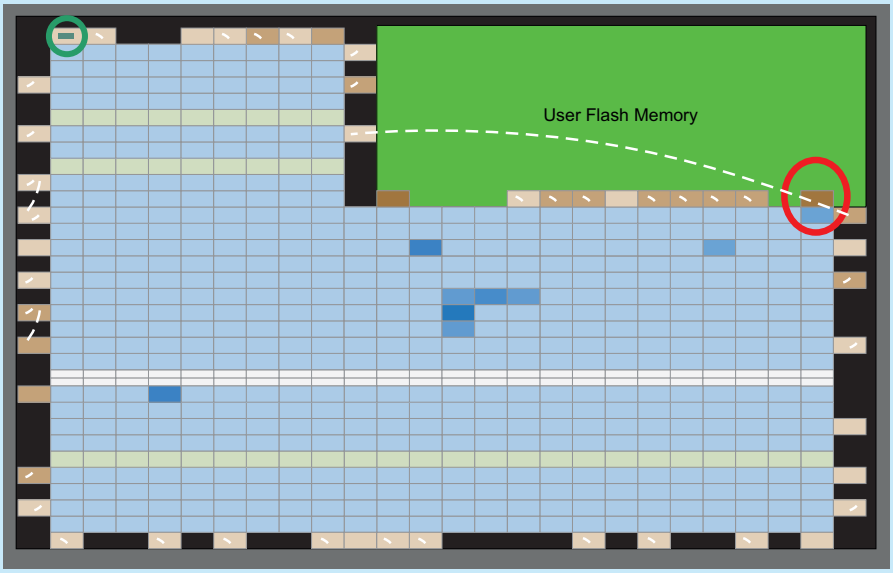
**Rysunek 18. Porównanie sygnałów zegarowych**



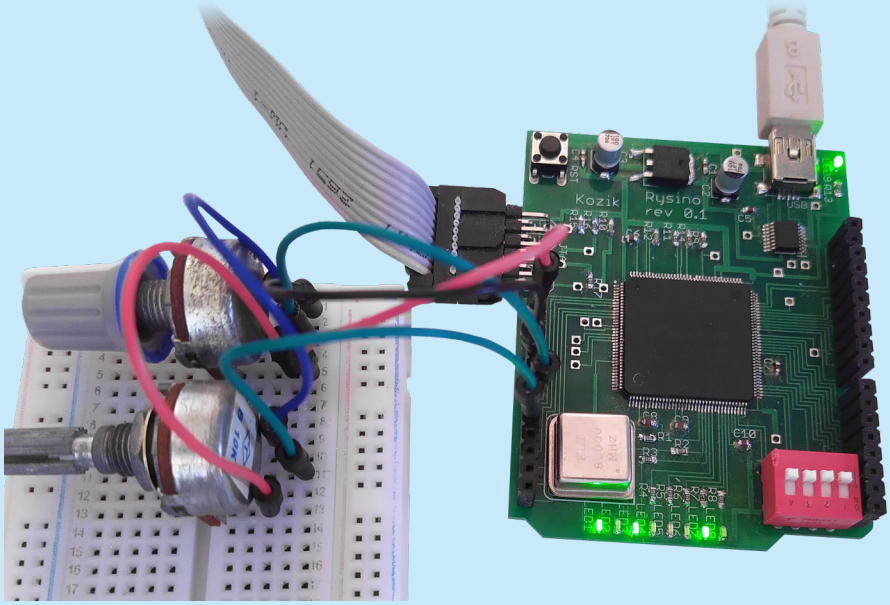
**Rysunek 19. Zbliżenie na moment odczytu wyników pomiarów**

| | Clock Name | Type | Period | Frequency | Rise | Fall | Duty Cycle | Divide by | Multiply by | Phase | Offset | Edge List | Edge Shift | Inverted | Master | Source | Targets |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | clk | Base | 125.000 | 8.0 MHz | 0.000 | 62.500 | | | | | | | | | | | { clk } |
| 2 | pll_inst\|al...pll1\|clk[0] | Ge...ed | 500.000 | 2.0 MHz | 0.000 | 250.000 | 50.00 | 4 | 1 | | | | | false | clk | pll_i...lk[0] | { pll_inst\|al...pll1\|clk[0] } |

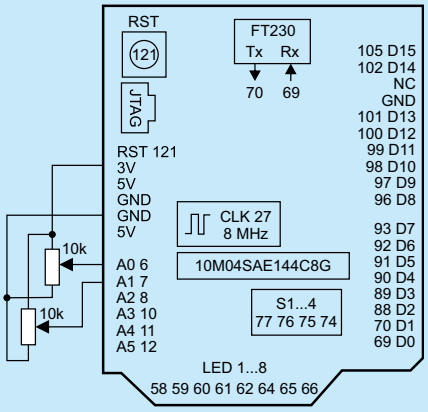
**Rysunek 20. Zegary znalezione w projekcie**




**Rysunek 21. Lokalizacja przetwornika ADC (czerwony) i pętli PLL (zielony) w strukturze układy FPGA**



**Fotografia 1. Gotowy model podczas testów**




**Rysunek 22. Sposób podłączenia potencjometrów do Rysino**

## Podsumowanie

W tym odcinku zapoznaliśmy się z działaniem przetwornika analogowo-cyfrowego, wbudowanego w strukturę układu MAX10. Użyjemy go w przyszłości, przy realizacji kilku projektów z zakresu cyfrowego przetwarzania sygnałów. W kolejnym odcinku nauczymy się korzystać z narzędzia Signal Tap, które pozwala podejrzeć rzeczywiste sygnały z wnętrza układu.

**Rafał Kozik**
**rafkozik@gmail.com**

**Bibliografia:**

[1] Intel MAX 10 Analog to Digital Converter User Guide, Intel 2020, *https://intel.ly/3fKlMTL*
[2] Pin Information for the Intel® MAX®10 10M04SA Device, Intel 2017, *https://intel.ly/2NgMuHp*
[3] Intel MAX 10 Clocking and PLL Overview, Intel 2018, *https://intel.ly/2Ng0RMf*
[4] Film z instrukcją generowania bloków IP i demonstracją działania projektu, *https://bit.ly/2Nja5r4*

Miesięcznik **„Elektronika Praktyczna”** (12 numerów w roku) jest wydawany przez AVT-Korporacja Sp. z o.o. we współpracy z wieloma redakcjami zagranicznymi.

**Wydawca:**
AVT-Korporacja Sp. z o.o.
03-197 Warszawa, ul. Leszczynowa 11
tel.: 22 257 84 99, faks: 22 257 84 00

**Adres redakcji:**
03-197 Warszawa, ul. Leszczynowa 11
tel.: 22 257 84 60
faks: 22 257 84 00
e-mail: redakcja@ep.com.pl
*www.ep.com.pl*

**Redaktor Naczelny:**
Wiesław Marciniak

**Redaktor Programowy, Przewodniczący Rady Programowej:**
Piotr Zbysiński

**Zastępca Redaktora Naczelnego, Redaktor Prowadzący:**
Damian Sosnowski

**Zastępca Redaktora Naczelnego, Menedżer Magazynu**
Marcin Karbowniczek

**Szef Pracowni Konstrukcyjnej:**
Grzegorz Becker, tel.: 22 257 84 58

**Redaktor strony internetowej** *www.ep.com.pl*
MAD Sp. z o.o.

**Zespół marketingu i reklamy:**
Katarzyna Gugała, tel.: 22 257 84 64
Bożena Krzykawska, tel.: 22 257 84 42
Grzegorz Krzykawski, tel.: 22 257 84 60

**Sekretarz Redakcji:**
Grzegorz Krzykawski, tel.: 22 257 84 60

**DTP i okładka:**
MAD Sp. z o.o.

**Stali Współpracownicy:**
Jacek Bogusz, Lucjan Bryndza, Jarosław Doliński, Andrzej Gawryluk, Krzysztof Górski, Tomasz Jabłoński, Michał Kurzela, Szymon Panecki, Sławomir Skrzyński, Ryszard Szymaniak, Adam Tatuś, Robert Wołgajew

**Uwaga!**
Kontakt z wymienionymi osobami jest możliwy via e-mail, według schematu: imię.nazwisko@ep.com.pl

**Prenumerata w Wydawnictwie AVT**
*www.avt.pl/prenumerata*
lub tel.: 22 257 84 22
e-mail: prenumerata@avt.pl
*www.sklep.avt.pl*, tel.: 22 257 84 66

**Prenumerata w RUCH S.A.**
*www.prenumerata.ruch.com.pl*
lub tel.: 801 800 803, 22 717 59 59
e-mail: prenumerata@ruch.com.pl

Wydawnictwo AVT-Korporacja Sp. z o.o. należy do **Izby Wydawców Prasy**