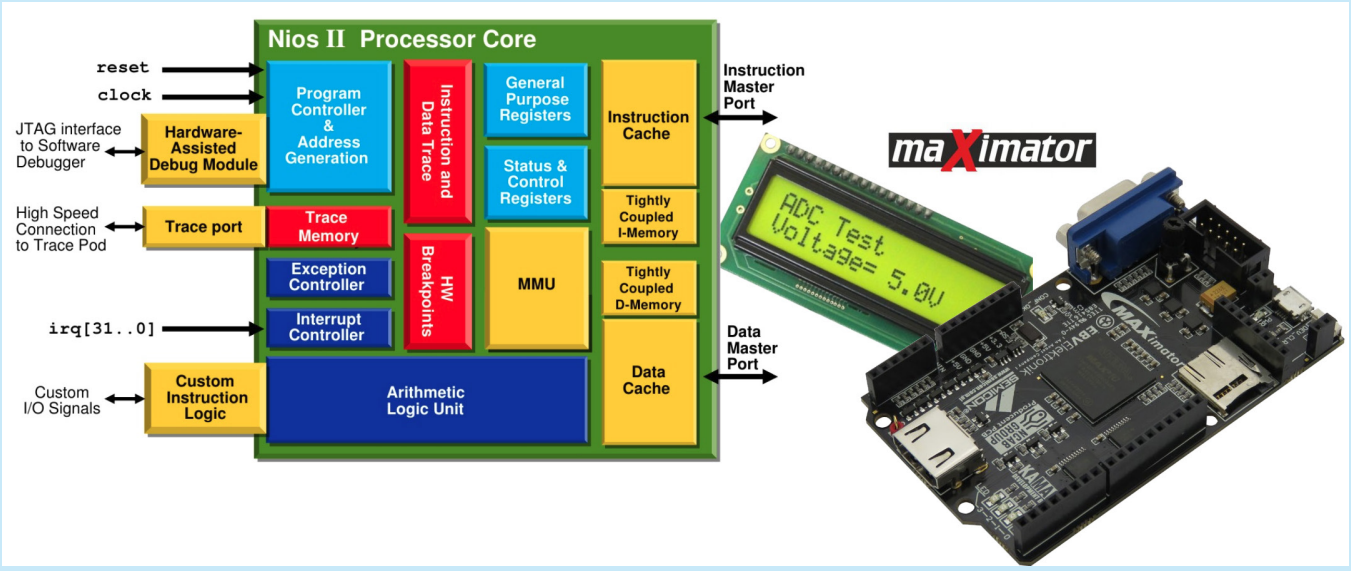


NIOS II na maXimatorze, czyli mikroprocesor w układzie FPGA (8)

Przetwornik analogowo-cyfrowy

Zajmiemy się wykorzystaniem ciekawej możliwości dostępnej w układzie FPGA zastosowanym na płytce maXimatora, którą jest wbudowany przetwornik analogowo-cyfrowy, umożliwiający dokonywanie pomiarów napięcia, czyli sygnałów analogowych. To bardzo ważna możliwość, gdyż wiele sygnałów w otaczającym nas świecie to wartości analogowe, które w prostszy lub bardziej skomplikowany sposób możemy przetworzyć na napięcie.

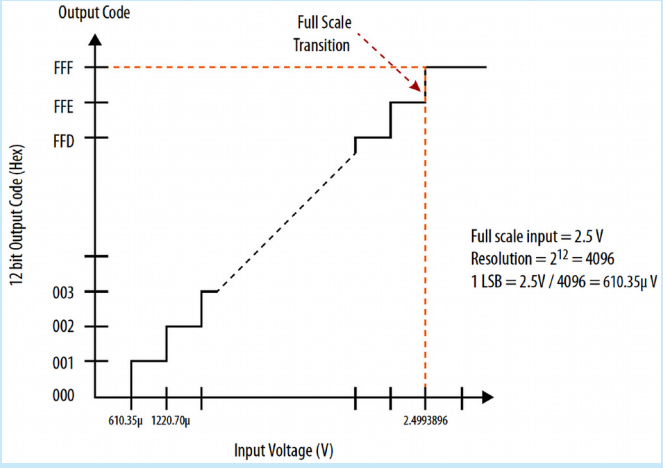
Do testów działania przetwornika A/C użyjemy czujnika temperatury STLM20 zamontowanego na module maXimator Expander. Oprócz tego poznamy także metodę, na przystosowanie bibliotek przygotowanych dla innych układów do pracy z naszym systemem, w myśl zasady, aby nie odkrywać koła na nowo – zrobimy to na przykładzie sterowania wyświetlaczem HD44780.

Krótkie wprowadzenie

Na samym początku w krótkich słowach wyjaśnijmy sobie czym jest przetwornik analogowo-cyfrowy (ADC, ang. *Analog to Digital Converter*). Jak sama nazwa wskazuje jest to układ, który przetwarza wartości analogowe (najczęściej napięcie), na wartości cyfrowe. Czyli w skrócie dokonujemy pomiaru napięcia i zapisujemy go w formie liczby. W zależności od rozdzielczości przetwornika liczba ta zapisana jest za pomocą większej lub mniejszej ilości bitów. Do przetwornika oprócz mierzonego sygnału należy doprowadzić napięcie referencyjne, czyli wartość, z którą porównywane będzie napięcie mierzone. O ile mówimy o przetwornikach nie posiadających wejść różnicowych, to zwykle układ taki dla napięcia równego 0 V będzie podawał liczbę 0, zaś dla napięcia równego napięciu odniesienia – maksymalną liczbę możliwą do zapisania na określonej ilości bitów.

Na **rysunku 1** pokazano idealną, „schodkową" charakterystykę przetwornika dla napięcia referencyjnego (w wypadku naszego ADC jest ono równoznaczne z pełnym zakresem pomiarowym, czyli *Full scale input*) wynoszącego 2,5 V – pokazuje ona w jakiś sposób 12-bitowy kod jest przypisywany do danego zakresu napięć. Jest to zjawisko kwantyzacji.

Drugim parametrem ważnym dla konwersji analogowo-cyfrowej jest także częstotliwość próbkowania – w systemach cyfrowych nie



Rysunek 1. Charakterystyka idealna przetwornika ADC w układzie FPGA rodziny MAX10 dla napięcia referencyjnego 2.5V Źródło: Intel MAX 10 Analog to Digital Converter User Guide 17.1

możemy mierzyć napięcia nieskończenie szybko, a także większa szybkość próbkowania wymaga bardziej zaawansowanej konstrukcji samego przetwornika ADC. Ten parametr ma rzecz jasna znaczenie szczególnie przy pomiarach przebiegów zmiennych, gdzie próbkowanie musi odbywać się z częstotliwością przynajmniej dwukrotnie wyższą niż najwyższa częstotliwość występująca w sygnale. W przeciwnym wypadku narażamy się na zjawisko aliasingu.

Nasz przetwornik w układzie MAX10 pozwala na pomiar napięć z zakresu 0-2,5 V, zaś na płytce (o czym wspominałem wcześniej) umieszczono odpowiednie diody, które chronią te piny przed podaniem zbyt dużego napięcia. Dodatkowo pin FPGA *D1* (*ADC1_6, ANIN3*) posiada wbudowany, możliwy do włączenia dzielnik napięcia 1:2, dzięki czemu możemy mierzyć napięcia w nieco szerszym zakresie (jednak nie możemy przekroczyć napięcia 3,3 V).

Ważną kwestią jest też fakt, że nasz układ posiada tylko jeden przetwornik ADC oraz multiplekser, który przełącza kanały, na których prowadzony jest pomiar. Dlatego jeśli będziemy prowadzić pomiary na 2 kanałach, przetwornik będzie musiał wykonywać te czynności sekwencyjnie: najpierw dokonać konwersji na pierwszym kanale, potem przełączyć multiplekser i dokonać konwersji na kolejnym kanale.

Czujnik temperatury STLM20

Układ ten nie wymaga chyba bardziej obszernego komentarza. Napięcie na jego wyjściu jest w przybliżeniu liniowo zależne od temperatury, według zależności:

$$T[^{\circ}C] = \frac{1,8663V - V_{out}[V]}{0,01169V/^{\circ}C}$$

Jest to najbardziej ogólny i najprostszy wzór – w dokumentacji czujnika podane są wzory optymalizowane dla różnych zakresów temperatury, a także wzór uwzględniający pewne nieliniowości czujnika, dzięki czemu możliwe jest bardziej precyzyjne obliczenie temperatury. Biorąc jedna pod uwagę nasze zastosowania oraz parametry zasilania całego układu przedstawiony przeze mnie wzór jest w 100% wystarczający.

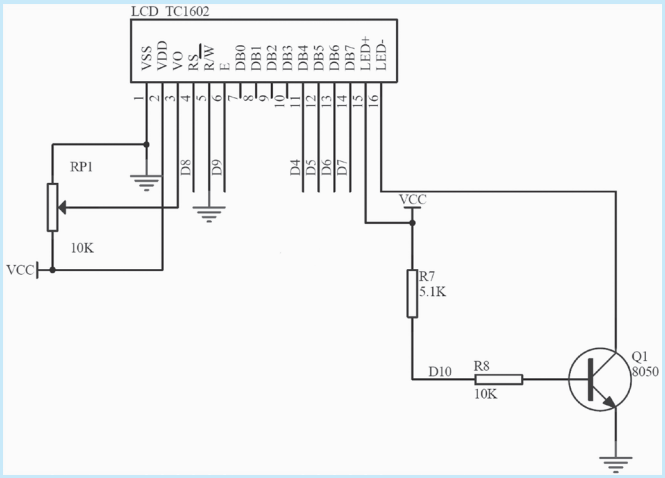
Wyświetlacz HD44780

Wyświetlacz ten to bardzo znany i lubiany bohater wszystkich osób rozpoczynających przygodę z mikrokontrolerami. Jest to wyświetlacz alfanumeryczny, czyli mogący wyświetlać litery i cyfry, oraz pewien zbiór symboli. Pozwala on także na definiowanie 8 wzorów znaków przez użytkownika, dzięki czemu możliwe jest zaimplementowanie znaków narodowych czy prostych symboli.

Jako iż w Internecie możemy znaleźć masę materiałów na temat samego wyświetlacza, w wielkim skrócie omówmy jego podłączenie do układu – schemat połączeń pokazano na **rysunku 2**. W tej konfiguracji wyświetlacz sterowany jest interfejsem 4-bitowym (podłączone tylko linie *DB4...7*), bez możliwości odczytu danych (m. in. flagi zajętości), z powodu zwarcia linii *R/W* do masy. Wyświetlacz musi być zasilany napięciem 5 V – istnieją też specjalne wyświetlacze

Tabela 1. Połączenia wyświetlacza

Pin wyświetlacza	Złącze Arduino	Pin FPGA maXimator	Projekt
DB4	D4	;H16	LCD_DB[0]
DB5	D5	G15	LCD_DB[1]
DB6	D6	G16	LCD_DB[2]
DB7	D7	F16	LCD_DB[3]
RS	D8	E15	LCD_CTL[0]
E	D9	E16	LCD_CTL[1]
Podświetlanie	D10	D15	LCD_CTL[2]



Rysunek 2. Schemat podłączenia wyświetlacza HD44780. Źródło: dokumentacja nakładki LCD Keypad shield DF Robot

które mogą pracować z napięciem 3,3 V. Potencjometr (*RP1*) służy do regulacji kontrastu wyświetlacza, zaś układ z tranzystorem (*Q1*) służy do sterowania podświetlaniem wyświetlacza. Jeśli montujemy sami taki układ na płytce stykowej możemy pominąć ten tranzystor i na stałe podłączyć pin 16 wyświetlacza do masy. Zwykle na wyświetlaczach znajdują się rezystory ograniczające prąd płynący przez moduł podświetlający, jednak teoretycznie możemy trafić na wyświetlacz bez tych rezystorów – wtedy koniecznie musimy zadbać o zamontowanie takiego rezystora. Osobiście nigdy nie spotkałem się z wyświetlaczem pozbawionym tych rezystorów. Rzecz jasna linie sterujące podłączone będą do układu FPGA – niezbędne połączenia wymieniono w **tabeli 1**.

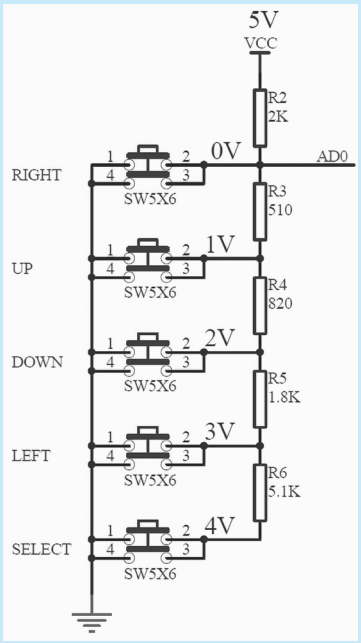
Klawiatura rezystancyjna, czyli jak podłączyć wiele przycisków do 1 wyprowadzenia procesora

Dodatkową „atrakcją" z jaką zetkniemy się stosując wyświetlacz umieszczony na module dla Arduino będzie klawiatura rezystancyjna (którą oczywiście możemy zbudować także na płytce stykowej). Schemat takiej klawiatury przedstawiono na **rysunku 3**.

Zasada działania takiego układu jest banalnie prosta – mamy do czynienia z dzielnikiem napięcia, którego napięcie wyjściowe zależy od tego, który przycisk został wciśnięty. Oczywiście pewną niedogodnością jest fakt, że jednocześnie możemy wykryć tylko jeden przycisk, jednak w wielu aplikacjach (np. budowa menu) taka funkcjonalność w zupełności nam wystarczy. Oczywiście w razie potrzeby możemy rozszerzać taki układ – jedynym ograniczeniem jest moment, w którym szumy układu nie pozwolą nam na każdorazowe i precyzyjne rozróżnienie przycisków.

Uważniejsi z Was na pewno zauważą, ze układ ten sprawi nam pewien problem – mianowicie na wyjściu otrzymywać będziemy napięcia z zakresu 0-5 V. Oczywiście jeśli budujemy układ sami rozwiązanie jest oczywiste – zasilić układ napięciem 3,3 V i użyć odpowiednich rezystorów, aby po naciśnięciu najniższego przycisku napięcie spadało poniżej 2,5 V, lub nawet polegać na zasilaniu 5 V i zabezpieczeniach płytki maXimator, jednak tak dobrać rezystory, aby po przyciśnięci najniższego przycisku napięcie także spadało poniżej 2,5 V. Co jednak w sytuacji, gdy jesteśmy postawieni przed gotowym modułem, w którym ciężko cokolwiek zmienić (np. na moim module rezystory są pod wyświetlaczem LCD, który jest przylutowany do płytki)? W takim wypadku najlepiej zastanowić się nieco dłużej nad problemem, aby stwierdzić, że wystarczy dołączyć dodatkowy rezystor pomiędzy wyjście tego układu (na schemacie oznaczone *AD0*) a masę. Rezystancja tego opornika powinna być co najwyżej taka, aby napięcie przy braku przyciśniętych przycisków

nie przekraczało 2,5 V, zaś z drugiej strony wartość ta nie może być zbyt mała – idealnie aby była ona równa lub nieco niższa od wartości rezystora oznaczonego na schemacie jako *R2*. Niestety te nie zawsze są zgodne z podanymi na schemacie, dlatego najlepiej zawsze użyć omomierza i przy module odłączonym od maXimatora zmierzyć rezystancję pomiędzy zasilaniem 5 V a wyprowadzeniem A0. Rezystor możemy dołączyć np. lutując go do płytki (zwykle cała powierzchnia płytek wypełniona jest polem masy, więc łatwo o dostęp do niej), lub wykorzystując moduły ze złączami do przykręcania kabelków (tzw. *Screw Shield*). W moim wypadku rezystor R2 miał wartość 3 kΩ, zaś jako rezystora ograniczającego napięcie użyłem najbliższy jaki miałem: 2,2 kΩ (**fotografia 4**).



Rysunek 3. Schemat klawiatury rezystancyjnej. Źródło: dokumentacja nakładki LCD Keypad shield DF Robot

Czas wyruszyć na pomiary

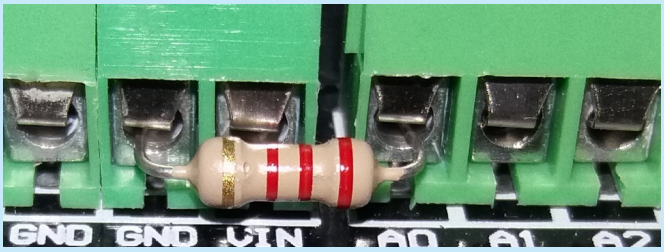
Po tym krótkim wstępie, który trochę napuchnął pod wpływem ilości informacji, zabierzmy się za dodanie przetwornika ADC do naszego układu. Zacznijmy także od minimalnego projektu, zawierającego jedynie konieczne do pracy procesora komponenty, moduł *JTAG UART*, moduł wyjściowy do sterowania 4-rema diodami na płytce (opcjonalnie) oraz timer, który już wcześniej przygotowaliśmy. Dodatkowym dokumentem, który dokumentuje przetwornik ADC i z którym powinniśmy się zaprzyjaźnić jest *Intel MAX 10 Analog to Digital Converter User Guide*. Dodajemy ADC. W *IP Catalog* wyszukujemy *ADC* i wybieramy *Altera Modular ADC Core* (**rysunek 5**).

Tradycyjnie omówmy po krótce parametry tego modułu:

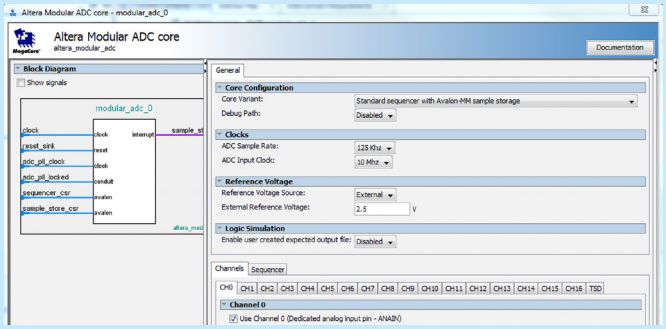
Core variant – pozwala na wybór różnych opcji kontrolowania próbkowania i pobierania próbek.

Domyślna opcja (*Standard sequencer with Avalon-MM sample storage*) powoduje, że do przetwornika ADC zostanie dołączony sekwencjoner wraz z pamięcią. Dzięki temu po wywołaniu polecenia konwersji (pomiaru) układ pobierze pomiary ze wskazanych kanałów i zapisze je w specjalnej pamięci, z której dane te będziemy mogli odczytać.

Druga opcja (*… with threshold violation detection*) dodatkowo dodaje funkcje wykrywania przekroczenia dla każdego aktywnego



Fotografia 4. Przykład montażu rezystora ograniczającego napięcie na wyjściu klawiatury rezystancyjnej



Rysunek 5. Okno konfiguracji IP Core ADC

kanału wartości minimalnej i maksymalnej, a informacje o takim darzeniu udostępnia za pomocą interfejsu Avalon-ST

Kolejna opcja (*… with external sample storage*) powoduje umieszczenie samego sekwencjonera bez pamięci. Zamiast niej dane są udostępnianie za pomocą interfejsu Avalon-ST

Ostatnia opcja (*ADC control core only*) pozostawia nas jedynie z układem kontrolującym przetwornik, wyposażonym w dwa interfejsy Avalon-ST – jeden do przesyłania komend (kolejnych numerów kanałów do przeprowadzenia pomiarów) oraz drugi, na którym udostępniane są wyniki pomiarów.

My w naszym projekcie wybierzmy pierwszą z opcji

Debug Path – opcja pozwala dodać do modułu ADC specjalny adapter pozwalający na monitorowanie pracy przetwornika za pomocą specjalnego modułu oprogramowania Quartus. Pozostawiamy tę opcję wyłączoną.

ADC Sample Rate – prędkość próbkowania – definiuje ile czasu przetwornik poświęca na pobranie jednej próbki. Jeśli korzystamy z więcej niż jednego kanału efektywna maksymalna prędkość próbkowania będzie wynikiem podziału tej częstotliwości na ilość używanych kanałów. W naszym przykładzie wybierzmy 125 kHz.

ADC Input Clock – definiuje jaką częstotliwość ma sygnał zegarowy taktujący przetwornik. Po wyborze prędkości próbkowania możemy wybrać jedną z częstotliwości taktowania dostępnych dla danej częstotliwości próbkowania. Przykładowo dla wybranej wcześniej częstotliwości 125 kHz możemy przetwornik taktować tylko sygnałem 10 MHz. Co więcej sygnał ten musi pochodzić z pętli PLL numer 1 lub 3 (o to powinien zadbać *Fitter*), a dokładnie z wyjścia *c0* tej pętli.

Reference Voltage Source – możemy wybrać skąd ma być pobierane napięcie referencyjne dla przetwornika. Do wyboru mamy wewnętrzne źródło (*Internal*) o napięciu 2.5V, lub podanie tego napięcia z zewnątrz (*External*) – na płytce maXimator pod odpowiednie wejście przetwornika podpięte jest także napięcie 2.5V. Napięcie to wpisujemy w kolejnym polu – *External Reference Voltage*.

Logic simulation – pozwala na symulowanie pracy przetwornika w czasie symulacji całego układu. Pozostawiamy opcję bez zamin.

Channels – w kolejnych zakładkach zaznaczamy czy chcemy używać danego kanału, czy nie. Do wyboru mamy kanał *CH0* (dedykowany, u nad podłączony do potencjometru na płytce maXimatora), 16 zwykłych kanałów (z czego na płytce maXimatora dostępnych mamy 6 z nich, dodatkowo dla kanału 8 możemy włączyć wewnętrzny podzielnik 1:2) oraz kanał *TSD*, czyli wewnętrzny czujnik temperatury układu FPGA (nie może on służyć do pomiaru temperatury otoczenia, gdyż układ FPGA się nagrzewa i to własne jego temperaturę możemy mierzyć). W naszym wypadku wybieramy kanały *CH0* oraz *CH15* (*ANIN0*).

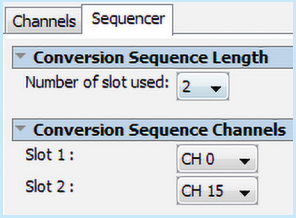
Sequencer – tu ustawiamy w jakiej sekwencji pobierane będą próbki z poszczególnych kanałów (**rysunek 6**).

Number of slot used – definiujemy jaka jest długość sekwencji (możemy wybrać od 1 do 64 elementów)

Conversion Sequence Channels – tu definiujemy jakie kanały będą próbkowane dla kolejnych kroków sekwencji (tu nazywane

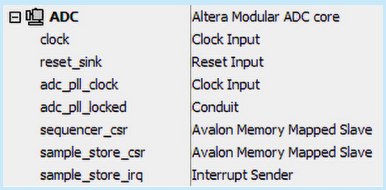
są slotami). Możemy w zależności od potrzeby kilkukrotnie w obrębie danej sekwencji próbkować ten sam kanał (aby móc potem np. wykonać uśrednianie)

W naszym wypadku ustawiamy parametry jak na rysunku 6.



Rysunek 6. Ustawienia sekwencjonera

Teraz możemy dodać nasz przetwornik ADC do systemu i zacząć zastanawiać się jak to wszystko podłączyć…



Rysunek 7. Wyprowadzenia modułu ADC

Zanim jednak do tego przystąpimy, musimy zrobić porządek związany z pętlą PLL – po pierwsze mamy taktować nasz przetwornik częstotliwością 10MHz (którą musimy wygenerować), a dodatkowo musi to być taktowanie z zegara c0 pętli PLL. Co więcej rzucając szybko okiem na wyprowadzenia modułu ADC (**rysunek 7**) – potrzebne będzie nam wyjście informujące o prawidłowej synchronizacji pętli PLL (*pll_locked*).

Kliknijmy zatem w *Platform Designer* na naszą pętlę PLL, a następnie wybierzmy *Edit Parameters…* i wykonujemy następujące czynności:

- W zakładce *[1] Paremeter Settings → Inputs/Locks* zaznaczamy opcję *Create 'locked' output.*
- W zakładce *[3] Output Clocks → clk c0* zmieniamy częstotliwość na 10 MHz.
- W zakładce *[3] Output Clocks → clk c1* aktywujemy to wyjście (*Use this clock*) oraz podajemy czestotliwosc 50 MHz.
- Klikamy 2-krotnie na *Finish*.
- Następnie musimy uporządkować sygnał zegarowy – jak zapewne się domyślacie teraz system będziemy taktować z wyjścia *c1*, zaś wyjście *c0* posłuży wyłącznie do taktowania przetwornika. W związku z tym musimy przepiąć sygnały zegarowe wszystkich komponentów na sygnał zegarowy *c1*.
- Następnie zabieramy się za przyłączenie modułu ADC.
- Dołączamy wejście *clock* do *c1*.
- Dołączamy reset.
- Dołączamy do *adc_pll_clock* sygnał zegarowy *c0*.
- Do *adc_pll_locked* podłączamy (taką samą metodą jaką używaliśmy przy sygnałach *Conduit* w czasie zabaw z I²C) wyjście *locked_conduit* z pętli PLL.
- *sequencer_csr* oraz *sample_store_csr* podłączamy do *data_master* z CPU.

sample_store_irq podłączamy do linii przerwań.

Zapewne zastanawiacie się dlaczego podłączaliśmy dwa porty do magistrali danych? otóż producent tak zaprojektował moduł ADC, że sekwencjoner i pamięć próbek mają osobne porty komunikacyjne i (co możecie sprawdzić) osobne adresy na magistrali. Jeśli już mówimy o adresach – przypominam o konieczności uruchomienia odpowiednich opcji dotyczących przypisania adresów i numerów przerwań. Po tym wszystkim zapisujemy i generujemy system, a następnie uruchamiamy jego kompilację. W tym czasie możemy już przygotować odpowiedni projekt bazowy wraz z BSP w *Eclipse*.

Programujemy woltomierz FPGA

Przetwornik ADC na pokładzie naszego układu może pracować w dwóch trybach – pojedynczego pomiaru, oraz pomiaru ciągłego. Dodatkowo może być generowane przerwanie po zakończeniu każdej sekwencji pomiarów (jednak tym razem nie będziemy korzystać z tej opcji). Korzystając z informacji w dokumentacji przetwornika ADC napiszmy 2 krótkie funkcje (**listing 1**), które pozwolą na inicjalizację oraz odczyt wartości z przetwornika ADC. Funkcja inicjalizująca upewnia się, ze przetwornik

Listing 1. Inicjalizacja i odczyt wartości z ADC

```
void adcInitSingle(void)
{
    adc_stop(ADC_SEQUENCER_CSR_BASE);
    adc_interrupt_disable(ADC_SAMPLE_STORE_CSR_BASE);
    adc_clear_interrupt_status(ADC_SAMPLE_STORE_CSR_BASE);
    adc_set_mode_run_once(ADC_SEQUENCER_CSR_BASE);
}

void adcReadSingle(uint32_t* source_ptr, uint32_t len)
{
    adc_start(ADC_SEQUENCER_CSR_BASE);
    adc_wait_for_interrupt(ADC_SAMPLE_STORE_CSR_BASE);
    adc_clear_interrupt_status(ADC_SAMPLE_STORE_CSR_BASE);
    alt_adc_word_read(ADC_SAMPLE_STORE_CSR_BASE, source_ptr, len);
    adc_stop(ADC_SEQUENCER_CSR_BASE);
}
```

jest wyłączony, następnie wyłącza generowanie przerwań i resetuje flagę przerwania, po czym ustawiany jest tryb pojedynczego pomiaru.

Więcej dzieje się za to w funkcji odczytu:

- uruchamiamy przetwornik,
- oczekujemy na flagę przerwania (flaga zostanie ustawiona, niezależnie od faktu, że nie spowoduje to wygenerowania przerwania) kasujemy ją,
- odczytujemy dane z przetwornika po czym upewniamy się, że przetwornik zostanie zatrzymany (mimo iż powinno to stać się automatycznie po dokonaniu pomiaru).

Komentarza może wymagać tu chyba tylko sam odczyt – podobnie jak w wypadku pracy z modułem SPI – podajemy wskaźnik na tablicę elementów 32-bitowych (source_ptr) oraz ilość słów, które mają zostać pobrane (len). Każde 32-bitowe słowo zawiera dane z kolejnego slotu – według ustawień Conversion Sequence Channels.

Należy tu zachować szczególną ostrożność, gdyż program nie jest w stanie sprawdzić ile elementów ma tablica, do której chcemy zapisać dane – zapis np. 3 elementów do 2 elementowej tablicy może skończyć się poważnymi problemami, gdyż trzeci element zostanie zapisany w bliżej nieznanym miejscu w pamięci RAM – jest to tak zwany wyciek pamięci.

Czego nam jeszcze brakuje? Funkcji, która pozwoli przeliczyć nam otrzymany wynik (z zakresu 0…4095) na napięcie (z zakresu 0-2.5 V). Tu także posłużymy się znanym nam już trickiem, dzięki któremu nie musimy stosować liczb zmiennoprzecinkowych:

```
uint32_t adcVoltage(uint32_t adcValue){
    return adcValue * 250000 / 4095;
}
```

Liczba 250000 to nic innego jak napięcie referencyjne 2.5V pomnożone 100000 razy, zaś funkcja zwraca nam napięcie wyrażone w setkach mikrowoltów. Ostatecznie w programie zapiszemy:

```
alt_putstr("CH0 [V] \t CH15 [V]\r\n");
adcInitSingle();
uint32_t adcData[2];
while (1)
{
    adcReadSingle(adcData, 2);
    adcData[0] = adcVoltage(adcData[0]);
    adcData[1] = adcVoltage(adcData[1]);
        printf("%lu.%05lu  \t  %lu.%05lu  \r\n",
adcData[0]/100000, adcData[0]%100000, adcData[1]/100000,
adcData[1]%100000);
    ALT_USLEEP(100000);
}
```

Tabela 2. Przykładowe wyniki pomiarów

Przycisk	Wartość	Próg
Brak	3320	
Select	2650	2985
Left	2058	2354
Down	1475	1766
Up	694	1084
Right	1	347

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

Dzięki temu będziemy otrzymywać na ekranie (w konsoli Nios II) wyniki pomiaru napięcia w woltach na odpowiednich kanałach, wybranych wcześniej. Napięcie na kanale 0 możemy zmieniać kręcąc potencjometrem na płytce, zaś napięcie kanału 15 to napięcie zmierzone na pinie ANIN0. Możemy w celu testów podpiąć do zestawu płytkę maXimator ekspander i zająć się pomiarem temperatury.

Analogowy pomiar temperatury

Bez zbędnych wstępów napiszmy prostą funkcję, która skonwertuje napięcie (w formacie jaki zwraca go wcześniej napisana funkcja) na temperaturę wyrażoną w dziesiątych częściach stopnia Celsjusza.

```
int32_t calculateTemp(uint32_t voltage){
    return ((186630 - voltage) * 10) / 1169;
}
```

Ponadto, zmieńmy główną funkcję do wyświetlania tych danych:

```
adcData[1] = adcVoltage(adcData[1]);
int32_t temp = calculateTemp(adcData[1]);
printf("%lu.%d\r\n", temp/10, abs(temp)%10);
```

Po uruchomieniu programu na pewno zauważymy, że odczyt temperatury jest mocno niestabilny. To samo zapewne zauważyliście w przypadku samego pomiaru napięcia… Co więc zrobić z tymi szumami? Czy walczyć z nimi analogowo, poprzez np. zmianę zasilania przetwornika? Niekoniecznie. W tej sytuacji rzeczywiście potrzebujemy odczytu pozbawionego szumów, ale niekoniecznie szybkiego, możemy więc posłużyć się bardzo prostą techniką, dostępną także w oscyloskopach, czyli uśrednianiem. Całą „zabawa" polega zatem na wykonaniu pomiaru kilka-kilkadziesiąt razy pod rząd, a następnie policzenia średniej z tychże pomiarów. Przykładowo zrealizować możemy to w następujący sposób:

```
int32_t temp = 0;
for(int i = 0 ; i < 100 ; i++){
    adcReadSingle(adcData, 2);
    temp += adcData[1];
}
temp /= 100;
temp = adcVoltage(temp);
temp = calculateTemp(temp);
```

Teraz już temperatura jest niezwykle stabilna. Podobny efekt pobrania wielu pomiarów możemy osiągnąć także za pomocą omawianej wcześniej konfiguracji sekwencjonera. Po osiągnięciu takiego pomiaru temperatury dobrym zadaniem domowym będzie wyświetlenie tej informacji na wyświetlaczu 7-segmentowym, a tymczasem przejdźmy do kolejnego zadania, jakim jest obsługa klawiatury rezystancyjnej.

Analogowe przyciski

Na początek zabawy z naszymi przyciskami (po ich podłączeniu zamiast maXimator Expandera albo na module, albo na płytce stykowej, zgodnie z tym, co powiedzieliśmy wcześniej) musimy dokonać ich „kalibracji". Modyfikując powyższy program mierzący temperaturę, tak aby wyświetlał uśrednioną wartość z przetwornika, zapisujemy jakie wartości są zwracane dla każdego wciśniętego przycisku, oraz wszystkich przycisków zwolnionych. Wartości te rzecz jasna segregujemy monotonicznie (np. malejąco), a następnie licząc (i zaokrąglając w dowolny sposób) średnią z sąsiednich wartości ustawiamy progi decyzyjne, czyli wartości według których rozgraniczymy który przycisk jest wciśnięty. Przykładowe wyniki pomiarów dla mojego modułu, oraz ustalone przeze mnie progi podaję w **tabeli 2**.

Następnie na warsztat weźmy już wcześniej opracowane funkcje dotyczące obsługi przycisków (Lekcja 4) – przecież cała nasza praca i w tym przypadku przyda się, jedynie z drobnymi modyfikacjami. I tak na samym początku musimy zmienić plik nagłówkowy tak, aby przygotować się na 5 przycisków, oraz aby podać w nim wyliczone przed chwilą wartości progów.

```
//definicje aliasów dla przycisków
#define KBD_NONE      255
#define KBD_S  0
#define KBD_L  1
#define KBD_D  2
#define KBD_U  3
#define KBD_R  4
#define KBD_N_S_TRH  2985
#define KBD_S_L_TRH  2354
#define KBD_L_D_TRH  1766
#define KBD_D_U_TRH  1084
#define KBD_U_R_TRH  347
//...
//tabela struktur - dla każdego przycisku
menu_item_t menu_items[5];
```

Następnie modyfikujemy funkcję odczytującą przedtem stan przycisków, tak aby była ona wstanie „zdekodować" przycisk na podstawie odczytanej wartości analogowej:

```
uint8_t kbd_read(uint32_t adcValue){
    if(adcValue < KBD_U_R_TRH)
    {
        return KBD_R;
    } else if(adcValue < KBD_D_U_TRH)
{
        return KBD_U;
    } else if(adcValue < KBD_L_D_TRH)
    {
        return KBD_D;
    } else if(adcValue < KBD_S_L_TRH)
    {
        return KBD_L;
    } else if(adcValue < KBD_N_S_TRH)
    {
        return KBD_S;
    } else {
        return KBD_NONE;
    }
}
```

Listing 2. Obsługa ADC

```
//Inicjalizacja ADC
adcInitCont();
//szybkie wypełnienie tablic definiujących akcje przycisków
for(int i = 0 ; i < 5 ; i++)
{
    menu_items[i].click.action = kbdCallbackClick;
    menu_items[i].click.parameter = i;
    menu_items[i].hold.action = kbdCallbackHold;
    menu_items[i].hold.parameter = i;
}
//Ustawienie przerwania od timera i uruchomienie timera
alt_ic_isr_register(TIMER0_IRQ_INTERRUPT_CONTROLLER_ID, TIMER0_IRQ, timer0Interrupt, NULL, NULL);
IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER0_BASE, ALTERA_AVALON_TIMER_CONTROL_START_MSK |
ALTERA_AVALON_TIMER_CONTROL_CONT_MSK | ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);

void timer0Interrupt(void* context)
{
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER0_BASE, 0);
    uint32_t adcData[2];
    adcReadCont(adcData,2);
    kbdCheck(adcData[1]);
}

void kbdCallbackClick(uint8_t param){
    printf("Click\t %u\r\n", param);
}

void kbdCallbackHold(uint8_t param){
    printf("Hold\t %u\r\n", param);
}
```

Przytoczony kod chyba mówi sam za siebie, więc pozwolę sobie pominąć jego komentarz. Jedyne co jeszcze zostało nam do zrobienia, to odpalenie timera i konfiguracja wszystkich elementów naszej układanki – odpowiedni fragment programu pokazano na **listingu 2**. Po uruchomieniu programu w konsoli systemu powinniśmy widzieć informacje o klikanych przyciskach (**rysunek 8**).

Jeśli mamy problemy, np. z brakiem wyłapywania kliknięcia za każdym razem, należy sprawdzić przede wszystkim podane progi, a także poeksperymentować z uśrednianiem (uważajmy jednak, aby nie spowodowało to zbyt długiego czasu procesora spędzonego w przerwaniu – pamiętajmy, ze zawsze możemy zwiększyć próbkowanie do 1 MHz z poziomu *Platform Designer*).

Listing 4. Definicje dostępu do pinów sterujących oraz opóźnienia, funkcja inicjalizująca piny sterujące, dwie funkcje odpowiedzialne za kontrolę podświetlania

```
/* Pin definitions */
#define HD44780_RS_LOW    IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(HD44780_CTL_BASE, HD44780_RS_PIN)
#define HD44780_RS_HIGH   IOWR_ALTERA_AVALON_PIO_SET_BITS(HD44780_CTL_BASE, HD44780_RS_PIN)
#define HD44780_E_LOW     IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(HD44780_CTL_BASE, HD44780_E_PIN)
#define HD44780_E_HIGH    IOWR_ALTERA_AVALON_PIO_SET_BITS(HD44780_CTL_BASE, HD44780_E_PIN)

#define HD44780_E_BLINK   HD44780_E_HIGH; HD44780_Delay(20); HD44780_E_LOW; HD44780_Delay(20)
#define HD44780_Delay(x)  ALT_USLEEP(5*x)
Funkcja wysyłania komendy do wyświetlacza:
static void TM_HD44780_Cmd4bit(uint8_t cmd)
{
    /* Set output port */
    IOWR_ALTERA_AVALON_PIO_DATA(HD44780_DATA_BASE, cmd);
    HD44780_E_BLINK;
}

static void TM_HD44780_InitPins(void)
{
    /* Init all pins */
    /* Set pins low */
    IOWR_ALTERA_AVALON_PIO_DATA(HD44780_DATA_BASE, 0);
    IOWR_ALTERA_AVALON_PIO_DATA(HD44780_CTL_BASE, 0);
}

void TM_HD44780_BacklightOn(void)
{
    IOWR_ALTERA_AVALON_PIO_SET_BITS(HD44780_CTL_BASE, HD44780_BL_PIN);
}

void TM_HD44780_BacklightOff(void)
{
    IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(HD44780_CTL_BASE, HD44780_BL_PIN);
}
```

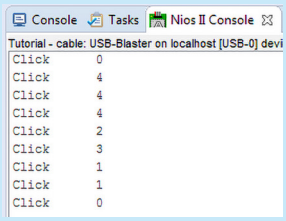
Wyświetlacz HD44780

Teraz zajmijmy się naszym ostatnim bohaterem – wyświetlaczem HD44780. Tym razem jednak nie będziemy wyważać otwartych drzwi! Jest to na tyle popularny wyświetlacz, że powstały już dziesiątki bibliotek do jego obsługi. Nie przejmujmy się tym, że może ciężko znaleźć jakąś dla systemu NIOS II – zaraz przystosujemy bibliotekę, która powstała dla zupełnie innej rodziny.

Uzupełniamy sprzęt

Oczywiście musimy dodać do naszego systemu porty wyjściowe (*PIO*), które pozwolą na sterowanie wyświetlaczem. Zgodnie z tabelką we wstępie stworzymy dwa osobne porty: pierwszy, 4-bitowy, oraz drugi, 3-bitowy, w którym dodatkowo włączymy opcję indywidualnego ustawiania i kasowania bitów.

Do systemu dodamy więc boki pokazane na **rysunku 9**.



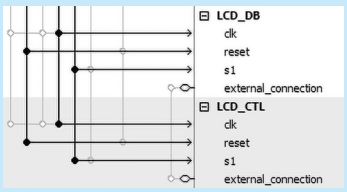
Rysunek 8. Wynik prawidłowego działania programu obsługującego klawiaturę

Biblioteka dla wyświetlacza

Osobiście zdecydowałem się na wybór biblioteki autorstwa Tilen Majerle, przygotowane dla procesorów STM32: ***http://bit.ly/2vqddrM***. Na początek pobierzmy pliki (kopie z momentu pisania tego artykułu dostępne w plikach): tm_stm32_hd44780.h oraz *.h, dodajmy do projektu folder HD44780, w nim umieśćmy te pliki (możemy zmienić ich nazwę usuwając człon dotyczący procesorów STM32) i koniecznie dodajmy je do projektu i ścieżki. Musimy usunąć wszelkie elementy związane z bibliotekami do STM32, w szczególności pliki nagłówkowe, i zastąpić je tymi związanymi z naszym systemem, zmieniamy także definicje pinów i adresów dostępu do peryferiów zgodnie z tym, jak definiujemy to dla systemu NIOS II. Zatem zmieniamy fragment zaczynający się od linijek pokazanych na **listingu 3**. Następnie musimy zmodyfikować dosłownie kilka funkcji odpowiedzialnych za sterowanie pinami GPIO – możemy znaleźć je albo przeglądając kod, albo kompilując go i patrząc na miejsca, gdzie zgłaszane są błędy (**listing 4**). Pierwsza linijka definiuje także ilość znaków w wierszu i ilość wierszy. Z kolei funkcja wyświetlająca przyjmuje współrzędne od których ma zacząć „pisać" po wyświetlaczu. Na koniec zostało nam tylko wykorzystać wyświetlacz do wyświetlenia jakichś informacji:

```
TM_HD44780_Init(16, 2);
TM_HD44780_BacklightOn();
TM_HD44780_Puts(0, 0, "  TEST HD44780  ");
```

Uffff… Chyba czas na podsumowanie?



Rysunek 9. Bloki PIO dodane dla kontrolowania wyświetlacza

Listing 3. Niezbędne zmiany definicji

```
/* definicje
#include "stm32fxxx_hal.h"
#include "defines.h"
#include "tm_stm32_delay.h"
#include "tm_stm32_gpio.h"
zmieniamy na: */
#include <stdint.h>
#include "system.h"
#include "sys/alt_sys_wrappers.h"
#include "altera_avalon_pio_regs.h"
/* 4 bit mode */
/* Control pins */
#define HD44780_CTL_BASE  LCD_CTL_BASE
/* RS - Register select pin */
#define HD44780_RS_PIN    (1<<0)
/* E - Enable pin */
#define HD44780_E_PIN     (1<<1)
/* BL - Backlight pin */
#define HD44780_BL_PIN    (1<<2)
/* Data pins */
#define HD44780_DATA_BASE LCD_DB_BASE
```

Podsumowanie i zadania

W czasie tych zajęć zdobyliśmy naprawdę sporo wiedzy – począwszy od samej obsługi przetwornika ADC, poprzez pomiar napięcia i temperatury, docierając wreszcie do budowy analogowych klawiatur. Wreszcie na zakończenie zmierzyliśmy się z najpopularniejszym wyświetlaczem świata i zobaczyliśmy jak można dostosować biblioteki do jego obsługi pod nasz system. Na zadanie domowe możecie:

- Napisać program (wraz z odpowiednim uzupełnieniem systemu), który wyświetlać będzie temperaturę na wyświetlaczach 7-segmentowych.
- Wyszukać informacje na temat wyświetlacza HD44780.
- Napisać program umożliwiający wyświetlanie tablicy znaków, jaką dysponuje nasz wyświetlacz.
- Nauczyć się definiować własne znaki (w celu wyświetlenia np. polskich znaków narodowych).

W wykonaniu powyższych zadań mogą pomóc m. in materiały dostępne na stronie, z której pobieraliśmy bibliotekę do obsługi wyświetlacza. Tymczasem za miesiąc zajmiemy się… sterowaniem jasności diod LED (jak się pewnie domyślacie za pomocą modulacji PWM) i napiszemy nasz pierwszy moduł działający na magistrali *Avalon-MM*. Czekajcie cierpliwie – dopiero teraz zacznie się robić ciekawie, gdyż wykorzystamy po pra pierwszy w pełni możliwości, jakie daje nam układ FPGA!

Piotr Rzeszut, AGH

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl