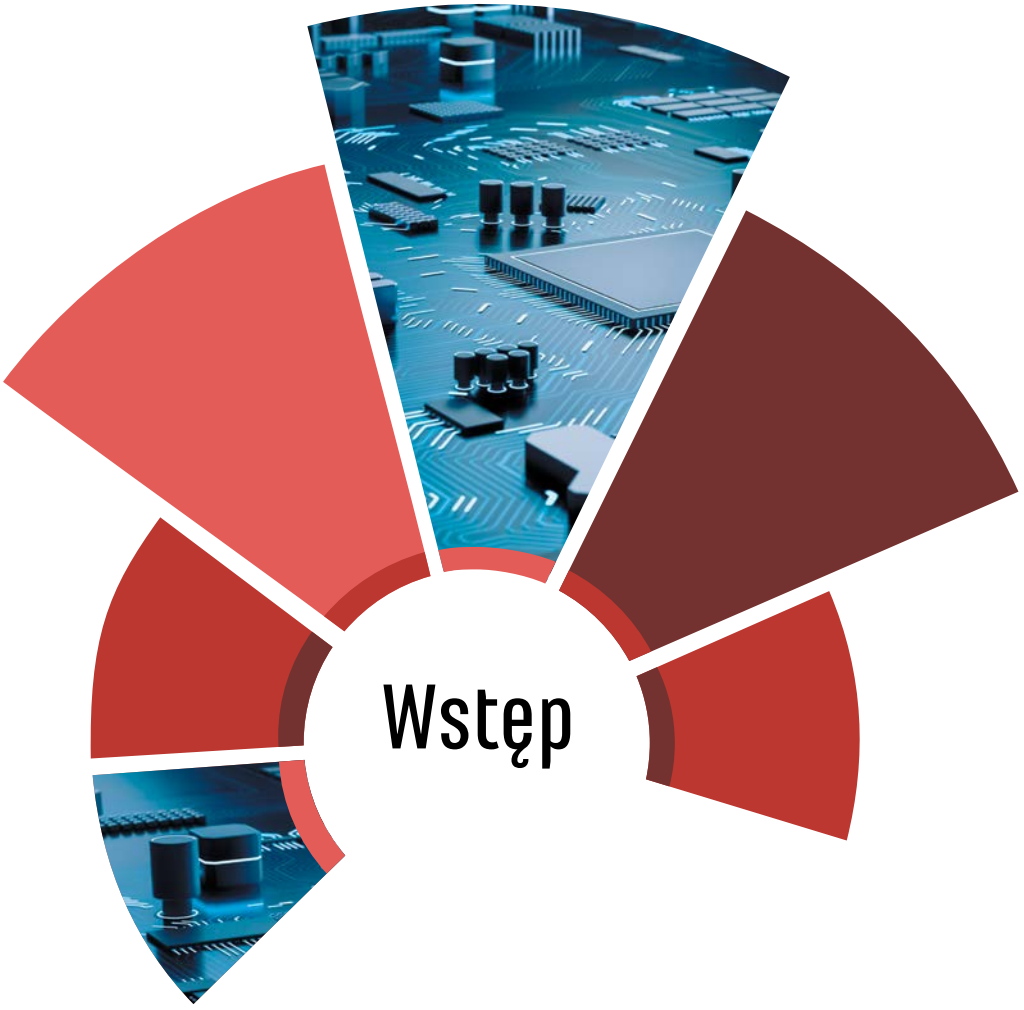

Wstęp

FPGA (ang. *field-programmable gate array*) są układami konfigurowalnymi, w których zrealizować można praktycznie dowolny obwód elektroniki cyfrowej – od prostej bramki NAND aż po układ mogący transmitować obraz za pomocą interfejsu HDMI. Możliwości są ograniczone jedynie zasobami dostępnymi w układzie, jego parametrami dynamicznymi oraz wyobraźnią projektanta. Układy FPGA wykorzystywane są wszędzie tam, gdzie zasoby standardowych mikrokontrolerów nie są wystarczające, a z drugiej strony projektowanie i produkcja układów scalonych (ASIC – ang. *application specific integrated circuit*) jest nieopłacalna ekonomicznie lub konieczna może być zmiana sposobu działania takiego podzespołu.

Niektóre układy FPGA (nazywane układami SoC – ang. *system on chip*) mają w swojej strukturze rdzenie mikroprocesorowe (najczęściej ARM) – wszystko dlatego, że niektóre funkcje jest prościej zrealizować w układzie FPGA (np. wspomniany interfejs HDMI, interfejs do kamery czy też wspomaganie specyficznych obliczeń), zaś inne stanowią idealne zadanie dla procesora (np. obsługa menu użytkownika). To właśnie z tego powodu w złożonych projektach wykorzystanie takiego połączenia pozwala osiągnąć optymalne efekty – lepiej wykonać połączenia między procesorem a specjalizowanym układem wewnątrz jednego „scalaka", niż umieszczać na płytce kolejny komponent i precyzyjnie projektować wysokoczęstotliwościowe połączenia do niego.

Nie zawsze jednak konieczne jest stosowanie potężnych układów SoC z wysokowydajnymi rdzeniami – często zadowoli nas wydajność średniej klasy mikrokontrolera i możliwość zastosowania tańszego układu. W takich przypadkach rozwiązaniem może być tzw. *soft processor* – skoro w układzie FPGA możemy umieścić dowolny układ logiczny – możemy w ten sposób umieścić tam także procesor. Takie podejście jest szeroko stosowane (np. w niektórych odbiornikach SDR, oscyloskopach) i dostępnych jest wiele rozwiązań – począwszy od w pełni płatnych (można w ten sposób wykorzystać m.in. rdzeń ARM Cortex-M1), a skończywszy na rozwiązaniach dostępnych na zasadach *open source* (jak np. implementacje procesorów RISC-V). Co więcej, wszyscy znaczący gracze na rynku układów FPGA udostępniają użytkownikom swoje rozwiązania, wraz z całym ekosystemem ułatwiającym pracę nad projektem. Właśnie takiemu rozwiązaniu, systemowi Nios II dostarczonemu przez Intel FPGA, poświęcona jest niniejsza książka.

O czym (nie) jest ta książka?

Książka ta zawiera zbiór ćwiczeń prowadzących krok po kroku przez budowę systemu mikroprocesorowego opartego na rdzeniu Nios II e – począwszy od najprostszej działającej konfiguracji, a skończywszy na projektowaniu własnych modułów. Każda część wzbogacona jest dodatkowo o komentarze, wyjaśniające kluczowe zagadnienia związane z omawianym tematem (np. interfejsem, problemem do rozwiązania). Ponadto przygotowano także ćwiczenia pokazujące, w jaki sposób za pomocą systemu Nios II obsługiwać powszechnie wykorzystywane komponenty, jak np. karty pamięci czy enkodery.

Z drugiej strony książka ta nie zastąpi kompletnego kursu programowania w języku C ani projektowania układów w języku opisu sprzętu VHDL. Przy założeniu, że Czytelnik sięga po tę pozycję uzbrojony w podstawową wiedzę na temat programowania mikrokontrolerów (np. AVR czy STM32), wydanie to zostało wzbogacone o dodatkowy rozdział autorstwa Jakuba Tyburskiego, przedstawiający najważniejsze zagadnienia związane z językiem VHDL, wykorzystane do projektowania dedykowanych modułów.

Oprogramowanie i zestaw sprzętowy

Do nauki konieczny jest rzecz jasna odpowiedni zestaw sprzętowy, umożliwiający praktyczną realizację wszystkich wskazówek. Z racji współpracy z firmą Kamami podczas przygotowania tej pozycji wykorzystany został zestaw maXimator z układem Intel FPGA z serii MAX10 (10M08DAF256C8G) i niektóre wskazówki odwołują się bezpośrednio do możliwości tej płytki edukacyjnej. Nie oznacza to jednak, że Czytelnik nie może użyć innego stosownego układu produkcji Intel FPGA – wszystkie zaprezentowane wskazówki można, czasem jedynie z drobnymi modyfikacjami (jak np. inne piny układu podłączone z jakimś interfejsem), zastosować do innego zestawu wyposażonego w FPGA z serii MAX10. Co więcej, można użyć też układów z innych serii Intel FPGA, mając jednak na uwadze w każdym przypadku różnice ich parametrów, szczególnie jeśli są to różnice „w dół" (np. mniejsza ilość wbudowanej pamięci RAM, mniejsza liczba elementów logicznych, brak wbudowanego przetwornika ADC) – w drugą stronę, wybierając „mocniejszą płytkę", nie ryzykujemy praktycznie nic. Schematy płytki maXimator oraz wszelkich dodatków firmy Kamami są dostępne *on-line*, więc każdy może sprawdzić sposób połączeń i ewentualne układy peryferyjne zdobyć we własnym zakresie. Największy problem stanowi dodanie złącza HDMI do zestawu, który go nie ma – niestety przesłanie sygnałów o częstotliwości rzędu 250 MHz może okazać się niemożliwe z wykorzystaniem prostych kabelków połączeniowych.

Tak czy inaczej jednak, jesteśmy nastawieni tylko i wyłącznie na układy FPGA firmy Intel, więc środowiskiem, którego użyjemy, będzie darmowy *Quartus Prime Lite Edition*, który możemy pobrać po bezpłatnej rejestracji na stronie producenta. W chwili pisania tego tekstu najnowszą dostępną wersją była wersja 18.1 i trzeba liczyć się z tym, że w nowszych wersjach mogą wystąpić mniejsze lub większe różnice. Pobierając oprogramowanie, musimy zadbać, aby koniecznie zaopatrzyć się w pełną wersję instalacyjną, a na pewno poniższe komponenty:

1. *Quartus Prime Lite Edition*,
2. *ModelSim-Intel FPGA Edition*,
3. *MAX 10 FPGA device suport* (lub jeśli korzystamy z innej rodziny układów – wybieramy właściwe pliki *device suport*).

Instalację rozpoczynamy, uruchamiając plik **.bat* w folderze z pobranymi wersjami instalacyjnymi, w jej czasie instalujemy także sterowniki dla *USB Blaster*.

Zestaw maXimator, tak jak każdy inny zestaw, ma mocne i słabe strony. Poniżej przedstawiam w tabeli elementy składowe oraz cechy zestawu, wraz ze swoimi subiektywnymi opiniami.

Komponent	Komentarz autora
maXimator	
Układ FPGA 10M08DAF256C8G	+ Rozsądny układ dla początkujących + Wbudowana pamięć konfiguracyjna + 2 pętle PLL (każda z 5 wyjściami zegarowymi) + Przetwornik ADC 12-bitowy z dostępnymi w zestawie 6 wejściami z zewnątrz +/– Ma średnią ilość pamięci RAM, która wystarczy do wielu zastosowań
Złącze HDMI	+ Niedostępne na wielu płytkach w tej klasie cenowej + Można uzyskać nawet 24-bitową głębię kolorów
Złącze VGA	+/– 3-bitowa głębia kolorów wystarcza do większości eksperymentów
Złącze karty micro SD (SPI)	+ Umożliwia wykorzystanie karty do przechowywania danych i ich wymiany z komputerem
Wejścia/wyjścia na złączach zgodnych z Arduino układy zapewniające zgodność ze standardem napięciowym 5 V	+ Możliwość podłączenia wielu modułów zgodnych z Arduino bez ich modyfikacji + Dodatkowe zabezpieczenie układu FPGA przed uszkodzeniem w przypadku błędnego połączenia +/– Liczba dostępnych pinów może nie wystarczyć do bardziej zaawansowanych eksperymentów – Zastosowane bufory mogą w pewnych specyficznych sytuacjach (duże obciążenie pojemnościowe) stwarzać problemy
Potencjometr	+ Możliwość zadawania wartości analogowej bez dodatkowych komponentów +/– Zajęty dedykowany pin wejścia analogowego
Punkt testowy GND	+ Możliwość pewnego podłączenia sondy oscyloskopowej
4 diody LED	
Przycisk	Można wykorzystać w dowolnym celu
Złącze dla modułu USB-UART	Kompatybilne z modułem ZL5USB
Złącze JTAG	
KAMAMI USB Blaster	
	+ Programator w pełni zgodny z oryginałem + Zatwierdzony przez Intel FPGA +/– Brak obudowy
maXimator expander	
Wyświetlacz 4 × 7-seg	+ Podstawowe peryferia służące do nauki różnych aspektów pracy z układami FPGA
3 przyciski	
Termometr analogowy	
2 diody WS2812	

10M08DAF256C8G czy 10M08DAF256C8GES

Niektóre wersje zestawu maXimator zostały wytworzone z użyciem, dostępnych we wcześniejszej fazie produkcji rzeczonego układu FPGA, egzemplarzy przeznaczonych m.in. do testów w aplikacjach użytkowników. Zostały one oznaczone końcówką ES. Układy te są praktycznie identyczne i nie powinniśmy zauważyć żadnych różnic, niezależnie do tego, którą wersję mamy na płytce. Tworząc projekty, dla porządku, wybierajmy taką wersję układu, jak zamontowana w naszym zestawie, ja jednak będę posługiwał się już w dalszej części tylko wersją docelową, bez końcówki ES.

Co jeszcze będzie nam potrzebne?

Oprócz zestawu z układem FPGA firmy Intel i ewentualnych komponentów zastępujących funkcjonalności zestawu maXimator i rozszerzenia maXimator expander, jeśli oczywiście chcemy zrealizować wszystkie zadania, potrzebować będziemy komponentów wymienionych w tabeli poniżej. Komponenty oznaczone * są opcjonalne i nie stanowią głównego elementu żadnego z wykonywanych ćwiczeń.

Komponent	Komentarz
Konwerter USB-UART	Moduł ZL5USB lub inny konwerter podający sygnały interfejsu UART w standardzie 3,3 V albo 5 V (konieczne wtedy jest podpięcie konwertera poprzez złącza Arduino).
Układ BMP280	Czujnik ciśnienia z przełączanymi interfejsami I^2C i SPI używany w czasie poznawania tych interfejsów.
Układy MCP23008 oraz MCP23S08	Ekspandery I/O z interfejsami I^2C i SPI. Układy te oraz układ BMP280 zastąpić możemy oczywiście innymi elementami, na których możliwe będzie przetestowanie tych dwóch interfejsów (choćby różne modele zegarów RTC).
Wyświetlacz zgodny z HD44780 oraz klawiatura rezystancyjna	Połączenia możemy wykonać na płytce stykowej lub używając modułu dla Arduino.
Mechaniczny enkoder inkrementalny	Posłuży do demonstracji możliwości obsługi takich enkoderów sprzętowo.
Moduł z terminalami połączeniowymi*	Tak zwany „Screw Shield", umożliwiający połączenie dodatkowych przewodów przy równoczesnym podłączeniu klasycznego modułu Arduino.
Moduł z diodami WS2812*	Dodatkowy moduł z większą liczbą diod WS2812 – „Pixel Shield".
Karta pamięci micro SD	Dowolna karta, na której nie przechowujemy aktualnie żadnych danych, powinna spełnić swoje zadanie.
Monitor komputerowy	Z wejściami VGA oraz HDMI/DVI.
Kabel VGA	
Kabel HDMI	Jeśli posiadany monitor z wejściem cyfrowym innym niż HDMI, musimy mieć stosowny kabel-przejściówkę.

Wiedza, jaką trzeba posiadać

Jak już wcześniej wspomniano, książka niniejsza nie jest kompendium pozwalającym na dogłębne poznanie wszystkich aspektów omawianego tematu. Zakładam, że Czytelnik posiada podstawową wiedzę z zakresu programowania w języku C (lub pokrewnych), a jeszcze lepiej, jeśli wiedza ta dotyczy programowania mikrokontrolerów. Wiedza z zakresu programowania układów FPGA czy języków opisu sprzętu (w szczególności VHDL) będzie pomocna, jednak nie jest wymagana i może zostać uzupełniona w trakcie lektury dodatku – w końcu okaże się, że układy FPGA wcale nie są tak skomplikowane, jak niektórzy twierdzą.

Materiały dodatkowe

Książce tej towarzyszą materiały dodatkowe, bez których realizacja niektórych zadań jest niemożliwa. Są one dostępne na stronie *www.wydawnictwo.btc.pl*.

Materiały te zorganizowane są w folderach, których nazwy rozpoczynają się od numeru rozdziału, którego dotyczą. W tych folderach znajdują się (niektóre pozycje mogą występować w zależności od tematu):

1. Folder *Projekt* zawierający finalną wersję projektu, jaką powinniśmy osiągnąć pod koniec danego rozdziału:
   a) projekt oprogramowania *Quartus*,
   a) */Nios* – projekt systemu mikroprocesorowego (*Platform Designer*),
   a) */Nios/software* – oprogramowanie oraz BSP (przygotowane w *Eclipse*),
   a) */Nios/software/TutorialXX* – kody źródłowe oprogramowania oraz biblioteki.
2. Plik *XX_Projekt_start.zip* zawierający wersję projektu, od której powinniśmy rozpocząć pracę w czasie lektury danego rozdziału (na wszelki wypadek, gdyby ktoś miał problemy i chciał zacząć „na świeżo").
3. Pliki *main_XX_....c* zawierające różne wersje pliku *main.c* oprogramowania procesora *Nios*, podsumowujące kluczowe fragmenty pracy nad danym zagadnieniem.
4. Dodatkowe materiały, jak na przykład noty katalogowe/dokumentacje/schematy stosowanych układów/modułów czy oprogramowanie wykorzystywane w czasie kursu (np. do generowania czcionek).

Kto jest kim?

O ile w przypadku programowania mikrokontrolerów kwestie używanego oprogramowania i rodzajów plików są niezwykle jasne, o tyle w czasie spotkań z systemem Nios II będziemy pracowali na wielu poziomach i w wielu programach. Aby nie pogubić się w gąszczu nowych, postanowiłem w **tabeli 0.1** zebrać wszystkie używane przez nas składowe oprogramowania, zaś w **tabeli 0.2** ważniejsze formaty plików, z jakimi mieć będziemy styczność. Na pewno wyraźnego zaznaczenia wymaga fakt, że pracować będziemy na dwóch głównych płaszczyznach. Po pierwsze, będziemy się zajmować projektem systemu mikroprocesorowego, który będzie potem przetłumaczony na

konfigurację układu FPGA – czyli w uproszczeniu odpowiednie połączenie bramek, przerzutników itp. Po drugie, dla takiego procesora będziemy przygotowywali oprogramowanie – podobnie jak na każdy inny mikrokontroler.

Tab. 0.1. Spis ważniejszych funkcji używanego oprogramowania

Nazwa programu lub okna/modułu	Zastosowanie
Quartus	Jego podstawowym zastosowaniem jest praca z plikami tworzonymi w językach opisu sprzętu i tłumaczenie ich na konkretną konfigurację (bramek, przerzutników, …) w układzie FPGA – czyli dokonywanie syntezy. Tu także tworzymy główny projekt systemu.
Platform Designer	Służy do tworzenia systemów mikroprocesorowych, np. w oparciu o rdzeń Nios II. W tym programie będziemy mogli do wybranego rdzenia podpinać różne układy peryferyjne (i je konfigurować), korzystając z interfejsu graficznego. Następnie na podstawie tego projektu zostaną wygenerowane pliki dla głównego programu Quartus (na podstawie których przygotowana zostanie potem konfiguracja układu FPGA) oraz pliki zawierające definicje podłączonych komponentów (układów peryferyjnych, np. URAT czy GPIO) niezbędne do tworzenia oprogramowania.
Pin Planner	Narzędzie pakietu Quartus służące do przypisywania fizycznych wyprowadzeń układu FPGA do konkretnych portów w projekcie.
ModelSim	Oprogramowanie do symulacji działania projektu w układzie FPGA.
Eclipse	Oprogramowanie, w którym będziemy pisać programy uruchamiane potem w zbudowanym przez nas systemie mikroprocesorowym. Pozwala także na debugowanie programu.
Signal Tap	Moduł wewnętrznego analizatora stanów logicznych, który pozwala na podglądanie pracy układu FPGA na żywo.

Tab. 0.2. Spis ważniejszych formatów plików

Rozszerzenie pliku	Zastosowanie
.qpf	Plik projektu programu Quartus.
.sof	Plik konfiguracyjny układu FPGA umożliwiający konfigurację nietrwałą (generowany przez Quartus po syntezie).
.pof	Plik konfiguracyjny układu FPGA umożliwiający konfigurację trwałą przez zapis do pamięci FLASH (generowany przez Quartus po syntezie).
.vhd / *.v*	Plik źródłowy w języku opisu sprzętu VHDL/Verilog (tworzone przez użytkownika lub częściowo generowane przez Platform Designer lub Model Sim – jako *testbench*).
.qsys	Plik projektu programu Platform Designer, przechowuje skład naszego systemu mikroprocesorowego.
.sopcinfo	Plik związany z projektem Platform Designer służący do wygenerowania BSP (zestawu driverów oraz opisu rejestrów naszego systemu mikroprocesorowego) – wykorzystywany przez Eclipse, generowany przez Platform Designer.
.c / *.h*	Pliki źródłowe/nagłówkowe tworzące oprogramowanie, używamy ich w Eclipse. Tworzone przez użytkownika lub generowane przez Eclipse.
.elf	Plik wynikowy po kompilacji oprogramowania w Eclipse używany przez nas zwykle podczas procesu debugowania.
.hex	Plik wynikowy po kompilacji oprogramowania w Eclipse używany przez nas zwykle jako plik inicjalizacyjny dla pamięci RAM.
.mif	Inny format pliku inicjalizującego pamięć RAM.
.tcl (*…_hw.tcl*)	Pliki skryptowe języka TCL. W naszym przypadku używane są one do przechowywania definicji tworzonych przez nas komponentów dla Platform Designer i są przez niego generowane w czasie tworzenia nowego komponentu.

Podziękowania

Pragnąłbym w szczególny sposób podziękować osobom, które przyczyniły się do powstania tej książki:

Rodzicom, za wspieranie w rozwijaniu pasji i wskazywanie drogi w życiu.

Panu Piotrowi Zbysińskiemu – za pomoc przy jej wydaniu oraz motywację do działania.

Wykładowcom z AGH w Krakowie, Panom: Ernestowi Jamro, Jerzemu Kasperkowi i Pawłowi Rajdzie, za cenne wskazówki w czasie zgłębiania tajników FPGA.

Panom: Łukaszowi Krzakowi oraz Piotrowi Chodorowskiemu, za zgodę na wykorzystanie kodów przez nich opracowanych.

Panu Jakubowi Tyburskiemu za cenne konsultacje oraz przygotowanie rozdziału o języku VHDL.

Panu Dominikowi Bieczyńskiemu, po rozmowach z którym w 2013 roku stałem się posiadaczem swojego pierwszego zestawu edukacyjnego z układem FPGA.

Wszystkim bezpośrednio związanym z przygotowaniem i wydaniem niniejszej książki, bez których moje teksty nie zyskałyby tak czytelnej formy.

Wielu innym osobom, których nie sposób wymienić, a które pomagały mi i uczyły wielu rzeczy, bez których nie mógłbym osiągnąć tego miejsca w życiu.

Lista poruszanych zagadnień praktycznych

Rozdział 1 – Pierwszy mikroprocesor w układzie FPGA

- Tworzenie projektu w *Quartus*
- Budowa podstawowego systemu z rdzeniem Nios II w *Platform Designer*
- Ustawienia pinów w *Pin Planner*
- Kompilacja projektu
- Programowanie układu FPGA
- Generowanie projektu oprogramowania w *Ecplipse*
- Wgrywanie oprogramowania do procesora
- Używanie konsoli JTAG UART

Rozdział 2 – Wyjście na świat, czyli rzecz o GPIO

- Dodawanie portów wejścia/wyjścia do systemu
- Obsługa programowa portów wejścia/wyjścia
- Integrowanie oprogramowania z plikiem konfiguracyjnym układu FPGA
- Używanie nieulotnej pamięci konfiguracyjnej w układzie FPGA

Rozdział 3 – Timery i przerwania, część 1

- Obsługa wyświetlaczy 7-segmentowych z multipleksowaniem
- Dodawanie do systemu timera
- Przerwania
- Wykorzystanie timera i przerwań do obsługi wyświetlacza
- Odmierzanie czasu za pomocą timera

Rozdział 4 – Timery i przerwania, część 2

- Przerwania zewnętrzne z portów wejścia/wyjścia
- Eliminacja drgań styków z wykorzystaniem timera
- Precyzyjny pomiar czasu z zastosowaniem timera
- Dostęp atomowy i wyłączanie przerwań

Rozdział 5 – UART

- Dodanie modułu UART do systemu
- Obsługa UART z wykorzystaniem standardowego wejścia/wyjścia
- Obsługa UART z użyciem przerwań i buforów kołowych

Rozdział 6 – Interfejs SPI

- Dodanie modułu SPI do systemu
- Obsługa przykładowych układów z interfejsem SPI

Rozdział 7 – Interfejs I^2C

- Dodanie modułu I^2C do systemu
- Podstawy tworzenia własnych komponentów *Platform Designer*
- Obsługa przykładowych układów za pomocą interfejsu I^2C

Rozdział 8 – Przetwornik analogowo-cyfrowy

- Dodanie przetwornika ADC do systemu
- Obsługa klawiatury rezystancyjnej
- Obsługa wyświetlacza HD44780

Rozdział 9 – Własne moduły *Avalon-MM* – generator PWM

- Magistrala *Avalon-MM*
- Tworzenie własnego modułu
- Generowanie sygnału PWM
- Tworzenie modułów z parametrami

Rozdział 10 – Własne moduły *Avalon-MM* – enkoder i wyświetlacz 7-segmentowy

- Sprzętowa obsługa wyświetlaczy 7-segmentowych
- Sprzętowa eliminacja drgań styków
- Sprzętowa obsługa enkoderów inkrementalnych
- Tworzenie modułów *Platform Designer* składających się z wielu plików źródłowych
- Konfiguracja rezystorów podciągających w układzie FPGA

Rozdział 11 – Własne moduły *Avalon-MM* – diody WS2812, czyli czas na kolor

- Podstawowy moduł sterujący diodami WS2812
- Tworzenie portu *Master* dla magistrali *Avalon-MM*
- Moduł sterujący diodami WS2812 z zewnętrzną, 2-portową pamięcią RAM

Rozdział 12 – Więcej pamięci, czyli karty SD

- Dodanie modułów do obsługi karty SD

- Przystosowanie biblioteki FatFs na podstawie przykładów dla procesorów AVR
- Metody porządkowania i współdzielenia własnych modułów

Rozdział 13 – VGA, czyli czas na duże wyświetlacze

- Projektowanie zaawansowanego modułu generatora znaków z wyjściem VGA
- Projektowanie własnych czcionek

Rozdział 14 – HDMI, czyli duży „cyfrowy" wyświetlacz

- Transmisja różnicowa
- Moduł generatora znaków z wyjściem HDMI

Rozdział 15 – Kilka drobiazgów, które nigdzie nie pasują, czyli… uzupełnienie

- Symulacja systemu Nios II z oprogramowaniem
- Symulacja pojedynczych komponentów systemu
- Używanie systemu Nios II jako komponentu w większym projekcie
- Wykorzystanie analizatora stanów logicznych *SignalTap*.

Rozdział 16 – Krótki kurs VHDL