Przegląd otwartych narzędzi dla układów FPGA

Otwarte narzędzia, służące do tworzenia oprogramowania dla mikrokontrolerów jednoukładowych, są powszechnie stosowane w projektach komercyjnych od dłuższego czasu. Obecnie jakość kodu generowanego przez kompilatory dla architektury ARM jest na tyle duża, że producentom środowisk programistycznych nie opłaca się tworzyć autorskich rozwiązań. Coraz powszechniejszą praktyką wśród producentów narzędzi komercyjnych jest tworzenie środowiska IDE wraz z odpowiednimi kreatorami, które wewnętrznie wykorzystują kompilator GCC, pozwalając, za pomocą kilku ruchów myszką, wygenerować szkielet projektu dla wybranego mikrokontrolera. Zupełnie inaczej wygląda sytuacja w przypadku narzędzi, służących do projektowania urządzeń, opartych na układach FPGA.

W przypadku układów FPGA do niedawna jedynym rozwiązaniem było skorzystanie z komercyjnych narzędzi, dostarczanych przez producentów. Narzędzia tego typu, w podstawowej wersji, są zazwyczaj darmowe, jednak nie mamy możliwości wglądu w kod źródłowy, co może być problemem dla niektórych rodzajów projektów. Przygotowanie otwartych narzędzi dla układów FPGA jest zadaniem zdecydowanie bardziej skomplikowanym w porównaniu do kompilatorów, gdzie dostępna jest specyfikacja instrukcji maszynowych procesora oraz ich kody maszynowe. Dzięki dokumentacji ISA niezależni programiści mogą bez większych kłopotów opracować odpowiednie narzędzia jak, kompilator asembler, itd., na bazie dokumentacji dostarczanej przez producenta. W przypadku układów FPGA dokumentacja dla pliku binarnego, służącego do konfiguracji FPGA oraz szczegóły architektoniczne są najczęściej pilnie strzeżoną tajemnicą. Dlatego opracowanie otwartych narzędzi wymaga wcześniejszego zastosowania inżynierii wstecznej, co jest zadaniem czasochłonnym. Należy wziąć pod uwagę, że nie wszystkie zawiłości uda się odkryć tym sposobem, a zatem finalne rezultaty mogą znacząco odbiegać od komercyjnych rozwiązań. Z drugiej strony, producenci oprogramowania, z uwagi na nakład czasu i środków poświęconych na rozwój narzędzi oraz przewagę konkurencyjną, nie mogą sobie pozwolić na publikację kodów źródłowych. Niemniej sytuacja nie jest tak beznadziejna, jak mogłoby się wydawać, ponieważ niektóre typy układów FPGA, wykorzystaniu podstawowej dokumentacji dostarczanej przez producenta oraz inżynierii wstecznej, zostały bardzo dobrze rozpracowane. Miłośnicy otwartych rozwiązań mogą się pokusić o stworzenie własnego sprzętu w pełni wolnego od komercyjnych rozwiązań. Obecnie, z otwartymi narzędziami najlepiej współpracują układy rodziny Lattice ICE40 oraz ECP5. Niektóre układy firmy XILINX udoskonalono pod tym względem, ale do poprawnej pracy nadal konieczne jest użycie narzędzia **bitgen** pochodzącego z pakietu ISE, więc nie możemy powiedzieć, że na tę chwilę jest to rozwiązanie całkowicie otwarte.

Przegląd narzędzi open source

Zanim przejdziemy do przeglądu dostępnych narzędzi, spójrzmy, jak wygląda synteza projektu w układzie FPGA (**tabela 1**).

Proces syntezy wygląda podobnie dla większości układów i rozpoczyna się od analizy oraz syntezy pliku źródłowego (najczęściej Verilog, VHDL), w wyniku czego powstaje ogólna lista połączeń, czyli schemat wewnętrzny układu. Lista jest generyczna i na tym etapie nie zawiera cech specyficznych dla topologii konkretnego układu FPGA. W kolejnym kroku „Place and Route" następuje proces rozmieszczenia oraz optymalizacji elementów, z uwzględnieniem topologii i architektury wybranego układu tak, aby długość połączeń wewnętrznych była jak najkrótsza, a liczba użytych elementów LUT jak najmniejsza. Efektem jest lista połączeń, specyficzna dla danego układu, która w kroku asemblacji zamieniana jest na plik *bitstream*, opisujący sposób połączenia bloków wewnętrznych układu. Plik ten jest dalej wykorzystywany bezpośrednio do konfiguracji FPGA lub do zaprogramowania pamięci konfiguracyjnej.

Odrębnym etapem jest analiza projektu, na który składa się analiza czasowa oraz symulacja logiczno-behawioralna, wykonywana bezpośrednio na komputerze PC, pozwalająca sprawdzić poprawność logiczną utworzonego projektu. W przypadku symulacji i wizualizacji, podobnie jak poprzednio, pliki HDL są syntezowane do ogólnej listy połączeń, zmienianej na model behawioralny, który może być symulowany bezpośrednio na komputerze PC. Efektem symulacji jest plik, zawierający przebiegi czasowe działającego układu, który możemy poddać analizie pod kątem poprawności.

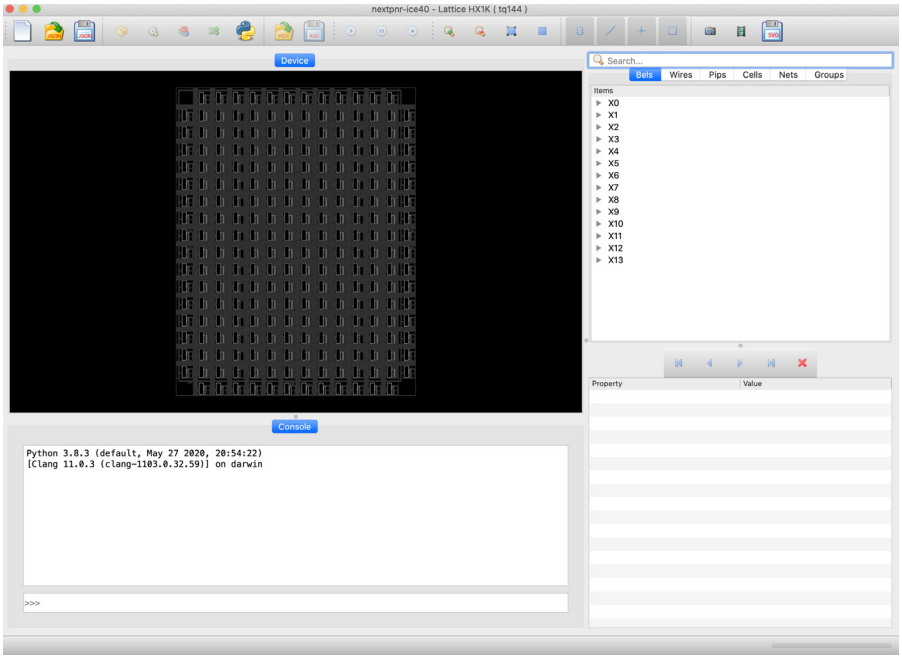
Jeśli chodzi o języki HDL, wspierane przez narzędzia open source, to obecnie wsparcie skupione jest na języku Verilog. VHDL wspierany jest w mniejszym stopniu, głównie w charakterze eksperymentalnym.

Przegląd rozpoczynamy od narzędzi związanych z symulacją oraz wizualizacją projektów, co jest zadaniem prostszym, ponieważ nie

Tabela 1. Synteza projektu w układzie FPGA

Analiza i synteza	HDL → NetLista.
Place and Route	Netlista → Netlista specyficzna dla układu.
Asembler	Netlista specyficzna dla układu → bitstream.
Analiza czasowa	Analiza zależności czasowych. Sprawdzenie założeń czasowych.
Symulacja i wizualizacja	Symulacja i wizualizacja oraz analiza układu na komputerze PC.

wymaga specyficznych danych na temat architektury wybranego układu. Tutaj najbardziej kompleksowym narzędziem jest „*Icarus Verilog*" (*http://iverilog.icarus.com/home*), dostępny na licencji GPL. Pakiet ten wspiera Verilog w standardzie 1995/2001/2005 oraz częściowo SystemVerilog. Kompilator zapewnia bardzo dokładną analizę składni i z reguły wykrywa dużo więcej problemów niż narzędzia komercyjne, a zatem świetnie nadaje się na etapie projektowania oraz symulacji. W wyniku działania narzędzia powstaje model behawioralny, który jest kompilowany do kodu pośredniego VVP. Kod pośredni może być uruchamiany za pomocą interpretera VVP, w efekcie czego powstaje plik wynikowy z przebiegami wyjściowymi. Tak powstały plik może być przeglądany za pomocą narzędzia GTKWave, które jest szeroko wykorzystywane przez wszelkiego rodzaju narzędzia symulacyjne. Symulacja działa niezbyt szybko, co jest główną wadą programu, ponieważ wykorzystywany jest kod pośredni, który musi być interpretowany przez maszynę wirtualną VVP.



Rysunek 1. Edytor graficzny NextPNR

Innym przydatnym narzędziem podczas symulacji i testowania może być projekt „*Verilator*" (*https://github.com/verilator/verilator*), dostępny na licencji LGPL, który również może być używany do symulacji układu na maszynie docelowej. W przeciwieństwie do poprzednio omówionego narzędzia, zamienia on kod Verilog na kod w języku C++, który następnie może być skompilowany z wykorzystaniem dowolnego kompilatora. Dzięki takiemu rozwiązaniu symulacja przebiega dużo szybciej, z uwagi na natywne wykonywanie kodu maszynowego przez procesor. Dodatkowo, do wygenerowanego kodu mogą być dodawane własne elementy programu w języku C++, dzięki czemu układ, który chcemy symulować, możemy połączyć z innym rzeczywistym sprzętem, np. wykorzystując porty GPIO płytki z Linuxem. Innym przykładowym zastosowaniem, w przypadku gdy mamy zaimplementowany jakiś procesor dla FPGA (np. RISCV), jest możliwość uruchamiania kodu dla tego procesora w trybie symulacji. Rozwiązanie to nie jest pozbawione wad, a największy problem to brak możliwości pisania bezpośrednio kodu testowego „test bench", ze względu na nieobsługiwane komendy, które nie są syntezowane do bramek logicznych. Dlatego w klasycznych przypadkach dużo lepszym rozwiązaniem będzie wykorzystanie oprogramowania „*Icarus Verilog*", który jest typowym narzędziem symulacyjnym.

Jeśli ktoś jest zainteresowany językiem VHDL, to narzędziem godnym uwagi jest GHDL (*http://ghdl.free.fr*), który podobnie jak *Verilator* zamienia VHDL bezpośrednio na kod maszynowy, wykorzystując wewnętrzne wywołania kompilatora GCC lub LLVM, do wygenerowania kodu maszynowego, który może być uruchomiony na komputerze. Wynikiem działania programu jest plik wynikowy z przebiegami czasowymi symulacji gHDL, który następnie może być analizowany.

Przejdźmy teraz do narzędzi związanych z syntezą układu w rzeczywistych układach FPGA. Tutaj wybór jest zdecydowanie mniejszy. Do dyspozycji mamy w zasadzie trzy narzędzia:

- Yosys: *http://www.clifford.at/yosys/about.html*
- Odin II: *https://github.com/verilog-to-routing/vtr-verilog-to-routing/tree/master/ODIN_II*
- Berkley ABC: *https://github.com/berkeley-abc/abc*

Dwa ostatnie narzędzia wykorzystywane są głównie w zastosowaniach akademickich oraz do badania różnych algorytmów „Place and Route", bez konkretnej implementacji dla układów FPGA. W związku z powyższym jedynym wyborem pozostaje **Yosys**, który zapewnia wsparcie dla układów **Xilinx7** oraz **Lattice iCE40**. Jest to stosunkowo nowe narzędzie, napisane w nowoczesny sposób, z wykorzystaniem języka C++. Jego główna zaleta to uywanie z bibliotek dedykowanych bloków dla układów FPGA, np. jeżeli w konkretnym układzie będziemy mieli zintegrowany komponent **BLOCK RAM**, **yosys** umożliwi nam skorzystanie z niego. Głównym ograniczeniem narzędzia jest to, że obecnie wspierany jest jedynie Verilog, ale w przyszłości planowane jest dodanie wsparcia dla VHDL.

Przejdźmy teraz do narzędzi związanych z rozmieszczaniem bloków oraz trasowaniem połączeń, umożliwiających przejście od ogólnej listy połączeń do netlisty, specyficznej dla danego układu. Na działanie tego typu narzędzi składają się następujące kroki: zgrupowanie elementów w większe bloki; rozmieszczenie poszczególnych bloków w układzie FPGA tak, aby znajdowały się jak najbliżej siebie; połączenie bloków jak najkrótszymi ścieżkami tak, aby osiągnąć jak największą częstotliwość pracy układu. Programy realizujące takie zadania to:

- **VPR**: (Versatile Placement and Routing) *https://www.eecg.utoronto.ca/~vaughn/vpr/vpr.html*
- **ArachnePNR**: *https://github.com/YosysHQ/arachne-pnr*
- **NextPNR**: *https://github.com/YosysHQ/nextpnr*

VPR jest wykorzystywany głównie w środowisku naukowym, do badania algorytmów „Place and Route" i nie znajduje zastosowań praktycznych. Kolejne narzędzie **ArachnePNR** jest narzędziem specyficznym dla układów *Lattice iCE40*. Obecnie, wsparcie dla niego zostało zarzucone, choć po dzień dzisiejszy jest powszechnie wykorzystywany w niektórych projektach. Wadą jest również to, że nie działa optymalnie.

Współcześnie polecanym narzędziem jest program **NextPNR**, który jest narzędziem niezależnym od układu. Aktualnie wspiera rodziny układów firmy Lattice: *iCE40* oraz *ECP5*. Trwają prace nad wsparciem dla układów firmy Xilinx. Dzięki modułowej budowie w łatwy sposób może być rozszerzany o wsparcie dla kolejnych rodzin układów. **NextPNR** zawiera również edytor graficzny, umożliwiając ręczne rozmieszczenie i trasowanie najbardziej krytycznych czasowo elementów (**rysunek 1**).

Ostatnim elementem potrzebnym do przygotowania projektu dla FPGA jest proces asemblacji, przekształcający listę komponentów specyficzną dla danego układu na *bitstream*, umożliwiający konfigurację układu docelowego. Jest to jeden z najpilniej strzeżonych przez producentów układów fragment specyfikacji. Z tej przyczyny format pliku konfiguracyjnego *bitstream* w otwartych narzędziach jest odtwarzany za pomocą inżynierii wstecznej. Do realizacji tego

zadania w zasadzie jedynym dostępnym otwartym narzędziem jest oprogramowanie **IceStorm**, współpracujące z rodziną układów iCE40.

Instalacja oprogramowania

Z uwagi na najlepsze wsparcie dla otwartych narzędzi, skupimy się na przygotowaniu oprogramowania, służącego do współpracy z układami iCE40 firmy Lattice. Są to stosunkowo nieskomplikowane oraz tanie układy, zawierające od 384 do 7680 komórek LUT, zawierają zintegrowaną pamięć **BlockRAM** o wielkości do 128 kilobitów, oraz pętlę PLL. Zawierają również zintegrowane kontrolery I²C oraz SPI, które mogą być wykorzystywane do konfiguracji układu za pomocą zewnętrznej pamięci czy mikrokontrolera. Istotną cechą z amatorskiego punktu widzenia jest to, że występują również w łatwych do montażu, w warunkach amatorskich, obudowach TQFP. Instalację środowiska opiszemy na przykładzie systemów operacyjnych Linux oraz OSX. W przypadku Windows z użyciem środowiska MINGW, proces instalacji będzie wyglądał podobnie.

Ubuntu Linux

Proces instalacji rozpoczynamy od kompilacji pakietu **IceStorm.** W tym celu należy doinstalować zależności za pomocą komendy `apt`:

```
sudo apt install build-essential clang bison
flex libreadline-dev gawk tcl-dev libffi-dev git
mercurial graphviz xdot pkg-config python python3
libftdi-dev qt5-default python3-dev libboost-all-dev
cmake libeigen3-dev
```

W kolejnym kroku należy pobrać z repozytorium git kod źródłowy pakietu **IceStorm**, skompilować go, a następnie zainstalować w systemie, co możemy zrealizować za pomocą sekwencji następujących komend:

```
git clone https://github.com/cliffordwolf/icestorm.
git icestorm
cd icestorm
make -j$(nproc)
sudo make install
```

Poprawność instalacji pakietu możemy potwierdzić, wpisując polecenie: `iceprog help`. Program zgłosi się listą dostępnych opcji.

Kolejnym narzędziem, które będziemy musieli zainstalować, to jest pakiet **nextPNR**. Do prawidłowej kompilacji potrzebna jest biblioteka QT5, którą możemy doinstalować za pomocą polecenia:

```
sudo apt install qt5-default
```

Po instalacji potrzebnych bibliotek możemy przystąpić do kompilacji oraz instalacji programu za pomocą następujących poleceń:

```
git clone https://github.com/YosysHQ/nextpnr nextpnr
cd nextpnr
cmake -DARCH=ice40 -DCMAKE_INSTALL_PREFIX=/usr/
local .
make -j$(nproc)
sudo make install
```

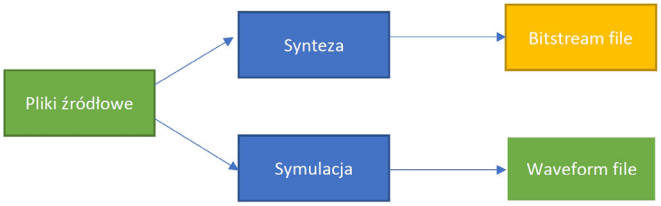
Weryfikację poprawności instalacji możemy przeprowadzić, wydając komendę: `nexpnrice40gui`, co spowoduje wyświetlenie okna programu, służącego do manualnego trasowania.

Do syntezy projektu brakuje nam jeszcze programu **Yosys.** Procedura instalacji oraz kompilacji przebiega podobnie jak w przypadku pozostałych projektów:

```
git clone https://github.com/cliffordwolf/yosys.git
yosys
cd yosys
make -j$(nproc)
sudo make install
```

Mamy już wszystkie narzędzia niezbędne do syntezy projektu w prawdziwym układzie. Potrzebować będziemy jeszcze narzędzi, służących do symulacji oraz wizualizacji – „Icarus Verilog" oraz „GTKWave". Ponieważ Ubuntu dysponuje tymi pakietami w managerze pakietów, wystarczy je doinstalować za pomocą polecenia:

```
sudo apt install iverilog gtkwave
```



Rysunek 2. Synteza oraz symulacja przykładowego projektu

Po wykonaniu powyższych czynności mamy już kompletne środowisko, które możemy wykorzystać do projektowania sprzętu, z wykorzystaniem układów FPGA rodziny iCE40.

OS-X

Instalacja oprogramowania dla systemu Mac OSX, dzięki managerowi pakietów **brew** oraz przygotowanym pakietom, jest zdecydowanie prostsza od instalacji w systemie Linux. Pierwszą czynnością, jeśli nie korzystaliśmy nigdy wcześniej z managera pakietów **brew**, jest jego instalacja według instrukcji, zamieszczonej na stronie domowej projektu: *https://brew.sh*.

Kiedy zainstalowaliśmy manager **brew,** możemy przystąpić do właściwej instalacji potrzebnych narzędzi, rozpoczynając od dodania odpowiedniego TAP-u:

```
brew tap lucckb/openfpga
```

W kolejnym kroku należy doinstalować potrzebne oprogramowanie, wpisując następujące polecenia:

```
brew install yosys
brew install icestorm
brew install nextpnr --without-arch-ecp5
brew install icarus-verilog
brew cask install gtkwave
```

Po wykonaniu powyższych czynności środowisko potrzebne do projektowania układów FPGA w systemie OSX jest gotowe do pracy.

Proces syntezy i symulacji przykładowego projektu

Spójrzmy teraz, w jaki sposób przeprowadzana jest synteza oraz symulacja przykładowego projektu, który będzie się składał z następujących plików:

- *blink_top*.v – główny moduł projektu, opisujący układ,
- *blink.v* – jeden z modułów projektu, implementujący przykładowy komponent,
- *blink_tb.v* – główny moduł, służący do symulacji tzw. *test bench*,
- *icecore_pins.pcf* – plik opisujący mapowanie modułu głównego do fizycznych wyprowadzeń układu FPGA.

Cykl projektowania możemy podzielić na dwa odrębne etapy (**rysunek 2**): symulacja, w wyniku której powstaje wynikowy plik, zawierający diagram przebiegów wyjściowych; synteza, tworząca w rezultacie plik *bitstream*, który może być wykorzystany do zaprogramowania pamięci konfigurującej układ FPGA.

Podczas symulacji wykorzystywany będzie plik *blink_tb.v*, stanowiący główny moduł symulacyjny, oraz plik *blink.v*, który stanowi jedyny moduł projektu. Plik *blink_top.v*, nie jest używany, ponieważ stanowi on moduł główny projektu, stosowany do syntezy. Do procesu symulacji wykorzystamy narzędzie „*Icarus Verilog*", wywołując je z linii komend w następujący sposób:

```
iverilog -o blink.vvp blink_tb.v blink.v
vvp blink.vvp -lxt2
```

Pierwsza komenda dokonuje kompilacji plików źródłowych do kodu pośredniego, w wyniku której powstaje plik *blink.vvp*, stanowiący kod dla maszyny wirtualnej **vvp**. Druga komenda uruchamia maszynę wirtualną, która wykonuje plik *blink.vvp*, w efekcie czego, zgodnie z zawartością pliku *blink_tb.v*, powstaje plik zawierający diagramy, będące efektem symulacji. Tak powstały plik możemy otworzyć bezpośrednio w programie *GTKWave*, w efekcie czego ujrzymy

przebiegi wewnątrz projektowanego układu (**rysunek 3**). W programie służącym do przeglądania przebiegów możemy wybrać interesujące nas sygnały i przeglądać je w całości lub oglądać ich fragmenty.

W procesie syntezy będziemy wykorzystywać pliki: *blink_top.v*, stanowiący główny moduł projektu *blink.v* oraz *icecore_pins.pcf*, opisujący mapowanie wejść i wyjść do fizycznych wyprowadzeń układu. Ponieważ mapowanie w komercyjnych środowiskach najczęściej realizowane jest w sposób graficzny, omówimy nieco szerzej zawartość tego pliku projektu, która wygląda w sposób następujący:

```
set_io clk 60
set_io led[0] 49 # B81/GBin5 L0
Blue led / S2 button
set_io led[1] 52 # B82/GBin4 L1
Green led / S1 button
set_io led[2] 55 # B91 L3 Yellow
led
set_io led[3] 56 # B94 cs Red led
```

Deklaracja modułu głównego projektu wygląda tak:

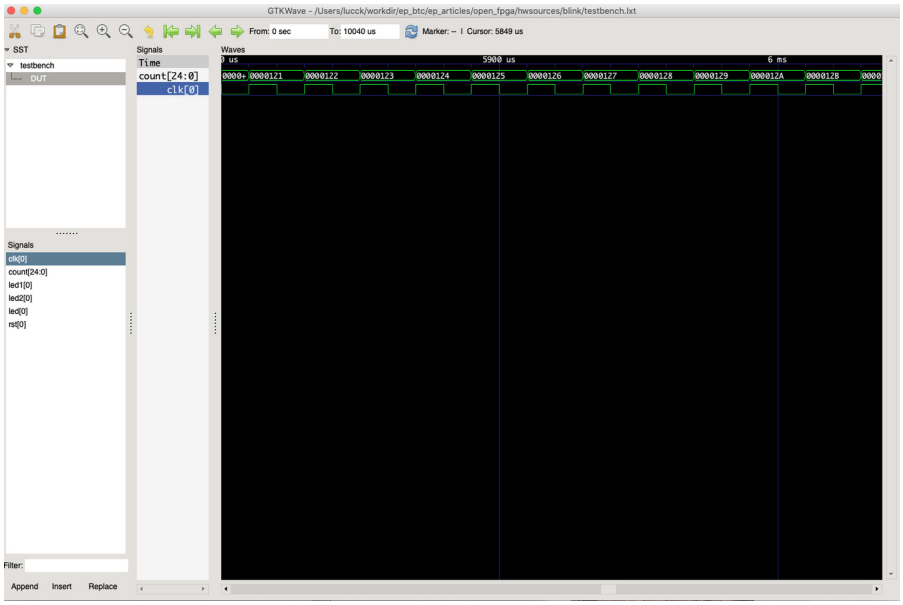
```
module chip (
        // 25MHz clock input
        input  clk,
        // Led outputs
        output [3:0] led
);
```

Poszczególne linie pliku mapującego zawierają zestaw komend *set_io*, nazwę sygnału w module głównym, a następnie numer fizycznego wyprowadzenia sygnału w układzie FPGA. Opcjonalnie możemy również dodać własny komentarz, posługując się znakiem #. Ponieważ w przykładowym projekcie wykorzystujemy obudowę TQFP, oznaczenia wyprowadzeń występują w postaci numerycznej.

Wróćmy teraz do głównego tematu, czyli syntezy, która realizowana jest w kilku krokach za pomocą następujących komend:

```
yosys -q -p "synth_ice40 -json blink.json" blink.v
blink_top.v
nextpnr-ice40 --hx8k --package tq144:4k --json blink.
json --asc blink.asc --pcf icecore_pins.pcf
icepack blink.asc blink.bin
```

W pierwszym poleceniu wywołujemy narzędzie **yosys** odpowiedzialne za wygenerowanie pliku netlisty. Jako pierwszy argument przekazywana jest opcja *synth_ice40*, pozwalająca na skorzystanie z modułów specyficznych dla rodziny układów iCE40. Następny argument *json* nakazuje wygenerowanie listy połączeń w formacie JSON. Jako kolejne argumenty przekazywane są nazwy wszystkich plików źródłowych, które biorą udział w syntezowaniu układu. Wynikiem działania polecenia jest plik o nazwie *blink.json*, zawierający listę połączeń.



Rysunek 3. Przebiegi wewnątrz projektowanego układu

Polecenie **nextpnr-ice40** odpowiedzialne jest za mapowanie komponentów oraz trasowanie połączeń wewnątrz układu FPGA. Pierwszy argument zawiera typ układu, dla którego syntezowany jest projekt, natomiast drugi argument zawiera rodzaj obudowy wybranego układu. Jako argument *json* przekazujemy plik z listą połączeń, wygenerowany wcześniej za pomocą narzędzia **yosys**. Kolejny argument *asc* określa nazwę pliku wyjściowego, natomiast w ostatnim argumencie przekazujemy plik, zawierający mapowanie wyprowadzeń układu. Wynikiem działania polecenia jest powstanie listy połączeń w postaci tekstowej, specyficznej dla układu iCE40.

W ostatnim poleceniu wywołujemy program **icepack**, którego zadaniem jest przekształcenie listy połączeń z postaci tekstowej do wynikowego pliku *bitstream*. Jako pierwszy argument polecenie przyjmuje netlistę specyficzną dla układu, wygenerowaną za pomocą narzędzia **nextpnr-ice40**, natomiast drugi argument stanowi nazwa pliku wyjściowego, który będzie zawierał finalny plik z zawartością *bitstream* dla układu FPGA. Tak powstały plik możemy następnie wykorzystać do konfiguracji układu FPGA lub zaprogramowania układu pamięci, z której układ FPGA odczyta konfigurację.

Lucjan Bryndza, EP
lucjan.bryndza@boff.pl