

# Tytuł: Basys3 Icy Tower

**Autorzy: Jan Wołowiec (JW),  
Wojciech Mrzygłód (WM)**

Ostatnia modyfikacja: 05.09.2019

1.	Repozytorium git projektu .....	1
2.	Wstęp .....	2
3.	Specyfikacja .....	2
3.1.	Opis ogólny algorytmu .....	2
3.2.	Tabela zdarzeń .....	4
4.	Architektura .....	5
4.1.	Schemat z IP Integratora .....	6
4.2.	Moduł: VGA Timing (RTL) .....	6
4.2.1.	Wejścia i wyjścia modułu .....	7
4.3.	Moduł: VGA Background (IP Core) .....	7
4.3.1.	Wejścia i wyjścia modułu .....	7
4.3.2.	Rejestry modułu i adresy na magistrali systemowej .....	8
4.4.	Moduł: VGA Block (IP Core) .....	8
4.4.1.	Wejścia i wyjścia modułu .....	10
4.4.2.	Rejestry modułu i adresy na magistrali systemowej .....	11
4.5.	Moduł: VGA Time Counter (IP Core) .....	12
4.5.1.	Wejścia i wyjścia modułu .....	12
4.5.2.	Rejestry modułu i adresy na magistrali systemowej .....	12
4.6.	Moduł: VGA Text Block (IP Core) .....	13
4.6.1.	Wejścia i wyjścia modułu .....	14
4.6.2.	Rejestry modułu i adresy na magistrali systemowej .....	14
4.7.	Moduł: VGA Keyboard Controller (IP Core) .....	15
4.7.1.	Wejścia i wyjścia modułu .....	16
4.7.2.	Rejestry modułu i adresy na magistrali systemowej .....	16
4.8.	Moduł: VGA Interrupt (RTL) .....	17
4.8.1.	Wejścia i wyjścia modułu .....	17
4.9.	Moduł: XADC (Hardware IP Core) .....	17
4.10.	Mapa pamięci .....	18
4.11.	Skrypt linkera gcc (lscript.ld) .....	18
4.12.	Pamięć tekstur (textures.h) .....	18
4.13.	Fpga interface (fpga_interface.hpp, fpga_interface.cpp) .....	18
4.14.	Gra (moduły Game, Floor, Level, Player, Point2d, Line2d) .....	19
4.15.	Klawiatura (moduł Keyboard) .....	20
4.16.	Menu (moduł Menu) .....	20
5.	Implementacja. Zaawansowanie na 05.09.2019 – 100% .....	20
6.	Film. Zaawansowanie na 05.09.2019 – 100% .....	20

## 1. Repozytorium git projektu

Adres repozytorium GITa:

[https://github.com/NVi5/Basys3\\_IcyTower.git](https://github.com/NVi5/Basys3_IcyTower.git)

## 2. Wstęp

Historia naszego projektu z Układów Cyfrowych zaczyna się w momencie, gdy nadszedł czas wyboru gry której podejmiemy się wykonać. Przeszukując w pamięci tytuły gier z którymi mieliśmy do czynienia, ciężko było wybrać coś odpowiedniego, gdyż większość z nich była zbyt złożona, a przez co niemożliwa do przeniesienia na nasz moduł fpga z tak ograniczonymi zasobami. Wybawieniem dla nas okazały się tytuły retro jak choćby wykonana przez nas wersja Icy Tower na Basysie 3. Gra ta cechuje się małą złożonością oraz powtarzalnością, a mimo to jako dzieci potrafiliśmy nad nią spędzić wiele godzin starając się osiągać coraz to lepsze wyniki. Jej prosty charakter pozwolił nam ją wykonać wykorzystując jedynie 11 tekstur o rozmiarze 16x16 pikseli. W ramach projektu musieliśmy przygotować bloki odpowiadające za wyświetlanie kolejnych elementów gry takich jak: gracz, platformy, tło, licznik itp. odpowiednio sterowanych przez mikroprocesor Microblaze, co pozwoliło na implementację algorytmów gry w prosty sposób, oraz bloki do obsługi interfejsów wejścia i wyjścia które opisaliśmy poniżej.

## 3. Specyfikacja

### 3.1. Opis ogólny algorytmu

Po włączeniu zasilania płytki basys3 wyświetlane jest menu zawierające dwie pozycje: start gry i wybór poziomu trudności. Do wyboru mamy trzy poziomy trudności – Easy, Medium i Hard. W tle wyświetlana jest jedna platforma, model gracza, tekstura tła i tekstura ścian bocznych. Do poruszania się po menu wykorzystujemy klawisze W i S (lub strzałki, ale mogą one nie działać w zależności od klawiatury jakiej użyjemy), a do potwierdzenia wyboru klawisz Enter.

Po wybraniu poziomu trudności i wybraniu opcji Start z menu rozpoczyna się gra. Gracz znajduje się na środku platformy o numerze 0. Pozostałe  $N - 1$  ( $N$  – liczba wyświetlanych platform, ustawiana dowolnie w programie) platform jest generowane losowo (za pomocą funkcji rand() z biblioteki standardowej języka C, funkcja rand() inicjalizowana jest szumem z ADC). Szerokość generowanych platform zależy od wybranego poziomu trudności (każdy poziom trudności ma zdefiniowany w oprogramowaniu zakres szerokości platform). W lewym górnym rogu ekranu wyświetlana jest liczba punktów (najwyższa platforma na którą udało się wskoczyć), przy lewej krawędzi wyświetlany jest licznik czasu – za każdym kiedy licznik osiągnie wartość 0, prędkość przesuwania platform zwiększa się, a w zależności od poziomu trudności okres licznika ma różną wartość.

Pozycja gracza jest obliczana na podstawie równań ruchu z odpowiednio ustawionym przyspieszeniem i krokiem czasowym.

Gry nie da się wygrać. Celem gracza jest dotarcie na jak najwyższą platformę. Gra kończy się kiedy pionowa pozycja gracza osiągnie wartość 0, czyli wtedy kiedy gracz „spadnie” poza dolną krawędź ekranu.

Naciskając A i D (lub strzałki w lewo i w prawo) sterujemy ruchem gracza w lewo i w prawo – zwiększana jest wtedy jego prędkość pozioma z każdą iteracją algorytmu. Po puszczeniu strzałki z każdą iteracją prędkość pozioma maleje o 10% kiedy gracz jest na platformie i 15% kiedy jest w „powietrzu”. Gracz może odbijać się od ścian bocznych z zachowaniem 80% prędkości przed odbiciem (ponadto wszystkie te współczynniki utraty prędkości można dowolnie modyfikować w programie). Po naciśnięciu spacji wykonujemy skok w górę – zwiększana jest prędkość pionowa o stały czynnik plus wartość prędkości poziomej. Skutkuje to tym, że im szybciej porusza się gracz, tym wyżej może skakać.

Po uruchomieniu gry platformy nie poruszają się. Zaczynają się poruszać dopiero po osiągnięciu przez gracza odpowiedniej wysokości (w zależności od ustawień programu). Kiedy podczas skoku pionowa pozycja gracza osiągnie wartość większą od obszaru rysowania wyświetlacza platformy są przesuwane o różnicę między pozycją pionową a maksymalną wysokością wyświetlania, a pozycja gracza jest ustawiana wtedy na maksymalną wysokość możliwą do wyświetlenia na monitorze. Daje to efekt przyspieszenia gry kiedy gracz miałby wyskoczyć poza obszar wyświetlania.

Kiedy gracz jest w powietrzu i spada (to znaczy kiedy jego prędkość pionowa jest ujemna), to obliczane są kolizje z platformami i ich odległości od gracza. Kolizje są wykrywane poprzez wyznaczenie punktu przecięcia dwóch

odcinków – pierwszy to odcinek pomiędzy punktem gdzie znajdował się gracz w poprzedniej iteracji a obliczoną pozycją gracza w obecnej iteracji, natomiast drugi odcinek to górna krawędź platformy. Obliczenia te są powtórzone dla każdej platformy. Kiedy jedna lub więcej kolizji zostanie wykrytych, to pozycja pionowa gracza zostaje zablokowana na najbliższej kolidującej z nim platformie. Gracz pozostaje zablokowany na platformie do czasu naciśnięcia spacji (w celu wykonania ponownego skoku), przemieszczenia się poza obszar platformy, lub osiągnięcia przez platformę pozycji 0 (co skutkuje zakończeniem gry).

Grę można w dowolnym momencie zatrzymać naciskając klawisz ESC. Wyświetlone zostaje wtedy menu pauzy zawierające obecny wynik i dwie opcje: wznowienie gry i jej zakończenie. Powrót do gry następuje po ponownym naciśnięciu klawisza ESC. Po zakończeniu gry wyświetlane jest menu z uzyskanym wynikiem i dwiema opcjami: wybór poziomu i restart.

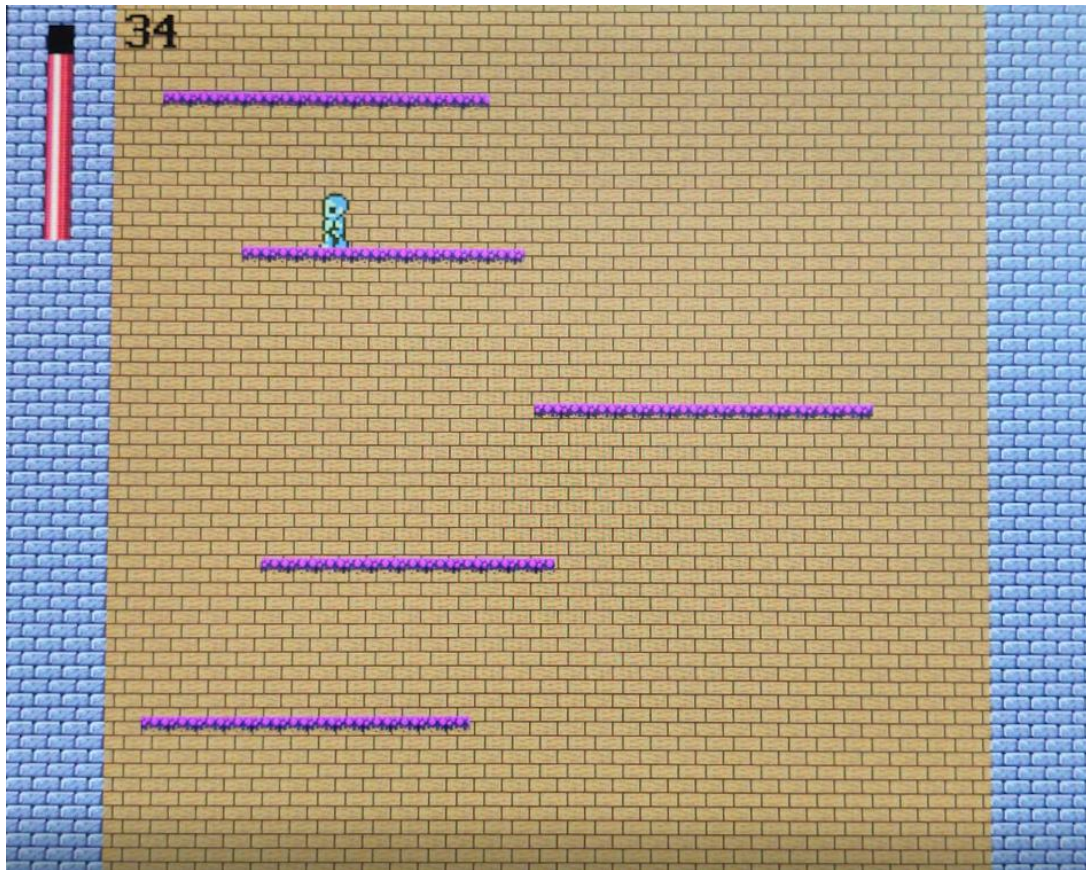
Ponadto tekstury w tle nie są statyczne. Mogą się poruszać z różnymi prędkościami. Tekstura ścian bocznych porusza się z taką samą prędkością co platformy, a tekstura tła porusza się wolniej co daje efekt paralaksy. Model gracza również zmienia się w zależności od kierunku w jakim się poruszamy lub kiedy nie poruszamy się. Dodatkowo co 50 platform zmienia się tekstura kolejnych platform. Tekstury platform przechowywane są w pamięci ram procesora, więc co do ich ilości ogranicza nas tylko rozmiar dostępnej pamięci.

Pozycja, rozmiar i skalowanie poszczególnych elementów ustawiane są za pomocą magistrali AXI4-Lite, a tekstury przesyłane są do bloków odpowiedzialnych za ich wyświetlanie za pomocą magistrali AXI4-Stream przy użyciu DMA w trybie MM2S (Memory Mapped to Stream).

Wszystkie obliczenia wykonywane są raz na jedną klatkę, a aktualizacja wyświetlanych elementów wykonywana jest poza obszarem wyświetlania. Kiedy liczniki hcount i vcount osiągną wartości odpowiednio X-1 i Y-1 (X i Y to rozdzielczość wyświetlacza np. w naszym przypadku będzie to X = 1280, Y = 1024), procesor otrzymuje sygnał przerwania, które informuje o zakończeniu wyświetlania obecnej klatki, możliwości aktualizacji wyświetlanego obrazu i możliwości rozpoczęcia obliczeń dla kolejnej klatki.



Rys 1. Co chcielibyśmy uzyskać (źródło: <https://www.spidersweb.pl/2016/08/icy-tower.html>).



Rys 2. Co udało się uzyskać

### 3.2. Tabela zdarzeń

Zdarzenie	Kategoria	Reakcja systemu
Włączenie zasilania, reset	Ekran startowy	Inicjalizacja generatora liczb losowych, peryferiów AXI, ustawienie elementów interfejsu na odpowiednich pozycjach, Wyświetlenie menu startowego, tła, ścian bocznych, modelu gracza, pierwszej platformy i licznika czasu
Naciśnięcie klawiszy strzałek góra/dół lub W/S	Menu gry	Zmiana wybranej opcji
Naciśnięcie klawisza Enter	Menu gry	Zatwierdzenie wybranej opcji
Wybranie opcji Level	Menu gry	Zmiana poziomu trudności
Wybranie opcji Start	Menu gry	Rozpoczęcie gry
Naciśnięcie strzałki w lewo lub A	Gra	Ruch w lewo, zmiana tekstury gracza
Naciśnięcie strzałki w prawo lub D	Gra	Ruch w prawo, zmiana tekstury gracza
Gracz przestaje się poruszać	Gra	Zmiana tekstury gracza
Naciśnięcie klawisza spacji	Gra	Skok, wysokość skoku zależna od prędkości poziomej
Gracz znajduje się powyżej pozycji 800 pikseli	Gra	Platformy zaczynają się poruszać
Naciśnięcie klawisza ESC	Gra	Zatrzymanie gry i wyświetlenie menu pauzy
Gracz porusza się w dół	Gra	Obliczanie kolizji

Kolizja została wykryta	Gra	Zablokowanie gracza na najbliższej kolidującej z nim platformie
Gracz jest poza obszarem wyświetlania	Gra	Przesunięcie platform o różnicę pozycji pomiędzy pozycją gracza a maksymalną pozycją możliwą do wyświetlenia, ustawienie gracza w maksymalnej pozycji możliwej do wyświetlenia
Gracz przemieścił się poza platformę	Gra	Gracz spada z platformy
Gracz uderza w ścianę	Gra	Odbicie z 80% prędkości poziomej
Gracz jest w powietrzu	Gra	Prędkość zmniejsza się o 15% z każdą klatką obrazu
Gracz jest na platformie	Gra	Prędkość zmniejsza się o 10% z każdą klatką obrazu
Osiągnięcie przez gracza pozycji 0 w pionie	Gra	Koniec gry, wyświetlenie menu końca gry
Licznik czasu osiąga wartość 0	Gra	Ponowne napełnienie licznika, przyspieszenie gry
Gracz wskakuje na kolejną platformę	Gra	Zwiększenie licznika punktów
Gracz przeskoczył kolejne 50 platform	Gra	Zmiana tekstury platform i zmniejszenie zakresu ich szerokości
Przesunięcie platform	Gra	Przesunięcie tekstury tła i ścian bocznych
Włączenie menu pauzy	Menu pauzy	Wyświetlenie obecnego wyniku i dwóch opcji menu: kontynuacja i wyjście z gry
Naciśnięcie klawisza ESC / wybór opcji kontynuacji	Menu pauzy	Wznowienie gry
Wybór opcji wyjścia z gry	Menu pauzy	Wyjście do menu gry
Zakończenie gry	Menu końca gry	Wyświetlenie uzyskanego wyniku i dwóch opcji – restart i wybór poziomu trudności
Wybranie opcji zmiany poziomu trudności	Menu końca gry	Zmiana poziomu trudności
Wybranie opcji restartu gry	Menu końca gry	Restart gry
Wyświetlenie jednej klatki obrazu	Sposób działania systemu	Odświeżenie elementów interfejsu, rozpoczęcie obliczeń dla kolejnej klatki
Wyświetlenie jednej platformy	Sposób działania systemu	Zmiana pozycji, rozmiaru i tekstury bloku sprzętowego wyświetlającego platformę
Przełączenie przełącznika SW0	Sposób działania systemu	Restart systemu

## 4. Architektura

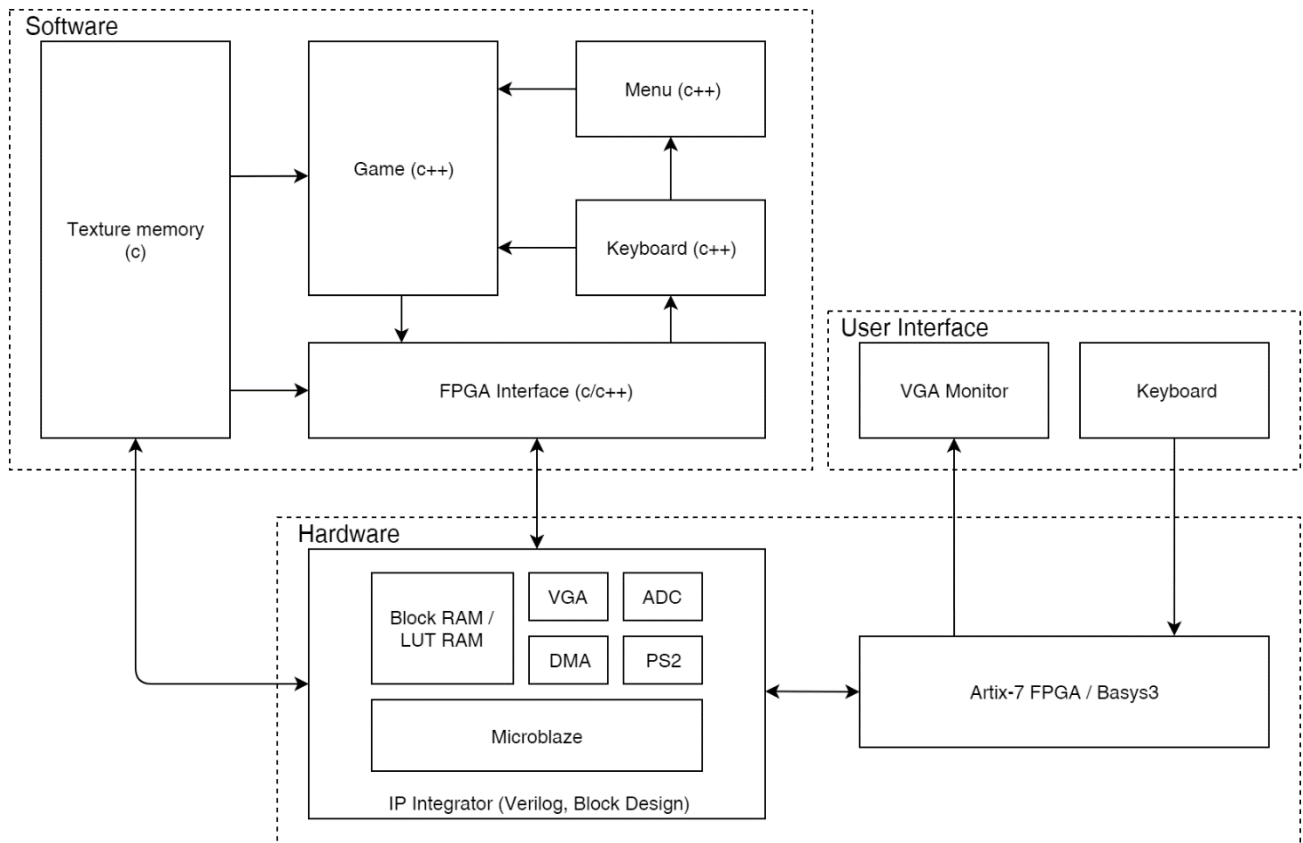
Projekt jest podzielony na trzy główne części (Rys 3.) - Software, Hardware i Interfejs użytkownika.

Blok software jest to program napisany na procesor Microblaze. Składa się z silnika gry, modułu wyświetlającego menu gry i modułu obsługi klawiatury. Poza tym zawiera jeszcze pamięć tekstur – ładowaną do pamięci RAM podczas programowania procesora i interfejs który łączy wysokopoziomowy kod gry z niskopoziomową obsługą sprzętu. Takie rozdzielenie poszczególnych elementów projektu pozwala na łatwe portowanie programu na inne architektury sprzętowe i rozwijanie oprogramowania będącego niezależnym od sprzętu (uniknięcie tzw. vendor lock-in).

Hardware to część projektu odpowiadająca za konfigurację FPGA i komunikację z urządzeniami interfejsu użytkownika. Ten blok został wykonany w programie Vivado, używając narzędzia IP integrator. Użyte

bloki sprzętowe FPGA (Hard IP Core) to: Block RAM i ADC. Gotowe IP Core z dostępnego repozytorium Vivado: Procesor Microblaze wraz z potrzebnymi komponentami do jego uruchomienia i elementami magistrali AXI4-Lite, DMA do przesyłania tekstur i UART do debugowania programu. Stworzone od podstaw peryferia AXI4-Lite, AXI4-Stream i bloki generujące przerwania: VGA Timing, VGA Background, VGA Block, VGA Time Counter, VGA Text Block, Keyboard Controller i VGA Interrupt. Wszystkie bloki odpowiedzialne za wyświetlanie elementów na monitorze połączone są w pipeline.

Ostatni blok to interfejs użytkownika. Do wyświetlania efektów działania programu został użyty monitor VGA, a do kontroli programu klawiatura podłączona do konwertera USB/PS2 na płycie Basys3.



Rys 3. Ogólny schemat architektury systemu.

#### 4.1. Schemat z IP Integratora

Z powodu dużego rozmiaru schematu został on załączony na końcu raportu.

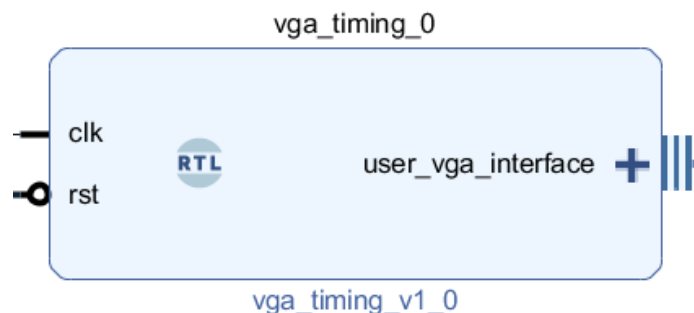
#### 4.2. Moduł: VGA Timing (RTL)

Moduł generujący sygnały potrzebne do działania interfejsu VGA i wyświetlania obrazu na monitorze. Jest to dokładna kopia modułu napisanego na zajęciach laboratoryjnych UEC2.

Moduł został napisany w verilogu i umieszczony na schemacie korzystając z możliwości środowiska vivado, które umożliwia automatyczną konwersję modułów veriloga na moduły IP integratora.



#### 4.2.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
clk	wejście	Zegar sygnału VGA
rst	Wejście (aktywne stanem niskim)	Reset synchroniczny modułu
user_vga_interface	wyjście	Wyjście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline

#### 4.3. Moduł: VGA Background (IP Core)

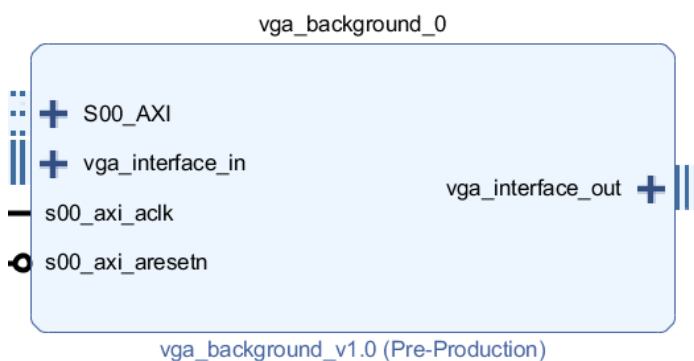
Moduł wyświetlający tło i teksturę ścian bocznych, dodatkowo z możliwością ich przesuwania, co może służyć do stworzenia efektu paralaksy.

Do stworzenia modułu wykorzystane zostały moduły wyświetlające tło i obrazek napisane na zajęciach laboratoryjnych UEC2. Wprowadzonymi zmianami są:

- Możliwość wyświetlania dwóch tekstur w zależności od licznika hcount
- Możliwość przesuwania tekstury w pionie o zadaną liczbę pikseli

Moduł został „zapakowany” w vivado jako AXI-4 Peripheral i posiada dwa rejestry – jeden do ustawienia przesunięcia tekstury tła, drugi do ustawienia przesunięcia tekstury ścian bocznych.

##### 4.3.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
S00_AXI	AXI4-Lite Slave	Port do podłączenia modułu do magistrali AXI4-Lite, umożliwia dostęp do rejestrów modułu.
s00_axi_aclk	Wejście	Zegar magistrali AXI4-Lite
s00_axi_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Lite aktywny stanem niskim
vga_interface_in	wejście	Wejście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
vga_interface_out	wyjście	Wyjście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline

#### 4.3.2. Rejestry modułu i adresy na magistrali systemowej

##### Adresy:

VGA\_BACKGROUND\_BASE = 0x44A2 0000

##### Rejestry:

VGA_BACKGROUND_SHIFT_BG	VGA_BACKGROUND_BASE+0x00
31	5 4 0
Not used	SHIFT_BG [4:0]

**SHIFT\_BG [4:0]** – przesunięcie tekstury tła w pionie o SHIFT\_BG pikseli.

VGA_BACKGROUND_SHIFT_SIDES	VGA_BACKGROUND_BASE+0x04
31	5 4 0
Not used	SHIFT_SIDES [4:0]

**SHIFT\_SIDES [4:0]** – przesunięcie tekstury ścian bocznych w pionie o SHIFT\_SIDES pikseli.

#### 4.4. Moduł: VGA Block (IP Core)

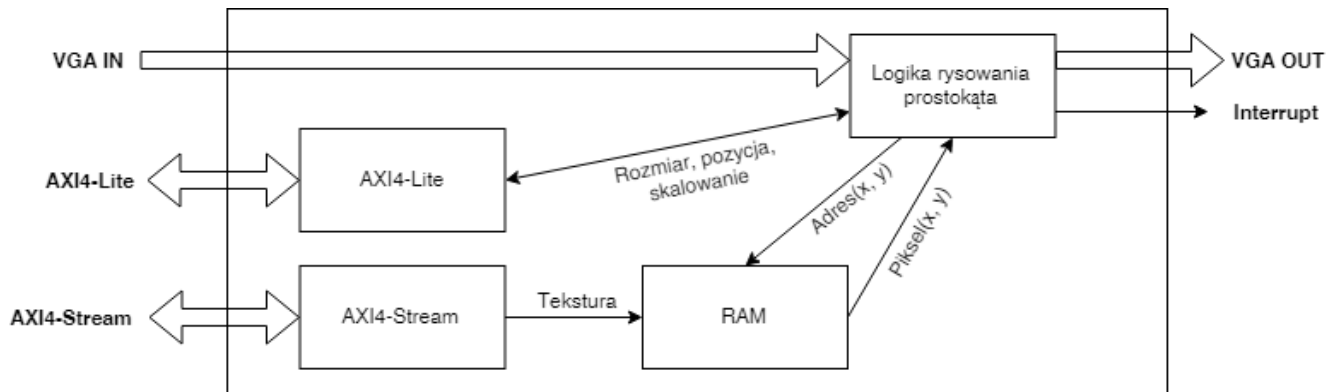
Ten moduł służy do wyświetlania na monitorze prostokąta o dowolnej pozycji, rozmiarze i teksturze na monitorze. Dodatkowo moduł obsługuje prostą przezroczystość tekstur (kolor 0x000 jest niewidoczny). Ponadto potrafi generować sygnał przerwania po zakończeniu rysowania prostokąta.

Moduł zawiera wewnątrz bloki:

- Moduł do obsługi magistrali AXI4-Lite (wygenerowany automatycznie)
- Moduł do obsługi magistrali AXI4-Stream (maszyna stanów)
- Pamięć RAM na teksturę bloku o rozmiarze ustawianym w parametrach modułu



- Logika rysowania prostokąta

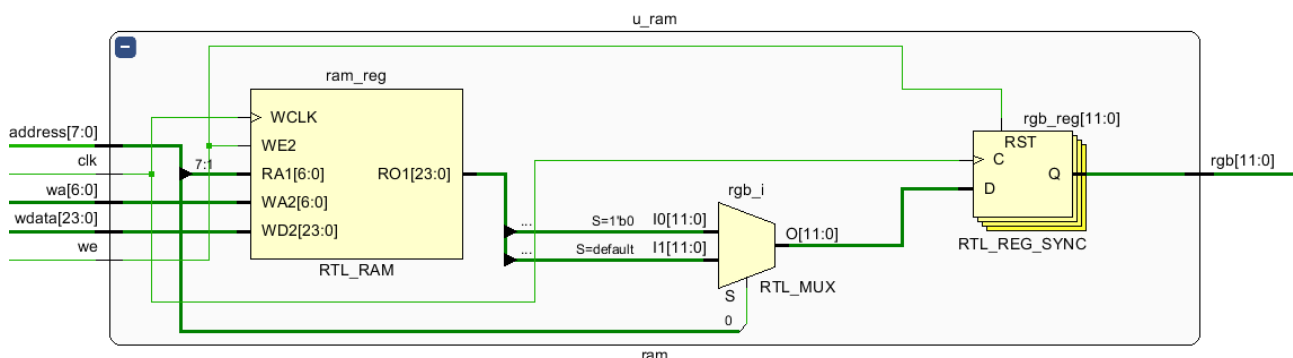


Rys 4. Schemat blokowy modułu i zasada działania.

Moduł został wygenerowany poprzez odpowiednie narzędzie do „pakowania” IP Corów w vivado (**Tools > Create and package new IP...**). Do obsługi magistrali AXI4-Lite został wykorzystany gotowy kod wygenerowany przez środowisko. A do zapisywania pamięci ram przez magistralę AXI4-Stream została napisana prosta maszyna stanów.

Zadaniem maszyny stanów jest zawsze utrzymywać sygnał **t\_ready** na poziomie wysokim, ponieważ pamięć RAM jest zawsze gotowa do zapisania. Gdy tylko master magistrali AXI4-Stream ustawi sygnał **t\_valid**, to sygnał **write\_enable (we)** pamięci ram jest ustawiany i zawartość linii **t\_data** magistrali jest wpisywana do pamięci RAM. Z każdym cyklem zegara magistrali AXI4-Stream zwiększany jest wskaźnik zapisu pamięci. Gdy master kończy transfer sygnalizuje to sygnałem **t\_last** w tym momencie maszyna stanów przechodzi do stanu IDLE, gdzie dezaktywuje sygnał **write\_enable** i ustawia wskaźnik zapisu na 0.

Aby w pełni wykorzystać szerokość magistrali AXI4-Stream, do pamięci jednocześnie są zapisywane informacje o kolorze dwóch pikseli. Dlatego szerokość adresu zapisu (**wa[6:0]**) jest o jeden bit mniejsza niż adresu odczytu (**address[7:0]**), a wyjście (**rgb[11:0]**) ma o połowę mniej bitów niż wejście (**wdata[23:0]**) – do pamięci zapisywane są 24 bity a odczytywane jest 12 bitów. Ten zabieg zwiększa przepustowość magistrali dwukrotnie.



Rys 5. Schemat RTL pamięci RAM.

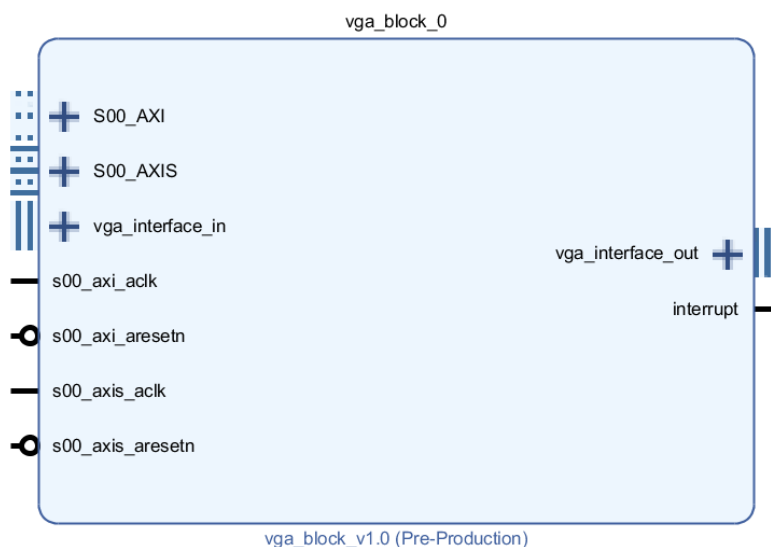
Moduł ten jest wykorzystany dwukrotnie – do rysowania platform (Floors) i do rysowania modelu gracza (Player). W przypadku rysowania platform jeden moduł rysuje wszystkie platformy. Kiedy następuje przerwanie wtedy ustawiane są pozycja, rozmiar i tekstura następnej platformy. Natomiast w bloku rysującym model gracza przerwanie zakończenia rysowania prostokąta nie jest wykorzystywane –

aktualizacja parametrów wyświetlania następuje po zakończeniu rysowania pełnej klatki w osobnym przerwaniu, które generuje inny moduł.

Dodatkowo, aby umożliwić płynne przesuwanie bloku na monitorze, została wprowadzona możliwość rysowania prostokąta na ujemnej wysokości. Aby to zrobić należy do rejestru odpowiedzialnego za ustawianie pozycji pionowej wartość większą, niż wysokość monitora. Wtedy prostokąt rysowany jest u góry monitora zaczynając od ostatniej linii. Daje to efekt płynnego „wyjeżdżania” prostokąta z nad obszaru rysowania.

Zapisywać pamięć RAM tego modułu można na dwa sposoby. Pierwszy to użycie IP AXI-Stream FIFO, jednak ten sposób zużywa zarówno dużo pamięci RAM, jak i blokuje procesor na czas transmisji. Co w przypadku naszego systemu gdzie liczy się szybki czas odpowiedzi na przerwania nie możemy na to pozwolić, aby procesor oczekiwał na zakończenie transferu nic nie robiąc. Dlatego w naszym systemie wszystkie te bloki współpracują z DMA w trybie MM2S (Memory Mapped to Stream). Pozwala to procesorowi na prowadzenie obliczeń w czasie kiedy trwa transmisja tekstury. Może to się odbywać w ten sposób, dlatego że pamięć instrukcji i pamięć RAM gdzie znajduje się gra są w innej przestrzeni adresowej i na innej magistrali, niż pamięć RAM z której korzysta DMA.

#### 4.4.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
S00_AXI	AXI4-Lite Slave	Port do podłączenia modułu do magistrali AXI4-Lite, umożliwia dostęp do rejestrów modułu.
S00_AXIS	AXI4-Stream Slave	Port do podłączenia modułu do magistrali AXI4-Stream, służy do zapisywania tekstur do pamięci RAM modułu, może współpracować z DMA w trybie MM2S lub z AXI-Stream FIFO.
s00_axi_aclk	Wejście	Zegar magistrali AXI4-Lite
s00_axi_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Lite aktywny stanem niskim

s00_axis_aclck	Wejście	Zegar magistrali AXI4-Stream
s00_axis_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Stream aktywny stanem niskim
vga_interface_in	wejście	Wejście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
vga_interface_out	wyjście	Wyjście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
interrupt	wyjście	Przerwanie informujące o zakończeniu rysowania danego obiektu

#### 4.4.2. Rejestry modułu i adresy na magistrali systemowej

##### Adresy:

VGA\_BLOCK\_BASE (Player) = 0x44A1 0000

VGA\_BLOCK\_BASE (Floors) = 0x44A0 0000

##### Rejestry:

VGA_BLOCK_POSITION		VGA_BLOCK_BASE + 0x00	
31	27 26	16 15	11 10 0
Not used	YPOS [26:16]	Not used	XPOS [10:0]

**XPOS [10:0]** – pozycja X bloku graficznego na monitorze.

**YPOS [26:16]** – pozycja Y bloku graficznego na monitorze.

VGA_BLOCK_SIZE		VGA_BLOCK_BASE + 0x04	
31	27 26	16 15	11 10 0
Not used	HEIGHT [26:16]	Not used	WIDTH [10:0]

**WIDTH [10:0]** – szerokość bloku graficznego na monitorze.

**HEIGHT [26:16]** – wysokość bloku graficznego na monitorze.

VGA_BLOCK_SCALE		VGA_BLOCK_BASE + 0x08	
31	16 15		0
YSCALE [31:16]		XSCALE [15:0]	

**XSCALE [15:0]** – skalowanie tekstury w poziomie.

**YSCALE [31:16]** – skalowanie tekstury w pionie.

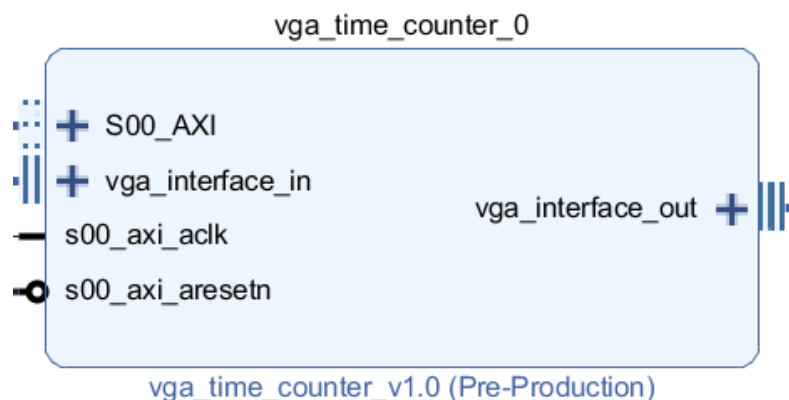
**Uwaga!** – wartości wpisane do rejestru **VGA\_BLOCK\_SCALE** są w formacie  $2^{\text{SCALE}}$ , nie ma możliwości skalowania tekstur o dowolny współczynnik (wpisanie 0x01 skaluje teksturę  $2^1$ , wpisanie 0x02 o  $2^2$ , itd.).

#### 4.5. Moduł: VGA Time Counter (IP Core)

Moduł ten ma za zadanie wyświetlanie licznika czasu na monitorze i jest analogiczny do modułu wyświetlającego prostokąt, który pisaliśmy na zajęciach. Jedynymi zmianami jest to, że teraz potrafi on wyświetlać teksturę od zadanego punktu w pionie a reszta wypełnienia prostokąta pozostaje czarna.

Moduł został „zapakowany” w vivado jako AXI-4 Peripheral i posiada dwa rejestry – jeden do ustawienia pozycji, drugi do ustawienia punktu od którego rysowana jest tekstura, czyli ustawienia licznika.

##### 4.5.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
S00_AXI	AXI4-Lite Slave	Port do podłączenia modułu do magistrali AXI4-Lite, umożliwia dostęp do rejestrów modułu.
s00_axi_aclk	Wejście	Zegar magistrali AXI4-Lite
s00_axi_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Lite aktywny stanem niskim
vga_interface_in	wejście	Wejście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
vga_interface_out	wyjście	Wyjście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline

##### 4.5.2. Rejestry modułu i adresy na magistrali systemowej

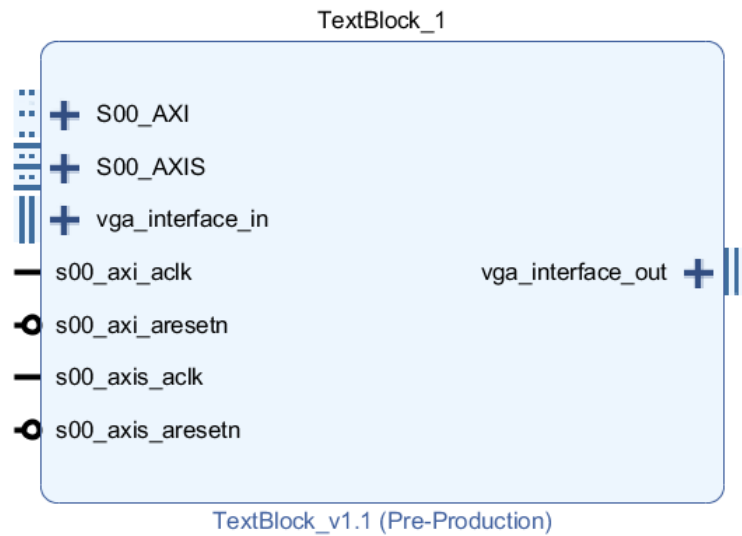
**Adresy:**

VGA\_TIME\_COUNTER\_BASE = 0x44A6 0000

**Rejestry:**

- Pozycja X
- Pozycja Y
- Kolor
- Skalowanie tekstu

#### 4.6.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
S00_AXI	AXI4-Lite Slave	Port do podłączenia modułu do magistrali AXI4-Lite, umożliwia dostęp do rejestrów modułu.
S00_AXIS	AXI4-Stream Slave	Port do podłączenia modułu do magistrali AXI4-Stream, służy do zapisywania tekstur do pamięci RAM modułu, może współpracować z DMA w trybie MM2S lub z AXI-Stream FIFO.
s00_axi_aclk	Wejście	Zegar magistrali AXI4-Lite
s00_axi_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Lite aktywny stanem niskim
s00_axis_aclk	Wejście	Zegar magistrali AXI4-Stream
s00_axis_aresetn	Wejście (aktywne stanem niskim)	Reset magistrali AXI4-Stream aktywny stanem niskim
vga_interface_in	wejście	Wejście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
vga_interface_out	wyjście	Wyjście sygnałów potrzebnych do generowania obrazu vga, umożliwia połączenie wielu modułów w pipeline
interrupt	wyjście	Przerwanie informujące o zakończeniu rysowania danego obiektu

#### 4.6.2. Rejestry modułu i adresy na magistrali systemowej

**Adresy:**

VGA\_TEXT\_BASE = 0x44A4 0000

### Rejestry:

VGA_TEXT_XPOS		VGA_TEXT_BASE + 0x00	
31	11 10		0
Not used		XPOS [10:0]	

**XPOS [10:0]** – pozycja X bloku tekstowego na monitorze.

VGA_TEXT_YPOS		VGA_TEXT_BASE + 0x04	
31	11 10		0
Not used		YPOS [10:0]	

**YPOS [10:0]** – pozycja Y bloku tekstowego na monitorze.

VGA_TEXT_SCALE		VGA_TEXT_BASE + 0x08	
31			0
SCALE [31:0]			

**SCALE [31:0]** – skalowanie tekstu.

**Uwaga!** – wartości wpisane do rejestru VGA\_TEXT\_SCALE są w formacie  $2^{\text{SCALE}}$ , nie ma możliwości skalowania tekstu o dowolny współczynnik (wpisanie 0x01 skaluje teksturę  $2^1$ , wpisanie 0x02 o  $2^2$ , itd.).

VGA_TEXT_COLOR		VGA_TEXT_BASE + 0x0C	
31	12 11		0
Not used		COLOR [11:0]	

**COLOR [11:0]** – Kolor tekstu w formacie 12 bitowym (RGB444).

## 4.7. Moduł: VGA Keyboard Controller (IP Core)

Moduł ten ma za zadanie pośredniczyć w komunikacji między procesorem, a klawiaturą PS2.

Do stworzenia modułu został wykorzystany moduł myszy wykorzystywany na zajęciach laboratoryjnych UEC2. Wprowadzonymi zmianami są:

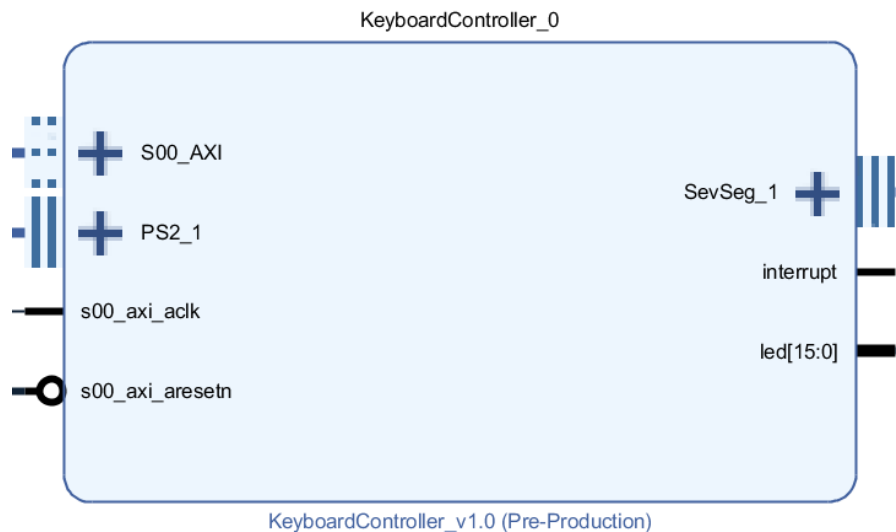
- Usunięto stany specyficzne dla myszy,
- Dostosowano moduł do obsługi klawiatury,

Dodatkowo powstał nadrzędny moduł który dekoduje otrzymane kody z klawiatury PS2 i ustawia odpowiedni stan w rejestrze „keys”, generując za każdym razem stan wysoki na wyjściu „interrupt”.

Moduł został „zapakowany” w vivado jako AXI-4 Peripheral i posiada jeden rejestr „keys”



#### 4.7.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
S00_AXI	AXI4-Lite Slave	Port do podłączenia modułu do magistrali AXI4-Lite, umożliwia dostęp do rejestrów modułu.
PS2_1	Inout, interfejs PS2	Port do podłączenia klawiatury z interfejsem PS2, lub klawiatury USB poprzez konwerter umieszczony na płycie.
s00_axi_aclk	wejście	Zegar magistrali AXI4-Lite
s00_axi_aresetn	wejście (aktywne stanem niskim)	Reset magistrali AXI4-Lite aktywny stanem niskim
SevSeg_1	wyjście	Wyjście sygnałów sterujących do wyświetlacza 7-segmentowego
interrupt	wyjście	Przerwanie informujące o naciśnięciu jednego z klawiszy sprawdzanych w module
led	wyjście	Wyjście sygnałów do sterowania diodami led

#### 4.7.2. Rejestry modułu i adresy na magistrali systemowej

##### Adresy:

KEYBOARD\_BASE = 0x44A3 0000

##### Rejestry:

KEYBOARD_KEYS	KEYBOARD_BASE + 0x00
31	7 6 0
Not used	KEYS [6:0]

**KEYS [6:0]** – każdy bit odpowiada stanowi poszczególnych przycisków sprawdzanych w module

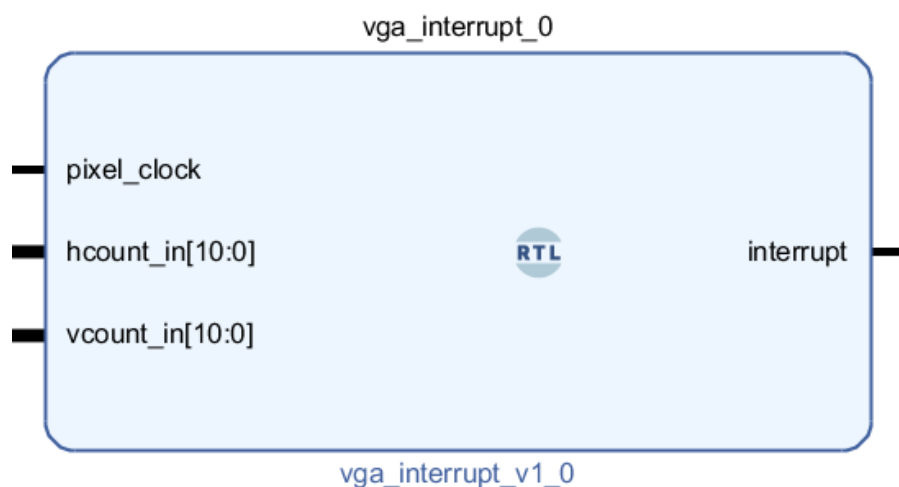
#### 4.8. Moduł: VGA Interrupt (RTL)

Moduł ten ma za zadanie wygenerować sygnał przerwania kiedy liczniki hcount i vcount osiągną odpowiednio X-1 i Y-1, gdzie X i Y to rozdzielczość monitora, w naszym przypadku jest to 1280 / 1024.

Według dokumentacji modułu kontrolera przerwań procesora Microblaze, w trybie synchronicznym wystarczy ustawić sygnał przerwania na okres jednego zegara. Co też poniższy moduł robi. W ten sposób sygnalizuje procesorowi zakończenie rysowania jednej pełnej klatki obrazu. W ten sposób możemy odświeżać pozycję obiektów na monitorze w sposób niezauważalny dla użytkownika.

Moduł został napisany w verilogu i umieszczony na schemacie korzystając z możliwości środowiska vivado, które umożliwia automatyczną konwersję modułów veriloga na moduły IP integratora.

##### 4.8.1. Wejścia i wyjścia modułu



Port	Typ (wejście/wyjście, master/slave)	Funkcja
pixel_clock	wejście	Zegar sygnału VGA
hcount_in	wejście	Licznik pozycji poziomej
vcount_in	wejście	Licznik pozycji pionowej
interrupt	wyjście	przerwanie

#### 4.9. Moduł: XADC (Hardware IP Core)

(Dokumentacja producenta:

[https://www.xilinx.com/support/documentation/user\\_guides/ug480\\_7Series\\_XADC.pdf](https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf))

Moduł przetwornika ADC, który wykorzystany jest do inicjalizowania generatora liczb pseudolosowych. Do jego obsługi użyty jest sterownik dostarczony przez producenta. Przykład użycia sterownika można znaleźć na stronie:

[https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/sysmon/examples/xsysmon\\_mon\\_polled\\_example.c](https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/sysmon/examples/xsysmon_mon_polled_example.c)

## 4.10. Mapa pamięci

Cell	Slave Interface	Base Name	Offset Ad... ^1	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_axi_intc	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_dma_0	S_AXI_LITE	Reg	0x41E0_0000	64K	0x41E0_FFFF
axi_dma_1	S_AXI_LITE	Reg	0x41E1_0000	64K	0x41E1_FFFF
axi_dma_2	S_AXI_LITE	Reg	0x41E2_0000	64K	0x41E2_FFFF
Floors	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
Player	S00_AXI	S00_AXI_reg	0x44A1_0000	64K	0x44A1_FFFF
vga_background_0	S00_AXI	S00_AXI_reg	0x44A2_0000	64K	0x44A2_FFFF
KeyboardController_0	S00_AXI	S00_AXI_reg	0x44A3_0000	64K	0x44A3_FFFF
TextBlock_0	S00_AXI	S00_AXI_reg	0x44A4_0000	64K	0x44A4_FFFF
xadc_wiz_0	s_axi_lite	Reg	0x44A5_0000	64K	0x44A5_FFFF
vga_time_counter_0	S00_AXI	S00_AXI_reg	0x44A6_0000	64K	0x44A6_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	32K	0xC000_7FFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF

Rys 6. Mapa Pamięci

## 4.11. Skrypt linkera gcc (lscript.ld)

Z powodu użycia DMA, do systemu musiała zostać dodana pamięć RAM dostępna dla peryferiów magistrali AXI4-Lite (axi\_bram\_ctrl\_0 o rozmiarze 32 Kb). Pamięć RAM procesora nie może zostać użyta, ponieważ podłączona jest do niego za pomocą innej magistrali danych.

(wyjaśnienie na forum Xilinx Community: <https://forums.xilinx.com/t5/Embedded-Processor-System-Design/Can-I-DMA-Microblaze-s-Local-Memory/td-p/905517>)

Dlatego też niezbędna była modyfikacja skryptu linkera. Dodane zostały sekcje .texture\_memory oraz .vga\_text. Sekcje te są kopią sekcji .bss, tylko umieszczone są w innej lokalizacji.

W programie zmienne i stałe umieszczane są w tej pamięci za pomocą odpowiednich atrybutów: `__attribute__((section(".vga_text")))` dla bufora tekstu i `__attribute__((section(".texture_memory")))` dla stałych zawierających tekstury.

## 4.12. Pamięć tekstur (textures.h)

Plik zawierający definicje wszystkich tekstur użytych w programie. Wszystkie zmienne tam zdefiniowane umieszczone są w sekcji .texture\_memory.

## 4.13. Fpga interface (fpga\_interface.hpp, fpga\_interface.cpp)

Moduł którego zadaniem jest połączenie wysokopoziomowego programu ze sprzętem.

API tego modułu (fpga\_interface.hpp) to pięć funkcji:

- void fpga\_interface\_initialize\_hardware(void) – funkcja inicjalizująca bloki sprzętowe takie jak DMA, ADC, inicjalizuje kontroler przerwań, i inicjalizuje generator liczb pseudolosowych

ośmioma kolejnymi próbkami szumu ADC. Funkcja ta musi zostać wywołana od razu po uruchomieniu programu. Funkcja nie przyjmuje argumentów i nic nie zwraca.

- `void fpga_interface_initialize(Game *)` – funkcja ustawiająca parametry początkowe wszystkich bloków odpowiedzialnych za wyświetlanie grafiki. Jako argument przyjmuje utworzoną instancję gry.
- `char *fpga_interface_get_buffer()` – funkcja zwracająca wskaźnik na bufor tekstowy znajdujący się w sekcji `.vga_text` służący instancji gry do wyświetlania informacji tekstowych. Funkcja nie przyjmuje argumentów.
- `uint32_t *fpga_interface_get_keyboard()` – funkcja zwracająca wskaźnik na zmienną 32 bitową przechowującą stan wciśniętych klawiszy na klawiaturze, używany przez klasę `Keyboard`. Funkcja nie przyjmuje argumentów.
- `void fpga_interface_run()` – funkcja, którą należy umieścić w pętli `main`, prowadzone są w niej obliczenia dotyczące stanu gry i sortowanie platform. Obecny sposób wyświetlania tego wymaga, dlatego że wszystkie platformy wyświetlane są jednym blokiem sprzętowym. Funkcja nie przyjmuje argumentów i nic nie zwraca.

Moduł `fpga_interface.cpp` zawiera definicje rejestrów wszystkich peryferiów używanych do wyświetlania obrazu na monitorze VGA.

Ponadto są tam też zaimplementowane funkcje:

- `dma_transfer_texture` – transfer tekstury przez DMA
- `interface_update` – funkcja wywoływana w przerwaniu od zakończenia wyświetlania klatki, aktualizuje pozycje elementów na monitorze, tak aby nie było to widoczne dla użytkownika.
- `block_update_handler` – funkcja wywoływana w przerwaniu od zakończenia wyświetlania obecnej platformy, przed wyświetlaniem kolejnej aktualizuje pozycję, rozmiar i teksturę.

#### 4.14. Gra (moduły `Game`, `Floor`, `Level`, `Player`, `Point2d`, `Line2d`)

Klasa `Game` implementuje wszystkie funkcjonalności opisane w rozdziale 3.1. w postaci maszyny stanów.

Na początku pliku `game.hpp` zawarte są definicje stałych wpływających na parametry gry.

Klasa również implementuje zestaw Getterów i Setterów umożliwiających pobranie informacji o stanie gry przez interfejs komunikujący się ze sprzętem.

Klasa `Player` przechowuje wszystkie informacje o aktualnym stanie gracza – pozycję, prędkość i przyspieszenie, posiada również metodę pozwalającą na obliczenie kolejnej pozycji gracza po czasie `delta_t`.

Klasa `Point2d` pozwala na tworzenie i manipulację punktami umieszczonymi w przestrzeni dwuwymiarowej.

Klasa `Line2d` umożliwia tworzenie linii pomiędzy dwoma punktami, oraz obliczanie kolizji pomiędzy nimi.

Klasa `Floor` dziedziczy po klasie `Line2d` dodając do niej teksturę, co tworzy platformę.

Klasa `Level` na podstawie wybranego poziomu trudności i postępu w grze wyznacza prędkość przesuwania się platform oraz ich wymiary.

#### 4.15. Klawiatura (moduł Keyboard)

Klasa Keyboard zawiera definicję klawiszy, wskaźnik na zmienną przechowującą stan klawiszy (ustawiany w konstruktorze) i jedną metodę:

- `bool GetKeyState(Keyboard::Key key)` – zwraca true lub false w zależności od tego czy klawisz określone w argumencie funkcji jest wciśnięty lub nie.

#### 4.16. Menu (moduł Menu)

Moduł którego zadanie jest kontrola bloku tekstowego Text Block, opisanego w sekcji 4.6. W zależności od zmiennych State, Position i Level pozwala on na wyświetlenie odpowiedniej treści na ekranie.

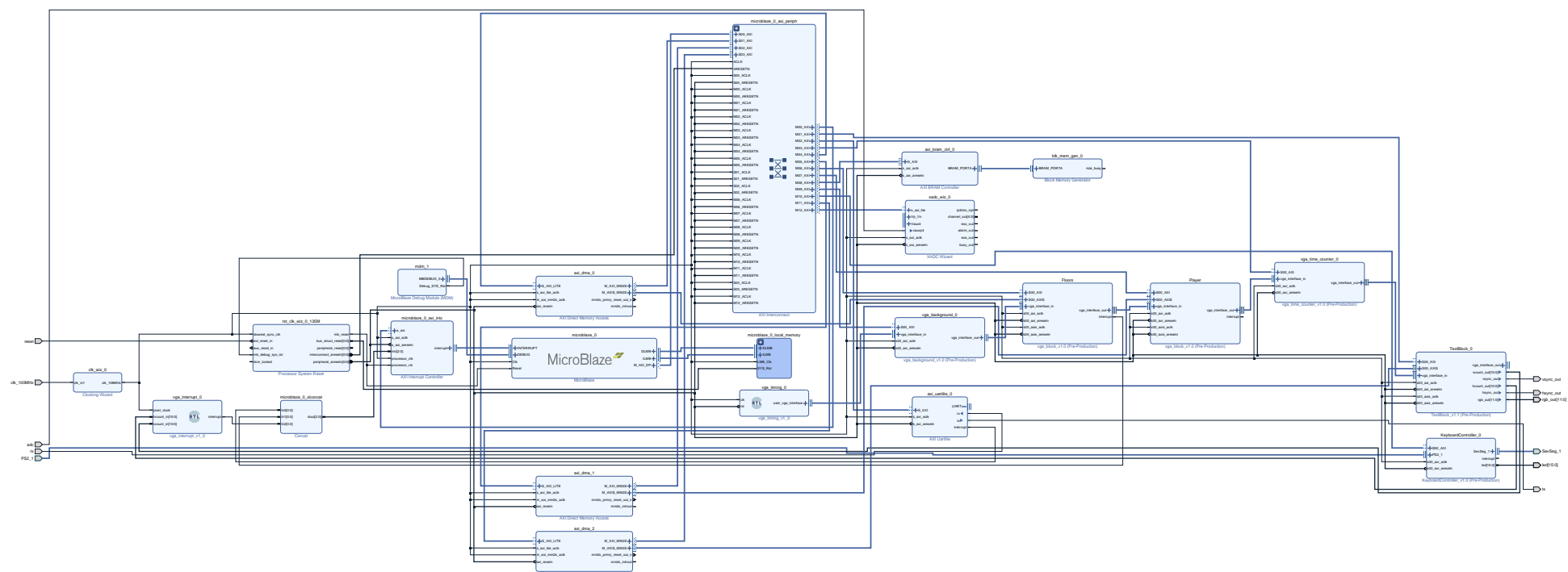
Wywołanie metody Draw powoduje aktualizację zawartości bufora tekstowego oraz docelowego stanu rejestrów kontrolujących moduł Text Block. Metody typu Set pozwalają na ustawienie wspomnianych wyżej zmiennych, natomiast metody typu Get pozwalają na pobranie docelowego stanu na jaki powinny być ustawione rejestry kontrolujące moduł Text Block.

### 5. Implementacja. Zaawansowanie na 05.09.2019 – 100%

### 6. Film. Zaawansowanie na 05.09.2019 – 100%

Link do ściągnięcia filmu:

<https://drive.google.com/file/d/1KpujNKRW4M4beiFdib6HPmreasmwTN8L>



## ***Instrukcja uruchomienia projektu***

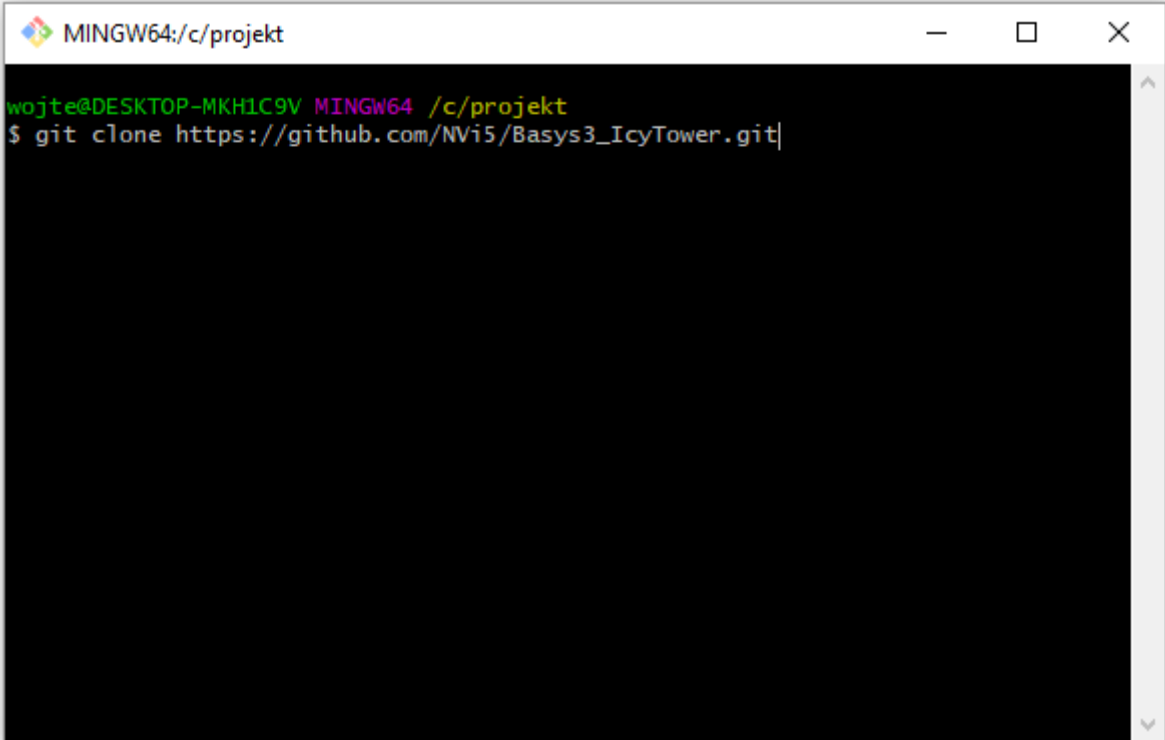
1. Tworzymy folder, do którego klonowane będzie repozytorium projektu.  
np. C:/projekt.

**Uwaga!** – ścieżka dostępu nie może zawierać spacji ani żadnych niestandardowych znaków, spowoduje to nieprawidłowe działanie Eclipse SDK.

2. Klonujemy repozytorium do utworzonego folderu.

Adres repozytorium:

[https://github.com/NVi5/Basys3\\_IcyTower.git](https://github.com/NVi5/Basys3_IcyTower.git)

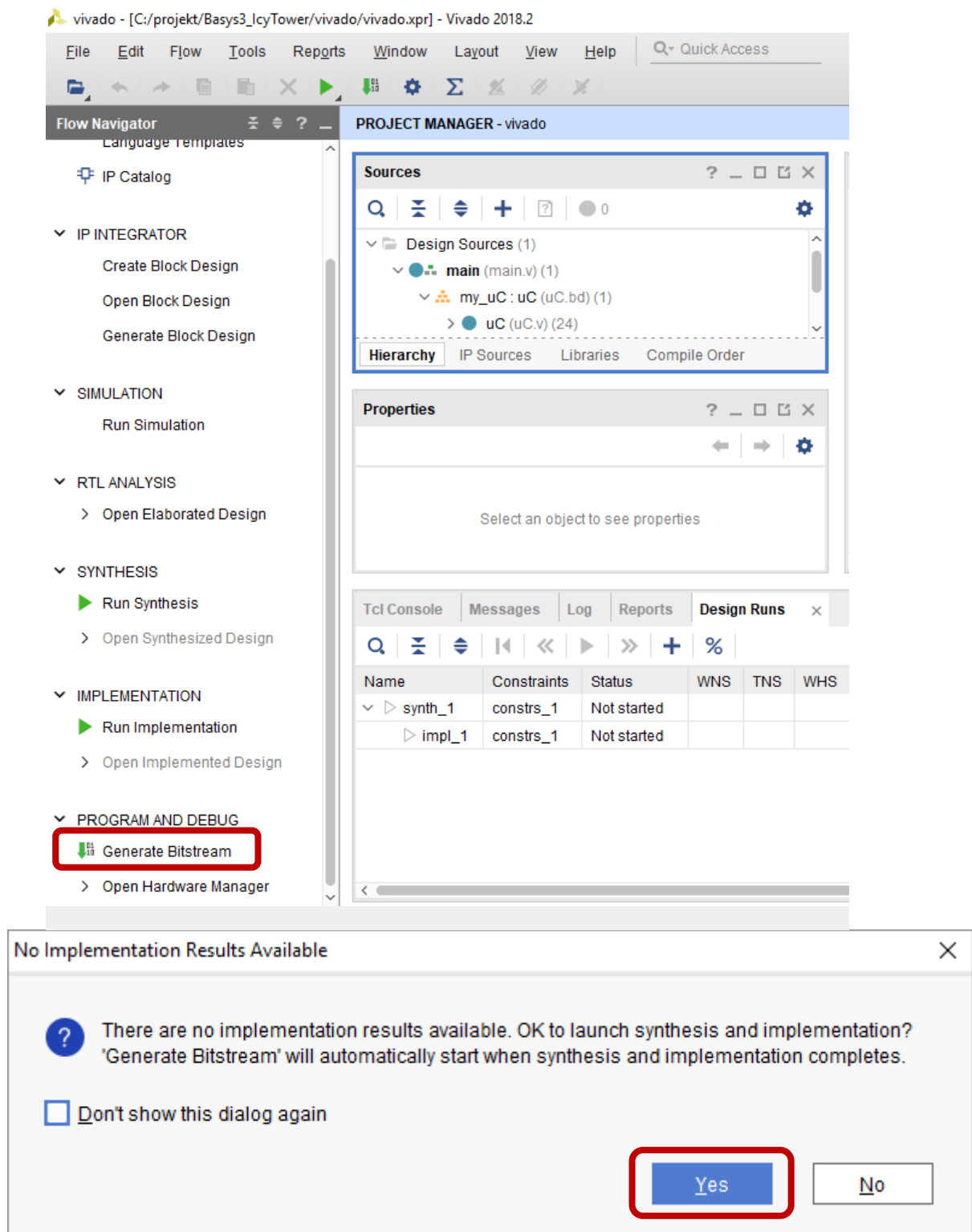


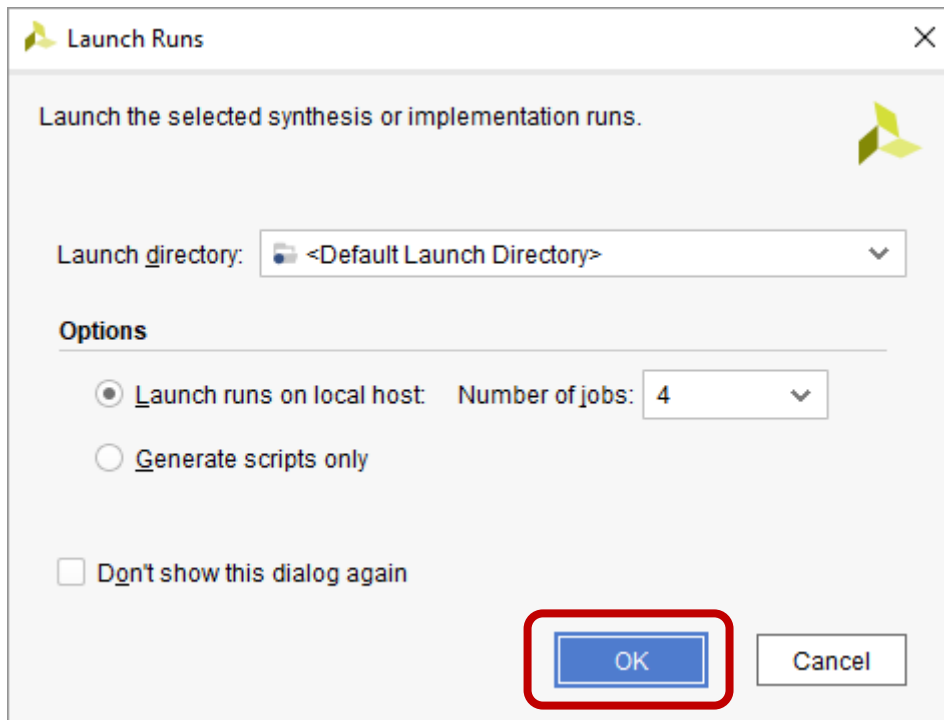
```
MINGW64:/c/projekt
wojte@DESKTOP-MKH1C9V MINGW64 /c/projekt
$ git clone https://github.com/NVi5/Basys3_IcyTower.git
```

3. Po sklonowaniu repozytorium przechodzimy do katalogu **./Basys3\_IcyTower/vivado** i otwieramy plik **vivado.xpr**, następnie czekamy na uruchomienie środowiska Vivado.

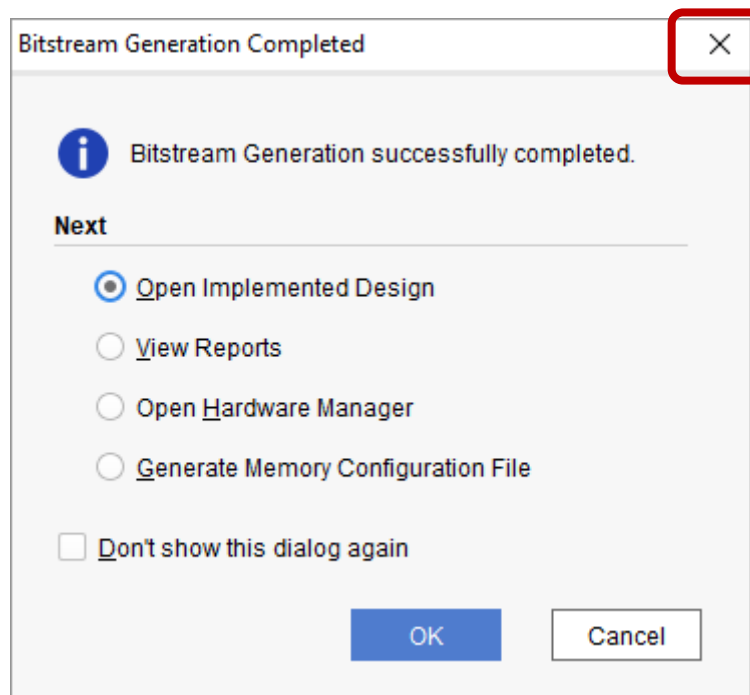


4. Po uruchomieniu Vivado wybieramy opcję Generate Bitstream, z domyślnymi ustawieniami (może to zająć ponad 40 minut).

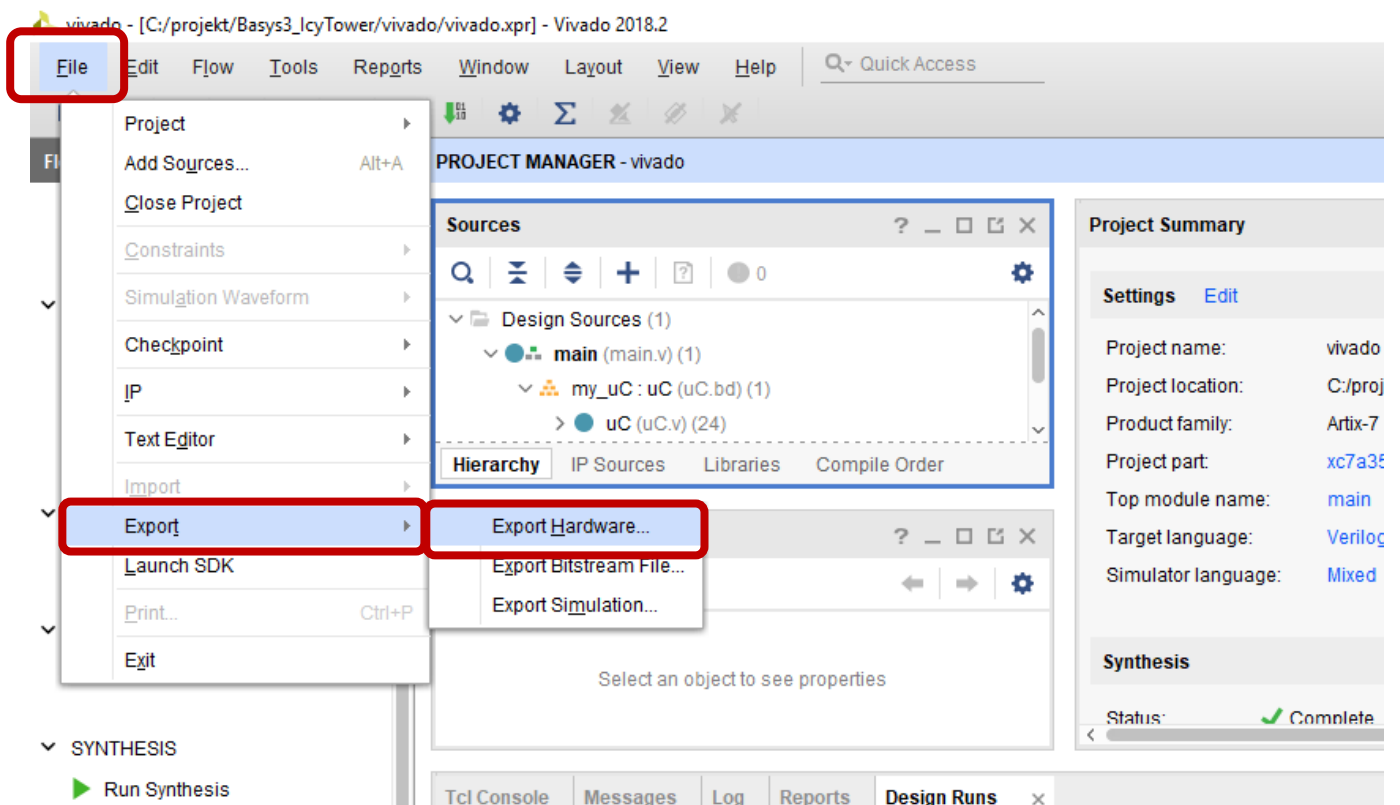




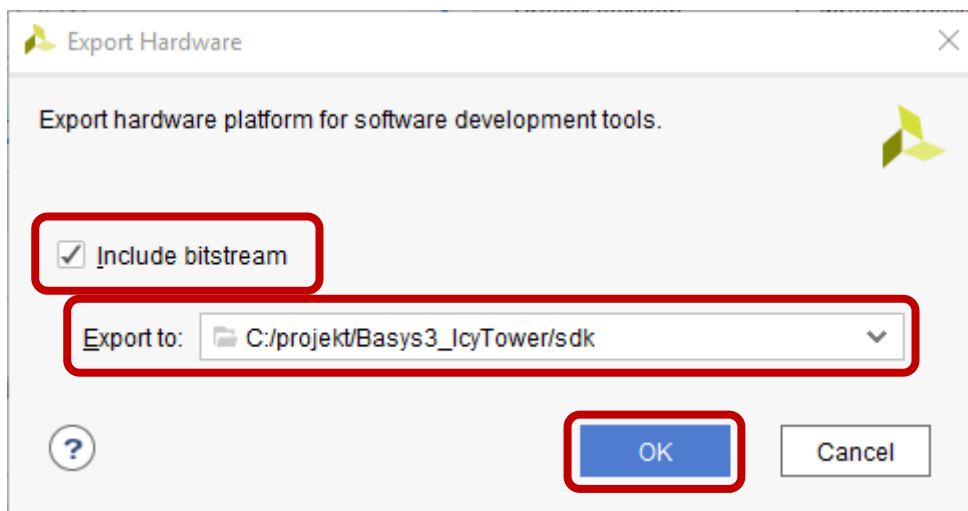
5. Po zakończeniu generowania bitstreamu **zamykamy okno Bitstream Generation Completed.**



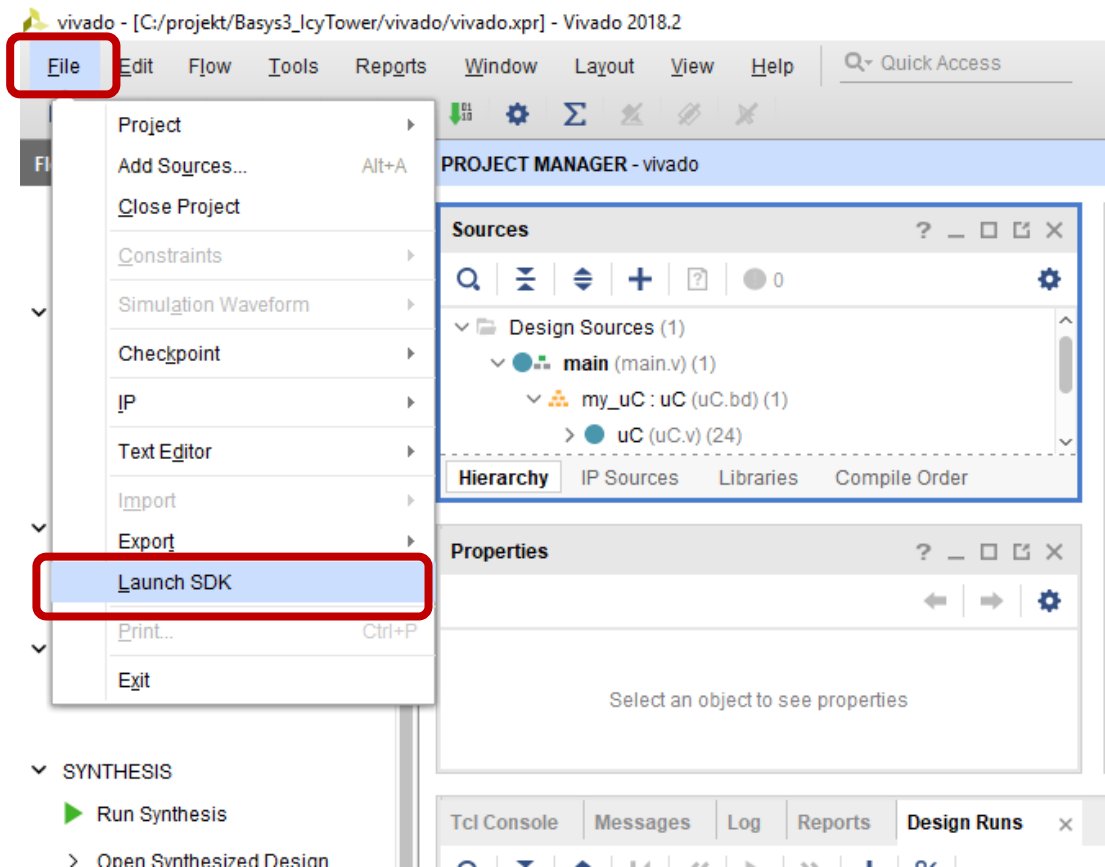
6. W Vivado wybieramy opcję **File > Export > Export hardware.**



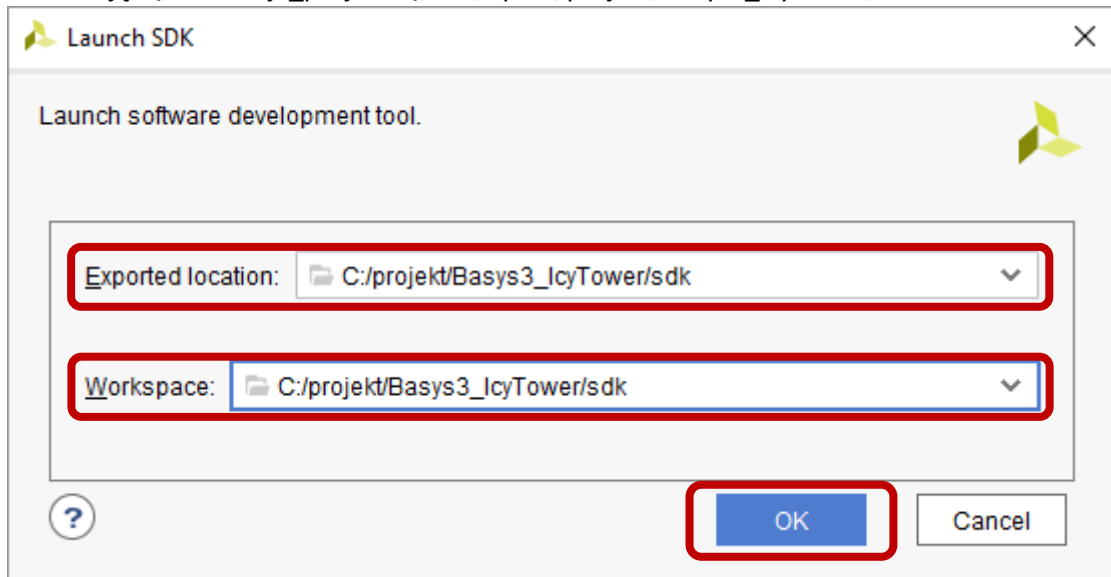
7. W oknie, które się pojawi wybieramy opcję **Include bitstream** i w polu **Export to:** wybieramy Chose Location i wskazujemy folder `${Lokalizacja_projektu }/sdk`, np. `C:/projekt/Basys3_IcyTower/sdk`.



8. W vivado wybieramy opcję **File > Launch SDK**.

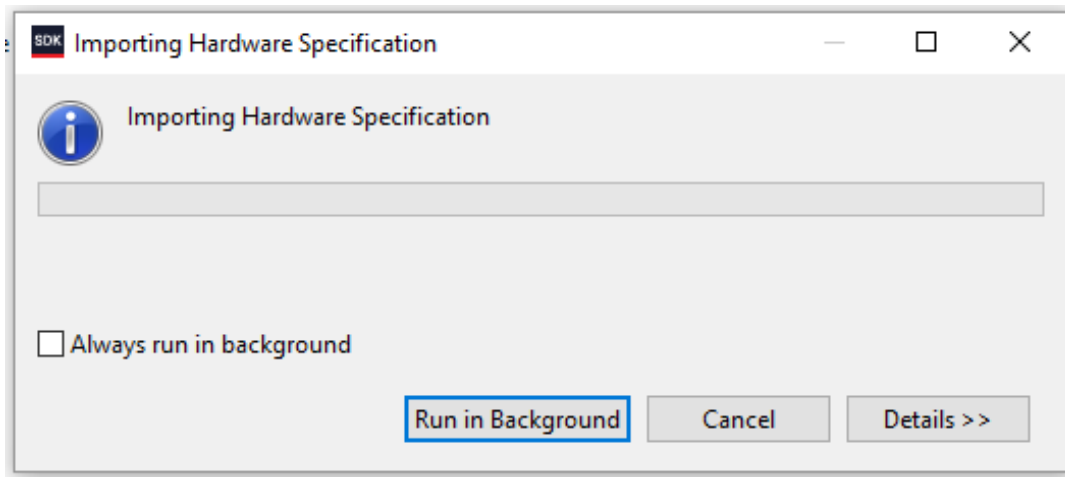


9. W oknie, które się pojawiło w obu przypadkach wybieramy Chose Location i wybieramy lokalizację `${Lokalizacja_projektu}/sdk`, np. `C:/projekt/Basys3_IcyTower/sdk`.

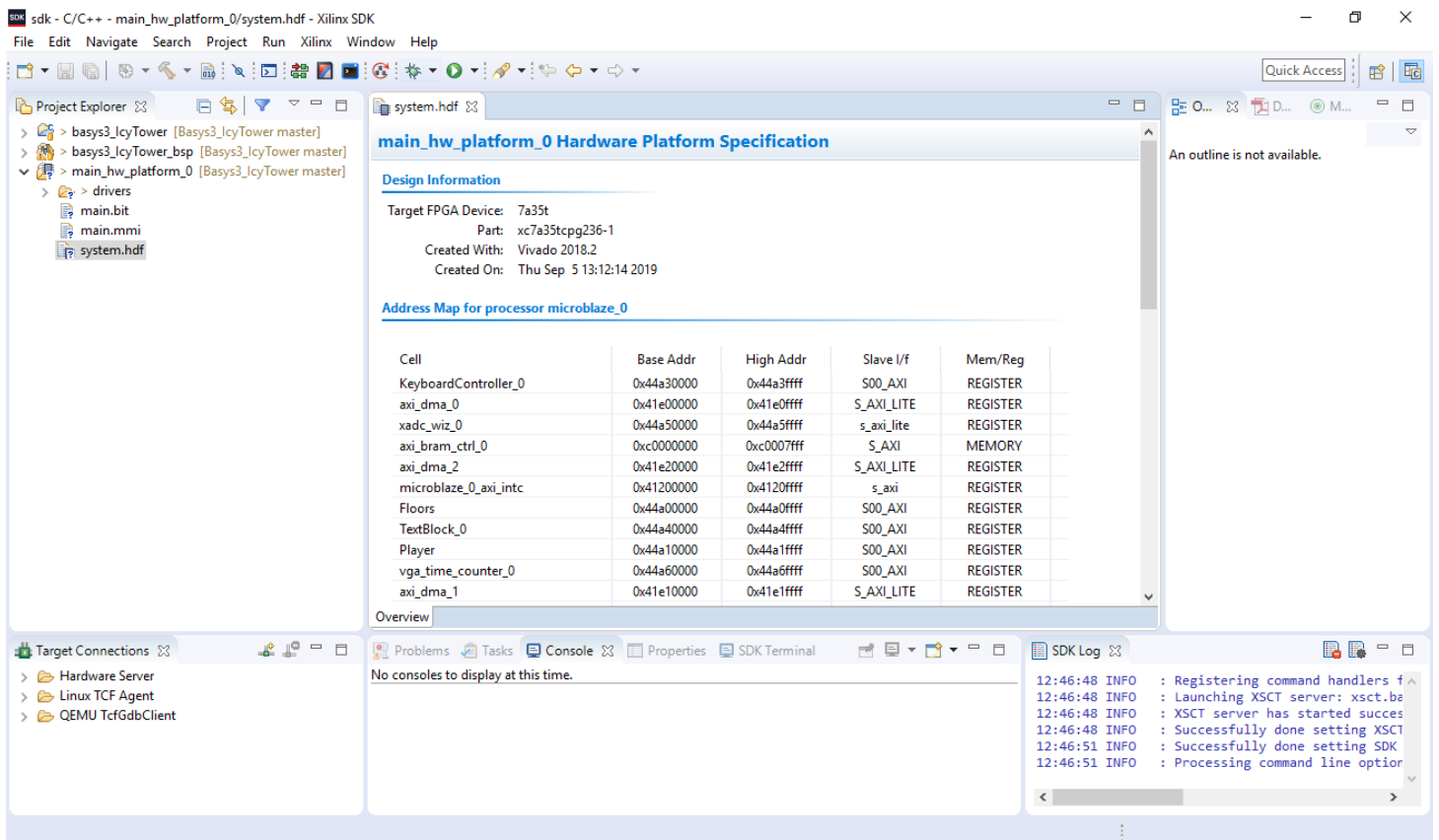


10. Czekamy na załadowanie się SDK.

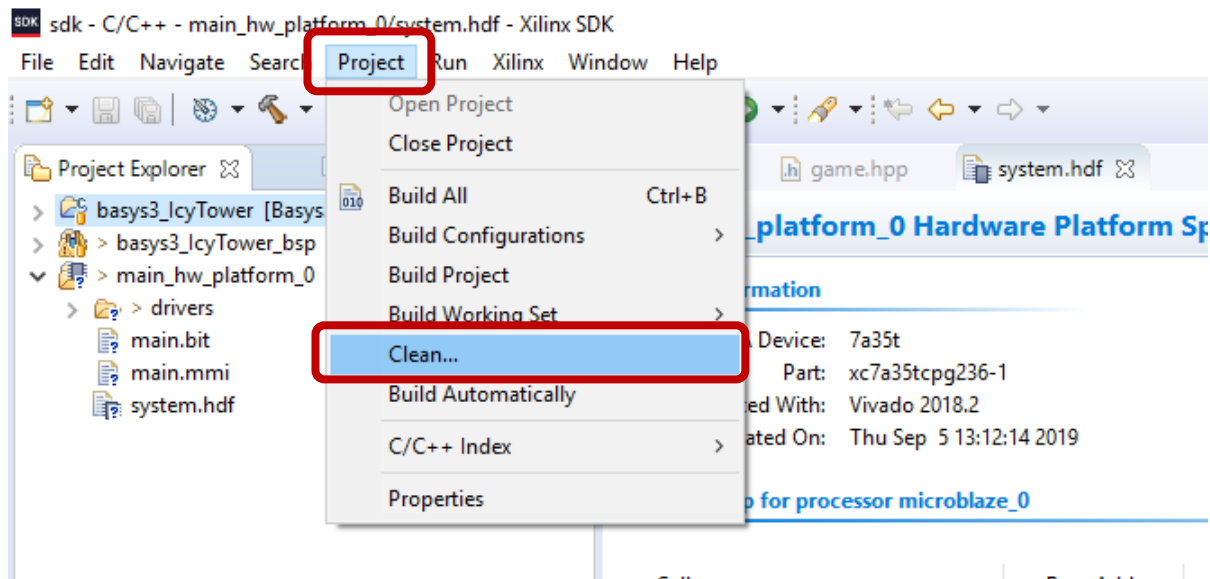
11. Czekamy na pojawienie się okna i zaimportowanie specyfikacji sprzętowej.



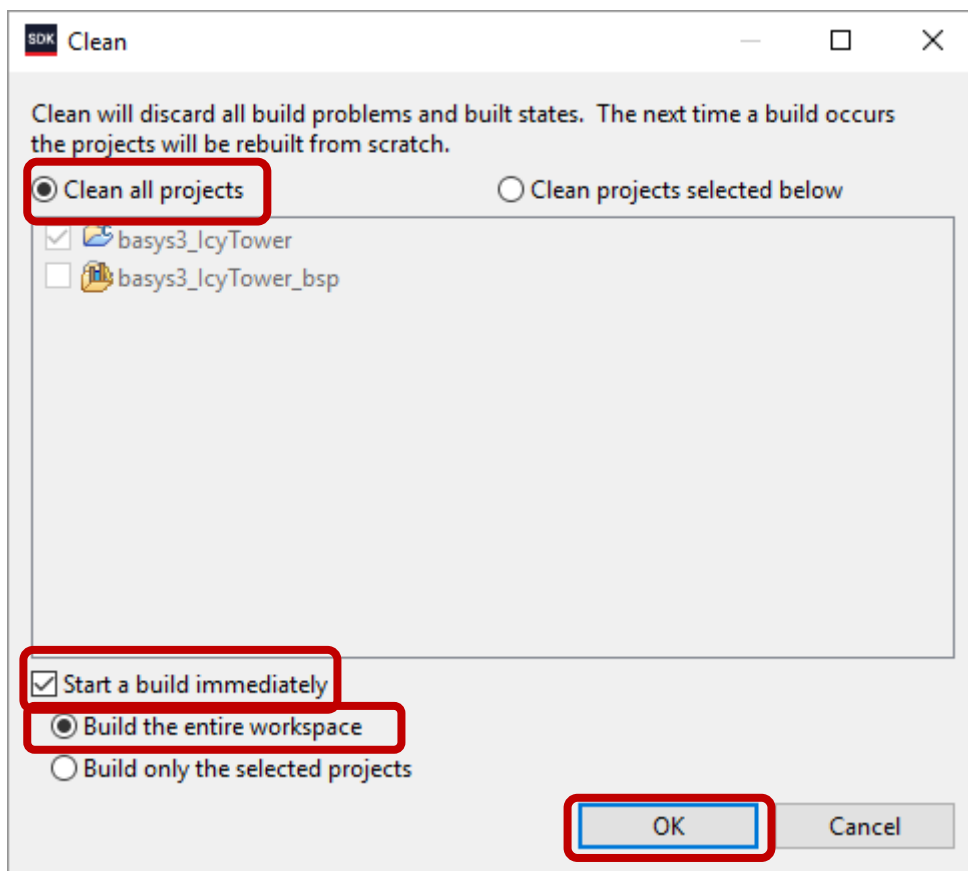
12. Po zakończeniu wszystkich operacji powinniśmy uzyskać taki widok w SDK



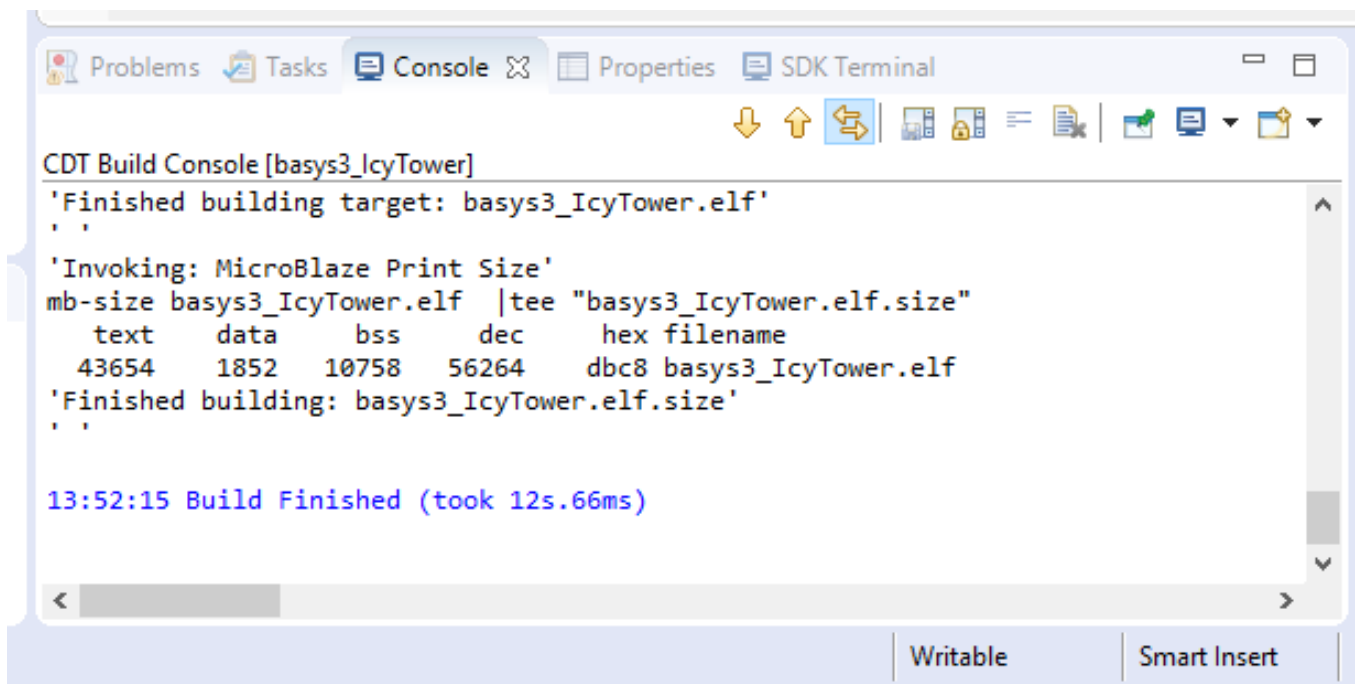
13. Wybieramy opcję **Project > Clean**.



14. W oknie, które się pojawi wybieramy opcje: **Clean all projects**, **Start a build immediately** i **Build the entire workspace**.



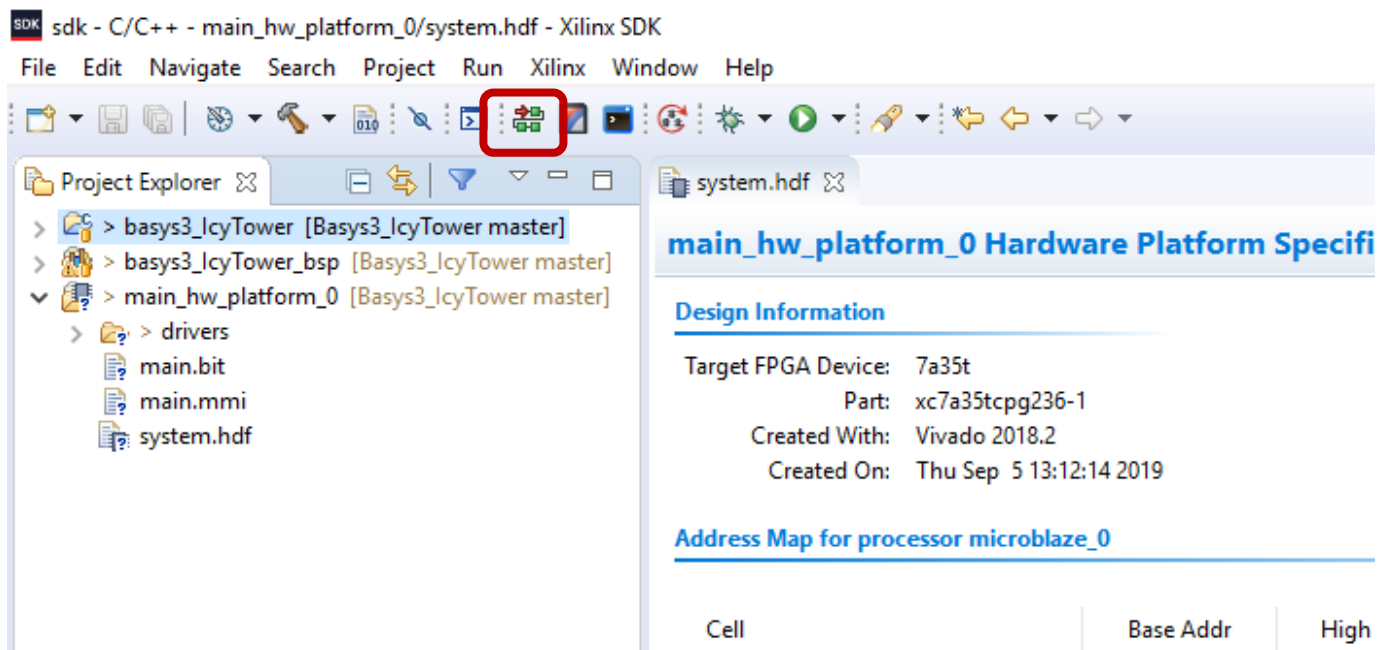
15. Po zakończeniu, w oknie Console powinniśmy zobaczyć następujący komunikat:



```
CDT Build Console [basys3_IcyTower]
'Finished building target: basys3_IcyTower.elf'
'Invoking: MicroBlaze Print Size'
mb-size basys3_IcyTower.elf |tee "basys3_IcyTower.elf.size"
  text    data    bss    dec    hex filename
  43654   1852   10758  56264  dbc8 basys3_IcyTower.elf
'Finished building: basys3_IcyTower.elf.size'

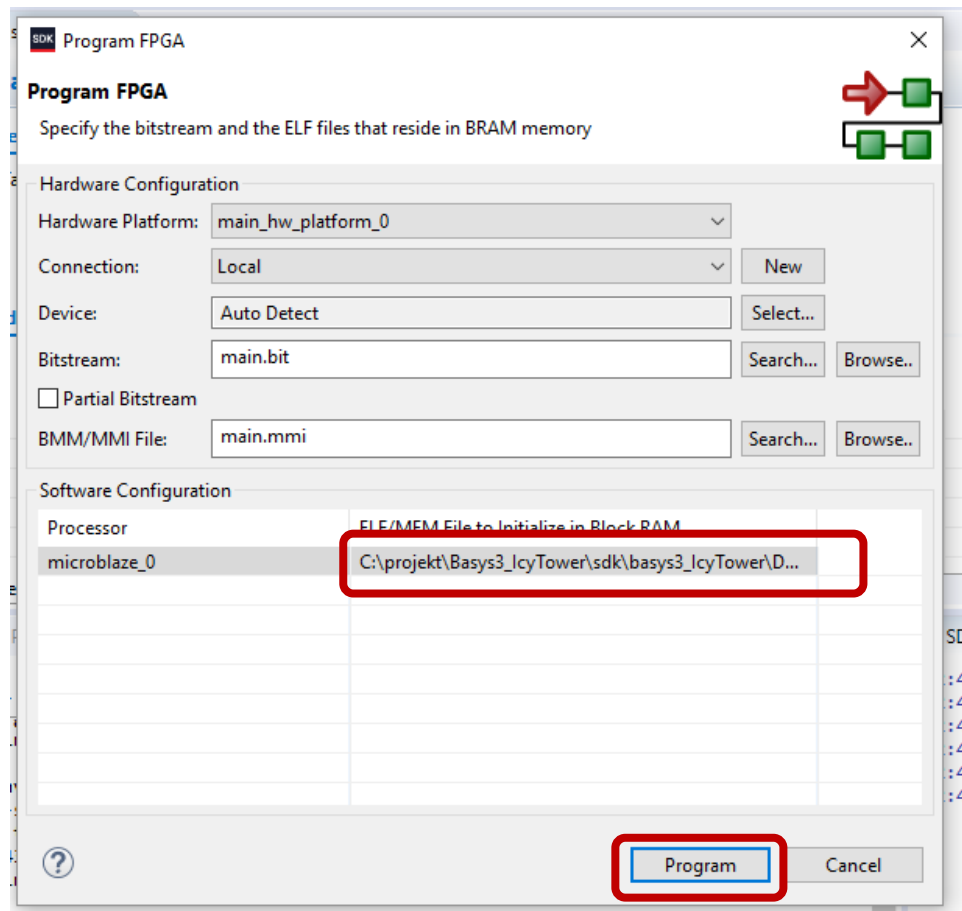
13:52:15 Build Finished (took 12s.66ms)
```

16. W tym momencie podłączamy płytkę Basys 3 do komputera i czekamy na jej wykrycie przez system. Następnie w SDK wybieramy opcję **Program FPGA**.



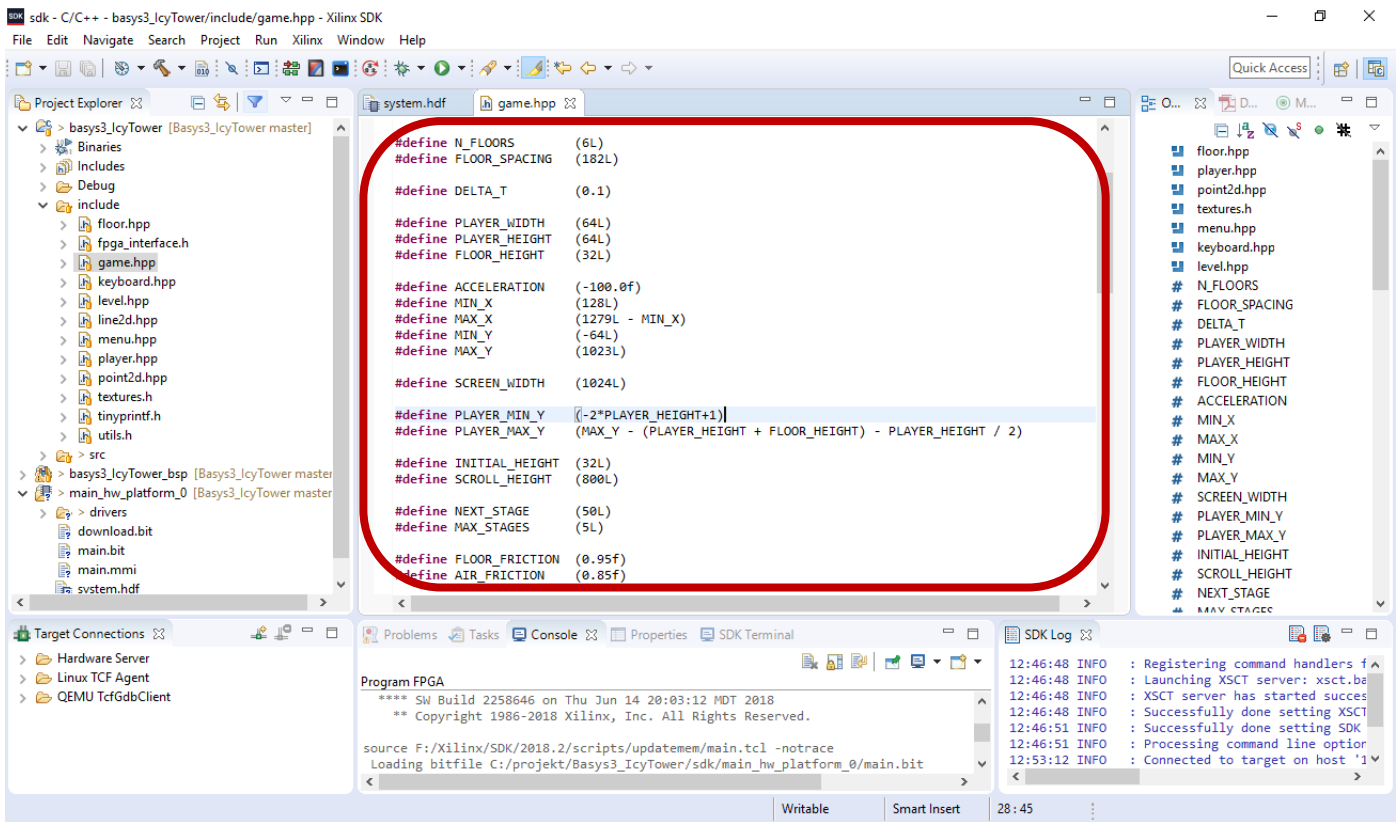


17. W oknie, które się pojawi konfigurujemy opcje tak jak na obrazku poniżej i w polu **EMI/ELF file to initialize in Block Ram** wybieramy wcześniej wygenerowany plik ELF (powinien pojawić się na rozwijanej liście).



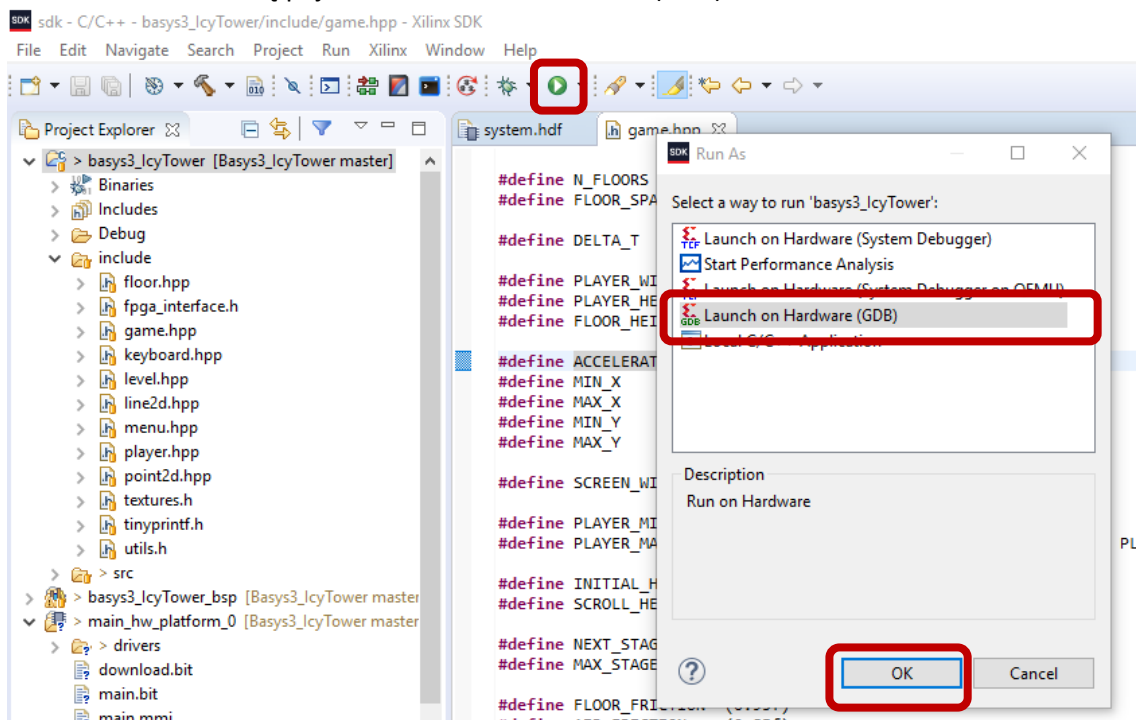
18. Po zakończeniu programowania, Płytką Basys 3 jest gotowa do podłączenia monitora, klawiatury i rozpoczęcia gry.

19. Dodatkowo w oknie projektu możemy otworzyć plik Basys3\_IcyTower/include/game.hpp, i zmieniać parametry gry.



20. Po zakończeniu zmian naciskamy Ctrl + s, aby zapisać i Ctrl + b, aby skompilować projekt.

21. Po skompilowaniu możemy zaktualizować program w pamięci procesora wybierając opcję **Run** i w oknie które się pojawiło **Launch on Hardware** (GDB).

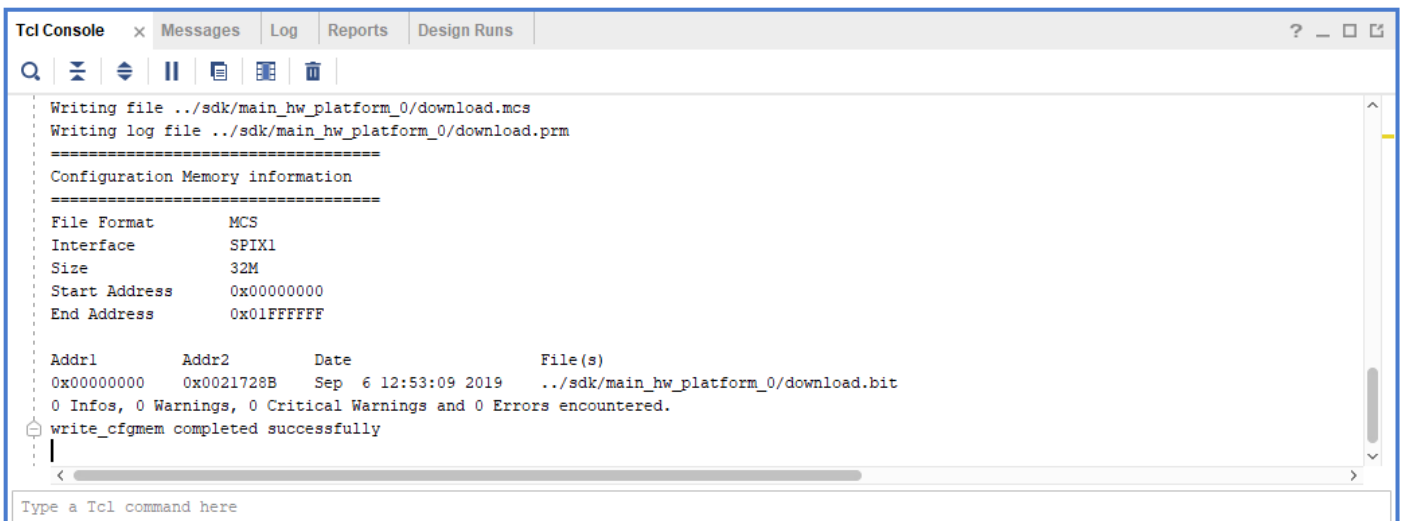


## ***Programowanie pamięci SPI.***

1. Aby zachować konfigurację FPGA i za każdym uruchomieniem płytki Basys 3 nasza gra została załadowana musimy skonfigurować pamięć SPI.  
**Uwaga! – wykonanie poprzedniej części instrukcji jest wymagane do zaprogramowania pamięci.**
2. Przełączamy się na okno Vivado.
3. W oknie Tcl Console wklejamy następującą komendę i wciskamy Enter:

```
write_cfgmem -format mcs -size 32 -interface SPIx1 -loadbit "up 0x0  
../sdk/main_hw_platform_0/download.bit" -file  
../sdk/main_hw_platform_0/download.mcs -force
```

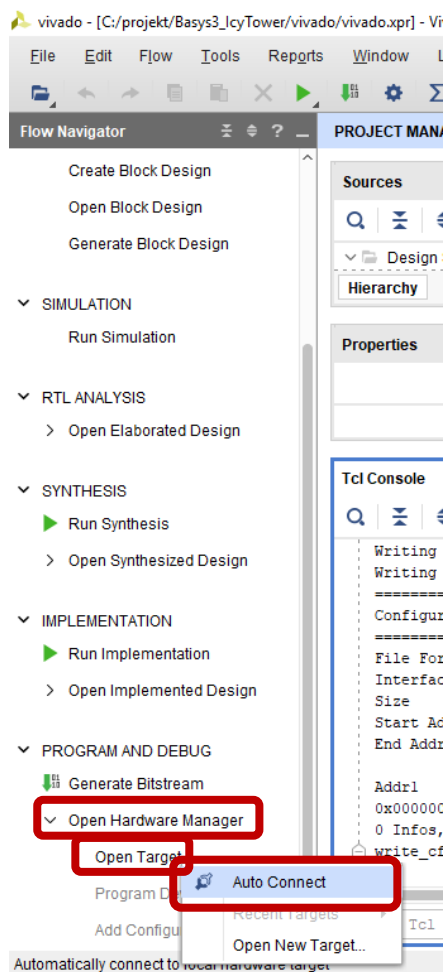
4. Po pozytywnym zakończeniu operacji zobaczymy komunikat:



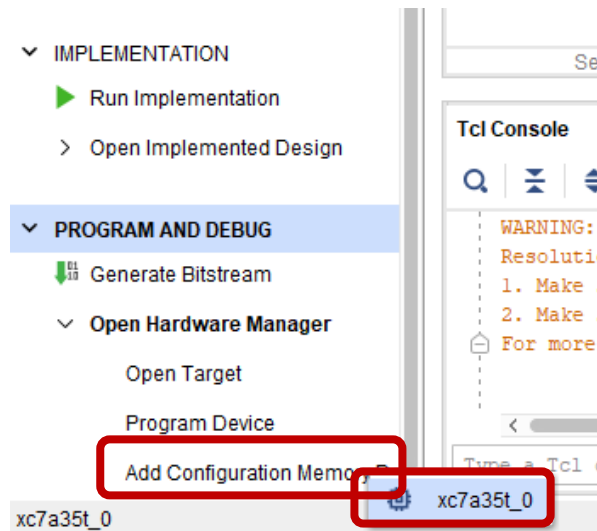
```
Tcl Console x Messages Log Reports Design Runs
Writing file ../sdk/main_hw_platform_0/download.mcs
Writing log file ../sdk/main_hw_platform_0/download.prm
=====
Configuration Memory information
=====
File Format      MCS
Interface       SPIX1
Size            32M
Start Address   0x00000000
End Address     0x01FFFFFF

Addr1      Addr2      Date      File(s)
0x00000000 0x0021728B Sep  6 12:53:09 2019 ../sdk/main_hw_platform_0/download.bit
0 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
write_cfgmem completed successfully
```

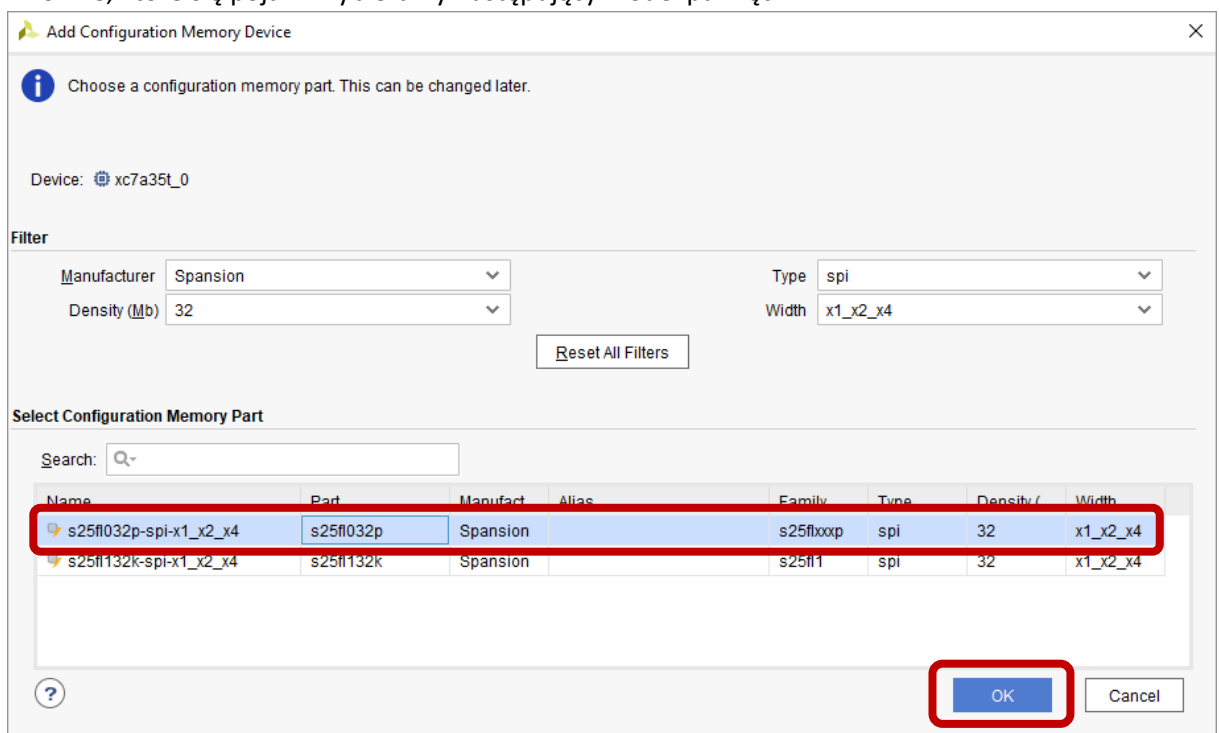
5. W vivado wybieramy opcję **Open Hardware Manager > Open Target > Auto Connect**



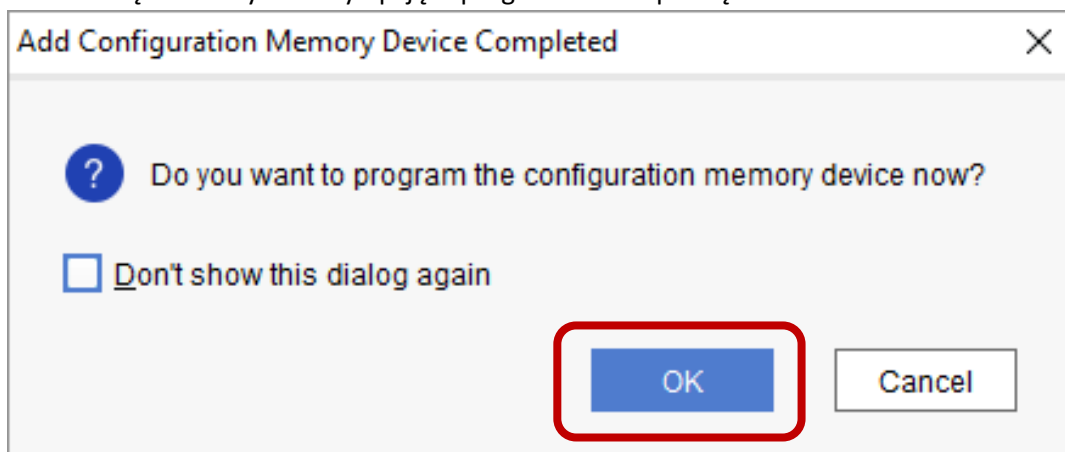
6. Po nawiązaniu połączenia w płytce Basys 3, wybieramy opcję **Add Configuration Memory Device**.



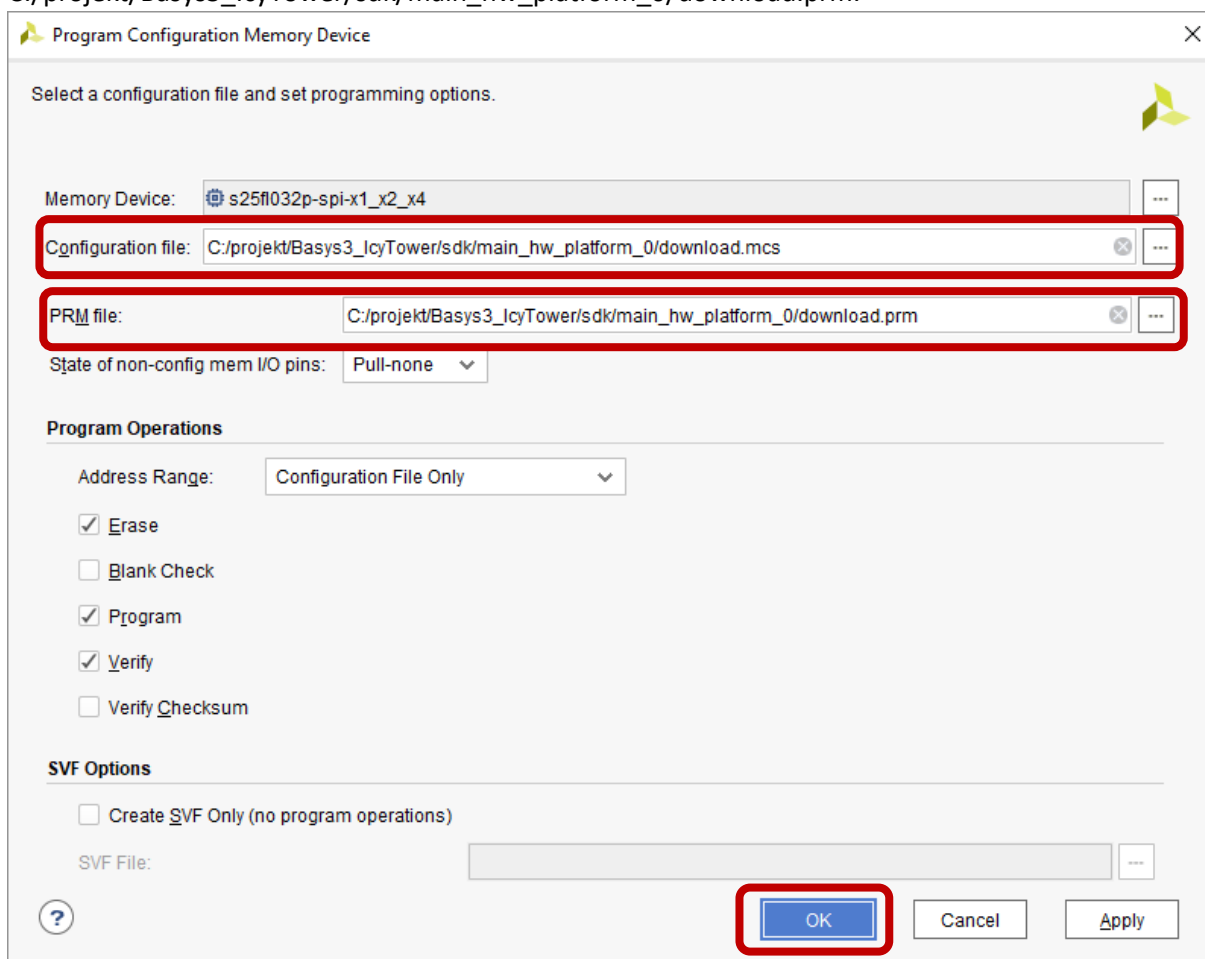
7. W oknie, które się pojawi wybieramy następujący model pamięci



8. Po naciśnięciu OK wybieramy opcję zaprogramowania pamięci



9. W oknie, które się pojawi wybieramy ścieżki dostępu do pliku download.mcs - `${Lokalizacja_projektu }\sdk/main_hw_platform_0/download.mcs`, np. `C:/projekt/Basys3_IcyTower/sdk/main_hw_platform_0/download.mcs`, oraz pliku download.prm - `${Lokalizacja_projektu }\sdk/main_hw_platform_0/download.prm`, np. `C:/projekt/Basys3_IcyTower/sdk/main_hw_platform_0/download.prm`.



10. Po zakończeniu programowania przełączamy zworkę **JP1** na płytce Basys 3 w pozycję **QSPI** i naciskamy przycisk **PROG** znajdujący się obok zworki.