

Tytuł: NORAD-A

Autorzy: Krzysztof Domański (KD), Igor Głowacz (IG)

Ostatnia modyfikacja: 30 sierpnia 2025

Spis treści

1. Repozytorium git	1
2. Wstęp	1
2.1. Czemu grafika wektorowa a nie rastrowa?	1
2.2. Skąd taka tematyka?	1
3. Specyfikacja	2
3.1. Opis ogólny algorytmu	2
3.2. Tabela zdarzeń	2
4. Architektura	3
4.1. Moduł: top rtl	4
4.1.1. Schemat blokowy	4
4.1.2. Porty	6
4.2. Rozprowadzenie sygnału zegara	6
5. Implementacja	7
5.1. Lista zignorowanych ostrzeżeń Vivado	7
5.2. Wykorzystanie zasobów	7
5.3. Marginesy czasowe	7
6. Konfiguracja sprzętu	8
7. Film	8

1. Repozytorium git

Adres repozytorium GITa: <https://github.com/kszdomagh/NORAD-A>¹

2. Wstęp

2.1. Czemu grafika wektorowa a nie rastrowa?

Już od dłuższego czasu bardziej interesowała mnie (KD) grafika wektorowa. Dużą inspiracją do projektu były także albumy muzyczne **Jerobeam'a Fenderson'a**², którego muzyka jest jednocześnie wizualnym pokazem obrazów wektorowych (przez podpięcie prawego oraz lewego kanału audio do oscyloskopu ustawionego w tryb XY). Dużą inspiracją były także zdjęcia, filmy odnośnie systemów wojskowych z lat 70-tych/80-tych³ które do wyświetlania używały właśnie grafiki wektorowej. Dużą pomocą przy tworzeniu projektu okazał się też film **Atari's Quادراسن Explained**⁴ autora *Retro Game Mechanics Explained*, który tłumaczy podstawy działania wektorowych wyświetlaczy dla konsol firmy Atari.

2.2. Skąd taka tematyka?

Bardzo dużą inspiracją była gra **DEFCON**⁵, a podczas ostatnich tygodni wykonywania projektu w sieci pojawiły się filmy z playtestów gry **Air Defence RADAR Simulator Game**⁶, która swoją tematyką zbliżona jest do naszej gry.

Filmy **Doktor Strangelove, czyli jak przestałem się martwić i pokochałem bombę** oraz **Fail Safe** także okazały się dużą inspiracją w trakcie tworzenia gry.

¹ repozytorium publiczne

² Jerobeam Fenderson - kanał YT: <https://www.youtube.com/channel/UCECl4aNz5hvuRzW5fgCOHKQ>

³ Grafiki takich systemów wojskowych znajdują się w README.md projektu na GitHub'ie

⁴ Film Atari's Quادراسن Explained: <https://youtu.be/smStEPSRKBs?si=ftE6p6bxmUlJ-sXg>

⁵ DEFCON gameplay: <https://youtu.be/GYYzCz19GbI?si=EsPSkeTq7oJ24e3cy>

⁶ RADAR Simulator Game: strona internetowa: <https://airdefendergame.com/>

3. Specyfikacja

3.1. Opis ogólny algorytmu

Gra nie posiada jednego centralnego modułu sterującego rozgrywką - cały design zbudowany jest w oparciu o moduły kontrolujące trzech przeciwników, stany trzech baz oraz moduł kierowania kursorem. Moduły te otrzymując sygnały z innych modułów oraz wejść ustalają swój stan według algorytmu pokazanego poniżej.

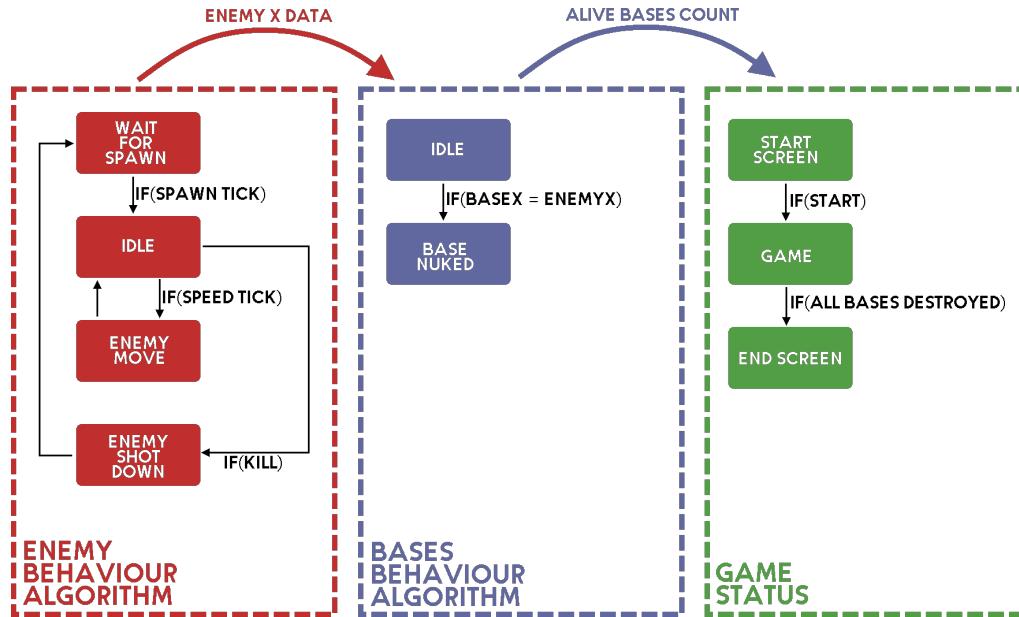


Fig. 1. Uproszczony algorytm przebiegu gry

Wyświetlanie działa niezależnie w stosunku do przebiegu gry.

3.2. Tabela zdarzeń

Zdarzenie	Kategoria	Reakcja systemu
btnU naciśnięty	Kontrola Kursorem	Ruch kurSORA w góRĘ
btnD naciśnięty	Kontrola Kursorem	Ruch kurSORA w dól
btnL naciśnięty	Kontrola Kursorem	Ruch kurSORA w lewo
btnR naciśnięty	Kontrola Kursorem	Ruch kurSORA w prawo
btnC naciśnięty	Kontrola Ogniem	Oddano strzał
Koordynaty kurSORA oraz jednego z przeciwników mieszczą się w zadeklarowanych tolerancjach gdy oddano strzał	Kontrola Ogniem	Dany przeciwnik jest niszczony oraz killcount jest inkrementowany o jeden.
Moduł enemy control otrzymuje sygnał spawn tick	Przeciwnicy	Jeżeli przeciwnik nie jest na ekranie to pojawia się po prawej stronie ekranu
Moduł enemy control otrzymuje sygnał spawn tick	Przeciwnicy	Jeżeli przeciwnik nie na ekranie to pojawia się po jego prawej stronie
Moduł enemy control otrzymuje sygnał speed tick	Przeciwnicy	Jeżeli przeciwnik jest na ekranie to porusza się o LSB DAC'a w lewo
Moduł enemy control otrzymuje sygnał base nuked	Przeciwnicy	Baza została zniszczona - moduł wyłącza się po zniszczeniu lub wylęcieniu samolotu poza ekran
Przeciwnik wylatuje poza ekran po lewej stronie base nuked	Przeciwnicy	Aby to się stało baza musiała zostać zniszczona - moduł wyłącza się.
X przeciwnika zrównuje się z X danej bazy	Bazy	Baza zostaje zniszczona oraz wystawia sygnał base nuked
Wszystkie bazy zostają zniszczone	Bazy	Następuje koniec rozgrywki.

Tabela 1. Tabela zdarzeń - rozgrywka

Zdarzenie	Kategoria	Reakcja systemu
Przełącznik SW0 podniesiony do góry	Kontrola wyświetlania	Następuje reset całego systemu
Przełącznik SW15 podniesiony do góry	Kontrola wyświetlania	Pokazana jest mapa USA - ekran gry.
Przełącznik SW15 opuszczony w dół	Kontrola wyświetlania	Pokazany jest ekran startowy.
Przełącznik SW14 podniesiony do góry	Kontrola wyświetlania	Pokazane jest logo kierunku MTM.
Wszystkie bazy zostają zniszczone	Kontrola wyświetlania	Pokazany jest ekran GAME OVER.
Żaden z przełączników nie jest podniesiony	Kontrola wyświetlania	Pokazany jest ekran startowy.

Tabela 2. Tabela zdarzeń - wyświetlanie

4. Architektura

Architektura systemu srowadza się na rozdzieleniu części odpowiedzialnej za logikę gry (moduł **game logic**), elementu wpisującego do pamięci (moduł **memory manage**) oraz elementu rysującego/odczytującego z pamięci (moduł **vector display**).

Żeby nie komplikować systemu zdecydowano się na użycie tylko jednego modułu RAM do którego najpierw wpisuje się dane wektorowe, następnie wyświetla się je, potem znów wpisuje... - jedynym minusem takiego rozwiązania jest okres bezczynności lampy oscyloskopowej w momencie wpisywania danych. W obecnym kształcie projektu okres ten jest jednak niezauważalny - pomiary wykazały, że wyświetlanie jednej klatki zajmuje maksymalnie 3 ms - "ekran" odświeża się z częstotliwością ponad 300 Hz!

Dane wektorowe należało także w jakichś sposób zapisać oraz odczytywać. Zdecydowano się przyjęcie następującej konwencji, używanej przy zapisie do pamięci przez moduł **memory_manage** jak i w czasie odczytu przez moduł **vector_display**:

```
MSB                                     LSB
18BIT: {8BIT X COORDINATE1, 8BIT Y COORDINATE1, 1BIT POS1 SIGNAL, 1BIT LINE1 SIGNAL}
18BIT: {8BIT X COORDINATE2, 8BIT Y COORDINATE2, 1BIT POS2 SIGNAL, 1BIT LINE2 SIGNAL}
18BIT: {8BIT X COORDINATE3, 8BIT Y COORDINATE3, 1BIT POS3 SIGNAL, 1BIT LINE3 SIGNAL}
...
```

gdzie:

- punkt zapisany jest przez dwa ośmiobitowe koordynaty;
- gdy sygnał POS zapisany jest jako 1: rysowany jest punkt w podanych koordynatach.
- gdy sygnał LINE zapisany jest jako 1: rysowana jest linia od poprzedniego punktu do punktu w podanych koordynatach.
- gdy sygnał LINE oraz POS zapisane są jako 1: jest to sygnał resetu - oznacza albo koniec rysowanego obiektu (w przypadku odczytu z pamięci ROM) lub koniec rysowanej klatki (w przypadku odczytu z pamięci RAM).

W celu ułatwienia pracy przy grafice wektorowej stworzono zestaw narzędzi do wyświetlania oraz generowania kodu dla pamięci ROM. Narzędzia te dostępne są w podlinkowanym repozytorium:

<https://github.com/kszdomagh/vector-display-development-tools.git>⁷

⁷ repozytorium publiczne

4.1. Moduł: top_rtl

Osoba odpowiedzialna: KD, IG

Moduł top posiada dwa większe pod-moduły:

- vector display - odpowiada za wyświetlanie oraz validację sygnałów wyjściowych
- game logic top - odpowiada w całości za przebieg gry

4.1.1. Schemat blokowy

Poniższy schemat blokowy przedstawia główne moduły znajdujące się w module **top rtl** (oprócz modułu **cursor manage**, który znajduje się w module **top basys3** ze względu na wejściowe sygnały będące powiązane z fizycznymi przyciskami na płytce). Kolory poszczególnych modułów oznaczają jakim zegarem są taktowane:

- **czerwony**: moduł taktowany jest zegarem szybkim: $80MHz$
- **niebieski**: moduł jest w pełni kombinacyjny⁸
- **zielony**: moduł taktowany jest zegarem wolnym: $5MHz$

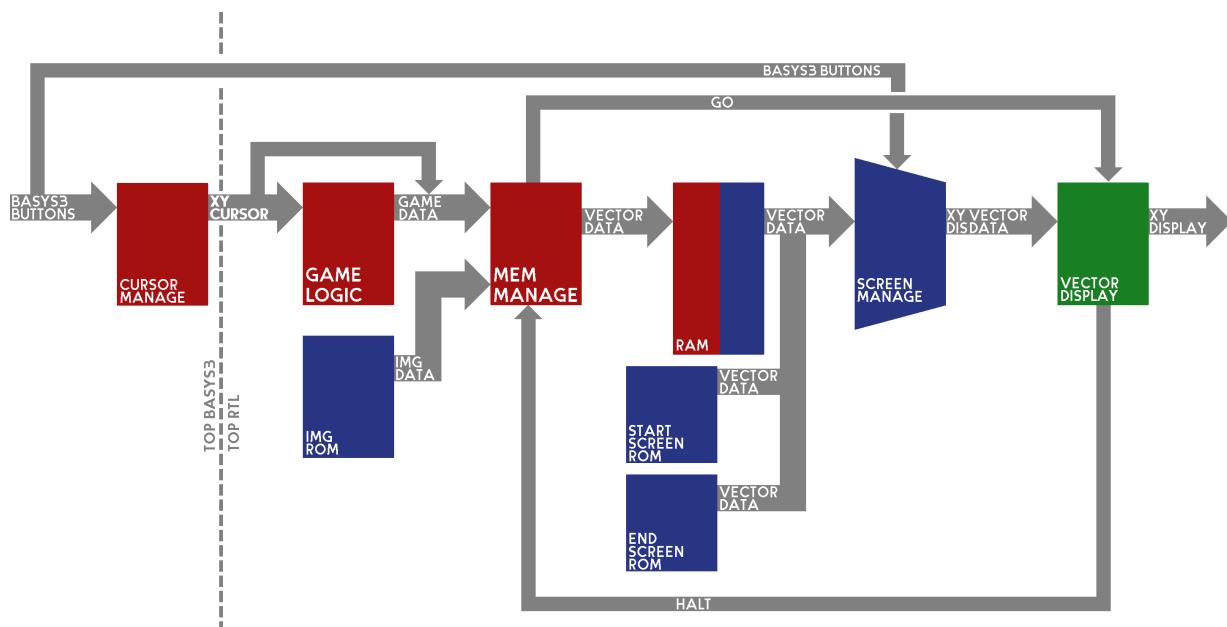
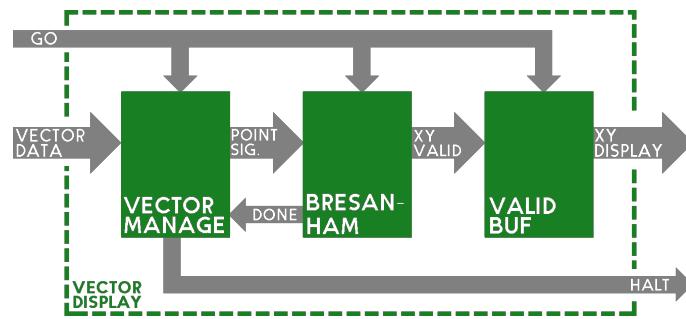


Fig. 2. Schemat blokowy modułu **top rtl**

Moduł top składa się z następujących modułów/podmodułów:

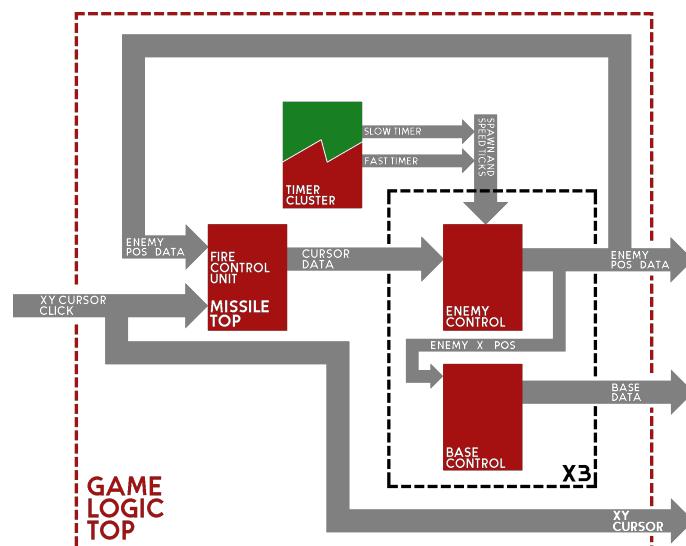
1. **cursor_manage**: zajmuje się konwersją stanu przycisków na pozycję XY kur索ra.
2. podmoduł **game_logic**: zajmuje się kontrolą obiektów w grze; przeciwników, baz oraz interakcjami między nimi.
3. **memory_manage**: na podstawie danych dostarczonych od modułu **game_logic** oraz grafik zapisanych w **img_rom** wpisuje grafiki wektorowe obiektów do modułu **RAM**. Po wpisaniu danych dla jednej klatki wysyła sygnał **GO** do podmodułu **vector_display**, który rysuje dane umieszczone w **RAM**-ie.
4. **img_rom**: zawiera wszystkie używane w grze grafiki wektorowe.
5. **ram**: pamięć, używana jako miejsce zapisu danych dla **memory_manage** oraz do odczytu dla **vector_display**.
6. **start_screen_rom**: zawiera grafikę wyświetlaną na początku rozgrywki.
7. **end_screen_rom**: zawiera grafikę wyświetlaną na koniec rozgrywki.
8. **screen_manage**: multiplekser; w zależności od stanu przycisków wyświetla albo ekran startowy, ekran końcowy lub ekran rozgrywki.
9. podmoduł **vector_display**: odczytuje dane z pamięci **ram** oraz rysuje je za pomocą dwóch sygnałów wyjściowych (x oraz y). Kiedy napotka koniec pamięci wysyła sygnał **HALT**.

⁸ UWAGA DO MODUŁU RAM: odczyt z modułu **ram** jest w pełni kombinacyjny; zapis do modułu **ram** jest taktowany zegarem szybkim $80MHz$

Fig. 3. Schemat blokowy podmodułu `vector_display`

Moduł `vector_display` składa się z następujących modułów:

- vector_manage**: zarządza całym procesem rysowania. Przygotowuje dane z pamięci dla modułu rysującego **bresanham** oraz czeka na ich narysowanie przed podaniem kolejnej paczki danych.
- bresanham**: wykorzystuje algorytm bresanham'a do rysowania linii od punktu do punktu.
- valid_buf**: sprawdza czy dane z modułu **bresanham** są poprawne i czy wynikają z procesu rysowania. Jeżeli nie to przytrzymuje poprzednie dane na wyjściu. Zapobiega to "drżeniu" sygnałów wyjściowych x oraz y; filzuje się dane które są wynikiem wewnętrznych obliczeń modułu do rysowania linii.

Fig. 4. Schemat blokowy podmodułu `game_logic_top`

Moduł `game_logic_top` składa się z następujących modułów:

- fire_control_unit** oraz **missile_top**: zarządzają danymi otrzymywanyymi z kurSORA oraz przycisków - w przypadku wystrzelenia oraz trafienia przeciwnika rakietą moduły wysyłają sygnał do modułu **enemy control** odpowiadającemu przeciwnikowi oraz inkrementują licznik *killcount*, który wyświetlany jest na wyświetlaczu 7-segmentowym.
- timer_cluster**: zparametryzowany zespół układów czasowych (timerów) - dostarcza rozsynchonizowanych sygnałów służących do generowania sygnałów *spawn tick* oraz *speed tick* które służą do pojawiania się przeciwników oraz poruszania nimi na ekranie. Brak synchronizacji między timerami wprowadza efekt pseudolosowości — zdarzenia są deterministyczne, ale ich rozkład czasowy wydaje się dla użytkownika losowy.
- trzy moduły **enemy_control**: kontroluje ruch oraz pojawianie się przeciwnika w oparciu o sygnały *spawn_tick* oraz *speed_tick*. W momencie zniszczenia bazy moduł wyłącza się.
- trzy moduły **base_control**: kontroluje czy bazy zostały zniszczone przez przeciwnika czy nie. Wysyła odpowiedni sygnał do modułu rysującego. W momencie zniszczenia wszystkich trzech baz gra kończy się.

Przez ciągłe tworzenie oraz "dobudowywanie" nowych modułów do działającego projektu, a także z racji braku modułów o tych samych wejściach oraz wyjściach⁹ zdecydowano się nie używać interface'ów - były one zbędne. Jedynym obiektem, który działa jako interface (jest re-używany przez kilka modułów oraz zawiera sygnały w tej samej kolejności) jest 18-bitowy sygnał danych wektorowych (`vector data` / `xy vector data`), który opisany został w rozdziale *Architektura*.

⁹ System projektowano jako zbiór niezależnych modułów - patrz rozdział *Opis ogólny algorytmu*.

4.1.2. Porty

a) top_rtl, inputs

nazwa portu	opis
[7:0] xcursor	8-bitowa pozycja x kurSORA
[7:0] ycursor	8-bitowa pozycja y kurSORA
button_click	btnC zostaŁ kliknięty - sygnał wystrzału rakiety
startgame	sygnał od przełącznika sw15 - pokazuje ekran startowy gdy niski, pokazuje główny ekran gry gdy wysoki.
mtm_show	sygnał od przełącznika sw15 - pokazuje logo kierunku MTM gdy wysoki.
show_death	sygnał służący do debugowania - overrides warunki dot. końca gry (generowane przez moduł game_logic oraz wyświetla ekran GAME OVER gdy wysoki).

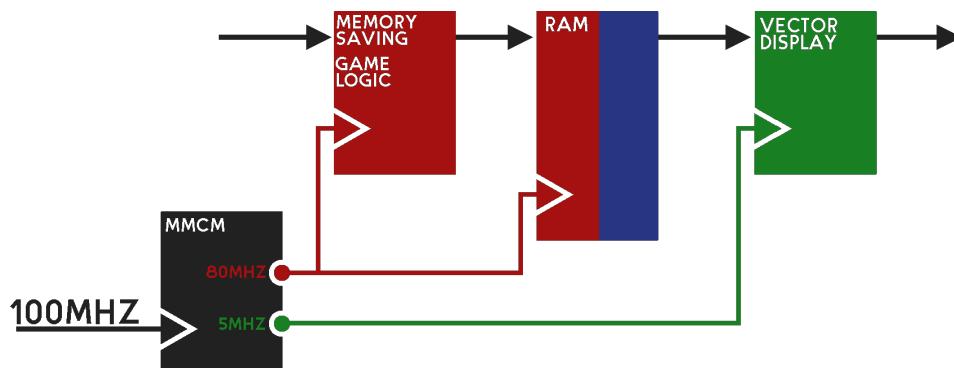
b) top_rtl, outputs

nazwa portu	opis
[7:0] xch	8-bitowy sygnał X dla wyświetlanego obrazu
[7:0] ych	8-bitowy sygnał y dla wyświetlanego obrazu
[7:0] killcount	liczba zestrzelonych samolotów
go_flag	flaga go - wysoka gdy moduł aktualnie jest w fazie rysowania obiektów na oscylkopie.
halt_flag	flaga halt - wysoka gdy moduł aktualnie jest w fazie wpisywania danych do pamięci ROM.

4.2. Rozprowadzenie sygnału zegara

Osoba odpowiedzialna: KD

Od początku tworzenia całego systemu zakładano użycie dwóch zegarów. Część odpowiedzialna za wyświetlanie musiała być taktowana wolnym zegarem (oscylatory analogowe, które docelowo planowano użyć jako wyświetlacze mają dość małe pasmo) oraz pozostałą częścią szybkim zegarem. Moduły o kolorze czerwonym taktowane są zegarem szybkim ($80MHz$), moduły zielone taktowane są zegarem wolnym ($5MHz$)¹⁰. Sygnały zegarowe generowane są przez moduł MMCM wygenerowany przez Vivado Clocking Wizard. Za częstotliwość wejściową przyjmuje się dostepny na płytce zegar $100MHz$.



¹⁰ Vivado Clocking Wizard nie pozwalał na ustalenie zegara wyjściowego modułu MMCM na równo $80MHz$ lub równo $5MHz$. MMCM generuje sygnały zegarowe o częstotliwościach dokładnie $79.558MHz$ oraz $4.980MHz$ o okresach kolejno: $12.549ns$ oraz $200.784ns$

5. Implementacja

5.1. Lista zignorowanych ostrzeżeń Vivado.

Identyfikator ostrzeżenia	Liczba wystąpień	Uzasadnienie
Synth 8-689	1	8-bitowy port sseg posiada w sobie jednobitową informację odnośnie wyświetlanej kropki na wyświetlaczu siedmiosegmentowym. Kropka nie jest używana w projekcie.
Synth 8-7129	2	Moduł <code>valid_buf</code> konwertuje dane z typu <code>unsigned</code> na <code>signed</code> poprzez obcięcie MSB, które nie są do niczego podłączone - jest to zabieg celowy, w projekcie nie pracuje się na ujemnych koordynatach/danych wektorowych.
Synth 8-3936	2	Vivado podczas syntezy początkowo przydzieliło 16-bitowe rejesty adresowe, które następnie zostały zoptymalizowane (obcięte) do faktycznie używanej szerokości 10 bitów. Ostrzeżenie nie wpływa na funkcjonalność modułu.
Synth 8-6430	2	Gdy adresy odczytu oraz zapisu w module RAM będą takie same może dojść do kolizji - moduły które korzystają z pamięci RAM były projektowane aby do tego nie doszło.
Synth 8-7080	1	Projekt jest za mały dla Vivado aby dzielić go na równoległe zadania, cp przypiszyłoby czas komplikacji. Ostrzeżenie nie ma wpływu na działanie programu.
Synth 8-3332	2	Ostrzeżenie dotyczy stanu <code>IDLE</code> w module <code>bresanham</code> , który nie jest bezpośrednio wywoływany w instrukcji <code>case</code> - jego zachowanie zdefiniowane jest w warunku <code>case default</code> .

5.2. Wykorzystanie zasobów

a) Post Synthesis

Zasób	Wykorzystanie (%)
LUT	9.78
FF	2.13
BRAM	1.00
IO	37.74
BUFG	12.50
MMCML	20.00

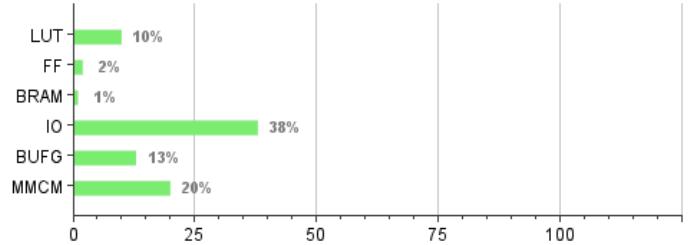


Fig. 5. Wykorzystanie zasobów Post-Synthesis

b) Post-Implementation

Zasób	Wykorzystanie (%)
LUT	9.72
FF	2.13
BRAM	1.00
IO	37.74
BUFG	9.38
MMCML	20.00

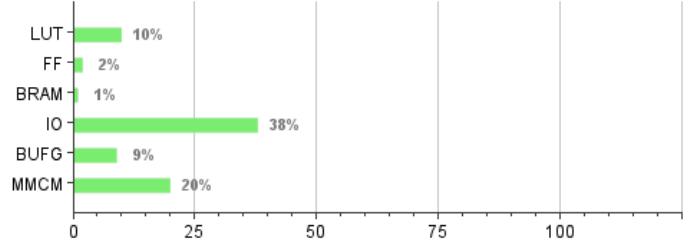


Fig. 6. Wykorzystanie zasobów Post Implementation

5.3. Marginesy czasowe

Aby sprostać wymaganiom timing-owym, w projekcie należało zejść z zegarem części odpowiedzialnej za logikę gry oraz wpisywanie danych wektorowych do pamięci RAM z początkowo zakładanych 100MHz do 80MHz¹¹.

Setup (Worst Slack)	2.272 ns
Hold (Worst Slack)	0.058 ns
Pulse Width (Worst Slack)	3.00 ns

¹¹ Wydawało się to prostszym rozwiązaniem, które nie generowało dużych nakładów pracy w porównaniu z metodą pipeline.

6. Konfiguracja sprzętu

Do gry potrzebny jest oscyloskop z conajmniej dwoma kanałami oraz trybem XY, dwa przetworniki cyfrowo-analogowe podłączone do portów JB oraz JC - kolejność bitów pasująca do wykonanej pracy przetworników podana jest module `top_basys3`.

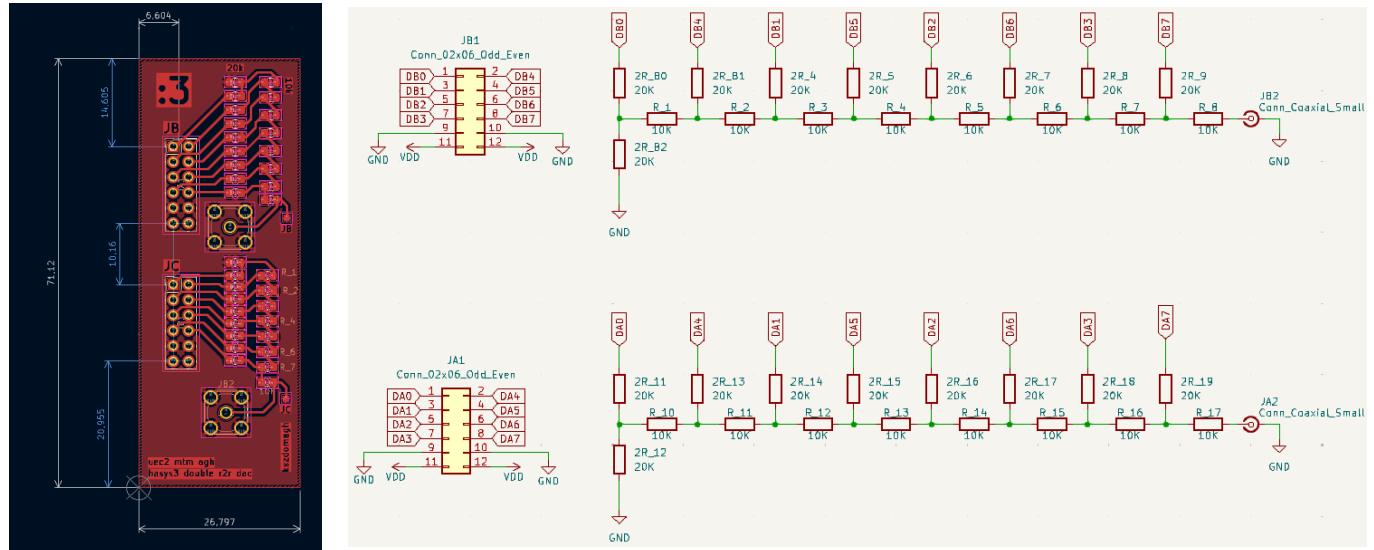


Fig. 7. Footprint oraz schemat pasywnych przetworników cyfrowo-analogowych, użytych w projekcie



Fig. 8. Zdjęcie wykonanej pary przetworników metodami domowymi.

```

1 ...
2 //      new smd DACs with sma connectors
3 .xch( {JB[3], JB[7], JB[2], JB[6], JB[1], JB[5], JB[0], JB[4]} ),
4 .ych( {JC[3], JC[7], JC[2], JC[6], JC[1], JC[5], JC[0], JC[4]} )
5 ...

```

7. Film

Film pokazujący działanie gry znajduje się pod następującym linkiem:

<https://youtu.be/zArHo4Ynv9o?feature=shared>¹²

¹² film publiczny