# Tytuł: **Flappy Bird**

Autorzy: Filip Kazek

Ostatnia modyfikacja: 31.08.2025

## Spis treści



## 1. Repozytorium git

Adres repozytorium GITa:

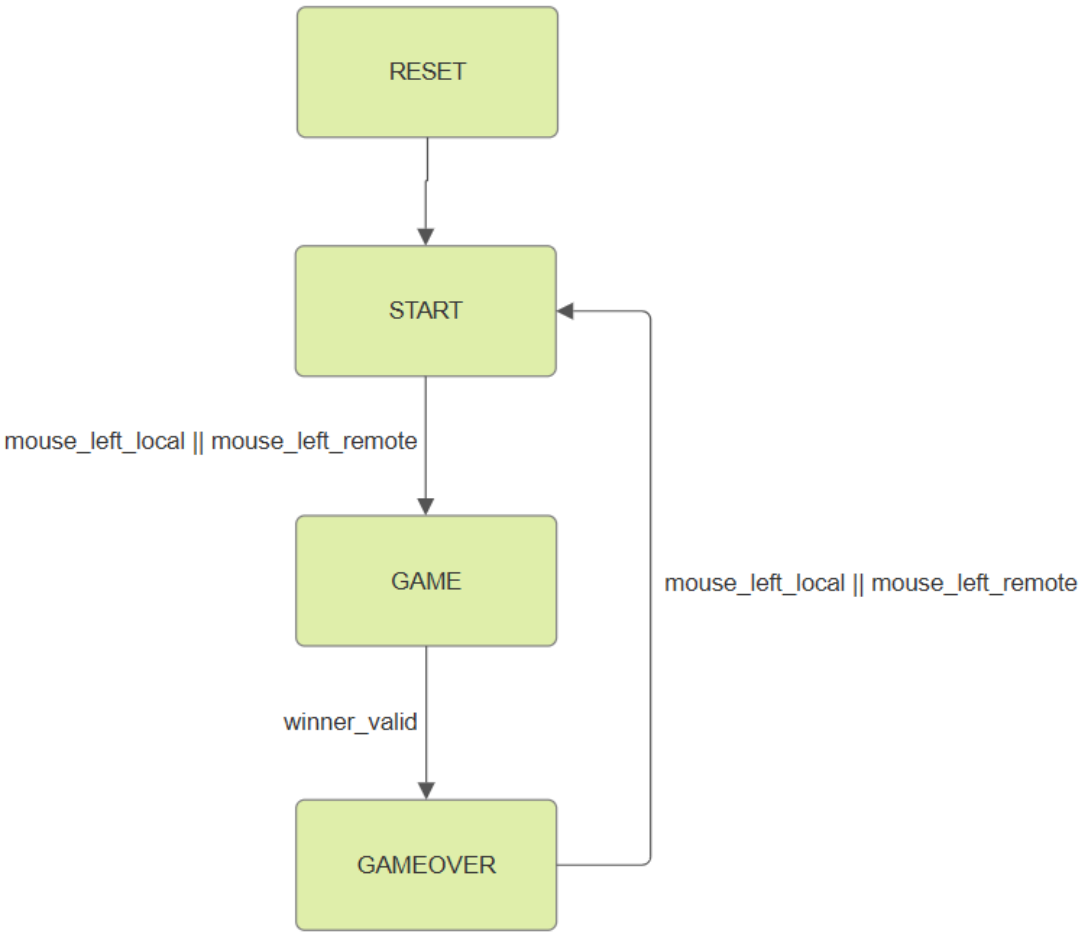
https://https://github.com/filipkazek/flappy_bird

## 2. Wstęp

*11 lat temu wybuchła popularność na ciężką gre Flappy Bird. Niektórzy mówili że jest nierealistyczna i niegrywalna, inni że trzeba być pro-graczem. Więc wracam z wersją dwuosobową, ciut prostszą rozgrywką (ale może dostosowana za pomocą parametrów aby była cięższa). Jako że zaimplementowanie i zrozumienie Microblaze pokonało mnie czasowo, zamiast ptaków scigają się dwie książki, ukochana (heh...) Symofnia C++ i RTL Modeling with SystemVerilog. To wszystko w scenerii naszej pięknej uczelni.*

# Specyfikacja

## 2.1. Opis ogólny algorytmu




**RESET:** FSM ustawia stan początkowy na **START**, zerując winner_latched.

**START:** FSM wyświetla ekran początkowy. Oczekuje na naciśnięcie przycisku (mouse_left_local || mouse_left_remote). Po klincięciu następuje przejście do stanu **GAME** i wygenerowanie game_rst w celu zresetowania logiki gry.

**GAME:** FSM steruje rozgrywką. Kliknięcia graczy (mouse_left_local || mouse_left_remote) przekazywane są do logiki sterującej ptakami.Jeśli winner_valid = 1 następuje przejście do stanu **GAMEOVER.**

**GAMEOVER:** FSM wyświetla ekran końcowy, zależne od winner_latched. Po ponownym (mouse_left_local || mouse_left_remote) następuje przejście do stanu **START.**

### Tabela zdarzeń

| Zdarzenie | Kategoria | Reakcja systemu |
|---|---|---|
| Reset | Ekran startowy | Uruchomienie gry, przejście do START, zerowanie logiki gry, ustawianie rur i ptaków w pozycje startowe. |
| Naciśnięcie przycisku myszy (lokalnej lub zdalnej) w stanie START | Ekran startowy | Przejście do GAME. |
| Kliknięcie myszy w trakcie gry (loklnej lub zdalnej) | Gra/bird_jump | Ptak otrzymuje prędkość skoku (velocity = JUMP_VELOCITY), zaczyna podskok. |
| Brak kliknięcia, upływ tick_cnt = TICK_MAX | Gra/bird_jump | Ptak opada (velocity += GRAVITY) |
| Ptak trafia w sufit lub ziemię | Gra/game_logic | Zwycięzca jest już znany, winner_valid = 1 |
| Ptak trafia w rurę | Gra/game_logic | Czekamy na kolizje drugiego ptaka pending = 1 |
| Oba ptaki uderzają w tą samą rure | Gra/game_logic | Remis, winner_code = 11, przejscie do GAMEOVER |
| Drugi ptak minie rure przy pending = 1 | Gra/game_logic | Zwycięzca ustalony. winner_valid=1 |
| Rury przesuwaja się tick_cnt=TICK_MAX | Gra/tube_render | Zmniejszanie tube[0,1,2] o TUBE_SPEED. Po wyjsicu za ekran, tuba resetowana na prawa strone i losowana nowa wysokość szczeliny. |
| winner_valid = 1 | Ekran końcowy | Przejście do GAMEOVER i rysowanie odpowiedniego napisu zależne od winner_latched. |
| Kliknięcie jednej z myszy | Ekran końcowy | Przejście do START. |

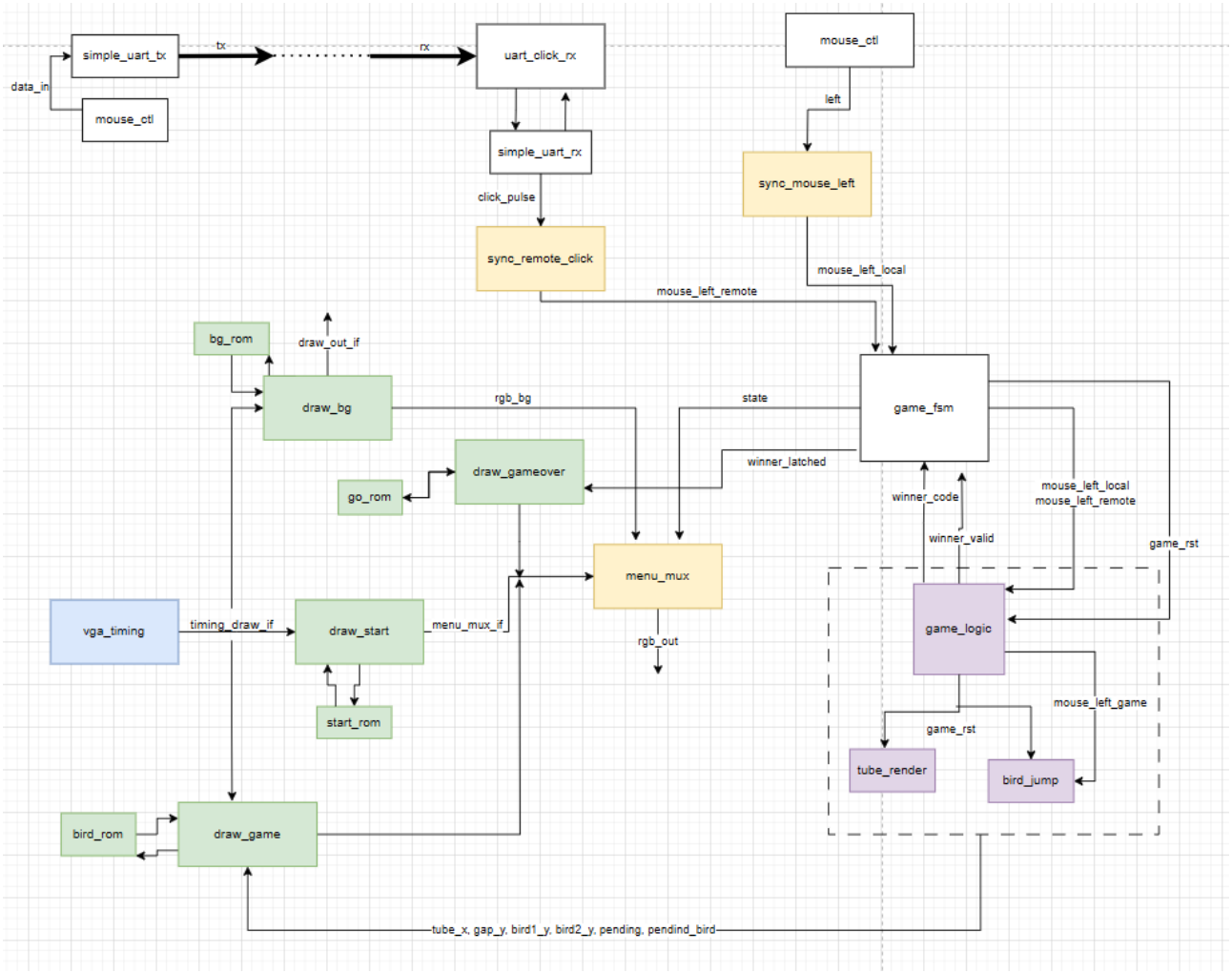
# 3. Architektura

## 3.1. Moduł: top

Osoba odpowiedzialna: Filip Kazek

W module posiadam wszystkie instancje. Komunikacja odbywa się za pomocą sygnałów gry oraz interfejsów VGA oraz RGB. Moduły draw korzystają z danych w pamięci ROM, bądź generują obrazki (tuby). Game_logic odpowiada za wykrywanie kolizji, ruch gracza, natomiast game_fsm kontroluje przebieg rozgrywki.(game_core.bit) Drugi Basys jest jako hub do myszki dla drugiego gracza (p2_hub.bit)

### 3.1.1. Schemat blokowy




### 3.1.2. Porty

#### *a) Mouse*

| nazwa portu | opis |
|---|---|
| PS2Clk | Zegar do komunikacji PS2 |
| PS2Data | Dane o kliknięciu myszki |

#### *b) vga – vga_ctl, output*

| nazwa portu | opis |
|---|---|
| vga_vs | sygnał synchronizacji pionowej VGA |
| vga_hs | Sygnał synchronizacji poziomej VGA |
| vga_r | Sygnał koloru czerwonego VGA |
| vga_g | Sygnał koloru zielonego VGA |
| vga_b | Sygnał koloru niebieskiego VGA |

***c) Piny***

| nazwa portu | opis |
|---|---|
| rx | Sygnał do odbiornika uart na głównym basysie |
| tx | Sygnał z basysa-hub |

### 3.1.3. Interfejsy

***a) rgb_if***

| nazwa sygnału | opis |
|---|---|
| rgb_start[11:0] | Sygnal do muxa do wyswietlania ekranu startowego |
| rgb_game[11:0] | Sygnał do muxa do wyswietlania gry |
| rgb_gameover11:0] | Sygnal do muxa do wyświetlania ekranu końcowego |
| valid_start | Flaga do muxa |
| valid_game | Flaga do muxa |
| valid_gameover | Flaga do muxa |

## 3.2. Rozprowadzenie sygnału zegara

*Używany układ dostarcza zegar o częstotliwości 100MHz. Z clk wizarda używam w całym projekcie 65MHz. Mogłem sobie na to pozwolić w przypadku myszki gdyż używam jej tylko do klikania, co w zupełności wystarcza.*

# 4. Implementacja

## 4.1. Lista zignorowanych ostrzeżeń Vivado.

| Identyfikator ostrzeżenia | Liczba wystąpień | Uzasadnienie |
|---|---|---|
| [Synth 8-7080] | 1 | Projekt jest za mały na równoległą synteze. |
| [Power 33-332] | 1 | Podpięty reset do duzej ilosci modułów nie wpływa negatwynie na działanie programu. |
| [Synth 8-7129] | 20 | Sygnały rysujące nie wykorzystują wszystkich portów z interfesju vga. |

## 4.2. Wykorzystanie zasobów

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 10368 | 20800 | 49.85 |
| FF | 860 | 41600 | 2.07 |
| BRAM | 38 | 50 | 76.00 |
| IO | 20 | 106 | 18.87 |
| BUFG | 2 | 32 | 6.25 |
| MMCM | 1 | 5 | 20.00 |

### Marginesy czasowe

Marginesy czasowe (WNS) dla setup i hold.

| | |
|---|---|
| Worst Negative Slack (WNS): | 0.363 ns |
| Total Negative Slack (TNS): | 0 ns |

| | |
|---|---|
| Worst Hold Slack (WHS): | 0.094 ns |
| Total Hold Slack (THS): | 0 ns |

## 5. Konfiguracja sprzętu

Wymagane połączenie głównego Basysa (top_vga_basys3.bit) złączem JB1 ze złączem JA1 Basysa-Huba(p2_hub.bit). Do każdego podłączyć myszke portem USB.

## 6. Film.

Link do ściągnięcia filmu:

https://drive.google.com/file/d/1Pq1olSknsgejTlglk3GJU1E38zRujOpz/view?pli=1