Tytuł: **Bomberman na MicroBlaze**

Autorzy: **Michał Oleksy (MO), Szymon Galara (SG)**

Ostatnia modyfikacja: 11.09.2019

1. Repozytorium git projektu

Adres repozytorium GITa:

https://github.com/dywanow/uec2_projekt

2. Wstęp

Pomysł wziął się z gry o tej samej nazwie wyprodukowanej w 1983 roku i z wielu klonów dostępnych w Internecie. My chcemy zrobić bardzo uproszczoną wersję. Dostępny będzie tryb dla dwóch graczy bez AI. Mapa będzie zbudowana z trzech elementów – ścieżka, ściana oraz pudełko. Każdy z graczy będzie posiadał bomby, które będą wybuchały po pewnym czasie od ich położenia. Gracze również będą mieli określoną liczbę żyć na rundę. Dla wszystkich obiektów występujących na mapie wykonamy tekstury 64x64. Sterowanie odbędzie się za pomocą klawiatury.

3. Specyfikacja

3.1. Opis ogólny algorytmu

Menu główne

Po zaprogramowaniu układu FPGA oraz procesora Microblaze na ekranie pojawia się menu. Tu do wybory mamy dwie opcje: rozpoczęcie gry lub pokazanie instrukcji. Opcję wybieramy klawiszem Spacja, pomiędzy opcjami przełączamy się klawiszami W i S.

Instrukcja gry

Po wybraniu opcji instrukcji pokazują się klawisze używane w grze oraz ich przeznaczenie dla każdego z graczy. Z ekranu instrukcji można powrócić do menu klawiszem Spacja.

Gra

Po wybraniu opcji rozpoczęcia gry pojawia się plansza zawierająca mapę oraz graczy. Oprócz tego po bokach planszy umieszczone są informacje o stanie graczy. Plansza ma rozmiar 16x16 kratek. Pojedyncza kratka ma wymiary 64x64 px. Na jednej kratce widoczny jest 1 element na raz, jednak może być kilka aktywnych elementów. Gracze mają przypisane ID – załóżmy 1 oraz 2. Graczem 1 steruje się za pomocą klawiszy WASD i Spacja, graczem 2 – 5123 i Enter. Poruszanie odbywa się za pomocą W/A/S/D oraz 5/1/2/3. Gracze poruszają się po kratce. Bomby są podkładane klawiszem Spacja/Enter. Po chwili od położenia bomby następuje eksplozja tworząca ogień. Gracz stojący w ogniu traci życie i jest „teleportowany” do pozycji startowej. Gdy co najmniej jeden z graczy straci wszystkie życia – gra się kończy.

Ekran końcowy

Po zakończeniu gry pojawia się ekran końcowy ukazujący informację o zwycięzcy lub remisie. Z ekranu można wyjść do menu klikając klawisz Spacja.

3.2. Tabela zdarzeń

Zdarzenie	Kategoria	Reakcja systemu
Naciśnięcie klawisza W/S	Menu	Zmiana opcji z menu
Naciśniecie klawisza Spacja na opcji startu gry	Menu	Rozpoczęcie gry – inicjalizacja obiektów areny
Naciśniecie klawisza Spacja na opcji instrukcji	Menu	Pokazanie instrukcji gry
Naciśnięcie klawisza Spacja	Instrukcja gry	Powrót do menu
Naciśnięcie klawisza W	Gra	Przemieszczenie gracza 1 o jedną pozycję w górę*
Naciśnięcie klawisza A	Gra	Przemieszczenie gracza 1 o jedną pozycję w lewo*
Naciśnięcie klawisza S	Gra	Przemieszczenie gracza 1 o jedną pozycję w dół*
Naciśnięcie klawisza D	Gra	Przemieszczenie gracza 1 o jedną pozycję w prawo*
Naciśnięcie klawisza Spacja	Gra	Położenie bomby przez gracza 1**
Naciśnięcie klawisza 5	Gra	Przemieszczenie gracza 2 o jedną pozycję w górę*
Naciśnięcie klawisza 1	Gra	Przemieszczenie gracza 2 o jedną pozycję w lewo*
Naciśnięcie klawisza 2	Gra	Przemieszczenie gracza 2 o jedną pozycję w dół*
Naciśnięcie klawisza 3	Gra	Przemieszczenie gracza 2 o jedną pozycję w prawo*
Naciśnięcie klawisza Enter	Gra	Położenie bomby przez gracza 2**
Przeminięcie czasu oczekiwania na eksplozję bomby	Gra	Usunięcie bomby z planszy, inicjacja eksplozji***
Kolizja eksplozji z pudełkiem	Gra	Usunięcie pudełka z planszy, zatrzymanie rozprzestrzeniania się eksplozji w danym kierunku
Kolizja eksplozji ze ścianą	Gra	Zatrzymanie rozprzestrzeniania się eksplozji w danym kierunku
Kolizja eksplozji z graczem	Gra	Usunięcie gracza z planszy, odjęcie życia, przeniesienie do pozycji startowej, przejście w stan Dead
Kolizja eksplozji z bombą	Gra	Usunięcie bomby z planszy, inicjacja eksplozji***
Przeminięcie czasu przebywania w stanie Dead przez gracza	Gra	Przejście gracza w stan Alive, pojawienie się na ekranie
Przeminięcie czasu aktywności eksplozji	Gra	Usunięcie eksplozji z planszy
Kolizja gracza z bonusową bombą	Gra	Zwiększenie liczby dostępnych bomb gracza o 1, usunięcie bonusowej bomby z planszy

Kolizja gracza z bonusowym życiem	Gra	Zwiększenie liczby żyć gracza o 1, usunięcie bonusowego życia z planszy
Co najmniej jeden z graczy nie ma żyć	Gra	Uruchomienie ekranu końcowego
Naciśnięcie klawisza Spacja	Ekran końcowy	Przejście do menu

*) Gracz może się przemieścić jeśli:

1. znajduje się w stanie Alive,
2. na pozycji docelowej nie ma elementu kolizyjnego, czyli jest ścieżka,
3. minął czas opóźnienia pomiędzy ruchami.

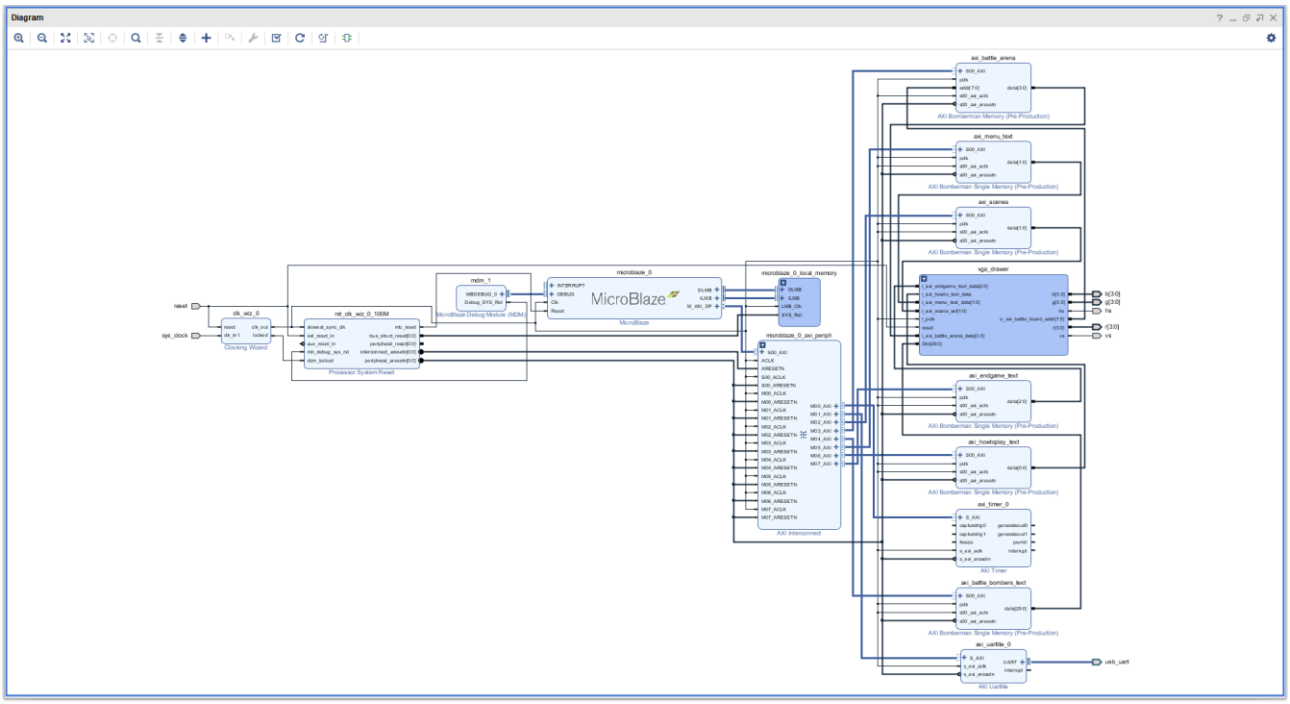
**) Bomba może zostać położona jeśli:

1. gracz ma jeszcze dostępne wolne bomby,
2. na miejscu docelowym nie ma bomby (miejsce docelowe to pozycja gracza kładącego bombę).

***) Eksplozja rozprzestrzenia się w 4 kierunkach i jest zbudowana z ognia (obiekt Fire).

4. Architektura

4.1. Główny schemat blokowy



System składa się z 3 głównych elementów:

- procesora MicroBlaze – na nim uruchamiamy zasadniczy kod gry
- modułów AXI będących pośrednikami komunkacji pomiędzy kodem C++ i sprzętem
- bloku rysującego vga_drawer

4.1.1. Moduł MicroBlaze

Po wstawieniu bloku MicroBlaze do schematu automatycznie dołączany jest moduł debugowania MDM, moduł resetu Processor System Reset oraz blok z pamięcią procesora. Zakres pamięci został ustawiony na 128 KB tak, aby możliwe było zaprogramowanie plikiem .elf (waży on około 90 KB).

4.1.2. Moduł Clocking Wizard

Generuje zegar wyjściowy clk_out o częstotliwości 148.5 MHz konieczny do prawidłowego wyświetlania w częstotliwości Full HD. Na tym zegarze działają wszystkie elementy synchroniczne systemu, włącznie z procesorem MicroBlaze.

4.1.3. Moduł AXI Bomberman Memory

Przechowuje informacje o typie elementu areny na danej pozycji i w zasadzie tylko tutaj ma zastosowanie.

Parametry:

- ADDR_WIDTH – szerokość adresu określa maksymalną liczbę komórek pamięci
- DATA_WIDTH – szerokość danych określa rozmiar pojedynczej komórki (reg) pamięci

Porty (oprócz standardowych AXI):

Port	Szerokość (bity)	Typ	Opis
pclk	1	IN	Na zboczu tego zegara następuje odczyt z pamięci
addr	ADDR_WIDTH	IN	Adres generowany przez moduł zewnętrzny
data	DATA_WIDTH	OUT	Dane odczytane pod adresem addr

a) Instancja axi_battle_arena

Parametr	Wartość
ADDR_WIDTH	8
DATA_WIDTH	4

Zawiera 256 komórek pamięci. Komórka ma rozmiar 3 bitów.

Opis pojedynczej komórki pamięci:

Adres	Możliwe wartości	Opis
3:0	0-8	Kod elementu

Opis kodów elementów:

Kod	Oznaczenie	Element
0	T_PATH	Ścieżka
1	T_OBS1	Pudełko
2	T_OBS2	Ściana
3	T_BOMB	Bomba
4	T_EXPL	Ogień
5	T_PLR1	Gracz 1
6	T_PLR2	Gracz 2
7	T_BOBO	Bonusowa bomba
8	T_BOLI	Bonusowe życie

4.1.4. Moduł AXI Bomberman Single Memory

Jest to praktycznie ten sam moduł co AXI Bomberman Memory, lecz tylko z 1 komórką pamięci. Jest stosowany we wszystkich pozostałych miejscach gdzie występuje komunikacja software – hardware.

Parametry:

- DATA_WIDTH – szerokość danych określa rozmiar pojedynczej komórki (reg) pamięci

Porty (oprócz standardowych AXI):

Port	Szerokość (bity)	Typ	Opis
pclk	1	IN	Na zboczu tego zegara następuje odczyt z pamięci
data	DATA_WIDTH	OUT	Dane odczytane z pamięci

a) Instancja axi_scenes

Parametr	Wartość
DATA_WIDTH	2

Przechowuje kod aktualnej sceny.

Opis pamięci:

Adres	Możliwe wartości	Opis
1:0	0-3	Kod sceny

Opis kodów scen:

Kod	Oznaczenie	Scena
0	MENU_ID	Menu
1	BATTLE_ID	Battle
2	ENDGAME_ID	Koniec gry
3	HOWTOPLAY_ID	Instrukcja gry

b) Instancja axi_menu_text

Parametr	Wartość
DATA_WIDTH	2

Przechowuje stan tła opcji.

Opis pamięci:

Adres	Możliwe wartości	Opis
0	0/1	Opcja: start gry 0 – kolor tła normalny 1 – kolor tła jaśniejszy
1	0/1	Opcja: instrukcja gry 0 – kolor tła normalny 1 – kolor tła jaśniejszy

c) Instancja axi_howtoplay_text

Parametr	Wartość
DATA_WIDTH	1

Przechowuje stan tła opcji.

Opis pamięci:

Adres	Możliwe wartości	Opis
0	0/1	Opcja: powrót do menu 0 – kolor tła normalny 1 – kolor tła jaśniejszy

d) Instancja axi_endgame_text

Parametr	Wartość
DATA_WIDTH	3

Przechowuje stan mrugania opcji oraz informacje na temat wyniku gry.

Opis pamięci:

Adres	Możliwe wartości	Opis
0	0/1	Opcja: przejście do menu 0 – kolor tła normalny 1 – kolor tła jaśniejszy
2:1	1-3	Kod wyniku gry

Opis kodów wyniku gry:

Kod	Oznaczenie	Opis
0	-	Stan teoretycznie niedozwolony, ma taki sam efekt jak DRAW
1	BOMBER1	Wygrywa gracz 1
2	BOMBER2	Wygrywa gracz 2
3	DRAW	Remis

e) Instancja axi_battle_bombers_text

Parametr	Wartość
DATA_WIDTH	30

Przechowuje dane do wyświetlenia przez moduł draw_bomber_text, przy czym w jednej pamięci są informacje dla obydwu graczy.

Opis pamięci:

Adres	Oznaczenie	Możliwe wartości	Opis
0	POS_X_DIG1_1	0/1	Starsza cyfra pozycji X gracza 1
4:1	POS_X_DIG0_1	0-9	Młodsza cyfra pozycji X gracza 1
5	POS_Y_DIG1_1	0/1	Starsza cyfra pozycji Y gracza 1
9:6	POS_Y_DIG0_1	0-9	Młodsza cyfra pozycji Y gracza 1
12:10	LIVES_1	0-4	Ilość żyć gracza 1
14:13	BOMBS_1	0-3	Ilość wolnych bomb gracza 1
15	POS_X_DIG1_2	0/1	Starsza cyfra pozycji X gracza 2
19:16	POS_X_DIG0_2	0-9	Młodsza cyfra pozycji X gracza 2
20	POS_Y_DIG1_2	0/1	Starsza cyfra pozycji Y gracza 2
24:21	POS_Y_DIG0_2	0-9	Młodsza cyfra pozycji Y gracza 2
27:25	LIVES_2	0-4	Ilość żyć gracza 2
29:28	BOMBS_2	0-3	Ilość wolnych bomb gracza 2

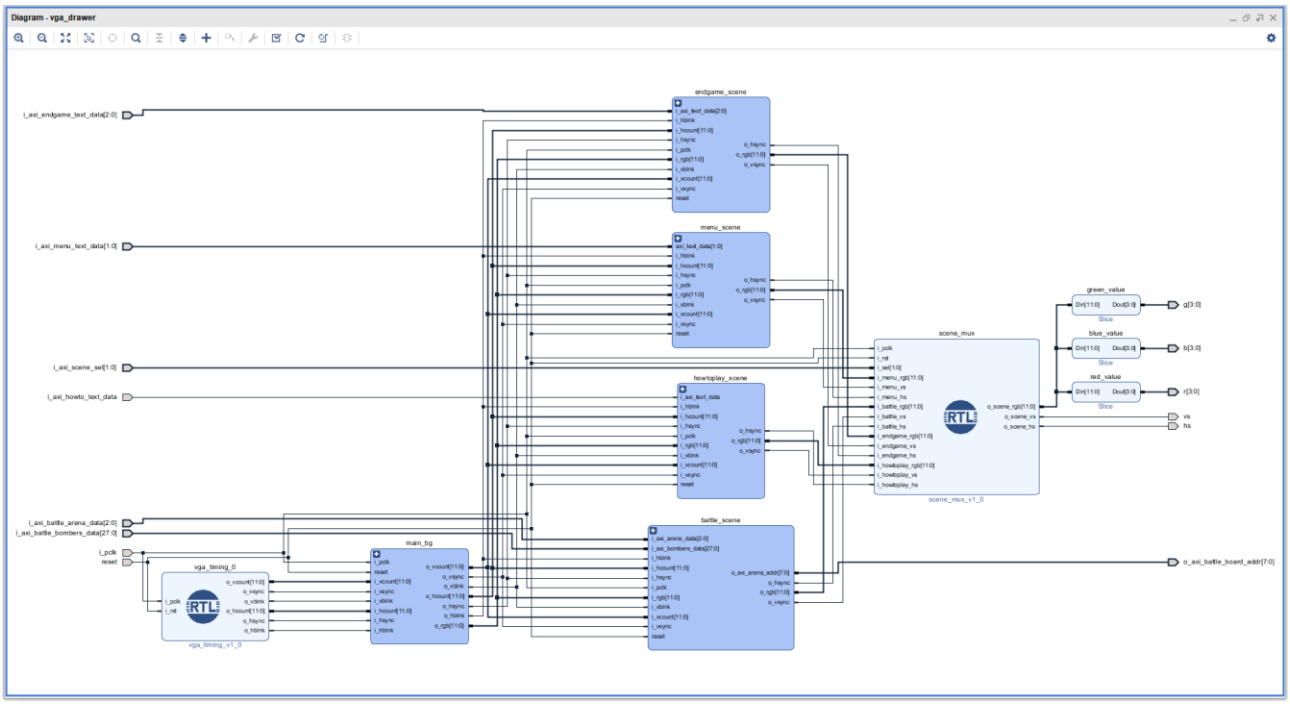
4.1.5. Moduł AXI Timer

Zapewnia możliwość zarządzania czasem w grze – ruchy gracza, detonacja bomb, migający tekst itd.

4.1.6. Moduł AXI Uartlite

Dzięki niemu sterowanie odbywa się przez klawiaturę.

4.2. Blok vga_drawer



Blok składa się z modułu generującego sygnały VGA, bloku rysującego tło główne (main_bg), bloków rysujących poszczególne sceny (menu_scene, howtoplay_scene, battle_scene, endgame_scene) oraz multipleksera wybierającego sygnały aktywnej sceny.

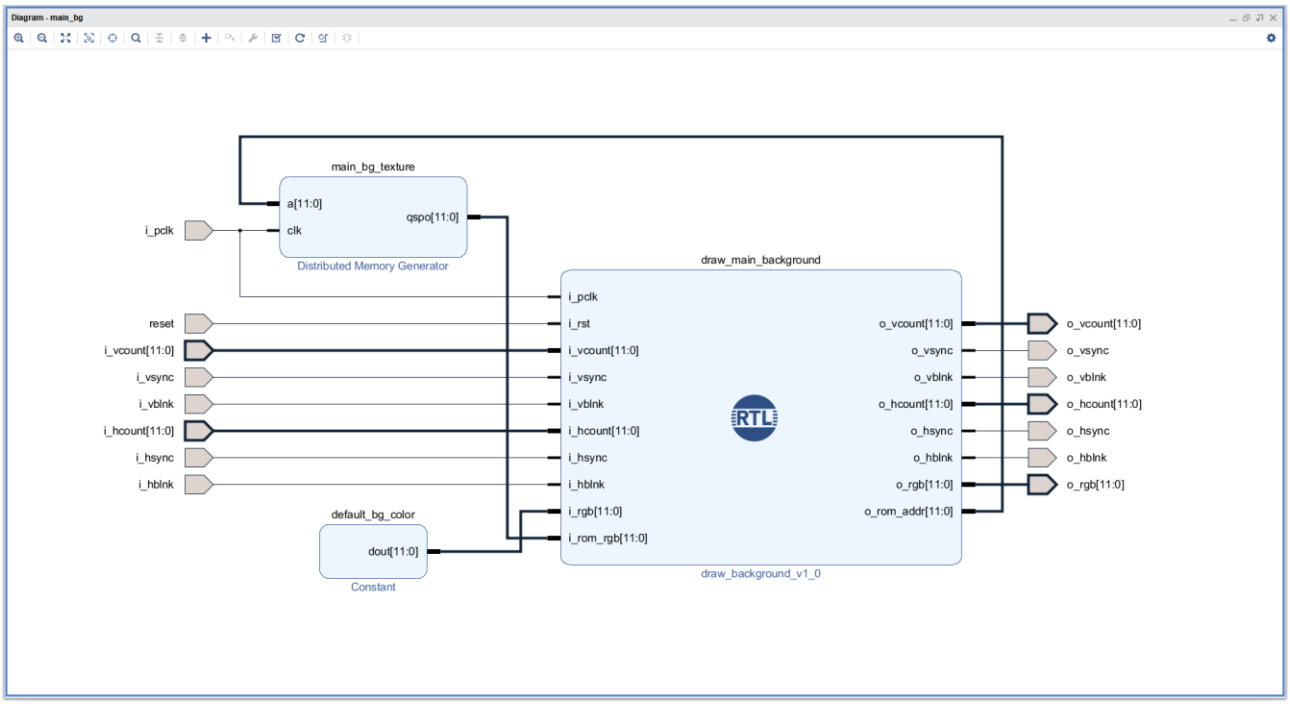
4.2.1. Moduł vga_timing

Generuje sygnały VGA dla rozdzielczości FHD 1920x1080 z częstotliwością 60 Hz.

4.2.2. Moduł scene_mux

Na podstawie danych odczytanych z pamięci modułu axi_scenes wybiera sygnały pochodzące od aktywnej sceny.

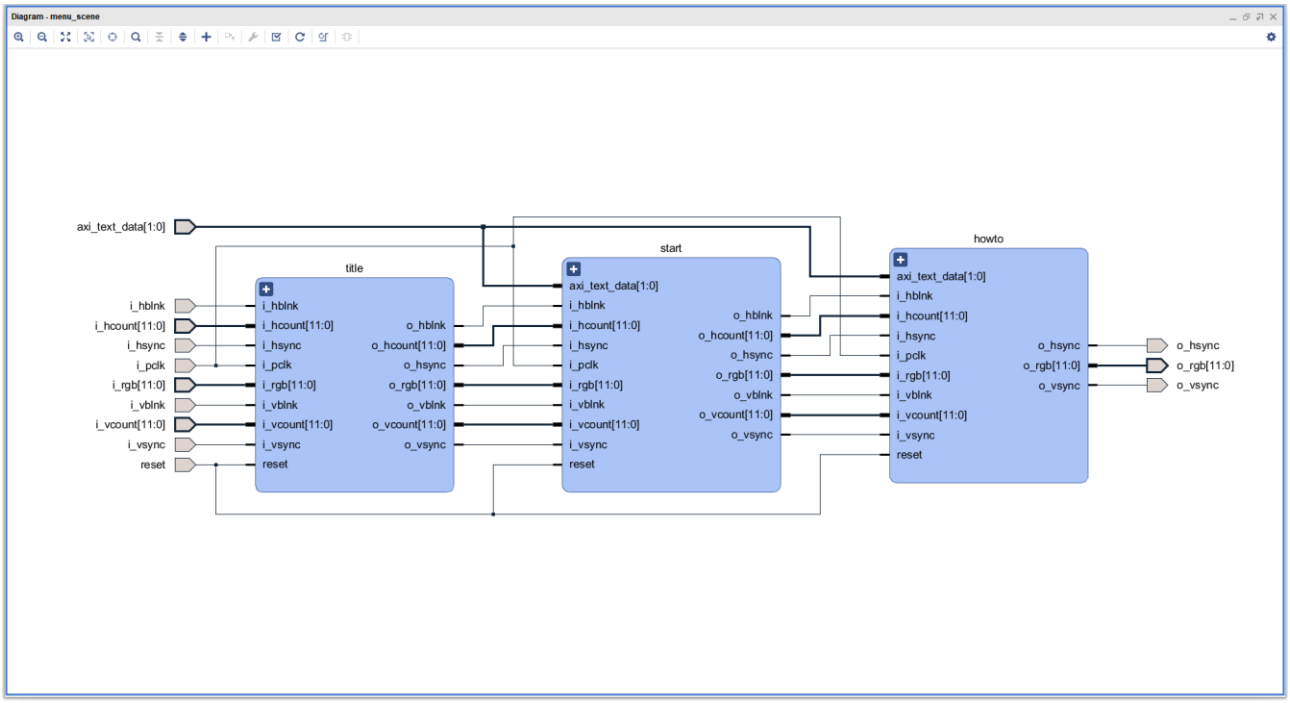
4.3. Blok main_bg



Rysuje tło główne korzystając z modułu draw_background.

draw_background

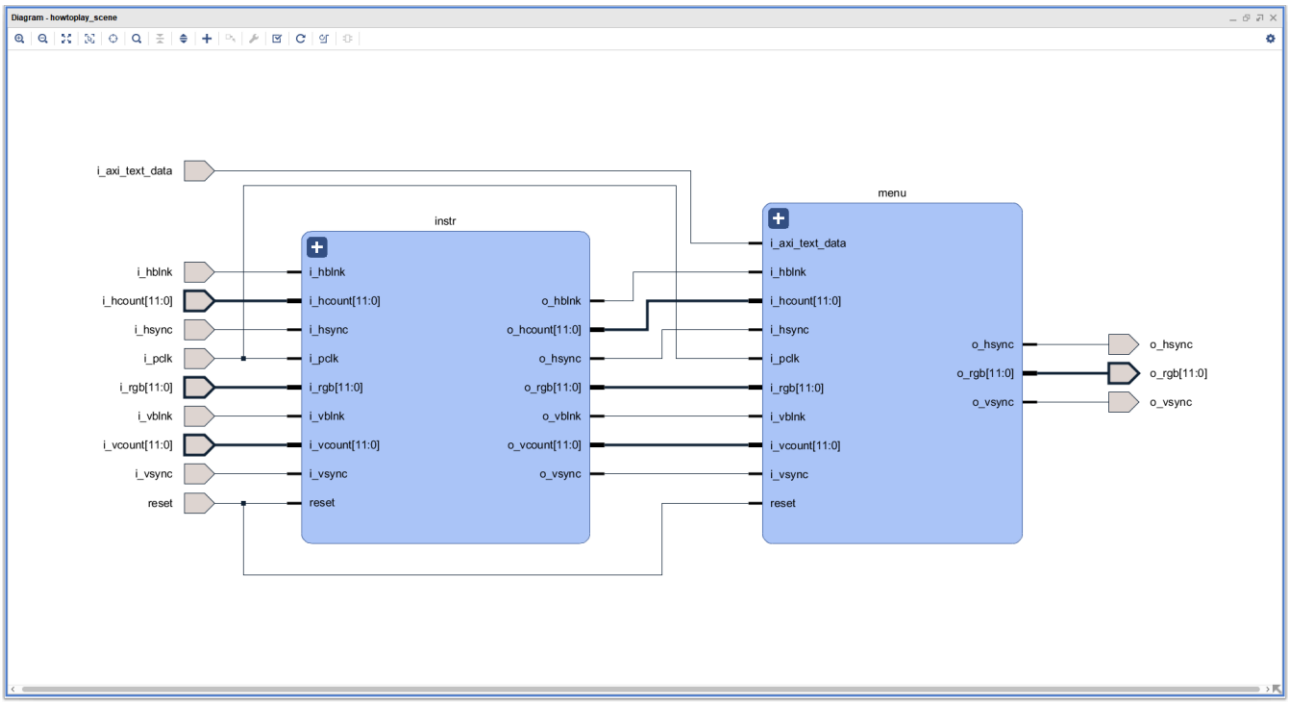
4.4. Blok menu_scene



Blok składa się z 3 mniejszych bloków odpowiedzialnych za rysowanie poszczególnych elementów menu: tytułu (title) oraz „przycisków” startu gry i instrukcji (start, howto).

draw_static_text, draw_blinking_text, draw_img_text, text_rom

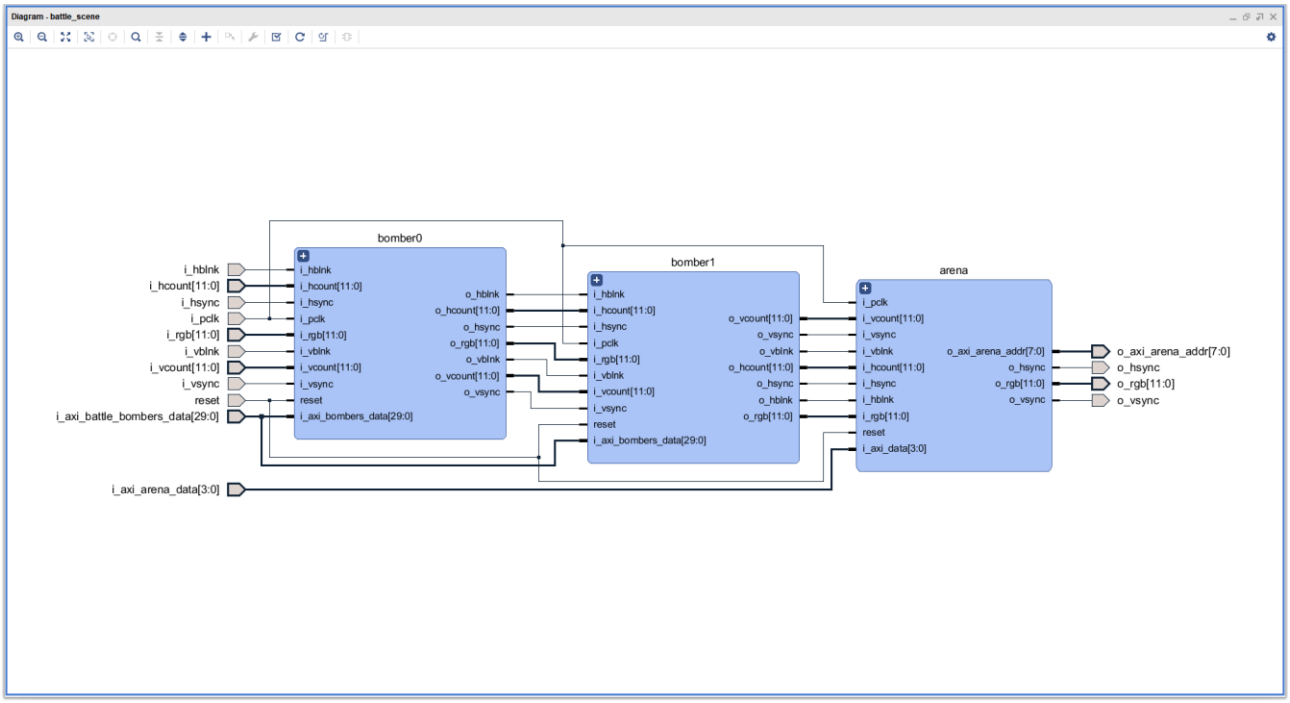
4.5. Blok howtoplay_scene



Blok składa się z 2 mniejszych bloków odpowiedzialnych za rysowanie poszczególnych elementów ekranu instrukcji: przypisania klawiszy (instr) oraz „przycisku" powrotu do menu (menu).

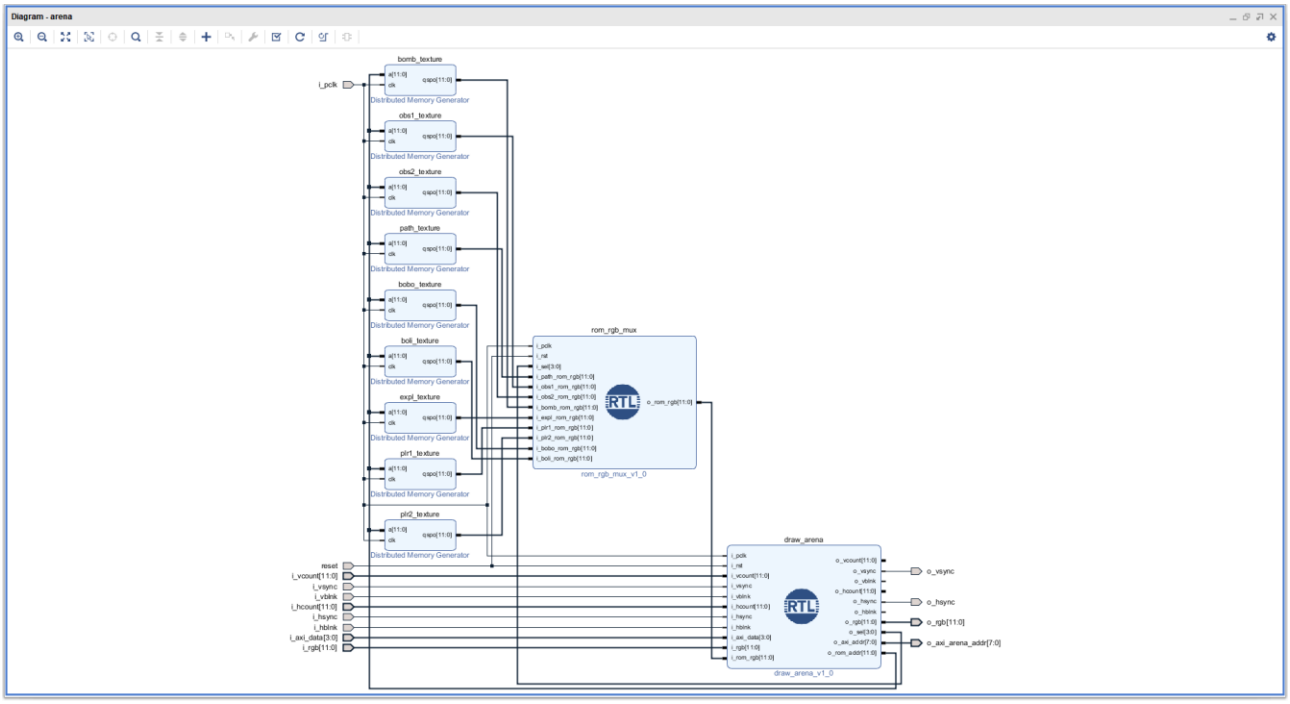
draw_static_text, draw_blinking_text, text_rom

4.6. Blok battle_scene



Blok składa się z 3 mniejszych bloków odpowiedzialnych za rysowanie poszczególnych elementów ekranu gry: areny (arena) oraz informacji o stanie graczy (bomber0, bomber1).

4.6.1. Blok arena



Składa się z 9 pamięci ROM przechowujących tekstury elementów areny, multipleksera ROM wybierającego konkretny kolor RGB oraz modułu rysującego arenę draw_arena.

Moduł draw_arena oblicza pozycję znormalizowaną rysowanego fragmentu areny i wysyła ją jako adres do modułu axi_battle_arena. Arena jest rysowana od XPOS w poziomie i YPOS w pionie, więc znormalizowana pozycja elementu wynosi:

$$\text{axi\_addr} = (\text{hcount} - \text{XPOS}) \;/\; 64 \; + (\text{vcount} - \text{YPOS}) \;/\; 64 \cdot 16$$

dla elementu o rozmiarze 64px x 64px i areny o rozmiarze 16 x 16 elementów. Jest to równoznaczne z:

```
assign axi_addrx = (i_hcount - H_MIN) >> 6;
assign axi_addry = (i_vcount - V_MIN) >> 6;
assign o_axi_addr = {axi_addry, axi_addrx};
```

Na tym samym zboczu zegara liczony jest adres koloru RGB:

$$\text{rom\_addr} = ((\text{hcount} - \text{XPOS}) \;\%\; 64) \; + ((\text{vcount} - \text{YPOS}) \;\%\; 64) \cdot 64$$

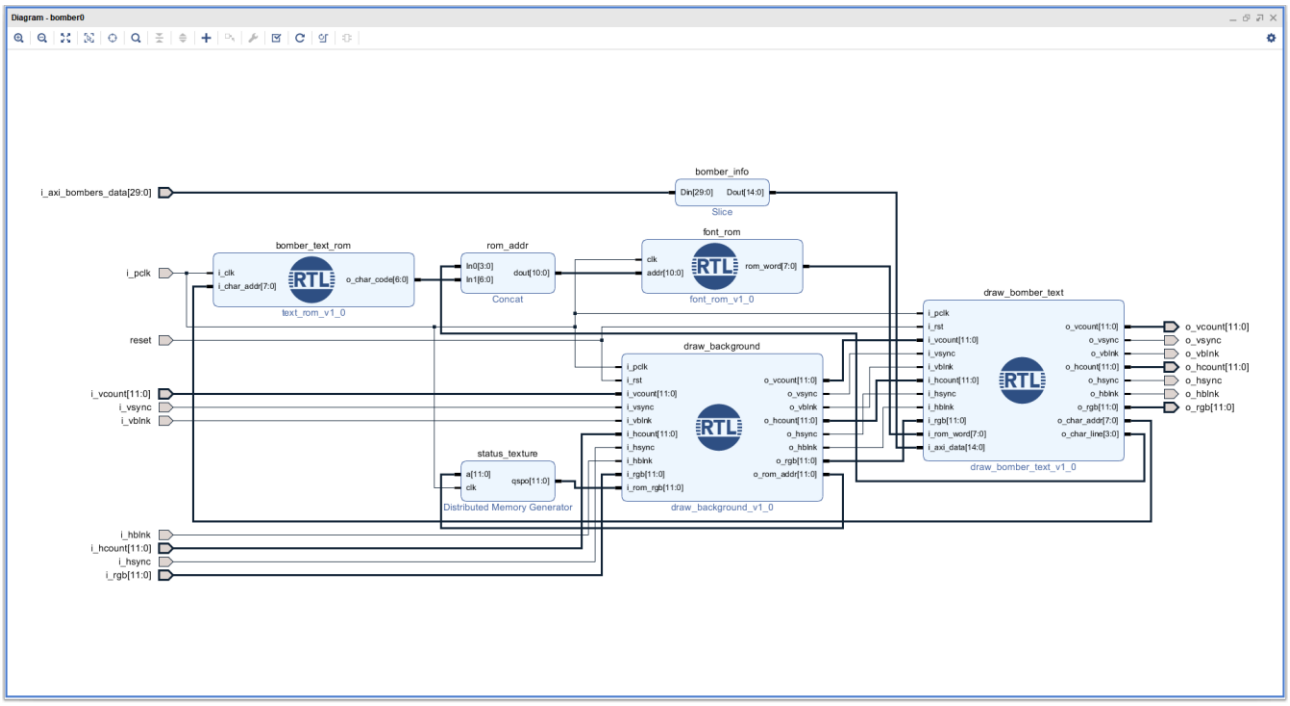
```
assign rom_addrx = (i_hcount - H_MIN) % ELEMENT_SIZE;
assign rom_addry = (i_vcount - V_MIN) % ELEMENT_SIZE;
assign o_rom_addr = {rom_addry, rom_addrx};
```

i następnie jest on wysyłany do wszystkich pamięci tekstur.

Kod elementu na obliczonej pozycji przychodzi z modułu axi_battle_arena z jednocyklowym opóźnieniem i jest on przepisywany do wyjścia sel, które jest podłączone z wejściem sel multipleksera. Z takim samym opóźnieniem pojawiają się kolory RGB na wejściach multipleksera. Wówczas rom_rgb_mux decyduje, który z kolorów przepisać na wyjście.

Finalnie, w zależności od analizowanej pozycji na ekranie, moduł draw_arena przepisuje kolor wejściowy i_rgb (tło) lub kolor wejściowy i_rgb_rom na swoje wyjście o_rgb. Wyjściowe sygnały VGA tego modułu są opóźnione o 2 cykle zegara w celu zapewnienia synchronizacji.

4.6.2. Bloki bomber0 i bomber1



Mają za zadanie rysować okienko statusu gracza.

Moduł draw_background rysuje tło dla okna statusu.

Wejście i_axi_data modułu draw_bomber_text jest połączone z wyjściem data modułu axi_bombers_text za pomocą IP Slice – dane gracza 1 znajdują się na 13 młodszych bitach, a gracza 2 na 13 starszych bitach. Moduł draw_bomber_text sprawdza, czy aktualnie przetwarzana pozycja na ekranie odpowiada pozycji którejś z danych statusu (te pozycje są zapisane jako parametry lokalne) i wysyła odpowiedni adres znaku do wejścia bomber_text_rom/i_char_addr.

W przypadku instancji bomber_text_rom, w pamięci ROM, poza samym tekstem wyświetlanym w okienku statusu, jest także lista znaków od 0 do 9:

```
1  Bomber #  status
2
3  Position: (  ,   )
4  Lives:
5  Bombs:
6
7  0123456789
8
```

Przykładowo, jeśli draw_bomber_text przetwarza pozycję danej LIVES, to jako adres znaku zostanie wysłany adres LIVES + OFFSET, gdzie OFFSET jest miejscem w pliku, w którym znajduje się lista znaków 0,1,…,9. Jeśli gracz będzie mieć 2 życia, to adres wynosi 2 + OFFSET, co wskazuje na znak ‘2’.

Adres wysyłany do pamięci ROM czcionki jest tworzony z kodu znaku (bomber_text_rom/o_char_code) oraz numeru obecnej linii znaku (draw_bomber_text/o_char_line) za pomocą IP Concat. Odczytane słowo z pamięci

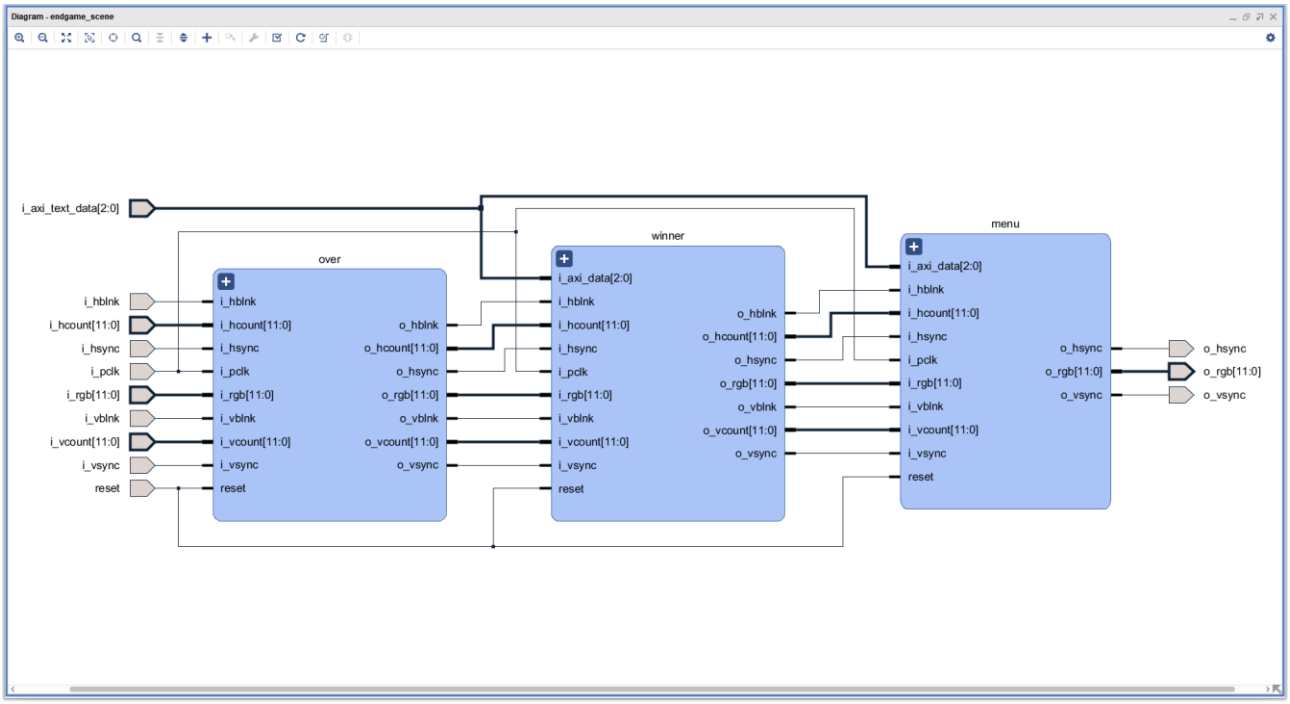
ROM jest przesyłane na wejście draw_bomber_text/i_rom_word. W module draw_bomber_text wybierany jest odpowiedni bit słowa i zapada decyzja o narysowaniu piksela znaku o kolorze COLOR (parametr), narysowaniu piksela tła okienka statusu, bądź przerysowaniu ekranu. Sygnały wewnątrz modułu są odpowiednio opóźnione.

Parametry modułu draw_bomber_text:

Parametr	Opis
ID	ID gracza, dla którego rysowane jest okno statusu
X_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji poziomej w pamięci ROM tekstu
Y_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji pionowej w pamięci ROM tekstu
SCALE_COEFF	Współczynnik skali użyty podczas rysowania tekstu 0 – oryginalna czcionka (8x16) 1 – powiększenie dwukrotne (16x32) 2 – powiększenie czterokrotne (32x64) itd.
X_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w poziomie
Y_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w pionie
X_MIN, X_MAX	Minimalna/maksymalna pozycja w poziomie Wyśrodkowanie tekstu w poziomie między nimi
Y_MIN, Y_MAX	Minimalna/maksymalna pozycja w pionie Wyśrodkowanie tekstu w pionie między nimi

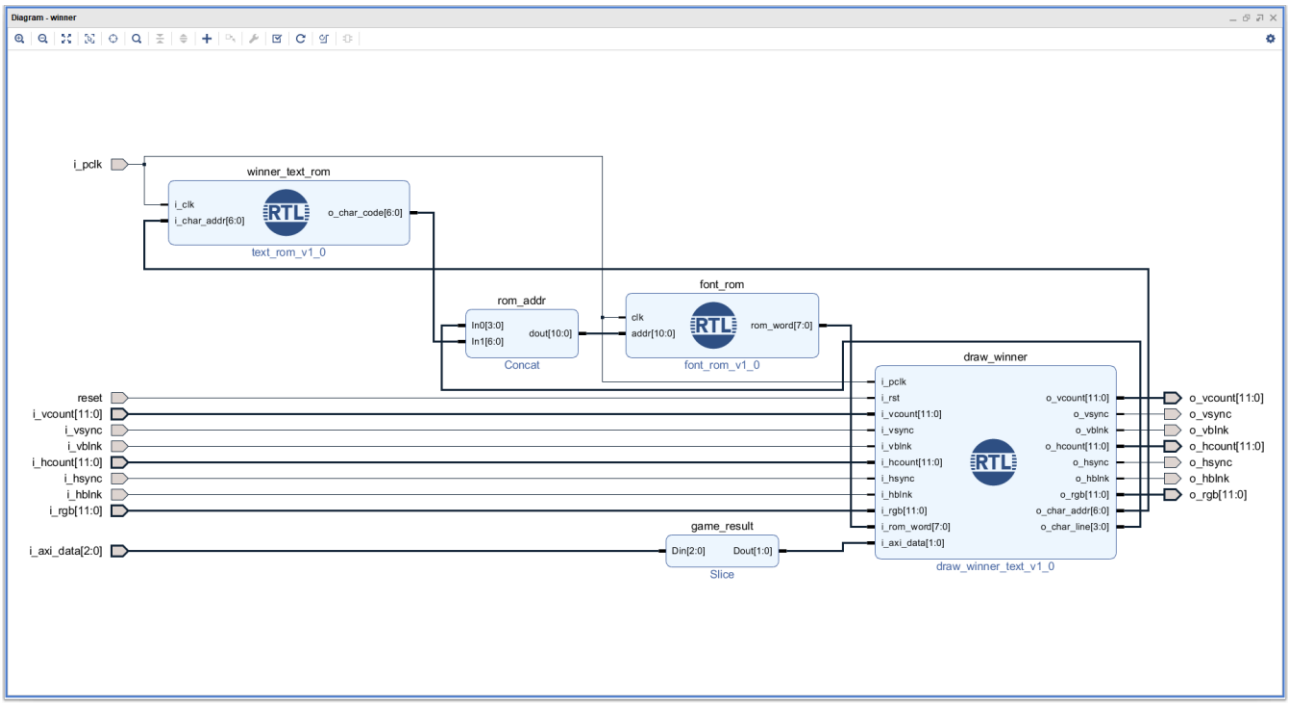
draw_background, text_rom

4.7. Blok endgame_scene



Blok składa się z 3 mniejszych bloków odpowiedzialnych za rysowanie poszczególnych elementów ekranu końcowego: napisu końca gry (over), informacji o wyniku gry (winner) oraz „przycisku” przejścia do menu (menu).

4.7.1. Blok winner



Składa się z pamięci ROM z tekstem (winner_text_rom), pamięci ROM czcionki oraz modułu draw_winner.

Moduł draw_winner otrzymuje z pamięci axi_endgame_text poprzez IP Slice game_result kod wyniku gry. Na podstawie tego kodu z ROMu wybierana jest odpowiednia linia. Jest to zrealizowane przy pomocy offsetów zdefiniowanych jako parametry lokalne. Na kolejnym zboczu zegara budowany jest adres przesyłany do font_rom. W końcu do wejścia draw_winner/i_rom_word trafia odczytane z font_rom słowo. Wewnątrz draw_winner wybierany jest odpowiedni bit i zapada decyzja o narysowaniu piksela znaku o kolorze COLOR (parametr), bądź przerysowaniu ekranu. Sygnały wewnątrz modułu są odpowiednio opóźnione.

text_rom

4.8. Moduły draw_static_text i draw_blinking_text

draw_static_text jest modułem rysującym tekst na ekranie, który nie zmienia się w żaden sposób w czasie. Musi być użyty z modułami: text_rom, IP Concat oraz font_rom.

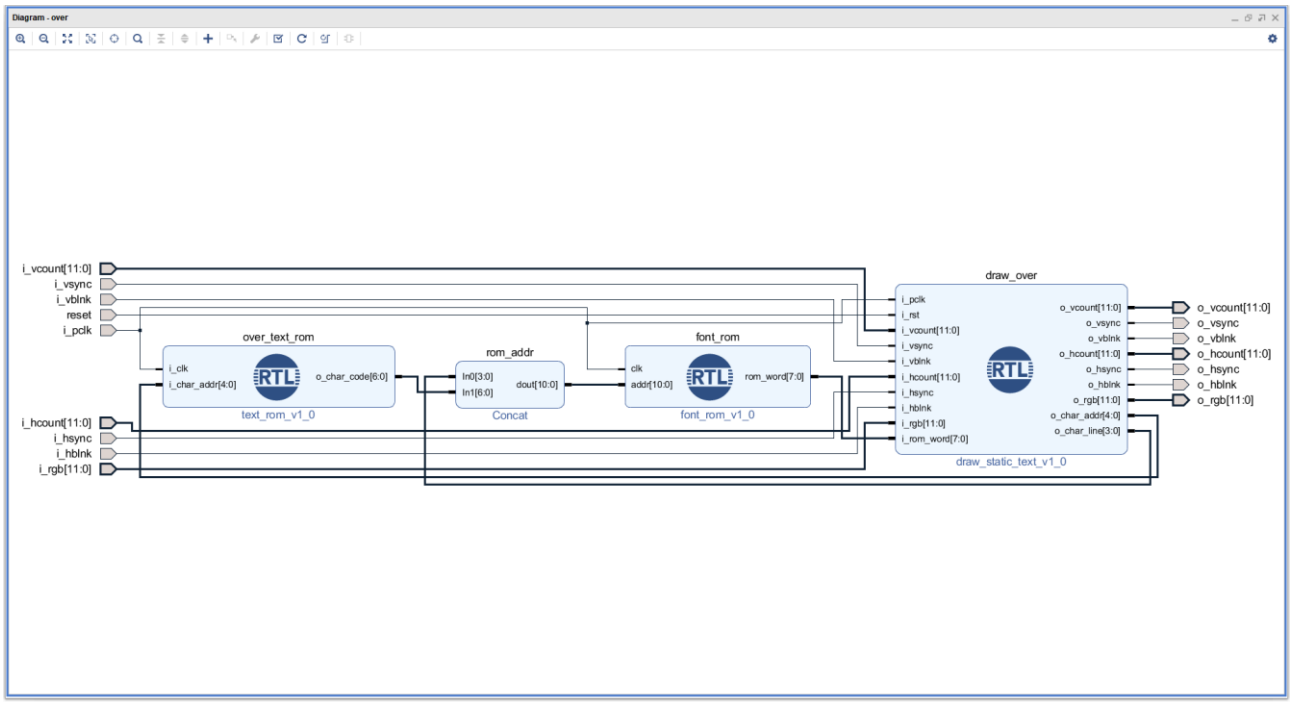
Na podstawie aktualnej pozycji w draw_static_text liczony jest adres znaku i wysyłany na wyjście draw_static_text/o_char_addr. Wyjście to połączone jest z wejściem text_rom/i_char_addr. W text_rom wybierany jest odpowiedni znak i jego kod jest wysyłany na wyjście text_rom/o_char_code. Kod znaku i numer linii znaku tworzą adres słowa ROM (tworzony przez IP Concat), który jest wysyłany do wejścia font_rom/addr. Na wyjściu font_rom/rom_word otrzymujemy 8-bitowe słowo, które jest przesyłane do draw_static_text/i_rom_word. Wewnątrz draw_static_text wybierany jest bit słowa. W zależności od tego bitu i aktualnej pozycji rysowany jest piksel tekstu lub tło.

draw_blinking_text działa na tej samej zasadzie jak draw_static_text, lecz dodatkowo sprawdza stan wejścia i_axi_data. Jeśli wybrany bit z i_rom_word jest zerowy, to rysowane jest tło. Jeśli i_axi_data jest równe 1 – rysowane jest przyciemnione tło (składowe R, G i B są zmniejszane o 1). W innym wypadku rysowane jest oryginalne tło. Daje to efekt mrugania.

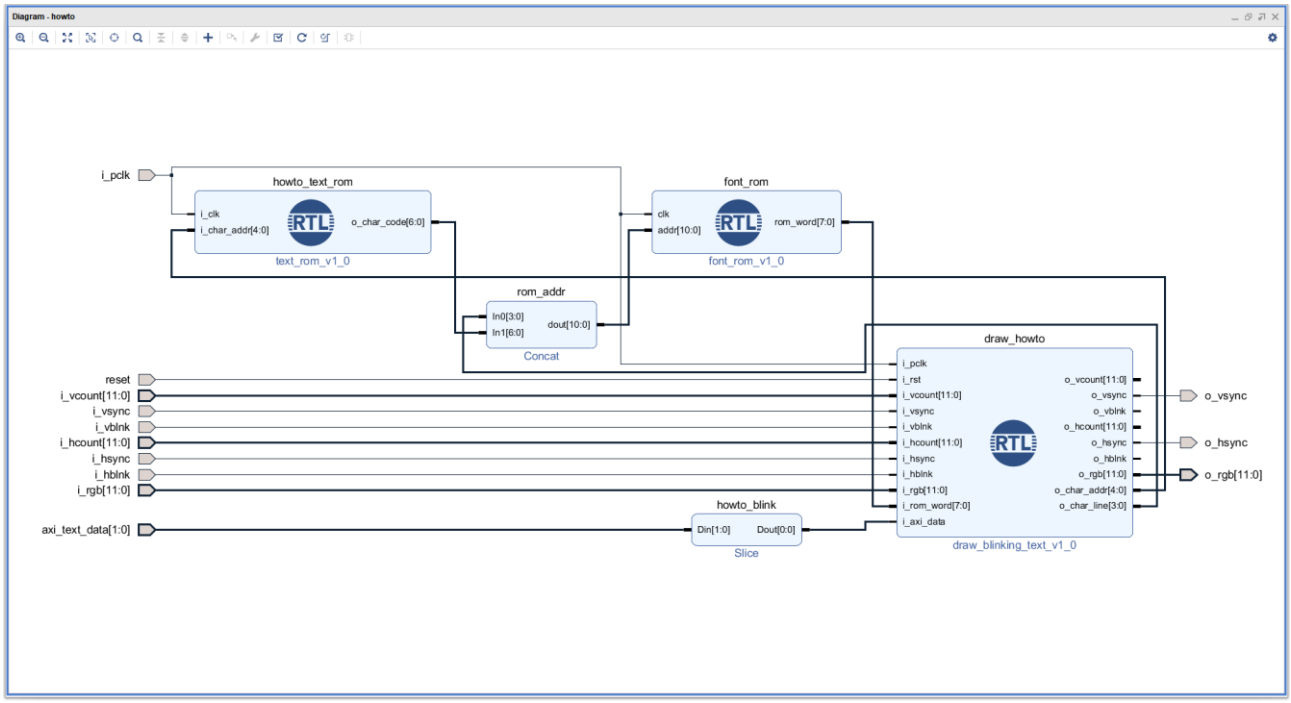
Opis parametrów:

Parametr	Opis
COLOR	Kolor rysowanego tekstu
SCALE_COEFF	Współczynnik skali użyty podczas rysowania tekstu 0 – oryginalna czcionka (8x16) 1 – powiększenie dwukrotne (16x32) 2 – powiększenie czterokrotne (32x64) itd.
X_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji poziomej w pamięci ROM tekstu
X_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w poziomie
X_MIN, X_MAX	Minimalna/maksymalna pozycja w poziomie Wyśrodkowanie tekstu w poziomie między nimi
Y_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji pionowej w pamięci ROM tekstu
Y_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w pionie
Y_MIN, Y_MAX	Minimalna/maksymalna pozycja w pionie Wyśrodkowanie tekstu w pionie między nimi

Przykład użycia draw_static_text dla endgame_scene/over:



Przykład użycia draw_blinking_text dla menu_scene/howto:



4.9. Moduł text_rom

text_rom jest modułem z pamięcią ROM tekstu. Plik z tekstem jest zapisany w formacie HEX i jego ścieżka podawana jest w parametrze.

Opis parametrów:

Parametr	Opis
PATH	Ścieżka do pliku .data
X_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji poziomej w pamięci ROM tekstu
Y_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji pionowej w pamięci ROM tekstu

4.10. Moduł draw_img_text

Rysuje tekst stosując kolor grafiki umieszczonej w pamięci ROM.

Opis parametrów:

Parametr	Opis
X_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji poziomej w pamięci ROM tekstu
Y_ADDR_WIDTH	Ilość bitów potrzebna do zaadresowania pozycji pionowej w pamięci ROM tekstu
SCALE_COEFF	Współczynnik skali użyty podczas rysowania tekstu 0 – oryginalna czcionka (8x16) 1 – powiększenie dwukrotne (16x32) 2 – powiększenie czterokrotne (32x64) itd.
X_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w poziomie
Y_CHAR_COUNT	Maksymalna liczba wyświetlanych znaków w pionie
X_MIN, X_MAX	Minimalna/maksymalna pozycja w poziomie Wyśrodkowanie tekstu w poziomie między nimi
Y_MIN, Y_MAX	Minimalna/maksymalna pozycja w pionie Wyśrodkowanie tekstu w pionie między nimi

Ten moduł jest użyty do rysowania tekstu tytułu w menu.

4.11. Moduł draw_background

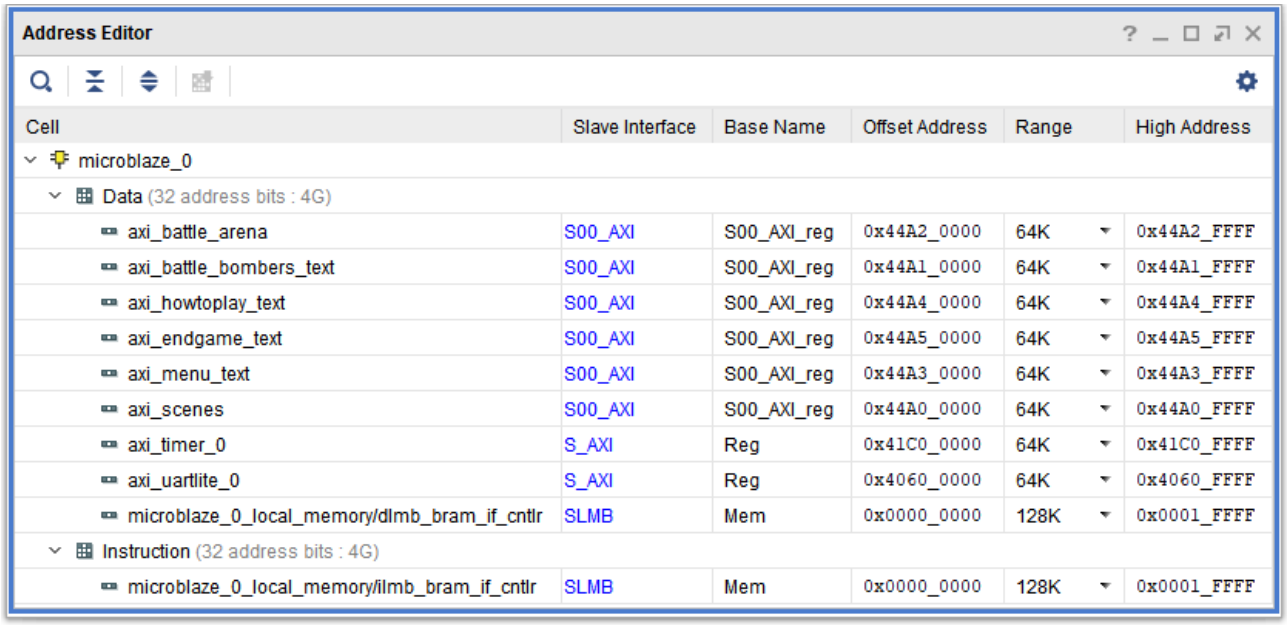
Rysuje tło korzystając z pamięci ROM 64px x 64px.

Opis parametrów:

Parametr	Opis
H_MIN	Pozycja, od której rysować w poziomie
H_MAX	Pozycja, do której rysować w poziomie
V_MIN	Pozycja, od której rysować w pionie
V_MAX	Pozycja, do której rysować w pionie

4.12. Sterowanie zawartością pamięci AXI Bomberman (Single) Memory

Każda instancja AXI ma przypisany adres bazowy. Jest on ustalany na etapie tworzenia schematu blokowego i można go sprawdzić pod zakładką Address Editor (tutaj nazwany Offset Address):



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
axi_battle_arena	S00_AXI	S00_AXI_reg	0x44A2_0000	64K	0x44A2_FFFF
axi_battle_bombers_text	S00_AXI	S00_AXI_reg	0x44A1_0000	64K	0x44A1_FFFF
axi_howtoplay_text	S00_AXI	S00_AXI_reg	0x44A4_0000	64K	0x44A4_FFFF
axi_endgame_text	S00_AXI	S00_AXI_reg	0x44A5_0000	64K	0x44A5_FFFF
axi_menu_text	S00_AXI	S00_AXI_reg	0x44A3_0000	64K	0x44A3_FFFF
axi_scenes	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	128K	0x0001_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	128K	0x0001_FFFF

Po zaimportowaniu hardware'u do SDK i regeneracji źródeł BSP aktualizowany jest plik xparameters.h, w którym zawarte są definicje dotyczące użytych modułów AXI, w tym adresy bazowe. W przypadku gotowych modułów AXI – jak np. Uartlite – adres bazowy jest niezbędny do używania gotowego API. Przykładowo, funkcja XUartLite_RecvByte oczekuje podania jako argument adresu bazowego instancji AXI Uartlite.

Ogólnie rzecz biorąc adres bazowy jest wskaźnikiem. W przypadku naszego AXI Bomberman (Single) Memory przypisanie wartości pod adres bazowy oznacza wysłanie danych do modułu AXI. Przykładowo poniższy kod ustawia aktywną scenę jako Battle (adres bazowy zgodny ze screenem powyżej):

```
#define AXI_SCENES_BASEADDR 0x44A00000
uint32_t *axi_scenes = reinterpret_cast<uint32_t*>(AXI_SCENES_BASEADDR);
*axi_scenes = 1;
```

Sposób wysyłania danych do obydwu modułów AXI Bomberman Single Memory i AXI Bomberman Memory nie rózni się niczym – polega na ustawieniu wartości wskaźnika. Jednak w przypadku AXI Bomberman Memory trzeba odpowiednio zbudować „ramkę danych".

AXI Bomberman Memory posiada wiele komórek pamięci w przeciwieństwie do AXI Bomberman Single Memory. Aby wskazać do której komórki chcemy coś wpisać należy na najmłodszych ADDR_WIDTH bitach wpisać adres komórki. Zatem nowa wartość jest przesunięta bitowo o ADDR_WIDTH w ramce. Wynika to z implementacji AXI Bomberman Memory:

```
always @(posedge s00_axi_aclk)
    if (s00_axi_awvalid && s00_axi_awready)
        memory[s00_axi_wdata[ADDR_WIDTH-1:0]] <= s00_axi_wdata[DATA_WIDTH+ADDR_WIDTH-1:ADDR_WIDTH];
```

W kodzie C++ metoda Arena::Draw przyjmuje jako argument adres bazowy instancji axi_battle_arena (drawer) i w pętli zapisuje do odpowiedniej komórki dane elementu:

```
for (const auto &ve : visible)
{
    *drawer = (ve->Code() << 8) + ve->GetNormalizedPosition();
}
```

Kod elementu jest przesuwany o 8 bitów w lewo, ponieważ ADDR_WIDTH jest równe 8.

5. Implementacja. Zaawansowanie na 10.09.2019 – 100%

6. Film. Zaawansowanie na 10.09.2019 – 100%

Link do ściągnięcia filmu:

https://drive.google.com/open?id=18dinKnmbFVfMRYR385YRHtESh4WQq4uN