PROJEKT INŻYNIERSKI

„Realizacja gry logicznej Mistrz Intelektu na bazie układu FPGA”

Michał GOLD
Nr albumu 293468

Kierunek: Automatyka i robotyka
Specjalność: Technologie informacyjne w automatyce i robotyce"

PROWADZĄCY PRACĘ
Dr inż. Tomasz Garbolino
KATEDRA Systemów Cyfrowych
Wydział Automatyki, Elektroniki i Informatyki

GLIWICE 2023

Tytuł pracy:

Realizacja gry logicznej Mistrz Intelektu na bazie układu FPGA

Streszczenie:

Celem pracy było zaprojektowanie i wykonanie układu realizującego grę logiczną "Mistrz intelektu" (ang. "Master Mind") w oparciu układ FPGA. Zaprojektowany układ wyświetla obraz gry na monitorze z interfejsem VGA i/lub HDMI. Sterowanie odbywa się poprzez przyciski obecne na osobnej płytce stykowej podłączonej do urządzenia. Projekt wyposażony jest również w głośnik, potrafiący odtwarzać proste melodie. Użytkownik jest w stanie konfigurować wiele parametrów gry, takich jak ilość dostępnych kolorów, rozmiar planszy oraz liczba dozwolonych prób odgadnięcia ukrytej sekwencji. Możliwe jest też ustawienie rozmiaru czcionki wyświetlanej na ekranie oraz palety barw, jaka jest wykorzystywana do rysowania wielu elementów.

Słowa kluczowe:

Mistrz Intelektu, FPGA, SystemVerilog, VGA, HDMI, MAXimator

Thesis title:

Implementation of the puzzle game "Master Mind" based on the FPGA

Abstract:

The aim of the work was to design and create a system implementing the "Master Mind" logic game based on the FPGA system. Designed system can display the game image on a monitor with a VGA and/or HDMI interface. The control is carried out using a separate breadboard connected of the device. The project is also equipped with a speaker that can play simple tunes. The user can configure many parameters of the game, such as the number of available colors, the size of the gameboard and the number of attempts to guess the hidden sequence. It is also possible to set the configuration of the font size and the color palette, that is used to draw various elements on the screen.

Keywords:

MasterMind, FPGA, SystemVerilog, VGA, HDMI, MAXimator

Spis treści

Rozdział 1

Wstęp

Praca Inżynierska składa się z modułu FPGA (Field-Programmable Gate Array), na którym zaimplementowana jest gra Mistrz Intelektu, znana także jako MasterMind. Urządzenie ma mieć możliwość wyświetlania obrazu na monitorze oraz oczywiście pozwalać graczowi, bądź graczom, na wygodne kontrolowanie gry. Powinno pozwalać także na dynamiczną modyfikację parametrów gry, takich jak np. paleta kolorów lub rozdzielczość wyświetlanego obrazu, Całość prezentuje także możliwości układów FPGA oraz rozwiązuje problem zaprojektowania rozbudowanej gry na systemie innym niż wykorzystującym standardowe procesory komputerowe. Celem autora projektu był wybór odpowiedniego układu FPGA, który nie byłby zbyt drogi, a jednocześnie umożliwiałby realizację projektu, zaprojektowanie systemu sterowania grą, jak i zaprojektowanie samej gry. Należało zaznajomić się z oprogramowaniem wykorzystywanym do programowania i testowania wybranego urządzenia. Problem poruszany w pracy inżynierskiej zawiera się więc w dziedzinie projektowania układów elektronicznych.

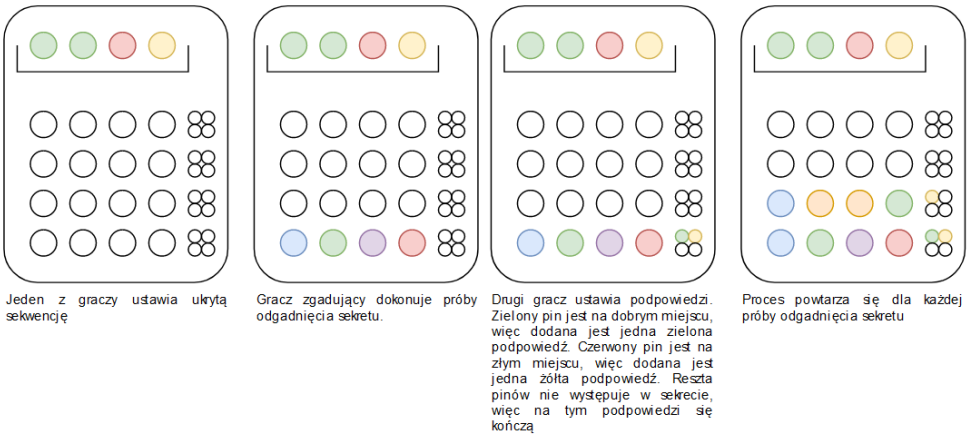
Praca obejmuje 7 rozdziałów. Ogólny zakres pracy w skróconej formie przedstawiono w rozdziale pierwszym. W rozdziale drugim przedstawiono rozszerzoną analizę tematu poruszanego w pracy, z szczegółowym opisaniem rodzaju urządzeń jakie są w projekcie wykorzystywane. Rozdział trzeci dotyczy opisu konkretnych urządzeń i oprogramowania jakie ostatecznie zostały wykorzystane do stworzenia projektu, wraz z uzasadnieniem ich wyboru. Czwarty rozdział prezentuje ostateczny wygląd projektu, zawiera szczegółowy opis możliwości gry i instrukcję korzystania z gotowego urządzenia. Szczegóły implementacji zawarte zostały w rozdziale kolejnym, to jest piątym. W rozdziale szóstym przedstawiono wykorzystane metody testowania tworzonego systemu oraz opis problemów wykrytych podczas tego procesu. Ostatni rozdział to podsumowanie całego projektu oraz propozycje jego dalszej rozbudowy.

Rozdział 2

Analiza tematu

2.1 Zasady gry MasterMind

Gra MasterMind, znana także jako Mistrz Intelektu, to popularna logiczna gra planszowa w której rywalizuje ze sobą 2 graczy. Gra składa się z planszy będącej siatką dziur, do których można wkładać różnokolorowe grzybki, nazywane dalej pinami. Jeden z wierszy posiada pokrywkę, która zasłania piny w nim ustawione. Obok każdego wiersza dziur znajdują się też kilka mniejszych dziur, na mniejsze piny, które występują tylko w 2 kolorach. Jeden z graczy na początku gry ustawia wybraną sekwencję kolorów pinów, nazywaną dalej ukrytą sekwencją lub sekretem, w rzędzie osłoniętym pokrywką, tak, aby drugi gracz jej nie znał. Następnie zadaniem drugiego gracza jest odgadnięcie tej sekwencji. W tym celu, ustawia w wolnym rzędzie dziur własną sekwencję pinów, która stanowi jego próbę odgadnięcia sekretu. Kolejno gracz który stworzył sekret, wkładając mniejsze piny w mniejsze dziury obok tego wiersza, określa jak bardzo podobna jest proponowana sekwencja do sekwencji ukrytej. Jeden z kolorów mniejszych pinów reprezentuje pin sekwencji o kolorze, który znajduje się w sekrecie na tej samej pozycji, co w sekwencji ułożonej przez zgadującego. Piny te nazywane będą dalej podpowiedziami zielonymi. Drugi kolor pinów podpowiedzi oznacza, że jeden z kolorów w sekwencji ułożonej przez zgadującego znajduje się w sekrecie, ale na innej pozycji. Piny te nazywane będą dalej żółtymi podpowiedziami. Gracz zgadujący nie wie, która podpowiedź odpowiada któremu pinowi sekwencji. Jego zadaniem jest domyślenie się tego. Każda podpowiedź odnosi się do dokładnie jednego pinu w sekwencji, oraz każdy pin sekwencji może być powiązany z maksymalnie jedną podpowiedzią. Przykładowy przebieg początku partii przedstawiono na Rys. 2.1.



Rys. 2.1: Przykładowy przebieg początku partii MasterMind

Gracz zgadujący wygrywa, jeśli uda mu się odgadnąć ukrytą sekwencję w dostępnej liczbie prób, będącej ilością wierszy na planszy. Jeśli mu się nie uda, wygrywa gracz układający sekret.

Gra ta jest bardzo asymetryczna. Jeden z graczy musi aktywnie eliminować możliwe kombinacje kolorów na podstawie poprzednio otrzymanych podpowiedzi, podczas gdy rola drugiego, nie licząc początkowego stworzenia sekretu, jest w pełni bierna, dlatego łatwo jest przerobić grę MasterMind na jednoosobową, gdzie gracz rywalizuje z komputerem, który automatycznie losuje sekwencje kolorów i wyznacza wartości podpowiedzi.

2.2 Układy FPGA

Układy FPGA (Field-Programmable Gate Array), znane po polsku także jako Bezpośrednio Programowalne Macierze Bramek, to układy scalone, które po odpowiednim zaprogramowaniu, są w stanie odtworzyć zachowanie teoretycznie dowolnego specjalizowanego układu scalonego. Wewnątrz układu FPGA znajduje się sieć połączeń pomiędzy wieloma (od dziesiątek do dziesiątek tysięcy, zależnie od modelu) jednostek logicznych. Na jednostki logiczne składa się wiele różnych elementów, najczęściej są nimi pamięć LUT (LookUp Table) wykorzystywana do przetwarzania sygnałów na podstawie przypisania danej sekwencji wyjść do sekwencji wejść, oraz przerzutniki, służące zapamiętywaniu danych. Często w FPGA występują też bloki bardziej specjalizowane, jak np. jednostki do obliczeń arytmetycznych, jednostki pamięci RAM lub Flash. Programowanie układów FPGA polega na odpowiednim skonfigurowaniu sieci połączeń pomiędzy tymi jednostkami logicznymi oraz inicjalizacji jednostek pamięci odpowiednimi wartościami, w taki sposób, aby urządzenie realizowało zaprojektowaną funkcję.

Układy FPGA bardzo często są wykorzystywane na etapie testowania projektu układów scalonych. Stanowią one dobrą alternatywę do specjalizowanych układów scalonych, ponieważ, o ile nie jest wymagana bardzo wysoka częstotliwość pracy (np. powyżej 500MHz), to przy odpowiedniej konfiguracji, pozwalają na prawie idealne odwzorowanie ich zachowania. W przeciwieństwie do nich nie wymagają takich dużych kosztów, jak stworzenie pojedynczego prototypu danego układu scalonego. Jeśli projektowany układ scalony nie ma być produkowany masowo, nic nie stoi na przeszkodzie aby również skorzystać z FPGA w finalnej wersji produktu, ze względu na mniejsze koszty. Może się wydawać, że podobną funkcję mogą spełniać mikroprocesory, lecz w przeciwieństwie do nich układy FPGA nie muszą być ograniczone częstotliwością taktowania zegara procesora, potrafiąc symulować dowolny układ bramek logicznych równolegle, o ile jednostki logiczne wewnątrz FPGA pomieszczą dany schemat. Fakt ten jest wykorzystywany przede wszystkim w implementacji układów kombinacyjnych. W przypadku tego projektu wymagane jest generowanie sygnałów z relatywnie wysoką częstotliwością, na potrzeby generowania sygnału obrazu, co byłoby bardzo trudne do osiągnięcia, o ile w ogóle możliwe, przy zastosowaniu wyłącznie mikroprocesora.

2.3 Język opisu sprzętu Verilog

Programowanie układów FPGA odbywa się nie za pomocą typowych, strukturalnych języków programowania jak C czy Asembler, lecz za pomocą tzw. języków opisu sprzętu. Język ten zamiast opisywać kolejne polecenia jakie ma wykonywać typowy procesor, opisuje jedynie zależności pomiędzy konkretnymi modułami, na które składają się rejestry, bloki realizujące funkcje logiczne i połączenia między nimi. Jednym z popularniejszych języków opisu sprzętu jest Verilog, którego najnowsza i najbardziej rozbudowana wersja nazywana jest SystemVerilog.

SystemVerilog pozwala na stworzenie obiektów zwanych modułami, do których przypisane są rejestry przechowujące liczby binarne, sieci przyjmujące wartości będące wynikiem funkcji logicznych wartości z innych rejestrów lub sieci, jaki i inne moduły, pomiędzy którymi mogą być wymieniane dane. Moduły posiadają też porty wejściowe i wyjściowe. Można zdefiniować także tzw. bloki „always”, kod w których zawarty ma być wykonywany jedynie w momencie wystąpienia zbocza na danym sygnale. Przykładowo, kod na Rys. 2.2 przedstawia definicję modułu TEST, który wraz z narastającym zboczem sygnału zegarowego clk dodaje liczbę będącą wartością portu wejściowego do wartości w 5-bitowym rejestrze, a do sieci stanowiącej port wyjściowy przypisuje wartość po zastosowaniu alternatywy wykluczającej (tj. operacji XOR) z wartością ustawioną w stałym parametrze X.

```
module TEST(
    input wire clk,
    input wire [4:0] in,
    output wire out
);
    parameter X = 5'b10110;
    reg [4:0] rejestr = 0;
    assign out = rejestr ^ X;
    always @(posedge clk) begin
        rejestr <= rejestr + in;
    end
endmodule
```

Rys. 2.2: Przykładowy kod w języku Verilog

Rozdział 3

Wymagania i narzędzia

3.1 Zestaw MAXimator

Pierwszym krokiem w stworzeniu gry było wybranie konkretnego modelu układu FPGA. Układy FPGA same w sobie są najczęściej ciężkie w integracji. Wymagają specjalistycznych narzędzi do lutowania na płytkach PCB (Printed Circuit Board) oraz zaprojektowania gęstych ścieżek na płytce PCB aby połączyć wyprowadzenia układu ze wszystkimi wykorzystywanymi portami i urządzeniami peryferyjnymi. Dlatego w sytuacjach, gdzie nie jest wymagana duża oszczędność w przestrzeni, często wykorzystywane są zestawy startowe, będące już gotowymi płytkami z wlutowanym układem FPGA i połączonymi z nim różnymi portami i np. diodami LED (Light Emiting Diode) sygnalizującymi stan urządzenia. Płytką będącą takim zestawem startowym, z jakiego postanowiono skorzystać, jest MAXimator wykorzystujący układ FPGA 10M08DAF256C8G z rodziny MAX10 firmy Altera, należącej do firmy Intel. Wyżej wspomniany układ FPGA posiada 8000 elementów logicznych i 378kb pamięci ulotnej. Oprócz samego FPGA, na płytce znajduje się także między innymi port VGA (Video Graphics Array), port HDMI (High Definition Multimedia Interface), port USB (Universal Serial Bus), złącze JTAG (Joint Test Action Group) służące do programowania, slot na kartę MicroSD, 4 diody LED oraz 15 wejść/wyjść cyfrowych ogólnego przeznaczenia [KL]. Zestaw startowy sprzedawany jest razem z programatorem USB Blaster PRO, który stanowi adapter pomiędzy złączem USB komputera i złączem JTAG płytki. Zestaw MAXimator w połączeniu ze swoją relatywnie niewielką ceną stanowi więc idealne urządzenie pozwalające na stworzenie projektu.

3.2 Oprogramowanie Quartus Prime Lite

W celu zaprogramowania zestawu MAXimator wymagane jest odpowiednie oprogramowanie. Dedykowanym oprogramowaniem jest Quartus Prime Lite, dostępny za darmo do pobrania ze strony internetowej firmy Intel. Jest to zintegrowane środowisko deweloperskie, pozwalające na edytowanie plików języków opisu sprzętu, jak np. SystemVerilog, konfigurację kompatybilnych układów FPGA oraz współpracę z licznymi zewnętrznymi narzędziami EDA (Engineering Design Automation).

Aby móc zacząć programować układ FPGA, należy go najpierw skonfigurować. W tym celu, po utworzeniu nowego projektu, w zakładce `Assigments > Device` trzeba wybrać z listy model urządzenia z którego się korzysta. W przypadku tego projektu, jest to 10M08DAF256C8G. Następnie, po naciśnięciu przycisku `Device And Pin Options`, można dokonać szczegółowej konfiguracji charakterystycznej dla tego modelu FPGA. Aby móc wykorzystać inicjalizację bloków pamięci w FPGA, co jest potrzebne do działania projektu, trzeba było jako opcję `Configuration Mode` ustawić „`Single Uncompressed Image with Memory Inicialization`”. Następnie w zakładce `Assigments > Pin Planner` należało przypisać funkcje i nazwy wyprowadzeń FPGA (Rys. 3.1). Stany tych wyprowadzeń będą powiązane z wejściami i wyjściami głównego modułu napisanego w języku SystemVerilog, który zostanie zrealizowany w macierzy FPGA.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
BTN_RAW_DEBUG	Input	PIN_B15	6	B6_N0	PIN_B15	2.5 V		12mA (default)			
BTN_RAW_DOWN	Input	PIN_J15	6	B6_N0	PIN_J15	2.5 V		12mA (default)			
BTN_RAW_ENTER	Input	PIN_H16	6	B6_N0	PIN_H16	2.5 V		12mA (default)			
BTN_RAW_LEFT	Input	PIN_L16	5	B5_N0	PIN_L16	2.5 V		12mA (default)			
BTN_RAW_RIGHT	Input	PIN_J16	6	B6_N0	PIN_J16	2.5 V		12mA (default)			
BTN_RAW_UP	Input	PIN_H15	6	B6_N0	PIN_H15	2.5 V		12mA (default)			
BUZZER_ON	Output	PIN_B16	6	B6_N0	PIN_B16	2.5 V		12mA (default)	2 (default)		
CLK0	Input	PIN_L3	2	B2_N0	PIN_L3	2.5 V		12mA (default)			
HDMI_CLK	Output	PIN_T2	3	B3_N0	PIN_T2	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_CLK(n)	
HDMI_D0	Output	PIN_T4	3	B3_N0	PIN_T4	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D0(n)	
HDMI_D1	Output	PIN_R2	3	B3_N0	PIN_R2	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D1(n)	
HDMI_D2	Output	PIN_R5	3	B3_N0	PIN_R5	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D2(n)	
LED0	Output	PIN_M16	5	B5_N0	PIN_M16	2.5 V		12mA (default)	2 (default)		
LED1	Output	PIN_N16	5	B5_N0	PIN_N16	2.5 V		12mA (default)	2 (default)		
LED2	Output	PIN_P16	5	B5_N0	PIN_P16	2.5 V		12mA (default)	2 (default)		
LED3	Output	PIN_R16	5	B5_N0	PIN_R16	2.5 V		12mA (default)	2 (default)		
VGA_B	Output	PIN_M1	2	B2_N0	PIN_M1	2.5 V		12mA (default)	2 (default)		
VGA_G	Output	PIN_N1	2	B2_N0	PIN_N1	2.5 V		12mA (default)	2 (default)		
VGA_HSYNC	Output	PIN_L1	2	B2_N0	PIN_L1	2.5 V		12mA (default)	2 (default)		
VGA_R	Output	PIN_R1	3	B3_N0	PIN_R1	2.5 V		12mA (default)	2 (default)		
VGA_VSYNC	Output	PIN_J1	1B	B1_N0	PIN_J1	2.5 V		12mA (default)	2 (default)		
HDMI_D0(n)	Output	PIN_T5	3	B3_N0	PIN_T5	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D0	
HDMI_D1(n)	Output	PIN_R3	3	B3_N0	PIN_R3	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D1	
HDMI_D2(n)	Output	PIN_R6	3	B3_N0	PIN_R6	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_D2	
HDMI_CLK(n)	Output	PIN_T3	3	B3_N0	PIN_T3	LVDS_E_3R		Maximu...fault)	2 (default)	HDMI_CLK	

Rys. 3.1: Przypisania wyprowadzeń wykorzystane w projekcie w oknie Pin Planner

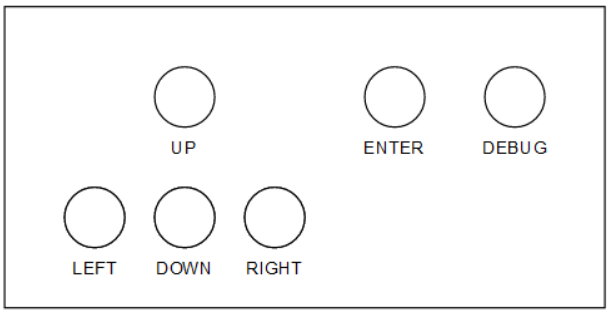
Miejsce, do którego połączone jest każde wyprowadzenie, co determinuje jego lokację na układzie FPGA, zostało odczytana z dokumentacji zestawu MAXimator [KL]. Większość wyprowadzeń pracuje jako zwykłe porty cyfrowe wejścia lub wyjścia, co sygnalizuje wartość „2.5V” w kolumnie „I/O Standard”. Wyjątek stanowią wyjścia związane z portem HDMI, które pracują w trybie różnicowym, opisanym dokładniej w podrozdziale 5.7. W związku z tym tryb pracy tych wyjść ustawiono jako LVDS_E_3R, gdzie LVDS oznacza Low Voltage Differential Signal [Int21b].

Rozdział 4

Specyfikacja zewnętrzna

4.1 Sterowanie i uruchomienie urządzenia

Sterowanie gry odbywa się poprzez zestaw przycisków znajdujących się na płytce stykowej połączonej poprzez krótkie przewody z zestawem MAXimator. Rozmieszczenie i nazwy przycisków zaprezentowano na Rys. 4.1.



Rys. 4.1: Układ przycisków do sterowania gry

Działanie większości z przycisków zależy od kontekstu, w jakim zostały naciśnięte. W każdym z kolejnych podrozdziałów, reprezentujących kolejne stany w jakich znajduje się gra, działanie to zostało szczegółowo omówione. Pewien wyjątek stanowi przycisk DEBUG, który nie służy do sterowania grą, lecz pozwala na włączenie trybu Debug ułatwiającego wykrywanie błędów podczas tworzenia gry. Jego wciśnięcie powoduje zapalenie się małego czerwonego kwadratu w lewej górnej części ekranu, sygnalizującego włączenie tego trybu. Po przejściu do trybu Debug, naciśnięcie odpowiedniej sekwencji pozostałych przycisków pozwala na bezpośrednią modyfikację wielu parametrów gry. Modyfikacji tej można dokonać, będąc w dowolnym stanie gry, bez potrzeby wchodzenia do menu opcji oraz pomijając ustalone w kodzie zakresy, z jakich te parametry mogą przyjmować wartości. Funkcja ta jednak powstała głównie w celach testowych i użytkownik nie powinien z niej korzystać. Wprowadzanie modyfikacji opcji w tym trybie może doprowadzić do sytuacji, gdzie niemożliwym jest doprowadzenie do wygranej, np.

poprzez zmniejszenie ilości dostępnych kolorów pinów w trakcie gry. Odradza się więc korzystanie z tego przycisku.

Płytka stykowa oprócz przycisków wyposażona jest także w głośniczek. Wydaje on dźwięk przy każdym naciśnięciu przycisku oraz odgrywa krótką melodię, kiedy partia gry zostanie zakończona. Można regulować głośność głośnika dostępną obok gałką potencjometru.

Proces uruchamiania samej gry jest bardzo prosty. Należy jedynie podłączyć płytkę MAXimator do zasilania, poprzez dostępne na płytce złącze USB, i gra automatycznie zacznie działać. Aby wyświetlić obraz generowany przez urządzenie należy podłączyć płytkę do monitora poprzez interfejs VGA i/lub HDMI. Identyczny obraz generowany jest cały czas przez obydwa te interfejsy, więc wybór ten nie ma znaczenia. Po uruchomieniu gry użytkownik zostaje powitany przez menu główne. Obraz ma stałą rozdzielczość 640x480. Niestety, może się zdarzyć, że interfejs HDMI będzie wyświetlać obraz w gorszej jakości niż VGA. Z powodów opisanych w podrozdziale 6.3 dotyczącym analizy generowanego obrazu HDMI, można zaobserwować np. brakujące kolumny pikseli. Zaleca się korzystanie z VGA jeśli jest to możliwe.

4.2 Menu Główne

Na środku ekranu widnieje tytuł gry, czyli MasterMind. Użytkownik, posługując się przyciskami UP oraz DOWN ma możliwość wyboru jednego z trzech dostępnych trybów gry, opisanych szczegółowo w podrozdziale „Rozgrywka”, lub przejścia do menu opcji. Aktualnie wybrany wiersz zostaje wyróżniony poprzez zmianę koloru jego czcionki i tła. Wybór użytkownik zatwierdza przyciskiem ENTER.

4.3 Menu Opcji

Menu opcji widoczne jest po wyborze odpowiedniego wiersza w menu głównym. Składa się z listy opcji. Jedynym wyjątkiem jest pierwszy element w liście, o nazwie „Back”, którego zatwierdzenie wyboru klawiszem ENTER powraca do menu głównego. Każda z pozostałych opcji modyfikuje parametry rozgrywki lub sposób wyświetlania obrazu na ekranie. Posługując się przyciskami UP oraz DOWN użytkownik wybiera opcję którą chce aktualnie zmodyfikować. Każda z opcji przyjmuje jako wartość liczbę całkowitą z odpowiedniego zakresu. Klawisze LEFT oraz RIGHT służą do zmniejszania i zwiększania o jeden wartości aktualnie wybranej opcji. Dodatkowo, po naciśnięciu

przycisku ENTER przechodzi się w tryb edytowania kolejnych cyfr opcji. W tym trybie aktualnie wybrana cyfra zacznie migać. Klawisze LEFT i RIGHT wtedy zamiast modyfikować wartość liczby, zmieniają to, która cyfra jest aktualnie wybrana, przesuwając w lewo lub w prawo wskaźnik wyboru pomiędzy kolejnymi cyframi. Modyfikacja wartości odbywa się poprzez klawisze UP i DOWN. Jeśli aktualnie wybrana jest cyfra jedności, to naciśnięcie UP lub DOWN zwiększy lub zmniejszy wartość opcji o 1, jeśli wybrana jest cyfra dziesiątek, to zwiększy lub zmniejszy wartość o 10, itd. Tryb ten jest pomocny w modyfikowaniu dużych liczb o dużą wartość, gdyż znacząco zmniejsza potrzebną liczbę naciśnięć klawiszy, a co za tym udzie też i czas potrzebny na zmodyfikowanie opcji. Jeśli po modyfikacji dowolnej z opcji jej wartość wyszłaby poza przewidywany zakres, to automatycznie zostanie przywrócona do maksymalnej lub minimalnej wartości z tego zakresu, zależnie w którą stronę zakres został przekroczony. W Tab. 4.1 przedstawiono przeznaczenie każdej z dostępnych opcji.

Tab. 4.1: Opis dostępnych do modyfikacji opcji w menu opcji

Nazwa Opcji	**Zakres**	**Wartość Domyślna**	**Działanie**
Pin Colors	2 - 21	6	Liczba kolorów jakie mogą przyjmować piny.
Pins Count	2 - 20	4	Liczba pinów w sekwencji.
Guesses	1 - 99	20	Liczba dozwolonych prób odgadnięcia sekwencji
Pixel Width Pixel Height	1 - 25	3	Rozmiar pikseli czcionki rastrowej, z jakich składają się napisy wyświetlane na ekranie.
Palette	0 - 4	0	Indeks aktualnie wybranej palety kolorów

Pierwsze 3 opcje odnoszą się bezpośrednio do rozgrywki, a 3 ostatnie jedynie do wyświetlania. Modyfikacja Pixel Width oraz Pixel Height sprawia, że pozycje większości elementów na ekranie są automatycznie przeliczane tak, aby zapewnić możliwie estetyczny i przejrzysty wygląd. Niestety, nie zawsze jest to możliwe, co powoduje, że niektóre ciągi znaków, jak np. nazwy opcji w menu opcji po ustawieniu zbyt dużej szerokości pikseli, mogą zostać zasłonięte przez inne, ważniejsze do wyświetlenia elementy, lub po prostu nie mieszczą się w całości na ekranie. Niezależnie jednak od rozmiaru pikseli, gra zawsze będzie w pełni funkcjonalna.

Modyfikacja palety kolorów zmienia elementy takie jak: kolor czcionki, kolor tła, kolor wybranych elementów i kolor tła wybranych elementów. Wartości tych kolorów są dobierane z jednej z wgranych pięciu palet.

Interfejs VGA na płytce MAXimator pozwala na wyświetlenie jedynie 8 kolorów, a jak można zauważyć, maksymalna dostępna do ustawienia ilość kolorów pinów to 21. Aby zwiększyć ilość dostępnych kolorów powyżej 8 zdecydowano się na rysowanie niektórych

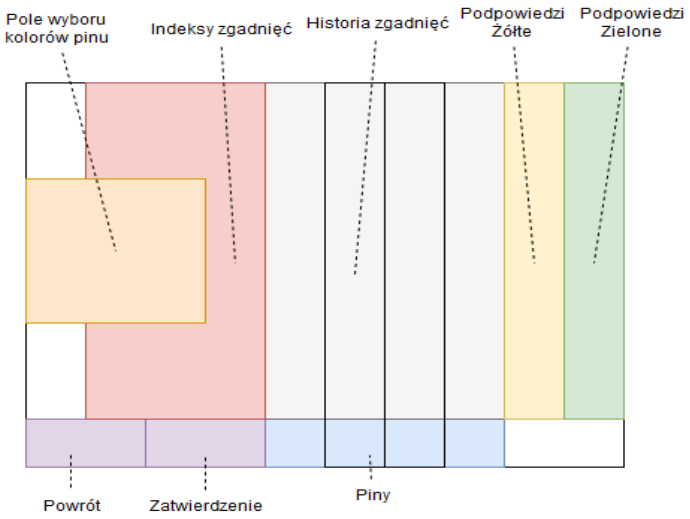
pinów jako kombinacji 2 kolorów, gdzie każdy wiersz pinu wyświetlany jest jako jeden z tych 2 wybranych kolorów. Przy zarezerwowaniu jednego koloru jako kolor tła i jednego koloru jako kolor zaznaczenia, i przy założeniu, że kolejność kolorów nie ma znaczenia, daje to $6+\frac{6\cdot 5}{2}=21$ możliwych kolorów.

4.4 Rozgrywka

Zaimplementowane zostały 3 tryby rozgrywki:

- Random
  Jedyny z trybów pozwalający na grę tylko jednemu graczowi. W tym trybie ukryta sekwencja losowana jest przez urządzenie. Zadaniem gracza jest jej odgadnięcie, jak w zwyczajnym Mistrzu Intelektu. Urządzenie automatycznie liczy i prezentuje graczowi podpowiedzi dla każdej próby. Jeśli po liczbie prób zdefiniowanych w menu opcji pod nazwą Guesses gracz nie odgadł pełnej sekwencji, to przegrywa.
- Custom
  W odróżnieniu od poprzedniego trybu, tutaj sekret nie jest losowany, ale ustawiany manualnie przez drugiego gracza, zwanego Setter'em, podczas gdy gracz zgadujący, zwany Guesser'em, nie patrzy na ekran. Reszta rozgrywki przebiega podobnie jak w trybie Random. Jeśli Guesser'owi uda się odgadnąć sekret w ustalonej maksymalnej liczbie dostępnych prób, to wygrywa, a jeśli nie, to wygrywa Setter.
- Rival
  Tryb ten działa podobnie jak Custom. Rozgrywka rozpoczyna się dobraniem sekretu przez Setter'a. Różnica od poprzednich trybów polega na tym, że teraz urządzenie nie podaje wartości automatycznie wyliczonych podpowiedzi. Zadaniem Setter'a jest zapamiętanie sekwencji jaką wybrał, i policzenie w głowie wartości żółtych i zielonych podpowiedzi, a następnie wprowadzenie ich manualnie po zatwierdzeniu próby odgadnięcia przez Guesser'a. Jeśli Setter się pomyli w wartościach podpowiedzi, to Guresser wygrywa. Taki tryb wprowadza większy dynamizm do rozgrywki, gdzie jeden z graczy w trybie Custom uczestniczył w grze wyłącznie na jej początku, a potem jedynie mógł się przyglądać poczynaniom rywala.

Niezależnie od wybranego trybu gry, rozkład elementów na ekranie jest zawsze taki sam. Rozkład ten przedstawiono na Rys. 4.2.



Rys. 4.2: Rozkład elementów na ekranie rozgrywki

Użytkownik zaznacza kolejne elementy na ekranie korzystając z przycisków LEFT i RIGHT. Możliwe do zaznaczenia pola to:

- Powrót
  Pole to znajduje się w lewym dolnym rogu ekranu. Zatwierdzenie jego wyboru przyciskiem ENTER powoduje powrót do menu głównego i anulowanie aktualnie rozgrywanej gry.
- Zatwierdzenie
  Pole to znajduje się na prawo od przycisku powrotu. Zatwierdzenie jego wyboru przyciskiem ENTER zależy od kontekstu. Jeśli jest się Guesser'em, to powoduje to wykonanie próby odgadnięcia sekretu. Do historii zgadnięć dopisany zostaje nowy wpis, z aktualną sekwencją kolorów pinów ustawioną w polach pinów. Jeśli jest się aktualnie Setter'em wybierającym ukrytą sekwencję, to zatwierdza on sekwencję ustawioną w polu pinów jako sekret, a następnie ustawia kolory pinów na domyślne, oczywiście po to aby Guesser nie znał sekretu. Jeśli aktualnie jest się Setter'em podającym wartości podpowiedzi w trybie Rival, to pole to zatwierdza aktualnie podaną wartość podpowiedzi.
- Piny
  Jest to zbiór pól reprezentujących kolorowe piny. Zaznaczenie dowolnego z nich podświetla tło całej kolumny historii zgadnięć w celu łatwiejszej orientacji. Naciśnięcie przycisku ENTER po zaznaczeniu danego pinu przechodzi do trybu

wyboru koloru. Po lewej stronie ekranu wyświetlone zostaje pole ze wszystkimi dostępnymi kolorami pinów. Za pomocą przycisków LEFT, RIGHT, UP i DOWN należy zaznaczyć kolor, jaki się chce by przyjął aktualny pin. Ponowne naciśnięcie klawisza ENTER powoduje ustawienie wybranego pinu na wybrany kolor i zamyka pole wyboru koloru.

Jeśli wybrany został tryb Rival, to po zatwierdzeniu pinów przez Guesser'a, pojawia się na górze ekranu komunikat tekstowy proszący Setter'a o podanie wartości żółtych, a potem zielonych podpowiedzi. Zmiana tych wartości odbywa się poprzez przyciski UP i DOWN.

Ponieważ liczba dostępnych prób odgadnięcia najprawdopodobniej będzie przekraczać ilość linijek jakie się mieszczą w historii zgadnięć, możliwe jest przewijanie tej historii za pomocą przycisków UP oraz DOWN. Przewijanie to działa w dowolnym momencie, za wyjątkiem wybierania kolorów pinów oraz podawania wartości podpowiedzi w trybie Rival. Kolejne próby odgadnięcia dopisywane są na szczyt historii, i jeśli aktualnie nie ma już wolnego miejsca na ekranie, a nowe zgadnięcie zostało zatwierdzone, to następuje automatyczne przewinięcie historii do góry, aby widoczne było przynajmniej jedno wolne miejsce.

Zakończenie rozgrywki odbywa się poprzez wyświetlenie na ekranie komunikatu „You Win" lub „Game Over" dla trybu Random, albo „Guesser Wins" lub „Setter Wins" dla pozostałych trybów. Dodatkowo, na ekranie wyświetlony zostaje na losowej pozycji animowany wzór reprezentujący wybuchające fajerwerki.

Rozdział 5

Specyfikacja wewnętrzna

5.1 Ogólna struktura kodu

Projekt składa się z wielu modułów napisanych w języku SystemVerilog, gdzie każdy z nich odpowiada za jakieś konkretny fragment logiki całego systemu. Niezbędnym jest, aby wszystkie te moduły mogły między sobą wymieniać generowane informacje, które to często są bardziej skomplikowane niż pojedyncze liczby, a także muszą być sparametryzowane w kompatybilny sposób. W tym celu stworzono plik „common.sv", który agreguje definicje typów strukturalnych danych i stałych parametrów wykorzystywanych w większości modułów.

Najważniejszym typem danych zdefiniowanym w tym pliku jest struktura „st_GAME_STATE", której to elementy jednoznacznie reprezentują prawie całość stanu całego systemu. Zawartość tej struktury opisano w tabeli 5.1.

Tab. 5.1: Elementy struktury st_GAME_STATE

Nazwa	Opis
state_name	Wartość typu enum, określająca w jakim ogólnym stanie aktualnie znajduje się system. Zależnie od tej wartości, wykonywany jest inny fragment kodu odpowiedzialny za logikę gry i generowanie obrazu. Przyjmuje jedną z następujących wartości: • GS_MAIN_MENU – użytkownik znajduje się w menu głównym • GS_OPTIONS – użytkownik znajduje się w menu opcji • GS_GAME – użytkownik rozgrywa partię Mistrza Intelektu • GS_GENERATE_PINS – aktualnie trwa generowanie losowej sekwencji kolorów sekretu.
navigation	Struktura typu „st_GS_NAVIGATION". Określa informację o aktualnym indeksie wybranego elementu przez użytkownika. Dokładne znaczenie wartości pól w tej strukturze zależy od wartości state_name.
options	Struktura typu „st_GS_OPTIONS". Przechowuje wartości wszystkich opcji możliwych do modyfikacji przez użytkownika na ekranie menu opcji.

render	Struktura typu „st_GS_RENDER”. Przechowuje wyliczone wartości potrzebne przy generowaniu obrazu, takie jak pozycje pikseli od których należy rysować dany element albo kolory konkretnych elementów. Wiele z tych wartości jest ściśle zależne od wartości opcji z menu opcji, dlatego aby uniknąć konieczności wyznaczania ich wielokrotnie przy każdej potrzebie ich użycia, struktura ta służy do przechowywania tych wartości po ich jednokrotnym wyliczeniu jedynie w momencie zmiany jednej z opcji.
board	Struktura typu „st_GS_BOARD”. Przechowuje stan o aktualne rozgrywanej partii Mistrza Intelektu. Są to takie informacje jak np. wybrany tryb gry, treść aktualnie wyświetlanego komunikatu tekstowego, aktualne przewinięcie historii zgadnięć.
firework	Struktura typu „st_GS_FIREWORK”. Przechowuje stan animacji fajerwerków, o ile mają być aktualnie wyświetlane ze względu na zwycięstwo jednego z graczy.

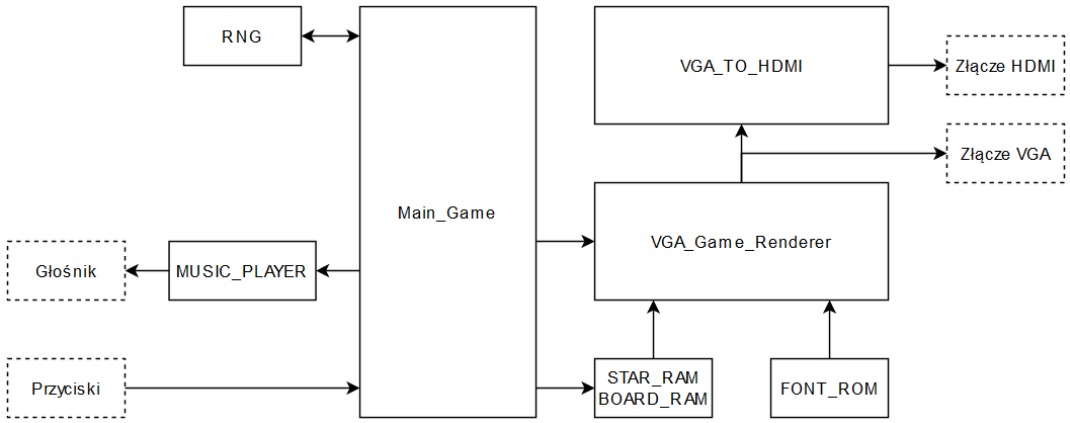
Oprócz struktury st_GAME_STATE, plik ten definiuje też wartości wielu stałych parametrów, jakimi są np. rozmiary (w bitach) różnych zmiennych, wartości minimalne i maksymalne jakie mogą przyjmować opcje z menu opcji, niezmienne parametry generacji obrazu VGA, wykorzystywane palety barw. Posiadanie definicji tych wszystkich elementów w jednym pliku pozwala na łatwą ich ewentualną modyfikację w kodzie, jeśli miałaby zajść taka potrzeba. Niweluje to potrzebę przeglądania całego projektu i poszukiwania każdego wystąpienia danego parametru, ponieważ każdy inny moduł odwołuje się do parametrów zdefiniowanych w pliku „common.sv”.

Niestety, niemożliwym jest przechowanie dosłownie całego stanu gry w strukturze st_GAME_STATE. Przykładowo, przy założeniach maksymalnej ilości prób odgadnięcia sekretu (99) i ilości pinów (20), urządzenie powinno być w stanie przechować każdą unikalną kombinacje kolorów pinów jaka została odgadnięta, w celu ich potencjalnego wyświetlenia w historii zgadnięć. Jeśli kolor pinu zapisywany jest jako wartość 5-bitowa (minimalna ilość bitów mogąca przyjąć 21 wartości, czyli maksymalną ilość kolorów), to potrzeba co najmniej 9900 bitów pamięci. Przechowywanie takiej ilości danych bezpośrednio w rejestrach zastosowanego układu FPGA, posiadającego jedynie 8000 jednostek logicznych, nie jest możliwe. Dlatego należało skorzystać z modułów typu RAM (Random Access Memory), których to parametryzowalne definicje udostępnia Quartus Prime Lite. Moduł ten wykorzystuje inny sektor układu FPGA, przystosowany właśnie do przechowywania dużej ilości danych. Podobnie postąpiono z modułem pamięci ROM (Read Only Memory). Różni się on tym od RAM, że nie pozwala na zapis danych, a jedynie na ich odczyt. Wykorzystywany jest więc do przechowywania dużej ilości niezmiennych danych, jakimi są znaki czcionki.

W Tab. 5.2. zamieszczono opis wykorzystanych w projekcie modułów, a na Rys. 5.1 zamieszczono schemat reprezentujący ich logiczne powiązanie.

Tab. 5.2: Opis modułów projektu

Nazwa modułu	**Opis modułu**
Main_Game	Główny moduł projektu. W nim znajdują się instancje wszystkich innych modułów wykorzystywanych w projekcie. Odpowiada za główną logikę gry i odczytywanie danych wejściowych użytkownika. Jedynie wewnątrz tego modułu dokonywana jest modyfikacja struktury „st_GAME_STATE”.
VGA_Game_Renderer	Odpowiada za generowanie sygnałów interfejsu VGA, na podstawie aktualnej wartości struktury „st_GAME_STATE”, podawanej jako wejście.
VGA_TO_HDMI	Odpowiada za przetworzenie sygnałów VGA wygenerowanych przez moduł „VGA_Game_Renderer”, aby można było ten sam obraz wysłać przez interfejs HDMI.
RNG	Odpowiada za generowanie liczb pseudolosowych.
MUSIC_PLAYER	Odpowiada za odgrywanie dźwięków na głośniku.
STAR_RAM	Moduł RAM, 512 słów 64-bitowych, odpowiadający za przechowywanie informacji o aktualnym stanie migających „gwiazdek” w tle gry.
BOARD_RAM	Moduł RAM, 4096 słów 8-bitowych, odpowiadający za przechowywanie historii prób odgadnięcia sekretu, wraz z przypisanymi im wartościami podpowiedzi
FONT_ROM	Moduł ROM 512 słów 4-bitowych, odpowiadający za przechowywanie czcionki wykorzystywanej do wyświetlania napisów i niektórych kształtów



Rys. 5.1: Schemat logiczny zależności modułów

Cały projekt wykorzystuje 7144 z 8064 (~89%) dostępnych jednostek logicznych oraz 39424 z 387072 (~10%) dostępnych jednostek pamięci.

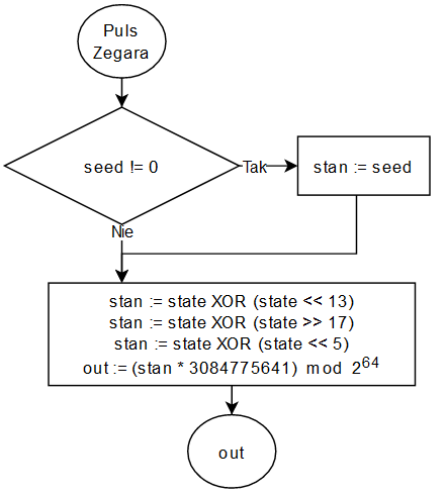
5.2 Generowanie sygnałów zegarowych

W projekcie wykorzystany jest moduł PLL (Phase Locked Loop). Jest to domyślnie zdefiniowany moduł odpowiedzialny za generowanie sygnału zegarowego o zadanej częstotliwości. Przetwarza on sygnał zegarowy obecny na odpowiednim wyprowadzeniu wejściowym układu FPGA (w przypadku zestawu MAXimator jest on generowany przez oscylator o częstotliwości 10MHz), modyfikując go poprzez zaimplementowane dzielniki i mnożniki częstotliwości, o konfigurowalnych wartościach. Sygnałem wyjściowym jest więc sygnał zegara o częstotliwości, będącej częstotliwością wejściową przemnożoną i podzieloną przez dane liczby całkowite. W przypadku tego projektu, wygenerowano 2 sygnały zegara: 1MHz, wybrany arbitralnie, oraz 252MHz, potrzebny do generacji obrazu VGA oraz HDMI.

5.3 Generowanie liczb losowych

W układach elektronicznych bardzo trudno uzyskać jest prawdziwą losowość. Bez zastosowania zewnętrznych urządzeń, które wykorzystują zjawiska związane z mechaniką kwantową lub teorią chaosu jest to praktycznie niemożliwe. Można jednak skorzystać z generatora liczba pseudolosowych, nazywanego także jako PRNG (Pseudo-Random Number Generator). Jest to deterministyczny algorytm, który w każdej swojej iteracji wykonania zwraca liczbę o danej ilości bitów, która tylko z pozoru jest losowa. Jeśli znałoby się w pełni stan wewnętrzny generatora, można by przewidzieć każdą przyszłą wygenerowaną liczbę. Na szczęście na potrzeby projektu nie potrzeba bardziej zaawansowanych algorytmów. Istnieje wiele różnych generatorów liczb pseudolosowych. Ze względu na prostotę w implementacji i nienajgorszą jakość losowości wybrano algorytm „XorshiftStar” [EA21], którego implementację zamieszczono na Rys. 5.2.

W module „Main_Game” zdefiniowano licznik 32-bitowy, nazywany dalej „time_counter”, który zwiększa swoją wartość wraz z każdym impulsem zegara 1MHz. Jako stan początkowy generatora, tak zwany „seed”, ustawiana jest wartość licznika „time_counter” w momencie pierwszego rozpoczęcia gry w trybie Random. Liczby losowe wykorzystywane są do generowania sekretu w trybie Random oraz do animowania efektów graficznych, jakimi są kolorowe gwiazdy w tle oraz fajerwerki. Wraz z każdym impulsem zegara 1MHz generowana jest kolejna losowa liczba, której bity, zależnie od kontekstu, interpretowane są jako kolor danego pinu sekretu albo pozycja gwiazdy lub fajerwerków w tle.



Rys. 5.2: Schemat blokowy algorytmu XorshiftStar

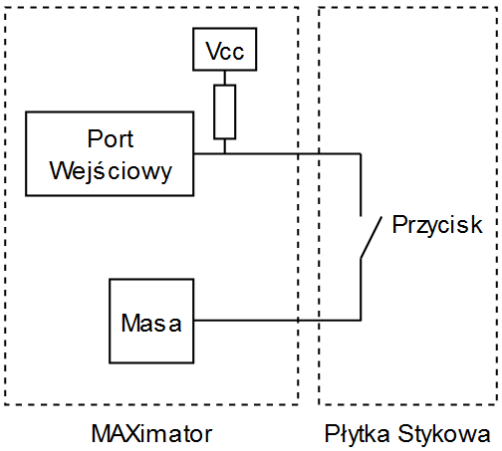
5.4 Inicjalizacja pamięci ROM

Inicjalizacja pamięci ROM odbywa się na etapie kompilacji projektu, poprzez odpowiedni plik konfiguracyjny w formacie „mif", reprezentujący strukturę bitów danego modułu pamięci. Moduł „FONT_ROM" wykorzystywany w projekcie powinien przechowywać wygląd czcionki rastrowej wykorzystywanej do ciągów znaków wyświetlanych na ekranie. Wybrano rozmiar czcionki jako 4x5 pikseli. Plik „mif" można stworzyć ręcznie, wprowadzając każdy znak ręcznie, lecz byłoby to niewygodnie i wprowadzałoby wiele problemów przy potencjalnej modyfikacji np. kolejności znaków lub ich rozmiaru. W celu ułatwienia tego zadania, stworzono skrypt w języku JavaScript, o nazwie „generate_font_mem_init.js", wewnątrz którego w przejrzysty sposób zdefiniowano tablicę reprezentującą wszystkie znaki czcionki. Skrypt oprócz scalenia wszystkich znaków i przetworzenia ich do formy liczb binarnych wpisanych do pliku „mif", wyznacza także wiele innych parametrów przydatnych w zintegrowaniu zawartości ROM-u z pozostałymi modułami. Parametrami tymi są adresy każdego znaku, ilość zdefiniowanych znaków, rozmiar szyny adresowej i rozmiar czcionki. Dodatkowo, w „generate_font_mem_init.js" zdefiniowano wszystkie ciągi znaków, jakie mogą być wyświetlane. Skrypt automatycznie generuje parametry typu tablicowego w SystemVerilog oraz długości tych ciągów znaków, zawierające indeksy kolejnych znaków w ciągu. Domyślnie znaki te zdefiniowane są tak aby były wyświetlane na ekranie z przerwą o szerokości 1 piksela pomiędzy nimi. Istnieją jednak znaki, jak np. litera „M" w tytule „MasterMind", których rozmiar wychodzi ponad założone pole 4x5. Dla takich znaków, nazwanych w skrypcie jako „Long Sprites", generowana jest dodatkowo tablica składająca się z zer i jedynek, zawierająca informacje, po których znakach w ciągu znaków

ma zostać rysowany odstęp, a po których nie. Wszystkie te parametry zostają zapisane w pliku „generated_params.vh”, który jest plikiem nagłówkowym dołączonym do pliku „common.sv”. Skrypt taki uruchamiany powinien być przy każdej modyfikacji zawartości ROM-u lub wyświetlanych ciągów znaków, przed kompilacją projektu. Generowanie tych parametrów do pliku nagłówkowego pozwala na bardzo łatwą podmianę zawartości ROM-u, a nawet zmianę całkowitą rozmiaru czcionki, nie tracąc w żaden sposób kompatybilności z wszystkimi modułami wykorzystywanymi w projekcie.

5.5 Obsługa przycisków

Przyciski połączone są z modułem MAXimator poprzez porty wejścia/wyjścia ogólnego przeznaczenia. Zgodnie ze schematem płytki MAXimator [KL], porty te połączone są z translatorem poziomu napięcia XS0108EPWR. W nocie katalogowej tego układu [TI07] widać, że wyprowadzenia wejścia/wyjścia połączone się z rezystorem Pull-Up. Oznacza to, że w przypadku niepodłączenia danego portu wejściowego, odczytana wartość będzie logiczną jedynką. Aby odczytać logiczne zero, należy zewrzeć dane wyprowadzenie z uziemieniem. Pozwala to więc na połączenie przycisków w bardzo prosty sposób przedstawiony na Rys. 5.3.



Rys. 5.3: Schemat połączenia przycisków do układu MAXimator

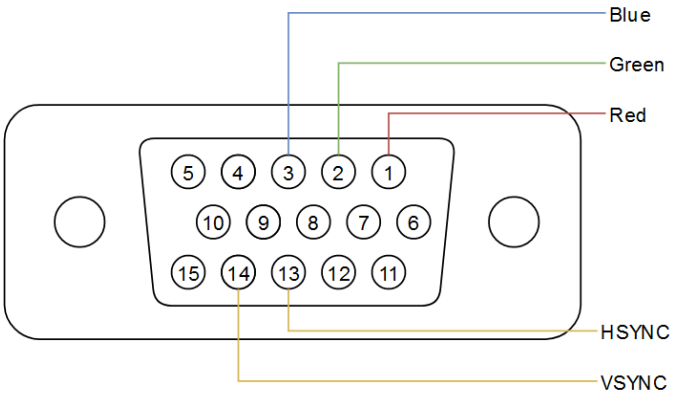
Samo połączenie przycisku nie gwarantuje poprawnego odczytania jego stanu. Wiele funkcji gry opiera się na wykrywaniu zbocza sygnału sterowalnego przez dany przycisk. Ze względu na mechaniczną naturę przycisków, podczas naciskania przycisku wewnętrzne metalowe styki wykonują wiele bardzo szybkich oscylacji, odbijając się od siebie, zanim wejdą w stały kontakt. Powoduje to wiele szybko następujących po sobie zmian stanu portu wejściowego, a co za tym idzie, wiele wykrytych zboczy sygnału. Aby zniwelować to niepożądane zjawisko, należy skorzystać z techniki zwanej „Debouncingiem”. Polega

ona na stworzeniu nowego modułu, którego wejściem będzie sygnał z przycisku. Innym wejściem jest też sygnał zegara, służący do odmierzenia czasu. Jeśli przez zadaną ilość taktów sygnału zegarowego sygnał wejściowy nie zmienił stanu, to dopiero wtedy jest on przepisywany na wyjście danego modułu. Dopiero w tak przetworzonym sygnale dokonywane jest wykrywanie zbocza, poprzez sprawdzenie czy w kolejnych taktach sygnału zegarowego stan sygnału wejściowego się nie zmienił. Liczba impulsów sygnału zegarowego po jakiej moduł „Debounce" uznaje, że sygnał wejściowy jest stabilny, nie może być zbyt mała, ponieważ nie zniwelowałoby to wszystkich odbić przycisku, ani zbyt duża, ponieważ wprowadzałoby to zauważalne dla użytkownika opóźnienie. Empirycznie stwierdzono, że okres 2^{11}, czyli 2048 pulsów zegara 1MHz, co odpowiada około 2ms, jest optymalny.

5.6 Generowanie obrazu VGA

Generowanie obrazu VGA odbywa się w module „VGA_Game_Renderer". Jednym z wejść modułu jest struktura „st_GAME_STATE". Na podstawie wartości elementów tej struktury wykonywana jest odpowiednia gałąź kodu odpowiedzialnego za generowanie obrazu.

Na interfejs VGA składają się 5 ważnych sygnałów, przestawionych na Rys. 5.4.

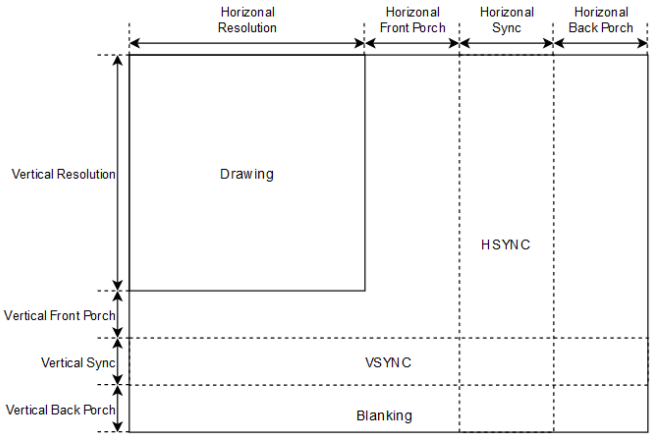


Rys. 5.4: Schemat wyprowadzeń portu VGA

Kolory kolejnych pikseli wysyłane są szeregowo, jako sygnały analogowe poprzez linie Red, Green i Blue. Ponieważ płytka MAXimator nie pozwala na sterowanie poziomem napięcia żadnego z tych sygnałów, a jedynie traktuje je jako sygnały cyfrowe, daje nam to łącznie 8 możliwych do wyświetlenia kolorów: czarny, czerwony, zielony, niebieski, cyjan, magenta, żółty i biały. Częstotliwość nadawanych pikseli zależy od wybranej rozdzielczości, a zegar o tej częstotliwości nazywany jest Zegarem Pikseli, lub

„Pixel Clock”. Sygnały HSYNC oraz VSYNC służą do przesłania informacji o rozdzielczości oraz synchronizacji z monitorem.

Przesyłanie obrazu poprzez interfejs VGA składa się z dwóch faz: fazy rysowania oraz fazy „Blanking”. Podczas fazy rysowania przesyłane są kolory pikseli, wykorzystując sygnały Red, Green oraz Blue. W fazie „Blanking” sygnały odpowiedzialne za kolory są zerowane, i dodatkowo sygnały HSYNC i VSYNC są odpowiednio włączane lub wyłączane, co przedstawiono na Rys. 5.5.



Rys. 5.5: Schemat faz generowania sygnału VGA

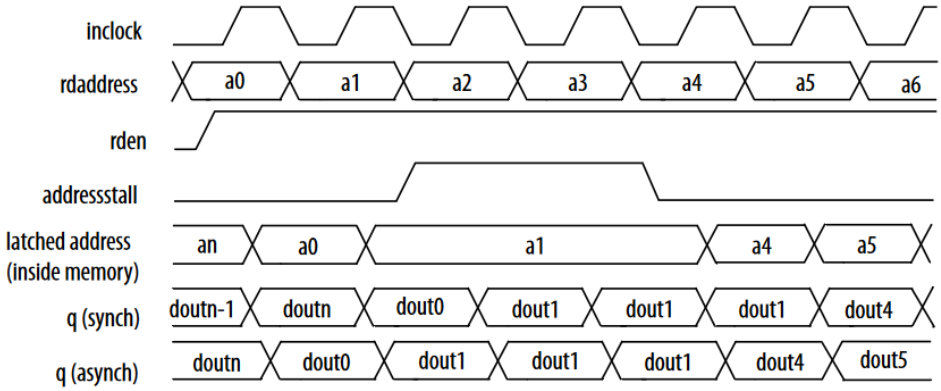
Odpowiednie dobranie czasów trwania każdej z faz i podfaz są jednoznacznie powiązane z powszechnie ustandaryzowanymi rozdzielczościami obsługiwanymi przez interfejs VGA [VE13]. Stworzony projekt jest w stanie skorzystać praktycznie z dowolnego zestawu parametrów każdej z faz. Zmiana rozdzielczości wymaga jedynie zmiany odpowiednich stałych parametrów w pliku „common.sv” oraz ustawienia odpowiedniej częstotliwości zegara PLL. Domyślnie zastosowana rozdzielczość to 640x480 z odświeżaniem 60Hz, dla której wartości poszczególnych parametrów zamieszczono w Tab. 5.3.

Tab. 5.3: Zastosowane parametry czasowe sygnału VGA

Parametr	**Wartość**	**Jednostka**
Częstotliwość Zegara Pikseli	25.175	MHz
Horizontal Resolution	640	Cykle Zegara Pikseli
Horizontal Front Porch	16	Cykle Zegara Pikseli
Horizontal Sync	96	Cykle Zegara Pikseli
Horizontal Back Porch	48	Cykle Zegara Pikseli
Vertical Resolution	480	Linie obrazu
Vertical Front Porch	10	Linie obrazu
Vertical Sync	2	Linie obrazu
Vertical Back Porch	33	Linie obrazu

Niestety, wbudowany w wykorzystanym układzie FPGA moduł PLL nie jest w stanie wygenerować sygnału zegarowego o częstotliwości dokładnie 25,175MHz. Na szczęście, zbliżona wartość częstotliwości 25,2MHz jest możliwa do wygenerowania i działa bez problemów, dlatego też z takiej skorzystano. Dodatkowo, warto dodać, że zegar ten nie jest generowany bezpośrednio przez PLL. PLL generuje zegar o częstotliwości 10x większej, czyli 252MHz, potrzebny do wygenerowania sygnału HDMI. Zegar ten jest wejściem osobnego dzielnika częstotliwości, który przetwarza go na wykorzystywany sygnał 25,2MHz. Podczas fazy testów, jeśli oba zegary 25,2MHz i 252MHz generowane były przez PLL, to napotkano wyraźne błędy w wyświetlaniu sygnału HDMI, dlatego zdecydowano się na osobny moduł dzielnika częstotliwości.

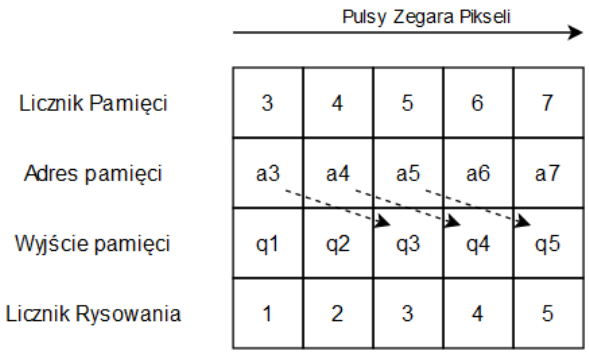
Moduł „VGA_Game_Renderer" taktowany jest sygnałem zegarowym 25,2MHz. Z każdym impulsem sygnału zegarowego, zwiększane są liczniki pozycji poziomej i pionowej, reprezentujące kolejne fazy generowania obrazu, i na podstawie ich wartości i stanu struktury „st_GAME_STATE", jeśli trwa aktualnie faza rysowania, generowany jest kolor aktualnie wyświetlanego piksela. Generowanie koloru piksela w tym samym takcie sygnału zegarowego, w którym ma być wyświetlony stwarza niestety pewien problem. Dostęp do danych przechowywanych w modułach RAM i ROM wymaga 3 impulsów sygnału zegarowego, co przedstawiono na Rys. 5.6, gdzie przykładowo wartość zapisana pod adresem „a1" pojawia się na wyjściu „q_sync" dopiero po 2 cyklach zegara taktującego pamięć.



Rys. 5.6: Schemat czasowy operacji odczytu z modułu RAM/ROM [Int21a]

Moduł „VGA_Game_Renderer" posiada definicje struktur reprezentujących 2 rodzaje liczników, nazwanych licznikami poziomymi i licznikami pionowymi. Liczniki te reprezentują wartości, takie jak aktualną fazę generowania sygnału VGA, aktualną linię/kolumnę rysowanego znaku czcionki, indeks aktualnego znaku w ciągu znaków itd. Z każdym taktem sygnału zegarowego inkrementowane są liczniki poziome, i z każdą narysowaną poziomą linią obrazu, liczniki pionowe. Ponieważ wartości zwracane przez moduły pamięci są de facto 2 cykle spóźnione, w module istnieją 3 instancje liczników poziomych, każda inkrementowana osobno i różniące się od siebie każda o jeden cykl

Zegara Pikseli od poprzedniej. Logika odpowiedzialna za wyznaczenie adresu z RAM-u lub ROM-u korzysta z pierwszego z liczników. Natomiast logika odpowiedzialna za wyznaczenie koloru piksela, który to zależny jest od danych zwróconych przez moduły pamięci, korzysta z ostatniego licznika. Jego stan jest opóźniony o 2 takty zegarowe względem stanu pierwszego licznika, co zobrazowano na Rys. 5.7.



Rys. 5.7: Schemat logiczny działania zastosowanego systemu wielu liczników

Dla liczników poziomych nie jest wymagane zastosowanie takiego opóźnienia, ponieważ czas fazy „Blanking” horyzontalnej w dowolnej konfiguracji nigdy nie jest krótszy niż 3 cykle Zegara Pikseli. W momencie przejścia do fazy „Blanking” wszystkie liczniki ustawiane są na stan reprezentujący początek kolejnej linii.

Taki system przestaje jednak działać, jeśli chcemy skorzystać z nie-bezpośredniego odwołania do pamięci. Przykładowo: wartości podpowiedzi w historii zgadnięć przechowywane są w BOARD_RAM, a wygląd cyfr przechowywany jest w FONT_ROM. Aby więc narysować odpowiednią cyfrę, należy najpierw otrzymać wartość zwrotną z modułu BOARD_RAM, a potem otrzymać wartość zwrotną z FONT_ROM, co wymagałoby łącznie 6 cykli zegara. Można ten problem rozwiązać dodając kolejną instancję liczników poziomych, tym razem opóźnionych o 6 cykli zegara. Ze względu na to, że w gotowej grze jest to jednak jedyny taki przypadek wymagający opóźnienia 6 cykli, postanowiono odwołać się do pamięci RAM w odpowiednim fragmencie fazy „Blanking”, odczytując wartości podpowiedzi dla kolejnej rysowanej linijki, i zapisując je do osobnego rejestru, do którego dostęp jest natychmiastowy. Sprawia to, że znaki, których wygląd ma zwrócić pamięć ROM, są znane przez cały czas trwania rysowania każdej linijki.

Ostatnim aspektem wartym wspomnienia jest to, że jeśli struktura „st_GAME_STATE” zmodyfikowana została w trakcie rysowania danej klatki obrazu, to obserwowane są wyraźne artefakty graficzne. Aby temu zapobiec, stworzono dwie instancje struktury „st_GAME_STATE”. Jedna z nich wykorzystywana jest przez moduł „VGA_Game_Renderer”, a jedna przez „Main_Game”. Instancja struktury w „Main_Game” modyfikowana jest bez żadnych ograniczeń, i kopiowana jest do struktury w module „VGA_Game_Renderer” jedynie wtedy, jeśli aktualnie system znajduje się w fazie „Vertical Sync”.

5.7 Generowanie obrazu HDMI

Przesyłanie obrazu HDMI jest bardzo podobne do VGA. Protokół HDMI jest wstecznie kompatybilny z protokołem DVI (Digital Visual Interface), którego to fazy są identyczne jak w przypadku VGA, dlatego można było skorzystać z tego samego kodu do generowania kolorów kolejnych pikseli oraz sygnałów HSYNC i VSYNC, jak w przypadku VGA. Różnica polega jedynie na tym, że kolory pikseli przesyłane są nie poprzez sygnał analogowy, ale jako sygnał cyfrowy. Wiąże się z tym jednak problem wymogu szybszego taktowanie zegara. Kolory każdego z 3 kanałów mają wartość 8-bitową. Nie są jednak przesyłane bezpośrednio przez odpowiednie wyprowadzenia, ale wymagają jeszcze zakodowania na 10-bitową wartość algorytmem TMDS (Transition Minimized Differential Signaling), który zapewnia większą odporność na zakłócenia. Algorytm ten jest w pełni opisany w specyfikacji HDMI [HD06].

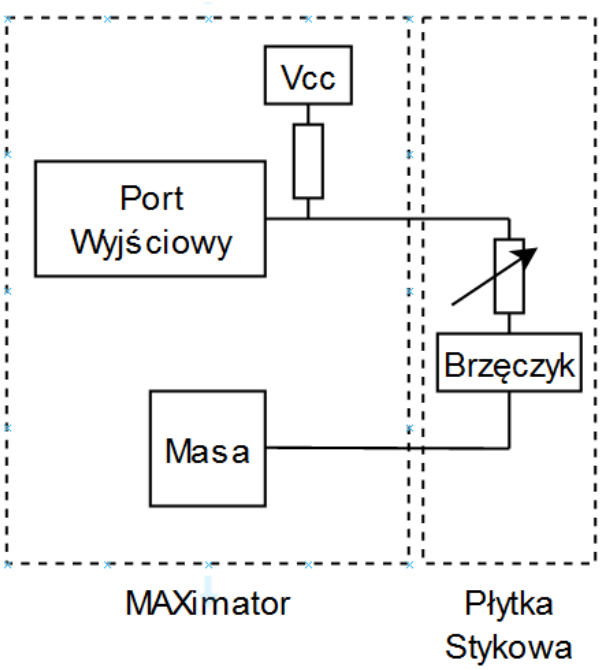
Ponieważ wysyłane wartości są 10-bitowe, potrzebny jest zegar o 10x większej częstotliwości niż Zegar Pikseli. Stąd też wartość 252MHz generowana przez PLL.

Sygnały przesyłane przez interfejs HDMI są sygnałami różnicowymi. Oznacza to, że każdy sygnał składa się z pary 2 linii sygnałowych o zawsze przeciwnych polaryzacjach. Takie podejście służy niwelacji interferencji fal elektromagnetycznych, jak i zmniejszeniu fal emitowanych przez sam przewód HDMI. Wykorzystywane są 4 sygnały przesyłane przez interfejs HDMI: Zegar Pikseli, Kanał TMDS0, Kanał TMDS1 i Kanał TMDS2, oraz odpowiadające im sygnały o przeciwnej polaryzacji. Przez Kanał TMDS0 przesyłane są wartości koloru niebieskiego oraz sygnały HSYNC i VSYNC, posiadające własne dedykowane kodowania 10-bitowe. Przez TMDS1 przesyłany jest kolor zielony, a przez TMDS2 kolor czerwony.

5.8 Generowanie dźwięku

W celu stworzenia dźwięku skorzystano z głośnika, a dokładniej brzęczyka piezoelektrycznego HPM14AX. Jeśli przez brzęczyk ten przepływać będzie prąd z jakąś częstotliwością, to wyda on dźwięk o takiej samej częstotliwości. Zastosowanie PLL do wygenerowania sygnału o odpowiedniej częstotliwości nie jest dobrym rozwiązaniem, ze względu na ilość potencjalnie potrzebnych do zagrania wysokości dźwięków. Wykorzystano więc już istniejący zegar 1MHz. Co impuls sygnału zegarowego inkrementowany jest licznik okresu trwania danej nuty. Nuty zdefiniowane są jako wyliczone wcześniej okresy drgań fal akustycznych dla wybranych standardowych klawiszy pianina. Następnie, jeśli licznik nut osiągnął połowę okresu aktualnie

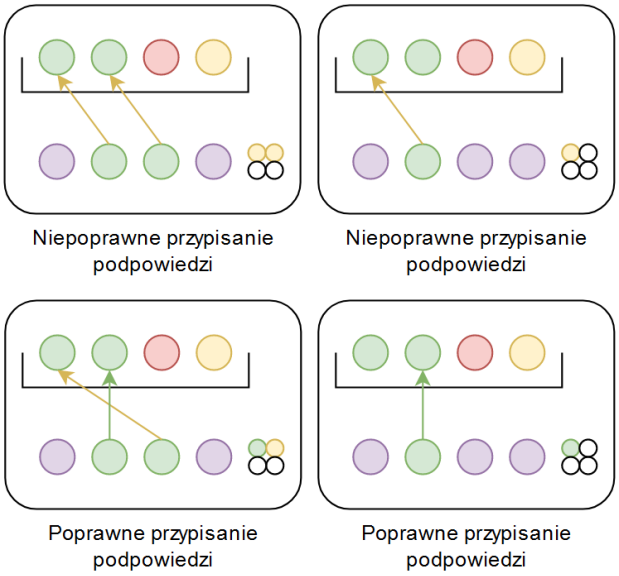
odtwarzanej nuty, następuje zmiana stanu sygnału podawanego na brzęczyk. Odtworzenie wielu nut przez zadany czas po kolei pozwala na odgrywanie dowolnej melodii. Zaimplementowano 3 różne melodie: wygranej, przegranej, i naciśnięcia przycisku. Podpięty szeregowo do brzęczyka potencjometr pozwala na regulację amplitudy napięcia, co przekłada się na głośność dźwięku. Schemat elektryczny połączenia brzęczyka zamieszczono na Rys. 5.8.



Rys. 5.8: Schemat połączenia brzęczyka do układu MAXimator

5.9 Liczenie wartości podpowiedzi

W celu wyznaczenia wartości podpowiedzi dla danej sekwencji kolorów pinów i sekretu, stworzono pomocnicze tablice, określające czy dany pin został już przypisany do którejś z podpowiedzi. Następnie należy porównać każdą kombinację pinu sekwencji ukrytej i pinu sekwencji zgadnięcia. Jeśli piny posiadają identyczny kolor i żaden z nich nie był jeszcze przypisany do podpowiedzi, to zwiększana jest wartość licznika podpowiedzi żółtych lub zielonych, zależnie czy piny znajdowały się na tej samej pozycji, a następnie oba piny ustawiane są jako już przypisane. Może się zdarzyć, że dany pin może być przypisany do zarówno podpowiedzi zielonej i żółtej. Sytuacja ta przedstawiona jest na Rys. 5.9. W takiej sytuacji wymaganym jest przypisanie podpowiedzi zielonej do tego pinu. Innymi słowy, jeśli dany pin może być przypisany do zielonej podpowiedzi, to musi być przypisany do zielonej podpowiedzi. Oznacza to, że najpierw należy wyznaczyć wszystkie podpowiedzi zielone, i dopiero spośród pinów niewykorzystanych do podpowiedzi zielonych wyznaczyć podpowiedzi żółte.



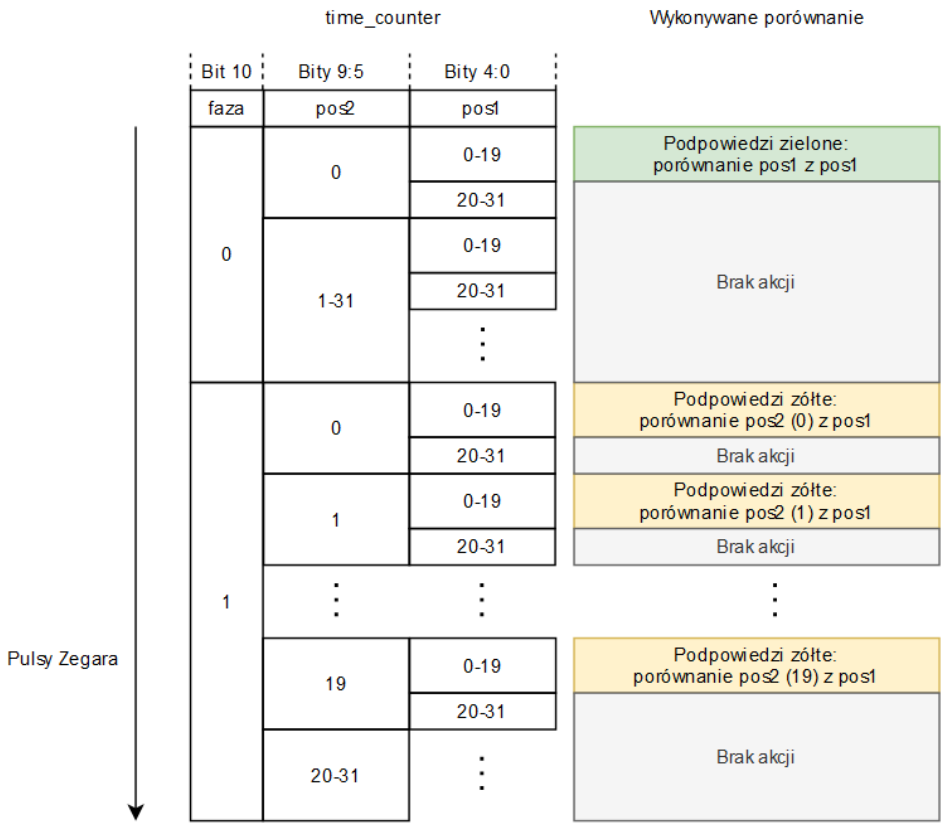
Rys. 5.9: Zobrazowanie przypisań podpowiedzi zielonych i żółtych

Początkowo naiwnie próbowano wykonać wszystkie te obliczenia w jednym cyklu zegara 1MHz. Niestety, maksymalna ilość możliwych kombinacji pinów wynosi 20^2, czyli 400. Implementacja operacji wykonania takiej liczby porównań wymagałaby prawdopodobnie kilkakrotności liczby dostępnych jednostek logicznych FPGA. Wynika to między innymi z tego, że porównań tych nie można wykonać równolegle. Przy każdym porównaniu musi być wiadome, czy w poprzednich porównaniach dany pin nie był przypisany już do żadnej podpowiedzi. Uniemożliwia to wgranie programu na urządzenie i oznacza potrzebę rozłożenia obliczeń na różne cykle zegara.

Najpierw chciano wykonać wszystkie obliczenia w czasie 20 cykli zegara. W pierwszym cyklu analizowane były piny na tych samych pozycjach, co powodowało wyznaczenie wszystkich zielonych podpowiedzi. W kolejnym cyklu analizowane były piny na pozycjach różniących się o 1, w kolejnym różnych o 2, itd., wyznaczając wszystkie podpowiedzi żółte. Wymaga to jednak wielokrotnego zastosowania operacji modulo (operator „%”), która ma relatywnie duże wymagania co do ilości zajmowanych jednostek logicznych. Niestety, projekt z takim rozwiązaniem wymagał 130% dostępnych jednostek logicznych. Postanowiono więc rozłożyć w czasie algorytm wyznaczania podpowiedzi tak bardzo, jak to tylko możliwe, czyli po 1 porównaniu na cykl, bez wykorzystywania operacji modulo.

Do podziału wykonywanej pracy skorzystano z dolnych 11 bitów licznika „time_counter”. Aby zapisać pozycję pinu potrzebna jest liczba 5-bitowa. Dolne 5 bitów licznika „time_counter” reprezentują pierwszą pozycję pinu, kolejne 5 bitów (bity od 9 do 5) reprezentują drugą pozycję pinu. Pozostały bit (o indeksie 10) reprezentuje to, czy aktualnie liczone są zielone podpowiedzi, czy żółte. Jeśli bity od 10 do 5 wynoszą 0, to liczone są podpowiedzi zielone dla pozycji określonej bitami od 4 do 0, o ile pozycja ta

jest mniejsza od ilości pinów w sekwencji. Jeśli bit 10 wynosi 1, to liczone są żółte podpowiedzi, porównując ze sobą piny na pozycjach określonych przez bity od 9 do 5 oraz od 4 do 0. Takie rozbicie licznika „timer_counter" pozwala na zrezygnowanie z operacji modulo. Implementacja tego algorytmu, razem z całością pozostałego projektu na moment jego stworzenia, wymagała 57% dostępnych jednostek logicznych. Ponieważ jednak obliczenia te są rozciągane w czasie, występuje opóźnienie równe 2^{10} cykli zegara 1MHz, czyli około 1ms, które musi zostać odczekane od momentu modyfikacji sekwencji. Przebieg algorytmu zobrazowano na Rys. 5.10



Rys. 5.10: Schemat logiczny rozłożenia wyznaczania podpowiedzi w czasie

Rozdział 6

Weryfikacja i walidacja

6.1 Środowisko symulacyjne QuestaSim

Środowisko Deweloperskie Quartus Prime Lite, jakie zostało wykorzystane do stworzenia projektu, posiada możliwość integracji z wieloma narzędziami EDA (Engineering Design Automation). Jednym z tych narzędzi jest środowisko symulacyjne QuestaSim. Narzędzie to jest darmowe, wymaga jedynie zarejestrowania się na stronie intel.com oraz uzyskania darmowej rocznej licencji. QuestaSim jest następcą innego popularnego narzędzia, o nazwie ModelSim. Pozwala na kompilację całego projektu i zasymulowanie jego działania, bez potrzeby wgrywania konfiguracji na fizyczne urządzenie. Dodatkowo pozwala na wypisywanie na ekranie informacji zdefiniowanych w module testowym, zapisywania i odczytywania informacji z plików, podglądu przebiegów zmian wybranych sygnałów. Aby skorzystać z tego narzędzia, należało ustawić je jako domyślne narzędzie symulacyjne w zakładce `Assigments > Settings > Simulation > Tool Name`. Następnie należało stworzyć nowy moduł testowy, nazywany testbenchem, który to nie będzie syntezowany do kodu na FPGA, a jedynie będzie służył stworzeniu instancji testowanych modułów oraz będzie definiował co jest podawane na wejście owego modułu. Dostępne są 2 tryby symulacji: RTL (Register Transfer Level) oraz Gate Level. Ta pierwsza jest szybsza, lecz mniej wierna rzeczywistemu zachowaniu układu. Ta druga symuluje każdy pojedynczy element wewnątrz FPGA, ale przez to jest wolniejsza.

Narzędzie to bardzo przydało się na wczesnych etapach projektu. Pozwalało podejrzeć przebiegi generowanych sygnałów VGA, co pozwoliło wykryć wiele błędów w kodzie związanych z niepoprawnym zachowaniem liczników. Wraz z rozwojem kodu niestety próby kompilacji plików w środowisku QuestaSim przestało działać. Po wielu próbach uruchamiania symulacji i ręcznego modyfikowania komend symulacyjnych, które to powinny być domyślnie generowane przez Quartus Prime, okazało się, że najprawdopodobniej QuestaSim inaczej traktuje globalne parametry niż Quartus Prime,

przez co z parametrów można korzystać wyłącznie w obrębie pliku, w którym są one wykorzystywane. Spowodowało to, że należało zrezygnować z korzystania z symulacji. Na szczęście symulacja, kiedy już stworzono podstawowy szkielet programu, nie była aż tak potrzebna, ponieważ większość błędów wynikała z, lub mogła być obserwowalna wyłącznie poprzez błędne wyświetlanie elementów na ekranie. Środowisko to nie posiada symulatora ekranu VGA ani HDMI, dlatego diagnostyka tych błędów z poziomu symulacji i tak byłaby dużo trudniejsza niż poprzez wgranie projektu na fizyczne urządzenie i obserwacja wygenerowanego obrazu.

6.2 Analiza czasowa

Pomocnym narzędziem zintegrowanym z Quartus Prime jest Timing Analyzer. Narzędzie to dokonuje analizy połączeń pomiędzy wszystkimi elementami wewnątrz FPGA, biorąc pod uwagę minimalne i maksymalne opóźnienie zegara, możliwą temperaturę urządzenia, czy wahania napięcia zasilającego urządzenie. Timing Analyzer na tej podstawie wyznacza czasy propagacji wszystkich sygnałów wewnętrznych, oceniając czy są one odpowiednio szybkie dla poprawnego działania urządzenia. Jeśli nie są, to zwraca informacje o błędach. W celu skorzystania z Timing Analyzer'a, należy stworzyć plik SDC (Synopsys Design Constraints), który definiuje parametry wykorzystanych sygnałów zegarowych oraz urządzeń peryferyjnych, mających pracować synchronicznie z FPGA. W przypadku tego projektu, jak widać na Rys. 6.1, zdefiniowano w pliku SDC zegar wejściowy o częstotliwości 10MHz, czyli okresie 100 ns, zarejestrowano dzielnik częstotliwości, a resztę sygnałów zegarowych wygenerowano automatycznie komendą „derrive_pll_clocks". Ponieważ FPGA nie pracuje z żadnymi urządzeniami peryferyjnymi z którymi synchronizacja jest krytyczna, wszystkie porty wejścia/wyjścia oznaczono jako „false_path".

```
create_clock -name main_clk -period 100.000 [get_ports CLK0]
derive_pll_clocks
create_generated_clock -name vga_from_tmds_clk
-source [get_pins {main_pll|altpll_component|auto_generated|pll1|clk[3]}]
-divide_by 10 [get_registers {CLK_DIV_BY_10:tmds_div|out}]
derive_clock_uncertainty
set_false_path -from [get_ports BTN_*]
set_false_path -to [get_ports VGA_*]
set_false_path -to [get_ports LED?]
set_false_path -to [get_ports HDMI_*]
```

Rys. 6.1: Zawartość wykorzystanego pliku SDC

6.3 Testy interfejsu HDMI

Podczas testów sygnał HDMI okazał się być bardzo niestabilny. Na niektórych testowanych monitorach działa bez najmniejszych problemów, na niektórych ucinane jest pierwsze kilkanaście kolumn pikseli, a na jeszcze innych występują artefakty w całym obrazie. Przypuszczalnie błędy te wynikają z niemożności zastosowania częstotliwości taktowania zgodnej ze standardem, czyli 251,75MHz.

Początkowo planowano, aby generowany obraz miał rozdzielczość 800x600 z odświeżaniem 60Hz. Zgodnie ze standardem, wymagałoby to Zegara Pikseli o częstotliwości 40MHz i zegara TMDS o częstotliwości 400MHz, które mogą zostać wygenerowane przez układ PLL. Niestety, zgodnie z notą katalogową wykorzystywany moduł FPGA 10M08DAF256C8G jest w stanie wygenerować sygnał wyjściowy z PLL o maksymalnej częstotliwości 402MHz [Int22]. Wymagana częstotliwość byłaby więc na granicy osiągalności. Niestety podczas testów wyżej wspomnianej rozdzielczości na bardzo prostym przykładzie udostępnianym przez producentów płytki MAXimator, generującym kolorowe paski na ekranie poprzez HDMI, monitor nie rozpoznawał sygnału HDMI jako poprawnego. Może to oznaczać, że wykorzystany moduł MAXimator nie obsługuje tak wysokich częstotliwości. Aspekt ten znacznie ogranicza liczbę dostępnych rozdzielczości możliwych do wygenerowania, jeśli chce się korzystać z interfejsu HDMI.

Rozdział 7

Podsumowanie i wnioski

Podsumowując, projekt można uznać za udany. Stworzono grę „Mistrz Intelektu" działającą na układzie FPGA. Gra pozwala na 3 tryby rozgrywki, zarówno dla jednego jak i dla 2 graczy. Możliwa jest także pełna konfiguracja parametrów rozgrywki. Można zmienić ilość pinów w sekwencji, ilość dostępnych kolorów pinów oraz ilość dozwolonych prób odgadnięcia sekwencji. Gra wyświetlana może być za pomocą interfejsu VGA oraz HDMI w rozdzielczości 640x480 z częstotliwością odświeżania 60Hz. Możliwa jest także modyfikacja parametrów wyświetlania obrazu, jakimi są rozmiar czcionki wyświetlanych napisów oraz palety wyświetlanych kolorów, co wpływa na kolory napisów, kolor tła oraz kolor zaznaczonych elementów. Urządzenie pozwala także na odtwarzanie dźwięków za pomocą dodanego brzęczyka. Na ekranie wyświetlane jest także kilka ozdobnych efektów graficznych, jakimi są migające gwiazdki w tle oraz fajerwerki po osiągnięciu zwycięstwa w grze. Elementy wyświetlane na ekranie są automatycznie przeskalowywane i przesuwane przy każdej zmianie dowolnego z parametrów, co zapewnia estetyczność wyglądu interfejsu gry przy dowolnych ustawieniach.

Całość gry udało się zrealizować w układzie FPGA, który to nie jest standardowym procesorem, który wykonuje sekwencyjnie polecenia, ale należało de facto stworzyć logikę całej gry za pomocą odpowiedniego ułożenia bramek logicznych. Użycie układu FPGA zamiast mikroprocesora było wymaganiem, ponieważ zwykły mikroprocesor miałby trudności z generowaniem sygnału wizyjnego, szczególnie HDMI. Każda pojedyncza operacja logiczna lub arytmetyczna wymagałaby często więcej niż jednego taktu zegara procesora. Kolejne wartości sygnałów HDMI muszą być wyznaczane z częstotliwością 252MHz, co wymagałoby procesora pracującego z częstotliwością rzędu gigaherców. Układy FPGA obchodzą ten problem, poprzez możliwość realizacji wielu operacji równolegle, gdzie procesor musiałby realizować je sekwencyjnie. Potrafią realizować dowolnie skomplikowaną funkcję logiczną w czasie jednego okresu zegara taktującego (pod warunkiem, że czas propagacji przez układ nie jest zbyt długi oraz dany schemat może się pomieścić na ograniczonej liczbie jednostek logicznych). Projektowanie macierzy FPGA wykonano w języku SystemVerilog, za pomocą środowiska deweloperskiego Quartus Prime Lite. W implementacji poszczególnych funkcji gry wykorzystano moduły

pamięci RAM oraz ROM. Stworzono system generowania kolorów pikseli na podstawie zawartości tych pamięci. Wymagało to rozwiązania problemu związanego z opóźnieniem 2 cykli sygnału zegarowego, jakie te moduły wprowadzają. Stworzono system wielu równoległych liczników inkrementowanych jednocześnie. Jeden z zestawów liczników służył wyznaczaniu adresu aktualnie odczytywanej komórki pamięci RAM lub ROM. Inny zestaw, którego stan jest opóźniony względem pierwszego o 2 cykle sygnału zegarowego, wykorzystywany był podczas przetwarzania informacji zwracanych przez moduły pamięci na odpowiedni kolor rysowanego piksela. Cały system pracował więc w 2 stanach jednocześnie, gdzie opóźnienie jednego ze stanów odpowiadało opóźnieniu wprowadzanemu przez moduły pamięci.

Czcionka rastrowa wykorzystywana do rysowania wielu elementów przechowywana jest w pamięci ROM, której to zawartość generowana jest na etapie kompilacji za pomocą zewnętrznego skryptu napisanego w języku JavaScript, co pozwala na wygodną modyfikację czcionki oraz treści wyświetlanych ciągów znaków, automatycznie wyznaczając adresy każdego znaku w pamięci oraz wiele przydatnych parametrów związanych z korzystaniem z tej pamięci.

Urządzenie jest w stanie generować obraz o rozdzielczości większej niż 640x480. Zmiana rozdzielczości jest kwestią podmiany wartości 8 stałych parametrów w jednym miejscu w kodzie źródłowym oraz zmiany generowanej częstotliwości wyjściowej układu PLL. Niestety zmiana rozdzielczości na większą sprawia, że obraz HDMI przestaje działać, co okazało się niemiłym zaskoczeniem podczas tworzenia projektu. Problemy te wynikły jednak nie z błędu w projekcie, ale z niewystarczającej maksymalnej częstotliwości działania układu FPGA na wybranym zestawie startowym MAXimator. Wygenerowanie obrazu HDMI o większej rozdzielczości wymaga zastosowania bardziej zaawansowanego układu niż początkowo zakładano.

Przykładowe pomysły na rozbudowę projektu to dodanie pamięci nieulotnej, np. wykorzystując slot kart MicroSD lub wbudowaną do FPGA pamięć Flash. W pamięci tej można by zapisywać aktualne ustawienia użytkownika, aby nie trzeba było ich zmieniać przy każdym restarcie urządzenia. Dokonano próby wykorzystania pamięci Flash, jednak niestety się nie powiodła. Z nieznanych przyczyn nie udało się zaobserwować zmian pamięci po nadpisywaniu jej nowymi wartościami.

Możliwym jest też ulepszenie niektórych algorytmów zaimplementowanych w projekcie. Przykładowo, algorytm wyznaczania wartości podpowiedzi, przedstawiony w podrozdziale 5.9, wykonuje obliczenia jedynie przez małą część czasu swojego działania. Tworząc ten algorytm, skupiono się na prostocie implementacji zamiast wydajności. Uznano jednak, że opóźnienie 1ms, jakie ten algorytm wprowadza, jest akceptowalnie małe.

Bibliografia

[EA21] E. Alkan, *A Comparative Study on Pseudo Random Number Generators in IoT devices*, 2021

[KL] Kamami Labs, *Altera MAX10 FPGA Evaluation Board User Manual*

[HD06] Hitachi Ltd., Matsushita Electric Industrial Co. Ltd., Philips Consumer Electronics, International B.V., Silicon Image Inc., Sony Corporation Thomson Inc., Toshiba Corporation, *High-Definition Multimedia Interface*
Specification Version 1.3a, 2006

[Int22] Intel, *Intel® MAX® 10 FPGA Device Datasheet*, 2022

[Int21a] Intel, *Intel® MAX™ 10 Embedded Memory User Guide*, 2021

[Int21b] Intel, *Intel® MAX® 10 High-Speed LVDS I/O User Guide*, 2021

[TI07] Taxas Instruments, *TXS0108EPWR Datasheet*, 2007

[VE13] VESA *VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)*, 2013

Dodatki

Spis skrótów i symboli

FPGA	Field-Programmable Gate Array
LUT	LookUp Table
PCB	Printed Circuit Board
LED	Light Emiting Diode
VGA	Video Graphics Array
HDMI	High Definition Multimedia Interface
USB	Universal Serial Bus
JTAG	Joint Test Action Group
EDA	Engineering Design Automation
RAM	Random Access Memory
ROM	Read Only Memory
LVDS	Low Voltage Differential Signal
PLL	Phase Locked Loop
PRNG	Pseudo-Random Number Generator
DVI	Digital Visual Interface
TMDS	Transition Minimized Differential Signaling
RTL	Register Transfer Level
SDC	Synopsys Design Constraints

Źródła

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

- skompresowany projekt Quartus Prime Lite, gdzie znajdują się pliki źródłowe projektu
- film pokazujący działanie zaprojektowanego i wykonanego urządzenia

Spis rysunków

Spis tablic