UKŁADY CYFROWE I SYSTEMY WBUDOWANE

Zadanie projektowe: Gra wykorzystująca potencjometr

**Autorzy:**

Wojciech Ormaniec, 226181

Bartosz Rodziewicz, 226105

**Prowadzący:** dr inż. Jarosław Sugier

**Grupa:** Czwartek, TN, 8:00

# Wprowadzenie

## Temat

Tematem naszego projektu było zrealizowanie gry wykorzystującej potencjometr jako input od gracza.

Zrealizowaliśmy go w formie prostej gry zręcznościowej polegającej na ruszaniu platformy gracza w lewo i w prawo, celem uniknięcia spadających obiektów. Zderzenie ze spadającym obiektem kończy grę.

## Sprzęt

Gra została napisana w języku VHDL na układ logiczny typu FPGA z rodziny Xilinx Spartan 3E FPGA Starter Kit Board, model XC3S500E.

Do gry wykorzystujemy input od gracza za pomocą potencjometru podłączonego do portu ADC płytki. Do obsługi ADC wykorzystujemy moduł *ADC\_Ctrl* przygotowany przez dr inż. Jarosława Sugiera.

Wyjście z gry wykorzystuje oczywiście obraz puszczany przez port VGA na monitor. Obsługę VGA napisaliśmy od zera. Obraz wyjściowy jest w rozdzielczości 800x600px, 72Hz.

## Teoria

### Potencjometr – ADC

Dokładny opis działania portu ADC znajduje się w User Guidzie, wypisanym w spisie literatury.

Port ADC na płytce, na której pracowaliśmy działa pod zegarem 50MHz. Posiada dwa kanały umożliwiając pracę potencjometru (jeden kanał) lub joysticka (dwa kanały). Nasz projekt wykorzystuje potencjometr, czyli jeden kanał – kanał A.

Są dwa moduły odpowiedzialne za jego pracę – „LTC 6912-1 AMP” i „LTC 1407A-1 ADC”.

Cała obsługa tego portu w naszym projekcie odbywa się za pomocą modułu czarnej skrzynki *ADC\_Ctrl*.

Aby wystartować port ADC należy skrzynce *ADC\_Ctrl* podać sygnał *AMP\_WE* i 8 bitowy sygnał *AMP\_DI* odpowiedzialny za czułość portu ADC na kanale A i B (jest on wykorzystywany przez moduł „LTC 6912-1 AMP”).

Mimo wykorzystania tylko jednego kanału nasz sygnał *AMP\_DI* to 00010001, czyli maksymalna czułość dla obu kanałów (dokładny opis innych wartości w User Guidzie).

Teoretycznie sygnał *AMP\_WE* można podać tylko raz na działanie programu jednak korzystając z rady dr Sugiera podajemy ten sygnał co klatkę obrazu, gdy przetwarzany jest piksel (0,0), ponieważ zapewnia to stabilniejszą pracę.

Aby odczytać aktualną wartość kanałów ADC należy skrzynce *ADC\_Ctrl* podać sygnał *ADC\_Start*. Skrzynka *ADC\_*Ctrl podaje wtedy sygnał *CONV*. Jest on wykorzystywany przez moduł „LTC 1407A-1 ADC” i powoduje on odczyt wartości i zwrócenie jej na sygnały *ADC\_DOA* i *ADC\_DOB* skrzynki *ADC\_Ctrl*.

Sygnał podawany na wyjściach *ADC\_DOA* i *ADC\_DOB* jest podawany w formie 14 bitowej wartości typu signed w przedziale [0x2000, 0x1FFF], czyli w dziesiętnym [-8192, 8191].

Sygnał *ADC\_Start* podajemy pod koniec klatki, w trakcie tzw. „retrace time”, w moemncie gdy rysowany byłby piksel (801, 801) (taki piksel nie istnieje w naszej rozdzielczości).

### VGA

Obraz VGA podawany jest liniami, od lewej do prawej, z góry na dół.

Co „tick” zegara na wyjściach *R*, *G* i *B* podaje się wartość jaki kolor powinien przyjąć piksel (jeden piksel na „tick” zegara).

Jednak poza podawaniem danych poszczególnych pikseli w każdej linii i pomiędzy końcem jednej klatki obrazu, a początkiem drugiej jest tzw. „retrace time”, połączony z sygnałami *HorizontalSync* i *VerticalSync*, który zapewnia odpowiednią synchronizacje obrazu na monitorze.

W trakcie „retrace-time” wartości RGB powinny być ustawione na 000.

Więcej szczegółów jest opisane przy opisie procesów odpowiedzialnych za VGA.

Dokładny opis jak to działa, na przykładzie poziomej linii i rozdzielczości 640x480px, 60Hz podany jest w User Guidzie do naszej płytki. Wartości dla naszej rozdzielczości 800x600px, 72Hz znaleźliśmy na stronie VGA Signal Timings, załączonej w spisie literatury.

## Kod

Projekt został napisany w całości w jednym module (plus moduł *ADC\_Ctrl*). Może być on nie do końca czytelny i zawierać różne tymczasowe, mało mówiące nazwy, jak i pliki, które nie są przez nas już w żaden sposób używane.

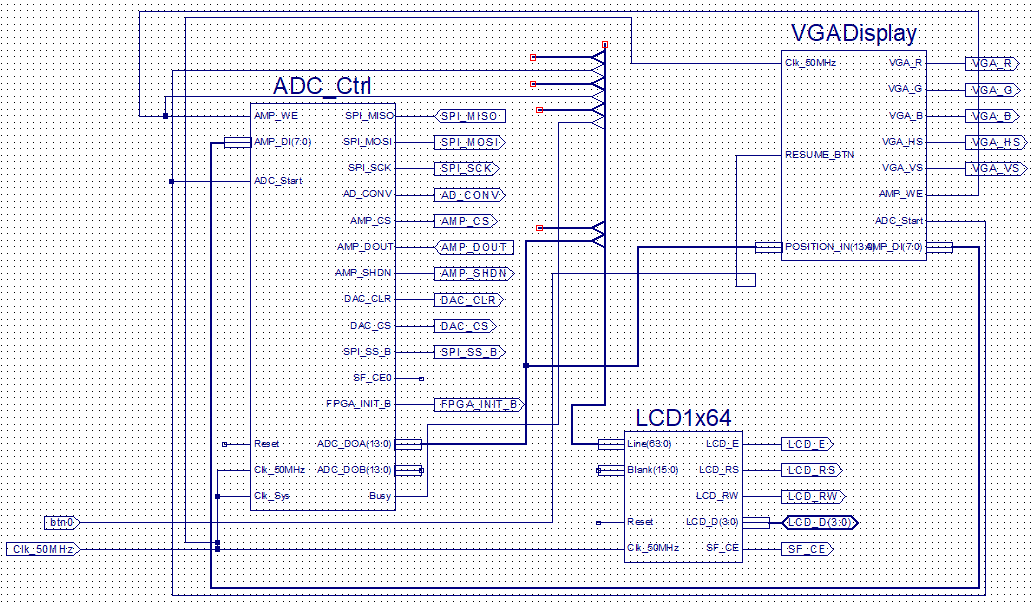
Zdarzyło się to z tego powodu, że pod koniec prac nad projektem planowane było posprzątanie kodu i pozbycie się nie potrzebnych rzeczy. Niestety nie starczyło nam czasu by to dokończyć. Na repozytorium w branchu *clean-up* znajduje się to co udało nam się zrobić w tej kwestii. Niestety z jakiegoś powodu przestał działać odczyt z potencjometru i nie byliśmy tego w stanie naprawić.

Bardzo za to przepraszamy, jeśli ktoś zdecydował się kontynuować pracę nad naszym projektem.

# Projekt

## Hierarchia źródeł

Naszym głównym źródłem w projekcie jest plik schematu (*adc\_test.sch*), który wygląda następująco:



Jak widać kod składa się z trzech modułów. Głównym modułem naszej gry, jest moduł *VGADisplay*. Jest to jedyny moduł napisany przez nas i zawiera on całą logikę gry, jak i obsługę VGA.

Moduł *ADC\_Ctrl* odpowiada za obsługę portu ADC, a *LCD1x64* za wyświetlanie liczb w formacie szesnastkowym na wbudowanym w płytkę wyświetlaczu. Ekran LCD był wykorzystywany przez nas do debugowania portu ADC i miał być usunięty w końcowym sprzątaniu kodu. Oba moduły są autorstwa dr Sugiera.

Dodatkowo poza schematem i modułami w projekcie znajdują się pliki UCF, które mapują I/O z naszego kodu na I/O płytki. Łącznie znajdują się trzy pliki UCF:

* *GenIO.ucf* – do obsługi zegara i przycisku restartu gry
* *ADC\_DAC.ucf* – do obsługi ADC
* *LCD.ucf* – do obsługi LCD (miał być usunięty)

Sygnały wejściowe i wyjściowe są w miarę łatwo zrozumiałe patrząc na schemat:

* *Clk\_50MHz* – główny zegar
* *Btn0* – sygnał z lewego przycisku, które resetuje grę
* Sygnały wyjściowe dla VGA – potrzebne do obsługi VGA (więcej o nich w opisie procesów odpowiedzialnych za obsługę i generowanie VGA)
* Sygnały dla ADC i LCD – obsługiwane przez moduły

Sygnały wewnętrzne pomiędzy modułowe to tylko sygnały służące do obsługi ADC napisane przez nas (wyjaśnione w opisie działania protokołu ADC).

## Moduły

Cała nasza gra została napisana w jednym module, więc ten podpunkt jest bardziej wyjaśnieniem działania poszczególnych procesów znajdujących się w module *VGADisplay*.

### Sygnały wejściowe i wyjściowe

|  |  |
| --- | --- |
| entity VGADisplay is  Port ( Clk\_50MHz : in STD\_LOGIC;  POSITION\_IN : in signed (13 downto 0);  RESUME\_BTN : in STD\_LOGIC;  VGA\_R : out STD\_LOGIC;  VGA\_G : out STD\_LOGIC;  VGA\_B : out STD\_LOGIC;  VGA\_HS : out STD\_LOGIC;  VGA\_VS : out STD\_LOGIC;  AMP\_WE : out STD\_LOGIC;  AMP\_DI : out STD\_LOGIC\_VECTOR (7 downto 0);  ADC\_Start : out STD\_LOGIC );  end VGADisplay; | Sygnał zegara  Nieprzetworzony sygnał z wejścia ADC, z kanału A  Sygnał z lewego przycisku, resetujący grę  Sygnał koloru czerwonego do wyjścia VGA  Sygnał koloru zielonego do wyjścia VGA  Sygnał koloru niebieskiego do wyjścia VGA  Sygnał HorizontalSync do wyjścia VGA  Sygnał VerticalSync do wyjścia VGA  Sygnał uruchamiający ADC  Sygnał czułości ADC  Sygnał wysyłający żądanie pobrania wartości ADC |

### Sygnały wewnętrzne

|  |  |
| --- | --- |
| architecture Behavioral of VGADisplay is  Signal vs\_counter : INTEGER;  Signal hs\_counter : INTEGER;  Signal playerPositionX : INTEGER := 400;  -- playerPositionY := 500;  type Point is array (1 downto 0) of INTEGER;  type BombArray is array (4 downto 0) of Point;  Signal bombsPosition : BombArray := ( (50, -2850), (200, -2650), (400, -2450), (600, -2250), (750, -2050) ); *-- -2000 to get few seconds before first bomb, 200 difrence between them to not get them falling all in the same time*  *-- bombsPosition(x)(1) -> x position; bombsPosition(x)(0) -> y position;*  Signal rand800 : unsigned (9 downto 0);  Signal colision : STD\_LOGIC := '0';  begin | Sygnał zliczający linijkę, która aktualnie jest rysowana na ekranie. Przyjmuje wartości z przedziału szerszego niż [0, 599] z uwagi na „retrace time”.  Sygnał zliczający aktualnie rysowany piksel w linijce (tak samo uwzględnia „retrace time”, a więc wartości ujemne stąd typ integer.  *vs\_counter* i *hs\_counter* to główne sygnały odpowiadające za synchronizację czasową wszystkich działań w module, zaraz po zegarze.  Sygnał przechowujący aktualną pozycję X gracza.  Pozycja Y gracza jest ustawiona na stałe w kodzie na wartość 500px  Nasz typ danych – punkt/piksel  Nasz typ – tablica 5 punktów  Sygnał będący tablicą, przechowujący pozycję 5 punktów – pięciu bomb, które aktualnie spadają na gracza. Przypisywane wartości to początkowe wartości gry. Wartości X są rozłożone w miarę równomiernie po całym ekranie, wartości Y mają duża wartość, by nie spadały przez chwilę od uruchomienia gry i dodatkowo mają pomiędzy sobą odstęp, by nie spadały w jednej linii.  Sygnał przechowujący aktualną „pseudolosową” wartość modulo 800, która jest wykorzystywana do losowej pozycji bomby po upadku  Sygnał przechowujący stan końca gry, gdy ‘1’ gra jest skończona i trzeba zresetować przyciskiem. |

### Procesy odpowiedzialne za działanie VGA

|  |
| --- |
| HorizontalSync : process ( Clk\_50MHz, hs\_counter ) is  begin  if ( rising\_edge(Clk\_50MHz) ) then  if ( hs\_counter < -64 ) then  VGA\_HS <= '0';  else  VGA\_HS <= '1';  end if;  end if;  end process; |

Proces *HorizontalSync* odpowiada za nadawanie sygnału *HorizontalSync* dla VGA. Nadawany jest on gdy *hs\_counter* jest w przedziale [-184, -65].

|  |
| --- |
| VerticalSync : process ( Clk\_50MHz, vs\_counter ) is  begin  if ( rising\_edge(Clk\_50MHz) ) then  if ( vs\_counter < -23 ) then  VGA\_VS <= '0';  else  VGA\_VS <= '1';  end if;  end if;  end process; |

Proces *VerticalSync* odpowiada za nadawanie sygnału *VerticalSync* dla VGA. Nadawany jest on gdy *vs\_counter* jest w przedziale [-29, -24].

|  |
| --- |
| PixelCounters : process ( Clk\_50MHz, hs\_counter, vs\_counter ) is  begin  if ( falling\_edge(Clk\_50MHz) ) then  if ( hs\_counter = 855 ) then  hs\_counter <= -184;  if ( vs\_counter = 636 ) then  vs\_counter <= -29;  else  vs\_counter <= vs\_counter + 1;  end if;  else  hs\_counter <= hs\_counter + 1;  end if;  end if;  end process; |

*PixelCounters* to główny proces zarządzający timingiem w naszej grze. Dba on by *hs\_ counter* i *vs\_counter* były w swoich zakresach.

Z tego co widzimy *hs\_counter* przyjmuje wartości w przedziale [-184, 855]. Okres ten dzieli się na:

* Sygnał Sync [-184, -65]
* Front Porch [-64, -1]
* Czas wyświetlania [0, 799]
* Back porch [800, 855]

Sygnał *vs\_counter* działa analogicznie, w przedziałach: [-29, -24], [-23, -1], [0, 599] i [600, 636].

### Proces odpowiedzialny za działanie ADC

### Proces odpowiedzialny za rysowanie

### Procesy logiki gry

### Procesy pomocnicze

# Implementacja

## Rozmiar

## Prędkość

## Podręcznik użytkownika

# Podsumowanie

## Uwagi krytyczne

## Możliwości dalszego rozwoju

# Spis literatury

* Repozytorium z naszym projektem - <https://github.com/baatochan/WAGDGame>
* Strona producenta - <https://www.xilinx.com/>
* Podręcznik użytkownika (Spartan-3E FPGA Starter Kit Board User Guide, User Guide, ug230.pdf) - <https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf>
* Strona z modułami dr Sugiera - <http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/>
* VGA Signal Timings - <http://tinyvga.com/vga-timing/800x600@72Hz>