Laboratorium Techniki cyfrowej

Dokumentacja projektu

Moduł generatora obrazu VGA

Szymon Miękina

Dominik Marek

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

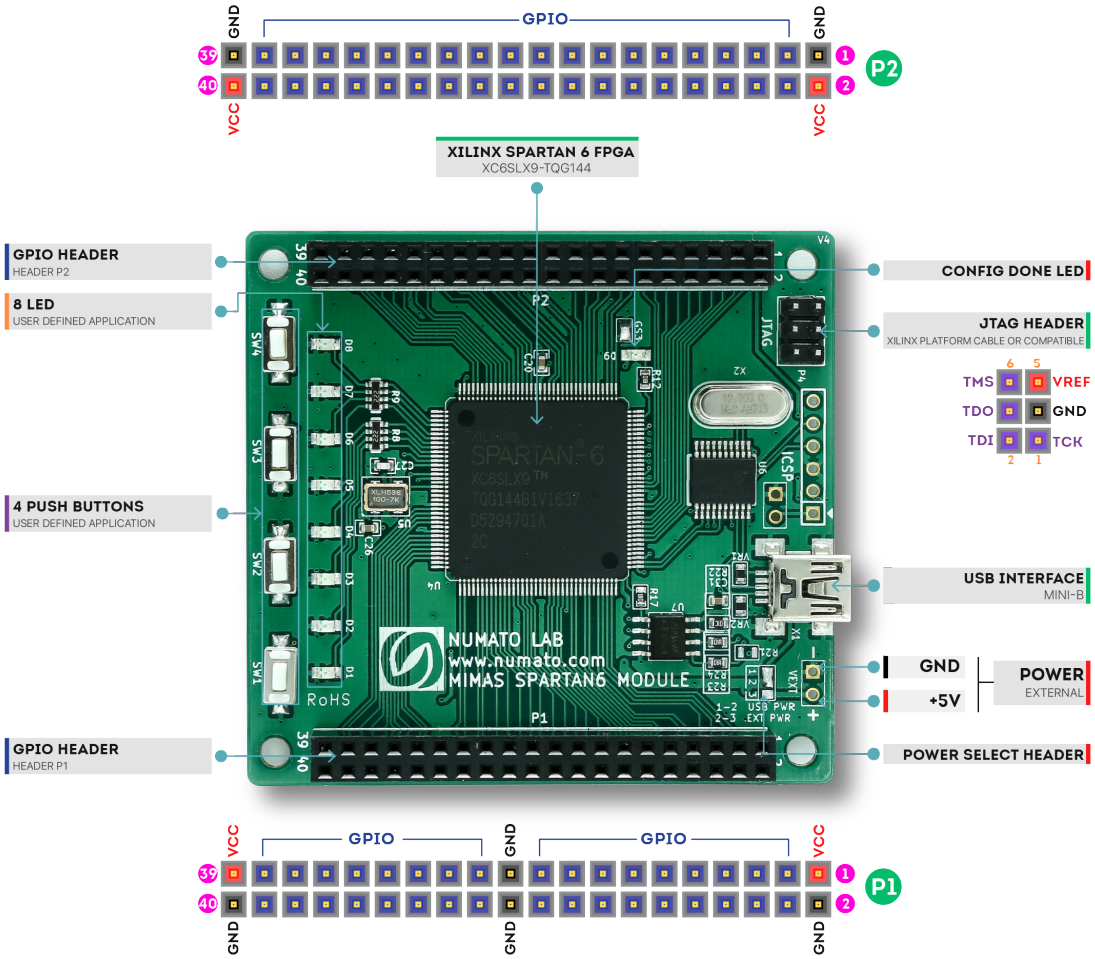
Spis treści

1 Opis projektu

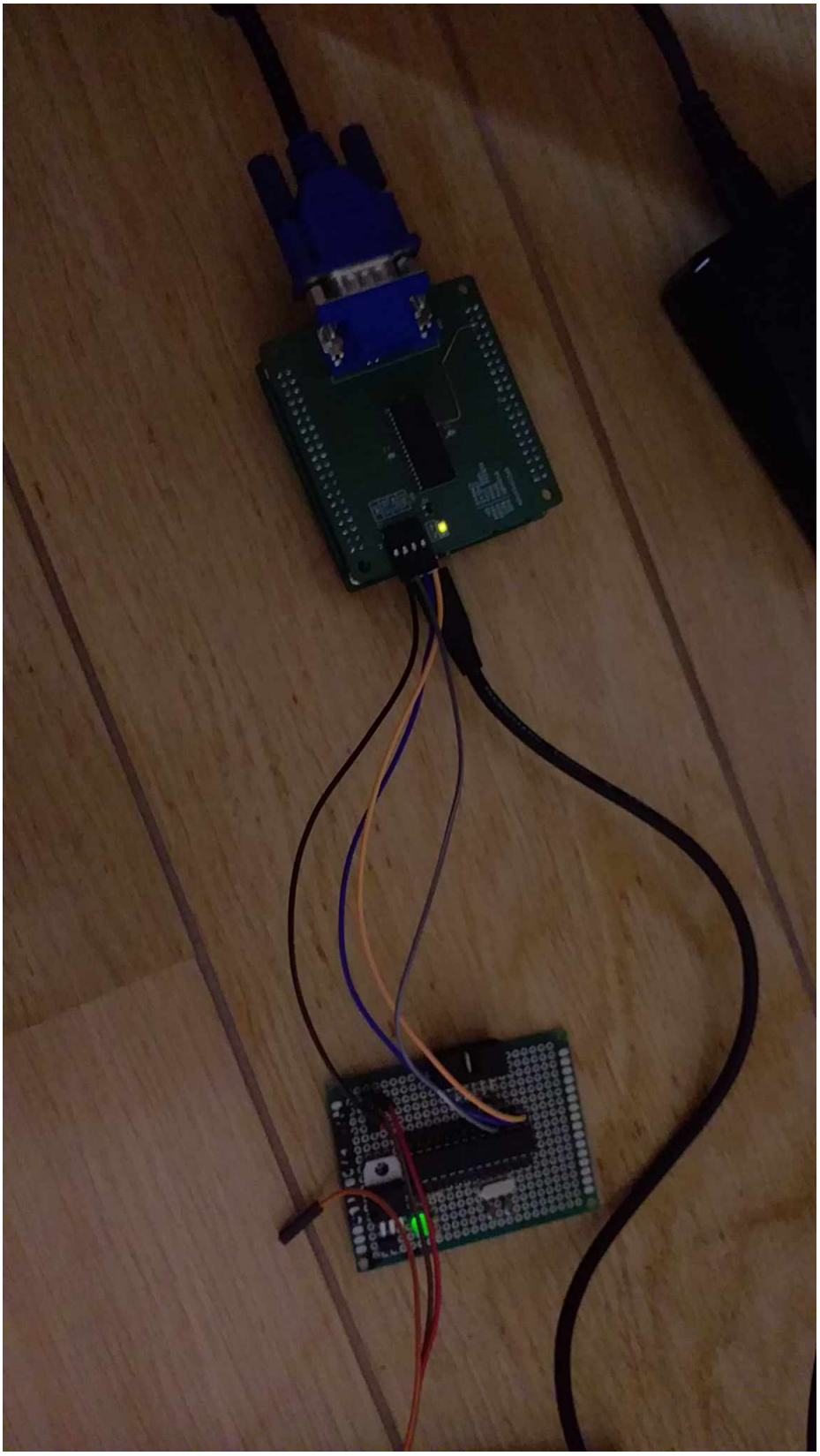
Celem projektu było napisanie programu ,wykorzystując język opisu sprzętu Verilog, który umożliwi wyświetlanie stabilnego obrazu za pomocą złącza VGA na programowalnym układzie logicznym.

2 Wykorzystany sprzęt

Do realizacji projektu wykorzystano moduł Mimas Spartan6 rozbudowany o zewnętrzną pamięć IS61LV5128AL 512K x 8 HIGH-SPEED CMOS STATIC RAM.



Rysunek 1: Płytka rozwojowa FPGA z układem Xilinx Spartan 6



Rysunek 2: Zdjęcie układu FPGA

3 Przygotwanie merytoryczne

Przystępując do pracy zapoznaliśmy się z ze składnią, działaniem oraz użytecznymi w projkcie bibliotekami jezyka Verilog.Kolejnym krokiem było zaznajomienie się z dokumentacją oraz paramterami technicznymi sygnału VGA o rozdzielczości 640 na 480 i częstotliwości odświeżania 60Hz.

3.1 Sygnał w standardzie 640x480@60Hz

W celu otrzymania sygnału o VGA o rozdzielczości 640 na 480 i odświeżaniu 60Hz, należało spełnić poniższe parametry sygnału o omawianym standardzie. Dokładna ich realizacja została omówiona w rozdziale poświęconym poszczególnym modułom projektu.

Tabela 1: General Timing Parameters

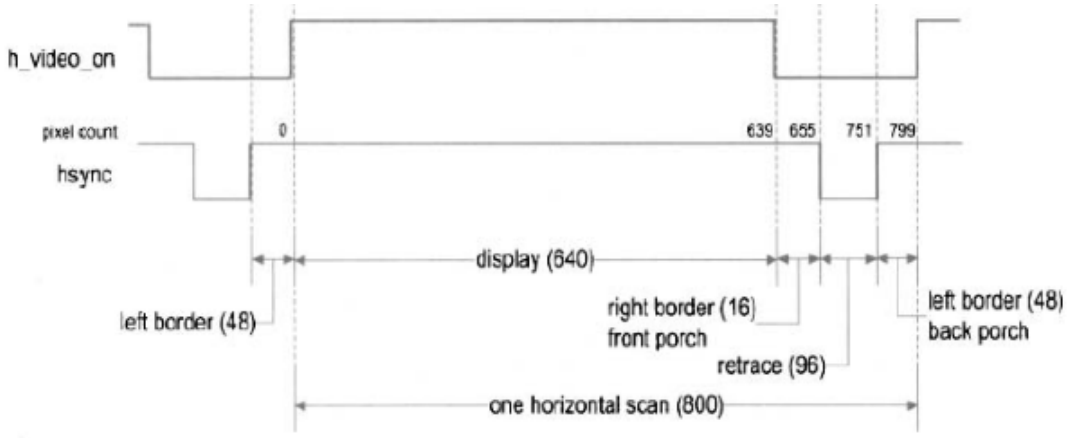
Parameter	**Value**
Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel frequency	25.175 MHz

Tabela 2: Horizontal Timing (Line)

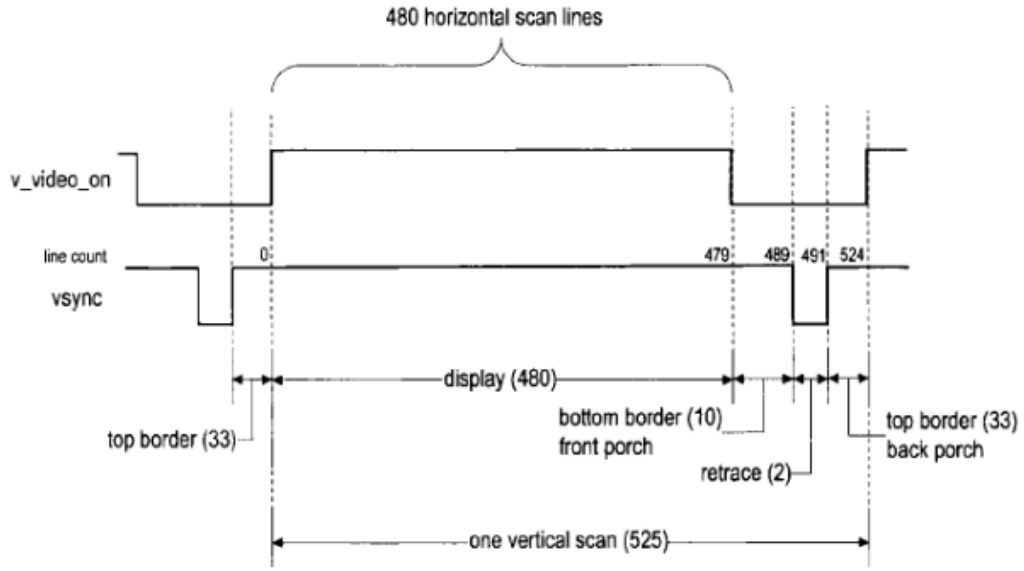
Scanline part	**Pixels**	**Time** [μs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Tabela 3: Vertical Timing (Frame)

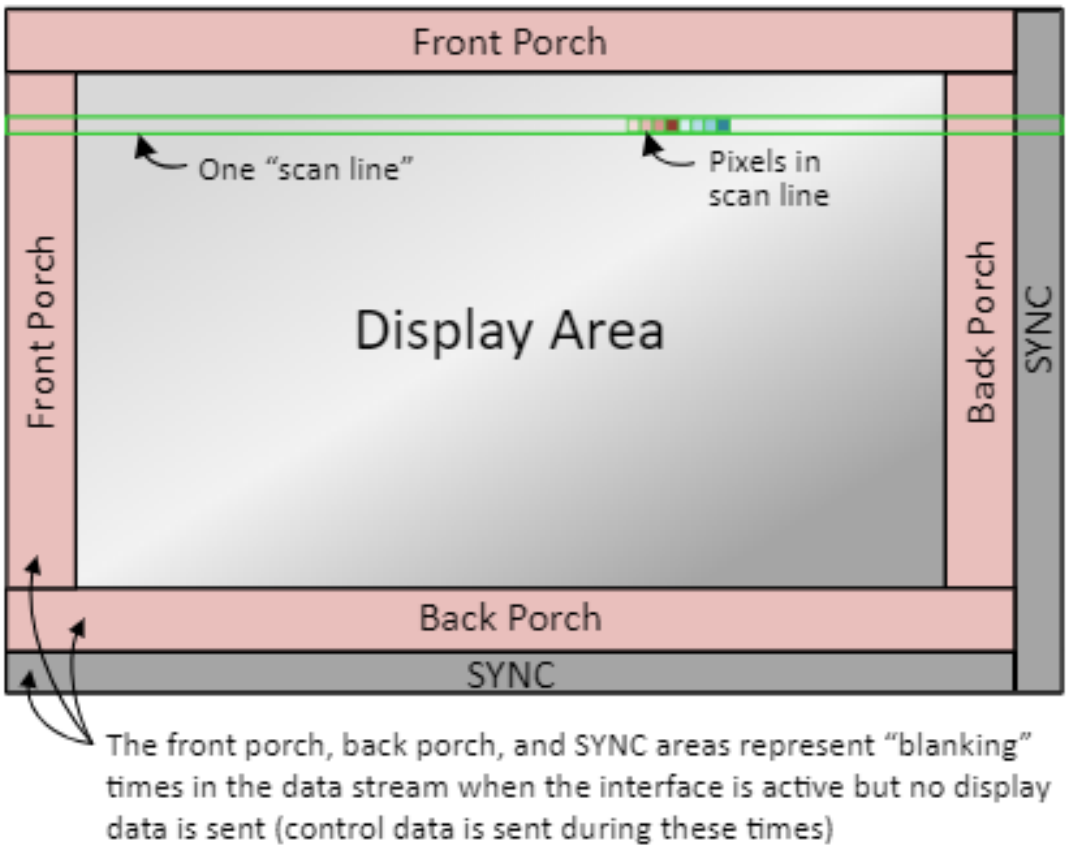
Frame part	**Lines**	**Time [ms]**
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656



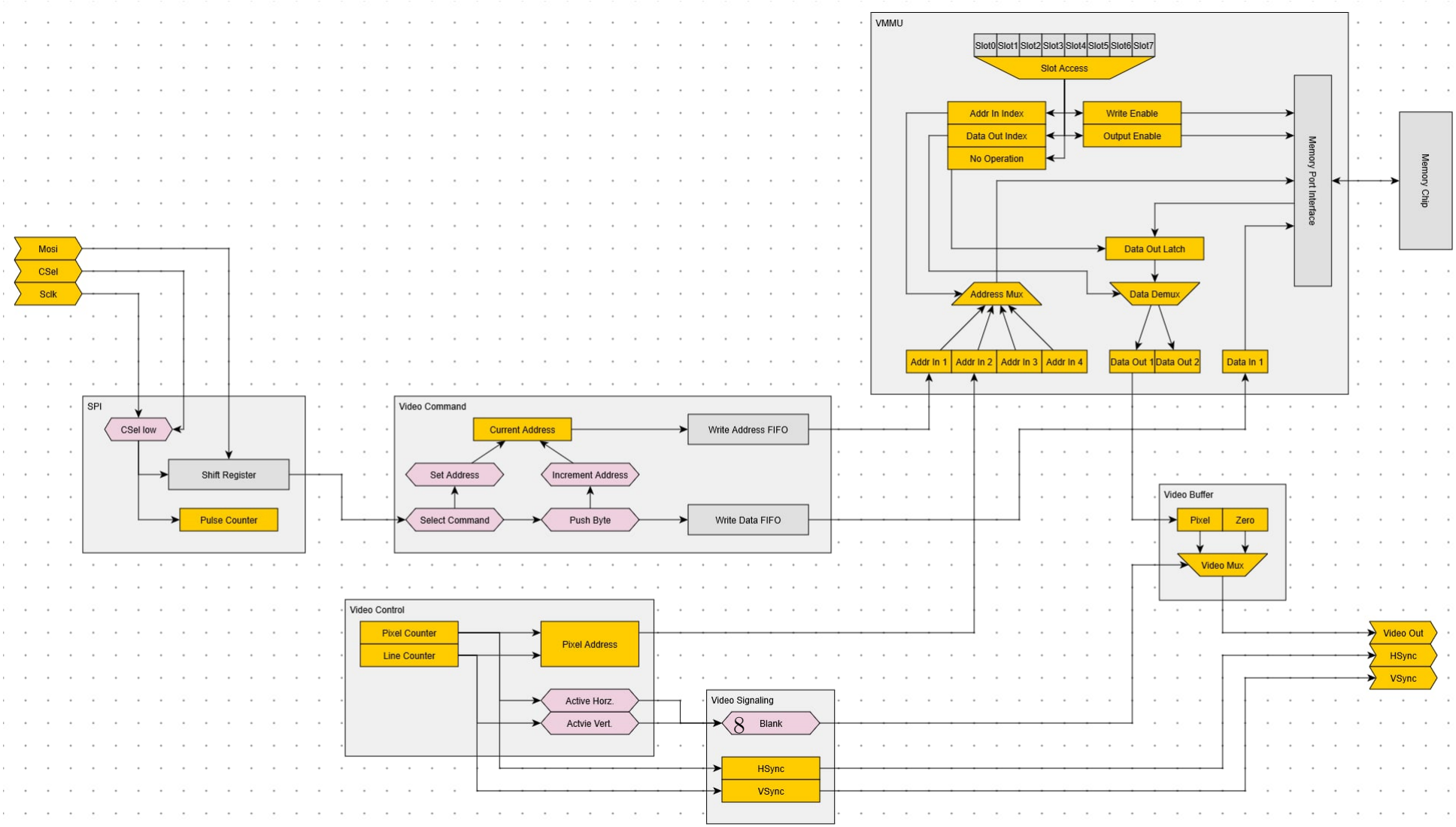
Rysunek 3: Schemat synchronizacji skanowania poziomego



Rysunek 4: Schemat synchronizacji skanowania pionowego



Rysunek 5: Sygnały sterujące wyświetlaniem obrazu.



Rysunek 6: Schemat połączeń między modułami w projekcie

4 Realizacja projektu

Po przeanalizowaniu wszystkich aspektów wymienionych wyżej, przystąpiliśmy do projektowania programu, na który składa się jedynaście poniższych modułów.

4.1 fifo.v

Pierwszy moduł naszego projektu to bufor typu FIFO (First-In-First-Out),który służy do przechowywania danych w sposób, w którym dane, które pierwsze zostały umieszczone w buforze, są również pierwsze pobierane.

Listing 1: Moduł fifo.v

```
1
2  module fifo #(
3      parameter BUFSIZE = 16, // buffer size
4      parameter IWIDTH = 4, // buffer index width
5      parameter WWIDTH = 8 // memory word width
6  ) (
7      input wire[WWIDTH-1:0] DataIn,
8      output reg[WWIDTH-1:0] DataOut,
9
10     input wire ClkIn,
11     input wire ClkOut,
12
13     output wire IsFull,
14     output wire IsEmpty
15 );
16
17 reg[WWIDTH-1:0] Buffer[BUFSIZE-1:0];
18 reg[IWIDTH-1:0] InIndex = 1'b0, OutIndex = 1'b0;
19
20 assign IsFull = InIndex + 1 == OutIndex;
21 assign IsEmpty = InIndex == OutIndex;
22
23 always @(posedge ClkIn) begin
24     if (!IsFull) begin
25         Buffer[InIndex] <= DataIn;
26         InIndex <= InIndex < BUFSIZE-1 ? InIndex + 1 : 0;
27     end
28 end
29
30 always @(posedge ClkOut) begin
31     if (!IsEmpty) begin
32          DataOut <= Buffer[OutIndex];
33         OutIndex <= OutIndex < BUFSIZE-1 ? OutIndex + 1 : 0;
34     end
35 end
36
37 endmodule
```

Wejścia i wyjścia:

1. **DataIn**: Dane wejściowe, które mają być zapisane do bufora FIFO.
2. **DataOut**: Dane wyjściowe pobierane z bufora FIFO.
3. **ClkIn**: Zegar dla operacji zapisu do bufora.
4. **ClkOut**: Zegar dla operacji odczytu z bufora.

5. **IsFull**: Sygnał wskazujący, czy bufor jest pełny.

6. **IsEmpty**: Sygnał wskazujący, czy bufor jest pusty.

Rejestry:

1. **Buffer**: Tablica rejestrów przechowujących dane w buforze FIFO.

2. **InIndex**: Licznik indeksu wskazujący na aktualną pozycję, do której będzie zapisywane dane.

3. **OutIndex**: Licznik indeksu wskazujący na aktualną pozycję, z której będą odczytywane dane.

W bloku always @(posedge ClkIn) następuje zapis danych do bufora FIFO. Jeśli bufor nie jest pełny (!IsFull), dane wejściowe (DataIn) są zapisywane do bufora na pozycji wskazywanej przez InIndex, a następnie InIndex jest inkrementowany.Gdy InIndex osiągnie maksymalny rozmiar bufora (BUFSIZE-1), wraca do początku. Wraz z każdym narastającym zboczem zegara ClkOut następuje odczyt danych z bufora FIFO. Jeśli bufor nie jest pusty (!IsEmpty), dane są odczytywane z bufora z pozycji wskazywanej przez OutIndex, a następnie OutIndex jest inkrementowany. Jeśli OutIndex osiągnie maksymalny rozmiar bufora (BUFSIZE-1), wraca do początku.

4.2 filter.v

Poniższy moduł to filtr cyfrowy, który wykonuje proste przetwarzanie sygnału wejściowego SignalIn za pomocą sekwencji rejestrów opóźniających.

Listing 2: Moduł filter.v

```
1  `timescale 1ns / 1ps
2
3  module filter #(
4      parameter STAGES = 8
5  ) (
6      input wire ClkIn,
7      input wire SignalIn,
8      output reg SignalOut
9  );
10
11 reg[STAGES-1:0] Stages;
12
13 wire AllHigh = &Stages;
14 wire AllLow = ~|Stages;
15
16 always @(posedge ClkIn) begin
17     Stages <= { Stages[STAGES-2:0], SignalIn };
18     if (AllHigh) begin
19         SignalOut <= 1'b1;
20     end
21     if (AllLow) begin
22         SignalOut <= 1'b0;
23     end
24 end
25
26 endmodule
```

Wejścia i wyjścia;

1. **ClkIn**: Zegar kontrolujący operacje w module.
2. **SignalIn**: Sygnał wejściowy, który ma być przetwarzany przez filtr.
3. **SignalOut**: Sygnał wyjściowy, który jest wynikiem przetwarzania przez filtr.

Rejstry i połaczenia:

1. **reg[STAGES-1:0] Stages**: Rejestry opóźniające, przez które przechodzi sygnał wejściowy.
2. **`wire AllHigh = & Stages`**: Połączenie sygnalizujące, czy wszystkie bity w Stages mają wartość 1.
3. **`wire AllLow = ~| Stages`**: Połączenie sygnalizujące, czy wszystkie bity w Stages mają wartość 0.

W każdym cyklu zegara (ClkIn), wartość rejestru Stages jest aktualizowana o nową wartość sygnału wejściowego SignalIn, a następnie sprawdzane są warunki, czy wszystkie bity w rejestrze Stages są ustawione na wysoki (AllHigh) lub niski (AllLow) stan logiczny, aby odpowiednio ustawić sygnał wyjściowy SignalOut.

4.3 spi.v

W tym module zaimplenetowany został prosty odbiornik SPI(Serial Peripheral Interface) czyli szeregowy interfejs urządzeń peryferyjnych.W naszym przypadku będziemy go używać do komunikacji na lini mikrokontroler poszczególne komponenty układu.

Listing 3: Moduł spi.v

```
module spi (
    input wire Clk,
    input wire Sclk,
    input wire Mosi,
    input wire CSel,
    output reg DataRecv,
    output reg[7:0] DataOut
);

reg[2:0] SclkSample;
always @(posedge Clk) SclkSample <= { SclkSample[1:0], Sclk };

wire SclkRise = SclkSample[2:1] == 2'b01;
wire SclkFall = SclkSample[2:1] == 2'b10;

reg[2:0] CSelSample;
always @(posedge Clk) CSelSample <= { CSelSample[1:0], CSel };

wire CSelActive = ~CSelSample[1];
wire CSelRise = CSelSample[2:1] == 2'b01;
wire CSelFall = CSelSample[2:1] == 2'b10;

reg[1:0] MosiSample;
always @(posedge Clk) MosiSample <= {MosiSample[0], Mosi};
```

```
25
26  wire MosiData = MosiSample[1];
27
28  reg[2:0] BitCnt = 3'b000;
29
30  reg[7:0] ByteDataRecv = 8'b00000000;
31
32  always @(posedge Clk) begin
33      if (CSelFall) begin
34           BitCnt <= 3'b000;
35       end else if (SclkRise) begin
36           BitCnt <= BitCnt + 3'b001;
37            ByteDataRecv <= {ByteDataRecv[6:0], MosiData};
38       end
39  end
40
41  always @(posedge Clk) begin
42       if (~CSelActive && (BitCnt == 3'b000)) begin
43           DataRecv <= 1'b1;
44          DataOut <= ByteDataRecv;
45       end else begin
46           DataRecv <= 1'b0;
47       end
48  end
49
50  endmodule
```

Wejścia i wyjścia:

1. **input wire Clk**: Wejście zegara(Clock signal).
2. **input wire Sclk**: Wejście zegara interfejsu SPI..
3. **input wire Mosi**: Wejście danych z mikrokontrolera(Master Out Slave In)
4. **input wire CSel**: Wejście selekcji układu (Chip Select).
5. **output reg DataRecv**: Wyjście informujące,czy odbierana jest odpowiednia ilość danych.
6. **output reg [7:0] DataOut**: Wyjście danych z układu (Master Out Slave In).

Rejestry wewnętrzne:

1. **reg [2:0] SclkSample**: Rejestr do próbkowania sygnału zegara interfejsu SPI.
2. **reg [2:0] CSelSample**: Rejestr do próbkowania sygnału wyboru interfejsu SPI.
3. **reg [2:0] MosiSample**: Rejestr do próbkowania sygnału danych wyjściowych interfejsu SPI.
4. **reg [2:0] BitCnt**: Licznik bitów używany do śledzenia, który bit danych jest przesyłany
5. **reg [7:0] ByteDataRecv**: Rejestr przechowujący dane przychodzące z linii Mosi.

Moduł wykorzystuje próbkowanie zegara Sclk i sygnału wyboru układu CSel oraz sygnału danych wyjściowych Mosi, aby śledzić transmisję danych w interfejsie SPI.Licznik BitCnt jest inkrementowany przy każdym zboczu narastającym zegara systemowego Clk, gdy CSel jest aktywne i Sclk jest w stanie wysokim. Otrzymane bity są zapisywane w ByteDataRecv. Następnie jeśli sygnału selekcji układu jest nieaktywny oraz otrzymano już cały bajt danych, sygnał DataRecv jest ustawiany na 1, co sygnalizuje gotowość danych. Dane są przekazywane przez DataOut.W przeciwnym razie sygnał DataRecv jest ustawiany na 0.

4.4 spitest.v

Moduł spitest jest testbenchem dla dwóch modułów: spi i vcmd, pozwalającym na symulację interakcji między nimi.

Listing 4: Moduł spitest.v

```
1  `timescale 1ns / 1ps
2
3  module spitest;
4
5      // Inputs
6      reg Clk = 1'b0;
7      reg Sclk = 1'b0;
8      reg Mosi = 1'b0;
9      reg CSel = 1'b1;
10
11     // Outputs
12     wire DataRecv;
13     wire [7:0] DataOut;
14
15     initial forever
16         #2 Clk <= ~Clk;
17
18     // Instantiate the Unit Under Test (UUT)
19     spi UUT (
20        .Clk(Clk),
21         .Sclk(Sclk),
22         .Mosi(Mosi),
23         .CSel(CSel),
24         .DataRecv(DataRecv),
25         .DataOut(DataOut)
26     );
27
28     task send_byte(input time PulseTime, input [7:0] Byte);
29         begin
30             CSel = 1'b0;
31             // for (j = 0; j < 8; j = j + 1)
32             #PulseTime;
33             send_pulse(PulseTime, (Byte>>7) & 1);
34             send_pulse(PulseTime, (Byte>>6) & 1);
35             send_pulse(PulseTime, (Byte>>5) & 1);
36             send_pulse(PulseTime, (Byte>>4) & 1);
37             send_pulse(PulseTime, (Byte>>3) & 1);
38             send_pulse(PulseTime, (Byte>>2) & 1);
39             send_pulse(PulseTime, (Byte>>1) & 1);
40             send_pulse(PulseTime, (Byte>>0) & 1);
41             Sclk = 1'b0;
42             #PulseTime;
43             CSel = 1'b1;
44             #PulseTime;
```

```
45          end
46      endtask
47
48      task send_pulse(input time PulseTime, input Data);
49          begin
50              Mosi = Data;
51              Sclk = 1'b0;
52              #PulseTime;
53              Sclk = 1'b1;
54              #PulseTime;
55          end
56      endtask
57
58      initial begin
59          // Wait 100 ns for global reset to finish
60          #100;
61
62          // Add stimulus here
63          send_byte(40, 8'b01000001);
64          send_byte(40, 8'b11000000);
65          send_byte(40, 8'b11000000);
66          send_byte(40, 8'b11000000);
67
68          #500;
69
70          send_byte(40, 8'b01000001);
71          send_byte(40, 8'b11000000);
72          send_byte(40, 8'b11000000);
73          send_byte(40, 8'b11000000);
74      end
75
76  endmodule
```

Na początku definiowane są wejścia (Sclk, Mosi, CSel) i wyjścia (DataRecv, DataOut) dla modułu spi, oraz wejścia (CmdDataIndex) i wyjścia (CmdMemAddr, CmdDataOut, CmdDataRdy) dla modułu vcmd. Następnie zdefiniowane są dwa zadania testowe.

Zadanie `send_byte` przyjmuje dwie wartości wejściowe: PulseTime, który określa czas trwania impulsu, oraz Byte, który jest bajtem do wysłania przez interfejs SPI. Najpierw linia CSel jest ustawiana na stan niski, co sygnalizuje rozpoczęcie transmisji. Następnie każdy bit bajtu Byte jest przesyłany pojedynczo poprzez wywołanie zadania `send_pulse` dla każdego bitu. Po wysłaniu wszystkich bitów, linia CSel ponownie jest ustawiana na stan wysoki, sygnalizując zakończenie transmisji. Zadanie `send_pulse` ustawia linię Mosi na wartość Data, ustawia zegar Sclk na stan wysoki, czeka przez określony czas PulseTime, ustawia zegar Sclk na stan niski i ponownie czeka przez PulseTime, symulując pojedynczy impuls zegara oraz przesyłanie jednego bitu danych przez linię MOSI.

W sekcji initial ustawiane są początkowe wartości sygnałów oraz oczekuje się 100 ns na zakończenie resetowania. Po zakończeniu resetowania generowane są impulsy SPI, symulujące wysyłanie danych do urządzenia. Każda wysyłka danych poprzedzona jest krótką przerwą (#500), co symuluje niewielkie opóźnienie między kolejnymi operacjami.

4.5 vbuffer.v

Moduł vbuffer odczytuje piksele za pomocą modułu zarządzania pamięcią i zapisuje je do bufora wyjściowego. Piksele są przechowywane w formacie RGB222 (6 bitów na piksel), co oznacza, że każde 4 piksele mogą być przechowywane w 3 bajtach pamięci

Listing 5: Moduł vbuffer.v

```
`timescale 1ns/1ps

module vbuffer #(
    parameter BPP = 6 // bits per pixel
) (
    input wire PixelClk,
    input wire ReqWrite,
    input wire Blank,
    input wire[7:0] DataIn,
    output reg[BPP-1:0] VideoOut
);

reg[BPP-1:0] Pixel = 6'b00_00_00;

always @(posedge PixelClk) begin
    if (Blank) VideoOut <= 1'b0;
    else VideoOut <= Pixel;
end

always @(negedge ReqWrite) begin
    Pixel = DataIn[BPP-1:0];
end

endmodule
```

Wejścia i wyjścia:

1. **input wire PixelClk**: Wejście zegara systemowego (Clock signal).
2. **input wire ReqWrite**: Sygnał żądania zapisu danych do bufora.
3. **input wire Blank**: Sygnał wygaszania.
4. **input wire [BPP-1:0] DataIn**: Dane wejściowe do zapisu
5. **output reg [BPP-1:0] VideoOut**: Dane wyjściowe.

Z każdym pozytywnym zboczem zegara(PixelClk) jeśli sygnał wygaszania(Blank) jest aktywny to sygnał VideoOut zostaje ustawiony na zero, co oznacza brak sygnału wideo na wyjściu.W przeciwnym razie VideoOut przyjmuje wartość bieżącego piksela przechowywanego w rejestrze Pixel.Następnie wraz z opadającym zboczem sygnału żądania zapisu danych do bufora(RegWrite) najmłodsze bity z DataIn (o liczbie bitów równej BPP) są zapisywane do rejestru Pixel.

4.6 vcmd.v

Moduł Video Command Processor (VCDM) będzie maszyną stanów odpowiednio przetwarzającą otrzymane sygnały z interfesju komunikacji (spi).

Listing 6: Moduł vcmd.v

```
1  `include "fifo.v"
2
3  module vcmd (
4    input wire ByteRecvClk,
5    input wire[7:0] ByteIn,
6    input wire ReadOutClk,
7
8    output wire HasReadData,
9    output wire[18:0] AddrOut,
10   output wire[7:0] DataOut
11 );
12
13 // TODO finish vcmd data
14
15 localparam Noop = 8'h00;
16 localparam SetXY = 8'h11;
17 localparam Write = 8'h20;
18
19 localparam ReadCmdId = 4'h0;
20 localparam ReadX = 4'h4;
21 localparam ReadY = 4'h5;
22 localparam ReadPixel = 4'h8;
23
24 reg[7:0] PosX = 1'b0;
25 reg[7:0] PosY = 1'b0;
26
27 reg[18:0] NextAddr = 1'b0;
28
29 reg[3:0] State = 1'b0;
30
31 wire BufferOverflow;
32 wire BufferUnderflow;
33 reg NextByte = 1'b0;
34 reg PushByte = 1'b0;
35 assign HasReadData = ~BufferUnderflow;
36
37 wire _BufferOverflow;
38 wire _BufferUnderflow;
39
40 reg[18:0] AddrIn;
41
42 fifo #(
43   .BUFSIZE(4),
44   .IWIDTH(2),
45   .WWIDTH(8)
46 ) DataFIFO (
47   .DataIn(ByteIn),
48   .ClkIn(PushByte),
49   .DataOut(DataOut),
50   .ClkOut(ReadOutClk),
51   .IsEmpty(BufferUnderflow),
52   .IsFull(BufferOverflow)
53 );
54
55 fifo #(
56   .BUFSIZE(4),
57   .IWIDTH(2),
58   .WWIDTH(19)
59 ) AddrFIFO (
60   .DataIn(AddrIn),
61   .ClkIn(PushByte),
62   .DataOut(AddrOut),
63   .ClkOut(ReadOutClk),
64   .IsEmpty(_BufferUnderflow),
65   .IsFull(_BufferOverflow)
```

```
66  );
67
68  always @(negedge ByteRecvClk) begin
69      PushByte <= 1'b1;
70  end
71
72
73  always @(posedge ByteRecvClk) begin
74       AddrIn <= NextAddr;
75
76      case (State)
77        ReadCmdId: begin
78            case (ByteIn)
79              Noop: State <= ReadCmdId;
80               SetXY: State <= ReadX;
81               Write: State <= ReadPixel;
82            endcase
83          end
84        ReadX: begin
85          PosX <= ByteIn;
86          State <= ReadY;
87        end
88        ReadY: begin
89          PosY <= ByteIn;
90            NextAddr <= { 2'b00, ByteIn[7:0], 1'b0, PosX[7:0] };
91          State <= ReadCmdId;
92        end
93          ReadPixel: begin
94            PushByte <= !BufferOverflow;
95            State <= ReadCmdId;
96            NextAddr <= NextAddr + 1'b1;
97          end
98        default: State <= ReadCmdId;
99      endcase
100 end
101
102 endmodule
```

Wejścia i wyjścia:

1. **input wire ByteRecvClk**: Zegar używany do synchronizacji odbieranych bajtów.
2. **input wire[7:0] ByteIn**: Sygnał reprezentujący konkretne polecenie.
3. **input wire ReadOutClk**: Zegar używany do synchronizacji danych odczytanych z FIFO.
4. **output wire [7:0] HasReadData**: Sygnał informujący, czy dostępne są dane do odczytu z FIFO.
5. **output reg [18:0] AddrOut**: Adres pamięci wyjściowej.
6. **output wire [7:0] DataOut**: Dane wyjściowe

Rejestry i połączenia:

1. **reg [18:0] PoxX**: Rejestr przechowujący współrzędną X
2. **reg [18:0] PoxX**: Rejestr przechowujący współrzędną Y

3. **reg [18:0] NextAddr**: 18-bitowy rejestr przechowujący następny adres pamięci, który za zostać zapisany W FIFO.

4. **reg[3:0] State**: 4-bitowy rejestr przechowujący aktualny stan maszyny stanowej.

5. **BufferOverflow**: Sygnał informujący o przepełnieniu bufora kolejki FIFO.

6. **BufferUnderflow**: Sygnał informujący o tym czy bufor kolejki FIFO jest pusty.

7. **NextByte**:Rejestr kontrolujący, czy należy przesłać następny bajt

8. **PushByte**: Sygnał sterujący zapisem danych do kolejki FIFO.

9. `assign HasReadData = ~BufferUnderflow`: Przpisanie do sygnału HasReadData negacji wartości sygnału BufferUnderflow, informujące tym samym czy w kolejce znajdują się dane do odczytu.

Konfigurujemy dwa bufory FIFO jeden do przechowywania danych wejściowych a drugi dla adresów pamięci.Następnie przy ujemnym zboczu zegara (ByteRecvClk) ustawiamy sygnał PushByte na 1, co oznacza, że dane będą zapisywane do buforów FIFO.Z każdym narastającym zboczem zegara ByteRecvClk wykonujemy następujące czynności w zależności od obecnego stanu reprezentowanego przez zmienną State:

- **ReadCmdId**: Stan, w którym oczekujemy na otrzymanie nowego polecenia (ByteIn). Po otrzymaniu polecenia, aktualizowany jest obecny stan (State) w zależności od wartości ByteIn:
  - Noop, pozostajemy w stanie ReadCmdId.
  - SetXY, stan jest zmieniany na ReadX.
  - równe Write, stan jest zmieniany na ReadPixel.
- **ReadY**: W tym stanie oczekujemy na otrzymanie wartości współrzędnej Y. Po otrzymaniu tej wartości (‘ByteIn‘), jest ona przypisywana do rejestru ‘PosY‘. Następnie ‘NextAddr‘ jest aktualizowany do wartości złożonej z bitów ‘PosY‘ i ‘PosX‘ w odpowiedniej kombinacji bitów: ‘ 2’b00, ByteIn[7:0], 1’b0, PosX[7:0] ‘. Po wykonaniu tych operacji, stan (‘State‘) jest ustawiany na ‘ReadCmdId‘, aby przygotować moduł do odczytu kolejnej komendy.
- **ReadPixel**: W tym stanie oczekujemy na zapisanie danych piksela. ‘PushByte‘ jest ustawiany na negację ‘BufferOverflow‘, co oznacza, że dane będą zapisywane do bufora FIFO, jeśli nie jest on pełny. Następnie ‘NextAddr‘ jest inkrementowany o 1, aby przygotować się do zapisu kolejnego piksela. Stan (‘State‘) jest ustawiany ponownie na ‘ReadCmdId‘, aby przygotować moduł do odczytu kolejnej komendy.
- **Default**: Jeśli aktualny stan nie pasuje do żadnego z powyższych przypadków, stan (‘State‘) jest ustawiany na ‘ReadCmdId‘, aby wrócić do oczekiwania na kolejną komendę.

W poniższej tabeli zostały przedstawione poszczególne komendy oraz działania wziązane z nimi.

Tabela 4: Odniesienia poleceń

Instrukcja	**Format**	**Opis**
NOOP	0x00	Brak czynności
SETXY	0x11	Ustawienie X (pierwszy bajt po komendzie) i Y (drugi bajt po komendzie)
WRITE	0x20	Wpisanie bajtu jako wartosci piksela (00rrggbb) i przesuniecie kursora w prawo

4.7 vcmdtest.v

Moduł VideoCommandProcessorTest to testbench dla dwóch modułów: spi i vcmd. Pozwala on przeprowadzenie symulacji interakcji między wymienionymi modułami.

Listing 7: Moduł vcmdtest.v

```
1  `timescale 1ns / 1ps
2
3  module vcmdtest;
4
5      // SPI
6
7      // Inputs
8      reg Sclk;
9      reg Mosi;
10     reg CSel;
11
12     // Outputs
13     wire DataRecv;
14     wire [7:0] DataOut;
15
16     // Instantiate the Unit Under Test (UUT)
17     spi Spi (
18         .Sclk(Sclk),
19         .Mosi(Mosi),
20         .CSel(CSel),
21         .DataRecv(DataRecv),
22         .DataOut(DataOut)
23     );
24
25     task send_byte(input time PulseTime, input [7:0] Byte);
26         begin
27             CSel = 1'b0;
28             send_pulse(PulseTime, (Byte>>7) & 1);
29             send_pulse(PulseTime, (Byte>>6) & 1);
30             send_pulse(PulseTime, (Byte>>5) & 1);
31             send_pulse(PulseTime, (Byte>>4) & 1);
32             send_pulse(PulseTime, (Byte>>3) & 1);
33             send_pulse(PulseTime, (Byte>>2) & 1);
```

```
34              send_pulse(PulseTime, (Byte>>1) & 1);
35              send_pulse(PulseTime, (Byte>>0) & 1);
36              CSel = 1'b1;
37          end
38      endtask
39
40      task send_pulse(input time PulseTime, input Data);
41          begin
42              Mosi = Data;
43              Sclk = 1'b1;
44              #PulseTime;
45              Sclk = 1'b0;
46              #PulseTime;
47          end
48      endtask
49
50      // VCMD
51
52      // Inputs
53      reg [1:0] DataIndex;
54
55      // Outputs
56      wire [17:0] CmdMemAddr;
57      wire [7:0] CmdDataOut;
58      wire CmdDataRdy;
59
60      // Instantiate the Unit Under Test (UUT)
61      vcmd UUT (
62          .CmdRecv(DataRecv),
63          .CmdIn(DataOut),
64          .MemAddr(CmdMemAddr),
65          .DataOut(CmdDataOut),
66          .DataIndex(CmdDataIndex),
67          .DataRdy(CmdDataRdy)
68      );
69
70      initial begin
71          // Initialize Inputs
72          Sclk = 0;
73          Mosi = 0;
74          CSel = 0;
75          DataIndex = 0;
76
77          // Wait 100 ns for global reset to finish
78          #100;
79
80          // Add stimulus here
81          // Very fast SPI clock (50MHz)
82          send_byte(10, 8'b01000001);
83          send_byte(10, 8'b11000000);
84          send_byte(10, 8'b11000000);
85          send_byte(10, 8'b11000000);
86
87          // Small delay between next pixel write
88          #160;
89
90          // Another byte at next location
91          send_byte(10, 8'b01000001);
92          send_byte(10, 8'b00000011);
93          send_byte(10, 8'b00000011);
94          send_byte(10, 8'b00000011);
95      end
96
97  endmodule
```

Na początku definiowane są wejścia (`Sclk`, `Mosi`, `CSel`) i wyjścia (`DataRecv`, `DataOut`)

dla modułu spi, oraz wejścia (CmdDataIndex) i wyjścia (CmdMemAddr, CmdDataOut, CmdDataRdy) dla modułu vcmd. Następnie definiowane są dwa zadania testowe. Send_byte przyjmuje dwie wartości wejściowe: PulseTime, który określa czas trwania impulsu, oraz Byte, który jest bajtem do wysłania przez interfejs SPI.W zadaniu send_byte najpierw ustawiana jest linia CSel na stan niski (CSel = 1'b0), co sygnalizuje urządzeniu docelowemu, że rozpoczyna się transmisja. Następnie każdy bit bajtu Byte jest przesyłany pojedynczo poprzez wywołanie zadania send_pulse. Dla każdego bitu wywoływane jest send_pulse z odpowiednim bitem przesuniętym do prawej ((Byte>>i) & 1), gdzie i to numer bitu. Po wysłaniu wszystkich bitów, linia CSel ponownie ustawiana jest na stan wysoki (CSel = 1'b1), sygnalizując zakończenie transmisji.Zadanie send_pulse ustawia linie Mosi na wartość Data, ustawia zegar Sclk na stan wysoki, czeka przez określony czas PulseTime, ustawia zegar Sclk na stan niski i ponownie czeka przez PulseTime. To symuluje pojedynczy impuls zegara oraz przesyłanie jednego bitu danych przez linię MOSI.
W sekcji initial ustawiane są początkowe wartości sygnałów oraz oczekuje się 100 ns na zakończenie resetowania.Po zakończeniu resetowania generowane są impulsy SPI, symulujące wysyłanie danych do urządzenia.Każda wysyłka danych poprzedzona jest krótką przerwą (#160), co symuluje niewielkie opóźnienie między kolejnymi operacjami.

4.8 vounter.v

Za pomocą modułu VideoCounter(vcounter) otrzymujemy licznki o zadanych w paramterach właściowściach.Dzięki czemu jesteśmy w stanie uzyskać liczniki umożliwiające wyzwalanie sygnałów o interesujących na częstotliwościach.

Listing 8: Moduł vcounter.v

```
`timescale 1ns/1ps

module vcounter #(
    parameter XBITS = 8, // number of bit for bit counter
    parameter YBITS = 8, // number of bit for bit line counter.
    parameter XMAX = 255, // max range for bit counter
    parameter YMAX = 255 // max range for line counter
) (
    input wire Clk,
    output reg[XBITS-1:0] PixelCounter,
    output reg[YBITS-1:0] LineCounter
);

initial PixelCounter <= 1'b0;
initial LineCounter <= 1'b0;

always @(posedge Clk) begin
    if (PixelCounter < XMAX) PixelCounter <= PixelCounter + 1'b1;
    else begin
        PixelCounter <= 1'b0;
        if (LineCounter < YMAX) LineCounter <= LineCounter + 1'b1;
        else LineCounter <= 1'b0;
    end
end

endmodule
```

Wejścia i wyjścia:

1. **input wire Clk**: Wejście głównego zegara
2. **output reg[XBITS-1:0] PixelCounter**: Wyjście rejestrowane, które przechowuje wartość licznika pikseli.
3. **output reg[XBITS-1:0] LineCounter**: Wyjście rejestrowane, które przechowuje wartość licznika lini.

Początkowo incjalizuje wartości obu liczników na zero.Następnie w każdym cyklu zegara sprawdzam czy wartość licznika pikseli nie osiągnała przewidzianego dla niej górnego zakresu(XMAX), jeśli ta wartość nie został osiągnięta to inkrementuje liczniki piskeli. W przeciwnym przypadku zrestuje licznik pikseli i jeśli wartości licznika lini jest mniejsza niz YMAX, to zwiekszam go jeden, gdy licznik osiągnie swoją wartość maksymalną to ustawiam go spowrotem na zero.

4.9 vctl.v

Ten moduł kontroluje generowanie adresów odczytu na podstawie aktualnych pozycji pikseli i linii oraz obszarów aktywnych obrazu.

Listing 9: Moduł vctl.v

```
1  `timescale 1ns/1ps
2
3  // Video Controller - read address generator
4
5  module vctl #(
6      parameter XWIDTH = 10,
7      parameter YWIDTH = 10,
8       parameter AWIDTH = 16,
9      parameter XMAX = 799,
10     parameter YMAX = 524,
11      parameter HDMIN = 3,
12      parameter HDMAX = 643,
13      parameter VDMIN = 524,
14      parameter VDMAX = 479
15 ) (
16     input wire PixelClk,
17     output reg[XWIDTH-1:0] PixelCnt,
18     output reg[YWIDTH-1:0] LineCnt,
19      output wire[AWIDTH-1:0] AddrOut,
20      output reg AddrClkOut,
21      output reg IsActHorz,
22      output reg IsActVert
23 );
24
25 initial PixelCnt <= 1'b0;
26 initial LineCnt <= 1'b0;
27 initial IsActHorz <= 1'b0;
28 initial IsActVert <= 1'b0;
29
30 assign AddrOut = { 2'b00, LineCnt[9:2], 1'b0, PixelCnt[9:2] };
31
32 always @(posedge PixelClk) begin
33     if (PixelCnt == XMAX) begin
34         PixelCnt <= 1'b0;
```

```
35          if (LineCnt == YMAX) LineCnt <= 1'b0;
36          else LineCnt <= LineCnt + 1'b1;
37      end else PixelCnt <= PixelCnt + 1'b1;
38
39      if (PixelCnt == HDMAX) IsActHorz <= 1'b0;
40      if (PixelCnt == HDMIN) IsActHorz <= 1'b1;
41
42      if (LineCnt == VDMAX) IsActVert <= 1'b0;
43      if (LineCnt == VDMIN) IsActVert <= 1'b1;
44
45      if (&PixelCnt[1:0] && IsActHorz) AddrClkOut <= 1'b1;
46      else AddrClkOut <= 1'b0;
47  end
48
49  endmodule
```

Parametry:

- `XWIDTH = 10`: Liczba bitów używanych do liczenia pikseli w poziomie.
- `YWIDTH = 10`: Liczba bitów używanych do liczenia linii w pionie.
- `AWIDTH = 16`: Liczba bitów używanych do adresowania pamięci.
- `XMAX = 799`: Maksymalna wartość licznika pikseli.
- `YMAX = 524`: Maksymalna wartość licznika linii.
- `HDMIN = 3`: Minimalna wartość licznika pikseli dla obszaru aktywnego poziomego.
- `HDMAX = 643`: Maksymalna wartość licznika pikseli dla obszaru aktywnego poziomego.
- `VDMIN = 524`: Minimalna wartość licznika linii dla obszaru aktywnego pionowego.
- `VDMAX = 479`: Maksymalna wartość licznika linii dla obszaru aktywnego pionowego.

Wejścia:

- `PixelClk`: Sygnał zegarowy używany do synchronizacji operacji.

Wyjścia:

- `PixelCnt[XWIDTH-1:0]`: Licznik pikseli, inkrementowany wraz z każdym nowym pikselem.
- `LineCnt[YWIDTH-1:0]`: Licznik linii, inkrementowany wraz z każdą nową linią.
- `AddrOut[AWIDTH-1:0]`: Adres wyjściowy, używany do odczytu danych z pamięci.
- `AddrClkOut`: Sygnał zegarowy dla adresu wyjściowego.
- `IsActHorz`: Sygnał informujący, czy generator obrazu pracuje w obszarze aktywnym poziomym.

- `IsActVert`: Sygnał informujący, czy generator obrazu pracuje w obszarze aktywnym pionowym.

Licznik pikseli (PixelCnt) jest inkrementowany z każdym narastającym zboczem zegara piksela (PixelClk). Gdy osiągnie maksymalną wartość (XMAX), zostaje wyzerowany, a licznik linii (LineCnt) jest inkrementowany. Sygnał IsActHorz jest ustawiany na wysoki poziom, gdy licznik pikseli znajduje się w obszarze aktywnym poziomym (od HDMIN do HDMAX), w przeciwnym razie jest ustawiany na niski poziom. Sygnał IsActVert jest ustawiany na wysoki poziom, gdy licznik linii znajduje się w obszarze aktywnym pionowym (od VDMIN do VDMAX), w przeciwnym razie jest ustawiany na niski poziom. Adres wyjściowy (AddrOut) jest generowany na podstawie aktualnych wartości liczników pikseli i linii. Sygnał zegarowy dla adresu wyjściowego (AddrClkOut) jest aktywny tylko gdy dwie najmniej znaczące bity licznika pikseli są ustawione na 1 i sygnał IsActHorz jest aktywny.

4.10 vmmu.v

Ten moduł VMMU (Video Memory Management Unit) kontroluje dostęp do pamięci poprzez przypisywanie czasowych slotów do operacji odczytu i zapisu. Dzięki temu zarządza przepływem danych między pamięcią wideo a innymi komponentami, kontrolując operacje odczytu i zapisu danych zgodnie z zaprogramowanymi slotami czasowymi, co umożliwia efektywne zarządzanie danymi w projekcie.

—bit—	7	6	5	4	3 - 2	1	0
—field—	NOP	-	-	-	SADDR	TDATA	R/W

Tabela 5: Struktura slotu pamięci

Powyższa tabela prezentuje wykorzystany w projekcie sposób zagospodarowania slotu pamięci.Kolejne bity przechowują następujące informacje:

1. **bit 0**: określa czy wykonywana będzie operacja zapisu lub odczytu danych
2. **bit 1**: specyfikuje bufor, z którego nastąpi zapis lub odczyt danych
3. **bity 2-3**: definiuje, z którego z czterech dostępnych adresów pamięci będziemy odczytywać informacje
4. **bity 4-6**: niewykorzystywane
5. **bit 7**: jeśli jest ustawiony nie należy podejmować żadnych operacji

Listing 10: Moduł vmmu.v

```
1 module vmmu #(
2     parameter AWIDTH = 19,
3     parameter DWIDTH = 8,
4     parameter TSSIZE = 8 // amount of time slots
```

```
5  ) (
6      input wire MemClk,
7
8      input wire[AWIDTH-1:0] ReqAddrSrc1,
9      input wire[AWIDTH-1:0] ReqAddrSrc2,
10     input wire[AWIDTH-1:0] ReqAddrSrc3,
11     input wire[AWIDTH-1:0] ReqAddrSrc4,
12
13     output reg[DWIDTH-1:0] ReqReadData1,
14     output reg ReadDataRdy1,
15     output reg[DWIDTH-1:0] ReqReadData2,
16     output reg ReadDataRdy2,
17     input wire[DWIDTH-1:0] ReqWriteData,
18      input wire HasWriteData,
19      output reg WriteDataRdy,
20
21     output reg[AWIDTH-1:0] MemAddrPort, // phy memory ports
22     inout wire[DWIDTH-1:0] MemDataPort,
23     output wire MemWriteEnable,
24     output wire MemOutputEnable
25 );
26
27 // memory time slot structure:
28 // bit 0   - read/write (0 = read, 1 = write)
29 // bit 1 - data dest/src (T - target id)
30 // bit 3:2 - address src (SS - source id)
31 // bit 6:4 - unused
32 // bit 7 - nop
33
34 parameter SlotsFile = "/home/ise/FPGA_SHARED/vga_src/slots.bin.mem";
35
36 reg[1:0] Phase = 1'b00;
37 reg MemNop = 1'b0;
38 reg WriteMode = 1'b0;
39 reg[7:0] Slots[TSSIZE-1:0];
40 reg[7:0] Slot;
41 reg[2:0] SlotIndex = 3'b000;
42
43 wire[7:0] ReqReadData;
44 reg[AWIDTH-1:0] NextAddr;
45
46 //initial begin
47 //    if (SlotsFile != "") $readmemb(SlotsFile, Slots);
48 //end
49
50 initial begin
51      Slots[0] = 8'b00000101; // write byte 1
52     Slots[1] = 8'b10000101; // noop write byte 2
53     Slots[2] = 8'b10000101; // noop write byte 3
54      Slots[3] = 8'b10000000; // noop
55      Slots[4] = 8'b10000000; // noop
56     Slots[5] = 8'b00000000; // read byte 1
57     Slots[6] = 8'b10000000; // noop read byte 2
58     Slots[7] = 8'b10000000; // noop read byte 3
59 end
60
61 localparam PhaseAddressSetup = 2'b00;
62 localparam PhaseDirectionChange = 2'b01;
63 localparam PhaseDataSetup = 2'b10;
64 localparam PhaseDataSample = 2'b11;
65
66 assign MemDataPort = !MemWriteEnable ? ReqWriteData : 8'bzzzzzzzz;
67 assign ReqReadData = MemDataPort;
68
69 assign MemWriteEnable = !WriteMode | !WriteDataRdy;
70 assign MemOutputEnable = WriteMode;
```

```
71
72  always @(posedge MemClk) begin
73      ReadDataRdy1 <= 1'b0;
74       ReadDataRdy2 <= 1'b0;
75
76       case (Phase)
77          PhaseAddressSetup: begin
78              Slot <= Slots[SlotIndex];
79              SlotIndex <= SlotIndex + 1'b1;
80
81                case (Slot[3:2])
82                  2'b00: MemAddrPort = ReqAddrSrc1;
83                  2'b01: MemAddrPort = ReqAddrSrc2;
84                  2'b10: MemAddrPort = ReqAddrSrc3;
85                  2'b11: MemAddrPort = ReqAddrSrc4;
86              endcase
87              WriteMode = Slot[0];
88                MemNop = Slot[7];
89                  if (WriteMode && !MemNop) WriteDataRdy = 1'b1;
90            end
91          PhaseDirectionChange: begin
92                // empty
93            end
94          PhaseDataSetup: begin
95                // empty
96            end
97          PhaseDataSample: begin
98                WriteDataRdy = 1'b0;
99                if (!WriteMode && !MemNop) begin
100                 case (Slot[1])
101                     1'b0: begin
102                         ReqReadData1 <= ReqReadData;
103                            ReadDataRdy1 <= 1'b1;
104                     end
105                     1'b1: begin
106                         ReqReadData2 <= ReqReadData;
107                              ReadDataRdy2 <= 1'b1;
108                     end
109                 endcase
110             end
111           end
112       endcase
113
114       Phase <= Phase + 1;
115 end
116
117 endmodule
```

Wejścia i wyjścia:

1. **input wire MemClk**: Wejście sygnału zegar pamięci
2. **input wire[AWIDTH-1:0] ReqAddrSrc1**: Pierwszy żądany adres danych do odczytu
3. **input wire[AWIDTH-1:0] ReqAddrSrc2**: Drugi żądany adres danych do odczytu
4. **input wire[AWIDTH-1:0] ReqAddrSrc3**: Trzeci żądany adres danych do odczytu
5. **input wire[AWIDTH-1:0] ReqAddrSrc4**: Czwarty żądany adres danych do odczytu
6. **output reg[DWIDTH-1:0] ReqReadData1**: Wyjście danych odczytanych z pamięci
7. **output reg ReadDataRdy1**: Sygnał wejściowy sterującym odczytem z pamięci

8. **output reg[DWIDTH-1:0] ReqReadData2**: Wyjście danych odczytanych z pamięci
9. **output reg ReadDataRdy2**: Sygnał wejściowy sterującym zapisem do pamięci
10. **input wire[DWIDTH-1:0] ReqWriteData**: Wejście reprezenetujące dane odczytane z pamięci
11. **input wire HasWriteData**: Sygnał wyzwalający zapis danych
12. **output reg WriteDataRdy**: Sygnał informujący o gotowości zapisu danych
13. **output reg[AWIDTH-1:0] MemAddrPort**: Wejście reprezentujace fizyczne porty pamięci
14. **inout wire[DWIDTH-1:0] MemDataPort**: Wejście/wyjście do zapisu/odczytu danych z pamięci
15. **output reg MemWriteEnable**: Sygnał sterujący zapisem do pamięci
16. **output reg MemOutputEnable**: Sygnał sterujący przekazywaniem danych z pamięci na wyjście

Następnie tworzymy następujące rejestry i połączenia:

1. **SlotsFile**: Ścieżka do pliku zawierającego struktury czasowe (time slots) pamięci
2. **reg Phase**: Rejstr kontrolujący fazę działania modułu.
3. **reg MemNop**: Rejstr wskazujący, czy bieżąca struktura czasowa jest operacją nop (no operation).
4. **reg WriteMod**: Rejestr określający tryb operacji (czy to operacja zapisu czy odczytu)
5. **reg [7:0] Slots[TSSIZE-1:0]**: Tablica rejestrów zawierająca struktury czasowe pamięci.
6. **reg [7:0] Slot**: Bieżąca struktura czasowa.
7. **reg [2:0] SlotIndex**: Indeks bieżącej struktury czasowej.
8. **wire[7:0] ReqReadData**: Dane odczytane z pamięci.
9. **wire[AQWIDTH-1:0] NextAddr**: Rejestr przechowujący adres następnego żądania zapisu

Tworzymy następujące przypisania:

$$assign MemDataPort = !MemWriteEnable ? ReqWriteData : 8'bzzzzzzzz$$

$$assign ReqReadData = MemDataPort;$$

$assignMemWriteEnable = !WriteEnable | WriteDataRdy;$

$assignMemOutputEnable = WriteEnable$

Zastosowanie pierwszego przypisania jest takie, że gdy nie jest wykonywany zapis do pamięci (!MemWriteEnable), MemDataPort jest ustawiany na ReqWriteData, czyli dane, które mają być zapisane w pamięci. W przeciwnym razie, MemDataPort jest ustawiany na wartość 8'bzzzzzzzz, co oznacza, że w tym przypadku wartość na MemDataPort jest niezdefiniowana.Drugie przypisuje dane z MemDataPort do sygnału ReqReadData, który jest wyjściem zawierającym dane odczytane z pamięci.Przypisanie MemWriteEnable = !WriteEnable — WriteDataRdy; oznacza, że MemWriteEnable będzie aktywne, gdy WriteEnable nie jest aktywne (czyli nie ma próby zapisu) lub jeśli WriteDataRdy jest aktywne (czyli dane są gotowe do zapisu).Ostatnie przypisanie oznacza, że MemOutputEnable będzie aktywne tylko wtedy, gdy WriteEnable będzie aktywne. To oznacza, że dane z pamięci będą przekazywane na wyjście tylko wtedy, gdy jest aktywna próba zapisu do pamięci.
Na początku fazy zegara, sygnały ReadDataRdy1, ReadDataRdy2 są resetowane do stanu niskiego, co oznacza, że moduł nie jest gotowy do przekazania danych odczytanych z pamięci.Następnie, w zależności od fazy zegara (Phase), podejmowane są poszczególne działania:

- **Phase AddressSetup** : Odpowiada za ustawianie adresów na podstawie wartości slotów czasowych. Początkowo bieżąca struktura czasowa Slot jest aktualizowana na podstawie wartości przechowywanej w tablicy Slots pod indeksem SlotIndex, który jest następnie inkrementowany o 1.Nastepnie adres portu pamięci MemAddrPort jest ustawiany na podstawie dwóch bitów Slot (to jest Slot[3:2]), co determinuje, który adres z żądań dostępu do pamięci (ReqAddrSrc1 do ReqAddrSrc4) będzie używany. Kolejno, sygnał WriteEnable jest ustawiany na wartość bitu Slot[0], co determinuje, czy operacja zapisu do pamięci jest aktywna.Rejestr MemNop jest ustawianay na podstawie siódmego bitu slotu, co określa, czy w bieżącym slocie nie będą podjemowane żadne operacje.Jeśli jest próba zapisu (WriteEnable jest aktywne) i bieżący slot czasowy nie jest operacją "nop" (!MemNop), wyjście WriteDataRdy jest ustawiane na aktywne , wskazując, że dane są gotowe do zapisu.
- **PhaseDirectionChange**:
- **PhaseDataSetup**:
- **PhaseDataSample**: Odpowiada za próbkowanie danych z pamięci w module VMMU.Na początku, sygnał WriteDataRdy jest ustawiany na stan niski (0), co oznacza, że dane nie są gotowe do zapisu.Następnie sprawdzane są warunki, czy moduł nie jest w trybie zapisu (!WriteMode) i czy bieżący slot czasowy nie jest operacją "nop" (!MemNop). Jeśli oba warunki są spełnione to na podstawie wartości znajudjące się

w rejestrze Slot pod pierwszym indeksem, odczytane dane z pamięci (ReqReadData) są kierowane do odpowiedniego wyjścia (ReqReadData1 lub ReqReadData2), a odpowiednie sygnały gotowości do odczytu danych (ReadDataRdy1 lub ReadDataRdy2) są ustawiane na stan wysoki

Na końcu każdej iteracji fazy zegara Phase jest inkrementowana o 1, aby przesunąć moduł do następnej fazy w następnym cyklu zegara.

4.11 vmmudummy.v

Moduł do testowania modułu zarządzania pamięcią dla danych o małym rozmiarze (256B).
Wejścia i wyjścia:

- *MemClk* (`reg`): Sygnał zegara używany do synchronizacji operacji pamięci.
- *ReqAddrSrc1* (`reg [18:0]`): Adres żądania od pierwszego źródła do odczytu danych z pamięci.
- *ReqAddrSrc2* (`reg [18:0]`): Adres żądania od drugiego źródła do odczytu danych z pamięci.
- *ReqAddrSrc3* (`reg [18:0]`): Adres żądania od trzeciego źródła do odczytu danych z pamięci.
- *ReqAddrSrc4* (`reg [18:0]`): Adres żądania od czwartego źródła do odczytu danych z pamięci.
- *ReqWriteData* (`reg [7:0]`): Dane do zapisania w pamięci.
- *WriteDataTrig* (`reg`): Sygnał wyzwalający operację zapisu danych do pamięci.
- *ReqReadData1* (`wire [7:0]`): Dane odczytane z pamięci na podstawie *ReqAddrSrc1*.
- *ReadDataRdy1* (`wire`): Flaga gotowości odczytu dla danych z *ReqAddrSrc1*.
- *ReqReadData2* (`wire [7:0]`): Dane odczytane z pamięci na podstawie *ReqAddrSrc2*.
- *ReadDataRdy2* (`wire`): Flaga gotowości odczytu dla danych z *ReqAddrSrc2*.
- *WriteDataRdy* (`wire`): Flaga gotowości do zapisu danych.
- *MemAddrPort* (`wire [18:0]`): Adres pamięci, na który będą zapisywane lub z którego będą odczytywane dane.
- *MemWriteEnable* (`wire`): Sygnał umożliwiający zapis do pamięci.

- *MemOutputEnable* (`wire`): Sygnał umożliwiający odczyt z pamięci.
- *MemDataPort* (`wire [7:0]`): Dwukierunkowy port danych pamięci, używany do zarówno odczytu jak i zapisu danych.

Rejestry :

- *DummyMem* (`reg [7:0] [255:0]`): Symulowana pamięć o rozmiarze 256 bajtów używana do testowania.

Początkowo w bloku initial wszystkie wejścia są inicjalizowane na 0. Przy każdym zboczu narastającym sygnału MemWriteEnable wykonywane są następujące operacje MemDataPort jest ustawiany w stan wysokiej impedancji (8'bzzzzzzzz). Po 10 jednostkach czasu MemDataPort przyjmuje wartość z symulowanej pamięci DummyMem na adresie MemAddrPort[7:0].

Listing 11: Moduł vmmudummy.v

```
1  `timescale 1ns / 1ps
2
3  // dummy memory intended for testing
4  // because of small amount of data being tested it has a small size of 256B
5
6  module vmmudummy;
7
8      // Inputs
9      reg MemClk;
10     reg [18:0] ReqAddrSrc1;
11     reg [18:0] ReqAddrSrc2;
12     reg [18:0] ReqAddrSrc3;
13     reg [18:0] ReqAddrSrc4;
14     reg [7:0] ReqWriteData;
15     reg WriteDataTrig;
16
17     // Outputs
18     wire [7:0] ReqReadData1;
19     wire ReadDataRdy1;
20     wire [7:0] ReqReadData2;
21     wire ReadDataRdy2;
22     wire WriteDataRdy;
23     wire [18:0] MemAddrPort;
24     wire MemWriteEnable;
25     wire MemOutputEnable;
26
27     // Bidirs
28     wire [7:0] MemDataPort;
29
30     // Instantiate the Unit Under Test (UUT)
31     vmmu uut (
32         .MemClk(MemClk),
33         .ReqAddrSrc1(ReqAddrSrc1),
34         .ReqAddrSrc2(ReqAddrSrc2),
35         .ReqAddrSrc3(ReqAddrSrc3),
36         .ReqAddrSrc4(ReqAddrSrc4),
37         .ReqReadData1(ReqReadData1),
38         .ReadDataRdy1(ReadDataRdy1),
39         .ReqReadData2(ReqReadData2),
40         .ReadDataRdy2(ReadDataRdy2),
41         .ReqWriteData(ReqWriteData),
42         .WriteDataTrig(WriteDataTrig),
43         .WriteDataRdy(WriteDataRdy),
```

```
44          .MemAddrPort(MemAddrPort),
45          .MemDataPort(MemDataPort),
46          .MemWriteEnable(MemWriteEnable),
47          .MemOutputEnable(MemOutputEnable)
48      );
49
50      // physical dummy memory interface
51      reg [7:0] DummyMem[255:0] = 0;
52
53      always @(posedge MemWriteEnable) begin
54          MemDataPort = 8'bzzzzzzzz;
55          #10;
56          MemDataPort = DummyMem[MemAddrPort[7:0]];
57      end
58
59      initial begin
60          // Initialize Inputs
61          MemClk = 0;
62          ReqAddrSrc1 = 0;
63          ReqAddrSrc2 = 0;
64          ReqAddrSrc3 = 0;
65          ReqAddrSrc4 = 0;
66          ReqWriteData = 0;
67          WriteDataTrig = 0;
68      end
69
70  endmodule
```

4.12 vsig.v

Ten moduł odpowiada za generowanie sygnałów synchronizacji poziomej (HSync) i pionowej (VSync) w projekcie generatora obrazu VGA o standardzie 640x480x60Hz.

Listing 12: Moduł vsig.v

```
1   `timescale 1ns/1ps
2
3   // Video Signalling Generator - generate HSync, VSync, blanks video RGB output
4
5   module vsig #(
6       parameter XWIDTH = 10,
7       parameter YWIDTH = 10,
8        parameter HSMIN = 653,
9        parameter HSMAX = 749,
10       parameter VSMIN = 491,
11       parameter VSMAX = 493
12  ) (
13      input wire PixelClk,
14      input wire[XWIDTH-1:0] PixelCnt,
15      input wire[YWIDTH-1:0] LineCnt,
16      input wire IsActHorz,
17       input wire IsActVert,
18      output reg HSync,
19      output reg VSync,
20       output wire Blank
21  );
22
23  assign Blank = !IsActHorz || !IsActVert;
24
25  always @(posedge PixelClk) begin
26      if (PixelCnt == HSMIN) HSync <= 1'b0;
27      if (PixelCnt == HSMAX) HSync <= 1'b1;
28
```

```
29      if (LineCnt == VSMIN) VSync <= 1'b0;
30      if (LineCnt == VSMAX) VSync <= 1'b1;
31  end
32
33  endmodule
```

Parametry:

- `XWIDTH = 10`: Liczba bitów używanych do liczenia pikseli w poziomie.
- `YWIDTH = 10`: Liczba bitów używanych do liczenia linii w pionie.
- `HSMIN = 653`: Minimalna wartość licznika pikseli dla sygnału synchronizacji poziomej (HSync).
- `HSMAX = 749`: Maksymalna wartość licznika pikseli dla sygnału synchronizacji poziomej (HSync).
- `VSMIN = 491`: Minimalna wartość licznika linii dla sygnału synchronizacji pionowej (VSync).
- `VSMAX = 493`: Maksymalna wartość licznika linii dla sygnału synchronizacji pionowej (VSync).

Wejścia:

- `PixelClk`: Sygnał zegarowy używany do synchronizacji operacji.
- `PixelCnt[XWIDTH-1:0]`: Licznik pikseli, który inkrementuje się wraz z kolejnymi pikselami generowanymi na wyjściu.
- `LineCnt[YWIDTH-1:0]`: Licznik linii, który inkrementuje się wraz z kolejnymi liniami generowanymi na wyjściu.
- `IsActHorz`: Sygnał informujący, czy generator obrazu pracuje w obszarze aktywnym poziomym.
- `IsActVert`: Sygnał informujący, czy generator obrazu pracuje w obszarze aktywnym pionowym.

Wyjścia:

- `HSync`: Sygnał synchronizacji poziomej.
- `VSync`: Sygnał synchronizacji pionowej.
- `Blank`: Sygnał informujący, czy obecnie generowany piksel jest w obszarze blankingu (czyli poza obszarem aktywnym).

Z każdym rosnącym zboczem zegara następuje generowanie sygnałów synchronizacji poziomej (HSync) i pionowej (VSync) na podstawie aktualnych wartości licznika pikseli (PixelCnt) i licznika linii (LineCnt).Gdy wartość licznika pikseli osiągnie minimalną wartość (HSMIN), sygnał HSync zostanie ustawiony na niski poziom.Gdy wartość licznika pikseli osiągnie maksymalną wartość (HSMAX), sygnał HSync zostanie ustawiony na wysoki poziom.Gdy wartość licznika linii osiągnie minimalną wartość (VSMIN), sygnał VSync zostanie ustawiony na niski poziom. Gdy wartość licznika linii osiągnie maksymalną wartość (VSMAX), sygnał VSync zostanie ustawiony na wysoki poziom.

4.13 vga.v

Jest to główny moduł naszego projektu, w którym wykorzystując wcześniej zaimplementowane moduły, będziemy generować sygnał VGA o rozdzielczości 640 na 480 oraz częstotliwości odświeżania 60Hz.W celu uzyskania stablinie wyświetlanego obrazu należało odpowiednio skonfigurować zegar główny oraz oba liczniki.

Listing 13: Moduł vga.v

```
1  `timescale 1ns/1ps
2  `include "vctl.v"
3  `include "vbuffer.v"
4  `include "vmmu.v"
5
6  module vga (
7      input wire MainClkSrc,
8      output wire[18:0] MemAddr,
9      inout wire[7:0] MemData,
10      output wire TestOut1,
11      output wire TestOut2,
12     output wire MemWE,
13     output wire MemOE,
14     output wire[5:0] ColorOut,
15     output wire HsyncOut,
16     output wire VsyncOut,
17     input wire Sclk,
18     input wire Mosi,
19     input wire CSel
20 );
21
22 wire PixelClkIbufg;
23 wire PixelClkBufg;
24 wire PixelClkDcmOut;
25 wire PixelClkSrc;
26
27 wire PixelDcmRst;
28 wire[7:0] PixelDcmStatus;
29 wire PixelDcmLocked;
30 wire PixelDcmClkFxStopped = PixelDcmStatus[2];
31
32 assign PixelDcmRst = PixelDcmClkFxStopped & ~PixelDcmLocked;
33
34 wire MemClkIbufg;
35 wire MemClkBufg;
36 wire MemClkDcmOut;
37 wire MemClkSrc;
38
39 wire MemDcmRst;
40 wire[7:0] MemDcmStatus;
41 wire MemDcmLocked;
```

```
42  wire MemDcmClkFxStopped = MemDcmStatus[2];
43
44  assign MemDcmRst = MemDcmClkFxStopped & ~MemDcmLocked;
45
46  IBUFG MainClkIbufgInst(
47      .I(MainClkSrc),
48      .O(MainClkIbufg)
49  );
50
51  // pixel clock using synthesized clock@25MHz
52  DCM_SP #(
53      .CLKIN_PERIOD(10), // 10ns
54      .CLK_FEEDBACK("1X"),
55      .CLKDV_DIVIDE(2.0), // not used
56      .CLKFX_MULTIPLY(2),
57      .CLKFX_DIVIDE(8)
58  ) PixelClkDcmInst (
59      .CLKIN(MainClkIbufg),
60      .CLKFB(1'b0),
61      .RST(PixelDcmRst),
62      .PSEN(1'b0),
63      .PSINCDEC(1'b0),
64      .PSCLK(1'b0),
65      .CLK0(),
66      .CLK90(),
67      .CLK180(),
68      .CLK270(),
69      .CLK2X(),
70      .CLK2X180(),
71      .CLKDV(),
72      .CLKFX(PixelClkDcmOut),
73      .CLKFX180(),
74      .STATUS(PixelDcmStatus),
75      .LOCKED(PixelDcmLocked),
76      .PSDONE(),
77      .DSSEN(1'b0)
78  );
79
80  BUFG PixelClkBufgInst (
81      .I(PixelClkDcmOut),
82      .O(PixelClkSrc)
83  );
84
85  // memory clock using synthesized clock@200MHz
86  DCM_SP #(
87      .CLKIN_PERIOD(10), // 10ns
88      .CLK_FEEDBACK("NONE"),
89      .CLKDV_DIVIDE(2.0), // not used
90      .CLKFX_MULTIPLY(4),
91      .CLKFX_DIVIDE(2) // 4 for 50MHz
92  ) MemClkDcmInst (
93      .CLKIN(MainClkIbufg),
94      .CLKFB(1'b0),
95      .RST(MemDcmRst),
96      .PSEN(1'b0),
97      .PSINCDEC(1'b0),
98      .PSCLK(1'b0),
99      .CLK0(),
100     .CLK90(),
101     .CLK180(),
102     .CLK270(),
103     .CLK2X(),
104     .CLK2X180(),
105     .CLKDV(),
106     .CLKFX(MemClkDcmOut),
107     .CLKFX180(),
```

```
108     .STATUS(MemDcmStatus),
109     .LOCKED(MemDcmLocked),
110     .PSDONE(),
111     .DSSEN(1'b0)
112 );
113 
114 BUFG MemClkBufgInst (
115     .I(MemClkDcmOut),
116     .O(MemClkSrc)
117 );
118 
119 wire ActHorz;
120 wire ActVert;
121 wire[18:0] PixelAddr;
122 reg CycleReadAddr = 1;
123 
124 wire[9:0] PixelCnt;
125 wire[9:0] LineCnt;
126 
127 vctl #(
128     .XWIDTH(10),
129     .YWIDTH(10),
130      .AWIDTH(19),
131     .XMAX(799),
132     .YMAX(524),
133      .HDMIN(3),
134      .HDMAX(643),
135      .VDMIN(524),
136      .VDMAX(479)
137 ) VideoCtl (
138     .PixelClk(PixelClkSrc),
139     .PixelCnt(PixelCnt),
140     .LineCnt(LineCnt),
141      .AddrOut(PixelAddr),
142      .AddrClkOut(),
143      .IsActHorz(ActHorz),
144      .IsActVert(ActVert)
145 );
146 
147 vsig #(
148     .XWIDTH(10),
149      .YWIDTH(10)
150 ) VideoSig (
151     .PixelClk(PixelClkSrc),
152     .PixelCnt(PixelCnt),
153     .LineCnt(LineCnt),
154      .IsActHorz(ActHorz),
155      .IsActVert(ActVert),
156      .HSync(HsyncOut),
157      .VSync(VsyncOut),
158      .Blank(Blank)
159 );
160 
161 wire SpiByteRdy;
162 wire[7:0] SpiByteRecv;
163 
164 spi Spi (
165     MemClkSrc,
166     Sclk,
167      Mosi,
168      CSel,
169      SpiByteRdy,
170      SpiByteRecv
171 );
172 
173 assign TestOut1 = SpiByteRdy;
```

```
174 assign TestOut2 = Sclk;
175
176 wire[7:0] ReadData;
177 wire[7:0] _ReadData;
178 wire ReadRdy;
179 wire _ReadRdy;
180
181 wire WriteDataOutClk;
182 wire HasWriteData;
183
184 wire[7:0] WriteData;
185 wire[18:0] WriteAddr;
186
187 reg[18:0] ReadAddr = 19'b000_0000_0000_0000_0000;
188
189 vcmd VideoCmd (
190      SpiByteRdy,
191     SpiByteRecv,
192      WriteDataOutClk,
193      HasWriteData,
194      WriteAddr,
195      WriteData
196 );
197
198 vmmu #(
199     .AWIDTH(19),
200     .DWIDTH(8),
201     .TSSIZE(8)
202 ) VMMU (
203     .MemClk(MemClkSrc),
204     .ReqAddrSrc1(PixelAddr),
205     .ReqAddrSrc2(WriteAddr),
206     .ReqAddrSrc3(1'b0),
207     .ReqAddrSrc4(1'b0),
208     .ReqReadData1(ReadData),
209     .ReadDataRdy1(ReadRdy),
210     .ReqReadData2(_ReadData),
211     .ReadDataRdy2(_ReadRdy),
212     .ReqWriteData(WriteData),
213      .HasWriteData(HasWriteData),
214      .WriteDataRdy(WriteDataOutClk),
215     .MemAddrPort(MemAddr),
216     .MemDataPort(MemData),
217     .MemWriteEnable(MemWE),
218     .MemOutputEnable(MemOE)
219 );
220
221 vbuffer #(
222     .BPP(6)
223 ) VideoBuffer (
224     .PixelClk(PixelClkSrc),
225     .ReqWrite(ReadRdy),
226     .Blank(Blank),
227     .DataIn(ReadData),
228     .VideoOut(ColorOut)
229 );
230
231 endmodule
```

Wejścia i wyjścia:

1. **input wire MainClkSrc**: Wejście głównego zegara systemu
2. **output wire [18:0] MemAddr**: Adres pamięci

3. **inout wire[7:0] MemData**: Dane pamięci
4. **output wire TestOut1**: Pierwsze wyjście testowe
5. **output wire TestOut2**: Drugie wyjście testowe
6. **output wire MemWE**: Sygnał wyjściowy sterujący zapisem do pamięci.
7. **output wire MemOE**: Sygnał wyjściowy sterujący przekazywaniem danych z pamięci na wyjście.
8. **output wire [5:0] ColorOut**: Wyjście koloru obrazu.
9. **output reg HsyncOut**: Rejstrujące wyjście synchronizacja poziomej
10. **output reg VsyncOut**: Rejstrujące wyjście synchronizacja pionowej
11. **input wire Sclk**: Wejście zegara dla modułu spi
12. **input wire Mosi**: Wejście danych z mikrokontrolera(Master Out Slave In)
13. **input wire Csel**: Wejście selekcji układu (Chip Select).

Następnie tworzymy cztery połączenia: *PixelClkIbufg*, *PixelClkBufg*, *PixelClkDcmOut* *PixelClkSrc*, które odpowiednio reprezentują wejściowe i wyjściowe zegary podczas wzmacniania sygnału buforami oraz konfigurowania ustawień menadżerem zegara.

Kolejno wartość sygnału resetu menadżera zarządzania zegarem(PixelDcmRst) ustawiamy jako wynik koniunkcji zmiennej przechowującej informację o zatrzymaniu zegra(PixelDcmClkFxStopped) oraz negacji zmienej repreznetującej sygnał blokady zegara(PixelDcmLocked). Dzięki powyższej operacji mamy pewność że sygnał resetu jest aktywny tylko, gdy zegar jest zatrzymany i nie ma blokady na moduł zarządzania zegarem.

Następnie używamy interfesju bufora wejściowego(IBUFG) w celu wzmocnienia sygnału zegarowego.Kolejno wzomcniony poprzednio sygnał konfiguruje za pomocą modułu Digital Clock Manager (DCM) aby uzyskać stabliny zegar o taktowaniu 25MHz, czyli zgodnym ze standardem projketowanego sygnału.Na koniec, już ustabilizowany i skonfigurowany sygnał zegarowy jest podawany na interfejs bufora globalnego (BUFG), który jest używany do dalszego wzmocnienia sygnału zegarowego i zapewnienia jego rozprowadzenia po całym układzie.

Wykonując analogiczne do wyżej opisanych kroków uzyskuję stabilny i odpo- wiednio skonfigurowany zegra dla pamięci o taktowaniu 100 MHz.

Następnie dołączamy moduły vctl, vsig odpowiedzialne za generowanie liczników pikseli i linii oraz adresów odczytu oraz sygnałów synchronizacji video oraz spi, vcmd odpowiedzialne za komunikację z pamięcią poprzez interfejs SPI i dekodowanie poleceń SPI, oraz moduł vmmu, kontrolujący dostęp do pamięci na podstawie adresów,

danych i sygnałów sterujących, zapewniając poprawne działanie operacji odczytu i zapisu.Załączamy, również moduł vbuffer służyący do buforowania danych wideo, co umożliwia odczytywanie pikseli zgodnie według reguł wyjaśnionych przy opisie owego modułu.

4.14 vgawritetest.v

Moduł vgawritetest.v jest testbenchem przeznaczonym do testowania modułu VGA poprzez symulację operacji zapisu danych.

Listing 14: Moduł vgawritetest.v

```
`timescale 1ns / 1ps

module vgawritetest;

    // Inputs
    reg MainClkSrc;
    reg Sclk;
    reg Mosi;
    reg CSel;

    // Outputs
    wire [18:0] MemAddr;
    wire MemWE;
    wire MemOE;
    wire [5:0] ColorOut;
    wire HsyncOut;
    wire VsyncOut;

    reg [8:0] Counter = 8'h00;

    // Bidirs
    wire [7:0] MemData;

    assign MemData = Counter[8:1];

    // Instantiate the Unit Under Test (UUT)
    vga UUT (
        .MainClkSrc(MainClkSrc),
        .MemAddr(MemAddr),
        .MemData(MemData),
        .MemWE(MemWE),
        .MemOE(MemOE),
        .ColorOut(ColorOut),
        .HsyncOut(HsyncOut),
        .VsyncOut(VsyncOut),
        .Sclk(Sclk),
        .Mosi(Mosi),
        .CSel(CSel)
    );

    task send_byte(input time PulseTime, input [7:0] Byte);
        begin
            CSel = 1'b0;
            // for (j = 0; j < 8; j = j + 1)
            #PulseTime;
            send_pulse(PulseTime, (Byte>>7) & 1);
            send_pulse(PulseTime, (Byte>>6) & 1);
            send_pulse(PulseTime, (Byte>>5) & 1);
            send_pulse(PulseTime, (Byte>>4) & 1);
            send_pulse(PulseTime, (Byte>>3) & 1);
            send_pulse(PulseTime, (Byte>>2) & 1);
            send_pulse(PulseTime, (Byte>>1) & 1);
```

```
53              send_pulse(PulseTime, (Byte>>0) & 1);
54              Sclk = 1'b0;
55              #PulseTime;
56              CSel = 1'b1;
57              #PulseTime;
58          end
59      endtask
60
61      task send_pulse(input time PulseTime, input Data);
62          begin
63              Mosi = Data;
64              Sclk = 1'b0;
65              #PulseTime;
66              Sclk = 1'b1;
67              #PulseTime;
68          end
69      endtask
70
71      initial begin
72          // Initialize Clock
73          MainClkSrc = 0;
74          forever #5 MainClkSrc = ~MainClkSrc;
75      end
76
77      integer i;
78
79      initial begin
80          // Initialize Inputs
81          Sclk = 0;
82          Mosi = 0;
83          CSel = 0;
84
85          // Wait 100 ns for global reset to finish
86          #500;
87
88          // Add stimulus here
89          // Very fast SPI clock (100MHz)
90          send_byte(10, 8'h20);
91          send_byte(10, 8'b11000000);
92          send_byte(10, 8'h20);
93          send_byte(10, 8'b11000000);
94          send_byte(10, 8'h20);
95          send_byte(10, 8'b11000000);
96
97          for (i = 0; i < 640; i = i + 1) begin
98              // Small delay between next pixel write
99              #160;
100             send_byte(10, 8'h11);
101             send_byte(10, 8'h00);
102             send_byte(10, 8'h10);
103
104             // Another byte at next location
105            send_byte(10, 8'h20);
106             send_byte(10, i);
107             send_byte(10, 8'h20);
108             send_byte(10, i);
109             send_byte(10, 8'h20);
110             send_byte(10, i);
111         end
112     end
113
114     always @(posedge MainClkSrc) Counter <= Counter + 1'b1;
115
116 endmodule
```

Na początku definiowane są wejścia (`MainClkSrc`, `Sclk`, `Mosi`, `CSel`) oraz wyjścia (`MemAddr`,

MemWE, MemOE, ColorOut, HsyncOut, VsyncOut, MemData) dla modułu VGA.Następnie definiowane są dwa zadania testowe send_pulse oraz textttsend_byte wykorzystywane do przesłania pojedynczego bajtu danych.Dokładny opis funkcjonalności owych zadaniach został przedstawiony w sekcji vcmdtest.v.W sekcji initial ustawiane są początkowe wartości sygnałów, oczekuje się 100 ns na zakończenie resetowania. Następnie generowane są impulsy SPI, symulujące wysyłanie danych do urządzenia. Każda wysyłka danych poprzedzona jest krótką przerwą, co symuluje niewielkie opóźnienie między kolejnymi operacjami, zapisywania kolejnych pikseli.

5 Bibliografia

- Standard sygnału 640x480: VGA-Timiing 640x480@60Hz
- Płytka rozwojowa fpga Spartan6: Mimas Spartan 6 FPGA development board
- Konfiguracja środowiska Xilinx ISE: Xilinx ISE Design Suite
- Język opisu sprzętu Verilog: IEEE Standard for Verilog Hardware Description Language
- Moduł pamięci: IS61LV5128AL 512K x 8 HIGH-SPEED CMOS STATIC RAM
- Konfiguracja Menadżera zegara cyfrowego:Xilinx Spartan-6 FPGA Clocking Resources User Guide
- Złącze vga: IBM Personal System/2 Hardware Interface Technical Reference
- Szeregowy interfejs urządzeń peryferyjnych(SPI) Serial Peripheral Interface