Wydział Elektroniki i Technik Informacyjnych

Systemy Komputerowe (SYKOM)

Obsługa rejestrów przez AXI-Lite

Kacper Capiga 331463
Karol Godlewski 331474

15 maja 2025

Spis treści

1 Cel laboratorium

Celem laboratorium jest zrozumienie oraz opanowanie obsługi protokołu AXI-Lite, poprzez napisanie modułu w języku Verilog, który umożliwia zapis i odczyt rejestrów z wykorzystaniem protokołu.

2 Użyte oprogramowanie

2.1 Icarus Verilog

Symulator języka opisu sprzętu Verilog, używany do symulacji i testowania projektów cyfrowych. Do pobrania ze strony https://bleyer.org/icarus/. Instalator zawiera również maszynę wirtualną vvp uruchamiającą pliki `.vvp`. Po zainstalowaniu korzysta się z niego za pomocą komendy `iverilog -o <file.vvp> <module.v>`, przyjmującej wiele argumentów pozycyjnych. Wygenerowany plik pośredni `.vvp` można następnie uruchomić za pomocą komendy `vvp <file.vvp>`, wówczas jeśli projekt zawierał instrukcje typu `$display` czy `$dumpfile` maszyna wirtualna wypisze informacje na standardowe wyjście lub do odpowiedniego pliku.

2.2 GTKWave

Narzędzie do wyświetlania przebiegów czasowych zapisanych w formacie VCD (ang. *Value Change Dump*) na licencji GNU do pobrania ze strony https://gtkwave.sourceforge.net/. Po zainstalowaniu narzędzia korzysta się z niego za pomocą komendy `gtkwave <file.vcd>`. Plik `.vcd` generuje się poprzez wyeksportowanie wszystkich zarejestrowanych sygnałów w module testującym w następujący sposób.

```
initial begin
  $dumpfile("dump.vcd"); // Nazwa pliku wynikowego
  $dumpvars(0, top_module); // Zarejestruj wszystkie sygnały z top_module
end
```

Kod 1: Eksportowanie zarejestrowanych sygnałów w module testującym

W trakcie pracy przydatna jest funkcja zapisywania konfiguracji, w tym wyświetlanych przebiegów, sposobu ich formatowania i kolorowania za pomocą funkcji `File > Write Save File As`. Zapisany plik konfiguracyjny można użyć jako trzeci argument pozycyjny przy korzystaniu z `gtkwave`. Plik konfiguracyjny nie zawiera informacji o samym przebiegu, jedynie sposób przedstawiania tych danych przez `gtkwave`.

3 Opis protokołu

Protokół AXI-Lite korzysta z 17 sygnałów podzielonych na 5 kanałów (2 kanały do odczytu i 3 do zapisu) opisanych w poniższych podsekcjach oraz sygnału zegarowego (`S_AXI_ACLK`) i resetu aktywnego w stanie niskim (`S_AXI_ARESETN`).

3.1 Operacja zapisu

Wszystkie sygnały zostały opisane z perspektywy urządzenia *slave*.

3.1.1 Write address channel

Jest to kanał służący do wymiany informacji o adresie, pod który *master* dokonuje zapisu. W skład tego kanału wchodzą następujące sygnały:

- `S_AXI_AWADDR` - 32 lub 64 bitowy (w zależności od architektury procesora) sygnał wejściowy oznaczający adres pod jaki *master* zapisuje dane,
- `S_AXI_AWVALID` - 1 bitowy sygnał wejściowy oznaczający, że adres wystawiony na szynie `S_AXI_AWADDR` jest prawidłowy,
- `S_AXI_AWREADY` - 1 bitowy sygnał wyjściowy oznaczający gotowość *slave'a* do przyjęcia adresu.

3.1.2 Write data channel

Kanał ten służy do przesyłania danych, które mają zostać zapisane pod adres wskazany w Write address channel. W jego skład wchodzą następujące sygnały:

- `S_AXI_WDATA` - 32 lub 64 bitowy (w zależności od architektury procesora) sygnał wejściowy, który zawiera DWORD lub QWORD (odpowiednio liczba 32 lub 64 bitowa) zapisania w rejestrze,
- `S_AXI_WSTRB` - 4 lub 8 bitowy (w zależności od architektury procesora) sygnał wejściowy, który oznacza bajty do zmodyfikowania w zapisywanym rejestrze,
- `S_AXI_WVALID` - 1 bitowy sygnał wejściowy oznaczający, że DWORD lub QWORD wystawiony na `S_AXI_WDATA` jest prawidłowy,
- `S_AXI_WREADY` - 1 bitowy sygnał wyjściowy oznaczający gotowość *slave'a* na przyjęcie danych do zapisu.

3.1.3 Write response channel

Ostatnim kanałem służącym do zapisu z wykorzystaniem AXI-Lite jest kanał odpowiedzi na zapis, który składa się z sygnałów:

- `S_AXI_BREADY` - 1 bitowy sygnał wejściowy oznaczający gotowość *master'a* na przyjęcie odpowiedzi,
- `S_AXI_BRESP` - 2 bitowy sygnał wyjściowy, odpowiedź *slave'a* na zapis,
- `S_AXI_BVALID` - 1 bitowy sygnał wyjściowy, oznaczający poprawność sygnału `S_AXI_BRESP`.

Transakcję zapisu można podzielić na 3 kroki:

1. *Master* wystawia adres na S_AXI_AWADDR, dane do zapisu na S_AXI_WDATA oraz oznacza bajty do zapisu na S_AXI_WSTRB. Oprócz tego podnosi sygnały kontrolne: S_AXI_AWVALID, S_AXI_WVALID oraz S_AXI_BREADY.

2. *Slave* podnosi sygnały S_AXI_AWREADY oraz S_AXI_WREADY.

3. *Handshake* został zakończony więc *master* może opuścić sygnały S_AXI_AWVALID oraz S_AXI_WVALID (może także zmienić dane na S_AXI_WSTRB, S_AXI_WDATA i S_AXI_AWADDR). *Slave* wystawia odpowiedź na zapis na S_AXI_BRESP oraz podnosi sygnał S_AXI_BVALID.

3.2 Operacja odczytu

Operacja odczytu wymaga wykorzystania 2 opisanych niżej kanałów.

3.2.1 Read address channel

Kanał ten zawiera analogiczne do Write address channel sygnały:

- S_AXI_ARADDR - 32 lub 64 bitowy (w zależności od architektury procesora) sygnał wyjściowy oznaczający adres spod jakiego *master* odczytuje dane,
- S_AXI_ARVALID - 1 bitowy sygnał wejściowy oznaczający, że adres wystawiony na szynie S_AXI_ARADDR jest prawidłowy,
- S_AXI_ARREADY - 1 bitowy sygnał wyjściowy oznaczający gotowość *slave'a* do przyjęcia adresu.

3.2.2 Read data channel

Kanał ten również jest bardzo podobny do Write data channel, z drobnymi różnicami: zawiera on dodatkowo S_AXI_RRESP (braku oddzielnego kanału na odpowiedź na odczyt) oraz brak kanału odpowiadającego za *strobe*. Wszystkie sygnały tego kanału to:

- S_AXI_RDATA - 32 lub 64 bitowy (w zależności od architektury procesora) sygnał wyjściowy, który zawiera DWORD lub QWORD (odpowiednio liczba 32 lub 64 bitowa) odczytane z rejestru,
- S_AXI_RVALID - 1 bitowy sygnał wyjściowy oznaczający, że DWORD lub QWORD wystawiony na S_AXI_WDATA jest prawidłowy,
- S_AXI_RREADY - 1 bitowy sygnał wejściowy oznaczający gotowość *master'a* na przyjęcie danych odczytanych spod wskazanego adresu,
- S_AXI_RRESP - 2 bitowy sygnał wyjściowy, odpowiedź *slave'a* na odczyt.

Transakcję odczytu można podzielić na 2 kroki:

1. *Master* wystawia adres na S_AXI_ARADDR i podnosi sygnały kontrolne S_AXI_ARVALID i S_AXI_RREADY, do momentu kiedy *slave* podniesie sygnał S_AXI_ARREADY.

2. *Handshake* został zakończony więc sygnały S_AXI_ARVALID (oraz S_AXI_ARADDR) mogą zostać opuszczone. *Slave* wystawia odczytane spod wskazanego rejestru dane na S_AXI_RDATA oraz podnosi S_AXI_RVALID i ustawia S_AXI_RRESP.

3.3 Znaczenie `S_AXI_BRESP` oraz `S_AXI_RRESP`

Sygnał `S_AXI_BRESP` może przyjąć jedną z następujących wartości:

- `00` (OKAY) - operacja zakończona sukcesem,
- `01` (EXOKAY) - operacja na wyłączność zakończona sukcesem (wartość nie używana w AXI-Lite, wykorzystywana tylko w pełnym AXI),
- `10` (SLVERR) - błąd *slave'a*.
- `11` (DECERR) - błąd dekodowania adresu.

4 Analiza kodu

4.1 Opis wejść i wyjść modułu AXI-Lite

Na listingu 2 widoczny jest kod w języku Verilog, który definiuje interfejs modułu. W skład interfejsu wchodzą sygnały omówione w sekcji 3. Oprócz tego moduł posiada parametr, który zmienia długość adresu, tym samym ograniczając liczbę możliwych do zaadresowania rejestrów do $2^{\texttt{ADDR\_WIDTH}-2}$.

```
module axil #(
    parameter ADDR_WIDTH = 4 // 4 registers: 0x00, 0x04, 0x08, 0x12
)(
    // Global signals
    input wire S_AXI_ACLK,
    input wire S_AXI_ARESETN,

    // AXI4-Lite slave interface

    // Write address channel
    input wire [ADDR_WIDTH-1:0] S_AXI_AWADDR, // Write address
    input wire S_AXI_AWVALID, // Write address valid
    output wire S_AXI_AWREADY, // Write address ready (slave is ready for address)

    // Write data channel
    input wire [31:0] S_AXI_WDATA, // Write data
    input wire [3:0] S_AXI_WSTRB, // Strobe (which bytes are valid)
    input wire S_AXI_WVALID, // Write data valid
    output wire S_AXI_WREADY, // Write data ready (slave is ready for data)

    // Write response channel
    input wire S_AXI_BREADY, // Response ready (Master can accept the response)
    output wire [1:0] S_AXI_BRESP, // Write response
    output wire S_AXI_BVALID, // Write response valid

    // Read address channel
    input wire [ADDR_WIDTH-1:0] S_AXI_ARADDR, // Read address
    input wire S_AXI_ARVALID, // Read address valid
    output wire S_AXI_ARREADY, // Read address ready (slave is ready for address)

    // Read data channel
    output wire [31:0] S_AXI_RDATA, // Read data
    output wire [1:0] S_AXI_RRESP, // Read response
    output wire S_AXI_RVALID, // Read valid (data on bus is valid)
    input wire S_AXI_RREADY // Read ready (master can read from bus)
);
```

Kod 2: Wejścia i wyjścia modułu

4.2 Wewnętrzne rejestry i kable

Listing 3 przedstawia definicję wewnętrznych rejestrów oraz kabli (ang. *wire*) modułu. Moduł zawiera dwa rejestry 32 bitowe do odczytu i zapisu przez AXI-Lite (`ctrl_reg` oraz `in_angle_reg`) oraz dwa rejestry 32 bitowe tylko do odczytu (`out_cos_reg` i `out_sin_reg`). Oprócz tego zawiera on jeszcze rejestry służące synchronizacji wyjść z sygnałem zegarowym.

```
    // Inner wires and registers
    wire axil_read_ready, axil_write_ready;
    reg axil_read_valid, axil_awready, axil_bvalid, axil_arready;
    reg [31:0] axil_read_data;

    // Our registers
    reg [31:0] ctrl_reg, in_angle_reg, out_cos_reg, out_sin_reg;

    // Strobe wires
    wire [31:0] ctrl_strb, in_angle_strb;
```

Kod 3: Zdefiniowanie rejestrów i połączeń wewnętrznych

4.3 Operacja zapisu

4.3.1 Write strobe

Listing 4 przedstawia funkcję aplikującą `S_AXI_WSTRB` do poszczególnych rejestrów. Generuje ona dla każdego z kabli po 4 przypisania warunkowe (oddzielne dla każdego bajtu), które przypisują każdemu bajtowi wartość zapisaną w rejestrze, jeżeli *strobe* dla tego bajtu jest zerem lub wartość wystawioną na `S_AXI_WDATA`.

```
    // Write strobe logic
    function [31:0] apply_wstrb;
                input [31:0] prior_data;
                input [31:0] new_data;
                input [3:0] wstrb;

                integer k;
                for(k = 0; k < 4; k = k+1)
                begin
                apply_wstrb[k*8 +: 8] = wstrb[k] ? new_data[k*8 +: 8] : prior_data[k*8 +: 8];
                end
        endfunction

    assign ctrl_strb = apply_wstrb(ctrl_reg, S_AXI_WDATA, S_AXI_WSTRB);
    assign in_angle_strb = apply_wstrb(in_angle_reg, S_AXI_WDATA, S_AXI_WSTRB);
```

Kod 4: Aplikowanie `S_AXI_WSTRB` do kabli, które następnie będą służyć do zapisu w rejestrach

4.3.2 Dekodowanie i zapis

Dekodowanie adresów i zapis do rejestrów (a także resetowanie rejestrów) przedstawiono na listingu 5. W momencie kiedy moduł jest gotowy do zapisu (`axil_write_ready` opisane w sekcji 4.3.3) następuje zapisanie wskazanego przez adres rejestru wartością po zaaplikowaniu `S_AXI_WSTRB`.

```
        // Write logic
        always @(posedge S_AXI_ACLK)
        if (!S_AXI_ARESETN)
        begin
        // Ctrl
                ctrl_reg <= 0;

        // Input
                in_angle_reg <= 0;
        end else if (axil_write_ready)
        begin
                case(S_AXI_AWADDR[ADDR_WIDTH-1:2])
                2'b00: ctrl_reg <= ctrl_strb;
                2'b01: in_angle_reg <= in_angle_strb;
                // out_sin_reg and out_cos_reg are ro (outputs)
                endcase
        end
```

Kod 5: Zapis do rejestrów

4.3.3 Gotowość na zapis

Wyjścia S_AXI_WREADY oraz S_AXI_AWREADY powinny być zsynchronizowane i podniesione w momencie kiedy *slave* nie wykonuje operacji zapisu, natomiast *master* wystawi S_AXI_WVALID oraz S_AXI_AWVALID. *Slave* nie jest jednak gotowy na zapis kolejnej wartości do rejestrów w momencie kiedy wysoko jest sygnał S_AXI_BVALID lub nisko S_AXI_BREADY, ponieważ oznacza to, że master nie odczytał jeszcze odpowiedzi na poprzedni zapis (poprzednia transakcja nie została zakończona). Wprowadzono również opóźnienie nie pozwalające na wystawienie sygnałów gotowości do zapisu wysoko przez 2 cykle zegarowe pod rząd (jest to rejestr, więc gdyby tego nie zrobiono, to *master* mógłby w tym samym cyklu wystawić S_AXI_BVALID oraz zażądać następnej operacji zapisu). Opis w języku Verilog wygląda tak jak na listingu 6.

```
    // Write ready signals
    always @(posedge S_AXI_ACLK)
    if (!S_AXI_ARESETN)
                axil_awready <= 1'b0;
        else
                axil_awready <= !axil_awready &&
                          (S_AXI_AWVALID && S_AXI_WVALID) &&
                          (!S_AXI_BVALID || S_AXI_BREADY);

    // AWREADY and WREADY are basically the same
    assign S_AXI_AWREADY = axil_awready;
    assign S_AXI_WREADY = axil_awready;

    assign axil_write_ready = axil_awready;
```

Kod 6: Sygnały S_AXI_WREADY oraz S_AXI_AWREADY

4.3.4 Odpowiedź na zapis

Sygnał odpowiedzi na zapis ma na stałe przypisaną wartość 2'b00 (OKAY), która jest prawidłowa dopiero w momencie wystawienia sygnału `S_AXI_BVALID`, którego podniesienie następuje po podniesieniu sygnałów gotowości do zapisu, aby w następnym cyklu zegarowym zakończyć transakcję (jeżeli *master* trzyma wysoko sygnał `S_AXI_BREADY`). Sytuacja ta przedstawiona jest w kodzie 7 poniżej.

```
    // Write response
        always @(posedge S_AXI_ACLK)
        if (!S_AXI_ARESETN)
                axil_bvalid <= 0;
        else if (axil_write_ready)
                axil_bvalid <= 1;
    else if (S_AXI_BREADY)
                axil_bvalid <= 0;

    assign S_AXI_BVALID = axil_bvalid;
```

Kod 7: Sygnał `S_AXI_BVALID`

4.4 Operacja odczytu

4.4.1 Dekodowanie i wystawianie DWORD z rejestru

Listing 8 przedstawia logikę odczytu z rejestrów, która jest analogiczna jak zapis, z tą różnicą, że można odczytywać spod wszystkich adresów.

```
    // Read logic
    always @(posedge S_AXI_ACLK)
    begin
    if (!S_AXI_ARESETN)
    begin
        axil_read_data <= 0;
        out_cos_reg <= 0;
        out_sin_reg <= 0;
    end else if (!S_AXI_RVALID || S_AXI_RREADY)
    begin
                case(S_AXI_ARADDR[ADDR_WIDTH-1:2])
            2'b00: axil_read_data <= ctrl_reg;
            2'b01: axil_read_data <= in_angle_reg;
            2'b10: axil_read_data <= out_cos_reg;
            2'b11: axil_read_data <= out_sin_reg;
                endcase
        end
    if (!axil_read_ready)
        axil_read_data <= 0;
    end
    assign S_AXI_RDATA = axil_read_data;
```

Kod 8: Dekodowanie adresu i operacja odczytu

4.4.2 Odpowiedź na odczyt

Podobnie jak w przypadku zapisu sygnał S_AXI_RVALID wystawiany jest w momencie kiedy pojawi się prawidłowy adres a *slave* jest gotowy na adres (dane na szynie S_AXI_RDATA wystawione są po otrzymaniu prawidłowego adresu), aby zakończyć transakcję w następnym cyklu zegara. Ustawienie S_AXI_ARREADY jako dopełnienia S_AXI_RVALID pozwala na niezwłoczne przejście z jednej operacji odczytu do drugiej.

```
    // Read response
    always @(posedge S_AXI_ACLK)
    if (!S_AXI_ARESETN)
        axil_read_valid <= 1'b0;
    else if (axil_read_ready)
        axil_read_valid <= 1'b1;
    else if (S_AXI_RREADY)
        axil_read_valid <= 1'b0;

    assign S_AXI_RVALID = axil_read_valid;

    // We are ready for read once address is valid and we are ready for address
    assign axil_read_ready = (S_AXI_ARVALID && S_AXI_ARREADY);

    // We are ready for next read when rvalid is down
    always @(*)
           axil_arready = !S_AXI_RVALID;

        assign S_AXI_ARREADY = axil_arready;
```

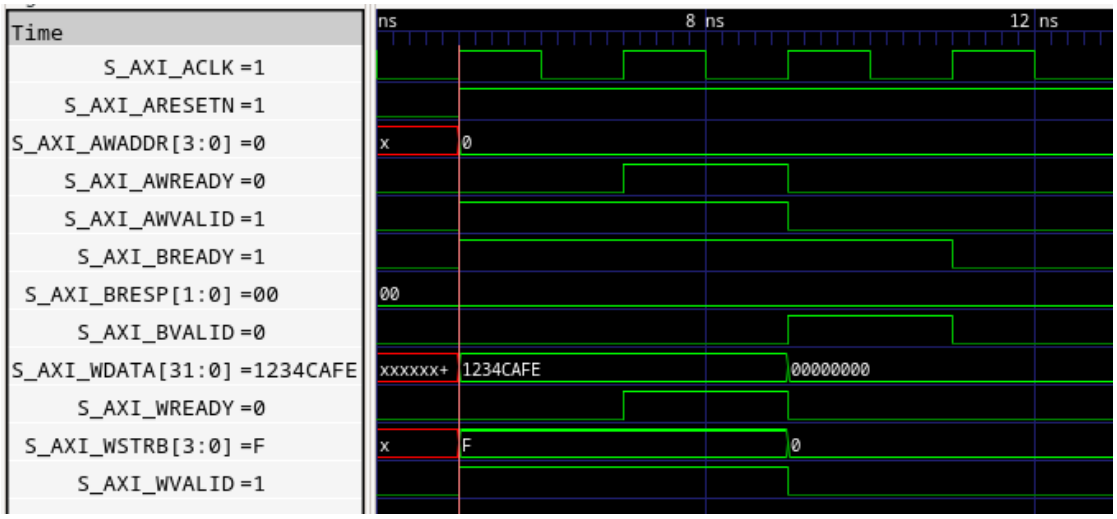
Kod 9: Odpowiedź na odczyt

5 Analiza wyników

5.1 Operacja zapisu

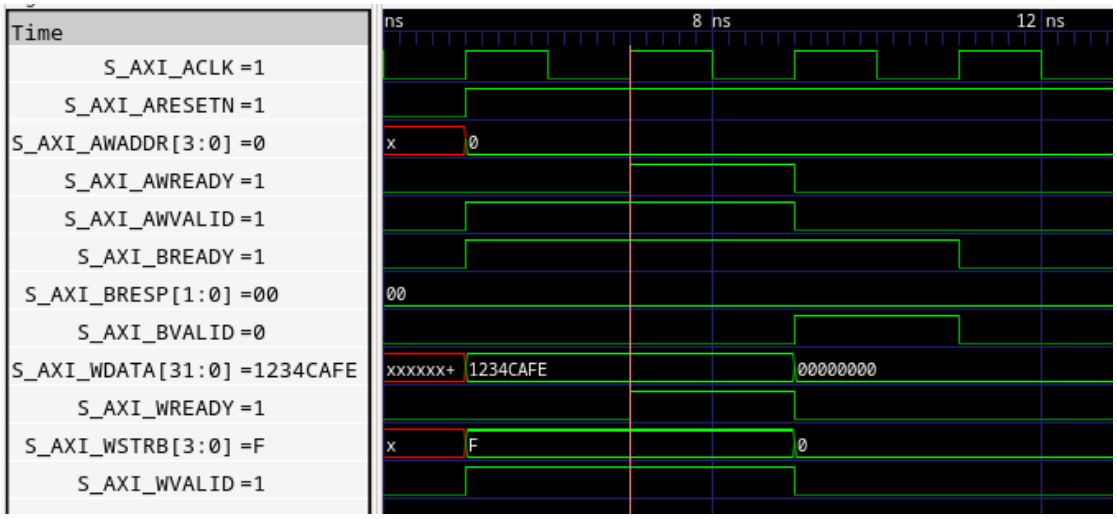
Zapis pełnej 32 bitowej liczby

Rysunek 1 przedstawia przebieg sygnałów podczas zapisu z wykorzystaniem opisanego modułu. Na przebiegu można zaobserwować, że *master* żądając zapisu wystawia adres, pod który chce zapisać na `S_AXI_AWADDR`, dane które chce zapisać na `S_AXI_WDATA` oraz oznacza, które bajty należy zapisać do rejestru ustawiając odpowiedni `S_AXI_WSTRB` (w tym przypadku 0xF oznacza binarne 1111, czyli wszystkie 4 bajty). Oprócz tego podnosi sygnały kontrolne: `S_AXI_AWVALID`, `S_AXI_WVALID` oraz `S_AXI_BREADY`, które oznaczają kolejno, że wystawiony adres jest prawidłowy, wystawione dane są prawidłowe oraz gotowość na przyjęcie odpowiedzi.



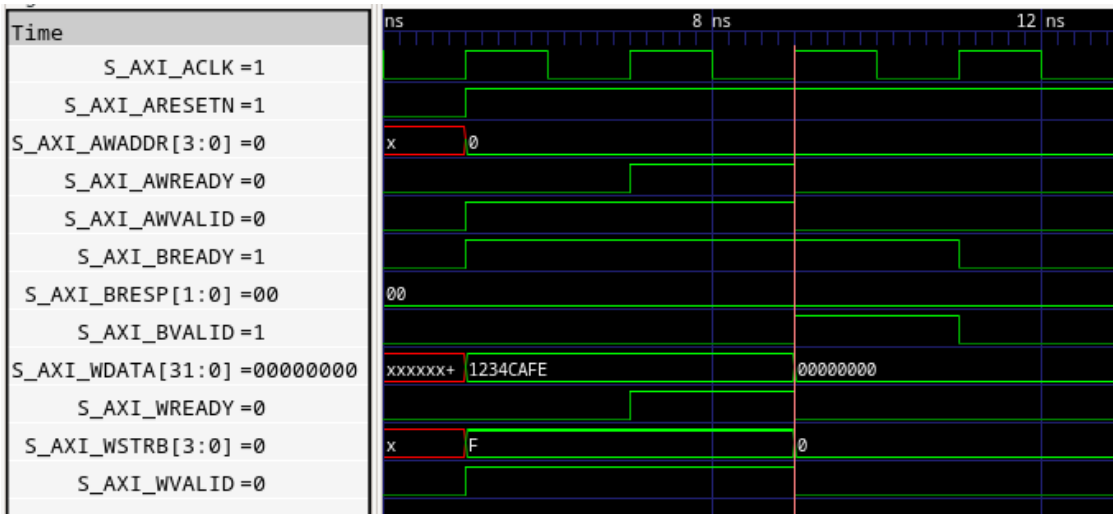
Rysunek 1: Przebieg sygnałów podczas operacji zapisu do rejestru `ctrl_reg` (pierwszy takt zegara).

Podczas drugiego taktu (przebieg na rysunku 2) *slave* podnosi sygnały `S_AXI_WREADY` oraz `S_AXI_AWREADY`, co oznacza gotowość do zapisu i pozwala *master'owi* opuścić sygnały `S_AXI_AWVALID` oraz `S_AXI_WVALID`, a także zmienić dane na `S_AXI_AWADDR`, `S_AXI_WDATA` oraz `S_AXI_WSTRB`.



Rysunek 2: Przebieg sygnałów podczas operacji zapisu do rejestru `ctrl_reg` (drugi takt zegara).

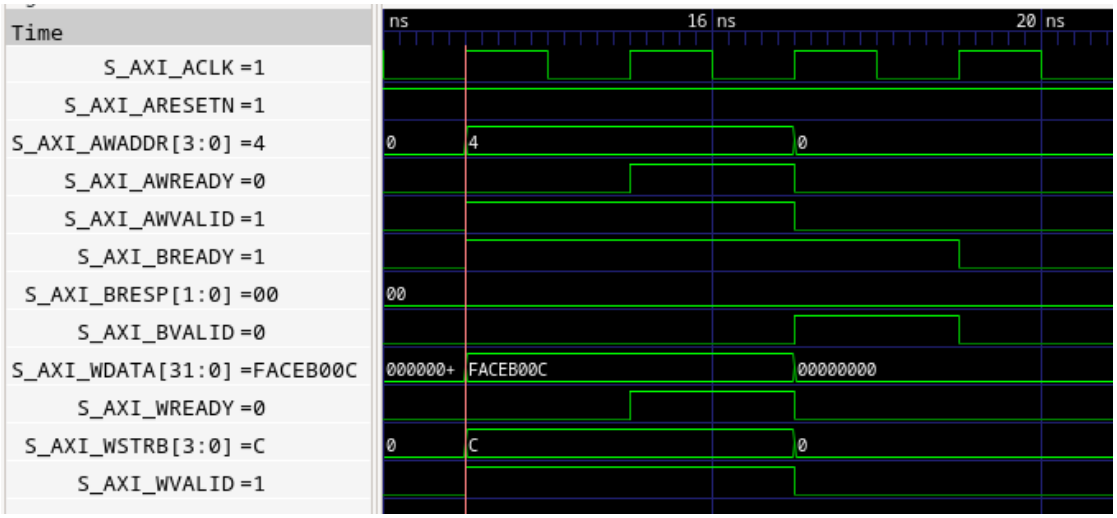
Trzeci takt zegara to wystawienie przez *slave'a* odpowiedzi na zapis na `S_AXI_BRESP` oraz potwierdzenie jej przez podniesienie `S_AXI_BVALID`. Sytuacja ta została przedstawiona na rysunku 3.



Rysunek 3: Przebieg sygnałów podczas operacji zapisu do rejestru `ctrl_reg` (trzeci takt zegara).

Zapis wybranych bajtów (wykorzystanie *strobe'a*)

Przebieg sygnałów I/O modułu nie różni się znacznie w przypadku zapisu z `S_AXI_WSTRB` innym niż 0xF. W przypadku przedstawionym na rysunku 4 jedyną różnicą jest `S_AXI_WSTRB` równy 0xC, oznacza to, że tylko 2 najstarsze bajty zostaną zapisane do rejestru pod adresem 4 (`in_angle_reg`). Z racji tego, że przed zapisem rejestr był wyzerowany, to zapisana w nim wartość po przedstawionej operacji wyniosła 0xFACE0000.

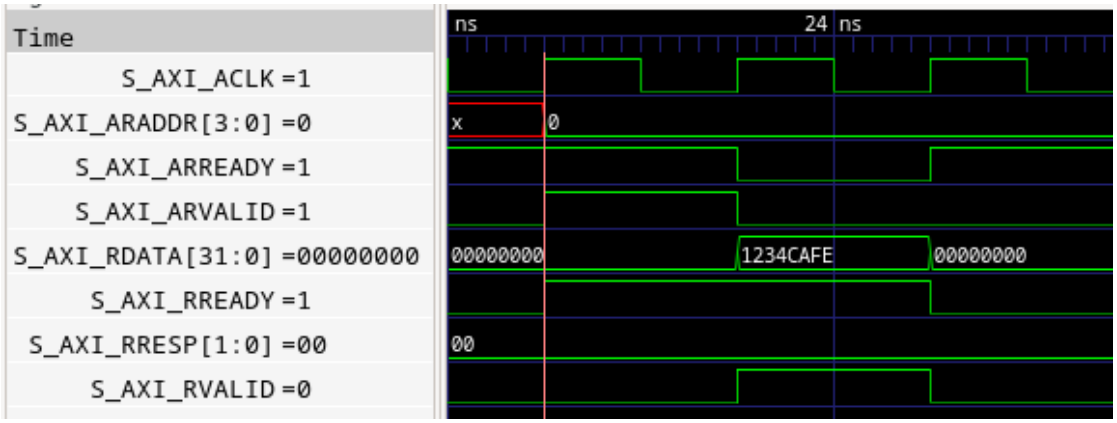


Rysunek 4: Zapis z wykorzystaniem `S_AXI_WSTRB` 0xC.

5.2 Operacja odczytu

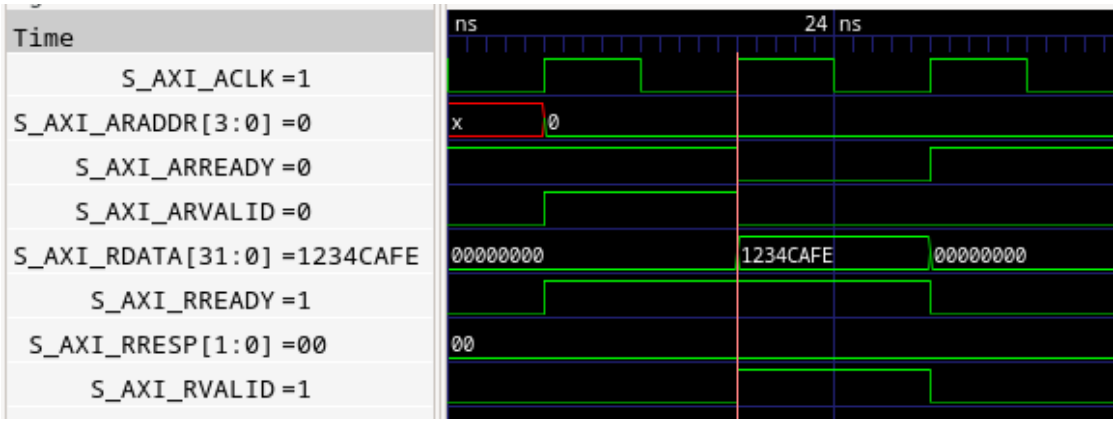
Odczyt spod adresu 0

Odczyt wartości rejestrów z wykorzystaniem AXI-Lite rozpoczyna się od wystawienia przez *master'a* adresu na `S_AXI_ARADDR` oraz podniesieniu sygnałów `S_AXI_ARVALID` i `S_AXI_ARREADY`.



Rysunek 5: Odczyt spod adresu 0 (pierwszy takt zegara).

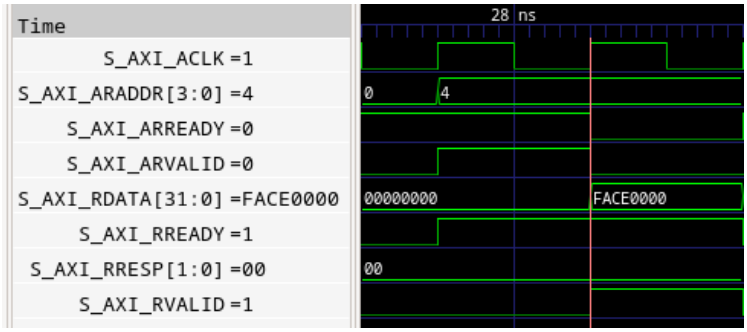
W następnym takcie *slave* wystawia wartość rejestru na `S_AXI_RDATA` i odpowiedź na `S_AXI_RRESP` oraz podnosi `S_AXI_RVALID`, a *master* opuszcza `S_AXI_ARVALID`. Sytuację tą przedstawia przebieg na rysunku 6. Odczytana wartość jest zgodna z oczekiwaniami.



Rysunek 6: Odczyt spod adresu 0 (drugi takt zegara).

Odczyt spod adresu 4

Rysunek 7 przedstawia przebiegi uzyskane w trakcie symulacji odczytu rejestru spod adresu 4. Zgodnie z oczekiwaniami otrzymana na `S_AXI_RDATA` liczba to 0xFACE0000.



Rysunek 7: Odczyt rejestru `in_angle_reg`.

6 Wnioski

W ramach laboratorium poszerzono wiedzę na temat działania protokołu AMBA AXI4-Lite. Z wykorzystaniem zdobytej wiedzy napisano prosty moduł *slave'a* korzystający z tego protokołu oraz zasymulowano jego działanie i przeanalizowano przebiegi sygnałów z symulacji w programie GTKWave.

Literatura

[1] ARM Limited. *AMBA AXI Protocol Specification Version 1.0*. ARM Limited, 2004. ARM IHI 0022B. URL: http://www.arm.com.

[2] Ph.D. Dan Gisselquist. Buidilng an axi-lite slave the easy way, Marzec 2020. URL: https://zipcpu.com/blog/2020/03/08/easyaxil.html.