Master Interface on Avalon Bus in VHDL

A data acquisition system is to be realized in a FPGA. A clock (Clk) synchronizes all the system. A specialized module receives data on an 8 bits bus named _DataAcquisition[7..0]_. A signal called _NewData_ specifies when a new data is received on this bus. This signal stay activated until acknowledged by the _DataAck_ signal activated by the module. This signal means that the data has been accepted and a new one can be received. Once _DataAck_ is activated, _NewData_ is deactivated at the next rising edge of the clock cycle. At the next clock, the _DataAck_ signal can be deactivated too.

And the cycle can start again.

With this mechanism, a DMA unit on the Avalon bus will take the data and copied it in memory. With every new data, the next memory position will be used until all the specified length of data is receive.

The _start address_ of write data in memory, the _length_ of the data transfer and the _enable_ of the transfers are seen as _registers_ in the programmable DMA controller to design.

When all the specified memory length is full, the start address is used again and the process continue until the processor send a stop command.

Problems to resolve

**Design the Master Programmable Interface specified.**

Treat the problem in the following manner:

1. Draw (properly) a block diagram of the entire processor-bus Avalon-programmable interface with the corresponding signals
2. Draw a timing diagram from acquisition until the Avalon Master transfer.
3. Propose a registers map of the interface. The registers can be reread
4. Describe the decomposition of the problem for the implementation of the interface.
5. Draw a symbol of the entity to achieve
6. Specify the module entity
7. Make the system architecture in VHDL
1. **Block diagram of the system in FPGA**

![Block diagram of the system in FPGA](image)

- **DataAcquisition[7..0]**
- **NewData**
- **DataAck**

**Avalon Bus**

- **Address, ByteEnable[3..0]**
- **DataRead**
- **DataWrite**

**Acquisition Interface**

- **Control (Read, Write, CS (Slave only))**
- **Control (WaitRequest, Master In)**

**With DMA**

- **(Master + Slave)**
2. Timing Diagramm

2 consecutive acquisitions cycles, with and without WaitRequest
Length of 7 transfers from address 0x100 000 → 0x100 006, and start again at 0x100000
3. Registers map

2 registers to realize the interface:

1. Address register: AcqAddress
2. Length register: AcqLength

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset Interface</th>
<th>Offset uP</th>
<th>Data width</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcqAddress</td>
<td>0</td>
<td>0</td>
<td>32 bits</td>
<td>Memory address where to start to put the data</td>
</tr>
<tr>
<td>AcqLength</td>
<td>1</td>
<td>4</td>
<td>32 bits</td>
<td>Length of the memory to stock the data</td>
</tr>
</tbody>
</table>

4. Problem decomposition

There are 3 principals units in the design:

1. Programmable interface module (interface registers)
   Interface with the Avalon bus in slave mode, receive the memory start address and length. It's possible to read them back.

2. Acquisition module
   Receive the data and acknowledge them

3. DMA Module, Avalon Master
   Make the transfers with the Avalon bus as a Master. Transfer the data byte by byte, initialize the start address at the end of the bloc transfer.

The parts 2 and 3 can be realized in the same process to simplify the VHDL code.

5. Symbol
6. Entity

Entity AcquModule is
Port(
    Clk : IN STD_LOGIC ;
    Reset_n : IN STD_LOGIC ;
    -- Acquisition
    DataAcquisition    : IN STD_LOGIC_VECTOR(7 downto 0) ;
    NewData    : IN STD_LOGIC ;
    DataAck    : OUT STD_LOGIC ;
    -- Avalon Slave :
    AS_Adr    : IN STD_LOGIC;
    AS_CS    : IN STD_LOGIC ;
    AS_Write    : IN STD_LOGIC ;
    AS_Read    : IN STD_LOGIC ;
    AS_DataWrite    : IN STD_LOGIC_VECTOR(31 downto 0) ;
    AS_DataRead    : OUT STD_LOGIC_VECTOR(31 downto 0) ;
    -- Avalon Master :
    AM_Adr    : OUT STD_LOGIC_VECTOR(31 downto 0) ;
    AM_WriteEnable    : OUT STD_LOGIC_VECTOR(3 downto 0) ;
    AM_Write    :  OUT STD_LOGIC ;
    AM_Read    :  OUT STD_LOGIC ;
    AM_DataWrite    : OUT STD_LOGIC_VECTOR(31 downto 0) ;
    AM_DataRead    : IN STD_LOGIC_VECTOR(31 downto 0) ;
    AM_WaitRequest    : IN STD_LOGIC
)
);
7. Architecture

Architecture Comp of AcquModule is
TYPE AcqState IS (Idle, WaitData, WriteData, AcqData);

Signal AcqAddress: STD_LOGIC_VECTOR(31 downto 0);
Signal AcqLength: STD_LOGIC_VECTOR(31 downto 0);
Signal CntAddress: STD_LOGIC_VECTOR(31 downto 0);
Signal CntLength: STD_LOGIC_VECTOR(31 downto 0);
Signal SM: AcqState;

Begin

-- Gestion des registres d'interface
pAvalon_Slave:
Process(Clk, Reset_n)
Begin
if Reset_n = '0' then
    AcqAddress <= (others => '0');
    AcqLength  <= (others => '0');
elsif rising_edge(Clk) then
    if AS_CS = '1' then
        if AS_Write = '1' then
            case AS_Adr is
                when '0' => AcqAddress <= AS_DataWrite;
                when '1' => AcqLength  <= AS_DataWrite;
                when others => null;
            end case;
        elsif AS_Read = '1' then
            case AS_Adr is
                when '0' => AS_DataRead <= AcqAddress;
                when '1' => AS_DataRead <= AcqLength;
                when others => null;
            end case;
        end if;
    end if;
End if;
End Process pAvalon_Slave;

-- Acquisition
pAcquisition:
Process(Clk, Reset_n)
Variable Indice : Integer Range 0 to 3;
Begin
if Reset_n = '0' then
    DataAck <= '0';
    SM <= Idle;
    AM_Write <= '0';
    AM_Read <= '0';
    AM_ByteEnable <= "0000";
    CntAddress <= (others => '0');
    CntLength  <= (others => '0');
elsif rising_edge(Clk) then
    if AcqLength /= X"0000_0000" then
        SM <= WaitData;
        CntAddress <= AcqAddress;
        CntLength <= AcqLength;
    end if;
if WaitData =>
    if NewData = '1' then
        SM <= WriteData;
        AM_Adr <= CntAddress;
        AM_Write <= '1';
        AM_DataWrite(7 downto 0)   <= DataAcquisition;
        AM_DataWrite(15 downto 8)  <= DataAcquisition;
        AM_DataWrite(23 downto 16) <= DataAcquisition;
    end if;
End if;
end if;
end if;
End Process pAcquisition;
AM_DataWrite(31 downto 24) <= DataAcquisition;
AM_ByteEnable <= "0000";
Indice := Conv_Integer(CntAddress(1 downto 0));
AM_ByteEnable(Indice) <= '1';
end if;
when WriteData =>
  if AM_WaitRequest = '0' then
    SM <= AcqData;
    AM_Write <= '0';
    AM_ByteEnable <= "0000";
    DataAck <= '1';
  end if;
when AcqData =>
  if NewData <= '0' then
    SM <= WaitData;
    DataAck <= '0';
    if CntLength /= 1 then
      CntAddress <= CntAddress + 1;
      CntLength <= CntLength - 1;
    else
      CntAddress <= AcqAddress;
      CntLength <= AcqLength;
    end if;
  end if;
end case;
end if;
End Process pAcquisition;

End Comp;
System schematic and test bench module.