Exemple d'un master accélérateur

sur bus Avalon

R.Beuchat

Système sur Bus Avalon

Un système embarqué est à réaliser sur FPGA et il est composé de:

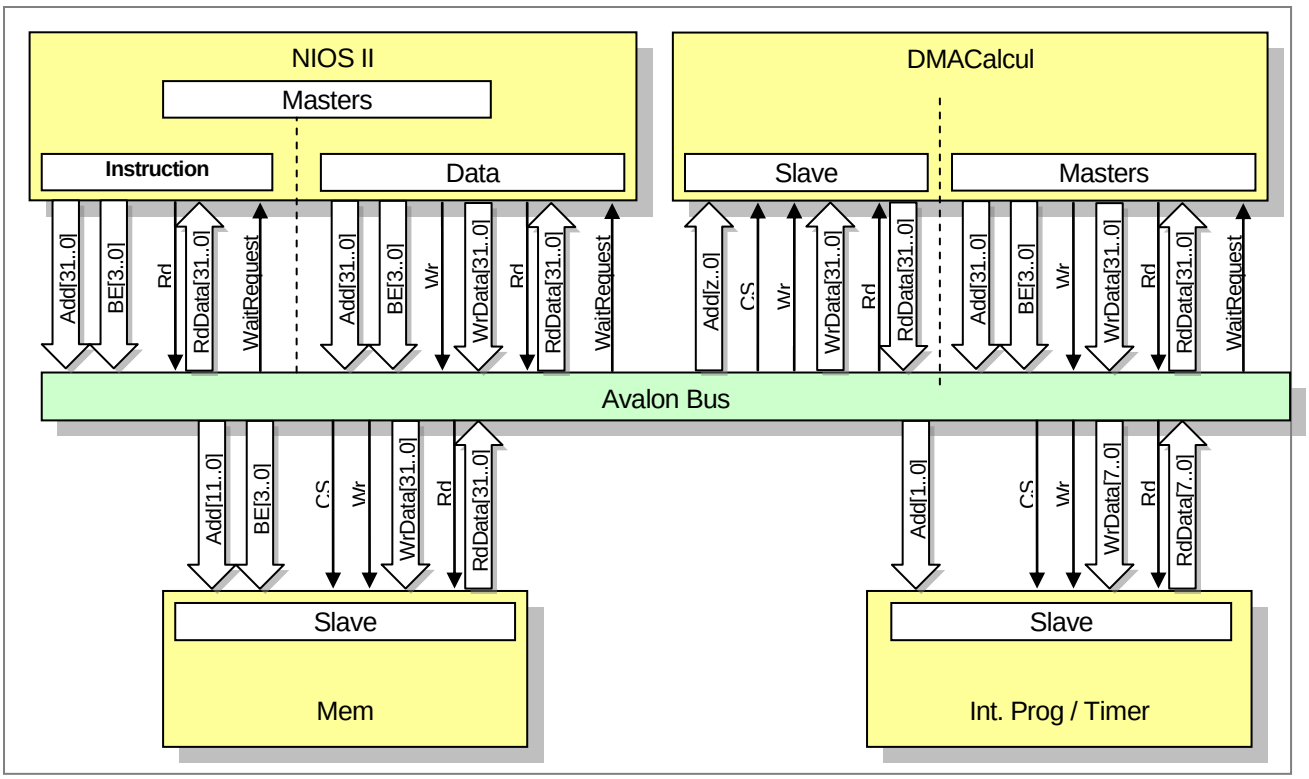
- un processeur NIOSII
- de la mémoire interne RAM de 16kBytes sur 32 bits de large
- une interface programmable Timer de 8 bits de large avec 4 registres internes. Connexion sur le bus Avalon, mode esclave model « *dynamic* ».
- Une unité **DMA accélératrice de calcul** permet de décharger le processeur de diverses opérations.
    - Cette unité est capable de lire des données sur 32 bits, d'effectuer des opérations sur les données lues et de mettre les résultats en mémoire à la suite des données lues en fin de traitement ;
    - Les données sont organisées, pour le programmeur en C, sous forme d'un tableau de type long ;
    - Le nombre d'éléments du tableau est plus petit que 65'535 éléments et doit être transmis à l'accélérateur, ainsi que l'adresse de début et la commande de départ du calcul ;
    - Un bit d'un registre de statut permet par scrutation de déterminer la fin des opérations pour récupérer les résultats.
    - Les 3 opérations suivantes sont à réaliser par le module accélérateur. Trouver les valeurs :
        - minimums (nombres signés !),
        - maximums (nombres signés !) et
        - le XOR des éléments du tableau.
    - Les données en mémoires sont des nombres entiers **signés** en complément à 2.
    - La programmation de cette unité est effectuée comme Avalon slave avec un bus de données de 32 bits de large

Traitez le problème de la façon suivante :

1. *4 pts* Dessinez un schéma bloc général de l'ensemble du système bus Avalon et modules. Indiquez clairement les signaux et leur taille entre les modules et le bloc du bus Avalon (Dessinez sur une feuille horizontalement).
2. *4 pts* **Le module accélérateur doit être développé en VHDL.** Déterminez les registres nécessaires du module accélérateur pour résoudre ce problème. Dessinez un plan de ces registres que vous documentez !
3. *4 pts* Dessinez un plan de la mémoire accessible par l'accélérateur et les registres associés du module accélérateur de calcul et faites en le lien.
4. *2 pts* Dessinez un schéma (symbole) du module accélérateur de calcul.
5. *2 pts* Définissez l'entité en VHDL de votre module accélérateur de calcul.
6. *6 pts* Réalisez une architecture en VHDL du module accélérateur de calcul ainsi conçu.
7. *4 pts* Ecrivez la fonction en C permettant de **lancer le calcul** et **d'attendre le résultat par scrutation**. La routine reçoit comme paramètres l'adresse du buffer et sa taille.

Réponses :

1. *4 pts* Dessinez un schéma bloc général de l'ensemble du système bus Avalon et modules. Indiquez clairement les signaux entre les modules et le bloc du bus Avalon.



Mémoire 16 kBytes --> 4 kMots de 32 bits (Quadlet) --> 2^{12} Quadlets --> 12 lignes d'adresse --> Adr[11..0]

Rappel: l'adresse côté esclave est une adresse Mot

Timer 4 registres --> 2 lignes d'adresses

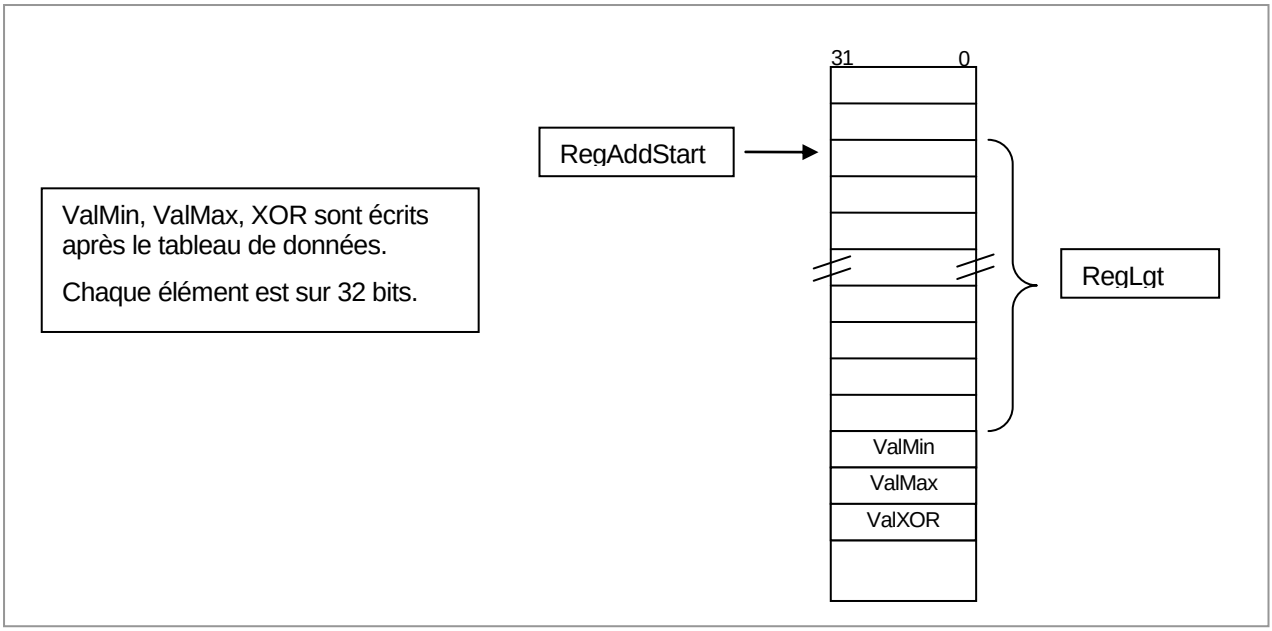
2. *4 pts* Le module DMA doit être développé en VHDL. Déterminez les registres nécessaires pour résoudre ce problème. Dessinez un plan de ces registres du module DMA que vous documentez !

Il faut :

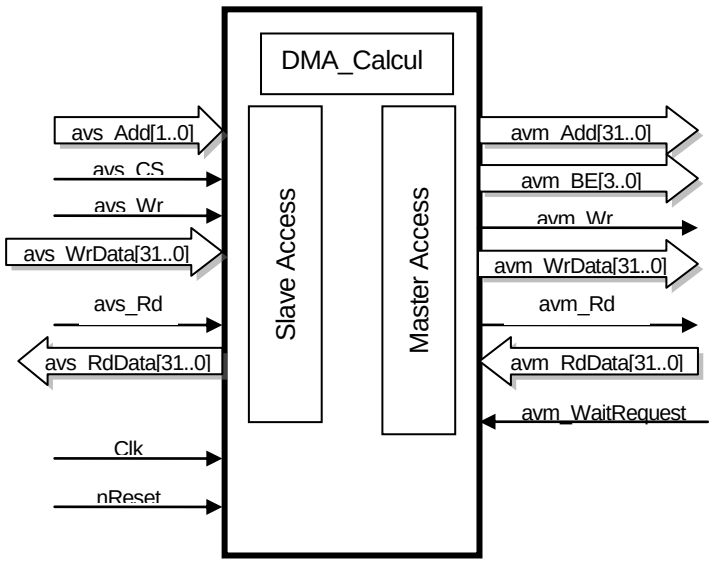
- un registre pour spécifier l'adresse en mémoire du tableau
- un registre pour sa longueur en nombre d'éléments
- un bit de commande pour démarrer le calcul
- un bit de statuts pour savoir quand le calcul est terminé

Adresses **Vue depuis DMA**	**Registres**	**Taille**	**Fonction**
0	RegAddStart	32	Adresse de début tableau
1	RegLgt	16	Nombre d'éléments dans le tableau 0..65535
2	RegCommand	8	Commande, bit0 : Start
3	RegStatus	8	Status, bit0 : Finish

3. *4 pts* Dessinez un plan de la mémoire et les registres associés du module DMA de calcul.



4. *4 pts* Dessinez un schéma (symbole) du module DMA de calcul.



5. *4 pts* Définissez l'entité de votre module DMA de calcul

```
entity DMA_Calcul is
port (
    clk:            in  std_logic;
    nReset:     in  std_logic;
-- Slave part
    avs_Add:        in  std_logic_vector(1 downto 0);
    avs_CS:     in  std_logic;
    avs_Wr:     in  std_logic;
    avs_Rd:     in  std_logic;
    avs_WrData: in  std_logic_vector(31 downto 0);
    avs_RdData: out std_logic_vector(31 downto 0);
-- Master part
    avm_Add:        out std_logic_vector(31 downto 0);
    avm_BE:     out std_logic_vector(3 downto 0);
    avm_Wr:     out std_logic;
    avm_Rd:     out std_logic;
    avm_WrData: out std_logic_vector(31 downto 0);
    avm_RdData: in  std_logic_vector(31 downto 0);
    avm_WaitRequest:    in  std_logic
);
end entity DMA_Calcul;
```

6. *6 pts* Réalisez une architecture du DMA de calcul ainsi conçu.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
architecture behv of DMA_Calcul is
-- internal registers
signal  RegAddStart:    std_logic_vector(31 downto 0);
signal  RegLgt:         std_logic_vector(15 downto 0);      -- Max 65535 data
signal  DataRd:         signed(31 downto 0);                -- Data Read from memory to use for calculus
signal  Start, Finish:  std_logic;                          -- Command, Status bits
signal  CntAdd:         unsigned(31 downto 0);              -- Address in memory
signal  CntLgt:         unsigned (15 downto 0);             -- Number of element to read
signal  ValMin, ValMax: signed(31 downto 0);                -- Result Min, Max
signal  ValXOR:         std_logic_vector (31 downto 0);     -- Result XOR

-- Machine d'états pour les accès DMA:
type SM is(Idle, LdParam, RdAcc, WaitRd, Calcul, WrMin, WrMax, WrXOR, WrEnd);

signal StateM: SM;

begin

-- Slave Wr Access, Programmation des registres de l'interface Avalon Slave
AvalonSlaveWr:
process(clk, nReset)
begin
    if nReset = '0' then
        RegAddStart <= (others => '0');
        RegLgt      <= (others => '0');
        Start       <= '0';
    elsif rising_edge(clk) then
        Start <= '0';
        if avs_CS = '1' and avs_Wr = '1' then
            case avs_Add is
                when "00" => RegAddStart    <= avs_WrData ;
                when "01" => RegLgt         <= avs_WrData(15 downto 0);
                when "10" => Start          <= avs_WrData(0);
                when others => null;
            end case;
        end if;
    end if;
end process AvalonSlaveWr;
```

```
-- Slave Rd Access, mettre 1 cycle d'attente car lecture synchrone
AvalonSlaveRd:
process(clk)
begin
    if rising_edge(clk) then
        if avs_CS = '1' and avs_Rd = '1' then
            avs_RdData <= (others => '0');      -- etat par default
            case avs_Add is
                when "00" => avs_RdData                <= RegAddStart ;
                when "01" => avs_RdData(15 downto 0)   <= RegLgt;
                when "10" => avs_RdData(0)             <= Start;
                when "11" => avs_RdData(0)             <= Finish;
                when others => null;
            end case;
        end if;
    end if;
end process AvalonSlaveRd;

-------------------------------------------------------------------------------------------
-- DMA Access
-- Ce séquenceur n'est pas optimisé,
-- le calcul pourrait être effectué lors de la lecture des données directement
-- Boucle pour lire toutes les données
-- Ecrit les données calculées après les calculs

AvalonMaster:
process(clk, nReset)
begin
    if nReset = '0' then
        Finish <= '0';
        CntAdd <= (others => '0');
        CntLgt <= (others => '0');
        ValMin <= to_signed(integer'high,32);  --X"7F FF FF FF";   -- Maximum possible en cpl à 2
        ValMax <= to_signed(integer'low,32);   --X"80 00 00 00";   -- Minimum possible en cpl'à 2
        ValXOR <= (others => '0');             -- Resultat initial pour XOR
        StateM <= Idle;
    elsif rising_edge(clk) then
        case StateM is
            when Idle =>                            -- Wait Commande Start
                    avm_Add <= (others => '0');     -- Init Avalon Master au repos
                    avm_BE  <= "0000";
                    avm_Wr  <= '0';
                    avm_Rd  <= '0';

                    if Start = '1' then
                        StateM <= LdParam;          -- Démarre la machine
                        Finish <= '0';
                    end if;
            when LdParam =>                         -- Charge paramètres de transferts
                    CntAdd <= unsigned(RegAddStart);    -- Unsigned pour calcul
                    CntLgt <= unsigned(RegLgt);
                    ValMin <= to_signed(integer'high, 32);  -- Au départ ValMin est la max possible
                    ValMax <= to_signed(integer'low, 32);   -- Au départ ValMax est la plus petite poss.
                    ValXOR <= (others => '0');
                    StateM <= RdAcc;
            when RdAcc =>                           -- Commence cycle Master de lecture
                    avm_Add <= std_logic_vector(CntAdd);
                    avm_BE <= "1111";
                    avm_Rd <= '1';
                    StateM <= WaitRd;
            when WaitRd =>                          -- Attend la donnée lue
                    if avm_WaitRequest = '0' then
                        DataRd <= signed(avm_RdData);   -- Lit la donnée et passe en signé
                        avm_Rd <= '0';
                        StateM <= Calcul;
                    end if;
            when Calcul =>                          -- Calcul
                    if DataRd < ValMin then         -- test signé
                        ValMin <= DataRd;           -- Nouveau Min
                    end if;
                    if DataRd > ValMax then     -- test signé
                        ValMax <= DataRd;           -- Nouveau Max
```

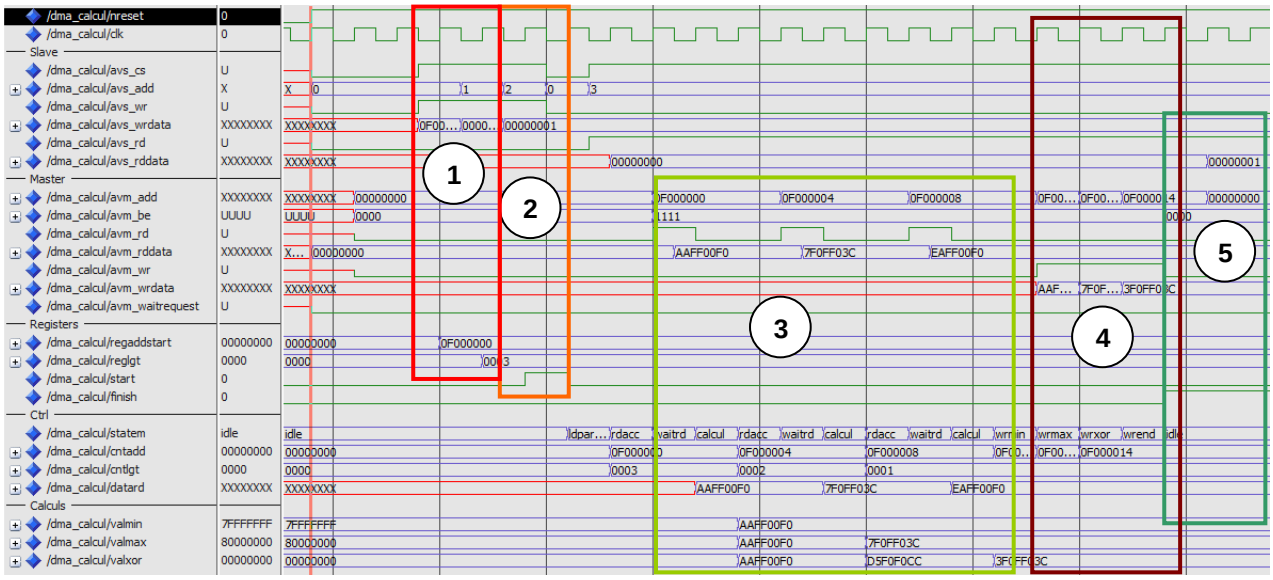
```
                    end if;

                    ValXOR <= ValXOR XOR std_logic_vector(DataRd);  -- Calcul XOR

                    CntAdd <= CntAdd + 4;       -- Pointe adresse suivante
                    if CntLgt > 1 then          -- Reste à lire ?
                        StateM <= RdAcc;            -- continue la lecture des données
                        CntLgt <= CntLgt -1;        -- et une de moins
                    else
                        StateM <= WrMin;            -- Si fini: Commence l'écriture en mémoire
                    end if;
            when WrMin =>                           -- Wr Min value after table
                    avm_Add <= std_logic_vector(CntAdd);    -- Pnte en mémoire après le tableau
                    avm_BE <= "1111";                   -- Sure 32 bits
                    avm_Wr <= '1';                      -- Ecrit
                    avm_WrData <= std_logic_vector(ValMin);
                    CntAdd <= CntAdd + 4;           -- Prépare adresse suivante
                    StateM <= WrMax;
            when WrMax =>                           -- Wr Max value after Min
                    if avm_WaitRequest = '0' then   -- Wait end of previous Wr
                        avm_Add <= std_logic_vector(CntAdd);-- OK procahine donnée
                        avm_BE <= "1111";
                        avm_Wr <= '1';
                        avm_WrData <= std_logic_vector(ValMax);
                        CntAdd <= CntAdd + 4;
                        StateM <= WrXOR;
                    end if;
            when WrXOR =>                           -- Wr XOR  part
                    if avm_WaitRequest = '0' then   -- Wait end of previous Wr
                        avm_Add <= std_logic_vector(CntAdd);
                        avm_BE <= "1111";
                        avm_Wr <= '1';
                        avm_WrData <= ValXOR;       -- Ecrit ValXOR
                        StateM <= WrEnd;
                    end if;
            when WrEnd =>
                    if avm_WaitRequest = '0' then   -- Wait end of previous Wr
                        avm_BE <= "0000";
                        avm_Wr <= '0';
                        Finish <= '1';              -- Active bit de status de fin
                        StateM <= Idle;             -- Prêt à recommencer
                    end if;

            when others => null;
        end case;
    end if;
end process AvalonMaster;

end architecture behv;
```



Une simulation de programmation des paramètres du module DMA :

1) *Programmation des registres Adresse, Longueur*
2) *Ecrit la Commande Start*
3) *Transferts par le DMA et calcul*
4) *Ecriture des 3 résultats, Min, Max, XOR, pas d'attente entre les cycles*
5) *Fin des opérations*

7. *4 pts* Ecrivez la fonction en C permettant de **lancer le calcul** et **d'attendre le résultat par scrutation**. La routine reçoit comme paramètres l'adresse du buffer et sa taille.

utilise les macro de io.h

```
void AccelCalcul(unsigned long *PntBuf, unsigned short LgBuf){
    // Ecrit l'adresse de base de la zone à calculer
    IOWR_32DIRECT(BASE_DMACalcul, 0*4, PntBuf);
    // Ecrit la longueur
    IOWR_32DIRECT(BASE_DMACalcul, 1*4, LgBuf);
    // Start calcul
    IOWR_32DIRECT(BASE_DMACalcul, 2*4, 1);
    // Attend la fin du calcul par scrutation, test bit 0 du statut, attend si == 0
    while( (IORD_32DIRECT(BASE_DMACalcul, 3*4) & (1<<0)) == 0){ }
}
```

*Avec **BASE_DMACalcul** l'adresse du contrôleur DMA réalisé.*