TELECOM Paris

IP PARIS

# Interfaces SystemVerilog, bus et protocoles. . .

## Techniques de codage, Etude d'une norme

Yves Mathieu

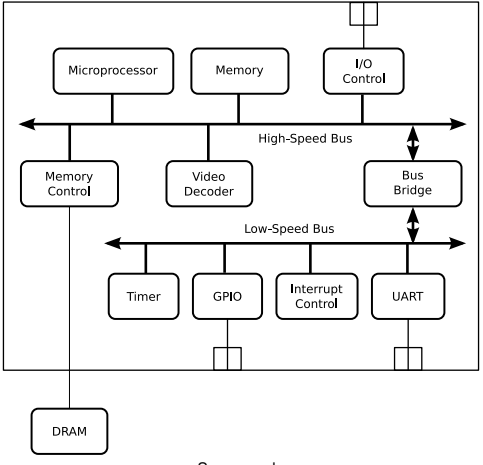
Institut Mines-Télécom

# Plan

# Introduction (1)

## Interconnection de blocs

Utilisation de bus de communication génériques




Source: edn

# Introduction (2)

## Rôle des normes de bus

- Unifier les interfaces entre blocs:
  - Faciliter le design des blocs
  - Faciliter le test des blocs (programmes de tests génériques)
  - Un système = un lego de blocs
  - Permettre la création d'outils de génération "Système" (Qsys...)
- Organiser l'arbitrage
- Quelle norme de bus utiliser ?
  - De nombreuses normes anciennes ou récentes (VCI, Ocp, Amba AHB, Amba APB, Amba AXI, Avalon, ...)
  - Pour nous Avalon, norme propriétaire (Altera), simple...

# Introduction (2)

## Rôle des normes de bus

- Unifier les interfaces entre blocs:
  - Faciliter le design des blocs
  - Faciliter le test des blocs (programmes de tests génériques)
  - Un système = un lego de blocs
  - Permettre la création d'outils de génération "Système" (Qsys...)
- Organiser l'arbitrage
- Quelle norme de bus utiliser ?
  - De nombreuses normes anciennes ou récentes (VCI, Ocp, Amba AHB, Amba APB, Amba AXI, Avalon, ...)
  - Pour nous Avalon, norme propriétaire (Altera), simple...

# Introduction (2)

## Rôle des normes de bus

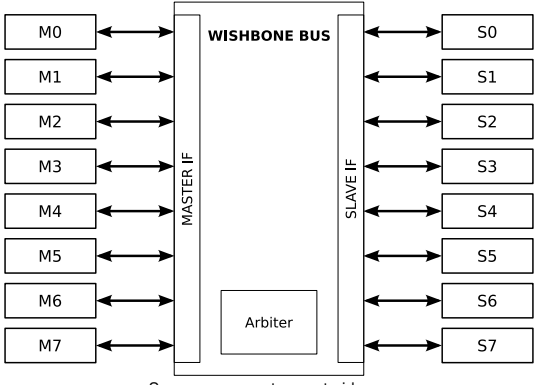
- Unifier les interfaces entre blocs:
  - Faciliter le design des blocs
  - Faciliter le test des blocs (programmes de tests génériques)
  - Un système = un lego de blocs
  - Permettre la création d'outils de génération "Système" (Qsys...)
- Organiser l'arbitrage
- Quelle norme de bus utiliser ?
  - De nombreuses normes anciennes ou récentes (VCI, Ocp, Amba AHB, Amba APB, Amba AXI, Avalon, ...)
  - Pour nous Avalon, norme propriétaire (Altera), simple...

# Introduction (3)

## Point de vue du concepteur

L'interconnection est un bloc de logique à part entière, qui contient au minimum:

- De la logique de multiplexage pour la connection du contrôle et des données
- De la logique d'arbitrage pour piloter la logique de multiplexage




Source: www.systemcentroid.com

**Attention** : la norme ne décrit que l'interconnection point à point.

# Introduction (4)

## Le problème...

- Beaucoup de déclarations à répéter dans les entêtes de modules.
- Beaucoup de code de protocole de communication a dupliquer dans les modules
- Des erreurs potentielles : tailles, directions des E/S
- Rapidement des centaines de signaux à déclarer dans les interconnections
- On aimerait
  - Regrouper les signaux par ensembles cohérents (bus PCI, bus USB,...)
  - Ne pas se soucier du sens des signaux dans l'interconnection
  - Définir de façon unique ces blocs d'interconnection
  - Propager les changements de définitions

# Introduction (4)

## Le problème...

- Beaucoup de déclarations à répéter dans les entêtes de modules.
- Beaucoup de code de protocole de communication a dupliquer dans les modules
- Des erreurs potentielles : tailles, directions des E/S
- Rapidement des centaines de signaux à déclarer dans les interconnections
- On aimerait
  - Regrouper les signaux par ensembles cohérents (bus PCI, bus USB,...)
  - Ne pas se soucier du sens des signaux dans l'interconnection
  - Définir de façon unique ces blocs d'interconnection
  - Propager les changements de définitions

# Introduction (4)

## Le problème...

- Beaucoup de déclarations à répéter dans les entêtes de modules.
- Beaucoup de code de protocole de communication a dupliquer dans les modules
- Des erreurs potentielles : tailles, directions des E/S
- Rapidement des centaines de signaux à déclarer dans les interconnections
- On aimerait
  - Regrouper les signaux par ensembles cohérents (bus PCI, bus USB,...)
  - Ne pas se soucier du sens des signaux dans l'interconnection
  - Définir de façon unique ces blocs d'interconnection
  - Propager les changements de définitions

# Introduction (4)

## Le problème...

- Beaucoup de déclarations à répéter dans les entêtes de modules.
- Beaucoup de code de protocole de communication a dupliquer dans les modules
- Des erreurs potentielles : tailles, directions des E/S
- Rapidement des centaines de signaux à déclarer dans les interconnections
- On aimerait
  - Regrouper les signaux par ensembles cohérents (bus PCI, bus USB,...)
  - Ne pas se soucier du sens des signaux dans l'interconnection
  - Définir de façon unique ces blocs d'interconnection
  - Propager les changements de définitions

# Introduction (4)

## Le problème...

- Beaucoup de déclarations à répéter dans les entêtes de modules.
- Beaucoup de code de protocole de communication a dupliquer dans les modules
- Des erreurs potentielles : tailles, directions des E/S
- Rapidement des centaines de signaux à déclarer dans les interconnections
- On aimerait
  - Regrouper les signaux par ensembles cohérents (bus PCI, bus USB,...)
  - Ne pas se soucier du sens des signaux dans l'interconnection
  - Définir de façon unique ces blocs d'interconnection
  - Propager les changements de définitions

# Plan

# L’ interface systemVerilog

- Regrouper des signaux et représenter par un port unique.
- Ne déclarer les signaux internes qu’un seule fois.
- Un module connecté à une interface n’est connecté qu’à un seul port.
- Des **function** et **task** peuvent être déclarées dans une interface:
  - Pas de duplication de codes identiques dans les modules.
  - Inclure du code de vérification dans la définition de l’interface

# L’ interface systemVerilog

- Regrouper des signaux et représenter par un port unique.
- Ne déclarer les signaux internes qu’un seule fois.
- Un module connecté à une interface n’est connecté qu’à un seul port.
- Des **function** et **task** peuvent être déclarées dans une interface:
  - Pas de duplication de codes identiques dans les modules.
  - Inclure du code de vérification dans la définition de l’interface

# L’ interface systemVerilog

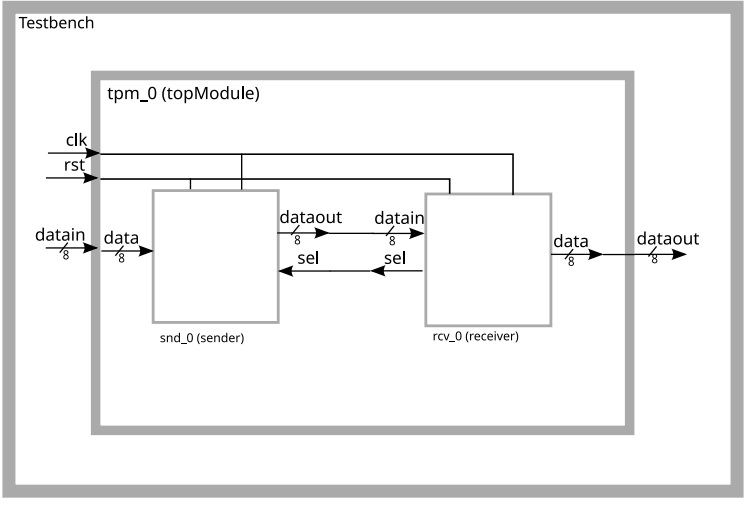
- Regrouper des signaux et représenter par un port unique.
- Ne déclarer les signaux internes qu’un seule fois.
- Un module connecté à une interface n’est connecté qu’à un seul port.
- Des **function** et **task** peuvent être déclarées dans une interface:
  - Pas de duplication de codes identiques dans les modules.
  - Inclure du code de vérification dans la définition de l’interface

# L’ interface systemVerilog

- Regrouper des signaux et représenter par un port unique.
- Ne déclarer les signaux internes qu’un seule fois.
- Un module connecté à une interface n’est connecté qu’à un seul port.
- Des **function** et **task** peuvent être déclarées dans une interface:
  - Pas de duplication de codes identiques dans les modules.
  - Inclure du code de vérification dans la définition de l’interface

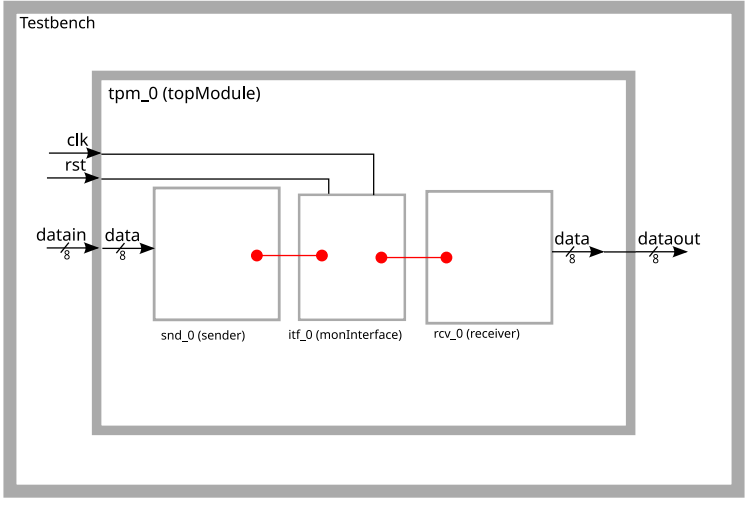
# L' interface systemVerilog

## Exemple sans interface



# L' interface systemVerilog

## Exemple avec interface



# L' interface systemVerilog

- Element hiérarchique au même titre qu'un module, mais...
  - Une **interface** ne peut contenir de **module** ou de primitives.
  - Une **interface** peut être utilisée comme port d'un **module**.
  - Une **interface** peut contenir des alternatives (**modport**).

# L' interface systemVerilog

## La déclaration

- Définition de signaux globaux (clk, rst)
- Définition de signaux propres à l'interface (Data, Sel)
- Définition de vues différentes de l'interface (modport)
- **Attention** : Une interface peut être paramétrée (comme un module)

```
interface monInterface (
    input logic clk, rst
  );

  logic [7:0] Data ;
  logic Sel ;

  modport master (
    input clk, rst, Data,
    output Sel
  );

  modport slave (
    input clk, rst, Sel,
    output Data
  );

endinterface
```

# L' interface systemVerilog

## La déclaration

- Définition de signaux globaux (clk, rst)
- Définition de signaux propres à l'interface (Data, Sel)
- Définition de vues différentes de l'interface (modport)
- **Attention** : Une interface peut être paramétrée (comme un module)

```
interface monInterface (
    input logic clk, rst
  );

  logic [7:0] Data ;
  logic Sel ;

  modport master (
    input clk, rst, Data,
    output Sel
  );

  modport slave (
    input clk, rst, Sel,
    output Data
  );

endinterface
```

# L' interface systemVerilog

## La déclaration

- Définition de signaux globaux (clk, rst)
- Définition de signaux propres à l'interface (Data, Sel)
- Définition de vues différentes de l'interface (modport)
- **Attention** : Une interface peut être paramétrée (comme un module)

```
interface monInterface (
    input logic clk, rst
  );

  logic [7:0] Data ;
  logic Sel ;

  modport master (
    input clk, rst, Data,
    output Sel
  );

  modport slave (
    input clk, rst, Sel,
    output Data
  );

endinterface
```

# L' interface systemVerilog

## La déclaration

- Définition de signaux globaux (clk, rst)
- Définition de signaux propres à l'interface (Data, Sel)
- Définition de vues différentes de l'interface (modport)
- **Attention** : Une interface peut être paramétrée (comme un module)

```
interface monInterface (
    input logic clk, rst
  );

  logic [7:0] Data ;
  logic Sel ;

  modport master (
    input clk, rst, Data,
    output Sel
  );

  modport slave (
    input clk, rst, Sel,
    output Data
  );

endinterface
```

# L' interface systemVerilog

## L'instanciation

- On instancie une interface comme un module.
- Les signaux globaux **clk** et **rst** seront accessibles via l'interface
- Les autres signaux de l'interface sont internes à l'interface

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout) ;

  monInterface itf_0(
       .clk(clk),
       .rst(rst)
        ) ;

  // Suite de la description

endmodule
```

# L' interface systemVerilog

## L'instanciation

- On instancie une interface comme un module.
- Les signaux globaux **clk** et **rst** seront accessibles via l'interface
- Les autres signaux de l'interface sont internes à l'interface

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout) ;

  monInterface itf_0(
       .clk(clk),
       .rst(rst)
        ) ;

  // Suite de la description

endmodule
```

# L' interface systemVerilog

## L'instanciation

- On instancie une interface comme un module.
- Les signaux globaux **clk** et **rst** seront accessibles via l'interface
- Les autres signaux de l'interface sont internes à l'interface

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout) ;

  monInterface itf_0(
       .clk(clk),
       .rst(rst)
        ) ;

  // Suite de la description

endmodule
```

# L' interface systemVerilog

## Accès à une interface via le port d'un module

- Le port **if_mst** est un modport **master** d'une interface de type **monInterface**.
- On aurait pu juste indiquer **monInterface if_mst**.
- Ou **interface if_mst**
- Choix du degré de vérification à la compilation
- On accède aux signaux de l'interface par un nom hiérarchique.

```
module receiver(
  monInterface.master if_mst,
  output logic[7:0] data
);

  logic [1:0] cmpt ;

  always_ff @(posedge if_mst.clk
      or posedge if_mst.rst)
  if(if_mst.rst)
    cmpt <= '0 ;
  else
    cmpt <= cmpt+1 ;

  assign if_mst.Sel = cmpt[1] ;
  assign data = if_mst.Data ;

endmodule
```

# L' interface systemVerilog

## Accès à une interface via le port d'un module

- Le port **if_mst** est un modport **master** d'une interface de type **monInterface**.
- On aurait pu juste indiquer **monInterface if_mst**.
- Ou **interface if_mst**
- Choix du degré de vérification à la compilation
- On accède aux signaux de l'interface par un nom hiérarchique.

```
module receiver(
  monInterface.master if_mst,
  output logic[7:0] data
);

  logic [1:0] cmpt ;

  always_ff @(posedge if_mst.clk
      or posedge if_mst.rst)
  if(if_mst.rst)
    cmpt <= '0 ;
  else
    cmpt <= cmpt+1 ;

  assign if_mst.Sel = cmpt[1] ;
  assign data = if_mst.Data ;

endmodule
```

# L' interface systemVerilog

## Accès à une interface via le port d'un module

- Le port **if_mst** est un modport **master** d'une interface de type **monInterface**.
- On aurait pu juste indiquer **monInterface if_mst**.
- Ou **interface if_mst**
- Choix du degré de vérification à la compilation
- On accède aux signaux de l'interface par un nom hiérarchique.

```
module receiver(
  monInterface.master if_mst,
  output logic[7:0] data
);

  logic [1:0] cmpt ;

  always_ff @(posedge if_mst.clk
      or posedge if_mst.rst)
  if(if_mst.rst)
    cmpt <= '0 ;
  else
    cmpt <= cmpt+1 ;

  assign if_mst.Sel = cmpt[1] ;
  assign data = if_mst.Data ;

endmodule
```

# L' interface systemVerilog

## Accès à une interface via le port d'un module

- Le port **if_mst** est un modport **master** d'une interface de type **monInterface**.
- On aurait pu juste indiquer **monInterface if_mst**.
- Ou **interface if_mst**
- Choix du degré de vérification à la compilation
- On accède aux signaux de l'interface par un nom hiérarchique.

```
module receiver(
  monInterface.master if_mst,
  output logic[7:0] data
);

  logic [1:0] cmpt ;

  always_ff @(posedge if_mst.clk
      or posedge if_mst.rst)
  if(if_mst.rst)
    cmpt <= '0 ;
  else
    cmpt <= cmpt+1 ;

  assign if_mst.Sel = cmpt[1] ;
  assign data = if_mst.Data ;

endmodule
```

# L' interface systemVerilog

## Accès à une interface via le port d'un module

- Le port **if_mst** est un modport **master** d'une interface de type **monInterface**.
- On aurait pu juste indiquer **monInterface if_mst**.
- Ou **interface if_mst**
- Choix du degré de vérification à la compilation
- On accède aux signaux de l'interface par un nom hiérarchique.

```
module receiver(
  monInterface.master if_mst,
  output logic[7:0] data
);

  logic [1:0] cmpt ;

  always_ff @(posedge if_mst.clk
      or posedge if_mst.rst)
  if(if_mst.rst)
    cmpt <= '0 ;
  else
    cmpt <= cmpt+1 ;

  assign if_mst.Sel = cmpt[1] ;
  assign data = if_mst.Data ;

endmodule
```

# L' interface systemVerilog

## Utilisation d'une interface comme port d'un module

```
module sender(
  monInterface.slave if_slv,
  input logic[7:0] data
);

 always_ff @(posedge if_slv.clk)
   if (if_slv.Sel)
     if_slv.Data <= data ;
   else
     if_slv.Data <= '0  ;

endmodule
```

# L' interface systemVerilog

## Instanciation et interconnection

- On instancie les deux modules **sender** et **receiver**.
- On connecte les deux modules à l'interface **itf_0**
- On connecte les autres signaux des modules.

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout
);
  monInterface itf_0(
        .clk(clk),
        .rst(rst)
        ) ;
  sender  snd_0(
             .if_slv(itf_0.slave),
             .data(datain)
            ) ;
  receiver rcv_0(
           .if_mst(itf_0.master),
           .data(dataout)
           ) ;
endmodule
```

# L' interface systemVerilog

## Instanciation et interconnection

- On instancie les deux modules **sender** et **receiver**.
- On connecte les deux modules à l'interface **itf_0**
- On connecte les autres signaux des modules.

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout
);
  monInterface itf_0(
        .clk(clk),
        .rst(rst)
        ) ;
  sender  snd_0(
             .if_slv(itf_0.slave),
             .data(datain)
            ) ;
  receiver rcv_0(
           .if_mst(itf_0.master),
           .data(dataout)
           ) ;
endmodule
```

# L' interface systemVerilog

## Instanciation et interconnection

- On instancie les deux modules **sender** et **receiver**.
- On connecte les deux modules à l'interface **itf_0**
- On connecte les autres signaux des modules.

```
module topModule(
  input logic clk,
  input logic rst,
  input logic [7:0] datain,
  output logic [7:0] dataout
);
  monInterface itf_0(
        .clk(clk),
        .rst(rst)
        ) ;
  sender  snd_0(
             .if_slv(itf_0.slave),
             .data(datain)
            ) ;
  receiver rcv_0(
           .if_mst(itf_0.master),
           .data(dataout)
           ) ;
endmodule
```

# Plan

# Avalon

## Caractéristiques

- Protocole propriétaire: Développé par la société Altera (Intel) pour les "System On Chip" sur FPGA.
- Pas d'usage autre que ce contexte, à la différence des protocoles normalisés qui permettent l'interopérabilité.
- Transaction **synchrone** de type **accès mémoire**: c'est à dire un **hôte** émettant des requêtes de lecture ou d'écriture vers un **agent**, à une adresse donnée.
- Différentes variantes du protocole et de paramêtrage pour optimiser les transferts et le matériel nécessaire.
- Dans cadre de ECE_4SEO4, les variantes du protocole utilisées seront:
  - **Pipelined Transfer with Variable Latency**
  - **Pipelined Transfer with Variable Latency and Burst**

# Avalon

## Caractéristiques

- Protocole propriétaire: Développé par la société Altera (Intel) pour les "System On Chip" sur FPGA.
- Pas d'usage autre que ce contexte, à la différence des protocoles normalisés qui permettent l'interopérabilité.
- Transaction **synchrone** de type **accès mémoire**: c'est à dire un **hôte** émettant des requêtes de lecture ou d'écriture vers un **agent**, à une adresse donnée.
- Différentes variantes du protocole et de paramêtrage pour optimiser les transferts et le matériel nécessaire.
- Dans cadre de ECE_4SEO4, les variantes du protocole utilisées seront:
  - **Pipelined Transfer with Variable Latency**
  - **Pipelined Transfer with Variable Latency and Burst**

# Avalon

## Caractéristiques

- Protocole propriétaire: Développé par la société Altera (Intel) pour les "System On Chip" sur FPGA.
- Pas d'usage autre que ce contexte, à la différence des protocoles normalisés qui permettent l'interopérabilité.
- Transaction **synchrone** de type **accès mémoire**: c'est à dire un **hôte** émettant des requêtes de lecture ou d'écriture vers un **agent**, à une adresse donnée.
- Différentes variantes du protocole et de paramêtrage pour optimiser les transferts et le matériel nécessaire.
- Dans cadre de ECE_4SEO4, les variantes du protocole utilisées seront:
  - **Pipelined Transfer with Variable Latency**
  - **Pipelined Transfer with Variable Latency and Burst**

# Avalon

## Caractéristiques

- Protocole propriétaire: Développé par la société Altera (Intel) pour les "System On Chip" sur FPGA.
- Pas d'usage autre que ce contexte, à la différence des protocoles normalisés qui permettent l'interopérabilité.
- Transaction **synchrone** de type **accès mémoire**: c'est à dire un **hôte** émettant des requêtes de lecture ou d'écriture vers un **agent**, à une adresse donnée.
- Différentes variantes du protocole et de paramêtrage pour optimiser les transferts et le matériel nécessaire.
- Dans cadre de ECE_4SEO4, les variantes du protocole utilisées seront:
  - **Pipelined Transfer with Variable Latency**
  - **Pipelined Transfer with Variable Latency and Burst**

# Avalon

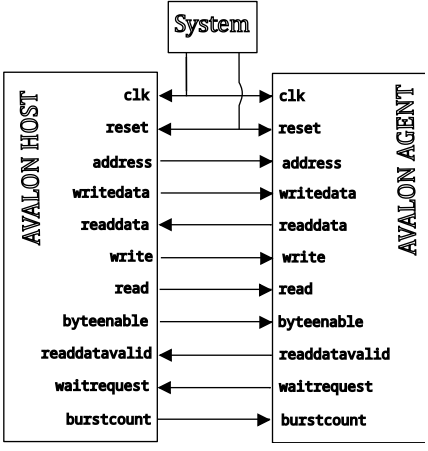
## Caractéristiques

- Protocole propriétaire: Développé par la société Altera (Intel) pour les "System On Chip" sur FPGA.
- Pas d'usage autre que ce contexte, à la différence des protocoles normalisés qui permettent l'interopérabilité.
- Transaction **synchrone** de type **accès mémoire**: c'est à dire un **hôte** émettant des requêtes de lecture ou d'écriture vers un **agent**, à une adresse donnée.
- Différentes variantes du protocole et de paramêtrage pour optimiser les transferts et le matériel nécessaire.
- Dans cadre de ECE_4SEO4, les variantes du protocole utilisées seront:
  - **Pipelined Transfer with Variable Latency**
  - **Pipelined Transfer with Variable Latency and Burst**

# Les signaux du bus Avalon

## reset et clk

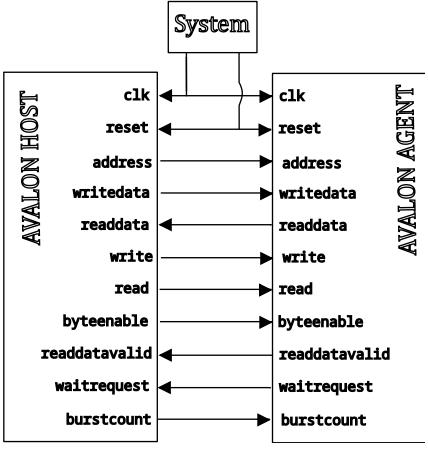
- Fournis par le système...
- **reset** : signal de reset actif à l'état haut
- **clk** : le bus Avalon est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les **hôtes** et **agents** communiquent à la fréquence du bus



# Les signaux du bus Avalon

## reset et clk

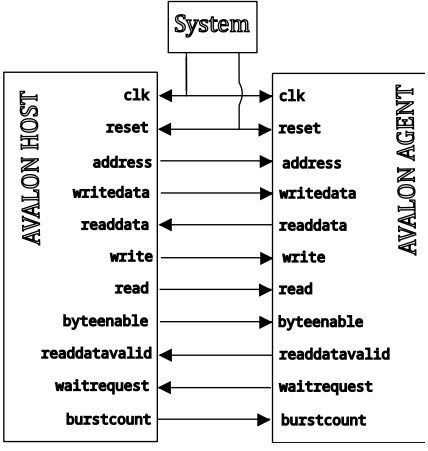
- Fournis par le système...
- **reset** : signal de reset actif à l'état haut
- **clk** : le bus Avalon est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les **hôtes** et **agents** communiquent à la fréquence du bus



# Les signaux du bus Avalon

## reset et clk

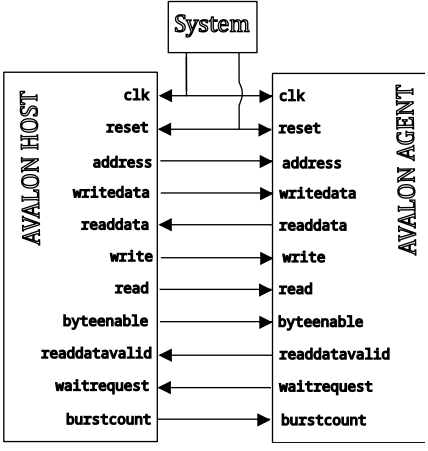
- Fournis par le système...
- **reset** : signal de reset actif à l'état haut
- **clk** : le bus Avalon est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les **hôtes** et **agents** communiquent à la fréquence du bus



# Les signaux du bus Avalon

## writedata, readdata et byteenable

- **writedata**/**readdata**: données à écrire ou lire
  - Largeur identique pour l'hôte et l'agent
  - Largeur identique en écriture et lecture
  - **DATA_BYTES** octets de large (paramètre fixe du bus)
- **byteenable** : masque de transfert
  - **DATA_BYTES** bits
  - Validation des octets **écrits**, inutile en lecture

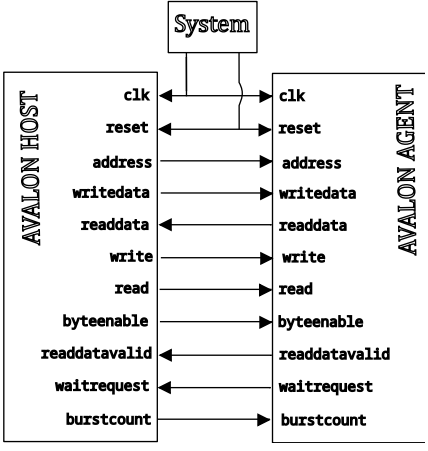



**Masques utiles**: Bus 32bits: 0001, 0010, 0100, 1000, 0011, 1100, 1111

# Les signaux du bus Avalon

## writedata, readdata et byteenable

- **writedata**/**readdata**: données à écrire ou lire
  - Largeur identique pour l'hôte et l'agent
  - Largeur identique en écriture et lecture
  - **DATA_BYTES** octets de large (paramètre fixe du bus)
- **byteenable** : masque de transfert
  - **DATA_BYTES** bits
  - Validation des octets **écrits**, inutile en lecture

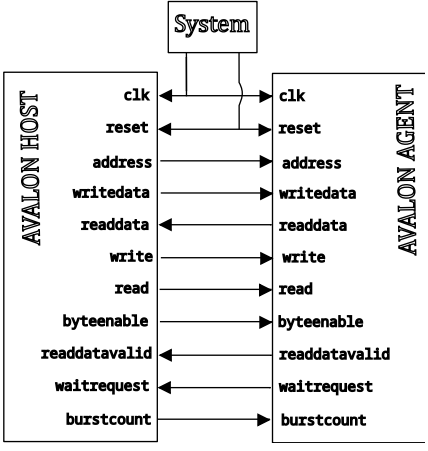



**Masques utiles**: Bus 32bits: 0001, 0010, 0100, 1000, 0011, 1100, 1111

# Les signaux du bus Avalon

## writedata, readdata et byteenable

- **writedata**/**readdata**: données à écrire ou lire
  - Largeur identique pour l'hôte et l'agent
  - Largeur identique en écriture et lecture
  - **DATA_BYTES** octets de large (paramètre fixe du bus)
- **byteenable** : masque de transfert
  - **DATA_BYTES** bits
  - Validation des octets **écrits**, inutile en lecture

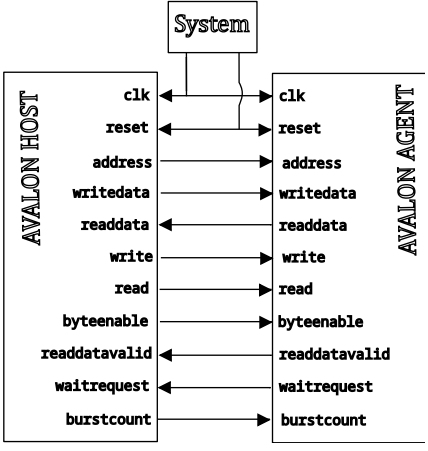



**Masques utiles**: Bus 32bits: 0001, 0010, 0100, 1000, 0011, 1100, 1111

# Les signaux du bus Avalon

## writedata, readdata et byteenable

- **writedata**/**readdata**: données à écrire ou lire
  - Largeur identique pour l'hôte et l'agent
  - Largeur identique en écriture et lecture
  - **DATA_BYTES** octets de large (paramètre fixe du bus)
- **byteenable** : masque de transfert
  - **DATA_BYTES** bits
  - Validation des octets **écrits**, inutile en lecture

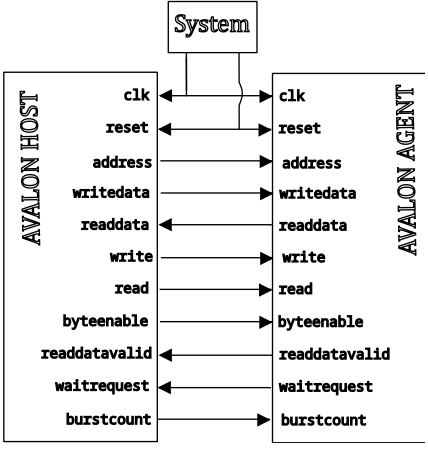



**Masques utiles**: Bus 32bits: 0001, 0010, 0100, 1000, 0011, 1100, 1111

# Les signaux du bus Avalon

## address

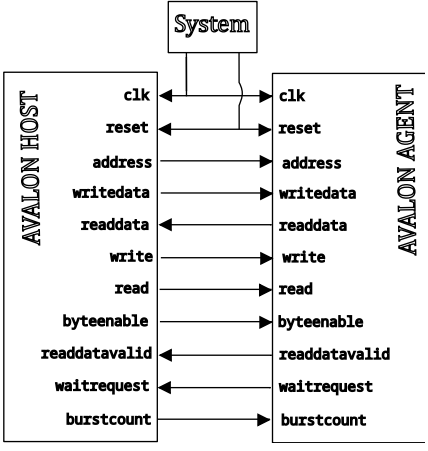
- **address** : Adresse du transfert (32 bits).
- Par convention l'adresse est alignée sur la taille du mot de donnée: address[1:0]=2'b00
- Les agents peuvent ignorer les deux bits de poids faible.



# Les signaux du bus Avalon

## address

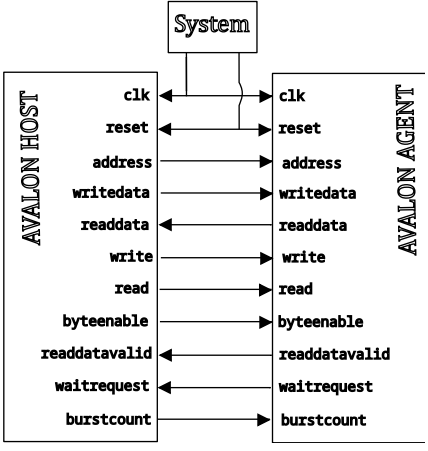
- **address** : Adresse du transfert (32 bits).
- Par convention l'adresse est alignée sur la taille du mot de donnée: address[1:0]=2'b00
- Les agents peuvent ignorer les deux bits de poids faible.



# Les signaux du bus Avalon

## address

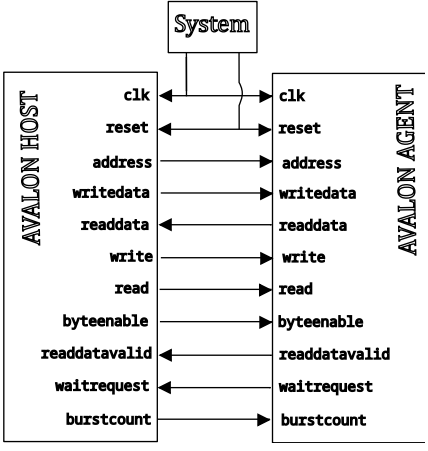
- **address** : Adresse du transfert (32 bits).
- Par convention l'adresse est alignée sur la taille du mot de donnée: address[1:0]=2'b00
- Les agents peuvent ignorer les deux bits de poids faible.



# Les signaux du bus Avalon

## signaux de contrôle: write, read, waitrequest, readdatavalid

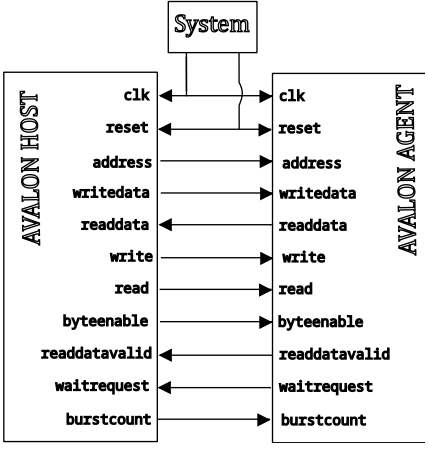
- **write**: l'hôte demande à faire une écriture
- **read**: l'hôte demande à faire une lecture
- **waitrequest**: l'agent demande à l'hôte d'attendre
- **readdatavalid**: l'agent signale la présence d'une donnée valide.



# Les signaux du bus Avalon

## signaux de contrôle: write, read, waitrequest, readdatavalid

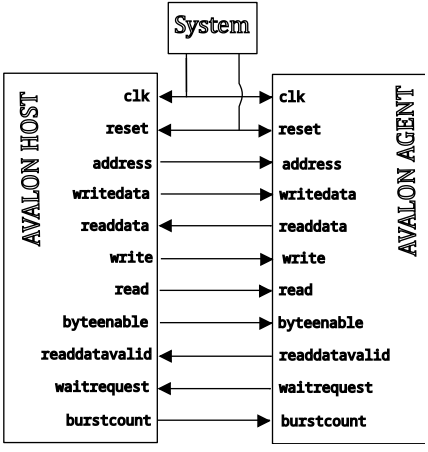
- **write**: l'hôte demande à faire une écriture
- **read**: l'hôte demande à faire une lecture
- **waitrequest**: l'agent demande à l'hôte d'attendre
- **readdatavalid**: l'agent signale la présence d'une donnée valide.



# Les signaux du bus Avalon

## signaux de contrôle: write, read, waitrequest, readdatavalid

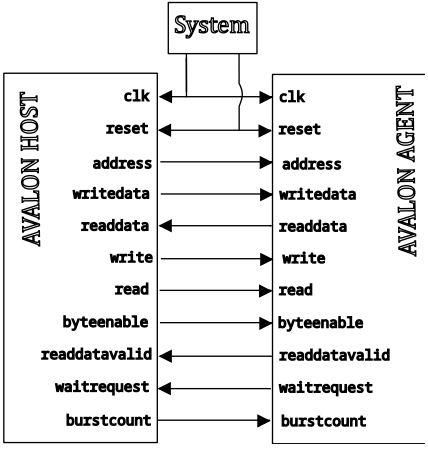
- **write**: l'hôte demande à faire une écriture
- **read**: l'hôte demande à faire une lecture
- **waitrequest**: l'agent demande à l'hôte d'attendre
- **readdatavalid**: l'agent signale la présence d'une donnée valide.



# Les signaux du bus Avalon

## signaux de contrôle: write, read, waitrequest, readdatavalid

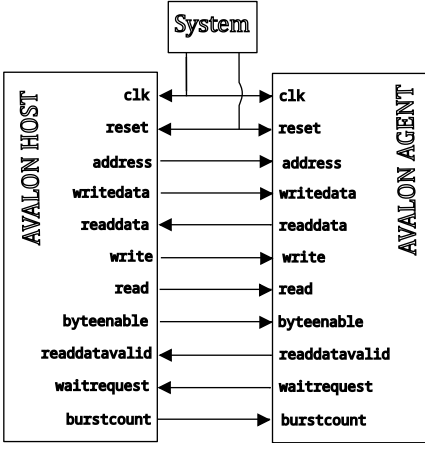
- **write**: l'hôte demande à faire une écriture
- **read**: l'hôte demande à faire une lecture
- **waitrequest**: l'agent demande à l'hôte d'attendre
- **readdatavalid**: l'agent signale la présence d'une donnée valide.



# Les signaux du bus Avalon

## burstcount

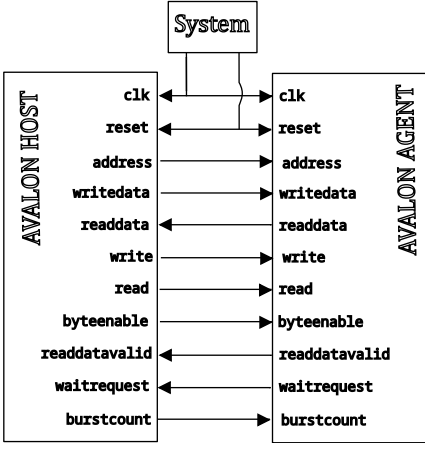
- **burstcount**=1 : l'hôte signale le transfert d'une donnée unique à l'adresse **address**.
- **burstcount**=N (>1) : l'hôte signale le transfert de N données consécutives à partir de l'adresse **address**.
- La taille maximale du burst est définie par un paramètre fixe du bus (**BURSTCOUNT_W**).



# Les signaux du bus Avalon

## burstcount

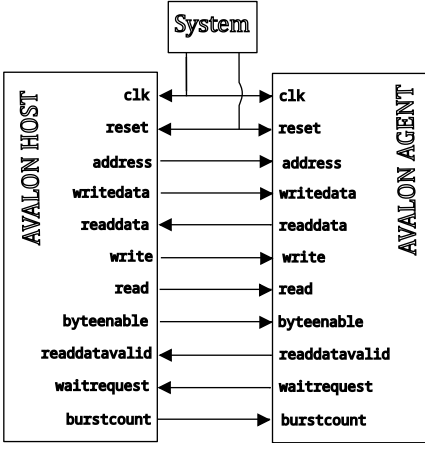
- **burstcount**
  - **burstcount**=1 : l'hôte signale le transfert d'une donnée unique à l'adresse **address**.
  - **burstcount**=N (>1) : l'hôte signale le transfert de N données consécutives à partir de l'adresse **address**.
  - La taille maximale du burst est définie par un paramètre fixe du bus (**BURSTCOUNT_W**).



# Les signaux du bus Avalon

## burstcount

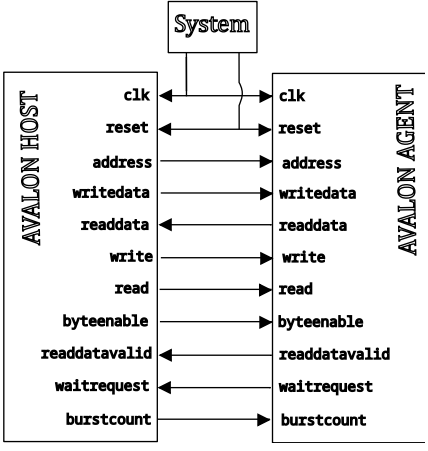
- **burstcount**
  - **burstcount**=1 : l'hôte signale le transfert d'une donnée unique à l'adresse **address**.
  - **burstcount**=N (>1) : l'hôte signale le transfert de N données consécutives à partir de l'adresse **address**.
  - La taille maximale du burst est définie par un paramètre fixe du bus (**BURSTCOUNT_W**).



# Les signaux du bus Avalon

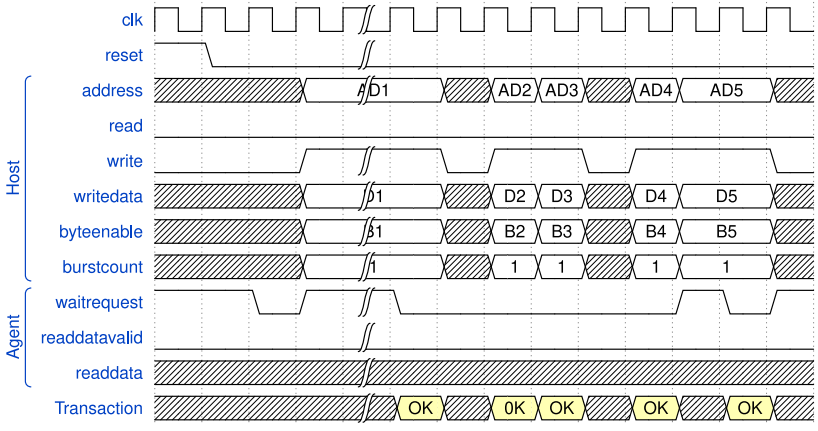
## burstcount

- **burstcount**
  - **burstcount**=1 : l'hôte signale le transfert d'une donnée unique à l'adresse **address**.
  - **burstcount**=N (>1) : l'hôte signale le transfert de N données consécutives à partir de l'adresse **address**.
  - La taille maximale du burst est définie par un paramètre fixe du bus (**BURSTCOUNT_W**).



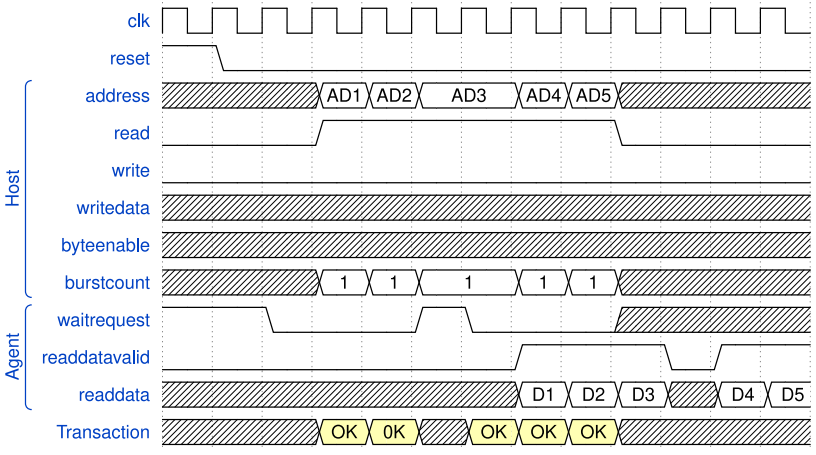
# Transactions en écriture Avalon

## Avec et sans attente demandée par l'agent



# Transactions en lecture Avalon

## "Avec et sans attente"



# Les transactions Avalon

## En résumé...

- La transaction a lieu lorsqu'il y a une demande (**read=1** ou **write=1**) et **waitrequest=0**.
- L'**hôte** peut enchaîner des requêtes sans interruption.
- L'**hôte** peut demander une lecture même s'il n'a pas reçu la donnée précédente. (sous conditions).
- L'**agent** renvoie les données dans l'ordre demandé par l'**hôte** avec une latence non définie à priori.
- L'**hôte** ne peux pas bloquer le retour des données (il doit être prêt à les recevoir).

# Les transactions Avalon

## En résumé...

- La transaction a lieu lorsqu'il y a une demande (**read=1** ou **write=1**) et **waitrequest=0**.
- L'**hôte** peut enchaîner des requêtes sans interruption.
- L'**hôte** peut demander une lecture même s'il n'a pas reçu la donnée précédente. (sous conditions).
- L'**agent** renvoie les données dans l'ordre demandé par l'**hôte** avec une latence non définie à priori.
- L'**hôte** ne peux pas bloquer le retour des données (il doit être prêt à les recevoir).

# Les transactions Avalon

## En résumé...

- La transaction a lieu lorsqu'il y a une demande (**read=1** ou **write=1**) et **waitrequest=0**.
- L'**hôte** peut enchaîner des requêtes sans interruption.
- L'**hôte** peut demander une lecture même s'il n'a pas reçu la donnée précédente. (sous conditions).
- L'**agent** renvoie les données dans l'ordre demandé par l'**hôte** avec une latence non définie à priori.
- L'**hôte** ne peux pas bloquer le retour des données (il doit être prêt à les recevoir).

# Les transactions Avalon

## En résumé...

- La transaction a lieu lorsqu'il y a une demande (**read=1** ou **write=1**) et **waitrequest=0**.
- L'**hôte** peut enchaîner des requêtes sans interruption.
- L'**hôte** peut demander une lecture même s'il n'a pas reçu la donnée précédente. (sous conditions).
- L'**agent** renvoie les données dans l'ordre demandé par l'**hôte** avec une latence non définie à priori.
- L'**hôte** ne peux pas bloquer le retour des données (il doit être prêt à les recevoir).

# Les transactions Avalon

## En résumé...

- La transaction a lieu lorsqu'il y a une demande (**read=1** ou **write=1**) et **waitrequest=0**.
- L'**hôte** peut enchaîner des requêtes sans interruption.
- L'**hôte** peut demander une lecture même s'il n'a pas reçu la donnée précédente. (sous conditions).
- L'**agent** renvoie les données dans l'ordre demandé par l'**hôte** avec une latence non définie à priori.
- L'**hôte** ne peux pas bloquer le retour des données (il doit être prêt à les recevoir).

# Une interface SystemVerilog pour le bus Avalon

```
interface avalon_if #(parameter DATA_BYTES=4, BURSTCOUNT_W=6) (input logic clk, input logic
reset);
```

```
logic  [31:0]   address;
logic  [DATA_BYTES-1:0]  byteenable;
logic  read;
logic  write;
logic  [8*DATA_BYTES-1:0] readdata;
logic  [8*DATA_BYTES-1:0] writedata;
logic  waitrequest;
logic  readdatavalid;
logic [BURSTCOUNT_W-1:0]  burstcount;
```

```
modport host (
  input  clk,
  input  reset,
  output address,
  output byteenable,
  output read,
  output write,
  output writedata,
  output burstcount,
  input  readdata,
  input  waitrequest,
  input  readdatavalid
);
```

```
modport agent (
  input  clk,
  input  reset,
  input  address,
  input  byteenable,
  input  read,
  input  write,
  input  writedata,
  input  burstcount,
  output readdata,
  output waitrequest,
  output readdatavalid
);
endinterface
```

# Avalon:Transferts de données en rafale

## "Permettre à l'esclave d'anticiper les traitements"

- Le signal **burstcount** indique le nombre de données consécutives à transférer.
- L'adresse de départ de la rafale est capturée par l'agent au début de la rafale
- Le signal **burstcount** est capturé par l'agent au début la rafale.
- Le signal **burstcount** n'est pas maintenu pendant la rafale.
- L'adresse de départ de la rafale n'est pas maintenue pendant la rafale.
- l'**agent** doit contenir un compteur pour gérer les adresses consécutives.

# Avalon:Transferts de données en rafale

## "Permettre à l'esclave d'anticiper les traitements"

- Le signal **burstcount** indique le nombre de données consécutives à transférer.
- L'adresse de départ de la rafale est capturée par l'agent au début de la rafale
- Le signal **burstcount** est capturé par l'agent au début la rafale.
- Le signal **burstcount** n'est pas maintenu pendant la rafale.
- L'adresse de départ de la rafale n'est pas maintenue pendant la rafale.
- l'**agent** doit contenir un compteur pour gérer les adresses consécutives.

# Avalon:Transferts de données en rafale

## "Permettre à l'esclave d'anticiper les traitements"

- Le signal **burstcount** indique le nombre de données consécutives à transférer.
- L'adresse de départ de la rafale est capturée par l'agent au début de la rafale
- Le signal **burstcount** est capturé par l'agent au début la rafale.
- Le signal **burstcount** n'est pas maintenu pendant la rafale.
- L'adresse de départ de la rafale n'est pas maintenue pendant la rafale.
- l'**agent** doit contenir un compteur pour gérer les adresses consécutives.

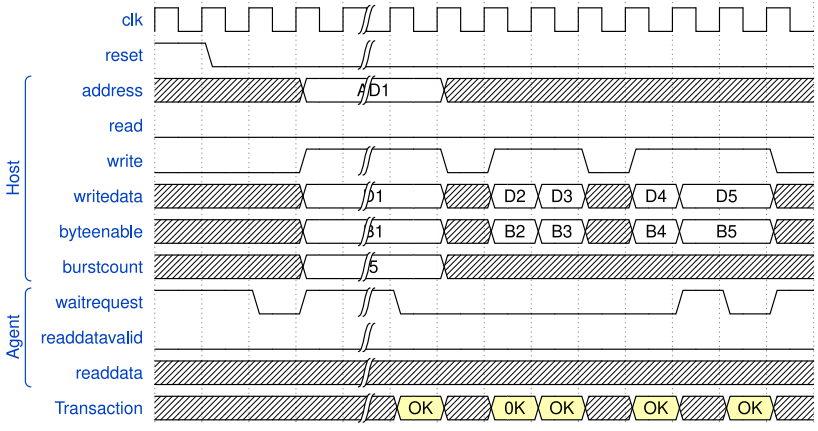
# Avalon:Transferts de données en rafale

## "Permettre à l'esclave d'anticiper les traitements"

- Le signal **burstcount** indique le nombre de données consécutives à transférer.
- L'adresse de départ de la rafale est capturée par l'agent au début de la rafale
- Le signal **burstcount** est capturé par l'agent au début la rafale.
- Le signal **burstcount** n'est pas maintenu pendant la rafale.
- L'adresse de départ de la rafale n'est pas maintenue pendant la rafale.
- l'**agent** doit contenir un compteur pour gérer les adresses consécutives.

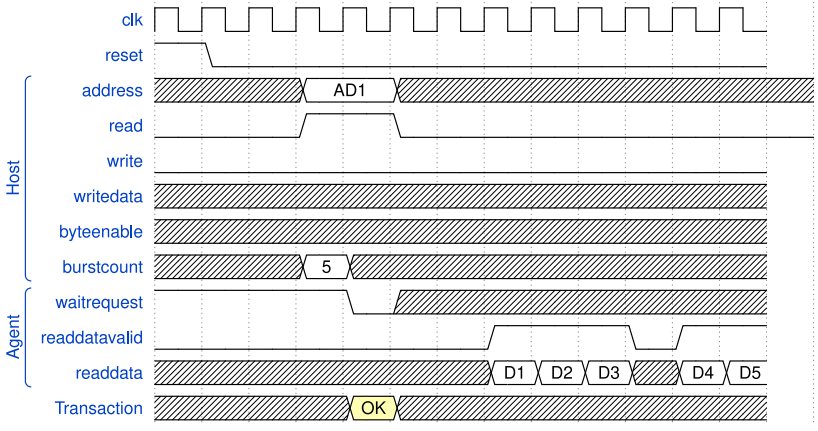
# Transactions en écriture Avalon: rafales

## Avec et sans attente demandée par l'agent



# Transactions en lecture Avalon: rafales

## Avec et sans attente demandée par l'agent



# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.

# Au travail...

## (1) Un contrôleur de mémoire à bus Avalon

- Coder un **agent** Avalon pour contrôler une mémoire synchrone interne du FPGA.
- Un module **unique** contenant à la fois le code de l'interface, et l'inférence de la mémoire.
- La taille de la mémoire (nombre de bit d'adresse) devra être paramétrable
- La taille par défaut de la mémoire sera de 2048 mots de 32 bits.
- Nous utiliserons une "interface" SystemVerilog pour décrire le bus Avalon
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.
- Deux versions du code.