TELECOM ParisTech

INSTITUT Mines-Télécom

université PARIS-SACLAY

SystemVerilog : Bus, protocoles et interfaces ...

Techniques de codage, étude d'une norme

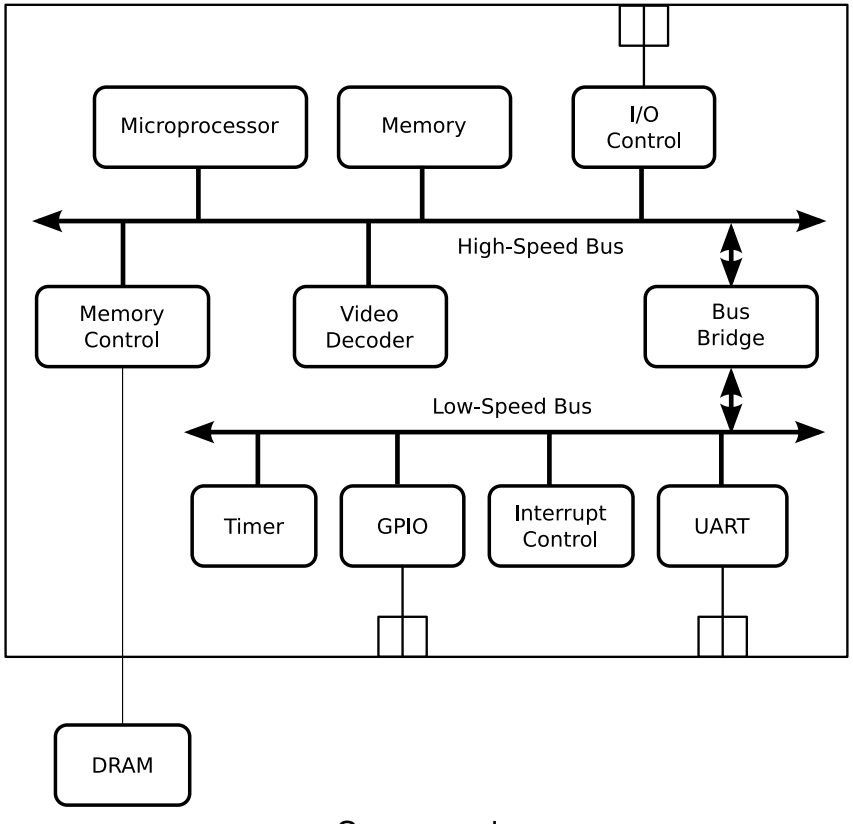
Yves Mathieu

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,Avallon,...)
  - Pour nous Wishbone, norme libre, simple et éventuellement boguée...

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,Avallon,...)
  - Pour nous Wishbone, norme libre, simple et éventuellement boguée...

Introduction (2)

Rôle des normes de bus

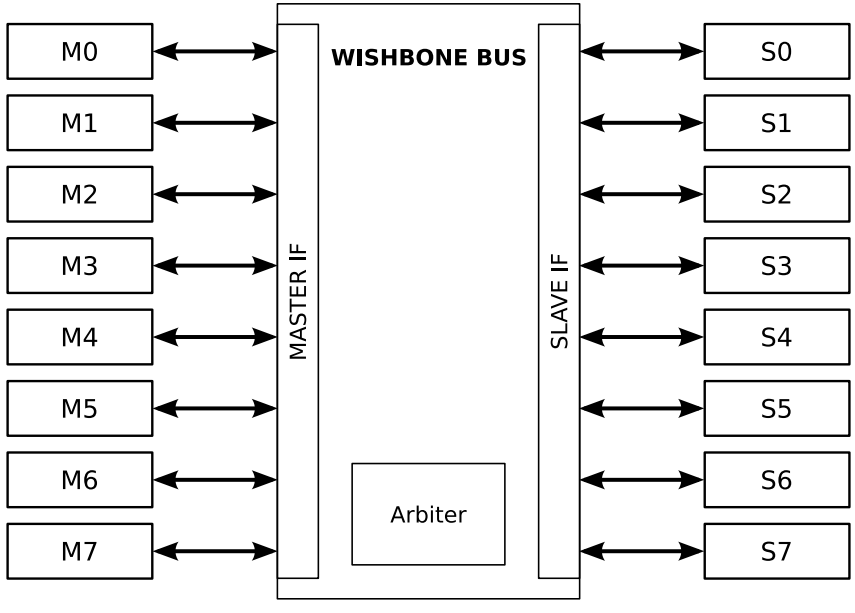
- Unifier les interfaces entre blocs:
  - Faciliter le design des blocs
  - Faciliter le test des blocs (programmes de tests génériques)
  - Un système = un lego de blocs
  - Permettre la création d'outils de génération "Système" (Qsys...)
- Organiser l'arbitrage
- Quelle norme de bus utiliser ?
  -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

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 déclarés 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 déclarés 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

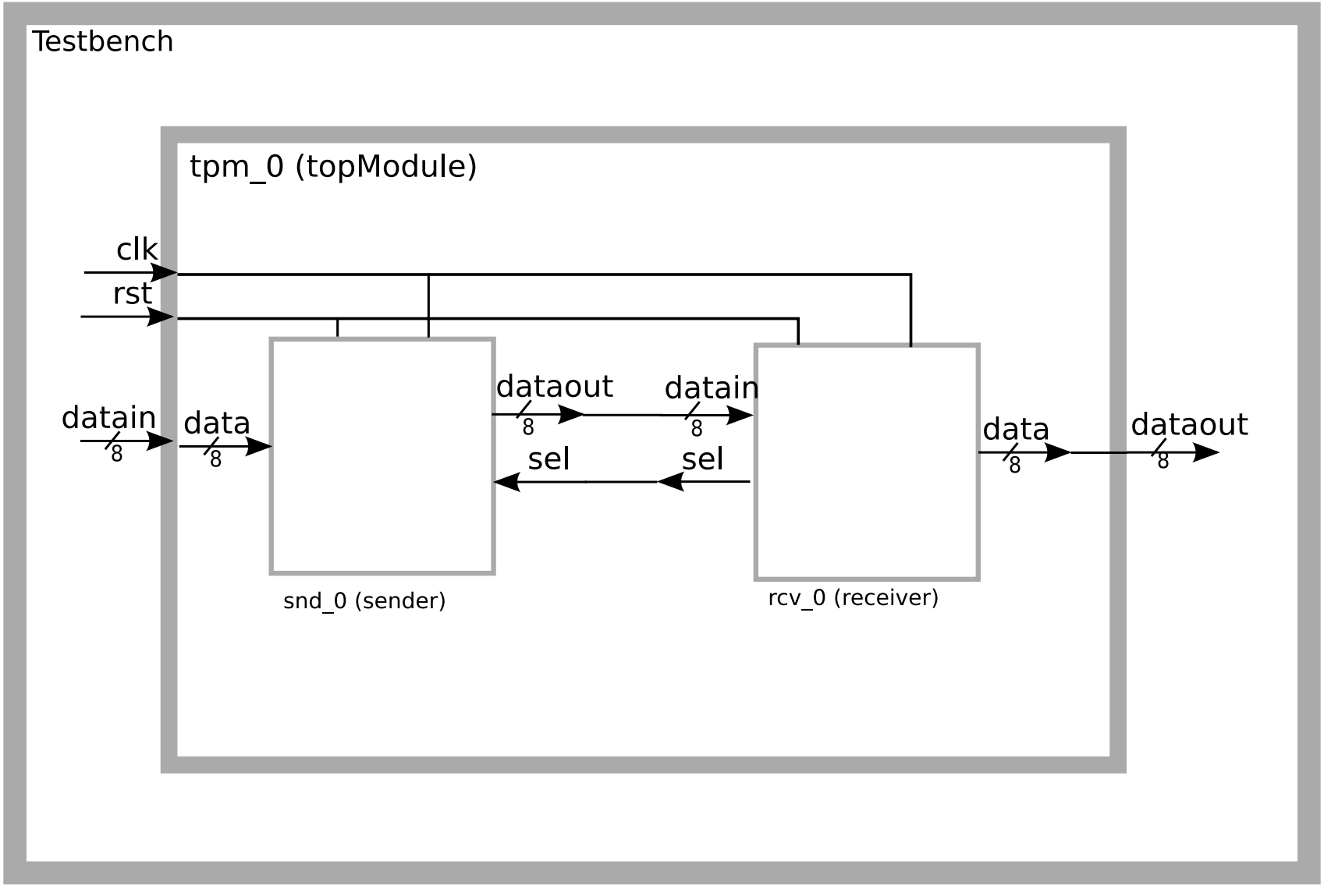
- Regrouper des signaux et représenter par un port unique.
- Ne déclarer les signaux internes qu'un seule fois.
- Un module connecté à une interface n'est connecté qu'à un seul port.
- Des **function** et **task** peuvent être déclarés dans une interface:
  - Pas de duplication de codes identiques dans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

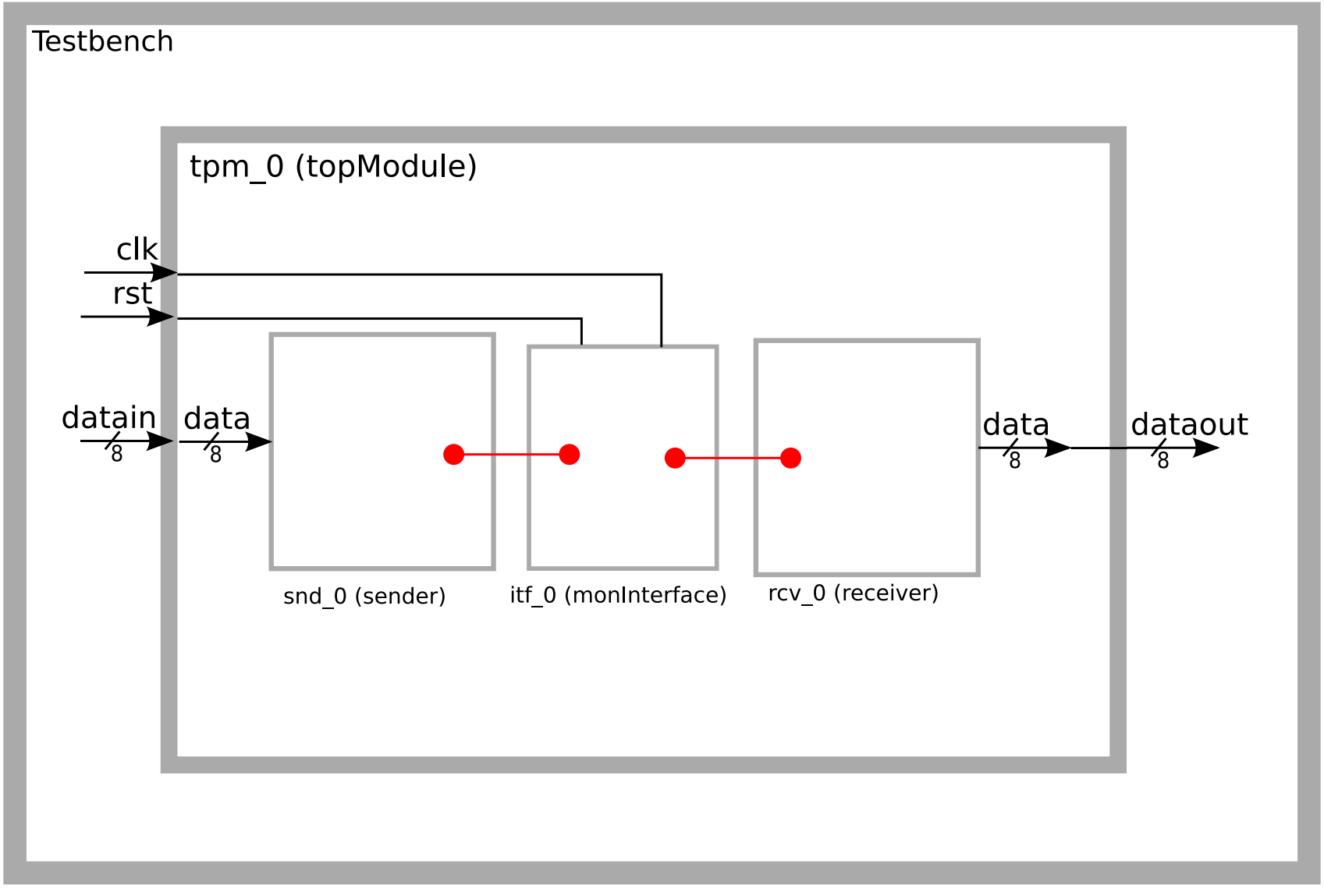
L' interface systemVerilog

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
```

Mise en pratique: Créez un fichier **monInterface.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique: Créez un fichier **monInterface.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique: Créez un fichier **monInterface.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique: Créez un fichier **monInterface.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **receiver.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **receiver.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **receiver.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **receiver.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **receiver.sv** , compilez le en vue d'une future simulation.

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
```

Mise en pratique : Créez un fichier **sender.sv** , compilez le en vue d'une future simulation.

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
```

Au travail : Créez un fichier **topModule.sv** , compilez

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
```

Au travail : Créez un fichier **topModule.sv** , compilez

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
```

Au travail : Créez un fichier **topModule.sv** , compilez

Au travail

Simulation et synthèse

- Créez un fichier **testbench.sv**, compilez le, lancez la simulation graphique.
- Observez bien la hiérarchie : l'interface est un bloc à art entière contenant des signaux
- Les signaux des interfaces ne font pas partie des modules...
- Synthétisez avec précision le module "receiver.sv", il y a des erreurs , pourquoi ?
- Ajoutez au projet le module "monInterface.sv", recommencez la synthèse.
- Examinez les entrées/sortie sur le shéma RTL généré, qu'en pensez vous ?
- Ajoutez les modules "sender" et "topModule", refaites la synthèse et vérifiez le résultat.

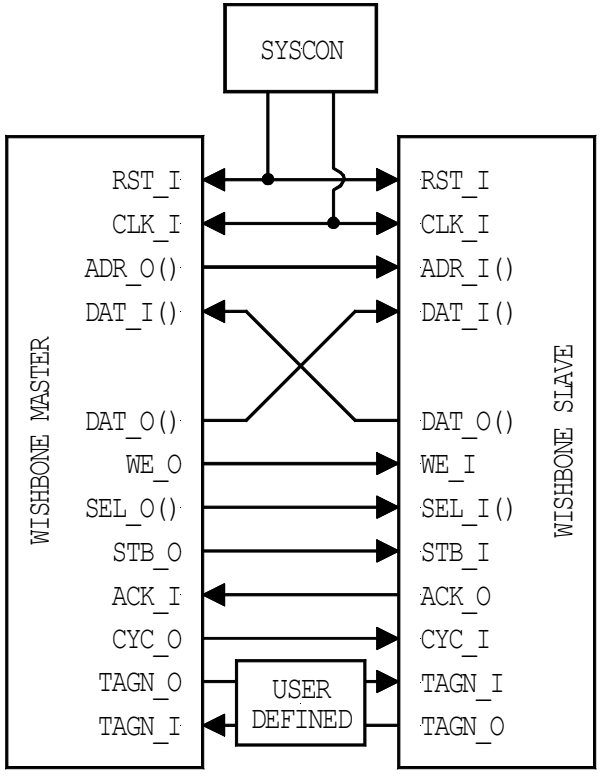
```
module testbench;
  logic [7:0] din  ;
  logic [7:0] dout ;
  bit clk, rst ;

  topModule tpm_0(clk,rst,din,dout) ;

  initial forever #5 clk = ~clk;

  initial begin
    rst = 1 ; #20 ; rst = 0 ;
  end

  always@(posedge clk)
  if(rst)
    din = 0 ;
  else
    din = din+1 ;
endmodule
```

Plan

Les signaux du bus Wishbone

signaux RST et CLK

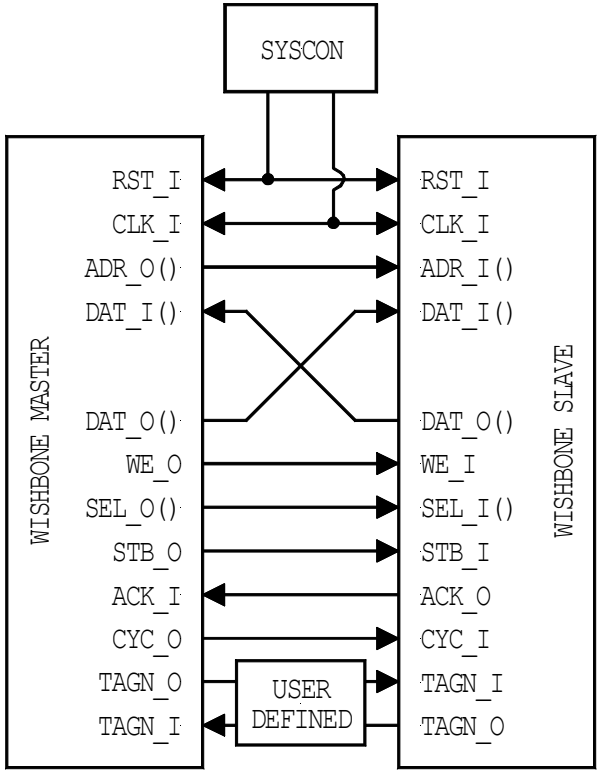
- Fournis par le système...
- RST : signal de reset actif à l'état haut
- CLK : le bus Wishbone est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les maîtres et esclaves communiquent à la fréquence du bus



Les signaux du bus Wishbone

signaux RST et CLK

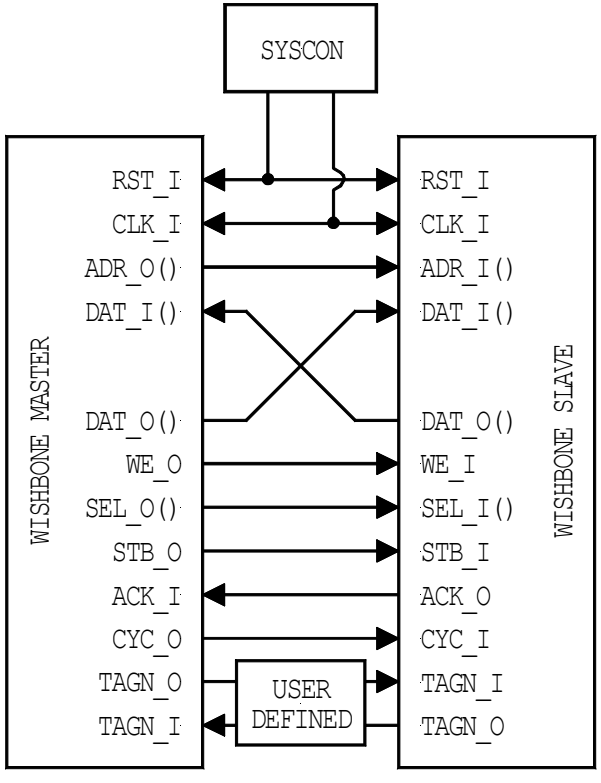
- Fournis par le système...
- RST : signal de reset actif à l'état haut
- CLK : le bus Wishbone est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les maîtres et esclaves communiquent à la fréquence du bus



Les signaux du bus Wishbone

signaux RST et CLK

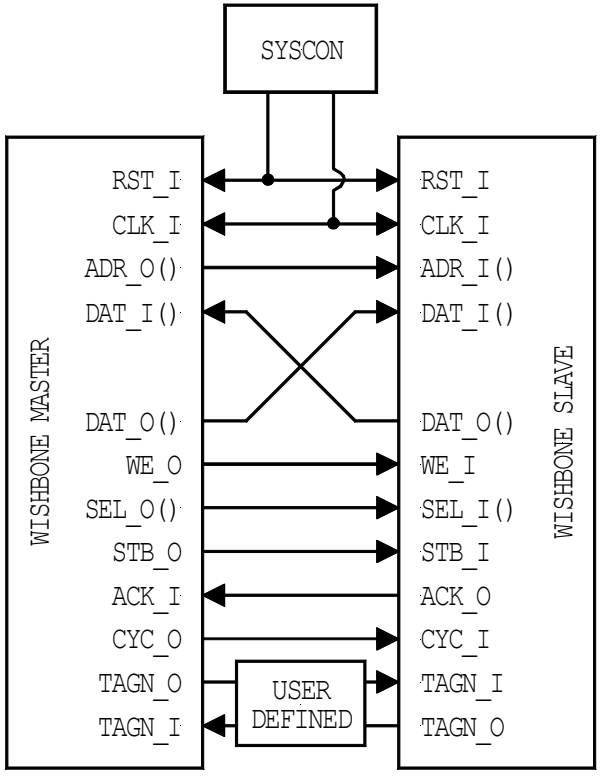
- Fournis par le système...
- RST : signal de reset actif à l'état haut
- CLK : le bus Wishbone est synchrone
  - les signaux sont validés par les fronts montants de l'horloge
  - les maîtres et esclaves communiquent à la fréquence du bus



Les signaux du bus Wishbone

signaux DAT et SEL

- DAT : Donnée à transférer
  - Canaux IN et OUT séparés
  - N = 1,2,4 ou 8 octets.
  - Largeur identique pour Maître et Esclave.
- SEL : masque de transfert
  - Mot de N bits
  - Validation des octets **écrits**

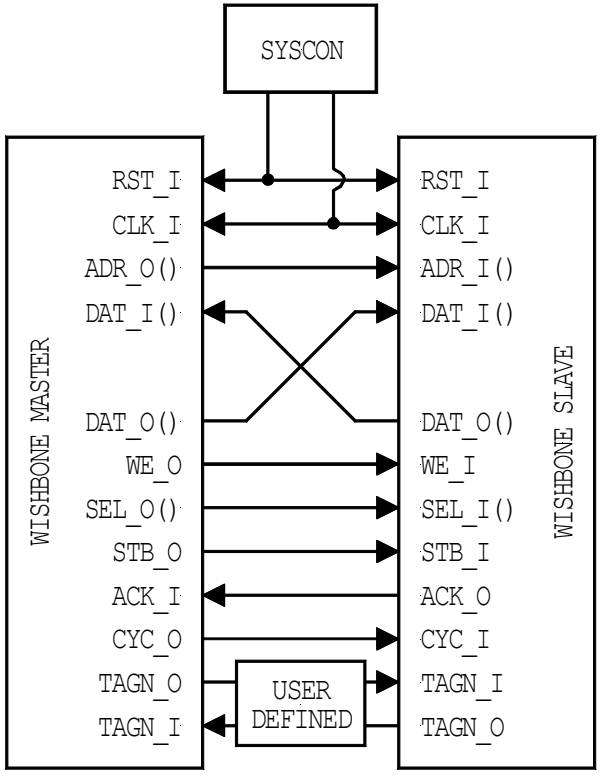


Attention: Bus 32bits, masques autorisés: 0001, 0010, 0100, 1000, 0011, 1100, 1111

Les signaux du bus Wishbone

signaux DAT et SEL

- DAT : Donnée à transférer
  - Canaux IN et OUT séparés
  - N = 1,2,4 ou 8 octets.
  - Largeur identique pour Maître et Esclave.
- SEL : masque de transfert
  - Mot de N bits
  - Validation des octets **écrits**

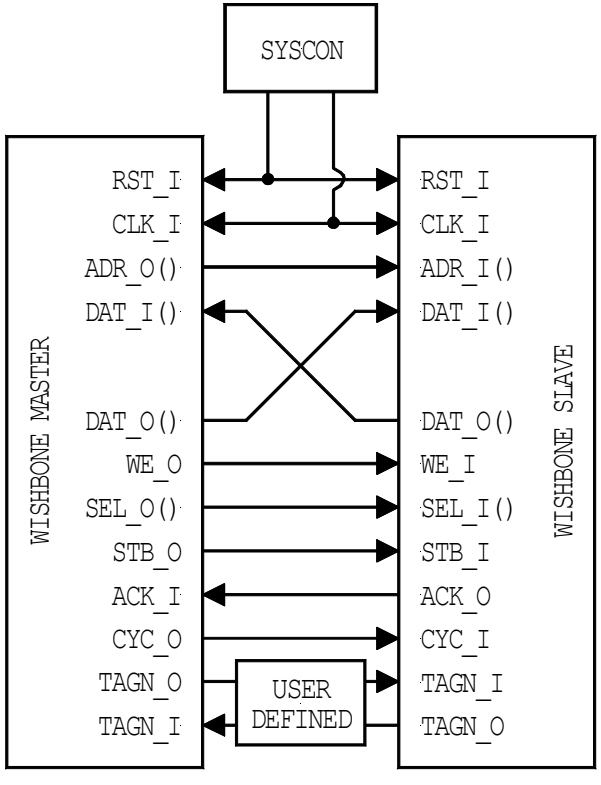


Attention: Bus 32bits, masques autorisés: 0001, 0010, 0100, 1000, 0011, 1100, 1111

Les signaux du bus Wishbone

signal ADR

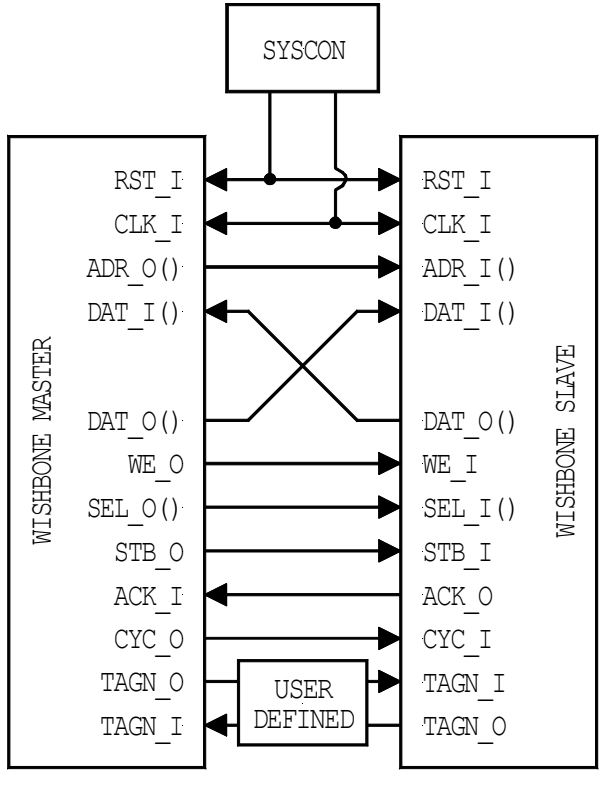
- ADR : Adresse du transfert
- Norme: adresse alignée sur la taille du mot
  - mots 32bits : ADR[max-1:2]
- Pratique: on transmet quand même des bits de poids faible
  - 32bits : ADR[1:0] = 00
  - les esclaves peuvent ignorer les bits de poids faible.
- Attention à l'«indianité »du bus (voir norme 3.5 DATA organisation)



Les signaux du bus Wishbone

signal ADR

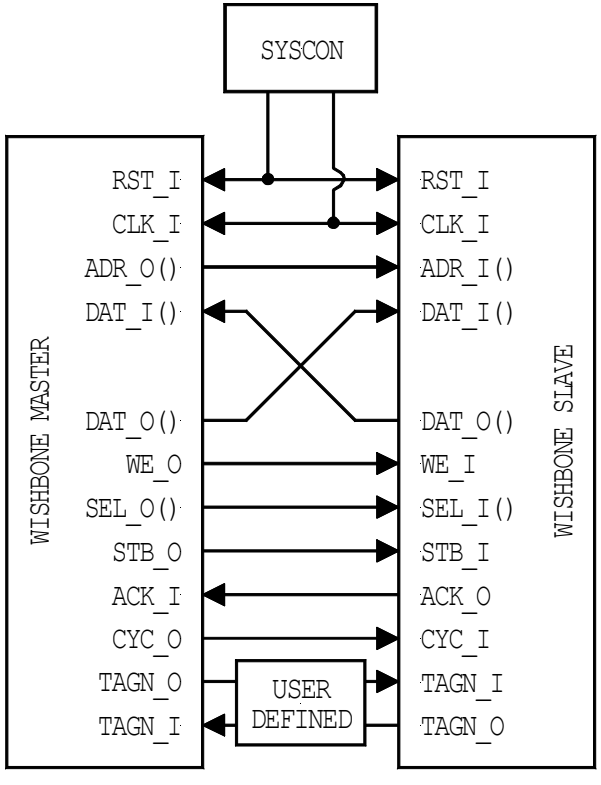
- ADR : Adresse du transfert
- Norme: adresse alignée sur la taille du mot
  - mots 32bits : ADR[max-1:2]
- Pratique: on transmet quand même des bits de poids faible
  - 32bits : ADR[1:0] = 00
  - les esclaves peuvent ignorer les bits de poids faible.
- Attention à l'«indianité »du bus (voir norme 3.5 DATA organisation)



Les signaux du bus Wishbone

signal ADR

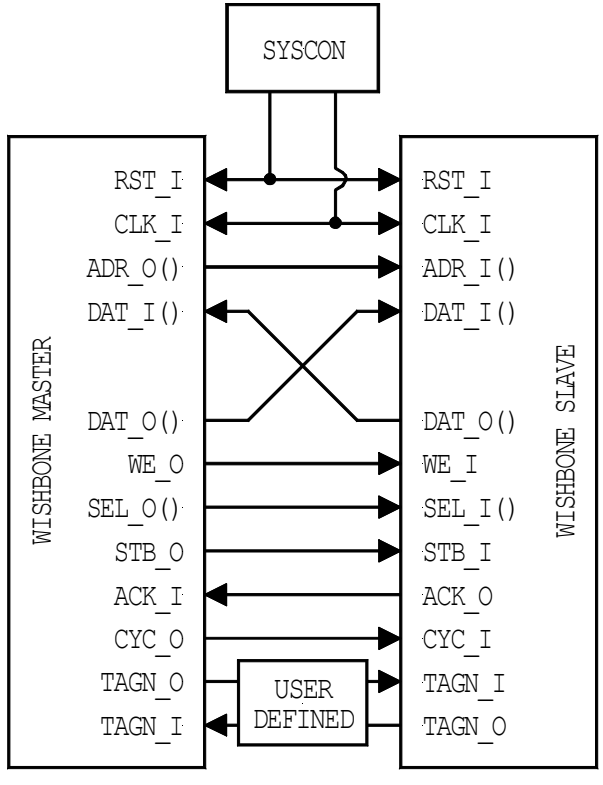
- ADR : Adresse du transfert
- Norme: adresse alignée sur la taille du mot
  - mots 32bits : ADR[max-1:2]
- Pratique: on transmet quand même des bits de poids faible
  - 32bits : ADR[1:0] = 00
  - les esclaves peuvent ignorer les bits de poids faible.
- Attention à l'«indianité »du bus (voir norme 3.5 DATA organisation)



Les signaux du bus Wishbone

signal ADR

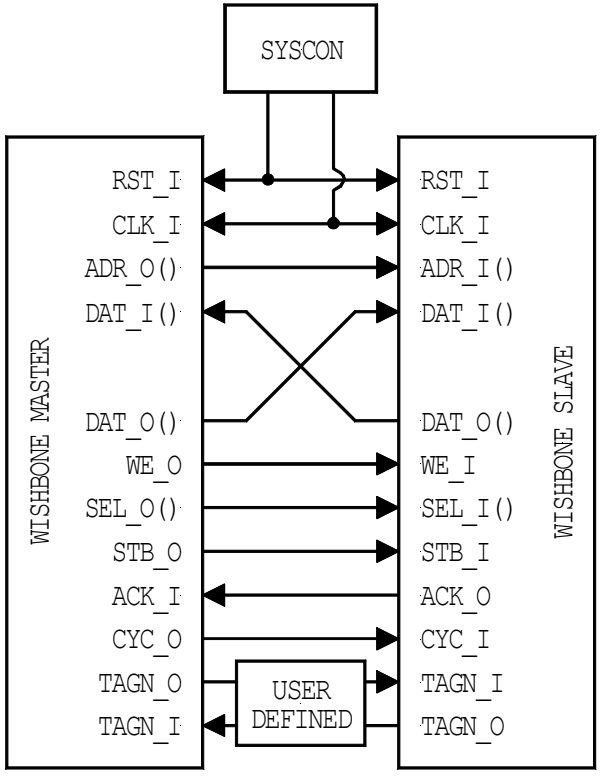
- ADR : Adresse du transfert
- Norme: adresse alignée sur la taille du mot
  - mots 32bits : ADR[max-1:2]
- Pratique: on transmet quand même des bits de poids faible
  - 32bits : ADR[1:0] = 00
  - les esclaves peuvent ignorer les bits de poids faible.
- Attention à l'«indianité »du bus (voir norme 3.5 DATA organisation)



Les signaux du bus Wishbone

signaux de contrôle WE, STB, CYC, ACK

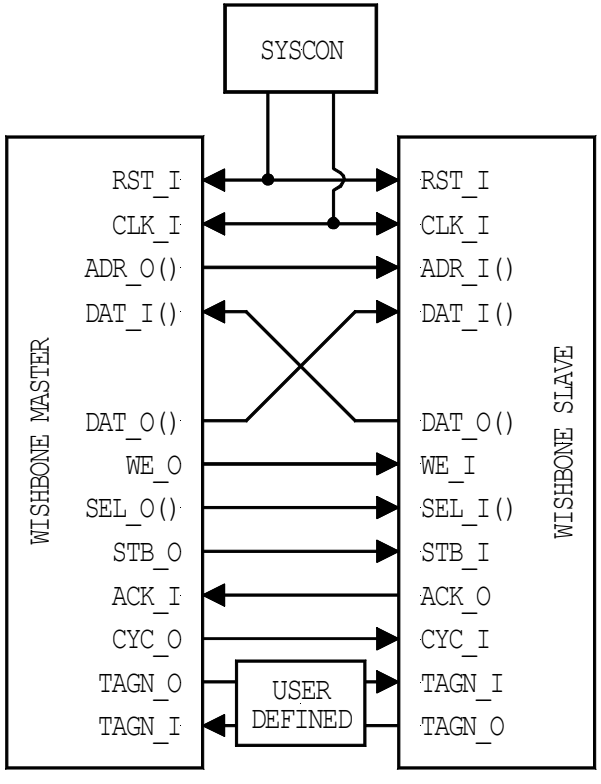
- CYC : le maître demande l'accès au bus
- STB : le maître valide un cycle d'accès (CYC=1)
- WE : WE=1/0 le cycle d'accès est en écriture/lecture
- ACK : l'esclave valide le cycle d'accès:
  - Ecriture effectuée
  - Donnée présente en lecture



Les signaux du bus Wishbone

signaux de contrôle WE, STB, CYC, ACK

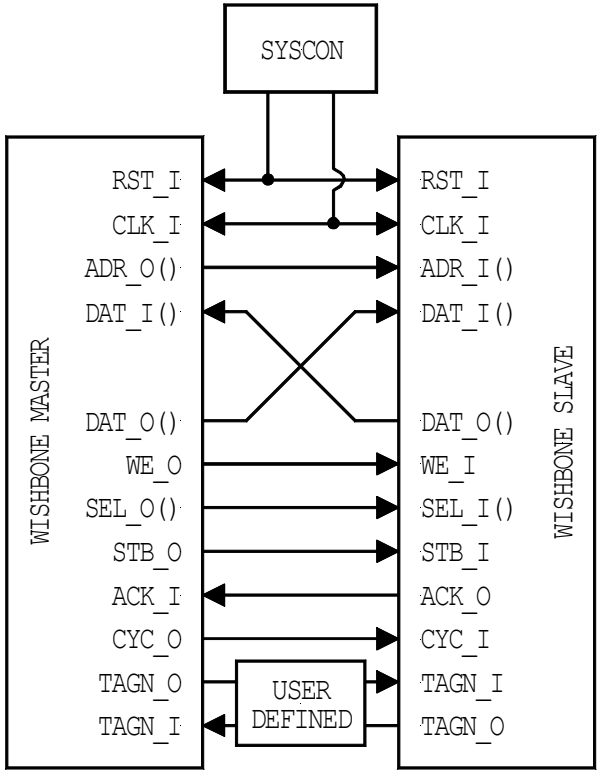
- CYC : le maître demande l'accès au bus
- STB : le maître valide un cycle d'accès (CYC=1)
- WE : WE=1/0 le cycle d'accès est en écriture/lecture
- ACK : l'esclave valide le cycle d'accès:
  - Ecriture effectuée
  - Donnée présente en lecture



Les signaux du bus Wishbone

signaux de contrôle WE, STB, CYC, ACK

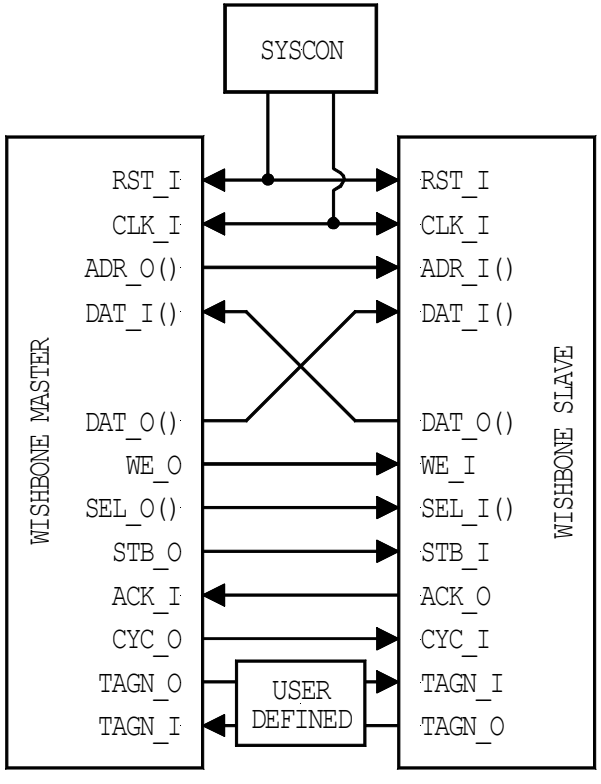
- CYC : le maître demande l'accès au bus
- STB : le maître valide un cycle d'accès (CYC=1)
- WE : WE=1/0 le cycle d'accès est en écriture/lecture
- ACK : l'esclave valide le cycle d'accès:
  - Ecriture effectuée
  - Donnée présente en lecture



Les signaux du bus Wishbone

signaux de contrôle WE, STB, CYC, ACK

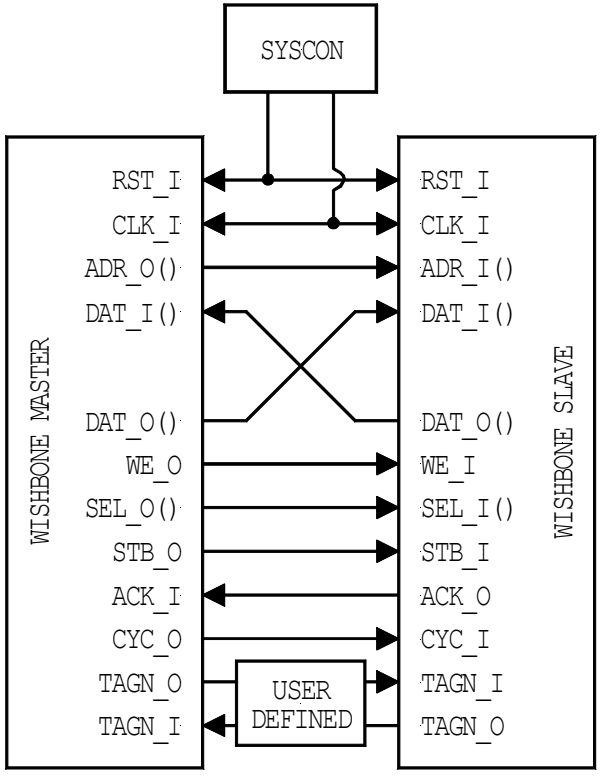
- CYC : le maître demande l'accès au bus
- STB : le maître valide un cycle d'accès (CYC=1)
- WE : WE=1/0 le cycle d'accès est en écriture/lecture
- ACK : l'esclave valide le cycle d'accès:
  - Ecriture effectuée
  - Donnée présente en lecture



Les signaux du bus Wishbone

Signaux auxiliaires

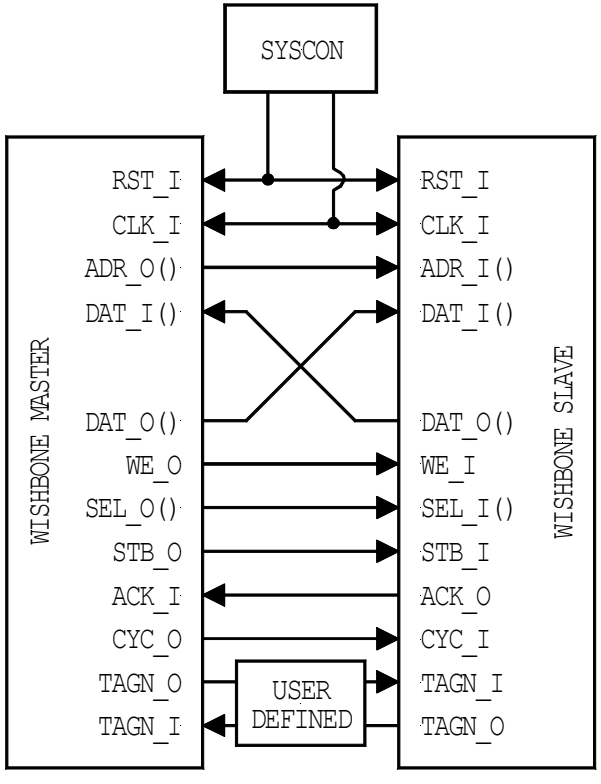
- Signaux non utilisés dans le cadre de SE204
  - RTY : l'esclave n'est pas prêt.
  - ERR : erreur signalée par l'esclave.
  - **Forcer 0 dans nos esclaves.**
- Signaux d'extension
  - TAGS : qualifier plus précisément une transaction...
  - Voir plus loin "Registered Feedback"



Les signaux du bus Wishbone

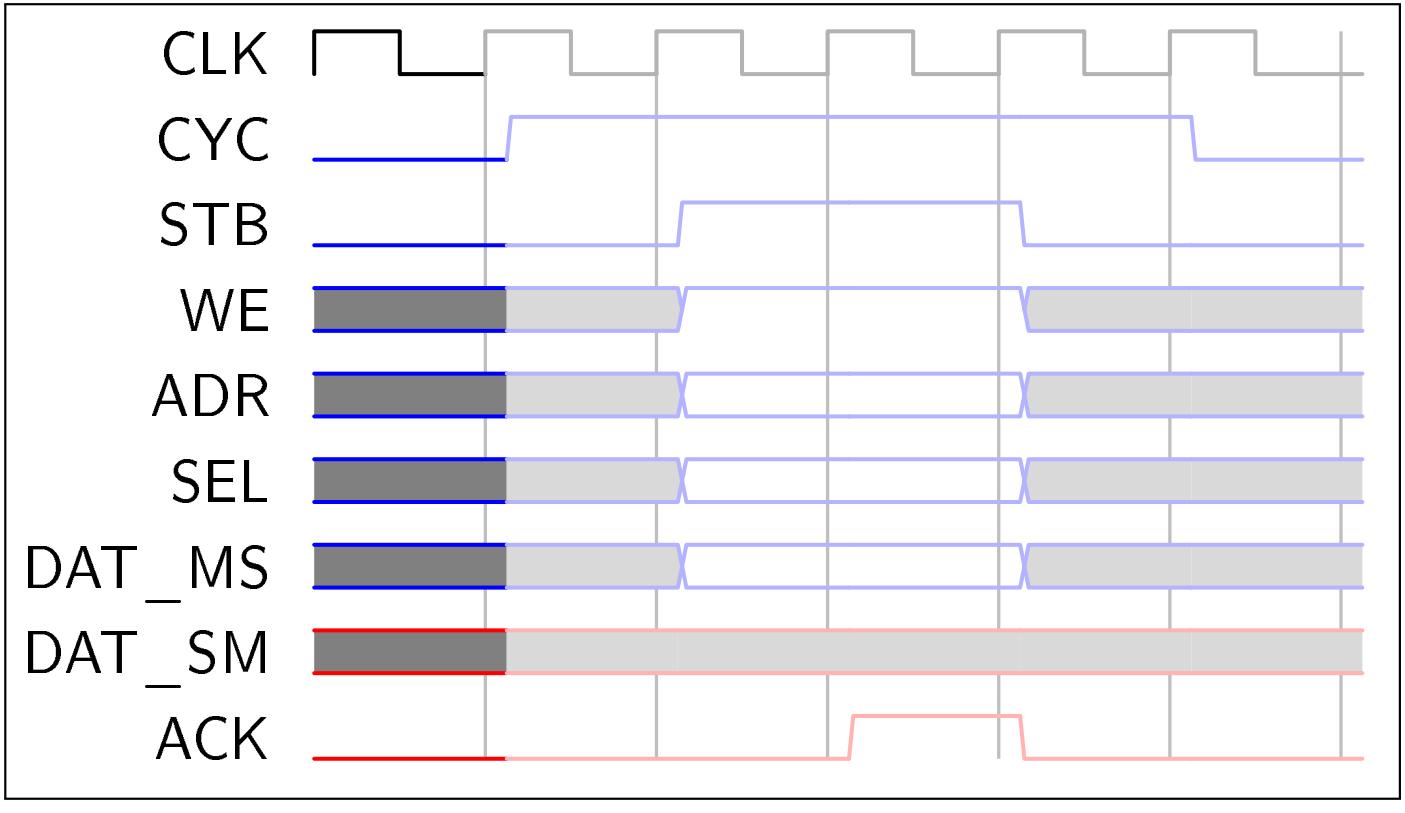
Signaux auxiliaires

- Signaux non utilisés dans le cadre de SE204
  - RTY : l'esclave n'est pas prêt.
  - ERR : erreur signalée par l'esclave.
  - **Forcer 0 dans nos esclaves.**
- Signaux d'extension
  - TAGS : qualifier plus précisément une transaction...
  - Voir plus loin "Registered Feedback"



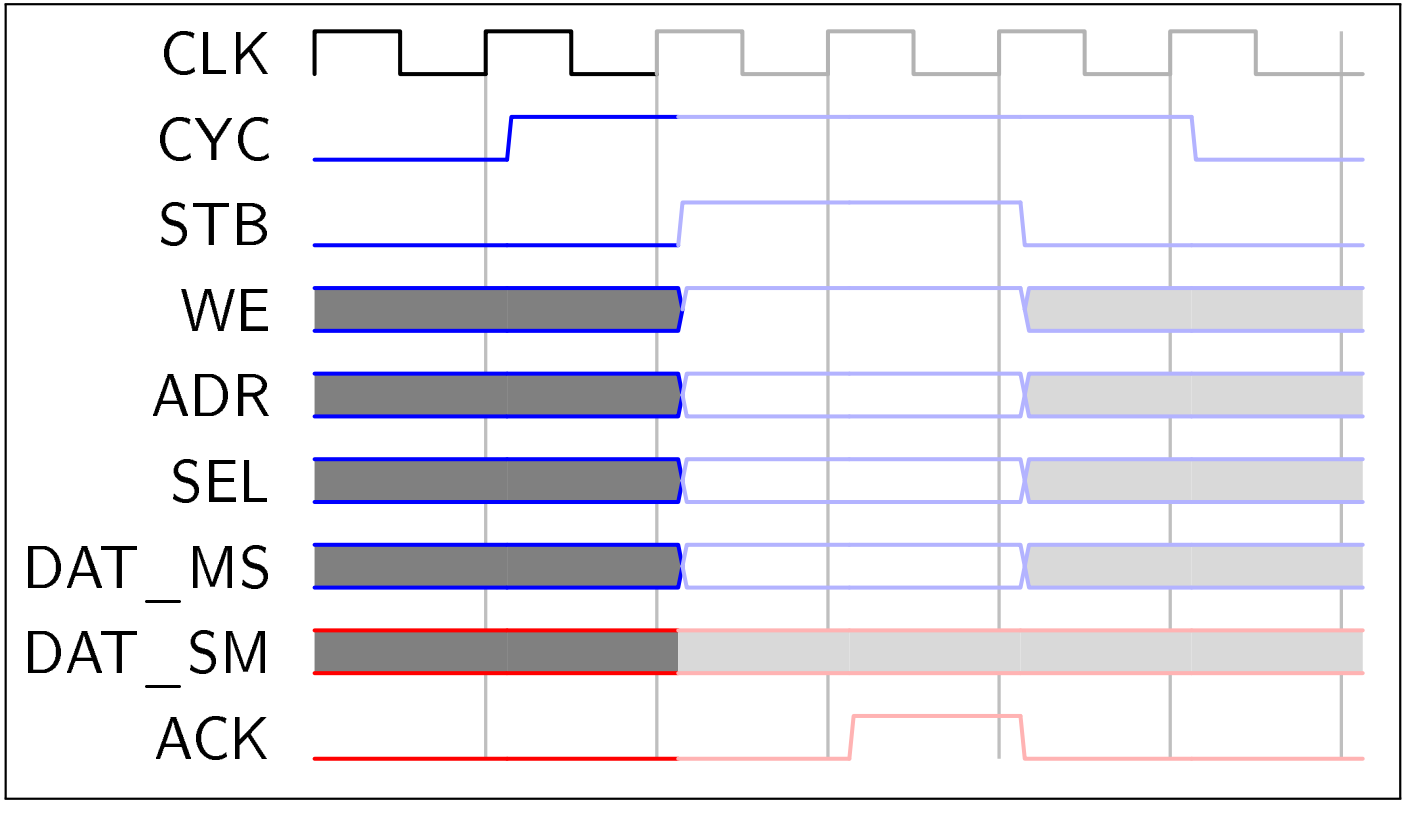
Une transaction en écriture Wishbone

"Classic Bus Cycle"



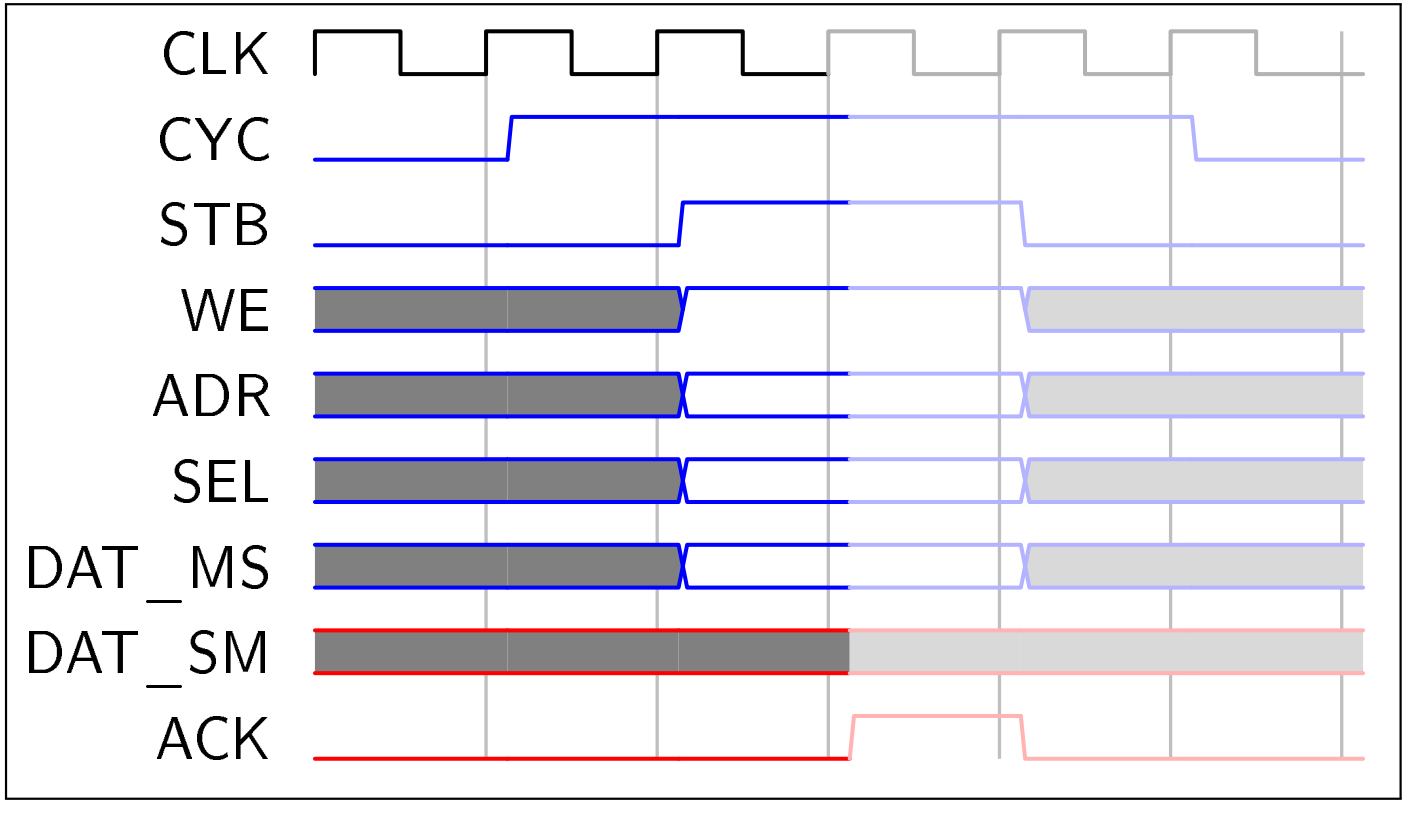
Une transaction en écriture Wishbone

"Classic Bus Cycle"



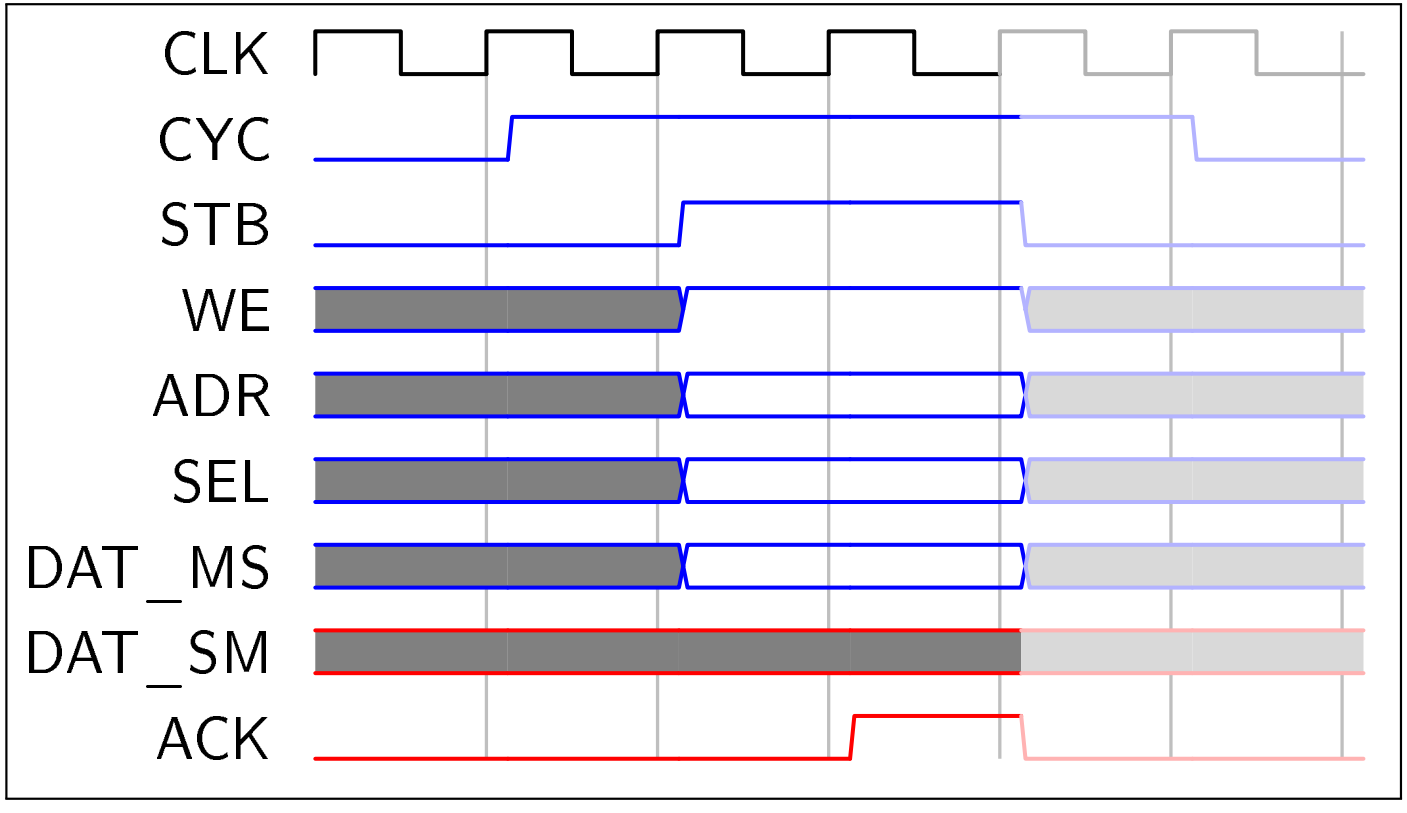
Une transaction en écriture Wishbone

"Classic Bus Cycle"



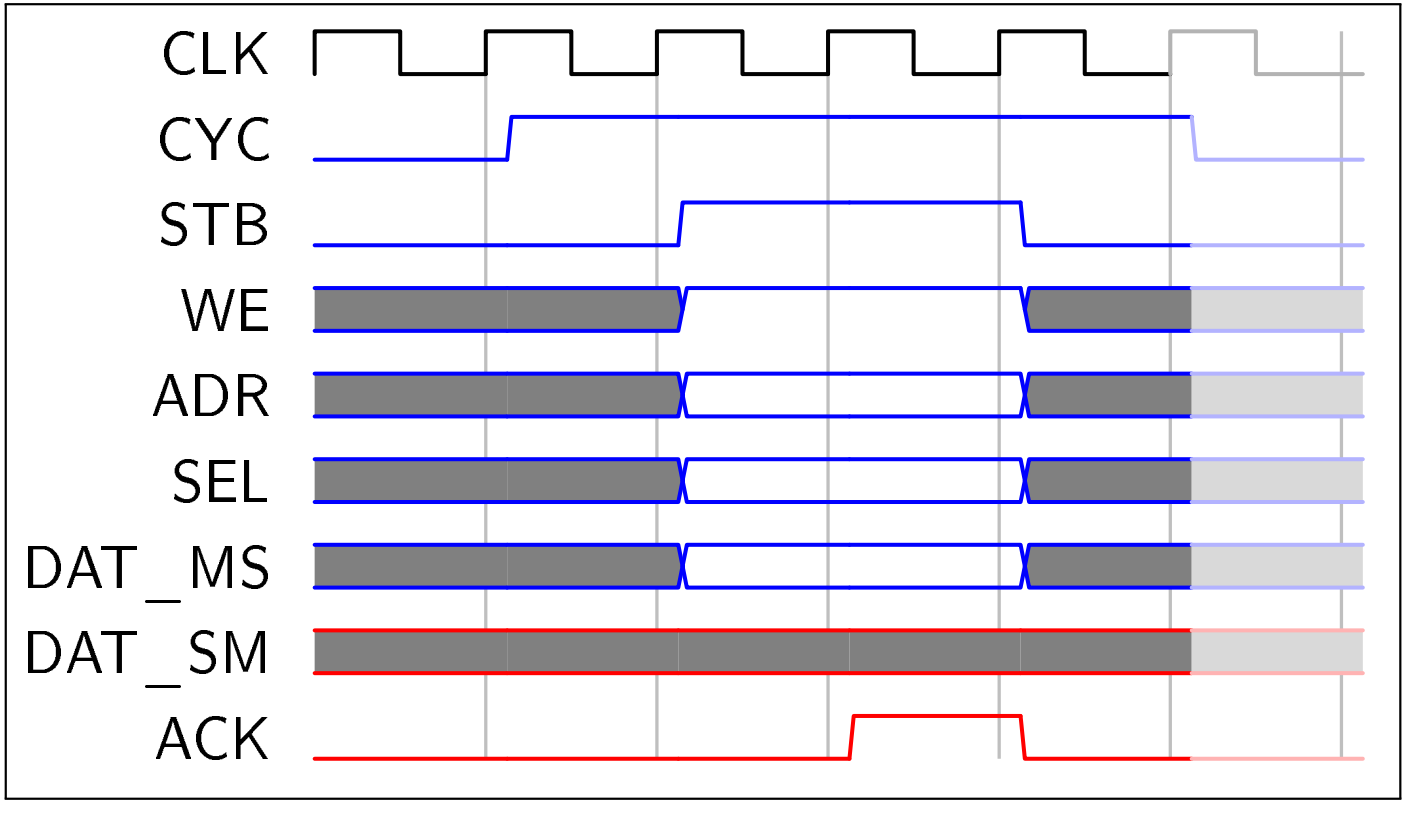
Une transaction en écriture Wishbone

"Classic Bus Cycle"



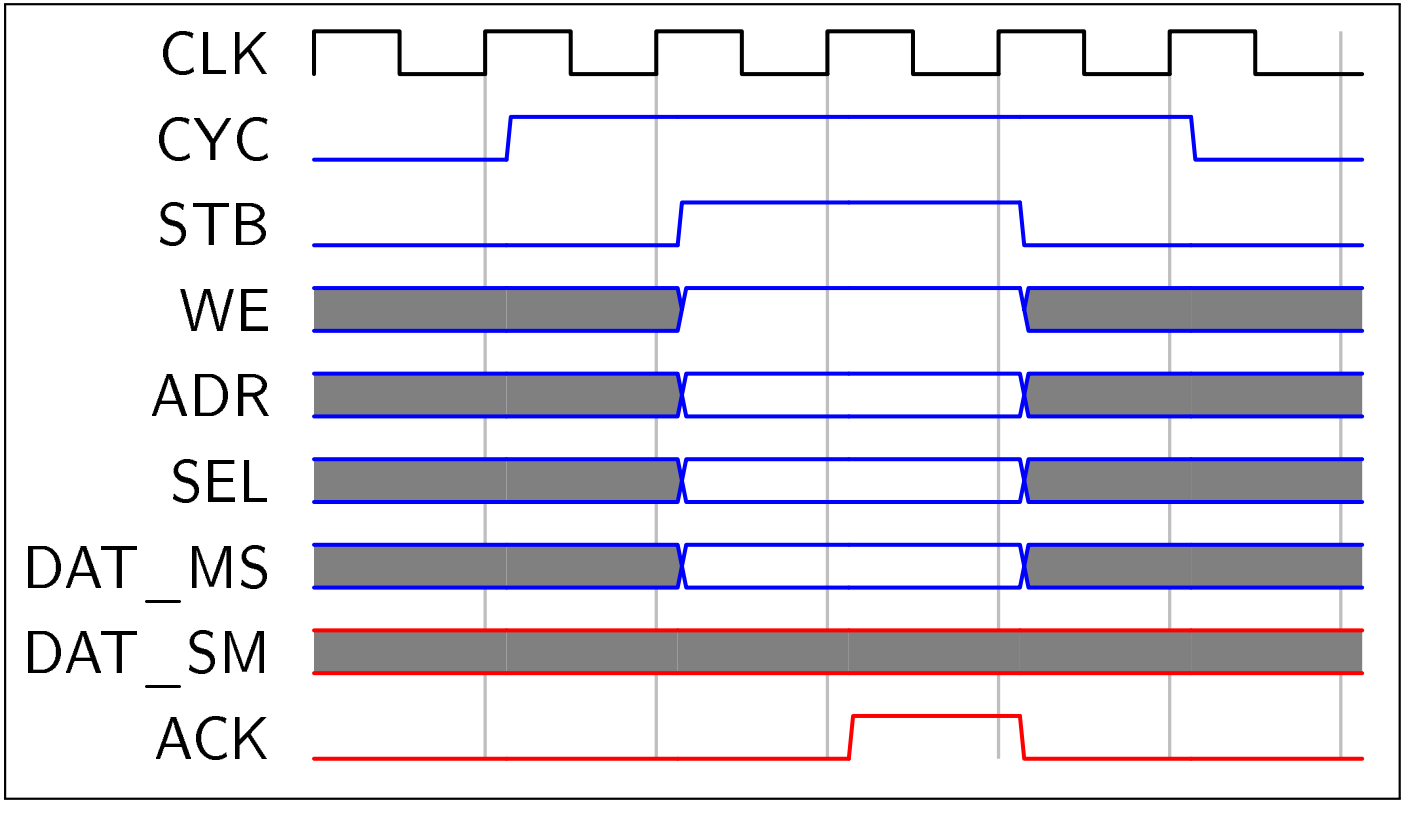
Une transaction en écriture Wishbone

"Classic Bus Cycle"



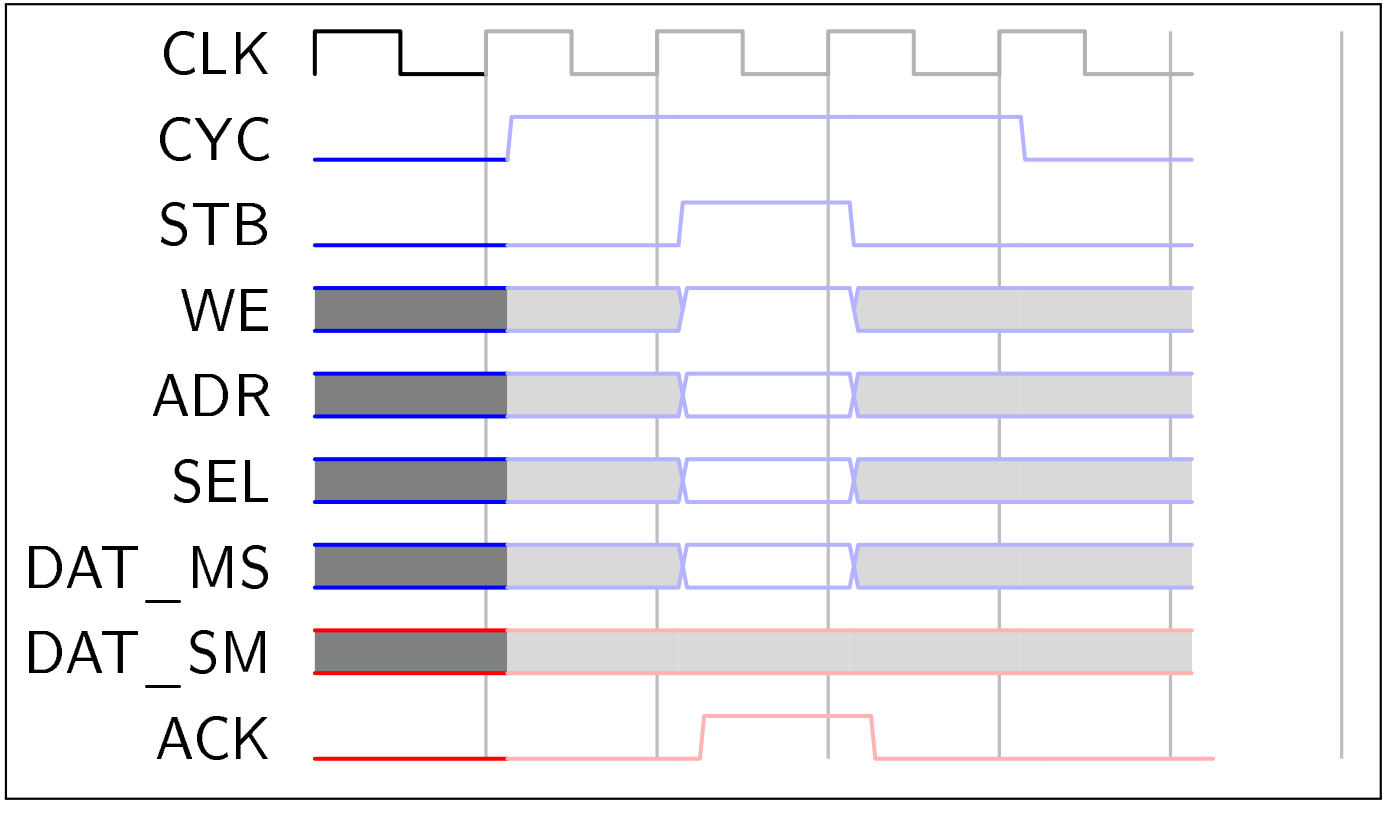
Une transaction en écriture Wishbone

"Classic Bus Cycle"



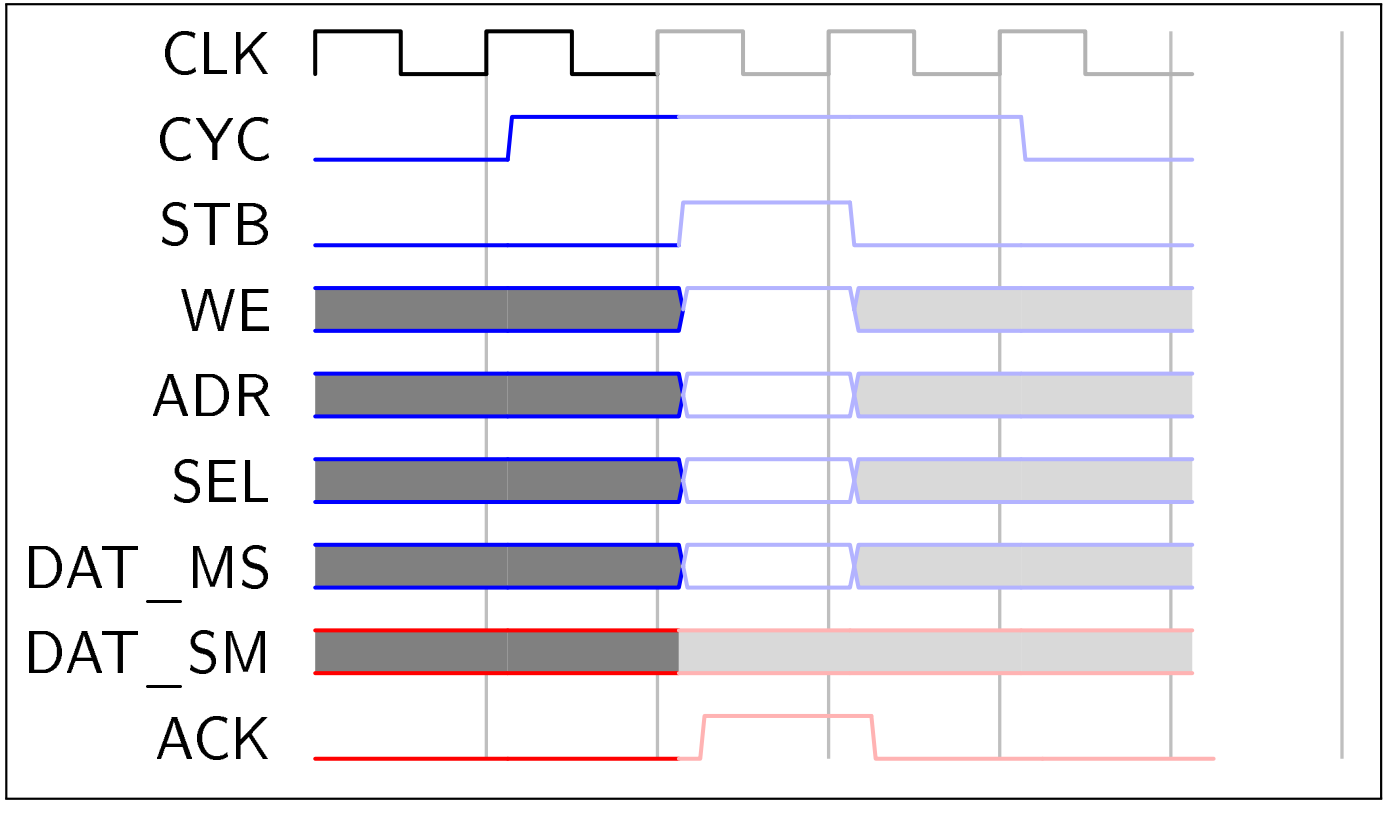
Une transaction en écriture Wishbone

Réponse de l'esclave en "Mealy"



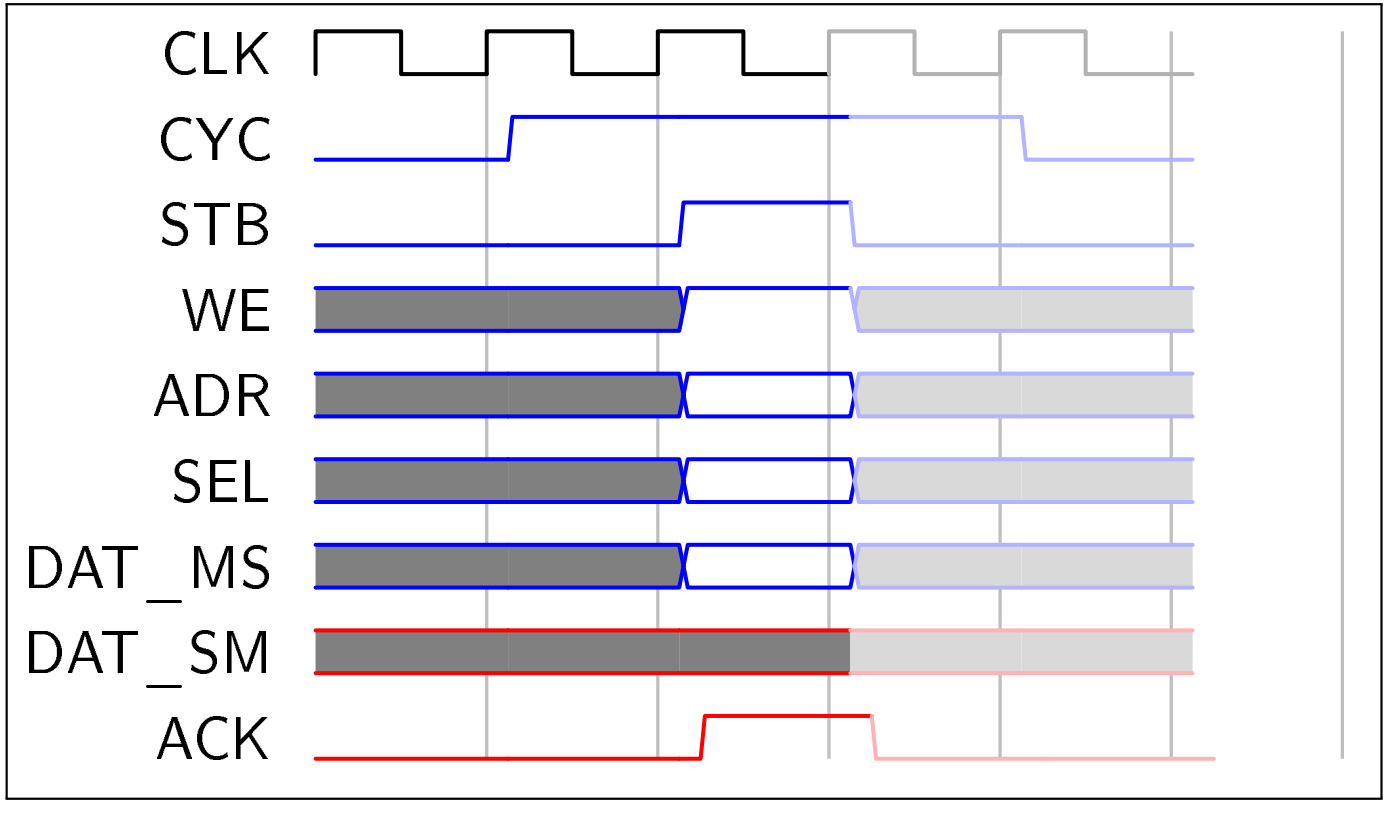
Une transaction en écriture Wishbone

Réponse de l'esclave en "Mealy"



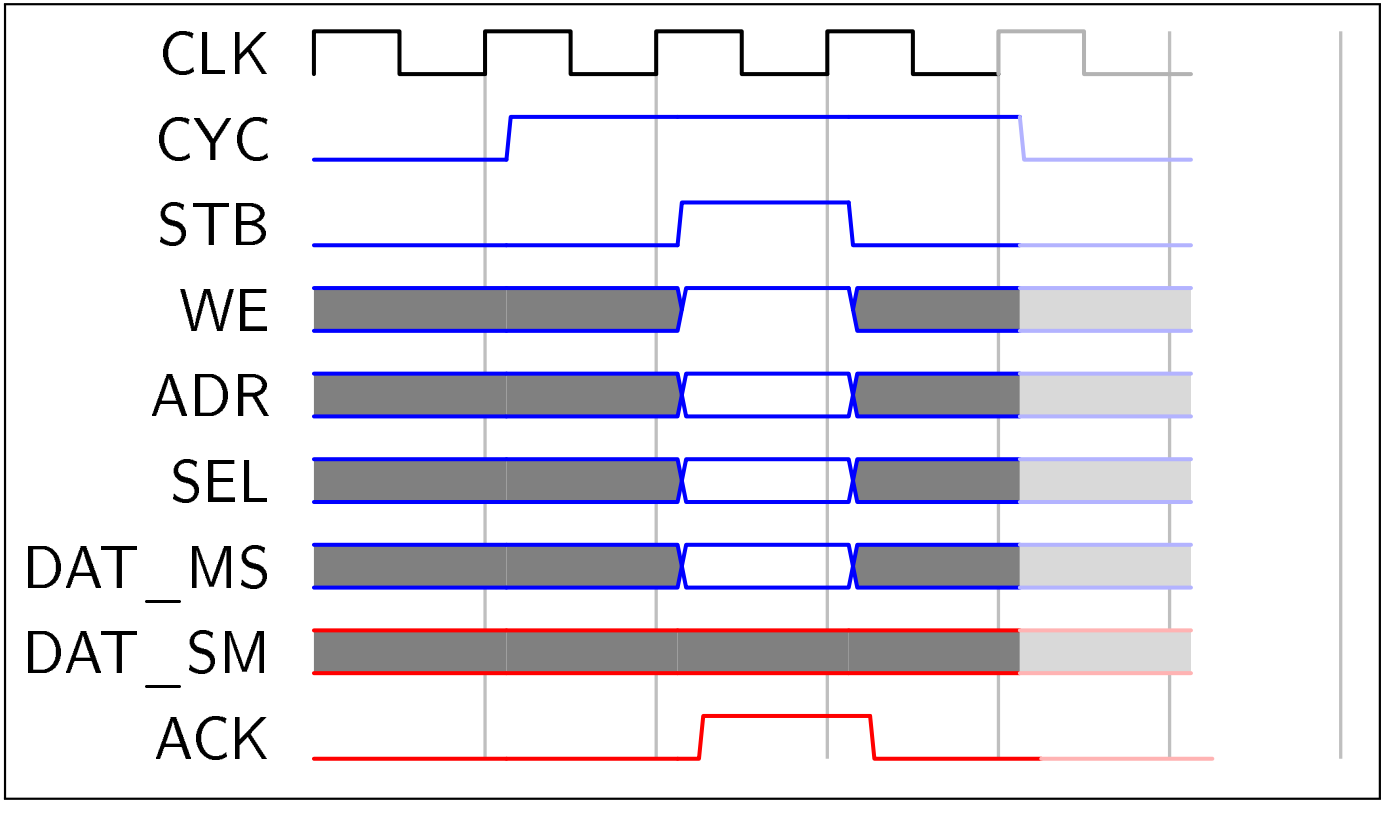
Une transaction en écriture Wishbone

Réponse de l'esclave en "Mealy"



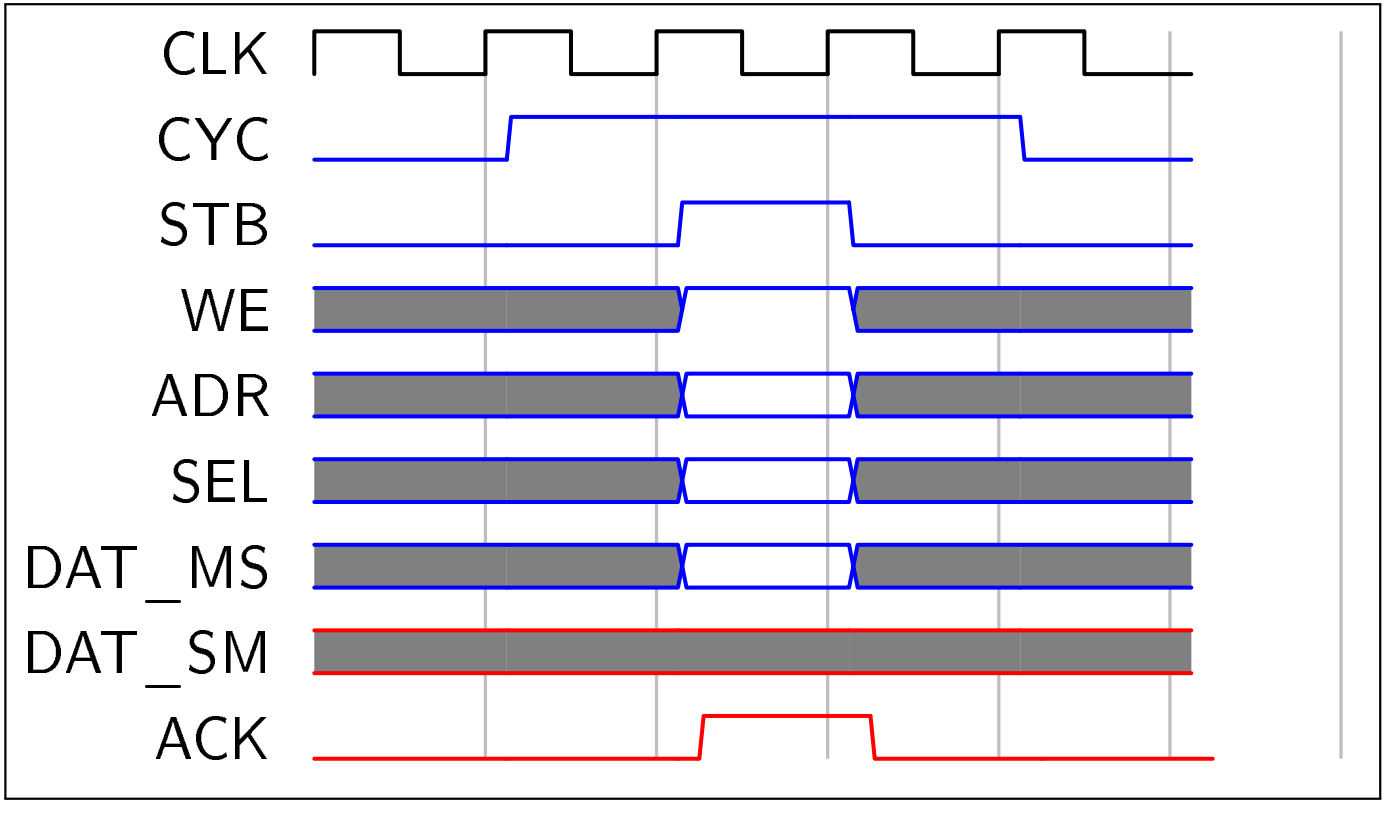
Une transaction en écriture Wishbone

Réponse de l'esclave en "Mealy"



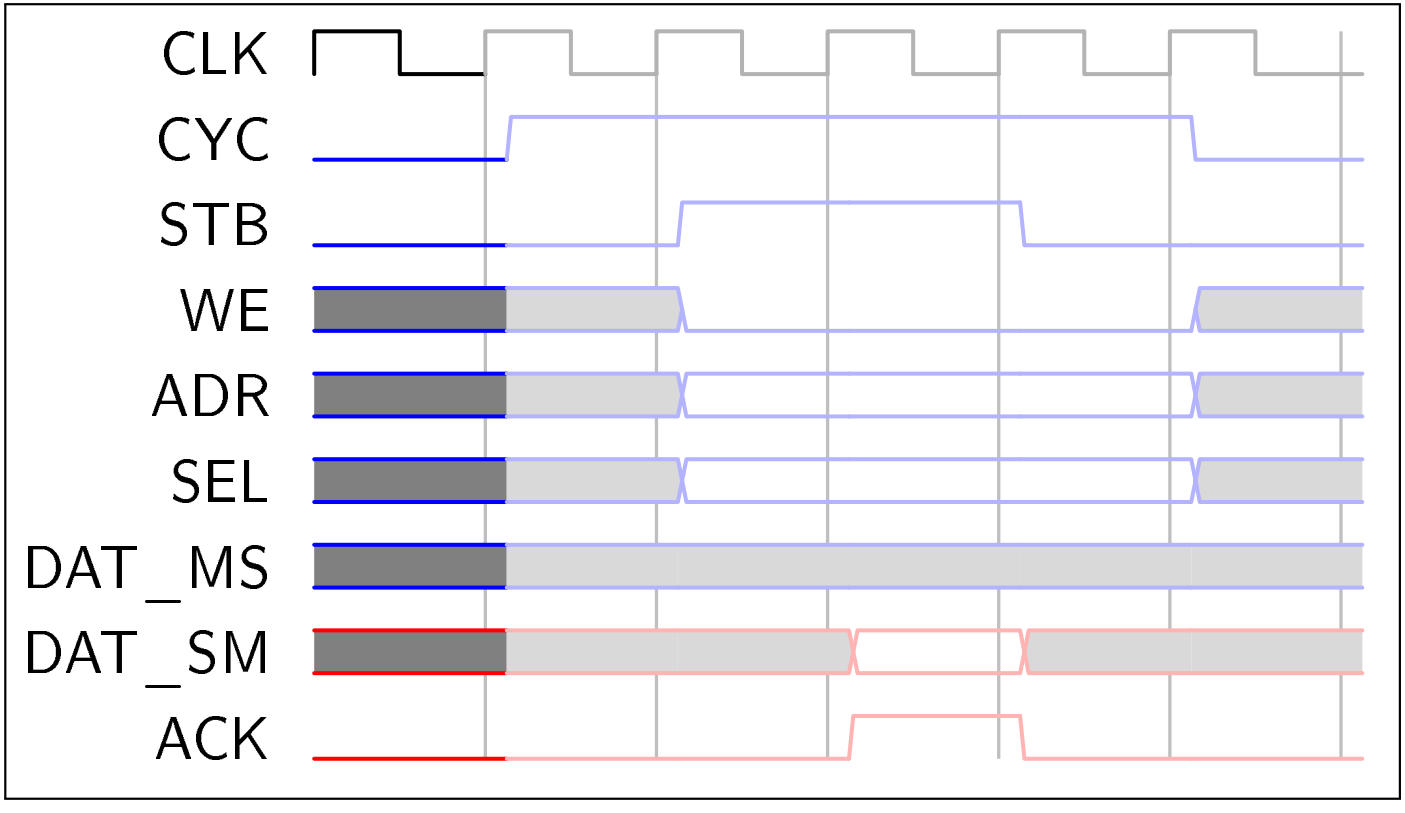
Une transaction en écriture Wishbone

Réponse de l'esclave en "Mealy"



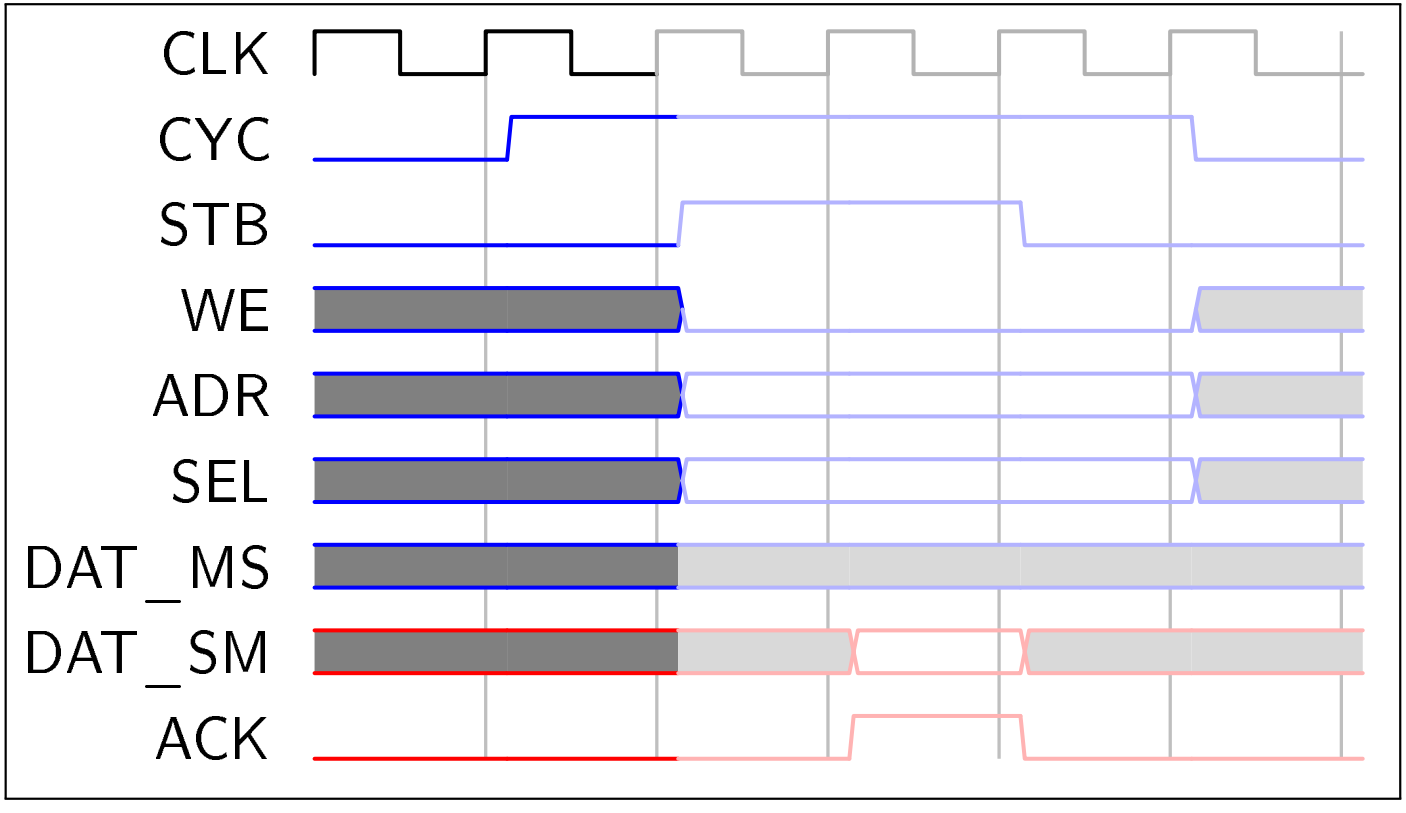
Une transaction en lecture Wishbone

"Classic Bus Cycle"



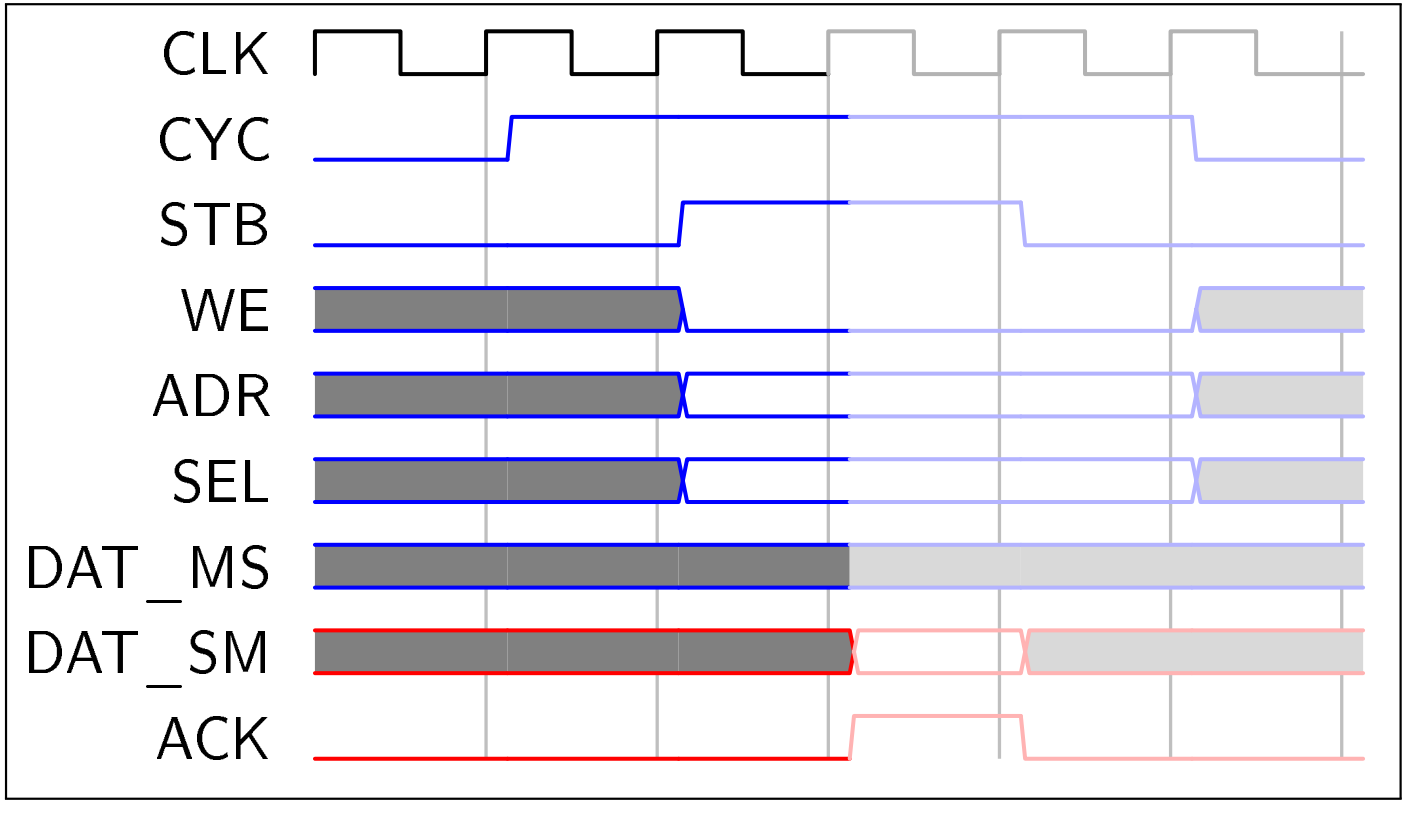
Une transaction en lecture Wishbone

"Classic Bus Cycle"



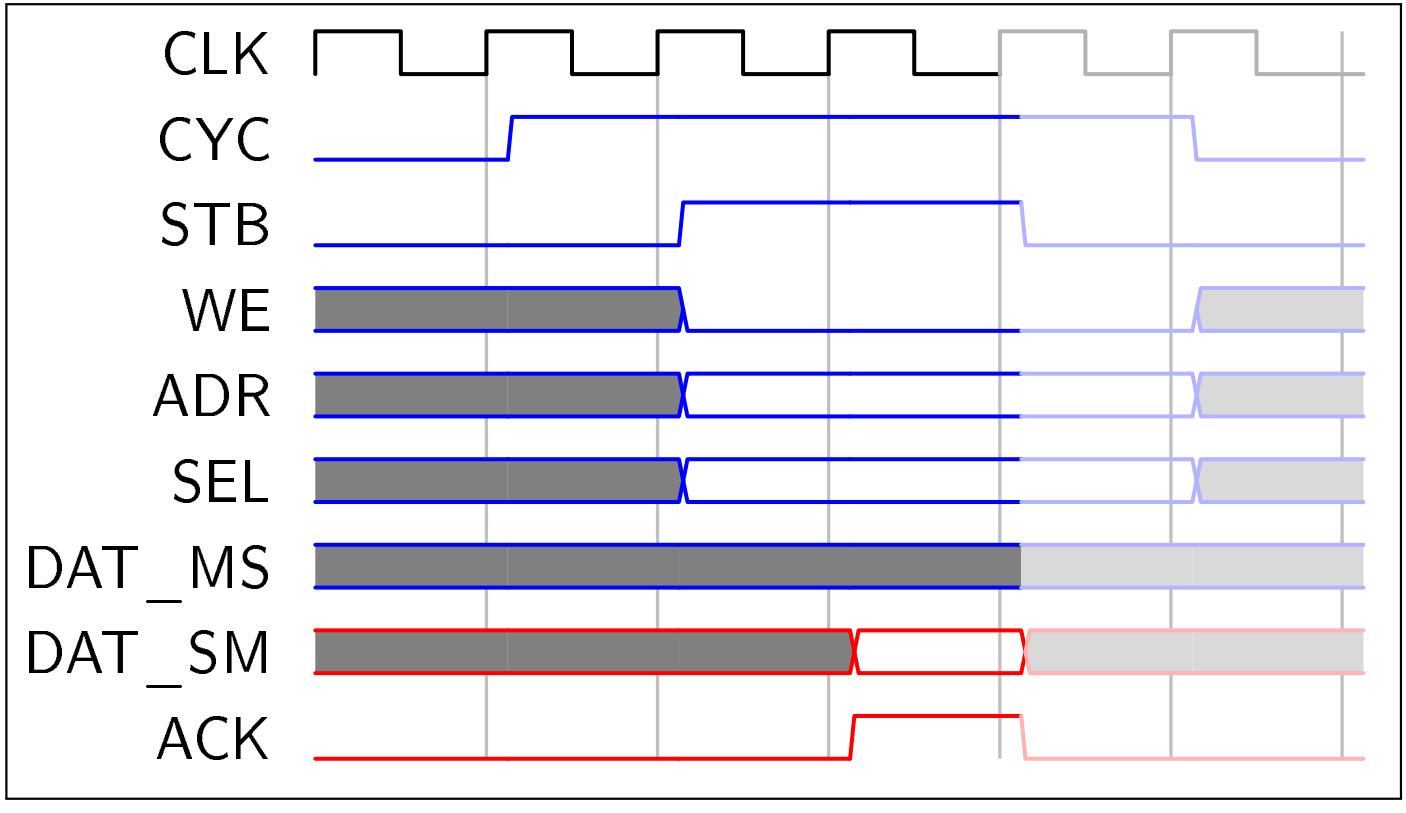
Une transaction en lecture Wishbone

"Classic Bus Cycle"



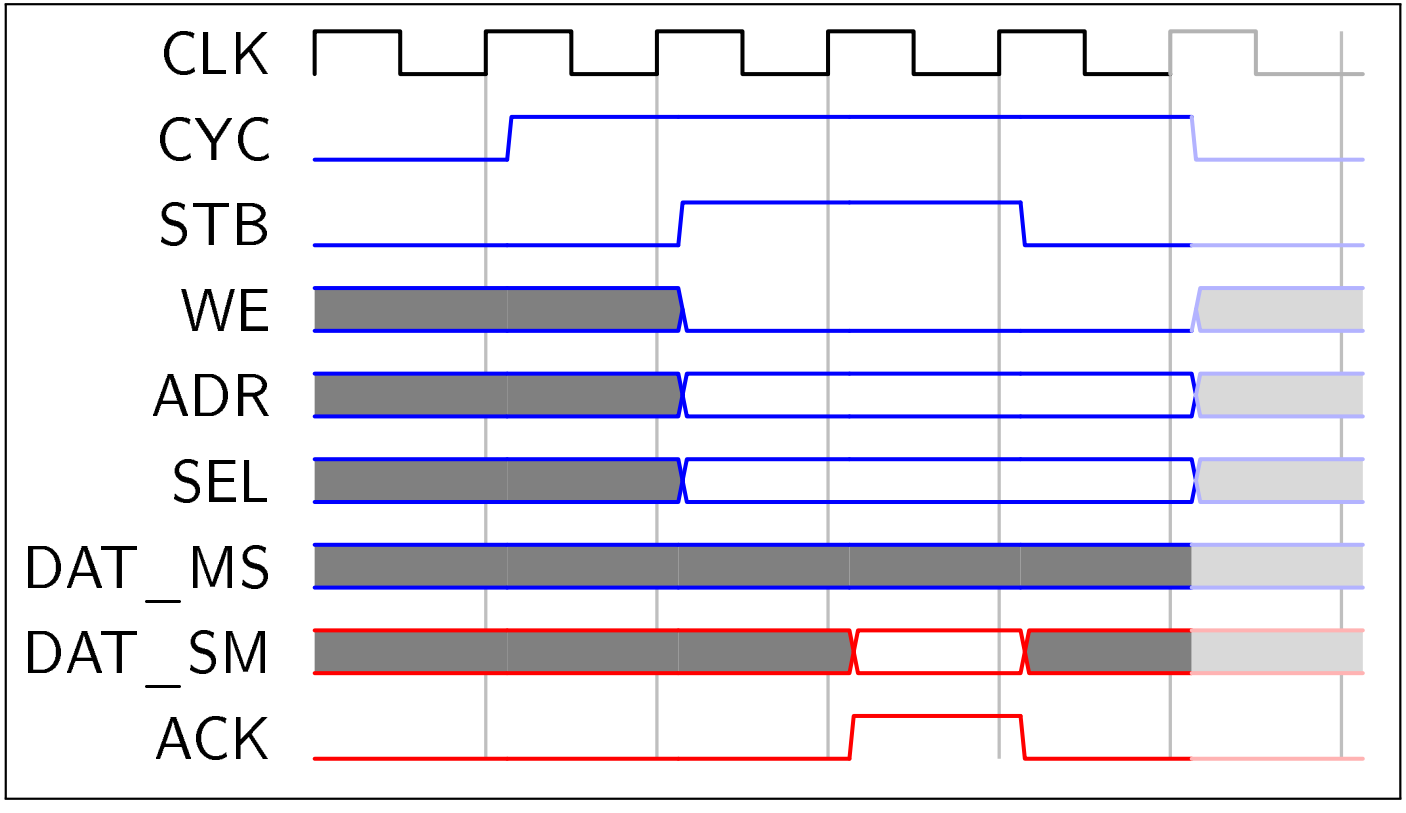
Une transaction en lecture Wishbone

"Classic Bus Cycle"



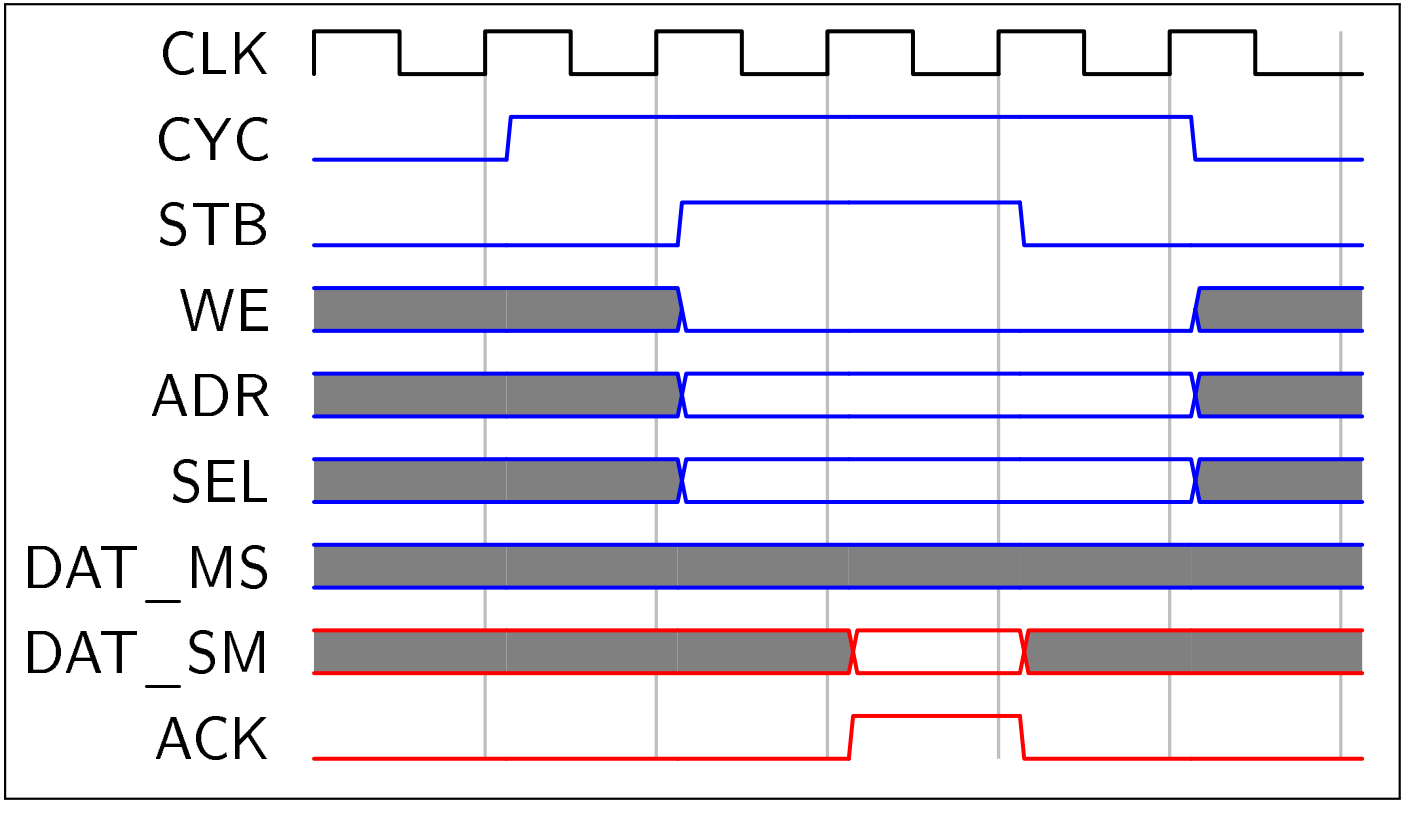
Une transaction en lecture Wishbone

"Classic Bus Cycle"



Une transaction en lecture Wishbone

"Classic Bus Cycle"



Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Les transactions

En résumé...

- Le maître maintient CYC tant qu'il a besoin de l'esclave.
- Le maître initie un cycle de transfert par STB.
- L'esclave détecte la coïncidence de CYC et STB.
- Le maître attend ACK sans relacher la requête.
- L'esclave répond au maître par ACK.
- ACK peut être calculé combinatoirement si l'esclave est suffisament rapide.
- ACK peut être envoyé un ou plusieurs cycles après STB en fonction de la latence de l'esclave.
- Le maître peut enchaîner des requêtes sans interruption (STB maintenu à un, changement de ADR)

Une interface SystemVerilog pour le bus Wishbone

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

```
logic  [31:0]  dat_sm ;
logic  [31:0]  dat_ms ;
logic  [31:0]  adr ;
logic          cyc;
logic  [3:0]   sel;
logic          stb;
logic          we;
logic          ack;
logic          err;
logic          rty;
logic [2:0]    cti;
logic [1:0]    bte;
```

```
modport master (
  output dat_ms,
  output adr ,
  output cyc ,
  output sel ,
  output stb ,
  output we  ,
  output cti ,
  output bte ,
  input  ack ,
  input  err ,
  input  rty ,
  input  dat_sm,
  input  clk,
  input  rst
) ;
```

```
modport slave (
  input dat_ms,
  input adr ,
  input cyc ,
  input sel ,
  input stb ,
  input we  ,
  input cti ,
  input bte ,
  output  ack ,
  output  err ,
  output  rty ,
  output  dat_sm,
  output  clk,
  output  rst
) ;
endinterface
```

Transferts de données en rafales

"Un protocole insuffisant"

- Le protocole tel que décrit n'est pas optimal pour des transferts de paquets de données en rafales.
- La latence de l'esclave (retard de N cycles) est perdue à chaque transfert en lecture:
  - Pour un retard de 1 cycle, la lecture en rafale est 2 fois plus lente que l'écriture.
- Solution:
  - Le maître utilise des TAGS pour prévenir l'esclave d'une requête d'un bloc de données
  - L'esclave peut alors anticiper les requêtes sans attendre le STB du maître.

Transferts de données en rafales

"Un protocole insuffisant"

- Le protocole tel que décrit n'est pas optimal pour des transferts de paquets de données en rafales.
- La latence de l'esclave (retard de N cycles) est perdue à chaque transfert en lecture:
  - Pour un retard de 1 cycle, la lecture en rafale est 2 fois plus lente que l'écriture.
- Solution:
  - Le maître utilise des TAGS pour prévenir l'esclave d'une requête d'un bloc de données
  - L'esclave peut alors anticiper les requêtes sans attendre le STB du maître.

Transferts de données en rafales

"Un protocole insuffisant"

- Le protocole tel que décrit n'est pas optimal pour des transferts de paquets de données en rafales.
- La latence de l'esclave (retard de N cycles) est perdue à chaque transfert en lecture:
  - Pour un retard de 1 cycle, la lecture en rafale est 2 fois plus lente que l'écriture.
- Solution:
  - Le maître utilise des TAGS pour prévenir l'esclave d'une requête d'un bloc de données
  - L'esclave peut alors anticiper les requêtes sans attendre le STB du maître.

"Registered FeedBack Cycle"

- signal CTI : Cycle type identifier (3 bits)
  - ’000’ : Classic Cycle
  - ’001’ : Constant address burst cycle
  - ’010’ : Incrementing burst cycle
  - ’...’ : Reserved
  - ’111’ : End of Burst
- signal BTE : Burst Type Extension (2 bits)
  - ’00’ : Linear burst
  - ’..’ : voir norme
- Les TAGs sont fournis et maintenus avec STB.
- Exemple : transférer N données avec un simple incrément d’adresse:
  - Transférer N-1 données avec le TAG (’010’,’00’)
  - Transférer la donnée finale avec le TAG (’111’,’00’)

"Registered FeedBack Cycle"

- signal CTI : Cycle type identifier (3 bits)
  - ’000’ : Classic Cycle
  - ’001’ : Constant address burst cycle
  - ’010’ : Incrementing burst cycle
  - ’...’ : Reserved
  - ’111’ : End of Burst
- signal BTE : Burst Type Extension (2 bits)
  - ’00’ : Linear burst
  - ’..’ : voir norme
- Les TAGs sont fournis et maintenus avec STB.
- Exemple : transférer N données avec un simple incrément d’adresse:
  - Transférer N-1 données avec le TAG (’010’,’00’)
  - Transférer la donnée finale avec le TAG (’111’,’00’)

"Registered FeedBack Cycle"

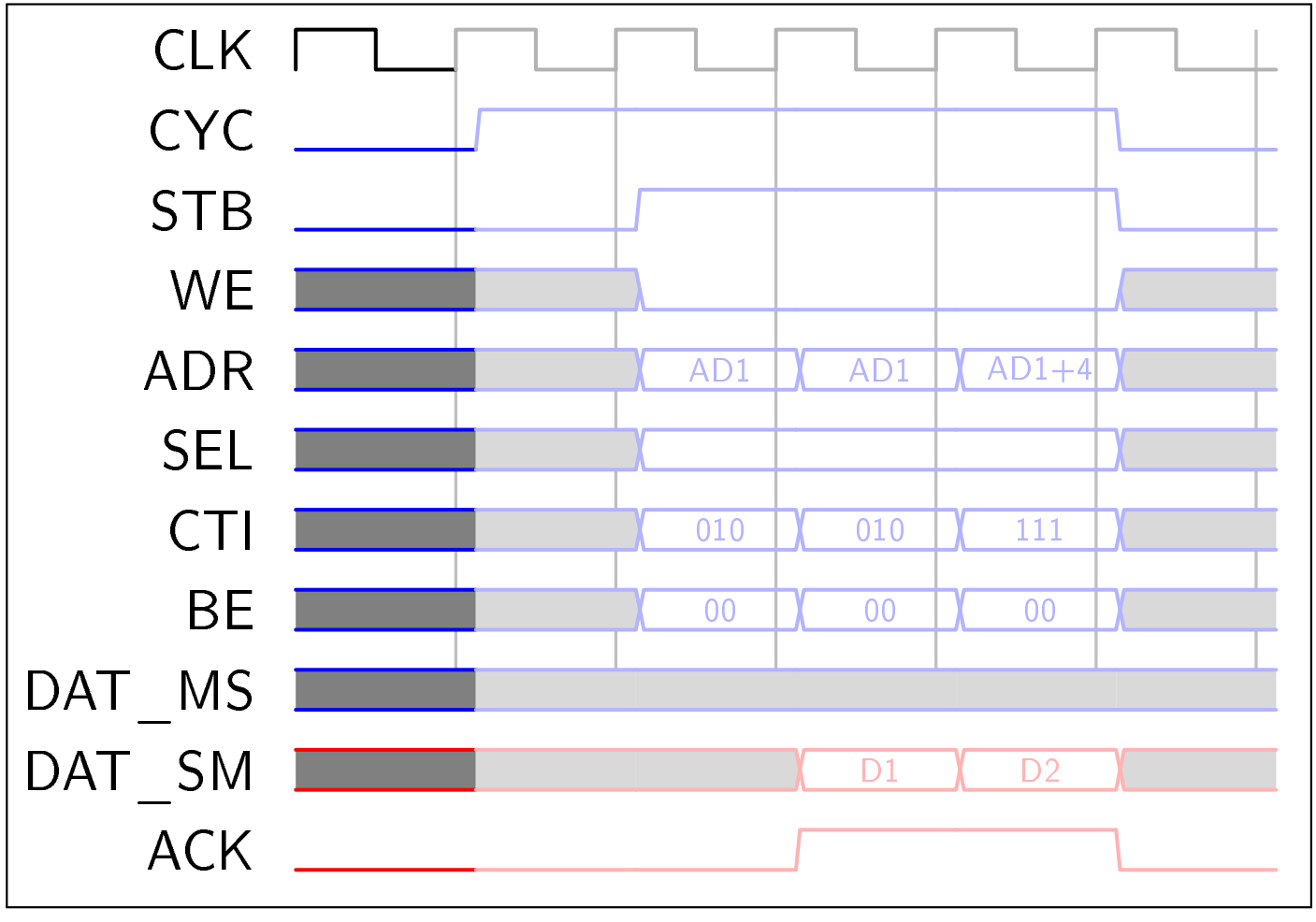
- signal CTI : Cycle type identifier (3 bits)
  - ’000’ : Classic Cycle
  - ’001’ : Constant address burst cycle
  - ’010’ : Incrementing burst cycle
  - ’...’ : Reserved
  - ’111’ : End of Burst
- signal BTE : Burst Type Extension (2 bits)
  - ’00’ : Linear burst
  - ’..’ : voir norme
- Les TAGs sont fournis et maintenus avec STB.
- Exemple : transférer N données avec un simple incrément d’adresse:
  - Transférer N-1 données avec le TAG (’010’,’00’)
  - Transférer la donnée finale avec le TAG (’111’,’00’)

"Registered FeedBack Cycle"

- signal CTI : Cycle type identifier (3 bits)
  - ’000’ : Classic Cycle
  - ’001’ : Constant address burst cycle
  - ’010’ : Incrementing burst cycle
  - ’...’ : Reserved
  - ’111’ : End of Burst
- signal BTE : Burst Type Extension (2 bits)
  - ’00’ : Linear burst
  - ’..’ : voir norme
- Les TAGs sont fournis et maintenus avec STB.
- Exemple : transférer N données avec un simple incrément d’adresse:
  - Transférer N-1 données avec le TAG (’010’,’00’)
  - Transférer la donnée finale avec le TAG (’111’,’00’)

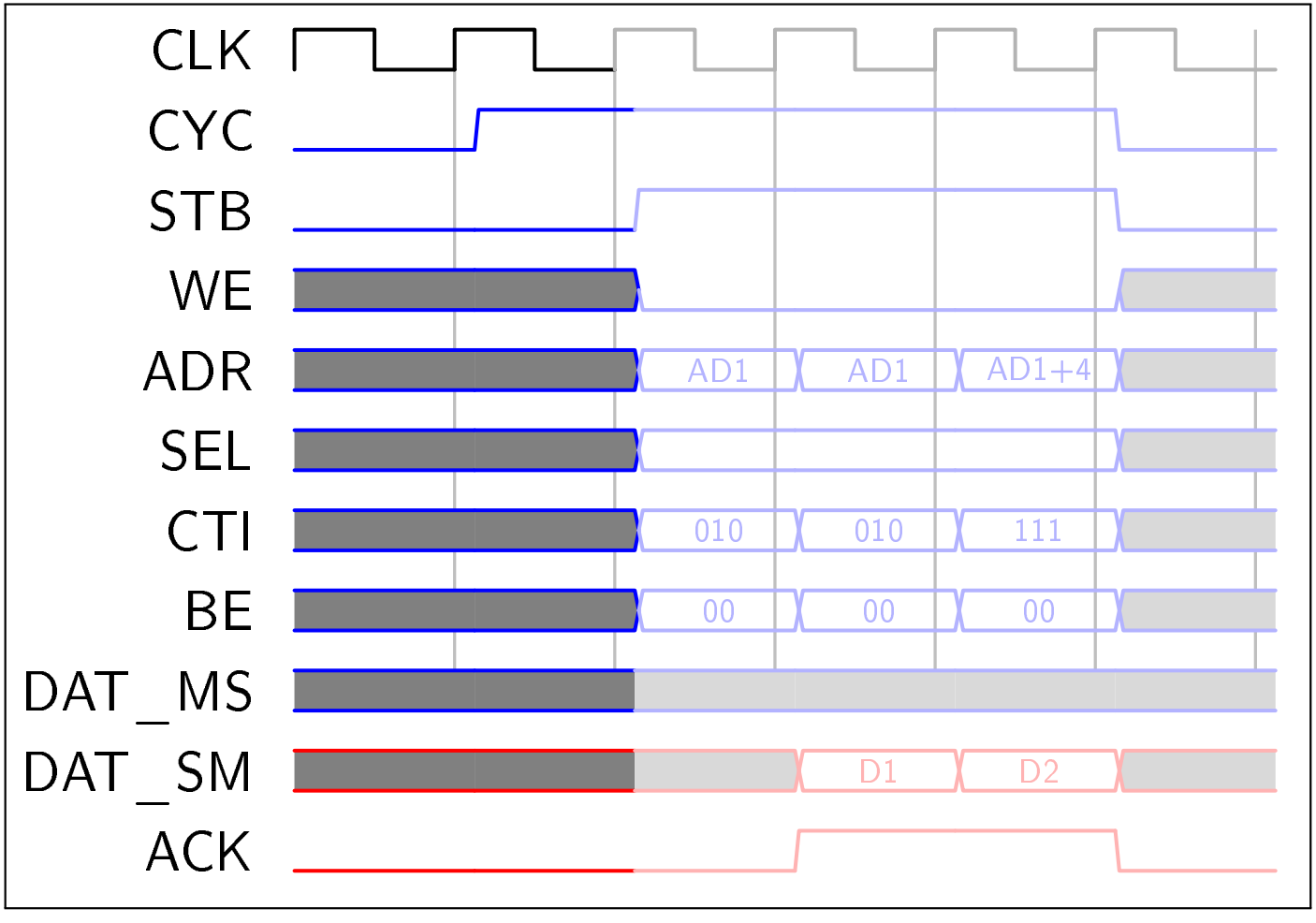
Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



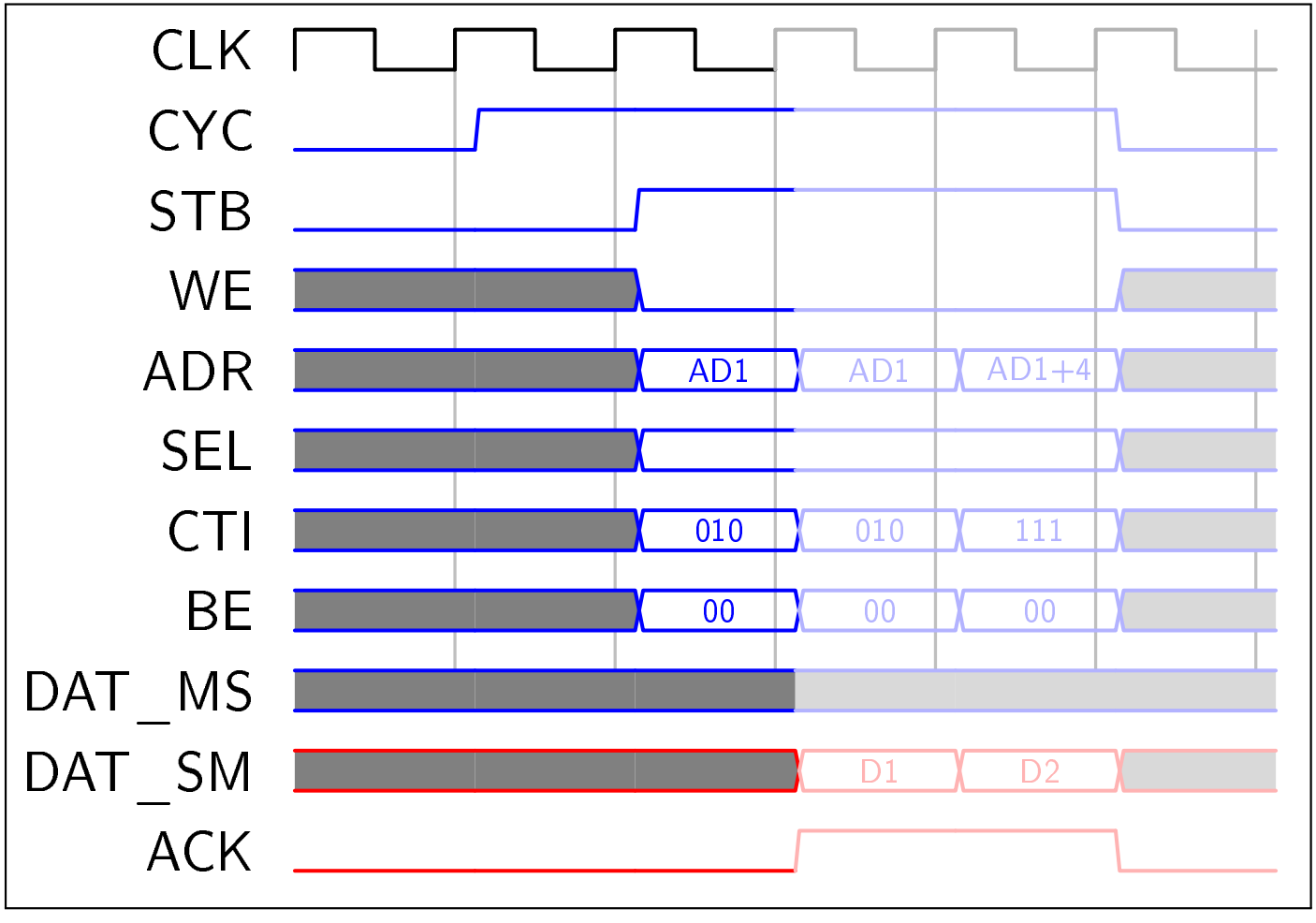
Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



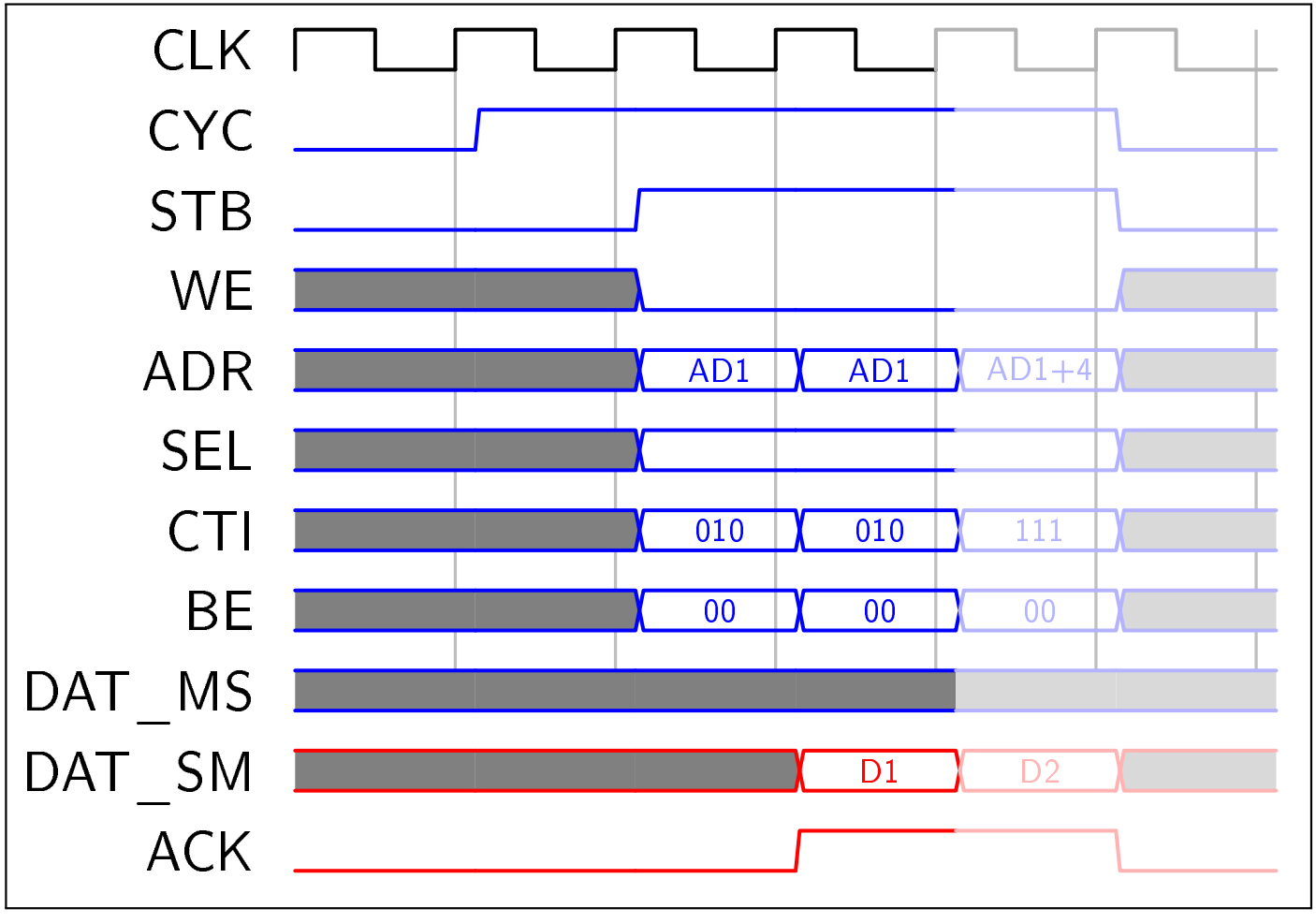
Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



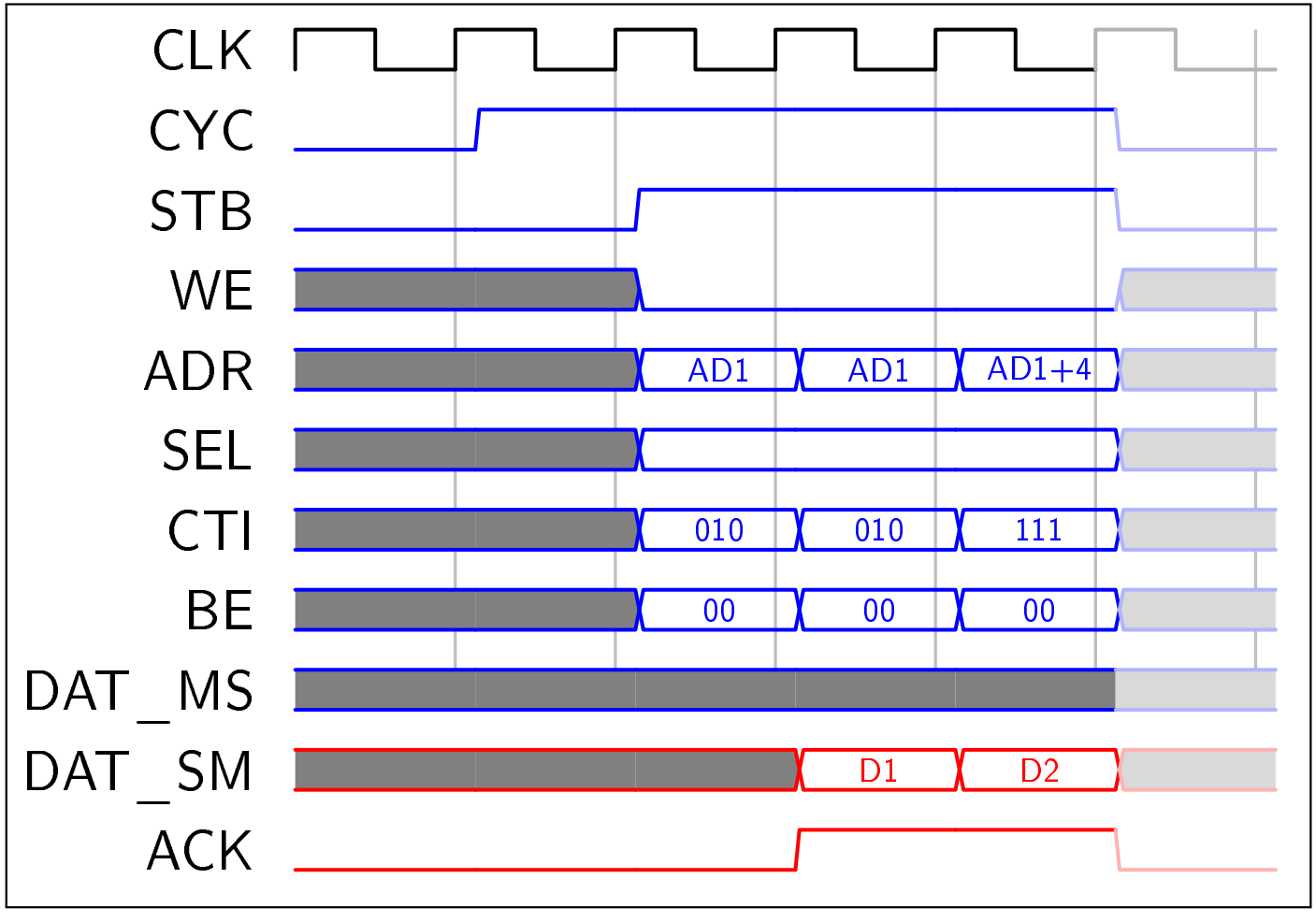
Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



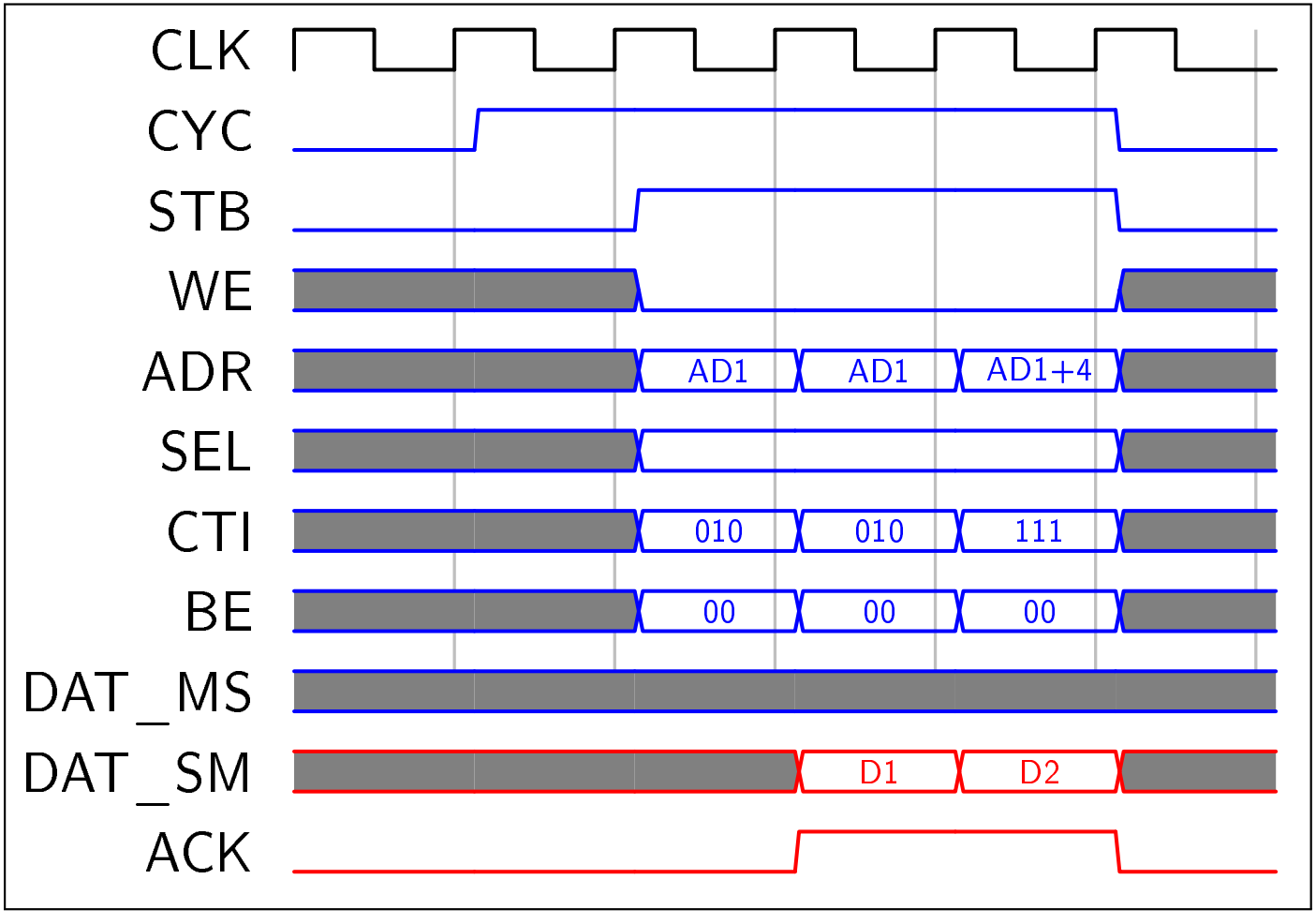
Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



Une rafale en lecture Wishbone

"Registered FeedBack Cycle"



Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Les transactions en rafale

En résumé...

- Tout maître peut utiliser ou non le mode Registered Feedback.
- Tout esclave peut supporter ou non le mode Registered Feedback.
- Tout type d'esclave doit pouvoir être connecté à tout type de maître.
- Même si le mode est "Incremental Burst" le maître doit incrémenter les adresses.
- En mode burst, le masque SEL doit rester constant...
- Un esclave en mode burst peut envoyer le ACK avant même la requête reçue.
- Dans ce cas il maintient tout ses signaux dans l'attente de la requête correspondante.
- Un esclave supportant le mode "Incremental Burst" doit disposer de son propre compteur d'adresses.
- Utile pour de la simple communication point à point en mode pipeline.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(1) Un contrôleur de mémoire à bus Wishbone

- Coder un **esclave** Wishbone 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.
- L'acquittement de l'écriture se fera de manière **combinatoire**.
- La mémoire étant synchrone, l'acquittement de la lecture aura **un cycle de retard**.
- 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 Wishbone.
- Le code devra être évidemment synthétisable.
- Nous fournissons (voir site SE204)
  - La définition de l'interface
  - Un programme de test faisant des requêtes aléatoires sur le bus.

Au travail...

(2) Un contrôleur de mémoire avec lecture en rafale

- Réalisez une nouvelle version de votre contrôleur de mémoire avec support du mode rafale en lecture.
- Le maître du testbench génère à la fois des cycles classiques, et des cycles «registered feedback ».
- N'oubliez pas de vérifier l'amélioration de vitesse en lecture.

Au travail...

(2) Un contrôleur de mémoire avec lecture en rafale

- Réalisez une nouvelle version de votre contrôleur de mémoire avec support du mode rafale en lecture.
- Le maître du testbench génère à la fois des cycles classiques, et des cycles «registered feedback ».
- N'oubliez pas de vérifier l'amélioration de vitesse en lecture.

Au travail. . .

(2) Un contrôleur de mémoire avec lecture en rafale

- Réalisez une nouvelle version de votre contrôleur de mémoire avec support du mode rafale en lecture.
- Le maître du testbench génère à la fois des cycles classiques, et des cycles «registered feedback ».
- N'oubliez pas de vérifier l'amélioration de vitesse en lecture.