Corneliu ZAHARIA

Rețele Integrate în Chip Protocoale

Editura Universității Transilvania din Brașov

2026

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr.
500091 Brașov
Tel.: 0268 476 050
Fax: 0268 476 051
E-mail: editura@unitbv.ro

Editură recunoscută CNCSIS, cod 81

ISBN 978-606-19-1880-5 (e-book)

Copyright © Autorul, 2026

Lucrarea a fost avizată de Consiliul Departamentului de Electronică și Calculatoare, Facultatea de Inginerie electrică și știința calculatoarelor a Universității Transilvania din Brașov.

CUPRINS

PREFAȚĂ 3

1. APB (Advanced Pheripheral Bus) 5

2. AHB (Advanced High-performance Bus) 23

3. AXI (Advanced eXtensible Interface) 55

4. ACE (AXI Coherency Extensions) 71

5. Ready-Valid 193

6. Request-Acknowledge (Cerere-Confirmare) cu 2 faze 200

7. Request-Acknowledge (Cerere-Confirmare) cu 4 faze 205

8. Data Stream (protocolul Flux de Date) 210

9. Protocolul de Interfață Imagine Pixel 214

10. Protocolul Push / Pop 228

11. Protocolul Stărilor (”Status Protocol”) 245

REFERINȚE 249

PREFAȚĂ

Această lucrare are un preponderent scop didactic, în domeniul sistemelor integrate ”în chip” SoC (System-on-Chip) complexe, sub-domeniul *rețelelor integrate în chip – RIC* (NoC - Network-On-Chip).

Studenții – mai ales cei de la programele de studii de master – dar și inginerii, pot valorifica în mod aplicativ o serie de cunoștințe (dobândite la studiile de licență), concepte și metode de bază privitoare la sistemele de calcul, microprocesoare, micro controlere, limbaje și tehnici de programare – inclusiv Limbaje de Descriere Hardware (HDL).

Principalii destinatari sunt studenții din domeniul de studii Inginerie Electronică, Telecomunicații şi Tehnologii Informaționale, de la programul de studii universitare de master Sisteme Electronice și de Comunicații Integrate. Pentru înțelegerea materiei prezentate, sunt necesare cunoștințe temeinice de electronică digitală (fiind foarte utilă și cunoașterea limbajelor de descriere hardware). Pe aceste baze, conceptele și metodele de bază privitoare la arhitectura sistemelor de calcul, microprocesoare, micro controlere, limbaje și tehnici de programare vor fi dezvoltate în competențe de implementare practică a rețelelor pe chip pentru proiectarea System-On-Chip (SOC). Rezultatele învățării vor consolida managementul complexității – mai ales privind protocoalele și standardele din domeniu, tehnicile de control şi interfațare. Materia e structurată începând cu protocoalele de nivel sistem – mai ușor de înțeles întrucât corespund interfețelor de comunicații. Celelalte protocoale prezentate în lucrare – cele de semnalizare (nivel tranzacție) și cele orientate pe fluxuri de date (nivel aplicație) – au un caracter mai specializat.

Managementul complexității RIC vizează – pe baza standardelor din domeniul microelectronicii – **protocoale** și **tehnici de control şi interfațare**. Acestea ocupă un loc central în studiul RIC, aspectele micro-electronice fiind quasi-complementare.

Din acest motiv, lucrarea de față tratează următoarea *taxonomie a protocoalelor RIC:*

PROTOCOALE DE <u>NIVEL SISTEM</u>

La nivel de magistrală integrată în chip (protocoale de Bus), AMBA (Advanced Microcontroller Bus Architectures – inițiate de ARM în anii ’90) corespund interfețelor de comunicații.

Cele mai importante sunt:

1. APB (Advanced Peripheral Bus)
2. AHB (Advanced High-performance Bus)
3. AXI (Advanced eXtensible Interface)
4. ACE (AXI Coherency Extensions)

PROTOCOALE DE SEMNALIZARE. MODELE DE ”HANDSHAKE” (NIVEL TRANZACȚIE)

5. Ready-Valid (”Disponibil-Valid”)
6. Request-Acknowledge (2 faze) – ”Cerere-Confirmare”
7. Request-Acknowledge (4 faze)

PROTOCOALE ORIENTATE PE FLUXURI DE DATE (NIVEL APLICAȚIE) - ”IP-block”

8. Data Stream Interface (”Interfață pentru fluxul de date”)
9. Image Pixel Interface (”Interfață pentru fluxuri video”)
10. Push-Pop

PROTOCOALE DE STARE ȘI SEMNALIZARE SUPLIMENTARĂ

11. Status Protocol

Pentru înțelegerea completă a modului de funcționare a protocoalelor s-a folosit utilitarul gratuit *Wavedrom* care permite exprimarea grafică a dependențelor temporale ale semnalelor folosite. Au fost incluse scripturile necesare pentru generarea formelor de undă pentru a permite studenților exersarea diferitelor scenarii de funcționare și analiza erorilor în comunicație.

Dintre referințele bibliografice, remarcăm monografiile [1], [2] și [3] – referirile din text sunt detaliate, cu trimiteri și la paginile din toate surse bibliografice.

1. APB (Advanced Pheripheral Bus)

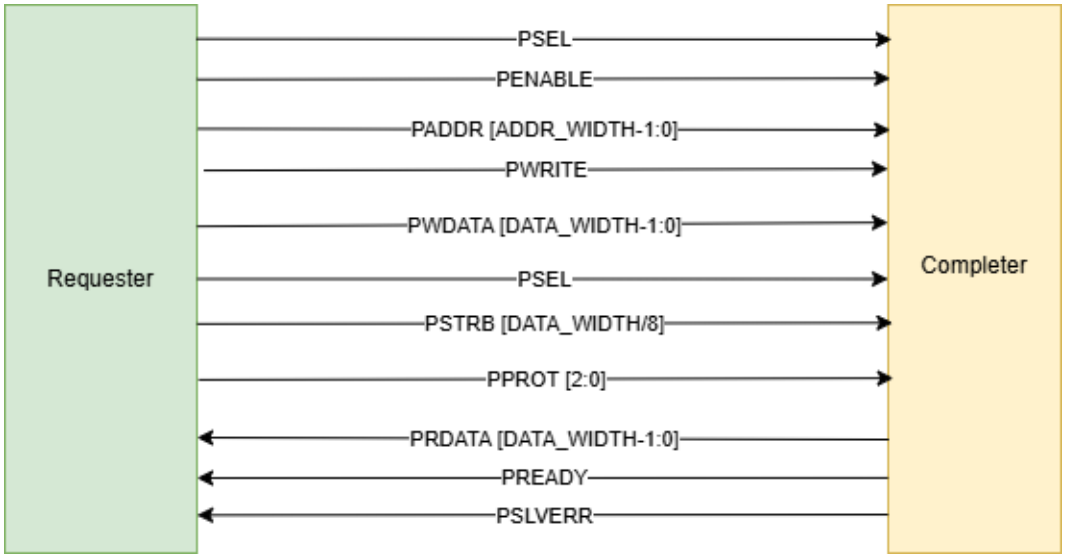
Protocolul AMBA APB corespunde unei interfețe cu costuri reduse - optimizată pentru un consum minim de energie și o complexitate minimală [1, pag. 23]. Interfața APB nu operează segmentat (”pipeline” / ”conductă”) ci funcționează după un protocol simplu, sincron. Fiecare transfer durează cel puțin două cicluri. Interfața APB este proiectată pentru a accesa grupuri de registre de control programabile – grupuri tipice pentru dispozitivele periferice mai lente [8, pag. 1]. Perifericele APB sunt de obicei conectate la memoria principală printr-o ”punte” (”bridge”) APB. De exemplu, o punte de la AXI ↔ APB ar putea fi folosită pentru a conecta mai multe periferice APB la un sistem de memorie AXI.

Transferurile APB sunt inițiate printr-o punte APB. O punte APB poate fi considerată *Solicitant.* O interfață periferică răspunde solicitărilor (”cererilor”) și se poate considera *Furnizor* [4, pag.362].

Descrierea semnalelor interfeței APB

Nume	Sursa	Lățime	Descriere
PCKL	Ceas	1	**PCLK** este un semnal de ceas. Toate semnalele APB sunt sincronizate cu frontul urcător al PCLK.
PRESETn	Reset bus sistem	1	**PRESETn** este semnalul de resetare și este **activ** pe valoarea ”0”**.** **PRESETn** este de obicei conectat direct la semnalul de resetare al magistralei de sistem.
PADDR	Solicitant	ADDR_WIDTH	**PADDR** este magistrala de adrese APB. **PADDR** poate avea o lățime de până la 32 de biți.
PPROT	Solicitant	3	**PPROT** indică tipul de acces al tranzacției (acces la date sau la instrucțiuni) și tipul de protecție (normal, privilegiat sau securizat) al tranzacției.
PNSE	Solicitant	1	Extensie pentru tipul de protecție.
PSELx	Solicitant	1	Selecție. Solicitantul generează un semnal **PSELx** pentru fiecare Furnizor. **PSELx** arată că

Nume	Sursa	Lățime	Descriere
			Furnizorul este selectat și că e necesar un transfer de date.
PENABLE	Solicitant	1	Activare. **PENABLE** indică al doilea și următoarele cicluri ale unui transfer APB.
PWRITE	Solicitant	1	Direcția. **PWRITE** indică un acces **APB WRITE** când este ”1” și un acces **APB READ** când este ”0”.
PWDATA	Solicitant	DATA_WIDTH	Scriere date. **Magistrala pentru date de scriere PWDATA este preluată de puntea APB ca Solicitant, în timpul ciclurilor de scriere când** PWRITE **este ”1”. PWDATA** poate avea lățimea de 8, 16 sau 32 de biți.
PSTRB	Solicitant	DATA_WIDTH/8	Impuls de scriere (”strobe”). **PSTRB** indică ce grupuri de câte 8 linii - pentru câte un octet - trebuie actualizate în timpul unui transfer de scriere (există câte un impuls de scriere pentru fiecare 8 biți din magistrala pentru date de scriere). **PSTRB**[n] corespunde PWDATA[(8n + 7):(8n)]. **PSTRB** nu trebuie să fie activ în timpul unui transfer de citire.
PREADY	Furnizor	1	Pregătit. **PREADY** este folosit pentru a extinde un transfer APB de către Furnizor
PRDATA	Furnizor	DATA_WIDTH	Citește datele. Magistrala pentru date de citire PRDATA este preluată de Furnizorul selectat în timpul ciclurilor de citire când PWRITE este ”0”. **PRDATA** poate avea lățimea de 8, 16 sau 32 de biți.
PSLVERR	Furnizor	1	Eroare de transfer. **PSLVERR** este un semnal opțional care poate fi activat de către Furnizor: ”1” indică eroare la un transfer APB.
PWAKEUP	Solicitant	1	Activare. **PWAKEUP** indică orice tip de activitate asociată cu interfața APB.
PAUSER	Solicitant	USER_REQ_WIDTH	Atributul cererii de la utilizator. **PAUSER** este recomandat să aibă o lățime maximă de 128 de biți.
PWUSER	Solicitant	USER_DATA_WIDTH	Atributul datelor scrise de către utilizator. **PWUSER** este recomandat să aibă o lățime maximă de DATA_WIDTH/2.
PRUSER	Furnizor	USER_DATA_WIDTH	Atributul datelor citite de utilizator. **PRUSER** este recomandat să aibă o lățime maximă de DATA_WIDTH/2.
PBUSER	Furnizor	USER_RESP_WIDTH	Atributul răspunsului dat de utilizator. **Se recomandă ca PBUSER** să aibă o lățime maximă de 16 biți.

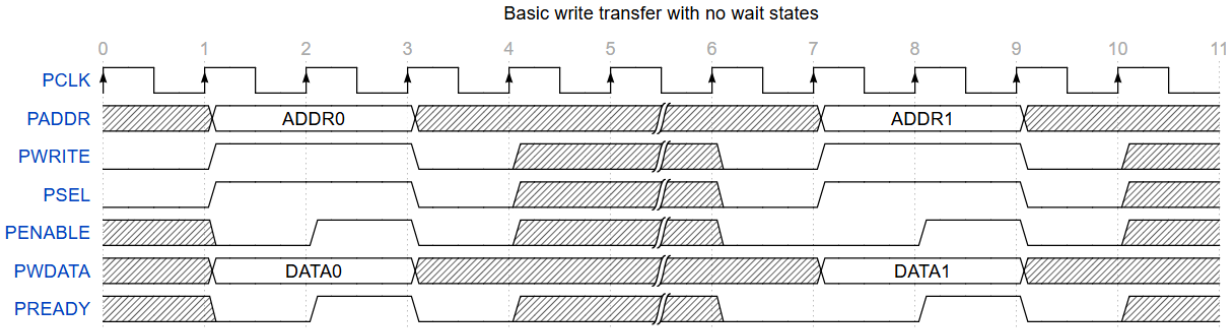


Transferuri de Scriere

Secțiunea următoare definește un transfer de scriere:

- Fără stări de așteptare (cu întârziere).
- Cu stări de așteptare (fără întârziere).

Toate semnalele prezentate în această secțiune sunt eșantionate pe frontul urcător al **PCLK**



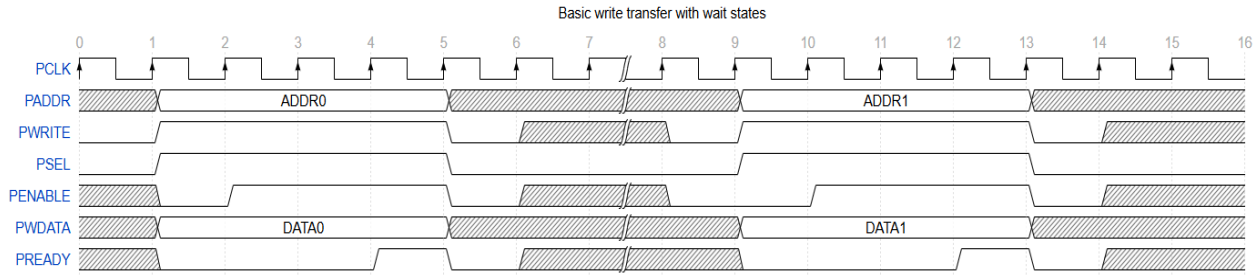

```
{signal: [
    {name: 'PCLK', wave: 'P.........'},
    {name: 'PADDR', wave: 'x2.xx|x2.xx', data: ['ADDR0', 'ADDR1']},
    {name: 'PWRITE', wave: '01.0x|01.0x'},
```

```
    {name: 'PSEL', wave: '01.0x|01.0x'},
    {name: 'PENABLE', wave: 'x010x|0.10x'},
    {name: 'PWDATA', wave: 'x2.xx|x2.xx', data: ['DATA0', 'DATA1',]},
    {name: 'PREADY', wave: 'x010x|0.10x'},
  ],
  config: { hscale: 2 },
    head:{
     text:'Basic write transfer with no wait states',
     tick:0,
     every:1  },  }
```

Faza de configurare a transferului de scriere are loc la ciclul 1. Semnalul select, PSEL, este activat (setat), ceea ce înseamnă că PADDR, PWRITE și PWDATA trebuie să fie valide.

Faza de acces a transferului de scriere este prezentată în ciclul 2, unde PENABLE este activ (setat). PREADY este activat (setat) de Furnizor pe frontul ascendent al PCLK, pentru a indica faptul că datele de scriere vor fi acceptate în ciclul 3. PADDR, PWDATA și orice alte semnale de control trebuie să fie stabile până la finalizarea transferului.

La finalul transferului, PENABLE este dezactivat (resetat). PSEL este de asemenea dezactivat, cu excepția cazului în care există un alt transfer către același periferic.




```
{signal: [
  {name: 'PCLK', wave: 'P......|........'},
  {name: 'PADDR', wave: 'x2...x.|x2...x..', data: ['ADDR0', 'ADDR1']},
  {name: 'PWRITE', wave: '01...0x|01...0x.'},
  {name: 'PSEL', wave: '01...0...1...0..'},
  {name: 'PENABLE', wave: 'x01..0x|0.1..0x.'},
  {name: 'PWDATA', wave: 'x2...x.|x2...x..', data: ['DATA0', 'DATA1',]},
  {name: 'PREADY', wave: 'x0..10x|x0..10x.'},
  ], config: { hscale: 2 },
  head:{
```

```
  text:'Basic write transfer with wait states',
  tick:0,
   every:1 }, }
```

În timpul unei faze de acces, când PENABLE este “1”, Furnizorul extinde transferul stabilind PREADY = “0”. În timp ce PREADY rămâne “0”, următoarele semnale rămân neschimbate:

- Semnalul de adresă, **PADDR**
- Semnalul de direcție, **PWRITE**
- Semnalul de selectare, **PSELx**
- Semnalul de activare, **PENABLE**
- Semnalul de scriere date, **PWDATA**

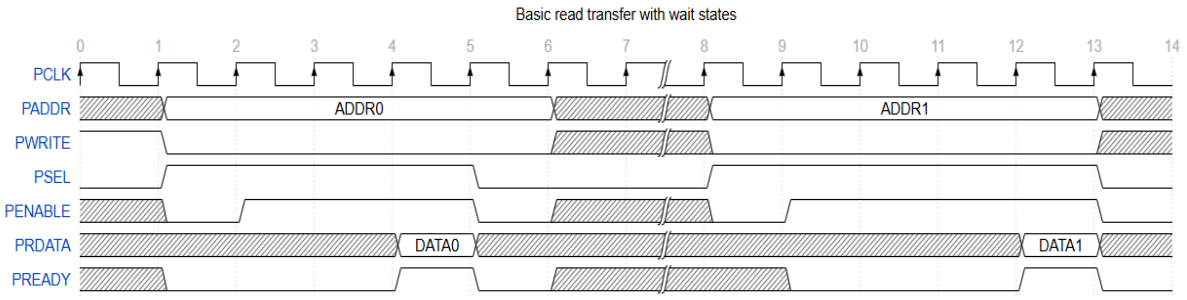
PREADY poate lua orice valoare când **PENABLE** este ”0”. Acest lucru asigură că perifericele care au acces în fix două cicluri pot continua **PREADY** = “1”.

Transferuri de citire

În această secțiune sunt descrise două tipuri de transfer de citire:

- Fără stări de așteptare
- Cu stări de așteptare

Toate semnalele prezentate în această secțiune sunt eșantionate pe frontul urcător al PCLK.



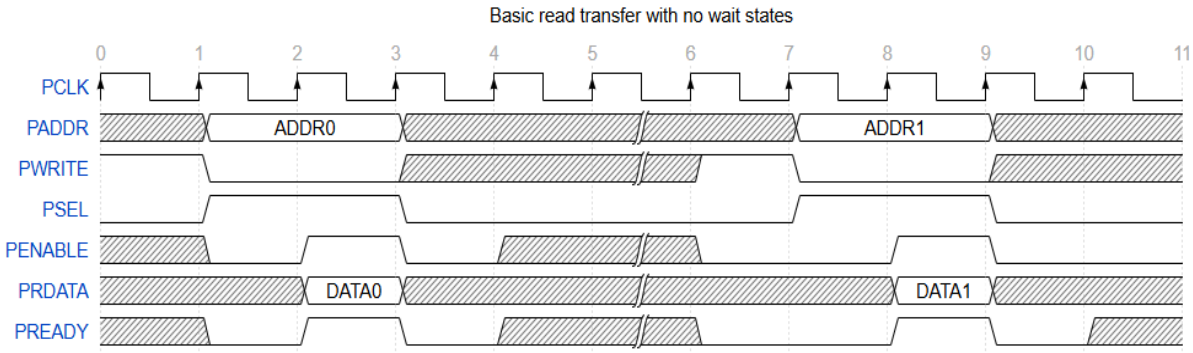

```
{signal: [
  {name: 'PCLK', wave: 'P.............'},
  {name: 'PADDR', wave: 'x2...x|2....x..', data: ['ADDR0', 'ADDR1']},
```

```
  {name: 'PWRITE', wave: 'x0...x|0....x.'},
  {name: 'PSEL', wave: '01...x|1....x.'},
  {name: 'PENABLE', wave: 'x01..0x|01..x.'},
  {name: 'PRDATA', wave: 'x...2x.|x..2x.', data: ['DATA0', 'DATA1',]},
  {name: 'PREADY', wave: 'x0..10x|x0.10x'},  ],
  config: { hscale: 2 },
  head:{
  text:'Basic read transfer with wait states',
  tick:0, every:1 },}
```

Transferul este prelungit dacă PREADY este fixat pe "0" în timpul unei faze de acces.

În timp ce PREADY rămâne pe "0", următoarele semnale rămân neschimbate:

- Semnalul de adresă, PADDR
- Semnalul de direcționare, PWRITE
- Semnalul de selectare, PSEL
- Semnalul de activare, PENABLE




```
{signal: [
    {name: 'PCLK', wave: 'P.........'},
    {name: 'PADDR', wave: 'x2.xx|x2.xx', data: ['ADDR0', 'ADDR1']},
    {name: 'PWRITE', wave: '10.x.|10.x.'},
    {name: 'PSEL', wave: '01.0...1.0.'},
    {name: 'PENABLE', wave: 'x010x|0.10.'},
    {name: 'PRDATA', wave: 'xx2xx|xx2xx', data: ['DATA0', 'DATA1',]},
    {name: 'PREADY', wave: 'x010x|0.10x'},
  ],
```

```
config: { hscale: 2 },
head:{
   text:'Basic read transfer with no wait states',
   tick:0,
   every:1},}
```

Sincronizarea semnalelor de adresare PADDR, scriere PWRITE, selecție PSEL și activare PENABLE este aceeași ca cea descrisă la transferurile de scriere. Furnizorul trebuie să ofere datele înainte de sfârșitul transferului de citire.

Răspunsul la erori

PSLVERR poate fi utilizat pentru a indica o condiție de eroare la un transfer APB. Pot apărea erori atât la tranzacțiile de citire, cât și la cele de scriere.

PSLVERR este considerat valid doar în ultimul ciclu al unui transfer APB, când PSEL, PENABLE și PREADY sunt toate "1".

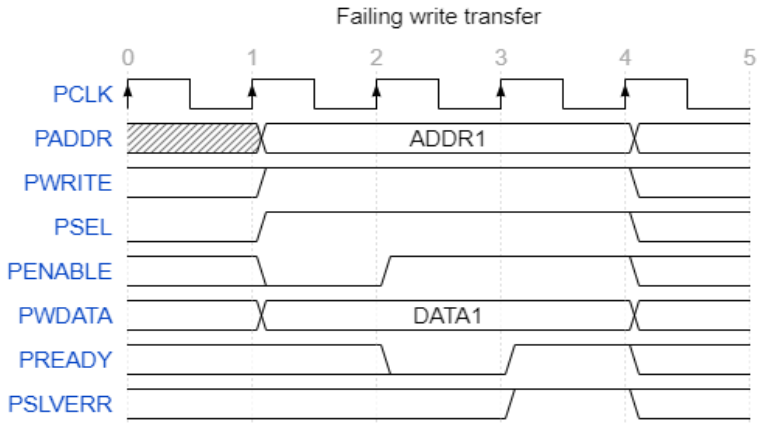
Se recomandă, dar nu este obligatoriu, ca PSLVERR să fie fixat pe "0" atunci când PSEL, PENABLE sau PREADY sunt "0".

Tranzacțiile cu eroare pot sau nu să schimbe starea perifericului - aceasta este specifică perifericului (oricare dintre stări este acceptabilă).

Când e semnalată o eroare în timpul unei tranzacții de scriere, aceasta nu înseamnă că registrul din periferic nu a apucat să fie actualizat.

Tranzacțiile de citire cu erori pot returna date invalide. Nu se impune ca perifericul să pună liniile magistralei de date la ”0” pentru o eroare de citire.
Un Solicitant care primește un semnal de eroare la un transfer de citire poate totuși să folosească datele.
Un Furnizor nu se poate baza pe un semnal de eroare pentru a preveni citirea unei valori de pe PRDATA.

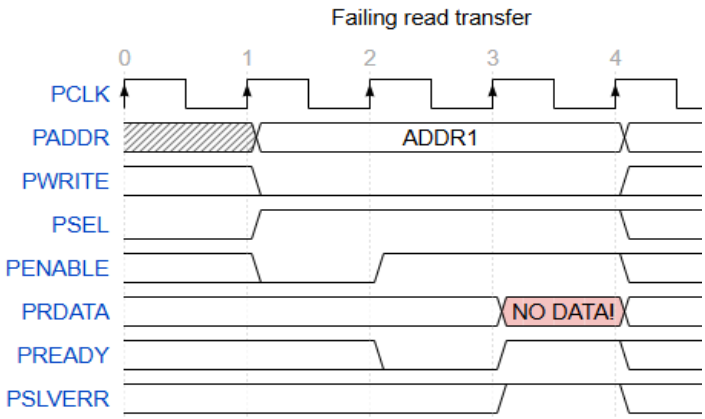
Furnizorii nu sunt obligați să ofere PSLVERR. Când un Furnizor nu oferă PSLVERR, intrarea corespunzătoare în Solicitant este legată la "0".




```
{signal: [
    {name: 'PCLK', wave: 'P....'},
    {name: 'PADDR', wave: 'x2..2', data: ['ADDR1']},
    {name: 'PWRITE', wave: '21..2'},
    {name: 'PSEL', wave: '01..2'},
    {name: 'PENABLE', wave: '201.2'},
    {name: 'PWDATA', wave: '22..2', data: ['','DATA1']},
    {name: 'PREADY', wave: '2.012'},
    {name: 'PSLVERR', wave:'2..12'}
  ],
  config: { hscale: 2 },
head:{
     text:'Failing write transfer',
     tick:0,
     every:1},}
```

În cazul unei scrieri, poate apărea o eroare la nivel de *Slave,* de exemplu, dacă registrul de la adresa ADDR1 poate fi doar citit.

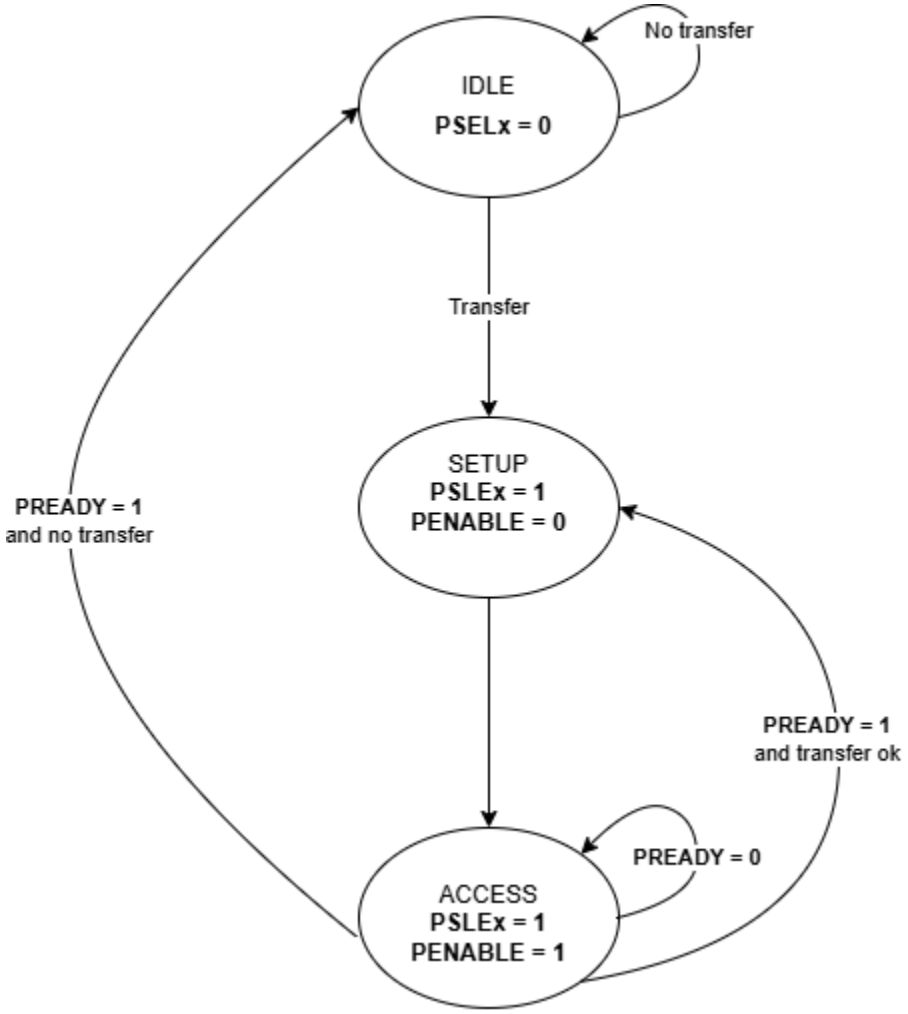
Eroare la un transfer de citire




```
{signal: [
    {name: 'PCLK', wave: 'P....'},
    {name: 'PADDR', wave: 'x2..2', data: ['ADDR1']},
    {name: 'PWRITE', wave: '20..2'},
    {name: 'PSEL', wave: '01..2'},
    {name: 'PENABLE', wave: '201.2'},
    {name: 'PRDATA', wave: '2..92', data: ['','NO DATA!']},
    {name: 'PREADY', wave: '2.012'},
    {name: 'PSLVERR', wave:'2..12'}
  ],
  config: { hscale: 2 },
  head:{
     text:'Failing read transfer',
     tick:0,
     every:1},}
```

Un transfer de citire se poate încheia și cu semnalizarea unei erori, indicându-se că nu sunt disponibile date valide citite (sau din alte motive specifice unui anumit design APB).

De asemenea, un transfer APB poate fi înțeles mai bine dacă fiecare etapă a tranzacției poate fi clasificată ca **o stare operațională.**



Mașina algoritmică de stare ***apb_fsm*** (APB Finite State Machine (automat cu stări finite)) parcurge următoarele stări:

IDLE - Aceasta este starea implicită a interfeței APB.

SETUP - Când este necesar un transfer, interfața trece în starea **SETUP**, unde este activat semnalul de selecție corespunzător, **PSELx**. Interfața rămâne în starea SETUP doar pentru un ciclu de ceas și se mută întotdeauna în starea **ACCESS** pe următorul front ascendent al ceasului.

ACCESS - Semnalul de activare, PENABLE, este activat în starea **ACCESS**.

Următoarele semnale nu trebuie să se schimbe în tranziția dintre **SETUP** și **ACCESS** și între cicluri în starea **ACCESS**:

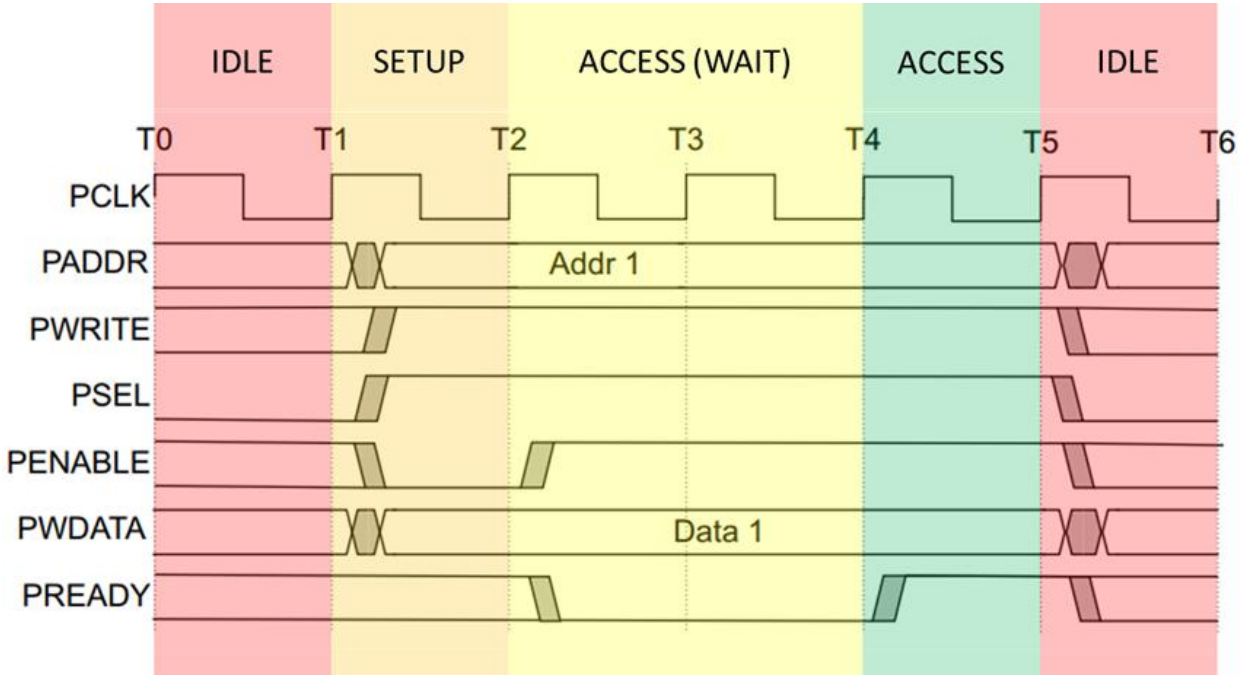
- **PADDR**
- PPROT
- **PWRITE**
- **PWDATA**, *doar pentru tranzacțiile de scriere*
- PSTRB
- PAUSER
- PWUSER

Ieșirea din starea ACCESS este controlată de semnalul **PREADY** de la Furnizor:

- Dacă **PREADY** este menținut "0" de către Furnizor, atunci interfața rămâne în starea ACCES.
- Dacă **PREADY** este trecut pe "1" de către Furnizor, atunci se iese din starea **ACCESS** și

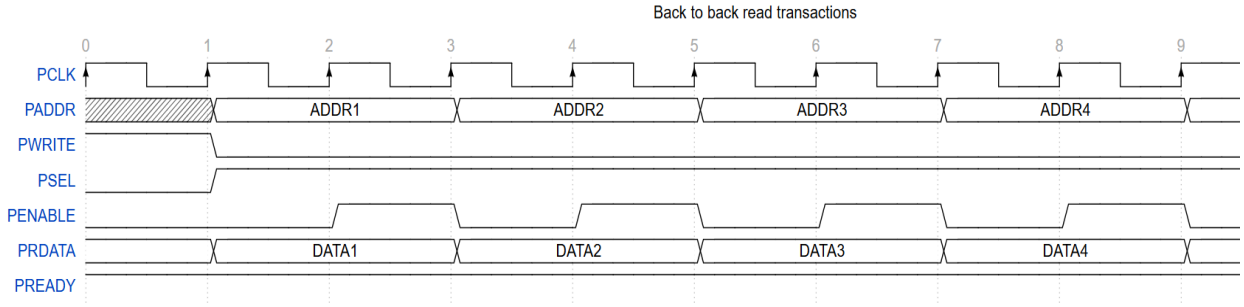
  -interfața revine în starea **IDLE** dacă nu sunt necesare alte transferuri.

  -interfața trece direct în starea **SETUP** dacă urmează un alt transfer.



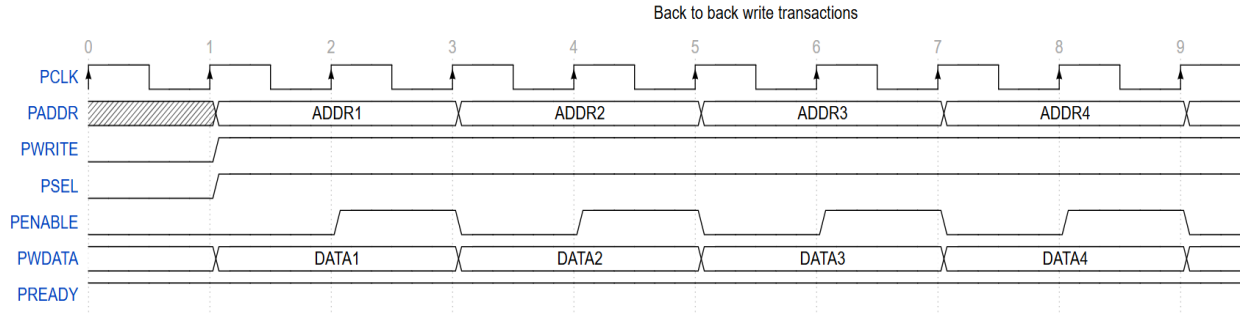
Transferuri ”b2b” (*back-to-back* ”imediat consecutive”)

Următoarea diagramă prezintă transferuri de **citire** *b2b* (fără cicluri idle între ele):



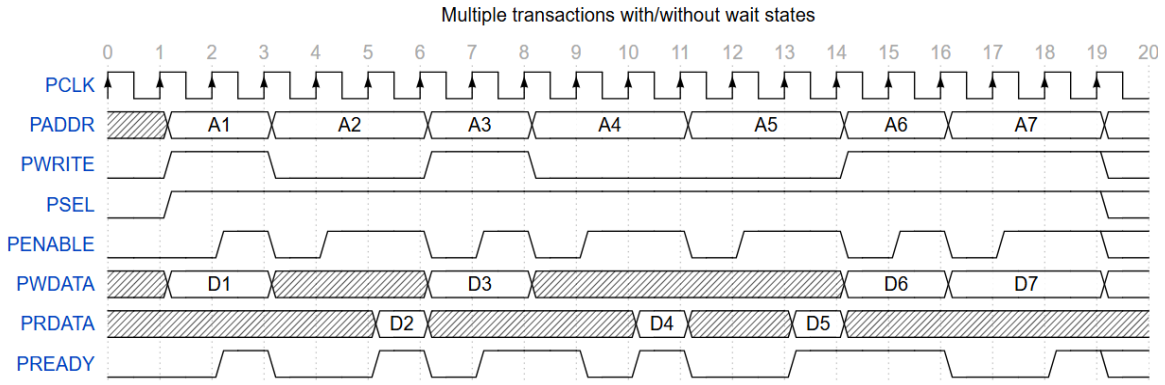

```
{signal: [
    {name: 'PCLK', wave: 'P.........'},
    {name: 'PADDR', wave: 'x2.2.2.2.2.', data: ['ADDR1','ADDR2','ADDR3','ADDR
4','']},
    {name: 'PWRITE', wave: '10........'},
    {name: 'PSEL', wave: '01.......0'},
    {name: 'PENABLE', wave: '0.101010101'},
    {name: 'PRDATA', wave: '22.2.2.2.2', data: ['','DATA1','DATA2','DATA3','D
ATA4',]},
    {name: 'PREADY', wave: '1........1'},
  ],
  config: { hscale: 3 },
  head:{
     text:'Back to back read transactions',
     tick:0,
     every:1},}
```

Următoarea diagramă prezintă un transfer *b2b* de **scriere**:



```
{signal: [
    {name: 'PCLK', wave: 'P.........'},
    {name: 'PADDR', wave: 'x2.2.2.2.2.', data: ['ADDR1','ADDR2','ADDR3','ADDR
4','']},
    {name: 'PWRITE', wave: '01........'},
    {name: 'PSEL', wave: '01........0'},
    {name: 'PENABLE', wave: '0.101010101'},
    {name: 'PRDATA', wave: '22.2.2.2.2', data: ['','DATA1','DATA2','DATA3','D
ATA4',]},
    {name: 'PREADY', wave: '1........1'},
  ],
  config: { hscale: 3 },
  head:{
     text:'Back to back write transactions',
     tick:0,
     every:1},}
```

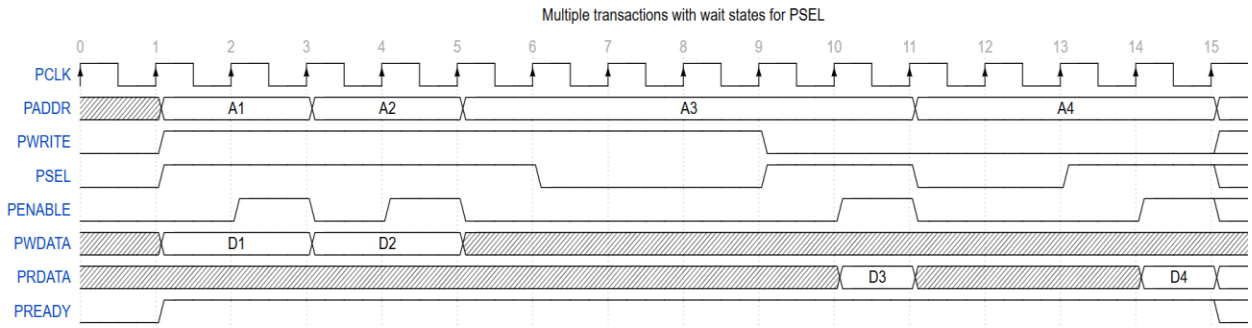
Tranzacții multiple, cu sau fără stări de așteptare




```
{signal: [
    {name: 'PCLK', wave: 'P.................'},
    {name: 'PADDR', wave: 'x2.2..2.2..2..2.2..2', data: ['A1','A2','A3','A4','A5','A6
','A7']},
    {name: 'PWRITE', wave: '01.0..1.0.....1....2'},
    {name: 'PSEL', wave: '01.............2'},
    {name: 'PENABLE', wave: '0.101.0101.01.0101.2'},
    {name: 'PWDATA', wave: 'x2.x..2.x.....2.2..2', data: ['D1','D3','D6','D7']},
    {name: 'PRDATA', wave: 'x....2x...2x.2xxxxxx', data: ['D2','D4','D5']},
    {name: 'PREADY', wave: '0.10.101.010.1..0.12'},
  ],
  config: { hscale: 1 },
  head:{
     text:'Multiple transactions with/without wait states',
     tick:0,
     every:1},}
```

Stări de așteptare pentru **PSEL**

Adăugarea unor stări de așteptare pentru PSEL implică extinderea duratei pentru care semnalul de selecție a perifericului rămâne activ – în acest fel se permite perifericului selectat timp suplimentar pentru a se pregăti în vederea transferului de date, asigurând astfel o comunicare fiabilă pe magistrala APB.



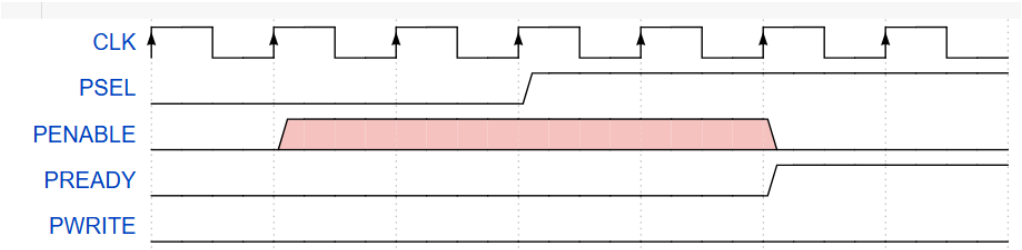

```
{signal: [
    {name: 'PCLK', wave: 'P..............'},
    {name: 'PADDR', wave: 'x2.2.2.....2...2', data: ['A1','A2','A3','A4']},
    {name: 'PWRITE', wave: '01.......0.....2'},
    {name: 'PSEL', wave: '01....0..1.0.1.2'},
    {name: 'PENABLE', wave: '0.1010....10..12'},
    {name: 'PWDATA', wave: 'x2.2.x.........', data: ['D1','D2']},
    {name: 'PRDATA', wave: 'x.........2xxx22', data: ['D3','D4']},
    {name: 'PREADY', wave: '01............2'},
  ],
  config: { hscale:  2},
  head:{
     text:'Multiple transactions with wait states for PSEL',
     tick:0,
     every:1},}
```

Erori de protocol

În această secțiune, prezentăm încălcări comune ale protocolului APB, cu accent pe probleme critice legate de sincronizarea și controlul semnalelor.

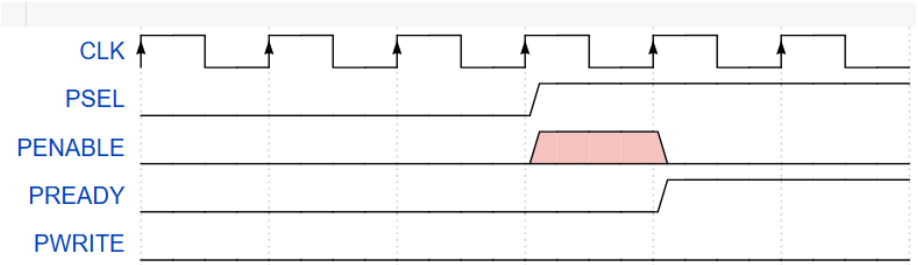
*Activarea **PENABLE** când nu este activ **PSEL***

Dacă are loc o selecție a unui periferic, activarea perifericului nu ar trebui să fie deja ”1”, altfel apar erori în sistem:




```
{
  signal: [
    { name: "CLK", "wave": "P......" },
    { name: "PSEL", "wave": "0..1...",},
    { name: "PENABLE", "wave": "09...0.",},
    { name: "PREADY", "wave": "0....1.", "data": ["NOT READY", "READY"] },
    { name: "PWRITE", "wave": "0......"},
  ],config: { "hscale": 2 }}
```

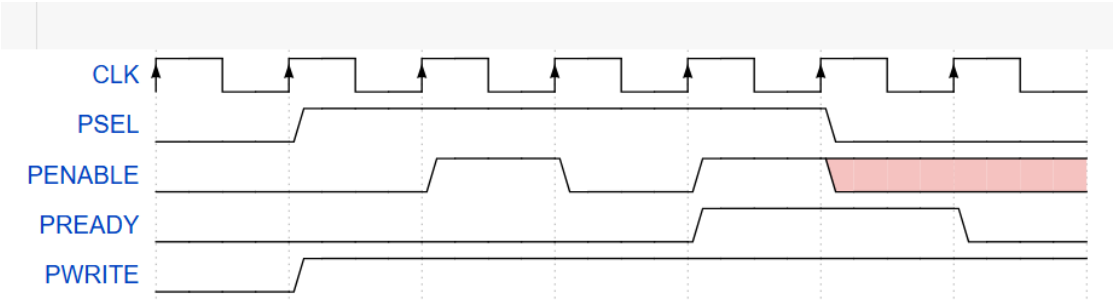
*Activarea **PENABLE** în același ciclu de ceas în care este activat **PSEL***



```
{
  "signal": [
    { name: "CLK", "wave": "P....." },
    { name: "PSEL", "wave": "0..1..",},
    { name: "PENABLE", "wave": "0..90.",},
    { name: "PREADY", "wave": "0...1.", "data": ["NOT READY", "READY"] },
    { name: "PWRITE", "wave": "0....."},
  ], config: { "hscale": 2 }}
```

Activarea **PENABLE** după **PSEL**

- De asemenea, poate apărea o eroare de protocol atunci când PSEL este dezactivat (trece pe ”0”), dar PENABLE rămâne activ, "1".

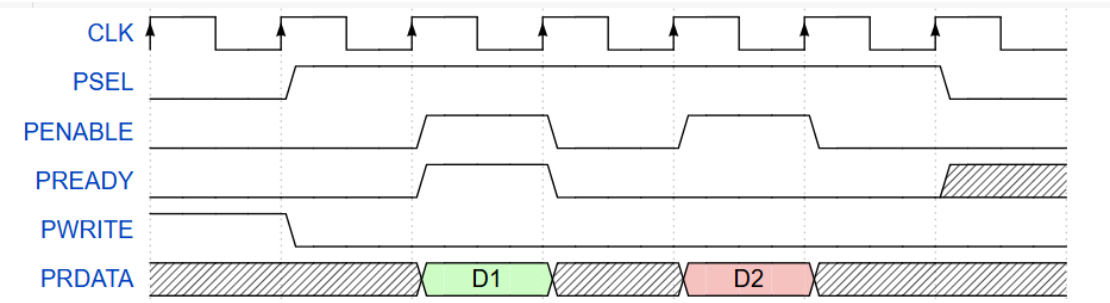


PENABLE rămâne “1” deși **PSEL** coboară la “0”.

```
{
  signal: [
    { name: "CLK", "wave": "P......" },
    { name: "PSEL", "wave": "01....0",},
    { name: "PENABLE","wave": "0.1019.", },
    { name: "PREADY", "wave": "0...1.0",},
    { name: "PWRITE", "wave": "01.....",},
  ],
  "config": { "hscale": 2 }
}
```

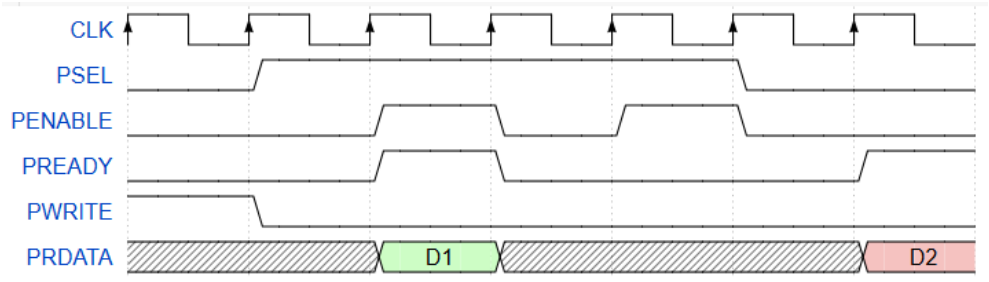
Eroare PREADY

- Un Furnizor nu ar trebui să preia magistrala de citire, de exemplu, dacă nu este pregătit (**READY**) să ducă la bun sfârșit respectiva tranzacție.




```
{signal: [
  {name: "CLK", "wave": "P......" },
  {name: "PSEL", "wave": "01....0",},
  {name: "PENABLE", "wave": "0.1010.", },
  {name: "PREADY", "wave": "0.10..x",},
  {name: "PWRITE", "wave": "10.....",},
  {name: "PRDATA", "wave": "x.7x9xx",data:['D1','D2']}
],config: { "hscale": 2 }}
```

- Acest caz poate fi valabil și dacă **PSEL** este dezactivat (trecut pe ”0”) fără ca Furnizorul să fie pregătit.



```
{signal: [
    {name: "CLK", "wave": "P......" },
    {name: "PSEL", "wave": "01...0.",},
    {name: "PENABLE", "wave": "0.1010.", },
    {name: "PREADY", "wave": "0.10..1",},
    {name: "PWRITE", "wave": "10.....",},
    {name: "PRDATA", "wave": "x.7x..9",data:['D1','D2']}
  ],
  config: { "hscale": 2 }}
```

2. AHB (Advanced High-performance Bus)

AMBA (Advanced Microcontroller Bus Architectures) **AHB** este o interfață de tip magistrală (deci potrivită pentru tehnici de sinteză hardware) utilizată în proiecte avansate [1, pag. 45].

AHB definește interfața dintre componente, cum ar fi ”Managerii”, sub-sistemele de interconectare și ”Subordonații” [8, pag. 1].

AMBA AHB implementează caracteristicile necesare pentru sisteme de înaltă performanță cu frecvențe de ceas ridicate, inclusiv:

- Transferuri în salvă (”rafală”)
- Funcționare sigură pe un singur front de ceas și implementare ”non-tristate” (fără stări ”high Z”, de înaltă impedanță)
- Lățimi configurabile ale magistralei de date și ale magistralei de adrese

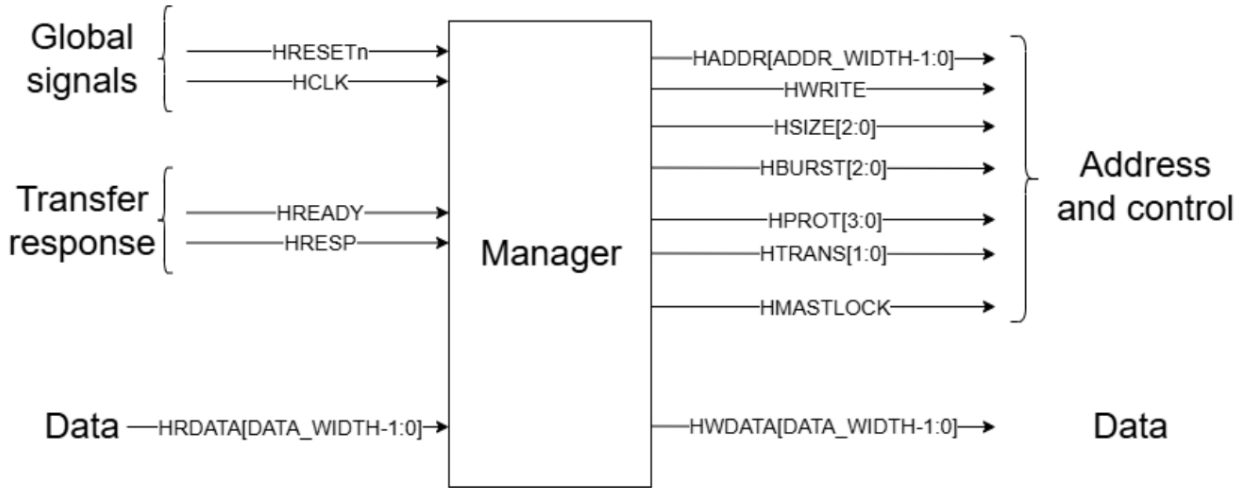
Ca și ”subordonați” AHB, cele mai des întâlnite sunt memoriile interne, interfețele de memorie externă și perifericele cu lățime de bandă mare [3, pag. 57].

Deși perifericele cu lățime de bandă redusă pot fi incluse ca subordonați AHB, din motive de performanță a sistemului ele se află de obicei pe magistrala avansată de periferice AMBA (APB).

Conectarea între AHB cu performanțe superioare și APB necesită un subordonat AHB dedicat – adaptor de tip ”bridge” (”pod”) [8, pag. 1].

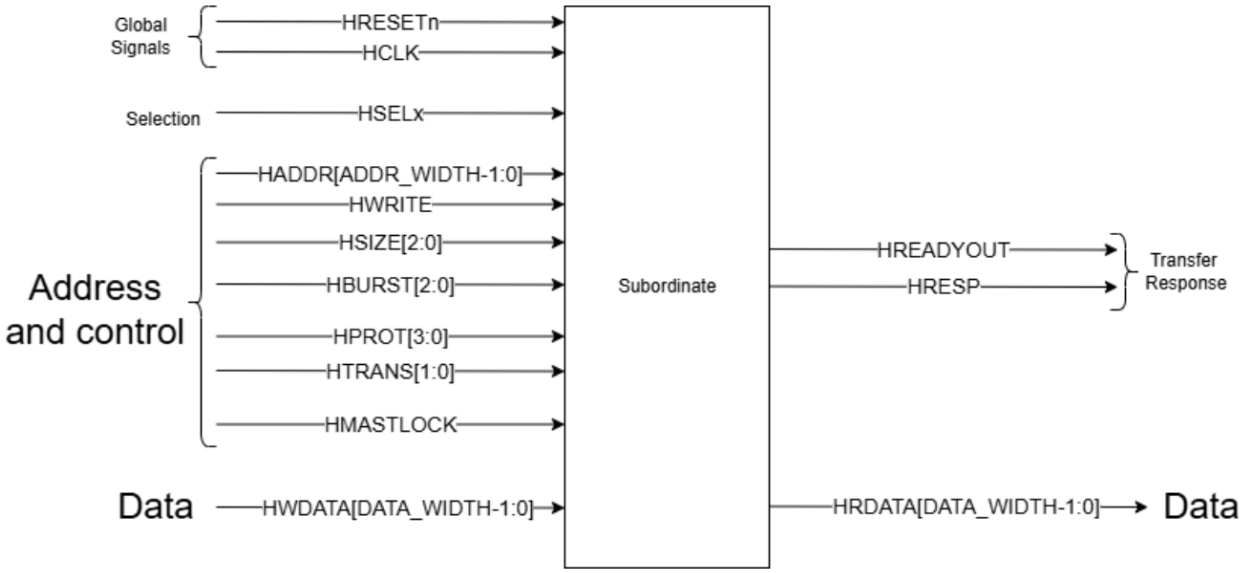
Principalele tipuri de componente ale unui sistem AHB – *Manager, Subordonat, Interconectare:*

Manager



Un Manager furnizează informațiile de adresare și control pentru a iniția operațiunile de citire și scriere.

Subordonat



Un Subordonat răspunde transferurilor inițiate de Managerii din sistem.

Subordonatul folosește semnalul de selecție HSELx de la decodor pentru a controla felul în care răspunde la un transfer pe bus.

Subordonatul semnalează Managerului:

- Finalizarea sau continuarea-extinderea transferului pe bus.
- Succesul sau eșecul transferului pe bus.

Interconectarea

O componentă de interconectare asigură conexiunea dintre Manageri și Subordonați într-un sistem.

- Un singur sistem Manager necesită doar utilizarea unui Decodor și a unui Multiplexor, așa cum este descris în secțiunile următoare.
- Un sistem multi-Manager necesită utilizarea unei interfețe care asigură arbitrajul și direcționarea semnalelor de la diferiți Manageri către Subordonații corespunzători. Această rutare este necesară pentru semnalizarea aferentă controlului, adresării, scrierii și citirii [3, pag. 75].

Decodorul

În toate implementările care folosesc doi sau mai mulți Subordonați există un singur Decodor care:

- decodează adresa fiecărui transfer,
- furnizează semnalul ”select” pentru Subordonatul implicat în transfer
- furnizează un semnal de control către multiplexor.

Multiplexorul

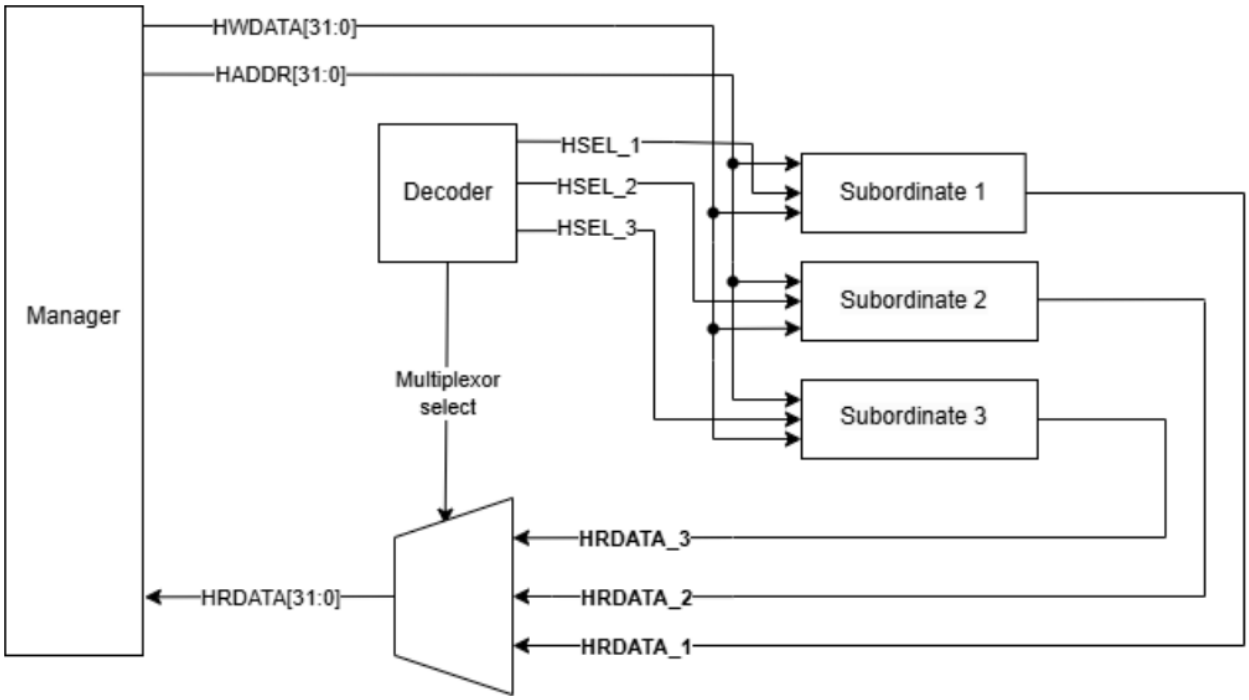
În toate implementările care folosesc doi sau mai mulți Subordonați există un singur Multiplexor pentru:

- semnalele de citire a datelor
- răspunsurile de la Subordonați către Manager

Schema bloc AHB

Figura următoare prezintă schema bloc a unui sistem AHB cu un Manager și trei Subordonați. Logica de Interconectare a magistralei constă dintr-un Decodor de adrese și un Multiplexor *Subordonați↔Manager*.

Decodorul furnizează adresa de selectare a Subordonatului corespunzător în faza de transfer al datelor. Multiplexorul redirecționează datele corespunzătoare de la ieșirea Subordonaților înapoi către Manager.



Parametri

Parametru	Descriere
DATA_WIDTH	Numărul de biți de date transferați între solicitant și furnizor
ADDR_WIDTH	Numărul de biți de adresă
HPROT_WIDTH	Numărul de biți pentru magistrala de protecție
HMASTER_WIDTH	Numărul de biți folosiți pentru a identifica mai mulți Manageri într-un sistem AHB mai complex

Semnale globale

Semnal	Sursă	Lățime	Descriere
HCLK	Ceas	1	Ceasul temporizează – cu comandă pe frontul urcător HCLK – toate transferurile de pe magistrală.
HRESETn	Reset sistem	1	Activ pe ”0”, resetează sistemul și magistrala.

Semnale de la Subordonat(i)

Semnal	Destinație	Lățime	Descriere
HRDATA	Solicitant / Multiplexor	DATA_WIDTH	În timpul operațiunilor de citire, magistrala transferă date de la Subordonatul selectat de Multiplexor către Manager. În cazul unui singur Manager, HRDATA este intrare în acesta.
HREADYOUT/HREADY	Solicitant / Multiplexor	1	Când este ”1”, semnalul HREADYOUT indică faptul că s-a încheiat un transfer pe bus. Acest semnal poate fi coborât pe ”0” pentru a prelungi transferul. În cazul unui singur Manager cu un singur Subordonat, HREADYOUT devine HREADY și reprezintă o intrare pentru Manager. În cazul multi-Manager/multi-Subordonați, HREADYOUT intră în logica de Interconectare care declanșează intrarea HREADY către Manager/Solicitant.
HRESP*	Multiplexor	1	Acest răspuns oferă Managerului informații suplimentare despre stadiul transferului. Când este ”0”, semnalul HRESP indică faptul că starea transferului este OK. Când este ”1”, semnalul HRESP indică faptul că starea transferului este ERROR.
HEXOKAY*	Multiplexor	1	”Exclusive Okay” - Indică succesul sau eșecul unui transfer exclusiv.

** Disponibil pentru versiunea AHB5 a protocolului*

Semnale de la Manager

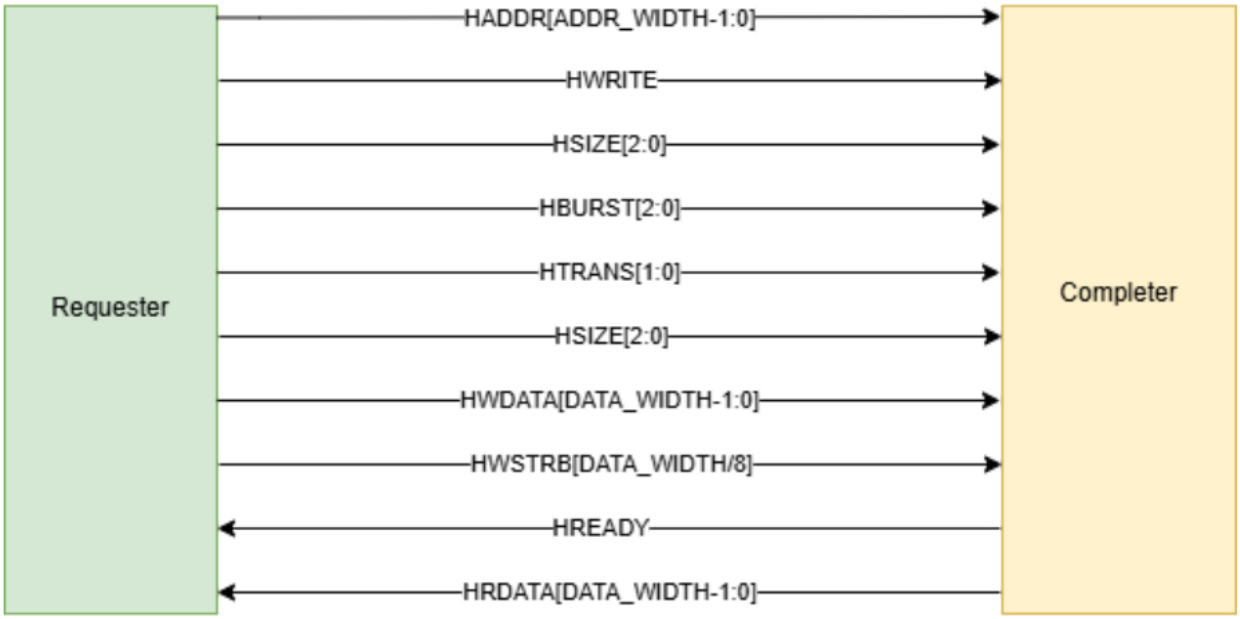
Semnal	Destinație	Lățime	Descriere
HADDR	Subordonat și Decodor	ADDR_WIDTH	Adresa transferului (către un spațiu organizat pe octeți). ADDR_WIDTH se recomandă să fie între 10 și 64.
HBURST	Subordonat	HBURST_WIDTH	Indică câte transferuri sunt în salvă (rafală) și cum incrementează adresa. Trebuie să fie 0 sau 3.
HMASTLOCK*	Subordonat	1	Indică faptul că transferul curent face parte dintr-o secvență ce nu poate fi întreruptă. Aceeași sincronizare cu semnalele de adresă și control.
HPROT*	Subordonat	HPROT_WIDTH	Controlul accesului protejat (informații asupra tipului de acces). HPROT_WIDTH trebuie să fie 0, 4, sau 7.
HSIZE	Subordonat	3	Mărimea transferului.
HNONSEC*	Subordonat și Decodor	1	Indică dacă transferul este sigur (sau nu).
HEXCL*	Monitor Acces Exclusiv	1	Indică dacă transferul face parte dintr-o secvență de acces Exclusiv.
HMASTER*	Monitor Acces Exclusiv și Subordonat	HMASTER_WIDTH	Identificator de Manager. Generat de Manager dacă poate controla mai multe fire de execuție (”threads”) cu acces Exclusiv. Modificat de Interfațăpentru a asigura că fiecare Manager e unic identificabil.
HTRANS	Subordonat	2	Indică tipul de transfer. Aceasta poate fi: IDLE; BUSY; NONSEQUENTIAL; SEQUENTIAL.
HWDATA	Subordonat	DATA_WIDTH	Transferă date de la Manager către Subordonați în timpul operațiunilor de scriere. DATA_WIDTH poate fi 8, 16, 32, 64, 128, 256, 512, sau 1024. Totuși, valori mai mici de 32 sau mai mari de 256 nu sunt recomandate.
HWSTRB	Subordonat	DATA_WIDTH/8	Semnal de validitate (”strobe”) pentru scriere. Dezactivat spre a indica intervalele când liniile active pentru octeții de scriere nu conțin date valide. Există 1 bit pentru fiecare 8 biți din HWDATA. HWSTRB[n] corespunde cu HWDATA[(8n)+7:(8n)].

Semnal	Destinație	Lățime	Descriere
			HWSTRB contează în faza de transfer de date și are aceleași reguli de validitate ca HWDATA.
HWRITE	Subordonat	1	Indică direcția de transfer. Când e ”1”, acest semnal indică un transfer de scriere și când e ”0” indică un transfer de citirer. Are aceeași sincronizare ca semnalele de adresă, însă trebuie să rămână constant pe tot parcursul unui transfer în salvă.

Semnalele marcate cu "*" sunt destinate arhitecturilor multi-manageri/subordonați. Aceste semnale sunt destinate unui singur manager multi-subordonați sau multi-manageri-multi-subordonați.

Diagrama un manager / un subordonat

În cele ce urmează, accentul va fi pus pe configurația unu-la-unu cu diagrama de mai jos. Managerul devine Solicitant, iar subordonatul Furnizor [3, pag. 57]. Logica de interconectare nu mai este necesară.



Transferuri de bază

Un transfer constă din două faze:

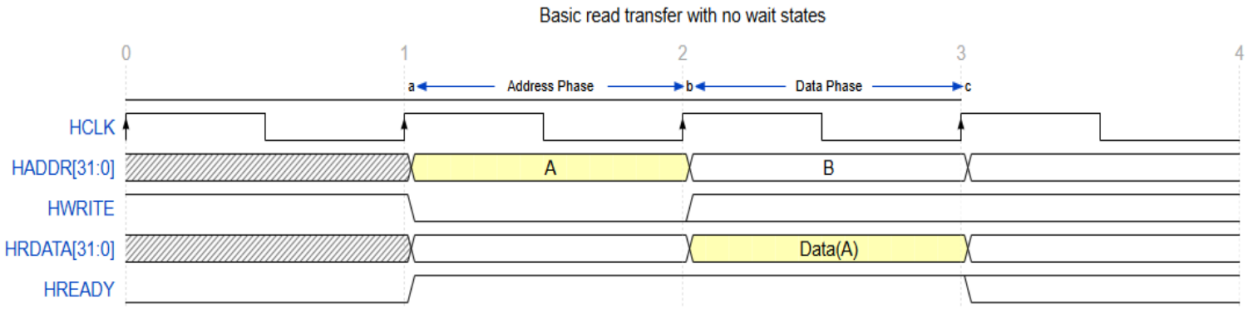
Adresarea : Durează un singur ciclu HCLK, cu excepția cazului în care este prelungită de transferul anterior pe magistrală.

Datele : Transferul lor s-ar putea să necesite mai multe cicluri HCLK. Se folosește semnalul HREADY pentru a controla numărul de cicluri de ceas necesare pentru a finaliza transferul.

HWRITE controlează direcția transferului de date către sau de la Manager. Prin urmare, când:

1. **HWRITE** este ”1”, se indică un transfer de scriere, iar Managerul transmite datele pe magistrala de date de scriere, **HWDATA**.
2. **HWRITE** este ”0”, se efectuează un transfer de citire, iar Subordonatul trebuie să furnizeze datele pe magistrala de date de citire, **HRDATA**.

Cel mai simplu transfer este fără stări de așteptare, transferul constând într-un **ciclu de adresare** și un **ciclu de date**. Mai jos sunt prezentate un transfer simplu de citire și un transfer simplu de scriere.

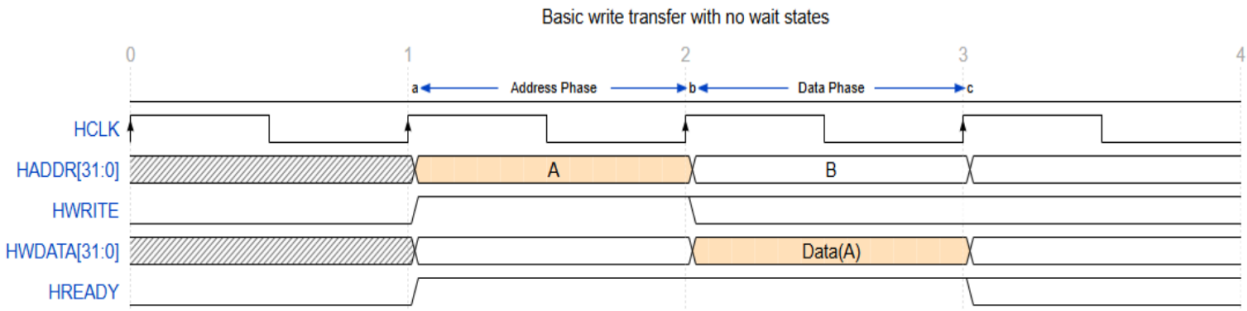



```
{signal: [
  {name: '', wave: '0..', node:'.abc'},
  {name: 'HCLK', wave: 'P...',},
  {name: 'HADDR[31:0]', wave: 'x322', data: ['A', 'B']},
  {name: 'HWRITE', wave: '102.'},
```

```
  {name: 'HRDATA[31:0]', wave: 'x232',data: ['','Data(A)','']},
  {name: 'HREADY', wave: '01.2'},
  ],
  edge: ['a<->b Address Phase', 'b<->c Data Phase'],
config: { hscale: 6 },
head:{
     text:'Basic read transfer with no wait states',
     tick:0,
     every:1
  },}
```

Se consideră ciclul 0 ca o tranzacție finalizată anterior care nu contează. Semnalizările ”hand-shake” pentru citire încep în ciclul 1 în faza de adresare, **HWRITE** este ”0”. În acest ciclu 1, adresa A este setată, **HREADY** este de asemenea activat, astfel încât Furnizorul este pregătit să finalizeze tranzacția în ciclul următor, al datelor, solicitând totodată fie o citire, fie o scriere pentru adresa B.

În cazul unei tranzacții de scriere fără stări de așteptare, desfășurarea este aproape la fel, excepțând evoluția semnalului HWRITE:




```
{signal: [
  {name: '', wave: '0...', node:'.abc'},
  {name: 'HCLK', wave: 'P...',},
  {name: 'HADDR[31:0]', wave: 'x422', data: ['A', 'B']},
  {name: 'HWRITE', wave: '012.'},
  {name: 'HWDATA[31:0]', wave: 'x242',data: ['','Data(A)','']},
  {name: 'HREADY', wave: '01.2'},
  ],
  edge: ['a<->b Address Phase', 'b<->c Data Phase'],
config: { hscale: 6 },
head:{
     text:'Basic write transfer with no wait states',
     tick:0,
     every:1
  },}
```

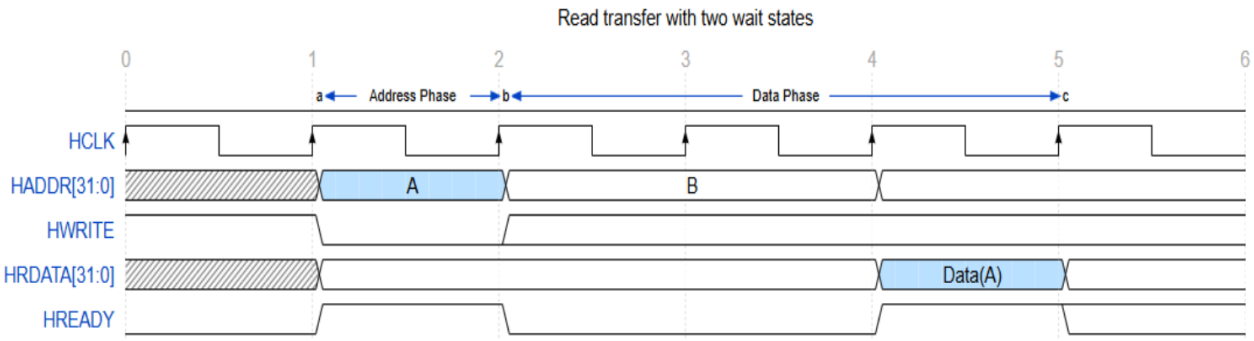
Într-un transfer simplu, fără stări de așteptare:

1. Solicitantul plasează semnalele de adresă și control pe magistrală pe frontul urcător al **HCLK**.

2. Furnizorul eșantionează apoi adresa și informația de control pe următorul front ascendent al **HCLK**.

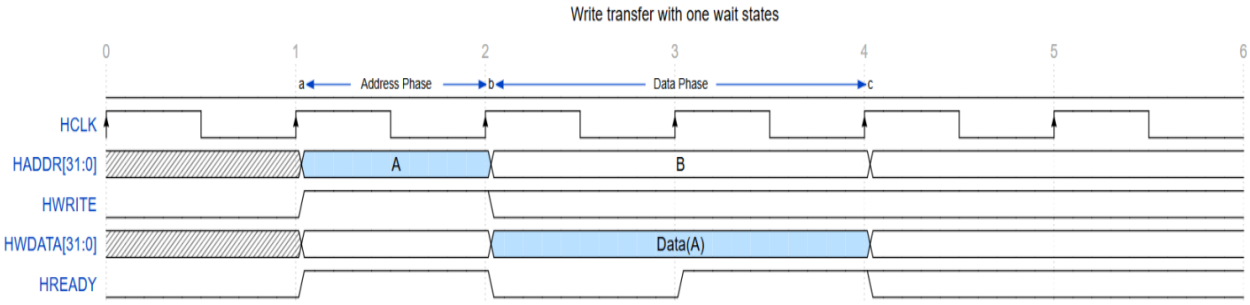
3. Un Furnizor poate introduce stări de așteptare în orice transfer spre a permite timpi suplimentari pentru finalizare. Semnalul HREADY este folosit pentru a controla desfășurarea transferurilor.

Transfer de citire cu două stări de așteptare:




```
{signal: [
  {name: '', wave: '0.....', node:'.ab..c'},
  {name: 'HCLK', wave: 'P.....',},
  {name: 'HADDR[31:0]', wave: 'x52.2.', data: ['A', 'B']},
  {name: 'HWRITE', wave: '102...'},
  {name: 'HRDATA[31:0]', wave: 'x2..52',data: ['','Data(A)','']},
  {name: 'HREADY', wave: '010.12'},
  ],
  edge: ['a<->b Address Phase', 'b<->c Data Phase'],
  config: { hscale: 4 },
  head:{
     text:'Read transfer with two wait states',
     tick:0,
     every:1
  },}
```

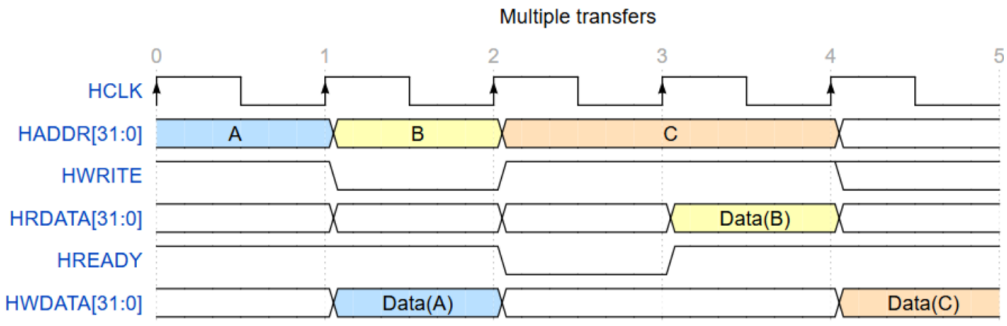
Transfer de scriere cu o stare de așteptare:




```
{signal: [
  {name: '', wave: '0.....', node:'.ab.c'},
  {name: 'HCLK', wave: 'P.....',},
  {name: 'HADDR[31:0]', wave: 'x52.2.', data: ['A', 'B']},
  {name: 'HWRITE', wave: '012...'},
  {name: 'HWDATA[31:0]', wave: 'x25.2.',data: ['','Data(A)','']},
  {name: 'HREADY', wave: '01012.'},
  ],
  edge: ['a<->b Address Phase', 'b<->c Data Phase'],
  config: { hscale: 5 },
  head:{
     text:'Write transfer with one wait states',
     tick:0,
     every:1
  },}
```

Notă - Pentru operațiunile de scriere, solicitantul menține datele stabile pe parcursul ciclurilor extinse. Pentru transferurile de citire, Furnizorul nu trebuie să furnizeze date valide până când transferul nu este pe punctul de a se finaliza.

Transferuri multiple, cu ambele tipuri de stări – cu și fără așteptare



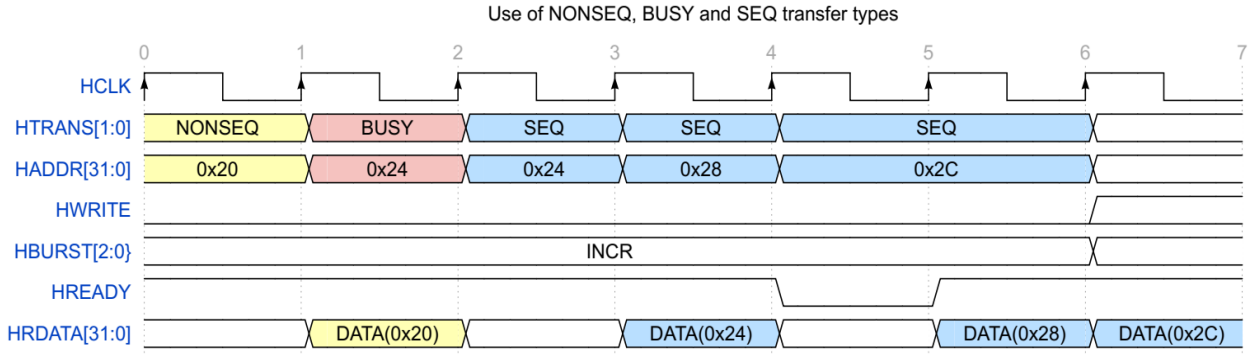
```
{signal: [
  {name: 'HCLK', wave: 'P....',},
  {name: 'HADDR[31:0]', wave: '534.2', data: ['A','B','C']},
  {name: 'HWRITE', wave: '101.2'},
  {name: 'HRDATA[31:0]', wave: '22232',data: ['','','','Data(B)','']},
  {name: 'HREADY', wave: '1.01.'},
  {name: 'HWDATA[31:0]', wave: '252.4',data: ['','Data(A)','','Data(C)']},
  ],
  config: { hscale: 3 },
  head:{
     text:'Multiple transfers',
     tick:0,
     every:1
  },}
```

- Transferurile către adresele A și C sunt fără stări de așteptare
- Transferul către adresa B are o stare de așteptare
- Extinderea fazei de date a transferului la adresa B are efectul de a extinde faza de adresare a transferului la adresa C.

Mai jos sunt prezentate transferurile care pot fi clasificate în unul dintre cele patru tipuri, pe baza **HTRANS[**1:0].

HTRANS[1:0]	TiP	Descriere
2'b00	IDLE	Indică faptul că nu este necesar niciun transfer de date. Managerul folosește un transfer IDLE atunci când nu dorește să efectueze un transfer de date. Subordonații trebuie să emită întotdeauna un răspuns OKAY (fără stări de așteptare) la transferurile IDLE ; transferul este ignorat de acești subordonați.
2'b01	BUSY	Tipul de transfer BUSY permite Managerilor să insereze cicluri IDLE (inactive) în mijlocul unei salve. Acest tip de transfer indică faptul că Managerul continuă cu o salvă, dar acest transfer nu poate avea loc imediat. Când un Manager folosește tipul de transfer BUSY, semnalele de adresă și control trebuie să corespundă următorului transfer din salvă.

HTRANS[1:0]	TiP	Descriere
2’b10	NONSEQ	Indică un transfer singular sau primul transfer al unei salve. Semnalele de adresă și control nu au legătură cu transferul anterior. Transferurile individuale pe magistrală sunt tratate ca salve de lungime unu și, prin urmare, tipul de transfer este NONSEQUENTIAL (nesecvențial).
2’b11	SEQ	Transferurile rămase din salvă sunt secvențiale (SEQUENTIAL), iar adresa corespunde transferului anterior. De asemenea, informația de control e identică cu cea pentru transferul anterior. Adresa va fi egală cu adresa transferului anterior plus mărimea transferului, în octeți ; mărimea transferului este indicate de semnalele HSIZE[2:0]. În cazul unei salve de tip ” wrapping” (”cu înfășurare”) adresa unui transfer care începe din interiorul unei linii cache, după ce ajunge la ultimul cuvânt al liniei se întoarce la începutul acesteia pentru a încheia transferul ”circular” al întregului conținut.




```
{signal: [
  {name: 'HCLK', wave: 'P......',},
  {name: 'HTRANS[1:0]',     wave: '39555.2',data: ['NONSEQ','BUSY','SEQ','SEQ
','SEQ']},
  {name: 'HADDR[31:0]', wave: '39555.2', data: ['0x20','0x24','0x24','0x28',
```

```
'0x2C']},
  {name: 'HWRITE', wave: '0.....2'},
  {name: 'HBURST[2:0}', wave: '2.....2', data:['INCR']},
  {name: 'HREADY', wave: '1...01.'},
  {name: 'HRDATA[31:0]', wave: '2325255',data: ['','DATA(0x20)','','DATA(0x2
4)','','DATA(0x28)','DATA(0x2C)']},
  ],
  config: { hscale: 3 },
  head:{
     text:'Use of NONSEQ, BUSY and SEQ transfer types',
     tick:0,
     every:1
  },}
```

Analiza detaliată pe fiecare ”tact” (T) al semnalului de ceas:

T0-T1: Citirea (pe 4 tacte) începe cu un transfer NONSEQ. NONSEQ indică tipul salvă.

T1-T2: Solicitantul nu poate genera al doilea tact și semnalizează BUSY (pentru a întârzia începutul celei de al doilea tact). Furnizorul oferă datele citite la primul tact.

T2-T3: Solicitantul este acum gata să înceapă al doilea tact, astfel că se semnalează un transfer SEQ. Solicitantul ignoră orice date furnizate de Furnizor pe magistrala de date de citite.

T3-T4 Solicitantul generează al treilea tact. Furnizorul oferă datele citite pentru al doilea tact.

T4-T5 Solicitantul generează ultimul tact. Furnizorul nu poate finaliza transferul și folosește HREADY pentru a introduce o stare singulară de așteptare.

T5-T6 Furnizorul oferă datele citite pentru al treilea tact.

T6-T7 Furnizorul oferă datele citite pentru ultimul tact.

Dimensiunea transferului

`HSIZE[2:0]` indică dimensiunea unui transfer de date

HSIZE[2]	HSIZE[1]	HSIZE[0]	Mărime	Descriere
0	0	0	8	Octet
0	0	1	16	Jumătate de cuvânt
0	1	0	32	Cuvânt
0	1	1	64	Cuvânt dublu
1	0	0	128	Linie de 4 cuvinte
1	0	1	256	Linie de 8 cuvinte
1	1	0	512	-
1	1	1	1024	-

Notă - Dimensiunea transferului indicată de HSIZE trebuie să fie mai mică sau egală cu lățimea magistralei de date. De exemplu, cu o magistrală de date pe 32 de biți, HSIZE trebuie să indice doar valorile 0b000, 0b001 sau 0b010. Semnalele HSIZE au aceeași sincronizare ca și magistrala de adrese. Totuși, ele trebuie să rămână constante pe tot parcursul unui transfer în salvă. HSIZE, împreună cu HBURST, determină limita de adresă pentru salvele ”cu înfășurare” (”wrapping bursts”).

Funcționarea în salve

În acest protocol sunt stipulate salve de lungime definită (4, 8 și 16 tacte), salve de lungime nedefinită precum și transferuri singulare.

Sunt reglementate salve cu incrementare și salve ”cu înfășurare”:

1. Salvele cu incrementare accesează locațiile secvențial (”la rând”), iar adresa fiecărui transfer din salvă este o incrementarea a adresei anterioare.
2. ”Înfășurarea” – un transfer poate începe din interiorul unei linii cache iar după ce se ajunge la ultimul cuvânt al liniei se revine la începutul acesteia pentru a încheia transferul ”circular” al întregului conținut. Limita de adresă este calculată ca produsul dintre numărul de tacte al unei salve și dimensiunea transferului. Numărul de tacte este controlat de HBURST, iar dimensiunea transferului este controlată de HSIZE.

Exemplificăm o salvă cu înfășurare de patru tacte pentru cuvinte de câte 4 octeți având prima adresă 0x34, apoi 0x38, 0x3C și, la limita segmentului de 16 octeți, ultima adresa se întoarce la 0x30 – a se vedea detaliile de mai jos.

HBURST[2:0] controlează tipul de salve:

HBURST[2:0]	Tip	**Descriere**
3'b000	SINGLE	Transfer într-o singură salvă
3'b001	INCR	Salvă cu incrementare, de lungime nedefinită
3'b010	WRAP4	Salvă cu înfășurare, de 4 tacte
3'b011	INCR4	Salvă cu incrementare, de 4 tacte
3'b100	WRAP8	Salvă cu înfășurare, de 8 tacte
3'b101	INCR8	Salvă cu incrementare, de 8 tacte
3'b110	WRAP16	Salvă cu înfășurare, de 16 tacte
3'b111	INCR16	Salvă cu incrementare, de 16 tacte

Managerii nu trebuie să inițieze o salvă cu incrementare care să depășească o limită de adresare de 1KB.

Managerii pot efectua transferuri singulare folosind fie o singură salvă de transfer, fie o salvă de lungime nedefinită.

Notă: Dimensiunea salvei indică numărul de tacte ale salvei și nu numărul de octeți transferați. Se poate calcula câte date sunt transferate într-o salvă prin înmulțirea numărului de tacte cu mărimea transferului de date la fiecare tact (indicat de HSIZE[2:0]).

Toate transferurile dintr-o salvă trebuie să fie aliniate la limita de adresare egală cu dimensiunea transferului. De exemplu, transferurile de cuvinte trebuie să se alinieze la limitele de adresare pe cuvinte (HADDR[1:0] = 0b00), iar transferurile pe jumătăți de cuvânt la limitele adresare respective (pe jumătăți de cuvinte) (HADDR[0] = 0).

Alinierea se recomandă și la adresarea transferurilor IDLE - pentru a evita avertismentele false în regim de simulare, aceasta se poate considera o regulă, nu o recomandare.

Formula de dimensionare a transferului cu înfășurare :

$$\text{Limita_inf_înfășurare} = \left(\text{INT}\left(\frac{\text{Adresă_start}}{\text{HSIZE} \times \text{HBURST}}\right)\right) \times (\text{HSIZE} \times \text{HBURST})$$

Explicitare:

* Adresă_start: Adresa unde începe transferul cu înfășurare.

HSIZE: Indică mărimea în octeți a fiecărui transfer. Mărimi uzuale sunt 1 octet, 2 octeți, 4 octeți (cuvânt), 8 octeți, etc.

HBURST: Numărul de transferuri din salvă. Lungimea uzuală a salvei este 1 (singlular), 4, 8, și 16 transferuri.

INT: partea întreagă

Examplu:

1. Adresa_start: 0x34 ; HSIZE: 4 octeți (mărimea unui cuvânt) ; HBURST: 4 transferuri
2. Produsul: HSIZE×HBURST=4×4=16 octeți
3. Împărțire: $34_{(16)} = 52_{(10)}$; 52/16=3,375
4. Parte întreagă: INT(3,375)=3
5. Înmulțire: 3×16=48 (=30 în hexadecimal)

Așadar, adresa de start pentru transferul circular (cu înfășurare) este 0x30.

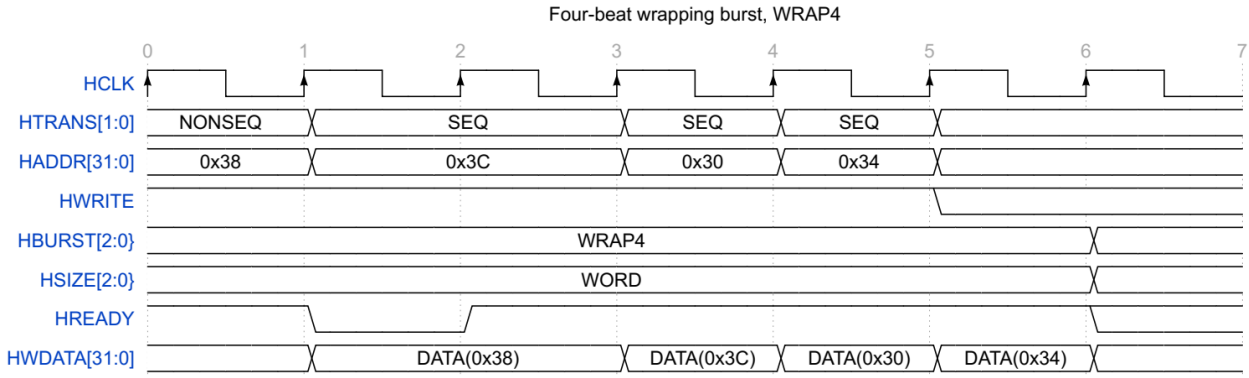
Înfășurarea (”Wrapping”): Primul transfer la adresa = 0x34 ; Al doilea transfer la adresa = 0x38 ; Al treilea transfer la adresa = 0x3C ; Al patrulea transfer la adresa = 0x30

Așadar, după ale treilea transfer, adresa ”se răsucește” înapoi la limita inferioară a ferestrei de transfer ”cu înfășurare”.

Al 5-lea transfer la adresa = 0x34 (deci ciclicitate) – acest mod de adresare continuă pe toată durata salvei.

Salva cu înfășurare, de 4 tacte, WRAP4

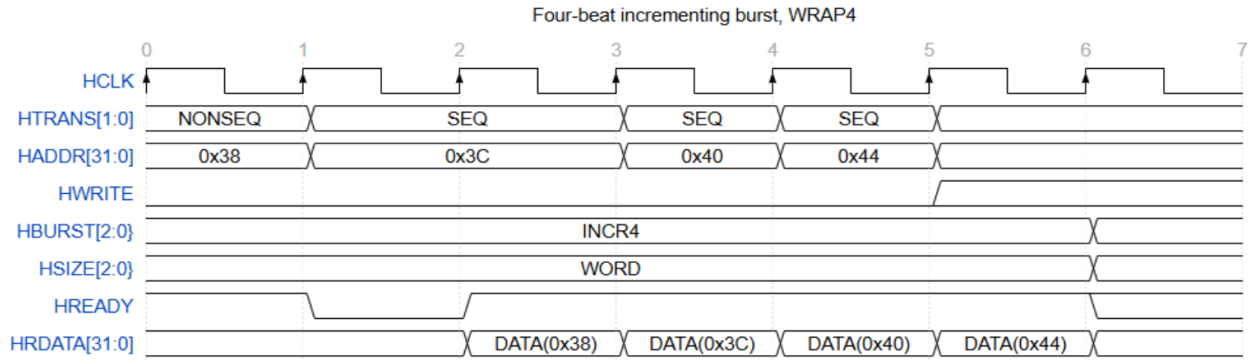
În diagrama de mai jos e prezentată o salvă cu înfășurare, de 4 tacte, cu o stare de așteptare pentru primul transfer. S-a considerat ca adresă de start 0x38, cu înfășurare (”wrapping”) la 0x30.




```
{signal: [
  {name: 'HCLK',                wave: 'P......',},
  {name: 'HTRANS[1:0]',    wave: '22.222.',  data: ['NONSEQ','SEQ','SEQ','SEQ
']},
  {name: 'HADDR[31:0]',     wave: '22.222.',  data: ['0x38','0x3C','0x30','0x
34']},
  {name: 'HWRITE',            wave: '1....2.'},
  {name: 'HBURST[2:0}',     wave: '2.....2', data:['WRAP4']},
  {name: 'HSIZE[2:0}',      wave: '2.....2', data:['WORD']},
  {name: 'HREADY',            wave: '101...2'},
  {name: 'HWDATA[31:0]',    wave: '22.2222',data: ['','DATA(0x38)','DATA(0x3
C)','DATA(0x30)','DATA(0x34)']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Four-beat wrapping burst, WRAP4',
     tick:0,
     every:1
  },}
```

Salva cu incrementare, de 4 tacte, INCR4

În diagrama de mai jos e prezentat un transfer de citire utilizând o salvă cu incrementare, de 4 tacte, cu o stare de așteptare adăugată pentru primul transfer. În acest caz adresa nu ”se înfășoară” la limita a 16 octeți așa încât după adresa 0x3C urmează un transfer la adresa 0x40.

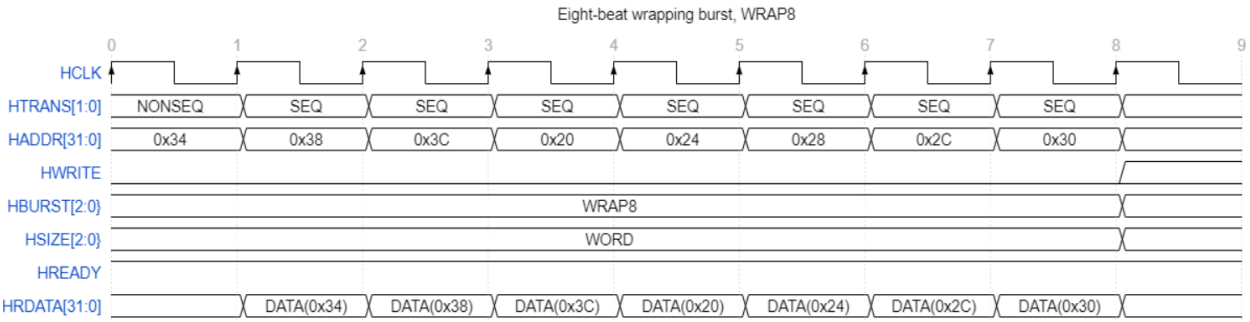



```
{signal: [
  {name: 'HCLK',      wave: 'P......',},
  {name: 'HTRANS[1:0]',   wave: '22.222.',  data: ['NONSEQ','SEQ','SEQ','SEQ
']},
  {name: 'HADDR[31:0]', wave: '22.222.',  data: ['0x38','0x3C','0x40','0x44
']},
  {name: 'HWRITE',    wave: '0....2.'},
  {name: 'HBURST[2:0}', wave: '2.....2', data:['INCR4']},
  {name: 'HSIZE[2:0}', wave: '2.....2', data:['WORD']},
  {name: 'HREADY',    wave: '101...2'},
  {name: 'HRDATA[31:0]', wave: '2.22222',data: ['','DATA(0x38)','DATA(0x3C)',
'DATA(0x40)','DATA(0x44)']},
  ],
config: { hscale: 3 },
head:{
     text:'Four-beat incrementing burst, WRAP4',
     tick:0,
     every:1
  },}
```

Totodată, se poate implementa o salvă de dimensiune mai mare.

În exemplul următor e prezentată o salvă WRAP8. Aplicând formula anterioară, pentru start la adresa 0x34, salva are 8 tacte - pentru transfer pe cuvinte de 4 octeți rezultă:

$$\text{Limita_inf_înfășurare} = \left(\text{INT}\left(\frac{0x34}{4\times 8}\right)\right) \times (4 \times 8) = 0x20$$

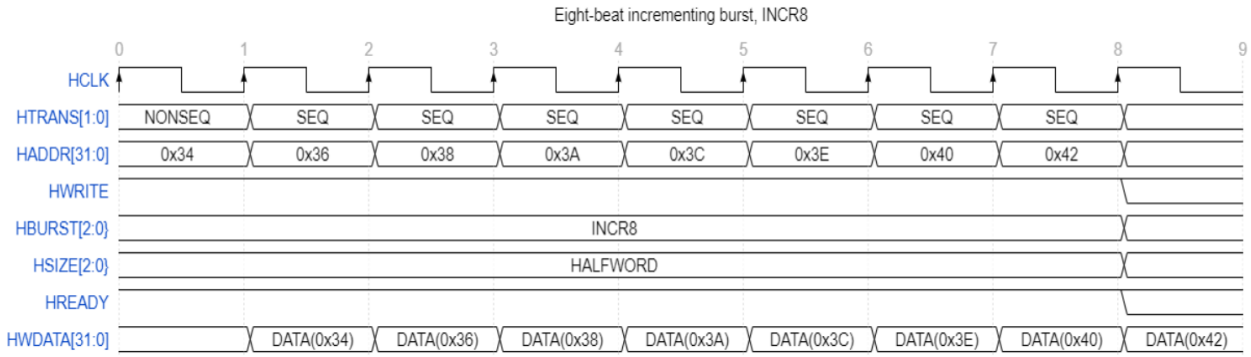


```
{signal: [
  {name: 'HCLK',                wave: 'P........',},
  {name: 'HTRANS[1:0]',   wave: '222222222',  data: ['NONSEQ','SEQ','SEQ','SE
Q','SEQ','SEQ','SEQ','SEQ']},
  {name: 'HADDR[31:0]',     wave: '222222222',  data: ['0x34','0x38','0x3C','
0x20','0x24','0x28','0x2C','0x30']},
  {name: 'HWRITE',            wave: '0.......2'},
  {name: 'HBURST[2:0}',     wave: '2.......2', data:['WRAP8']},
  {name: 'HSIZE[2:0}',      wave: '2.......2', data:['WORD']},
  {name: 'HREADY',            wave: '1........'},
  {name: 'HRDATA[31:0]',    wave: '222222222',data: ['','DATA(0x34)','DATA(0x
38)','DATA(0x3C)','DATA(0x20)','DATA(0x24)','DATA(0x2C)','DATA(0x30)']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Eight-beat wrapping burst, WRAP8',
     tick:0,
     every:1
  },}
```

Un transfer incremental se desfășoară astfel:

- Adresa de start: Solicitantul specifică adresa de start pentru transfer.
- Incrementul adresei: Pentru fiecare tact ulterior, adresa este incrementată de dimensiunea cuvântului. De exemplu, dacă dimensiunea cuvântului este de 4 octeți, adresa crește cu 4 pentru fiecare tact.
- Succesiunea adreselor: Pentru fiecare tact, adresele se succeed crescător, fără înfășurare.

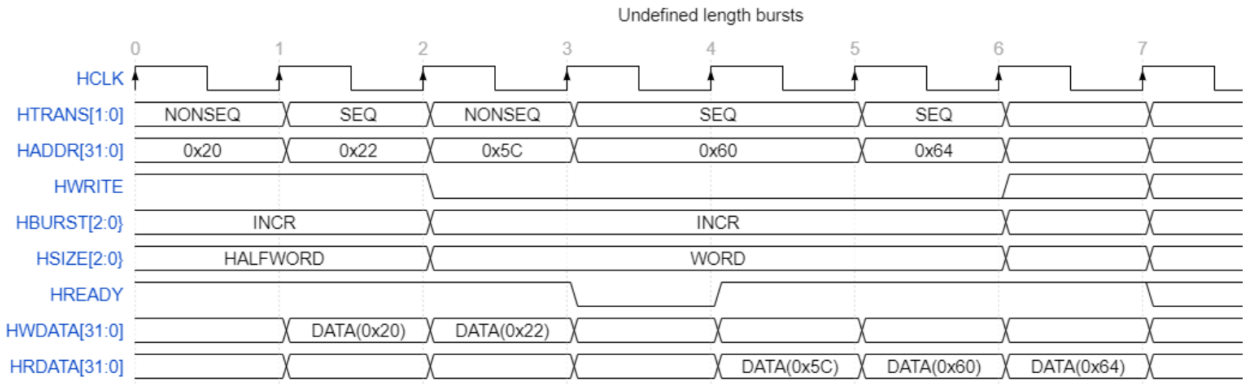
Iată un exemplu de transfer pe jumătăți de cuvinte – așadar cu incremental 2 pentru adrese. Pentru această salvă cu incrementare, adresele depășesc limita multiplu de 16.



```
{signal: [
  {name: 'HCLK',                wave: 'P........',},
  {name: 'HTRANS[1:0]',   wave: '222222222',  data: ['NONSEQ','SEQ','SEQ','SE
Q','SEQ','SEQ','SEQ','SEQ']},
  {name: 'HADDR[31:0]',     wave: '222222222',  data: ['0x34','0x36','0x38','
0x3A','0x3C','0x3E','0x40','0x42']},
  {name: 'HWRITE',            wave: '1.......2'},
  {name: 'HBURST[2:0}',     wave: '2.......2', data:['INCR8']},
  {name: 'HSIZE[2:0}',      wave: '2.......2', data:['HALFWORD']},
  {name: 'HREADY',            wave: '1.......2'},
  {name: 'HWDATA[31:0]',    wave: '222222222',data: ['','DATA(0x34)','DATA(0x
36)','DATA(0x38)','DATA(0x3A)','DATA(0x3C)','DATA(0x3E)','DATA(0x40)','DATA(0
x42)']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Eight-beat incrementing burst, INCR8',
     tick:0,
     every:1
  },}
```

Salvă cu incrementare, de lungime nedefinită - INCR

Lungimea acestor salve nu e predeterminată, transferul terminându-se atunci când Solicitantul decide să termine salva.




```
{signal: [
  {name: 'HCLK',                wave: 'P.......',},
  {name: 'HTRANS[1:0]',   wave: '2222.222',  data: ['NONSEQ','SEQ','NONSEQ','
SEQ','SEQ']},
  {name: 'HADDR[31:0]',     wave: '2222.222',  data: ['0x20','0x22','0x5C','0
x60','0x64']},
  {name: 'HWRITE',            wave: '1.0...22'},
```

```
  {name: 'HBURST[2:0}',     wave: '2.2...22', data:['INCR','INCR']},
  {name: 'HSIZE[2:0}',      wave: '2.2...22', data:['HALFWORD','WORD']},
  {name: 'HREADY',            wave: '1..01..2'},
  {name: 'HWDATA[31:0]',    wave: '22222222',data: ['','DATA(0x20)','DATA(0x2
2)']},
  {name: 'HRDATA[31:0]',    wave: '22222222',data: ['','','','','DATA(0x5C)',
'DATA(0x60)','DATA(0x64)']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Undefined length bursts',
     tick:0,
     every:1
  },}
```

În diagramă au fost prezentate două salve:

- Prima e o salvă de scriere formată din două transferuri pe jumătate de cuvânt, începând de la adresa 0x20. Aceste adrese cresc cu increment 2.
- A doua e o salvă de citire formată din 3 transferuri pe cuvinte începând de la adresa 0x5C. Aceste adrese cresc cu increment 4.

Transferul BUSY cu salvă de lungime fixă

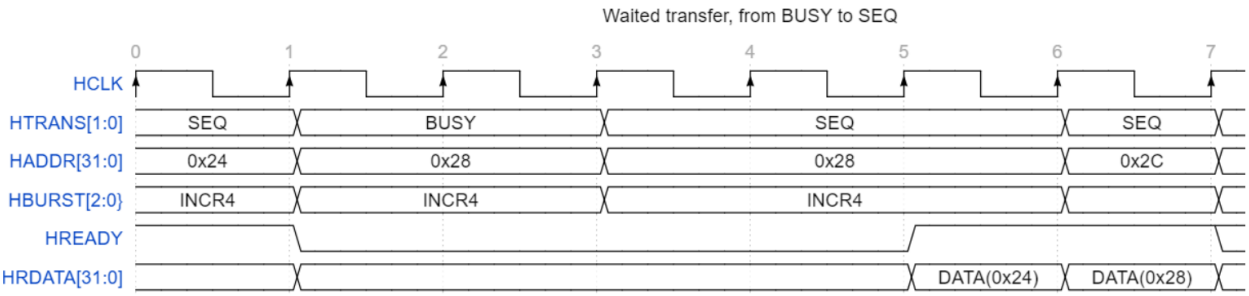
În timpul unui transfer cu stări de așteptare, pentru o salvă de lungime fixă, Solicitantul are permisiunea de a schimba tipul de transfer de la BUSY la SEQ. Când **HTRANS** semnalizează schimbarea tipului de transfer la SEQ, Managerul trebuie să mențină **HTRANS** neschimbat până ce **HREADY** devine ”1”.

Notă:

Deoarece transferurile BUSY se pot insera doar între tactele successive ale unei salve, cele de mai sus nu se aplică unei salve SINGLE, ci numai următoarelor tipuri de salve:

- INCR4, INCR8, și INCR16.
- WRAP4, WRAP8, și WRAP16.

Diagrama de mai jos prezintă un transfer cu stări de așteptare în cadrul unei salve de lungime fixă, cu schimbarea tipului de transfer de la BUSY la SEQ.




```
{signal: [
  {name: 'HCLK',                wave: 'P.......',},
  {name: 'HTRANS[1:0]',    wave: '22.2..22',  data: ['SEQ','BUSY','SEQ','SEQ']},
  {name: 'HADDR[31:0]',     wave: '22.2..22',  data: ['0x24','0x28','0x28','0x2C
']},
  {name: 'HBURST[2:0}',     wave: '22.2..22', data:['INCR4','INCR4','INCR4']},
  {name: 'HREADY',           wave: '10...1.2'},
  {name: 'HRDATA[31:0]',    wave: '22...222',data: ['','','DATA(0x24)','DATA(0x2
8)','']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Waited transfer, from BUSY to SEQ',
     tick:0,
     every:1
  },}
```

Detaliere pe tactele de ceas:

T0-T1: Solicitantul inițiază următorul tact al salvei INCR4 pentru adresare 0x24.

T1-T3: Solicitantul inserează un transfer BUSY la adresa 0x28. Furnizorul inserează stări de așteptare cu HREADY = ”0”.

T3-T4: Solicitantul schimbă tipul de transfer la SEQ și inițiază următorul tact al salvei pentru adresare 0x28.

T4-T6: Cu HREADY = ”0”, Solicitantul menține HTRANS constant.

T5-T6: Tactul pentru adresarea 0x24 se încheie cu HREADY = ”1”.

T6-T7: Al 3-lea tact al5 transferului INCR4 cu adresare 0x28 se încheie și Solicitantul inițiază ultimul tact pentru adresare 0x2C.

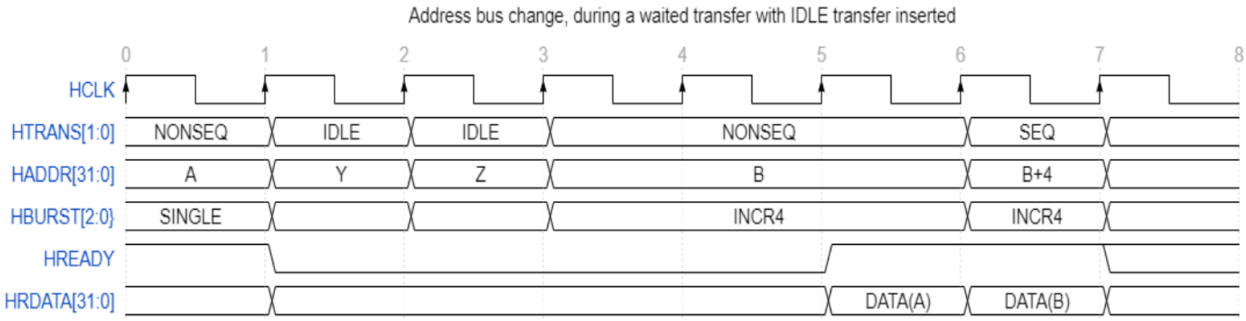
Un alt aspect important al protocolului AHB este permisiunea acordată Solicitantului de a schimba adresele în timpul stărilor de așteptare.

Când Furnizorul cere stări de așteptare, un Solicitant poate face o singură modificare de adresă ; modificarea nu se poate face:

- în timpul unui transfer IDLE ;
- după un răspuns ERROR (cazul Solicitanți Multipli).

- În timpul unui transfer IDLE

În timpul unui transfer cu stări de așteptare, când HTRANS semnalizează o schimbare a tipului de transfer la NONSEQ, Solicitantul trebuie să mențină adresa constantă până când HREADY devine ”1” :




```
{signal: [
  {name: 'HCLK',                wave: 'P.......',},
  {name: 'HTRANS[1:0]',    wave: '2222..22',  data: ['NONSEQ','IDLE','IDLE','N
ONSEQ','SEQ']},
  {name: 'HADDR[31:0]',     wave: '2222..22',  data: ['A','Y','Z','B','B+4
']},
  {name: 'HBURST[2:0}',     wave: '2222..22', data:['SINGLE','','','INCR4','I
NCR4']},
  {name: 'HREADY',            wave: '10...1.2'},
  {name: 'HRDATA[31:0]',    wave: '22...222',data: ['','','DATA(A)','DATA(B)
','']},
  ],
    config: { hscale: 3 },
     head:{
     text:'Address bus change, during a waited transfer with IDLE transfer in
serted',
     tick:0,
     every:1
  },}
```

T0-T1: Solicitantul inițiază o salvă SINGLE cu adresare A.

T1-T2: Solicitantul inserează un transfer IDLE la adresa Y. Furnizorul inserează o stare de așteptare cu HREADY = ”0”.

T2-T3: Solicitantul inserează un transfer IDLE la adresa Z.

T3-T4: Furnizorul schimbă tipul de transfer la NONSEQ și inițiază un transfer INCR4 la adresa B. Nu sunt permise alte modificări de adrese până ce HREADY nu devine ”1”.

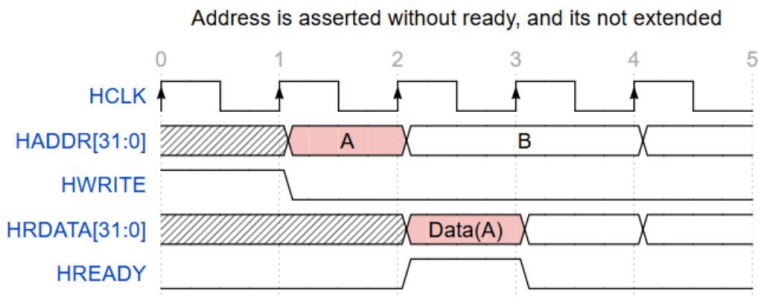
T5-T6: Salva SINGLE la adresa A se încheie cu HREADY = ”1” iar Managerul inițiază primul tact pentru adresa B.

T6-T7: Se încheie primul tact pentru transferul INCR4 la adresa B și Managerul inițiază primul tact pentru adresa B+4

Erori de protocol

Cele mai simple erori de protocol AHB sunt următoarele:

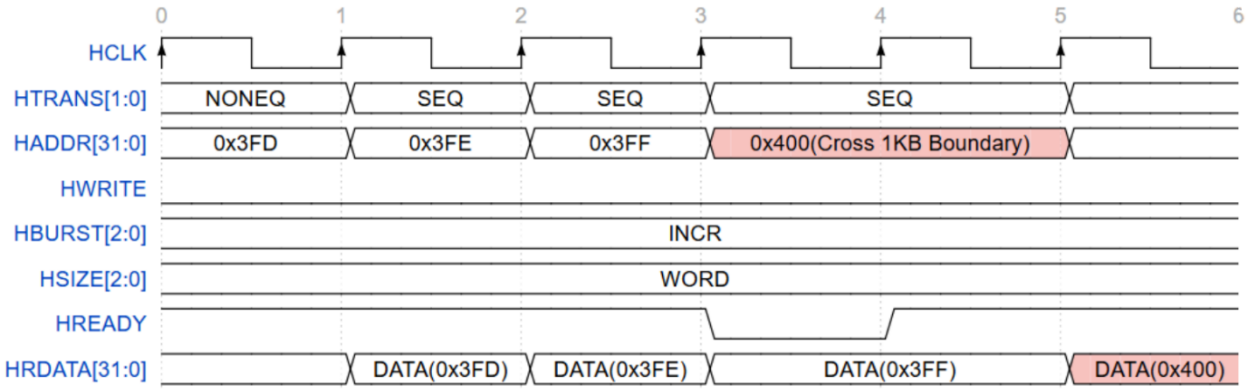
- Furnizorul nu este pregătit să răspundă la adresare, iar Solicitantul face modificări incorecte pe magistrala de adresă:




```
{signal: [
  {name: 'HCLK', wave: 'P....',},
  {name: 'HADDR[31:0]', wave: 'x92.2', data: ['A', 'B']},
  {name: 'HWRITE', wave: '10...'},
  {name: 'HRDATA[31:0]', wave: 'x.922',data: ['Data(A)','']},
  {name: 'HREADY', wave: '0.10.'},
  ],
config: { hscale: 2 },
```

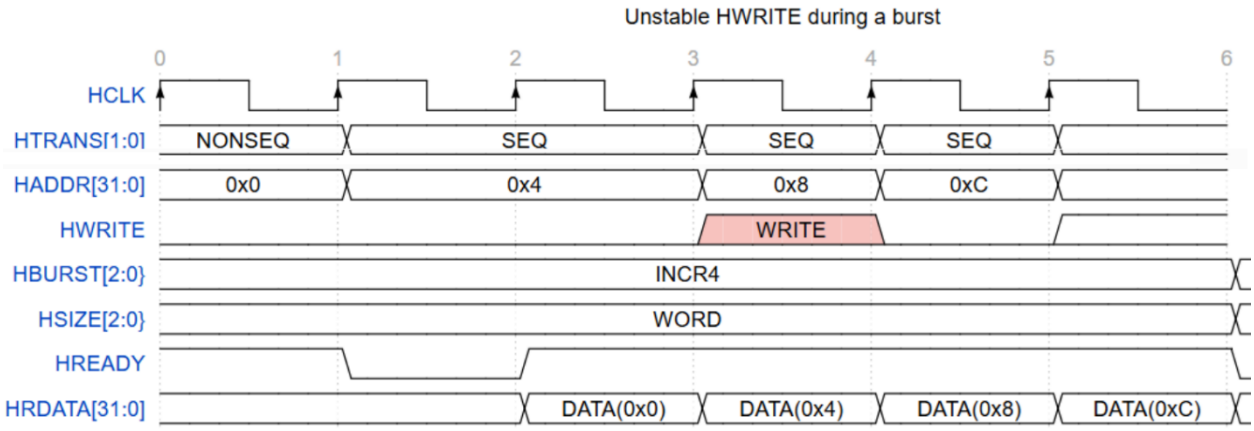
```
head:{
     text:' Address is asserted without ready, and its not extended',
     tick:0,
     every:1
  },}
```

- Solicitantul încearcă să inițieze o salvă cu incrementare care depășește limita de adresare de 1KB.




```
{signal: [
    { name: "HCLK", "wave": "P....." },
    { name: "HTRANS[1:0]", "wave": "2222.2", "data": ["NONEQ", "SEQ", "SEQ",
"SEQ"] },
    { name: "HADDR[31:0]", "wave": "2229.2", "data": ["0x3FD","0x3FE", "0x3FF
", "0x400(Cross 1KB Boundary)"] },
    { name: "HWRITE", "wave": "0....." },
    { name: "HBURST[2:0]", "wave": "2.....", "data": ["INCR"] },
    { name: "HSIZE[2:0]", "wave": "2.....", "data": ["WORD"] },
    { name: "HREADY", "wave": "1..01." },
    { name: "HRDATA[31:0]", "wave": "2222.9", "data": ["", "DATA(0x3FD)", "DA
TA(0x3FE)", "DATA(0x3FF)","DATA(0x400)"] },
  ],
  config: { "hscale": 3 },
  head: {
    text: "Four-beat incrementing burst crossing 1KB boundary",
    tick: 0,
    every: 1
  }
}
```

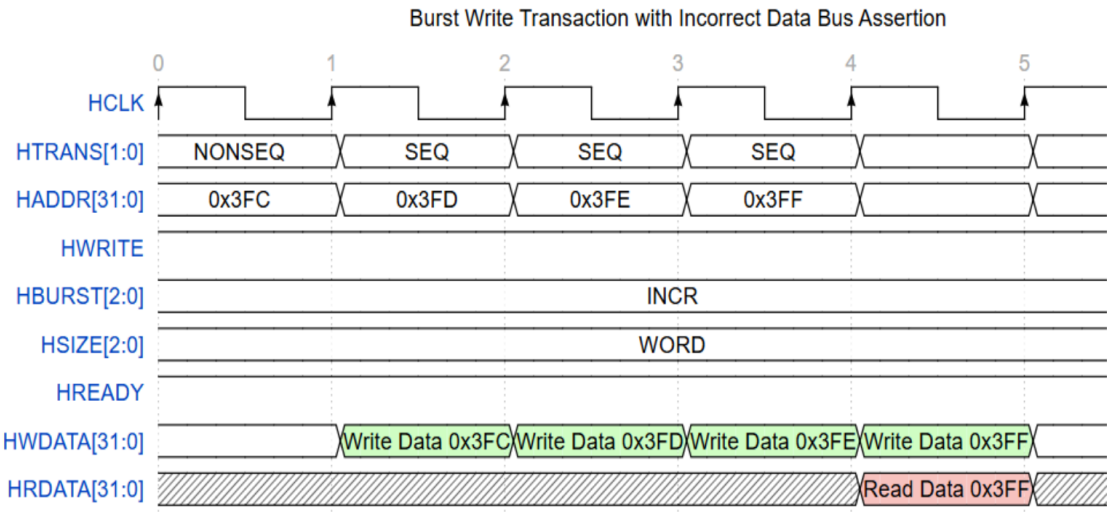
- Pot apărea mai multe erori din starea **HWRITE**, în mijlocul unei salve - când HTRANS indică un transfer SEQ, HWRITE ar trebui să fie stabil și să nu-și schimbe valoarea, HWRITE poate fi setat din nou când începe un alt transfer NONSEQ.




```
{signal: [
  {name: 'HCLK',      wave: 'P.....',},
  {name: 'HTRANS[1:0]',   wave: '22.222',  data: ['NONSEQ','SEQ','SEQ','SEQ
']},
  {name: 'HADDR[31:0]', wave: '22.222',  data: ['0x0','0x4','0x8','0xC']},
  {name: 'HWRITE',    wave: '0..902', data:['WRITE']},
  {name: 'HBURST[2:0}', wave: '2.....2', data:['INCR4']},
  {name: 'HSIZE[2:0}', wave: '2.....2', data:['WORD']},
  {name: 'HREADY',    wave: '101...2'},
  {name: 'HRDATA[31:0]', wave: '2.22222',data: ['','DATA(0x0)','DATA(0x4)','D
ATA(0x8)','DATA(0xC)']},
  ],
  config: { hscale: 3 },
  head:{
     text:'Unstable HWRITE during a burst',
     tick:0,
     every:1
  },}
```

- Alte tipuri de erori pot proveni de la valori **HWDATA** sau **HRDATA** instabile pe magistrală ; de exemplu, în timpul unei salve de citire, datele ar trebui să apară evident pe magistrală dar **HWDATA** nu se activează.

- Un alt tip obișnuit de eroare de protocol este alocarea unei magistrale greșite de citire/scriere.




```
{signal: [
  {name: 'HCLK',     wave: 'P.....',},
  {name: 'HTRANS[1:0]',   wave: '222222',  data: ['NONSEQ','SEQ','SEQ','SEQ
']},
  {name: 'HADDR[31:0]', wave: '222222',  data: ['0x3FC','0x3FD','0x3FE','0x3F
F']},
  {name: 'HWRITE',   wave: '1....2'},
  {name: 'HBURST[2:0}', wave: '2....2', data:['INCR']},
  {name: 'HSIZE[2:0}', wave: '2....2', data:['WORD']},
  {name: 'HREADY',   wave: '1....2'},
  {name: 'HWDATA[31:0]', wave: '277772',data: ['','Write Data 0x3FC','Write D
ata 0x3FD','Write Data 0x3FE','Write Data 0x3FF']},
  {name: 'HRDATA[31:0]', wave: 'x...9x',data: ['Read Data 0x3FF','']},
  ],
config: { hscale: 3 },
head:{
     text:'Incorrect Data Bus Assertion',
     tick:0,
     every:1
  },}
```

Protocolul AHB (Advanced High-performance Bus) suportă o varietate de tranzacții de bază, fiecare concepută pentru a facilita transferul eficient al datelor între dispozitivele Solicitante și cele Furnizoare. Aceste tranzacții includ:

Tranzacții unice de citire/scriere: Acestea sunt cele mai simple forme de tranzacții în care are loc un singur transfer de date între Solicitant și Furnizor. Solicitantul oferă adresa și semnalele de control, iar Furnizorul răspunde cu datele cerute și confirmă operația de scriere.

Tranzacții în salvă: Tranzacțiile în salvă permit transferuri multiple de date într-o singură operațiune, îmbunătățind debitul și eficiența. Solicitantul specifică tipul de salvă (de exemplu, cu incrementare sau cu înfășurare) și dimensiunea, iar Furnizorul răspunde corespunzător. Tranzacțiile în salvă trebuie să respecte reguli specifice, cum ar fi încălcarea unor limite de adresare (de exemplu, 1KB).

Tranzacții secvențiale: Aceste tranzacții implică o serie de transferuri de date către adrese consecutive. Solicitantul inițiază secvența, iar Furnizorul deservește fiecare transfer la rând, menținând integritatea secvenței.

Tranzacții non-secvențiale: În aceste tranzacții, Solicitantul poate specifica accesul la adrese neconsecutive. Fiecare transfer este independent, iar Furnizorul trebuie să poată gestiona eficient schimbările de adresă.

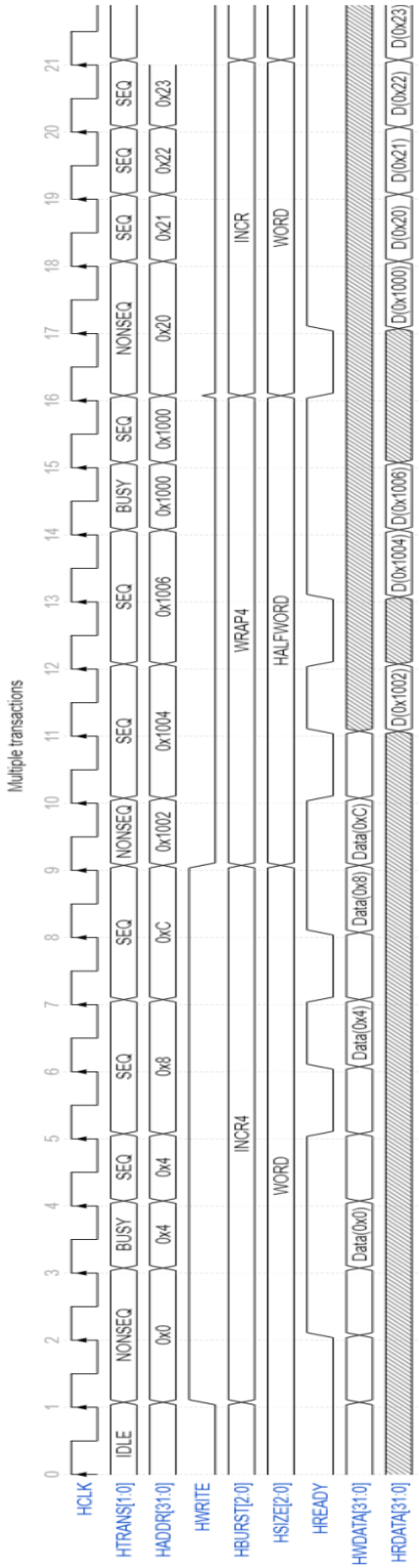
Concluzie

Protocolul AHB (Advanced High-performance Bus) susține o varietate de tipuri de tranzacții, asigurând performanță și eficiență optimă în diferite scenarii.

Totuși, este esențială respectarea regulilor protocolului, cum ar fi menținerea stabilității semnalelor de control și evitarea încălcărilor limitelor de adresare, pentru a preveni erorile și a asigura un transfer fiabil de date.

Gestionarea corectă a acestor tranzacții permite integrarea și funcționarea fără întreruperi a mai multor componente într-un SoC (System-on-Chip) [3, pag. 11].

Alte exemple de tranzacții



```
{"signal": [
    { "name": "HCLK", "wave": "P....................." },
    { "name": "HTRANS[1:0]", "wave": "22.222.2.22.2.222.2222", "data": ["IDLE
","NONSEQ","BUSY","SEQ", "SEQ", "SEQ", "NONSEQ","SEQ", "SEQ", "BUSY","SEQ","N
ONSEQ","SEQ", "SEQ", "SEQ"]},
    { "name": "HADDR[31:0]", "wave": "22.222.2.22.2.222.222", "data": ["","0x
0", "0x4","0x4","0x8", "0xC","0x1002","0x1004","0x1006","0x1000","0x1000","0x
20","0x21","0x22","0x23"]},
    { "name": "HWRITE", "wave": "01.......0......0....."},
    { "name": "HBURST[2:0]", "wave": "22.......2......2....2", "data": ["","I
NCR4","WRAP4","INCR"]},
    { "name": "HSIZE[2:0]", "wave": "2........2......2....2", "data": ["WORD
","HALFWORD","WORD"]},
    { "name": "HREADY", "wave": "0.1..0101.0101..01...."},

    { "name": "HWDATA[31:0]", "wave": "22222222222x..........", "data": ["","
","","Data(0x0)", "","","Data(0x4)","", "Data(0x8)","Data(0xC)"]},
    { "name": "HRDATA[31:0]", "wave": "x....x.....2x22xx22222", "data": ["D(0
x1002)","D(0x1004)","D(0x1006)","D(0x1000)","D(0x20)","D(0x21)","D(0x22)","D
(0x23)"]},
  ],
  "config": { "hscale": 2 },
  "head": {
    "text": "Multiple transactions",
    "tick": 0,
    "every": 1
  }
}
```

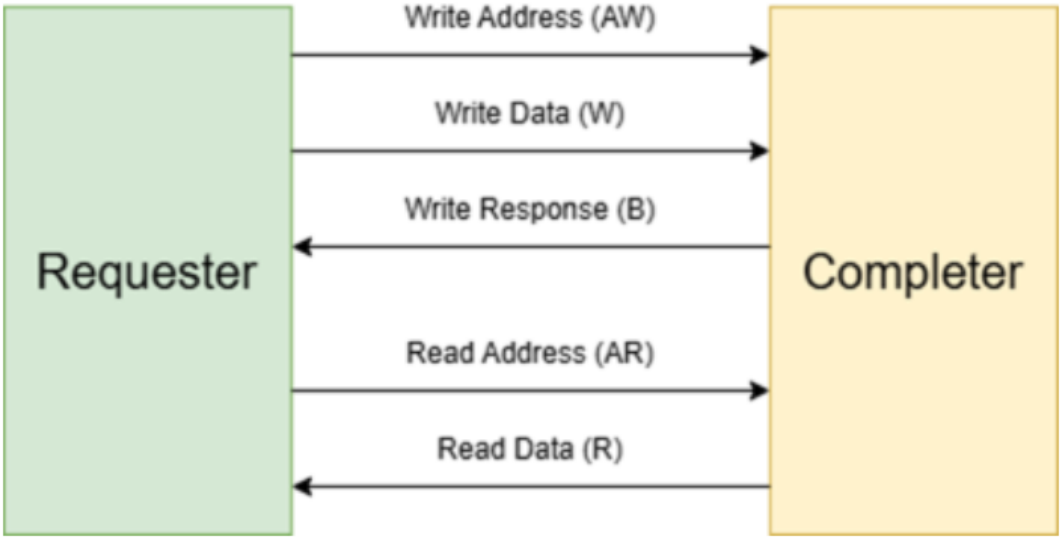
3. AXI (Advanced eXtensible Interface)

Caracteristici

1. **Faze separate pentru adresare/control și date** – AXI decuplează fazele de adresare/control și date în canale separate.
2. **Suport pentru transferuri de date nealiniate, folosind impulsuri de sincronizare (”strobe”) a octeților** – AXI permite accesul la date nealiniate, ceea ce înseamnă că datele pot începe de la orice octet al adresei, mai degrabă decât să fie limitate la adrese aliniate cu cuvintele.
3. **Folosește tranzacții bazate pe salvă (”burst”) fiind suficientă doar adresa de start** – În loc să trimită o nouă adresă pentru fiecare transfer, AXI acceptă tranzacții bazate pe salvă (”rafală”). Se transmite doar adresa de pornire și se calculează adresele ulterioare.
4. **Canale de date separate pentru citire și scriere** – tranzacțiile de citire și scriere utilizează canale separate, permițând operațiuni de citire/scriere simultane.
5. **Suport pentru finalizarea tranzacțiilor fără ordonare** – Răspunsurile pot fi returnate fără ordonare în funcție de disponibilitatea memoriei sau a perifericelor. AXI oferă identificatori de tranzacție (ID-uri pe canalele AR și AW) pentru a urmări răspunsurile.
6. **Permite adăugarea ușoară a registrelor pentru a asigura completarea timpilor** – AXI permite introducerea registrelor tip ”pipeline” (”conductă”) între tranzacții pentru completarea temporizărilor. Fiecare canal funcționează independent, ceea ce înseamnă că registrele pot fi introduse fără a întrerupe protocolul [8, pag. 1].

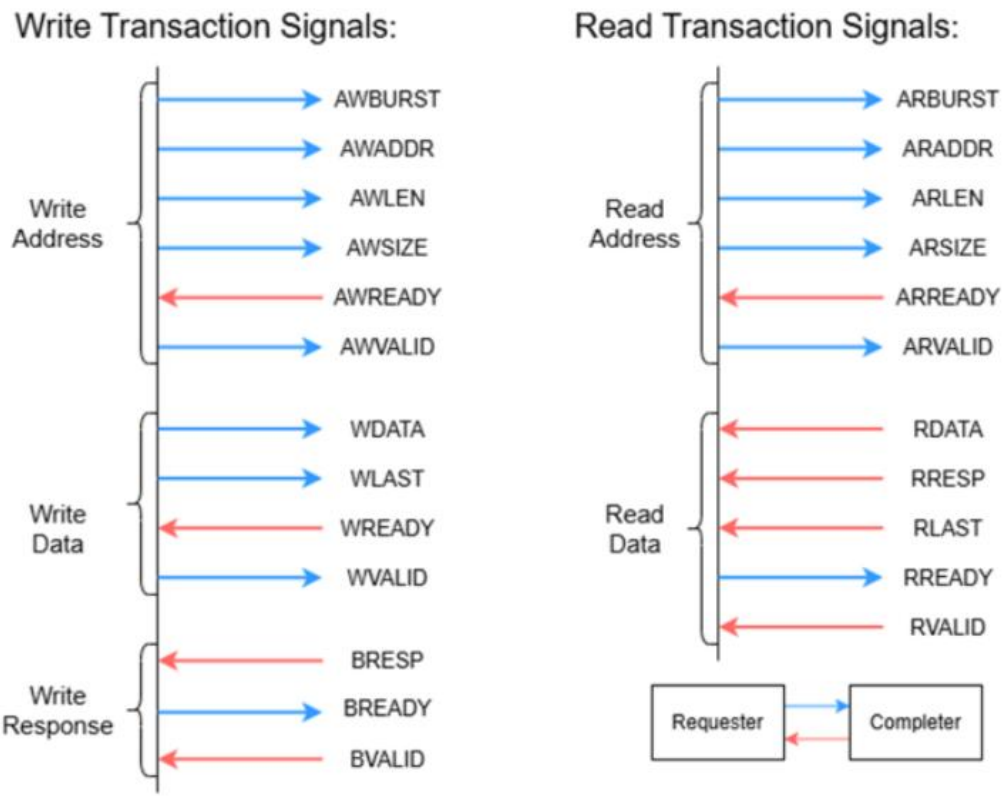
Protocolul AXI [1, pag. 219] este bazat pe salvă (”burst”) și definește cinci canale de tranzacție independente, ca în figura următoare:

- Adresare pentru scriere (AW)
- Date de scris (W)
- Răspuns la scriere (B)
- Adresare de citire (AR)
- Date citite (R)



Diagramă de transfer - între solicitant (Requester) și furnizor (Completer)

Definirea canalului – Așa cum e ilustrat în figura următoare, fiecare dintre cele cinci canale independente constă dintr-un set de semnale de informație și semnale VALID și READY (”Disponibil”) care oferă un mecanism de ”hand-shaking” (”strângeri de mâini”) bidirecțional. Sursa de informații utilizează semnalul VALID pentru a anunța când sunt disponibile informații valide despre adresă, date sau controlul pe canal. Destinatarul folosește semnalul READY pentru a arăta când e gata pentru a accepta informația. Atât canalul de date de citire, cât și canalul de date de scriere includ, de asemenea, un semnal LAST (”Ultimul”) pentru a indica transferul elementului de date final într-o tranzacție.



Semnalizare în tranzacțiile de Scriere și Citire

Protocolul AXI necesită asigurarea următoarelor relații:

- Un răspuns la scriere trebuie să urmeze întotdeauna ultimul transfer de scriere (dintr-o tranzacție de scriere).
- Datele citite trebuie să urmeze întotdeauna adresa de citire a datelor.
- Semnalizările ”handshake” ale canalului trebuie să respecte următoarele reguli de dependență:
    - Semnalul VALID al interfeței AXI care trimite informații nu trebuie să depindă de semnalul READY al interfeței AXI care primește aceste informații.
    - O interfață AXI care primește informații poate aștepta până când detectează un semnal VALID înainte de a genera semnalul READY corespunzător

Structura adresei

1. Protocolul AXI este bazat pe salve (rafale) – ”burst”. Solicitantul începe fiecare salvă cu transmiterea (către furnizor) a informațiilor de control și a adresei primului octet din tranzacție. Pe măsură ce decurge salva, furnizorul (”respondentul”) trebuie să calculeze adresele transferurilor ulterioare (din salvă).
2. O salvă nu trebuie să depășească o limită de adresare de 4 KB. Această interdicție împiedică o salvă să adreseze de fapt doi sau mai mulți furnizori. De asemenea, se limitează numărul de incrementări de adrese pe care trebuie să le accepte un furnizor.
3. AXI are următoarele reguli care guvernează utilizarea salvelor:
   - Pentru salvă cu revenire (ciclică – a adresei) – ”wrapping burst”, lungimea salvei trebuie să fie 2, 4, 8 sau 16.
   - Terminarea anticipată a salvelor nu este acceptată.

AxLEN descrie lungimea (”LENgth”) a tranzacției – numărul de transferuri.

- AXI3: Lungimea salvei = AxLEN[3:0] + 1; (interval de 1-16 transferuri într-o tranzacție)

- AXI4: Lungimea bustului = AxLEN[7:0] + 1; (interval de 1-256 transferuri într-o tranzacție)

AxSize descrie numărul maxim de octeți din fiecare transfer de date.

AxSIZE[2:0]	**Octeți în transfer**
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

AxBURST descrie tipul de salvă al tranzacției:

AxBURST[1:0]	Tip de salvă	Descriere	Utilizare
0b00	FIXED	Adresa este aceeași pentru fiecare transfer în salvă	Acces repetat la aceeași locație, util pentru FIFO (stiva First In–First Out)
0b01	INCR	Adresa pentru fiecare transfer în salvă este o incrementare a adresei pentru transferul anterior. Valoarea de incrementare depinde de dimensiunea transferului.	Accesează memoria secvențială normală.
0b10	WRAP	O salvă cu revenire este similară cu o salvă incrementală, cu excepția faptului că adresa revine ciclic la o adresă inferioară dacă este atinsă o limită superioară (de adresare).	Acces la o linie (locație) din Cache
0b11	(Rezervat)		

Următoarele restricții se aplică salvelor cu revenire:

- Adresa de început trebuie să corespundă cu dimensiunea fiecărui transfer.
- Lungimea salvei trebuie să fie de 2, 4, 8 sau 16 transferuri.

O salvă cu revenire decurge în felul următor:

- Cea mai mică adresă utilizată de salvă este aliniată la dimensiunea totală a datelor care urmează să fie transferate, adică la ((dimensiunea fiecărui transfer în salvă) × (numărul de transferuri în salvă)). Această adresă este definită ca *limita de încadrare*.

- După fiecare transfer, adresa crește în același mod ca și pentru o salvă INCR. Cu toate acestea, dacă această adresă incrementată este ((mărimea ciclului) + (dimensiunea totală a datelor care urmează să fie transferate)), atunci adresa revine la valoarea mărimii ciclului.

- Primul transfer din salvă poate utiliza o adresă care este mai mare decât mărimea ciclului, sub rezerva restricțiilor care se aplică salvelor cu revenire (WRAP). Adresa se încadrează pentru orice salvă WRAP atunci când prima adresă este mai mare decât mărimea ciclului.

Structura răspunsului la citire și scriere

Protocolul AXI oferă semnalizare de răspuns atât pentru tranzacțiile de citire, cât și pentru cele de scriere:

- Pentru tranzacțiile de citire, informațiile de răspuns de la un subsistem ”Slave” (”subordonat”) sunt semnalizate pe canalul de date de citire.

- Pentru tranzacțiile de scriere, informațiile de răspuns sunt semnalizate pe canalul de răspuns la scriere.

Răspunsurile sunt semnalizate prin:

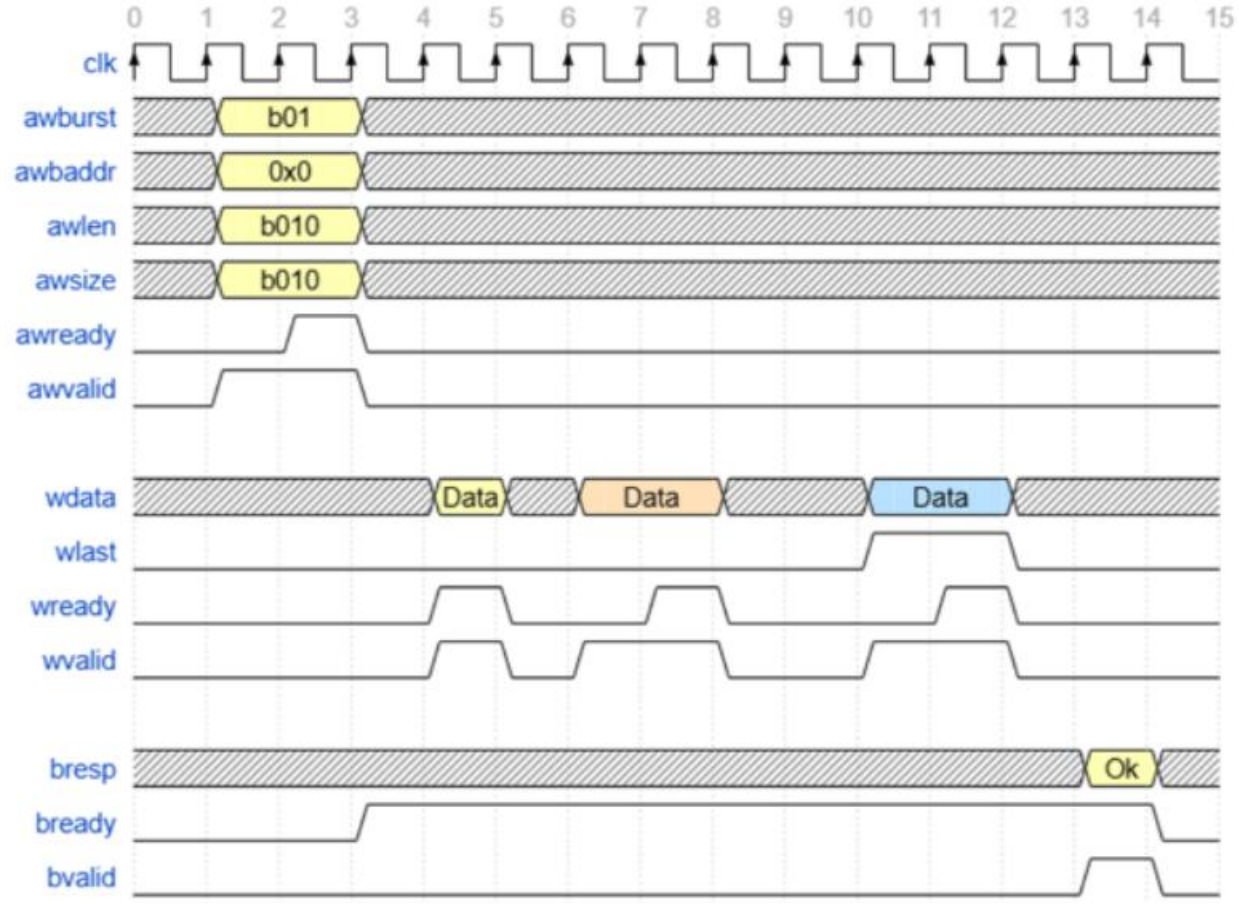
- RRESP[1:0], pentru transferuri de citire.

- BRESP[1:0], pentru transferuri de scriere.

RRESP[1:0] / BRESP[1:0]	**Răspuns**	**Descriere**
0b00	OKAY	- Succesul unui acces normal. - Eșecul unui acces exclusiv. - Un acces exclusiv la un furnizor care nu acceptă acces exclusiv.
0b01	EXOKAY	- Indică succesul unui acces exclusiv. Acest semnal poate fi dat numai ca răspuns la o citire sau scriere exclusivă.
0b10	SLVERR	- Indică o tranzacție nereușită. Pentru a simplifica monitorizarea și depanarea sistemului, specificațiile recomandă ca acest răspuns să fie utilizat numai pentru condiții de eroare și nu pentru semnalizarea normală a unor evenimente așteptate. Exemple de condiții de eroare furnizor sunt: - FIFO sau umplere buffer / sub-utilizare buffer. - Dimensiune de transfer neacceptată. - Scriere încercată într-o locație din care se poate doar citi (”read-only”). - expirare timp (”timeout”) la nivel ”Slave”. - încercare de acces la o funcție dezactivată sau oprită.
0b11	DECERR	- Indică faptul că interfața nu poate decoda cu succes un acces la furnizor. - Protocolul AXI necesită ca toate transferurile de date pentru o tranzacție să fie finalizate, chiar dacă apare o eroare. Orice componentă care oferă un răspuns DECERR trebuie să îndeplinească această cerință.

Exemple de tranzacții

Tranzacție de scriere



Exemplu – salvă de scriere

Figura de mai sus reprezintă o tranzacție de scriere, după cum urmează:

Pe canalul de adresă de scriere:

- AwBURST[1:0] = b01; (salvă incrementală)

- AwLEN[7:0] = b010; (3 transferuri)

- AwSIZE[2:0] = b010; (4 octeți într-un transfer)

- La primul ciclu de ceas, salva este inițiată prin setarea "awvalid" SUS.

- La al 2-lea ciclu de ceas se semnalizează "awready", ceea ce înseamnă că furnizorul este gata să primească o salvă.

- La al 3-lea ciclu de ceas, semnalizarea ”hand-shake" este finalizată, atât "awready", cât și "awvalid" sunt SUS, furnizorul ar trebui să înceapă să deservească salva.

Pe canalul de date de scriere:

- La ciclurile de ceas 4, 6, 10, datele sunt trimise către furnizor cu semnal "wvalid".
- La ciclurile de ceas 5, 8, 12 ore, "awready" și "awvalid" sunt ambele SUS, astfel încât are loc semnalizarea ”hand-shake” de transfer.

- La al 10-lea ciclu de ceas se setează "wlast", ceea ce indică faptul că acesta este ultimul transfer în salvă.

Pe canalul de date de răspuns la scriere:

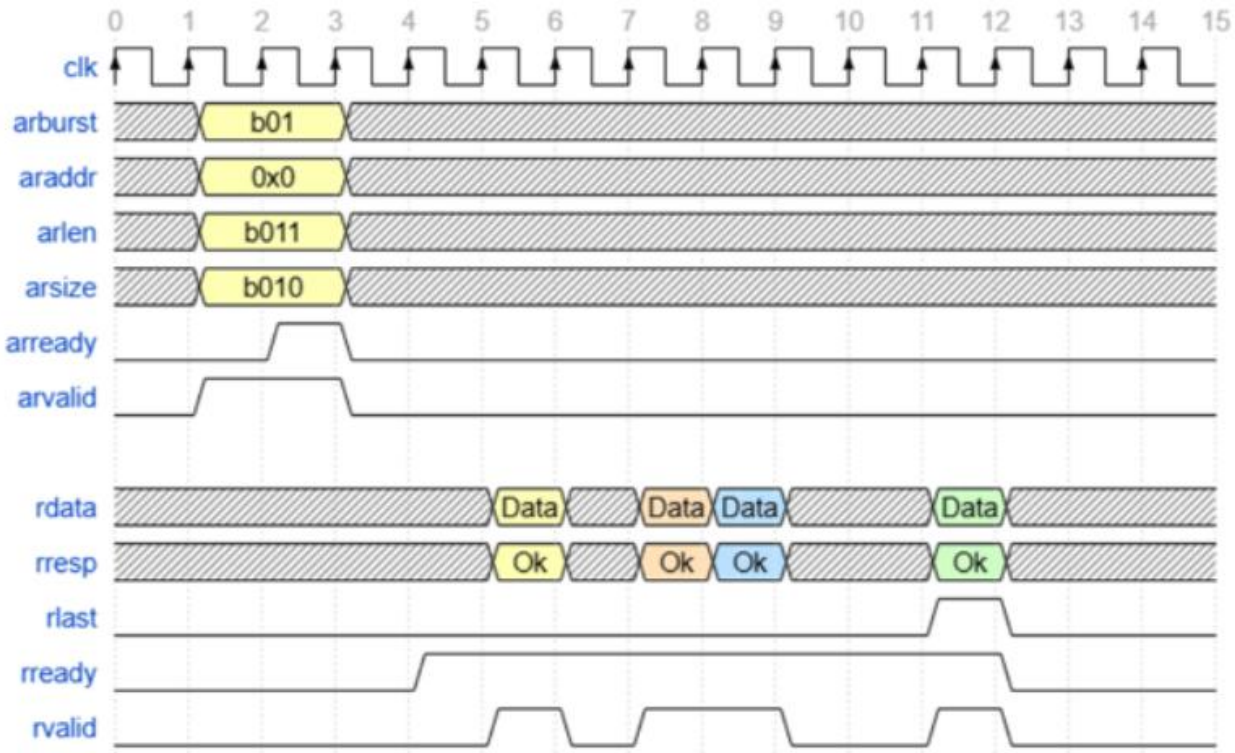
- La al 13-lea ciclu de ceas, furnizorul trimite un răspuns cu semnale "bresp" (care indică "OK" – scrierea a reușit) și "bvalid".

- Solicitantul setează semnalul "bready" atunci când este gata să primească răspunsul, în acest caz, în al 3-lea ciclu de ceas.

```
{signal: [
{name: 'clk'    , wave: 'P.............
'                                    },
{name: 'awburst', wave: 'x3.x..........' , data: ['b01
']                    },
{name: 'awbaddr', wave: 'x3.x..........' , data: ['0x0
']                    },
{name: 'awlen'  , wave: 'x3.x..........' , data: ['b010
']                   },
{name: 'awsize' , wave: 'x3.x..........' , data: ['b010
']                   },
{name: 'awready', wave: '0.10..........
'                                    },
{name: 'awvalid', wave: '01.0..........
'                                    },
{                                                                    }
,
{name: 'wdata'  , wave: 'x...3x4.x.5.x..' , data: ['Data'  , 'Data', 'Data
']},
{name: 'wlast'  , wave: '0.........1.0..
'                                    },
{name: 'wready' , wave: '0...10.10..10..
```

```
'                                    },
{name: 'wvalid' , wave: '0...101.0.1.0..
'                                    },
{                                                                    }
,
{name: 'bresp'  , wave: 'x...........3x' , data: ['Ok
']                    },
{name: 'bready' , wave: '0..1.........0
'                                    },
{name: 'bvalid' , wave: '0...........10
'                                    },
]                ,
head:{
 tick: 0,
 every:1
} ,
}
```

Tranzacția de citire



Exemplu de salvă de citire

Figura anterioară reprezintă o tranzacție de citire, după cum urmează:

Pe canalul de citire a adresei:

- ArBURST[1:0] = b01; (salvă incrementală)

- ArLEN[7:0] = b011; (4 transferuri) - ArSIZE[2:0] = b010; (4 octeți într-un transfer)

- La primul ciclu de ceas, salva este inițiată prin setarea "arvalid" SUS;

- La al 2-lea ciclu de ceas este setat "arready", ceea ce înseamnă că furnizorul este gata să primească o salvă;

- La al 3-lea ciclu de ceas semnalizarea ”hand-shake” este finalizată, atât "arready" cât și "arvalid" sunt SUS, furnizorul ar trebui să înceapă să deservească salva;

Pe canalul de date de citire:

- Solicitantul setează semnalul "rready" atunci când este gata să primească date, în acest caz, în al 4-lea ciclu de ceas;

- La ciclurile de ceas 5, 7, 8, 11 datele sunt trimise de furnizor cu semnalele "rresp" (care indică "OK" – citirea a reușit) și "rvalid";

- La ciclurile de ceas 6, 8, 9 și 12, "arready" și "arvalid" sunt ambele SUS, astfel încât are loc semnalizarea ”hand-shake” de transfer, transferurile 2 și 3 fiind ”back to back”;

- La al 11-lea ciclu de ceas este setat "rlast", ceea ce indică faptul că acesta este ultimul transfer în salvă.

```
{signal: [
{name: 'clk'    , wave: 'P.............
'                                       },
{name: 'arburst', wave: 'x3.x..........' , data: ['b01
']                          },
{name: 'araddr' , wave: 'x3.x..........' , data: ['0x0
']                          },
{name: 'arlen'  , wave: 'x3.x..........' , data: ['b011
']                         },
{name: 'arsize' , wave: 'x3.x..........' , data: ['b010
']                          },
{name: 'arready', wave: '0.10..........
'                                       },
{name: 'arvalid', wave: '01.0..........
'                                       },
{
```

```
},
{name: 'rdata'  , wave: 'x....3x45x.7x..' , data: ['Data'  , 'Data', 'Data',
'Data']},
{name: 'rresp'  , wave: 'x....3x45x.7x..' , data: ['Ok'    , 'Ok'  , 'Ok'  ,
'Ok']  },
{name: 'rlast'  , wave: '0.........10..
'                                        },
{name: 'rready' , wave: '0...1.......0..
'                                        },
{name: 'rvalid' , wave: '0....101.0.10..
'                                        },
]                 ,
head:{
 tick: 0 ,
 every:1
} ,
}
```

Tranzacții restante O tranzacție restantă este o tranzacție care a fost inițiată, dar nu a primit încă un răspuns de la furnizor. AXI acceptă mai multe tranzacții restante, permițând solicitantului să emită mai multe solicitări fără a aștepta finalizarea fiecărei înainte de a o trimite pe următoarea. Această capabilitate crește eficiența și randamentul sistemului. Numărul de tranzacții restante care pot fi gestionate este determinat de construcția interfeței AXI și de capabilitățile dispozitivelor furnizor.

Identificatori de tranzacție Un solicitant poate utiliza identificatori de tranzacție pentru a identifica tranzacții separate care trebuie efectuate în ordine. Toate tranzacțiile cu aceeași valoare AXI ID trebuie să rămână ordonate (”tranzacții ordonate”), dar nu există nicio restricție privind ordonarea tranzacțiilor cu valori ID diferite (tranzacții în afara ordinii). Prin utilizarea ID-urilor AXI, un solicitant poate emite tranzacții fără a aștepta finalizarea tranzacțiilor anterioare (tranzacții restante). Acest lucru poate îmbunătăți performanța sistemului, deoarece permite procesarea paralelă a tranzacțiilor [2, pag. 217].

Canal de tranzacție	**ID tranzacție**
Canal adresă scriere	AWID
Canal scriere date *(implementat doar în AXI3)*	WID
Canal răspuns scriere	BID
Canal adresă citire	ARID
Canal citire date	RID

Ordonarea citirii datelor

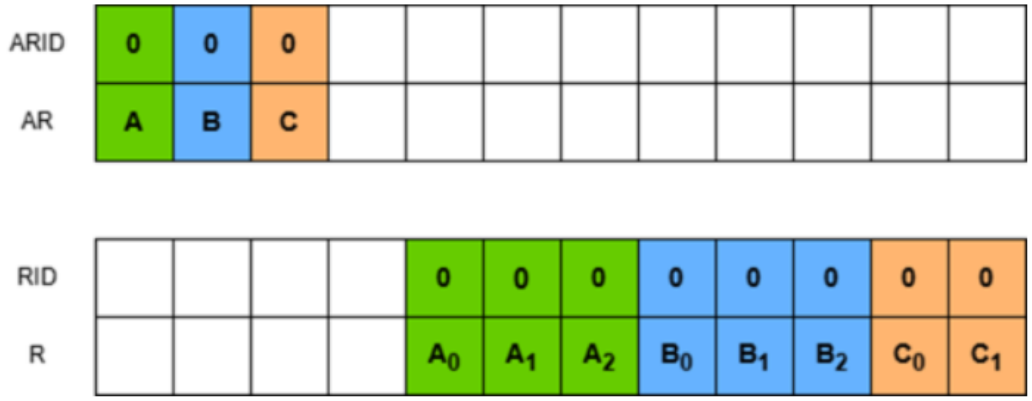
1. Subsistemul ”Slave” (”Subordonat”) - trebuie să se asigure că valoarea RID a oricăror date returnate se potrivește cu valoarea ARID a adresei la care răspunde.
2. Interfața trebuie să se asigure că datele citite dintr-o secvență de tranzacții cu aceeași valoare ARID care vizează diferiți ”subordonați” (”Slaves”) sunt primite de controller în ordinea în care a emis adresele.
3. Adâncimea de reordonare a datelor de citire este numărul de adrese în așteptare în sub-sistemul ”Slave” care pot fi reordonate. Un subordonat ”Slave” care procesează toate tranzacțiile în ordine are o adâncime de reordonare a datelor de citire egală cu unu. Adâncimea de reordonare a datelor citite este o valoare statică care trebuie specificată de proiectantul sub-sistemului ”Slave” (subordonat).
4. Nu există niciun mecanism pe care un controller să-l poată folosi pentru a determina adâncimea de reordonare a datelor de citire a unui sub-sistem ”Slave” (subordonat).

Tranzacție de citire în afara ordinii:



1. Tranzacțiile cu ID-uri diferite sunt trimise de către solicitant
2. Furnizorul returnează datele într-o ordine diferită de cea în care au fost solicitate

Tranzacție ordonată de citire:

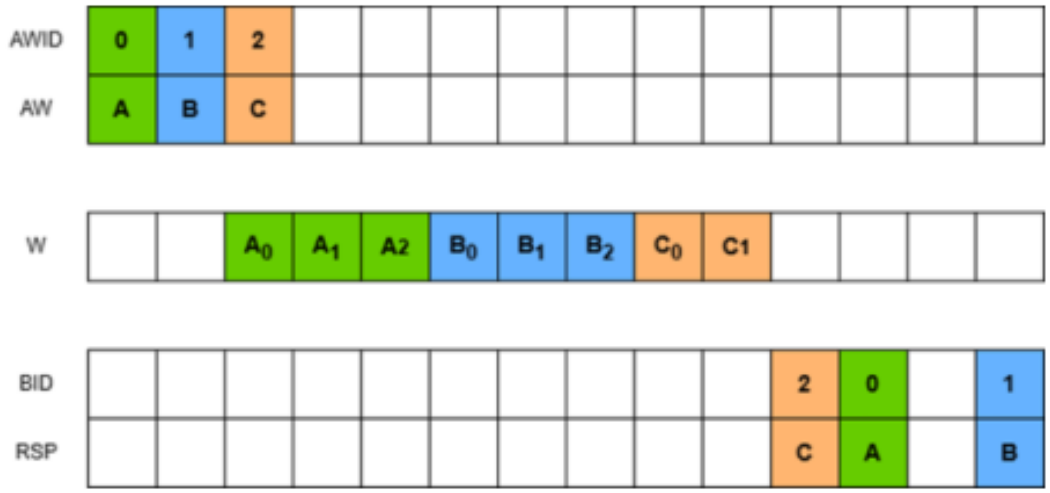


1. Tranzacțiile cu același ID sunt trimise de solicitant
2. Furnizorul returnează datele în ordinea în care au fost solicitate

Ordonarea scrierii datelor

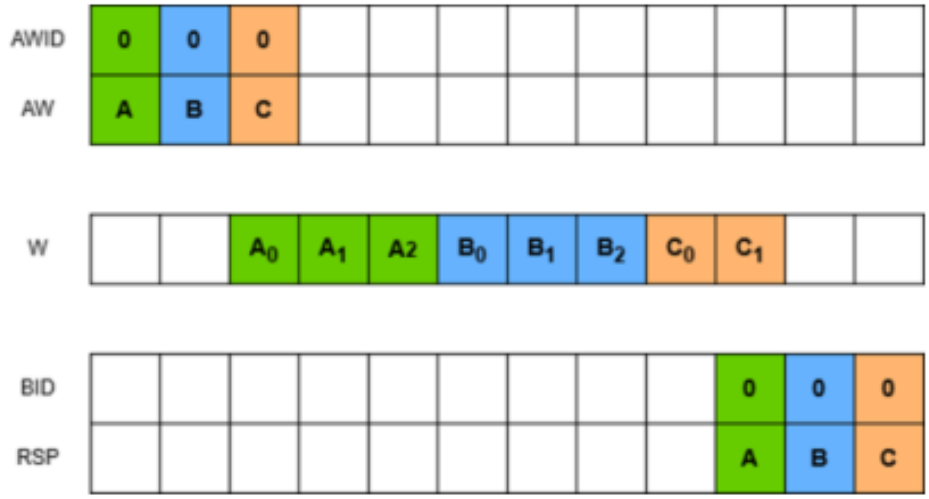
1. Un controller trebuie să trimită date la scriere în aceeași ordine în care emite adresele de tranzacție.
2. O interfață care combină tranzacții de scriere de la diferite controllere trebuie să asigure redirecționarea datelor la scriere în ordinea adresării.
3. Intercalarea datelor de scriere cu ID-uri diferite a fost permisă în AXI3, dar e considerată depășită în AXI4 și versiunile ulterioare. Specificațiile AMBA AXI și ACE versiunea F conțin mai multe detalii despre intercalarea datelor la scriere.

Tranzacție de scriere în afara ordinii:



1. Tranzacțiile cu ID-uri diferite sunt trimise de către solicitant, datele fiind trimise în aceeași ordine ca și adresele;
2. Furnizorul răspunde într-o ordine diferită de cea în care au fost efectuate tranzacțiile de scriere.

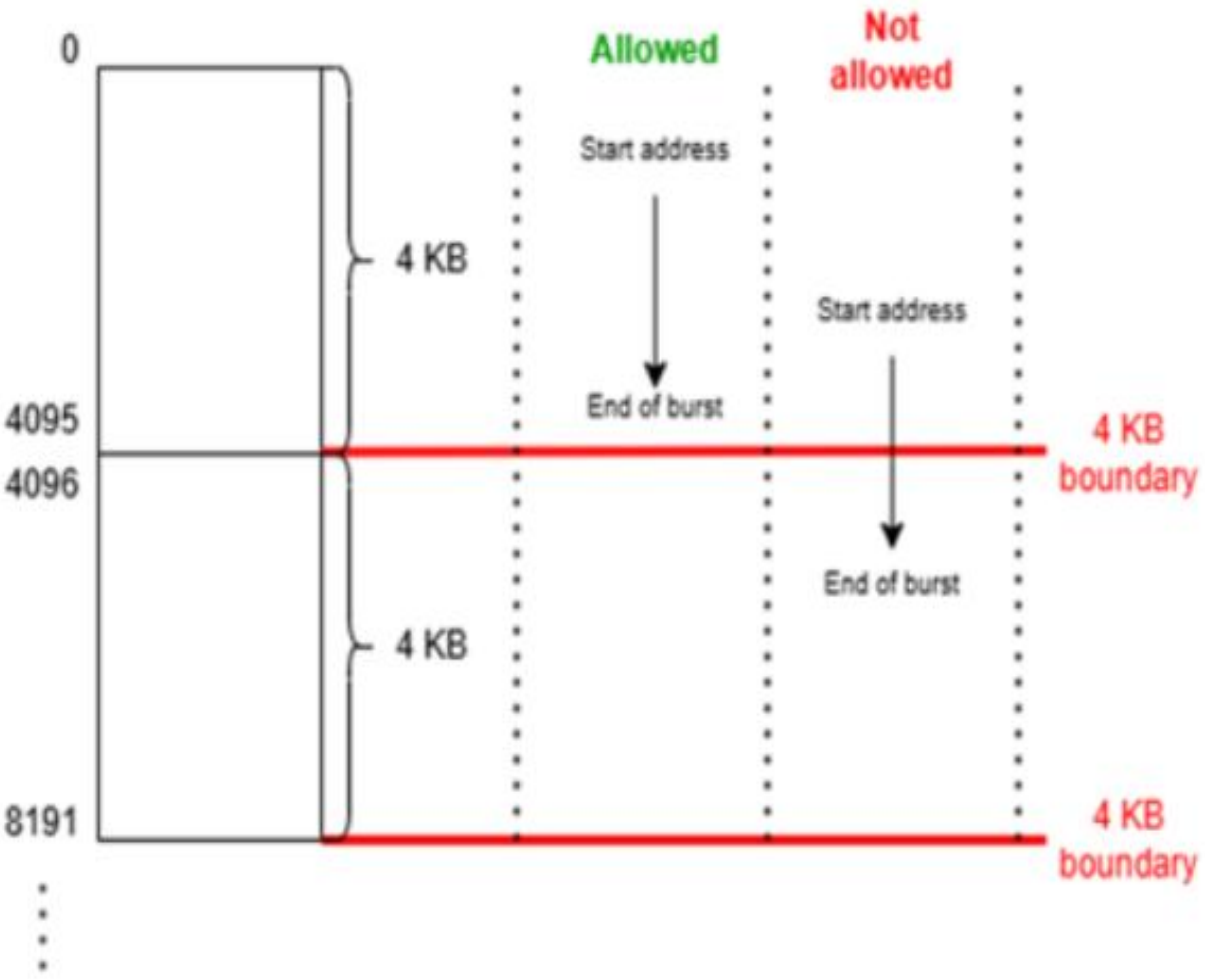
Tranzacție ordonată de scriere:



1. Tranzacțiile cu același ID sunt trimise de către solicitant, datele fiind trimise în aceeași ordine ca și adresele;
2. Furnizorul răspunde în aceeași ordine în care au fost efectuate tranzacțiile de scriere.

Limita de adresare de 4KB

O salvă nu trebuie să depășească o limită de adresare de 4 KB. Această interdicție este în vigoare pentru a evita posibilitatea de a accesa doi sau mai mulți furnizori în timpul unei salve.


0
Allowed
Not allowed
Start address
4 KB
Start address
End of burst
4095
4096
4 KB boundary
End of burst
4 KB
8191
4 KB boundary

4. ACE (AXI Coherency Extensions)

Caracteristici

1. Suport pentru coerența Cache

Protocolul de coerență ACE asigură că toți solicitanții respectă corectitudinea datelor în orice locație de adresă dată, asigurând că există o singură copie ori de câte ori apare o scriere la respectiva locație [1, pag. 209]. După fiecare stocare într-o locație, alți solicitanți pot obține o nouă copie a datelor pentru propria memorie cache locală – doar în acest fel va fi permisă existența mai multor copii. ACE extinde protocolul AXI pentru a susține coerența cache în sisteme multi-core [8, pag. 1]. Se asigură așadar că se menține o vizualizare coerentă a memoriei chiar dacă, astfel, sunt mai multe cache într-un sistem [5, pag. 222].

2. Canalele ”Snoop”

ACE introduce canale suplimentare pentru operațiunile de monitorizare (supraveghere - ”snoop”), care sunt utilizate pentru a menține coerența între cache-uri. Aceste canale includ:

- Adresa Snoop (AC – ”Address Channel”)

- Răspunsul Snoop (CR)

- Datele Snoop (CD)

În plus față de acestea, ACE păstrează și cele cinci canale de tranzacție independente de la AXI:

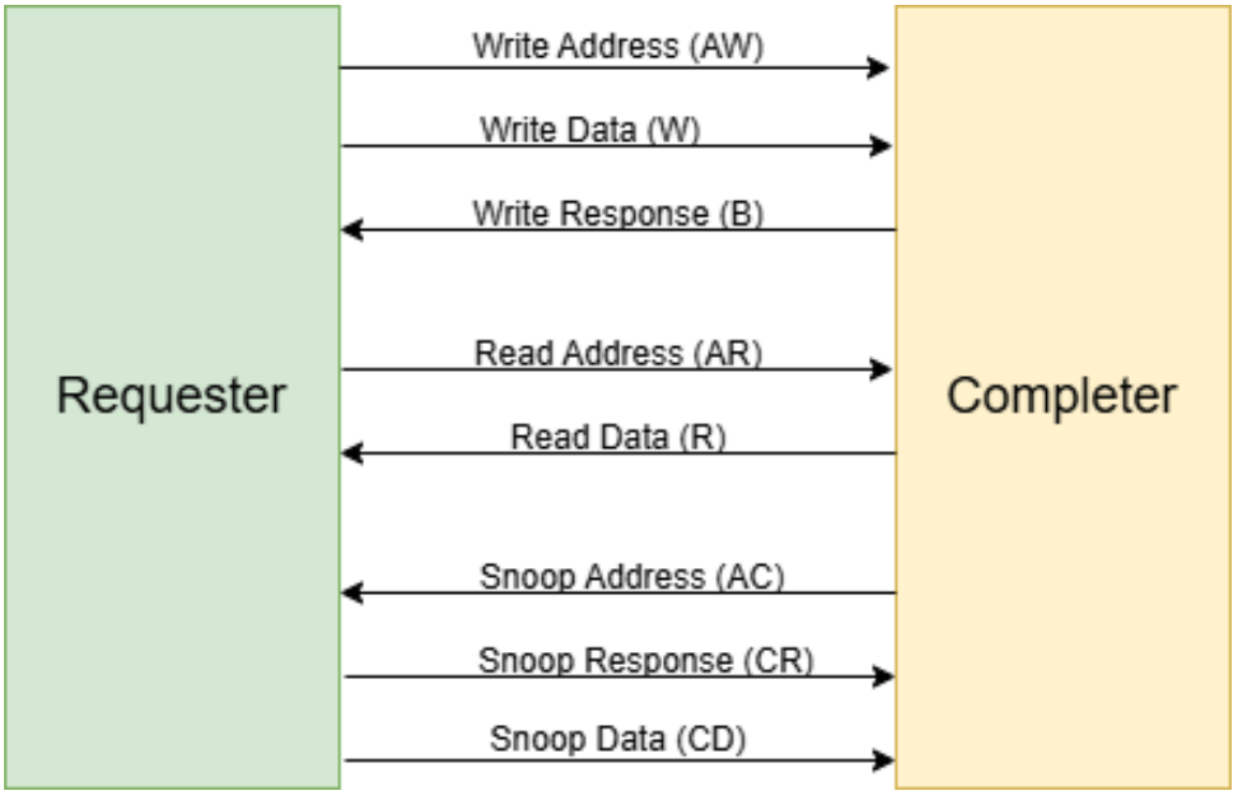
- Adresă de scriere (AW)

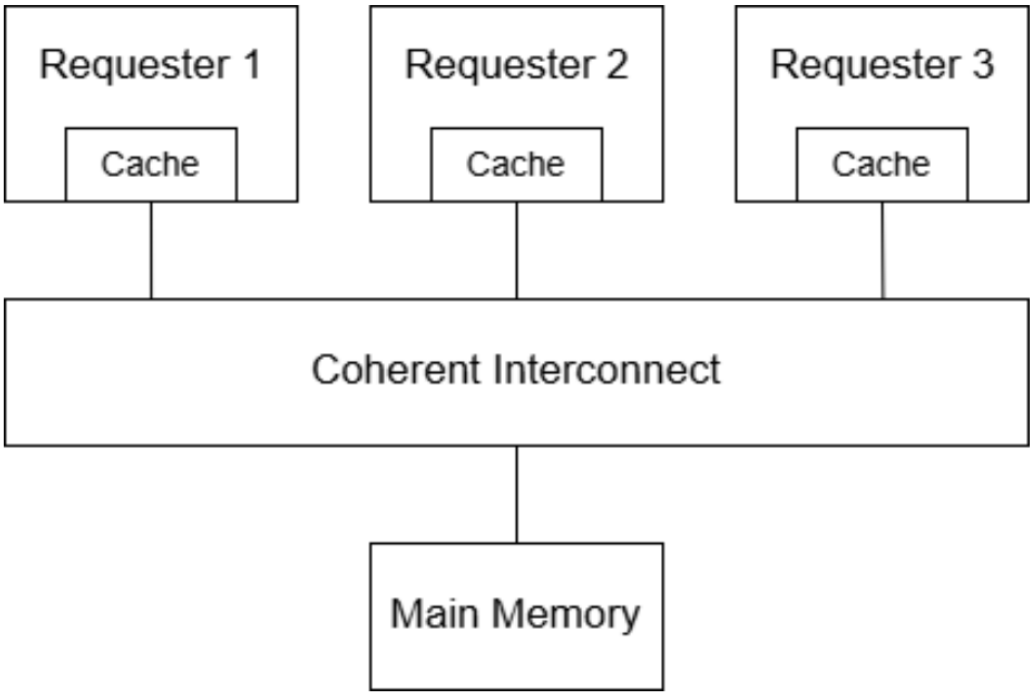
- Date de scriere (W)

- Răspuns de scriere (B)

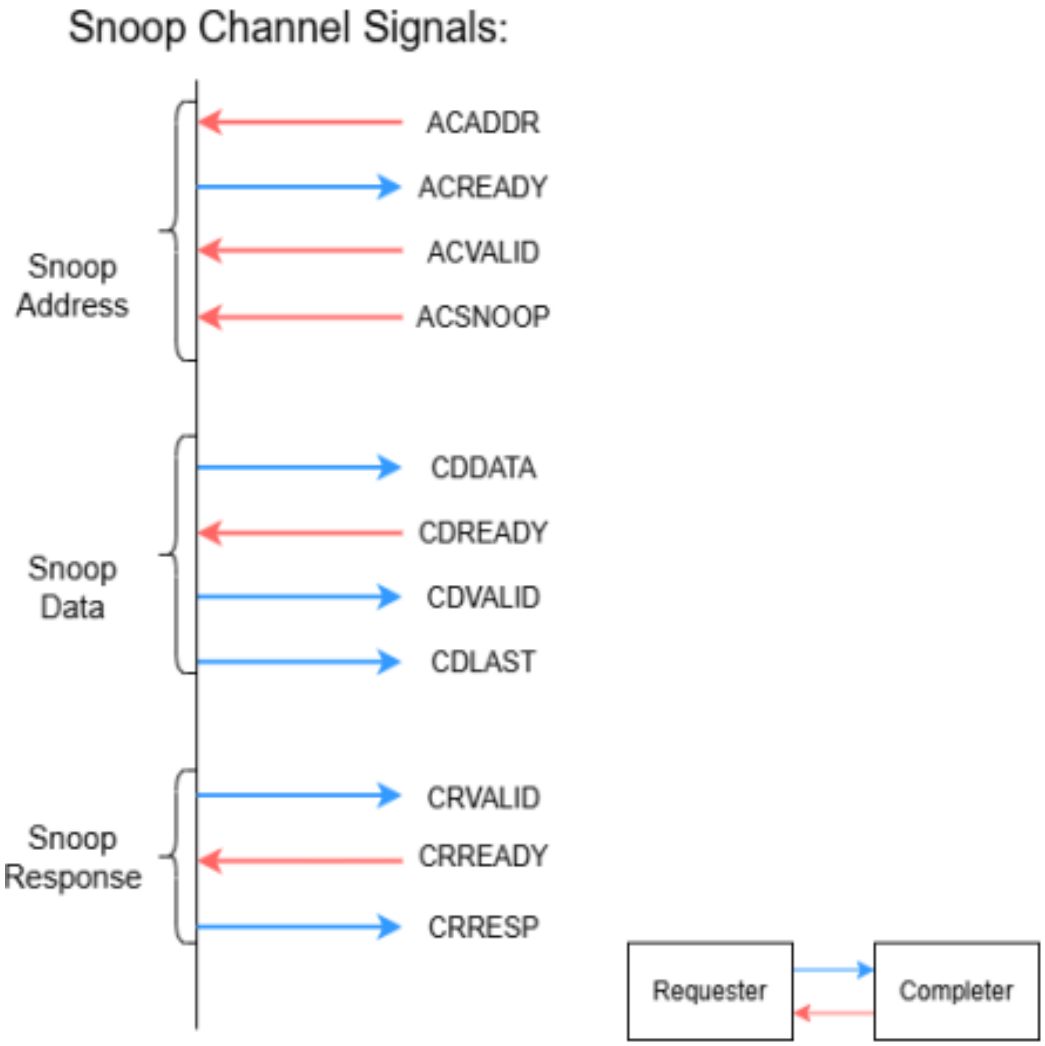
- Adresă de citire (AR)

- Date de citire (R)





Sistemul de Coerență



Canal	Semnal	Lățime	Sursă	Descriere
Canal adresare la scriere	AWADDR	[31:0]	Solici-tant	Adresa primului transfer dintr-o tranzacție de scriere.
	AWID	[3:0]	Solici-tant	Etichetă de identificare pentru o tranzacție de scriere.
	AWBURST	[1:0]	Solici-tant	Tip de salvă - indică modul în care se modifică adresa după fiecare transfer dintr-o tranzacție de scriere.
	AWCACHE	[3:0]	Solici-tant	Indică modul în care este stabilită o tranzacție de scriere pentru a avansa în sistem.

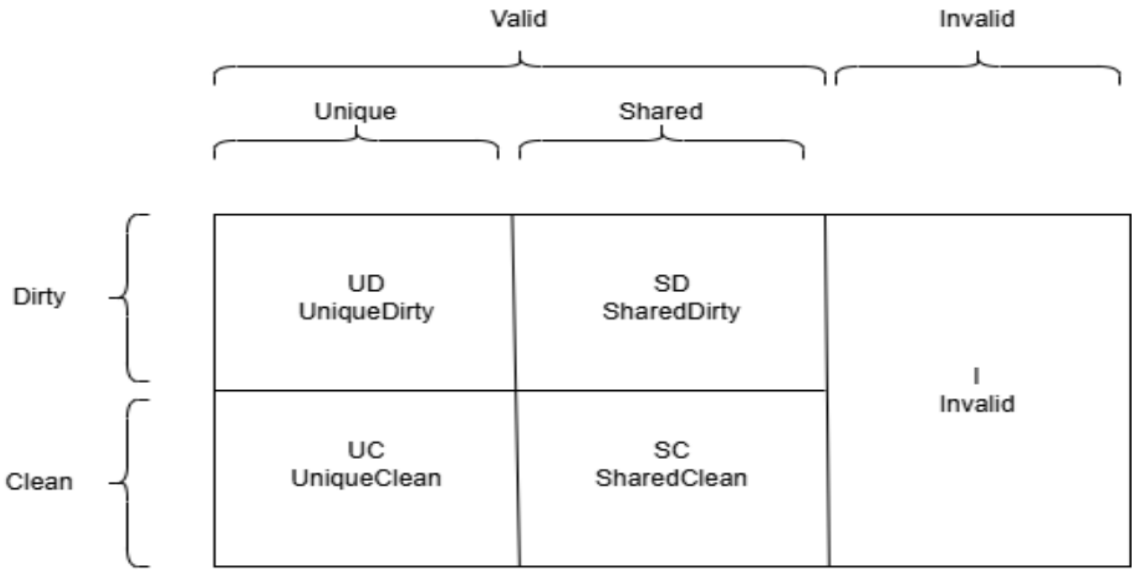
Canal	Semnal	Lățime	Sursă	Descriere
	AWLEN	[7:0]	Solicitant	Lungimea, numărul exact de transferuri de date într-o tranzacție de scriere.
	AWPROT	[2:0]	Solicitant	Atributele de protecție ale unei tranzacții de scriere.
	AWSIZE	[2:0]	Solicitant	Dimensiune, numărul de octeți din fiecare transfer de date.
	AWQOS	[3:0]	Solicitant	Identificatorul calității serviciului.
	AWREGION	[3:0]	Solicitant	Indicator de regiune.
	AWUSER	[3:0]	Solicitant	Extensie definită de utilizator.
	AWLOCK	[0:0]	Solicitant	Caracteristicile de detaliu (”atomice”) ale unei tranzacții de scriere.
	AWVALID	[0:0]	Solicitant	Adresă de scriere validă.
	AWREADY	[0:0]	Interfața	Adresă de scriere gata.
	AWSNOOP	[2:0]	Solicitant	Tip ”Snoop” pentru scrieri partajabile.
	AWDOMAIN	[1:0]	Solicitant	Domeniul de partajare.
	AWBAR	[1:0]	Solicitant	Indicator tip barieră pentru sincronizarea la scriere.
	AWUNIQUE	[0:0]	Solicitant	Permisiune unică pentru starea cache.
Canal adresare la citire	ARADDR	[31:0]	Solicitant	Adresa citire.
	ARID	[3:0]	Solicitant	ID-ul tranzacției de citire
	ARBURST	[1:0]	Solicitant	Tip de salvă.
	ARCACHE	[3:0]	Solicitant	Comportamentul cache.
	ARLEN	[7:0]	Solicitant	Numărul de transferuri de date.
	ARPROT	[2:0]	Solicitant	Atribute de protecție.
	ARSIZE	[2:0]	Solicitant	Dimensiunea transferului.
	ARQOS	[3:0]	Solicitant	Calitatea serviciilor.
	ARREGIUNE	[3:0]	Solicitant	Indicator de regiune.

Canal	Semnal	Lățime	Sursă	Descriere
	ARUSER	[3:0]	Solicitant	Extensie definită de utilizator.
	ARLOCK	[0:0]	Solicitant	Caracteristici atomice.
	ARVALID	[0:0]	Solicitant	Adresa validă la citire.
	ARREADY	[0:0]	Interfața	Adresa gata pentru citire.
	ARSNOOP	[2:0]	Solicitant	Tip ”Snoop” pentru citiri partajabile.
	ARDOMENIU	[1:0]	Solicitant	Domeniul de partajare.
	ARBAR	[1:0]	Solicitant	Indicator tip barieră pentru sincronizarea la citire.
Canal de date la citire	RID	[3:0]	Interfața	ID-ul datelor citite
	RDATA	[31:0]	Interfața	Citire date.
	RRESP	[1:0]	Interfața	Răspuns la citire.
	RLAST	[0:0]	Interfața	Ultimul transfer la citire.
	RUSER	[7:0]	Interfața	Extensie definită de utilizator.
	RVALID	[0:0]	Interfața	Date valide pentru citire.
	RREADY	[0:0]	Solicitant	Citire date gata.
Canal de scriere a datelor	WDATA	[31:0]	Solicitant	Scriere date.
	WLAST	[0:0]	Solicitant	Ultimul transfer la scriere.
	WSTRB	[3:0]	Solicitant	Impuls de sincronizare (”strobe”) la scriere.
	WUSER	[3:0]	Solicitant	Extensie definită de utilizator.
	WVALID	[0:0]	Solicitant	Date valide pentru scriere.
	WREADY	[0:0]	Interfața	Scriere date gata.
	WACK	[0:0]	Interfața	Confirmarea scrierii.
Canal de răspuns	OFFER	[3:0]	Interfața	ID-ul răspunsului.

Canal	Semnal	Lățime	Sursă	Descriere
	BRESP	[1:0]	Inter-fața	Răspuns la scriere.
	BUSER	[3:0]	Inter-fața	Extensie definită de utilizator.
	BVALID	[0:0]	Inter-fața	Răspuns valid.
	BREADY	[0:0]	Solici-tant	Răspuns gata.
Canalul de adrese ”Snoop”	ACADDR	[31:0]	Inter-fața	Adresa ”Snoop”.
	ACSNOOP	[3:0]	Inter-fața	Tip ”Snoop”.
	ACPROT	[2:0]	Inter-fața	Atribute de protecție.
	ACVALID	[0:0]	Inter-fața	Adresa ”Snoop” este validă.
	ACREADY	[0:0]	Solici-tant	Adresa ”Snoop” este gata.
Canalul de date ”Snoop”	CDDATA	[31:0]	Solici-tant	Date ”Snoop”.
	CDLAST	[0:0]	Solici-tant	Ultimul transfer de supraveghere (”Snoop”).
	CDVALID	[0:0]	Solici-tant	Datele ”Snoop” sunt valide.
	CDREADY	[0:0]	Inter-fața	Date ”Snoop” gata.
Canal de răspuns ”Snoop”	CRRESP	[4:0]	Solici-tant	Răspunsul ”Snoop”.
	CRVALID	[0:0]	Solici-tant	Răspunsul ”Snoop” este valid.
	CRREADY	[0:0]	Inter-fața	Răspunsul ”Snoop” este gata.

3. Stările coerente ale memoriei Cache

ACE acceptă următoarele stări coerente ale memoriei Cache: UniqueDirty (UD), UniqueClean (UC), SharedClean (SC), SharedDirty (SD) și Invalid (I):



3.1 UniqueDirty

Această stare indică faptul că linia (locația) cache este controlată exclusiv de un procesor și a fost modificată (alterată, afectată, compromisă, devenind ”murdară"). Datele din această linie cache vor fi diferite de cele stocate în memoria principală.

3.2 UniqueClean

Această stare indică faptul că linia cache este deținută exclusiv de un procesor și nu a fost modificată (e încă curată - "clean"). Datele din această linie cache sunt aceleași cu cele stocate în memoria principală.

3.3 SharedClean

Această stare indică faptul că linia cache este partajată între mai multe procesoare și nu a fost modificată de niciunul dintre acestea. Datele sunt "curate" deoarece sunt identice cu cele din memoria principală.

3.4 SharedDirty

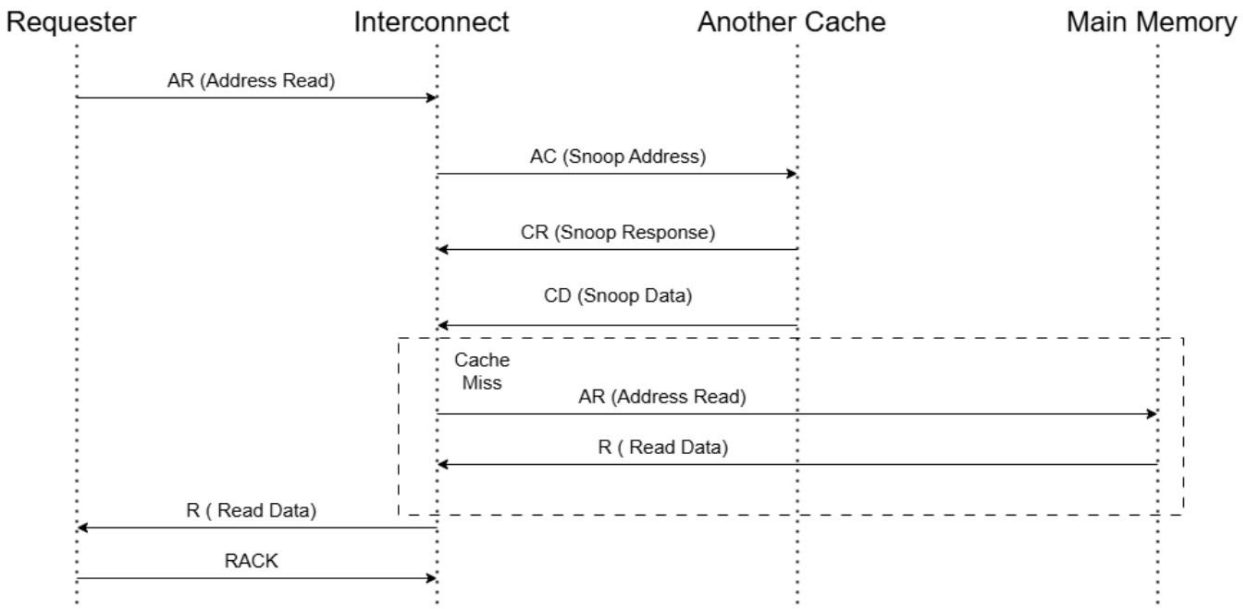
Această stare indică faptul că linia cache este partajată între mai multe procesoare și a fost modificată de cel puțin unul dintre ele. Datele sunt alterate ("murdare") fiind diferite de cele din memoria principală.

3.5 Invalid

Această stare indică faptul că linia cache nu este validă. Datele din această linie cache nu sunt considerate a fi o copie validă a datelor din memoria principală.

4. Tranzacțiile ACE

Tranzacții la citire



Succesiunea tranzacțiilor la citire

1. Tranzacția ReadNoSnoop

ReadNoSnoop este o tranzacție de citire care este utilizată într-o regiune de memorie ce nu poate fi partajată cu alți solicitanți. Tranzacția ReadNoSnoop este o operațiune de citire necoerentă în protocolul ACE. Este folosită atunci când solicitantul nu necesită interogarea – ”spionarea” (”Snooping”) a altor memorii cache. Așadar sunt utilizate numai canalul de citire a adresei și canalul de citire a datelor.

Cerințele tranzacției sunt:

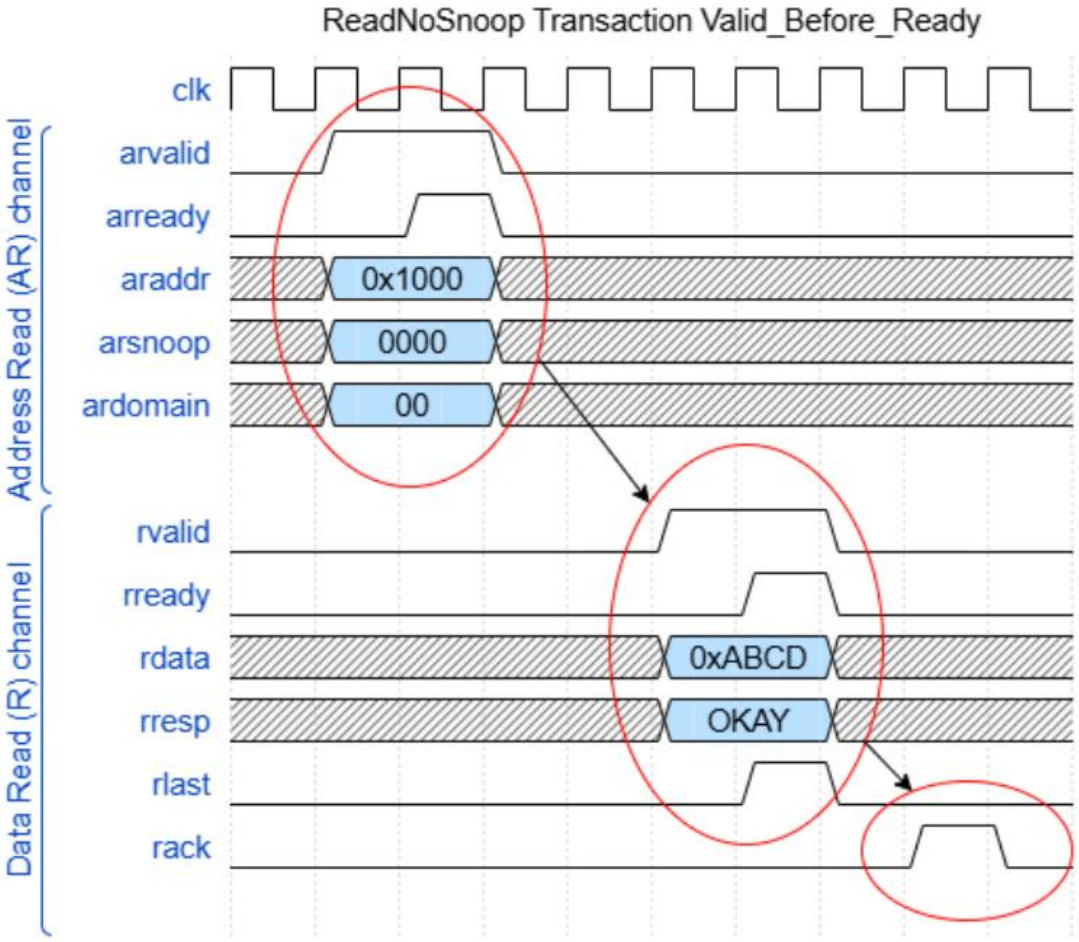
- răspunsul + IsShared trebuie să fie invalidat
- răspunsul + PassDirty trebuie să fie invalidat.

Modificări uzuale ale stării liniei cache pentru ReadNoSnoop

Stare Inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
Invalid(I)	Invalid(I)	00
Invalid(I)	UniqueClean(UC)	00

Alte modificări permise ale stării liniei cache pentru ReadNoSnoop

Stare Inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	Invalid(I)	00
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	Invalid(I)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedDirty(SD)	UniqueDirty(UD)	00



Succesiunea tranzacțiilor ReadNoSnoop

Succesiunea tranzacțiilor:

1. Solicitantul emite o cerere de citire pe canalul AR:

+ ARVALID este setat pentru a indica validarea informației de adresă.

+ ARADDR = 0x1000 specifică adresa vizată pentru citire.

+ ARSNOOP = 0x0 este codificat ca 0000, indicând o tranzacție ReadNoSnoop .

+ ARDOMAIN= 0x00 setează domeniul ca non-partajabil.

+ ARREADY este setat de furnizor pentru a arăta că acceptă (că e ”gata” pentru) transfer.

2. Interfața direcționează cererea direct către furnizor (de exemplu, controlerul de memorie) fără a iniția nicio tranzacție ”Snoop” pe canalul AC.

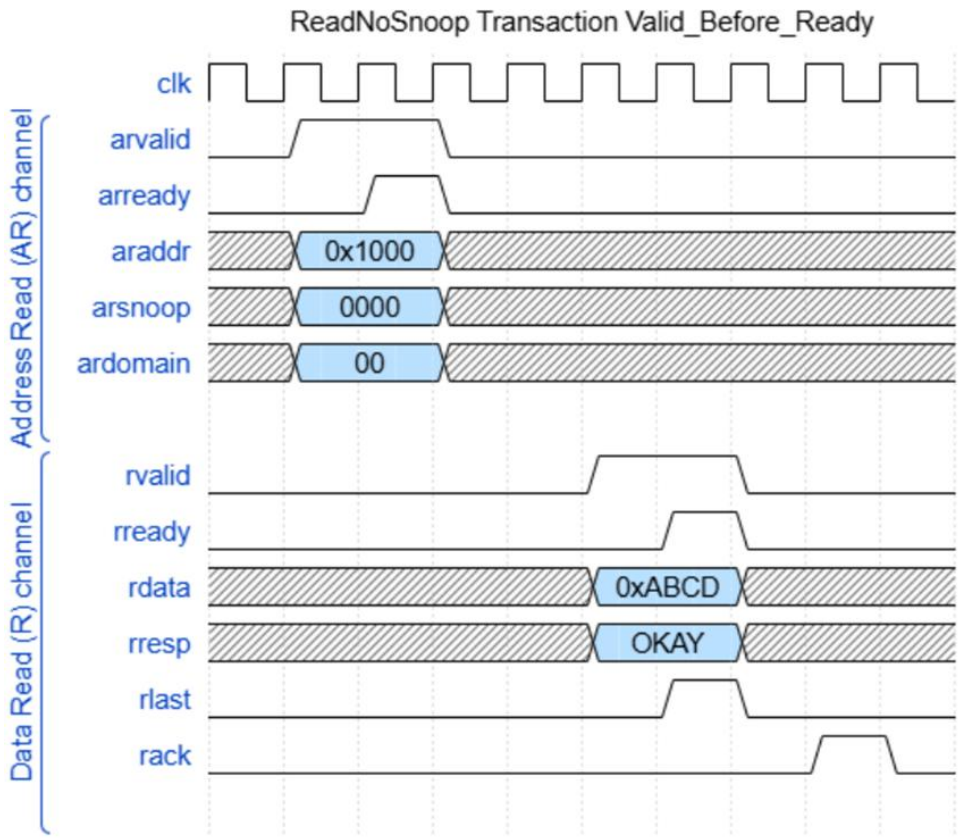
3. Furnizorul răspunde cu datele citite pe canalul R:

- RVALID este setat pentru a indica date valide pentru citire.
- RDATA = 0xABCD conține datele furnizate.
- RRESP[3:0] = 0000 indică un răspuns OKAY:
    - [1:0] = OKAY
    - [2] PassDirty bit = 0
    - [3] IsShared bit = 0
- RLAST este setat spre a indica ultimul transfer din salvă.
- RREADY este setat de solicitant indicând că e gata să accepte datele.
- Solicitantul setează RACK pentru a confirma citirea și a semnaliza finalizarea tranzacției.

Nu apare semnalizare legată de supraveghere (”spionaj” – ”Snoop”) pe tot parcursul tranzacției, deoarece mecanismele de coerență sunt ocolite (”bypassed”).

Exemple pentru alte tipuri de semnalizări ”handshake”:

validare anticipată (înainte de semnalizarea disponibilității):



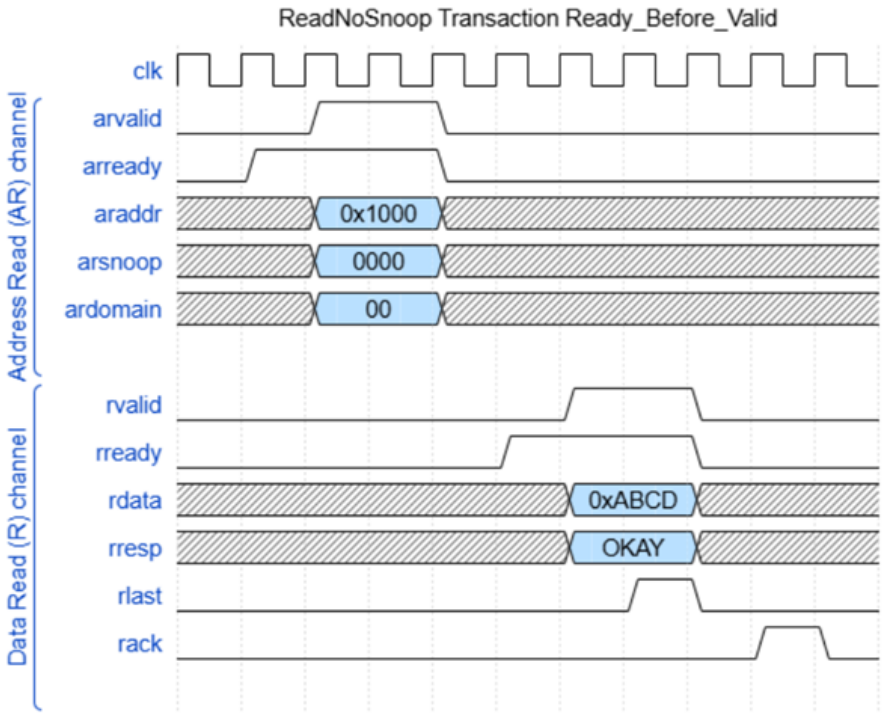

```
{
  signal: [
    {name: "clk",         wave: "p........"                 },

    ["Address Read (AR) channel",
    {name: "arvalid",     wave: "01.0......"                 },
    {name: "arready",     wave: "0.10......"                 },
    {name: "araddr",      wave: "x5.x......", data: "0x1000" },
    {name: "arsnoop",     wave: "x5.x......", data: "0000"   },
    {name: "ardomain",    wave: "x5.x......", data: "00"     },
    {                                                       },
    ],
    ["Data Read (R) channel",
    {name: "rvalid",      wave: "0....1.0.."                 },
    {name: "rready",      wave: "0.....10.."                 },
    {name: "rdata",       wave: "x....5.x..", data: "0xABCD" },
    {name: "rresp",       wave: "x....5.x..", data: "OKAY"   },
    {name: "rlast",       wave: "0.....10.."                 },
```

```
    {name: "rack",       wave: "0.......10"                 },
    {                                                       },
    ]

  ],
  head: { text: "ReadNoSnoop Transaction Valid_Before_Ready" },
  config : {hscale:1                                        }
}
```

Stilul Ready_Before_Valid (”Finalizare înainte de Validare”) – forme de undă ReadNoSnoop



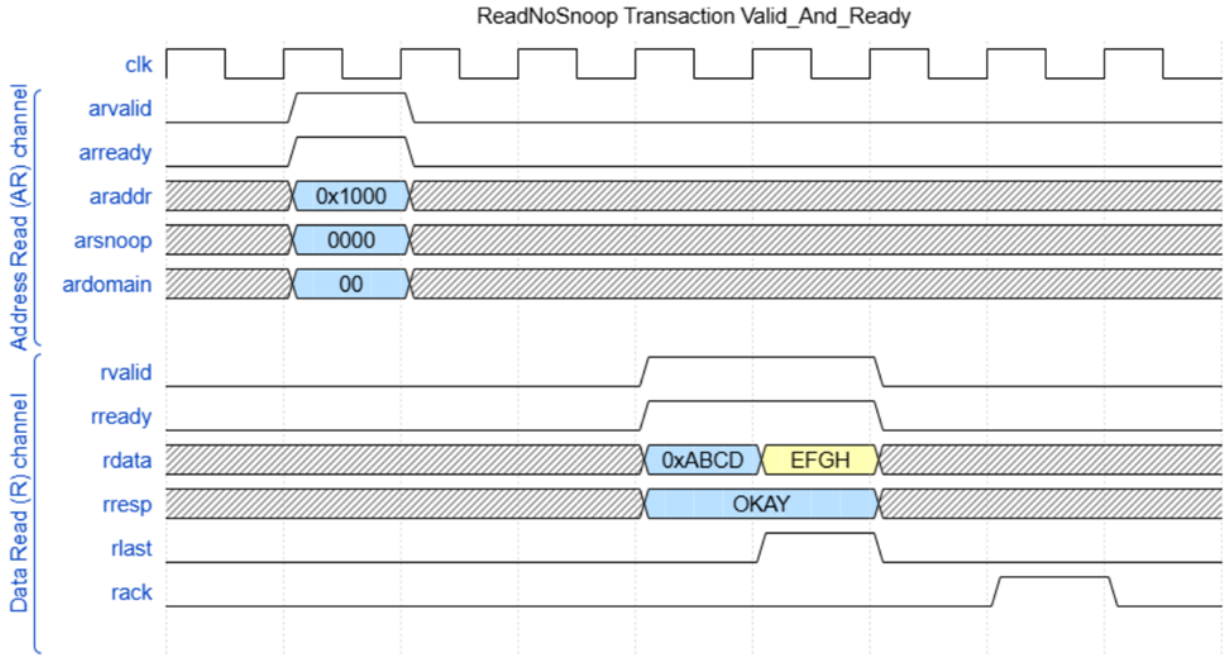

```
{
  signal: [
    {name: "clk"     , wave: "p........."                 },

    ["Address Read (AR) channel",
    {name: "arvalid" , wave: "0.1.0....."                 } ,
    {name: "arready" , wave: "01..0....."                 } ,
    {name: "araddr"  , wave: "x.5.x....."   , data: "0x1000"},
    {name: "arsnoop" , wave: "x.5.x....."   , data: "0000"  } ,
    {name: "ardomain", wave: "x.5.x....."   , data: "00"    } ,
    {                                                       } ,
    ],
    ["Data Read (R) channel",
    {name: "rvalid"  , wave: "0.....1.0.."                } ,
```

```
    {name: "rready"   , wave: "0....1..0.."                    } ,
    {name: "rdata"    , wave: "x.....5.x.."    , data: "0xABCD"},
    {name: "rresp"    , wave: "x.....5.x.."    , data: "OKAY"  } ,
    {name: "rlast"    , wave: "0......10.."                    } ,
    {name: "rack"     , wave: "0........10"                    } ,
    {                                                          } ,
    ]

  ],
  head: { text: "ReadNoSnoop Transaction Ready_Before_Valid" },
  config : {hscale:1                                           }
}
```

Stilul (”flavor”) Ready_And_Valid (”Finalizare și Validare”) – forme de undă ReadNoSnoop




```
{
  signal: [
    {name: "clk"      , wave: "p........"                                },

    ["Address Read (AR) channel",
    {name: "arvalid"  , wave: "010......"                                } ,
    {name: "arready"  , wave: "010......"                                } ,
    {name: "araddr"   , wave: "x5x......"    , data: "0x1000"            } ,
    {name: "arsnoop"  , wave: "x5x......"    , data: "0000"              } ,
    {name: "ardomain" , wave: "x5x......"    , data: "00"                } ,
    {                                                                    } ,
    ],
    ["Data Read (R) channel",
    {name: "rvalid"   , wave: "0...1.0.."                                } ,
    {name: "rready"   , wave: "0...1.0.."                                } ,
```

```
    {name: "rdata"    , wave: "x...53x.."    ,data: ["0xABCD", "EFGH"]} ,
    {name: "rresp"    , wave: "x...5.x.."    , data: "OKAY"            } ,
    {name: "rlast"    , wave: "0....10.."                              } ,
    {name: "rack"     , wave: "0......10"                              } ,
    {                                                                  } ,
    ]

  ],
  head: { text: "ReadNoSnoop Transaction Valid_And_Ready"            },
  config : {hscale:1.5                                               }
}
```

2. Tranzacția ReadOnce

Un solicitant utilizează tranzacția ReadOnce (”O singură citire”) pentru a citi date pe care nu trebuie să le copieze în memoria cache. Pentru tranzacția ReadOnce, dacă memoria cache care furnizează datele menține pentru linia cache starea Unique (”Unice”), nu e nevoie să se modifice starea liniei cache în Shared (”Partajat”) după tranzacția ReadOnce. Dacă solicitantul supravegheat (”spionat” – ”Snooped”) are o copie a liniei cache, atunci această specificație *recomandă* transferul datelor. Dacă solicitantul supravegheat (”spionat” – ”Snooped”) are linia cache în starea ”Dirty”, atunci datele *trebuie* transferate. Această specificație recomandă ca starea liniei cache să fie setată apoi ca ”Clean”. Deși este permisă trecerea stării liniei cache ca ”Dirty”, aceasta necesită ca interfața să scrie linia cache înapoi în memoria principală și starea liniei cache să fie trecută fie la”SharedClean”, fie la valoarea ”Invalid”.

Modificări uzuale ale stării ReadOnce a liniei cache

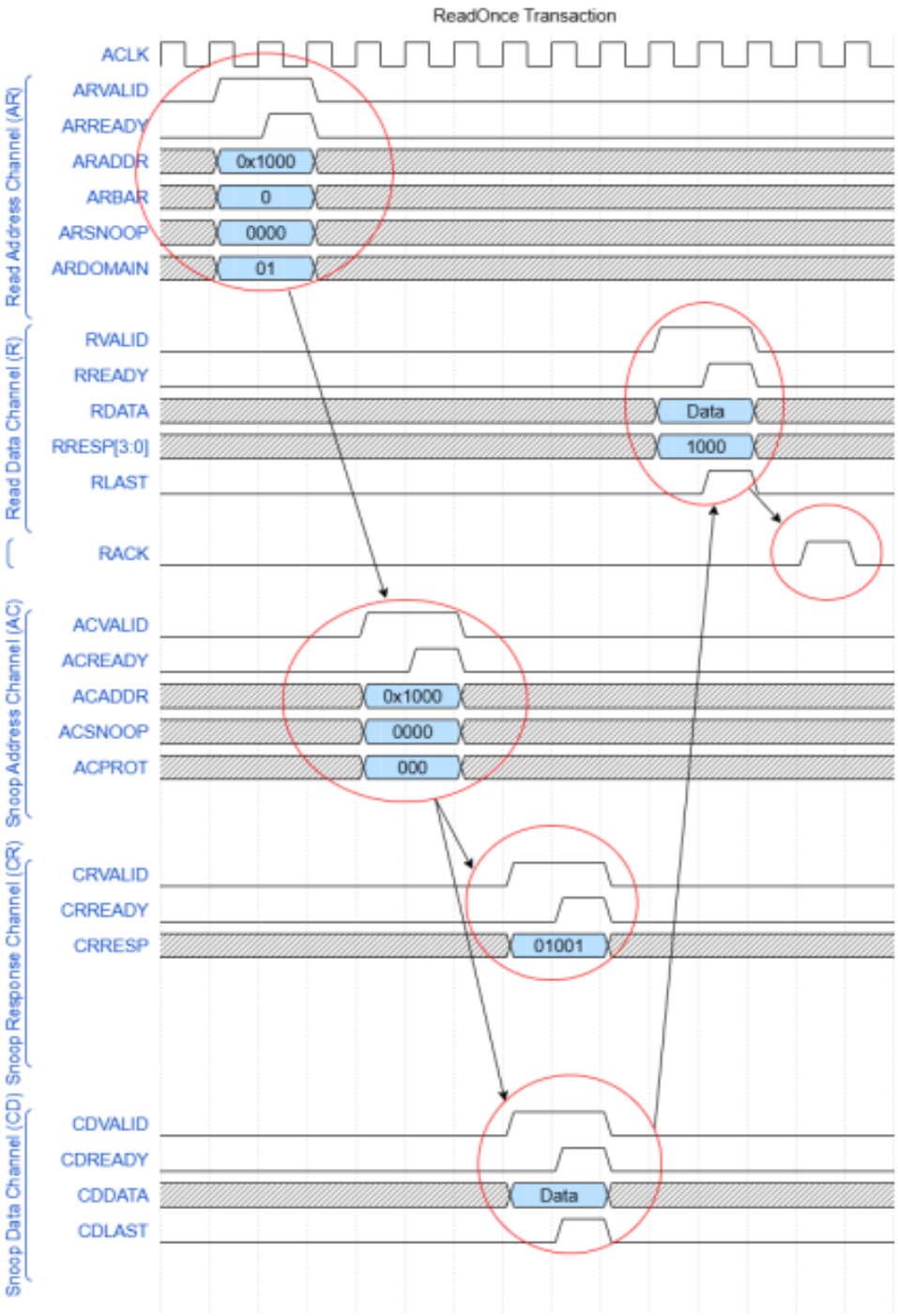
Starea inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
Invalid(I)	Invalid(I)	00
Invalid(I)	Invalid(I)	10

Alte schimbări permise ale liniei cache la ReadOnce

Starea inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	SharedClean(SC)	10
SharedDirty(SD)	UniqueDirty(UD)	00
SharedDirty(SD)	SharedDirty(SD)	10

- Modificările permise ale stării liniei cache și răspunsurile PassDirty și IsShared asociate supravegherii (”Snoop”) pentru tranzacția ”Snoop” ReadOnce:

Starea inițială a Liniei Cache	**Starea finală a Liniei Cache**	**CRRESP[2] (IsShared)**	**CRRESP[3] (PassDirty)**	**Descriere**
Invalid (I)	Invalid (I)	0	0	Linia rămâne nevalidă, deoarece la ReadOnce datele nu se memorează în cache
Unique Clean (UC)	Invalid (I)	0	0	Linia poate fi invalidată dacă memoria cache alege să o șteargă
Unique Clean (UC)	Unique Clean (UC)	0	1	Linia poate rămâne unică dacă solicitantul nu memorează datele în cache
Unique Clean (UC)	Shared Clean (SC)	0	1	Linia devine partajată pe măsură ce un alt solicitant o citește
Unique Dirty (UD)	Invalid (I)	1	0	Linia poate fi invalidată după transmiterea datelor ”murdare”
Unique Dirty (UD)	Unique Dirty (UD)	0	1	Linia poate rămâne ”murdară” dar unică dacă solicitantul nu o memorează în cache
Unique Dirty (UD)	Shared Clean (SC)	1	1	Linia poate deveni partajată și ”curată” după transmiterea datelor ”murdare”
Unique Dirty (UD)	Shared Dirty (SD)	0	1	Linia poate deveni partajată păstrându-și starea de ”murdară”
Shared Clean (SC)	Invalid (I)	0	0	Linia poate fi invalidată dacă memoria cache alege să șteargă datele
Shared Clean (SC)	Shared Clean (SC)	0	1	Linia rămâne în starea partajată și ”curată”
Shared Dirty (SD)	Invalid (I)	1	0	Linia poate fi invalidată după ce se transmit date ”murdare”
Shared Dirty (SD)	Shared Clean (SC)	1	1	Linia poate deveni ”curată” după ce se transmit date ”murdare”
Shared Dirty (SD)	Shared Dirty (SD)	0	1	Linia rămâne în stare partajată și ”murdară”



Tranzacția ReadOnce

Succesiunea tranzacțiilor:

1. Solicitantul emite o cerere de citire pe canalul AR:
    + `ARVALID` este setat pentru a indica validarea informațiilor despre adresă.
    + `ARADDR` = 0x1000 specifică adresa vizată pentru citire.
    + `ARSNOOP` = 0000 indică o tranzacție ReadOnce.

+ ARDOMAIN= 01 sau 10 setează domeniul ca partajabil intern sau extern (”Inner Shareable” respectiv ”Outer Shareable”).

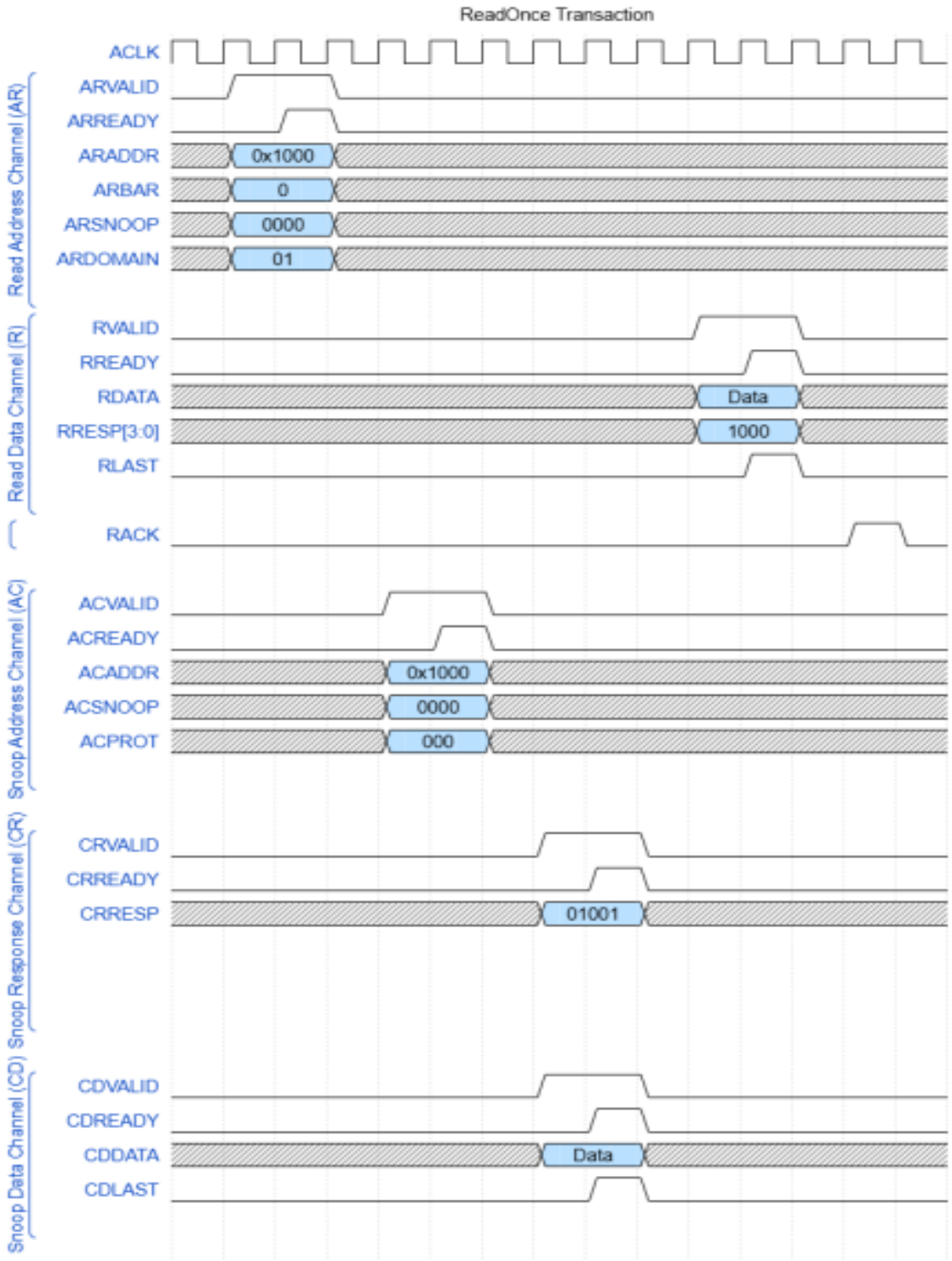
+ ARBAR = 0 setează atributele de protecție (acces la date securizat, fără privilegii).

+ ARREADY este setat de interfață pentru a da acceptul în vederea transferului.

2. Interfața direcționează cererea către controller-ele de cache implicate, prin canalul AC:
   - ACADDR = 0x1000 corespunde adresării originale a cererii.
   - ACSNOOP = 0000 indică o tranzacție ”Snoop” tip ReadOnce.
   - ACPROT = 000 setează atributele de protecție.
   - ACVALID și ACREADY trebuie setate înainte de începutul oricărui răspuns sau al oricărui transfer de date.
3. Controller-ele cache răspund pe canalul CR:
   - CRVALID este setat doar după ce sunt setate atât ACVALID cât și ACREADY.
   - CRRESP = 01001 indică:
     - WasUnique = 0
     - IsShared = 1
     - PassDirty = 0
     - Error = 0
     - DataTransfer = 1
4. Datele de coerență sunt transferate pe canalul CD:
   - CDVALID este setat după ce sunt setate atât ACVALID cât și ACREADY.
   - Solicitantul poate seta CDVALID fără a fi nevoie să aștepte CRREADY sau CDREADY.
5. Furnizorul răspunde cu datele citite, pe canalul R:
   - RVALID este setat spre a indica faptul că datele citite sunt valide.
   - RDATA conține datele citite.
   - RRESP[3:0] = 1000 indică:
     - OKAY = 00
     - IsShared = 1
     - PassDirty = 0
   - RLAST este setat spre a indica ultimul transfer din salvă.
   - RREADY este setat de solicitant indicând că e gata să accepte datele.
6. Solicitantul setează RACK pentru a confirma citirea și a semnaliza finalizarea.

Tranzacția se finalizează fără a scrie datele în memoria cache a solicitantului.

Stilul Valid_Before_Ready (”Validare înainte de finalizare”) – forme de undă ReadOnce




```
{
  "signal": [
    {name: 'ACLK', wave: 'p.............'                    },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '01.0..........'              },
```

```
      {name: 'ARREADY' , wave: '0.10.........'                    },
      {name: 'ARADDR'  , wave: 'x5.x.........', data: ['0x1000'] },
      {name: 'ARBAR'   , wave: 'x5.x.........',  data: ['0']      },
      {name: 'ARSNOOP' , wave: 'x5.x.........', data: ['0000']   },
      {name: 'ARDOMAIN', wave: 'x5.x.........', data: ['01']     },
      {                                                          }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0........1.0..'                 },
      {name: 'RREADY'    , wave: '0.........10..'                 },
      {name: 'RDATA'     , wave: 'x........5.x..', data: ['Data'] },
      {name: 'RRESP[3:0]', wave: 'x........5.x..', data: ['1000'] }, // IsSh
ared(1) + PassDirty(0) + OKAY (00) - SharedClean
      {name: 'RLAST'     , wave: '0.........10..'                 },
      {                                                          }
    ],

    [
      {name: 'RACK', wave: '0...........10'                      },

    ],
    {                                                            },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0...1.0.......'                   },
      {name: 'ACREADY', wave: '0....10.......'                   },
      {name: 'ACADDR' , wave: 'x...5.x.......', data: ['0x1000']  },
      {name: 'ACSNOOP', wave: 'x...5.x.......', data: ['0000']   },
      {name: 'ACPROT' , wave: 'x...5.x.......', data: ['000']    },
      {                                                          }
    ],
      {                                                          },

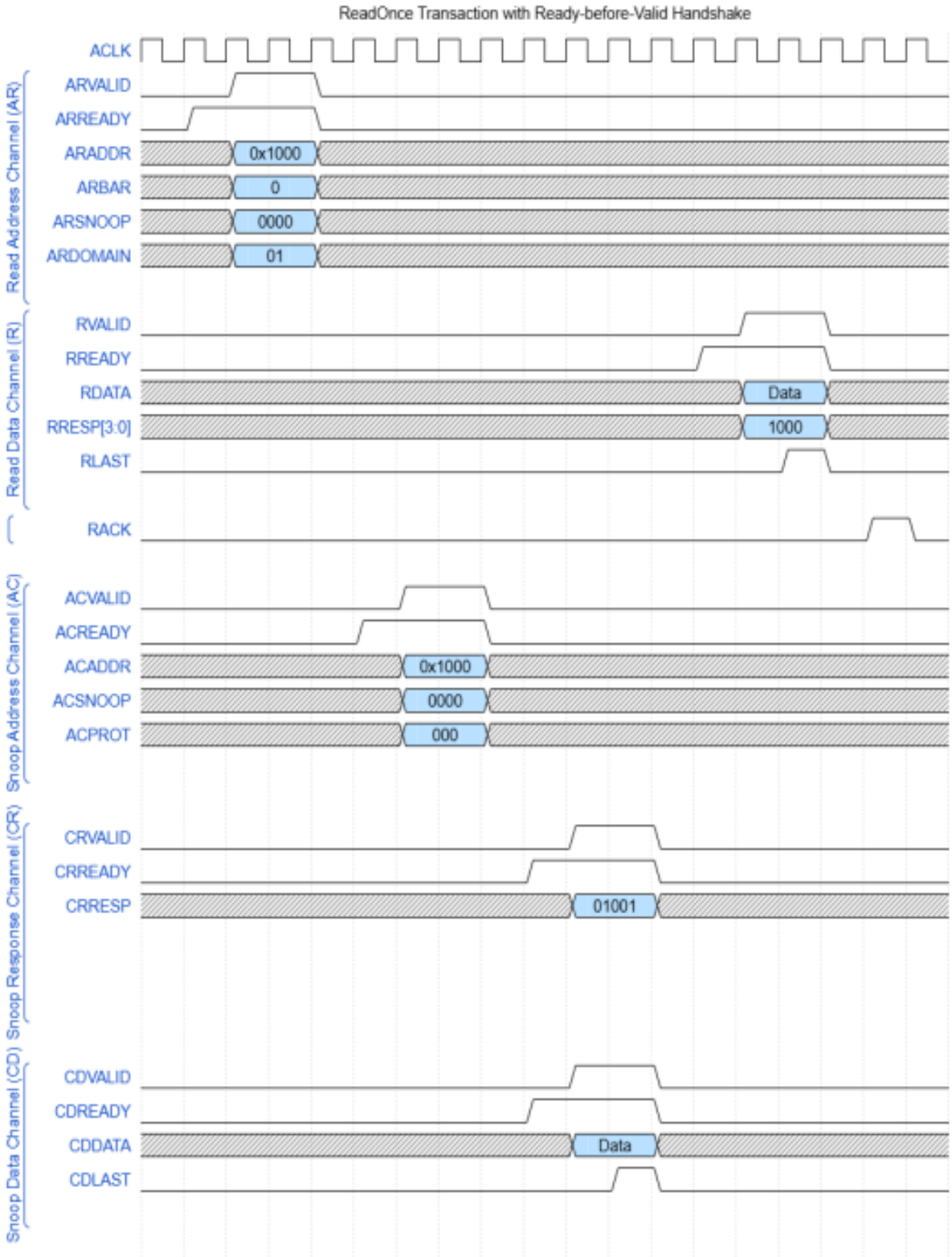
    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0......1.0.....'                  },
      {name: 'CRREADY', wave: '0.......10.....'                  },
      {name: 'CRRESP' , wave: 'x......5.x.....', data: ['01001']   }, // WasU
nique(0) + IsShared(1) + PassDirty(0) + Error(0) + DataTransfer(1)
      {                                                          },
      {                                                          },
      {                                                          }
    ],
      {                                                          },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0......1.0.....'                  },
      {name: 'CDREADY', wave: '0.......10.....'                  },
      {name: 'CDDATA' , wave: 'x......5.x.....', data: ['Data']    },
      {name: 'CDLAST' , wave: '0.......10.....'                  },
      {                                                          }
    ],
      {                                                          },
  ],

  "head": {  "text": "ReadOnce Transaction "                     }

}
```

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – forme de undă ReadOnce



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.................
'                                },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0.1.0.............
'                           },
      {name: 'ARREADY' , wave: '01..0.............
'                           },
      {name: 'ARADDR'  , wave: 'x.5.x.............', data: ['0x1000
']        },
      {name: 'ARBAR'   , wave: 'x.5.x.............', data: ['0
']           },
      {name: 'ARSNOOP' , wave: 'x.5.x.............', data: ['0000
']         },
      {name: 'ARDOMAIN', wave: 'x.5.x.............', data: ['01
']           },

{                                                                    }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0............1.0..
'                          },
      {name: 'RREADY'    , wave: '0............1..0..
'                          },
      {name: 'RDATA'     , wave: 'x............5.x..', data: ['Data
']        },
      {name: 'RRESP[3:0]', wave: 'x............5.x..', data: ['1000
']        }, // IsShared(1) + PassDirty(0) + OKAY (00) - SharedClean
      {name: 'RLAST'     , wave: '0.............10..
'                          },

{                                                                    }
    ],

    [
      {name: 'RACK', wave: '0...............10
'                              },

    ],

{                                                                      },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.....1.0.........
'                            },
      {name: 'ACREADY', wave: '0....1..0.........
```

```
'                           },
      {name: 'ACADDR' , wave: 'x.....5.x.........', data: ['0x1000
']         },
      {name: 'ACSNOOP', wave: 'x.....5.x.........', data: ['0000
']          },
      {name: 'ACPROT' , wave: 'x.....5.x.........', data: ['000
']           },

{                                                                   }
    ],

{                                                                   },

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0.........1.0......
'                           },
      {name: 'CRREADY', wave: '0........1..0......
'                           },
      {name: 'CRRESP' , wave: 'x.........5.x......', data: ['01001
']          }, // WasUnique(0) + IsShared(1) + PassDirty(0) + Error(0) + Data
Transfer(1)

{                                                                   },

{                                                                   },

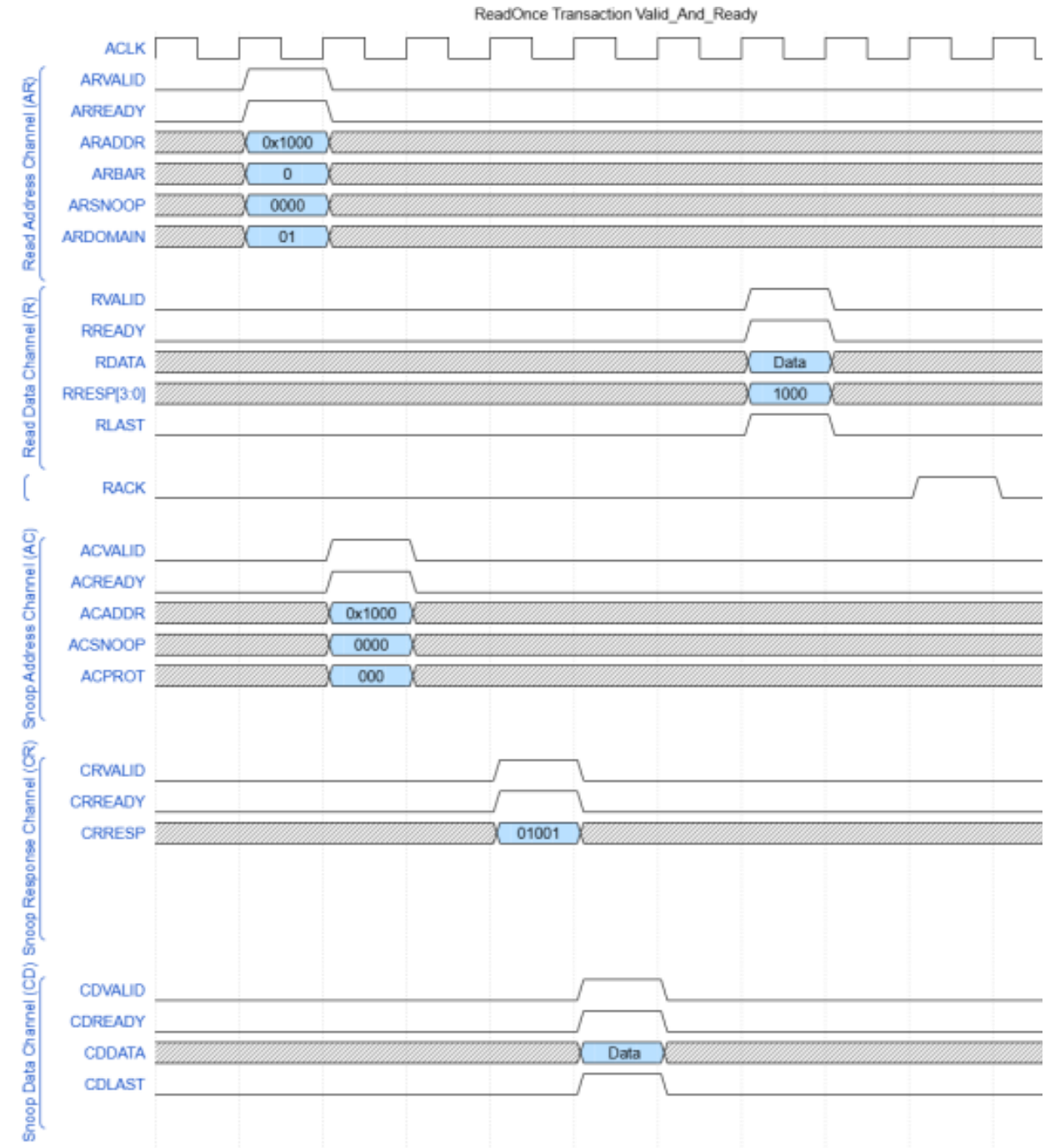
{                                                                   }
    ],

{                                                                   },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0.........1.0......
'                           },
      {name: 'CDREADY', wave: '0........1..0......
'                           },
      {name: 'CDDATA' , wave: 'x.........5.x......', data: ['Data
']          },
      {name: 'CDLAST' , wave: '0.........10......
'                           },
    {                                                                   }
    ],

{                                                                   },
  ],
  "head": {  "text": "ReadOnce Transaction with Ready-before-Valid Handshake
" }
}
```

Stilul Ready_And_Valid (”Finalizare și validare”) – forme de undă ReadOnce




```
{
  "signal": [
    {name: 'ACLK', wave: 'p.........'                          },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '010........'                    },
      {name: 'ARREADY' , wave: '010........'                    },
      {name: 'ARADDR'  , wave: 'x5x........',  data: ['0x1000'] },
```

```
      {name: 'ARBAR'   , wave: 'x5x........',  data: ['0']      },
      {name: 'ARSNOOP' , wave: 'x5x........',  data: ['0000']   },
      {name: 'ARDOMAIN', wave: 'x5x........',  data: ['01']     },
      {                                                        }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0......10..'                  },
      {name: 'RREADY'    , wave: '0......10..'                  },
      {name: 'RDATA'     , wave: 'x......5x..', data: ['Data'] },
      {name: 'RRESP[3:0]', wave: 'x......5x..', data: ['1000'] }, // IsShared
(1) + PassDirty(0) + OKAY (00) - SharedClean
      {name: 'RLAST'     , wave: '0......10..'                  },
      {                                                        }
    ],

    [
      {name: 'RACK', wave: '0........10'                       },

    ],
    {                                                          },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.10.......'                    },
      {name: 'ACREADY', wave: '0.10.......'                    },
      {name: 'ACADDR' , wave: 'x.5x.......', data: ['0x1000']  },
      {name: 'ACSNOOP', wave: 'x.5x.......', data: ['0000']    },
      {name: 'ACPROT' , wave: 'x.5x.......', data: ['000']     },
      {                                                        }
    ],
      {                                                          },

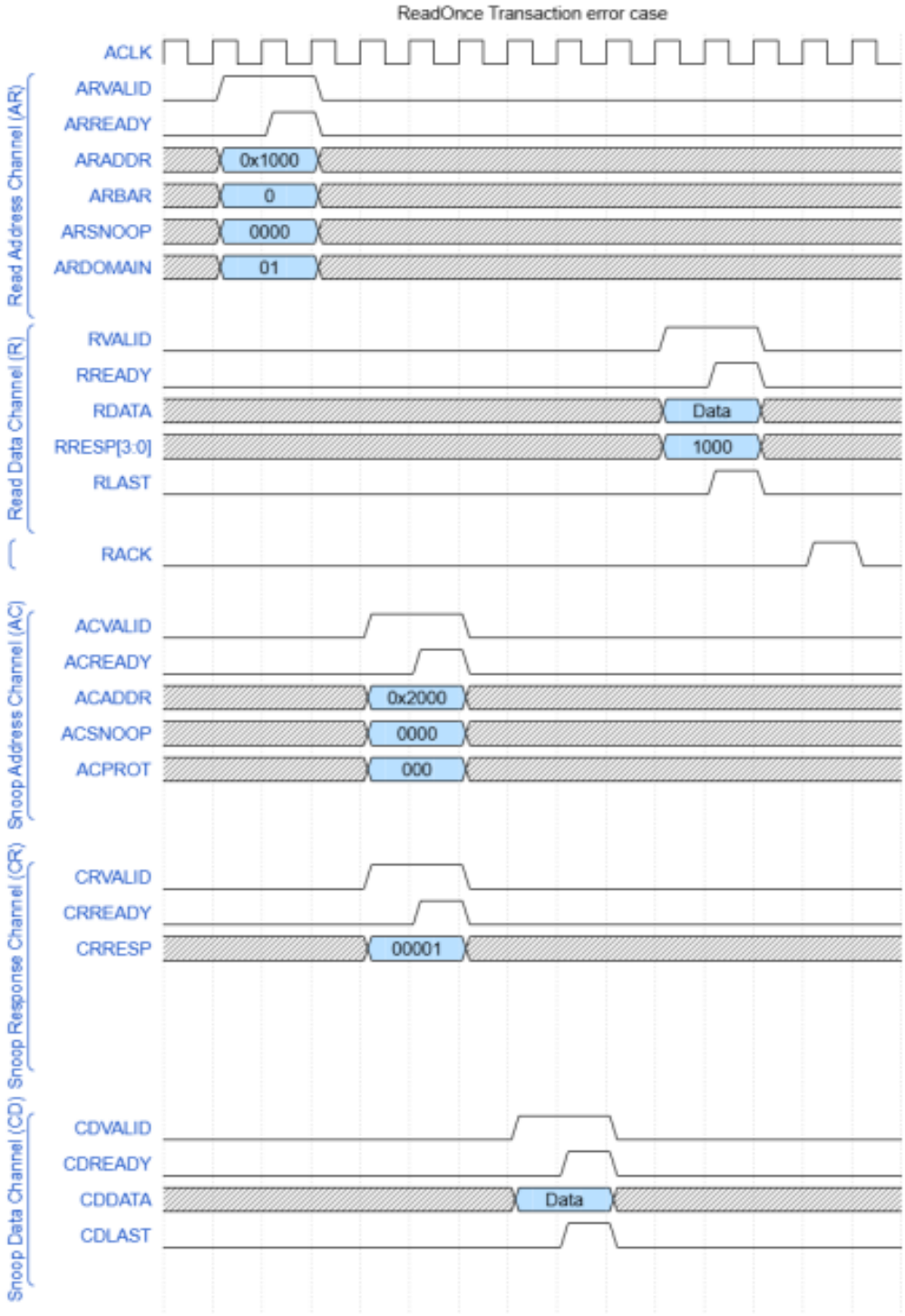
    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...10.....'                    },
      {name: 'CRREADY', wave: '0...10.....'                    },
      {name: 'CRRESP' , wave: 'x...5x.....', data: ['01001']   }, // WasUniqu
e(0) + IsShared(1) + PassDirty(0) + Error(0) + DataTransfer(1)
      {                                                          },
      {                                                          },
      {                                                        }
    ],
      {                                                          },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0....10....'                    },
      {name: 'CDREADY', wave: '0....10....'                    },
      {name: 'CDDATA' , wave: 'x....5x....', data: [' Data']   },
      {name: 'CDLAST' , wave: '0....10....'                    },
      {                                                        }
    ],
      {                                                          },
  ],

  "head": {  "text": "ReadOnce Transaction Valid_And_Ready"    },
  config : {hscale: 1.5 }

 }
```

- Solicitantul trebuie să aștepte ca atât ACVALID cât și ACREADY să fie setate înainte de a seta CRVALID. În exemplul de *eroare* de mai jos, solicitantul setează CRVALID înainte să se încheie ”strângerea de mână” (”handshake”) ACVALID/ACREADY.



Tranzacția ReadOnce eronată

```
{

  "signal": [
    {name: 'ACLK', wave: 'p.............'                        },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '01.0..........'                   },
      {name: 'ARREADY' , wave: '0.10..........'                   },
      {name: 'ARADDR'  , wave: 'x5.x..........', data: ['0x1000'] },
      {name: 'ARBAR'   , wave: 'x5.x..........', data: ['0']      },
      {name: 'ARSNOOP' , wave: 'x5.x..........', data: ['0000']   },
      {name: 'ARDOMAIN', wave: 'x5.x..........', data: ['01']     },
      {                                                           }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0........1.0..'                  },
      {name: 'RREADY'    , wave: '0.........10..'                  },
      {name: 'RDATA'     , wave: 'x.........5.x..', data: ['Data'] },
      {name: 'RRESP[3:0]', wave: 'x.........5.x..', data: ['1000'] }, // IsSh
ared(1) + PassDirty(0) + OKAY (00) - SharedClean
      {name: 'RLAST'     , wave: '0.........10..'                  },
      {                                                           }
    ],

    [
      {name: 'RACK', wave: '0...........10'                       },

    ],
    {                                                             },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0...1.0........'                    },
      {name: 'ACREADY', wave: '0....10........'                    },
      {name: 'ACADDR' , wave: 'x...5.x........', data: ['0x2000']  },
      {name: 'ACSNOOP', wave: 'x...5.x........', data: ['0000']    },
      {name: 'ACPROT' , wave: 'x...5.x........', data: ['000']     },
      {                                                           }
    ],
      {                                                           },

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...1.0........'                    },
      {name: 'CRREADY', wave: '0....10........'                    },
      {name: 'CRRESP' , wave: 'x...5.x........', data: ['00001']   }, // WasU
nique(1) + IsShared(0) + PassDirty(0) + Error(0) + DataTransfer(1)
      {                                                           },
      {                                                           },
      {                                                           }
```

```
    ],
      {                                                                 },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0......1.0.....'                         },
      {name: 'CDREADY', wave: '0.......10.....'                         },
      {name: 'CDDATA' , wave: 'x......5.x.....', data: ['Data']         },
      {name: 'CDLAST' , wave: '0.......10.....'                         },
      {                                                                 }
    ],
      {                                                                 },
  ],


  "head": {  "text": "ReadOnce Transaction error case"                  }

}
```

3. Tranzacția ReadClean

- ReadClean este o tranzacție de citire care este utilizată într-o regiune de memorie care poate fi partajată cu alți solicitanți. Este garantat că o tranzacție ReadClean nu va transfera responsabilitatea pentru actualizarea memoriei principale către solicitantul (inițiator al tranzacției). De obicei, o tranzacție ReadClean este utilizată de un solicitant care dorește să obțină o copie curată a unei linii cache, de exemplu un solicitant având un cache cu scriere directă (*și* în memoria principală).
- Pentru o tranzacție ReadClean de supraveghere (”Snoop”), dacă responsabilitatea pentru scrierea liniei cache înapoi în memoria principală este transferată interfeței, așa cum indică setarea răspunsului PassDirty ”Snoop”, respectiva linie cache este scrisă înapoi în memoria principală imediat. Această specificație recomandă ca linia cache să fie marcată ”Dirty” (”murdară”) în cache-ul supravegheat (”snooped”).

Cerințele privind răspunsul la tranzacție sunt:

• Răspunsul IsShared indică dacă și alte cache conțin această linie sau nu.

• Răspunsul PassDirty trebuie să fie dezactivat (resetat).

Modificări uzuale ale stării liniei cache pentru ReadClean

Stare Inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
Invalid(I)	UniqueDirty(UD)	00
Invalid(I)	SharedClean(SC)	10

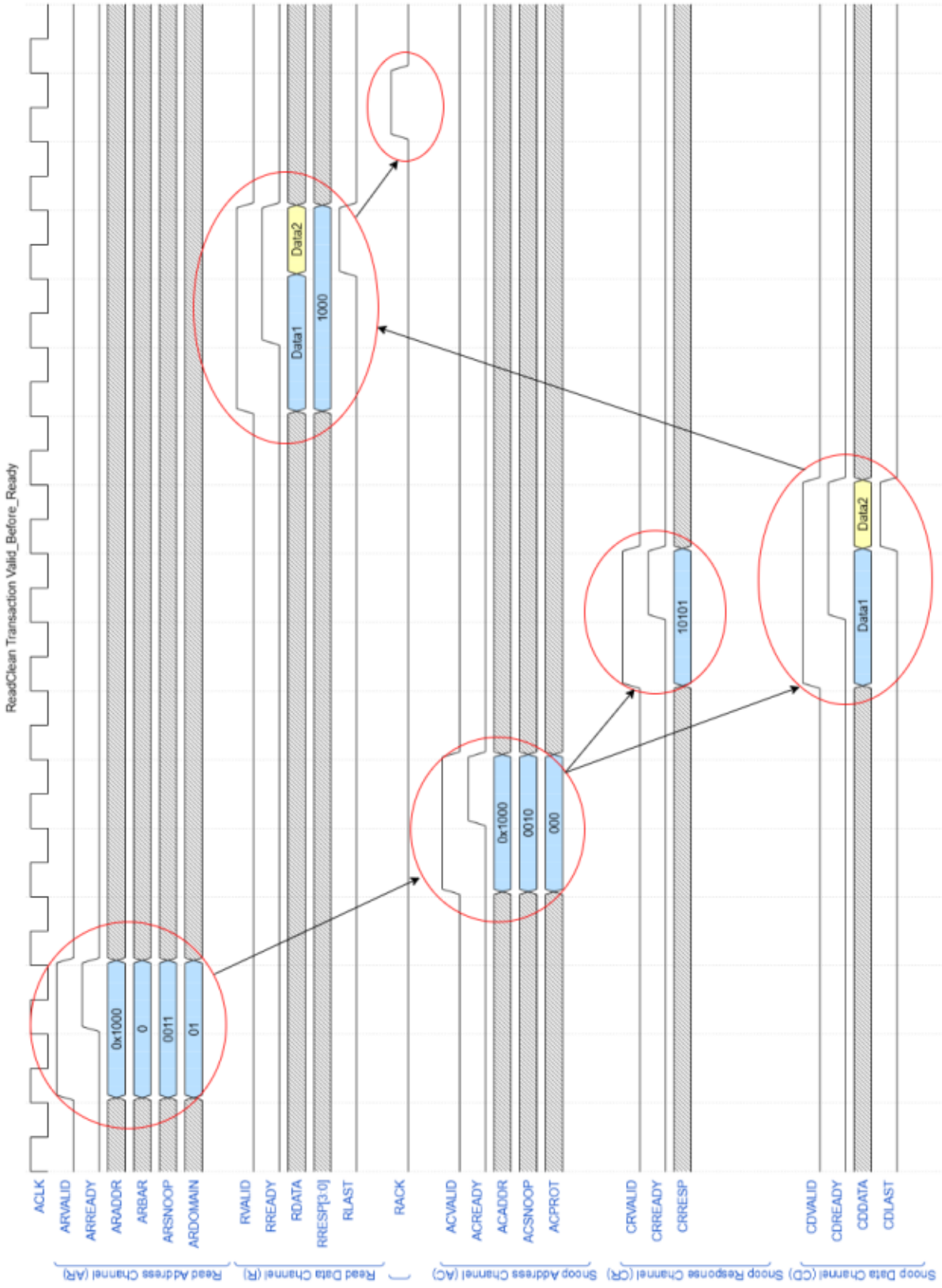
Alte modificări permise ale stării liniei cache pentru ReadClean

Stare Inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	SharedClean(SC)	10
SharedDirty(SD)	UniqueDirty(UD)	00
SharedDirty(SD)	SharedDirty(SD)	10

- Toate modificările permise ale stării liniei cache și răspunsurile ”Snoop” PassDirty și IsShared asociate la tranzacțiile ”Snoop” ReadClean, ReadShared și ReadNotSharedDirty.

Stare Inițială Cache	**Stare Finală Cache**	**CRRESP[2]**	**CRRESP[3]**	**Descriere**
Invalid	Invalid (I)	0	0	Linia cache rămâne invalidă
Unique Clean	Invalid (I)	0	0	Linia cache este ștearsă în ciuda faptului că este singura copie ”curată”
Unique Clean	Shared Clean (SC)	0	1	Linia cache rămâne ”curată”, dar este acum partajată cu alte cache
Shared Clean	Invalid (I)	0	0	Linia cache este ștearsă în timp ce există alte copii ”curate”
Shared Clean	Shared Clean (SC)	0	1	Linia cache partajată își menține starea ”curată”
Unique Dirty	Invalid (I)	1	0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Unique Dirty	Shared Clean (SC)	1	1	Datele ”murdare” sunt transferate și linia cache devine partajată ”curată”
Unique Dirty	Shared Dirty (SD)	0	1	Datele ”murdare” sunt transferate și linia cache devine partajată, dar rămâne ”murdară”
Shared Dirty	Invalid (I)	1	0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Shared Dirty	Shared Clean (SC)	1	1	Datele ”murdare” sunt transferate și linia cache devine partajată ”curată”
Shared Dirty	Shared Dirty SD	0	1	Datele ”murdare” sunt transferate și linia cache devine partajată, dar rămâne ”murdară”

Tranzacția ReadClean:



Desfășurarea tranzacției:

1. Faza Address Read (canalul AR)

- Solicitantul setează ARVALID pentru a indica o cerere validă de citire

- Canalul AR transmite următoarele informații:

    - ARADDR : 0x1000 (adresa zonei memoriei vizate pentru citire)

    - ARBAR : 0 (Indicator tip barieră pentru sincronizarea la citire)

    - ARSNOOP : 0011 (codul unei tranzacții ReadClean)

    - ARDOMAIN: 01 (indică un domeniul ca partajabil extern -”Outer Shareable”)

2. Faza de adresare coerentă (canalul AC)

- Inițiatorul / interfața redirecționează cererile de coerență către alte sub-sisteme cache

- Canalul AC semnalizează:
    - ACVALID: Setat spre a indica o tranzacție validă de coerență
    - ACADDR: 0x1000 (corespunde adresării originale a cererii)
    - ACSNOOP: 0010 (indică tipul operației de supraveghere ”Snoop”)
    - ACPROT: 000 (setează atributele de protecție)

3. Faza de răspuns pentru coerență (canalul CR)

- Cache supravegheate (”Snooped”) răspund cu informațiile de coerență
    - CRVALID: Setat spre a indica un răspuns valid de coerență
    - CRRESP: 01001(cod al răspunsului de coerență, indicând starea cache)
    - Canalul CR depinde de canalul AC - răspunsurile de coerență pot fi generate numai după primirea unei cereri de coerență valide pe canalul AC

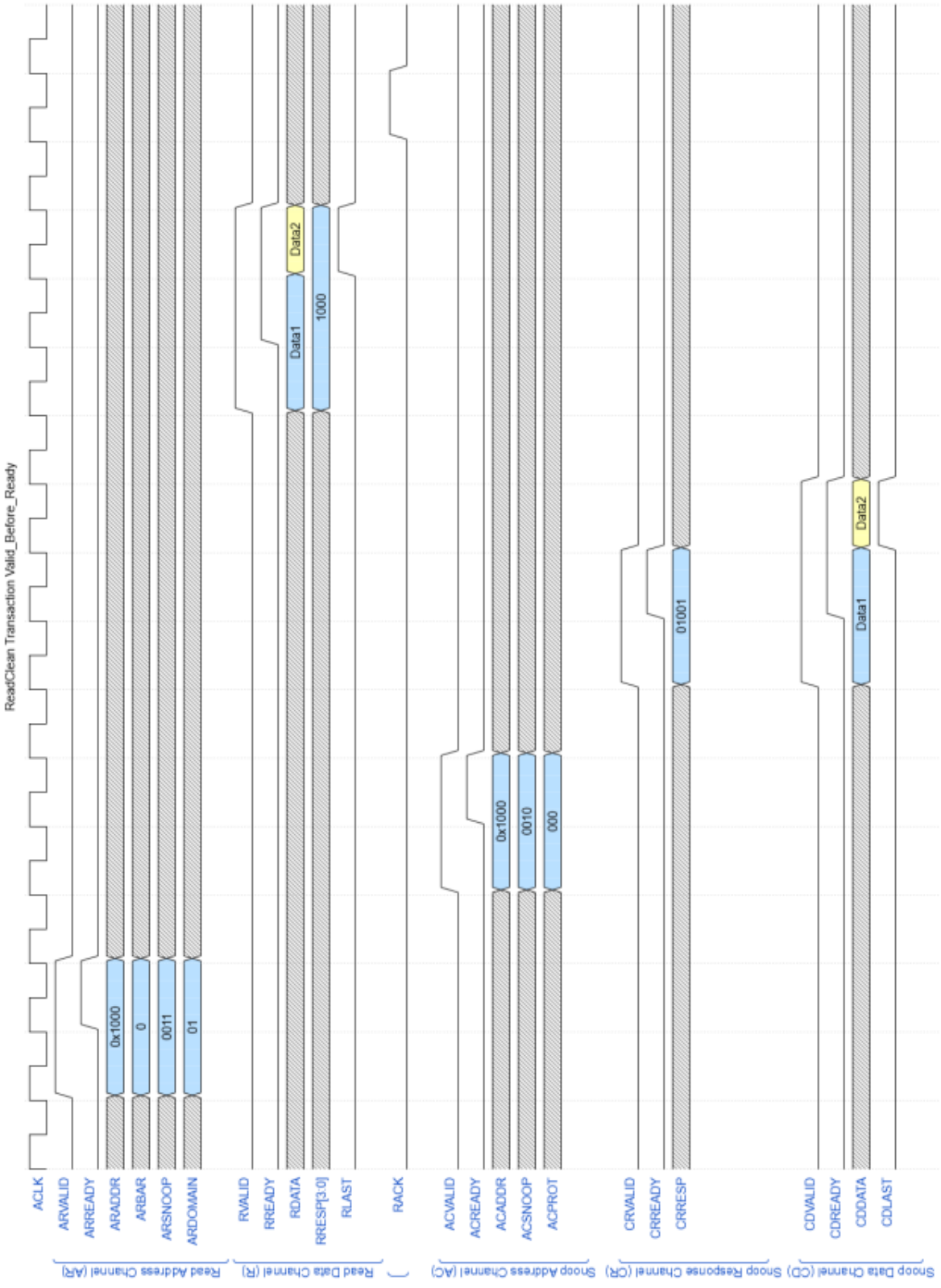
- Răspunsul de coerență nu trebuie să aștepte informațiile de coerență; aceste canale funcționează independent, odată ce este recepționată cererea AC

4. Faza coerenței datelor (canalul CD)
   - Dacă un cache are o copie validă a datelor solicitate, o furnizează
     - `CDVALID`: Setat spre a indica validitatea datelor în condiții de coerență
     - `CDDATA` : Conține Date1 și Date2 (datele solicitate de la alt cache)
     - CDLAST : Indică ultimul transfer de date al salvei
     - Canalul CD depinde numai de canalul AC – transferul datelor solicitate poate începe numai după primirea unei cereri de coerență valide pe canalul AC
     - Canalul CD funcționează independent de canalul CR; niciunul nu trebuie să aștepte terminarea semnalizării pe celălalt canal
5. Faza de citire a datelor (canalul R)
   - Memoria sau controlerul de cache returnează datele cerute solicitantului inițial
     - `RVALID`: Setat spre a indica validitatea datelor citite
     - `RDATA`: Conține Date1 și Date2 (datele cerute)
     - `RRESP`: 1000 (cod al răspunsului)
     - `RLAST`: Indică ultimul transfer de date al salvei

6. Completarea tranzacției
   - `RACK`: Semnal de confirmare care finalizează tranzacția

Stilul Valid_Before_Ready (”Validare înainte de finalizare”) –

forme de undă ReadClean



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                    },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '01.0.............
'                                  },
      {name: 'ARREADY' , wave: '0.10.............
'                                  },
      {name: 'ARADDR'  , wave: 'x5.x.............', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x5.x.............', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x5.x.............', data: ['0011
']            },  // ReadClean
      {name: 'ARDOMAIN', wave: 'x5.x.............', data: ['01
']              },

{                                                                                }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.........1..0..
'                             },
      {name: 'RREADY'    , wave: '0..........1.0..
'                             },
      {name: 'RDATA'     , wave: 'x.........5.3x..', data: ['Data1', 'Data2
'] },
      {name: 'RRESP[3:0]', wave: 'x.........5..x..', data: ['1000
']           },  // IsShared(1) + PassDirty(0) + OKAY(00) - SharedClean
      {name: 'RLAST', wave: '0............10..
'                                  },

{                                                                                }
    ],

    [
      {name: 'RACK', wave: '0.............10
'                                   },
    ],

{                                                                                  },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0...1.0.........
'                                  },
      {name: 'ACREADY', wave: '0....10.........
'                                  },
      {name: 'ACADDR',  wave: 'x...5.x.........', data: ['0x1000
']            },
      {name: 'ACSNOOP', wave: 'x...5.x.........', data: ['0010
```

```
'] },  // Corrected to 0010 for ReadClean snoop
      {name: 'ACPROT', wave: 'x...5.x.........', data: ['000
'] },

{                                                                    }
    ],

{                                                                      },

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0......1.0.......
'                              },
      {name: 'CRREADY', wave: '0.......10.......
'                              },
      {name: 'CRRESP' , wave: 'x......5.x.......', data: ['01001
'] },  // WasUnique(0) + IsShared(1) + PassDirty(0) + Error(0) +
DataTransfer(1)

{                                                                    },

{                                                                    },

{                                                                    }
    ],

{                                                                      },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0......1..0......
'                              },
      {name: 'CDREADY', wave: '0.......1.0......
'                              },
      {name: 'CDDATA' , wave: 'x......5.3x......', data: [' Data1', 'Data2
']  },
      {name: 'CDLAST' , wave: '0........10......
'                              },

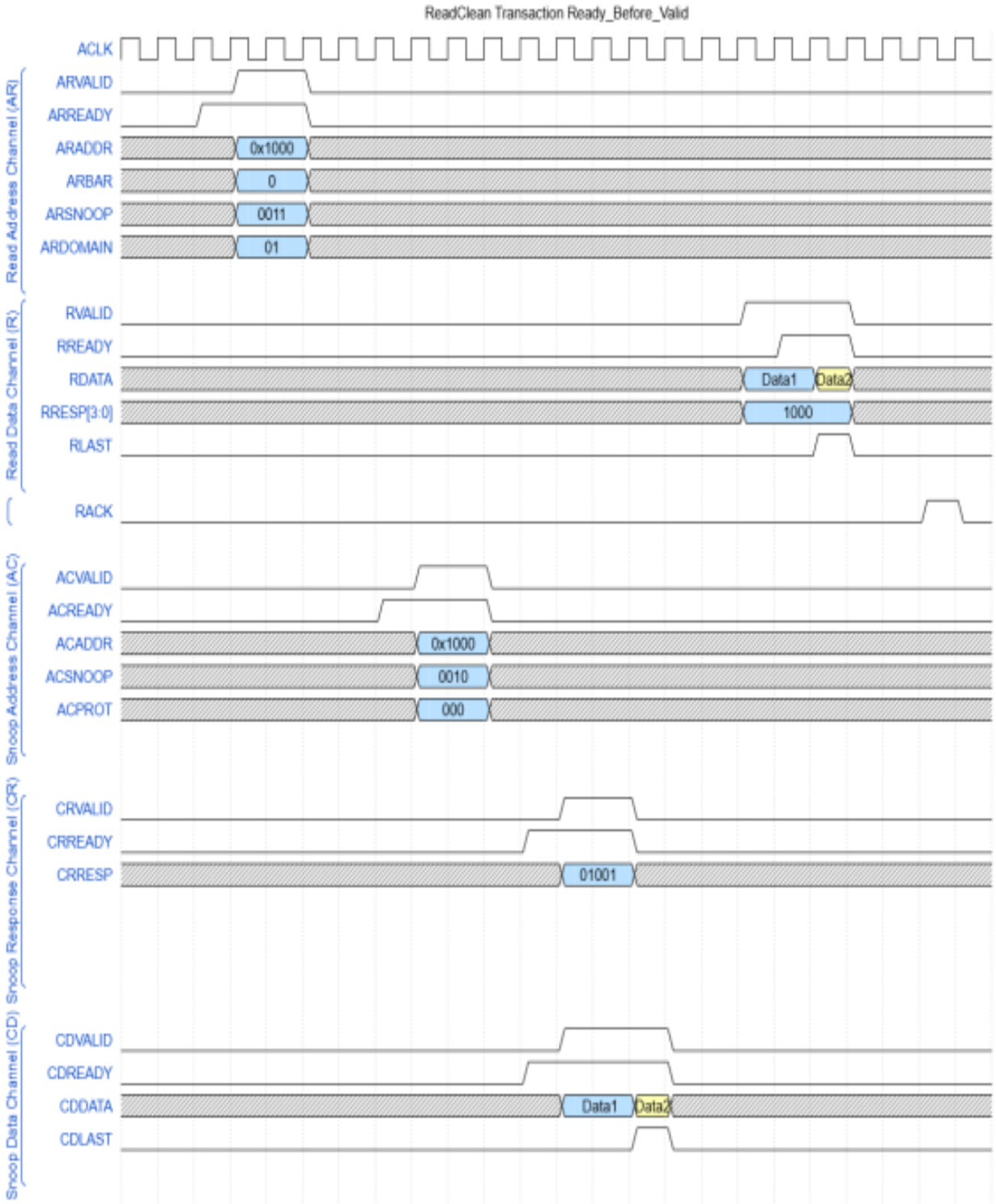
{                                                                    }
    ],

{                                                                      },
  ],

  "head": {"text": "ReadClean Transaction Valid_Before_Ready
"                 },
  config: {hscale:
1                                                  }
}
```

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) –

forme de undă ReadClean



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                                 },
      {name: 'ARREADY' , wave: '0.1..0.................
'                                 },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0011
']             },  // ReadClean
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']               },

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                                  },
      {name: 'RREADY'    , wave: '0.............1.0...
'                                  },
      {name: 'RDATA'     , wave: 'x.............5.3x...', data: ['Data1',
'Data2'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['1000
']           },  // IsShared(1) + PassDirty(0) + OKAY(00) - SharedClean
      {name: 'RLAST', wave: '0.................10...
'                                 },

{
}
    ],

    [
      {name: 'RACK', wave: '0...................10
'                                   },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
'                                  },
      {name: 'ACREADY', wave: '0......1..0.............
'                                  },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x1000
']           },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0010
']             },  // Corrected to 0010 for ReadClean snoop
      {name: 'ACPROT', wave: 'x.......5.x.............', data: ['000
```

```
']               },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0..........1.0.........
'                              },
      {name: 'CRREADY', wave: '0..........1..0.........
'                              },
      {name: 'CRRESP' , wave: 'x..........5.x.........', data: ['01001
']             },  // WasUnique(0) + IsShared(1) + PassDirty(0) + Error(0) +
DataTransfer(1)

{
},

{
},

{
}
    ],

{
},

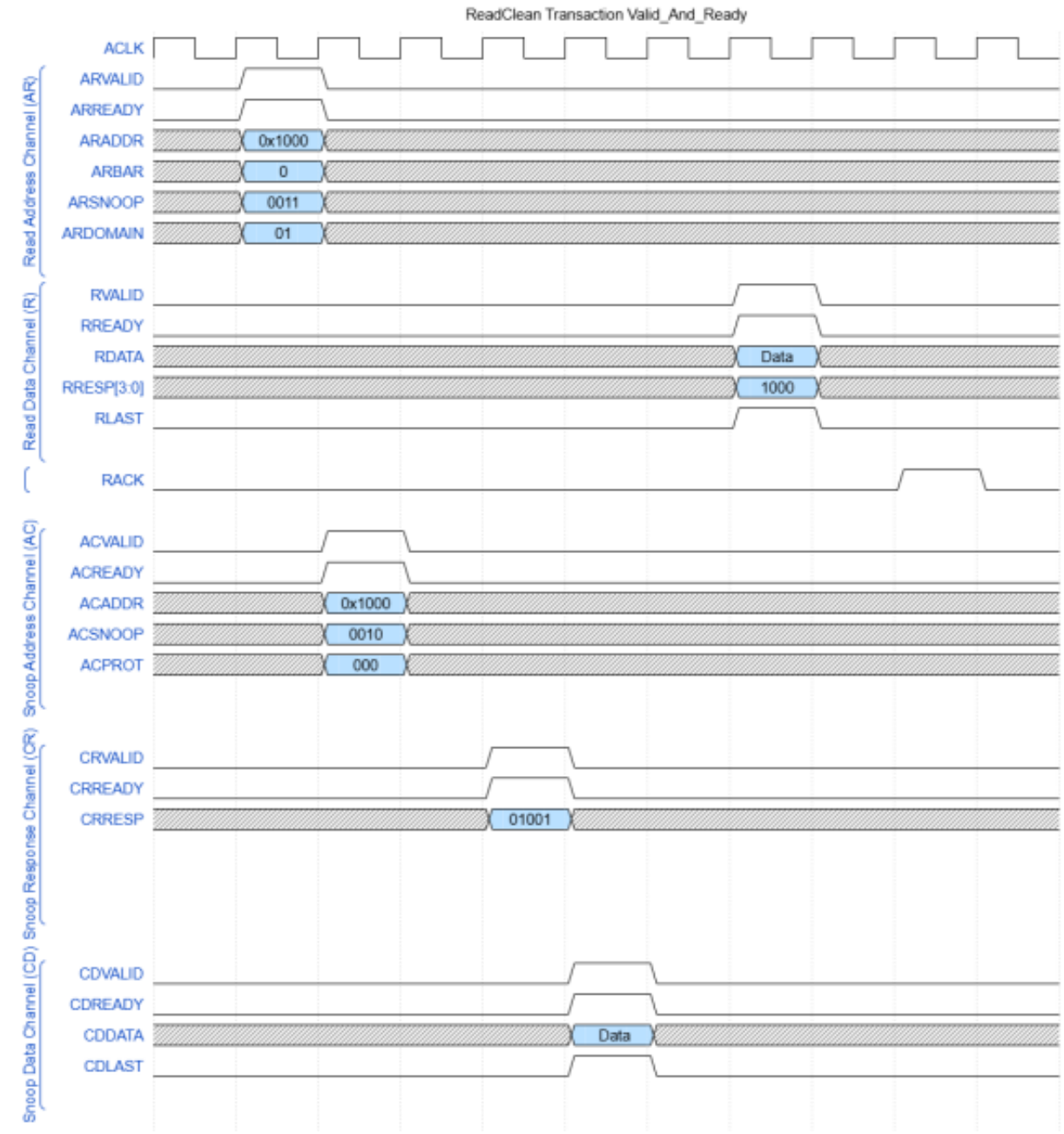
    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0..........1..0........
'                              },
      {name: 'CDREADY', wave: '0..........1...0........
'                              },
      {name: 'CDDATA' , wave: 'x..........5.3x........', data: [' Data1', 'D
ata2']   },
      {name: 'CDLAST' , wave: '0............10........
'                              },

{
}
    ],

{
},
  ],

  "head": {"text": "ReadClean Transaction Ready_Before_Valid
"                        },
  config: {hscale:
1                                                                  }
}
```

Stilul Ready_And_Valid (”Finalizare și validare”) – forme de undă ReadClean




```
{
  "signal": [
    {name: 'ACLK', wave: 'p.........'                          },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '010........'                   },
      {name: 'ARREADY' , wave: '010........'                   },
      {name: 'ARADDR'  , wave: 'x5x........', data: ['0x1000'] },
      {name: 'ARBAR'   , wave: 'x5x........', data: ['0']      },
```

```
      {name: 'ARSNOOP' , wave: 'x5x........', data: ['0011']   },  // ReadCle
an
      {name: 'ARDOMAIN', wave: 'x5x........', data: ['01']     },
      {                                                        }
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0......10..'                 },
      {name: 'RREADY'    , wave: '0......10..'                 },
      {name: 'RDATA'     , wave: 'x......5x..', data: ['Data'] },
      {name: 'RRESP[3:0]', wave: 'x......5x..', data: ['1000'] },  // IsShare
d(1) + PassDirty(0) + OKAY(00) - SharedClean
      {name: 'RLAST'     , wave: '0......10..'                 },
      {                                                        }
    ],

    [
      {name: 'RACK', wave: '0........10'                       },
    ],
    {                                                          },

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.10.......'                    },
      {name: 'ACREADY', wave: '0.10.......'                    },
      {name: 'ACADDR', wave: 'x.5x.......', data: ['0x1000']   },
      {name: 'ACSNOOP', wave: 'x.5x.......', data: ['0010']    },  // Correct
ed to 0010 for ReadClean snoop
      {name: 'ACPROT', wave: 'x.5x.......', data: ['000']      },
      {                                                        }
    ],
    {                                                          },

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...10.....'                    },
      {name: 'CRREADY', wave: '0...10.....'                    },
      {name: 'CRRESP', wave: 'x...5x.....', data: ['01001']    },  // WasUniq
ue(0) + IsShared(1) + PassDirty(0) + Error(0) + DataTransfer(1)
      {                                                        },
      {                                                        },
      {                                                        }
    ],
    {                                                          },

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0....10....'                    },
      {name: 'CDREADY', wave: '0....10....'                    },
      {name: 'CDDATA', wave: 'x....5x....', data: [' Data']    },
      {name: 'CDLAST', wave: '0....10....'                     },
      {                                                        }
    ],
    {                                                          },
  ],

  "head": {"text": "ReadClean Transaction Valid_And_Ready"     },
  config: {hscale: 1.5                                         }
}
```

Funcționarea ReadClean: `ARSNOOP = 0011` ; domeniu partajabil extern: `ARDOMAIN = 01` ; supraveghere (”Snoop”) tip ReadShared: `ACSNOOP = 0010` ; acces la date securizat, privilegiat: `ACPROT = 000` ; răspuns pentru coerență și furnizare date: `CRRESP = 01001` ; răspuns ce indică succesul citirii: `RRESP = 1000`.

În alte cazuri, dacă linia cache este contaminată (”murdară”), responsabilitatea de a scrie înapoi în memoria principală este transferată interfeței.

4. ReadNotSharedDirty

ReadNotSharedDirty este o tranzacție de citire care este utilizată într-o regiune de memorie care poate fi partajată cu alți solicitanți. Tranzacția ReadNotSharedDirty indică faptul că solicitantul care cere citirea poate accepta o linie cache care se află în orice stare, cu excepția stării SharedDirty. Aceasta înseamnă că linia cache poate fi trecută ca și ”curată” (fie unică, fie partajată) sau că linia cache poate fi transmisă ca unică și ”murdară”. Pentru ca această tranzacție să se finalizeze cu succes, poate primi diverse combinații de răspunsuri IsShared și PassDirty de la alți agenți din sistem. IsShared indică faptul că datele sunt partajate între mai multe procesoare, iar PassDirty indică faptul că datele sunt ”murdare” și e necesar să fie actualizate.

Combinațiile valide pentru finalizarea unei tranzacții ReadNotSharedDirty sunt următoarele:

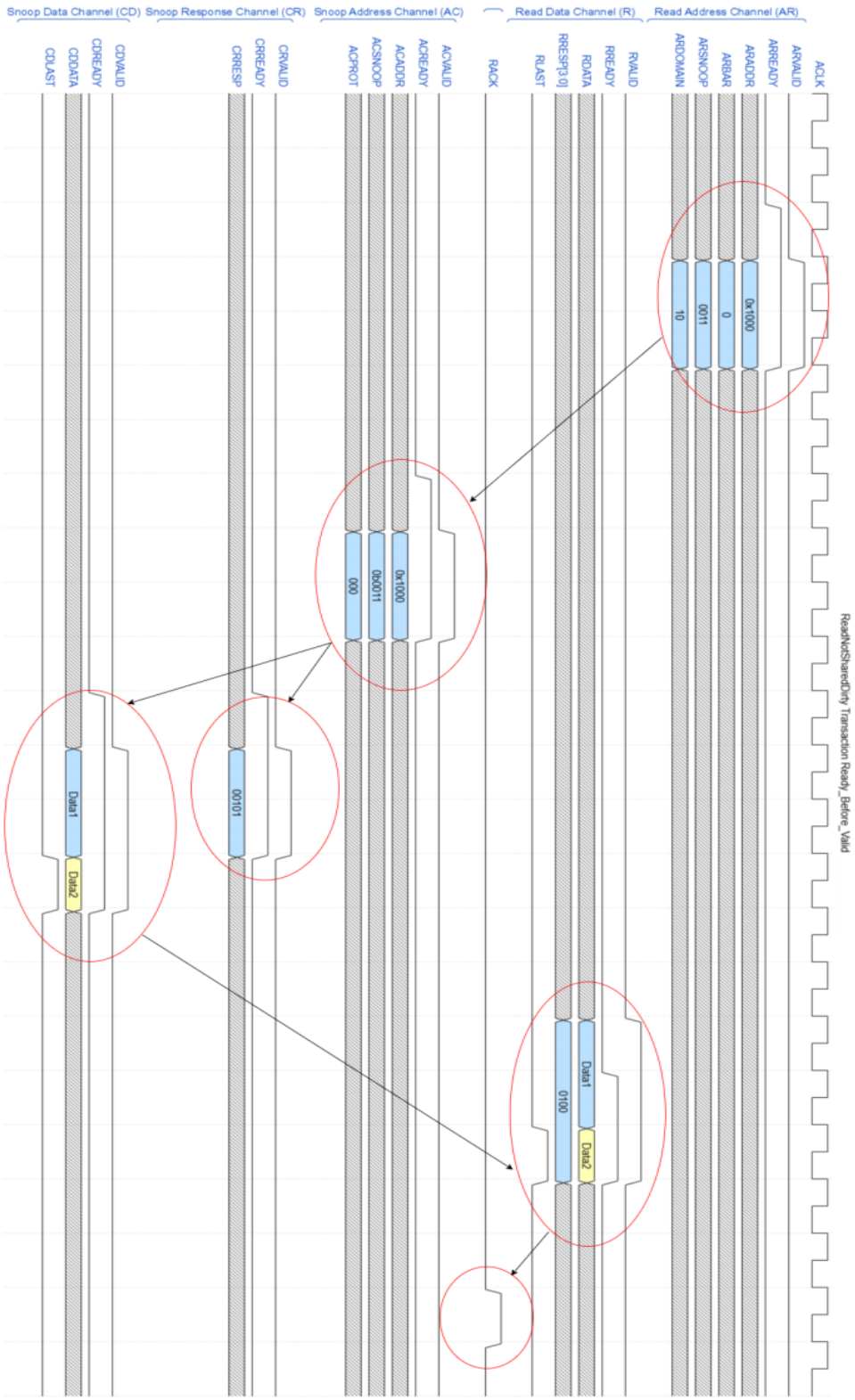
- IsShared nu este setat, PassDirty nu este setat ;
- IsShared este setat, PassDirty nu este setat ;
- IsShared nu este setat, PassDirty este setat.

Privitor la răspunsul la tranzacție, cerințele sunt:

- Răspunsul IsShared indică faptul că linia cache este partajată sau unică.
- Răspunsul PassDirty indică faptul că linia cache este ”curată” sau ”murdară”.
- Dacă răspunsul IsShared indică faptul că linia cache este partajată, atunci răspunsul PassDirty trebuie să indice că linia cache este ”curată”.

Forme de undă pentru tranzacția ReadNotSharedDirty –

stilul Ready_Before_Valid (”Finalizare înainte de validare”)



Succesiunea tranzacțiilor:

1. Solicitantul emite o cerere de citire pe canalul AR:
    - ARVALID este setat pentru a indica informații valide despre adresă.
    - ARADDR specifică adresa zonei din care se citește.
    - ARSNOOP = 0100 indică o tranzacție ReadNotSharedDirty.
    - ARDOMAIN = 01 sau 10 setează domeniul ca partajabil intern sau extern (”Inner Shareable” respectiv ”Outer Shareable”).
    - ARBAR = 0 setează atributele de protecție (acces securizat la date, neprivilegiat).
    - ARREADY este setat de interfață pentru a da acceptul în vederea transferului.
2. Interfața redirecționează cererea către controlerele de cache implicate, prin canalul AC:
    - ACADDR corespunde adresării originale a cererii.
    - ACSNOOP = 0100 indică o tranzacție de supraveghere (”Snoop”) tip ReadNotSharedDirty.
    - ACPROT = 000 setează atributele de protecție.
    - ACVALID și ACREADY trebuie setate înainte de a iniția orice răspuns tip ”Snoop” sau orice transfer de date.
3. Controlerele cache răspund pe canalul CR:
    - CRVALID este setat numai după ce atât ACVALID cât și ACREADY sunt setate.
    - CRRESP arată dacă linia cache a fost ”murdară” (contaminată) și dacă datele se vor transmite totuși.
4. Datele de coerență sunt transferate pe canalul CD:
    - CDVALID este setat numai după ce atât ACVALID cât și ACREADY sunt setate.

- CDDATA conține datele din linia cache supravegheată (”Snooped”) dacă aceasta e ”murdară” (contaminată).
- CDLAST este setat este setat spre a indica ultimul transfer din salvă.
- Solicitantul setează CDVALID fără să mai aștepte CRREADY sau CDREADY.

5. Furnizorul răspunde cu datele citite, pe canalul R:
    - RVALID este setat spre a indica validitatea datelor citite.
    - RDATA conține datele citite.
    - RRESP[3:0] conține informații ce arată dacă:
        - Datele sunt ”curate” sau ”murdare” (contaminate)
        - Citirea este exclusivă sau datele sunt partajate.
        - Starea pe care o va indica solicitantul: UniqueClean sau UniqueDirty.

- RLAST este setat spre a indica ultimul transfer din salvă.
- RREADY este setat de solicitant indicând că e gata să accepte datele.

6. Solicitantul setează RACK pentru a confirma citirea și a semnaliza finalizarea tranzacției.

Tranzacția se finalizează odată ce datele sunt returnate - fie dintr-un alt cache (dacă s-a semnalat contaminarea - ”murdărirea”), fie din memorie (dacă sunt ”curate” sau nu sunt disponibile în altă parte). Datele pot fi stocate în cache într-o stare UniqueClean sau SharedClean (”unice/partajate – curate”), în funcție de regulile de coerență ale sistemului.

Modificări uzuale ale stării liniei cache pentru ReadNotSharedDirty

Stare Inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
Invalid	UniqueClean (UC)	00
Invalid	UniqueDirty (UD)	01
Invalid	SharedClean (SC)	10

Alte schimbări permise ale liniei cache pentru ReadNotSharedDirty

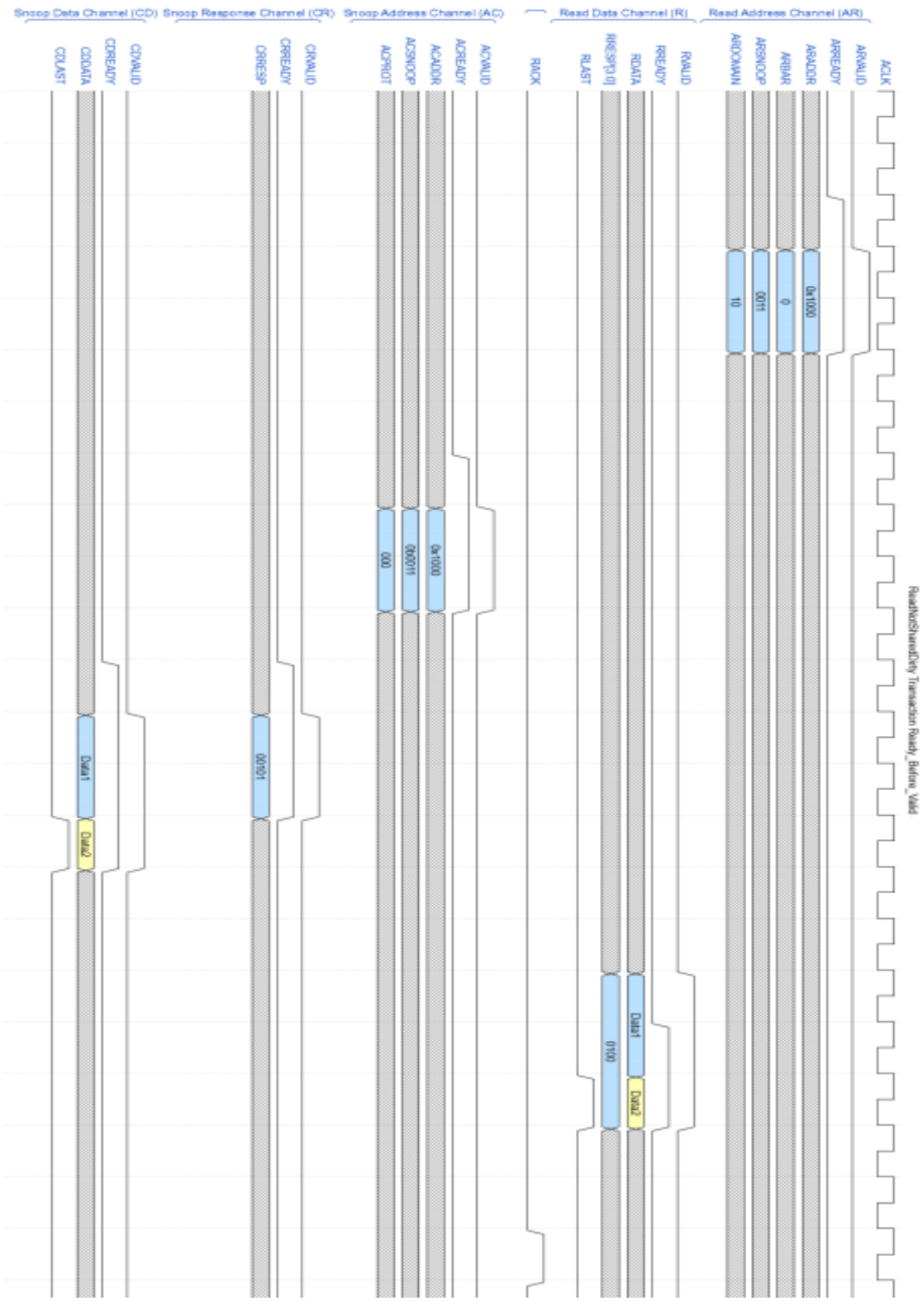
Starea inițială Cache	Stare Cache următoare	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	UniqueDirty(UD)	01
SharedClean(SC)	SharedClean(SC)	10
SharedDirty(SD)	SharedClean(SC)	00
SharedDirty(SD)	SharedDirty(SD)	10

- Modificările permise ale stării liniei cache și răspunsurile PassDirty și IsShared asociate supravegherii (”Snoop”) pentru tranzacțiile ReadClean, ReadShared, and ReadNotSharedDirty:

Starea inițială Cache	**Stare Cache finală**	**CRRESP[2]**		**CRRESP[3]**	**Descriere**
Invalid	Invalid (I)	0		0	Linia cache rămâne nevalidă
Unique Clean	Invalid (I)	0		0	Linia cache e ștearsă în ciuda faptului că este singura copie ”curată”
Unique Clean	Shared Clean (SC)	0		1	Linia cache rămâne ”curată”, dar este acum partajată cu alte cache
Shared Clean	Invalid (I)	0		0	Linia cache este ștearsă în timp ce există alte copii curate
Shared Clean	Shared Clean (SC)	0		1	Linia cache își menține starea de ”curată” & partajată
Unique Dirty	Invalid (I)	1		0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Unique Dirty	Shared Clean (SC)	1		1	Datele ”murdare” sunt transferate și linia cache devine ”curată” & partajată
Unique Dirty	Shared Dirty (SD)	0		1	Linia cache păstrează starea ”murdară”, dar trece la starea partajată
Shared Dirty	Invalid (I)	1		0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Shared Dirty	Shared Clean (SC)	1		1	Datele ”murdare” sunt transferate și linia cache devine ”curată” & partajată
Shared Dirty	Shared Dirty SD	0		1	Linia cache păstrează starea ”murdară” și partajată

Stilul Ready_Before_Valid (”Finalizare înainte de Validare”) –

forme de undă ReadNotSharedDirty



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                              },
      {name: 'ARREADY' , wave: '0.1..0.................
'                              },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0011
']             },  // ReadNotSharedDirty
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['10
']              },

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0..............1..0...
'                            },
      {name: 'RREADY'    , wave: '0..............1.0...
'                            },
      {name: 'RDATA'     , wave: 'x..............5.3x...', data: ['Data1',
'Data2'] },
      {name: 'RRESP[3:0]', wave: 'x..............5..x...', data: ['0100
']           },  // IsShared(0) + PassDirty(1) + OKAY(00) - UniqueDirty
      {name: 'RLAST'     , wave: '0..............10...
'                            },

{
}
    ],

    [
      {name: 'RACK', wave: '0.....................10
'                                  },
    ],

{
```

```
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
'                              },
      {name: 'ACREADY', wave: '0......1..0.............
'                              },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x1000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b0011
']             },  // ReadNotSharedDirty
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']                },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                              },
      {name: 'CRREADY', wave: '0..........1..0.........
'                              },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']              },  // WasUnique(0) + IsShared(0) + PassDirty,(1) + Error(0) +
DataTransfer(1)

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
```

```
'                                },
      {name: 'CDREADY', wave: '0..........1...0........
'                                },
      {name: 'CDDATA' , wave: 'x...........5.3x........', data: [' Data1', 'D
ata2']   },
      {name: 'CDLAST' , wave: '0.............10........
'                                },

{
}
    ],

{
},
  ],

  "head": {"text": "ReadNotSharedDirty Transaction Ready_Before_Valid
"                 },
  config:
  {hscale:
1                                                                    }
}
```

5. ReadShared

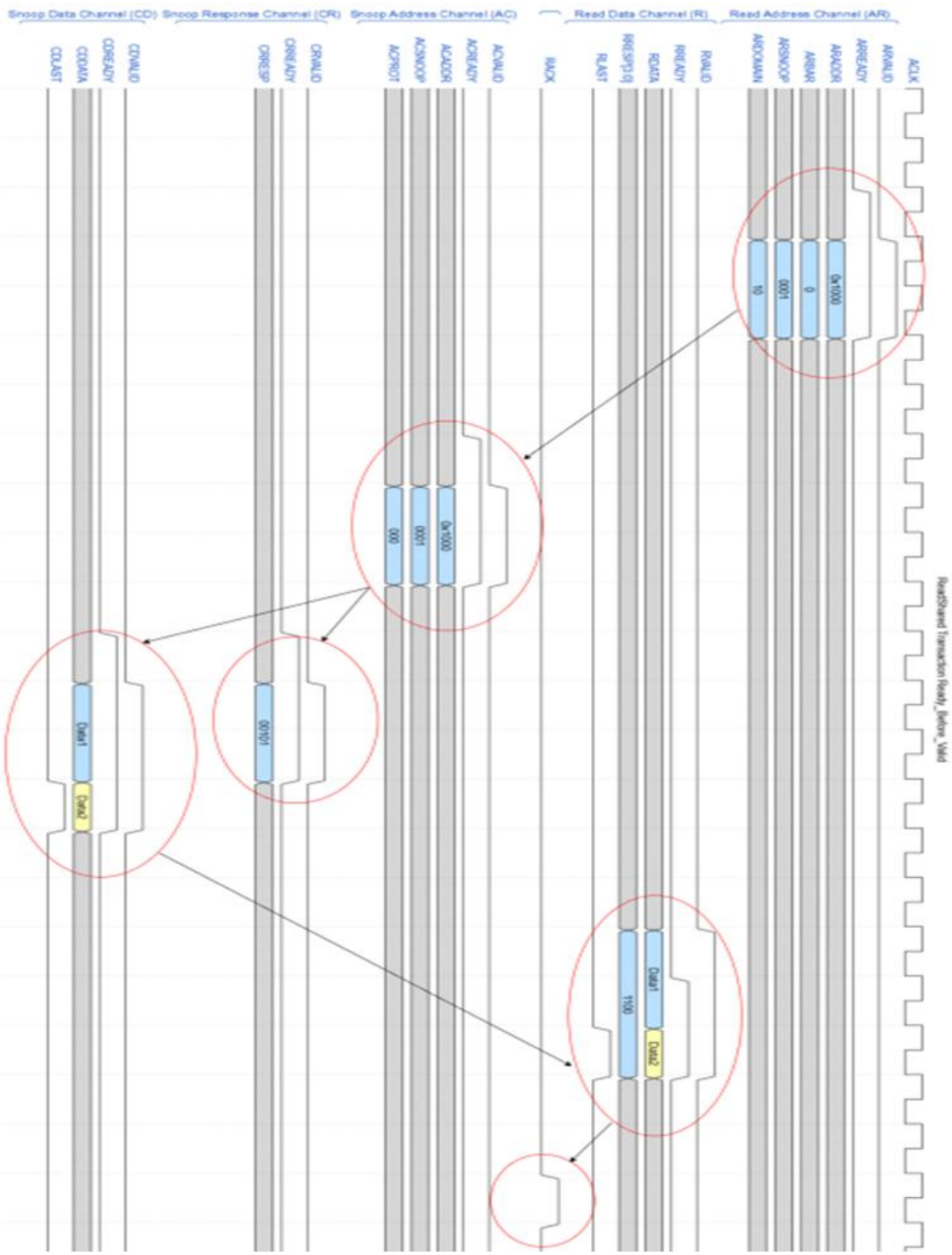
ReadShared este tot o tranzacție de citire care este utilizată într-o regiune de memorie ce poate fi partajată cu alți solicitanți. O tranzacție ReadShared se poate încheia cu orice combinație de răspunsuri IsShared și PassDirty. De obicei, o tranzacție ReadShared e folosită de un solicitant (înregistrat în cache) care preia date și poate accepta orice stare a liniei cache.

Cerințele privind răspunsurile la tranzacție sunt:

- Răspunsul IsShared arată că linia cache este partajată sau unică.
- Răspunsul PassDirty arată că linia cache e ”curată” sau ”murdară” (contaminate). Tranzacția ReadShared indică faptul că solicitantul unei citiri poate accepta linia cache în orice stare.

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) –

forme de undă ReadShared



1. Solicitantul emite o cerere de citire pe canalul AR:
    - ARVALID este setat pentru a indica informații valide despre adresă.
    - ARADDR = 0x0100 specifică adresa zonei din care se citește.
    - ARSNOOP = 0001 indică o tranzacție ReadShared.
    - ARDOMAIN = 01 or 10 setează domeniul ca partajabil intern sau extern (”Inner Shareable” respectiv ”Outer Shareable”).
    - ARBAR = 0 setează atributele de protecție (acces securizat la date, neprivilegiat).
    - ARREADY este setat de interfață pentru a da acceptul în vederea transferului.

2. Interfața redirecționează cererea către controlerele de cache implicate, prin canalul AC:
    - ACADDR = 0x0100 corespunde adresării originale a cererii.
    - ACSNOOP = 0001 indică o tranzacție de supraveghere (”Snoop”) tip ReadShared.
    - ACPROT = 000 setează atributele de protecție.
    - ACVALID și ACREADY trebuie setate înainte de a iniția orice răspuns tip ”Snoop” sau orice transfer de date.

3. Controlerele cache răspund pe canalul CR:
    - CRVALID este setat numai după ce atât ACVALID cât și ACREADY sunt setate.
    - CRRESP poate indica:
        - Dacă linia cache a fost în starea Modificată sau Exclusivă, atunci:
            - Linia cache furnizează datele (dacă ea deține versiunea lor cea mai recentă), apoi
            - îți retrogradează starea la Partajată sau Invalidă.
        - Dacă linia cache era deja în starea Partajată, această stare poate fi păstrată.

4. Datele de coerență sunt transferate pe canalul CD:
    - CDVALID este setat numai după ce atât ACVALID cât și ACREADY sunt setate.
    - Solicitantul setează CDVALID fără să mai aștepte CRREADY sau CDREADY.
    - CDDATA conține datele (de exemplu Date1 și Date2).
    - CDLAST este setat spre a indica ultimul transfer de date al salvei.

5. Furnizorul răspunde cu datele citite pe canalul R:
    - RVALID este setat pentru a indica date valide pentru citire.
    - RDATA = 0x0661 (exemplul din figură) conține datele furnizate.
    - RRESP[3:0] = 1000 indică:
        - OKAY = 00
        - IsShared = 1
        - PassDirty = 0
    - RLAST este setat spre a indica ultimul transfer din salvă.
    - RREADY este setat de solicitant indicând că e gata să accepte datele.
    - Solicitantul setează RACK pentru a confirma citirea și a semnaliza finalizarea tranzacției.

Tranzacția se finalizează cu datele stocate în cache în starea SharedClean, permițându-le să fie partajate cu alte procesoare fără a afecta coerența.

Modificări uzuale ale stării liniei cache pentru ReadShared

Stare Inițială Cache	Stare finală Cache	RRESP[3:2] IsShared/PassDirty
Invalid(I)	UniqueClean (UC)	00
Invalid(I)	UniqueDirty (UD)	01
Invalid(I)	SharedClean (SC)	10
Invalid(I)	SharedDirty (SD)	11

Alte modificări permise ale stării liniei cache pentru ReadShared

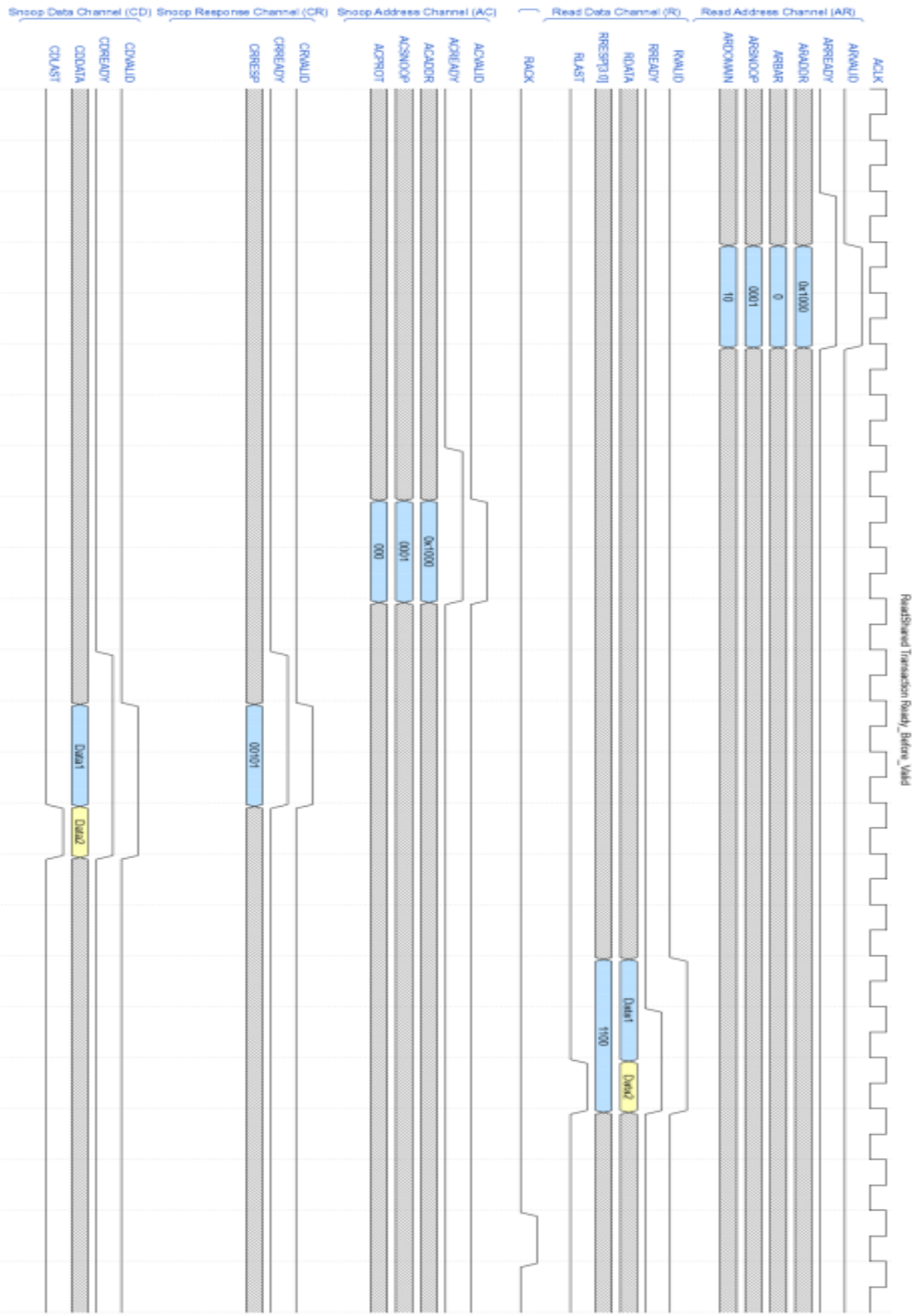
Stare Inițială Cache	Stare finală Cache	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	UniqueDirty(UD)	01
SharedClean(SC)	SharedClean(SC)	10
SharedClean(SC)	SharedDirty(SD)	11
SharedDirty(SD)	UniqueDirty(UD)	00
SharedDirty(SD)	SharedDirty(SD)	10

- Toate modificările permise ale stării liniei cache și răspunsurile PassDirty și IsShared asociate supravegherii (”Snoop”) pentru tranzacțiile ReadClean, ReadShared și ReadNotSharedDirty.

Stare Inițială Cache	**Stare Finală Cache**	**CRRESP [2]**	**CRRESP [3]**	**Descriere**
Invalid	Invalid (I)	0	0	Linia cache rămâne invalidă
Unique Clean	Invalid (I)	0	0	Linia cache este ștearsă în ciuda faptului că este singura copie ”curată”
Unique Clean	Shared Clean (SC)	0	1	Linia cache rămâne ”curată”, dar este acum partajată cu alte cache
Shared Clean	Invalid (I)	0	0	Linia cache este ștearsă în timp ce există alte copii ”curate”
Shared Clean	Shared Clean (SC)	0	1	Linia cache partajată își menține starea ”curată”
Unique Dirty	Invalid (I)	1	0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Unique Dirty	Shared Clean (SC)	1	1	Datele ”murdare” sunt transferate și linia cache devine partajată ”curată”
Unique Dirty	Shared Dirty (SD)	0	1	Linia cache devine partajată dar rămâne ”murdară”
Shared Dirty	Invalid (I)	1	0	Datele ”murdare” sunt transferate și linia cache este ștearsă
Shared Dirty	Shared Clean (SC)	1	1	Datele ”murdare” sunt transferate și linia cache devine partajată ”curată”
Shared Dirty	Shared Dirty SD	0	1	Linia cache partajată își menține starea ”murdară”

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV –

forme de undă pentru tranzacția ReadShared



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                              },
      {name: 'ARREADY' , wave: '0.1..0.................
'                              },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0001
']             },  // ReadShared
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['10
']              },

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.................1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............5.3x...', data: ['Data1',
'Data2'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['1100
']           },  // IsShared(1) + PassDirty(1) + OKAY(00) - SharedDirty
      {name: 'RLAST', wave: '0.................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0.....................10
'                                  },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
```

```
'                                },
      {name: 'ACREADY', wave: '0......1..0.............
'                                },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x1000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0001
']               },  // ReadShared
      {name: 'ACPROT', wave: 'x.......5.x.............', data: ['000
']                 },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0..........1.0.........
'                                },
      {name: 'CRREADY', wave: '0..........1..0.........
'                                },
      {name: 'CRRESP' , wave: 'x..........5.x.........', data: ['00101
']              },  // WasUnique(0) + IsShared(1) + PassDirty,(1) + Error(0) +
DataTransfer(1)

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0..........1..0........
'                                },
      {name: 'CDREADY', wave: '0..........1...0........
'                                },
      {name: 'CDDATA' , wave: 'x..........5.3x........', data: [' Data1', 'D
ata2']   },
      {name: 'CDLAST' , wave: '0............10........
'                                },

{
```

```
}
    ],

{
},
  ],

  "head": {"text": "ReadShared Transaction Ready_Before_Valid
"                          },
  config: {hscale:
1                                                                  }
}
```

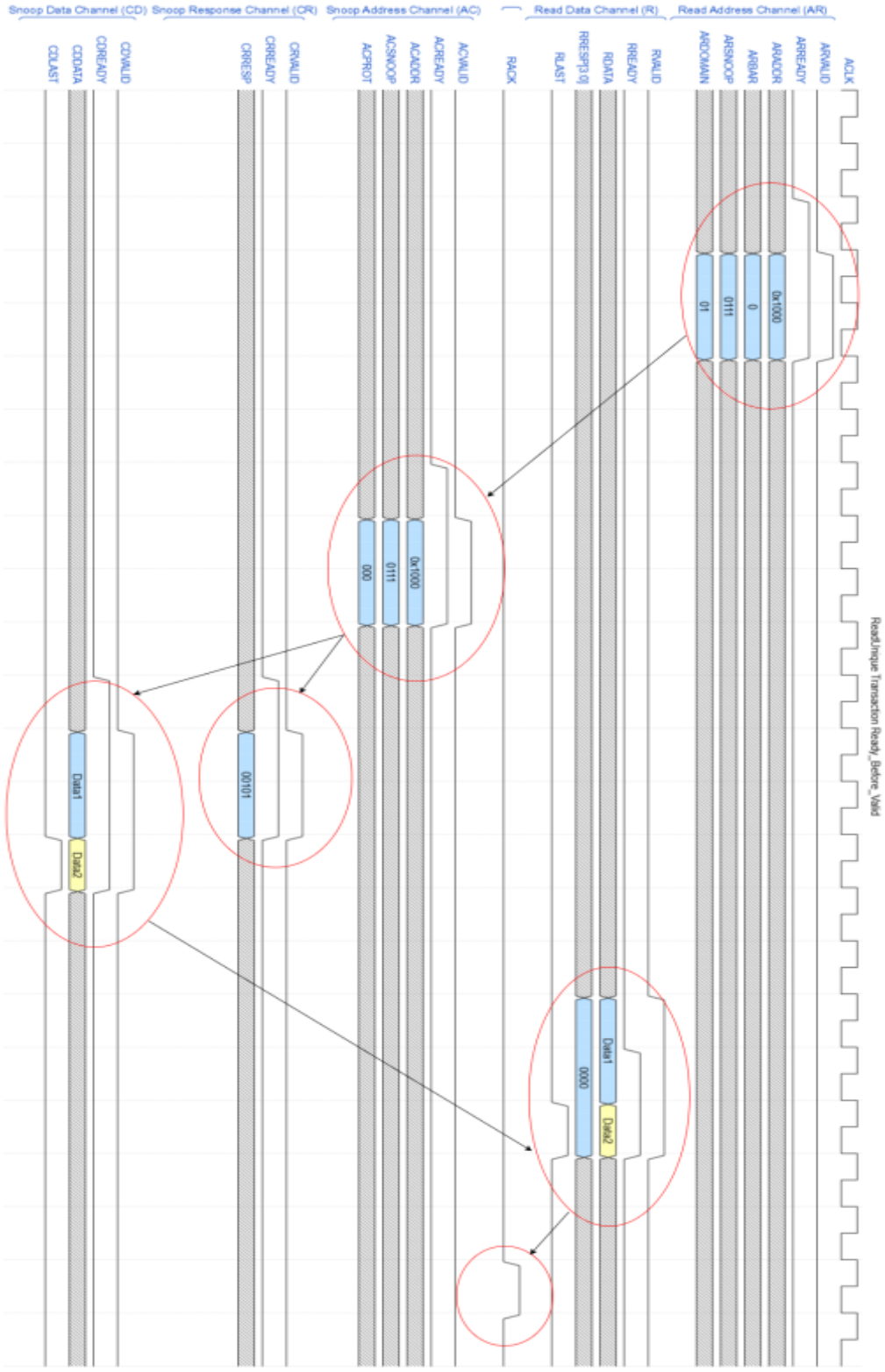
6. Tranzacția ReadUnique

O tranzacție ReadUnique este efectuată într-o regiune de memorie care poate fi partajată cu alți solicitanți. Tranzacția preia o copie a datelor și, de asemenea, asigură că linia cache poate fi menținută în starea Unique, ceea ce permite Solicitantului să efectueze o operațiune de stocare în linia cache. De obicei, o tranzacție ReadUnique este utilizată atunci când solicitantul inițiator efectuează o stocare parțială în linia cache fără să aibă o copie a acestei linii cache.

Cerințele privind răspunsurile la tranzacție sunt:

- Răspunsul IsShared trebuie (dezactivat) resetat pentru a indica faptul că linia cache este unică.
- Răspunsul PassDirty trebuie să indice când linia cache este ”curată” sau ”murdară”.

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă pentru tranzacția ReadUnique



1. Solicitantul emite o cerere de citire pe canalul AR:
    - ARVALID este setat pentru a indica informații valide despre adresă.
    - ARADDR specifică adresa zonei din care se citește.
    - ARSNOOP = 0111 indică o tranzacție ReadUnique.
    - ARDOMAIN = 01 or 10 setează domeniul ca partajabil intern sau extern (”Inner Shareable” respectiv ”Outer Shareable”).
    - ARBAR = 0 setează atributele de protecție (acces securizat la date, neprivilegiat).
    - ARREADY este setat de interfață pentru a da acceptul în vederea transferului.

2. Interfața redirecționează cererea către controlerele de cache implicate, prin canalul AC:
    - ACADDR corespunde adresării originale a cererii.
    - ACSNOOP = 0111 indică o tranzacție de supraveghere (”Snoop”) tip ReadUnique.
    - ACPROT = 000 setează atributele de protecție.
    - ACVALID and ACREADY trebuie setate înainte de a iniția orice răspuns tip ”Snoop” sau orice transfer de date.

3. Controlerele cache răspund pe canalul CR:
    - CRVALID este setat numai după ce atât ACVALID cât și ACREADY sunt setate.
    - Dacă linia cache a fost în starea Modificată:
        - Datele din ea trebuie rescrise în memorie sau furnizate direct solicitantului, apoi conținutul acestei copii locale trebuie invalidat.
    - Toate copiile partajate sau ”curate” din celelalte cache sunt de asemenea invalidate pentru a asigura exclusivitatea.

4. Datele de coerență sunt transferate pe canalul CD:
    - CDVALID este setat după ce atât ACVALID cât și ACREADY sunt setate.
    - CDDATA conține datele din cache supravegheat (”Snooped”) cache (dacă sunt ”murdare”).
    - CDLAST este setat spre a a indica ultimul transfer de date al salvei.
    - Solicitantul setează CDVALID fără să mai aștepte CRREADY sau CDREADY.

5. Furnizorul răspunde cu datele citite pe canalul R:
    - RVALID este setat pentru a indica date valide pentru citire.
    - RDATA conține datele furnizate.
    - RRESP[3:0] conține informații ce arată dacă:
        - Datele sunt ”curate” sau ”murdare” (contaminate)
        - Citirea este exclusivă sau datele sunt partajate.
        - Starea pe care o va indica solicitantul: UniqueClean sau UniqueDirty.
    - RLAST este setat spre a indica ultimul transfer din salvă.
    - RREADY este setat de solicitant indicând că e gata să accepte datele.

6. Solicitantul setează RACK pentru a confirma citirea și a semnaliza finalizarea tranzacției.

Tranzacția se finalizează odată ce datele sunt aduse (”fetched”) – fie din memorie sau dintr-un alt cache (dacă acesta conținea cea mai recentă versiune). Solicitantul primește o copie exclusivă a liniei cache, asigurându-se scrieri parțiale sigure fără a fi afectată coerența.

Modificări uzuale ale stării liniei cache pentru ReadUnique

Stare Inițială Cache	Stare Cache finală	RRESP[3:2] IsShared/PassDirty
Invalid(I)	UniqueClean (UC)	00
Invalid(I)	UniqueDirty (UD)	01

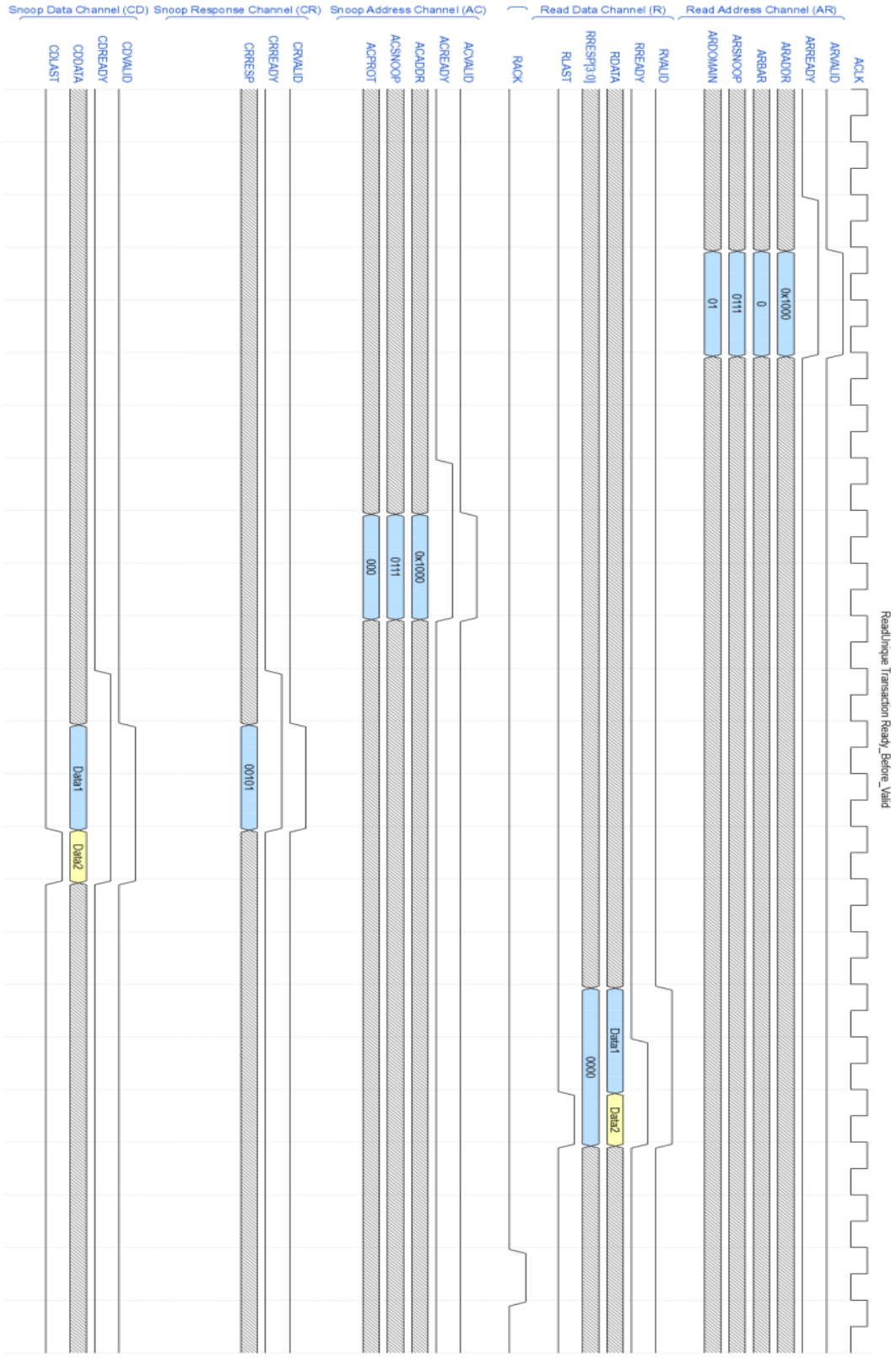
Alte modificări permise ale stării liniei cache pentru ReadUnique

Stare Inițială Cache	Stare Cache finală	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	UniqueDirty(UD)	01
SharedDirty(SD)	UniqueDirty(UD)	00

Modificări permise ale stării liniei cache pentru ReadUnique

Stare Inițială Cache	Stare Cache finală	CRRESP[3:2] PassDirty/IsShared
Invalid(I)	Invalid(I)	00
UniqueClean(UC)	Invalid(I)	00
UniqueDirty(UD)	Invalid(I)	10
SharedClean(SC)	Invalid(I)	00
SharedDirty(SD)	Invalid(I)	10

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă pentru tranzacția ReadUnique



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                                },
      {name: 'ARREADY' , wave: '0.1..0.................
'                                },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0111
']             },  // ReadUnique
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']              },  //Inner Shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............5.3x...', data: ['Data1',
'Data2'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['0000
']           },  // IsShared(0) + PassDirty(0) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0.................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0...................10
'                                  },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
'                                 },
      {name: 'ACREADY', wave: '0......1..0.............
'                                 },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x1000
']           },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0111
']              },  //ReadUnique
```

```
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']              },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                              },
      {name: 'CRREADY', wave: '0..........1..0.........
'                              },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']             },  // WasUnique(0) + IsShared(0) + PassDirty,(0) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                              },
      {name: 'CDREADY', wave: '0..........1...0........
'                              },
      {name: 'CDDATA' , wave: 'x...........5.3x........', data: [' Data1', 'D
ata2']   },
      {name: 'CDLAST' , wave: '0............10........
'                              },

{
}
    ],

{
},
  ],

  "head": {"text": "ReadUnique Transaction Ready_Before_Valid
"                          },
  config: {hscale:
1                                                                  }
}
```

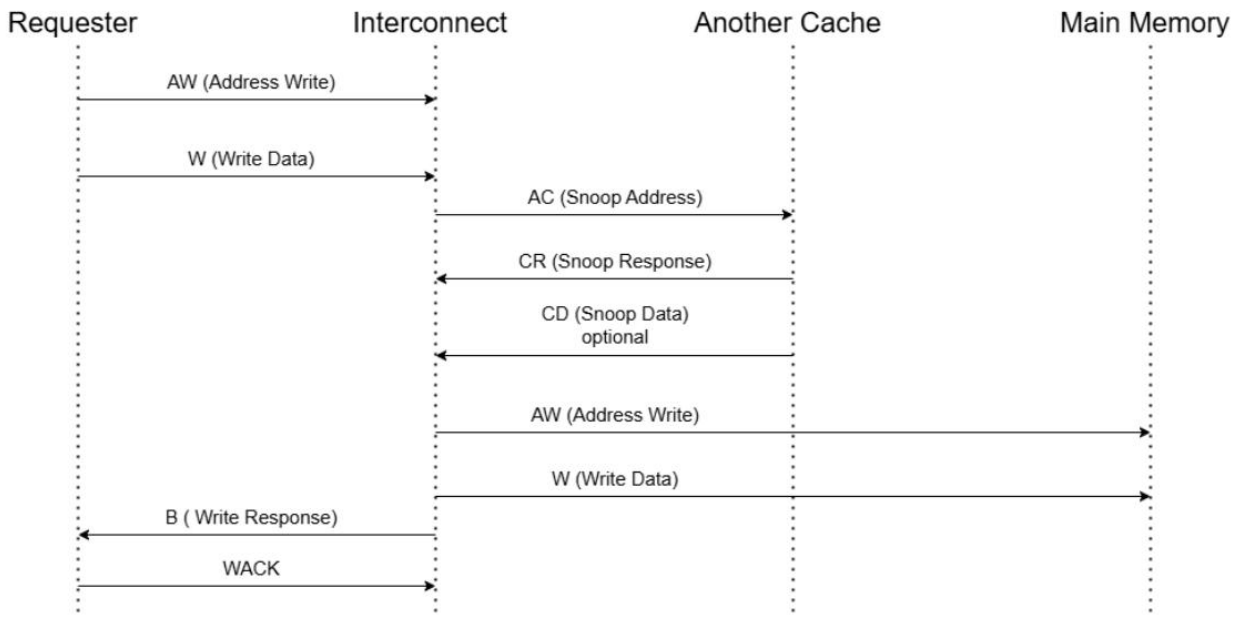
Tranzacții de scriere

Tranzacții care nu pot fi supuse unui regim normal de supraveghere (”Snoop”):

Următoarele tranzacții nu pot fi vizibile pe canalul de adrese ”Snoop” al unui solicitant înregistrat în cache:

- WriteNoSnoop
- WriteUnique
- WriteLineUnique
- WriteBack WriteClean
- WriteEvict

Semnalizarea tranzacțiilor de scriere:



1. Tranzacția WriteNoSnoop

O tranzacție WriteNoSnoop este utilizată pentru scrierea într-o regiune de memorie care nu poate fi partajată cu alți solicitanți.

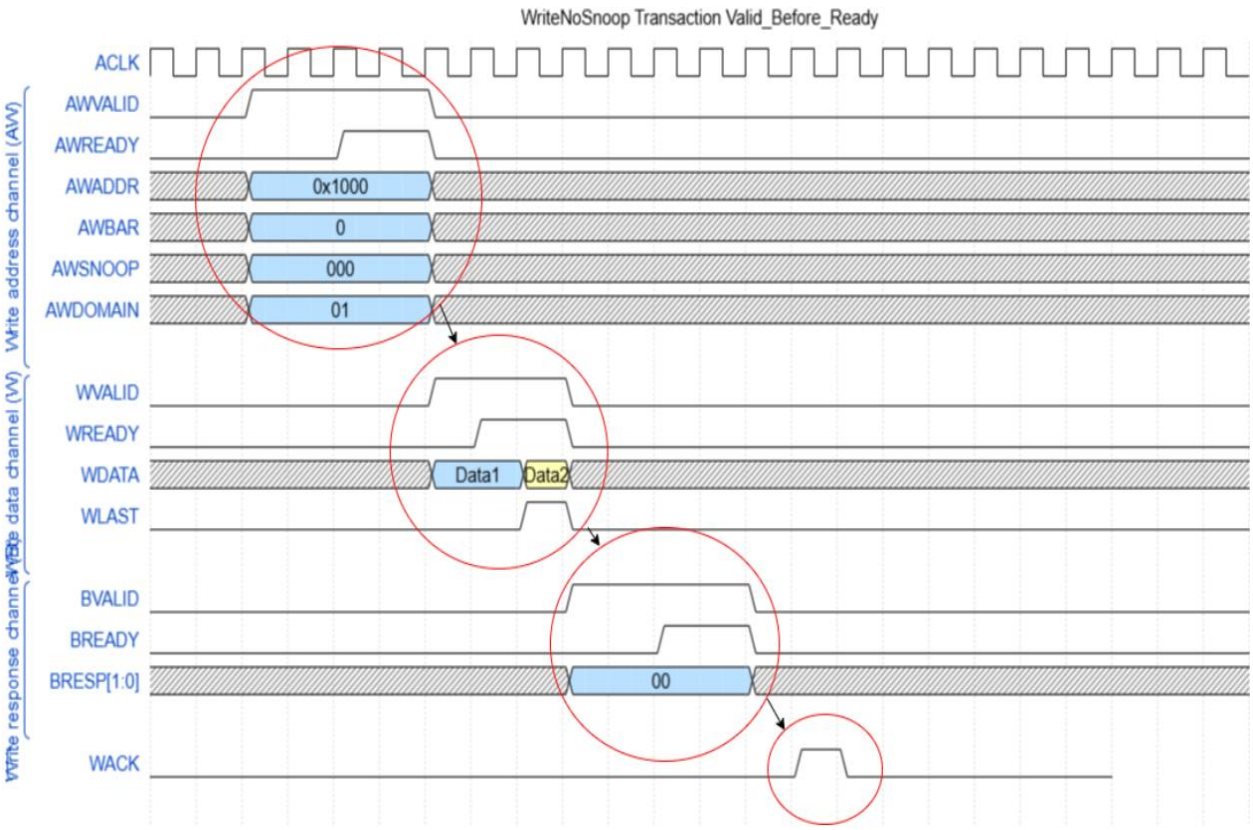
O tranzacție WriteNoSnoop poate rezulta din:

- O acțiune din program, cum ar fi o operațiune de stocare.
- O actualizare a memoriei principale cu înregistrare într-o linie cache care se află într-o regiune de memorie nepartajabilă.

O tranzacție WriteNoSnoop este o scriere non-coerentă utilizată atunci când:

- Zona de memorie nu poate fi partajată, ceea ce înseamnă că niciun alt solicitant nu este de așteptat să o acceseze sau să înregistreze în cache din acea memorie.
- Datele scrise nu necesită difuzarea în alte cache sau ”spionarea” de către alte sub-sisteme.

Stilul Valid_Before_Ready (”Validare înainte de finalizare”) – forme de undă WriteNoSnoop



1. Faza Write Address – scrierea adresei (pe Canalul AW)

- AWVALID este setată: Solicitantul stabilește o adresă validă.
- AWREADY este setată: Furnizorul acceptă această adresă.
- AWADDR = 0x1000: Adresa vizată pentru scriere.
- AWSNOOP = 000 : Indică o tranzacție WriteNoSnoop.
- AWDOMAIN = 01 : Domeniu partajabil intern.
- AWBAR = 0 : (acces) Fără restricții (”fără barieră”). Solicitantul inițiază o scriere fără a solicita acces exclusiv sau a declanșa operațiuni de supraveghere (”Snoop”).

2. Faza Write Data – scrierea datelor (pe Canalul W)

- WVALID și WREADY sunt setate: Datele sunt transmise și acceptate.
- WDATA = Data1, Data2: Două cicluri de scriere a datelor.
- WLAST: Indică ultimul ciclu al fazei de scriere a datelor. Solicitantul transmite datele care trebuie scrise la respectiva adresă.

3. Faza Write Response – scrierea răspunsului (pe canalul B)

- BVALID și BREADY sunt setate: Furnizorul trimite un răspunsul.
- BRESP = 00: răspuns OKAY, indicând că scrierea s-a efectuat cu succes. Tranzacția se încheie cu succes.

Schimbări uzuale de stare a liniei cache la WriteNoSnoop

Starea Inițială Cache	Starea finală Cache uzuală
Invalid(I)	Invalid(I)
UniqueClean(UC)	UniqueClean(UC)
UniqueDirty(UD)	UniqueClean(UC)

Alte schimbări uzuale de stare a liniei cache la WriteNoSnoop

Starea Inițială Cache	Starea finală Cache uzuală
SharedClean(SC)	UniqueClean(UC)
SharedDirty(SD)	UniqueClean(UC)

```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                    },

    ["Write address channel (AW)",
      {name: 'AWVALID' , wave: '0.1...0.................
'                                 },
      {name: 'AWREADY' , wave: '0...1.0.................
'                                 },
      {name: 'AWADDR'  , wave: 'x.5...x.................', data: ['0x1000
']           },
      {name: 'AWBAR'   , wave: 'x.5...x.................', data: ['0
']                },
      {name: 'AWSNOOP' , wave: 'x.5...x.................', data: ['000
']              },  // WriteNoSnoop
      {name: 'AWDOMAIN', wave: 'x.5...x.................', data: ['01
']               },  //Inner Shareable

{
}
    ],

    ["Write data channel (W)",
      {name: 'WVALID'    , wave: '0.....1..0..............
'                            },
      {name: 'WREADY'    , wave: '0......1.0..............
'                            },
      {name: 'WDATA'     , wave: 'x.....5.3x..............', data: ['Data1',
'Data2'] },
      {name: 'WLAST'     , wave: '0.......10..............
'                            },

{
}
    ],

    ["Write response channel (B)",
      {name: 'BVALID'    , wave: '0........1...0..........
'                            },
      {name: 'BREADY'    , wave: '0.........1.0..........
'                            },
```

```
      {name: 'BRESP[1:0]', wave: 'x........5...x.........', data: ['00
']             },

{
}
    ],

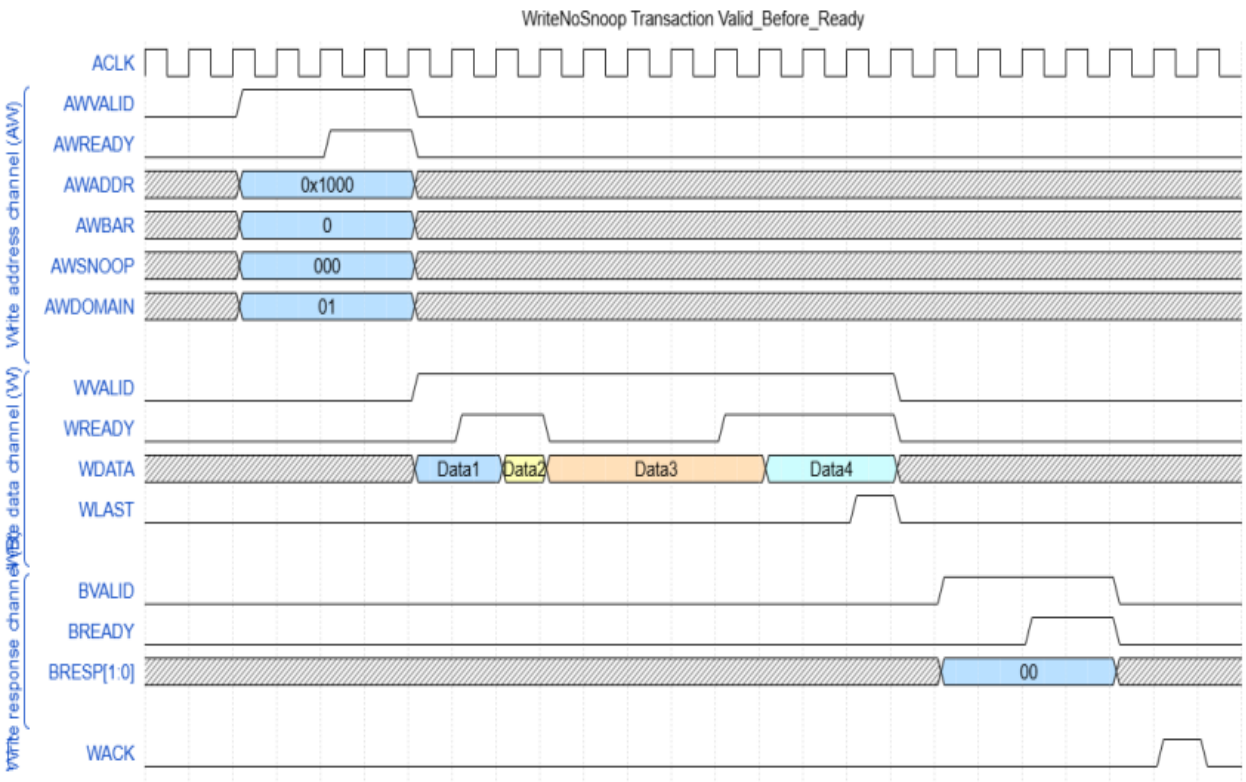
    [
      {name: 'WACK', wave: '0.............10.....
'                                  },
    ],

{
},


  ],

  "head": {"text": "WriteNoSnoop Transaction Valid_Before_Ready
"                       },
  config: {hscale:
1                                                          }
}
```

Un alt caz WriteNoSnoop (VBR):



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                                 },

    ["Write address channel (AW)",
      {name: 'AWVALID' , wave: '0.1...0.................
'                                                 },
      {name: 'AWREADY' , wave: '0...1.0.................
'                                                 },
      {name: 'AWADDR'  , wave: 'x.5...x.................', data: ['0x1000
']                                 },
      {name: 'AWBAR'   , wave: 'x.5...x.................', data: ['0
']                                    },
      {name: 'AWSNOOP' , wave: 'x.5...x.................', data: ['000
']                                      },  // WriteNoSnoop
      {name: 'AWDOMAIN', wave: 'x.5...x.................', data: ['01
']                                       },  //Inner Shareable

{
            }
    ],

    ["Write data channel (W)",
      {name: 'WVALID'    , wave: '0.....1.........0.......
'                                               },
      {name: 'WREADY'    , wave: '0......1.0...1...0.......
'                                               },
      {name: 'WDATA'     , wave: 'x.....5.34....6..x.......', data: ['Data1',
'Data2', 'Data3', 'Data4'] },
      {name: 'WLAST'     , wave: '0...............10.......
'                                               },

{
            }
    ],

    ["Write response channel (B)",
      {name: 'BVALID'    , wave: '0.................1...0..
'                                               },
      {name: 'BREADY'    , wave: '0...................1.0..
'                                               },
      {name: 'BRESP[1:0]', wave: 'x.................5...x..', data: ['00
']                                 },

{
```

```
            }
    ],

    [
      {name: 'WACK', wave: '0.....................10
'                                                    },
    ],

{
              },


  ],

  "head": {"text": "WriteNoSnoop Transaction Valid_Before_Ready
"                                              },
  config: {hscale:
1
}
}
```

2. Tranzacția WriteBack

O componentă cache a solicitantului folosește o tranzacție WriteBack pentru a scrie o linie ”murdară” înapoi în memoria principală (pentru a elibera astfel o linie cache ce poate fi utilizată pentru o altă adresă). Componenta cache a solicitantului nu păstrează o copie a acestei liniii cache ”murdare”. O tranzacție WriteBack reprezintă o scriere ce poate fi efectuată atât în regiunile de memorie partajabile cât și în cele nepartajabile.

Modificări uzuale ale stării liniei cache pentru WriteBack într-o zonă partajabilă de memorie

Stare Inițială Cache	Stare Cache Finală
UniqueDirty(UD)	Invalid(I)
SharedDirty(SD)	Invalid(I)

Modificări uzuale ale stării liniei cache pentru WriteBack într-o zonă ne-partajabilă de memorie

Stare Inițială Cache	Stare Cache Finală
UniqueDirty(UD)	Invalid(I)
SharedDirty(SD)	Invalid(I)

O tranzacție WriteBack decurge după standardul AXI pentru scriere, folosind canalele AW (Address Write), W (Write Data) și B (Write Response), fără a fi necesară utilizarea canalelor de supraveghere (”Snoop”).

3. Tranzacția WriteClean

O tranzacție WriteClean este un tip de operație de scriere folosit atât în regiunile de memorie partajabile, cât și în cele nepartajabile în cadrul protocolului AMBA ACE (AXI Coherency Extensions) [7, pag. 205]. E destinată în special pentru rescrierea liniilor de cache ”murdare” în memorie sau într-un cache ierarhic inferior (nivel L+1) fără a de-aloca linia din cache-ul local.

- Scopul principal al unei tranzacții WriteClean este de a actualiza memoria cu cea mai recentă versiune a unei linii de cache care a fost modificată (de exemplu a devenit ”murdară” – cu conținut alterat), păstrând totodată conținutul în linia cache.
- Este folosită de obicei în scenarii în care se dorește ”curățarea” liniei cache, dar să o păstreze totuși alocată pentru citiri sau scrieri viitoare.

În regiunile partajabile, tranzacția WriteClean asigură actualizarea memoriei, menținând linia cache într-o stare SharedClean sau UniqueClean, în funcție de regulile de coerență ale sistemului. Acest lucru permite subsistemului cache să continue să participe la mecanismul de coerență fără a fi nevoie să recupereze conținutul liniei după scriere.

Modificări uzuale ale stării liniei cache pentru WriteClean într-o zonă partajabilă de memorie

Stare Inițială Cache	Stare Cache Finală
UniqueDirty(UD)	UniqueClean(UC)
SharedDirty(SD)	SharedClean(SC)

Modificări uzuale ale stării liniei cache pentru WriteClean într-o zonă ne-partajabilă de memorie

Stare Inițială Cache	Stare Cache Finală
UniqueDirty(UD)	UniqueClean(UC)
SharedDirty(SD)	UniqueClean(UC)

O tranzacție WriteClean decurge după standardul AXI pentru scriere, folosind canalele AW (Address Write), W (Write Data) și B (Write Response), fără a fi necesară utilizarea canalelor de supraveghere (”Snoop”).

4. Tranzacția WriteEvict

O tranzacție WriteEvict poate fi utilizată atunci când se șterge (se evacuează – ”evict”) o linie de cache ”curată”. Această tranzacție este folosită pentru a scrie linia la un nivel inferior al ierarhiei de cache (cum ar fi un cache L3 sau un cache de nivel de sistem). O tranzacție WriteEvict nu este necesară pentru a actualiza memoria principală. O tranzacție WriteEvict trebuie folosită doar în următoarele circumstanțe:

- Când linia cache e menținută în starea UniqueClean.
- Când linia de cache nu a fost preluată în mod ”speculativ” dintr-un alt domeniu cu alt tip de partajare.

O tranzacție WriteEvict poate fi anulată. O componentă a sistemului poate folosi proprietatea WriteEvict_Transaction pentru a declara dacă suportă tranzacții WriteEvict. Prin acest mecanism, orice solicitant trebuie să permită dezactivarea tranzacțiilor WriteEvict pentru a asigura funcționarea corectă și cu versiuni anterioare ale interfeței ACE. Dacă proprietatea WriteEvict_Transaction nu este declarată, ea se consideră implicit falsă (”capabilitate inexistentă”).

Modificări uzuale ale stării liniei cache pentru WriteEvict

Stare Inițială Cache	Stare Cache Finală
UniqueClean(UC)	Invalid(I)

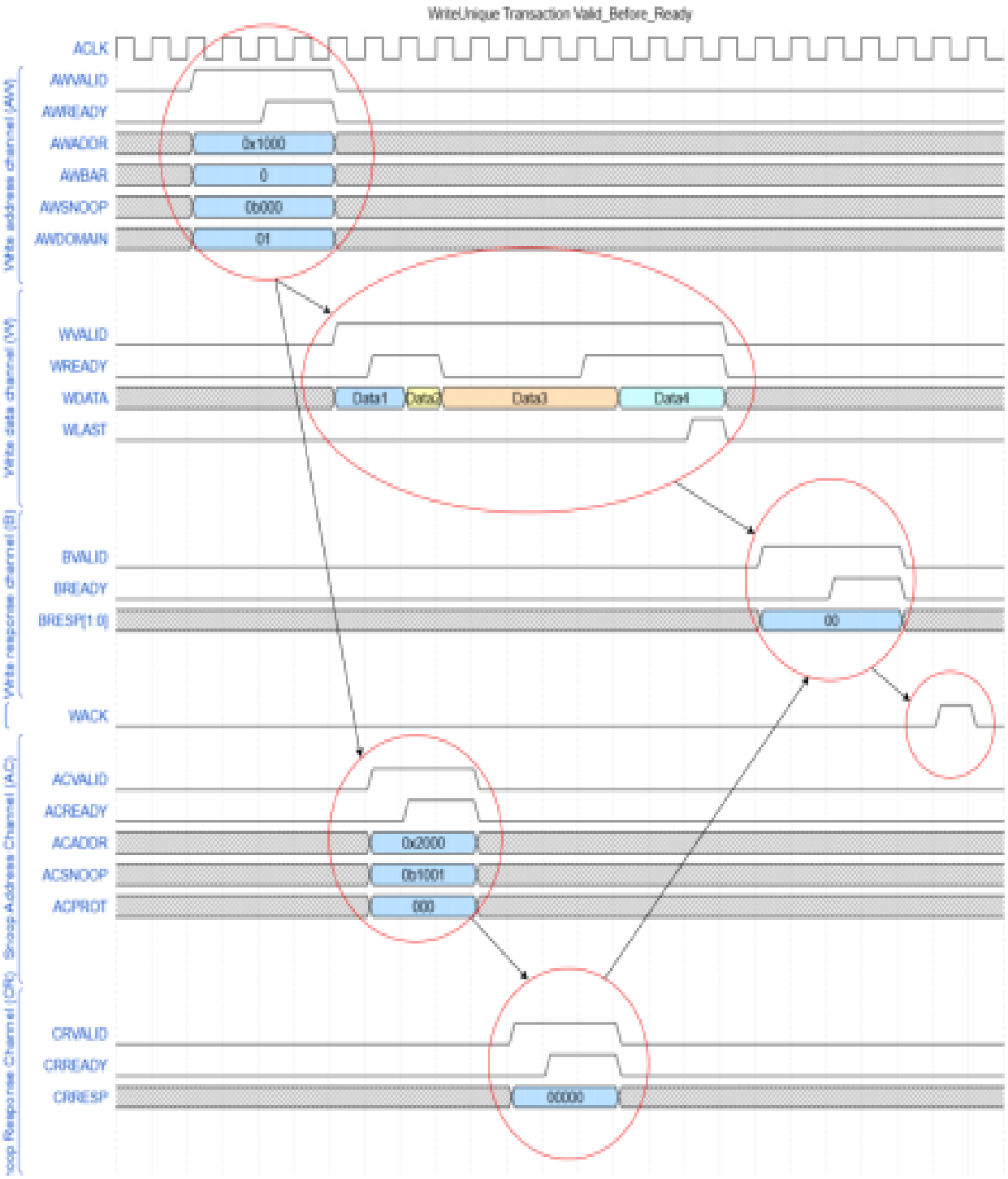
O tranzacție WriteEvict decurge după standardul AXI pentru scriere, folosind canalele AW (Address Write), W (Write Data) și B (Write Response), fără a fi necesară utilizarea canalelor de supraveghere (”Snoop”).

5. Tranzacția WriteLineUnique

O tranzacție WriteLineUnique poate fi folosită pentru a elimina toate copiile unei linii cache înainte de a inița o tranzacție de scriere. Tranzacția WriteLineUnique trebuie folosită doar atunci când se scrie o linie de cache completă (toți octeții din linia de cache sunt scriși prin tranzacție).

Stilul Valid_Before_Ready (”Validare înainte de finalizare”) –

forme de undă WriteLineUnique



1. Inițiere

   Solicitantul trimite o cerere de scriere cu:

   - AWSNOOP = WriteLineUnique (0b001)
   - AWDOMAIN = Domeniul partajabil (partajabil intern sau extern, 0b01, 0b10)
   - AWADDR = Adresa liniei cache
   - AWVALID = 1

2. Faza de supraveghere (”Snoop”). Tranzacția permisă este CleanInvalid (Snoop).
   Interfața trimite cereri de supraveghere (”snoop”) către alți agenți (sub-sisteme) pentru a:

   - Invalida orice copii ale liniei cache.
   - Acorda proprietatea exclusivă destinatarului.

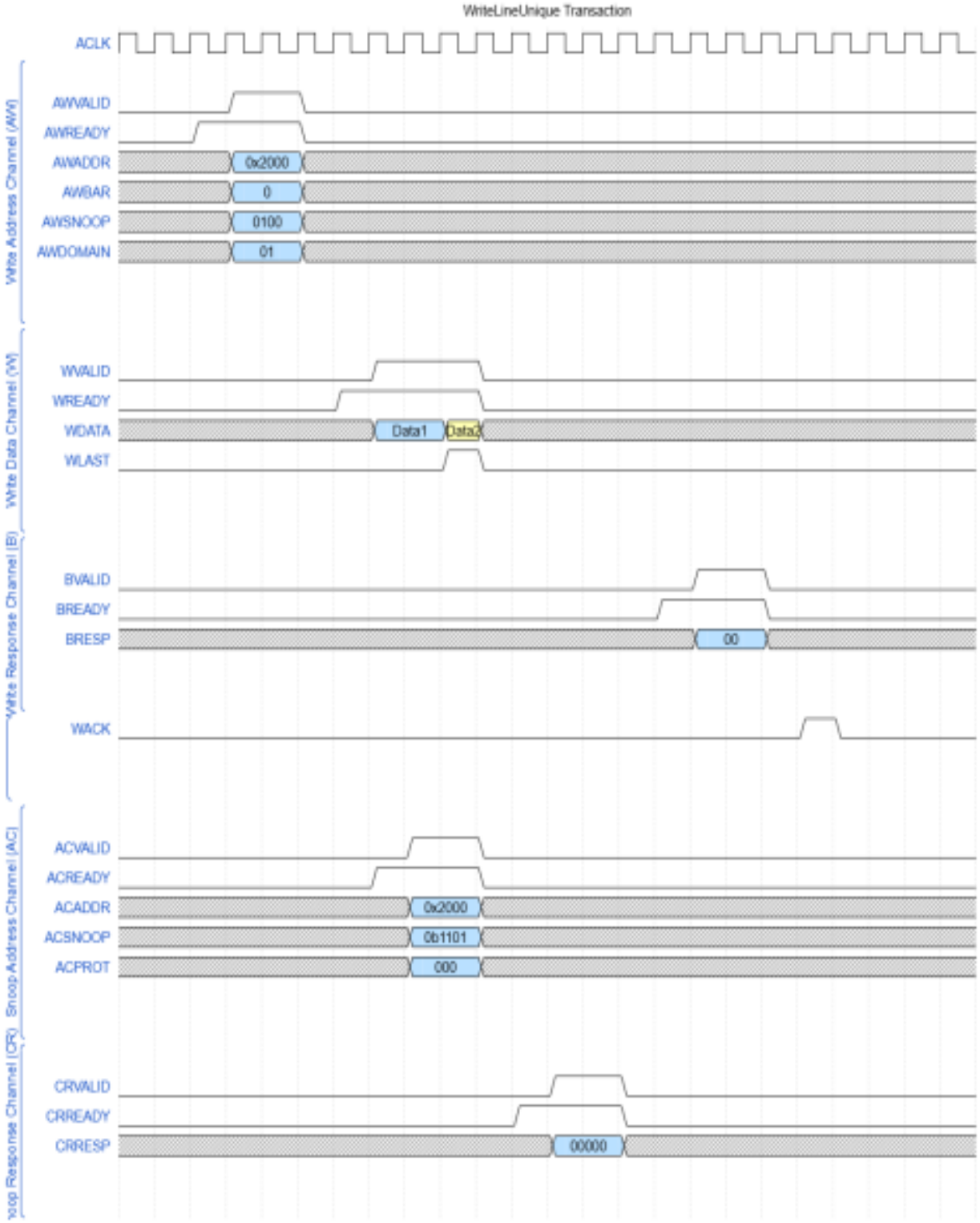
3. Transferul datelor

   - Solicitantul trimite datele complete din linia cache, pe canalul W.
   - Furnizorul acceptă datele atunci când este activat (setat) WREADY.

4. Încheiere

   - Furnizorul trimite un răspuns pe canalul B.
   - Solicitantul primește răspunsul și finalizează tranzacția.

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă WriteLineUnique



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................'},

    ["Write Address Channel (AW)",
      {},
      {name: 'AWVALID' , wave: '0..1.0.................'},
      {name: 'AWREADY' , wave: '0.1..0.................'},
      {name: 'AWADDR'  , wave: 'x..5.x.................', data: ['0x2000']},
      {name: 'AWBAR'   , wave: 'x..5.x.................', data: ['0']},
      {name: 'AWSNOOP' , wave: 'x..5.x.................', data: ['0100']},
// WriteLineUnique
      {name: 'AWDOMAIN', wave: 'x..5.x.................', data: ['01']},   /
/ Inner Shareable
      {},
      {}
    ],

    ["Write Data Channel (W)",
      {},
      {name: 'WVALID' , wave: '0......1..0............'},
      {name: 'WREADY' , wave: '0.....1...0............'},
      {name: 'WDATA'  , wave: 'x......5.3x............', data: ['Data1', 'Da
ta2']},
      {name: 'WLAST'  , wave: '0........10............'},
      {},
      {}
    ],

    ["Write Response Channel (B)",
      {},
      {name: 'BVALID' , wave: '0..............1.0.....'},
      {name: 'BREADY' , wave: '0.............1..0.....'},
      {name: 'BRESP'  , wave: 'x..............5.x.....', data: ['00']},  //
OKAY
      {},
      {}
    ],

    ["Snoop Address Channel (AC)",
      {},
```

```
      {name: 'ACVALID', wave: '0.......1.0.............'},
      {name: 'ACREADY', wave: '0......1..0.............'},
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000']},
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1101']},
// MakeInvalid
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000']},
      {},
      {}
    ],

    ["Snoop Response Channel (CR)",
      {},
      {name: 'CRVALID', wave: '0...........1.0.........'},
      {name: 'CRREADY', wave: '0..........1..0.........'},
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00000']},
// No data transfer
      {},
      {}
    ]
  ],

  "head": {"text": "WriteLineUnique Transaction"},
  "config": {"hscale": 1}
}
```

6. Tranzacția WriteUnique

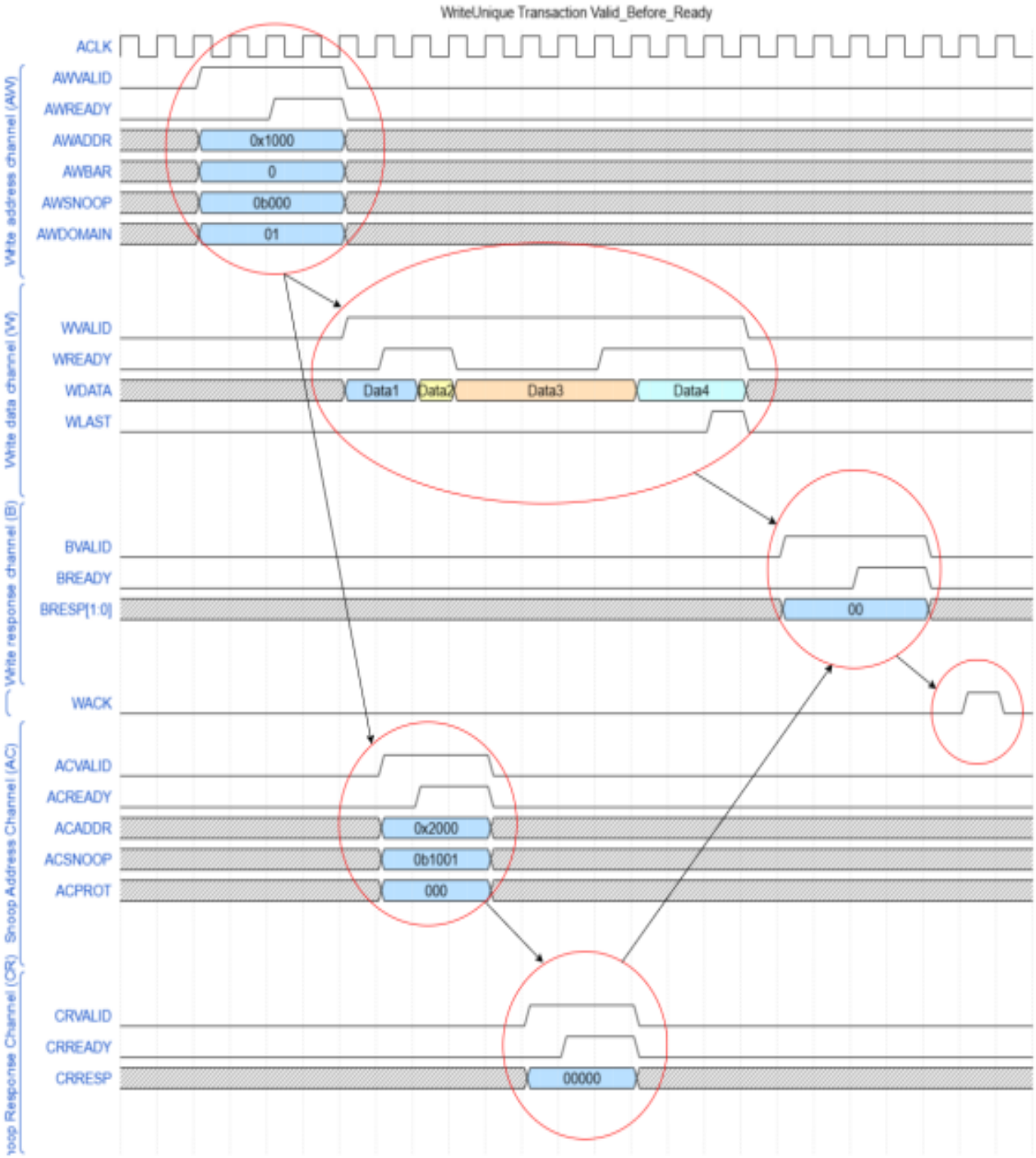
O tranzacție WriteUnique poate fi folosită pentru a elimina toate copiile unei linii de cache înainte de a iniția o tranzacție de scriere.

Tranzacția WriteUnique poate fi folosită atunci când se scrie o linie de cache completă sau parțială și asigură că datele ”murdare” sunt copiate în memorie înainte de efectuarea tranzacției de scriere.

Desfășurarea tranzacției:

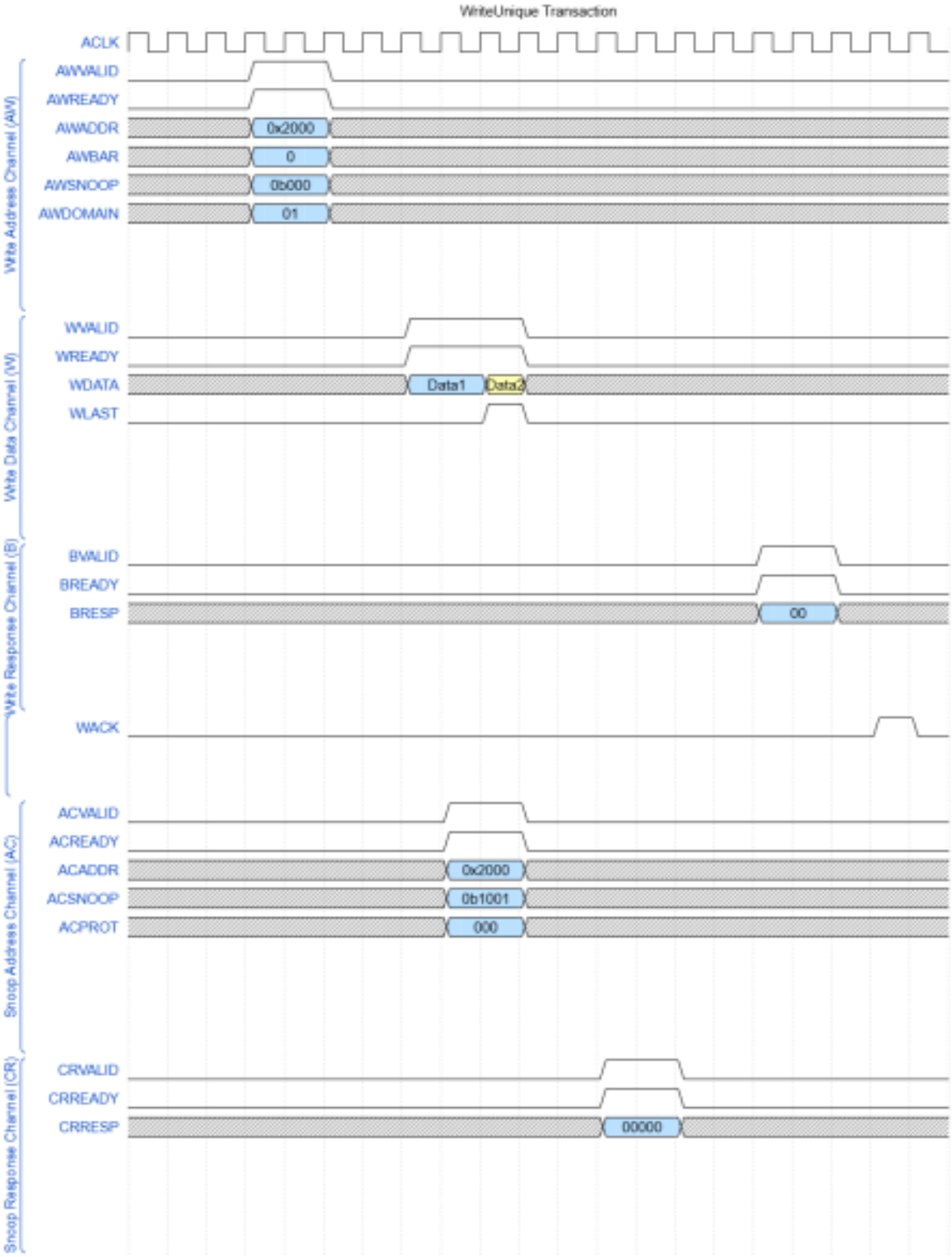
Stilul Valid_Before_Ready (”Validare înainte de finalizare”) –

forme de undă WriteUnique



1. Faza Write Address (canalul AW)
    - AWVALID și AWREADY activate (setate): Aceasta indică generarea și acceptarea unei adrese valide.
    - AWADDR = 0x2000: Adresa la care se scrie.
    - AWSNOOP = 0b000 : Această valoare indică o tranzacție WriteUnique.
    - AWDOMAIN = 01 : Domeniu partajabil intern.
    - AWBAR = 0 : (acces) Fără restricții (”fără barieră”). Solicitantul cere acces exclusiv la linia cache de la adresa 0x2000.
2. Faza de supraveghere (”Snoop”) – canalul AC
    - ACVALID și ACREADY setate (activate): Interfața trimite o cerere de supraveghere (”snoop”) către alte sub-sisteme master.
    - ACSNOOP = 0b1001: Supraveghere (”snoop”) de tip CleanInvalid – celelalte cache trebuie să șteargă și să invalideze copiile pe care le dețin.
    - ACADDR = 0x2000: Aceeași cu adresa de scriere. Se asigură astfel că niciun alt master nu deține o copie validă a liniei cache.
3. Faza Snoop Response (canalul CR)
    - CRVALID și CRREADY activate (setate): Master-ul supus supravegherii (”snooped”) răspunde.
    - CRRESP = 00000: Indică faptul că nu este nevoie de transfer de date (nu există date ”murdare” în alte cache). Supravegherea (”snoop”) se finalizează cu succes, iar master-ul inițiator are acum proprietatea exclusivă asupra datelor.
4. Faza Write Data (canalul W)
    - WVALID și WREADY activate (setate): Datele sunt trimise și acceptate.
    - WDATA = Data1, Data2: Două cicluri de scriere a datelor.
    - WLAST: Indică ultimul ciclu de scriere. Master-ul scrie datele pe linia cache al cărei unic deținător este acum.
5. Faza Write Response (canalul B)
    - BVALID și BREADY activate (setate): Sub-sistemul Slave răspunde.
    - BRESP = 00: Răspuns OKAY, indicând succesul tranzacției. Scrierea este acum completă.

Stilul Valid_And_Ready (”Validare și finalizare”) – VAR – forme de undă WriteUnique



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................'},

    ["Write Address Channel (AW)",
      {name: 'AWVALID' , wave: '0..1.0.............'},
      {name: 'AWREADY' , wave: '0..1.0.............'},
      {name: 'AWADDR'  , wave: 'x..5.x.............', data: ['0x2000']},
      {name: 'AWBAR'   , wave: 'x..5.x.............', data: ['0']},
      {name: 'AWSNOOP' , wave: 'x..5.x.............', data: ['0b000']}, //
WriteUnique
      {name: 'AWDOMAIN', wave: 'x..5.x.............', data: ['01']},   // I
nner Shareable
      {},
      {},
      {}
    ],

    ["Write Data Channel (W)",
      {name: 'WVALID' , wave: '0......1..0.........'},
      {name: 'WREADY' , wave: '0......1..0.........'},
      {name: 'WDATA'  , wave: 'x......5.3x.........', data: ['Data1', 'Data2
']},
      {name: 'WLAST'  , wave: '0........10.........'},
      {},
      {},
      {},
      {}
    ],

    ["Write Response Channel (B)",
      {name: 'BVALID' , wave: '0..............1.0..'},
      {name: 'BREADY' , wave: '0..............1.0..'},
      {name: 'BRESP'  , wave: 'x..............5.x..', data: ['00']},  // OKA
Y
      {},
      {},
      {}
    ],
    [
      {name: 'WACK', wave: '0.................10
'                                                 },
      {},
      {},
    ],

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.........'},
      {name: 'ACREADY', wave: '0.......1.0.........'},
```

```
      {name: 'ACADDR' , wave: 'x.....5.x.........', data: ['0x2000']},
      {name: 'ACSNOOP', wave: 'x.....5.x.........', data: ['0b1001']},  //
CleanInvalid
      {name: 'ACPROT' , wave: 'x.....5.x.........', data: ['000']},
      {},
      {},
      {},
      {}
    ],

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0..........1.0.....'},
      {name: 'CRREADY', wave: '0..........1.0.....'},
      {name: 'CRRESP' , wave: 'x..........5.x.....', data: ['00000']},  //
No data transfer
      {},
      {},
      {},
      {}
    ]
  ],

  "head": {"text": "WriteUnique Transaction with Snoop Channels"},
  "config": {"hscale": 1}
}
```

WriteUnique Valid_Before_Ready (”Validare înainte de finalizare”) – VBR:

```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                                  },

    ["Write address channel (AW)",
      {name: 'AWVALID' , wave: '0.1...0.................
'                                          },
      {name: 'AWREADY' , wave: '0...1.0.................
'                                          },
      {name: 'AWADDR'  , wave: 'x.5...x.................', data: ['0x1000
']                               },
      {name: 'AWBAR'   , wave: 'x.5...x.................', data: ['0
']                                    },
      {name: 'AWSNOOP' , wave: 'x.5...x.................', data: ['0b000
']                                },  // WriteUnique
      {name: 'AWDOMAIN', wave: 'x.5...x.................', data: ['01
']                                   },  //Inner Shareable
```

```
{
                }
    ],

    ["Write data channel (W)",

{
                },
      {name: 'WVALID'    , wave: '0.....1..........0.......
'                                                 },
      {name: 'WREADY'    , wave: '0......1.0...1...0.......
'                                                 },
      {name: 'WDATA'     , wave: 'x.....5.34....6..x.......', data: ['Data1',
'Data2', 'Data3', 'Data4'] },
      {name: 'WLAST'     , wave: '0...............10.......
'                                                 },

{
                },

{
                }
    ],

    ["Write response channel (B)",

{
                },
      {name: 'BVALID'    , wave: '0.................1...0..
'                                                 },
      {name: 'BREADY'    , wave: '0...................1.0..
'                                                 },
      {name: 'BRESP[1:0]', wave: 'x.................5...x..', data: ['00
']                                              },

{
                },

{
                }
    ],

    [
      {name: 'WACK', wave: '0.....................10
'                                                    },
    ],
    ["Snoop Address Channel (AC)",
```

```
{
            },
      {name: 'ACVALID', wave: '0......1..0.............
'                                                 },
      {name: 'ACREADY', wave: '0.......1.0.............
'                                                 },
      {name: 'ACADDR' , wave: 'x......5..x.............', data: ['0x2000
']                                  },
      {name: 'ACSNOOP', wave: 'x......5..x.............', data: ['0b1001
']                                   },  // CleanInvalid
      {name: 'ACPROT' , wave: 'x......5..x.............', data: ['000
']                                     },

{
            },

{
            }
    ],

    ["Snoop Response Channel (CR)",

{
            },
      {name: 'CRVALID', wave: '0..........1..0..........
'                                                 },
      {name: 'CRREADY', wave: '0...........1.0..........
'                                                 },
      {name: 'CRRESP' , wave: 'x..........5..x..........', data: ['00000
']                                    },  // No data transfer

{
            },

{
            }
    ],

  ],

  "head": {"text": "WriteUnique Transaction Valid_Before_Ready
"                                               },
  config: {hscale:
1
}
}
```

Tranzacții pentru Mentenanța Cache

Sub-sistemele solicitantului folosesc tranzacții de mentenanță (întreținere) pentru a accesa și întreține cache ale altor sub-sisteme solicitante din sistem.

În particular, aceste tranzacții de întreținere permit sub-sistemelor master să verifice efectul operațiunilor de încărcare și stocare asupra unor cache din sistem care altfel nu pot fi accesate.

Acest proces este denumit de obicei ”mentenanța software” pentru cache.

Difuzarea directivelor de mentenanță cache se poate ”propaga” și către cache următoare, permițând întreținerea tuturor sub-sistemelor cache dintr-un sistem.

1. Tranzacția CleanShared

O tranzacție CleanShared este folosită pentru a asigura că toate copiile unei anumite locații de memorie (stocate în cache diferite) sunt ”curate”, corespunzând datelor din memoria principală, fără a fi fost modificate local.

În timp ce o tranzacție CleanShared este în desfășurare, solicitantul are permisiunea de a scrie pe acea linie, astfel încât aceasta poate deveni ”murdară” înainte ca tranzacția CleanShared să se finalizeze.

Desfășurarea tranzacției:

1. Solicitantul inițiază o tranzacție de citire cu:
   - `ARSNOOP` = CleanShared (0b1000)
   - `ARADDR` = 0x1000

- ARVALID = 1
- Interfața este deja pregătită (ARREADY = 1) înainte ca ARVALID să fie activat (setat), permițând ca semnalizările (”hand-shaking”) să înceapă imediat.

2. Interfața trimite o cerere de supraveghere (”snoop”) pe canalul AC:
   - ACSNOOP = CleanShared (0b1000)
   - ACADDR = 0x1000
   - ACVALID = 1
   - Interfața este de asemenea pregătită (ARREADY = 1) înainte de activarea ARVALID.
   - Cache supravegheat (”snooped”) răspunde cu:
     - CRRESP = 01101:
       - WasUnique = 0
       - IsShared = 1
       - PassDirty = 1
       - Error = 0
       - DataTransfer = 1 → Indicates SharedClean
     - CRVALID = 1, CRREADY = 1
   - Cache supravegheat (”snooped”) returnează date ”murdare” pe canalul CD:
   - CDDATA = Data1, Data2
   - CDVALID = 1, CDREADY = 1
   - CDLAST = 1 la sfârșitul salvei.

4. Canalul Read Data
   - Interfața returnează datele ”curățate” către solicitant:
   - RDATA = Data1, Data2

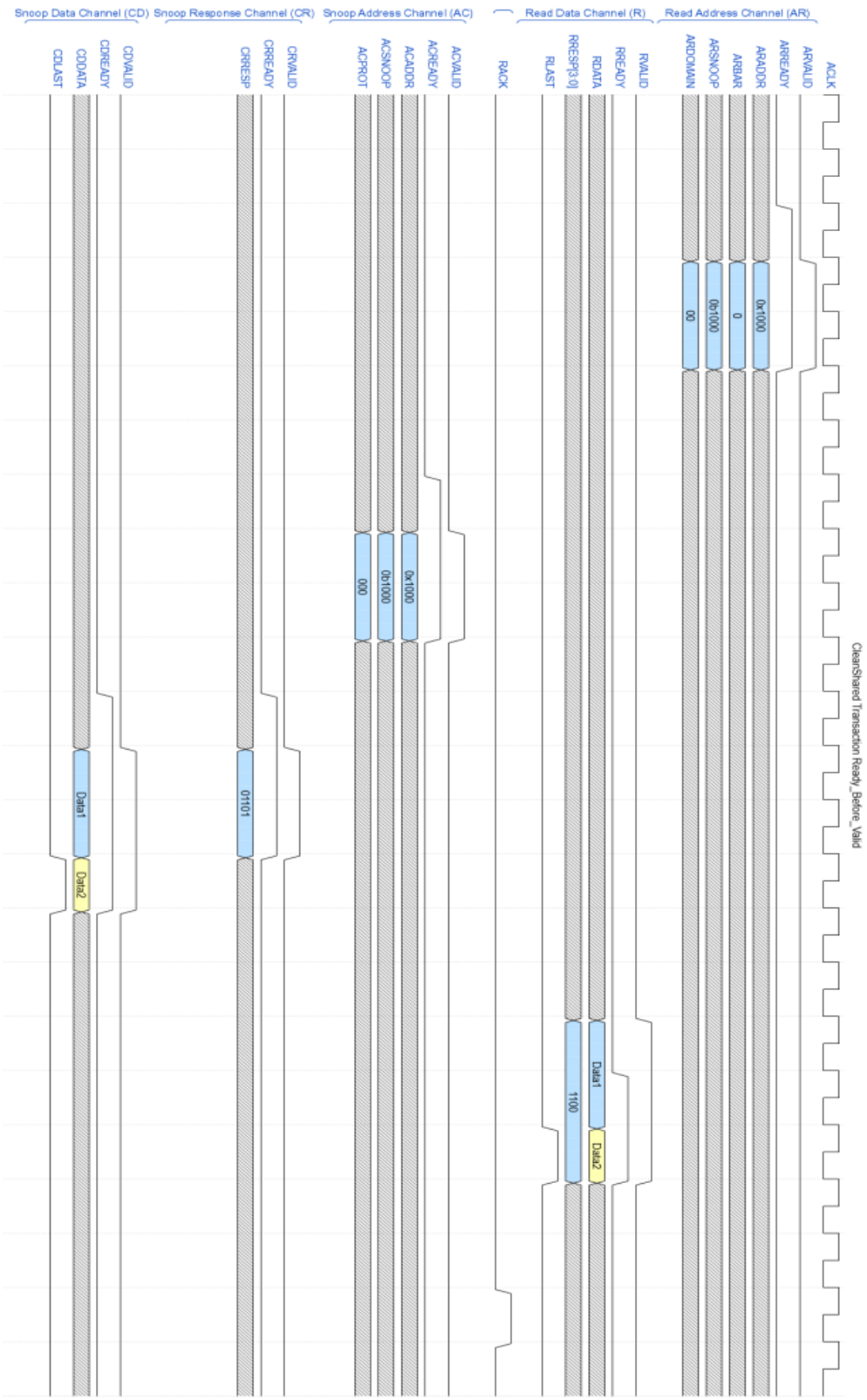
- RRESP = 1100:
    - IsShared = 1
    - PassDirty = 1
    - OKAY = 00 → Indică UniqueClean
- RVALID = 1, RREADY = 1
- RLAST = 1 la final

5. Încheiere
    - Solicitantul confirmă completarea citirii cu: RACK = 1

Modificări uzuale ale stării liniei cache pentru CleanShared

Stare Initială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
Invalid(I)	Invalid(I)	00
Invalid(I)	Invalid(I)	10
UniqueClean(UC)	UniqueClean(UC)	00
SharedClean(SC)	UniqueClean(UC)	00
SharedClean(SC)	SharedClean(SC)	10

CleanShared



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                              },
      {name: 'ARREADY' , wave: '0.1..0.................
'                              },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x1000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0b1000
']           },  // CleanShared
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['00
']               },  //Non-shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0..............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............5.3x...', data: ['Data1',
'Data2'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['1100
']           },  // IsShared(1) + PassDirty(1) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0...................10
'                                  },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
```

```
'                                },
      {name: 'ACREADY', wave: '0......1..0.............
'                                },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x1000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1000
']             },  //CleanShared
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']                },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                                },
      {name: 'CRREADY', wave: '0..........1..0.........
'                                },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['01101
']              },  // WasUnique(0) + IsShared(1) + PassDirty,(1) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                                },
      {name: 'CDREADY', wave: '0..........1...0........
'                                },
      {name: 'CDDATA' , wave: 'x...........5.3x........', data: [' Data1', 'D
ata2']   },
      {name: 'CDLAST' , wave: '0............10........
'                                },

{
```

```
}
    ],

{
},
  ],

  "head": {"text": "CleanShared Transaction Ready_Before_Valid
"                        },
  config: {hscale:
1                                                                  }
}
```

2. Tranzacția CleanInvalid

Un sub-sistem solicitant folosește o tranzacție CleanInvalid pentru a efectua o operație de curățare și invalidare asupra cache aferente altor sub-sisteme. Dacă un cache supravegheat (”snooped”) care deține o linie cache ”curată” primește o directivă (o comandă) CleanInvalid, va trebui să elimine copia locală a liniei cache. Dacă un cache supravegheat (”snooped”) care deține o linie cache ”murdară” primește o directivă CleanInvalid, va trebui să furnizeze conținutul acelei linii cache spre a putea fi scris în memoria principală și să elimine copia lui locală a liniei cache.

Modificări uzuale ale stării liniei cache pentru CleanInvalid

Stare Initială Cache	Stare Cache Finală	PassDirty, CRRESP[2]	IsShared, CRRESP[3]
Invalid(I)	Invalid(I)	0	0
UniqueClean(UC)	Invalid(I)	0	0
UniqueDirty(UD)	Invalid(I)	1	0
SharedClean(SC)	Invalid(I)	0	0
SharedDirty(SC)	Invalid(I)	1	0

Desfășurarea tranzacției

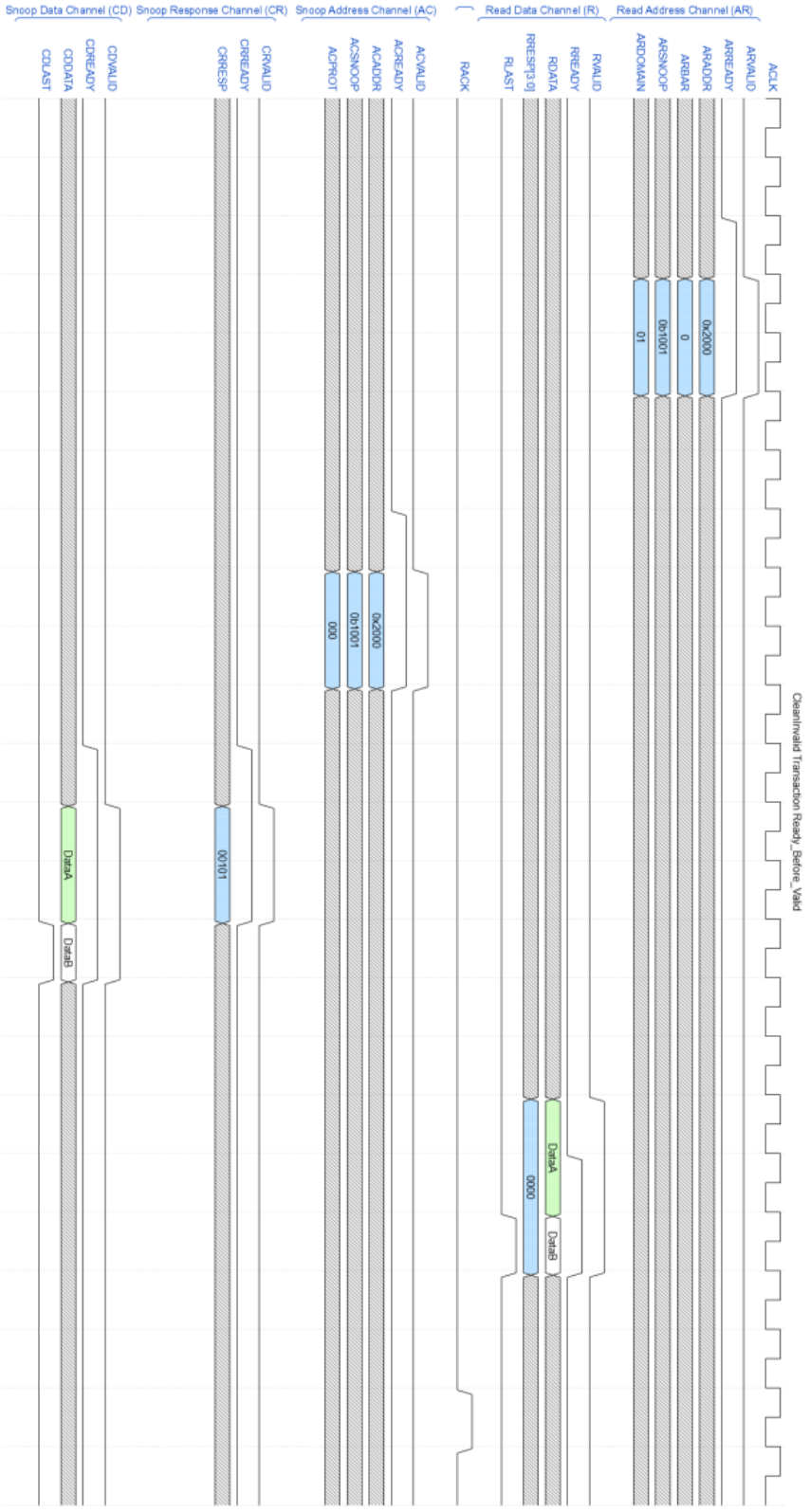
1. Inițiere – adresarea de citire – canal Read Address (AR)
   - Solicitantul inițiază o tranzacție de citire cu:
     - ARSNOOP = CleanInvalid (0b1001)
     - ARADDR = 0x2000
     - ARVALID = 1
2. Interfața este deja pregătită (ARREADY = 1) înainte ca ARVALID să fie activat (setat), permițând ca semnalizările (”hand-shaking”) să înceapă imediat.

3. Interfața trimite o cerere de supraveghere (”snoop”) pe canalul AC:
     - ACSNOOP = CleanInvalid (0b1001)
     - ACADDR = 0x2000
     - ACVALID = 1
     - ACREADY = 1 înainte de ACVALID (este permis un ”handshake” imediat).
   - Fiecare cache supravegheat (”snooped”) evaluează calitatea copiei deținute:
   - Dacă e ”murdară”, atunci:
     - Trimite datele prin canalul CD.
     - Răspunde cu CRRESP = 00101:
       - WasUnique = 0
       - IsShared = 0
       - PassDirty = 1
       - Error = 0
       - DataTransfer = 1 → Indică SharedClean
   - Dacă e ”curată” atunci invalidează linia.
   - Dacă nu e disponibilă, răspunde corespunzător.

- Răspunsul la supraveghere (”Snoop”):
    - CRVALID = 1, CRREADY = 1
    - Date ”murdare” returnate pe canalul CD:
        - CDDATA = DataA, DataB
        - CDVALID = 1, CDREADY = 1
        - CDLAST = 1 la final.

4. Citirea datelor – pe canalul (R)
    - Interfața returnează datele ”curățate” solicitantului:
    - RDATA = DataA, DataB
    - RRESP = 0000:
        - IsShared = 0
        - PassDirty = 0
        - OKAY = 00 → Indică UniqueClean
    - RVALID = 1, RREADY = 1
    - RLAST = 1 la terminarea datelor
5. Încheierea
    - Solicitantul confirmă finalizarea citirii:
        - RACK = 1

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă CleanInvalid



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                                },
      {name: 'ARREADY' , wave: '0.1..0.................
'                                },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x2000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0b1001
']           },  // CleanInvalid
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']               },  // Inner Shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............7.2x...', data: ['DataA',
'DataB'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['0000
']           },  // IsShared(0) + PassDirty(0) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0.................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0.....................10
'                                  },
    ],

{
},

    ["Snoop Address Channel (AC)",
```

```
      {name: 'ACVALID', wave: '0.......1.0.............
'                              },
      {name: 'ACREADY', wave: '0......1..0.............
'                              },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1001
']             },  // CleanInvalid
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']                },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                              },
      {name: 'CRREADY', wave: '0..........1..0.........
'                              },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']              },  // WasUnique(0) + IsShared(0) + PassDirty(1) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                              },
      {name: 'CDREADY', wave: '0..........1...0........
'                              },
      {name: 'CDDATA' , wave: 'x...........7.2x........', data: ['DataA', 'Da
taB']    },
      {name: 'CDLAST' , wave: '0............10........
'                              },
```

```
{
}
    ],

{
},
  ],

  "head": {"text": "CleanInvalid Transaction Ready_Before_Valid
"                        },
  "config": {"hscale":
1                                                          }
}
```


3. Tranzacția CleanUnique

O tranzacție CleanUnique este folosită într-o regiune de memorie care poate fi partajată cu alte sub-sisteme master.

Tranzacția CleanUnique asigură că:

- Linia cache poate fi menținută în starea Unique. Aceasta permite sub-sistemului master să efectueze o operațiune de stocare în linia cache, dar tranzacția nu furnizează o copie a datelor și pentru master.
- Datele stocate într-un alt cache în starea ”murdare” sunt scrise în memoria principală și toate celelalte copii ale liniei cache sunt șterse.

De obicei, o tranzacție CleanUnique este folosită înainte de o stocare parțială a liniilor cache într-o zonă de memorie partajabilă, atunci când master-ul deține deja o copie a datelor.

Cerințele de răspuns la tranzacție sunt:

- Răspunsul IsShared trebuie dezactivat (resetat) pentru a indica faptul că linia de cache este unică.
- Răspunsul PassDirty trebuie dezactivat (resetat).

Ca răspuns la supraveghere (”Snoop”) aceasta folosește același cod `0b1001` pentru `ACSNOOP` ca și tranzacția CleanInvalid descrisă anterior.

La finalizarea unei tranzacții CleanUnique, master-ul care a inițiat-o primește permisiunea de a stoca datele în linia cache.

Dacă linia cache era în starea Invalidă înainte de această operațiune de stocare, atunci (pentru o corectă alocare) zona de stocare trebuie să aibă dimensiunea completă a unei linii cache.

După stocarea completă a liniei cache ea trece în starea UniqueDirty.

Stocarea trebuie să aibă loc ”atomizat”, odată cu finalizarea tranzacției CleanUnique.

Prin urmare, orice supraveghere (”snoop”) inițiată după tranzacția CleanUnique trebuie amânată până când această stocare este finalizată.

Desfășurarea tranzacției

1. Inițiere – adresarea de citire – canal Read Address (AR)
   - Solicitantul inițiază o tranzacție de citire cu:
     - `ARSNOOP` = CleanUnique (0b1011)
     - `ARADDR` = 0x2000
     - `ARVALID` = 1

2. Interfața este deja pregătită (ARREADY = 1) înainte ca ARVALID să fie activat (setat), permițând ca semnalizările (”hand-shaking”) să înceapă imediat.

3. Interfața trimite o cerere de supraveghere (”snoop”) pe canalul AC:
     - `ACSNOOP` = CleanInvalid (0b1001)
     - `ACADDR` = 0x2000
     - `ACVALID` = 1

- Interfața este pregătită (ACREADY = 1) înainte de ACVALID.
- Cache supravegheat (”snooped”) răspunde cu:
    - CRRESP = 00101:
        - WasUnique = 0
        - IsShared = 0
        - PassDirty = 1
        - Error = 0
        - DataTransfer = 1 → Indică SharedClean
    - CRVALID = 1, CRREADY = 1
- Cache supravegheat (”snooped”) returnează date ”murdare” (alterate) pe canalul CD:
    - CDDATA = DataA, DataB
    - CDVALID = 1, CDREADY = 1
    - CDLAST = 1 la terminarea datelor.

4. Citirea datelor – pe canalul (R)
    - Interfața returnează datele ”curățate” solicitantului:
    - RDATA = DataA, DataB
    - RRESP = 0000:
        - IsShared = 0
        - PassDirty = 0
        - OKAY = 00 → Indică UniqueClean
    - RVALID = 1, RREADY = 1
    - RLAST = 1 la ultimul ciclu de citire a datelor.

5. Încheierea
    - Solicitantul confirmă finalizarea citirii:
        - RACK = 1

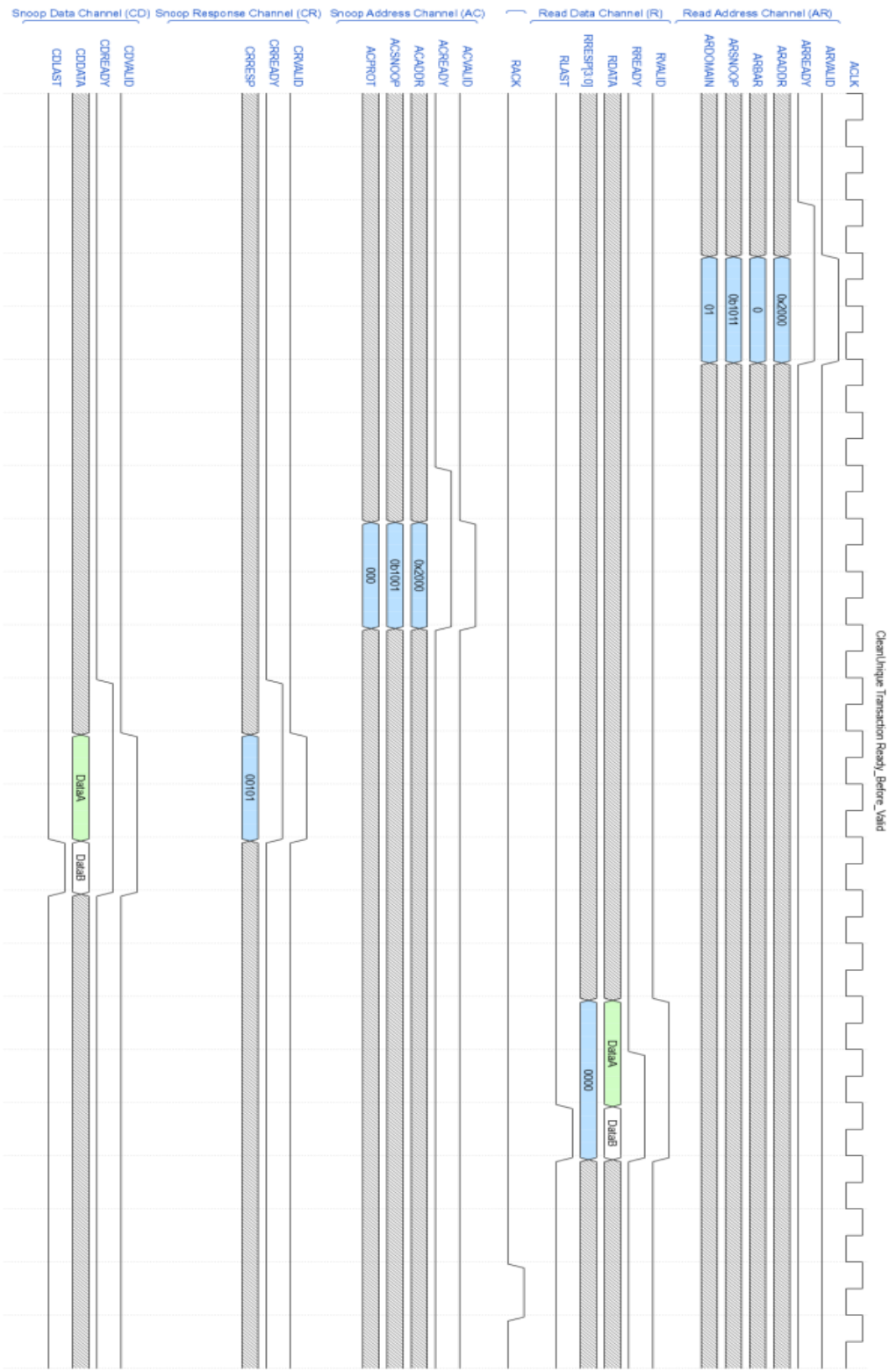
Modificări uzuale ale stării liniei cache pentru CleanUnique

Stare Inițială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
SharedClean(SC)	UniqueClean(UC)	00
SharedDirty(SD)	UniqueDirty(UD)	00

Alte modificări permise ale stării liniei cache pentru CleanUnique

Stare Inițială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
Invalid(I)	Invalid(I)	00
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă pentru tranzacția CleanUnique



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                                },
      {name: 'ARREADY' , wave: '0.1..0.................
'                                },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x2000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0b1011
']           },  // CleanUnique
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']               },  // Inner Shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............7.2x...', data: ['DataA',
'DataB'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['0000
']           },  // IsShared(0) + PassDirty(0) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0..................10
'                                 },
    ],

{
},

    ["Snoop Address Channel (AC)",
```

```
      {name: 'ACVALID', wave: '0.......1.0.............
'                              },
      {name: 'ACREADY', wave: '0......1..0.............
'                              },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1001
']             },  // CleanInvalid
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']                },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                              },
      {name: 'CRREADY', wave: '0..........1..0.........
'                              },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']              },  // WasUnique(0) + IsShared(0) + PassDirty(1) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                              },
      {name: 'CDREADY', wave: '0..........1...0........
'                              },
      {name: 'CDDATA' , wave: 'x...........7.2x........', data: ['DataA', 'Da
taB']    },
      {name: 'CDLAST' , wave: '0............10........
'                              },
```

```
{
}
    ],

{
},
  ],

  "head": {"text": "CleanUnique Transaction Ready_Before_Valid
"                        },
  "config": {"hscale":
1                                                          }
}
```


4. Tranzacția MakeInvalid

Un solicitant folosește o tranzacție MakeInvalid pentru a efectua o operațiune de invalidare asupra cache din alte sub-sisteme. Dacă un cache supravegheat (”snooped”) e obiectul unei tranzacții MakeInvalid, el trebuie să șteargă copia locală, dar nu este obligat să furnizeze datele, chiar dacă linia cache este ”murdară”. O tranzacție MakeInvalid difuzează așadar (”broadcast”) o directivă de invalidare a sub-sistemelor cache. Poate fi folosită atât în regiuni de memorie partajabile cât și în cele nepartajabile. O tranzacție MakeInvalid este folosită pentru a se asigura că nu mai există nicăieri în cache copii ale unei locații principale de memorie.

Desfășurarea tranzacției

1. Inițiere – adresarea de citire – canal Read Address (AR)
   - Solicitantul inițiază o tranzacție de citire cu:
     - `ARVALID` = 1
     - `ARSNOOP` = MakeInvalid (0b1101)
     - `ARADDR` = 0x2000
   - Interfața este deja pregătită (ARREADY = 1) înainte ca ARVALID să fie activat (setat), permițând ca semnalizările (”hand-shaking”) să înceapă imediat.

2. Interfața trimite o cerere de supraveghere (”snoop”) pe canalul AC:
    - ACSNOOP = MakeInvalid (0b1101)
    - ACADDR = 0x2000
    - ACVALID = 1
- Interfața este pregătită (ACREADY = 1) înainte de ACVALID.
- Cache supravegheat (”snooped”) răspunde cu:
    - CRRESP = 00101:
        - WasUnique = 0
        - IsShared = 0
        - PassDirty = 1
        - Error = 0
        - DataTransfer = 1 → Indică SharedClean
    - CRVALID = 1, CRREADY = 1
- Cache supravegheat (”snooped”) returnează date ”murdare” (alterate) pe canalul CD:
    - CDDATA = DataA, DataB
    - CDVALID = 1, CDREADY = 1
    - CDLAST = 1 la terminarea datelor.
3. Citirea datelor – pe canalul (R)
    - Interfața returnează datele ”curățate” solicitantului:
        - RDATA = DataA, DataB
        - RRESP = 0000:
            - IsShared = 0
            - PassDirty = 0
            - OKAY = 00 → Indică UniqueClean
        - RVALID = 1, RREADY = 1
        - RLAST = 1 la ultimul ciclu de citire a datelor.

4. Încheierea
    - Solicitantul confirmă finalizarea citirii:
        - RACK = 1

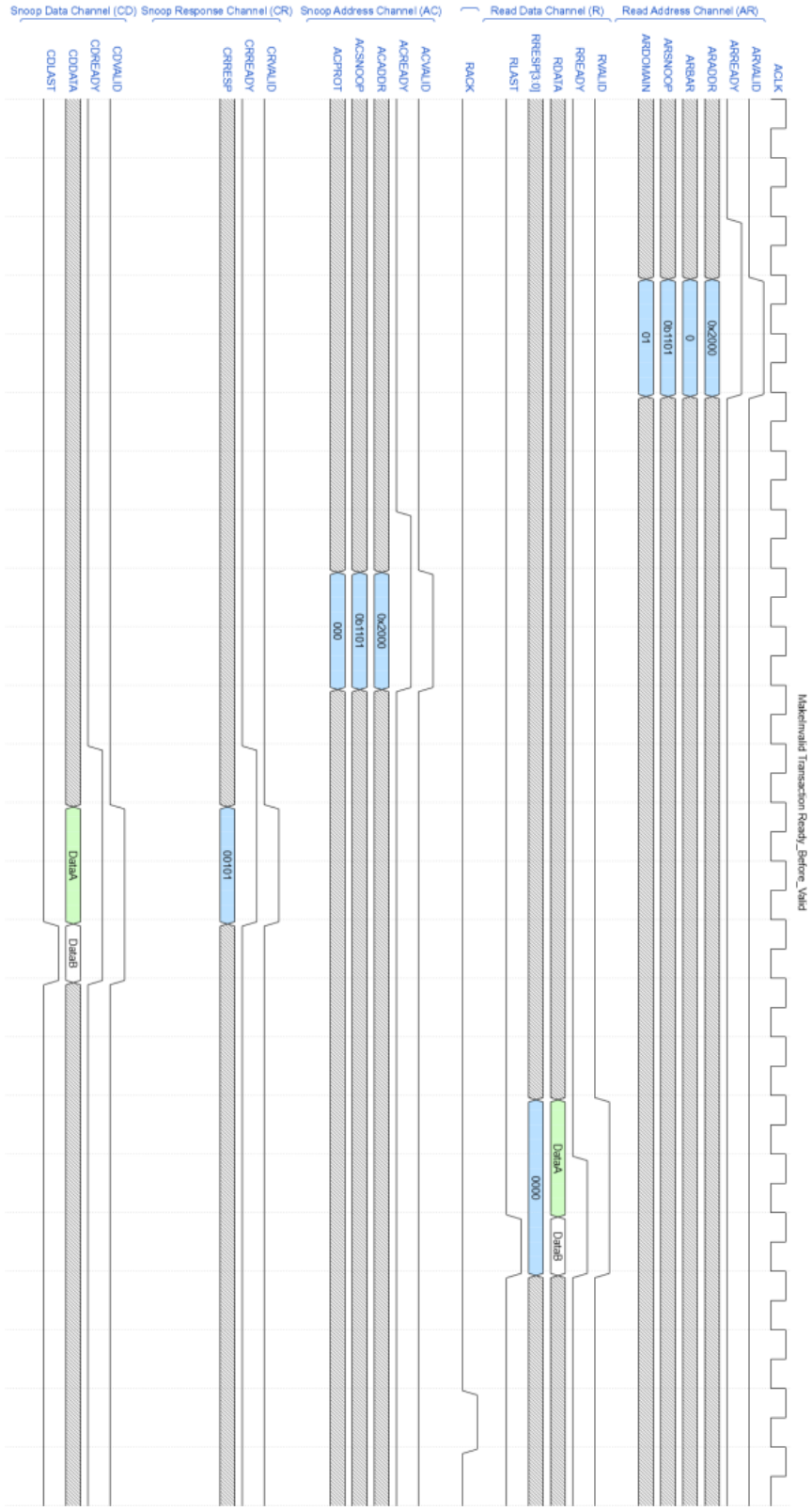
Modificări uzuale ale stării liniei cache pentru MakeInvalid

Stare Initială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
Invalid(I)	Invalid(I)	00

Alte modificări permise ale stării liniei cache pentru MakeInvalid

Stare Initială Cache	Stare Cache Finală	PassDirty, CRRESP[2]	IsShared, CRRESP[3]
Invalid(I)	Invalid(I)	0	0
UniqueClean(UC)	Invalid(I)	0	0
UniqueDirty(UD)	Invalid(I)	0	0
UniqueDirty(UD)	Invalid(I)	1	0
SharedClean(SC)	Invalid(I)	0	0
SharedDirty(SC)	Invalid(I)	0	0
SharedDirty(SC)	Invalid(I)	1	0

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă pentru tranzacția MakeInvalid



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                              },
      {name: 'ARREADY' , wave: '0.1..0.................
'                              },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x2000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0b1101
']           },  // MakeInvalid
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']               },  // Inner Shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............7.2x...', data: ['DataA',
'DataB'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['0000
']           },  // IsShared(0) + PassDirty(0) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0..................10
'                                  },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
```

```
'                               },
      {name: 'ACREADY', wave: '0......1..0.............
'                               },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000
']            },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1101
']            },  //MakeInvalid
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']               },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                               },
      {name: 'CRREADY', wave: '0..........1..0.........
'                               },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']             },  // WasUnique(0) + IsShared(0) + PassDirty(1) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},

    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                               },
      {name: 'CDREADY', wave: '0..........1...0........
'                               },
      {name: 'CDDATA' , wave: 'x...........7.2x........', data: ['DataA', 'Da
taB']    },
      {name: 'CDLAST' , wave: '0............10........
'                               },

{
```

```
}
    ],

{
},
  ],

  "head": {"text": "MakeInvalid Transaction Ready_Before_Valid
"                        },
  "config": {"hscale":
1                                                          }
}
```


5. Tranzacția MakeUnique

Un solicitant folosește o tranzacție MakeUnique atunci când stochează o linie cache completă. Tranzacția MakeUnique invalidează toate celelalte copii ale respectivei linii cache. O tranzacție MakeUnique este folosită într-o regiune de memorie care poate fi partajată cu alte sub-sisteme master.

Tranzacția MakeUnique asigură că:

- Linia cache poate fi menținută în starea Unique. Aceasta permite master-ului să efectueze o operațiune de stocare în linia cache, dar tranzacția nu furnizează o copie a datelor și pentru master.
- Toate celelalte copii ale liniei cache sunt șterse. Tranzacția MakeUnique trebuie inițiată de acel master care efectuează o operațiune completă de stocare a liniei de cache.

În multe implementări, sistemul poate opta să folosească MakeInvalid în loc de MakeUnique pentru a simplifica gestiunea coerenței. Acest lucru asigură că niciun alt master nu păstrează o copie, ceea ce garantează accesul exclusiv pentru masterul solicitant.

Cerințele de răspuns la tranzacție sunt:

- Răspunsul IsShared trebuie dezactivat (resetat) pentru a indica faptul că linia de cache este unică.
- Răspunsul PassDirty trebuie dezactivat (resetat).

Modificările de stare a liniei cache pentru o tranzacție MakeUnique sunt diferite față de toate celelalte tranzacții întrucât o tranzacție MakeUnique trebuie să fie cuplată cu o operațiune de stocare completă a liniei de cache.

Desfășurarea tranzacției

1. Inițiere – adresarea de citire – canal Read Address (AR)

    - Solicitantul inițiază o tranzacție de citire cu:
        - ARSNOOP = MakeUnique (0b1100)
        - ARADDR = 0x2000
        - ARVALID = 1

2. Interfața este deja pregătită (ARREADY = 1) înainte ca ARVALID să fie activat (setat), permițând ca semnalizările (”hand-shaking”) să înceapă imediat.

3. Interfața trimite o cerere de supraveghere (”snoop”) pe canalul AC:

        - ACSNOOP = MakeInvalid (0b1101)
        - ACADDR = 0x2000
        - ACVALID = 1

- Interfața este pregătită (ACREADY = 1) înainte de ACVALID.

- Cache supravegheat (”snooped”) răspunde cu:

    - CRRESP = 00101:
        - WasUnique = 0
        - IsShared = 0
        - PassDirty = 1
        - Error = 0
        - DataTransfer = 1 → Indică SharedClean
    - CRVALID = 1, CRREADY = 1

- Cache supravegheat (”snooped”) returnează date ”murdare” (alterate) pe canalul CD:
    - CDDATA = DataA, DataB
    - CDVALID = 1, CDREADY = 1
    - CDLAST = 1 la terminarea datelor.

4. Citirea datelor – pe canalul (R)

- Interfața returnează datele ”curățate” solicitantului:
- RDATA = DataA, DataB
- RRESP = 0000:
    - IsShared = 0
    - PassDirty = 0
    - OKAY = 00 → Indică UniqueClean

- RVALID = 1, RREADY = 1
- RLAST = 1 la ultimul ciclu de citire a datelor.

5. Încheierea
    - Solicitantul confirmă finalizarea citirii:
        - RACK = 1

Modificări uzuale ale stării liniei cache pentru MakeUnique

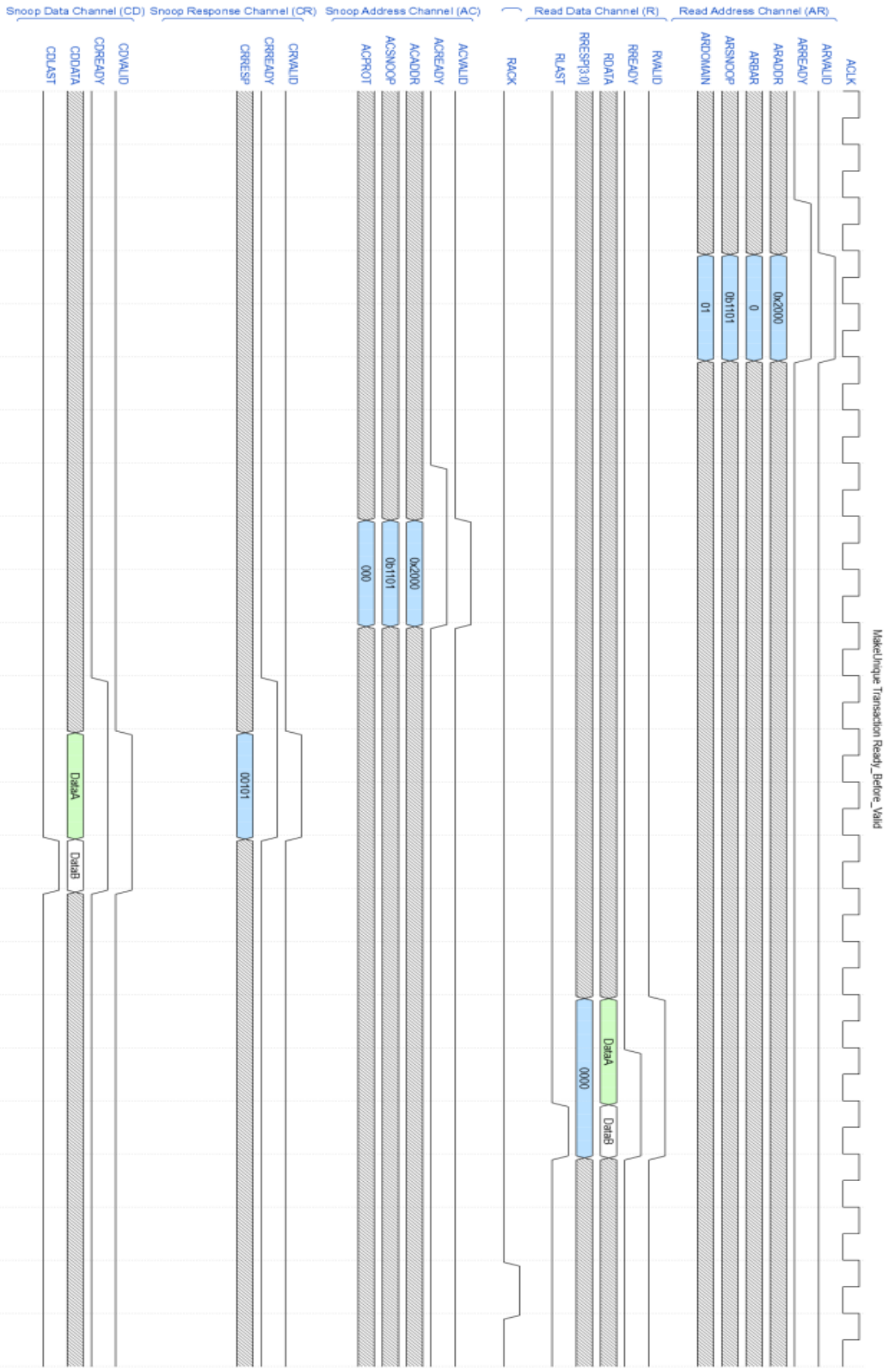
Stare Initială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
Invalid(I)	UniqueDirty(UD)	00
SharedClean(SC)	UniqueDirty(UD)	00
SharedDirty(SD)	UniqueDirty(UD)	00

Alte modificări permise ale stării liniei cache pentru MakeUnique

Stare Initială Cache	Stare Cache Finală	RRESP[3:2] IsShared/PassDirty
UniqueClean(UC)	UniqueClean(UC)	00
UniqueDirty(UD)	UniqueDirty(UD)	00

Din punct de vedere al răspunsului în regim de supraveghere (”Snoop”) această tranzacție este văzută la fel ca MakeInvalid, utilizând codificarea `0b1101` pentru `ACSNOOP`. Așa cum se poate vedea și în exemplul de mai jos, formele de undă sunt la fel ca la tranzacția MakeInvalid pe canalul Read Address și în ce privește codurile `ACSNOOP` și `ARDOMAIN`.

Stilul Ready_Before_Valid (”Finalizare înainte de validare”) – RBV – forme de undă pentru tranzacția MakeUnique



```
{
  "signal": [
    {name: 'ACLK', wave: 'p.....................
'                                  },

    ["Read Address Channel (AR)",
      {name: 'ARVALID' , wave: '0..1.0.................
'                              },
      {name: 'ARREADY' , wave: '0.1..0.................
'                              },
      {name: 'ARADDR'  , wave: 'x..5.x.................', data: ['0x2000
']           },
      {name: 'ARBAR'   , wave: 'x..5.x.................', data: ['0
']                },
      {name: 'ARSNOOP' , wave: 'x..5.x.................', data: ['0b1100
']           },  //MakeUnique
      {name: 'ARDOMAIN', wave: 'x..5.x.................', data: ['01
']               },  //Inner Shareable

{
}
    ],

    ["Read Data Channel (R)",
      {name: 'RVALID'    , wave: '0.............1..0...
'                             },
      {name: 'RREADY'    , wave: '0.............1.0...
'                             },
      {name: 'RDATA'     , wave: 'x.............7.2x...', data: ['DataA',
'DataB'] },
      {name: 'RRESP[3:0]', wave: 'x.............5..x...', data: ['0000
']           },  // IsShared(0) + PassDirty(0) + OKAY(00) - UniqueClean
      {name: 'RLAST', wave: '0................10...
'                                },

{
}
    ],

    [
      {name: 'RACK', wave: '0..................10
'                                 },
    ],

{
},

    ["Snoop Address Channel (AC)",
      {name: 'ACVALID', wave: '0.......1.0.............
```

```
'                                },
      {name: 'ACREADY', wave: '0......1..0.............
'                                },
      {name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000
']             },
      {name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1101
']             },  //MakeInvalid
      {name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000
']                },

{
}
    ],

{
},

    ["Snoop Response Channel (CR)",
      {name: 'CRVALID', wave: '0...........1.0.........
'                                },
      {name: 'CRREADY', wave: '0..........1..0.........
'                                },
      {name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00101
']              },  // WasUnique(0) + IsShared(0) + PassDirty(1) + Error(0) +
DataTransfer(1) - SharedClean

{
},

{
},

{
}
    ],

{
},
    ["Snoop Data Channel (CD)",
      {name: 'CDVALID', wave: '0...........1..0........
'                                },
      {name: 'CDREADY', wave: '0..........1...0........
'                                },
      {name: 'CDDATA' , wave: 'x...........7.2x........', data: ['DataA', 'Da
taB']    },
      {name: 'CDLAST' , wave: '0............10........
'                                },

{
}
```

```
    ],

{
},
  ],

  "head": {"text": "MakeUnique Transaction Ready_Before_Valid
"                        },
  "config": {"hscale":
1                                                          }
}
```

Fluxul tranzacției:

```
{ signal: [
{name: 'ACLK', wave: 'p.....................'                        },
["Write Address Channel (AW)",
{                                                                    },
{name: 'AWVALID' , wave: '0..1.0................'                     },
{name: 'AWREADY' , wave: '0.1..0................'                     },
{name: 'AWADDR'  , wave: 'x..5.x................', data: ['0x2000'] },
{name: 'AWBAR'   , wave: 'x..5.x................', data: ['0']      },
{name: 'AWSNOOP' , wave: 'x..5.x................', data: ['0100']   }, //
WriteLineUnique
{name: 'AWDOMAIN', wave: 'x..5.x................', data: ['01']     }, //
Inner Shareable {          },
{                                                                    }
],
["Write Data Channel (W)",
{                                                                    },
{name: 'WVALID' , wave: '0......1.0............'                     },
{name: 'WREADY' , wave: '0.....1..0............'                     },
{name: 'WDATA'  , wave: 'x......5.x............', data: ['Data1']   }, //
Only one beat sent
{name: 'WLAST'  , wave: '0........10...........'                     }, //
Premature end
{                                                                    },
{                                                                    } ],
["Write Response Channel (B)", {                                     },
{name: 'BVALID' , wave: '0..............1.0.....'                    },
{name: 'BREADY' , wave: '0.............1..0.....'                    },
{name: 'BRESP'  , wave: 'x..............5.x.....', data: ['00']      }, //
OKAY
{                                                                    },
{                                                                    } ],
["Snoop Address Channel (AC)",
{                                                                    },
{name: 'ACVALID', wave: '0.......1.0............'                     },
```

```
{name: 'ACREADY', wave: '0......1..0.............'                    },
{name: 'ACADDR' , wave: 'x.......5.x.............', data: ['0x2000']   },
{name: 'ACSNOOP', wave: 'x.......5.x.............', data: ['0b1101']   }, //
MakeInvalid
{name: 'ACPROT' , wave: 'x.......5.x.............', data: ['000']      },
{                                                                       },
{                                                                       } ],
["Snoop Response Channel (CR)", {                                       },
{name: 'CRVALID', wave: '0...........1.0.........'                      },
{name: 'CRREADY', wave: '0...........1..0.........'                     },
{name: 'CRRESP' , wave: 'x...........5.x.........', data: ['00000']     }, //
No indication of data issue
{                                                                       },
{                                                                       } ] ],
"head": {"text": "Error: WriteLineUnique with Incomplete Data Transfer"},
"config": {"hscale": 1                                                  }
}
```

Exercițiu:

Să se scrie rutina de tratare a erorii ReadClean în cazul următoarelor încălcări ale protocolului ACE:

- ARSNOOP primește valoarea incorectă 0000 (ReadNoSnoop) în loc de 0011 (ReadClean).
- ARDOMAIN primește valoarea incorectă 00 (Non-shareable), ceea ce e incompatibil cu ReadClean.
- ARADDR se modifică înainte de activarea (setarea) ARREADY.
- ARVALID este dezactivat (resetat) înainte de ARREADY, întrerupând semnalizarea ”handshake”.
- Nu e declanșată niciuna din semnalizările pe canale ”Snoop” (AC, CR, CD), ceea ce ar fi fost obligatoriu pentru ReadClean.

```
{ signal: [
{name: 'ACLK', wave: 'p.............'                                  },
["Read Address Channel (AR)",
{name: 'ARVALID' , wave: '01.0.........'                                }, //  D
easserted early
{name: 'ARREADY' , wave: '0..10.........'                               },
{name: 'ARADDR'  , wave: 'x.5x.........', data: ['0x1000', '0x2000'] }, //  C
```

```
hanged mid-handshake
{name: 'ARBAR'   , wave: 'x.5x.............', data: ['0']                },
{name: 'ARSNOOP' , wave: 'x.5x.............', data: ['0000']             }, //  S
hould be 0011
{name: 'ARDOMAIN', wave: 'x.5x.............', data: ['00']               }, //  S
hould be 01 or 10
{                                                                         }
],
["Read Data Channel (R)",
{name: 'RVALID'    , wave: '0.........1..0..'                             },
{name: 'RREADY'    , wave: '0..........1.0..'                             },
{name: 'RDATA'     , wave: 'x.........5.3x..', data: ['Data1', 'Data2'] },
{name: 'RRESP[3:0]', wave: 'x.........5..x..', data: ['0000']           }, //  N
ot SharedClean
{name: 'RLAST'     , wave: '0..........10..'                              },
{                                                                         }
],
[
{name: 'RACK', wave: '0............10'                                    },
],
{                                                                         },
["Snoop Address Channel (AC)",
{name: 'ACVALID', wave: '0.............'                                  }, //  M
issing
{name: 'ACREADY', wave: '0.............'                                  },
{name: 'ACADDR' , wave: 'x.............'                                  },
{name: 'ACSNOOP', wave: 'x.............'                                  },
{name: 'ACPROT' , wave: 'x.............'                                  },
{                                                                         } ],
{                                                                         },
["Snoop Response Channel (CR)",
{name: 'CRVALID', wave: '0.............'                                  }, //  M
issing
{name: 'CRREADY', wave: '0.............'                                  },
{name: 'CRRESP' , wave: 'x.............'                                  },
{                                                                         },
{                                                                         },
{                                                                         } ],
{                                                                         },
["Snoop Data Channel (CD)",
{name: 'CDVALID', wave: '0.............'                                  }, //  M
issing
{name: 'CDREADY', wave: '0.............'                                  },
{name: 'CDDATA' , wave: 'x.............'                                  },
{name: 'CDLAST' , wave: '0.............'                                  },
{                                                                         } ],
{                                                                         },
],
"head": {"text": "Severely Incorrect ReadClean Transaction"               },
config: {hscale: 1                                                        }
                                                                          }
```

```
{ signal: [
{name: 'ACLK', wave: 'p.....................'                          },
["Write Address Channel (AW)",
{                                                                      },
{name: 'AWVALID' , wave: '0..1.0................'                      },
{name: 'AWREADY' , wave: '0.1..0................'                      },
{name: 'AWADDR'  , wave: 'x..5.x................', data: ['0x2000']  },
{name: 'AWBAR'   , wave: 'x..5.x................', data: ['0']       },
{name: 'AWSNOOP' , wave: 'x..5.x................', data: ['0100']    }, // Writ
eLineUnique
{name: 'AWDOMAIN', wave: 'x..5.x................', data: ['01']      }, // Inne
r Shareable {           },
{                                                                      }
],
["Write Data Channel (W)",
{                                                                      },
{name: 'WVALID' , wave: '0......1..0............'                      },
{name: 'WREADY' , wave: '0.....1...0............'                      },
{name: 'WDATA'  , wave: 'x......5.5x............', data: ['Data1', 'Data2
']    }, //  Only one beat sent
{name: 'WLAST'  , wave: '0.......10.......10.....'                     }, //  Pre
mature end
{                                                                      },
{                                                                      } ],
["Write Response Channel (B)", {                                       },
{name: 'BVALID' , wave: '0..............1.0.....'                      },
{name: 'BREADY' , wave: '0.............1..0.....'                      },
{name: 'BRESP'  , wave: 'x..............5.x.....', data: ['00']       }, // OKAY
{                                                                      },
{                                                                      } ],
["Snoop Address Channel (AC)",
{                                                                      },
{name: 'ACVALID', wave: '0.......1.0............'                      },
{name: 'ACREADY', wave: '0......1..0............'                      },
{name: 'ACADDR' , wave: 'x.......5.x............', data: ['0x3000']   },
{name: 'ACSNOOP', wave: 'x.......5.x............', data: ['0b1101']   }, // Make
Invalid
{name: 'ACPROT' , wave: 'x.......5.x............', data: ['000']      },
{                                                                      },
{                                                                      } ],
["Snoop Response Channel (CR)", {                                      },
{name: 'CRVALID', wave: '0..........1.0........'                       },
{name: 'CRREADY', wave: '0.........1..0........'                       },
{name: 'CRRESP' , wave: 'x..........5.x........', data: ['00000']     }, //  No
indication of data issue
{                                                                      },
{                                                                      } ] ],
"head": {"text": "Error: WriteLineUnique with Incomplete Data Transfer"},
"config": {"hscale": 1                                                 }
}
```

5. Ready-Valid

Protocolul Ready-Valid (”Gata-Valid”) funcționează pe baza semnalului de ceas și a semnalelor de control [1, pag. 37].

Ready (”Gata”): Semnalizează că Furnizorul este pregătit să primească date.

Valid: Indică faptul că Solicitantul validează datele pentru o tranzacție.

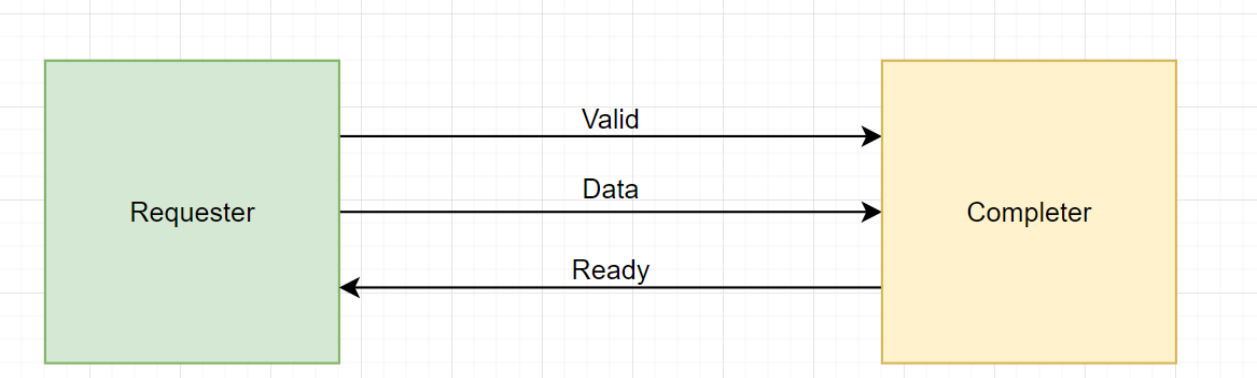
Data (”Date”): Aceasta este informația oferită pe un bus de date.

Aspecte esențiale:

[3, pag. 23]

1. Transferul de date are loc doar atunci când ambele semnale (Ready și `Valid`) sunt active (”1”) în același ciclu de ceas.
2. Protocolul este sincronizat cu ceasul – deci transferul datelor depinde de ciclurile de ceas.

Diagrama Solicitant - Furnizor:



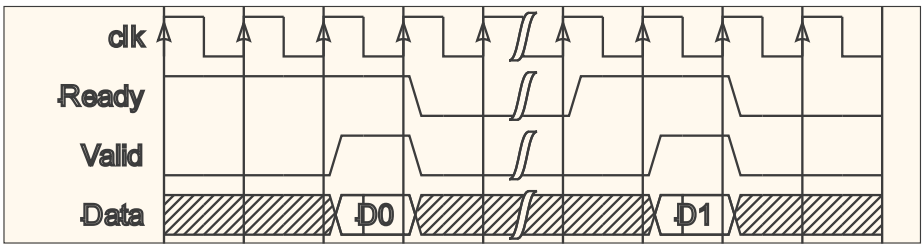
Parametrii interfeței Ready-Valid:

Parametru	Descriere
DATA_WIDTH	Numărul de biți de date transferați de la Solicitant la Furnizor când Valid și Ready sunt ”1”.

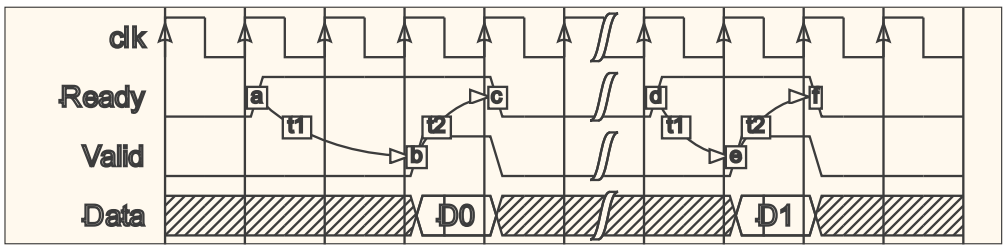
Descrierea porturilor:

Nume port	Descriere	Direcția (față de Solicitant)	Dimensiune (biți)
Valid	Modulul Solicitant trimite un bit "Valid". Are date de transferat.	Ieşire (O)	1
Ready	Modulul Furnizor trimite un bit "Ready" – poate accepta date.	Intrare (I)	1
Data	Datele transferate între Solicitant și Furnizor: stabile cât timp Valid = ”1”, eșantionate când Valid = ”1” și Ready = ”1” simultan.	Output (O)	DATA_WIDTH

Protocolul standard



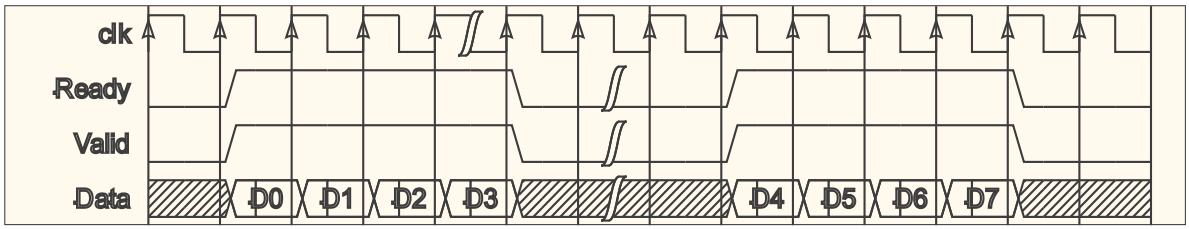

```
{signal: [
  {name: 'clk', wave: 'P...|....'},
  {name: 'Ready', wave: '1..0|1.0.'},
  {name: 'Valid', wave: '0.10|.10.'},
  {name: 'Data', wave: 'x.3x|.4x.', data: ['D0', 'D1']},
]}
```




```
{signal: [
  {name: 'clk', wave: 'P....|....'},
  {name: 'Ready', wave: '01..0|1.0.', node:'.a..c.d.f'},
```

```
  {name: 'Valid', wave: '0..10|.10.', node:'...b...e'},
  {name: 'Data', wave: 'x..3x|.4x.', data: ['D0', 'D1']},
],edge:[
  'a~->b t1',
  'b~->c t2',
  'd~->e t1',
  'e~->f t2',  ] }
```

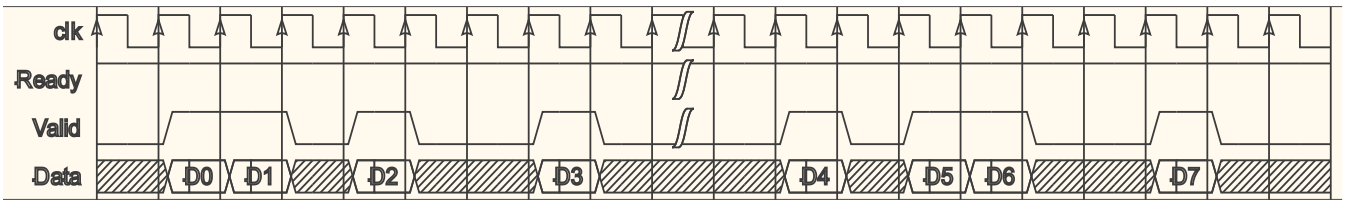
În cazul tranzacțiilor imediat consecutive *(b2b)*, Solicitantul menține semnalul Valid activ între tranzacții consecutive, iar Furnizorul menține semnalul Ready activ, permițând astfel un flux continuu de date, transferate de la un ciclu de ceas la următorul.




```
{signal: [
  {name: 'clk', wave: 'P...|.........'},
  {name: 'Ready', wave: '01...0|.1...0.'},
  {name: 'Valid', wave: '01...0|.1...0.'},
  {name: 'Data', wave: 'x3456x|x3456xx', data: ['D0', 'D1','D2', 'D3','D4', '
D5','D6', 'D7']},
]}
```

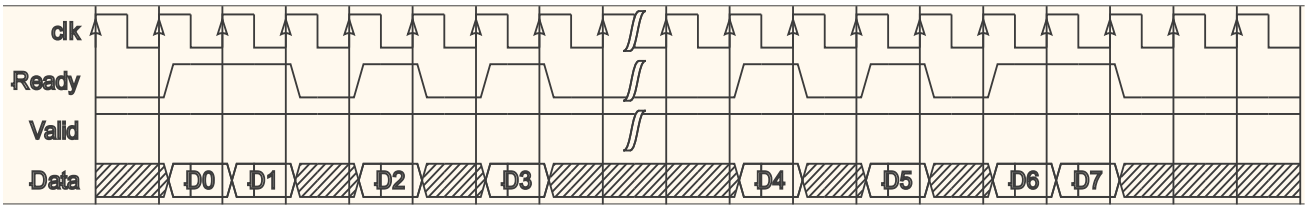
Alte tipuri de tranzacții folosind protocolul Ready-Valid:

- **Ready before Valid** *(Disponibil înainte de Valid)*: În acest caz, semnalul Ready este activat (setat) înainte ca semnalul Valid să fie activat (setat).



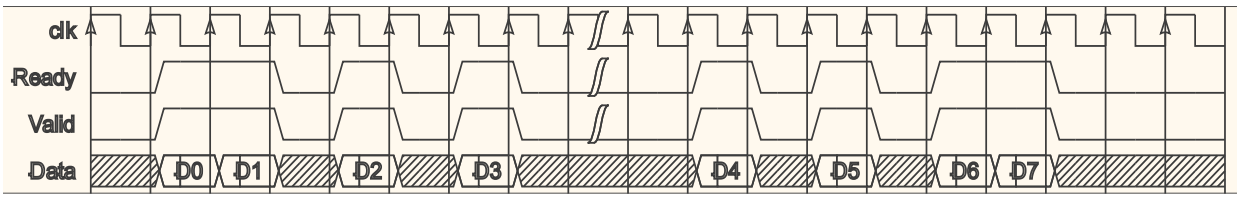

```
{signal: [
  {name: 'clk', wave: 'P........|.........'},
  {name: 'Ready', wave: '1........|.........'},
  {name: 'Valid', wave: '01.010.10|.101.0.10.'},
  {name: 'Data', wave: 'x34x5xx6x..3x45x.6x.', data: ['D0','D1','D2','D3','D4
','D5','D6','D7']},
]}
```

- **Valid before Ready** *(Valid înainte de Disponibil):* În acest caz, semnalul Valid este activat (setat) înainte ca semnalul Ready să fie activat (setat). Solicitantul indică faptul că are date gata de trimis înainte ca Furnizorul să fie pregătit să le primească.



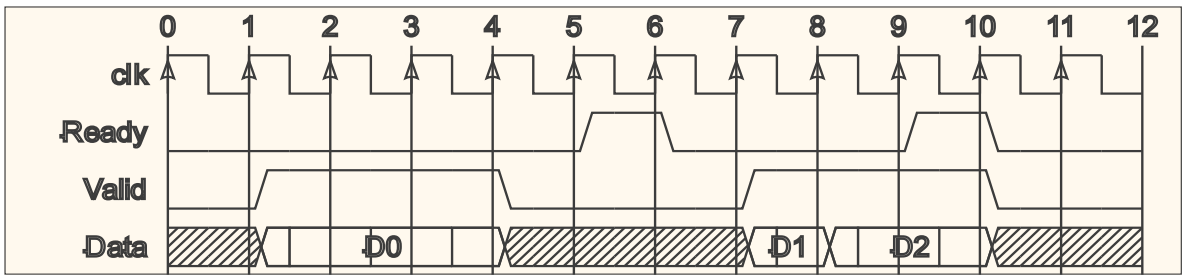

```
{signal: [
  {name: 'clk', wave: 'P.......|.........'},
  {name: 'Ready', wave: '01.01010|.10101.0..'},
  {name: 'Valid', wave: '1.......|.........'},
  {name: 'Data', wave: 'x34x5x6x..3x4x56x..', data: ['D0','D1','D2','D3','D4
','D5','D6','D7']},
]}
```

- **Ready and Valid** *(Disponibil și Valid):* Transferul de date are loc atunci când ambele semnale, Ready și Valid, sunt activate simultan în același ciclu de ceas.




```
{signal: [
  {name: 'clk', wave: 'P.......|.........'},
  {name: 'Ready', wave: '01.01010|.10101.0..'},
  {name: 'Valid', wave: '01.01010|.10101.0..'},
  {name: 'Data', wave: 'x34x5x6x..3x4x56x..', data: ['D0','D1','D2','D3','D4
','D5','D6','D7']},
]}
```

Dacă Furnizorul nu a finalizat o tranzacție și Solicitantul modifică datele – adică inițiază o nouă tranzacție înainte ca cea anterioară să fie finalizată – se consideră că protocolul este încălcat.




```
{signal: [
  {name: 'clk', wave: 'P..........'},
  {name: 'Ready', wave: '0....10..10.'},
  {name: 'Valid', wave: '01..0..1..0.'},
  {name: 'Data', wave: 'x9..x..92.x.', data: ['D0', 'D1', 'D2']},
  ],
  config: { hscale: 1 },
  head:{
  tick:0,
  every:1  },}
```

Prima eroare prezentată aici începe în primul ciclu:

1. În ciclul 1, Solicitantul inițiază o cerere simultan cu datele D0, dar cererea nu este satisfăcută. La ciclul 4, fără ca datele să fie preluate de Furnizor (Ready = ”1”), acest lucru cauzează o încălcare a protocolului.

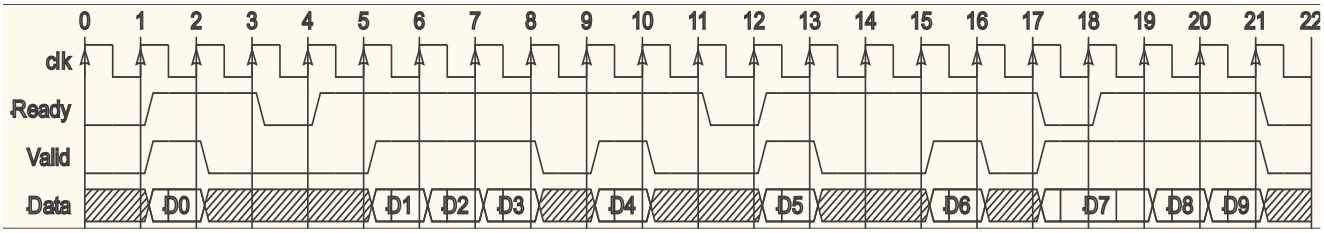
2. Mai mult, în ciclul 5, Furnizorul este "pregătit" (Ready) pentru o tranzacție, dar în următorul ciclu de ceas nu mai este disponibil pentru a primi date, ceea ce nu este considerat o eroare de protocol.

Al doilea tip de eroare:

3. În ciclul 7, Solicitantul inițiază o cerere "Validă" cu datele D1 dar, fără ca această cerere să fie servită, Solicitantul schimbă datele de la D1 la D2. Aceasta este considerată din nou o eroare, deoarece modulul Solicitant a modificat datele fără ca modulul Furnizor să deservească tranzacția anterioară.

În concluzie, pe parcursul unei tranzacții, datele trebuie să rămână stabile până la finalizarea tranzacției sau orice tranzacție trebuie deservită înainte ca o cerere nouă să fie inițiată.

Alte tipuri de tranzacții:




```
{signal: [
  {name: 'clk', wave: 'P.....................'},
  {name: 'Ready', wave: '01.01......01....01..0'},
  {name: 'Valid', wave: '010..1..010.10.101...0'},
  {name: 'Data', wave: 'x3x..456x3x.5x.6x3.45x', data: ['D0','D1','D2','D3','
D4','D5','D6','D7','D8','D9']},
],
config: { hscale: 1 },
```

```
head:{
tick:0,
every:1  },}
```

6. Request-Acknowledge (Cerere-Confirmare) cu 2 faze

Aspecte esențiale:

[8, pag. 1]

1. Protocol de comunicație lent;
2. Protocolul acceptă tranzacții imediat consecutive *(b2b)*, dar perioada minimă a tranzacțiilor este una la fiecare două cicluri de ceas succesive;
3. Protocol sincron [2, pag. 153] ;
4. Pentru un transfer de date este necesar ca ambele semnale – cerere și confirmare – să fie active [3, pag. 66].

Interfața

1. Parametri

Nume parametru	Descriere
DATA_WIDTH	Numărul de biți de date.

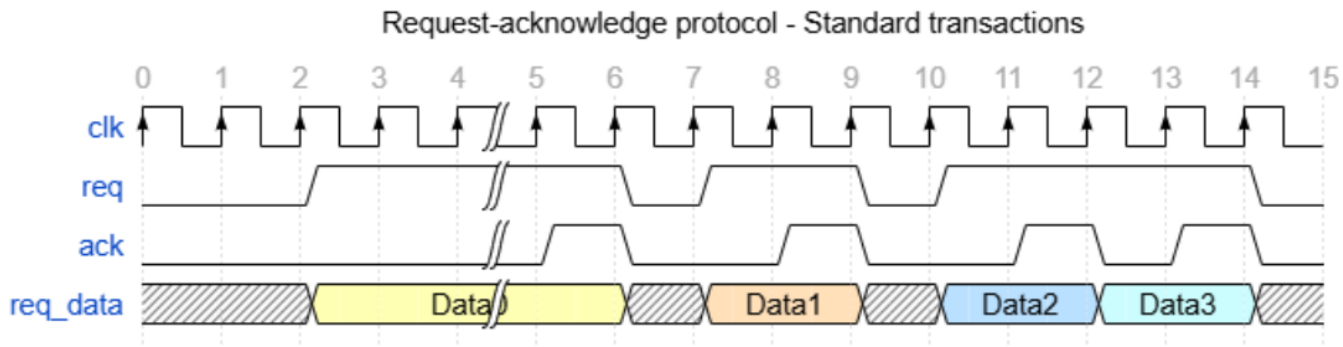
2. Descrierea porturilor:

Nume	Dimensiune	Direcția (față de Solicitant)	Descriere
req	1	O	Semnal de Cerere.
ack	1	I	Semnal de Confirmare.
req_data	DATA_WIDTH	O	Date transmise.
ack_data	DATA_WIDTH	I	Date confirmate - opțional.

Diagramă bloc (Solicitant – Furnizor)



Forme de undă




```
{
  signal: [
    {name: 'clk', wave: 'P...|.........'},
    {name: 'req', wave: '0.1.|.01.01...0'},
    {name: 'ack', wave: '0...|10.10.1010'},
    {name: 'req_data', wave: 'x.3.|.x4.x5.6.x', data: ['Data0', 'Data1', 'Dat
a2', 'Data3']},
      {}, ],
  config: { hscale: 1 },
  head:{
   text :'Request-acknowledge protocol - Standard transactions',
   tick :0,
   every:1},
  foot:{ text:'Figure 1'},
}
```

Cele trei tranzacții standard ale protocolului Request-Acknowledge reprezentate în figura de mai sus se desfășoară în felul următor:

- La al doilea ciclu de ceas, Solicitantul activează semnalul de Cerere (req=”1”) simultan cu semnalul de date (req_data = Data0).

Aceste semnale rămân stabile până când e recepționat semnalul de Confirmare (ack = ”1”).

Furnizorul dezactivează (resetează) semnalul de Confirmare după un ciclu de ceas de la activare;

- La al 7-lea ciclu de ceas, Solicitantul activează semnalul de Cerere (req=”1”) simultan cu semnalul de date (req_data = Data1).

De data aceasta, Furnizorul detectează Cererea la ciclul 8 și activează semnalul de Confirmare (ack = ”1”).

Solicitantul primește Confirmarea la două cicluri de ceas după ce cererea este formulată;

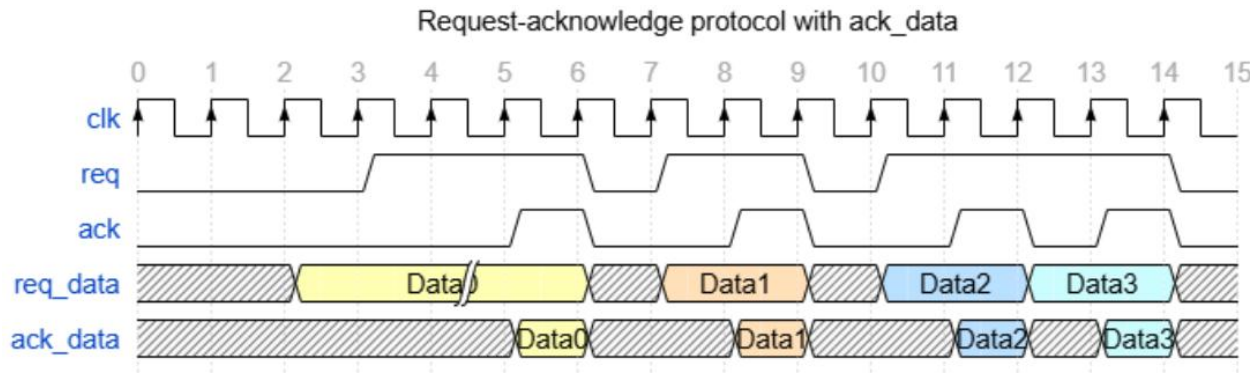
- În cazul unei tranzacții imediat consecutive *(b2b)*, Cererea este formulată similar cu cazurile anterioare.

- La ciclul 12, Solicitantul detectează Confirmarea și decide să inițieze o nouă Cerere, menținând semnalul de Cerere activ în timp ce modifică datele.

Ambele Cereri sunt Confirmate prin activarea (setarea) semnalului ack.

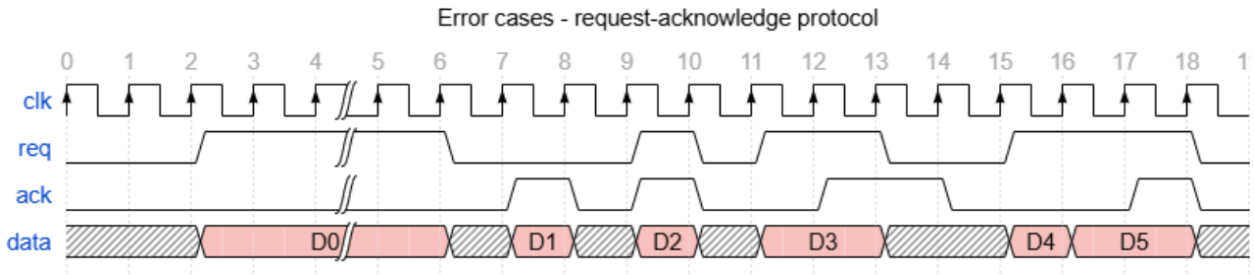
Protocolul Request-Acknowledge poate funcționa și cu ack_data, așa cum este exemplificat în figura de mai jos.

Tranzacțiile urmează același curs precum cele din figura de mai sus, cu excepția faptului că atunci când Solicitantul primește Confirmarea, aceasta este însoțită de ack_data.




```
{
  signal: [
    {name: 'clk', wave: 'P.............'},
    {name: 'req', wave: '0..1..01.01...0'},
    {name: 'ack', wave: '0....10.10.1010'},
    {name: 'req_data', wave: 'x.3.|.x4.x5.6.x', data: ['Data0', 'Data1', 'Dat
a2', 'Data3']},
    {name: 'ack_data', wave: 'x....3x.4x.5x6x', data: ['Data0', 'Data1', 'Dat
a2', 'Data3']},
      {}, ],
  config: { hscale: 1 },
  head:{
   text:'Request-acknowledge protocol with ack_data',
   tick:0,
   every:1 },
 foot:{text:'Figure 2'},
}
```

Cazuri de eroare



```
{
  signal: [
    {name: 'clk', wave: 'P...|.............'},
    {name: 'req', wave: '0.1.|.0..101.0.1..0'},
    {name: 'ack', wave: '0...|..1010.1.0..10'},
    {name: 'data', wave: 'x.9.|.x9x9x9.x.99.x', data: ['D0', 'D1', 'D2', 'D3
', 'D4', 'D5']},
      {}, ],
    config: { hscale: 1 },
    head:{
     text :'Error cases - request-acknowledge protocol',
     tick : 0,
     every: 1},
   foot:{ text:'Figure 3'},
}
```

Figura ilustrează 5 cazuri de eroare care pot apărea în cadrul protocolului Request-Acknowledge cu 2 faze.

- La al doilea ciclu de ceas, Solicitantul activează semnalul de Cerere (req=”1”), dar nu primește o Confirmare de la Furnizor; Cererea este abandonată la ciclul 6, cu semnalarea unei erori de protocol.
- La ciclul 7, Furnizorul activează semnalul de Confirmare (ack = ”1”) fără să existe o Cerere, ceea ce duce la o eroare de protocol.
- La ciclul 9, Cererea este activată de Solicitant (req = ”1”); simultan, Furnizorul activează semnalul de Confirmare. Aceasta constituie o eroare de protocol, deoarece la momentul în care semnalul de Confirmare este activat (ciclul 9), Furnizorul nu a primit încă Cererea.
- Cererea inițiată la ciclul 11 este formulată corect, dar Furnizorul nu dezactivează (resetează) Confirmarea la un ciclu de ceas după activarea acesteia, rezultând o eroare de protocol asemănătoare celei din al doilea caz (Confirmarea unei tranzacții fără existența unei Cereri).
- La ciclul 15, Cererea este activată, împreună cu datele; totuși, apare o eroare de protocol deoarece datele se schimbă înainte de primirea Confirmării.

7. Request-Acknowledge (Cerere-Confirmare) cu 4 faze

Aspecte esențiale:

[8, pag. 1]

- Protocol de comunicație lent;
- Protocol sincron cu transfer de date controlat;
- Protocolul acceptă tranzacții imediat consecutive (b2b), dar perioada minimă a tranzacțiilor este una la fiecare două cicluri de ceas succesive;
- Pentru un transfer de date este necesar ca ambele semnale – cerere și confirmare – să fie active.
- Spre deosebire de protocolul cu 2 faze, protocolul cu 4 faze necesită ca semnalele să revină la starea inițială (revenirea la zero – ” return-to-zero”) înainte ca următoarea tranzacție să poată începe.

Interfața

1. Parametri

Nume parametru	Descriere
DATA_WIDTH	Numărul de biți de date.

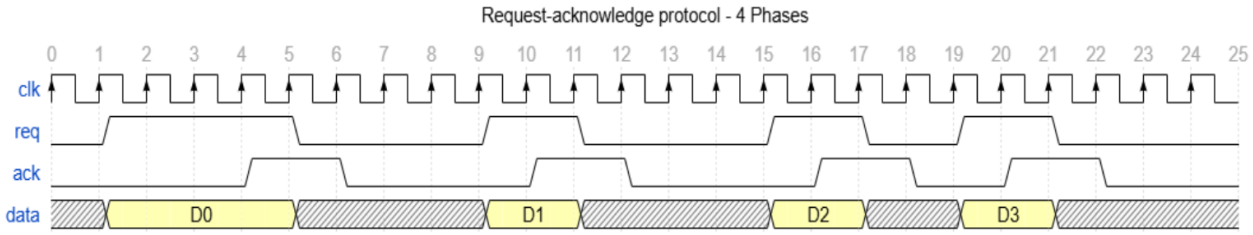
2. Descrierea porturilor:

Nume	Dimensiune	Direcția (față de Solicitant)	Descriere
req	1	O	Semnal de Cerere.
ack	1	I	Semnal de Confirmare.
data	DATA_WIDTH	O	Date transmise.

Diagramă bloc (Solicitant – Furnizor)



Forme de undă



Tranzacții standard

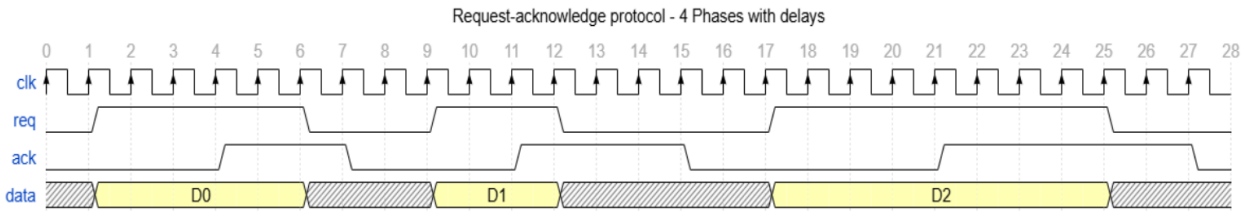
```
{
  signal: [
  { name: "clk", "wave": "P......................" },
  { name: "req", "wave": "01...0...1.0...1.0.1.0..." },
  { name: "ack", "wave": "0...1.0...1.0...1.0.1.0.." },
  { name: "data", "wave": "x3...x...3.x...3.x.3.x...", "data": ["D0", "D1", "
D2", "D3"]  } ],
    config: {"hscale": 1 },
    head:
    {text : "Request-acknowledge protocol - 4 Phases",
     tick : 0,
     every: 1  },
    foot:   { "text": "Figure 1" }
}
```

Cele trei tranzacții standard ale protocolului Request-Acknowledge reprezentate în figura de mai sus se desfășoară în felul următor:

- La ciclul 1 de ceas, Solicitantul activează semnalul de Cerere (req=”1”) simultan cu semnalul de date D0. Furnizorul activează semnalul de Confirmare (ack = ”1”) la ciclul 4. După faza revenirii la zero (” return-to-zero”) a protocolului, ambele semnale sunt dezactivate (req=”0”, ack=”0”) până la ciclul 6.
- La al 9-lea ciclu de ceas începe o tranzacție rapidă cu req=”1” și datele D1. Confirmarea urmează la ciclul 10, cu o încheiere cu revenire la zero (”return-to-zero”) până la ciclul 12.
- Tranzacțiile imediat consecutive *(b2b)* încep la ciclul 15 cu datele D2. După faza cu revenire la zero, un decalaj de numai un ciclu de ceas asigură temporizarea până ce a doua Cerere e inițiată la ciclul 19 cu datele D3.

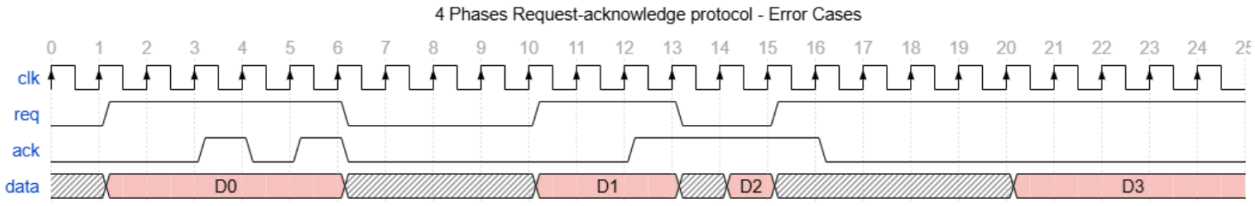
În cazul protocolului Request-Acknowledge în 4 faze, pot apărea întârzieri. Așa cum se poate vedea în figura de mai jos, nu este necesar ca semnalul de Cerere să fie dezactivat (resetat) imediat după primirea Confirmării. În plus, Confirmarea nu trebuie dezactivată (resetată) instantaneu după dezactivarea Cererii.

- De la ciclul 1 până la 8 – tranzacție cu Cerere care e dezactivată cu o întârziere după activarea Confirmării;
- De la ciclul 9 până la 16 – tranzacție cu Confirmare dezactivată cu întârziere;
- De la ciclul 17 până la 28 – tranzacție cu dezactivare întârziată atât pentru semnalul de Cerere cât și pentru cel de Confirmare.



```
{
  signal: [
    { name: "clk", "wave": "P........................." },
    { name: "req", "wave": "01....0..1..0....1.......0.."},
    { name: "ack","wave": "0...1..0...1...0.....1.....0" },
    { name: "data", "wave": "x3....x..3..x....3.......x..", "data": ["D0", "D
1", "D2", "D3"] } ],
  config: { "hscale": 1 },
  head  : {
    text : "Request-acknowledge protocol - 4 Phases with delays",
    tick : 0,
    every: 1 },
  foot  :   { "text": "Figure 2"}
}
```

Cazuri de eroare




```
{
  signal: [
    { name: "clk", "wave": "P......................." },
    { name: "req", "wave": "01....0...1..0.1........" },
    { name: "ack", "wave": "0..1010.....1...0........" },
    { name: "data", "wave": "x9....x...9..x9x....9....", data: ["D0", "D1", "
D2", "D3"]}],
  config: { "hscale": 1 },
  head  : {
    text : "4 Phases Request-acknowledge protocol - Error Cases",
    tick : 0  ,
    every: 1 },
  foot  : { "text": "Figure 3"}
}
```

Figura ilustrează 4 cazuri de eroare care pot apărea în cadrul protocolului Request-Acknowledge cu 4 faze.

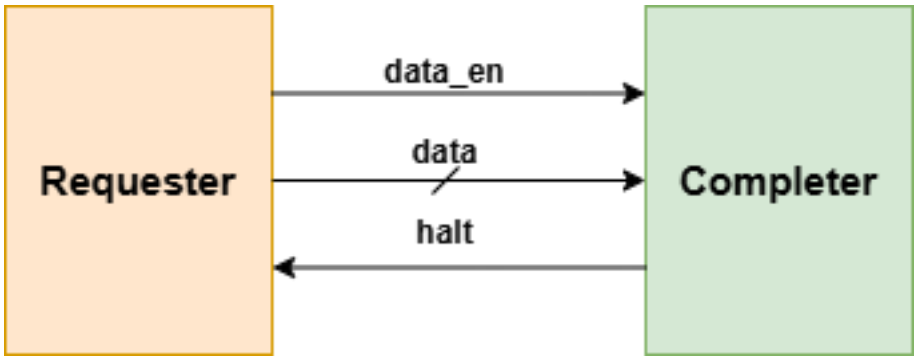
- În ciclul 1, Solicitantul activează semnalul de Cerere (req=”1”) cu datele D0, intrând în faza 1. Furnizorul încalcă protocolul prin bascularea de două ori a semnalului ack (ciclurile 3-6) timp în care semnalul de Cerere se menține activ. Această Confirmare instabilă alterează faza 2 a protocolului, în care ack ar fi trebuit să se mențină ”1” până la dezactivarea req, astfel că e împiedicată completarea corectă a fazelor 3 și 4.
- În ciclul 10, Solicitantul inițiază o nouă tranzacție cu datele D1 (faza 1). Furnizorul activează semnalul ack (faza 2), dar în timpul fazei de revenire la zero (”return-to-zero”) când ambele semnale ar trebui să fie ”0” (faza 4), Solicitantul activează o Cerere prematură în ciclul 15 când ack a rămas ”1”. Această suprapunere de semnalizări ”handshake” încalcă cerințele protocolului privitoare la separarea completă între tranzacții.
- Instabilitatea datelor survine în timpul acestei tranzacții cu suprapunere, cu schimbarea datelor de la D2 la D3 fără ca precedentele semnalizări ”handshake” să fi fost completate în toate cele 4 faze. S-a arătat astfel cum o încălcare de protocol (Cerere prematură) poate duce la erori (instabilitatea datelor).
- În fine, între ciclurile 16 și 25, după ce datele s-au modificat la D3, Furnizorul omite să dea o Confirmare a Cererii, lăsând protocolul blocat la faza 1. Această depășire a timpului nu permite completarea semnalizărilor ”handshake” și ar necesita mecanisme de revenire din eroare.

8. Data Stream (protocolul Flux de Date)

Descriere

Protocolul *Flux de Date* (”Data Stream”) este un protocol simplificat de comunicare folosit pentru transferul datelor între sub-sisteme [3, pag.57]. În acest protocol, tranzacția este inițiată de Solicitant, care activează (setează) semnalul data_en pentru a iniția transferul de date. Furnizorul recepționează semnalul și datele când capacitatea limitată a acestuia nu mai permite decât preluarea unui anumit număr de unități de date. Acest prag numeric este parametrizat și poate varia între diferite implementări [1, pag. 131].

Figura de mai jos ilustrează diagrama bloc a protocolului Flux de Date:



Nume	Direcție	Dimensiune	Descriere
data_en	Solicitant	1	Indică începutul transferului; Solicitantul transmite date doar când data_en este activat (”1”).
data	Solicitant	[DWITH-1:0]	Datele transferate
halt	Furnizor	1	Semnalează atingerea limitei de capacitate a Furnizorului, când nu se poate accepta mai mult de un număr parametrizat de unități de date (N).

Parametri

Acest protocol definește mai mulți parametri configurabili care influențează comportamentul și momentul tranzacțiilor:

DWIDTH: Specifică lățimea magistralei de date folosită pentru operațiunile de citire și scriere.

Determină câți biți de date pot fi transferați într-o singură tranzacție.

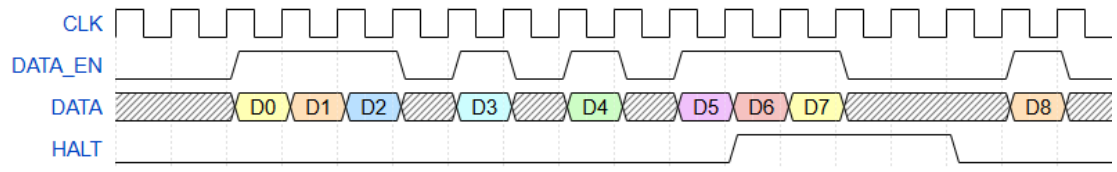
HWIDTH: Specifică numărul de unități de date care pot fi transferate după ce semnalul HALT este activat (setat), înainte ca stiva FIFO să se umple.

În exemplele de mai jos, parametrul HALT este setat la 2.

Forme de undă:

Notă: Pentru aceste exemple presupunem că Solicitantului i se permite să transmită două unități de date suplimentare după ce detectează că semnalul de oprire HALT a fost activat.

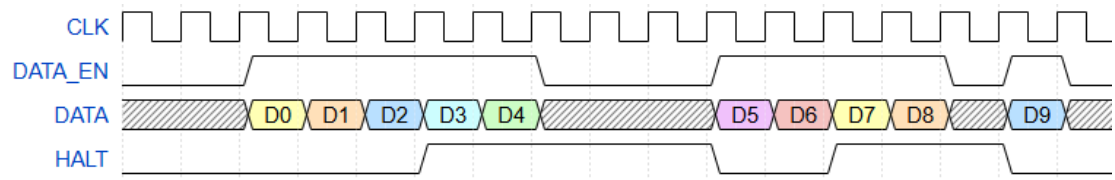
Figura de mai jos prezintă transferurile normale de date în protocolul Flux de Date




```
{signal: [
  {name: 'CLK', wave: 'p.................'},
  {name: 'DATA_EN', wave: '0.1..010101..0..10'},
  {name: 'DATA', wave: 'x.345x6x7x893x..4x', data: ['D0', 'D1','D2','D3','D4
','D5','D6','D7','D8','D9']},
  {name: 'HALT', wave  :'0.........1...0..'},
],

}
```

Figura de mai jos exemplifică funcționalitățile halt și data_en

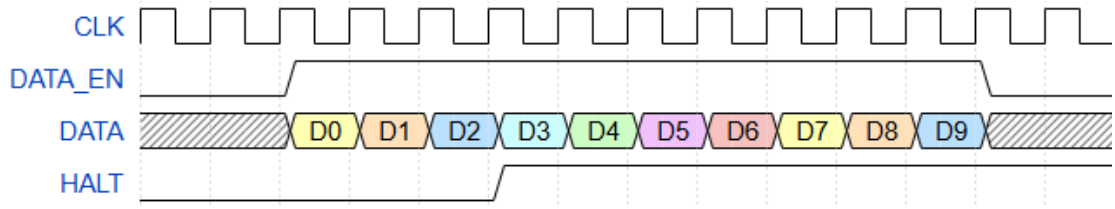



```
{signal: [
  {name: 'CLK', wave: 'p.............'},
  {name: 'DATA_EN', wave: '0.1....0..1...010'},
  {name: 'DATA', wave: 'x.34567x..8934x5x', data: ['D0', 'D1','D2','D3','D4',
'D5','D6','D7','D8','D9']},
  {name: 'HALT', wave  :'0....1....0.1..0.'},
],

}
```

Cazuri de eroare:

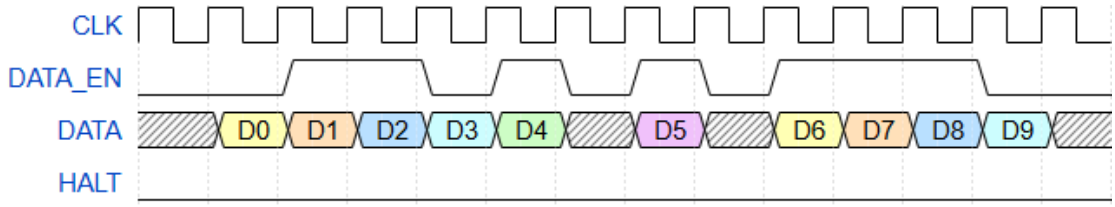
În figura de mai jos, Solicitantul nu încetează transmiterea datelor chiar și atunci când semnalul halt este activat




```
{signal: [
  {name: 'CLK', wave: 'p............'},
  {name: 'DATA_EN', wave: '0.1........0.'},
  {name: 'DATA', wave: 'x.3456789345x.', data: ['D0',
'D1','D2','D3','D4','D5','D6','D7','D8','D9']},
  {name: 'HALT', wave  :'0....1........'},
],

}
```

Figura de mai jos ilustrează transmiterea datelor fără data_en




```
{signal: [
  {name: 'CLK', wave: 'p.............'},
  {name: 'DATA_EN', wave: '0.1.010101..0.'},
  {name: 'DATA', wave: 'x34567x8x3456x', data: ['D0', 'D1','D2','D3','D4','D5
','D6','D7','D8','D9']},
  {name: 'HALT', wave  :'0.............'},
],
}
```

9. Protocolul de Interfață Imagine Pixel

Aspecte esențiale

1. Protocol sincron;
2. Permite tranzacții imediat consecutive *(b2b)* ;
3. Pixelul inițial dintr-un cadru trebuie transmis împreună cu semnalele de început de cadru (pix_sof) și de început de linie (pix_sol);
4. Ultimul pixel dintr-un cadru trebuie să fie însoțit de semnalele de sfârșit de cadru (pix_eof) și de sfârșit de linie (pix_eol);
5. Primul pixel din fiecare linie trebuie trimis cu semnalul de început al liniei (pix_sol);
6. Ultimul pixel din fiecare linie trebuie transmis alături de semnalul de la sfârșitul liniei (pix_eol);
7. Pixelii intermediari dintr-o linie trebuie trimiși doar împreună cu semnalul de valoare a pixelului (pix_val);
8. Pentru un transfer de date reușit, semnalele de sincronizare a pixelilor (pix_sof, pix_eof, pix_sol, pix_eol) trebuie activate corect, iar datele trebuie însoțite de pix_val semnal.

Interfața

1. Parametri

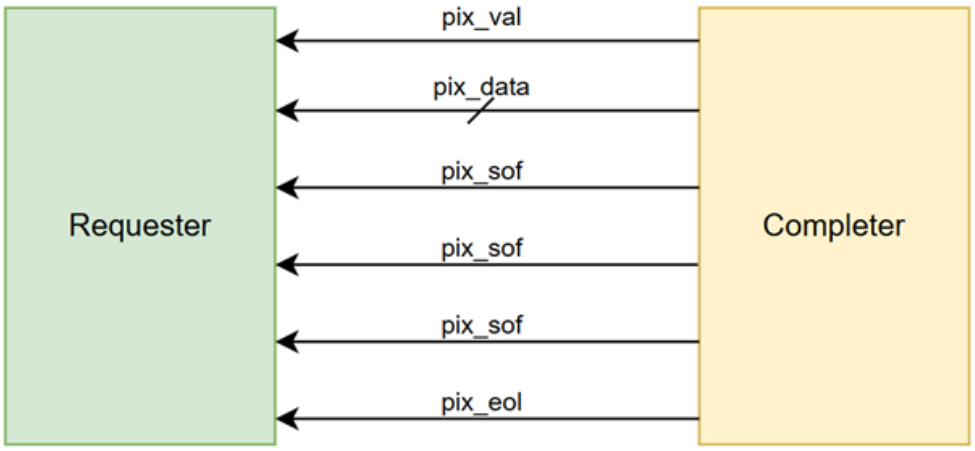
Nume parametru	Descriere
DATA_WIDTH	Numărul de biți de date.

2. Descrierea porturilor

Nume	Dimensiune	Direcția	Descriere
pix_val	1	I	Semnal pixel valid.
pix_rdy	1	O	Semnal pixel disponibil – utilizat în versiunea IPI cu răspuns Ready (”Disponibil”).

Nume	Dimensiune	Direcția	Descriere
pix_data	DATA_WIDTH - 1; PIXEL_PER_CYCLE - 1;	I	Valoare pixel.
pix_sof	1	I	Start cadru.
pix_eof	1	I	Sfârșit cadru.
pix_sol	1	I	Start linie.
pix_eol	1	I	Sfârșit linie.
HSYNC**	1	I	Sincronizare orizontală.
VSYNC**	1	I	Sincronizare verticală.

Diagramă bloc (Solicitant – Furnizor)



Forme de undă

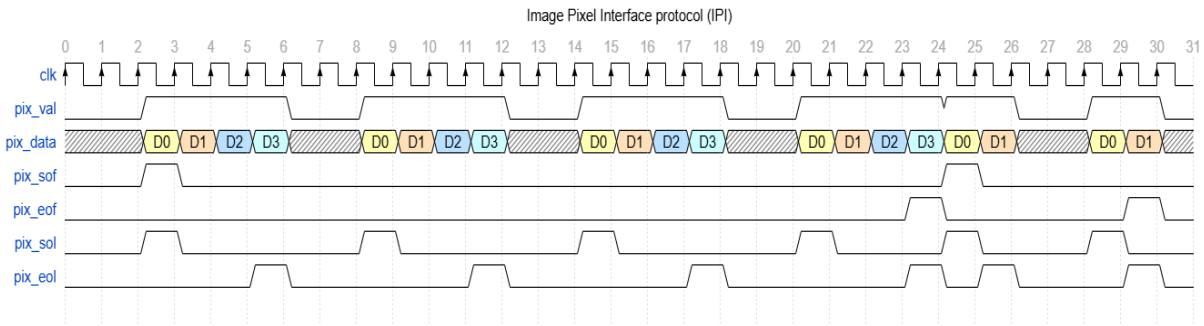
Pentru protocolul standard, se iau în considerare două cadre, așa cum este ilustrat mai jos. Primul cadru are lățimea = 4 și înălțimea = 4, al doilea are lățimea = 2 și înălțimea = 2. Formele de undă corespunzătoare transferului lor sunt reprezentate în figura de mai jos.

D0	D1	D2	D3
D0	D1	D2	D3
D0	D1	D2	D3
D0	D1	D2	D3

Frame 1

D0	D1
D0	D1

Frame 2

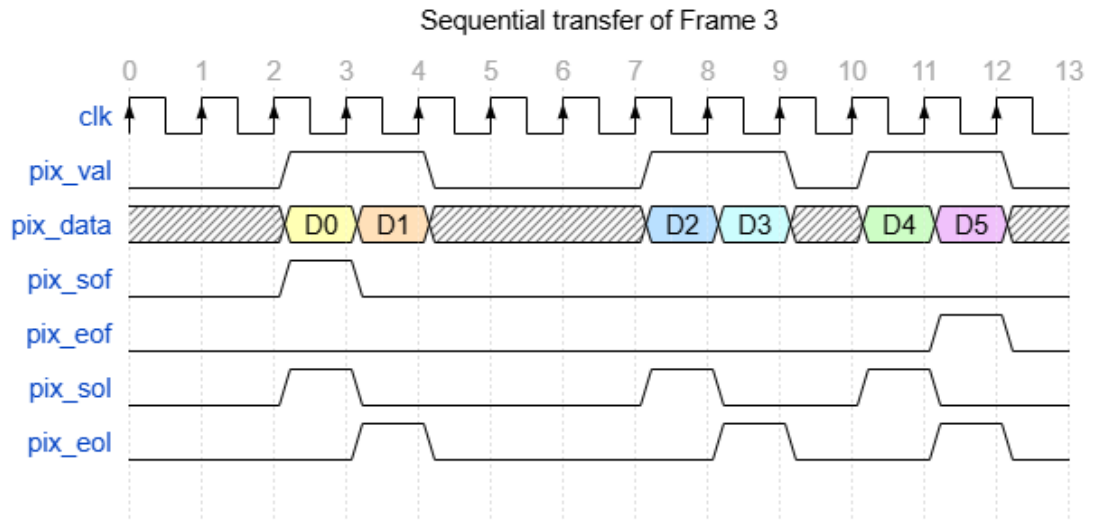



```
{
  signal:
   [
    {name: 'clk'    , wave: 'P.............................'},
    {name: 'pix_val', wave: '0.1...0.1...0.1...0.1...1.0.1.0'},
    {name: 'pix_data', wave: 'x.3456x.3456x.3456x.345634x.34x', data: ['D0','
D1', 'D2', 'D3', 'D0', 'D1', 'D2', 'D3', 'D0', 'D1', 'D2', 'D3', 'D0','D1', '
D2', 'D3', 'D0', 'D1', 'D0', 'D1']},
    {name: 'pix_sof', wave: '0.10...................10.....'},
    {name: 'pix_eof', wave: '0.....................10....10'},
    {name: 'pix_sol', wave: '0.10....10....10....10..10..10.'},
    {name: 'pix_eol', wave: '0....10....10....10....1010..10'},
    {},
    ],
  config: {hscale: 1},
  head:{
   text:'Image Pixel Interface protocol (IPI)',
   tick:0,
   every:1
  },
  foot:{text:'Figure 1'},
}
```

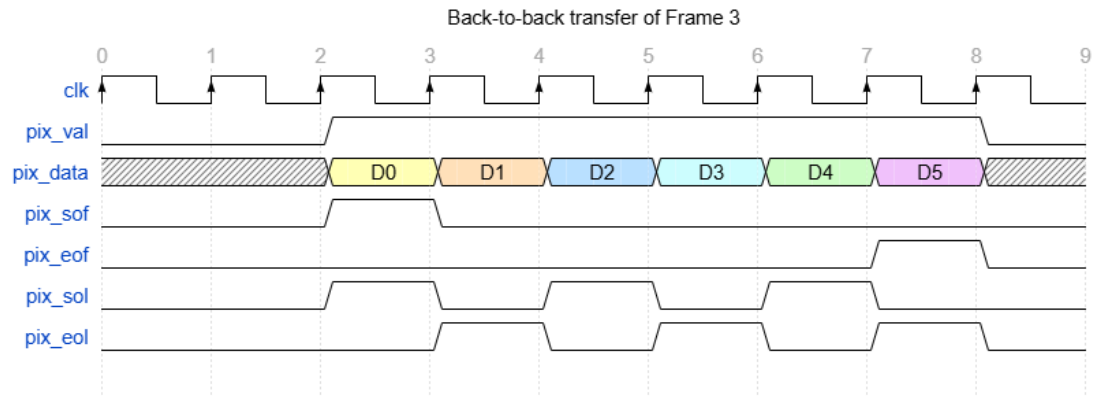
- La al doilea ciclu de ceas, semnalul *pix_sof* (Start of Frame – început de cadru) este activat (setat), marcând începutul unui nou cadru. Simultan, *pix_val* este activat (setat), indicând că datele pixelilor de pe *magistrala pix_data* sunt valide. Prima linie de date ale pixelilor (D0, D1, D2, D3) este transmisă. În plus, semnalul *pix_sol* (Start of Line – început de linie) este activat (setat), indicând începutul unei noi linii, iar pix_eol (End of - Line) este activat (setat) la al 5-lea ciclu de ceas, marcând sfârșitul acestei linii;
- Pentru același cadru, dar pentru o linie nouă, *pix_sol* este (setat) la al 8-lea ciclu de ceas, se transmite o nouă linie de pixeli, capătul liniei fiind marcat cu *pix_eol*;
- Pentru primul cadru, sunt transmise în total 4 linii de pixeli, iar sfârșitul cadrului este marcat cu *pix_eof* la ciclul 23;
- La ciclul 24, *pix_sof* va fi afirmat imediat după căderea pix_eof (transfer imediat consecutiv – *b2b* – al cadrelor). Transferul pixelilor din cadrul 2 este identic cu cel din cadrul 1 (toți pixelii valizi trebuie să fie însoțiți de *pix_val*, începutul unei linii de pixeli este marcat cu *pix_sol*, sfârșitul unei linii marcat cu *pix_eol*, începutul/sfârșitul unui cadru fiind marcat cu *pix_sof*/*pix_eof*);

Pentru un cadru cu următoarele dimensiuni – lățime = 2, înălțime = 3 – așa cum este reprezentat mai jos, transferul său poate avea loc fie secvențial, fie imediat consecutiv (*b2b* – ”spate în spate”) așa cum e ilustrat în cele două figuri de mai jos.

D0	D1
D2	D3
D4	D5




```
{
  signal: [
    {name: 'clk'     , wave: 'P.....'},
    {name: 'pix_val' , wave: '0.1.0..1.01.0'},
    {name: 'pix_data', wave: 'x.34x..56x78x', data: ['D0', 'D1', 'D2', 'D3',
'D4', 'D5']},
    {name: 'pix_sof', wave: '0.10.....'},
    {name: 'pix_eof', wave: '0.....10'},
    {name: 'pix_sol', wave: '0.10...10.10.'},
    {name: 'pix_eol', wave: '0..10...10.10'},
      {}, ],
  config: {hscale: 1},
  head:{
   text:'Sequential transfer of Frame 3',
   tick:0,
   every:1
  },
  foot: {text:'Figure 2'},
}
```




```
{
  signal:[
    {name: 'clk'      , wave: 'P........'},
    {name: 'pix_val' , wave: '0.1.....0'},
    {name: 'pix_data', wave: 'x.345678x', data: ['D0', 'D1', 'D2', 'D3','D4',
'D5']},
    {name: 'pix_sof', wave: '0.10.....'},
    {name: 'pix_eof', wave: '0......10'},
    {name: 'pix_sol', wave: '0.101010.'},
    {name: 'pix_eol', wave: '0..101010'},
      {},],
  config: {hscale: 2},
  head:{
    text:'Back-to-back transfer of Frame 3',
    tick:0,
    every:1
  },
  foot:{text:'Figure 3'},
}
```

- În cazul unei tranzacții secvențiale, intervalul până la activarea (setarea) semnalului valid poate fi configurabilă ; tot configurabilă poate fi și perioada dintre linii și cadre.

- În cazul în care cadrele sunt transmise imediat consecutiv *(b2b),* în următorul ciclu de ceas după ce *pix_eof* coboară la ”0”, este imediat activat *pix_sof*.

Cazuri de eroare

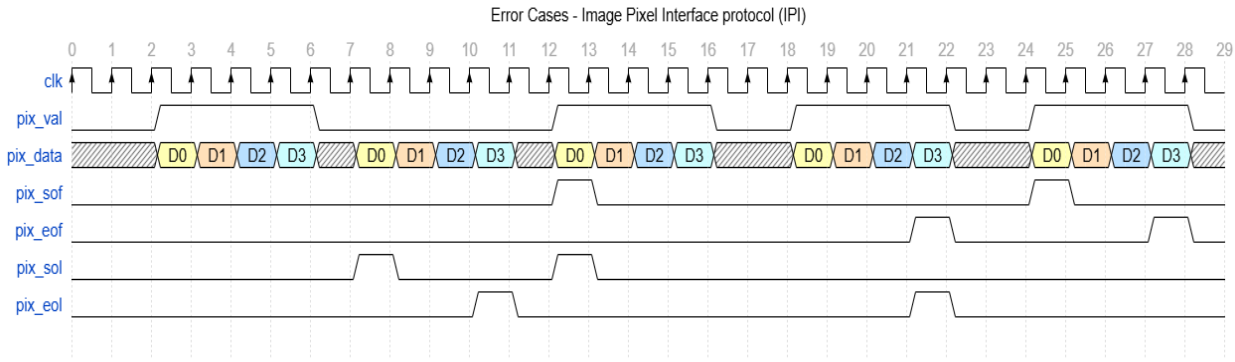
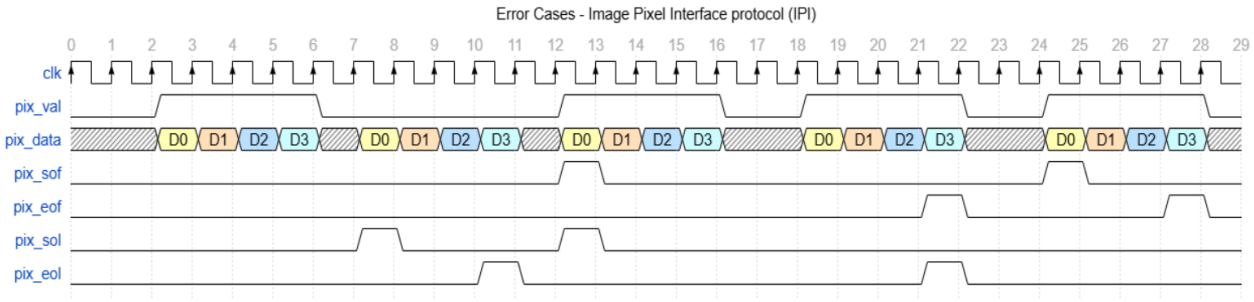


Figure 4

```
{
  signal: [
    {name: 'clk'      , wave: 'P.........................'},
    {name: 'pix_val' , wave: '0.1...0.....1...0.1...0.1...0'},
    {name: 'pix_data', wave: 'x.3456x3456x3456x.3456x.3456x', data: ['D0', 'D
1', 'D2', 'D3', 'D0', 'D1', 'D2', 'D3','D0', 'D1', 'D2', 'D3', 'D0', 'D1', 'D
2', 'D3', 'D0', 'D1', 'D2', 'D3']},
    {name: 'pix_sof', wave: '0..........10.........10...'},
    {name: 'pix_eof', wave: '0..................10....10'},
    {name: 'pix_sol', wave: '0......10...10.............'},
    {name: 'pix_eol', wave: '0.........10.........10.....'},
      {},],
  config: {hscale: 1},
  head:{
   text:'Error Cases - Image Pixel Interface protocol (IPI)',
   tick:0,
   every:1
  },
  foot:{text:'Figure 4'},
}
```

Figura de mai jos ilustrează posibile cazuri de eroare care pot apărea în acest protocol.



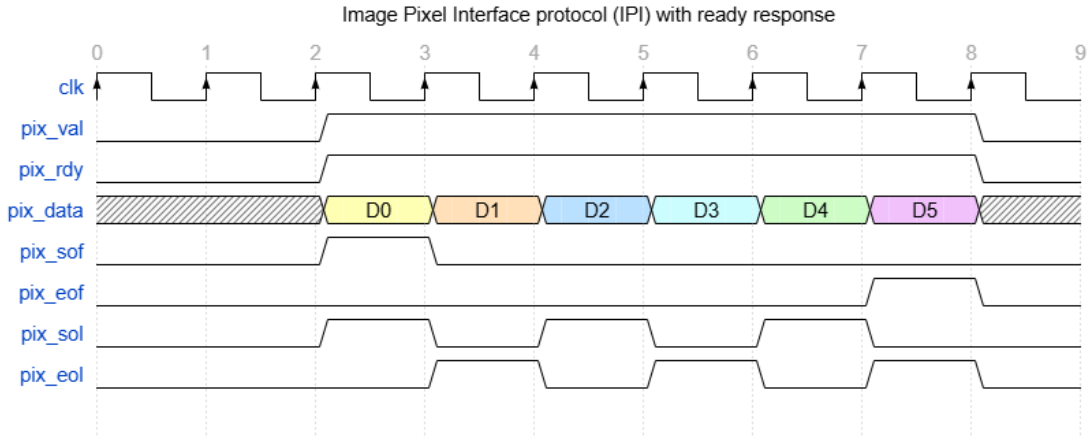
1. La al doilea ciclu, o linie de pixeli (D0, D1, D2, D3 pe pix_data) este transferată împreună cu semnalul *valid*. Totuși, nici *pix_sof* nici *pix_sol* nu au fost activate și, prin urmare, acest lucru va duce la o eroare – solicitantul nu este anunțat de transferul unui nou cadru. Aceeași eroare apare atunci când, după încheierea unei linii sau a unui cadru, pixelii noi sunt transferați fără a fi însoțiți de *pix_sof* sau *pix_sol;*
2. La al 7-lea ciclu, o nouă linie de pixeli corespunzătoare primului cadru este pusă pe *pix_data*, dar nu este validă deoarece *pix_val* nu este activat; acest lucru duce la o eroare – solicitantul nu consideră pixelii valizi din cauza absenței semnalului *valid;*
3. La al 12-lea ciclu, se inițiază un nou transfer de pixeli, împreună cu semnalele corespunzătoare (*pix_val* împreună cu *pix_data*, *pix_sof* și *pix_sol* sunt activate împreună cu primul pixel) – totuși, la ultimul pixel din linie, *pix_eol* nu se activează, rezultând o eroare – desincronizarea pixelilor, prin urmare, următorii care vor sosi pe magistrală vor fi considerați în aceeași linie cu precedenții;
4. La ciclul 24, este inițiat un nou transfer de pixeli, însă, din cauza absenței *pix_sol* și *pix_eol*, Solicitantul este anunțat de noul cadru, dar nu și de o linie nouă, rezultând o eroare.

Interfață pentru fluxuri video (IPI - Image Pixel Interface) cu răspuns de disponibilitate (”Ready”)

Funcționarea normală

Această versiune de protocol cu semnalizare inversă [2, pag. 239] adaugă un mecanism crucial de control al fluxului printr-un *semnal pix_rdy* de la receptor către emițător. Aceasta permite receptorului să transmită "în sens invers" un semnal (”back pressure” – "presiune inversă") care arată când se pot accepta date noi, prevenind umplerea buffer-ului și pierderea datelor. Finalizarea tranzacțiilor necesită ca semnalele *pix_val* și *pix_rdy* să fie activate simultan, permițând o comunicare sigură între sub-sisteme care funcționează la viteze diferite de procesare.

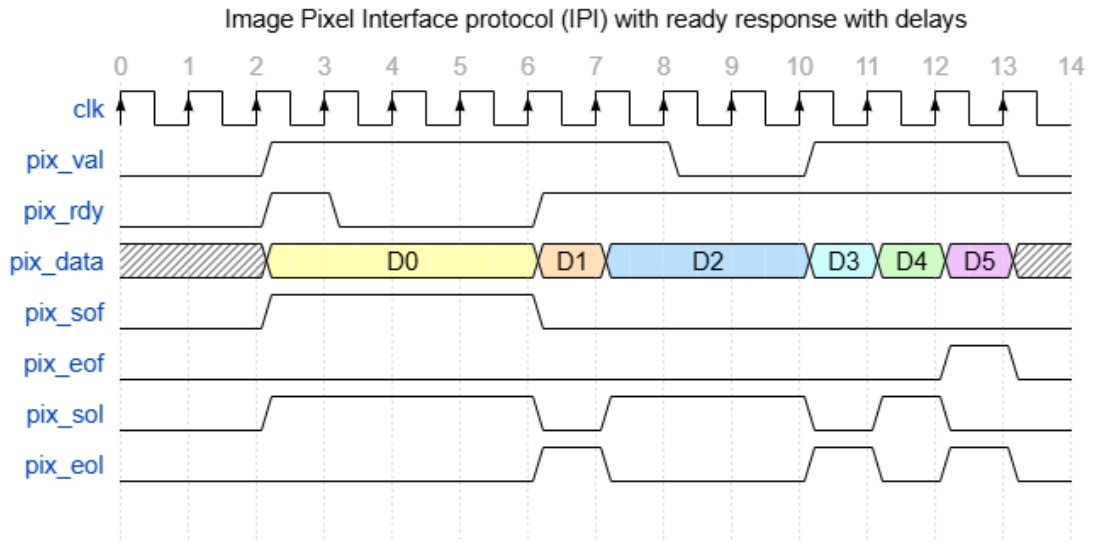
Figura următoare e folosită pentru a exemplifica această variantă a protocolului IPI.
Sunt ilustrate tranzacții imediat consecutive *(b2b)* în care destinatarul semnalizează în mod eficient disponibilitatea de a accepta date.



```
{
  signal: [
    {name: 'clk'     , wave: 'P........'},
    {name: 'pix_val' , wave: '0.1.....0'},
    {name: 'pix_rdy' , wave: '0.1.....0'},
    {name: 'pix_data', wave: 'x.345678x', data: ['D0', 'D1', 'D2', 'D3', 'D4
', 'D5']},
    {name: 'pix_sof' , wave: '0.10.....'},
    {name: 'pix_eof' , wave: '0......10'},
    {name: 'pix_sol' , wave: '0.101010.'},
    {name: 'pix_eol' , wave: '0..101010'},
      {}, ],
  config:{hscale: 1 },
  head  :{
    text:'Image Pixel Interface protocol (IPI) with ready response',
    tick:0,
    every:1
  },
  foot  :{text:'Figure 5'},
}
```

- La ciclul 2, semnalul de pixel valid (*pix_val*) se activează, indicând că date valide de pixeli sunt disponibile pe magistrala pix_data. Simultan, semnalul ”ready” - disponibil (*pix_rdy*) este activat, arătând că receptorul este pregătit să accepte aceste date;
- Tot la ciclul 2, semnalul *pix_sof* este activat, marcând începutul unui nou cadru.
- Din ciclurile 2 până la 8, cu ambele semnale *pix_val*, și *pix_rdy* active (”1”), pixelii D0 până la D5 sunt transmiși secvențial pe magistrala *pix_data* fără întrerupere;
- În timpul transmisiei, la ciclul 5, semnalul *pix_sol* este activ, indicând începutul unei noi linii în cadru;
- Spre finalul transmisiei, la ciclul 7, semnalul *pix_eof* este activat, marcând finalizarea cadrului;
- În timpul ultimului transfer de pixeli, semnalul *pix_eol* indica, de asemenea, sfârșitul liniei curente.

Întreaga transmisie are loc eficient, fără cicluri de ceas irosite, deoarece receptorul rămâne disponibil pe toată durata procesului.




```
{
  signal: [
    {name: 'clk'      , wave: 'P.............'},
    {name: 'pix_val' , wave: '0.1.....0.1..0'},
    {name: 'pix_rdy' , wave: '0.10..1.......'},
    {name: 'pix_data', wave: 'x.3...45..678x', data: ['D0', 'D1', 'D2', 'D3',
'D4', 'D5']},
    {name: 'pix_sof' , wave: '0.1...0.......'},
    {name: 'pix_eof' , wave: '0...........10'},
    {name: 'pix_sol' , wave: '0.1...01..010.'},
    {name: 'pix_eol' , wave: '0.....10..1010'},
      {}, ],
  config:{hscale: 1 },
  head  :{
    text:'Image Pixel Interface protocol (IPI) with ready response with delay
s',
    tick:0,
    every:1
  },
  foot  :{text:'Figure 6'},
}
```

- Similar cu cazul anterior fără întârzieri, transmisia începe cu impulsuri de ceas la ciclul 0, iar pix_val se activează în ciclul 2 pentru a indica date valide;

- Totuși, în acest caz, există perioade în care *pix_rdy* coboară la ”0”, creând intermitențe în transmiterea datelor:
    - Transmisia D0 este extinsă pe ciclurile 2-4 din cauza că *pix_rdy* este ”0” în ciclurile 3-4;
    - Transmiterea D2 este extinsă pe ciclurile 7-9;
    - Transmisia D3 este extinsă pe ciclurile 9-11;
- În ciuda acestor întârzieri, protocolul asigură că nu se pierd date – transmisia se oprește pur și simplu când pix_rdy este ”0” și se reia când revine la ”1”;
- Semnalul *pix_sof* încă marchează începutul cadrului la ciclul 2, iar *pix_eof* marchează sfârșitul cadrului la ciclul 13;
- Semnalele *pix_sol* și *pix_eol* funcționează similar cu cazul anterior, fără întârzieri, dar sincronizarea lor este prelungită din cauza întârzierilor induse de receptor;
- Ca urmare a acestor întârzieri, aceiași 6 pixeli (D0-D5) care erau transmiși în 6 cicluri de ceas în cazul anterior fără întârzieri necesită acum 14 cicluri de ceas pentru a fi transmiși, așa cum este ilustrat în figură.

Interfață pentru fluxuri video – versiunea HSYNC & VSYNC

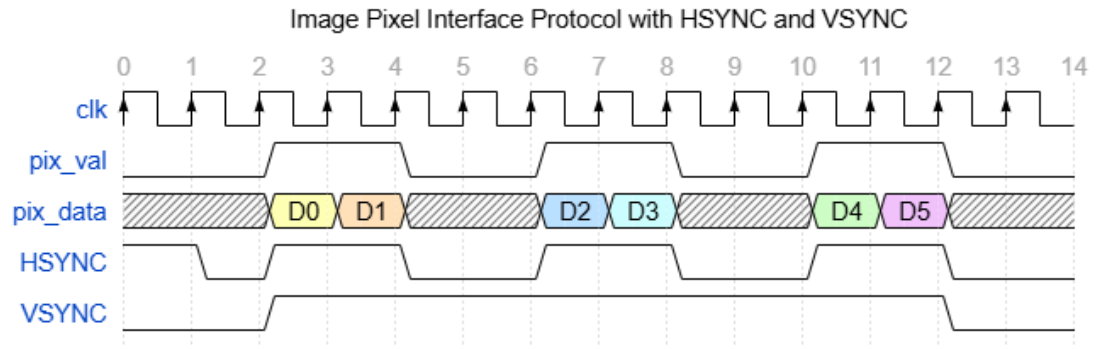
Funcționarea normală

În această versiune de protocol, în loc să se folosească semnalele *pix_sof/eof/sol/eol* , se folosesc **HSYNC** (sincronizare orizontală) și **VSYNC** (sincronizare verticală) pentru a coordona sincronizarea transmiterii datelor pixelilor.

1. Datele pixelilor sunt transmise într-un model de scanare ”raster” (de la stânga la dreapta și de sus în jos);
2. Semnalele **HSYNC** indică începutul și sfârșitul fiecărei linii orizontale;

3. Semnalele **VSYNC** marchează granițele dintre cadrele complete.

Pentru a ilustra această versiune a protocolului, cadrul 3 este folosit ca exemplu de referință, cu semnalizarea completă aferentă prezentată în figura de mai jos.




```
{
  signal: [
    {name: 'clk'      , wave: 'P.............'},
    {name: 'pix_val' , wave: '0.1.0.1.0.1.0.'},
    {name: 'pix_data', wave: 'x.34xx56xx78xx', data: ['D0', 'D1', 'D2', 'D3',
'D4', 'D5']},
    {name: "HSYNC"    , wave: "101.0.1.0.1.0."},
    {name: "VSYNC"    , wave: "0.1.........0."},
      {}, ],
  config:{hscale: 1 },
  head   :{
    text :'Image Pixel Interface Protocol with HSYNC and VSYNC',
    tick :0,
    every:1
  },
  foot   :{text:'Figure 7'},
}
```

- La ciclul 2, semnalul **VSYNC** este activat, indicând începutul unui nou cadru. Simultan, **HSYNC** trece și el la ”1”, marcând începutul primei linii de pixeli;
- La ciclul 2, semnalul de pixel valid (*pix_val*) devine ”1”, indicând că datele valide ale pixelilor sunt disponibile pe magistrala pix_data. Primii pixeli D0 și D1 sunt transmiși secvențial, în timp ce *pix_val* rămâne activ;
- În ciclul 4, **HSYNC** este dezactivat (coborât la ”0”), marcând sfârșitul primei linii. Semnalul *pix_val* coboară pe ”0”, indicând că nu se transmit date valide în această perioadă de blocare a baleiajului pe orizontală;

- La ciclul 6, **HSYNC** urcă din nou la ”1” pentru a marca începutul celei de-a doua linii. Semnalul *pix_val* este apoi activat, iar pixelii D2 și D3 sunt transmiși pe magistrala pix_data;
- Urmând același model, **HSYNC** este activat la ciclul 10 pentru a indica a treia linie, în timpul căreia pixelii D4 și D5 sunt transmiși în timp *ce pix_val* este activ;
- În cele din urmă, la ciclul 12, semnalul **VSYNC** este dezactivat, indicând sfârșitul cadrului după ce toate cele trei linii au fost transmise. Semnalul **HSYNC** este, de asemenea, dezactivat.

10. Protocolul Push / Pop

O stivă FIFO (”First In, First Out”) interacționează de obicei cu un controler pentru încărcarea/descărcarea sincronă a datelor. Generic, stivele FIFO pot fi clasificate fie ca **sincrone,** fie ca **asincrone**.

Modelul FIFO Poate fi aplicat în diverse configurații [9, pag. 684]:

1. **Managementul memoriei**: În informatică, modelul FIFO este adesea folosit în programarea gestiunii memoriei, de exemplu pentru buffer-e, secvențierea tipăririi, accesul la discuri.
2. **Transmiterea/Stocarea datelor**: Cozile FIFO sunt folosite în transmiterea datelor, unde pachetele de date sunt trimise și primite în aceeași ordine [6, pag. 11].
3. **Sisteme de operare/Execuția instrucțiunilor de procesare**: În sistemele de operare, modelul FIFO este folosit pentru arbitrajul accesului egal la resurse (”round-robin”) și în alte contexte de streaming (transmitere de fluxuri) [2, pag. 281].

Operațiuni Push (”încărcare” în stivă)

Încărcarea unui element în stivă presupune:

- **Verificarea spațiului**: Se verifică dacă stiva are spațiu disponibil (etapă necesară doar dacă folosește se folosește o stivă cu memorie limitată).
- **Incrementarea pointerului**: Incrementează pointerul de memorie pentru a indica următoarea locație disponibilă în stivă.
- **Stocarea** datelor: Stochează noile date la locația de memorie indicată de pointerul stivei.

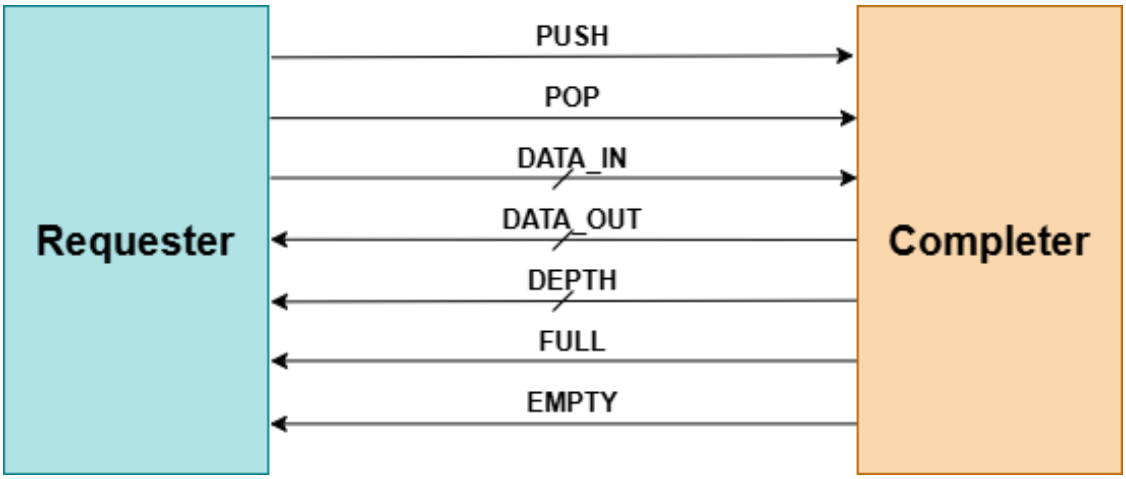
Operațiuni Pop (”extragere” din stivă)

Operațiunea Pop extrage elementul aflat sus, din stivă.

Aceasta implică:

- **Verificarea dacă este goală** (deja): Se verifică dacă stiva s-a golit.
- **Citire Date**: Se citesc datele din locația de memorie indicată de pointerul stivei.
- **Decrementare pointer**: Se scade valoarea indicatorului stivei, pentru a adresa intrarea anterioară.

Diagrama bloc FIFO



Aceste semnale — și anume "plin", "gol", "încarcă" (Push) și "extrage" (Pop) — joacă roluri cruciale în asigurarea manipulării corecte a datelor și în prevenirea erorilor precum ”revărsare” (”overflow”) sau (încercare de) ”extragere dintr-o stivă goală„ – ”underflow”.

Semnal	Sursă	Dimensiune	Descriere
clk	Sistem	1	Ceas
reset	Sistem	1	Reset asincron – activ pe ”0„
data_in	Solicitant	[DWITH-1:0]	Scriere date în FIFO - dimensiunea datelor poate fi variabilă (stabilită prin parametrul DATAWITH)
push	Solicitant	1	Semnalul (sau operațiunea) PUSH este folosit(ă) pentru a adăuga un element structurii de date. În contextul unei stive, această operație ”încarcă” elementul în partea de sus a stivei.
pop	Solicitant	1	Semnalul (sau operațiunea) POP este folosit pentru a elimina cel mai recent element adăugat din structura de date. În contextul unei stive, această operație ”extrage” elementul din partea superioară a stivei
depth	Furnizor	[DEPTH_WIDTH-1:0]	Semnalul DEPTH (”adâncime” – a stivei) este folosit pentru a păstra informația privind dimensiunea stivei FIFO – el crește/scade când se execută o operațiune PUSH/POP. Când stiva e goală adâncimea este 0, când stiva e plină adâncimea ia valoarea maximă (a capacității stivei FIFO).
data_out	Furnizor	[DWITH-1:0]	Datele de ieșire citite din stiva FIFO - dimensiunea datelor poate fi variabilă (stabilită de parametrul DATAWIDTH)
full	Furnizor	1	Semnalul FULL (”plin”) indică faptul că structura de date a atins capacitatea maximă și nu poate accepta noi elemente până când unele din acestea nu sunt eliminate
empty	Furnizor	1	Semnalul EMPTY (”gol”) indică faptul că structura de date nu conține în prezent niciun element.

Parametri:

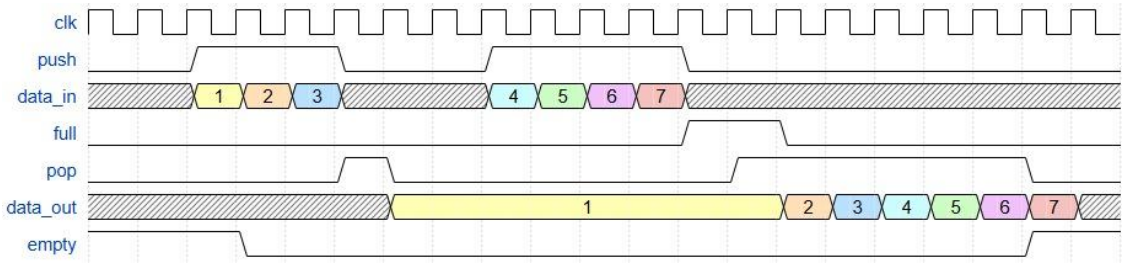
Acest protocol definește mai mulți parametri configurabili care influențează comportamentul și momentul tranzacțiilor:

DWIDTH: Definește lățimea magistralei de date folosită pentru operațiunile de citire și scriere. Determină câți biți de date pot fi transferați într-o singură tranzacție.

DEPTH_WIDTH: Specifică numărul maxim de unități de date care pot fi stocate și transferate printr-o stivă FIFO. Acesta definește capacitatea totală a stivei FIFO (în unități de date).

POP_CONFIG: Specifică numărul de cicluri de ceas necesare pentru a finaliza o operațiune pop într-o stivă FIFO.

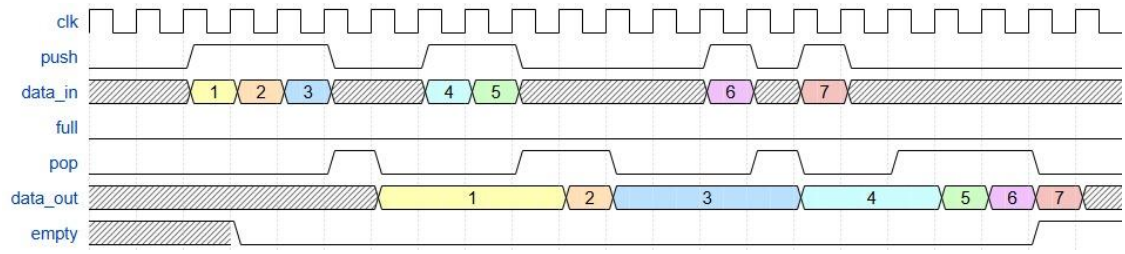
FIFO de adâncime 6




```
{signal: [
   {name: 'clk', wave: 'p.....................'},
   {name: 'push', wave: '0.1..0..1...0........'},
   {name: 'data_in', wave: 'x.345x..6789x........', data: ['1', '2', '3', '4
','5','6', '7']},
   {name: 'full', wave: '0...........1.0......'},
   {name: 'pop', wave: '0....10......1.....0.'},
   {name: 'data_out', wave: '....x.3.......456789x', data: ['1','2','3','4','
5','6','7']},
   {name: 'empty', wave: '1..0..............1.'}
],

head:{
     text:'Bounded fifo (6 depth)',
     tick:0,
     every:1},}
```

FIFO de adâncime 6 – transferuri multiple



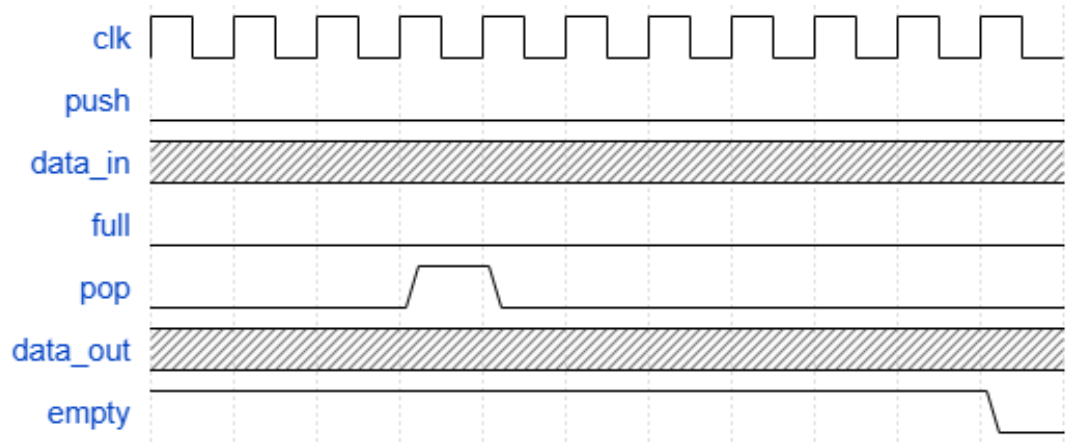

```
{signal: [
   {name: 'clk', wave: 'p.....................'},
   {name: 'push', wave: '0.1..0.1.0...1010.....'},
   {name: 'data_in', wave: 'x.345x.67x...8x9x.....', data: ['1', '2', '3', '4
','5','6', '7']},
```

```
   {name: 'full', wave: '0.....................'},
   {name: 'pop', wave: '0....10..1.0..10.1..0.'},
   {name: 'data_out', wave: '....x.3...45...6..789x', data: ['1','2','3','4',
'5','6','7']},
   {name: 'empty', wave: '.1.0..............1.'}
],

head:{
     text:'   Bounded fifo ( 6 depth) - multiple transfer',
     tick:0,
     every:1},}
```

Cazuri de eroare

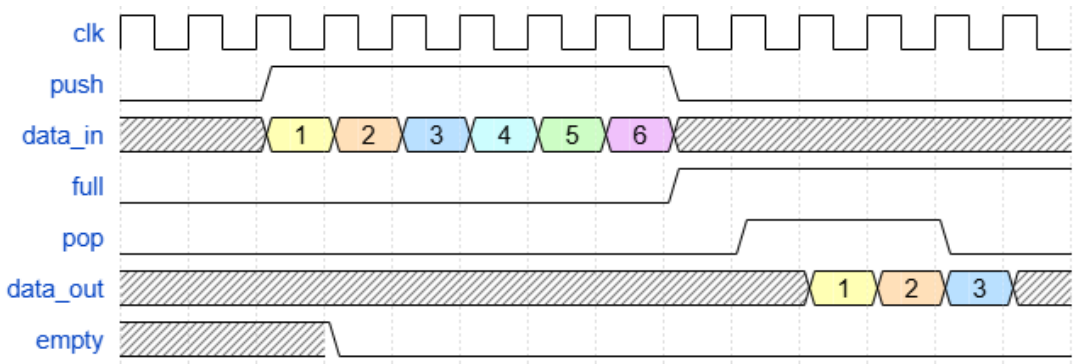
Tentativă de operațiune POP când stiva e goală




```
{signal: [
   {name: 'clk', wave: 'p.........'},
   {name: 'push', wave: '0.........'},
   {name: 'data_in', wave: 'x.........', data: ['1', '2', '3', '4','5','6',
'7']},
   {name: 'full', wave: '0.........'},
   {name: 'pop', wave: '0..10......'},
   {name: 'data_out', wave: '....x......', data: ['1','2','3','4','5','6','7
']},
   {name: 'empty', wave: '1........0'}
],
```

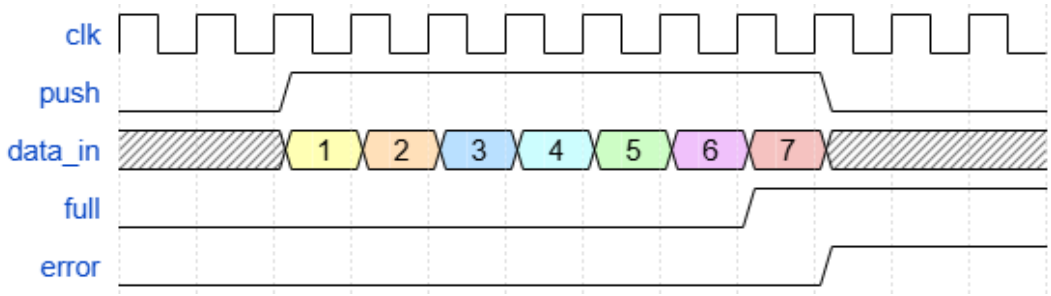
```
head:{
     text:'Try to make a pop transaction when is empty',
     tick:0,
     every:1},}
```

Semnalul ”plin” nu e dezactivat când se execute operațiunea POP




```
{signal: [
   {name: 'clk', wave: 'p.............'},
   {name: 'push', wave: '0.1.....0.....'},
   {name: 'data_in', wave: 'x.345678x.....', data: ['1', '2', '3', '4','5','6
', '7']},
   {name: 'full', wave: '0.......1.....'},
   {name: 'pop', wave: '0........1..0.'},
   {name: 'data_out', wave: '....x.....345x', data: ['1','2','3','4','5','6',
'7']},
   {name: 'empty', wave: '.1.0.........'}
],
head:{
     text:'Full signal is not deasserted when pop transaction occurs',
     tick:0,
     every:1},}
```

Încercare de PUSH când stiva e plină

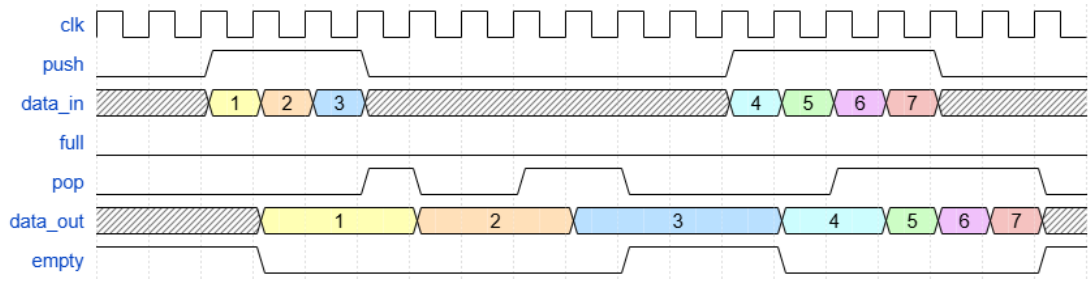



```
{signal: [
   {name: 'clk', wave: 'p..........'},
   {name: 'push', wave: '0.1......0..'},
   {name: 'data_in', wave: 'x.3456789x..', data: ['1', '2', '3', '4','5','6',
'7']},
   {name: 'full', wave: '0.......1...'},
   {name: 'error', wave: '0........1..'}
],

head:{
     text:'Try to make a push transaction when is full',
     tick:0,
     every:1},}
```

Situații speciale:

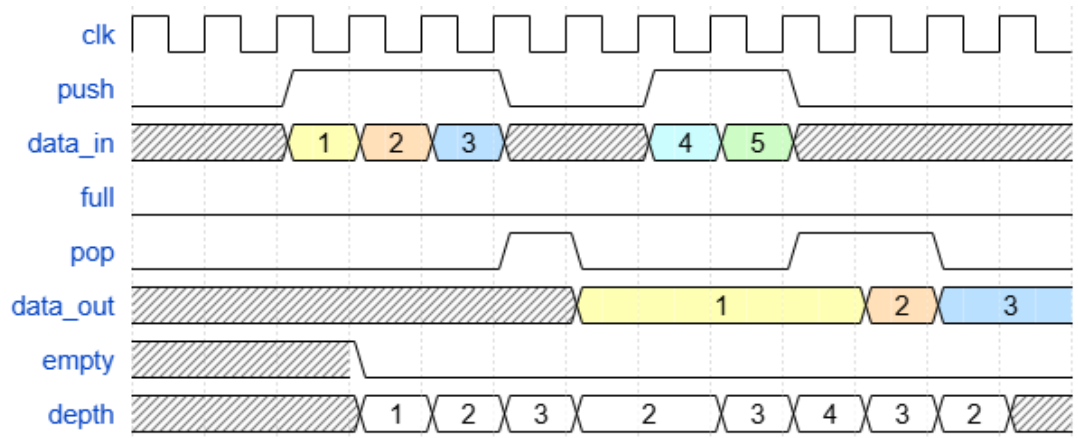
Citire anticipată




```
{signal: [
   {name: 'clk', wave: 'p................'},
   {name: 'push', wave: '0.1..0.....1...0..'},
   {name: 'data_in', wave: 'x.345x.....6789x..', data: ['1', '2', '3', '4','
5','6', '7']},
   {name: 'full', wave: '0................'},
   {name: 'pop', wave: '0....10.1.0...1...0'},
   {name: 'data_out', wave: '.x.3..4..5...6.789x', data: ['1','2','3','4','5
```

```
','6','7']},
   {name: 'empty', wave: '1..0......1..0....1'}
],
config: { },
head:{
     text:'Read ahead',
     tick:0,
     every:1},}
```

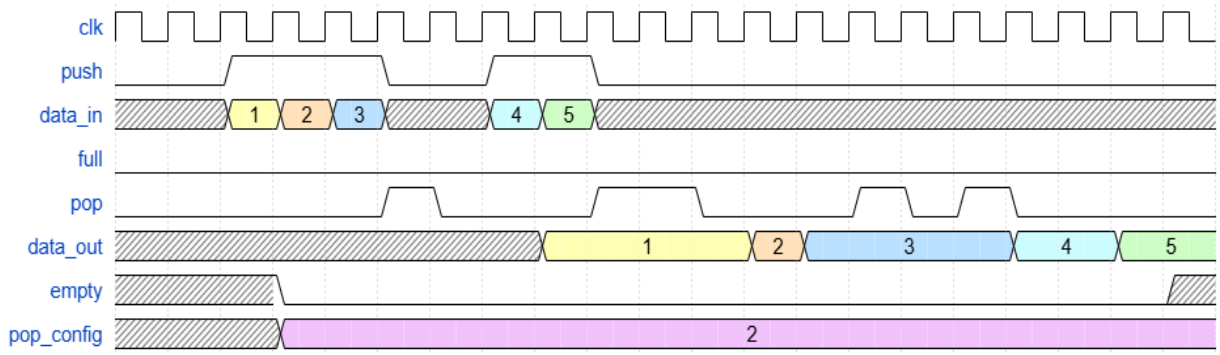
FIFO de adâncime 6 – exemplu de semnal DEPTH




```
{signal: [
   {name: 'clk', wave: 'p.....'},
   {name: 'push', wave: '0.1..0.1.0...'},
   {name: 'data_in', wave: 'x.345x.67x...', data: ['1', '2', '3', '4','5','6
', '7']},
   {name: 'full', wave: '0.....'},
   {name: 'pop', wave: '0....10..1.0.'},
   {name: 'data_out', wave: '....x.3...45.', data: ['1','2','3','4','5','6','
7']},
   {name: 'empty', wave: '.1.0.....'},
   {name: 'depth', wave: '.x.2222.2222x', data: ['1','2','3','2','3','4','3',
'2']}
],

head:{
     text:'   Bounded fifo ( 6 depth) - depth signal example',
     tick:0,
     every:1},}
```

FIFO de adâncime 6 – exemplu de semnal pop_config




```
{signal: [
   {name: 'clk', wave: 'p.....................'},
   {name: 'push', wave: '0.1..0.1.0...........'},
   {name: 'data_in', wave: 'x.345x.67x...........', data: ['1', '2', '3', '4
','5','6', '7']},
   {name: 'full', wave: '0.....................'},
   {name: 'pop', wave: '0....10..1.0..1010...'},
   {name: 'data_out', wave: '....x...3...45...6.7.', data: ['1','2','3','4','
5','6','7']},
   {name: 'empty', wave: '.1.0................x'},
   {name: 'pop_config', wave: '.x.8.................', data: ['2']}
],

head:{
     text:'Bounded fifo ( 6 depth) - Pop config signal example',
     tick:0,
     every:1},}
```

Metode SystemVerilog pentru gestiunea cozilor

Metoda	Descriere
size()	Returnează numărul de elemente din coadă
insert()	Inserează elementul dat în poziția indicată de index
delete()	Șterge elementul la poziția specificată de index
push_front()	Inserează elementul dat la începutul cozii
push_back()	Inserează elementul dat la sfârșitul cozii
pop_front()	Elimină primul element al cozii și îi returnează valoarea
pop_back()	Elimină ultimul element al cozii și îi returnează valoarea

Stiva FIFO asincronă

O stivă FIFO asincronă este un tip de memorie buffer care permite transferul de date între două domenii de ceas ce operează la frecvențe diferite. De obicei, aceste stive sunt folosite în sisteme unde datele trebuie să treacă dintr-un domeniu de ceas în altul, ceea ce este denumit generic "traversarea domeniului de ceas". Astfel, FIFO asincron ajută la sincronizarea fluxului de date între două sisteme care funcționează cu ceasuri diferite. FIFO asincron asigură un schimb de date fiabil (fără pierderi sau coruperi de date).

Semnal	Sursă	Dimensiune	Descriere
wr_clk	Sistem	1	Ceas scriere
rd_clk	Sistem	1	Ceas citire
reset	Sistem	1	Reset asincron – activ pe ”0„
data_in	Solicitant	[DWITH-1:0]	DATA_IN: Datele de intrare care sunt scrise în stiva FIFO - dimensiunea datelor poate fi variabilă (fixată prin parametrul DATAWITH)
push	Solicitant	1	Semnalul (sau operațiunea) PUSH este folosit(ă) pentru a adăuga un element structurii de date. În contextul unei stive, această operație ”încarcă” elementul în partea de sus a stivei.
pop	Solicitant	1	Semnalul (sau operațiunea) POP este folosit pentru a elimina cel mai recent element adăugat din structura de date. În contextul unei stive, această operație ”extrage” elementul din partea superioară a stivei
data_out	Furnizor	[DWITH:0]	Datele de ieșire citite din stiva FIFO - dimensiunea datelor poate fi variabilă (stabilită de parametrul DATAWIDTH)
full	Furnizor	1	Semnalul FULL (”plin”) indică faptul că structura de date a atins capacitatea maximă și nu poate accepta noi elemente până când unele din acestea nu sunt eliminate
empty	Furnizor	1	Semnalul EMPTY (”gol”) indică faptul că structura de date nu conține în prezent niciun element.

Principiul de funcționare

Scrierea: Datele sunt scrise în stiva FIFO în domeniul ceasului de scriere. Pointerul de scriere incrementează la fiecare operație de scriere.

Citirea: Datele sunt citite din stiva FIFO în domeniul ceasului de citire. Pointerul de citire incrementează cu fiecare operație de citire.

Mecanismul de sincronizare: Deoarece operațiunile de citire și scriere sunt pilotate de ceasuri diferite, se folosesc tehnici de sincronizare precum contoare cu cod Gray și sincronizatoare cu ceasuri duale pentru a preveni metastabilitatea [2, pag. 153].

Codul Gray în FIFO asincrone

Specific codului Gray este că două valori succesive diferă cu un singur bit. Este folosit în proiectarea FIFO asincrone pentru a transfera în siguranță valorile pointerilor între diferite domenii de ceas.

Deoarece doar un bit se schimbă la un moment dat, codul Gray reduce riscul de metastabilitate atunci când se transferă valori de pointer între domenii de ceas.

Previne operațiunile incorecte de citire/scriere cauzate de contoare binare parțial actualizate (dacă se modifică doi sau mai mulți biți ai numărului oferit de un contor oarecare când acesta incrementează, pot apare valori de tranziție eronate, pentru scurt timp, dacă unul din acei biți s-a modificat și celălalt n-a apucat să se actualizeze).

Codificarea Gray permite compararea sigură a pointerilor de citire și scriere din diferite domenii de ceas.

Cum funcționează codul Gray în FIFO?

Conversia de la cod binar la cod Gray: Pointerii de scriere și citire sunt convertiți în cod Gray înainte de a fi transmiși între domeniile de ceas.

Sincronizare: Pointerii codați Gray sunt sincronizați folosind sincronizatoare bazate pe bistabile (”flip-flop”) fiind astfel evitată metastabilitatea.

Conversia de la cod Gray la cod binar:

După sincronizare, pointerii codați Gray sunt convertiți înapoi în cod binar pentru decodarea adreselor.

Într-un FIFO asincron, semnalul empty (”gol”) este generat în domeniul ceasului de citire și depinde de compararea read_ptr cu o versiune sincronizată a write_ptr.

Deoarece write_ptr provine dintr-un alt domeniu de ceas, trebuie sincronizat cu cel puțin două bistabile pentru a evita metastabilitatea.

Mai mult, pointerii sunt de obicei codificați Gray pentru a reduce erorile în timpul traversării domeniului de ceas.

Acești pași (conversia în cod Gray + sincronizare dublă) introduc o întârziere de aproximativ 2-3 cicluri de ceas în detectarea stării de golire (”EMPTY”).

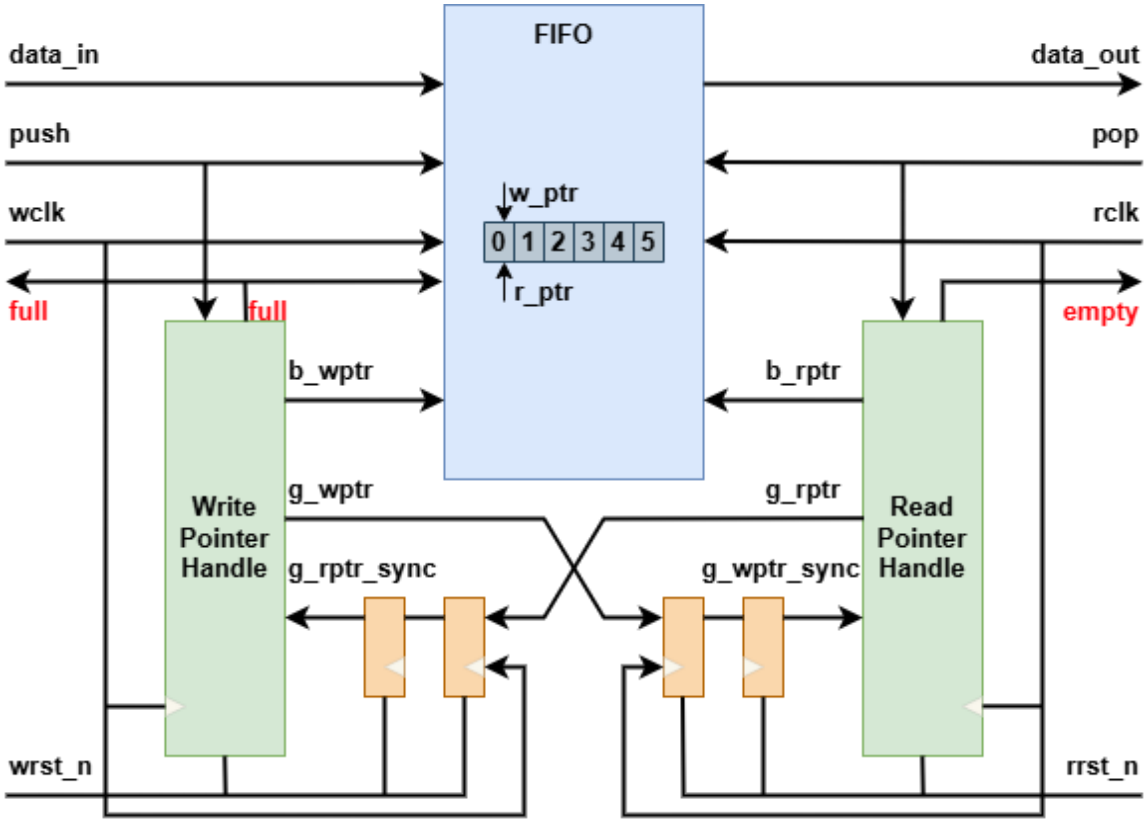
În sincronizarea FIFO cu două ceasuri, sunt necesare de obicei câteva cicluri dde ceas, după e ceas – în funcție de implementare – pentru ca semnalele de stare precum FULL și EMPTY să se sincronizeze între domeniile de ceas.

În exemplele de mai jos, latența sincronizării este de 2 cicluri de ceas.

Activarea acestor semnale are loc pe frontul corect ă a ceasului: - *EMPTY* este activat pe ceasul de citire - *FULL* este activat pe ceasul de scriere.

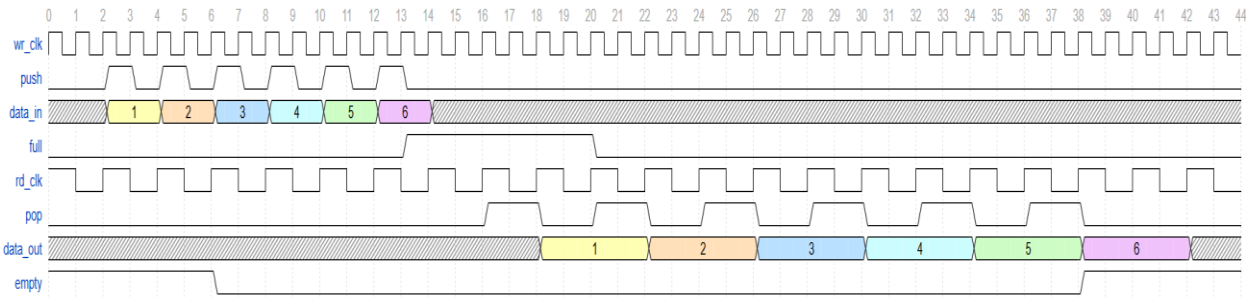
Totuși, din cauza latenței de sincronizare a ceasului dual, dezactivarea acestor semnale este întârziată și are loc doar după ce logica de sincronizare confirmă actualizarea stării FIFO la trecerea de la un domeniu de ceas la celălalt [3, pag. 103].

Diagrama bloc a stivei FIFO asincrone



(sursa: vlsiverify.com)

Forme de undă pentru stiva FIFO asincronă:




```
{signal: [
  {name: 'wr_clk', wave: 'p.....................................'},
  {name: 'push', wave: '0.101010101010.........................'},
  {name: 'data_in', wave: 'x.3.4.5.6.7.8.x.....................', dat
```

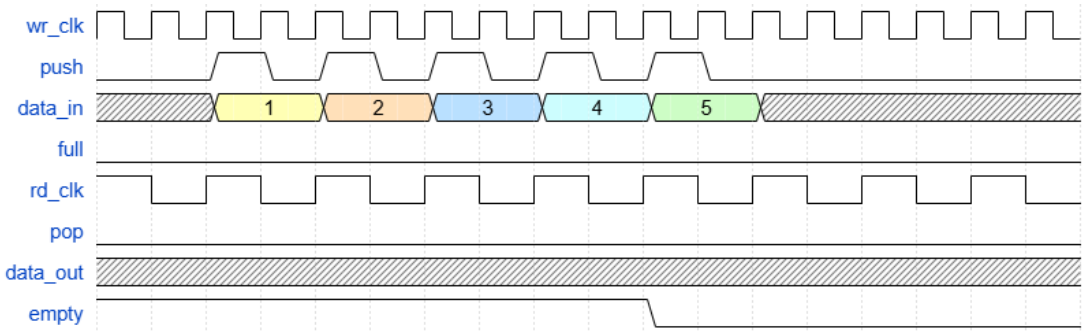
```
a: ['1', '2', '3', '4','5','6']},
  {name: 'full', wave: '0.............1.....0.....................'},
  {name: 'rd_clk', wave  :'hlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhl'},
  {name: 'pop', wave: '0..............1.0.1.0.1.0.1.0.1.0.1.0.....'},
  {name: 'data_out', wave: '..............x..3...4...5...6...7...8...x.', da
ta: ['1','2','3','4','5','6']},
  {name: 'empty', wave: '1...0.....................................1.'}

],

head:{
     text:'Asynchronous FIFO waveform',
     tick:0,
     every:1},}
```

Cazuri de eroare

Stiva FIFO asincronă: semnalul empty nu e dezactivat la operațiunea PUSH

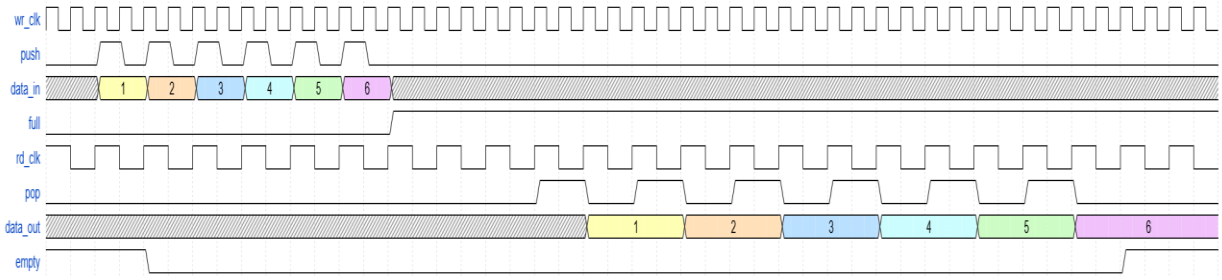



```
{signal: [
  {name: 'wr_clk', wave: 'p.................'},
  {name: 'push', wave: '0.1010101010......'},
  {name: 'data_in', wave: 'x.3.4.5.6.7.x.....', data: ['1', '2', '3', '4','5
','6']},
  {name: 'full', wave: '0.................'},
  {name: 'rd_clk', wave  :'hlhlhlhlhlhlhlhlhl'},
  {name: 'pop', wave: '0.................'},
  {name: 'data_out', wave: '.................', data: ['1','2','3','4','5','
6']},
```

```
  {name: 'empty', wave: '1.........0.......'}

],
head:{
     text:'Asynchronous FIFO: Empty signal is not deasserted when push transa
ction occurs',
     tick:0,
     every:1},}
```

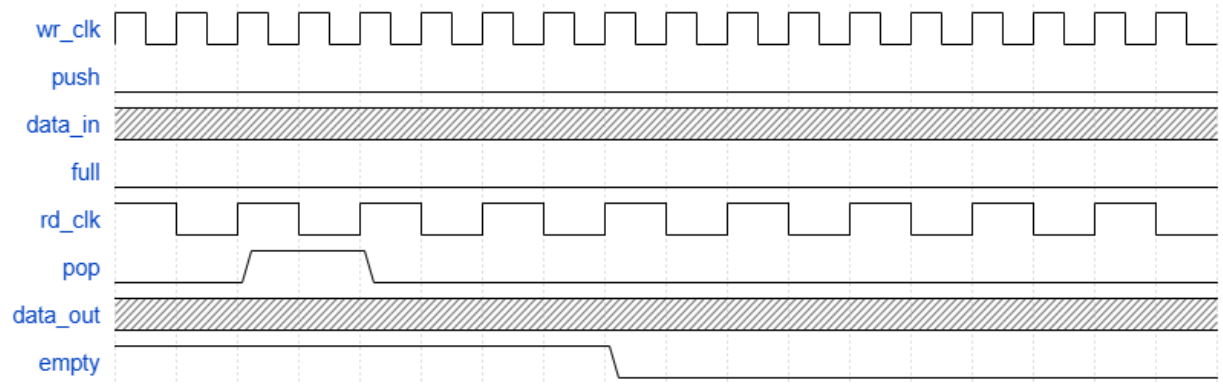
Stiva FIFO asincronă: Semnalul full nu este dezactivat la operațiunea POP




```
{signal: [
  {name: 'wr_clk', wave: 'p.........................................'},
  {name: 'push', wave: '0.101010101010.............................'},
  {name: 'data_in', wave: 'x.3.4.5.6.7.8.x.............................',
data: ['1', '2', '3', '4','5','6']},
  {name: 'full', wave: '0............1.............................'},
  {name: 'rd_clk', wave  :'hlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhlhl
'},
  {name: 'pop', wave: '0................1.0.1.0.1.0.1.0.1.0.1.0.....'},
  {name: 'data_out', wave: '................x..3...4...5...6...7...8.....
', data: ['1','2','3','4','5','6']},
  {name: 'empty', wave: '1...0.....................................1...'}

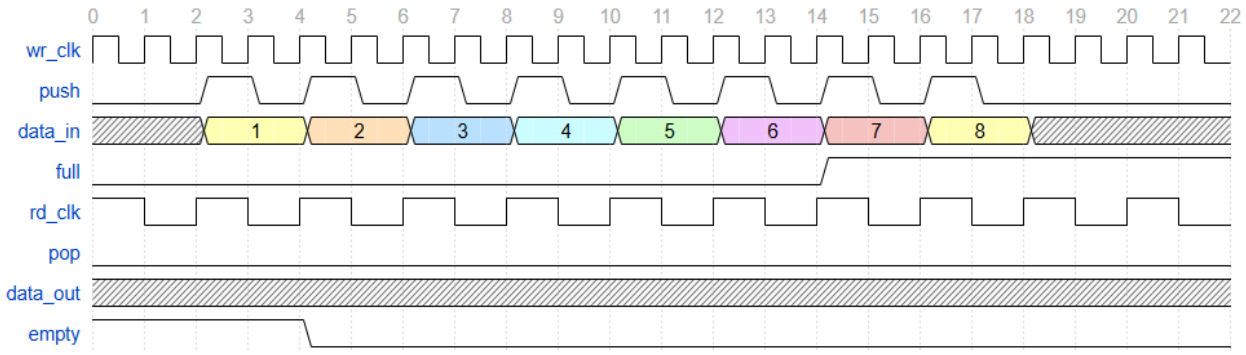
],
head:{
     text:'Asynchronous FIFO:  Full signal is not deasserted when pop transac
tion occurs, full is not asserted right',
     tick:0,
     every:1},}
```

Stiva FIFO asincronă: Se încearcă POP când stiva e goală




```
{signal: [
  {name: 'wr_clk', wave: 'p.................'},
  {name: 'push', wave: '0.................'},
  {name: 'data_in', wave: 'x..........x.....', data: ['1', '2', '3', '4','5
','6']},
  {name: 'full', wave: '0.................'},
  {name: 'rd_clk', wave  :'hlhlhlhlhlhlhlhlhl'},
  {name: 'pop', wave: '0.1.0.............'},
  {name: 'data_out', wave: '.................', data: ['1','2','3','4','5','
6']},
  {name: 'empty', wave: '1.......0.........'}
],
head:{
     text:' Asynchronous FIFO: Try to make a pop transaction when is empty
',
     tick:0,
     every:1},}
```

Stiva FIFO asincronă: Se încearcă PUSH când stiva e plină



```
{signal: [
  {name: 'wr_clk', wave: 'p.....................'},
  {name: 'push', wave: '0.1010101010101010....'},
  {name: 'data_in', wave: 'x.3.4.5.6.7.8.9.3.x...', data: ['1', '2', '3', '4
','5','6', '7', '8']},
  {name: 'full', wave: '0.............1.......'},
  {name: 'rd_clk', wave  :'hlhlhlhlhlhlhlhlhlhlhl'},
  {name: 'pop', wave: '0.....................'},
  {name: 'data_out', wave: '.....................', data: ['1','2','3','4','
5','6']},
  {name: 'empty', wave: '1...0................'}

],
head:{
     text:' Asynchronous FIFO:   Try to make a push transaction when is full
',
     tick:0,
     every:1},}
```

11. Protocolul Stărilor (”Status Protocol”)

Descrierea modelului de *”control prin stare”*

Semnal: stare (n biți) Semnalele de stare, pe o magistrală de n biți, indică starea actuală de funcționare a unui sub-sistem hardware implicat într-o operațiune cu mai multe faze [3, pag. 23].

Fiecare din aceste valori codificate binar reprezintă o stare specifică în ciclul de viață al respectivei operațiuni.

Stările definite de protocol sunt:

000 (RESET) – Modulul este în starea sa inițială de resetare. Toate registrele și semnalele interne sunt neinițializate (”0”) sau setate la valori implicite. De obicei se trece în această stare la pornire sau după o resetare globală a sub-sistemului.

001 (ÎNCEPUTUL TRANSFERULUI) – Sub-sistemul a fost activat și începe să se inițializeze pentru o nouă operațiune sau tranzacție.

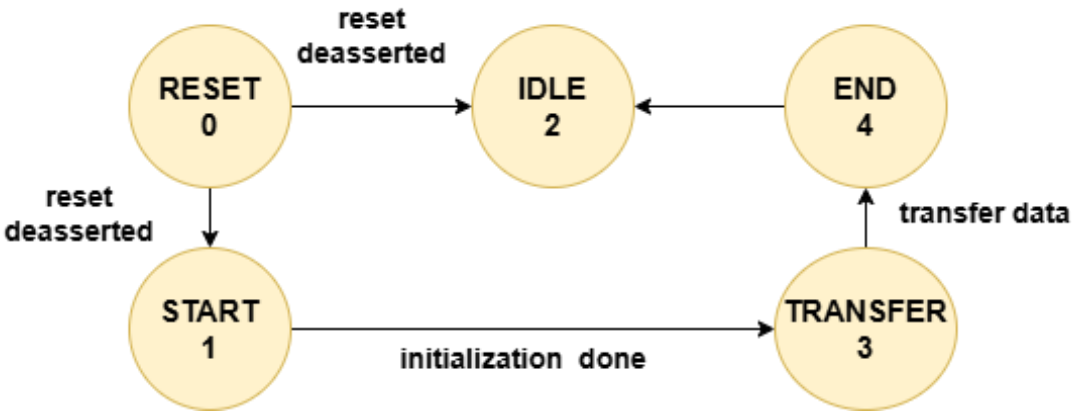
010 (IDLE) – Sub-sistemul este în stare de așteptare, fără operațiuni active. Rămâne în această stare până când este primită o cerere validă de servire.

011 (TRANSFER) – Sub-sistemul efectuează activ un transfer de date. Aceasta indică faptul că semnalele de control precum valid și ready sunt activate, iar datele sunt transferate.

100 (SFÂRȘITUL TRANSFERULUI) – Ultimele date au fost transferate. Această stare semnalează finalizarea operațiunii curente. Apoi, sub-sistemul revene de obicei la starea IDLE sau la starea RESET.

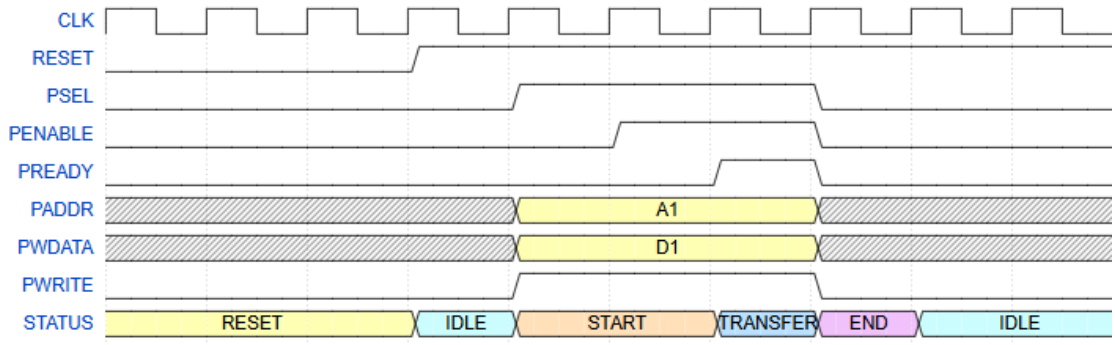
Tranzițiile de stare sunt declanșate de logica de control intern și de semnale externe precum start, valid, ready și done (”terminare”) [3, pag. 117].

Monitorizarea semnalului de stare permite sincronizarea, analiza performanței la nivel de sistem și depanarea (”debugging”).

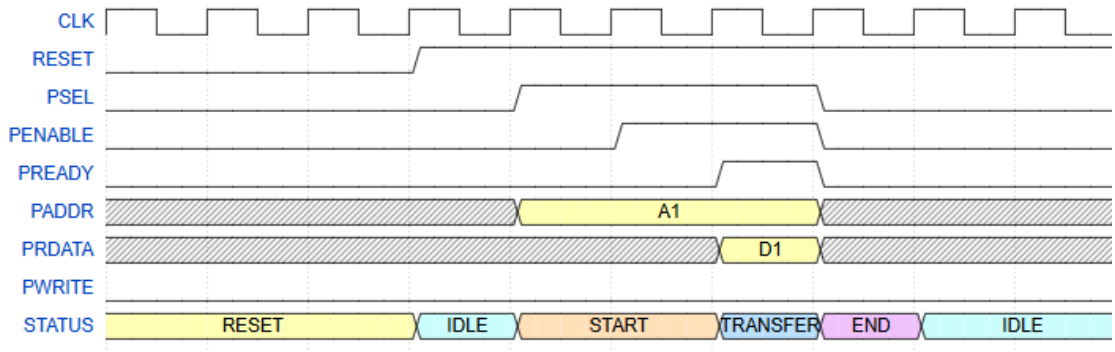


Automatul cu stări finite

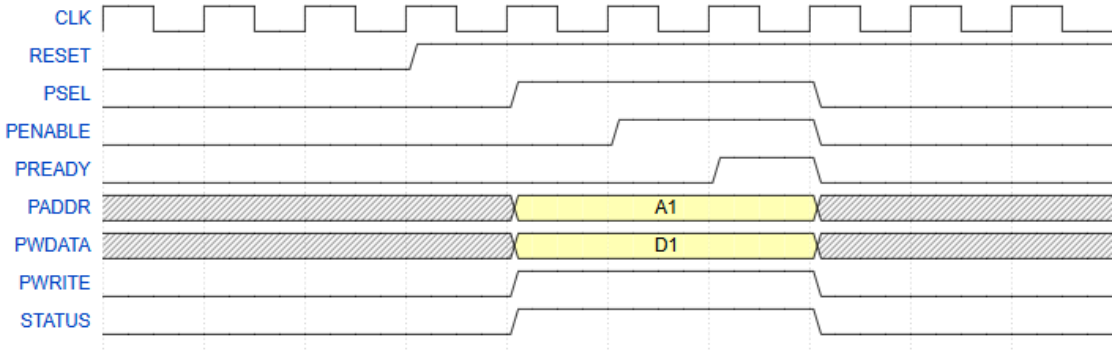
Forme de undă



Transfer de scriere cu semnalizarea stării

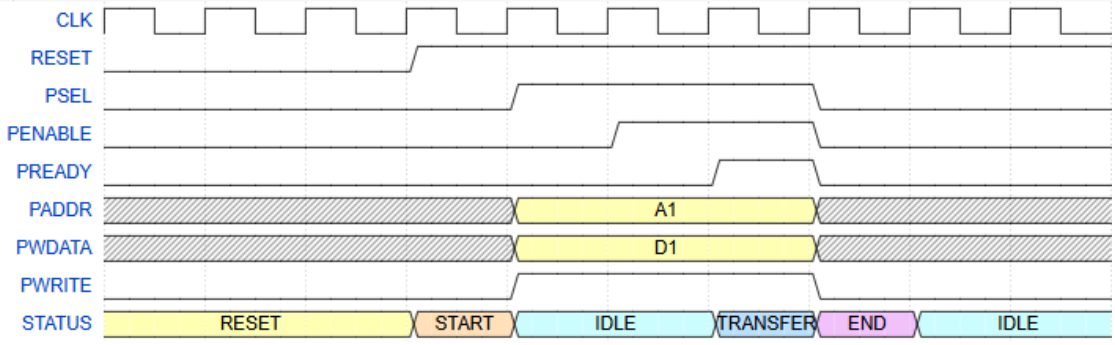


Transfer de citire cu semnalizarea stării

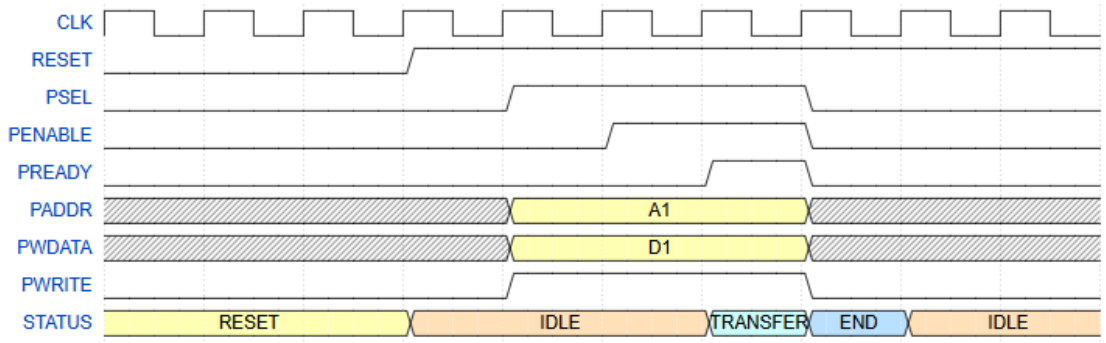


Transfer de scriere – varianta cu un bit de stare

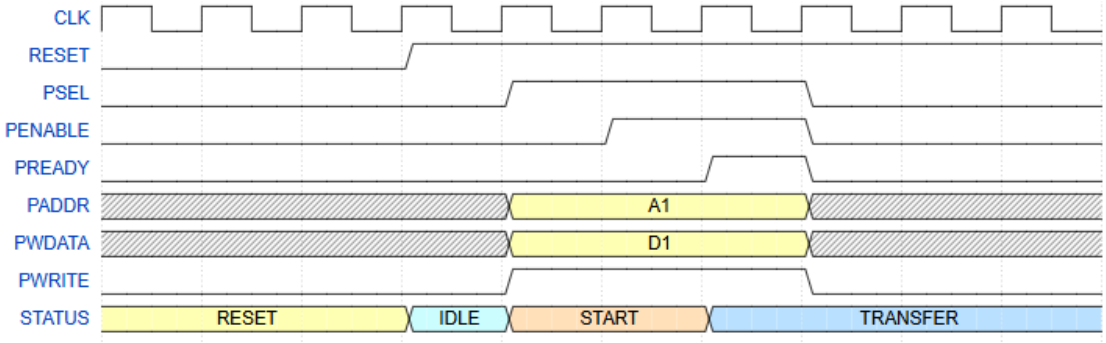
Cazuri de eroare



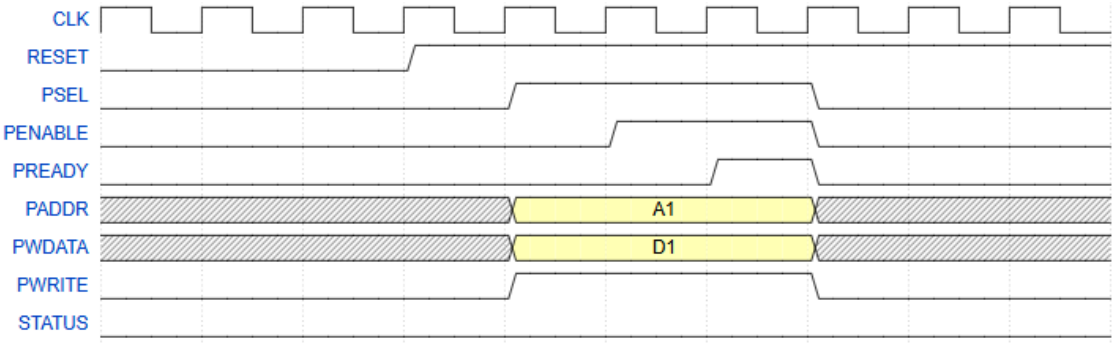
Starea Start semnalizată greșit



Începerea unui transfer de citire fără semnalizare Start



Transfer de scriere cu blocarea mașinii algoritmice în starea Transfer



Transfer de scriere cu ne-activarea semnalului de Stare (variante de un bit)

REFERINȚE

[1] G. De Micheli and L. Benini, ”Networks on Chips: Technology and Tools” - Morgan Kaufmann - San Francisco, CA, USA, 2006

[2] A. Jantsch and H. Tenhunen, ”Networks on Chip” - Kluwer Academic Publishers - Boston, MA, USA, 2004.

[3] N. Enright Jerger, L.-S. Peh, and T. M. Pinkston, ”On-Chip Networks” - Morgan & Claypool Publishers - San Rafael, CA, USA, 2017.

[4] V. S. Reddy, J. Sachin Rewaria, R. Kumar Thota, C. Anil Kumar and V. Kumar Chikoti, "Design and Verification of APB Protocol," *2025 3rd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, Coimbatore, India, 2025

[5] P. Dukare, A. Gokhale and V. Ingale, "Development of AMBA ACE protocol," *7th International Conference on Computing in Engineering & Technology (ICCET 2022)* Conference, 2022, pp. 222-225

[6] W.J. Dally, B.P. Towles - Principles and practices of interconnection networks – Elsevier - 2004.

[7] C.Grecu, A.Ivanov, P.Pande, A.Jantsch, E.Salminen, U.Ogras, R.Marculescu - Towards open network-on-chip benchmarks - First International Symposium on Networks-on-Chip (NOCS'07), 2007, May - (pp. 205-212) - IEEE – https://jantsch.se/AxelJantsch/papers/2007/NOCS-Benchmarks.pdf

[8] „AMBA CHI, ACE, ACE-Lite, AXI, AHB, APB and OCP transaction protocols” - Arteris S.A., https://www.arteris.com/learn/axi-protocol-arm-ace-ahb-apb/

[9] W.J. Dally, B.P. Towles - "Route packets, not wires: on-chip inteconnection networks." In Proceedings of the 38th annual Design Automation Conference, pp. 684-689. 2001 - www.cs.uaf.edu/2012/fall/cs441/641_papers/route_packets_not_wires_2001_dally.pdf