ARHITECTURA CALCULATOARELOR 2003/2004
CURSUL 11

4.2.4 Sincronizarea magistralei

Magistralele pot fi împărțite în două categorii distincte pe baza ceasului de lucru:

- Magistrala sincronă are o linie pilotată de un oscilator cu cuarț. Semnalul pe această linie constă dintr-un semnal dreptunghiular cu frecvența de ordinul Mhz (unități, zeci sau sute). Toate activitățile de pe magistrală necesită un numar întreg de cicluri, numite cicluri de magistrală.
- Magistrala asincronă, nu are un ceas. Ciclii de magistrală pot fi de orice mărime şi nu este necesar să fie aceeaşi pentru toate perechile de componente.

În cele ce urmează vom examina fiecare tip de magistrală.

4.2.4.1 Magistrale sincrone

Ca **exemplu** despre modul de funcționare al unei magistrale, considerăm temporizarea din figura 4.11. În acest exemplu se va folosi un ceas cu frecvența de 40 Mhz, care dă un ciclu de magistrală de 25 nanosecunde. În timp ce acesta poate părea mai puțin rapid în comparație cu un CPU la 1 Ghz sau chiar mai mult, până de curând existau doar câteva magistrale cu mult mai rapide. De exemplu, magistrala ISA care se gasea pe toate PC-urile bazate pe tehnologie Intel lucrau la o frecvență de 8,33 Mhz şi chiar popularul tip de magistrala PCI functionează la 33 sau 66 Mhz. Motivele din care multe tipuri de magistrale nu sunt atât de rapide pe cât s-ar dori sunt: probleme de proiectare cum ar fi desincronizarea magistralelor şi necesitatea compatibilității cu modelele anterioare.

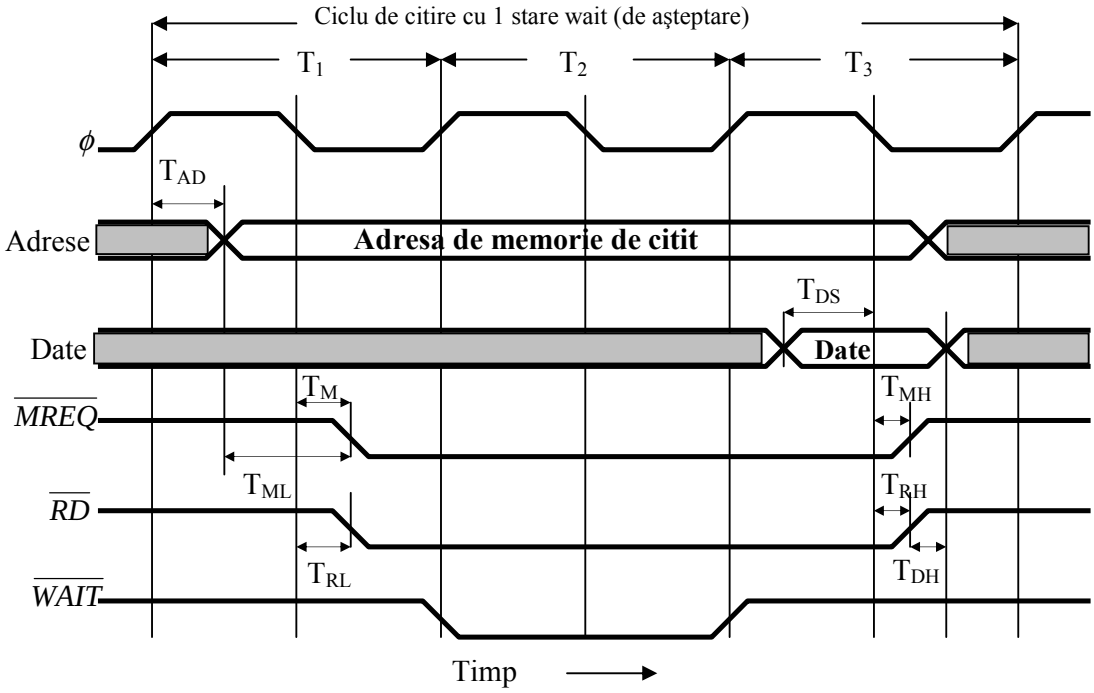

Figura 4.11 Exemplu de ciclu de magistrală sincronă

În exemplul nostru, vom accepta că citirea din memorie durează 40 nanosecunde din momentul în care adresa este stabilă. După cum vom vedea pe scurt, cu aceşti parametri, va fi nevoie de trei cicli de magistrală pentru a citi un cuvânt din memorie. Primul ciclu începe la frontul crescator al lui T_1, iar al treilea se termină la frontul crescător al lui T_4, după cum apare în figură. Să notăm faptul că niciunul dintre fronturile ascendente ori descendente n-a fost desenate vertical, deoarece nici un semnal electric nu-şi poate schimba valoarea fără să treacă o perioadă de timp, oricât de mică. În acest exemplu vom presupune că este nevoie de 1 ns pentru ca un semnal să se schimbe dintr-o stare în alta. Ceasul, liniile de adresă, de date, $\overline{MREQ}$, $\overline{RD}$ şi $\overline{WAIT}$ sunt toate desenate pe aceeaşi scală de timp.

Începutul lui T_1 este definit de frontul crescător al ceasului. În intervalul T_1 CPU-ul pune adresa cuvântului pe care-l doreşte extras din memorie pe liniile de adrese. Deoarece adresa nu este o singură valoare, precum ceasul, nu o putem desena ca o singură linie în figură: va apare ca 2 linii, cu o intersectare la momentul în care adresa se schimbă. Mai mult, haşura anterioară intersecției indică faptul că valoarea haşurată nu este semnificativă. Folosind acelasi mod de haşurare, vom observa cum conținutul liniilor de date nu este semnificativ până în intervalul T_3.

- După ce liniile de adrese au avut timp să se stabilizeze la noile valori, sunt activate $\overline{MREQ}$ şi $\overline{RD}$. Primul semnal indică faptul că memoria (spre deosebire de o componentă I/O) este accesată, iar al doilea semnal că memoria este accesată pentru citire (negată pentru scriere). Având în vedere faptul că memoria necesită 40 ns. după ce adresa e stabilă (în primul ciclu de ceas) aceasta nu poate furniza datele cerute în intervalul T_2.
- Pentru a-i spune CPU-ului să nu aştepte, memoria activează linia $\overline{WAIT}$ la începutul lui T_2. Această acțiune va insera timpi de aşteptare – stări wait (cicli de magistrală suplimentari) - până când memoria a terminat lucrul la operațiunea curentă şi $\overline{WAIT}$ devine negat. În exemplul nostru, a fost inserat un timp de aşteptare T_2 deoarece memoria funcționează prea încet.
- La începutul lui T_3, când este sigur că va avea datele în cursul ciclului curent, memoria furnizează $\overline{WAIT}$ negat.
- În timpul primei jumătăți a lui T_3 memoria pune datele pe liniile de date.
- La frontul descrescător al lui T_3, CPU-ul citeşte liniile de date, păstrând valorile într-un registru intern.
- După citirea datelor, CPU-ul furnizează $\overline{MREQ}$ şi $\overline{RD}$ negate. Dacă e nevoie, un alt ciclu de memorie poate începe la următorul front crescător al ceasului.

În specificațiile de temporizare din figura 4.12, 8 simboluri care apar în diagrama de timp din figura 4.11 sunt explicate:

- TAD, de exemplu, este intervalul de timp între frontul crescător al ceasului T_1 şi setarea liniilor de adresă. Potrivit specificatiilor de temporizare, TAD ≤ 11 ns Aceasta înseamnă că producatorul CPU-ului garantează ca în timpul oricărui ciclu de citire, CPU-ul va furniza adresa spre a fi citită în cel mult 11 ns de la mijlocul frontului crescător al lui T_1.
- Specificațiile de temporizare indică deasemenea necesitatea ca datele să fie pe liniile de date la cel puțin TDS (5 ns) înainte de frontul descrescător al lui T_3, pentru a-le acorda timp să se stabilizeze înainte ca CPU-ul să citească datele. Combinația de reguli pe TAD şi TDS înseamnă că, în cel mai rău caz, memoria va avea numai 62,5-11-5 = 46,5 ns de la momentul când adresa apare până când trebuie să furnizeze datele. Deoarece 40 ns sunt suficiente, chiar

şi în cel mai rău caz, o memorie de 40 ns poate întotdeauna răspunde în timpul lui T_3. O memorie de 50 nanosecunde ar trebui să insereze o a doua stare wait şi să răspundă în timpul lui T_4.

- Specificăţiile de temporizare garantează faptul că adresa va fi setată cu cel puţin 6 ns înainte ca $\overline{MREQ}$ să fie activat. Acest interval de timp poate fi important dacă $\overline{MREQ}$ face selecţia pe cipul de memorie deoarece unele memorii necesită un timp de setare a adresei anterior selecţiei cipului. În mod normal, proiectantul de sisteme n-ar trebui să aleagă un cip de memorie care necesită un timp de setare a adresei de 10 ns.
- Regulile impuse pentru TM şi pe TRL înseamnă că $\overline{MREQ}$ şi $\overline{RD}$ vor fi ambele activate în cele 8 ns de la frontul descrescător de ceas din T_1. În cel mai rău caz, cipul de memorie va avea numai 25 + 25 – 8 –5 = 37 ns la dispoziţie, după activarea lui $\overline{MREQ}$ şi $\overline{RD}$, pentru a pune datele pe magistrală. Această regulă este suplimentară şi independentă faţă de intervalul de 40 de ns necesar după ce adresa este stabilă.
- TMH şi TRH stabilesc cât de mult timp este necesar pentru negarea lui $\overline{MREQ}$ şi $\overline{RD}$ după ce datele au fost citite.
- În final TDH spune cât timp memoria trebuie să ţină datele pe magistrală după ce $\overline{RD}$ a fost negat. În ceea ce priveşte CPU-ul din cazul nostru, memoria poate înlătura datele de pe magistrală imediat după ce RD a fost negat. Pentru unele CPU-uri datele trebuie ţinute stabile ceva mai mult timp.

Simbol	Parametrii	Min	Max	Unit
TAD	Îtârzierea furnizării adresei		11	ns
TML	Stabilizarea adresei înainte de $\overline{MREQ}$	6		ns
TM	Întârzierea $\overline{MREQ}$ faţă de frontul descrescător al lui Φ în T_1		8	ns
TRL	Întârzierea $\overline{RD}$ faţă de frontul descrescător al lui Φ în T_1		8	ns
TDS	Timpul de stabilizare a datelor înaintea frontului descrescător al lui Φ	5		ns
TMH	Întârzierea $\overline{MREQ}$ faţă de frontul descrescător al lui Φ în T_3		8	ns
TRH	Întârzierea $\overline{RD}$ faţă de frontul descrescător al lui Φ în T_3		8	ns
TDH	Timpul de păstrare a datelor după negarea lui $\overline{RD}$	0		ns

Figura 4.12 Specificaţii pentru intervalelel de temporizare

Figura 4.12 este o versiune foarte simplificată a regulilor de temporizare reale. În realitate, sunt specificate multe alte momente critice. Cu toate acestea, ne dă o foarte bună idee despre modul de funcţionare al magistralelor.

O ultimă idee ar fi că semnalele de control pot fi accesate "high" sau "low". Depinde de proiectanţii de magistrale să hotărască cea mai convenabilă variantă. Alegerea este arbitrară. Unii pot privi acest fapt ca echivalent al alegerii unui programator de a reprezenta blocurile libere pe disc într-un bitmap cu "0" sau "1".

4.2.4.2 Magistrale asincrone

Cu toate că magistralele sincrone sunt uşor de controlat din cauza intervalelor mici de timp, au unele probleme. De exemplu, totul funcționează în multipli de cicli de ceas. Dacă un CPU şi memoria pot completa un transfer în 3,1 cicli, trebuie să folosească totuşi 4 cicli deoarece ciclii fracționali nu sunt permişi.

Mai mult, odată ce un ciclu de magistrală a fost ales, şi memoria şi plăcile I/O au fost construite pentru acesta, este dificil să se poată folosi avantajele îmbunătățirilor ulterioare în tehnologie. De exemplu, să presupunem ca la câțiva ani după ce sistemul din paragraful anterior a fost construit, noi memorii apar cu timpi de acces de 20 ns în loc de 40 ns. Acestea ar putea înlătura timpii de aşteptare, mărind viteza sistemului. Apoi să presupunem că apar memorii de 10 ns. Nu ar mai exista nici un fel de îmbunătățire a performanței, pentru că timpul minim de citire pentru acest proiect este de 2 cicli.

Simplu vorbind, dacă o magistrală sincronă are o colecție eterogenă de componente, unele rapide şi altele mai puțin rapide, magistrala trebuie să urmeze modul de lucru al celei mai lente componente, caz în care cele rapide nu vor mai funcționa la adevăratul lor potențial.

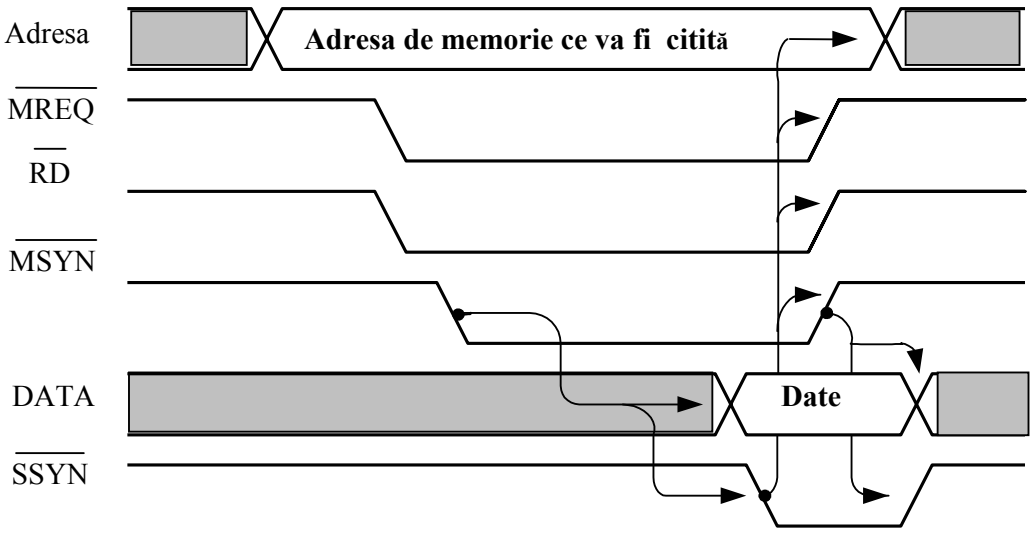

Figura 4.13 Exemplu de funcționare pe o magistrală asincronă

Tehnologia combinată poate fi accesibilă, folosind magistralele asincrone, adică cele fără ceas, precum se arată în figura 4.13. În locul sincronizării tuturor componentelor la ceas, după ce masterul de magistrală a activat adresa, $\overline{MREQ}$, $\overline{RD}$ şi orice mai are nevoie, trimite un semnal special pe care-l vom numi $\overline{MSYN}$ (Master SYNcronization). Când magistrala "slave" primeşte acest semnal, îşi indeplineşte sarcina pe cât de repede posibil. Când termină, trimite un semnal $\overline{SSYN}$ (Slave SYNcronization).
Imediat după ce componenta master primeşte semanlul $\overline{SSYN}$, ştie că datele sunt disponibile, aşa că le citeşte, apoi liniile de adresă devin inactive, împreună cu $\overline{MREQ}$, $\overline{RD}$ şi $\overline{MSYN}$. Când slave vede inactivarea lui $\overline{MSYN}$, ştie că s-a terminat ciclul, inactiveaza $\overline{SSYN}$, şi ne

aflăm iarăşi în situaţia originală, cu toate semnalele dezactivate, aşteptând următoarea componentă master.

Diagramele de temporizare în cazul magistralelor asincrone (şi uneori chiar al magistralelor sincrone) folosesc săgeți pentru a arăta cauza şi efectul, precum în figura 4.13. Accesarea $\overline{MSYN}$ duce la activarea liniilor de date şi deasemenea face ca şi componenta slave să activeze $\overline{SSYN}$. Activarea $\overline{SSYN}$, la rândul ei, cauzează dezactivarea liniilor de adresă, $\overline{MREQ}$, $\overline{RD}$ şi $\overline{MSYN}$. În final, dezactivarea lui $\overline{MSYN}$ cauzează dezactivarea lui $\overline{SSYN}$, care finalizează operaţiunea de citire.

Setul de semnale care apar în exemplul anterior este numit legatură totală – full handshake. Faza esențială a operaţiunii constă în patru evenimente:

1. Este emis semnalul $\overline{MSYN}$,
2. $\overline{SSYN}$ este emis ca răspuns la MSYN,
3. $\overline{MSYN}$ este negat ca răspuns la $\overline{SSYN}$,
4. $\overline{SSYN}$ este negat ca răspuns la negarea $\overline{MSYN}$.

Este evindent că legăturile totale sunt independente de ceasuri. Fiecare eveniment e cauzat de un eveniment anterior, nu de un impuls de ceas. Dacă o pereche anume master - slave funcţionează încet, nu va afecta în nici un mod o altă pereche master - slave care este mult mai rapidă.

Avantajul unei magistrale asincrone ar trebui să fie evident în acest moment, dar adevărul este că majoritatea magistralelor sunt sincrone, deoarece construirea sistemelor sincrone este mult mai facilă. CPU-ul doar transmite semnalele şi memoria doar reacţionează. Nu există reacţie inversă (cauză şi effect), iar în cazul în care componentele au fost alese corespunzător, totul va funcţiona perfect fără crearea unei legături propriu-zise. Deasemenea, se fac o mulţime de investiţii în tehnologia de magistrale sincrone.

4.2.5 Arbitrarea magistralelor

Până acum, am stabilit tacit că avem de a face doar cu un singur master pentru magistrală, CPU-ul. În realitate, chipurile I/O trebuie să treacă în regim de master de magistrală pentru a scrie şi a citi memoria şi, de asemenea, să producă întreruperi. De asemenea, este posibil ca şi coprocesoarele să aibă nevoie de a lucra în regim de master de magistrală. Atunci apare întrebarea: “Ce se va întâmpla dacă două sau mai multe componente trec în regim de master de magistrală simultan?” Pentru a preveni haosul, un mecanism de arbitrare a magistralelor este necesar.

Mecanismele de arbitrare a magistralelor pot fi centralizate sau descentralizate. Să studiem întâi problema arbitrajului **centralizat**.

O formă simplă de arbitraj centralizat este prezentată în figura 4.14a. În această schemă, un arbitru de magistrală hotărăşte cine va trece în regim master de magistrală. Multe CPU-uri au posibilitatea de arbitrare integrată în chip, dar uneori este nevoie de un chip separat. Magistrala conţine o singură linie pentru cererea SAU – cablat (wired – OR) care poate fi transmisă oricând de către unul sau mai multe componente. Nu este posibil pentru arbitru să

ştie câte componenete solicită magistrala la un moment dat. Singurele categorii pe care le poate distinge sunt câteva cereri sau nici o cerere.

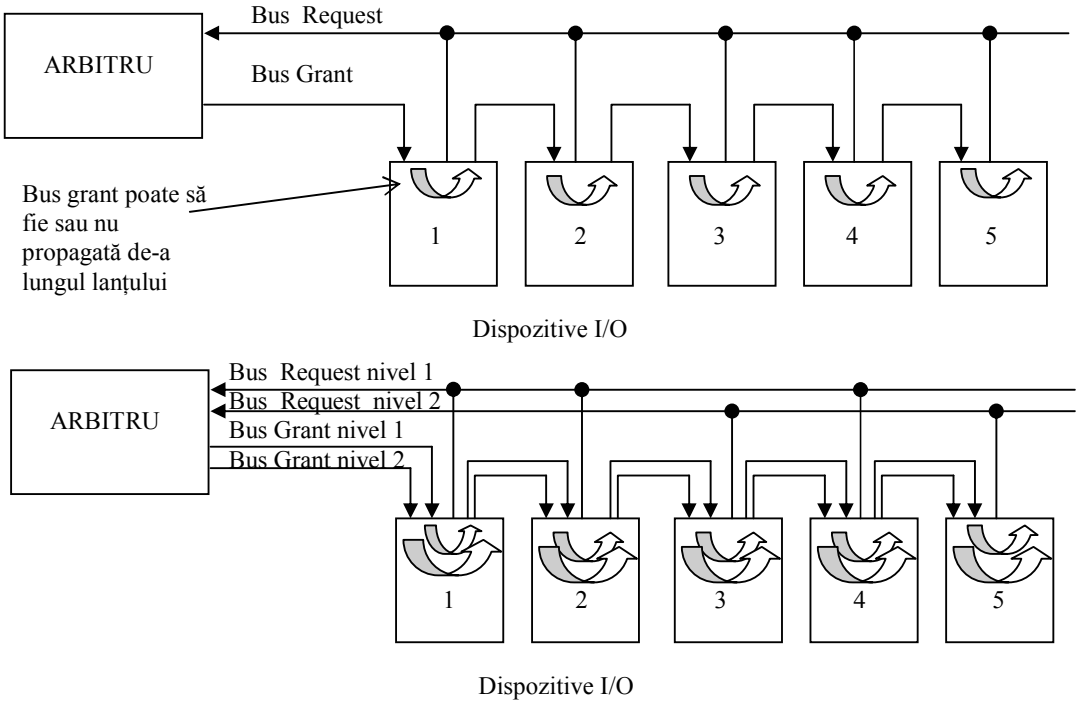

Figura 4.14 (a) Un arbitru centralizat pe un nivel folosind daisy chain. (b) Acelaşi arbitru, dar cu două nivele

Când arbitrul vede o cerere de utilizare a magistralei, permite accesarea ei activând linia magistralei de acceptare. Această linie este conectată în serie prin toate componentele I/O. Când componenta cea mai apropiată fizic de arbitru primeşte aprobarea, verifică dacă a făcut o cerere. Dacă a făcut-o, preia controlul magistralei, dar nu trimite aprobarea în continuare pe linie. Dacă nu a facut nici o cerere, trimite în continuare pe linie aprobarea, la următoarea componentă, care se comportă lafel, şi tot aşa până când o componentă acceptă şi preia controlul magistralei. Această schemă este numită înlănțuire de tip "daisy chain" şi are proprietatea că toate componentele au fixate un anumit nivel de prioritate depinzând de cât de aproape sunt de arbitru. Cea mai apropiată componentă câştigă.

Pentru a eluda prioritățile implicite bazate pe distanța de la arbitru, multe magistrale au nivele de prioritate multiple. Pentru fiecare nivel de prioritate apar câte o linie de cerere şi o linie de acceptare. Figura 4.14b are 2 nivele, 1 şi 2 (magistralele adevărate au adeseori 4, 8 sau 16 nivele). Fiecare componentă este ataşată la unul din nivelele de cerere ale magistralei, componentele care au nevoie de accesarea mai rapidă a magistralei având o prioritate mai mare. În figura 4.14b componentele 1, 2 şi 4 folosesc prioritatea 1 în timp ce componentele 3 şi 5 folosesc prioritatea 2. Dacă nivelele de prioritate multiple sunt cerute simultan, arbitrul permite accesarea numai celui cu prioriatea cea mai mare. Între componente de aceeaşi prioritate, se foloseşte înlănțuirea de tip daisy. In figura 4.14b, în cazul conflictelor, componenta 2 câştigă asupra componentei cu numărul 4, care câştigă asupra celei cu numărul 3. Componenta 5 are cea mai mică prioritate din cauza faptului că este la sfârşitul lanțului daisy cu cea mai mică prioritate.

Ca o observație, nu este tehnic necesară legarea liniei de acceptare a nivelului 2 a magistralei serial prin componentele 1 şi 2 având în vedere faptul că ele nu pot emite cereri pe ea. Oricum, ca o convenție de implementare, este mai uşor să se lege toate liniile de acceptare prin toate componentele decât să se facă legături speciale care depind de prioritatea componentelor.

Unele circuite de arbitrare au o a treia linie, pe care componenta o activează când aceasta a acceptat cererea şi preia controlul magistralei. Îndată ce aceasta a trimis acest semnal de confirmare (acknowledge), liniile de cerere şi acceptare pot fi dezactivate. Ca rezultat, alte componente pot cere accesul asupra magistralei în timp ce prima componentă foloseşte magistrala. Până când transferul curent este terminat, următorul master de magistrală va fi deja selectat şi poate porni imediat ce linia de confirmare a fost dezactivată. Este momentul în care următoarea rundă de arbitrare poate începe. Această schema impune încă o linie de magistrală şi mai multă capacitate logică pentru fiecare componentă, dar foloseşte mai bine ciclurile magistralei.

În sistemele în care memoria este pe magistrala principală, CPU-ul trebuie să concureze împreună cu celelalte componente I/O pentru magistrală la aproape fiecare ciclu. O soluție comună pentru această situație este alocarea celei mai mici priorități pentru CPU, astfel el primeşte acces la magistrală numai când nimeni altcineva nu cere accesul. Ideea aici este că CPU-ul poate aştepta oricând, dar componentele I/O trebuie să primească controlul magistralei rapid, altfel pierzându-se datele. Discurile rotindu-se la viteze mari nu pot aştepta. Această problemă este evitată în multe sisteme moderne punând memoria pe o magistrală separată de celelalte componente I/O, pentru a nu mai concura pentru accesul la magistrală.

Arbitrajul **descentralizat** al magistralei este, de asemenea, posibil.

De exemplu, un computer ar putea avea 16 linii de cerere pentru priorități. Când o componentă vrea să folosească magistrala, trimite un semnal de cerere. Toate componentele monitorizează toate liniile de cerere, astfel ca la sfârşitul fiecărui ciclu fiecare componentă ştie dacă a fost aceea cu cea mai mare prioritate astfel ştiind dacă îi este permis să folosească magistrala în timpul următorului ciclu. În comparație cu arbitrajul centralizat, această metodă de arbitrare are nevoie de mai multe linii de magistrală, dar evită potențialul cost al arbitrului. De asemenea, limitează numărul de componente la numărul de linii de cerere.

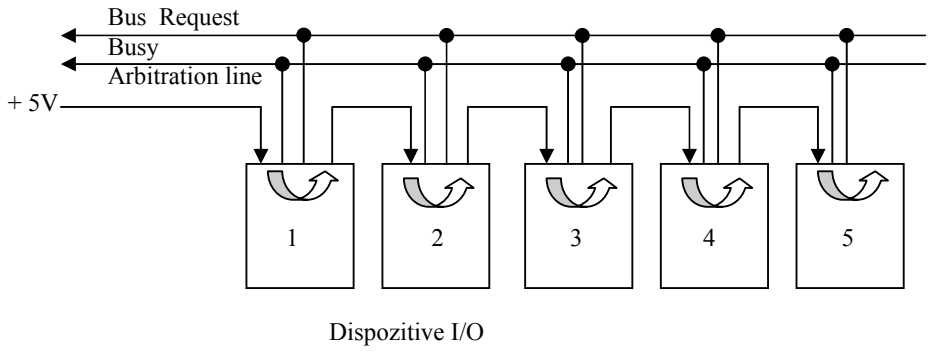

Figura 4.15 Arbitrarea descentralizată a magistralei.

Alt tip de arbitrare descentralizată a magistralei, prezentată în figura 4.15, foloseşte doar 3 linii, oricâte componente ar fi prezente. Prima linie de magistrală este o linie SAU - cablată pentru cereri la magistrală. A doua linie este numită BUSY şi este activată de masterul de magistrală curent. A treia linie este folosită pentru a arbitra magistrala. Ea este legată de toate componentele folosind metoda daisy chain. Capul acestui lanț este ținut în regim de activare legându-l la sursa de energie de 5 volți, de exemplu.

Când nici o componentă nu doreşte accesul la magistrală, linia de arbitrare este propagată la toate componentele. Pentru a primi controlul la magistrală, o componentă verifică întâi dacă magistrala nu este ocupată şi semnalul de control pe care îl primeşte, pe IN, este activ. Dacă IN este dezactivat, nu poate deveni master pe magistrală şi dezactivează OUT. Dacă IN este activ, componenta dezactivează OUT, ceea ce cauzează ca vecinul să vadă IN inactiv şi să furnizeze OUT inactiv. Astfel, vecinii din amonte văd IN inactiv şi inactivează OUT. Când se termină procesul, numai o componentă va primi IN activ şi va dezactiva OUT. Această componentă devine master de magistrală, transmite BUSY şi OUT, şi începe transferul. Simpla analiză a problemei va scoate la lumină faptul că, componenta cea mai din stânga care vrea accesul la magistrală îl primeşte. Astfel, această schemă este similară cu cea a înlănțuirii daisy chain, exceptând prezența arbitrului, astfel este mai ieftina, mai rapida şi nu poate fi afectată de defectarea arbitrului.