3. MAGISTRALA SYSTEM MANAGEMENT BUS

Această lucrare de laborator prezintă magistrala *System Management Bus* (SMBus). După o prezentare generală a magistralei SMBus, sunt descrise transferurile de biți și de date, este prezentată procedura de arbitrare a magistralei, sunt subliniate deosebirile dintre magistrala SMBus și magistrala I^2C și sunt detaliate o parte a protocoalelor de comandă. În continuare este prezentat controlerul SMBus Intel, incluzând registrele și comenzile sale, ca și utilizarea sa cu dispozitive I^2C. Aplicațiile au ca scop detectarea dispozitivelor conectate la magistrala SMBus a calculatorului, citirea conținutului memoriilor SPD din sistem și decodificarea conținutului acestor memorii.

3.1. Prezentare generală a magistralei SMBus

Magistrala *System Management Bus* (SMBus) este o magistrală serială simplă cu doar două linii de semnal. Această magistrală se poate utiliza pentru comunicația dintre diferite dispozitive de sistem și dintre aceste dispozitive și restul sistemului. Principiile de funcționare ale magistralei SMBus sunt similare cu cele ale magistralei I^2C. Există însă mai multe deosebiri între cele două magistrale, deosebiri care vor fi prezentate ulterior.

SMBus reprezintă o magistrală de control pentru operații de gestiune a sistemului și de gestiune a energiei. Un sistem poate utiliza magistrala SMBus pentru a transfera mesaje la și de la diferite dispozitive în locul utilizării unor linii de control individuale, ceea ce permite reducerea numărului de pini și a firelor de interconectare. Un dispozitiv poate utiliza magistrala SMBus pentru a furniza informații despre producător, pentru a furniza numărul modelului dispozitivului, pentru a raporta diferite tipuri de erori, pentru a primi parametri de control și pentru a returna starea dispozitivului.

Magistrala SMBus a fost propusă inițial de Intel ca o legătură între o baterie inteligentă, un încărcător al bateriei și un microcontroler care comunică cu restul sistemului. Specificațiile SMBus au fost actualizate de *System Management Interface Forum* (www.powersig.org). Versiunea curentă a specificațiilor SMBus este 3.2, publicată în 2022. Deși primele versiuni ale magistralei SMBus au fost elaborate în primul rând pentru baterii inteligente, versiunile mai recente permit conectarea unei varietăți largi de dispozitive, cuprinzând senzori ai sistemului, circuite de memorie și dispozitive de comunicație.

La magistrala SMBus se pot conecta două tipuri de dispozitive, *master* și *slave*. Fiecare dispozitiv conectat la magistrală este adresabil printr-o adresă unică. În general, un dispozitiv *master* inițiază un transfer pe magistrală între acesta și un singur dispozitiv *slave*, furnizând semnalul de ceas necesar pentru transfer. Un caz de excepție apare în timpul setării inițiale a magistralei, când un singur dispozitiv *master* poate iniția simultan tranzacții cu dispozitive *slave* multiple. Un dispozitiv *slave* poate recepționa date transmise de către dispozitivul *master* sau poate furniza date la dispozitivul *master*.

Similar cu magistrala I^2C, SMBus este o magistrală multi-*master*, permițând conectarea la magistrală a mai multor dispozitive *master*. Totuși, un singur dispozitiv poate controla magistrala la un moment dat. Deoarece mai multe dispozitive *master* pot încerca să preia controlul asupra magistralei, magistrala SMBus pune la dispoziție un mecanism de arbitrare care se bazează pe conexiunea ȘI cablat a tuturor dispozitivelor la magistrală.

Dispozitivele SMBus pot fi alimentate de sursa de alimentare a magistralei sau de o altă sursă de alimentare (ca în cazul bateriilor inteligente). În funcție de versiunea SMBus,

tensiunea nominală a sursei de alimentare a magistralei, V_{DD}, este de 5 V (minim 4,5 V), 3 V (minim 2,7 V), sau 1,8 V (minim 1,62 V). Figura 3.1 ilustrează un exemplu de implementare a unei magistrale SMBus de 5 V cu un dispozitiv alimentat de sursa de alimentare a magistralei. În același timp, există un alt dispozitiv conectat la liniile magistralei SMBus, care este alimentat de o sursă de 3 V. Aceste dispozitive vor putea comunica între ele cât timp ele aderă la specificațiile electrice ale magistralei SMBus.

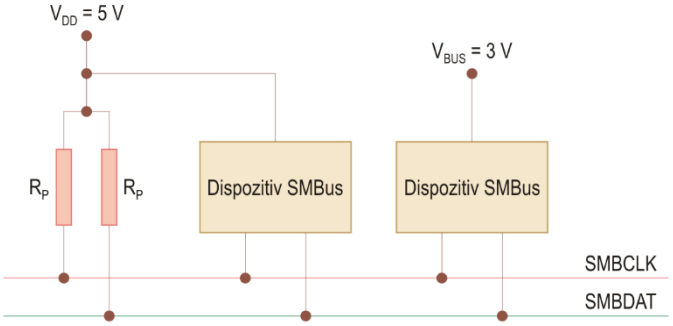

Figura 3.1. Exemplu de topologie a magistralei SMBus.

Linia de ceas SMBCLK și linia de date SMBDAT sunt ambele bidirecționale, conectate la tensiunea de alimentare prin rezistențele R_P; liniile pot fi conectate și printr-o sursă de curent. Etajele de ieșire ale dispozitivelor conectate la magistrală trebuie să fie cu drenă deschisă sau cu colector deschis pentru a realiza funcția ȘI cablat. Atunci când magistrala este liberă, ambele linii sunt în starea logică 1. Un dispozitiv poate plasa o linie a magistralei în starea logică 0 prin aducerea liniei la nivelul definit de tensiune joasă (maxim 0,4 V). Atunci când dispozitivul eliberează o linie a magistralei, aceasta va fi adusă în starea logică 1 de către rezistența R_P.

Versiunile 1.0 și 1.1 ale specificațiilor SMBus, elaborate în special pentru baterii inteligente, au definit numai caracteristici electrice cu putere redusă. Aceste caracteristici sunt potrivite pentru sisteme la care conservarea energiei este cel mai important aspect. Versiunea 2.0 a specificațiilor SMBus a introdus o nouă clasă de caracteristici electrice cu putere mai ridicată. Aceste caracteristici asigură robustețea necesară, de exemplu, pentru a permite ca magistrala SMBus să traverseze un conector PCIe, permițând ca dispozitivele SMBus de pe plăcile de extensie PCIe să comunice cu alte dispozitive SMBus de pe placa sistem și de pe alte plăci de extensie PCIe. Această versiune a redus și tensiunea nominală V_{DD} de la 5 V la 3 V. Toate aceste versiuni specifică o frecvență a semnalului de ceas între 10 KHz și 100 KHz; cele mai multe implementări curente utilizează însă o frecvență a semnalului de ceas de la 50 KHz la 100 KHz.

Versiunea 3.0 a specificațiilor SMBus a redus tensiunea nominală V_{DD} de la 3 V la 1,8 V. Această versiune a adăugat și noi protocoale de date pentru citirea sau scrierea valorilor de 32 sau 64 de biți și a introdus funcționarea la frecvențele mai ridicate ale semnalului de ceas de 400 KHz și 1 MHz.

Pentru dispozitivele conforme cu caracteristicile electrice cu putere redusă, specificațiile SMBus indică un curent minim de 100 µA și un curent maxim de 350 µA prin rezistența R_P. Astfel, valoarea minimă a rezistenței R_P trebuie să fie de 14,28 KΩ. Pentru dispozitivele conforme cu caracteristicile electrice cu putere mai ridicată, specificațiile indică o cerință a curentului minim absorbit de 4 mA, ceea ce determină o valoare minimă a rezistenței R_P de 1,25 KΩ.

O facilitate opțională a magistralei SMBus este mecanismul de verificare a erorilor pachetelor (*Packet Error Checking*), care permite îmbunătățirea fiabilității și a robusteții comunicației. Această facilitate a fost introdusă în versiunea 1.1 a specificațiilor SMBus. Este implementată prin adăugarea unui cod de eroare al pachetului (*Packet Error Code* – PEC) la sfârșitul fiecărui mesaj. Codul PEC este calculat din toți octeții unui mesaj prin utilizarea unui cod ciclic redundant de 8 biți (CRC-8). Octetul PEC este adăugat la mesaj de către dispozitivul care a furnizat ultimul octet de date.

3.2. Transferuri de biți și de date

3.2.1. Transferuri de biți

Pentru ca datele să fie valide, linia SMBDAT trebuie să fie stabilă în perioada în care linia SMBCLK este în starea logică 1. Linia de date își poate modifica starea numai atunci când linia SMBCLK este în starea logică 0. Această cerință este ilustrată în figura 3.2.

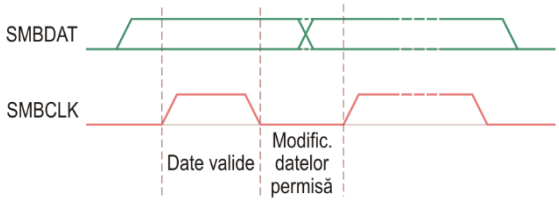

Figura 3.2. Validitatea datelor pe magistrala SMBus.

Fiecare transfer începe cu o condiție de START și se termină cu o condiție de STOP. Aceste condiții sunt generate întotdeauna de către dispozitivul *master*. Condițiile de START și de STOP sunt ilustrate în figura 3.3 și sunt definite mai jos.

- O condiție de START este definită printr-o tranziție din starea logică 1 în starea logică 0 a liniei SMBDAT în timp ce linia SMBCLK este în starea logică 1.
- O condiție de STOP este definită printr-o tranziție din starea logică 0 în starea logică 1 a liniei SMBDAT în timp ce linia SMBCLK este în starea logică 1.

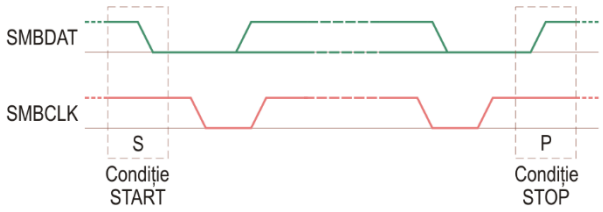

Figura 3.3. Condiții de START și de STOP pe magistrala SMBus.

După o condiție de START, magistrala este considerată ocupată. Magistrala devine liberă din nou după un anumit timp urmând condiției de STOP sau după ce liniile SMBCLK și SMBDAT rămân ambele în starea logică 1 pentru mai mult de 50 µs.

3.2.2. Transferuri de date

Pe magistrala SMBus, octeții sunt transferați începând cu bitul cel mai semnificativ (c.m.s.). Fiecare octet transferat pe magistrală trebuie urmat de un bit de confirmare (ACK). Impulsul de ceas corespunzător bitului ACK este generat de către dispozitivul *master*. Dispozitivul transmițător, *master* sau *slave*, eliberează linia SMBDAT în timpul ciclului de ceas ACK. Pentru confirmarea unui octet, receptorul trebuie să aducă linia SMBDAT în starea logică 0 în timpul ciclului de ceas ACK. Un exemplu de transfer de date este ilustrat în figura 3.4.

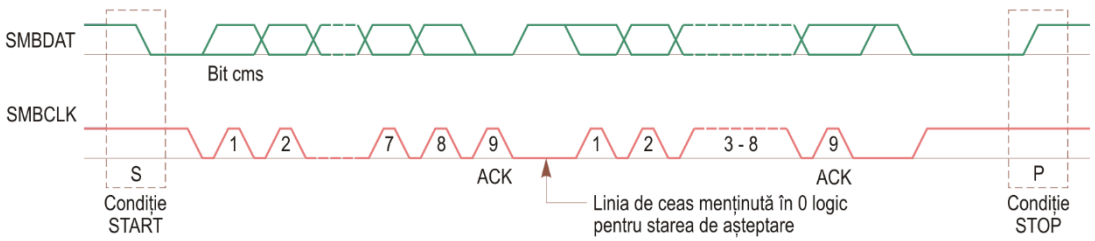

Figura 3.4. Exemplu de transfer de date pe magistrala SMBus.

Observație

- Un dispozitiv conectat la magistrala SMBus trebuie să confirme întotdeauna propria adresă. Această cerință este utilizată de controlerul SMBus pentru a detecta prezența unui dispozitiv detașabil pe magistrală.

Este posibil ca un receptor să nu confirme un octet de date și să transmită un bit NACK. Pentru aceasta, receptorul trebuie să mențină linia SMBDAT în starea logică 1 în timpul ciclului de ceas ACK. Un dispozitiv *slave* poate decide să nu confirme un octet, altul decât octetul de adresă, în următoarele situații:

- Dispozitivul *slave* este ocupat cu execuția unei operații în timp real sau datele solicitate nu sunt disponibile. Atunci când dispozitivul *master* detectează un bit NACK, trebuie să genereze o condiție de STOP pentru a abandona transferul. Ca o alternativă, dispozitivul *slave* poate extinde starea logică 0 a semnalului de ceas (în modul ilustrat în figura 3.4) pentru un timp de până la 25 ms pentru a-și termina operațiile.
- Dispozitivul *slave* detectează o comandă invalidă sau date invalide. În acest caz, dispozitivul *master* trebuie să genereze o condiție de STOP și să reîncerce execuția tranzacției.
- Dacă un dispozitiv *master* receptor este implicat într-o tranzacție, acesta trebuie să semnaleze dispozitivului *slave* transmițător să nu mai trimită date prin transmiterea unui bit NACK după ultimul octet transmis de dispozitivul *slave*. Dispozitivul *slave* transmițător trebuie să elibereze linia de date pentru a permite dispozitivului *master* să genereze o condiție de STOP.

Transferurile de date pe magistrala SMBus au următorul format. După condiția de START, dispozitivul *master* depune pe magistrală adresa de 7 biți a dispozitivului *slave* pe care dorește să îl adreseze. Adresa de 7 biți este urmată de un al optulea bit (R/W#) indicând direcția transferului de date; dacă R/W# este 0, transferul este de scriere, iar dacă R/W# este 1, transferul este de citire. Un transfer de date este terminat cu o condiție de STOP generată de dispozitivul *master*.

Anumite protocoale SMBus necesită ca dispozitivul *master* să genereze o condiție de START repetată urmată de adresa dispozitivului *slave* fără să genereze mai întâi o condiție de STOP.

3.3. Arbitrajul de magistrală

Un dispozitiv *master* poate începe un transfer numai dacă magistrala este în starea inactivă. Unul sau mai multe dispozitive pot genera o condiție de START în același timp. Deoarece este posibil ca dispozitivele care au generat condiția de START să nu detecteze faptul că alte dispozitive *master* solicită utilizarea magistralei, se utilizează o procedură de arbitraj. Arbitrajul se realizează cu ajutorul liniei SMBDAT în timpul în care linia SMBCLK este în starea logică 1. Un dispozitiv *master* care transmite un nivel logic 1 pe linia SMBDAT în timp ce alte dispozitive *master* transmit un nivel logic 0 pe această linie pierde controlul asupra magistralei în ciclul de arbitrare.

Dispozitivul *master* care a pierdut arbitrajul poate continua să furnizeze impulsuri de ceas până la terminarea transmisiei sau recepției octetului la care a pierdut arbitrajul. Dacă arbitrajul are loc între două dispozitive *master* și primul dispozitiv *master* dorește generarea unei condiții de START repetate, în timp ce al doilea dispozitiv *master* dorește plasarea unui bit de 0 pe magistrală, primul dispozitiv *master* va recunoaște faptul că nu poate genera condiția de START și va pierde arbitrajul. Dacă primul dispozitiv *master* dorește generarea unei condiții de START repetate, în timp ce al doilea dispozitiv *master* dorește plasarea unui bit de 1 pe magistrală, al doilea dispozitiv *master* va detecta condiția de START repetată și va pierde arbitrajul. Dacă ambele dispozitive *master* doresc să genereze o condiție de START repetată în aceeași poziție de bit, arbitrajul va continua pe fiecare bit de date.

Acest mecanism de arbitraj necesită ca dispozitivele *master* care participă într-un ciclu de arbitrare să monitorizeze starea liniei SMBDAT în timpul arbitrării. Odată ce un dispozitiv *master* a câștigat arbitrarea, arbitrajul este invalidat până când magistrala va fi din nou în starea inactivă.

3.4. Protocoale de comenzi

Un dispozitiv SMBus tipic are un set de comenzi prin care datele sale pot fi citite sau scrise. Fiecare comandă este transmisă unui dispozitiv utilizând un protocol specific de comandă care este descris de specificațiile SMBus. Versiunea 3.0 a acestor specificații definește un număr de 15 protocoale de comenzi. Dispozitivele nu trebuie să recunoască toate protocoalele definite în specificațiile SMBus.

Protocoalele tipice de comenzi conțin un cod de comandă de 8 biți. Argumentele comenzilor și valorile returnate de acestea pot varia în lungime. Fiecare protocol de comandă (cu excepția protocolului *Quick Command*) are două variante: una cu octetul *Packet Error Code* (PEC) și una fără octetul PEC.

Protocoalele de comenzi pot avea unul din următoarele formate generale:

- Un dispozitiv *master* transmițător transmite date la un dispozitiv *slave* receptor. Direcția transferului nu este schimbată în acest caz.
- Un dispozitiv *master* citește date de la un dispozitiv *slave* imediat după primul octet. La primul bit de confirmare (furnizat de dispozitivul *slave* receptor), dispozitivul *master* transmițător devine un dispozitiv *master* receptor (iar dispozitivul *slave* receptor devine un dispozitiv *slave* transmițător).
- În cazul formatului combinat, în timpul schimbării direcției în cadrul unui transfer, dispozitivul *master* generează o condiție de START repetată și transmite octetul cu adresa dispozitivului *slave* cu bitul R/W# setat la 1. Dispozitivul *master* receptor termină transferul prin transmiterea unui bit NACK după ultimul octet al transferului și apoi prin generarea unei condiții de STOP.

În această secțiune sunt descrise numai o parte a protocoalelor de comenzi SMBus. Pentru simplificarea prezentării, variantele protocoalelor de comenzi cu octetul PEC nu sunt descrise. Se utilizează următoarele simboluri pentru descrierea protocoalelor de comenzi:

- S: O condiție de START generată de un dispozitiv *master*.
- Sr: O condiție de START repetată generată de un dispozitiv *master*.
- P: O condiție de STOP generată de un dispozitiv *master*.
- Wr: Indică un transfer de scriere; bitul 0 (R/W#) al octetului de adresă va fi 0.
- Rd: Indică un transfer de citire; bitul 0 (R/W#) al octetului de adresă va fi 1.
- A: Un bit ACK transmis de la un dispozitiv *master* sau un dispozitiv *slave*.
- N: Un bit NACK transmis de la un dispozitiv *master* sau un dispozitiv *slave*.

Observații

- Un număr deasupra unui câmp de date reprezintă lungimea în biți a acelui câmp.
- Condițiile de START și de STOP sunt tranziții și nu biți, fiind reprezentate fără un contor de biți deasupra simbolurilor lor. Condiția de START repetată este, de asemenea, o tranziție și nu un bit.
- Câmpurile conținând informații transmise de la un dispozitiv *slave* la un dispozitiv *master* sunt reprezentate cu o culoare mai închisă.

3.4.1. Protocolul *Quick Command*

În cazul protocolului *Quick Command* (figura 3.5), bitul R/W# al adresei dispozitivului *slave* reprezintă comanda. Acest bit se poate utiliza pentru a valida/invalida o funcție a

dispozitivului sau pentru a valida/invalida un mod de funcționare cu putere consumată redusă. Nu sunt date transmise sau recepționate.

Figura 3.5. Protocolul *Quick Command*.

3.4.2. Protocolul *Send Byte*

Protocolul *Send Byte* (figura 3.6) se poate utiliza pentru a transmite o comandă codificată pe un octet la un dispozitiv *slave*. Octetul codului de comandă urmează după adresa dispozitivului *slave*. Întregul octet sau o parte a octetului transmis poate contribui la comandă. De exemplu, cei 7 biți mai semnificativi ai octetului pot specifica o acțiune de executat, cum ar fi trecerea dispozitivului într-un anumit mod operațional, în timp ce bitul cel mai puțin semnificativ poate specifica validarea sau invalidarea unei funcții a dispozitivului.

Figura 3.6. Protocolul *Send Byte*.

Observație

- Interpretarea codului de comandă este specifică dispozitivului. Specificațiile SMBus nu definesc o listă cu coduri de comandă pentru a fi utilizate cu toate dispozitivele SMBus.

3.4.3. Protocolul *Receive Byte*

Protocolul *Receive Byte* (figura 3.7) este similar cu protocolul *Send Byte*; singura deosebire este direcția transferului, care este de la dispozitivul *slave* la dispozitivul *master*. Acest protocol poate fi utilizat pentru citirea informațiilor de la un dispozitiv *slave*. Un bit NACK indică sfârșitul transferului de citire.

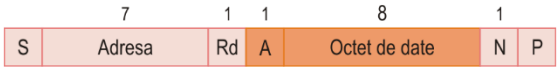

Figura 3.7. Protocolul *Receive Byte*.

3.4.4. Protocoalele *Write Byte* și *Write Word*

Protocoalele *Write Byte* și *Write Word* (figura 3.8) permit transmiterea unui octet sau a doi octeți de date la un dispozitiv *slave*. Dispozitivul *master* transmite adresa dispozitivului slave urmată de un bit de scriere Wr. Dispozitivul *slave* returnează un bit ACK, iar dispozitivul *master* transmite codul de comandă. Dispozitivul *slave* returnează din nou un bit ACK înainte ca dispozitivul *master* să transmită octetul sau cuvântul de date (primul fiind octetul inferior). Dispozitivul *slave* confirmă fiecare octet, iar întregul transfer este terminat cu o condiție de STOP.

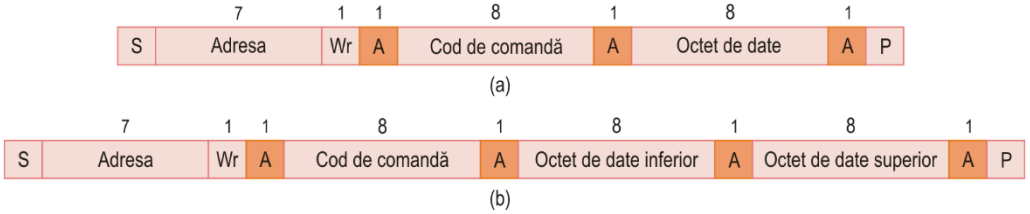

Figura 3.8. (*a*) Protocolul *Write Byte*; (*b*) Protocolul *Write Word*.

3.4.5. Protocoalele *Read Byte* și *Read Word*

Protocoalele *Read Byte* și *Read Word* (figura 3.9) permit citirea unui octet sau a doi octeți de date de la un dispozitiv *slave*. Dispozitivul *master* trebuie să transmită mai întâi o comandă la dispozitivul *slave*. Apoi trebuie să urmeze comanda cu o condiție de START repetată pentru a indica un transfer de citire de la dispozitivul adresat. Dispozitivul *slave* returnează apoi unul sau doi octeți de date. Nu există o condiție de STOP înaintea condiției de START repetate. Bitul NACK indică sfârșitul transferului de citire.

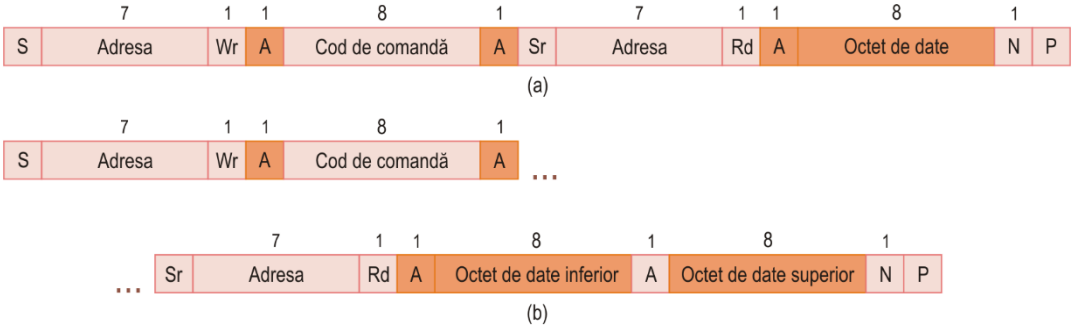

Figura 3.9. (*a*) Protocolul *Read Byte*; (*b*) Protocolul *Read Word*.

3.4.6. Protocolul *Process Call*

În cazul protocolului *Process Call* (figura 3.10), dispozitivul *master* transmite un cod de comandă urmat de doi octeți de date, iar apoi așteaptă ca dispozitivul *slave* să returneze o valoare de doi octeți dependentă de datele transmise. Acest protocol este o combinație a protocolului *Write Word* urmat de protocolul *Read Word* fără codul de comandă al protocolului *Read Word* și condiția de STOP a protocolului *Write Word*. Nu există o condiție de STOP înaintea condiției de START repetate. Bitul NACK indică sfârșitul transferului.

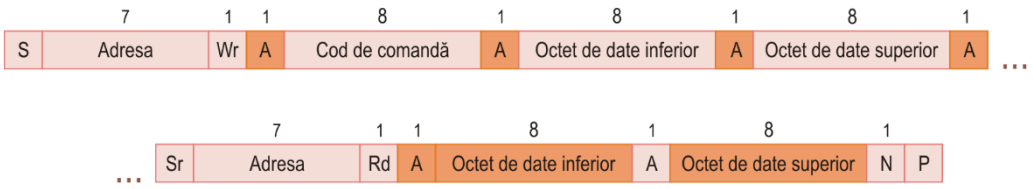

Figura 3.10. Protocolul *Process Call*.

3.4.7. Protocoalele *Block Write* și *Block Read*

În cazul protocolului *Block Write* (figura 3.11 (*a*)), dispozitivul *master* transmite mai întâi adresa unui dispozitiv *slave* și un bit de scriere, urmat de un cod de comandă. Apoi, dispozitivul *master* transmite un contor de octeți care indică numărul octeților de date care vor urma în cadrul mesajului. Contorul de octeți poate fi zero. Acest protocol permite transferul unui număr maxim de 255 octeți de date.

Protocolul *Block Read* (figura 3.11 (*b*)) diferă de protocolul *Block Write* prin faptul că este inserată o condiție de START repetată pentru a permite schimbarea direcției de transfer. Un bit NACK precedând imediat condiția de STOP indică sfârșitul transferului de citire.

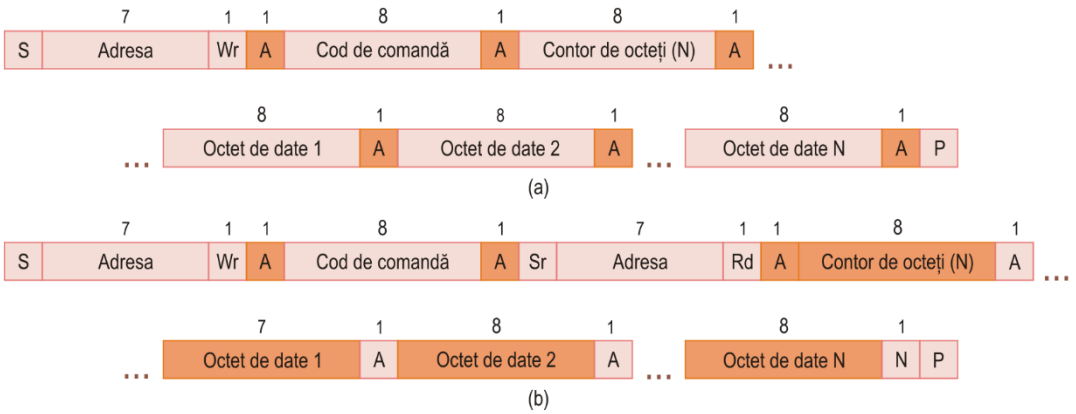

Figura 3.11. (*a*) Protocolul *Block Write*; (*b*) Protocolul *Block Read*.

3.4.8. Protocolul *Block Write-Block Read Process Call*

Protocolul *Block Write-Block Read Process Call* (figura 3.12) are două părți. În prima parte a protocolului, dispozitivul *master* transmite adresa dispozitivului *slave*, un bit de scriere, codul de comandă și un contor de octeți de scriere (M) care specifică numărul de octeți care urmează a fi transmiși în prima parte a protocolului. Contorul de octeți de scriere poate fi zero. În a doua parte a protocolului, dispozitivul *master* generează o condiție de START repetată, iar apoi transmite adresa dispozitivului *slave* și un bit de citire. Dispozitivul *slave* va transmite un contor de octeți de citire (N) și un număr de N octeți de date. Contorul de octeți de citire poate fi diferit de contorul de octeți de scriere și poate fi zero. Contorul de octeți combinat (M + N) nu trebuie să depășească 255. De notat că nu există o condiție de STOP înaintea condiției de START repetate.

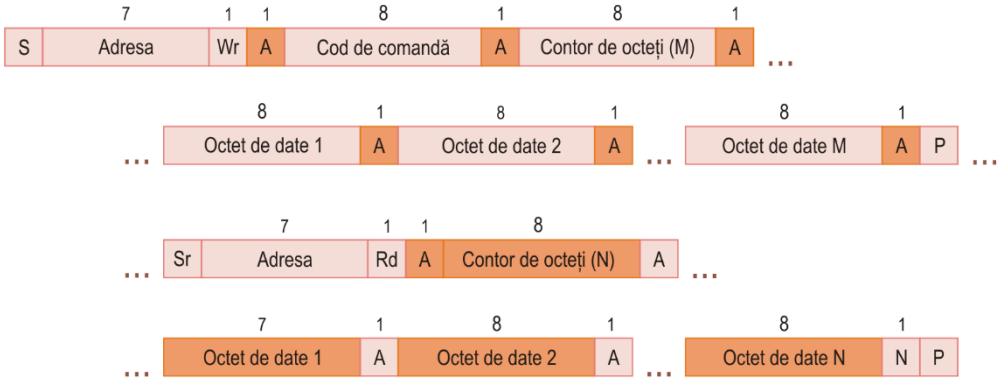

Figura 3.12. Protocolul *Block Write-Block Read Process Call*.

3.5. Deosebiri între magistralele SMBus și I²C

Magistrala SMBus provine din magistrala I²C, astfel încât cele două magistrale sunt foarte similare. Totuși, există mai multe deosebiri între aceste magistrale în ceea ce privește caracteristicile electrice, sincronizarea, modurile de funcționare și protocoalele. Cele mai importante deosebiri sunt descrise în continuare.

- Valorile minime și maxime ale tensiunii de alimentare V_{DD} sunt specificate în mod diferit pentru magistralele SMBus și I²C. Specificațiile SMBus constrâng tensiunile de alimentare nominale ale dispozitivelor atașate la magistrală la un minim de 1,8 V și un maxim de 5 V. Specificațiile I²C sunt mai tolerante în privința valorilor tensiunii de alimentare.
- Specificațiile I²C și SMBus definesc în mod diferit tensiunile de intrare de nivel scăzut și de nivel înalt pentru liniile magistralei. Specificațiile I²C definesc tensiunea de

intrare de nivel scăzut (V_{IL}) ca fiind 30% din tensiunea V_{DD}, iar tensiunea de intrare de nivel înalt (V_{IH}) ca fiind 70% din tensiunea V_{DD}. Specificațiile SMBus definesc praguri fixe pentru aceste tensiuni; V_{IL} este definit ca maxim 0,8 V și V_{IH} ca minim 1,35 V. Totuși, chiar și în condițiile specificațiilor diferite pentru pragurile tensiunii de intrare, în general, dispozitivele I^2C și SMBus vor fi interoperabile în gama tensiunilor de alimentare permise de specificațiile SMBus.

- Specificațiile SMBus limitează la 10 ms durata maximă a timpului în care un dispozitiv *master* poate extinde starea logică 0 a semnalului de ceas în cadrul fiecărui octet al mesajului. De asemenea, există o limită de 25 ms a timpului total în care un dispozitiv *slave* poate extinde starea logică 0 a semnalului de ceas în cadrul fiecărui mesaj. O altă restricție asupra funcționării magistralei SMBus este intervalul de depășire a timpului (*timeout*) de 35 ms, după care se presupune că magistrala este blocată și toate dispozitivele atașate la magistrală trebuie să reseteze interfețele lor de I/E. Spre deosebire de specificațiile SMBus, specificațiile I^2C permit unui dispozitiv *master* sau *slave* să mențină linia de ceas în starea logică 0 pentru un timp nelimitat și nu este definit un interval de depășire a timpului.
- Specificațiile SMBus definesc o magistrală cu caracteristici electrice cu putere redusă pentru aplicații la care consumul de energie trebuie minimizat, ca în cazul sistemelor alimentate de la baterie. Magistrala I^2C nu are o specificație similară.
- Specificațiile SMBus indică o frecvență minimă de funcționare de 10 KHz, în timp ce specificațiile I^2C nu indică o frecvență minimă a magistralei.
- Specificațiile I^2C nu impun ca un dispozitiv *slave* să confirme întotdeauna propria adresă. În consecință, dacă un dispozitiv nu confirmă propria adresă, controlerul nu poate detecta dacă dispozitivul este ocupat, s-a defectat, sau a fost îndepărtat de pe magistrală. Pentru a elimina această confuzie, specificațiile SMBus impun ca un dispozitiv *slave* să confirme întotdeauna propria adresă.
- Specificațiile I^2C definesc doar trei tipuri de protocoale sau cicluri de magistrală: ciclu de scriere, ciclu de citire și ciclu cu format combinat (aceste cicluri sunt descrise în secțiunea 3.7.1). Specificațiile SMBus definesc un număr mai mare de protocoale de comenzi care se pot utiliza pentru comunicația cu dispozitivele. De aceea, nu toate dispozitivele I^2C vor recunoaște protocoalele de comenzi SMBus.
- O deosebire importantă între magistralele SMBus și I^2C este modul în care este controlat numărul de octeți transferați în timpul transferurilor de scriere și de citire pe blocuri. În timpul ciclurilor de scriere I^2C, dispozitivul *slave* este cel care determină dimensiunea transferului de date prin transmiterea unui bit NACK după ultimul octet de date acceptat. Însă, în timpul protocolului SMBus *Block Write* dispozitivul *master* determină dimensiunea transferului prin transmiterea contorului de octeți ca parte a protocolului; după ultimul octet transmis, dispozitivul *master* așteaptă o confirmare normală (bit ACK) de la dispozitivul *slave*. În timpul ciclurilor de citire I^2C, dispozitivul *master* este cel care controlează dimensiunea transferului. După ce recepționează ultimul octet de date, dispozitivul *master* transmite un bit NACK și generează o condiție de STOP, terminând ciclul. Însă, în timpul protocolului SMBus *Block Read* dispozitivul *slave* este cel care determină dimensiunea transferului. După ce returnează ultimul octet de date, dispozitivul *slave* transmite un bit NACK, iar apoi dispozitivul *master* generează o condiție de STOP, terminând transferul.
- Un dispozitiv *slave* I^2C transmite un bit NACK pentru a indica faptul că nu mai poate accepta octeți de date. Un dispozitiv *slave* SMBus transmite un bit NACK pentru a indica recepția unei comenzi sau a unei date invalide.

- În timpul mai multor protocoale SMBus, se transmite un octet al codului de comandă după adresa dispozitivului *slave*. Dispozitivele *slave* I^2C interpretează acest octet ca primul octet de date pentru scriere din blocul de date.
- Protocoalele SMBus pentru citirea datelor de la un dispozitiv utilizează, în general, o condiție de START repetată. Este posibil ca anumite dispozitive I^2C să nu recunoască o condiție de START repetată. O încercare de utilizare a unui protocol SMBus cu această condiție pentru citirea datelor de la un dispozitiv I^2C poate produce rezultate neașteptate.

3.6. Controlerul SMBus Intel

Componenta *Platform Controller Hub* (PCH) a seturilor de circuite Intel actuale conține un controler SMBus, care reprezintă dispozitivul PCIe 31, funcția 3 (D31:F3) sau funcția 4 (D31:F4). Facilitățile oferite de acest controler depind de seria setului de circuite. De exemplu, controlerul SMBus al setului de circuite Intel din seria 8 utilizat de calculatoarele din laborator permit o frecvență de funcționare de până la 100 KHz (cu viteze de transfer de până la 100 Kbiți/s) și se conformează specificațiilor SMBus cu versiunea 2.0. Acest controler permite comunicația și cu numeroase dispozitive care sunt compatibile cu magistrala I^2C.

3.6.1. Registrele controlerului SMBus

Controlerul SMBus Intel conține două categorii de registre: registre de configurație PCI și registre de I/E. Secțiunile următoare descriu cele mai importante registre din fiecare categorie.

3.6.1.1. Registre de configurație PCI

Registrele de configurație PCI ale controlerului SMBus Intel cuprind registrele antetului de configurație PCI și un singur registru specific dispozitivului PCI. Aceste registre pot fi accesate utilizând fie mecanismul de configurație compatibil PCI, fie mecanismul de configurație îmbunătățit PCIe. Registrele de configurație PCI sunt enumerate în tabelul 3.1. Fiecare intrare a tabelului conține deplasamentul unui registru de configurație PCI, mnemonica, numele și dimensiunea sa. Deplasamentul este relativ la adresa de bază a spațiului de configurație alocat pentru funcția 3 sau funcția 4 a dispozitivului PCIe 31.

Tabelul 3.1. Registrele de configurație PCI ale controlerului SMBus Intel.

Deplasament	Mnemonica	Nume registru	Dimensiune (biți)
0x00	VID	Vendor Identification	16
0x02	DID	Device Identification	16
0x04	PCICMD	PCI Command	16
0x06	PCISTS	PCI Status	16
0x08	RID	Revision Identification	8
0x09	PI	Programming Interface	8
0x0A	SCC	Sub-Class Code	8
0x0B	BCC	Base Class Code	8
0x10	SMBMBAR0	Memory Base Address Register 0	32
0x14	SMBMBAR1	Memory Base Address Register 1	32
0x20	SMB_BASE	SMBus Base Address	32
0x2C	SVID	Subsystem Vendor Identification	16
0x2E	SID	Subsystem Identification	16
0x3C	INT_LN	Interrupt Line	8
0x3D	INT_PN	Interrupt Pin	8
0x40	HOSTC	Host Configuration	8

Observație

- În fișierul antet SMBus.h, care va fi utilizat pentru aplicații, registrele de configurație PCI ale controlerului SMBus sunt definite într-o structură numită `SMBUS_CFG`.

Registre ale antetului de configurație PCI

Registrele antetului de configurație PCI ale controlerului SMBus (cu deplasamente între 0x00 și 0x3D) au aceleași funcții ca și registrele generale ale antetului de configurație PCI descrise în lucrarea de laborator *Magistrala PCI Express*. Registrul SMBMBAR0 conține pe pozițiile 31..8 partea inferioară a adresei de bază pentru registrele de I/E mapate în memorie, în timp ce registrul SMBMBAR1 conține pe pozițiile 31..0 partea superioară a adresei de bază pentru registrele de I/E mapate în memorie. Registrul SMB_BASE conține pe pozițiile 15..5 adresa de bază pentru registrele de I/E mapate în spațiul de I/E.

HOSTC – Registrul *Host Configuration*

Registrul HOSTC este specific controlerului SMBus. Biții acestui registru sunt descriși în continuare.

- Biții 7..5 sunt rezervați.
- Bitul 4 (SPD Write Disable): Dacă este setat, scrierea la adresele SMBus între 0x50 .. 0x57 este invalidată. Acest domeniu de adrese este utilizat de memoriile SPD (*Serial Presence Detect*); acestea sunt memorii EEPROM cu acces serial utilizate în modulele de memorie DIMM.
- Bitul 3 (Soft SMBus Reset): Dacă este setat, automatul de stare și circuitele controlerului SMBus sunt resetate.
- Bitul 2 (I2C_EN): Dacă este setat, anumite protocoale de comenzi ale controlerului SMBus sunt modificate pentru a putea comunica cu dispozitive I^2C. Efectele setării acestui bit sunt discutate în secțiunile 3.6.2.5 și 3.6.2.6.
- Bitul 1 (SMB_SMI_EN): Dacă este setat, orice sursă de întrerupere SMBus va fi rutată pentru a genera o întrerupere specială *System Management Interrupt* (SMI).
- Bitul 0 (HST_EN): Dacă este setat, controlerul SMB este validat pentru a executa comenzi.

3.6.1.2. Registre de I/E

Cele mai importante registre de I/E ale controlerului SMBus sunt enumerate în tabelul 3.2.

Tabelul 3.2. Registrele de I/E ale controlerului SMBus Intel.

Deplasament	Mnemonica	Nume registru	Dimensiune (biți)
0x00	HST_STS	Host Status	8
0x02	HST_CNT	Host Control	8
0x03	HST_CMD	Host Command	8
0x04	XMIT_SLVA	Transmit Slave Address	8
0x05	HST_D0	Host Data 0	8
0x06	HST_D1	Host Data 1	8
0x07	Host_BLOCK_dB	Host Block Data Byte	8
0x08	PEC	Packet Error Check	8
0x0C	AUX_STS	Auxiliary Status	8
0x0D	AUX_CTL	Auxiliary Control	8

Observații

- Registrele de I/E ale controlerului SMBus pot fi accesate fie ca registre mapate în memorie utilizând conținutul registrelor SMBMBAR0 și SMBMBAR1 ca adresă de bază, fie ca registre mapate în spațiul de I/E utilizând conținutul registrului SMB_BASE ca adresă de bază. Deplasamentele sunt aceleași atât pentru registrele mapate în memorie cât și pentru registrele mapate în spațiul de I/E.

- În fișierul antet SMBus.h registrele de I/E mapate în memorie ale controlerului SMBus sunt definite într-o structură numită `SMBUS_REG`.

HST_STS – Registrul *Host Status*

Registrul HST_STS conține biți de stare setați de controlerul SMBus. Un bit de stare poate fi șters de program prin scrierea valorii 1 în poziția bitului respectiv. Scrierea valorii 0 în oricare poziție de bit nu are efect. Biții acestui registru sunt descriși în continuare.

- Bitul 7 (BYTE_DONE_STS): Dacă este setat, controlerul a recepționat un octet (în cazul unei comenzi de citire pe bloc) sau a terminat transmisia unui octet (în cazul unei comenzi de scriere pe bloc) atunci când bufferul de 32 de octeți nu este validat. Acest bit nu are semnificație pentru transferurile pe blocuri atunci când bufferul de 32 de octeți este validat.
- Bitul 6 (INUSE_STS): Dacă este setat, controlerul SMBus este în uz de către un fir de execuție software. Acest bit poate fi utilizat ca semafor de către diferitele fire de execuție software care trebuie să utilizeze controlerul SMBus. Programul poate testa starea acestui bit până când acesta devine 0, după care poate utiliza controlerul. Citirile ulterioare ale acestui bit vor returna 1. Scrierea acestui bit cu valoarea 1 va reseta valoarea următoare citită la 0.
- Bitul 5 (SMBALERT_STS): Dacă este setat, sursa unei întreruperi a fost semnalul *SMBALERT#*. Acesta este un semnal de întrerupere opțional al magistralei SMBus care poate fi utilizat de către un dispozitiv *slave* pentru a semnala controlerului că are un mesaj de transmis. Utilizarea acestui semnal nu este descrisă în această lucrare de laborator.
- Bitul 4 (FAILED): Dacă este setat, sursa unei întreruperi a fost o tranzacție eșuată pe magistrală. Acest bit este setat ca răspuns la setarea bitului KILL al registrului *Host Control* pentru a termina o tranzacție.
- Bitul 3 (BUS_ERR): Dacă este setat, sursa unei întreruperi a fost o coliziune în timpul execuției unei tranzacții.
- Bitul 2 (DEV_ERR): Dacă este setat, sursa unei întreruperi a fost fie un câmp invalid al comenzii, fie o eroare de depășire a timpului (*timeout*) generată de controler.
- Bitul 1 (INTR): Dacă este setat, sursa unei întreruperi a fost terminarea cu succes a ultimei comenzi. Bitul INTR nu este dependent de starea bitului INTREN al registrului *Host Control*. Dacă bitul INTREN nu este setat, atunci bitul INTR va fi setat la terminarea cu succes a unei comenzi, deși întreruperea nu va fi generată.
- Bitul 0 (HOST_BUSY): Dacă este setat, indică faptul că o comandă a controlerului SMBus este în curs de execuție. În timp ce acest bit este setat, nu trebuie accesat niciun registru al controlerului, cu excepția registrului *Host Block Data Byte*. Acest bit poate fi utilizat pentru detectarea momentului în care controlerul a terminat execuția unei comenzi.

HST_CNT – Registrul *Host Control*

Registrul HST_CNT permite specificarea comenzii de executat de către controler, inițierea execuției comenzii specificate și validarea generării unei întreruperi atunci când comanda a fost terminată. Biții acestui registru sunt descriși în continuare.

- Bitul 7 (PEC_EN): Dacă este setat, controlerul SMBus va executa tranzacția pe magistrală cu adăugarea fazei *Packet Error Checking* (PEC). Pentru tranzacții de scriere, valoarea octetului PEC este transferată din registrul PEC. Pentru tranzacții de citire, octetul PEC este încărcat în registrul PEC. Acest bit trebuie setat înaintea operației de scriere în care este setat bitul START.

- Bitul 6 (START): Atunci când acest bit este setat, controlerul va începe execuția comenzii specificate în câmpul SMB_CMD al registrului *Host Control*. Toate registrele necesare pentru execuția comenzii trebuie setate înainte de setarea acestui bit.
- Bitul 5 (LAST_BYTE): Acest bit este utilizat pentru comenzile *Block Read* și *I^2C Read*. Programul setează acest bit pentru a indica faptul că următorul octet va fi ultimul octet care trebuie recepționat în blocul de date. Aceasta determină transmiterea de către controler a unui bit NACK (în locul unui bit ACK) după recepționarea ultimului octet.
- Biții 4..2 (SMB_CMD): Programul înscrie în acest câmp codul comenzii de executat de către controler. Codurile comenzilor permise și denumirile lor sunt enumerate mai jos; comenzile sunt descrise în secțiunea 3.6.2.

  000: *Quick Command*;
  001: *Send/Receive Byte*;
  010: *Write/Read Byte*;
  011: *Write/Read Word*;
  100: *Process Call*;
  101: *Block Write/Read*;
  110: *I^2C Read*;
  111: *Block Write-Block Read Process Call*.

- Bitul 1 (KILL): Dacă este setat, controlerul abandonează tranzacția curentă, setează bitul de stare FAILED și generează o întrerupere dacă întreruperile sunt validate. Odată setat, acest bit trebuie șters de program pentru a permite funcționarea normală a controlerului.
- Bitul 0 (INTREN): Dacă este setat, acest bit validează generarea unei întreruperi atunci când execuția unei comenzi este terminată.

HST_CMD – Registrul *Host Command*

Conținutul registrului *Host Command* este transmis de controler în câmpul *Command Code* al protocoalelor SMBus în timpul execuției mai multor comenzi.

XMIT_SLVA – Registrul *Transmit Slave Address*

Registrul *Transmit Slave Address* este înscris de program cu adresa dispozitivului *slave* și cu bitul de direcție. Conținutul acestui registru este transmis de controler în câmpul *Address* al oricărui protocol SMBus.

- Biții 7..1 (Address): Acest câmp conține adresa de 7 biți a dispozitivului *slave*.
- Bitul 0 (RW): Acest bit indică direcția transferului. Valoarea 0 indică un transfer de scriere, în timp ce valoarea 1 indică un transfer de citire.

HST_D0 – Registrul *Host Data 0*

Pentru comanda *Write Byte*, registrul *Host Data 0* este înscris de program cu octetul de date care trebuie transmis în câmpul *Octet de date* al protocolului SMBus. Pentru comanda *Write Word* și comanda *Process Call*, acest registru este înscris de program cu octetul de date care trebuie transmis în câmpul *Octet de date inferior* al protocolului SMBus. Pentru comenzile *Block Write* și *Block Write-Block Read Process Call*, acest registru este înscris de program cu numărul de octeți care trebuie transferați, număr care va fi transmis în câmpul *Contor de octeți* al protocolului SMBus.

Observație

- Pentru comenzile *Block Write* și *Block Write-Block Read Process Call*, registrul *Host Data 0* trebuie înscris cu o valoare între 1 și 32 pentru contorul de octeți. O valoare 0 sau mai mare decât 32 va conduce la un rezultat impredictibil, deoarece controlerul nu testează validitatea contorului de octeți.

În cazul comenzii *Read Byte*, după terminarea comenzii, registrul *Host Data 0* va conține octetul de date citit. În cazul comenzilor *Read Word* și *Process Call*, acest registru va conține octetul de date recepționat în câmpul *Octet de date inferior* al protocolului SMBus. În cazul comenzilor *Block Read* și *Block Write-Block Read Process Call*, acest registru va conține numărul de octeți care trebuie recepționați, număr care este recepționat în câmpul *Contor de octeți* al protocolului SMBus.

Observație

- În cazul comenzii *I^2C Read*, fiecare octet de date citit este memorat în registrul *Host Block Data Byte* și nu în registrul *Host Data 0*.

HST_D1 – Registrul *Host Data 1*

Pentru comenzile *Write Word* și *Process Call*, acest registru este înscris de program cu octetul de date care trebuie transmis în câmpul *Octet de date superior* al protocolului SMBus. În cazul comenzilor *Read Word* și *Process Call*, acest registru va conține octetul de date recepționat în câmpul *Octet de date superior* al protocolului SMBus. Alte comenzi nu utilizează acest registru.

Host_BLOCK_dB – Registrul *Host Block Data Byte*

Host Block Data Byte este fie un registru, fie un pointer într-un buffer de 32 de octeți, după cum bitul E32B a fost setat sau nu în registrul *Auxiliary Control*. Atunci când bitul E32B a fost șters, acesta este un registru conținând un octet de date care trebuie transmis cu un transfer de scriere pe bloc sau recepționat cu un transfer de citire pe bloc. Atunci când bitul E32B a fost setat, acest registru este utilizat ca un pointer pentru a accesa bufferul de 32 de octeți. Acest pointer este resetat la 0 prin citirea registrului *Host Control*. Pointerul este apoi incrementa automat la fiecare acces la acest registru.

Detalii despre utilizarea registrului *Host Block Data Byte* sunt prezentate în secțiunea 3.6.2.6.

PEC – Registrul *Packet Error Check*

Pentru o comandă de scriere, registrul *Packet Error Check* este înscris cu codul CRC de 8 biți reprezentând octetul PEC înainte de lansarea comenzii. Pentru o comandă de citire, octetul PEC este încărcat în acest registru de pe magistrală, iar apoi poate fi citit de program.

AUX_STS – Registrul *Auxiliary Status*

Funcția principală a registrului *Auxiliary Status* este de a semnala o eroare CRC într-un mesaj recepționat.

- Bitul 0 (CRC Error – CRCE): Dacă este setat, indică faptul că mesajul recepționat conține o eroare CRC. Atunci când acest bit este setat, și bitul DEV_ERR al registrului *Host Status* va fi setat.

AUX_CTL – Registrul *Auxiliary Control*

Registrul *Auxiliary Control* permite validarea sau invalidarea bufferului de 32 de octeți și validarea sau invalidarea adăugării codului CRC la un mesaj.

- Bitul 1 (Enable 32-Byte Buffer – E32B): Dacă este setat, registrul *Host Block Data Byte* este un pointer în bufferul de 32 de octeți, în loc de un singur registru de date.

Aceasta permite comenzilor pe blocuri să transmită sau să recepționeze până la 32 de octeți de date înaintea generării unei întreruperi de către controler.

- Bitul 0 (Automatically Append CRC – AAC): Dacă este setat, controlerul va adăuga în mod automat codul CRC la mesaj. Acest bit nu trebuie modificat în timpul tranzacțiilor pe magistrală.

3.6.2. Comenzile controlerului SMBus

Comenzile recunoscute de controlerul SMBus implementează protocoalele SMBus care au fost descrise în secțiunea 3.4. În plus, controlerul recunoaște comanda *I^2C Read*, care permite citirea datelor de la dispozitive compatibile I^2C care nu implementează protocoalele SMBus de citire pe blocuri.

Pentru lansarea unei anumite comenzi, programul trebuie să execute operațiile următoare:

1. Așteaptă până când controlerul SMBus termină o comandă precedentă, când acesta șterge bitul HOST_BUSY din registrul *Host Status*.
2. Setează registrele controlerului care sunt necesare pentru comanda respectivă; registrele care trebuie setate pentru fiecare comandă sunt prezentate în secțiunile 3.6.2.1 .. 3.6.2.8.
3. Înscrie în registrul *Host Control* un octet cu codul comenzii în câmpul SMB_CMD și bitul START setat la 1. Presupunând că nu se utilizează întreruperi, bitul 0 (INTREN) al acestui octet trebuie să fie 0.

După lansarea unei comenzi, se utilizează registrul *Host Status* pentru a determina evoluția comenzii. În cazul în care comanda se termină cu succes, bitul INTR va fi setat. Dacă dispozitivul nu răspunde cu un bit ACK și are loc depășirea timpului alocat tranzacției, va fi setat bitul DEV_ERR. Dacă programul setează bitul KILL în registrul *Host Control* în timpul execuției comenzii, tranzacția va fi abandonată și va fi setat bitul FAILED în registrul *Host Status*.

Presupunând că nu se utilizează întreruperi, programul trebuie să execute următoarele operații pentru a determina dacă o comandă lansată s-a terminat și pentru a testa modul de terminare al comenzii:

1. Așteaptă terminarea execuției comenzii, când controlerul setează fie bitul INTR din registrul *Host Status*, fie unul din următorii biți din același registru: FAILED, BUS_ERR sau DEV_ERR.
2. După terminarea comenzii, dacă bitul INTR este setat, comanda s-a terminat cu succes. Dacă unul din biții FAILED, BUS_ERR sau DEV_ERR este setat, comanda s-a terminat cu o eroare și programul poate semnala tipul erorii. În ambele cazuri, programul trebuie să șteargă biții de stare din registrul *Host Status* prin citirea conținutului registrului și rescrierea valorii citite în același registru.

3.6.2.1. Comanda *Quick Command*

Înaintea lansării unei comenzi *Quick Command*, programul trebuie să forțeze bitul I2C_EN la 0 în registrul *Host Configuration* și bitul PEC_EN la 0 în registrul *Host Control*. Registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu codul comenzii (0 sau 1) în bitul RW.

Observație

- Bitul PEC_EN din registrul *Host Control* trebuie înscris înaintea operației de scriere în care bitul START este setat în același registru.

3.6.2.2. Comanda *Send/Receive Byte*

Pentru o comandă *Send Byte*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu octetul de date care trebuie transmis.

Pentru o comandă *Receive Byte*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (1) în bitul RW. Octetul de date recepționat va fi memorat în registrul *Host Data 0*.

3.6.2.3. Comanda *Write/Read Byte*

Pentru o comandă *Write Byte*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii, iar registrul *Host Data 0* trebuie înscris cu octetul de date care trebuie transmis.

Pentru o comandă *Read Byte*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (1) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. Octetul de date citit va fi memorat în registrul *Host Data 0*.

3.6.2.4. Comanda *Write/Read Word*

Pentru o comandă *Write Word*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. Registrul *Host Data 0* trebuie înscris cu octetul de date mai puțin semnificativ care trebuie transmis, iar registrul *Host Data 1* trebuie înscris cu octetul de date mai semnificativ care trebuie transmis.

Pentru o comandă *Read Word*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (1) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. După terminarea comenzii, octetul de date mai puțin semnificativ citit va fi memorat în registrul *Host Data 0*, iar octetul de date mai semnificativ citit va fi memorat în registrul *Host Data 1*.

3.6.2.5. Comanda *Process Call*

Înaintea lansării unei comenzi *Process Call*, programul trebuie să forțeze fie bitul I2C_EN la 0 în registrul *Host Configuration*, fie bitul PEC_EN la 0 în registrul *Host Control*.

Observație

- Execuția unei comenzi *Process Call* cu biții I2C_EN și PEC_EN ambii setați la 1 produce rezultate nedefinite.

Pentru execuția acestei comenzi, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. Registrul *Host Data 0* trebuie înscris cu octetul de date mai puțin semnificativ care trebuie transmis, iar registrul *Host Data 1* trebuie înscris cu octetul de date mai semnificativ care trebuie transmis. După terminarea comenzii, octetul de date mai puțin semnificativ citit va fi memorat în registrul *Host Data 0*, iar octetul de date mai semnificativ citit va fi memorat în registrul *Host Data 1*.

Observație

- La funcționarea în modul I^2C, cu bitul I2C_EN din registrul *Host Configuration* setat, protocolul implementat se modifică prin faptul că nu se transmite codul comenzii ca parte a mesajului.

3.6.2.6. Comanda *Block Write/Read*

Înaintea lansării unei comenzi *Block Write*, programul trebuie fie să forțeze bitul I2C_EN la 0 în registrul *Host Configuration*, fie să forțeze bitul PEC_EN în registrul *Host Control* și bitul AAC în registrul *Auxiliary Control* la 0.

Comanda *Block Write/Read* poate utiliza bufferul de 32 de octeți al controlerului SMBus. Acest buffer poate fi validat prin setarea bitului E32B al registrului *Auxiliary Control*. Pentru o comandă de scriere, programul umple bufferul de 32 de octeți cu datele care trebuie transmise, iar pentru o comandă de citire, controlerul umple bufferul de 32 de octeți cu datele recepționate. Controlerul va genera o întrerupere numai după transmiterea sau recepția a 32 de octeți, sau atunci când contorul de octeți a ajuns la zero.

Observație

- La funcționarea în modul I²C, cu bitul I2C_EN din registrul *Host Configuration* setat, controlerul nu va utiliza bufferul de 32 de octeți pentru nicio comandă pe bloc.

Pentru o comandă *Block Write*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii, iar registrul *Host Data 0* trebuie înscris cu contorul de octeți indicând numărul de octeți care vor fi transmiși. Contorul de octeți nu poate fi 0. După cum bufferul de 32 de octeți a fost validat sau nu, octeții de date care trebuie transmiși vor fi înscriși în registrele controlerului în modul specificat în continuare.

- Dacă bufferul de 32 de octeți a fost validat, programul va scrie până la 32 de octeți în registrul *Host Block Data Byte*. Controlerul va transmite adresa dispozitivului *slave*, codul comenzii, contorul de octeți și toți octeții de date din bufferul de 32 de octeți.
- Dacă bufferul de 32 de octeți a fost invalidat, programul va scrie un singur octet de date în registrul *Host Block Data Byte*. După ce controlerul transmite adresa dispozitivului *slave*, codul comenzii și contorul de octeți, va transmite octetul de date din registrul *Host Block Data Byte* și va seta bitul BYTE_DONE_STS din registrul *Host Status*. Dacă mai sunt octeți de transmis, programul va scrie următorul octet de date în registrul *Host Block Data Byte* și va șterge bitul BYTE_DONE_STS. Controlerul va transmite apoi următorul octet de date.

Observație

- La funcționarea în modul I²C, cu bitul I2C_EN din registrul *Host Configuration* setat, protocolul *Block Write* implementat se modifică prin faptul că nu se transmite contorul de octeți conținut în registrul *Host Data 0* ca parte a mesajului.

Pentru o comandă *Block Read*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (1) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. După ce controlerul transmite adresa dispozitivului *slave* și codul comenzii, recepționează contorul de octeți în registrul *Host Data 0*. După cum bufferul de 32 de octeți a fost validat sau nu, octeții de date vor fi recepționați în mod diferit, după cum se specifică în continuare.

- Dacă bufferul de 32 de octeți a fost validat, controlerul va memora octeții de date recepționați în bufferul de 32 de octeți. În cazul în care contorul de octeți a ajuns la zero sau bufferul de 32 de octeți s-a umplut, controlerul va genera o întrerupere și va seta bitul BYTE_DONE_STS din registrul *Host Status*. Programul va citi apoi octeții de date în mod individual din registrul *Host Block Data Byte*, după care va șterge bitul BYTE_DONE_STS.

Observație

- Programul trebuie să execute o citire a registrului *Host Control* pentru a reseta pointerul în bufferul de 32 de octeți înaintea citirii registrului *Host Block Data Byte*.

- Dacă bufferul de 32 de octeți a fost invalidat, controlerul va memora un octet recepționat în registrul *Host Block Data Byte*. Apoi, controlerul va genera o întrerupere și va seta bitul BYTE_DONE_STS din registrul *Host Status*. Programul va citi octetul de date din registrul *Host Block Data Byte* și va șterge bitul BYTE_DONE_STS. Controlerul va recepționa apoi următorul octet de date.

3.6.2.7. Comanda *I²C Read*

Comanda *I²C Read* permite execuția unei operații de citire pe bloc cu anumite dispozitive compatibile I²C, cum sunt memoriile EEPROM seriale. De exemplu, această comandă permite accesul la dispozitive care utilizează ciclul cu format combinat I²C și necesită transmiterea unui singur octet după adresa dispozitivului *slave*. În mod tipic, acest octet corespunde unei adrese (deplasament) în cadrul circuitului de memorie.

La execuția unei comenzi *I²C Read*, controlerul SMBus transmite adresa dispozitivului *slave* urmată de conținutul registrului *Host Data 1*. Controlerul generează apoi o condiție de START repetată, transmite din nou adresa dispozitivului *slave* și începe recepția unui număr de octeți de date de la dispozitivul *slave*. După ce s-a recepționat numărul necesar de octeți de date, controlerul transmite un bit NACK și generează o condiție de STOP, terminând transferul.

Comanda *I²C Read* permite doar modul de adresare de 7 biți al magistralei I²C.

Înaintea lansării unei comenzi *I²C Read*, programul trebuie să forțeze bitul PEC_EN din registrul *Host Control* la 0 și bitul AAC din registrul *Auxiliary Control* la 0.

Observații

- Execuția unei comenzi *I²C Read* cu bitul PEC_EN setat la 1 produce rezultate nedefinite.
- Execuția unei comenzi *I²C Read* este permisă de controler indiferent de setarea bitului I2C_EN din registrul *Host Configuration*.

Înaintea lansării unei comenzi *I²C Read*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu bitul de direcție (0) în bitul RW. Registrul *Host Data 1* trebuie înscris cu octetul care trebuie transmis dispozitivului *slave*; de exemplu, acest octet poate fi adresa de început de la care trebuie citit conținutul circuitului de memorie.

Observații

- Pentru comanda *I²C Read*, bitul RW din registrul *Transmit Slave Address* trebuie setat la 0; această setare este diferită de setarea pentru alte comenzi de citire, pentru care bitul RW trebuie setat la 1, corespunzător direcției de citire.
- Deoarece se transmite un singur octet după adresa dispozitivului *slave*, comanda *I²C Read* nu se poate utiliza cu dispozitive care necesită transmiterea mai multor octeți după adresa dispozitivului *slave*, cum sunt memoriile care necesită o adresă de doi octeți.

Pentru recepționarea datelor de la dispozitivul *slave*, programul trebuie să execute următoarele operații pentru fiecare octet de date:

1. Testează starea bitului BYTE_DONE_STS din registrul *Host Status* și așteaptă până când controlerul setează acest bit la 1, ceea ce înseamnă că a recepționat un octet de date și l-a memorat în registrul *Host Block Data Byte*.
2. Citește conținutul registrului *Host Block Data Byte* într-un buffer din memorie.
3. Șterge bitul BYTE_DONE_STS din registrul *Host Status*.
4. Dacă octetul recepționat este penultimul octet, setează bitul LAST_BYTE din registrul *Host Control* și continuă cu pasul 1 (setarea acestui bit va determina controlerul să

transmită un bit NACK în locul unui bit ACK după recepția ultimului octet). În caz contrar (dacă octetul recepționat nu este penultimul octet), continuă cu pasul 1.

5. Dacă octetul recepționat este ultimul octet, șterge bitul LAST_BYTE din registrul *Host Control* și operația este terminată. În caz contrar, continuă cu pasul 1.

3.6.2.8. Comanda *Block Write-Block Read Process Call*

Comanda *Block Write-Block Read Process Call* implementează protocolul SMBus cu același nume, care a fost descris în secțiunea 3.4.8. Totuși, există următoarele deosebiri între această comandă și protocolul corespunzător descris în specificațiile SMBus:

- La implementarea comenzii, niciun contor de octeți, contorul de octeți de scriere (M) și contorul de octeți de citire (N), nu poate fi zero, în timp ce în specificația protocolului oricare contor de octeți poate fi zero.
- La implementarea comenzii, contorul de octeți combinat (M + N) nu trebuie să depășească 32, în timp ce în specificația protocolului contorul de octeți combinat nu trebuie să depășească 255.

Observație

- Bitul E32B din registrul *Auxiliary Control* trebuie setat înaintea lansării acestei comenzi.

Pentru execuția comenzii *Block Write-Block Read Process Call*, registrul *Transmit Slave Address* trebuie înscris cu adresa dispozitivului *slave* în câmpul Address și cu direcția transferului (0) în bitul RW. Registrul *Host Command* trebuie înscris cu codul comenzii. Registrul *Host Data 0* trebuie înscris cu contorul de octeți de scriere indicând numărul de octeți de date care vor fi transmiși. Octeții de date care vor fi transmiși trebuie înscriși în registrul *Host Block Data Byte*, octet cu octet; aceștia vor fi memorați în bufferul de 32 de octeți.

După transmiterea adresei dispozitivului *slave*, a codului comenzii, a contorului de octeți de scriere și a octeților de date din bufferul de 32 de octeți, controlerul recepționează contorul de octeți de citire și îl memorează în registrul *Host Data 0*. Controlerul recepționează apoi un număr de octeți de date indicat de contorul de octeți de citire, îi memorează în bufferul de 32 de octeți, generează o întrerupere și setează bitul BYTE_DONE_STS în registrul *Host Status*. Programul trebuie să execute o citire a registrului *Host Control* pentru a reseta pointerul în bufferul de 32 de octeți, să citească octeții de date individual din registrul *Host Block Data Byte* și să șteargă bitul BYTE_DONE_STS.

3.7. Utilizarea controlerului SMBus Intel cu dispozitive I²C

Magistrala I²C este utilizată de o mare varietate de dispozitive, ca memorii EEPROM, ceasuri de timp real și diferite tipuri de senzori, inclusiv senzori de temperatură. Controlerul SMBus Intel poate realiza comunicația cu multe din aceste dispozitive, pe lângă comunicația cu dispozitive SMBus native.

Există mai multe deosebiri între ciclurile de magistrală I²C și protocoalele SMBus; unele din aceste deosebiri au fost discutate în secțiunea 3.5. Totuși, prin selecția atentă a unor comenzi ale controlerului și prin anumite setări ale registrelor este adeseori posibilă stabilirea unei comunicații între controlerul SMBus și dispozitive *slave* I²C. De exemplu, controlerul SMBus permite o setare pentru validarea modului I²C; aceasta se obține prin setarea bitului I2C_EN din registrul *Host Configuration*. În plus, controlerul dispune de o comandă specifică I²C numită *I²C Read*, care a fost descrisă în secțiunea 3.6.2.7. Cu toate acestea, niciuna din aceste setări nu plasează controlerul într-un mod 100% compatibil I²C.

3.7.1. Cicluri de magistrală I²C

Specificațiile magistralei I²C definesc trei tipuri de bază ale ciclurilor de magistrală: ciclu de scriere, ciclu de citire și ciclu cu format combinat. Fiecare tip de ciclu funcționează în

modul bloc, putând transfera mai mulți octeți de date. În această secțiune se prezintă tipurile de bază ale ciclurilor de magistrală I²C și se subliniază deosebirile dintre fiecare tip de ciclu și un protocol SMBus corespunzător.

3.7.1.1. Ciclul de scriere I²C

Ciclul de scriere I²C este ilustrat în figura 3.13. Dispozitivul *master* poate transmite octeți multipli de date la dispozitivul *slave*; numărul de octeți nu este limitat. Numărul de octeți transferați este determinat de către dispozitivul *slave*; după un anumit număr de octeți recepționați, dispozitivul *slave* transmite un bit NACK, iar apoi dispozitivul *master* generează o condiție de STOP. Acest ciclu diferă de protocolul SMBus *Block Write*, la care dispozitivul *master* determină dimensiunea transferului prin transmiterea doar a numărului cerut de octeți în timpul transferului.

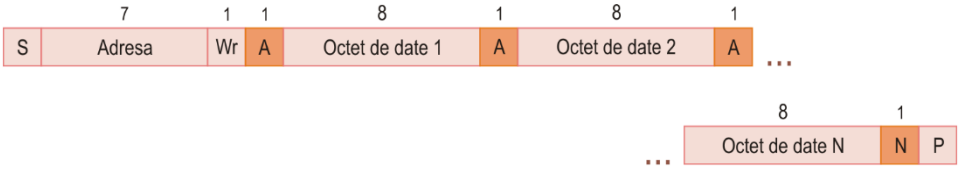

Figura 3.13. Ciclu de scriere I²C.

O altă deosebire față de protocolul SMBus *Block Write* este faptul că în timpul ciclului de scriere I²C octeții de date sunt transmiși imediat după adresa dispozitivului *slave*, în timp ce la protocolul SMBus se transmite un cod al comenzii și un contor de octeți înaintea transmiterii octeților de date. Aceasta poate preveni posibilitatea utilizării protocolului *Block Write* cu dispozitive I²C. O altă problemă posibilă este că un dispozitiv *slave* I²C va transmite un bit NACK după recepția ultimului octet de date, în timp ce, conform protocolului SMBus, dispozitivul *master* așteaptă un bit ACK de la dispozitivul *slave* chiar și după ultimul octet de date.

3.7.1.2. Ciclul de citire I²C

Ciclul de citire I²C este ilustrat în figura 3.14. Dispozitivul *master* poate citi octeți multipli de date de la dispozitivul *slave*; numărul de octeți nu este limitat. Numărul de octeți transferați este determinat de către dispozitivul *master*; după un anumit număr de octeți recepționați, dispozitivul *master* transmite un bit NACK, iar apoi generează o condiție de STOP.

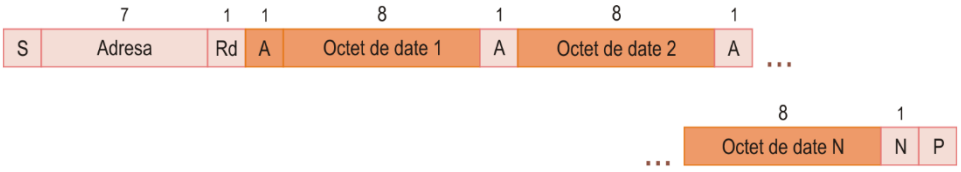

Figura 3.14. Ciclu de citire I²C.

Ciclul de citire I²C diferă semnificativ de protocolul SMBus *Block Read*. În timpul ciclului de citire I²C, octeții de date sunt recepționați de către dispozitivul *master* imediat după ce acesta transmite adresa dispozitivului *slave*. În timpul protocolului SMBus *Block Read*, după ce transmite adresa dispozitivului *slave*, dispozitivul *master* transmite un cod al comenzii, iar apoi generează o condiție de START repetată, transmite din nou adresa dispozitivului *slave* și așteaptă un contor de octeți de citire de la dispozitivul *slave* înaintea octeților de date. Datorită acestor deosebiri, protocolul SMBus *Block Read* nu se poate folosi pentru comunicația cu dispozitive care utilizează ciclul de citire I²C.

3.7.1.3. Ciclul cu format combinat I²C

Ciclul cu format combinat I²C este ilustrat în figura 3.15. În prima parte a acestui tip de ciclu, dispozitivul *master* transmite adresa dispozitivului *slave*, după care transmite un anumit număr de octeți de date la dispozitivul *slave* până când acest dispozitiv transmite un bit NACK. În partea a doua a acestui tip de ciclu, dispozitivul *master* generează o condiție de START repetată, transmite din nou adresa dispozitivului *slave*, recepționează un anumit număr de octeți de date de la dispozitivul *slave*, transmite un bit NACK și generează o condiție de STOP.

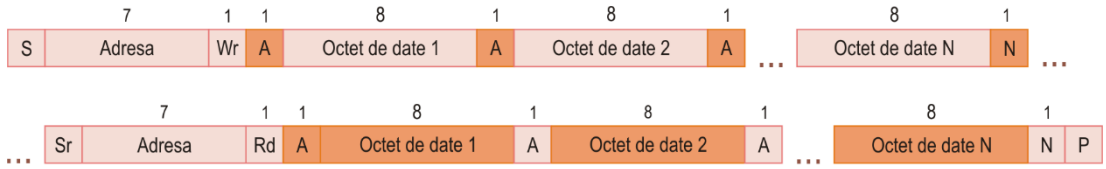

Figura 3.15. Ciclu cu format combinat I²C.

Pentru comunicația cu dispozitive I²C care utilizează formatul combinat I²C, o posibilitate ar fi utilizarea protocolului SMBus *Block Write-Block Read Process Call*. Totuși, una din probleme este că în timpul acestui protocol dispozitivul *master* transmite un cod de comandă și un contor de octeți de scriere, care nu fac parte din ciclul cu format combinat I²C. O altă problemă este că la protocolul SMBus, după generarea unei condiții de START repetate și transmiterea adresei dispozitivului *slave*, dispozitivul *master* așteaptă recepționarea unui contor de octeți de citire, care de asemenea nu face parte din ciclul cu format combinat I²C. De aceea, protocolul SMBus *Block Write-Block Read Process Call* nu se poate folosi pentru comunicația cu dispozitive care utilizează ciclul cu format combinat I²C.

3.7.2. Comenzi compatibile I²C ale controlerului SMBus

Protocolul utilizat de mai multe comenzi ale controlerului SMBus poate fi modificat într-o anumită măsură pentru a fi mai compatibil cu ciclurile de magistrală I²C prin setarea bitului I2C_EN din registrul *Host Configuration*. Efectele setării bitului I2C_EN sunt următoarele:

- Comanda *Block Write* nu va transmite contorul de octeți;
- Comanda *Process Call* nu va transmite codul comenzii.

Observație

- Nu trebuie confundată setarea bitului I2C_EN cu selectarea comenzii *I²C Read*. Setarea bitului I2C_EN modifică execuția comenzilor *Block Write* și *Process Call*, în timp ce selectarea comenzii *I²C Read* permite implementarea unui ciclu special cu format combinat I²C, după cum s-a descris în secțiunea 3.6.2.7.

Atunci când controlerul SMBus trebuie utilizat cu un anumit dispozitiv I²C, nu există o regulă generală pentru a determina dacă este posibilă comunicația cu acel dispozitiv. Trebuie analizat cu atenție catalogul dispozitivului pentru a determina care cicluri I²C sunt necesare și dacă un anumit ciclu poate fi generat sau nu de controlerul SMBus. Următoarele comenzi ale controlerului SMBus pot fi considerate compatibile cu anumite dispozitive I²C în condițiile specificate:

- *Quick Command*: Transmite adresa dispozitivului *slave* și bitul R/W# la dispozitiv; acest bit specifică o comandă pentru dispozitiv.
- *Send Byte*: Transmite un singur octet de date din registrul *Host Command* la dispozitiv.

- *Receive Byte*: Recepționează un singur octet de date de la dispozitiv în registrul *Host Data 0*.
- *Write Byte*: Transmite doi octeți de date la dispozitiv, primul din registrul *Host Command* și al doilea din registrul *Host Data 0*.
- *Read Byte*: Transmite un singur octet de date din registrul *Host Command* la dispozitiv și recepționează un singur octet de date de la dispozitiv în registrul *Host Data 0*.
- *Write Word*: Transmite trei octeți de date la dispozitiv, primul din registrul *Host Command*, al doilea din registrul *Host Data 0*, iar al treilea din registrul *Host Data 1*.
- *Read Word*: Transmite un singur octet de date din registrul *Host Command* la dispozitiv și recepționează doi octeți de date de la dispozitiv, primul în registrul *Host Data 0* și al doilea în registrul *Host Data 1*.
- *Process Call* cu I2C_EN = 1: Transmite doi octeți de date la dispozitiv, primul din registrul *Host Command* și al doilea din registrul *Host Data 0*, și recepționează doi octeți de date de la dispozitiv, primul în registrul *Host Data 0* și al doilea în registrul *Host Data 1*.
- *Block Write* cu I2C_EN = 1: Transmite *N* octeți de date la dispozitiv, primul octet din registrul *Host Command* și ceilalți *N*–1 octeți din bufferul de 32 de octeți sau individual din registrul *Host Block Data Byte*.

Observație

- Comanda *Block Write* a controlerului SMBus poate transfera maximum 32 de octeți de date atunci când bufferul de 32 de octeți este validat, deși magistrala I²C nu limitează dimensiunea transferului.

- *I²C Read*: Transmite un singur octet de date din registrul *Host Data 1* la dispozitiv și recepționează *N* octeți de date de la dispozitiv, în mod individual în registrul *Host Block Data Byte*.

3.7.3. Exemple de utilizare a dispozitivelor I²C cu controlerul SMBus

În această secțiune se prezintă două exemple pentru conectarea unor dispozitive compatibile I²C la controlerul SMBus. Primul exemplu este pentru conectarea unei memorii EEPROM seriale, iar al doilea exemplu este pentru conectarea unui convertor analog-digital.

3.7.3.1. Utilizarea memoriei EEPROM Microchip 24LC01B

Circuitul Microchip Technology 24LC01B este o memorie EEPROM serială cu capacitatea de 1 Kbit. Circuitul este organizat ca un bloc de memorie de 128 x 8 biți. Acest circuit este utilizat în mod frecvent în modulele de memorie DIMM ca memorie EEPROM SPD (*Serial Presence Detect*).

Ne vom referi doar la operațiile de citire permise de această memorie, deși circuitul permite și operații de scriere. Această memorie permite următoarele operații de citire: citire de la adresa curentă; citire aleatorie; citire secvențială.

Citire de la adresa curentă

Circuitul 24LC01B are un pointer intern de adresă la ultimul octet accesat în timpul unei operații precedente. Acest pointer este inițializat cu zero la punerea sub tensiune a circuitului și este incrementat automat după fiecare acces de citire sau scriere. Ciclul de citire de la adresa curentă este ilustrat în figura 3.16.

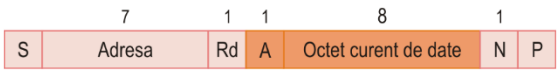

Figura 3.16. Ciclu de citire de la adresa curentă a circuitului de memorie Microchip 24LC01B.

Pentru ciclul de citire de la adresa curentă, controlerul transmite adresa dispozitivului *slave* și bitul R/W# setat la 1 (bitul Rd). Circuitul transmite un bit ACK și apoi transmite octetul curent de date din memorie. Este simplu de observat că acest ciclu poate fi generat utilizând comanda *Receive Byte* a controlerului SMBus.

Citire aleatorie

O operație de citire aleatorie permite accesul la oricare locație de memorie, indiferent de accesele precedente. Ciclul de citire aleatorie este ilustrat în figura 3.17. După transmiterea adresei dispozitivului *slave* și a bitului R/W# setat la 0 (bitul Wr), controlerul transmite adresa de 8 biți a locației care trebuie accesată, apoi generează o condiție de START repetată și transmite din nou adresa dispozitivului *slave*. Circuitul setează pointerul intern de adresă și apoi transmite octetul de date din locația adresată.

Figura 3.17. Ciclu de citire aleatorie a circuitului de memorie Microchip 24LC01B.

Revizuind protocoalele SMBus, se poate observa că ciclul de citire aleatorie este similar cu protocolul SMBus *Read Byte*. Se poate utiliza comanda *Read Byte* a controlerului SMBus pentru a genera acest ciclu, registrul *Host Command* conținând adresa locației.

Citire secvențială

O operație de citire secvențială poate fi inițiată de controler în mod similar cu o operație de citire aleatorie, transmițând adresa dispozitivului *slave*, bitul Wr, adresa primei locații care trebuie accesată, urmată de o condiție de START repetată și adresa dispozitivului *slave* (figura 3.18). Circuitul setează pointerul intern de adresă și apoi transmite un bloc de octeți de date începând de la locația adresată. Circuitul așteaptă ca ciclul să fie terminat de controler prin transmiterea unui bit NACK după ultimul octet de date (n + x) și generarea unei condiții de STOP.

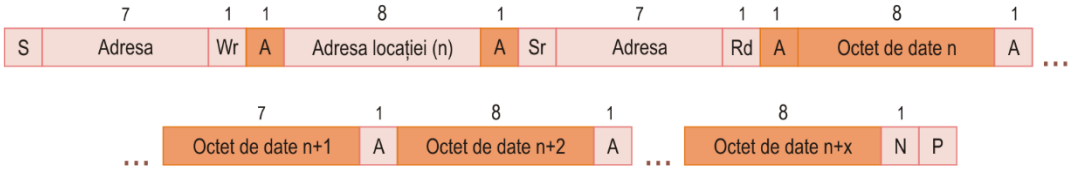

Figura 3.18. Ciclu de citire secvențială a circuitului de memorie Microchip 24LC01B.

Acest ciclu poate fi generat utilizând comanda *I^2C Read* a controlerului SMBus. Registrul *Host Data 1* trebuie încărcat cu adresa locației înaintea lansării comenzii. Programul trebuie să monitorizeze octeții de date citiți și să seteze bitul LAST_BYTE din registrul *Host Control* după recepția penultimului octet pentru a asigura transmiterea de către controler a unui bit NACK după ultimul octet.

3.7.3.2. Utilizarea convertorului analog-digital LTC2481

Circuitul LTC2481 este un convertor analog-digital de 16 biți al firmei Linear Technology. Acest circuit poate fi configurat printr-o interfață I^2C pentru a obține un câștig programabil de la 1 la 256, pentru a digitiza un semnal extern sau temperatura de la un senzor integrat, pentru a selecta suprimarea zgomotelor frecvenței de rețea (50 Hz, 60 Hz sau simultan 50Hz și 60 Hz) și pentru a selecta un mod rapid 2x.

Convertorul LTC2481 are două registre, un registru de ieșire și un registru de configurație. Registrul de ieșire conține rezultatul ultimei conversii. Registrul de configurație este programabil de către utilizator și permite setarea modului de funcționare al convertorului. Convertorul recunoaște mai multe operații de scriere și de citire. Operațiile de scriere permit inițierea unei noi conversii și setarea modului de funcționare al convertorului. Operațiile de citire permit citirea a până la trei octeți de la circuit, care cuprind rezultatul ultimei conversii și conținutul registrului de configurație.

Inițierea unei noi conversii

O nouă conversie poate fi inițiată cu un simplu ciclu de scriere. Pentru acest ciclu, controlerul SMBus transmite adresa dispozitivului și un bit Wr (figura 3.19). Circuitul transmite un bit ACK pentru a confirma ciclul de scriere. Atunci când controlerul termină ciclul prin generarea unei condiții de STOP, circuitul inițiază o nouă conversie. Rezultatul acestei conversii poate fi obținut printr-un ciclu de citire ulterior.

Figura 3.19. Ciclu de scriere pentru inițierea unei noi conversii la circuitul LTC2481.

Acest ciclu de scriere poate fi generat utilizând comanda *Quick Command* a controlerului SMBus.

Inițierea unei noi conversii cu actualizarea configurației

Se poate utiliza un ciclu modificat de scriere pentru a iniția o nouă conversie și pentru a înscrie o nouă configurație în registrul de configurație. După ce controlerul SMBus transmite adresa dispozitivului și un bit Wr, dispozitivul confirmă ciclul de scriere. Controlerul transmite apoi un octet de date conținând noua configurație (figura 3.20). Dispozitivul transmite un bit ACK bit și controlerul generează o condiție de STOP, moment în care dispozitivul inițiază o nouă conversie.

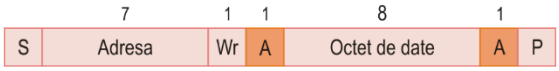

Figura 3.20. Ciclu de scriere pentru inițierea unei noi conversii și actualizarea configurației circuitului LTC2481.

Acest ciclu de scriere modificat poate fi generat utilizând comanda *Send Byte* a controlerului SMBus.

Citire continuă

Atunci când nu este necesară schimbarea configurației după fiecare ciclu de conversie, rezultatul conversiei poate fi citit în mod continuu. Atunci când dispozitivul termină o conversie, acesta poate fi adresat pentru o operație de citire. La sfârșitul unei operații de citire, începe o nouă conversie. Dacă ciclul de conversie nu este terminat și dispozitivul recepționează o comandă validă, acesta transmite un bit NACK indicând faptul că ciclul de conversie este în curs.

Ciclul pentru citirea unei valori de 24 de biți de la convertor este ilustrat în figura 3.21. După recepționarea adresei dispozitivului *slave* și a unui bit Rd, convertorul transmite trei octeți de date la controler, care transmite apoi un bit NACK și generează o condiție de STOP.

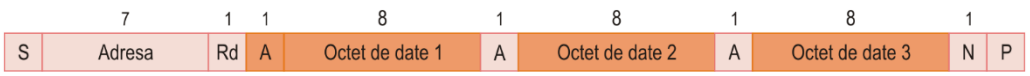

Figura 3.21. Ciclu de citire pentru citirea rezultatului ultimei conversii și a registrului de configurație al circuitului LTC2481.

Deoarece acest ciclu necesită citirea a trei octeți de la dispozitiv, ciclul trebuie generat utilizând una din comenzile pe bloc ale controlerului SMBus. Totuși, niciuna din comenzile disponibile nu permite generarea acestui ciclu, astfel încât operația de citire continuă nu poate fi lansată în execuție utilizând controlerul SMBus.

Scriere/citire continuă

După terminarea unui ciclu de conversie, se poate actualiza registrul de configurație al convertorului LTC2481 și se pot citi trei octeți de date de la circuit cu ciclul ilustrat în figura 3.22.

Figura 3.22. Ciclu de scriere, citire și start conversie al circuitului LTC2481.

Pentru acest ciclu, controlerul transmite adresa dispozitivului *slave*, un bit Wr și un octet de date conținând informația de configurație. Controlerul generează apoi o condiție de START repetată și transmite din nou adresa dispozitivului slave. Dispozitivul returnează trei octeți de date la controler, care transmite un bit NACK și generează o condiție de STOP.

Acest ciclu poate fi generat utilizând comanda *I^2C Read* a controlerului SMBus. Registrul *Host Data 1* trebuie încărcat cu informația de configurație înaintea lansării comenzii. Programul trebuie să seteze bitul LAST_BYTE din registrul *Host Control* după recepția celui de-al doilea octet de date de la dispozitiv, ceea ce va determina transmiterea de către controler a unui bit NACK și generarea unei condiții de STOP după cel de-al treilea octet recepționat.

3.8. Aplicații

3.8.1. Răspundeți la următoarele întrebări:

a. Care sunt îmbunătățirile magistralei SMBus introduse de versiunea 2.0 și versiunea 3.0 a specificațiilor SMBus?

b. Care sunt principalele deosebiri între magistrala SMBus și magistrala I^2C?

c. Care este deosebirea dintre ciclul de citire I^2C și protocolul SMBus *Block Read*?

d. Care este efectul setării bitului I2C_EN din registrul *Host Configuration* al controlerului SMBus Intel comparativ cu selectarea comenzii *I^2C Read* a acestui controler?

3.8.2. Deschideți proiectul creat pentru aplicația 2.7.5 din lucrarea de laborator *Magistrala PCI Express*. Copiați în directorul proiectului fișierul SMBus.h, disponibil pe pagina laboratorului în arhiva SMBus.zip. Adăugați la proiect fișierul SMBus.h, deschideți fișierul sursă AppScroll.cpp și adăugați o directivă `#include` pentru a include fișierul antet SMBus.h. În fișierul sursă AppScroll.cpp, scrieți o funcție care returnează adresa de bază (de tip `WORD`) a registrelor de I/E mapate în spațiul de I/E al controlerului SMBus. Parametrul de intrare al acestei funcții este pointerul la antetul de configurație PCIe al controlerului SMBus, determinat în aplicația 2.7.5. Apelați această funcție, memorați adresa de bază returnată de funcție într-o variabilă globală și afișați adresa de bază.

Observații

- În antetul de configurație PCIe, registrul care conține adresa de bază a registrelor de I/E mapate în spațiul de I/E al controlerului SMBus este SMB_BASE. Adresa de bază se află în cuvântul inferior al acestui registru de 32 de biți. Bitul 0 al adresei de bază care s-a citit din acest registru trebuie resetat la 0.

- Pentru un procesor AMD, în cazul în care controlerul SMBus este identificat pe magistrala 0, dispozitivul 20 și funcția 0, adresa de bază a registrelor de I/E nu se poate determina prin citirea registrelor antetului de configurație PCIe. În acest caz, funcția va returna adresa de bază 0x0B00.

3.8.3. Extindeți aplicația 3.8.2 cu o funcție care abandonează tranzacția curentă a controlerului SMBus. Parametrul de intrare al funcției este adresa de bază a registrelor de I/E mapate în spațiul de I/E al controlerului SMBus. Funcția setează bitul KILL din registrul *Host Control*, așteaptă un anumit timp (de exemplu, 100 ms), iar apoi șterge bitul KILL. Funcția returnează valoarea 0 dacă tranzacția curentă a fost abandonată și valoarea 1 în caz contrar. Tranzacția a fost abandonată dacă bitul HOST_BUSY din registrul *Host Status* nu este setat și bitul FAILED din același registru este setat.

3.8.4. Continuați aplicația 3.8.2 cu o funcție care transmite comanda *Receive Byte* unui dispozitiv SMBus. Parametrii de intrare ai funcției sunt indicatorul la fereastra aplicației (de tip `hWnd`) și adresa dispozitivului SMBus (de tip `BYTE`). Funcția returnează valoarea 0 dacă execuția comenzii *Receive Byte* s-a terminat cu succes, caz în care bitul INTR din registrul *Host Status* este setat. Funcția returnează valoarea 1 dacă execuția comenzii s-a terminat cu o eroare, caz în care unul din următorii biți ai registrului *Host Status* este setat: FAILED, BUS_ERR sau DEV_ERR.

Observații

- Definiți deplasamentele registrelor utilizate de funcție și măștile de biți necesare pentru aceste registre cu directive `#define` la începutul fișierului AppScroll.cpp.
- La apelul funcțiilor driverului HW pentru accesarea unui registru, adresa registrului este formată prin adunarea deplasamentului său la adresa de bază a registrelor de I/E ale controlerului SMBus.

3.8.5. Continuați aplicația 3.8.2 pentru a identifica dispozitivele conectate la magistrala SMBus a calculatorului. În funcția principală (`AppScroll`) a aplicației, după determinarea adresei de bază a registrelor de I/E ale controlerului SMBus utilizând funcția scrisă pentru aplicația 3.8.2, transmiteți comanda *Receive Byte* dispozitivelor cu adrese cuprinse între 0x10 și 0x7F utilizând funcția scrisă pentru aplicația 3.8.4. Dacă execuția acestei comenzi se termină cu succes, afișați un mesaj indicând faptul că s-a găsit un dispozitiv SMBus și afișați adresa dispozitivului. În plus, afișați tipul dispozitivului detectat, pe baza următoarelor domenii de adrese utilizate de anumite dispozitive SMBus:

- 0x18..0x1F: Senzori termici ai unei memorii SPD;
- 0x30..0x37: Protecție la scriere a unei memorii SPD;
- 0x40..0x47: Ceas de timp real;
- 0x50..0x57: Memorie SPD.

3.8.6. Extindeți aplicația pentru identificarea dispozitivelor conectate la magistrala SMBus cu o funcție care transmite comanda *Read Byte* unui dispozitiv SMBus. Parametrii de intrare ai funcției sunt indicatorul la fereastra aplicației (de tip `hWnd`), adresa dispozitivului SMBus (de tip `BYTE`) și codul comenzii (de tip `BYTE`). Funcția returnează aceleași valori ca și funcția care transmite comanda *Receive Byte*, scrisă pentru aplicația 3.8.4.

3.8.7. Extindeți aplicația pentru identificarea dispozitivelor conectate la magistrala SMBus cu o funcție care citește și afișează conținutul unei memorii SPD. Parametrii de intrare ai funcției sunt indicatorul la fereastra aplicației (de tip `hWnd`) și adresa dispozitivului SMBus (de tip `BYTE`). Funcția returnează aceleași valori ca și funcția care transmite comanda *Receive Byte*, scrisă pentru aplicația 3.8.4. Operațiile care trebuie executate de această funcție sunt următoarele:

1. Transmite comanda *Read Byte* dispozitivului (o memorie SPD), cu codul comenzii setat la 0. La recepția acestei comenzi, memoria SPD va reseta pointerul său intern și va returna primul octet din memorie.
2. Depune octetul recepționat în prima locație a unui buffer de 512 octeți și, pe baza acestui octet, determină numărul de octeți utilizați din memoria SPD. Acest număr este indicat de biții 3..0 ai octetului recepționat. Dacă acești biți sunt 0001, 128 de octeți sunt utilizați în memoria SPD, dacă biții sunt 0010, 256 de octeți sunt utilizați, iar dacă biții sunt 0011, 384 de octeți sunt utilizați. Pentru alte combinații, se presupune că numărul de octeți utilizați este 512.
3. Transmite în mod repetat (într-o buclă) comanda *Receive Byte* dispozitivului și memorează octeții recepționați începând cu a doua locație a bufferului de 512 de octeți. Contorul de iterații trebuie să fie numărul de octeți utilizați în memoria SPD minus 1.
4. Afișează conținutul memoriei SPD. Se afișează în fiecare linie trei cifre în zecimal reprezentând deplasamentul unei zone de 8 octeți din memorie, urmate de 8 octeți de date în hexazecimal.

În funcția principală a aplicației, apelați funcția descrisă anterior pentru fiecare memorie SPD detectată pe magistrala SMBus; memoriile SPD au adrese cuprinse între 0x50 și 0x57.

3.8.8. Modificați aplicația 3.8.7 pentru a decodifica o parte a informațiilor citite dintr-o memorie SPD în locul afișării conținutului memoriei. Pentru decodificarea conținutului memoriei SPD, utilizați fișierul antet SPD.h, disponibil pe pagina laboratorului în arhiva SPD.zip. Trebuie decodificate următoarele informații:

- Revizia SPD;
- Tipul memoriei DRAM;
- Tipul modulului de memorie;
- Densitatea (capacitatea) circuitelor SDRAM;
- Numărul de bancuri interne;
- Tensiunea nominală a modulului;
- Tipul memoriei și frecvența magistralei, pe baza duratei minime a ciclului de ceas;
- Valoarea minimă a latenței semnalului CAS (ns);
- Valoarea minimă a întârzierii dintre semnalele RAS și CAS (ns);
- Valoarea minimă a timpului de preîncărcare RAS (ns);
- Producătorul modulului de memorie;
- Numărul de serie al modulului;
- Numărul modulului de memorie;
- Producătorul circuitelor DRAM.

Observație

- Fișierul antet SPD.h definește structuri pentru a simplifica decodificarea conținutului memoriei SPD. Fiecare structură conține un octet și un pointer la un șir de caractere. Pe baza valorii unui octet din memoria SPD, se poate afișa direct șirul de caractere corespunzător semnificației octetului. Fișierul conține și comentarii care descriu semnificația celor mai importanți octeți din memoria SPD.

3.8.9. Extindeți aplicația 3.8.7 cu o funcție care transmite comanda *I²C Read* unui dispozitiv SMBus și memorează într-un buffer numărul specificat de octeți recepționați de la dispozitiv. Parametrii de intrare ai funcției sunt următorii: indicatorul la fereastra aplicației (de tip `hWnd`); adresa dispozitivului SMBus (de tip `BYTE`); codul comenzii (de tip `BYTE`); un pointer la bufferul de recepție (de tip `PBYTE`); numărul de octeți care trebuie recepționați (de tip `int`). Dacă numărul specificat de octeți care trebuie recepționați este 0, funcția determină numărul de octeți care trebuie recepționați în modul descris la pasul 2 al operațiilor care

trebuie executate pentru aplicația 3.8.7. Funcția returnează aceleași valori ca și funcția care transmite comanda *Receive Byte*, scrisă pentru aplicația 3.8.4. Funcția trebuie să execute operațiile descrise în secțiunea 3.6.2.7 pentru recepția fiecărui octet de date de la dispozitiv.

3.8.10. Modificați aplicația 3.8.7 pentru a utiliza comanda *I²C Read* pentru citirea conținutului unei memorii SPD în locul comenzilor *Read Byte* și *Receive Byte*. Modificați funcția care citește și afișează conținutul unei memorii SPD pentru a apela funcția scrisă pentru aplicația 3.8.9 pentru transmiterea comenzii *I²C Read* la memoria SPD. La apelul acestei funcții, specificați valoarea 0 pentru codul comenzii (ceea ce va reseta pointerul intern al memoriei) și valoarea 0 pentru numărul de octeți care trebuie recepționați (în acest fel, numărul de octeți care trebuie recepționați va fi determinat pe baza primului octet citit din memorie).

Bibliografie

[1] Fan, Roger, "SMBus Quick Start Guide", Application Note AN4471, Rev. 1, Freescale Semiconductor Inc., 2012, http://cache.freescale.com/files/32bit/doc/app_note/AN4471.pdf.

[2] Fleming, Sam, "Interfacing I²C Devices to an Intel SMBus Controller", White Paper, Intel Corporation, 2009, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/smbus-controller-i2c-devices-paper.pdf.

[3] Intel Corporation, "Intel 8 Series/C220 Series Chipset Family Platform Controller Hub (PCH)", Datasheet, May 2014, http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8-series-chipset-pch-datasheet.pdf.

[4] JEDEC Solid State Technology Association, "SPD Annex L: Serial Presence Detect (SPD) for DDR4 SDRAM Modules", Release 6, SPD4.1.2.L-6, Nov. 2020.

[5] JEDEC Solid State Technology Association, "Standard Manufacturer's Identification Code", JEP106BC, Feb. 2021.

[6] Linear Technology Corporation, "LTC2481 – 16-Bit ΔΣ ADC with Easy Drive Input Current Cancellation and I²C Interface", 2005, http://cds.linear.com/docs/en/datasheet/2481fd.pdf.

[7] Melexis Microelectronic Integrated Systems, "SMBus Communication with MLX90614", Application Note, Rev. 004, 2008, http://www.generationrobots.com/media/SMBus-communication-with-MLX90614.pdf.

[8] Microchip Technology Inc., "24AA01/24LC01B 1K I²C Serial EEPROM", Datasheet DS21711J, 2009, http://ww1.microchip.com/downloads/en/DeviceDoc/21711J.pdf.

[9] System Management Interface Forum, "System Management Bus (SMBus) Specification", Version 3.0, 20 December 2014, http://pmbus.org/Assets/PDFS/Public/SMBus_3_0_20141220.pdf.