Universitatea Politehnica Timișoara
Facultatea de Automatică și Calculatoare
Calculatoare și Tehnologia Informației

Diagnoza pentru sisteme infotainment in industria automotive

Proiect de diplomă

Mihai Bădescu

Conducător științific:
Conf. dr. ing. ***Lucian Prodan***

Timișoara
2018

Cuprins

1. Introducere

1.1. Apariția pieței. Statistici privind industria auto globală

Încă din momentul de când oamenii au început să comunice între ei, prezența comerțului a devenit necesară. Dacă la început, oamenii se mulțumeau cu puține lucruri și își produceau tot ce era necesar traiului, cu timpul, pe măsura dezvoltării civilizației, nevoile au crescut și nu au mai putut fi satisfăcute decât prin schimb, creându-se adevărate curente și căutări reciproce. Aceste curente au cunoscut o dezvoltare continuă, ajungând în final să fie soluționate prin comerț.

La început, schimburile se efectuau în mod direct, produs contra produs, constituind cea mai veche metodă de cumpărare, trocul. Totul a fost mai simplu atunci când s-a început folosirea unei mărfi intermediare, moneda. De aici, totul s-a descompus în două operațiuni: vânzarea și cumpărarea și astfel a luat naștere conceptul de piață, cerere și ofertă.

Piața este în general orice varietate de sisteme, proceduri, instituții, relații sociale sau infrastructuri unde are loc comerțul, se schimbă bunuri și servicii, ceea ce e o parte constituentă a economiei.

Cererea reprezintă cantitatea dintr-un anumit bun sau serviciu care se cere pe o piață la un moment dat și la un nivel al prețului.

Oferta reprezintă cantitatea dintr-un anumit bun sau serviciu pe care producătorii sunt dispuși să o ofere pe piață la un moment dat și la un anumit nivel al prețului.

Luând în considerare cele mai recente tendințe din domeniul tehnologiei, adică tendința de a renunța pe cât posibil la butoanele fizice în favoarea ecranului tactil, cererea pentru acesta fiind din ce în ce mai mare, fapt ce se poate observa prin expansiunea acestui fenomen pe ramura telefoanelor mobile, tabletelor, netbookurilor, notebookurilor, stațiilor de informare. Astfel, pentru o cerere așa de mare, oferta trebuie sa fie pe măsură.

Astfel, pentru a satisface cerințele pieței, a început migrarea de la tastatura fizică la ecranul tactil, fenomen ce poartă numele de modificarea cererii. Acest fenomen poate fi un rezultat al modificării unuia sau a mai mulți determinanți cum ar fi: veniturile consumatorilor, prețul produselor conexe, prețul așteptat în viitor, gustul consumatorilor, numărul clienților potențiali.

Pe același criteriu, din dorința oamenilor de a avea o interfață din ce în ce mai dinamică, mai interactivă și mai atractivă, cererea pentru a folosi și în mașină diferite afișaje care să fie controlate printr-o atingere, duce la răspândirea acestui curent și în domeniul automotive . Treptat se dorește eliminarea butoanelor fizice și înlocuirea acestora cu diferite interfețe tactile, dispozitive controlate prin gesturi, voce etc. .

Numărul de mașini vândute din anul 1990 până în prezent.

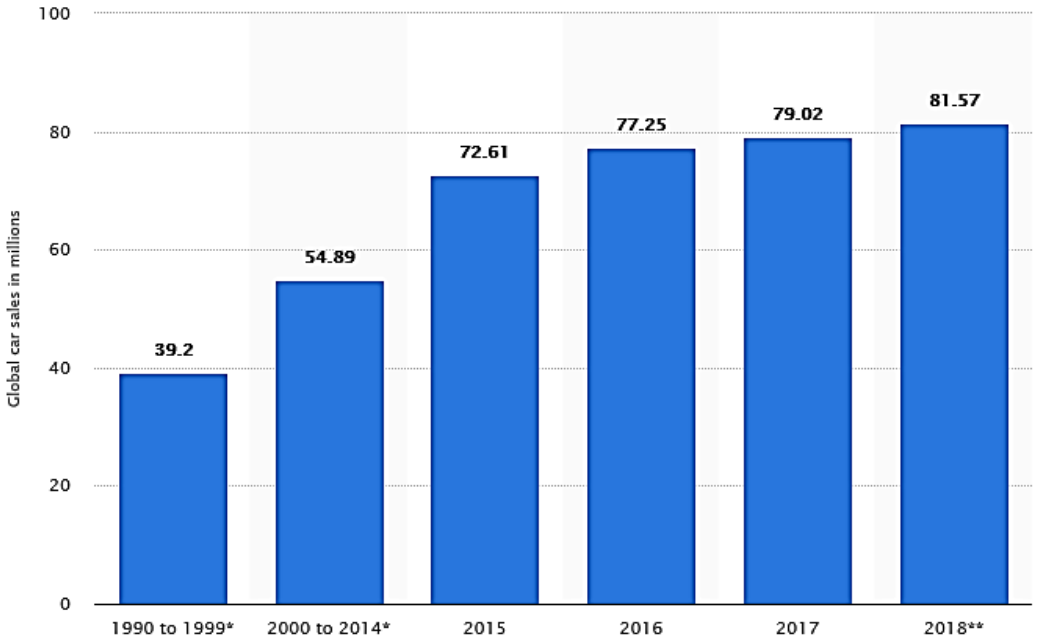


FIG. 1.1.1

Vânzările globale de autoturisme au depășit cifra de 80 milioane de vehicule în 2018. Împreună cu China, Statele Unite se numără printre cele mai mari piețe auto din întreaga lume, atât în ceea ce privește producția, cât și vânzarea. Aproximativ 6,9 milioane de autoturisme au fost vândute clienților din S.U.A. în 2016, iar aproximativ 4 milioane de mașini au fost produse aici în același an. Statele Unite au devenit o piață auto-cheie la începutul anilor 1900, când Ford a introdus producția de mașini pe linia de asamblare pentru a fabrica modelul T. În prezent, Ford Motor Company rămâne printre principalii producători de autoturisme, iar Ford Focus este unul dintre cele mai bine vândute vehicule compacte din lume în 2016. Din punct de vedere al veniturilor, Toyota, Volkswagen și Daimler au trecut în fruntea listei principalilor producători de automobile, în timp ce industria furnizorilor componentelor auto a fost dominată de Bosch, Continental, Denso și Magna.

Promovate de inițiative globale, cum ar fi Acordul de la Paris, mai multe țări din întreaga lume adoptă controale mai stricte privind emisiile pe modele noi de vehicule. Ca atare, autovehiculele încep să-și extindă afacerea în sectorul mobilității electrice. Germania este de așteptat să deschidă drumul cu producția electrică, proiectată pentru a ajunge la aproximativ 1,3 milioane de unități până în 2021.

În decursul următorului deceniu, tehnologiile legate de internet și vehiculele autonome vor determina o nouă revoluție în sectorul automobilelor. Piața globală a componentelor hardware de conducere autonome este de așteptat să crească de la 400 de milioane de dolari SUA în 2015 la 40 de miliarde de dolari SUA în 2030.

1.2. Diferența dintre generații și tehnologii

Interiorul unei mașini din anii 1990:

FIG. 1.2.1

FIG. 1.2.2

Putem observa câteva butoane cu diferite funcționalități care acum ni se par banale, obligatorii, dar în acea vreme erau chiar dotări de top. Putem observa următoarele butoane cu funcționalitățile lor:

- Dezaburire lunetă
- Activare proiectoare ceață
- Avarii
- Radio-casetofon
- Ventilator
- Poziție ventilație
- Temperatură
- Martori centură
- Martori frână
- Martori ABS

Interiorul unei mașini din anul 2017:

FIG. 1.2.3

În acest caz putem observa un sistem multimedia cu ecran tactil capabil sa realizeze o sumedenie de funcționalități:

- music player
- navigație
- conexiune bluetooth cu telefonul mobil
- conexiune wi-fi
- afișare informații mașină (consum carburant, presiunea în anvelope) etc.

Butonul fizic

Butonul este un tip de comutator utilizat pentru controlul unor aparate sau numai a unor funcții ale acestora. Un comutator electric este orice dispozitiv folosit pentru a întrerupe fluxul de electroni într-un circuit. Întrerupătoarele sunt dispozitive binare în esență: ele sunt complet activate ("închise") sau complet dezactivate ("deschise").

Cel mai simplu tip de comutator este unul în care doi conductori electrici sunt puși în contact unul cu celălalt prin mișcarea unui mecanism de acționare. Alte întrerupătoare sunt mai complexe, conținând circuite electronice capabile să pornească sau să se oprească, în funcție de anumiți stimuli fizici (cum ar fi lumina sau câmpul magnetic). În orice caz, ieșirea finală a oricărui comutator va fi (cel puțin) o pereche de terminale de conectare care fie vor fi conectate

împreună prin mecanismul de contact intern al comutatorului ("închis"), fie nu vor fi conectate împreună ("deschis") .

Tipuri de comutatoare:

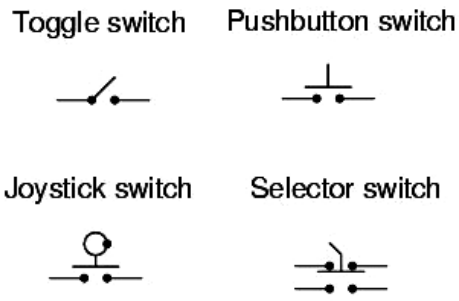


FIG. 1.2.4

Ecrane tactile

Există multe tehnologii ale ecranelor tactile, dar cele mai des întâlnite sunt cele rezistive și cele capacitive.

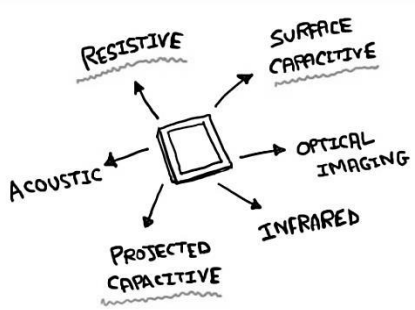


FIG. 1.2.5

Ecranele rezistive rezistă literalmente atingerii și dacă ecranul este apăsat destul de puternic se poate simți o ușoară deformare a acestuia, acest fenomen este ceea ce face ca ecranele rezistive să funcționeze.

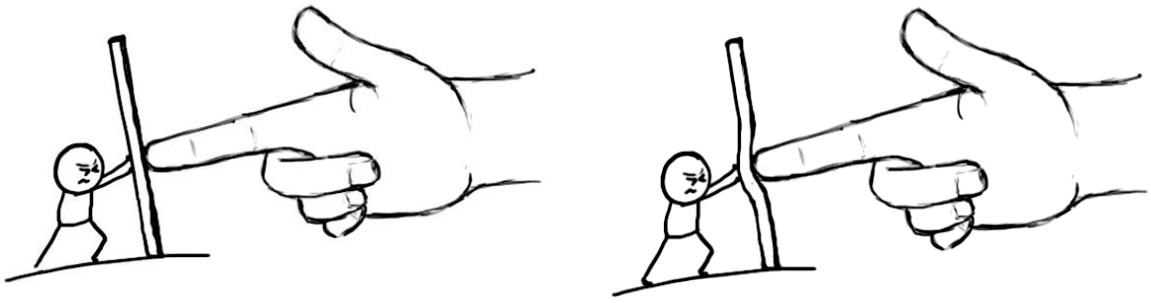
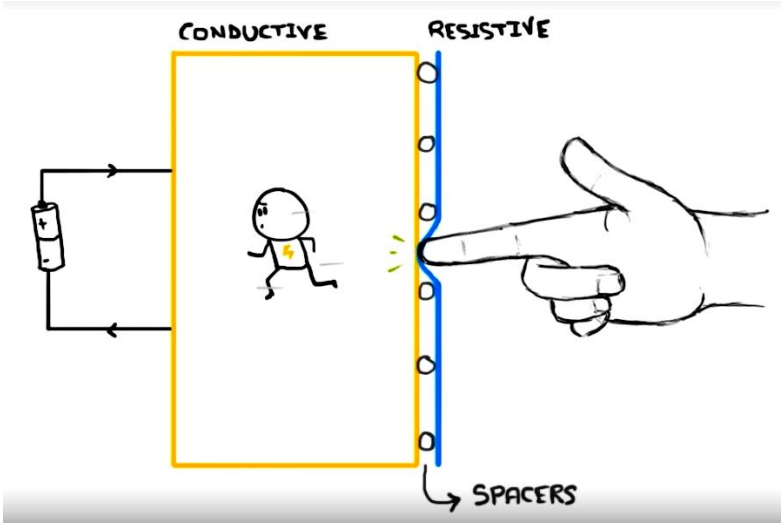


FIG. 1.2.6

Ecranele tactile rezistive au două straturi: conductiv și rezistiv. Acestea sunt separate prin puncte mici numite distanțiere. Curentul electric circulă prin stratul conductiv tot timpul, dar când ecranul este apăsat, stratul rezistiv intră în contact cu stratul conductiv.



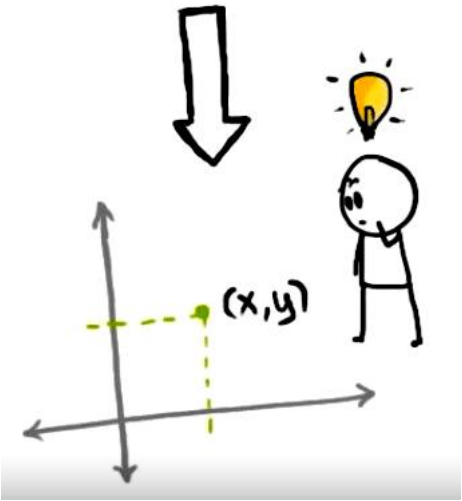


FIG. 1.2.7

Prin urmare curentul electric se schimbă în punctul de contact și funcția ce corespunde acelui punct se efectuează.

Ecranele rezistive sunt durabile și rezistente, dar este mai greu de citit de pe acestea din cauza straturilor multiple care reflectă lumina ambientală. Lumina ambientală este lumina din jur reflectată înapoi de pe ecran.

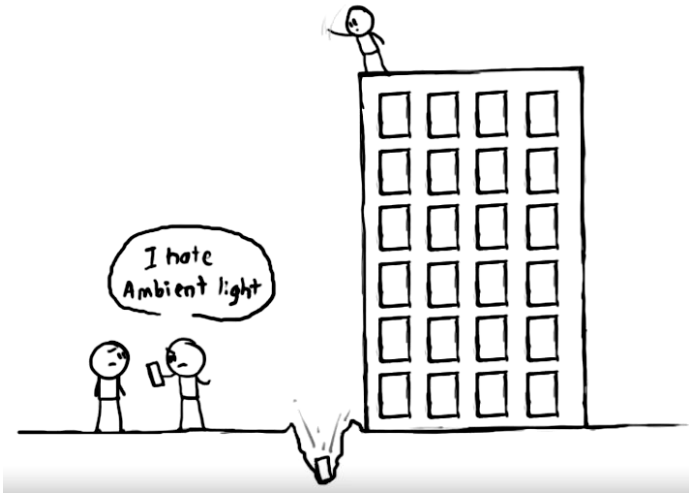


FIG. 1.2.8

Spre deosebire de ecranele rezistive, cele **capacitive** nu folosesc presiunea degetului pentru a face schimbări în fuxul electric. În schimb ele funcționează cu tot ce are o încărcătură electrică, incluzând pielea omului.

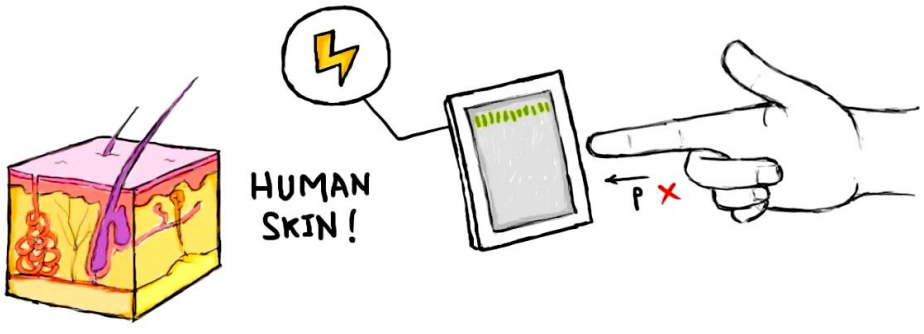


FIG. 1.2.9

Ecranele tactile capacitive sunt făcute folosind materiale precum cuprul sau ITO (indium tin oxide), capabile de a stoca încărcături electrice într-o rețea electrostatică de fire mici (fiecare mai mic decât un fir de păr uman).

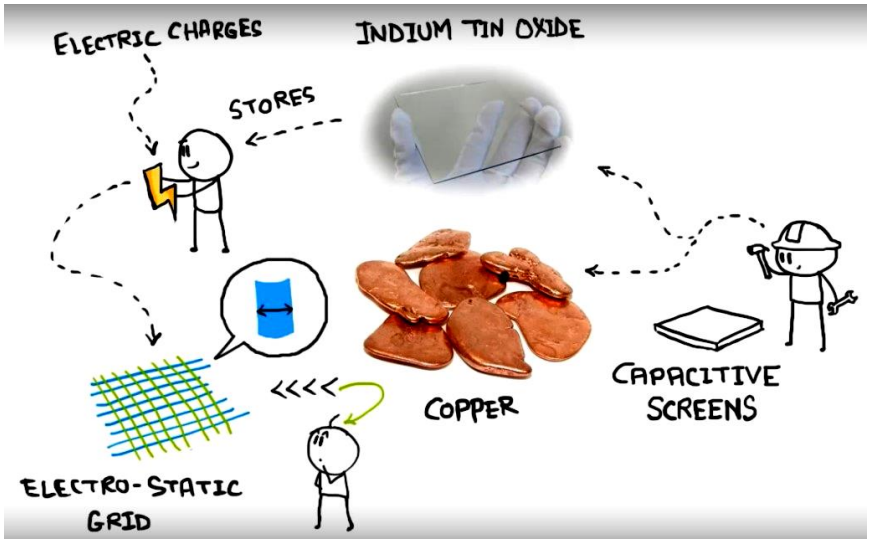


FIG. 1.2.10

Există un substrat de sticlă, un strat conductor, un strat protector, un controler și electrozi la colțuri. Electrozii aplică un voltaj scăzut la stratul conductor, creând un câmp electrostatic uniform. Când un deget atinge ecranul, o mică încărcătură electrică este transferată către deget pentru a închide circuitul. Se creează o cădere de tensiune în acel punct de pe ecran. Locația acestui punct este înregistrată de controler și acesta este modul de funcționare a ecranului tactil capacitiv.

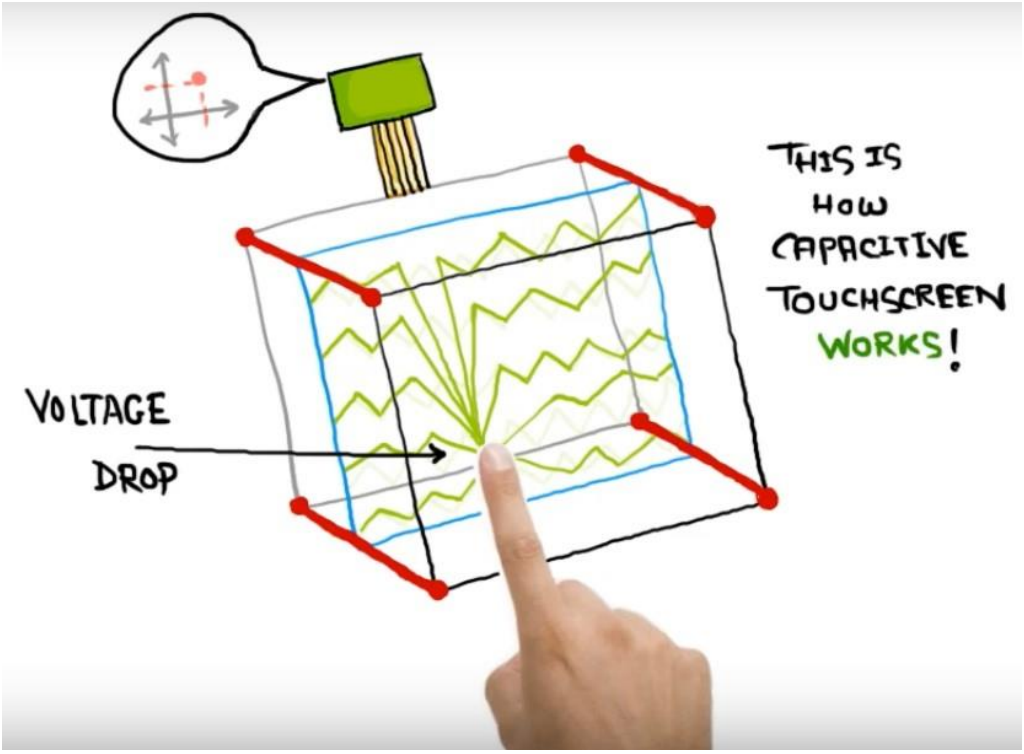


FIG. 1.2.11

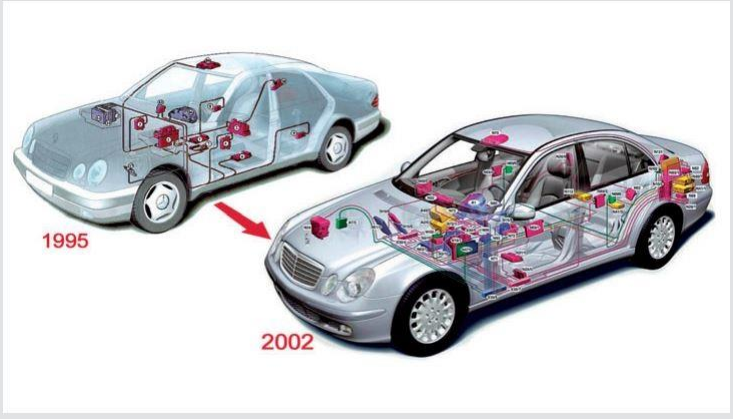
O serie de avantaje privind interfețele tactile:

- Viteză
- Ușor de folosit
- Permite o altă dimensiune a ecranului dispozitivelor
- Accesibilitate
- Durabilitate
- Ușor de curățat
- Reducerea costurilor

1.3. Diagnosticarea la bord și importanța acesteia

Diagnosticarea la bord (OBD on-board diagnostics) este un termen auto care se referă la capacitatea de autodiagnosticare și de raportare a vehiculului. Sistemele OBD dau proprietarului vehiculului sau tehnicianului accesul la starea diferitelor subsisteme de vehicule. Cantitatea de informații de diagnosticare disponibile prin OBD a variat de la introducerea sa în versiunile anterioare din 1980 ale computerelor de la bord. Versiunile anterioare ale sistemului OBD ar activa o lumină indicatoare de funcționare defectuoasă în cazul în care a fost detectată o problemă, dar nu ar furniza nici o informație cu privire la natura problemei. Implementările moderne ale OBD utilizează un port standard de comunicații digitale pentru a furniza date în timp real, pe lângă o serie standardizată de coduri de diagnosticare a defecțiunilor sau DTC-uri (Diagnostic Trouble Code), care permit identificarea și remedierea rapidă a disfuncționalităților din interiorul vehiculului.

Creșterea rapidă a funcțiilor electronice în vehicule din a doua jumătate a anilor 1980 a dus, la început, la multe soluții insulare, care au împiedicat luarea în considerare a conceptelor globale în domeniul arhitecturilor electrice/electronice. La începutul anilor 1990 a început o fază de consolidare care a fost marcată de dezvoltarea structurilor electrice/electronice și a topologiei asociate rețelelor dintr-o perspectivă globală. Aceasta a însemnat că modulele electrice/electronice și rețeaua lor ar putea susține o poziție necontestată în vehicul. Recunoașterea faptului că multe funcții ar putea fi implementate în mod sensibil numai cu ajutorul electronicii au predominat. Deci imaginea electronicii sa transformat din a fi un rău necesar în a fi o cheie pentru noi, interesante și inovatoare funcții. Cele trei noduri de bus ale vehiculelor din 1989 au devenit astăzi mai mult de 70 în vehicule echipate în mod similar.



Development of electronic net-working based on the example of the E-Class model series W210 (1995) and W211 (2002).

FIG. 1.3.1

Software-ul de bază se ridică la aproximativ 10 milioane de linii de cod de programare.

Această tendință nu a fost fără consecințe pentru diagnosticare. Cu douăzeci de ani în urmă, capacitatea de diagnosticare a unei funcții nu a fost luată în considerare până la scurt timp înainte de lansarea producției. Astăzi, funcțiile de diagnosticare de bază există de obicei încă din B-Sample (sistem cu toate funcțiile complete, fără o testare robustă). Manipularea diagnosticului s-a îmbunătățit semnificativ. O condiție prealabilă este că specificarea diagnosticului trebuie mutată la începutul procesului de dezvoltare.

1.4. Prezentarea proiectului MIB ABT W

Proiectul descris în următoarele rânduri este un proiect al grupului VAG (Volkswagen Audi Group) ce urmează a fi scos pe piață anul acesta. Un modul al acestui proiect constituie tema acestei lucrări. Produsul pe care se implementează software-ul este sistemul infotainment din mașină, mai precis afișajul și panoul de control. Ne vom axa în cele ce urmează pe două dintre controlerele de pe acest afișaj: microcontrolerul unui buton tactil capacitiv și controlerul gazdă (host controller).

Motivul acestui proiect este dorința de a înlocui microcontrolerul butonului tactil de pe panoul de control, Cypress PSoC4, cu microcontrolerul Atmel AT-Tiny T1616. Întreaga interacțiune a acestui microcontroler cu controlerul gazdă trebuie să rămână la fel, mai exact driver-ul de controlerului gazdă (host controller) nu trebuie să sufere nici o modificare.

Următorul grafic prezintă sistemul tactil din interiorul dispozitivului ABT și componentele sale constând din senzor, FPC, PCB, microcontroler gazdă, controler tactil de afișare și buton de comandă. Componenta senzor generală conține mai multe elemente : senzor pentru ecranul tactil, o bară de culisare, un buton capacitiv obligatoriu (albastru), care poate să pornească ABT și un buton capacitiv HOME opțional (verde). Ambele butoane capacitive sunt controlate de microcontrolerul butonului tactil, așa cum se arată în figură.

Termeni și prescurtări
ABT Anzeige und Bedienteil (afișajul și panoul de control)
FPC Flexible Printed Circuit
MIB Modularer Infotainment Baukasten
PCB Printed Circuit Board
POR Power-On-Reset
VW Volkswagen

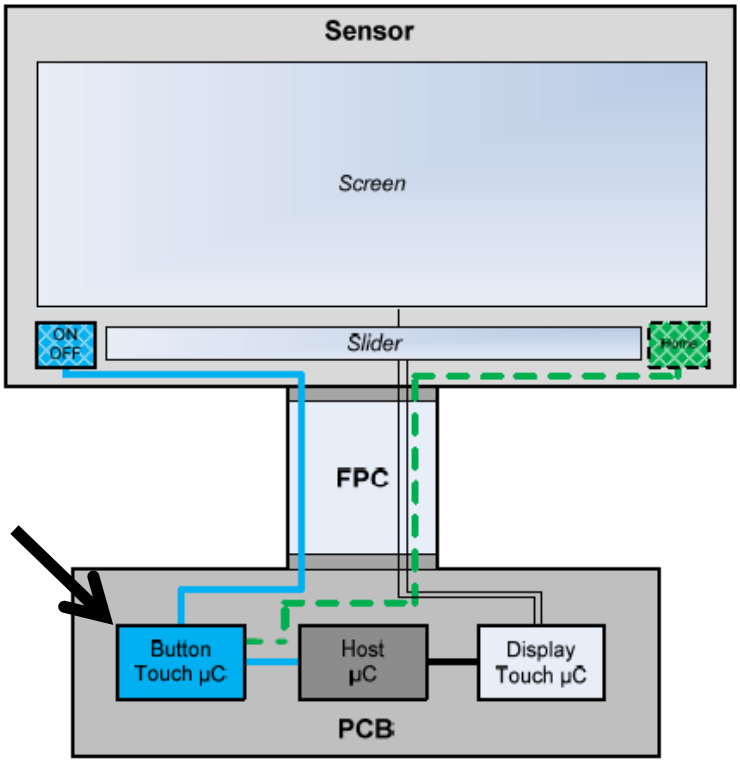


FIG. 1.4.1

În cele ce urmează ne vom axa pe componenta senzorială capabilă să pornească ABT și mai precis, pe microcontrolerul acestei componente. Voi expune o privire de ansamblu al funcționalităților ce se doresc a fi implementate pentru acest proiect.

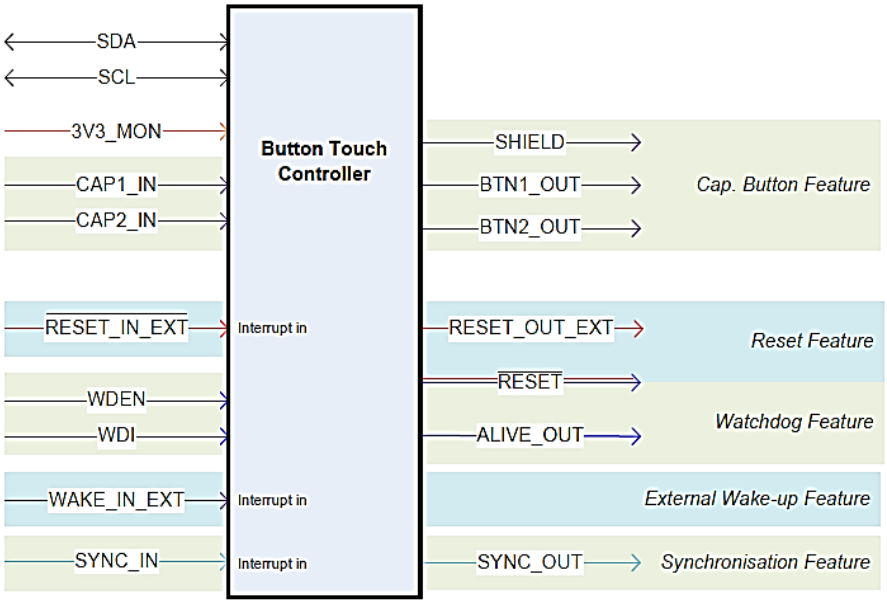


FIG. 1.4.2

SDA	I2C data line
SCL	I2C clock line
3V3_MON	Monitoring the 3V3 voltage supply of the host µC
SHIELD	Active shield driving line
CAPx_IN	Capacitive sensing line connected to the button sensor
BTNx_OUT	Digital output indicating the touch status
ALIVE_OUT	Output signal indicating the alive status of the button touch µC
RESET	Output line to force a reset of the connected host µC
RESET_IN_EXT	Input signal to force a reset of ABT from external control
RESET_OUT_EXT	Output signal to force a reset of external devices
WDEN	Watchdog enable
WDI	Watchdog Input
WAKE_IN_EXT	Input signal to overrule the capacitive sensing and set the button output signal accordingly
SYNC_IN	Input synchronization signal
SYNC_OUT	Output of generated synchronization signal to other components to be syncronized

FIG. 1.4.3

Pe lângă aceste funcționalități descrise mai sus, s-a dorit și implementarea unui **mod diagnoză** a sistemului, descris prin cerințele de mai jos:

- Următoarele două servicii fundamentale de diagnoză trebuie să fie implementate în codul aplicației (I2C bus): readMemoryByAdress**,** writeMemoryByAdress**.**
- readMemoryByAdress va accesa RAM-ul complet, ROM-ul și EEPROM-ul de asemenea.
- writeMemoryByAdress va modifica numai datele EEPROM.
- Ambele servicii de diagnoză trebuie să poată gestiona un număr flexibil de octeți.
- Orice funcție internă, cum ar fi calculul sumelor de control ale datelor EEPROM, se realizează prin scrierea unei valori specifice la o anumită adresă în RAM.
  Prin interogarea adresei RAM, funcția specificată va fi executată cu o întârziere de durată mică.
- Dacă există servicii necesare pentru bootloader, ele trebuie analizate.
- La cerere, trebuie furnizat un număr de versiune software de 4 octeți.
  Trebuie definit conținutul numărului versiunii (de exemplu, bootloader + versiune software de aplicație, câte 2 octeți fiecare).

Acest mod de diagnoză reprezintă tema principală a acestei lucrări, iar implementarea cerințelor de mai sus reprezintă partea aplicativă.

2. Proiectarea și implementarea modulului de diagnoză

2.1. Componenta software

Întregul proiect va fi dezvoltat în limbajul de programare C, fiind cel mai important și folosit limbaj pentru aplicațiile încorporate, în timp real, în industria automobilelor. Acest lucru se datorează, în mare măsură, flexibilității și potențialului său de portabilitate într-o gamă largă de hardware. Motivele specifice pentru utilizare sa:

- Pentru multe din microprocesoarele utilizate, dacă există alt limbaj de programare disponibil în afară de limbajul de asamblare, atunci este de obicei C. În majoritatea cazurilor alte limbaje nu sunt disponibile pentru hardware.
- Complexitatea sporită a aplicațiilor face ca utilizarea unui limbaj de nivel înalt să fie mai adecvat decât limbajul de asamblare.
- C poate genera un cod de dimensiune mai mică decât alte limbaje de nivel înalt

Nici un limbaj de programare nu poate garanta că executabilul final se va comporta așa cum intenționa programatorul. Există o serie de probleme care pot apărea în orice limbaj de programare, ele putând fi clasificate astfel:

- Programatorul face greșeli (ex. este ușor să tastezi "="(atribuire) în locul "=="(comparație logică))
- Programatorul înțelege greșit limbajul (ex. setul de reguli pentru prioritatea operatorilor)
- Compilatorul nu face ceea ce așteaptă programatorul
- Compilatorul conține erori (fiind și el la rândul lui un software)

Pentru a încerca să evităm erorile prezentate mai sus, codul trebuie scris conform regulilor MISRA (The Motor Industry Software Reliability Association) – set de instrucțiuni de dezvoltare a softwareului pentru limbajul de programare C. Obiectivele sale sunt de a facilita siguranța codului, securitatea, portabilitatea și fiabilitatea în contextul sistemelor încorporate.

Mediul de dezvoltare al software-ului este Atmel Studio 7 și am folosit platforma start.atmel.com pentru a genera un proiect nou și pentru a configura componentele software și setările dispozitivului.

2.2. Componenta hardware

Microcontrolerul Atmel-AVR-ATtiny1614-1616-1617

Introducere

Modelul ATtiny1614 / 1616/1617 este membru al seriei microcontrolerelor tinyAVR1, folosind

procesorul AVR® pe 8 biți cu multiplicator de hardware, care rulează cu o frecvență de până la 20MHz și cu memorie Flash 16KB, 2KB de SRAM și 256B de EEPROM, într-o rețea de 14, 20 și 24 de pini. Seria tinyAVR1 utilizează cele mai noi tehnologii cu o arhitectură flexibilă și cu putere redusă, inclusiv sistem de evenimente și SleepWalking, caracteristici analogice exacte și periferice avansate. Interfețele touch capacitive cu senzorul de proximitate și ecranul acționat sunt susținute cu controlerul tactil QTouch® integrat.

Caracteristici

• Procesor
- CPU pe 8 biți AVR®
- Rularea la 20MHz
- Acces I / O cu ciclu unic
- Controlor de întrerupere pe două nivele
- Multiplicatorul hardware cu două cicluri

• Memorii
- Memorie flash auto-programabilă de 16KB în sistem
- 256B EEPROM
- 2KB SRAM

• Sistem
- Resetare la pornire (POR)
- detecția brown-out (BOD)
- Opțiuni de clock intern și extern:
  • Oscilator RC cu putere redusă de 16/20 MHz
  • Oscilator RC intern cu 32,768 kHz Ultra Low Power (ULP) cu precizie de ± 10%, ± 2% mărimea pasului de calibrare
  • Oscilator de cristal extern de 32.768 kHz
  • Intrare clock extern
- Interfață de programare și depanare cu un singur pin (UPDI)
- Trei moduri de sleep:
  • Inactiv cu toate perifericele care rulează pentru trezire imediată
  • Așteptare
    - Funcționare configurabilă a perifericelor selectate
    - periferice SleepWalking

• Power Down cu funcționalitate limitată de trezire
  • Periferice
    - un timer / contor de 16 biți tip A cu registru de perioadă dedicat, 3 canale de comparare (TCA)
    - Două timer / numărător de 16 biți de tip B cu captare de intrare (TCB)
    - un timer / contor de 12 biți tip D optimizat pentru aplicații de control (TCD)
    - contor de timp real (RTC) de 16 biți care rulează de la oscilatorul extern cristal sau intern RC

- USART
- Interfață serial periferică Master / Slave (SPI)
- Master / slave I2C cu potrivire adresă dublă
  - Modul standard (Sm, 100kHz)
  - Modul rapid (Fm, 400kHz)
  - Modul rapid plus (Fm +, 1MHz)
- Configurabil Custom Logic (CCL) cu două tabele de căutare programabile (LUT)
- Trei comparatoare analogice (AC) cu întârziere redusă de propagare
- Două convertoare analoge la digital de 115 biți (ADC) de 10 biți
- Trei convertoare digitale la analogice pe 8 biți (DAC) cu un singur canal extern
- Cinci referințe interne de tensiune selectabile: 0.55V, 1.1V, 1.5V, 2.5V și 4.3V

- Scanarea automată a memoriei CRC
- Timer de supraveghere a ferestrelor (WDT) cu oscilator separat pe cip
- Controler tactil periferic (PTC)
  - Butoane cu atingere capacitivă
  - Trezire la atingere
- întreruperea externă a tuturor pinilor cu scop general

• I / O și pachete:
- 12 până la 22 de linii I / O programabile
- SOIC150 cu 14 pini
- 20-pini QFN 3x3 și SOIC300
- 24-pin QFN 4x4

• Domenii de temperatură:
- -40 ° C până la 105 ° C
- -40 ° C până la 125 ° C Dispozitiv de temperatură Opțiuni disponibile

• Clasamente de viteză:
- 0-5MHz @ 1.8V - 5.5V
- 0-10MHz @ 2.7V - 5.5V
- 0-20MHz @ 4.5V - 5.5V

Pinout

24-pin QFN

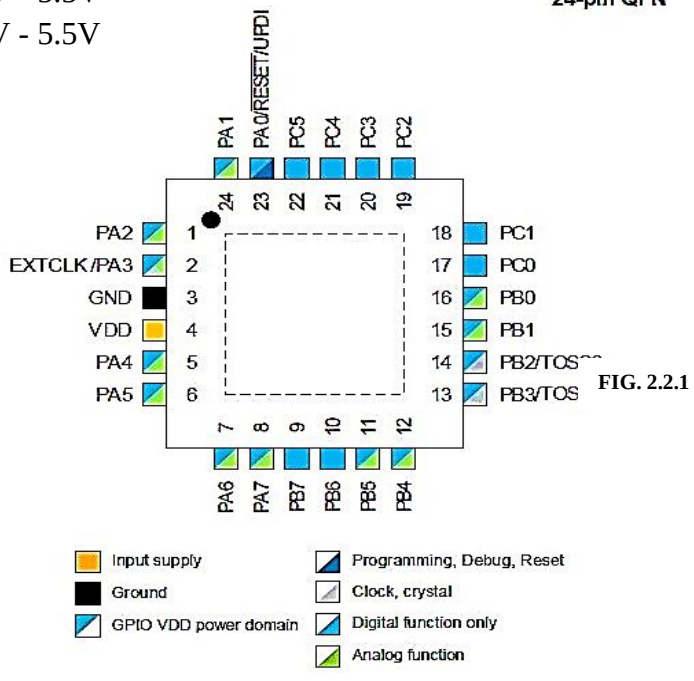


FIG. 2.2.1

Block Diagram

ATtiny1614/1616/1617 Block Diagram

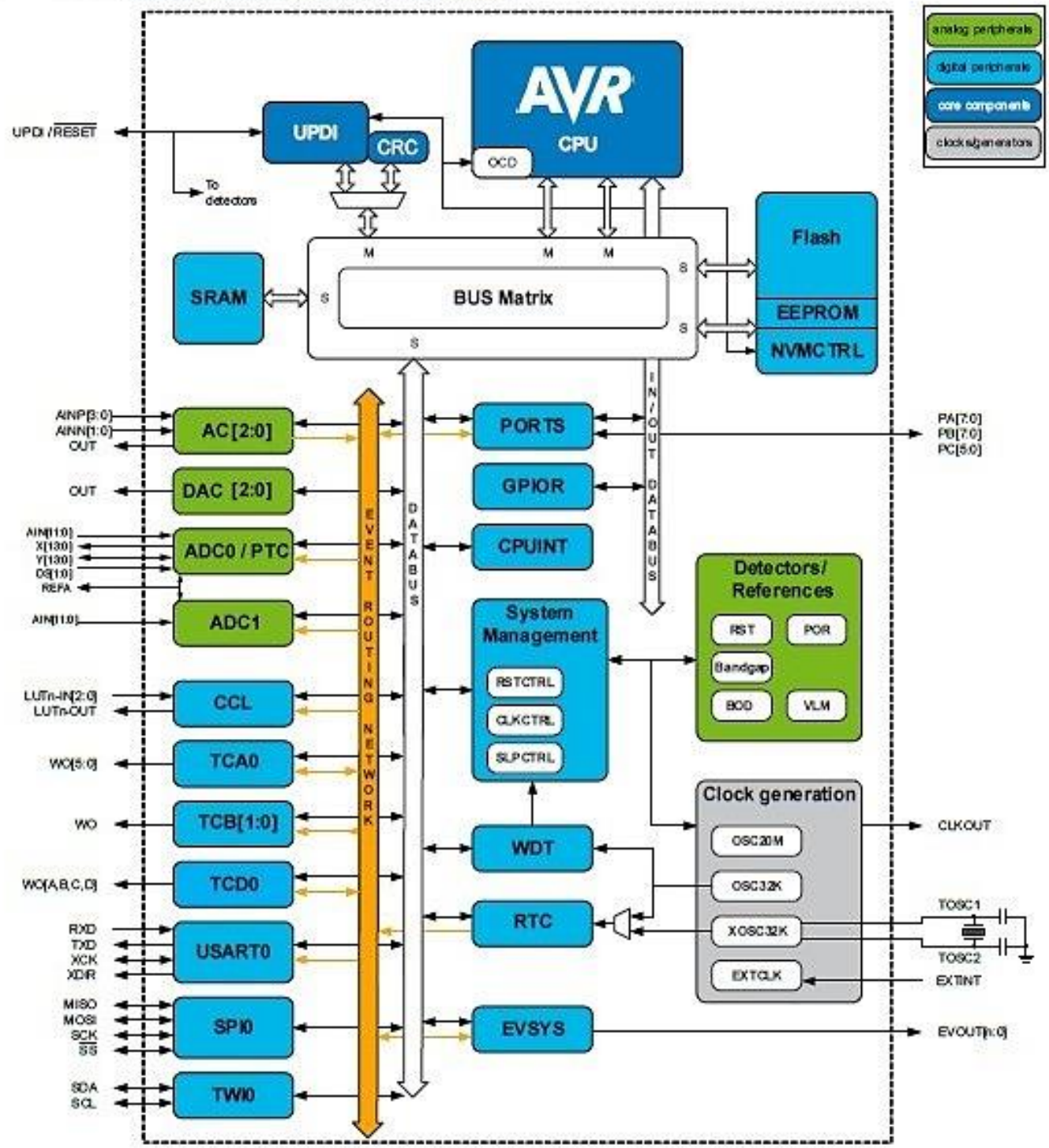


FIG. 2.2.2

Atmel At-Tiny – Xplained Board

Primul hardware folosit pentru dezvoltarea acestui software a fost această placă de dezvoltare Atmel At-Tiny Xplained board.

Descriere

Kitul de evaluare Atmel® ATtiny817 Xplained Pro este o platformă hardware pentru evaluarea microcontrolerului ATtiny817.
Suportat de platforma de dezvoltare integrată Atmel Studio, kitul oferă acces ușor la caracteristicile Atmel800 și explică modul de integrare a dispozitivului într-un design personalizat.
Seturile de evaluare pentru seria Xplained Pro MCU includ un debugger încorporat și nu sunt necesare instrumente externe pentru a programa sau depana ATtiny817.
Seturile de extensie Xplained Pro oferă periferice suplimentare pentru a extinde caracteristicile plăcii și facilitarea dezvoltării unui design personalizat.

Caracteristici

• Microcontrolerul ATtiny817
• Două butoane mecanice
• Două butoane QTouch®
• Cristal de 32.768kHz
• Debugger încorporat
    - Identificare automată pentru identificarea plăcii în Atmel Studio
    - Un LED galben
    - Un LED de power-on
    - Depistarea simbolică a tipurilor de date complexe, inclusiv informații despre domeniul de aplicare
    - Programare și depanare, inclusiv măsurători de putere
    - Interfață pentru datele de intrare: SPI, I2C, două GPIO-uri
    - Portul COM Virtual (CDC)
• Circuite de măsurare a curentului încorporat, cu suport Atmel Data Visualizer pentru vizualizarea datelor
• USB alimentat
• Suportat cu exemple de aplicații în programul Atmel Start

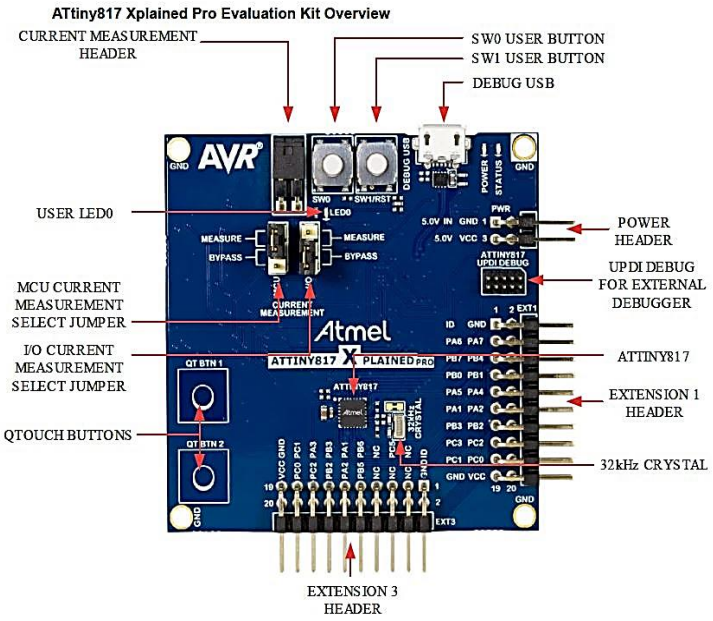


FIG. 2.2.3

Observație: Pentru a putea dezvolta întreaga funcționalitate cerută de proiect, a fost înlocuit microcontrolerul ATtiny817 (Flash memory 8K) cu Attiny1616 (Flash memory 16K).

Hardware-ul proiectului

Reprezentarea schematică a întregului mediu de lucru necesar pentru dezvoltarea software-ului folosind hardware-ul proiectului.

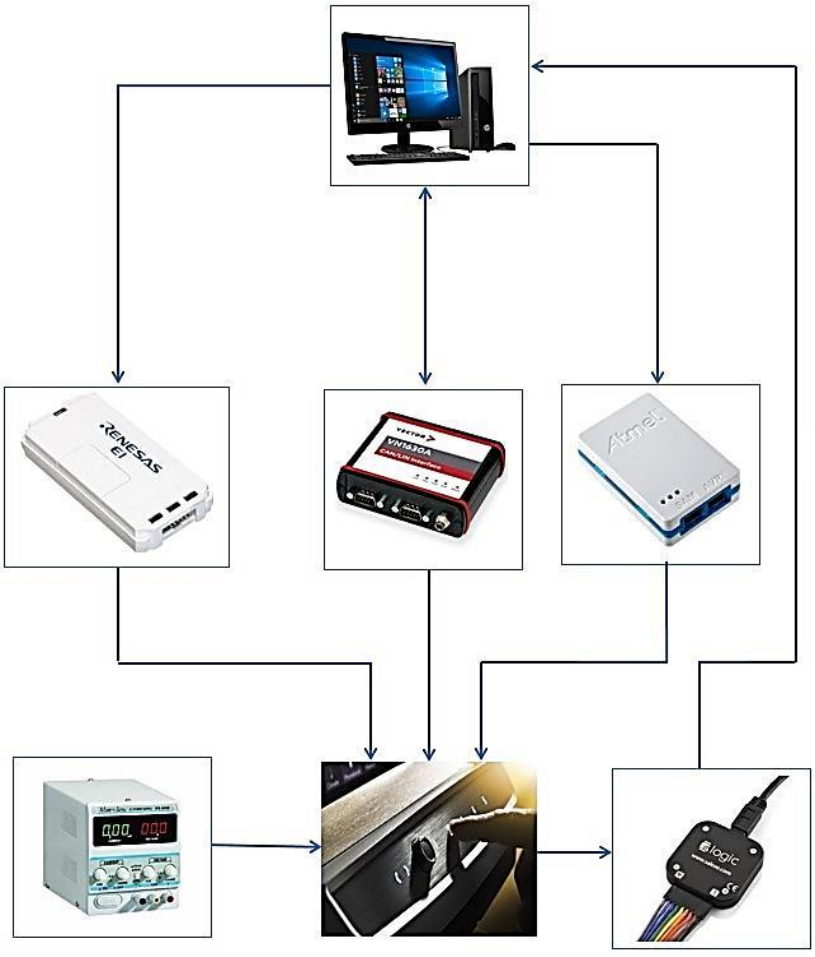


FIG. 2.2.4

2.3. Prezentare generală TWI

Comunicarea este realizata între un Host Controler (Master) și At-Tiny (Slave) prin intermediul interfeței TWI (Two Wire Interface) ce este echivalentul I2C-ului.

Interfața TWI este o interfață de comunicare bidirecțională, cu două fire. Acesta este compatibilă cu I2C și System Management Bus (SMBus). Singurul hardware extern necesar pentru implementarea magistralei este un pull-up rezistor pe fiecare linie de magistrală. Orice dispozitiv conectat la magistrala trebuie să acționeze ca un master sau un slave. Masterul inițiază o tranzacție de date prin adresarea unui slave pe magistrală și comunicând dacă dorește să

Block Diagram

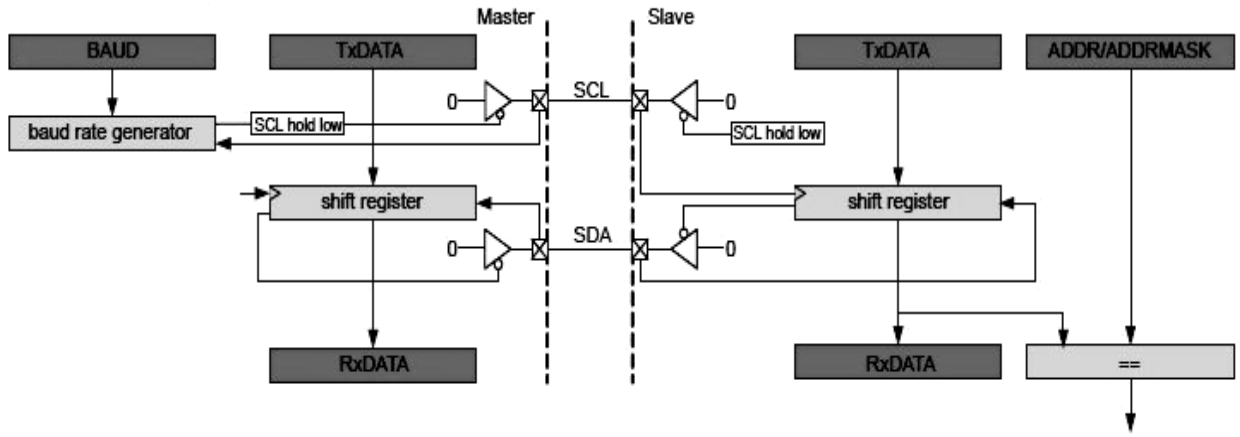


FIG. 2.3.1

transmită sau să primească date. O magistrală poate avea mai mulți slave și unul sau mai mulți masteri care pot prelua controlul magistralei. Un proces de arbitraj se ocupă de prioritate dacă mai mult de un master încearcă să transmită date în același timp. Mecanismele de rezolvare a conflictelor de pe magistrală sunt inerente protocolului. Perifericul TWI suportă funcționalitatea master și slave. Funcțiile master și slave sunt separate una de alta și pot fi activate și configurate separat. Modulul master acceptă funcționarea și arbitrajul cu magistrală multi-master. Acesta conține generatorul ratei de transfer. Toate frecvențele de bus 100kHz, 400kHz și 1MHz sunt acceptate. Comanda rapidă și modul inteligent pot fi activate pentru a declanșa automat operații și reducerea complexității software-ului. Modulul slave implementează potrivirea adreselor pe 7 biți și recunoașterea generală a apelurilor de adresă în hardware. Adresarea pe 10 biți este, de asemenea, acceptată. Slave-ul continuă să funcționeze în toate modurile de sleep, inclusiv în modul power-down. Aceasta permite slave-ului să trezească dispozitivul din toate modurile de

Signal Description

Signal	Description	Type
SCL	Serial clock line	Digital I/O
SDA	Serial data line	Digital I/O

FIG. 2.3.2

repaus.
Perifericul TWI va detecta condițiile START și STOP, coliziunile cu magistrala și erorile din magistrală. Arbitrajul pierdut, erorile, coliziunile și menținerea clock-ului pe magistrală sunt, de asemenea, detectate și indicate în flag-uri separate de stare disponibile atât în modul master, cât și în modul slave. Acest dispozitiv oferă o instanță a dispozitivului periferic TWI, TWI0.

Descriere funcțională

Concepte generale TWI Bus

Circuitul TWI este o metodă simplă și eficientă de interconectare a mai multor dispozitive pe o magistrală de serie. Un dispozitiv conectat la magistrala poate fi un master sau un slave, în cazul în care master-ul controlează magistrala și toată comunicarea.

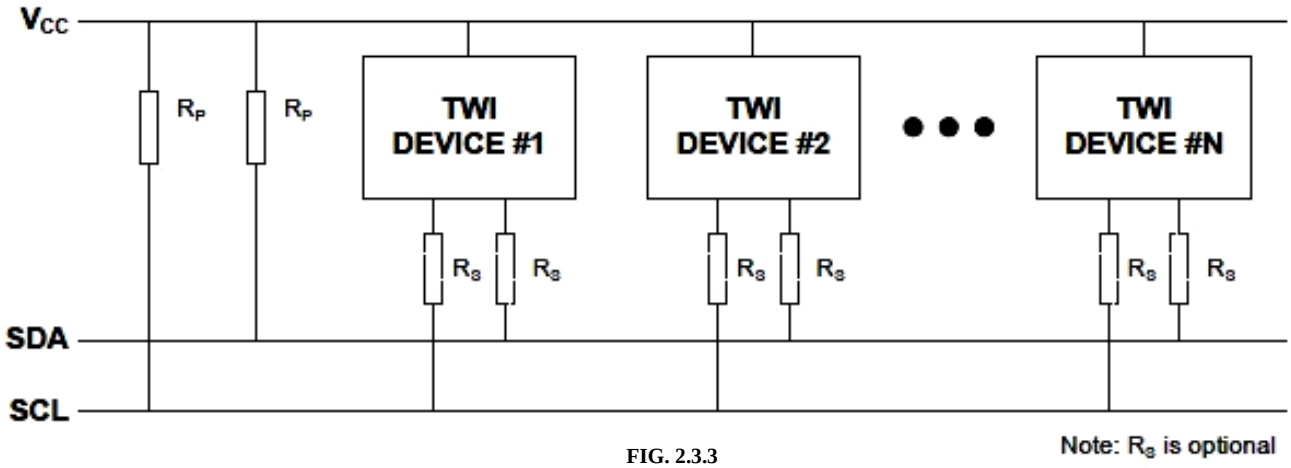


FIG. 2.3.3

Topologie magistralei TWI

O adresă unică este atribuită tuturor dispozitivelor slave conectate la magistrala, iar master-ul va folosi aceasta pentru a adresa un slave și pentru a iniția o tranzacție de date. Mai mulți masteri pot fi conectați la aceeași magistrală, numită mediu multi-master. Este prevăzut un mecanism de arbitraj pentru rezolvarea accesului bus-ului în rândul masterilor, deoarece numai un singur dispozitiv master poate controla bus-ul în orice moment. Un dispozitiv poate conține atât logică master cât și slave și poate emula mai multe dispozitive slave răspunzând la mai multe adrese. Un master indică începerea unei tranzacții prin emiterea unei condiții START (S) pe magistrala.

Se trimite un pachet de adresă cu o adresă slave (ADDRESS) și o indicație dacă masterul dorește să citească sau să scrie date (R / W). După transferarea tuturor pachetelor de date (DATA), masterul emite o condiție STOP (P) pe magistrală pentru a încheia tranzacția. Receptorul trebuie să confirme (A) sau să nu recunoască (A) fiecare octet primit.

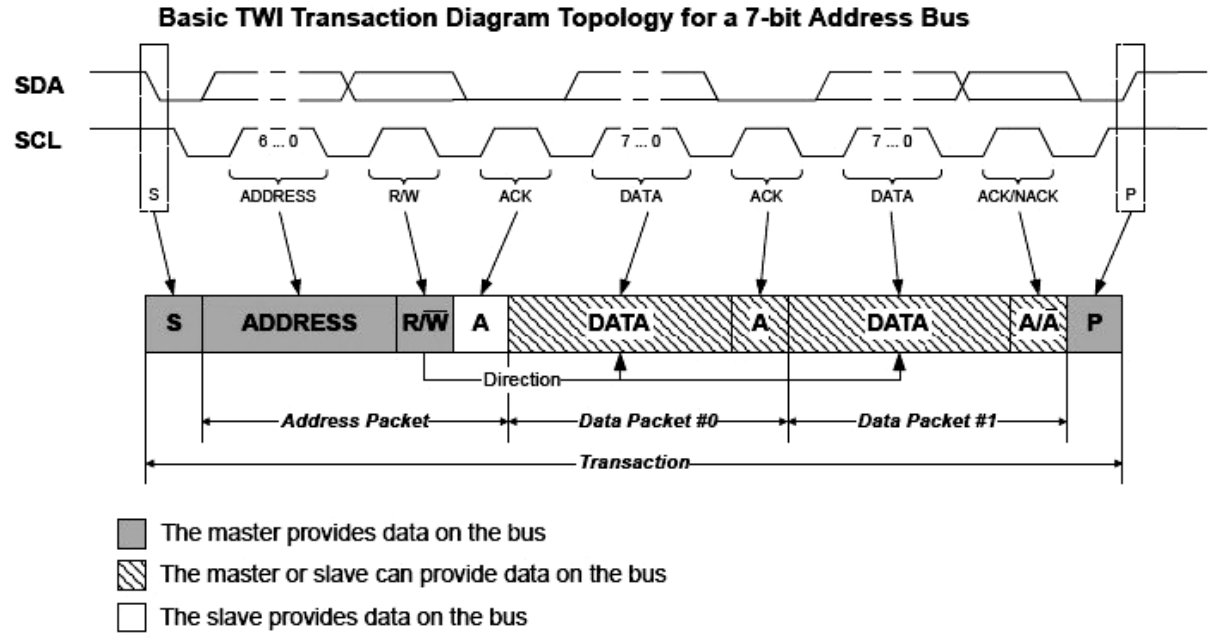


FIG. 2.3.4

Inițializare

Pentru a porni TWI ca Master, scrieți un '1' bitului ENABLE din registrul Master Control A (TWI.MCTRLA), urmată de scrierea adresei slave în registrul Adresa Master (TWI.MADDR). Registrul TWI.MADDR are, de asemenea, un bit R / W care indică dacă masterul transmite sau primește. Registrul de date (TWI.MDATA) este scris în cazul în care masterul transmite date. Pentru a activa TWI-ul ca Slave, scrieți Adresa Slave (ADDR) în TWI.SADDR și scrieți un '1' la Bit ENABLE în registrul Slave Control A (TWI.SCTRLA). Perifericul TWI va aștepta să primească un octet adresat acestuia.

Condiții START și STOP

Două condiții unice sunt utilizate pentru marcarea începutului (START) și terminarea (STOP) a unei tranzacții. Master-ul emite o condiție START (S) prin indicarea unei tranziții de la linia SDA până la linia SCL în timp ce linia SCL este menținută ridicată. Master-ul încheie tranzacția

emițând o condiție STOP (P), indicată printr-o tranziție de la low-to-high pe linia SDA, în timp ce linia SCL este menținută ridicată.

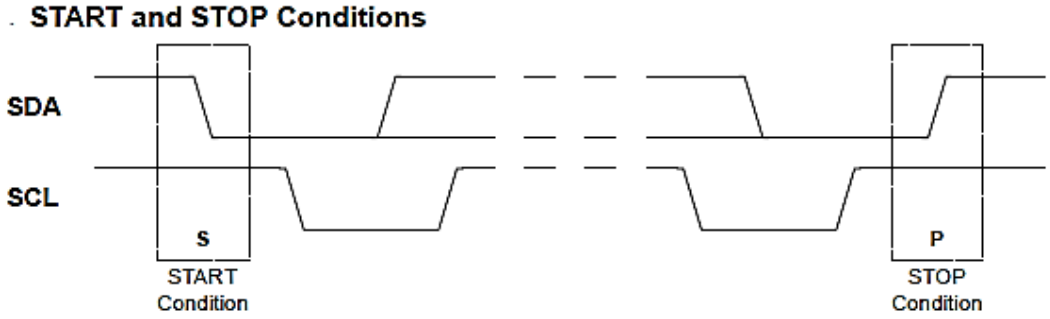


FIG. 2.3.5

În timpul unei singure tranzacții pot fi emise condiții START multiple. O condiție START care nu este urmată în mod direct o condiție STOP se numește o condiție START repetată (Sr).

Bit Transfer

După cum este ilustrat în figură, un bit transferat pe linia SDA trebuie să fie stabil pentru întreaga perioadă înaltă a liniei SCL. În consecință, valoarea SDA poate fi modificată numai în perioada low a clock-ului. Acest lucru este asigurat în hardware de către modulul TWI.

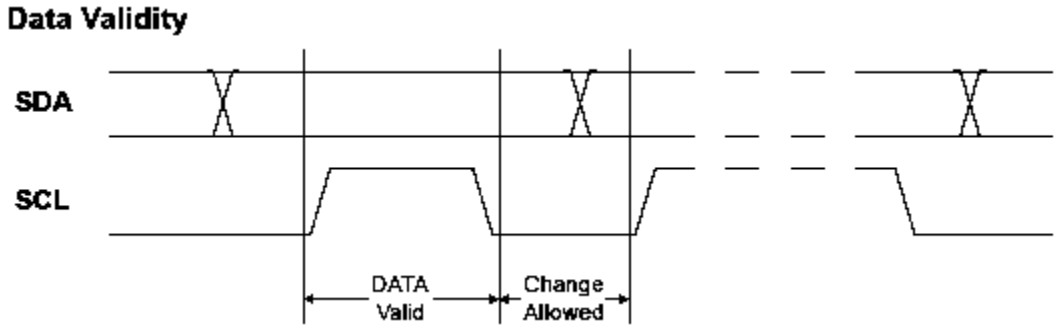


FIG. 2.3.6

Combinarea transferurilor de biți duce la formarea adreselor și a pachetelor de date. Aceste pachete constau din opt biți de date (un octet) cu cel mai semnificativ bit transferat mai întâi, plus un singur bit de not-acknowledge (NACK) sau acknowledge (ACK). Aparatul adresat semnalează ACK trăgând linia SCL în low în timpul celui de-al nouălea ciclu de clock și semnalizează NACK lăsând linia SCL ridicată.

Pachetul de adrese

După condiția START, este trimisă o adresă pe 7 biți urmată de un bit de citire / scriere (R / W). Acest lucru este întotdeauna transmis de master. Un slave care își recunoaște adresa va confirma (ACK) adresa prin tragerea liniei de date în low pentru următorul ciclu SCL, în timp ce ceilalți

slavei ar trebui să păstreze liniile TWI high și să aștepte următoarea START și adresă. Adresa, bitul R / W și bitul de confirmare, combinat este pachetul de adresă. Este permis numai un singur pachet de adresă pentru fiecare condiție START, de asemenea, atunci când este utilizată o adresare pe 10 biți. Bitul R / W specifică direcția tranzacției. Dacă bitul R / W este low, acesta indică o tranzacție de scriere din partea masterului, iar masterul va transmite datele după ce slave-ul și-a confirmat adresa. Dacă bitul R / W este high, acesta indică o tranzacție de citire din partea masterului, iar slave-ul va transmite datele sale după confirmarea adresei sale.

Pachetul de date

Un pachet de adresă este urmat de unul sau mai multe pachete de date. Toate pachetele de date sunt lungi de nouă biți, constând dintr-un octet de date și un bit de confirmare. Bitul de direcție din pachetul de adrese anterior determină direcția în care sunt transferate datele.

Tranzacţie

O tranzacție este transferul complet de la START la starea STOP, inclusiv orice condiții START repetate între ele. Standardul TWI definește trei moduri de tranzacție fundamentale: master write, master read și o tranzacție combinată. Figura de mai jos ilustrează tranzacția de scriere principală. Masterul inițiază tranzacția emițând o condiție START (S) urmată de un pachet de adresă cu bitul de direcție setat la zero (ADDRESS + W).

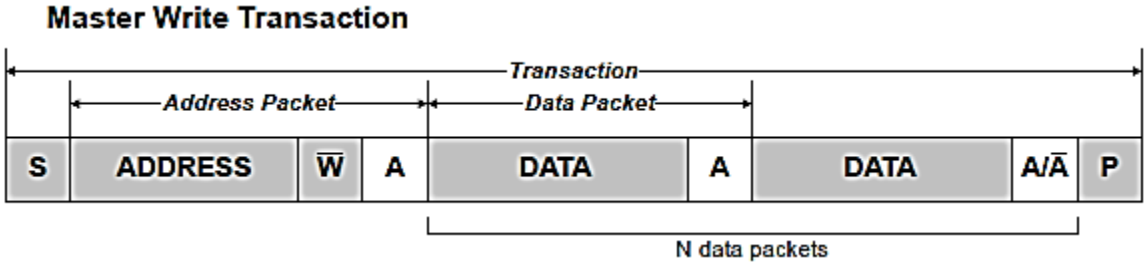


FIG. 2.3.7

Presupunând că slave-ul recunoaște adresa, masterul poate începe transmiterea datelor (DATA) iar slave-ul va ACK sau NACK (A / !A) fiecare octet. Dacă nu trebuie transmise pachete de date, masterul termină tranzacția emițând o condiție STOP (P) imediat după pachetul de adresă. Nu există restricții la numărul de pachete de date care pot fi transferate. Dacă slave-ul semnalează un NACK la date, masterul trebuie să presupună că slave-ul nu mai poate recepționa date și termină tranzacția.

Figura de mai jos ilustrează tranzacția de citire principală. Masterul inițiază tranzacția prin emiterea unei condiții START, urmată de un pachet de adresă, cu un bit de direcție setat la unu (ADDRESS + R). Slave-ul adresat trebuie să confirme adresa pentru ca masterul să aibă permisiunea de a continua tranzacția.

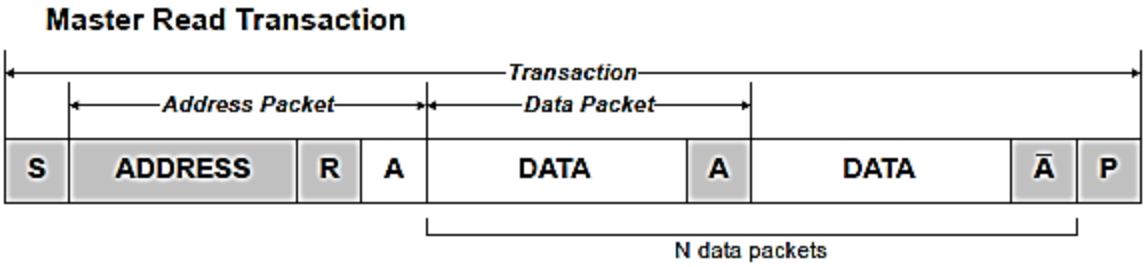


FIG. 2.3.8

Presupunând că slave-ul recunoaște adresa, masterul poate începe să primească date de la slave. Nu există nici o limitare a numărului de pachete de date care pot fi transferate. Slaveul transmite datele în timp ce masterul semnalează ACK sau NACK după fiecare octet de date. Maestrul încheie transferul cu un NACK înainte de a emite o condiție STOP.

Figura de mai jos ilustrează o tranzacție combinată. O tranzacție combinată constă în mai multe tranzacții de citire și scriere separate prin condiții START repetate (Sr).

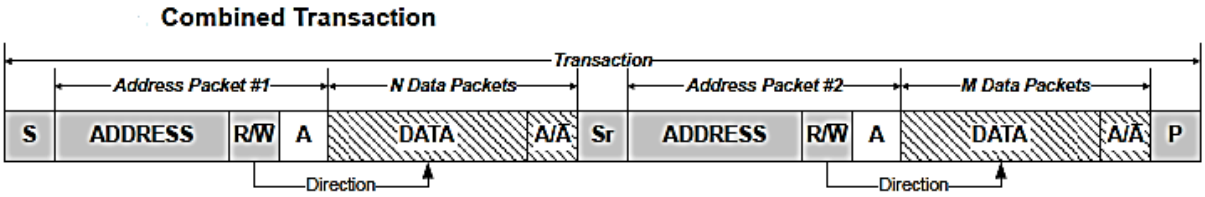


FIG. 2.3.9

Arbitraj

Un master poate porni o tranzacție numai dacă a detectat că magistrala este inactivă. Deoarece magistrala TWI este o magistrală multi-master, este posibil ca două dispozitive să poată iniția o tranzacție în același timp. Acest lucru are ca rezultat mai mulți masteri care accesează magistrala în același timp. Acest lucru este rezolvat folosind o schemă de arbitraj în care masterul pierde controlul asupra magistralei dacă nu este capabil să transmită un nivel ridicat pe linia SDA. Masterii care pierd arbitrajul trebuie să aștepte până o condiție STOP înainte de a încerca să recâștige arbitrarea busului. Dispozitivele slave nu sunt implicate în procedura de arbitraj.

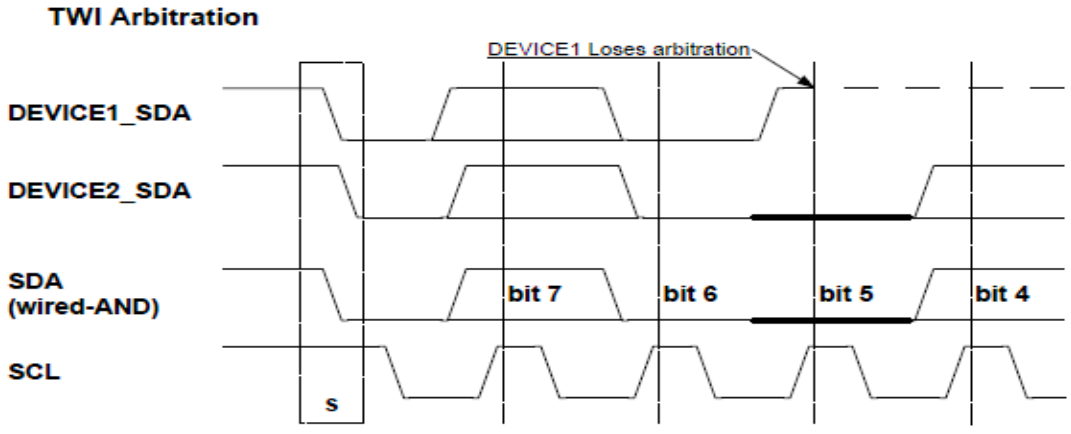


FIG. 2.3.10

Figura de mai sus arată un exemplu în care doi masteri TWI se luptă pentru a iniția o tranzacție pe bus. Ambele dispozitive pot emite o condiție START, dar DEVICE1 pierde arbitrajul atunci când încearcă să transmită un nivel ridicat (bitul 5), în timp ce DEVICE2 transmite un nivel scăzut.
Arbitrajul dintre o condiție START repetată și un bit de date, o condiție STOP și un bit de date sau o stare START repetată și o stare STOP nu sunt permise și vor necesita o manipulare specială prin software-ul.

Sincronizare
Un algoritm de sincronizare a clock-ului este necesar pentru rezolvarea situațiilor în care mai mult de un master încearcă să controleze linia SCL în același timp. Figura de mai jos arată un exemplu în care doi masteri concurează pentru controlul clock-ului. Linia SCL este rezultatul operației AND al celor două ieșiri de clock ale masterilor.

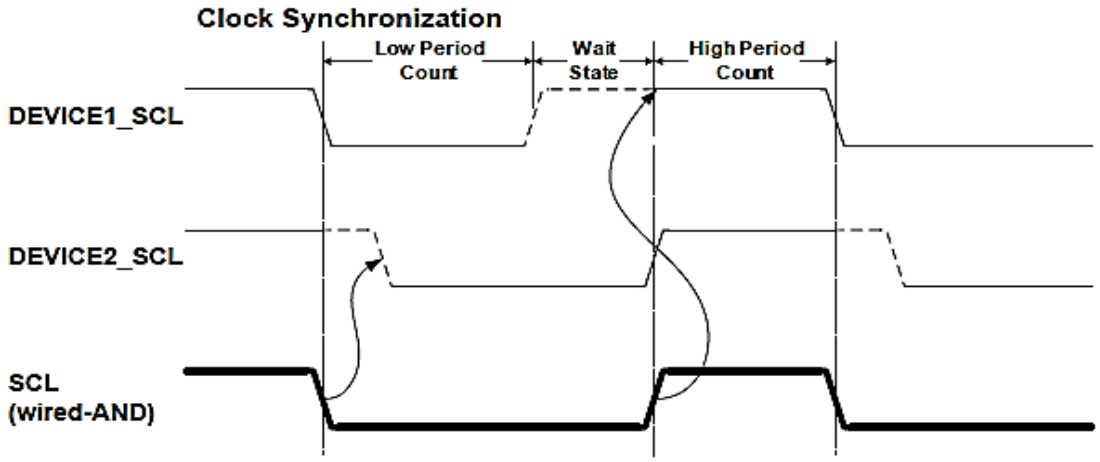


FIG. 2.3.11

O tranziție de la linia SCL va duce la scăderea liniei pentru toți masterii de pe magistrală și vor începe să își cronometreze perioada de clock. Durata de sincronizare a perioadei de clock poate varia între masteri.
Când un master (DEVICE1 în acest caz) și-a finalizat perioada redusă, acesta eliberează linia SCL. Cu toate acestea, linia SCL nu va fi ridicată până când toți masterii nu au eliberat-o. În consecință, linia SCL va fi menținută de dispozitivul cu cea mai lungă perioadă de timp (DEVICE2). Dispozitivele cu perioade scurte de low trebuie să introducă o stare de așteptare până când clock-el este eliberat. Toți masterii își încep perioada high când linia SCL este eliberată de toate dispozitivele și a crescut. Dispozitivul care își încheie prima perioadă de timp înaltă (DEVICE1) forțează linia de clock - low și procedura se repetă. Rezultatul este că dispozitivul cu cea mai scurtă perioadă de clock determină perioada high, în timp ce perioada low a clock-ului este determinată de dispozitivul cu cea mai lungă perioadă de clock.

Operațiunea TWI Slave

Slave-ul TWI este byte-oriented cu întreruperi opționale după fiecare byte. Există date separate ale slave-ului și flag-uri de întrerupere de adresa /stop. Flag-urile de întrerupere pot fi de asemenea utilizate pentru operațiuni de întreținere. Există flag-uri de stare dedicate pentru a indica recepția ACK / NACK, clock hold, coliziunea, eroarea busului și direcția de citire / scriere.
Când este setat un flag de întrerupere, linia SCL este forțată să scadă. Acest lucru va oferi timp slave-ului să răspundă sau să manipuleze datele și, în majoritatea cazurilor, va necesita interacțiuni software. Modul de recunoaștere a adresei poate fi activat pentru a permite slave-ului să răspundă la toate adresele recepționate.

Primirea pachetelor de adrese

Atunci când modulul slave TWI este configurat corespunzător, va aștepta detectarea unei condiții START. Când se întâmplă acest lucru, octetul de adresă succesiv va fi recepționat și verificat de logica de potrivire a adresei, iar slave-ul va activa o adresă corectă și va stoca adresa în registrul de date. Dacă adresa primită nu se potrivește, slave-ul nu va confirma și nu va stoca adresa și va aștepta o nouă condiție START. Semnalul de întrerupere adresă / stop slave este setat când se detectează o condiție START reușită de un octet de adresă valid.

O condiție START imediat urmată de o condiție STOP este o operație ilegală și este setat flag-ul de eroare de magistrală. Flag-ul direcției R / W reflectă bitul de direcție primit cu adresa. Acest lucru poate fi citit de software pentru a determina tipul de operațiune în curs de desfășurare. În funcție de starea bitului de direcție R / W, apare unul dintre cele patru cazuri distincte (S1 până la S4) urmând pachetul de adresă. Diferitele cazuri trebuie tratate în software.

Cazul S1: Acceptarea pachetului de adrese - setarea setului de direcții

Dacă este setat flag-ului direcției R / W, aceasta indică o operație de citire principală. Linia SCL este forțată de către slave să cadă, întinzând clock-ul busului. Dacă ACK-ul este trimis de slave, hardware-ul slave va seta semnalul de întrerupere a datelor indicând că datele sunt necesare pentru transmitere. Datele, START repetate sau STOP pot fi primite după aceasta. Dacă NACK este trimis de slave, slave-ul va aștepta o nouă condiție START și o potrivire a adresei.

Cazul S2: Acceptarea pachetului de adrese

Dacă indicatorul de direcție R / W este șters, aceasta indică o operație de scriere master. Linia SCL este forțată să scadă, întinzând clock-ul busului. Dacă ACK-ul este trimis de slave, slave-ul va aștepta recepția datelor. Datele, START repetate sau STOP pot fi primite după aceasta. Dacă este trimis NACK, slave-ul va aștepta o nouă stare START și o potrivire a adresei.

Cazul S3: Coliziune

Dacă slave-ul nu este capabil să trimită un nivel înalt sau NACK, flag-ul de coliziune este setat și va dezactiva datele și ieșirea de confirmare din logica slave. Clock-ul este eliberat. Este acceptată o stare START sau o condiție START repetată.

Cazul S4: Starea STOP primită

Când se primește condiția STOP, se va seta flag-ul de adresa / stop al slave-ului, indicând faptul că a apărut o condiție STOP și nu o potrivire a adresei.

Primirea pachetelor de date

Slave-ul va ști când a fost recepționat cu succes un pachet de adresă cu bitul de direcție R / W setat pe 0. După ce a confirmat acest lucru, slave-ul trebuie să fie pregătit să primească date. Când se recepționează un pachet de date, flag-ul de întrerupere a datelor este setat, iar slave-ul trebuie să indice ACK sau NACK. După indicarea unui NACK, slave-ul trebuie să aștepte o stare STOP sau o stare repetată START.

Transmiterea pachetelor de date

Slave-ul va ști când a fost recepționat cu succes un pachet de adresă cu bitul de direcție R / W setat pe 1. Apoi, poate începe trimiterea datelor prin scrierea în registrul de date slave. Când este finalizată transmisia de pachete de date, este setat flag-ul de întrerupere a datelor. Dacă masterul indică NACK, slave-ul trebuie să oprească transmiterea datelor și să aștepte o stare STOP sau o stare repetată START.

Întreruperi

Available Interrupt Vectors and Sources

Offset	Name	Vector Description	Conditions
0x00	Slave	TWI Slave interrupt	• DIF: Data Interrupt Flag in SSTATUS set • APIF: Address or Stop Interrupt Flag in SSTATUS set
0x02	Master	TWI Master interrupt	• RIF: Read Interrupt Flag in MSTATUS set • WIF: Write Interrupt Flag in MSTATUS set

FIG. 2.3.12

Când se produce o condiție de întrerupere, semnalul de întrerupere corespunzător este setat în registrul Master (TWI.MSTATUS) sau Registrul de stare Slave (TWI.SSTATUS). Atunci când mai multe condiții de cerere de întrerupere sunt suportate de un vector de întrerupere, cererile de întrerupere sunt ORed împreună într-o cerere de întrerupere combinată către controlerul de întrerupere. Utilizatorul trebuie să citească registrul INTFLAGS al perifericului pentru a determina care dintre condițiile de întrerupere sunt prezente.

Name: SSTATUS
Offset: 0x0B
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	DIF	APIF	CLKHOLD	RXACK	COLL	BUSERR	DIR	AP
Access	R/W	R/W	R	R	R/W	R/W	R	R
Reset	0	0	0	0	0	0	0	0

Bit 7 – DIF: Data Interrupt Flag
This flag is set when a slave byte transmit or byte receive operation is successfully completed without any bus error. The flag can be set with an unsuccessful transaction in case of collision detection (see description of the COLL status bit). Writing a '1' to its bit location will clear the DIF. However, normal use of the TWI does not require the DIF flag to be cleared by using this method, since the flag is automatically cleared when:

1. Writing to the slave DATA register.
2. Reading the slave DATA register.
3. Writing a valid command to the CTRLB register.

The DIF flag can be used to generate a slave data interrupt (see description of the DIEN control bit in TWI.CTRLA).

Bit 6 – APIF: Address or Stop Interrupt Flag
This flag is set when the slave address match logic detects that a valid address has been received or by a stop condition. Writing a '1' to its bit location will clear the APIF. However, normal use of the TWI does not require the flag to be cleared by this method since the flag is cleared using same software interactions as described for the DIF flag.

The APIF flag can be used to generate a slave address or stop interrupt (see description of the AIEN control bit in TWI.CTRLA). Take special note of that the slave stop interrupt shares the interrupt vector with slave address interrupt.

Bit 5 – CLKHOLD: Clock Hold
If read as '1', the slave clock hold flag indicates that the slave is currently holding the TWI clock (SCL) low, stretching the TWI clock period. This is a read only bit that is set when an address or data interrupt is set. Resetting the corresponding interrupt will indirectly reset this flag.

Bit 4 – RXACK: Received Acknowledge
This bit is read only and contains to the most recently received acknowledge bit from the master.

Bit 3 – COLL: Collision
If read as '1', the transmit collision flag indicates that the slave has not been able to transmit a high data or NACK bit. If a slave transmit collision is detected, the slave will commence its operation as normal, except no low values will be shifted out onto the SDA line (i.e., when the COLL flag is set to '1' it disables the data and acknowledge output from the slave logic). DIF flag will be set to '1' at the end as a result of internal completion of unsuccessful transaction. Similarly when collision occurs because slave is not been able to transmit NACK bit, it means address match already happened and APIF flag is set as a result. APIF/DIF flags can only generate interrupt whose handlers can be used to check for the collision. Writing a '1' to its bit location will clear the COLL flag. However, the flag is automatically cleared if any START condition (S/Sr) is detected.

This flag is intended for systems where address resolution protocol (ARP) is employed. However, a detected collision in non-ARP situations indicates that there has been a protocol violation and should be treated as a bus error.

Bit 2 – BUSERR: Bus Error
The BUSERR flag indicates that an illegal bus condition has occurred. An illegal bus condition is detected if a protocol violating start (S), repeated start (Sr), or stop (P) is detected on the TWI bus lines. A start condition directly followed by a stop condition is one example of protocol violation. Writing a '1' to its bit location will clear the BUSERR flag. However, normal use of the TWI does not require the BUSERR to be cleared by this method. A robust TWI driver software design will assume that the entire packet of data has been corrupted and restart by waiting for a new START condition (S). For bus errors to be detected, the master must be enabled (ENABLE bit in TWI.MCTRLA is '1'), and the system clock frequency must be at least four times the SCL frequency.

Bit 1 – DIR: Read/Write Direction
This bit is read only and indicates the current bus direction state. The DIR bit reflects the direction bit value from the last address packet received from a master TWI device. If this bit is read as '1', a master read operation is in progress. Consequently a '0' indicates that a master write operation is in progress.

Bit 0 – AP: Address or Stop
When the TWI slave address or stop interrupt flag (APIF) is set, this bit determines whether the interrupt is due to address detection or a stop condition.

Value	Name	Description
0	STOP	A stop condition generated the interrupt on APIF.
1	ADR	Address detection generated the interrupt on APIF.

FIG. 2.3.13

2.4. Diagnoza: mod de implementare

- Ne bazăm pe 3 întreruperi principale
    - Întreruperea de Adresă
    - Întreruperea de Date
    - Întreruperea de Stop
- Două automate de stare
    - Unul ce interpretează comenzile de scriere de la slave la master X
    - Unul ce interpretează comenzile de citire de la slave la master
- Funcții:
    - Inițializare I2C
    - Pregătire comunicare
    - Funcția de întrerupere
    - Manipulare întrerupere adresă
    - Manipulare întrerupere date
    - Manipulare întrerupere STOP
    - Citirea informațiilor dispozitivului
    - Citirea stării butonului
    - Resetarea registrului pentru controlul întreruperilor
    - Calcul sumă de control
    - Stocare sumă de control
    - Furnizare sumă de control
    - Scriere bloc 126 bytes (126, deoarece ultimii doi bytes sunt alocați sumei de control)
    - Citire bloc de 128 bytes
    - Salt în bootloader
    - Salt în aplicație (software reset)
    - Citire I2C
    - Scriere I2C
    - Trimite confirmare
    - Tranzacție completă

  Enumerație:

    - Adresele comenzii masterului

Pentru a iniția o comunicare, masterul trebuie întotdeauna sa trimită o comandă de write la început, comandă ce va avea în componența ei adresa slave-ului. Dacă masterul va începe cu o comandă de read, atunci vom notifica o eroare.

După ce am identificat că masterul dorește să se adreseze slave-ului nostru, vom aștepta altă secvență în care masterul o să indice dacă dorește să scrie sau să citească.

În cazul în care acesta dorește să scrie, imediat după secvența de adresă, slave-ul preia datele trimise de master și le prelucrează.

Dacă masterul dorește să scrie, slave-ul așteaptă secvența de date trimisă de master, după care execută comanda primită de la acesta.

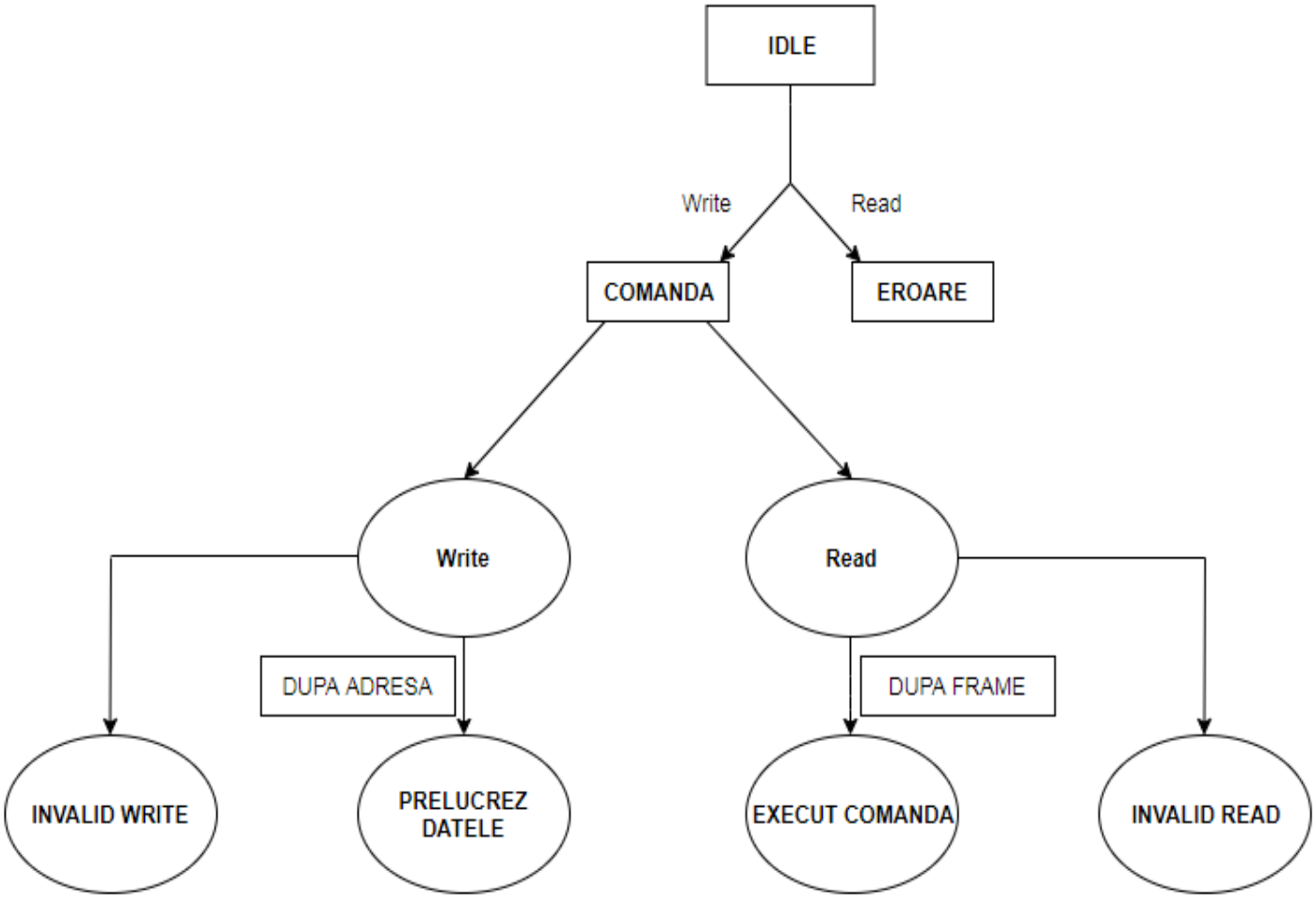


FIG. 2.4.1

Register Summary - TWI

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0				SDASETUP	SDAHOLD[1:0]		FMPEN	
0x01	Reserved									
0x02	DBGCTRL	7:0								DBGRUN
0x03	MCTRLA	7:0	RIEN	WIEN		QCEN	TIMEOUT[1:0]		SMEN	ENABLE
0x04	MCTRLB	7:0					FLUSH	ACKACT	CMD[1:0]	
0x05	MSTATUS	7:0	RIF	WIF	CLKHOLD	RXACK	ARBLOST	BUSERR	BUSSTATE[1:0]	
0x06	MBAUD	7:0	BAUD[7:0]							
0x07	MADDR	7:0	ADDR[7:0]							
0x08	MDATA	7:0	DATA[7:0]							
0x09	SCTRLA	7:0	DIEN	APIEN	PIEN			PMEN	SMEN	ENABLE
0x0A	SCTRLB	7:0						ACKACT	CMD[1:0]	
0x0B	SSTATUS	7:0	DIF	APIF	CLKHOLD	RXACK	COLL	BUSERR	DIR	AP
0x0C	SADDR	7:0	ADDR[7:0]							
0x0D	SDATA	7:0	DATA[7:0]							
0x0E	SADDRMASK	7:0	ADDRMASK[6:0]							ADDREN

FIG. 2.4.2

Inițializare I2C

- Setarea adresei slaveului: scriem Adresa Slave (ADDR) în TWI.SADDR .
- Activarea I2C: portul Multiplexer (PORTMUX) poate fie să activeze sau să dezactiveze funcționalitatea pinilor.

Register Summary - PORTMUX

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0			LUT1	LUT0		EVOUT2	EVOUT1	EVOUT0
0x01	CTRLB	7:0				TWI0		SPI0		USART0
0x02	CTRLC	7:0			TCA05	TCA04	TCA03	TCA02	TCA01	TCA00
0x03	CTRLD	7:0							TCB1	TCB0

FIG. 2.4.3

- Pentru a activa I2C avem trebuie să setăm bitul corespunzător TWI0 în PORTMUX.CTRLB .
- Setarea timpului SDA la 8 ciclii: scriem 1 pe bitul 4 al TWI0.CTRLA
- Setare registru SCTRLA cu :
    - DIEN: Data Interrupt Enable
    - APIEN: Address or Stop Interrupt Enable
    - PIEN: Stop Interrupt Enable
    - PMEN: Address Recognition Mode
    - SMEN: Smart Mode Enable
    - ENABLE: Enable TWI Slave
- Apelare Tranzacție completă

Tranzacție completă

- Setare TWI0.SCTRLB cu valoarea 0xddddd010.

Bit	7	6	5	4	3	2	1	0
						ACKACT	CMD[1:0]	
Access						R/W	R/W	R/W
Reset						0	0	0

The ACKACT bit is not a flag or strobe, but an ordinary read/write accessible register bit.

Value	Name	Description
0	ACK	Send ACK
1	NACK	Send NACK

Command Settings

CMD[1:0]	DIR	Description
0x0 - NOACT	X	No action
0x1	X	Reserved
0x2 - COMPTRANS	Used to complete a transaction.	
	0	Execute Acknowledge Action succeeded by waiting for any START (S/Sr) condition.
	1	Wait for any START (S/Sr) condition.
0x3 - RESPONSE	Used in response to an address interrupt (APIF).	
	0	Execute Acknowledge Action succeeded by reception of next byte.
	1	Execute Acknowledge Action succeeded by slave data interrupt.
	Used in response to a data interrupt (DIF).	
	0	Execute Acknowledge Action succeeded by reception of next byte.
	1	Execute a byte read operation followed by Acknowledge Action.

The acknowledge action bits and command bits can be written at the same time.

FIG. 2.4.4

Funcția de întrerupere

- Verificăm ce fel de întrerupere a apărut
  - **De adresă**
  - **De date**
  - **De stop**

  apelând corespunzător fiecărei întreruperi, funcție ce o manipulează.

Manipulare întrerupere adresă

- Are un parametru ce indică direcția comunicării (citire sau scriere)
- Apelăm funcția **Pregătire comunicare** cu parametrul funcției de manipulare întrerupere adresă
- Verificăm direcția de comunicare (master vrea să scrie sau să citească)
    - Dacă master vrea să citească:
        - Setăm starea sistemului ca fiind I2C_SendToMaster
        - Verificăm dacă adresa slave a fost setată; True -> Apelăm funcția automatului ce interpretează comenzile de scriere de la slave la master
    - Dacă master vrea să scrie:
        - Setăm starea sistemului ca fiind I2C_ReceiveFromMaster
    - Trimitem confirmare.

Manipulare întrerupere date

- Verificăm dacă în bufferul de I2C mai avem spațiu disponibil
    - Dacă această condiție este îndeplinită avem următoarele cazuri (switch):
        - I2C_SendToMaster : Apelăm funcția de scriere I2C și incrementăm indexul bufferului.
        - I2C_ReceiveFromMaster : Dacă adresa slave-ului nu a fost salvată, o salvam și setăm un flag corespunzător altfel apelăm funcția de citire I2C și incrementăm indexul bufferului.
    - Dacă această condiție nu este îndeplinită setăm starea sistemului în idle.
    - Trimitem confirmare.

Manipulare întrerupere STOP

- Apelăm funcția de tranzacție completă
- Dacă starea sistemului este I2C_ReceiveFromMaster și adresa slave-ului a fost salvată apelăm funcția automatului ce interpretează comenzile de citire de la slave la master.
- Setăm starea sistemului în idle

Pregătire comunicare

- Resetăm indexul bufferului
- Dacă slave-ul este pregătit să recepționeze, setăm un flag corespunzător

Trimite confirmare

Setare TWI0.SCTRLB cu valoarea 0xddddd011.

Bit	7	6	5	4	3	2	1	0
						ACKACT	CMD[1:0]	
Access						R/W	R/W	R/W
Reset						0	0	0

The ACKACT bit is not a flag or strobe, but an ordinary read/write accessible register bit.

Value	Name	Description
0	ACK	Send ACK
1	NACK	Send NACK

Command Settings

CMD[1:0]	DIR	Description
0x0 - NOACT	X	No action
0x1	X	Reserved
0x2 - COMPTRANS	Used to complete a transaction.	
	0	Execute Acknowledge Action succeeded by waiting for any START (S/Sr) condition.
	1	Wait for any START (S/Sr) condition.
0x3 - RESPONSE	Used in response to an address interrupt (APIF).	
	0	Execute Acknowledge Action succeeded by reception of next byte.
	1	Execute Acknowledge Action succeeded by slave data interrupt.
	Used in response to a data interrupt (DIF).	
	0	Execute Acknowledge Action succeeded by reception of next byte.
	1	Execute a byte read operation followed by Acknowledge Action.

The acknowledge action bits and command bits can be written at the same time.

FIG. 2.4.5

Automatul ce interpretează comenzile de scriere de la slave la master

- Avem următoarele cazuri în funcție de comanda primită de la master:
  - SlaveReadDeviceInfo
    - Apelăm funcția de citire a informațiilor dispozitivului
  - SlaveProvideCheckSum
    - Citim din EEPROM ultimii doi bytes şi îi punem în bufferul de I2C
  - SlaveGetButtonData
    - Apelăm funcția de citire a stării butonului
  - SlaveGetBlockBytes
    - Apelăm funcția de citire a unui bloc de 128 bytes
  - SlaveGetConfigBytes
    - Not aplicable
  - SlaveGetDataFromAdress
    - Not aplicable

Automatul ce interpretează comenzile de citire de la slave la master

- Avem următoarele cazuri în funcție de comanda primită de la master:
  - SlaveCalculateChackSum
    - Apelăm funcția de citire a unui bloc de 128 bytes (întregul EEPROM)
    - Apelăm funcția de stocare a sumei de control
  - SlaveSetBlockBytes
    - Scriem un bloc de date în EEPROM (126 bytes de date + 2 bytes sumă de control)
  - SlaveSoftwareReset
    - Apelăm funcția de salt la începutul adresei aplicației
  - SlaveStartFlash
    - Apelăm funcția de salt în bootloader
  - SlaveSetConfigBytes
    - Not aplicable
  - SlaveSetDataToAddress
    - Not aplicable

Citirea informațiilor dispozitivului

- Preluăm numărul versiunii de software al bootloaderului şi al aplicației în bufferul de I2C prin atribuire directă, deoarece aceste variabile sunt definite în cod

Citirea stării butonului

- Din modulul senzorului tactil preluăm informații referitoare la starea butonului, ce ne indica dacă acesta este apăsat sau nu.

Calcul sumă de control

- Această funcție are doi parametrii, unu este bufferul – pointer, iar celălalt este dimensiunea bufferului.
- Suma de control este calculată printr-un xor între toți bytes bufferului

Stocare sumă control

- Se realizează printr-o scriere în EEPROM pe ultimele două poziții ale memoriei
- Variabila folosită pentru calculul sumei este pe 16 biți, astfel că pe peultima poziție din EEPROM scriem MSB al sumei, iar pe ultima poziție, scriem LSB.

Furnizare sumă control

- Este operația inversă a stocării, din EEPROM citim ultimii boi bytes și îi transmitem pe I2C

Scriere bloc 126 de bytes

- Apelăm o interfață Atmel de scriere în EEPROM care are trei parametrii și anume: adresa de la care să se scrie, bufferul și numarl de bytes

Citirea unui bloc de 128 de bytes

- Este operația inversă scrierii, se folosește o interfață Atmel, dar pentru citire având trei parametrii: adresa de la care se citește, bufferul în care se citește și numărul de bytes doriți să se citească

Salt în bootloader

- Dezactivăm toate întreruperile
- Resetăm regiștrii de întrerupere
- Salt indirect la adresa bootloaderului

Salt în aplicație

- Cu ajutorul unui software reset se va face saltul din bootloader în aplicație
- Setând un bit în registrul RSTCTRL

Name: SWRR
Offset: 0x01
Reset: 0x00
Property: Configuration Change Protection

Bit	7	6	5	4	3	2	1	0
								SWRE
Access	R	R	R	R	R	R	R	R/W
Reset	0	0	0	0	0	0	0	0

Bit 0 – SWRE: Software Reset Enable
When this bit is written to '1', a software reset will occur.

This bit will always read as '0'.

FIG. 2.4.6

Funcția de scriere și funcția de citire a I2C-ului

- Aceste funcții au fost implementate folosind registrul de date SDATA
- Pentru scriere pe I2C am pus date byte cu byte în acest registru
- Pentru citirea de la I2C am preluat datele byte cu byte din acest registru

3. Testare

3.1. Mediul de testare

Sistemul testat este sistemul SW pentru controlerul Atmel AT-Tiny T1616, cu privire la serviciul de diagnoză, comunicarea pe magistrala I2C.

Se va test funcționalitatea modulului Diag pentru a vedea dacă se comportă așa cum ne dorim și pentru detectarea eventualelor bug-uri introduse nevoit în timpul implementării.

Există două tipuri de teste:

- teste manuale (folosind placa de dezvoltare): cu ajutorul unui dispozitiv ce simulează masterul (TRACII Telos), trimitem anumite comenzi prin I2C (Observație: necesită rezistor pull-up). Acest mod de testare s-a folosit la începutul dezvoltării modulului, deoarece hardware-ul proiectului nu era disponibil.
- teste automate: se trimit comenzi prin intermediul magistralei I2C la dispozitivul AT-Tiny și se observă modul în care sistemul reacționează verificând semnalele și flag-urile corespunătoare.

Există un singur sistem SW disponibil pentru AT-Tiny și varianta corespunzătoare este utilizată pe baza hardware-ului testat:

- platforma de dezvoltare (sistemul AT-Tiny 1617 eXplained Pro și sistemul SW cu BLOGIC_nHwUsed macro care conține valoarea BLOGIC_nAtTinyExpBoard).

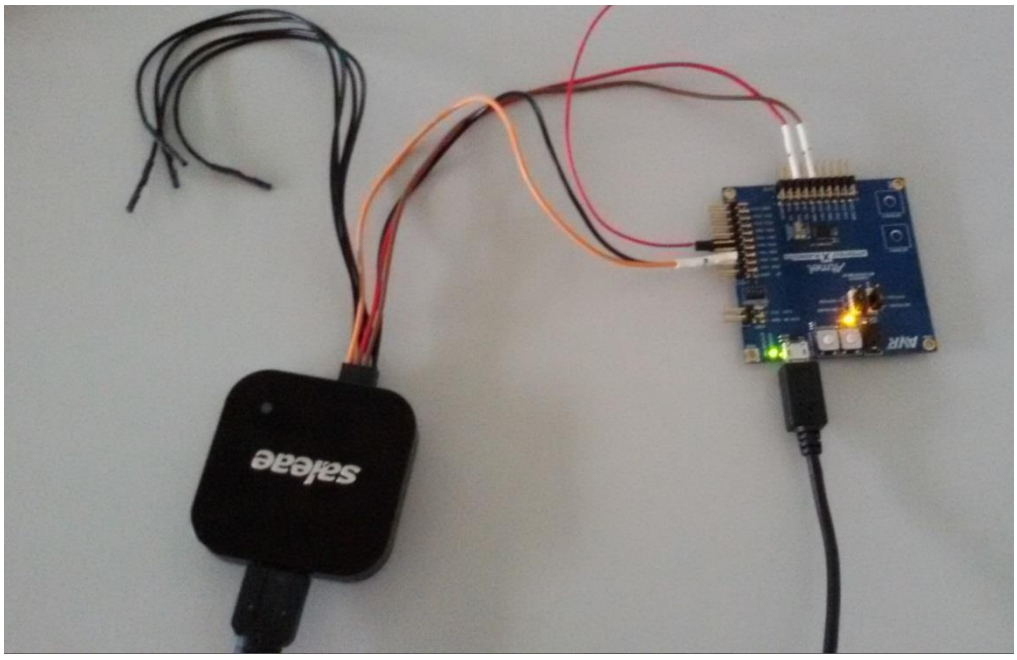


FIG. 3.1.1

- proiectul VW_MIB_ABT_W (versiunea PCB hw B0301 și sistemul sw cu BLOGIC_nHwUsed macro care conține valoarea BLOGIC_nPCBSample_H20).

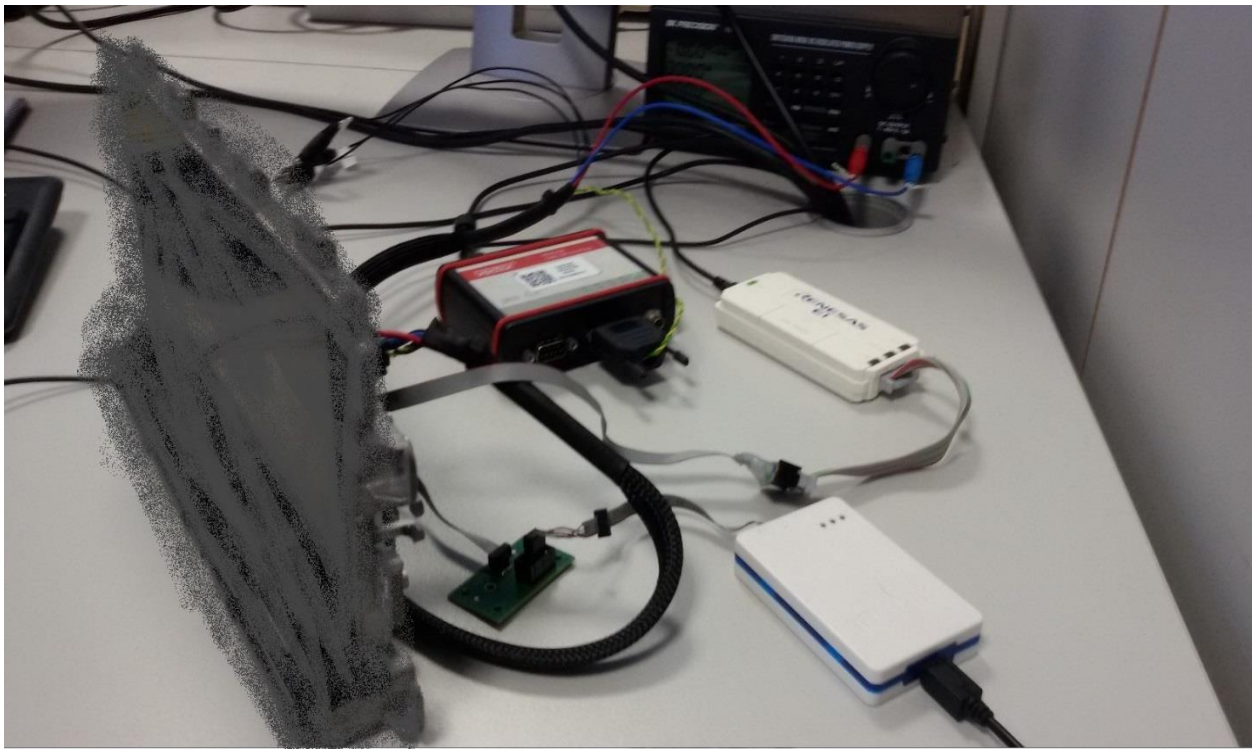


FIG. 3.1.2

Instrumente hardware
• Generator de semnal
• Analizor logic Saleae (sau osciloscop)
• Atmel ICE debugger și adaptor
• E1 debugger și adaptor
• Can case

Instrumente software
• Atmel Visual Studio 7
• Instrumentul Saleae LLC
• Green Hills Multi 2000 (MULTI_V850_7_1_4_2017_1_5)
• Visual Studio 2013
• Vector CANoe 8.2
• Telos I2C Studio

3.2. Testarea modulului de diagnoză

Pentru a testa modulul de diagnoză am folosit harware-ul proiectului. Aceste teste sunt făcute automat prin simularea unei comunicații reale din mașină cu ajutorul unu CAN case.

Exemplu de cerințe ale proiectului:

- Testarea comenzii de read memory
- Testarea comenzii de write memory
- Testarea comenzii de a furniza informații despre dispozitiv
- Testarea comenzii de a sări în bootloader
- Testarea comenzii de calcul al sumei de control
- Testarea stării butonului

În tabele de mai jos sunt descrise testele făcute pentru acest modul de diagnoză.

TestCaseID	Test Case Name	Prio	Input	Expected result
DIAG1	Read CS device information: HW ID, FW ID	1	1. Start Canoe 2. From Canoe, press key 'd' to start the test. 3. From Canoe, press key 'D' to request response.	In Canoe trace, check message with id 0x60: - the first two bytes will contain the boot-loader sw version and application sw version (check against BLOGIC_nBL_SOFTWARE_VERSION and BLOGIC_nSOFTWARE_VERSION_NUMBER defines)
DIAG2	Write a block of 126 bytes configuration data to CS device	1	1. Start Canoe 2. Start MULTI debugger for MIB and ATMEL ICE debugger. 3. From Canoe, press key 'w' to start the test. 4. From Canoe, press key 'W' to request response.	Stop (Pause) the ATMEL debugger and check the EEPROM memory for the 126 bytes and compare the content with the content of this array CAPSENSETST_au8WriteConfig from MULTI debugger.
DIAG3	Read the raw button data bytes from CS device	1	1. Start Canoe. 2. Start MULTI debugger for MIB and ATMEL ICE debugger. 3. From Canoe, press key 't' to start test. 4. From Canoe, press key 'T' to request response.	In Canoe trace, check message with id 0x60: - The first 2 bytes from the received message must be equal with the value of *(ptc_qtlib_node_stat1[sensor_node].node_acq_signals);* -> add this in ATMEL debugger watch.
DIAG4	Read a block of 128 bytes from CS device	1	1. Start Canoe. 2. Start MULTI debugger for MIB and ATMEL ICE debugger. 3. From Canoe, press key 'b' to start the test. 4. From Canoe, press key 'B' to get the response.	Stop (Pause) the ATMEL debugger and check the EEPROM memory for the 128 bytes and compare the content with the content of this array CAPSENSETST_au8StoreConfigBytes from MULTI debugger.
DIAG5	SW reset of the CS device	1	1. Start Canoe. 2. Start MULTI debugger for MIB and ATMEL ICE debugger. 3. Put a breakpoint where the function (on AT-Tiny side): IICB_vJumpToAppStartAddress (); is called. 4. From Canoe, press key 'r' to start the test.	The breakpoint is reached.

FIG. 3.2.1

TestCaseID	Test Case Name	Prio	Input	Expected result
DIAG6	Start Flash	1	1. Start Canoe. 2. Start MULTI debugger for MIB and ATMEL ICE debugger. 3. Put a breakpoint where the function (on AT-Tiny side): IICB_vStartFlash(); is called. 4. From Canoe, press key 'f' to start the test.	The breakpoint is reached.
DIAG7	Checksum	1	1. Start Canoe. 2. Write AT-Tiny eeprom: 2a. From Canoe, press key 'w'. This will send a command to AT-Tiny to write to eeprom. 2b. From Canoe, press key 'W'. This will check to see if the eeprom was written successfully. 2c. Make a power off/power on reset 3. From Canoe, press key 'c'. This command will ask AT-Tiny to calculate the checksum on data stored in eeprom. 4. From Canoe, press key '5'. This command will get the checksum from AT-Tiny. 5. From Canoe, press key 'C'. This command will get the response.	In Canoe trace, check message with id 0x60: - first two bytes represent the checksum calculated by test module - next two bytes represent the checksum calculated by AT-Tiny Both checksums must be identical in order for the test to pass.
DIAG8a	Invalid read command	1	1. Start Canoe. 2. From Canoe, press key 'x' (this will send a read command that is not recognized by At-Tiny). 3. From Canoe, press key 'd' to start the test. 4. From Canoe, press key 'D' to request response.	In Canoe trace, the message with id 0x60 will contain device information (same output as for DIAG1). With this test we prove that the state machine is still ok and commands are interpreted correctly by At-Tiny.
DIAG8b	Invalid write command	1	1. Start Canoe. 2. From Canoe, press key 'X' (this will send a write command that is not recognized by At-Tiny). 3. From Canoe, press key 'd' to start the test.	In Canoe trace, the message with id 0x60 will contain device information (same output as for DIAG1). With this test we prove that the state machine is still ok and commands are interpreted correctly by At-Tiny.

FIG. 3.2.2

TestCaseID	Test Case Name	Prio	Input	Expected result
DIAG11a	Load parameters from EEPROM if content is valid	1	1.Load file from sw.sys.at.tiny.1617.ref /eeprom/ EEPROMconfig_Xplained_Valid.eep or EEPROMconfig_PCB_Valid.eep (depends on test device: X-plain Board or PCB) 2. In ATTiny debugger, put a breakpoint in function IICB_u8ValidParametersFromEEPROM. 3. Reset and start the ATTiny debugger	Because EEPROM content is valid, function return eOk (check IICB_enTypeOfValidData variable).
DIAG11b	Load parameters from ROM if content is not valid	1	1.Load file from sw.sys.at.tiny.1617.ref /eeprom/ EEPROMconfig_Xplained_Invalid.eep or EEPROMconfig_PCB_Invalid.eep (depends on test device: X-plain Board or PCB) 2. In ATTiny debugger, put a breakpoint in function IICB_u8ValidParametersFromEEPROM. 3. Reset and start the ATTiny debugger	Because EEPROM content is not valid, function return enOk (check IICB_enTypeOfValidData variable).

FIG. 3.2.3

Deoarece am atașat capturi ale liniilor de I2C, avem nevoie să vedem care sunt adresele comenzilor trimise pe magistrală.

```
/* command to get device information */
IICB_nenSlaveReadDeviceInfo         =   0x93u,

/* command to calculate checksum */
IICB_nenSlaveCalculateChecksum      =   0x86u,

/* command to provide checksum */
IICB_nenSlaveProvideChecksum        =   0x89u,

/* command to set configuration byte */
IICB_nenSlaveSetConfigByte          =   0xF5u,

/* command to read configuration byte */
IICB_nenSlaveGetConfigByte          =   0x85u,

/* command to get button raw data */
IICB_nenSlaveGetRawButtonData       =   0x95u,
```

```
/* command to write 126 bytes to EEPROM */
IICB_nenSlaveSetBlockBytes          =   0x00u,

/* command to read 128 bytes from EEPROM */
IICB_nenSlaveGetBlockBytes          =   0x03u,

/* command to do a soft-ware reset of AT-Tiny */
IICB_nenSlaveSoftwareReset          =   0x85u,

/* command to jump to boot-loader */
IICB_nenSlaveStartFlash             =   0x87u,

/* command to read data from a specific address */
IICB_nenSlaveGetDataFromAdress      =   0xF3u,

/* command to write data to a specific address */
IICB_nenSlaveSetDataToAddress       =   0xF4u
```

FIG. 3.2.4

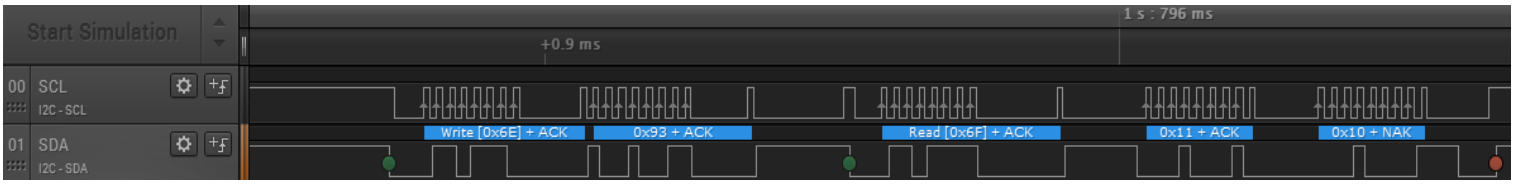


FIG. 3.2.5 **Comanda de citire a informațiilor dispozitivului (0x11 – SW version, 0x10 – BL version)**

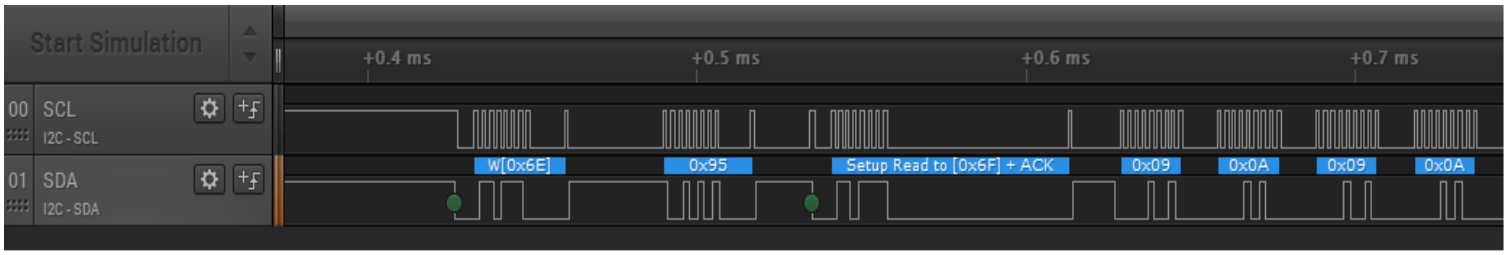


FIG. 3.2.6 **Comanda de citire a stării butonului când acesta este neapăsat.**

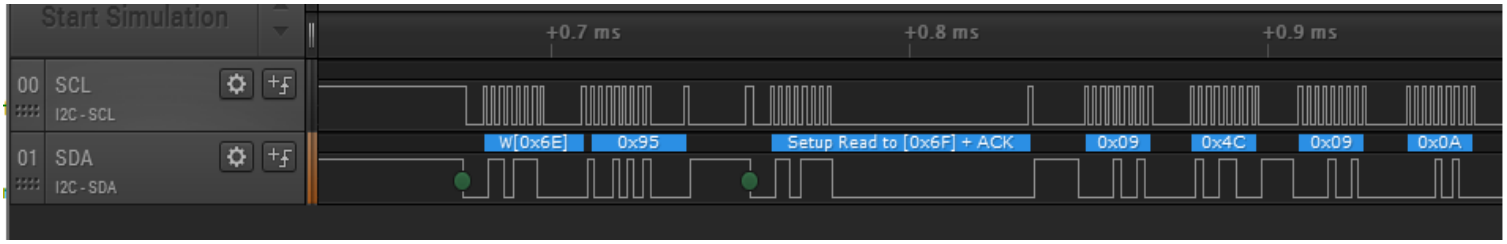


FIG. 3.2.7 **Comanda de citire a stării butonului când acesta este apăsat.**

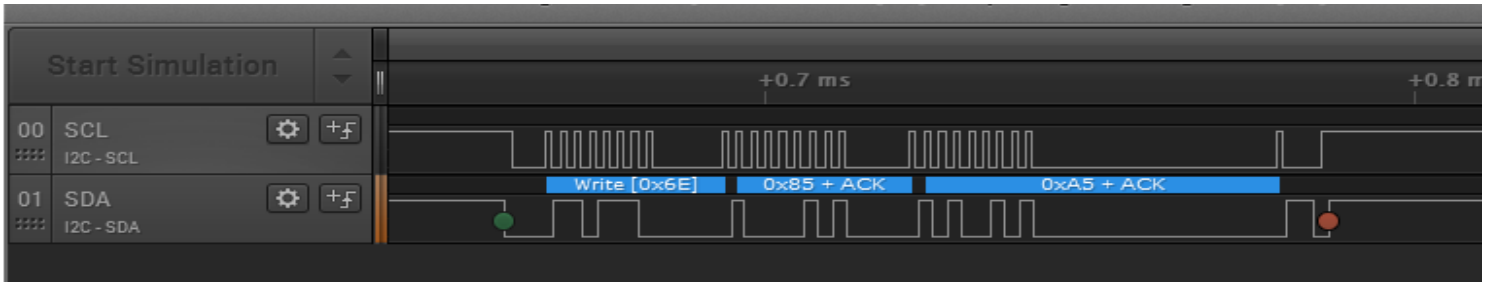


FIG. 3.2.8 **Comanda ce generează un software reset.**

FIG. 3.2.9 **Comanda furnizării sumei de control.**

4. Concluzii

Întregul proiect a fost dezvoltat în cadrul echipei mele, fiind primul proiect de o importanță semnificativă la care am participat de la început și până la sfârșit. Această experiență m-a ajutat să îmi formez o imagine mai bine conturată a ceea ce înseamnă meseria de programator. La început nu am realizat foarte bine rolul meu și spun asta deoarece nu vedeam ca pe o normalitate ca cineva cu ani de experiență în spate, aproape câți anii mei de viață, să îmi ceară mie părerea și să îmi dea voie să implementez un modul strict după abordarea mea. Dar mai apoi am înțeles că, eu fiind cel ce se află în spatele tastaturii, decizia îmi aparține, abordarea este a mea, trebuie să satisfac cerințele clientului, să îi furnizez funcționalitatea dorită, el nu știe sau nu vrea să afle ce se întâmplă în spate.

De altfel mi-am îmbunătățit și abilitatea de a căuta informația necesară într-un mod mai eficient. Având în vedere că această informație necesară implementării proiectului nu s-a aflat într-un singur document, am putut observa la alte proiecte ulterioare că ritmul stângaci întâlnit anterior a început să se schimbe spre mai bine, mai rapid, mai eficient.

Alt fapt pe care eu îl consider un avantaj este că am putut să îmi lărgesc orizontul cunoștințelor teoretice și practice dobândite în cadrul universității. Totodată mi-am dezvoltat capacitatea de a lucra în echipă, capacitatea de a direcționa obiectivele individuale spre obiective comune.

Având în vedere că metodologia folosită la mine în echipă se bazează pe SCRUM, am învățat să îmi estimez timpul de lucru foarte aproape de realitate și totodată am învățat să prioritizez sarcinile atribuite.

Cu siguranță nu toate lucrurile au mers perfect. Pe parcursul dezvoltării acestui modul am întâlnit următoarele probleme:

- Placa de dezvoltare cu o memorie mai mică decât fișierele generate necesare pentru flash.
- Lipsa unei plăci de dezvoltare de rezervă (una din două plăci s-a ars, timp de așteptare mare până a fost reparată).
- Hardware-ul de proiect a necesitat lipituri extra și înlocuirea unor rezistoare.
- Un singur debugger Atmel-ICE.
- Neînțelegere între mine, responsabilul de proiect din Germania și echipa de bootloader în privința unor comenzi de diagnoză (unele dintre adresele comenzilor au fost alese de mine).
- Implementare eronată a automatului ce interpreta comenzile de I2C.
- Lăsând active unele întreruperi în modulul nostru, afecta unele funcționalități ale bootloaderului

În ciuda tuturor problemelor apărute pe parcursul dezvoltării software-ului am reușit să livrăm proiectul la timp și cu toate funcționalitățile dorite de client.

5. Bibliografie

1. Introducere
   [1] – Nicolae Postavaru, “Curs de marketing”, Editura Bucuresti, 1997.
   [2] – Adriana Badescu, “Management”, Editura Bucuresti: Nemira, 1998.
   [3] – TheCuriousEngineer Channel, https://www.youtube.com/watch?v=FyCE2h_yjxI
   [4] – ***. https://www.statista.com/statistics/200002/international-car-sales-since-1990
   [5] – ***. https://www.statista.com/topics/1487/automotive-industry

2. Proiectare și implementare
   [1] – Atmel-AVR-ATtiny1614-1616-1617-Datasheet_Complete.pdf
   [2] – Atmel-42805-QTouch-Modular-Library-Peripheral-Touch-Controller_User-Guide.pdf
   [3] – Atmel-42745-ATtiny817-Xplained-Pro_User Guide.pdf
   [4] – PS_ABTW_TouchButtonSubSystem_V9.pdf – document intern Continental Automotive S.R.L.
   [5] – ATTiny1616_v0.5_Portlist_09012018_HPI_C.xlsx – document intern Continental Automotive S.R.L.

3. Testare
   [1] – sysfware1ts1.docm – document intern Continental Automotive S.R.L.
   [2] – EepromDataHandlingTiny.xlsx – document intern Continental Automotive S.R.L.

Anexă - Lista figurilor