Politehnica University Timișoara

Faculty of Automation and Computers

**Computers and Information Technology**

**Modul diagnoză pentru panoul de control al sistemului multimedia in industria automotive**

*Candidate:*

**Name SURNAME**

*Supervisors:*

Prof. dr. eng. **Name SURNAME**

Conf. dr. eng. **Name SURNAME**

Șl. dr. eng. **Name SURNAME**

As. dr. eng. **Name SURNAME**

Timișoara

2017

1. Introducere

Încă din momentul de când oamenii au început să comunice între ei, prezența comerțului s-a făcut necesară. Dacă la început, primii oameni 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 displayului touch, cererea pentru acesta fiind din ce în ce mai mare, se poate observa această expansiune 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 touch screen, 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.

Interiorul unei mașini din anii 1990:





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

Interiorul unei mașini din anul 2017:



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.

**Diferența dintre generații și tehnologii**

**Butonul fizic**

Butonul este un tip de comutator utilizat pentru controlul unor aparate sau numai a unor funcții ale acestora. Este un dispozitiv electronic, de obicei funcționează ca un întrerupător electric, având rolul de a închide și deschide circuitul. Un circuit străbătut de un curent electric este un circuit închis, iar cel prin care nu trece curent electric este un circuit deschis.

**Touch screen**

Există multe tehnologii ale touch screenu-ului, dar cele mai des întâlnite sunt cele rezistive și cele capacitive.

Ecranele rezistive rezistă literalmente atingerii și dacă este apăsat destul de puternic se poate simți ecranul ușor, acesta este ceea ce face ca ecrane rezistive să funcționeze. 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, 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.

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. 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 ). 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.

Departamentul din care fac parte se numește Instrumentation & Driver și rolul lui principal este de a dezvolta soluții si servicii de conectare, control si funcționare a vehiculelor.

Un obiectiv important este dezvoltarea de soluții inteligente care să facă conducerea mai sigură și mai confortabilă. Prin urmare, conectăm vehiculele cu șoferii și pasagerii, cu alte vehicule și mediul înconjurător.

Business Unit-ului HMI Instrumentation & Driver lucrează la posibilitățile de prelucrare și prezentare optimă a informațiilor. Accentul este pus pe prioritizarea informațiilor afișate pe diferite ecrane și indicatoare pentru o bună și plăcută desfășurare a condusului.

Produsele noastre : cluster, head-up display, secondary display, interior camera.

Proiectul ce urmează a fi descris în această lucrare este un proiect al grupului VAG în care se dorește înlocuirea microcontrolerului Cypress PSoC4 cu microcontrolerul Atmel AT-Tin T1616.Privind driver-ul de Host controller, acesta nu trebuie să sufere nici o modificare.

**Prezentare generală**

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 obligatoriu (albastru), care poate să trezească ABT și un buton capacitiv 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

Având în vedere că ambele butoane vor fi comandate cu microcontrolerul butonului tactil, în ceea ce urmează ne vom axa pe acest uC.

Privire de ansamblu al funcționalităților ce se doresc a fi implementate.



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 I2C trebuie să fie implementate în  
  codul aplicației: **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).

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 microporcesoarele 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.

1. **Proiectare si implementare**

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.

* + 1. **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  
• 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  
 - Un USART cu generator de rata baud fractionata, autobaud si detectie de start-cadru  
 - 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, glisoare și roți  
 • Trezire la atingere  
 • Ecran condus pentru o performanță îmbunătățită a umidității și a zgomotului  
 • Până la 14 capacități de auto-capacitate și până la 49 de canale de capacitate reciprocă  
 - 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  




* + 1. **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®  
• Un LED galben pentru utilizator  
• cristal de 32.768kHz  
• Două plăcuțe de extensie Xplained Pro  
• Debugger încorporat  
 - Identificare automată pentru identificarea plăcii în Atmel Studio  
 - Un LED galben  
 - Un LED de putere de bord  
 - 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



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

* + 1. **Project hardware**















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

**TWI**

**Prezentare generală**  
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 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 repaus de pe modul de potrivire a adresei TWI. Este posibil să dezactivați potrivirea adreselor pentru a permite ca acestea să fie tratate în software.  
Perifericul TWI va detecta condițiile START și STOP, coliziunile cu magisrala și erorile din magistrala. 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**  
TWI oferă un bus simplu, bidirecțional, cu două fire, compus dintr-o linie de clock serial  
(SCL) și o linie de date seriale (SDA). Cele două linii sunt linii cu colector deschis (prin cablu) și rezistoare pull-up (Rp) fiind singurele componente externe necesare pentru a conduce bus-ul. Rezistoarele pull-up oferă un nivel ridicat pe linii atunci când nici unul dintre dispozitivele conectate nu conduc magistrala.  
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.



**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 proprietății bus-ului în rândul masterilor, deoarece numai un singur dispozitiv principal 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.



The master provides the clock signal for the transaction, but a device connected to the bus is allowed to stretch the low-level period of the clock to decrease the clock speed.

**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). TWI.MADDR registrul are, de asemenea, un bit R / W care indică dacă Masterul transmite sau primește. Maestrul 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.



În timpul unei singure tranzacții pot fi emise condiții START multiple. O condiție START care nu urmărește 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.



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 ăn 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 26-6 ilustrează tranzacția de scriere principală. Masterul inițiază tranzacția emise o condiție START (S) urmată de un pachet de adresă cu bitul de direcție setat la zero (ADDRESS + W).



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 26-7 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.



Presupunând că slave-ul recunoaște adresa, masterul poate începe să primească date de la slave. Nu există nicio 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 26-8 ilustrează o tranzacție combinată. O tranzacție combinată constă în mai multe tranzacții de citire și scriere separate prin condiții START repetate (Sr).



**Arbitraj**Un master poate porni o tranzacție de bus numai dacă a detectat că magistrala este inactivă. Deoarece magistrala TWI este o magistrală multimaster, este posibil ca două dispozitive să poată iniția o tranzacție în același timp. Acest lucru are ca rezultat mai mulți masteri care accesează busul 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 când busul se oprește (adică, așteaptă o condiție STOP) înainte de a încerca să recâștige arbitrarea busului. Dispozitivele slave nu sunt implicate în procedura de arbitraj.



Figura 26-10 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 26-11 arată un exemplu în care doi masteri concurează pentru controlul clock-ului. Linia SCL este rezultatul cablului-AND al celor două ieșiri de clock ale masterilor.

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 mic. Durata de sincronizare a perioadei de clock mic 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 l-au eliberat. 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 steag 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. Figura 26-16. arată funcționarea slave TWI. Simbolurile formelor de diamant (SW) indică unde este necesară interacțiunea software-ului.  
Numărul de întreruperi generate este menținut la minimum prin manipularea automată a majorității condițiilor. Comanda rapidă poate fi activată pentru a declanșa automat operațiile și pentru a reduce complexitatea software-ului. Modul de recunoaștere a adresei poate fi activat pentru a permite slavei 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 slaveul va activa o adresă corectă și va stoca adresa în registrul de date. Dacă adresa primită nu este o potrivire, slaveul 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 adresă generală de apel va seta de asemenea pavilionul de întrerupere.  
O condiție START imediat urmată de o condiție STOP este o operație ilegală și este setat stegul de eroare de magistrală.  
Steagul 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 de biți și de bus a direcției 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 steagul direcției R / W, aceasta indică o operație de citire principală. Linia SCL este forțată să scadă de către slave, întinzând ceasul 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, slaveul va aștepta o nouă condiție START și o potrivire a adresei.

**Cazul S2**: Acceptarea pachetului de adrese - Ștergerea direcției Șterse Dacă indicatorul de direcție R / W este șters, aceasta indică o operație de scriere master. Linia SCL este forțată să scadă, întinzând ceasul busului. Dacă ACK-ul este trimis de slave, slaveul va aștepta recepția datelor. Datele, START repetate sau STOP pot fi primite după aceasta. Dacă este trimis NACK, slaveul va aștepta o nouă stare START și o potrivire a adresei.

**Cazul S3**: Coliziune  
Dacă robotul nu este capabil să trimită un nivel înalt sau NACK, steagul de coliziune este setat și va dezactiva datele și ieșirea de confirmare din logica slave. Ceasul este eliberat. Este acceptată o START sau o condiție START repetată.

**Cazul S4**: Starea STOP primită  
Când se primește condiția STOP, se va seta semnalizatorul adresa / stop-ul slave, 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 un bit de direcție R / W. După ce a confirmat acest lucru, slaveul trebuie să fie pregătit să primească date. Când se recepționează un pachet de date, parametrul de întrerupere a datelor este setat, iar slave-ul trebuie să indice ACK sau NACK. După indicarea unui NACK, slaveul 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 adrese cu setul de biți direcți R / W. Apoi, poate începe trimiterea datelor prin scrierea în registrul de date slave. Când este finalizată transmisia de pachete de date, este setat stegul 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.

**Intreruperi**



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.



**Obiectul de testare**

Sistemul testat este sistemul SW pentru controlerul Atmel AT-Tiny T1616, cu privire la mai multe servicii:  
 - alive supervision of the HC

- alive signal of the BTN controller

- synchronized sensing, output of sync signal

- reset handling

- waking of the system on button touch.

- I2C communication

Testarea contine si celelalte functionalitati

**Abordare de testare**Toate testele caracteristicilor de bază și configurația pachetului SYSFWARE vor fi efectuate prin inițierea secvențelor de testare corecte pe baza cerințelor clientului.

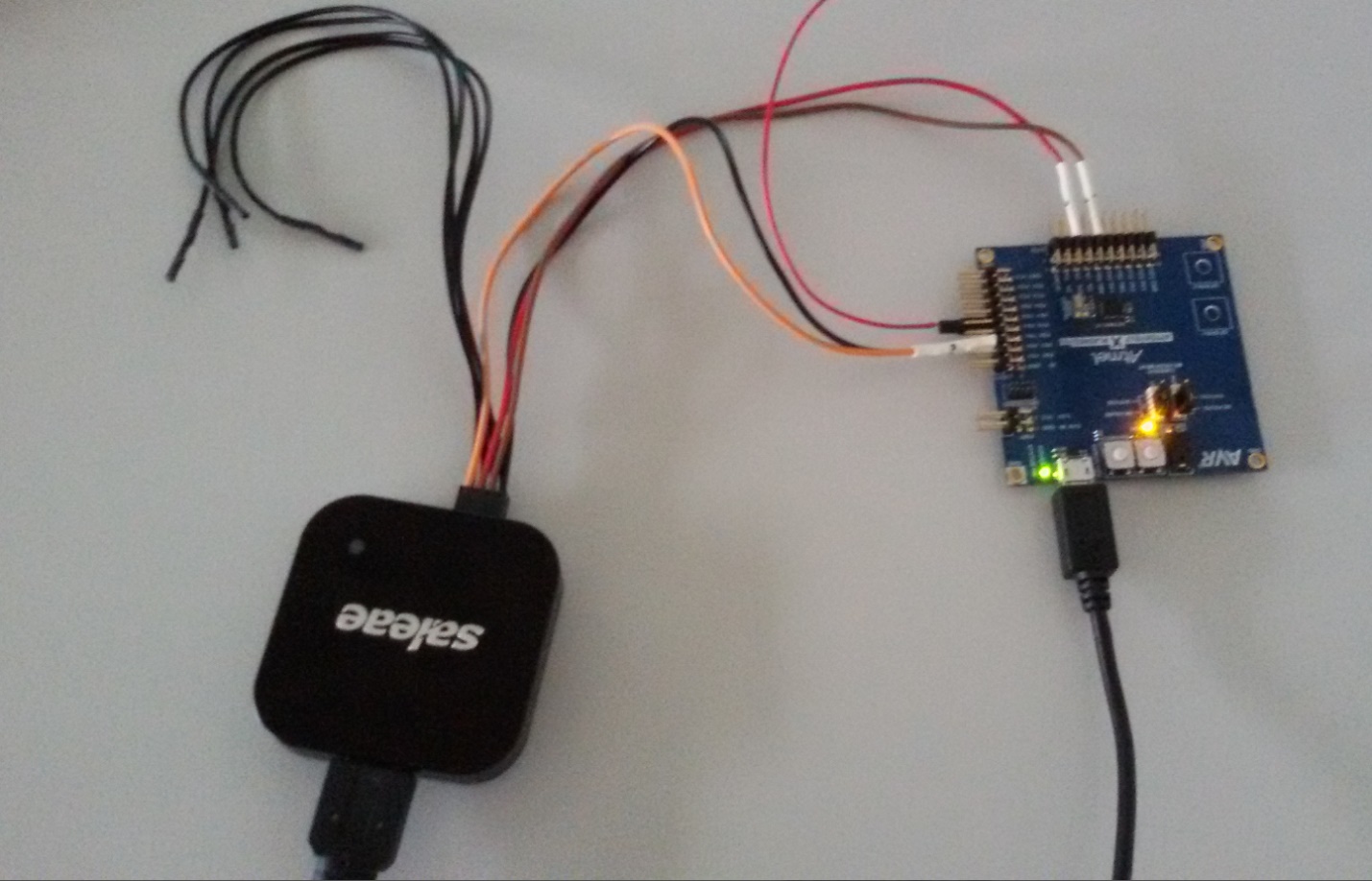
**Scopul testelor**  
Testarea funcționalității modulului SYSFWARE pentru a vedea dacă se comportă așa cum ne dorim și pentru detectarea eventualelor bug-uri introduse nevoit în timpul implementării.

**Strategia de testare**Există două tipuri de teste:  
- teste manuale: în cazul în care este necesar să verificați dacă led-ul este pornit sau oprit, atingeți sau eliberați butonul tactil, verificați semnalele generate cu instrumentul Saleae (sau osciloscop)  
- teste automate: trimiteți comenzi prin intermediul dispozitivului I2C la dispozitivul AT-Tiny și vedeți modul în care sistemul reacționează verificând semnalele și flag-urile corespunătoare.

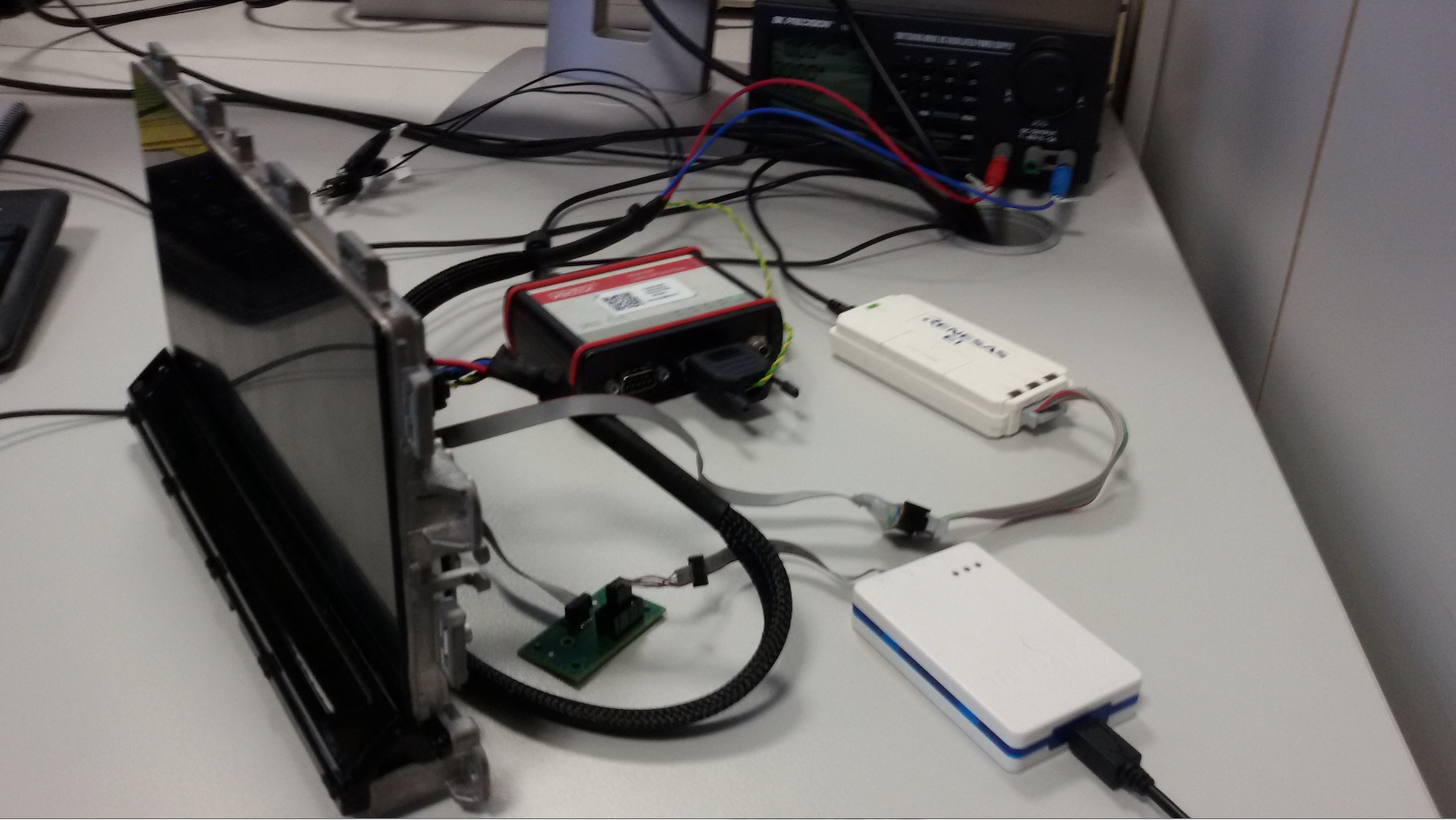
**Mediul de testare**

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).



- proiectul VW\_MIB\_ABT\_W (versiunea PCB hw B0301 și sistemul sw cu BLOGIC\_nHwUsed macro care conține valoarea BLOGIC\_nPCBSample\_H20).



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

**TESTE**

Exista 3 categorii mari de teste:

1. **Testarea funcționalităților de touch** (touch buttons, transitions between operational modes)
2. **Testarea stărilor sistemului** (alive indication, watchdog, logic reset, external wakeup)
3. **Testarea modului diagnoză** (read memory by address, write memory by address, read 128 bytes from EEPROM, write 126 bytes to EEPROM, read CS device information, read raw button data, reset AT-Tiny device, start flash, prepare checksum, provide checksum)
4. **Testarea funcționalităților de touch**

Pentru a testa aceste funcționalități am folosit placa de dezvoltare Attiny817 Xplain Board.

Exemplu de câteva cerințe ale proiectului ce au trebuit a fi respectate:

* În cazul în care nu este disponibil un semnal de sincronizare pe SYNC\_IN, atingeți butonul  
  controlerul trebuie să treacă de la synchronized scan la free runing scan
* Sistemul trece in free runing mode dacă nu este detectată o tranziție a semnalului SYNC\_IN pentru un t > [C\_SYNC\_TIMEOUT]
* Semnalul de ieșire SYNC\_OUT va fi setat corespunzător [C\_GEN\_SYNC DELAY]  
  milisecunde după frontul crescator al semnalului de intrare SYNC\_IN.
* Tranzitia T1

Modul normal de funcționare va fi activat când se va verifica atingerea degetului pe butonul de trezire  
SAU  
semnalul de intrare 3V3\_MON are un nivel HIGH pentru t> = [C\_MAINPOWER\_ON\_TIME].

Următoarele porturi trebuie configurate în consecință.

[ALIVE\_OUT]: HIGH level

[RESET\_OUT\_EXT]: LOW level

[RESET]: HIGH level

[SYNC\_OUT]: Depends on SYNC\_IN

[SHIELD]: Depends on scan status

[BTNx\_OUT]: Depends on touch status

* Tranziția T2

3V3\_MON este LOW pentru t> = [C\_MAINPOWER\_OFF\_TIME].  
Porturile de ieșire trebuie să intre într-o stare adecvată pentru a consuma minimum actual.

**Exemplu de teste**

| **TestCaseID** | **Test Case Name** | **Prio** | **Input** | **Expected result** |
| --- | --- | --- | --- | --- |
| TOUCH1 | SYNC\_OUT : Low lvl | 1 | [TBSS\_REQ\_045; TBSS\_REQ\_047]  1. Set BLOGIC\_nHwUsed from blogic\_ce.h to BLOGIC\_nAtTinyExpBoard and compile the system.  2. Connect tested pins to Saleae (3V3\_MON – PA4, SYNC\_IN – PA5, SYNC\_OUT – PB5).  3. Set 3V3\_MON to HIGH(use a power supply and set the voltage to 3.3 V).  4. Connect the eXplained emulator board to an USB cable  (any 5.0 V USB PC port).  5. Start the system and check the expected results. | Check the output signal for SYNC\_OUT: should be constant on low level.  The module is in free running mode.  BLOGIC\_enModState = BLOGIC\_nenFreeRunningMode; |
| TOUCH2 | SYNC\_OUT : alternates from Low to High lvl | 1 | [TBSS\_REQ\_080]  1. Repeat first 4 steps described above. 2. Using the signal generator, apply an input signal of 100 ms (80 ms on Low and 20 ms on High) on PA5 pin from EXT1 connector on the PCB.  3. Start the system and check the expected results. | SYNC\_OUT depends on SYNC\_IN and both signals looks similar with below remark :  1) SYNC\_IN get on High level and SYNC\_OUT will get on HIGH after 2 ms;  2) SYNC\_IN get on Low and in the next clock cycle, SYNC\_IN get on Low as well (after around 1 ms). |
| TOUCH3 | Touch button status - touched | 1 | [TBSS\_REQ\_030]  The output signal BTNx\_OUT indicates the status of the assigned touch button.  If the touch button is evaluated as touched, BTNx\_OUT is set to HIGH level.  Touch the button and evaluate the result. | BTNx\_OUT is set to HIGH level. |
| TOUCH4 | Touch button status - released | 1 | [TBSS\_REQ\_030]  The output signal BTNx\_OUT indicates the status of the assigned touch button.  If the touch button is evaluated as released (not touched), BTNx\_OUT is to be set to LOW level.  Touch and then release the button and evaluate the result. | BTNx\_OUT is to be set to LOW level. |
| TOUCH5b | Transition from low power mode to normal mode (based on host 3V3 pin) | 1 | [TBSS\_REQ\_036]  Make sure that reset logic functionality is active.  The input signal 3V3\_MON has HIGH level for t >=  [C\_MAINPOWER\_ON\_TIME]. Apply 3V3 input signal to 3V3\_MON pin. | The following ports shall be configured accordingly.  [RESET\_OUT\_EXT]: LOW level  [RESET]: HIGH level  [SYNC\_OUT]: Depends on SYNC\_IN  [SHIELD]: Depends on scan status  [BTNx\_OUT]: Depends on touch status |
| TOUCH6 | Transition from normal operation mode to low power operation mode | 1 | [TBSS\_REQ\_037]  The Low Power Operation Mode shall be activated if the input signal 3V3\_MON is LOW for t >= [C\_MAINPOWER\_OFF\_TIME].  1. In ATTiny debugger, put a breakpoint in function BLOGIC\_vEnterLowPowerMode. | The output ports shall enter a proper state in order to consume minimum current (less than 30 uA – [TBSS\_REQ\_051]).  The breakpoint is reached. |

1. **Testarea stărilor sistemului**

Pentru a testa aceste funcționalități am folosit atat placa de dezvoltare Attiny817 Xplain Board cat si hardware-ul de proiect.

Exemplu de câteva cerințe ale proiectului ce au trebuit a fi respectate:

* Alive indication va fi dezactivată automat la intrarea în modul de funcționare cu consum redus de energie.
* Această funcție va fi dezactivată automat la intrarea în bootloader.
* Dacă alive indication este inactiv, semnalul ALIVE\_OUT este menținut la nivelul LOW.
* Microcontrolerul trebuie să informeze controlorul gazdă în mod frecvent despre nivelul său de viață, prin comutarea unui port
* Alive indication trebuie activată automat la intrarea în modul normal de funcționare.
* Alive indication trebuie să fie activat numai dacă este asigurată funcționalitatea obișnuită (de exemplu, supravegherea supravegherii de supraveghere).
* Funcția watchdog este controlată de semnalul de intrare WDEN
* Pentru a activa funcționalitatea, semnalul de intrare WDEN trebuie să fie setat la nivelul HIGH constant.
* Pentru a dezactiva funcționalitatea, semnalul de intrare WDEN trebuie setat la un nivel LOW constant.
* Timpul de supraveghere începe cu activarea funcționalității. Dezactivarea înseamnă că temporizatorul va fi resetat.

1. **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

1. **Concluzii** - o elegantă combinație narativă dintre ideile propuse în introducere, coroborate cu rezultatele obținute. Opțional, se vor prezenta pași de dezvoltare pentru viitor.

DESCRIERE IMPLEMENTARE MOD DIAGNOZĂ

* 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



**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.

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.



**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.



**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 și 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 și 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 pozișie 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**

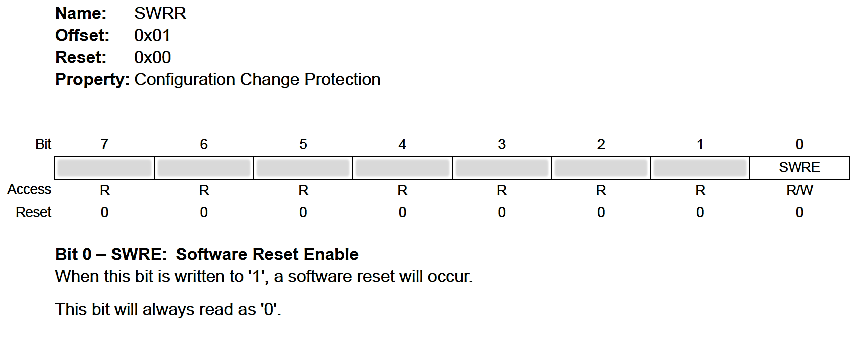
* Este operația inversă scrierii, se folosește o interfață Atmel, dar pentru citire având trei parametrii: adresa de la care se citește, bufferul în care se citește și numărul de bytes doriți să se citească

**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



**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