2016•2017
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef
Efficiënte softwareontwikkeling voor ARM-processoren

Promotor :
ing. Luc COENEGRACHT

Promotor :
WILLIAM KNOORS

Robin Moons
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2016•2017
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Efficiënte softwareontwikkeling voor ARM-processoren

Promotor :
ing. Luc COENEGRACHT

Promotor :
WILLIAM KNOORS

Robin Moons
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Woord vooraf

Als masterproef voor mijn studie master in de Industriële Ingenieurswetenschappen Elektronica-ICT aan de UHasselt i.s.m. KU Leuven, heb ik een opdracht uitgevoerd bij Bits & Bytes nv te Hasselt. Dit was een zeer interessante opdracht die volledig binnen mijn interessegebied valt. De combinatie hardware- softwareontwikkeling voor de domotica-markt vind ik zeer boeiend.

Ik werd tijdens mijn masterproef begeleid door dhr. William Knoors, zaakvoerder van Bits & Bytes nv, en ing. Luc Coenegracht, docent van de KU Leuven. Vooraf wisten we dat het geen gemakkelijke opdracht ging worden maar dat motiveerde mij des te meer om het einddoel te behalen. Vooraleer ik verder ga, wil ik een aantal mensen bedanken.

Eerst en vooral wil ik mijn externe promotor, dhr. William Knoors bedanken. Hij heeft me doorheen het hele jaar bijgestaan en niet enkel technische maar ook economische aspecten van productontwikkeling aangeleerd. Ook wil ik mijn interne promotor ing. Luc Coenegracht bedanken. Hij heeft mijn masterproef van dichtbij opgevolgd en voor vragen kon ik steeds bij hem terecht. Verder bedank ik de collega's bij Bits & Bytes, mijn klasgenoten en de proffen en docenten die me geholpen hebben met de uitvoering van mijn masterproef. Ten slotte, maar niet minst belangrijke, gaat mijn dank uit naar mijn familie en vrienden, die mij altijd gesteund hebben en zonder wie ik nooit geraakt was waar ik nu ben.

Robin Moons, juni 2017.

Inhoudsopgave

Lijst met tabellen

Lijst van figuren

Verklarende woordenlijst

Hieronder volgt een alfabetische lijst met enkele termen en afkortingen die in deze masterthesis voorkomen samen met hun betekenis.

ARM	Acorn RISC Machine, genoemd naar de voormalige firma Acorn
B&B Home	Het Bits & Bytes Home domoticasysteem
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPIO	General Purpose Input Output, afkorting gebruikt als algemene omschrijving voor processor in- uitgangen
I2C	Inter IC Communicatie, communicatieprotocol tussen IC's
IC	Integrated Circuit
IDE	Integrated Development Environment, software ontwikkelomgeving
IO	Input Output, afkorting gebruikt als algemene omschrijving voor processor in-uitgangen
IR	infrarood
ISR	Interrupt Service Routine
J-Link	Type debugprobe van fabrikant Segger
JTAG	Joint Test Action Group
LPC1768	ARM Cortex-M3 microcontroller van NXP
MAC	Media Acces Control, laag in het OSI model.
mbedOS	Mbed Operating System, besturingssysteem voor mbed platformen
NXP	Electronica fabrikant
OLED	Organic LED, organische led
OS	Operating System
OSI model	Open System Interconnection model, een conceptueel model dat een netwerkstructuur beschrijft
PHY	Fysical layer, netwerkinterface-ic
RISC	Reduced Instruction Set Computer
RS232	Standaard voor seriële communicatie
RTOS	Real Time Operating System
SMIC	Smart Media IP Controller, product van Bits & Bytes nv
SWD	Serial Wire Debug, communicatieprotocol
SWIDIS	Display Switch, product van Bits & Bytes nv
TCP/IP	Transmission Control Protocol / Internet Protocol
UTP	Unshielded Twisted Pair, soort netwerkkabel

Abstract

Bij Bits & Bytes nv te Hasselt wordt het B&B Home domoticasysteem ontwikkeld. Dit unieke systeem zorgt voor een complete domotica-ervaring door integratie van onder andere verlichting, klimaatsturing, beveiligings- en multimedia-apparatuur. Voor de ontwikkeling van een reeks nieuwe producten is Bits & Bytes op zoek naar een geschikte, complete ontwikkelomgeving voor softwareontwikkeling toegepast op ARM-processoren. Momenteel wordt er gebruik gemaakt van een gratis, online tool waarbij er geen uitgebreide debugmogelijkheden zijn.

Deze masterproef onderzoekt verschillende IDE's en debugtools die geschikt zijn voor Bits & Bytes. Ook wordt er onderzoek gedaan naar een goed versiebeheersysteem en geschikte documentatietools. De meest geschikte tools worden geïnstalleerd en gebruikt voor de softwareontwikkeling van de B&B Home smart media IP controller. Dit toestel, op basis van een ARM-processor, zorgt voor de complete aansturing van een homecinema installatie. Via een ethernetnetwerk communiceert dit toestel met de B&B Home domoticaserver, en zorgt voor de aansturing van media-apparatuur via infrarood, RS232 en digitale IO.

Door de nieuwe tools en IDE is er nu een snellere en efficiëntere ontwikkelprocedure. Deze zullen ook voor toekomstige projecten met ARM microcontrollers toegepast worden. Door implementatie en gebruik van de onderzochte tools is de software voor de smart media IP controller nu klaar en na een laatste testfase zal de productie en implementatie in het B&B Home systeem gestart worden.

Abstract in English

At Bits & Bytes nv in Hasselt the B&B Home domoticasystem is developed. This unique system takes care of a complete domotica experience by integrating among others lighting, climate control and security and multimedia devices. For the development of a new series of products, Bits & Bytes is looking for an appropriate, complete development environment for software development on ARM processors. Currently a free online tool is used without suitable debug possibilities

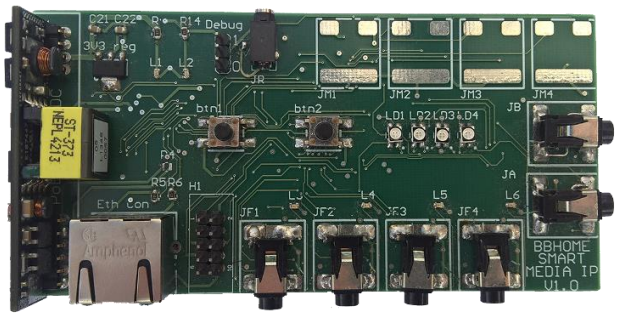
This Master's thesis investigates the different IDE's and debugtools that are suited for Bits & Bytes. In addition research has been conducted to find a good version control system and suitable documentation tools. The most suited tools will be installed and used for the software development of the B&B Home Smart Media IP Controller. This device based on an ARM-processor takes care of the complete control of a home cinema installation. The device communicates through an Ethernet network with the BBHome domotica server and takes care of the control of media devices via infrared, RS232 and digital IO's.

Because of the new tools and IDE there is now a quicker and more efficient development procedure. This will be used for future projects with ARM microcontrollers. By implementation and use of the researched tools, the software for the smart media IP controller is now ready and after a last test phase the production and implementation in the B&B Home system will be started.

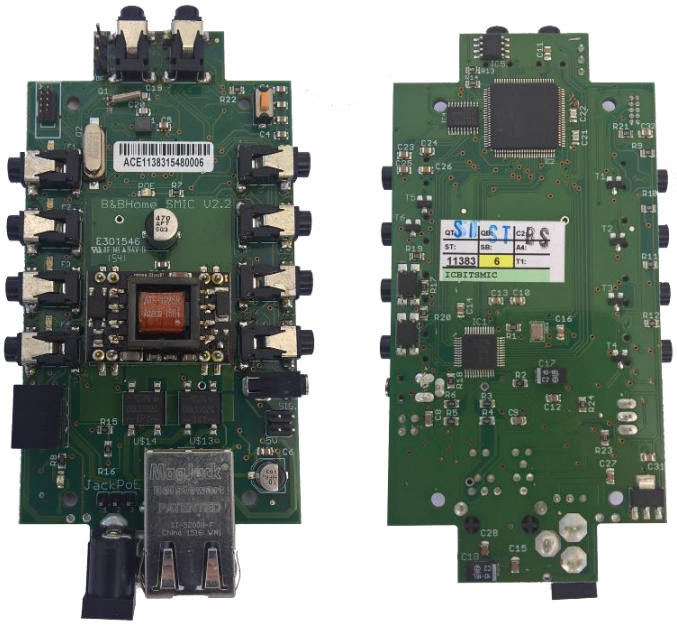
1 Inleiding

1.1 Bits & Bytes nv

Bits & Bytes is een bedrijf opgericht in 1984 en heeft sindsdien een uitstekende reputatie opgebouwd op het gebied van domotica en IT. Bij Bits & Bytes wordt het B&B Home domotica systeem ontwikkeld. Dit unieke systeem houdt rekening met comfort, veiligheid, en de energiekosten [1]. Gedurende een projectstage tijdens mijn opleiding professionele bachelor, aan de UC Leuven-Limburg (2014-2015) heb ik kennis gemaakt met Bits & Bytes. Ik heb er een prototype van de B&B Home Smart Media IP Controller (SMIC) ontwikkeld. Dit is een toestel om allerhande multimedia apparaten aan te sturen zoals: TV, setup-box, stereoketen en nog veel meer. Via dit toestel is het mogelijk om al uw multimedia te koppelen aan het B&B Home systeem. Zo kunnen verschillende acties geautomatiseerd worden voor een verhoging van het comfort van de gebruiker. Ieder apparaat heeft niet langer zijn eigen afstandsbediening nodig, maar alles wordt centraal gestuurd via het B&B Home systeem [2]. Ondertussen is de hardware voor de SMIC volledig klaar en moet enkel de software nog verder ontwikkeld worden.



Figuur 1: SMIC prototype

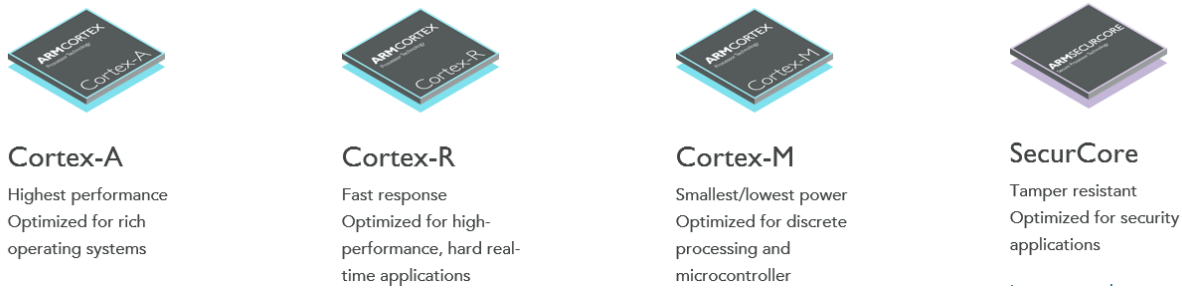


Figuur 2: SMIC V2.2

1.2 ARM-processoren

ARM-processoren zijn niet meer weg te denken uit onze samenleving, ze worden gebruikt in smartphones, tablets, TV's, auto's, wasmachines, sensoren, en nog veel meer. Meer dan 50% van de smartphones ter wereld, zijn uitgerust met ARM-processoren. Ze worden geproduceerd door verschillende fabrikanten en zijn steeds gebaseerd op de ARM-architectuur. ARM staat voor Acorn RISC Machine. Acorn Computers Ltd was een Brits bedrijf dat eind jaren '80 processoren en computers ontwierp. Toen stroomverbruik een kritische rol ging spelen bij de ontwikkeling van processoren voor draagbare apparaten was Acorn een stap voor op zijn concurrent Intel. Ook de samenwerking met Apple zorgde voor een boost in de ontwikkeling van de ARM-technologie. Tegenwoordig produceert ARM geen processoren meer, maar verkoopt het enkel de licenties van hun technologieën. Deze licenties worden beheerd door ARM Holdings. De productie gebeurt door bekende producenten zoals Atmel, STMicroelectronics, Intel, nVidea,… . Zij gebruiken de ARM-architectuur voor het bouwen van hun product-specifieke processoren.

De ARM-familie bestaat uit de Cortex-A, Cortex-R, Cortex-M en de SecurCore reeks. De Cortex-A serie is ontwikkeld voor hoog performante systemen en dit soort processoren worden gebruikt in producten zoals de Apple iPhone en de Microsoft Surface RT. De Cortex-R serie is dan weer ontwikkeld voor tijd-kritische systemen, bijvoorbeeld voor de medische- en automobielsector. De Cortex-M processor serie bevat een breed gamma aan energy efficiënte, low cost processoren voor embedded toepassingen. Zij worden gebruikt in Internet of Things apparaten, en allerhande embedded systemen zoals slimme sensoren. Het is deze reeks processoren die in deze thesis besproken worden.
Tenslotte is er ook nog de SecurCode reeks. Deze processoren zijn ontwikkeld voor veiligheidstoepassingen zoals elektronische ticketsystemen en smartcards [3]–[5].



Figuur 3: Vergelijk ARM-processoren[4]

1.3 ARM-processoren toegepast bij Bits & Bytes nv

Bij Bits & Bytes maakt men ondertussen ook gebruik van enkele ARM processoren. Bij de ontwikkeling van de SMIC is een LPC1768 microcontroller gebruikt. Dit is een microcontroller van NXP gebaseerd op de ARM Cortex-M3 architectuur. Ook voor de ontwikkeling van de B&BHome SWIDIS is een ARM microcontroller gebruikt, de LPC4088 gebruikt. Dit is een krachtigere (Cortex-M4) processor, ook van NXP.

Voor de ontwikkeling van toekomstige producten wil Bits & Bytes gebruik maken van de nieuwste processor technologieën. Maar het is de bedoeling dat overal dezelfde ontwikkelomgeving gebruikt kan worden. Het is dus geen vereiste dat NXP processoren gebruikt worden, maar de te gebruiken IDE moet deze zeker ondersteunen omdat ze al in twee voorgaande producten verwerkt zijn. We specifiëren het onderzoek dus naar de Cortex-M reeks, maar houden rekening met alle mogelijke fabrikanten van Cortex-M processoren.

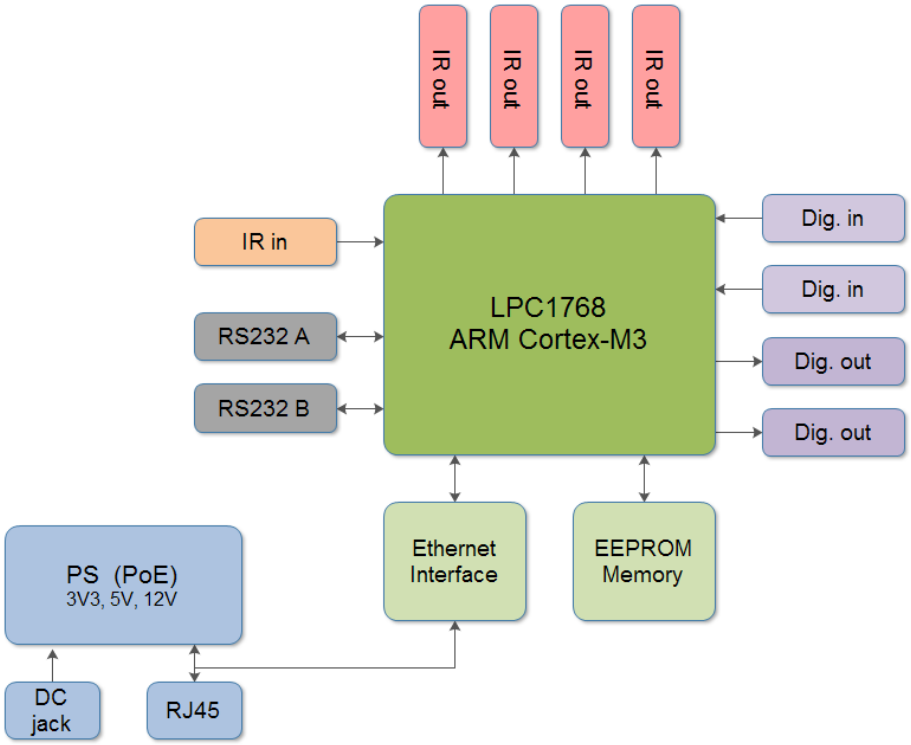
1.3.1 B&B Home SWIDIS

De B&BHome SWIDIS is een multifunctionele schakelaar voorzien van een OLED display. Deze schakelaar kan aangesloten worden met enkel één UTP-kabel. Door het gebruik van een Power over Ethernet switch wordt zowel voeding als data via de UTP-kabel aangeleverd. Met deze hoogtechnologische schakelaar is het mogelijk om een volledig B&B Home domoticasysteem te bedienen.

Figuur 4: B&BHome SWIDIS

1.3.2 B&B Home SMIC

Zoals aangehaald in de inleiding is de SMIC (Smart Media IP Controller) een toestel voor het aansturen van multimedia apparatuur. Verder in deze scriptie zal er beschreven worden hoe de software voor dit toestel ontwikkeld is. Ter verduidelijking wordt eerst een kort overzicht gegeven van de hardware. Onderstaand is het blokschema van de meest recente versie te zien.



Figuur 5: B&BHome SMIC blokschema [2]

De module bestaat uit verschillende ingangen en uitgangen die geprogrammeerd zullen worden. Zo zijn er twee seriële poorten (RS232 A en B), vier infrarood uitgangen (IR out), één infrarood ingang, twee digitale ingangen (Dig. in), en twee digitale uitgangen (Dig. out). Ook is er nog een EEPROM geheugen dat via het I2C protocol communiceert met de processor. Deze zal dienen voor het opslaan van gebruikersinstellingen. Tenslotte is er de ethernet interface die TCP/IP communicatie met de domoticaserver mogelijk maakt. Een uitgebreide beschrijving van de programmatie van deze onderdelen volgt in het tweede deel van deze thesis.

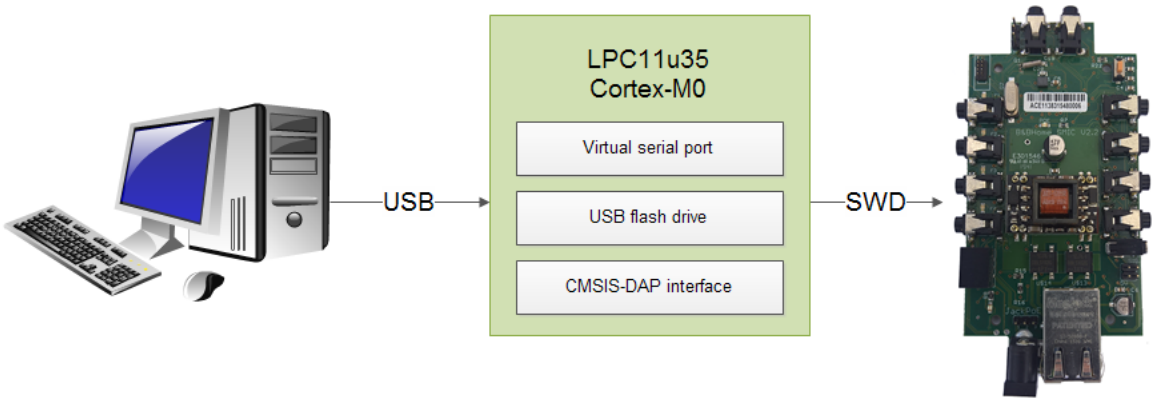
1.3.3 B&B Home Programmer

Gelijktijdig met de B&BHome SMIC is de B&BHome Programmer ontwikkeld. Deze maakt het mogelijk om de SMIC te programmeren via “drag ‘n drop”. Wanneer de programmer aangesloten wordt aan een PC zullen de drivers automatisch geïnstalleerd worden en zal de programmer verschijnen als USB flash station. Dat maakt het mogelijk om binairy (.bin) bestanden te kopiëren naar dit station. Deze binary bestanden zijn de programma’s die de IDE genereert na het uitvoeren van een ‘build’. Ze bevatten dus de firmware voor de processor van de SMIC.
Om dit mogelijk te maken moet er ook firmware op de programmer geplaatst worden. De gebruikte processor voor de programmer is een LPC11u35, dit is een Cortex-M0 processor van NXP. Het is een kleine en goedkope processor, maar het heeft ook geïntegreerde USB drivers. Dit maakt het mogelijk om firmware via USB op de processor te plaatsen. De firmware en de schema’s die nodig zijn om dit circuit te bouwen is open source en kan

gevonden worden op de mbed ontwikkelaars website [6]. Deze wordt officieel ondersteund door ARM.

Figuur 6: B&BHome Mbed Programmer

Vereenvoudigd voorgesteld is deze programmer een communicatie tool die enerzijds via USB communiceert met een PC en anderzijds via SWD communiceert met de te programmeren processor. Dit wordt voorgesteld in onderstaand blokschema. SWD is een afgeleide van het JTAG protocol.



Figuur 7: functionaliteit programmer blokschema

Zoals op het blokschema te zien, heeft de programmer dus drie functionaliteiten. Enerzijds dient het als flashstation om de te programmeren bestanden naar de programmer te kopiëren. Anderzijds communiceert het met de SMIC via een CMSIS-DAP. Dit is een gestandaardiseerde manier om met een ARM Cortex microcontroller te communiceren via USB. Het is net deze interface die de 'vertaling' van USB naar JTAG of SWD voorziet. Tenslotte kan de programmer hierdoor ook dienst doen als virtuele seriële poort, bijvoorbeeld nuttig bij het debuggen.

In de ontwikkelingsfase van de SMIC is de online programmeeromgeving van ARM mbed gebruikt [7]. Deze online IDE genereert het binary bestand dat met behulp van de B&B Home

programmer op de SMIC geplaatst wordt. Deze IDE biedt echter geen geavanceerde foutopsporing mogelijkheden. Hiervoor is ook een geavanceerdere debugtool nodig dan de B&B Programmer.

1.4 Probleemstelling

Voor de ontwikkeling van een reeks nieuwe producten is Bits & Bytes op zoek naar een stabiele ontwikkelomgeving voor producten, zoals de SMIC, gebaseerd op de huidige generatie ARM-processoren. De huidige gratis, online, tool [8] voor het schrijven en compileren van code voldoet niet om op een efficiënte manier software te ontwikkelen. Ook moet het mogelijk zijn om aan real-time foutopsporing te doen. Enkele belangrijke vereisten zijn:

- Snel en gemakkelijk programmeren van het flash-geheugen.
- Gericht en efficiënt foutopsporing toelaten.
- Goede prijs/kwaliteit verhouding van de te gebruiken software en debugprobe.
- De tools moeten bruikbaar zijn voor een breed gamma van ARM-processoren.

1.5 Doelstelling

Om in de toekomst op een efficiënte manier nieuwe producten te ontwikkelen moet er een vaste ontwikkelomgeving bepaald worden. Hiervoor moet het meest geschikte softwarepakket gekozen worden en de meest geschikte debugprobe. Met deze tools zal in de tweede fase ook de software voor de SMIC ontworpen worden.

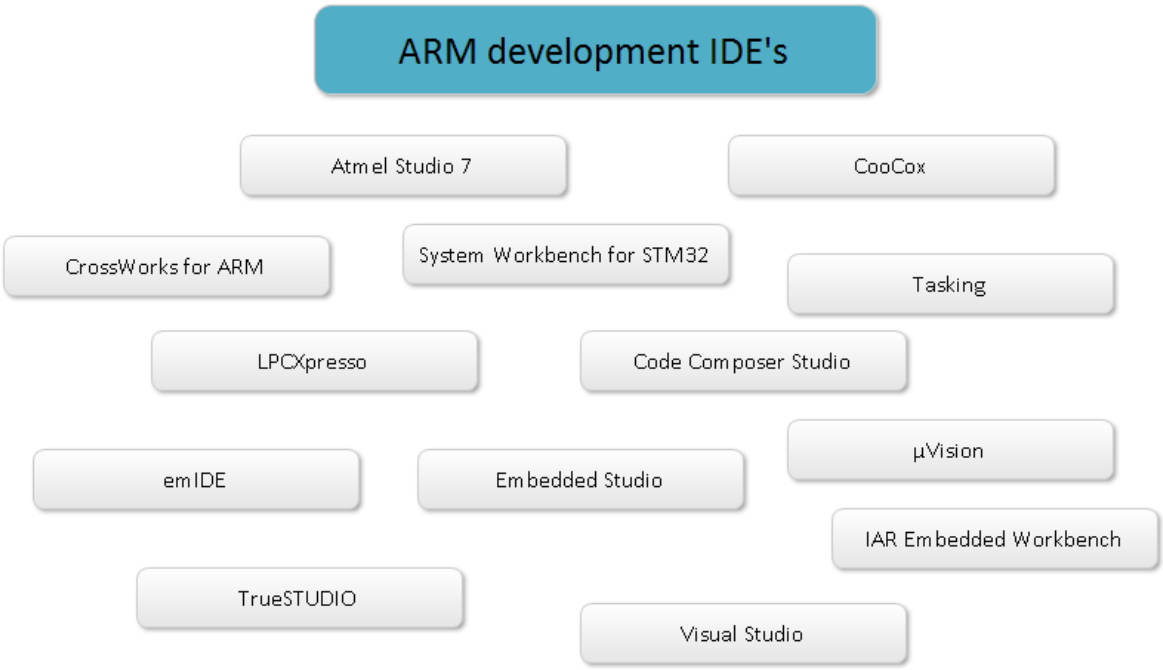
1.6 Realisatie

In eerste instantie zal er beslist worden welke processoren voldoen aan de vereisten van Bits & Bytes. Afhankelijk daarvan worden verschillende softwarepakketten vergeleken. De pakketten die in aanmerking komen worden, indien mogelijk, uitgetest en vergeleken met elkaar. Ook zullen enkele verdelers van deze softwarepakketten gecontacteerd worden om hun expertise en commerciële info te vergelijken met de uitgevoerde studie.
Uit deze studie zal een besluit getrokken worden. In samenspraak met Bits & Bytes zal er beslist worden welke software voor hun het meest geschikt is.
De tweede fase is het ontwikkelen van de code voor de SMIC. Hierbij zullen verschillende aspecten aan bod komen:

- TCP/IP communicatie met de server (ontwikkeling van een protocol);
- interrupt gebaseerde seriële communicatie;
- interrupt gebaseerd infrarood ontvangen;
- integratie van verschillende infrarood protocollen (RC-5, RC-6, NEC,…);
- aansturing van digitale IO.

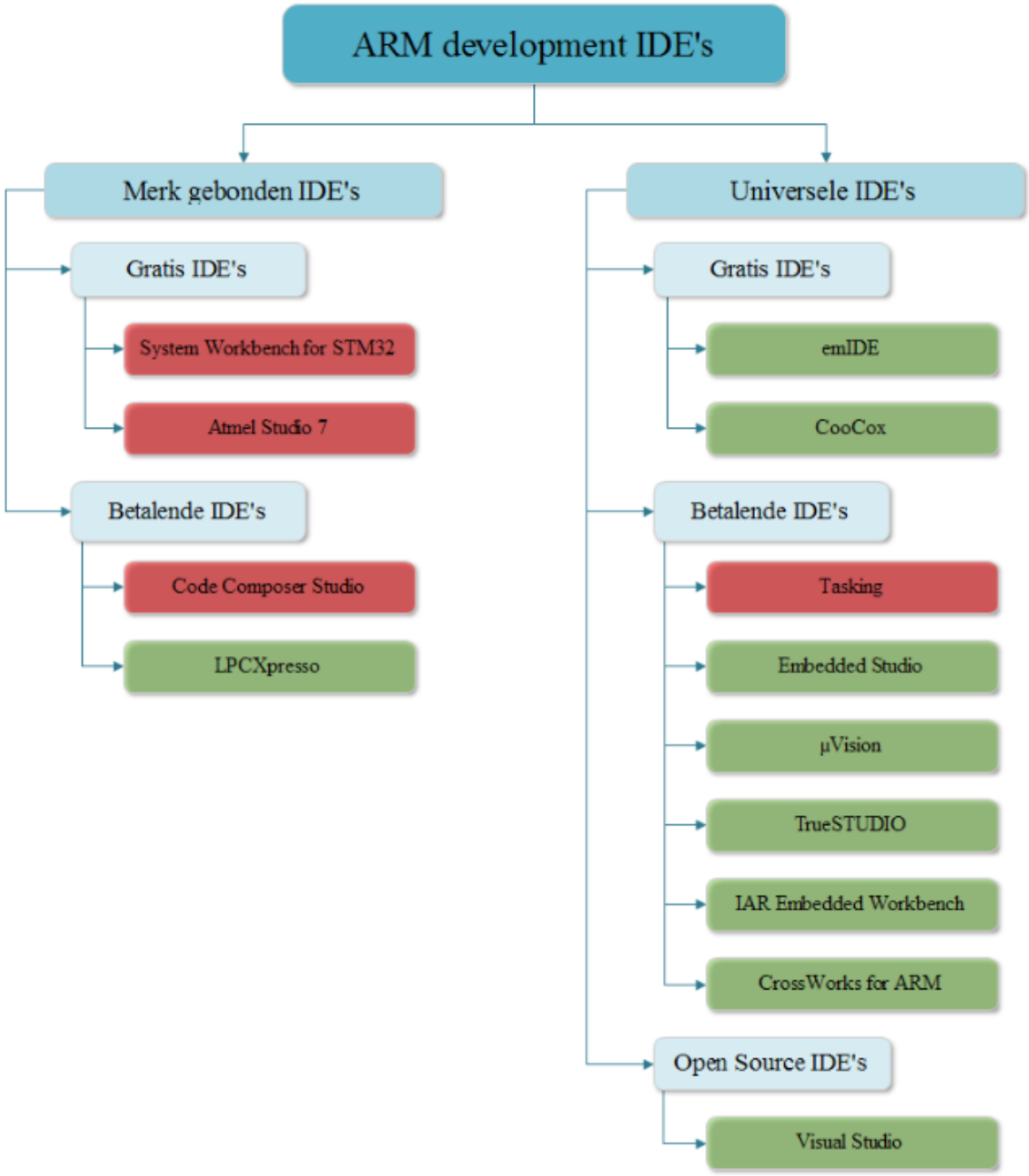
2 Vooronderzoek: ARM Development tools

Om te bepalen welke de meest geschikte IDE en debug probe zijn is er een literatuuronderzoek gebeurd naar de bestaande tools. Na een uitgebreid onderzoek zijn volgende IDE's gevonden:



Figuur 8: ARM Development IDE's

Vervolgens zijn de mogelijkheden van deze IDE's kort onderzocht en is onderstaand overzicht opgesteld. Iedere tool wordt ook kort besproken, en op basis van een eerste selectie zijn er al enkele tools die niet verder zullen worden opgenomen in het onderzoek.



Figuur 9: ARM Development IDE's geordend

2.1 Merk-specifieke tools

Enkele fabrikanten zoals STMicroelektronics, Microschip en Texas Instruments hebben hun eigen IDE voor ontwikkeling met hun processoren, meestal ook gekoppeld aan hun eigen debug probes. Onderstaand een kort overzicht.

System Workbench for STM32

Deze tool dient voor het schrijven van software voor STM32 32-bit ARM Cortex microcontrollers. De STM32 serie wordt geproduceerd door STMicroelectronics. Zij hebben een uitgebreide reeks aan ARM microcontrollers. Deze firma heeft geen eigen commerciële IDE voor softwareontwikkeling. Ze verwijzen wel door naar enkele andere tools zoals CoIDE, Atollic, IAR, etc. deze worden verder in deze tekst besproken. Er is wel een "Open STM32 Community", daar kan "System Workbench for STM32" gedownload worden. Dit is een open-source programma voor het programmeren van STM32 processoren. Omdat deze tool enkel STM32 processoren ondersteund is deze niet verder opgenomen in het onderzoek [9] [10].

Debug probe: ST-LinkV2

Dit is de debug probe van STMicroelektronics, net als bovenstaande IDE is deze enkel geschikt voor de STM8 en STM32 microcontroller familie. Daarom wordt deze debug probe niet verder onderzocht [11].

Atmel Studio 7

Deze IDE is van de firma Microchip. Het is een IDE voor de volledige reeks Atmel processoren, zowel voor 8 en 32 bits AVR microcontrollers als voor ARM Cortex-M gebaseerde microcontrollers.
Het grote voordeel is dat deze software gratis is, de beperkende factor is echter dat het enkel hun eigen Atmel serie microcontrollers ondersteunt. Indien het mogelijk is om de processorkeuze te beperken tot de Atmel-reeks is het zeker aangeraden om deze IDE verder te onderzoeken en uit te testen.

Debug probes Microchip: Atmel-ICE, AVR Dragon, JTAGICE3, AVR ONE!, SAM-ICE

Net als een geavanceerde IDE heeft Microchip ook verschillende debug probes en programmers voor de Atmel-serie. We gaan hier niet verder op in, omwille van dezelfde reden dat we Atmel Studio niet verder onderzoeken [12].

Code Composer Studio

Dit is IDE van Texas Instruments. Hier is een betalende versie van, alsook een 90-dagen proefversie. Het ondersteunt een breed portfolio aan Texas Instruments embedded processoren, maar biedt geen support voor andere merken. Ook deze IDE wordt dus niet verder vergeleken.

Debug probes:

Ook Texas Instruments heeft verschillende debug probes en programmers voor hun eigen reeks processoren [13].

LPCXpresso

Dit is een IDE van fabrikant NXP. Deze reeks is geschikt voor de LPC-familie. De tool kan gebruikt worden voor code ontwikkeling en foutopsporing. Nadelig is dat deze tool merk-specifiek is, maar de vereiste was om support te bieden voor LPC processoren, dus is deze tool wel nuttig om verder te onderzoeken. De gratis versie heeft een code restrictie tot 256 kb. De pro versie zou 495 dollar kosten maar NXP vermeldt dat er momenteel geen officiële verdeler is van hun software en debug probe. (okt. 2016) [14].

Debug probe: LPC-Link2

NXP biedt ook een low-cost debug probe aan die compatibel is met hun eigen IDE. Dit is een evaluatiebordje dat, mits installatie van de juiste firmware, geconfigureerd kan worden om verschillende ontwikkelingstools en IDE's te ondersteunen.

2.2 Universele tools:

Er zijn fabrikanten die tools ter beschikking stellen, of verkopen, die meerdere processor-merken ondersteunen. Dit zijn voor ons de gewenste IDE's. Er zijn in deze categorie gratis en/of open-source producten alsook commerciële software voor ARM-development.

Tasking

Tasking is een onderdeel van Altium. Dit is een fabrikant van allerhande geavanceerde softwaretools voor het ontwerpen van PCB's, FPGA's en dus ook embedded software. Deze IDE is echter meer gericht op embedded systemen voor de automotive markt. Het ondersteunt de Cortex-M familie, processoren van onder andere: Atmel, Freescale, Infineon Technologies, Silicon Labs, Cypress, STMicroelectronics en Texas Instruments. Veel merken maar geen NXP processoren. Om deze reden is deze IDE niet verder opgenomen in de vergelijking [15].

Embedded Studio

Dit is een krachtige IDE voor ARM microcontrollers. De producent "SEGGER" is gespecialiseerd in tools voor embedded systemen. Zij verkopen de IDE Embedded Studio, maar ook debug probes; deze worden verder in deze tekst besproken, alsook 'production programmers'. Er is een evaluatie- en een studentenversie beschikbaar. De software ondersteunt de gehele Cortex-M familie processoren, inclusief de LPC-serie van NXP. Deze tool zal verder onderzocht worden en vergeleken met andere interessante tools [16].

Debug probes: J-Link en J-Trace

SEGGER produceert ook zijn eigen debug probes, de J-Link en de J-Trace. Dit zijn probes die door zeer veel IDE's ondersteund worden zoals verder uit deze tekst zal blijken. Van de J-Link bestaat ook een educational versie, deze is aangekocht via de KU Leuven en is gebruikt voor het vergelijken en testen van de IDE's [17].

µVison

Dit is zowat de meest bekende IDE voor de ontwikkeling met ARM processoren. Het wordt geproduceerd door de firma Keil in samenwerking met de ARM. Keil is gespecialiseerd in Embedded development tools. Keil heeft verschillende versies, zoals DS-5 Development studio en Keil MDK. Maar de meest volledige versie is de µVision IDE. Het combineert projectmanagement, code ontwikkeling, foutopsporing en nog meer, in één omgeving. Het grote nadeel is het prijskaartje dat aan deze software vast hangt. De Lite / proefversie heeft een coderestrictie tot 32kb, de professionele software is duur. Een exacte prijsvergelijking volgt verder in deze tekst. µVision wordt wel verder in de vergelijking opgenomen want behalve de prijs zijn er niet meteen nadelen op te merken [18].

Debug probe: ULink

Net zoals SEGGER heeft Keil ook een eigen debug probe de "ULink". Er zijn hiervan drie versies: de ULink Pro, ULink Pro D, en de ULink 2. Net als de J-Link wordt de ULink door veel andere IDE's ondersteund [19].

TrueSTUDIO

Dit is sinds kort een bekende IDE van de firma Atollic. TrueSTUDIO is het enige product van Atollic. Zij zijn dus gespecialiseerd in ARM development. TrueSTUDIO is een zeer volledige IDE en ondersteunt zowat het hele Cortex-M gamma. Er zijn geavanceerde fout-opsporingsmogelijkheden en configuratie tools. Atollic produceert geen debug probes maar hun IDE is compatibel met tal van bekende debug probes, zoals de eerder besproken J-Link en ST-Link. De gratis versie bevat geen codelengte restrictie. De Pro-versie is betalend. Er kan echter ook een evaluatieversie van de Pro-versie aangevraagd worden. Dit is gebeurd voor dit onderzoek, om TrueSTUDIO te vergelijken met de andere IDE's. Met de Pro-versie is veel geavanceerdere foutopsporing mogelijk [20].

IAR Embedded Workbench

IAR Systems is net zoals SEGGER en Keil gespecialiseerd in tools voor ontwikkeling van embedded systemen. Echter niet enkel voor ARM-processoren maar het ondersteunt ook een hele reeks van andere architecturen. Er is een gratis 30 dagen proefversie te downloaden. Er zijn ook geavanceerde foutopsporingsmogelijkheden met "C-SPY", de geïntegreerde debugger van IAR [21].

Debug probe: I-Jet

IAR produceert ook hun eigen debug probe, de I-jet (trace). Deze bied verschillende debug mogelijkheden, het nadeel is dat deze probe merkgebonden is [22].

CrossWorks for ARM

Dit is ontwikkeling van Rowley Associates. Deze IDE was de basis voor Embedded Studio van SEGGER. Deze IDE ondersteunt een brede range aan Cortex-M processoren, alsook de Cortex-A, en -R reeks. Verder wordt er weinig info gegeven op hun site. De IDE is compatibel met de eerder besproken J-Link, ULINK2 en ST-Link. Er is een proefversie en een betalende versie. Na het uittesten van de proefversie zal een verdere vergelijking gemaakt worden [23].

emIDE

Dit is een gratis tool voor ontwikkeling van embedded applicaties. Het ondersteunt de eerder besproken J-link debug adapter. De IDE ondersteunt een breed gamma Cortex-M processoren, alsook LPC-serie van NXP. Het programma oogt eerder ietwat basic, maar het is gratis software dus zeker de moeite om verder te onderzoeken [24].

CooCox CoIDE

Ook dit is een gratis IDE die zich focust op de ontwikkeling voor Cortex-M processoren. Er is een grote community en code kan zeer gemakkelijk geïmporteerd en gedeeld worden met deze community [25].

Debug probe: CoLinkEx

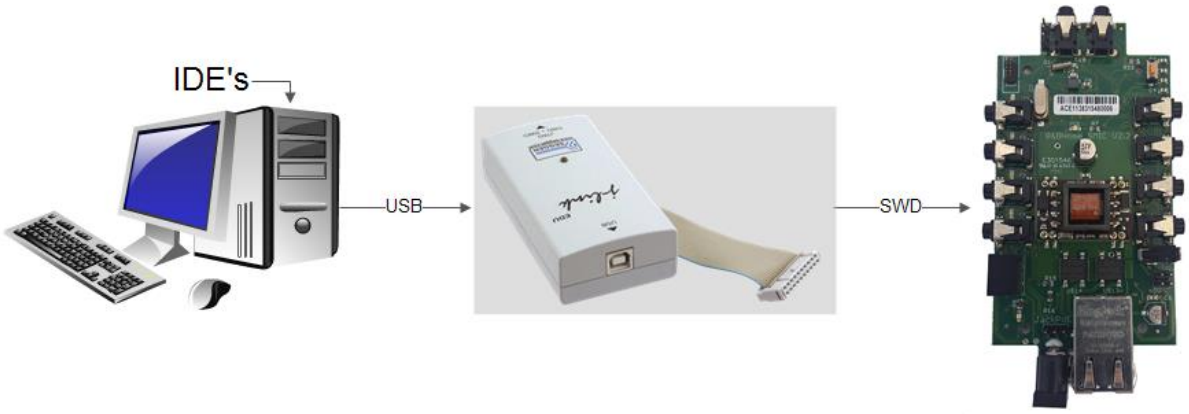
CooCox heeft ook een eigen (low-cost) debug adapter voor ARM Cortex-M microcontrollers. Het ondersteunt foutopsporing en het is een open platform. De firmware kan zonder kosten geüpdatet worden [26].

Visual Studio

Visual Studio is een zeer ruime IDE van Microsoft. Dit kan gebruikt worden voor het schrijven van mobiele apps, windows applicaties, tot webapplicaties en dergelijke. Maar mits het installeren van VisualGDB en nog enkele andere tools is het mogelijk om ook embedded software voor ARM processoren te ontwikkelen. Visual Studio heeft geen ingebouwde compiler voor ARM Cortex-M maar deze kan wel toegevoegd worden. Verder onderzoek van deze open-source mogelijkheden volgt verder in deze thesis [27].

2.3 Conclusie

Uit de literatuurstudie is gebleken dat er negen IDE's geschikt leken, na een eerste onderzoek. De bedoeling was om minder (vier à vijf) IDE's over te houden voor de praktische vergelijking. Er zijn echter geen extra argumenten gevonden om andere IDE's te elimineren. Daarom zullen, aan de hand van een eenvoudige test, de IDE's getest worden op gebruiksvriendelijkheid. Daarna kunnen de overblijvende IDE's uitvoeriger getest worden, en vergeleken worden met elkaar. Wat deze testen precies inhouden wordt verder in deze tekst besproken. Ook is gebleken dat bijna iedere fabrikant zijn eigen debugprobe heeft. Maar iedere IDE die besproken is, is ook compatibel met de J-Link debug probe van SEGGER. Er is een educational versie van deze probe aangekocht om de eerste testen mee uit te voeren. Zo krijgen we onderstaande testopstelling.



Figuur 10: Eerste testopstelling

3 Vergelijkende studie van IDE's

3.1 Globale studie

Uit het vorige hoofdstuk is gebleken dat er na de literatuurstudie negen geschikte IDE's gevonden zijn. Aangezien er aan de hand van de online documentatie van de fabrikanten geen IDE's konden geëlimineerd worden, waren bijkomende testen nodig. Al deze testen zijn uitgevoerd op een SMIC prototype, want al de software moet sowieso met deze hardware compatibel zijn.

Met iedere IDE is een uitgebreid "blinky programma" geschreven. Een "blinky programma" is een eenvoudig programma dat een led laat knipperen. Hiervoor zijn enkele gewone leds aangesloten op de IR-uitgangen van de SMIC. Er is een teller bijgevoegd om de debug mogelijkheden van de IDE te testen en de tellerwaarde te monitoren. Ter referentie is de online ontwikkelomgeving van het mbed platform gebruikt [8]. Dit is een basisontwikkelomgeving zonder debug mogelijkheden. Hierin kan zeer snel een project aangemaakt worden en code ontwikkeld voor bijvoorbeeld de LPC1768 processor van de SMIC. Ook is het mogelijk om software van deze online IDE te exporteren naar offline IDE's zoals diegene die in deze thesis worden besproken.

Het testprogramma zal een teller verhogen om de 0.5 seconden. Afhankelijk van de tellerwaarde zullen bepaalde leds branden. De bedoeling was om dit basisprogramma met iedere IDE te ontwikkelen, de processor te programmeren en met de J-Link debugprobe de tellerwaarde van de variabele "teller" te monitoren. Op basis van de gebruiksvriendelijkheid zijn de IDE's beoordeeld.

LPCXpresso

Bij de eerste test met deze IDE was er nog steeds geen officiële verdeler te vinden. Ook de supportaanvraag werd niet beantwoord. Met behulp van geïntegreerde voorbeelden in deze IDE is het niet gelukt om een programma te downloaden naar de SMIC. Door deze problemen en het feit dat deze IDE enkel geschikt is voor NXP processoren, is deze IDE geschrapt uit de lijst.

CooCox CoIDE

Tijdens het literatuuronderzoek leek deze tool veelbelovend. Echter op het moment van testen is de site, en dus ook de support voor een viertal weken offline geweest. Het is een gratis tool, maar support en continuïteit zijn zeer belangrijke factoren voor professioneel gebruik. Vanwege deze redenen is er niet meer verder gewerkt met deze IDE.

emIDE

De configuratie en koppeling van de J-Link debugprobe met deze IDE gaat vlot en intuïtief. Een eenvoudig tellerprogramma downloaden naar de SMIC is ook nog gemakkelijk. Wanneer er echter bibliotheken toegevoegd moeten worden voor het aansturen van GPIO, ontstaan er al snel veel compilatieproblemen. De verwijzing naar de fouten is echter niet nauwkeurig en soms zelfs verkeerd. Het is ook niet mogelijk om voor deze IDE een project te exporteren vanuit de online mbed IDE. Verder is dit een gratis IDE, waardoor het moeilijker is om professionele support te krijgen. Ook de online documentatie en fora zijn eerder beperkt. Deze IDE wordt dus niet verder in de vergelijking opgenomen.

µVision (trail versie)

Hierin is het mogelijk om een project van de online mbed IDE te importeren. Standaard is µVision geconfigureerd om de eigen ULink debug adapter te gebruiken. Deze instellingen aanpassen voor gebruik met de J-Link is een klein beetje omslachtig. Na correcte configuratie is het snel mogelijk om de demo werkend te krijgen. De debug mogelijkheden zien er ook opvallend uitgebreid uit. Deze zullen in de volgende fase, met de volledige versie van µVision, getest worden.

Embedded Studio

Deze IDE biedt de mogelijkheid om Keil µVision projecten te importeren. De online mbed IDE kan exporteren naar dit formaat. De fabrikant van deze IDE is dezelfde als die van de J-Link, de configuratie ging dus zeer vlot. Het ontwikkelen van de code, programmeren van de controller en starten van het debug proces gaan ook zeer vlot. Vandaar dat deze IDE nog verder getest zal worden.

IAR Embedded Workbench

Deze IDE biedt net als Embedded Studio de mogelijkheid om µVision projecten te importeren. Echter gaat het hier niet zo vlot. Er blijven compilatieproblemen optreden en de configuratie van de J-Link is redelijk omslachtig. Dit is waarschijnlijk omdat IAR eigen debugprobes heeft en standaard is de IDE geconfigureerd voor deze probes. Wanneer alle componenten van IAR gebruikt worden, lijkt dit een handige omgeving. In combinatie met de J-Link werkt het echter niet zo goed. Vandaar dat deze IDE niet verder in de vergelijking opgenomen wordt.

CrossWorks for ARM

Ook hier is het mogelijk om een µVision project te importeren. Echter bij het compileren gaan er verschillende zaken mis. Bij het instellen van de IDE voor gebruik met de J-Link is het programma zelfs enkele keren gecrasht. Omwille van deze reden wordt deze IDE niet verder vergeleken.

TrueSTUDIO

Het is niet zo eenvoudig om met deze IDE een basis, compileerbaar project te maken voor de SMIC. Zelfs aan de hand van een handleiding is dit in eerste instantie niet gelukt. De Eclips gebaseerde omgeving is ook niet zo intuïtief en gebruiksvriendelijk. Er is ook geen rechtstreekse export mogelijk vanuit de online mbed IDE naar deze offline tool. Deze IDE wordt dus ook niet verder vergeleken.

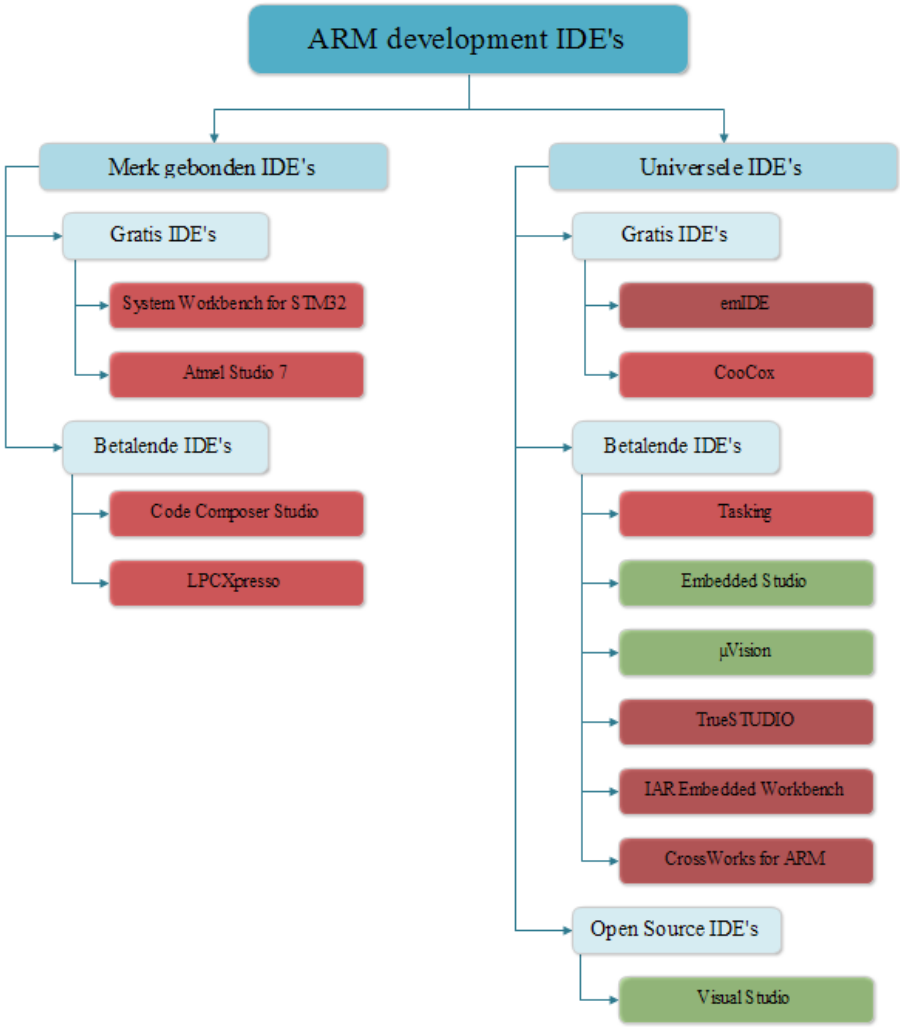
Visual Studio (VisualGDB)

Aan de hand van de “New Project wizard” is het met deze tool zeer gemakkelijk om een “blinky” voorbeeld aan te maken. In de wizard is de configuratie van de J-Link ook zeer eenvoudig. Zowel de standaard bibliotheken van NXP zijn beschikbaar, alsook de bibliotheken van mbed die op een hoger niveau liggen. Op gebied van gebruiksvriendelijkheid is deze tool op het eerste zicht bij één van de beste, vandaar dat deze verder in de vergelijking opgenomen zal worden.

Besluit

Over het algemeen is opgevallen dat Eclips gebaseerde IDE’s zoals TrueSTUDIO minder intuïtief en gebruiksvriendelijk zijn dan de andere. Uiteindelijk waren er drie IDE’s die er echt

boven uitstaken nl.: Embedded Studio van Segger, µVision van Keil en de VisualGDB plugin van Visual Studio (Microsoft). Zo krijgen we onderstaand overzicht. Met deze IDE's was het op een snelle en eenvoudige manier mogelijk om een programma te schrijven met de gekende bibliotheken, dit te compileren en te downloaden naar de SMIC. Ook het plaatsen van breekpunten om zo de code te analyseren gaat zeer vlot.



Figuur 11: overzicht IDE's na tweede selectie

Met de overblijvende IDE's zullen uitgebreidere testen uitgevoerd worden op gebied van foutopsporingmogelijkheden, versiebeheer en gebruiksvriendelijkheid.

3.2 Geavanceerde studie

In dit deel van het onderzoek zijn de overblijvende IDE's grondig getest en vergeleken. Aan de hand van testcode zijn de debugmogelijkheden uitgetest en is de gebruiksvriendelijkheid verder onderzocht.

3.2.1 Basis debugging

De code die gebruikt is voor de geavanceerde test is te vinden in "Bijlage B: Code geavanceerde test IDE's". Er is gebruik gemaakt van twee threads. In de ene thread zullen de leds knipperen, de andere thread wordt gebruikt om een infrarood signaal te ontvangen en op te slaan in een tijdelijk geheugen. Wanneer een nieuw signaal ontvangen wordt, zal dit het vorige overschrijven. Alle drie de IDE's kunnen breekpunten plaatsen, zowel in de

hoofdlus als in de interruptroutines. Ook is het mogelijk om de variabelen live te monitoren. De basis debug mogelijkheden zijn dus voor de IDE's ongeveer gelijk.

3.2.2 Extra features

Vooral µVision heeft een hoop extra features die nuttig kunnen zijn voor het debuggen. Er is onder andere een "instruction trace" functie. Hierbij kunnen alle uitgevoerde instructies gelogd worden. Maar de SMIC is hiervoor hardwarematig niet uitgerust met de juiste pinaansluiting. Voor toekomstige projecten is het interessant om deze aansluitingen toch te voorzien.
Ook beschikt µVision over een "logic analyzer". Hiermee kunnen de ADC en analoge in- / uitgangen gemonitord worden. Maar ook dit is voor de SMIC niet van toepassing omdat voor dit toestel geen analoge IO gebruikt worden.

3.2.3 Support

Zowel µVision als Embedded studio hebben een zeer uitgebreide handleiding, bij VisiualGDB is deze iets minder uitgebreid. Het contact met de supportdienst is wel zeer snel en efficiënt. Ieder probleem wordt binnen de 24h aangepakt en indien nodig wordt een persoonlijke build aangeleverd.

3.2.4 (Real Time) Operating System

Een belangrijk aspect voor de ontwikkeling van embedded software is het gebruikte Operating System (OS) en het vaak daarmee samenhorende Real Time Operating System (RTOS). VisualGDB ondersteunt zowel mbedOS en maakt gebruik van hetzelfde RTOS als µVision van Keil, namelijk 'Keil RTX'. MbedOS is een groot open-source OS dat door zeer veel ontwikkelaars gebruikt wordt. Dit is ook het OS wat voor eerdere projecten binnen Bits & Bytes gebruikt is. Er is dus ook geen extra kost voor het gebruik van dit OS en RTOS. Bij de aankoop van het ontwikkelpakket van Keil 'µVision', zijn de middleware componenten inbegrepen, dus ook het RTOS. Segger heeft een eigen RTOS namelijk 'embOS', maar het ondersteunt ook 'FreeRTOS', dit is zoals de naam doet vermoeden een gratis RTOS. Voor het gebruik van embOS moet er wel een extra pakket aangekocht worden bij Segger waarvan de prijs start bij €1245.

3.2.5 Prijsvergelijking

Iedere IDE bestaat meestal uit verschillende versies met ieder zijn prijs. Onderstaand een kort overzicht en prijsvergelijking.

Embedded studio:

De gratis versie heeft geen beperking in code lengte maar er zijn wel verschillende vergrendelde functionaliteiten die belangrijk zijn. De basisversie biedt de meest gebruikte debug functionaliteiten maar is snel beperkt qua bruikbare bibliotheken. Hoe duurder de versie, hoe meer bibliotheken (zoals RTOS) inbegrepen zijn.
Voor een volledig en actuele prijsvergelijking wordt er doorverwezen naar de website van de fabrikant:
https://www.segger.com/pricelist-embedded-studio.html

µVision:

µVision biedt een totaalpakket aan, alle zaken zoals RTOS en andere middleware componenten zijn inbegrepen in het pakket met de IDE. De duurdere versies bieden ondersteuning voor andere processorreeksen zoals de Cortex-A serie.
Voor een volledig en actuele prijsvergelijking wordt er doorverwezen naar de website van de fabrikant:
http://www2.keil.com/mdk5/uvision/

VisualGDB:

Doordat dit enkel een plugin is en geen volledige IDE is de prijs ook veel lager in vergelijking met de andere IDE's. Visual Studio is een bekende en gratis IDE met veel mogelijkheden, dat maakt de combinatie met VisualGDB net zo interessant. Ook VisualGDB heeft enkele versies, deze verschillen in debug functionaliteit en ondersteuning voor andere processorreeksen.
Voor een volledig en actuele prijsvergelijking wordt er doorverwezen naar de website van de fabrikant:
http://visualgdb.com/buy/

Hieronder staat een indicatieve prijsvergelijking van de verschillende versies die voor deze vergelijking van toepassing kunnen zijn.

Tabel 1: Prijsvergelijking IDE's

Embedded studio		**VisualGDB**		**µVision**	
Cortex-M edition	€ 998,00	Embedded	€ 80,00	Cortex-M	€ 3 340,00
ARM edition	€ 1 498,00	Custom	€ 160,00	Plus	€ 5 740,00
Pro - Cortex-M edition	€ 9 980,00	Ultimate	€ 289,00	Professional	€ 8 260,00

3.3 Besluit

Om te bepalen welke de meest geschikte IDE is, werd onderstaande tabel opgesteld. De scores en het gewicht zijn nog objectief bepaald door de tester, maar ze geven een goede indicatie van de ervaringen met de IDE's. De scores zijn in overleg met de ontwikkelaars en het bedrijf toegekend om op die manier een besluit te vormen.

Tabel 2: Vergelijk IDE's

Vereiste / optie / eig.	Citeria	Gewicht	Embedded Studio	µVision	VisualStudio + VisualGDB	Embedded Studio	µVision	VisualStudio + VisualGDB
Vereiste	prijs	8	3	2	9	24	16	72
Vereiste	gebruiksvriendelijkheid	9	7	8	9	63	72	81
Vereiste	support	9	8	8	8	72	72	72
Vereiste	gebruik van breekpunten	10	8	8	8	80	80	80
Vereiste	debug in RTOS omgeving	9	8	8	8	72	72	72
Vereiste	variable watch	10	7	8	7	70	80	70
Vereiste	compatibel met MBED compiler	8	5	5	9	40	40	72
Eigenschap	Gebruik RTOS	7	6	6	8	42	42	56
Eigenschap	toegang tot (gratis) bibliotheken	9	3	5	7	27	45	63
optie	werking intellisense	7	6	6	8	42	42	56
optie	Eventrecorder	8	0	10	0	0	80	0
optie	ETM-Tracing	8	0	10	0	0	80	0
optie	Logic Analyzer	8	0	10	0	0	80	0
			totaalscore zonder opties			490	519	638
			totaalscore met opties			532	801	694

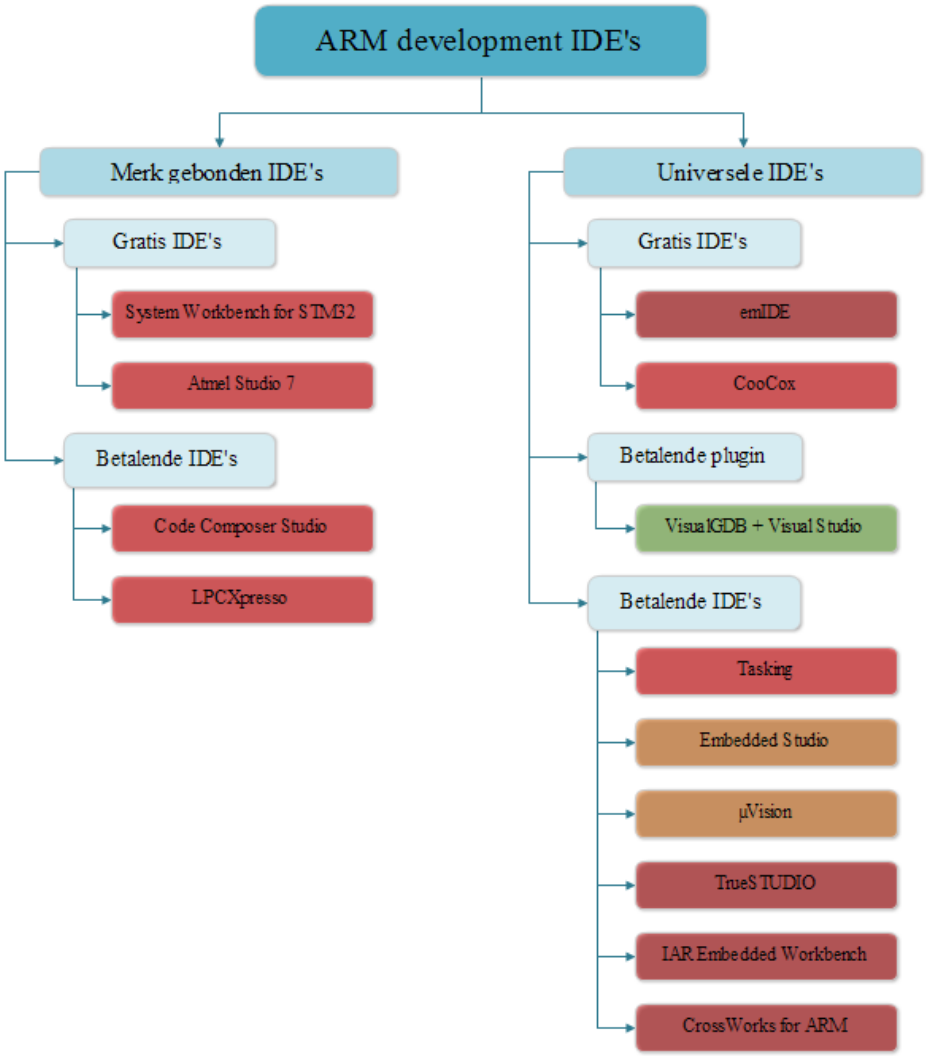
We kunnen hiermee twee rankings opstellen. Als we alle features bekijken dan bekomen we onderstaande ranking:

1) µVision (801)
2) VisualStudio + VisualGDB (694)
3) Embedded Studio (532)

Deze ranking geeft aan wat de beste IDE is op gebied van prijs/kwaliteit. Er wordt echter geen rekening gehouden met de toepasbaarheid voor deze IDE voor de projecten van Bits & Bytes. Wanneer we hiermee rekening gaan houden kunnen we een tweede ranking opstellen.

1) VisualStudio + VisualGDB (638)
2) µVision (519)
3) Embedded Studio (490)

We zien dat de plugin VisualGDB hieruit als beste komt. Dit komt vooral door het grote prijsverschil en de voorgeschiedenis van het gebruik van mbedOS binnen Bits & Bytes nv. Er is dan ook besloten om de 'custom' versie van deze tool aan te kopen. Deze biedt alle uitgebreide test- en debugfuncties die nodig zijn. Tenslotte kunnen we een laatste overzicht maken van de onderzochte IDE's.



Figuur 12: finaal resultaat vergelijk IDE's

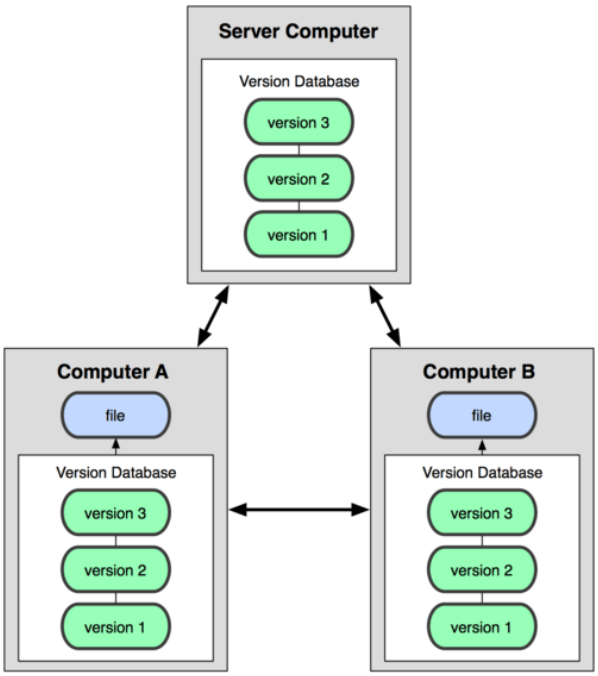
4 Versiebeheer

Binnen het concept "efficiënte softwareontwikkeling" is versiebeheer tegenwoordig niet meer weg te denken. Versiebeheer is een systeem om veranderingen in een bestand of een groep van bestanden bij te houden. In principe kan dit voor iedere vorm van bestand of project. Onderstaand wordt versiebeheer besproken toegepast voor softwareontwikkeling.

Wanneer een programmeur bepaalde werkende software heeft geschreven zal deze hiervan een back-up willen maken. Dit kan bijvoorbeeld door de broncode te kopiëren naar een nieuwe map. De naam van de map kan de datum, versie en naam van het project bevatten. Dit is een manier om versiebeheer toe te passen. Je kan zonder problemen terug gaan naar een vorige versie. Maar deze methode is ook zeer foutgevoelig. Je kan bijvoorbeeld vergeten welke versie nu juist in welke map zit, of onbedoeld bestanden heen en weer kopiëren.
Om deze problemen te vermijden bestaan er vele versiebeheersystemen. Een op dit moment zeer bekend en veelgebruikt systeem is 'Git', ook wel bekend van 'GitHub'. Dit is de meest populaire hostingsite gebaseerd op Git.

4.1 Git

Git is een gedistribueerd versiebeheersysteem, dit wil zeggen dat clients de hele opslagplaats ('repository') kopiëren. Wanneer de server of één van de clients crashen dan kan de data van een andere locatie terug gebruikt worden om de server te herstellen. Iedere checkout is dus een volledige back-up van alle data. Dit wordt voorgesteld op onderstaande figuur.



Figuur 13: Diagram van een gedistribueerd versiebeheersysteem[28]

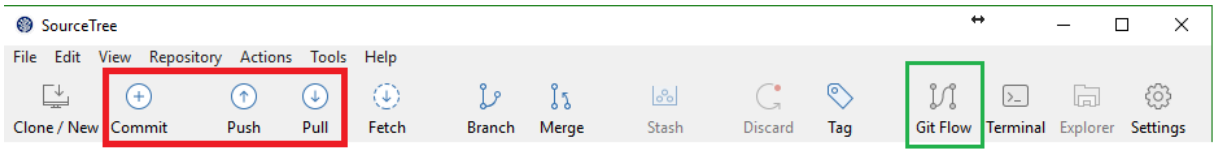
Git beheert de bestanden op een zeer efficiënte manier. De ongewijzigde bestanden zal Git niet elke keer opnieuw opslaan, maar enkel een verwijzing naar het eerder identieke bestand wordt bijgehouden. Hierin onderscheidt Git zich ten opzichte van vele andere versiebeheersystemen. Git heeft echter nog andere voordelen. Doordat de volledige kopie lokaal staat is er geen info van een andere computer in het netwerk nodig. Dit maakt dat Git zeer snel kan werken. Je kan dus ook offline werken. Git heeft ook integriteit, dit wil zeggen dat alles wat met Git wordt opgeslagen een controlegetal ('checksum') krijgt. Hierdoor is het onmogelijk om een map of bestand te wijzigen zonder dat Git ervan weet. Het zorgt er ook

voor dat er geen informatie kwijt kan geraken, of bestanden corrupt kunnen geraken. Met het gebruik van Git is het ook zeer moeilijk om een actie te doen die niet ongedaan gemaakt kan worden. Je kan experimenteren zonder het gevaar te lopen jezelf in nesten te werken.

4.2 SourceTree

Git kan je gebruiken in commandolijn modus. Via een gitbash is het mogelijk om alle acties met Git uit te voeren. Hiervoor zal je de Git syntax moeten leren kennen. Er zijn echter ook grafische tools die het werken met Git mogelijk gebruiksvriendelijker en efficiënter maken. SourceTree is een voorbeeld van zo een (gratis) tool. Met deze tool kan je Git repositories toevoegen, beheren, en branches visualiseren.

De drie voornaamste acties die je met Git kan uitvoeren zijn 'pull', 'commit' en 'push'. Met een 'pull' zal je een bepaalde versie van een opslagplaats kopiëren van de server naar je lokale werkplek (desktop / laptop). Via de 'commit' actie(s) worden wijzigingen opgeslagen en voorzien van commentaar. Vervolgens wordt, met een 'push', de lokale versie naar de server gekopieerd en samengevoegd met de bestaande versie op de server. Deze acties zijn met SourceTree eenvoudig uit te voeren.

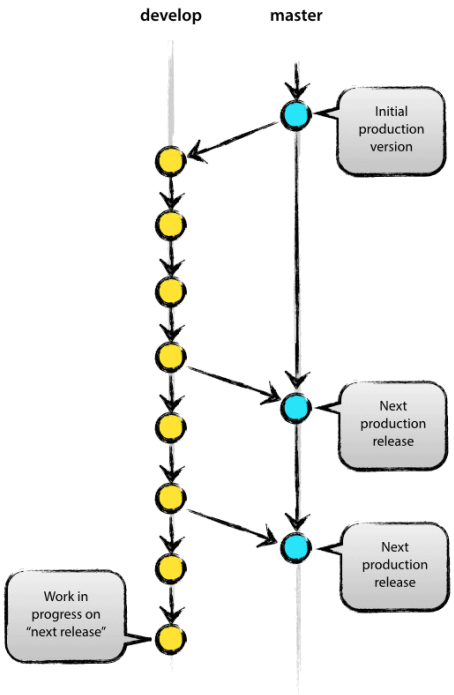


Figuur 14: Source tree controls

4.3 GitFlow

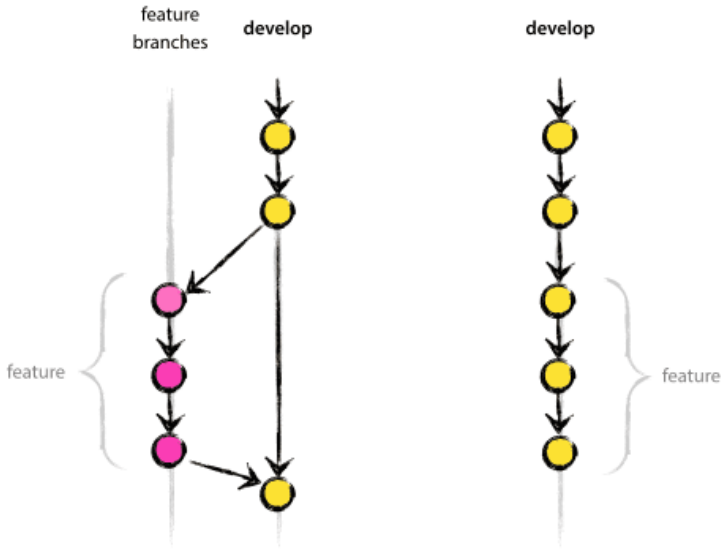
Voor een efficiënt gebruik van Git worden 'branches' gebruikt. Een 'branch' kan voorgesteld worden als een tijdlijn. Op deze tijdlijnen worden mijlpalen aangeduid, deze stellen de commits voor. Hoe deze branches precies gebruikt worden is volledig naar de keuze van de gebruiker. Er zijn echter bepaalde werkwijzen en structuren die meer en meer standaard worden in de wereld van softwareontwikkeling. Eén van die methodes is 'Git Flow' die ook geïntegreerd is in SourceTree.

Aan de basis van dit model zijn er twee branches: 'master' en 'develop'. Deze "tijdlijnen" lopen oneindig door en lopen parallel naast elkaar. Op de 'master branch' worden de softwareversies bijgehouden die 'production-ready' zijn. Dit is een softwareversie die volledig klaar is. De 'develop branch' dient voor de ontwikkeling van software, hierin worden alle versies bijgehouden gedurende het ontwikkelingsproces. Van zodra een nieuwe versie van de software klaar is zal de 'develop branch' samengevoegd ('merge') worden met de 'master branch'.



Figuur 15: Git flow 1[29]

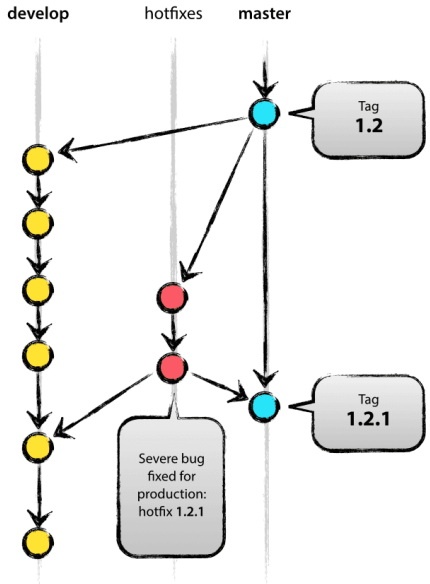
Het GitFlow model bevat echter ook nog enkele ondersteunende branches. Zo is er de 'feature branch', deze wordt gebruikt voor het ontwikkelen van nieuwe software specificaties of onderdelen van de software. Bij het aanmaken van een 'feature branch' zal er een kopie gemaakt worden van de laatste development versie. Wanneer de feature klaar is zal deze samengevoegd worden met de, op dat moment, meest recente development versie. Zo kan deze feature meegenomen worden naar een komende softwarerelease of geweigerd worden wanneer de testresultaten tegenvallen.



Figuur 16: Git flow 2[29]

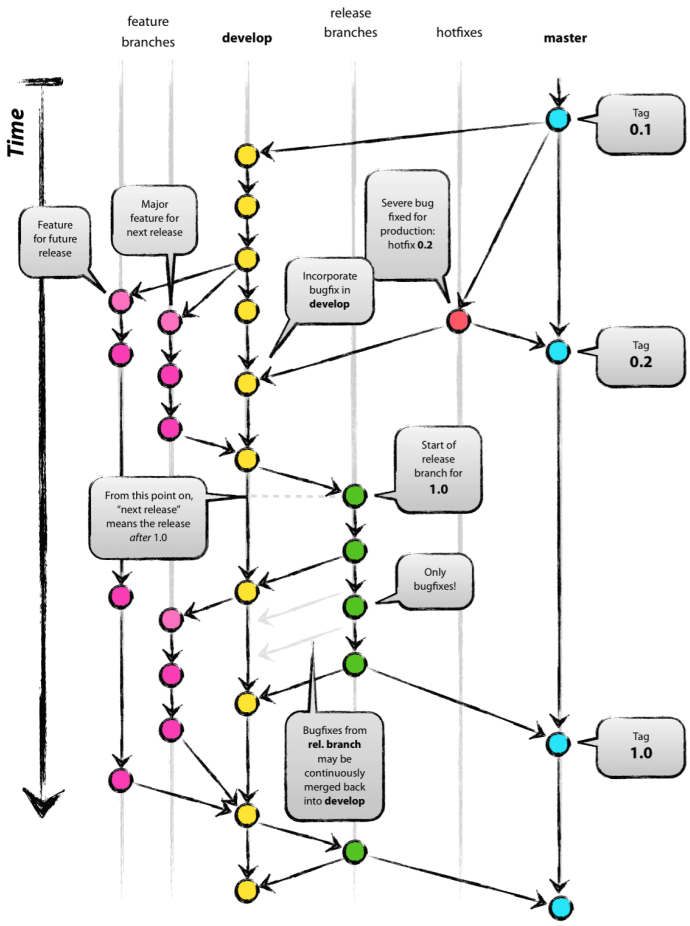
Verder is er ook de 'release branch'. Deze wordt gebruikt net voor een production release, in deze fase wordt de software afgewerkt. Zo kunnen er versienummers toegevoegd worden, build datums etc. Doordat deze (belangrijke) details op een aparte branch gebeuren kan de develop branch ondertussen nieuwe features ontvangen voor een volgende release. Wanneer de versie in de release branch volledig klaar is zal deze branch samengevoegd worden met

zowel de develop branch als de master branch. Op de master branch verschijnt dan de nieuwste versie. Er kan dan een 'Tag' geplaatst worden om een referentie te maken naar deze versie. Ook wordt de release branch op dit moment samengevoegd met de develop branch. Zo zijn alle eventuele oplossingen die in de release gemaakt zijn up-to-date in de develop branch.
Tenslotte is er nog de 'hotfix branch'. Deze dient om eventuele problemen die ontdekt worden in gereleasete versies snel op te lossen. Kritieke fouten worden in deze branch opgelost waarna de nieuwe versie met de master en develop branch samengevoegd worden. Zo zal er een nieuwe (sub)versie gereleaset worden en is de development branch uiteraard voorzien van de nieuwe oplossingen.



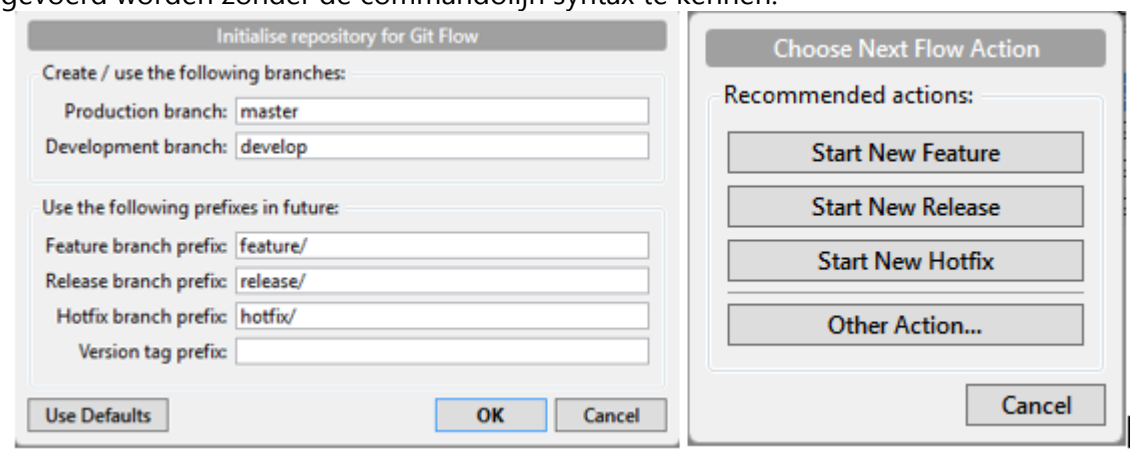
Figuur 17: git flow 3[29]

Een volledig Git Flow model kan dus op onderstaande manier voorgesteld worden.



Figuur 18: Git flow 4[29]

Alle acties kunnen volledig via commando's uitgevoerd worden, maar zijn ook geïntegreerd in SourceTree. Wanneer Git Flow geïnitialiseerd is kunnen de verschillende acties eenvoudig uitgevoerd worden zonder de commandolijn syntax te kennen.



Figuur 19: SourceTree Git flow initialisation

Een voorbeeld van het gebruik van Git met SourceTree volgens de 'regels' van GitFlow is terug te vinden in de bijlage.

5 Softwareontwikkeling voor de B&BHome SMIC

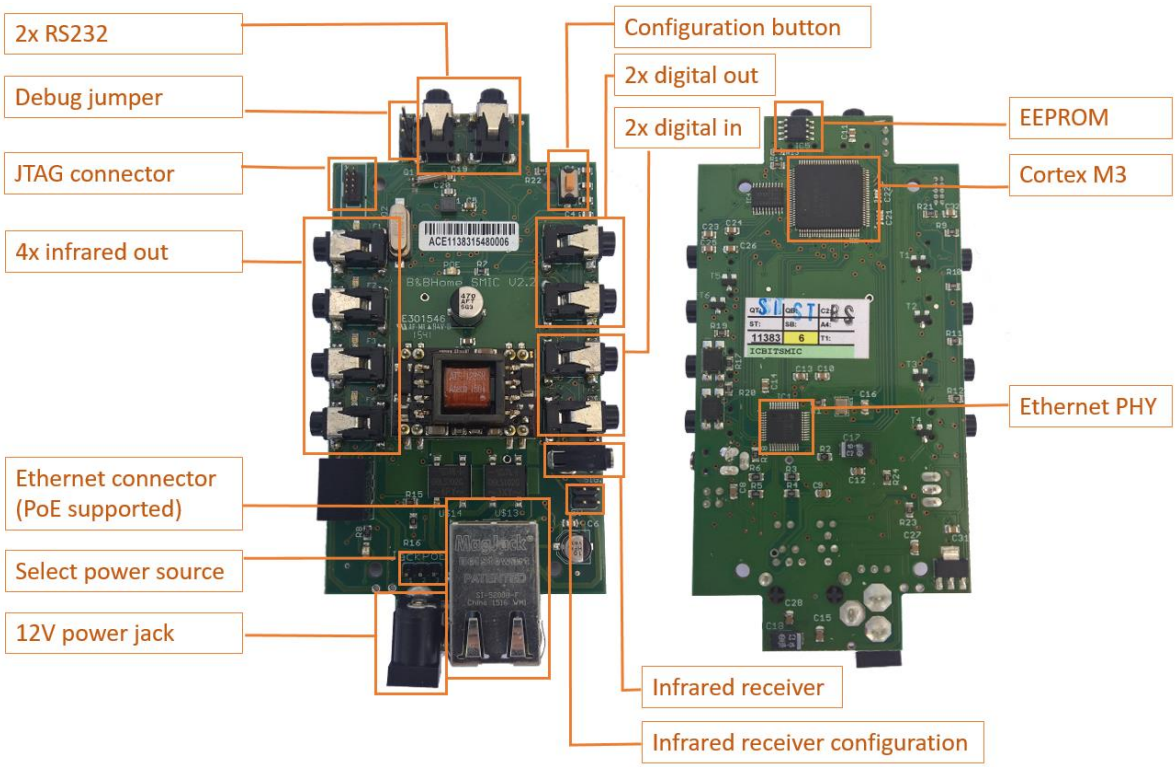
In dit deel wordt de ontwikkeling van software voor de B&BHome Smart Media IP Controller besproken. De tools en werkmethoden die in voorgaande hoofdstukken besproken zijn worden hiervoor toegepast.

Eerst wordt de aan te sturen hardware besproken en vervolgens de nodige achtergrond informatie, die nodig is voor het ontwikkelen van de code. Hierin worden onder andere de verschillende infrarood protocollen besproken. Vervolgens worden de softwarestructuur en de verschillende testen die zijn uitgevoerd besproken.

Tenslotte is er ook nog een hoofdstuk over softwaredocumentatie.

5.1 Omschrijving van de hardware

De SMIC bevat verschillende onderdelen die het mogelijk maken om op veel manieren te communiceren met andere apparaten. Op onderstaande foto is een overzicht te zien:



Figuur 20: Hardwareoverzicht SMIC

De meest voorname verbinding is de netwerkverbinding. Doormiddel van een ethernet PHY kan een verbinding opgezet worden en zal de SMIC communiceren met de server via een TCP/IP-verbinding. PHY is een afkorting voor de fysieke laag (physical layer) van het OSI model. Een PHY verbindt de link laag, ook wel MAC laag genoemd, met het fysieke medium, de kabel. Het zorgt dus voor het zenden en ontvangen van ethernet frames.

Vervolgens zijn er verschillende transportmedia aanwezig die zorgen voor het aansturen en lezen van de media-apparatuur.
Allereerst zijn er vier infrarood uitgangen. Doordat hiervoor gewone IO pinnen van de processor gebruikt zijn kan het volledige signaal gegenereerd worden door de processor en

is het toestel in staat om data via allerhande protocollen te zenden. De aansluiting gebeurt met 3.5mm mono jacks. Vervolgens wordt met behulp van een mosfetschakeling een infrarood led gestuurd. Er is ook een oranje led en de nodige voorschakelweerstand toegevoegd. De infrarood led zal zorgen voor de datatransmissie, de oranje led is ter indicatie omdat infraroodlicht niet zichtbaar is. Met deze infraroodschakelingen zal het mogelijk zijn om de werking van afstandsbedieningen na te bootsen.

De SMIC kan niet enkel infrarood commando's zenden, maar ook ontvangen. Hiervoor is één kanaal voorzien. De ontvangen infrarood commando's worden gedecodeerd en via de TCP/IP verbinding naar de server gestuurd. Ook hier gebruiken we een gewone IO pin. Hierop kan dan eender welke IR receiver aangesloten worden. Via een 2.5mm stereo jack wordt de receiver gevoed met 5V, verder is ook de grond en de IO pin aangesloten. De meeste infrarood receivers bevatten een interne demodulator. Dit wil zeggen dat ze zelf de carrierfrequentie weg filteren.

Er zijn ook twee RS232 in-/uitgangen voorzien op de SMIC. Deze zijn via een SP3223 IC verbonden met de processor. De SP3223 zal het spanningsverschil van 3.3V naar 12V wegwerken op een intelligente manier. Dit wil zeggen dat het IC in rust is wanneer er geen datatransmissie is. In rust verbruikt het IC slechts 1µA. Via de seriële poorten kunnen TV's, projectoren en dergelijke aangestuurd worden. Deze toestellen kunnen via de seriële interface ook feedback geven, bijvoorbeeld om aan te geven dat ze aangeschakeld zijn.

Verder zijn er twee digitale ingangen en uitgangen voorzien. Voor de uitgangen worden ook hier mosfets gebruikt die zorgen dat er 12V pulsen gestuurd kunnen worden. De uitgangen kunnen bijvoorbeeld dienen voor de aansturing van een triggerpuls van 12V, voor het aanschakelen van een audioversterker. De ingangen zijn galvanisch gescheiden. Doormiddel van een optocoupler-schakeling kan er veilig hardware aangesloten worden. Deze kunnen bijvoorbeeld dienen voor de stroommeting van een toestel.

Tenslotte zijn er nog enkele hardware elementen die zorgen voor de werking en configuratie van de SMIC. Er is een EEPROM geheugen voorzien waarin instellingen kunnen opgeslagen worden. Deze 16K EEPROM van Microchip (24AA16) bestaat uit 8 blokken van 256 bytes. Via een I2C verbinding kan er van / naar dit geheugen gelezen en geschreven worden.
Verder zijn er nog enkele jumpers voorzien die zorgen voor de voedings- en debug instellingen.

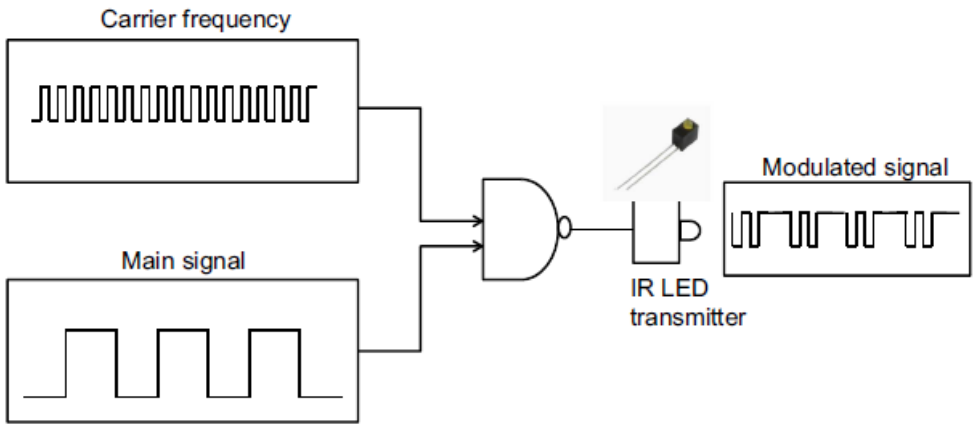
5.2 Achtergrondinformatie

Om de software op te bouwen is er nog informatie nodig met betrekking tot de gebruikte protocollen, zowel voor het infrarood als het seriële gedeelte. Deze informatie is tot stand gekomen door het vergaren van online documentatie, raadpleging van verschillende fora en het 'revers engineeren' van verschillende afstandsbedieningen.

5.2.1 Beschrijving van infraroodprotocollen

Voor de SMIC zijn er vijf protocollen geïmplementeerd. Onderstaand een beknopte maar nuttige samenvatting van de protocollen. Deze gegevens zijn belangrijk voor het juist zenden, en ontvangen van de infrarood codes. De software voor het zenden en ontvangen van infraroodsignalen is opgebouwd op basis van deze tabellen.

Infrarood protocollen zijn gebaseerd op een gemoduleerd signaal. De protocollen die in dit project toegepast zijn maken gebruik van een 36 of 38kHz carrierfrequentie. Dit kan symbolisch als volgt voorgesteld worden.



Figuur 21: Infrarood modulatie[30]

Het carriersignaal moet typisch een dutycycle hebben van 25 à 30%. Wanneer dit berekend wordt voor de gebruikte frequenties bekomen we onderstaande gegevens.

Tabel 3: Berekening carrier signaal

Frequentie	Periode (1/Frequentie)	1/3 periode ,T high	2/3 periode, T low
36kHz	27.777778µs	9.26µs	18.52µs
38kHz	26.315789µs	8.77µs	17.54µs

Hieruit besluiten we dat we met een T high van 9µs en een T low van 18µs weinig afwijken van de specificaties. We hebben deze waardes nodig wanneer we ons carriersignaal moeten genereren bij het zenden van een IR protocol.

Figuur 22: Carriersignaal grafische voorstelling

Voor het ontvangen van infrarood signalen wordt een ontvanger gebruikt met ingebouwde demodulator. Zo ontvangen we enkel het "main signal" op de processor. De data die het "main signal" bevat is protocol afhankelijk. Softwarematig zal iedere hoog- en laagperiode

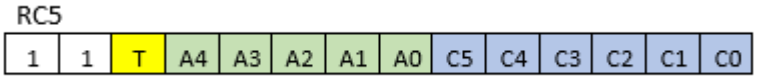
van het infraroodsignaal gemeten worden. Afhankelijk van deze tijden kan het gebruikte protocol bepaald worden.

De timing van de infrarood signalen kan uiteraard nooit exact zijn. Door de transmissie kunnen de tijden wijzigen. Fysische verschijnselen zoals diffractie en reflectie kunnen de timing van het signaal beïnvloeden. Wanneer we een infrarood signaal ontvangen en willen decoderen moeten we dus rekening houden met deze timing problemen. Daarom rekenen we steeds een marge van +20% en -20% op de theoretische timing. Door het berekenen van deze tijden en het analyseren van de protocollen kunnen we voorwaarden opstellen waarmee we softwarematig de infraroodsignalen kunnen classificeren volgens het overeenkomstige protocol.

RC5 protocol

Dit protocol is ontwikkeld door Phillips en wordt door veel fabrikanten gebruikt. Het grote voordeel van dit protocol is dat er extra adressen voorzien zijn voor de aansturing van verschillende soorten apparatuur. Door dit open protocol is het mogelijk om zelf functionaliteit toe te voegen.

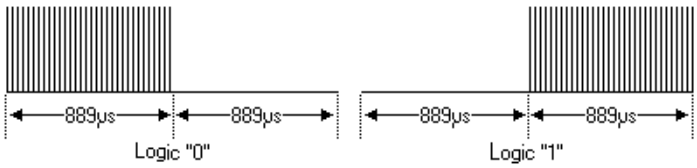
Een RC5 code start steeds met twee startbits, gevolgd door een togglebit en vervolgens een 5-bit adres en een 6-bit commando. We kunnen dit grafisch op onderstaande manier voorstellen.



Figuur 23: RC5 code

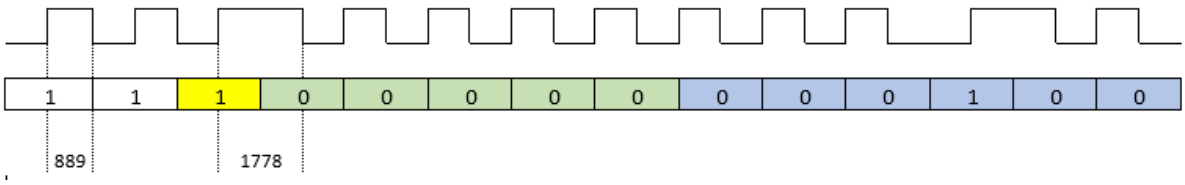
De togglebit is toegevoegd om te detecteren of een knop ingedrukt blijft of meermaals ingedrukt wordt. Stel dat de gebruiker het getal 11 naar een TV wil sturen moeten we een onderscheid kunnen maken tussen het ingedrukt houden van knop "1" of het herhaaldelijk indrukken van de knop. Hiervoor zal bij iedere druk op een knop de togglebit van waarde veranderen. Wanneer de knop ingedrukt blijft, blijft de waarde van de togglebit onveranderd.

Het RC5 protocol maakt gebruik van bi-fase modulatie, ook wel "Manchester coding" genoemd. Dit wil zeggen dat een logische '1' voorgesteld wordt door een signaal dat 889µs hoog is en 889µs laag. Een logische '0' wordt gemaakt met het omgekeerde, laag gevolgd door hoog.



Figuur 24: RC5 modulatie [31]

Wanneer we het signaal grafisch uittekenen zien we dat er door het al dan niet op één volgen van een '1' en een '0' zich twee periodetijden kunnen voordoen. Namelijk 889µs en 1778µs. Deze tijden zijn voornaam voor de programmatie van het infrarood zenden en ontvangen.



Figuur 25: RC5 voorbeeld pulstrein

Samengevat:

Tabel 4: samenvatting RC5 tijden

RC5			
Carrier frequentie	36kHz	**- 20% marge**	**+ 20% marge**
Carrier - tijd hoog	9 µs	20% marge van één periode: 178 µs	
Carrier - tijd laag	18 µs		
Signaal - tijd één periode hoog/laag	889 µs	711 µs	1067 µs
Signaal - tijd twee periodes hoog/laag	1778 µs	1600 µs	1956µs
Signaal - totaal (14 bits)	24892 µs	19914 µs	29870 µs

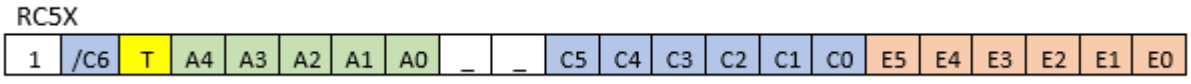
Een RC5 signaal kan herkend worden aan de hand van volgende eigenschappen:

- De totale tijd moet tussen bovenstaande parameters liggen.
- De eerste gemeten tijd moet altijd één of twee periodes lang zijn.
- De langst gemeten tijd kan maximum één of twee periodes lang zijn.

RC5X protocol

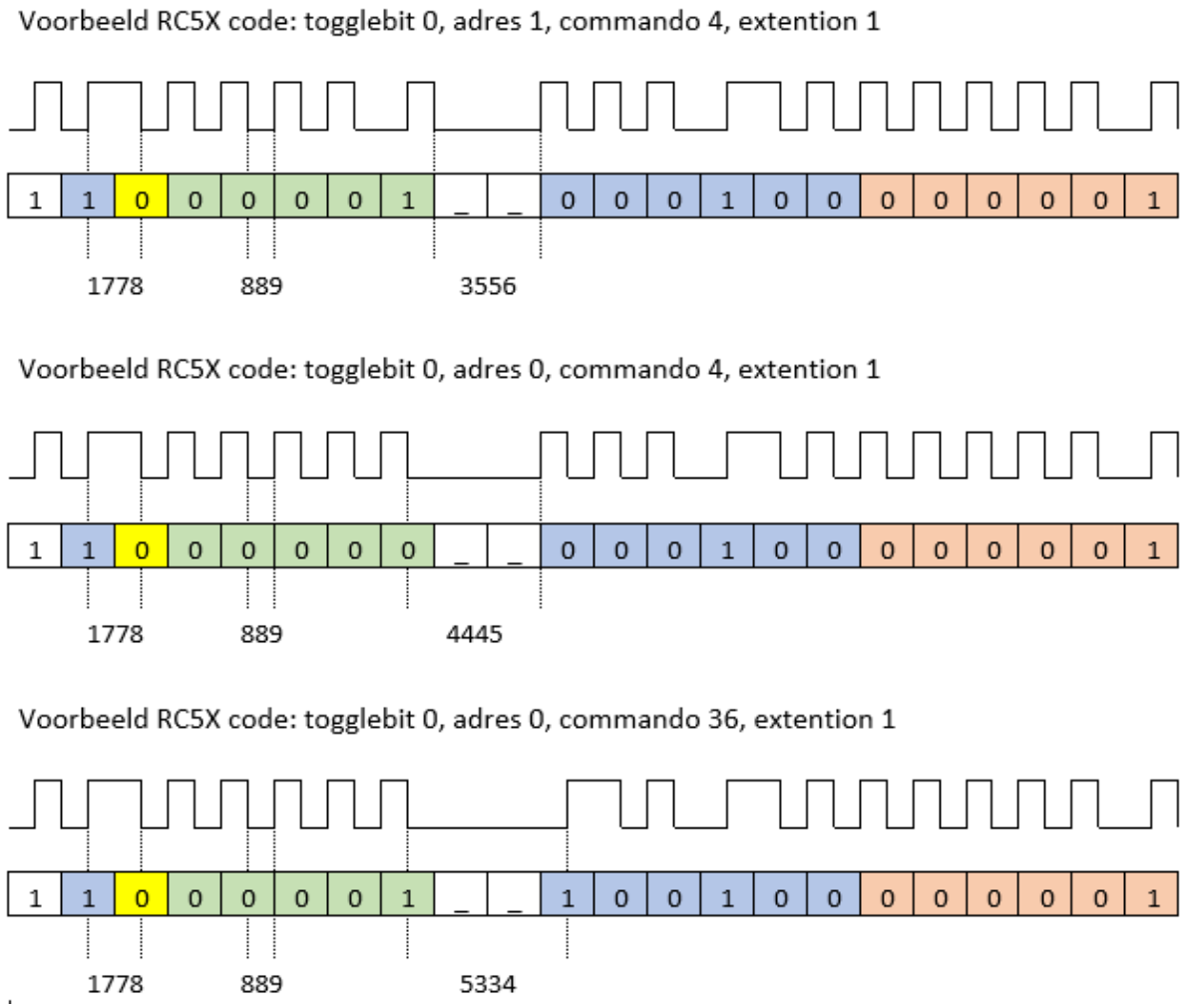
De 'X' in de naam van dit protocol staat voor 'extended'. Dit is dus een uitbreiding op het hiervoor besproken RC5 protocol. Dit protocol wordt onder andere gebruikt voor de aansturing van een Marantz audio versterker. Met de extention-data kunnen we bijvoorbeeld een volume instellen.

Adres -> adres van de versterker,
Commando -> "volume"
Extention -> "niveau".



Figuur 26: RC5X code

De tweede startbit wordt bij dit protocol gebruikt om het commando met één bit uit te breiden. De bit zal de inverse waarde van de meest betekenende commando bit voorstellen. Verder wordt het protocol uitgebreid met 6 extra bits. Om aan te geven dat het signaal gecodeerd is met een RC5X protocol wordt tussen het zenden van het adres en het commando de communicatie twee bittijden laag gehouden. Hierdoor krijgen we vijf verschillende periodetijden. Dit wordt geïllustreerd met enkele voorbeelden:



Figuur 27: RC5X voorbeelden pulstrein

Samengevat:

Tabel 5: samenvatting RC5X tijden

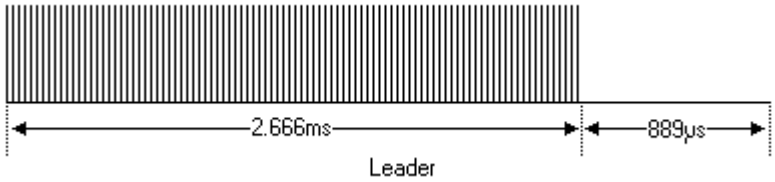
RC5X			
Carrier frequentie	36kHz	**- 20% marge**	**+ 20% marge**
Carrier - tijd hoog	9 µs	20% marge van één periode: 178 µs	
Carrier - tijd laag	18 µs		
Signaal - tijd één periode hoog/laag	889 µs	711 µs	1067 µs
Signaal - tijd twee periodes hoog/laag	1778 µs	1600µs	1956µs
Signaal - tijd vier periodes laag	3556 µs	3378 µs	3734 µs
Signaal - tijd vijf periodes laag	4445 µs	4267 µs	4623 µs
Signaal - tijd zes periodes laag	5334 µs	5156 µs	5512 µs
Signaal - totaal (20 bits + 2x laag)	39116 µs	31293 µs	46939 µs

Een RC5X signaal kan herkend worden aan de hand van volgende eigenschappen:

- De totale tijd moet tussen bovenstaande parameters liggen.
- De eerste gemeten tijd moet altijd één of twee periodes lang zijn.
- De langst gemeten tijd ligt tussen de minimum waarde van “tijd vier periodes laag” en “tijd zes periodes laag” en de tweede langst gemeten tijd moet één of twee periodes lang zijn.

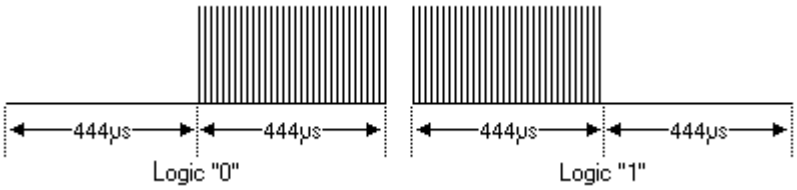
RC6 protocol

Het RC6 protocol is de opvolger van het RC5 en RC5X protocol. De werking is gelijkaardig maar de modulatietechniek is hier bijgewerkt. De "leader puls" bestaat uit een signaal dat 2666µs hoog is gevolgd door 889µs laag.



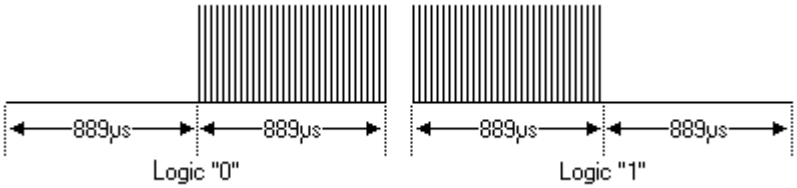
Figuur 28: RC6 leader modulatie [31]

De "normale" bits hebben zijn net als bij RC5 en RC5X bifase gecodeerd. Echter met een andere timing:



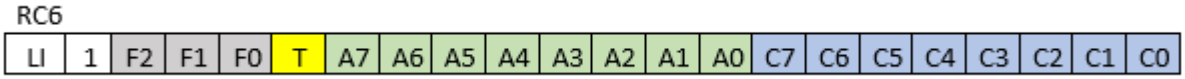
Figuur 29: RC6 normale bits modulatie [31]

Voor de toggle bit wordt het dubbele van deze timing gebruikt:



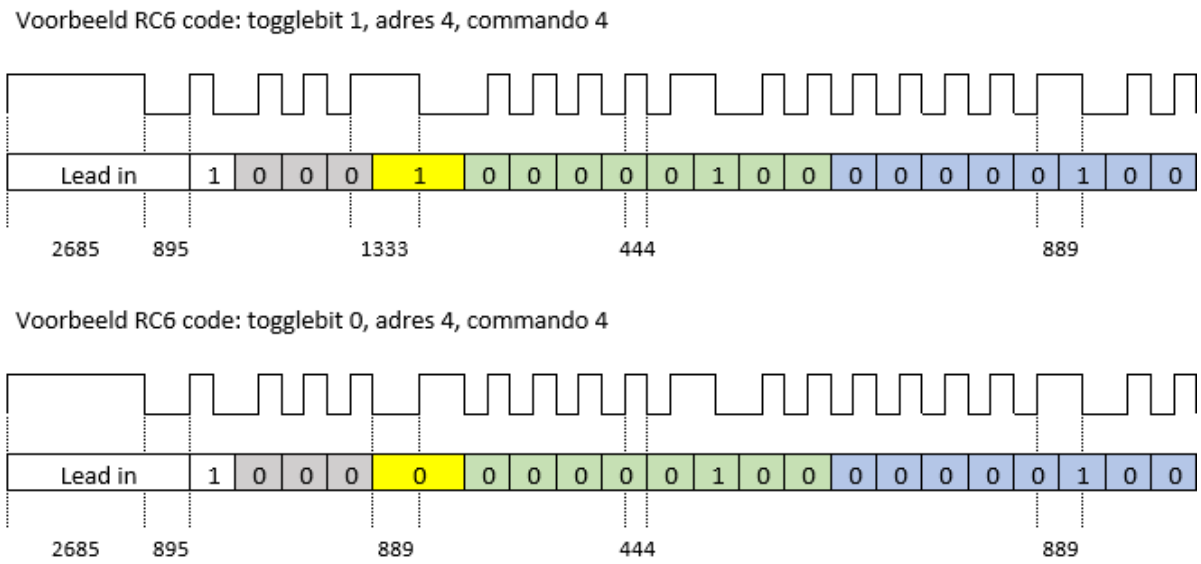
Figuur 30: RC6 toggle bit modulatie [31]

Tevens is ook de grootte van het adres en commando aangepast. Een RC6 signaal kan als volgt worden voorgesteld;



Figuur 31: RC6 code

Eerst wordt de lead in burst gestuurd. Hiermee wordt de ontvanger correct afgesteld. Vervolgens volgt een start bit (die altijd hoog is) en drie field bits (die meestal laag zijn). Dan worden acht adres bits en acht commando bits verzonden. Door het gebruik van een andere timing voor de toggle bit kunnen er in dit signaal ook verschillende tijden gemeten worden. Afhankelijk van de bit voor en na de togglebit. Hiervan zijn twee voorbeelden gegeven in onderstaande figuur.



Figuur 32: RC6 voorbeelden pulstrein

Samengevat:

Tabel 6: samenvatting RC6 tijden

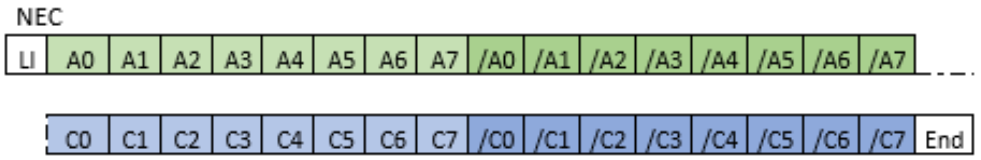
RC6			
Carrier frequentie	36kHz	**- 20% marge**	**+ 20% marge**
Carrier - tijd hoog	9 µs	20% marge van één periode: 89 µs	
Carrier - tijd laag	18 µs		
Signaal - tijd één periode hoog/laag	444 µs	355 µs	533 µs
Signaal - tijd twee periodes hoog/laag	889 µs	800 µs	978 µs
Signaal - tijd drie periodes hoog/laag	1333 µs	1244 µs	1422 µs
Signaal - lead in hoog	2685 µs	2148 µs	3222 µs
Signaal - lead in laag = 2 periodes	889 µs	800 µs	978 µs
Signaal - totaal (lead in + toggle + 20 bits)	23138 µs	18510 µs	27766 µs

Een RC6 signaal kan herkend worden aan de hand van volgende eigenschappen:

- De totale tijd moet tussen bovenstaande parameters liggen.
- De eerste en de tweede puls moeten voldoen aan de opgegeven waarde.

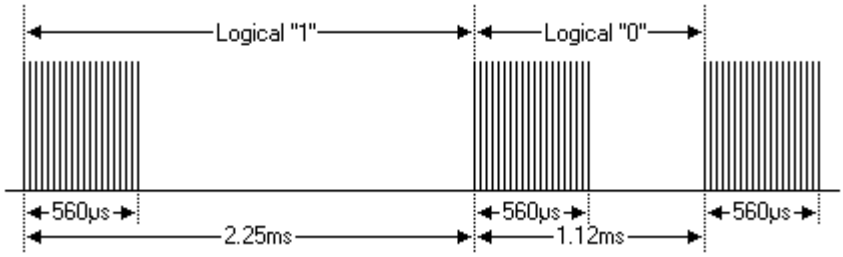
NEC protocol

Dit protocol werd ontwikkeld door de gelijknamige fabrikant 'NEC'. Het is een veel gebruikt protocol voor aansturing van onder andere LG en Pioneer apparaten. Dit protocol stuurt net zoals RC6 eerst een "lead in burst" van 9000µs hoog en 4500µs laag. Vervolgens zal het 8 bit adres, en zijn geïnverteerde waarde verzonden worden. Daarna volgt het 8 bit commando en zijn geïnverteerde waarde. Tenslotte volgt nog een "end burst".



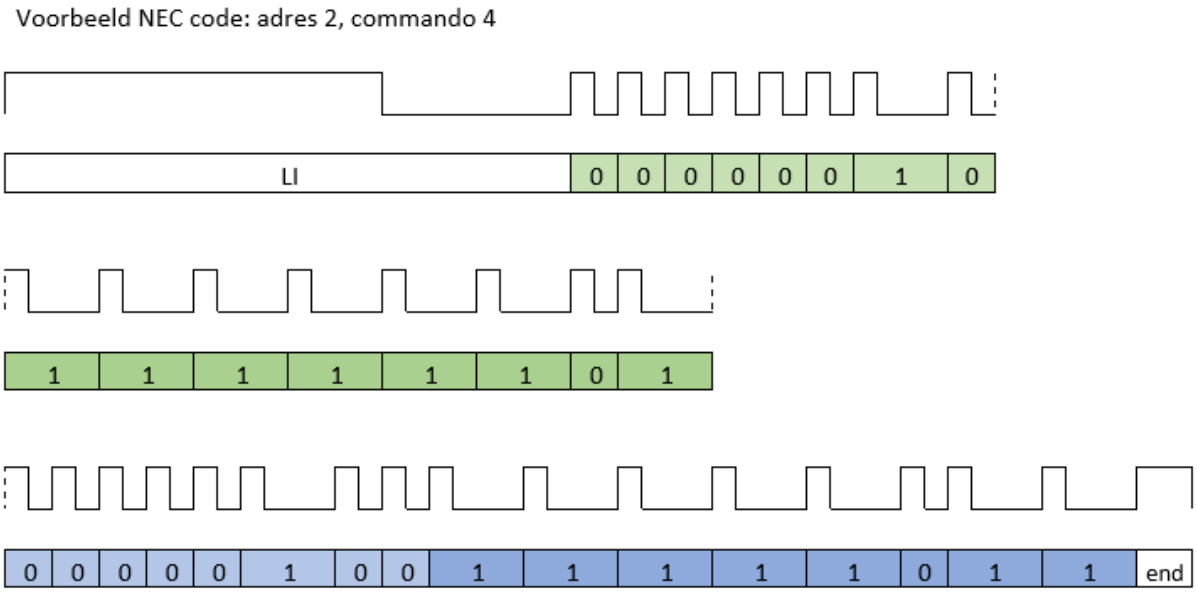
Figuur 33: NEC code

Ook de modulatie techniek is verschillend van de voorgaande besproken technieken. Een digitale ‘1’ zal voorgesteld worden door 560µs hoog gevolgd door 1690µs laag. Een digitale ‘0’ zal voorgesteld worden door 560µs hoog gevolgd door 560µs laag. De “end burst” is een puls van 560µs hoog.



Figuur 34: NEC modulatie [31]

In vergelijking met de RC protocollen is er geen togglebit in dit protocol. In het NEC protocol zal iedere code steeds hetzelfde verzonden worden. Wanneer een gebruiker een knop van een afstandsbediening ingedrukt houdt zal de code niet herhaald worden, maar een specifieke “repeat code” zal verzonden worden. De repeat code bestaat uit 9000µs hoog gevolgd door 2250µs laag en de “end burst” van 560µs hoog.



Figuur 35: NEC voorbeeld pulstrein

Samengevat:

Tabel 7: samenvatting NEC tijden

NEC			
Carrier frequentie	38kHz	**- 20% marge**	**+ 20% marge**
Carrier - tijd hoog	9 µs	20% marge van één periode: 112 µs	
Carrier - tijd laag	18 µs		
Signaal - tijd één periode hoog/laag	560 µs	448 µs	672 µs
Signaal - tijd 3 periodes laag	1690µs	1578 µs	1802 µs
Signaal - lead in hoog	9000 µs	7200 µs	10800 µs

Signaal - lead in laag	4500 µs	3600 µs	5400 µs
Signaal – rpt laag	2250 µs	1800 µs	2700 µs
Signaal - totaal (lead in + 32 bit + end)	67980 µs	54384 µs	81576 µs

Een NEC signaal kan herkend worden aan de hand van volgende eigenschappen:
- De totale tijd moet tussen de bovenstaande parameters liggen.
- De eerste en tweede puls moeten voldoen aan de opgegeven waarde.

Pronto protocol

Het pronto infrarood protocol, ook wel "ProntoEdit HEX formaat" genoemd, is ontwikkeld door Philips. Zij hadden een reeks "pronto" producten op de markt zoals universele afstandsbedieningen, en seriële verlengers. Typerend aan deze producten is dat ze gebruik maakten van het Pronto infrarood protocol. In 2010 heeft Philips de productie van de Pronto productserie stopgezet.

De Pronto toestellen konden infrarood signalen van eender welk protocol ontvangen en opslaan. Het ging de carrierfrequentie detecteren en de hoog- laagtijden meten zoals bij voorgaande protocollen in deze thesis besproken is. De hoog- laagtijden werden opgeslagen in hexadecimaal formaat en zo ontstond het "ProntoEdit HEX formaat". Met de carrierfrequentie en de hoog- laagtijden is het zeer eenvoudig om een signaal te reproduceren. Door deze eenvoud zijn veel databases ontstaan die pronto codes voor verschillende apparatuur bevatten. Veel fabrikanten zoals bijvoorbeeld LG geven hun infrarood codes vrij in het gebruikte protocol (NEC) maar ook in pronto formaat. Hierdoor blijft het pronto formaat, ook na het stopzetten van de Pronto producten door Philips, een veel gebruikt protocol.

Een "pronto code" is dus niet meer dan een exacte omschrijving van de timing van het signaal. Om het formaat uit te leggen werken we best aan de hand van een voorbeeld. Onderstaand is een NEC code weergegeven in pronto formaat:

0000 006C 0022 0002 015B 00AD 0016 0041 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0016 0016 0016 0016 0016 0016 0041 0016 0016 0016 0041 0016 0016 0016 0041 0016 0041 0016 0041 0016 0041 0016 06FB 015B 0057 0016 0E6C

We kunnen dit nu ontleden als volgt:

0000: Geeft aan dat het een unieke code is (werd gebruikt voor het inlezen met een pronto apparaat).

006C: Geeft informatie over de carrier frequentie 0x006C → 108 decimaal.
De formule voor de berekening van de frequentie is als volgt:

$$\frac{1000000}{N * 0.241246} = \frac{1000000}{108 * 0.241246} = 38.33kHz$$

We kunnen dus berekenen dat de periode T = 1/f = 1/38333Hz = 26.087µs. Dit komt ongeveer overeen met de 27µs periode tijd die eerder in deze thesis besproken is.

0022: Dit is de “one sequence” die het aantal hoog- laag paren voor het te zenden commando bevat.

0002: Dit is de “repeat sequence” dit zijn de codes die herhaaldelijk zullen verzonden worden bij het ingedrukt houden van een knop. Wanneer we het NEC protocol bekijken zien we dat de repeat code inderdaad zeer kort is.

Vanaf het vijfde getal vinden we de hoog- laag tijden terug. Dit zijn de tijden waarmee de infraroodlijn effectief laag en hoog gemaakt worden. Ieder paar is de hexadecimale voorstelling van een hoog- laagtijd. We dienen deze waardes enkel nog te vermenigvuldigen met de periodetijd (26.087µs)

Vb.:

015B 00AD	decimaal wordt dit: 347, 173.	x 26.087µs = 9022µs hoog, 4498µs laag
0016 0041	decimaal wordt dit: 22, 65	x 26.087µs = 572µs hoog, 1690µs laag

…

Op deze manier kunnen we het volledige signaal reconstrueren.

5.2.2 RS232 communicatie

De twee RS232 poorten maken het mogelijk om via een serieel protocol te communiceren met andere apparaten. Bijvoorbeeld Marantz versterkers en sommige LG tv's kunnen via seriële commando's aangestuurd worden.

Wanneer er data ontvangen wordt op de SMIC via RS232, zal er beslist moeten worden wanneer de data volledig ontvangen is. Doordat we protocol onafhankelijk werken kan er niet naar een specifiek "end of transmission"-karakter gezocht worden. Daarom wordt er softwarematig een timer gestart iedere keer er een karakter ontvangen wordt. Wanneer er na een bepaalde tijd geen karakter meer ontvangen wordt, weten we dat het bericht volledig ontvangen is. Hoe lang deze tijd is, is afhankelijk van het formaat en de gebruikte baudrate.

Een data frame bestaat altijd uit:

1 startbit – 5 tot 9 databits – een pariteitbit – 1 of 2 stopbits.

Eén van de meest gebruikte configuraties is "8N1", waarbij de '8' staat voor 8 databits. De 'N' voor none / geen pariteitbit, en de '1' voor één stopbit. Andere configuraties zijn ook mogelijk. De snelheid waarmee verzonden wordt, wordt de baudrate genoemd. Baudrate wordt uitgedrukt in "bits per seconde" (bps). Eén van de meest gebruikte baudrates is 9600 bps, andere standaarden zijn; 1200, 2400, 4800, 19200, 38400, 57600 en 115200 bps.

De transmissieduur van één karakter voor een configuratie van 9600 bps, 8N1 kan nu als volgt berekend worden:

Eén karakter is 8 bit groot. Per pakket wordt er een extra startbit, pariteitsbit en stopbit toegevoegd. In totaal 11 bits per karakter.

$$karakters\ per\ seconde = \frac{9600\ bits/s}{11\ bits} = 872$$

$$tijd\ per\ karakter = \frac{1\ 000\ 000\ \mu s}{872} = 1146\ \mu s$$

Een meer universele formule is dus:

$$tijd\ per\ karakter = \frac{1\ 000\ 000\ \ \mu s}{\frac{baudrate}{11}}$$

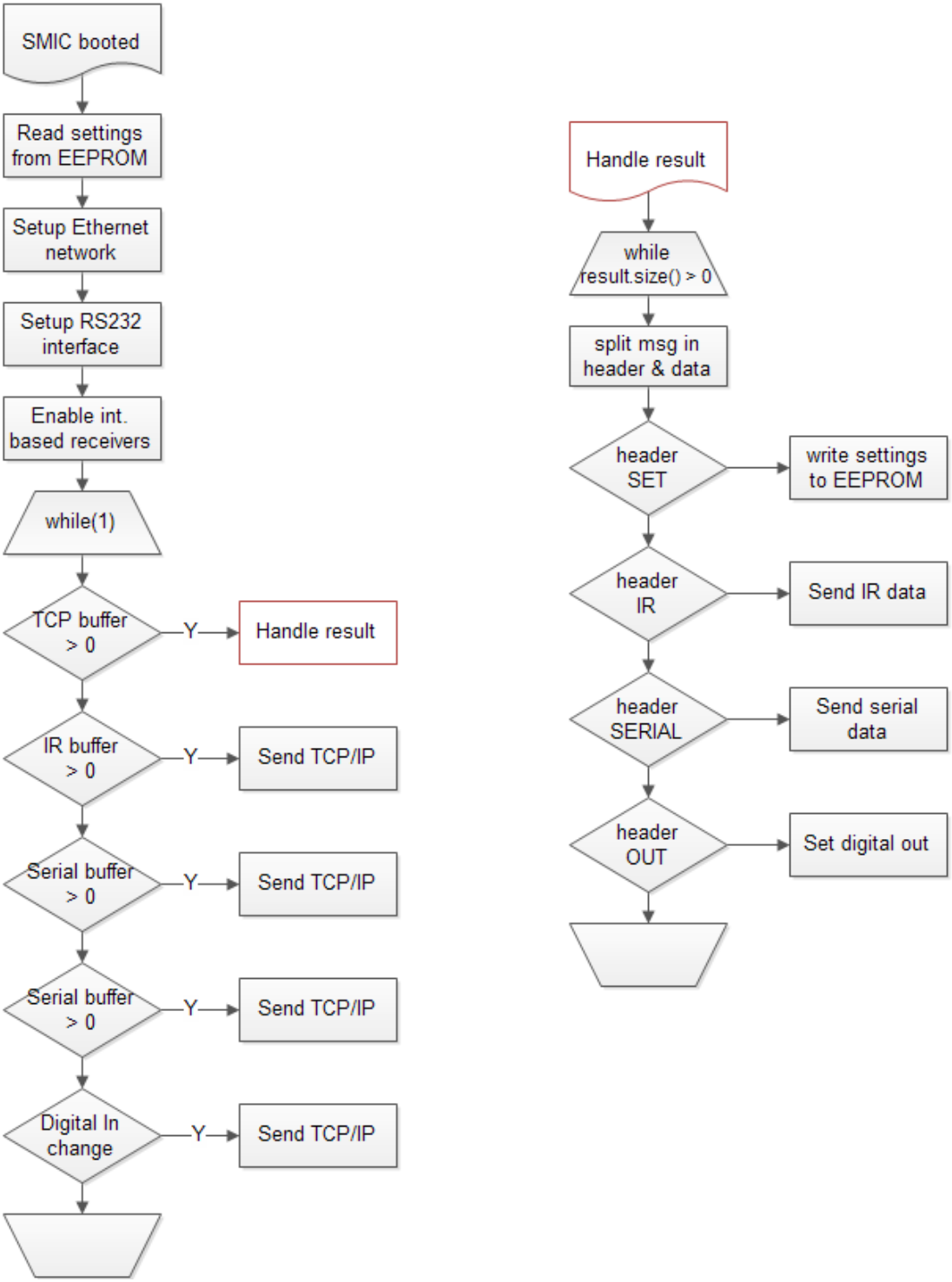
Om de ontvanger af te stellen wordt als vuistregel de tijd van 4 à 6 karakters genomen, om te bepalen wanneer een bericht ontvangen is. We kunnen deze dus instellen door de formule toe te passen:

$$Timout\ ontvangen\ compleet = \frac{1\ 000\ 000\ \ \mu s}{\frac{baudrate}{11}} * 6$$

5.3 Omschrijving van de software

5.3.1 Structuur

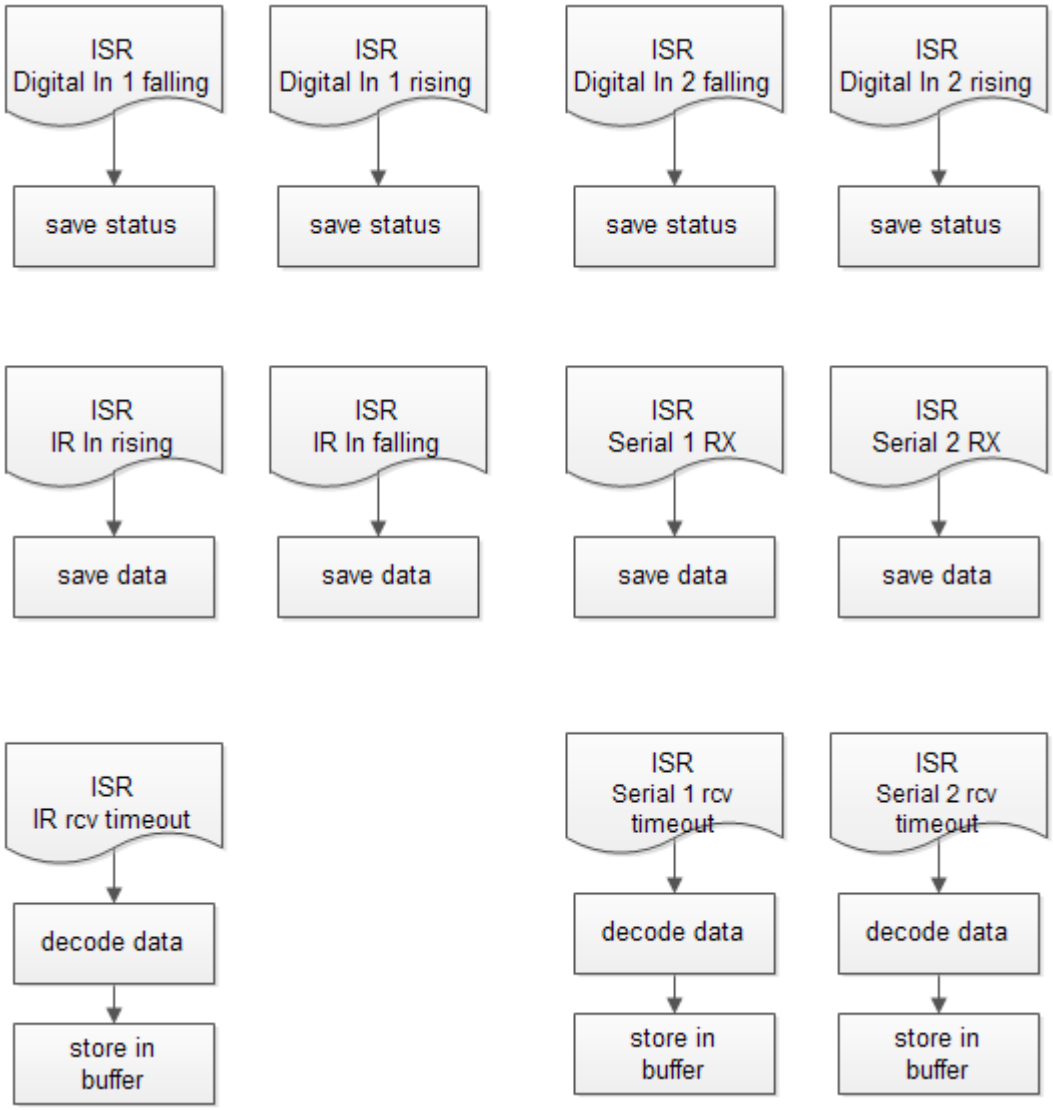
De werking van de code wordt verduidelijkt aan de hand van enkele flowcharts. Samen met de documentatie in de bijlage is het mogelijk om de opbouw van de code te volgen.



Figuur 36: Flowchart main loop

Ontvangen van data via infrarood, RS232 en digital In

Alle veranderingen aan ingangen worden via interrupt handlers opgevangen en de data wordt bijgehouden. Om te bepalen wanneer een bericht volledig ontvangen is, worden er "timeout" timers gestart. Wanneer deze aflopen zal de ontvangen data gedecodeerd worden en zullen de buffers gevuld worden. In de hoofd-programma-lus zal continu gecontroleerd worden of de buffers volledige berichten bevatten. Indien dat zo is zal het bericht via TCP/IP naar de server gestuurd worden.



Figuur 37: Interrupt service routines

Ontvangen van data via TCP/IP

De TCP/IP buffer wordt gevuld met data die van de server afkomstig is via het netwerk. Deze berichten worden afgehandeld door de SMIC. Afhankelijk van de header wordt er bepaald of het gaat om nieuwe instellingen, infrarood data, RS232 data of een wijziging van de digitale uitgangen.

Door op deze manier met buffers te werken zal data real time ontvangen kunnen worden. De hoofdlus fungeert als dispatcher die de verschillende buffers zal controleren en afhandelen indien nodig.

5.3.2 Toekenning van het MAC-adres

Omdat er via TCP/IP gecommuniceerd wordt tussen de server en de SMIC dienen deze voorzien te zijn van een MAC-adres. Dit is een uniek identificatienummer dat een apparaat moet hebben in een ethernetnetwerk. Het MAC-adres wordt ook wel hardware adres of fysiek adres genoemd.

De mbed bibliotheek zal het MAC-adres halen uit de interfacechip die op de prototypebordjes zorgt voor de communicatie tussen PC en de processor. Op de SMIC is deze interfacechip echter niet aanwezig, dus moet het MAC-adres op een andere manier gegenereerd worden. Dit kan op basis van het unieke serienummer van de processor.

Met behulp van de IAP-bibliotheek kunnen we het serienummer van de processor lezen. IAP staat voor "In-Application Programming". Hiermee kan het intern flash geheugen van de processor gelezen en geschreven worden.

Het serie nummer bestaat uit vier, 32 bit getallen. Het eerste getal (index [0]) is hetzelfde als het processor ID en dus niet uniek. Een MAC-adres bestaat uit zes, acht bit, getallen. Om een uniek MAC-adres voor alle SMIC apparaten te genereren is er gekozen om in de eerste drie bytes "B&B" te schrijven. De andere drie bytes worden opgevuld met de minst significante bit van de drie unieke getallen van het serienummer.

```
extern "C" void mbed_mac_address(char * mac)
{
    int *serial_number;
    serial_number = iap.read_serial();
    mac[0] = 'B';
    mac[1] = '&';
    mac[2] = 'B';
    mac[3] = 0x000000ff & serial_number[1];
    mac[4] = 0x000000ff & serial_number[2];
    mac[5] = 0x000000ff & serial_number[3];
}
```

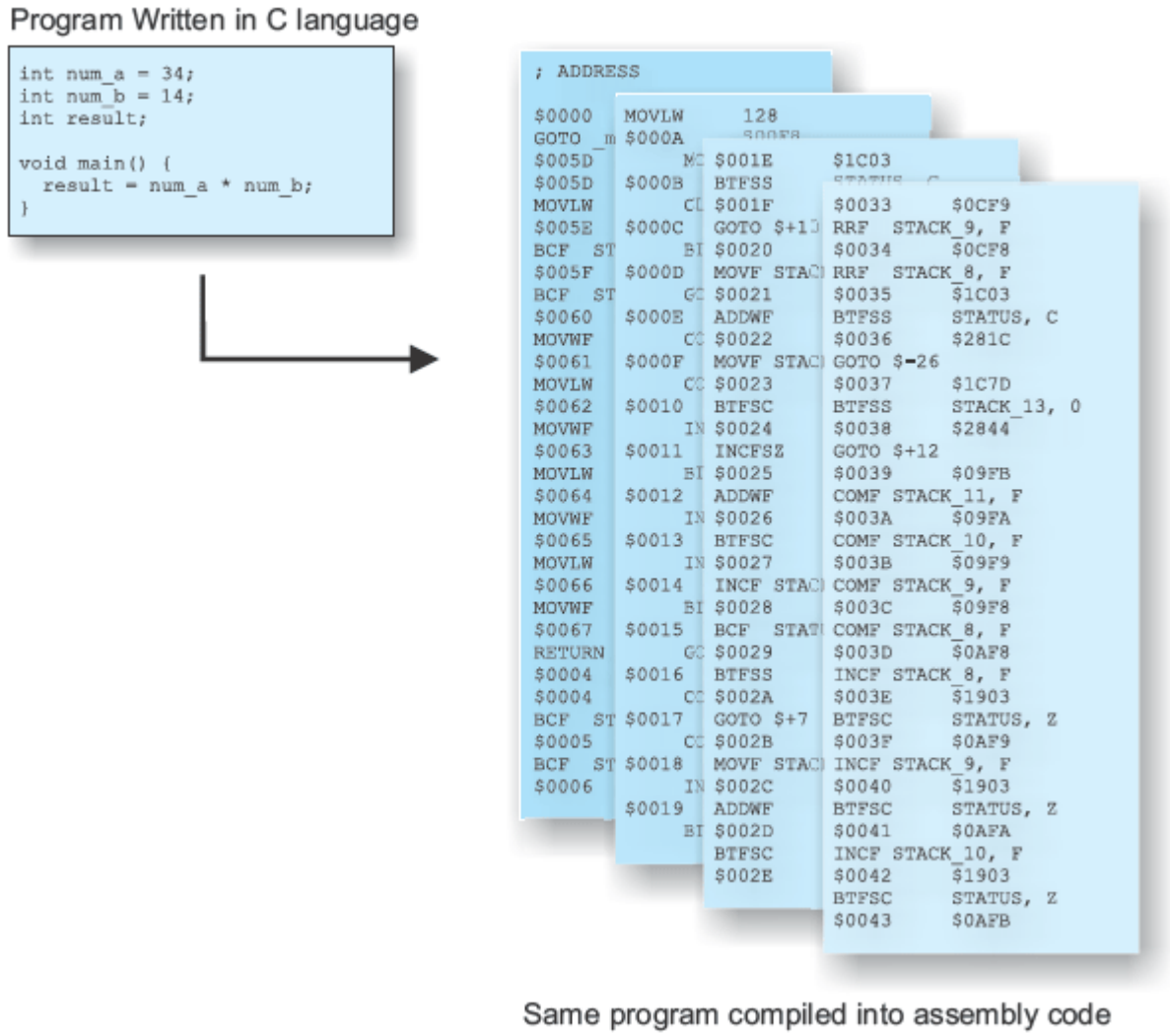
Figuur 38: functie mbed_mac_address

Doordat de functie mbed_mac_address in de mbed bibliotheek 'zwak' gedefinieerd is, kan deze functie zelf overschreven worden. De linker zal dan de zelfgeschreven functie gebruiken in plaats van de standaard functie.

5.3.3 Valkuilen bij softwareontwikkeling voor embedded systemen

Er zijn verschillende talen om microcontrollers te programmeren. Assembler is een programmeertaal op het laagste niveau. Alle hardware wordt bediend via Assembly codes. Een hogere programmeertaal zoals C moet worden gecompileerd. Hierdoor worden stukken C-code omgevormd naar Assembly instructies. De Assembly instructies worden op hun beurt geassembleerd tot machinecode [32].

C is al geruime tijd de standaardtaal voor de programmatie van microcontrollers. De reden waarom C "gebruiksvriendelijker" is dan Assembly is weergegeven op onderstaande afbeelding.



Figuur 39: Assembly vs C code [33]

Met de komst van 32 bit processoren is het ook mogelijk om C++ te compileren voor embedded systemen. Met het gebruik van C++ is het mogelijk om veel compactere code te schrijven. Alle functionaliteiten van C zitten ook in C++, daarbovenop kunnen de voordelen van C++ gebruikt worden, waaronder het gebruik van object georiënteerd programmeren.

Door het beperkt geheugen van een microcontroller kan C++ echter niet op dezelfde manier gebruikt worden als op een computer. Er is zelfs een dialect "Embedded C++" of ook wel "C+" gedefinieerd. Dit is een afgeslankte versie van C++.
De kunst bestaat erin om C++ op een juiste manier te gebruiken. In dit hoofdstuk worden de valkuilen van C++ voor embedded software en de aandachtspunten bij het schrijven van embedded software toegelicht.

Beperkte heap grootte

Embedded systemen hebben een zeer beperkte heap grootte. Hierdoor kan er best geen dynamische geheugenallocatie gebeuren. Vermijd dus ook het gebruik van de operatoren "new" en "delete".

Het is ook best om de klasse "string" te vermijden in software voor embedded systemen. Deze klasse maakt onderliggend gebruik van de operatoren "new" en "delete". Bijvoorbeeld de substring functie zal het resultaat van de functie kopiëren naar een nieuwe string.

Een veilige manier om met objecten te werken is door ze globaal te declareren en alleen geheugenallocatie te doen bij de opstart van de microcontroller. Er worden dus ook geen destructors opgeroepen voor deze objecten. Ze worden aangemaakt bij startup en blijven in gebruik totdat het systeem uitgeschakeld wordt (device power down).

Het gebruik van deze techniek is snel merkbaar bij het standaard "blinky led" voorbeeld van mbedOS.

```
1  #include "mbed.h"
2
3  DigitalOut myled(LED1);
4
5  int main() {
6      while(1) {
7          myled = 1;
8          wait(0.2);
9          myled = 0;
10         wait(0.2);
11     }
12 }
```

Figuur 40: Blinky led voorbeeld

"myled" is een object van de klasse "DigitalOut" en wordt globaal gedeclareerd. De destructor zal nooit gestart worden voor dit object omdat de functie "main()" nooit stopt.

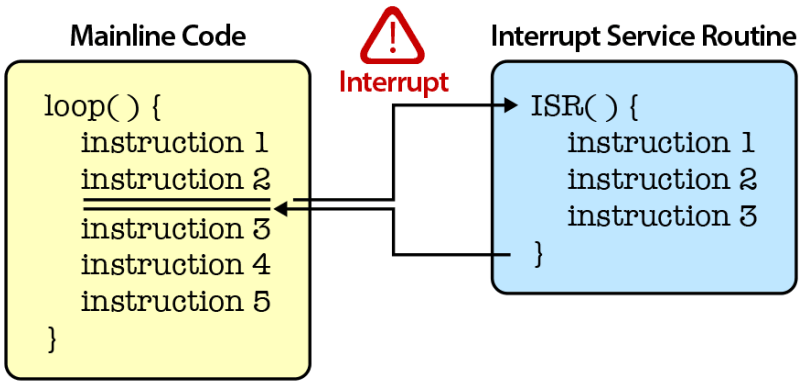
Behalve door globale declaratie kunnen objecten ook als lokale variabele gedeclareerd worden. In dit geval wordt er geheugen op de stack gealloceerd. Hierdoor kunnen problemen zoals "stack overflow" ontstaan.

Op deze manier kan object georiënteerd programmeren toegepast worden voor embedded systemen zonder problemen te krijgen met heapgeheugen. Door globale allocatie weet de linker al hoeveel geheugen er in gebruik is en kan deze, bij overflow, een error geven.

Er kan dus gesteld worden dat het gebruik van dynamisch geheugen mag, zolang de heap niet volloopt. Maar bij situaties waarbij het dynamisch geheugen afhangt van externe factoren zoals een ethernetverbinding is het beter om zo weinig mogelijk dynamisch geheugen te alloceren. Want wanneer er dynamisch gereserveerde buffers gebruikt worden voor de opslag van de ontvangen berichten kan er een probleem ontstaan op momenten wanneer een grote vloed van berichten ontvangen wordt.

Blocking code in interrupt routines

Een andere valkuil is het afhandelen van Interrupt Service Routines (ISR) ook wel "interrupt handler" genoemd. Dit is zowel in C als in C++ een belangrijk aandachtspunt. Wanneer er (meestal door hardware) een interrupt gebeurt, zal de programmalus op dat moment onderbroken worden en de ISR uitgevoerd worden, waarna de programmalus terug hervat wordt.



Figuur 41: Interrupt Service Routine [34]

De onderbreking van de programmalus, het uitvoeren van de ISR, moet zo kort mogelijk zijn. Beperk daarom de code in een ISR tot het wijzigen van variabelen of het uitvoeren van berekeningen. Vermijd zeker:

- wait functies;
- oneindige while lussen;
- blocking calls;
- gebruik van printf;
- gebruik van malloc en new.

Indien er in een ISR toch code uitgevoerd wordt die een lange tijd in beslag neemt zou het kunnen dat er een andere ISR niet kan uitgevoerd worden. Tijdens het uitvoeren van een ISR worden er namelijk geen andere interrupt routines gedetecteerd.

Een andere situatie waarbij de code in een ISR beperkt dient te worden, is het uitlezen van hardware registers. Neem bijvoorbeeld een seriële interface. Wanneer er data ontvangen wordt, zal deze in een register geschreven worden en zal de RX interruptvlag hoog gezet worden. De CPU ziet dit en kan een ISR uitvoeren. Indien het te lang zou duren voor de ISR uitgevoerd wordt, kan er al nieuwe data ontvangen zijn en is de oude data overschreven in het register.

5.3.4 Programmeerstijl volgens C++ standaarden

In deze sectie worden enkele tips beschreven voor het optimaal gebruik van C++ voor embedded systemen.

Gebruik van "const" vs "define".

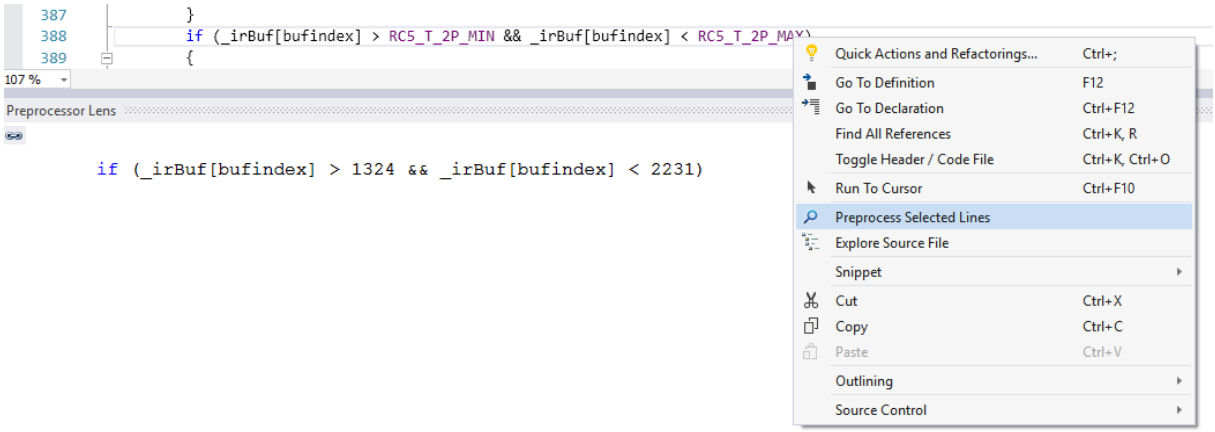
In de taal C worden vaak macro's gebruikt. Met behulp van de notatie "#define" worden bepaalde waardes gedefinieerd. Voor het compileren zal de preprocessor al deze waardes vervangen. Het gebruik van define maakt de code dus vooral leesbaarder.

Bijvoorbeeld:

```
#define AGE 10        // het woord "AGE" zal voor het compileren vervangen worden door
                      het getal 10.
```

Doordat de preprocessor overal het woord "AGE" zal gaan vervangen door het getal 10 zou dit ongewenste effecten kunnen opleveren en verkeerde zaken vervangen worden.

De gebruikte IDE heeft hiervoor een ingebouwde functie met "Preprocess Selected Line" wordt getoond hoe de preprocessor de code vertaalt.



Figuur 42: Preprocessor voorbeeld

Een beter alternatief is het gebruik van "const" in C++. Deze variabelen zijn "type safe" en zullen in het ROM geheugen bewaard worden.

Bijvoorbeeld:

```
const int AGE = 10    // AGE stelt nu integerwaarde 10 voor.
```

Het adres van een "const" variabele kan ook gebruikt worden als parameter. In sommige gevallen is het nog beter om "constexpr" te gebruiken. Deze waarden zullen sowieso niet in het RAM geheugen terecht komen. Van "constexpr" variablen kan geen adres genomen worden maar ze kunnen wel gebruikt worden om bijvoorbeeld de grootte van een array te declareren. Op deze manier is het nooit meer nodig om "define" te gebruiken en is het

programma 'type safe'. Over het algemeen is het dus aangeraden om "const" of "constexpr" te gebruiken.

Gebruik van het referentietype i.p.v. pointers.

Sinds de C++11/C++14 standaard wordt er voorkeur gegeven om geen ruwe pointers meer mee te geven als parameter aan een functie maar om de "call by reference" methode te gebruiken. Hierbij wordt een referentie naar een bepaalde variabele of object meegegeven. Doordat er geen kopie wordt gemaakt, wordt er dus ook geen extra geheugenruimte gereserveerd. De referentie is louter een andere benaming voor dezelfde plaats in het geheugen. Hierdoor is het niet nodig dat de gebruiker van de functie de manier van werken met pointers kent. Intern in de functies wordt er wel met pointers gewerkt. Pointers worden dus enkel nog gebruikt bij dynamisch gereserveerde objecten, voor de overige objecten kan het referentietype gebruikt worden. In die nieuwe standaarden worden de unieke en shared pointers (unique_ptr en shared_ptr) ondersteund, hierdoor daalt de kans op geheugenlekken drastisch.

5.4 Software testen

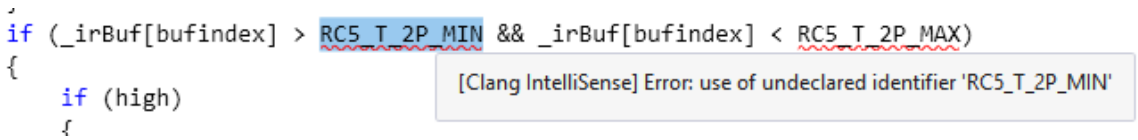
Een zeer belangrijk onderdeel van softwareontwikkeling is het testen van de software. Er zijn verschillende soorten en vormen van testen die tijdens de ontwikkelingsfase zouden moeten gebeuren. Vooral bij embedded systemen is het zeer belangrijk om er 100% zeker van te zijn dat het toestel nooit zal vastlopen of crashen. Het is dus zeker belangrijk om dit op zoveel mogelijk manieren te testen en te voorkomen.

De testmethodes die bij het ontwikkelen van de software voor de SMIC toegepast zijn worden in dit hoofdstuk besproken.

5.4.1 Statisch testen

Een eerste vorm gebeurt door de IDE, dit is "statisch testen". Statisch testen duidt op alle testen die gebeuren alvorens de code uitgevoerd wordt.

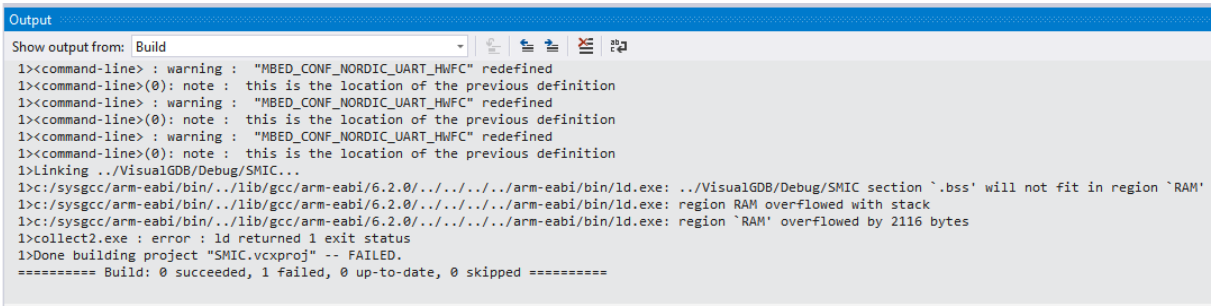
Allereerst is er de IDE die de code zal checken op syntax fouten en deze zal markeren. De gebruikte tool "VisualGDB" maakt gebruik van "Clang intellisense", deze ondersteunt de C+11 standaard en is direct gelinkt aan de Clang compiler.



Figuur 43: Syntax controle voorbeeld

Daarna volgt het compileren. De compiler vormt de C en C++ code om naar Assembler code, als hierin fouten optreden zal de compiler deze tonen. VisualGDB zorgt ervoor dat de file waarin gewerkt wordt automatisch om een bepaald interval gecompileerd wordt. Zo worden fouten snel gedetecteerd, en niet pas wanneer de programmeur de compiler start.

Na het compileren zal de linker de verschillende assembler modules samenvoegen tot een uitvoerbaar programma. Het checkt hierbij of het programma past binnen het beschikbare geheugen. Indien niet zal deze ook een error boodschap tonen en wordt het programma niet uitgevoerd.



Figuur 44: Voorbeeld linker error

5.4.2 Dynamisch testen

Wanneer het programma in werking is kunnen dynamische testen uitgevoerd worden. Het doel van dynamisch testen is om de grenzen van het toestel op te zoeken, hiervoor veiligheden inbouwen en er uiteraard voor zorgen dat het nooit zal crashen terwijl het programma loopt.

Deze testen worden onderverdeeld in:

- impliciet vs expliciet testen
- blackbox vs whitebox testen

Bij het impliciet testen wordt er getest wat de werking van een bepaalde functie of stuk code is. Expliciet testen checkt of de software voldoet aan de vooropgestelde eisen.

Blackbox testen zijn testen waarbij er niets van de werking gekend is door de tester. Meestal worden er eerst whitebox testen uitgevoerd. Deze worden door de ontwikkelaar zelf gedaan om tussendoor te checken of de functies al dan niet voldoen aan de verwachtingen.

Tijdens de softwareontwikkeling voor de SMIC zijn steeds whitebox testen uitgevoerd om de functionaliteit van klasses en hun methodes te testen. Aan de hand van deze testen is getracht om alle mogelijke fouten op te vangen. Zo is er bijvoorbeeld getest wat de code doet met berichten die via TCP/IP ontvangen worden en niet het gewenste formaat hebben.

Vervolgens zijn er ook “failsafe” testen uitgevoerd om prestaties van het systeem in uitzonderlijke omstandigheden te testen. Bijvoorbeeld: “Wat gebeurt er wanneer de netwerkverbinding verbroken wordt?”, “Wat als er met meerdere afstandsbedieningen gelijk data verzonden wordt naar de SMIC?”. Aan de hand van de resultaten is de software herschreven en veiliger gemaakt.

In de laatste ontwikkelingsfase zijn er testcases ontworpen om de prestaties van het toestel te testen. Zo is er een opstelling gemaakt die zorgt voor een continue aansturing van het toestel via alle mogelijke kanalen.

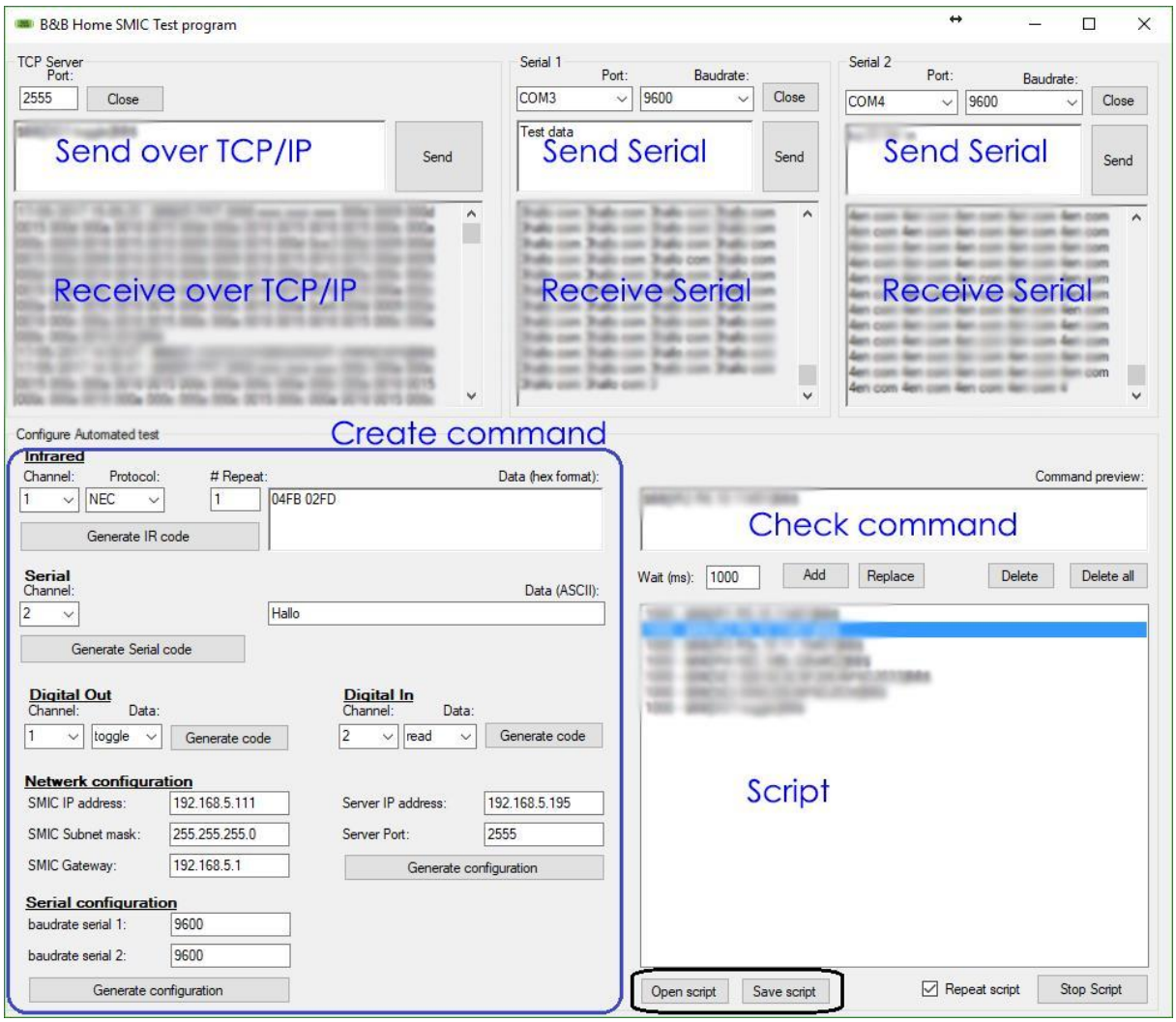
Enkele laatste testen, die nog uitgevoerd moeten worden, zijn de blackbox testen. Hierbij zullen zowel gebruikers als installateurs het toestel testen. De gebruikers zullen het toestel bedienen en er zal gekeken worden of het toestel voldoet aan de verwachtingen. De installateurs zullen ook testen hoe het toestel geconfigureerd moet worden en of er hierbij fouten zouden kunnen optreden.

5.4.3 Stresstest en durability test

Wanneer de software klaar is, zijn nog stresstesten en een durability test uitgevoerd.
De stresstest zal nagaan hoe het toestel reageert bij abnormaal gebruik. Bijvoorbeeld wanneer er zeer snel, abnormaal veel data verwerkt moet worden.
De durability test zal uitwijzen hoe het toestel na verloop van tijd reageert. Hiermee wordt ook getest of er geen geheugenlekken zijn.

Om deze testen uit te voeren is software ontwikkeld die de werking van de B&B Home server nabootst. Er zijn ook extra data loggers toegevoegd die de ontvangen data via de TCP/IP verbinding en de seriële verbindingen monitort. Zo kunnen alle kanalen getest worden. Daarnaast zijn er velden voorzien waarmee de configuratie voor de SMIC opgebouwd kan worden.

De software is geschreven in C#.NET.



Figuur 45: Grafische test interface

Via de grafische interface is het mogelijk om verschillende commando's op te bouwen en met de commando's een script te maken. Zulke scripts kunnen opgeslagen worden en later terug geopend. Zo kunnen bepaalde testscenario's opgebouwd worden. Met de checkbox "repeat script" rechts onderaan, kunnen scripts steeds weer herhaald worden. Op deze manier is het eenvoudig om voor lange tijd repetitieve testen uit te voeren. Dat is handig voor de stress- en durability test.

5.4.4 Watchdog timer

Omdat er nooit voor de volle 100% getest kan worden op onregelmatigheden wordt bij embedded systemen vaak een watchdog timer toegevoegd. Dit is een veiligheid die ervoor zal zorgen dat de processor gereset wordt, en het programma dus opnieuw gestart wordt.

De watchdog detecteert wanneer er een softwarefout is opgetreden waardoor het programma "hangt". De timer wordt gestart met een waarde die groter is dan de grootste delay, die in het slechtste geval voorkomt, in het programma. Een voorbeeld van het gebruik van een watchdog timer wordt gegeven in onderstaande code.

Wanneer de functie "wdt.kick()" niet binnen de 10 seconden wordt opgeroepen zal het systeem resetten. De main-lus zal bij een systeem opstart steeds nagaan of deze werd uitgelokt door de watchdog timer of door een handmatige heropstart van het systeem.

```
#include "mbed.h"

Watchdog wdt;

int main(){
    //check if the watchdog caused a reset
    if (wdt.causedReset())
    {
        printf("System booted, watchdog caused reset");
    }
    else
    {
        printf("System booted, normal");
    }
    //setup the watchdog timer
    wdt.kick(10.0)

    while(1){
        //do some stuff
        //...

        //end of main loop, so "kick" to reset watchdog timer avoid a reset
        wdt.kick();
    }
}
```

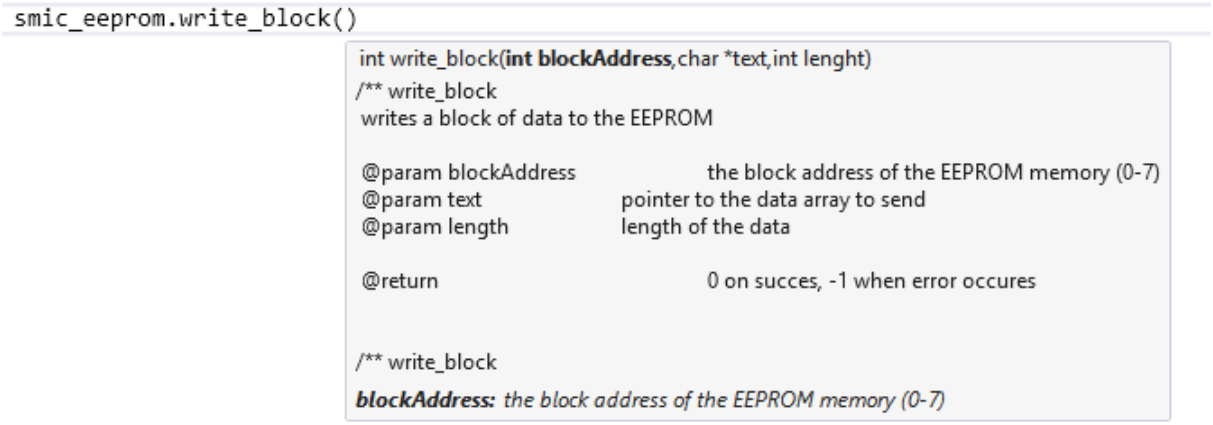
Figuur 46: Voorbeeld watchdog timer

5.5 Software documentatie

Een belangrijk aspect van softwareontwikkeling is het schrijven van de nodige documentatie. Vaak wordt hieraan door ontwikkelaars te weinig tijd besteed, of wordt het op de foute manier geschreven waardoor het zijn doel mist.

Goede documentatie zorgt echter voor een betere leesbaarheid, en maakt de code veel makkelijker onderhoudbaar. Vooral wanneer er met meerdere ontwikkelaars aan één project gewerkt wordt is commentaar essentieel.

De meeste moderne IDE's geven met behulp van intellisense ook hints over de parameters die een functie verwacht, indien de commentaar correct geschreven is. Onderstaand is hiervan een voorbeeld te zien.



Figuur 47: Voorbeeld intellisense met gebruik van correcte commentaar

Ook zijn er verschillende IDE's die automatisch commentaar kunnen genereren. Meestal is deze maar in beperkte mate correct, maar mits enkele aanvullingen is zo snelle en juiste commentaar geschreven.

Deze masterproef onderzoekt twee tools die mogelijk geschikt zijn.

5.5.1 GhostDoc

Dit is een Visual Studio extensie die automatisch XML documentatie kan genereren van methods en properties op basis van parameters, variabele namen en andere contextuele informatie [35]. Met een Pro licentie is het mogelijk om commentaar te genereren voor een volledige klasse. Dit bespaart een hele hoop tijd. GhostDoc ondersteunt C#, VisualBasic, C++ en JavaScript. De gratis (proef)versie van GhostDoc is echter niet bedoeld voor commerciële doeleinden.

Hoewel deze tool zeer volledig is en goed werkt, zijn de licentieprijzen te hoog voor een kleiner bedrijf zoals Bits & Bytes. Daarom is er gezocht naar een (gratis) alternatief.

5.5.2 Doxygen

Deze tool is één van de open-source standaarden voor het genereren van C++ documentatie. Andere ondersteunde programmeertalen zijn; C, C#, PHP, Java, Python, Fortran, en VHDL. Een groot voordeel in vergelijking met GhostDoc, is dat de tool IDE onafhankelijk werkt. De documentatie wordt rechtstreeks gegenereerd van de broncode.

Doxygen kan ook zodanig geconfigureerd worden dat de tool codestructuren van niet gedocumenteerde bronfiles genereert. Dit kan zeer handig zijn om snel een weg te vinden in grote code distributies.

De tool genereert documentatie zowel in HTML als Latex formaat. De tool wordt ook door de mbed community gebruikt voor het documenteren van het volledige mbed OS en andere bibliotheken.

Voor het documenteren van de SMIC software is dan ook voor Doxygen gekozen. De gegenereerde commentaar is als bijlage te vinden bij deze thesis.

6 Besluit

De vooropgestelde doelstellingen van deze masterthesis; het opzetten van een ontwikkelomgeving, en deze toepassen voor de softwareontwikkeling van de SMIC, zijn bereikt. De ontwikkelomgeving die opgebouwd is, is echter wel specifiek gekozen voor de firma Bits & Bytes. Er is rekening gehouden met hun eisen en budgetten. Voor een ander bedrijf of instelling zouden allicht andere keuzes gemaakt zijn.

Enkele facetten van de software zoals de opbouw van het protocol en het formaat voor het communiceren tussen de B&B Home domoticaserver en de SMIC zijn in deze thesis niet besproken. Hiervoor is een aparte handleiding geschreven die enkel voor Bits & Bytes bestemd is. Dit om het bedrijfsgeheim niet te schenden en de masterthesis toch openbaar te kunnen voorstellen en verdedigen.

De software voor de SMIC is volledig ontwikkeld en zal na een grondige testfase geïmplementeerd worden in het domoticasysteem. Er zijn enkele zaken in de code toegevoegd die de functionaliteit in van de SMIC in de toekomst kunnen uitbreiden. Zo is bijvoorbeeld het NEC protocol toegevoegd als extra infrarood protocol. Ook is er een functie ingebouwd die de “rauwe” infrarood data kan inlezen. Zo kan een signaal van een onbekend protocol toch door de server gedecodeerd worden. Ook zijn er nog enkele geheugenblokken in het EEPROM geheugen voorzien die eventueel kunnen dienen voor het loggen van gebeurtenissen.

De ontwikkelaars bij Bits & Bytes zullen de geïmplementeerde ontwikkeltools in de toekomst gebruiken voor de ontwikkeling van nieuwe producten. De Smart Media IP Controller zal een mooie aanvulling zijn voor het B&B Home domoticasysteem.

Literatuurlijst

[1] "Bits & Bytes nv." [Online]. Available: www.bitsbytes.be. [Accessed: 05-Oct-2016].

[2] R. Moons, "Ontwerp van BBHome smart media IP controller," UC Leuven-Limburg, 2015.

[3] E. Solutions, "ARM ® Cortex ® -A Portfolio ARMv8-A," pp. 1–6, 2016.

[4] "ARM processors for the widest range of devices, from sensors to servers." [Online]. Available: http://www.arm.com/products/processors. [Accessed: 28-Sep-2016].

[5] "ARM Processors Selection Guide ®," 2013.

[6] "mbed HDK." [Online]. Available: https://developer.mbed.org/handbook/mbed-HDK.

[7] "mbed compiler." [Online]. Available: https://developer.mbed.org/compiler.

[8] mbed, "mbed developer site." [Online]. Available: https://developer.mbed.org/. [Accessed: 09-Oct-2016].

[9] "STM32 32-bit ARM Cortex MCUs." [Online]. Available: http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus.html?querycriteria=productId=SC1169. [Accessed: 05-Oct-2016].

[10] "STM32 Community." [Online]. Available: http://www.openstm32.org/. [Accessed: 05-Oct-2016].

[11] "ST-LINK/V2." [Online]. Available: http://www.st.com/en/development-tools/st-link-v2.html?icmp=tt3469_gl_qron_apr2016. [Accessed: 05-Oct-2016].

[12] "In-System Debugging." [Online]. Available: http://www.atmel.com/Microsite/atmel-studio/debugging-simulation.aspx. [Accessed: 05-Oct-2016].

[13] "Entry-level debug through full-capability development." [Online]. Available: http://www.ti.com/lsds/ti/tools-software/debug.page. [Accessed: 05-Oct-2016].

[14] "LPCXpresso IDE v8.2.2." [Online]. Available: http://www.nxp.com/products/software-and-tools/software-development-tools/software-tools/lpc-microcontroller-utilities/lpcxpresso-ide-v8.2.2:LPCXPRESSO#. [Accessed: 12-Oct-2016].

[15] "Tasking." [Online]. Available: http://www.altium.com/tasking/overview. [Accessed: 05-Oct-2016].

[16] "Embedded Studio." [Online]. Available: https://www.segger.com/embedded-studio.html. [Accessed: 05-Oct-2016].

[17] "Debug Probes - J-Link and J-Trace." [Online]. Available: https://www.segger.com/jlink-debug-probes.html. [Accessed: 05-Sep-2016].

[18] "µVision IDE." [Online]. Available: http://www2.keil.com/mdk5/uvision/. [Accessed: 12-Oct-2016].

[19] "ULINK Debug and Trace Adapters." [Online]. Available: http://www2.keil.com/mdk5/ulink. [Accessed: 12-Oct-2016].

[20] "The best FREE C/C++ IDE for ARM development." [Online]. Available: http://atollic.com/. [Accessed: 12-Oct-2016].

[21] "IAR Embedded Workbench." [Online]. Available: https://www.iar.com/iar-embedded-workbench/#!?architecture=ARM. [Accessed: 12-Oct-2016].

[22] "Debugging and trace probes." [Online]. Available: https://www.iar.com/iar-embedded-workbench/add-ons-and-integrations/in-circuit-debugging-probes/. [Accessed: 12-Oct-2016].

[23] "CrossWorks." [Online]. Available: http://www.rowley.co.uk/index.htm. [Accessed: 12-Oct-2016].

[24] "emIDE." [Online]. Available: http://www.emide.org/index.html. [Accessed: 12-Oct-2016].

[25] "CooCox Free/open ARM Cortex-M Development Tool-chain." [Online]. Available: http://www.coocox.org/. [Accessed: 12-Oct-2016].
[26] "CoLinkEx." [Online]. Available: http://www.coocox.org/hardware/colinkex.php. [Accessed: 12-Oct-2016].
[27] "Developing Mbed firmware with Visual Studio." [Online]. Available: http://visualgdb.com/tutorials/arm/mbed/. [Accessed: 12-Oct-2016].
[28] "git -- local branching on the cheap." [Online]. Available: https://git-scm.com/book/nl/v1/Aan-de-slag-Het-wat-en-waarom-van-versiebeheer. [Accessed: 30-Nov-2016].
[29] V. Driessen, "A successful Git branching model," 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model/. [Accessed: 14-Nov-2016].
[30] ST, "Application note Implementation of transmitters and receivers for infrared remote control protocols with MCUs of the STM32F0 and STM32F3 Series."
[31] sbprojects, "IR Remote Control Theory," *12 03 2016*. [Online]. Available: http://www.sbprojects.com/knowledge/ir/. [Accessed: 03-Mar-2017].
[32] K. Goris, "Mechatronica Inleiding tot de PIC microcontroller Kristof Goris," 2008.
[33] "Programming languages." [Online]. Available: https://learn.mikroe.com/ebooks/piccprogramming/chapter/programming-languages/.
[34] "ISR drawing." [Online]. Available: https://learn.adafruit.com/assets/21334.
[35] "GhostDoc." [Online]. Available: http://submain.com/products/ghostdoc.aspx?utm_campaign=ghostdoc&utm_medium=listing&utm_source=visualstudiogallery. [Accessed: 05-May-2017].

Bijlagen

Bijlage A: Code eenvoudige test IDE's

```
#include <mbed.h>

DigitalOut g_LED1(P2_5);
DigitalOut g_LED2(P2_4);
DigitalOut g_LED3(P2_3);
DigitalOut g_LED4(P2_2);

int main()
{
	int teller = 0;
	while (1)
	{
		teller++;
		g_LED1 = !g_LED1;
		wait_ms(500);
		if (teller > 5)
		{
			g_LED2 = 1;
		}
		if (teller > 10)
		{
			g_LED3 = 1;
		}
		if (teller > 15)
		{
			g_LED4 = 1;
		}
		if (teller > 20)
		{
			g_LED2 = 0;
			g_LED3 = 0;
			g_LED4 = 0;
			teller = 0;
		}
	}
}
```

Bijlage B: Code geavanceerde test IDE's

```
#include <mbed.h>
#include <rtos.h>
#include <SysprogsProfiler.h>

DigitalOut g_LED1(P2_5);
DigitalOut g_LED2(P2_4);
DigitalOut g_LED3(P2_3);
DigitalOut g_LED4(P2_2);

InterruptIn irIn(P0_5);
Timer timer;
int tFalling = 0;
int tRising = 0;
int measurement = 0;
#define BUFFSIZE 50
int irBufferTime[BUFFSIZE];
char irBufferValue[BUFFSIZE];
int ir_index = 0;
bool timerStarted = false;

int numberOfCommands = 0;

static void irThreadBody(const void *)
{
	while (1)
	{
		int timeRunning = timer.read_us();
		if (timeRunning > 10000)
		{
			timer.reset();
			timer.stop();
			timerStarted = false;
			numberOfCommands++;
			ir_index = 0;
			for (int i = 0; i < BUFFSIZE; i++)
			{
				irBufferTime[i] = 0;
				irBufferValue[i] = 0;
			}
		}
	}
}

void risingIr()
{
	if (timerStarted == false)
	{
		timer.start();
		timerStarted = true;
	}
	g_LED4 = 0;
	tRising = timer.read_us();
	irBufferTime[ir_index] = tRising;
	irBufferValue[ir_index] = 0;
	timer.reset();
	ir_index++;
}

void fallingIr()
{
	if (timerStarted == false)
	{
```

```
                timer.start();
                timerStarted = true;
        }
        g_LED4 = 1;
        tFalling = timer.read_us();
        irBufferTime[ir_index] = tFalling;
        irBufferValue[ir_index] = 1;
        timer.reset();
        ir_index++;
}

int main()
{
        InitializeInstrumentingProfiler();

        irIn.rise(&risingIr);
        irIn.fall(&fallingIr);

        Thread irThread(irThreadBody);

        int teller = 0;
        while (1)
        {
                teller++;
                g_LED1 = !g_LED1;
                wait_ms(500);
                if (teller > 5)
                {
                        g_LED2 = !g_LED2;
                }
                if (teller > 10)
                {
                        g_LED3 = !g_LED3;
                }
                if (teller > 15)
                {
                        g_LED2 = 0;
                        g_LED3 = 0;
                        teller = 0;
                }

        }
}
```

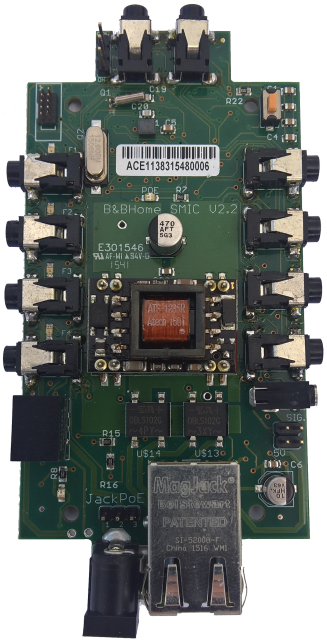
Bijlage C: Softwaredocumentatie van de B&B Home Smart Media IP Controller

Volgend document bevat de automatisch gegenereerde documentatie door Doxygen zoals besproken in hoofdstuk 5.5 “Software documentatie”.

SMIC software documentation

Generated by Doxygen 1.8.13

Author: Robin Moons



Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 SMIC_debug Class Reference

`#include <SMIC_debug.h>`

Public Member Functions

- SMIC_debug (RawSerial &debugPort, DigitalIn &debugJumper)
- void WriteTerminal (char const ∗message)

2.1.1 Detailed Description

Class to debug on SMIC HW v2.2 over serial port

Author

Robin Moons

2.1.2 Constructor & Destructor Documentation

2.1.2.1 SMIC_debug()

```
SMIC_debug::SMIC_debug (
            RawSerial & debugPort,
            DigitalIn & debugJumper )
```

Constructor SMIC_debug

Parameters

debugPort	: the serial interface to send the data
debugJumper	: the pin with the Y/N debugjumper

2.1.3 Member Function Documentation

2.1.3.1 WriteTerminal()

```
void SMIC_debug::WriteTerminal (
            char const * message )
```

Write data to the serial port

Parameters

message	: pointer to the message to send

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_debug.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_debug.cpp

2.2 SMIC_eeprom Class Reference

`#include <SMIC_eeprom.h>`

Public Types

- enum block {
  BLOCK_NETWORK = 1, **BLOCK_SERIAL** = 2, **BLOCK_IR_RCV** = 3, **BLOCK_IR_SEND** = 4,
  BLOCK_5 = 5, **BLOCK_6** = 6, **BLOCK_7** = 7 }

Public Member Functions

- SMIC_eeprom (I2C &dev, SMIC_debug &debugger)
- int write_block (int blockAddress, const char ∗text, int length)
- int getData (char ∗result, int resultSize, int blockAddress, char ∗parameter, int parameterSize)
- int read_block (char ∗result, int size, int block)
- int load_default ()

2.2.1 Detailed Description

SMIC EEPROM interface handles read and write operations to / from EEPROM memory on SMIC

Copyright (c) 2017 Bits & Bytes nv

Author

Robin Moons

2.2.2 Member Enumeration Documentation

2.2.2.1 block

`enum SMIC_eeprom::block`

Enums to define the 8 memoryblocks in the EEPROM

2.2.3 Constructor & Destructor Documentation

2.2.3.1 SMIC_eeprom()

```
SMIC_eeprom::SMIC_eeprom (
            I2C & dev,
            SMIC_debug & debugger )
```

Constructor SMIC_eeprom

Parameters

dev	: the i2c interface
debugger	: the serial debug interface

2.2.4 Member Function Documentation

2.2.4.1 getData()

```
int SMIC_eeprom::getData (
            char * result,
            int resultSize,
            int blockAddress,
            char * parameter,
            int parameterSize )
```

getData gets the specified data from the EEPROM

Parameters

result	: pointer to the array where the result will be stored
resultSize	: size of the array where the result will be stored
blockAddress	: the blockaddres of the EEPROM memory (0-7)
parameter	: pointer to the array with the parameter to search
parameterSize	: size of the array with the parameter to search

Returns

: 0 on succes, -1 when error occures

2.2.4.2 load_default()

```
int SMIC_eeprom::load_default ( )
```

load_default loads the EEPROM with the default settings

Returns

: 0 on succes, -1 when error occures

2.2.4.3 read_block()

```
int SMIC_eeprom::read_block (
            char * result,
            int size,
            int block )
```

read_block reads a block of data from the eeprom

Parameters

result	: pointer to the array where the result will be stored
size	: size of array where the result will be stored
block	: block address to read from

Returns

: 0 on succes, -1 when error occures

2.2.4.4 write_block()

```
int SMIC_eeprom::write_block (
            int blockAddress,
            const char * text,
            int length )
```

write_block writes a block of data to the EEPROM

Parameters

blockAddress	: the block address of the EEPROM memory (0-7)
text	: pointer to the data array to send
length	: length of the data

Returns

: 0 on succes, -1 when error occures

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_eeprom.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_eeprom.cpp

2.3 SMIC_irReceive Class Reference

`#include <SMIC_irReceive.h>`

Public Member Functions

- SMIC_irReceive (InterruptIn &pin, SMIC_debug &debug)
- void enable (bool en)
- int readIrRcvBuffer (char *pointer, int size)
- void enableProto (bool en)

Protected Member Functions

- void startReading (bool startRising)
- void stopReading ()
- bool isReading ()
- void handleRisingEdge ()
- void handleFallingEdge ()
- void decode ()
- char classifyCode ()
- string decode_rc5 ()
- string decode_rc5x ()
- string decode_rc6 ()
- string decode_nec ()
- string decode_pronto ()
- void incWriteIndex ()
- void incReadIndex ()
- void irFalling ()
- void irRising ()
- void irTimeout ()

2.3.1 Detailed Description

SMIC infrared receive interface with internal buffer to store received data

Copyright (c) 2017 Bits & Bytes nv

Author

Robin Moons

2.3.2 Constructor & Destructor Documentation

2.3.2.1 SMIC_irReceive()

```
SMIC_irReceive::SMIC_irReceive (
            InterruptIn & pin,
            SMIC_debug & debug )
```

Create a Ir receive interface at the specified pin

Parameters

pin	: the pin where you connect the ir receiver (with internal demodulator)
debug	: the debuginterface (SMIC_hardware)

2.3.3 Member Function Documentation

2.3.3.1 classifyCode()

```
char SMIC_irReceive::classifyCode ( )  [protected]
```

classify the code to the known protocols

2.3.3.2 decode()

```
void SMIC_irReceive::decode ( )  [protected]
```

analyse the received code and decode it

2.3.3.3 decode_nec()

```
string SMIC_irReceive::decode_nec ( )  [protected]
```

decode protocol in NEC format

Returns

: the nec code

2.3.3.4 decode_pronto()

```
string SMIC_irReceive::decode_pronto ( )  [protected]
```

decode protocol in Pronto format

Returns

: the pronto code

2.3.3.5 decode_rc5()

```
string SMIC_irReceive::decode_rc5 ( )  [protected]
```

decode protocol in rc5 format

Returns

: the rc5 code

2.3.3.6 decode_rc5x()

```
string SMIC_irReceive::decode_rc5x ( )  [protected]
```

decode protocol in rc5x format

Returns

: the rc5x code

2.3.3.7 decode_rc6()

```
string SMIC_irReceive::decode_rc6 ( )  [protected]
```

decode protocol in rc6 format

Returns

: the rc6 code

2.3.3.8 enable()

```
void SMIC_irReceive::enable (
            bool en )
```

Enables or disables the receiver

Parameters

en	: true = enable ; false = disable

2.3.3.9 enableProto()

```
void SMIC_irReceive::enableProto (
            bool en )
```

Enables or disables the receiver to decode in pronto format

Parameters

en	: true = enable ; false = disable

2.3.3.10 handleFallingEdge()

```
void SMIC_irReceive::handleFallingEdge ( )  [protected]
```

read and reset the timer, restart the timeout

2.3.3.11 handleRisingEdge()

```
void SMIC_irReceive::handleRisingEdge ( )  [protected]
```

read and reset the timer, restart the timeout

2.3.3.12 incReadIndex()

```
void SMIC_irReceive::incReadIndex ( )  [protected]
```

increment the read index of circular buffer

2.3.3.13 incWriteIndex()

```
void SMIC_irReceive::incWriteIndex ( )  [protected]
```

increment the write index of circular buffer

2.3.3.14 irFalling()

```
void SMIC_irReceive::irFalling ( )  [protected]
```

handle the falling edge interrupt routine

2.3.3.15 irRising()

```
void SMIC_irReceive::irRising ( )  [protected]
```

handle the rising edge interrupt routine

2.3.3.16 irTimeout()

```
void SMIC_irReceive::irTimeout ( )  [protected]
```

handle the timeout interrupt routine

2.3.3.17 isReading()

```
bool SMIC_irReceive::isReading ( )  [protected]
```

Get the receiver status

Returns

: true if is receiving, false if not receiving

2.3.3.18 readIrRcvBuffer()

```
int SMIC_irReceive::readIrRcvBuffer (
            char * pointer,
            int size )
```

Read the Ir receive buffer

Parameters

pointer	: char pointer to store the data
size	: size of the chararray where we store the data

Returns

: the number of bytes received

2.3.3.19 startReading()

```
void SMIC_irReceive::startReading (
            bool startRising )  [protected]
```

Start the readtimer

Parameters

startRising	: true when the first edge is rising edge, false if falling edge

2.3.3.20 stopReading()

```
void SMIC_irReceive::stopReading ( )  [protected]
```

Stop and reset the timer

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_irReceive.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_irReceive.cpp

2.4 SMIC_irSend Class Reference

`#include <SMIC_irSend.h>`

Public Member Functions

- SMIC_irSend (PinName pin, SMIC_debug debugger)
- int sendIr (char *data, int length)
- bool isSending ()

Protected Member Functions

- int generateArray_rc5 (int adres, int command)
- int generateArray_rc5x (int adres, int command, int extention)
- int generateArray_rc6 (int adres, int command)
- int generateArray_nec (int adres, int command)
- int generateArray_nec_rpt (int times)
- int generateArray_pronto (char *data, int size)
- void toggle_tbit_rc5 ()
- void toggle_tbit_rc5x ()
- void toggle_tbit_rc6 ()
- void timeout ()
- void sendBuffer (int protocol, int repeat)
- void clearBuffer ()

2.4.1 Detailed Description

Class to create IR send pin for SMIC HW v2.2

Author

Robin Moons

2.4.2 Constructor & Destructor Documentation

2.4.2.1 SMIC_irSend()

```
SMIC_irSend::SMIC_irSend (
            PinName pin,
            SMIC_debug debugger )
```

Create a Ir send interface, in the specified pin

Parameters

pin	: the pin where you connect the ir flasher
debugger	: the debuginterface (SMIC_hardware)

2.4.3 Member Function Documentation

2.4.3.1 clearBuffer()

```
void SMIC_irSend::clearBuffer ( )  [protected]
```

Clear the _sendBuffer

Should be called afther each code is sended the needed times

2.4.3.2 generateArray_nec()

```
int SMIC_irSend::generateArray_nec (
            int adres,
            int command )  [protected]
```

Converts the NEC adres and command to the binary values

fils the _sendBuffer with the NEC binary array

Parameters

adres	: the NEC address
command	: the NEC command

Returns

: 0 on succes, -1 on failure

2.4.3.3 generateArray_nec_rpt()

```
int SMIC_irSend::generateArray_nec_rpt (
            int times )  [protected]
```

Generate the NEC repeat sequence

Returns

: 0 on succes, -1 on failure

2.4.3.4 generateArray_pronto()

```
int SMIC_irSend::generateArray_pronto (
            char * data,
            int size )  [protected]
```

Converts the pronto data to the binary values

fils the _sendBuffer with the pronto binary array

Parameters

data	: pointer to pronto data
size	: size of pronto data

Returns

: 0 on succes, -1 on failure

2.4.3.5 generateArray_rc5()

```
int SMIC_irSend::generateArray_rc5 (
            int adres,
            int command )  [protected]
```

Converts the RC5 adres and command to the binary values

fils the _sendBuffer with the RC5 binary array

Parameters

adres	: the RC5 address
command	: the RC5 command

Returns

: 0 on succes, -1 on failure

2.4.3.6 generateArray_rc5x()

```
int SMIC_irSend::generateArray_rc5x (
            int adres,
            int command,
            int extention )  [protected]
```

Converts the RC5x adres, command and extention to the binary values

fils the _sendBuffer with the RC5x binary array

Parameters

adres	: the RC5x address
command	: the RC5x command
extention	: the RC5x extention

Returns

: 0 on succes, -1 on failure

2.4.3.7 generateArray_rc6()

```
int SMIC_irSend::generateArray_rc6 (
            int adres,
            int command )  [protected]
```

Converts the RC6 adres and command to the binary values

fils the _sendBuffer with the RC6 binary array

Parameters

adres	: the RC6 address
command	: the RC6 command

Returns

: 0 on succes, -1 on failure

2.4.3.8 isSending()

```
bool SMIC_irSend::isSending ( )
```

Get the status of the Ir interface

Returns

: true if interface is sending, false if not

2.4.3.9 sendBuffer()

```
void SMIC_irSend::sendBuffer (
            int protocol,
            int repeat )  [protected]
```

Starts the transmission of data in _sendBuffer

Parameters

protocol	: the protocol, needed for correct start
repeat	: the number of times the signal should be repeated

2.4.3.10 sendIr()

```
int SMIC_irSend::sendIr (
            char * data,
            int length )
```

Send an ir code

Parameters

data	: the pointer to the array containing the coded ir data
length	: the size of the data

Returns

: 0 on succes, -1 on failure

2.4.3.11 timeout()

```
void SMIC_irSend::timeout ( )  [protected]
```

Callback function when interrupt of _timeout occures

sends the data in the _sendBuffer with the correct timing

2.4.3.12 toggle_tbit_rc5()

```
void SMIC_irSend::toggle_tbit_rc5 ( )  [protected]
```

Toggle the toggle bit of RC5 code Should be called afther each sended RC5 code

2.4.3.13 toggle_tbit_rc5x()

```
void SMIC_irSend::toggle_tbit_rc5x ( )  [protected]
```

Toggle the toggle bit of RC5x code

Should be called afther each sended RC5x code

2.4.3.14 toggle_tbit_rc6()

```
void SMIC_irSend::toggle_tbit_rc6 ( )  [protected]
```

Toggle the toggle bit of RC6 code Should be called afther each sended RC6 code

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_irSend.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_irSend.cpp

2.5 SMIC_network Class Reference

`#include <SMIC_network.h>`

Public Member Functions

- SMIC_network (SMIC_debug debugger, Watchdog &wdt)
- void Set_network (const char *smic_ip_adres, const char *smic_netmask, const char *smic_gateway, const char *server_ip_adres, int server_port)
- void TCPsocket_connect_loop ()
- void TCPSocket_send (char *data, int length)
- int TCPSocket_receive (char *data, int size)

Protected Member Functions

- void incSendIndex ()
- void incMemIndex ()

2.5.1 Detailed Description

SMIC network interface with internal RX TX buffers Copyright (c) 2017 Bits & Bytes nv

Author

Robin Moons

2.5.2 Constructor & Destructor Documentation

2.5.2.1 SMIC_network()

```
SMIC_network::SMIC_network (
            SMIC_debug debugger,
            Watchdog & wdt )
```

Create a netwerk interface

Parameters

debugger	: the debuginterface (SMIC_hardware)
wdt	: the watchdog interface that should be kicked while in connect loop

2.5.3 Member Function Documentation

2.5.3.1 incMemIndex()

```
void SMIC_network::incMemIndex ( )  [protected]
```

incement the memory index

2.5.3.2 incSendIndex()

```
void SMIC_network::incSendIndex ( )  [protected]
```

increment the send index

2.5.3.3 Set_network()

```
void SMIC_network::Set_network (
            const char * smic_ip_adres,
            const char * smic_netmask,
            const char * smic_gateway,
            const char * server_ip_adres,
            int server_port )
```

Set the client and server settings and connect to network

will block if connection fails

Parameters

smic_ip_adres	: the client ip address
smic_netmask	: the client subnetmask
smic_gateway	: the client gateway
server_ip_adres	: the server ip address
server_port	: the server port to setup TCP/IP connection

2.5.3.4 TCPsocket_connect_loop()

```
void SMIC_network::TCPsocket_connect_loop ( )
```

Connect to the TCP Socket

Will block if connection fails

2.5.3.5 TCPSocket_receive()

```
int SMIC_network::TCPSocket_receive (
            char * data,
            int size )
```

Read the TCPIP receive buffer

non blocking

Parameters

data	: pointer where the data will be stored
size	: size of array where the data will be stored

Returns

: number of received bytes.

2.5.3.6 TCPSocket_send()

```
void SMIC_network::TCPSocket_send (
            char * data,
            int length )
```

Send over TCPIP connection

non blocking

Parameters

data	: pointer where the data is stored
length	: lenth of data

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_network.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_network.cpp

2.6 SMIC_serial Class Reference

`#include <SMIC_serial.h>`

Public Member Functions

- SMIC_serial (RawSerial &port, int baudrate)
- void enableReceive ()
- void send (char ∗data, int size)
- void send_hex (char ∗data, int size)
- int readRcvBuffer (char ∗data, int size)

Protected Member Functions

- void handleRxInt ()
- void handleTimeout ()
- void incWriteIndex ()
- void incReadIndex ()

2.6.1 Detailed Description

SMIC serial interface with internal buffers for interruptbased Receive Copyright (c) 2017 Bits & Bytes nv

Author

Robin Moons

2.6.2 Constructor & Destructor Documentation

2.6.2.1 SMIC_serial()

```
SMIC_serial::SMIC_serial (
            RawSerial & port,
            int baudrate )
```

Create a serial send/receive bufferd interface

Parameters

port	: the serial port
baudrate	: the baudrate

2.6.3 Member Function Documentation

2.6.3.1 enableReceive()

```
void SMIC_serial::enableReceive ( )
```

Enable serial receive

2.6.3.2 handleRxInt()

```
void SMIC_serial::handleRxInt ( )  [protected]
```

Interrupthandler for rx

2.6.3.3 handleTimeout()

```
void SMIC_serial::handleTimeout ( )  [protected]
```

Interrupt handler for receive timeout

2.6.3.4 incReadIndex()

```
void SMIC_serial::incReadIndex ( )  [protected]
```

increment the read index

2.6.3.5 incWriteIndex()

```
void SMIC_serial::incWriteIndex ( )  [protected]
```

increment the write index

2.6.3.6 readRcvBuffer()

```
int SMIC_serial::readRcvBuffer (
            char * data,
            int size )
```

read the receivebuffer

Parameters

data	: pointer to the array where the readed data will be stored
size	: size of the array where the readed data will be stored

Returns

: number of readed bytes

2.6.3.7 send()

```
void SMIC_serial::send (
            char * data,
            int size )
```

send data over serial interface (not buffered)

Parameters

data	: pointer to the char array to send
size	: size of the array to send

2.6.3.8 send_hex()

```
void SMIC_serial::send_hex (
            char * data,
            int size )
```

send raw (hexadecimal) data over serial interface (not buffered)

Parameters

data	: pointer to the char array to send
size	: size of the array to send

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_serial.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/SMIC_serial.cpp

2.7 Watchdog Class Reference

`#include <Watchdog.h>`

Public Member Functions

- void kick (float s)
- void kick ()
- bool causedReset ()

2.7.1 Detailed Description

Class to create watchdog for SMIC HW v2.2

Author

Robin Moons

2.7.2 Member Function Documentation

2.7.2.1 causedReset()

```
bool Watchdog::causedReset ( )
```

Check if the watchdog caused the reset

Returns

: true if watchdog caused the reset, false if not

2.7.2.2 kick() [1/2]

```
void Watchdog::kick (
            float s )
```

Kick the watchdog

Parameters

s	: Time in seconds

2.7.2.3 kick() [2/2]

```
void Watchdog::kick ( )
```

Kick the watchdog

The documentation for this class was generated from the following files:

- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/Watchdog.h
- C:/Users/Robin/Documents/Git_Repositories/SMIC_v2_2/src/SMIC/SMIC/Watchdog.cpp

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Efficiënte softwareontwikkeling voor ARM-processoren

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2017**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Moons, Robin

Datum: **3/06/2017**