Omprogrammering av en XMEGA32A4U-mikrokontroller via Bluetooth

Daniel Fors

Diplomarbete i datateknik
Handledare: Jerker Björkqvist och Hannu Toivonen
Institutionen för informationsteknologi
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
December 2020

Sammanfattning

Fler och fler apparater kontrolleras av digitala system, till och med en strömavbrytare kan innehålla en mikrontroller. Denna utveckling har möjliggjort att produkter förbättras efterhand med hjälp av mjukvaruuppdateringar, något som bland annat kan ses i bilindustrin. Diplomarbetet handlar om ett projket genomfört åt företaget Mirka Ab inom detta område. Projektet gick ut på att utveckla fast programvara, vilket skulle möjliggöra trådlösa uppdateringar, åt en familj av handhållna slipmaskiner företaget sålde.

Avhandlingen beskriver de tekniska förutsättningar som fanns. Främst gällande den XMEGA32A4U mikrokontroller som slipmaskinerna styrdes av samt det nRF8001 Bluetooth-modem som fanns tillgängligt. Därpå behandlas design- och utvecklingsarbetet som utfördes för att förverkliga projektet. Resultatet blev en fristående startladdare under 4 KiB med vars hjälp slipmaskinenernas styrprogram kunde uppdateras genom en Bluetooth-anslutning.

Förord

Jag vill börja med att tacka Mirka Ab och produktutvecklingschef Caj Nordström för möjligheten att utföra mitt arbete vid Mirkas utvecklingsenhet. Det här är ett diplomarbete som dragit ut på tiden och mångas tålamod har säkerligen satts på prov. Tiden rann ut till den grad att min första handledare Hannu Toivonen hann gå i pension. Jag hoppas att vederbörande njuter av pensionstiden!

Jag vill speciellt tacka mina föräldrar Fjalar och Mayvor Fors för deras uthållighet med mig samt för korrekturläsning av avhandlingen. Min bror Joakim Fors med fru Maria förtjänar också tack för det stöd de gett mig. Därtill vill jag även tacka min morbrors familj – familjen Lillqvist i allmänhet och Joel Lillqvist synnerhet för att han sporrat mig att slutföra skrivandet.

Daniel Fors

Innehåll

Tabeller i

Figurer ii

Förkortningar iii

1 Introduktion 1
1.1 Projektets målsättning . . . 1
1.2 Disposition . . . 2

2 Styrenhet 4
2.1 Atmel AVR ATxmega32A4U . . . 4
2.1.1 Minne . . . 5
2.1.2 Säkerhet . . . 6

3 Kommunikation 7
3.1 Nordic Semiconductor nRF8001 . . . 8
3.2 ACI-gränssnittet . . . 8
3.3 Bluetooth Low Energy . . . 9
3.3.1 GATT . . . 9
3.4 Mirka Diagnostics . . . 11

4 Planering 13
4.1 Kravbild . . . 13
4.1.1 Fristående kod . . . 14
4.1.2 Kommunikationkodens utrymmeskrav . . . 15
4.2 Slutfas av kravsättningen . . . 15
4.2.1 Påtänkta förbättringar . . . 16
4.2.2 Bluetooth-motpart . . . 17
4.3 Design av startladdaren . . . 17
4.3.1 Bluetooth-anslutning . . . 18

5 Genomförande 23
5.1 Slipmaskinens programvara . . . 23
5.1.1 Övergång till och låsning i omprogrammeringsläge . . . 23
5.1.2 Modifikation av Nordic Semiconductors kod . . . 24
5.1.3 Kommunikationsrör . . . 25
5.1.4 Uppdateringsprocessen . . . 26
5.1.5 Mellanlagring av emottagen information . . . 27
5.1.6 Upplåsning och kontrolltal . . . 28

5.2 Diagnostics på andra sidan anslutningen 29
5.2.1 Gränssnitt 29
5.2.2 Programavbild 30
5.3 Test av den nya startladdaren 31
5.3.1 Det normala inbyggda programmet spelar roll 31
5.3.2 Oavslutade och felaktiga uppdateringar 31

6 Diskussion **36**
6.1 Vidareutveckling av omprogrammerare för DEROS 36
6.1.1 Putsning av slipmaskinens omprogrammeringskod . . . 36
6.1.2 Distribution av uppdateringar 37
6.1.3 Fel funna i efterhand 38
6.2 Tankar kring trådlösa uppdateringar 39
6.2.1 Säkerhet 39
6.2.2 Överföringshastighet 39

7 Slutsats **41**

8 Referenser **42**

Tabeller

3.1 Batteritjänst med sin karakteristika 10
5.1 Kommandooktetter 26
5.2 Exempel-rad i en Intel HEX-fil 30

Figurer

4.1 Presumtiv flödesplan för slipmaskinen. 21
4.2 Presumtiv flödesplan för Diagnostics. 22
5.1 Tillståndsdiagram för startladdaren. 28
5.2 Uppdateringssekvens. 32
5.3 Slutgiltig flödesplan för startladdaren. 33
5.4 Slutgiltig flödesplan för startladdaren. 34

Förkortningar

AES	Advanced encryption standard, krypteringsmetod
ACI	Application controller interface, kommunikationsgränssnitt från Nordic Semiconductors
ATT	Attribute protocol, överföringsprotokoll som används av GATT
BLE	Bluetooth low energy, Bluetooth protkoll med tyngdpunkt på låg strömförbrukning
CRC	Cyclic redundancy check, kontrolltalsberäkning av data
DEROS	Direct electric random orbital sander, familj av slipmaskiner från företaget Oy Mirka Ab
DES	Data encryption standard, krypteringsmetod
DMA	Direct memory access, kommunikationskanal som tillåter överföring mellan minnen och kringutrustning utan processorns iblandning
EPROM	Erasable programmable read-only memory, endast läsbart minne som kan programmeras i efterhand men även raderas med hjälp av ultraviolett strålning
EEPROM	Electrically erasable programmable read-only memory, endast läsbart minne som kan programmeras i efterhand men även raderas med elektriska signaler
FGPA	Field-programmable gate array, grindmatris som kan programmeras om av en kund
GAP	Generic access profile, ramverk för hur Bluetooth-enheter hittar varandra och påbörjar anslutningar
GATT	Generic attribute profile, protokoll som beskriver hur datakommunikation sker med Bluetooth Low Energy
JTAG	Joint test action group, en standard utvecklad för felsökning av kretsar men vars gränssnitt numera även bland annat används för att programmera omprogrammerbara kretsar
Mirka	Oy Mirka Ab, tidigare Oy KWH Mirka Ab
NOP	No-op, en instruktion som kan använda klockcyklar men inte utför något egentligt arbete
PROM	Programmable read-only memory, endast läsbart minne vars innehåll kan programmeras efter kretsen producerats
ROM	Read-only memory, endast läsbart minne vars innehåll bestäms då kretsen produceras
SPI	Serial peripheral interface, serieportsgränssnitt

SPM	Store program memory, instruktion för skrivning till programminnet
SRAM	Static random access memory, en typ av arbetsminne
UUID	Universally unique identifier, universiellt unik identifikationskod

1 Introduktion

I dagsläget kontrolleras många vardagliga maskiner med mjukvara istället för med analoga styrenheter. Kylskåp[18], tvättmaskiner[11], till och med strömbrytare[9] kan innehålla någon sorts digitalt kontrollsystem. Denna stora utbredning av mjukvara har också lett till nya möjligheter. Bland dessa är förmågan att enkelt och billigt förbättra deras prestanda även efter att varan lämnat produktionslinjen, något som är nästintill omöjligt för de flesta produkter med analoga styrmekanismer.

I stor utsträckning har dock dylika uppgraderingar i praktiken varit begränsade till persondatorer och telefoner samt anordningar som är tätt kopplade till dem, såsom nätverksutrustning. Många sådana styrenheter har nämligen inte varit byggda med uppdateringar i åtanke. Därmed har de haft minnen av den typ som inte kan omprogrammeras eller saknat enkla kommunikationskanaler för att ändra styrenhetens beteende. Det betyder att det krävs en del extra arbete med mjukvara för att möjliggöra uppdateringar vilket bidrar till en kostnadsökning för producenten.

Allt detta har också en baksida. Uppdateringar ger inte enbart möjlighet för producenten eller välmenande personer att förändra en styrenhets beteende, men även åt illasinnade aktörer. Att dylika säkerhetsproblem är verkliga kan inte minst ses i det så kallade sakernas internet där olika anslutna apparater och maskiner kan utgöra en säkerhetsrisk från flera olika perspektiv[10].

1.1 Projektets målsättning

Projektet har sin begynnelse hos Mirka och idéer kring just uppdateringar. Företaget tillverkar slipmaterial samt slip- och polermaskiner och de bedriver ett konstant utvecklingsarbete inom denna bransch. Projektet handlar specifikt om en samling slipmaskiner med produktnamnet DEROS som är tillgängliga från företaget. Maskinerna var vid tidpunkten för arbetet med detta diplomarbete begränsade till sitt ursprungliga fasta program efter att den lämnat fabriken och det fanns inte någon uppdateringsmöjlighet. Det här ville personalen på utvecklingsenheten ändra på. De ville tillåta omprogrammering av maskinerna även i efterhand. Primärt var planen avsedd för företagets bruk i dess underhållscentraler för att kunna förbättra produkten i efterhand, men även för att potentiellt använda möjligheten under produktionsfasen.

Tanken var att ett existerande Bluetooth-modem i maskinerna skulle användas, då detta var den enda datakommunikationsvägen eftersom datauttag för kablar saknades. Bluetoothchippet användes vid tillfället för att hämta diagnostikloggar från slipmaskinerna och för att i specialsammanhang rent av styra deras motor. Kommunikationsdelen var således avklarad, det var endast en fråga om att undersöka resten av systemets förmågor och efteråt utföra det praktiska arbetet. Mirka ställde därför följande mål:

- Undersök om det finns tekniska förutsättningar för omprogrammering givet den dåvarande hårdvaran.
- Om så är fallet: utveckla det fasta programmet vidare så att denna möjlighet införlivas.
- Utveckla ett program för persondatorer som är kapabelt att utnyttja den förbättrade programvaran.

1.2 Disposition

Kapitel 2 börjar med en kort diskussion om omprogrammerbara kretsar; den tekniska utveckligen som möjliggjort uppdateringar av styrenheter i fältet. Avsnittet fortsätter med en presentation av den mikrokontroller som används av Mirkas DEROS-maskiner. Fokus läggs på mikrokontrollerns minne samt dess inbyggda krypteringsmetoder och kontrolltalsberäkning då dessa aspekter haft störst relevans för projektet.

I kapitel 3 behandlas därefter kommunikationsaspekten kring projektet och dess betydelse. Först ges en allmän blick över kommunikationskanaler. Sen tittas på det gränssnitt som användes för att kommunicera med slipmaskinens trådlösa modemet i projektet. Det fortsätter med att betrakta Bluetooth Low Energy-protokollet och dess GATT-server. Sist diskuteras Mirkas Diagnostics program och hur det kommunicerade med slipmaskinerna när projektet startades.

Kapitel 4 handlar om planering kring det praktiska förverkligandet. Först utforskas de begränsningar som projektet hade. Både de rent tekniska begränsningarna som sätts av hårdvaran men även de krav som ställdes på omprogrammeraren från Mirkas sida. Därefter diskuteras förbättringar som kunde göras om projekttid och minnesutrymme fanns tillgängligt. Slutligen diskuteras den preliminära designplanen av omprogrammeraren.

I kapitel 5 tas genomförandet av den planerade designen upp och de praktiska bekymmer som uppkom under projektets gång. Kapitlet börjar med information om programmeringen av slipmaskinen och efteråt arbetet på Diagnostics. Efteråt behandlas de tester som gjordes av den slutgiltiga produkten.

Kapitel 6 innehåller tankar kring hur man skulle kunna vidareutveckla den produkt som projktet levererade samt tekniska fel som hittats i projektkoden under skrivandet av denna avhandling. Några synpunkter kring trådlösa uppdateringar läggs även fram.

I kapitel 7 summeras projektet.

2 Styrenhet

Tidigare kontrollerades maskiner av antingen mekaniska styrenheter eller statiska elektriska kretsar. De var skräddarsydda för ett specifikt ändamål och när de en gång blivit tillverkade var möjligheten att modifiera deras beteende synnerligen begränsat. Nya idéer och funktioner krävde att ett helt nytt fysiskt kontrollsystem byggdes. Introduktionen av bland annat mikrokontroller har emellertid öppnat för en realistisk möjlighet att förändra styrenheten i efterhand. Med sänkta kostnader har deras användning brett ut sig inom allt fler områden och de kan nu hittas i även de mest elementära maskiner.

En mikrokontroller är i sig inte tillräcklig, utan lika kritiskt är hur dess programvara lagras. De första mikrokontrollerna använde nämligen minnen av typen ROM, PROM eller EPROM. Innehållet på ett ROM fastställs under tillverkningen av minnet medan ett tomt PROM endast kan programmeras en enda gång. Sålunda skulle man behöva byta ut hela minnet för att ändra vad mikrokontrollern kör. Gällande ett EPROM finns möjligheten att radera dess innehåll och skriva ett nytt program till den, men metoden som används för detta lämpar sig dåligt för ändamål utanför utvecklingsarbetet. Det är först när minneskretsar som kan omprogrammeras i fältet började tillverkas som uppdateringar blev en rimlig möjlighet. Här bör nämnas att uppdateringsmöjligheter kan skapas även utan en mikrokontroller genom att till exempel använda omprogrammerbara grindmatriser.

Trots att omprogrammerbara minnen numera inte är ovanliga är det ändå långt från att alla mikrokontrollers programminnen lätt kan ändras, det finns nämligen andra aspekter som även sätter sina begränsningar. Bland annat används ofta speciella omprogrammerare för att ändra innehållet i en mikrokontroller. För att förbigå den krävs en del utvecklingsarbete. Dessutom krävs en kommunikationskanal till mikrokontrollern. Båda dessa medför en extra kostnad som måste rättfärdigas med ett verkligt behov. Mirkas slipmaskin uppfyllde emellertid båda dessa krav.

2.1 Atmel AVR ATxmega32A4U

Hjärnan i Mirkas DEROS-maskiner är en ATxmega32A4U-mikrokontroller, en av flera tillgängliga variationer i XMEGA-familjen från Atmel. XMEGA-familjen är specialiserade på inbyggda system och har, jämfört med en modern persondator eller smarttelefon, synnerligen begränsad minneskapacitet

och beräkningskapacitet. Klockfrekvensen hos en 32A4U är som högst 32 MHz med de inbyggda klockkällorna[4] och de medföljande minnena räknas endast i kibioktetter. 4 kibioktetter av flyktigt arbetsminne och 42 kibioktetter av icke-flyktigt minne varav hela 40 kibioktetter är programminne. Fördelen med normala mikrokontrollrar som dessa är deras låga kostnad och låga strömförbrukning.

2.1.1 Minne

Mikrokontrollrarna i XMEGA-familjen har tre primära minnestyper: flashminne som innehåller programkod, SRAM som arbetsminne samt EEPROM för icke-flyktig informationslagring[3]. Det finns även minnesadresser till olika in- och utportar samt moduler. Utöver dessa finns ett antal säkringsoktetter och låsbitar, speciella minnen som används för att konfigurera mikrokontrollern. Användningen av omprogrammerbart flashminne är en del av pusslet som möjliggjorde projektets genomförande. All programkod måste däremot befinna sig i flashminnet, kod på andra minnesplatser kan inte exekveras av processorn.

Programminnet delas in i två huvuddelar, tre totalt. Den viktigaste indelningen är den mellan applikationsavsnittet som är 36 kibioktetter och startladdaravsnittet som är 4 kibioktetter stort. Utöver de två nämnda huvuddelarna är applikationsavsnittet vidare delat i två. Av 40 kibioktetter är 32 ämnade för programkod. De resterande 4 kibioktetterna är avsatta för att ge utrymme åt icke-flyktig information om man vill spara sådant på flashminnet. Trots denna uppdelning kan hela programminnet innehålla programkod, men kod som befinner sig i startladdarområdet har vissa specialegenskaper.

Programvaran i mikrokontrollern kan inte direkt skriva till de icke-flyktiga minnena. Istället måste man utnyttja speciella mellanlager. För att fylla mellanlagren används en kombination av pekare och register, därefter kan mellanlagrets skrivas till en viss position i det icke-flyktiga minnet. I båda fallen berättar man först åt styrenheten för de icke-flyktiga minnena vad som skall utföras medan instruktionen SPM påbörjar själva processen. Instruktionen är därmed kritisk om man vill programmera om mikrokontrollern genom dess egna mjukvara istället för att förlita sig på en extern programmerare. SPM-instruktionen är endast funktionell i startladdarområdet, inom resterande programminne ses den som en NOP-instruktion. En speciell egenskap då dylika skrivningar sker är att kod inom startladdarsektionen kan fortsätta exekveras under tiden förändringar av applikationsdelen pågår. I den

händelse att flashminnets startladdarområde ändras kan likafullt ingen kod exekveras och processorn står stilla tills ändringen slutförts. Alla tre delar av programminnet kan ha sina egna skriv- och läsrättigheter vilka bestäms med hjälp av låsbitar.

2.1.2 Säkerhet

Ett minnesområdes rättigheter kan endast ändras till ett strängare läge via mikrokontrollerns egna kod. Till exempel kan ett minnesområde, som man till exempel inte får läsa, inte låsas upp genom någon instruktion. Det gör det möjligt att hindra en obehörig från att enkelt ta till sig innehållet även om den har tillgång till omprogrammeringsmöjligheten. Ändringar av låsbitarna kräver en extern omprogrammerare och dess användning leder till att hela programminnet först töms.

Två krypteringsmotorer finns färdigt inbyggda i mikrokontrollern: DES och AES. DES använder sig av en 64 bitars nyckel, dock är egentligen endast 56 av dem själva nyckeln. Datainnehållet i denna krypteringsmetod bygger på block av 8 oktetter. För att utnyttja denna kryptering behövs endast det normala processorregistret samt en instruktion för att utföra den. Här bör påpekas att DES inte längre anses vara en tillräckligt säker krypteringsmetod[16].

AES kryptering å andra sidan har sin egen modul som utför arbetet. Den bygger på en 128 bitars version av metoden och arbetar på 128 bitar i taget, eller 16 oktetter. Modulen har sitt eget register där nyckeln och datan sätts, varvid modulen kan startas genom att skriva till ett speciellt AES-kontroll-register. Efter 375 kringutrustningsklockcyklar[1] är processen färdig och ett status register ändras därefter.

Mikrokontrollern har även en modul för kontrolltalsberäkning med två olika metoder: CRC-16 (CRC-CCITT) med polynomet $x^{16}+x^{12}+x^5+1$ och CRC-32 (IEEE 802.3) med polynomet $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$. Bitarna i resultatet vänds bakfram och inverteras. Dessa kan köras antingen på programminnet, på en dataström som går genom en DMA-kanal, eller genom direkt datainmatning. När flashminnet är källan kan kontrolltalet antingen beräknas för hela minnet eller valda delar av densamma, men CRC-32 används automatiskt i vardera fallen.

[1] Minst samma klockfrekvens som processorn

3 Kommunikation

Att föra över en uppdatering till den inbyggda enheten är ett annat grundkrav. Någon sorts kommunikationskanal till mikrokontrollerns minne krävs, alternativt att hela minnet lätt kan bytas ut. Moderna mikrokontrollrar har vanligtvis någon sorts kabelgränssnitt som möjliggör programmering och felsökning av enheten. Det kan handla om en serieport kombinerat med ett JTAG-gränssnitt[5]. En speciell anordning kallad en programmerare sitter ofta mellan mikrokontrollern och en dator, där kontakten till datorn använder ett USB kontaktdon.

Metoden fungerar under utvecklingsprocessen men att göra användningen mer utbredd är besvärligt på grund av kravet på att inneha den nämnda programmeraren. Speciellt om man vill tillåta slutanvändare att uppgradera det fasta programmet krävs vanligare kommunikationskanaler. Om man vill ha en kabelanslutning blir USB det nästintill självklara valet. Denna standard har en enorm utbredning och återfinns överallt inom inom den moderna datavärlden. Att köpa en ny persondator som saknar USB-uttag lär vara svårt och även surfplattor samt smarttelefoner har ofta någon variant av USB. En kabel har också sina fördelar då det ger en väldigt säker kommunikationskanal i dubbel bemärkelse. Kontakten till mikrokontrollern är mer pålitlig då dess signal inte behöver bekymra sig över fysiska barriärer eller frekvensstörningar. Det är också svårare för någon annan att avlyssna kommunikationen.

Att de trådlösa kanalerna inte har någon lika självklar standard är också en nackdel. Bluetooth och Wi-Fi (IEEE-802.11) är troligtvis de två vanligaste av de som finns[1], men olikt USB är det långt ifrån säkert att en bordsdator har någondera av dem. Valet mellan dem är mer en fråga om tekniska fördelar med bättre räckvidd för Wi-Fi men lägre strömförbrukning med Bluetooth. Trots att en kabelanslutning verkar mer attraktiv på grund av USB så finns det också fördelar med att använda ett trådlöst protokoll, främst med tanke på kontaktdonet som krävs. Det ställer sina klara krav både på de fysiska dimensionerna av enheten men även på kontaktdonets tillgänglighet. Oberoende av dess för- och nackdelar så var kommunikationsvägen i projektet redan givet; det skulle ske genom ett nRF8001 Bluetooth-modem.

3.1 Nordic Semiconductor nRF8001

Chippet är byggt för att sköta trådlös kommunikation enligt Bluetooth 4.0 specifikationerna[13]. Mera preciserat avses de som gäller Bluetooth Low Energy, också kallat Bluetooth Smart. Modemet är ämnat för kringutrustning som en server i klientserver-förhållandet. Det var redan en komponent av det existerande kretskortet i DEROS:arna och stod därmed i direkt kontakt med mikrokontrollern.

3.2 ACI-gränssnittet

Mellan mikrokontrollern och modemet användes ACI som kommunikationsprotokoll, med 5-stifts SPI som den underliggande fysiska kanalen. Inom SPI sätts nRF8001 som en slav, men detta förhållande fungerar inte som vanligt där den andra parten styr hela förloppet. Modemet kan nämligen när som helst fråga efter ett kommunikationsfönster med mikrokontrollern, olikt det normala SPI-förloppet där slaven alltid passivt väntar på att ett kommunikationsfönster påbörjas. Det här åstadkoms genom att nyttja två stift för att meddela när ett fönster skall öppnas, olikt det normala SPI-protkollet som endast använder ett. De två kanalerna benämns REQN och RDYN. Mikrokontrollern signalerar via REQN när den är redo att kommunicera och nRF8001 svarar via RDYN. Det motsatta händer när modemet är den som vill kommunicera något, den signalerar via RDYN och mikrokontrollern svarar med RDYN när den är redo.

Kommunikationsgränssnittet till modemet kan delas i två kategorier: systemkommandon och datakommandon. Systemkommandon är sådana som styr själva nRF8001-chippet och dess beteende medan datakommandon handlar om lokalt påbörjade datakommunikation via Bluetooth med en ansluten motpart. Modemet svarar å sin sida med hjälp av så kallade händelser. Systemhändelser skickas som svar på systemkommandon eller om något hänt med modemet, såsom att en Bluetooth-anslutning avslutats. Datahändelser är i sin tur svar på fullföljda datakommandon eller skapade av nRF8001-modemet då den har fått information från en ansluten motpart.

Dataöverföringar påbörjas inte av direkta Bluetooth GATT-kommandon. Istället används en högre abstraktionsnivå som bygger på virtuella rör. Dessa rör är länkade till specifika karakteristikor med inställningar för bland annat dataflödesriktning. Mikrokontrollern styr användningen av dessa rör

som på nRF8001-modemets sida blir översatta till de nödvändiga Bluetooth-kommandona.

3.3 Bluetooth Low Energy

Länken utåt påbörjas med en annan Bluetooth-enhet genom GAP-ramverket. Här återfinns de grundläggande funktionerna för att enheterna skall finna varandra och påbörja en anslutning. Den har emellertid inte desto större betydelse för överföringen i sig, här kommer istället GATT-protokollet in i bilden. Före GATT behandlas vidare kan det påpekas att det finns en viss begränsning gällande överföringshastighet från modemets sida som inte är låst i BLE-specifikationerna. Chippet klarar nämligen endast av paket med en nyttolast på 20 oktetter. Därtill kan modemet skicka och ta emot högst ett paket för varje kommunikationsintervall[2]. Intervallen som tillåts enligt specifikationerna är mellan 7,5 ms – 4000 ms. Då nRF8001 chippet stöder ett kommunikationsintervall på 7,5 ms, kombinerat med 20 oktetter stora data paket[6], blir den maximal överföringshastigheten $\frac{20\,\text{oktett}}{7{,}5\,\text{ms}} \approx 2667\frac{\text{oktett}}{\text{s}}$.

3.3.1 GATT

För att börja förstå GATT kan man starta med dess syfte. Det är ämnat i första hand för enklare enheter som en användare endast vill få en begränsad mängd information från, inte för stora dataflöden mellan två fulländade datorer. I GATT-ramverket agerar en av enheterna som en klient medan den andra är en server. Serverrollen är framförallt ämnad för kringutrustning, till exempel en enkel vädersensor. Klient är den som påbörjar händelserna genom att göra förfrågningar av servern. I det nämnda fallet frågas kanske efter temperatur eller vindhastighet. Här bör nämnas att rollerna inte nödvändigtvis måste vara de samma hela tiden, de kan ändras enligt behov.

ATT använder sig av något som kallas attribut för att tillhandahålla data, de innehåller den information som en klient kan vara intresserad av. De är lagrade i servern och kan helt enkel sägas vara värden med några extra kringliggande egenskaper. Det är själva värdet och några variationer av olika tillstånd[2] som berättar hur en klient kan använda dem. Sen finns även attributets typ i formen av ett universellt unikt ID, eller UUID. Slutligen har

[2]tillstånd är definierad av protokoll på en högre abstraktionsnivå

Tabell 3.1: Batteritjänst med sin karakteristika

Attribut					
Handtag	Typ	Värde			Rättigheter
0xLLLL	0x2800	0x180F (Tjänstens UUID)			0x02 (Läs)
0xNNNN	0x2803	0x12 (Karakteristikans egenskaper)	0xMMMM (Handtag för karakteristikans attribut)	0x2A19 (Karakteristikans UUID)	0x02 (Läs)
0xMMMM	0x2A19	0x? (Karakteristikans värde)			0x12 (Läs och underrätta)
0xVVVV	0x2902	0x0001 (Karakteristikans inställning)			0x0A (Läs och skriv)

attributet ett handtag, en kortare id-kod som används av datamängdsskäl under kommunikationstillfällena. Detta handtag är inte unikt och kan variera mellan olika servrar, något som ett UUID inte bör göra. Det är endast 16 bitar långt i jämförelse med ett UUID som bör vara 128 bitar.

GATT använder attributen som sin grund och bygger vidare på dem. Här är det så kallade karakteristikor som innehåller de intressanta värdena. En eller flera karakteristikor sätts sedan i logiska grupperingar som kallas tjänster. En tjänst använder minst ett attribut och en karakteristika minst två; fler kan emellertid utnyttjas för att vidare definiera hur de bör brukas. Här kommer UUID:n in i bilden på nytt, GATT-profile har nämligen färdigt definierade attribut-UUID:n för allt utom den som innehåller karakteristikans värde. Attributet för en tjänst har exempelvis attribut-UUID:n 0x2800. Notera att detta är ett UUID för ett attribut som definerar en tjänst. Tjänsten i sig behöver även det ett UUID, men detta UUID är valbart och lagras i attributets värdepost.

Det finns ett antal 16 bitars[3] UUID:s definierade av Bluetooth SIG som det är fritt fram att använda för tjänster och karakteristikor. Batteritjänsten Battery Service är ett exempel på ett sådant. Dess UUID är 0x1811 och den innehåller endast en karakteristika: batterinivån i procent. Tabell 3.1 visar hur batteritjänsten ser ut från attributets synvinkel. Enhetsinformationstjänsten Device Information är ett annat exempel som har fler karakteristikor, såsom tillverkarens namn och det fasta programmets version. Om ingen av de färdigt definierade tjänsterna eller karakteristikorna passar in på det tilltänkta

[3]De är egentligen förkortningar för ett spann av 128 bitars UUID:n

användningsområdet är det även möjligt att skapa sina egna enligt behov. Olikt de 16 bitar långa från Bluetooth SIG bör dessa vara 128 bitar långa.

Rättigheterna för de speciella GATT-profiltyperna av attribut är tämligen fixerade. När det gäller värdet på en karakteristika är det emellertid mera fritt, såvida man förstås inte använder en UUID som har rättigheter fastställda av Bluetooth SIG. I tabellen med batteritjänsten markeras med rött hur rättigheterna först definieras av karakteristikans egenskaper och hur de relaterar till attributet som innehåller karakteristikans värde. I exemplet är läsrättigheter obligatoriska för batterinivåskarakteristikan för en batteritjänst, medan man får välja om man vill implementera underrättningsmöjligheten. Om å andra sidan man implementerar den så måste man även ha ett attribut för karakteristikainställning.

Genom att nyttja UUID:n kan BLE-enheter således kommunicera. Servern berättar vilka tjänster den har och deras karakteristikor är. Klienten i sin tur vet vilka, om någon, är av intresse med hjälp av deras UUID och får även veta vilka handtag servern använder för dem. När så gjorts kan klienten fråga efter värdet från en karakteristika via dess handtag.

3.4 Mirka Diagnostics

Utöver slipmaskinens mjuk- och hårdvara måste även persondatorsidan tas i beaktande. Här hade Mirka utvecklat ett eget program för att hämta diagnostikloggar från slipmaskinen, lämpligt nog kallat Diagnostics. Programmet som var gjort i C# och begränsat till Microsoft Windows hade den tekniska möjligheten att påbörja och avsluta en BLE-anslutning med en DEROS. Persondatorer, speciellt bordsdatorer, har inte alltid något Bluetooth modem. Därför används och förväntas en Bluegiga BLED112-dosa vara inkopplad via USB för att garantera denna förmåga. Dosan behandlas som en virtuell serieport. Diagnostics bruk av dosan bygger på Microsofts System.IO.Ports kombinerat med dosans BGAPI[15] samt Mirkas egen kodbas för att hantera kommunikationen över Bluetooth.

Efter att en anslutning hade etablerats kunde programmet både ta emot information från slipmaskinen, men även skicka sina egna kommandon till den om så krävdes. Att ta emot data krävdes givetvis för att få körningsinformationen och därmed bättre kunna diagnostisera långsiktiga problem med slipmaskinen. Att skicka kommandon å andra sidan hade inte mycket

med loggning att göra. Det enskilt längsta kommandot var ett namngivningskommando som ändrade hur slipmaskinen presenterade sig själv inom Bluetoothramverket och var helt enkelt en mindre förfining. De andra kommandona var endast väldigt korta och var del av ett annat utvecklingsprojekt som gick ut på att styra slipmaskinens motor via Bluetooth. Att sända större mängder data var emellertid varken en påtänkt eller utvecklad förmåga.

4 Planering

Projektet må ha sin grund hos Mirka men någon långtgående planering fanns inte. Det var endast ett vagt framtida utvecklingsmål som handlade om att tillåta uppdatering av deras DEROS-slipmaskiner. Omprogrammering av DEROS:arna var vid den tiden endast möjligt med hjälp av speciella omprogrammerare, en metod som inte lämpade sig utanför utvecklingslabbet. Förutom detta slutmål var endast kommunikationskanalen i form av slipmaskinens BLE-modem satt i sten. Vidare krav på hur projektet skulle genomföras fanns inte. De specifika detaljer kring själva designen och förverkligandet hamnade därmed inom projektets ramar. När det kom till slutgiltiga beslut om designen fanns personal tillgänglig och kravsättningen kunde tas fram i samråd med den.

4.1 Kravbild

Första steget i projektet blev därmed att utforska de begränsningar som arbetet befann sig under och med hjälp av dessa skapa en lista av tydliga krav. Ett av de tydligaste tekniska hindren kretsade kring mikrokontrollerns omprogrammeringsinstruktion: SPM. Denna instruktion kan, olikt de flesta andra, endast utföras från en del av XMEGA:ns programminne, ett 4 kibioktetter stort område som benämndes Boot Loader Section. Det stod klart att denna minnesdel måste tas i bruk av projektet, åtminstone delvis. Arbetet fortskred därför med att analysera hur det bäst skulle nyttjas.

Området kunde användas för vilket ändamål som helst, men dess uttalade ändamål var just för att innehålla startladdare och omprogrammeringskod. Det hade nämligen även en annan fördel; endast kod i denna del av minnet kunde vara aktiv då en SPM-instruktion genomfördes. Utöver detta fanns även andra speciella egenskaper. Mikrokontrollern kunde startas i området istället för den normala minnesplatsen och området kunde även ha sina egna skriv- och läsrättigheter. I termer av minnesadressering var dess plats i slutet av flashminnet och fortfarande helt tomt, då en stor del av det total programminnet fortfarande var outnyttjat av Mirka. Tidigt under arbetets gång kom därför idén att begränsa projektet till denna startladdarsektion i så stor utsträckning som möjligt.

4.1.1 Fristående kod

Det fanns goda skäl för att vilja genomföra idén. En tät koppling mellan den normala styrkoden och den nya koden skulle innebära att Mirka i framtiden alltid måste beakta hur förändringar av styrkoden påverkar omprogrammeringskoden. Det här vore problematiskt då projektet utförs av en källa utanför Mirka och de därmed inte automatiskt får en förståelse för hur koden fungerar. En grundläggande tanke kring projektet blev därför att projektkoden kunde hanteras som nästintill en statisk modul som satts in i programminnet.

Därför strävades det efter en så stor funktionell separation mellan deras nuvarande kodbas som möjligt. Utvecklare hos Mirka skulle inte behöva någon närmare kännedom om hur omprogrammeringskoden fungerar. De resterande 32 kibioktetter av programmeringsminne skulle vara i ett kontinuerligt stycke och det skulle vara möjligt att åtnjuta allt detta minne utan desto vidare tanke på omprogrammeraren. Vidare så skulle samma omprogrammeringskod kunna användas i nya produkter såvida modemet var detsamma och mikrokontrollern från XMEGA A4U-familjen.

För att uppnå denna avskiljning måste mikrokontrollern vara i ett nollställt läge då omprogrammeringskoden startas, då den inte förutsätts ha någon kunskap om styrningen av slipmaskinen. Om den inte är nollställd då detta sker kunde annars någon utport vara aktiv, till exempel till motorn. Om så var fallet fanns möjligheten att slipmaskinen i värsta fall blev en säkerhetsfara. Potentialen fanns även för att det skulle skada maskinen på något sätt. En konsekvens av detta är att slipmaskinen givetvis inte kan användas under omprogrammeringskedet. Detta ter sig acceptabelt då det handlar om en arbetsmaskin som rimligtvis sällan uppdateras. Den har heller inte någon livsviktig funktion som kräver att den är i konstant användning.

Frågan om nollställningen var ett annat skäl till att vilja begränsa projektet till startladdarsektionen. XMEGA:n kan, som nämndes tidigare, konfigureras för att starta i detta minnesområde. En omstart är ett enkelt sätt att säkerställa att XMEGA:n befinner sig i ett nollställt läge. Alternativet medför två krångligheter. Till att börja med innebär det en hel del extra kod i styrkoden, vilket går emot separationsprincipen. Det kräver också att man väldigt noggrant beaktar i vilket läge styrkoden befinner sig, stänger av signaler därefter och då detta gjorts hoppar till det korrekta minnesområdet. Emedan detta är fullt möjligt så leder det till mer arbete och ökar utrymmet för misstag.

4.1.2 Kommunikationkodens utrymmeskrav

En av följderna av separationen från styrkoden var att omprogrammeringskoden inte längre hade det normala inbyggda programmets kommunikationskod till sitt förfogande, den måste istället innehålla en egen kopia av densamma. Mirka använde sig av ett Bluetooth-modem från Nordic Semiconductor och de använde även den kod som gjorts tillgängligt från nämnda företaget för att kommunicera med chippet. Här fanns emellertid ett stort orosmoln; koden tog många oktetter av minnesutrymme i dess dåvarande form och var alltför stort för startladdarområdet. Före ytterligare kravsättning blev det därmed väsentligt att fastställa huruvida antingen denna kod kunde reduceras eller ny kommunikationskod produceras som rymdes inom 4 kibioktetter med en viss marginal. Annars måste projektplanen ändras därefter.

Omprogrammeringen i sig hade inte några större krav på kommunikationsmöjligheter. En fullt fungerande, om än spartansk, omprogrammerare krävde endast möjligheten att starta en Bluetooth-uppkoppling samt att ta emot datapaket. Att ha en så enkel omprogrammerare tycktes dock onödigt användarovänligt. Genom att också inkludera sändning av datapaket från programmet samt möjligheten att avsluta en uppkoppling kan en mycket mer fulländad omprogrammerare skapas. En genomgång av kommunikationskoden gjordes med detta i beaktande. Det visade sig att en tillräcklig reduktion potentiellt var möjlig, emellertid skulle endast ett fullt försök på detta säkerställa huruvida så var fallet och därför utfördes en långtskridande reduktion redan nu. Arbetet beskrivs mer grundligt i del 5.1.2 av avhandlingen.

4.2 Slutfas av kravsättningen

Då arbetet med kommunikationskoden hade slutförts fanns möjligheten att mer definitivt, om än inte slutgiltigt, fixera omprogrammeringskodens plats till startladdaravsnittet av programminnet. I samband med detta kom frågan om omprogrammeringens omfattning på tapeten. Behövdes förmågan att modifiera startladdaren också, eller var det tillräckligt att den endast var förmögen att ändra kod i applikationsområdet? Det visade sig att endast uppdatering av applikationsområdet efterfrågades. Således blev de expanderade kravlistan följande:

1. Maximalt 4 kibioktetter av kod, Nordic Semiconductors kommunikationskod inräknat.

2. Kör omprogrammeringskoden endast då mikrokontrollern är i ett nollställt läge.
3. Starta en Bluetooth-uppkoppling genom nRF8001 modemet.
4. Motta ny programmeringskod via Bluetooth-uppkopplingen.
5. Använd den emottagna informationen för att uppdatera programmeringsminnet.

4.2.1 Påtänkta förbättringar

Förutom listan på strikta krav gjordes även en lista med begärliga förbättringar. Dessa ansågs inte strikt nödvändiga, men om utrymme fanns så gav de en vägvisning gällande vidareutveckling. Den viktigaste av dem är den påtvingade korrekthetskontrollen av programminnet, utan vilken man inte får återgå till den normala styrkoden. Avsaknaden av en sådan typ av säkerhetskontroll leder till att slipmaskinen i värsta fall börjar köra en korrumperad och därmed rent av farlig styrkod. På grund av detta är det nästintill ett krav att ha en sådan kontroll, snarare än en förbättring.

En annan viktig förbättring, om en inte så kritisk, var möjligheten att alltid starta omprogrammeringskoden oberoende av vilken sorts programkod det fanns i resten av programminnet. En sådan förfining hindrar slipmaskinen från att permanent låsa sig och bli oanvändbar om en felaktig styrkod på något sätt satts in. Den kan alltid skrivas över med fungerande kod.

Att höja säkerheten gentemot obehöriga element var också med på listan. Att tillåta en utomstående part att använda sig av omprogrammeraren för att installera sin egna programvara i slipmaskinen ter sig kanske som något rent av positivt. Samtidigt är det dock även en attackvektor som en fientligt sinnad person kan utnyttja för att antingen förstöra slipmaskinen utan desto vidare spår, eller skada någon annan. Emedan detta troligtvis inte är någon större fara så ger kryptering utöver detta också skydd åt affärshemligheter. Ny styrkod kan distribueras genom öppna kanaler för uppdatering av slipmaskiner och trots detta kan inte någon annan läsa dem utanför Mirkas utvecklingslabb.

Sist på listan är småsaker som förhöjer användarerfarenheten men som annars är tämligen oviktiga. Följaktligen blev de tillagda i kravlistan i denna ordning:

6. Kontrollera att ny styrkod oförvanskad skrivits till programminnet före

slipmaskinen tillåts använda styrkoden.
7. Säkerställ att omprogrammeraren alltid kan nås oberoende av innehållet i resten av programminnet.
8. Förse slipmaskinen med ett mått av säkerhet genom att förhindra utomstående från att installera programvara som inte kommer från Mirka.
9. Låt omprogrammeraren använda krypterad kod av affärshemlighetsskäl.
10. Kommunicera resultatet av en korrekthetskontroll av styrkoden.
11. Signalera att slipmaskinen befinner sig i sitt omprogrammeringsläge.

4.2.2 Bluetooth-motpart

Förutom slipmaskinen måste det även finnas en motpart i Bluetooth-anslutningen som kan använda sig av omprogrammeringsmöjligheten. För ändamålet föll valet naturligt på Mirkas Diagnostics program. Programmet var redan byggt för att kommunicera med slipmaskinen och var kapabelt att utföra de nödvändiga Bluetooth-operationerna. Programmets omfattning kunde enkelt expanderas med en funktion för att skicka ny kod till slipmaskinen. Förövrigt användes programmet redan vid Mirkas underhållsplatser och av utvecklingslaget, denna förändring skulle endast leda till en minimal störning jämfört med att skapa ett helt nytt program . Oavsett så sattes det inte några ytterligare krav från denna sida.

4.3 Design av startladdaren

Med de utarbetade kraven som grund var det möjligt att påbörja arbetet kring en flödesplan. Som tidigare nämndes var planen att använda Diagnostics för att hantera överföringen av ny kod till slipmaskinen. Det föll sig därmed naturligt att hela processen skulle skötas via samma program utan att behöva röra någonting annat, förutom att givetvis koppla i och sätta på slipmaskinen. Utgångsläget vore då att det normala inbyggda programmet på slipmaskinens sida är startat och aktivt. Ett enkelt Bluetooth-kommando påbörjar därefter alltihop.

Nackdelen med detta förslag är att det ställer krav på den normala styrkoden, om de inte uppfylls så blir det omöjligt att göra en omprogrammering. Även om det inte var strikt nödvändigt var ett önskemål att omprogrammerings-

koden alltid skulle kunna nås; en alternativ väg måste finnas. Utvägen är att alltid börja i startladdarområdet och köra en del av dess kod. Först efteråt får slipmaskinen fortsätta vidare. Detta val har också sin baksida. Slipmaskinen är trots allt i första hand ett arbetsverktyg, att skapa någon längre fördröjning mellan att slipmaskinen sätts på och att den kan användas kan inte ses som acceptabelt. Således kan maskinen inte heller lämnas väntande på en Bluetooth-anslutning och efterföljande kommando på detta vis.

Följaktligen bestämdes det att två varianter skulle implementeras. Den mer användarvänliga metoden använder sig endast av Diagnostics programmet, där allt kontrolleras via Bluetooth-kommandon men förlitar sig på den normala styrkoden. Reservplanen å andra sidan kräver vidare hantering av slipmaskinen först. Till att börja med skulle mikrokontrollerna alltid starta i startladdarområdet, men inte lämna där någon längre tid. För att så snabbt som möjligt kunna fortsätta vidare så skulle den helt enkelt kontrollera vilka knappar som var nedtryckta under startögonblicket. Den går då endast längre in i omprogrammeringskoden om rätt kombination är nedtryckt, annars hoppar den vidare till det normala programmet.

Speciellt med tanke på den andra varianten blir det nu viktigt att informera användaren om att omprogrammeringsläget är aktivt. Maskinerna i DEROS-familjen har ett par lysdioder för att indikera status eller problem. Med deras hjälp går det att indikera att slipmaskinen inte längre befinner sig i ett normalt läge. En normal användare kan då se att någonting gått snett om de i misstag startat specialläget. Någon som ämnat använda omprogrammeraren å andra sidan kan vara säker på att det korrekta läget startats och kan använda det för felsökning. Om de inte får kontakt med slipmaskinen via Bluetooth efter ett tag så beror det troligtvis på något fel kring Bluetooth-sidan. Igångsättningen av Bluetooth-modemet är nämligen det logiska steget strax efteråt. Då detta gjorts går maskinen in i ett vänteläge; ingenting mer finns att göra utan en Bluetooth-anslutning.

4.3.1 Bluetooth-anslutning

Diagnostics måste senast nu börja ha en aktiv roll. Programmet måst köras på en dator som kan använda sig av Bluetooth Smart. På grund av den korta räckvidden borde datorn och slipmaskinen inte befinna sig allt för lång borta från varandra, högst ett antal meter beroende på vilka hinder som befinner sig emellan dem. Mirka hade gjort programmet så att det hade en konstant uppdaterande lista av närliggande slipmaskiner. Då den korrekta

slipmaskinen dykt upp i listan var det möjligt att starta en uppdatering. Om omprogrammeraren startats manuellt skulle den dyka upp på denna lista och användaren hade möjligheten att påbörja en anslutning. I det fallet att Diagnostics hade kommenderat omprogrammeringsläget gjordes detta anslutningsförsök automatiskt.

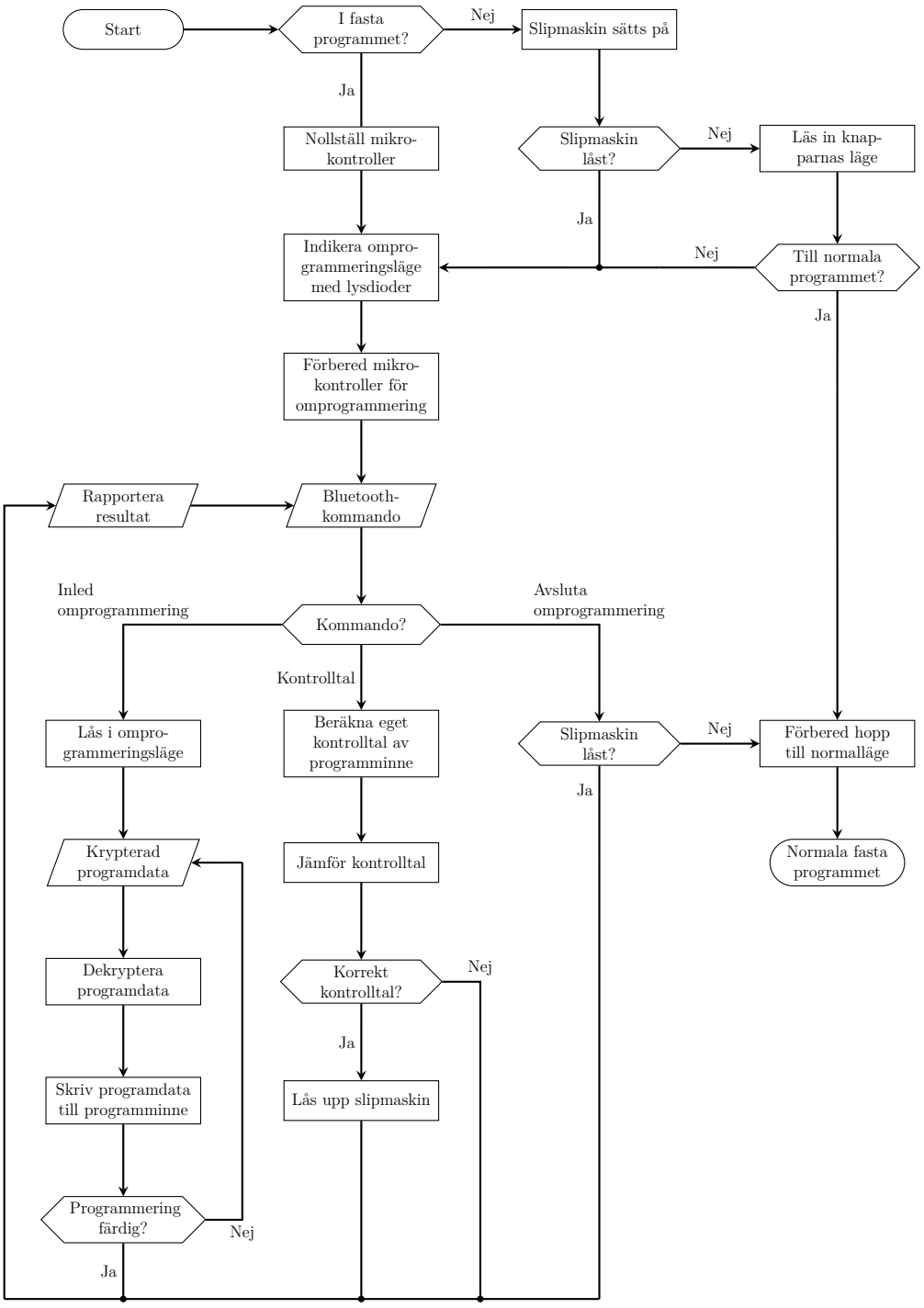
Uppdateringsprocessen påbörjas från Diagnostics sida genom att användaren väljer en lämplig krypterad fil med det fasta programmet. Partiella uppdateringar var inte i kravbilden, hela det befintliga inbyggda programmet byttes ut på en gång och endast 32 kibioktetter stora avbilder sänds därför. När valet gjorts indikerar Diagnostic åt slipmaskinen att en omprogrammeringssession skall startas, vilket låser slipmaskinen i detta läge. Från denna punkt var det inte längre möjligt att återgå till det normala fasta programmet, även om uppdatering avbröts direkt. En omstart av slipmaskinen skulle endast leda tillbaka till omprogrammeraren. När det är avklarat påbörjas en dataöverföring strax efteråt.

Gällande vidare datahantering fanns en viss begränsning då det gällde datainnehållets storlek. Förutom flashminnet samt mottagning- och sändningsregistren fanns det endast 5 kibioktetter av lagringskapacitet i mikrokontrollern; 4 kibioktetter av arbetsminne och 1 kibioktett av icke-flyktigt minne. Därmed fanns det inte någon möjlighet att först ta emot all data och efteråt påbörja omprogrammeringen. Mottagningen, dekrypteringen samt skrivprocessen måste ske delvis samtidigt. Efter att hela dataöverföringen genomförts började Diagnostics vänta på slipmaskinens bekräftelse att den sista omprogrammeringen var färdig. Denna försäkran skickas från DEROS:en då en sida av applikationsminnet bytts ut.

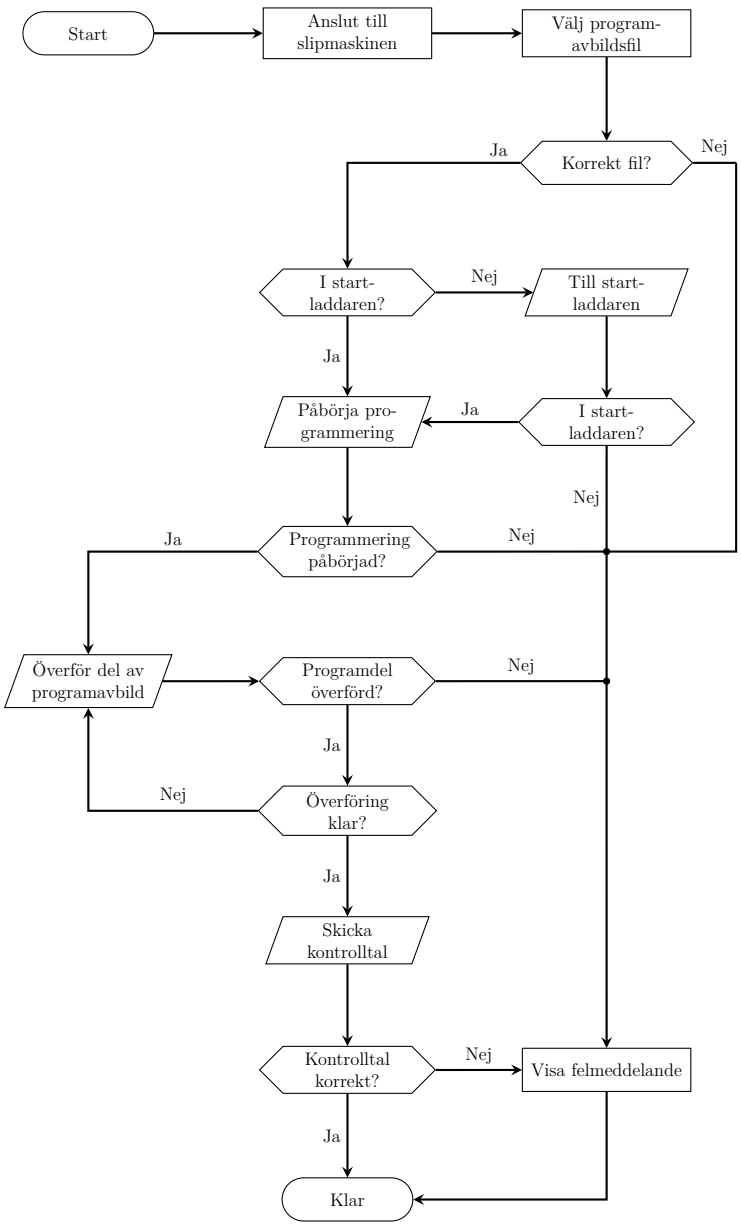
För att återgå till ett normalt läge krävdes nu att slipmaskinen återigen låses upp. Om den normala proceduren följs skickas först ett kontrolltal från Diagnostics till slipmaskinen direkt efter den bekräftat att omprogrammering är färdig. Slipmaskinen tar emot kontrolltalet och gör sin egen beräkning av ett kontrollvärde baserat på vad som verkligen skrivits till programminnet. Är de två värdena lika kan slipmaskinen återgå till sitt upplåsta läge, annars fortsätter den att vara låst. Oberoende sänds ett svar tillbaka till Diagnostics beträffande resultatet av kontrollen. Ett positivt svar från slipmaskinen leder till att Diagnostics befaller ett slut på omprogrammeringsläget och efteråt att startladdaren avslutas. Om inget fel inträffar börjar slipmaskinen därefter köra den nyinstallerade programvaran.

En avbruten process leder till ett felmeddelande från Diagnostics. I det låsta

läget accepterar DEROS:en endast två saker: en ny dataöverföring eller ett öppningskommando. Emellertid är öppningskommandot inte tillgängligt från Diagnostics rakt av, då det kräver ett kontrolltal som är okänt om ingen diskavbild finns tillgänglig. Därmed kan det endast skickas i samband med att en ny programvara sänts. Om maskinen är olåst accepteras utöver dataöverföring även ett omstartskommando. Det slutgiltiga flödesschemat i designstadiet ses i figur 4.1 och 4.2 för startladdaren respektive Diagnostics. En muntlig framställning av denna design blev accepterad av Mirka.



Figur 4.1: Presumtiv flödesplan för slipmaskinen.



Figur 4.2: Presumtiv flödesplan för Diagnostics.

5 Genomförande

Utvecklingsmiljön för projektet var AVR Studio, stödd av en extern Atmel-ICE omprogrammerare. Ett kretskort identiskt med det i färdiga DEROS maskiner användes också, emellertid med den modifikationen att kontakter för den externa omprogrammeraren var fastlödda i mönsterkortet. Det som saknades var själva chassit med motorn, något som inte borde inverka på omprogrammeringskoden. En komplett slipmaskin användes dock i slutskedet för att verifiera den nya koden i slutet av projektet. Tack vare Atmel-ICE-programmerarens felsökningsegenskaper samt användningen av ett fullständigt kretskort var kodningsprocessen relativt enkel. En ny kod kunde köras direkt och därmed var det möjligt att bekräfta att den fungerade som tänkt.

5.1 Slipmaskinens programvara

5.1.1 Övergång till och låsning i omprogrammeringsläge

Implementationsfasen följde flödesschemat i hög grad. Därmed blev den första uppgiften att bestämma exakt hur man skulle gå in i det djupare omprogrammeringsläget och hur den skulle låsas i detta läget om så krävs. Genom en säkringsoktett ändrades slipmaskinens startpunk till startladdarområdet av minnet. Därmed låg det initiala beteendet väl inom projektets ramar. För att garantera att omprogrammeraren alltid var tillgänglig inlästes läget för två knappar under startskedet. Knapparna i fråga var de som vanligen kontrollerade hastigheten på slipmaskinsmotorn. Om de var nedtryckta fortsatte koden djupare in i omprogrammeraren.

Den alternativa metoden påbörjades med ett Bluetooth-kommando från Diagnostics. Att hoppa direkt från den normala styrkoden till omprogrammeraren hade förkastats redan i kravbildsskedet. Detta då det fanns en risk för mycket extra arbete eftersom mikrokontroller måste sättas i ett neutralt läge innan hoppet gjordes. En omstart krävdes. Att omstarten gjordes genom mjukvaran är en viktig detalj. XMEGA:n har nämligen ett statusregister som håller reda på orsaken till en omstart. I den valda lösningen lästes statusregistret och om mikrokontrollern hade startats om på grund av ett mjukvarukommando fortsatte koden vidare in i startladdaren. Omstarten gjordes endast efter att Mirkas styrkod använts för att stanna. En kort fördröjning sattes även in mellan att motorn beodrats att stanna och själva omstarten.

Slipmaskinen skulle även alltid tvingas in i omprogrammeraren efter en omprogrammering påbörjats men inte korrekt slutförts. Beträffande låsningsstatusens lagringsplats krävdes ett icke-flyktigt minne och då fanns det två val. Antingen användes flashminnet eller EEPROM:en. Av de två valen finns det flera negativa sidor med programminnet. Att använda det för att spara en variabel skulle reducera utrymmet för annan kod. Därtill är det endast möjligt att ändra variabel med kod som befinner sig i startladdarområdet. När man skriver till programminnet så är det även omöjligt att endast manipulera någon enskild oktett, istället måste man skriva en hel minnessida. Valet föll därmed på EEPROM:en, en oktett av detta minne togs i besittning för att hantera låsningsvariabeln. Om den antog ett värde annat än 0 och den rätta knappkombinationen inte var nedtryck hoppade koden vidare till startpunkten för applikationsdelen i flashminnet. Om värdet var 0 gick koden vidare i startladdaren.

Att använda statusregistren som en väg in i omprogrammeraren har sin baksida, den bryter nämligen mot separationsprincipen. Det normala inbyggda programmet kan inte använda sig av mjukvaruomstarter för andra ändamål. Ett nästan fullgott alternativ finns också om man tar situationen som råder kring omstarten till omprogrammeraren i beaktande. Enligt planen sker detta endast när en uppdatering är förberedd från Diagnostics sida. Således skulle alltid en dataöverföring ske med dess associerade låsning direkt efter omstarten, såvida inga problem uppstår. Istället för att använda statusregistret vore det möjligt att utnyttja låsningsvariabeln. En annan lösning är att helt enkelt skapa en ny variabel i EEPROM:en, någon sorts icke-flyktigt minne krävs trots allt.

Om omprogrammeraren startades efter de begynnande kontrollerna nollställdes statusregistret först. Sedan sattes avbrottsvektorer till startladdarområdet, LED lampor tändes, klockfrekvensen ökades till 32 MHz och så vidare. Efter att de grundläggande inställningarna gjorts fanns en slutgiltig uppgift: att sätta igång Bluetooth-modemet.

5.1.2 Modifikation av Nordic Semiconductors kod

Kontrollen av modemet baserade sig i hög grad på kod från Nordic Semiconductor tillsammans med programmet nRFgo Studio. Programmet kan generera det konfigurationsmeddelande som skickas till modemet för att ställa in GATT-tjänster. Majoriteten av arbetet kring koden gjordes egentligen redan under kravsättningen då kodens storlek var en stor identifierad fall-

grop, vilket redan nämnts i 4.1.2. Oförändrad skulle koden inte rymmas i startladdarsektionen. Här handlade det om att vaska fram de bitar som var väsentliga för projektets del och ta bort allt annat. Fokus började kring de olika ACI-kommandon till nRF8001-modemet som fanns tillgängliga. Av dessa var många irrelevanta för projektets vidkommande. Här kan nämnas testkommandon, temperaturkontroll av chippet samt allmänt anrop.

Proceduren som användes för att avlägsna onödig kod var en kombination av två metoder. Ett första pass gjordes för att manuellt identifiera och avlägsna funktioner för de kommandon som inte behövdes. När detta gjorts sattes loggfunktioner in i koden och ett testprogram kördes där endast de nödvändiga Bluetooth-kommandona skickades och mottogs. Med hjälp av loggarna från denna metod kunde fler onåbara kodstycken och variabler hittas. Likväl verifierades det att de var onödiga genom att läsa koden före de ströks.

Fastän metoderna gav goda resultat var de ändå otillräckliga. Ytterligare minskningar krävdes för att den egna omprogrammeringskoden också skulle få plats i startladdardelen av minnet. Därmed började en tillplattning av koden. Detta skedde genom att finna funktionsanrop som inte behövdes. Först lokaliserades de platser där funktionsanrop gjordes och därefter vad de utförde. Det visade sig att ett flertal funktioner endast kallades från enskilda platser i koden. Följaktligen var det möjligt att eliminera funktionsanropet genom att flytta funktionens kod till den plats där anropet gjorts, med lämpliga modifikationer för att ta variablers räckvidd i beaktande. Borttagningen av dessa funktionsanrop samt de andra småjusteringarna som gjordes sparade ytterligare ett antal oktetter av minne. Det kombinerade resultat blev en kod under 4 kibioktetter. I det utvecklingsstadium detta arbete utfördes fanns det ingen garanti att utrymmet skulle räcka till. Ett försök var emellertid värt att göra för att få en uppfattning om hur mycket minne den resterande omprogrammeringskoden skulle kräva.

5.1.3 Kommunikationsrör

Endast en karakteristika användes för all relevant kommunikation. Genom ett ACI-rör får mikrokontrollern reda på när data anländer. Av de 20 oktetter som kan överföras per kommunikationsintervall är det första alltid satt åsido åt en kommandooktett vars värde är länkat till en viss uppgift i enlighet med tabell 5.1. Frånsett modemets hantering av den fysiska kommunikationslänken besvaras endast tre kommandon av omprogrammeringskoden. Kontrolltalskontrollen får svar över huruvida kontrollen var giltig eller

Tabell 5.1: Kommandooktetter

Kommandooktett	Kommando	Nyttolast	
0xB0	Omstart		
0xB1	Kontrolltalskontroll	4 oktetter	Kontrolltal
0xB2	Ändra programmeringsläge	1 oktett	På och av
0xB3	Rensa skrivbuffert		
0xB4	Fyll skrivbuffert	17 oktetter	Programdata
0xB5	Skrivbuffert till programminne	2 oktetter	Sidadress

inte. Porgrammeringslägesändringen svaras med läget som slipmaskinen nu befinner sig i; den beordrande enheten måste själv kontrollera att svaret överensstämmer med vad som förväntades. Slutligen får omstartskommandot ett svar om en omstart till ny styrkod inte är möjlig. Dessa svar utförs genom att modifiera det karakteristiska värdet som lagras lokalt i modemet varifrån de efteråt kan läsas via Bluetooth-anslutningen.

5.1.4 Uppdateringsprocessen

Med kommunikationsbiten i skick var det dags att bearbeta den inkommande dataströmmen. Valet av kryptering kom först. Egen implementering av en krypteringsmetod var möjlig, emellertid skulle det kräva extra utrymme. XMEGA:n däremot hade inbyggda krypteringsmetoder, DES och AES, vars användning inte tog flera rader av kod i anspråk. Enligt den kunskap som fanns tillgänglig var AES kryptering mångfalt mer säker och valet fall därför på den. Som konsekvens behövdes 16 oktetter av data för varje dekrypteringscykel. Den 128 bitars långa AES-nyckeln lagrades som en variabel i startladdarens minnesområde

Efter dekryptering måste informationen skrivas till programminnet i något skede. Som nämnts tidigare fanns det p.g.a. bristande lagringsutrymme inte någon möjlighet att vänta tills hela dataöverföringen var klar. Att mikrokontrollern endast var kapabel av att göra sidstora skrivningar var också en begränsning. En XMEGA32A4U:s applikationsområde är 128 sidor stor och hade därmed 128 ord, eller 256 oktetter, per sida. Det vore givetvis tänkbart att göra mindre skrivningar genom att först läsa in en sida från programminnet, modifiera den del som kommit in och sedan skriva tillbaka sidan. Det innebär dock att man utsätter flashminnet för många gånger fler skrivcykler. Av det skälet undveks en dylik lösning.

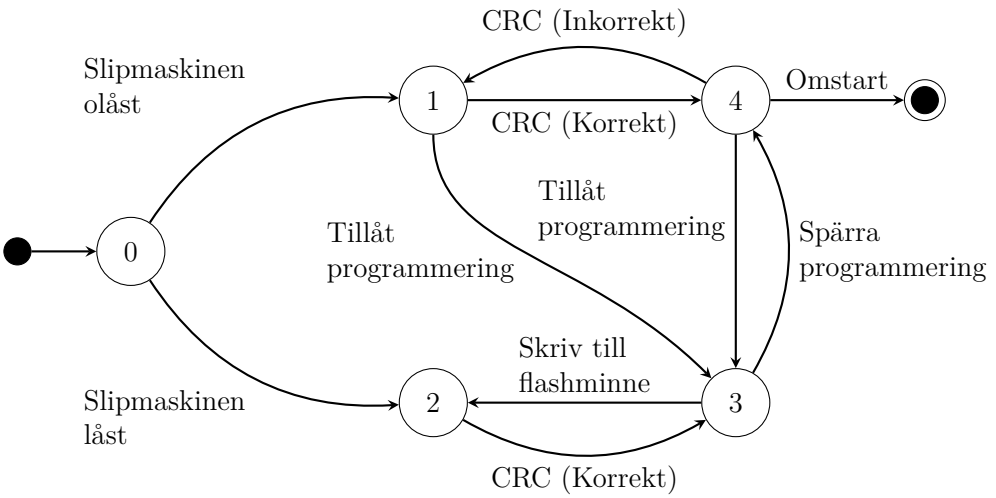
5.1.5 Mellanlagring av emottagen information

Situationen var med andra ord att endast 20 oktetter stora paket av det totalt 32 kibioktetter stora programmet anlände per kommunikationsfönster. Av dessa var det högst 16 oktetter som gick att dekryptera medan hela 256 oktetter krävdes före en fullgod skrivning gjordes. Det fanns ett klart behov av mellanlager. Tacksamt nog hade AES modulen sitt egna nyckel- och tillståndsminne och det fanns även en undansatt buffert för skrivningar till flashminnet. Således laddades nyckeln in direkt i AES modulen då en dataöverföring påbörjats. Allteftersom data anlände flyttades den först över till AES modulens tillståndsminne och dekrypterades. Den dekrypterade information flyttas därefter till skrivbufferten.

Istället för att fylla datapaketen användes endast 17 oktetter. Den första oktetten användes för att bestämma vilken adress i skrivbufferten innehållet hör till, de resterande 16 oktetterna är en del av den krypterade programavbilden. Lösningen valdes av läsbarhetsskäl. Med 16 oktetter av data blir det möjligt att ta emot och dekryptera allting på en gång för att sedan flytta det direkt till skrivbufferten. Det blir därmed också lättare att följa händelseförloppet i koden för någon som inte är insatt i den. En annan aspekt som togs i beaktande var utrymmeskraven för en sådan kod. Vinsten i dataöverföringshastighet som ett 20 oktetter stor paket skulle inbringa är inte tillräckligt stor för att göra det värt mödan.

Med en fylld skrivbuffert kan innehållet skrivas till programminnet. Detta sker genom ett nytt kommando vilket även bestämmer sidan av minnet det skrivs till. Sidan raderas först med samma instruktion som utför skrivningen. En fullständig omprogrammering innebär därmed 2048 datapaket med deras dekryptering samt 128 skrivkommandon. Eftersom kommunikationskoden baserar sig på avbrott var det värt att beräkna om nya kommandon kunde anlända innan det föregående hade behandlats.

XMEGA:n är inställd till en klockfrekvens av 32 MHz då omprogrammering sker och det lägsta kommunikationsintervallet är 7,5 ms. 240000 klockcykler hinner ske under denna tid. Även den långsammaste utnyttjade processen som beskrivs i mikrokontrollerns manual kan göras hundratals gånger under denna tidsrymd. Som ett belysande exempel tar dekrypteringen av 16 oktetter via AES hela 375 klockcyklar i anspråk. AES modulen kan hantera ungefär 1,3 mebioktetter per sekund, i praktiken troligtvis något mindre då detta inte inkluderar tiden det tar att fylla . Dataflödet via Bluetooth å andra sidan är som högst runt 2,1 kibioktetter per sekund via Bluetooth, en enorm



Figur 5.1: Tillståndsdiagram för startladdaren.

skillnad. Det finns emellertid instruktioner vars exekveringstid inte är exakt fastlagd i XMEGA-manualen. Skrivoperationen till minnet är en av dem, endast ett typvärde på 8 ms för att radera och skriva en sida av flashminne. Genom att direkt använda typvärdet är kapacitet runt 1,3 mebioktetter per sekund vilket vida överträffar överföringshastigheten. Experiment visade också att skrivprocessen inte var någon flaskhals. Dataöverföringen blev den begränsande faktorn för uppdateringshastigheten.

5.1.6 Upplåsning och kontrolltal

Den sista biten av pusslet var upplåsningen av slipmaskinen med hjälp av korrekthetskontrollen. Också den kan enkelt göras av mikrokontrollern. Funktionalitet fanns för att göra både CRC-16 och CRC-32 beräkningar på innehållet av hela eller valda delar av programminnet. Kontrollen påbörjades då ett Bluetooth-meddelande innehållande ett kontrolltal anlände. Därefter startades en CRC-32 beräkning av applikationsminnet. När slutresultatet fåtts jämfördes de två talen med varandra och om de var lika tilläts upplåsning. Ett skilt kommando måste därefter skickas för att EEPROM:ens låsvariabel verkligen skulle ändras. Oberoende av vad resultat blev skickades ett meddelande med utgången tillbaka via Bluetooth för båda kommandona.

Figur 5.1 visar de tillstånd som omprogrammeringskoden i startladdaren kan befinna sig i. Notera att här beaktas endast kontrolltalet och låsningen, inte

några andra variabler eller buffertar. Då slipmaskinen är låst kommer den alltså alltid att hoppa till tillstånd 2 och för att komma därifrån krävs ett korrekt kontrolltal. Omprogrammeringskommandon accepteras endast i tillstånd 2 och 3, men i tillstånd 3 är det möjligt att låsa upp slipmaskinen utan något kontrolltal eftersom ingen förändring av programminnet ännu gjorts i det skedet.

5.2 Diagnostics på andra sidan anslutningen

Beträffande Diagnostics handlade det mesta av arbetet om tillägg till koden, orginalkoden lämnades i stora drag oförändrad. Gränssnitten måste givetvis ändras något, men förutom det fanns även ett problem med kommunikationskoden som måste bearbetas. Handtagen för de olika attributen som används av nRF8001 ställs in under dess konfigurationsfas och detta var något den existerande Diagnostics-koden använde sig av. Den förväntade sig att de alltid var desamma. Startladdaren hade dock färre attribut än den normala styrkoden vilket ledde till att hanteringsnumren även ändrades.

Här bör poängteras att konfigurationsdatan som skickades till modemet genererades av ett externt program från Nordic Semiconductor där man inte direkt fick ställa in hanteringsnummer. Det skulle vara möjligt att analysera utdatan från detta program och därmed kanske även göra en manuell justering så att hanteringsnumren för startladdaren var desamma som för det normala fasta programmet. Den lösning som valdes var emellertid att ändra Diagnostics så att det istället tog reda på de rätta hanteringsnumren från slipmaskinen. Då en anslutning påbörjades efterfrågades numren för attributen genom deras UUID:n. Således inverkade inte längre förändringar i hanteringsnumret alls. Detta är även mer i linje med hur det är tänkt att UUIDen och hanteringsnummer skall hanteras i Bluetooth Smart-protokollet.

5.2.1 Gränssnitt

Få förändringar gjordes på det existerande gränssnittet. Huvudsakligen handlade det endast om ett tillägg av ett nytt menyalternativ. Alternativet blev tillgängligt då programmet hade kontakt med en slipmaskin. När det valdes öppnades ett filväljningsfönster varigenom en Intel HEX-fil[7] kunde väljas. När så hade gjorts doldes det gamla gränssnittet helt så att man inte skulle kunna t.ex. avsluta en anslutning under en pågående omprogrammering. I

Tabell 5.2: Exempel-rad i en Intel HEX-fil

	Datalängd	Adress	Datatyp	Data	Kontrolltal
:	0A	0000	00	FFFFFFFFFFFFFFFFFFFF	00

samband med döljandet visades dock ett nytt fönster; ett annat tillskott till gränssnittet.

Efter att en fil valts hade inte slutanvändaren längre någon inverkan på uppdateringsprocessen, allt skedde automatiskt. Situationen kunde följas genom förloppsindikatorn. När processen avslutades visades sen ett extrafönster om omprogrammeringen lyckats eller om något fel uppstått. Meddelandet kan sen bekräftas och om omprogrammeringen slutfördes korrekt befinner sig Diagnostics i utgångsläget; ansluten till slipmaskinen med det normala gränssnittet, förhoppningsvis med en ny version av det fasta programmet i DEROS:en.

5.2.2 Programavbild

Den förväntade filen för en uppdatering är en Intel HEX-fil[4] av det kompilerade fasta programmet. Ett val som gjorts eftersom sådana filer fanns att få direkt från AVR Studio. Inga strikta kontroller gjordes av filen, varken av dess innehåll eller dess storlek. Slutresultatet tvingades dock till att vara minst 32 kibioktetter stort. Om filen är av det förväntade formatet bör den emellertid ha rader av typen som ses i tabell 5.2.

Filen har ett kolon som starttecken och därefter följer ett antal tecken med olika sorters information kodade som oktetter i hexformat. Utöver vad som syns i tabell 5.2 avslutas också raderna med två tecken för radbyte. För projektets ändamål var enbart rader med en datatyp av 0x00, koden för datafält, av intresse. Endast dessa behandlas därför av programmet då en fil läses in.

Uppdateringsprocessen var helt sekventiell och om någon del av den misslyckades totalt avbröts sekvensen direkt och måste därefter manuellt påbörjas på nytt. Endast under själva dataöverföringsdelen fanns ett mått av tolerans. Figur 5.2 visar denna del av förloppet. Om ett meddelande här inte verkade ha kommit fram gjordes nya försök ett begränsat antal gånger. Figur 5.3 och figur 5.4 visar det verkliga flödesschemat för startladdaren vilket kan

[4]Den slutgiltiga distributionsmetoden ligger utanför projektets ramar

jämföras med den ursprungliga planen i figur 4.1.

5.3 Test av den nya startladdaren

Alla tänkbara testscenarier blev inte genomförda under projektfasen. Därtill blev de tester som gjordes utförda endast ett begränsat antal gånger. Testerna fokuserade på vad som antogs vara typiska felkällor under normala förhållanden.

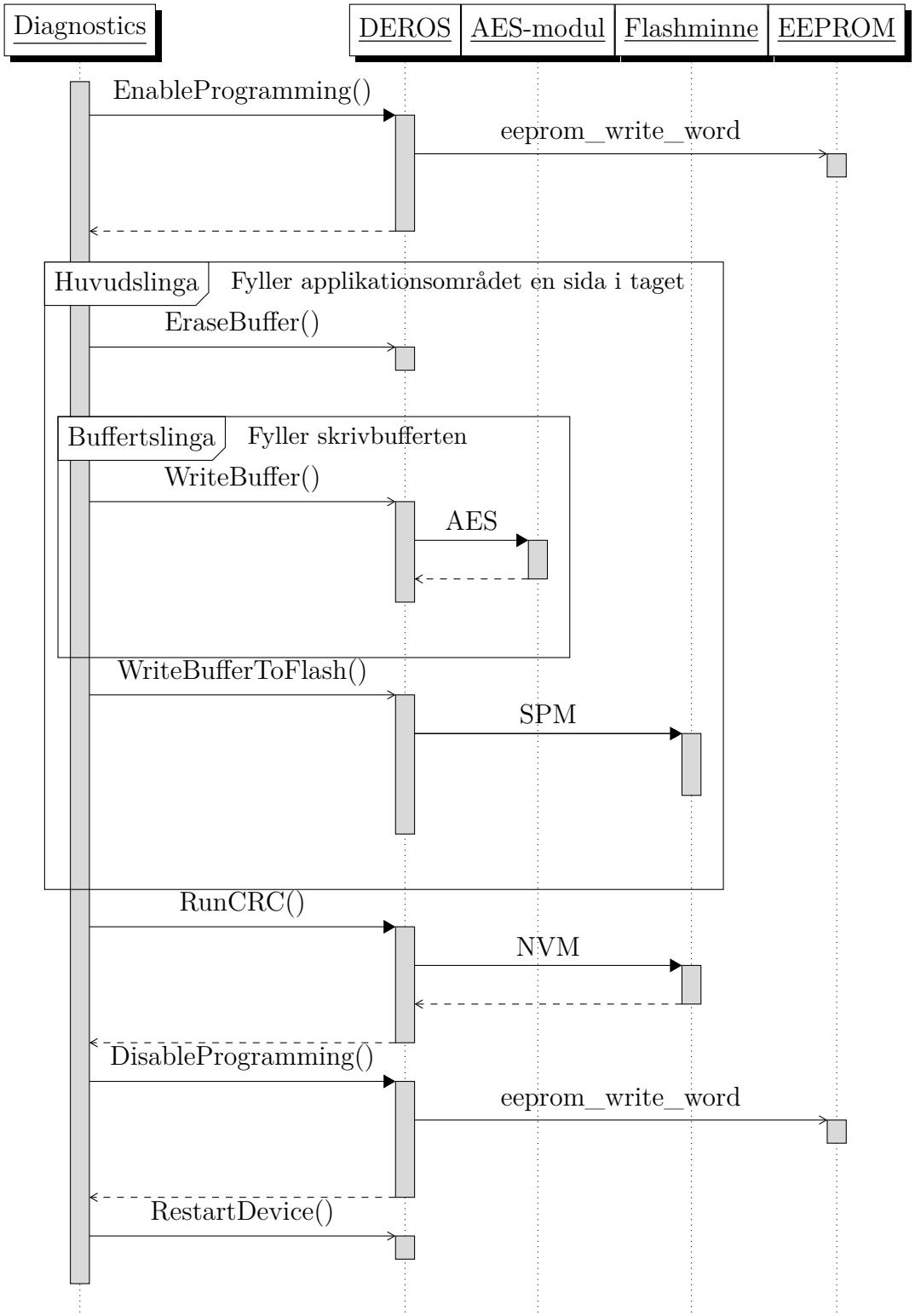
- Test av beteendet då Diagnostics kommenderar påbörjandet av en uppdatering samtidigt som slipmaskinsmotorn körs.
- Test av beteendet då en omprogrammering lämnas oavslutad på grund av strömförlust.
- Test av beteendet då en omprogrammering lämnas oavslutad på grund av en avbruten anslutning.
- Test av beteendet då en uppdatering blir förvanskad.
- Test av beteendet då applikationssektionen är tom.

5.3.1 Det normala inbyggda programmet spelar roll

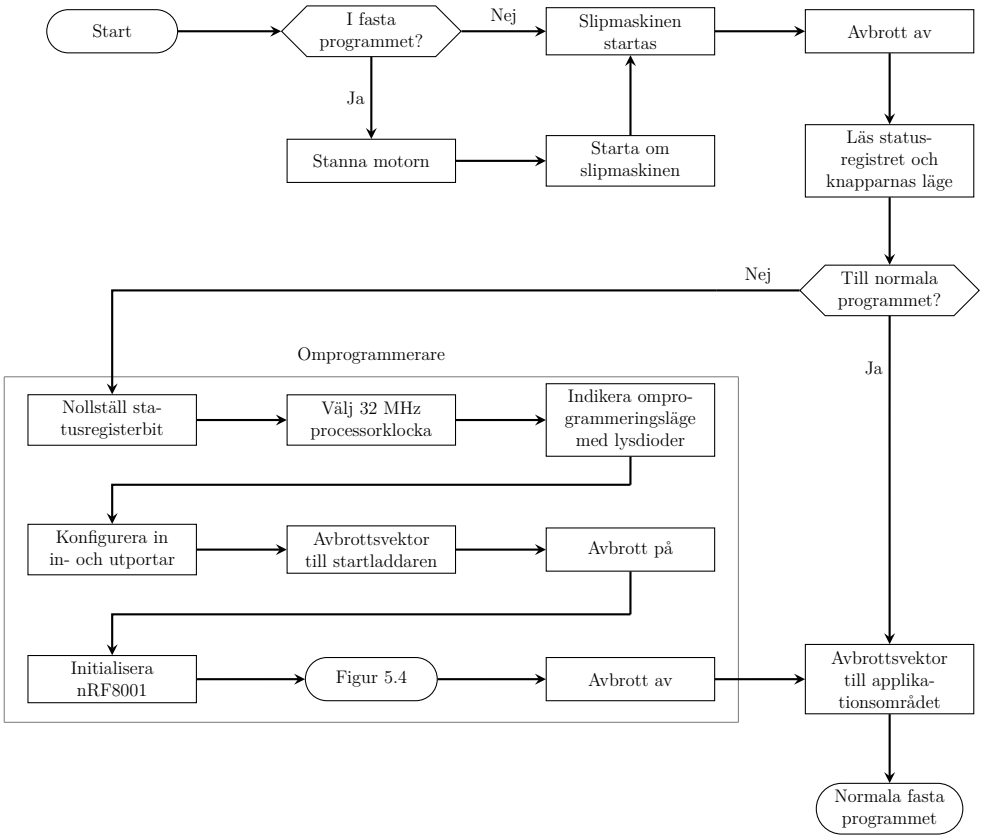
Den första punkten på listan handlar inte strikt om endast startladdaren, utan här är det i stor grad Mirkas egen styrkod som regerar. Den modifiering som gjorts tillät att en uppdatering skulle påbörjas i vilket skede som helst, även då slipmaskinen var i bruk och motorn roterade. Det här representerade ett litet orosmoment, helst borde motorn vara stilla före omprogrammeringen påbörjas. I sig skall det inte vara någon fara även om den körs eftersom omstarten nollställer alla kontrollsignaler och motorn då stannar. Likväl testades detta scenario för att säkerställa att inget oförutsett inträffade. Inga problem påträffades.

5.3.2 Oavslutade och felaktiga uppdateringar

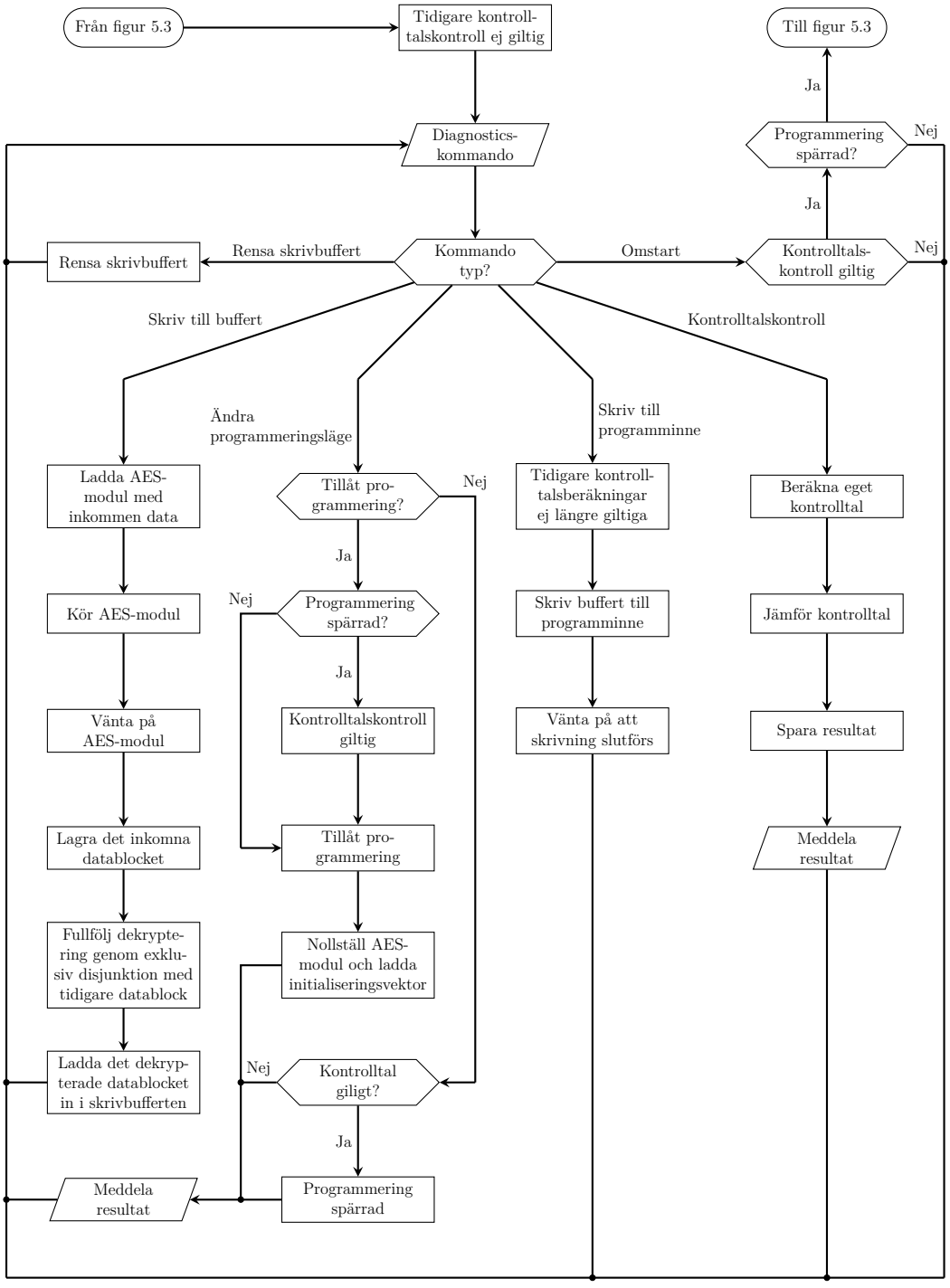
Att en uppdatering avbryts eller sker felaktigt var den i huvudsak största påtänkta felkällan. Strömförlust kan ske under alla förhållanden och den korta räckvidden på Bluetooth-modem betyder att en anslutning också enkelt kan avbrytas. Här testades två saker: fungerade låsningen korrekt och var det



Figur 5.2: Uppdateringssekvens.



Figur 5.3: Slutgiltig flödesplan för startladdaren.



Figur 5.4: Slutgiltig flödesplan för startladdaren.

möjligt att påbörja en ny omprogrammering efter att den föregående hade avbrutits. De listade metoderna användes för att simulera felen.

- Diagnostics tvingades att stänga under en pågående uppdatering.
- Strömmen bröts till slipmaskinen under en pågående uppdatering.
- Bluetooth-modemet avlägsnades under en pågående uppdatering.
- En diskavbild sändes och efteråt användes ett inkorrekt kontrolltal för att försöka låsa upp slipmaskinen på nytt.

Alla tester gick som planerat. Kontakt med slipmaskinen kunde återfås och ingen möjlighet att hoppa till applikationsdelen av minnet hittades efter att slipmaskinen hade blivit låst.

6 Diskussion

Med projektets huvudåtagande i hamn väcks tankar om framtida förbättringar. Arbetet har också medfört en del funderingar kring värdet av omprogrammering i fältet. Att utveckla en uppdateringsförmåga medför en kostnad i både personalresurser för arbetet men även i programminne. En större attackyta skapas också vilket betyder ett större krav på säkerhetstänkande. Ett klart behov bör därför identifieras innan ett dylikt projekt sätts igång.

6.1 Vidareutveckling av omprogrammerare för DEROS

Bristen på programminne för startladdaren var en konstant brännpunkt under projektets gång. Även om det inte förhindrade förverkligandet av varken grundkraven eller de förbättringar som påtänkts under kravsättningsfasen begränsades likväl omprogrammerarens funktionalitet. Små förfiningar kunde ha gjorts redan med några tiotal oktetter mera utrymme. Potentialen för förbättringar finns också med Diagnostics vilka inte utfördes då de låg utanför projektets ramar.

Under skrivprocessen av diplomarbetet har även en del småfel i koden uppdagats, primärt på Diagnostics sida. Emedan de inte påverkar den efterfrågade funktionaliteten i sig visar det på värdet av en genomgång med lite distans. En kontroll gjordes trots allt redan då det praktiska arbetet utfördes, men eftersom allt verkade fungera korrekt och koden inte analyserades noggrannare slank felen förbi. Avsaknaden av ett mera formellt testschema och en avsaknad av enhetstester var bidragande faktorer.

6.1.1 Putsning av slipmaskinens omprogrammeringskod

I slutresultatet hade Diagnostics ansvaret för att uppdateringssekvensen följdes. Koden i startladdaren kontrollerade nästan inte alls att sekvensen fortsatte i den förväntade ordningen. Kravet på ett korrekt kontrolltal i slutändan var den största, och givetvis också viktigaste, begränsningen startladdaren satte. Här finns potential för förbättringar. Bland annat vore det möjligt att kontrollera huruvida för många försök att skriva till flashbufferten gjorts. Om så är fallet kunde t.ex. fler skrivningar förhindras tills ett nytt omprogrammeringsförsök inleds. Att slå ihop tillstånd 1 och 4 i figur 5.1 är en annan

sak på listan av förbättringar.

Från Mirkas tidigare Bluetooth-kod ärvdes även UUID:na i projektet. För tjänsten blev det 0x0002 och för karakteristikan 0x0006. Tyvärr är det här rentav felaktigt då de är 16 bitar långa och därför inte fritt tillgängliga. Ifall man använder 16 bitars UUID:en bör de uppfylla de användningsändamål de satts att tjäna, vilket de nu inte gör. Som tidigare nämndes i del 3.3.1 bör man istället använda 128 bitar långa UUID:en då man behöver en egen specialanpassad tjänst eller karakteristika.

Mera långtgående reduceringar i utrymmeskravet för Nordic Semiconductors förändringar av kommunikationskoden torde även vara möjligt, trots det arbete som lades ner på denna bit. Ett exempel på en relativt stor datapost är konfigurationsmeddelandet till modemet. Detta lagras för tillfället i flashminnet. På bekostnad av EEPROM:en kunde man här frigöra cirka $\frac{1}{2}$ kibioktett av programminne.

6.1.2 Distribution av uppdateringar

En aspekt som definitivt låg utanför projektets ramar, men som samtidigt var nära kopplat med det, är uppdateringsfilerna. Som nämnts under rubrik 5.2.2 användes normala Intel HEX-filer av Diagnostics. Krypteringen gjordes av programmet strax före innehållet skickades till slipmaskinen. Detta betyder givetvis att nyttan med krypteringen minskar avsevärt. Även om endast underhållscentraler får tillgång till Diagnostics är det ändå mångfalt fler möjligheter för att krypteringsnyckeln skall spridas till någon obehörig. Omprogrammeringskoden hade en begränsad kapacitet för partiella uppdateringar; Diagnostics kan välja till vilken sida av programminnet skrivningar görs. Följaktligen är det möjligt att endast leverera filer med en del av det fasta programmet om förändringarna är små.

En mera lämplig metod vore att utvecklingsenheten vid Mirka skapade filer innehållande krypterade versioner av programavbilden och det nödvändiga kontrolltalet. Dessa filer skulle i sin tur användas av Diagnostics när en uppdatering utförs. Med säkerhet i åtanke finns även ett annat problem med projektets kod, ty diskavbildens kontrolltal skickades helt utan kryptering till slipmaskinen. Sådana läckor av information om det okrypterade innehållet underminerar krypteringen vilket i värsta fall kan leda till att någon kan

arbeta fram den hemliga nyckeln[5]. Egentligen borde den skickas i en krypterad form. Orsaken till att ingen kryptering utfördes av kontrolltalet var det knappa utrymmet vilket inte medgav att så gjordes.

Oberoende av hur distributionen sker kan inläsningen av en fil snyggas till. Som nu är fallet görs ingen riktig kontroll av filen alls. Om filen är korrekt är allt frid och fröjd men Diagnostics borde åtminstone försöka upptäcka om filen är förvanskad. En start på en sådan kontroll existerar delvis redan. Koden försöker dela in filen i rader och analysera dem. Men denna analys sätter inga spärrar, felaktiga rader ignoreras helt enkelt. Felhanteringen borde alltså utvidgas och därtill borde filstorleken kontrolleras från början.

6.1.3 Fel funna i efterhand

I början av en uppdateringssekvens avslutar Diagnostics en anslutning och startar den på nytt om slipmaskinen är i startladdaren. Det korrekta beteendet bör vara att den inte gör så. Likväl finns det en poäng med detta beteende eftersom det garanterar att anslutningen fortfarande fungerar. Senare under sekvens skickas även ett EraseBuffer kommando i början av varje uppdateringsloop, vilket ses i figur 5.2, medan det egentligen endast bör skickas en gång strax före huvudslingan startar. Ingendera av dessa fel har någon större inverkan på omprogrammerarens funktionalitet.

Två fel som emellertid är av vikt sker under inläsningen av en fil. Som nämnts analyseras posterna i en HEX-fil även om ingenting görs åt fel som hittas. Ett av felen är att adressposten ignoreras, datainnehållet fogas helt enkelt ihop i en sekventiell ordning. Det andra felet har att göra med kontrolltalet för en rad. I den befintliga koden beräknas inte kontrolltalet egentligen, istället jämförs filens egna innehåll med sig själv vilket självfallet alltid blir det samma. Detta bör naturligtvis korrigeras. Felen upptäcktes inte under testskedet eftersom inget test gjordes på förvanskade HEX-filer. Förväntan på att distributionsmetoden skulle ändras av Mirka senare var så stark att avsaknaden på ett sådant testscenario förbisågs.

[5]Nyckeln kan tas fram ändå eftersom mikrokontrollerns skydd av minnet också kan angripas

6.2 Tankar kring trådlösa uppdateringar

Det praktiska arbetet utfördes 2015 och utveckligen har stadigt gått vidare sen dess. Redan då fanns emellertid trådlösa uppdaterings för bland annat bilar och telefoner. Till min kännedom var dock möjligheten att uppdatera programvaran på verktyg såsom slipmaskiner ytterst begränsad vid denna tidpunkt och detta gäller fortfarande. Trådlösa uppdateringar har ett mervärde men det kan även leda till problem och är definitivt en kostnadsökning.

6.2.1 Säkerhet

Säkerhetsrisken som skapas är en av de stora baksidorna med att ha en kommunikationskanal. Den förstärks ytterligare när omprogrammering av systemet tillåts. Tyvärr är det även ett område som ofta ignoreras då man inte ser potentialen för problem. Till exempel överbelastningsattacker som sker genom osäkra apparater är ett så stort problem att det forskas för att reducera deras inverkan[14]. För att undvika dylika olägenheter bör man därför alltid ifrågasätta nyttan med en lätt tillgänglig kommunikationskanal.

En annan säkerhetsaspekt gäller själva programavbilden och uppdateringsprocessen av enheten. Om programbilden är okrypterad kan en tredje part lätt få tag på styrkoden och modifiera den och därefter illvilligt uppdatera maskiner med dem. I fallet med DEROS:en är räckvidden begränsad, men om maskinen skulle vara kopplade till internet vore problemet mycket större. För att förhindra dylikt bör en stark kryptering användas och mikrokontrollern bör modifieras så att dess kod inte kan läsas. XMEGA:n har till exempel säkringsoktetter som kan förhindra andra från att enkelt läsa innehållet i programminnet. Säkerhetsåtgärder såsom dessa är emellertid inte absoluta hinder då det är möjligt att få information ur mikrokontrollern[8][17], men det är inte trivialt.

6.2.2 Överföringshastighet

Om uppdateringar endast sker ett fåtal gånger under en produktlivscykel är överföringshastigheten generellt av ringa betydelse. För ett inbyggt system handlar det ofta om små minnesstorlekar. Även med de långsammaste generellt använda överföringsmetoderna tar det alltså inte mycket tid i anspråk. Överföringen kan även effektiveras genom att inte sända överflödig

information[12]. Trots detta är det värt att fundera kring nackdelarna med en begränsad överföringshastighet. Mirkas idé om att utföra uppdateringar vid underhållscentraler kan här vara ett belysande exempel. Hur ofta kommer de att utföras och hur mycket är personalens tid värd gentemot kostnaden av ett snabbare modem? Då tanken är att uppdateringar endast görs av slutanvändare ser kostnadskalkylen annorlunda ut.

Utöver användarvänlighet ger även en hög överföringshastighet ett visst mått av flexibilitet i produktionssammanhang. Det är sannolikt att mikrokontrollerna köps in från underleverantör. Om den beställs färdigprogrammerad kan programvaran vara föråldrad redan då slutprodukten sätts ihop. Om uppdateringar kan göras snabbt är det realistiskt att införliva dem i produktionsprocessen och garantera att den senaste programversionen finns i maskinen när den lämnar fabriken. Frågan blir då hur värdefullt det är med en trådlös kommunikationskanal istället för en trådbunden?

7 Slutsats

Detta diplomarbete har behandlat arbetet som gjorts för att möjliggöra trådlös uppdateringen av en slipmaskins mikrokontroller: en XMEGA32A4U. Hela projektet utfördes genom en modifikation av slipmaskinens fasta program samt med modifieringar av Mirkas Diagnostics programvara som användes av bordsdatorer. Den befintliga hårdvaran, som inkluderade ett Bluetooth-modem, användes utan modifikation. Det visade sig vara fullt möjligt att fullfölja projektet. Det största bekymret visade sig vara storleken på den kod som skötte den trådlösa kommunikationen. Även efter att oanvända delar av kommunikationsprotokollet skalats bort användes nästa allt av minnet i XMEGA:ns startladdarområde. Endast ett väldigt begränsat antal bytes lämnade kvar för all annan funktionalitet som krävdes. Tack vare XMEGA:ns inbyggda funktioner såsom AES-128 hantering och CRC kontroller var det ändå möjligt att implementera en viss grad av säkerhet utöver den efterfrågade grunduppdateringsmöjligheten.

8 Referenser

[1] Albarakati, Aiman J. The trend of an embedded undersized range wireless communication technologies. *International Journal of Computer Applications*, 118(5):34–37, 5 2015.

[2] Asbjørn. nrf8001: Number of packets per connection event [forumkommentar]. `https://devzone.nordicsemi.com/f/nordic-q-a/3908/nrf8001-number-of-packets-per-connection-event`, 9 2014. [Hämtad 2020-11].

[3] Atmel. Xmega a manual. `http://ww1.microchip.com/downloads/en/DeviceDoc/doc8077.pdf`, 2012. [Revision 8077I–AVR–11/2012; hämtad 2020-04].

[4] Atmel. Xmega a4u datasheet. `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8387-8-and16-bit-AVR-Microcontroller-XMEGA-A4U_Datasheet.pdf`, 2014. [Revision Atmel-8387H-AVR-ATxmega16A4U-32A4U-64A4U-128A4U-Datasheet_09/2014; hämtad 2020-04].

[5] Das, Amitabh, Da Rolt, Jean, Ghosh, Santosh, Seys, Stefaan, Dupuis, Sophie, Di Natale, Giorgio, Flottes, Marie-Lise, Rouzeyre, Bruno, och Verbauwhede, Ingrid. Secure jtag implementation using schnorr protocol. *Journal of Electronic Testing*, 29(2):193–209, 04 2013.

[6] Group, Bluetooth Special Interest. Bluetooth specification version 4.0. `https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=456433`, 6 2010. [Hämtad 2020-04].

[7] Intel. Hexadecimal object file format specification, 1 1988.

[8] Kizhvatov, Ilya. Side channel analysis of avr xmega crypto engine. I: *Proceedings of the 4th Workshop on Embedded Systems Security*, WESS '09. Association for Computing Machinery, 2009. ISBN 9781605587004.

[9] Mahmud, Moammar. Light switch with microcontroller. `https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=20130102&DB=EPODOC&locale=&CC=GB&NR=2492403A&KC=A&ND=1`, 1 2013. [Hämtad 2020-04].

[10] Maras, Marie-Helen. Internet of Things: security and privacy implications. *International Data Privacy Law*, 5(2):99–104, 04 2015. ISSN 2044-3994. URL `https://doi.org/10.1093/idpl/ipv004`.

[11] Mariotti, Costantino. Multiprogram washing machine with an electromechanical programmer and a digital microcontroller. https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=19951220&DB=EPODOC&locale=&CC=EP&NR=0687761A1&KC=A1&ND=1, 12 1995. [Hämtad 2020-04].

[12] Mazumder, B. och Hallstrom, J. O. An efficient code update solution for wireless sensor network reprogramming. I: *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, ss 1–10, 2013.

[13] Nordic Semiconductor. nrf8001 product specification 1.3. https://www.nordicsemi.com/-/media/DocLib/Other/Product_Spec/nRF8001PSv13.pdf, 3 2015. [Version 1.3; hämtad 2020-04].

[14] Om Kumar, C U och Sathia Bhama, Ponsy R. K. Detecting and confronting flash attacks from iot botnets. *The Journal of Supercomputing*, 75(12):8312–8338, 12 2019.

[15] Silicon Labs. Bluegiga bluetooth smart software. https://www.silabs.com/documents/public/reference-manuals/Bluetooth_Smart_Software-BLE-1.7-API-RM.PDF, 12 2018. [Hämtad 2020-11].

[16] Singh, Gurpreet. A study of encryption algorithms (rsa, des, 3des and aes) for information security. *International Journal of Computer Applications*, 67(19), 2013.

[17] Strobel, D., Oswald, D., Richter, B., Schellenberg, F., och Paar, C. Microcontrollers as (in)security devices for pervasive computing applications. *Proceedings of the IEEE*, 102(8):1157–1173, 2014.

[18] Taizhou LG Electronics Refrigeration. Microcontroller for refrigerator. https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=20080702&DB=EPODOC&locale=&CC=CN&NR=101211163A&KC=A&ND=1, 7 2008. [Hämtad 2020-04].