Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Grmek

Zajem zaslonske slike iz vgrajenih naprav

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V letalih se uporablja vrsta vgrajenih naprav z grafičnimi zasloni. Za potrebe izdelave uporabniške dokumentacije implementirajte rešitev, s katero bo mogoče zajeti zaslonsko sliko naprav v letalu. Rešitev implementirajte na mikrokrmilniku STM32F7 ter na osebnem računalniku. Za prenos slike iz vgrajene naprave v osebni računalnik uporabite vodilo CAN.

Zahvaljujem se družini in prijateljem za vso podporo v času študija ter mentorju za pomoč pri izdaji te naloge. Posebna zahvala gre tudi sodelavcem iz podjetja Pipistrel, ki so mi omogočili pripravo diplomske naloge v sodelovanju s podjetjem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CAN	controller area network	krmilno omrežje
CANH	CAN high	visoka linija CAN
CANL	CAN low	nizka linija CAN
CRC	cyclic redundancy check	preverjanje ciklične redundance
HAL	hardware abstraction layer	plast abstrakcije strojne opreme
ISO	international organization for standardization	mednarodna organizacija za standardizacijo
LSB	least significant byte	najmanj pomemben bajt
MISO	master in, slave out	iz sužnja v gospodarja
MOSI	master out, slave in	iz gospodarja v sužnja
MSB	most significant byte	najbolj pomemben bajt
OSI	open systems interconnection	medsebojno povezovanje odprtih sistemov
PNG	portable network graphics	prenosna omrežna grafika
RAD	rapid application development	hiter razvoj aplikacij
RAM	random access memory	bralno pisalni pomnilnik
RGB565	red green blue, 5 6 5	rdeča zelena modra, 5 6 5
RLE	run length encoding	verižno kodiranje
ROM	read only memory	bralni pomnilnik

kratica	**angleško**	**slovensko**
SCK	serial clock	zaporedna ura
SPI	serial peripheral interface	zaporedni periferni vmesnik
SS	slave select	izbor sužnja

Povzetek

Naslov: Zajem zaslonske slike iz vgrajenih naprav

Avtor: Rok Grmek

Cilj diplomske naloge je bil razvoj sistema za zajem zaslonske slike iz obstoječih vgrajenih naprav podjetja Pipistrel. Omenjene naprave omogočajo povezljivost v omrežje CAN, prav tako pa je s primernim vmesnikom možno v omrežje CAN priključiti tudi osebni računalnik. Ciljni sistem torej predvideva aplikacijo za osebni računalnik, s katero lahko uporabnik preko omrežja CAN zahteva zaslonsko sliko iz vgrajene naprave in programski modul za vgrajeno napravo, ki poskrbi za izvedbo omenjene akcije na vgrajeni napravi. Razvoj sistema je potekal po metodologiji RAD, kjer je najprej bil pripravljen prototip, ki obsega le ključno funkcionalnost brez kakršnihkoli optimizacij, za tem pa je bilo dodanih več izboljšav. Te se nanašajo predvsem na pomanjkanje pomnilnika na strani vgrajene naprave in nizko pretočnost omrežja CAN. Končna implementacija je bila preizkušena na dveh različnih vgrajenih napravah.

Ključne besede: zaslonska slika, vgrajena naprava, krmilno omrežje.

Abstract

Title: Capturing screenshots from embedded devices

Author: Rok Grmek

The goal of the diploma thesis was the development of a system for capturing screenshots from existing Pipistrel embedded devices. These devices offer CAN network connectivity, and with an appropriate interface, a personal computer can also be connected to the CAN network. The target system therefore envisages a personal computer application, where a user can request a screenshot from the embedded device via the CAN network and a software module for the embedded device, which ensures that the said action is performed on the embedded device. The development of the system was carried out according to the RAD methodology, where a prototype with only the key functionality and without any optimizations was prepared first, and then several improvements were added later. These mainly concern the lack of memory on the embedded device side and the low bandwidth of the CAN network. The final implementation was tested on two different embedded devices.

Keywords: screenshot, embedded device, controller area network.

Poglavje 1

Uvod

Podjetje Pipistrel se ukvarja z razvojem naprednih letalskih rešitev. Na tem področju je nepogrešljiva tudi elektronika, vendar zaradi specifičnih potreb podjetja, pogosto ne najdemo celovitih rešitev, ki bi omogočale točno želen nabor funkcionalnosti. Posledično se podjetje ukvarja tudi z razvojem lastnih vgrajenih naprav, kar zajema pripravo strojne in programske opreme po meri.

Nekatere od Pipistrelovih vgrajenih naprav služijo zgolj krmiljenju in usklajevanju posameznih delov letalskega sistema in uporabnik z njimi nima neposredne interakcije. Drugi tip vgrajenih naprav pa služi kot uporabniški vmesnik. Te naprave uporabniku omogočajo nadzor vseh ključnih komponent sistema (npr. vpogled v stanje baterijskih paketov v električnem letalu). Primer Pipistrelove vgrajene naprave, ki služi kot uporabniški vmesnik, je prikazan na sliki 1.1.

1.1 Motiv

Za uporabnika, ki se prvič sreča z novo vgrajeno napravo, ki služi kot uporabniški vmesnik, je zelo pomembna dokumentacija v obliki priročnika z navodili za uporabo. Na tem mestu podjetje potrebuje rešitev, ki bi omogočala zajem zaslonske slike iz omenjenih vgrajenih naprav. Zaradi obstoječih vgrajenih naprav smo pri rešitvi omejeni z že razvito strojno opremo, na kateri za-

Slika 1.1: Primer Pipistrelove vgrajene naprave [11]

jem zaslonske slike ni bil posebej predviden, hkrati pa rešitev ne sme vplivati na delovanje trenutne programske opreme. Kljub omejitvam pa vse obstoječe vgrajene naprave omogočajo povezljivost v omrežje CAN, v katero lahko s primernim vmesnikom povežemo tudi osebni računalnik. Smiselna rešitev bo torej sestavljena iz dodatnega programskega modula za vgrajeno napravo in aplikacije za osebni računalnik. Aplikacija bo omogočala identifikacijo razpoložljivih vgrajenih naprav v omrežju CAN, pošiljanje zahtevkov za zajem zaslonske slike izbrani vgrajeni napravi, prenos zajete slike do aplikacije in shranjevanje prenesene slike. Modul za vgrajeno napravo pa bo zagotovil, da se bo ta primerno odzivala na vse omenjene akcije, kar vključuje tudi pridobivanje zaslonske slike iz grafičnega krmilnika.

1.2 Pregled

V naslednjih poglavjih je najprej opisan sistem za zajem zaslonske slike, kjer so podrobneje predstavljeni ključni elementi vgrajene naprave (procesor in obstoječa programska oprema, grafični krmilnik ter vodilo SPI), elementi

sistema na strani osebnega računalnika (tip aplikacije in uporabljeni vmesnik CAN) in tudi samo omrežje CAN.

Predstavljen je razvoj rešitve, ki je potekal po metodologiji RAD. Ta vključuje pripravo prvega hitrega prototipa, za tem pa sledi več zaporednih vrednotenj in izboljšav vse do končne rešitve.

Končna rešitev je podrobneje opisana iz treh vidikov. V obliko končnega avtomata je prikazano delovanje programskega modula za vgrajeno napravo, po elementih grafičnega uporabniškega vmesnika je pojasnjeno delovanje aplikacije za osebni računalnik, na koncu pa je neposredno naveden še protokol komuniciranja med vgrajeno napravo in osebnim računalnikom.

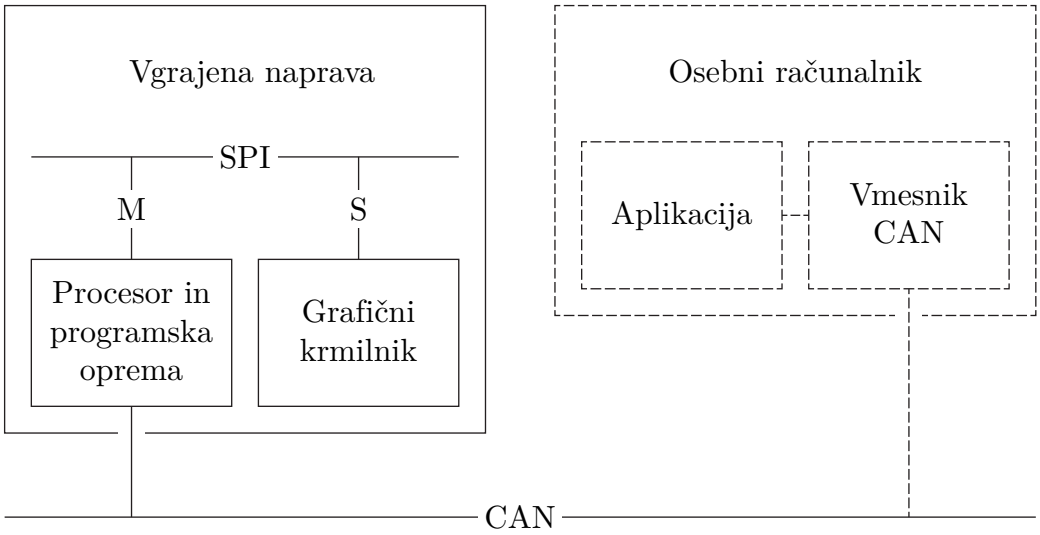
S testiranjem pokažemo smiselno delovanje končne rešitve in navedemo okvirne čase trajanja nekaterih daljših akcij za dve različni vgrajeni napravi podjetja Pipistrel.

V zadnjem poglavju je na kratko povzet rezultat diplomske naloge in njegova uporabnost v praksi.

Poglavje 2

Opis sistema

Sistem, s katerim bomo lahko zajeli zaslonsko sliko iz vgrajenih naprav, je sestavljen iz dveh delov - iz same vgrajene naprave in osebnega računalnika, oba pa sta povezana v omrežje CAN, preko katerega lahko komunicirata. Simbolično je sistem prikazan na sliki 2.1, kjer je s polno črto označen obstoječi del sistema, katerega delovanje bomo prilagodili z dodatnim programskim modulom za vgrajeno napravo, s črtkano črto pa je označen novo dodani del sistema. Posamezni elementi so podrobneje opisani v nadaljevanju.



Slika 2.1: Simbolični prikaz sistema

2.1 Vgrajena naprava

2.1.1 Procesor in programska oprema

Eden od pomembnejših delov vgrajenih naprav je zagotovo procesor. Pipistrelove vgrajene naprave, ki služijo kot uporabniški vmesnik in bi zanje radi omogočili zajem zaslonske slike, uporabljajo procesor iz družine STM32F7x7. Za to družino procesorjev je značilno:

- jedro Arm Cortex-M7,
- takt sistemske ure do 216 MHz,
- 1 do 2 MB ROM,
- 512 KB RAM,
- do 28 vmesnikov za komunikacijo (tudi SPI in CAN) [14].

Procesor skrbi za izvajanje obstoječe programske opreme, ki je v našem primeru napisana v programskem jeziku C in smiselno razdeljena na več opravil. Za pravilno usklajevanje in izvajanje opravil skrbi operacijski sistem v realnem času FreeRTOS [2]. Pri zajemu zaslonske slike bosta za nas pomembni le dve opravili - opravilo za grafiko, kjer upravljamo grafični krmilnik in lahko na tem mestu zajamemo zaslonsko sliko ter opravilo za komunikacijo, kjer obdelujemo sporočila CAN, kar nam omogoča komunikacijo z osebnim računalnikom, povezanim v omrežje CAN.

Oba potrebna vmesnika za komunikacijo (SPI in CAN) sta že uporabljena v obstoječi programski opremi, kar pomeni, da sta oba tudi že primerno inicializirana. Prav tako pa je v obstoječi programski opremi že pripravljen nabor funkcij za upravljanje grafičnega krmilnika z vmesnikom SPI ter vrsti za prejemanje in pošiljanje sporočil CAN, ki se odzivata ob prekinitvah vmesnika CAN. Omenjeni del programske opreme je implementiran z uporabo knjižnice HAL [13]. Ta knjižnica predstavlja plast abstrakcije strojne opreme, kar nam omogoča enotno upravljanje vmesnikov kljub nekaterim razlikam v izvedbi strojne opreme.

Vmesnik SPI je inicializiran s parametri, ki določajo: način gospodarja (SPI_MODE_MASTER), popolnoma dvosmerno komunikacijo (SPI_DIRECTION_2LINES), 8 bitne prenose (SPI_DATASIZE_8BIT), privzeto nizko stanje urinega signala (SPI_POLARITY_LOW), začetek vzorčenja ob prvi urini fronti (SPI_PHASE_1EDGE), programski izbor sužnja (SPI_NSS_SOFT), deljenje frekvence vhodne ure s 4 (SPI_BAUDRATEPRESCALER_4), prenose z najbolj pomembnim bitom na začetku (SPI_FIRSTBIT_MSB) in onemogočeno računanje CRC (SPI_CRCCALCULATION_DISABLE).

Za vmesnik CAN pa so nastavljeni parametri, ki določajo: deljenje frekvence vhodne ure z 9 (CAN_PRESCALER_9), običajen način delovanja (CAN_MODE_NORMAL), največ 1 časovni kvant na bit za resinhronizacijo (CAN_SJW_1TQ), 3 časovne kvante za prvi segment bita (CAN_BS1_3TQ), 4 časovne kvante za drugi segment bita (CAN_BS2_4TQ), onemogočen način komunikacije s časovnim proženjem (CAN_TTCM_DISABLE), omogočeno avtomatsko ugašanje vodila (CAN_ABOM_ENABLE), onemogočen način avtomatskega prebujanja (CAN_AWUM_DISABLE), onemogočeno neavtomatsko ponovno pošiljanje (CAN_NART_DISABLE), onemogočen način zaklepanja vrste za prejemanje (CAN_RFLM_DISABLE) in onemogočeno pošiljanje sporočil brez razvrščanja po prioriteti (CAN_TXFP_DISABLE).

2.1.2 Grafični krmilnik

Za generiranje grafične vsebine, ki je prikazana na zaslonu, se uporablja zunanji grafični krmilnik iz družine FT81x. Tega v celoti upravljamo preko vodila SPI, pri čemer je večina komunikacije namenjena zgolj pisanju in branju po pomnilniku. Pomnilnik je razdeljen na:

- splošno namenski pomnilnik (1024 KB),
- seznam ukazov za prikazovanje (8 KB),
- registre (4 KB),
- krožni medpomnilnik z ukazi za koprocesor (4 KB).

Začetno konfiguracijo grafičnega krmilnika zapišemo v registre, za tem pa ukaze pišemo v krožni medpomnilnik z ukazi za koprocesor. Pri tem vedno najprej preverimo registra z začetnim in končnim naslovom krožnega medpomnilnika in se prepričamo, da imamo na razpolago dovolj prostega pomnilnika za pisanje želenega ukaza, sicer pa počakamo, da koprocesor prevzame nekaj ukazov. Po uspešnem pisanju ukaza, posodobimo še register s končnim naslovom krožnega medpomnilnika [3].

Za generiranje grafične vsebine najprej pošljemo ukaz, ki označuje začetek seznama ukazov za prikazovanje, za tem pa sledi seznam ukazov za prikazovanje, s katerimi generiramo grafično vsebino (npr. za izris tanke rdeče vodoravne črte: debelina(1), barva(255, 0, 0), začni(črto), oglišče(20, 20), oglišče(40, 20)). Koprocesor bo ob tem pisal seznam ukazov za prikazovanje še v temu namenjen naslovni prostor, pred tem pa bo po potrebi počakal, da se prejšnji seznam dokončno obdela. Na koncu dodamo še ukaz za zamenjavo seznama ukazov za prikazovanje, s katerim zahtevamo obdelavo trenutnega seznama [3].

Zajem zaslonske slike iz vgrajene naprave je možen, ker grafični krmilnik podpira ukaz za shranjevanje trenutno aktivne slike, ustvarjene iz zadnjega seznama ukazov za prikazovanje. Slika se pri tem shrani v splošno namenski pomnilnik grafičnega krmilnika, na naslov, ki ga kot parameter podamo ukazu. Pri tem moramo paziti, da s sliko ne prepišemo drugih podatkov, saj so v splošno namenskem pomnilniku shranjene tudi prednaložene slike in pisave, za beleženje zasedenega pomnilnika pa moramo poskrbeti sami [3].

2.1.3 Vodilo SPI

Vodilo SPI omogoča popolnoma dvosmerno, sinhrono, serijsko komunikacijo med gospodarjem in enim ali več sužnji. Gospodar za vsak prenos najprej izbere sužnja (do vsakega sužnja posebej vodi ločen signal SS), takoj za tem pa začne z generiranjem urinega signala SCK. Ob tem lahko gospodar hkrati pošilja podatke z generiranjem signala MOSI in prejema podatke z branjem signala MISO (izbrani suženj pa obratno). V posameznem prenosu običajno

izmenjamo 8 bitov, lahko pa tudi več [8].

Pomembno je, da imata gospodar in suženj usklajeno konfiguracijo polaritete in faze urinega signala. SPI namreč dopušča 4 različne konfiguracije. Ura je lahko privzeto v nizkem stanju in z vsakim pulzom preide v visoko stanje ali pa obratno. Prav tako pa lahko izberemo, ali bomo prvi bit vzorčili ob prvi urini fronti ali pa šele ob drugi [8].

SPI sužnji gospodarju velikokrat omogočajo dostop do dela njihovega pomnilnika, kar omogoča konfiguracijo in upravljanje takih naprav. V takih primerih mora gospodar običajno najprej poslati ciljni pomnilniški naslov z dodatnim bitom, ki označuje, ali gre za bralni ali pisalni dostop, za tem pa sledi branje oz. pisanje podatkov na izbrani naslov.

2.2 Osebni računalnik

2.2.1 Aplikacija

Pri izbiri platforme za razvoj aplikacije nimamo posebnih omejitev. Zaradi ostalih programskih orodij, ki jih uporabljajo pri podjetju Pipistrel, bomo tudi aplikacijo za zajem zaslonske slike iz vgrajenih naprav pripravili za operacijski sistem Microsoft Windows. Uporabili bomo programski jezik C# in knjižnico Windows Forms [5], ker ta omogoča hiter razvoj aplikacij z grafičnim vmesnikom za operacijski sistem Microsoft Windows.

2.2.2 Vmesnik CAN

Aplikacija bo komunicirala z vgrajeno napravo preko omrežja CAN. Zato potrebujemo vmesnik CAN, ki bo osebnemu računalniku omogočil povezljivost v omrežje CAN. Uporabili bomo vmesnik Peak PCAN-USB, ki ponuja tudi aplikacijski programski vmesnik PCAN-Basic [10] za programski jezik C#. PCAN-Basic ponuja več funkcij za upravljanje vmesnika CAN. V aplikaciji potrebujemo le osnovne funkcionalnosti, ki jih bomo dosegli s funkcijami:

- `TPCANStatus Initialize(TPCANHandle Ch, TPCANBaudrate Btr)`,

- `TPCANStatus Read(TPCANHandle Ch, out TPCANMsg MsgBfr)`,
- `TPCANStatus Write(TPCANHandle Ch, ref TPCANMsg MsgBfr)`.

S prvo funkcijo poskrbimo za inicializacijo vmesnika, preostali dve funkciji pa sta namenjeni prejemanju in pošiljanju sporočil CAN.

2.3 Omrežje CAN

Omrežje CAN ponuja komunikacijo med več napravami v omrežju, kjer vse naprave delujejo kot gospodar, vsa sporočila pa so vedno namenjena vsem napravam. Protokol obsega fizično in povezovalno plast ISO/OSI referenčnega modela [15].

Na fizični plasti sta predvidena signala CANH in CANL. Gre za diferencialni par, ki omogoča robustno odpornost na motnje. Pomemben pa je tudi princip dominantnega in recesivnega bita. Če dve napravi hkrati v omrežje pošljeta ena dominanten, druga pa recesiven bit, bo obveljal dominanten bit. To je v primeru omrežja CAN logična ničla [15].

Povezovalna plast ponuja 4 različne tipe okvirjev - podatkovni okvir, oddaljeni okvir, okvir napake in okvir preobremenitve [15].

Najpogostejši je podatkovni okvir, ki je očitno namenjen prenosu podatkov. V grobem je ta razdeljen na arbitražno, podatkovno, CRC in potrditveno polje. Arbitražno polje vsebuje v običajnem okvirju 11-bitni identifikator, v razširjenem okvirju pa 29-bitni identifikator. V obeh primerih polje vsebuje še bit za zahtevo oddaljenega pošiljanja, ki je v primeru podatkovnega okvirja vedno dominanten. Podatkovno polje lahko vsebuje od 0 do 8 bajtov aplikacijskih podatkov. CRC polje predstavlja 16 bitno kontrolno vsoto za zaznavanje napak. Zadnje, potrditveno polje pa je namenjeno potrjevanju sporočila. Pošiljatelj na to mesto vedno postavi recesiven bit, vsaka naprava, ki sporočilo prejme brez napak, pa bit prepiše z dominantnim bitom. Če nobena naprava ne potrdi sporočila, potem pošiljatelj začne s ponovnim pošiljanjem [15].

Z oddaljenim okvirjem lahko zahtevamo podatke od druge naprave. Ta okvir je po obliki zelo podoben podatkovnemu okvirju. Razlika je v bitu za zahtevo oddaljenega pošiljanja, poleg tega pa ta okvir ne vsebuje podatkovnega polja [15].

Okvir napake je po obliki drugačen od prejšnjih dveh okvirjev. Tega pošlje naprava, ki zazna napako v nekem sporočilu, pošljejo ga pa tudi vse ostale naprave, ki so prejele okvir napake (pri tem se s štetjem napak prepreči, da bi naprave z okvirji napake zasedle omrežje). Naprava, ki je poslala sporočilo z napako, za tem ponovno pošlje sporočilo [15].

Okvir preobremenitve je po obliki podoben okvirju napake. Pošlje ga naprava, ki je preobremenjena s prometom. Običajno pa se v takem primeru poveča čas med posameznimi sporočili [15].

V omrežju se lahko zaradi večjega števila gospodarjev zgodi, da več naprav hkrati začne s pošiljanjem sporočila. Na tem mestu je pomembna arbitraža, ki daje prednost sporočilom z nižjim identifikatorjem. Dokler se pomembnejši biti identifikatorjev pri hkrati pošiljanih sporočilih ujemajo, lahko vse naprave nadaljujejo s pošiljanjem. Ko pa neka naprava v omrežje pošlje recesiven bit identifikatorja (logična enica) in signal preide v dominantno stanje (logična ničla), to pomeni, da neka druga naprava hkrati pošilja sporočilo z nižjim identifikatorjem. V takem primeru mora naprava z višjim identifikatorjem prenehati s pošiljanjem. Na tak način se brez kakršnegakoli konflikta v omrežju obdrži le sporočilo z najnižjim identifikatorjem [15].

Poglavje 3

Razvoj rešitve

Razvoj sistema za zajem zaslonske slike iz vgrajenih naprav je potekal po metodologiji RAD. Gre za način razvoja, kjer namesto tradicionalnega začetnega načrtovanja celotnega sistema, na začetku le identificiramo ključno funkcionalnost in jo implementiramo. Za tem sledi več zaporednih izboljšav, kjer vsakič analiziramo prejšnjo rešitev in bodisi odpravimo kakšno pomanjkljivost ali pa dodamo novo funkcionalnost. Tak razvoj je zanimiv tudi iz časovnega vidika. Pri tradicionalnem razvoju težko omejimo čas razvoja, saj imamo točno določen nabor funkcionalnosti, ki smo si ga zadali z začetnim načrtovanjem in posledično se lahko dejanski čas razvoja precej razlikuje od razpoložljivega časa. Po drugi strani nam RAD omogoča, da v poljubni zaporedni fazi zaključimo razvoj in imamo na tak način implementiran nabor najpomembnejših funkcionalnosti, ki smo jih lahko dosegli v danem času [1].

V nadaljevanju so podrobneje opisane vse faze razvoja od prvega prototipa do končne rešitve. Programski modul za vgrajeno napravo smo pripravljali v razvojnem okolju IAR Embedded Workbench [4], aplikacijo za osebni računalnik pa v razvojnem okolju Microsoft Visual Studio [7].

3.1 Prvi prototip

S prvim prototipom implementiramo le ključno funkcionalnost - zajem zaslonske slike iz vgrajene naprave, ki je zagotovo priključena v omrežje CAN in je tudi edina razpoložljiva naprava v omrežju CAN. Identifikacijo naprav in naslavljanje zahtevkov za enkrat preskočimo.

Aplikacija v tem prototipu vsebuje en sam gumb, s katerim zahtevamo tri zaporedne akcije - zajem zaslonske slike na vgrajeni napravi, prenos slike do aplikacije in shranjevanje v datoteko. Ob kliku na gumb aplikacija pošlje zahtevek za omenjene akcije (sporočilo CAN s točno določenimi podatki). Ko vgrajena naprava v opravilu za komunikacijo prejme omenjeno sporočilo, preide v stanje za zajem zaslonske slike. Opravilo za grafiko v tem stanju pošlje grafičnemu krmilniku ukaz za zajem celotne zaslonske slike v njegov pomnilnik. Ko se omenjeni ukaz izvrši in je celotna zaslonska slika shranjena v pomnilniku grafičnega krmilnika, naprava preide v stanje za prenos slike. V tem stanju opravilo za grafiko v vsaki iteraciji glavne zanke prenese del slike iz pomnilnika grafičnega krmilnika v pomnilnik procesorja, opravilo za komunikacijo pa omenjene podatke pošilja v omrežje CAN, kjer jih aplikacija prebere in shrani. Ko se prenos zaključi, aplikacija pretvori shranjene podatke v format PNG in zapiše datoteko na disk. Pretvorbo izvedemo z razredom Bitmap [6], ki omogoča shranjevanje slikovnega objekta v več različnih formatov. Za tem naprava preide spet v privzeto stanje, kjer čaka na morebitni nov zahtevek za zajem zaslonske slike.

3.2 Zajem slike po delih

Prvi prototip sicer uspešno deluje, vendar le, ko imamo v grafičnem krmilniku dovolj razpoložljivega pomnilnika za celotno zaslonsko sliko. Zaradi obstoječe programske opreme pri Pipistrelovih vgrajenih napravah je pomnilnik grafičnega krmilnika običajno že precej zaseden s prednaloženimi grafičnimi elementi in prvi prototip v večini primerov ne bi deloval.

Omenjeni problem lahko rešimo tako, da sliko zajamemo po delih. Na

podlagi velikosti razpoložljivega pomnilnika v grafičnem krmilniku najprej izračunamo, kolikšen del slike sploh lahko shranimo s posameznim delnim zajemom. Nato vsak del slike zajamemo in posredujemo aplikaciji na enak način, kot smo to izvedli pri prvem prototipu za celotno sliko. Aplikacija sliko pretvori in shrani v datoteko, ko se zaključijo vsi delni prenosi.

3.3 Onemogočanje izrisa nove slike

Zaslonska slika se lahko med delnimi prenosi spreminja, kar pomeni, da obstaja možnost, da bomo pri prejšnjem prototipu zajeli sliko, ki bo sestavljena iz manjših delov, zajetih v različnih časih in posledično z različno vsebino. Tega seveda nočemo in takšno delovanje ni pravilno.

Težavo odpravimo tako, da opravilu za grafiko onemogočimo izris nove vsebine med zajemom posameznih delov zaslonske slike, na koncu pa izris spet omogočimo.

3.4 Zajem slike v pomnilnik procesorja

Prenos slike preko omrežja CAN je omejen s hitrostjo 500 kbit/s, pri čemer je velik del te že izkoriščen s prometom ostalih naprav v omrežju. Pri prejšnjem prototipu je izris nove vsebine na zaslonu onemogočen, odkar začnemo z zajemom prvega dela slike, dokler ne zajamemo vseh delov slike, to pa lahko zaradi vmesnega relativno počasnega pošiljanja v omrežje CAN traja tudi nekaj 10 sekund.

Vodilo SPI nas pri prenosu iz pomnilnika grafičnega krmilnika do pomnilnika procesorja bistveno manj omejuje. Grafični krmilnik je edina naprava na tem vodilu, hitrost prenosa pa lahko doseže 18 Mbit/s. Celotno sliko lahko torej po delih shranimo v pomnilnik procecsorja v nekaj sekundah in le v tem času onemogočimo izris nove vsebine na zaslonu. Za tem spet omogočimo izris in začnemo s pošiljanjem celotne slike iz pomnilnika procesorja v omrežje CAN.

3.5 Stiskanje slike

Prejšnji prototip ima podobno pomanjkljivost kot prvi prototip. Zaradi obstoječe programske opreme na nekaterih napravah procesor nima dovolj razpoložljivega pomnilnika, da bi lahko shranil celotno sliko.

Problem lahko rešimo s stiskanjem slike. S tem ohranimo vse prednosti prejšnjega prototipa in odpravimo le pomanjkanje razpoložljivega pomnilnika v procesorju. Grafični krmilnik bo torej večkrat zapored zajel del zaslonske slike v svoj pomnilnik, pri prenosu v pomnilnik procesorja pa bomo ta del slike še stiskali. Za stiskanje uporabimo kodiranje RLE. Gre za enostaven način stiskanja brez izgub, kjer namesto surovega zaporedja barv slikovnih pik, zapišemo, kolikokrat se posamezna barva v zaporedju slikovnih pik ponovi (npr. zaporedje AAAABBCDDDDD bi stisnili v 4A2B1C5D) [9].

3.6 Prenos s kontrolno vsoto

Trenutna rešitev deluje dobro na vseh obstoječih napravah, vendar kljub zanesljivosti omrežja CAN obstaja primer, ki lahko vodi do nepopolnega prenosa zaslonske slike. V obstoječi programski opremi vgrajenih naprav je vrsta za pošiljanje sporočil CAN implementirana tako, da v primeru polne vrste novo sporočilo enostavno zavržemo. To se lahko zgodi tudi med prenosom zaslonske slike do aplikacije, če v krajšem časovnem obdobju več programskih modulov poskuša poslati večjo količino sporočil v omrežje CAN.

Potrebujemo torej mehanizem, ki bo omogočil zaznavanje nepopolnih prenosov. Podatkom dodamo kontrolno vsoto, ki se izračuna najprej na strani pošiljatelja in se nato ob podatkih pošlje do prejemnika. Ta iz prejetih podatkov še sam izračuna kontrolno vsoto in jo primerja s tisto, ki jo je izračunal in poslal pošiljatelj. Kadar se kontrolni vsoti ne ujemata, vemo, da prenos ni bil uspešen in lahko še enkrat zahtevamo prenos. V našem primeru za kontrolno vsoto uporabimo kar števec sporočil CAN.

3.7 Dodatne funkcionalnosti

Kot zadnjo izboljšavo dodamo obstoječemu prototipu še možnost identifikacije razpoložljivih vgrajenih naprav v omrežju CAN in naslavljanje zahtevkov. Poleg tega v aplikaciji omogočimo ločeno pošiljanje zahtevkov za zajem zaslonske slike v pomnilnik procesorja vgrajene naprave in pošiljanje zahtevkov za prenos zadnje zajete slike iz vgrajene naprave do aplikacije. Prenesena slika je v aplikaciji najprej na voljo za predogled, s klikom na gumb pa jo lahko shranimo na disk. Za boljšo uporabniško izkušnjo se po vsaki zahtevani akciji v večje tekstovno polje izpiše vrstica z izidom akcije, ob daljših akcijah, kot je prenos slike, pa je grafično prikazan tudi napredek.

Poglavje 4

Končna implementacija

V tem poglavju je podrobneje opisana zadnja različica rešitve iz prejšnjega poglavja. Predstavljen je programski modul za vgrajeno napravo, aplikacija za osebni računalnik in komunikacijski protokol med njima.

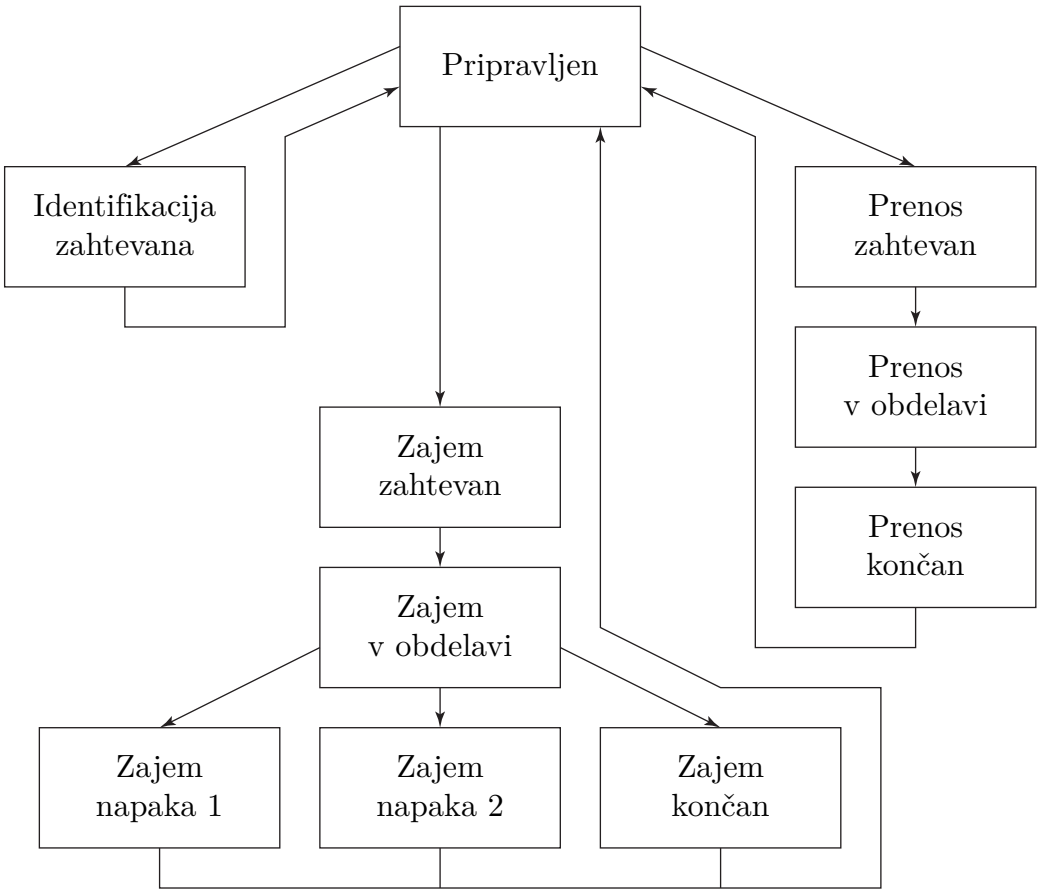
4.1 Programski modul za vgrajeno napravo

Obstoječi programski opremi za vgrajeno napravo smo dodali nov modul, ki je vključen zgolj s klici funkcij:

- `void screenshoter_process_screenshot_request(void)`,
- `void screenshoter_parse_message(can_msg *received)`,
- `void screenshoter_send_message(void)`.

Prva funkcija je namenjena delu z grafičnim krmilnikom in jo kličemo vsako iteracijo glavne zanke v opravilu za grafiko, v enem od stanj pa ta izvrši zajem zaslonske slike iz grafičnega krmilnika v pomnilnik procesorja. Drugo funkcijo kličemo v opravilu za komunikacijo, kadar iz vrste prejetih sporočil CAN prevzamemo novo sporočilo, kot argument pa funkciji podamo kazalec na omenjeno sporočilo. Tretja funkcija je tudi del opravila za komunikacijo in jo kličemo vsako iteracijo glavne zanke, ta pa po potrebi dodaja nova sporočila CAN v vrsto za pošiljanje.

Delovanje modula lahko prikažemo s končnim avtomatom. Vsa možna stanja in prehodi (brez označenih pogojev za prehajanje) so prikazani na sliki 4.1, v nadaljevanju pa so stanja še opisana.



Slika 4.1: Končni avtomat, ki prikazuje delovanje modula

Pripravljen

To je začetno stanje, kjer modul čaka na morebitni zahtevek za eno od treh razpoložljivih akcij. Tega prejme od aplikacije za osebni računalnik v obliki sporočila CAN, kjer je s podatki v sporočilu določeno, za kateri zahtevek gre. V primeru zahtevka za identifikacijo naprave modul preide v stanje *Identifikacija zahtevana*, v primeru zahtevka za zajem zaslonske slike iz grafičnega krmilnika v pomnilnik procesorja preide v

stanje *Zajem zahtevan*, v primeru zahtevka za prenos zajete slike do aplikacije pa preide v stanje *Prenos zahtevan*.

Identifikacija zahtevana

V tem stanju se naprava identificira s sporočilom CAN, ki vsebuje 6 bajtov dolgo identiteto naprave. Ta je v modulu določena kot konstanta, ki jo za različne naprave posebej definiramo (uporabimo ASCII kodirane znake, da bo aplikacija lepše prikazala seznam razpoložljivih naprav). Ko modul omenjeno sporočilo CAN uspešno doda v vrsto za pošiljanje, takoj preide nazaj v stanje *Pripravljen*.

Zajem zahtevan

Tudi v tem stanju modul v omrežje CAN pošlje sporočilo, s katerim potrdi, da je bil zahtevek za zajem prejet, za tem pa preide v stanje *Zajem v obdelavi*.

Zajem v obdelavi

Celoten proces zajemanja zaslonske slike iz grafičnega krmilnika v pomnilnik procesorja se izvrši v opravilu za grafiko, kar med drugim tudi preprečuje izris nove vsebine na zaslonu med posameznimi delnimi zajemi. Najprej modul izračuna, kakšna je največja možna velikost slike, ki jo lahko zajame v pomnilnik grafičnega krmilnika. Če je pomnilnika premalo tudi za najmanjši delni zajem, preide v stanje *Zajem napaka 1*, sicer pa začne z delnimi zajemi. Vsak del slike zajame tako, da najprej grafičnemu krmilniku pošlje ukaz za zajem dela slike v njegov pomnilnik. Ko se ta ukaz izvrši, modul začne s prenosom zajetega dela slike iz pomnilnika grafičnega krmilnika v pomnilnik procesorja, ob tem pa sliko še stiska. Delne zajeme ponavlja, dokler ne zajame celotne slike. Če med tem že prej zapolni ves razpoložljivi pomnilnik procesorja, zajemanje prekine in preide v stanje *Zajem napaka 2*, sicer pa ob uspešno zaključenem zajemu celotne slike preide v stanje *Zajem končan*.

Zajem napaka 1

Stanje označuje, da je pri zajemu slike prišlo do napake zaradi po-

manjkanja pomnilnika grafičnega krmilnika. V omrežje CAN modul pošlje sporočilo, ki označuje omenjeno napako, nato pa preide v stanje *Pripravljen*.

Zajem napaka 2

Napaka 2 kaže na pomanjkanje pomnilnika procesorja. Na enak način kot pri napaki 1 tudi tukaj modul pošlje sporočilo CAN in za tem preide v stanje *Pripravljen*.

Zajem končan

Tudi v primeru uspešnega zajema modul to sporoči aplikaciji za osebni računalnik s sporočilom CAN, nato pa spet preide v stanje *Pripravljen*.

Prenos zahtevan

V tem stanju modul potrdi prejet zahtevek s sporočilom CAN, ob enem pa v sporočilo vključi še velikost stisnjenih podatkov v bajtih ter širino in višino slike v slikovnih pikah. Velikost stisnjenih podatkov omogoča aplikaciji rezervacijo primerno velikega dela pomnilnika za shranjevanje prejetih podatkov, z znano širino in višino slike pa lahko aplikacija pripravi slikovni objekt. Ko modul pošlje omenjeno sporočilo CAN, začne s prenosom tako, da preide v stanje *Prenos v obdelavi*.

Prenos v obdelavi

Prenos zaporedja podatkov se izvede tako, da modul v omrežje CAN zaporedoma pošilja sporočila, pri čemer v vsako vključi delček podatkov (protokol CAN namreč omogoča največ 8 bajtov na sporočilo). Med pošiljanjem se računa še kontrolna vsota, ki bo v našem primeru enaka številu poslanih sporočil CAN. Da z relativno dolgim prenosom modul ne bi blokiral ostalih aktivnosti, vsako iteracijo glavne zanke v opravilu za komunikacijo pošlje kvečjemu 20 sporočil CAN. Ko pošlje celotno zaporedje podatkov, preide v stanje *Prenos končan*.

Prenos končan

Po končanem prenosu modul pošlje sporočilo v omrežje CAN, s kate-

rim aplikaciji sporoči, da je prenos zaključen, v sporočilo pa vključi še kontrolno vsoto, da bo aplikacija lahko preverila prejete podatke. Na koncu spet preide v stanje *Pripravljen*.

4.2 Aplikacija za osebni računalnik

Aplikacija ponuja grafični uporabniški vmesnik, preko katerega lahko uporabnik zahteva posamezne akcije ter spremlja odzive. Ta je prikazan na sliki 4.2, v nadaljevanju pa so posamezni elementi podrobneje opisani. Pri načrtovanju vmesnika smo si pomagali s priporočili iz vira [12].

Slika 4.2: Uporabniški vmesnik aplikacije

Gumb *Identify available devices*

Ob kliku na gumb aplikacija pošlje zahtevek za identifikacijo. Na odzive naprav čaka 1 sekundo, vsako napravo, ki se v tem času odzove na zahtevek za identifikacijo, pa shrani v seznam razpoložljivih naprav.

Seznam razpoložljivih naprav

Seznam omogoča pregled razpoložljivih naprav, uporabnik pa lahko iz seznama izbere napravo, na katero bodo naslovljeni nadaljnji zahtevki za zajem in prenos zaslonske slike.

Gumb *Take a screenshot*

Ob kliku na gumb aplikacija pošlje izbrani napravi zahtevek za zajem zaslonske slike. Pri tem počaka kvečjemu 1 sekundo na potrditev zahtevka in nato v primeru uspešne potrditve počaka še na sporočilo s statusom akcije - napaka 1, napaka 2, zajem končan.

Gumb *Download image*

Ob kliku na gumb aplikacija pošlje izbrani napravi zahtevek za prenos zajete slike. Pri tem spet počaka kvečjemu 1 sekundo na potrditev zahtevka in nato v primeru uspešne potrditve rezervira del pomnilnika, v katerega se bodo med prenosom shranjevali prejeti podatki. Konec prenosa je označen s posebnim sporočilom naprave, to sporočilo pa vsebuje tudi kontrolno vsoto. Če je bil prenos uspešen, aplikacija iz prejetih podatkov pripravi sliko in jo prikaže v polju za prikaz zaslonske slike.

Prikaz napredka

Ta element prikazuje napredek pri posameznih akcijah, najbolje pa pride do izraza pri prenosu zajete slike, saj lahko točno pokažemo delež končanega prenosa.

Tekstovno polje z izidi akcij

V to polje aplikacija izpisuje izide posameznih akcij ter opozorila, če npr. pozabimo izbrati napravo za naslavljanje zahtevkov, poskusimo

shraniti sliko, preden jo sploh prenesemo, zahtevamo neko akcijo, ko je druga akcija še v teku ...

Polje za prikaz zaslonske slike

To polje je namenjeno prikazu zadnje prenesene slike in nam omogoča predogled, preden sliko shranimo na disk.

Gumb *Save as*

Ob kliku na gumb aplikacija ponudi običajen dialog za shranjevanje datotek, v katerem lahko izberemo lokacijo, ime in format za shranjevanje zadnje prenesene slike.

4.3 Komunikacijski protokol

Celoten protokol komuniciranja med programskim modulom za vgrajeno napravo in aplikacijo za osebni računalnik je posredno razviden že iz prejšnjih dveh razdelkov tega poglavja, v tem razdelku pa je ta naveden v obliki vseh možnih sporočil CAN. Pri vseh sporočilih uporabljamo razširjen podatkovni okvir in vedno enak 29-bitni identifikator, tip sporočila pa je vedno določen s prvim bajtom. Če neki podatek obsega več bajtov, je to označeno s podpisanim indeksom, kjer 0 označuje LSB, najvišji indeks pa MSB.

Prvi del protokola je namenjen identifikaciji razpoložljivih vgrajenih naprav v omrežju CAN. Osebni računalnik lahko pošlje zahtevek za identifikacijo (tabela 4.1), vse razpoložljive vgrajene naprave pa bodo odgovorile z identifikacijskim odzivom (tabela 4.2).

Pošiljatelj	Osebni računalnik			
Dolžina	1			
Podatki	0x10	/	/	/
	/	/	/	/

Tabela 4.1: Zahtevek za identifikacijo

Pošiljatelj	Vgrajena naprava			
Dolžina	7			
Podatki	0x11	ID_5	ID_4	ID_3
	ID_2	ID_1	ID_0	/

Tabela 4.2: Identifikacijski odziv

Drugi del protokola obsega sporočila za zajem zaslonske slike. Osebni računalnik lahko pošlje zahtevek, v katerem navede identiteto izbrane naprave (tabela 4.3). Odzove se le naprava z ujemajočo se identiteto, kot odgovor pa osebni računalnik najprej prejme sporočilo, ki označuje začetek izbrane akcije (tabela 4.4). Na koncu prejme še status, s katerim se je akcija zaključila - napaka 1, napaka 2, zajem končan (tabela 4.5).

Pošiljatelj	Osebni računalnik			
Dolžina	7			
Podatki	0x20	ID_5	ID_4	ID_3
	ID_2	ID_1	ID_0	/

Tabela 4.3: Zahtevek za zajem

Pošiljatelj	Vgrajena naprava			
Dolžina	1			
Podatki	0x21	/	/	/
	/	/	/	/

Tabela 4.4: Začetek zajema

Pošiljatelj	Vgrajena naprava			
Dolžina	2			
Podatki	0x22	Status	/	/
	/	/	/	/

Tabela 4.5: Konec zajema

Za prenos slike osebni računalnik naslovi vgrajeno napravo na enak način kot pri zajemu slike (tabela 4.6). Kot odgovor najprej prejme sporočilo, ki označuje začetek prenosa (tabela 4.7). To sporočilo vsebuje velikost stisnjenih podatkov za prenos v bajtih ter višino in širino zaslonske slike v slikovnih pikah. Za tem sledi več zaporednih sporočil s podatki (tabela 4.8). Vsako tako sporočilo vsebuje dve RLE kodirani zaporedji slikovnih pik - število zaporednih slikovnih pik prve barve in enako tudi za naslednje zaporedje druge barve, kjer je posamezna barva kodirana s 16 biti v formatu RGB565. Za konec prenosa osebni računalnik prejme sporočilo, ki vsebuje kontrolno vsoto, s katero lahko preveri uspešnost prenosa (tabela 4.9).

Pošiljatelj	Osebni računalnik			
Dolžina	7			
Podatki	0x30	ID_5	ID_4	ID_3
	ID_2	ID_1	ID_0	/

Tabela 4.6: Zahtevek za prenos

Pošiljatelj	Vgrajena naprava			
Dolžina	7			
Podatki	0x31	$Velikost_1$	$Velikost_0$	$Širina_1$
	$Širina_0$	$Višina_1$	$Višina_0$	/

Tabela 4.7: Začetek prenosa

Pošiljatelj	Vgrajena naprava			
Dolžina	7			
Podatki	0x32	Število 0	Barva 0_1	Barva 0_0
	Število 1	Barva 1_1	Barva 1_0	/

Tabela 4.8: Prenos zaporedja podatkov

Pošiljatelj	Vgrajena naprava			
Dolžina	5			
Podatki	0x33	K. vsota$_3$	K. vsota$_2$	K. vsota$_1$
	K. vsota$_0$	/	/	/

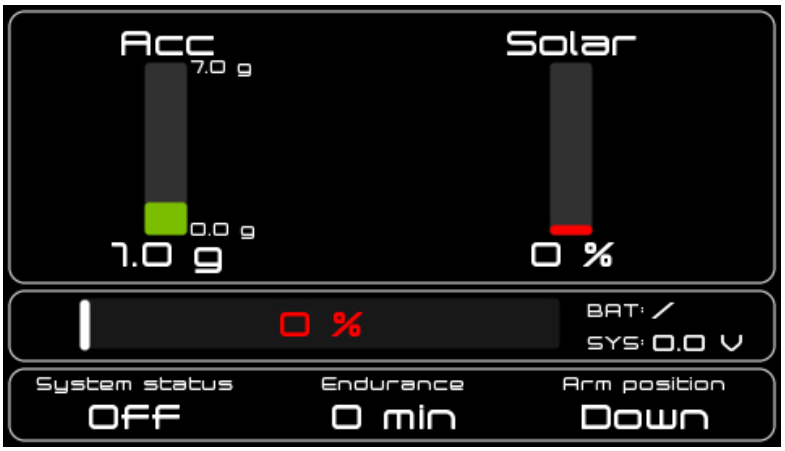
Tabela 4.9: Konec prenosa

Poglavje 5

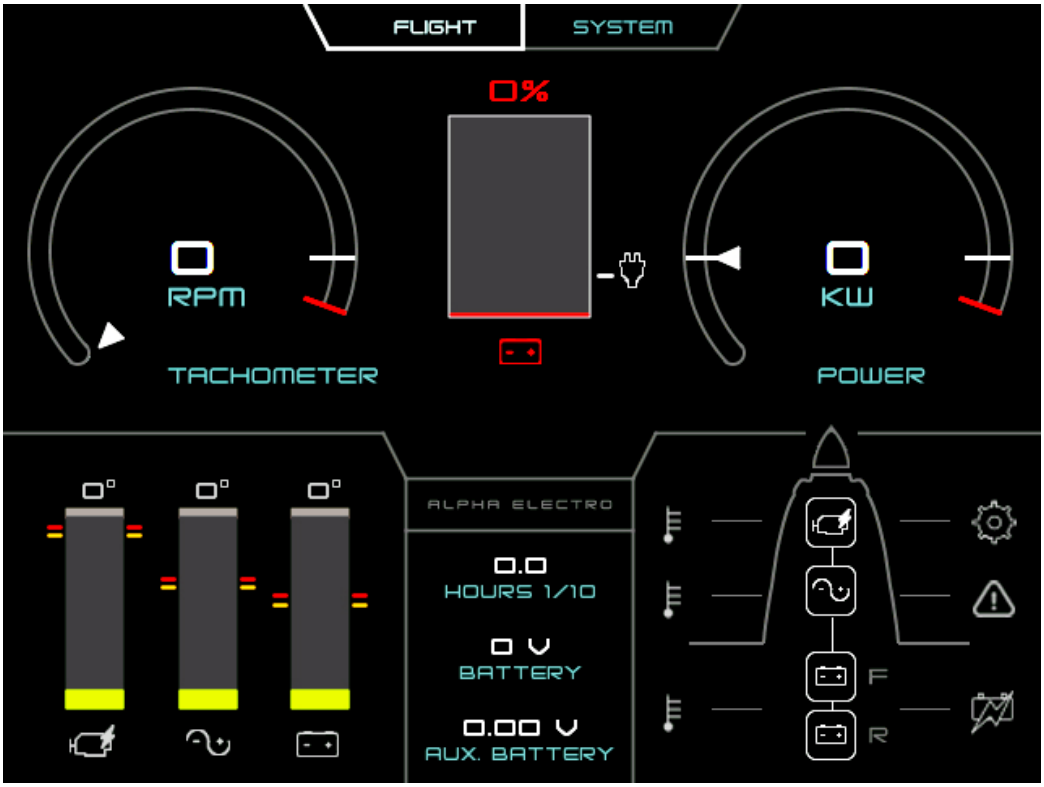
Testiranje

Delovanje sistema smo preizkusili na dveh različnih vgrajenih napravah podjetja Pipistrel - P550 in P600. Obe napravi in osebni računalnik smo povezali v omrežje CAN. V aplikaciji smo najprej zahtevali identifikacijo naprav, napravi pa sta se po zahtevku za identifikacijo pojavili na seznamu razpoložljivih naprav.

Iz obeh naprav smo zajeli, prenesli in shranili zaslonsko sliko ter se prepričali, da zajeti sliki res ustrezata zaslonskim slikam iz časa zajemanja in ne vsebujeta kakršnihkoli napak. Slika 5.1 prikazuje zaslonsko sliko, zajeto iz naprave P550, slika 5.2 pa zaslonsko sliko, zajeto iz naprave P600.



Slika 5.1: Zaslonska slika, zajeta iz naprave P550



Slika 5.2: Zaslonska slika, zajeta iz naprave P600

Zajem zaslonske slike lahko traja nekaj sekund, prenos slike do aplikacije pa tudi nekaj 10 sekund. Za boljši občutek bralca smo na obeh napravah te čase izmerili. Vsako meritev smo izvedli 10-krat, rezultati pa so navedeni v tabeli 5.1. Ker ima naprava P550 nekoliko manjši zaslon (480 x 272 slikovnih pik) v primerjavi z napravo P600 (640 x 480 slikovnih pik), so temu primerno različni tudi izmerjeni časi.

Naprava	Čas zajema	Čas prenosa
P550	1,210 ± 0,037 s	5,456 ± 0,016 s
P600	2,840 ± 0,028 s	14,633 ± 0,045 s

Tabela 5.1: Rezultati meritev

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi je predstavljen razvoj sistema za zajem zaslonske slike iz vgrajenih naprav podjetja Pipistrel.

Pripravili smo aplikacijo za osebni računalnik, s katero lahko uporabnik izvede več akcij. Najprej lahko poizve, katere naprave so v omrežju CAN razpoložljive za zajem zaslonske slike. Od izbrane naprave lahko zahteva zajem, ob tem pa se zaslonska slika shrani v pomnilnik procesorja vgrajene naprave. Zadnje zajeto zaslonsko sliko lahko uporabnik prenese do aplikacije, kjer je na voljo predogled, na koncu pa jo lahko shrani kot datoteko v formatu PNG.

Poleg aplikacije smo pripravili tudi programski modul za vgrajeno napravo, ki ga lahko enostavno dodamo obstoječim vgrajenim napravam. Modul omogoča odzive na vse razpoložljive akcije aplikacije. Zaslonska slika se zaradi pomanjkanja pomnilnika grafičnega krmilnika zajema v več delih, posamezni deli slike pa se ob shranjevanju v pomnilnik procesorja vgrajene naprave še stiskajo.

Sistem ponuja še nekaj prostora za izboljšave. Najbolj problematičen je prenos zaslonske slike od vgrajene naprave do osebnega računalnika. Ta prenos poteka preko omrežja CAN, katerega prvotni namen je robustno pošiljanje manjših količin podatkov (največ 8 bajtov na sporočilo) in ne pošiljanje večjih podatkovnih tokov. Slednje je sicer izvedljivo, vendar lahko

prenos slike traja tudi nekaj 10 sekund. Žal smo v okviru te diplomske naloge omejeni z obstoječimi napravami, kjer zajem zaslonske slike ni bil posebej predviden in posledično naprave nimajo pripravljenega primernega vmesnika za prenos slike do osebnega računalnika. Sicer bi lahko kot primer izboljšave na vgrajeno napravo dodali režo za pomnilniško kartico. Naprava bi lahko sliko že takoj ob prenosu iz grafičnega krmilnika shranjevala v primernem formatu na pomnilniško kartico. Aplikacijo bi torej uporabili le za pošiljanje zahtevkov, lahko bi pa sistem v celoti prestavili na napravo, če bi zahtevek za zajem prožili s posebno kombinacijo tipk na napravi.

Kljub nekoliko daljšemu prenosu slike preko omrežja CAN pa je taka rešitev za podjetje bolj smiselna kot poseganje v strojno opremo obstoječih naprav. Podjetje Pipistrel namreč ta sistem že uporablja, uporabniki pa z njim nimajo težav.

Literatura

[1] David Avison and Guy Fitzgerald. Rapid application development (RAD). In *Information Systems Development: Methodologies, Techniques and Tools*, pages 128–134. McGraw-Hill Education, 2006.

[2] Richard Barry. The FreeRTOS Kernel. Dosegljivo: `https://www.freertos.org`, 2019. [Dostopano: januar 2019].

[3] BRTChip. FT81X Series Programmers Guide. Dosegljivo: `https://brtchip.com/wp-content/uploads/Support/Documentation/Programming_Guides/ICs/EVE/FT81X_Series_Programmer_Guide.pdf`, 2018. [Dostopano: januar 2019].

[4] IAR Systems. IAR Embedded Workbench. Dosegljivo: `https://www.iar.com/iar-embedded-workbench`, 2019. [Dostopano: januar 2019].

[5] Microsoft. Windows Forms. Dosegljivo: `https://docs.microsoft.com/en-us/dotnet/framework/winforms`, 2017. [Dostopano: januar 2019].

[6] Microsoft. Bitmap Class. Dosegljivo: `https://docs.microsoft.com/en-us/dotnet/api/system.drawing.bitmap`, 2019. [Dostopano: januar 2019].

[7] Microsoft. Visual studio IDE. Dosegljivo: `https://visualstudio.microsoft.com/vs`, 2019. [Dostopano: januar 2019].

[8] Motorola. SPI Block Guide V03.06. Dosegljivo: https://opencores.org/usercontent/doc/1499360489, 2003. [Dostopano: januar 2019].

[9] James D Murray and William vanRyper. Run-Length Encoding (RLE). In *Encyclopedia of Graphics File Formats*, pages 160–170. O'Reilly Media, 1996.

[10] PEAK-System Technik GmbH. PCAN-Basic. Dosegljivo: https://www.peak-system.com/PCAN-Basic.239.0.html?L=1, 2018. [Dostopano: januar 2019].

[11] Pipistrel. Taurus Electro. Dosegljivo: https://www.pipistrel.si/plane/taurus-electro/overview, 2014. [Dostopano: januar 2019].

[12] Ben Shneiderman and Catherine Plaisant. Content Organization. In *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, pages 262–268. Pearson, 2010.

[13] STMicroelectronics. Description of STM32F7 HAL and Low-layer drivers. Dosegljivo: https://www.st.com/resource/en/user_manual/dm00189702.pdf, 2017. [Dostopano: januar 2019].

[14] STMicroelectronics. STM32F7x7. Dosegljivo: https://www.st.com/en/microcontrollers/stm32f7x7.html, 2019. [Dostopano: januar 2019].

[15] Texas Instruments. Introduction to the Controller Area Network (CAN). Dosegljivo: http://www.ti.com/sloa101-aaj, 2016. [Dostopano: januar 2019].