Development of an Interactive Annunciator Panel

Bachelorarbeit von
Lukas Klass
IRS-18-S-052

Betreuer:
Prof. Dr. rer. nat. Alfred Krabbe
Dr. rer. nat. Holger Jakob

Institut für Raumfahrtsysteme, Universität Stuttgart
Mai 2018

UNIVERSITÄT STUTTGART
INSTITUT FÜR RAUMFAHRTSYSTEME
im Raumfahrtzentrum Baden-Württemberg (RZBW)

Pfaffenwaldring 29 · 70569 Stuttgart · Tel.: (0711) 685-62375 · Fax: (0711) 685-63596 · www.irs.uni-stuttgart.de

Aufgabenstellung Bachelorarbeit

für Herrn Lukas Klass

Development of an interactive Annunciator Panel for the SOFIA Telescope

Motivation:

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is an airborne observatory with a German built 2.7-meter telescope onboard. At altitudes up to 15 km it performs astrophysical measurements at wavelengths of 1 to 300 µm which cannot be observed from the ground due to atmospheric water vapor absorption. In the visible spectral range, SOFIA takes advantage of the short term and worldwide mobility of the aircraft to observe localized astronomical phenomena such as eclipses and stellar occultation by bodies of our solar system.

The topic of this bachelor thesis is to extend the current software of the annunciation panel to an interactive control device. This shall be derived from an existing concept of operation and form the base for future software updates. The candidate has to gain a sound knowledge about the functionality of the telescope's subsystems and how to program the annunciator panel in the language C. Going through the different steps of a software development plan will be an essential part of this thesis. This includes verifying the airworthiness of the new software, as the annunciation panel is mission critical component of SOFIA telescope.

The thesis consists of the following tasks:

- Becoming familiar with the hardware architecture and the software layout.
- Defining a project plan and system requirements
- Developing the new software load
- Documenting the applied changes and the lessons learned
- Define testing procedures and verify the airworthiness of the new software load
- Install the new software load in the aircraft

Empfangsbestätigung:
Ich bestätige hiermit, dass ich die Aufgabenstellung sowie die rechtlichen Bestimmungen und die Studien- und Prüfungsordnung gelesen und verstanden habe.

Betreuer intern: Dr. rer. nat. Holger Jakob
Bearbeitungsbeginn: 01.12.2017

Einzureichen spätestens: 31.05.2018

Prof. Dr. rer. nat. Alfred Krabbe
(Verantwortlicher Hochschullehrer)

L. Klass

Unterschrift des/der Studierenden

Erklärungen

Hiermit versichere ich, **Klass**, **Lukas,** dass ich diese ***Bachelorarbeit*** selbstständig mit Unterstützung der Betreuer angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit oder wesentliche Bestandteile davon sind weder an dieser noch an einer anderen Bildungseinrichtung bereits zur Erlangung eines Abschlusses eingereicht worden.

Ich erkläre weiterhin, bei der Erstellung der Arbeit die einschlägigen Bestimmungen zum Urheberschutz fremder Beiträge entsprechend den Regeln guter wissenschaftlicher Praxis[1] eingehalten zu haben. Soweit meine Arbeit fremde Beiträge (z.B. Bilder, Zeichnungen, Textpassagen etc.) enthält, habe ich diese Beiträge als solche gekennzeichnet (Zitat, Quellenangabe) und eventuell erforderlich gewordene Zustimmungen der Urheber zur Nutzung dieser Beiträge in meiner Arbeit eingeholt. Mir ist bekannt, dass ich im Falle einer schuldhaften Verletzung dieser Pflichten die daraus entstehenden Konsequenzen zu tragen habe.

Palmdale, CA, 02.05.2018 L. Klass

.....

Ort, Datum, Unterschrift

Hiermit erkläre ich mich damit einverstanden, dass meine ***Bachelorarbeit*** zum Thema:

Development of an interactive Annunciator Panel for the SOFIA Telescope

in der Institutsbibliothek des Instituts für Raumfahrtsysteme ohne Sperrfrist öffentlich zugänglich aufbewahrt und die Arbeit auf der Institutswebseite sowie im Online-Katalog der Universitätsbibliothek erfasst wird. Letzteres bedeutet eine dauerhafte, weltweite Sichtbarkeit der bibliographischen Daten der Arbeit (Titel, Autor, Erscheinungsjahr, etc.).

Nach Abschluss der Arbeit werde ich zu diesem Zweck meinen Betreuern neben dem Prüfexemplar eine weitere gedruckte sowie eine digitale Fassung übergeben.

Der Universität Stuttgart übertrage ich das Eigentum an diesen zusätzlichen Fassungen und räume dem Institut für Raumfahrtsysteme an dieser Arbeit und an den im Rahmen dieser Arbeit von mir erzeugten Arbeitsergebnissen ein kostenloses, zeitlich und örtlich unbeschränktes, einfaches Nutzungsrecht für Zwecke der Forschung und der Lehre ein. Falls in Zusammenhang mit der Arbeit Nutzungsrechtsvereinbarungen des Instituts mit Dritten bestehen, gelten diese Vereinbarungen auch für die im Rahmen dieser Arbeit entstandenen Arbeitsergebnisse.

Palmdale, CA, 02.05.2018 L. Klass

.....

Ort, Datum, Unterschrift

[1] Nachzulesen in den DFG-Empfehlungen zur „Sicherung guter wissenschaftlicher Praxis“ bzw. in der Satzung der Universität Stuttgart zur „Sicherung der Integrität wissenschaftlicher Praxis und zum Umgang mit Fehlverhalten in der Wissenschaft“

Preface

First of all, I thank my mentor Prof. Dr. rer. nat. Alfred Krabbe at the University of Stuttgart as well as my supervisor Dr. rer. nat. Holger Jakob for their friendly and helpful support during my bachelor thesis.

Furthermore I thank Dipl.-Inf. Oliver Rohe sincerely for his support in the C programming language, setting up the Makefile and his SD-Card bootloaders. Without his Makefile, the compiling process would have been much more complicated.

For letting me be part of the awesome DSI team, I would like to thank Michael Beck, Christian Fischer, Nadine Fischer, Michael Huetwohl, Yannick Lammen, Sarah Peter, Andreas Reinacher, Andreas Siggelkow, Alexander Steiner, Rainer Strecker, Oliver Zeile and the rest of the DSI team. I had a great time both at work and in our free time. I really enjoyed goind on adventures with you.

In addition, I thank my parents and family for letting me go to Palmdale and for their great support.

I thank my girlfriend Anja Mrzyglod for coming with me to the US and for her great support. I really enjoyed traveling through California with her. I wouldn't have done it without her.

The German Academic Exchange Service (DAAD) supported me with a PROMOS scholarship, for which I would like to thank for. It helped me to finance this amazing experience in the United States of America.

Finally, I thank Michael Stock and his company Stock Flight Systems for their support during the hardware analysis and the troubleshooting.

Contents

Preface i

Abstract v

Zusammenfassung vii

Acronyms ix

1 Introduction 1
- 1.1 SOFIA . . . 2
- 1.2 Telescope Assembly . . . 4
- 1.3 Telescope Optics . . . 5
- 1.4 Telescope Software and Electronics . . . 6
- 1.5 Annunciator Panel . . . 8

2 System Analysis 11
- 2.1 Hardware . . . 11
  - 2.1.1 Display and GPU . . . 11
  - 2.1.2 Spartan-3 FPGA . . . 12
  - 2.1.3 Periphery . . . 14
- 2.2 Software . . . 15
  - 2.2.1 MicroBlaze Bootloaders . . . 16
  - 2.2.2 CAN Application . . . 17
  - 2.2.3 Display Application . . . 18

3 Preparatory Work 19
- 3.1 Software Design Process . . . 19
- 3.2 Source Code Repository . . . 20
- 3.3 MicroBlaze Toolchain . . . 20
- 3.4 Makefile & Linker Script . . . 21
- 3.5 SD-Card Bootloader . . . 21

4 Implementation & Findings 23
- 4.1 State Model of the new Software . . . 24
- 4.2 Annunciation Colors . . . 25
- 4.3 Acknowledge & Refresh . . . 25
- 4.4 UTC Clock . . . 25
- 4.5 Housekeeping Page . . . 26

4.6 Error Handling 27
4.7 Continuous Built-In Test 29
4.8 Built-In Test on Demand 30
4.9 Task Scheduler 31
4.10 Findings 34
4.10.1 System Performance 34
4.10.2 Boot Timing Issues 36
4.10.3 CAN Bus Problems 37

5 Conclusion **39**
5.1 Future Updates & Upgrades 39
5.2 Alternative Systems 40

Glossary **41**

Bibliography **43**

Abstract

Observing with SOFIA requires a quick and reliable way to make its operators aware of the current telescope state, for what the Annunciator Panel (ANPA) is the central annunciation system. So far, the panel only served as a passive system without any possibility for user interaction. Furthermore, it didn't offer any indication of its own failure state.
For these reasons, the ANPA's capabilities are extended to an interactive and a more reliable system in the scope of this thesis. This includes the capability to acknowledge new alerts and to request a refresh of the display from the Master Control Processor (MCP). Test mechanisms for error detection are developed, which check the system's health state both continuously and on demand. As some housekeeping parameters are of special importance during troubleshooting and maintenance on the ground by the DSI personnel, two housekeeping pages are included in the new software. Additionally, performance issues of the current system are investigated and different solutions developed.
So far, only insufficient hardware and software documentation existed for the Annunciator Panel. However, the manufacturer of the ANPA, Stock Flight Systems, delivered the complete development environment for the software development. Based on this, the functionality of hardware and software is analyzed and documented.
As the Annunciator Panel is a mission critical telescope system on the aircraft, the DSI development process is applied to ensure the airworthiness of the new software. In this context, the requirements for the system are identified and the design of the new functionalities is documented in detail. Subsequently, the new software is tested in cooperation with NASA Quality Assurance (QA), using a dedicated testing procedure, in order to verify the airworthiness of the new system.

Zusammenfassung

Effizientes Beobachten mit SOFIA setzt eine zuverlässige Anzeige des Teleskop-Status voraus, wofür das Annunciator Panel (ANPA) das zentrale Anzeigeelement darstellt. Bisher diente das Panel lediglich als passives Anzeigesystem ohne jegliche Interaktionsmöglichkeit. Darüber hinaus bot es keinerlei Fehlerindikation, die einen Rückschluss auf korrektes oder fehlerhaftes Verhalten ermöglichte.
Aus diesen Gründen wird im Rahmen dieser Bachelorarbeit das ANPA zu einem interaktiven und zuverlässigerem System erweitert. Dies beinhaltet die Möglichkeit neue Statusmeldungen zu bestätigen und eine Aktualisierung des Bildschirminhaltes vom MCP anzufordern. Ebenfalls werden Testmechanismen zur Fehlererkennung entwickelt, die sowohl kontinuierlich als auch auf Abfrage die Funktionsbereitschaft des Systems prüfen. Da während des Betriebs und der Wartung des Teleskops am Boden durch das DSI Personal einige Sensormesswerte wichtig sind, werden zwei Übersichtsseiten zur Darstellung dieser Werte in der neuen Software vorgesehen. Zusätzlich wird die Leistungsfähigkeit des Systems untersucht und verschiedene Ansätze zur Verbesserung entwickelt.
Bisher existierte für das Annunciator Panel nur unzureichende Hardware und Software Dokumentation. Vom Hersteller wurde jedoch die gesamte Entwicklungsumgebung für die Entwicklung der Software geliefert. Anhand dieser wird zunächst die Funktionsweise der Hardware und bisherigen Software nachvollzogen und dokumentiert. Basierend auf diesem Wissen können anschließend die neuen Anforderungen mithilfe einer simulierten Flugzeugumgebung im HiL-Labor implementiert werden.
Da es sich beim Annunciator Panel um ein missionskritisches Teleskopsystem im Flugzeug handelt, wird für die Entwicklung der neuen Software der DSI Entwicklungsprozess angewendet. In diesem Rahmen werden zuerst die Anforderungen an das System herausgearbeitet und das Design der neuen Funktionalitäten genau dokumentiert. Anschließend wird das neue System gemeinsam mit der NASA Qualitätssicherung mithilfe einer Testprozedur geprüft, um die Betriebstauglichkeit für SOFIA nachzuweisen.

Acronyms

ANPA	Annunciator Panel
ATCU	Attitude Control Unit
BIT	Built-In Test
BRAM	Block Random Access Memory
CAN	Controller Area Network
ConOps	Concept of Operations
CPU	Central Processing Unit
CVS	Concurrent Version System
DCC	Diagnostics and Control Computer
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DPRAM	Dual-Port Random Access Memory
DSI	Deutsches SOFIA Institut
FCM	Focus Center Mechanism
FFI	Fine Field Imager
FPI+	Focal Plane Imager Plus
FPU	Floating Point Unit
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IRIG	Inter Range Instrumentation Group Timecode
ISR	Interrupt Service Routine
KAO	Kuiper Airborne Observatory
LMB	Local Memory Bus
MCCS	Mission Controls and Communications System
MCP	Master Control Processor

NASA	National Aeronautics and Space Agency
NECS	Network Extended Control System
PDU	Power Distribution Unit
PLB	Processor Local Bus
PM	Primary Mirror
PWCU	Pressure Window Control Unit
PWM	Pulse Width Modulation
RAM	Random Access Memory
RDACU	Rotation Drive Assembly Control Unit
SD-Card	Secure Digital Memory Card
SI	Science Instrument
SM	Secondary Mirror
SMA	Secondary Mirror Assembly
SMCU	Secondary Mirror Control Unit
SMM	Secondary Mirror Mechanism
SMO	SOFIA Science Mission Operations
SOFIA	Stratospheric Observatory for Infrared Astronomy
SRAM	Static Random Access Memory
TA	Telescope Assembly
TASCU	Telescope Assembly Servo Control Unit
TCM	Tilt Chop Mechanism
TM	Tertiary Mirror
TO	Telescope Operator
TRC	Tracker
UPS	Uninterruptable Power Supply
USRA	Universities Space Research Association
UTC	Coordinated Universal Time
V&V	Verification & Validation
VIS	Vibration Isolation System
WFI	Wide Field Imager
Wine	Wine Is Not an Emulator
XCL	Xilinx Cache Link

1 Introduction

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is an airplane based observatory for the Mid and Far Infrared range and the successor of the now shut down Kuiper Airborne Observatory (KAO). As a result of the success of KAO like the discovery of the Uranus rings, scientists decided in 1985 to build an all new and more advanced airborne observatory. SOFIA offers an improved angular and spectral resolution and a better sensitivity compared to the KAO [1].
The development of SOFIA started in late 1996 with the first contracts between the National Aeronautics and Space Agency (NASA) and the Deutsches Zentrum für Luft- und Raumfahrt (DLR) [15]. SOFIA had its first flight in April 2007 and its First Light in May 2010. The Full Operational Capabilities were reached in 2014. As the plane is designed for an operational 20 years, SOFIA can be active until 2034. During about 120 flights [19] of up to ten hours every year, scientists now observe the sky over North America, the Pacific and the Southern Hemisphere.

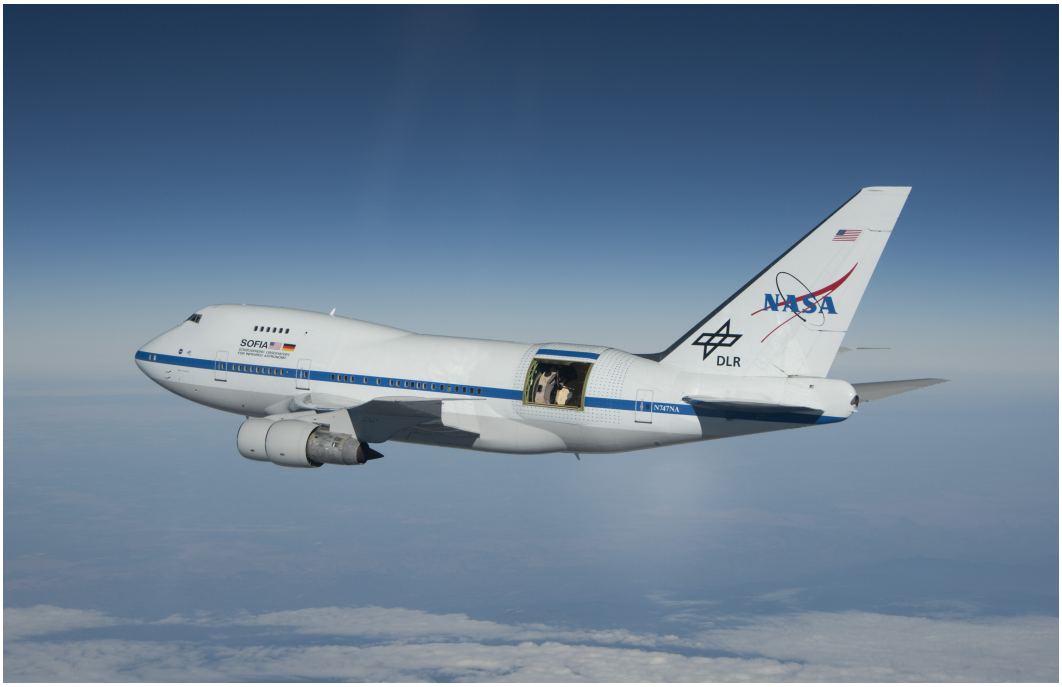


Figure 1.1: Stratospheric Observatory for Infrared Astronomy in flight with the Cavity Door open [21]

1.1 SOFIA

SOFIA is a joint research project of NASA and DLR. It consists of a modified Boeing 747 Special Performance, which carries a 2.7 m diameter Infrared telescope in the rear fuselage.
Compared to a 747-100, the Boeing 747 SP is shorter and lighter, but offers the same wing size. To maintain the flight stability, the empennage has been increased. By virtue of these changes, the airplane is now able to carry more payload or to fly longer than a commercial 747.
In order to allow the telescope a clear view of the sky, several modifications have been carried out on the plane. An open cavity in the aft contains the telescope itself. It is separated from the passenger area and carried by a pressure bulkhead as shown in Figure 1.2. To protect the telescope from any environmental effects like condensation during descent, a shutter door was installed. The door is only opened at night at the operation height of 37000 - 45000 feet [19] or during maintenance.
The Stratospheric Observatory for Infrared Astronomy is financially divided between the two partners by 80 % (NASA) and 20 % (DLR) [3]. Universities Space Research Association (USRA) has been contracted by NASA to maintain the plane and provide the SOFIA Science Mission Operations (SMO). The task of the DLR was to develop, build and now to maintain the Telescope Assembly. In order to fulfill these tasks, the Deutsches SOFIA Institut (DSI) was founded in 2004 at the University of Stuttgart.
As SOFIA is financed by public funds, visitors like science teachers are welcome to experience a flight and the accompanying science.

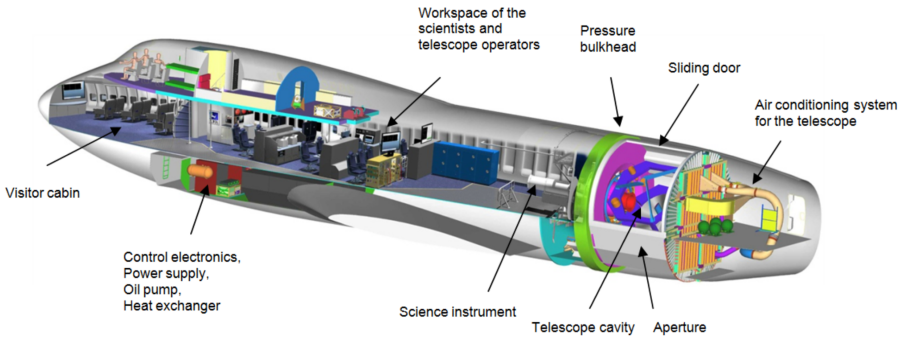


Figure 1.2: Sectional view of SOFIA
The telescope cavity in the back is separated by a pressure bulkhead from the cabin where the Telescope Operator (TO)s and scientists operate SOFIA.

At the operation altitude, SOFIA flys above 99 % of the atmospheres water vapor which absorbs some ranges of the infrared spectrum [1]. Because of this, SOFIA is able to observe a wider spectrum of the Infrared radiation compared to ground stationed observatories like the ones on Mauna Kea, as shown in Figure 1.3.

Although satellite observatories like Herschel, Hubble or James Webb Space Telescope are not influenced by any disturbances, they are difficult to maintain and offer a very limited life time [21]. Airborne observatories like SOFIA in contrast offer the possibility to change the science instruments and to maintain the telescope easily and therefore serve as a versatile scientific platform. Upgrades can be carried out quickly and much cheaper compared to satellite based telescopes.
The main objectives of SOFIA are [3]:

- star birth and death
- formation of new solar systems
- identification of complex molecules in space
- planets, comets and asteroids in our solar system
- nebulae and dust in galaxies (or, ecosystems of galaxies)
- black holes at the center of galaxies

Therefore, the scientific community can apply for observations with SOFIA which may be granted by a committee in a peer review [1].
Currently, the three American instruments FORCAST, HAWC+, EXES and the two German instruments FIFI-LS and GREAT with different wavelength sensitivity and spectral resolution are available for the observations. Additionally, the Focal Plane Imager Plus (FPI+) shown in Figure 1.4 can be used for further observations like occultations from remote locations, as only brightness measurements are relevant. [16]

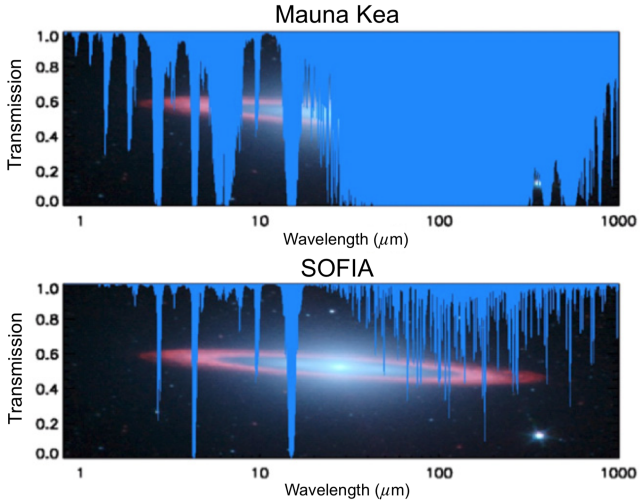


Figure 1.3: Comparison between Airborne and Ground Based Observatories. Some parts of the Infrared spectrum are invisible on the ground, because of the atmospheric water vapor absorption. [3]

1.2 Telescope Assembly

The Telescope Assembly has been developed by a consortium led by the companies MAN and Kayser Threde. It is divided into the telescope cavity and the cabin side, separated by a pressure bulkhead.
On the cabin side of the bulkhead, the Vibration Isolation System (VIS) uncouples the Telescope Assembly (TA) from aircraft vibrations and locks it during take off and landing. This side of the Telescope Assembly is built up in different layers, where the outer cradle is formed by the Coarse Drive. It uses four electric motors to elevate the whole Telescope Assembly between about 20°and 60°.
One layer further inside is the Fine Drive which is used for smaller rotations in all three Degrees of Freedom. Permanent magnetic rotors and static electromagnetic coils enable precise movements of the telescope.
The inner cradle of the cabin sided TA is the spheric Hydrostatic Oil Bearing in the center of the bulkhead. It contains an oil film of 50 µm thickness and 50 bar pressure. This allows an easy and smooth rotation of the TA around all three axes and additionally creates a pressure boundary between the cabin side and the cavity side of the telescope. A set of Spherical Sensors provide information about the position of the telescope.
In order to operate the telescope correctly, cabin side and cavity side of the TA need to be in balance. This can be done by adding counterweight plates on the cabin side as shown in Figure 1.4. Additionally, the Balancer Drives can be used for fine-tuning the telescope balance in all three dimensions during the flight, as the Science Instruments (SIs) lose weight due to coolant evaporation.
The movements of the telescope in relation to a predefined inertial coordinate system are measured by three gyroscopes. This displacement can be expressed as a quaternion, which describes the rotation from one coordinate system to another. The inertial coordinate system in turn can be referenced by a constant quaternion to the Equatorial Reference System based on the earth's equatorial plane. This enables a precise orientation of the telescope within inertial space and therefore an exact pointing on the target.
The Secondary Mirror Assembly (SMA) in front of the Primary Mirror (PM) is the most integrated system of the TA and contains the Focus Center Mechanism (FCM) and Tilt Chop Mechanism (TCM). Both mechanisms are indispensable for SOFIA's imaging goals. As the cavity side of the telescope undergoes temperature differences from ground to stratospheric temperatures, thermal expansion and shrinkage of structural elements cause the Secondary Mirror (SM) to move out of focus. Therefore, the FCM is able to move the SM in three translational and two rotational axes.
The TCM allows a fast and precise tilt of the SM in any direction without inducing moments on the TA. This is used for chopping, where the SM is tilted to a specified position in order to observe a different region in the sky. The background of the neighbor region can then be used for subtracting the background radiation from the target.

1.3 Telescope Optics

SOFIA's telescope is a Cassegrain Telescope in a Nasmyth configuration. A Cassegrain Telescope consists of a parabolic Primary Mirror (PM) with a very short focal length. Its focus is elongated by an opposing hyperbolic Secondary Mirror (SM). The classical Cassegrain Telescope has an opening in the PM for the secondary mirror beam. In this case, the Science Instruments would be mounted behind the Primary Mirror.
As there is only limited space within an aircraft, SOFIA's telescope has an dichroic Tertiary Mirror (TM) on top of the PM. It separates the Secondary Mirror beam into the infrared spectrum and the visible spectrum and diverts both by 90° through the horizontal Nasmyth-Tube. The infrared spectrum is observed by the SI, which is attached to the Nasmyth-Tube using the Science Instrument Flange. The visible part of the beam is diverted to the FPI+ to enable star tracking for stabilizing the telescope and additional observations. The Wide Field Imager (WFI) and Fine Field Imager (FFI) for star tracking and telescope guidance are mounted on the telescope structure in the cavity and will be soon upgraded to WFI+ and FFI+.

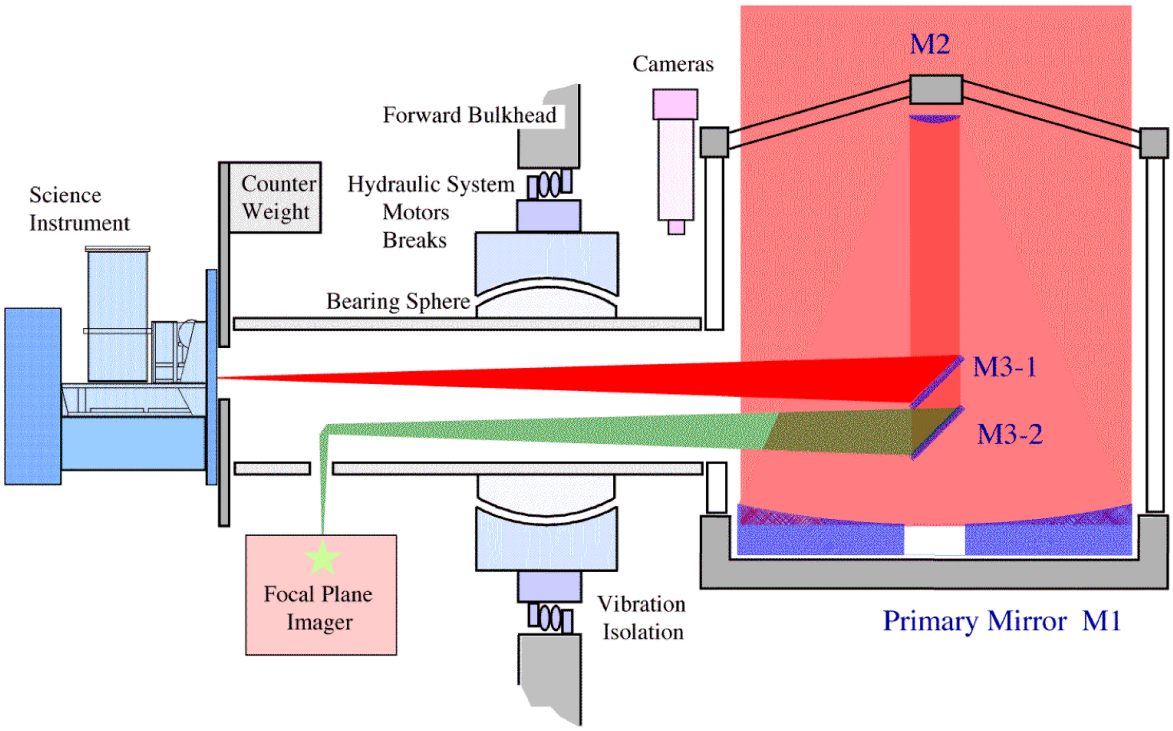


Figure 1.4: SOFIA Cassegrain Optics [14]
The Primary Mirror focuses the image on an opposing Secondary Mirror. The Secondary Mirror beam is then split and diverted by the Tertiary Mirror through the Nasmyth-Tube to the Science Instrument.
Counterweights keep the cabin side and the cavity side of the TA in balance.

1.4 Telescope Software and Electronics

Besides the telescope mechanics, the telescope electronics and software is indispensable for operating SOFIA. It allows controlling the telescope and enables an active image stabilization to improve the image quality.
The telescope control is divided into numerous subsystems with specific tasks, connected via Ethernet or CANaerospace to the MCP. This computer is in charge of synchronizing and controlling the different subsystem as well as monitoring their system states as shown in Figure 1.5.

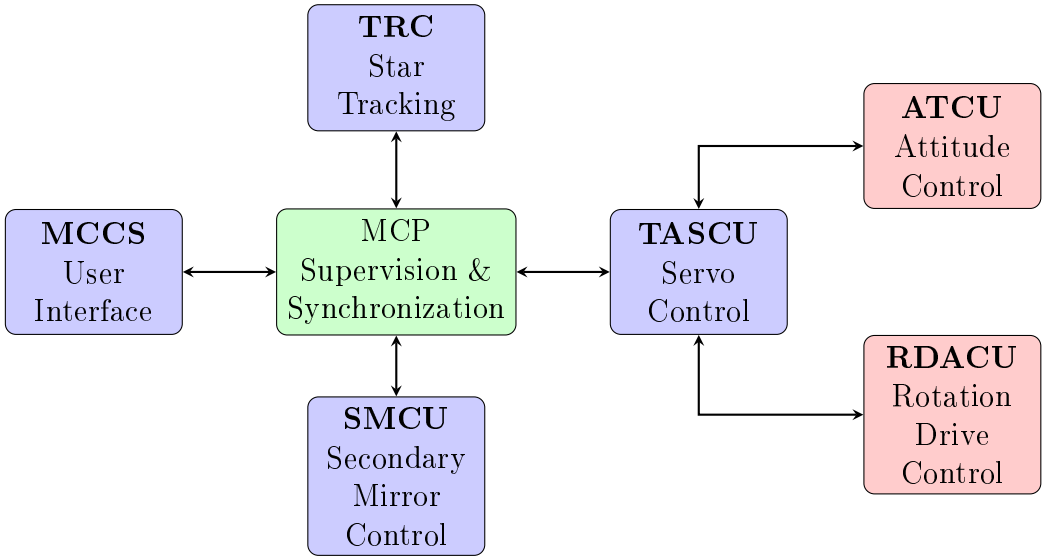


Figure 1.5: Ethernet Network Topology of the TA

The Secondary Mirror Control Unit (SMCU) is responsible for controlling the Secondary Mirror Mechanism (SMM) and therefore for the telescope focus and the chopping, using specially adapted control loops.
Star tracking and telescope guiding is made possible by the Tracker (TRC), as this unit evaluates the images of WFI, FFI and FPI+. Currently, the Tracker is a separate physical unit, but a combined TRCU (TRC and MCP) is under development.
The telescope's elevation can be adjusted using the Coarse Drive and the alignment on the target by the Fine Drive. For adjusting the Center of Gravity, four Balance Drives are available. All of these drives are controlled by the Telescope Assembly Servo Control Unit (TASCU). Therefore, the TASCU communicates with the Attitude Control Unit (ATCU) and the Rotation Drive Assembly Control Unit (RDACU) to get Gyro and Spherical Sensor data as shown in Figure 1.5. Additional data for the TASCU is provided by sensors connected to several Network Extended Control System (NECS) Nodes from Stock Flight Systems. They are connected via CANaerospace buses to the TASCU as shown in Figure 1.6. These measurements are provided by the TASCU for other units in form of Housekeeping Data sent to the MCP.

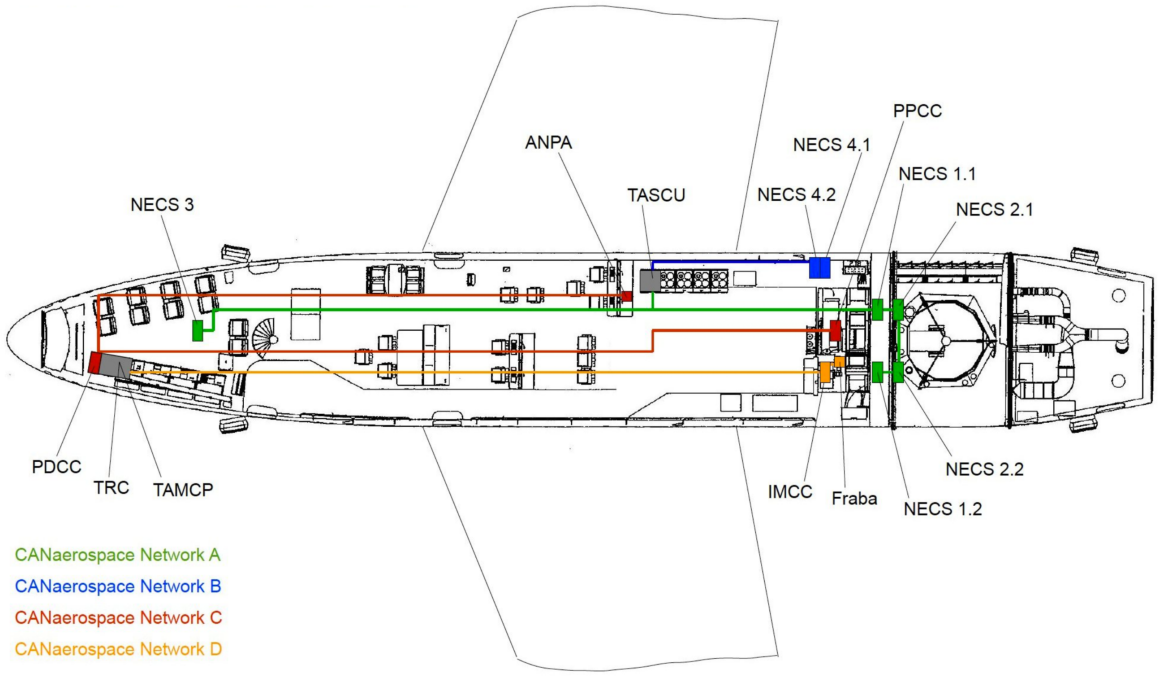


Figure 1.6: CAN Network Topology [18]
The CANaerospace network is divided into four buses A–D. CANaerospace Network A and B connect the different NECS Nodes to the TASCU, which makes use of most of the sensor data. The network C connects the Power Distribution Unit (PDU), the Pressure Window Control Unit (PWCU) & Primary Mirror Electronics and most important for this thesis, the Annunciator Panel, to the MCP. CANaerospace Network D is used for connecting the FPI+ and a position encoder within the Delay Line to the Tracker.

The Mission Controls and Communications System (MCCS) forms the front end of the telescope control, offering a Java based GUI and a Command Interface for the Science Instruments. It enables the two Telescope Operators to command and monitor the telescope with a few mouse clicks. The MCCS itself sends the TO's and SI's commands to the MCP which forwards them to the corresponding TA subsystem.

1.5 Annunciator Panel

In order to operate the telescope in a proper way, the Telescope Operators and the ground personnel need to know the state of the telescope and its subsystems. Per Layout Plan, an Annunciator Panel is situated at the TO console and displays both GO and no-GO states of the telescope systems. The annunciations are divided into alerts (red), warnings (yellow), nominal operation (green) and neutral information (white). Every new alert is first displayed inverted in black letters on a red background to indicate a user acknowledge is necessary. The ANPA is part of the CANaerospace Bus C as shown in Figure 1.7.

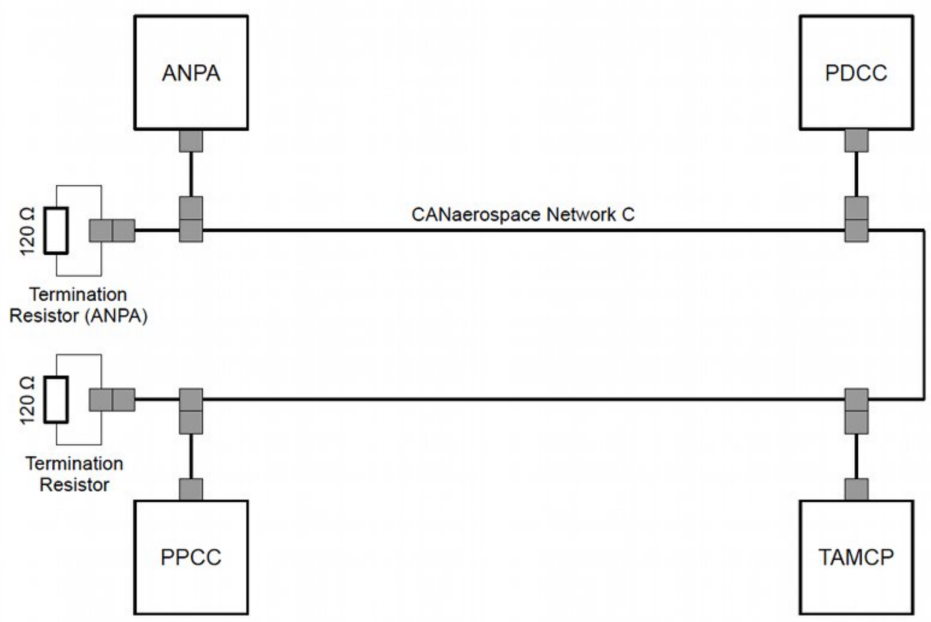


Figure 1.7: CAN Bus C Network Topology with the Annunciator Panel [18].
TAMCP is equal to the MCP referred in this thesis.

Until now, the panel had only a passive role as no direct user interaction with the ANPA was possible. It was updated incrementally with no possibility for the user to refresh the displayed annunciations. For acknowledging new annunciations, a Software Annunciator Panel is offered by the MCCS. The ANPA offered no status indication to make the user aware of any ANPA internal error. These issues complicated the TO's and the ground personnel's work unnecessarily and impacted their situational awareness.
By enhancing the ANPA's capabilities with interactivity and error detection mechanisms, the users can stay focused on their actual tasks.

As the MCCS is managed and operated by NASA, a separate tool for displaying important telescope parameters in the absence of the MCCS was needed. Even though the Diagnostics and Control Computer (DCC) offers an interface to some Housekeeping Parameters, some information, like the Power Supply voltages and currents, is not available. Adding these Housekeeping Parameters into the ANPA enables the DSI engineers and the ground personnel a faster and more efficient way to work.
This said, a new Concept of Operations [6] was derived (as required by the DSI Top Level Design Process in section 3.1). The detailed requirements for the new software are listed in the Software Requirements Specifications [9]. The Concept of Operation for the new software features the following functionalities:

- **More annunciation colors** allow a more precise distinction between the different annunciation types.
- **Interactivity** allows the user to acknowledge new alerts or commanding a refresh of the displayed annunciations directly on the ANPA
- **Housekeeping Pages** display selected Housekeeping Parameters provided by the MCP. This is especially useful for the ground personnel.
- **Error Handling** ensures the nominal operation of the Annunciator Panel and makes it robust against external and internal sources of errors.
- **Built-In Tests** track the state of the system and its surrounding systems and make the user aware of any off-nominal states.

In the following Chapter 2, the detailed hardware and software layout of the Annunciator Panel will be explained in detail. Chapter 3 describes the preliminary work in advance of the development of new functionalities. As there was no sufficient documentation of neither the hardware nor the software, the given explanations are the result of a reverse engineering process of the ANPA. Chapter 4 explains the design and implementation of the functionalities above in the new software. This follows the presentation of the findings during the software development in Section 4.10. In particular, design related performance issues will be analyzed and discussed. The thesis closes with a brief summary and an outlook for future updates and upgrades in Chapter 5.

Figure 1.8: ANPA-2 Unit with the new Software

2 System Analysis

Before a new improved design for the Annunciator Panel can be developed, the current hardware and software needs to be analyzed in detail. Due to the lack of a sufficient system description and sparsely documented source code, the system needed to be reverse engineered. Building on the deep knowledge of the existing functions and the hardware architecture, new functionalities could be developed. A more detailed hardware description can be found in the newly created System Description Document [8]. As the current hardware design led to performance issues as described in section 4.10.1, it is explained in detail within this chapter.

2.1 Hardware

The first generation Annunciator Panel from Kayser-Threde was carried out as dedicated hardware for SOFIA. It consisted of a FPGA and 40 x 4 predefined color patterns for displaying the annunciations, carried out as 5.75" DZUS rack housing. The current hardware is the second generation Annunciator Panel which was introduced in 2011. It was originally designed by Stock Flight Systems as Rotax 912iS Engine Management Unit and contains only a modified software for the usage as a drop-in replacement, 100% compatible to the original Annunciator Panel. A Proof of Concept for new software functionalities and interactivity was carried out by an intern (M. Heck) in 2014/2015. The hardware consists of three basic components:

- TFT-LCD Display Unit
- GPU: PicoMOD DCU-Lx (F & S)
- FPGA: Spartan-3 (Xilinx)

Besides that, there are different GPIO-Ports, two CAN and a serial interface, four buttons, a rotary encoder, a bi-color LED and a SD-Card interface available.[1]

2.1.1 Display and GPU

The display unit is an 640 x 480 px TFT-LCD display. It is controlled by a PicoMOD DCU-Lx Graphics Processing Unit (GPU) which offers two basic operation modes: compiler mode and terminal mode [2]. The PicoMOD processor is delivered with a compiled and flashed software developed by Stock Flight Systems, containing only the company's logo for the startup screen.

[1] The GPIO-Ports, CAN bus 2 and the serial interface are currently unused.

2.1.2 Spartan-3 FPGA

Physically, the Spartan-3 family architecture consists of five fundamental programmable elements, as shown in Figure 2.1 [24]:

- **Configurable Logic Blocks (CLBs)** contain RAM-based Look-Up Tables to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be configured to perform a wide variety of logical functions as well as to store data.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM)** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

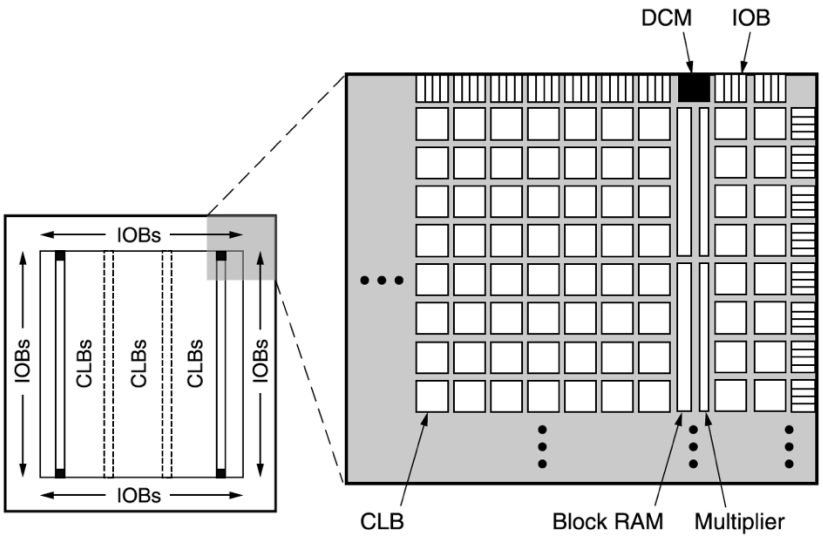


Figure 2.1: Physical FPGA Layout [24]
The Spartan-3 FPGA consists of five fundamental elements: Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), Block RAM, Multiplier blocks and Digital Clock Manager blocks (DCM).

As the CLBs and IOBs are only RAM based, the FPGA's elements need to be reconfigured after each power cycle. An external Flash Memory stores therefore the FPGA's configuration permanently in form of a bit stream which is loaded directly after powering up the system. By flashing a new bit stream, the FPGA layout can be changed.

The current configuration divides the FPGA into several subunits with additional peripheral cores, as shown in Figure 2.2.

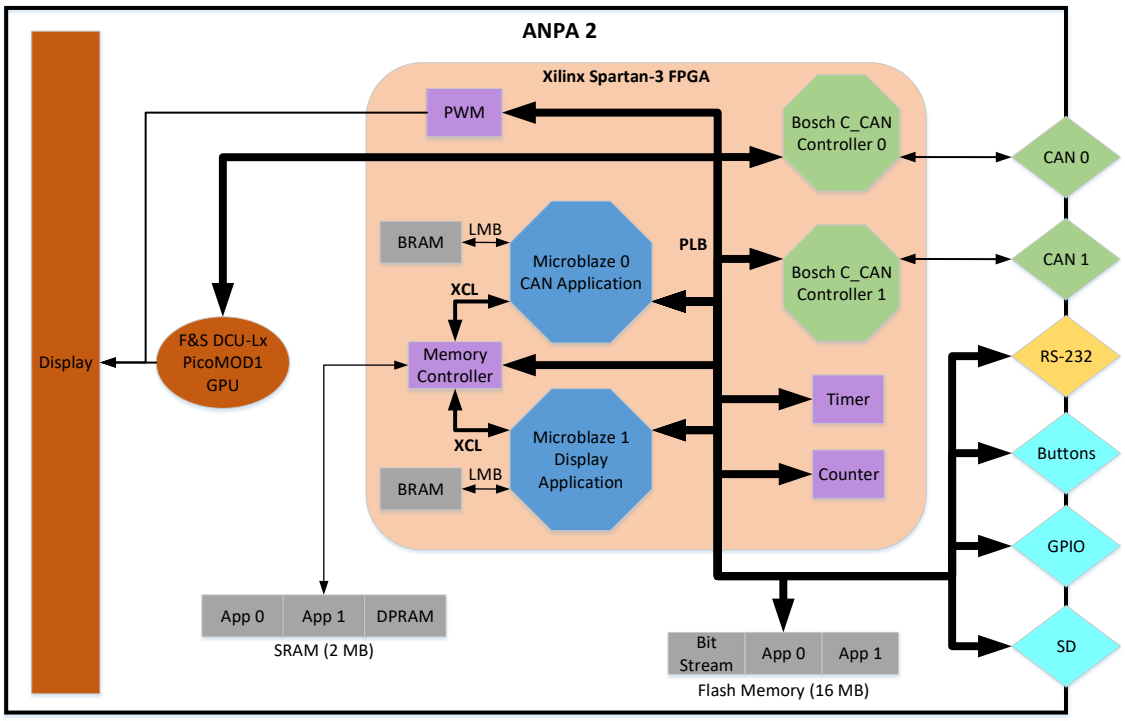


Figure 2.2: Simplified ANPA2 FPGA Layout as it is used.
All components are connected by Processor Local Bus (PLB) (bold lines), Local Memory Bus (LMB) (only Block Random Access Memory (BRAM)), Xilinx Cache Link (XCL) (only Static Random Access Memory (SRAM)) or direct wiring.
Additional converters from the PLB to the serial interfaces, buttons and GPIO are not displayed.

MicroBlaze Microcontrollers

Two programmable MicroBlaze Microcontrollers (0 and 1) are defined in the FPGA layout. Several blocks of BRAM[2] are connected to them via LMB. During the initialization process of the FPGA, both memories are loaded from the Flash Memory with distinct bootloaders, which are part of the FPGA's bit stream. After the completion of the initialization process, both MicroBlazes set their program counter to the address `0x50` and start executing their bootloader as explained in section 2.2.1. The actual application of each Microcontroller is stored in separate partition of the Flash Memory.

[2] 8 KB to MicroBlaze 0 and 32 KB to MicroBlaze 1

An external SRAM contains the application code during runtime and a special section called Dual-Port Random Access Memory (DPRAM) for data exchange between the Microcontrollers as shown in Figure 2.2.
In general, MicroBlaze 0 is responsible for controlling the CAN communication with the CAN participants. It controls the two CAN controllers and copies new messages to a defined section of the shared memory (`DPRAM`).
MicroBlaze 1 on the other hand is responsible for processing the new CAN messages in the DPRAM and reacting to user inputs. Additionally, it commands the display processor PicoMOD through a serial connection.

Bosch C_CAN Controllers

Two Bosch C_CAN Controller IP Cores are implemented in the FPGA design. Each controller is responsible for transmitting and receiving messages on this bus and connected to a CAN bus as shown in Figure 2.2. Both of them hold several registers for configuration and control. The layout of these registers can be found in the Bosch C_CAN Controller User's Manual [17] and will be used later in the software for configuring the controllers and reading/transmitting CAN messages. In the SOFIA design, only bus 0 is used for communication with the MCP.

Peripheral IP Cores

Several peripheral IP Cores are included in the FPGA design in addition to the two MicroBlaze CPUs and Bosch C_CAN Controllers. A Pulse Width Modulation (PWM) Controller is used for modulating a square wave signal to dim the displays background illumination.
For the timing of the custom applications executed on the MicroBlazes, a timer is configured to generate interrupts every millisecond. These interrupts trigger the execution of an Interrupt Service Routine (ISR) as explained in section 2.2.
An additional counter IP Core is used to count the ticks of the rotary encoder on the front panel.
All peripheral IP Cores can be accessed by both MicroBlaze Microcontrollers.

2.1.3 Periphery

The state of the front panel buttons is available as single byte register value. Each bit in this variable represents the state of a button.
The Annunciator Panel also offers a serial RS-232 interface to send and receive messages for debugging purposes. A connection can be established using the following settings: 115.200 baud, 8 data bits per character, 1 stop bit and no parity bit.
A SD-Card reader enables reading and writing data to an external medium. It is used for updating the applications of the two Microcontrollers as explained in section 2.2.1.

2.2 Software

The two MicroBlaze Microcontrollers and the PicoMOD GPU make use of a distinct software. As the software of the PicoMOD is only used for the start up image and can't be changed, it will be neglected in this chapter.
Both of the MicroBlazes provide their own bootloader that manages the startup and provides basic functionalities. They load and execute a distinct application from the flash memory which provide additional functionalities like CAN communication or display driving. These applications are the target for changes of the ANPA. They share the `DPRAM` for exchanging CAN messages. The layout of this shared memory is shown in Figure 2.3.

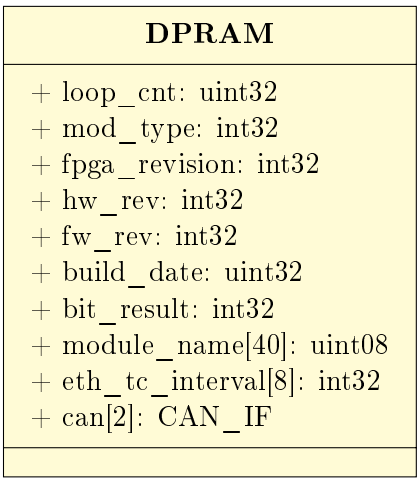


Figure 2.3: DPRAM Structure

It contains several variables to indicate version information. Furthermore two structures `CAN_IF` buffer the received and the pending outgoing CAN messages for the two channels. To avoid collisions, no variable and buffer will be written by both applications.
The two applications currently also use a common software layout: During the startup they initialize the hardware, interrupts and their variables. This includes setting up the aforementioned timer to generate interrupts every millisecond. The corresponding ISR increments a global timing variable `low_time` . After the initialization process, an infinite main loop is executed. Every cycle of this loop must take less than one millisecond. At the end of each cycle, the execution is paused by a spin lock[3], waiting for the timer interrupt to increment the timer variable `low_time` and thereby triggering the next cycle. Within this loop the actual tasks of the application are being processed. Some of these tasks may be executed in every loop cycle, whereas some will be executed with a defined period. The arrangement and execution time of these tasks will be discussed later in chapter 4.9.

[3]The thread waits within a loop ("spin") for the release of the lock.

2.2.1 MicroBlaze bootloaders

As explained in chapter 2.1.2, each MicroBlaze Microcontroller comes with its own bootloader stored in the BRAM. These bootloaders are executed after the FPGA's initialization process finishes. In the beginning, the bootloader of MicroBlaze 0 searches for updates on the SD-Card. These updates must be stored as S-Record files and named `mb0.srd` for MicroBlaze 0 and `mb1.srd` for MicroBlaze 1. If these files are found, the new software versions are flashed to specific addresses in the external Flash Memory. The bootloader running on MicroBlaze 1 waits during the update process for MicroBlaze 0 to finish.
After this, both bootloaders start mapping the contents of the S-Record files in the Flash Memory to their memory locations, as specified by the Linker. In case of the current software layout, the targeted memory is the external SRAM module. However, it is possible to map parts of the application's code to different memory types and addresses like the BRAM. This will become relevant in Section 4.10.1.
As soon as the mapping of the applications is finished, both MicroBlaze CPUs set their program counter to the start address of the corresponding application and start with their execution. During normal operation, the execution won't return to the bootloaders.

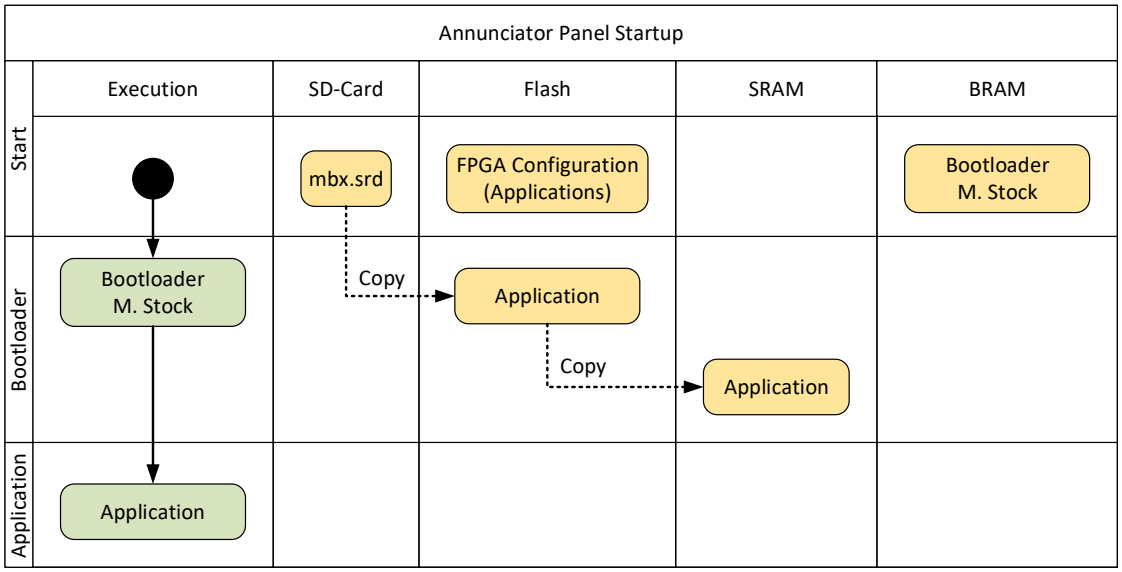


Figure 2.4: Bootloader Activity Diagram
After the FPGA is initialized, the bootloaders from Stock Flight Systems are being executed. If any software updates are found on the SD-Card, they are copied to the corresponding address in the Flash Memory. Afterwards, the code is mapped to the SRAM, to be then executed.

2.2.2 CAN Application

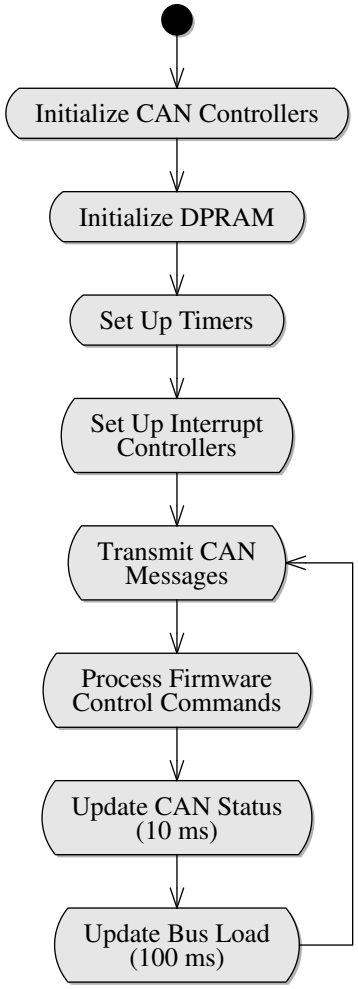


Figure 2.5: CAN Application Activity Diagram

The CAN Application executed on MicroBlaze 0 is responsible for processing CAN messages. During the startup of the software, both CAN controllers and the `DPRAM` are set up and initialized. Additionally to the loop timer, two interrupts are set up to be triggered at the reception of new CAN messages by one of the two Bosch C_CAN Controllers. The application then executes the following tasks within the main loop: At first, and during every cycle, the CAN transmission buffer in the shared `DPRAM` is checked for new messages. If pending messages are detected, they are being copied to the transmission buffer of the corresponding Bosch C_CAN Controller. This follows the processing of firmware control commands which can be sent from MicroBlaze 1 to MicroBlaze 0 using a dedicated section within the `DPRAM` . During every 10th execution of the infinite loop, the status of the Bosch C_CAN Controllers is written to the `DPRAM` for further processing. Afterwards, the bus usage counters in the shared memory are updated every 100 ms.
An Interrupt Service Routine will be executed in case of a newly arrived message on either one of the Bosch C_CAN Controllers. This routine checks the new message for correctness and then copies it to the structure variable `DPRAM` . They will then be processed by the Display application on MicroBlaze 1. In case of any bus disturbances, the controller detaches from the bus and switches to the "Bus Off" mode as explained in Section 4.10.3. This triggers an interrupt on MicroBlaze 0 which starts the reinitialization process of the CAN controller.

Period / ms	Task
1	Transmit pending CAN messages
1	Process firmware control commands
10	Update the CAN status and increment the firmware loop counter
100	Update the tx/rx message and bit counters
60 000	Update the board temperature

Table 2.1: Timing of the CAN Application Tasks

2.2.3 Display Application

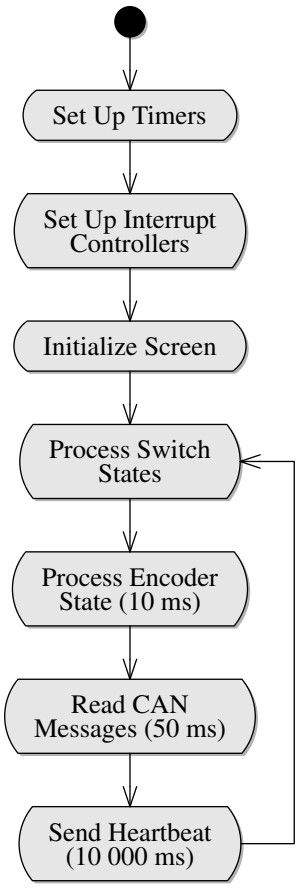


Figure 2.6: Display Application Activity Diagram

The Display application is executed on MicroBlaze 1 and responsible for servicing the display and reacting to user commands. During the boot process, the hardware is initialized. This includes setting up the timer for the interrupt, setting up the peripheral inputs like buttons, rotary encoder and SD-Card as well as initializing the PWM for controlling the display's brightness. After waiting 10 seconds for the PicoMOD GPU to exit its boot sequence, the DCU-Lx is configured. At first, the Stock Flight Systems company logo is displayed and the DCU-Lx switched to terminal mode. This follows setting up several parameters like font, cursor positioning mode and line wrap. During the first 4000 cycles of the infinite loop, the four pages of the DCU-Lx are being initialized. The annunciation grid is drawn to page 0 and 1 for later switching between the pages. A lamp test pattern is drawn to page 2 in order to test the screens capabilities by pressing the encoder button.
Afterwards the application serves repeatedly tasks withing the infinite main loop. The states of the softkeys are evaluated in every execution cycle of the loop. As there is currently no action assigned to any button, nothing will be executed. The state of the rotary encoder counter is checked every 10 ms. If the state has changed, the display brightness will be adjusted by the PWM controller. The incoming CAN message buffer in the `DPRAM` serviced by MicroBlaze 0 is checked every 50 ms. In case of new messages, the CAN identifier, message code and service code are being evaluated and the proper action performed. In order to enable the MCP to check the Annunciator Panel's operation status, a heartbeat message is sent to the MCP every 10000 ms.

Period / ms	Task
1	Process switch states
10	Process encoder state
50	Read CANaerospace messages received from CAN channel 0
10 000	Send a heartbeat message

Table 2.2: Timing of the Display Application Tasks

3 Preparatory Work

Given the state of the software and the documentation, developing and compiling the new software for the Annunciator Panel requires some preparatory work. This includes creating a build environment containing the source code files, installing the compiler and linker executables and setting up a makefile. The development of the software will be guided by a development process.

3.1 Software Design Process

The DSI Top Level Design Process [25] was derived from the NASA Systems Engineering Handbook and divides the process of software development into 4 phases.
The first phase is triggered by a request for a new feature. This request needs to be analyzed and broken down into top level requirements to the hardware and software. These requirements are used to estimate a budget, personnel and resources estimation. This knowing, a project plan defining and assigning the milestones of the project needs to be developed. A Concept of Operations (ConOps) describes the different use cases which the new software needs to cover. Additionally, the interfaces of new software are designed and procedures for Verification & Validation (V&V) of the systems prepared. The first phase ends by a System Requirements Review.
This follows the second phase of the development process. It mainly consists of the preliminary design process, which may lead to the design of different solutions with a subsequent selection. The testing procedures for the V&V process are devised and the schedule updated. A Preliminary Design Review checks the feasibility of the proposed system design.
During the third phase of the software development, the preliminary design and the interfaces are finalized and reviewed in the Critical Design Review. After the approval, the new features are implemented in the software and tested. The developed code must be documented in a way enabling an easy understanding of its purposes.
The last phase of the software development consists of testing the new software. During the process of Verification & Validation, the software is checked for meeting all of the defined requirements and if the functionalities work properly. A user manual provides information on how to use the new system. A lessons learned document is used to record any improvements for future projects.
Normally these four phases are followed by the operation and possible system upgrades. At the end of the system's life cycle, it needs to be decommissioned.

3.2 Source Code Repository

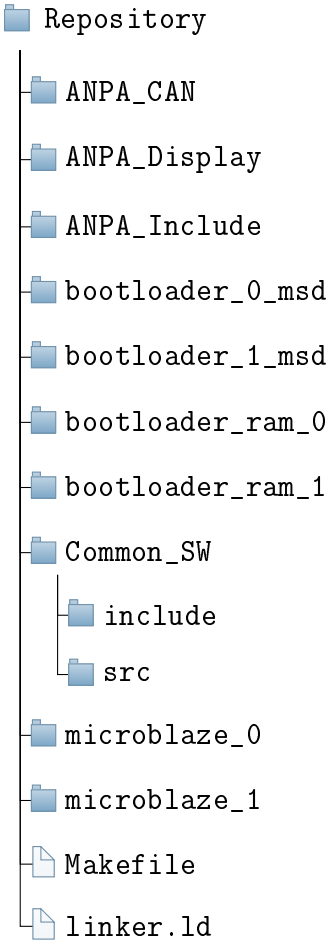


Figure 3.1: Folder Structure

In order to enhance the traceability of the applied software changes, the folder structure containing the C source files and libraries used for the software was checked in a Concurrent Version System (CVS) repository on the DSI server. It is organized as Figure 3.1 shows.
The first two folders `ANPA_CAN` and `ANPA_Display` contain the source files for the CAN- and Display-application. `ANPA_Include` contains the application's header files. These source and header files are target of the desired changes of the current software. The following `bootloader_x_msd` folders contain the original bootloader code of Stock Flight Systems, whereas the folder `bootloader_ram_x` contain the new SD-Card bootloader code as described in section 3.5. `Common_SW` includes libraries used by the bootloaders and the applications. Some MicroBlaze-specific library functions are contained in the `microblaze_x` folders. The last items `Makefile` and `linker.ld` in the build environment define the rules for the build process.
Newer software builds will only differ in the files contained by the folders `ANPA_CAN` , `ANPA_Display` and `ANPA_Include` .
The initial software version 1.6 was committed with the tag `F20130429_001` , whereas `F20180409_001` was used for tagging the new and most recent version 2.0.

3.3 MicroBlaze Toolchain

The MicroBlaze toolchain developed by Xilinx provides the needed tools for the build process. They are derived from the GCC and GNU Binutils libraries and modified for the MicroBlaze architecture. As the toolchain is only compatible to Windows and Linux, a solution for Mac using Wine, a Mac compatible runtime environment for Windows applications, was found. The final build was compiled using the DSI Linux server with the following software versions:

MicroBlaze toolchain:	12.3 (built 2010-07-14)
GCC:	4.1.2 (built 2007-02-14)
Binutils:	2.16 (built 2005)

3.4 Makefile & Linker Script

A single Makefile defines the rules for the build process. Setting up the Makefile properly is mandatory for building an executable application and prevents mistakes due to architectural problems or missing libraries.
It is perfectly tailored for the mentioned repository structure, referencing all listed folders. New source or header files placed in the given directories will be detected and included to the build process.
In order to execute parts of the generated binary objects from different memory types and regions, a linker script was set up. Therefore, the default linker script from Xilinx was extended to be able to link the code to the BRAM section as needed in chapter 4.10.1.
The Makefile offers the following rules for building the project:

- all: Build all executables
- clean: Delete all executables in the build environment

For building the applications, the command `make clean all` is recommended.

3.5 SD-Card Bootloader

Developing a new software comes along with a debugging process of the new code with several iterations. During the debugging process of the ANPA, the new software needs to be flashed to the Flash Memory to be then tested, which leads to a high number of Flash Memory writes. Because of the limited number of possible writes to a Flash Memory[1], this proceeding causes a shortened lifespan of the Flash Memory and the ANPA.
In order to avoid this, two new second level bootloaders were developed. Their purpose is to load the applications directly from the SD-Card into the SRAM instead of writing them to the Flash Memory in advance. These second level bootloaders are flashed once to the flash memory instead of the actual applications. During the startup process, the initial bootloaders of Stock Flight Systems load and execute the second level bootloaders. These in turn search the SD-Card for application files named `mb0_ram.srd` for MicroBlaze 0 and `mb1_ram.srd` for MicroBlaze 1. If these files are found, they are being mapped to the targeted memory and executed afterwards.

[1] about 100 000 according to [23]

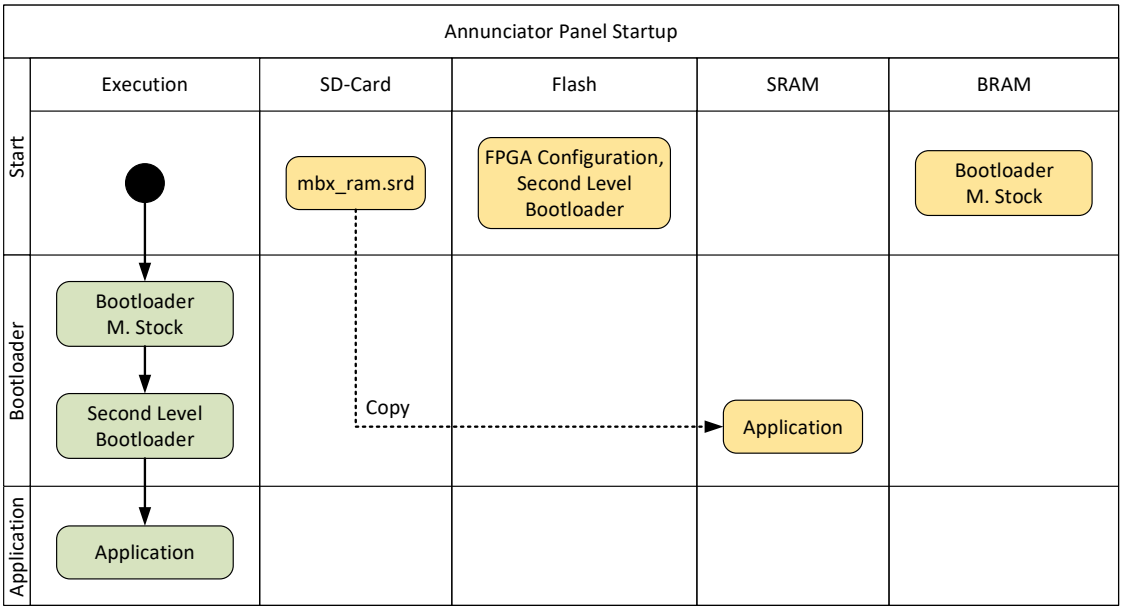


Figure 3.2: Annunciator Panel Startup during Development
After the FPGA is initialized, the bootloaders from Stock Flight Systems are being executed. These in turn execute the new second level bootloaders, which map the application from the SD-Card to the targeted memory, to be then executed.

This procedure not only safes lifetime of the Annunciator Panel but also speeds up the debugging process as flashing is only required once. It must be noticed, that this process makes the usage of an SD-Card indispensable and therefore is not recommended for operation. However, these new second level bootloaders will be only used for debugging purposes and not in the flight software.

4 Implementation & Findings

The new Annunciator Panel software was developed, based on the latest available software version 1.6, which was released 2013 by Stock Flight Systems. In the beginning, this code was analyzed in Section 2.2 to be then optimized and cleaned up in order to have a solid base to start with. Afterwards the requirements were implemented as defined in the Software Requirements Document [9]. The implementation of the most important requirements will be explained in this Section. A more detailed description of the software layout and the implementation of the functionalities can be found in the System Description Document [8].

Figure 4.1: Revised Annunciation Page Layout
The title section of the screen defines the annunciation columns with the annunciations listed in the center of the screen. The bottom of the screen is used to label the softkeys and to display the UTC time.

4.1 State Model of the new Software

The new software now features four different pages, as shown in Figure 4.2. The Annunciation Page is defined as the Home Page and will be displayed after booting the system. From there, the Housekeeping Pages and the Built-In Test (BIT) can be reached. As demanded by Requirement #6.1, a way back to the Annunciation Pages is provided on every page. Additionally, the system will automatically return to the Annunciation Page after 20 s of user inactivity (Requirement #2.1b).

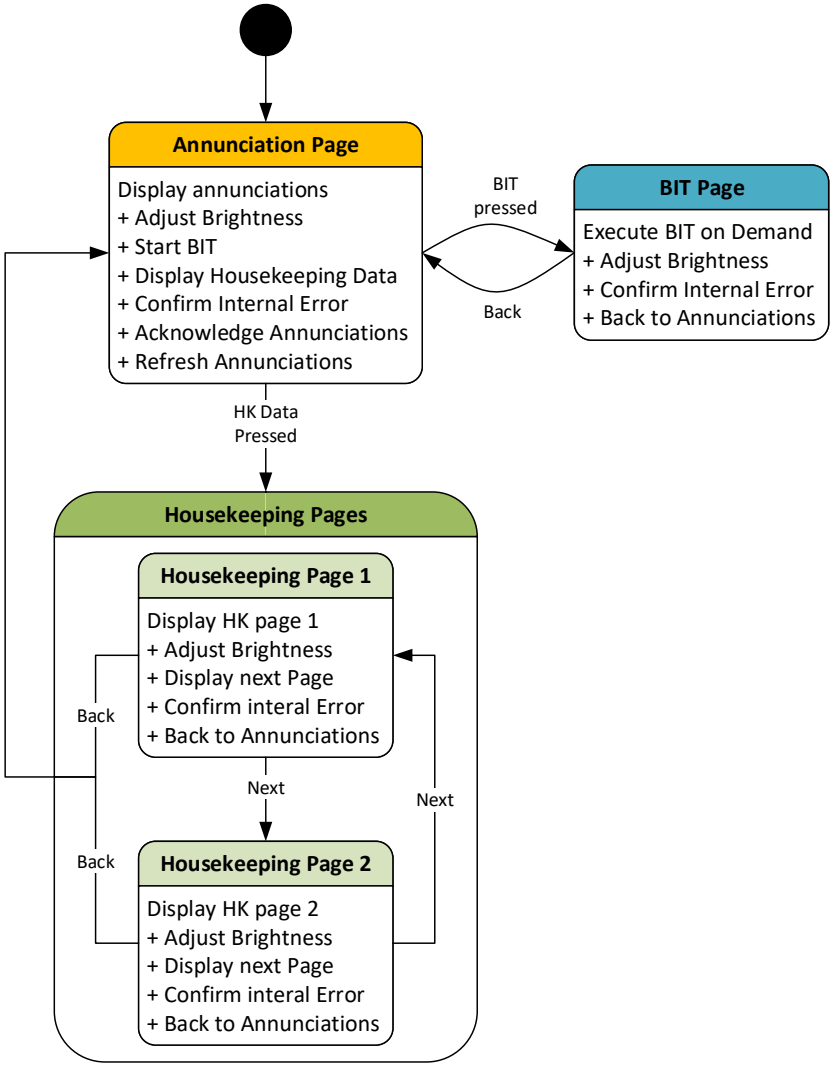


Figure 4.2: State Diagram of the new Software
All pages are reachable from the central Annunciation Page.

4.2 Annunciation Colors

In order to allow a more precise distinction between the different annunciations, more colors were introduced. So far, white was used for neutral annunciations, green for nominal operation, yellow for warnings and red for indicating alerts. According to Requirement #2.4a [9] cyan, magenta, orange and blue were added to the list of available colors. Therefore, one of the reserved bits in the service code of the annunciation message was used. Currently, only cyan is used for indicating advisories to the user, but more colors may be used in a future update of the MCP software. Figure 4.1 shows examples for annunciations in all currently used colors.

4.3 Acknowledge & Refresh

Up to now, new alerts could only be acknowledged using the Software Annunciator Panel on the MCCS. In case of outdated annunciations, the only possibility to indirectly request a refresh from the MCP was to command a lamp test on the MCCS. This means that the ANPA was not able to provide its annunciation function independently from the MCCS, as Requirement #2.7 and #2.8 [9] are demanding.
Therefore, two new CANaerospace messages were established to send these two requests from the ANPA to the MCP. By pushing the right softkey shortly, an acknowledge request is sent to the MCP. This in turn acknowledges the unacknowledged alerts and updates the displayed annunciations. A long push of the right softkey triggers the transmission of a refresh request message by the ANPA. The MCP reacts to this message by refreshing all 40 indicator fields on the Annunciator Panel.

4.4 UTC Clock

Even though the ANPA doesn't have an internal Real Time Clock, the Annunciator Panel offers the possibility for time measurement during runtime, as the execution of a main loop of the Display application is triggered by a timer controlled ISR every millisecond. The start value of the internal clock is provided by the MCP, which uses the IRIG-B time signal from the aircraft's GPS sensor. As explained in section 4.7, the ANPA now may synchronize its internal clock with the MCP every minute during the continuous Built-In Test. The UTC time is displayed above the SD-Card slot as shown in Figure 4.3 and requested by Requirement #2.5d [9] indicating the time and the nominal operation of the Annunciator Panel. This allows every user to determine the ANPA's proper operation with one quick look.

4.5 Housekeeping Page

On the aircraft, the DSI is responsible for the Telescope Assembly and NASA for the aircraft and the MCCS. During the flight preparation and maintenance work by the DSI team, the MCCS may not be available for displaying Housekeeping Data. The DCC offers a limited interface to these Housekeeping Parameters, but some values may not be available within this system. In order to allow an easy access to the most important values, two Housekeeping Data Pages were integrated in the new ANPA software as requested by the Requirements #3.1 - #3.14 [9]. They can be displayed and switched by clicking the left softkey.
Therefore, several new CANaerospace messages were introduced to enable the ANPA to request these parameters from the MCP. Whenever the MCP receives these request messages, it responds by sending the requested values back to the Annunciator Panel. The ANPA on the other side eventually needs to cast these values to the appropriate type and store them for the later display process.
Due to current performance restrictions as described in section 4.10.1, the values are only updated once per second (see Requirement #3.3).

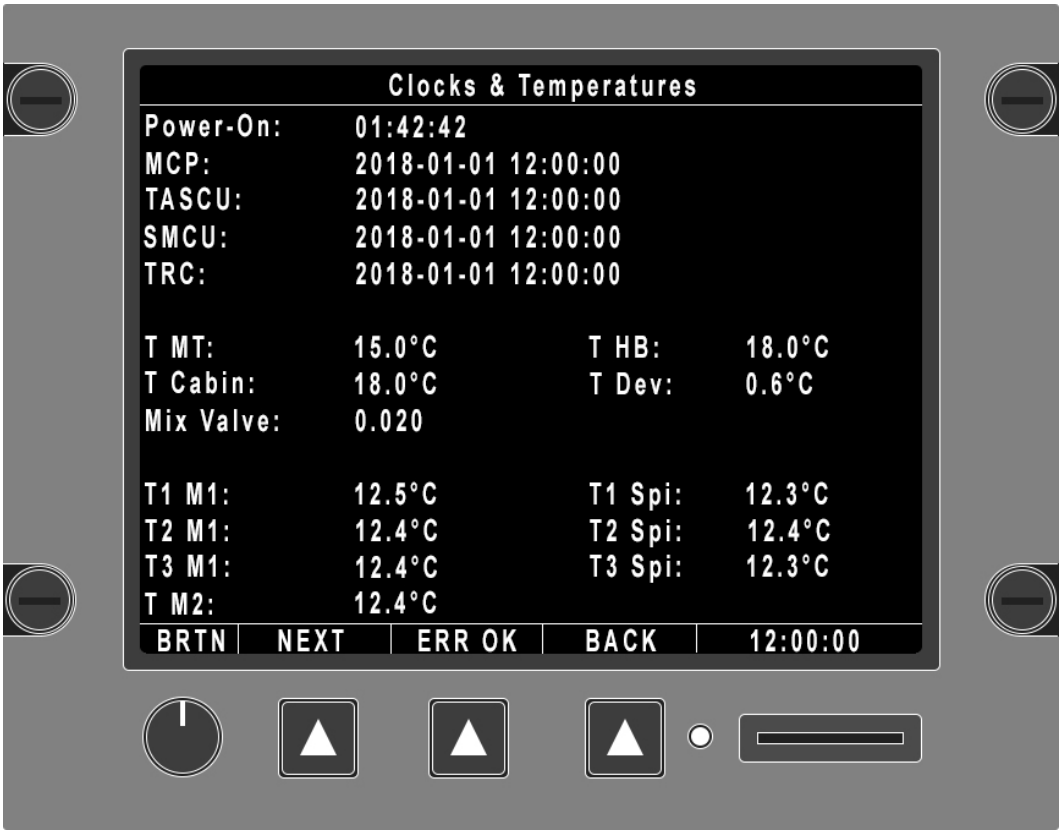


Figure 4.3: The Housekeeping Page 1 displays the timestamps of several TA subsystems and a selection of temperatures of the hydraulic system and the Cavity.

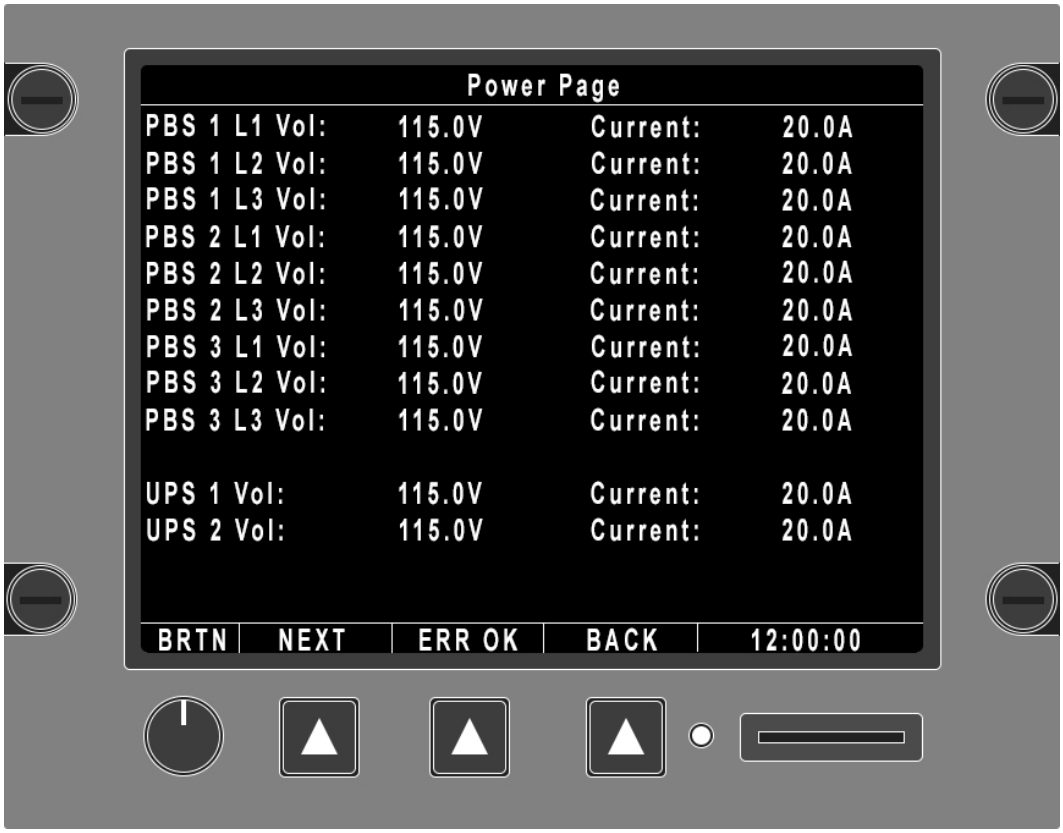


Figure 4.4: The Housekeeping Page 2 displays the voltages and currents of the Power Buses and the Uninterruptable Power Supply

4.6 Error Handling

Up to now, the Annunciator Panel offered no internal error detection and failed silently. Due to the divided system design, a system crash didn't lead to a black screen, but to freeze the current annunciations. Because there was no internal error detection, the user was not able to tell if the shown annunciations were still being updated and therefore relied on the outdated annunciations. This may lead to critical situations were the user was no longer aware of the system alerts.
As a first step, a clock was introduced in section 4.4 to display the UTC time which serves additionally as a heartbeat indication of the ANPA.
As stated in the Requirements #5.3 - #5.5 [9], in a second step, an error handler was created to catch internal errors and inform the user about their occurrence. The handler consists of an error queue, two functions to add and remove errors from the queue, and a routine to display the errors. The queue itself is designed as a linked list, that is ordered by the priority of the error. Each function was redesigned to detect and report their most common errors. If an error is detected, a new node is added to the queue at the position according to the error's priority. A second routine later checks

the error queue for errors and performs countermeasures based on the error code. If the error couldn't be fixed, an error message for the error with the highest priority instead of the clock is displayed above the SD-Card slot. Additionally, the LED blinks red. In case of a non responsive subsystem, the corresponding annunciation column header is backgrounded in red rather than displaying an error message. The full list with all available error messages and their criteria and impacts is available in the ANPA User Manual [11].
A situation with a non responsive TASCU and a CAN bus error is shown in Figure 4.5.

Figure 4.5: Error situation with a CAN bus error and the TASCU not responding.

The user is able to confirm and thereby remove an error message from the queue by pushing the center softkey with the label "ERR OK". Confirming all errors in the queue is possible by a long push of the same softkey. Both functionalities are accessible on all pages.
However, it is only possible to confirm error messages which are displayed above the SD-Card slot. Subsystem connection errors can't be confirmed. They will be removed automatically, after the subsystem becomes responsive again.

4.7 Continuous Built-In Test

In order to ensure a correct operation of the Annunciator Panel and to be able to identify problems, two Built-In Test (BIT) mechanisms were implemented. A continuous BIT is executed every minute and checks several system parameters. An additional Built-In Test on Demand can be commanded by the user and focuses more on testing the graphical capabilities as an extension to the existing lamp test. The continuous BIT performs several internal and external checks as demanded by the Requirements #5.1 and #5.2 in [9]. If any error is detected, the corresponding error, as listed in the User Manual [11], will be thrown.
In detail, the following checks are performed:

- **Board temperature**: The board temperature is checked to be within reasonable range (-5°C – 50°C).
- **Power-on time**: The Annunciator Panel's power-on time needs to be under 49 days, as the timing variable `low_time` from chapter 2.2 will overflow.
- **Connection to MCP, TASCU, SMCU and TRC**: In order to check the connection to these four subsystems, the ANPA requests their timestamps from the MCP five seconds before executing the Built-In Test. If the timestamps are less or equal to the timestamps received during the previous BIT, the connection is marked as faulty.
- **Connection to DCU-Lx**: Five seconds before the Built-In Test is performed, the software version of the DCU-Lx is requested. If the answer doesn't match the default "DCUWINCE/8BPP", a connection error is thrown.
- **SD-Card**[1]: This test checks the SD-Card for availability and for free space. If the card is full, an space error indicates issues with the SD-Card.
- **CAN status**: The status of the Bosch C_CAN Controller 1 is read from the DPRAM and evaluated. The states are defined in the Bosch C_CAN User Manual [17].
- **CAN usage**: The CAN usage is calculated by the Bosch C_CAN Controller and available in the DPRAM. After applying a moving average with a timespan of one minute, the usage of CAN Bus 1 must be under 80%.

[1] only during the start up BIT

4.8 Built-In Test on Demand

Additionally to the continuous BIT, the user can command a BIT on demand which focuses on testing the graphical capabilities as defined in the Requirements #4.1 - #4.7 [9]. The test can be commanded by a long push on the encoder button and starts with displaying a lamp test pattern. This follows displaying a red, green and blue background and dimming the brightness from 0% to 100%. Afterwards, a result page summarizes the test's outcome as shown in Figure 4.6.
The following checks are performed additionally to the graphical test:

- Board temperature
- Power-on time
- Connection to MCP
- SD-Card
- CAN status & usage

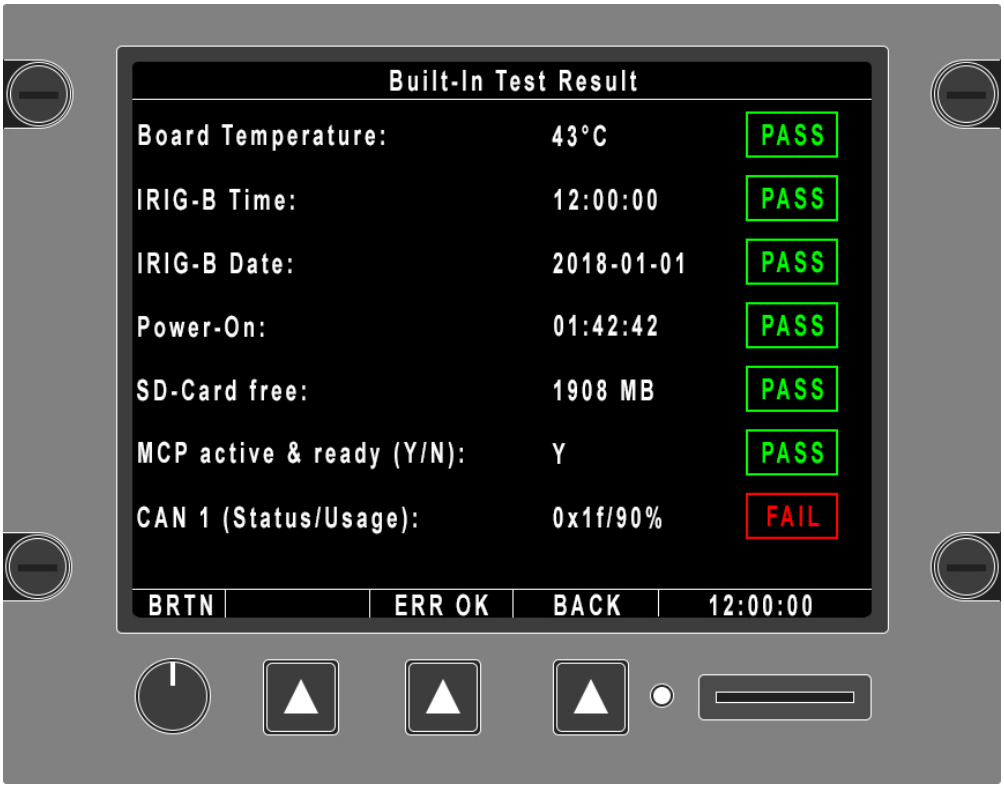


Figure 4.6: Built-In Test on Demand Result Page
All tests were successful except for the CAN bus check, which failed due to a bus overload.

4.9 Task Scheduler

The system architecture of the Annunciator Panel causes two performance issues as described in section 4.10.1. Due to these issues, some commands lead to an extended execution time, longer than the mentioned 1 ms. Displaying a Housekeeping Page for example consumes about 35 ms of CPU time, as several strings have to be concatenated. The combination of the static timing scheme as mentioned in section 2.2.3 and the violation of the 1 ms requirement may cause a non-deterministic behavior of the ANPA. In order to keep the reliability of the Annunciator Panel high, the software now features a task scheduler. Therefore, the infinite loop's tasks were broken down into several smaller tasks as shown in Table 4.1. An Interrupt Service Routine is triggered every millisecond and adds new tasks to a task queue based on the period of each task. The infinite loop was reduced to only check this task queue for new tasks and execute them. This ensures that no task will be omitted and therefore leads to a deterministic behavior of the ANPA. Prioritizing the tasks guarantees the timely execution of the most important task, whereas less significant are postponed instead of dropped completely.
The task queue in detail is an ordered linked list, as shown in Figure 4.7, where each node represents a task. For registering a new task, a new node is added to the list at the corresponding position.

#	Task	Period / ms	Priority
1	Update pages	1	100
2	Switch operation mode	(1)	100
3	Service display buffer	(1)	50
4	Check softkeys and encoder	10	10
5	Check CAN messages	50	10
6	Service error queue, display time	50	50
7	Calculate CAN usage	100	100
8	Control front panel LED	500	200
9	Check page time out	1 000	200
10	Send heartbeat message	10 000	200
11	Perform continuous BIT	60 000	50

Table 4.1: Tasks of the Display Application
Tasks in brackets are only executed if a condition is fulfilled. High priorities a represented by small numbers.

Figure 4.8 shows the Activity Diagram of the new Software with the task scheduler and the system initialization. The process of updating the annunciations, after the user requested a refresh, is shown in Figure 4.9.

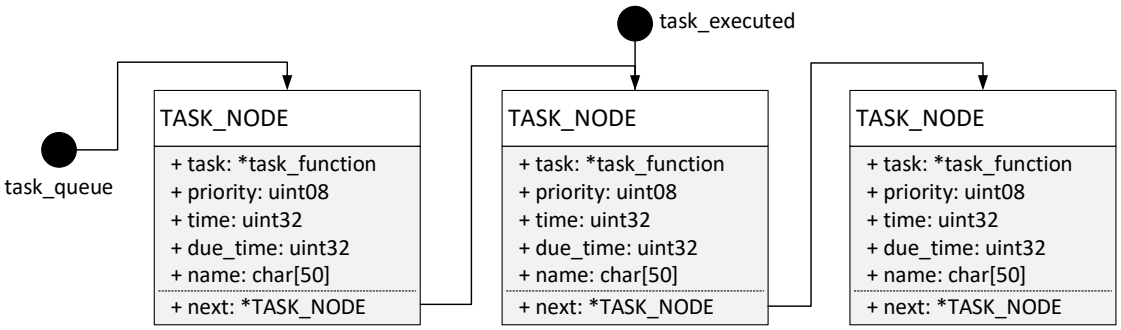


Figure 4.7: Layout of the Task Queue

The Task Queue is designed as a linked list where each node represents a task. The two pointers `task_queue` and `task_executed` are used to indicate the current entry point of the queue and the point of execution.

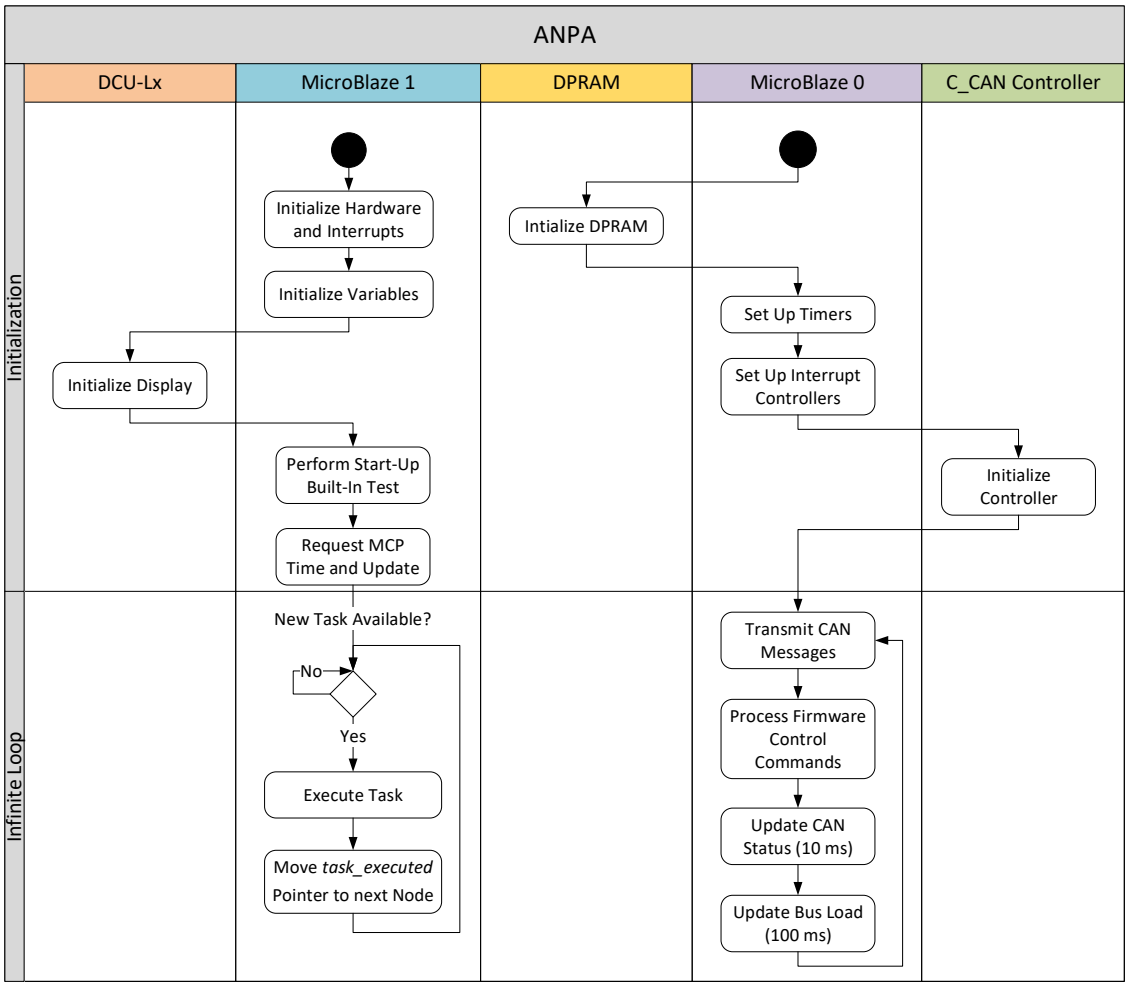
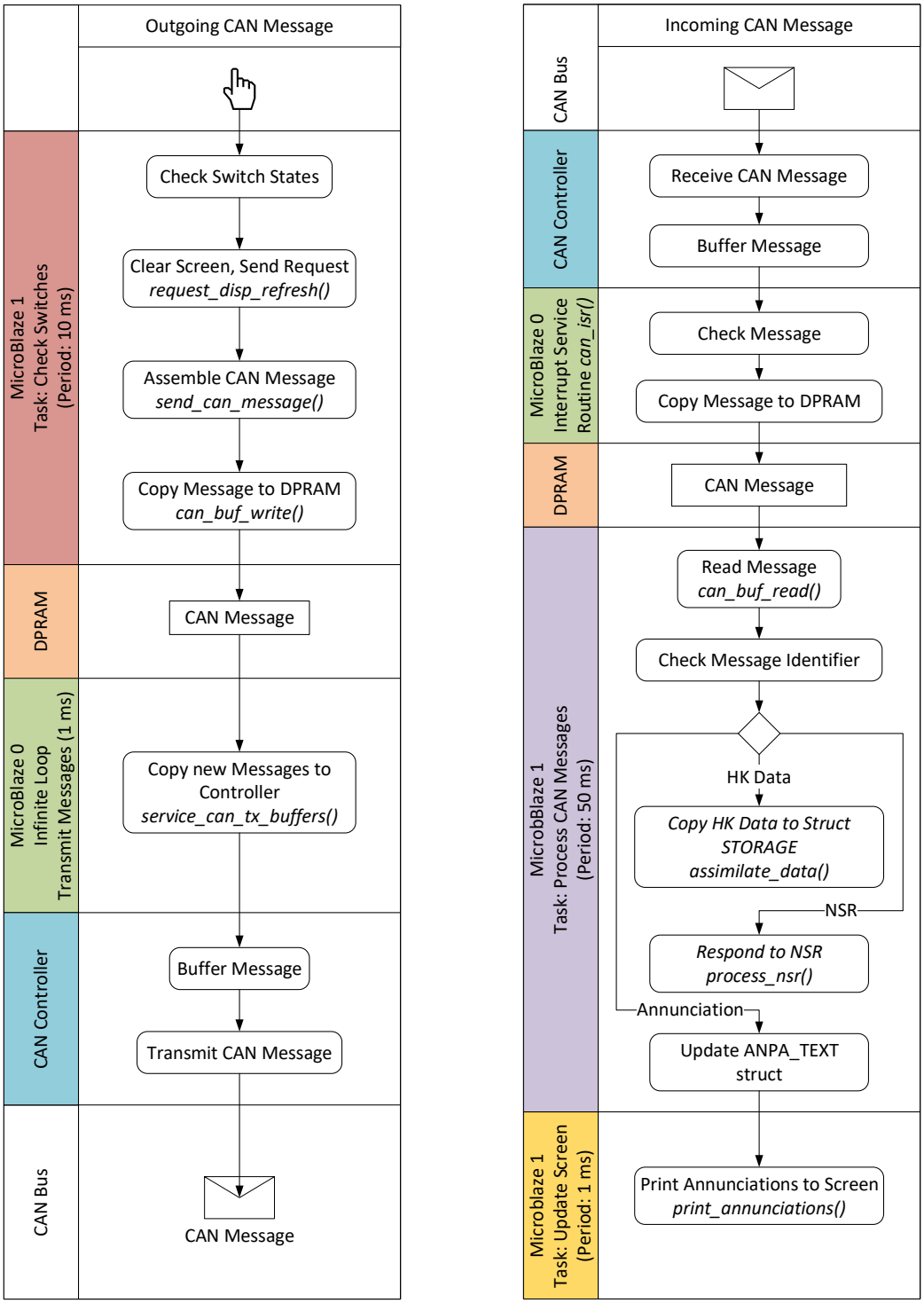


Figure 4.8: Activity Diagram of the new Software 2.0



(a) Refresh Request (b) Incoming Annunciation

Figure 4.9: CAN Message Flow
The internal processes of the MCP are neglected here.

4.10 Findings

4.10.1 System Performance

During the development of the new Annunciator Panel software, memory performance issues were discovered. Due to the current FPGA layout as shown in Figure 2.2, the two bootloaders are executed from the BRAM, but the applications from the SRAM. As the external SRAM chip is connected by the Processor Local Bus (PLB) to the two MicroBlazes, the connection is much slower than the connection to the internal BRAM using the Local Memory Bus (LMB). Therefore, a memory controller IP Core was included into the FPGA design, to allow data and instruction caching for parts of the SRAM. This IP Core uses parts of the FPGA's BRAM for caching purposes and offers additional Xilinx Cache Link (XCL) connections to the MicroBlazes. If a value within the cached range is addressed, the faster XCL bus is used, rather than the slower PLB. In detail, the first block of 1 MB of the total 2 MB SRAM is being cached. As the Display application is by default linked to the second block of 1 MB, the application's code and values are outside of the cached range. This leads to an increased execution time of the program code.
The code in Listing 1 was used to measure the impact of these issues. This program measures the average speed of 10,000 variable assignments, additions, integer multiplications & divisions and floating point multiplications & divisions, and prints the results to the serial console. In order to compare the performance of different memory types and locations, the code was linked to BRAM, a cached and an uncached SRAM region. The results of these measurements are listed in table 4.2.

Operation	SRAM (uncached, default)	SRAM (cached)	BRAM
Variable Assignment	36	2	1
Integer Addition	168	9	5
Integer Multiplication	1510	71	(1301)
Integer Division	4910	266	(4701)
Float Multiplication	15406	744	(15197)
Float Division	9484	449	(9275)

Table 4.2: Memory Performance Measurement Results in Processor Cycles using Software Multiplication and Division and Floating Point Unit
The results in brackets are invalid, as the libraries for software multiplication and division are stored within uncached SRAM regions.

```
// Variables
volatile int var_1, var_2 = 123, var_3 = 456, i;
int tick_counter_loop, tick_counter_addition, loops = 10000;

// Empty loop performance
tick_counter_loop =
        XTmrCtr_GetTimerCounterReg(XPAR_TMRCTR_1_BASEADDR, 1);
for (i = 0; i < loops; i++) {}
tick_counter_loop =
        XTmrCtr_GetTimerCounterReg(XPAR_TMRCTR_1_BASEADDR, 1) -
                tick_counter_loop;

// Addition performance
tick_counter_addition =
        XTmrCtr_GetTimerCounterReg(XPAR_TMRCTR_1_BASEADDR, 1);
for (i = 0; i < loops; i++)
        var_1 = var_2 + var_3;
tick_counter_addition =
        XTmrCtr_GetTimerCounterReg(XPAR_TMRCTR_1_BASEADDR, 1) -
                tick_counter_addition;

// Print results
xil_printf("Addition: %d\n\r",
        (tick_counter_addition - tick_counter_loop) / loops);
```

Listing 1: Addition Performance Measurement Code
The program starts with defining all used variables and measuring the execution time of an empty for-loop with 10,000 cycles. Afterwards, the performance of 10,000 additions is measured and the difference, divided by the number of loop cycles, printed.

As the results show, the uncached SRAM causes the highest latency, which is 36 times lower when using the fast BRAM. A typical variable assignment requires two memory accesses, first the instruction, including its value, needs to be loaded from the memory to a register to be then stored back to the variable's address in the memory. As the used memory region in the SRAM is not cached, both memory accesses need to be performed during the execution time. Therefore, each access takes at least three cycles for the bus arbitration and more cycles for data transmission, as stated in [22]. Additional delays are caused by the SRAM chip.
If the program code and data is stored in a cached region of the SRAM, the execution is in average 18 times faster for each operation, as both instructions and data will be cached.

In case of the execution from the BRAM, a variable assignment takes only one clock cycle. This means, that the instruction is read, executed and the new value written to the memory within one clock cycle.
As Table 4.2 shows, the execution time of a multiplication or division is significant longer than for an addition. The original project file of the FPGA layout states, that hardware Multiplication and Division Units for integer values and a Floating Point Unit (FPU) were included in the design of the MicroBlaze Microcontrollers. Nevertheless, the code was compiled by default using a software Multiplication and FPU. Only the hardware Integer Division Unit was used. In order to measure the difference between the software and the hardware units, the test above was executed a second time, using the hardware Multiplication, Division and FPU. Table 4.3 lists the results of this test.

Operation	SRAM (uncached)	SRAM (cached)	BRAM
Variable Assignment	36	2	1
Integer Addition	168	9	5
Integer Multiplication	10	5	7
Integer Division	168	43	40
Float Multiplication	168	13	10
Float Division	168	37	35

Table 4.3: Memory Performance Measurement Results in Processor Cycles using Hardware Multiplication and Division and Floating Point Unit

This test shows a significant improved performance of the calculations, when using the hardware Multiplication, Division and FPUs. However, both bootloaders were compiled using only the hardware Division Unit, but no hardware Multiplication and FPU. This combination may cause difficulties and needs to be further investigated before being integrated in a future software. As multiplications and divisions are only used for timestamp conversion and CAN usage calculation, the impact of these issues is low.
Both of these tests showed that the two MicroBlaze Microcontrollers can offer a much higher performance than it is currently used. As both capabilities were discovered after the design freeze of the software and require more testing, the recommendation is to include them in a already prepared future software update.

4.10.2 Boot Timing Issues

The analysis of the current architecture revealed complicated timing dependencies between the two MicroBlaze Microcontrollers. If not properly adjusted, the two Microcontrollers run into interferences during the boot process and runtime. Therefore, several sleep commands were included in the code by Stock Flight Systems. As new functionalities or an enhanced system performance change this timing scheme, it has to be carefully readjusted.

4.10.3 CAN Bus Error Handling

Robert Bosch GmbH developed the CAN Bus in 1983 as a two wired serial bus with symmetrical signal transmission. Based on this protocol, Stock Flight Systems developed CANaerospace for aeronautical applications in 1998. This higher-layer CAN protocol is used for SOFIA's CAN communication.
According to the CAN protocol ([4], [5]), each CAN controller must have two error counters: a Transmit Error Counter (TEC) and a Receive Error Counter (REC). In case any bus participant detects an error, a special formatted error message (Error Frame) will be sent to the other participants. The reception of an Error Frame leads to the incrementation of the error counters following specific rules. If a message was successfully transmitted or received, the corresponding error counter is decremented.
If any error counter exceeds the limit of 127, the CAN controller switches to the mode "Error Passive". This can be detected and signalized to the user by the continuous Built-In Test. A recovery from this mode is possible by several successfully received messages.
In case any error counter exceeds the limit of 255, the controller switches to the "Bus Off" mode and therefore detaches from the bus. An automatic recovery from this mode is not specified, as the controller is fully disconnected from the bus. Besides a power cycle of the controller, a recovery sequence has to be initiated by the host which is in this case MicroBlaze 0. The data sheet of the used Bosch C_CAN controller states: "If the device goes *busoff*, it will set **Init** of its own accord, stopping all bus activities. Once **Init** has been cleared by the CPU, the device will then wait for 129 occurrences of *Bus Idle* (129 * 11 consecutive *recessive* bits) before resuming normal operations. At the end of the *busoff* recovery sequence, the Error Management Counters will be reset" [17].
This means that the Init bit in the status register must be actively reset to initiate the recovery sequence, if MicroBlaze 0 detects the bus detachment.
The ANPA software version 1.6 by Stock Flight Systems detected the switch to "Bus Off" within an Interrupt Service Routine which was triggered by the status change. Within this ISR, not only the init bit was reset, but also the complete controller reconfigured, leading to another status change which therefore triggered the ISR for a second time. This process repeated until the C_CAN controller switched back to the nominal operation mode.
The current approach to initiate the recovery sequence didn't lead always to a successful recovery of the controller and therefore left the Annunciator Panel without communication. The newly developed software handles the bus off situation differently: the continuous Built-In Test checks the C_CAN controller status and detects the bus detachment. If this situation is detected, an error message is displayed to inform the user. Additionally, the routine for performing error countermeasures resets the init bit in the controller's status register every 10 seconds to initiate its recovery.

5 Conclusion

The development of the new Annunciator Panel software 2.0 was successful. All requirements were implemented and tested. Subsequently, the software was accepted by NASA QA and successfully installed on the spare units and will be installed on the aircraft as soon as it returns from its C-check. Furthermore, a deep understanding of the ANPA system and its limitations could be gained and documented. Based on this detailed knowledge of the hard- and software, future updates can be developed.
Two performance issues of the current system were solved during the development of the new software. By caching the display application, the performance of the Display application could be increased by a factor of 18. Additionally, the activation of hardware arithmetic units again enhanced the system performance. By changing the CAN Bus recovery mechanism, the reliability of the system was improved significantly, as a power cycle of the MCP causes a bus detachment of the ANPA.
A set of documents was generated during the design process as part of the acceptance package:

- Project Plan [7]
- Concept of Operation [6]
- System Requirements [9]
- Software Development & System Description [8]
- Verification & Validation Matrix [12]
- Test Procedure [10]
- Version Description Document [13]
- User Manual [11]

5.1 Future Updates & Upgrades

Performance

As mentioned in Section 4.10.1, due to the slow connection of the SRAM, the current FPGA design of the ANPA is causing a performance bottleneck of the applications. By moving the program code of the display application to a cached region of the SRAM,

the overall performance of the Annunciator Panel could be increased by a factor of about 18.
A second possibility is to execute both applications from the BRAM, which would increase the system's performance by another factor of 2. However, this would require increasing the size of available BRAM, in order to fit both applications. Currently, the FPGA bitstream is limiting the available BRAM to 32 KB for MicroBlaze 0 and 8 KB for MicroBlaze 1. Theoretically, the available BRAM of the used Spartan 3 FPGA is 216 KB. This means, that besides the already to the two MicroBlazes assigned BRAM and the BRAM used for the SRAM cache, there is still 144 KB of BRAM unused. This unused BRAM could be assigned to the two CPUs in a new FPGA bitstream.
Additional performance could be gained by configuring the applications to use the hardware multiplication and floating point units.

Additional Functionalities

Only a few Housekeeping Parameters are currently displayed on the two Housekeeping Pages. As there are many hundred values available in the MCP, more values could be displayed on new Housekeeping Pages.
A menu structure could make use of the ANPA's buttons and rotary encoder to display more and complex content, as demonstrated in a prototype version created by Manuel Heck.

5.2 Alternative Systems

The current Annunciator Panel system is based on hardware from the year 2000. The used FPGA offers limited possibilities for design upgrades as its logic cells are nearly completely used. Additionally, the DCU-Lx offers only the terminal mode for drawing and printing to the screen which is very limited in its possibilities and its connection to MicroBlaze 1. Due to the divided hardware design of the ANPA, upgrades of the system are very limited, as each hardware was specialized on its task. By simplifying this divided architecture to a single processor FPGA design, the available resources can be used more effective and timing issues could be avoided.
In order to further improve and develop the permanent annunciation system to a system that can gradually replace the DCC laptop with its temporary status, a hardware upgrade is highly advisable. More computing power enables displaying costly graphical indicators for hundreds of Housekeeping parameters offered by the TA. Furthermore, the ANPA could be evolved to a tool for controlling the telescope independently from the MCCS and the DCC.
A System on a Chip (SoC) like the Raspberry Pi in combination with a CAN breakout board and a touchscreen display could offer similar capabilities as the current ANPA. The Raspberry Pi offers enough computing power and is well maintained and widely used.

Glossary

Application	A computer program, which is designed to run several tasks in order to benefit the user, is called an application. In this thesis, the term application is used for the two top level programs (CAN and Display) executed on the Annunciator Panel
Bootloader	A bootloader is a small computer program, which initializes or loads a bigger software or operating system.
Bosch C_CAN Controller	A CAN controller IP Core developed and distributed by Robert Bosch GmbH.
Building	Building describes the process of generating an executable file from C code. In the beginning of the build process, the code is analyzed for lexical, syntactic and semantic correctness. This following, the code is optimized and then translated by a compiler to machine understandable object files. Finally, a linker combines the different object files to a single executable file. This process is usually done by the GNU tool GCC.
CANaerospace	CANaerospace is a higher layer CAN protocoll. It is adapted to the concept of integrated modular avionics in airplanes.
DCU-Lx	The Display Control Unit-Lx is a display controller application by F&S Elektronik Systeme running on a PicoMOD computer.
Flash Memory	A Flash Memory is a non volantile memory, which can be electonically written and erased. It is used for storing data permanently.
Flashing	Flashing describes the process of writing a new firmware version to a non volatile memory of an electronic device.

FPGA	A Field Programmable Gate Array (FPGA) "is an integrated circuit that can be customized for a specific application. Unlike traditional CPUs, FGPAs are "field-programmable", meaning they can be configured by the user after manufacturing. FPGAs contain programmable logic blocks that can be wired in different configurations. These blocks create a physical array of logic gates that can be used to perform different operations". [20]
IP Core	An Intellecutal Property (IP) Core is a reusable chip or system design, which can be purchased for integration in an own system. IP Cores for FPGAs are written in Hardware Description Language (HDL)
Linker Script	A linker script is written in the linker command language and controls the placement of the created assembly code in the memory. It is executed by the linker during the Build Process.
Makefile	A makefile is a file, containing rules for building a program. This includes the declaration of the source file and library paths, as well as the names of the compiler and linker executable. The makefile is read by the build automation tool make, which automatically invokes the different stages of the build process.
MicroBlaze	A MicroBlaze is a Microcontroller IP Core, defined by Hardware Description Language (HDL). It can be impemented in FPGAs of the company Xilinx.
Microcontroller	A processor containing not only arithmetic units, but also peripherial interfaces like General Purpose Inputs and Outputs (GPIO), is called a microcontroller.
PicoMOD	A PicoMOD is a credit card sized single circuit board computer with an ARM based processor by F & S Elektronic Systems.
SRD	A S-Record (.srd) file contains executable, ascii-encoded binary code as combination of memory address and data package.

Bibliography

[1] DLR. *About SOFIA*. 2018. URL: http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10465/706_read-264/#/gallery/285 (visited on 01/04/2018).

[2] F & S Elektronik Systeme GmbH. *DCU-Lx First Steps*. Dec. 22, 1999.

[3] Gehrz, R. D.; Becklin, E. E. *The Stratospheric Observatory for Infrared Astronomy (SOFIA)*. 2008. DOI: 10.1117/12.790973. URL: https://doi.org/10.1117/12.790973.

[4] *Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. Standard. Geneva, CH: International Organization for Standardization, Dec. 2015.

[5] *Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit*. Standard. Geneva, CH: International Organization for Standardization, Dec. 2016.

[6] Klass, L. *ANPA2 - Concept of Operations*. Tech. rep. SOF-DSI-PLA-5364.1-0002. DSI, 2018.

[7] Klass, L. *ANPA2 - Project Plan*. Tech. rep. SOF-DSI-REP-5364.1-0001. DSI, 2018.

[8] Klass, L. *ANPA2 - Software Development and System Description*. Tech. rep. SOF-DSI-SDD-5364.1-0001. DSI, 2018.

[9] Klass, L. *ANPA2 - System Requirements Specifications*. Tech. rep. SOF-DSI-SPE-5364.1-0001. DSI, 2018.

[10] Klass, L. *ANPA2 - Test Procedure*. Tech. rep. SOF-DSI-PRO-5364.1-0001. DSI, 2018.

[11] Klass, L. *ANPA2 - User Manual*. Tech. rep. SOF-DSI-MAN-5364.1-0001. DSI, 2018.

[12] Klass, L. *ANPA2 - Verification & Validation Matrix*. Tech. rep. SOF-DSI-PLA-5364.1-0001. DSI, 2018.

[13] Klass, L. *ANPA2 - Version Description Document*. Tech. rep. SOF-DSI-VDD-5364.1-0001. DSI, 2018.

[14] Krabbe, A. *Becoming reality: the SOFIA telescope*. 2003. DOI: 10.1117/12.458647. URL: https://doi.org/10.1117/12.458647.

[15] Kunz, N. *The Making of SOFIA*. Tech. rep. NASA, 2016.

[16] NASA. *SOFIA to Make Advance Observations of Next New Horizons Flyby Object*. July 9, 2017. URL: https://www.nasa.gov/feature/sofia-to-make-advance-observations-of-next-new-horizons-flyby-object (visited on 05/07/2018).

[17] Robert Bosch GmbH. *C_CAN User's Manual*. June 6, 2000. URL: http://www.keil.com/dd/docs/datashts/silabs/boschcan_ug.pdf (visited on 04/17/2018).

[18] Stock, M. *SOFIA CANaerospace Network System Description*. SOF-DSI-ICD-5350.0-0001. Stock Flight Systems, 2014.

[19] Temi, P. et al. "The SOFIA Observatory at the Start of Routine Science Operations: Mission Capabilities and Performance". In: *The Astrophysical Journal Supplement Series* 212.2 (2014), p. 24. URL: http://stacks.iop.org/0067-0049/212/i=2/a=24.

[20] The Tech Terms Computer Dictionary. *FPGA Definition*. 2018. URL: https://techterms.com/definition/fpga (visited on 05/01/2018).

[21] USRA. *About SOFIA*. 2018. URL: https://www.sofia.usra.edu/public/about-sofia/ (visited on 01/04/2018).

[22] Xilinx. *LogiCORE IP Processor Local Bus (PLB) Product Specification*. 2010. URL: https://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf (visited on 04/06/2018).

[23] Xilinx. *Platform Flash In-System Programmable Configuration PROMs*. June 6, 2016. URL: https://www.xilinx.com/support/documentation/data_sheets/ds123.pdf (visited on 04/17/2018).

[24] Xilinx. *Spartan-3 FPGA Family Data Sheet*. June 27, 2013. URL: https://www.xilinx.com/support/documentation/data_sheets/ds099.pdf (visited on 04/17/2018).

[25] Zeile, O. *DSI Top Level Design Process*. Tech. rep. SOF-DSI-QM-2000.0-0000 R00. DSI, June 26, 2012.