A NETWORK PROCESSING NODE FOR LIGHT UNMANNED AIRCRAFT

Timothy William Arrowsmith
University of Kentucky, tim.arrowsmith@uky.edu

Abstract of Thesis

A NETWORK PROCESSING NODE FOR LIGHT UNMANNED AIRCRAFT

Over the last decade, research into unmanned and autonomous vehicles has greatly increased. With applications ranging from science and exploration to humanitarian and military efforts, the rising need for autonomous vehicles demands constant innovation and growth. The Intelligent Dependable Embedded Architectures (IDEA) lab at the University of Kentucky is continually launching research oriented programs [1]. A few key projects focus on the development of Unmanned Aerial Vehicles (UAV). Through this research, at the University of Kentucky, the need to develop a reliable, lightweight, node based hardware for use in light UAVs and other unmanned and autonomous vehicles became apparent. This paper addresses the design and implementation of a network processing node for light UAVs. This system utilizes a Controller Area Network (CAN) noise tolerant communications bus, a low power ZigBee Wireless Network for expanded inner plane communications and Silicon Laboratories C8051F041 microcontrollers to provide the necessary inputs/output and data processing. The final result will be a flight ready light UAV featuring distributed processing nodes to handle the servo communications and controls.

KEYWORDS: Unmanned Aerial Vehicle, Embedded Processing, Avionics Bus Communications, Controller Area Network, ZigBee Wireless Network

A NETWORK PROCESSING NODE FOR LIGHT UNMANNED AIRCRAFT

By

Timothy William Arrowsmith

____
Director of Thesis

____
Director of Graduate Studies

____

THESIS

Timothy William Arrowsmith

The Graduate School
University of Kentucky
2007

A NETWORK PROCESSING NODE FOR LIGHT UNMANNED AIRCRAFT

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering at the University of Kentucky

By

Timothy William Arrowsmith

Lexington, Kentucky

Director: Dr. James E. Lumpp, Professor of Electrical Engineering

Lexington, Kentucky

2007

TABLE OF CONTENTS

List of Tables……………..……………………………………………………………v

List of Figures…………………………………………………………………………vi

1 Introduction 1
  1.1 Background 1
  1.2 Motivation 2
    1.2.1 BIG BLUE 2
    1.2.2 AIRCAT/ Southern Komfort 4
    1.2.3 UAV Network Design 6
  1.3 Problem Definition 7
    1.3.1 Requirements 8
    1.3.2 Alternatives 8
    1.3.3 Objectives 9
  1.4 Thesis Overview 10
2 Hardware Design 11
  2.1 Introduction 11
  2.2 Microcontroller and Peripherals 11
    2.2.1 Processor 11
    2.2.2 Communications 12
    2.2.3 Power 12
    2.2.4 Headers/Connectors 13
    2.2.5 Versions/Revisions 13
  2.3 Wired Communication 14
    2.3.1 CAN Networking 15
  2.4 Wireless Communication 16
    2.4.1 ZigBee/802.15.4 17
    2.4.2 Modules 19
    2.4.3 Antenna Design 20
  2.5 Summary 21
3 Software Design 22
  3.1 Introduction 22
  3.2 Generic Initialization 22
  3.3 CAN Driver 23
  3.4 Wireless Driver 24
    3.4.1 Wireless Network Design Considerations 24
    3.4.2 XBee Operation Modes 26
    3.4.3 Custom Wireless API 27
  3.5 IDEAnix 29
  3.6 Summary 30
4 Implementation 31
  4.1 Introduction 31
  4.2 Overview of Design 31
  4.3 CAN Module Hardware 32
    4.3.1 C8051F041 Microcontroller 32
    4.3.2 Communications 33
    4.3.3 Power 35
    4.3.4 Headers/ Connectors 36
    4.3.5 Other Hardware 37

4.4 Wireless Hardware 38
4.4.1 Daughter Board 39
4.4.2 Module Interface 40
4.5 Software 40
4.6 Fabrication 41
4.6.1 Board CAD 41
4.6.2 On-campus PCB Prototyping 42
4.6.3 Professional PCB Prototyping 44
4.6.4 PCB Population 44
4.7 Summary 46
5 Testing 47
5.1 Introduction 47
5.2 Preliminary Testing 47
5.2.1 Bit Flipping, SI Labs Application Example 47
5.2.2 Servo/Power Noise Testing 48
5.2.3 Range Testing 50
5.3 Application Testing 50
5.3.1 IDEAnix Integration 51
5.3.2 Flight Simulation Testing 52
5.4 Future Work 54
5.5 Summary 55
6 Summary 56
7 Appendix 57
8 References 61
9 Vita 64

LIST OF TABLES

Table 3.1 – Function Error Codes 29
Table 4.1 – Parts List and Component Pricing for the Tiny Interface Module (12/4/2006) 32
Table 5.1 – Packet Errors Versus Data Rate and Range 50

LIST OF FIGURES

Figure 1.1 – Left Image- Predator, Right Image- Global Hawk [3] 1
Figure 1.2 – Single Processor Design 3
Figure 1.3 – Distributed Processing 3
Figure 1.4 – BIG BLUE 3 Flight Photo 4
Figure 1.5 – AIRCAT Airframe [7]. 5
Figure 1.6 – Entry for the 2006 AUVSI Student Competition 6
Figure 1.7 – Target UAV Network Design 7
Figure 1.8 – ReadyUAV Conceptual Design 7
Figure 1.9 – Silicon Laboratories C8051F040 Development Kit vs. the Custom Module 9
Figure 2.1 – Example of CAN Bus Configuration 16
Figure 2.2 – 802.15.4/ZigBee Layer Model 18
Figure 2.3 – Freescale MC13201 Transceiver 19
Figure 2.4 – MaxStream XBee-Pro Module 20
Figure 3.1 – Default Pin Configuration. 22
Figure 3.2 – CAN Driver API 23
Figure 3.3 - ZigBee Network Topologies 25
Figure 3.4 – Wireless Driver Input\Output Block Diagram 28
Figure 3.5 – Network Software Framework 30
Figure 4.1 – Tiny Interface Module Hardware Block Diagram 31
Figure 4.2 – Silicon Labs C8051F041 Microcontroller 33
Figure 4.3 – CAN Transceiver 34
Figure 4.4 – Maxim Dual RS-232 Transceiver 34
Figure 4.5 – Capacitors, Pull-up Resistors, and Voltage Regulator 35
Figure 4.6 – Module Headers 37
Figure 4.7 – Reset Button and Power and Status LEDs 38
Figure 4.8 – Wireless Daughterboard Evolution 39
Figure 4.9 – Daughter Board Mounted on Module 40
Figure 4.10 – TIMCAN Hardware Board Footprints (Oldest on Left to Newest on Right) 41
Figure 4.11 – A) Multisim and B) Ultiboard Screen Captures 42
Figure 4.12 – ECE Department Milling Machine 43
Figure 4.13 – ECE Department Through Hole Plating Bath 43
Figure 4.14 – ECE Department Solder Paste System 45
Figure 4.15 – ECE Department Solder Reflow Machine 45
Figure 4.16 – Hand-Solder Station 46
Figure 5.1 – Preliminary Testing Configuration 48
Figure 5.2 – Servo Pulses from Module 49
Figure 5.3 – Servo Pulse with Load 49
Figure 5.4 – CAN Transmit vs. Receive Delay 51
Figure 5.5 – Simulation Block Diagram 52
Figure 5.6 – Piccolo Ground Station Running a Simulation for Testing 52
Figure 5.7 – Six Volt Power Line with Load 53
Figure 5.8 – In Aircraft Testing of Module and CAN Bus 54
Figure 5.9 – Module Mounted in a Hobby Box 55

1 Introduction

1.1 Background

Unmanned Aerial Vehicles (UAVs) have been developing and changing for as long as humans have understood flight. Once primarily seen as military assets, many of the early unmanned aircraft were barely more than guided missiles. With continual development however, the technology surrounding unmanned flight grew, along with the number of applications. Now many scientific and educational institutions realize the value of the implementation of UAVs. These tools can provide cheap and effective ways to gather information and survey areas and situations that once were not possible.

Two of the better known UAVs are the United States Military's Global Hawk and Predator, seen in **Figure 1.1.** The Predator is considered a medium-altitude, long-endurance, remotely piloted craft **[2]**. This craft typically operates with a ground station crew of three, including one pilot and two people to monitor and operate sensors, communicating with the craft directly or via satellite.

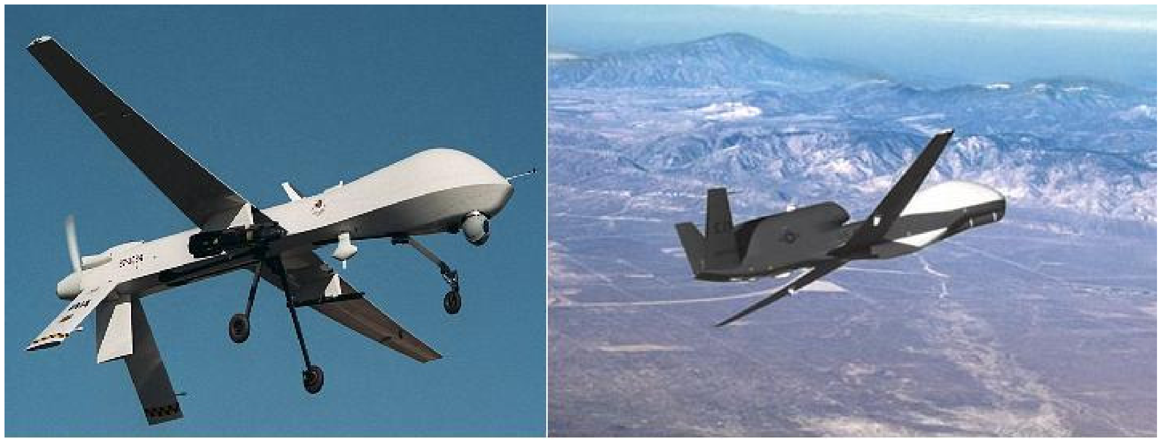


Figure 1.1 – Left Image- Predator, Right Image- Global Hawk [3]

In contrast to the Predator, the Global Hawk is a fully autonomous UAV. This craft is capable of autonomously taxiing, taking off, completing an aerial mission, returning and landing. The ground station crew for Global Hawk is solely responsible for monitoring the craft's health and status and updating mission parameters **[4]**. Though well known both of these systems are too large and expensive for most non-military applications.

One way that unmanned flight reaches into more applications is the parallel development of autopilots. The integration of a well designed autopilot eliminates the need for trained and experienced pilots to design, develop, test, and finally implement a UAV. Most autopilot systems consist of an in-plane sensor and control device connected via radio to a ground station. The ground station is responsible for notifying the flight unit of the flight path and parameters, as well as allowing a user to modify flight elements in real time. The in-plane unit is also responsible for guiding the aircraft's flight control surfaces.

1.2 Motivation

There have been several Unmanned Aircraft Systems (UAS) platforms designed, implemented and tested at the University of Kentucky, each having its own unique features and designs. Three UAS platforms implemented at the University of Kentucky are BIG BLUE, the 3rd Annual AUVSI competition airframe AIRCAT and the 4th Annual AUVSI competition airframe Southern Komfort. Due to the size of these projects the need for a modular firmware design was quickly realized and through the many iterations of design and physical implementations issues with noise and payload integration became apparent. These issues cause delays in flight time and flight testing, as well as increased development time.

1.2.1 BIG BLUE

The Baseline Inflatable-wing Glider, Balloon Launched Unmanned Experiment (BIG BLUE) was the University Of Kentucky College Of Engineering's first journey into UAS design **[5]**. As part of a NASA workforce development grant this program introduces students to a multidisciplinary aerospace project with the goal of designing and testing novel Mars exploration aircraft technology. The first generations focused on development of inflatable/rigidizable wings along with a custom flight control system seen in **Figure 1.2**. This system focused around a single microcontroller utilizing functions that communicated information directly between the various sensors, actuators and input/output. This system was unable to support fault tolerance and due to the direct communication between functions, a single, small error could result in catastrophic system failures.

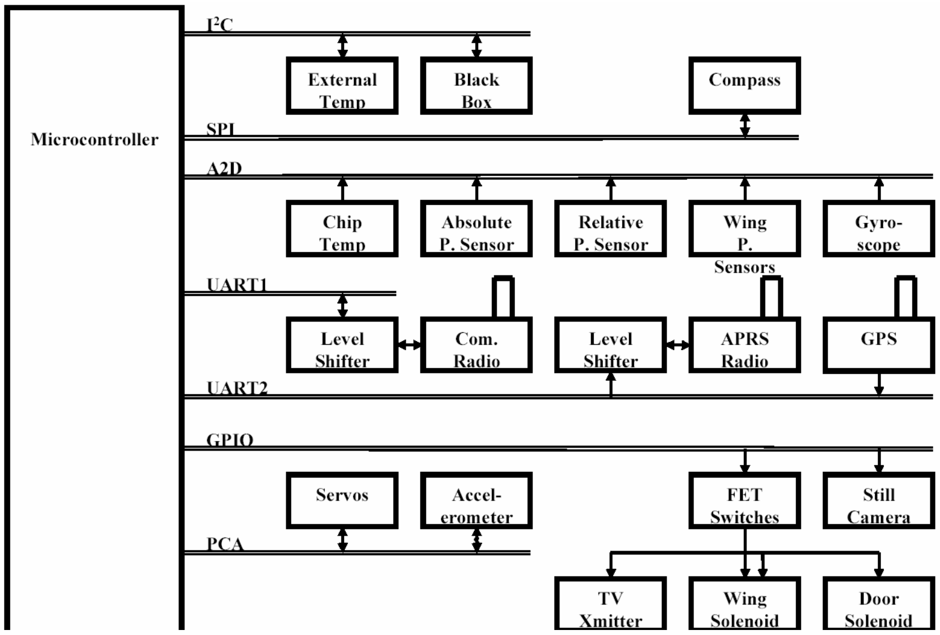


Figure 1.2 – Single Processor Design

The second generation of BIG BLUE utilized distributed processing integrating several processors into the design. These processors, communicating through a common data bus (**Figure 1.3**), provided hardware redundancy, having duplicate sets of sensors tied directly into some of the mission critical components. This system still suffered from the single point of failure provided by the I^{2}C communication bus. This I2C bus also added additional hazard by being susceptible to noise, potentially delivering incorrect or incomplete data transmissions.

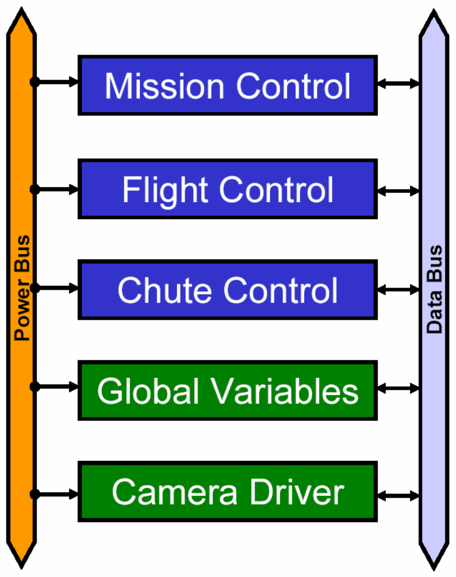


Figure 1.3 – Distributed Processing

The third generation of BIG BLUE focused on integrating a single microcontroller mission controller, a single daughterboard housing various sensors/actuators and an embedded operating system. The operating system selected was MicroC/OS-II **[6]** because it is preemptive, task orientated, and includes libraries that support all common functionality and data structures of a high-level operating system. These combined features allow the use of a modular firmware design. This characteristic was desired in a large development project such as BIG BLUE since it allows the design elements to be divided among the participating teams.

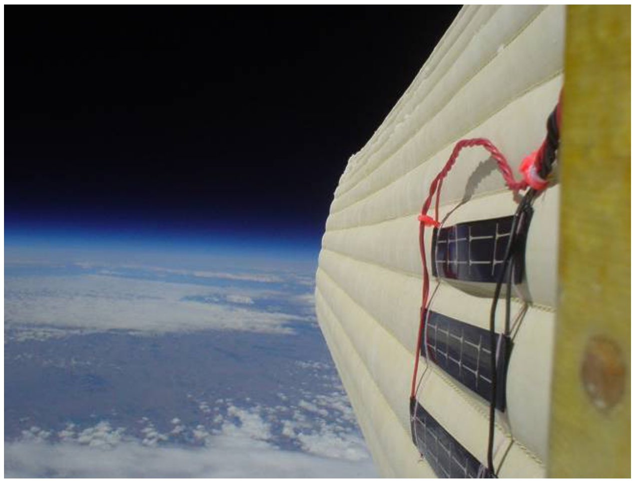


Figure 1.4 – BIG BLUE 3 Flight Photo

1.2.2 AIRCAT/ Southern Komfort

The Airborne Intelligent Research Craft for Autonomous Technology (AIRCAT) system was designed by a combination of electrical and mechanical engineering students under a workforce development grant in the spring of 2005. The goal of this project was to develop a completely custom, novel entry to the 3rd Annual AUVSI Student Unmanned Aerial Vehicle (UAV) Competition. The AIRCAT system consists of a custom airframe, onboard autopilot, camera system, battery based power system and ground station.

Figure 1.5 – AIRCAT Airframe [7]**.**

The onboard electronics of the AIRCAT airframe consists of a central mission control processor, a MicroPilot MP2028 autopilot, a two-way data modem, infrared and visible light digital cameras, and Amateur Television (ATV) streaming real-time video **[8**]. The mission control processor is an 8-bit microcontroller from Silicon Laboratories. This system incorporated the embedded real-time OS MicroC/OS-II into its firmware design due to the need for dedicated code segments, to monitor and control multiple subsystems in real-time. Because several subsystems including ATV, the data modem and the MP2028 communications radio needed to transmit live signals at the same time, electromagnetic interference (EMI) problems occurred. This required the incorporation of shielding into the design. In order to reduce the effects of the EMI, a more physically modular design was investigated. Such a modular design would use a noise tolerant communications bus to connect a distributed set of nodes together preventing noise propagation throughout the system.

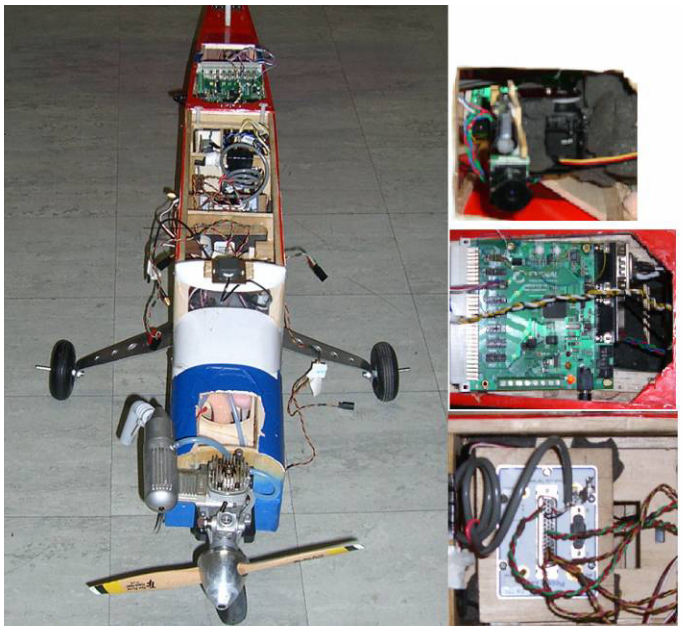


Figure 1.6 – Entry for the 2006 AUVSI Student Competition

Southern Komfort is the most recent UAS platform designed at the University of Kentucky competed at the 4th annual AUVSI UAV Competition in June 2006 **[9]**. As seen in **Figure 1.6** this UAS utilizes a conventional high-wing R/C model airframe, a Piccolo Autopilot and custom electronics for flight control and mission payload control. This was coupled with high-bandwidth communications and state-of-the-art image processing capable of merging collected images into a composite photograph of a large geographic area. Though with the integration of the new autopilot system the craft experienced a great improvement in flight reliability, the need still existed to develop a fault tolerant distributed system.

1.2.3 UAV Network Design

The target design of a light UAV network, seen in **Figure 1.7**, focuses on distributed processing utilizing a fault/noise tolerant communication bus. This system could then be easily utilized by many applications including the 2007 AUVSI Student UAV Competition. By utilizing the distributed processing a digital signal can be used to transmit servo positioning data to the respective network node. This node can then decode the data and produce the necessary analog servo signal which only needs to be transmitted a short distance, reducing the effects of EMI. Another application would be in the development of ReadyUAV.

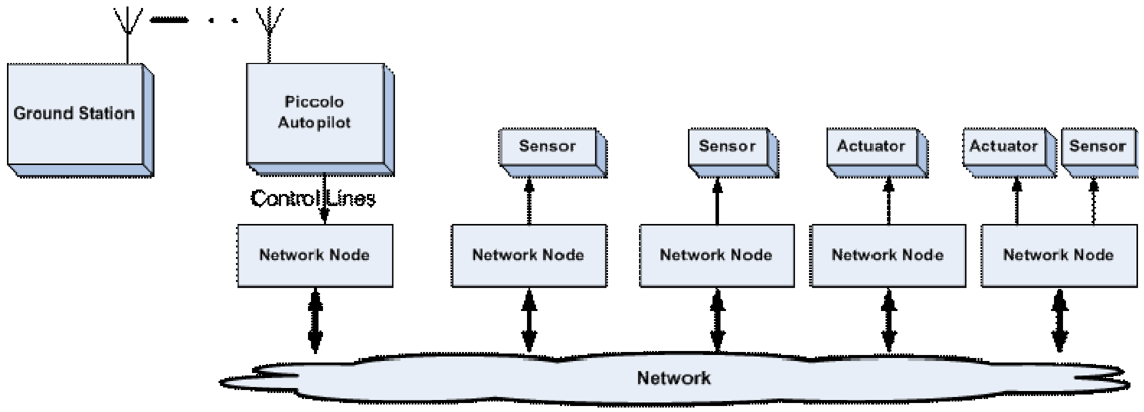


Figure 1.7 – Target UAV Network Design

ReadyUAV is an in-development prototype for a rapidly deployable, easily reconfigurable, light UAV. This aircraft is designed to allow a novice user to quickly deploy the craft in a variety of support or reconnaissance roles. A modular frame allows the user to quickly change out various payloads, including cameras, autopilots, or other sensor packages, while maintaining a standard nose, tail and set of wings (**Figure 1.8**). The University of Kentucky IDEA lab developed the necessary tools to design the software layer. Ardea (Automatically Reconfigurable Distributed Embedded Architecture) will provide the ability to dynamically reconfigure with the addition or subtraction of software nodes **[10][11]**. This system simply needs the hardware framework to support Ardea. By utilizing a bus network that automatically reconfigures with the addition or subtraction of components, ReadyUAV can become a reality.

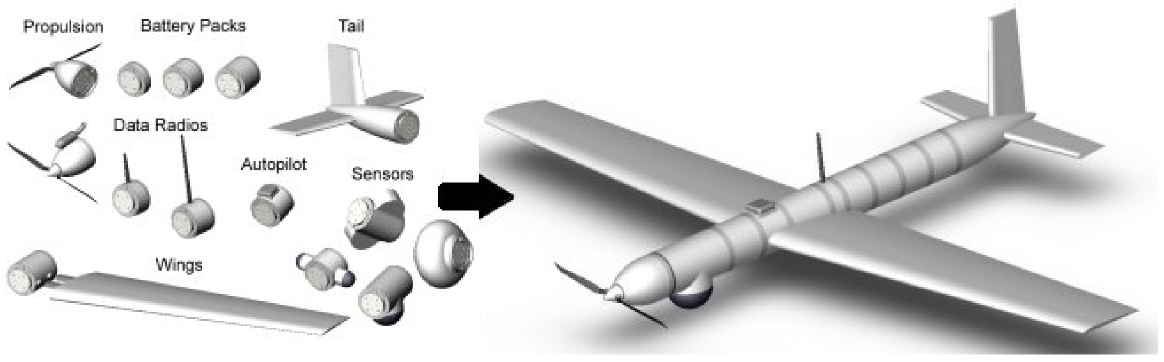


Figure 1.8 – ReadyUAV Conceptual Design

1.3 Problem Definition

In order to produce a rapidly deployable, easily reconfigurable, light UAV, a solid communications network is required. This network, wired or wireless, must efficiently deliver control information to the various different nodes, and in turn control the respective flight sensors or servos. Beyond this capability, the system should handle the addition or removal of payloads without affecting system or flight stability. Finally, the system must provide this functionality in a package that is easily understood and integrated into existing and future embedded light aircraft designs.

1.3.1 Requirements

The development of electronics affiliated with light unmanned aircraft must satisfy many of the same base requirements that guide the development of the aircraft itself. The end result must maintain a small form factor, be lightweight, and have reasonable power consumption. Beyond these base requirements, the resulting system must completely satisfy current functionality demands as well as provide room for future growth. The current demands include the ability to seamlessly integrate the IDEAnix operating system that is being developed in parallel, utilize the reliable Controller Area Network (CAN) wired network, and developed on a silicon laboratories 8051 microcontroller. The future growth is addressed by the exploration and implementation of an 802.15.4/ZigBee based wireless network layer as well as providing additional hardware resources. The specifics of each hardware and software requirement are explored in the respective hardware and software design chapters.

1.3.2 Alternatives

Currently there are no commercial off-the-shelf (COTS) alternatives available that satisfy the design requirements. One of the closest matches is a C8051F040 development kit from Silicon Laboratories **[12]**. Based off the same line of microcontrollers targeted in development of the custom module, this option offers all of the same processing functionality. This development kit, however, is too large for light unmanned aircraft as can be seen by the comparison in **Figure 1.9**.

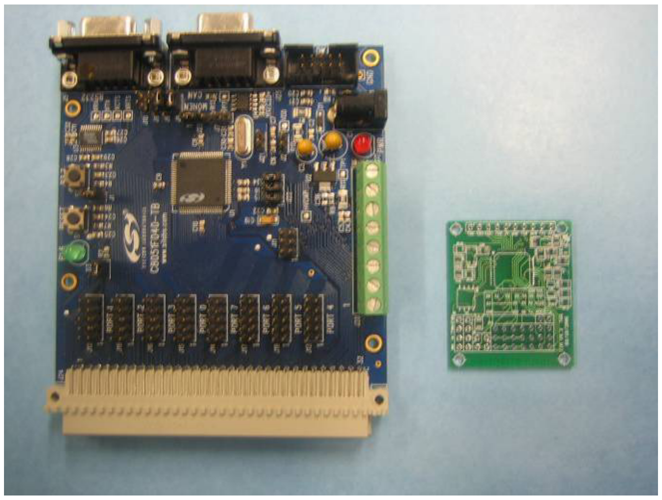


Figure 1.9 – Silicon Laboratories C8051F040 Development Kit vs. the Custom Module

Other microcontroller's such as Freescale Semiconductor's HC08GZ family or the Micro/sys Inc. MCB58 also lack the critical design element of being easily integrated with IDEAnix and other lab code repositories **[13]**[**14**]. Manufacturers specialize in selling microcontrollers and other components to be included as part of consumers' designs. The commercial market for developing networking hardware for light unmanned aircraft is not large enough to support the costs of development.

From the perspective of developing a wirelessly networked version of the system Silicon Laboratories again provides the best COTS alternative through a joint project with Helicomm **[15]**. This module, designed including a Silicon Laboratories C8051F133 microcontroller, would provide the wireless link and easy integration utilizing current code repositories. As with the development of a wired solution, however, there are no exact matches. This option would not provide the CAN networking that is provided with the base module and daughter card design. There are a number of alternative radio options available that could be utilized in the base module and daughter card design. The 802.15.4/ZigBee compatible XBee Pro module from MaxStream was chosen to support the wireless functionality. The rationale for choosing this module over alternatives is covered in the Hardware Design chapter.

1.3.3 Objectives

The objective of this design is to provide a lightweight, low power, small form-factor distributed networking node for aiding the development of light unmanned aircraft at the University of Kentucky. This Tiny Interface Module should be able to be quickly adapted by students working to develop current and future light unmanned aircraft systems, as well as provide a stable platform for a majority of future lab embedded design applications. The end result should meet the performance of COTS alternatives and ultimately provide a more customized solution than existing options.

1.4 Thesis Overview

The text above provides background information into the history and development of UAVs, internally and externally to the University of Kentucky, as well as outlining the objectives and requirements that guided the development of the Tiny Interface Module. The second chapter will address the specific hardware design decisions that are required to provide a solution that provided a balance between convenience and size. In chapter three, the software design decisions surrounding the future utilization of the module as well as the development of a wireless driver are discussed. Chapter four covers the implementation of the module with the final selection of hardware components as well as fabrication and population of the Tiny Interface Module. Chapter five describes the testing that was performed, including the integration of IDEAnix, to verify the module's operation along with and explanation of the test results. Finally, chapter six concludes with a summary.

2 Hardware Design

2.1 Introduction

This chapter addresses the major decisions behind the hardware design of the Tiny Interface Module. This process involved a researching existing design techniques and principals and applications, as well as collaborating with the students and professors what features would most likely be needed and utilized in the module. This process begins with the selection process for the base microcontroller and supporting peripherals. Next, the factors leading to the selection of the main communications bus are addressed. Finally, the design decisions for selecting a wireless network are covered, followed by a summary.

2.2 Microcontroller and Peripherals

The Tiny Interface Module has been designed to provide a small form factor processing and input/output device for light Unmanned Aerial Vehicles. This functionality requires a large degree of flexibility to adapt to any UAV application while maintaining a simplified set of features. Providing this balance of functionality and simplicity required careful selection of processing, communications, power regulation, and headers/connectors.

2.2.1 Processor

Selecting the microcontroller to handle the processing needs of any systems is a sensitive task. The key factors to consider during selection are the system constraints, operating temperature range, power consumption, packaging and cost. Next, the baseline performance must be determined by examining the data range to be handled and the precision, as well as speed, of calculations. Finally, the system input/output, program memory and RAM requirements must be investigated for any unusual or special requirements **[16**]. Influenced by any number of these factors, the final decision is based on what microcontroller will provide the required functionality and a welcoming user experience. For the development of small form-factor processing in light UAVs, the Silicon Laboratories' 8051 product line's stability, capabilities and cost efficiency distinguished it as the best option.

As the platform of choice for a number of previous and ongoing projects, the Silicon Laboratories 8051 microcontrollers have earned the respect of the IDEA lab. More importantly, however, they have also grown a respectable repository of experience and intellectual property exclusive to the lab. Being able to quickly adapt and integrate existing code will help to ensure the module's success.

The specific product line chosen was the C8051F04x. This line was selected since it provided a stable 8051 based embedded processing platform with integrated Controller Area Network (CAN) communications capabilities. The microcontroller specifications are covered in Section 4.3.1.

2.2.2 Communications

The main communications protocol chosen must allow for reliable distributed embedded processing. Through earlier research and experimentation performed within the IDEA lab, the Controller Area Network **[17]** was determined to provide the needed speed, flexibility and reliability necessary to support a small-aircraft avionics bus **[18]**. The selection of CAN is discussed further in **Section 2.3**.

Though selection of the main communication protocol was crucial, it was not the only communication option included in the hardware. Other communications protocols supported by the Silicon Laboratories C8051F04x microcontroller line include universal asynchronous receiver/transmitter (UART), Inter-Integrated Circuit (I^2C), and serial peripheral interface (SPI) **[20]**. These communication protocols provide added functionality to the module through the addition of peripherals or testing and user interaction. With the goal of future growth, a wireless networking option was also developed. The CAN and wireless networking options are discussed further in **Sections 2.3** and **2.4** respectively.

2.2.3 Power

The power system for the Tiny Interface Module required the ability to regulate an input power source to a level that was usable by the chosen components and the conditioning of that power to ensure stability. Historically, linear regulators are the base for most power supplies used in electronics **[21]**. Voltage regulators take an input voltage, possibly varying within a set range, and provide constant output voltage independent of the load. Selection of an appropriate voltage regulator can be done by noting requirements of: the maximum load current, the type of input voltage source (AC or battery), the output voltage precision, the quiescent (idle) current and any special required features such as a shutdown pin or error flag.

The power conditioning portion of the power system is performed by using bypass or decoupling capacitors. These parts are used to store small amounts of energy locally so that changes in current demand can be responded to quickly. This aids in reducing the effect of line the inductance resulting from quickly changing current demands in PCB lines. It is important to place these capacitors as close to the board component that they are bypassing to minimize line inductance, between the capacitor and the part, as well as to ensure the capacitor is operating at an frequency band where it is most effective **[22]**. The effective frequency is essentially the time it takes for the transient current demand of the part to see relief from the capacitor. This distance should be less than ¼ wavelength, ideally placement is at least $^{1}/_{10}$th of quarter wavelength.

2.2.4 Headers/Connectors

Choosing the headers and connectors to interface with the module relied heavily on footprint, weight, personal preferences and availability. Options such as DB-9 and RJ-45 offered convenience due to their availability as standard serial, or Ethernet, connectors. They do, however, have the disadvantage of bulky end connectors. In a small form-factor design, bulky connectors can quickly dissipate the design's size and weight advantages.

Many customized connector options are available through electronics hardware distributors. These options have all been carefully designed and fabricated for use in various systems. Unfortunately, when dealing with custom connectors, inflexibility and supply availability are always a concern. These factors do not rule out using a custom connector, but they should be carefully surveyed before committing to a particular option.

One other option considered was to use a basic straight-pin header. This choice provides a small footprint, minimal weight, and great flexibility for the development of custom connectors. Also, as a staple component of any electronics lab, the supply of acceptable part compliments is often large. Straight-pin headers do, however, require that all harnesses be custom fabricated and do not offer the physical connection security that comes with the clips and screw-posts of some options.

2.2.5 Versions/Revisions

As with any project, the Tiny Interface Module development underwent several major design revisions. These versions were marked by minor changes in the design and integration goals, resulting in significant hardware design changes. The first stage of revisions was marked by adding the ability to isolate the power line, supplied by the custom CAN harness, and the voltage supply line for the servo headers. An additional servo header was also added in this stage to allow a single module to directly interface with all four of the Piccolo autopilot's servo outputs. The next version involved the reduction of the board footprint to match that of a standard RC aircraft servo and the addition of mounting holes. Finally the latest revision came with the addition of a RS-232 level shifter to simplify harnessing, a larger voltage regulator to compensate for the wireless module's increased current demand, and connection of the microcontroller's I^2C lines to a header.

2.3 Wired Communication

As cited earlier, the decision to use a Controller Area Network (CAN) bus as the main form of communication came about through previous IDEA lab research and experimentation. It is, however, still important to understand other available options. An obvious option exists to use some form of Ethernet communication. With this option, fault tolerance can be achieved, but as a networking system designed for large scale networks, the hardware overhead would be cost and size prohibitive.

Another option would be to implement a proprietary option such as LonTalk from Echelon Corporation. This option however has the potential hazard of containing "Black Box" components **[23]**. Without understanding the core functions of the network, there is a risk that the designed solution will be poorly tailored to the desired application. Also, when dealing with proprietary solutions, there is the possibility of becoming locked into a required or limited set of development tools.

Two of the more promising alternatives to CAN are the Time-Triggered Protocol (TTP) and the newly developed FlexRay. FlexRay, developed in cooperation by companies such as BMW, Bosch, DaimlerChrysler, and Freescale, provides up to a 20 Mbps data rate, increased data frame size and provides the framework for integrated fault tolerance through a dual bus design and a Bus Guardian **[24]**. TTP is targeted at a broad range of safety critical applications focusing on safety, composability, the ability to modify or add functions without affecting system stability, and not limiting the data rate or throughput **[25]**. With the currently available hardware TTP can support 25Mbps synchronous transmission and can have data payloads up to 240 bytes.

Ultimately, though, alternatives to CAN do exist. CAN however, is the correct networking choice for developing this light UAV system. CAN provides the necessary hardware to support design growth and advances in the fault tolerance. This solution also has a large selection of communication hardware and a large system design knowledgebase to aid in future developments.

2.3.1 CAN Networking

CAN was first developed in the mid 1980's as a reliable communications bus for automotive applications. With the increased complexity of vehicles, it was no longer sufficient to connect each related automotive component directly. Instead of each component having analog and digital connections, all controllers and peripherals need only one CAN connection **[26]**. This original specification featured a start-of-frame bit, 11-bit arbitration identifier, remote transmission request bit, data length code, data field, cyclic redundancy check, acknowledge slot (for receiving nodes to acknowledge receiving data) and finally an end of frame delimiter.

With the ever gaining acceptance of CAN and its possible application in other system designs, CAN 2.0, featuring a 29-bit identifier, was released in 1991 **[17]**. This option, referred to as CAN 2.0B or extended, allowed a flag to be thrown between the 11-bit arbitration identifier and the remote transmission request telling the receiving device that another 18-bit arbitration identifier would follow. After the 18-bit id is received, the CAN message is the same as the 11-bit version. CAN 2.0B is backwards compatible with CAN 2.0A, the 11-bit identifier, whereas a CAN 2.0A device will simply ignore a CAN 2.0B message.

The CAN functionality included with the Silicon Laboratories C8051F04x microcontrollers can operate at bit rates up to 1Mbit/second **[20]**. The integrated Bosch CAN controller includes 32 message objects that can be configured to send or receive data. All data filtering and transmission protocols are handled within the CAN controller and not Silicon Laboratories proprietary CIP-51 processing core. All interaction between the core and the CAN controller is performed though Special Function Registers (SFRs). The bulk of the CAN bus's tolerance to noise and interference comes from the use of differential signaling. An example of a typical CAN bus configuration can be seen in **Figure 2.1**.

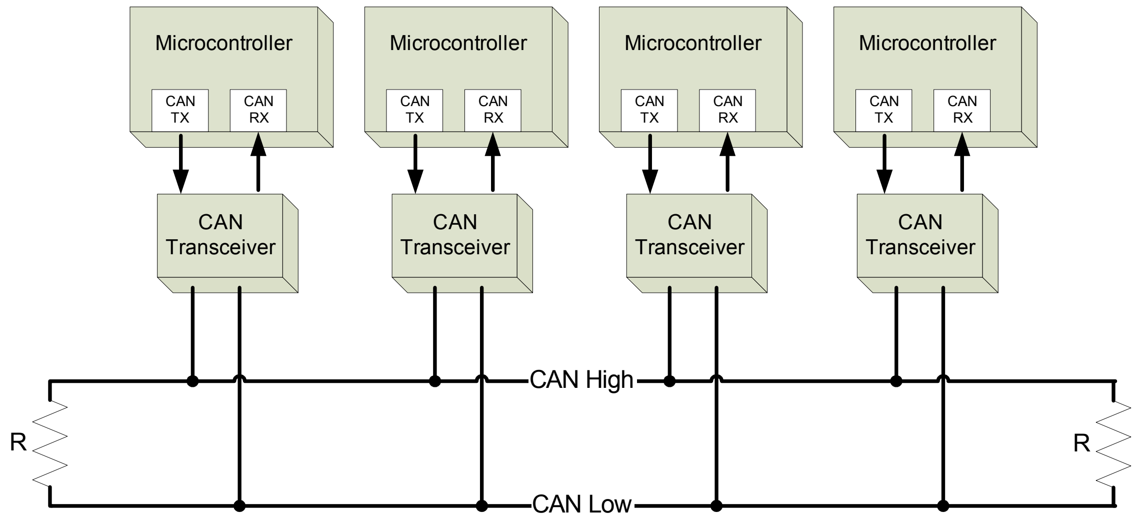


Figure 2.1 – Example of CAN Bus Configuration

One of the biggest advantages of utilizing a CAN bus is the use of balanced differential signaling. The differential signaling functions by having both lines in a resting state of approximately 1.5 volts. When data is transmitted a high value on the bus will take the voltage on CAN High from 1.5 to 2.5 volts, while taking CAN Low from 1.5 to .5 volts. The current in each signal line flows in equal but opposite directions; this in combination with twisted pair cabling creates a field-canceling effect for low noise emissions **[19]**.

2.4 Wireless Communication

Recently there has been a great effort to expand wireless connectivity. Many designs are focused around the industrial, scientific, and medical (ISM) frequency bands. In the United States the most common of these frequency bands are 900MHz, 2.4 GHz, and 5.8 GHz. There are many options utilizing these frequencies, the most commonly known include Wireless Local Area Network, WLAN, (802.11), Wireless Personal Area Networks, WPAN, (802.15), and Broadband Wireless Metropolitan Area Network, WiMAX, (802.16) **[27]**[**28**][**29**]. Of these available standards, the WPAN is the best option to produce the desired in-plane networking.

Within the WPAN specification, however, there exist four task groups: 802.15.1 – Specifications for Wireless Persona Area Networks, 802.15.2 – Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in Unlicensed Frequency Band, 802.15.3 – High Rate Wireless Personal Area Networks, and 802.15.4 – Low Rate Wireless Personal Area Networks. At this time, the most notable of these are Bluetooth (802.15.1) and ZigBee (802.15.4).

Bluetooth and ZigBee provide competitive and comparable features. Bluetooth, however, is designed primarily for ad-hoc type networks with one master and multiple slave devices. The typical Bluetooth network is seen in devices that have relatively high power demands and can be recharged frequently. A standard ZigBee node is designed to have a battery lifespan of months to years. Bluetooth networks currently have a limitation of eight active nodes **[30]**. ZigBee, on the other hand, utilizes star, mesh, and cluster tree networks and therefore has the addressing capabilities to support over 65,000 nodes per network **[31]**. As a tradeoff for the increased node capacity, ZigBee networks communicate at serial speeds of at most 115200 bits per second (bps) as opposed to Bluetooth's 1 Mbps.

The benefits of using a wired network include higher bandwidth communications than in wireless, with good physical connections data is more likely to arrive at the intended destination, and wired connections can be extremely noise tolerant. WPAN solutions, however, have advantages with greater flexibility. They can add and remove nodes without infrastructure changes and extend the local network externally over short distances to other nearby nodes or debug interfaces.

2.4.1 ZigBee/802.15.4

The standard chosen to implement the wireless networking functionality of the Tiny Interface Module is based off the IEEE 802.15.4 -2003 standard **[32]**. This specification operates at 868MHz in Europe, 915MHz in the United States or 2.4 GHz worldwide utilizing Direct Sequence Spread Spectrum (DSSS). The physical layer (PHY) and medium access control (MAC) layer control access to the radio channels utilizing Carrier Sense Multiple Access with Collision Detection (CSMA-CA). With CSMA-CA each node in a network must notify the other nodes that it intends to transmit before data is transmitted **[33]**. **Figure 2.2** is a high level view of how the layers of 802.15.4 and ZigBee relate.

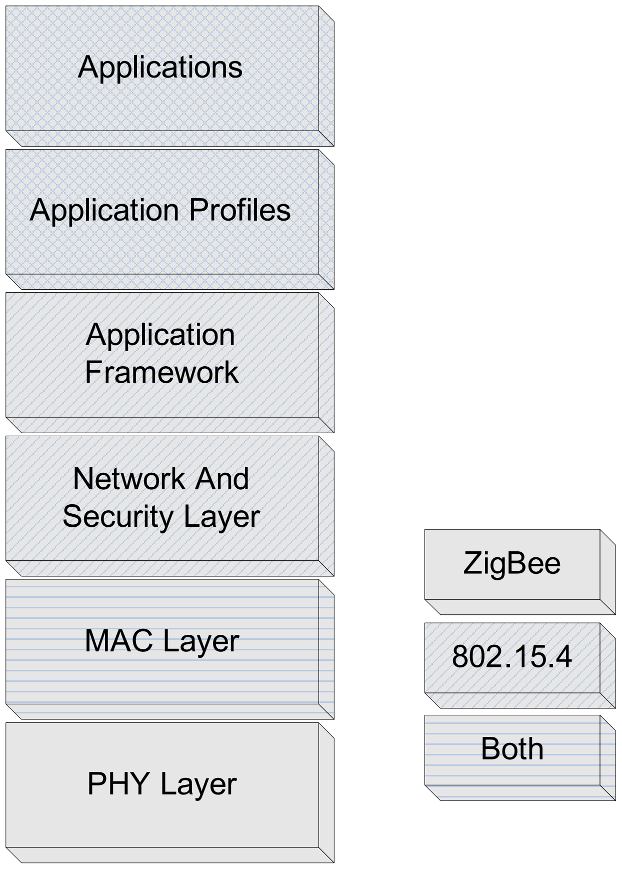


Figure 2.2 – 802.15.4/ZigBee Layer Model

On top of the PHY and MAC layers, additional ZigBee layers are implemented. These layers, defined by the ZigBee Alliance, are the network, security and application layers. It is through these layers that functionality, such as forming star, mesh, and cluster-tree network topologies, is implemented.

The ZigBee platform has many advantages and disadvantages. ZigBee is a newly developing standard which provides ample opportunity to work with and shape early prototypes and designs. The target audience for ZigBee development and integration is in low-power, low-cost, simple home, office and industrial applications. The low-power and low-cost features make this option very appealing to the light UAV market. Due to bandwidth limitations, 115 Kbps versus the CAN 1Mbps, this option will require careful implementation as to ensure that the system does not become overloaded.

2.4.2 Modules

To implement a ZigBee/802.15.4 network, there are numerous manufacturers and components available. The products range from Freescale Semiconductor's line of simple low power transceivers to more complete systems like MaxStream's XBee module **[34]**. There are merits and disadvantages associated with the selection of each option. For example a Freescale Semiconductor MC13201 (**Figure 2.3**) provides full 802.15.4 compliance in a small form-factor part. This part can be connected as a slave device, via SPI, to most microcontrollers **[35]**. The cost of having such a small form factor component is that it only provides a transceiver. To fully implement this part into a working network also requires the addition of an external crystal oscillator, software customization, and the development of an antenna.

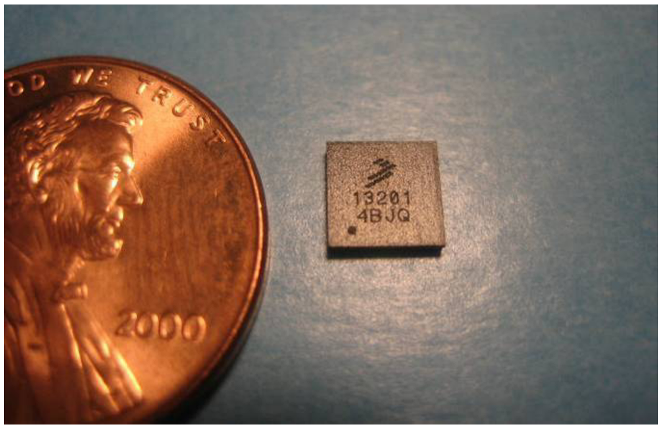


Figure 2.3 – Freescale MC13201 Transceiver

If space is available, there are several manufacturers that provide systems that include simple transceiver microcontroller combinations. These options are power hungry compared to the single transceiver options, demanding as much as 100mA to operate versus the Freescale component's mere 30mA. The benefits of the transceiver/microcontroller module come with the simplicity of integration and operation. These modules include all required hardware, so often startup and communication is simply a matter of supplying the correct voltage and connecting to a UART or PC serial port. Some modules such as the MaxStream XBee (**Figure 2.4**) even include the necessary antenna.

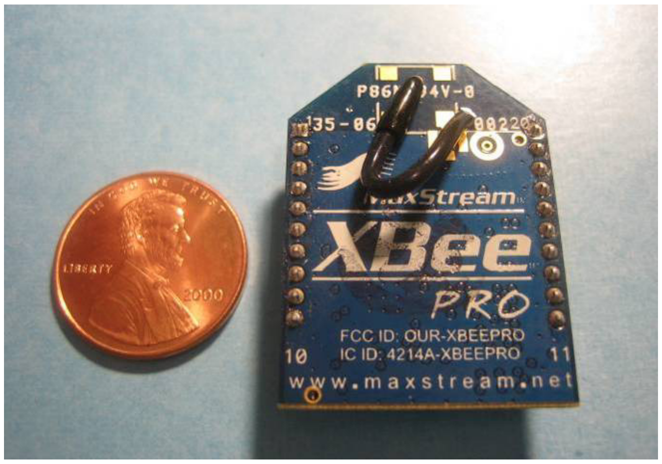


Figure 2.4 – MaxStream XBee-Pro Module

Ultimately all of the options are ZigBee/802.15.4 compliant so selection of the correct module involves finding a balance between ease of integration, cost, weight, and size.

2.4.3 Antenna Design

With many of the available ZigBee options, the choice of antenna is left to the designer. There are three commonly discussed methods of interfacing antennas with the modules. The first option often described is to fabricate an antenna as part of the components PCB board. This option requires the least number of additional components. To do it correctly, however, requires an advanced understanding and familiarity with antenna design as well as stringent controls over the PCB fabrication processes and materials to ensure consistency.

Another option available is to use a chip style antenna. These antennas are professionally designed and fabricated for particular operating frequencies. A typical 2.4 GHz chip antenna is not much larger than a grain of rice and with the correct filtering could produce good results. However, like PCB fabricated antennas, this option requires strict design consideration for component placement and board properties.

The final major option is to utilize some form of wire or external antenna. This option can be as simple as a flexible wire, as is included with certain XBee modules. Other options, as seen with the Integration ZigBee module, require adaptors and connectors to make the connection to a standard antenna with a Subminiature Version A (SMA) connector or similar antenna.

2.5 Summary

In this chapter the hardware design aspects of developing a network processing node for light UAVs were described. The considerations for choosing the microcontroller and peripherals were addressed, followed by an overview of the options for selecting, and the features of, the main wired and wireless communications bus. The next chapter explores the decisions and development of the software required to implement the system functionality.

3 Software Design

3.1 Introduction

This chapter explores the design and development of the software layers of the light UAV system and the Tiny Interface Module. The first section covers the use of preprocessor directives and generic initialization functions to streamline user integration into the module. Section 3.3 describes the design and functionality of the custom CAN networking driver. The next section introduces the custom wireless networking driver design and development. In Section 3.5, the custom operating system (OS) designed to be implemented on the module, is surveyed. The chapter concludes with a summary and comments section.

3.2 Generic Initialization

In order to fulfill the design objective of making the Tiny Interface Module quickly adaptable by students, it was necessary to eliminate the need for each module's base functionality to be individually designed. This was done by standardizing a base set of pin configurations and by creating generic initialization functions to allow user configuration of key components. The base pin configuration, seen in **Figure 3.1**, ensures that the listed features are always available and connected to the same module pin. In the case of the UART level shifting, keeping the same output pin is crucial.

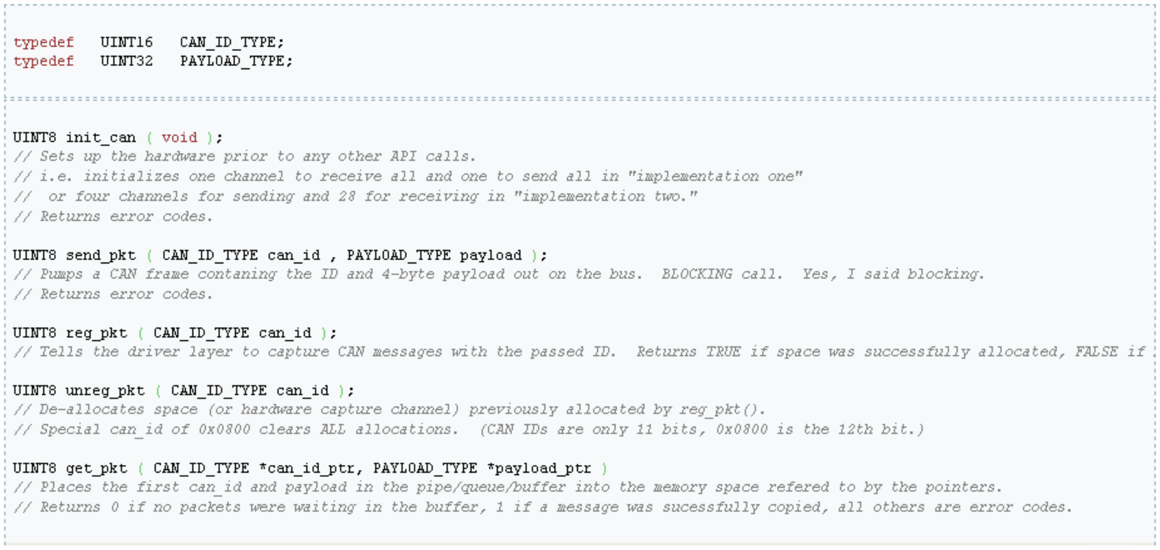
```
    // P0.0  -  TX0  (UART0), Open-Drain, Digital
    // P0.1  -  RX0  (UART0), Open-Drain, Digital
    // P0.2  -  SCK   (SPI0), Open-Drain, Digital
    // P0.3  -  MISO  (SPI0), Open-Drain, Digital
    // P0.4  -  MOSI  (SPI0), Open-Drain, Digital
    // P0.5  -  NSS   (SPI0), Open-Drain, Digital
    // P0.6  -  SDA  (SMBus), Open-Drain, Digital
    // P0.7  -  SCL  (SMBus), Open-Drain, Digital

    // P1.0  -  TX1  (UART1), Open-Drain, Digital
    // P1.1  -  RX1  (UART1), Open-Drain, Digital
```

Figure 3.1 – Default Pin Configuration.

The generic initialization functions provide an equally important role by providing the user with the ability to change key options without adjusting the function coding. In some cases, the changes may simply be a baud rate or clock rate; in other cases, it may involve configuring a block of inputs or outputs. The generic initialization functions include configuring UART 0 and 1, CAN communications, wireless communications, the programmable counter array (PCA), general purpose input/output (GPIO) and the system clock. Development of the PCA and GPIO were included as part of the IDEAnix OS developed by Darren J. Brown through the University of Kentucky. The CAN communications functions were the focus of the research performed by Nithyananda S. Jeganathan. The UART and system clock functions were developed by Darren J. Brown and modified to be included as part of the wireless driver. The modification was in response to a concurrent need and slightly different design requirements. Finally, development of the wireless communications functions was part of the wireless daughter board implementation of this project.

3.3 CAN Driver

Development of a CAN driver was performed by Nithyananda S. Jeganathan at the University of Kentucky as part of his graduate research project. The communications specification used to guide the development went through five major revisions. Through this progression the specification developed from one set of seven functions into a Driver API and a User API. The User API will be not addressed because it functions exclusively between the OS and the user. The Driver API, however, describes the interaction between the CAN and the OS. The other key design decisions were that the driver would only support 11-bit identifiers and would contain a standardized four byte data packet that can be padded with zeros if necessary. These simplifications are intended to enable the network to be quickly implemented and easily adapted by students. Optimization and handling of 29-bit identifiers is targeted for future development. The basic structure of the driver follows the Application Programming Interface (API) seen in **Figure 3.2**.



```
typedef   UINT16   CAN_ID_TYPE;
typedef   UINT32   PAYLOAD_TYPE;

UINT8 init_can ( void );
// Sets up the hardware prior to any other API calls.
// i.e. initializes one channel to receive all and one to send all in "implementation one"
//  or four channels for sending and 28 for receiving in "implementation two."
// Returns error codes.

UINT8 send_pkt ( CAN_ID_TYPE can_id , PAYLOAD_TYPE payload );
// Pumps a CAN frame contaning the ID and 4-byte payload out on the bus.  BLOCKING call.  Yes, I said blocking.
// Returns error codes.

UINT8 reg_pkt ( CAN_ID_TYPE can_id );
// Tells the driver layer to capture CAN messages with the passed ID.  Returns TRUE if space was successfully allocated, FALSE if

UINT8 unreg_pkt ( CAN_ID_TYPE can_id );
// De-allocates space (or hardware capture channel) previously allocated by reg_pkt().
// Special can_id of 0x0800 clears ALL allocations.  (CAN IDs are only 11 bits, 0x0800 is the 12th bit.)

UINT8 get_pkt ( CAN_ID_TYPE *can_id_ptr, PAYLOAD_TYPE *payload_ptr )
// Places the first can_id and payload in the pipe/queue/buffer into the memory space refered to by the pointers.
// Returns 0 if no packets were waiting in the buffer, 1 if a message was sucessfully copied, all others are error codes.
```

Figure 3.2 – CAN Driver API

The **init_can** function is a general purpose initialization function used to initially configure the CAN hardware. The **send_pkt** function sends the CAN frame across the bus of the passed variable values **can_id** and **payload**. **Reg_pkt** registers to receive frames with the CAN packet ID of **can_id**. **Unreg_pkt** removes the registration for receiving a packet ID of **can_id**, therefore the packet will be ignored. Finally, **get_pkt** is used to return the current buffered data packet using the pointers **can_id_ptr** and **payload_ptr**. This last function is only implemented for non-IDEAnix applications. IDEAnix uses an included custom function to pass data up to tasks running in the OS space.

3.4 Wireless Driver

Like the CAN driver, development of a comprehensive wireless driver could constitute and independent research project. Because this was incorporated as one aspect of a larger development, a base level of functionality with minimal code overhead and design time was developed. Using this approach, research into different network design theories as well as ways to utilize included XBee, and custom design, application programming interfaces (APIs) was conducted. The end result is a wireless driver providing the required fundamental OS API functionality. With user exposure to this functionality future research will be dedicated to developing a more feature rich implementation.

3.4.1 Wireless Network Design Considerations

Implementation of a network requires careful consideration of a number of key factors. These factors are all based around ensuring that the data sent is received as intended. The first obstacle to be overcome to make sure data is able to reach the intended destination. The three network topologies that are utilized by ZigBee are star, mesh and cluster tree. Seen in **Figure 3.3**, the star topology relies on a single centralized coordinator node to receive and relay all communications. A cluster tree has groupings, clusters, of nodes that utilize a single coordinator node per group to communicate with the network's main coordinator. The final network topology is a mesh, which utilizes many nodes equally to share the burden of relaying data from one point to another.

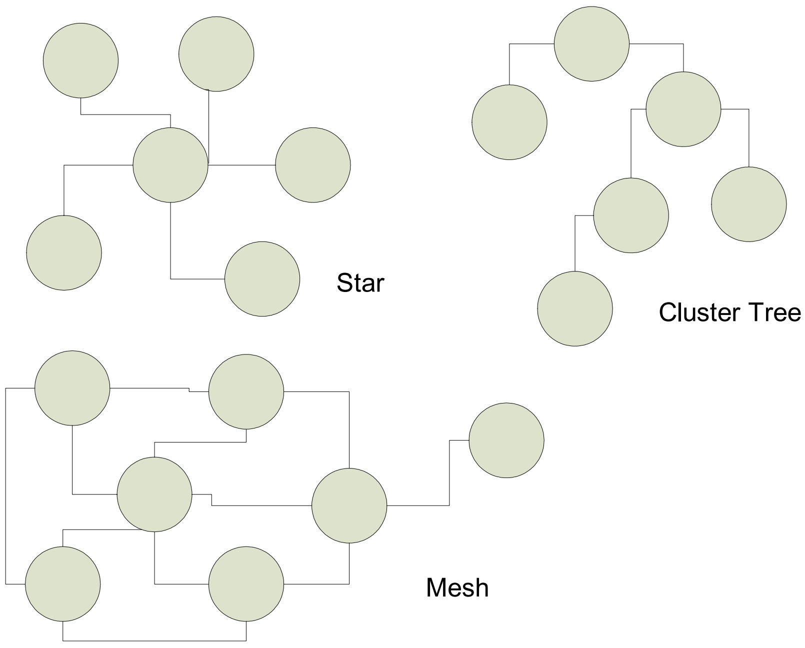


Figure 3.3 - ZigBee Network Topologies

XBee-Pro module utilizes point-to-point, point-to-multipoint, and peer-to-peer topologies. Point-to-point can be seen as a simplified form of a star network, where there is just direct communication between two nodes. Point-to-multipoint functions as the star network described above. Finally, peer-to-peer provides functionality similar to that of a mesh network, with all nodes able to communicate directly with others.

Once a network topology is chosen, the second step is to ensure that the data received is correct. This is done in two forms, error checking of the received data and ensuring that no malicious data is accepted. There are many current techniques and algorithms available to check data for errors, including Checksums, Hamming Codes and Cyclic Redundancy Checks (CRC) [**36**], all with inherent strengths and weaknesses. Checksums are very commonly used in conjunction with CRCs, as a second level of protection in data communication systems such as IP, TCP, and UDP [**37**]. The CRC ensures data transmission integrity while the checksum verifies that data was not corrupted by routers or the transmitting or receiving hosts. The XBee-Pro module includes a similar checksum CRC combination. As part of the 802.15.4 specification a 16-bit CRC is included as part of the MAC layer [**32**]. Utilizing the checksum is an option that can be incorporated into the XBee module API communication frames discussed in the next section.

System security is an area that could be easily overlooked in development of networks for light UAVs. With the development of the wired communications bus, the system is isolated and therefore has no need to include measures to prevent a malicious user from interfacing with the system. Wireless systems do not afford this luxury, since all communications are transported over an open medium: air. Of the competitive technologies to ZigBee, Bluetooth is the only other technology to provide a complete approach to security **[38]**. ZigBee uses a multi-level design, providing the advanced encryption standard (AES) as part of the IEEE - 802.15.4 standard, and then providing additional security key generation within the ZigBee standard.

The final concern when developing a wireless network is anticipating the effects of using the crowded industrial, scientific and medical (ISM) radio band. The ISM band is intended for use by devices, industrial, scientific, medical or domestic, that "generate and use locally RF energy" **[39]**. Devices within this range are regulated by Federal Communications Commission (FCC) and are required to be "constructed in accordance with good engineering practice." Despite regulations, there is no way to ensure that the communication bandwidth will be available. In tests to study the effects of the closely related IEEE - 802.11 and Bluetooth, the possibility exists for physical layer timeouts and increased packet error rates **[40]**. This interference unfortunately is something that cannot be avoided while using wireless technology. Through testing and good system design, the goal is that these effects can be minimized.

3.4.2 XBee Operation Modes

The MaxStream XBee module discussed in earlier sections is designed to be easily integrated into many applications. To accommodate this, the module is preconfigured to function out of the box by simply connecting the respective UART transmit and receive lines as well as power and ground connections. As a result the default software configuration is very flexible. The XBee module is initially configured in what is referred to as Transparent Mode **[34]**. Transparent Mode instructs the module to operate as a "serial line replacement", sending and receiving all data from the respective data pins. This is performed using a combination of clear channel assessment, to ensure that no other node is currently broadcasting, and a 100 character buffer.

The other mode of operation is referred to as API operation. This frame-based API allows for greater interaction with the modules networking capabilities from within the application code. This mode has an extensive set of frames to allow for module configuration and status messages to be exchanged. These frames also contain a checksum byte to ensure that the frame data is correct.

The XBee API has advantages over alternatives, including integration of the traditional modem AT commands in to the frame structure, to program the module, and provisions of status feedback. It does, however, add a great deal of complexity to the code development by adding handshaking between the XBee module and the microcontroller. Because a broadcast network is desired, much of the XBee API functionality must be disabled. Therefore the system ultimately relies on the custom data frame discussed is the next section.

3.4.3 Custom Wireless API

Development of a wireless driver for the Tiny Interface Module requires the creation of a generic set of functions that can be utilized both individually by a user and by integration into the IDEAnix operating system being developed in parallel. To facilitate integration, the wireless driver's functions would use the same nomenclature and variable types that were developed for the CAN driver. The CAN driver API can be seen in **Figure 3.2** of **Section 3.3**.

The wireless network utilizes a peer-to-peer topology, with lost or corrupt data packets being ignored. Like with the CAN driver, the wireless driver consists of an initialization function, send and get packet functions and register and un-register functions. An additional function, request packet, was created to aid in the code design and testing and is available to users.

The basic flow of wireless communications can be seen in **Figure 3.4**. Interactions with the XBee network occurs by either a UART character transmit or receive. Incoming characters are handled by the UART 1 Interrupt Service Routine (ISR), which is triggered with each incoming or outgoing character. The ISR first checks if the data is incoming or outgoing. If it is incoming the character is compared against a counter to see if a complete frame has arrived. If a complete frame has not yet arrived, the character is added to a buffer and the ISR clears the flag to wait for the next character. If the frame is complete, the ID is checked against the list of registered IDs. If there is a match, the frame is copied to a user accessible buffer to be accessed using the get packet function or pushed to the OS buffers using IDEAnix functions. Otherwise, the data is dumped.

The process of handling outgoing data is much simpler. The OS or user calls the send packet function passing an ID of type unsigned integer and a payload of type unsigned long. These values are then placed in a frame with start and stop delimiters and transmitted across the UART1 transmit line.

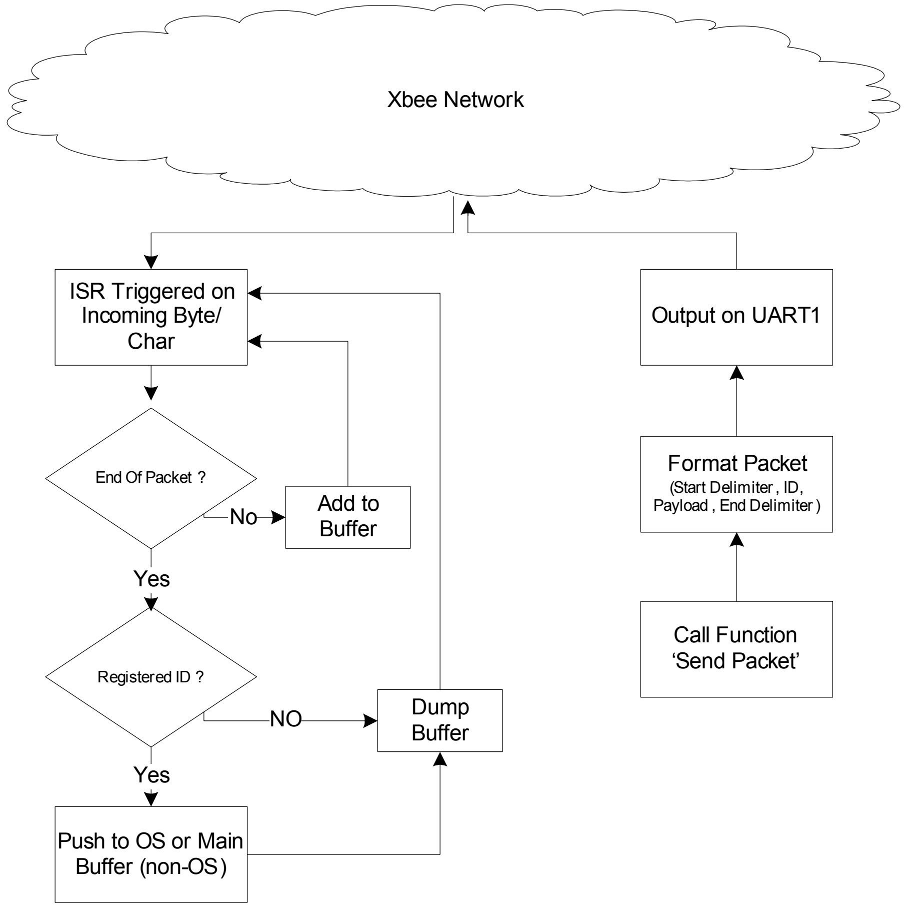

Xbee Network
ISR Triggered on Incoming Byte/ Char
End Of Packet ?
No
Add to Buffer
Yes
Registered ID ?
NO
Dump Buffer
Yes
Push to OS or Main Buffer (non-OS)
Output on UART1
Format Packet (Start Delimiter, ID, Payload, End Delimiter)
Call Function 'Send Packet'

Figure 3.4 – Wireless Driver Input\Output Block Diagram

The register and un-register functions are straightforward. The OS calls either function passing the respective ID as a parameter. Then in both cases, the ID is compared against the current ID array. In the case of registering an ID, if the ID is not found a free slot is located and the ID recorded. With un-registering IDs the ID, if found, is replaced with a placeholder value representing an open position. In both the register and un-register functions if the desired condition is not met an error is returned.

If any of the send, receive, register or un-register functions encounter unexpected errors or are unable to complete an action, an error code is returned. The set of error codes seen in
Table 3.1 were developed to accurately convey the error and to allow a system to respond accordingly.

Error Code Name	Value	Description
WRLS_DRIVER_NOERROR	1	Function completed successfully
WRLS_DRIVER_ERROR	200	Un-explained error returned
WRLS_MSGOBJS_FULL	201	Send data error (future use)
WRLS_MSGID_NOTFOUND	202	Requested ID was not found
WRLS_MSGDATA_NOTFOUND	203	Requested Data was not found
WRLS_DUPLICATE_ID_ERR	204	Registering ID was already found
WRLS_UNREG_ID_ERR	205	Un-registering ID was not found
WRLS_MSGID_FULL	206	Maximum Number of IDs registered

Table 3.1 – Function Error Codes

3.5 IDEAnix

IDEAnix was developed by Darren J. Brown at the University of Kentucky as a build of the embedded operating system MicroC/OS-II ported to the 8051 hardware architecture. This build provides initialization functions required for the Silicon Labs C8051F04x series microcontrollers as well as a message routing layer (MeRL). The inspiration behind the development of IDEAnix was the necessity to provide code independence and modularity. In a task driven operating system, as is provided by the MicroC/OS-II core, user tasks are not aware of other tasks. This feature, in combination with the message routing capabilities of MeRL, provides a platform that can be fully reconfigurable and utilized successfully within a fault tolerant system.

MeRL, seen in **Figure 3.5**, functions by utilizing ID values to filter and broadcast data across the supplied network, as well as to distribute the data internally. The network type is not specified by either IDEAnix or MeRL. The only requirement is that the network driver follows the same Driver API as was specified for CAN.

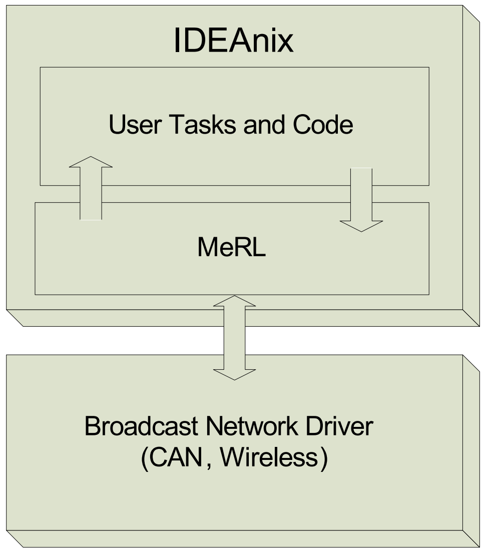


Figure 3.5 – Network Software Framework

3.6 Summary

This chapter covered the software designed to support the Tiny Interface Module. The major components are the CAN and wireless drivers as well as the custom OS build, IDEAnix. Chapter four will explore the implementation of the module.

4 Implementation

4.1 Introduction

This chapter discusses the specifics of the components derived from Sections 2 and 3. Presentation of the components follows an order similar to that of the original descriptions. The chapter starts with the hardware, processor, communications, power, headers/connectors and other hardware of note. The next section gives detailed specifications for the wireless hardware. Third, the chapter explains the external contributions, as well as the contributions made as part of this project, to the software design. The fourth section gives an overview into the various stages and different techniques utilized for prototype fabrication. The chapter concludes with a summary and comments.

4.2 Overview of Design

As addressed in previous sections, the Tiny Interface Module was designed to provide the base hardware layer for a light UAV system bus. As seen in **Figure 4.1**, the final system consists of a main board providing almost all of the above described functionality with the wireless capabilities being supplied by the optional wireless daughterboard.

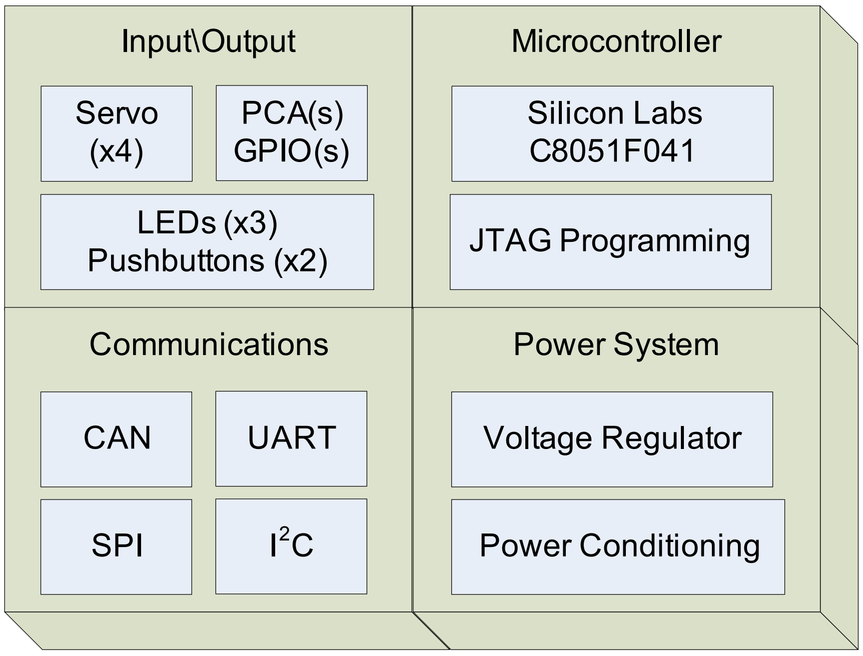


Figure 4.1 – Tiny Interface Module Hardware Block Diagram

4.3 CAN Module Hardware

This section discusses the specific components and hardware selected for implementation of the Tiny Interface Module. **Table 4.1** lists the Tiny Interface Module's components, estimated unit costs and total costs. The pricing information was collected from Digikey Corporation, except for the PCB board cost which is quoted from PCBExpress for a batch size of 15 modules.

TIMCAN	P/N	Number	Unit Price	Total
13 Pin Dual Header	929665-02-13-ND	1.0	3.1140	3.1140
19 Pin Single Header	929647-09-19-ND	1.0	2.0310	2.0310
MCU	336-1206-ND	1.0	15.5700	15.5700
Switch	401-1427-1-ND	2.0	0.4170	0.8340
500mA Voltage Regulator	TPS77633D	1.0	1.7500	1.7500
LED (green)	L62505CT	1.0	0.2160	0.2160
LED (red)	L62501CT-ND	1.0	0.2220	0.2220
LED (yellow)	L62507CT	1.0	0.2490	0.2490
CAN transceiver	296-11654-1	1.0	3.0400	3.0400
1.0u Tantalum CAP	511-1467-1-ND	16.0	0.2470	3.9520
10.0u Tantalum CAP	511-1473-1-ND	1.0	0.2830	0.2830
.1u Ceramic CAP	PCC1812CT	6.0	0.0740	0.4440
56 Resistor	P56ACT-ND	3.0	0.0414	0.1242
120 Resistor	P120ACT-ND	1.0	0.0770	0.0770
1K Resistor	P1.0KACT-ND	7.0	0.0414	0.2898
10K Resistor	P10KACT-ND	2.0	0.0770	0.1540
Level Shifter	MAX3233ECWP+G36	1.0	8.8000	8.8000
PCB Manufacture	PCBExpress	1.0	20.2000	20.2000
			Total	**$61.35**

Table 4.1 – Parts List and Component Pricing for the Tiny Interface Module (12/4/2006)

4.3.1 C8051F041 Microcontroller

As discussed in Section 2.2.1 the Silicon Laboratories C8051F04x product line was chosen as the processing base for the Tiny Interface Module. This product line consists of eight versions all with slightly different functionality or form-factor. Each version features the same core components, including a 25 mega-instructions per second 8051 core, flash, CAN, 2 UARTs, I^{2}C, SPI, programmable counter arrays and general purpose input/output. Product numbers ending in an even number feature a 100-pin thin quad flat pack (TQFP) package; odd numbers represent a 64-pin TQFP package **[20]**. Since board space is a premium, and the number of inputs/outputs is limited, the 100-pin package was not necessary. Therefore, the 64-pin C8051F041 processor, shown in **Figure 4.2**, was chosen from the Silicon Laboratories product line. The key feature of the 041 is the 12-bit analog-to-digital converter that provides greater resolution for future applications.

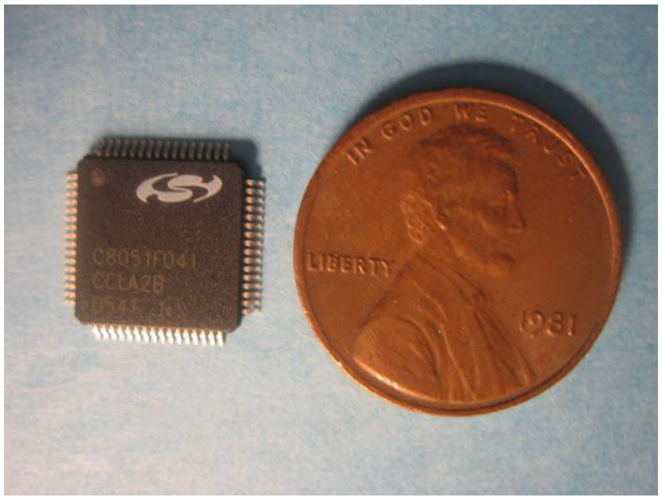


Figure 4.2 – Silicon Labs C8051F041 Microcontroller

4.3.2 Communications

The Tiny Interface Module features four main forms of wired communication including, Controller Area Network (CAN), Universal Asynchronous Receiver/Transmitter (UART), I^{2}C compatible System Management Bus (SMBus), and Serial Peripheral Interface Bus (SPI). Both the SMBus and SPI communications can connect directly to the C8051F041 processor. In order to have CAN communications or be able to communicate with the UART via a computer or other RS-232 device, transceivers are necessary.

The communication implementation decisions for development of the Tiny Interface Module hardware involved the selection and implementation of Controller Area Network (CAN) and RS-232 transceivers. The CAN transceiver, seen in **Figure 4.3**, is from Texas Instruments. The SN65HVD230 is a 3.3 volt part so it can function off the same voltage supple lines as the microcontroller. This part also features a low current standby mode of 370 micro amps. To ensure that a faulty node does not disturb the rest of the bus this part also features an open-circuit fail safe and “glitch-free power-up and power-down” **[41]**.

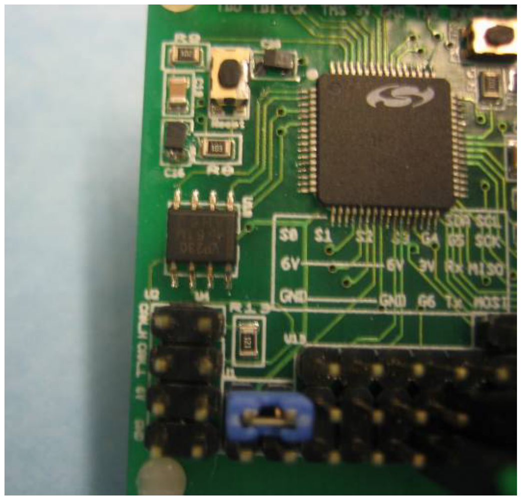


Figure 4.3 – CAN Transceiver

Research into RS-232 level shifters resulted in the selection of the Maxim MAX3233ECWP+G36, **Figure 4.4**. This dual RS-232 Transceiver runs off a single 3.3 volt supply, meets EIA/TIA-232 specifications, can transmit at up to 250kbps and has an auto shutdown feature for power savings **[42]**. This functionality is provided in a wide SOIC with internal capacitors to save board space and complexity

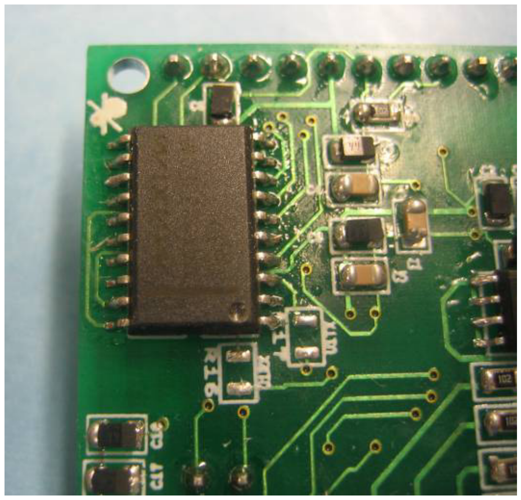


Figure 4.4 – Maxim Dual RS-232 Transceiver

Also seen in **Figure 4.4** are two unpopulated resistors (R16, R17) these components serve as jumpers enabling and disabling the level shifting of UART1. This feature was added so that UART1 could also be accessed directly by components such as the XBee module. If the user desires the level shifter UART1 resistors R16 and R17 must be populated with 0 ohm short circuits.

The final two provided communications protocols, I^{2}C and SPI, do not need a transceiver or other enabling hardware. The microcontroller pins are therefore routed to pin headers that will be discussed in Section 4.3.4.

4.3.3 Power

The power system design for the Tiny Interface Module has three main parts: the voltage regulator to provide a voltage level usable by the microcontroller, pull-up resistors for interfacing with the target application, servos, and capacitors to condition the power supplied to the module and its peripherals. The Tiny Interface Module utilized two models of voltage regulators during the prototyping stage. The first regulator, a Texas Instruments TPS76533, featured a low quiescent current of 35 micro amps while sourcing 3.3 volts at 150 milliamps **[43]**. This part could handle input voltages of up to 13.5 volts. The second voltage regulator, chosen due to the increased current demands of the wireless daughter module, was also from Texas Instruments. The TPS77533 has features similar to that of the original voltage regulator. The main difference is its ability to source 500 milliamps while still maintaining a low quiescent current of 85 micro amps **[44]**. The final regulator is seen in **Figure 4.5**.

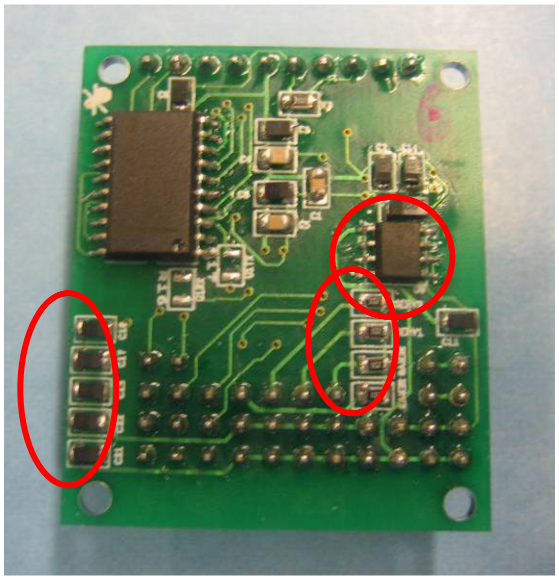


Figure 4.5 – Capacitors, Pull-up Resistors, and Voltage Regulator

The second part of the power system design was enabling direct connection of R/C servos to the board. This required the use of four pull-up resistors, shown in **Figure 4.5**. Since most servos for R/C places are six volt parts, it was necessary to provide a way to ‘pull-up’ the 3.3 volt microcontroller output to the necessary six volts. By predefining the servo pins as open-drain within the microcontroller, the appropriate pins float to six volts for digital high or are pulled to ground for digital low outputs.

The final objective was to condition the power entering the board and key components. As discussed in section 2.2.3 the use of tantalum and ceramic capacitors in the system were intended to aid in the availability of consistent and stable power. In addition to the added capacitors, which were dictated by the component specifications, one 1 micro farad tantalum and one .1 micro farad ceramic capacitor were added to each voltage input into the microcontroller. Also, a series of four tantalum capacitors were added on the servo power supply lines as seen in **Figure 4.5**. These capacitors, currently 1 micro farad each, could be increased in value to provide more consistent power, if later deemed necessary, on the extremely noisy six volt servo power lines.

4.3.4 Headers/ Connectors

As previously stated, one of the main design features of the Tiny Interface Module is its ability to be easily integrated into R/C aircraft. This is largely facilitated through the careful selection of headers and connectors. Seen in **Figure 4.6** the header across the top of the board provides JTAG programming and two level shifted UARTs. The 8-pin header at the bottom left of the board provides two 4-pin CAN connections; each 4-pin connector having a CAN-high, CAN-low, six volt power, and ground connection.

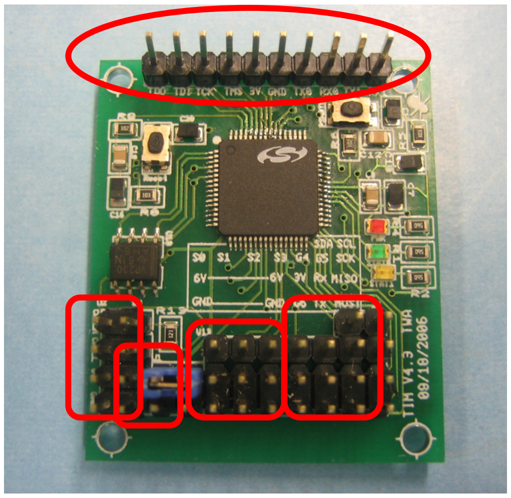


Figure 4.6 – Module Headers

The next 4-pin block provides the ability to jumper or isolate the six volt CAN power line from the six volt lines connecting to the servo headers. Isolating the two power buses gives the ability to utilize a separate battery or other power supply for module powering and servo control. This is needed if the servos require a different voltage than comes across the 4-pin CAN connector or if the servos pull the six volt line below the tolerances of the module's voltage regulator.

Immediately to the right of the 4-pin block are the four 3-pin servo headers. These headers use the same spacing and pin order of most R/C servo connectors. The top pin is the data line from the microcontroller. The middle pin is the six volt line. Finally, the bottom pin is the ground connection. The remaining pins serve as three general purpose input/output lines (GPIO), a three volt power supply line for various applications, a 2-pin non-level shifted UART communications, a 3-pin SPI and a 2-pin I^2C.

4.3.5 Other Hardware

Along with the required functionality already discussed the Tiny Interface Module also includes two other features for user convenience. First, the module has three status Light Emitting Diodes (LEDs), **Figure 4.7**. The first LED at the top is tied between the 3.3 volt power and ground to show that the board has power. The other two LEDs are tied to GPIO pins on the microcontroller for user defined functionality.

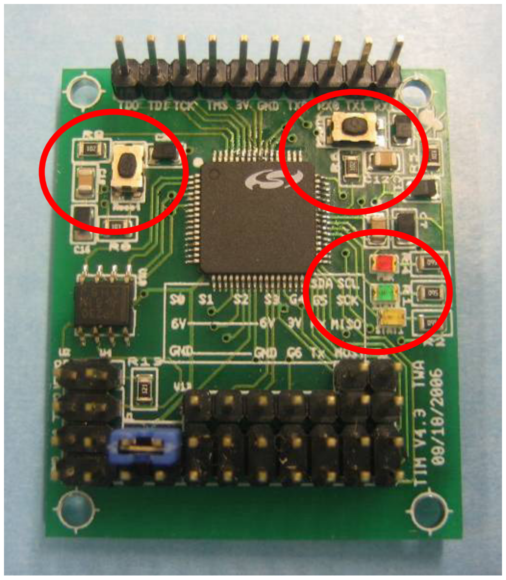


Figure 4.7 – Reset Button and Power and Status LEDs

The second convenience feature implemented on the Tiny Interface Module was the addition of two push-buttons. One push-button (**Figure 4.7**) acts as a soft reset for the microcontroller. The other push-button is tied to a pin on the microcontroller to be used as a user defined input.

4.4 Wireless Hardware

Development of the Tiny Interface Module wireless hardware, like all other module components, went through several revisions. The original concept was to design and fabricate a module fundamentally identical to the CAN based module. The goal was to find a simple wireless solution that would allow the CAN transceiver to be exchanged for a wireless transceiver. Due to design decisions discussed in Chapter 2 however, this solution was not feasible.

Instead a custom daughterboard was developed to accompany the CAN based module. The module that was chosen to integrate into the daughter board was the MaxStream XBee-Pro 802.15.4/ZigBee module **[34]**. A version of this module that operated at 900MHz was selected to avoid unnecessary interference with the Piccolo Autopilot's 2.4GHz radio communications. The daughterboard went through several revisions as seen in **Figure 4.8**. After much investigation and trial and error, the simplest solution proved to be the best option.

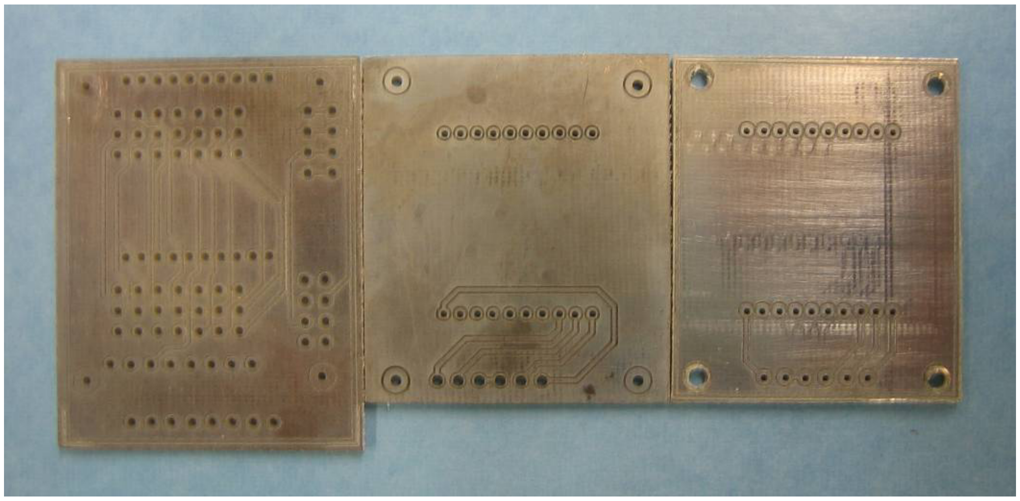


Figure 4.8 – Wireless Daughterboard Evolution

4.4.1 Daughter Board

The development of the daughterboard hardware involved providing stable electrical and physical connection for the XBee module. This process consisted of interfacing four pins from the CAN based module, 3.3 volt power, ground, UART transmit, and UART receive, to the corresponding pins on the XBee wireless module. The PCB was approximately the same dimensions as the CAN based module allowing the CAN based modules' mounting holes to be used to physically secure the daughterboard. This simple design had two major benefits. First, this provided good connection between the microcontroller and the XBee module with a minimum of wiring and PCB complexity. Secondly, due to the simple design, the board could be completely designed, fabricated and populated in-house, a benefit further elaborated in section 4.6.

The final design is seen in **Figure 4.9**. It can be seen that the copper traces and solder connections are isolated to one side of the PCB, eliminating the need for through-hole plating. Jumper wires are used to electrically connect the daughterboard to the CAN based module. This board can then be mounted to a Tiny Interface Module using simple nylon bolts and spacers.

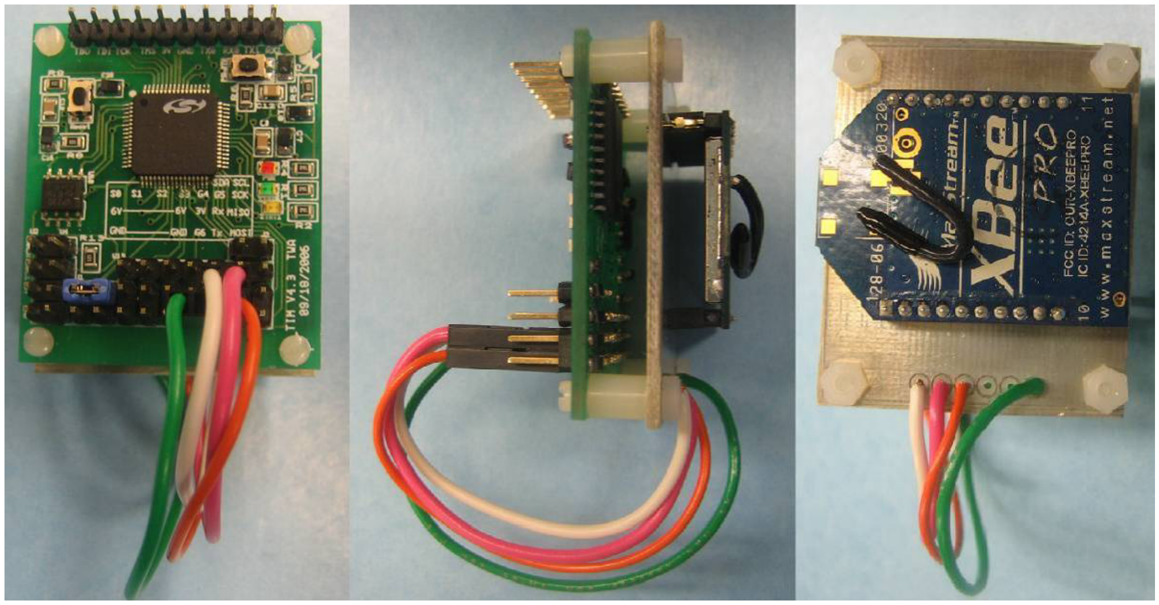


Figure 4.9 – Daughter Board Mounted on Module

4.4.2 Module Interface

The XBee module plugs into a one millimeter spacing socket on the daughter board. Though the XBee module has many optional pins for future expansion, the basic necessary pins are the 3.3 volt power, ground, and UART transmit and receive. These pins are connected to the large straight-pin header on the CAN based module. This header has the necessary 3.3 volt, ground, and UART connections. Care must be taken to ensure that the Tiny Interface Module has not been configured to level shift the UART1 lines. This is done by un-populating the two 0 ohm resistors identified by the silk screen as R16 and R17. If this is done correctly, the module can be communicated with and configured using basic AT commands.

4.5 Software

Though the Tiny Interface Module was designed to be flexible, allowing it to adapt to any light UAV application, basic hardware initialization and a wireless driver were completed to facilitate the module's operation. By providing this functionality as part of the Tiny Interface Module's development it allowed the module to be utilized, and tested, independent of the design status of IDEAnix. In parallel projects software design, of IDEAnix and the CAN driver, was completed by Darren Brown and Nithyananda S. Jeganathan as part of their respective research at the University of Kentucky. A simple set of system initializations was needed for the cases without operating systems in addition to developing a driver for implementing wireless functionality. The code as implemented and tested on the module is described in chapter 3.

4.6 Fabrication

Basic module fabrication consists of three major steps: board computer aided design (CAD), PCB prototyping and PCB population. Section 2.2.5 described the hardware versions and revisions through which the Tiny Interface Module progressed. While the device features were selected and revised, there was also constant revision to the physical board design. As seen in **Figure 4.10,** the Tiny Interface Module PCB board continually increased in component density throughout the progression. The final populated module measured approximately 1 5/8 inches by 1 3/8 inches by 1/2 inch. This was possible due to improvements in both board manufacturing techniques and the optimization and reorganization of component placement.

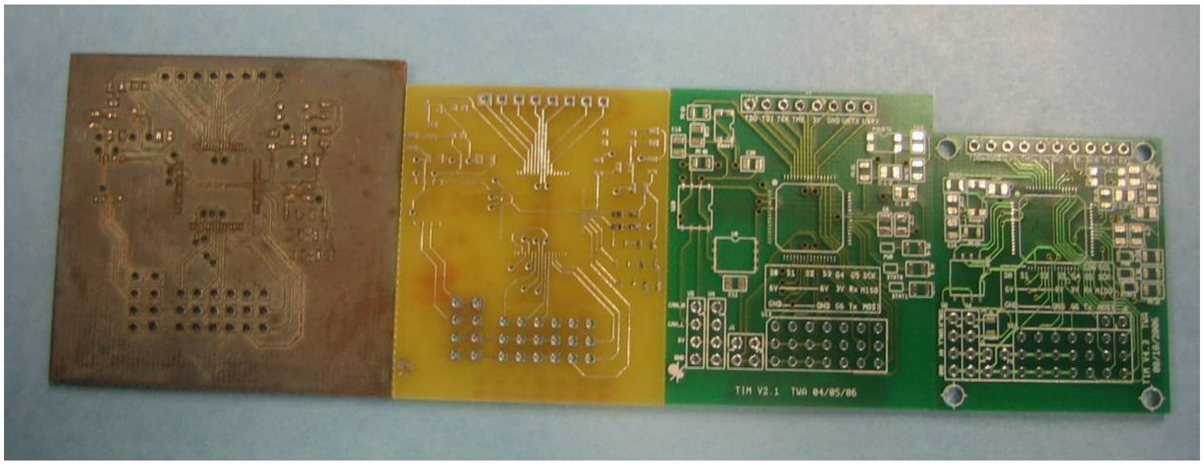
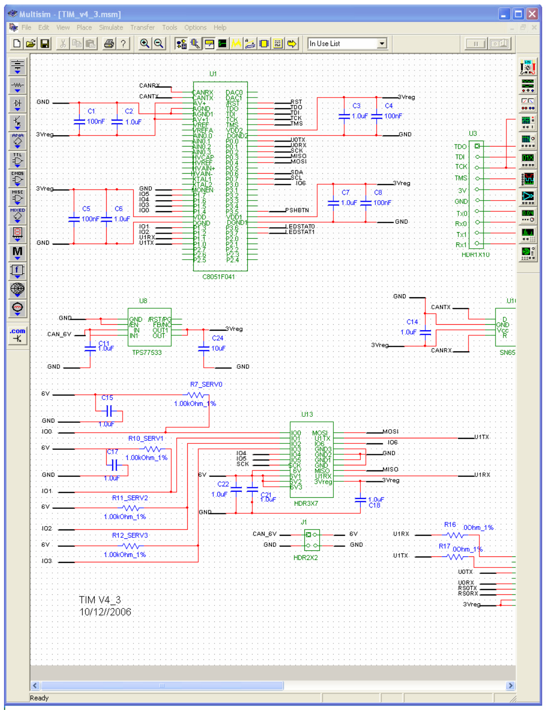


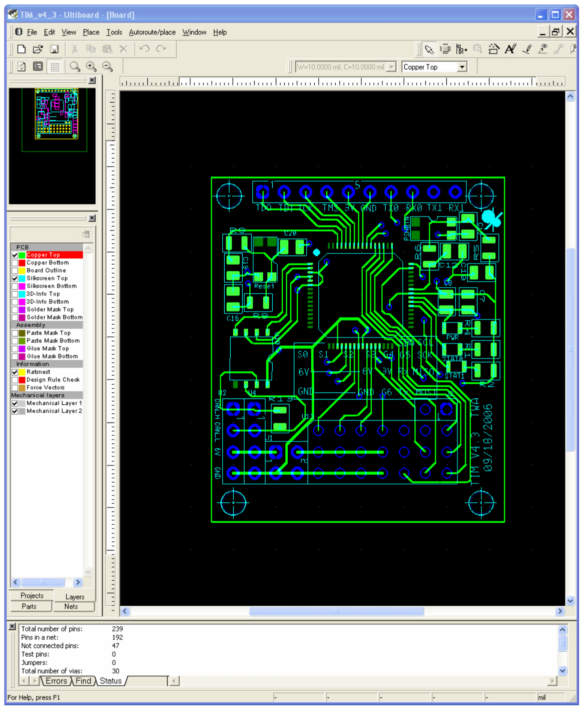
Figure 4.10 – TIMCAN Hardware Board Footprints (Oldest on Left to Newest on Right)

4.6.1 Board CAD

The programs used for development of the Tiny Interface Module were Multisim and Ultiboard. Multisim is a schematic design program. Functioning similar to most schematic programs, Multisim, shown in **Figure 4.11 A**, is used for creation and connection of the various components of the module. These schematics are then exported directly into Ultiboard. Once in Ultiboard, shown in **Figure 4.11 B**, the components are placed and the copper connections between associated pins are made. From Ultiboard, the final design can then be exported as a gerber file to be utilized in the PCB fabrication.



A)



```
Total number of pins:      239
Pins in a net:             192
Not connected pins:        47
Test pins:                 0
Jumpers:                   0
Total number of vias:      30
```

B)

Figure 4.11 – A) Multisim and B) Ultiboard Screen Captures

4.6.2 On-campus PCB Prototyping

During the early stages of the Tiny Interface Module development and final development of the daughter board, on-campus resources were used for PCB prototyping. First, this process included importing the gerber files developed in Ultiboard into the milling machine's custom milling software. A blank PCB board was then placed in the milling machine seen in **Figure 4.12**. Two sided copper PCB boards, meaning the board consisted of a non-conductive layer with a copper on top and on bottom, were used. It was at this point that all through-holes and vias are drilled for two sided boards.

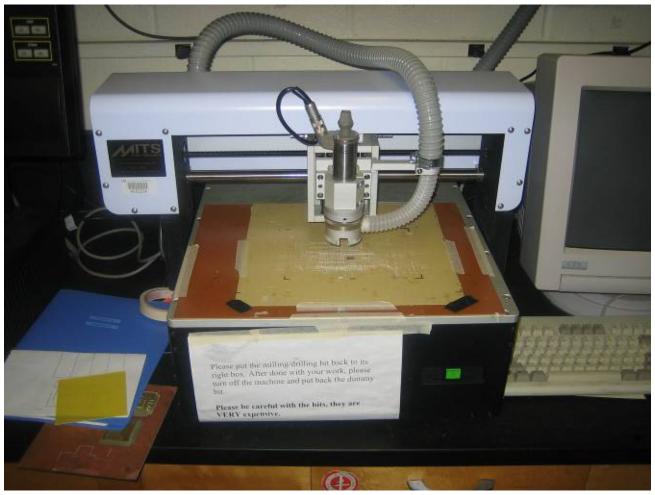

MITS
Please put the milling/drilling bit back to its right box. After done with your work, please turn off the machine and put back the dummy bit.
Please be careful with the bits, they are VERY expensive.

Figure 4.12 – ECE Department Milling Machine

After the drilling is completed, the board is removed from the machine. The holes are then treated with a conductive ink and the board is placed in a plating bath where the combination of current and the conductive ink forms a copper lining within the holes (**Figure 4.13**). The board is then dried and returned to the milling machine where each side's copper traces are formed. This is done by cutting narrow gaps on either side of the desired trace isolating it from the rest of the copper layer.

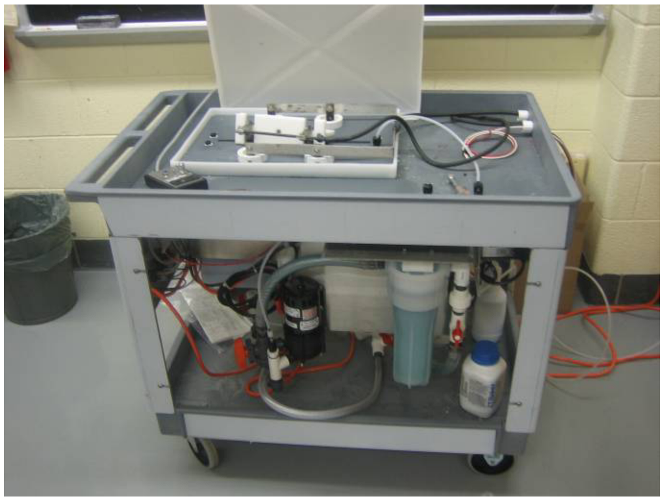


Figure 4.13 – ECE Department Through Hole Plating Bath

The final step of this process is to tin the copper. The tinning process consists of using a tin based chemical solution to treat the board. The solution reacts with the copper, resulting in a surface that is tolerant to oxidation and is easier to work with soldering.

4.6.3 Professional PCB Prototyping

There are a number of PCB prototyping options available through the internet. PCBExpress, now part of Sunstone Circuits, was chosen because it had a history of satisfactory results and provided an Education Sponsorship Program. The process that PCBExpress uses, as described on the website **[45]**, involves drilling board holes, applying thin copper deposits within the holes, and applying a photosensitive dryfilm around the board pads and traces. Next, an electrochemical process is used to build the copper on the desired surfaces, and then the board is tinned. Finally the dryfilm is removed and all exposed copper, not protected by the tin, is etched. If the option is purchased, the board, with exception of the solder pads, has a solder mask applied and then the board is dipped in a tank of solder coating the pads. Finally, the board is labeled using a silk screening process, and the outline is cut.

Though professional board fabrication comes at a premium price both financially and in increased fabrication times, the benefits of material and process consistency are often worth the cost for bulk or complex PCB fabrication. It is worth noting, however, that for a simple board design like the wireless daughterboard, in-house manufacturing easily provides a reliable prototype.

4.6.4 PCB Population

Independent of where the PCB was created, the final stage of fabrication is board population. For the Tiny Interface Module this process was completed using a combination of solder reflow and hand soldering. Solder reflow involves placing a small amount of solder paste on each component pad on a side of the PCB. This is done using a solder paste system like the one seen in **Figure 4.14**. Next the components for this side are placed using a pick and place machine or by hand using tweezers.

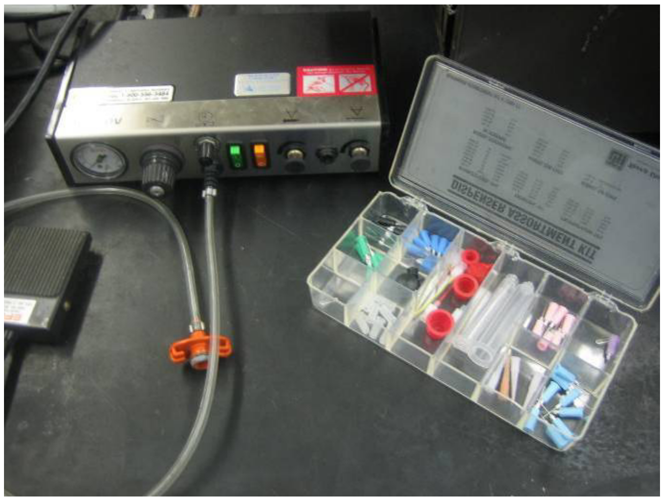


Figure 4.14 – ECE Department Solder Paste System

The board is then placed on the reflow machine. The reflow machine heats and cools the board as it is slid from right to left across plates of increasing, then rapidly decreasing, temperature (**Figure 4.15**). This process allows the solder paste to become a liquid and adhere to both the components and the pads. When the solder is then quickly cooled, the two are bonded physically and electrically with the hardened conductive solder.

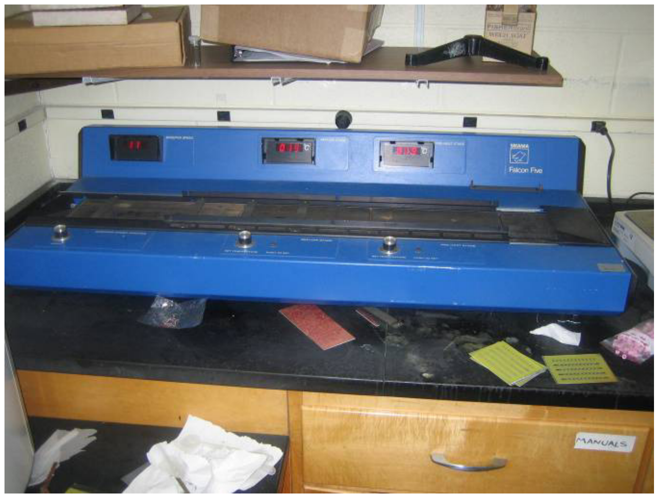


Figure 4.15 – ECE Department Solder Reflow Machine

After verifying that the newly formed solder joints are correct, the next step is to solder by hand the remaining parts. These parts either could not go through the reflow machine, like the headers, or they are on the bottom side of the PCB. This process is done using standard hand soldering equipment such as a soldering iron with a very fine tip, tweezers for holding and placing the small surface mount parts, and a solder wick for cleaning up or correcting poor connections (**Figure 4.16**).

Figure 4.16 – Hand-Solder Station

4.7 Summary

This chapter presented the implementations of the hardware and software aspects of the Tiny Interface Module. The next chapter will present the various types and stages of testing as well as the results.

5 Testing

5.1 Introduction

Previous chapters have discussed the goals and design decisions for development of the Tiny Interface Module. This chapter covers the module's testing process and results. The chapter is concluded with a future work section and a summary.

5.2 Preliminary Testing

Testing of the Tiny Interface Module was completed in two phases: preliminary and application testing. Preliminary testing included assessing the Tiny Interface Module's basic functionality and performance. From this phase of testing errors in board fabrication and population would be caught as well as establishing baseline hardware performance data. It was crucial for the module to pass all of these tests before more complex and sensitive scenarios were pursued.

5.2.1 Bit Flipping, SI Labs Application Example

The basic functionality of the Tiny Interface Module was first testing by utilizing a modified form of a Silicon Laboratories application example included with the C8051F04x development kit **[46]**. The base code was provided by the microcontroller manufacturer as a means to illustrate a working CAN bus. This served as an appropriate test because part of the module's design goal was to provide a standardized coding platform, requiring only minor modifications to utilize existing code libraries. Key alterations to the application note include utilizing an internal timing source, adjusting the CAN timing to use the new timing source, and adding a UART output and respective interrupts. The final key change was to define the module's input/output pins to toggle on a button press and send a message over the UART. The test layout can bee seen in **Figure 5.1**. The CAN bus connects a Silicon Laboratories C8051F040 development board, running the unmodified application note code, to a Tiny Interface Module. The development board could be replaced by another module if desired. The module is connected via RS-232 to a PC for a status display. Finally, a scope probe can be used to monitor the toggling input/output pins to verify operation.

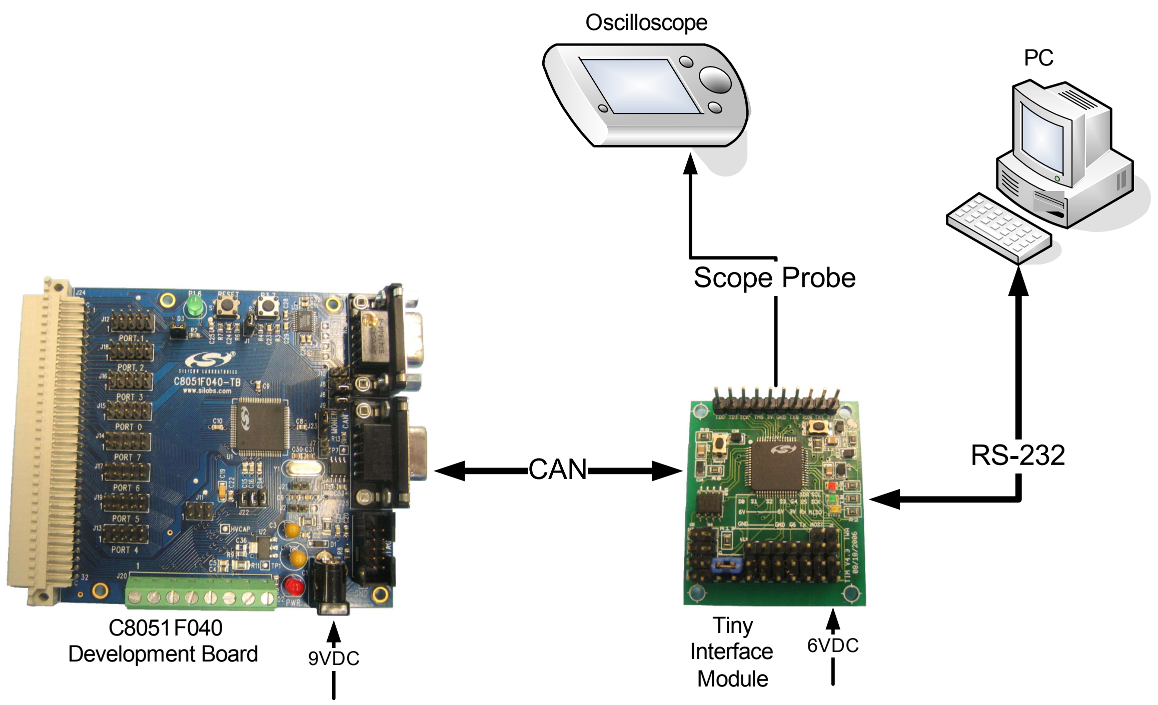


Figure 5.1 – Preliminary Testing Configuration

This test effectively determines if the CAN bus is working, that all solder connections to the input/output pins of the microcontroller are made, and that the RS-232 level shifter is connected and functioning.

5.2.2 Servo/Power Noise Testing

Noise introduce to the Tiny Interface Module by the connection of servos was a concern during the system design. Therefore once the general pin functionality of the module had been tested, it was important to determine a servo's true effect. First, using the PCA of the microcontroller, a servo pulse output was produced. As seen in **Figure 5.2** the output, using the pull-up resistors, was approximately a six volt pulse, between one and two milliseconds in length, every 20 milliseconds.

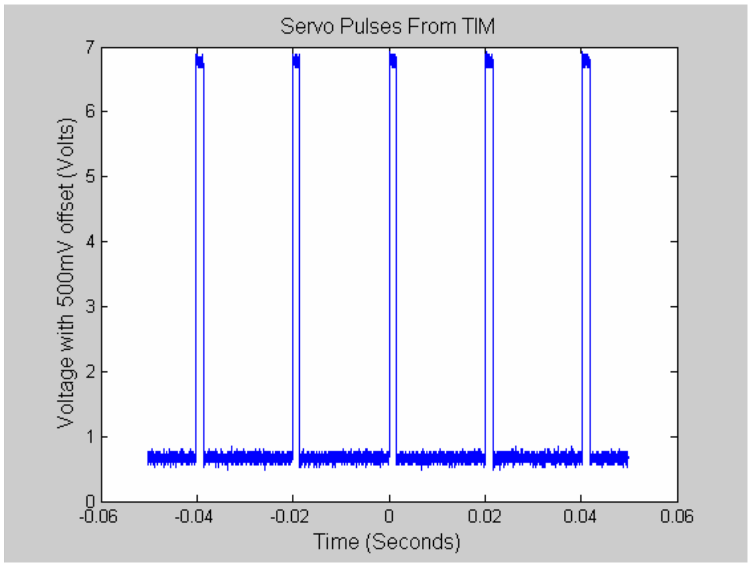


Figure 5.2 – Servo Pulses from Module

This same pulse was then applied with a Futaba S3010 servo attached, and then placed under load (**Figure 5.3**). The resulting pulse did exhibit a change, dropping from approximately 6.2 volts to just under 6 volts, but the results were still well within servo operation tolerances [**47**].

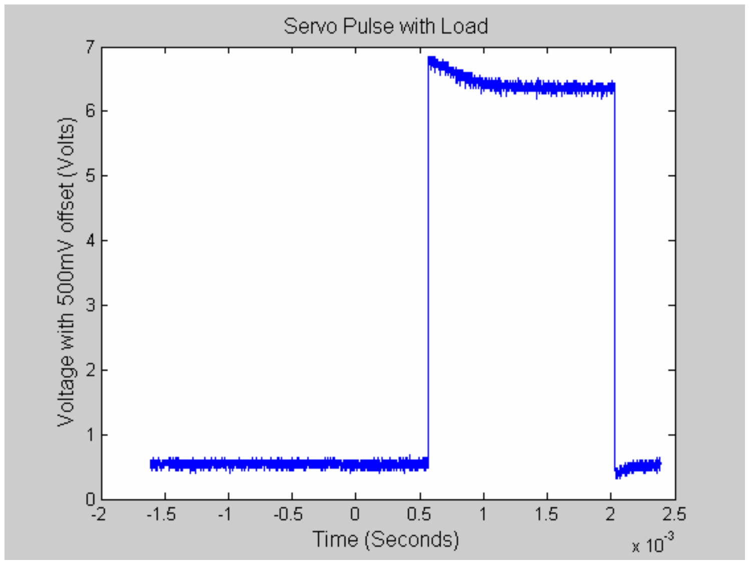


Figure 5.3 – Servo Pulse with Load

5.2.3 Range Testing

The last of the preliminary tests focused on the wireless functionality. Observing wireless input/output and communications between two Tiny Interface Modules is proof enough that basic code can work. The modules, however, needed to be tested to see the actual throughput and distances that could be expected. Since the wireless module was chosen to function as part of an intra-plane network, it was not necessary to determine if the one mile range claimed in the XBee module's data sheet was obtainable. A test was instead designed to determine operation at ranges within 100 feet. **Table 5.1** shows the number of missed packets at each range and data rate.

	Range		
Data rate	**1 meter**	**10 meters**	**30 meters**
9600	0	0	0
57600	2	2	2
115200	0	7	10

Table 5.1 – Packet Errors Versus Data Rate and Range

The errors are for each ten-thousand packets that were passed between the modules. A packet consists of 5 bytes, a 1 byte header and 4 bytes of data. It is worth noting that in range tests conducted at 1 meter and 115200 in a lab, the error rate jumped to 72 per/10000 packets. This most likely should be contributed to the existence of an 802.11 wireless network **[40]**. By the design of the wireless driver, bad packets are ignored, and with the high frequency that servo data is updated, the occasional bad packet would cause the servo to maintain its current position for an additional fraction of a second.

5.3 Application Testing

The second phase of Tiny Interface Module testing was done through application testing. This phase relied on the creation of realistic scenarios to test the modules' performance. To aid in the credibility of the test results, it was imperative that the tests utilize as much of the final flight hardware as possible. Therefore all components were placed within the airframe utilizing the airframe network cabling and batteries. This phase was also completed in two stages: IDEAnix integration and flight simulation.

5.3.1 IDEAnix Integration

This stage was completed primarily by Darren J. Brown because, by design, the goal was to quickly and easily adapt and execute the IDEAnix OS and test application on the Tiny Interface Module. Integration progressed as anticipated with the Tiny Interface Module providing all of the necessary hardware. The first step to verify the system was functioning correctly was to use a task to send a pulse across the CAN and have another module's task react accordingly. As seen from the oscilloscope screen capture in **Figure 5.4,** this interaction occurred successfully. The originating module transitions from a high to low value on a send, while the receiving module transitions from a low to high upon receiving data. From this test a baseline IDEAnix CAN transmission delay could also be established by taking the time delay between the transitions. This means that the minimum delay for IDEAnix to transmit data from one task to another task on another module was approximately 640 microseconds.

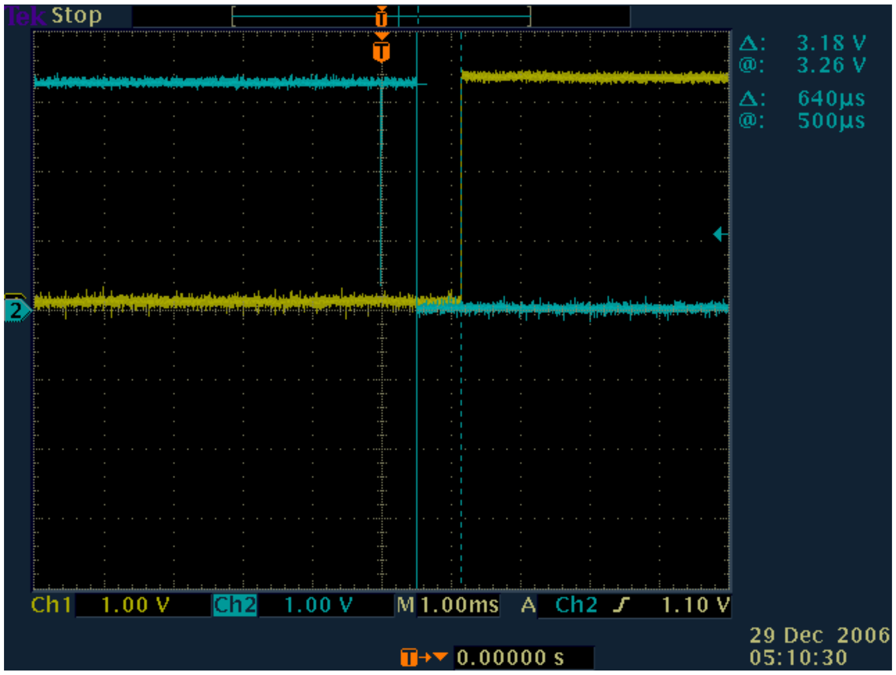


Figure 5.4 – CAN Transmit vs. Receive Delay

It is important to note that data is strictly baseline. Once future applications and code are developed, additional testing could be performed to see how the time delay is affected due to a high traffic as well as lower priority versus higher priority tasks.

5.3.2 Flight Simulation Testing

The successful completion of a flight simulation was the final test of the functionality of the Tiny Interface Module. The simulation, seen in **Figure 5.5**, demonstrates that the module can handle the demands of a flight ready system.

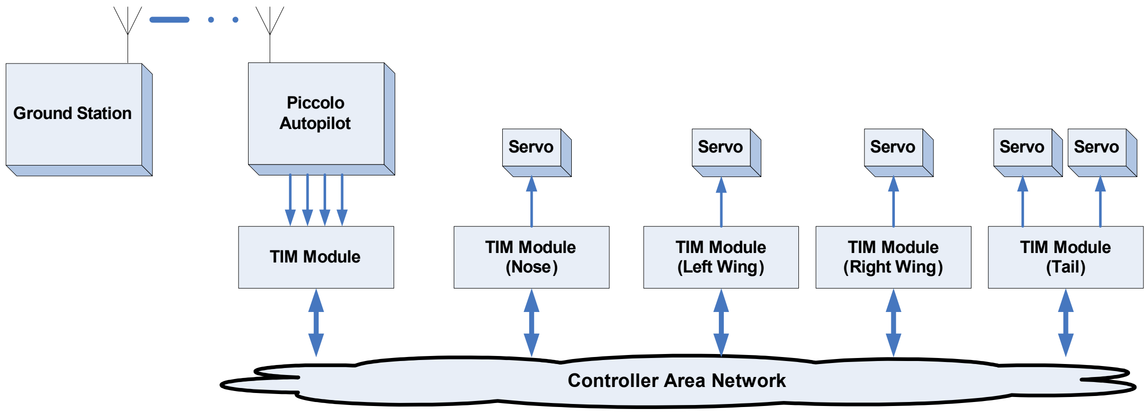


Figure 5.5 – Simulation Block Diagram

This system includes receiving commands from a ground station, shown in **Figure 5.6**, to a Piccolo autopilot. Next, the Piccolo outputs four pulse-width modulation (PWM) servo signals which are inputted, converted to long integer values, and encapsulated in a CAN packet using a Tiny Interface Module. Then the four other modules on the wired CAN network are able to receive the packets with servo position data and control the respective servo(s).

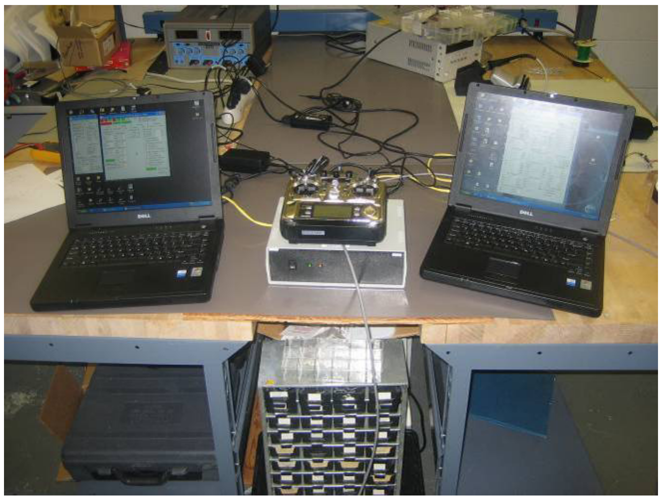


Figure 5.6 – Piccolo Ground Station Running a Simulation for Testing

As with the preliminary testing, the concern for power reliability with the servos attached still existed. Beyond being able to successfully run flight simulations, additional load was put on the servos during testing, shown in **Figure 5.7**, to ensure that the power would not drop below the servo required 4.8 volts.

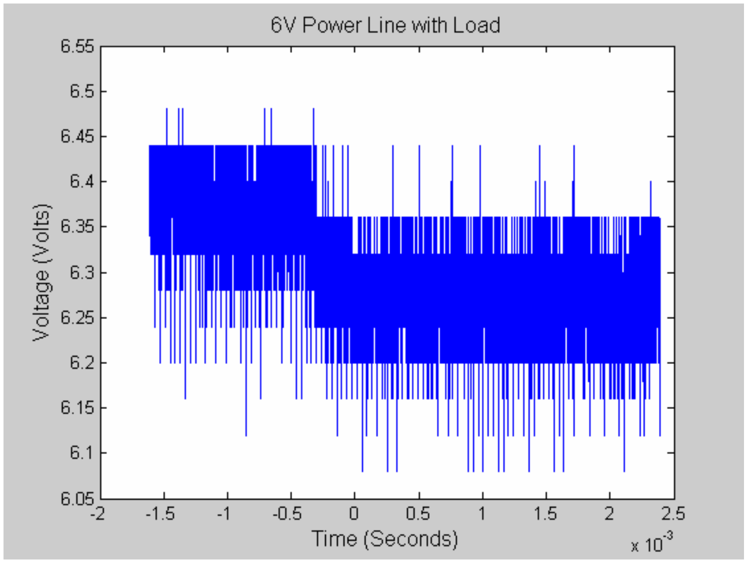


Figure 5.7 – Six Volt Power Line with Load

The final aircraft can be seen in **Figure 5.8.** This system includes (from right to left) a single tail module controlling the elevator and rudder, two modules in the center of the plane one, bringing in and parsing the autopilot signals and one controlling the throttle and nose gear, and finally one module in each wing controlling the respective ailerons.

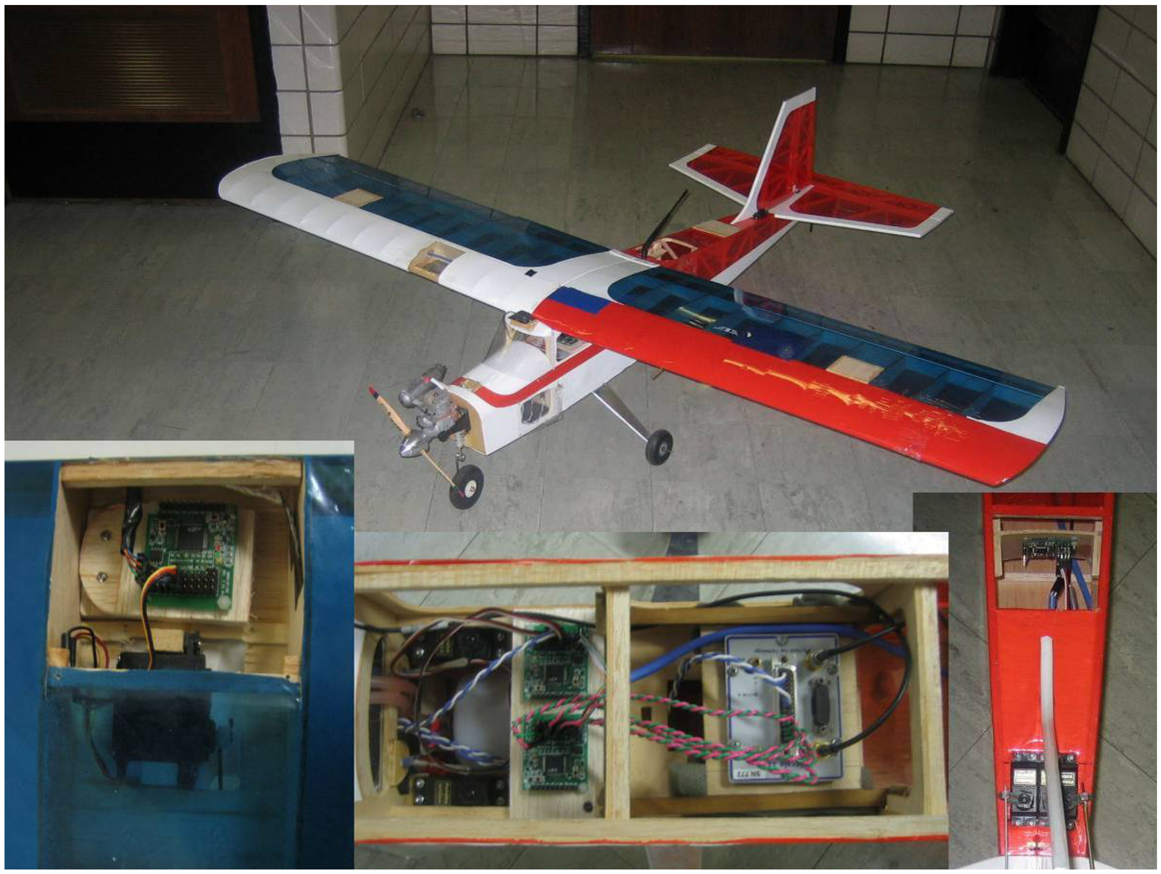


Figure 5.8 – In Aircraft Testing of Module and CAN Bus

The next stage of this test would be to conduct flight tests. Flight testing is a very time intensive task that is completely contingent on current weather conditions. When the appropriate conditions are available the system, from the flight simulations, is flight ready.

5.4 Future Work

Despite the successful test results illustrated above, with any design there is the possibility of future development. The two major areas of focus would be the addition of a measure of environmental tolerance and further development of the wireless system. Groundwork for the wireless system has been laid but a good deal could be done to keep up with the new and ever changing ZigBee protocol, as well as establishing ways to ensure wireless reliability.

Another area that could be addressed is the development of the Tiny Interface Module's environmental tolerance. There exist many options from mounting the module within a hobby box to utilizing special board, and component, conformal coatings. The hobby box prototype seen in **Figure 5.9** was developed as a preliminary test to further understand the cost, benefits and module mounting process. This initial test proved to be very costly, time intensive, and sacrificed the module's weight and size advantages.

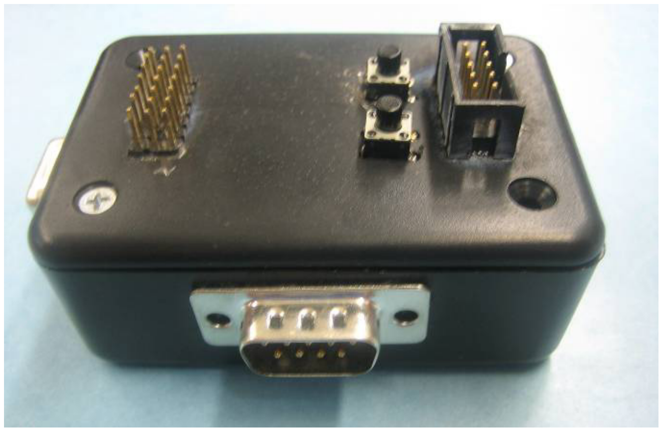


Figure 5.9 – Module Mounted in a Hobby Box

It is still evident, however, that other options exist that may require fewer design sacrifices while still providing module protection. Ultimately, the addition of environmental tolerance would improve the robustness and user-friendliness of the Tiny Interface Module.

5.5 Summary

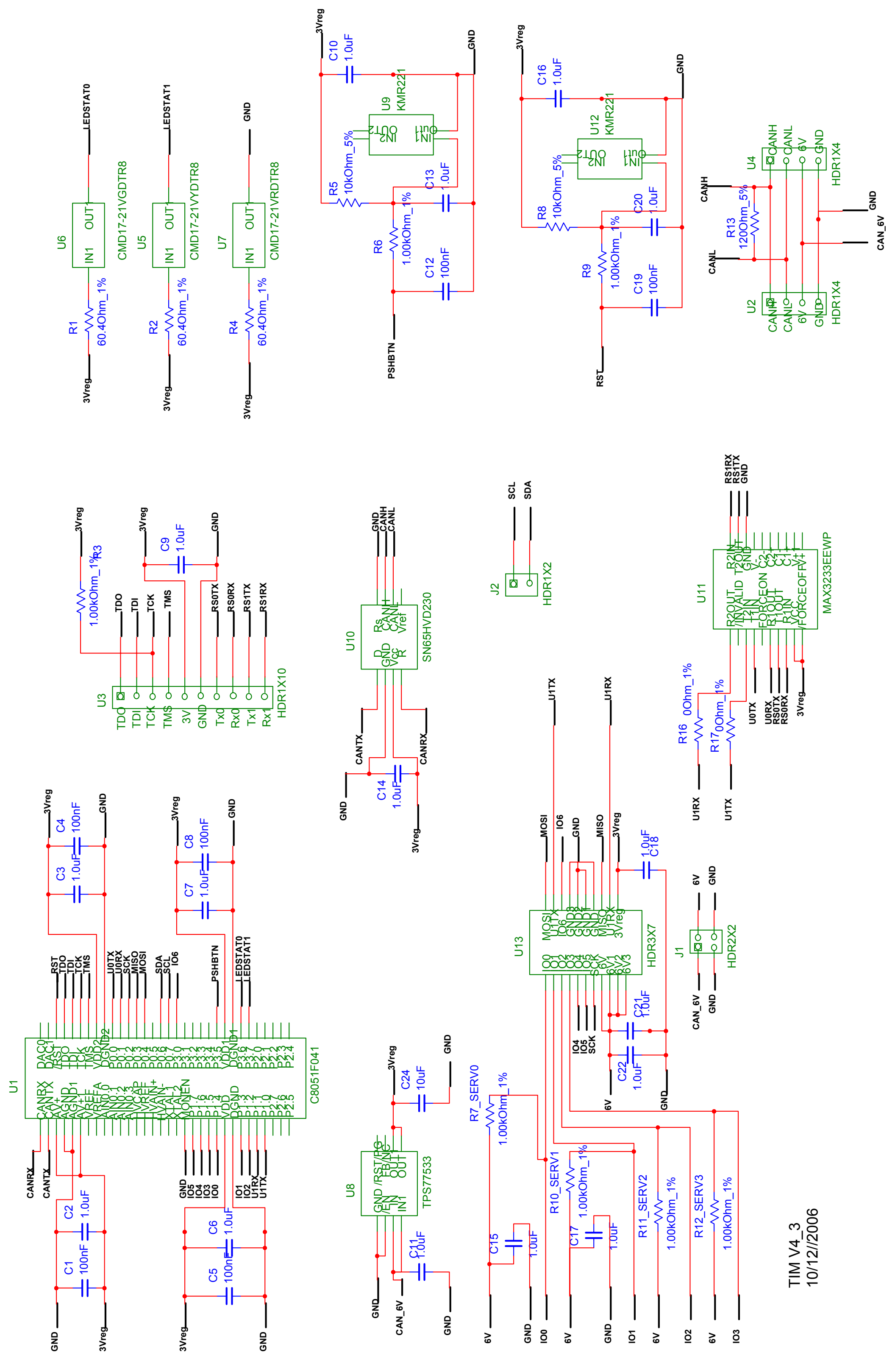
This chapter presented the Tiny Interface Module's testing and results and areas for future development. From this data the success of the design can be properly assessed in chapter 6.

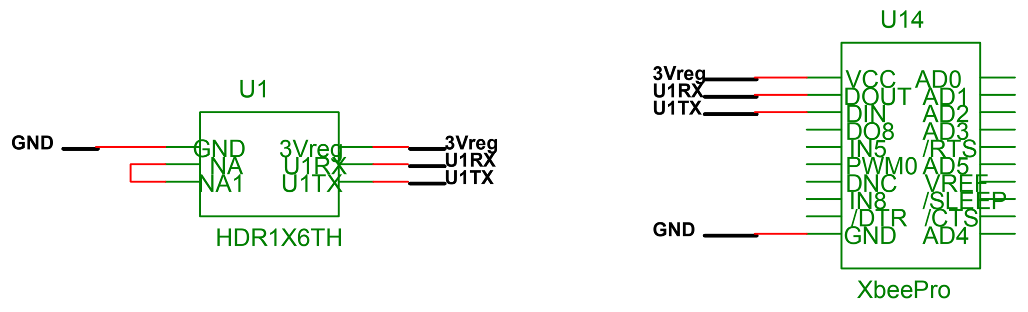
6 Summary

The goal of this project was to produce a network processing node for a rapidly, deployable, easily reconfigurable, light UAV. The first chapter provided the background and motivation of the node design. In chapter two the hardware design was discussed. Chapter three looked at the design of the software providing a generic initialization and wireless driver. Next in chapter four the design implementation was discussed. Finally the test designs and results were discussed. From these test results it can be concluded that the goals of the design were achieved since:

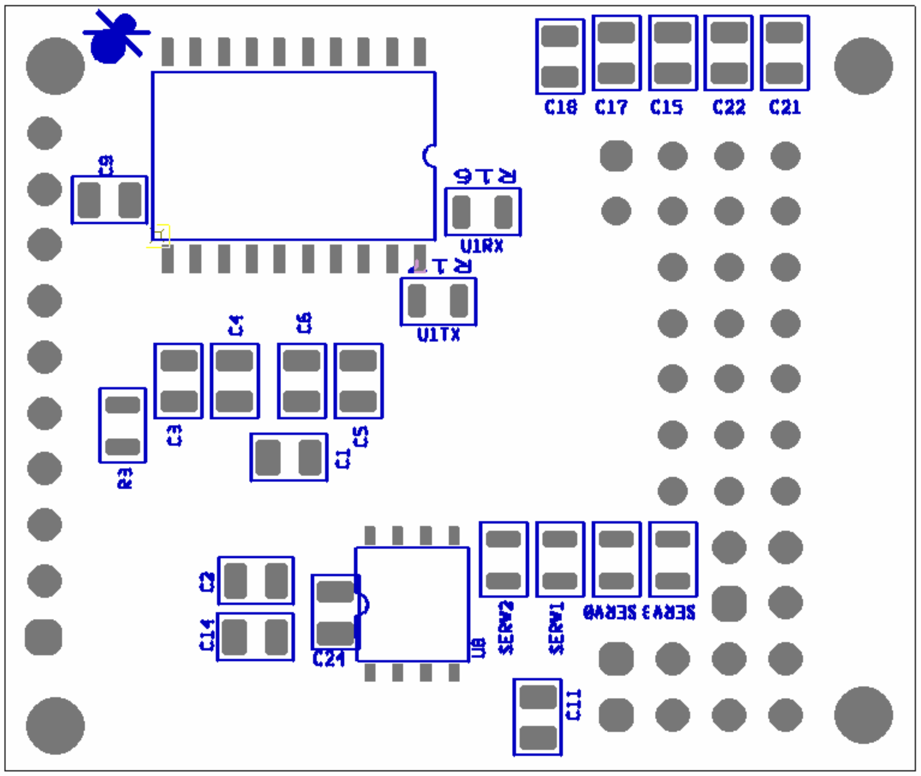
1 The target UAV system design (**Figure 1.7**) was successfully designed and tested (**Figure 5.5**).
2 The CAN communications network was able to deliver control information to the distributed Tiny Interface Modules and respective servos.
3 The node provided the necessary headers and connections to support direct interfacing with servos.
4 The node was sufficiently small to be distributed throughout a light aircraft.
5 The module successfully utilized CAN, IDEAnix, and a Silicon Laboratories 8051 microcontroller.
6 Finally, the node provides the foundation for future growth with the wireless network layer and additional hardware resources.

7 Appendix

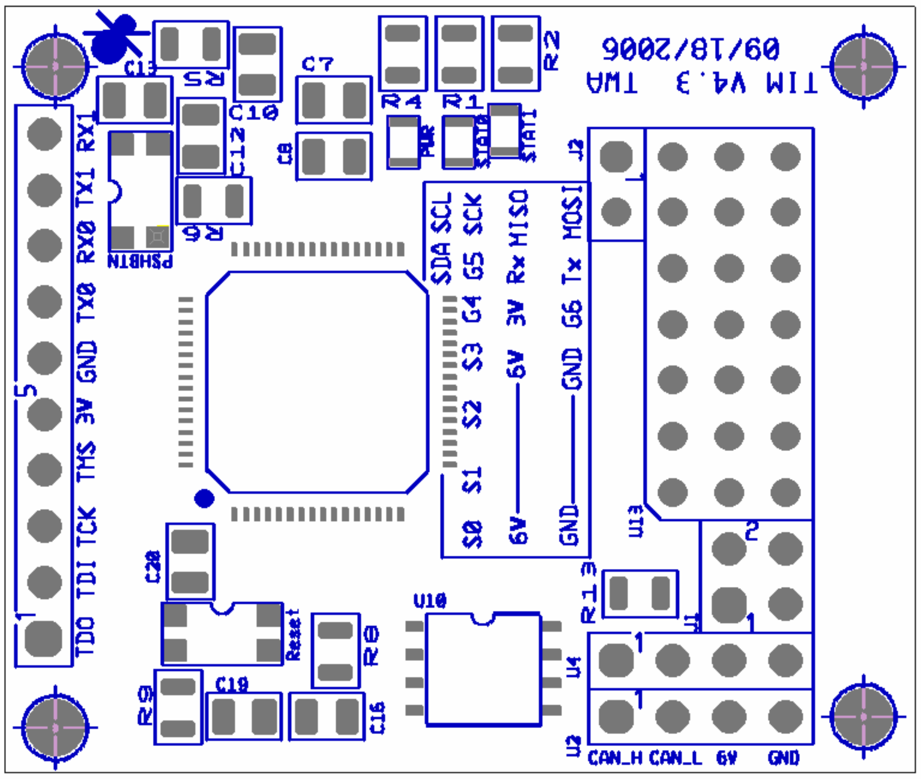

U6
CMD17-21VGDTR8
R1
60.4Ohm_1%
LEDSTAT0
U5
CMD17-21VYDTR8
R2
LEDSTAT1
U7
CMD17-21VRDTR8
R4
U9
KMR221
C10
1.0uF
R5
10kOhm_5%
R6
1.00kOhm_1%
C13
C12
100nF
PSHBTN
U12
C16
R8
R9
C20
C19
RST
U4
U2
HDR1X4
CANH
CANL
6V
GND
R13
120Ohm_5%
CAN_6V
3Vreg
U3
HDR1X10
TDO
TDI
TCK
TMS
RS0TX
RS0RX
RS1TX
RS1RX
R3
C9
U10
SN65HVD230
CANTX
CANRX
C14
J2
HDR1X2
SCL
SDA
U11
MAX3233EEWP
R16
0Ohm_1%
R17
U1RX
U1TX
U0TX
U0RX
U1
C8051F041
C1
C2
C3
C4
C5
C6
C7
C8
U8
TPS77533
C11
C24
10uF
U13
HDR3X7
C18
C21
C22
J1
HDR2X2
R7_SERV0
R10_SERV1
R11_SERV2
R12_SERV3
C15
C17
IO0
IO1
IO2
IO3
MOSI
MISO
SCK
IO6
TIM V4_3
10/12//2006



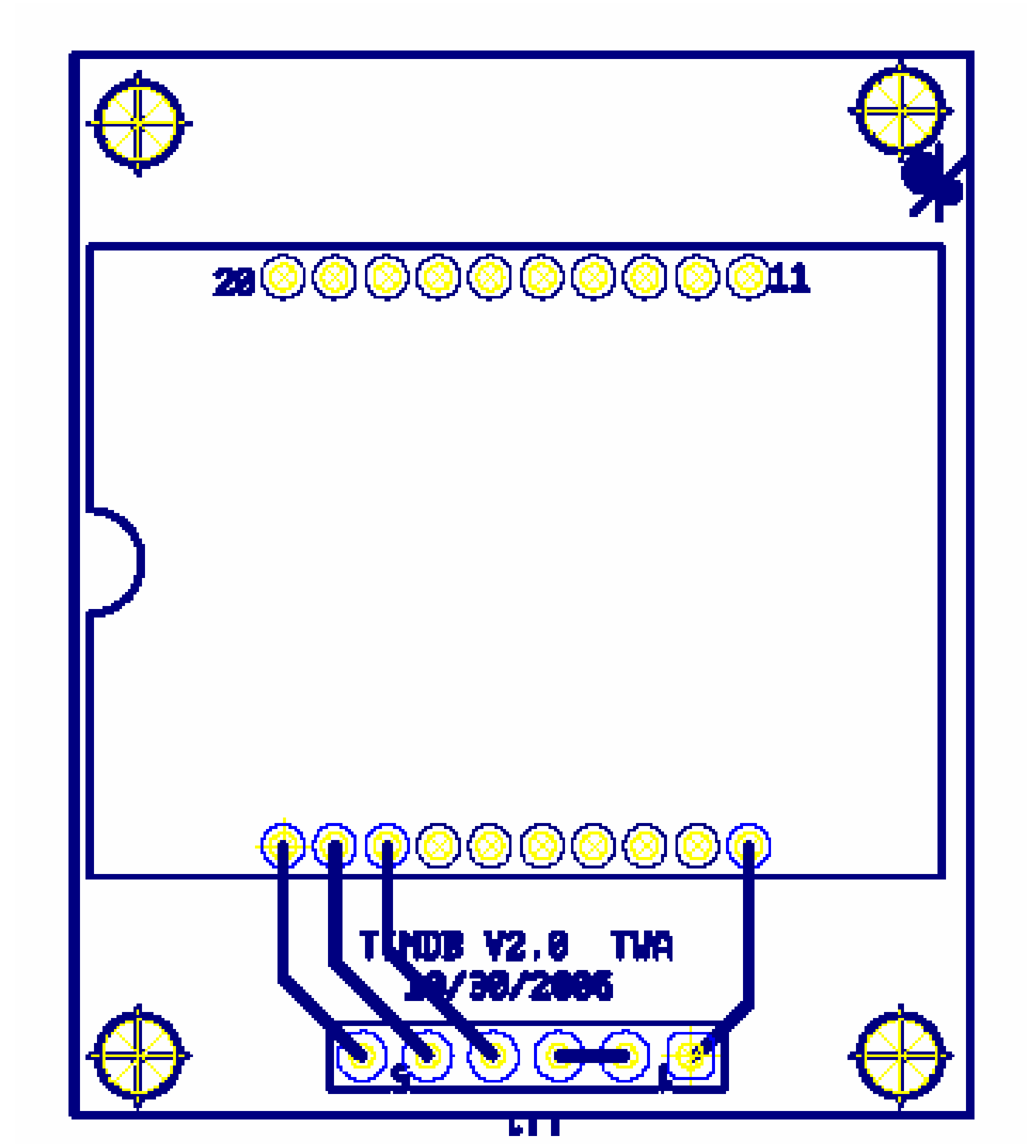
Wireless Daughter Board V2_0
11/10/2006


C18
C17
C15
C22
C21
C9
R16
U1RX
R17
U1TX
C4
C6
C3
C1
C5
R3
C2
C14
C24
U8
SERV2
SERV1
SERV0
SERV3
C11

B)


TIM V4.3 TWA
09/18/2006
C13
R5
C10
C7
R4
R1
R2
C12
C8
PWR
STAT0
STAT1
J2
RX1
TX1
RX0
TX0
GND
3V
TMS
TCK
TDI
TDO
5
1
PSHBTN
R6
SDA SCL
G5 SCK
Rx MISO
Tx MOSI
G4 3V G6
S3 6V GND
S2
S1
S0 6V GND
U13
2
1
C20
Reset
R8
U10
R13
U1
U4
U2
R9
C19
C16
CAN_H CAN_L 6V GND

A)


20
11
TCMDB V2.0 TWA
10/30/2006

8 References

[1] Intelligent Embedded Dependable Architectures Lab, University of Kentucky, http://www.engr.uky.edu/idea

[2] U.S. Air Force Fact Sheet "MQ-1 Predator Unmanned Aerial Vehicle," http://www.af.mil/factsheets/factsheet.asp?fsID=122 , January 2007.

[3] Photos Courtesy of the United States Air Force, http://www.af.mil/

[4] U.S. Air Force Fact Sheet "Global Hawk," http://www.af.mil/factsheets/factsheet.asp?fsID=175 , October 2005.

[5] A. Simpson, O. Rawashdeh, S. Smith, J. Jacob, W. Smith, and J. Lumpp. "BIG BLUE: A High-Altitude UAV Demonstrator of Mars Airplane Technology" IEEEAC paper #1436. IEEE Aerospace Conference. Big Sky, Montana. March 2005.

[6] Jean Labrosse, MicroC/OS-II The Real-Time Kernel Second Edition. CMPBooks, 2002.

[7] Photo Courtesy of the University of Kentucky College of Engineering, http://www.engr.uky.edu/news/archives/2005/AIRCATfirstcompetition.htm

[8] Darren Brown, Tim Arrowsmith, Alicia Mylin, Robert Koontz Adam Fox, Neal Phelps, Daniel Thomas, Johnathan Rowe, Robert Jones, David Jackson, Adam Groves, and Michael Sama. "AIRCAT: Airborne Intelligent Research Craft for Autonomous Technology." 2005 AUVSI Student UAV Competition Paper

[9] Darren Brown, Craig Collins, Dale McClure, Ala Aldin Meriden, Phillip Profitt, Matt Smith, and Van Yadack. "University of Kentucky Aerial Robotics Team: 2006 AUVSI Student UAV Competition Design." 2006 AUVSI Student UAV Competition Paper

[10] O. Rawashdeh, D. Feinauer, C. Harr, G. Chandler, D. Jackson, A. Groves, and J. Lumpp. "A Dynamically Reconfiguring Avionics Architecture for UAVs" AIAA-2005-7050. AIAA Infotech@Aerospace Conference. Arlington, Virginia. September 2005.

[11] O. Rawashdeh and J. Lumpp. "A Technique for Specifying Dynamically Reconfigurable Embedded Systems" IEEEAC paper #1435. IEEE Aerospace Conference. Big Sky, Montana. March 2005.

[12] Silicon Laboratories, http://www.silabs.com

[13] Freescale Semiconductor, http://www.freescale.com

[14] Micro/sys, http://www.embeddedsys.com

[15] Helicomm, http://www.helicomm.com/

[16] Vaglica, J.J., Gilmour, P.S., "How to select a microcontroller," Spectrum, IEEE, vol. 27, no. 11, pp.106-109, 1990.

[17] Robert Bosch GmbH, CAN Specification 2.0, 1991.

[18] G. Chandler, C. Harr, O. Rawashdeh, D. Feinauer, D. Jackson, A. Groves, and J. Lumpp. "Wireless Extension of an Avionics Bus for Prototyping and Testing Reconfigurable UAVs" AIAA-2005-7151. AIAA Infotech@Aerospace Conference. Arlington, Virginia. September 2005. http://www.engr.uky.edu/idea/wiki/lib/exe/fetch.php?id=publications%3Adefault&cache=cache&media=publications:0509_aiaa_chandler.pdf

[19] Corrigan, S., "Introduction to the Controller Area Network (CAN)" Texas Instruments Application Report August 2002

[20] Silicon Laboratories C8051F04x Datasheet, http://www.silabs.com

[21] Simpson, C., "Linear and Switching Voltage Regulator Fundamentals," 1995 from http://www.national.com

[22] XILINX, Appl. Note XAPP623 (v2.1), Feb. 28 2005.

[23] Dean, A., "Embedded Communication Network Pitfalls," http://www.embedded.com/ , 1997

[24] National Instruments, "FlexRay Automotive Communication Bus," http://zone.ni.com/devzone/cda/main

[25] TTA-Group, "TTP – Frequently Asked Questions," http://www.tttech.com 2004

[26] National Instruments, "Controller Area Network (CAN) Overview," http://zone.ni.com/devzone/cda/main

[27] IEEE standard 802.11 – 1999 "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," Reaffirmed June 2003

[28] IEEE standard 802.15.1-2005 "Wireless medium access control (MAC) and physical layer (PHY) specification for wireless personal area networks (WPANs)." June 2005

[29] IEEE standard 802.16 - 2004 "Air Interface for Fixed Broadband Wireless Access Systems," October 2004

[30] Bluetooth Specification "Core v2.0 + EDR," http://www.bluetooth.org November 2004

[31] Craig, W. C., "ZigBee" Wireless Control That Simply Works," http://www.zigbee.org

[32] IEEE standard 802.15.4 - 2003 "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," October 2003

[33] ZigBee Specification Version 1.0, December 14 2004, http://www.zigbee.org

[34] MaxStream XBee-Pro Product Manual, http://www.maxstream.net , July 2006

[35] Freescale Semiconductor MC13201 Datasheet, http://www.freescale.com , 2006

[36] Feldmeier, D. C., “Fast Software Implementation of Error Detection Codes,” IEEE/ACM Trans. Networking, vol. 3(6), pp. 640-651, December 1995.

[37] Stone, J., Greenwald, M., Partridge, C., Hughes, J., “Performance of Checksums and CRC’s over Real Data,” IEEE/ACM Trans. Networking, vol.6 no.5, pp. 529-543, October 1998.

[38] Sikora, A. “ZigBee Competitive Technology Analysis,” http://www.zigbee.org

[39] United States Code of Federal Regulations Chapter 1 Part 18.101, http://www.access.gpo.gov/nara/cfr/waisidx_05/47cfr18_05.html

[40] Sikora, A., Groza, V. F., “Coexistence of IEEE802.15.4 with other Systems in the 2.4 GHz-ISM-Band,” Instrumentation and Measurement Technology Conference, Ottowa, Canada May 2005

[41] Texas Instruments SN65HVD232 Datasheet, http://www.ti.com , 2002

[42] Maxim Integrated Products MAX3233E Datasheet, http://www.maxim-ic.com , 2004

[43] Texas Instruments TPS76533 Datasheet, http://www.ti.com , 1999

[44] Texas Instruments TPS77533 Datasheet, http://www.ti.com , 2003

[45] PCBExpress Printed Circuit Tutorial, http://www.pcbexpress.com/technical/tutorial.php , 2007

[46] Silicon Laboratories C8051F04x Development Kit User’s Guide, http://www.silabs.com , 2006

[47] Futaba, http://www.futabarc.com

9 Vita

Personal

Birth Place: Evansville, IN, USA
Date of Birth: November 21, 1981

Education

Bachelor of Science (B.S.) in Electrical Engineering, May 2005
Department of Electrical and Computer Engineering
University of Kentucky, Lexington, KY, USA

Work Experience

Research Assistant
08/04 - 12/06
Intelligent Dependable Embedded Architectures (IDEA) Laboratory
University of Kentucky, Lexington, KY

Co-operative Education Student
08/02 – 05/04
ADTRAN, INC., Huntsville, AL

Computer Technician
01/01 – 08/03
Interdisciplinary Human Development Institute (IHDI)

Honors

University Scholars Program
Finalist of the National Co-op of the Year Competition
College of Engineering Academic Scholarship
University of Kentucky Merit Scholarship
Ambassador for the University of Kentucky College of Engineering
Dean's List
Tau Beta Pi Honor Society
Eta Kappa Nu International Electrical Engineering Honor Society

Publications

Darren Brown, Tim Arrowsmith, Alicia Mylin, Robert Koontz Adam Fox, Neal Phelps, Daniel Thomas, Johnathan Rowe, Robert Jones, David Jackson, Adam Groves, and Michael Sama. "AIRCAT: Airborne Intelligent Research Craft for Autonomous Technology." 2005 AUVSI Student UAV Competition Paper