UAV Pirates and SilenTrack Integration

June 4th, 2010
Professor John Saghri

Approved By:

Travis Dean	_____
Hushnak Singh	_____
Ashley Wager	_____
Matt Woolridge	_____

I. INTRODUCTION

The goal of integrating the SilenTrack system with the UAV Pirating system is to allow for loitering UAVs to be pirated faster and at greater distances than the old UAV Pirating system was capable due to decision making and visual confirmation by the user. Since the pirating system was already setup and verified, our tasks were broken into 5 main areas: hardware, takeover firmware, takeover software, pirating algorithm, and SilenTrack interface. The hardware circuitry was in place for the most part, but was lacking in documentation, so it was documented and transferred from individual circuits on two breadboards to a single PCB and modified to run on a single power source to reduce parts count and to reduce the troubleshooting time with all the loose connection possibilities. The PCB also made the system look cleaner and made it more durable. The takeover firmware needed to be able to generate a signal from the Nexys board, and the takeover software needed control of the signal from the PC. The design restriction here was the available data path width from the PC to the FPGA across the USB connection. We decided early that we had to use the remap channels and the Tx and Rx channel registers to be able to transmit the data to the FPGA. We also found out that the Interlink Buddy Box connection can only do 8 channels, so we could work with the old USB IP core without having to regenerate the user logic for additional registers. The SilenTrack communication was outlined for us by the Raytheon Team. They provided us with the information to communicate with the system and provided some sample data and a data format for how everything is transmitted. We had to write an interface for this that would establish a connection with SilenTrack and read in the appropriate data, discarding a lot of the extra data SilenTrack automatically sends. The pirating algorithm was difficult to design due to the unknown effects of sending out signals to a random UAV with unknown channel mapping. We designed a brute force system where we lock out the original pilot, save the steady state settings, test changing the data on each channel and observing the effects until we have the three channel mapping being done by the system. Further detail on each member's contributions are available in the member contributions portion towards the end of the report and in a summary table in the appendix.

General Problem Summary

Remote Controlled (R/C) aircraft are inexpensive, quickly built, simple to fly, and are capable of delivering biological or explosive weapons. Conventional detection of these small Unmanned Aerial Vehicles (UAVs) is difficult, and destruction of the craft using sophisticated weaponry is not cost effective.

Client Overview

Our customer is Raytheon, a technology leader specializing in defense, homeland security, and other government markets throughout the world. With a history of innovation spanning more than 80 years, Raytheon provides state-of-the-art electronics, mission systems integration, and other capabilities in the areas of sensing, effects, command, control, communications and intelligence systems, as well as a broad range of mission support services.

Related Work

Systems are in progress that can triangulate the position of an enemy transmitter and also optically track an enemy aircraft.

II. PRODUCT DEFINITION

Need Statement

Due to their typically small size and low cost, unfriendly unmanned aerial vehicles (UAVs) can be difficult or nearly impossible to detect using conventional radar or to eliminate using advanced weaponry; with the ability to cause significant harm to the public and provide unnoticed surveillance to an enemy, a system capable of both detecting and taking control over such aircraft would greatly increase the defensive abilities of our armed forces, police, and other security professionals.

Requirements

Marketing

To interface a UAV pirating system with a real time three dimensional video tracking system in order to automatically and continuously scan and detect the presence of suspicious UAVs and their appropriate radio frequency control signals. The system will alert the user of potential threats, decode detected control signals, and take control over the UAV.

Engineering

The integration of the UAV pirating system with the SilenTrack video tracking system will allow the combined system to automatically identify and track a suspicious air vehicle visually as well as identify and decode the RF control signal. Once a threat has been identified, the system will execute an algorithm to associate each control channel with the corresponding flight control surface. Once the control channels have been decoded the user can use a hand held radio controller to maneuver the drone which is now under their command.

Constraints

As this project is more "proof of concept" than a polished system that would be suitable for marketing, we constrain some of the requirements in order to focus on obtaining control of an enemy aircraft that may have non-standard channel mapping. Rather than scan the entire 72MHz range, we will conduct testing with a few channels of our choosing. We can achieve this by interfacing a common channel scanner available at any hobbyist store. These scanners have transistor logic to indicate which channels are active, which will serve as inputs for our digital system. A software implantation written in the C programming language will provide the logic necessary to alert the user of active signals and to use the channel information to set the output transmitter accordingly. Software will also determine the number of active servo channels and will attempt to make intelligent assumptions to determine the servo channel mapping. After mapping these assumed channels to our transmitter, the pilot may activate the high-powered transmitter and direct the drone to a safe location.

III. DESIGN

Overview of team process

Since this was an integration project, the specifications were rigidly set on both the SilenTrack side and on the existing UAV technology. This simultaneously gave us guidance and restrictions in our design work. A lot of the requirements for the interface were already outlined in the old documentation and what the SilenTrack team provided us, and we spent a lot of time in the early stages figuring out how we were going to integrate all the components, so when we split into our own parts, we designed all our components for easy integration.

System Diagram

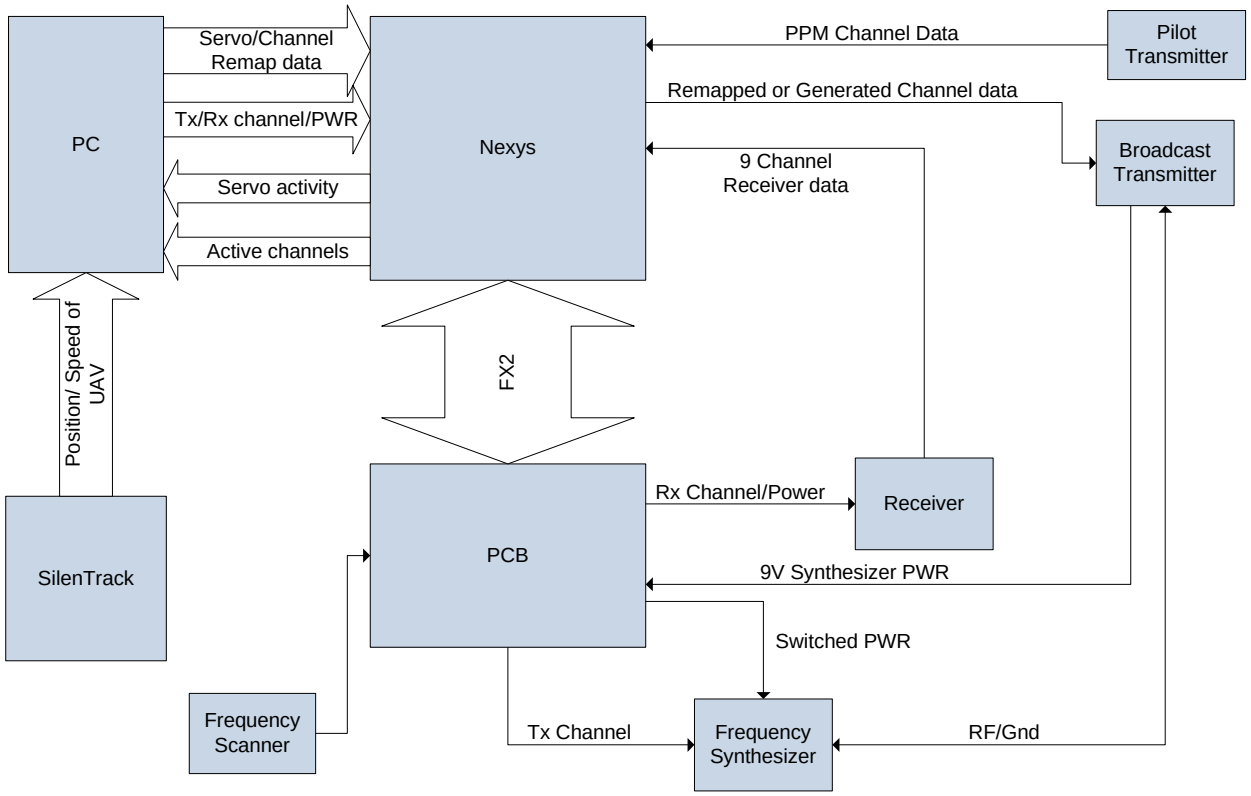

Figure 1 - System Diagram

Hardware Block Diagrams

Receiver Reader

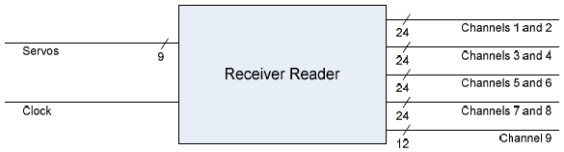

Figure 2 - Receiver Reader VHDL module

The receiver reader is a VHDL module running on the Nexys board that reads the servo information from the detected enemy signal. When the UAV Pirate system tunes the receiver into the correct channel, the receiver outputs information to the servos in pulses - the length of the pulse determines the angle of the servo. The receiver reader measures the lengths of all these pulses, so that the Nexys board can send servo positions to the GUI. The servo activity then gets displayed in bars for the user.

Buddy Box Module

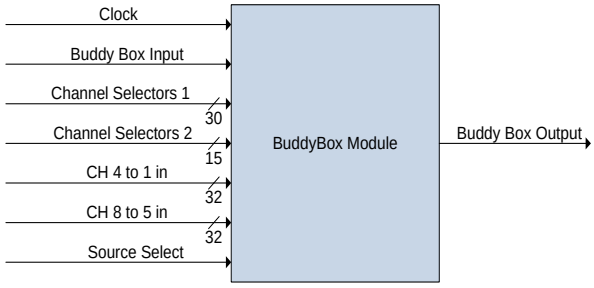

Figure 3 - Buddy Box VHDL Module

To simplify the piloting of aircraft whose servos are "scrambled," the UAV pirate system can reconfigure the channel layout as needed. It takes advantage of the “buddy box” interface: a cable by which the joystick positions of one transmitter, acting as a “trainer”, can be passed to a transmitter which then sends the signal out. Instead of connecting these two transmitters directly together, the Nexys board rearranges the channels in between the trainer and the transmitter. The trainer’s signal output goes into the Buddy Box module, which reads the information and then sends a rearranged signal

to the transmitter. The buddy box module combines the old Channel Remapper with the new direct signal generation module. A high signal on Source Select outputs a signal generated from the channel data, while a low signal outputs remap data. The CH 4 to 1 in and CH 8 to 5 in are each 32 bit signals, 8 bits for each of the 8 channels that the buddy box connection supports.

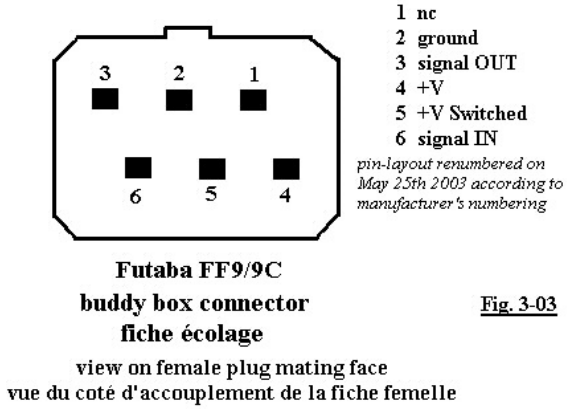

Figure 4 - Buddy Box Connection pins

Frequency Synthesizer and Channel Selectors

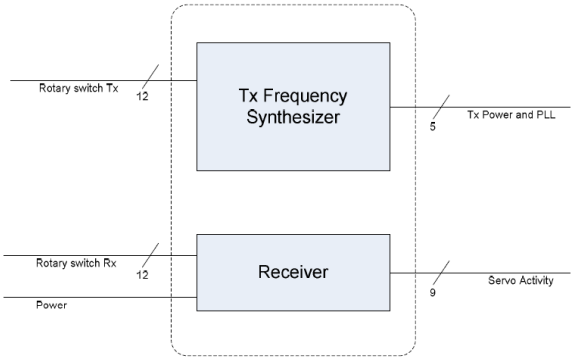

Figure 5 - Frequency Synthesizer and Receiver

The frequency synthesizers allow the user to choose any of the channels to receive or transmit on. Channels 11-60 are selectable via rotary switches on the back of each module, but our system is able to digitally tune these modules using the information given in the following truth table. The channel selector uses digital switch ICs to allow the FPGA development board to short different pins on the rotary switch to ground to set different channels (refer to truth table).

Figure 6 - Channel Selector Hardware diagram

Position on Rotary Switch	Rotary Switch Pins (rearranged to show truth table)					
	5	6	1	2	3	4
0	1	1	1	1	Always Ground	Always Ground
1	1	1	1	0		
2	1	1	0	1		
3	1	1	0	0		
4	1	0	1	1		
5	1	0	1	0		
6	1	0	0	1		
7	1	0	0	0		
8	0	1	1	1		
9	0	1	1	0		
	0= ground		1= high voltage			

Figure 7 - Truth Table for Channel Selector switches

Frequency Checker

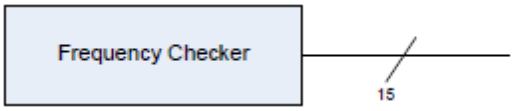

Figure 8 - Frequency Checker

The frequency checker allows the user to view which channels are in use on the 72 MHz range. The checker was an off the shelf product that constantly iterates through each of the 50 channels and simply lights up an led if there is activity on that frequency. Our module taps into the frequency checker by reading the values being passed to the LED columns and rows. The checker's LEDs are a 5 by 10 grid that are controlled by a MOSFET at the beginning of each row and column. For the rows, the signal is controlled by a PMOSFET normally high at 2.9V and drops to 0V. The columns are NMOSFETs and are normally low at 0V and rising to 400mV. Both rows and columns operate at a 60Hz frequency. When a single led has both its appropriate row and column transistors turned on, current is allowed to flow and the led turns on. The signals coming off the frequency checker are run through a comparator

circuit used to pull up the voltages of the signals. Since the voltages coming off the rows and columns are 2.9V and .4V respectively, they must be pulled up to at least 3.3V for the Nexys 2 board to recognize the signals. LM239AJ comparator chips are used with a 5V rail to pull the voltages of the frequency checker to just above 3V. From there, the signals are sent to the Nexys board through the FX2 connector.

Nexys 2 FPGA Development Board

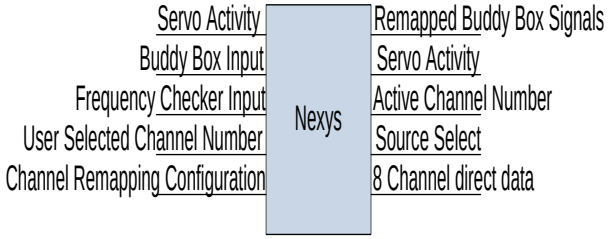

Figure 9 - Nexys 2 Board System diagram

The Nexys 2 FPGA development board is used as the main processing unit in the system. Processing algorithms are used to communicate between the user and the hardware. All modules are connected to this unit and it serves as the major terminal to direct communication between them.

Software Algorithms

User Interface

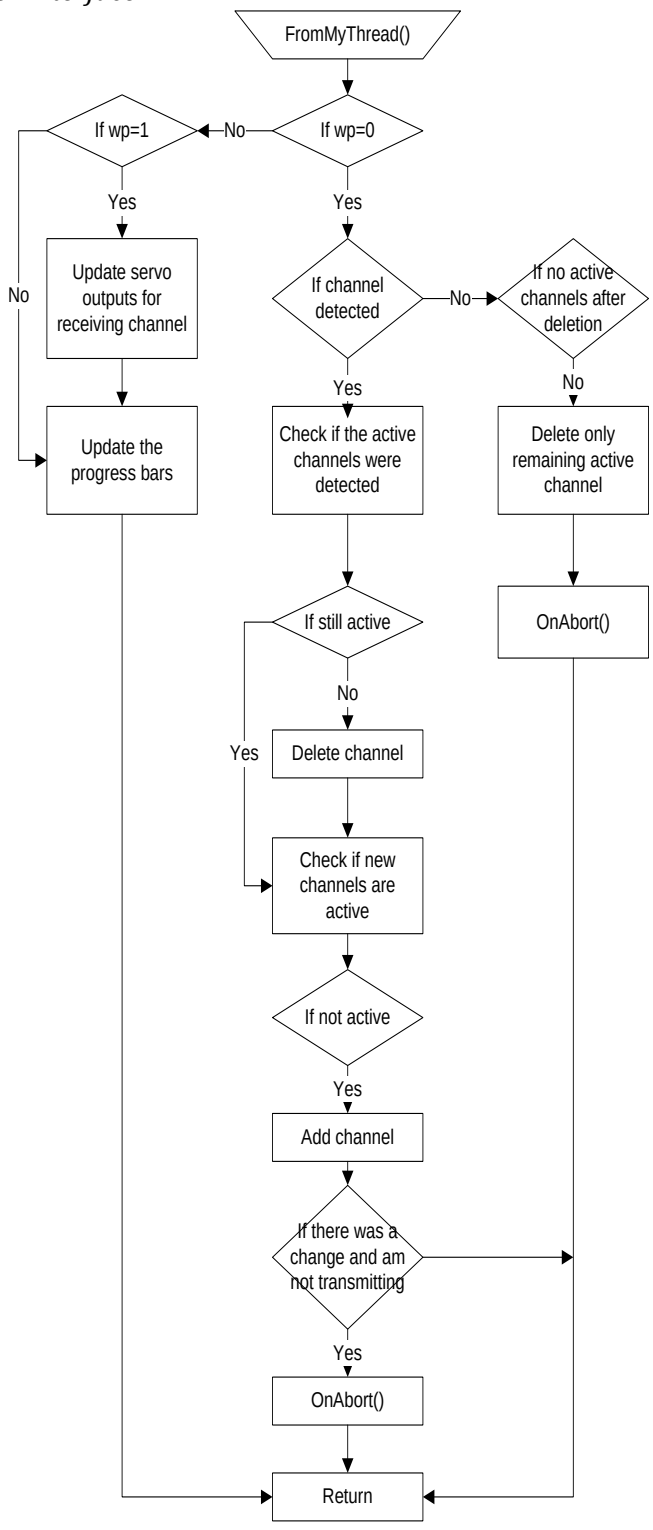

Figure 10 -FromMyThread Software Flowchart

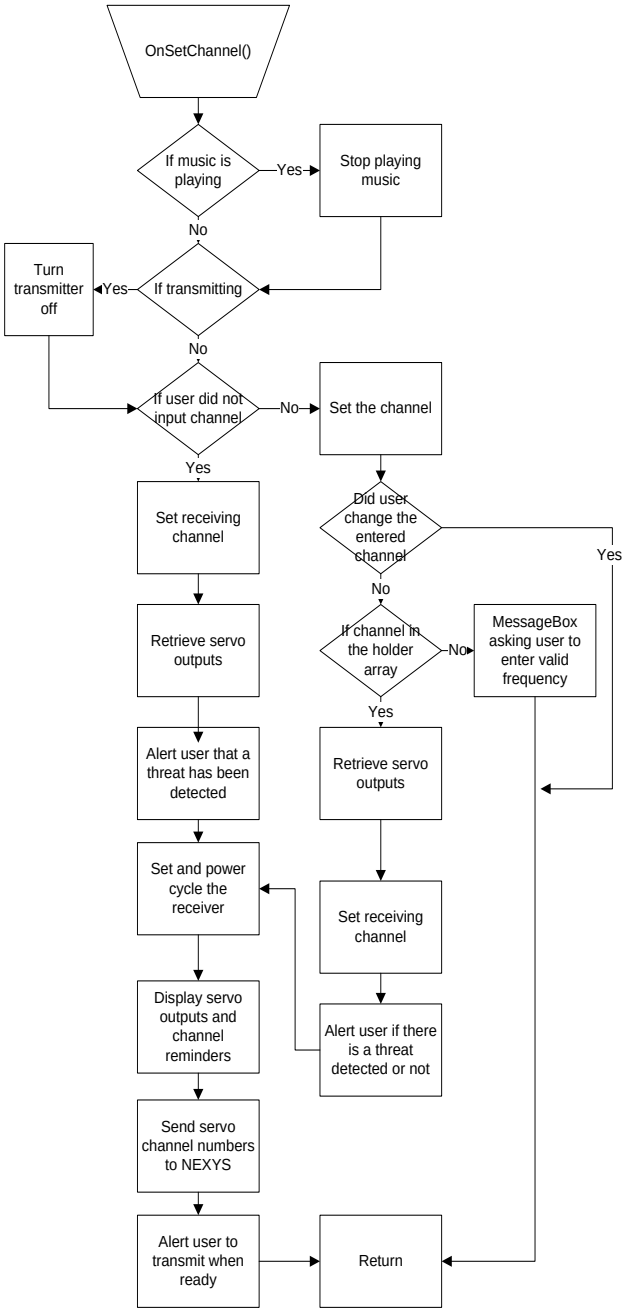

Figure 11 - OnSetChannel Software Flowchart

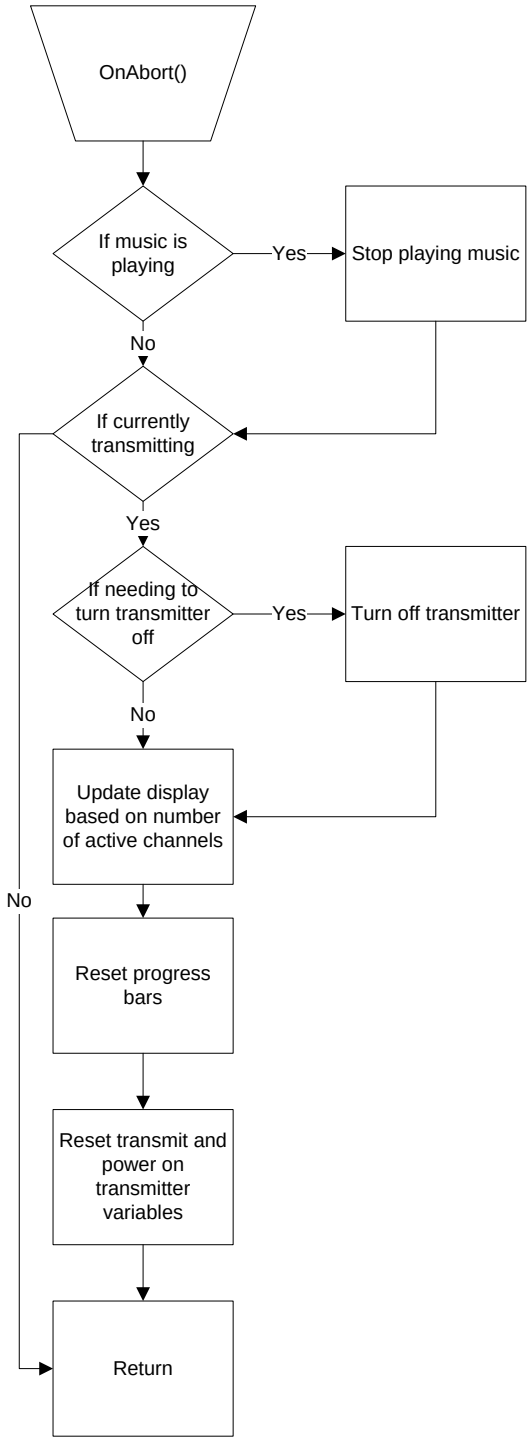

Figure 12 - OnAbort Software Flowchart

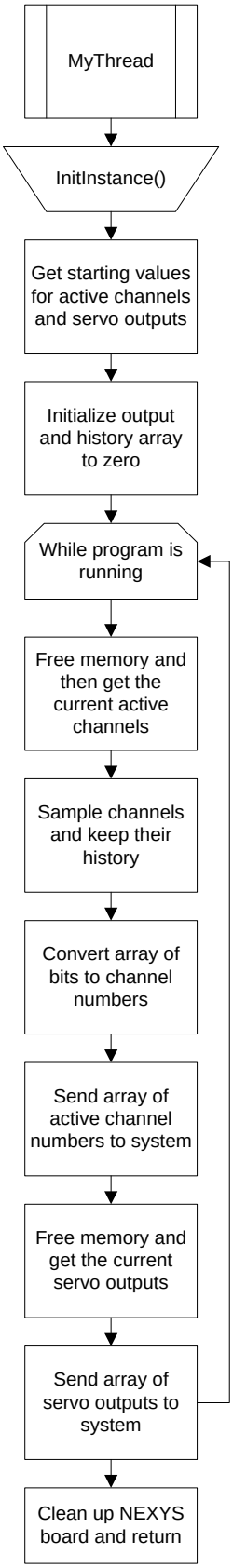

Figure 13 - MyThread Software Flowchart

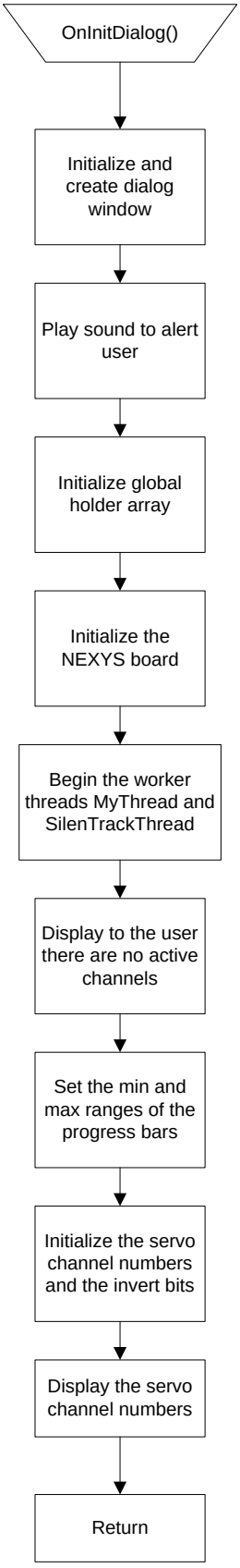

Figure 14 - OnInitDialog Software Flowchart

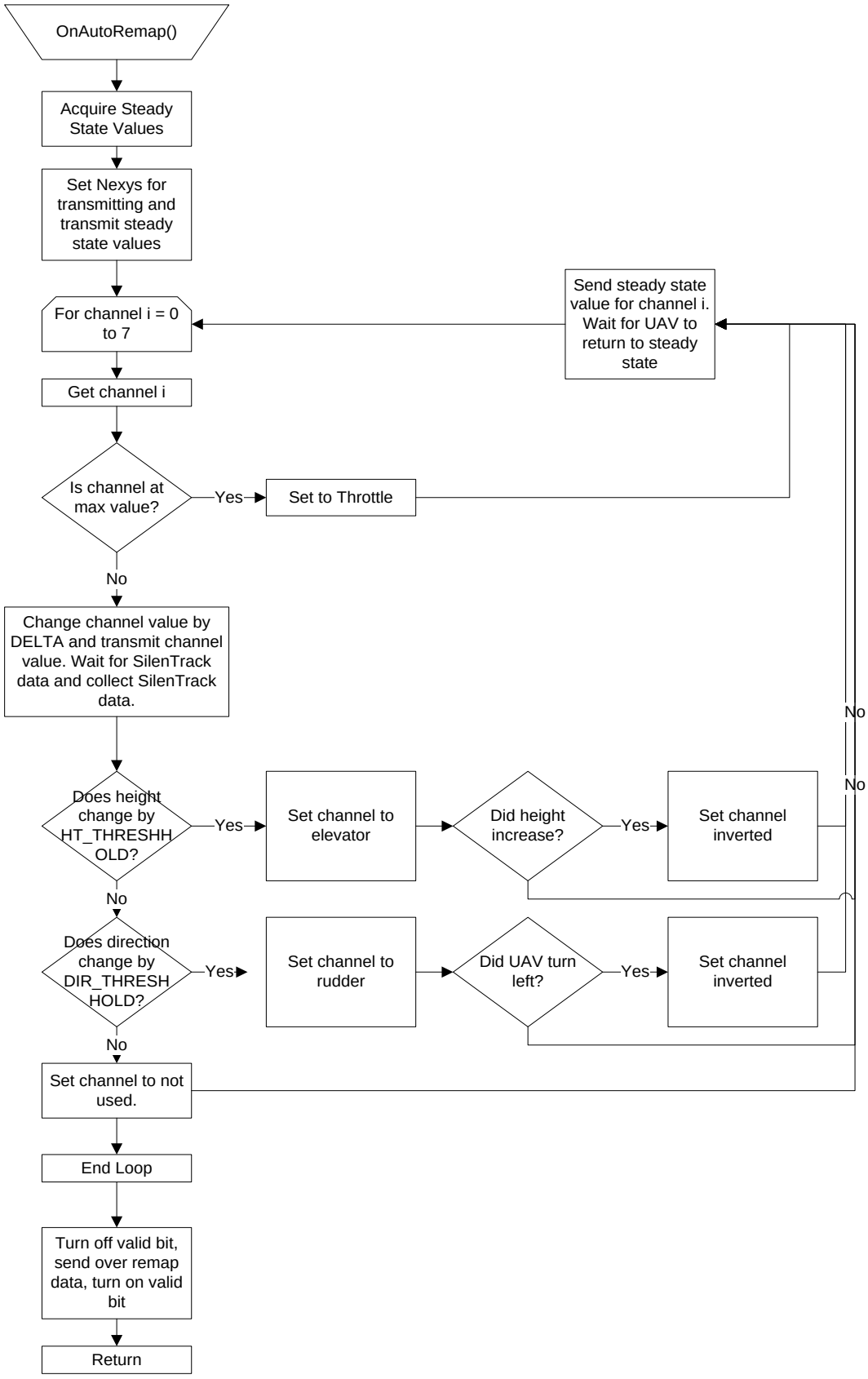

Figure 15 - OnAutoRemap Software Flowchart

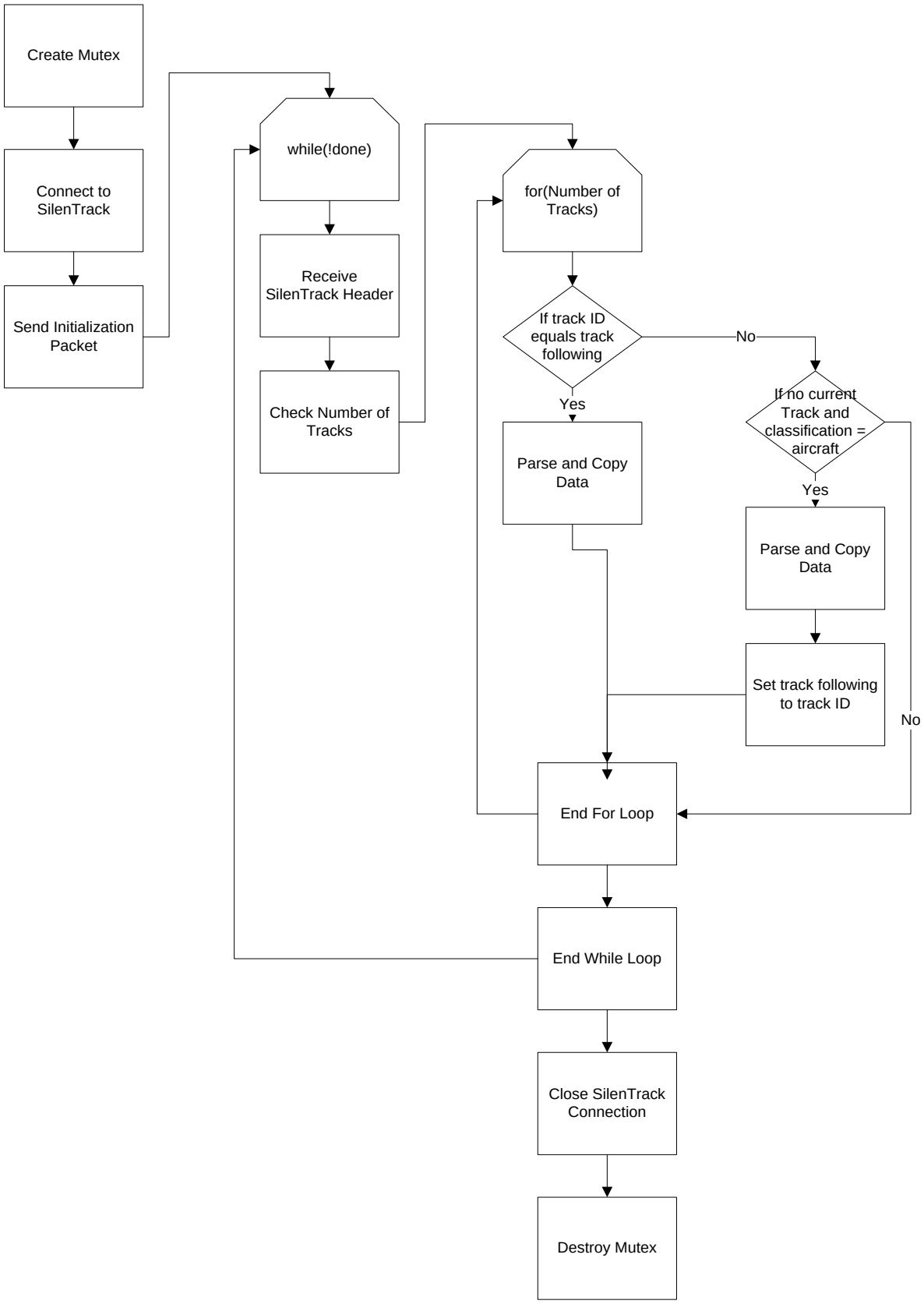

Figure 16 - SilenTrack Thread

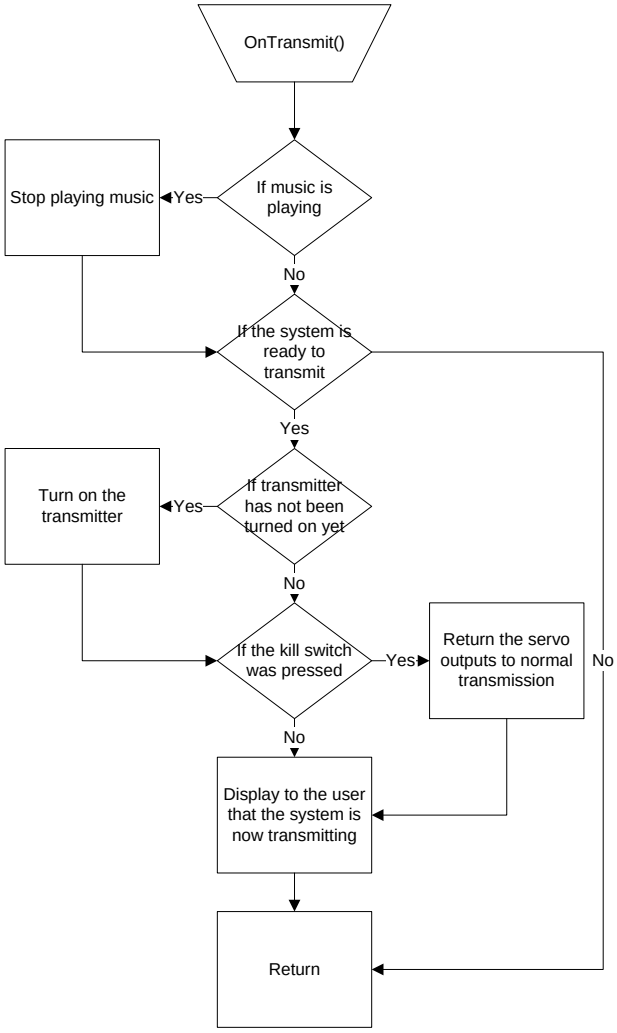

Figure 17 - OnTransmit Software Flowchart

Receiver Reader
(VHDL described in words rather than flowcharts)

The receiver reader is a VHDL module that connects to the servo pins of a receiver and reads in the pulse lengths of each signal. It works with a 100MHz clock (period of 1 us) so that the length of a pulse, in us, can be counted and output in 12-bit vectors. (A pulse is when the voltage is high; the servo signals idle low and switch high for a pulse.) 9 of these 12-bit vectors, for up to 9 received servo channels, are output on Channels1and2, ..., Channels7and8, and Channel9.

Channels1and2(11 downto 0) stores channel 1, with bit 11 being the MSB and bit 0 the LSB. The number is unsigned, and represents the most recent pulse length (in us) seen on the input pin Servos(0). Channels1and2(23 downto 12) stores channel 2, with bit 23 being the MSB and bit 12 the LSB. This number represents the most recent pulse length seen on Servos(1). This same pattern follows for all other channels, except that Channel9, being only 12 bits long, stores only channel 9.

The receiver reader's input pins should be connected to the signal pins for every servo coming out of the PCM receiver.

Buddy Box Module
(VHDL described in words rather than flowcharts)

Buddy Box Module is a VHDL module that reads one PPM signal over the buddy box interface and outputs a 8-channel PPM signal based on the input. Each of the 8 output channels can be configured to be any input channel, a short pulse, or a long pulse. Any of these can also be "inverted," which turns a short pulse into a long pulse and vice versa.

The pulse measurer in ChannelRemapper.vhd reads the buddy box signal and outputs the pulse lengths, in us, of each channel into ChannelsData.

9 channel mappers, in ChannelMapper.vhd, configure RemappedChannelsData based on ChannelsData. Each of these takes ChannelsData as input, has an input ChannelSelector to decide what channel will get routed to ChannelOut (numbers 0-8 selects channels 1-9, number 9 is a short pulse, number 10 is a long pulse). If the ChannelInvert bit is high, the output will be the inverse of the selected channel.

Finally, the BuddyBoxOutput in OutputModule.vhd outputs a 9-channel PPM signal based on the data in RemappedChannelsData. Physically, we cut into the signal line of a Futaba trainer cable to remap the signal between the trainer and the transmitter. On the trainer side, the PPM out pin is connected to the BuddyBoxIn pin on the Nexys board, and on the transmitter side we connect the PPM in pin to the BuddyBoxOut pin on the Nexys board. The V+ switched pins and the ground pins are connected together, with ground wire also connecting to the Nexys ground.

This module also has 2 32-bit inputs for direct signal output on 8 channels. Each of the 8 channels can be varied from 0x00 to 0xFF to go from minimum to maximum pulse width on that channel. Switching between the remap module and the direct signal generation is handled by the Source Select signal which selects direct output when high and remap output when low.

Scurvy.c

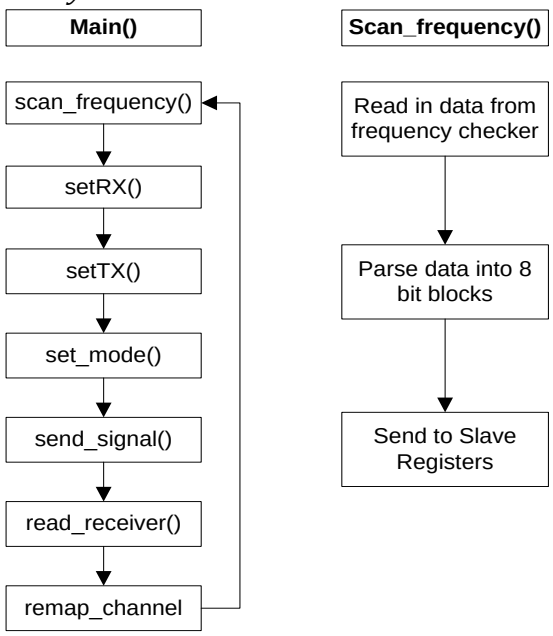

Figure 18 - Scurvy Main and Scan_Frequency software flowcharts

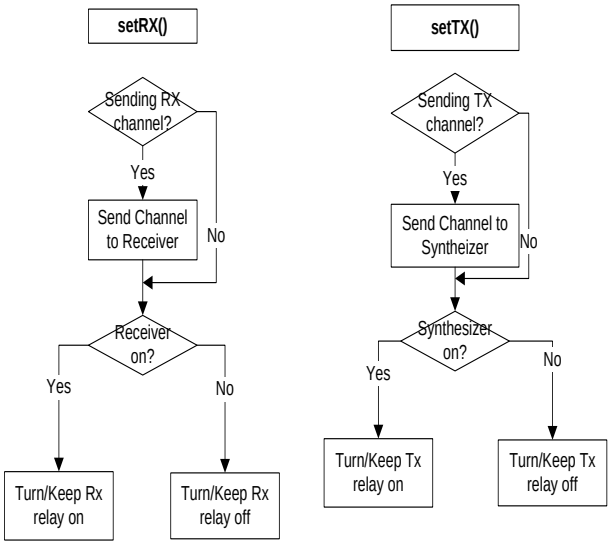

Figure 19 - setRX and setTX Software Flowcharts

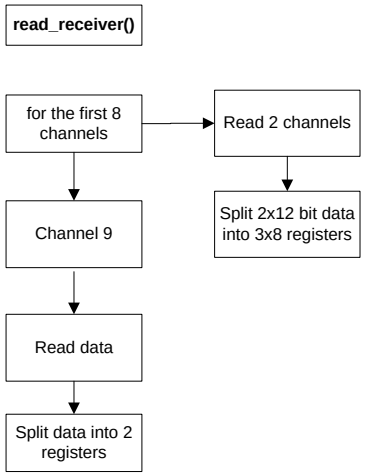

Figure 20 - read_receiver Software Flowchart

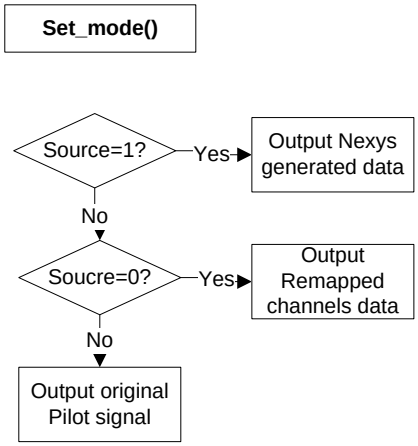

Figure 21 - Set_mode Software Flowchart

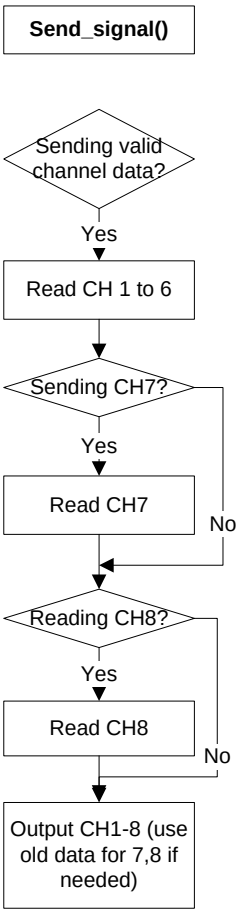

Figure 22 - Send_signal Software Flowchart

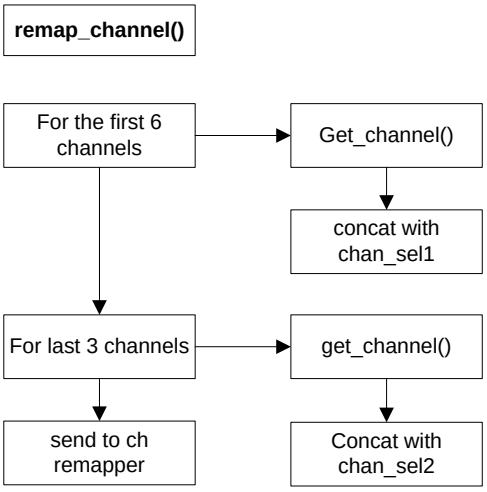

Figure 23 - remap_channel Sotware Flowchart

Nexyscomm.cpp
(used to communicate between the PC and Nexys 2)

Setup_nexys()
This function is used to set up and initialize communication between the PC and Nexys board. It first initializes the device table and tries to connect to the Nexys board. If the device is not found, then it exits and displays an error message. If the device is found then it gets the device name and opens up the data connection for the board. If either if these function calls fails then they will display an error message and exit as well. At the end of the function, the board is ready to communicate with the PC.

Cleanup()
This function is used to close the data communication between the PC and Nexys board and should always be called upon ending the program.

Get_scanner()
Get_scanner is used to get the data from the Nexys board concerning the frequency checker. It reads in 7 bytes of data representing the 50 channels displayed on the checker. These bytes are stored in a char array which is returned at the end of the function.

Cs_set_channel()
Cs_set_channel will take the channel passed in and send the appropriate values to the Nexys board. The channel passed in is a single integer with allowable values ranging from 11 to 60. The ones and tens digit is calculated separately each as a 4 bit number, and then concatenated together to form an 8 bit value and sent to the appropriate data registers for the transmitter and receiver. As an example, if the requested channel is 17, then the lower 4 bits of the sent data would be 7 (0111) and the upper 4 bits would be 1 (0001) so your data register would be 00010111.

Cs_power_on()
This function will either power cycle the receiver or turn on and off the transmitter based on the value passed in. If the value is a 1, then the receiver will be power cycled by sending a 1 to the CS_PWR data register, will sleep for half a second, and then turn it back off by passing a 0 to the same register. If the value sent to

cs_power_on is a 2, then if the transmitter is currently turned off, it will be turned on, and visa versa.

Import_receiver()
Import_receiver() is used to read in the values being read by the receiver. The algorithm utilizes a loop that calculates 2 channels at a time, because that is how the data is paired up by the Nexys board. In total, each channel is represented by 12 bits, but data can only be received in 8 bit increments. The data registers are set up that each channel's lower 8 bits of data are in its own register. Its upper 4 bits of data is paired up with another channel's upper bits to form an 8 bit number. The main goal of this function is to take these registers and put each channel's data back together. Starting at channel 1, the loop reads in the data for the lower 8 bits of the odd and next even channel (i.e. would process channels 1 and 2 together). Then it reads in the register that contains the upper bits for both channels. After splitting those upper bits apart, it concatenates the values to the lower 8 bits and creates an integer representing the entire channel. It returns an array of integers, each integer representing a channel.

Remap()
Remap() basic function is to tell the Nexys board how to remap the channels. It takes the data from the GUI and passes it down to the board where it is implemented in the VHDL. Each channel is represented by a 5 bit number. The lower 4 bits designate which channel it is being remapped to, and the most significant bit is the invert bit. To be passed down to the Nexys board, the channels data (least significant 4 bits) are paired up to form 8 bit values and all the invert bits are in their own register. This function creates those registers and passes the data down to the Nexys board.
Send_to_nexys()
This function does exactly what it says, send the data to the specified register using the dpcutil library provided by Digilent.

Rcv_from_nexys()
This function gets the data from the specified register using the dpcutil library provided by Digilent.

send_to_nexys()
Send to Nexys is a function to write data to the specified register using the dpcutil library provided by Digilent.

set_mode()
Set mode is a new function that writes the mode and current power states of the transmitter and receiver to the mode/power register, CS_PWR. The Mode/Power register is described furthur in the Appendix.

set_tx()
Set_tx changes the channel on the transmitter and turns it on. It also updates the mode/power register on the PC side so the computer does not accidentally turn off the transmitter when updating that register with different information. There is a 250 ms sleep in this program to ensure proper channel switching time is allowed.

set_rx()
Set_rx does the same thing as set_tx, except for the receiver. The power cycle delay in the receiver is 500 ms instead of 250 ms.

Components

Nexys 2 FPGA Development Board

Figure 24 - Nexys 2 FPGA Developement Board

Frequency Checker

Figure 25 - Frequency Checker

Frequency Synthesizer

Figure 26 - Frequency Synthesizer

Futaba 9C Transmitter

Figure 27 - Futaba 9C Transmitter

Futaba PCM Receiver

Figure 28 - Futaba R319 DPS PCM Receiver

System Integration and Testing

Test Plan

Since the components are all varied in their functionality, each component had its own tests to verify functionality. For the hardware testing, we built the new board, reconfigured the old firmware to work with the new hardware and tested all the functionality of the system to make sure everything worked as it was with the old circuitry.

For the Takeover firmware, the main task was to test that the signal being generated from the Nexys board was being accepted by the dedicated transmitter and by the RC receiver. To test this, the switches were used as the 8 bit control, and the same data was sent on all the channels after creating a VHDL wrapper for what would be the final Buddy Box Module. The takeover software was tested by writing a new nexys_comm file and a console application to run the command to test all the features. The console application tested all the old features as well as the shared registers and signal generation from the Nexys Board. The verification of the Nexys board generating takeover signals was a 5 second loop that moved the Elevators and Rudder from one extreme to the center to the opposite extreme and changed the throttle to values between 0 and 30%.

The SilenTrack communication had the additional challenge of security and clearance. We were unable to test this until recently because we had no access to the SilenTrack system. After our first visit to Raytheon we were able to modify our software to successfully establish communication with SilenTrack and process the data it was sending out to a format we could use.

The test for the Pirating algorithm is to first measure how accurately we can retransmit the steady state signal, then check the effects of the signals we send out to measure the effects. After this, we have to verify that the software is recognizing specific maneuvers associated with changes in the signal such as increased throttle and changes in the rudder or elevator.

Results and Analysis

The hardware has been successfully changed to use a single power supply, have all components on one board and to have more rugged modification to COTS parts. It has been tested with the original firmware and software as well as the new firmware and the console app that tested all the function of the new firmware.

The takeover firmware and software were originally tested with an oscilloscope to make sure the waveforms measured the PPM output of the Futaba 9C transmitters. The signal was then connected through the Buddy Box connection to make sure it could control servos hooked up to a receiver. We also implemented the test procedure described in the test section to display a test pattern on the airplane to verify that we are controlling the airplane from the PC.

So far we have only had two chances to meet with Raytheon to test our software with SilenTrack. In the first meeting we had no cameras, so we could only test our communication with the computers. We didn't have success with it initially since some of the information we had on establishing a connection was incorrect. After a few modifications we were able to communicate with SilenTrack and parse the data correctly. We verified this at our meeting with the SilenTrack team in El Segundo. We were able to read the data for pedestrians and vehicles in the parking lot and covert them to the right data format.

Due to the limited interaction and time with the SilenTrack team we have not had a chance to test the pirating algorithm. In the first interaction we had no cameras, so we would not be able to observe anything through the SilenTrack system. During the second visit we were not able to fly an airplane because the SilenTrack system is setup over the Raytheon employee parking lot, and it was filled up on the day we went to test out the rest of the system. So, in the interest of not causing unnecessary property damage, we were only allowed to record movement of the UAV while controlled by a person and record the corresponding SilenTrack data.

User Interface

The Graphical User Interface (GUI) is coded using C++ in Visual Studio 2008. First, communication with the Nexys board needed to be tested to make sure the new communication protocols worked properly. Manually inputted data was sent via the Nexys communication to simulate the sending of channel data and then switching back to remap data. Second, testing with SilenTrack communication was done with sample SilenTrack data. This was done in two ways. The first way this was tested was by running through a loop of sample output data from SilenTrack and verifying that the information was passing through and being parsed properly. The second way was by connecting the GUI with the SilenTrack system on-site at Raytheon in El Segundo and collecting live data from the parking lot. The last part of the GUI testing requires testing the channel remap algorithm. Since this test requires both the use of the SilenTrack system and an actual flight of a UAV, this piece has not been fully tested yet. The SilenTrack system has only recently been made available for us to use at a Raytheon facility, and we have not had the opportunity to fly a UAV at the same time and location where we could use SilenTrack. Based on these difficulties, it sounds reasonable to use a passive algorithm that doesn't take control of the plane while it remaps the channels, but instead remaps channels as it observes a human controller flying the plane. This latter method will be a slower way to remap channels. For instance, it would require a controller to change the plane's height before the algorithm can figure out which channel is associated with elevation. A future test is being planned with the UAV and SilenTrack at a Raytheon facility in Goleta.

System Integration:

After each module was verified as working correctly they were integrated together. This test relied heavily upon successful integration of SilenTrack and Nexys communication for all components to function together properly. The system needed to be tested at a Raytheon site to make use of the live SilenTrack system.

BUDGET AND JUSTIFICATION

Custom PCB

The custom PCB was used to replace the previous bread board implementation of circuits to perform level shifting of frequency checker output and circuits to program 72MHz receiver and frequency synthesizer. The Custom PCB produced marked improvement in system reliability and stability.

Futaba TP-FSM RF Frequency Synthesizer Module

This COTS component is used to synthesize any frequency on the 72 MHz band. One was purchased to create a parallel upgraded system to replace the original UAV Pirates system. A Parallel system was built to allow the original system to remain operational until the new system was tested and verified to be fully operational.

Futaba R319 DPS 9-Channel Receiver Module

This COTS component is used to receive any frequency on the 72 MHz band. One was purchased to create a parallel upgraded system to replace the original UAV Pirates system.

Hobbico 72MHz Frequency Checker

This COTS component is used to scan the 72MHz frequency band to check for activity on all channels. One was purchased to create a parallel upgraded system to replace the original UAV Pirates system. A second unit was purchased to replace a unit that was damaged by accidental over voltage.

4.8V Rechargeable Battery Pack

The system was modified to run all Pirate components except for the transmitter on 4.8V. One additional battery pack was purchased to extend the run time of Pirate system.

Voltmeter
The voltmeter plugs into the battery bus to allow the user to monitor battery voltage. If battery voltage is getting low the user can change out one of the battery packs for a fully charged one without having to power off the system.

Relay SSR 1A Mini-SIP
These relays are used to power cycle the transmitter and receiver. Two were purchased to create a parallel upgraded system to replace the original UAV Pirates system.

IC Quad Bilateral Switch 14-DIP
These switches are used to program the appropriate channels for the receiver and synthesizer. Eight were purchased to create a parallel upgraded system to replace the original UAV Pirates system and to have several spares.

IC Quad Voltage Comparator 14-DIP
The comparators are used to perform voltage level shifting to interface the 72MHz frequency checker with the Nexys board. Eight were purchased to create parallel upgraded system to replace the original UAV Pirates system and to have several spares.

Connectors, Cables, and Hardware
Various connectors and cables were purchased and modified to interface the various COTS modules, the custom PCB, and the Nexys board. Hardware was purchased to mount the individual components to a Plexiglas project board.

Multiplex Easy Star RC Airplane Fuselage
Main structural component of airplane used to test our design and verify range of signals. One fuselage was ordered to replace the original one that was severely damaged while learning to fly the RC plane.

BILL OF MATERIALS

Item	Qty	Unit Price	Extended Price
Custom PCB	2	$163.20	$326.40
Futaba TP-FSM RF Frequency Synthesizer Module	1	$99.98	$99.98
Futaba R319 DPS 9-Channel Receiver Module	1	$189.99	$189.99
Hobbico 72MHz Frequency Checker	2	$28.99	$57.98
Futaba 4.8V Rechargeable Battery Pack	1	$34.99	$34.99
Voltmeter	1	$17.99	$17.99
Relay SSR 1A Mini-SIP	2	$13.43	$26.86
IC Quad Bilateral Switch 14-DIP	8	$0.29	$2.32
IC Quad Voltage Comparator 14-DIP	8	$0.57	$4.56
Multiplex Easy Star RC Airplane Fuselage	1	$28.49	$28.49
FX-2 Connector	1	$8.63	$8.63
Male Header 6 pin 2 row	2	$1.23	$2.46
Male Header 40 pin 2 row	2	$5.43	$10.86
Female Header 40 pin 2 row	1	$6.43	$6.43
Male Header 20 pin 2 row	6	$3.68	$22.08
Female Header 20 pin 2 row	6	$4.68	$28.08
Total			$868.10

Figure 29 - Bill of Materials

Team Member Contributions

Travis Dean

My responsibility was primarily with developing and integrating an algorithm to have the GUI automatically remap the channels. My job also included modifications to the GUI as needed to integrate this functionality. The GUI changes were simply adding a button to start the algorithm after the system is set to receive channel data.

The algorithm required much more thought. The idea behind the algorithm is that the system will take a steady state snapshot of the UAV and store the data locally. Then, a small adjustment is made to one channel, enough to cause the plane to move out of the steady state. The system looks at the SilenTrack data after changing a channel, and determines if the UAV moved out of the steady state. If it did, then the system knows that the channel it changed correlates to the change in the aircraft. For instance, if the plane increased in height, then the changed channel is associated with the elevator. After each change, the UAV is returned to its steady state so as to not interfere with the next change.

The algorithm work also required me to coordinate with Hushnak on the Nexys communication protocols to ensure that the GUI could switch between sending channel remap data and actual channel servo data. We even added a valid bit to confirm to the Nexys board when data on the line was valid for reading.

In addition to work on the algorithm, I assisted Ashley in the SilenTrack communication. Most of this help was during testing and integration phases to integrate SilenTrack with the GUI and ensure proper communication with the SilenTrack system.

Hushnak Singh

My responsibilities for this project were the takeover firmware, takeover software as well as be the Team Manager. As team manager I was responsible for coordinating meeting times for our weekly meetings as well as preparing the slides for our progress. I assisted on the development of the algorithm for the channel association algorithm, helped troubleshoot issues in the hardware redesign and helped modify COTS parts for integration.

On the takeover firmware, I first analyzed the signals coming from the Futaba transmitter, looking at the signal times and arrangement for control of the eight channels. I measured these on an oscilloscope and designed a VHDL module to create signals with the least data input possible. I reduced the signal construction from the old remap module's 12 bit to an 8 bit data system. The benefit is that eight channels only take 8 registers to transmit full control of a UAV instead of 14 channels like the 9 channel FPGA to PC read uses. This is done by building the minimum pulse times into the VHDL and using the full range of the 8 bit integer to control the pulse from the minimum to maximum durations. The tradeoff of using only 8 bits is that the precision is changed to 4µs instead of 1µs for the pulse width time. When compared to the pulse times of 600µs to 1.5µs, this is not a significant disadvantage as most servos will not respond to pulse changes of 4µs anyways.

For the takeover software I modified the existing Microblaze code to allow switching between remap data and Nexys generated data for the Buddy Box connection, and to read the data from the USB registers and pass it along to the Buddy Box Module to generate data for all 8 channels. I modified the frequency channel changing functions on the FPGA and on the PC side to have the waits on the PC side instead of on the FPGA because they can be done with the Sleep function on the PC where you can have feedback that you are in sleep, but they have to be done with useless executions of loops on the FPGA.

Since we are using 9 registers for two functions each, I also modified and tested a data read mode system and a correct data validation system. Because the USB IP core can only do one way communication on each register, the PC is assigned the responsibility of sending the right information, and the Nexys board will process it all based on the Mode/Power register. The two functions most affected by this are the Remap and Send_Signal functions. These both read

from a series of registers, so they check for the proper data transmission mode as well as making sure that the data is valid before they start reading anything from the registers. The Tx and Rx channel sets don't use the validation because they only read from one register, but they both check for the right mode so they don't accidentally try to set the transmit or receive channel to a value of a signal on channel 7 or 8.

I also modified and created a lot of code to test all the functionality of these modules as well as the rest of the firmware with the new software. I developed a console application with a text interface to test all the functions of the Nexys_Comm file and all the functions in the scurvy.c file based on the values in the USB registers.

Ashley Wager

My primary responsibility was integrating the Pirates GUI with Raytheon's SilenTrack by implementing the necessary changes. This included spawning a new thread in the program that connects to SilenTrack and continually receiving and parsing the track packets.

The SilenTrack thread uses the packets it receives to continuous scan for an Ariel Vehicle. Once it has found one of these the thread saves the ID of this track and begins to check the SilenTrack for this track only. Every time the SilenTrack finds the track in the packet it is looking for it parses that data and copies over to our defined structure to allow Travis's algorithm in the main thread to access the track data easily. This caused a shared data problem with the algorithm in the main thread. This problem was solved by enclosing every access to the data structure between a wait and a release of a mutex.

In addition to integrating the GUI with SilenTrack I also helped with the overall functionality of the GUI. Most of this help consisted of fixing any errors that were encountered and cleaning up disorderly code.

Matt Woolridge

My responsibilities for this project included hardware development, budget and parts management, and contribution to overall system design, specifications, and testing.

My immediate contribution to this project was being actively involved in the initial development of the new system features, requirements, and specifications. As a team member I proposed ideas of how the system should work and what would be needed to achieve that. Throughout the project I continued to provide input on system design changes.

My contributions in hardware development included the layout, ordering, and assembly of a custom PCB to replace the original breadboard implementation of the frequency scanner interface circuits, the receiver channel programming circuits, and the frequency synthesizer programming circuits. I also redesigned the original Pirate system to run on a single rechargeable supply voltage. This modification eliminates the cost of replacing expensive alkaline batteries and simplifies the overall power distribution portion of the system. To extend the operating time of the system when running on battery power I added a bus type connection that allows multiple rechargeable battery packs to be connected in parallel to the system board. Also, to allow the user to monitor the voltage of the batteries I integrated a small LED display voltmeter that can be plugged into the battery bus. If the user notices the voltage is running low they can swap out one of the battery packs without losing power to the system. I was also responsible for modifying all off the shelf items that were integrated into the system. I put a lot of effort into making the system as durable and reliable as possible. It became apparent very early that loose and broken connections between the numerous components was creating a troubleshooting nightmare so much care was taken to make all connectors and cables as rugged as possible. Besides reducing the troubleshooting time required on our part, these improvements will allow future teams to focus on further system development and not system maintenance. I was also responsible for system hardware documentation. This included creating detailed and accurate circuit diagrams and wiring diagrams. Good documentation resources are key to ensuring future teams can understand how the system works and to facilitate future modifications. The accompanying user's manual

was updated to reflect all changes made to the system and contains detailed interconnect diagrams, wiring diagrams, pin outs, and an illustrated parts breakdown.
As parts and budget manager I was responsible for submitting parts requests, ordering parts, and keeping track of our remaining budget. These actions were coordinated through the Cal Poly Project Based Learning manager and our Raytheon project advisor.
Through all phases of the project I performed or assisted in testing and verification of the system and its numerous sub components.

APPENDICES

Get Pirates program to work in VS

The following four downloads are required for the Pirates GUI program to operate properly. Downloading and installing the correct version is important as newer versions of the software will not work.

Microsoft Visual Studio 2008 SP1
http://www.microsoft.com/downloads/details.aspx?FamilyId=FBEE1648-7106-44A7-9649-6D9F6D58056E&displaylang=en

Microsoft Chart control for ms.net framework 3.5
http://www.microsoft.com/downloads/details.aspx?FamilyId=130F7986-BF49-4FE5-9CA8-910AE6EA442C&displaylang=en

Microsoft chart controls add-on for Microsoft Visual Studio 2008
http://www.microsoft.com/downloads/details.aspx?FamilyId=1D69CE13-E1E5-4315-825C-F14D33A303E9&displaylang=en

Digilent adept 1.1 – **Make sure to download version 1.1**, not the newer version
http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT

Team Member Contributions

Name	Roles and Responsibilities	
Travis Dean	Software Developer	Implemented Pirating Algorithm Cleaned up old GUI Integrated SilenTrack thread Modified main thread to use modified firmware Made new changes to Gui
Hushnak Singh	Team Manager Firmware Developer	Coordinated group meetings and Teleconferences Designed and implemented UAV control signals on Nexys Board Wrote software to adjust control signals from PC Wrote software to test all the hardware and new firmware Implemented shared data registers for different data types Cleaned up power cycling for better control from the PC
Ashley Wager	Software Developer	Designed Silentrack Interface Designed data exchange between SilenTrack and Pirates Resolved race condition in data sharing Assisted in data validation for shared FPGA USB registers
Matt Woolridge	Parts Acquisition Hardware Developer	Redesigned hardware to work on one battery Combined all the circuitry on one board Assembled and tested the PCB and modified COTS parts Improved and corrected the circuit diagrams from last year Ordered and Procured tools and parts

System Assembly

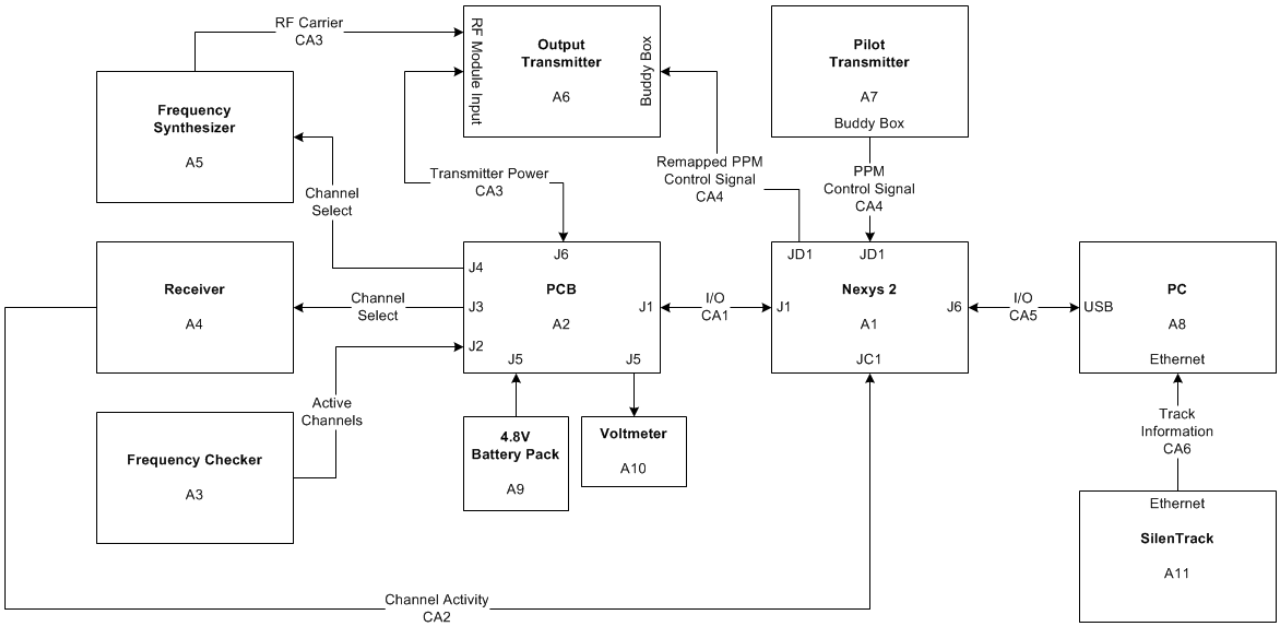

Figure 30 - System Interconnect Diagram

Assembly	Name	Description
A1	Nexys 2	Interfaces Pirate GUI with RF modules
A2	PCB	Performs level shifting and channel programming interface between COT modules and Nexys 2
A3	Frequency Checker	Monitors 72 MHz band for active channels
A4	Receiver	Receives servo control signals being sent to UAV
A5	Frequency Synthesizer	Generates appropriate RF carrier frequency
A6	Output Transmitter	Converts pilot transmitter or system generated PPM control signals and transmits them to UAV
A7	Pilot Transmitter	Radio controller for user to control UAV
A8	PC	Interfaces with SilenTrack and runs Pirate GUI
A9	4.8V Battery Pack	Rechargable battery pack to power Pirate system
A10	Voltmeter	Plugin LED Voltmeter with 9V battery
A11	SilenTrack	Raytheon 3D Video Tracking System
CA1	Nexys 2 / PCB Interface Cable	40 Conductor Ribbon Cable
CA2	Receiver Output Cable	Custom Cable
CA3	Frequency Synthesizer Output Cable	Custom Cable
CA4	Buddy Box Cable	Buddy Box Cable
CA5	Nexys 2 / PC Interface Cable	USB to mini USB Cable
CA6	PC / SilenTrack Interface Cable	CAT5 Network Cable

Figure 31 - System Assembly Description

PCB Schematic

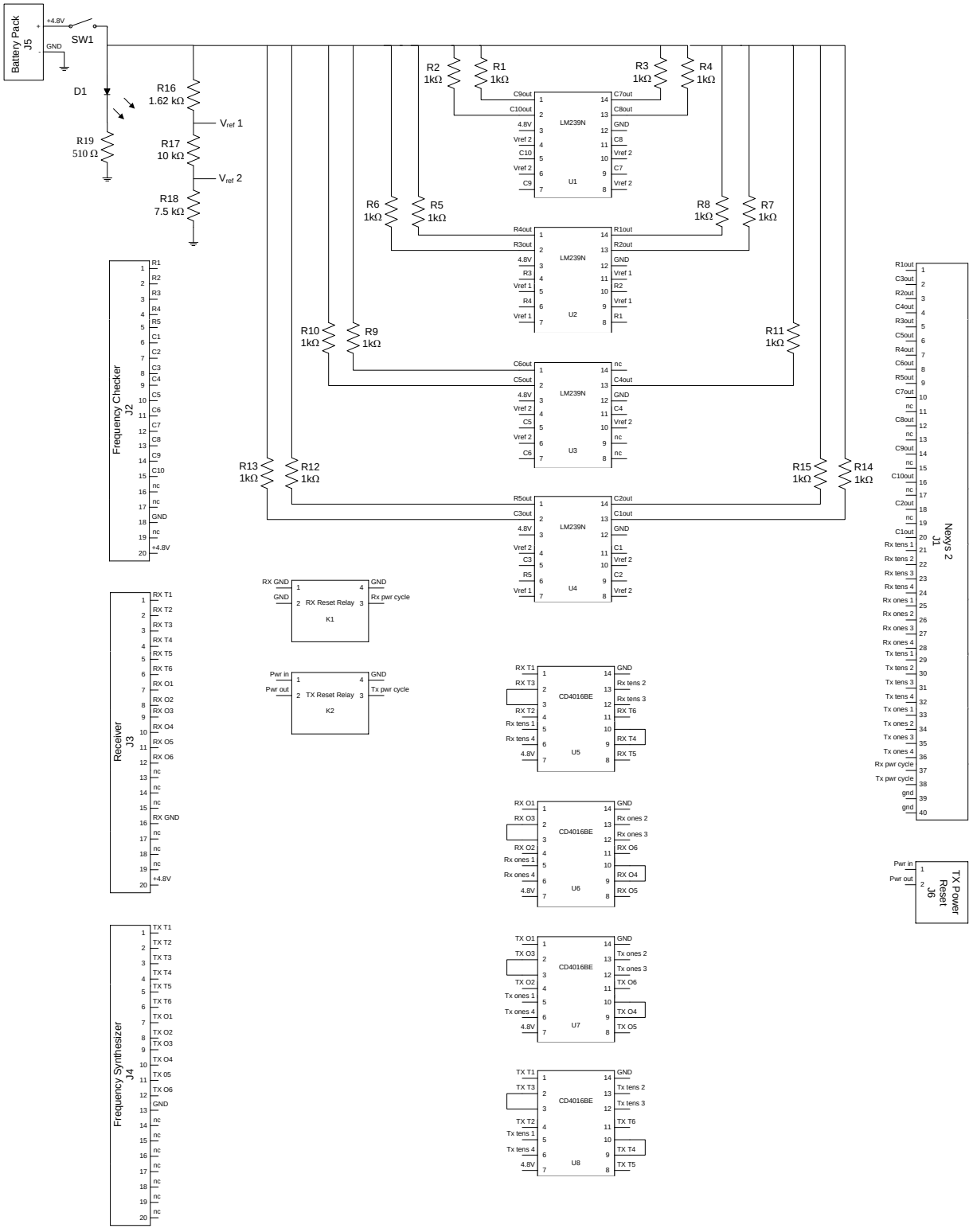

Figure 32 - Pirate PCB Schematic

Mode/Power Register Explained

CS_PWR Register Usage								
bit	7	6	5	4	3	2	1	0
usage	Source Select	Reg 7-12	Tx/CH7	Rx/CH8	Valid data	unused	Tx Power	Rx Power
1	Nexys generated	Channel Data	CH7	CH8	Valid		on	on
0	Remap Buddy Box input	Remap Data	TX channel	Rx channel	Not valid		off	off

Bit 0 - 0x01 1=Receiver powered on
Bit 1 - 0x02 1=Transmitter powered on
Bit 2 - 0x04 Not used
Bit 3 - 0x08 1=Data is valid to be read (Does not include validation for RXON or TXON)
Bit 4 - 0x10 1=Ch8 data, 0=RX_Channel
Bit 5 - 0x20 1=Ch7 data, 0=TX_Channel
Bit 6 - 0x40 1=Sending Ch 1-6 Direct Data, 0=Sending Remap/invert data
Bit 7 - 0x80 1=Transmit from Channel Data, 0=Transmit from Remap data