WIRELESS USB

A Design Project Report
Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)

by
Sean Jason Keller
Project Advisor: Dr. Bruce R. Land
Degree Date: January 2004

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Wireless USB

Author: Sean J. Keller

Abstract: The Universal Serial Bus (USB) v1.1 has a range limited by the electrical properties of the connecting cables and strict end to end maximum signal delays. By stretching the USB protocol, these strict end to end maximum signal delays can be circumvented, allowing USB data to be transmitted over a wireless connection. This allows for greatly increased range and new application of devices. Prototyping evidence suggests that a low-power, low-cost, and compact Wireless USB System can be created.

Report Approved by

Project Advisor: _____ Date: _____

Executive Summary

The USB specification allows for USB cable no longer then 5 meters long. Yet, in many circumstances, placing peripheral more then 5 meters away from a PC is desirable. With additional USB hardware, it is possible to place a peripheral up to 30 meters away from a computer's root hub, but this is a difficult and expensive solution to implement with no possibilities for further expansion.

The best solution to this problem is to allow USB device connections without requiring a physical link, making USB a wireless paradigm. This report explains how to implement functionally transparent wireless USB, a USB cable replacement. The solution requires a detailed understanding of the USB at a physical and protocol level, and it requires a great deal of digital logic to implement.

The system requires two custom built PCBs, each controlled by a small Atmel ATMega8L microcontroller. The protocol logic engine, the USB serial interface engine, and a communication channel are programmed into a Cypress Detla39K CPLD. The wireless link is implemented with a 2.4GHz high-speed transceiver pair, Micro Linear's ML2724 FSK half-duplex transceiver. Various resistors, capacitors, transceivers, connectors, headers, and switches glue the system together.

The system is almost fully functional despite of an overwhelming number of complications and design hurdles. The PCBs and all components work correctly. The RS232 debugging interface and the bus sharing logic within the CPLD work as expected. USB clock and data recovery is functioning. The RF link is fully functional, and error free data transmission, acknowledgement, re-transmission, and reception work correctly. Finally, device connection, disconnection, and speed detection work correctly. The protocol handling state machines are not fully functional.

I) Introduction

On September 23, 1998, four giants in the personal computer arena: Compaq, Intel, Microsoft, and NEC published the final draft of the Universal Serial Bus Revision 1.1 Specification. Since its inception, USB has become the standard peripheral interface in all personal computers. It is a robust and practical interface with many strengths and few weaknesses. However, it is inherently limited by the fact that it is a bus and requires a physical connection via a USB cable between the device and root hub. Cables are bulky, difficult to manage, and limited in length.

The electrical constraints of the specification place an upper bound of 5 meters on the length of any USB cable.[1] Yet, in many circumstances, placing peripheral more then 5 meters away from a PC is desirable. By using 5 hubs and six 5 meter cables, it is possible to place a device 30 meters away from a root hub. Not only is this physically cumbersome and expensive to implement, but it is still limiting with respect to moving a peripheral away from a computer. Even if the device does not need to be farther then 30 meters from the root hub, it may be very difficult to chain numerous hubs and cables together to reach it. Perhaps for example, the device is best situated a few rooms away from the computer to which it needs to be attached; connecting it with a series of cables and hubs may be impossible, so a wireless solution is necessary.

[1] Anderson, Don. Universal Serial Bus System Architecture. Reading: Addison-Wesley Developers Press, 1997. pp. 56.

II) Design Problem and Requirements

The primary goal of this project is to design a system which allows the logical attachment of any USB v1.1 device to any USB v1.1 root hub without a physical connection between the two. This wireless USB system must act as a cable replacement; neither the host nor the device will require modification to function correctly in the wireless environment. These goals initially lead to the following requirements:

- The system must perform its described function (it must allow the logical attachment of any USB v1.1 device to any USB v1.1 hub without a physical connection between the two
- The system will be USB v1.1 compliant
- Both full-speed and low-speed devices should be supported
- The system should be physically compact, portable, and able to operate off of batteries
- The strict timing and data rate requirements in the USB v1.1 specification must be met
- The system should operate over a range of at least 30 meters without line of sight
- The design should be cost-effective and commercially viable

After thorough research and some project development the requirements needed modification. The justification and reasoning for the alterations is found in a later section. This is the final set of requirements:

- The system must perform its described function (it must allow the logical attachment of any low-speed USB v1.1 device to any USB v1.1 root-hub without a physical connection between the two
- Only low-speed devices will be supported
- The strict timing and data rate requirements in the USB v1.1 specification must be adhered to at the physical level, but they may be circumvented at the protocol level
- The system should operate over a range of at least 30 meters without line of sight

- The design should pave the way for a cost-effective and commercially viable solution

III) The Range of Solutions

At first glance, the most important design decision in this project appears to be how to implement the wireless link. Should the devices communicate with a line of sight infrared or laser signal, or should they use a radio frequency band? What are the bandwidth and latency requirements of the link? Does the carrier signal have to be within a certain frequency range? All of these questions are important, but the crux of any solution for this system is: Should it and can it abide by the strict timing requirements of the USB, or should it 'trick' the USB and follow the timing rules at a physical level while stretching the protocol to buy time at a transaction level. This project uses the latter approach, and reasoning as to why the former approach may not be possible can be found in the design section of this report.

Keeping the design requirements in mind, wireless link selection is fairly straight forward. Use of any light based wireless transmission system will require a line of sight, and thus cannot be used. Using a radio frequency greater the 5 GHz will most likely result in a range of less then 30 meters, and using a frequency less then 900 MHz will almost certainly result in a data rate too slow to be practical for USB application. This project uses the unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band for the wireless link. It provides an environment with the possibility of having a reasonable mixture of high data rate and long transmission range.

Using a PCB (Printer Circuit Board) is the most practical way to design a system with a large number of interconnects between a numerous of devices. Using a breadboard or solder-board for a project of this size is unrealistic and unreliable. Routing wires, placing chips, and mounting components which are only available in SMD packages on either a breadboard or solder-board is tedious and error prone work. On the other hand, with PCB design software components can be logically connected and disconnected without regard to physical placement or attachment. Also, most modern PCB design software packages use sophisticated algorithms to automatically route the physical layout.

Using low-power 3.3V devices is pretty much a necessity if battery power is ever to be considered as an option. An alternate design using 5.0V components would also

work but would almost certainly consume more power during both active operation and in a suspended state.

Using Surface mount devices is necessary for a design to be cost effective in mass production. In production, SMDs are robotically placed and are generally soldered in large batches using infrared, thermal, or wave techniques. They also tend to be much smaller then their through-holed counterparts and aid in keeping PCBs smaller and cheaper. Using all through holed components is certainly not a viable solution.

IV) A USB Primer

IV.1) Introduction

This Primer is provided as an extremely brief introduction to USB. It contains a broad overview of the environment and some detailed information relevant to this project. Other specifics, especially timing constraints and protocol definitions will be described as needed throughout the report. A reader familiar with any modern serial signaling environment including USB should find the content in this primer sufficient for following this report but may wish to read sections 5 to 5.10, 7, and 8 in the USB v1.1 specification. An unfamiliar reader should read through the USB v1.1 specification sections 1 to 5.10 and 7 to 9.2. Any reader may find sections 10 and 11 useful for reference.

IV.2) Background

The Universal Serial Bus v1.1 was developed to overcome many of the obstacles involved with adding new peripherals to a PC. Prior to USB, the end-user needed some technical know-how to setup and install most peripherals. There was a mixture of different interfaces that were too slow, too expensive, or too complicated for most people to use. The personal computer industry was booming, but it was difficult for most people to attach new devices to their personal computers. USB solved these problems with the following features:

- Only simple low-cost hardware is needed to support the bus
- 127 devices can be attached to the bus
- The physical connector is small, simple to plug in and nearly impossible to plug in incorrectly
- Low-speed 1.5Mbps and Full-speed 12Mbps devices are supported
- No new system resources are used when devices are attached
- USB devices are hot pluggable and can support plug and play
- Error detection and recovery are available at the hardware level

- Four different transfer types are available to support an extremely broad class of devices

These key features and many others made USB the primary PC peripheral interface.

IV.3) System Architecture

The USB system is a mixture of hardware and software following a master and slave model. The master is the host system and the slaves are the USB devices. All transactions begin with the host, but devices are permitted to signal a wakeup event if they are currently in a low-power suspended state and need to resume operation. From the USB v1.1 specification, the communication flow is as follows:

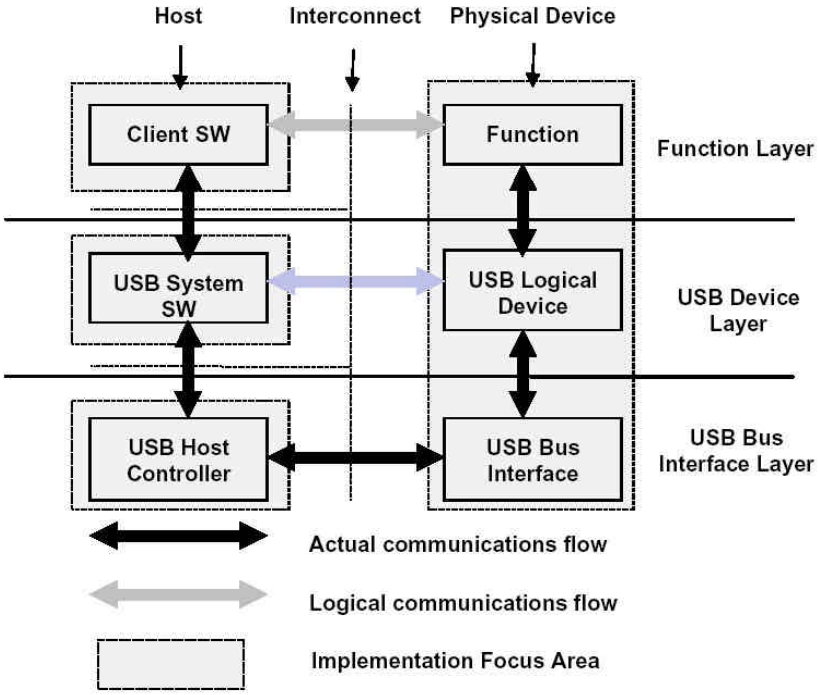


Figure 1: USB Communication Flow[2]

"The USB client initiates a transfer when it calls USB system software and requests a transfer. USB client drivers supply a memory buffer used to store data when transferring data to or from the USB device. Each transfer between a given register (or endpoint) within a USB device and the client driver occurs via a communication pipe that USB

[2] Compaq, Intel, Microsoft, NEC. Universal Serial Bus Specification v1.1. 1998. [Online Book]. URL: http://www.usb.org/developers/docs/usbspec.zip. pp. 26.

system software establishes during device configuration. USB system software splits the client's requests into individual transactions that are consistent with the bus bandwidth requirements of the device and the USB protocol mechanism. The requests are passed to the USB Host Controller Driver which in turn schedules transactions to be performed over the USB. The host controller performs the transactions based on the contents of a transfer descriptor that is built by the HCD….Each transaction results in data being transferred either from the client buffer to the USB device or from the device to the buffer…When the entire transfer has completed, USB system software notifies the client driver."[3] The host controller is physically connected to the root hub, which consists of a hub-controller and a repeater; thus, every USB transaction originates at the root hub.[4]

Four classes of transactions are available on USB: control, interrupt, isochronous, and bulk. Control transfers are required to setup and configure devices immediately after attachment. Control transactions are delivered with a best effort approach and are not guaranteed bus bandwidth, but data integrity and successful delivery is ensured with Cyclic Redundancy Checks (CRCs) and retransmission of lost or bad data. Interrupt transfers are used for data polling at a rate of 1ms to 255ms; this is ideal for peripherals such as a keyboard or mouse. They have a guaranteed maximum service period and retry failed transfers at the next period. Data integrity and successful delivery are ensured with CRCs and retransmission of lost or bad data. Isochronous transfers are used for constant-rate error tolerant delivery such as for audio or video I/O. They have guaranteed bandwidth with bounded latency and data integrity is ensured with CRCs, but failed data deliveries are not retried. Finally, bulk transfers are used for transferring large chunks of data on an available bandwidth basis, such as for printers or external storage devices. Data integrity and delivery are ensured with CRCs and retransmission of lost or corrupt packets, but neither bandwidth nor latency is guaranteed. All of these transfer types have specific timing constraints and maxim data-payload sizes that are negotiated at the time of device attachment.[5]

---

[3] Anderson, Don. Universal Serial Bus System Architecture. Reading: Addison-Wesley Developers Press, 1997. pp. 39.

[4] Anderson, Don. Universal Serial Bus System Architecture. Reading: Addison-Wesley Developers Press, 1997. pp. 33.

[5] Compaq, Intel, Microsoft, NEC. Universal Serial Bus Specification v1.1. 1998. [Online Book]. URL: http://www.usb.org/developers/docs/usbspec.zip. pp. 31-49.

The USB operates at two data rates, full-speed at 12Mbps and low-speed at 1.5Mbps. Hubs are responsible for controlling this dual speed environment. Low-speed devices are limited in both data rate and available transfer types. Unlike full-speed devices, low-speed devices cannot perform isochronous or bulk transfers. They are limited to performing only control and interrupt transfers.

IV.4) Signaling Environment

The USB has no dedicated clock signal. Rather, data is transmitted differentially with an implicit clock. A SYNC (Synchronization) field which contains an SOP (Start of Packet) delimiter is transmitted before any data to allow the receiver of the data to perform clock recovery. This is generally accomplished by using a DPLL (Digital Phase-Locked Loop) clocked at four times the incoming data rate. The encoding scheme used by the USB ensures that this DPLL running at 4x can obtain a phase lock before the SOP and can maintain lock throughout the entire transmission until an EOP (End of Packet) delimiter is received out of band.

The USB cable contains 4 signals: Vcc, Ground, D+, and D-. Vcc to Ground provides 5V DC power at a maximum current of 500mA. The differential signals swing between 0V and 3.3V. Hubs pull down the differential lines on downstream ports to ground via a 15KO resistor and downstream devices pull up either the D+ or the D- line to 3.3V across a 1.5KO resistor. Low-speed devices pull up D- while full-speed devices pull up D+. When no device is connected to the downstream port of a hub, due to the pull-downs, both D+ and D- rest at 0V; this bus state is called SE0 (Single Ended Zero) and is used for out of band signaling and disconnect detection. When the device is attached and either D+ or D- is pulled up but not driven, the bus is idling and in a data J state or idle state. Data J corresponds to differential 0 for low-speed devices and differential 1 for full-speed devices. The opposite differential signal is called Data K and is differential 1 for low-speed devices and differential 0 for full-speed devices. All USB protocol is defined in terms of Data J, K and SE0. In a mixed speed environment, hubs are responsible for logically inverting low-speed data before passing it into the full-speed domain.

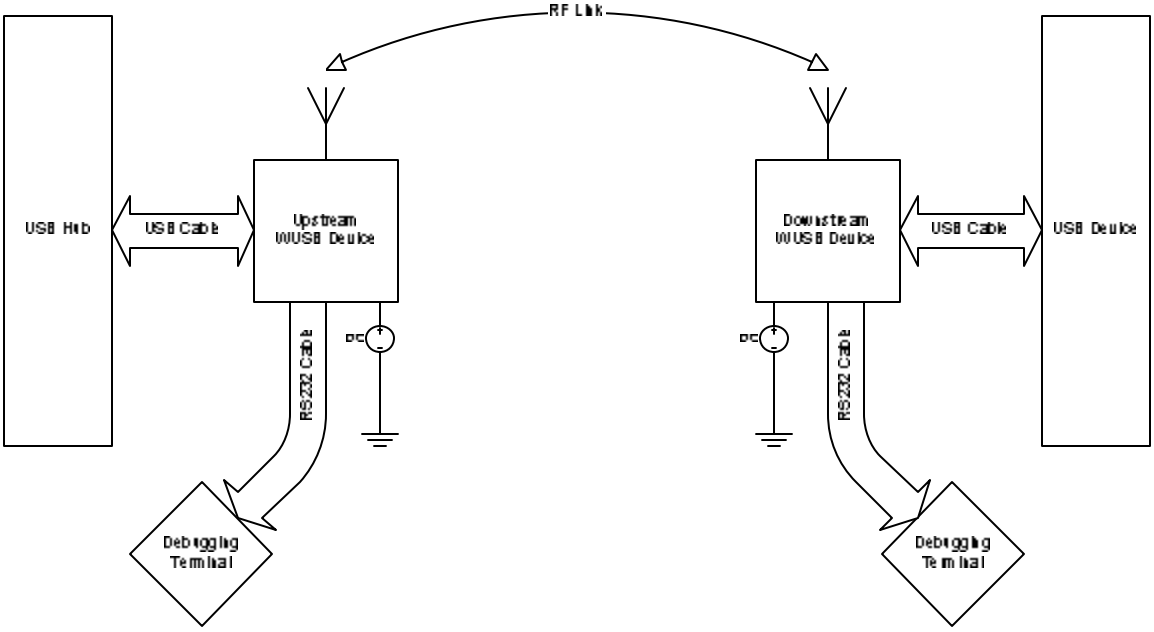
Logical data is bit stuffed and then encoded using NRZI (Non Return to Zero Invert) encoding. Under NRZI encoding, a logical 0 is represented by a transition from J to K or K to J on the bus, while a 1 is represented by no change. Bit stuffing is used to ensure the DPLL clock recovery logic maintains a lock. Before the Data is NRZI encoded, a logical zero is inserted after six consecutive ones. This ensures that the maximum run length of J or K will be 7 bit-times.

Every USB transfer begins with a SYNC pulse train. It consists of 3 periods of 1 K bit-time followed by one J bit-time and is terminated with 2 bit-times of K. The 2 bit-times of K are used by hardware to detect a SOP, while the 3 prior pulses are used to obtain a phase-lock to the incoming data. Similarly every USB transfer ends with an EOP. It consists of 2 bit-times of SE0 followed by 1 bit-time of J.

V) Design and Implementation

V.1) the Premise

The underlying premise is that the system must act as a USB cable replacement and logically follow the block diagram displayed in Figure 2.



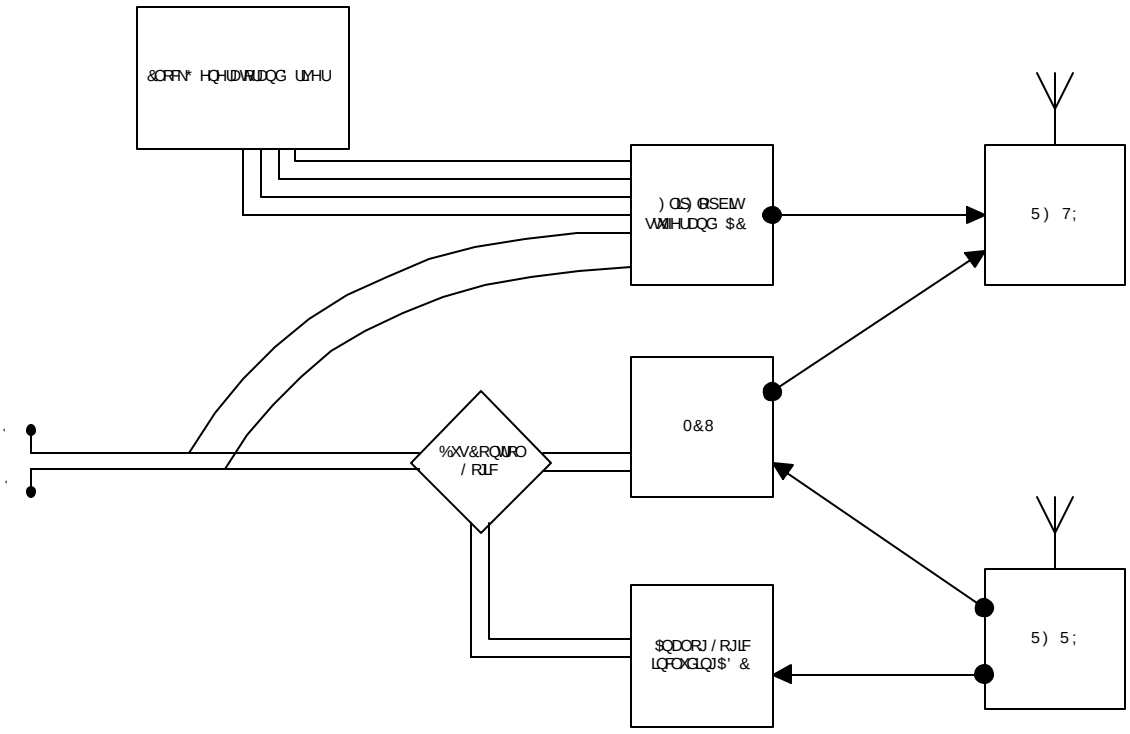


Figure 3: First approach to designing a Wireless USB System

The wireless link consisted of two simple analog 2.4 GHz NTSC video transmitters and receivers and a side channel used for control signaling and maintaining state between the two devices. The incoming USB data was sampled serially at 12 MHz, regardless of the device speed, and through the use of flip flops and out of phase clocks was converted to a 3 MHz parallel data stream which could be amplitude modulated and transmitted over the RF link. Incoming RF data was sent through an ADC and clocked at 12 MHz for output onto the USB. The bus direction was controlled by a microcontroller attached to tri-states, and logical connect and disconnect of a device could also be controlled using the MCU. I deemed the 12 MHz sampling rate sufficient for both 12Mbps full-speed device and 1.5Mbps low-speed devices. On the USB, the full-speed clock runs at 12 MHz, and the low-speed clock runs at 1.5 MHz. 12 MHz is a multiple of 1.5 MHz, so using a 12 MHz sampling scheme would work for both device classes.

After more research and thought, I realized that not only did this design not work, but the approach was fundamentally incorrect. First and foremost, the flip-flops were powered by an independent 12 MHz clock with no phase relationship between it and the USB's embedded data clock. It would be possible for a USB data edge to fall directly on

the rising edge of the 12 MHz sample clock resulting in an unreliable sample. This could be fixed by using a phase-locked loop to maintain a phase relationship, but this would require stepping a bit outside of the electrical domain and into the USB protocol domain. The SYNC would be consumed while obtaining a phase-lock, so buffering of data and SYNC regeneration would be necessary.

The simplest and least protocol dependant solution was to use an analog phase-locked loop to lock to the USB, but after careful research and development this turned out to be impossible. An analog phase-locked loop works by using a phase detector to control a voltage controlled oscillator to phase-lock a reference frequency to the incoming data signal. When the incoming data signal and the output of the VCO differ in phase, an electrical pulse with a width proportional to the phase difference is filtered and passed back to the VCO, forming a simple feedback system. By tuning the filter, it is possible to reliably phase-lock to the incoming data signal. Unfortunately, how quickly the loop locks and how long it stays locked are inversely related. This relationship is easy to derive, but outside of the scope of this report. Using an analog PLL for USB clock recovery would require that the loop lock in 6 bit-times, to catch the SOP, but stay locked for up to 7 bit-times without edge transitions, the maximum run-length of J or K on the USB. This is a fundamental impossibility.

The only feasible solution to this clock recovery problem is to use a digital phase-locked loop. The design of such requires a 4 times over-sampling clock and a fairly straight forward state machine. The DPLL doesn't maintain nearly as accurate a phase relationship as the analog PLL would, but it does guarantee the generation of a derived data sample clock with valid USB data on every rising edge. It turns out that this entity is required for any digital USB interface to function correctly.

The next flaw with this initial design was the amplitude modulated RF link. The bandwidth of an NTSC video signal is just under 6 MHz, so at first glance the 3 MHz amplitude modulated signal should transmit correctly over this link. Clearly a 3 MHz true square wave can be recovered by using only the fundamental 3 MHz sine wave which will fit into the 6 MHz of bandwidth. Unfortunately, the 3 MHz signal in this system requires more then 3 MHz of bandwidth, as it is constructed with various analog levels corresponding to different logical data. In the time domain, all of the possible

analog level transitions create spikes at various harmonics in the frequency domain. The fundamental 3 MHz signal can still be transmitted and received, but all of the necessary harmonics created from various level transitions fall well outside of this 6 MHz window.

It soon became clear that my initial design had some underlying flaw. I was trying to stay within the time constraints of the bus and not interrupt communication at the protocol level. This just does not work. A transaction will timeout if no response is received within 16 to 18 bit-times from when the EOP completes.[6] If the upstream Wireless USB Device (WUSBD) is connected to the last hub in a 5 hub chain then the delay from that hub upstream may be as long as 15 bit-times, leaving 1 bit-time for the wireless transaction to generate a response. This is impossible, due to the necessary SYNC consumption and data buffering for the DPLL to recover the clock. The only possible solution is to require that the upstream WUSBD be connected directly to the root hub, providing a 16 bit-time transmission and response window. Still, the upstream device would have to transmit data in real time and have a response from the downstream side in less then 16 bit-times from when the upstream hub signaled an EOP. This requires extremely high-speed data processing and a high-speed full-duplex wireless link with little to no transmission overhead. It also provides no possibility for RF data acknowledgment and no possibility for RF data integrity checking with a CRC or similar check. All in all, this solution cannot work reliably, and probably is not realizable even if reliability is not a concern. Finally, this solution cannot scale to higher bus rates such as for high-speed transmissions available in USB 2.0.

[6] Compaq, Intel, Microsoft, NEC. Universal Serial Bus Specification v1.1. 1998. [Online Book]. URL: http://www.usb.org/developers/docs/usbspec.zip. pp. 134.

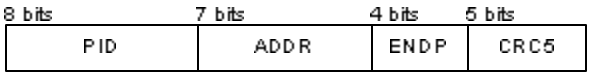
V.3) the Working Model

V.3.a) the Critical Timing Algorithm

The only way to build a robust wireless USB system is break out of the 16 bit-time bus timeout limitation, while still maintaining logical transparency to both the root hub and the USB device. This was a major design hurdle, and there was no obvious solution. I realized that this project could not be completed unless I resolved this issue. Fortunately, I thought of a solution. Unfortunately the solution does not work for isochronous transfers. And while I suspect there is a solution for isochronous transfers by using a fairly complicated buffering and flow control scheme, I was not and have not been able to design a working model. So, I made the design decision to only support low-speed USB devices and thus eliminated the need to support isochronous transfers.

The solution is very logical and easy to explain, but complicated to build and difficult to synthesize in hardware, as it requires extensive protocol manipulation. However, in order to understand the solution, it is important to understand some of the basics of the USB transaction protocol. First, low-speed devices only support control and interrupt transfers. Second, every packet is packaged between a SYNC and EOP.

Every interrupt transfer begins with the host transmitting an IN or an OUT token onto the bus (detailed in Figure 4).



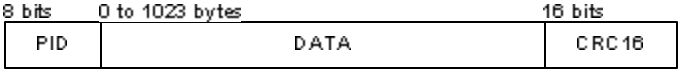
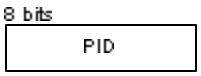


Figure 5: Data Packet Format

For low-speed devices, the data field is a maximum of 8 bytes, and the CRC16 covers the entire DATA field. Upon error-free receipt of the DATA packet, the function will return a Handshake Packet to the Host (detailed in Figure 6).



Avoiding the 16 bit-time timeout with an Interrupt OUT transactions is the simplest case. Upon detecting an Interrupt OUT transaction the upstream WUSBD emulates a USB function stalling for flow control reasons. The upstream WUSBD begins buffering and passing the data wirelessly to the downstream WUSBD and within 16 bit-times of receiving the DATA Packet's EOP, the upstream WUSBD generates a NAK handshake and sends it to the host. All subsequent transactions generated by the host are NAKed in this same manner and are not sent to the downstream WUSBD. When the downstream WUSBD receives the transmitted data, it transmits the same OUT Token and DATA Packet to the function, and wirelessly returns the function's handshake response. The upstream WUSBD buffers this response and upon receipt of another Interrupt OUT transaction from the host responds to the host with the buffered function response. To create a robust system, timeouts indicating an RF link failure are used to bring both the downstream and upstream busses back to idle.

Avoiding bus timeout with Interrupt IN transactions is slightly more complicated. Upon detecting an Interrupt OUT transaction the upstream WUSBD emulates a USB function with no data available for the Host. The upstream WUSBD begins buffering and passing the IN token wirelessly to the downstream WUSBD and within 16 bit-times of receiving the IN token's EOP, the upstream WUSBD generates a NAK handshake and sends it to the host. This retires the transaction, but without causing a bus error. The Host will simply retry this same transaction at a later time. All subsequent transactions generated by the host are NAKed in this same manner and are not sent to the downstream WUSBD. When the downstream WUSBD receives the transmitted data, it transmits the same IN token to the function, and wirelessly returns the function's DATA, NAK, or STALL response. Within 16 bit-times of the IN token's EOP transmission onto the downstream bus, the downstream WUSBD generates a corrupted ACK and transmits it to the function. The function will respond to the corrupted ACK with a retransmission of the previous response. This response is dropped and corrupted ACKs are transmitted until a response is received from the upstream WUSBD. When the upstream WUSBD receives the DATA sent from the downstream WUSBD, it buffers it and waits until the host generates another Interrupt IN transaction at which point it replies with the DATA in the buffer. The Host's response to this buffered DATA is then wirelessly transmitted to

the downstream WUSBD which buffers it and sends it to the function upon receipt of its initial response. To create a robust system, timeouts indicating an RF link failure are used to bring both the downstream and upstream busses back to idle. The only problem with this solution is if the transaction takes too long, the data being returned from the downstream WUSBD may be stale, as the function may have already buffered in new data for transmission. Conveniently, the fastest low-speed interrupt polling rate is 10ms, which is still a very large window compared to 16 low-speed bit-times. 16 low-speed bit times equal about 10us, so a 10ms window is 3 orders of magnitude longer.

For Control Transfers, the timeout prevention scheme for the Setup phase is tricky, since the function must ACK or timeout the bus. Upon detecting a Setup transaction the upstream WUSBD begins buffering and passing the Setup token and DATA0 token wirelessly to the downstream WUSBD. Within 16 bit-times of receiving the DATA0 token's EOP, the upstream WUSBD generates an ACK handshake and sends it to the host. The Host then continues onto the Data phase or the Status phase. Upon receipt of the Setup transaction, the downstream WUSBD buffers the data and sends it to the function which responds with an ACK or times out. This response is transmitted to the upstream WUSBD. If the optional Data phase is used, the upstream WUSBD replies to all subsequent IN or OUT tokens with a NAK back to the Host. Once the Setup response arrives from the downstream WUSBD, the Data phase can continue if the response is a NAK. If the response is a timeout, the upstream WUSBD stops responding with NAKs and lets the bus timeout. If the Data phase continues, the bus stalling logic follows the same algorithm as the Interrupt IN and OUT transactions. Finally, once the optional Data phase completes, or if there is no Data phase, the Status phase occurs, which also uses the same timeout prevention scheme as the Interrupt IN or OUT transaction depending on the Status stage direction.

V.3.b) the Heart of the System

A great deal of digital logic is necessary to implement a DPLL, buffer control, and the timeout prevention scheme. Some sort of programmable logic is needed to make this possible. I chose a CPLD (Complex Programmable Logic Device) programmable at the RTL (Register Transfer Level) as the heart of this system. An

FPGA (Field Programmable Gate Array) would have also worked very well and in retrospect would have been a better solution. Using a CPLD seemed like a very reasonable solution as they tend to fit complex state machines very well, and they have predictable input to output latencies. On the other hand, FPGAs cannot guarantee timing, and are not built specifically for use with large state machines. I chose to use a CPLD by Cypress semiconductor, because they offer a wide selection of devices, an extremely cost-effective software suite with built in support for coding, synthesizing, and simulating all of their CPLDs. They also offer a low-cost development board and programming dongle compatible with most of their product families. I intended to use a device in one of their low-cost smaller CPLD families, ULTRA3700, but as I neared code completion it became apparent that I had too much logic and too many signals to be able to fit my design to this logic family. This was primarily due to the limitations of the PIM (Programmable Interconnect Matrix) which glues logic together across different macro-cells. My code didn't require too many macro-cells; rather it required too many macro-cell interconnections. An FGPA would have worked better in this situation, but I was already committed to using Cypress devices, so I chose to use a different product family, Delta 39K. The CPLDs in this family are much larger in terms of usable gates and have embedded RAM cells, but they are substantially more expensive then the ULTRA3700 product line and are volatile. Yet, they are extraordinarily versatile. They can guarantee input to output timing of signals even with drastic code changes. Also, even after pin-locking signals, the Delta 39K devices can almost always route regardless of code changes. The volatility issue is easily solved by using a small Lithium Ion battery to power the CPLD in a suspended state when it is no longer being used and powered by a transformer based DC power supply. I realized that there was no need to bother with this as this revision of the wireless USB system would only be practical for design and debugging purposes. A commercially viable version would need to use a low cost FPGA or an ASIC. The schematic for the CPLD is simply the synthesized VHDL (see Appendix (7)).

V.3.c) the RF Link

With a timeout prevention solution in hand, the RF link becomes substantially less critical. I still hoped to someday support full-speed USB, so I wanted a reasonably high-speed RF connection. Even though the timeout prevention scheme applies to full-speed USB and the RF link does not need to be very high bandwidth, full-speed transfers would be noticeably slow with a low bandwidth link. I chose a small SMD 2.4 GHz transceiver by Micro Linear, the ML2724. It has a data rate of up to 1.5Mbps, runs on 3.3V supply and can support a maximum run-length of 4 zeros or ones at 1.5Mbps. It uses DSSS (Direct Sequence Spread Spectrum) modulation to improve range and has in integrated PLL. It supports a low-power suspend state, and it allows for over 40 non-overlapping user selectable channels.

In order to avoid building the necessary and complicated external analog circuitry for this device, I opted to sample a starter kit which included a pair of these transceivers mounted on a very compact 4 layer PCB with all of the external circuitry needed to make them run. These RF daughter boards were attached to a development platform created by Micro Linear. I simply pulled off the RF daughter board and integrated it into my own design.

V.3.d) the Bus Interface

The physical connection to the bus is via standard USB connectors soldered to the PCB. The data connection is through a cheap USB transceiver by Maxim, the MAX3451E. It has an internal 1.5K? that can be enabled and disabled on either the D+ or D- bus. This allows the upstream WUSBD to logically disconnect from the bus while still being physically connected. It can be suspended for low-power operation and has integrated ±15kV ESD protection for the D+ and D- busses. It also has a differential output which is logically equal to D+ minus D-; this is quite convenient as it eliminates extra analog circuitry to perform this task. Yet, the most important reason for using this transceiver is that it is electrically USB v1.1 compliant and eliminates any impedance concerns that would need to be addressed when directly driving the bus form the CPLD.

V.3.e) the MCU

Setting up an RF link, maintaining the link, and initializing the CPLD are accomplished with an Atmel MCU, the ATMega8L. It is a small SMD device requiring a 3.3V DC supply. It contains one USART and many I/O ports. It is a very fast and low power device, and the simple RISC architecture allows for quick coding in C and ASM. All code is available in Appendix (9).

V.3.f) the UARTs

In order to easily debug the CPLD and have a fast communication channel between the CPLD and the MCU, I decided to synthesize a UART in the CPLD. Fortunately, a very nearly working public source UART[7] written in VHDL is available online from opencores.org (see Appendix (8) for VHDL). The UART is very simple, and has a standard data interface conforming to the open source IP core interconnection standard called Wishbone. I wrote a simple Wishbone interface and connected it to a buffer, providing me with an excellent simple UART. I only needed to modify the UART VHDL slightly to make it work with the Delta39K family of CPLDs.

In order to debug the platform, communicate with the RF board, and send data between the MCU and CPLD, I shared the MCU RX and TX lines, the ML2724's DIN and DOUT, and the CPLD UART. The sharing logic is controlled by 2 outputs CNTRL1 and CNTRL2 from the MCU to the CPLD. The bus sharing logic is implemented in the CPLD by using several multiplexers. The sharing scheme allows for connection of the MCU UART to the RS232 transceiver by TI (a copy of the MAX3221), the ML2724's DIN and DOUT, and the UART in the CPLD. It also allows for connection of the UART in the CPLD to the RS232 transceiver for debugging output.

V.3.f) the RF interface

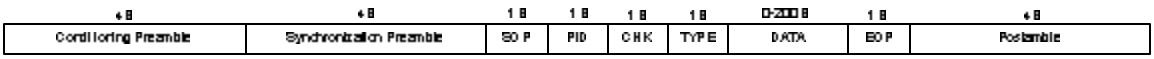
After initialization by the MCU, the ML2724 is fairly simple to interface with. It has a Data Input DIN, a Data Output DOUT, OE (Output Enable), and an analog receive signal strength indicator (RSSI). The MCU interfaces with transceiver via its UART. The UART is designed for this sort of task and has built in clock recovery and data

[7] Carton, Phillip. "MiniUART." 2003. URL: http://www.opencores.org/projects/miniuart2/ (17 Dec. 2003).

encapsulation logic. At 250Kbps, the data rate is lower then the achievable 1.5Mbps, but this is not too important. RSSI is connected to the negative input of a comparator. The positive input of the comparator is connected to the center tap of a variable resistor with Vcc and GND on the other 2 leads. The output of the comparator is connected to a port pin of the MCU. The MCU monitors the comparator and buffers data when the output is high. RSSI is proportional to the log of the receive field strength and ranges from 0.0 to 2.5V. Since the variable resistor sets the voltage level of the negative input of the comparator it also sets the receive threshold. I found the 2.4GHz frequency range to be fairly noisy, so I set the variable resistor to have a center voltage of 0.85V which corresponds to a threshold of about -60dBm.[8] Finally, since field strength decreases with distance, the range of reliable transmission and reception is also determined by this center voltage, and reducing it will increase the range but increase the signal to noise ratio.

V.3.g) the RF protocol

All RF transactions begin with a 4 byte Conditioning Preamble consisting of periodic 1 to 0 transitions. The Conditioning Preamble is required by the ML2724 to calibrate the radio. The next 4 bytes are the Synchronization Preamble which consists of a sequence of zeros and ones which force the UART into proper frame synchronization. The next byte is a Start of Packet (SOP). This is followed by the Header which consists of a Packet ID (PID) byte, a data checksum byte (CHK), and a Packet Type (TYPE) byte. After the header, Manchester encoded data is transmitted, followed by an End of Packet (EOP) byte. Finally, a 4 byte Postamble ramps off the transmission to keep RSSI high long enough to correct for any lead or lag of data with respect to RSSI.



V.3.h) the RF Algorithm

The algorithm works as follows. The upstream MCU acts as the master and the downstream MCU acts as a slave. The upstream MCU always initiates an RF transaction by sending data to the downstream MCU. The downstream MCU must validate the data and respond with an ACK if the data is not corrupted. The upstream MCU will timeout and retransmit the data if an ACK is not received within 200us. If the upstream MCU retransmits nine times unsuccessfully, the transaction is aborted and an error is returned. In order to transmit data from the downstream MCU to the upstream MCU, the upstream MCU sends a packet requesting data, if the downstream MCU receives the packet correctly, it responds with a data packet with data or empty if it has no data to send. If the packet is not received or is corrupted no data is returned and the upstream MCU retransmits the data request after a 200us timeout; if the request is retransmitted nine times unsuccessfully, the transaction is terminated and an error is returned. The downstream MCU waits for an ACK and follows the same scheme regarding timing out and retransmission.

The PID field is very important as it corrects for lost ACKs. The transmitter of a packet increments the PID every time an ACK is received. The receiver of a packet includes the received packet's PID in the ACK. If the ACK is lost, the transmitter will not increment its PID, and it will re-transmit the last packet. The receiver will note the duplicate PID, drop the data and reply with an ACK.

The checksum field over the data is not as robust as a polynomial generator CRC, but it is simple, cheap to calculate and good enough, as all low-speed USB transactions include a CRC.

Manchester encoding[9] of the data is necessary as the maximum run length of zero or one that can be reliably transmitted through the ML2724 is 4. Manchester encoding the data ensures that no more then 2 consecutive ones or zeros will be transmitted. Unfortunately, Manchester encoding the data doubles the data length. Fortunately, the maximum un-encoded data length that I could stuff into the MCU's RAM is about 100 bytes, which is more then enough room to buffer any USB transaction.

[9]Micro Linear. "ML2724SK-01 2.4GHz 1.5Mbps RADIO Code Base." April, 2003.

V.3.i) the Clocks

Although a single clock running at 12Mhz could be used by the CPLD to derive all the necessary clocks for this system, I chose to use an external programmable clock chip by Cypress, the CY39100, a low power, suspendable, and extremely accurate clocking solution. With a 3.3V Vcc and a 12 MHz SMD crystal, the CY39100 generates a 6.00000 MHz clock for the USB DPLL, a 4.00000 MHz clock for the RF DPLL, a 1.50000 MHz clock for transmitting data onto the USB, and a 1.00000 MHz clock for transmitting data into the RF DIN. In order to allow the CY39100 to be suspended, the MCU is independently clocked with a 4.0000 MHz quartz crystal.

V.3.j) the Digital Phase Locked Loop

The DPLL in the CPLD is necessary to recover the USB clock. The loop runs at 6.00000 MHz, sampling the USB which is clocked at ¼ the frequency at 1.50000 MHz. The loop expects to sample 4 consecutive high periods and then 4 consecutive low periods 3 times during the USB sync. If the clocks are out of phase and it appears that the 6.00000 MHz clock is too fast, the low period of the derived clock is shortened and if it appears the 6.00000 MHz clock is too slow, the high period of the derived clock is lengthened.

V.3.k) EOP Detection

An End of Packet is technically an out of band signal, asynchronous to the USB clock. A valid EOP is generated when in the 4x domain, 3 or more SE0s are detected followed by a J. Or when 3 or more SE0s are detected followed by a K (i.e. out of band dribble) followed by a J.

V.k.l) Device Attachment

After resetting, the downstream CPLD simply waits for device attachment. I detect attachment by having the CPLD poll the D+ and D- lines waiting for one to be pulled high. A simple debouncing state machine ensures accurate speed detection. Once attachment has occurred, another state machine monitors D+ and D- for an SE0

condition. If SE0 occurs for 3us, the CPLD signals device detachment and notifies the MCU.

V.3.m) the External Buffer

The largest amount of data that ever needs to be stored in a FIFO is from an Interrupt OUT transaction and contains an OUT token along with a DATA packet with a payload of at most 8 bytes. The token is 3 bytes long, the DATA packet is at longest 11 bytes, consisting of 1 byte for the PID, 8 bytes for data, and 2 bytes for CRC16. The total data length is then 14 bytes=112 bits. The NRZI overhead requires storage at worst 112 bits x 7/6 = 130.67 bits long. If the SOPs and EOPs use 1 bit each then the total max buffer size required is 131+4=135bits long by 3 bits wide. I chose the cheapest readily available low power 3.3V FIFO with a minimum length of 135 bits and minimum width of 3 bits. It is a TI SN74ALVC7806 and is 256 x 18 bits. It can be clocked with asynchronous load and store clocks at frequencies up to 25 MHz.

V.3.n) the Remaining Components

All CPLD code is written in VHDL and is fully synthesizable. DC power is provided to components from a TI SMD 3.3V voltage regulator, UA78M00. The same regulator in a 5.0V version is used to provide USB Vbus for the function attached to the downstream WUSBD. The RS232 transceiver is connected to a standard 9-pin DIN for debugging via serial I/O.

V.3.o) the PCB

The printed circuit boards have 2 layers, top and bottom. They do not have a solder mask, the minimum trace width is 7mils, and the minimum trace spacing is 7mils. The traces and vias are tinned, and only 6 drill sizes are used. The CPLD, the MCU, and the programmable clock generator have ground planes on the top layer within their footprints. See: Appendix (2) for the schematic, Appendix (3) for a picture of the layout, and Appendix (5) for photographs of the hardware.

VI) Results

VI.1) Meeting the Initial Design Requirements

The Wireless USB System described in this report is not yet fully functional. I designed and fabricated 2 revisions of the PCBs before yielding a working test-bed. All components and connections on the current boards are working as expected. The 2.4GHz RF link is fully functional, and data can be transmitted reliably via ACKs, checksums, re-transmissions, and timeouts. The UART sharing scheme is working correctly and data can flow in both directions between a terminal and the CPLD, between the CPLD and the MCU, between the MCU and the terminal, and between the MCU and the RF boards. USB device connection and disconnection and speed detection works on the downstream board, and the upstream board can receive this information and perform logical attachment on the upstream bus. USB Clock recovery and generation works correctly and is detailed in Appendix (6). USB data can be read, but the state machines to handle the various USB transactions are not fully functional yet. The state machines to prevent USB bus timeouts are not functioning yet either.

Although the system currently fails to meet the primary goal, allowing the logical attachment of any low-speed USB v1.1 device to any USB v1.1 root-hub without a physical connection between the two, it does meet most of the initial design goals: only low-speed devices are supported, a solid RF link can be established with a range in the order of tens of meters not requiring line of sight, all device interconnects are on a PCB, all components require only a 3.3V DC supply, almost all components are surface mountable, the strict timing and data rate requirements in the USB v1.1 are adhered to at the physical level but are circumvented at the protocol level.

The only other goal not yet achieved is creating a cost-effective and commercially viable solution. The cost of components for each board is nearly $100, mainly due to the cost of the CPLD (see Appendix (4) for the Bill of Materials).

VI.2) Generalizations

I set out to design transparent wireless USB, and in the process I essentially designed a USB bridge. The fact that the connection is a 2.4GHz wireless link is

interesting and makes the device practical, but it could just as easily be a TCP/IP connection or a simple RS232 connection. I cannot see a reason to use a TCP/IP link for USB device connection, but it is certainly possible and is interesting to ponder. Bridging USB to RS232 may be a very interesting project. The upstream board could be replaced by software for the PC. One could create a RS232/USB Host Controller to interface with the USB System software and the RS232 connection to the downstream board. This would allow attachment of low-speed USB devices via RS232, extending the USB paradigm to very old PC architectures.

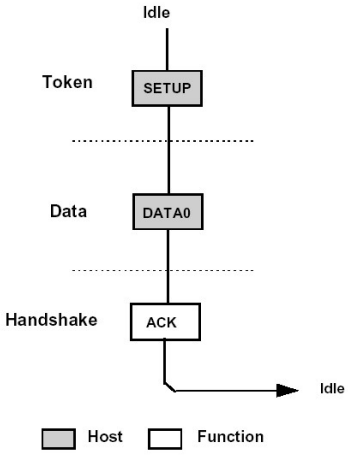
VII) Conclusions

This project was simply too large for any one person to complete in any reasonable amount of time. I am happy with all that I have accomplished, and I feel that the hardware and software I have built is an excellent test-bed and development environment for this project. The current system is nearly working, and is certainly capable of implementing wireless transparent USB.

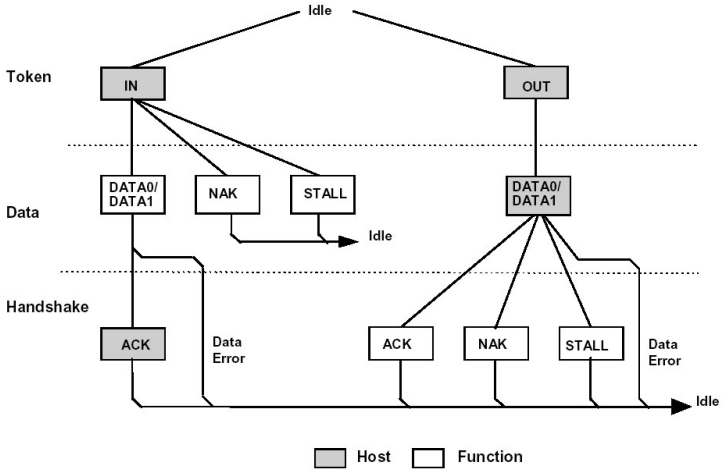
I have designed many different versions of the hardware and software, starting from scratch more then once. Each time, I have succeeded in designing a better system, with more capabilities and fewer limitations. Yet, as expected, the current boards which are capable of implementing wireless USB could be improved.

- Ideally, I would pick an MCU with 2 hardware UARTs. This would greatly ease various difficulties and timing constraints imposed by my UART bus sharing logic. Also, having more the 1024 bytes of RAM would be very convenient for increasing various transmit and receive buffers.
- The Delta39K CPLD is an excellent development device, but I have decreased the size of the VHDL code substantially, and the CPLD should be smaller and static. I suspect that using a cheap Xilinx FPGA is the best solution.
- The ML2724 boards are excellent, but after breaking out of the strict USB timeout constraints, it is readily apparent that a smaller, cheaper, and slower RF board would suffice.
- The Cypress programmable clock generator is a little buggy and unreliable. This may be due to the fact that I did not follow all of the recommended layout rules, or it may be due to ground bounce and other factors. Regardless, it should be replaced with a small, cheap 6.0000MHz oscillator. All other clocks can be easily derived within the CPLD.

Appendix (1): Data Flow of Interrupt and Control Setup Transactions

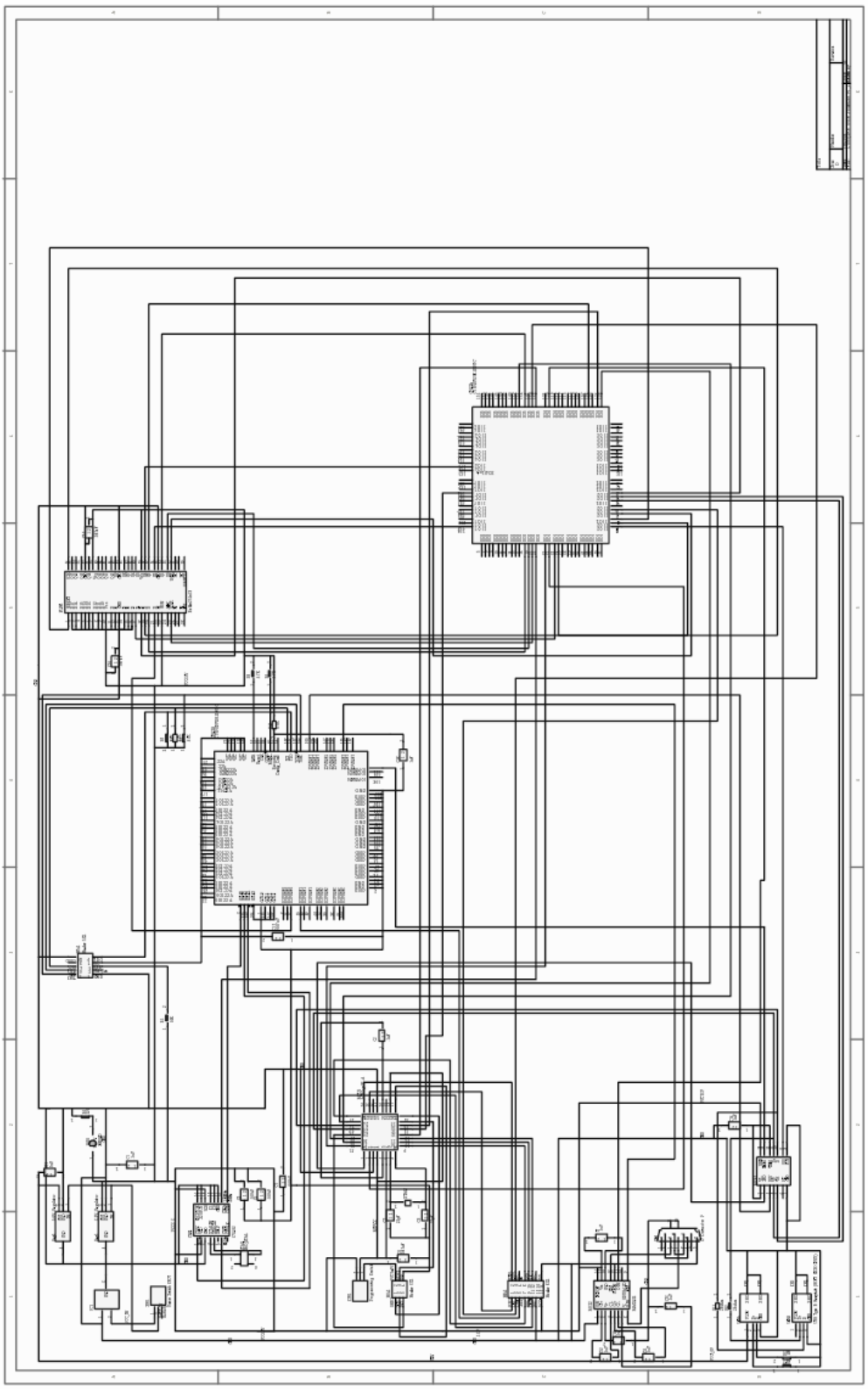


Control Setup Transaction

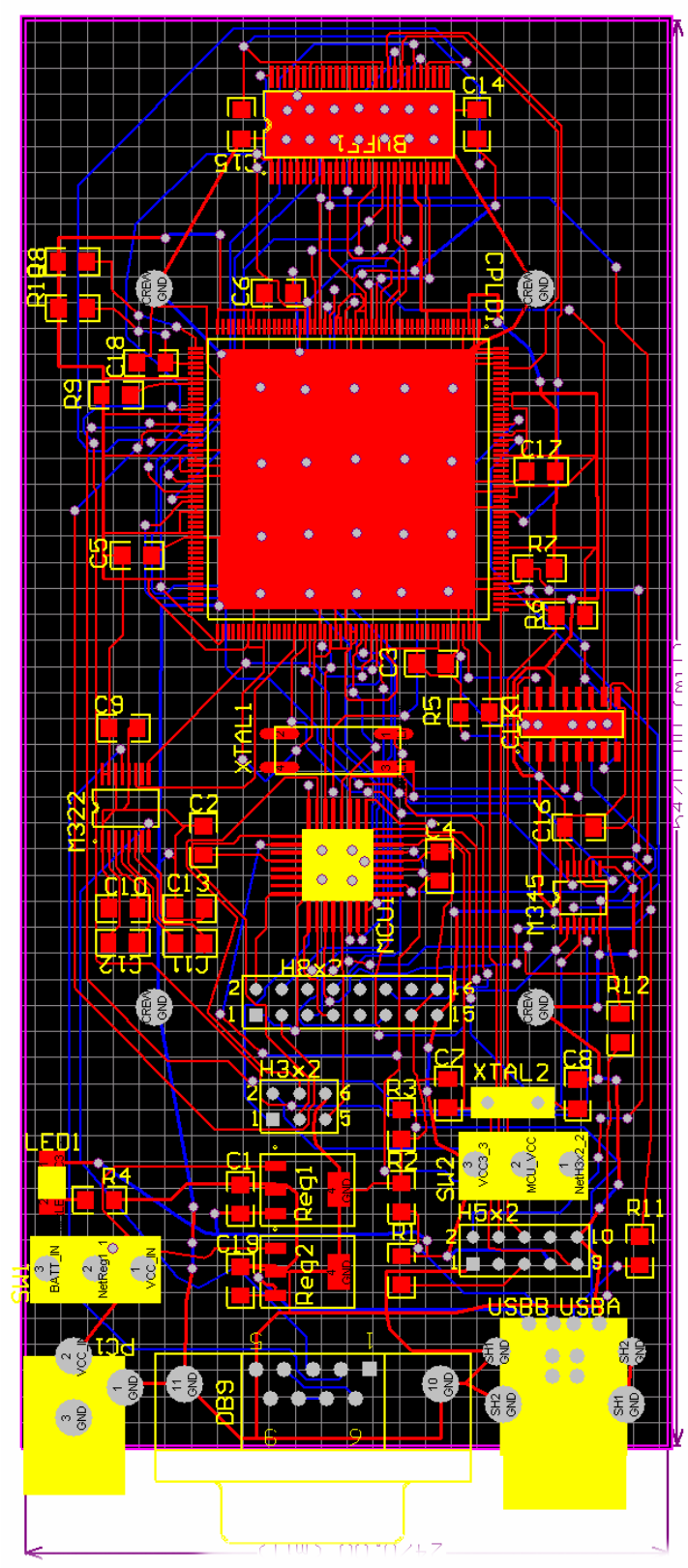


Interrupt Transaction Format

Appendix (2): Board Schematic



Appendix (3): Board Layout



Appendix (4): Bill of Materials (BOM)

Wireless USB Bill of Materials
Version 1.2

Sean Keller
sjk26@cornell.edu

(F) Function Side Only
(H) Host Side Only
(B) Both Host and Function

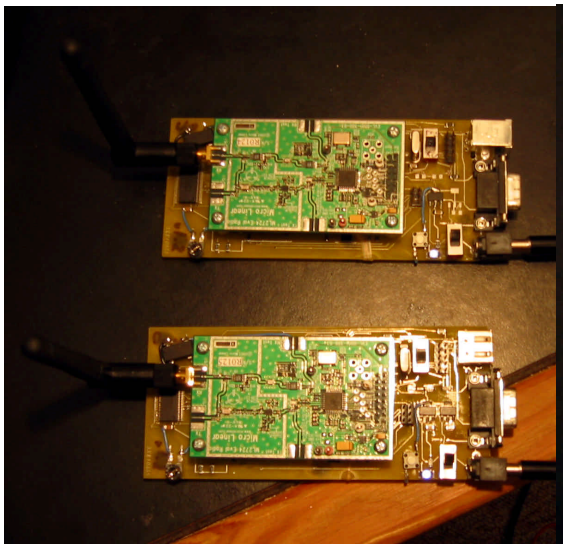
For a Single Host or Function Board:

Qty	Board	Reference	Part Number	Manufacturer	Description	Value	Unit Price in USD	Number x Cost in USD
1	B	P1	RAPC712	Switchcraft Inc.	Male DC Power Jack	N/A	1.08	1.08
8	B	C3-C6 C14 C15 C17 C18	C1206C473K5RACTU	Kemet	Capacitor SMD 1206	.047uF	0.0165	0.132
9	B	C1 C2 C9-C13 C16 C19	C1206C104J5RACTU	Kemet	Capacitor SMD 1206	.1uF	0.0253	0.2277
2	B	C7 C8	1206N220J500NT	BC Components	Capacitor SMD 1206	22pF	0.0146	0.0292
1	B	C18	12062R105K7BB0D	Yageo America	Capacitor SMD 1206	1uF	0.0245	0.0245
1	B	R1	9C12063A1002FKHFT	Yageo America	Resistor SMD 1206	10K	0.0088	0.0088
2	F	R2 R3	9C12063A1502FKHFT	Yageo America	Resistor SMD 1206	15K	0.0088	0.0176
1	B	R4	9C12063A2200FKHFT	Yageo America	Resistor SMD 1206	220	0.0088	0.0088
6	B	R5-10	9C12063A4701FKHFT	Yageo America	Resistor SMD 1206	4.7K	0.0088	0.0528
2	B	R11-12	9C12063A24R0FKHFT	Yageo America	Resistor SMD 1206	24	0.0088	0.0176
2	B	SW1 SW2	1101M2S3CQE2	ITT Industries	SPDT Switch	N/A	3.96	7.92
1	B	Reg1	UA78M33CDCYR	Texas Instruments	3.3V Regulator	N/A	0.56	0.56
1	F	Reg2	UA78M05CDCYR	Texas Instruments	5.0V Regulator	N/A	0.56	0.56
1	B	H3x2	4-103186-0	AMP/Tyco	Straight Dual-Row Male Header	3x2	0.073	0.073
1	B	H5x2	4-103186-0	AMP/Tyco	Straight Dual-Row Male Header	5x2	0.122	0.122
1	B	H8x2	4-103186-0	AMP/Tyco	Straight Dual-Row Male Header	8x2	0.195	0.195
1	B	MCU1	ATMEGA8L-8AI	Atmel	Atmel 3.3V MCU	N/A	5.61	5.61
1	B	CPLD	CY39100V208B-125NTC	Cypress	Deltat 39K 100K Gate CPLD	N/A	67.95	67.95
1	B	BUFF1	SN74ALVC7806-25DL	TI	Asynchronous FIFO	N/A	0	0
1	B	M345	MAX3221IPWR	TI	UART level shifter	N/A	1.65	1.65
1	B	XTAL1	CM309A12.000MABJT	Citizen America Corp	SMD 12Mhz XTAL	N/A	1.51	1.51
1	B	XTAL2	MP040	CTS	XTAL ocsilator 8mhz	N/A	0.94	0.94
1	B	M322	MAX34521-EEUD	Maxim	USB transceiver	N/A	0	0
1	B	USBA	542310419	Molex	USB A recepticle	N/A	0	0
1	B	USBB	670681000	Molex	USB B recepticle	N/A	0	0
1	B	CLK1	CY2292F	Citizen America Corp	Programmable Clock	N/A	4.46	4.46
1	B	LED1	LTST-T670TBKT	Lite On	SMD Blue LED	N/A	1.38	1.38
1	B	DB9	747844-4	AMP/Tyco	DSUB 9-pin Femaile Receptor	N/A	1.94	1.94
					Part Cost Function Board			96.469
					Part Cost Host Board			95.8914

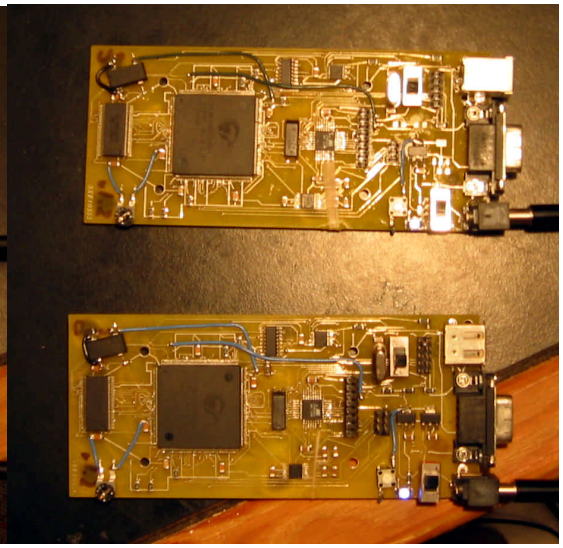
Appendix (5): Photographs of the Boards

Top View

PCBs with ML2724 RF daughterboards



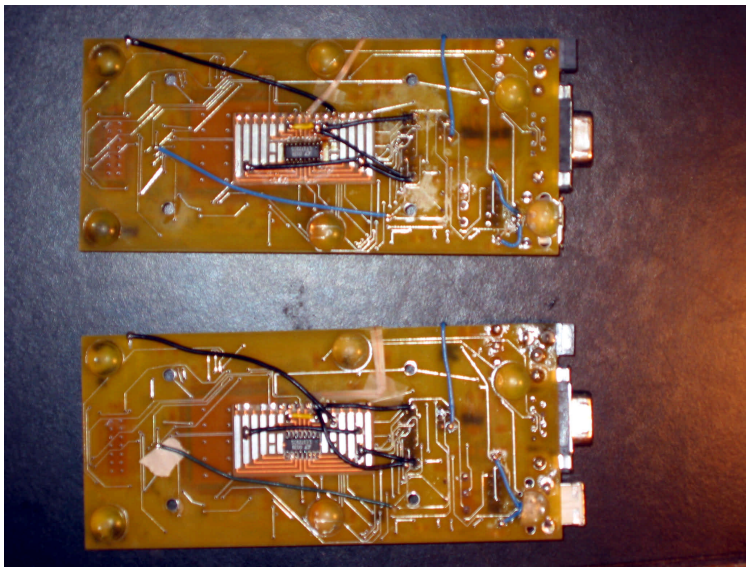
PCBs without the RF daughterboards



Top: upstream boards

Bottom: downstream boards

Bottom View

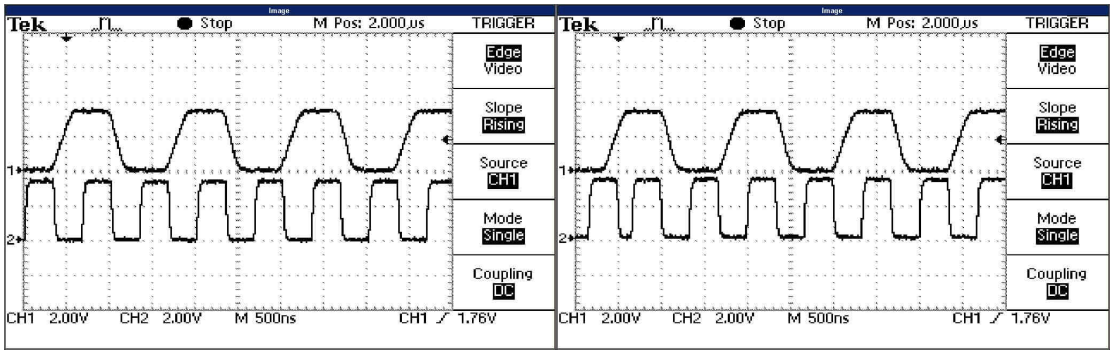


Top: upstream boards

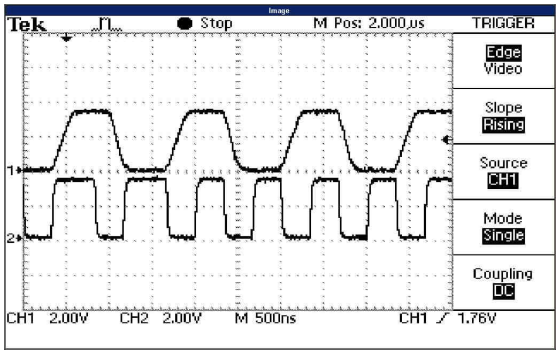
Bottom: downstream boards

Appendix (6): DPLL Locking on to USB SOP

- Top channel is the USB
- Bottom channel is DPLL output



clean sync **shortened low period to sync**



lengthened high period to sync

Appendix (7): VHDL

```
--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 BUS_TX Package
-- Author:               Sean Keller
--                                 sjk26@cornell.edu
------------------------------------------------
-- Created:              July 27, 2003
-- Revision:     1.0 : July 28, 2003
--                                 BUILT :)
-- Revision:     1.1 : July 31, 2003
--                                 Fixed Minor Bugs
--                                 Commented the code
-- Revision:     1.2 : July 31, 2003
--                                 Renamed to USB_TX
-- Revision:     1.3 : August 1, 2003
--                                 Took care of done and K
--                                 outer file must assert tx_CLEAR
--                                 to exit from K...done will be
--                                 asserted for one T in this file
--                                 to let the outer file clear bit
--                                 tx_CLEAR
-- Revision:     1.4 : August 4, 2003
--                                 Killed the enable
-- Revision:     1.5 : August 5, 2003
--                                 Added code to transmit a NAK and a
--                                  corrputed ACK
--                                 Made din_buff 5 bits
------------------------------------------------
-- Known Bugs:   None
------------------------------------------------
-- References:   USB v1.1 Spec
------------------------------------------------
--
-- Copyright (C) 2003     Sean Keller
--                                                    sjk26@cornell.edu
------------------------------------------------
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package USB_TX is
  component SEND_USB_DATA port(
  rst,clk                 : in std_logic;  --reset, and clock in
  full_speed              : in std_logic;  --'0' is low '1' is full
  tx_STALL                : in std_logic;  --'1' means send a stall PID
  tx_BUFFER               : in std_logic;  --dump the buffer onto the bus
  tx_CLEAR                : in std_logic;  --go back to idle
  tx_BADACK               : in std_logic;  --send a broken ACK
  tx_NAK          : in std_logic;  --send a NAK
  din_buff                : in std_logic_vector(4 downto 0);
  clk_buff_rd             : out std_logic; --the buffer's clock
  buff_notoe              : out std_logic; --buffer's not oe
  buff_nrst               : out std_logic; --buffer's not reset
  dplu                    : out std_logic; --D+ line
  dmin                    : out std_logic; --D- line
  done                    : out std_logic);--'1' transfer complete
  end component;
end USB_TX;


library ieee;
use ieee.std_logic_1164.all;
entity SEND_USB_DATA is port (
  rst,clk                 : in std_logic;  --reset, and clock in
  full_speed     : in std_logic;          --'0' is low '1' is full
  tx_STALL                : in std_logic;  --'1' means send a stall PID
  tx_BUFFER               : in std_logic;  --dump the buffer onto the bus
  tx_CLEAR                : in std_logic;  --go back to idle
  tx_BADACK               : in std_logic;  --send a broken ACK
  tx_NAK          : in std_logic;  --send a NAK
  din_buff                : in std_logic_vector(4 downto 0);
  clk_buff_rd             : out std_logic; --the buffer's clock
  buff_notoe              : out std_logic; --buffer's not oe
  buff_nrst               : out std_logic; --buffer's not reset
  dplu                    : out std_logic; --Differential Output
```

```
  dmin                  : out std_logic; --D- line
  done                  : out std_logic); --'1' transfer complete
  end SEND_USB_DATA;

architecture usb_tx_arch of SEND_USB_DATA is
  type states is (idle,SOP1,SOP2,STALL1,EOP1,EOP2,EOP3,EOP4,BUFF1,BUFF2,BUFF3,SK,
                                  NAK1,BAK1);
  signal ps                                         : states;
  signal J                                          : std_logic;
  signal K                                          : std_logic;
  signal buff_usb_out_en  : std_logic;
  signal count                              : integer range 0 to 7;
  --datas for the USB
  signal SOP                                : std_logic_vector(7 downto 0);
  signal STALL,NAK,BAK            : std_logic_vector(7 downto 0);
begin
  K<=not full_speed;
  J<=full_speed;
  SOP<=(K,J,K,J,K,J,K,K);
  STALL<=(K,K,K,J,K,J,K,K);
  NAK<=(K,J,J,K,J,J,K,K);
  BAK<=(J,K,K,J,K,K,J,J);

  neg_logic:process(rst,clk,buff_usb_out_en)
  begin
    if(rst='1') then
         --
         elsif(buff_usb_out_en='0') then
     clk_buff_rd<='0';
         elsif (falling_edge(clk)) then
           clk_buff_rd<=clk;
         end if;
  end process;

  state_comb:process(rst,clk,buff_usb_out_en,din_buff,count)
  begin
    if(rst='1') then
           done<='0';
           buff_usb_out_en<='0';
           dplu<='0';
           dmin<='0';
           ps<=idle;
    elsif rising_edge(clk) then
           case ps is
             when SK =>
                   buff_notoe<='1';
                   dplu<=K;
                   dmin<=not K;
                   if(tx_CLEAR='1') then
                     ps<=idle;
                          done<='1';
                   end if;
            when idle =>
                   buff_notoe<='1';
                   dplu<=J;
                   dmin<=not J;
                   done<='0';
                   if(tx_STALL='1' or tx_NAK='1' or tx_BADACK='1') then
                     --Put a Stall Packet onto the USB
                     ps<=SOP1;
                   elsif(tx_BUFFER='1') then
                     --Start Driving the USB from the Buffer Contents
                     ps<=BUFF1;
                   else
                     ps<=idle;
                   end if;
                when BUFF1 =>
                   ps<=BUFF2;
                   count<=7;
                when BUFF2 =>
                   --Start dumping a SOP onto the USB
                   dplu<=SOP(count);
                   dmin<=not SOP(count);
                   if (count=0) then
                          ps<=BUFF3;
             --By setting this, on the falling edge of the clk,
                          -- the buffer's clock will be set to the 1.5Mhz clk
                          -- and the buffer will be taken out of reset
                          buff_usb_out_en<='1';
                   else
                          ps<=BUFF2;
```

```
  count<=count-1;
        end if;
       when BUFF3 =>
         --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR--
         if(din_buff(2)='1') then
               --This means do nothing
               ps<=EOP4;
               dplu<='0';
               dmin<='0';
           buff_usb_out_en<='0'; --don't let buffer's clock be driven
               buff_notoe<='1';      --The buffer's onto the USB is high Z
           buff_nrst<='0';       --Reset the buffer's pointers
elsif(din_buff(1)='1') then
  --This means a EOP needs to be put onto the USB
               --Begin by driving an SE0 on USB
      ps<=EOP2;
           dplu<='0';
           dmin<='0';
           buff_usb_out_en<='0'; --don't let buffer's clock be driven
               buff_notoe<='1';      --The buffer's onto the USB is high Z
           buff_nrst<='0';       --Reset the buffer's pointers
         else
           buff_usb_out_en<='1';
               ps<=BUFF3;
           buff_notoe<='0';   --Start Driving the USB with the buffer contents
      dplu<=din_buff(0);
           dmin<=not din_buff(0);
         end if;
       when SOP1 =>
         count<=7;
         ps<=SOP2;
       when SOP2 =>
         dplu<=SOP(count);
         dmin<=not SOP(count);
         if (count=0) then
           count<=7;
               if(tx_STALL='1') then
             ps<=STALL1;
               elsif(tx_BADACK='1') then
                 ps<=BAK1;
               else  --must be NAK
                 ps<=NAK1;
               end if;
         else
           count<=count-1;
         end if;
       when STALL1 =>
         dplu<=STALL(count);
         dmin<=not STALL(count);
         if (count=0) then
           ps<=EOP1;
         else
           count<=count-1;
         end if;
       when NAK1 =>
         dplu<=NAK(count);
         dmin<=not NAK(count);
         if (count=0) then
           ps<=EOP1;
         else
           count<=count-1;
         end if;
       when BAK1 =>
         dplu<=BAK(count);
         dmin<=not BAK(count);
         if (count=0) then
           ps<=EOP1;
         else
           count<=count-1;
         end if;
       when EOP1 =>
         dplu<='0';
         dmin<='0';
         ps<=EOP2;
       when EOP2 =>
         dplu<='0';
         dmin<='0';
         ps<=EOP3;
       when EOP3 =>
         dplu<=J;
         dmin<=not J;
```

```
                  ps<=EOP4;
                when EOP4 =>
                  --set the global signal done, so that the outer glue logic
                  -- will set the USB bus direction to receive
                  done<='1';
                  dplu<=J;
                  dmin<=not J;
                  --go back to the initial state, idling
                  ps<=idle;
          end case;
        end if;
  end process;
end usb_tx_arch;

--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 USB_RX_SE0
-- Author:               Sean Keller
--                                 sjk26@cornell.edu
------------------------------------------------
-- Created:              August 5, 2003
-- Revision:     1.0 : August 5, 2003
--                                 Built
------------------------------------------------
-- TODO:         None
------------------------------------------------
-- Known Bugs:   None
------------------------------------------------
-- References:   USB v1.1 Spec
------------------------------------------------
--
-- Copyright (C) 2003    Sean Keller
--                                                 sjk26@cornell.edu
------------------------------------------------
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package USB_RX_SE0 is
  component RECV_USB_SE0 port(
  rst,clk                 : in std_logic;  --reset and 4x clk
  enable         : in std_logic;  --enable
  full_speed     : in std_logic;          --'0' is low '1' is full
  dplu                    : in std_logic;  --usb D+
  dmin                    : in std_logic;  --usb D-
  SE0s                    : out std_logic; --A short SE0
  SE0l                    : out std_logic; --A long SE0
  EOP                     : out std_logic);--End of Packet
  end component;
end USB_RX_SE0;

library ieee;
use ieee.std_logic_1164.all;
entity RECV_USB_SE0 is port(
  rst,clk                 : in std_logic;  --reset and 4x clk
  enable         : in std_logic;  --enable
  full_speed              : in std_logic;  --'0' is low '1' is full
  dplu                    : in std_logic;  --usb D+
  dmin                    : in std_logic;  --usb D-
  SE0s                    : out std_logic; --A short SE0
  SE0l                    : out std_logic; --A long SE0
  EOP                     : out std_logic);--End of Packet
end RECV_USB_SE0;

architecture my_usb_se0_design of RECV_USB_SE0 is
  type States is (RESET,A,B,C,D,PAUSE1,PAUSE2,PAUSE3,PAUSE4,PAUSE5,PAUSE6,PAUSE7,
                                  S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11);
  signal state                    : States;
  signal J                                : std_logic;
  signal K                                : std_logic;
begin
  K<=not full_speed;
  J<=full_speed;

  se0_eop_detect: process(enable,clk,rst,dplu,dmin)
  begin
    if(rst='1' or enable='0') then
      SE0s<='0';
           SE0l<='0';
           EOP<='0';
```

```
  state<=RESET;
elsif(rising_edge(clk)) then
  case state is
  when RESET =>
    SE0l<='0';
         SE0s<='0';
         EOP<='0';
    if((dplu='0') and (dmin='0')) then
           state<=A;
         else
           state<=RESET;
    end if;
  when A =>
    if((dplu='0') and (dmin='0')) then
           state<=B;
         else
           state<=RESET;
         end if;
  when B =>
    SE0s<='1';
    if((dplu='0') and (dmin='0')) then
           state<=C;
         else
           state<=RESET;
         end if;
  when C =>
    if((dplu='0') and (dmin='0')) then
           state<=S1;
         elsif((dplu=J) and (dmin=not J)) then
           state<=PAUSE1;
           EOP<='1';
    else
           state<=D;
         end if;
  when D =>
    SE0s<='0';
    EOP<='1';
         state<=PAUSE1;
  when PAUSE1 =>
    state<=PAUSE2;
  when PAUSE2 =>
    state<=PAUSE3;
  when PAUSE3 =>
    state<=PAUSE4;
  when PAUSE4 =>
    state<=PAUSE5;
  when PAUSE5 =>
    state<=PAUSE6;
  when PAUSE6 =>
    state<=PAUSE7;
  when PAUSE7 =>
    state<=RESET;
  when S1 =>
    if((dplu='0') and (dmin='0')) then
           state<=S2;
         elsif((dplu=J) and (dmin=not J)) then
           state<=PAUSE1;
           EOP<='1';
    else
           state<=D;
         end if;
  when S2 =>
    if((dplu='0') and (dmin='0')) then
           state<=S3;
         elsif((dplu=J) and (dmin=not J)) then
           state<=PAUSE1;
           EOP<='1';
    else
           state<=D;
         end if;
  when S3 =>
    if((dplu='0') and (dmin='0')) then
           state<=S4;
         elsif((dplu=J) and (dmin=not J)) then
           state<=PAUSE1;
           EOP<='1';
    else
           state<=D;
         end if;
  when S4 =>
    if((dplu='0') and (dmin='0')) then
```

```
                state<=S5;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S5 =>
          if((dplu='0') and (dmin='0')) then
                state<=S6;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S6 =>
          if((dplu='0') and (dmin='0')) then
                state<=S7;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S7 =>
          if((dplu='0') and (dmin='0')) then
                state<=S8;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S8 =>
          if((dplu='0') and (dmin='0')) then
                state<=S9;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S9 =>
          if((dplu='0') and (dmin='0')) then
                state<=S10;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S10 =>
          if((dplu='0') and (dmin='0')) then
                state<=S11;
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        when S11 =>
          if((dplu='0') and (dmin='0')) then
                state<=C;
                SE0l<='1';
                SE0s<='0';
              elsif((dplu=J) and (dmin=not J)) then
                state<=PAUSE1;
                EOP<='1';
          else
                state<=D;
              end if;
        end case;
    end if;
  end process;
end my_usb_se0_design;

--------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 USB_RX
```

```
-- Author:               Sean Keller
--                              sjk26@cornell.edu
-------------------------------------------------
-- Created:              August 1, 2003
-- Revision:     1.0 : August 1, 2003
--                              Built
-- Revision:     1.1 : August 4, 2003
--                              Took out tristated output
--                              Made dout_buff 5bits
-------------------------------------------------
-- TODO:         Still needs to be simulated
--                              Need to Take global bus sets and clrs
--                               out of reset and put in outer glue file
-------------------------------------------------
-- Known Bugs:   None
-------------------------------------------------
-- References:   USB v1.1 Spec
-------------------------------------------------
-- Comments:     The PAUSE state is used to hold the outgoing status
--                               signals high for 2 clk Ts
--                              Enable is now used globally to put shared busses
--                               into high Z
-------------------------------------------------
--
-- Copyright (C) 2003     Sean Keller
--                                                 sjk26@cornell.edu
-------------------------------------------------
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package USB_RX is
  component RECV_USB_DATA port(
  rst,clk                 : in std_logic;  --reset and phase locked clock in
  din,enable              : in std_logic;  --differential in from USBT and en
  full_speed              : in std_logic;  --'0' is low '1' is full
  usb_SE0s                : in std_logic;  --A short SE0
  usb_SE0l                : in std_logic;  --A long SE0
  usb_EOP                 : in std_logic;  --End of Packet
  --Buffer Control
  clk_buff_wr             : out std_logic; --external buffer write clock
  dout_buff               : out std_logic_vector(4 downto 0);  --buffer data out
  buff_nrst               : out std_logic; --buffer reset
  --Status Signals
  usb_IDLE                : out std_logic;
  usb_SOP                 : out std_logic;
  usb_K                   : out std_logic;
  usb_SETUP               : out std_logic;
  usb_IN          : out std_logic;
  usb_OUT                 : out std_logic;
  usb_ACK                 : out std_logic;
  usb_NAK                 : out std_logic;
  usb_STALL               : out std_logic);
  end component;
end USB_RX;

library ieee;
use ieee.std_logic_1164.all;
entity RECV_USB_DATA is port(
  rst,clk                 : in std_logic;  --reset and phase locked clock in
  din,enable              : in std_logic;  --differential in from USBT and en
  full_speed              : in std_logic;  --'0' is low '1' is full
  usb_SE0s                : in std_logic;  --A short SE0
  usb_SE0l                : in std_logic;  --A long SE0
  usb_EOP                 : in std_logic;  --End of Packet
  --Buffer Control
  clk_buff_wr     : out std_logic;         --external buffer write clock
  dout_buff               : out std_logic_vector(4 downto 0);  --buffer data out
  buff_nrst               : out std_logic; --buffer reset
  --Status Signals
  usb_IDLE                : out std_logic;
  usb_SOP                 : out std_logic;
  usb_K                   : out std_logic;
  usb_SETUP               : out std_logic;
  usb_IN          : out std_logic;
  usb_OUT                 : out std_logic;
  usb_ACK                 : out std_logic;
  usb_NAK                 : out std_logic;
  usb_STALL               : out std_logic);
end RECV_USB_DATA;
```

```
architecture my_usb_rx_design of RECV_USB_DATA is
  type USB_States is (idle,J1A,J2A,K1A,K2A,K1B,K2B,SK,SOP,PAUSE,IN1,IN2,IN3,IN4,
                      OUT1,OUT2,OUT3,OUT4,ACK1,ACK2,ACK3,ACK4,NAK1,NAK2,NAK3,
                                          NAK4,STALL1,STALL2,STALL3,STALL4,SETUP1,SETUP2,SETUP3,
                                          SETUP4,UK,UJ,UKK,UJJ,UKJ,UJK,UKKK,UKKJ,UKJK,UKJJ,UJKK,
                                          UJKJ,UJJK,UJJJ);
  type SUB_States1 is (RSTS,START,STOP);
  signal ps                       : USB_States;
  signal J                        : std_logic;
  signal K                        : std_logic;
  signal flopped_din              : std_logic;
  signal buffstate                : SUB_States1;
  signal buff_usb_in_en   : std_logic;
begin
  K<=not full_speed;
  J<=full_speed;

  usb_into_buffer: process(enable,clk,rst,buff_usb_in_en,usb_EOP,usb_SE0s)
  begin
    if(rst='1') then
      buffstate<=RSTS;
           dout_buff<="11111";
           buff_usb_in_en<='0';
         elsif(enable='0') then
           buff_nrst<='1';
      buffstate<=RSTS;
           dout_buff<="11111";
           clk_buff_wr<='0';
           buff_usb_in_en<='0';
   elsif(rising_edge(clk)) then
           flopped_din<=din;
           case buffstate is
             when RSTS =>
           if(buff_usb_in_en='1') then
                     buff_nrst<='1';
                     clk_buff_wr<=not clk;
                dout_buff(0)<=flopped_din;
                     dout_buff(1)<='0';
                     dout_buff(2)<='0';
                dout_buff(3)<='0';
                dout_buff(4)<='0';
                     buffstate<=START;
              else
                     clk_buff_wr<='0';
                          buffstate<=RSTS;
                end if;
               when START =>
                 if(usb_SE0s='1') then
                clk_buff_wr<='0';
                          buffstate<=START;
           elsif(usb_EOP='1') then
                 clk_buff_wr<=not clk;
                          dout_buff(0)<='0';
            dout_buff(1)<='1';
            dout_buff(2)<='0';
                dout_buff(3)<='0';
                dout_buff(4)<='0';
                          buffstate<=STOP;
           elsif(usb_SE0l='1') then
                 clk_buff_wr<=not clk;
                          dout_buff(0)<='0';
            dout_buff(1)<='0';
            dout_buff(2)<='1';
                dout_buff(3)<='0';
                dout_buff(4)<='0';
                          buffstate<=STOP;
              else
                clk_buff_wr<=not clk;
                          dout_buff(0)<=flopped_din;
                     dout_buff(1)<='0';
                     dout_buff(2)<='0';
                dout_buff(3)<='0';
                dout_buff(4)<='0';
                          buffstate<=START;
              end if;
                 when STOP =>
                   clk_buff_wr<='0';
              if(buff_usb_in_en='0') then
                          buffstate<=RSTS;
                   else
                          buffstate<=STOP;
```

```
                end if;
        end case;
      end if;
end process;

usb_sm:process(rst,clk,din,ps,enable)
begin
  if(rst='1') then
        ps<=idle;
        usb_SOP<='0';
        usb_K<='0';
        usb_SETUP<='0';
        usb_STALL<='0';
        usb_ACK<='0';
        usb_NAK<='0';
        usb_IN<='0';
        usb_OUT<='0';
        usb_IDLE<='1';
      elsif(enable='0') then
        ps<=idle;
        usb_SOP<='0';
        usb_K<='0';
        usb_SETUP<='0';
        usb_STALL<='0';
        usb_ACK<='0';
        usb_NAK<='0';
        usb_IN<='0';
        usb_OUT<='0';
        usb_IDLE<='1';
       elsif(rising_edge(clk)) then
        case ps is
          when idle =>
                usb_SOP<='0';
                usb_K<='0';
                usb_SETUP<='0';
                usb_STALL<='0';
                usb_ACK<='0';
                usb_NAK<='0';
                usb_IN<='0';
                usb_OUT<='0';
                buff_usb_in_en<='0';
                usb_IDLE<='1';
                if(din=J) then
                  ps<=J1A;
                else
                  ps<=K1B;
                end if;
     when SK =>
                if(din=K) then
                  ps<=SK;
                else
                  ps<=idle;
                end if;
               when SOP =>
        if(din=K) then
                  ps<=UK;
                else
                  ps<=UJ;
                end if;
               when J1A =>
                 if(din=J) then
                   ps<=idle;
                 else
                   ps<=K1A;
                 end if;
               when K1A =>
                 if(din=K) then
                   ps<=idle;
                 else
                   ps<=J2A;
                 end if;
               when J2A =>
                 if(din=J) then
                   ps<=idle;
                 else
                   ps<=K2A;
                 end if;
               when K2A =>
                 if(din=K) then
                   ps<=SOP;
                        usb_SOP<='1';
```

```
            --start buffering data
            buff_usb_in_en<='1';
      else
        buff_usb_in_en<='0';
        ps<=J2A;
      end if;
    when K1B =>
      if(din=K) then
        ps<=K2B;
      else
        ps<=J2A;
      end if;
    when K2B =>
if(din=K) then
        ps<=SK;
            usb_K<='1';
      else
        ps<=idle;
      end if;
    when UK =>
      if(din=K) then
        ps<=UKK;
      else
        ps<=UKJ;
      end if;
    when UJ =>
      if(din=J) then
        ps<=UJJ;
      else
        ps<=UJK;
      end if;
    when UKK =>
      if(din=K) then
        ps<=UKKK;
      else
        ps<=UKKJ;
      end if;
    when UJJ =>
      if(din=J) then
        ps<=UJJJ;
      else
        ps<=UJJK;
      end if;
    when UJK =>
      if(din=J) then
        ps<=UJKJ;
      else
        ps<=UJKK;
      end if;
    when UKJ =>
      if(din=K) then
        ps<=UKJK;
      else
        ps<=UKJJ;
      end if;
    when UKKK =>
      if(din=J) then
        ps<=STALL1;
      else
        ps<=idle;
      end if;
    when UKKJ =>
      if(din=J) then
        ps<=SETUP1;
      else
        ps<=idle;
      end if;
    when UKJK =>
      if(din=K) then
        ps<=IN1;
      else
        ps<=idle;
      end if;
    when UKJJ =>
      if(din=K) then
        ps<=NAK1;
      else
        ps<=idle;
      end if;
    when UJKK =>
      if(din=J) then
```

```
                    ps<=ACK1;
                  else
                    ps<=idle;
                  end if;
                when UJKJ =>
                  if(din=J) then
                    ps<=OUT1;
                  else
                    ps<=idle;
                  end if;
                when UJJK =>
                  ps<=idle;
                when UJJJ =>
                  ps<=idle;
                when IN1 =>
                  ps<=IN2;
                when IN2 =>
                  ps<=IN3;
                when IN3 =>
                  ps<=IN4;
                when IN4 =>
                  ps<=PAUSE;
                  usb_IN<='1';
                when OUT1 =>
                  ps<=OUT2;
                when OUT2 =>
                  ps<=OUT3;
                when OUT3 =>
                  ps<=OUT4;
                when OUT4 =>
                  ps<=PAUSE;
                  usb_OUT<='1';
                when ACK1 =>
                  ps<=ACK2;
             when ACK2 =>
                  ps<=ACK3;
                when ACK3 =>
                  ps<=ACK4;
                when ACK4 =>
                  ps<=PAUSE;
               usb_ACK<='1';
                when NAK1 =>
                  ps<=NAK2;
                when NAK2 =>
                  ps<=NAK3;
                when NAK3 =>
                  ps<=NAK4;
                when NAK4 =>
                  ps<=PAUSE;
                  usb_NAK<='1';
                when STALL1 =>
                  ps<=STALL2;
                when STALL2 =>
                  ps<=STALL3;
                when STALL3 =>
                  ps<=STALL4;
                when STALL4 =>
                  ps<=PAUSE;
                  usb_STALL<='1';
                when SETUP1 =>
                  ps<=SETUP2;
                when SETUP2 =>
                  ps<=SETUP3;
                when SETUP3 =>
                  ps<=SETUP4;
                when SETUP4 =>
                  ps<=PAUSE;
                  usb_SETUP<='1';
                when PAUSE =>
                  ps<=idle;
          end case;
        end if;
  end process;
end my_usb_rx_design;

--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 Timer Package
-- Author:               Sean Keller
--                               sjk26@cornell.edu
```

```
-------------------------------------------------
-- Created:              July 27, 2003
-- Revision:     1.0 : July 27, 2003
--                               Created
--                               Counter1 is 16bits
-- Revision:     1.1 : July 28, 2003
--                               Added Counter2 6bits
-------------------------------------------------
-- Known Bugs:    NONE
-------------------------------------------------
-- References:  "VHDL for Programmable Logic"
--                               By: Kevin Skahill, pages 199-205
--                               Addison Wesley Publishing Inc.
--                               Menlo Park, CA 1996
-------------------------------------------------
--
-- Copyright (C) 2003     Sean Keller
--                                                  sjk26@cornell.edu
-------------------------------------------------
--------------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;
package TIMER is
  component COUNTER1 port(
  rst,clk                 : in std_logic;          --reset and clock in
  uf                      : out std_logic; --underflow
  time_in                 : in std_logic_vector(15 downto 0));
  end component;
  component COUNTER2 port(
  rst,clk                 : in std_logic;  --reset and clock in
  uf                      : out std_logic; --underflow
  time_in                 : in std_logic_vector(5 downto 0));
  end component;
end TIMER;

library ieee;
library work;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity COUNTER1 is port (
  rst,clk                 : in std_logic;  --reset and 1.5Mhz clock in
  uf                      : out std_logic; --underflow
  time_in                 : in std_logic_vector(15 downto 0));
end COUNTER1;

library ieee;
library work;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity COUNTER2 is port (
  rst,clk                 : in std_logic;  --reset and 1.5Mhz clock in
  uf                      : out std_logic; --underflow
  time_in                 : in std_logic_vector(5 downto 0));
end COUNTER2;

architecture my_timer1 of COUNTER1 is
signal cnt : std_logic_vector(15 downto 0);
begin
  count: process(rst,clk,time_in,cnt)
  begin
  if rst='1' then
    cnt<=time_in;
        uf<='0';
  elsif rising_edge(clk) then
        cnt<=cnt-1;
        if(cnt="0000000000000000") then
          uf<='1';
        end if;
  end if;
  end process;
end my_timer1;

architecture my_timer2 of COUNTER2 is
signal cnt : std_logic_vector(5 downto 0);
begin
  count: process(rst,clk,time_in,cnt)
  begin
  if rst='1' then
```

```
    cnt<=time_in;
         uf<='0';
  elsif rising_edge(clk) then
         cnt<=cnt-1;
         if(cnt="000000") then
           uf<='1';
         end if;
  end if;
  end process;
end my_timer2;

--------------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 RF_TX Package
-- Author:               Sean Keller
--                               sjk26@cornell.edu
-------------------------------------------------
-- Created:              July 29, 2003
-- Revision:     1.0 : July 29, 2003
--                               BUILT
-- Revision:     1.1 : August 1, 2003
--                               Modified CNTRL signals to be length 10 always
--                               Took out buffer sets from reset
-- Revision:     1.2 : August 4, 2003
--                               Took out signal en as it was unused
--                               Took out tristates
--                               Changed clk='0' to falling edge
--                               Took out all explcit set and clear of clk_rd
--                               Took out enable
--                               Made din_buffer 5bits
-------------------------------------------------
-- Known Bugs:   SEE CODE :)...need to ensure a EOP will be present or force 1
-------------------------------------------------
-- References:   USB v1.1 Spec
-------------------------------------------------
--
-- Copyright (C) 2003    Sean Keller
--                                                  sjk26@cornell.edu
-------------------------------------------------
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package RF_TX is
  component SEND_RF_DATA port(
  rst,clk                 : in std_logic;  --reset, enable, and clock in
  full_speed              : in std_logic;  --'0' is low '1' is full
  tx_BUFFER               : in std_logic;  --dump the buffer onto the bus
  tx_SE0         : in std_logic;  --send out a SE0 for undeterminded length
  tx_k                    : in std_logic;  --send state K for undetermined length
  clk_buff_rd             : out std_logic; --the buffer's clock
  buff_notoe              : out std_logic; --buffer's not oe
  buff_nrst               : out std_logic; --buffer's not reset
  dout_rf                 : out std_logic; --output to the RFTrans
  done                    : out std_logic; --'1' transfer complete
  din_buff                : in std_logic_vector(4 downto 0));
  end component;
end RF_TX;

library ieee;
use ieee.std_logic_1164.all;
entity SEND_RF_DATA is port (
  rst,clk                 : in std_logic;  --reset, enable, and clock in
  full_speed              : in std_logic;  --'0' is low '1' is full
  tx_BUFFER               : in std_logic;  --dump the buffer onto the bus
  tx_SE0         : in std_logic;  --send out a SE0 for undeterminded length
  tx_k                    : in std_logic;  --send state K for undetermined length
  clk_buff_rd             : out std_logic; --the buffer's clock
  buff_notoe              : out std_logic; --buffer's not oe
  buff_nrst               : out std_logic; --buffer's not reset
  dout_rf                 : out std_logic; --output to the RFTrans
  done                    : out std_logic; --'1' transfer complete
  din_buff                : in std_logic_vector(4 downto 0));
end SEND_RF_DATA;

architecture my_design of SEND_RF_DATA is
  type states is (idle,PRE1,PRE2,PRE3,PRE4,PRE5,PRE6,DATA1,DATAJ1,DATAJ2,DATAJ3,
                  DATAJ4,DATAK1,DATAK2,DATAK3,EOP1,EOP2,EOP3,EOP4,EOP5,EOP6,EOP7,
                                 EOP8,EOP9,SE01,SE0a,SE0,K1,K2,K3,DONE1);
  signal ps                             : states;
```

```
  signal J                                  : std_logic;
  signal K                                  : std_logic;
  signal buff_rf_out_en           : std_logic;
  signal sendEOP                  : std_logic;
  signal count                              : integer range 0 to 7;
  --datas for the RF
  signal SOP                                : std_logic_vector(7 downto 0);
  signal PREAMBLE                 : std_logic_vector(7 downto 0);

begin
  K<=not full_speed;
  J<=full_speed;
  PREAMBLE<=(K,J,K,J,K,J,K,J);
  SOP<=(K,J,K,J,K,J,K,K);

  neg_logic:process(clk,rst,buff_rf_out_en)
  begin
    if(rst='1') then
          ---
        elsif(buff_rf_out_en='0') then
          clk_buff_rd<='0';
        elsif(falling_edge(clk)) then
          clk_buff_rd<=not clk;
        end if;
  end process;

  state_combs:process(rst,clk,buff_rf_out_en,tx_BUFFER,tx_SE0,din_buff)
  begin
    if(rst='1') then
          done<='0';
          buff_rf_out_en<='0';
          dout_rf<='0';----------------------????????????????????
          ps<=idle;
    elsif rising_edge(clk) then
          case ps is
            when idle =>
                  dout_rf<='0';
                  if(tx_BUFFER='1' or tx_SE0='1') then
                    ps<=PRE1;
                  else
                    ps<=idle;
                  end if;
                when PRE1 =>
                  ps<=PRE2;
                  count<=7;
                --TRANSMIT 4 copies of PREAMBLE to center the other data slicer
                when PRE2 =>
                  dout_rf<=PREAMBLE(count);
                  if (count=0) then
                        ps<=PRE3;
                        count<=7;
                  else
                        ps<=PRE2;
            count<=count-1;
                  end if;
                when PRE3 =>
                  dout_rf<=PREAMBLE(count);
                  if (count=0) then
                        ps<=PRE4;
                        count<=7;
                  else
                        ps<=PRE3;
            count<=count-1;
                  end if;
                when PRE4 =>
                  dout_rf<=PREAMBLE(count);
                  if (count=0) then
                        ps<=PRE5;
                        count<=7;
                  else
                        ps<=PRE4;
            count<=count-1;
                  end if;
                when PRE5 =>
                  dout_rf<=PREAMBLE(count);
                  if (count=0) then
                        ps<=PRE6;
                        count<=7;
                  else
                        ps<=PRE5;
            count<=count-1;
```

```
          end if;
         --send a SOP
         when PRE6 =>
           dout_rf<=SOP(count);
           if (count=0) then
                 if(tx_BUFFER='1') then
                   ps<=DATA1;
                   buff_rf_out_en<='1';
                 elsif(tx_SE0='1') then
                   ps<=SE01;
                   buff_rf_out_en<='0';
                   buff_notoe<='1';
               buff_nrst<='0';
                 end if;
           else
                 ps<=PRE6;
    count<=count-1;
           end if;
         when DATA1 =>
           --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR!!!!!!!!!--
  if(din_buff(1)='1') then
       ps<=EOP1;
                 dout_rf<=J;
    buff_rf_out_en<='0';
                 buff_notoe<='1';
           else
             buff_notoe<='0';
                 dout_rf<=din_buff(0);
        if(din_buff(0)=J) then
                   ps<=DATAJ1;
                   buff_rf_out_en<='1';
                 else
                   ps<=DATAK3;
                   buff_rf_out_en<='0';
           end if;
           end if;
when DATAJ1 =>
           --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR!!!!!!!!!--
  if(din_buff(1)='1') then
       ps<=EOP1;
                 dout_rf<=J;
    buff_rf_out_en<='0';
                 buff_notoe<='1';
           buff_nrst<='0';
           else
    buff_rf_out_en<='1';
             buff_notoe<='0';
             dout_rf<=din_buff(0);
             if(din_buff(0)=J) then
                   ps<=DATAJ2;
                 else
                   ps<=DATAK1;
                 end if;
           end if;
when DATAJ2 =>
  if(din_buff(1)='1') then
                          --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR!!!!!!!!!--
       ps<=EOP1;
                 dout_rf<=J;
    buff_rf_out_en<='0';
                 buff_notoe<='1';
           buff_nrst<='0';
           else
    buff_rf_out_en<='1';
             buff_notoe<='0';
             dout_rf<=din_buff(0);
             if(din_buff(0)=J) then
                   ps<=DATAJ3;
                 else
                   ps<=DATAK1;
                 end if;
           end if;
when DATAJ3 =>
           if(din_buff(1)='1') then
       ps<=EOP1;
                 dout_rf<=J;
    buff_rf_out_en<='0';
                 buff_notoe<='1';
           buff_nrst<='0';
           else
             ps<=DATAJ4;
```

```
            buff_notoe<='1';
            buff_rf_out_en<='0';
            dout_rf<=K;
          end if;
        when DATAJ4 =>
          ps<=DATAK1;
          buff_notoe<='0';
  buff_rf_out_en<='1';
          dout_rf<=din_buff(0);
when DATAK1 =>
  if(din_buff(1)='1') then
                           --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR!!!!!!!!!--
       ps<=EOP1;
                dout_rf<=J;
    buff_rf_out_en<='0';
                buff_notoe<='1';
            buff_nrst<='0';
          else
    buff_rf_out_en<='1';
            buff_notoe<='0';
            dout_rf<=din_buff(0);
            if(din_buff(0)=K) then
                  ps<=DATAK2;
               else
                 ps<=DATAJ1;
               end if;
          end if;
when DATAK2 =>
  if(din_buff(1)='1') then
                           --NEED TO TIMEOUT OR ENSURE THIS WILL OCCUR!!!!!!!!!--
       ps<=EOP1;
                dout_rf<=J;
    buff_rf_out_en<='0';
                buff_notoe<='1';
            buff_nrst<='0';
          else
            buff_notoe<='0';
            dout_rf<=din_buff(0);
            if(din_buff(0)=K) then
                  ps<=DATAK3;
                  buff_rf_out_en<='0';
               else
                  ps<=DATAJ1;
                  buff_rf_out_en<='0';
               end if;
          end if;
        when DATAK3 =>
          ps<=DATAJ1;
  buff_notoe<='1';
          buff_rf_out_en<='1';
          dout_rf<=J;
when SE01 =>
          dout_rf<=J;
          ps<=EOP1;
        when EOP1 =>
          dout_rf<=K;
          ps<=EOP2;
        when EOP2 =>
  dout_rf<=K;
          ps<=EOP3;
        when EOP3 =>
  dout_rf<=K;
          ps<=EOP4;
        when EOP4 =>
          dout_rf<=K;
          ps<=EOP5;
        when EOP5 =>
          dout_rf<=J;
          ps<=EOP6;
        when EOP6 =>
          if(tx_K='1') then
            dout_rf<=K;
               ps<=K1;
          else
            dout_rf<=J;
            ps<=EOP7;
          end if;
        when EOP7 =>
          if(tx_BUFFER='1') then
            dout_rf<=K;
            ps<=EOP8;
```

```
                  else
                    dout_rf<=J;
                         ps<=SE0a;
                  end if;
                when EOP8 =>
                  dout_rf<=J;
                  ps<=EOP9;
                when EOP9 =>
                  dout_rf<=K;
                  ps<=DONE1;
                when K1 =>
                  dout_rf<=K;
                  ps<=K2;
                when K2 =>
                  dout_rf<=J;
                  ps<=K3;
                when K3 =>
                  dout_rf<=K;
                  ps<=DONE1;
                when SE0a =>
                  dout_rf<=K;
                  ps<=SE0;
                when SE0 =>
                  dout_rf<=K;
                  ps<=DONE1;
                when DONE1 =>
                  --results in trailing datas...but that is ok
                  dout_rf<='0';
                  done<='1';
                  ps<=idle;
          end case;
        end if;
  end process;
end my_design;


----------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 RF_SM Package
-- Author:               Sean Keller
--                                 sjk26@cornell.edu
-------------------------------------------------
-- Created:              July 27, 2003
-- Revision:     1.0 : July 27, 2003
--                                 Copied over from wusb_host
--                                 Added more states and handled J/K/SE0 better
-- Revision:     1.1 : July 28, 2003
--                                 Fixed J/K/SE0 and added timeout for k
-- Revision:     1.2 : July 30, 2003
--                                 Modified CNTRL condition from SOP
--                                 MUST ALWAYS HAVE J,K,K,K,K to get to cntrl
-- Revision:     1.3 : July 31, 2003
--                                 Added more code
--                                 Commented Code
--                                 Renamed to RF_RX...more telling
-- Revision:     1.4 : July 31, 2003
--                                 Added code
--                                 Moved some logic to falling edge
--                                 DONE
-- Revision:     1.5 : August 3, 2003
--                                 Moved all rising edge logic into one process
--                                 no longer tristate buffer outputs
--                                 removed some halt logic
--                                 Made dout_buffer 5bits
-------------------------------------------------
-- Known Bugs:   NONE...but needs to be simulated
-------------------------------------------------
-- References:   USB v1.1 Spec
-------------------------------------------------
--
-- Copyright (C) 2003    Sean Keller
--                                                 sjk26@cornell.edu
-------------------------------------------------
----------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package RF_RX is
  component RECV_RF_DATA port(
  rst,clk                : in std_logic;  --reset and phase locked clock in
```

```
  din,enable              : in std_logic;  --incoming from data slicer and en
  full_speed              : in std_logic;  --'0' is low '1' is full
  buff_nrst               : out std_logic; --'0' is reset the buffer
  clk_buff_wr             : out std_logic; --the buffer's write clock
  dout_buff               : out std_logic_vector(4 downto 0);
  rf_CLR         : out std_logic; --'1' got a CLR
  rf_SOP         : out std_logic; --'1' got RF SOP
  rf_SE0         : out std_logic; --'1' got hold SE0 till unknown
  rf_EOP         : out std_logic; --'1' got RF EOP
  rf_K                    : out std_logic; --'1' steady K
  rf_IDLE                 : out std_logic);--'1' J the USB
  end component;
end RF_RX;

library ieee;
library wusb_lib;
library cypress;
use ieee.std_logic_1164.all;
use wusb_lib.FIFOS.all;
use wusb_lib.TIMER.all;
use cypress.std_arith.all;
entity RECV_RF_DATA is port (
  rst,clk                 : in std_logic;  --reset and phase locked clock in
  din,enable              : in std_logic;  --incoming from data slicer and en
  full_speed              : in std_logic;  --'0' is low '1' is full
  buff_nrst               : out std_logic; --'0' is reset the buffer
  clk_buff_wr             : out std_logic; --the buffer's write clock
  dout_buff               : out std_logic_vector(4 downto 0);
  rf_CLR         : out std_logic; --'1' got a CLR
  rf_SOP         : out std_logic; --'1' got RF SOP
  rf_SE0         : out std_logic; --'1' got hold SE0 till unknown
  rf_EOP         : out std_logic; --'1' got RF EOP
  rf_K                    : out std_logic; --'1' steady K
  rf_IDLE                 : out std_logic);--'1' J the USB
end RECV_RF_DATA;

architecture my_design of RECV_RF_DATA is
  type RF_States is (idle,J1A,J1A1,J1A2,J2A,J2A1,J2A2,K1A,K1A1,K1A2,K2A,K2A1,
                     K2A2,K1B,K1B1,K1B2,SOP,K1D,J1C,J2C,J3C,J4C,J5C,J6C,J7C,
                                        K1E,K2E,K3E,K4E,SPEC,CNTRL1,CNTRL2,CNTRL3,CNTRL4,EOP1,
                                        EOP2,SE0a,SE0,SKa,SK,CLR);
  type FIFO_States is (RESET,START,STOP,DBLBUFF1);
  signal state                    : RF_States;
  signal J                        : std_logic;
  signal K                        : std_logic;
  signal cnt_uflow                : std_logic;              --for the counter
  signal cnt_rst          : std_logic;
  signal time_in          : std_logic_vector(15 downto 0);
  signal flopped_din              : std_logic;              --registered data input
  signal storage_bin              : std_logic;              --registered temp storage
  --fifo10 signals
  signal fifo10_clk               : std_logic;
  signal fifo10_rd                : std_logic;
  signal fifo10_wr                : std_logic;
  signal fifo10_rdinc             : std_logic;
  signal fifo10_wrinc             : std_logic;
  signal fifo10_rdptrclr  : std_logic;
  signal fifo10_wrptrclr  : std_logic;
  signal fifo10_din               : std_logic;
  signal fifo10_dout              : std_logic;
  signal fifo10_rf_in_en  : std_logic;
  signal fifo10_drop              : std_logic;
  signal fifo10_wrptr             : integer range 0 to 9;
  signal fifo10state              : FIFO_States;
  --buffer controls
  signal buff_halt                : std_logic;
  signal start_buffering  : std_logic;

begin
  K<=not full_speed;
  J<=full_speed;
  time_in<="0011101010011000";    --10ms
  TIMEOUT: COUNTER1 port map(rst=>cnt_rst,clk=>clk,uf=>cnt_uflow,time_in=>time_in);
  FIFO10: fifo10 port map(rst=>rst,clk=>fifo10_clk,rd=>fifo10_rd,wr=>fifo10_wr,
                          rdinc=>fifo10_rdinc,wrinc=>fifo10_wrinc,rdptrclr=>fifo10_rdptrclr,

wrptrclr=>fifo10_wrptrclr,data_in=>fifo10_din,data_out=>fifo10_dout,
                                                    wrptr_out=>fifo10_wrptr);
  fifo10_clk<=not clk;
  fifo10_din<=flopped_din;
```

```
usb_rf_buffer:process(enable,rst,clk,fifo10_rf_in_en,fifo10_drop,storage_bin,start_buffering,fifo10_do
ut,fifo10_wrptr,buff_halt)
  begin
    if(rst='1') then
          fifo10_rf_in_en<='0';
          buff_halt<='0';
      fifo10state<=RESET;
          --fifo is in pseudo reset
          fifo10_wrptrclr<='1';
          fifo10_rdptrclr<='1';
          fifo10_wr<='0';
          fifo10_rd<='0';
      fifo10_rdinc<='0';
      fifo10_wrinc<='0';
          fifo10_drop<='0';
          flopped_din<='0';
          storage_bin<='0';
          start_buffering<='0';
          dout_buff<="11111";
        elsif(enable='0') then
      buff_nrst<='1';
      clk_buff_wr<='0';
          dout_buff<="11111";
          fifo10_rf_in_en<='0';
          buff_halt<='0';
      fifo10state<=RESET;
          --fifo is in pseudo reset
          fifo10_wrptrclr<='1';
          fifo10_rdptrclr<='1';
          fifo10_wr<='0';
          fifo10_rd<='0';
      fifo10_rdinc<='0';
      fifo10_wrinc<='0';
          fifo10_drop<='0';
    elsif(falling_edge(clk)) then  --BE GENTLE HERE!!!
          if(fifo10_wrptr=0 and start_buffering='1') then
            if(rf_SE0='1') then
              clk_buff_wr<='0';
                elsif(rf_EOP='1') then
          --need to write this into the buffer still
              clk_buff_wr<=clk;
              buff_nrst<='1';
                  dout_buff(0)<='0';
              dout_buff(1)<='1';
              dout_buff(2)<='0';
              dout_buff(3)<='0';
              dout_buff(4)<='0';
        elsif(rf_CLR='1') then
          --need to write this into the buffer still
                clk_buff_wr<=clk;
              buff_nrst<='1';
                  dout_buff(0)<='0';
              dout_buff(1)<='0';
              dout_buff(2)<='1';
              dout_buff(3)<='0';
              dout_buff(4)<='0';
                 else
                   if (fifo10_drop='1') then
                          --turn the buffer clock back on
                     buff_nrst<='1';
                  dout_buff(0)<=storage_bin;
                      dout_buff(1)<='0';
                      dout_buff(2)<='0';
                  dout_buff(3)<='0';
                  dout_buff(4)<='0';
                     else
                   if(buff_halt='1') then
                         clk_buff_wr<='0';
                              --buff_halt<='0';
                         storage_bin<=fifo10_dout;
           else
                              clk_buff_wr<=clk;
                              --dump the data from the fifo into the buffer
             --start the buffer clock...(won't start if halt='1' and a drop is needed)
                         buff_nrst<='1';
                         --save a copy of the first fifo_element in case this is a drop
                    dout_buff(0)<=fifo10_dout;
                         dout_buff(1)<='0';
                         dout_buff(2)<='0';
                     dout_buff(3)<='0';
```

```
                dout_buff(4)<='0';
                        --don't set wrinc...it will be zero if a drop else already set to inc
                       end if;
                end if;
               end if;
        end if;
      end if;
end process;

state_comb:process(enable,cnt_uflow,clk,rst,din,state,J,K)
begin
  if(rst = '1') then
    state<=SE0;
         rf_CLR<='0';
         rf_SE0<='1';
         rf_EOP<='0';
         rf_SOP<='0';
         rf_K<='0';
    rf_IDLE<='0';
         cnt_rst<='1';
       elsif (enable='0') then
         rf_CLR<='0';
    rf_SE0<='1';
         rf_EOP<='0';
         rf_SOP<='0';
         rf_K<='0';
    rf_IDLE<='0';
    state<=SE0;
         cnt_rst<='1';
  elsif (cnt_uflow='1') then
         --SK time out occured
         state<=SE0;
         cnt_rst<='1';
       elsif rising_edge(clk) then
         case state is
           when EOP2 =>
                 state<=idle;
                 if(din=K) then
                   --signal EOP
                   rf_EOP<='1';
                 end if;
               when SE0 =>
                 rf_SOP<='0';
                 rf_CLR<='0';
                 rf_EOP<='0';
                 rf_K<='0';
                 rf_SE0<='1';
                 rf_IDLE<='0';
                 if(din=J) then
                   state<=J1A1;
                 else
                   state<=K1B1;
                 end if;
               when SK =>
                 rf_SOP<='0';
                 rf_CLR<='0';
                 rf_EOP<='0';
                 rf_K<='1';
                 rf_SE0<='0';
                 rf_IDLE<='0';
                 if(din=J) then
                   state<=J1A2;
                 else
                   state<=K1B2;
                 end if;
      when idle =>
                 --In this state, host device is not driving the bus
                 -- The 1.5K pullup is setting state J
                 cnt_rst<='1';
                 rf_SOP<='0';
                 rf_CLR<='0';
                 rf_EOP<='0';
                 rf_K<='0';
                 rf_SE0<='0';
                 rf_IDLE<='1';
                 if(din = J) then
                   state<=J1A;
                 else
                   state<=K1B;
                 end if;
               when SOP =>
```

```
                    rf_SOP<='1';
                    rf_CLR<='0';
                    rf_EOP<='0';
                    rf_K<='0';
                    rf_SE0<='0';
                    rf_IDLE<='0';
                    if(din = K) then
                      state<=K1D;
                    else
                      state<=J1C;
                    end if;
                  when CLR =>
                    rf_CLR<='1';
                    state<=idle;
-----------SOP FROM idle------------
                  when J1A =>
                    if(din=J) then
                      state<=idle;
                    else
                      state<=K1A;
                    end if;
                  when K1A =>
                    if(din=K) then
                      state<=idle;
                    else
                      state<=J2A;
                    end if;
                  when J2A =>
                    if(din=J) then
                      state<=idle;
                    else
                      state<=K2A;
                    end if;
                  when K2A =>
                    if(din=K) then
                      state<=SOP;
                           --BEGIN BUFFERING
                           --By setting this bit, on the next rising clk edge
                           -- The data will be pushed into the fifo...this is
                           -- cool, as in the current state, the next rising
                           -- clk edge will hold the first valid data
                           fifo10_rf_in_en<='1';
                    else
                      state<=J2A;
                    end if;
                  when K1B =>
                    if(din=K) then
                      state<=idle;
                    else
                      state<=J1A;
                    end if;
-----------------------------------
                  when K1D =>
                    rf_SOP<='0';
                    if(din = K) then
                      state<=idle;
                    else
                      state<=J1C;
                    end if;
                  when J1C =>
                    buff_halt<='0';
                    rf_SOP<='0';
                    if(din=J) then
                      state<=J2C;
                    else
                      state<=K1E;
                    end if;
                  when J2C =>
                    if(din=J) then
                      state<=J3C;
                    else
                      state<=K1E;
                    end if;
             when J3C =>
                    if(din=J) then
            state<=J4C;
                    else
                      state<=K1E;
                           --DROP THIS DATA
                           --Stop reading the fifo and don't increment rdptr on
                           -- the next falling edge of clk...next fifo rd will be
```

```
              -- from the next unread bit.
                   --Let a dummy write occur
                   --Note that a drop should occur with fifo10_drop='1'
                   --Don't increment the wrptr, the next write will be to
                   -- the next free bit
                   --Also halt the next buffer write...the current write will
                   -- happen simaltaneous to this
                   fifo10_rd<='0';
              fifo10_rdinc<='0';
                   fifo10_drop<='1';
                   fifo10_wrinc<='0';
                   buff_halt<='1';
            end if;
when J4C =>
            if(din=J) then
              state<=idle;
            else
              state<=J5C;
            end if;
          when J5C =>
            if(din=J) then
              state<=idle;
            else
              state<=J6C;
            end if;
          when J6C =>
            if(din=J) then
              state<=idle;
            else
              state<=J7C;
            end if;
          when J7C =>
            if(din=J) then
              state<=idle;
            else
              state<=K4E;
            end if;
          when K1E =>
            buff_halt<='0';
            if(din=K) then
              state<=K2E;
            else
              state<=J1C;
            end if;
          when K2E =>
            if(din=K) then
              state<=K3E;
            else
              state<=J1C;
            end if;
          when K3E =>
            if(din=K) then
              state<=K4E;
            else
              state<=J1C;
                   --DROP THIS DATA
                   --Stop reading the fifo and don't increment rdptr on
                   -- the next falling edge of clk...next fifo rd will be
              -- from the next unread bit.
                   --Let a dummy write occur
                   --Note that a drop should occur with fifo10_drop='1'
                   --Don't increment the wrptr, the next write will be to
                   -- the next free bit
                   --Also halt the next buffer write...the current write will
                   -- happen simaltaneous to this
                   fifo10_rd<='0';
              fifo10_rdinc<='0';
                   fifo10_drop<='1';
                   fifo10_wrinc<='0';
                   buff_halt<='1';
            end if;
          when K4E =>
            if(din=K) then
              state<=idle;
            else
              state<=SPEC;
            end if;
          when SPEC =>
            if(din=K) then
              state<=CNTRL1;
            else
```

```
                      state<=CNTRL2;
                    end if;
                  when CNTRL1 =>
                    if(din=J) then
                      state<=idle;
                    else
                      state<=CNTRL3;
                    end if;
                  when CNTRL2 =>
                    if(din=K) then
                          state<=EOP1;
                    else
                      state<=CNTRL4;
                    end if;
                  when EOP1 =>
                    if(din=J) then
                      state<=EOP2;
                    else
                      state<=idle;
                    end if;
                  when CNTRL3 =>
                    if(din=K) then
                      state<=CLR;
                    else
                          state<=SKa;
                    end if;
                  when CNTRL4 =>
                    if(din=K) then
                          state<=SE0a;
                    else
                      state<=idle;
                    end if;
                  when SE0a =>
                    if(din=K) then
                      rf_SE0<='1';
                          state<=SE0;
                    else
                      state<=idle;
                    end if;
                  when SKa =>
                    if(din=K) then
                          state<=SK;
                          rf_K<='1';
                          --start the K timeout timer
                          cnt_rst<='0';
                    end if;
-----------SOP FROM SE0------------
                  when J1A1 =>
                    if(din=J) then
                      state<=SE0;
                    else
                      state<=K1A1;
                    end if;
                  when K1A1 =>
                    if(din=K) then
                      state<=SE0;
                    else
                      state<=J2A1;
                    end if;
                  when J2A1 =>
                    if(din=J) then
                      state<=SE0;
                    else
                      state<=K2A1;
                    end if;
                  when K2A1 =>
                    if(din=K) then
                      state<=idle;
                    else
                      state<=J2A1;
                    end if;
                  when K1B1 =>
                    if(din=K) then
                      state<=SE0;
                    else
                      state<=J1A1;
                    end if;
-----------------------------------
-----------SOP FROM K------------

        when J1A2 =>
```

```
                  if(din=J) then
                    state<=SK;
                  else
                    state<=K1A2;
                  end if;
                when K1A2 =>
                  if(din=K) then
                    state<=SK;
                  else
                    state<=J2A2;
                  end if;
                when J2A2 =>
                  if(din=J) then
                    state<=SK;
                  else
                    state<=K2A2;
                  end if;
                when K2A2 =>
                  if(din=K) then
                    state<=idle;
                  else
                    state<=J2A2;
                  end if;
                when K1B2 =>
                  if(din=K) then
                    state<=SK;
                  else
                    state<=J1A2;
                  end if;
--------------------------------
      end case;


           --flop the data in on the rising edge of the clock
           flopped_din<=din;
           case fifo10state is
             when RESET =>
          if(fifo10_rf_in_en='1') then
                        --Start butting data into the fifo at bit 0
                        --on the falling edge clock data will enter the fifo
                        -- then the wrptr is incremented
                        fifo10state<=START;
                        fifo10_wr<='1';
                        fifo10_wrinc<='1';
                        fifo10_wrptrclr<='0';
               fifo10_rd<='0';
            fifo10_rdinc<='0';
               fifo10_rdptrclr<='1';
                        fifo10_drop<='0';
             --clk_buff_wr<='0';
                        start_buffering<='0';
              else
                        --fifo is off
                             --buffer is off
                             fifo10state<=RESET;
               fifo10_wr<='0';
            fifo10_wrinc<='0';
            fifo10_wrptrclr<='1';
               fifo10_rd<='0';
            fifo10_rdinc<='0';
               fifo10_rdptrclr<='1';
                        fifo10_drop<='0';
             --clk_buff_wr<='0'; --------------------IS THIS OK????????
                        start_buffering<='0';
                      end if;
                     when START =>
                       --the fifo lags the din by 1/2T
                       --the buffer lags the fifo by 1/2T
                       fifo10_rf_in_en<='0';
                       if(fifo10_wrptr=0) then
                         start_buffering<='1';
            if(rf_SE0='1' or rf_EOP='1' or rf_K='1' or rf_CLR='1') then
                           fifo10state<=STOP;
              fifo10_wrptrclr<='1';
              fifo10_rdptrclr<='1';
              fifo10_wr<='0';
              fifo10_rd<='0';
              fifo10_rdinc<='0';
              fifo10_wrinc<='0';
                        else
                          fifo10state<=DBLBUFF1;
```

```
                        fifo10_rdptrclr<='0';
              --read data from the start of the fifo and put into buffer
                       fifo10_rd<='1';
                       fifo10_rdinc<='1';
                            fifo10_wr<='1';
                          end if;
              elsif(fifo10_drop='1') then
            --The dropped data will be overwritten on falling_edge(clk)
                        --Resume normal fifo and buffer operations
                     fifo10_wr<='1';
            fifo10_wrinc<='1';
                        fifo10state<=START;
                        fifo10_drop<='0';
                  else
            --Incomding data enters the fifo normally
                        --The fifo write pointer is incremented after
                        -- each write occurs
                        --fifo10_wrinc controlled by drop
                        fifo10state<=START;
                  end if;
                when DBLBUFF1 =>
                  start_buffering<='1';
                  fifo10_rdptrclr<='0';
           if(rf_SE0='1' or rf_EOP='1' or rf_K='1' or rf_CLR='1') then
                        fifo10state<=STOP;
            fifo10_wrptrclr<='1';
            fifo10_rdptrclr<='1';
            fifo10_wr<='0';
            fifo10_rd<='0';
            fifo10_rdinc<='0';
            fifo10_wrinc<='0';
                  else
                        if(fifo10_drop='1') then
                          fifo10state<=DBLBUFF1;
                 --don't increment read pointer...and make fifo10_dout a high Z
                     fifo10_wr<='1';
                          fifo10_wrinc<='1';
                     fifo10_rd<='0';
                     fifo10_rdinc<='0';
                          fifo10_drop<='0';
                        else
                          fifo10state<=DBLBUFF1;
                          fifo10_rdptrclr<='0';
              --read data from the start of the fifo and put into buffer
                        fifo10_rd<='1';
                        fifo10_rdinc<='1';
                             fifo10_wr<='1';
                       end if;
                  end if;
                when STOP =>
                  --This state gives 1 extra clock cycle before RESET to allow
                  -- the last data to be written to the buffer
                  fifo10state<=RESET;
               fifo10_wrptrclr<='1';
               fifo10_rdptrclr<='1';
               fifo10_wr<='0';
               fifo10_rd<='0';
           fifo10_rdinc<='0';
           fifo10_wrinc<='0';
            end case;

    end if;
  end process state_comb;

end my_design;

--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 FIFOS Package
-- Author:               Sean Keller
--                               sjk26@cornell.edu
------------------------------------------------
-- Created:              July 30, 2003
-- Revision:     1.0 : July 30, 2003
--                               Built
-- Revision:     1.1 : July 31, 2003
--                               Added wrptr_out
-- Revision:     1.2 : August 3, 2003
--                               Made the output no longer tristated
------------------------------------------------
```

```
-- Known Bugs:    NONE
-------------------------------------------------
-- References:    For fifo heavily used:
--                                "VHDL for Programmable Logic"
--                                By: Kevin Skahill, pages 218-219
--                                Addison Wesley Publishing Inc.
--                                Menlo Park, CA 1996
-------------------------------------------------
--
-- Copyright (C) 2003     Sean Keller
--                                                      sjk26@cornell.edu
-------------------------------------------------
--------------------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;
package FIFOS is
  component fifo10 port(
    clk,rst                       : in std_logic;
    rd,wr,rdinc,wrinc             : in std_logic;
    rdptrclr,wrptrclr             : in std_logic;
         wrptr_out                : out integer range 0 to 9;
    data_in                       : in std_logic;
    data_out                      : out std_logic);
  end component;
end FIFOS;

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;
entity fifo10 is port (
  clk,rst                         : in std_logic;
  rd,wr,rdinc,wrinc               : in std_logic;
  rdptrclr,wrptrclr               : in std_logic;
  wrptr_out                       : out integer range 0 to 9;
  data_in                         : in std_logic;
  data_out                        : out std_logic);
end fifo10;

architecture archfifo of fifo10 is
  constant depth : integer := 10;
  type fifo_ary is array(depth-1 downto 0) of std_logic;
  signal fifo                     : fifo_ary;
  signal wrptr, rdptr     : integer range 0 to depth-1;
  signal dmuxout          : std_logic;
begin
  wrptr_out<=wrptr;
  reg_array:process(rst,clk)
  begin
    if (rst = '1') then
          for i in fifo'range loop
            fifo(i)<='0';
          end loop;
        elsif rising_edge(clk) then
          if (wr = '1') then
            fifo(wrptr)<=data_in;
          end if;
        end if;
  end process reg_array;

  read_count:process(rst,clk)
  begin
    if (rst = '1') then
          rdptr<=0;
        elsif rising_edge(clk) then
          if(rdptrclr = '1') then
            rdptr<=0;
          elsif (rdinc = '1') then
            rdptr<=rdptr+1;
          end if;
        end if;
  end process read_count;

  write_count:process(rst,clk)
  begin
    if (rst = '1') then
          wrptr<=0;
        elsif rising_edge(clk) then
```

```
          if (wrptrclr = '1') then
            wrptr<=0;
          elsif (wrinc = '1') then
            wrptr<=wrptr+1;
          end if;
        end if;
  end process write_count;

  --data output is multiplexed
  dmuxout<=fifo(rdptr);

  --three states
  three_state:process(rd,dmuxout)
  begin
    if (rd = '1') then
          data_out<=dmuxout;
        else
          data_out<='0';
        end if;
  end process three_state;

end archfifo;

----------------------------------------------------------------------
--------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 DPLL Package
-- Author:               Sean Keller
--                               sjk26@cornell.edu
--------------------------------------------------
-- Created:              July 2, 2003
-- Revision:     1.0 : July 2, 2003
--                               Built
-- Revision:     1.1 : July 11, 2003
--                               Changed rcv,J,K to single bit
--                               Elminated SE0
-- Revision:     1.2 : July 13, 2003
--                               Debugged DPLL with real USB
--                               Added to library and package
--                               Added low_s signal for J/K
--                               Modified usb_clk generation
--                               --added usb_clk_gen process
--                               Beautified state_comb code
--                               Tested and Working DPLL for
--                               --low-speed USB with 6Mhz clk
-- Revision:     1.3 : July 14, 2003
--                               Got Rid of J/K flip-flop
--                               Made usb_clk a buffer
--                               Cleaned up
-- Revision:     1.4 : July 16, 2003
--                               Added Fifo
-- Revision:     1.5 : July 17, 2003
--                               Removed FIFO...need external
-- Revision:     1.6 : July 25, 2003
--                               Changed low_s to full_s
-- Revision:     1.7: July 31, 2003
--                               Renamed file to DPLL_CLK
--                               Renamed usb_clk to dpll_clk
--------------------------------------------------
-- Known Bugs:   NONE
--------------------------------------------------
-- References:   For fifo heavily used:
--                               "VHDL for Programmable Logic"
--                               By: Kevin Skahill, pages 218-219
--                               Addison Wesley Publishing Inc.
--                               Menlo Park, CA 1996
--------------------------------------------------
--
-- Copyright (C) 2003    Sean Keller
--                                                  sjk26@cornell.edu
--------------------------------------------------
----------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package DPLL is
  component D_CLK port(
    rst,clk      : in std_logic;          --reset and clock inputs
    rcv          : in std_logic;          --signal to lock to
    full_s       : in std_logic;          --full-speed device, low_s='0'
    dpll_clk     : buffer std_logic);--dpll and latched clock
```

```
  end component;
end DPLL;

library ieee;
use ieee.std_logic_1164.all;
entity D_CLK is port (
  rst,clk        : in std_logic;
  rcv            : in std_logic;
  full_s : in std_logic;
  dpll_clk       : buffer std_logic);
end D_CLK;

architecture dpll_state_machine of D_CLK is
  type States is (idle,resume,sb,sf,s0,s1,s2,s3,s4,s5,s6,s7);
  signal ps, ns : States;
  signal J : std_logic;
  signal K : std_logic;
begin
  K<=not full_s;
  J<=full_s;
  state_comb:process(rst,rcv,ps,J,K)
  begin
    if(rst = '1') then
      ns<=idle;
    else
          case ps is
        when idle =>
          if(rcv = K) then
            ns<=resume;
          else
            ns<=idle;
          end if;
        when resume =>
          if(rcv = J) then
            ns<=s5;
          else
            ns<=resume;
          end if;
        when s0 =>
          if(rcv = K) then
            ns<=s1;
          else
            ns<=s5;
          end if;
        when s1 =>
          ns<=s3;
        when s2 =>
          if(rcv = K) then
            ns<=s0;
          else
            ns<=s5;
          end if;
        when s3 =>
          if(rcv = K) then
            ns<=s2;
          else
            ns<=sf;
          end if;
        when s4 =>
          if(rcv = J) then
            ns<=s5;
          else
            ns<=s1;
          end if;
        when s5 =>
          ns<=s7;
        when s6 =>
          if(rcv = J) then
            ns<=s4;
          else
            ns<=s1;
          end if;
        when s7 =>
                    if(rcv = J) then
            ns<=s6;
          else
            ns<=sb;
          end if;
        when sb =>
          ns<=s2;
        when sf =>
```

```
          ns<=s6;
      end case;
    end if;
  end process state_comb;

  dpll_clk_gen:process(ps)
  begin
    case ps is
      when idle =>
        dpll_clk<='0';
      when resume=>
        dpll_clk<='0';
      when s0 =>
        dpll_clk<='0';
      when s1 =>
                dpll_clk<='0';
      when s2 =>
                dpll_clk<='1';
      when s3 =>
             dpll_clk<='1';
      when s4 =>
                dpll_clk<='0';
      when s5 =>
                dpll_clk<='0';
      when s6 =>
        dpll_clk<='1';
      when s7 =>
        dpll_clk<='1';
      when sb =>
        dpll_clk<='1';
      when sf =>
        dpll_clk<='1';
    end case;
  end process dpll_clk_gen;

  state_clk:process(clk)
  begin
    if rising_edge(clk) then
      ps<=ns;
    end if;
  end process state_clk;

end dpll_state_machine;

--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 Clock Generator
-- Author:               Sean Keller
--                               sjk26@cornell.edu
------------------------------------------------
-- Revision:     1.0
--                               Generates Various symmetric clocks from a source clock
--                               presumably running at 6.00000MHz
------------------------------------------------
-- Known Bugs:   NONE
------------------------------------------------
-- References:   NONE
------------------------------------------------
--
-- Copyright (C) 2004    Sean Keller
--                                                 sjk26@cornell.edu
------------------------------------------------
--------------------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;

entity CLK_GEN is
  port (
     CLK_I : in  std_logic;  -- clock
     CLK_DIV4_O : out  std_logic;
     CLK_DIV600_O: out  std_logic);
end CLK_GEN;


architecture Behaviour of CLK_GEN is
  signal cnt1 : std_logic_vector(8 downto 0);
  signal cnt2 : std_logic_vector(0 downto 0);
```

```
begin
  counter_100us:process(CLK_I,CLK_DIV4_O,CLK_DIV600_O,cnt1,cnt2)
  begin
  if rising_edge(CLK_I) then
        if(cnt1="100101100") then --this is decimal 300
          cnt1<="000000000";
          CLK_DIV600_O<=not CLK_DIV600_O;
    else
          cnt1<=cnt1+1;
        end if;
        if(cnt2="1") then
          cnt2<="0";
          CLK_DIV4_O<=not CLK_DIV4_O;
        else
          cnt2<=cnt2+1;
        end if;
  end if;
  end process;
end Behaviour;

----------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 Device Connection Code
-- Author:               Sean Keller
--                                 sjk26@cornell.edu
------------------------------------------------
-- Revision:     1.0
--                                 Determins device attachments state
--                                 and device speed
------------------------------------------------
-- Known Bugs:   NONE
------------------------------------------------
-- References:   USB v1.1 Spec (www.usb.org)
------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                                                 sjk26@cornell.edu
------------------------------------------------
----------------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;

entity DEVICE_CONNECT is port(
  rst,clk                 : in std_logic;           --reset and 4x clk
  dplu                    : in std_logic;           --usb D+
  dmin                    : in std_logic;           --usb D-
  ATTACHED_O     : out std_logic; --device connected
  FULLS_O                 : out std_logic);--Full speed/Low speed
end DEVICE_CONNECT;

architecture Behave of DEVICE_CONNECT is
  type States1 is (RESET,A,B,CONNECTED);
  signal state1           : States1;
  signal attached: std_logic;
  signal fulls            : std_logic;
  signal SE0cnt           : std_logic_vector(4 downto 0);
begin
  ATTACHED_O<=attached;
  FULLS_O<=fulls;
  connect_detect: process(clk,rst,dplu,dmin,attached,fulls,SE0cnt,state1)
  begin
    if(rst='1') then
          attached<='0';
          fulls<='0';
          state1<=RESET;
          SE0cnt<="00000";
        elsif rising_edge(clk) then
     if (attached='0') then
           case state1 is
           when RESET =>
             attached<='0';
             if((dplu='0') and (dmin='0')) then
                   state1<=RESET;
                else
                  state1<=A;
                  fulls<=dplu;
             end if;
```

```
            when A =>
              if((dplu='0') and (dmin='0')) then
                    state1<=RESET;
                  elsif(fulls=dplu) then
                    state1<=B;
                  else
                    state1<=RESET;
                  end if;
            when B =>
              if((dplu='0') and (dmin='0')) then
                    state1<=RESET;
                  elsif(fulls=dplu) then
                    state1<=CONNECTED;
                    attached<='1';
                  else
                    state1<=RESET;
                  end if;
            when CONNECTED =>
              state1<=CONNECTED;
            end case;
          elsif (attached='1') then
            if(SE0cnt="10010") then       --dec18=3us
              SE0cnt<="00000";
                  attached<='0';
                  state1<=RESET;
            elsif((dplu='0') and (dmin='0')) then
              SE0cnt<=SE0cnt+1;
            else
              SE0cnt<="00000";
            end if;
          end if;
    end if;
  end process;
end Behave;

----------------------------------------------------------------------
----------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 Downstream Main Module
-- Author:               Sean Keller
--                               sjk26@cornell.edu
----------------------------------------------------
-- Revision:     1.0
--                               Implements the main downstream hardware
----------------------------------------------------
-- Known Bugs:   NONE
----------------------------------------------------
-- References:   USB v1.1 Spec (www.usb.org)
----------------------------------------------------
--
-- Copyright (C) 2003    Sean Keller
--                                                   sjk26@cornell.edu
----------------------------------------------------
----------------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;

entity simplelogic is port(
  ACK_I                   : in std_logic;           --MCU ACK input
  ACK_O                   : out std_logic; --MCU ACK outupt
  reset                   : in std_logic;           --GLOBAL RESET
  CLK_6000K               : in std_logic;           --6.0MHZ clock in
  mcu_rxd                 : out std_logic; --MCU's UART RX pin
  dout_rf                 : out std_logic; --data out to the RFTrans
  uart_txd                : in std_logic;           --MAX3221 TX pin
  uart_rxd                : out std_logic; --MAX3221 RX pin
  mcu_txd                 : in std_logic;           --MCU's UART TX pin
  rf_rx_on                : out std_logic; --'1' for recv '0' for trans
  mcu_rf_rx_en    : in std_logic;           --MCU signal to enable RF RX
  din_rf          : in std_logic;           --incoming from data slicer
  mcu_cntrl1    : in std_logic;             --control input 0 from MCU
  mcu_cntrl2    : in std_logic;             --control input 1 form MCU
  not_oe          : out std_logic; --Controls USB output/input
  enum                    : out std_logic; --GND to turn off pullups
  vm                      : inout std_logic;
  vp                      : inout std_logic;
  usb_diff                : in std_logic;
  full_speed      : out std_logic); --MAX3451 SPD set for fullspeed
```

```
 attribute pin_avoid of simplelogic: entity is "56 156 157 158 162";
 attribute pin_numbers of simplelogic: entity is "din_rf:30 " &
                "reset:31 ACK_O:32 full_speed:33 vm:65 vp:68 dout_rf:70 not_oe:83 uart_rxd:120 " &
                "enum:122 mcu_rxd:123 mcu_rf_rx_en:126 mcu_cntrl1:133 rf_rx_on:134 "&
                "uart_txd:135 mcu_txd:137 CLK_6000K:153 ACK_I:176 usb_diff:190 mcu_cntrl2:193";
end simplelogic;

architecture my_arch of simplelogic is

component UART
  generic(BRDIVISOR: INTEGER range 0 to 65535 := 130); -- Baud rate divisor
  port(
        -- Wishbone signals
     WB_CLK_I : in  std_logic;  -- clock
     WB_RST_I : in  std_logic;  -- Reset input
     WB_ADR_I : in  std_logic_vector(1 downto 0); -- Adress bus
     WB_DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
     WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
     WB_WE_I  : in  std_logic;  -- Write Enable
     WB_STB_I : in  std_logic;  -- Strobe
     WB_ACK_O : out std_logic;     -- Acknowledge
          -- process signals
     IntTx_O  : out std_logic;  -- Transmit interrupt: indicate waiting for Byte
     IntRx_O  : out std_logic;  -- Receive interrupt: indicate Byte received
     BR_Clk_I : in  std_logic;  -- Clock used for Transmit/Receive
     TxD_PAD_O: out std_logic;  -- Tx RS232 Line
     RxD_PAD_I: in  std_logic);  -- Rx RS232 Line
  end component;

  component CLK_GEN
  port(
     CLK_I : in  std_logic;  -- clock
     CLK_DIV4_O : out  std_logic;
     CLK_DIV600_O: out  std_logic);
  end component;

  component DEVICE_CONNECT
  port(
    rst,clk              : in std_logic;          --reset and 4x clk
    dplu         : in std_logic;          --usb D+
    dmin         : in std_logic;          --usb D-
    ATTACHED_O   : out std_logic; --device connected
    FULLS_O              : out std_logic);--Full speed/Low speed
  end component;

  --various packet types
  constant CNTRL_PKT: std_logic_vector(7 downto 0) := "01010101";
  constant USB_PKT        : std_logic_vector(7 downto 0) := "10101010";
  constant LS             : std_logic_vector(7 downto 0) := "10100011";
  constant FS             : std_logic_vector(7 downto 0) := "10001000";
  constant EOT_PKT        : std_logic_vector(7 downto 0) := "01110010";
  constant RESET_D        : std_logic_vector(7 downto 0) := "11001001";

  --main States
  type MainStates is     (RST,ATTACH1,ATTACH2,ATTACH3,ATTACH4,ATTACH5,
                                                ATTACH6,ATTACH7);
  signal ps                 : MainStates;  --present state
  signal WB_ADR_O : std_logic_vector(1 downto 0);
  signal WB_DAT_O : std_logic_vector(7 downto 0);
  signal WB_DAT_I : std_logic_vector(7 downto 0);
  signal WB_WE_O  : std_logic;
  signal WB_STB_O : std_logic;
  signal WB_ACK_I : std_logic;
  signal WBwrite  : std_logic;
  signal WBwriteACK : std_logic;
  signal IntTx_I  : std_logic;
  signal IntRx_I  : std_logic;
  signal TxD_PAD_O: std_logic;  -- Tx RS232 Line
  signal RxD_PAD_I: std_logic;  -- Rx RS232 Line
  signal device_attached : std_logic;
  signal CLK_10K  : std_logic;
  signal CLK_1500K: std_logic;
  signal fullspeed: std_logic;
  signal ackO     : std_logic;
  --signal tempcnt        : std_logic_vector(7 downto 0);
begin
  rf_rx_on<=mcu_rf_rx_en; --this is no longer used and just passes along
  enum<='0';                                --GND to turn off pullups Vdd to turn on pullups
  WB_ADR_O <= "00";
  full_speed<=fullspeed;
```

```
ACK_O<=ackO;

Uart_MAP : UART
generic map(BRDIVISOR => 26) --6 gives 250000bps +-0% : 26 gives 57600bps +-.16%
port map(CLK_10K,reset,WB_ADR_O,WB_DAT_O,WB_DAT_I,WB_WE_O,WB_STB_O,WB_ACK_I,

       IntTx_I,IntRx_I,CLK_6000K,TxD_PAD_O,RxD_PAD_I);
CLK_GENERATOR : CLK_GEN
port map(CLK_6000K,CLK_1500K,CLK_10K);
CONNECTION_LOGIC : DEVICE_CONNECT
port map(reset,CLK_6000K,vp,vm,device_attached,fullspeed);

--fixme
--disconnect_detected<='0';
--ACK_O<=full_speed;
--fixme

pop : process(IntTx_I,CLK_10K,CLK_6000K,reset,ps,device_attached)
begin
  if(reset='1') then
    not_oe<='1';        --Vdd to disable the USB outputs
         ps<=RST;
         WBwrite<='0';
  elsif rising_edge(CLK_6000K) then
    if(device_attached='0') then
           ps<=RST;
         end if;
         case ps is
           when RST =>
                 if(device_attached='1') then
                   ps<=ATTACH1;
                   WB_DAT_O<=CNTRL_PKT;
                        WBwrite<='1';
                 else
                   WBwrite<='0';
                   ps<=RST;
          not_oe<='1';  --Vdd to disable the USB outputs
                 end if;
               when ATTACH1 =>
                 if(WBwriteACK='1') then
                   ps<=ATTACH2;
                 else
                   ps<=ATTACH1;
                 end if;
               when ATTACH2 =>
                 if(WBwriteACK='0') then -- and tempcnt="11111100") then
                   ps<=ATTACH3;
                   if(fullspeed='1') then
                         WB_DAT_O<=FS;
                        else
                          WB_DAT_O<=LS;
                        end if;
                        WBwrite<='1';
                 else
                   ps<=ATTACH2;
                 end if;
               when ATTACH3 =>
                 if(WBwriteACK='1') then
                   ps<=ATTACH4;
                 else
                   ps<=ATTACH3;
                 end if;
               when ATTACH4 =>
                 if(WBwriteACK='0') then -- and tempcnt="11111100") then
                   ps<=ATTACH5;
                        WB_DAT_O<=EOT_PKT;
                        WBwrite<='1';
                 else
                   ps<=ATTACH4;
                 end if;
               when ATTACH5 =>
        if(WBwriteACK='1') then
                   ps<=ATTACH6;
                 else
                   ps<=ATTACH5;
                 end if;
               when ATTACH6 =>
        if(WBwriteACK='0') then
                   ps<=ATTACH7;
                        WBwrite<='0';
                 else
```

```
                    ps<=ATTACH6;
                  end if;
                when ATTACH7 =>
                  --wait for reset from host
                end case;
         end if;
  end process;

  --MCU Interface via Wishbone and UART
  --Wait for UART ready, and WBwrite set
  -- then set ACK_O and wait for ACK_I to
  -- go high then blast the data out the
  -- UART and clear the flags
  WB:process(CLK_10K,reset,WBwrite,ackO,ACK_I)
  begin
    if(reset='1') then
           WBwriteACK<='0';
      WB_WE_O <= '1';  --DATA is going out
      WB_DAT_O <= "00000000";
           WBwriteACK <= '0';
           ackO <= '0';
    elsif rising_edge(CLK_10K) then
           if(IntTx_I='1' and WBwriteACK='0' and WBwrite='1') then
        if(ackO='0') then
              ackO<='1';
                end if;
                if(ACK_I='1') then
                  ackO<='0';
                  WB_WE_O<='1';
              WB_STB_O<='1';
                   WBwriteACK<='1';
                end if;
          else
            ackO<='0';   --seems like a good idea
            WB_STB_O<='0';
                WBwriteACK<='0';
          end if;
         end if;
  end process;


  --TRISTATES:
  shared_busses:process(mcu_cntrl1,mcu_cntrl2,uart_txd,mcu_txd,mcu_rf_rx_en,din_rf,TxD_PAD_O)
  begin
  if(mcu_cntrl1='0' and mcu_cntrl2='0') then
    dout_rf<='0';
    mcu_rxd<=uart_txd;
         uart_rxd<=mcu_txd;
         RxD_PAD_I<='0';
  elsif(mcu_cntrl1='1' and mcu_cntrl2='0') then
    dout_rf<=mcu_txd;
         mcu_rxd<=din_rf;
         uart_rxd<='0';
         RxD_PAD_I<='0';
  elsif(mcu_cntrl1='0' and mcu_cntrl2='1') then
    dout_rf<='0';
    mcu_rxd<='0';
         uart_rxd<=TxD_PAD_O;
         RxD_PAD_I<=uart_txd;
  elsif(mcu_cntrl1='1' and mcu_cntrl2='1') then
    dout_rf<='0';
    mcu_rxd<=TxD_PAD_O;
         uart_rxd<='0';
         RxD_PAD_I<=mcu_txd;
  end if;
  end process;
end my_arch;


--------------------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 FIFOS
-- Author:               Sean Keller
--                                 sjk26@cornell.edu
------------------------------------------------
-- Created:              July 30, 2003
-- Revision:     1.0 : July 30, 2003
--                                 Built
-- Revision:     1.1 : July 31, 2003
--                                 Added wrptr_out
```

```
-- Revision:     1.2 : August 3, 2003
--                               Made the output no longer tristated
-- Revision:     1.3 : December 27: 2004
--                               No longer a library
--                               Made fifo 16 bytes long x 1 byte wide
------------------------------------------------
-- Known Bugs:   NONE
------------------------------------------------
-- References:   For fifo heavily used:
--                               "VHDL for Programmable Logic"
--                               By: Kevin Skahill, pages 218-219
--                               Addison Wesley Publishing Inc.
--                               Menlo Park, CA 1996
------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                                                  sjk26@cornell.edu
------------------------------------------------
--------------------------------------------------------------------------------


library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;

entity fifo16x1 is
  port(
  clk,rst                         : in std_logic;
  rd,wr,rdinc,wrinc       : in std_logic;
  rdptrclr,wrptrclr       : in std_logic;
  wrptr_out                       : out integer range 0 to 15;
  data_in                         : in std_logic_vector(7 downto 0);
  data_out                        : out std_logic_vector(7 downto 0));
end fifo16x1;

architecture archfifo of fifo16x1 is
  constant depth : integer := 16;
  type fifo_ary is array(depth-1 downto 0) of std_logic_vector(7 downto 0);
  signal fifo                     : fifo_ary;
  signal wrptr, rdptr     : integer range 0 to depth-1;
  signal dmuxout          : std_logic_vector(7 downto 0);
begin
  wrptr_out<=wrptr;
  reg_array:process(rst,clk)
  begin
    if (rst = '1') then
           for i in fifo'range loop
             fifo(i)<="00000000";
           end loop;
         elsif rising_edge(clk) then
           if (wr = '1') then
             fifo(wrptr)<=data_in;
           end if;
         end if;
  end process reg_array;

  read_count:process(rst,clk)
  begin
    if (rst = '1') then
           rdptr<=0;
         elsif rising_edge(clk) then
           if(rdptrclr = '1') then
             rdptr<=0;
           elsif (rdinc = '1') then
             rdptr<=rdptr+1;
           end if;
         end if;
  end process read_count;

  write_count:process(rst,clk)
  begin
    if (rst = '1') then
           wrptr<=0;
         elsif rising_edge(clk) then
           if (wrptrclr = '1') then
             wrptr<=0;
           elsif (wrinc = '1') then
             wrptr<=wrptr+1;
           end if;
         end if;
```

```
  end process write_count;

  --data output is multiplexed
  dmuxout<=fifo(rdptr);

  --three states
  three_state:process(rd,dmuxout)
  begin
    if (rd = '1') then
          data_out<=dmuxout;
        else
          data_out<="00000000";
        end if;
  end process three_state;

end archfifo;

----------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- File:         USB v1.1 Upstream Main Module
-- Author:               Sean Keller
--                               sjk26@cornell.edu
-------------------------------------------------
-- Revision:     1.0
--                               Implements the main upstream hardware
-------------------------------------------------
-- Known Bugs:   NONE
-------------------------------------------------
-- References:   USB v1.1 Spec (www.usb.org)
-------------------------------------------------
--
-- Copyright (C) 2004    Sean Keller
--                                                 sjk26@cornell.edu
-------------------------------------------------
----------------------------------------------------------------------

library ieee;
library cypress;
use ieee.std_logic_1164.all;
use cypress.std_arith.all;

entity simplelogic is port(
  ACK_I                 : in std_logic;          --MCU ACK input
  ACK_O                 : out std_logic; --MCU ACK outupt
  reset                 : in std_logic;          --GLOBAL RESET
  CLK_6000K             : in std_logic;          --6.0MHZ clock in
  mcu_rxd               : out std_logic; --MCU's UART RX pin
  dout_rf               : out std_logic; --data out to the RFTrans
  uart_txd              : in std_logic;          --MAX3221 TX pin
  uart_rxd              : out std_logic; --MAX3221 RX pin
  mcu_txd               : in std_logic;          --MCU's UART TX pin
  rf_rx_on              : out std_logic; --'1' for recv '0' for trans
  mcu_rf_rx_en    : in std_logic;          --MCU signal to enable RF RX
  din_rf          : in std_logic;          --incoming from data slicer
  mcu_cntrl1    : in std_logic;            --control input 0 from MCU
  mcu_cntrl2    : in std_logic;            --control input 1 form MCU
  not_oe          : out std_logic; --Controls USB output/input
  enum                  : out std_logic; --GND to turn off pullups
  vm                    : inout std_logic;
  vp                    : inout std_logic;
  usb_diff              : in std_logic;
  full_speed      : out std_logic); --MAX3451 SPD set for fullspeed

 attribute pin_avoid of simplelogic: entity is "56 156 157 158 162";
 attribute pin_numbers of simplelogic: entity is "din_rf:30 " &
                "reset:31 ACK_O:32 full_speed:33 vm:65 vp:68 dout_rf:70 not_oe:83 uart_rxd:120 " &
                "enum:122 mcu_rxd:123 mcu_rf_rx_en:126 mcu_cntrl1:133 rf_rx_on:134 "&
                "uart_txd:135 mcu_txd:137 CLK_6000K:153 ACK_I:176 usb_diff:190 mcu_cntrl2:193";
end simplelogic;

architecture my_arch of simplelogic is

  component fifo16x1
  port(
    clk,rst                             : in std_logic;
    rd,wr,rdinc,wrinc     : in std_logic;
    rdptrclr,wrptrclr     : in std_logic;
    wrptr_out                   : out integer range 0 to 15;
    data_in                             : in std_logic_vector(7 downto 0);
    data_out                    : out std_logic_vector(7 downto 0));
```

```
  end component;

  component UART
  generic(BRDIVISOR: INTEGER range 0 to 65535 := 130); -- Baud rate divisor
  port(
        -- Wishbone signals
     WB_CLK_I : in  std_logic;  -- clock
     WB_RST_I : in  std_logic;  -- Reset input
     WB_ADR_I : in  std_logic_vector(1 downto 0); -- Adress bus
     WB_DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
     WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
     WB_WE_I  : in  std_logic;  -- Write Enable
     WB_STB_I : in  std_logic;  -- Strobe
     WB_ACK_O : out std_logic;     -- Acknowledge
          -- process signals
     IntTx_O  : out std_logic;  -- Transmit interrupt: indicate waiting for Byte
     IntRx_O  : out std_logic;  -- Receive interrupt: indicate Byte received
     BR_Clk_I : in  std_logic;  -- Clock used for Transmit/Receive
     TxD_PAD_O: out std_logic;  -- Tx RS232 Line
     RxD_PAD_I: in  std_logic);  -- Rx RS232 Line
  end component;

  component CLK_GEN
  port(
     CLK_I : in  std_logic;  -- clock
     CLK_DIV4_O : out  std_logic;
     CLK_DIV600_O: out  std_logic);
  end component;

  --various packet types
  constant CNTRL_PKT: std_logic_vector(7 downto 0) := "01010101";
  constant USB_PKT        : std_logic_vector(7 downto 0) := "10101010";
  constant LS             : std_logic_vector(7 downto 0) := "10100011";
  constant FS             : std_logic_vector(7 downto 0) := "10001000";
  constant EOT_PKT        : std_logic_vector(7 downto 0) := "01110010";
  constant RESET_D        : std_logic_vector(7 downto 0) := "11001001";

  --main States
  type MainStates is      (RST,ATTACH1,ATTACH2,ATTACH3,ATTACH4,
                                                 ATTACH5,ATTACH6,RESET1,RESET2,RESET3,
                                                 RESET4,RESET5,RESET6,RESET7,RESET8,
                                                 RESET9);
  signal ps                 : MainStates;  --present state
  signal WB_ADR_O : std_logic_vector(1 downto 0);
  signal WB_DAT_O : std_logic_vector(7 downto 0);
  signal WB_DAT_I : std_logic_vector(7 downto 0);
  signal WB_WE_O  : std_logic;
  signal WB_STB_O : std_logic;
  signal WB_ACK_I : std_logic;
  signal WBwrite  : std_logic;
  signal WBwriteACK : std_logic;
  signal WBread  : std_logic;
  signal WBreadACK : std_logic;
  signal IntTx_I  : std_logic;
  signal IntRx_I  : std_logic;
  signal TxD_PAD_O: std_logic;  -- Tx RS232 Line
  signal RxD_PAD_I: std_logic;  -- Rx RS232 Line
  signal device_attached : std_logic;
  signal CLK_10K  : std_logic;
  signal CLK_1500K: std_logic;
  signal fullspeed: std_logic;
  signal ackO     : std_logic;
  signal MFwrite  : std_logic;
  signal mcufifo_clk: std_logic;
  signal mcufifo_rd       : std_logic;
  signal mcufifo_wr : std_logic;
  signal mcufifo_rdinc: std_logic;
  signal mcufifo_wrinc: std_logic;
  signal mcufifo_rdpc: std_logic;
  signal mcufifo_wrpc: std_logic;
  signal mcufifo_wrpo: integer range 0 to 15;
  signal mcufifo_din: std_logic_vector(7 downto 0);
  signal mcufifo_dout: std_logic_vector(7 downto 0);
  signal derived_reset: std_logic;
  signal SE0_detected: std_logic;
  signal SE0cnt: std_logic_vector(4 downto 0);
begin
  rf_rx_on<=mcu_rf_rx_en; --this is no longer used and just passes along
  --enum<='0';                              --GND to turn off pullups Vdd to turn on pullups
  WB_ADR_O <= "00";
  full_speed<=fullspeed;
```

```
ACK_O<=ackO;

mcufifo : fifo16x1
port map(mcufifo_clk,reset,mcufifo_rd,mcufifo_wr,mcufifo_rdinc,mcufifo_wrinc,
                      mcufifo_rdpc,mcufifo_wrpc,mcufifo_wrpo,mcufifo_din,mcufifo_dout);

Uart_MAP : UART
generic map(BRDIVISOR => 26) --6 gives 250000bps +-0% : 26 gives 57600bps +-.16%
port map(CLK_10K,reset,WB_ADR_O,WB_DAT_O,WB_DAT_I,WB_WE_O,WB_STB_O,WB_ACK_I,

       IntTx_I,IntRx_I,CLK_6000K,TxD_PAD_O,RxD_PAD_I);
CLK_GENERATOR : CLK_GEN
port map(CLK_6000K,CLK_1500K,CLK_10K);

--fixme
--disconnect_detected<='0';
--ACK_O<=full_speed;
--fixme

pop : process(IntTx_I,CLK_10K,CLK_6000K,reset,ps)
begin
  if(reset='1') then
    not_oe<='1';        --Vdd to disable the USB outputs
         ps<=RST;
         WBread<='0';
    derived_reset<='1';
         enum<='0';              --no device attached
         fullspeed<='0';--set to lowspeed device
         MFwrite<='0';
         WBwrite<='0';
         WB_DAT_O<="00000000";
  elsif rising_edge(CLK_6000K) then
         case ps is
           when RST =>
             WB_DAT_O<="00000000";
                  WBwrite<='0';
                  MFwrite<='0';
        not_oe<='1';    --Vdd to disable the USB outputs
            derived_reset<='1';
            enum<='0';          --no device attached
            fullspeed<='0';     --set to lowspeed device
                 if(ACK_I='1') then
                   ps<=ATTACH1;
                        WBread<='1';
                 else
                   WBread<='0';
                   ps<=RST;
                 end if;
      when ATTACH1 =>
                 derived_reset<='0';
                 if(WBreadACK='1') then
                   WBread<='0';
                   ps<=ATTACH2;
                 else
                   WBread<='1';
                   ps<=ATTACH1;
                 end if;
               when ATTACH2 =>
                 if(IntRx_I='1') then
                   if(WB_DAT_I=CNTRL_PKT) then
                          ps<=ATTACH3;
                          WBread<='1';
                        else
                          ps<=RST;
                          WBread<='0';
                        end if;
                 else
                   ps<=ATTACH2;
                        WBread<='0';
                 end if;
               when ATTACH3 =>
                 if(ACK_I='1') then
                   ps<=ATTACH4;
                        WBread<='1';
                 else
                   WBread<='0';
                   ps<=ATTACH3;
                 end if;
               when ATTACH4 =>
                 if(WBreadACK='1') then
                   WBread<='0';
```

```
          ps<=ATTACH5;
        else
          WBread<='1';
          ps<=ATTACH4;
        end if;
      when ATTACH5 =>
        if(IntRx_I='1') then
          if(WB_DAT_I=LS) then
                ps<=ATTACH6;
                WBread<='1';
                fullspeed<='0';
                device_attached<='1';
              elsif(WB_DAT_I=FS) then
                ps<=ATTACH6;
                WBread<='1';
                fullspeed<='1';
                device_attached<='1';
              else
                ps<=RST;
                WBread<='0';
              end if;
        else
          ps<=ATTACH5;
               WBread<='0';
        end if;
        --WBread<='0';
      when ATTACH6 =>
        --device is now attached
        enum<='1';--turn on pullups
        if(SE0_detected='1') then
          ps<=ATTACH6;
               --ps<=RESET1;
        else
          ps<=ATTACH6;
        end if;
      when RESET1 =>
        WBwrite<='1';
        if(WBwriteACK='1') then
          ps<=RESET2;
        else
          ps<=RESET1;
        end if;
      when RESET2 =>
        if(WBwriteACK='0') then
          ps<=RESET3;
               WB_DAT_O<=USB_PKT;
               WBwrite<='1';
        else
          ps<=RESET2;
        end if;
      when RESET3 =>
if(WBwriteACK='1') then
          ps<=RESET4;
        else
          ps<=RESET3;
        end if;
      when RESET4 =>
        if(WBwriteACK='0') then
          ps<=RESET5;
               WB_DAT_O<=RESET_D;
               WBwrite<='1';
        else
          ps<=RESET4;
        end if;
      when RESET5 =>
if(WBwriteACK='1') then
          ps<=RESET6;
        else
          ps<=RESET5;
        end if;
      when RESET6 =>
        if(WBwriteACK='0') then
          ps<=RESET7;
               WB_DAT_O<=EOT_PKT;
               WBwrite<='1';
        else
          ps<=RESET6;
        end if;
      when RESET7 =>
if(WBwriteACK='1') then
          ps<=RESET8;
```

```
                else
                  ps<=RESET7;
                end if;
              when RESET8 =>
       if(WBwriteACK='0') then
                  ps<=RESET9;
                       WBwrite<='0';
                else
                  ps<=RESET8;
                end if;
              when RESET9 =>
                --wait for host to send setup
              end case;
       end if;
end process;

SE0detecter:process(CLK_6000K,derived_reset,device_attached)
begin
 if(derived_reset='1') then
   device_attached<='0';
        SE0_detected<='0';
        SE0cnt<="00000";
 elsif(rising_edge(CLK_6000K)) then
   if(device_attached='1') then
     if(SE0cnt="10010") then    --dec18=3us
           SE0cnt<="00000";
           SE0_detected<='1';
         elsif((vp='0') and (vm='0')) then
           SE0cnt<=SE0cnt+1;
         else
           SE0cnt<="00000";
                SE0_detected<='0';
         end if;
       end if;
 end if;
end process;


--for writing to the MCU fifo
MF:process(reset)
begin
  if(reset='1') then
    mcufifo_clk<='0';
    mcufifo_rd<='0';
    mcufifo_wr<='0';
    mcufifo_rdinc<='0';
    mcufifo_wrinc<='0';
    mcufifo_rdpc<='0';
    mcufifo_wrpc<='0';
    mcufifo_din<="00000000";
       end if;
end process;

--MCU Interface via Wishbone and UART
WB:process(CLK_10K,reset,WBreadACK,ackO,ACK_I,WBwrite,WBwriteACK)
begin
  if(reset='1') then
        WBreadACK<='0';
        WBwriteACK<='0';
    WB_WE_O <= '0';  --DATA is coming in
         ackO <= '0';
  elsif rising_edge(CLK_10K) then
        if(WBreadACK='0' and WBread='1') then
              if(ACK_I='1') then
                ackO<='1';
                WB_WE_O<='0';  --data is coming in
            WB_STB_O<='1';
                WBreadACK<='1';
              end if;
        elsif(IntTx_I='1' and WBwriteACK='0' and WBwrite='1') then
     if(ackO='0') then
           ackO<='1';
              end if;
              if(ACK_I='1') then
                ackO<='0';
                WB_WE_O<='1';
            WB_STB_O<='1';
                WBwriteACK<='1';
              end if;
        else
          ackO<='0';    --seems like a good idea
```

```
            WB_STB_O<='0';
                 WBwriteACK<='0';
                 WBreadACK<='0';
           end if;
         end if;
  end process;

  --TRISTATES:
  shared_busses:process(mcu_cntrl1,mcu_cntrl2,uart_txd,mcu_txd,mcu_rf_rx_en,din_rf,TxD_PAD_O)
  begin
  if(mcu_cntrl1='0' and mcu_cntrl2='0') then
    dout_rf<='0';
    mcu_rxd<=uart_txd;
         uart_rxd<=mcu_txd;
         RxD_PAD_I<='0';
  elsif(mcu_cntrl1='1' and mcu_cntrl2='0') then
    dout_rf<=mcu_txd;
         mcu_rxd<=din_rf;
         uart_rxd<='0';
         RxD_PAD_I<='0';
  elsif(mcu_cntrl1='0' and mcu_cntrl2='1') then
    dout_rf<='0';
    mcu_rxd<='0';
         uart_rxd<=TxD_PAD_O;
         RxD_PAD_I<=uart_txd;
  elsif(mcu_cntrl1='1' and mcu_cntrl2='1') then
    dout_rf<='0';
    mcu_rxd<=TxD_PAD_O;
         uart_rxd<='0';
         RxD_PAD_I<=mcu_txd;
  end if;
  end process;
end my_arch;
```

Appendix (8): Hardware UART

```
--------------------------------------------------------------------------------
-- Title      : UART
-- Project    : UART
--------------------------------------------------------------------------------
-- File        : MiniUart.vhd
-- Author      : Philippe CARTON
--               (philippe.carton2@libertysurf.fr)
-- Organization:
-- Created     : 15/12/2001
-- Last update : 8/1/2003
-- Platform    : Foundation 3.1i
-- Simulators  : ModelSim 5.5b
-- Synthesizers: Xilinx Synthesis
-- Targets     : Xilinx Spartan
-- Dependency  : IEEE std_logic_1164, Rxunit.vhd, Txunit.vhd, utils.vhd
--------------------------------------------------------------------------------
-- Description: Uart (Universal Asynchronous Receiver Transmitter) for SoC.
--    Wishbone compatable.
--------------------------------------------------------------------------------
-- Copyright (c) notice
--    This core adheres to the GNU public license
--
--------------------------------------------------------------------------------
-- Revisions       :
-- Revision Number :
-- Version         :
-- Date    :
-- Modifier        : name <email>
-- Description     :
--
--------------------------------------------------------------------------------

library ieee;
   use ieee.std_logic_1164.all;

entity UART is
  generic(BRDIVISOR: INTEGER range 0 to 65535 := 130); -- Baud rate divisor
  port (
-- Wishbone signals
     WB_CLK_I : in  std_logic;  -- clock
     WB_RST_I : in  std_logic;  -- Reset input
     WB_ADR_I : in  std_logic_vector(1 downto 0); -- Adress bus
     WB_DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
     WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
     WB_WE_I  : in  std_logic;  -- Write Enable
     WB_STB_I : in  std_logic;  -- Strobe
     WB_ACK_O : out std_logic;    -- Acknowledge
-- process signals
     IntTx_O  : out std_logic;  -- Transmit interrupt: indicate waiting for Byte
     IntRx_O  : out std_logic;  -- Receive interrupt: indicate Byte received
     BR_Clk_I : in  std_logic;  -- Clock used for Transmit/Receive
     TxD_PAD_O: out std_logic;  -- Tx RS232 Line
     RxD_PAD_I: in  std_logic);  -- Rx RS232 Line
end UART;

-- Architecture for UART for synthesis
architecture Behaviour of UART is

  component Counter
  generic(COUNT: INTEGER range 0 to 65535); -- Count revolution
  port (
     Clk      : in  std_logic;  -- Clock
     Reset    : in  std_logic;  -- Reset input
     CE       : in  std_logic;  -- Chip Enable
     O        : out std_logic); -- Output
  end component;

  component RxUnit
  port (
     Clk    : in  std_logic;  -- system clock signal
     Reset  : in  std_logic;  -- Reset input
     Enable : in  std_logic;  -- Enable input
     ReadA  : in  Std_logic;  -- Async Read Received Byte
     RxD    : in  std_logic;  -- RS-232 data input
     RxAv   : out std_logic;  -- Byte available
     DataO  : out std_logic_vector(7 downto 0)); -- Byte received
  end component;
```

```
  component TxUnit
  port (
     Clk    : in  std_logic;  -- Clock signal
     Reset  : in  std_logic;  -- Reset input
     Enable : in  std_logic;  -- Enable input
     LoadA  : in  std_logic;  -- Asynchronous Load
     TxD    : out std_logic;  -- RS-232 data output
     Busy   : out std_logic;  -- Tx Busy
     DataI  : in  std_logic_vector(7 downto 0)); -- Byte to transmit
  end component;

  signal RxData : std_logic_vector(7 downto 0); -- Last Byte received
  signal TxData : std_logic_vector(7 downto 0); -- Last bytes transmitted
  signal SReg   : std_logic_vector(7 downto 0); -- Status register
  signal EnabRx : std_logic;  -- Enable RX unit
  signal EnabTx : std_logic;  -- Enable TX unit
  signal RxAv   : std_logic;  -- Data Received
  signal TxBusy : std_logic;  -- Transmiter Busy
  signal ReadA  : std_logic;  -- Async Read receive buffer
  signal LoadA  : std_logic;  -- Async Load transmit buffer
  signal Sig0   : std_logic;  -- gnd signal
  signal Sig1   : std_logic;  -- vcc signal

  begin
  sig0 <= '0';
  sig1 <= '1';
  Uart_Rxrate : Counter -- Baud Rate adjust
     generic map (COUNT => BRDIVISOR)
     port map (BR_CLK_I, sig0, sig1, EnabRx);
  Uart_Txrate : Counter -- 4 Divider for Tx
     generic map (COUNT => 4)
     port map (BR_CLK_I, Sig0, EnabRx, EnabTx);
  Uart_TxUnit : TxUnit port map (BR_CLK_I, WB_RST_I, EnabTX, LoadA, TxD_PAD_O, TxBusy, TxData);
  Uart_RxUnit : RxUnit port map (BR_CLK_I, WB_RST_I, EnabRX, ReadA, RxD_PAD_I, RxAv, RxData);
  IntTx_O <= not TxBusy;
  IntRx_O <= RxAv;
  SReg(0) <= not TxBusy;
  SReg(1) <= RxAv;
  SReg(7 downto 2) <= "000000";

  -- Implements WishBone data exchange.
  -- Clocked on rising edge. Synchronous Reset RST_I
  WBctrl : process(WB_CLK_I, WB_RST_I, WB_STB_I, WB_WE_I, WB_ADR_I)
  variable StatM : std_logic_vector(4 downto 0);
  begin
     if Rising_Edge(WB_CLK_I) then
       if (WB_RST_I = '1') then
             ReadA <= '0';
             LoadA <= '0';
           else
             if (WB_STB_I = '1' and WB_WE_I = '1' and WB_ADR_I = "00") then -- Write Byte to Tx
          TxData <= WB_DAT_I;
               LoadA <= '1';   -- Load signal
       else LoadA <= '0';
       end if;
            if (WB_STB_I = '1' and WB_WE_I = '0' and WB_ADR_I = "00") then -- Read Byte from Rx
              ReadA <= '1';   -- Read signal
       else ReadA <= '0';
       end if;
     end if;
     end if;
  end process;
  WB_ACK_O <= WB_STB_I;
  WB_DAT_O <=
     RxData when WB_ADR_I = "00" else  -- Read Byte from Rx
     SReg when WB_ADR_I = "01" else    -- Read Status Reg
     "00000000";
end Behaviour;

-------------------------------------------------------------------------------
-- Title      : UART
-- Project    : UART
-------------------------------------------------------------------------------
-- File       : utils.vhd
-- Author     : Philippe CARTON
--              (philippe.carton2@libertysurf.fr)
-- Organization:
-- Created    : 15/12/2001
-- Last update : 8/1/2003
-- Platform   : Foundation 3.1i
-- Simulators  : ModelSim 5.5b
```

```
-- Synthesizers: Xilinx Synthesis
-- Targets     : Xilinx Spartan
-- Dependency  : IEEE std_logic_1164
--------------------------------------------------------------------------------
-- Description: VHDL utility file
--------------------------------------------------------------------------------
-- Copyright (c) notice
--    This core adheres to the GNU public license
--
--------------------------------------------------------------------------------
-- Revisions       :
-- Revision Number :
-- Version         :
-- Date    :
-- Modifier        : name <email>
-- Description     :
--
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Revision list
-- Version   Author                Date                        Changes
--
-- 1.0      Philippe CARTON  19 December 2001                  New model
--           philippe.carton2@libertysurf.fr
-- 1.1      Sean Keller      02 January 2004                                    Fixed
synchronizer
--           seankeller@yahoo.com
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Synchroniser:
--    Synchronize an input signal (C1) with an input clock (C).
--    The result is the O signal which is synchronous of C, and persist for
--    one C clock period.
--------------------------------------------------------------------------------
library IEEE,STD;
use IEEE.std_logic_1164.all;

entity synchroniser is
   port (
      C1 : in std_logic;-- Asynchronous signal
      C :  in std_logic;-- Clock
      O :  out std_logic);-- Synchronised signal
end synchroniser;

architecture Behaviour of synchroniser is
   signal C1A : std_logic;
   signal C1S : std_logic;
   signal R : std_logic;
begin
   RiseC1A : process(C1,R)
   begin
     if (R='1') then
            R<='0';
            C1A<='0';
     elsif Rising_Edge(C1) then
       C1A <= '1';
          end if;
   end process;

   SyncP : process(C,R)
   begin
     if Rising_Edge(C) then
            if (C1S = '1') then
               C1S <= '0';
                    R<='1';
            elsif (C1A = '1') then
          C1S <= '1';
       end if;
          end if;
   end process;
   O <= C1S;
end Behaviour;

--------------------------------------------------------------------------------
-- Counter
--    This counter is a parametrizable clock divider.
--    The count value is the generic parameter Count.
--    It is CE enabled. (it will count only if CE is high).
```

```
--    When it overflow, it will emit a pulse on O.
--    It can be reseted to 0.
--------------------------------------------------------------------------------
library IEEE,STD;
use IEEE.std_logic_1164.all;

entity Counter is
  generic(Count: INTEGER range 0 to 65535); -- Count revolution
  port (
     Clk      : in  std_logic;  -- Clock
     Reset    : in  std_logic;  -- Reset input
     CE       : in  std_logic;  -- Chip Enable
     O        : out std_logic); -- Output
end Counter;

architecture Behaviour of Counter is
begin
  counter : process(Clk,Reset)
     variable Cnt : INTEGER range 0 to Count-1;
  begin
     if Reset = '1' then
        Cnt := Count - 1;
        O <= '0';
     elsif Rising_Edge(Clk) then
        if CE = '1' then
           if Cnt = 0 then
              O <= '1';
              Cnt := Count - 1;
           else
              O <= '0';
              Cnt := Cnt - 1;
           end if;
        else O <= '0';
        end if;
     end if;
  end process;
end Behaviour;
--------------------------------------------------------------------------------
-- Title      : UART
-- Project    : UART
--------------------------------------------------------------------------------
-- File        : Rxunit.vhd
-- Author      : Philippe CARTON
--               (philippe.carton2@libertysurf.fr)
-- Organization:
-- Created     : 15/12/2001
-- Last update : 8/1/2003
-- Platform    : Foundation 3.1i
-- Simulators  : ModelSim 5.5b
-- Synthesizers: Xilinx Synthesis
-- Targets     : Xilinx Spartan
-- Dependency  : IEEE std_logic_1164
--------------------------------------------------------------------------------
-- Description: RxUnit is a serial to parallel unit Receiver.
--------------------------------------------------------------------------------
-- Copyright (c) notice
--    This core adheres to the GNU public license
--
--------------------------------------------------------------------------------
-- Revisions       :
-- Revision Number :
-- Version         :
-- Date    :
-- Modifier        : name <email>
-- Description     :
--
--------------------------------------------------------------------------------
library ieee;
   use ieee.std_logic_1164.all;

entity RxUnit is
  port (
     Clk    : in  std_logic;  -- system clock signal
     Reset  : in  std_logic;  -- Reset input
     Enable : in  std_logic;  -- Enable input
     ReadA  : in  Std_logic;  -- Async Read Received Byte
     RxD    : in  std_logic;  -- RS-232 data input
     RxAv   : out std_logic;  -- Byte available
     DataO  : out std_logic_vector(7 downto 0)); -- Byte received
end RxUnit;
```

```
architecture Behaviour of RxUnit is
  signal RReg    : std_logic_vector(7 downto 0); -- receive register
  signal RRegL   : std_logic;                    -- Byte received
begin
  -- RxAv process
  RxAvProc : process(RRegL,Reset,ReadA)
  begin
     if ReadA = '1' or Reset = '1' then
        RxAv <= '0';  -- Negate RxAv when RReg read
     elsif Rising_Edge(RRegL) then
        RxAv <= '1';  -- Assert RxAv when RReg written
     end if;
  end process;

  -- Rx Process
  RxProc : process(Clk,Reset,Enable,RxD,RReg)
  variable BitPos : INTEGER range 0 to 10;   -- Position of the bit in the frame
  variable SampleCnt : INTEGER range 0 to 3; -- Count from 0 to 3 in each bit
  begin
     if Reset = '1' then -- Reset
        RRegL <= '0';
        BitPos := 0;
     elsif Rising_Edge(Clk) then
        if Enable = '1' then
           case BitPos is
              when 0 => -- idle
                 RRegL <= '0';
                 if RxD = '0' then -- Start Bit
                    SampleCnt := 0;
                    BitPos := 1;
                 end if;
              when 10 => -- Stop Bit
                 BitPos := 0;    -- next is idle
                 RRegL <= '1';   -- Indicate byte received
                 DataO <= RReg;  -- Store received byte
              when others =>
                 if (SampleCnt = 1 and BitPos >= 2) then -- Sample RxD on 1
                    RReg(BitPos-2) <= RxD; -- Deserialisation
                 end if;
                 if SampleCnt = 3 then -- Increment BitPos on 3
                    BitPos := BitPos + 1;
                 end if;
           end case;
           if SampleCnt = 3 then
              SampleCnt := 0;
           else
              sampleCnt := SampleCnt + 1;
           end if;

        end if;
     end if;
  end process;
end Behaviour;

--------------------------------------------------------------------------------
-- Title      : UART
-- Project    : UART
--------------------------------------------------------------------------------
-- File        : Txunit.vhd
-- Author      : Philippe CARTON
--               (philippe.carton2@libertysurf.fr)
-- Organization:
-- Created     : 15/12/2001
-- Last update : 8/1/2003
-- Platform    : Foundation 3.1i
-- Simulators  : ModelSim 5.5b
-- Synthesizers: Xilinx Synthesis
-- Targets     : Xilinx Spartan
-- Dependency  : IEEE std_logic_1164
--------------------------------------------------------------------------------
-- Description: Txunit is a parallel to serial unit transmitter.
--------------------------------------------------------------------------------
-- Copyright (c) notice
--    This core adheres to the GNU public license
--
--------------------------------------------------------------------------------
-- Revisions       :
-- Revision Number :
-- Version         :
-- Date    :
-- Modifier        : name <email>
```

```
-- Description     :
--
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity TxUnit is
  port (
     Clk    : in  std_logic;  -- Clock signal
     Reset  : in  std_logic;  -- Reset input
     Enable : in  std_logic;  -- Enable input
     LoadA  : in  std_logic;  -- Asynchronous Load
     TxD    : out std_logic;  -- RS-232 data output
     Busy   : out std_logic;  -- Tx Busy
     DataI  : in  std_logic_vector(7 downto 0)); -- Byte to transmit
end TxUnit;

architecture Behaviour of TxUnit is

  component synchroniser
  port (
     C1 : in std_logic;    -- Asynchronous signal
     C :  in std_logic;    -- Clock
     O :  out Std_logic);-- Synchronised signal
  end component;

  signal TBuff    : std_logic_vector(7 downto 0); -- transmit buffer
  signal TReg     : std_logic_vector(7 downto 0); -- transmit register
  signal TBufL    : std_logic;  -- Buffer loaded
  signal LoadS    : std_logic;     -- Synchronised load signal

begin
  -- Synchronise Load on Clk
  SyncLoad : Synchroniser port map (LoadA, Clk, LoadS);
  Busy <= LoadS or TBufL;

  -- Tx process
  TxProc : process(Clk, Reset, Enable, DataI, TBuff, TReg, TBufL)
  variable BitPos : INTEGER range 0 to 10; -- Bit position in the frame
  begin
     if Reset = '1' then
        TBufL <= '0';
        BitPos := 0;
        TxD <= '1';
     elsif Rising_Edge(Clk) then
        if LoadS = '1' then
           TBuff <= DataI;
           TBufL <= '1';
        end if;
        if Enable = '1' then
           case BitPos is
              when 0 => -- idle or stop bit
                 TxD <= '1';
                 if TBufL = '1' then -- start transmit. next is start bit
                    TReg <= TBuff;
                    TBufL <= '0';
                    BitPos := 1;
                 end if;
              when 1 => -- Start bit
                 TxD <= '0';
                 BitPos := 2;
              when others =>
                 TxD <= TReg(BitPos-2); -- Serialisation of TReg
                 BitPos := BitPos + 1;
           end case;
           if BitPos = 10 then -- bit8. next is stop bit
              BitPos := 0;
           end if;
        end if;
     end if;
  end process;
end Behaviour;
```

Appendix (9): C Code

Downstream Code (downstream.c)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
-- Revision:     1.0
--               Implements the main even loop for the downstream WUSB
--               v1.2 ATMEGA8L MCU
-------------------------------------------------
-- Known Bugs:   INCOMPLETE
-------------------------------------------------
-- References:   NONE
-------------------------------------------------
--
-- Copyright (C) 2004    Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
---------------------------------------------------------------------*/

#include <mega8.h>
#include <stdio.h>
#include <delay.h>
#include "rf.h"
#include "globals.h"

#define Sreset 1
#define Sattach 2

void getattached();

extern unsigned char one_ms_counter;
extern unsigned char rx_data[];
extern unsigned char rx_data_len;
extern unsigned char DIRECTION;
unsigned char tx_data_valid;
unsigned char tx_data_buffer[RX_TX_BUFFER_SIZE_DIV2];
unsigned char low_speed;  //set if lowspeed device
unsigned char device_attached;    //set if idle detected
unsigned char state;

void main(){
  DIRECTION=RX;
  global_init();
  terminal_init();
  rf_init();
  UART_mode(RS232,1);
  tx_data_valid=0;
  low_speed=0;
  device_attached=0;
  putchar('>');
  delay_ms(10);
  RESET=0;
  ACK_O=0;
  state=Sreset;
  while(1){
    if(state==Sreset){
      getattached();
      if (device_attached==1){
        state=Sattach;
      }
      else{
        state=Sreset;
      }
    }
    else if(state==Sattach){
    //wait for reset
    }
  }
}

void getattached(){
    char i,k;
    unsigned char tx_data_len;
    unsigned char data_sent;
```

```
    if((device_attached==0) && (ACK_I==1)){
      i=0;
      UART_mode(UCPLD,1);
      //UART_mode(HUART,1);
      ACK_O=1;   //don't bother with flow control for each packet, so leave high until complete
      while(1){
        //put a timeout in here
        tx_data_buffer[i]=getchar();
        if (tx_data_buffer[i]==EOT_PKT){
          break;
        }
        i++;
      }
      delay_ms(100);
      UART_mode(RS232,1);
      putstringf("i=");
      putchar(i+'0');
      putstringf("\r\n");
      ACK_O=0;
      if (!(tx_data_buffer[0]==CNTRL_PKT)){ // || !(tx_data_buffer[1]==LS || tx_data_buffer[1]==FS)){
        putstringf("RESET1\r\n");
        RESET=1;
        delay_ms(10);
        RESET=0;
      }
      else{
        if(tx_data_buffer[1]==LS){
          low_speed=1;
        }
        else{
          low_speed=0;
        }
        device_attached=1;
        tx_data_len=i;
        tx_data_valid=1;
        data_sent=0;
        while(data_sent==0){
          if(rx_packet(PT_DATA,1)){
            if(*(rx_data+2)==PT_DATA_REQ){
              if(tx_data_valid){
                delay_ms(100);
                if(encode_and_send_packet(tx_data_buffer,tx_data_len,PT_DATA)){
                  data_sent=1;
                  UART_mode(RS232,1);
                  putstringf("\r\nDATA SENT CORRECTLY\r\n");
                }
                else{
                  data_sent=0;
                  device_attached=0;
                  RESET=1;
                  delay_ms(10);
                  RESET=0;
                  UART_mode(RS232,1);
                  putstringf("\r\nCOULD NOT TRANSMIT DATA\r\n");
                }
              }
              else{
                encode_and_send_packet(tx_data_buffer,0,PT_DATA);
                UART_mode(RS232,1);
                putstringf("\r\nSENDING NULL DATA\r\n");
               }
            }
          }
          else{
            UART_mode(RS232,1);
            putstringf("\r\nAWAITING DATA REQUEST\r\n");
          }
          UART_mode(RADIO,1);
        }
      }
    }
}
```

Upstream Code (upstream.c)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
```

```
------------------------------------------------
-- Revision:     1.0
--               Implements the main even loop for the upstream WUSB
--               v1.2 ATMEGA8L MCU
------------------------------------------------
-- Known Bugs:   INCOMPLETE
------------------------------------------------
-- References:   NONE
------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                        sjk26@cornell.edu
------------------------------------------------
-------------------------------------------------------------------------*/


#include <mega8.h>
#include <stdio.h>
#include <delay.h>
#include "rf.h"
#include "globals.h"

#define Sreset   1
#define Sattach  2
#define Swait1   3

void getattached();

extern unsigned char one_ms_counter;
extern unsigned char rx_data[];
extern unsigned char rx_data_len;
extern unsigned char DIRECTION;
unsigned char tx_data_valid;
unsigned char tx_data_buffer[RX_TX_BUFFER_SIZE_DIV2];
unsigned char low_speed;  //set if lowspeed device
unsigned char device_attached;    //set if idle detected
unsigned char state;
unsigned char func;

void main(){
  unsigned char i;
  DIRECTION=TX;
  global_init();
  terminal_init();
  rf_init();
  UART_mode(RS232,1);
  func=DIRECTION;
  tx_data_valid=0;
  low_speed=0;
  device_attached=0;
  putchar('>');
  delay_ms(10);
  RESET=0;
  ACK_O=0;
  state=Sreset;
  while(1){
    if(state==Sreset){
      getattached();
      if (device_attached==1){
        state=Sattach;
      }
      else{
        state=Sreset;
      }
    }
    else if(state==Sattach){
      UART_mode(UCPLD,1);
      ACK_O=1;
      while(ACK_I==0){}
      ACK_O=0;
      putchar(CNTRL_PKT);
      ACK_O=1;
      while(ACK_I==0){}
      ACK_O=0;
      if(low_speed){
        putchar(LS);
      }
      else{
        putchar(FS);
      }
      //ACK_O=1;
```

```
      //while(ACK_I==0){}
      //ACK_O=0;
      //putchar(EOT_PKT);
      delay_us(10);
      //state=Swait1;
    }
    else if(state=Swait1){
      delay_ms(200);
      UART_mode(RS232,1);
      putstringf("SENT RESET TO CPLD\r\n");

      //current mode is UCPLD
      //need to get a reset from bus
      delay_ms(10);
      UART_mode(UCPLD,1);
      ACK_O=1;   //don't bother with flow control for each packet, so leave high until complete
      while(1){
        //put a timeout in here
        tx_data_buffer[i]=getchar();
        if (tx_data_buffer[i]==EOT_PKT){
          break;
        }
        i++;
      }
      delay_ms(100);
      UART_mode(RS232,1);
      putstringf("GOT RESET:");
      putchar(tx_data_buffer[0]);
      putchar(tx_data_buffer[1]);
      putchar(tx_data_buffer[2]);
      putstringf("\r\n");
      ACK_O=0;
    }
  }
}

void getattached(){
  unsigned char i;
  if(func==REQ_DATA){
    func=TX;
    if(rx_packet(PT_DATA,3) && (*(rx_data+2)==PT_DATA)){
      //UART_mode(RS232,1);
      if(rx_data_len==HEADER_SIZE_DIV2){
        //putstringf("\r\nNO DATA READY\r\n>");
      }
      else{
        //UART_mode(RS232,1);
        //putstringf("\r\nDATA RECEIVED: ");
        for(i=HEADER_SIZE_DIV2; i<rx_data_len; i++){
          putchar(rx_data[i]);
        }
        if(*(rx_data+HEADER_SIZE_DIV2)=CNTRL_PKT){
          if(*(rx_data+HEADER_SIZE_DIV2+1)=LS){
            low_speed=1;
          }
          else{
            low_speed=0;
          }
          device_attached=1;
          UART_mode(RS232,1);
          putstringf("\r\nDevice Attached LS is:");
          putchar('0'+low_speed);
          putstringf("\r\n");
        }
      //putchar('\r');
      //putchar('\n');
      //putchar('>');
      }
    }
    else{//display error go back to tx
      //UART_mode(RS232,1);
      //putstringf("DATA REQUEST FAILED(2)\r\n>");
    }
    RF_RX_EN=0;  //TX enabled
    delay_us(125);        //wait for stable tx
  }
  else if(func==TX){
    if(encode_and_send_packet(tx_data_buffer,0,PT_DATA_REQ)){
      //UART_mode(RS232,1);
      //putstringf("\r\nDATA REQUEST SENT OK\r\n>");
      func=REQ_DATA;
```

```
    }
    else{
      //UART_mode(RS232,1);
      //putstringf("\r\nDATA REQUEST FAILED(1)\r\n>");
      func=TX;
    }
  }
  else{
    func=TX;
  }
}
```

Test Code (rs232-rf.c)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
//This is designed for the WUSB v1.2 modules
//Both boards should be connected via RS232 connections at 57600bps to an
// ANSI terminal
//The RF-link is a 250000bps UART connection to microlinear ML2724 boards
//The global #define in globals.h, DIR, must be set to RX or TX
//The TX code has a simple command interface and an '>' prompt. Typing
// 's' or 'S' will allow you to send whatever ASCI you type, terminated by
// 'Enter', to the RX board.  The maximum text length is set by MAX_CMD_SIZE.
// Note that MAX_CMD_SIZE absolutely cannot exceed RX_TX_DATA_SIZE_DIV2 -
// (HEADER_SIZE_DIV2 + len(SOP) + len(EOP)) which is currently in decimal:
// 128 - (3 + 1 + 1) = 123 bytes
//The RF channel is reliable via ACK packets, timeouts and data checksums
// (see rf.c for more details)
//The RX code dumps the received ASCI onto the RS232 connection.
//Important global routines, variables, costants, and MCU initiallization
// code is located in globals.h and globals.c
//All of the radio initializationa, protocal, and data encoding code is
// located in rf.c and rf.h
//-- Revision:   1.0
// The code works, but the data_get code is all commented out below.
// It is very close to working, but it seems to cause the boards to halt
// after a variable amount of time.  I have not determined the cause, but
// I suspect it is a stack overflow proplem
//-- Revision:   1.1
//The data_get is fixed now, type x at the tx prompt to get data, type x
// at the rx prompt until it is recognized (the polling is slow) and then
// enter the data to buffer
-------------------------------------------------
-- Known Bugs:   INCOMPLETE
-------------------------------------------------
-- References:   NONE
-------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                        sjk26@cornell.edu
-------------------------------------------------
---------------------------------------------------------------------*/

#include <mega8.h>
#include <stdio.h>
#include <delay.h>
#include "rf.h"
#include "globals.h"

extern unsigned char one_ms_counter;
extern unsigned char rx_data[];
extern unsigned char rx_data_len;
extern unsigned char DIRECTION;
unsigned char tx_data_valid;
unsigned char tx_data_buffer[RX_TX_BUFFER_SIZE_DIV2];
unsigned char low_speed;  //set if lowspeed device
unsigned char device_detected;    //set if idle detected

void main(){
  char i,k;
  unsigned char cmd[MAX_CMD_SIZE];
  unsigned char func;
  unsigned char cmdi=0;
  unsigned char cmd_ready=0;
  unsigned char cmd_len=0;
```

```
unsigned char buffering=0;
DIRECTION=DIR;
global_init();
terminal_init();
rf_init();
UART_mode(RS232,1);
func=DIRECTION;
tx_data_valid=0;
low_speed=0;
device_detected=0;
putchar('>');
while(1){
if(DIRECTION==TX){
  if(func==REQ_DATA){
    func=TX;
    if(rx_packet(PT_DATA,4) && (*(rx_data+2)==PT_DATA)){
      UART_mode(RS232,0);
      if(rx_data_len==HEADER_SIZE_DIV2){
        putstringf("\r\nNO DATA READY\r\n>");
      }
      else{
        putstringf("\r\nDATA RECEIVED: ");
        for(i=HEADER_SIZE_DIV2; i<rx_data_len; i++){
          putchar(rx_data[i]);
        }
        putchar('\r');
        putchar('\n');
        putchar('>');
      }
    }
    else{//display error go back to tx
      UART_mode(RS232,0);
      putstringf("\r\nDATA REQUEST FAILED(2)\r\n>");
    }
    RF_RX_EN=0;//TX enabled
    delay_us(125);      //wait for stable tx
  }
  else if((func==TX)||(func=SEND)){
    if(UCSRA & 0b10000000){              //RX done bit
      cmd[cmdi]=getchar();
      if(cmdi==MAX_CMD_SIZE-1){
        cmd[cmdi]='\r';  //force linefeed
      }
      putchar(cmd[cmdi]);
      if(cmd[cmdi]==8){  //backspace
        if(cmdi==0){
          putchar('>');
        }
        else{
          putchar(' ');
          putchar(cmd[cmdi]);
          cmdi--;
        }
      }
      else if(cmd[cmdi]=='\r'){
        putchar('\n');
        putchar('>');
        cmd[cmdi]='\0';
        cmd_ready=1;
        cmd_len=cmdi;
        cmdi=0;
      }
      else{
        cmdi++;
      }
    }
    if(cmd_ready){
      cmd_ready=0;
      switch(func){
        case(TX):
          if((cmd[0]=='S') || (cmd[0]=='s')){
            func=SEND;
            putchar(8);
            putstringf("SENDING>");
          }
          else if((cmd[0]=='X') || (cmd[0]=='x')){
            putchar(8);
            putstringf("SENDING DATA REQUEST");
            if(encode_and_send_packet(cmd,cmd_len,PT_DATA_REQ)){
              func=REQ_DATA;
            }
```

```
            else{
              putstringf("\r\nDATA REQUEST FAILED(1)\r\n>");
              func=TX;
            }
          }
          else{
            func=TX;
          }
          break;
        case(SEND):
          func=TX;
          if(encode_and_send_packet(cmd,cmd_len,PT_DATA)){
            putstringf("\r\nDATA SENT CORRECTLY");
            //putchar('0'+func);
            putstringf("\r\n>");
          }
          else{
            putstringf("\r\nCOULD NOT TRANSMIT DATA\r\n>");
          }
          break;
        default:
          func=BAD;
      }
    }
    /*
    //speed detection...low-speed if d- high, full-speed if d+ high
    if(((Dplu==1)||(Dmin==1))&&(device_detected==0)){
      low_speed=Dmin;
      device_detected=1;
    }
    if(device_detected==1){
      if(Dplu==1 || Dmin==1){
        if(low_speed==Dmin) device_detected=2;
        else device_detected=0;
      }
      else
        device_detected=0;
      }
    else if(device_detected==2){
      device_detected=3;
      if(low_speed==1){
        putstringf("LOW SPEED DEVICE DETECTED\r\n");
      }
      else{
        putstringf("FULL SPEED DEVICE DETECTED\r\n");
      }
    }*/
  }
}
else if(DIRECTION==RX){
  if(rx_packet(PT_DATA,1)){
    if(*(rx_data+2)==PT_DATA_REQ){
      if(tx_data_valid){
        delay_ms(100);
        if(encode_and_send_packet(tx_data_buffer,cmd_len,PT_DATA)){
          UART_mode(RS232,1);
          putstringf("\r\nDATA SENT CORRECTLY\r\n");
        }
        else{
          UART_mode(RS232,1);
          putstringf("\r\nCOULD NOT TRANSMIT DATA\r\n");
        }
      }
      else{
        encode_and_send_packet(tx_data_buffer,0,PT_DATA);
        UART_mode(RS232,1);
        putstringf("\r\nSENDING NULL DATA\r\n");
       }
    }
    else{
      UART_mode(RS232,1);
      putstringf("RECEIVED DATA: ");
      for(i=HEADER_SIZE_DIV2; i<rx_data_len; i++){
        putchar(rx_data[i]);
      }
      putchar('\r');
      putchar('\n');
    }
    //UART_mode(RADIO,0);
  }
```

```
    UART_mode(RS232,0);
    for(i=0;i<100;i++){
      if(UCSRA & 0b10000000) break;
      delay_us(250);
    }
    if(UCSRA & 0b10000000){       //RX done bit
        cmd[0]=getchar();
        if(cmd[0]=='x'){
          delay_ms(50);
          putstringf("BUFFER DATA>");
          buffering=1;
          while(buffering){
            tx_data_valid=0;
            if(UCSRA & 0b10000000){                //RX done bit
              cmd[cmdi]=getchar();
              if(cmdi==MAX_CMD_SIZE-1){
                cmd[cmdi]='\r';  //force linefeed
              }
              putchar(cmd[cmdi]);
              if(cmd[cmdi]==8){  //backspace
                if(cmdi==0){
                  putchar('>');
                }
                else{
                  putchar(' ');
                  putchar(cmd[cmdi]);
                  cmdi--;
                }
              }
              else if(cmd[cmdi]=='\r'){
                putchar('\n');
                putchar('>');
                cmd[cmdi]='\0';
                cmd_ready=1;
                cmd_len=cmdi;
                cmdi=0;
              }
              else{
                cmdi++;
              }
            }
            if(cmd_ready){
               cmd_ready=0;
               for(i=0;i<cmd_len;i++){
                 tx_data_buffer[i]=cmd[i];
               }
               tx_data_valid=1;
               buffering=0;
            }
          }
        }
        UART_mode(RADIO,0);
    }
    else{
      UART_mode(RADIO,0);
    }
  }
  }
}
```

Global Initialization Code (globals.c)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
-- Revision:     1.0
--               Sets up the AtMega8L MCU, initializes the UART,
--               handles the timer2 overflow interrupt
-------------------------------------------------
-- Known Bugs:   NONE
-------------------------------------------------
-- References:   NONE
-------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                        sjk26@cornell.edu
-------------------------------------------------
```

```
------------------------------------------------------------------*/


#include <mega8.h>
#include "rf.h"
#include "globals.h"

unsigned char timer_reload;
unsigned char one_ms_counter;
unsigned char MODE;
unsigned char DIRECTION;
extern unsigned char rx_buffer[];
extern unsigned char tx_buffer[];

void global_init(){
  char i;
  //SET BOARD TYPE
  //DIRECTION=DIR;
  DDRB=0b11111111;
  DDRC=0b11001110;
  PORTC=0b00000000;
  DDRD=0b11101111;
  PORTD=0b00000000;
  ACSR=0x80;      //Analog Comparator off (allows use of digital I/O on D6,7)
  CNTRL1=0;       //UART is set for RS232 COMM
  CNTRL2=0;
  MODE=RS232;
  CNTRL2=0;       //not used yet
  ACK_O=0;
  RESET=1;        //CPLD is in RESET wrt USB

  //initialize the timers
  TCCR2=0x03;     //(divide by 32) @4MHz =  8us per tick
  TIMSK=0x40;     //timer2 overflow interrupt enable
  //set the reload to be 130=255-125, 125x8us=1ms;
  timer_reload=130;

  //for debugging
  for(i=0; i<RX_TX_BUFFER_SIZE-1; i++){
    tx_buffer[i]=0;
  }
  for(i=0; i<RX_TX_BUFFER_SIZE-1; i++){
    rx_buffer[i]=0;
  }

}

void terminal_init(){
  //UBRR=(4000000L/(9600*16))-1;
  UCSRA=0b00100010;  //enable u2x
  UBRRH=0;
  UBRRL=8;// for 57.6k 25/51 for 9600 u2x0/1;
  UCSRB=0b00011000;
  UCSRC=0b10000110;  //bit 3 is set for 2 stop bits, 5 and 4 for odd parity
}

//switches the UART to and from RS232 and the Transceiver via the PLD
//the delay functionality is no longer needed and ignored
void UART_mode(unsigned char mode, unsigned char delay){
  if(MODE==mode){
    return;
  }
  else{
    MODE=mode;
    while (!(UCSRA & 0b11100000)){};
    if(delay) delay_ms(30);
    UCSRB=0b00000000;     //turn off uart
    if(mode==RS232){
      CNTRL1=0;
      CNTRL2=0;
      UCSRA=0b00100010;  //enable u2x
      UBRRH=0;
      UBRRL=8;            // for 57.6k
    }
    else if(mode==RADIO){
      CNTRL1=1;
      CNTRL2=0;
      UCSRA=0b00100000;  //disable u2x
      UBRRH=0;
      UBRRL=0;            // for 250k
    }
```

```
    else if(mode==HUART){
      CNTRL1=0;
      CNTRL2=1;
    }
    else if(mode==UCPLD){
      CNTRL1=1;
      CNTRL2=1;
      UCSRA=0b00100010;  //enable u2x
      UBRRH=0;
      UBRRL=8;            // for 57.6k
    }
    UCSRB=0b00011000;     //turn on uart
    if(delay) delay_ms(30);
  }
}

void putstringf(flash unsigned char* str){
  unsigned char index=0;
  while((str[index]!='\0') && (index<255)){
    putchar(str[index++]);
  }
}

//timer 0 overflow ISR
#pragma savereg-
interrupt [TIM2_OVF] void timer0_overflow(void){
  #asm
    push r30
    push r31
    in   r30,SREG
    push r30
  #endasm
  //reload to force 1 mSec overflow
  TCNT2=timer_reload;
  one_ms_counter++;
  #asm
    pop r30
    out SREG,r30
    pop r31
    pop r30
  #endasm
}
#pragma savereg+
```

RF Code (rf.c)

```
/*-----------------------------------------------------------------------
------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
------------------------------------------------
-- Revision:     1.0
//This contains all radio interfacing, protocal, and encoding code
//All outgoing data is manchester encoded and dumped into tx_buffer[]
//All incomding data is buffered into rx_buffer[] and manchester
// decoded into rx_data.  The manchester encoding doubles the data
// size but ensures a run length of no more then 2 zeros or ones.
//The RF frequency can be set in rf_init()
//The Header contains PID:CHKSM:TYPE
//If a packet does not get an ACK it is resent
//If a valid packet with valid checksum is received it is put into
//rx_data and the rx_data_valid flag is set
//The number of times a transmitter retries a packet is TX_RETRIES
//rx_packet blocks until receiving a valid packet or until timing out
// after the number of 100us passed in as an unsigned char
//PID is incremented circularly and used to identify each packet
//NOTE: I refferenced the publically available source code that comes
// with Micro Linear's ML2724 Start Kit.  Most of this code is
// completely mine, but I did copy the Manchester Encode and Decode
// routines along with the man_en_code[] man_de_codes[] and
// center_freq[] arrays.
------------------------------------------------
-- Known Bugs:    NONE
------------------------------------------------
-- References:    NONE
------------------------------------------------
--
-- Copyright (C) 2004     Sean Keller
--                        sjk26@cornell.edu
```

```
-------------------------------------------------
-----------------------------------------------------------------------*/


#include <mega8.h>
#include <delay.h>
#include "rf.h"
#include "globals.h"

flash unsigned char man_en_codes[] = {
  0x55, 0x56, 0x59, 0x5a, 0x65, 0x66, 0x69, 0x6a,
  0x95, 0x96, 0x99, 0x9a, 0xa5, 0xa6, 0xa9, 0xaa
  };

flash unsigned char man_de_codes[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00,
  0x00, 0x02, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00
};

//fc=center_freq[x]*1.024MHz
flash unsigned int center_freq[] = {
    2346, 2348, 2350, 2352, 2354, 2356, 2358, 2360, 2362, 2364,
    2366, 2368, 2370, 2372, 2374, 2376, 2378, 2380, 2382, 2384,
    2386, 2388, 2390, 2392, 2394, 2396, 2398, 2400, 2402, 2404,
    2406, 2408, 2410, 2412, 2414, 2416, 2418, 2420, 2422, 2424
};

//globals for the radio
unsigned char tx_buffer[RX_TX_BUFFER_SIZE];
unsigned char rx_buffer[RX_TX_BUFFER_SIZE];
unsigned char rx_data[RX_TX_BUFFER_SIZE_DIV2];
//manchester encode doubles the size of the data
// so the buffers containing the encoded datat must
// be twice the size of the data
unsigned char rx_data_len;
unsigned char rx_data_valid;
unsigned char rx_timeout;
unsigned char retries;
unsigned char PID;
unsigned char HEADER[HEADER_SIZE];//PID:CHKSUM
extern unsigned char one_ms_counter;

//This function turns on the RF boards, writes to the board's data
//registers: 0, 1, and 2, and clears/sets some globals
void rf_init(){
  unsigned int temp;
  SLEEP=1;                        //RF is not asleep (activer low)
  delay_ms(250);          //regulator startup delay
  XCEN=0;                         //RF is not powered yet
  REF_CNTRL=1;                    //RF clock on
  RF_RX_EN=1;                     //RX enabled
  EN=1;                           //3 wire serial interface idle
  DATA=0;                        //3 wire serial interface idle
  CLK=0;                          //3 wire serial interface idle
  temp=0;
  delay_ms(100);          //conservative glitch prevention
  write_rf_reg(temp);             //default val for register 0
  //default val for register 1 and freq=2482.176MHz
  write_rf_reg(temp | (center_freq[39]<<2) | 1);
  write_rf_reg(temp | 2);         //defualt val for register 2
  delay_ms(100);
  XCEN=1;                         //Turn radio on
  delay_ms(100);
  if(DIRECTION==TX) PID=33;
  else PID=55;
  rx_data_valid=0;                //No Valid data yet
  tx_data_valid=0;
}

//Writes into the ML2724 DATA 0,1,2 registers
void write_rf_reg(unsigned int val){
  char i;
  EN = 0;   //enable 3 wire serial interface
  for(i=16;i>0;i--){ //write msb to lsb
    CLK = 0;
    #asm
    nop
    nop
    nop
    nop
    #endasm
```

```
    // write bit to radio data port (MSB first)
    DATA = (val >> i-1) & 0x01;
    #asm
    nop
    nop
    nop
    nop
    #endasm
    CLK = 1;
    #asm
    nop
    nop
    nop
    nop
    #endasm
  }
  CLK = 0;
  EN = 1;        //disable 3 wire serial interface
  DATA=0;
  // if write was to channel frequency reg, allow for filter tuning
  if((val & 0x01) == 0x01){
    delay_ms(2);
  }
}

//pass in data and length to tx a packet
//returns 1 if sucessful, 0 otherwise
unsigned char encode_and_send_packet(unsigned char* data, unsigned char len, unsigned char type){
  unsigned char tx_buffer_size;
  unsigned char checksum=0;
  retries=0;
  UART_mode(RADIO,1);
  tx_buffer_size=man_encode(data, len, tx_buffer);
  //the checksum and PID are part of the header not data
  checksum=calc_checksum(data, len);
  man_encode(&PID, 1, HEADER);
  man_encode(&checksum, 1, &HEADER[2]);
  man_encode(&type, 1, &HEADER[4]);
  //send wait for ACK recurse until reaching TX_RETRIES
  while(retries<TX_RETRIES){
    RF_RX_EN=0;           //TX enabled
    delay_us(125);        //wait for stable tx
    tx_packet(tx_buffer_size);
    retries+=wait_for_ACK();
    delay_us(100);
  }
  PID++;
  UART_mode(RS232,1);
  if(retries==TX_RETRIES)
    return 0;
  else
    return 1;
}

// return a 1 if valid data received
// if the packet is an ACK, do not reply
unsigned char rx_packet(unsigned char type, unsigned char timeout_100ms){
  unsigned char sop_index = RX_TX_BUFFER_SIZE;
  unsigned char eop_index = RX_TX_BUFFER_SIZE;
  unsigned char index;
  unsigned char rx_buffer_index;
  unsigned char checksum=0;
  unsigned char newcount=0;
  rx_data_valid=0;
  UART_mode(RADIO,1);
  RF_RX_EN=1;             //RX enabled
  delay_us(125); //wait for stable rx
  rx_buffer_index=0;
  rx_timeout=0;
  one_ms_counter=0;
  if(timeout_100ms>0){
    //turn on interrupts for timer overflow counter
    #asm
      sei
    #endasm
  }
  while((RSSI==0) && (rx_timeout==0)){
    if(one_ms_counter>100){
      one_ms_counter=0;
      if(++newcount>=timeout_100ms){
        rx_timeout=1;
```

```
      }
    }
  }
  #asm
    cli
  #endasm
  while((RSSI==1) && (rx_buffer_index<RX_TX_BUFFER_SIZE-1) && (rx_timeout==0)){
    rx_buffer[rx_buffer_index++] = rx_byte();
  }

  //find index of start and end of packet markers
  // start at last byte received and work backward through buffer
  for(index=rx_buffer_index; index>0; index--){
    if(rx_buffer[index-1] == START_OF_PACKET){
      sop_index = index-1;
    }
    if(rx_buffer[index-1] == END_OF_PACKET){
      eop_index = index-1;
    }
  }

  if(sop_index == RX_TX_BUFFER_SIZE || eop_index == RX_TX_BUFFER_SIZE){
    for(index=0;index<RX_TX_BUFFER_SIZE_DIV2; index++){
      rx_data[index]=0;
    }
    rx_data_len=0;
  }
  else{
    // decode packet betwen sopIndex and eopIndex
    decode_packet(sop_index, eop_index);
    if((type==PT_DATA) || (type==PT_DATA_REQ)){
      delay_ms(10);
      //verify the checksum
      checksum=calc_checksum(&rx_data[HEADER_SIZE_DIV2],rx_data_len-HEADER_SIZE_DIV2);
      //deal with duplicate data/lost ACK
      if(rx_data[0]==PID){
        encode_and_send_ACK(rx_data[0]);
      }
      else{
        if(checksum==*(rx_data+1)){
          rx_data_valid=1;
          encode_and_send_ACK(rx_data[0]);
        }
      }
    }
  }

  if ((DIRECTION==TX) && (DEBUG==1)){
    UART_mode(RS232,1);
    putchar('P');
    putchar(PID);
    putchar('R');
    putchar('0' + retries);
    putchar('\n');
    putchar('\r');
    UART_mode(RADIO,1);
  }

  return rx_data_valid;
}


//simply sends empty packet (PID is key)
void encode_and_send_ACK(unsigned char pid){
  unsigned char tx_buffer_size;
  PID=pid;
  man_encode(&PID, 1, HEADER);
  RF_RX_EN=0;             //TX enabled
  delay_us(125); //wait for stable tx
  tx_packet(0);
  RF_RX_EN=1;             //RX enabled
  delay_us(125); //wait for stable rx
}


//wait for ACK for 200us
unsigned char wait_for_ACK(){
  rx_packet(PT_ACK,2);
  if(rx_data[0]==PID){
    return TX_RETRIES+5;
  }
```

```
  else{
    return 0x01;
  }
}

//sends a tx_packet of size tx_packet size
void tx_packet(unsigned char tx_buffer_size){
  char count;
  // send radio conditioning preamble
  count = CONDITION_PREAMBLE_LENGTH;
  while(count--){
    tx_byte(CONDITION_PREAMBLE);
  }
  // send UART synch preamble
  tx_byte(SYNC_PREAMBLE_1);
  tx_byte(SYNC_PREAMBLE_2);
  tx_byte(SYNC_PREAMBLE_3);
  tx_byte(SYNC_PREAMBLE_4);
  // send start of packet marker
  tx_byte(START_OF_PACKET);
  //Send the header PID:CHKSUM
  for(count=0;count<HEADER_SIZE;count++){
    tx_byte(HEADER[count]);
  }
  // send packet contents
  for(count=0; count<tx_buffer_size; count++){
    tx_byte(tx_buffer[count]);
  }
  // send end of packet mark
  tx_byte(END_OF_PACKET);
  // send postamble
  count = POSTAMBLE_LENGTH;
  while(count--){
    tx_byte(POSTAMBLE_BYTE);
  }
  delay_us(100);
}

//machester decode the data into rx_data
void decode_packet(unsigned char sop_index, unsigned char eop_index){
  unsigned char size;
  unsigned char i;
  size = eop_index - sop_index - 1;
  rx_data_len = man_decode((unsigned char*)&rx_buffer[sop_index+1], size);
  //clear RX buffer
  for (i=0; i<RX_TX_BUFFER_SIZE-1; i++){
    rx_buffer[i]=0;
  }
}

//blocking tx of a byte
void tx_byte(unsigned char data){
  while (!(UCSRA & 0b00100000));
  UDR=data;
}

//blocking rx of a byte
unsigned char rx_byte(){
  while(!(UCSRA & 0b10000000));
  return UDR;
}

//manchester encodes the data passed in into the tx_buffer and returns the
// new data length
unsigned char man_encode(unsigned char* data, unsigned char len, unsigned char* dest){
  unsigned char count=0;
  unsigned char index=0;
  while(count<len){
    //tx_buffer[count]=data[count];
    dest[index++] = man_en_codes[data[count] & 0x0f];
    dest[index++] = man_en_codes[(data[count]>>4) & 0x0f];
    count++;
  }
  return index;
  //return count;
}

//manchester decodes the data passed in into rx_data and returns the
// new data length
unsigned char man_decode(unsigned char* data, unsigned char len){
  unsigned char packet_index=0;
```

```
  unsigned char rx_index=0;
  unsigned char reconstruct;
  unsigned char buffer_value;
  while(rx_index < len){
    reconstruct=0;
    buffer_value = data[rx_index++];
    reconstruct |= man_de_codes[(buffer_value >> 4) & 0x0f] << 2;
    reconstruct |= man_de_codes[(buffer_value) & 0x0f];
    buffer_value = data[rx_index++];
    reconstruct |= man_de_codes[(buffer_value >> 4) & 0x0f] << 6;
    reconstruct |= man_de_codes[(buffer_value) & 0x0f] << 4;
    rx_data[packet_index++] = reconstruct;
  }
  return packet_index;
}

//calculates and returns simple sum of each data byte mod 256
unsigned char calc_checksum(unsigned char* data, unsigned char len){
  unsigned char i;
  unsigned char sum;
  sum=0;
  for(i=0; i<len; i++){
    sum+=data[i];
  }
  return sum;
}
```

RF Header (rf.h)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
-- Revision:     1.0
--               These are the critical defines for the RF boards
--               the RX_TX_BUFFER_SIZE is 255 bytes and
--               RX_TX_BUFFER_SIZE_DIV2 is 127 bytes
-------------------------------------------------
-- Known Bugs:   NONE
-------------------------------------------------
-- References:   NONE
-------------------------------------------------
--
-- Copyright (C) 2004    Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
---------------------------------------------------------------------*/

#define RX_TX_BUFFER_SIZE                  0xFF
#define RX_TX_BUFFER_SIZE_DIV2             0x79
#define START_OF_PACKET                    0xE7
#define END_OF_PACKET                      0x7E
#define CONDITION_PREAMBLE_LENGTH          0x08
#define CONDITION_PREAMBLE                 0x55
#define SYNC_PREAMBLE_LENGTH               0x04
#define SYNC_PREAMBLE_1                    0x44
#define SYNC_PREAMBLE_2                    0x11
#define SYNC_PREAMBLE_3                    0x45
#define SYNC_PREAMBLE_4                    0x15
#define POSTAMBLE_LENGTH           0x04
#define POSTAMBLE_BYTE                     0xAA
//TX_RETRIES defines the number of times a transmitter should
//re-transmit data after timing out from not receiving an ACK
#define TX_RETRIES                         0x09
//The byte length of the encoded header and true header
#define HEADER_SIZE                        0x06
#define HEADER_SIZE_DIV2           0x03
//Packet Types PT
#define PT_ACK                             0x33
#define PT_DATA                            0x44
#define PT_DATA_REQ                        0x55

//function prototypes
void rf_init();
void write_rf_reg(unsigned int);
void tx_byte(unsigned char);
unsigned char rx_byte();
```

```
void tx_packet(unsigned char);
unsigned char rx_packet(unsigned char, unsigned char);
unsigned char man_encode(unsigned char*, unsigned char, unsigned char*);
unsigned char man_decode(unsigned char*, unsigned char);
unsigned char encode_and_send_packet(unsigned char*, unsigned char, unsigned char);
void decode_packet(unsigned char, unsigned char);
unsigned char wait_for_ACK();
void encode_and_send_ACK(unsigned char);
unsigned char calc_checksum(unsigned char*, unsigned char);
```

Global Header (global.h)

```
/*---------------------------------------------------------------------
-------------------------------------------------
-- Project:              Wireless USB
-- Platform              USB v1.1w WUSB v1.2 boards
-- Author:               Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
-- Revision:     1.0
--               Contains important global defines
--               DIR (TX/RX) sets the master direction to be transmit
--                (TX) or receive (RX)
--               DEBUG 1 turns on some more RS232 output
--               MAX_CMD_SIZE is the maximum size ASCI input string and
--                it must not exceed RX_TX_DATA_SIZE_DIV2 -
--                (HEADER_SIZE_DIV2 + len(SOP) + len(EOP)) which is
--                currently in decimal: 128 - (3 + 1 + 1) = 123 bytes
-------------------------------------------------
-- Known Bugs:   NONE
-------------------------------------------------
-- References:   NONE
-------------------------------------------------
--
-- Copyright (C) 2004    Sean Keller
--                       sjk26@cornell.edu
-------------------------------------------------
---------------------------------------------------------------------*/

/* PORT CONNECTS
PC0-IN   :RSSI Analog Transmit/Receive Power
PC1-OUT  :XCEN Enables ML2724 Regulator
PC2-OUT  :SLEEP (Places ML2724 into sleep mode)
PC3-OUT  :REF_CNTRL (ENables ML2724 Clock)
PC4-IN   :USB D-
PC5-IN   :USB D+
PD0-sPE  :UART RXD
PD1-SPE  :UART TXD
PD2-OUT  :RESET
PD3-OUT  : PLD's (ACK_I) MCU's ACK_O
PD4-IN   : PLD's (ACK_O) MCU's ACK_I
PD5-OUT  :EN (ML2724 Serial Bus Enable)
PD6-OUT  :DATA (ML2724 Serial Data Line)
PD7-OUT  :CLk (ML2724 Serial Clock)
PB0-OUT  :MCU_CNTRL1
PB1-OUT  :MCU_CNTRL2
PB2-OUT  :rf_rx_en

CLKA:1.5Mhz
CLKB:6MHz
CLKC:1Mhz
CLKD:4Mhz
*/

#define DIR TX
#define DEBUG 0
#define MAX_CMD_SIZE 50
#define RSSI PINC.0
#define XCEN PORTC.1
#define SLEEP PORTC.2
#define REF_CNTRL PORTC.3
#define Dmin PINC.4
#define Dplu PINC.5
#define RESET PORTD.2
#define ACK_O PORTD.3
#define ACK_I PIND.4
#define EN PORTD.5
#define DATA PORTD.6
#define CLK PORTD.7
```

```
#define CNTRL1 PORTB.0
#define CNTRL2 PORTB.1
#define RF_RX_EN PORTB.2
#define RX 0
#define TX 1
#define SEND 2
#define REQ_DATA 3
#define BAD 255
//usb types
#define CNTRL_PKT 85
#define USB_PKT  170
#define LS 163
#define FS 136
#define EOT_PKT 114
#define RESET_D 201

/* quartz crystal frequency [Hz] */
#define xtal 4000000L
/* Baud rate */
#define baud 9600

#define RS232 1
#define RADIO 2
#define HUART 3
#define UCPLD 4

//function prototypes
void global_init();
void terminal_init();
void UART_mode(unsigned char, unsigned char);
void putstringf(flash unsigned char*);
```

References

Anderson, Don. Universal Serial Bus System Architecture. Reading, Massachusetts: Addison-Wesley Developers Press, 1997.

Compaq, Intel, Microsoft, NEC. *Universal Serial Bus Specification Revision 1.1*. 1998. [Online Book]. URL: http://www.usb.org/developers/docs/usbspec.zip.

Carton, Phillip. "MiniUART." 2003. http://www.opencores.org/projects/miniuart2/ (17 Dec. 2003).

Cypress Semiconductor Corporation. "Designing With Cypress In-System Reprogrammable (ISR) CPLDs for PC Cable Programming." May, 1999.

Cypress Semiconductor Corporation. "Using the Delta39K ISR Prototype Board." January, 2001.

"Designing a Robust USB Serial Interface Engine (SIE)." http://www.usb.org/developers/whitepapers/siewp.pdf (12 May 2003).

Hagen, Jon B. Radio-Frequency Electronics (Circuits and Applications). Cambridge, UK: Cambridge University Press, 1996.

Micro Linear. "ML2724 2.4GHz Low-IF 1.5Mbps FSK Transceiver Final Data Sheet." April, 2003.

Micro Linear. "ML2724SK-01 2.4GHz 1.5Mbps RADIO Code Base." April, 2003.

Micro Linear. "ML2724SK-01 2.4GHz 1.5Mbps RADIO Starter Kit v1.7 User's Guide." April, 2003.

Skahill, Kevin. VHDL for Programmable Logic. Menlo Park, California: Addison-Wessley Publishing Company, 1996.

(Note: Also referenced data sheets and white papers for every component used in this project)