EE 552

Final Report

Wired CDMA Network

Charlie McCarthy (0262730)
crm1@ualberta.ca

Stephen Somogyi (0372486)
ssomogyi@ee.ualberta.ca

Jessamyn Smith (0384675)
jessamyn@ee.ualberta.ca

Kevin Rudko (0264010)
krudko@ecn.ab.ca

Sebastien Durand (0346679)
sdurand@ualberta.ca

Submitted November 27, 2000

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors except as follows:

1. LCD initialization routine was adapted from references [6] & [7]
2. Test bench code was based on reference [8]
3. Mux 2_1 taken from reference [9]
4. ADC Support circuitry taken from manufacturers application notes (ADC1175-50)

Signatures

Charles McCarthy	*Stephen Somogyi*	*Sebastien Durand*
Jessamyn Smith	*Kevin Rudko*	

Abstract

CDMA (code division multiple access) allows simultaneous digital transmission by multiple devices over the same channel. Each device is assigned a unique code, known as a chip sequence, which is orthogonal to all other codes in the network. CDMA is usually done on wireless networks. However, to keep the analogue design simple, the units in this project are connected via a wired network. The design is modularized so that the code could be migrated to a wireless network with very few changes. The final implementation includes keypad input, LCD output, an encoder, a decoder, and a control entity. The chip sequences, station IDs, and modes are assigned using dipswitches. Each unit has the following mode parameters: clock speed, input source, input latch, and transmission enable. This design required 90% of the logic cells and 2% of the memory on the EPF10K20RC240-4 chip.

Table of Contents

Achievements

Hardware

The hardware simulator was designed and built on schedule and functioned reasonably well as designed. There were initial delays in the manufacturing process. An entire PCB panel was destroyed in the photo-exposing process. This was remedied by replacing the chemical batch that was responsible. The next batches worked out perfectly except for the existence of a few trace bridges that were trimmed clean. Components were ordered from various suppliers and also were received on time, except for the ADC chips that came in last minute. The assembly of the boards went quickly and painlessly and all functioned nearly perfectly. The CSAB(Cdma Signal Adding Board) had mistakes in the traces and had to be slightly altered but performed as designed once fixed.

The performance from a functional standpoint is exactly as the design intended. The circuits perform all of the proper CDMA signal manipulations and properly send this information back to the FPGA. The problem that occurs comes into play once the speed is increased past a value of 300kHz. At this frequency the noise ratio is large and the signal is lost. It is believed that ringing in the output is responsible for this occurrence. The higher the bit rate, the less time the ringing has to die out and thus drowns the signal. Additional line termination was implemented with little change. Thus the ringing appears to be linked to the feedback delay of the op-amp. Changes at the output cannot be stabilized quickly enough and the voltage oscillates.

At present time, a final finished panel is being prepared for the keypad, LCD and mode switches. This is being done to give the project a proper finished look and to protect wires from getting broken in transport and during display.

Decoder

The major difficulty in building the decoder was keeping its size down. Initially, the project design required two channels for communication, since a wireless implementation would need two frequencies. However, with a wired network, one channel is sufficient for simultaneous transmission and reception. Thus, to keep costs down, as well as conserve chip space, it was decided to implement a single channel network.

Another issue was the number of simultaneous decoder modules. The decoder is capable of simultaneous despreading of data from any number of stations. However, as this number of stations increases, the size of the decoder module increases substantially. With a final network size of three stations, and limited LCD display area, it was decided to allow simultaneous decoding from two stations only.

One of the necessary operations in the decoding process is the calculation of dot products. Normally this requires multiplication; however, with one of the operands being a chip sequence, addition/subtraction is sufficient. The initial design for this module was a simple behavioural description. In attempts to reduce chip usage, it was re-written twice structurally, each using different components. Unfortunately, in the end, neither structural design was smaller than the original behavioural model.

The final decoder is very advanced. The major space savings come from the use of on-chip RAM to store the correlator tallies. Originally, large registers were used, which is a poor use of logic cells. Also, the RAM allows increasing the tally range for each sample without greatly affecting the chip usage (e.g. increasing from 4-bit to 8-bit, for both stations, increases the design size by only 12 logic cells). Finally, the decoder is fully parameterized. At compilation time, the engineer can arbitrarily select the chip sequence length, over-sampling rate, or number of stations to simultaneously decode. These changes are effected by changing constants, require no other VHDL code changes, and are limited only by the available chip space.

LCD

The LCD proved more challenging than expected, in part because the documentation is not entirely accurate. The initialization sequence set out in the data sheets did not work. Luckily there was an application note that detailed the correct sequence. There is also the issue of space: the LCD has two lines of 16 characters each, but using the second line is not as simple as using the first. In the final implementation, only the first line is used. The LCD does not allow the input to be written to an arbitrary location on the screen, although some combination of writes and shifts should produce this effect. This being rather difficult to implement, the LCD module simply rewrites every spot on the screen during one write cycle. The final implementation does display the desired information, albeit in a more cramped format than originally intended.

Missed Goals

The original project specification called for a wireless CDMA network. It was difficult to find suitable parts, and even if acquired, would have added greatly to the complexity and cost of the project. As well, two frequencies would be required (because we can't listen on the same frequency we are transmitting on). However, since the purpose of this course is digital design, the lack of wireless communications is not necessarily a big loss.

Even after the project changed focus from wireless to wired, the intent was to use two channels. In this way, the transition to a true wireless system would be straightforward. However, due to space limitations on the FPGA, we could not implement a dual-channel architecture. For example, after optimization, the decoder for one channel uses approximately 45% of the chip. With total chip usage at 85%, there is simply not enough extra space for a second decoder.

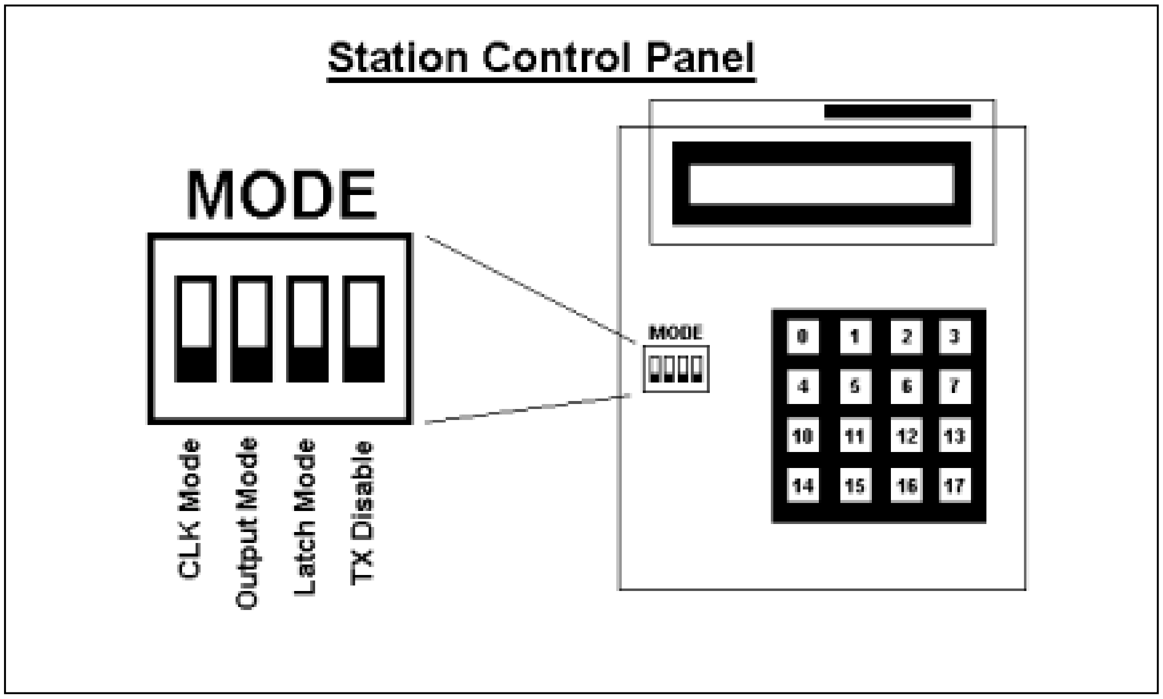
More importantly, many of the "advanced" features described in the project specification were not included due to lack of chip space and time to implement. The most significant of these was the concept of a base station. Without a base station, many of the other features were no longer applicable, such as dynamically assigned chip sequences and recovering after base station failure. However, other features such as direct peer-to-peer communication are no longer necessary, since they are counterproductive over a single communication channel (whereas they would be useful for a dual-channel implementation).

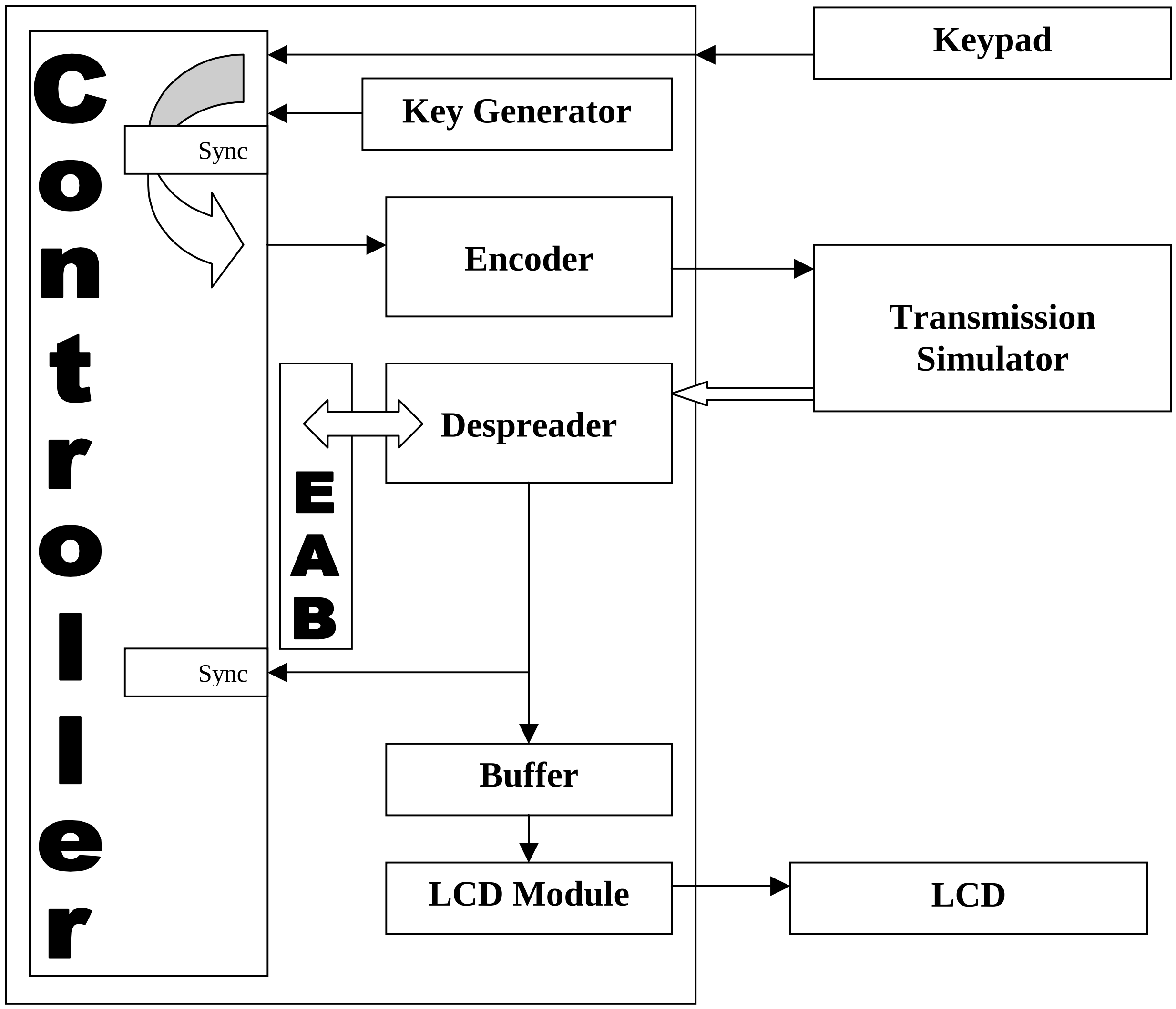
Description of Operation

Each station is capable of transmitting any Chip Sequence (CS) and receiving any CS. Each station has a control panel with a keypad and an LCD screen, for input and output. The station has four independent simultaneous modes of operation. These modes are determined with the control panel dipswitches. These modes are:

- **Clock Mode:** this mode selects transmission of data between a low and high speed. When the user changes the speed of transmission, the station automatically goes into initialization allowing the system to lock onto the new transmission speed.
- **Output Mode:** this mode selects between two kinds of output. The first being characters pressed from the keypad and second, a continuous stream of incrementing bytes.
- **Latched Mode:** while the keypad output mode is selected, selecting the latched mode will cause the chip to send the same pressed value over and over. Otherwise the value is sent only once. This mode has no effect in stream output mode.
- **TX Disable Mode:** this mode setting is available so that the number of transmitting stations can be limited. This setting is mode useful in stream mode.

Generally speaking this network is capable of almost any type of data transmission. These test cases simply prove the performance of the CDMA network. The following is the general layout of the Unit.




Controller
Sync
Keypad
Key Generator
Encoder
Transmission Simulator
Despreader
EAB
Sync
Buffer
LCD Module
LCD

Depending on the mode, a character will be obtained from the keypad or the byte-generator. On the keypad, the first 8 keys are used for data with the rest reserved for control functions. In latched mode, once a key is pressed that value will be sent continuously until another key is pressed or the mode is changed. The byte-generator uses a counter to cycle through the possible values of the 7-bit data. The data, whether from the byte-generator or keypad, is given an odd parity bit and then sent to the encoder as an 8-bit packet. There it is spread over the chip sequence as set by the dipswitches.

The transmission simulator simply takes signals that are meant to be additive (Chip sequence value of positive one and thus of the base phase) and adds them. Likewise, it takes the signals that are meant to be subtractive (Chip sequence value of –1 or 0 and thus out of phase by 180^o) and subtracts them. This sum is exactly what would happen in the air if we were able to purchase a Phase Shift Keying transmitter / receiver.

The input into the decoder comes from an Analogue to Digital Converter. We take in the most significant 4 bits to obtain the necessary resolution to be able to detect 3 different devices talking at the same time. This value is sampled at 4 times the expected transmission rate to improve synchronization. The correlator will lock on to the most probable phase and start decoding the signal. The transmitter will actually send a sequence of 20 1's and then 8 zeros (impossible with data that has odd parity) to signify the beginning of the real data.

Once the controller notices the 8 zeros, it realizes that the receiver has locked on to the signal and a byte is coming. The buffer is then allowed to start buffering bits that come from the de-spreader. These bytes are sent to the LCD and analyzed on the way for a possible error in transmission (parity). If the device is in slow mode, the LCD will display the data and the unit which sent that data. Since the LCD cannot update quickly enough to display each character in fast mode, it will display the total number of bytes received and the number of incorrect bytes.

These are trivial examples of how a network can be used. This implementation can be easily included to have wireless capability given the availability of wireless components. Due to chip size though, only one de-spreader can be implemented at a time limiting us to one frequency at a time.

CDMA Data Sheet

An individual CDMA station consists of the Altera UP1 Board, a Wired CDMA Board (WCB), and a control panel consisting of a multi-line LCD display, mode switches, and a 16 button Keypad. The design is programmed onto the FLEX10k EPF10RC240-4 Field Programmable Gate Array, using 1037 of the logic cells (90%). Each station allows for continuous transmission and reception. Up to two incoming signals may be decoded simultaneously. Multiple CDMA stations may be connected with telephone cables through the CDMA Signal Adder Board (CSAB) to form a Wired CDMA network.

Features

- 25 MHz operating frequency
- Up to 3 stations
- 4000 bps (per station)
- 5V ± 10% power supply
- Low power consumption (200 mA)

Modes may be selected using the four dipswitches on the Station Control Panel. The modes are as follows

- Clock Speed : slow (0) or fast (1)
- Output Mode : keypad (0) or internally generated stream (1)
- Latched Mode (only used for keypad input) : single transmission (0) or repeating transmission (1)
- TX Disable Mode : transmission enabled (0) and disabled (1)

In keypad output mode, the LCD displays the most recently received data along with the number of the station that sent the data. In the stream output mode, the LCD displays the number of bytes received and the number of incorrect bytes.

FPGA I/O Pins

Name	Pin Number	Direction	Description
Clock	91	Input	System clock (25 MHz)
Reset	28	Input	Active low
Rx (0..3)	118,119,116,117	Input	Receive from WCB
Clock Mode	75	Input	Slow(0) / Fast (1)
Output Mode	74	Input	Keypad (0) / Stream (1)
Latched Mode	73	Input	Latch keypad input if high (1)
TX Disable Mode	72	Input	Enable(0) / Disable(1) Transmission
Local_CS (0..1)	33,34	Input	Station chip sequence
Remote1_CS (0..1)	39,38	Input	Chip sequence of remote station 1
Remote2_CS (0..1)	41,40	Input	Chip sequence of remote station 2
KP_ROW (0..3)	67,68,70,71	Input	Keypad response
KP_COL (0..3)	63-66	Output	Keypad output signals
CDMA_LOW	109	Output	Transmit CS equivalent 0
CDMA_HIGH	110	Output	Transmit CS equivalent 1
LCD_Enable	45	Output	LCD Enable
LCD_RW	46	Output	LCD Write (0) / Read(1)
LCD_Select	48	Output	LCD Control Instructions (0) / Data Output (1)
LCD_Data(0..7)	49-51,53-56,61	Output	LCD Data
LCD_high_bit	62	Input	LCD busy signal
Sample_Clock	114	Output	ADC clock input

Total Pin Count: 38 (20 input, 18 output)

Assembling the CDMA Network

Each station consists of an Altera UP1 Board and a Wired CDMA Board (WCB). The WCB connects to the UP1 through Expansion Header B. To connect the network, attach each WCB to the CDMA Signal Adder Board via a telephone cable.

Design Details

Encoder

Overview

The encoder operates on a slower clock than the rest of the chip. It also works continuously regardless of whether or not it actually has data to send. Once a chip sequence is loaded in, the encoder immediately loads data into register A. It will continue to cycle through the chip sequence, properly handling the spreading, while the bits get shifted out of the register. When it shifts from one register to the other it loads a new value from the outside world into the register it just left. It is up to the outside world to have a byte available long enough before the taken signal is asserted.

This encoder is capable of continuous transmission.

Spreading

This is the key to CDMA. The ability for multiple devices to talk at the same time is granted so long as they send a sequence that is linearly orthogonal to the rest of the devices' sequence. By generating orthogonal codes, we can have multiple devices, using different codes, transmitting at the same time.

For example:

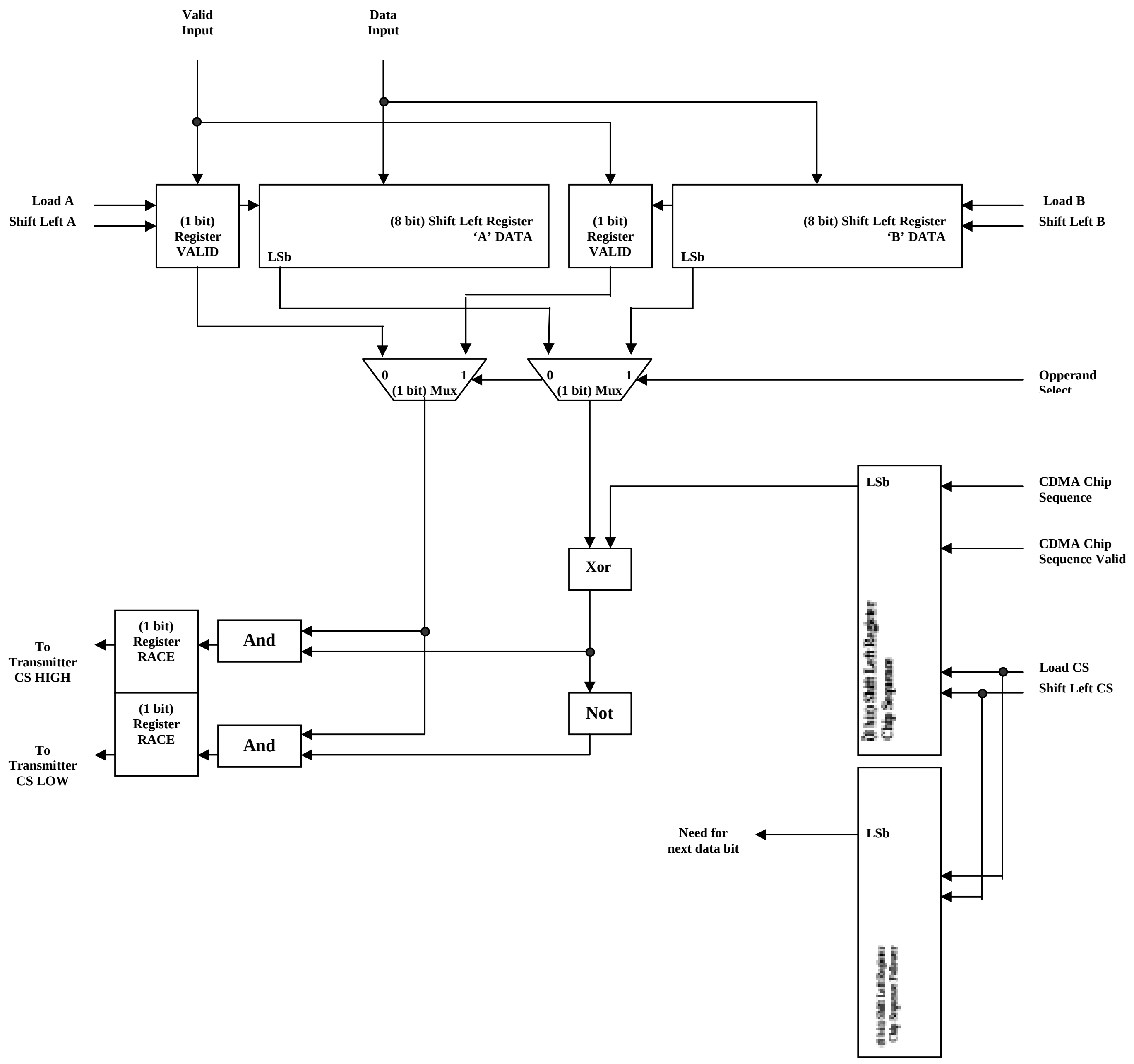
If the Chip Sequence is 0101 (only 4 bits) then a bit would be transmitted as follows:

For a logic level high, send the original chip sequence (0101)
For a logic level low, send the original chip sequence complimented (1010).

Now the actual sending of 0101 is a bit more complicated than it may seem. The "zero" bit is not the absence of a carrier signal, but rather its presence out of phase by 180^{O}. This is the key to the orthogonal nature of the codes. A "zero" must be able to cancel out a "one" transmitted from another device. Likewise, transmission of a "one" bit is the presence of the carrier frequency in phase.

So every digital bit to be transmitted is spread over the entire chip sequence and sent simultaneously with all the other devices.

Layout


Valid Input
Data Input
Load A
Shift Left A
(1 bit) Register VALID
(8 bit) Shift Left Register 'A' DATA
LSb
(1 bit) Register VALID
(8 bit) Shift Left Register 'B' DATA
LSb
Load B
Shift Left B
0
1
(1 bit) Mux
0
1
(1 bit) Mux
Opperand Select
LSb
CDMA Chip Sequence
CDMA Chip Sequence Valid
Xor
(8 bit) Shift Left Register
Chip Sequence
Load CS
Shift Left CS
To Transmitter CS HIGH
(1 bit) Register RACE
And
(1 bit) Register RACE
And
Not
To Transmitter CS LOW
Need for next data bit
LSb
(8 bit) Shift Left Register
Chip Sequence Follower

Decoder

The purpose of the decoder is to recover data from the incoming CDMA stream. There are two key phases during this process: first, bit 1 of the incoming chip sequence is identified, and second, decoded bits are generated. Before these procedures are discussed, however, some design decisions must be documented.

Several stations can transmit simultaneously, and since no attempt is made to synchronize these signals at the bit level, the input must be sampled at a higher frequency than the data is transmitted. Two times over-sampling is insufficient, as a station that is exactly 180º out of phase with the receiver will be invisible. Therefore, four-times over-sampling of the stream is used. In order to calculate the dot product, all the samples for an entire chip sequence must be saved. For such a sequence of length 8 (and 4x over-sampling), 32 sample values must be stored. A simple shift register accomplishes this.

The purpose of initial synchronization, as introduced above, is to locate the first bit of a station's chip sequence. In order words, one out of the 32 samples must be identified and used for subsequent decoding. During initial synchronization, a tally of "good" decodes is kept for each sample, and the first sample to reach a critical sum is chosen. Consider this: if all the chip sequences are orthogonal and perfectly synchronized, the dot product calculation on the correct bit will only result in one of three values: -8, 0, or +8. Thus, a "good" decode is defined as one in which the dot product is less than –6 or greater than +6. To calculate dot product, the following samples are used: 1st, 5th, 9th, 13th, 17th, 21st, 25th, and 29th (this considers one sample from every over-sampling period – the most recent sample is 1st). If a "good" decode is detected, the tally for the current sample is incremented. As soon as one sample tally reaches the critical value (in our case, 15), the initial synchronization process is complete. To conserve logic cells, these tally counts are stored in on-chip RAM.

Once the initial synchronization is complete, subsequent decoding is straightforward. Essentially, a sample count is used, and one decoded bit is output every 32 samples. In this case, the dot product calculation proceeds as before. As well, the dot product value must meet the same critical values to produce a valid bit. If this value is between –6 and +6, it is assumed that the station under scrutiny did not send a data bit at the current time, and hence no valid data bit is generated at the decoder.

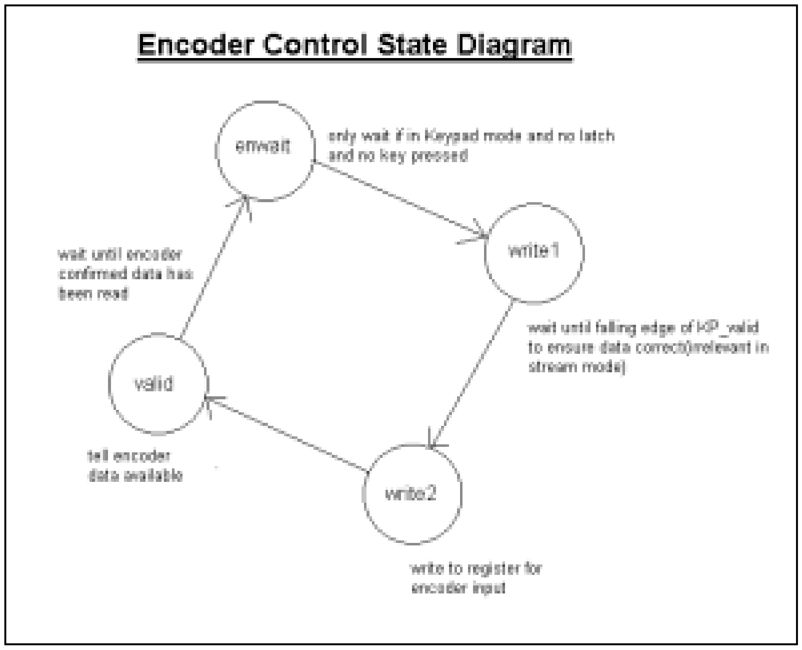
This design makes provision to decode from two stations simultaneously. The 32 most recent samples are only stored once, of course, as the data from all stations is encoded in the same samples. However, all other components – dot product calculator, tally memory, and decoding FSM – are duplicated, one for each decoded station.

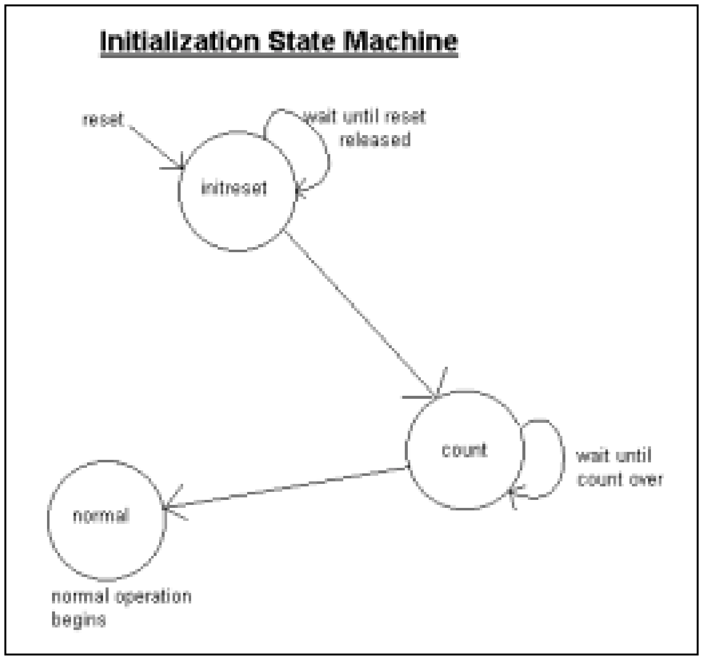
Controller

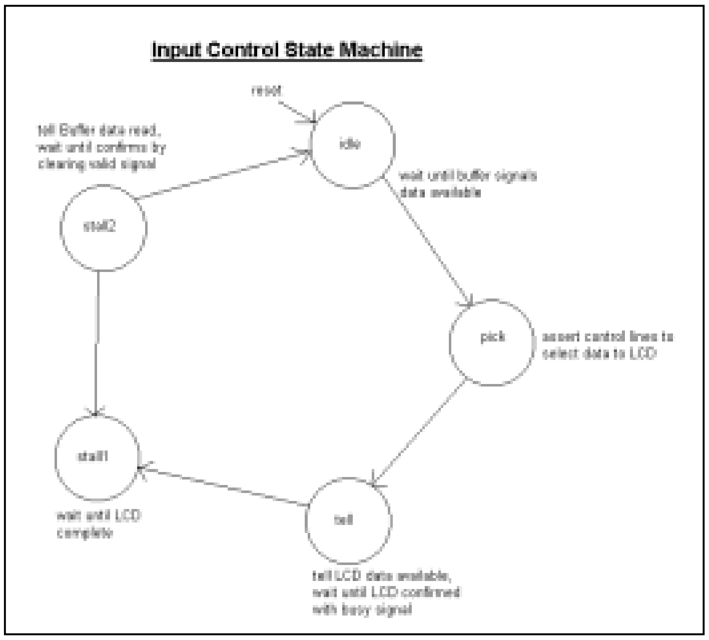
The controller is responsible for the interconnection and communication between internal modules. The controller contains a module that distributes different clock speeds for different functions carried out by different segments of the configuration. Because the user has the ability to change the transmission rate of the CDMA chip, the controller thus changes the clock rates to key modules that rely on this change of speed. The controller is also responsible for initializing transmission, both on start-up and after a user change in clock speed.

Initialization processes for transmission are simply designed to send a number of standard bytes so that the decoder circuitry can lock onto the signal. After initialization is complete, the controller then moves on to normal transmission routines. Depending on the output mode, the controller oversees the transfer of data from the source to the encoder for transmission. The source can either be the keypad or the byte counter, which is incidentally also used during initialization.

The receiver side of the chip starts receiving data right from power-up. When an initialization occurs the controller must watch for these bytes and hold off transferring this useless data to the LCD. Once the controller has found that initialization has been completed, it oversees the transfer of this data to the LCD for output. Depending on the mode of I/O, the LCD will either receive the actual values coming in from the network or the statistical data of bytes and errors since initialization. The controller will watch the parity of the incoming data and tally the number of errors that are received.


Encoder Control State Diagram
enwait
only wait if in Keypad mode and no latch and no key pressed
write1
wait until falling edge of KP_valid to ensure data correct(irrelevant in stream mode)
write2
write to register for encoder input
valid
tell encoder data available
wait until encoder confirmed data has been read




Input Control State Machine
reset
idle
wait until buffer signals data available
pick
assert control lines to select data to LCD
tell
tell LCD data available, wait until LCD confirmed with busy signal
stall1
wait until LCD complete
stall2
tell Buffer data read, wait until confirms by clearing valid signal

Hardware

At this stage of development, we are focussing on creating a functional CDMA system. Instead of attempting an RF implementation, we have decided to create a hardwired version. This section covers the different stages of this hardware.

The CDMA signal is sent out from the UP1 prototyping board into our “Wired CDMA Board” (WCB). Here, the transmitted signal is sent by wire to the “CDMA Signal Adding Board” (CSAB). Here, incoming CDMA signals from other FPGA boards are combined and transmitted back to each WCB. The WCB then converts this signal back to a digital format recognized by the FPGA configuration. All of the hardware supports two-channel use and can support up to eight stations on the network.

Wired CDMA Board (WCB)

This board incorporates both the transmitter and receiver for each CDMA station. It is designed to plug directly into one of the three UP1 prototyping headers. Power is supplied from the UP1 and further filtered to reduce noise that would reduce performance. The board layouts are given in the Appendix.

The transmitter circuit shown below, Figure H-1, converts the CDMA signal from the two digital CDMA signals +1, -1 to analogue voltages. This voltage is buffered with the LM6132 high slew-rate op-amp. At this stage of testing, the system appears to be able to run at a stream rate near 50Kbps. The potentiometer fed output ground allows the proper reference for zero output that should be 2.5V. The value of the potentiometer is low enough that the return ground has relatively low impedance.

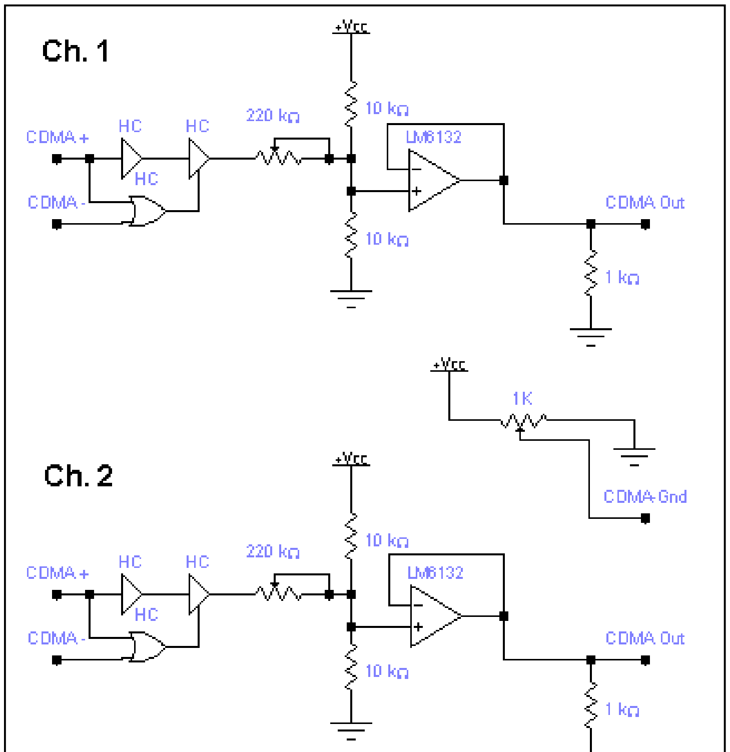


Figure H-1

Once the signal has been combined on the CSAB, it returns to the receiver side of the WCB, Figure H-2. Here the signal is buffered with another LM6132 op-amp. This signal is then fed into the ADC1175-50 8-

bit ADC. The stream decoder in the FLEX10 samples the voltage sent into the ADC and reads the value as a binary result. It is possible that the stream decoder will not use all eight bits of this information

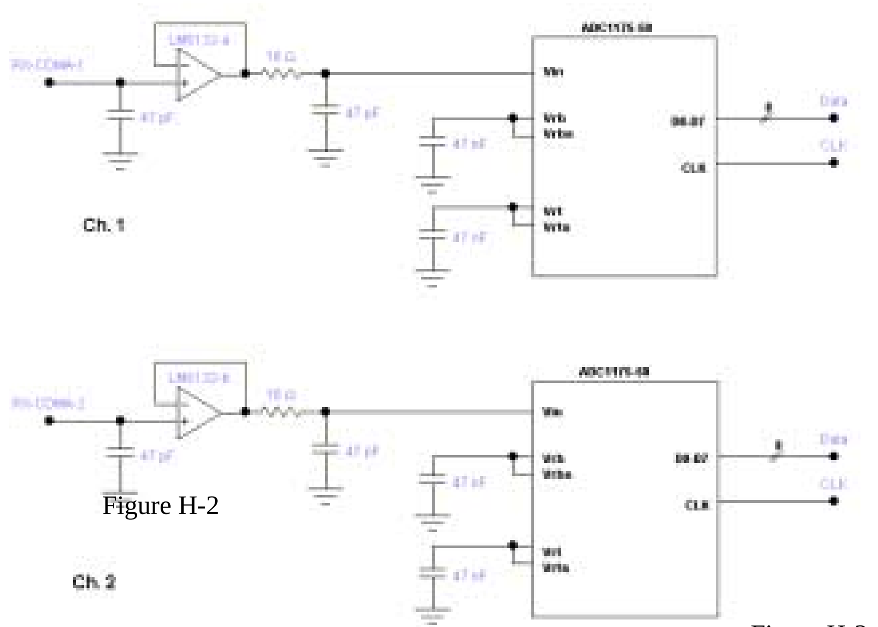
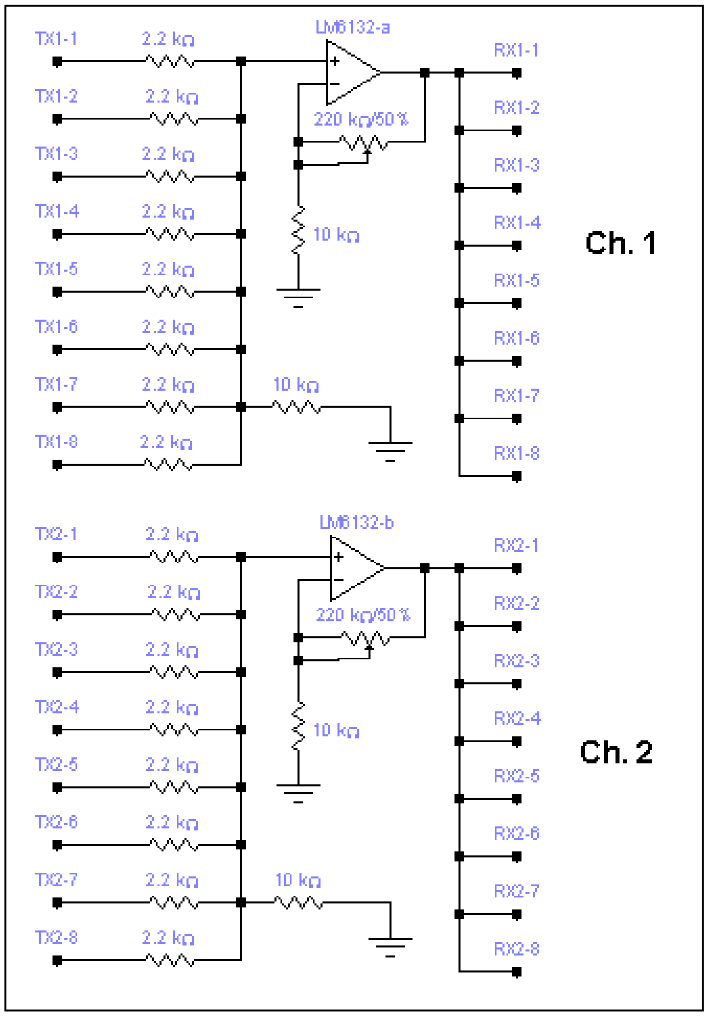


Figure H-2

Figure H-2

CDMA Signal Adder Board (CSAB)

This board is responsible of tying all of the CDMA stations together into one network. The board has inputs for up to eight stations. Each station transits a signal to the CSAB. Each signal is added together in a summing circuit using the LM6132 OP-AMP. The result, which is buffered by the OP-AMP, is then transmitted back to each WCB to be decoded. This board has its own power supply and should be isolated from the rest of the network's power supplies. The CSAB also has a potentiometer for each of the two channels to adjust the gain of the output to ensure proper decoding. See Figure H-3.


LM6132-a
TX1-1 TX1-2 TX1-3 TX1-4 TX1-5 TX1-6 TX1-7 TX1-8
2.2 kΩ
220 kΩ/50 %
10 kΩ
RX1-1 RX1-2 RX1-3 RX1-4 RX1-5 RX1-6 RX1-7 RX1-8
Ch. 1
LM6132-b
TX2-1 TX2-2 TX2-3 TX2-4 TX2-5 TX2-6 TX2-7 TX2-8
RX2-1 RX2-2 RX2-3 RX2-4 RX2-5 RX2-6 RX2-7 RX2-8
Ch. 2

Parts List

WCB

2	LM6132M	*
2	ADC1175-50CIJM	*
1	74HC32M	*
1	74HC126M	*
2	3.9uH choke	*
4	10uF tantalum capacitor	*
8	47nF film capacitor	*
4	47pF ceramic capacitor	*
4	1Ω resistor	*
2	10Ω resistor	*
4	22kΩ resistor	*
2	220KΩ 20-turn potentiometer	
1	1kΩ 20-turn potentiometer	
1	6-6 telephone jack, PC-mount	
1	72-pin dual-inline male header (cut to 60 pin)	
*	These components surface mount	

CSAB

1	LM6132M	(surface mount)
1	1uF tantalum capacitor	
1	100uF electrolytic capacitor	
1	1A bridge rectifier	
16	2.2KΩ resistor	
4	10KΩ resistor	
1	220kΩ 20-turn potentiometer	
8	6-6 telephone jack, PC-mount	

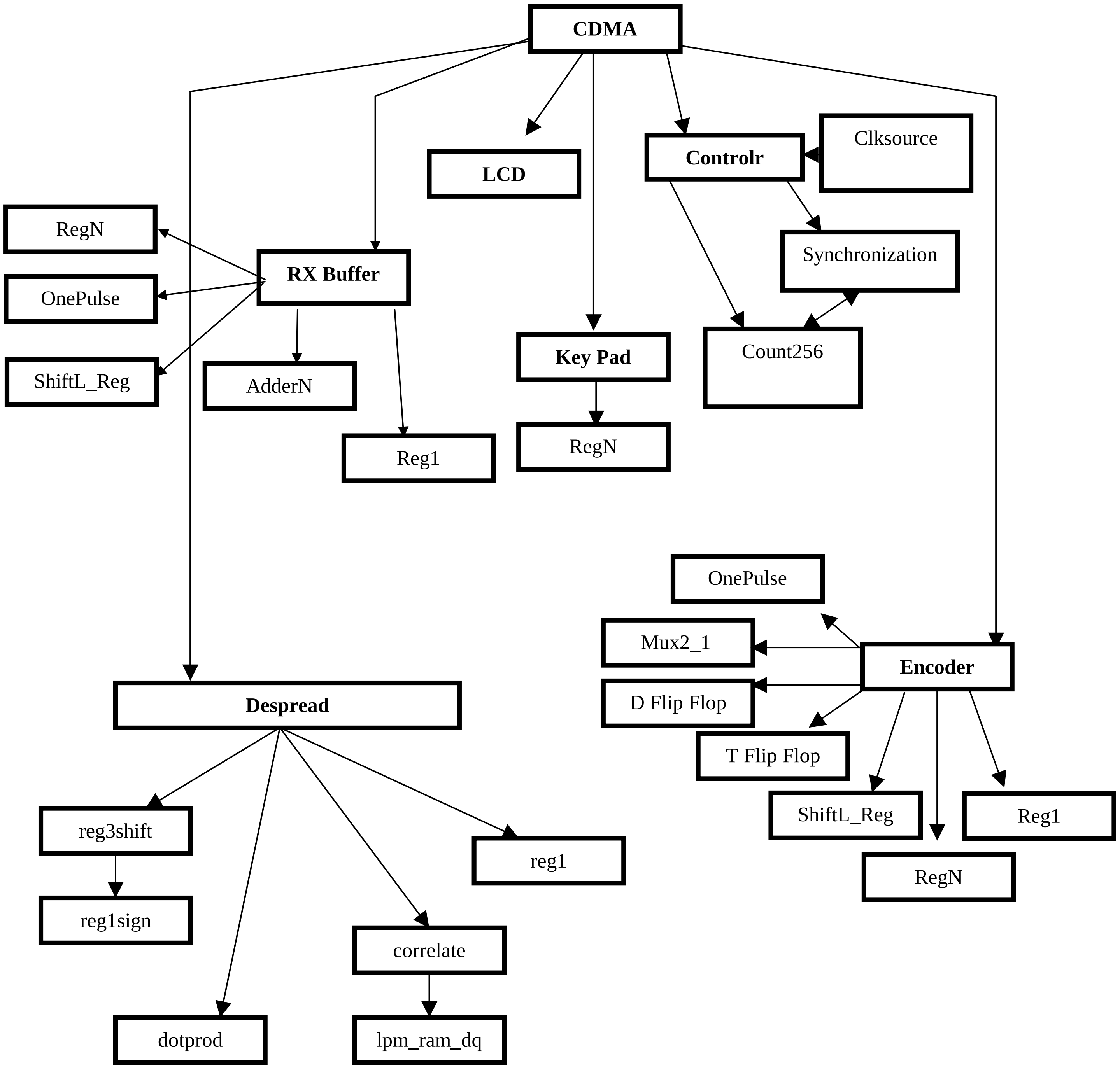
General Parts

3	Altera 10K20 UP1 boards
3	4x4 keypads
3	Multi-line LCD displays (16x2)
3	4 Switch Dip Package (8 pin)
3	72-pin dual-inline male headers (cut to 60 pin)

FPGA Usage

Component	**Subcomponent**	**Logic Cells**		**Memory bits**	
Controlr			109		0
Despread	Correlate (x2)	104 (208)		128 (256)	
	Dotprod (x2)	107 (214)		0	
	Reg3shift	128		0	
	Total for despread		515		256
Encoder			79		0
RxBuff (x2)			28		0
LCD			204		0
Keypad			29		0
Total for CDMA			**1037**		**256**

Diagram of Design Hierarchy



Experiments

Decoder

One experiment was run, using a C program and GNUplot. Its purpose was to test the quality of our chip sequences and to simulate the decoding process. The program simulates the decoding of a signal with several asynchronous transmissions. It takes as input the selected CDMA codes, and the transmission sequence for each station. As output, it generates a series of files: one data file for each station and one command file for each station. The command files instruct GNUplot to open the data file, and plot the data correctly.

For this experiment, the CDMA codes had width 8, and an over-sampling of 4 was used. Dot products are calculated using the station's chip sequence as one vector, and the received sample values as the other. The graph contains three series:

1. "User X Data" - indicates the correct value that should be received for this user. It stays at this value for the first four samples (i.e. throughout the first chip in the sequence) and drops to -50 elsewhere to indicate where the sequence should NOT be despread correctly at these times. The received value, (-1, 0, or +1), is multiplied by 32 for easy comparison with other series.
2. “User X Dot Prod" - is the dot product with taking one sample during each over-sampling period (this is the method actually used for this project). So the dot product vectors will be as wide as the chip sequences. This result is multiplied but the over-sampling rate (4) for easy comparison.
3. "User X DP Quad Sum" - this is the result of adding four adjacent dot products, each calculated as above for the second series. In effect, this averages the dot product values. This series is just for comparison.

The first experiment uses the follow chip sequences:

User 0:	00000000
User 1:	01010101
User 2:	00110011
User 3:	00001111

The analysis is very interesting. Users 0 and 2 had reasonable reception of their data. In other words, at the start of a chip sequence, the data series and the dot product series match up well. User 1 did not work well at all. The dot product changed from -32 to +32 on nearly every chip. This would be extremely difficult to despread accurately. User 3 shows the ideal case. The correlation (i.e. dot product) peaks at the beginning of each new chip sequence, and is much lower elsewhere. This would be relatively easy to decode.

In response to the dismal results of the first experiment, many sets of chip sequences were tried. Also, the requirements changed at this point in the project's development: only three stations, and hence only three chip sequences, are required. This eases the difficulty in finding suitable codes. Ultimately, the following codes were chosen:

Station 0:	00110101
Station 1:	00001111
Station 2:	00010110

The results of the experiment on these codes are shown following. It is readily apparent that these codes should allow for relatively accurate synchronization. Note that the synchronization is not perfect – with CDMA this can never happen. Instead, by using statistical techniques, the most likely bit is chosen for initial synchronization.

Hardware

Once the hardware boards were completed, they were physically tested to compare the Electronics Workbench simulation results. A test entity was constructed which encapsulated the encoder module with some simple supporting logic to display the value read by from the ADC onto the seven segment LED displays. The hardware was connected to the UP-1 board and we watched the waveforms on an oscilloscope. We found from experiment the degree of latency through the ADC as well as the maximum output speed of the hardware. The latency was found to be 3 clock cycles, almost an entire sample width for one bit. We realized the implications this fact had in trying to synchronize a station's transmission with what it received. Even so, due to chip space and time this was not completely explored. Secondly, we found the top transmission rate to be just under 50kbps and have been continuing to improve this speed even during writing of this report.

References

Previous EE 552 Projects

1. http://www.ee.ualberta.ca/~elliott/ee552/projects/1997f/radiotx/radiotx.html
2. http://www.ee.ualberta.ca/~elliott/ee552/projects/1999_w/RFTx_Rx/
3. http://www.ee.ualberta.ca/~elliott/ee552/projects/1999f/rf_telemetry/
4. http://www.ee.ualberta.ca/~elliott/ee552/projects/1999f/Drivers_Ed/drivers_ed_final.pdf
5. http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1997f/lcd/lcd.html
6. http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1998f/LCD_Driver/app.htm
7. http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1998f/LCD_driver/

Internet Resources

8. http://www.ee.ualberta.ca/~elliott/ee552/labs/lab5.html
9. http://www.ee.ualberta.ca/~ee480/lab/lab1/guide/behavioral/Mux2_1.vhd
10. http://www.abacom-tech.com/
11. http://www.hobbytron.net/
12. http://www.freeradio.org/tech/amp1wassembley.html
13. http://www.us-epanorama.net/radio.html
14. http://www.altera.com/documents/ds/dsf10k.pdf
15. http://www.ee.ualberta.ca/~elliott/ee552/AlteraDoc/flex10k_datasheet.pdf
16. http://www.ee.ualberta.ca/~elliott/ee552/AlteraDoc/up1_board_documentation.pdf

Text Resources

17. “Spread Spectrum Systems with Commercial Applications” Third Edition, by Robert Dixon.

Charlie’s CMPE 401 presentation

18. ftp://web:web@24.108.95.68/webpage/EE552/CDMA Final.ppt

Appendix A: CDMA

CDMA allows multiple users to simultaneously transmit data on one frequency and have these data successfully decoded at the receiver. This project uses Direct Sequence CDMA (DS-CDMA), in which each station is assigned a spreading code, known as a chip sequence. These codes are chosen so as to be orthogonal to each other (alternately, they exhibit very low – ideally zero – cross-correlation). When a station wishes to send a data bit, a permutation of the chip sequence is transmitted. Thus, if the chip sequence is *k* chips (bits) long, the unit must transmit *k* bits for each data bit. The theoretical maximum bandwidth for a single user is therefore reduced by a factor of *k*. However, with chip sequences of this length, up to *k* simultaneous users are supported. So the total system bandwidth retains its theoretical maximum value.

Wireless Physical Layer

Normally, DS-CDMA wireless is implemented with phase shift keying (PSK – either binary or quadrature). We attempted to locate parts that would take care of the RF aspect (modulation, transmission, reception and demodulation), but were unable to. The search for suitable parts continues. In this case, the FPGA would handle spreading of the data signal prior to modulation, and despreading of the received signal after demodulation.

For CDMA, the transmitter must be able to send three different values: +1, –1 and 0. Their meanings are summarized in the table below.

Value	**Action**	**Transmission (PSK)**
+1	Encode logic High	$A\cos(\omega_c t + \phi)$
0	Don’t transmit	No transmission
–1	Encode logic Low	$A\cos(\omega_c t + \phi + \pi) \equiv -A\cos(\omega_c t + \phi)$

In order to function correctly, CDMA logic High and Low must cancel each other during the reception process. Here we see the advantage of PSK: this linear additive property is a natural function of the modulation technique and does not require extra logic. On the other hand, to implement a CDMA system in FSK would be much more difficult because it does not have this additive property. Thus for the remainder of this section, CMDA values will be presented as {–1, +1,0} rather than a modulated cosine.

Example: "How CDMA is actualized on the Physical Layer"

Two units want to transmit, they already have their orthogonal Chip Sequences (CS) and we want to understand how the Physical Layer handles the input. A wants to transmit a logical low while B wants to transmit a logical high

Chip Sequences
```
A: 0101
B: 1100
C: 0010
```

So if station A wanted to transmit a logical high, it would send it's chip sequence. If it wanted to send a logical Low, it would send the chip sequence inverted. (i.e every 0 becomes a 1 and vice versa)

To transmit a 1 in the CS, the transmitter would send a set magnitude on the carrier frequency.

To transmit a 0 in the CS, the transmitter would send the same set magnitude on the carrier frequency shifted in phase by 180 degrees.

Transmission
```
A sends LOW  = 0101  (physically Tx) 1 -1  1 -1
B sends HIGH = 1100  (physically Tx) 1  1 -1 –1
C is silent  = 0010  (physically Tx) 0  0  0  0
(Signals Linearly Add)         SUM = 2  0  0 -2

So the Receiver sees only the addition of all the
contributions to the airwaves.
```

Station A(• operator is the dot product)
$(sum)\bullet(CS_A) = (2,0,0,-2)\bullet(1,-1,1,-1)/4 = -4/4 = \mathbf{-1}$
Station B
$(sum)\bullet(CS_B) = (2,0,0,-2)\bullet(1,1,-1,-1)/4 = 4/4 = \mathbf{1}$
Station C
$(sum)\bullet(CS_C) = (2,0,0,-2)\bullet(-1,-1,1,-1)/4 = 0/4 = \mathbf{0}$

Wired Physical Layer

A CDMA system does not need to be wireless. Due to the difficulty in finding suitable wireless components, we will use wires to emulate the wireless environment. We will create this wired system in a manner that will facilitate a transition to wireless, should appropriate parts become available.

In order to emulate the wireless system, we must examine its properties:

1) Pulses of the same phase linearly add
2) Pulses of opposite phase cancel
3) A unit may not transmit at any given time (i.e. no pulses)

To accommodate these properties, a simple voltage adder circuit will be used. For transmission, each unit will output two bits. Three of the four possible values are used, corresponding to {–1, +1,0}. These are assigned voltages of -5V, +5V and 0V respectively. The voltage adder circuit can be easily built from an op-amp, and will take the transmission lines from all units as input. This sum will then be wired back to each unit, where an ADC will convert the voltage to a usable digital input.

This approach is very similar the wireless approach. Signals add and subtract as appropriate. No attempt is made to synchronize the signals, so the adder output will change asynchronously with regard to any of the particular units' clocks. The main difference is that the wired approach only has one input component (the voltage) per channel, whereas the wireless has two, since the PSK signal is decoded into in-phase and quadrature components.

Orthogonality and Decoding

Orthogonality is critical to the above CDMA system. If each chip sequence is not perfectly orthogonal to all others, it is impossible to accurately decode the data streams under all circumstances. However, generating orthogonal codes is relatively straightforward. A very commonly used set is the Walsh functions, which has 64 such codes.

In the example above, the ideal spreading/despreading process was introduced. This method is used in the current IS-95 cellular telephone standard, on the forward link (base station $\rightarrow$ mobiles). Since the single base station generates all transmissions on the forward link, it can ensure that the chip sequences are correctly synchronized. However, this cannot be said for the reverse link (mobiles $\rightarrow$ base station). In this direction, each mobile transmits completely asynchronously. Thus the base station must be able to simultaneously track, despread and demodulate each transmission. Unfortunately, although pre-generated codes like the Walsh functions have zero correlation (i.e. are orthogonal) when synchronized, as soon as the codes begin to shift (by as little as one bit), their correlation increases dramatically, to the point where accurate despreading is no longer assured.

Commercial CDMA systems solve this problem by using continuously generated chip sequences. Suitable sequences are called PN (pseudo-random noise) sequences, and are easily generated using a feedback shift register. This method also provides better performance in noise, a virtually unlimited number of simultaneous users (although per-user performance degrades as the number of users increases) and improved security. However, none of these issues are important factors for our project, and since PN sequences are difficult to implement, they will not be used. This difficulty comes from the greatly increased complexity of the despreading subsystem due to the new requirement to search out the current location in the PN sequence.

Another approach is to use a set of pre-generated chip sequences that are orthogonal regardless of shifting. In order to achieve this, for *n* users, the codes must be greater than *n* bits long. For 4 users, the chip sequences must be of length 8. Note that half of the theoretical bandwidth is now wasted. However, this loss of bandwidth is a common solution – for example, the IS-95 cellular system uses 128 chip PN sequences, even though only 64 simultaneous users are supported. Four codes that share this special orthogonality are shown below:

```
        00000000
        01010101
        00110011
        00001111
```

Unfortunately, this system has its own problems. The first three codes are virtually impossible to synchronize, because a shifted version of them appears identical. The fourth code may or may not be acceptable, depending on the initial synchronization method. If the decoder expects the other station to send only logic high values, then this chip sequence is acceptable since it is distinct. However, if the other station sends a combination of logic highs and lows, this sequence is not acceptable, since it will produce a good match as either "00001111" or "11110000," resulting in correct synchronization only about 50% of the time.

The ultimate solution is to find a series of codes that works well together. These must be orthogonal; however, they need not necessarily be orthogonal regardless of shifting. The key feature of these codes is that when rotated, they always appear distinct from the original code. Please refer to the *Experiments* section for a description of the method used for testing potential codes.

The final issue is decoding. Even in the wired system, the asynchronous nature of the incoming signal causes great difficulty. A possible solution is to sample at four times the chip period. This ensures that every transmitted pulse will be examined. A tracking subsystem then starts, checking every sample period for a match with its chip sequence. This subsystem must operate at $4\times8=32$ times the data rate (taking into account the 4 times oversampling and the 8 chips/data bit). Also, a separate tracking subsystem is required for every chip sequence that must be decoded – therefore four would be required to simultaneously decode all four stations.

To track, the subsystem will perform the dot product operation on 32 sequential bits of incoming data. Every value will be stored. Values that are close to the chip length (i.e. –8 and +8) demonstrate a higher correlation, and thus indicate where the first bit of the periodic chip sequence begins. Values in between these multiples indicate positions that are not likely to be the beginning of a chip sequence. Tracking will be a continual process (i.e. will constantly check for high correlation). However, once the “phase” of the incoming signal has been determined with respect to the chip sequence, decoding of subsequent data bits only requires the dot product operation.

Appendix B: VHDL Code

Index to VHDL code

Appendix C: Simulations and Test Cases

Index to Simulations and Test Cases

CDMA

This simulation tests the global function of the project. The clock is set to the UP-1 clock rate and all inputs and Chip Sequence settings are input as well. The length of simulation required to test decoder functions in the final project is prohibitively long. Thus, encoder output and LCD signals are the only practical signals to verify.

Encoder

All considerations for the encoder are on the following single waveform.

The test cases are preceded with a section to prove that nothing happens until the reset is asserted and the Load Chip Sequence is asserted. The chip sequence used is 10101100. Then the entire encoder will only function if ShiftL_CS is asserted and held high.

Then the first test case is to demonstrate that the first byte will always be outputting nothing because the registers are cleared and thus have invalid data. During this case, Register B is loaded with “invalid data” specifically.

The second test case demonstrates that Register B (that was loaded in first test case) is indeed outputting nothing because it is invalid data. During this case, Register A will be loaded with valid data.

The third test case starts to output register ‘A’ that has valid data. Visual inspection will prove that the chip sequence is being outputted. During this the Register B will be loaded with valid data for next case

The fourth test case proves continuous transmission and transmits register B.

The fifth case proves again that there is continuous transmission.

The sixth case demonstrates the behavior of the encoder after a reset occurs. Right after the reset, register A is loaded and the whole encoder waits until the chip sequence is loaded. Once the Chip Sequence is loaded, and shiftL_CS is held high.

Decoder

Reg3shift

This is a relatively simple shift register, except that it shifts data in chunks of 4 bits. The testing procedure simply makes sure that this register shifts as directed, and that it can handle a relatively high clock speed (in this case 25MHz, which is the fastest speed we can run the FPGA at).

Dotprod

Since this module is purely combinational, it is simple to test. Two dot product calculations are performed:

Calculation 1:

Index	Received Digit (binary)	Value of Received Digit	Chip Sequence Value	Dot Product Operation	Cumulative Sum
0	0011	+3	1	+3	3
1	0000	0	1	0	3
2	1111	-1	0	+1	4
3	1110	-2	1	-2	2
4	1101	-3	0	+3	5
5	0011	+3	1	+3	8
6	0010	+2	1	+2	10
7	0001	+1	1	+1	11

Calculation 2:

Index	Received Digit (binary)	Value of Received Digit	Chip Sequence Value	Dot Product Operation	Cumulative Sum
0	0110	+6	0	-6	-6
1	0011	+3	1	+3	-3
2	1001	-7	1	-7	-10
3	1010	-6	1	-6	-16
4	0001	+1	1	+1	-15
5	1111	-1	0	+1	-14
6	1100	-4	0	+4	-10
7	0101	+5	0	-5	-15

For the first calculation, the output should be 11 (in decimal), which it is (the *dotprod* output is in decimal). The output for the second calculation is negative. Using two's complement, -15 = 1110001 in binary (7-bit). Convert this value to decimal using unsigned arithmetic, and the answer is 113, which agrees with the simulation.

Correlate

Since this module is simply a large state machine, all states are tested. The following four tests are sufficient. Each is a part of one long simulation, with different time intervals chosen to highlight different aspects of the required functionality.

The first test verifies that the initialize procedure works correctly. The state alternates between state 1 and 2, as the counter (*corrCount*) increments to 31. The RAM address increments accordingly, and at each address a value of zero is written to memory. After the initialization, the FSM correctly moves into the first correlation state.

The second test shows several aspects of the FSM. After the *nextRecClock* signal goes high, the FSM waits three clock cycles before reading the dot product value. The current RAM address increments with every *nextRecClock*, so that the correct tally is incremented. Finally, this checks that the tally is only incremented if the dot product value meets the criteria (greater than +6 or less than –6). When *dpInternal* is 3 or 7C (equal to –4), the FSM does not move to state 7, where the incremented tally is stored. However, for 8 or 79 (equal to –7), the tally count is incremented, as required.

The third test demonstrates the initial synchronization process. To test, *dpIntenral* is left at 8, which is an acceptable value for incrementing the tally count. Eventually (around 110us), initial correlation is complete, and the FSM enters state 8 (the first decoding state). This initial synchronization will be tested more thoroughly from the despread simulations, when we can set the chip input directly (rather than the case here of setting the dot product value, which is rather tedious).

The fourth test shows the decoding process. Here, initial synchronization is complete. The *corrCount* variable counts to 31, incrementing after every *nextRecClock*, so that the correct number of samples is ignored before a new data bit is generated. The *dpCount* variable only changes when the dot product value is needed. Finally, the valid data signal goes high for only one clock cycle, as required.

Despread

This module is built structurally from components that have already been verified. As such, the purpose of this simulation is to ensure that these components interact correctly. As above, one long simulation was run, and important segments have been highlighted, as described below.

The two chip sequences are 00001111 and 00110101. However, so both stations do not synchronize simultaneously, the second sequence is phase shifted by 180º, to form 01010011. Both stations continuously send logic high values, for simplicity. Thus the receive data is as follows:

Bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Chip Sequence 1	0	0	0	0	1	1	1	1
Chip Sequence 2	0	1	0	1	0	0	1	1
Encoded Sequence 1	-1	-1	-1	-1	+1	+1	+1	+1
Encoded Sequence 2	-1	+1	-1	+1	-1	-1	+1	+1
Sum	-2	0	-2	0	0	0	+2	+2
Binary (4-bit signed)	1110	0000	1110	0000	0000	0000	0010	0010

The first test overviews the entire simulation. It takes approximately 160us for the first station to synchronize, and is followed quickly by the second station. At 180us, one station is told that its synchronization is bad. For the other station, this occurs at 225us. In both cases, the initial synchronization process is begun again.

The second test shows the interval from 160us to 220us. The receive (sample) clock is visible here. As expected, the two stations synchronize at different times – this is shown by the *dataoutValid* signal asserting for the different stations at different times. Once the first station's *invalidSync* signal is asserted at 182us, it no longer produces any valid data.

The third test shows the first station re-synchronizing. Afterwards, only its *dataoutValid* is asserted, since the other station has not synchronized yet. This test mainly shows that the decoder is capable of re-synchronizing to a station, a feature that had not been tested previously.

Keypad

Keypad keys:

```
 0  1  2  3
 4  5  6  7
 8  9 10 11
12 13 14 15
```

The keypad module is inactive until one of the bits in the row_input is set high. This corresponds to a user pushing a button on the keypad and making a connection. Once some activity is detected, the debounce state is entered. Basically, this latches the first transition. If after a set time the input is still high, each column is polled in turn (i.e. the bits of col_check are set high one at a time). If setting a given column produces a non-zero bit in row_input, the key_pressed is determined from the row and column numbers. valid is set and held for one clock cycle as soon as a key has been pressed; the keypad holds the value until another button (or reset) is pressed.

The timing simulation begins with a reset to ensure that all values are initialized. At 90 ns, row_input is set high on the fourth row, indicating that a button has been pressed. The row_input is set to all zeroes, except during the cycle when the second column is being polled. This means that the button in the second column and fourth row has been pressed, i.e. 13. The state machine then returns to the waitforkey state. The next key is detected in the fourth row and third column, i.e. button 11. Finally, at time 800 ns, the reset is tested to ensure that all values are returned to their initial states.

LCD

The LCD output screen consists of two rows of eight characters each. At this point, only the first line is being used. There are two modes for the output: a) bytes & errors and b) unit & data. For example:

Mode a)

B	:	1	2	3	4		E	:	0	0	1	2			

Mode b)

D	e	v	i	c	e	:	1		D	a	t	a	:	3	

Note: for the purposes of simulation, the clock rate was set higher than it is in practice, and the states wait for a smaller number of clock pulses. Even so, the initialization sequence is rather long (from 0 – 2.1 us). During initialization, the LCD is given time to warm up, the modes are set, the display is turned on and off, and the cursor is moved to the home position. During initialization and the write sequence, done is held low to indicate that the LCD is not ready for new input.

At time 2.13 us, done is high (lcd is ready for input) and data_valid is high, so the LCD enters a write sequence. Mode is low, so the values written are as follows:

44	65	76	69	63	65	3A	31	A0	44	61	74	61	3A	30	A0
D	e	v	i	c	e	:	1		D	a	t	a	:	0	

At time 3.3 us, the LCD enters another write sequence, but this time mode is high.

42	3A	31	32	33	34	A0	45	3A	34	33	32	31	A0	A0	A0
B	:	1	2	3	4		E	:	4	3	2	1			

At time 4.36 us, the LCD enters another write sequence with mode is high, this time with different data.

42	3A	31	31	31	31	A0	45	3A	34	34	34	34	A0	A0	A0
B	:	1	1	1	4		E	:	4	4	4	4			

Finally, reset is tested at time 5.4 us.

Controller

Sourceclk

This module is simply a 23-bit counter. There is a need for different clock speeds within the chip to do different tasks properly. All 23 bits are brought out of this module and each bit is made accessible as a clock. Each consecutive bit will have a clock period exactly twice the previous bit. Therefore, there is a large range of usable clock speeds to choose from. The simulation test, merely tests that the counter does indeed count.   Because the value of the entire counter is inconsequential, there is no reset or set function.

Output Mode Simulations

Initialization
This occurs every time that the module experiences power-up, reset or a change in the clock speed mode. The controller sends a series of twenty bytes of all ones to the encoder for transmission, followed by a null byte. This simulation ensures that all control lines are asserted properly to ensure data is transmitted during initialization.

All of the following Output modes initiate immediately after initialization has occurred.

Stream Mode: I/O-M = 1, LatchM = N/A

In this mode, data is send continuously. Each byte of data sent is an increment of the previous. This is done with an 8-bit counter that is fed into a register that is then fed into the encoder and sent in CDMA code. The simulation tests that the counter does indeed get incremented and that the encoder input register gets written to feed the encoder. Once the register is written, the simulation test that the encoder is told that data is ready and that the controller responds to its receipt of data.

Keypad Mode – Non Latching: I/O-M = 0, LatchM = 0

In this mode, data in the form of a key value is sent to the encoder for transmission when the key has been pressed. The value is sent into a register, same as above. Then sent for transmission. The simulation tests that the controller does not allow new transmission until a button has been pressed. It is then tested that the value is only sent once.

Keypad Mode – Latched: I/O-M = 0, LatchM = 1

Same as above, however, even if a key has not been pressed again, the encoder is sent the same byte value repeatedly and this value is sent continuously. The simulation is identical to the Non-Latching mode. However, in this simulation it is checked that the value is sent continuously.

Input Simulations

Two instances were tested with the inputs. Since two different stations can transmit at once, two inputs are available.

The first simulation dealt with data arriving from one station. The simulation tested that the controller would acknowledge the data presence, select it and send it to the LCD. As well, the controller should only send this information to the LCD if the I/O mode is set for stream.

The second simulation dealt with the possibility of data arriving from two stations. The simulation tests that the same occurs as the previous test but then continues to test if the controller then selects the other data and displays it.

CounterN

The counter is designed to count to 4096 (2^12). The counter increments on the rising edge of its input clock. The count can be reset at any time by using the reset line. The simulation shows both operations.

Appendix D: Test Benches

Index to Test Benches

Cdma_Test:

This test bench instantiates two CDMA units and sets one to listen to itself and the other. The two units are connected using a VHDL transmission simulator, which adds the output signals from the two units and feeds them into unit 2.

Appendix E: Data Sheets

Index to Data Sheets