Wednesday, June 28, 2000

18-551

Spring 2000

CDMA Modem with LPC Voice Compression

Felipe E. Conill

Erik Dykema

Julian Gomez

Table of Contents

General Introduction

The structure of this final report will be a focus on the three fundamental blocks involved in the construction of our CDMA modem. We will start by describing the particulars of LPC voice coding. After this discussion we will describe the CDMA standard and at the end we will deal with the issues of using an external DAC/ADC connection.

We hope that by reading this report one can see the issues involved in building such a system on the TMS320Cx family of DSP's.

Linear Predicting Code Compression and Decompression (LPC)

Background:

Linear Predictive Coding is one of the most powerful techniques available for encoding and transmission of speech signals. LPC analysis can reduce a human voice signal to an extremely low bit rate digital signal, suitable for transmission over low bandwidth digital and packet channels. Linear Prediction was first introduced as a method for coding speech by Atal and Schroeder in 1967, and in 1982 it became a federal government standard (FS-1015)for speech compression. LPC-10 was attractive to the government because it's low bit rate made transmission of the signal resistant to jamming and channel noise.

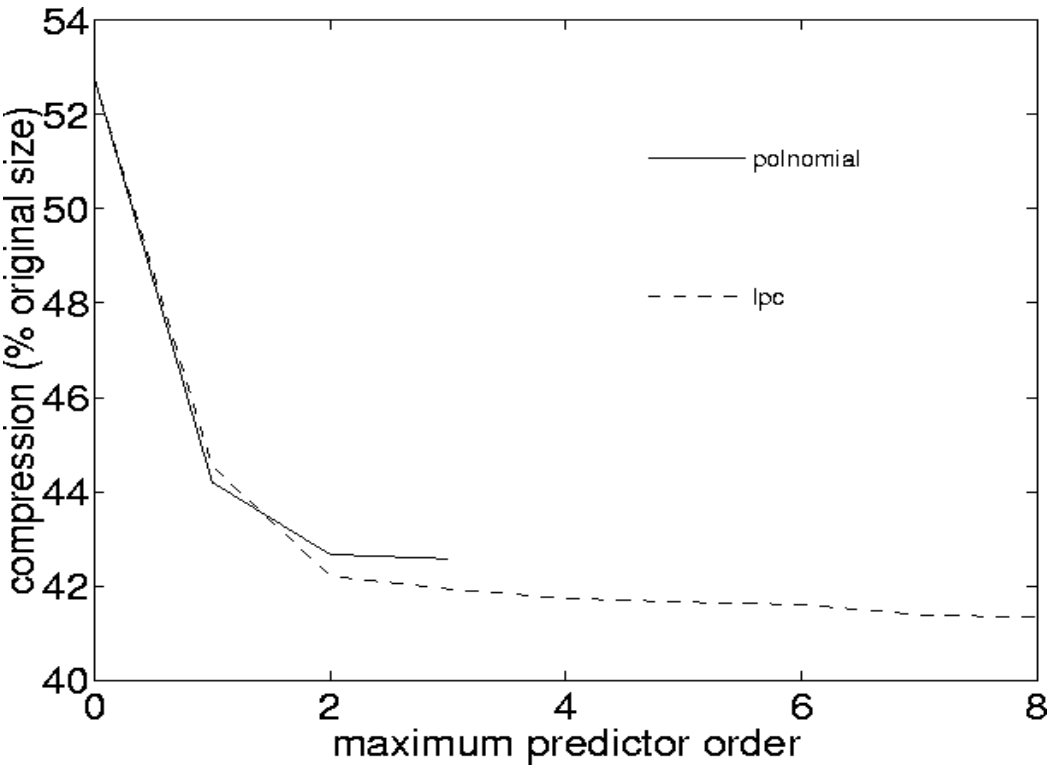
Signal Information and Assumptions:

LPC starts with the assumption that the speech signal is produced by a buzzer at the end of a tube. The vocal cords produce the buzz, which is characterized by its intensity and frequency. The vocal tract forms the tube, which is characterized by its resonances, which are called formants.[1]

LPC analyzes the speech signal by estimating the formants, removing their effects from the speech signal, and estimating the intensity and frequency of the remaining buzz. The process of removing the formants is called inverse filtering, and the remaining signal is called the residue. The filters which perform

[1]http://asylum.sf.ca.us/pub/u/howitt/lpc.tutorial.html

the analysis are described by differential equations. Equations of a higher order are more computationally intensive, produce more output, and more accurately predict the signal. LPC-10 uses a 10^{th} order filter to describe it the signal.



The figure above shows the differences in compression between different order filters. It is obvious that the move from a 1^{st} to a 2^{nd} order filter produces a big jump in compression, while the move from a 2^{nd} to a 6^{th} order filter is not as dramatic. A 10^{th} order filter produces the most efficient compression and prediction, a higher order filter would not appreciably improve the compression.

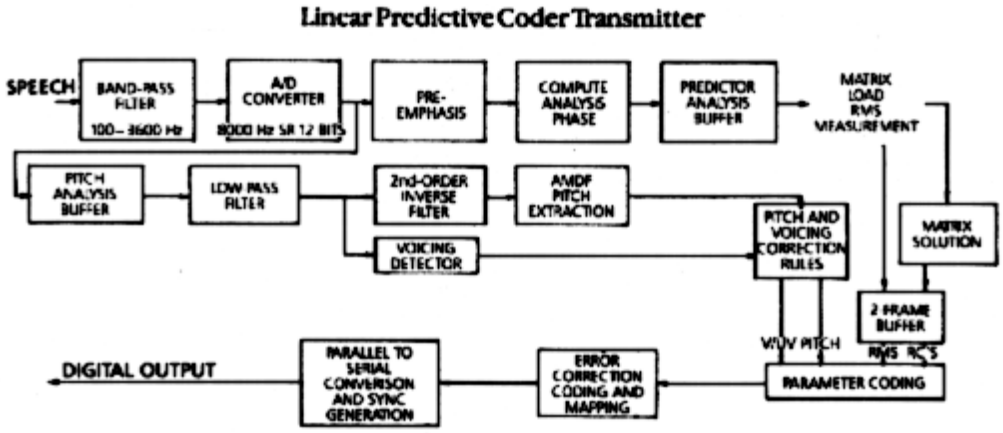
The numbers which describe the formants and the residue can be stored or transmitted somewhere else. LPC synthesizes the speech signal by reversing the process: use the residue to create a source signal, use the formants to create a filter and run the source through the filter, resulting in speech. Because speech

signals vary with time, this process is done on short chunks of the speech signal, which are called frames.

The FS-1015 standard implementation vocoder[2] which we used in our project takes as input a linear 8 kHz voice signal, which is broken up into frames of 180 samples. This frame is reduced to 54 bits which describe the speech signal. 44 frames are processed per second, yielding a bit rate of 2400 bits per second.

Analysis:

The computationally intensive part of the LPC-10 algorithm is the determination of the formants from the speech signal. The solution is to use a difference equation which attempts to express each sample as a linear combination of previous samples. This equation is known as a linear predictor, which is why this technique is known as Linear Predictive Coding.[3]



[2] Source available for download at http://www.arl.wustl.edu/~jaf/lpc
[3] Http://asylum.sf.ca.us/pub/whowitt/lpc.tutorial.html

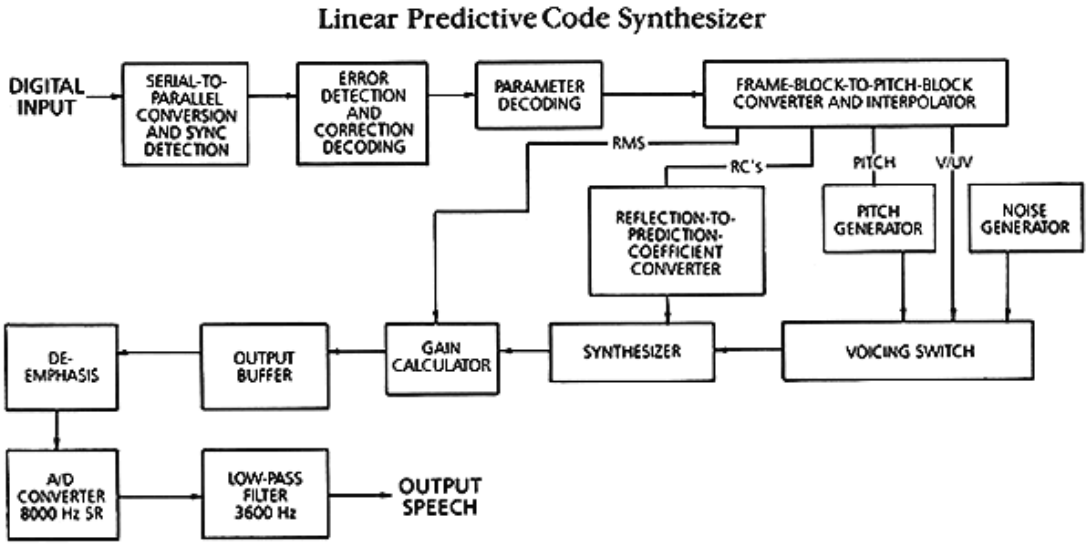
These formats, once found, describe the mechanisms responsible for producing the speech. However, it is necessary to have more information about the signal to accurately reconstruct it, including the pitch period, gain, and voicing.

Voicing is a term used to describe the origin of the speech signal. Voice is produced by the throat and mouth filtering a source sound. This source is either produced by the vibration of the vocal cords (voiced), or is a random hiss produced by air coming out of the lungs. For voiced segments of speech, the source is assumed to be a train of impulses. The pitch period describes the spacing of the impulses. For unvoiced segments, a random noise generator is used as the source. For either type of voicing, the gain describes the amplification of different segments.

Synthesis:

The synthesis step is basically the reverse of the analysis step. The 54 bits of input for one frame (22.5 ms) are read in and used to determine the values for the filters, voicing, pitch period, gain, etc.

One thing that is different between the analysis and synthesis is that the synthesizer attempts to do some smoothing. It keeps one frame from the past in its buffers and attempts to smooth the output signal by comparing the two. This ultimately causes the output signal to be delayed by one frame.



Advantages and Drawbacks:

LPC-10 has a few major advantages over other methods of compression. Its extremely low bit rate makes it ideal for low-bandwidth applications. At 2400 BPS, more than 20 simultaneous conversations could take place on the bandwidth of one 56k modem! The government suggests that this low bit rate makes it resistant to jamming and channel noise, which increases it's worth for military applications.

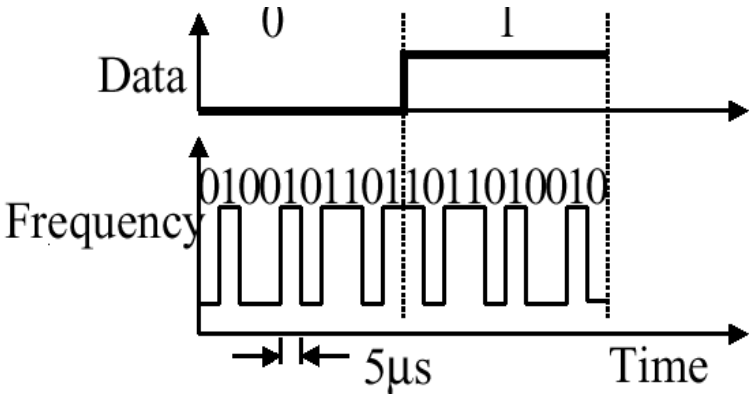
LPC-10 also has some drawbacks, however. It is a vocoder, not a waveform encoder, and therefore is inappropriate for the transmission of music, other sounds, or anything but voice. The algorithm is very computationally intensive, and requires a somewhat powerful embedded processor or DSP to perform its tasks, which makes it too expensive for low-cost applications. The quality is also not the best, introducing a lot of static and also a tinny sound to voice. With regard to channel noise, its low bit rate could be a boon and a curse

at once, as the code is not very redundant. A few bit errors and the frame is lost. This is, of course, mitigated with packet level error detection and correction.

Code Division Multiple Access (CDMA)

Introduction:

CDMA is a way to have more users on a channel by using a spectrum larger that what is needed to transmit the signal. This is done using a set of orthogonal spread spectrum codes to encode each user's transmission. In our project we implemented a Direct Sequence Spread spectrum system. A DS-CDMA application is one were you spread the original sequence with a spreading code in order to make it occupy more bandwidth, taking up a wider spectrum and hence being more resistant to noise.



The main points of the procedure are the following:

- A random binary string is used to modulate the transmitted signal. This random string is called the spreading code.

- The data bits are mapped into a pattern of "chips" and mapped back into a bit at the destination. The number of chips that represent a bit is the spreading ratio.
- The higher the spreading ratio, the more the signal is resistant to interference. The lower the spreading ratio, the more bandwidth is available to the user.

We used a periodic code that was to be used for spreading the signal. After looking into spreading codes we decided to use a 16 bit Gold code. A Gold code is a type of spreading code with a high correlation value when it is aligned and low cross correlation values with other Gold codes. This type of code is generated with a Feedback Shift Register.

The stream that was coming in from the LPC was a 2400 bits/s digital stream. After spreading it with our Gold code it would become a 38.4 Kbps stream. This was implemented in our program storing the incoming bits (a spread sample) in shorts. A short is 16 bits so therefore one second of LPC data would be 150 shorts. After spreading the bits each bit would be stored in a short (16 bits) which is that bit up-sampled by 16 and then spread. After this was done we would proceed to the modulation.

Binary Phase Shift Keying:

For Modulation we used BPSK. Basically, BPSK works in the following way: if the bit is a one it will transmit a sine wave but only from 0 to a 180-degree

phase. If the bit is a 0 it will transmit at sine wave from 180 to a 360-degree phase shift. At the demodulator we will just use a matched filter to decide whether the bit is a one or a zero. In a channel with no noise present one could just add the samples incoming to derive whether a bit is a 1 or a zero.

Since the components in our system are not ideal for DC transmission the sine wave started at the middle point from the range we had to transmit to the AD/DA. Since our DAC used only 10 bits we formatted the sine to in a 10 bit format. The array that encompassed our sine wave is the following:

```
unsigned short bpsk[8] = {0x1800, 0x1C00, 0x1FFC, 0x1C00, 0x1800,
                          0x1400, 0x0000, 0x1400};
```

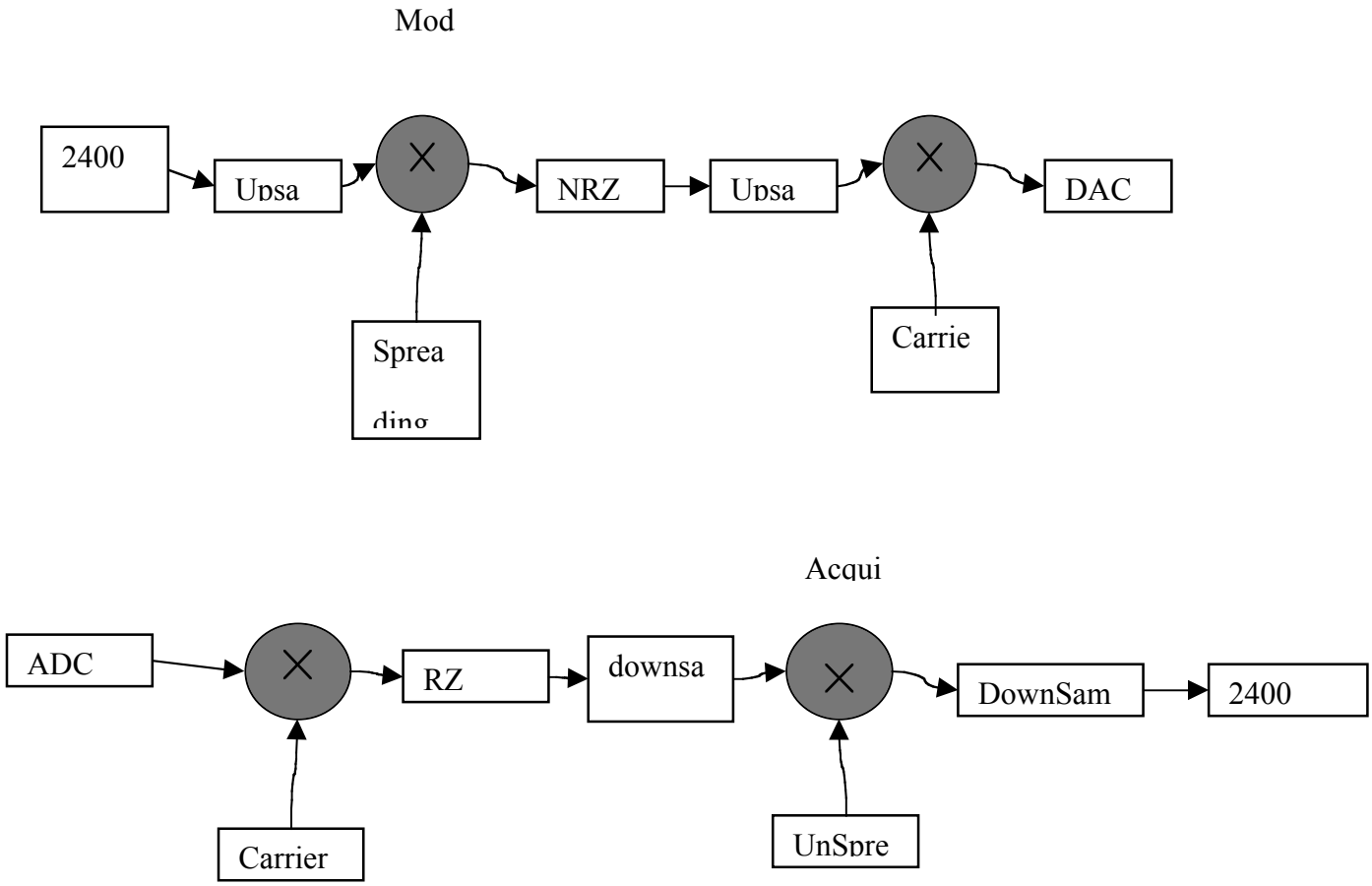
This array samples each bit with 4 samples of a sine and then passes it on to the DAC already with the remaining bits configured for the specific input out DAC takes. Therefore at the end we would have a 153.6 Kbps stream that could be easily handled by our DAC/ADC channel since the fastest they could go is 1 MSps and we only need 0.1536 MSPs.

For our project we had to modify some specific portions of our code to make sure that the real-time for the CDMA part was accomplished. Basically after modifying the way we modulated the data we were able to achieve 4 to 5 parallel operations in the kernel of our loop for the modulation.

Synchronizing:

One aspect of our project, which was done for simplification, is that since everything was implemented in the same EVM we decided to use the same clock for the receiver and the transmitter. This is not a real life situation since normally the case is the contrary. For example, in a cellular network the receiver (i.e. a cell phone) has a clock that is not synchronized with the sender. Therefore one needs to implement tracking algorithm to make sure the alignment of the code and the CDMA code is done. This is performed with different algorithms that were not needed in our program since they were running with the same clock and were therefore always aligned.

The data flow for our whole channel is the following:



Communication Channel - Daughterboard

Introduction:

As a means to simulate a channel, we decided to develop a CODEC that would transmit the CDMA-Encoded data outside of the EVM. We needed a Digital-to-Analog-to-Digital system which would run at a fast enough rate to handle a real-time communication. We opted to make this system outside the EVM because its CODEC would not go at fast enough sample rates, 48KSPS, while we needed close to 700KSPS. This daughterboard-system would contain a Digital-to-Analog converter, an Analog-to-Digital converter, and an 80-pin daughterboard connector to fit the EVM's interface. It would communicate with the DSP via the Multi-channeled Buffered Serial Port (McBSP), using a clock signal derived from the processor by the Timers. We searched for the appropriate DAC and ADC extensively through the Texas Instruments site, as well as other commercial daughterboards made for the C6x EVM. We came across products like a daughterboard which made an AD-DA conversion, the opposite of what we needed. We also found one which would sample at 1MSPS, convert from DA-AD, but were forced to disregard it because of a $1200.00 price tag. At this point we decided to order some samples of the following two converters and connector, the first from TI and the other from Samtec to build our own daughterboard.

Materials Used:

1. Texas Instruments TLV 5604 Quad DAC: This is a quadruple 10-bit voltage output converter with a 4-wire serial interface for a glueless interface to the TMS320 serial port. It is programmed with a 16-bit serial word comprised of a DAC address, individual DAC control bits, and a 10-bit data value. It has a dual operation mode: high-power or low-power. At high power (Vcc=5V) it can operate at a maximum clock rate of 20 MHz, or 1.25 MSPS, which is the mode we used for our implementation. Upon the arrival of our TLV 5604 sample IC's, we were amazed by their minute size, but the challenge of building it into the daughterboard was welcome.
2. Texas Instruments TLV 1572 ADC: This is a 10-bit successive-approximation converter which accepts an analog input range from 0 to Vcc and digitizes it at a maximum 1.25 MSPS (at high power, 20MHz clock). It is made to communicate with digital microprocessors via a serial port that interfaces directly to the TMS320 DSP family without additional glue logic. Also upon the arrival of the TLV 1572 samples we were amazed by its even smaller size, half of that of the TLV 5604 since this converter comes in an 8 pin package, as opposed to the DAC's 16 pins.
3. Samtec TFM-140-32-S-D-LC 80-pin Connector: This is the mating connector to the C67 EVM Board. It's attached to the daughterboard and matches the Expansion Peripheral Interface Connector (EPIC) J7.
4. RadioShack PC Board Kit: This kit contains the etching solution, the permanent ink solvent, a permanent marker, as well as a few of the copper-

plated boards. We used it to draw the layout for connecting the pins from the DAC, ADC and Connector.

EVM Resources Used:

1. Multi-channeled Buffered Serial Port: We had two available McBSP's, 0 and 1. 0 was used to communicate with the board's CODEC, while port 1 was available for our use. The McBSP provides a serial connection to external peripherals via the J7 EPIC. The Sample Rate Generator (SRG) of the McBSP configures the Data transmission and reception. This is the meat of the communication process, for it generates the frame sync signals as described by the control registers, and obtains a clock signal for this purpose from either an external or an internal source. In our case, we chose an external source, which was the one coming from the Timers of the EVM through a TOUT pin and feeding both the converters and the SRG. The McBSP has a total of 8 32-bit control registers, which are configured to provide the necessary signals we need coming out of the EPIC pins:
   - Serial Port Control Register (SPCR): This is one of the two registers which control the serial port. It contains fields such as Interrupt Modes, Digital Loopback Mode, as well as other set and reset bits.
   - Pin Control Register (PCR): This, the second of the serial port control registers, contains fields which set the Transmitter mode, Receive mode, Frame sync modes, Clock modes, Clock polarity, Frame sync polarity, as well as some status pins.

- Receive and Transmit Control Registers (RCR & XCR): The fields here set the RCV and XMIT frame phases, frame length, element (sample) length, data delays, as well as other features which were less important in our particular implementation.
- Sample Rate Generator Register (SRGR): These fields configure the SRG functions, such as clock source, clock synchronization, clock edge polarity, frame period, and frame width.
- Multichannel Control Register (MCR): The McBSP allows for multiple channels to be independently selected for transmission and reception. Since we just needed one channel, a single frame, with a single word, and just 16 bits of data, all the options in this register where set to their defaults.
- RECV/XMIT Channel Enable Registers (R/XCER): These registers enable any of the 32 elements for receive and transmit. We didn't need to bother with these, so we just used their default values.

2. Timers: The functionality of the timers ranges from event counters to CPU interrupts, DMA synchronization and pulse generation. For our project, we needed to obtain a clock signal to run the converters as well as the SRG of the McBSP. We hence used the pulse generating function of the EVM Timers. There are three registers involved in this process, the Period Register, Counter Register, and the Control Register. We did not need to count events, so the Counter Register was of no importance to us. The

Period Register was used to calculate the speed of the output clock signal, and the Control Register was set up to generate pulses.

Procedure:

1. Hardware: The most interesting aspect of the daughterboard part of the project was the hardware building process. Because of the size of the IC's, the soldering was going to be a great challenge. We employed a method in which we would use a thin-point permanent pen to draw what would be extensions to the chips' pins. Then, these extensions were soldered according to the following table, which we developed from the EPIC and converters' pinouts[4]:

CONNECTIONS:			
DAC Pin #	NAME	Description	Destination
1	DVDD	Digital Supply (5 V)	5,6,9, or 10 Vcc
2	~PD	Power-down Pin (Never active)	5,6,9, or 10 Vcc
3	~LDAC	Load DAC (keep low)	3,4,7 or 8 Grnd
4	DIN	Serial data input (digital in)	36 XDX1
5	SCLK	Serial clock input (20 MHz)	45 TOUT
6	~CS	Chip select	59 XRESET'
7	FS	Frame sync input.	35 XFSX1
8	DGND	Digital ground (just ground)	3,4,7 or 8 GRN
9	AGND	Analog ground (just ground)	3,4,7 or 8 GRN
10	REFINCD	Volt. Ref input for DACS C & D	----------
11	OUTD	X-ignore	----------
12	OUTC	X-ignore	----------
13	OUTB	X-ignore	----------
14	OUTA	DAC A Output (Analog)	ADC # 4(AIN)
15	REFINAB	Volt. Ref input for DACS A & B	5,6,9, or 10 Vcc

[4] The pinouts can be found in page A-7 of the TMS320C6201/6701 EVM Technical Reference, TI Literature Number SPRU305

16	AVDD	Analog Supply	5,6,9, or 10 Vcc
ADC Pin #	NAME	Description	Destination
1	~CS	Chip select	59 XRESET'
2	VREF	Reference voltage input	5,6,9 or 10 Vcc
3	GND	Ground	3,4,7 or 8 GRN
4	AIN	Analog Input	D A C # 14(OUTA)
5	SCLK	Serial clock input (20 MHz)	45 TOUT
6	VCC	Digital Supply	5,6,9 or 10 Vcc
7	FS	Frame sync input.	41 XFSR1
8	DO	Digital Output	42 XDR1

Once we drew the 'extensions,' we dipped the copper-covered board in the solution which would remove any copper which wasn't protected by the permanent ink. This left think copper lines, which were exposed by using the permanent ink solvent. These lines were then connected to their respective pins from the converter to the EPIC, according to the above table, by using soldered wires. As can be seen on the picture of the daughterboard (Apendix A), the soldering was a very delicate process given the proximity of the pins on the converters as well as on the connector. To make the soldering process easier, the pins which weren't used were stripped off. In the end, very clean solder was performed thanks to a wonderful product called Solder Weld, which greatly facilitated the soldering of pins to the wires without soldering them to their neighboring pins. Magnifying glasses were used to make sure there were no unnecessary/unwanted connections made accidentally. After the daughterboard was put together by gluing the individual components (ADC, DAC, Connector) to a common platform, other wires were attached

(not soldered) to specific points which we considered important to be used as probes during our testing phases, like the Timer clock output (TOUT), the transmit and receive frame sync, and the output of the DAC (input to ADC).

SOFTWARE: Interfacing the daughterboard to the EVM was no easy task. It consisted of 5 parts:

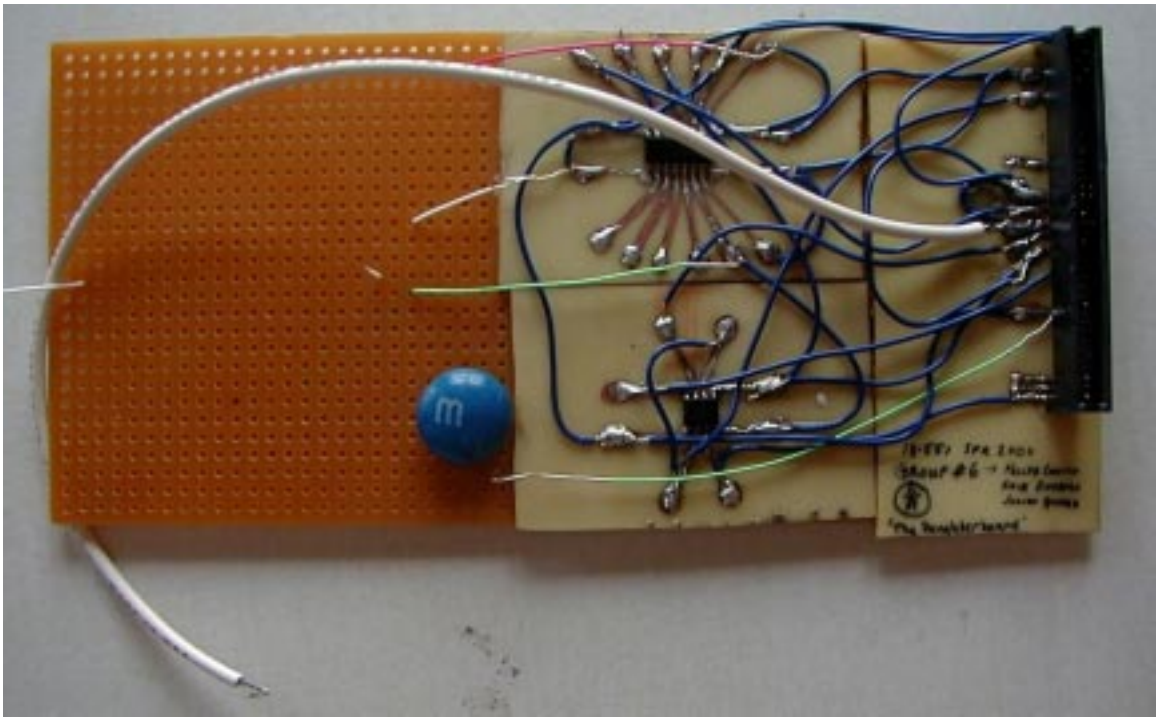
- Daughterboard Initialization: This would send a reset signal to the daughterboard, and keep XRESET high, which was inverted on the EPIC pin and connected to the converters' chip select lines (which were active low).
– McBSP Initialization: Although at first we used the Mcbsp_setup macros, in the end we decided to determine the value of each of the fields on the 8 control registers and set it by using the function mcbsp_init().
- Timer Initialization: This step selected a Timer channel and set it up to output a clock signal, which was measured with the oscilloscope to be at a rate of 13 MHz, thus allowing the converters to work at a rate of around 800 KHz.
-Interrupt Initialization: This step was taken from the echo.c we used for the class lab assignments. It initialized an XMIT and a RCV interrupt vector to copy data from a buffer to the DXR/DRR of the McBSP. The CPU interrupts were also enabled and configured at this point.
-Data Transfer: With all the configurations in place, a loop running infinitely (until cancelled), would copy data from the output buffer to the DXR and from the DRR to the input buffer using the previously declared interrupts.

Conclusion:

The most challenging part of our project involved the hardware construction and its testing. The fact that the soldering and chips were so small made it hard to maintain the proper level of quality control required for testing such implementation. For example, because the soldering process was so complicated, the iron was in contact with the copper strips and also the chip pins for somewhat extended periods of time (30-60 secs). This created the uncertainty of whether or not a part of the IC was damaged due to the prolonged heat exposure. We tried de-soldering the first IC's and soldering new ones once we had better practice with the soldering technique. This didn't yield any new positive results, which led us to believe the problem could be coming from an unwanted connection between two lines. Also, after spending extended periods of time testing the most basic of all signals, the clock pulse generated by the Timer, and not finding it, we realized that the board we were working with was probably damaged, as it worked once we moved to another lab station. On the other hand, while the software didn't demand any complex algorithms, the McBSP and Timers initializations involved the setting of 12 32-bit registers, virtually bit-by-bit. This high number of combinations, added to the uncertainty that both the EVM and the Daughterboard were not defective, made our attempt to transmit the signal through this CODEC channel unsuccessful. If we had managed to have a more controlled environment (a professionally made daughterboard, or a commercial one), our chances would have definitely improved drastically. Also, if there was more documentation on interfacing a

daughterboard to the C67 EVM specifically (i.e. other projects from professionals, or TI), we would have been better oriented in respect to the hardware and software debugging process. Anyhow, we knew since the beginning that we were getting into a challenging project, one which would involve a lot of crazy variables as well as lots of investigation about the insides of the DSP EVM. This just made our experience more exciting and fulfilling; even though we could not manage to have a completely successful demo, we feel we accomplished a whole lot. Perhaps future projects will learn from our observations and achieve a working CDMA Modem.

Apendix A:



References

A prior 551 project that did LPC compression:

http://www.ece.cmu.edu/~ee551/Old_projects/projects/s_97_8

Schumaker, Sharma, Ting, 1997

Speech Coding: A Tutorial Review, Spanias, pp. 42-56,

http://www.eas.asu.edu/~spanias/papers/review.ps

Online Tutorials for speech compression and LPC:

http://asylum.sf.ca.us/pub/u/howitt/lpc.tutorial.html

http://www.cteh.ac.il/departments/commEng/academic/staff/Noam_Amir/speech/lpc/lpc_basics2.html

Source code for LPC-10 on UNIX:

http://www.arl.wustl.edu/~jaf/lpc