CHAPTER IV

IMPLEMENTATION ON C6711 DSK

4.1 Introduction to C6711 DSK

4.1.1 DSK Testing

This example is very important for people, who begin to use CCS and DSK board. The examples will illustrate some of the features of software and hardware. Through this chapter, Users can become familiar with using CCS to control the DSK. This thesis is an important starting point for properly using the CCS and DSK by giving three main examples which are necessary to the understanding towards basic knowledge in undergraduate level.

The DSK testing can be used for confirmation of correct operation and installation. We have to launch CCS from the icon on the desktop. After that, we have to select GEL from menu bar, choose check DSK and click on Quick Test (figure 69). The following message is then displayed as seen in figure 70.

This assumes that the first three switches, USER_SW1, USER_SW2, and USER_SW3, are all in the up (ON) position. Change the switches to (110x)2 so that the first two switches are up and press the third switch down, and the fourth switch is not used.

Repeat the procedure to select GEL → check DSK → Quick Test and verify that the value of the switches is now 3 (figure 70). Moreover, you can set the value of the first three user switches from 0 to 7. Within your program you can then direct the execution of your code based on these eight values. Note that the Quick Test cycles the LEDs three times.

A confidence test program example is included with the DSK to test and verify proper operation of the major components of the DSK, such as interrupts, LEDs, SDRAM, DAM, serial ports and timers.

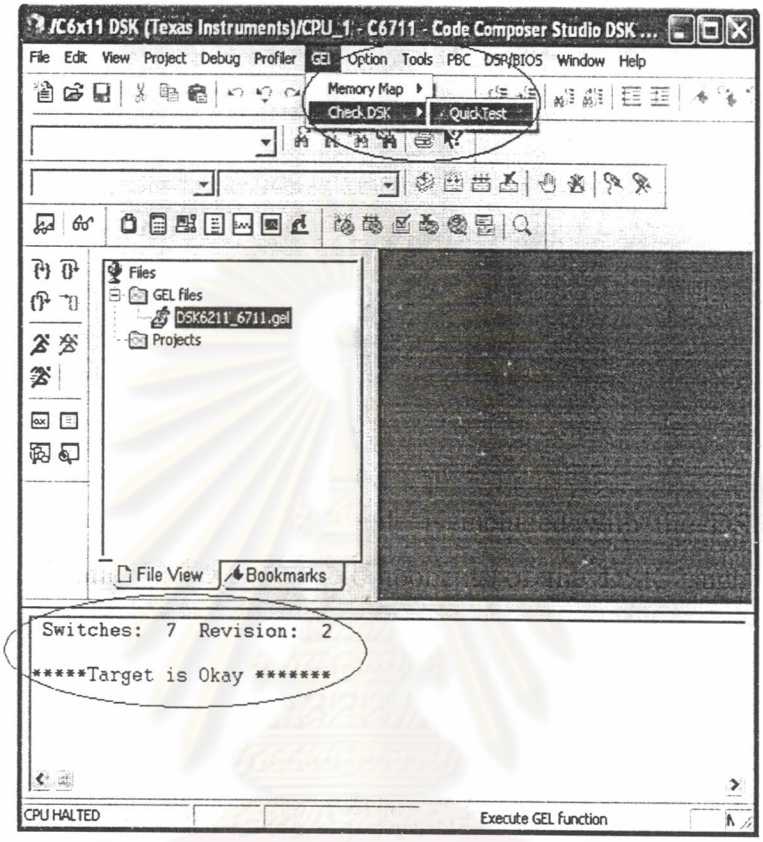


Figure 69 Quick Testing of DSK board.

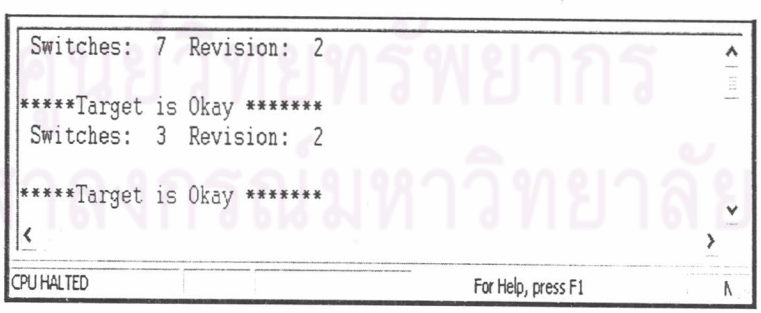


Figure 70 Quick Testing of USER_SW1, USER_SW2, and USER_SW3.

4.2 The Necessary Files

Writing C program is very difficult for checking some errors in the program. Especially, debugging of C program in TMS320C6x is also difficult so we need to write some important support files for reduce the complex problems. There are some header files (.h) for defining address, function for interrupt, and function prototypes. Other support files are initialization the DSK files (.c), linker command files (.cmd), and assembly files (.asm). Next, we will show the detail of each support file.

- C6xdsk.h: this file defines addresses of external memory interface, the serial port, etc. The file includes with TI support file.
- C6xinterrupts.h: the C6xinterrupt.h file contains init function for interrupt, which use for enable the center processing unit (CPU).
- C6xdskinit.h: this file is a header file, which contains the function prototypes.
- C6xdskinit.c: this file contains several function used for the example codec poling (codec_poll) included with CCS. It includes functions to initialize the DSK the codec, the serial ports, and for input/output signal.
- C6Xdsk.cmd: this file is one of support file, which is a simple linker command file. This file divides in to two parts: memory, and section. We can set starting address for each memory part and length of them in the first part. Next, we can set type of parameter to the right part of memory in the second part.
- Vector_11.asm: this file uses to select interrupt INT11, a branch instruction to the interrupt service routine (ISR) c_int11 located in the C program, which we write to request to using interrupt INT11.

4.3 Sine Generation with Eight Points (*sin8_intr*)

This example is just simple program for generating sine table 8 points. The benefit of this program is shown some feature of CCS for editing, building a project, accessing the code generation tools, and running a program on the C6711 processor.

The *sin8_intr* program focus on some of using CCS tools, it is useful to understand the program. This program uses some useful functions. They are necessary functions for implementing in real-time system.

1. *comm_intr* function is called that is located in the communication support file c6xdskinit.c. It initializes the DSK, the AD535 codec onboard the DSK, and the two mulitichannel buffered serial ports (McBSP) on the C6711 processor.
2. *c_int11* is a command to execution proceeds to interrupt service routine (ISR), which has address to be specific in the file vectors_11.asm.
3. *output_sample* function is located in the communication support file c6xdskinit.c. This function is called to output the first data value in the buffer.

The state *while (1)* within the function *main* use for creating an infinite loop to wait for an interrupt, which occurs every sample period T = 1/Fs = 1/8000 = 0.125 ms. It means that eight data values are output to generate a sinusoidal signal within one period. The period of the output signal is T = 8(0.125 ms) = 1ms.

Following, a step by step approach to use CCS from starting a project until showing the plotting in the CCS and subsequently in the real scope is to be demonstrated.

4.3.1 Create Project

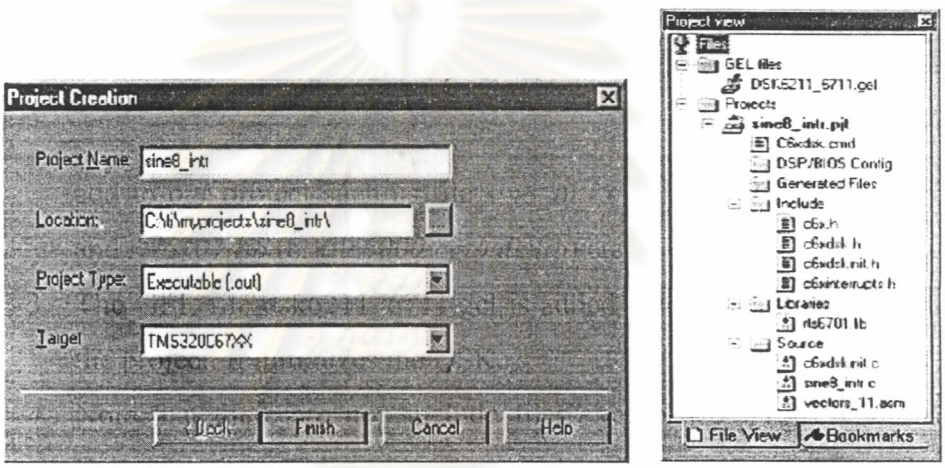
In this section we illustrate how to create a project, adding the necessary files for building the project *sin8_intr*.

1. To create the project files *sin8_intr*. Select project → New Type *sin8_intr* for project name as show in figure 4.3a. This project file is save *sin8_intr* (*c:\ti\myprojects*). The *sin8_intr. pjt* file stores project information on build options, source filenames, and dependencies.
2. To add files to project. Select project → Add files to project. Look in *sin8_intr,* Files of type C source files and open the files that we want to

add to our project such as file *sin8_intr.c*, *vectors_11.asm*, *c6xdsk.cmd*, and *rts6701.lib* (*c:\ti\c6000\cgtools\lib*) etc.

3. The GEL file dsk6211_6711.gel is added automatically when we create the project. It initializes the DSK.
4. Note that there are no "include files" yet. Select Project → Scan All Dependencies. This adds the header files: *c67xdsk.h*, *c6xdskinit.h*, *c6xinterrupts.h*, and *c6x.h*.

The complete creation the project is shown in figure 71.b.



(a) (b)

Figure 71. CCS Project View window for *sin8_intr*: (a) creating project; (b) project files.

4.3.2 Code Generation and Options

In this section, we will describe about various options, which are associated with the code generation tools: C compiler and linker to build a project.

Compiler Option

Select Project → Build Options. We will see a CCS window Build Options for compiler in figure 72. Select the following for the compiler option: (a) Basic (for

Category), (b) Speed most critical (for Target Version), (c) Full Symbolic Debug (for Generate Debug Info), (d) Speed most critical (for Opt Speed vs. size), and (e) None (for Opt Level and Program Level Opt). The resulting compiler option is

-gks

The *–k* option is keep the assembly source file *sin8_intr.am.* The *–g* option is to enable symbolic debugging information, useful during the debugging process and used in conjunction with the option *–s* to interlist the C source file with the assembly source file *sin8_intr.asm* generated. The *–g* option disables many code optimizations to facilitate the debugging process.

Select "Default" for Target Version invokes a fixed-point implementation. The C6711 based DSK can use either fixed or floating-point processing.

If No Debug is selected (for Generate Debug Info), and *–o3: File* is selected (for Opt Level), the Compiler option is automatically changed to

-ks -o3

The *–o3* option invokes the highest level of optimization for performance or execution speed. For now, speed is not critical (neither is debugging). Use the compiler option *–gks* (we can type it directly in the complier command window). Initially, one would not optimize for speed but to facilitate debugging.

Linker Option

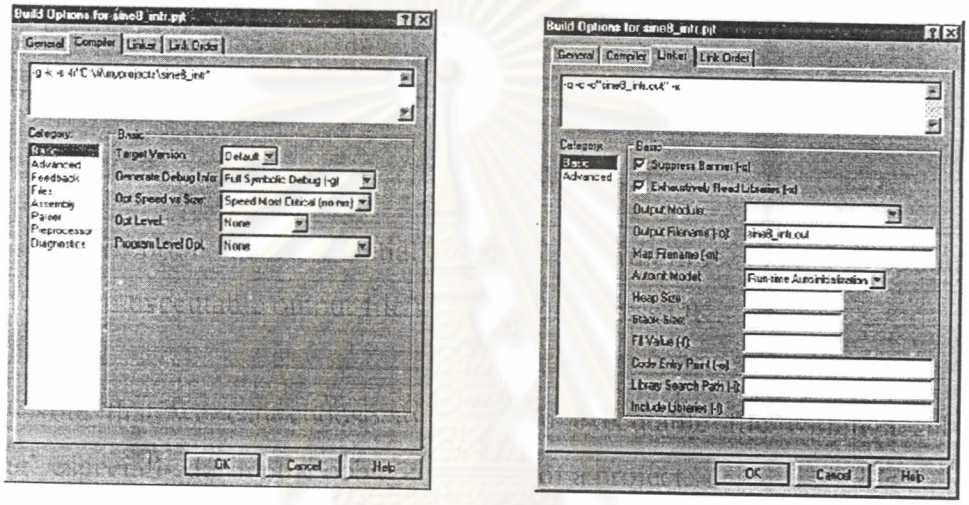
Click on linker (from CCS Build Options) and select Absolute Executable (for Output Module), *sin8_intr.out* (for Output Filename), and Run-time Autoinitialization (for Autoinit Model). The output filename defaults to the name of the *.pjt* filename. The linker option should be displayed as in figure 72(b).

- g –c –o "sin8_intr.out" -x

The *–c* option is used to initialize variables at run time, and the *–o* options is to name the liked executable output file *sin8_intr.out*. Press OK.

Note that we can choose to store the executable file within a subfolder "Debug" especially during the debugging stage of a project.

Again, these various options can be typed directly within the appropriate command windows.



(a) (b)

Figure 72 CCS Build Options: (a) compiler; (b) linker.

Building and Running the Project

The project *sin8_intr* can now be built and run

1. Build this project as *sin8_intr*. Select Project → Rebuild All. Or press the toolbar with the three down arrows. This compiles and assembly all the C files using *cl6x* and assembles the assembly files *vectors_11.asm* using *asm6x*. The resulting object files are then linked with the run-time library support file *rts6701.lib* using *lnk6x*. This creates an executable file *sin8_intr.out* that can be loaded into the C6711 processor and run. Note that the commands for compiling, assembling, and linking are performed with the Build Option. A log file *cc_build_Debug.log* is

created that shows the files that are compiled and assembled, along with the compiler options selected. It also lists the support functions that are used. Figure 73 shows several windows within CCS from the project *sin8_intr*.

2. Select File → Load Program in order to load *sin8_intr.out* by clicking on it (CCS includes an option to load the program automatically after a build). It should be in the project *sin8_intr* folder. Select Debug → Run, or use the toolbar with the "running man". Connect a speaker to the OUT connector (J6) on the DSK. We should hear a tone.

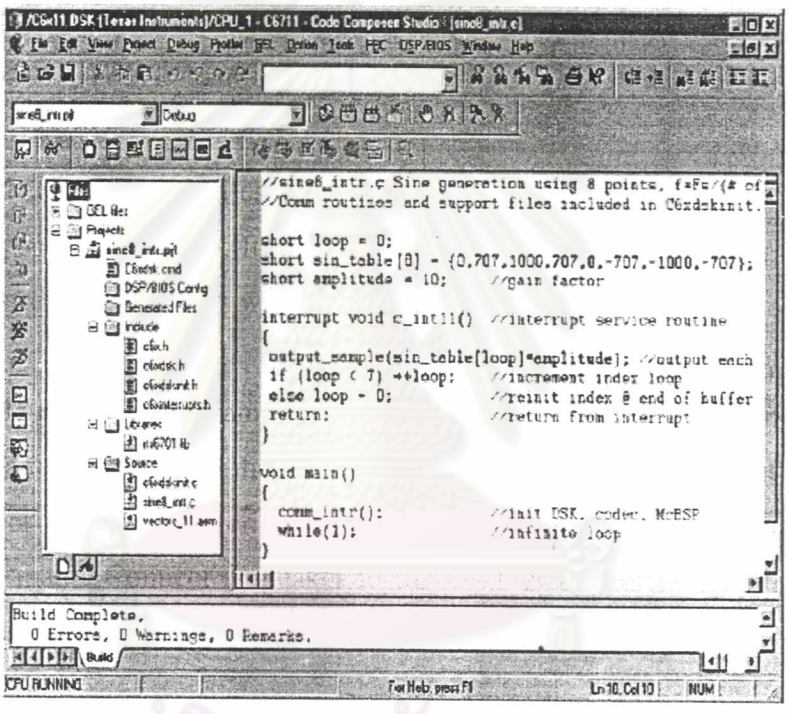


Figure 73 CCS windows for project *sin8_intr*.

The sampling rate Fs of the codec is fixed at 8 kHz. The frequency generated is f = Fs / (number of points) = 8 kHz / 8 = 1 kHz. Connect the output of the DSK to an oscilloscope to verify a 1 kHz sinusoidal signal with amplitude of approximately 0.85V p-p (peak to peak).

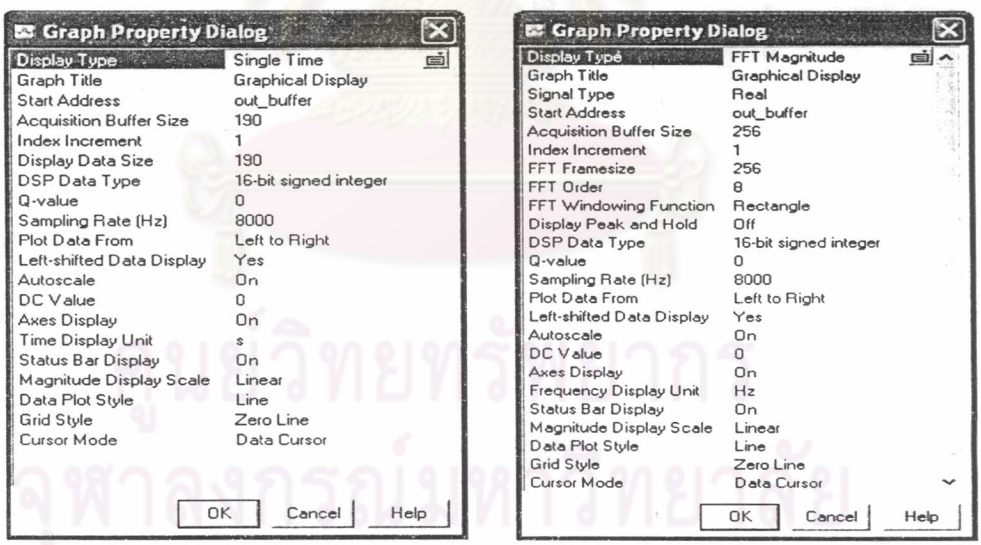
Plotting with CCS

In this section, we investigate changing some commands to the *sin8_intr* for plotting the sinusoidal wave on CCS.

The output buffer is being updated continuously every 256 points (we can readily change the buffer size). Use CCS to plot the current output data stored in the buffer *out_buffer*.

1. Select View → Graph → Time/Frequency.

2. Change the Graph Property Dialog so that the options in figure 74(a) are selected for a time-domain plot (use the pull-down menu when appropriate). The starting address of the output buffer is *out_buffer*. The other options can be left as default. Figure 75 shows both time-domain and frequency-domain plots of 1 kHz sine wave.



(a) (b)

Figure 74 CCS Graph Property Dialog for *sin8_intr _plot*

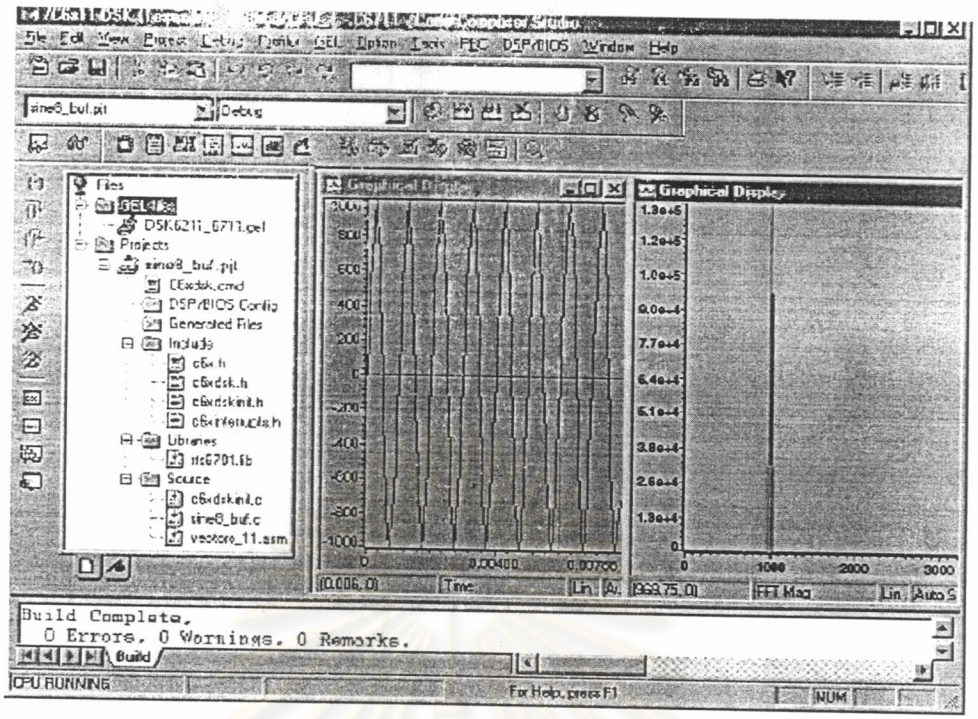


Figure 75 CCS window with both time-domain and frequency-domain plots of a 1 kHz sine wave.

4.4 BPSK Transmitter and Receiver with Phase Lock Loop (PLL) on Single Board Implementation

This experiment emphasizes on basic transmitter/receiver, which is contained in this thesis. In order to help students have a necessary understanding on the communication system. The experiment will be spited into a number of experiments such as bit generation, BPSK modulation/demodulation, and synchronization (see in figure 76).

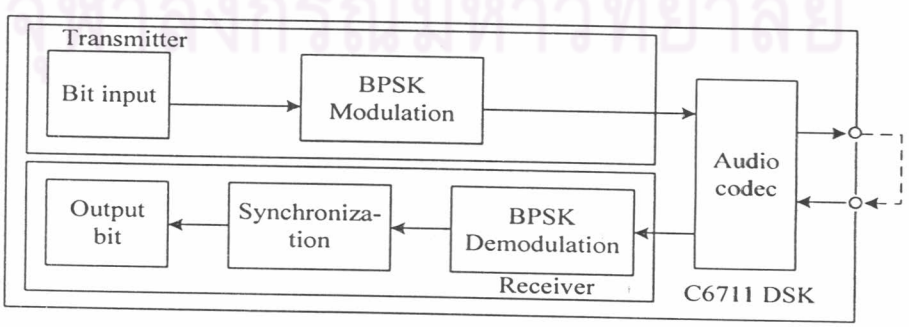


Figure 76 BPSK transmitter and receiver.

BIT INPUT: We can use either constant 10-bit digit or a bit stream output from PN bit generator as an input to the BPSK modulation.

TRANSMITTER PART: BPSK is a digital modulation technique that separates bits by shifting the carrier 180 degrees. A carrier frequency signal is chosen in order to be known by both the transmitter and the receiver. Each bit is encoded as a phase shift in the carrier at some predetermined period. When bit 0 is sent, the carrier is transmitted with no phase shift, and bit 1 is sent, the carrier is phase-shifted by 180 degrees.

RECEIVER PART: phase lock loop is used by receiver. The receiver must be able to lock onto the phase of a received signal in order to distinguish between 1s and 0s. To determine the phase of an incoming sinusoid, the maximum of the correlation coefficient is calculated between the received sinusoid and a sinusoid offset by a phase estimate. The correlation coefficient, Y, between two sinusoids is given by

$$Y = \int_0^{2\pi} \sin(t \cdot \omega + \phi_{carrier}) \cdot \sin(t \cdot \omega + \phi_{est}) \qquad (10)$$

The received sine wave has a phase of $\phi_{carrier}$, and an estimate of the phase is ϕ_{est}. The correlation coefficient has a maximum value when $\phi_{carrier}$ and ϕ_{est} are equal.

To determine this maximum, we need to begin with an initial estimate of ϕ_{est}. For every period of the incoming signal, the signal is correlated with a sine wave that has a phase slightly larger and slightly smaller than ϕ_{est}. This yields two values for the correlation coefficient, one at $\phi_{est} + \varepsilon$ and the other at $\phi_{est} - \varepsilon$. The difference between these two values gives an approximation of the derivative of the correlation coefficient. Using the difference between the correlation coefficients at $\phi_{est} + \varepsilon$ and $\phi_{est} - \varepsilon$ as an estimate of the derivative, a new value for ϕ_{est} is calculated using

$$\phi_{est} = \phi_{est} + (Y_{+\varepsilon} - Y_{-\varepsilon}) \qquad (11)$$

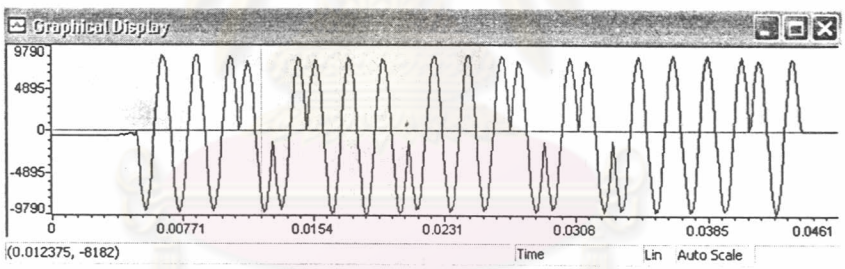
Where

$$Y_{+\varepsilon} = \int_0^{2\pi} \sin(t \cdot \omega + \phi_{carrier}) \cdot \sin(t \cdot \omega + \phi_{est} + \varepsilon) \quad (12)$$

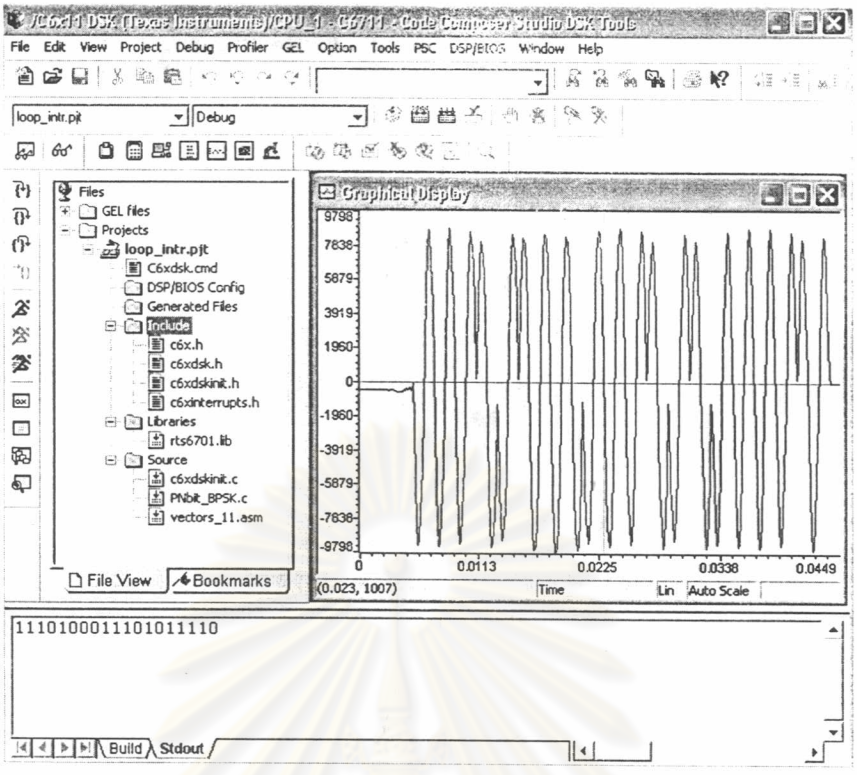
$$Y_{-\varepsilon} = \int_0^{2\pi} \sin(t \cdot \omega + \phi_{carrier}) \cdot \sin(t \cdot \omega + \phi_{est} - \varepsilon) \quad (13)$$

This process is repeated every time a full period of the incoming sine wave is received. Eventually, $\phi_{carrier}$ and ϕ_{est} will be equal and the derivative estimated by the difference in the correlation coefficient $\phi_{est} + \varepsilon$ and $\phi_{est} - \varepsilon$ will be 0. When this occurs, the receiver is considered locked onto the signal.

The result of BPSK transmitter and receiver with Phase Lock Loop (PLL) on single board is shown in figure 77(a) and 77(b).



(a)



(b)

Figure 77 (a) Receiver waveform, (b) CCS plot of a received sequence, representing a BPSK modulated signal: sequence of {1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0}.

4.5 Phase Lock Loop Implementation

This phase lock loop implementation consists of two DSKs. The first one generates a 500 Hz sine wave with eight unique phases but a constant frequency which is output to codec. Subsequently, the output signal is sent into the second DSK which is BPSK with phase lock loop (PLL) demodulator.

The sine wave is generated from first DSK which connects to scope as shown in figure 78.

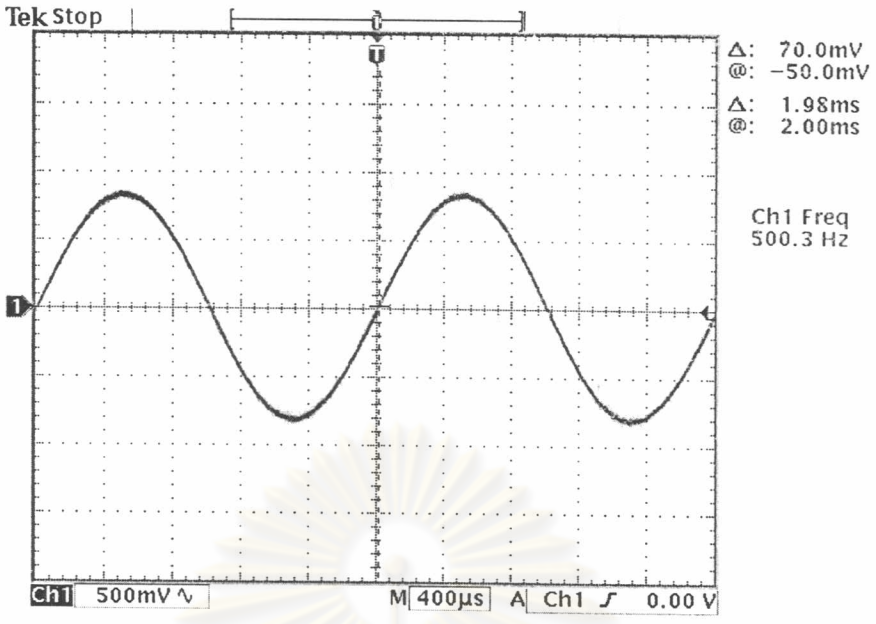
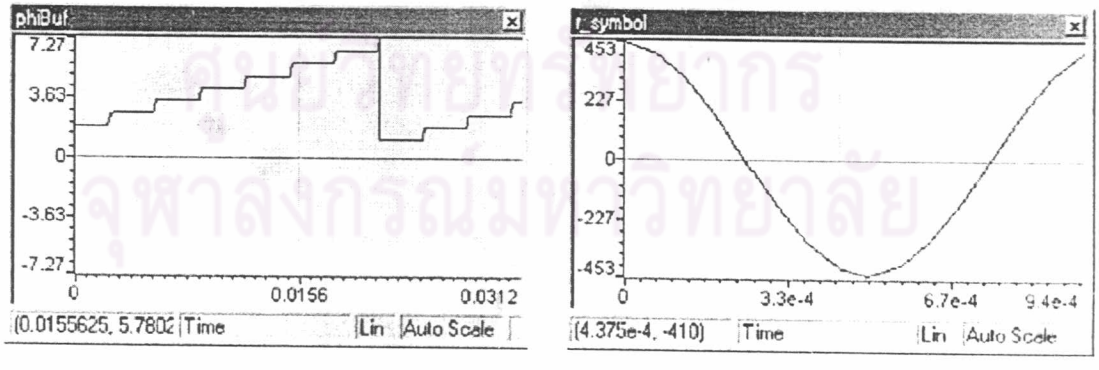


Figure 78 First DSK output of a generated 500 Hz sine wave with a varying phase.

Next, we measure the received input sinusoid on CCS from first DSK in figure 79(a). This plot is obtained within CCS using *phiBuf* as the starting address, with 500 points as the acquisition and display size, and a 32-bit float. In the figure 79(b) shows a CCS plot of the PLL output buffer that receives only one period of the sine wave. To use a starting address of *r_symbol*, an acquisition and display size of 16, and a 16-bit signed integer.



(a) (b)

Figure 79 CCS plot of a PLL demodulator: (a) output showing eight different amplitudes; (b) output buffer that receives only one period.

We measure the signal on both of DSKs for showing the sine waveform to scope. In the figure 80, there are sine waveform generated from first DSK on channel 1 and receiver sine waveform second DSK on channel 2.

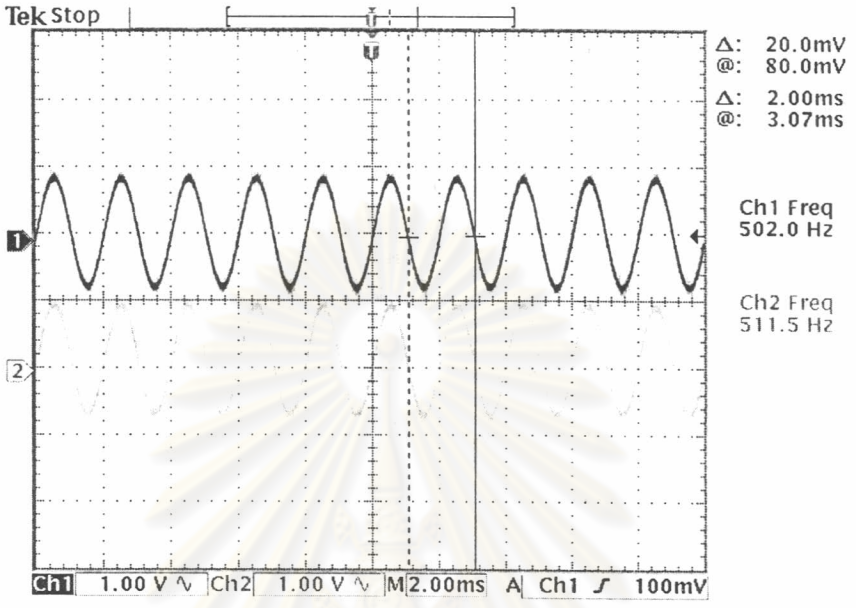


Figure 80 Sine 500Hz output from codec to scope: channel 1 is output from first DSK and channel 2 is output from second DSK

4.6 Phase Shift Keying Modulation and Demodulation Implementation

Phase shift keying (PSK) is a method of transmitting and receiving digital signals in which the phase of a transmitted signal is varied to convey information. Several schemes can be used to accomplish PSK, the simplest one being binary PSK (BPSK), using only two signal phases: 0° and 180°. If the phase of the wave is 0°, then the signal state is low, and if the phase of the wave is 180° (if phase reverses), the signal state is high (biphase modulation). More complex forms of PSK employ four or eight-wave phases, allowing binary data to be transmitted at a faster rate per phase change. In four-phase modulation, the possible phase angles are 0°, +90°, -90°, and 180°; each phase shift can represent two bits per symbol. In eight-phase modulation, the possible phase angles are 0°, +45°, -45°, 90°, -90°, +135°, -135°, and 180°; each phase shift can represent 4 bits per symbol.

4.6.1 Binary Phase Shift Keying

A single data channel modulates the carrier. A single bit transition, 1 to 0 or 0 to 1, causes a 180° phase shift in the carrier (figure 81). Thus, the carrier is modulated by the data. Detection of a BPSK signal uses the following: (1) a squarer that yields a DC component and a component at $2f_c$; (2) a bandpass filter to extract the f_c component; (3) a frequency divider, the output of which is multiplied by the input. The result is lowpass filtered to yield a PCM signal.

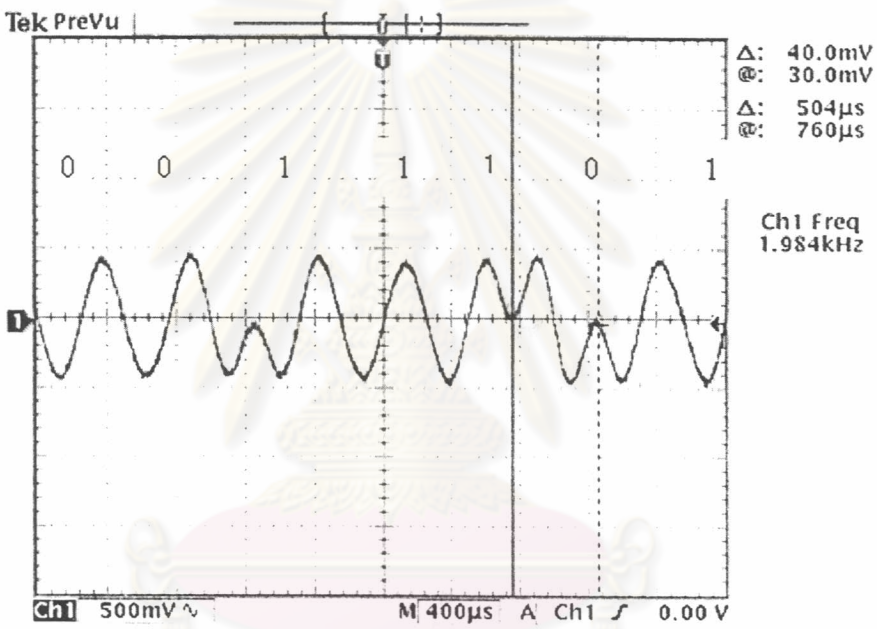


Figure 81 BPSK modulator output obtained with a scope.

4.6.2 Quadrature Phase Shift Keying

Quadrature phase shift keying (QPSK) is a modulation scheme in which the phase is modulated while the frequency and the amplitude are kept fixed. There are four phases, each of which is separated by 90° (figure 82). These phases are sometimes referred to as states and are represented by a pair of bits. Each pair is represented by a particular waveform, called a symbol, to be sent across the channel after modulating the carrier. The receiver demodulates the signal and look at the

recovered symbol for each possible combination of data bits in a pair, QPSK creates four different symbols, one for each pair, by changing an in-phase (I) and a quadrature (Q) gain.

The QPSK transmitter system uses both sine and cosine at the carrier frequency to transmit two separate massage signals, sI[n] and sQ[n], referred to as the in-phase and quadrature signals, respectively. Both the in-phase and quadrature signals can be recovered, allowing transmission with twice the amount of signal information at the same carrier frequency.

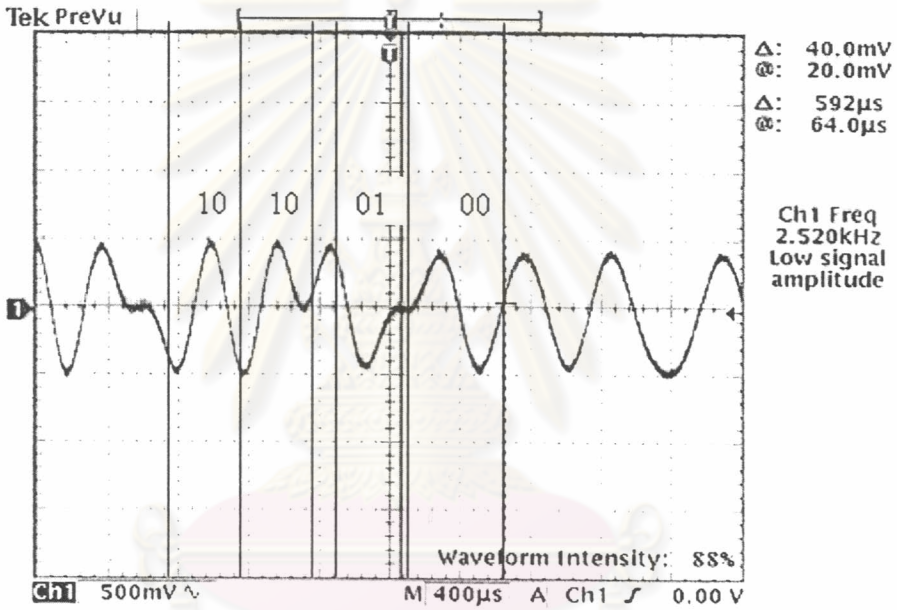


Figure 82 QPSK modulator output obtained with a cope.

4.7 Transmitter and Receiver on Separate DSK Board

This implementation uses two DSK boards to set as transmitter on first DSK which conveys the analog signal through a codec to second DSK. The second DSK is set to be receiver which demodulates some information from carrier waveform. We will give detail about the implementation below.

An input sample is obtained and stored in a memory location, which contains 16 bits. Depending on the type of PSK (two-level or four-level), appropriate masking is used. For BPSK, an input value is segmented into sixteen 1-bit components; for

QPSK, it is fractioned into 8 dibits. This is achieved by masking the input with the appropriate values, 0x0001, and 0x0003, respectively. In order to obtain the next segment to be processed, the previous input data is shifted once for BPSK or twice for QPSK.

Following the extraction of segments, values are assigned to sinusoids with corresponding phases. In BPSK, there are only two phases: 0° and 180° for bits 0 and 1, respectively. However, for QPSK, we need four phases (0°, 90°, 180°, and 270°) corresponding to 00, 01, 11, and 10 see in figure 83. This mapping is used in accordance with gray encoding. This minimizes the error caused by interference during the transmission of the signal by maximizing the distance between symbols with most different bits on constellation diagram. Each input sample is represented with 16 bits. Every sampled data contains 16 segments for BPSK, and 8 segments for QPSK. Since each symbol is transmitted by a sinusoid generated digitally by four points, an input sample is acquired every 64 and 32 output samples for BPSK and QPSK, respectively.

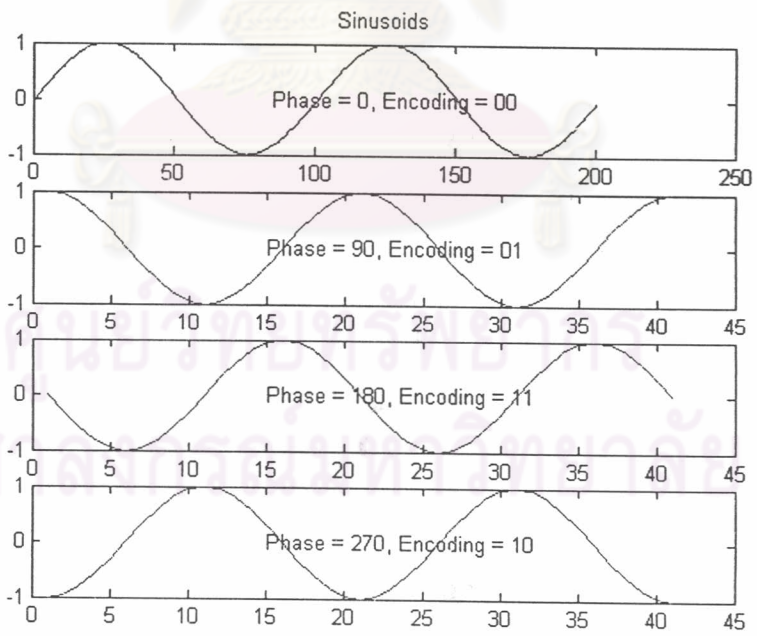


Figure 83 sinusoids with four phases (0°, 90°, 180°, and 270°)

At the PSK receiver, each sinusoid is mapped into the corresponding symbols composed of 1 bit for BPSK or 2 bits for QPSK. The extracted symbols are then aligned in the newly constructed 16-bit value by appropriate left shifts. The sample is then sent to the codec, and the original waveform is regenerated (see in figure 86 and figure 87).

Transmitting from one DSK and receiving from another DSK involves synchronization issue that requires symbol clock recovery and adaptive equalizer (using a PLL).

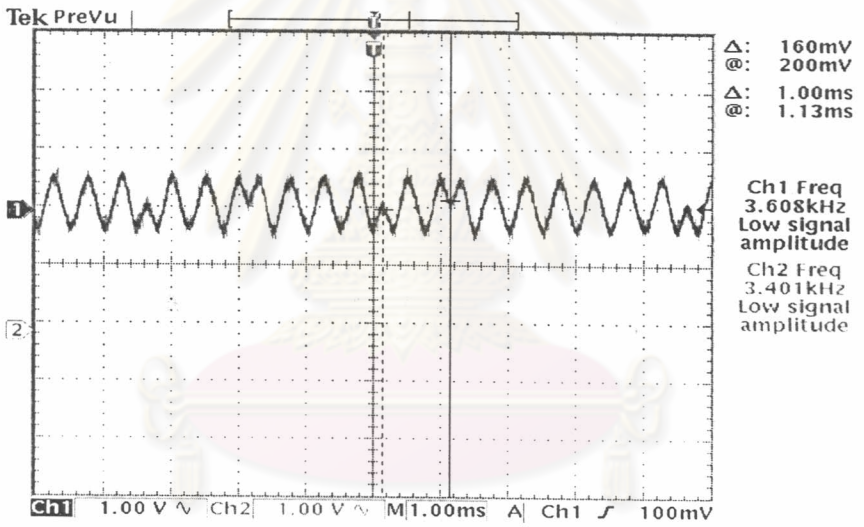


Figure 84 BPSK output from codec to scope: channel 1 is output from first DSK and channel 2 is output from second DSK.

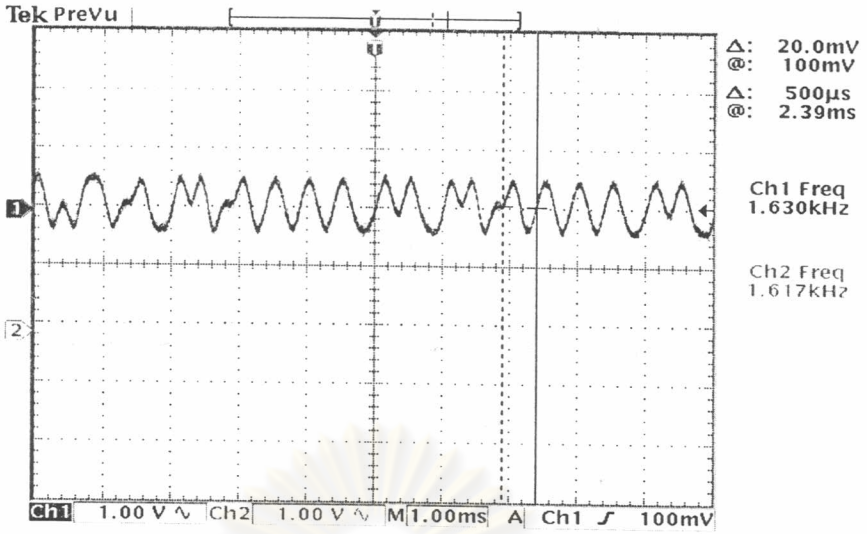


Figure 85 QPSK output from codec to scope: channel 1 is output from first DSK and channel 2 is output from second DSK.