Teaching Real-time Sonar With The C6711 DSK and MATLAB

George W.P. York, Cameron H.G. Wright / Michael G. Morrow, Thad B. Welch
U.S. Air Force Academy, CO / U.S. Naval Academy, MD

Abstract

A sonar system serves as an excellent platform for teaching DSP topics such as beamforming, sampling theory, demodulation, filtering, image processing, and Doppler velocity estimation. While MATLAB simulations are useful for teaching the basic theory, many of these concepts are more easily taught to undergraduates if appropriate real-time demonstrations and laboratory experiences are available. The challenge of transitioning from MATLAB to real-time hardware is often the expense and a steep learning curve for the students. This paper describes a real-time DSP sonar educational platform based around the programming ease of MATLAB and the low-cost Texas Instruments C6711 digital signal processing starter kit. Classroom uses of this platform are discussed; the software is available at `http://www.usna.edu/EE/links/ee_links.htm`.

1. Introduction

The components of a DSP-based sonar system (phased-array transmitter/receiver; beamformer; time-gain compensation, multi-rate sampling, quadrature demodulation, filtering, image processing, Doppler velocity estimate, etc) are at the heart of numerous military and commercial systems. In addition to sonar systems, these components can be found in radar, medical ultrasound, satellite communications, cellular/PCS, and software radios[1-7]. Historically, the theoretical aspects of sonar systems have been covered in a graduate level DSP course. However, given the diverse topical interest in concepts such as beamforming, we believe that an understanding of the theories and implementation techniques necessary to construct the various subsystems of sonar are becoming an essential part of an undergraduate EE education.

2. Teaching Sonar: Software or Hardware?

How do we teach such concepts to undergraduates? Computer-based demonstrations can be highly effective with students for many DSP topics[8]. We can take advantage of the fact that the software package MATLAB[9] and its related toolboxes have become a mainstay in most EE programs. Given our students' familiarity with MATLAB, computer exercises that implement sonar theory seem to be a natural approach. But where does the sensor array data come from to demonstrate beamforming? Shall our students generate MATLAB simulated array data? Even if this data is realistic in nature, the time spent generating this data may detract from our primary pedagogical objective. Should the professor generate the data or perhaps provide real data? Another consideration is that our students are not impressed with a "canned demo." Ideally then, we would have a real-time system to use as a teaching tool.

In the past, proceeding beyond a MATLAB-only simulation to a real-time hardware implementation has been impeded by a very abrupt transition, in terms of both cost and the learning curve of unfamiliar systems and software. By developing a software and hardware bridge between MATLAB and real-time DSP hardware, we have made it possible to smoothly and incrementally transition from simulation to a full hardware implementation, all while retaining the impressive capabilities of the MATLAB display engine. Using this approach, students are able to develop and enhance their own sonar system, experimenting with the various processing stages like beamforming.

3. Hardware Requirements

Our students are already very familiar with MATLAB, but we also want them to learn more about hardware-based digital signal processing (DSP) and this seems to be a perfect opportunity. For the primary DSP hardware, our main criteria were low cost, sufficient processing power, and a versatile software development environment. We chose to construct our educational platform around the Texas Instruments C6711 DSK, which makes use of the VLIW/SIMD architecture of the TMS320C6711 microprocessor. We have had good results with teaching other DSP concepts using the C6x DSK, and felt that sonar would also benefit from this approach[10].

The C6711 DSK has the following advantages:

- An excellent software development environment (Code Composer Studio)
- Performs roughly 1.2 billion instructions per second and 600 MFLOPS
- Plenty of memory (4 MB)
- Relatively inexpensive ($195 academic price)
- Offers both floating point (2 single precision per cycle) and SIMD fixed point (4 16-bit per cycle)

The C6711 DSK's principle disadvantage is that it only has a telephone quality (maximum 8 kHz), single channel codec. To minimize the cost we decided to design for audio frequencies. Most of the available audio frequency ADCs are either one, two, or four channel devices. A four-channel system was selected since a two-channel system did not provide beams that were well defined, and an eight-channel system was unnecessary for an educational platform. For ease of implementation, we desired simultaneous sampling of all four input channels.

For a cost of $99, TI offers a 12-bit, four-channel, simultaneous sampling (f_s = 150 kHz/channel) ADC daughtercard compatible with the C6711 DSK known as the THS1206 Evaluation Module (EVM).   This is over-designed for our audio range, but ideal for 50 kHz ultrasound transducers planned for a future upgrade. A small battery-powered preamplifier was constructed to provide the correct signal level from the microphones to the ADC, and could easily be assembled by students.

The ADC is triggered directly by a DSK Timer so that it places no burden on the DSK CPU resources. At the end of each conversion set, the CPU is interrupted and it reads the four samples from the converter. If additional CPU resources are required for processing, the DSK code could be modified to offload this task to the CPU's direct memory access (DMA) controller. In this particular implementation that was unnecessary.

The 4 input channels were designed for beamforming for a 4-element array receiver. For the transmit beam, we are currently using the C6711 DSK single channel codec driving an omni-directional speaker. While transmitting and receiving both with a phased array would offer a higher quality (and more complicated) system, the omni-directional transmit followed by beamforming with a phased array is sufficient for our pedagogical purpose and reduces the cost and computational load.

4. Educational Requirements

While are students are not proficient at DSP programming, TI's Code Composer Studio or C6711 C or assembly coding, they do know how to use MATLAB. What we needed was a tool that allowed for algorithm development in MATLAB. Once the student was comfortable with what they had learned, it would facilitate the migration of the algorithm---in part or whole---onto the DSP hardware. The desired progression would be as follows.

1. Study the traditional DSP theory,
2. Use MATLAB with simulated data,
3. Use MATLAB with real-world data,
4. Implement the process (in part or whole) in real-time on the TI DSK hardware
5. Repeat to improve the design or to develop new features.

The third step of this process presents a practical problem. While MATLAB now has a very capable data acquisition (DAQ) toolbox that allows for direct data acquisition and data insertion into the MATLAB workspace which works with a number of different DAQ hardware boards, it does *not* support programmable DSP systems such as a DSK. Even if the DAQ Toolbox could somehow be used with a DSK, you could not avoid the fact that this method would be too slow to allow the transition to step four: a real-time implementation. Since we wish to minimize multiple software environments in the interest of time (for students and faculty), a single development environment solution is highly desirable. For this reason, we developed a direct MATLAB to DSK interface.

5. MATLAB to DSK Interface Software

The interface between MATLAB and the DSK is encapsulated into a generalized interface command set that supports multiple input and output channels, variable sample rates, various triggering configurations, and variable frame sizes. The specific commands available are described in the Appendix. The interface was developed using MATLAB 's “mex” facility and Microsoft Visual C++, and is centered on an object that encapsulates the hardware interface between the host PC and the DSK. The TI application-programming interface (API) furnished with the DSK allows operation under Windows 9x/NT. Our interface software requires that the DSK tools be installed on the computer, and that the two files C6X_DAQ.DLL and DAQ_SIMUL.OUT be placed in a MATLAB-accessible directory. At the most basic level, this interface allows a novice user to operate the DSK as a data acquisition board with a simple command sequence, with no requirement to know how to use Code Composer or how to program in C.

Initially, all signal processing can be done in the MATLAB environment using “live” data acquired from the DSK. As the students progress, they can move processing functions from MATLAB down to the DSK by altering the DSK code (that was used to create the DAQ_SIMUL.OUT file), and still continue to use MATLAB as a graphical display engine.

The interface’s ease of use is best illustrated by the sample MATLAB m-file listed below. The m-file is the complete sequence of commands necessary to use MATLAB and the DSK to form a single-channel real-time oscilloscope.

```
% codec_scope.m
%
% initialize the DSK parameters
c6x_daq('Init', 'daq_codec.out');
c6x_daq('FrameSize', 500);
Fs = c6x_daq('SampleRate', 8000)
numChannels = c6x_daq('NumChannels', 1)
c6x_daq('TriggerMode', 'Auto');
c6x_daq('TriggerSlope', '+');
c6x_daq('TriggerValue', 0.2);
c6x_daq('TriggerChannel', 1);
c6x_daq('LoopbackOn');
c6x_daq('QueueSize', 100);
c6x_daq('FlushQueues');
c6x_daq('GetSettings');

% do double-buffered plotting for speed
data = c6x_daq('GetFrame');
P1=plot(data(:,1),'g');
axis([0 FrameSize -1.1 1.1])
set(gcf,'doublebuffer','on')

while 1 > 0
      data = c6x_daq('GetFrame');
      set(P1,'ydata',data)
      drawnow
end
```

This interface allows complete control over the acquisition process with no knowledge of the actual DSK software or hardware operation. Sample m-files, and the source code to support the DSK's single channel CODEC, are available from the authors.

6. Phased-Array Sonar Stages / Algorithms

Given this MATLAB-DSK platform, we teach these primary sonar processing stages/algorithms.

6.1 Omni-Directional Transmitter

With a single speaker available, we transmit an omni-directional pulse. We start with a 2-cycle sinusoid (corresponding to a spatial resolution ~4 ft). Having a fully programmable sonar system allows the students to easily explore concepts such as the trade-off of pulse length (detectability) versus spatial resolution, and more elaborate chirps (e.g., broadband chirp sweeping a frequency range) than a simple narrowband sinusoid.

6.2 Phased-Array Receiver: TGC and Beamforming

With a four-channel system, a uniformly spaced linear sensor array consisting of four omni-directional microphones is depicted in Figure 1. After sampling a frame of 2000 samples at 100K samples/sec/channel, the data from each channel is first equalized with respect to each channel assuming the root-mean-square error should be the same. Since the sound wave attenuates over distance, the time gain compensation (TGC) stage amplifies the channel data with respect to depth. The students can do a quick experiment to determine if the attenuation is linear or logarithmic.

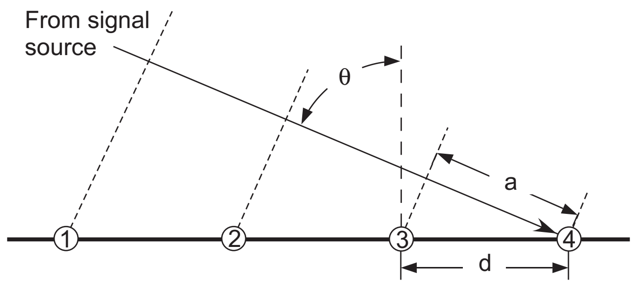

Figure 1. A linear sensor array consisting of four omni-directional microphones, labeled 1-4. $d = \lambda/2$

We then start our students with the most basic beam-forming algorithm: delay-and-sum[11]. Assuming the sensor array is in the far field region of the signal source, then the arriving wavefronts may be assumed to be linear. Then, for each sample integer delay *n*, the respective beam angle θ (angle normal to the sensor array axis in Figure 1 can be computed from

$$\sin\theta = \frac{a}{d} = 2n\frac{f}{f_s}$$

where f is the frequency of the signal and f_s is the sample frequency[11]. We selected $f = 1$ kHz and $f_s = 100$kHz, which leads to an ability to form 101 different beams for $-90° < \theta < 90°$.

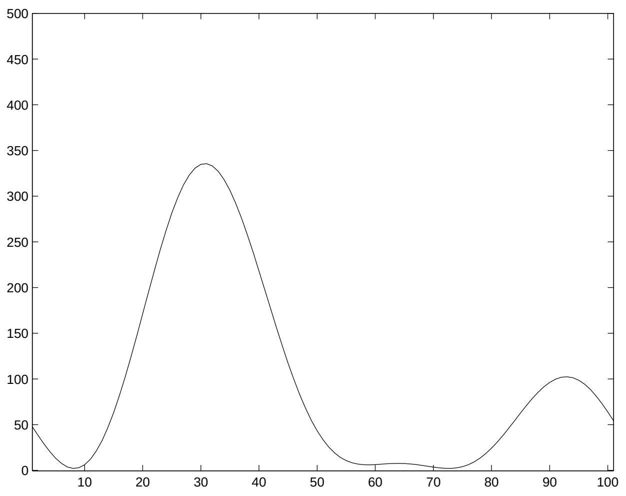

Figure 2. An example of the real-time DSP beamformer display for a signal located at beam 30, or where $\theta \approx 24.8°$ left of broadside. Beam 92 shows a reflection.

We first demonstrate the real-time DSP beamformer by receiving a continuous 1 kHz wave, and plotting in MATLAB the received energy for each beam angle (summing over many samples in time) as shown in Figure 2. The *y*-axis is non-normalized beam energy. On the *x*-axis, 1 <= *x* <= 50 are the beams left of broadside and 52 <= *x* <= 101 are the beams right of broadside, and beam 51 is broadside (or $n = 0$). The signal in Figure 2 is located at beam 30 (21 delays left of center), which equates to $\theta = \arcsin\left(21/50\right)$ or $\theta = 24.8°$ left of broadside. Beam 92 shows a reflection from a wall. The display can be modified to show angle directly, but this format is instructive for the students. Our students discover the usefulness of 101 beams using a four-element array, along with other beamforming considerations such as the effect of the number of sensors, the sensor element spacing, and the sensor element weighting on the resulting beam characteristics.

We then demonstrate the basics of real-time sonar by transmitting the 2-cycle pulse, receiving 2000 samples (corresponding to a spatial depth of about 20 ft) per channel, and plotting an image of the 101 beams (x-axis) versus depth (y-axis) before demodulation as shown in Figure 3.

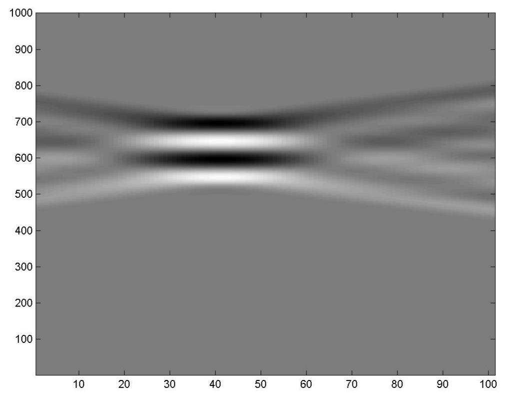

Figure 3. Echo image of reflected 2-cycle pulse before demodulation (x-axis is 101 beams; y-axis is distance)

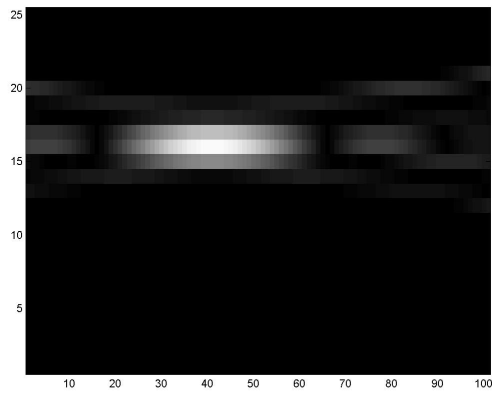

Figure 4. Echo Image of 2-cycle pulse after demodulation and 40:1 decimation.

6.3 Demodulation

Demodulation is then used to remove the 1 kHz carrier frequency to recover the echo signal. In quadrature demodulation the received signal is multiplied by $\cos(2\pi ft) + j\sin(2\pi ft)$ resulting in a complex signal $I(t) + jQ(t)$, retaining both magnitude and phase of the signal[7]. A low pass filter is then performed to remove the duplicate signal at 2*f*, leaving the echo signal as the DC component. At this stage, the student is able to practice basic DSP concepts such as designing a digital FIR filter for the correct cut-off, seeing trade-offs of simple uniform rectangular filters, Bartlett, Hamming, Hanning, and Blackman filters[1] and practical limitations such as window size and computation time. This stage can also be used to teach multi-rate filtering and decimation, as our signal is highly over sampled (i.e., 1 kHz versus 100 kHz).

The echo image can then be computed by taking the magnitude of the signal, $B(t) = \sqrt{I^2(t) + Q^2(t)}$, as shown in Figure 4, after decimating 40 to 1. Note the poor resolution using a 2 cycles of 1 kHz. We then demonstrate to the students the improved resolution using a higher frequency pulse. When more complex signals, such as chirps, are transmitted, we demonstrate recovering the signal using a matched filter (correlation) with a known chirp signal.

6.4 Other Filters

A sonic image is often noisy, so we use this opportunity to teach some speckle reduction techniques, such as temporal compounding (persistence)[7]. Temporal compounding enhances stationary signals while reducing the time-varying noise by averaging the current unfiltered image, B_{in}, with the previous output image, B_{out}:

$$B_{out}(k) = a \cdot B_{out}(k-1) + (1-a)B_{in}(k)$$

where k is the frame number and a is the weight (or persistence). Having a real-time DSK sonar system (versus MATLAB simulation only) is very beneficial for the student to see the impact of a persistence filter at the actual frame rate. Persistence strengthens stationary signals, but too much persistence caused blurring of fast moving objects.

6.5 Polar Conversion

Another practical consideration is illustrating the trouble involved in implementing the simple geometric transformation of the data (stored in memory in rectangular coordinates, Figure 4) into the proper polar coordinates for displaying the echo image (Figure 6). Each Cartesian output pixel value *P(x,y)* must be interpolated from its respective surrounding polar vector data $B(\theta, r)$ by (1) calculating the address in memory (or array indexes) of the input data $B(\theta, r)$ (i.e., a polar conversion, requiring $\theta = \arctan(y/x)$ and $r = \sqrt{x^2 + y^2}$ for each output pixel *P(x,y)*; (2) calculating the appropriate interpolation coefficient weights by the spatial fractional offset between each *P(x,y)* and its neighboring polar $B(\theta, r)$; and (3) computing the interpolation[12]:

$$P(x,y) = (1-\beta)\left[(1-\alpha)B(\theta,r) + \alpha B(\theta+1,r)\right] + \beta\left[(1-\alpha)B(\theta,r+1) + \alpha B(\theta+1,r+1)\right]$$

where α is the fractional offset in the angle θ direction and β is the fractional offset in the range r direction, illustrated in Figure 5.

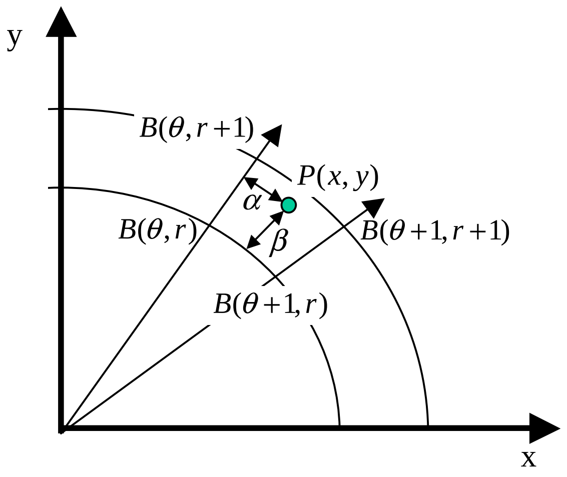

Figure 5. Interpolation for Polar Conversion

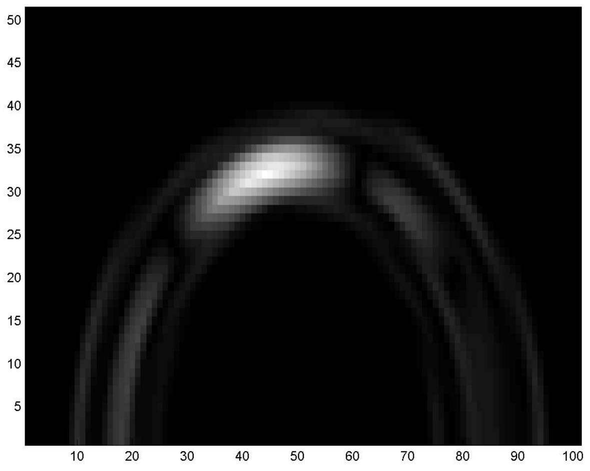

Figure 6. Echo Image of Figure 4 after polar transformation.

This polar conversion is a useful teaching aid for illustrating interpolation and aliasing problems, as well as real-time computing considerations. To aid real-time computation, we use a simple bilinear interpolation and precompute the computationally intense operations in steps 1 and 2 in a lookup table.

6.6 Doppler Imaging

If there is time in the course we also like to introduce methods for Doppler imaging and computing the velocity of moving objects. Since our system is fully programmable, we can demonstrate both continuous wave (using a remote speaker target) or pulsed Doppler techniques (using omni-directional transmitter). Spectral Doppler (or Spectrogram) can be demonstrated by continuously sampling the same range bin for several ensembles, the calculating and plotting the FFT of the ensembles versus time[7]. The frequency is proportional to the velocity toward or away from the sensor array.

To create a spatial velocity image (corresponding to the echo image of Figure 6), the FFT is too computationally intense to compute for each pixel in real-time. Thus, an autocorrelation technique can be presented, by estimating the velocity at each range bin by computing the change in phase, $\Delta\phi$, at each range bin t:

$$\Delta\phi(t) = \arctan\left(\frac{\sum_{e=0}^{E-2} (Q_e(t) I_{e+1}(t) - I_e(t) Q_{e+1}(t))}{\sum_{e=0}^{E-2} (I_e(t) I_{e+1}(t) - Q_e(t) Q_{e+1}(t))} \right)$$

where the denominator and numerator are respectively the real and imaginary part of the first lag of autocorrelation, and E is the ensemble size (or number of frames), typically varying from 4 to 16[7].

7. Classroom Uses of the System

After learning the sonar theory and implementing the appropriate algorithms in MATLAB, students can benefit greatly from seeing their work in action. Echo energies can be calculated to identify the location of a given reflector either off-line or in real-time, but the real-time operation seems to have a far greater (and more enduring) learning effect on students. They have considerable fun moving the reflectors around while the sonar display shows the movement.

The real-time implementation on the DSK, is noticeably faster than the MATLAB implementation as shown for the delay-and-sum algorithm in Table 1. If there is time in the course, we also introduce students to real-time programming issues, such as floating point versus fixed point, C versus assembly, SIMD operations, software pipelining, cache versus DMA data transfers, and double buffering[12].

Table 1. Performance excecuting delay-and-sum algorithm. Host PC: 350 MHz

Pentium II, with 64 MB Ram, windows 98; 6711 DSK: 150 MHz

Calculation Method	Screen Updates (frames/sec)
Brute force MATLAB	0.8
Vectorized MATLAB	3.3
DSK Hardware	15.0

Now that the system has been built, it will be used in a number of courses during the next year at both the Naval and Air Force Academies. Once the students become familiar this sonar development system in a course such as *Advanced DSP*, it will make an excellent springboard for students to develop more advanced projects for their Senior Design projects (with hopefully a greater success rate). We believe this approach to teaching DSP applications will develop better student skills in MATLAB, C, C++, algorithm development, system interfacing, and integration. Finally, this system provides the ability to expose our EE students to DSP system development not only in traditional DSP courses, but also in communications and computer engineering courses. Multidisciplinary exposure to the power of hardware-based DSP will help to develop needed skills in the next generation of DSP engineers.

8. Conclusions

We have developed an educational framework that will allow our students to smoothly transition from multi-channel high-speed data acquisition to a real-time DSP system implementation such as beamforming and sonar. This process allows real-world data to be gathered and used in the algorithm development and design process while maintaining a link to MATLAB.

The authors freely distribute the software portion of this system for educational, non-profit use, and invite user comments and suggestions for improvement. This package also includes *DAQ_CODEC.OUT*, a file to support data acquisition using the DSK's native codec. The software may be downloaded from *http://www.usna.edu/EE/links/ee_links.htm* and interested parties are invited to contact the authors via e-mail.

Bibliography

1. Oppenheim, A.V., *Applications of Digital Signal Processing.* Prentice Hall, 1978.
2. Skolnik, M.L., *Introduction to Radar Systems.* McGraw-Hill, 1980.
3. Burdic, W.S., *Underwater Acoustic System Analysis.* Prentice Hall, 1984.
4. Johnson, D.H. & Dudgeon D.E., *Array Signal Processing: Concepts and Techniques.* Prentice Hall, 1993.
5. Lee W.C.Y., *Mobile CellularTelecommunications: Analog and Digital Systems.* McGraw-Hill, 1995.
6. Manolakis, D.G., Ingle V.K., & Kogon S.M., *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering, and Array Processing.* McGraw-Hill, 1995.

7. York G. & Kim Y., "Ultrasound Processing and Computing: Review and Future Directions," Chapter in *Annual Review of Biomedical Engineering*, Vol. 1, 1999, pp 559-588.
8. Yoder, M.A., McClellan, J.H., & Schafer, R.W., "Experiences in teaching DSP first in the ECE curriculum," in *Proceedings of the 1997 ASEE Annual Conference*, June 1997. Paper 1220-06.
9. The MathWorks, Inc., Natick, MA, *MATLAB: The Language of Technical Computing*, 1999.
10. Morrow, M.G., Welch, T.B., & Wright, C.H.G., "An inexpensive software tool for teaching real-time DSP," in *Proceedings of the 1st IEEE DSP in Education Workshop*, (Hunt TX), IEEE Signal Processing Society, Oct 2000.
11. Morrow, M.G., Welch, T.B., Wright, C.H.G., & York G.W.P., "Demonstration Platform for Real-Time Beamforming," *26th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Salt Lake City, UT, May 2001.
12. York G., Basoglu C., and Kim Y., "Real-Time Ultrasound Scan Conversion on Programmable Mediaprocessors", *SPIE Medical Imaging*, Vol. 3335, 1998, pp. 252-262.

GEORGE W.P. YORK, PhD, is currently assigned to Ft Meade, MD, and will be returning to teach at U.S. Air Force Academy in 2002. From 1994–1997 he was an Assistant Professor in the Department of Electrical Engineering at USAFA. His research interests include signal and image processing, embedded computer design, and ultrasound imaging. He is a member of ASEE and IEEE. Email: george.york@ieee.org

CAMERON H. G. WRIGHT, PhD, PE, is an Associate Professor in the Department of Electrical Engineering at the U.S. Air Force Academy. His research interests include signal and image processing, biomedical instrumentation, communications systems, and laser/electro-optics applications. He is a member of ASEE, IEEE, SPIE, NSPE, Tau Beta Pi, and Eta Kappa Nu. Email: c.h.g.wright@ieee.org

MICHAEL G. MORROW, PE, is a Faculty Associate in the Department of Electrical and Computer Engineering at the University of Wisconsin. Previously he was a Master Instructor in the Department of Electrical Engineering at the U.S. Naval Academy. His research interests include real-time digital systems, embedded systems, and software engineering. He is a member of ASEE and IEEE. Email: morrow@ieee.org

THAD B. WELCH, PhD, PE, is an Assistant Professor in the Department of Electrical Engineering at the U.S. Naval Academy. From 1994–1997 he was an Assistant Professor in the Department of Electrical Engineering at the U.S. Air Force Academy. His research interests include multicarrier communication system design and analysis, RF channel measurements, and real-time signal processing. He is a member of ASEE and Eta Kappa Nu and a senior member of the IEEE. Email: t.b.welch@ieee.org

Appendix: API Function for C6x DSK to MATLAB Interface

Command argument types are denoted as follows:

< >	optional argument
#	numeric argument
‘arg’	text argument
X	Matlab variable

Command	Syntax	Description
Init	C6X_DAQ(‘Init’, ‘Filename’, <#>)	Initializes the C6X DSK and the Matlab interface. The filename of the desired COFF file must be supplied. The optional # argument is used to specify which parallel port to use (1 or 2). The default is 1.
Version	C6X_DAQ(‘Version’)	Displays the version numbers of the C6X_DAQ.dll and DAQ.out files in use.
GetSettings	C6X_DAQ(‘GetSettings’)	Displays the current settings in use.
LoopbackOn	C6X_DAQ(‘LoopbackOn’)	Echoes DSK input data directly to the DSK output. Useful for monitoring.
LoopbackOff	C6X_DAQ(‘LoopbackOff’)	Turns off loopback.
QueueSize	X = C6X_DAQ(‘QueueSize’, #)	Sets the queue size to the value of the second argument, or the maximum queue size, whichever is less. Returns the actual queue size.
FlushQueues	C6X_DAQ(‘FlushQueues’)	Flushes the transmit and receive queues on the DSK.
FrameSize	X = C6X_DAQ(‘FrameSize’, #)	Sets the frame size to the value of the second argument, or the maximum frame size, whichever is less. Returns the actual frame size.
NumChannels	X = C6X_DAQ(‘NumChannels’ #)	Sets the number of active channels to the value of the second argument, or the maximum supported channels, whichever is less. Returns the actual number of active channels.
SampleRate	X = C6X_DAQ(‘SampleRate’, #)	Sets the sample rate to the value of the second argument, or the maximum/minimum sample rate, whichever is less. Return the actual sample rate.
TriggerMode	C6X_DAQ(‘TriggerMode’, ‘arg’)	Sets the trigger mode to one of three mode values – ‘Auto’, ‘Immediate’, or ‘Normal’.
TriggerSlope	C6X_DAQ(‘TriggerSlope’, ‘arg’)	Sets the trigger slope to positive (‘+’) or negative (‘-‘).
TriggerValue	C6X_DAQ(‘TriggerValue’, #)	Sets the trigger value to the passed value. This should be a number such that –1.0 < x < +1.0.
TriggerChannel	C6X_DAQ(‘TriggerChannel’, #)	Sets the trigger channel to the passed value.
GetFrame	X = C6X_DAQ(‘GetFrame’)	Gets a frame of data from the DSK, and returns it in the matrix X. The data is organized on a column per channel basis.
SendFrame	C6X_DAQ(‘SendFrame’, X)	Sends the frame of data in X to the DSK. X must be the correct size for the current frame size and number of channels.
SwapFrame	C6X_DAQ(‘SwapFrame’, X)	Sends the frame of data in X to the DSK, then retrieves a frame of data from the DSK. X must be the correct size for the current frame size and number of channels.
UserRead	C6X_DAQ(‘UserRead’, #1, #2, X)	Performs a user-defined read of DSK data by passing first the command (#1) to the DSK, then reading #2 elements of 32 bit data from a buffer on the DSK into the MATLAB variable X.
UserWrite	C6X_DAQ(‘UserWrite’, #1, #2, X)	Performs a user-defined write to the DSK by passing first the command (#1) to the DSK, then writing #2 elements of 32 bit data from MATLAB variable X to a buffer on the DSK.
Close	C6X_DAQ(‘Close’)	Closes the DLL connection with the DSK.