REAL-TIME DSP USING "SEE-THROUGH"

Cameron H. G. Wright
Department of Electrical and Computer Engineering
University of Wyoming, WY

Thad B. Welch
Department of Electrical and Computer Engineering
Boise State University, ID

Michael G. Morrow
Department of Electrical and Computer Engineering
University of Wisconsin-Madison, WI

Abstract

Engineering educators teaching digital signal processing (DSP) have found that using hands-on exercises for students can smooth the transition from theory to practical real-time DSP. However, before significant learning can begin using such exercises, students must build their confidence in the hardware and software platforms. When using audio signals, a "talk-through" project accomplishes this. For introducing more complicated signals such as video, the authors propose the use of a "see-through" project. This paper provides a description of a see-through project on a high-performance real-time DSP platform, discusses how such a project can lead to better follow-on learning using more advanced projects, and provides some initial results of classroom use.

Introduction

Digital signal processing (DSP) is now considered one of the "must know" topics by most employers of new electrical and computer engineering (ECE) graduates. While DSP may be taught in various ways, it is generally agreed by engineering educators that a solid understanding of many fundamental DSP topics is more fully realized when students are required to implement selected DSP algorithms in real-time (typically in C) [1]. While non-real-time (i.e., off-line) algorithm implementations with tools such as MATLAB or LabVIEW are easier to include in courses, and require more modest hardware and software, experience has shown there is significant benefit for students to including real-time DSP in the curriculum.

The best approach seems to be one that uses both types of exercises: non-real-time and real-time. With regard to non-real-time exercises, it's clear that interactive learning, exercises, and demonstrations to students using off-line methods are very useful for helping them to build an initial mental model [2–6]. However, taking the next step by requiring students to make the transition to real-time DSP implementations has been shown to cement a more complete understanding of DSP topics [7].

Since the late 1990's, the authors of this paper have reported on proven DSP teaching methodologies, hardware and software solutions, and various DSP tools that have helped motivate both students and faculty to implement real-time DSP-based systems, and thereby improve education in signal processing and related topics [8–17]. This support to educators teaching DSP includes a textbook (now in its second edition) and a web site that specifically helps both professors and students (along with working engineers) master a variety of real-time DSP concepts [18, 19].

When using real-time DSP in courses, we have observed that there can be an initial stumbling

block, which can greatly impede student progress. Specifically, we have found that when students are first making the transition from the more "comfortable" world of off-line signal processing (typically using MATLAB) to the unfamiliar world of real-time DSP, they must quickly establish confidence in the hardware and software platform before significant learning can begin. Without such confidence, students quickly assume that any errors or incorrect results from their DSP efforts must be due to the platform. With such a mindset, students will almost never investigate further to uncover other possible reasons for the flawed outcome, mainly because they are not yet comfortable with the new platform. To establish such confidence in the platform, a highly simplified "stripped down" first exercise is used, that merely tests the ability of the platform to correctly acquire input data samples and provide unmodified samples as output. No signal processing algorithm is performed by the processor to modify the samples; this exercise is meant to be just "sample in, sample out."

Correct output thus confirms for the student that proper initialization and configuration of all the hardware has taken place, that a correct interrupt vector table is being used, that the appropriate interrupt service routines (ISRs) are executing, that proper operation of the ADC and DAC of the intended codec is taking place, that correct execution of the compile-link-load software development chain was performed, and even verifies the correct connection of all the necessary cables and wires. A problem with any of these items would cause this first exercise to fail, at which point the instructor can guide the student toward a resolution of the problem. Once the first exercise is successfully completed, students are considerably more likely to take an appropriately critical and investigative approach to any errors they may encounter in the more sophisticated exercises that follow. With such confidence in the platform established, real learning can proceed. Some professors may consider such a "do nothing" program to be a trivial exercise, but we have been convinced by our own and our colleagues' experiences that skipping such a step significantly impedes learning.

The Talk-Through Real-Time DSP Exercise

When dealing with signals in the audio range, this first exercise has been called *talk-through*. In talk-through, an analog audio signal is acquired by codec's ADC channels at the desired sample frequency, and (assuming correct operation) the same audio signal is output from the codec's DAC channels (within the limitations imposed by factors such as quantization error, of course). Very simple modifications to the talk-through exercise can introduce basic concepts such as aliasing, quantization, left+right versus left–right channel combinations, and so forth. Despite the fact that the audio signal used is often not speech, and is most often some kind of music, the generic name "talk-through" has stuck.

Audio signals provide a rich source of inputs for a wide variety of real-time DSP exercises. When students hear the result of given algorithm, they tend to remember more than if they just looked at "before and after" plots of the signal. There may come a time, however, when the professor wants to introduce the students to a more complicated signal than audio, perhaps one at a higher frequency and wider bandwidth. One such signal that is readily available using low-cost equipment and one that also seems to excite students even more than audio is a video signal. When making the transition to video signals, the authors propose the use of a logical extrapolation of talk-through that we call a *see-through* project.

The See-Through Real-Time DSP Exercise

The authors' current preference for high-performance real-time DSP hardware to be used in the classroom is the relatively new, and surprisingly inexpensive, OMAP-L138 Low Cost Development Kit (LCDK) by Texas Instruments [20, 21]. One of the many advantages of this real-time DSP platform is the plethora of I/O choices. Most germane to this discussion about "see-through" is the video input and video output capability of the board, as shown in Fig. 1.

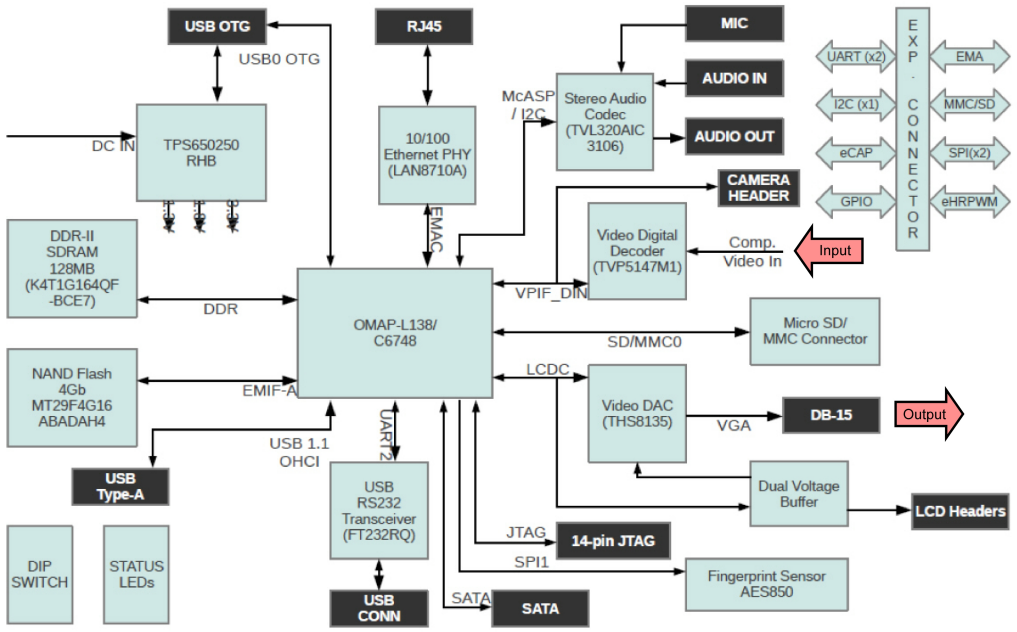

Figure 1: Block diagram of the LCDK from Texas Instruments. Note the video input and output capability indicated by the red arrows on the right side of the figure.

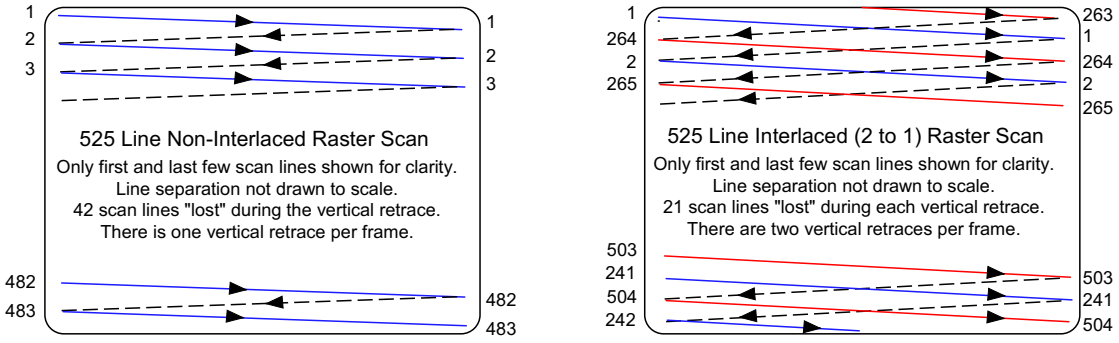

Figure 2: A comparison of progressive scan (left) and interlaced scan (right) video display.

The NTSC Video Standard

The video input of the LCDK is intended for a standard definition television-quality composite video signal such as NTSC, PAL or SECAM. NTSC, for example, is an analog baseband signal with a nominal 6 MHz bandwidth [22, 23]. Note that historically, NTSC has also been called RS-170A and EIA-170A, where the "A" suffix denotes a color-capable signal (RS-170 is monochrome only). The video display format defined by NTSC is interlaced, with two fields (sometimes called the odd and even fields) of every other scan line required to create one complete frame. The rate defined by NTSC is approximately 60 fields per second and thus 30 frames per second.* Each NTSC field consists of 262.5 scan lines; two fields per frame results in 525 scan lines per frame, of which a maximum of only 483 lines can be visible on the display screen due to factors such as synchronizations and vertical blanking/retrace. A comparison of progressive scan versus interlaced scan video display format is shown in Fig. 2. The "scan" aspect of progressive and interlaced display comes from the assumption of a *raster scan*, which is the "line-by-line" method used by cathode ray

*More precisely, NTSC is 59.94 fields/sec and 29.97 frames/sec.

tube (CRT) displays to present video information. Newer displays such as LCD screens do not use raster scan per se, but by necessity use preprocessing electronics to be compatible with that type of signal.

The original purpose of interlaced video was to conserve bandwidth for a signal intended for over-the-air broadcast. However, interlaced video standards such as NTSC became so widespread, and compatible equipment became so readily available and inexpensive, that such standards are often used even if the signal is never intended for broadcast. Interestingly, digital video cameras (such as those using CCD or CMOS focal plan array sensors) do not capture an image as a raster scan (neither progressive nor interlaced), but for compatibility reasons often covert the image to a signal format such as NTSC. Thus while modern cameras and displays don't "need" to employ either type of raster scan format, historical compatibility reasons (and affordability) keep these types of "older" signals of current interest.

To add color information in a compatible way to the original monochrome RS-170 signal, NTSC has subcarriers defined for luma, chroma, and audio. Luma is basically brightness, and is equivalent to the monochrome part of the video signal. Chroma has two components, modulated in quadrature, and encodes the necessary color information. A basic spectral diagram of the NTSC signal is shown in Fig. 3. From the preceding discussion and Fig. 3, one can readily see that the NTSC video signal is a significant step up in complexity from a typical 20 kHz bandwidth audio signal sampled at 48 kHz, and the pitfalls for the student are that much greater.

Video Input and Output on the LCDK

On the LCDK board, the analog video input is digitized, decoded, and formatted by the TVP5147M1 digital video decoder. This chip includes a high performance ADC stage (sampling at up to 30 Msps) and the necessary circuitry for extracting and providing at the decoder chip output the separated luma and chroma information as individual 10-bit data streams (using the intermediate steps of YUV $\rightarrow$ YCbCr $\rightarrow$ luma (Y) and chroma (C) format). The chroma data stream uses 5 bits/sample each for Cb and Cr, in an alternating interleaved pattern of [CbCrCbCr$\cdots$]; the luma data stream uses the full 10 bits/sample for Y. No audio demodulation is performed by the TVP5147M1 decoder chip. While the LCDK implementation of this decoder chip only takes advantage of a small subset of the available modes, the user must still fully configure the chip via the on-board I^2C bus to set the appropriate mode of operation.

The analog video output from the LCDK is formatted and produced by the THS8135 video DAC; this chip accepts digital input in either YCbCr or RGB format, and supplies a standard VGA (incorporating analog RGB) signal as output. On the LCDK, this output connects to a DB-15 connector.

At first glance, a "see-through" exercise for real-time DSP using an LCDK seems straightforward. Simply connect an analog video camera that outputs an NTSC signal to the LCDK's video input, connect a VGA-compatible monitor to the DB-15 VGA output, write a bare-bones "do nothing" real-time DSP program to bring in the input and send out the output... but in reality it's not nearly that easy. The initialization of the LCDK must include extensive configuration code to set the proper I/O, interrupt enable, interrupt vector table, initial setup of the TVP5147M1 digital video decoder, DMA channels, and so forth, which is not trivial but only needs to be done at start-up. However, the actual real-time frame-to-frame operation isn't straightforward either. While the TVP5147M1 digital video decoder outputs YCbCr, and one of the input modes of the THS8135 video DAC accepts YCbCr, the LCDK board is constructed in such a way that the configuration pins on the THS8135 chip are hardwired to set the input mode to RGB only. Thus at a minimum, a conversion from YCbCr to RGB must be included as part of the real-time see-through code.

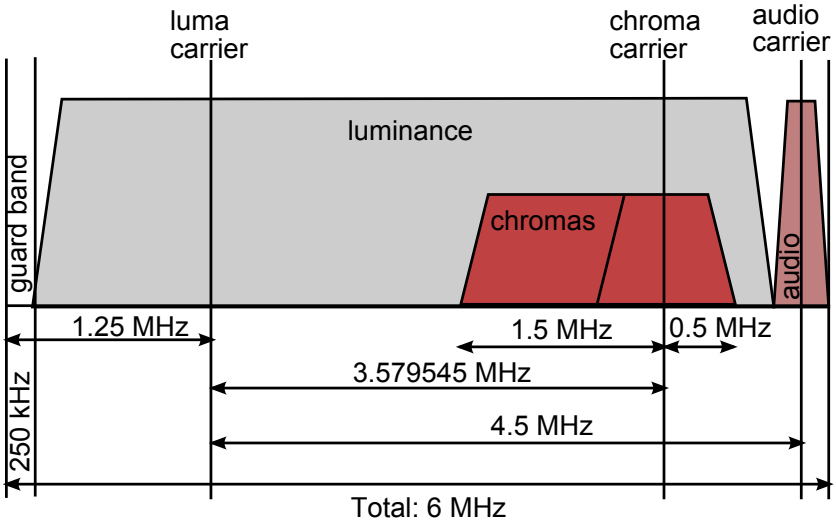

Figure 3: Spectrum of a typical NTSC video signal.

Conversion from YCbCr to RGB is a simple linear relationship. Defined most basically, independent of the number of bits per sample, the conversion is

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.3441 & -0.7141 \\ 1.0 & 1.772 & 0.00015 \end{bmatrix} \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix}$$

where it is assumed that the values are normalized such that the range of the RGB values and the luma Y values is $[0, 1]$, the range of the chroma Cb and Cr values is $[-0.5, +0.5]$, and any head-room or toe-room has been removed by rescaling to full-range as needed. Variations incorporating head-room and toe-room compress the range of allowable quantization levels to compensate for noise and channel distortions anticipated in a signal to be broadcast over-the-air. Note that there are minor variations on the conversion coefficients shown above depending upon what version ITU standard is appropriate. The normalized values shown here are correct for ITU-R BT.601, which are the same ones used for the JPEG and MPEG standards.

The YCbCr $\rightarrow$ RGB conversion method shown above could be directly implemented in the real-time DSP code. However, since some processing had to be performed inside the see-through ISR anyway, we took it as an opportunity to explore certain aspects of the OpenCV library and the TMS320C6748 SYS/BIOS Software Development Kit (SDK) provided by Texas Instruments (TI); see reference 24. In particular, we used basic parts of the "Facedetect" and "VPIF Loopback" example projects from the SDK.

Implementation Demo

The TMS320C6748 SYS/BIOS SDK incorporates various TI utility functions and the OpenCV library. OpenCV is an open source computer vision library originally created by the Intel Corporation in 1999, then transferred to the non-profit OpenCV.org group in 2012 [25]. While a see-through exercise, by definition, should perform minimal processing, we wished to retain the ability to call various OpenCV functions and utility routines for more complicated follow-on projects that would be based on the see-through project.

The CCS example projects as supplied with the SDK required the full 2 GB SDK, but stripping this down to only the necessary device drivers and library files dropped this to just under 20 MB (OpenCV itself, once precompiled by the user, is 18 MB of that total). The original example projects also used the SYS/BIOS real-time operating system (RTOS) from TI, which we have found is *not* a good pedagogical choice for first exposure to students. The SYS/BIOS RTOS adds considerable complexity, overhead, and executable file

size to support functionality that we don't want or need for projects which are intended for "beginning" real-time DSP students. Therefore, we further stripped down and borrowed from the SDK example projects, removing the dependence on the SYS/BIOS RTOS, and created a real-time project in the manner recommended in reference 18.

In the see-through project, `main.c` initializes the hardware and the video camera, then calls the function `Process_Video` in an endless loop. The `Process_Video` function waits for a frame to be captured before proceeding. At the end of each incoming frame from the camera, an interrupt is generated.

There are two ISRs, `VPIFIsr` and `LCDIsr`. The `VPIFIsr` ISR brings in the video data and stores it using separate frame buffers for the luma and chroma; a double buffer (i.e., ping-pong) method is used for each. At this point, the `Process_Video` function resumes and performs the YCbCr → RGB conversion via a call to the `cbcr422sp_to_rgb565_c` TI utility function. Note that this conversion from YCbCr to RGB does not occur inside an ISR. The `LCDIsr` ISR continuously points the appropriate frame buffer (now containing RGB values) to the output raster buffer for DMA transfer and display on the monitor; this ISR also has a placeholder where some real-time image processing algorithm can be executed if desired. Since most standard image processing algorithms assume RGB values as the starting point (not YCbCr), this is the best location for such a placeholder, since the conversion to RGB has already occurred. As a simple test, a the `cvRectangle` OpenCV function is called here to place a small blue rectangle on the screen over the video data.

It should be noted that, for improved speed, the frame buffers for this project are created in the high-speed DDR RAM of the LCDK's L1 cache rather than in the much slower external DDR RAM memory space.

The real-time see-through project provided an excellent demonstration of bringing in a video signal from a camera and sending it out for display on a monitor, as shown in Fig. 4. Note the blue rectangle displayed at the lower left of the image, on top of the real-time video image. This "simple" real-time demo is highly motivating for students, and provides many opportunities to segue into various basic concepts. For example, sampling and aliasing can be discussed in terms of measuring the frame rate of the system by imaging a variable-speed rotating disk with a high-contrast marker on it. As the disk speed is steadily increased, it appears to start slowing down when the rotational rate exceeds half the frame rate (i.e., $F_s/2$) and aliasing occurs. The disk appears to be stationary when the rotational rate equals the frame rate (i.e., F_s). This is also a good time to show students that, while the fully optimized "Release Build" of the project can run at the full 29.97 frames/sec rate of NTSC, the non-optimized "Debug Build" can only run at approximately 6.5 frames/sec.

A see-through program can be effectively used as a confidence-building exercise for students, as discussed previously, and can also be instrumental in the creation of particularly motivating demonstrations.

Classroom Results

Having available a video signal and a see-through real-time DSP program proved to be fortuitous. During the Fall 2013 semester, we spent additional time discussing sampling during our "Digital Signal Processing" class at Boise State University. As always, these discussions included traditional MATLAB simulations augmented by real-time demonstrations using a function generator and a digital sampling oscilloscope (DSO). Aliasing and bandpass sampling were demonstrated in both the time and frequency domains.

Using the see-through program, visual aliasing and rotation rate measurement were also demonstrated using a motor-driven, rotating disk and a strobe light. This demonstration, received quite en-

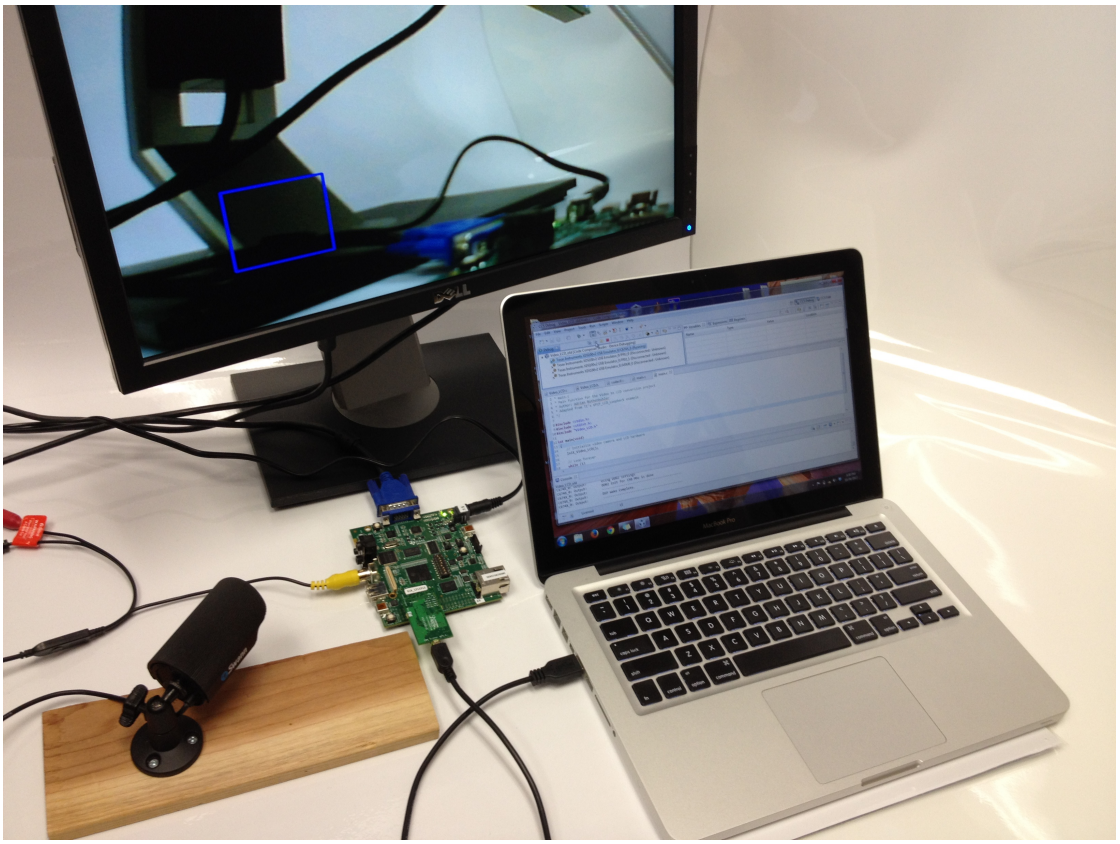

Figure 4: A demonstration of the see-through real-time DSP exercise.

thusiastically by the students, also reinforced the parallel concepts of samples per rotation for mechanical systems versus samples per period for an electrical signal.

Additionally, a collaborative, multidisciplinary educational opportunity, combining the seemingly disparate fields of Music Education and Electrical and Computer Engineering (ECE), took place. The ECE faculty member brought his entire Digital Signal Processing (DSP) class with him to the Music Education event. This combined meeting of the two groups of students with phenomenally different backgrounds led to a unique learning opportunity for all those present. The ECE students gained an appreciation for the talent level of the Music Education students and the challenges of sampling live audio signals. This included a review of the sampling theorem and a discussion of multirate spectral analysis. The multirate portions of the audio signal analysis dealt with decimation of spectrally compact signals. Aliasing was demonstrated through both improper sampling and proper sampling with improper decimation.

Finally, a practical sample rate problem was presented as an in-class design challenge. Specifically, even though we offer a separate "Digital Image Processing" course, the challenge was to determine the frame rate of a real-time image processing system. This again made use of the see-through program.

In addition to facilitating sampling discussions, the see-through program can provide student motivation for other DSP topics. For example, after presenting the basic concept of an analog video signal such as the NTSC signal shown in Fig. 3, we can discuss the details of RGB versus YCbCr and how they represent color images with two very different encoding strategies. Many other possibilities readily come to mind.

At the end of the semester, student opinions of

various topics in the course were measured using a five-point Likert-scale survey. The survey item most pertinent to this paper was,

"The practical application of signal processing (specifically, the use of real-time imaging hardware) helped me better understand the underlying concepts." The allowed responses were,

1. strongly disagree
2. disagree
3. undecided
4. agree
5. strongly agree.

The average score of the 20 survey respondents to this statement was 4.30, with a standard deviation of 0.733. Of the 20, no students circled the "2 - disagree" response or the "1 - strongly disagree" response. See Figure 5 for the complete results. Note that this small sample size is inadequate to draw any statistically firm conclusions, but we still are confident of the general results.

We have used a "talk-through" program for many years in our DSP courses and workshops, for the reasons mentioned in this paper. However, we have recently noticed that many students today, for whom video availability via their mobile phone or via web sites such as YouTube is taken for granted, seem to get more excited with "see-through" than with "talk-through." Since this higher level of excitement leads to greater interest and better student engagement, the additional complexity of implementing a "talk-through" program may very well be worth it for many educators.

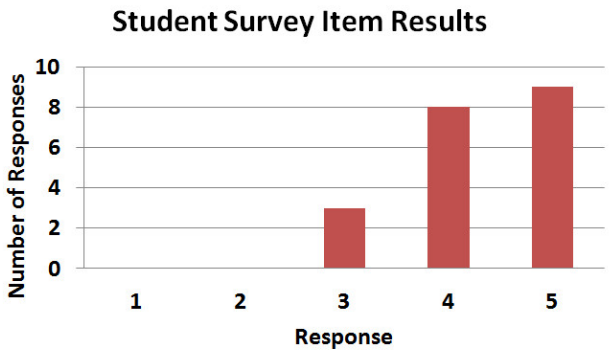

Figure 5: Raw data for student responses to the item, "The practical application of signal processing (specifically, the use of real-time imaging hardware) helped me better understand the underlying concepts."

Conclusions

Building student confidence in the real-time platform early on by using a bare-bones first project, such as talk-through or see-through, is a valuable pedagogical approach. Skipping this step can significantly reduce student motivation to pursue the real cause of incorrect results from a real-time DSP exercise, because many students are otherwise quick to "blame" the platform. These very basic programs are also surprisingly helpful for providing motivational demonstrations.

When moving up in signal complexity from audio (i.e., talk-through) to video (i.e., see-through), a host of additional considerations and complications must be addressed. Not only is the video signal itself more complicated, but the configuration and use of the input and output chips specific to video are more challenging to use than a typical audio codec. An additional requirement when using the LCDK for video see-through is the need for conversion from YCbCr to RGB.

We have built such a see-through project and successfully run it at the full frame rate of NTSC video using the LCDK. In the future, we plan to more completely remove the reliance on pieces of the original example projects supplied by TI with the TMS320C6748 SYS/BIOS Software Development Kit. This will provide more opportunities to more clearly show students how to make use of the video capabilities of the LCDK, which we hope will encourage them to attempt more advanced student projects that would incorporate video.

Any faculty members who teach DSP are strongly encouraged to incorporate demonstrations and hands-on experience with real-time hardware for their students, and to include simple projects such as talk-through and see-through as confidence-building exercises. To support our colleagues in this endeavor, we have made various resources available on the web [19, 26].

Acknowledgment

Adrian Rothenbuhler, an electrical engineer at Hewlett-Packard, in Boise, ID, was instrumental in developing the initial "see-through" implementation as part of his graduate work at Boise State University.

References

1. C. H. G. Wright, T. B. Welch, D. M. Etter, and M. G. Morrow, "Teaching DSP: Bridging the gap from theory to real-time hardware," *ASEE Comput. Educ. J.*, pp. 14–26, July–September 2003.

2. C. S. Burrus, "Teaching filter design using MATLAB," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 20–30, Apr. 1993.

3. R. F. Kubichek, "Using MATLAB in a speech and signal processing class," in *Proceedings of the 1994 ASEE Annual Conference*, pp. 1207–1210, June 1994.

4. R. G. Jacquot, J. C. Hamann, J. W. Pierre, and R. F. Kubichek, "Teaching digital filter design using symbolic and numeric features of MATLAB," *ASEE Comput. Educ. J.*, pp. 8–11, January–March 1997.

5. J. H. McClellan, C. S. Burrus, A. V. Oppenheim, T. W. Parks, R. W. Schafer, and S. W. Schuessler, *Computer-Based Exercises for Signal Processing Using* MATLAB 5. MATLAB Curriculum Series, Upper Saddle River, NJ (USA): Prentice Hall, 1998.

6. J. W. Pierre, R. F. Kubichek, and J. C. Hamann, "Reinforcing the understanding of signal processing concepts using audio exercises," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3577–3580, Mar. 1999.

7. T. B. Welch, M. G. Morrow, and C. H. G. Wright, "Teaching practical hands-on DSP with MATLAB and the C31 DSK," *ASEE Comput. Educ. J.*, pp. 13–20, April–June 2001.

8. C. H. G. Wright and T. B. Welch, "Teaching DSP concepts using MATLAB and the TMS320C31 DSK," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3573–3576, Mar. 1999.

9. M. G. Morrow, T. B. Welch, C. H. G. Wright, and G. W. P. York, "Demonstration platform for real-time beamforming," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, pp. 2693–2696, May 2001.

10. C. H. G. Wright, T. B. Welch, D. M. Etter, and M. G. Morrow, "Teaching hardware-based DSP: Theory to practice," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 4148–4151, May 2002.

11. T. B. Welch, R. W. Ives, M. G. Morrow, and C. H. G. Wright, "Using DSP hardware to teach modem design and analysis techniques," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. III, pp. 769–772, Apr. 2003.

12. T. B. Welch, C. H. G. Wright, and M. G. Morrow, "Caller ID: An opportunity to teach DSP-based demodulation," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. V, pp. 569–572, Mar. 2005. Paper 2887.

13. T. B. Welch, C. H. G. Wright, and M. G. Morrow, "Teaching rate conversion using hardware-based DSP," in *Proceedings of the*

IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. III, pp. 717–720, Apr. 2007.

14. C. H. G. Wright, M. G. Morrow, M. C. Allie, and T. B. Welch, "Enhancing engineering education and outreach using real-time DSP," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. III, Apr. 2008.

15. T. B. Welch, C. H. G. Wright, and M. G. Morrow, "Software defined radio: inexpensive hardware and software tools," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2934–2937, Mar. 2010.

16. M. G. Morrow, C. H. G. Wright, and T. B. Welch, "winDSK8: A user interface for the OMAP-L138 DSP board," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2884–2887, May 2011.

17. M. G. Morrow, C. H. G. Wright, and T. B. Welch, "Real-time DSP for adaptive filters: A teaching opportunity," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2013.

18. T. B. Welch, C. H. G. Wright, and M. G. Morrow, *Real-Time Digital Signal Processing: From MATLAB to C with C6x DSPs*. Boca Raton, FL (USA): CRC Press, 2nd ed., 2012.

19. "RT-DSP website." `http://www.rt-dsp.com`.

20. M. G. Morrow, C. H. G. Wright, and T. B. Welch, "An inexpensive approach for teaching adaptive filters using real-time DSP on a new hardware platform," *ASEE Comput. Educ. J.*, pp. 72–78, October–December 2013.

21. Texas Instruments, "L138/C6748 Development Kit (LCDK)," 2013. `http://processors.wiki.ti.com/index.php/LCDK_User_Guide`.

22. K. B. Benson and J. Whitaker, *Television Engineering Handbook*. New York: McGraw-Hill, revised ed., 1992.

23. Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing and Communications*. Prentice-Hall, 2002.

24. "Texas Instruments SYS/BIOS real-time kernel," 2013. `http://www.ti.com/tool/sysbios`.

25. "OpenCV website," 2013. `http://opencv.org/`.

26. Educational DSP (eDSP), L.L.C., "DSP resources for TI DSKs." `http://www.educationaldsp.com/`.

Biographical Information

Cameron H. G. Wright, Ph.D, P.E., is with the Department of Electrical and Computer Engineering at the University of Wyoming, Laramie, WY. His research interests include signal and image processing, real-time embedded computer systems, biomedical instrumentation, and engineering education. He is a member of ASEE, IEEE, SPIE, BMES, NSPE, Tau Beta Pi, and Eta Kappa Nu. E-mail: `c.h.g.wright@ieee.org`

Thad B. Welch, Ph.D, P.E., is with the Department of Electrical and Computer Engineering at Boise State University, ID. His research interests include implementation of communication systems using DSP techniques, DSP education, multicarrier communication systems analysis, and RF signal propagation. He is a member of ASEE, IEEE, Tau Beta Pi, and Eta Kappa Nu. E-mail: `t.b.welch@ieee.org`

Michael G. Morrow, P.E., is with the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison, WI. His research interests include real-time digital systems, embedded system design, software engineering, curriculum design, and educational assessment techniques. He is a member of ASEE and IEEE. E-mail: `morrow@ieee.org`