Int. J. Engng Ed. Vol. 21, No. 4, pp. 587–595, 2005
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2005 TEMPUS Publications.

Transition from Simulink to MATLAB in Real-Time Digital Signal Processing Education*

WOON S. GAN
School of Electrical & Electronic Engineering, Nanyang Technological University, Singapore 6397. E-mail: ewgan@ntu.edu.sg

SEN M. KUO
Dept. of Electrical Engineering, Northern Illinois University, DeKalb, IL 60115 98, USA

In this paper, we propose a two-level approach for teaching digital signal processing (DSP) from basic concepts to the level of developing DSP software for real-time implementations on programmable DSP processors. In our approach, MATLAB and Simulink make the transition from theory to application easy and enjoyable. We use many interesting DSP demonstrations and examples for students to 'see' the effects of signal processing in Simulink; and then ask students to 'do' hands-on exercises in Simulink and MATLAB. The emphasis of 'seeing' and 'doing' can capture the students' attentions, cultivate their interests, and motivate their curiosities. This effective learning approach also allays fear of DSP that has been previously tagged as too theoretical and mathematically intensive.

INTRODUCTION

REAL-TIME DSP is now becoming an important engineering education subject at most colleges and universities. Because of the fast-growing demands in consumer, communication and high-end multimedia products that utilize DSP technologies, there is a strong motivation to teach more students and engineers real-time DSP algorithms, implementations, and applications to meet the increased challenges from industry. This work is based on the efforts in conducting a real-time DSP design course at Nanyang Technological University in Singapore and Northern Illinois University in the USA. Continual feedback from students and faculty members help us to fine-tune the details of course content, demonstrations, and exercises presented in this course. In addition, we have recently compiled these works into the textbook titled *Digital Signal Processors: Architectures, Implementations and Applications* [1] and other related publications [2–5].

In this paper, we highlight this effective approach and showcase an example on how to handle this two-level teaching approach, and how to effectively use both MATLAB [6] and Simulink [7]. Several important MATLAB toolboxes, tools, and Simulink blocksets include Signal Processing Toolbox [8], Filter Design and Analysis Tool (FDATool), Signal Processing Tool (SPTool), Filter Design Toolbox [9] (split into Filter Design Toolbox [20] and Fixed-Point Toolbox [21] in the latest MATLAB v7), Fixed-Point Blockset [10] (called Simulink Fixed Point [22] in the latest Simulink v6.1), and Signal Processing Blockset [11] are used in this course. All M-files, Simulink files, and speech wave files used in this paper are available on the author's web site: http://eeeweba.ntu.edu.sg/DSPLab/geg/web.htm. In the second section, Simulink and its blocksets are used to demonstrate DSP concepts and its applications. In this paper, we use a simple finite-impulse response (FIR) filter as an example for graphic equalizer applications. Then we show how to write MATLAB programs for processing signals stored in data files. We use double-precision floating-point programs for verifying DSP algorithms first, and then convert them to fixeded-point programs using MATLAB built-in and user-written functions. This 'Simulink-first, MATLAB-second' process provides a systematic approach of understanding the DSP principles and real-time implementations.

USING SIMULINK IN UNDERSTANDING DSP CONCEPTS

Simulink provides an exploratory and verification tool for both floating-point and fixed-point DSP systems and applications. A simple example shown in Fig. 1 highlights the implementation of a 3-band graphic equalizer, which consists of three FIR filters connected in parallel. The first filter is a lowpass filter with 1 kHz cutoff frequency; the second filter is a bandpass filter with cutoff

* Accepted 2 April 2005.

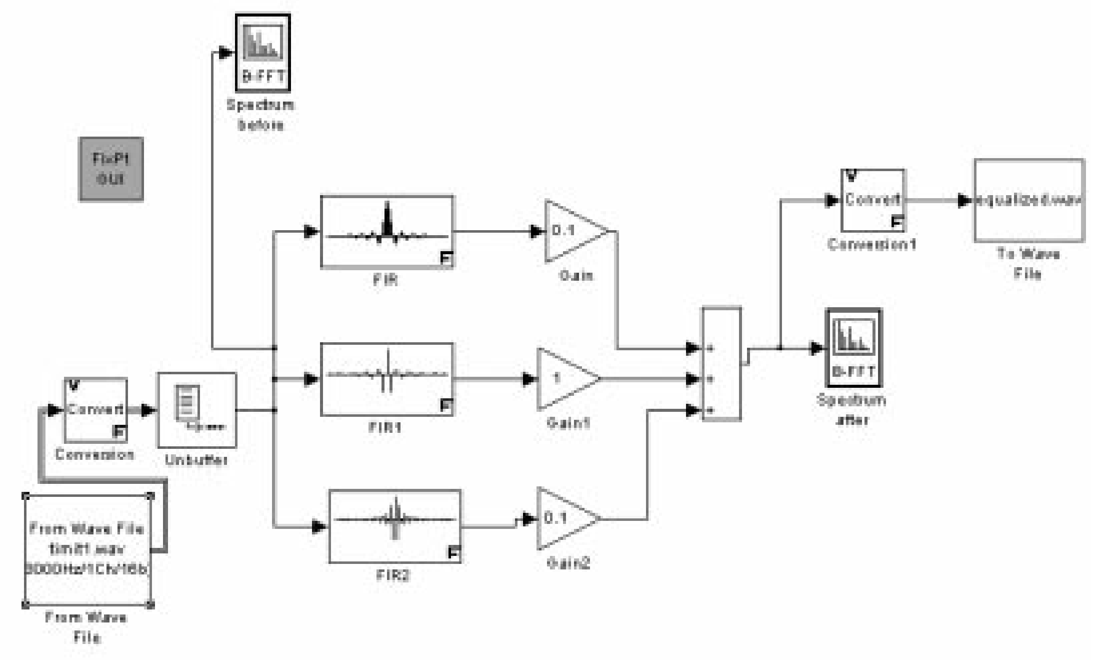


Fig. 1. A 3-band graphic equalizer using Simulink blocksets with double-precision floating-point representations.

frequencies at 1.2 kHz and 2.8 kHz; and the third filter is a highpass filter of cutoff frequency at 3 kHz. The sampling rate is 8 kHz and the transition width is 200 Hz. The passband and stopband ripples are set to 0.5dB and –50dB, respectively. In addition, the equiripple design method is used for designing the FIR filters. The outputs from these filters are weighted by a slider gain. Audio signals are processed by these parallel filters, with different gain settings to control the signal strength for bass, middle, and treble. The audible effects can be evaluated directly by subjective listening using the sound input/output (I/O) blocks, and by objective viewing using the spectrum scopes.

After verifying the DSP algorithms using double-precision floating-point simulations, we can replace the floating-point FIR filters by fixed-point filters using the Fixed-Point Blockset (or Simulink Fixed Point) as shown in Fig. 2. Note that the audio signal must be converted to fixed-point data format before passing to the fixed-point filters. The fixed-point data format used in this case is the Q.15 format, which is commonly used in most 16-bit fixed-point DSP processors. The Q.15

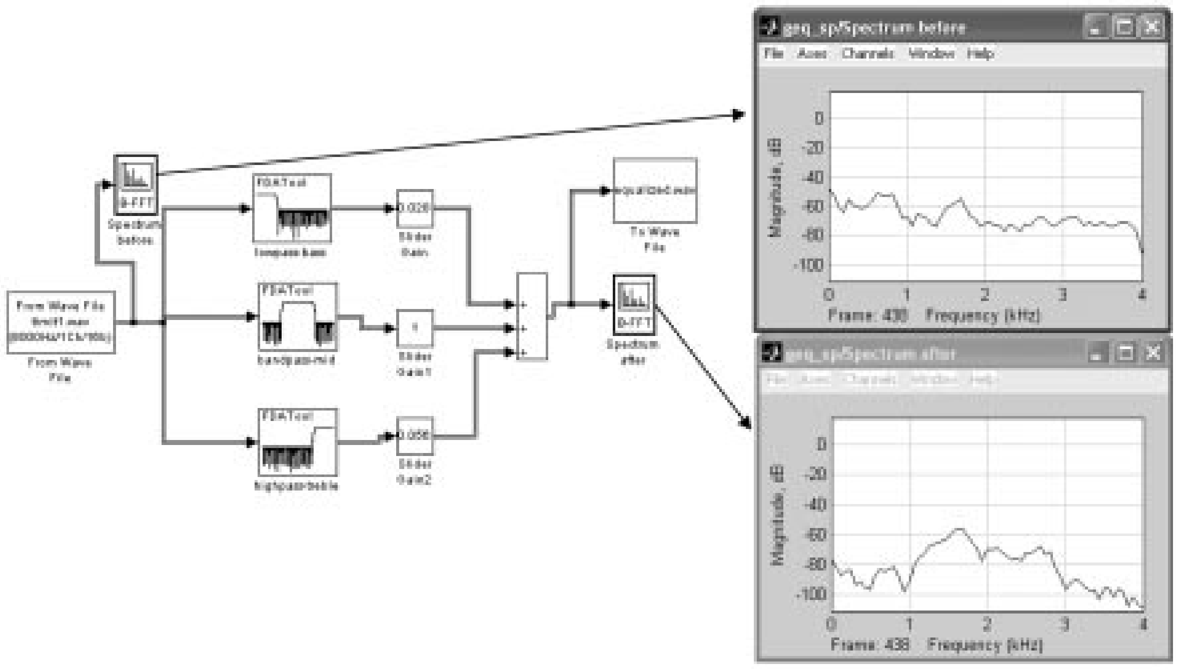
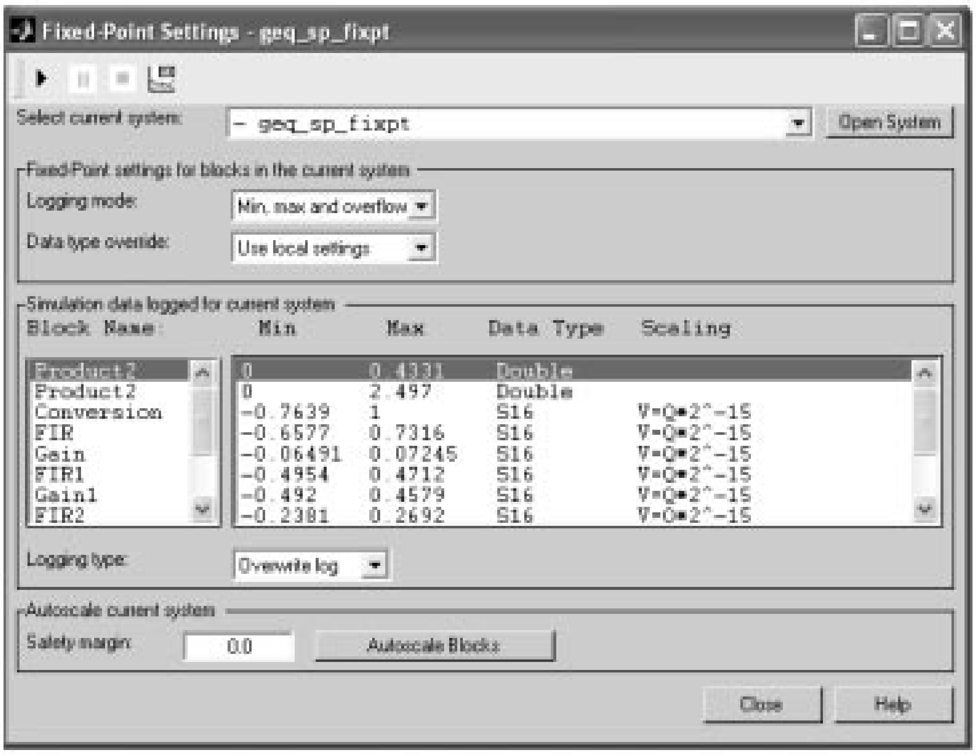


Fig. 2. A Q.15 graphic equalizer using Simulink with fixed-point blocksets.



Select current system: - geq_sp_fixpt

Fixed-Point settings for blocks in the current system
Logging mode: Min, max and overflow
Data type override: Use local settings

Simulation data logged for current system

Block Name:	Min	Max	Data Type	Scaling
Product2	0	0.4331	Double	
Product2	0	2.497	Double	
Conversion	-0.7639	1	S16	V=Q*2^-15
FIR	-0.6577	0.7316	S16	V=Q*2^-15
Gain	-0.06491	0.07245	S16	V=Q*2^-15
FIR1	-0.4954	0.4712	S16	V=Q*2^-15
Gain1	-0.492	0.4579	S16	V=Q*2^-15
FIR2	-0.2381	0.2692	S16	V=Q*2^-15

Logging type: Overwrite log

Autoscale current system
Safety margin: 0.0

Fig. 3. A display of FixPt GUI obtained by running the Q.15 graphic equalizer.

format uses a sign bit and 15 fractional bits to represent fractional numbers in the range of –1 to $1–2^{-15}$ in 2s complement format. The Fixed-Point Blockset (or Simulink Fixed Point) allows students to easily compare the results of the fixed-point Q.15 implementations with double-precision floating-point implementations.

A FixPt GUI block can be used to identify overflow problems, data types, and scaling factors at each block in the system shown in Fig. 2. Signal range in the simulations can be reported, and the scaling factors of each block can be fine tuned. An example of this information displayed in the FixPt GUI block is shown in Fig. 3.

The Simulink allows students to 'see' the changes on-the-fly for new settings, structures, and algorithms used in the systems. Several modifications are suggested for students to learn more hands-on experiments:

1. Design FIR filters with the same specifications using a window or least-square method. Students can compare the filter orders required from different design methods and identify the method that results in the lowest order. A suitable window must be chosen in the window method to meet the given specifications.
2. Use the equiripple method to tighten the filter specifications by reducing the passband and stopband ripples. In addition, a further reduction of the transition width can be achieved by increasing the required filter order. This trade-off is critical since it dictates both memory and processing resources needed in real-time DSP processing.
3. Replace all FIR filters with IIR filters that meet the same frequency specifications. Compare the computational requirements between the use of FIR and IIR filters. It is important to ensure the stability of IIR filters by guaranteeing that all poles are inside the unit circle. A cascade of second-order IIR filters must be used to reduce the sensitivity of finite-wordlength effects for the fixed-point implementation shown in Fig. 3.

By doing these exercises, students can grasp the concepts of designing FIR and IIR filters, and use them in interesting speech/audio applications. A brief introduction on the theory of FIR and IIR filters' structures and design techniques is sufficient for asking students to do these hands-on exercises. Other topics such as spectrum analyzer, audio effects, and noise reduction applications can also be easily performed using Simulink. Interested readers can refer to the reference [1] for more examples and exercises.

USING MATLAB IN TEACHING REAL-TIME DSP

There are many advantages in programming DSP algorithms in MATLAB instead of using C programs. These include (1) ease of code profiling (benchmarking), (2) able to use a powerful set of fixed-point functions from the Filter Design Toolbox, (3) quick visualization and comparison of results between fixed- and floating-point implementations, and (4) seamless link between MATLAB and Simulink. In addition, this first-level of fixed-point simulation in MATLAB helps to speed up subsequent code development in C and assembly languages for real-time DSP implementations.

The first task for students to learn MATLAB programming is to write an M-file for implementing the 3-band graphic equalizer (presented in Section 2) based on the floating-point arithmetic. Profiling of the code efficiency can be performed at this stage to identify the time-critical sections of code. Subsequently, students are asked to convert the floating-point numbers into 16-bit integer numbers. This important step is required for converting the floating-point C to fixed-point C programs. To check for any discrepancy, the fixed-point results from MATLAB are compared with that obtained from the Simulink with fixed-point blocksets.

Writing floating-point M-files

The 3-band graphic equalizer shown in Fig. 1 can be converted to a MATLAB script (M-file). Many existing signal processing functions from the Signal Processing Toolbox can be used directly, and results can be quickly evaluated to verify the correctness by comparing to the corresponding Simulink results obtained in the previous section. The same 16-bit signal samples and the double-precision filter coefficients derived from the Simulink given in Fig. 1 are first loaded into the MATLAB workspace. A MATLAB program is listed below which gives the MATLAB script of 3-band graphic equalizer using built-in signal processing functions:

```
clear all;
%read speech signal . . .
[y,fs,b]=wavread('Timit1.wav');
%load filters' coefficients . . .
load geq_coeff
%Gain setting of GEQ . . .
gain_lp = 0.1;
gain_bp = 1;
gain_hp = 0.1;
%perform filtering of LPF
out_lp = filter(Num_lp,1,y).
  *gain_lp;
%perform filtering of BPF
out_bp = filter(Num_bp,1,y).
  *gain_bp;
%perform filtering of HPF
out_hp = filter(Num_hp,1,y).
  *gain_hp;
%sum outputs . . .
out_total = out_lp+out_bp+out_hp;
wavwrite(out_total,fs,b,'processed.
  wav');
% plot results
figure(1), psd(y); title('Spectrum
  plot of input');
figure(2),psd(out_total);
  title('Spectrum plot of output');
```

The next step of programming exercises is to convert the M-file listed above to a C-like program without using MATLAB built-in functions. For this purpose, the MATLAB function filter is replaced by two user-written functions, `datamov.m` and `filt.m`, as listed below, giving user-written MATLAB functions for FIR filtering:

```
function x = datamov(x,ntap,input)
for k = ntap:-1:2
x(k)=x(k-1);
end
x(1)=input;
function out = filt(x,w,ntap,gain)
acc = 0;
for i = 1:1:ntap
acc = w(i)*x(i)+acc;
end
out=acc*gain;
```

The datamov function listed above refreshes the signal buffer (tapped-delay-line) when a new input sample has arrived at the FIR filter. The x, ntap, and input arguments denote the delay-line vector, number of coefficients in the filter, and the incoming input sample, respectively. For a sample-by-sample processing, input is a scalar which represents the latest data sample. The filt function performs the multiply-add operations of the FIR filtering after the signal buffer has been updated. The input arguments w and gain represent the filter-coefficient vector and the equalizer gain that will be applied to the output of the filter. By writing these two functions, students have a better understanding on how digital FIR filters can be implemented in C for the actual DSP processors.

Block FIR filtering can also be written for a block of incoming data samples at every call of the block-filter function. A user-written M-file for the block FIR filtering is listed below as a user-written MATLAB functions for block FIR filtering;

```
function [out,x] =
  blkfilt(x,w,ntap,blk_size,
  binput,gain)
for n = 1:1:blk_size
for k = ntap:-1:2
x(k)=x(k-1);
end
x(1)=binput(n);
acc = 0;
for i = 1:1:ntap
acc = w(i)*x(i)+acc;
end
out(n)=acc*gain;
end
```

Note that the blkfilt.m function combines the tapped-delay-line update and the FIR filtering operations. The major difference between the `blkfilt.m` and the combination of `datamov.m` and `filt.m` is that the former takes in a block of input samples (`binput` of `size` `blk_size`), and performs FIR filtering over the whole block of data. The advantage of using block filtering is that

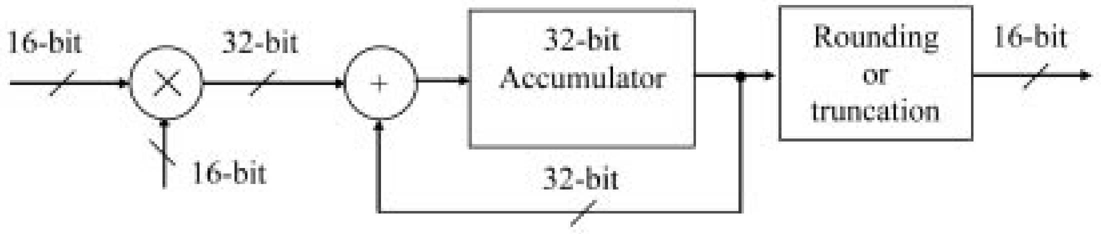


Fig. 4. Block diagram of the multiply-accumulate functional block in DSP processor.

a smaller function overhead is incurred when processing the same number of data samples. This advantage is also true if we combined `data-mov.m` and `filt.m` into a single function.

Profiling of the MATLAB code can be executed simply by clicking on **Start** (an icon at the bottom-left of Matlab command window) → **MATLAB** → **Profiler**, follow by typing the name of M-file to be run in the pop-up window. Compared to the sample-by-sample processing approach, the block filtering technique reduces the number of times the function is called, thus reducing execution time. For example, more than 6 times faster can be achieved when a block of 32 samples is used. However, careful pointer and memory management is required for the block filtering approach. In addition, the latency of obtaining output sample is longer when compared to the sample-by-sample processing.

It is noted that MATLAB code can be executed faster if the program is written in a vector form or using array operations. However, in order to write a program which has closer resemblance to the actual C code, we use 'for-loop' in DSP programming. In addition, function overloading is also available in MATLAB for implementing the same function that takes in different data types. This useful feature allows users to investigate the effects of using different floating- and fixed-point arithmetic. In the following section, fixed-point MATLAB code is written to illustrate the actual fixed-point arithmetic being performed on real fixed-point DSP processors.

Writing fixed-point M-file

In the 16-bit fixed-point FIR filtering, filter coefficients and data samples are multiplied and the products are accumulated to generate the filter output. It is important to note that multiplication of two 16-bit numbers results in a 32-bit product, thus requiring 32-bit memory to store the result. Therefore, a typical accumulator in 16-bit DSP processors must be at least 32 bits. The final 32-bit result is rounded (or truncated) for storing in 16-bit memory. Figure 4 shows the block diagram of a typical multiply-add functional blocks in 16-bit fixed-point DSP processors. In order to prevent arithmetic overflow, certain techniques are normally employed. For example, increase the number of bits for the accumulator. These additional bits are known as the guard bits. The second method is to saturate the overflowed output to its maximum or minimum value. However, the most effective technique is to scale the signals at different nodes of the system to guarantee that the amplitude of signal is less than one.

A Q.15 fixed-point graphic equalizer can be written in MATLAB as listed below for the 3-band graphic equalizer using Q.15 format:

```
%M-file for performing 16-bit Q.15
  fixed-point graphic EQ
clear all;close all;
%read speech signal . . .
[y,fs,b]=wavread('Timit1.wav');
y_int = round(32768*y./max(abs(y)));
%load filters' coefficients . . .
load geq_coeff
Num_lp_int = round(32768*Num_lp);
Num_bp_int = round(32768*Num_bp);
Num_hp_int = round(32768*Num_hp);
%Gain setting of GEQ . . .
gain_lp_int = 3277; %0.1;
gain_bp_int = 32767; %0.99;
gain_hp_int = 3277; %0.1;
%perform filtering of LPF
out_lp_int = round(filter
  (Num_lp_int,1,y_int)/32768);
out_lp_q15 = round((out_lp_int.
  *gain_lp_int)/32768);
%perform filtering of BPF
out_bp_int = round(filter
  (Num_bp_int,1,y_int)/32768);
out_bp_q15 = round((out_bp_int.
  *gain_bp_int)/32768);
%perform filtering of HPF
out_hp_int = round(filter
  (Num_hp_int,1,y_int)/32768);
out_hp_q15 = round((out_hp_int.
  *gain_hp_int)/32768);
%sum outputs . . .
out_total_q15 =
  out_lp_q15+out_bp_q15+out_hp_q15;
out_total_q15 = out_total_q15/32768;
%Check for overflow . . .and use
  saturation mode . . .
max_pos = 1-2^-15;
max_min = -1;
ind_pos = find(out_total_q15 >
  max_pos);
out_total_q15(ind_pos) = max_pos;
ind_min = find(out_total_q15 < -1);
out_total_q15(ind_min) = max_min;
wavwrite(out_total_q15
  ,fs,b,'processed.wav');
```

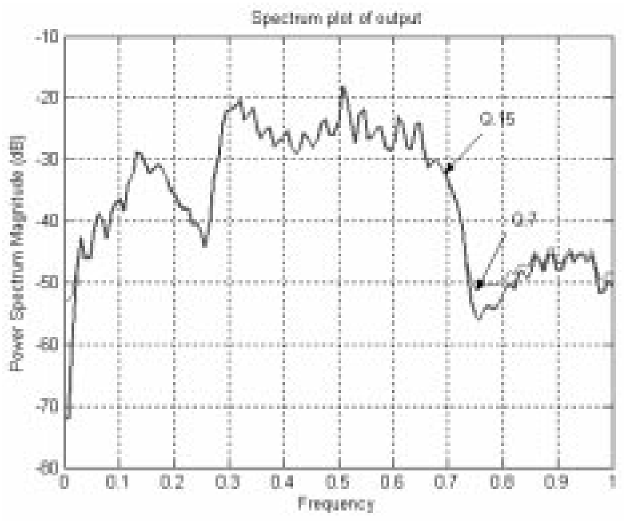


Fig. 5. Spectra of filter output using Q.15 and Q.7 formats.

```
%plot results . . .
figure(1), psd(y); title('Spectrum
  plot of input');
figure(2), psd(out_total_q15);
  title('Spectrum plot of output');
figure(3), plot(out_total_q15);
title('Time-domain plot for output
  signal');
```

The above listing shows a need to convert Q.15 numbers to the equivalent 16-bit integer numbers. This is because the Q.15 fractional data is not a standard data type in C or assembly code, which uses the integer (int) data. This conversion can be carried out by multiplying the Q.15 numbers within the range of ± 1 (excluding +1) with a scalar of 32,768. The resulting integers can then be used in C or assembly programs for the fixed-point processors. A detailed description of the fixed-point arithmetic can be found in [1]. To convert the final results back from the integer format to the equivalent Q.15 fractional numbers, they need to be divided by 32,768. These operations are illustrated in the fixed-point MATLAB code listing preceding. As explained earlier, an additional step is required to check for possible overflow of data, and saturation mode is implemented in the code. Any positive or negative overflow is clipped to its maximum positive and negative number of $+1-2^{-15}$ and -1, respectively.

Filter Design Toolbox also allows users to specify the required data formats. For example, to represent the filter coefficients, input, and all its operations using Q.15 format, the following quantized object needs to be defined:

```
q = quantizer('fix',[16,15],
  'round','saturate'); % Q.15 format
```

The input arguments 'fix' [16, 15], 'round', 'saturate' represent fixed-point format using 16-bit wordlength with 1 sign bit and 15 fractional bits. All arithmetic results are rounded, and the saturation mode is used. The double-precision floating-point coefficients can then be converted

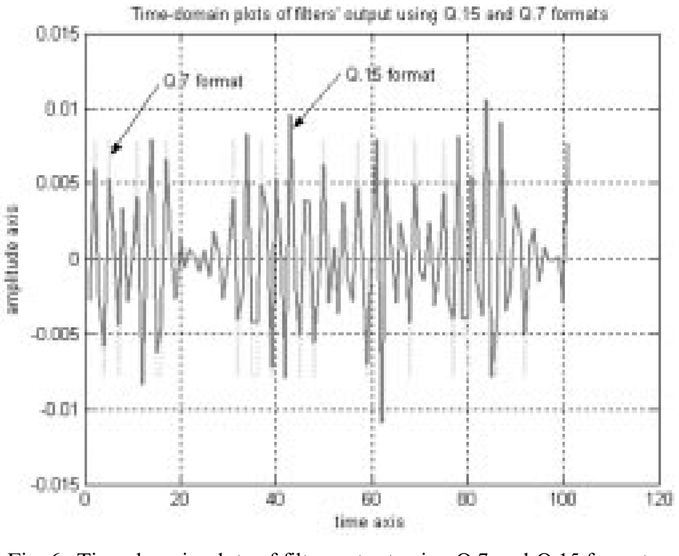


Fig. 6. Time domain plots of filter output using Q.7 and Q.15 formats.

to their Q.15 equivalents by executing the following MATLAB commands:

```
Num_lp_qfilt =
  qfilt('fir',{Num_lp},q);
Num_bp_qfilt =
  qfilt('fir',{Num_bp},q);
Num_hp_qfilt =
  qfilt('fir',{Num_hp},q);
```

The quantized FIR filtering can be computed as follows:

```
out_lp_qfilt =
  filter(Num_lp_qfilt,y).
  *quantize(q,gain_lp);
```

Here, the filter function is an overloaded function that takes in the quantized coefficients, and the quantize function quantizes the gain to the quantized object q. Figures 5 and 6 show the power spectra and time-domain waveforms of the filter output using Q.7 and Q.15 formats.

The Q.7 format is an 8-bit word which consists of 1 sign bit and 7 fractional bits. Therefore, it can be seen from Fig. 5 that the power spectrum plot using Q.7 arithmetic cannot be below –52dB. As shown in Fig. 6, the smallest magnitude that can be represented in Q.7 format is $2^{-7} \sim 0.0078125$; while that using Q.15 format is $2^{-15} \sim 0.000030517$. A more stringent FIR filter design specification (such as stopband attenuations and ripples) can also bring out the differences between fixed-point Q-formats and floating-point implementations.

In order to assess how well the students have grasped the concepts of designing and implementing digital filter, students are tasked to complete a series of exercises that tests their understanding. In addition, the students would also been asked to investigate on some fixed-point programming issues. We grade the students based on a Q&A session and a written report. Some of the exercises include:

- Design a 10-octave-band graphic equalizer to cover the audio signal with frequency up to 20 kHz.
- Use a multirate filter for implementing the graphic equalizer.
- Replace the FIR filters with IIR filters and determine their design advantages and problems.
- Use the MATLAB C-compiler to compile the M-files into C codes that can be run on PC. The compiled C code can also be ported to different DSP processor platforms.

EXTENSION TO REAL-TIME IMPLEMENTATION USING MATLAB/ SIMULINK

There are several important extensions that enhance the real-time DSP programming features of MATLAB/Simulink tools. They include:

1. Embedded Target for Texas Instruments (TI) C2000 DSP [12], which generates code for control systems on C2000 processors.
2. Embedded Target for TI C6000 DSP [13], which performs real-time prototyping and system deployment on C6000 processors.
3. Real-Time Workshop Embedded Coder [14], which generates production code for embedded systems.
4. MATLAB Link for Code Composer [15], which verifies and validates embedded software on Texas Instruments' C2000, C5000, C6000, OMAP, and TMS470 processors.
5. Embedded targets for Infineon's [16] and Motorola's [17] microcontrollers.

The main feature of these tools is that efficient working C code can be generated directly from the Simulink design. This greatly speeds up the development of DSP applications. Continual improvements on these tools will lead to the generation of more efficient code, in terms of memory, speed, and power consumption. The latest product, Real-Time Workshop Embedded Coder, enables users to generate, test, and deploy C code for used in real-time embedded systems. In addition, it allows hand-written C code to be added into the Simulink system for simulation and code generation for stand-alone applications.

These latest toolboxes are useful educational tools to illustrate the real-time applications as they are being designed, and are powerful research platforms for verifying the DSP algorithms. However, for generating more efficient code, fine-tuning, low-level programming, and using optimized library functions from the chip manufacturer are still needed for embedded DSP applications.

Another trend in the latest tool development is the integration of MATLAB environment with the code development software from the DSP chip vendors. For example, the MATLAB Link for Code Composer allows transfer of data between MATLAB and Texas Instruments' DSP processors. Data can also be acquired from the I/O channels of the DSP board to the MATLAB for further analysis.

In addition to programmable DSP processors, field programmable gate array (FPGA) is becoming an attractive platform for many DSP based systems. This is because FPGA provides a reconfigurable solution for implementing DSP applications, and normally has a higher processing power than programmable DSP processors for some specific applications. In the latest MATLAB version 7, a new product, called Filter Design HDL Coder [23] for programming the FPGA is released. Many FPGA developers, such as Alterra [18] and Xilinx [19], have introduced DSP builder or system generator that integrate DSP algorithm development, simulation, and verification capabilities of MATLAB and Simulink with their system-level design tools.

These signal processing design tools will create additional dimension to the teaching and learning of real-time DSP. More advanced and complex applications can now be introduced to bring students closer to the real-world products (for example, MP3 players, Bluetooth communication systems, video streaming for Internet, etc.) that they are familiar with. Some of these examples can also be found in the MATLAB Central under the file exchange website: http://www.mathworks.com/matlabcentral/fileexchange/loadCategory.do

CONCLUSIONS

The two-stage 'Simulink-first, MATLAB-second' approach eases the learning of theoretical and real-time DSP, and also makes the learning process more enjoyable. It provides an all-in-one platform for transferring data between Simulink and MATLAB, using filter design and fixed-point functions, and obtaining graphical plots for analysis. These powerful tools and extensions allow students to transit from concept to reality, which is an important differentiable factor for universities focusing on training practical-oriented engineers. For more information on a complete step-by-step approach in developing DSP software, readers can refer to many examples listed in [1].

Acknowledgments—This paper originated from the work carried out in the School of EEE at NTU and the EE Department at NIU. It is based on the book titled 'Digital Signal Processors: Algorithms, Implementations, and Applications,' published by the authors.

REFERENCES

1. S. M. Kuo and W. S. Gan, *Digital Signal Processors*, Upper Saddle River, NJ: Prentice-Hall, 2004.
2. S. M. Kuo and B. H. Lee, *Real-Time Digital Signal Processing*, Chichester, NY: John Wiley & Sons, 2001.
3. S. M. Kuo and G. D. Miller, An innovative course emphasizing real-time digital signal processing applications, *IEEE Trans. Education*, **39**(2) May 1996, pp. 109–113.
4. W. S. Gan, et. al., Rapid prototyping system for teaching real-time digital signal processing, *IEEE Trans. Education*, **43**(1) Feb. 2000, pp. 19–24.
5. W. S. Gan, Teaching and learning the hows and whys of real-time digital signal processing, *IEEE Trans. Education*, **45**(4) Nov. 2002, pp. 336–343.
6. The MathWorks, *MATLAB User's Guide*, *Version 6.5.1*, 2004.
7. The MathWorks, *Simulink User's Guide*, *Version 5.1*, 2004.
8. The MathWorks, *Signal Processing Toolbox User's Guide, Version 6*, 2003.
9. The MathWorks, *Filter Design Toolbox User's Guide*, *Version 2.5*, 2003.
10. The MathWorks, *Fixed-Point Blockset User's Guide*, *Version 4*, 2003.
11. The MathWorks, *DSP Blockset User's Guide*, *Version 5*, 2003.
12. The MathWorks, *Embedded Target for the TI TMS320C2000^{TM} DSP Platform Version 1*, 2003.
13. The MathWorks, *Embedded Target for the TI TMS320C6000^{TM} DSP Platform*, *Version 1*, 2002.
14. The MathWorks, *Real-Time Workshop® Embedded Coder User's Guide, Version 3*, 2003.
15. The MathWorks, *MATLAB Link for Code Composer Studio Development Tools User's Guide*, *Version 1.3*, 2003
16. The MathWorks, *Embedded Target for Infineon C166® Microcontrollers User's Guide, Version 1*, 2002.
17. The MathWorks, *Embedded Target for Motorola MPC555 User's Guide*, *Version 1*, 2002.
18. Alterra, FPGAs Provide Reconfigurable DSP Solutions, white paper, August 2002, www.altera.com.
19. Xilinx Inc., System Generation for DSP, brochure, 2002, www.xilinx.com.
20. The MathWorks, *Filter Design Toolbox User's Guide, Version 3*, 2004
21. The MathWorks, *Fixed-Point Toolbox User's Guide*, *Version 1*, 2004.
22. The MathWorks, *Simulink Fixed Point User's Guide*, *Version 5*, 2004.
23. The MathWorks, *Filter Design HDL Coder User's Guide*, *Version 1*, 2004.

Woon-Seng Gan received his B.Eng. (1st Class Hons) and Ph.D. degrees, both in Electrical and Electronic Engineering from the University of Strathclyde, UK in 1989 and 1993 respectively. He joined the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore as a Lecturer and Senior Lecturer in 1993 and 1998 respectively. Currently, he is an Associate Professor. He teaches several undergraduate, postgraduate and industry courses on Digital Signal Processing and Real-Time Signal Processing Implementation. His research interests include adaptive signal processing, psycho-acoustical signal processing and real-time digital signal processing. is a Senior Member of IEEE, Member of Audio Engineering Society and Professional Engineers of Singapore. He has recently co-authored the book *Digital Signal Processors: Architectures, Implementations, and Applications*, Prentice-Hall 2005.

Sen M. Kuo received the B.S. degree from National Taiwan Normal University, in1976 and the MS and Ph.D. degrees from the University of New Mexico, in 1983 and 1985,

respectively. He is currently a Professor and Chair in the Department of Electrical Engineering, Northern Illinois University, DeKalb, IL. In 1993, he was with Texas Instruments, Houston, TX. He is the leading author of three books: *Active Noise Control Systems* (Wiley, 1996), *Real-Time Digital Signal Processing* (Wiley, 2001), and *Digital Signal Processors* (Prentice-Hall, 2005). He served as a consultant to several companies on developments of real-time DSP applications such adaptive echo cancelers. He holds seven US patents, and has published over 150 technical papers. His research focuses on active noise and vibration control, real-time DSP applications, adaptive echo and noise cancellation, digital audio applications, and digital communications. Prof. Kuo received the IEEE first-place transactions (Consumer Electronics) paper award in 1993, and the faculty-of-year award in 2001 for accomplishments in research and scholarly area.