# THE USE OF CONCURRENT CODES IN COMPUTER PROGRAMMING AND DIGITAL SIGNAL PROCESSING EDUCATION

William Bahn, Leemon Baird, and Michael Collins
Department of Computer Science
United States Air Force Academy
USAFA, CO 80840
719 333-8782
William.Bahn@usafa.af.mil

**ABSTRACT**

Introducing relevant and meaningful real-world applications in exercises intended to teach traditional undergraduate topics is difficult; limitations on depth and scope frequently result in only a passing relationship between application and exercise. However, the recent development of concurrent coding theory and its application to keyless omnidirectional jam resistant communication offers one opportunity to achieve this because the underlying problem is readily understood and the algorithms are tractable at the introductory level. Furthermore, no exotic or expensive hardware is needed to construct functioning implementations – a computer with a soundcard, microphone, and speaker is sufficient. As a result, students gain an appreciation for a real-world problem of growing importance including meaningful insight into how that problem can be addressed while maintaining a focus on the primary material being taught.

## INTRODUCTION

Perhaps the most common approach, at least historically, of presenting introductory material in science and engineering curricula is to focus strictly on the material being taught and devise student exercises that are similarly narrow in focus. This is essentially a "bottom up" approach wherein students first develop a wide set of narrow skills and then begin combining those skills in progressively larger problems as they progress through the curriculum. While many students might learn adequately in such a program of study, other students may become frustrated and quit early in their studies because they feel they are working in a vacuum learning a bunch of disjointed topics that have little relation to each other or the real world.

To counter this, a growing trend is to adopt a more "top down" approach by integrating real-world problems and applications into introductory science and engineering courses. Unfortunately, the depth and scope limitations of such courses frequently relegate the "real world" aspects to little more than buzzwords having only a fleeting relationship to the exercise at hand. For example, one popular computer programming text attempts to link course material to several "grand engineering challenges" facing society, one of which is designing large aircraft to be more fuel efficient. The tie to the course material is that aircraft designers use wind tunnel test data relating lift to angle of attack and that the data is generally analyzed using computer programs. But, after leading students to expect that the programs they will be writing will somehow provide a bit of insight into the analysis of such data, the actual exercises involve nothing more than sorting a list of numbers. Student's frequently see this as a

"bait and switch" tactic and are understandably annoyed that they have been required to read material that, in the end, has no relevance to what they are actually asked to do.

What is needed are real-world problems that have substance to them yet can be explored using only the tools and techniques available to lower division engineering students. Similarly, it is important that students work exercises that actually require an understanding of the real-world problem being discussed to prevent them from feeling that they have been mislead. Unfortunately, such problems are few and far between and this will probably only be aggravated as technology progresses. None-the-less, suitable contemporary problems do occasionally arise – the challenge being to recognize them as such and take advantage of them in the classroom.

The recent development of concurrent coding theory is one such example. Concurrent codes permit jam-resistant omnidirectional communications without shared secrets. While real world applications are in the radio-frequency domain, demonstration applications can be implemented using image and audio files that use identical encoding, decoding, and (in the case of audio files) signal processing.

Finally, educators have at their disposal a script-driven open-source application program that gives instructors a fine degree of control between tasks that are performed by a "black box" and tasks that the students are expected to perform for themselves.

## CONCURRENT CODES

Present omnidirectional jam-resistant systems are not suitable for use in large ad hoc networks such as those that are an integral part of the Global Information Grid that is in the early stages of implementation. The reason is that all such systems rely on a "shared secret" key and the secure distribution and management of such keys on such a scale is simply not feasible. Concurrent codes permit the implementation of a system that does not rely on shared secrets but that has a roughly comparable degree of jam resistance. Space limitations do not permit a meaningful discussion of the underlying problem of omnidirectional jam resistance, what concurrent codes are, or how they can address the growing need for keyless jam-resistant systems. The interested reader is directed to the original technical report [1] that introduced concurrent coding theory. In essence, however, concurrent codes are designed to allow the extraction of all possible messages that are consistent with the received data and, by using highly asymmetric transmission channels, the jammer is relegated to flooding the receiver with false messages since it is virtually impossible to block legitimate messages.

### The BBC Algorithms

The BBC algorithms are the first (and, at present, only practical) encoding and decoding algorithms based on concurrent coding theory. Detailed descriptions of the algorithms are contained in the tech report [1] for, as short as they are, space limitations preclude their inclusion here.

### The BBC Application

Before delving into the use of concurrent codes in course exercises, the open-source BBC application will be discussed briefly since instructors can utilize it to narrow the focus of their students' efforts. The BBC application is a standalone ANSI-C compliant processing engine that permits users to develop scripts that carry out a host of

processing tasks related to concurrent codes and the various ways to represent data packets. A detailed discussion of either its capabilities or its inner workings is beyond the scope of this paper. Instead, a couple of simple example scripts will be examined to gain a feel for how the program is used. How these scripts can be modified and used to support course exercises will be discussed in later sections.

```
// BMP.bbc - BBC Script Template for BMP files
// ===================================================
CLEAR -a
CODEC -m 512 -e 1000 -k 50
PACKET -e 1
MSG -r "pic_msgs.txt"
ENCODE -m
PACKET -d
BMP -w "demo.bmp"
ECHO Mark-up "demo.bmp", save, then
PAUSE -r
BMP -r "demo.bmp"
PACKET -d
MSG -e
DECODE -s
MSG -d -w "received.txt"
```

The BMP.bbc script first clears any existing data and configuration information from the processing engine. It then configures the encoder/decoder (codec) for messages that are 512 bits long with an additional 50 checksum bits appended to the end of the message and codewords that are 1000 times as long as the base message (512,000 bits). It then creates a packet with an expansion of one, making it the same size as a single codeword. The messages are read from the indicated text file where each line in the file is a separate message. The messages are then encoded into the packet and the packet statistics, most notably the overall packet density, are displayed. The packet is then formatted as a Windows bitmap file and written to disk. The next line echoes a message to the User and the following line pauses until the User hits ENTER (and issues a corresponding message that completes the partial message in the ECHO command). The User can then use any image editing tool, such as Microsoft Paint, to add additional black marks to the image and save it back on top of itself. After hitting ENTER, the BBC program reads the file, displays its statistics, erases all messages in the internal message list, and decodes the packet adding each message found to the message list. Finally the message list is displayed on the screen and written to a text file with one message per line.

```
// WAV.bbc - BBC TX/RX Script Template for the WAV files
// ===================================================
CLEAR -a
CODEC -m 128 -e 100 -k 50
PACKET -e 1
MSG -r "tx.txt"
ENCODE -m
WAV -c -b 1000 -s 11025 -o 0 -e 80 -f 5000 -g -w "tx.wav"
ECHO Record "rx.wav" and
PAUSE -r
```

```
// Configure Filter
// http://www.dsptutor.freeuk.com/remez/RemezFIRFilterDesign.html
// High Pass
// Fs  = 11.025 kHz
// F_o =  4.4kHz (0.4*Fs)
// Transition = 1100Hz (0.1*Fs)
// Passband Ripple: 1 dB
// Stopband Rejection: 40 dB
FILTER -x 0   0.026326745981419188
FILTER -x 1   2.3394244186887697E-4
FILTER -x 2  -0.062132178678304895
FILTER -x 3   0.1601061482350591
FILTER -x 4  -0.25254345256890826
FILTER -x 5   0.2905859513171658
FILTER -x 6  -0.25254345256890826
FILTER -x 7   0.1601061482350591
FILTER -x 8  -0.062132178678304895
FILTER -x 9   2.3394244186887697E-4
FILTER -x 10  0.026326745981419188

WAV -r "rx.wav"
FILTER -s
RAD -f 95.0 -s
WAV -l 2.0 -n 0 1
DSC -1 10.0 -0 10.0 -s
WAV -p 1.00
PACKET -d
MSG -e
DECODE -e 10000 -s
MSG -w "rx.txt"
```

The WAV.bbc script starts out similar to the BMP.bbc script by configuring the basic codec, reading messages from a text file, and encoding them into a packet. The next line configures the transmitter's modulator by setting the packet bit rate to 1 kbps, the sampling rate to 11.025 kHz, and the packet delay (origin) to 0 samples. It further defines a transmitted mark to consist of a 5 kHz sinusoidal burst lasting 80% of a bit duration. Finally, it generates the waveform data, writes it to a Windows Wave file, and then pauses to give the User the opportunity to play the resulting file, perhaps along with others that have been generated, and combine them by recording them into a new file.

After the User hits RETURN, the script creates a finite impulse response filter by explicitly defining its coefficients and then, after reading the data from the input audio file, applies the filter. A simple infinite impulse response radiometer is then defined wherein 95% of the prior output is combined with 5% of the present input. The resulting waveform is then clipped at a maximum amplitude of 2 and the result is normalized to a range of 0 to 1. The next step implements a Schmitt trigger with, in this case, both the rising and falling thresholds set to 10% of the normalized range. Following this, the binary packet is produced from the waveform data by declaring a mark (HI) to exist if the discrimator output is HI anywhere within a 1.0*bit-period window centered on the nominal bit center. After this, the rest of the processing is similar to the BMP.bbc script except that the decoder is told to assume that the transmitting and receiving oscillators might be off by as much as 1% (10000 ppm).

## STUDENT EXERCISES BASED ON CONCURRENT CODES

Concurrent codes lend themselves to student exercises in either computer programming or digital signal processing. To avoid making assumptions about the order in which the corresponding courses are taken, the exercises in these two areas are presented as independent of the material in the other course as possible.

The BBC application can be used to support the course either by performing those steps that are outside the scope of the course or by letting the student work with a completely functioning suite of tools that they incrementally replace with their own code and/or processing blocks.

### Programming Exercises

While the exercises discussed can be implemented in most programming languages, ANSI-C compliant code is the framework for this discussion. In the context of a semester course in computer programming, an entire syllabus can be devised where nearly every example and exercise is an incremental step in the development of a significant final program that performs a narrow subset of the BBC application's capabilities. Following is one possible sequence of exercises, and the topics introduced with each one, that could be considered.

1) Implement a very simple command-prompt environment
   Console I/O, string operations, loops, selection statements.
2) Encode messages entered via the keyboard using a fixed codec configuration.
   Simple bit-level operations, text file output, array operations, hash functions.
3) Configure the codec via command line, keyboard, and/or configuration file.
   Text file input, command line processing, simple script processing.
4) Decode message packets.
   Recursion, dynamic memory allocation, data structures, linked-lists.
5) Represent message packets as Windows BMP files.
   Binary file output, standard file formats.
6) Extract packets from Windows BMP files.
   Binary file input.

It's important to note that the BBC application can be used to perform any tasks that the students have not implemented on their own yet. As a result, from the very first exercise, students have a complete working system that they can continually use to verify their own code's functionality. In addition, since the exercises progressively build on each other, students are motivated not to take the normal approach of forgetting about an exercise the moment it is turned in.

It's also worth noting that some of these topics might be beyond the desired scope of the course or they might be introduced earlier than when the instructor desires. In most cases there are simple ways to reorder or recraft the exercises so that the troublesome topics are either eliminated or isolated inside instructor-supplied functions. For example, decoding message packets is by far the most sophisticated task and some instructors might choose to place it last in the sequence. Other instructors might choose to forego

having students write hash functions (although the hash functions needed are quite tame) and simply supply the hash function code up front.

Finally, it should be noted that the students do not need to ever be exposed to any digital signal processing, either by working only with image files or by using instructor-supplied scripts when working with audio files.

**Digital Signal Processing Exercises**

It is unlikely that an entire semester syllabus in digital signal processing could be crafted around concurrent codes, but using them as the underlying topic for several exercises constituting perhaps one quarter to one half of the course is not unreasonable. Furthermore, a DSP course probably would not have students implement the coding and decoding algorithms, especially if these were being done in a separate programming course. Instead, the course would use the BBC application to perform these steps while the students focus on understanding, designing and applying the various filters.

Initially, the students would design the filters and enter the coefficients into a BBC script. This allows them to focus on the math involved in the filter design and not get sidetracked by the housekeeping involved in actually implementing the filter. Then, with the math understood, they could implement the actual filters and filter the waveforms separately from the BBC application using a program they've written or a commercial application such as Excel or MatLab. Since the BBC application can read and write the data at any point in the processing, either as an ASCII text file or as a Windows Wave file, the instructor can easily choose which parts of the processing the students are to perform separately.

Another aspect that makes concurrent codes a good test bed for an introductory DSP course is that students can tackle the problem in incremental steps. They can first encode the packet into a waveform file and then decode that same waveform file, allowing them to perform their signal processing on ideal waveforms. They can then play and record the file in order to see the effect that filtering has on helping with the inevitable distortion and noise. Finally, then can see how their processing is able to deal with actual jamming signals from their peers.

**CONCLUSIONS**

Concurrent codes offer an opportunity to craft student exercises in either a computer programming or a digital signal processing course that have meaningful relationships to the growing real-world problem of keyless jam resistance. Furthermore, students can benefit by the availability of the BBC application program that allows them to work with a complete encoding/decoding system at each step of their work so that they can verify successful implementation of the portion of the problem they have been tasked with. The end result should be greater student interest in the exercises with a correspondingly better mastery and retention of the course material.

**REFERENCES**

[1] Baird III, L.C., Bahn, W.L., Collins, M.D., Jam-resistant communication without shared secrets through the use of concurrent codes, *Technical Report USAFA-TR-2007-01*, United States Air Force Academy, 2007.