TEST EQUIPMENT PLUS

BB60 Demo Manual

Operation and Programming Guide

Andrew Montgomery, Justin Crooks

10/9/2012

Contents

Introduction

The purpose of the FM demo application is to show the power of the 20MHz real-time streaming bandwidth of the BB60A. The device is capable of streaming the bandwidth at 80 million 14-bit samples per second. In this demo we stream the entire FM band from 88 – 108 MHz. All you need is a low cost external antenna.

In real-time we collect the 80 million samples, display the frequency spectrum, and demodulate the audio of two FM channels chosen by the user. The user interacts with the graticule display to select which two FM stations play on the left and right audio speakers.

Additionally the demo provides the ability to save the FM band to disk for later playback. With the use of newer solid state drives and simple RAID arrays, you can store all 80 million time domain samples per second. Later playback from a saved session will reveal a snapshot of every FM station from the time of recording!

Additional Introduction for Programmers

The FM demo is a great place to get started in programming the BB60A. While most end users will not be using the BB60A for analog FM demodulation, the FM demo illustrates many of the features of a application/system interfacing the BB60A. Such as interfacing the API, multi-threaded real-time signal

processing, and high performance disk I/O for applications requiring RF recording elements. All programming elements were chosen for simplicity and illustration in this demo.

This manual will detail a high and low level flow of information through this system. Read the section titled “Programming Guide” to get started.

Operation

Installation

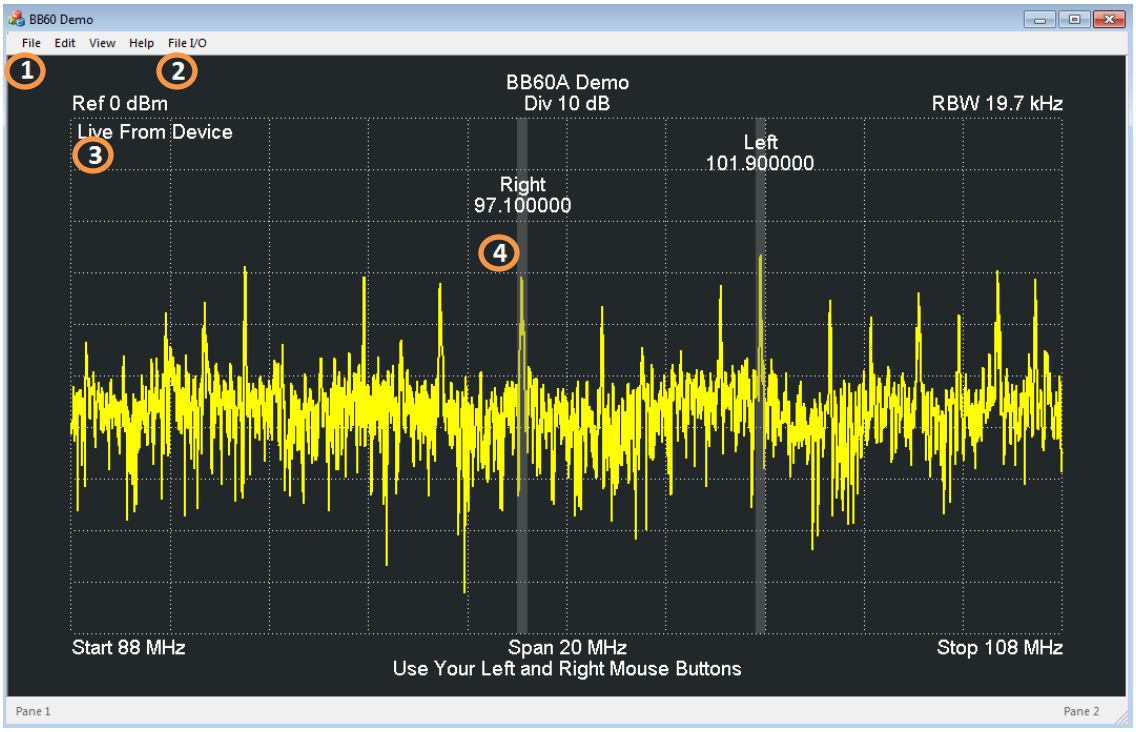
To run, the executable needs to be in the same directory as the API (“BB60APIW32.dll”) and the OpenGL extension library (“glew32.dll”). The API can be downloaded from the SignalHound website. The OpenGL extension library can be downloaded from glew.sourceforge.net. Both files are distributed with the BB60A installer which can be found on the website or on the installation disk shipped with the BB60A.

1) If you installed the BB60A spectrum analyzer software from the software disk provided, the demo executable is found in the same directory as the main software.
2) If you have downloaded this demo from the website and you have the BB60A software installed, you only need to place the demo executable in the same directory as the main software executable.
3) If you downloaded the demo from the website and have not installed the main software you will need to install the main application on your system. Once installed place the demo executable in the main BB60A directory.

Running

The program itself shows you the FM spectrum from 88 – 108 MHz allowing you to “navigate” with your mouse and listen to any FM station in the band. The audio playback from the device or from a capture file is real-time. Feel free to listen to a capture file multiple times and listen to different stations each time to be convinced the entire 20MHz band is there.

Upon running the program for the first time, you should see something similar to the image below if you have a BB60A attached.



A quick glance at various features of the display.

1) File Menu – Here is where you will find the functionality to connect/disconnect a BB60A. If the program is launched while a BB60A is attached, a connection will be established automatically.
2) File I/O Menu – Here you will find all options for reading and writing to/from disk.
3) State Text – Shows the state of the demo, possible states are
    a. **Idle** – Inactive.
    b. **Live From Device** – Retrieving and showing data from the device.
    c. **Live Recording** – Data retrieved from the device and saving to disk.
    d. **Playback From File** – Data is being retrieved from a file.
4) Channel Bars – There are two channel bars, moved by using the left and right mouse buttons. The channels bars control the sound to the left and right speakers.

If no device is attached during program launch, the program will launch in an idle state.

Playback

To begin a playback from a data capture navigate to the File I/O menu and select “From Disk”. A file select dialog will appear. Navigate to the directory containing the capture xml file and open it.

Opening the file will trigger the playback to begin. From here you can interact with the interface by clicking on various portions of the spectrum. Using your left and right mouse buttons, you can control which FM stations are audible on your left/right speakers.

When the capture data is exhausted, the application returns to its previous state. Replaying the capture is as simple as reopening the xml file.

Programming Guide

This section outlines the code behind this demo. This section will help you understand the structure of the code and how it works. The demo was programmed in C++ using Microsoft's MFC Library. It was built in Visual Studio 2008 and follows the traditional Document View architecture.

Setup and Requirements

If all you desire is to view the code, nothing is needed but a text editor, preferably one that is geared towards source code editing.

If you wish to compile this program, this is a list of what is needed.

1) Microsoft Visual Studio 2008 or later, with the capability to compile MFC applications.
2) Intel IPP Signal Processing Library. A free trial version is available which provides a developer with access to all development tools.
3) The .dll's and .lib's provided in the demo, BB60APIW32.dll/.lib, glew32.dll/.lib

Program Flow

Below is an image of a high level flow chart showing the larger components in this system interacting across various threads of execution. The four main thread flows are shown.

1) The main thread: This is the entry point into this program. The thread is responsible for creating all the objects in the system as well as Windows message handling and drawing the user interface.
2) Processing loop thread: Contains the main loop, responsible for interfacing all objects and coordinating the flow of information through the system.
3) Disk I/O thread – Responsible for interfacing the disk and reading/writing the files to/from disk.
4) Audio processing and playback thread – Responsible for managing the threads which perform the digital downconversion/decimation/and demodulation. Also responsible for sending the demodulated audio signal to the audio devices. The "thread" in the diagram represents 5 different threads performing real-time signal processing communicating through the use of circular buffers and events.

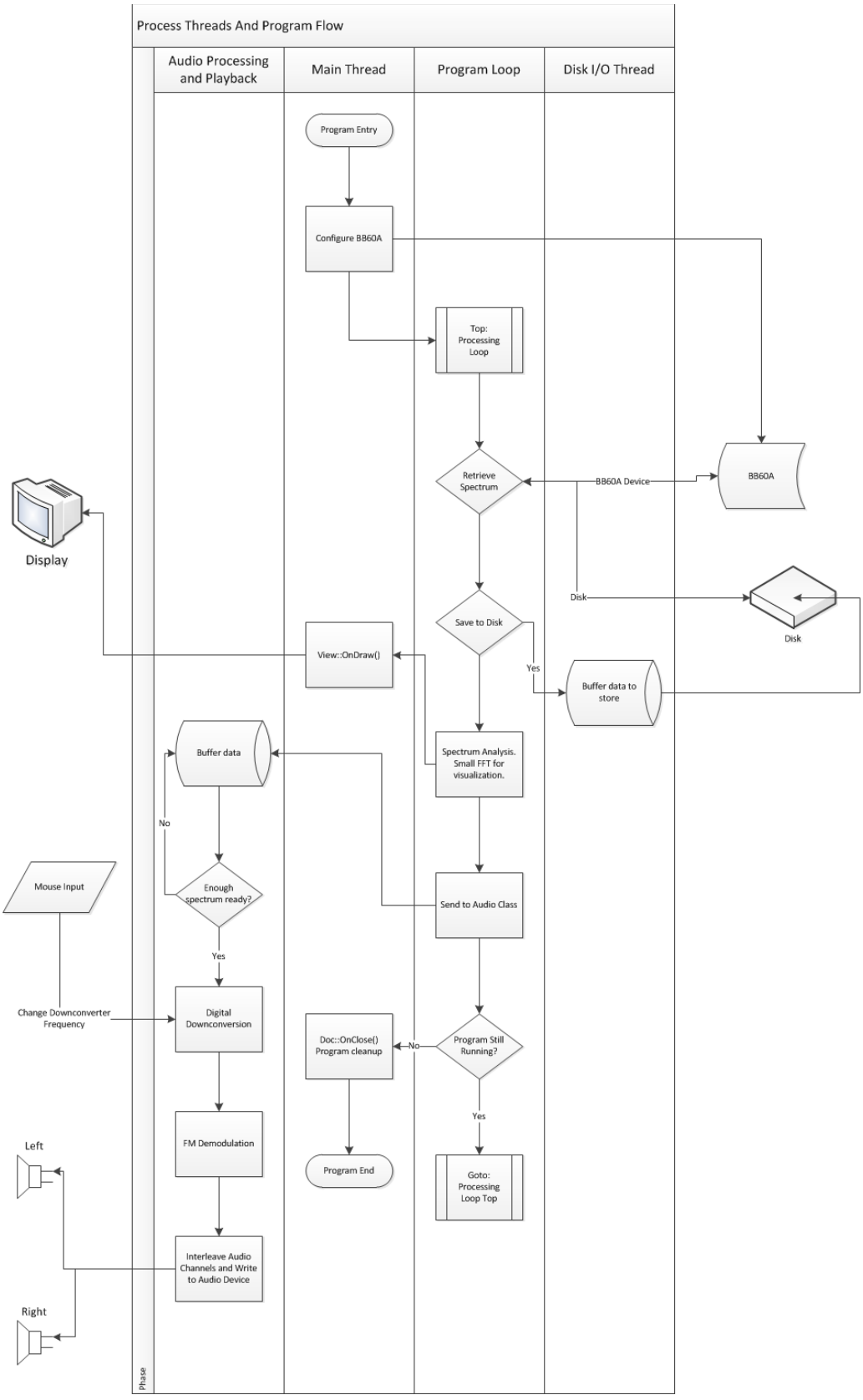


Figure 1: High Level Flow Diagram

Program Structure

There are four main parts to this demo application.

1) Interfacing the BB60A device.
2) Drawing the frequency spectrum.
3) FM-demodulation of the spectrum.
4) Disk Input/output

The program uses Microsoft's MFC Libraries. MFC uses the Document-View architecture. In an ideal MFC application all data is stored and modified in the Document(just a C++ class), while the View(another C++ class) creates the interface based on the data stored in the Document. In this demo, we follow this paradigm. All data/variables needed for signal processing and operation are stored and created in the Document. The view is a friend class of the Document and has access to any data it needs to rasterize the display. The View will also contain any data needed only to draw.

Interfacing the BB60A

To understand this section on interfacing it is recommended you look at the BB60A API programming manual.

In the demo we utilize the API to retrieve the raw data from the device. All device interfacing happens in the document. Very little configuration is needed to begin retrieving data from the device for this program. This program configures the device for the "raw pipe" mode and chooses settings which help bring out the best signal for the FM band.

Drawing the Frequency Spectrum

Drawing is done primarily with OpenGL. The view is updated about 30 times per second with a new snapshot of the spectrum. A small 8192 length FFT is performed to create the graticule. All drawing is done in the View class.

Audio Demodulation

Demodulation is performed in the AudioStream class. The class buffers 10.24 million samples before demodulating to the resulting audio. Once the 10.24 million samples are collected, 4 threads begin the work of demodulating. Each thread handles either a left or right channel, and the real or imaginary portion of the input signal. First, the input is mixed with the channels frequency, and then digitally downconverted, through a process of FIR and IIR filters and downsampling. See the function AudioStream::processThread() for this process.

Once the downconversion takes place, a new thread demodulates the left and right channels. See AudioStream::demodAndScale(). This function determines the phase using the downsampled real/imaginary parts of the input signal, which is then used to construct the audio signal. Two more filters are then ran on the audio signal, for FM-deemphasis and to maintain nyquist for the final downsampling.

The result of this is two 4096 length audio signals which are interleaved and written out to the PCs audio device.

This processing structure is choreographed through the use of Win32 events and circular buffers.

Saving to Disk

The problem of saving the data to disk is addressed in this demo. All disk I/O happens in the DiskController class. To achieve real-time 160 MB/s rates all saving is done in a separate thread, and data is allowed to be buffered in the event of interruption. Through testing we have determined interruption can happen unexpectately and without notice, due to the program not operating under Real-Time thread priorities. We have found it necessary to buffer up to a ½ second worth of data and trigger the thread to save new data with Win32 events. We do no such buffering when reading in from disk, as we are willing to accept small interruptions (may cause slight stuttering on playback).

Our saving scheme uses a XML file to denote the settings of the device, and we save incremental 1GB binary files stored as unsigned shorts. The binary files are named similarly to the XML file, and are sequentially numbered to denote ordering.

Contact Information

For programming related questions please contact Andrew Montgomery at aj@teplus.com. For hardware specific questions relating to the operation of the BB60A please contact justin@teplus.com. We are interested in hearing feedback, criticism, and praise. If you would like to see a new demo, let us know, we might be able to help.