GR-DSP
GNU Radio extensions for OMAP processors utilizing the C64x+ DSP
By: Almohanad (Al) Fayez
1/27/2011

- **Summary:**

To summarize what's needed for DSP to be integrated with GNU Radio please look at figure 1. Looking at it from a bottom up approach:

- There's a program running on the DSP performing some functionalities, FIR filtering in this case.
- The DSP is able to communicate with the GPP using DSPLink, a TI shared memory interface.
- The GPP side of DSPLink is encapsulated in a shared library.
- There are GNU Radio blocks which utilize the DSP, those blocks use the GPP side DSPLink shared library to communicate with the DSP. It specifies the function it wants from the DSP and it sets them up, e.g. filter coefficients. Figure 2 shows how the DSP based GNU Radio blocks operate like.

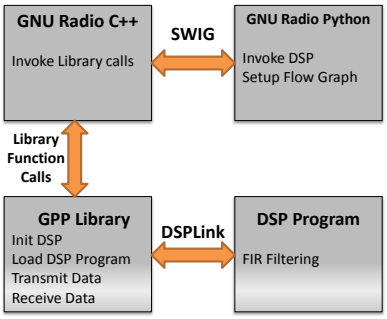

Figure 1. Summary [1]

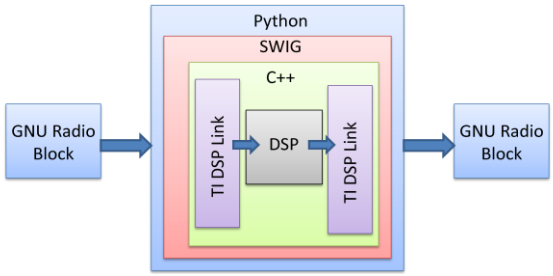

Figure 2. Application level abstraction of the DSP in the OMAP processor. [1]

- **Assumptions:**
  1- You know what openembedded is.
  2- You're using Angstrom 2010 as your OS.
  3- You're using Openembedded and you can build a base filesystem for a Beagleboard and/or USRP E100

- **What's Included:**
  - dsplink_code:
    - dsp
      - The DSP side source code. Includes a dependency to the IQ_MATH_c64x+ TI library.
      - I use the cpExe.sh script to compile and copy the executable, loopAl2.out, to a know location.
      - You should copy over the "loopAl2" folder into the DSPLink DSP examples folder … more about this later.

        - The generated excutable "loopAl2.out" MUST be in the same folder as the flowgraph you'll be invoking the DSP from … for now the executable name is hard coded.
    - Gpp
        - Includes source code for building the DSPLink GPP library. It's a bit of a hack, you have to take the *.o from it and use Openembedded to cross compile the shared library.
        - I use the buildLib.sh script to compile and copy the necessary files to their directory so I can use OE to build the GPP library.
        - You should copy over the "loopAl2" folder into the DSPLink GPP examples folder.
    - Sample_makefiles
        - When you install dsplink you need to modify your system makefiles to point to all the right tools and directories. I've included what mine look like as a reference.
    - InitBeagle.sh
        - An init script I have to load various TI drivers. They should be loaded at startup instead of loading them this way. Please note you don't really need cmemk or sdmak drivers for gr-dsp to work.
- OE_recipes:
    - Gpp_lib
        - Includes the necessary recipe to build the install packages for the GPP.
        - The "Files" folder has a header file and the *.o needed to build the library. Please note the "buildLib.sh" script should move the files here from the GPP DSPLink directory.
    - Gr-dsp
        - Includes a recipe to build the GNU Radio blocks which utilize the DSP.
        - Also includes a tar with the source files.
    - I place all of what's here in the gnuradio recipe folder in the OE but you can make a separate folder if you'd like.
    - You can run "bitbake –b recipe"
- Omap_sample_flowgraphs
    - Sample flowgraphs which use DSP based blocks … not really the best looking flowgraphs.
    - IMPORTANT: make sure that you have the DSP executable "loopAl2.out" in the same folder as your flowgraph or you won't be able to load the DSP.
- Pre-baked-packages-2_6_37
    - Contains packages you can install to use the GNU Radio DSP extensions.
    - You can use them to avoid dealing with any OE related pain.

- **Installing TI software**:

Please start by reading through the following reference
http://ossie.wireless.vt.edu/trac/wiki/BeagleBoard

The info on the website is for older versions of the TI software. You can ignore the change for "STD_KRNL_FLAGS" variable and you need to remember to update your Rules.mk file. If you're able to re-compile and run all of DSPLink examples then you're good to go.

Please note that there is a bug in the OMAP processor as indicated on the OSSIE website, you need to use the power management utility lpmOFF.xv5T to turn off the DSP and lpmON.xv5T to turn it back on when you're loading different DSP executables. The LPM utilities are installed vi the "ti-lpm" package.

My preference is to build my DSPLink apps outside of the OE, if you've used OE long enough you'll know what I mean, some day you'll decide to start a fresh build from scratch and rm –r the temp build file and there goes your code. Follow the instructions on the OSSIE website to install XDC, DSPBios, DSPLink, LPM. For more info on installing DSPLink look at the InstallGuide_Linux_OMAP3530.pdf guide in the doc folder.

- Building OE Recipes:
    - Build the GPP-lib recipe before the gr-dsp recipe.

- Installing packages on the Beagleboard or E100:
    - Opkg install "package name"
    - Opkg install --force-reinstall "package name"
        - Force the reinstall of a package without opkg complaining that the package already exists.
        - There's a bug where when I reinstall a package after modifying it opkg occasionally complains about the package trying to re-write an existing file. When that happens just run the command again after it fails and it should work.
    - Opkg remove "package name"
        - When in doubt remove a package before reinstalling it.

- List of Blocks:
    - **Dsp.init**()
        - Initializes the DSP.
    - **Dsp.clean**()
        - De-allocates DSPLink related resource.
    - **Dsp.fir_ccf**(coeff, scaling, interpolation, input sig, output sig, init DSP, block_id)
        - Coefficients = normalized coefficients, for now the filter tap size is restricted to 490.
        - scaling factor = [0, 15]
            - 2 ^ scaling_factor
        - Interpolation factor = I wouldn't use the interpolation feature … needs fixing.
        - input signature
            - 0 = normalized, input needs to be scaled before being sent to the DSP … the DSP is fixed point.
            - 1 = input is already scaled properly by a factor equal "scale", just pass it to the DSP.

        - output signature
            - 0 = normalize output.
            - 1 = keep output in the same scale factor as it was on the DSP.
        - initialize DSP: (set to 0 use dsp.init() instead)
            - 1 = yes, please init
            - 0 = no, will init using dsp.init()
        - block ID
            - must pick either 0 or 1 for block id.
            - Will be more agile sometime in the future.
            - There's support for only two blocks for now.
    - Dsp.fir_ccf_fm_demod(coeff, scaling, interpolation, input sig, output sig, init DSP, block_id)
        - Implements a ccf filter followed by FM demod before sending the data back to the DSP.
    - Dsp.fir_ccf_fm_demod_decim(coeff, scaling, decimation, input sig, output sig, init DSP, block_id)
        - Implements a ccf filter followed by FM demod and decimates the buffer before sending the data back to the DSP. It assumes that user will filter the result with an FFF filter on the GPP to finish performing the filter … will change as soon where it's all done on the DSP.
    - Dsp.fir_fff
        - I wouldn't use it … at least not now something needs to be fixed.

- DSP Program Development:
    - Debugging DSP code using a JTAG:
        - Soon
    - TI Library Dependencies:
        - Soon

- Make the guide better:
    - Soon

[1] Fayez, A. S., Chen, Q., & Nounagnon, J. B. (December 1, 2010). Leveraging Embedded Heterogeneous Processors for Software Defined Radio Applications. *SDR 2010.* Wshington, DC