07-3 Using the DSP on the BeagleBoard via **c6run**

Mark A. Yoder

So how do I use the DSP?

- Skip the DSP altogether
  - Use Arm Neon floating-point
- Bare Metal Approach
  - Get out the hardware manual and learn where the shared memory is…
- Something in between – Use a framework
  - c6run
- Find prepackaged software that uses the DSP
  - Gstreamer (http://www.gstreamer.net/)
    - open source multimedia framework

c6run Overview

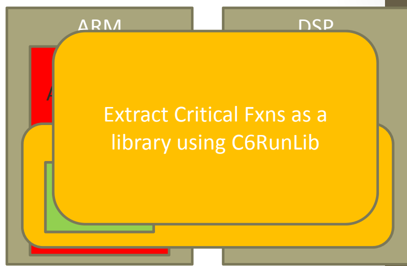
- Adapted from: http://processors.wiki.ti.com/index.php/Introduction_to_C6Run
- Open Source Project, hosted on http://gforge.ti.com
- Intends to provide an abstracted mechanism for getting code running on the DSP
- Project Goals
  - Introduce DSP Development to ARM/Linux developer
  - Simplify simple application/function offloading to the DSP
  - Start getting Linux and open-source community using the DSP

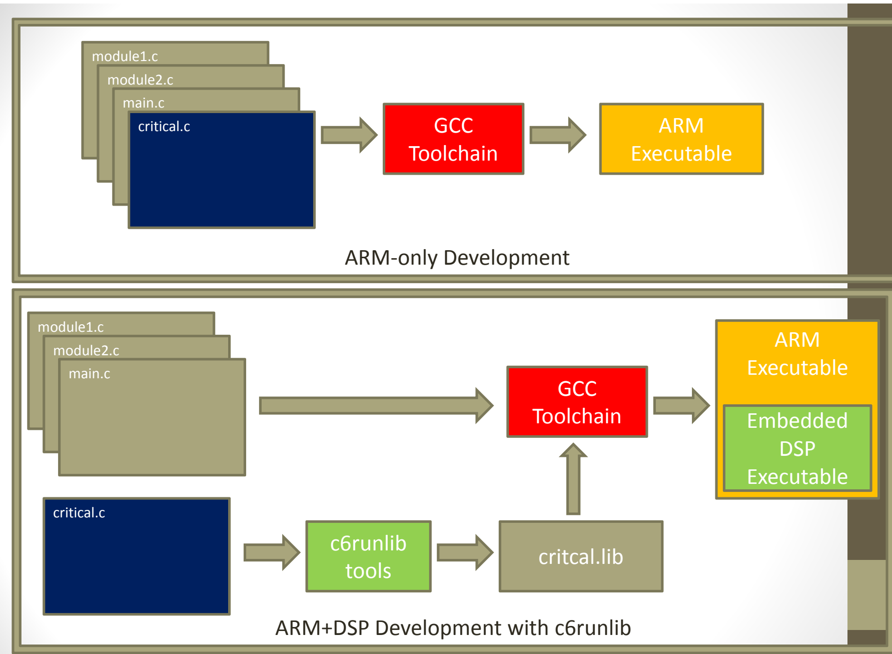
Project Details

- Currently consists of several components:
  - C6Run**App**: Run an entire application on the DSP
  - C6Run**Lib**: Partition an application between the ARM and DSP

C6Run**Lib**

- Goal: automate building an ARM-side library from user's C code of critical functions
- User app can call into library, and calls will be executed on the DSP
- Consists of automating creation of framework, hiding other TI specific bits





Example C6RunLib Usage

```
$ c6runlib-cc -c -O2 –o dummy.o dummy.c
```

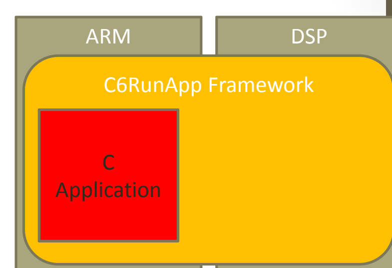
- Above converts C code containing critical functions to C6000 object file
- Also analyzes global C functions and generates ARM-side remote procedure call stubs

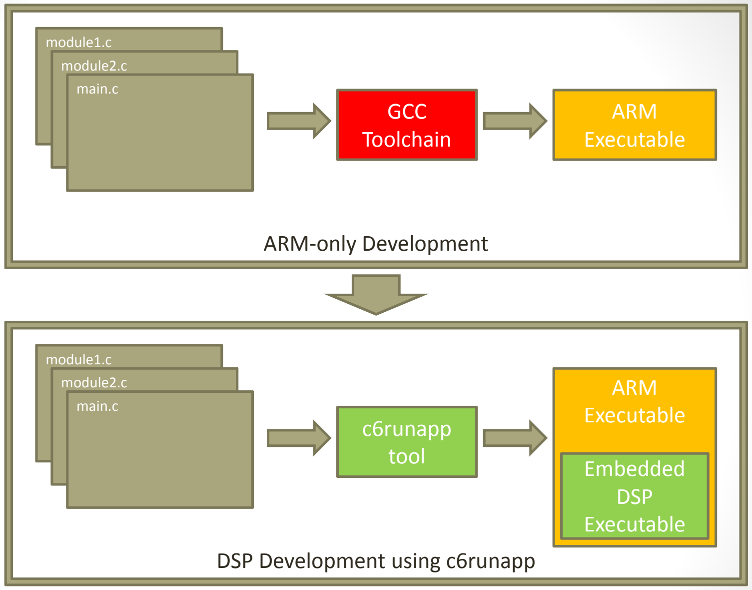
```
$ c6runlib-ar rcs dummy_dsp.lib dummy.o
```

- Add object file to library dummy_dsp.lib
- Underneath, the dummy.o object file is linked to a DSP executable and compiled into the framework
- Framework object file and stubs object file is archived into the lib
- ARM-side stubs resolve symbols for ARM application when built against the library

C6Run**App**

- Entire application retargets to DSP, but with C I/O access to ARM/Linux



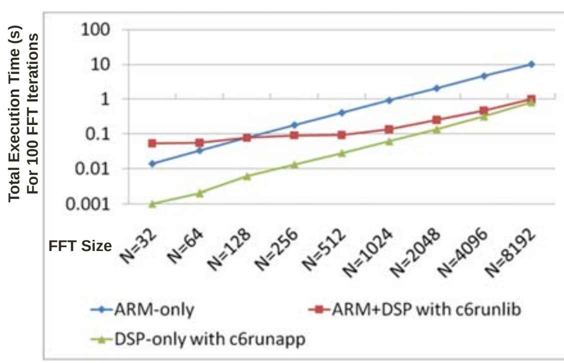


Example C6RunApp Usage

```
$ c6runapp-cc -o hello_world hello_world.c
```

- Compiles hello_world.c to C6000 object file, which is then linked into a DSP executable
- Executable is compiled into the ARM side framework, which is used to build an ARM-side executable called hello_world
- Notes on the c6runapp-cc cross-compiler tool:
  - Front end script wraps the TI C6000 Codegen tools (specifically cl6x)
  - Supports many GCC options and translates them to appropriate cl6x options
  - Many GCC optimization/warning options are silently ignored
  - Unknown options passed directly to cl6x command-line

Complex FFT Performance



- FFT runs ~10x faster on DSP than on ARM
- Small FFT size, overhead dominates, running on DSP does not provide advantage
- Larger FFT size, overhead absorbed, running on DSP provides advantage

SoC: ARM9 + Floating-Point DSP
CPU Frequency: 300MHz
Code & Data Location: External DDR2 Memory
Instruction and Data Cache: Enabled
Single-precision floating-point data buffers.

c6run Hands on

- ~~C6run must compile on a host computer~~
- Once compiled, binaries are copied to the Beagle
- Try:

```
$ cd /c6run_build
$ ./unloadmodules.sh
$ ./loadmodules.sh
$ ./environment.sh
$ cd examples/c6runapp/hello_world/
$ ./hello_world_arm
Hello SPEd world, from C6RunApp!
$ ./hello_world_dsp
Hello SPEd world, from C6RunApp!
```

c6run Hands on

- Explore the other examples

```
./c6runapp:
cio_example  emqbit  hello_world
./c6runapp/cio_example:
cio_example.c  cio_example_arm   cio_example_dsp
./c6runapp/emqbit:
bench_arm  bench_dsp  cfft.c  cfft.h  cfft_arm    cfft_dsp
./c6runapp/hello_world:
hello_world.c  hello_world_arm   hello_world_dsp
./c6runlib:
emqbit
./c6runlib/emqbit:
bench_arm      bench_dsp      cfft.c  cfft_arm        cfft_dsp
bench_arm.lib  bench_dsp.lib  cfft.h  cfft_arm.lib  cfft_dsp.lib
```

Details – cd c6run_target/examples/c6runlib/emqbit

```
# ./cfft_arm
N=16,nTimes=100: 0.000458 s
N=32,nTimes=100: 0.00119 s
N=64,nTimes=100: 0.004486 s
N=128,nTimes=100: 0.00708 s
N=256,nTimes=100: 0.018005 s
N=512,nTimes=100: 0.034149 s
N=1024,nTimes=100: 0.076507 s
N=2048,nTimes=100: 0.1651 s
N=4096,nTimes=100: 0.364411 s
N=8192,nTimes=100: 0.79776 s
N=16384,nTimes=100: 1.72101 s
```

```
# ./cfft_dsp
N=16,nTimes=100: 0.015533 s
N=32,nTimes=100: 0.022827 s
N=64,nTimes=100: 0.036712 s
N=128,nTimes=100: 0.07547 s
N=256,nTimes=100: 0.154419 s
N=512,nTimes=100: 0.331909 s
N=1024,nTimes=100: 0.731232 s
N=2048,nTimes=100: 1.59704 s
N=4096,nTimes=100: 3.48624 s
N=8192,nTimes=100: 7.62561 s
N=16384,nTimes=100: 16.6948 s
```

Details – main_cfft.c

```
int main () {
…
  for (N = (1 << MINPOW2), n = 0; N < (1
<< MAXPOW2); N = N << 1, n++)
  {
    complex *in = new_complex_vector(N);
    complex *out = new_complex_vector(N);
    fft_init (N);
    // Copy input data and do one FFT
    memcpy (out, in, (N) * …);
    fft_exec (N, out);
    nTimes = ITERATIONS;
    t0 = get_timestamp();
    for (i = 0; i < nTimes; i++) {
      memcpy (out, in, (N) * …);
      fft_exec (N, out);
    }
    t1 = get_timestamp();
    secs = (t1 - t0) / 1000000.0L;

    free (in);
    free (out);
    fft_end ();

    fprintf (stderr,
"N=%d,nTimes=%d: %g s\n", N,
nTimes, secs);
  }

  return 0;
}
```

c6run_build/examples/c6runlib/emqbit

Details – main_cfft.c

```
static complex *new_complex_vector(int size) {
  int i;
  complex *new;
  new = (complex *) malloc(sizeof(complex) * size);
  for(i = 0; i < size; ++i)
  {
    new[i].r = (float)rand()/(float)RAND_MAX - 0.5;
    new[i].i = (float)rand()/(float)RAND_MAX - 0.5;
  }
  return new;
}
```

Details – cfft.c

```
void fft_init (int N) {
…
  tableW = malloc ((N / 2) * sizeof (complex));
  bndx = malloc (N * sizeof (int));
  ndx = malloc ((N / 2) * sizeof (int));

  ndx[0] = 0;
  for (i = 1; i < N / 2; i = i * 2)
  {…}
  bndx[0] = 0;
  for (i = 1; i < N; i = i * 2)
  {…}
  for (i = 0; i < N / 2; i++) {
    tableW[i].r = cos (ndx[i] * 2.0F * M_PI / (float) N);
    tableW[i].i = -sin (ndx[i] * 2.0F * M_PI / (float) N);
  }
}
```

Details – cfft.c

```
void fft_exec (int N, complex * in) {
  unsigned int n = N;
  unsigned int a, b, i, j, k, r, s;
  complex w, p;
  for (i = 1; i < N; i = i * 2)  {
    n = n >> 1;
    for (k = 0; k < i; k++)    {
      w = tableW[k];
      r = 2 * n * k;
      s = n * (1 + 2 * k);
      for (j = 0; j < n; j++) {
        a = j + r;
        b = j + s;
        cmult (p, w, in[b]);      //6 flop
        csub (in[b], in[a], p);   //2 flop
        cadd (in[a], in[a], p);   //2 flop
      }
    }
  }
}
```

Details – main_cfft.c

```
int main () {
…
  for (N = (1 << MINPOW2), n = 0; N < (1
<< MAXPOW2); N = N << 1, n++)
  {
    complex *in = new_complex_vector(N);
    complex *out = new_complex_vector(N);
    fft_init (N);
    // Copy input data and do one FFT
    memcpy (out, in, (N) * …);
    fft_exec (N, out);
    nTimes = ITERATIONS;
    t0 = get_timestamp();
    for (i = 0; i < nTimes; i++) {
      memcpy (out, in, (N) * …);
      fft_exec (N, out);
    }
    t1 = get_timestamp();
    secs = (t1 - t0) / 1000000.0L;

    free (in);
    free (out);
    fft_end ();

    fprintf (stderr,
"N=%d,nTimes=%d: %g s\n", N,
nTimes, secs);
  }

  return 0;
}
```

Where's the DSP?

- **`malloc`** is overridden to use **`cmem`**
- **`Makefile`** decides what is run on the DSP and ARM

Details – Makefile – ARM

```
ARM_TOOLCHAIN_PREFIX  ?= arm-none-linux-gnueabi-
ifdef ARM_TOOLCHAIN_PATH
 ARM_CC := $(ARM_TOOLCHAIN_PATH)/bin/$(ARM_TOOLCHAIN_PREFIX)gcc
 ARM_AR := $(ARM_TOOLCHAIN_PATH)/bin/$(ARM_TOOLCHAIN_PREFIX)ar
else
 ARM_CC := $(ARM_TOOLCHAIN_PREFIX)gcc
 ARM_AR := $(ARM_CROSS_COMPILE)ar
endif
# Get any compiler flags from the environment
ARM_CFLAGS = $(CFLAGS)
ARM_CFLAGS += -std=gnu99 \
 -Wdeclaration-after-statement -Wall -Wno-trigraphs \
 -fno-strict-aliasing -fno-common -fno-omit-frame-pointer \
 -c -O3
ARM_LDFLAGS = $(LDFLAGS)
ARM_LDFLAGS+=-lm -lpthreadARM_ARFLAGS = rcs
```

Details – Makefile – 'C6x

```
C6RUN_TOOLCHAIN_PREFIX=c6runlib-
ifdef C6RUN_TOOLCHAIN_PATH
  C6RUN_CC :=
       $(C6RUN_TOOLCHAIN_PATH)/bin/$(C6RUN_TOOLCHAIN_PREFIX)cc
  C6RUN_AR :=
       $(C6RUN_TOOLCHAIN_PATH)/bin/$(C6RUN_TOOLCHAIN_PREFIX)ar
else
  C6RUN_CC := $(C6RUN_TOOLCHAIN_PREFIX)cc
  C6RUN_AR := $(C6RUN_TOOLCHAIN_PREFIX)ar
endif

C6RUN_CFLAGS = -c -O3
C6RUN_ARFLAGS = rcs --C6Run:replace_malloc
```

Sharing Memory

- The ARM and the DSP share memory
  - Have to pass pointers to the buffers
- No need to copy from DSP to ARM
  - Very little overhead
- ARM uses a memory management unit (MMU)
  - maps virtual addr to physical addr
- DSP doesn't have an MMU!

Sharing Memory

- ARM pointers (*outputBuffer*, *inputBuffer*) are virtual addresses
- DSP pointers (*outputBuffer*, *inputBuffer*) are physical addresses
- C6Run automatically provides the code needed to map from the virtual address space to the physical

Sharing Memory

- Bigger problem
  - *outputBuffer* and *inputBuffer* are allocated at run time using the standard C routine **malloc**
  - malloc allocates contiguous memory of the desired size
- Contiguous in the *virtual* space
  - probably not contiguous in the *physical* space
- Causes problems for the DSP

Sharing Memory

- Solution:
  - loader flag **`--C6Run:replace_malloc`**
  - Tells the loader to replace malloc with cmem
- **cmem** is an API and library for managing one or more blocks of physically contiguous memory
- Provides address translation services
  - e.g. virtual to physical translation

Sharing Memory

- If you are uncomfortable with replacing all **`mallocs`** with **`cmem`**
  - Remove **`--C6Run:replace_malloc`**
  - Call

`C6RUN_MEM_malloc(N * sizeof(short))`

when you have memory to share with the DSP

cmem

- Where does cmem allocate the memory?
  - A section of the Beagle's RAM that Linux doesn't control

```
$ cat /proc/cmdline
console=tty0 console=ttyS2,115200n8
consoleblank=0 mpurate=auto buddy=none
camera=lbcm3m1 vram=24M omapfb.mode=dvi:hd720
mem=99M@0x80000000 mem=384M@0x88000000
omapfb.vram=0:12M,1:8M,2:4M
omapdss.def_disp=dvi root=/dev/mmcblk0p2 rw
rootfstype=ext3 rootwait
```

cmem

- 99M bytes start at 0x8000 0000 and 384M start at 0x8800 0000
- Where does the first block of addresses end?

```
$ bc
obase=16
99*1024*1024        6300000
ibase=16
6300000+80000000    86300000
```

- Block ends at 0x8630 0000

cmem

- How does cmem know that?
- Look in ~/c6run_build/loadmodules.sh

```
DSP_REGION_START_ADDR="0x86300000"
DSP_REGION_END_ADDR="0x88000000"
CMEMK_OPTS="phys_start=$DSP_REGION_STAR
  T_ADDR phys_end=$DSP_REGION_END_ADDR
  allowOverlap=1"
modprobe cmemk ${CMEMK_OPTS}
```

cmem

- How does C6Run know what memory is used?
  - Look in ~/c6run_build/environment.sh
  - At the bottom you'll see:

```
PLATFORM_CFLAGS='
-DDSP_REGION_BASE_ADDR=0x86300000
-DDSP_REGION_CMEM_SIZE=0x01000000
-DDSP_REGION_CODE_SIZE=0x00D00000
-DLPM_REQUIRED
-DDSP_HAS_MMU'
```

Looking inside c6run

- Edit Makefile
  - add **--C6Run:debug** to the FLAGS:

```
C6RUN_CFLAGS = -c -O3 -D_DEBUG_ --C6Run:debug
C6RUN_ARFLAGS = rcs --C6Run:replace_malloc --C6Run:debug
```

- Then recompile everything:

```
$ make dsp_clean
$ make dsp_exec
$ ls -sh
```

- Many interesting files appear

ARM vs. DSP

- How do I specify what runs on the ARM and what runs on the DSP?
- **`Makefile`**
  - `main_cfft.c` is run on the ARM
  - `cfft.c` is run on the DSP

```
EXEC_SRCS := main_cfft.c
EXEC_DSP_OBJS := $(EXEC_SRCS:%.c=dsp/%.o)
```

Details – make dsp_exec

```
dsp_exec: dsp/.created dsp_lib $(EXEC_DSP_OBJS)
      $(ARM_CC) $(ARM_LDFLAGS) $(CINCLUDES) -o cfft_dsp \
                               dsp/main_cfft.o cfft_dsp.lib
dsp_lib: dsp_lib/.created $(LIB_DSP_OBJS)
      $(C6RUN_AR) $(C6RUN_ARFLAGS) cfft_dsp.lib dsp_lib/cfft.o
dsp/%.o : %.c
      $(ARM_CC) $(ARM_CFLAGS) $(CINCLUDES) -o $@ $<
dsp_lib/%.o : %.c
      $(C6RUN_CC) $(C6RUN_CFLAGS) $(CINCLUDES) -o $@ $<
dsp/.created:
      @mkdir -p dsp
      @touch dsp/.created
dsp_lib/.created:
      @mkdir -p dsp_lib
      @touch dsp_lib/.created
```

```
EXEC_SRCS := main_cfft.c
EXEC_DSP_OBJS := $(EXEC_SRCS:%.c=gpp/%.o)
```

Details – make gpp_exec

```
gpp_exec: gpp/.created gpp_lib $(EXEC_ARM_OBJS)
      $(ARM_CC) $(ARM_LDFLAGS) $(CINCLUDES) -o cfft_arm \
                               gpp/main_cfft.o cfft_arm.lib
gpp_lib: gpp_lib/.created $(LIB_ARM_OBJS)
      $(ARM_AR) $(ARM_ARFLAGS) cfft_arm.lib gpp_lib/cfft.o
gpp/%.o : %.c
      $(ARM_CC) $(ARM_CFLAGS) $(CINCLUDES) -o $@ $<
gpp_lib/%.o : %.c
      $(ARM_CC) $(ARM_CFLAGS) $(CINCLUDES) -o $@ $<
gpp/.created:
      @mkdir -p gpp
      @touch gpp/.created
gpp_lib/.created:
      @mkdir -p gpp_lib
      @touch gpp_lib/.created
```

Conclusions

- Using the DSP is easy as
  - Moving the functions to run on the DSP into their own files
  - Editing the Makefile to show what runs where