Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Configuring the C/C++ Run-Time Header for Blackfin® Processors

Contributed by Steve K

Rev 1 – May 18, 2004

Introduction

The C/C++ run-time header (CRT) is the library code that is executed when the processor jumps to the `start` address on reset. The CRT sets the machine into a known state and calls `_main`[1]. This EE-Note describes the Blackfin® processor's CRT, and explains how to rebuild or reconfigure it.

The document begins with an overview of the CRT, which describes what the CRT does and does not do. Each CRT activity is then examined, followed by configuration instructions.

CRT Overview

The CRT:

- Sets up required system services, such as stack and event-handling
- Disables system services that power up in an undefined state, such as circular buffers and hardware loops
- Initializes standard library facilities, such as standard input/output (`stdio`)
- Invokes C++ constructors, as required

The CRT ensures that when execution enters `_main`, the processor's state obeys the C Application Binary Interface (ABI), and that global data declared by the application have been initialized as required by the C/C++ standards. It arranges things so that `_main` appears to be "just another function" invoked by the normal function invocation procedure.

Not all applications require the same configuration. For example, C++ constructors are invoked only for applications that contain C++ code. The list of optional configuration items is long enough that determining whether to invoke each one in turn at runtime would be overly costly. As a convenience, the Blackfin processor's CRT is supplied pre-built in several different configurations, which can be specified at link-time via LDF macros.

The CRT is used for projects that use C or C++. Projects configured with VDK use a different run-time header, not described here (although the principles are the same). Assembly language projects do not provide a default run-time header; you must provide your own.

The source for the Blackfin processor's CRT is located under the VisualDSP++® installation directory, in the file `basiccrt.s`, in the directory `Blackfin\lib\src\libc\crt`.

The list of operations performed by the CRT can include (not necessarily in the following order):

- Setting registers to known/required values
- Disabling hardware loops
- Disabling circular buffers
- Setting up default event handlers and enabling interrupts

[1] In this document, all symbols are written with assembly linkage, except where otherwise noted. Thus, `_main` is used, rather than `main()`.

- Initializing the Stack and Frame Pointers
- Enabling the cycle counter
- Configuring the memory ports used by the two DAGs
- Setting the processor clock speed
- Copying data from flash to RAM
- Initializing device drivers
- Setting up memory protection and caches
- Changing processor interrupt priority
- Initializing profiling support
- Invoking C++ constructors
- Invoking `_main`, with supplied parameters
- Invoking `_exit` on termination

What the CRT Does Not Do

The CRT does not set up physical memory hardware. For example, external SDRAM is not configured by the CRT in any way. This is left to the boot loader because it is possible (and even likely) that the CRT itself will need to be moved into external memory before being executed.

The CRT in Detail

The CRT is provided as a single assembly source file. It is assembled different ways to serve different types of applications, running on different processors. One source file is used to generate CRTs for the following devices:

- ADSP-BF531 Blackfin processors
- ADSP-BF532 Blackfin processors
- ADSP-BF533 Blackfin processors
- ADSP-BF535 Blackfin processors
- ADSP-BF561 Blackfin processors
- AD6532 processors

Declarations

The CRT begins with a number of preprocessor directives that "include" the appropriate platform-definition header and set up a few constants:

`IVBl` and `IVBh` give the address of the Event Vector Table.

`UNASSIGNED_VAL` is a bit pattern that indicates that the register/memory location has not yet been written to by the application. This value forms a marker at the end of the call stack, and, if `UNASSIGNED_FILL` is non-zero, will be written into the Dregs and Pregs.

`INTERRUPT_BITS` is the default interrupt mask. By default, it enables the lowest-priority interrupt, `IVG15`. This default mask can also be overridden at runtime by your own version of `__install_default_handlers`; see the Default Event Handlers section for details.

For some platforms, `SYSCFG_VALUE` is the initialization value for the System Configuration register (`SYSCFG`).

`SET_CLOCK_SPEED` determines whether the CRT will attempt to improve the processor's default clock speed. For ADSP-BF561 Blackfin processors, ADSP-BF535 Blackfin processors, and AD6532 processors, the clock speed is not altered. For other processors, clock speed depends on the silicon revision; a number of accompanying definitions follow which may be used to change the clock speed up to approximately 600 MHz.

`start` and Register Settings

The CRT declares its first code label as `start`. This required label is referenced by `.LDF` files, which explicitly resolve this label to the processor's reset address.

First, the CRT disables facilities that could be enabled on start-up, due to their random power-up states, as follows:

- `SYSCFG` is set to `SYSCFG_VALUE`, according to anomaly #42 for ADSP-BF532 Blackfin processors, anomaly #22 for ADSP-BF531 and ADSP-BF533 Blackfin processors, and

anomaly #12 for ADSP-BF561 Blackfin processors.

- Hardware loops are disabled. This prevents the jump-back-to-loop-start behavior should the "loop bottom" register correspond to the start of an instruction.
- Circular buffer lengths are set to zero. The CRT makes use of the Iregs and calls functions that may use them. Furthermore, the C/C++ ABI requires that circular buffers are disabled on entry to (and exit from) compiled functions, so the circular buffers must be disabled before invoking `_main`.

Event Vector Table

The Event Vector Table is cleared. No event handlers are defined except the reset vector (fixed), emulation events (not touched by the C ABI), and `IVG15`. The processor's lowest-priority event, `IVG15`, is set to point to `supervisor_mode`, a label that appears later in the CRT.

The `___cplb_ctrl` control variable is examined to determine whether caching or memory protection is required. If it is, an exception handler is required to process possible events raised by the memory system. Therefore, the default handler, `___cplb_hdr`, is installed into the exception entry of the Event Vector Table.

For details on `___cplb_ctrl`, refer to "Caching and Memory Protection" in the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for Blackfin Processors* [1].

Stack Pointer and Frame Pointer

The Stack Pointer (`SP`) is set to point to the top of the stack, as defined in the `.LDF` file by the symbol `ldf_stack_end`. Specifically, the Stack Pointer is set to point just past the top of the stack. Because stack pushes are pre-decrement operations, the first push first moves the Stack Pointer so that it refers to the actual stack top.

The User Stack Pointer (`USP`) and Frame Pointer (`FP`) are set to point to the same address.

Twelve bytes are then claimed from the stack. This is because the C ABI requires callers to allocate stack space for the parameters of callees, and that all functions require at least twelve bytes of stack space for registers `R0-R2`. Therefore, the CRT claims these twelve bytes as the incoming parameters for functions called before invoking `_main`.

Default Event Handlers

The CRT sets up one or two event handlers (one for changing processor mode; optionally, another for handling memory system events), as reflected by the event-enable bit mask. You may install additional handlers; for your convenience, the CRT calls a function to do this. The function, `__install_default_handlers`, is an empty stub, which you may replace with your own function that installs additional or alternative handlers, before the CRT enables events.

The function's C prototype is:

```
short _install_default_handlers(short mask);
```

The CRT passes the default enable mask, (`INTERRUPT_BITS`) as a parameter, and considers the return value to be an updated enable mask; if you install additional handlers, you must return an updated enable mask to reflect this.

Cycle Counter

The CRT enables the cycle counter, so that the `CYCLES` and `CYCLES2` registers are updated. This is not necessary for general program operation, but is desirable for measuring performance.

DAG Port Selection

For ADSP-BF531, ADSP-BF532, ADSP-BF533, and ADSP-BF561 Blackfin processors, the CRT configures the DAGs to use different ports for accessing memory. This reduces stalls when the DAGs issue memory accesses in parallel.

Clock Speed

If `SET_CLOCK_SPEED` is set to one, the CRT changes the default clock speed from approximately 300 MHz to approximately 600 MHz, provided the silicon is of a suitable revision. This is determined by reading the `DSPID` memory-mapped register and ensuring that the silicon version is 0.2 or greater.

To change the clock speed, the CRT uses two library routines with the following C prototypes:

```
int pll_set_system_vco(int msel, int df, int
lockcnt)

int pll_set_system_clocks(int csel, int
ssel)
```

These two routines return zero for success and negative for error, but the CRT does not test their return values.

EZ-KIT Lite™ boards use a 27 MHz on-board clock. The CRT uses the following constants as parameters to these routines to set the PLL and clocks:

- VCO = 27*0x16 = 594 MHz
- CCLK = VCO = 594 MHz
- SCLK = VCO/5 = 118 MHz

Memory Initialization

Memory Initialization is a two-stage process:

1. At link-time, the Memory Initializer utility processes the .DXE file to generate a table of code and data memory areas that must be initialized during booting.
2. At runtime, when the application starts, the run-time library function `_mi_initialize` processes the table to copy the code and data from the Flash device to volatile memory.

If the application has not been processed by the Memory Initializer, or if the Memory Initializer did not find any code or data that required such movement, the `_mi_initialize` function returns immediately.

The CRT does not enable external memory. The configuration of physical memory hardware is the responsibility of the boot loader and must be complete before the CRT is invoked.

Device Initialization (`FIOCRT`)

The CRT calls `_init_devtab` to initialize device drivers that support `stdio`. This routine:

- Initializes the internal file tables
- Invokes the `init` routine for each device driver registered at build-time
- Associates `stdin`, `stdout`, and `stderr` with the default device driver

Device initialization is an optional part of the CRT; it is only performed when the CRT is assembled with `FIOCRT` defined.

For information on the device drivers supported by `stdio`, refer to "Extending I/O Support to New Devices" in the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for Blackfin Processors* [1].

CPLB Initialization

If instruction or data Cache Protection Lookaside Buffers (CPLBs) are to be enabled, the CRT calls `_cplb_init` to do this, passing the value of `___cplb_ctrl` as a parameter. For details on CPLB initialization, refer to "Caching and Memory Protection" in the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for Blackfin Processors* [1].

Enable Interrupts

The interrupt-enable bit mask returned from `__install_default_handlers` (but defaulting to `INTERRUPT_BITS`) is used to enable interrupts. At this point, the processor is still operating at Reset priority, so pending events are still blocked from occurring.

Lower Processor Priority

The CRT lowers the process priority to the lowest supervisor mode level (IVG15). It first raises IVG15 as an event, but this event cannot be serviced while the processor remains at the higher priority level of Reset. The CRT sets RETI to be the label usermode, which is an empty infinite loop, and then does a "return" from the Reset interrupt (i.e., it executes an RTI instruction).

The processor mode changes to user mode and recommences execution at the RETI address (i.e., the usermode label, which also happens to be the following instruction). However, before this infinite loop starts executing, the IVG15 event, which is pending, is taken; now that the processor level has dropped, the IVG15 event is allowed to proceed.

The handler for IVG15 was set up earlier in the CRT and is the label supervisor_mode, which follows immediately after the infinite usermode loop. Thus, execution flows from the "return" from Reset level to the supervisor_mode label, while changing processor mode from the highest supervisor level to the lowest supervisor level.

If other events are enabled (memory system exceptions or other events installed via your own version of the default handlers stub), they could be taken between the return from Reset and entering IVG15. Therefore, the remaining parts of the CRT may not execute when event handlers are triggered.

The CRT's first action after entering IVG15 is to re-enable the interrupt system so that other higher-priority interrupts can be processed.

Mark Registers (UNASSIGNED_FILL)

If UNASSIGNED_FILL is non-zero, the R2-R7 and P0-P5 registers are set to UNASSIGNED_VAL.

Terminate Stack Frame Chain

Each stack frame is pointed to by the Frame Pointer and contains the previous values of the Frame Pointer and RETS. The CRT pushes two instances of UNASSIGNED_VAL onto the stack, indicating that there are no further active frames. The C++ exception support library uses these markers to determine whether it has walked back through all active functions without finding one with a catch for the thrown exception.

Again, the CRT allocates twelve bytes for outgoing parameters of functions that will be called from the CRT.

Profiler Initialization (PROFCRT/PROFGUIDE)

If PROFCRT is defined, the instrumented-code profiling library is initialized by calling monstartup. This routine zeroes all counters and ensures that no profiling frames are active. Instrumented-code profiling is specified with the -p, -p1, and -p2 compiler switches. If any of the object files have been compiled to include this profiling, the prelinker will detect this and will set link-time macros to include a profiler-enabled version of the CRT.

The instrumented-code profiling library uses stdio routines to write the accumulated profile data to stdout or to a file. If PROFCRT is defined, FIOCRT must also be defined. The default .LDF files include code for this.

If PROFGUIDE is defined, the current version of the CRT has a call to ___start_prof. This is legacy code, and PROFGUIDE should not be defined; it will be removed in a later revision. (Profile-Guided Optimization is implemented within the simulators, and not within the run-time libraries.)

C++ Constructor Invocation (CPLUSCRT)

If CPLUSCRT is defined, ___ctorloop is invoked to run all of the global-scope constructors.

Each global-scope constructor has a pointer to it located in the `ctor` section. When all of these sections are gathered together during linking, they form a table of function pointers. The `___ctorloop` library function walks the length of this table, invoking each constructor in turn. The `.LDF` file arranges for the special object `crtn.doj` to be linked after the CRT; it contains a terminator for the ctor table.

Argument Parsing (`FIOCRT`)

If `FIOCRT` is defined, `__getargv` is called to parse any provided arguments (normally an empty list) into the global array, `__Argv`. This function returns the number of arguments found which, along with `__Argv`, forms the `argc` and `argv` parameters for `_main`. If `FIOCRT` is not defined, `argc` is set to zero and `argv` is set to an empty list, statically defined within the CRT.

Calling `_main` and `_exit`

`_main` is called, using the `argc` and `argv` just defined. Embedded programs are not expected to return from `_main`, but many legacy and non-embedded programs do. Therefore, the return value from `_main` is immediately passed to `_exit` to gracefully terminate the application. `_exit` is not expected to return.

Configuring the CRT

Available CRT Options

The preceding description notes where the CRT conditionally executes some operations. These can be divided into build-time and run-time options.

The build-time options are:

- `FIOCRT`: initializes device drivers and parses command-line arguments
- `CPLUSCRT`: invokes global-scope C++ constructors
- `PROFCRT`: initializes instrumented-code profiling (requires `FIOCRT`)
- `UNASSIGNED_FILL`: fills registers with a default "unused" pattern
- `SET_CLOCK_SPEED`: raises clock speed to 600 MHz, if appropriate
- `_WORKAROUNDS_ENABLED`: enables silicon-revision anomalies[2]
- Processor-specific operations

These options are set in different permutations to produce the various CRTs supplied in pre-assembled form. `FIOCRT`, `CPLUSCRT`, `PROFCRT`, and `_WORKAROUNDS_ENABLED` are indicated as having been set during assembly by filename suffices "f", "c", "p", and "y", respectively. Thus, `crtsc532.doj` has been assembled with `CPLUSCRT`, for ADSP-BF532 Blackfin processors, and `crtsfp535.doj` has been assembled with `FIOCRT` and `PROFCRT` for ADSP-BF535 Blackfin processors. The "s" suffix indicates that the processor invokes `_main` in supervisor mode, and is historical; the two CRTs `crtsc532.doj` and `crtc532.doj` are identical.

`UNASSIGNED_FILL` is not defined for any of the pre-assembled CRT objects.

The run-time options are:

- Install exception handler for memory protection system exceptions
- Set the clock speed if the processor is of the appropriate silicon revision
- Install CPLBs and initialize the memory protection system and any caching

Since `SET_CLOCK_SPEED` is pre-defined to be zero or one according to processor type (build-time) and the silicon revision is checked before proceeding (run-time), changing the clock speed falls mid-way between build-time and run-time options.

[2] Not used in the CRT for the VisualDSP++ 3.5 release.

The default .LDF files select the CRT to be used, according to various link-time macros. Defining the macro at link-time directs the .LDF file to select a CRT that was assembled with the corresponding macro defined. Table 1 shows this relationship.

This LDF macro defined at link-time	Selects CRT that had this macro defined at assembly-time
USE_FILEIO	FIOCRT
USE_PROFILER	PROFCRT
USE_PROFILER0	PROFCRT
USE_PROFILER1	PROFCRT
USE_PROFILER2	PROFCRT
__cplusplus	CPLUSCRT
__WORKAROUNDS_ENABLED	_WORKAROUNDS_ENABLED

Table 1. Relationship Between LDF Macros and CRT Macros

There is some reliance on convention here. For example, an object compiled with compiler switch -p will contain calls to the instrumented-code profiling library. Therefore, it will need a CRT assembled with PROFCRT defined. The prelinker will detect this, and will specify the USE_PROFILER0 macro at link-time. This process assumes that the .LDF file will select an appropriate CRT as a result.

All default .LDF files specify FIOCRT, even if USE_FILEIO is not supplied, so that the stdio device driver necessary for _printf will be initialized.

Modifying the CRT Configuration

To produce your own modified version of the CRT:

1. Copy basiccrt.s from Blackfin\lib\src\libc\crt in the VisualDSP++ installation directory, and add it to your project.
2. Modify the copy, as necessary.
3. If the project uses C++, add -DCPLUSCRT=1 to the options for this file.
4. If the project uses instrumented-code profiling, add -DPROFCRT=1 to the options for this file.
5. If the project uses the default .LDF file, use the Expert Linker to create a custom .LDF file for the project.
6. Open the .LDF file for editing in a source window.
7. Modify the line that defines $OBJECTS, to not mention CRT. For example, if using the ADSP-BF533_C.ldf file, change it from:

```
$OBJECTS = CRT, $COMMAND_LINE_OBJECTS
  ,cplbtab533.doj ENDCRT;
```

to:

```
$OBJECTS = $COMMAND_LINE_OBJECTS
  ,cplbtab533.doj ENDCRT;
```

When you rebuild the project, the local copy of the CRT will be assembled and linked in with the project as one of the $COMMAND_LINE_OBJECTS, and the standard version of the CRT will not be linked in (which would cause multiply-defined symbol errors).

References

[1] *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for Blackfin Processors*. Rev. 2.2. October 2003. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – May 18, 2004 *by Steve K.*	Initial Release