ANALOG DEVICES

Technical Notes on using Analog Devices' DSP components and development tools
Contact our technical support by phone: (800) ANALOG-D or e-mail: dsp.support@analog.com
Or visit our on-line resources http://www.analog.com/dsp and http://www.analog.com/dsp/EZAnswers

ADSP-TS101S TigerSHARC® Processor Boot Loader Kernels Operation

Contributed by Boris Lerner

April 01, 2003

Introduction

This EE-note explains the functional operation of the power-on booting procedure and the boot loader kernels for the ADSP-TS101S TigerSHARC® processor. A loader kernel is a program executed by the DSP that is appended to your application code by the linker utility (linker.exe) of the VisualDSP++™ development tools which is executed by the DSP at boot time to allow the processor to initialize its internal and external memory sections defined in the application code.

The loader kernel is a self-modifying program which is transferred into the DSP's internal memory. The ADSP-TS101 supports three booting methods; EPROM booting (via the external port), host booting (via an external host processor or another TigerSHARC), or link booting (via the DSP's link ports.) Therefore, there are three distinct loader kernels to support each of the processor's booting modes.

Booting Procedure For The ADSP-TS101S

The booting mode is selected by the /BMS pin of the DSP. While the processor is held in reset, the /BMS pin is an active input. If /BMS is sampled low during reset, EPROM boot mode is selected; after the /RESET signal of the DSP is de-asserted, the /BMS pin becomes an output acting as the EPROM chip select. If /BMS is sampled high during reset, the TigerSHARC will be in an IDLE state, waiting for a host boot or a link port boot to occur.

Additionally, there is a weak internal pull-down resistor on the /BMS pin, but it is important to note here that the pull-down may not be sufficient, depending upon the external line loading on this pin. Thus, an external pull-down resistor may be necessary to select EPROM booting mode. If host or link boot is desired, /BMS must be held high during reset and may be tied directly to VCC.

Each booting method is described in detail in the following sections.

EPROM Boot

When EPROM boot mode is selected, the TigerSHARC initializes its external port DMA channel 0 to transfer 256 32-bit words of code from the boot EPROM into the TigerSHARC's internal memory block 0, locations 0x00-0xFF. The corresponding interrupt vector (for DMA channel 0) is initialized to 0. Thus, upon completion of the DMA, the TigerSHARC continues its program execution from location 0x00. It is intended that these 256 words of code act as a boot loader to initialize the rest of the TigerSHARC's memory. Analog Devices provides a default boot loader kernel source file with the VisualDSP++ development tools, called "TS101_prom.asm", which can be used as a reference.

The default EPROM boot loader works in conjunction with the loader utility (elfloader.exe) supplied with the VisualDSP++ tools. The loader utility takes the user's executable file (*.dxe*) from their project and the boot loader executable file (default: *TS101_prom.dxe*) and produces the EPROM loader output file (*.ldr*). This loader output file defines how the various blocks of TigerSHARC's internal and external memory are to be initialized during the booting process. Its format is described in figure 1. The format of the block tag word is described in figure 2.

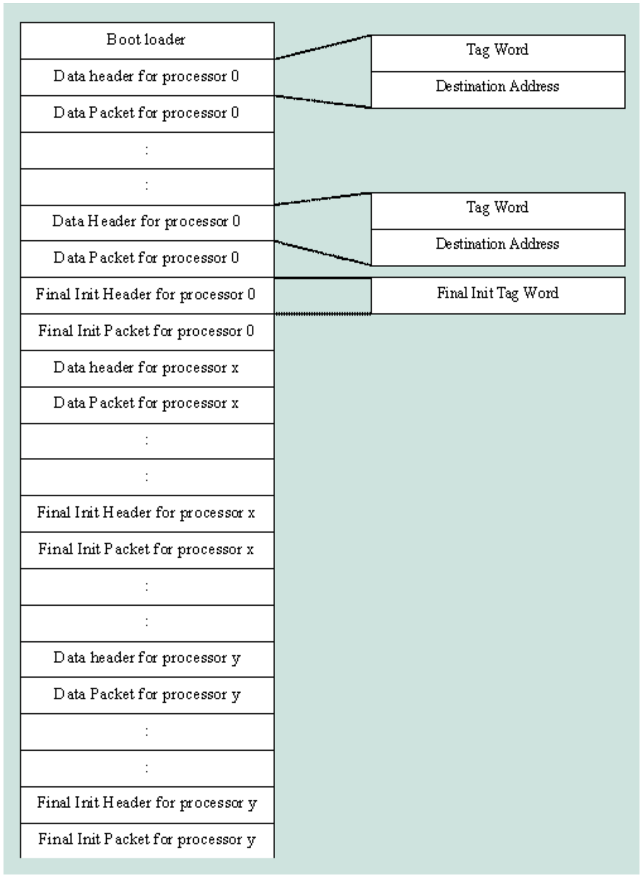
Boot loader
Data header for processor 0
Data Packet for processor 0
:
:
Data Header for processor 0
Data Packet for processor 0
Final Init Header for processor 0
Final Init Packet for processor 0
Data header for processor x
Data Packet for processor x
:
:
Final Init Header for processor x
Final Init Packet for processor x
:
:
Data header for processor y
Data Packet for processor y
:
:
Final Init Header for processor y
Final Init Packet for processor y
Tag Word
Destination Address
Tag Word
Destination Address
Final Init Tag Word

Figure 1 EPROM Loader Format

31:30	29:27	26:16	15:0
TYPE	ID	RESERVED	COUNT

Type:	0=Final init, 1=Non-zero init, 2=Zero Init
ID:	ID of the processor to which the block belongs
COUNT:	Number of 32-bit words in the block

Figure 2 Tag Word Format

The supplied boot loader (*TS101_prom.dxe*) operates as described below:

1. After the boot loader kernel is loaded into the internal memory block 0 of the DSP, the DMA0 interrupt exits the TigerSHARC from its idle state. The DSP then begins execution of the boot loader at location 0x00000000. At this stage, the TigerSHARC is at the interrupt level of DMA0 and, thus, further DMA0 and global (PMASK[60]) interrupts are disabled.
2. The boot loader kernel sets the NMOD bit in the SQCTL register to ensure that the DSP will be running in supervisor mode. An RDS instruction reduces the current interrupt to a subroutine level. Next, DMA0 and global interrupts are enabled again.
3. DMA0 is configured to move data from the boot EPROM starting at address 0x0400 of the EPROM (0x0000-0x03ff was the boot loader) to the DSP's internal memory starting at address 0x00000000. The DMA routine will start the DMA by programming the TCBs, advance the prom pointer and sit in IDLE until the DMA interrupt wakes it up and sends the program sequencer to point to the DMA0 interrupt vector. There, an RTI instruction returns to the DMA routine, which in turn, returns to continue execution of the boot loader kernel.
4. Since this is *not* a link port boot, all of the link port DMA channel control registers are reset and all of the link port DMAs are disabled.
5. The processor ID (for this processor) is computed and stored in register xR10.
6. Next, the boot loader kernel parses the blocks of data from the EPROM. Two words are moved to memory locations 0x00000000 and 0x00000001. These are the tag words of the block to follow. In the first word, bits 31:30 are the block TYPE (0=final init, 1=non-zero init, 2=zero init), bits 29:27 are the processor ID, bits 26:16 are reserved, and bits 15:0 are

the block COUNT. The second tag word is a pointer to the DESTINATION address.

7. The ID of the block is compared to the processor ID stored in register xR10. If the two IDs are not the same, then this EPROM block is skipped. The size of the block to be skipped is obtained using the values from TYPE and COUNT.

8. If the IDs are the same, then the TYPE is examined.

9. If the TYPE is 1, COUNT number of words are moved one word at a time via location 0x00000000 to the DESTINATION address. Once finished, the algorithm returns to step 5 and repeats.

10. If the TYPE is 2, COUNT number of zeros are moved to the DESTINATION address. Once finished, the algorithm returns to step 5 and repeats.

11. If TYPE is 0, the boot loader kernel performs the "Final Init", i.e. it overwrites itself with the user application code. A final DMA of 256 words into addresses 0x00000000-0x000000ff with a wake up from an IDLE condition would do this, but in this case, this would begin execution of the user application code at interrupt level of DMA0. To avoid this scenario, the following algorithm is used:

    a. The first four instructions of the user application code (destined to reside in locations 0x00000000-0x00000003) are DMAed from the EPROM and stored in the registers xR11:8.

    b. The following code is written into locations 0x00000000-0x00000003:

    ```
    RETI = 0;;
    NOP;;
    RTI (NP); Q[j31+=0] = xR11:8;;
    ```

    c. The DMA0 interrupt vector is set to 0x00000000.

    d. The Branch Target Buffer is invalidated (BTBINV) to clear any cached branches.

    e. The DMA is setup to transfer 252 words of user code destined to 0x00000004-0x000000ff.

    f. The DMA is started by writing to the TCBs; then processor goes into an IDLE state.

    g. When the DMA is finished, the DMA complete interrupt wakes the TigerSHARC up and the program sequencer jumps to the to code inserted at 0x00000000 in step b, and begins execution of this code.

    h. This code executes the following instruction,

    ```
    RTI (NP); Q[j31+=0]=xR11:8;;
    ```

    which reduces the interrupt level to none, puts the user application code into locations 0x00000000-0x00000003, and the sequencer jumps to address 0x00000000 to continue execution (since RETI is set to 0x00000000). The user application code starts cleanly at 0x00000000, with no interrupt level. Note that the "no predict" (NP) option is necessary so that this RTI instruction does not cache into the BTB.

It is important to note if external memory which is being used in the system requires special initialization (such as SDRAM, for example), then this memory needs to be configured by the boot loader kernel. This memory configuration *must* precede its initialization in the boot loader kernel. Thus, the boot loader has to be modified by the user and re-built to their specific application and system requirements.

Host Boot

When a host or link boot is selected, the TigerSHARC enters an idle state after reset, waiting for the external host processor or link port to boot it. Host booting uses the

TigerSHARC's AUTODMA (either channel), both of which are initialized to transfer 256 words of code to the TigerSHARC's internal memory block 0, locations 0x00-0xFF. The corresponding interrupt vector is initialized to point to address 0x00. Thus, upon completion of the DMA, the TigerSHARC continues its execution at memory location 0x00. It is intended that these 256 words of code act as a boot loader to initialize the rest of the TigerSHARC's memory. Analog Devices supplies a default boot loader, named "TS101_host.asm", with the VisualDSP++ development tools.

The default host boot loader works in conjunction with the loader utility supplied with the VisualDSP++ development tools. The loader utility takes the user's project executable file and the boot loader executable ("TS101_host.dxe") and produces the host loader file, with the filename extension, *.LDR. The host loader file defines how the various blocks of the TigerSHARC's internal and external memory segments are to be initialized. Its format is described in figure 1. Format of the block tag word is described in figure 2.

In the following procedure, the AUTODMA0 register can be replaced with the AUTODMA1 register, with the appropriate changes made to the boot loader code to reflect this change in register usage.

The supplied boot loader uses the AUTODMA0 register and works as described below:

1. After the boot loader is loaded, the AUTODMA0 interrupt wakes the TigerSHARC up and starts the execution of the host boot loader kernel at location 0x00000000. At this stage, the TigerSHARC is at the interrupt level of the AUTODMA0 and, thus, further AUTODMA0 in this channel and global (PMASK[60]) interrupts are disabled.

2. The ADSP-TS101 AUTODMA0 channel is, at reset, initialized for quad-word DMAs. This boot loader uses single word DMAs to facilitate block parsing of the *. LDR file, (which may not be quad-word aligned). Thus, before any loading, the host must properly initialize the AUTODMA0 TCB register. This bootloader resides in an ASCII or an INCLUDE file, occupying locations 0x0000 - 0x0ff. Thus, the host must first initialize the AUTODMA0 TCB to move 256 words of data into 0x00000000-0x000000ff of the TigerSHARC in normal (i.e. 32-bit) words, then generate an interrupt that will vector to address 0x00000000 (the AUTODMA0 interrupt vector is already preset to address 0x00000000, by default.) Since the TCB is active at reset and since writing to an active TCB causes an error, the TCB must be disabled first. Thus, the following values must be loaded into the TCB:

   To disable the TCB:

   ```
   DP = 0x00000000
   DX, DY, DI - do not matter
   ```

   To re-enable the TCB:

   ```
   DP = 0x53000000
   DY - does not matter
   DX = 0x01000001
   DI = 0x00000000
   ```

3. The host, using the AUTODMA0 channel it just initialized, transfers all of the words of the *. LDR file to the TigerSHARC. The first 256 words of this loader file get AUTODMAed to the TigerSHARC's memory locations 0x00000000-0x000000ff. At this point the interrupt takes over and vectors the program sequencer to begin execution to location 0x00000000, i.e. the beginning of the loader. At this stage, the TigerSHARC is at interrupt level of AUTODMA0 and, thus, further AUTODMA0 and global (PMASK[60]) interrupts are disabled. Subsequent words sent by the host get parsed by the loader, as described below.

4. The loader sets the NMOD bit in the SQCTL register to ensure supervisor mode. Then, an RDS instruction reduces the interrupt to a

subroutine level. Next, AUTODMA0 and global interrupts are enabled again.

5. The AUTODMA0 interrupt vector is set to the address of the label "_dma_int".

6. The processor ID is computed and stored in register xR10.

7. Since this is not a link port boot, all of the link port controls are reset and all link port DMAs are disabled.

8. Registers xR3:0 will be used to start AUTODMA0. Register xR1 contains the count which will vary (and modify which is always one), thus xR1 will be set individually before starting AUTODMA0. Register xR3 is set to 0x53000000 and xR0 is set to 0x00000000, i.e. to normal word, interrupt enabled, destination 0x00000000 of TigerSHARC's internal memory.

9. Next, the loader parses the blocks of data from the host. It sets up to transfer two words. These are the tag words of the block to follow. In the first word, bits 31:30 are block TYPE (0=final init, 1=non-zero init, 2=zero init), bits 29:27 are the processor ID, bits 26:16 are reserved and bits 15:0 are the block COUNT. The second tag word is pointer to the DESTINATION address.

10. The ID of the block is compared to the ID stored in xR10. If the IDs are not the same, the block is skipped (if TYPE=0 or 1, by getting 255 or COUNT number of words from the host without any store) and steps starting with 9 are repeated.

11. If the IDs are the same, the TYPE field is examined.

12. If the TYPE field is 1, COUNT number of words are moved one word at a time via AUTODMA0 to the DESTINATION. Once finished, the steps starting with 9 are repeated.

13. If the TYPE field is 2, COUNT number of zeros are moved to the DESTINATION address. Once finished, the steps starting with 9 are repeated.

14. If the TYPE field is 0, the loader performs the "final init", i.e. it overwrites itself with the user application code. An AUTODMA0 of 256 words to addresses 0x00000000-0x000000ff with a wake up from IDLE would do this, but this scenario would start user application code execution at the interrupt level of AUTODMA0. To avoid this, the following algorithm is used:

    a. The first four instructions of the user code (destined to locations 0x00000000-0x00000003) are AUTODMAed from the host and stored in the registers xR11:8.

    b. The following code is written into locations 0x00000000-0x00000003:

    ```
    RETI = 0;;
    IMASKH = yR0;;
    RTI (NP); Q[j31+=0] = xR11:8;;
    ```

    c. The AUTODMA0 interrupt vector is set to 0x00000000.

    d. Register yR0 is preset to the value 0x80000000. This will be the new value for IMASKH to disable all interrupts except emulation.

    e. The Branch Target Buffer is invalidated (BTBINV) to clear any cached branches.

    f. AUTODMA0 is setup to transfer 252 words of user code destined to addresses 0x00000004-0x000000ff.

    g. AUTODMA0 is started by writing to the TCBs and then processor goes into IDLE.

    h. When the AUTODMA0 is finished, its interrupt wakes the TigerSHARC up and jumps execution to code inserted at 0x00000000 in step b.

    i. The following code is executed:

    ```
        IMASKH = yR0;;
    ```

    which disables all global interrupts, then the following instruction line is executed:

```
RTI (NP); Q[j31+=0]=xR11:8;;
```

which reduces the interrupt level to none, puts user code into locations 0x00000000-0x00000003 and jumps execution to 0x00000000 (since RETI is set to 0x00000000). The user code starts cleanly at 0x00000000, with no interrupt level. Note that (NP) option is necessary so that RTI does not cache into BTB.

It is important to note that if external memory that requires special setup (such as SDRAM) needs to be initialized by the loader, then that memory's setup has to precede its initialization in the boot loader. Thus, the boot loader has to be modified by the user and re-built.

Also, it is important to note that the host has to be careful not to overrun the AUTODMA0 buffer. Thus, the host has to monitor the AUTODMA0 status and start a new DMA only when the status is "active" (i.e. the DMA has freed the AUTODMA0 buffer). During this monitoring the host has to be careful to allow the DSP access to the external bus, otherwise software deadlock may occur, (i.e. a situation where the DSP never finishes an external transaction and the host never gets an acknowledge to start a new transaction.)

An example code of a TigerSHARC acting as a master, host booting another TigerSHARC is provided with the examples of VisualDSP; source code is in the file called, "mpmaster.asm".

Link Boot

When a host or link boot is selected, the TigerSHARC enters an idle state after reset, waiting for the host or link port to boot it. A link boot can use any one of the TigerSHARC's link ports, all of whose DMAs are initialized to transfer 256 words of code to TigerSHARC's memory block 0, locations 0x00-0xFF. The corresponding DMA interrupt vectors are initialized to 0. Thus, upon completion of the link DMA, TigerSHARC continues its execution at location 0x00. It is intended that these 256 words of code act as a boot loader to initialize the rest of TigerSHARC's memory. Analog Devices supplies a default boot loader, called "TS101_link.asm" with the VisualDSP set of tools.

The default link boot loader works in conjunction with the loader utility supplied with VisualDSP. The loader utility takes the user's project executable file (*.DXE) and the boot loader executable file, (default: TS101_link.dxe) and produces the link loader file output file, *.LDR. The LDR file defines how the various blocks of TigerSHARC's internal and external memory are to be initialized. Its format is described in figure 1. Format of the block tag word is described in figure 2.

The supplied boot loader works as described below:

1. After the boot loader is loaded, the link port's DMA interrupt wakes the TigerSHARC up and starts the execution of the loader at location 0x00000000. At this stage, the TigerSHARC is at the interrupt level of the link port DMA and, thus, further link port DMA and global (PMASK[60]) interrupts are disabled.

2. The constant LINK defines which link port is used for booting. It is set to 1 (i.e. link port 1) in this code. If a different link port is used, the constant value has to be changed and the code re-built.

3. The loader sets the NMOD bit in the SQCTL register to insure supervisor mode. Then, an RDS instruction reduces the interrupt to a subroutine level. Next, link port DMAs and global interrupts (PMASK[60]) are enabled again.

4. The link port receive DMA interrupt vector is set to the address of the label "_dma_int". Unused link ports are cleared and disabled, and link port DMAs are disabled.

5. The Link port control registers are initialized.

6. The DMA that will bring the data in from the link port will do this one quad word at a time. Registers XR3:0 are preset with the required values for the TCB.

7. The data from the link port is read by the subroutine "_read_word". The data from the link port is always in a quad-word format, but the processor needs to parse it in as single 32-bit words one at a time, an internal FIFO buffer is maintained. This is implemented as a circular buffer in memory locations 0x00-0x03, register J2 is dedicated as the read pointer to the buffer and, thus, J2, JB2 and JL2 are all initialized accordingly. The execution flow of "_read_word" is as follows:

   a. First J2 is checked to see if it has wrapped back to 0 (i.e. all the data in the buffer has been read) and, if it has not, go to step "d" to read the next piece of data from the buffer.

   b. Another quad word is brought into the buffer from the link port. To avoid data coming in at precisely the time the DMA is started (which will cause data miss due to silicon errata #147), the corresponding LSTAT is monitored in a loop waiting for the receive buffer to be full. Then, the corresponding link port DMA is started by writing registers XR3:0 to the TCB, and the routine waits for the DMA interrupt in IDLE.

   c. When the new quad word arrives from the link port, a DMA interrupt wakes the processor up from IDLE and execution is branched to "_dma_int", where a "NOP;;" instruction followed by an "RTI;;" instruction returns it back to one instruction past the IDLE. (*Note that the "NOP;;" instruction is necessary here, since an "RTI;;" instruction is not allowed to be the first instruction of an ISR.*)

   d. The data from the buffer pointed to by J2 is read into xR4 and J2 is incremented circularly.

8. Unlike other boot modes, here the processor ID is not used, since this loader does not support a multiprocessor (MP) boot.

9. Next, the loader parses the blocks of data from the link port. Two words are moved to yR8 and J0. These are the tag words of the block to follow. In the first word, bits 31:30 are block TYPE (0=final init, 1=non-zero init, 2=zero init), bits 29:16 are reserved and bits 15:0 are the block COUNT. The second tag word is the pointer to DESTINATION.

10. If TYPE is 1, COUNT number of words are moved one word at a time via "_read_word" to the DESTINATION address. Once finished, the algorithm goes to step 9.

11. If TYPE is 2, the COUNT number of zeros are moved to the DESTINATION address. Once finished, the algorithm goes to step 9.

12. If TYPE is 0, the loader performs the "final init", i.e. it overwrites itself with the user application code. The following algorithm is used:

    a. The first 28 instructions of user application code (destined to locations 0x00000000-0x0000001B) are moved from the link port via "_read_word" and stored in the registers xR31:8 and yR31:28.

    b. The interrupt service routine at "_dma_int" is relocated to 0x04-0x05.

    c. Twenty two instructions of the subroutine "_last_patch_code" are relocated to locations 0x06-0x1B.

    d. The Branch Target Buffer is invalidated (BTBINV) to clear cached branches.

    e. The link port interrupt vector is set to address 0x04 (the location now containing the interrupt service routine as a result of step b).

    f. Register yR1 is initialized to the value 0x80000000; this value will be written to IMASKH to disable global interrupts at start of user code (note that the emulation interrupt is left enabled).

g. Register J0 is initialized to 0x1C (first location past relocated "_last_patch_code") and LC0 is initialized to 0xE4 (number of words left in the final init to be read).

h. At this stage, locations 0x04-0x1B are initialized as follows:

```
0x04:  _relocated_dma_int:
nop;;
rti(NP);;

0x06:  _relocated_read_word:
// if J2 -> start of the buffer...
comp(j2,0);;
// ...bring in more data
if njeq, jump _relocated_read_buffer (NP);;

_relocated_wait_for_data:
yr2 = LSTATx;;
ybitest r2 by 3;;
if ySEQ, jump _relocated_wait_for_data NP);;

// start the DMA
DCx = xr3:0;;
// wait till DMA interrupts
idle;;

_relocated_read_buffer:
// read the word from the buffer
xr4 = cb[j2+=1];;
// and return
cjmp (ABS) (NP);;

0x0F:  _relocated_final_init1:
//read word
call _read_word (NP);;
// write it
[j0 += 1] = xr4;;
if NLC0E, jump _relocated_final_init1 (NP);;

// disable all ints except emulation
IMASKH = yr1;;
// Link disable and clear
LCTLx = yr0;;

// overwrite 0x00-0x03
Q[j31 + 0] = xr11:8;;
// overwrite 0x04-0x07
Q[j31 + 4] = xr15:12;;
// overwrite 0x08-0x0b
Q[j31 + 8] = xr19:16;;
// overwrite 0x0c-0x0f
Q[j31 + 0xc] = xr23:20;;
// overwrite 0x10-0x13
Q[j31 + 0x10] = xr27:24;;
// overwrite 0x14-0x17
Q[j31 + 0x14] = xr31:28;;

// overwrite 0x18-0x1b, start at 0
jump 0 (ABS) (NP); Q[j31 + 0x18] = yr31:28;;
```

i. The code execution jumps to 0x0F, i.e. "_relocated_final_init1" shown above.

j. Locations 0x1C-0xFF are filled with data from the link port. Note that the instruction "call _read_word (NP);;" at "_relocated_final_init1" is a relative call. Thus, it actually calls "_relocated_read_word" and overwriting the old code of "_read_word" does not cause any problems.

k. Now link port receiving is finished, correct data is in 0x1C-0xFF, and the data that should be in 0x00-0x1B is in registers xR31:8 and yR31:28. The remaining code overwrites memory locations 0x00-0x17 with the data in xR31:8 and, finally, the last line of code overwrites locations 0x18-0x1B (including itself) with data from yR31:28 while executing an absolute jump to 0x00.

l. The user code starts at 0x00 cleanly.

It is important to note that if external memory that requires special setup (such as SDRAM) needs to be initialized by the loader, then that memory's setup has to precede its initialization in the boot loader. Thus, the boot loader has to be modified by the user and re-built.

Document History

Version	Description
April 01, 2003 by G. Fowler.	Added Introduction and re-formatted document
Feb 01, 2002 by B. Lerner.	Initial Draft Release