ThreadX SMP Startup Code and Example

Startup and Demonstration Code for the Cortex-A9:

EL requires from Ambarella the following deliverables before EL can initiate this porting effort:

- Startup code that should run before the IAR ***"__iar_program_start***" processing. This code should setup the SMP environment including the shared memory system, TLBs, periodic timer interrupt, etc. It should also start execution of the additional core(s). Each additional core shall perform some low-level initialization like setting up a stack pointer, etc. and then jump to the "***addition_core_entry***" in the demo program. After everything is setup, the primary core calls the ***"__iar_program_start***" routine to initialize all the compiler information. The IAR startup routine will then call ***"main***". This startup code will be integrated into the official ThreadX release.

- The demo program will have 2 entry functions, namely ***"main***" and ***"additional_core_entry***". The ***"main***" function will only be called from the primary boot core, while the "***additional_core_entry***" will only be called by the additional core(s).

- The following is some pseudocode for what the demo program should look like:

```
volatile unsigned long   timer_interrupt_count;
volatile unsigned long   core_0_run_count;
volatile unsigned long   core_0_interrupt_count;
volatile unsigned long   core_1_run_count;
volatile unsigned long   core_1_interrupt_count;
volatile unsigned long   core_1_release_flag;    /* Must be cleared by startup code prior
                                                    to releasing core 1.  */

volatile unsigned long protection_owning_core = 0xFF; /* Nobody has the lock  */


void timer_interrupt_processing(void)
{
   timer_interrupt_count++;
}

void core_0_interrupt_handler(void)
{
   core_0_interrupt_count++;   /* Received inter-processor interrupt!  */
}

void core_1_interrupt_handler(void)
{
   core_1_interrupt_count++;  /* Received inter-processor interrupt!  */
}

int main(void)
{

   /* Release other core.  */
   core_1_release_flag =  1;
```

```
     /* Enable interrupts!  */
    enable_ints();

     /* Loop to get/release inter-core lock.   */
     while (1)
     {

          /* Get inter-core lock.  */
          get_lock();

          /* Determine if the lock is valid.  It should be 0xFF at this point!  */
          if (protection_owning_core != 0xFF)
          {
                while(1) { } /* Stick here – error!  */
          }

          /* Set the owner to core 0.  */
          protection_owning_core =  0;

          /* Send inter-core interrupt to core 1.  */
          interrupt_core(1);

         /* Set the owner back to the non-owner value.  */
         protection_owning_core =  0xFF;

        /* Release the inter-core lock.  */
        release_lock();

        /* Increment the run count.  */
       core_0_run_counter++;
     }

}

int additional_core_entry(void)
{

    /* Wait here for core 0 to get to “main”.  */
    while (core_1_release_flag == 0)
    {
    }

     /* Enable interrupts!  */

     /* Loop to get/release inter-core lock.   */
     while (1)
     {

          /* Get inter-core lock.  */
         get_lock();

          /* Determine if the lock is valid.  It should be 0xFF at this point!  */
          if (protection_owning_core != 0xFF)
          {
                while(1) { } /* Stick here – error!  */
          }

          /* Set the owner to core 1.  */
          protection_owning_core =  1;

          /* Send inter-core interrupt to core 0.  */
          Interrupt_core(0);

         /* Set the owner back to the non-owner value.  */
         protection_owning_core =  0xFF;

        /* Release the inter-core lock.  */
        release_lock();

        /* Increment the run count.  */
       core_1_run_counter++;
     }

}
```