===
Code in initialization:

```
   tcDspFirmware::set_firmware_base((void *)0x66000000);

   tcDspInterruptDispatch::set_hw_interrupt_level(1, FPGA_HW_INTERRUPT_LEVEL);

   tcDspFirmware::set_vector_enable(1, 0xffff, false);

        tcDspFirmware::get_version_info(0, &aCoreVer);

        Debug_printf("FPGA Vers: %d.%d", aCoreVer.ver1.bits.major, aCoreVer.ver1.bits.minor);
        Debug_printf("DSP Vers: %d.%d", VERSION_ID_MAJOR, VERSION_ID_MINOR);
```

===
DMA monitoring task sets up DMA and waits for interrupt:

```
int
tcDataAcq::MonitorDma(void)
{
    bool success;

    tcDspQDMA *pQDMA = tcDspQDMA::GetInstance();  // need to make sure the QDMA object is created
outside of ISR

    pQDMA->Initialize(QDMA_HW_INTERRUPT_LEVEL);

    MyCore->RegisterCallback((tfFsir2Callback)DDCCallback);
    MyCore->Enable();


    while (terminate == false)
    {


       // wait for DMA completion
        SEM_pend(mhDmaComplete, SYS_FOREVER);  // wait for DMA initiated in ISR to complete

    //  release buffer to the processing thread
        success = MBX_post(ghBufferMbx, &mpAcqArray, 0);

    // if call failed, discard the buffer
        if (success == false)
        {
           // count mbx overflows?
           APP::DebugPrint("MBX overflow");

        }

    }

    APP::MyCore->Disable();

    return (0);
}


void
tcDataAcq::Callback(void)
{
       volatile float *fifoAddress;
       float junk;


    fifoAddress = (volatile float *)APP::MyCore->GetFifoAddress();


// schedule DMA from ADC FIFO to offset in sample buffer

    tcDspQDMA::GetInstance()->ReadFromFIFOEx
             (&mhDmaComplete,                    // register semaphore
             (void *)fifoAddress,                    // pointers to source and destination
             (void *)&mpAcqArray,
             NWORDS,                           // 32-bit words to transfer
             1,                                 // write data sequentially (no stride)
             EDMA_OPT_ESIZE_32BIT,              // word size
             EDMA_OPT_PRI_HIGH,                 // priority
             true);                             // from ISR context


    return;
}
```

==
FPGA Core access code:

```
tcMyCore::tcMyCore (void *apAddress)
{
    // set base register
   mpBaseAddr = (unsigned short *)apAddress;

   mfCallback = NULL;


//  insert any FPGA register initialization code here:



// get core int level and vector

   tcDspFirmware::get_version_info(apAddress, &maCoreVer);

   mnIntLevel = maCoreVer.ver0.bits.level;
   mnIntVector = maCoreVer.ver0.bits.vector;

        tcDspInterruptDispatch::register_isr_callback
        (mnIntLevel, mnIntVector, interrupt_dispatch, (Arg)this);


    // print version info
    tcDspFirmware::print_version_info(apAddress);

}


//////////////////////////////////////////////////////////////////////////////
///
/// Default destructor.
///
//////////////////////////////////////////////////////////////////////////////

tcMyCore::~tcMyCore ()
{
    tcDspInterruptDispatch::unregister_isr_callback
       (mnIntLevel, mnIntVector, interrupt_dispatch, (Arg)this);

}



void *tcMyCore::GetFifoAddress(void)
{
   void *address;
   address =  (void *) &mpBaseAddr[FIFO_OFFSET];
   return (address);

}


void
tcMyCore::RegisterCallback(tfCallback cf)
{
    mfCallback = cf;
    return;
}

/////////////////////////////////////////////////////////////////////////////
///
/// Static interrupt dispatch routine.  Required because of the hidden this
/// pointer associated with a member function, which cannot be passed
/// directly to the interrupt dispatcher.
///
/// @param  ahMyObject   The "this->" pointer for the instance of
///                      tcDspAdcBase associated with this ISR.
///
/// @return 0
///
/////////////////////////////////////////////////////////////////////////////
int
tcMyCore::interrupt_dispatch(Arg ahMyObject)
{
       if (((tcMyCore *)ahMyObject)->mfCallback != NULL)
    {
       ((tcMyCore *)ahMyObject)->mfCallback();
       }

    return (0);
}
```