Clinton Bess

Dr. Venkatesan Muthukumar

TivaC Lab 13 - uDMA

CPE 403

Checklist for Lab 13

- ☑ *A text/word document of the initial code with comments*
- ☑ *In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.*
- ☑ *Provide a permanent link to all main and dependent source code files only (name them as LabXX-TYY, XX-Lab# and YY-task#)Screenshots of debugging process along with pictures of actual circuit*
- ☑ *Video link of demonstration.*

Code for Experiment

Task 1:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/udma.h"

// Define source and destination buffers
#define MEM_BUFFER_SIZE         1024
static uint32_t g_ui32SrcBuf[MEM_BUFFER_SIZE];
static uint32_t g_ui32DstBuf[MEM_BUFFER_SIZE];

// Define errors counters
static uint32_t g_ui32DMAErrCount = 0;
static uint32_t g_ui32BadISR = 0;

// Define transfer counter
static uint32_t g_ui32MemXferCount = 0;

// The control table used by the uDMA controller.  This table must be aligned to a 1024 byte boundary.
#pragma DATA_ALIGN(pui8ControlTable, 1024)
uint8_t pui8ControlTable[1024];

// Library error routine
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

// uDMA transfer error handler
void uDMAErrorHandler(void) {
        uint32_t ui32Status;

        // Check for uDMA error bit
        ui32Status = ROM_uDMAErrorStatusGet();

        // If there is a uDMA error, then clear the error and increment the error counter.
        if (ui32Status) {
                ROM_uDMAErrorStatusClear();
                g_ui32DMAErrCount++;
        }
}

// uDMA interrupt handler. Run when transfer is complete.
void uDMAIntHandler(void) {
        uint32_t ui32Mode;
```

```
        // Check for the primary control structure to indicate complete.
        ui32Mode = ROM_uDMAChannelModeGet(UDMA_CHANNEL_SW);
        if (ui32Mode == UDMA_MODE_STOP) {
                // Increment the count of completed transfers.
                g_ui32MemXferCount++;

                // Configure it for another transfer.
                ROM_uDMAChannelTransferSet(UDMA_CHANNEL_SW, UDMA_MODE_AUTO,
                                g_ui32SrcBuf, g_ui32DstBuf, MEM_BUFFER_SIZE);

                // Initiate another transfer.
                ROM_uDMAChannelEnable(UDMA_CHANNEL_SW);
                ROM_uDMAChannelRequest(UDMA_CHANNEL_SW);
        }

        // If the channel is not stopped, then something is wrong.
        else {
                g_ui32BadISR++;
        }
}

// Initialize the uDMA software channel to perform a memory to memory uDMA transfer.
void InitSWTransfer(void) {
        uint32_t ui32Idx;

        // Fill the source memory buffer with a simple incrementing pattern.
        for (ui32Idx = 0; ui32Idx < MEM_BUFFER_SIZE; ui32Idx++) {
                g_ui32SrcBuf[ui32Idx] = ui32Idx;
        }

        // Enable interrupts from the uDMA software channel.
        ROM_IntEnable(INT_UDMA);

        // Place the uDMA channel attributes in a known state. These should already be disabled by
default.
        ROM_uDMAChannelAttributeDisable(UDMA_CHANNEL_SW,
        UDMA_ATTR_USEBURST | UDMA_ATTR_ALTSELECT | (UDMA_ATTR_HIGH_PRIORITY |
        UDMA_ATTR_REQMASK));

        // Configure the control parameters for the SW channel.  The SW channel
        // will be used to transfer between two memory buffers, 32 bits at a time,
        // and the address increment is 32 bits for both source and destination.
        // The arbitration size will be set to 8, which causes the uDMA controller
        // to rearbitrate after 8 items are transferred.  This keeps this channel from
        // hogging the uDMA controller once the transfer is started, and allows other
        // channels to get serviced if they are higher priority.
        ROM_uDMAChannelControlSet(UDMA_CHANNEL_SW | UDMA_PRI_SELECT,
        UDMA_SIZE_32 | UDMA_SRC_INC_32 | UDMA_DST_INC_32 |
        UDMA_ARB_8);

        // Set up the transfer parameters for the software channel.  This will
        // configure the transfer buffers and the transfer size.  Auto mode must be
        // used for software transfers.
        ROM_uDMAChannelTransferSet(UDMA_CHANNEL_SW | UDMA_PRI_SELECT,
        UDMA_MODE_AUTO, g_ui32SrcBuf, g_ui32DstBuf,
        MEM_BUFFER_SIZE);

        // Now the software channel is primed to start a transfer.  The channel
        // must be enabled.  For software based transfers, a request must be
        // issued.  After this, the uDMA memory transfer begins.
        ROM_uDMAChannelEnable(UDMA_CHANNEL_SW);
```

```
        ROM_uDMAChannelRequest(UDMA_CHANNEL_SW);
}

int main(void) {

        ROM_FPULazyStackingEnable();

        ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);

        ROM_SysCtlPeripheralClockGating(true);

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
        ROM_SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UDMA);

        ROM_IntEnable(INT_UDMAERR);
        ROM_uDMAEnable();

        ROM_uDMAControlBaseSet(pui8ControlTable);

        InitSWTransfer();

        while (1) {
        }
}
```

Video Link to Demo

NONE