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