**uDMA Lab**

**Video with the following name found in Dropbox:**

VIDEO.mp4

**Part 1: uDMA demo**

// Lab12a - DMA

#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)

{

}

}

**Part 2: Ping Pong with UART**

// Lab12b - UART / DMA Ping-Pong

#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/uart.h"

#include "driverlib/udma.h"

// Buffer definitions

#define UART\_TXBUF\_SIZE 256

#define UART\_RXBUF\_SIZE 256

static uint8\_t g\_pui8TxBuf[UART\_TXBUF\_SIZE];

static uint8\_t g\_pui8RxPing[UART\_RXBUF\_SIZE];

static uint8\_t g\_pui8RxPong[UART\_RXBUF\_SIZE];

// Error counter

static uint32\_t g\_ui32DMAErrCount = 0;

// Transfer counters

static uint32\_t g\_ui32RxPingCount = 0;

static uint32\_t g\_ui32RxPongCount = 0;

// uDMA control table aligned to 1024-byte boundary

#pragma DATA\_ALIGN(ucControlTable, 1024)

uint8\_t ucControlTable[1024];

// Library error routine

#ifdef DEBUG

void

\_\_error\_\_(char \*pcFilename, uint32\_t ui32Line)

{

}

#endif

// uDMA error handler

void

uDMAErrorHandler(void)

{

uint32\_t ui32Status;

ui32Status = ROM\_uDMAErrorStatusGet();

if(ui32Status)

{

ROM\_uDMAErrorStatusClear();

g\_ui32DMAErrCount++;

}

}

// UART interrupt handler. Called on completion of uDMA transfer

void

UART1IntHandler(void)

{

uint32\_t ui32Status;

uint32\_t ui32Mode;

ui32Status = ROM\_UARTIntStatus(UART1\_BASE, 1);

ROM\_UARTIntClear(UART1\_BASE, ui32Status);

ui32Mode = ROM\_uDMAChannelModeGet(UDMA\_CHANNEL\_UART1RX | UDMA\_PRI\_SELECT);

if(ui32Mode == UDMA\_MODE\_STOP)

{

g\_ui32RxPingCount++;

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1RX | UDMA\_PRI\_SELECT,

UDMA\_MODE\_PINGPONG,

(void \*)(UART1\_BASE + UART\_O\_DR),

g\_pui8RxPing, sizeof(g\_pui8RxPing));

}

ui32Mode = ROM\_uDMAChannelModeGet(UDMA\_CHANNEL\_UART1RX | UDMA\_ALT\_SELECT);

if(ui32Mode == UDMA\_MODE\_STOP)

{

g\_ui32RxPongCount++;

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1RX | UDMA\_ALT\_SELECT,

UDMA\_MODE\_PINGPONG,

(void \*)(UART1\_BASE + UART\_O\_DR),

g\_pui8RxPong, sizeof(g\_pui8RxPong));

}

if(!ROM\_uDMAChannelIsEnabled(UDMA\_CHANNEL\_UART1TX))

{

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1TX | UDMA\_PRI\_SELECT,

UDMA\_MODE\_BASIC, g\_pui8TxBuf,

(void \*)(UART1\_BASE + UART\_O\_DR),

sizeof(g\_pui8TxBuf));

ROM\_uDMAChannelEnable(UDMA\_CHANNEL\_UART1TX);

}

}

// Initialize UART uDMA transfer

void

InitUART1Transfer(void)

{

uint32\_t ui32Idx;

// Initialize transmit buffer with some data

for(ui32Idx = 0; ui32Idx < UART\_TXBUF\_SIZE; ui32Idx++)

{

g\_pui8TxBuf[ui32Idx] = ui32Idx;

}

// Enable UART1 and make sure it can run while the CPU sleeps

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART1);

ROM\_SysCtlPeripheralSleepEnable(SYSCTL\_PERIPH\_UART1);

// Configure and enable the UART with DMA

ROM\_UARTConfigSetExpClk(UART1\_BASE, ROM\_SysCtlClockGet(), 115200,

UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |

UART\_CONFIG\_PAR\_NONE);

ROM\_UARTFIFOLevelSet(UART1\_BASE, UART\_FIFO\_TX4\_8, UART\_FIFO\_RX4\_8);

ROM\_UARTEnable(UART1\_BASE);

ROM\_UARTDMAEnable(UART1\_BASE, UART\_DMA\_RX | UART\_DMA\_TX);

// Loopback mode

HWREG(UART1\_BASE + UART\_O\_CTL) |= UART\_CTL\_LBE;

ROM\_IntEnable(INT\_UART1);

// Receive channel setup for ping and pong

ROM\_uDMAChannelAttributeDisable(UDMA\_CHANNEL\_UART1RX,

UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_USEBURST |

UDMA\_ATTR\_HIGH\_PRIORITY |

UDMA\_ATTR\_REQMASK);

ROM\_uDMAChannelControlSet(UDMA\_CHANNEL\_UART1RX | UDMA\_PRI\_SELECT,

UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_8 |

UDMA\_ARB\_4);

ROM\_uDMAChannelControlSet(UDMA\_CHANNEL\_UART1RX | UDMA\_ALT\_SELECT,

UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_8 |

UDMA\_ARB\_4);

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1RX | UDMA\_PRI\_SELECT,

UDMA\_MODE\_PINGPONG,

(void \*)(UART1\_BASE + UART\_O\_DR),

g\_pui8RxPing, sizeof(g\_pui8RxPing));

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1RX | UDMA\_ALT\_SELECT,

UDMA\_MODE\_PINGPONG,

(void \*)(UART1\_BASE + UART\_O\_DR),

g\_pui8RxPong, sizeof(g\_pui8RxPong));

// Transmit channel setup for a basic transfer

ROM\_uDMAChannelAttributeDisable(UDMA\_CHANNEL\_UART1TX,

UDMA\_ATTR\_ALTSELECT |

UDMA\_ATTR\_HIGH\_PRIORITY |

UDMA\_ATTR\_REQMASK);

ROM\_uDMAChannelAttributeEnable(UDMA\_CHANNEL\_UART1TX, UDMA\_ATTR\_USEBURST);

ROM\_uDMAChannelControlSet(UDMA\_CHANNEL\_UART1TX | UDMA\_PRI\_SELECT,

UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_8 | UDMA\_DST\_INC\_NONE |

UDMA\_ARB\_4);

ROM\_uDMAChannelTransferSet(UDMA\_CHANNEL\_UART1TX | UDMA\_PRI\_SELECT,

UDMA\_MODE\_BASIC, g\_pui8TxBuf,

(void \*)(UART1\_BASE + UART\_O\_DR),

sizeof(g\_pui8TxBuf));

// Enable both channels

ROM\_uDMAChannelEnable(UDMA\_CHANNEL\_UART1RX);

ROM\_uDMAChannelEnable(UDMA\_CHANNEL\_UART1TX);

}

// main code

int

main(void)

{

volatile uint32\_t ui32Loop;

ROM\_FPULazyStackingEnable();

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_16MHZ);

ROM\_SysCtlPeripheralClockGating(true);

// GPIO setup for LEDs

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

ROM\_GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

// GPIO setup for UART

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

ROM\_SysCtlPeripheralSleepEnable(SYSCTL\_PERIPH\_UART0);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

ROM\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Enable uDMA

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UDMA);

ROM\_SysCtlPeripheralSleepEnable(SYSCTL\_PERIPH\_UDMA);

ROM\_IntEnable(INT\_UDMAERR);

ROM\_uDMAEnable();

ROM\_uDMAControlBaseSet(ucControlTable);

// Initialize the uDMA/UART transfer

InitUART1Transfer();

// Blink the LED while the transfers occur

while(1)

{

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2);

SysCtlDelay(SysCtlClockGet() / 20 / 3);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

SysCtlDelay(SysCtlClockGet() / 20 / 3);

}

}