**Angelica Semenec**

**012179180**

**Lab 3**

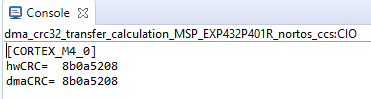
**DMA And Interrupt Mechanisms on MSP432 MCU**

**Purpose**

The purpose of this lab is to understand the MSP432 DMA and its functionality as well as some of the interrupt mechanisms.

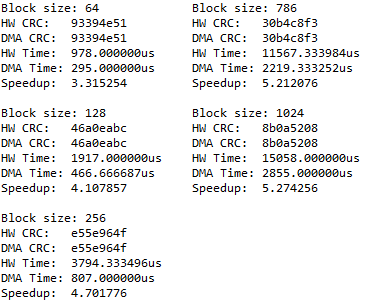
**Exercise 1**

|  |
| --- |
| **crc32\_transfer\_calculation.c** |
| /\* DriverLib Includes \*/  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  /\* Standard Includes \*/  #include <stdint.h>  #include <string.h>  #include <stdbool.h>  #include <stdio.h>  /\* Statics \*/  static volatile uint32\_t crcSignature;  /\* DMA Control Table \*/  #if defined(\_\_TI\_COMPILER\_VERSION\_\_)  #pragma DATA\_ALIGN(controlTable, 1024)  #elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_)  #pragma data\_alignment=1024  #elif defined(\_\_GNUC\_\_)  \_\_attribute\_\_ ((aligned (1024)))  #elif defined(\_\_CC\_ARM)  \_\_align(1024)  #endif  uint8\_t controlTable[1024];  uint8\_t data\_array[1024]; //Data array  #define CRC32\_SEED 0xFFFFFFFF //CRC Seed  int main(void)  {  /\* Halting Watchdog \*/  MAP\_WDT\_A\_holdTimer();  /\* Copy data to accelerator \*/  /\* Hardware method of computing checksum \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);  int ii;  for (ii = 0; ii < sizeof(data\_array); ii++)  MAP\_CRC32\_set8BitData(data\_array[ii], CRC32\_MODE);  uint32\_t hwCRC = MAP\_CRC32\_getResult(CRC32\_MODE);  printf("\nhwCRC= %08x\n", hwCRC);  /\* Configuring DMA module \*/  MAP\_DMA\_enableModule();  MAP\_DMA\_setControlBase(controlTable);  /\* Setting Control Indexes. In this case we will set the source of the  \* DMA transfer to our random data array and the destination to the  \* CRC32 data in register address\*/  MAP\_DMA\_setChannelControl(UDMA\_PRI\_SELECT,  UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_8 | UDMA\_DST\_INC\_NONE | UDMA\_ARB\_1024);  MAP\_DMA\_setChannelTransfer(UDMA\_PRI\_SELECT,  UDMA\_MODE\_AUTO, data\_array,  (void\*) (&CRC32->DI32), 1024);  /\* Assigning/Enabling Interrupts \*/  MAP\_DMA\_assignInterrupt(DMA\_INT1, 0);  MAP\_Interrupt\_enableInterrupt(INT\_DMA\_INT1);  MAP\_Interrupt\_enableMaster();    /\* Enabling DMA Channel 0 \*/  MAP\_DMA\_enableChannel(0);  /\* Reinitialize result register in CRC32 accelerator \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);  /\* Forcing a software transfer on DMA Channel 0 \*/  MAP\_DMA\_requestSoftwareTransfer(0);  while(1)  {  MAP\_PCM\_gotoLPM0();  }  }  /\* Completion interrupt for DMA \*/  void DMA\_INT1\_IRQHandler(void)  {  MAP\_DMA\_disableChannel(0);  crcSignature = MAP\_CRC32\_getResult(CRC32\_MODE);  printf("dmaCRC= %08x\n", crcSignature);  } |

****

**Exercise 2**

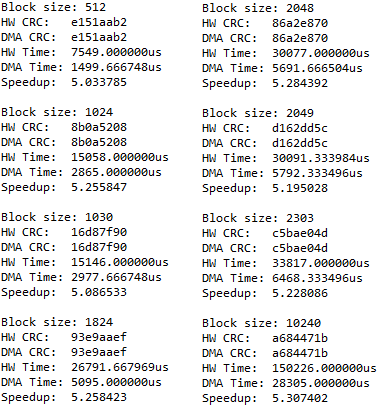
|  |
| --- |
| **crc32\_transfer\_calculation.c** |
| /\* DriverLib Includes \*/  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  /\* Standard Includes \*/  #include <stdint.h>  #include <string.h>  #include <stdbool.h>  #include <stdio.h>  /\* Statics \*/  static volatile uint32\_t crcSignature;  /\* DMA Control Table \*/  #if defined(\_\_TI\_COMPILER\_VERSION\_\_)  #pragma DATA\_ALIGN(controlTable, 1024)  #elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_)  #pragma data\_alignment=1024  #elif defined(\_\_GNUC\_\_)  \_\_attribute\_\_ ((aligned (1024)))  #elif defined(\_\_CC\_ARM)  \_\_align(1024)  #endif  uint8\_t controlTable[1024];  uint8\_t data\_array[1024]; //Data array  int size\_array[] = {64, 128, 256, 786, 1024}; //Size array to test timing  #define CRC32\_SEED 0xFFFFFFFF //CRC Seed  volatile int dma\_done; //Flag to signal when DMA is done  int main(void)  {  uint32\_t t0, t1, tHW, tDMA;  float tHW\_us, tDMA\_us, speedup;  /\* Halting Watchdog \*/  MAP\_WDT\_A\_holdTimer();  /\* Configuring DMA module \*/  MAP\_DMA\_enableModule();  MAP\_DMA\_setControlBase(controlTable);  /\* Setting Control Indexes. In this case we will set the source of the  \* DMA transfer to our random data array and the destination to the  \* CRC32 data in register address\*/  MAP\_DMA\_setChannelControl(UDMA\_PRI\_SELECT,  UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_8 | UDMA\_DST\_INC\_NONE | UDMA\_ARB\_1024);  /\* Assigning/Enabling Interrupts \*/  MAP\_DMA\_assignInterrupt(DMA\_INT1, 0);  MAP\_Interrupt\_enableInterrupt(INT\_DMA\_INT1);  MAP\_Interrupt\_enableMaster();  /\* Setup a timer \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1,  TIMER32\_32BIT, TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  uint32\_t clk\_freq = MAP\_CS\_getMCLK(); //Gets the clock frequency  printf("\nClock frequency: %uHz\n\n", clk\_freq);  int jj;  for (jj = 0; jj < 5; jj++) //For loop to iterate over each data size  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t0  /\* Copy data to accelerator \*/  /\* Hardware method of computing checksum \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);  int ii;  for (ii = 0; ii < size\_array[jj]; ii++)  MAP\_CRC32\_set8BitData(data\_array[ii], CRC32\_MODE);  uint32\_t hwCRC = MAP\_CRC32\_getResult(CRC32\_MODE);  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t1  tHW = t0 - t1;  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t0  MAP\_DMA\_setChannelTransfer(UDMA\_PRI\_SELECT,  UDMA\_MODE\_AUTO, data\_array,  (void\*) (&CRC32->DI32), size\_array[jj]); //Number of data items updated  //&CRC32->DI32 address pointer to data input of CRC  /\* Enabling DMA Channel 0 \*/  MAP\_DMA\_enableChannel(0); //Channel disabled after DMA finished; Must be called  /\* Reinitialize result register in CRC32 accelerator \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);    dma\_done = 0; //Reset DMA flag before transfer  /\* Forcing a software transfer on DMA Channel 0 \*/  MAP\_DMA\_requestSoftwareTransfer(0);  while(!dma\_done)  {  }  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t1  tDMA = t0 - t1;  tHW\_us = ((float)tHW/(float)clk\_freq)\*1000000;  tDMA\_us = ((float)tDMA/(float)clk\_freq)\*1000000;  speedup = (float)tHW/(float)tDMA;  printf("Block size: %d\n", size\_array[jj]); //Block size  printf("HW CRC: %08x\n", hwCRC); //HW CRC  printf("DMA CRC: %08x\n", crcSignature); //DMA CRC  printf("HW Time: %fus\n", tHW\_us); //HW Time  printf("DMA Time: %fus\n", tDMA\_us); //DMA Time  printf("Speedup: %f\n\n", speedup); //Speedup  }  }  /\* Completion interrupt for DMA \*/  void DMA\_INT1\_IRQHandler(void)  {  MAP\_DMA\_disableChannel(0);  crcSignature = MAP\_CRC32\_getResult(CRC32\_MODE);  dma\_done = 1;  } |



As the block size is increased, the speedup of the DMA-calculated CRC value also increases. The DMA is able to transfer the data much faster than the hardware method for the CRC calculation. At a block size of 64, the speedup is 3.3. At a block size of 1024, the speedup from utilizing the DMA is 5.2. The hardware method utilizes a *for loop* to iterate through each index within the data array before passing the designated item to the CRC calculator. At each iteration, the sub-routine *MAP\_CRC32\_set8BitData()* is called. This requires many memory accesses, which significantly slows down the process. The DMA is able to access memory independently of the processor and so the processor is left available for other calculations, which leads to an increase in the speedup of the system.

**Exercise 3**

|  |
| --- |
| **crc32\_transfer\_calculation.c** |
| /\* DriverLib Includes \*/  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  /\* Standard Includes \*/  #include <stdint.h>  #include <string.h>  #include <stdbool.h>  #include <stdio.h>  /\* Statics \*/  static volatile uint32\_t crcSignature;  /\* DMA Control Table \*/  #if defined(\_\_TI\_COMPILER\_VERSION\_\_)  #pragma DATA\_ALIGN(controlTable, 1024)  #elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_)  #pragma data\_alignment=1024  #elif defined(\_\_GNUC\_\_)  \_\_attribute\_\_ ((aligned (1024)))  #elif defined(\_\_CC\_ARM)  \_\_align(1024)  #endif  uint8\_t controlTable[1024];  uint8\_t data\_array[10240]; //Data array  int size\_array[] = {512, 1024, 1030, 1824, 2048, 2049, 2303, 10240}; //Larger data sizes to transfer  int size;  #define CRC32\_SEED 0xFFFFFFFF //CRC Seed  volatile int dma\_done; //Flag to signal when DMA is done  int main(void)  {  uint32\_t t0, t1, tHW, tDMA;  float tHW\_us, tDMA\_us, speedup;  /\* Halting Watchdog \*/  MAP\_WDT\_A\_holdTimer();  /\* Configuring DMA module \*/  MAP\_DMA\_enableModule();  MAP\_DMA\_setControlBase(controlTable);  /\* Setting Control Indexes. In this case we will set the source of the  \* DMA transfer to our random data array and the destination to the  \* CRC32 data in register address\*/  MAP\_DMA\_setChannelControl(UDMA\_PRI\_SELECT,  UDMA\_SIZE\_8 | UDMA\_SRC\_INC\_8 | UDMA\_DST\_INC\_NONE | UDMA\_ARB\_1024);  /\* Assigning/Enabling Interrupts \*/  MAP\_DMA\_assignInterrupt(DMA\_INT1, 0);  MAP\_Interrupt\_enableInterrupt(INT\_DMA\_INT1);  MAP\_Interrupt\_enableMaster();  /\* Setup a timer \*/  MAP\_Timer32\_initModule(TIMER32\_0\_BASE, TIMER32\_PRESCALER\_1,  TIMER32\_32BIT, TIMER32\_FREE\_RUN\_MODE);  MAP\_Timer32\_startTimer(TIMER32\_0\_BASE, 0);  uint32\_t clk\_freq = MAP\_CS\_getMCLK(); //Gets the clock frequency  printf("\nClock frequency: %uHz\n\n", clk\_freq);  int jj;  for (jj = 0; jj < 8; jj++) //For loop to iterate over each data size  {  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t0  /\* Copy data to accelerator \*/  /\* Hardware method of computing checksum \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);  int ii;  for (ii = 0; ii < size\_array[jj]; ii++)  MAP\_CRC32\_set8BitData(data\_array[ii], CRC32\_MODE);  uint32\_t hwCRC = MAP\_CRC32\_getResult(CRC32\_MODE);  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t1  tHW = t0 - t1;  t0 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t0  /\* Reinitialize result register in CRC32 accelerator \*/  MAP\_CRC32\_setSeed(CRC32\_SEED, CRC32\_MODE);  size = size\_array[jj]; //Get size of data array  do  {  MAP\_DMA\_setChannelTransfer(UDMA\_PRI\_SELECT,  UDMA\_MODE\_AUTO, data\_array + (jj \* 1024),  (void\*) (&CRC32->DI32), (size >= 1024) ? 1024 : size); //Number of data items updated  //Is size greater than or equal to 1024? If it is, transfer 1024. Otherwise, transfer rest of size    /\* Enabling DMA Channel 0 \*/  MAP\_DMA\_enableChannel(0); //Channel disabled after DMA finished; Must be called  dma\_done = 0; //Reset DMA flag before transfer  /\* Forcing a software transfer on DMA Channel 0 \*/  MAP\_DMA\_requestSoftwareTransfer(0);  while(!dma\_done)  {  }  size -= 1024;  }while(size>0);  crcSignature = MAP\_CRC32\_getResult(CRC32\_MODE); //Get result  t1 = MAP\_Timer32\_getValue(TIMER32\_0\_BASE); //t1  tDMA = t0 - t1;  tHW\_us = ((float)tHW/(float)clk\_freq)\*1000000;  tDMA\_us = ((float)tDMA/(float)clk\_freq)\*1000000;  speedup = (float)tHW/(float)tDMA;  printf("Block size: %d\n", size\_array[jj]); //Block size  printf("HW CRC: %08x\n", hwCRC); //HW CRC  printf("DMA CRC: %08x\n", crcSignature); //DMA CRC  printf("HW Time: %fus\n", tHW\_us); //HW Time  printf("DMA Time: %fus\n", tDMA\_us); //DMA Time  printf("Speedup: %f\n\n", speedup); //Speedup  }  }  /\* Completion interrupt for DMA \*/  void DMA\_INT1\_IRQHandler(void)  {  MAP\_DMA\_disableChannel(0);  dma\_done = 1;  } |

****

As the block size is increased, the speedup can also increase. However, the increase in speedup for data sizes between 512 and 10240 fluctuates between 5.0 and 5.3. The speedup for a block size of 1024 is 5.25 whereas the speedup for a block size of 1030 is 5.08. The highest values of speedup occur in multiples of 1024 (2048 and 10240), which is the maximum transfer size allowed by the DMA. This is different to the output in Exercise 2 where the speedup always increased for the larger blocks of data.