**Angelica Semenec**

**012179180**

**Lab 5**

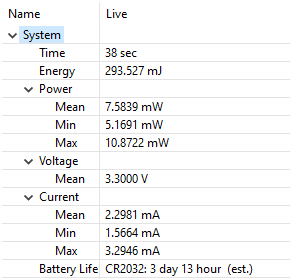
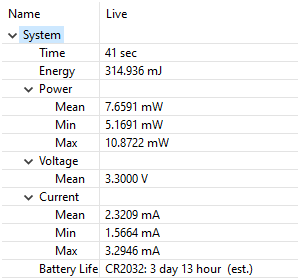
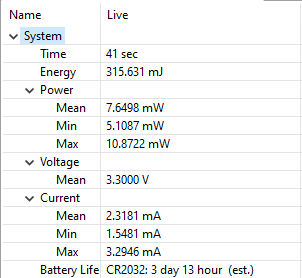
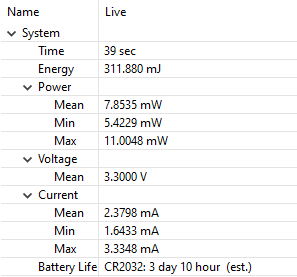
**Power Consumption and EnergyTrace on MSP432 MCU**

**Purpose**

The purpose of this lab is to become familiarized with the power consumption of the MCU using the tool EnergyTrace, which measures and displays the application’s energy profile.

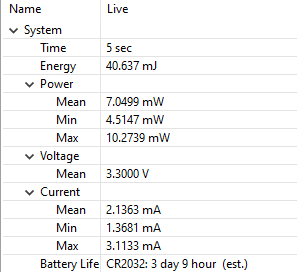
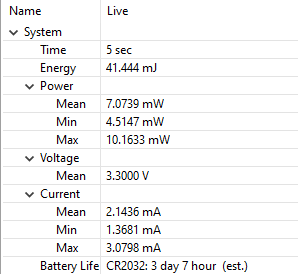
**Exercise 1**

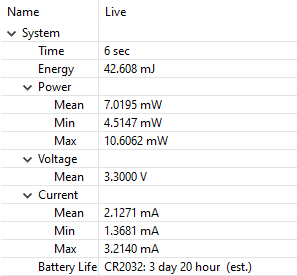
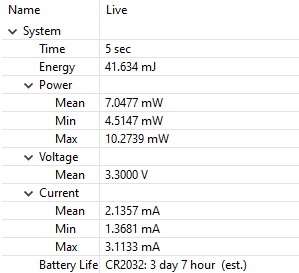
***Exercise 1.1 Software Method***

****

|  |  |  |
| --- | --- | --- |
| ***Measurement*** | ***Energy (mJ)*** | ***Mean Current (mA)*** |
| **1** | 311.880 | 2.3798 |
| **2** | 315.631 | 2.3181 |
| **3** | 314.936 | 2.3209 |
| **4** | 293.527 | 2.2981 |
| **Average** | 308.994 | 2.3292 |

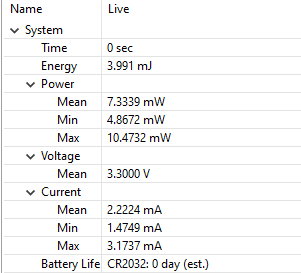
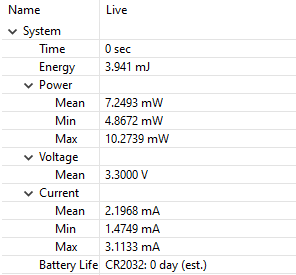
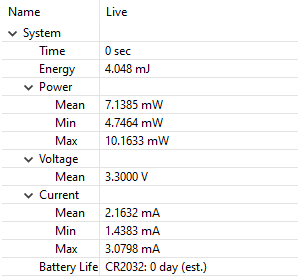
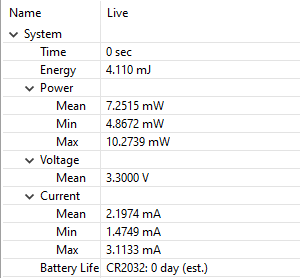
***Exercise 1.2 Hardware Method***





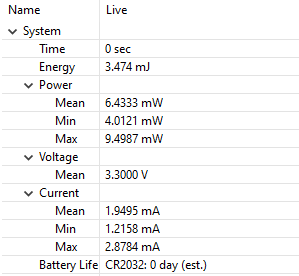
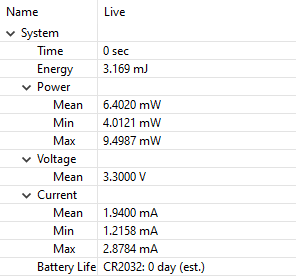
|  |  |  |
| --- | --- | --- |
| ***Measurement*** | ***Energy (mJ)*** | ***Mean Current (mA)*** |
| **1** | 41.444 | 2.1436 |
| **2** | 40.637 | 2.1363 |
| **3** | 41.634 | 2.1357 |
| **4** | 42.608 | 2.1271 |
| **Average** | 41.581 | 2.1357 |

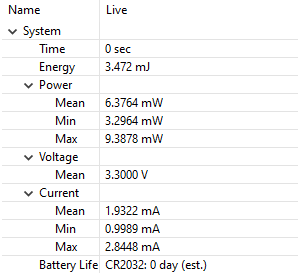
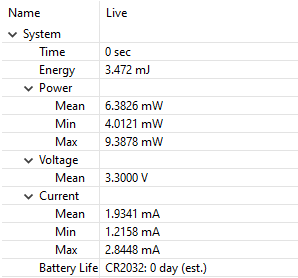
***Exercise 1.3 DMA Method***

**

|  |  |  |
| --- | --- | --- |
| ***Measurement*** | ***Energy (mJ)*** | ***Mean Current (mA)*** |
| **1** | 3.991 | 2.2224 |
| **2** | 4.110 | 2.1974 |
| **3** | 4.048 | 2.1632 |
| **4** | 3.941 | 2.1968 |
| **Average** | 4.023 | 2.1950 |

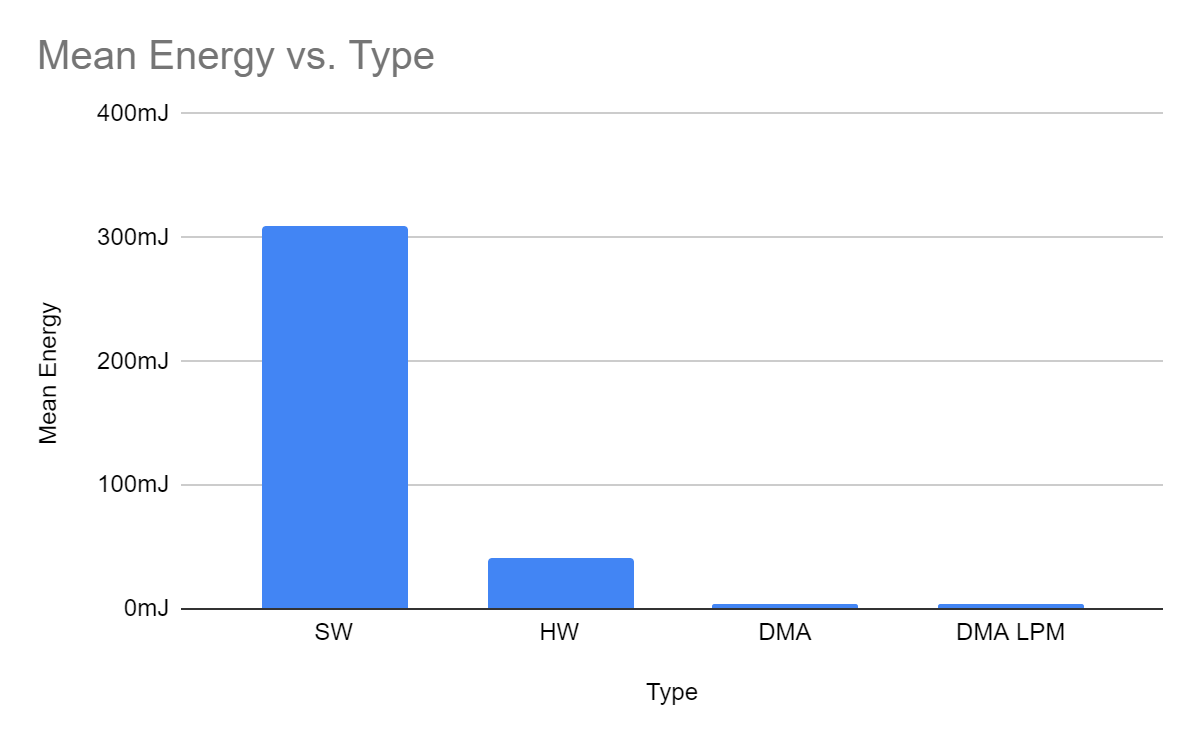
***Exercise 1.4 DMA Low-Power Mode Method***

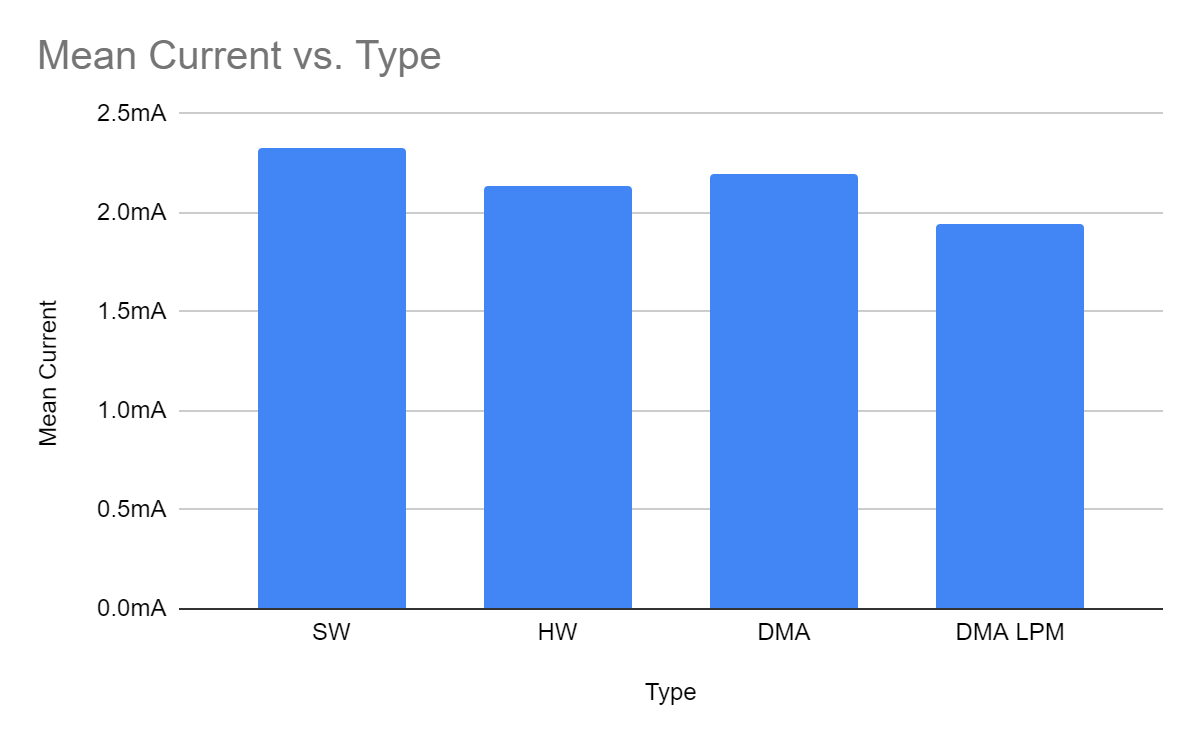
******

******

|  |  |  |
| --- | --- | --- |
| ***Measurement*** | ***Energy (mJ)*** | ***Mean Current (mA)*** |
| **1** | 3.169 | 1.9400 |
| **2** | 3.474 | 1.9495 |
| **3** | 3.472 | 1.9341 |
| **4** | 3.472 | 1.9322 |
| **Average** | 3.397 | 1.9390 |

***Exercise 1.5***

****

****

|  |  |  |
| --- | --- | --- |
| ***Method*** | ***Mean Energy (mJ)*** | ***Mean Current (mA)*** |
| **Software** | 308.994 | 2.3292 |
| **Hardware** | 41.581 | 2.1357 |
| **DMA** | 4.023 | 2.1950 |
| **DMA Low-Power Mode** | 3.397 | 1.9390 |

Comparing the different methods through their overall energy consumption reveals some dramatic differences, however the differences between the methods concerning their current is much more subtle.

The software calculation used significantly more energy to do the computation when compared with any other method. This method also used the highest mean current. The hardware method used significantly less energy than the software method, but was still significantly higher than either of the DMA methods. There is only a small difference between the DMA method in active mode and the DMA method in low-power mode (DMA LPM) when comparing average energy. The DMA LPM consumed the least amount of energy as well as the smallest mean current.

The software method of calculation is energy-intensive and had the highest mean current when compared to the other methods tested. Each instruction is carried out through software, which involves many memory accesses. The hardware method uses less energy than the software method because much of the computation is done through hardware designed to accelerate the computation. This significantly reduces the number of memory accesses, thus reducing the overall power consumption. The DMA method allows for the most energy-efficient transfer from memory as it leaves the CPU available for computation and reduces the energy needed to make memory transfers. The DMA method in low-power mode further reduces the average energy and current because program execution is suspended during the DMA transfer, which means that the processor is not consuming as much energy.

**Appendix**

|  |
| --- |
| **main.c** |
| #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #include <stdio.h>  #include <stdlib.h>  #define CRC32\_POLY 0xEDB88320  #define CRC32\_INIT 0xFFFFFFFF  /\* 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];  #define DATA\_SIZE 1024 // Cannot be greater than 1024 for DMA method  uint8\_t myData[DATA\_SIZE];  static uint32\_t calculateCRC32(uint8\_t\* data, uint32\_t length);  volatile uint32\_t hwCalculatedCRC, swCalculatedCRC, dmaCalculatedCRC;  #define LOOP\_COUNT 500 // Number of iterations to do  //#define DO\_PRINTF  #define USE\_SW\_METHOD  #define USE\_HW\_METHOD  #define USE\_DMA\_METHOD  #define USE\_DMA\_LPM\_METHOD  int dma\_channel\_number = 0;  volatile bool dma\_done;  int main(void)  {  int loop;  /\* Stop WDT \*/  MAP\_WDT\_A\_holdTimer();  #ifdef DO\_PRINTF  printf("Lab 5. Exercise 1.\n");  #endif  #if defined(USE\_DMA\_METHOD) || defined(USE\_DMA\_LPM\_METHOD)  /\* Configuring DMA module \*/  MAP\_DMA\_enableModule();  MAP\_DMA\_setControlBase(controlTable);  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();  for (loop=0; loop<LOOP\_COUNT; loop++)  {  MAP\_CRC32\_setSeed(CRC32\_INIT, CRC32\_MODE);  MAP\_DMA\_setChannelTransfer(UDMA\_PRI\_SELECT, UDMA\_MODE\_AUTO, (void\*)myData, (void\*) (&CRC32->DI32), sizeof(myData));  /\* Enabling DMA Channel 0 \*/  MAP\_DMA\_enableChannel(0);  /\* Forcing a software-trigger transfer on DMA Channel 0 \*/  dma\_done = 0;  MAP\_DMA\_requestSoftwareTransfer(0);  while(!dma\_done)  {  #ifdef USE\_DMA\_LPM\_METHOD  MAP\_PCM\_gotoLPM0();  #endif  }  dmaCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;  #ifdef DO\_PRINTF  printf("dmaCalculatedCRC=%x\n", dmaCalculatedCRC);  #endif  }  #endif  #ifdef USE\_HW\_METHOD  for (loop=0; loop<LOOP\_COUNT; loop++)  {  MAP\_CRC32\_setSeed(CRC32\_INIT, CRC32\_MODE);  uint32\_t ii;  for (ii = 0; ii < sizeof(myData); ii++)  MAP\_CRC32\_set8BitData(myData[ii], CRC32\_MODE);  /\* Getting the result from the hardware module \*/  hwCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;  #ifdef DO\_PRINTF  printf("hwCalculatedCRC=%x\n", hwCalculatedCRC);  #endif  }  #endif  #ifdef USE\_SW\_METHOD  for (loop=0; loop<LOOP\_COUNT; loop++)  {  /\* Calculating the CRC32 checksum through software \*/  swCalculatedCRC = calculateCRC32((uint8\_t\*) myData, sizeof(myData));  #ifdef DO\_PRINTF  printf("swCalculatedCRC=%x\n", swCalculatedCRC);  #endif  }  #endif  #ifdef DO\_PRINTF  printf("Done.\n");  #endif  abort(); // Stop program execution immediately  }  /\* Completion interrupt for DMA \*/  void DMA\_INT1\_IRQHandler(void)  {  dma\_done = 1;  }  /\* Standard software calculation of CRC32 \*/  static uint32\_t calculateCRC32(uint8\_t\* data, uint32\_t length)  {  uint32\_t ii, jj, byte, crc, mask;;  crc = 0xFFFFFFFF;  for(ii=0;ii<length;ii++)  {  byte = data[ii];  crc = crc ^ byte;  for (jj = 0; jj < 8; jj++)  {  mask = -(crc & 1);  crc = (crc >> 1) ^ (CRC32\_POLY & mask);  }  }  return ~crc;  } |