LINGI 2315 - Evaluation 3 - Project **P1**

Name	Delcoigne Ben	Noma	38771700

Explanation of your implementation for question 1 with relevant screenshots of your code

I created a task called gsensor (which was created for our homework) that constantly reads the value of the gsensor. Once done, I must select the axis, for that I read from the dispsw (i had to create the fpga_sw variable as usual). Once done, decided which value to read. Depending on the switch, ret will be the x, y or z value. I also set a shift 13,12 or 11 bits.
I then take the value (ret) and set the nth bit to 1; this enables me to deduce whether it was the x, y and z axis on reception. Indeed; At the reception i just have to figure out which switch was set, unset it and display the value.

Note: i start from 11 bits shift because I need 10 bits to display the g value (which goes up to 1024); when expecting bigger values, the shift should be higher but it's a quick fix

```c
for(;;){
    XL345read(&x,&y,&z);

    MTXLOCK_STDIO();
    //printf("%d:%d:%d\n",x, y, z);
    MTXUNLOCK_STDIO();

    //I MUST NOW SEND THAT TO THE NIOS USING SPI
    uint32_t sw = alt_read_word(fpga_sw);
    int ret;
    int shift;
    switch(sw){
    case 1:
        ret = x;
        shift = 13;
        break;
    case 2:
        ret = y;
        shift = 12;
        break;
    default:
        ret=z;
        shift = 11;

    }

    if(ret<0){
        ret = -ret;
    }

    //Ret is now a uint32_t with the last bits set to x
    MTXLOCK_STDIO();
    printf("Switch reading was %d; ret is: %d; adding the value bit: %d; initial; %d\n",(int)sw, ret, ret|1<<
    MTXUNLOCK_STDIO();
    int tosend = ret|1<<shift;
    int xstatus = (tosend>>13)&1;
    int ystatus = (tosend>>12)&1;
    int zstatus = (tosend>>11)&1;
    printf("axis: %d, %d, %d --- sending %d\n", xstatus, ystatus, zstatus, tosend);
    //i set
    alt_write_word(fpga_spi + SPI_TXDATA,tosend);//Writing the value on the nios
```

```c
//      IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_NIOS_BASE, serial);
//  }
    serial = IORD_ALTERA_AVALON_SPI_RXDATA(SPI_NIOS_BASE);
    printf("Recieving value in NIOS: %d\n", serial);


    int xstatus = (serial>>13)&1;
    int ystatus = (serial>>12)&1;
    int zstatus = (serial>>11)&1;
    int shift = 11;
    if(xstatus) shift = 13;
    if(ystatus) shift = 12;

    int value = serial & ~(1<<shift);
    printf("VALUE: %d\n", value );
    if(shift ==13){
        printf("Axis: X\n");
    }
    if(shift ==12){
        printf("Axis: Y\n");
    }
    if(shift ==11){
        printf("Axis: Z\n");
    }
//  if(serial<0){
```

For the SPI, (right) I just decode the value by doing the inverse operation, and deduce which axis was used. This is what was shown at the demo.
Note:
I could have done much simpler. I could just have read the switch state from the SPI. This is much simpler: I don't have to encode the bits and can just plainly send the value of the gsensor to the nios. Code of SPI becomes:

```c
    }
}
    serial = IORD_ALTERA_AVALON_SPI_RXDATA(SPI_NIOS_BASE);
    int switches = IORD_ALTERA_AVALON_SPI_RXDATA(DIPSW_PIO_NIOS_BASE);
    printf("Recieving value in NIOS: %d\n", serial);

    switch(sw){
    case 1:

        printf("AXIS: X\n");
        break;
    case 2:
        printf("AXIS: Y\n");
        break;
    default:
        printf("AXIS: Z\n");

    }
```

Name	Delcoigne Ben	Noma	38771700

Explanation of your implementation for question 2 with relevant screenshots of your code

For question 2; I used the transfer example from myapp_DMA and repeated the transfer three times using a for loop.

```c
    printf("DMA tests WITHOUT ACP \n");
    for(int loopings=0; loopings<3; loopings++){
        //==============================================================TRANSFER1
        memset((void *) DMA_Src, 0x55, DMA_Size);
        memset((void *) DMA_Dst, 0xAA, DMA_Size);
        Tick = G_OStimCnt;
        i=0;
        DMA_Src       = (uint8_t *) 0x30000000;//edit - 0x30000000
        DMA_Dst       = (uint8_t *) 0x20000000;//Edit - 0x20000000
        DMA_Size      = 10000000;//EDIt 30000000
        DMA_OpMode[i++] = DMA_CFG_EOT_ISR;
#if (USE_ACP == 1)
        DMA_OpMode[i++] = DMA_CFG_NOCACHE_SRC;
        DMA_OpMode[i++] = DMA_CFG_NOCACHE_DST;
#endif
        DMA_OpMode[i++] = 0;
        DMA_Err = dma_xfer(0,
                (uint8_t *)(ACPwrt + (uintptr_t) DMA_Dst), 1, MEMORY_DMA_ID,
                (uint8_t *)(ACPrd  + (uintptr_t) DMA_Src), 1, MEMORY_DMA_ID,
                1, 1, DMA_Size,
                DMA_OPEND_NONE, NULL, (intptr_t)0,
                &DMA_OpMode[0], &DMA_XferID, OS_MS_TO_TICK(1000));

        Tick = G_OStimCnt - Tick;
        if (DMA_Err != 0) {
            printf("\ndma_xfer() reported the error #%d\n", DMA_Err);
        } else {
            Err = 0;
            for (i=0; i<DMA_Size; i++)
                if (*(DMA_Src+i) != *(DMA_Dst+i)) {
                    Err++;
                    if (Err < 5)
                        printf("Error in DMA transfert : %x instead of %x at %x (%x)\n", *(DMA_Dst+i), *(DMA_Src+i), (unsigned int)
                }
            printf("DMA Test: Size: %d - Xfer: %d ms - %d MB/s \n\n", DMA_Size, (OS_TIMER_US*Tick)/1000, (Tick == 0) ? 0 : DMA_Size
        }
        MTXUNLOCK_STDIO();
    }
    //==============================================================END OF TRANSFER 1
```

Afterwards, I enable the ACP:

```c
//NOW ENABLIGN ACP
        ACPwrt = acp_enable(-1, 0, 0, 0);               /* Page 0 (0x00000000->0x3FFFFFFF) is set-up    */
        ACPrd  = acp_enable(-1, 0, 0, 1);               /* to use ACP for both read & write             */
```

And repeat the for loops described above. We notice the transfer is faster without ACP which makes sense since ACP checks consistency across cache&memory.