Elias Michael
Marino Lorenzo
Sansone Giacomo
Valenza Arianna

DMA Exercises
Operating Systems for Embedded Systems

27.02.23

1 Exercise 1 - Hacking a system, safety at risk

This exercise simulates a DMA attack to the control unit charged to monitor the state of the payload system in a spacecraft. The aim is to cause the failure of the mission, changing willingly the parameters of the sensors to shut down the on board computer OBC, which cannot operate in particular harsh conditions.

1.1 Purpose

The purpose of this exercise is to configure the DMA and UART to be able to transmit data to the system's memory and displaying these data on the LCD display. A struct `parameters` is used to store the temperature, pressure and CRC. Data must be sent to this struct. All needed tasks are created and the fields for UART and DMA are written but commented.

1.2 Configuring UART

All fields needed for the UART configuration are already written in the `APP_TSK` task. Assign these fields such that:

- RS232 must be connected to UART2 peripheral, therefore COM2 must be used.
- The Uart peripheral clock is `UART_CCLK_DIV_4` (CPU_CLOCK / 4).
- Set DLL, DLM, DivaddVal and MulVal fields such that the Baud rate is 9600.
- Enable the FIFO, Enable DMA mode and disable the UART interrupt.
- Trigger the DMA request interrupt each time 1 element is transferred.
- Set word size to 8 bits, Stop to 1 bit, enable even parity bit.

Hint: Use the macros already defined in the `uart.h` file.
Hint: To calculate the baud rate see the 14.4.12.1 section of the User Manual

1.3 Configure PuTTY

To simulate the data coming from the sensors on UART it is necessary to set up a terminal emulator, like PuTTY, with serial connection to the board. The UART parameters must be set as specified in the previous point: 8 data bits, 1 stop bit, parity bit enabled, none flow control.

1.4 Configuring the DMA

Still in the `APP_TSK`, all parameters for configuring the DMA are also written. Assign these fields such that:

- Channel 1 is selected for GPDMA.
- Transfer size should be equal to the size of the struct `myparam`.
- Destination address is the struct `myparam`.
- Source address is the receive buffer register of the UART.
- Both the source and destination burst size is 1 byte.
- Whether the source and destination increment is set or not is left to the student to determine.
- Both source and destination transfer width are 1 byte.
- DMA should generate an interrupt at the end of the transfer.
- The transfer type, source peripheral and destination peripheral are also left to the student to determine.
- Linked list is not used.

Hint: Use the macros already defined in the `dma.h` file.

1.5 `DISPLAY_TSK`

This task is simply used to display the current value of the temperature and Pressure on the LCD. Make use of the `tostring()` function.

1.6 `LOCK_TSK`

The role of this task is to shut down the OBC based on the values of the temperature, pressure and CRC transmitted through UART. There are 2 conditions that must be satisfied:
1) CRC condition: Originally the CRC value is computed by the sensor and then sent to the board, then the LOCK_TSK has to compute the CRC once again to make sure that the data was transferred correctly. If the received CRC and the computed CRC are equal, the condition is satisfied. In this exercise the CRC should be computed manually and then sent by putty. The CRC is computed as follows: All bytes of the temperature (4 bytes) and the pressure (2 bytes) must be XORed, the result is the CRC which is also on 1 byte.
2)Random condition based on the temperature and pressure: the exact formula for this condition doesn't really matter, we considered that whenever the sum of the temperature and pressure is greater than 300, the OBC is shut down.

1.7 `DMA_IRQHanlder`

There are no specific instructions for the Handler of the DMA request, write some code in order to make the previous 2 tasks work as intended.

2 Exercise 2 - Steal the password

The goal of this exercise is to find a password that is hard-coded inside the code.

2.1 What is hard-coded password

An explanation of what an hard-coded password is can be found here.
The idea is to avoid having a variable of some kind with the password written inside; instead, we use some assembly instruction to check that the input value is correct. In general, embedding the credentials inside the code is considered by CWE as the 15th most dangerous software weakness in 2022 (source).
Let's make an example. Suppose that a password is an 8 bit number. A simple way to control the value that was inserted would be

```
// ...
real_password = 0xA2;
if(input_password == real_password)
    // correct password
else
    // wrong password
```

To hard code the same password, we may write

```
if((input_password & 0x01) == 0){
    if(input_password & 0x02) == 1{
        if(input_password & 0x04) == 0){
            if(input_password & 0x08) == 0){
                if(input_password & 0x10) == 0){
                    if(input_password & 0x20) == 1){
                        if(input_password & 0x40) == 0){
                            if(input_password & 0x80) == 1){
                                // correct password
                            }
                        }
                    }
                }
            }
        }
    }
}
// wrong password
```

In this way, each bit of the password will be checked using a `CMP` ARM instruction, and the values `0` and `1` will be stored inside the binary of the code.
As an advantage of this technique, you don't have any variable which stores the password in plain text, so you can avoid simple binary exploitation tricks.
However, there are multiple ways to detect the password either way, especially if an attacker has access to the code (although this is not always true: for instance, you can work on the execution time of the program: the more it requires to tell you a password is wrong, the more if-cases are executed and the more the password is true).

2.2 Set up DMA configuration

Inside a function `passwordcheck` the real password is used to check the correspondence with the passed parameter, as a sequence of if statements. You do not have access to the source code of this function.
However you are able to access the DMA controller, modifying both the source address and the size of the transfer (that are the macros `SOURCEADDRESS` and `TRANSFERSIZE`) can help you to access to the reserved area of memory.
The `password` array must be filled with the password you think is correct. The function `passwordcheck` will be executed on that array, and you will be able to know if the one you discovered is right.
If you choose the right destination, the content that is read by the DMA controller will be shown on the screen of the board in hexadecimal. You must decode the ARM code on screen to discover the real password.

2.3 Use of buttons

The board will work as follows:

- By pressing the button 1, you will start the DMA transfer from the selected area of memory;
- By pressing the button 2, you will see the values of the destination buffer of the DMA transfer on the screen;
- By pressing the button 3, you can see if the input password is correct or not.

2.4 Important considerations

Please, do not use the debugger to get the assembly code. Try to work as if you were only able to use the DMA. Do not either use the debugger to find the address of the function. You have access to the `.map` file of the project (stored inside the `listings/` folder of the project).
Hint 1: What is the `.map` file?
Hint 2: Once you are able to access the hidden function, you should compare the shown binary opcodes with the manual in order to understand what the code is performing ARM Manual here.
Hint 3: Do not be fooled by the endianess, pay attention also in the .map file when reading the source address (Thumb instructions).