Overview on DMA Attacks

02NPSOV - Operating Systems For Embedded Systems

Elias Michael   - s307276
Marino Lorenzo   - s317703
Sansone Giacomo - s307761
Valenza Arianna - s317742

Table of contents

01

DMA and protection

What are DMA and memory protection?

What is DMA?

Direct Memory Access:

Direct Memory Access is a feature that allows hardware devices to access the memory directly, independently of the CPU.

Why we want to use it?

Normally, a device read or write data to memory through the CPU. With DMA, the device can bypass the CPU and access memory directly. This speeds up data transfer and frees up the CPU to perform other tasks.

Where can we find it?

First party DMA

A.K.A self-DMA, is a type of DMA where the device itself manages the data transfer between its own buffer and the system memory. The device initiates the transfer and controls the data flow without the involvement of the CPU.
First-party DMA is commonly used in devices such as disk drives and network adapters.

Third party DMA

A.K.A bus-master DMA, consists of a separate DMA controller that manages the data transfer between a device and system memory. In this case, the device sends a request to the DMA controller, and the controller performs the transfer.
It is found on several SoC.

How it works

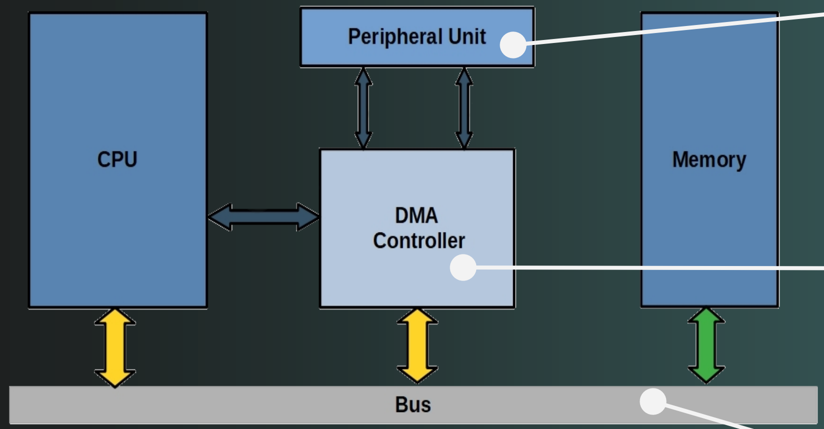


Image take from open4tech.com

Device request

The device request a dma trasfer, providing the source/destination physical addresses

Prepare the transfer

DMA controller process the request

Data transfer

The DMA controller accesses the system bus and copies the requested data.

What is memory protection?

Memory protection is a feature that protects the system from unauthorized access and modification of system memory (e.g. prevent a process from accessing memory that has not been allocated to it). It is an essential component of computer security and helps to prevent malicious programs from accessing or modifying sensitive data.

How it is implemented

Segmentation

The memory is divided in distinct segments, each with its own protection level

Access control

A Software mechanism that grants access to memory, based on the user privileges.

Virtual memory

Virtual addresses are mapped into physical addresses through an MMU

02

DMA Attacks

Why DMA can be dangerous?

DMA Attacks

Definition

A DMA attack is defined as when an attacker gains access to the victim’s system memory, enabling him to read/write from/to the memory bypassing the system’s CPU

Characteristics

DMA attacks are easier to implement on embedded systems than on general purpose computer. After the DMA is set, data transfers can be done without the aid of neither the CPU nor the operating system.

Example of DMA attacks

Access and/or modify reserved information

Extend control over the kernel itself

Malware injection

Countermeasures

Using an IOMMU

It limits the memory addresses accessible by a peripheral

NX-BIT

It determines whether the content of that page can be used for code execution or not.

ASLR

It is a memory-protection technique that randomizes the memory layout.

Secure Boot

It is a process that verifies the integrity of the system’s boot process.

DMA on LPC1768

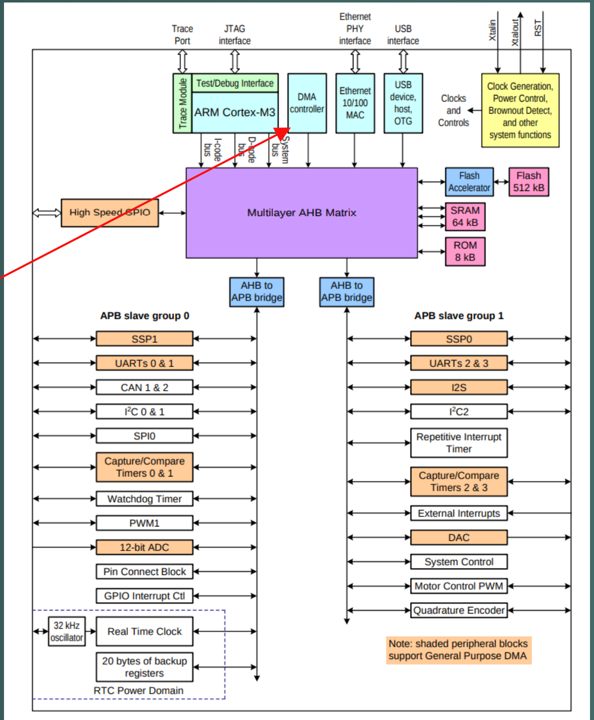
03

Getting start with a DMA of a real SoC

LPC 1768: an SoC

The LPC1768 is a Cortex®-M3 microcontroller for embedded applications.

The DMA controller allows peripheral-to memory, memory-to-peripheral, and memory-to-memory transactions. Each of the 8 DMA stream provides unidirectional serial DMA transfers for a single source and destination. It is highly configurable.



This image comes from the LPC1768 user manual

Our DMA library

```
struct dma_transfer{
    uint32_t    channel;
    uint32_t    size;
    uint32_t    dest;
    uint32_t    source;
    uint32_t    source_burst_size;
    uint32_t    dest_burst_size;
    uint32_t    source_increment;
    uint32_t    dest_increment;
    uint32_t    source_transfer_width;
    uint32_t    dest_transfer_width;
    uint32_t    terminal_count_enable;
    uint32_t    transfer_type;
    uint32_t    source_peripheral;
    uint32_t    dest_peripheral;
    uint32_t    lli_head;
};

void init_DMA(uint8_t);
int  transfer_DMA(struct dma_transfer*);
void lli_add_elem(struct dma_lli_element**, struct dma_lli_element*);
extern void DMA_IRQHandler(void);
```

A DMA transaction requires the GPDMA controller to be initialized using **init_DMA**. Then, the parameters of the transaction are specified inside an instance of the structure **dma_transfer**, which is passed as parameter to the **transfer_DMA** function. This function sets all the registers of the peripheral according to the specification of the user manual of the SOC.

Avoid low-level code!

Since most of the values of the parameters are low-level dependent, we defined many macros to initialize the different elements without the need to look at the user manual. More details about the usage of the library can be found on our documentation. The image below shows and example of initialization, that is self-explained

```
dmaRecvTransfer.channel = DMA_CHANNEL_1;
dmaRecvTransfer.size = BUFFER_SIZE;
dmaRecvTransfer.dest = (uint32_t)&myparam;
dmaRecvTransfer.source = (uint32_t)&LPC_UART2->RBR;
dmaRecvTransfer.source_burst_size = DMA_BURST_SIZE_1;
dmaRecvTransfer.dest_burst_size = DMA_BURST_SIZE_1;
dmaRecvTransfer.source_increment = DMA_NO_INCREMENT;
dmaRecvTransfer.dest_increment = DMA_INCREMENT;
dmaRecvTransfer.source_transfer_width = DMA_TRANSFER_WIDTH_8;
dmaRecvTransfer.dest_transfer_width = DMA_TRANSFER_WIDTH_8;
dmaRecvTransfer.terminal_count_enable = DMA_TERMINAL_COUNT_ENABLE;
dmaRecvTransfer.transfer_type = DMA_P2M;
dmaRecvTransfer.source_peripheral = DMA_UART2_RX;
dmaRecvTransfer.dest_peripheral = DMA_NO_PERIPHERAL;
dmaRecvTransfer.lli_head = 0;
transfer_DMA(&dmaRecvTransfer);
```

A serial connection

To allow a connection with the board/SoC, we used the UART peripheral. We can send and receive data through a serial terminal on our laptop (PuTTY on Windows, tio on Linux). To do this, we need an RS232-USB cable and to initialize the UART peripheral inside the board. Don't worry, we wrote a library for it as well!

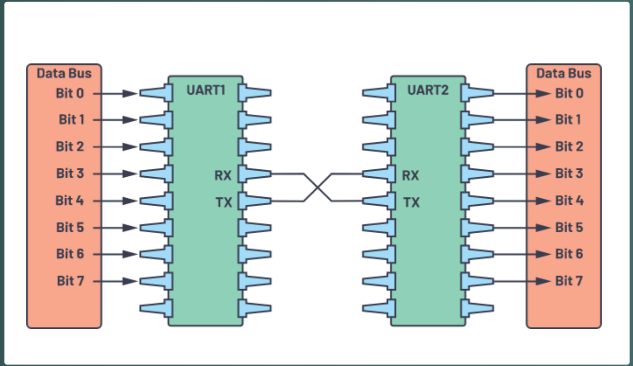


Image take from here

Our UART library

The library is similar to the one for DMA, and its characteristics can be read inside the documentation. We used this peripheral since it allows a direct transfer to memory through DMA: data are sent/received directly without the need of the CPU to handle them.

```c
struct uart_parameters{
    uint32_t    uart_n;
    uint32_t    clock_selection;
    uint32_t    DLL;
    uint32_t    DLM;
    uint32_t    DivAddVal;
    uint32_t    MulVal;
    uint32_t    fifo_enable;
    uint32_t    dma_mode;
    uint32_t    trigger_level;
    uint32_t    rdv_int;
    uint32_t    word_size;
    uint32_t    stop_bit_n;
    uint32_t    parity_active;
    uint32_t    parity_type;
};

int init_uart(struct uart_parameters*);
```

This allows us to simulate a real transfer from the outside world, and understand how you can modify certain values to **hack the system**.

04

Exercises on DMA attack

Hands-on exercises

Different purposes

Safety at risk

Break the safety system of the payload of a spacecraft.

Steal the password

Capture the hardcoded password of an embedded system.

Safety at risk/1

Setup

Keil uVision5

IDE used to source code editing and program debugging

PuTTY

Terminal Emulator needed to simulate the flow of incoming external data to the board

LandTiger LPC1768

Physical embedded system connected to host pc through UART connection

Safety at risk/2

DMA

Direct Memory Access speeds up the data transfer with sensors and permits the DMA attack.

LCD

Screen outputs the status of the system continuously.

LandTiger v2.0
LPC1768 (source)

UART

The system receives data from sensors over UART and executes operations based on the incoming data.

CRC is applied to the received data and if the condition is not respected actions are not performed.

Hard-coded credentials

15th

Position on 2022 Common Weakness Enumeration (CWE™) Top 25 Most Dangerous Software Weaknesses list

Steal the password

Execute a DMA attack to access to critical memory area, using .map file to identify the address.

Steal the password consists in decoding ARM instructions that are used to check if the input password is correct.

LCD will display the receiving buffer of the DMA transfer in hexadecimal.

Source code of the password check is not provided.

This work was possible thanks to *Politecnico di Torino* and to *EMNESS Project*

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**