Workshop - ECU Firmware Analysis

Focus on the CAN driver

Anthony Rullier **@deadeert**

Quarkslab

Table of Contents

What a MCU is ? - Generalities (1/2)

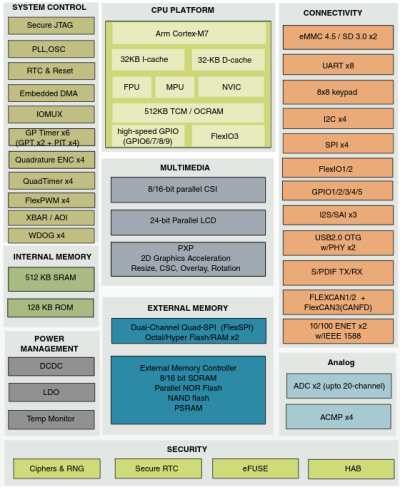


Figure: I.MX RT1060 General Diagram

What a MCU is ? - Generalities (2/2)

Micro-controller embeds numerous controllers and peripherals

- It ships with core(s) where instructions are fetched, decoded and executed
- It has an internal memory where its firmware is stored and executed in place.
- It has controllers / unit that offers built-in connectivity for protocols such as USB, Ethernet, CAN, ...
- Its capacity might be easily extended using external memory, external peripheral using standard protocols such as I2C, SPI, (S)UART, ...

MCU runs code

- located inside it's internal flash or using an external memory
- firmware can be stocked using several different formats.

Loading Firmware - Firmware Format

Some generalities regarding firmware storing:

- MCU generally stores their firmware in **records**, the most common are
  - **ihex**[1]: Intel hex record
  - **srec**[2]: Motorola s-record
- The records are ASCII printable.
- Not a lot of thing to remind:
  - A data record generally contain an **address** and **length** and a **payload**.
  - several library allow to convert such **records** to binary file.

[1] https://fr.wikipedia.org/wiki/HEX_(*Intel*)
[2] https://en.wikipedia.org/wiki/SREC_(file_format)#/media/File:Motorola_SREC_Chart.png

TP1 - Convert firmware to binary file

Write a script to convert hex records to raw binary file

- Use *Firmwares/Teensy32/Grehack_VirtECU.ino.hex* file.
- See **bincopy** python library[3].
- Correction will be available in file *Correction/Scripts/shex2raw.py*.

[3] https://pypi.org/project/bincopy/

What a MCU is ? - Peripheral Access (1/3)

Each peripheral general benefits from a set of registers

- These registers are memory mapped in physical memory space (MMIO).
- These registers can have various purposes.
  - Configure the peripheral.
  - Probe status / Query Informations.
  - Acknowledge interruption.
- All the information are generally provided across documentations.

What a MCU is ? - Peripheral Access (2/3)

CAN memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4002_4000	Module Configuration Register (CAN0_MCR)	32	R/W	D890_000Fh	44.3.2/1048
4002_4004	Control 1 register (CAN0_CTRL1)	32	R/W	0000_0000h	44.3.3/1053
4002_4008	Free Running Timer (CAN0_TIMER)	32	R/W	0000_0000h	44.3.4/1056
4002_4010	Rx Mailboxes Global Mask Register (CAN0_RXMGMASK)	32	R/W	FFFF_FFFFh	44.3.5/1057
4002_4014	Rx 14 Mask register (CAN0_RX14MASK)	32	R/W	FFFF_FFFFh	44.3.6/1058
4002_4018	Rx 15 Mask register (CAN0_RX15MASK)	32	R/W	FFFF_FFFFh	44.3.7/1059
4002_401C	Error Counter (CAN0_ECR)	32	R/W	0000_0000h	44.3.8/1059
4002_4020	Error and Status 1 register (CAN0_ESR1)	32	R/W	0000_0000h	44.3.9/1061
4002_4028	Interrupt Masks 1 register (CAN0_IMASK1)	32	R/W	0000_0000h	44.3.10/ 1065
4002_4030	Interrupt Flags 1 register (CAN0_IFLAG1)	32	R/W	0000_0000h	44.3.11/ 1066
4002_4034	Control 2 register (CAN0_CTRL2)	32	R/W	00B0_0000h	44.3.12/ 1068
4002_4038	Error and Status 2 register (CAN0_ESR2)	32	R/W	0000_0000h	44.3.13/ 1071
4002_4044	CRC Register (CAN0_CRCR)	32	R	0000_0000h	44.3.14/ 1072
4002_4048	Rx FIFO Global Mask register (CAN0_RXFGMASK)	32	R/W	FFFF_FFFFh	44.3.15/ 1073
4002_404C	Rx FIFO Information Register (CAN0_RXFIR)	32	R	Undefined	44.3.16/ 1074
4002_4880	Rx Individual Mask Registers (CAN0_RXIMR0)	32	R/W	Undefined	44.3.17/ 1075
4002_4884	Rx Individual Mask Registers (CAN0_RXIMR1)	32	R/W	Undefined	44.3.17/ 1075
4002_4888	Rx Individual Mask Registers (CAN0_RXIMR2)	32	R/W	Undefined	44.3.17/ 1075
4002_488C	Rx Individual Mask Registers (CAN0_RXIMR3)	32	R/W	Undefined	44.3.17/ 1075

Figure: FlexCAN mapping for K20 family

What a MCU is ? - Peripheral Access (3/3)

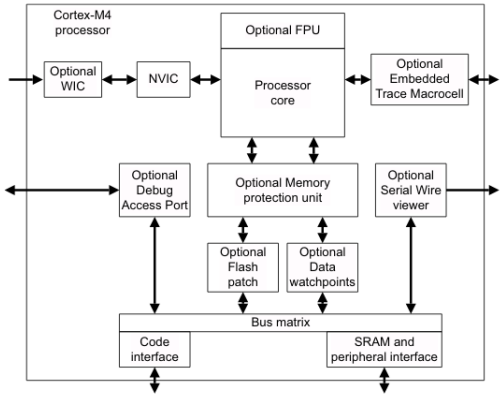


Figure: Peripherals overview for Cortex M4

Loading Firmware - Loader: its benefits

When reversing a raw MCU firmware of a known architecture

- If documentation is available, a lot of information can be extracted.
  - One of the most important: the **base address**.
  - Other information such as specific controller structures:
    - **interrupt vectors**.
    - **peripheral mappings**.
    - their associated **control registers**.
- Any other information that can put light on immediate or operand values.
- Putting all these information together, a dedicated loader can be written.

Loading Firmware - Loader: what is does

0x0000_0000–0x07FF_FFFF	Program flash and read-only data (Includes exception vectors in first 1024 bytes)	All masters
0x0800_0000–0x0FFF_FFFF	Reserved	—
0x1000_0000–0x13FF_FFFF	FlexNVM	All masters
0x1400_0000–0x17FF_FFFF	FlexRAM	All masters
0x1800_0000–0x1BFF_FFFF	Reserved	—
0x1C00_0000–0x1FFF_FFFF	SRAM_L: Lower SRAM (ICODE/DCODE)	All masters
0x2000_0000–0x200F_FFFF	SRAM_U: Upper SRAM bitband region	All masters
0x2010_0000–0x21FF_FFFF	Reserved	–
0x2200_0000–0x23FF_FFFF	Aliased to SRAM_U bitband	Cortex-M4 core only
0x2400_0000–0x3FFF_FFFF	Reserved	–
0x4000_0000–0x4007_FFFF	Bitband region for peripheral bridge 0 (AIPS-Lite0)	Cortex-M4 core & DMA/EzPort
0x4008_0000–0x400F_EFFF	Bitband region for peripheral bridge 1 (AIPS-Lite1)	Cortex-M4 core & DMA/EzPort
0x400F_F000–0x400F_FFFF	Bitband region for general purpose input/output (GPIO)	Cortex-M4 core & DMA/EzPort

(a) Memory Mapping for K20 family

Address	Name	Type	Required privilege	Reset value
0xE000E100-0xE000E11C	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000
0XE000E180-0xE000E19C	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000
0XE000E200-0xE000E21C	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000
0XE000E280-0xE000E29C	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000
0xE000E300-0xE000E31C	NVIC_IABR0-NVIC_IABR7	RW	Privileged	0x00000000
0xE000E400-0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000
0xE000EF00	STIR	WO	Configurable[a]	0x00000000

(b) NVIC registers for Cortex M4

What is loading?

- Raw firmware does not contain directly **interpretable metadata information**.
- Loader tells SRE how to **parse** a specific file, being well documented (ELF) or unknown (our case).
- Loading can be done **manually**, especially when you don't have information on the target (no documentation, header files, ...)

Loading Firmware - Loader: the benefits

What loader does?

- It is mapping firmware inside address space of your SRE.
- It is creating segments corresponding to memory layout of execution environment (not necessary included in the binary file).
- It is renaming addresses, provide information, creating structures, ...

What are the benefits?

- Mapping all the address space will **enhance** cross-references detection.
- Address **names** will be also provided to **disassembly output**.
- It helps to **infer code** purpose and **avoid wasting** time in analysing useless code.
- The more information you provide, the less you will have to go back and forth with your documentation / header files.
- Adding information after binary has been loaded would sometimes require to manually spread the information.

Loading Firmware - Base address

How to find a base address ?

- Use binbloom[4] tool.
- How it's works ? Several heuristic can be used:
  - Detect point of interest such data strings, pointer arrays, functions, other valuable heuristic.
  - Search pointers referencing these elements, numerous candidates bases addresses will be identified.
  - From all the candidates, select the one that reference the more points of interest in the firmware.

[4] https://github.com/quarkslab/binbloom

TP3 - Write your Loader

Materials for the exercise

- Refer to skeleton available *Exercices/LoaderIDA/teensy32_K20_cortexM4.py*
- Use previous converted binary file.

Steps[5]

- Fingerprint your firmware **accept_file()**
- Provide processor type inf: **set_processor_type()**
- Map segments:**add_segm()**, fix permissions: **fix_perm()**
- Renames addresses: **set_name()**, add comments: **set_cmt()**
- Map binary file in SRE address space: **file2base()**
- Put your file in your *$IDAHOME/loaders/* directory
- Open your firmware with IDA PRO

[5] https://hex-rays.com/products/ida/support/idapython_docs/

TP4 - TL: Enhance Functions Detection

Steps

- Propose a way to detect function missed by SRE engine.
- Implement it using SRE' API.
- Compare your results with the other peoples.

Correction

See **Corrections/IDA/functions_detector.py**.

What a MCU is ? - Interrupt Controller Overview

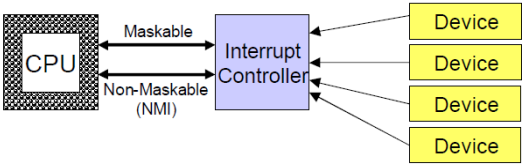


Figure: Overview of a generic Interrupt Controller

What a MCU is ? - Interrupt Controller Internals

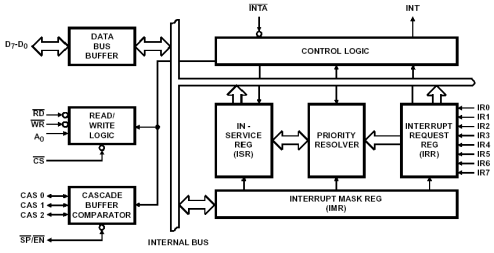


Figure: Internals of a generic Interrupt Controller[6]

[6] http://fundasbykrishna.blogspot.com

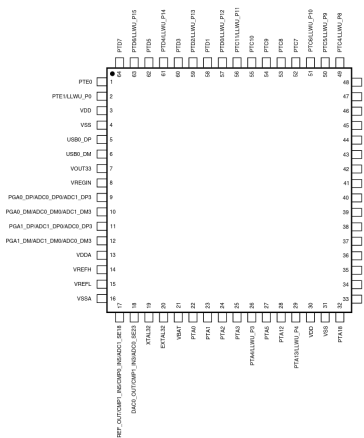
What a MCU is ? - Exceptions and Interruptions

Definition might change according to the architecture

- Exceptions and Interruptions can be software and/or hardware driven.
- Exceptions are generally errors to be handled
- Interruption can be generated by software or by hardware
- Interruption are generally configurable:
  - they can be nested
  - they can be temporary/permanently (de)activated / ignored
  - can be associated to a priority[7]
- Exceptions are generally not ignorable.
- Interruptions came from different sources (internal core(s), external modules plugged on interrupt controller)

[7] see Scheduling

Pin Configuration - Pins and alternate functions



(a) Ports of 64pins MCU from NXP K20 family

64 LQFP _QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
24	PTA2	JTAG_TDO/ TRACE_SWO EZP_DO	TSI0_CH3	PTA2	UART0_TX	FTM0_CH7				JTAG_TDO/ TRACE_SWO	EZP_DO
25	PTA3	JTAG_TMS/ SWD_DIO	TSI0_CH4	PTA3	UART0_RTS_b	FTM0_CH0				JTAG_TMS/ SWD_DIO	
26	PTA4/ LLWU_P3	NMI_b/ EZP_CS_b	TSI0_CH5	PTA4/ LLWU_P3		FTM0_CH1				NMI_b	EZP_CS_b
27	PTA5	DISABLED		PTA5	USB_CLKIN	FTM0_CH2		CMP2_OUT	USB_TX_BCLK	JTAG_TRST_b	
28	PTA12	CMP2_IN0	CMP2_IN0	PTA12	CAN0_TX	FTM1_CH0			USB_TXOE	FTM1_QD_ PHA	
29	PTA13/ LLWU_P4	CMP2_IN1	CMP2_IN1	PTA13/ LLWU_P4	CAN0_RX	FTM1_CH1			USB_TX_FS	FTM1_QD_ PHB	
30	VDD	VDD	VDD								
31	VSS	VSS	VSS								
32	PTA18	EXTAL0	EXTAL0	PTA18		FTM0_FLT2	FTM_CLKIN0				
33	PTA19	XTAL0	XTAL0	PTA19		FTM1_FLT0	FTM_CLKIN1		LPTMR0_ALT1		
34	RESET_b	RESET_b	RESET_b								
35	PTB0/ LLWU_P5	ADC0_SE8/ ADC1_SE8/ TSI0_CH0	ADC0_SE8/ ADC1_SE8/ TSI0_CH0	PTB0/ LLWU_P5	I2C0_SCL	FTM1_CH0			FTM1_QD_ PHA		
36	PTB1	ADC0_SE9/ ADC1_SE9/ TSI0_CH6	ADC0_SE9/ ADC1_SE9/ TSI0_CH6	PTB1	I2C0_SDA	FTM1_CH1			FTM1_QD_ PHB		
37	PTB2	ADC0_SE12/ TSI0_CH7	ADC0_SE12/ TSI0_CH7	PTB2	I2C0_SCL	UART0_RTS_b			FTM0_FLT3		
38	PTB3	ADC0_SE13/ TSI0_CH8	ADC0_SE13/ TSI0_CH8	PTB3	I2C0_SDA	UART0_CTS_ b/ UART0_COL_b			FTM0_FLT0		
39	PTB16	TSI0_CH9	TSI0_CH9	PTB16		UART0_RX		FB_AD17	EWM_IN		
40	PTB17	TSI0_CH10	TSI0_CH10	PTB17		UART0_TX		FB_AD16	EWM_OUT_b		
41	PTB18	TSI0_CH11	TSI0_CH11	PTB18	CAN0_TX	FTM2_CH0	I2S0_TX_BCLK	FB_AD15	FTM2_QD_ PHA		
42	PTB19	TSI0_CH12	TSI0_CH12	PTB19	CAN0_RX	FTM2_CH1	I2S0_TX_FS	FB_OE_b	FTM2_QD_ PHB		
43	PTC0	ADC0_SE14/ TSI0_CH13	ADC0_SE14/ TSI0_CH13	PTC0	SPI0_PCS4	PDB0_EXTRG		FB_AD14	I2S0_TXD1		
44	PTC1/	ADC0_SE15/	ADC0_SE15/	PTC1/	SPI0_PCS3	UART1_RTS_b	FTM0_CH0	FB_AD13	I2S0_TXD0		

(b) Alternate functions for these pins

What a MCU is ? Port configuration

Port have numerous capacities (alternate function)

- A port can only handle one function at the same time
- Each port generally benefits from one or several control register:
  - The port function can be defined (analog, GPIO, other support protocols)
  - Associated interruption(s) can be configured
  - Specific hardware behavior can be set (pull up, pull down, ...)
- All the features offered by a MCU may not be usable at once.

11.14.1 Pin Control Register n (PORTx_PCR*n*)

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LK	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

(a) Ports' Generic Control Register definition for K20 Family

10–8 MUX	Pin Mux Control

Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot.

The corresponding pin is configured in the following pin muxing slot as follows:

000	Pin disabled (analog).
001	Alternative 1 (GPIO).
010	Alternative 2 (chip-specific).
011	Alternative 3 (chip-specific).
100	Alternative 4 (chip-specific).
101	Alternative 5 (chip-specific).
110	Alternative 6 (chip-specific).
111	Alternative 7 (chip-specific).

(b) Definition of pin function in control register

What a MCU is ? - Scheduler

MCU must run several tasks in parallel

- It often manages several peripherals that need to access other internal resources, roughly called ”hardware scheduling”
- It generally has to manage several little programs who access internal resources and peripherals.
- Conflicts can happen when several softwares or/and hardware objects request the same resource.
  - Strategy must be defined to grant/postpone/deny access to the resources.
  - It resides on strategy based on elements such as operation criticality.
  - Various scheduling polices[8] exist from the simplest to more complex.
- Scheduling are generally interrupt driven, standing on MCU features.
  - **Systick** or more generally (internal or external) clocks.
  - Watchdog.
  - Any other resources able to generate cycled interrupt.

[8] https://en.wikipedia.org/wiki/Scheduling_(computing)Scheduling_disciplines

TP5 - Analyse Scheduler

Steps

- Study interruptions that could provide scheduling feature.
- Use the loader.
- Search strings **attribute**, **no-return**, ...
- Could you explain how scheduling seems to occur ?

What a MCU is ? - Memory Management

Several Memory Units

- MMU : Memory Management Units
  - Allow to set protection on physical address ranges.
  - Exception (kind of interrupt) are generated when a fault is detected.
  - Can support various permission (combination or R/W/X).
- MPU : Memory Protection Unit
  - Support memory virtualization[9]
  - Used by OSes to optimize memory management operations (access, control,...).
  - MMU is more expensive and more complex to setup.
  - More evolved unit can even manage external peripheral accesses.

[9] https://en.wikipedia.org/wiki/Virtual_memory

What a MCU is ? - CPU Modes

MCU or more generally CPU offers memory right management capacities

- Two or more operating modes can be used (ex: user/system).
- In MCU world it intends to protect critical resources from a unstable software / weird glitches rather than providing security.
- To switch from an operating mode to another dedicated instructions are offered.
  - These interruptions generate interruption or exception (according the architecture).
  - To enter the interrupt routine context-saving[10] (implicit or explicit) mechanism is used.
  - The current operating mode is generally stored inside a system register.
  - To resume execution, associated instruction is used, ensuring implicit or not context-restoring.
- Cheap, old and tiny MCU generally uses a single mode, the more privileged one.
- Finding a bug in a driver, or any reachable routine could have disastrous consequences in such situation.

[10] https://stackoverflow.com/a/21741290

CPU Modes

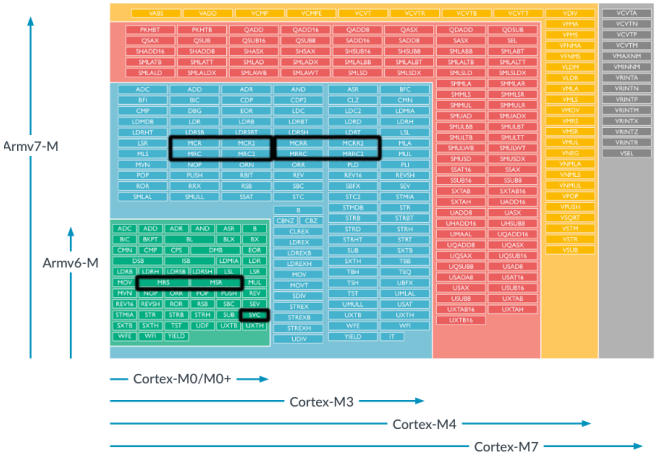


Figure: ARM v6-7 Cortex M Instruction Set

TP6 - Right management

Steps

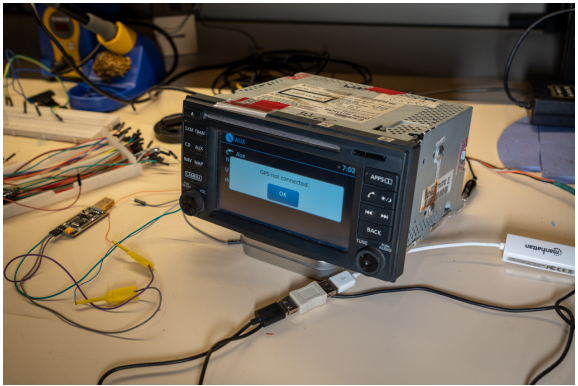
- Find out the privileged instructions for ARM Cortex M4.
- Search usage of such instruction (**MRS**, **MSR**).
- Do you see any right management ? (Operation on PSR bit mode)

What a ECU is ? - Electronic in a car

Electronic in car is all about:

- **Sensors**: give information about specific parameters
- **Actuators**: actions/motions on specific element
- **ECU**: Electronic Control Unit , uses numerous sensors and actuators
- **User Interface / UX**: Give feedback on the user. Meters, Display Panels
- Nowadays, **80** to more than **100** ECUs are embedded in a **single** car!

What a ECU is ? - Example of Nissan IVI



(a) IVI Nissan Overview[a]

[a] https://github.com/ea/bosch_headunit_root

What a ECU is ? Communications

ECUs are interconnected

- ECUs need to exchange **a lot** of information.
- Several protocols exist:
  - **Flexray**[11]: 20Mb/s
  - **KWP2000**[12]: replaced by CAN
  - **CAN** and **CAN-FS**: widely used on a car, mandatory in the US
  - **Ethernet**: physical layer (100BaseTX) differs from the classical laptop (10BaseT)

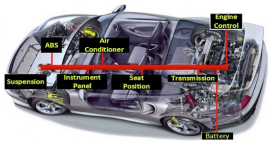


Figure: Internals of a generic Interrupt Controller

[11] https://fr.wikipedia.org/wiki/FlexRay
[12] https://en.wikipedia.org/wiki/Keyword_Protocol_2000

What a ECU is ? CAN Trames (1/2)

CAN is an old protocol patented by Bosch

- It is an old and quite simple protocol in comparison to other protocols such as Ethernet.
- It uses **two wires** following **NRZ**[13] format (don't use neutral voltage).
- It is **half-duplex**, the two lines are simultaneous used for a single communication.
- It is **stateless** protocol.

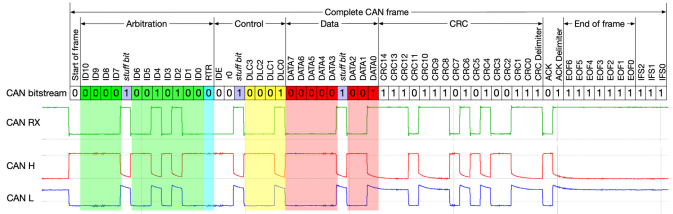


Figure: CAN Frame[14]

[13] https://fr.wikipedia.org/wiki/Non_Return_to_Zero

What a ECU is ? CAN Trames (2/2)

(a) Standard CAN Frame

(b) Extended CAN Frame

Not a lot of fancy things, only to remind

- Arbitration ID: the destination address to the end node.
  - Can be **standard** (STD)
  - Can be **extended** (EXT)
- A frame can carry at maximum **8 bytes** of data, the data is indicated in **DLC** field.

[15] https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/

TP7 - RE: Identify CAN information

Materials for the exercise

- Use your loader to and scripts (or one available among Correction folder)
- AMIE[16]

Steps

- Find ISR handlers related to CAN.
- Identify relevant structures.
- Use cross-references to identify other functions related to CAN.

[16] https://github.com/NeatMonster/AMIE.git

CAN MailBox layout

The memory area from 0x80 to 0x47C is used by the Mailboxes.

Table 44-69. Message buffer structure

	31 30 29 28	27 24	23	22	21	20	19 18 17 16	15 8	7 0
0x0		CODE		SRR	IDE	RTR	DLC	TIME STAMP	
0x4	PRIO	ID (Standard/Extended)						ID (Extended)	
0x8	Data Byte 0		Data Byte 1					Data Byte 2	Data Byte 3
0xC	Data Byte 4		Data Byte 5					Data Byte 6	Data Byte 7
	= Unimplemented or Reserved								

Figure: MailBox layout

CAN Buffers

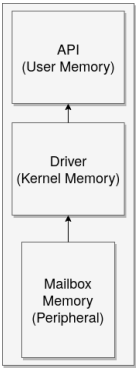


Figure: MailBox layout

TP8 - RE: Identify CAN buffers (1/2)

Goal

Identify CAN related buffers corresponding to the different layers: peripheral, driver and applicative.

Steps

- Identify the input and the output elements. (think registers)
- Use shortcut (Alt Top/Down arrows, k) to navigate and tags
- Create enum, structures to clarify (mailbox layout, mailbox status,...)

TP8 - RE: Identify CAN buffers (2/2)

Emualtion

- Using emulation, with specially crafted data:
  - Overall CAN structure located at 0x1FFFA0B8. Emulate initialization function then output the structure to a file.
  - Mailbox can be generated using documentation, then stored in emulator at address 0x40024080. Pay attention to the status (FULL, OVERRUN, ...)
  - IFLAG1 register can be set to 0x00000001 (interrupt for mailbox num 0).
- At the end you can track your data and see how it's past to the upper level (ISOTP).
- What about function at 0x13AC ?

Exec Trace - Explaination

What we learn

- The input data **AAAAAAAA** is finally copied at **145B** and **145D**.
- It is copied to a erroneous offset, most probably because of wrong memory initialization.
- From here:
  - Look at the argument of the *memcpy*.
  - Find out what is the destination buffer.
  - You have probably identify the upper buffers (that will be used by the applicative).
  - At the end the corresponding address seems to be **0x1FFFA0B8** + **0xA4** + **8**.
  - Search regex *.*0xA4.**, and find out where the data might be used.

Taint Analysis

What is taint analysis

- Taint Analysis use DSE engine to track instruction that manipulate specific addresses or memory ranges.
- DSE is a technique combining concrete execution alongside symbolic execution
- DSE engine are complex and cannot be applied to every use-case.

Example of tools managing taint analysis

- Triton [17]
- Angr [18]
- miasm [19] see depgraph.
- manticore [20]

[17] https://github.com/JonathanSalwan/Triton
[18] https://github.com/angr/angr
[19] https://github.com/cea-sec/miasm/
[20] https://github.com/trailofbits/manticore/

TP10 - TL: Implement Taint Analysis engine

Goal

Using Taint Analysis, track every instruction using mailbox in the CAN ISR handler.

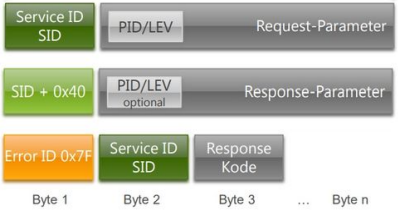
Steps

- Use skeleton available at **Exercices/Triton/taint.py**.
- Fill the required CAN, Mailbox and IFLAG structures.
- Run the engine.

What a ECU is? Diagnostic on Car

UDS stands for Unified Diagnostic System

- A standard exists and is used by *all* car manufacturers (ISO 14229-1.).
- It defines several **services**[21]:
  - Security Access (SA): intended to unlock critical diagnostics features
  - Routine Control: here OEM, Tier-1 can use their own unstandardized features.
  - Read/Write Data by ID: access some data.
  - Transfer Data: send raw blob of data (involved in ECU update)



[21] https://en.wikipedia.org/wiki/Unified_Diagnostic_ServicesServices

What a ECU is? ISOTP (1/3)

ISOTP is a fragmentation/reassembly protocol

- CAN frames can only contains 8 bytes of data.
- ISOTP defines a way to carry bigger payload of data (until 4096 bytes).
- UDS protocol relies on ISOTP.

Key points

- 4 types of frame (Single Frame, First Frame, Consecutive Frame, Flow Control).
- Kind of frame is defined by its first byte of payload.
- Messages are always padded and must be 8 bytes longs.

What a ECU is ? ISOTP (2/3)

CAN-TP Header

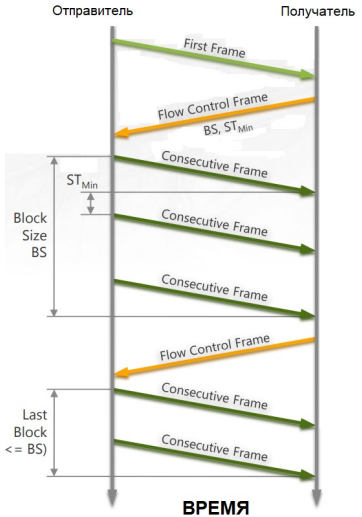
Bit offset	7 .. 4 (byte 0)	3 .. 0 (byte 0)	15 .. 8 (byte 1)	23..16 (byte 2)	....
Single	0	size (0..7)	Data A	Data B	Data C
First	1	size (8..4095)		Data A	Data B
Consecutive	2	index (0..15)	Data A	Data B	Data C
Flow	3	FC flag (0,1,2)	Block size	ST	

Figure: ISOTP Frame Types

Constants

- 4096 bytes is the max size
  - N_PCI_SF = 0
  - N_PCI_FF = 0x10
  - N_PCI_CF = 0x20
  - N_PCI_FC = 0x30

What a ECU is? ISOTP (3/3)



TP11 - RE: Identify the ISOTP main routine

Goal

Find the ISOTP main routine based on the knowledge you can have.

Steps

1. Find ISOTP related constants
2. Identify instructions could be used to perform constant comparisons
3. Use search function to locate specific constant
4. Try to identify the layer' buffers

TP ISOTP

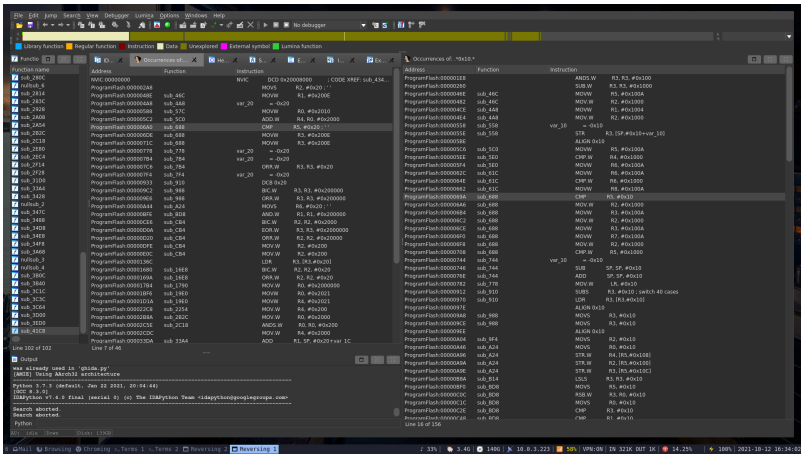


Figure: Crossing the constant locations

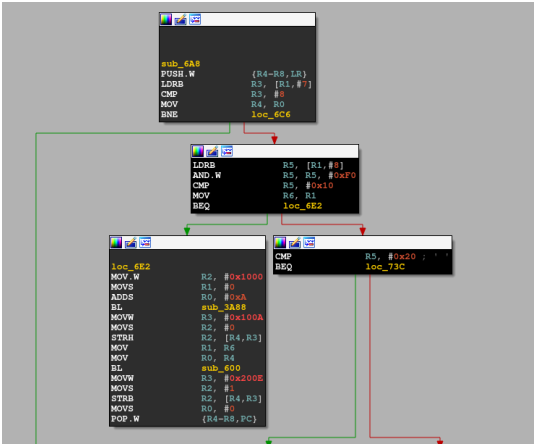


Figure: ISOTP handler

TP12 - RE: Identify the UDS main routine

Goal

Find the UDS handler.

Steps

- Identify the Service ID constants.
- Find a function using several of these constants (manually or using a script).

Change firmware

Due to compilation option mistake, for this step uses firmware located at *Firmwares/Teensy_UDS/*. Apologies..

UDS Handler

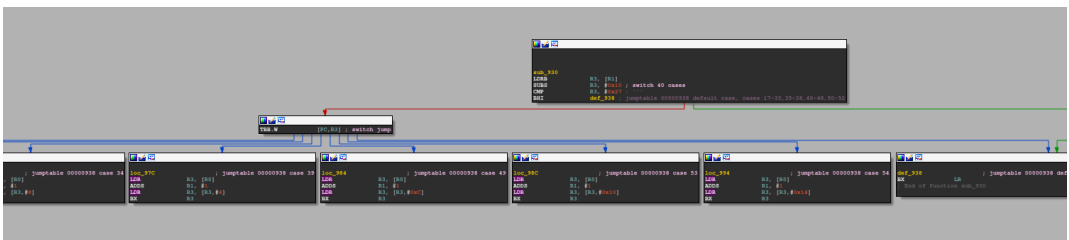


Figure: ISOTP handler

TP13 - RE: Identify the Security Access algorithm

Goal

Find the UDS handler

Steps

- Identify the Service ID constants.
- Identify which keyword could help you.
- Find a function using several of these constants[22].

Change firmware

Due to compilation option mistake, for this step uses firmware located at *Firmwares/Teensy_UDS/*. Apologies..

[22] https://github.com/polymorf/findcrypt-yara

TP14 - TL: Write a fuzzer harness

Goals

- Find out which CAN related components may be interesting to fuzz.
- Write accordingly a fuzzer harness.

TP14-1 Fuzzing: CAN Driver case

Points of Interests

- The inputs, how it is mutated. (cf. mailbox layout)
- Crash detection, how to handle it in the context. (stack buffers)

Materials

- You can use *Exercices/Fuzzing/isr_can_xor/isr_can_xor_harness.py*

TP14-2 Fuzzing: ISOTP case

Fuzzing network protocols

- Network protocols are difficult when it comes to fuzzing.
- What kind of data to test reassembly mechanism ?
  - Program need to get several 8 bytes payload.
  - The full buffer could be better in this case.
  - Could you cheat and can fill all the mailboxes using the data provided by AFL?
- Firmware can be patched in order to avoid calling material related functions.

Fuzzing : Designing the harness

Fuzzing the harness

- Use coverage to see where the struggle is:
  - is it related to uninitialized data?
  - is it related to useless function?
  - do you need to stub or emulate the behavior?
- Add tracing information to your emulation script.

Materials

- You can use *Exercices/Fuzzing/can_handler_python/can_handler_4A8_fuzz.py*

Coverage information

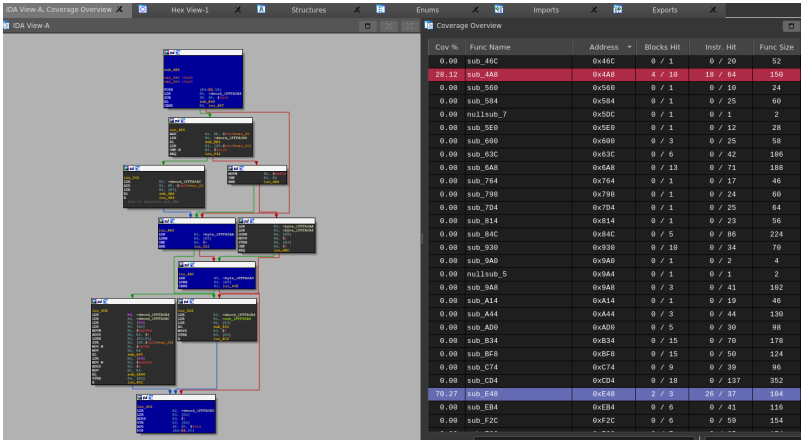


Figure: Visualize coverage with lighthouse[23]

[23] https://github.com/gaasedelen/lighthouse/tree/master/coverage

Thank you

Contact information:

Email: contact@quarkslab.com

Phone: +33 1 58 30 81 51

Website: https://www.quarkslab.com

Quarkslab