Introduction to Embedded Systems

3. Hardware Software Interface

Prof. Dr. Marco Zimmerling

NES NETWORKED EMBEDDED SYSTEMS LAB

UNI FREIBURG

Join the Course on ILIAS!

Access via: https://nes-lab.org/

→ Courses

→ Introduction to Embedded Systems

→ ILIAS

- Login: RZ username + password
- Course password: **es-0x8af**

Resources:

- Forum, course schedule, Zoom/BBB links, important announcements
- Slides and recording after each lecture
- Exercise sheets before each exercise
- Slides, recordings, and exercise solutions after each exercise

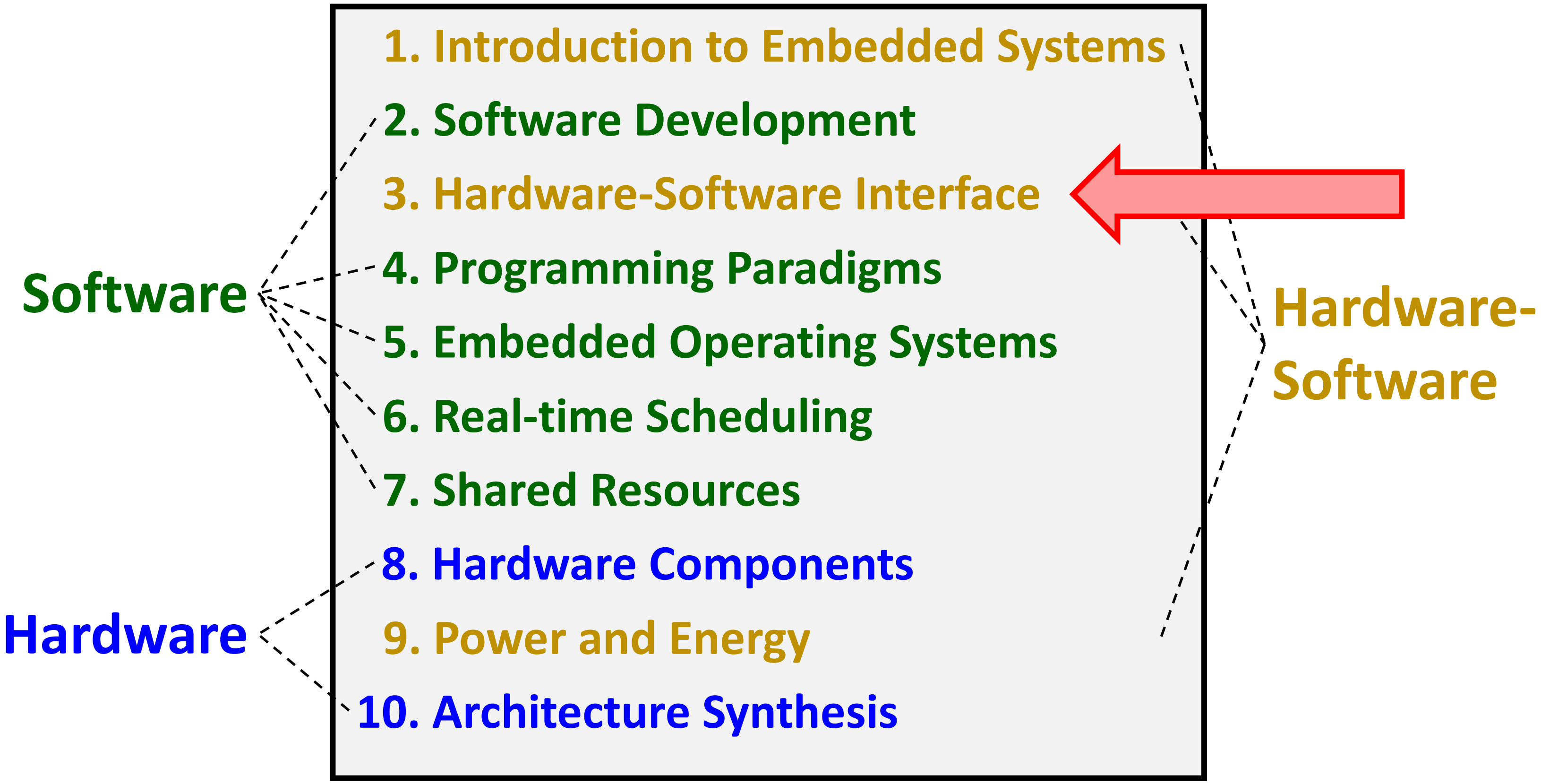
Schedule

Schedule:

- Today (October 25): No exercise
- Next week (November 1): All Saints’ Day
- In two weeks (November 8): Lecture and exercise
- **Check regularly on ILIAS for updates!**

Do you remember?

Where we are …

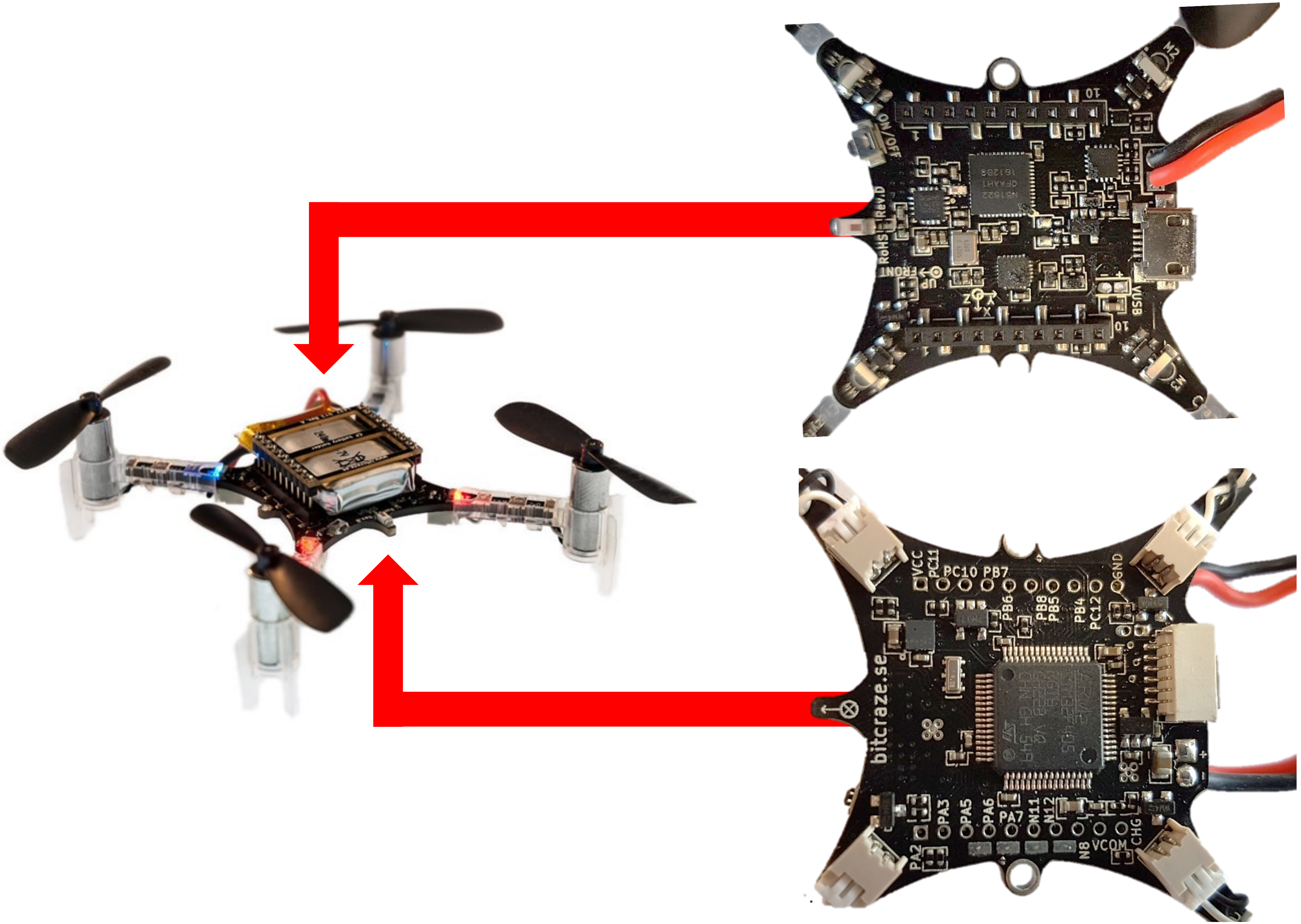


1. Introduction to Embedded Systems
2. Software Development
3. Hardware-Software Interface
4. Programming Paradigms
5. Embedded Operating Systems
6. Real-time Scheduling
7. Shared Resources
8. Hardware Components
9. Power and Energy
10. Architecture Synthesis

Software

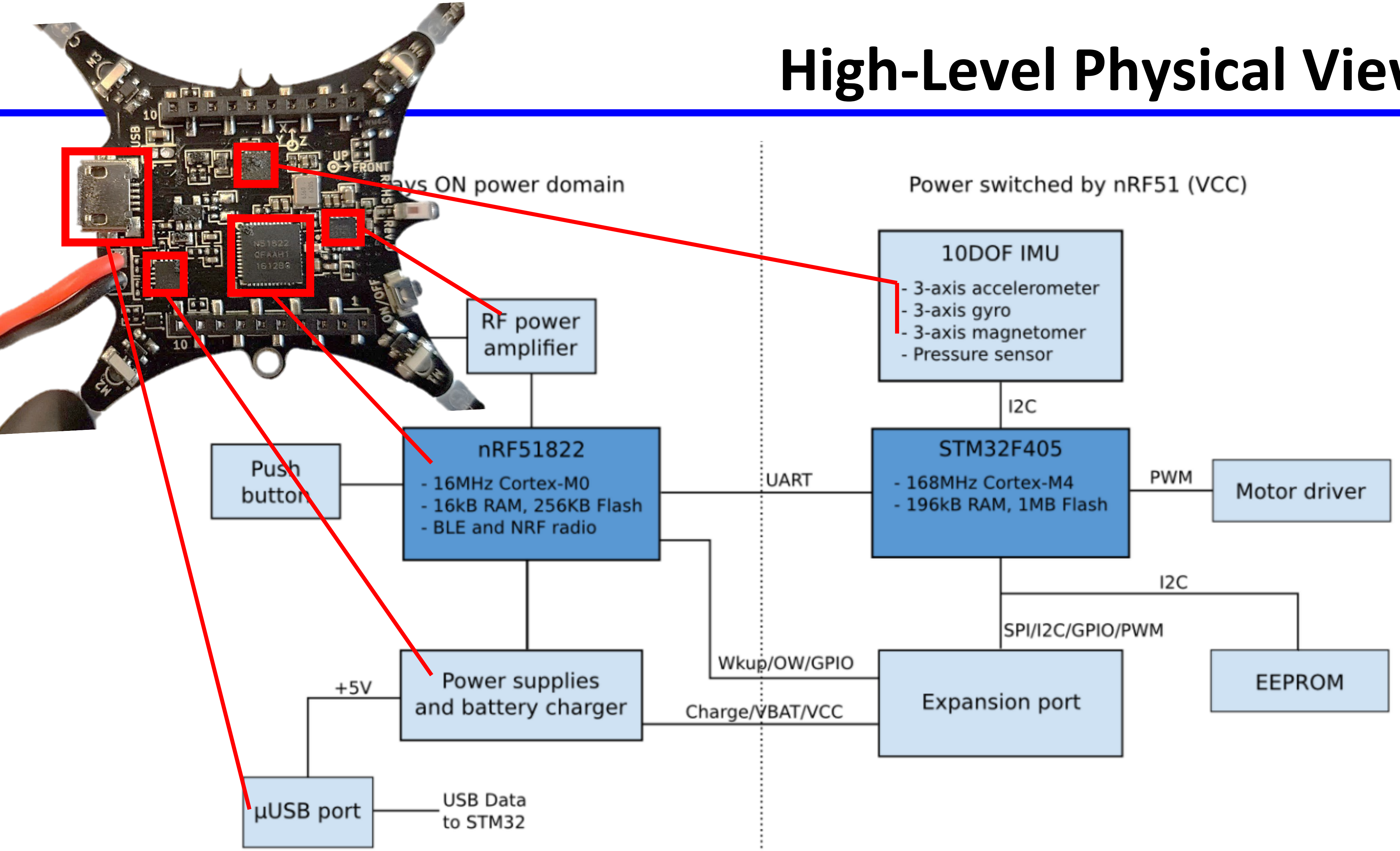
Hardware

Hardware-Software



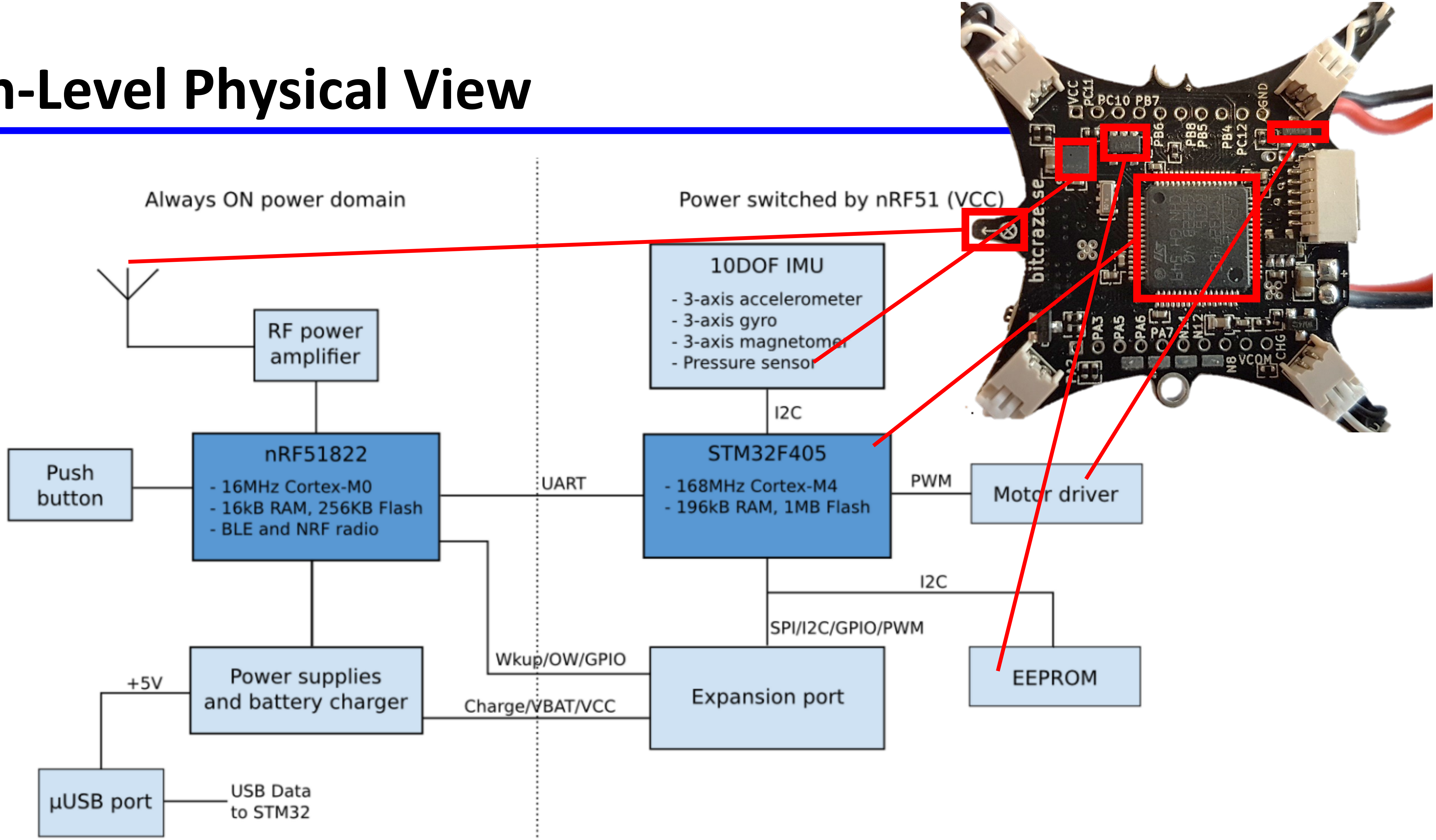
M1
M2
M3
ON/OFF
RoHS Rev.D
UP FRONT
VUSB
10
bitcraze.se
VCC
PC11
PC10 PB7
PB6
PB8
PB5
PB4
PC12
GND
PA2
PA3
PA5
PA6
PA7
N11
N12
N8
VCOM
CHG

High-Level Physical View



Crazyflie 2.0 system architecture

High-Level Physical View



Always ON power domain

Power switched by nRF51 (VCC)

RF power amplifier

Push button

nRF51822
- 16MHz Cortex-M0
- 16kB RAM, 256KB Flash
- BLE and NRF radio

UART

10DOF IMU
- 3-axis accelerometer
- 3-axis gyro
- 3-axis magnetometer
- Pressure sensor

I2C

STM32F405
- 168MHz Cortex-M4
- 196kB RAM, 1MB Flash

PWM

Motor driver

I2C

SPI/I2C/GPIO/PWM

EEPROM

Wkup/OW/GPIO

Power supplies and battery charger

+5V

Charge/VBAT/VCC

Expansion port

µUSB port

USB Data
to STM32

Crazyflie 2.0 system architecture

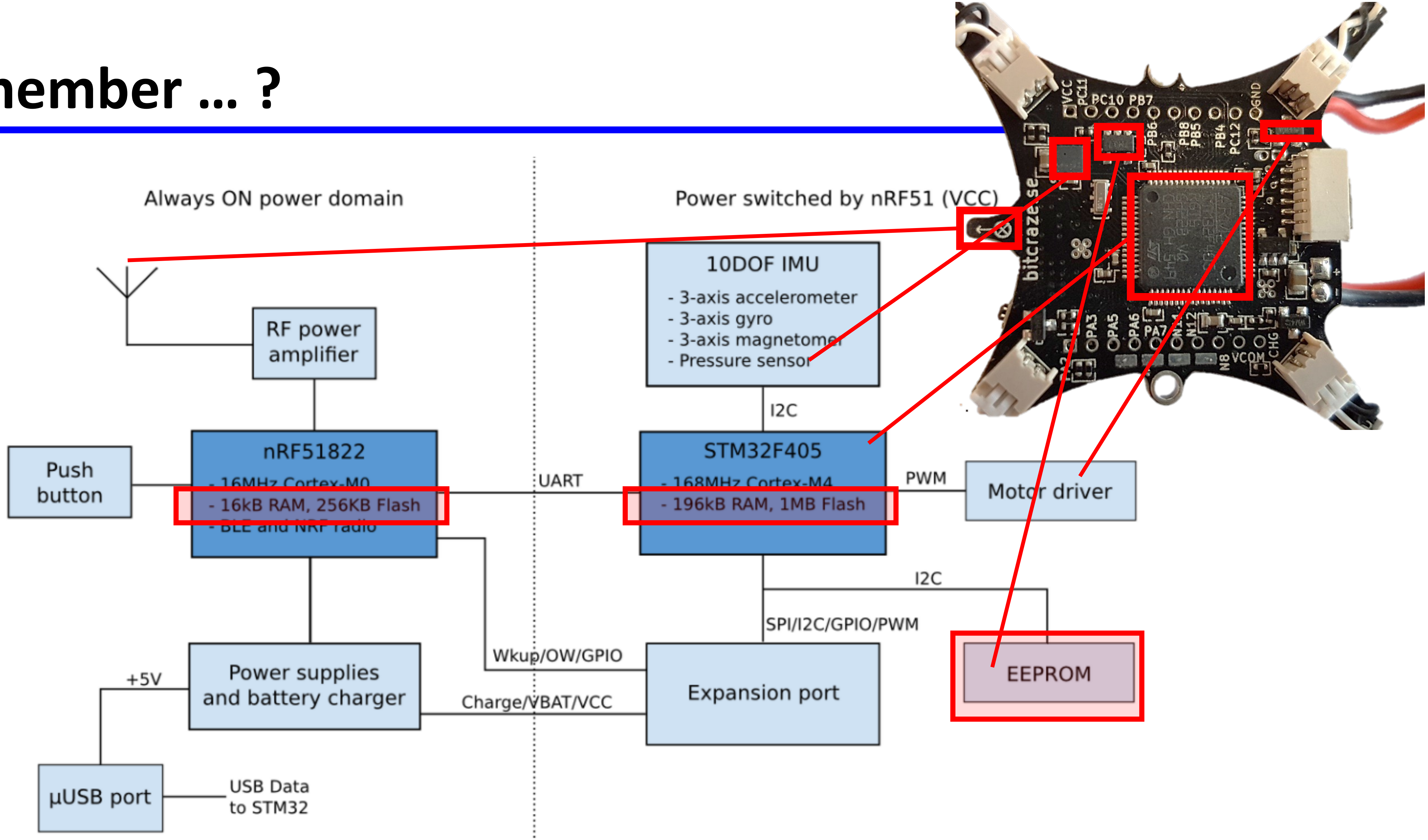
What you will learn …

Hardware-Software Interfaces in Embedded Systems

- *Storage*
  - SRAM / DRAM / Flash
  - Memory Map
- *Input and Output*
  - UART Protocol
  - Memory Mapped Device Access
  - SPI Protocol
- *Interrupts*
- *Clocks and Timers*
  - Clocks
  - Watchdog Timer
  - System Tick
  - Timer and PWM

Storage

Remember … ?



Always ON power domain

Power switched by nRF51 (VCC)

RF power amplifier

Push button

nRF51822
- 16MHz Cortex-M0
- 16kB RAM, 256KB Flash
- BLE and NRF radio

UART

STM32F405
- 168MHz Cortex-M4
- 196kB RAM, 1MB Flash

10DOF IMU
- 3-axis accelerometer
- 3-axis gyro
- 3-axis magnetometer
- Pressure sensor

I2C

PWM

Motor driver

I2C

SPI/I2C/GPIO/PWM

EEPROM

Wkup/OW/GPIO

Expansion port

Power supplies and battery charger

+5V

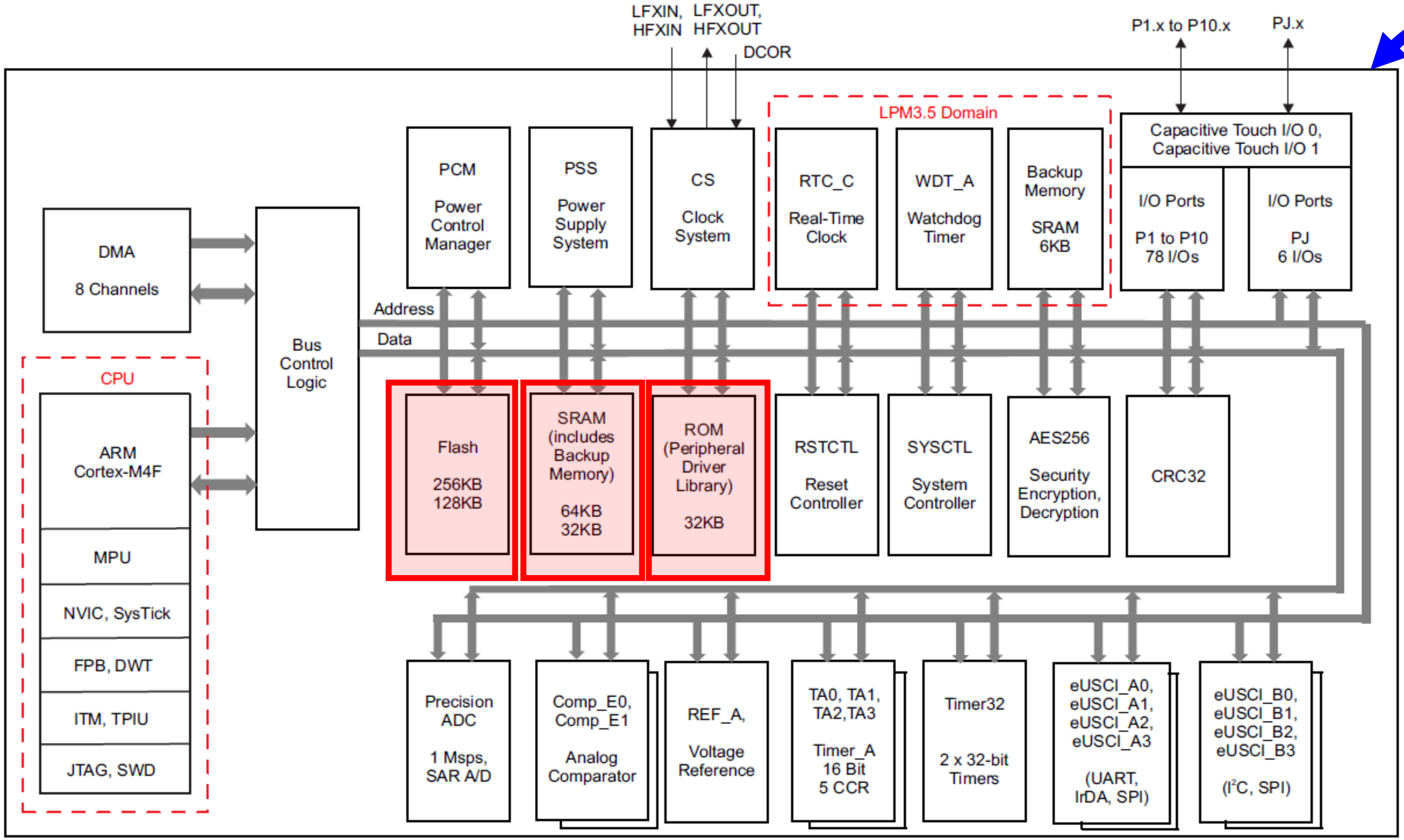
Charge/VBAT/VCC

µUSB port

USB Data to STM32

Crazyflie 2.0 system architecture

MSP432P401R



LFXIN, HFXIN

LFXOUT, HFXOUT

DCOR

P1.x to P10.x

PJ.x

DMA

8 Channels

Bus Control Logic

Address

Data

CPU

ARM Cortex-M4F

MPU

NVIC, SysTick

FPB, DWT

ITM, TPIU

JTAG, SWD

PCM

Power Control Manager

PSS

Power Supply System

CS

Clock System

LPM3.5 Domain

RTC_C

Real-Time Clock

WDT_A

Watchdog Timer

Backup Memory

SRAM 6KB

Capacitive Touch I/O 0, Capacitive Touch I/O 1

I/O Ports

P1 to P10 78 I/Os

I/O Ports

PJ 6 I/Os

Flash

256KB 128KB

SRAM (includes Backup Memory)

64KB 32KB

ROM (Peripheral Driver Library)

32KB

RSTCTL

Reset Controller

SYSCTL

System Controller

AES256

Security Encryption, Decryption

CRC32

Precision ADC

1 Msps, SAR A/D

Comp_E0, Comp_E1

Analog Comparator

REF_A,

Voltage Reference

TA0, TA1, TA2,TA3

Timer_A 16 Bit 5 CCR

Timer32

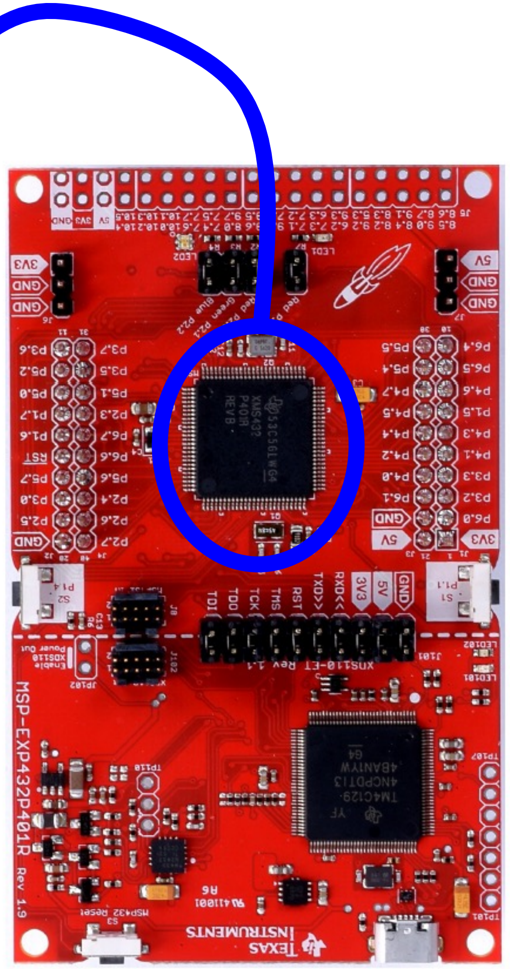
2 x 32-bit Timers

eUSCI_A0, eUSCI_A1, eUSCI_A2, eUSCI_A3

(UART, IrDA, SPI)

eUSCI_B0, eUSCI_B1, eUSCI_B2, eUSCI_B3

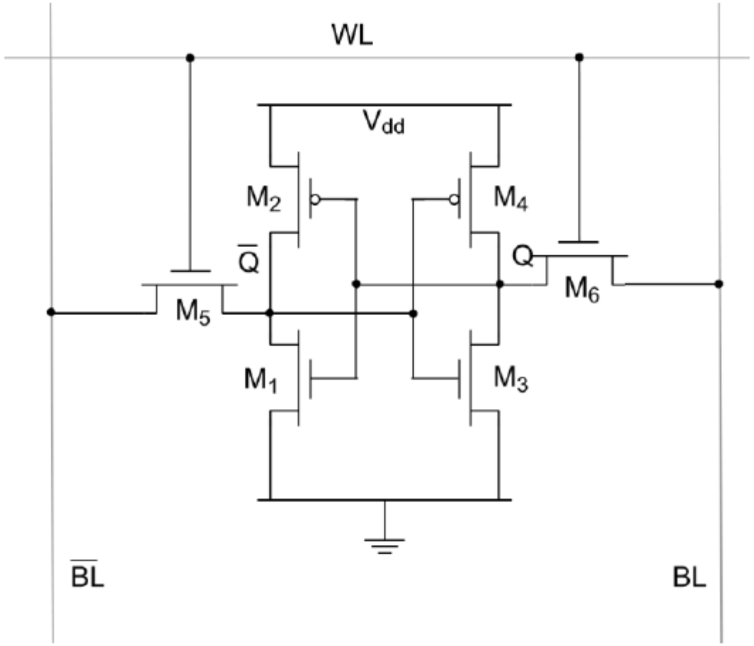
(I^2C, SPI)



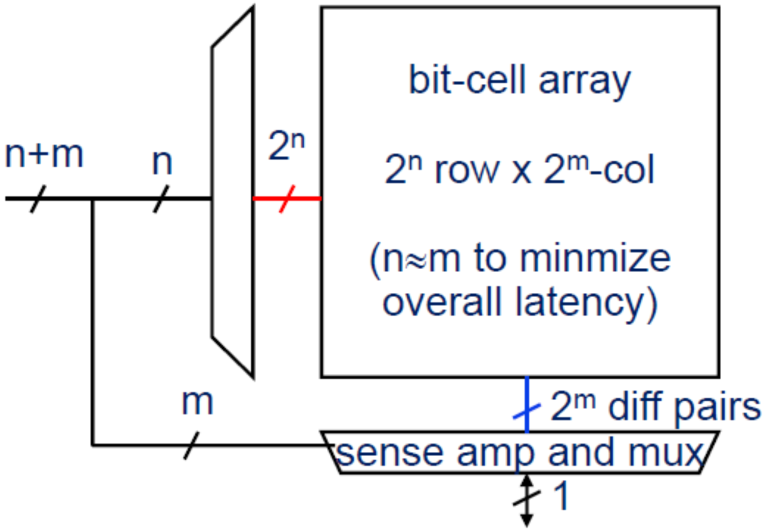
Storage

SRAM / DRAM / Flash

Static Random Access Memory (SRAM)



- *Single bit is stored in a bi-stable circuit*
- *Static Random Access Memory* is used for
  - caches
  - register file within the processor core
  - small but fast memories
- *Read:*
  1. Pre-charge all bit-lines to average voltage
  2. decode address (n+m bits)
  3. select row of cells using 2^n single-bit word lines (WL)
  4. selected bit-cells drive all bit-lines BL (2^m pairs)
  5. sense difference between bit-line pairs and read out
- *Write:*
  - select row and overwrite bit-lines using strong signals



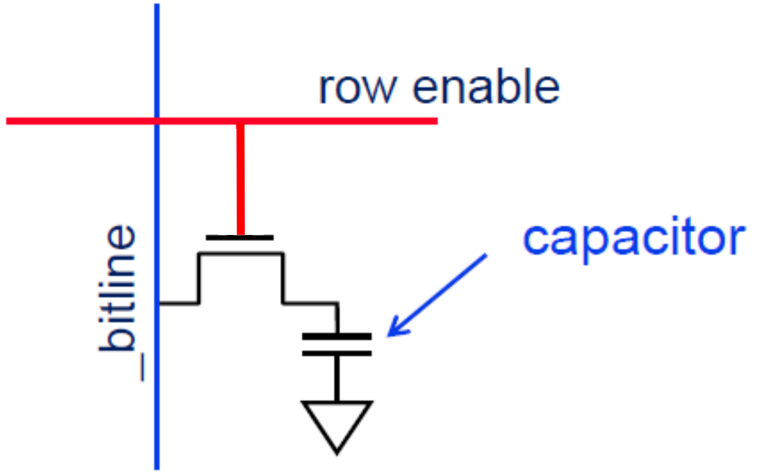
Dynamic Random Access (DRAM)

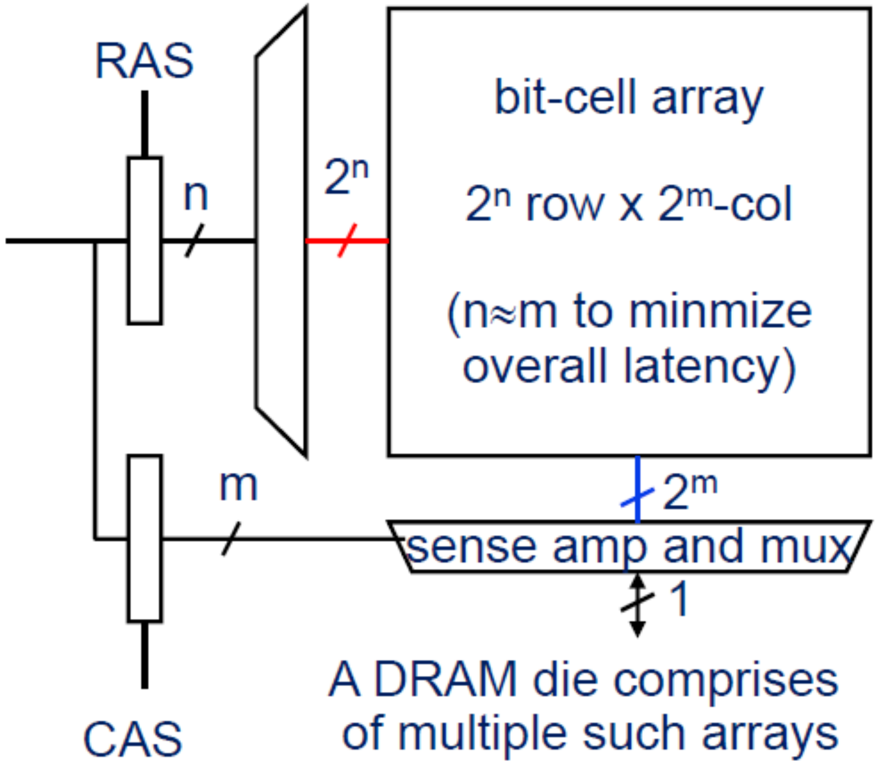
Single bit is stored as a charge in a capacitor

- Bit cell loses charge when read, bit cell drains over time
- Slower access than with SRAM due to small storage capacity in comparison to capacity of bit-line.
- Higher density than SRAM (1 vs. 6 transistors per bit)

DRAMs require *periodic refresh* of charge

- Performed by the memory controller
- Refresh interval is tens of ms
- DRAM is unavailable during refresh



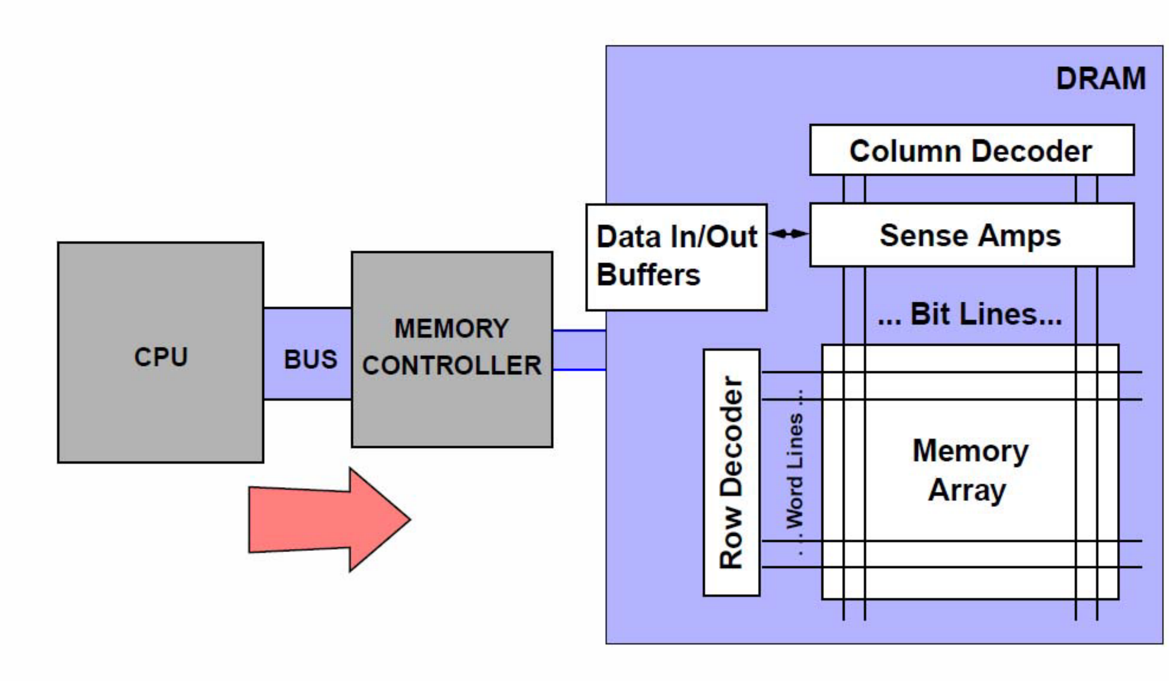


A DRAM die comprises of multiple such arrays

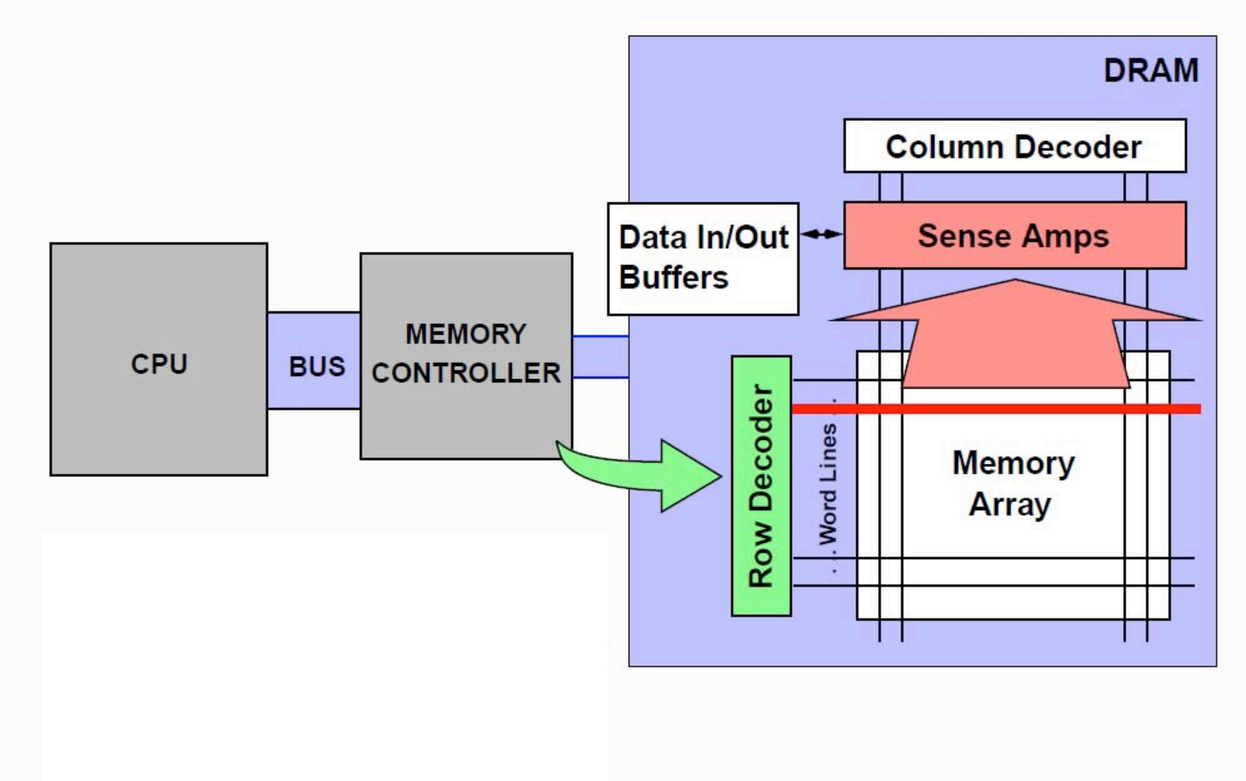
(RAS/CAS = row/column address select)

DRAM – Typical Access Process

1. Bus Transmission

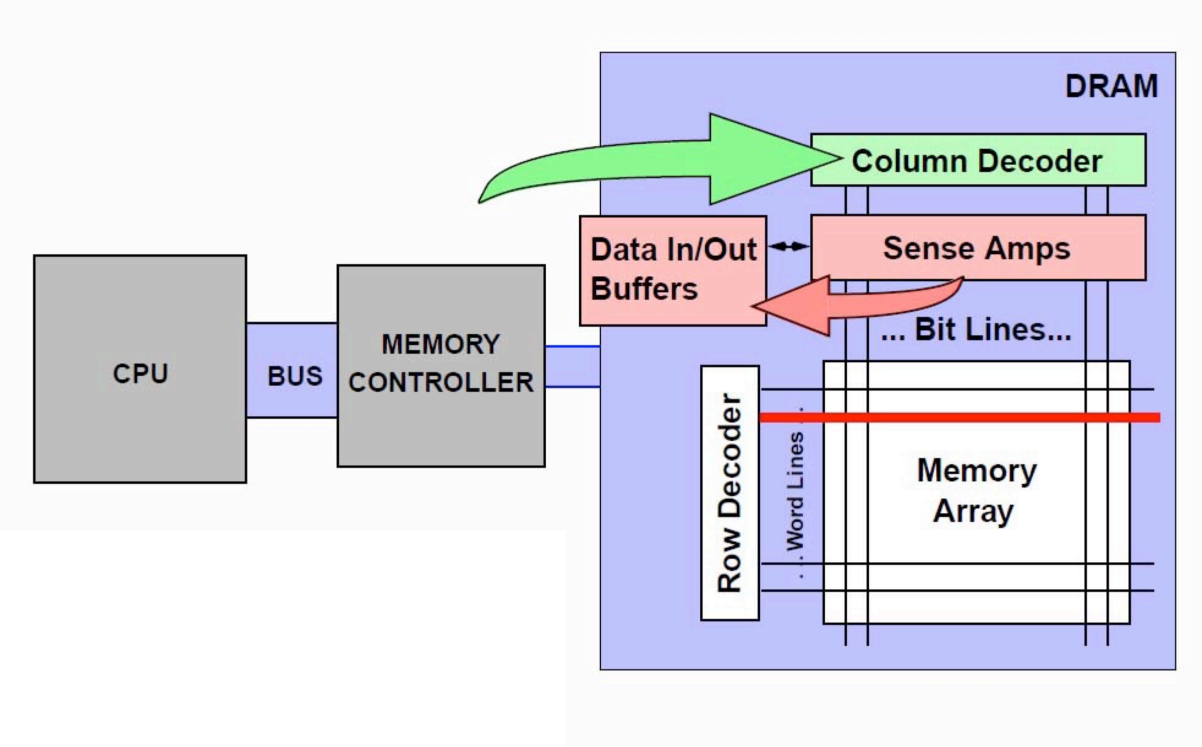


2. Precharge and Row Access

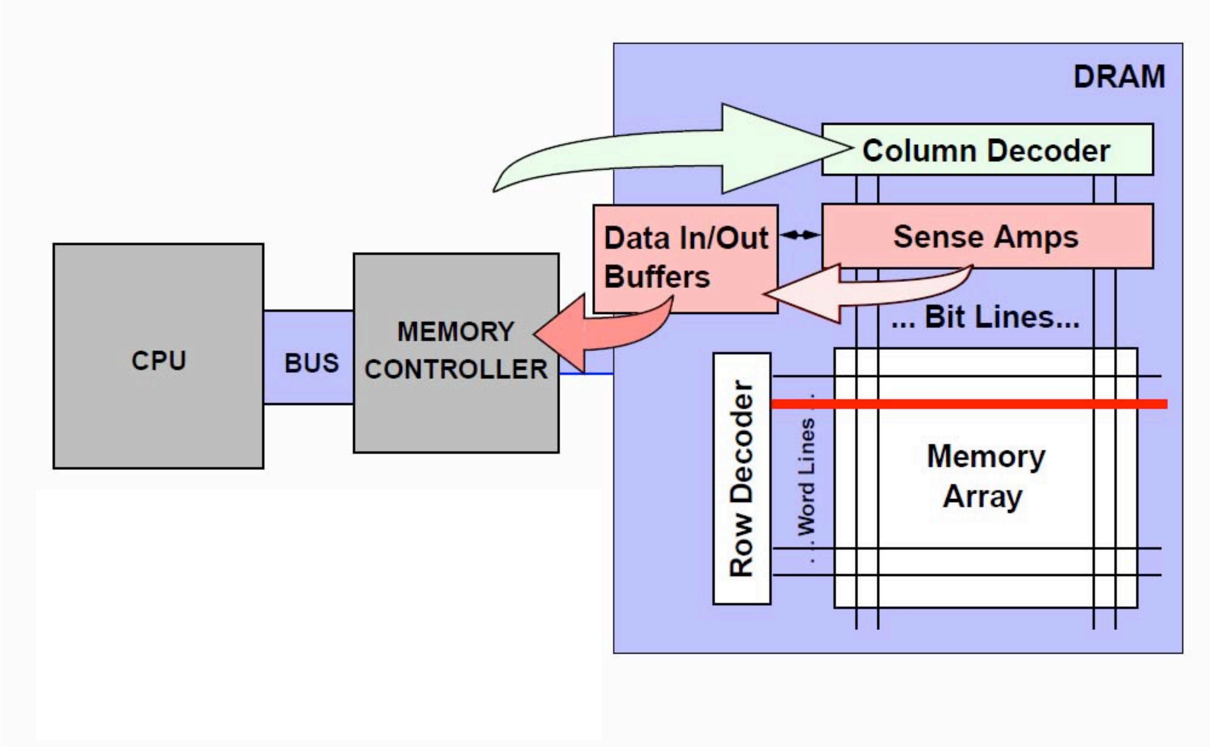


DRAM – Typical Access Process

3. Column Access



4. Data Transfer and Bus Transmission

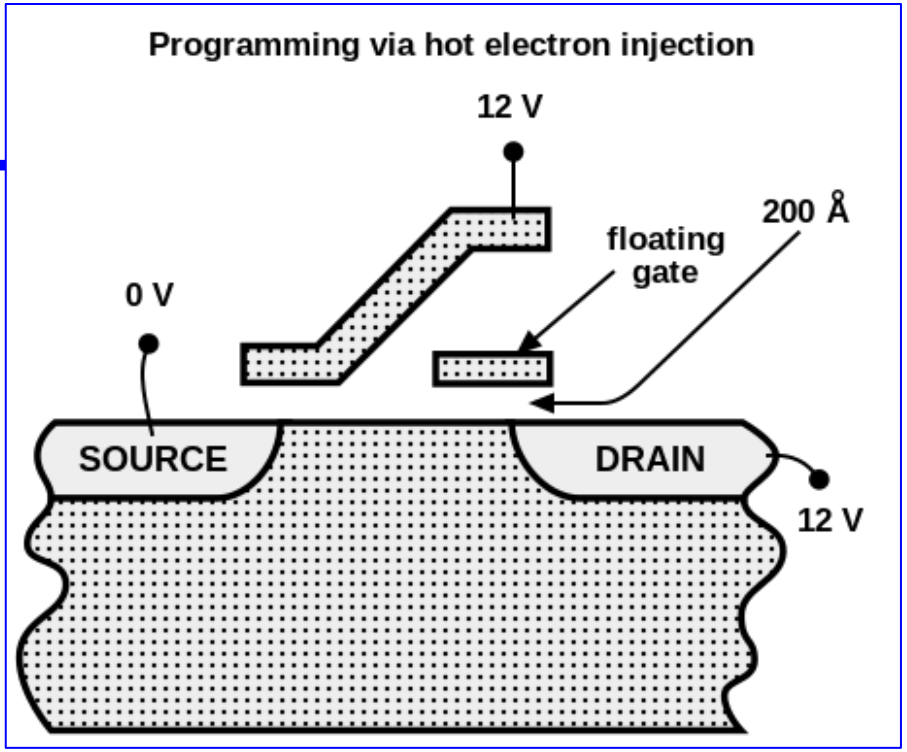


Flash Memory

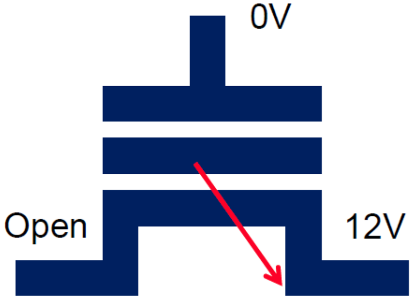
Electrically modifiable, non-volatile storage

Principle of operation:

- Transistor with a second “floating” gate
- Floating gate can trap electrons
- This results in a detectable change in threshold voltage

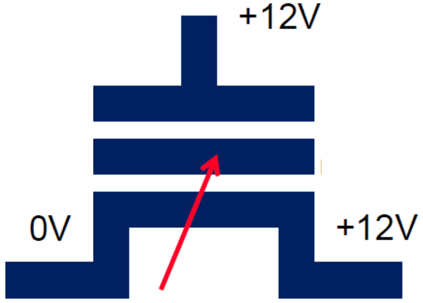


Erasing to logical “1”



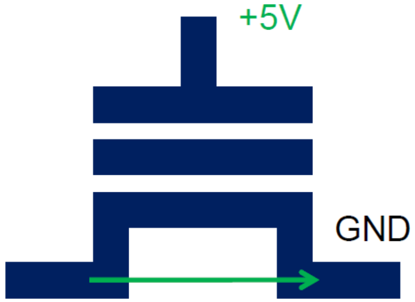
“Quantum tunneling”
Drains charge from FG

Programming (=writing) to logical “0”



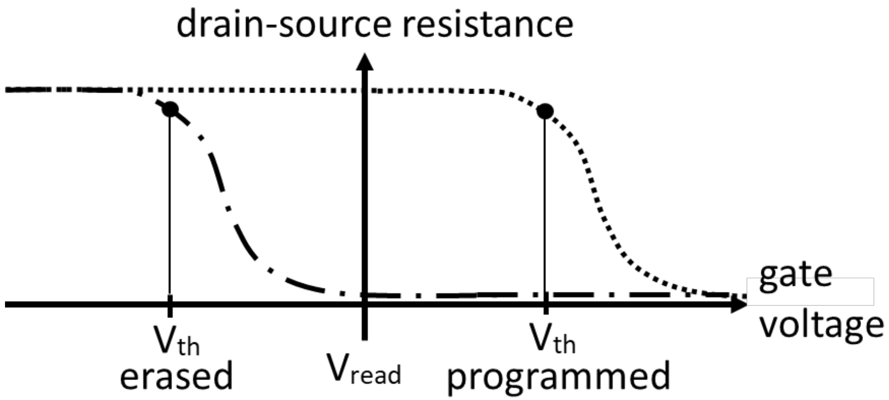
“Hot-electron injection”
traps charge in FG

Reading

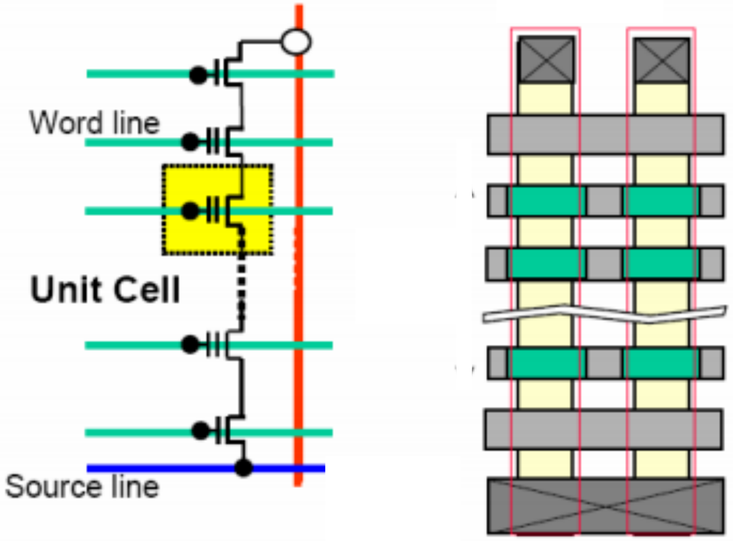
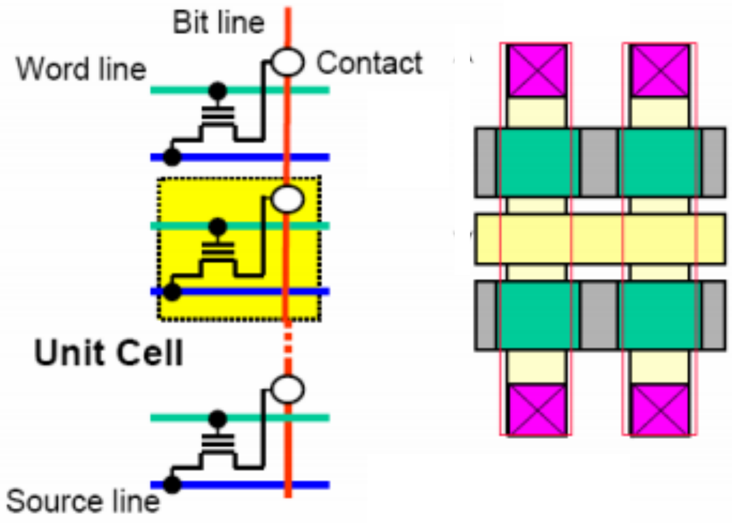
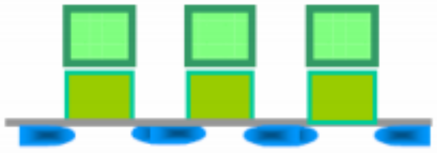
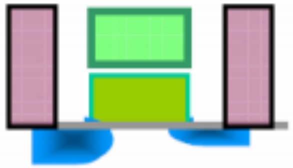


Turn on low Vt or High Vt?

Detect I_{on} to read 0 or 1



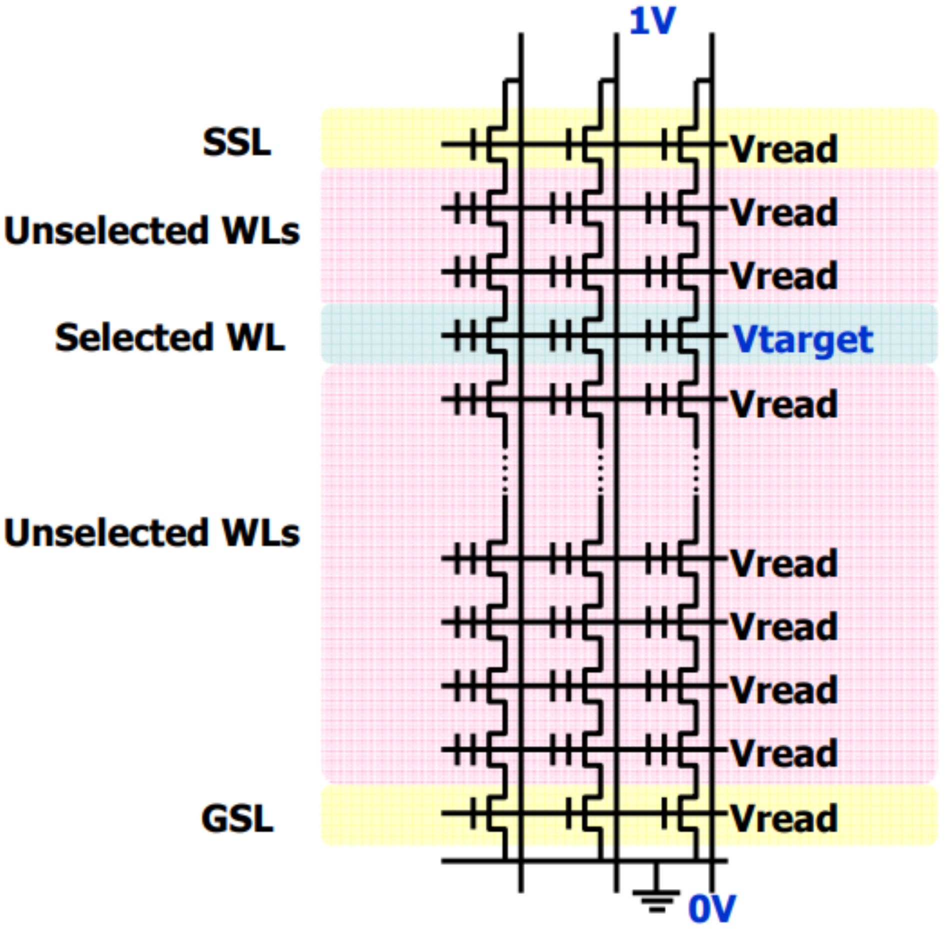
NAND and NOR Flash Memory

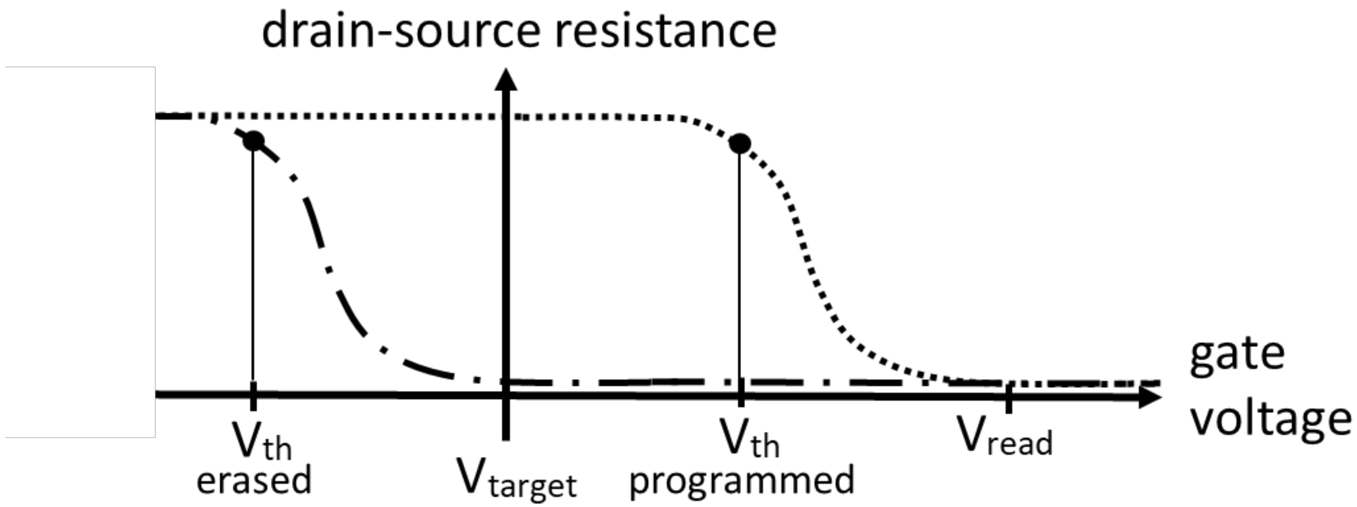
	NAND	NOR
Cell Array & Size		 Bit line Word line Contact Unit Cell Source line
Cross-section		
Features	Small Cell Size, High Density Low Power ➔ *Mass Storage*	Fast random access ➔ *Code Storage*

Example: Reading out NAND Flash

Selected word-line (WL) : Target voltage (V_{target})

Unselected word-lines : V_{read} is high enough to have a low resistance in all transistors in this row





Storage

Memory Map

Example: Memory Map in MSP432

Available memory:

- The MSP432P401R processor has built in 256kB flash memory, 64kB SRAM and 32kB ROM (Read Only Memory).

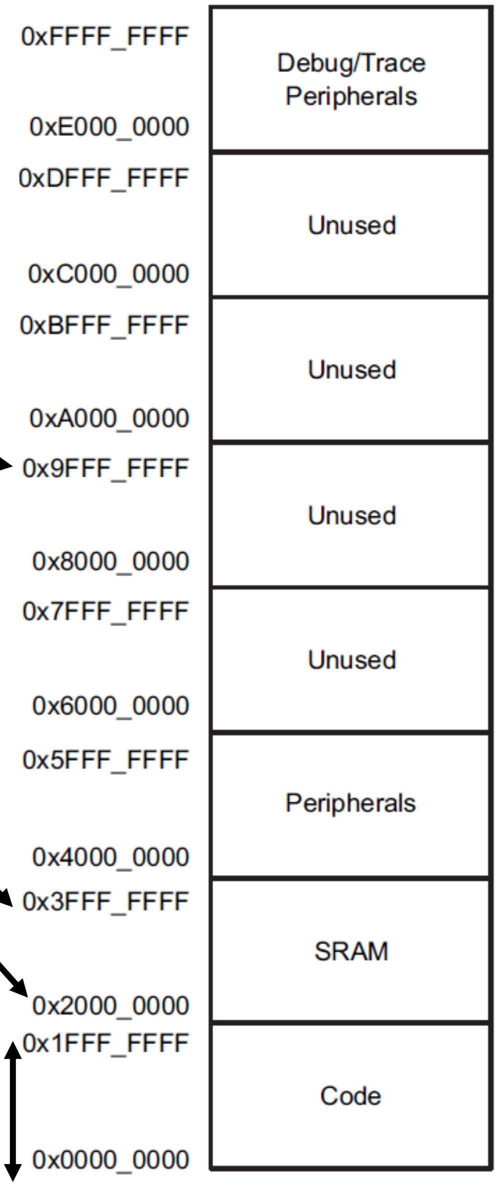
Address space:

- The processor uses 32 bit addresses. Therefore, the addressable memory space is 4 GByte (= 2^{32} Byte) as each memory location corresponds to 1 Byte.
- The address space is used to address the memories (reading and writing), to address the peripheral units, and to have access to debug and trace information (memory mapped microarchitecture).
- The address space is partitioned into zones, each one with a dedicated use. The following is a simplified description to introduce the basic concepts.

Example: Memory Map in MSP432

Memory map:

hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit



0xFFFF_FFFF
Debug/Trace Peripherals
0xE000_0000

0xDFFF_FFFF
Unused
0xC000_0000

0xBFFF_FFFF
Unused
0xA000_0000

0x9FFF_FFFF
Unused
0x8000_0000

0x7FFF_FFFF
Unused
0x6000_0000

0x5FFF_FFFF
Peripherals
0x4000_0000

0x3FFF_FFFF
SRAM
0x2000_0000

0x1FFF_FFFF
Code
0x0000_0000

0011 1111 1111
0010 0000 0000

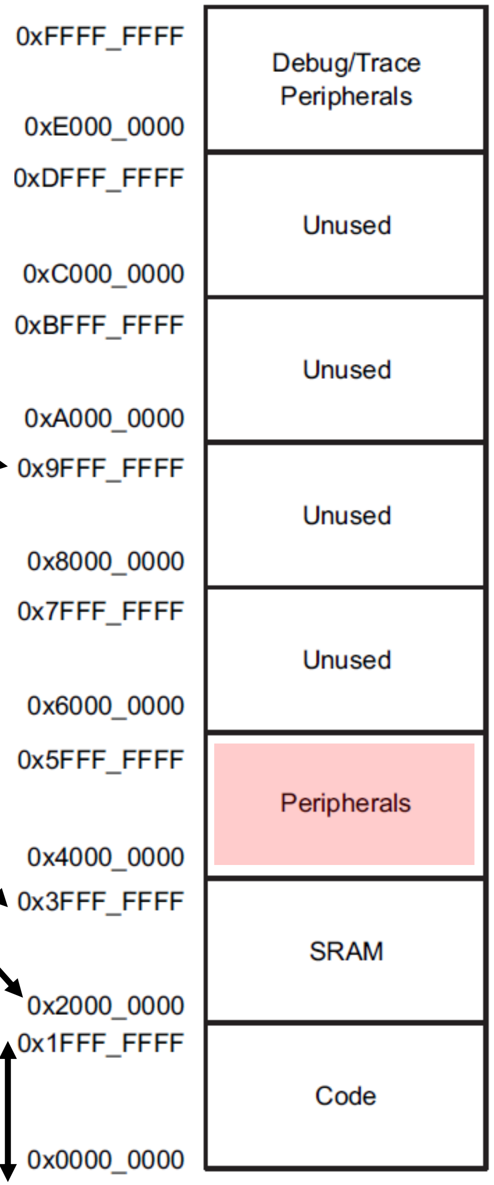
diff. = 0001 1111 .... 1111 →
2^{29} different addresses

capacity = 2^{29} Byte =
512 MByte

Example: Memory Map in MSP432

Memory map:

hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit



0011 1111 …. 1111

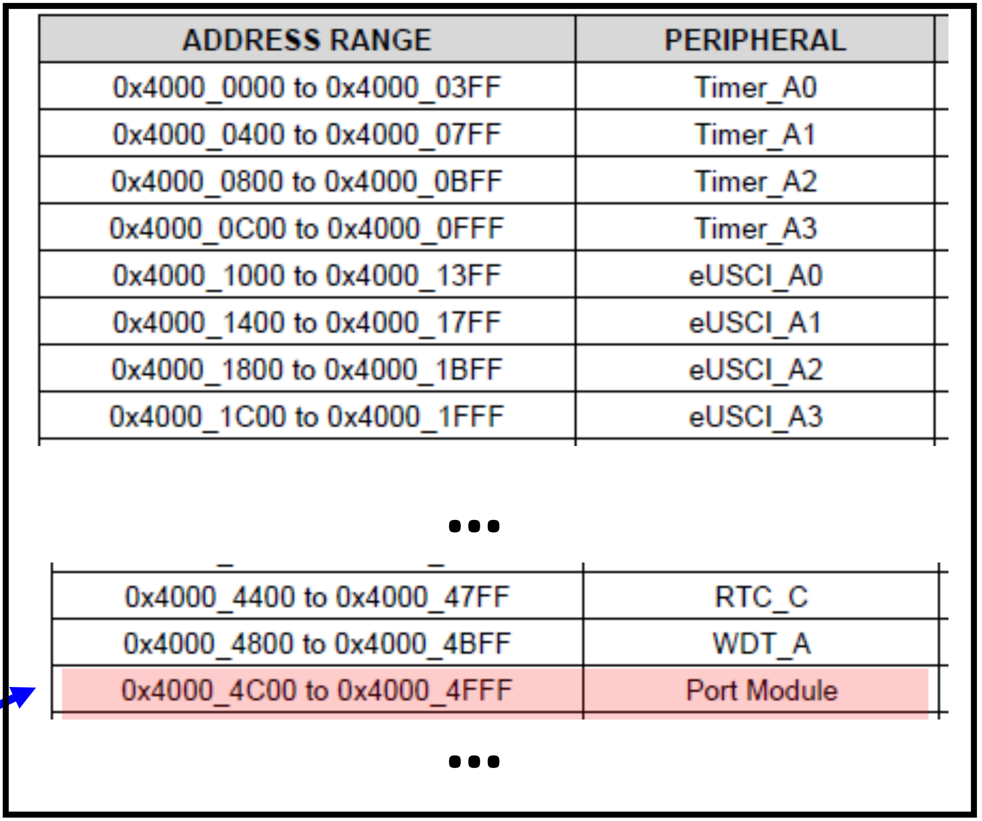
0010 0000 …. 0000

diff. = 0001 1111 …. 1111 →

2^{29} different addresses

capacity = 2^{29} Byte =

512 MByte



ADDRESS RANGE	PERIPHERAL
0x4000_0000 to 0x4000_03FF	Timer_A0
0x4000_0400 to 0x4000_07FF	Timer_A1
0x4000_0800 to 0x4000_0BFF	Timer_A2
0x4000_0C00 to 0x4000_0FFF	Timer_A3
0x4000_1000 to 0x4000_13FF	eUSCI_A0
0x4000_1400 to 0x4000_17FF	eUSCI_A1
0x4000_1800 to 0x4000_1BFF	eUSCI_A2
0x4000_1C00 to 0x4000_1FFF	eUSCI_A3

…

0x4000_4400 to 0x4000_47FF	RTC_C
0x4000_4800 to 0x4000_4BFF	WDT_A
0x4000_4C00 to 0x4000_4FFF	Port Module

…

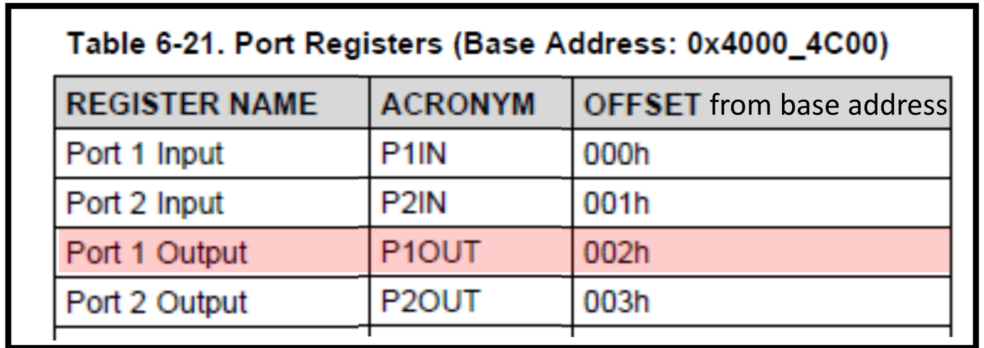


Table 6-21. Port Registers (Base Address: 0x4000_4C00)

REGISTER NAME	ACRONYM	OFFSET from base address
Port 1 Input	P1IN	000h
Port 2 Input	P2IN	001h
Port 1 Output	P1OUT	002h
Port 2 Output	P2OUT	003h

Example: Memory Map in MSP432

Memory map:

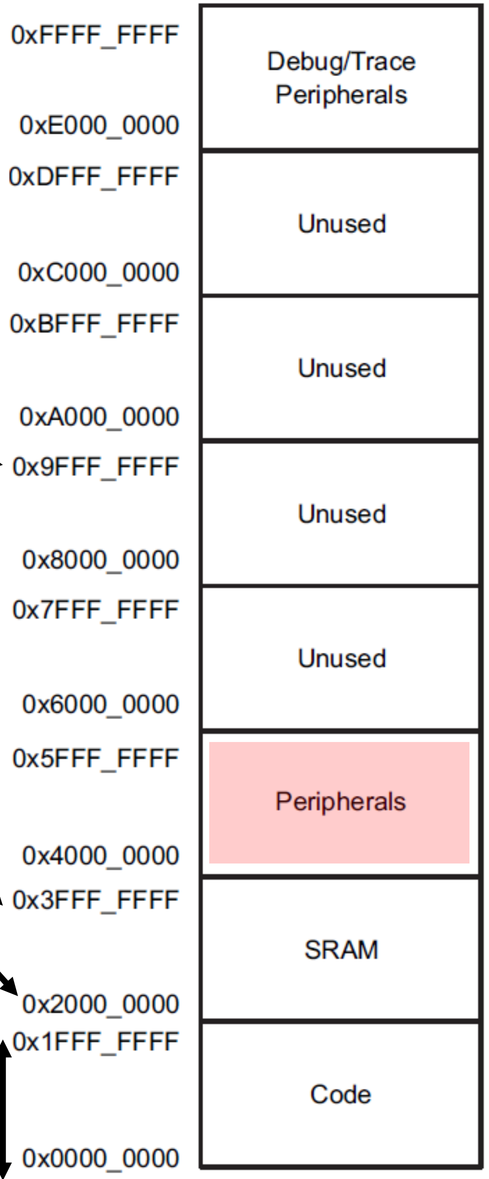
hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit

0011 1111 1111

0010 0000 0000

diff. = 0001 1111 .... 1111 → 2^{29} different addresses

capacity = 2^{29} Byte = 512 MByte



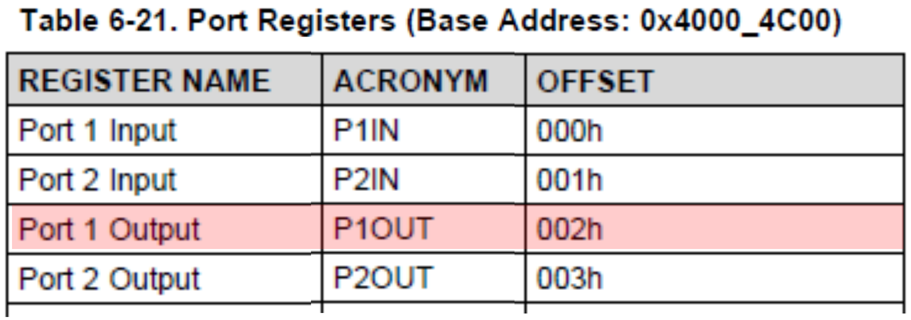
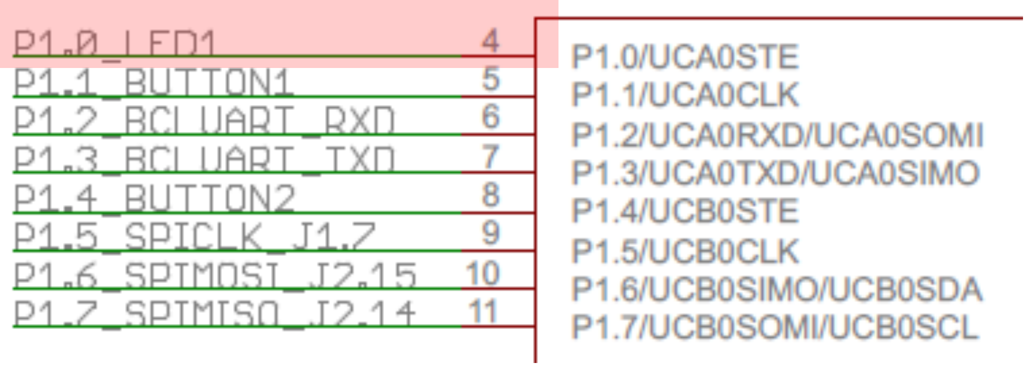


Table 6-21. Port Registers (Base Address: 0x4000_4C00)

REGISTER NAME	ACRONYM	OFFSET
Port 1 Input	P1IN	000h
Port 2 Input	P2IN	001h
Port 1 Output	P1OUT	002h
Port 2 Output	P2OUT	003h

Schematic of LaunchPad:



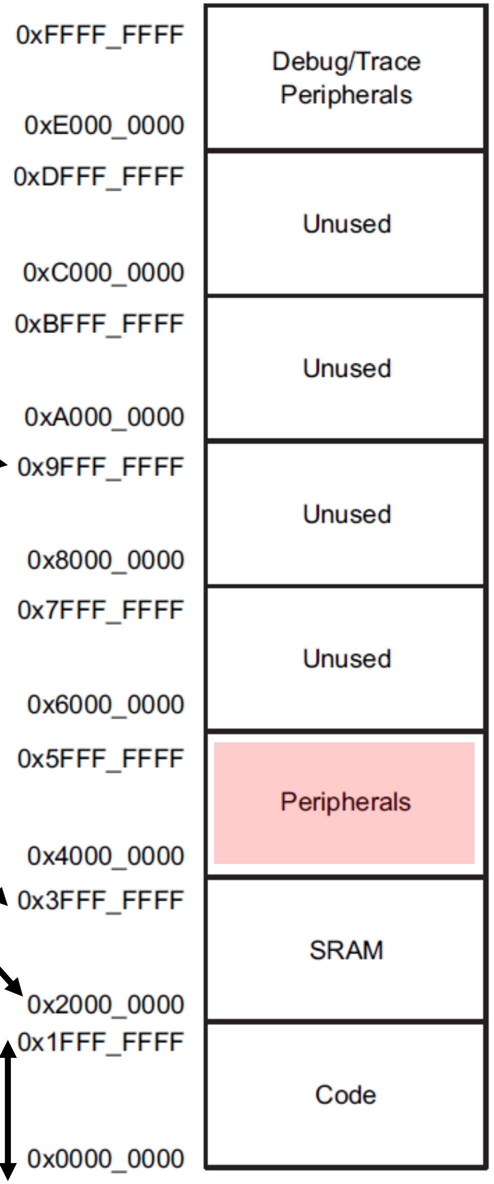
LED1 is connected to Port 1, Pin 0

How do we toggle LED1 in a C program?

Example: Memory Map in MSP432

Memory map:

hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit



0011 1111 …. 1111

0010 0000 …. 0000

diff. = 0001 1111 …. 1111 → 2^{29} different addresses

capacity = 2^{29} Byte = 512 MByte

Many necessary elements are missing in the sketch below, in particular the configuration of the port (input or output, pull up or pull down resistors for input, drive strength for output).

```
…
//declare p1out as a pointer to an 8Bit integer
volatile uint8_t*  p1out;

//P1OUT should point to Port 1 where LED1 is connected
p1out = (uint8_t*) 0x40004C02;

//Toggle Bit 0 (Signal to which LED1 is connected)
*p1out = *p1out ^ 0x01;
```

^ : XOR

Example: Memory Map in MSP432

Memory map:

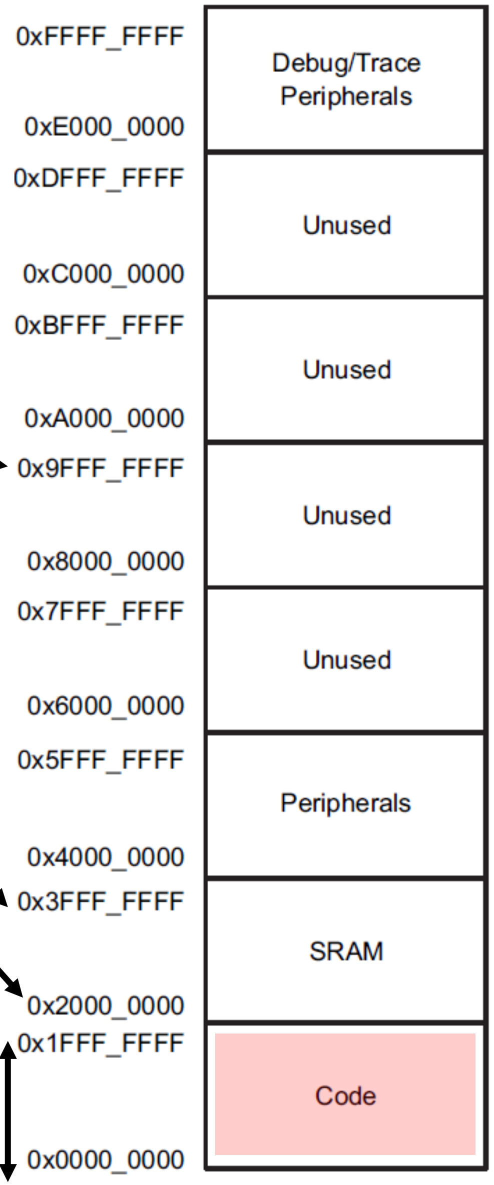
hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit

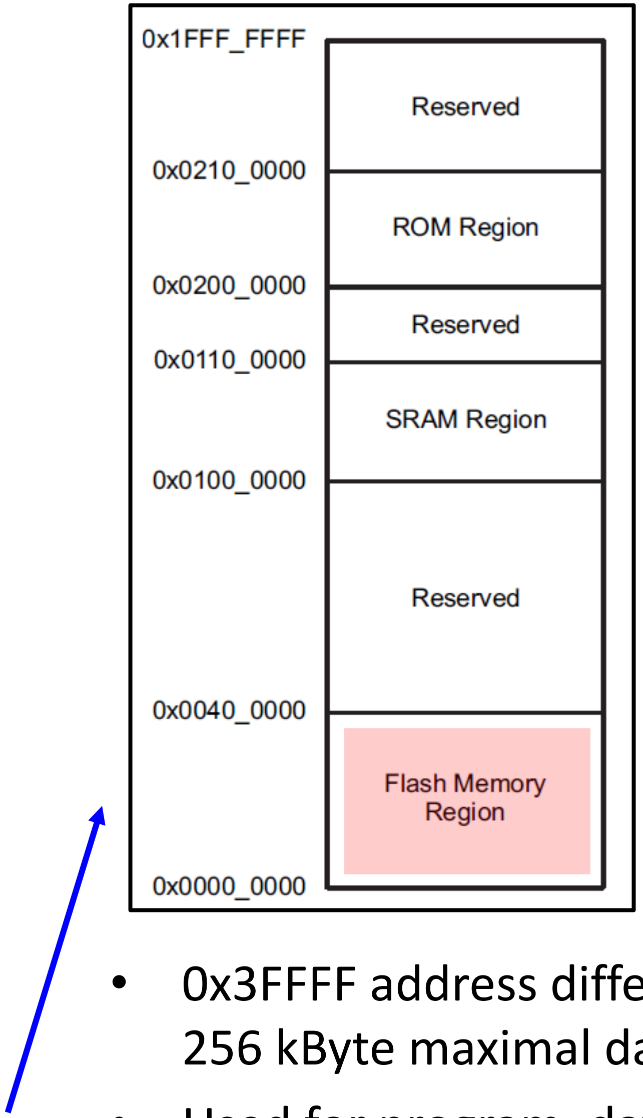
0011 1111 1111

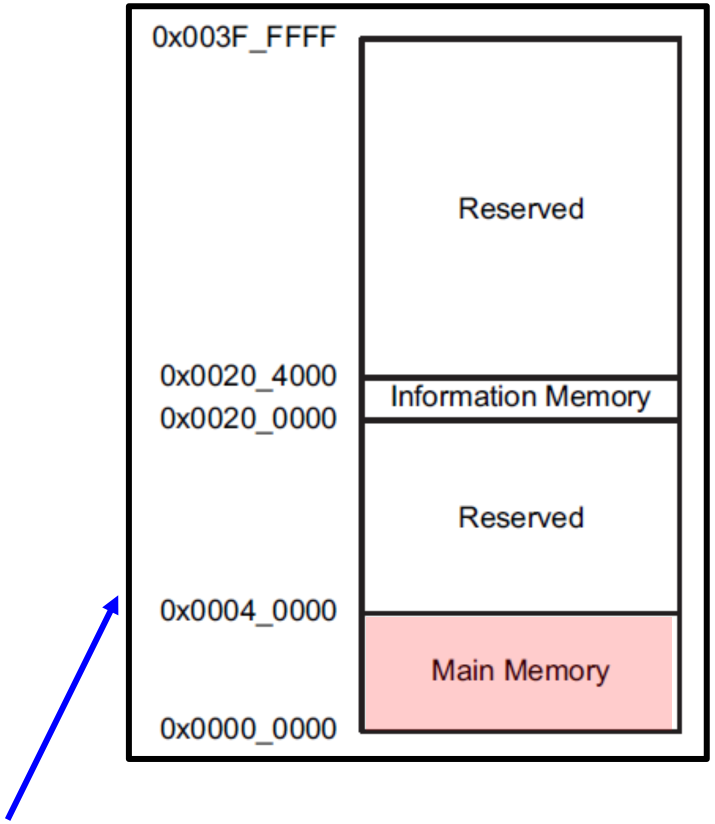
0010 0000 0000

diff. = 0001 1111 .... 1111 → 2^{29} different addresses

capacity = 2^{29} Byte = 512 MByte







- 0x3FFFF address difference = 4 * 2^{16} different addresses → 256 kByte maximal data capacity for Flash Main Memory
- Used for program, data and non-volatile configuration.

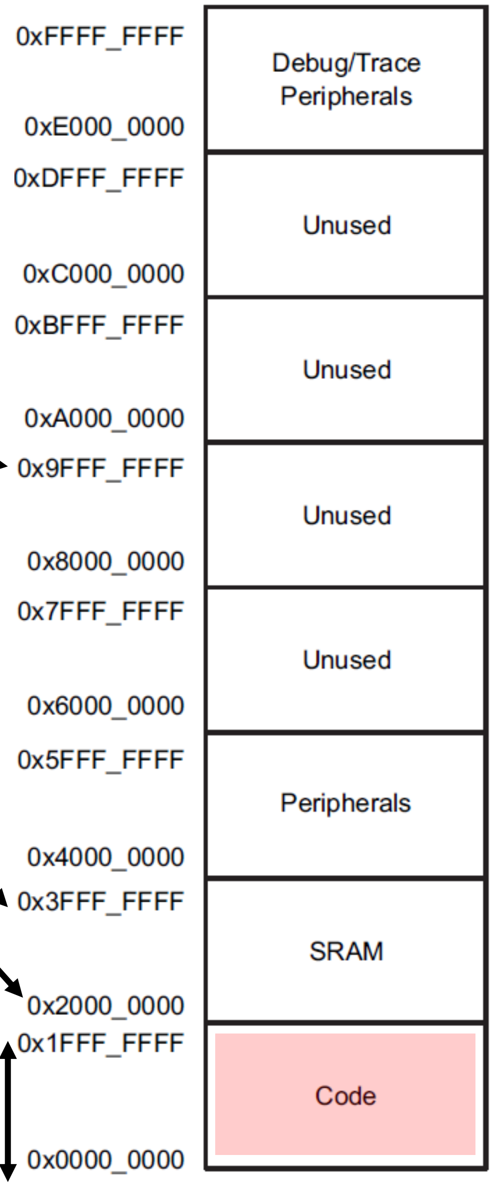
Example: Memory Map in MSP432

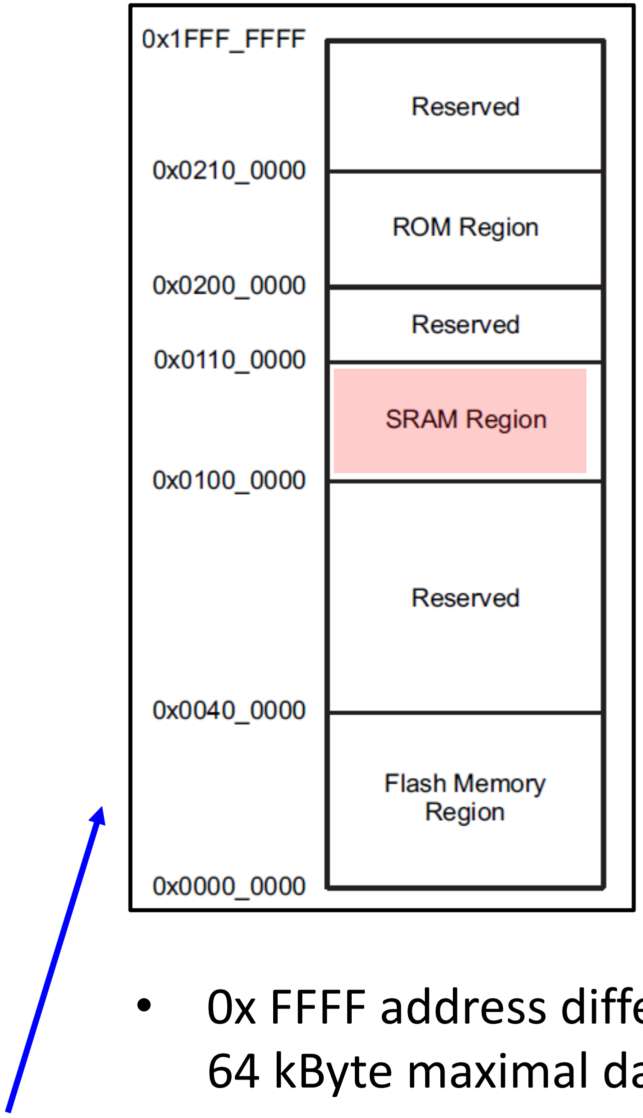
Memory map:

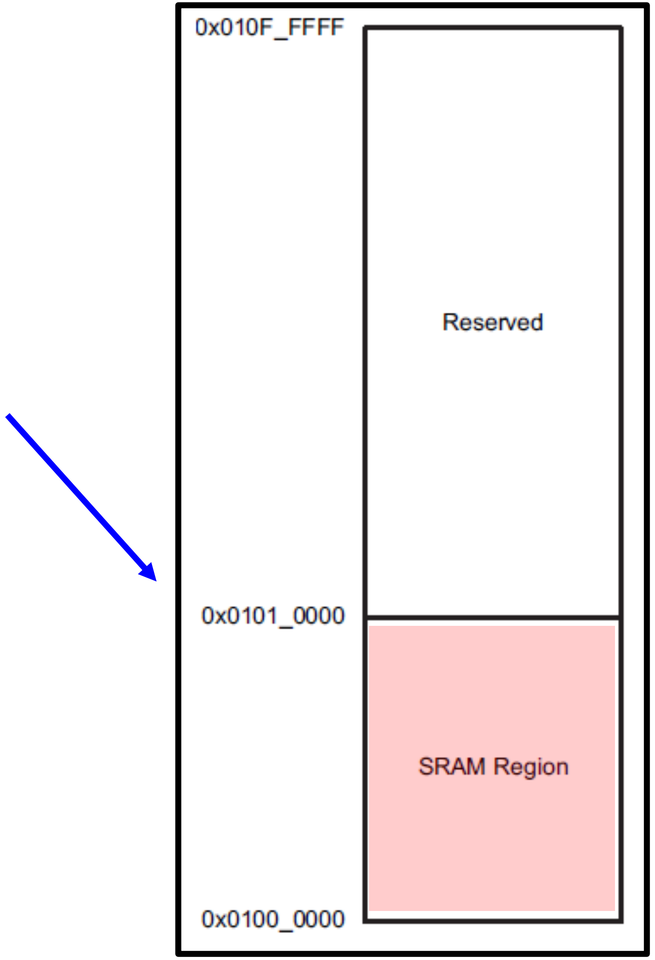
hexadecimal representation of a 32 bit binary number; each digit corresponds to 4 bit

0011 1111 1111
0010 0000 0000

diff. = 0001 1111 .... 1111 →
2^{29} different addresses
capacity = 2^{29} Byte =
512 MByte







- 0x FFFF address difference = 2^{16} different addresses → 64 kByte maximal data capacity for SRAM Region
- Used for program and data.

Input and Output

Device Communication

Very often, a processor needs to *exchange information with other processors* or devices. To satisfy various needs, there exists many different *communication protocols*, such as

- ***UART*** (Universal Asynchronous Receiver-Transmitter)
- ***SPI*** (Serial Peripheral Interface Bus)
- ***I2C*** (Inter-Integrated Circuit)
- ***USB*** (Universal Serial Bus)

- As the principles are similar, we will just explain a representative of an asynchronous protocol (*UART,* no shared clock signal between sender and receiver) and one of a synchronous protocol (*SPI* , shared clock signal).

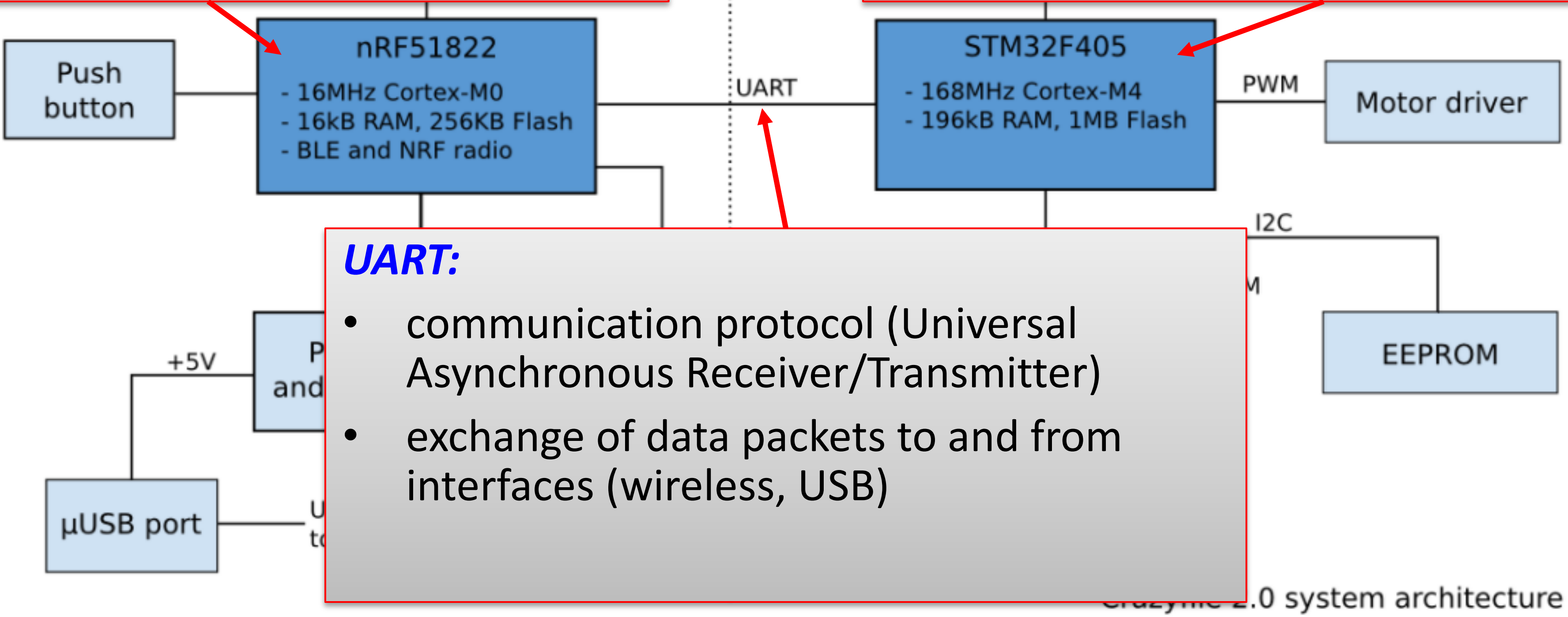
Remember?

low power CPU

- enabling power to the rest of the system
- battery charging and voltage measurement
- wireless radio (boot and operate)
- detect and check expansion boards

higher performance CPU

- sensor reading and motor control
- flight control
- telemetry (including the battery voltage)
- additional user development
- USB connection



UART:

- communication protocol (Universal Asynchronous Receiver/Transmitter)
- exchange of data packets to and from interfaces (wireless, USB)

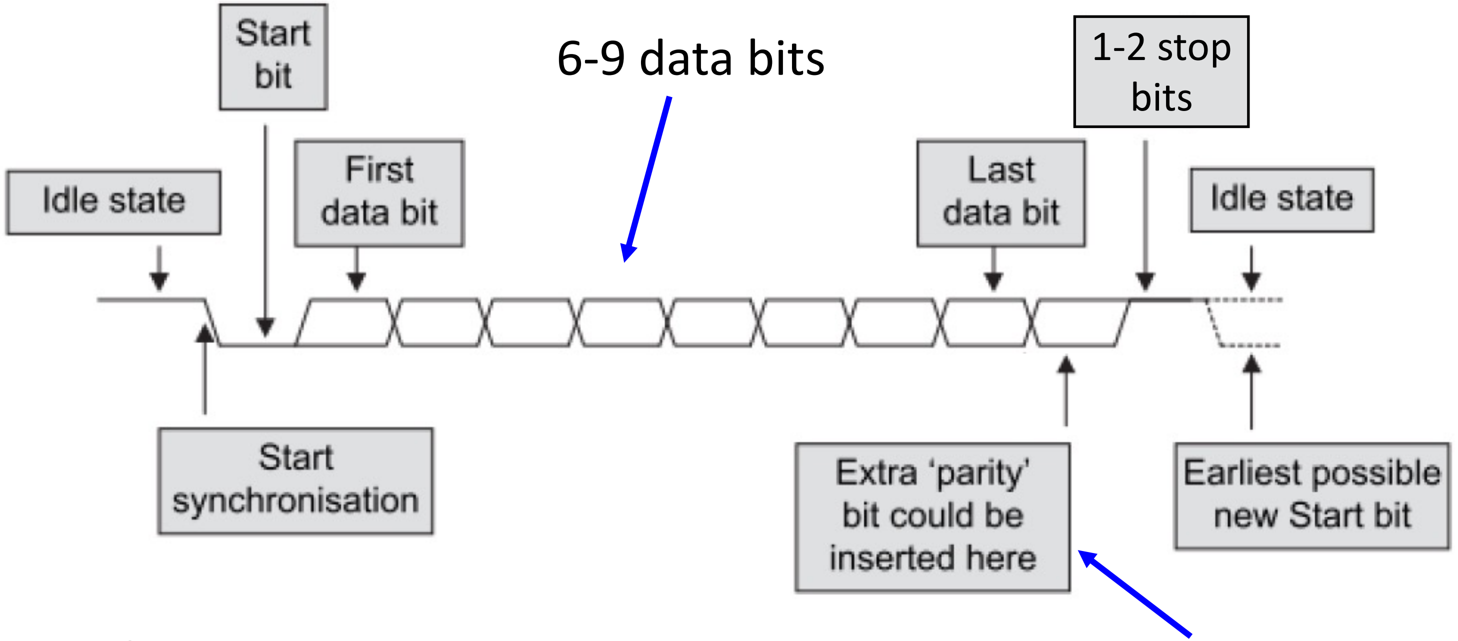
Crazyflie 2.0 system architecture

Input and Output

UART Protocol

UART

- *Serial communication* of bits via a single signal, i.e. UART provides parallel-to-serial and serial-to-parallel conversion.
- Sender and receiver need to *agree on the transmission rate*.
- Transmission of a serial packet starts with a start bit, followed by data bits and finalized using a stop bit:

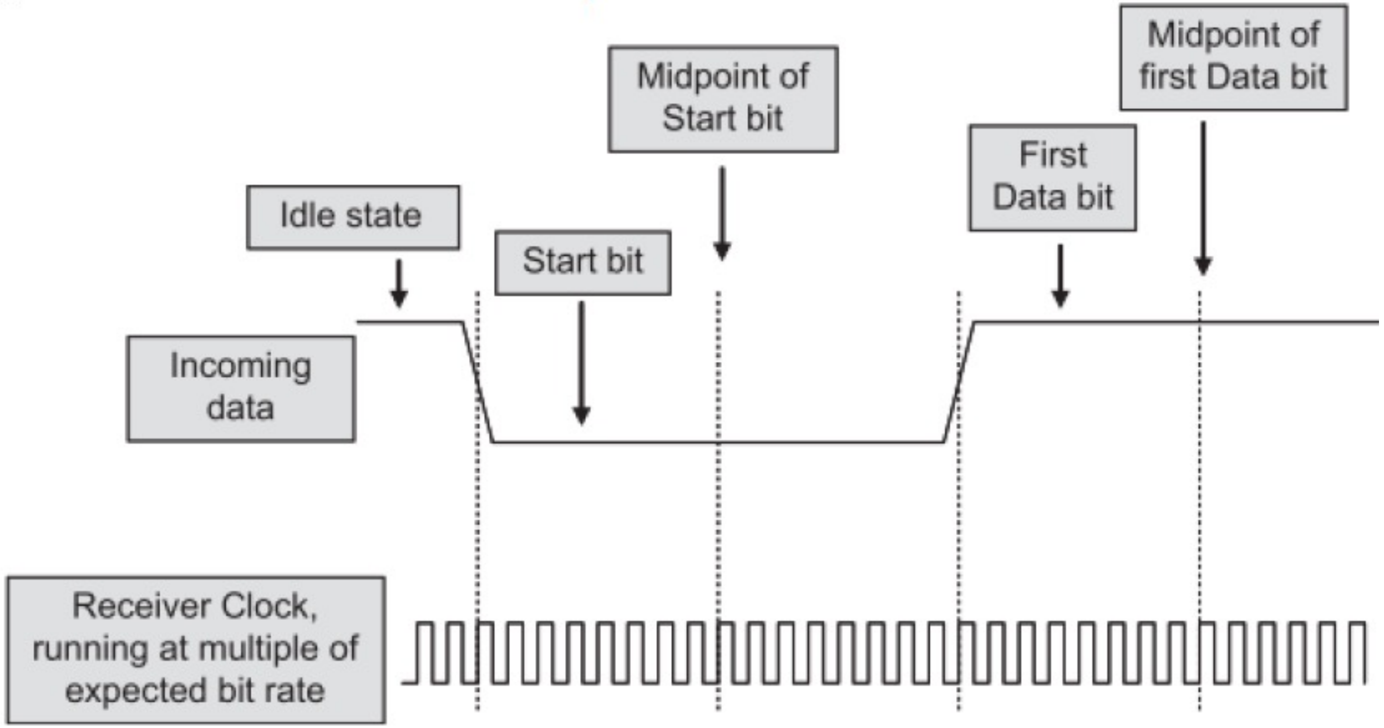


for detecting single bit errors

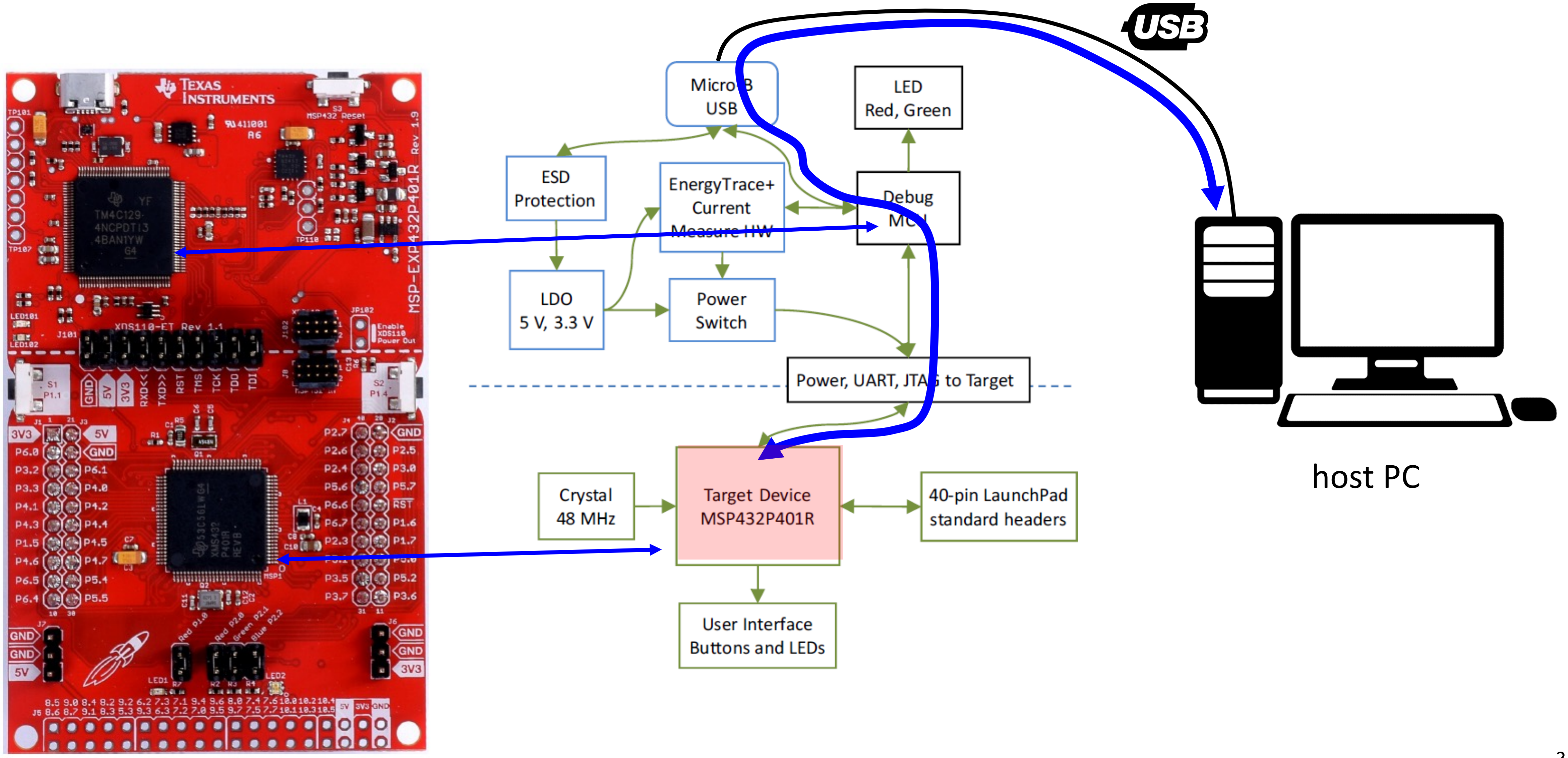
- There exist many variations of this simple scheme.

UART

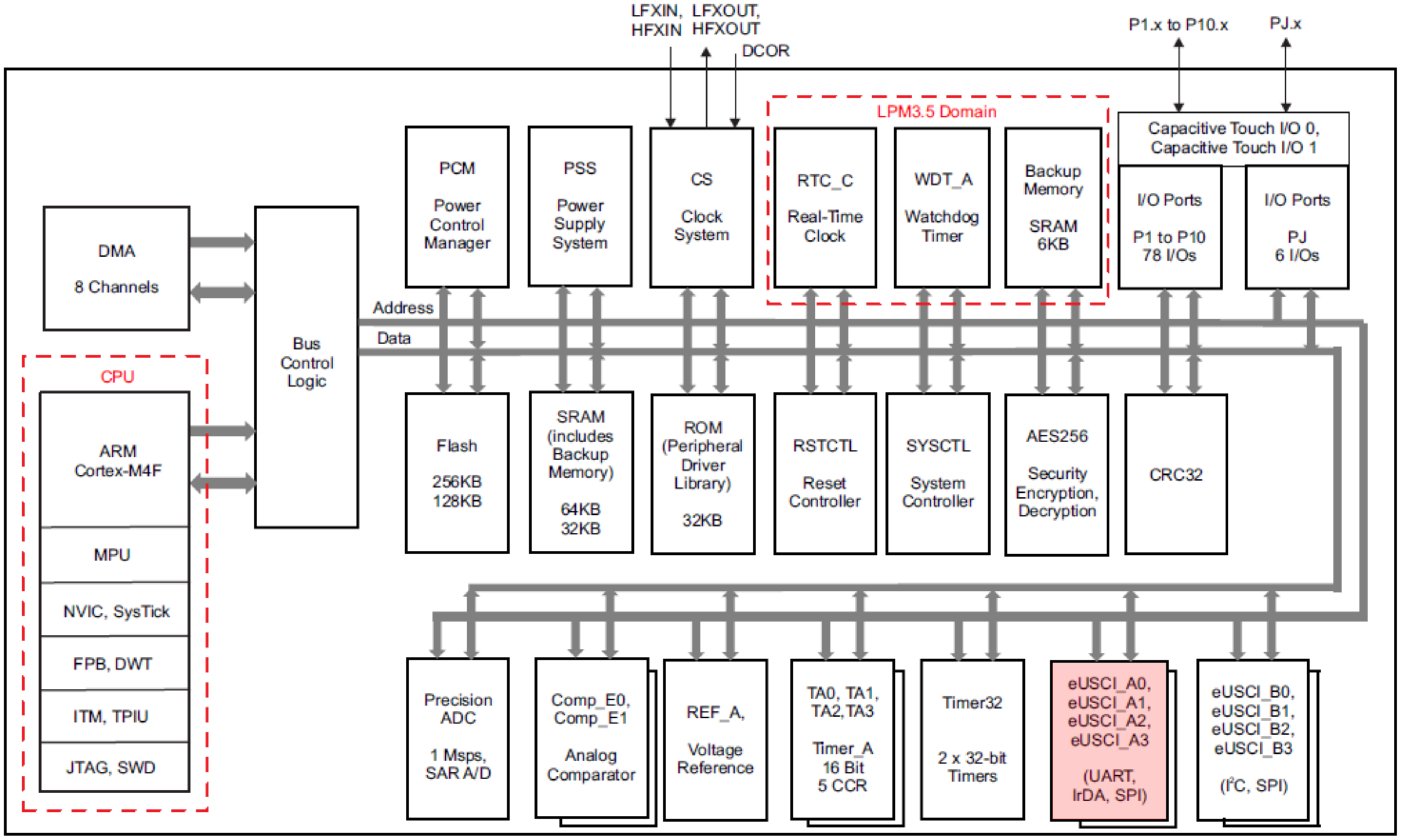
- The receiver runs an *internal clock* whose frequency is an exact multiple of the expected bit rate.
- When a *Start bit* is detected, a counter begins to count clock cycles e.g. 8 cycles until the midpoint of the anticipated Start bit is reached.
- The clock counter counts a further 16 cycles, to the middle of the first *Data bit*, and so on until the *Stop bit*.


Midpoint of first Data bit
Midpoint of Start bit
First Data bit
Idle state
Start bit
Incoming data
Receiver Clock, running at multiple of expected bit rate

UART with MSP432


USB
Micro-B USB
LED Red, Green
ESD Protection
EnergyTrace+ Current Measure HW
Debug MCU
LDO 5 V, 3.3 V
Power Switch
Power, UART, JTAG to Target
Crystal 48 MHz
Target Device MSP432P401R
40-pin LaunchPad standard headers
User Interface Buttons and LEDs
host PC

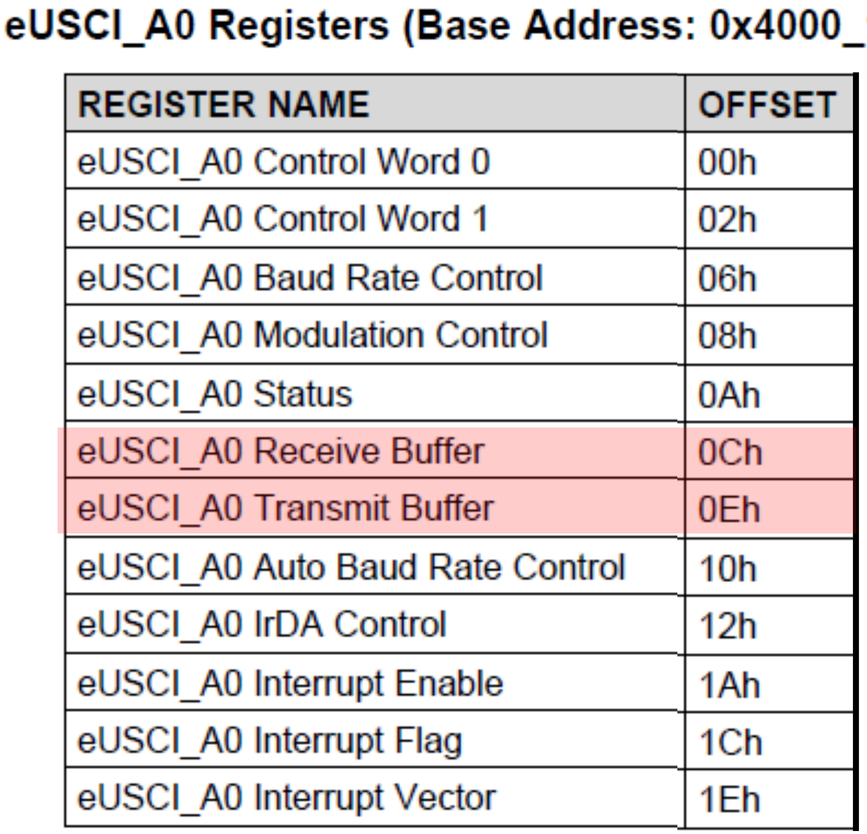
UART with MSP432



Input and Output

Memory Mapped Device Access

Memory-Mapped Device Access



eUSCI_A0 Registers (Base Address: 0x4000_1000)

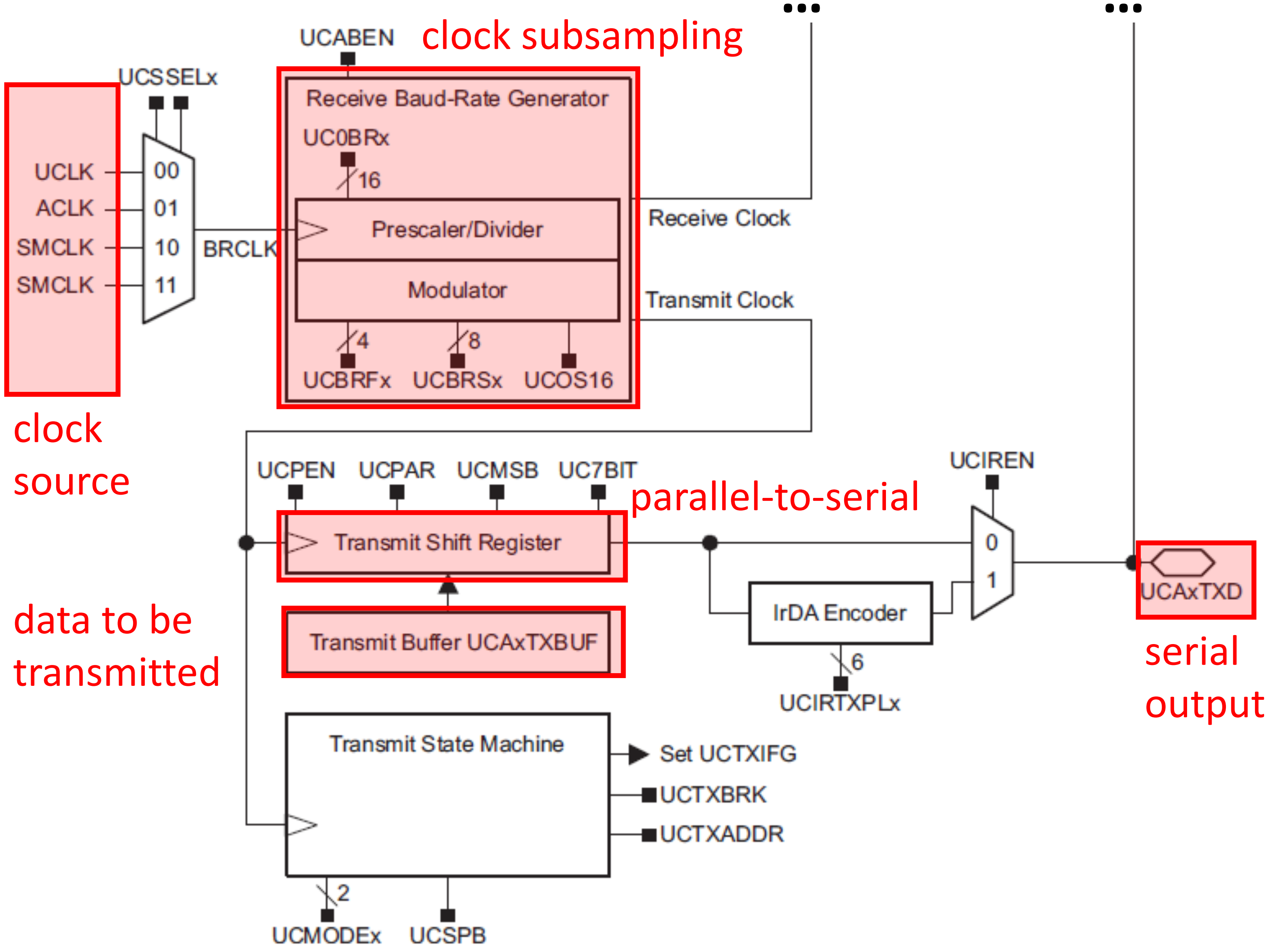
REGISTER NAME	OFFSET
eUSCI_A0 Control Word 0	00h
eUSCI_A0 Control Word 1	02h
eUSCI_A0 Baud Rate Control	06h
eUSCI_A0 Modulation Control	08h
eUSCI_A0 Status	0Ah
eUSCI_A0 Receive Buffer	0Ch
eUSCI_A0 Transmit Buffer	0Eh
eUSCI_A0 Auto Baud Rate Control	10h
eUSCI_A0 IrDA Control	12h
eUSCI_A0 Interrupt Enable	1Ah
eUSCI_A0 Interrupt Flag	1Ch
eUSCI_A0 Interrupt Vector	1Eh

- *Configuration of Transmitter and Receiver must match*; otherwise, they can not communicate.
- Examples of configuration parameters:
  - transmission rate (baud rate, i.e., symbols/s)
  - LSB or MSB first
  - number of bits per packet
  - parity bit
  - number of stop bits
  - interrupt-based communication
  - clock source

in our case: bit/s

buffer for received bits and bits that should be transmitted

Transmission Rate



Clock subsampling:

- The clock subsampling block is complex, as one tries to match a large set of transmission rates with a fixed input frequency.

Clock Source:

- Let us assume SMCLK = 3MHz
- Quartz frequency = 48 MHz, is divided by 16 before connected to SMCLK

Example:

- Transmission rate 4800 bit/s
- 16 clock periods per bit (see 3-34)
- Subsampling factor = 3*10^6 / (4.8*10^3 * 16) = 39.0625

Software Interface

Part of C program that *prints a character to a UART* terminal on the host PC:

```
...
static const eUSCI_UART_Config uartConfig =
{
  EUSCI_A_UART_CLOCKSOURCE_SMCLK,                    // SMCLK Clock Source
  39,                                                // BRDIV  = 39 , integral part
  1,                                                 // UCxBRF =  1 , fractional part * 16
  0,                                                 // UCxBRS =  0
  EUSCI_A_UART_NO_PARITY,                            // No Parity
  EUSCI_A_UART_LSB_FIRST,                            // LSB First
  EUSCI_A_UART_ONE_STOP_BIT,                         // One stop bit
  EUSCI_A_UART_MODE,                                 // UART mode
  EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION}; // Oversampling Mode
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
          GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION );      //Configure CPU signals
UART_initModule(EUSCI_A0_BASE, &uartConfig);                 // Configuring UART Module A0
UART_enableModule(EUSCI_A0_BASE);                            // Enable UART module A0

UART_transmitData(EUSCI_A0_BASE,'a');                        // Write character ‘a’ to UART

...
```

data structure uartConfig contains the configuration of the UART

use uartConfig to write to eUSCI_A0 configuration registers

start UART

base address of A0 (0x40001000), where A0 is the instance of the UART peripheral

Software Interface

Replacing UART_transmitData(EUSCI_A0_BASE,'a') by a *direct access to registers*:

```
...
volatile uint16_t*  uca0ifg = (uint16_t*) 0x4000101C;
volatile uint16_t*  uca0txbuf = (uint16_t*) 0x4000100E;
...
// Initialization of UART as before
...
while (!((*uca0ifg >> 1) & 0x0001));
*uca0txbuf = (char) 'g'; // Write to transmit buffer
...
```

declare pointers to UART configuration registers

wait until transmit buffer is empty

write character ‘g’ to the transmit buffer

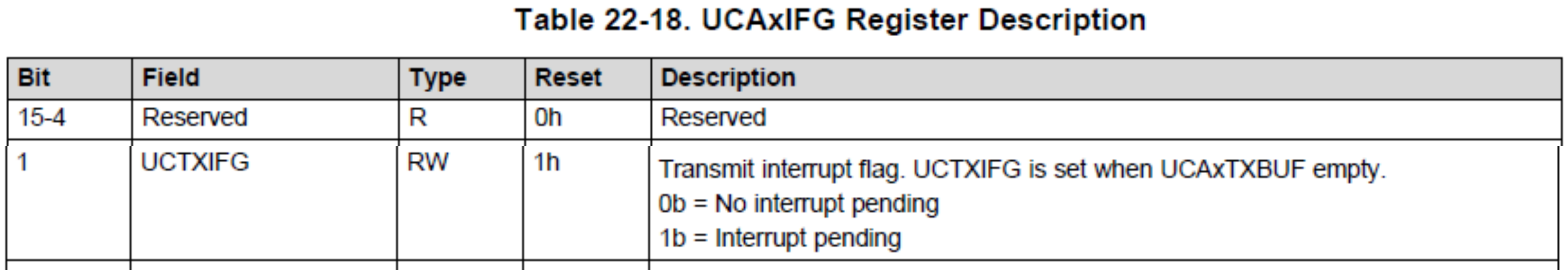


Table 22-18. UCAxIFG Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending

shift 1 bit to the right

```
!((*uca0ifg >> 1) & 0x0001)
```

expression is ‘1’ if bit UCTXIFG = 0 (buffer not empty).