Computer Architecture & Real-Time Operating System

# 10. I/O Devices


**Prof. Jong-Chan Kim**

**Dept. Automobile and IT Convergence**

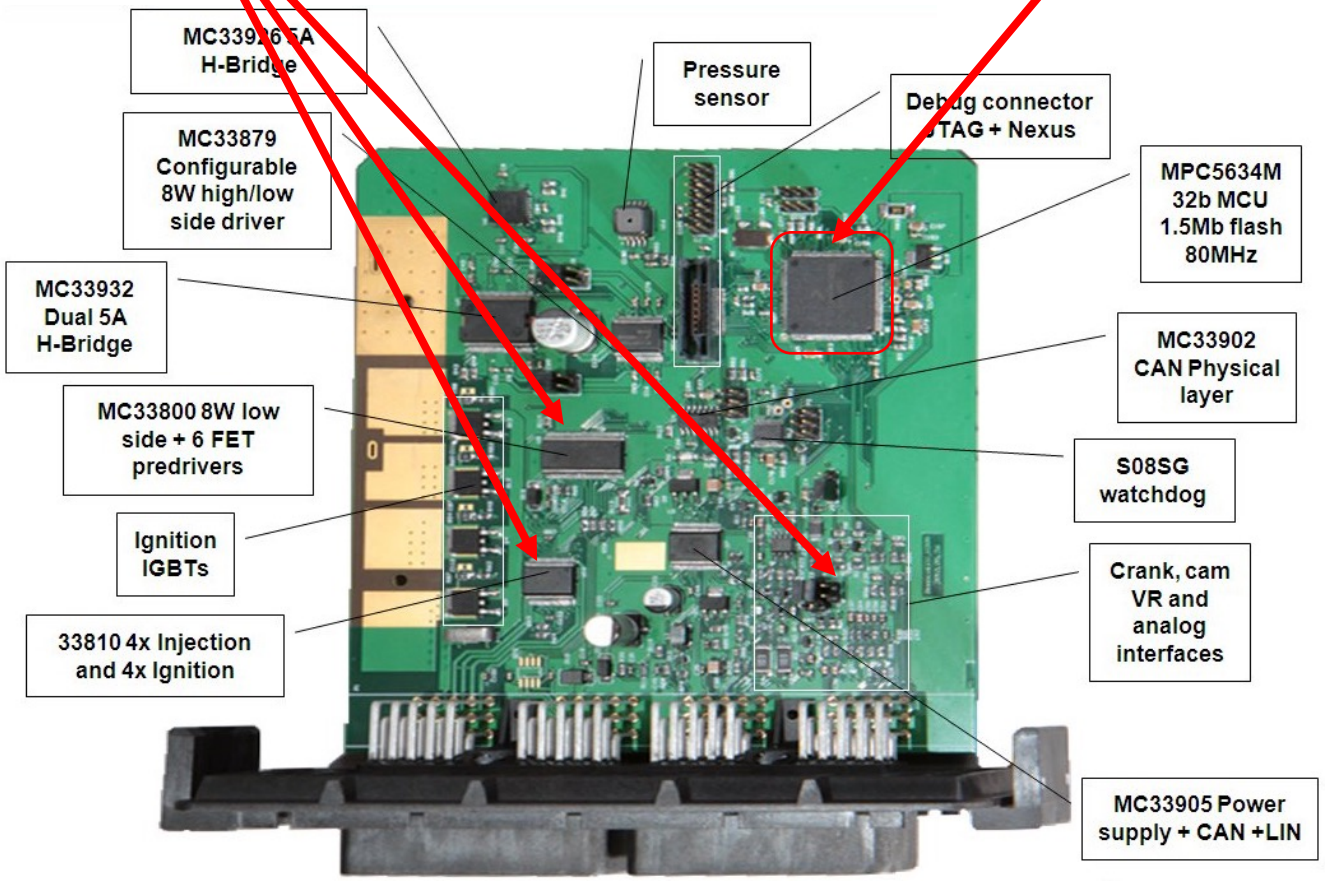
국민대학교
KOOKMIN UNIVERSITY

# Peripheral Devices for PC


# On-Chip and On-Board Peripherals for ECU

Application-Specific
On-Board Peripherals

MCU (CPU + Memory + On-Chip Peripherals)




- GPIO
- ADC
- DAC
- Timer
- UART
- SPI
- I2C
- …

ECU Connector to Sensors and Actuators

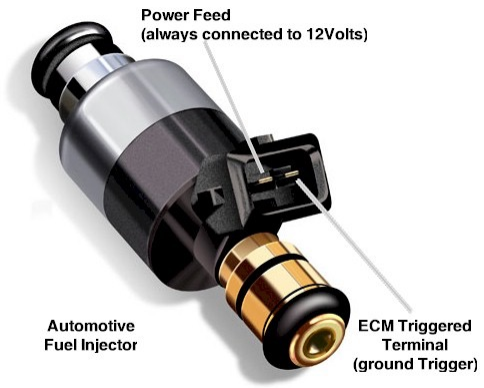
# Engine Sensors and Actuators

Accelerator Pedal Position Sensor


Throttle Position Sensor (TPS)

Throttle body + Motor + TPS

Crankshaft Position Sensor

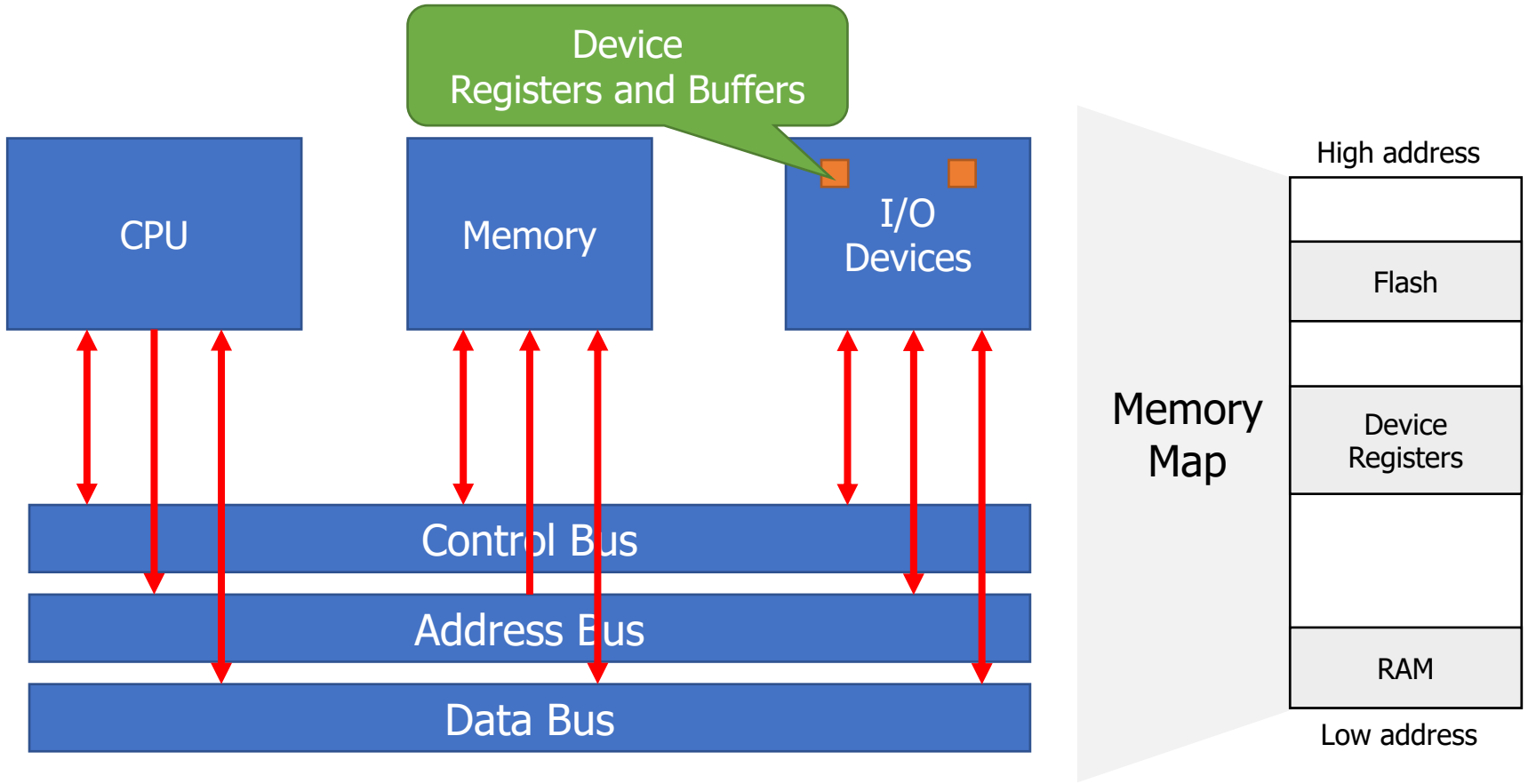



Fuel Injector

Spark Plug

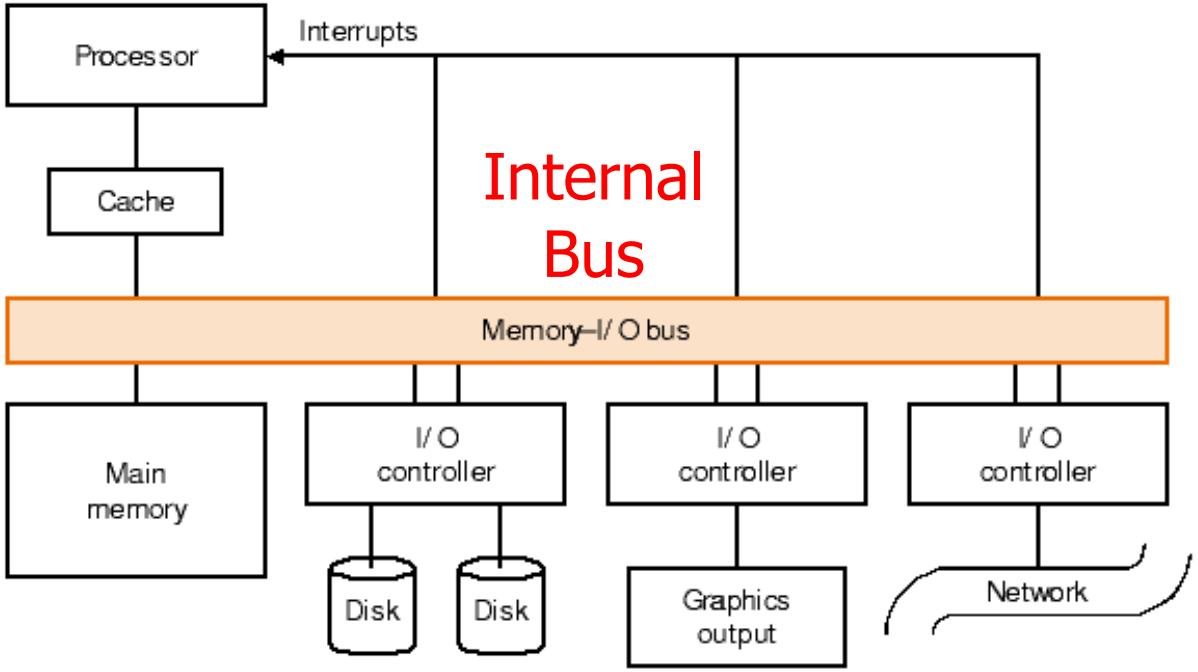
# System Bus and I/O Devices

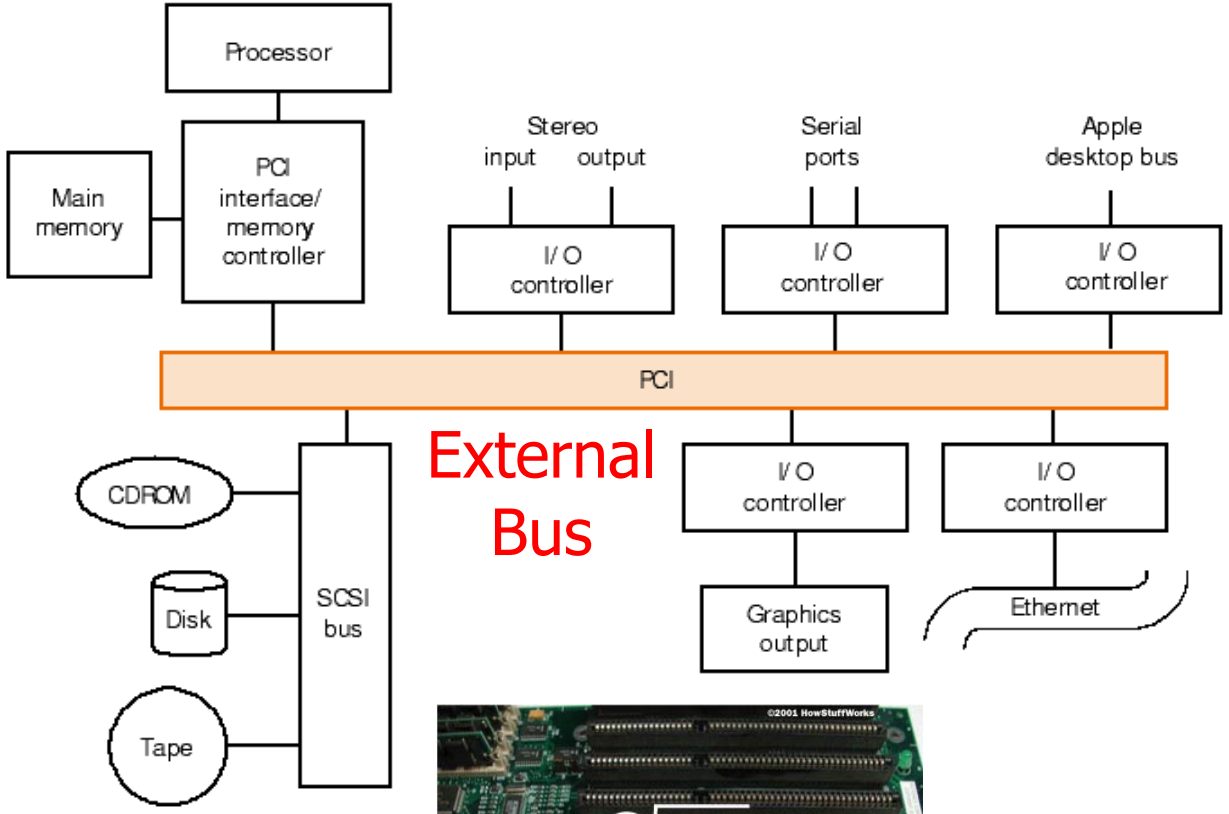
- I/O devices are just like memory from the perspective of CPU
- Memory-mapped I/O: device registers are mapped in address space

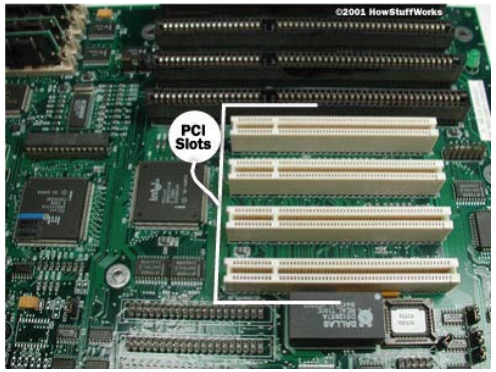



https://en.wikipedia.org/wiki/System_bus

# Internal Bus vs External Bus




PCI Bus

# Memory-Mapped I/O vs. Port-Mapped I/O

- Memory-Mapped I/O
  - Device registers are mapped in the address space
  - Variables and device registers are accessed in the same way (i.e., pointers)

- Port-Mapped I/O (or Isolated I/O)
  - Device registers exist in a separated (isolated) I/O space
  - Dedicated instructions for reading and writing the I/O space
  - Can have extra pins and isolated I/O buses directly connected to CPU
  - Almost deprecated (rarely seen only in old I/O devices with Intel CPUs)

# Port-Mapped I/O Example

```
static inline uint8_t inb(uint16_t port)
{
    uint8_t ret;
    asm volatile ( "inb %1, %0"
                   : "=a"(ret)
                   : "Nd"(port) );
    return ret;
}

static inline void outb(uint16_t port, uint8_t val)
{
    asm volatile ( "outb %0, %1" : : "a"(val), "Nd"(port) );
}
```

Source: http://wiki.osdev.org/Inline_Assembly/Examples

# Memory-mapped I/O Example

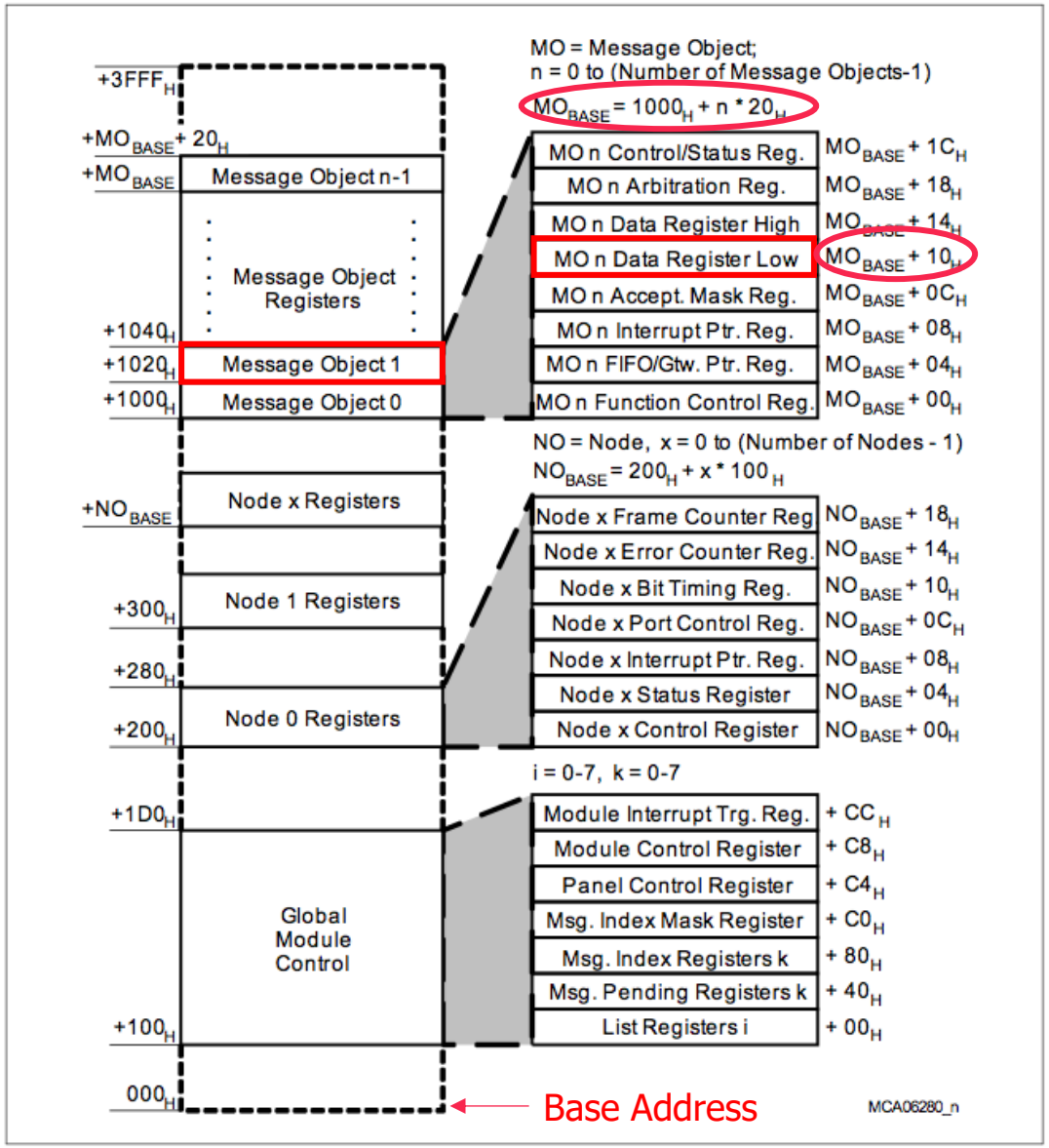



**Figure 19-23 MultiCAN Register Address Map**

TriCore TC1797 User Manual

**Table 19-5 Registers Address Space - MultiCAN Kernel Registers**

| Module | Base Address | End Address | Note |
|---|---|---|---|
| CAN | $F000\ 4000_H$ | $F000\ 7FFF_H$ | - |

```
u32_t data;

/* read */
#define CAN 0xF0004000
data = *(volatile u32_t *)(CAN + 0x1000 + 1 * 0x20 + 0x10);

/* write */
*(volatile u32_t *)(CAN + 0x1000 + 1 * 0x20 + 0x10) = data;
```

$MO_{BASE}$

Structure representing the memory map

```
/** \\brief  CAN object */
typedef volatile struct _Ifx_CAN
{
    Ifx_CAN_CLC      CLC;                    /**< \brief 0, CAN Clock Control Register */
    unsigned char    reserved_4[4];          /**< \brief 4, \internal Reserved */
    Ifx_CAN_ID       ID;                     /**< \brief 8, Module Identification Register */
    Ifx_CAN_FDR      FDR;                    /**< \brief C, CAN Fractional Divider Register */
    unsigned char    reserved_10[216];       /**< \brief 10, \internal Reserved */
    Ifx_CAN_OCS      OCS;                    /**< \brief E8, OCDS Control and Status */
    Ifx_CAN_KRSTCLR  KRSTCLR;                /**< \brief EC, Kernel Reset Status Clear Register */
    Ifx_CAN_KRST1    KRST1;                  /**< \brief F0, Kernel Reset Register 1 */
    Ifx_CAN_KRST0    KRST0;                  /**< \brief F4, Kernel Reset Register 0 */
    Ifx_CAN_ACCEN1   ACCEN1;                 /**< \brief F8, Access Enable Register 1 */
    Ifx_CAN_ACCEN0   ACCEN0;                 /**< \brief FC, Access Enable Register 0 */
    Ifx_CAN_LIST     LIST[16];               /**< \brief 100, List Register */
    Ifx_CAN_MSPND    MSPND[8];               /**< \brief 140, Message Pending Register */
    unsigned char    reserved_160[32];       /**< \brief 160, \internal Reserved */
    Ifx_CAN_MSID     MSID[8];                /**< \brief 180, Message Index Register */
    unsigned char    reserved_1A0[32];       /**< \brief 1A0, \internal Reserved */
    Ifx_CAN_MSIMASK  MSIMASK;                /**< \brief 1C0, Message Index Mask Register */
    Ifx_CAN_PANCTR   PANCTR;                 /**< \brief 1C4, Panel Control Register */
    Ifx_CAN_MCR      MCR;                    /**< \brief 1C8, Module Control Register */
    Ifx_CAN_MITR     MITR;                   /**< \brief 1CC, Module Interrupt Trigger Register */
    Ifx_CAN_MECR     MECR;                   /**< \brief 1D0, Measure Control Register */
    Ifx_CAN_MESTAT   MESTAT;                 /**< \brief 1D4, Measure Status Register */
    unsigned char    reserved_1D8[40];       /**< \brief 1D8, \internal Reserved */
    Ifx_CAN_N        N[4];                   /**< \brief 200, Node object */
    unsigned char    reserved_600[2560];     /**< \brief 600, \internal Reserved */
    Ifx_CAN_MO       MO[256];                /**< \brief 1000, Message objects */
    unsigned char    reserved_3000[4096];    /**< \brief 3000, \internal Reserved */
} Ifx_CAN;
```

# Memory-mapped I/O Example

**PANCTR**
**Panel Control Register** **($1C4_H$)** **Reset Value: $0000\ 0301_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PANAR2 | | | | | | | | PANAR1 | | | | | | | |
| rwh | | | | | | | | rwh | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | RBUSY | BUSY | PANCMD | | | | | | | |
| r | | | | | | rh | rh | rwh | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **PANCMD** | [7:0] | rwh | **Panel Command**<br>This bit field is used to start a new command by writing a panel command code into it. At the end of a panel command, the NOP (no operation) command code is automatically written into PANCMD. The coding of PANCMD is defined in **Table 19-7**. |
| **BUSY** | 8 | rh | **Panel Busy Flag**<br>$0_B$ Panel has finished command and is ready to accept a new command.<br>$1_B$ Panel operation is in progress. |
| **RBUSY** | 9 | rh | **Result Busy Flag**<br>$0_B$ No update of PANAR1 and PANAR2 is scheduled by the list controller.<br>$1_B$ A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available. |
| **PANAR1** | [23:16] | rwh | **Panel Argument 1**<br>See **Table 19-7**. |
| **PANAR2** | [31:24] | rwh | **Panel Argument 2**<br>See **Table 19-7**. |
| **0** | [15:10] | r | **Reserved**<br>Read as 0; should be written with 0. |

- r: read
- w: write
- h: hardware

```
typedef struct _Ifx_CAN_PANCTR_Bits
{
    unsigned int PANCMD : 8;                /**< \brief [7:0] Panel Command (rwh) */
    unsigned int BUSY : 1;                  /**< \brief [8:8] Panel Busy Flag (rh) */
    unsigned int RBUSY : 1;                 /**< \brief [9:9] Result Busy Flag (rh) */
    unsigned int reserved_10 : 6;           /**< \brief \internal Reserved */
    unsigned int PANAR1 : 8;                /**< \brief [23:16] Panel Argument 1 (rwh) */
    unsigned int PANAR2 : 8;                /**< \brief [31:24] Panel Argument 2 (rwh) */
} Ifx_CAN_PANCTR_Bits;

typedef union
{
    unsigned int          U;
    signed int            I;
    Ifx_CAN_MO_DATAH_Bits B;
} Ifx_CAN_MO_DATAH;
```

```
#define CAN ((Ifx_CAN *) 0xF0004000)

CAN->PANCTR.U = 0x10101010; /* accessing the whole content */
CAN->PANCTR.B.PANCMD = 0x11; /* accessing a certain field */
```

# Volatile Keyword

- Always use the volatile keyword when accessing device registers
- Volatile tells the compiler "do not use any optimization techniques"
  - What if reading a device register multiple times in a function?
  - What if writing to a device register multiple times in a function?

**Without volatile**

godbolt.org

C++ source #1

```
unsigned long function(void)
{
    *(unsigned long *)(100) = 111;
    *(unsigned long *)(100) = 112;
    *(unsigned long *)(100) = 113;
    return *(unsigned long *)(100);
}
```

ARM gcc trunk (linux) (C++, Editor #1, Compiler #1)

ARM gcc trunk (linux) -O3

```
function():
        movs    r3, #0
        movs    r0, #113
        str     r0, [r3, #100]
        bx      lr
```

Output (0/0) ARM gcc trunk (linux) - cached (2970B) ~165 lines filtered

Apply maximum optimizations

**With volatile**

godbolt.org

C++

```
unsigned long function(void)
{
    *(volatile unsigned long *)(100) = 111;
    *(volatile unsigned long *)(100) = 112;
    *(volatile unsigned long *)(100) = 113;
    return *(volatile unsigned long *)(100);
}
```

ARM gcc trunk (linux) (C++, Editor #1, Compiler #1)

ARM gcc trunk (linux) -O3

```
function():
        movs    r3, #0
        movs    r0, #111
        movs    r1, #112
        movs    r2, #113
        str     r0, [r3, #100]
        str     r1, [r3, #100]
        str     r2, [r3, #100]
        ldr     r0, [r3, #100]
        bx      lr
```

Output (0/0) ARM gcc trunk (linux) - cached (3247B) ~171 lines filtered

# What will happen without volatile?

godbolt.org

COMPILER EXPLORER

Add... More Backtrace intel Solid Sands Share Policies Other

C++ source #1

C++

```cpp
unsigned long function(void)
{
    unsigned long a;
    a = *(volatile unsigned long *)(100);
    a = *(volatile unsigned long *)(100);
    a = *(volatile unsigned long *)(100);
    return a;
}
```

ARM gcc trunk (linux) (C++, Editor #1, Compiler #1)

ARM gcc trunk (linux) -O3

```arm
function():
        movs    r3, #0
        ldr     r2, [r3, #100]
        ldr     r2, [r3, #100]
        ldr     r0, [r3, #100]
        bx      lr
```

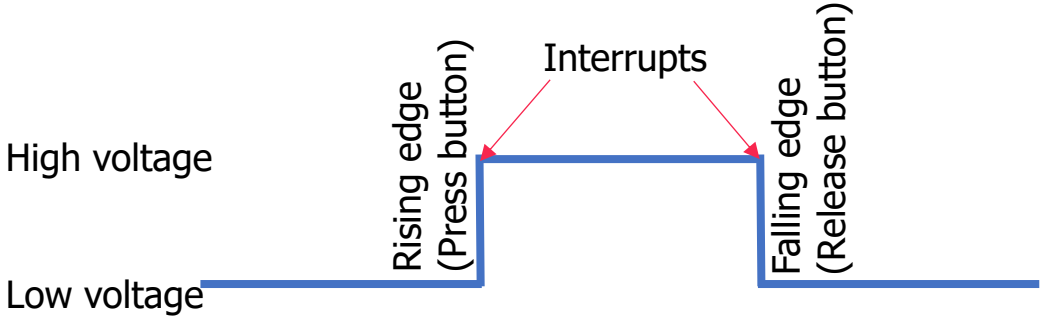
Output (0/0) ARM gcc trunk (linux) - 948ms (3827B) ~219 lines filtered

Read the new cookie policy Compiler Explorer uses cookies and other related techs to serve you Don't consent Consent

# Polling vs. Interrupt

- Polling
  - CPU is always busy
  - No hardware support

- Interrupt
  - CPU is not busy
  - Interrupt service routine (or interrupt handler)
  - Interrupt vector table



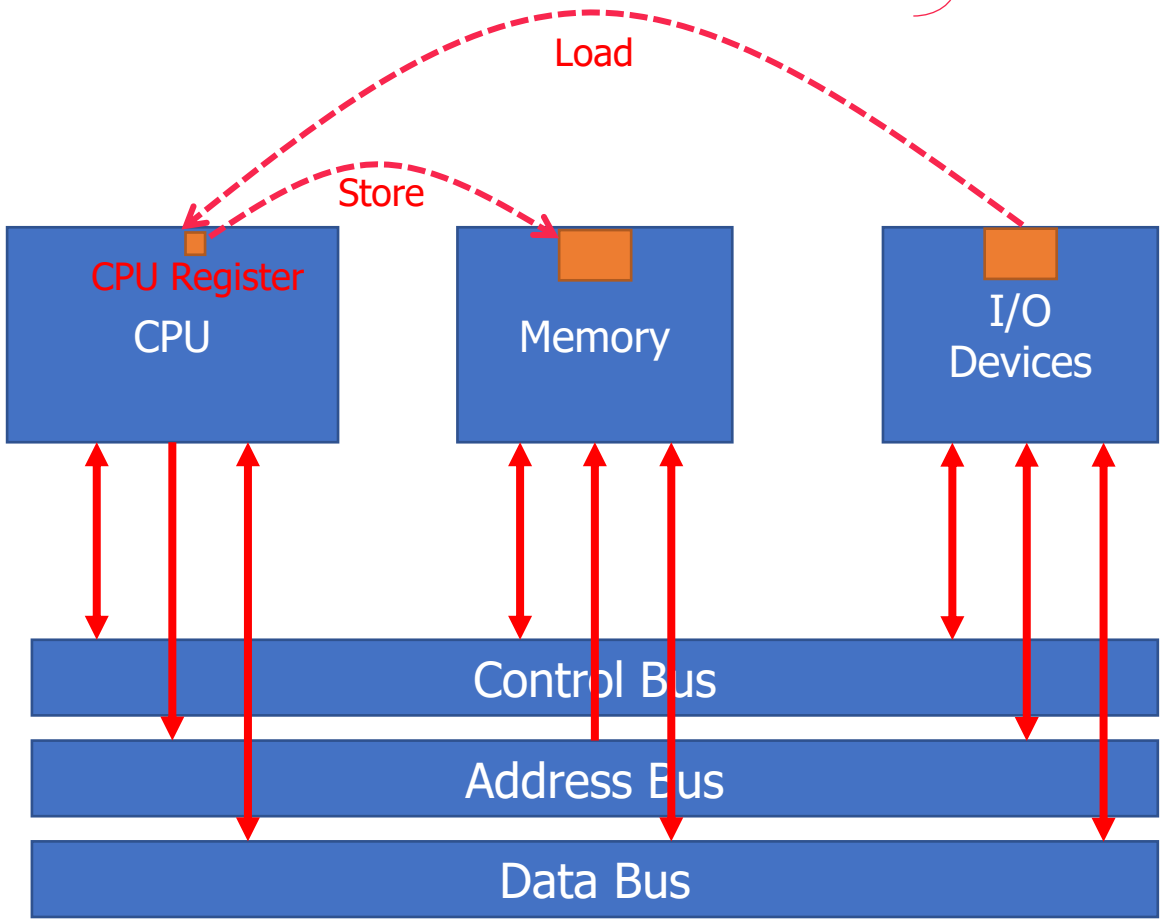

```
while (1) {
    b = read_button_register;
    if (b == pushed) {
        break;
    }
}
turn_on_led;
```

```
ISR(button_isr)
{
    turn_on_led;
}
```

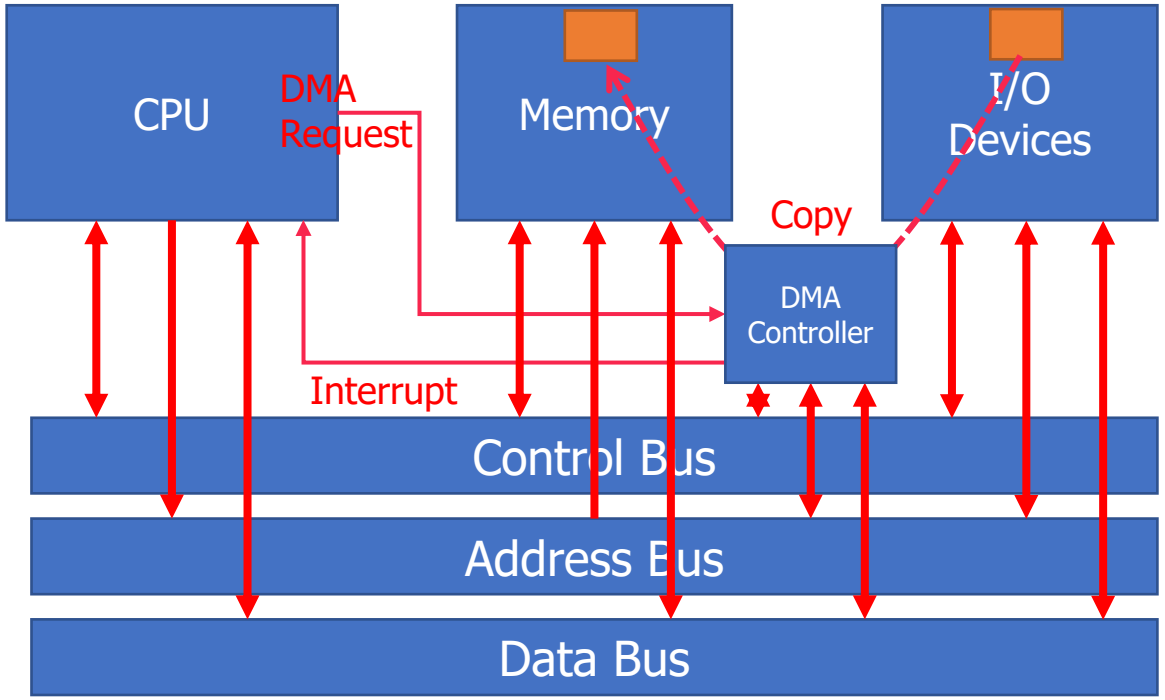
# Memcpy vs. DMA (Direct Memory Access)

- Memcpy
  - Load from device to register
  - Store from register to memory

  Loop




- DMA
  - CPU request DMA to DMA controller
  - DMA controller copy from device to mem
  - DMA controller notify completion to CPU



# Summary

- Memory-mapped I/O vs. Port-mapped I/O
- Polling vs. Interrupt
- Memcpy vs. DMA