# Prise en main du système
# Hello World

MI074 - 2

## Objectif 1/2

L'objectif de cette séance est de présenter en détails la programmation de la première application sur la plateforme. Nous allons voir ainsi (pas forcément dans cet ordre) :

- le mapping mémoire
- le code de boot
- le makefile
- la synchronisation entre les processeurs
- la commande de périphérique
- les directives au compilateur en C
- l'allocation des piles de démarrage

Ce séance est une préparation directe de la séance de TME

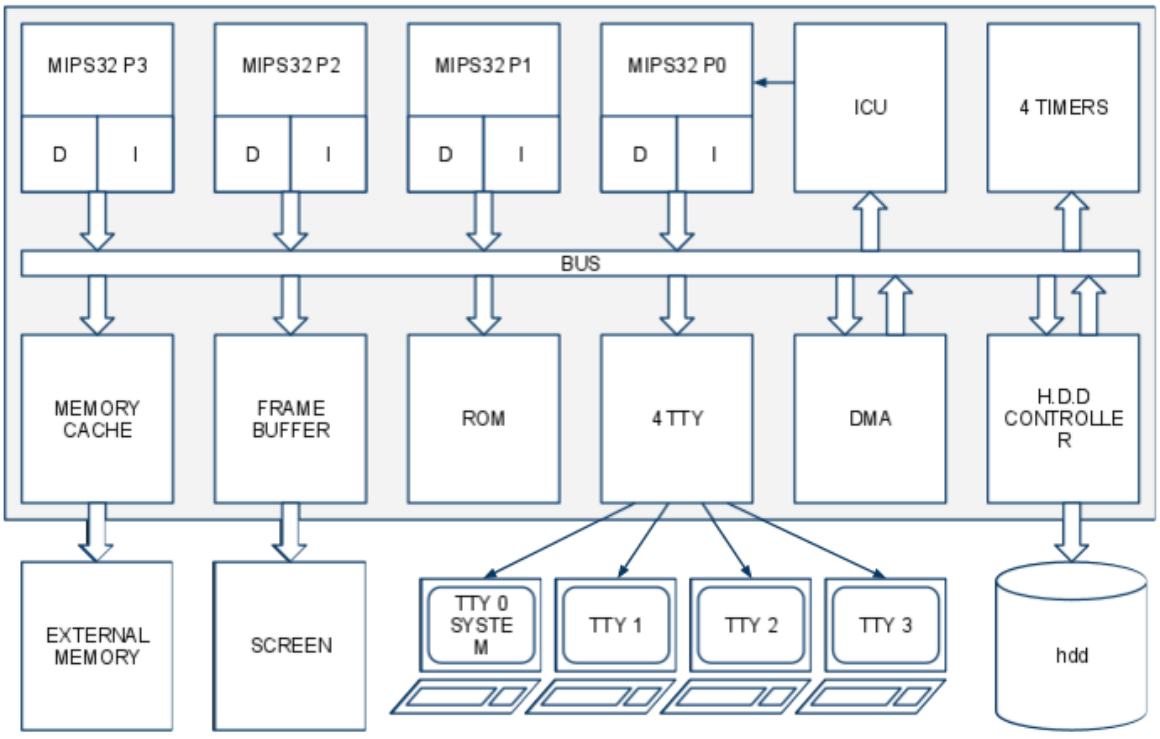
# Objectif



Les programmes supports ne seront pas dans l'OS.
C'est seulement préparatoire.

Chaque processeur affiche "Hello World" sur "son" TTY
- entièrement en assembleur
- en assembleur et en C

# La plateforme matérielle



# description de la mémoire

## fichier : segmentation.h

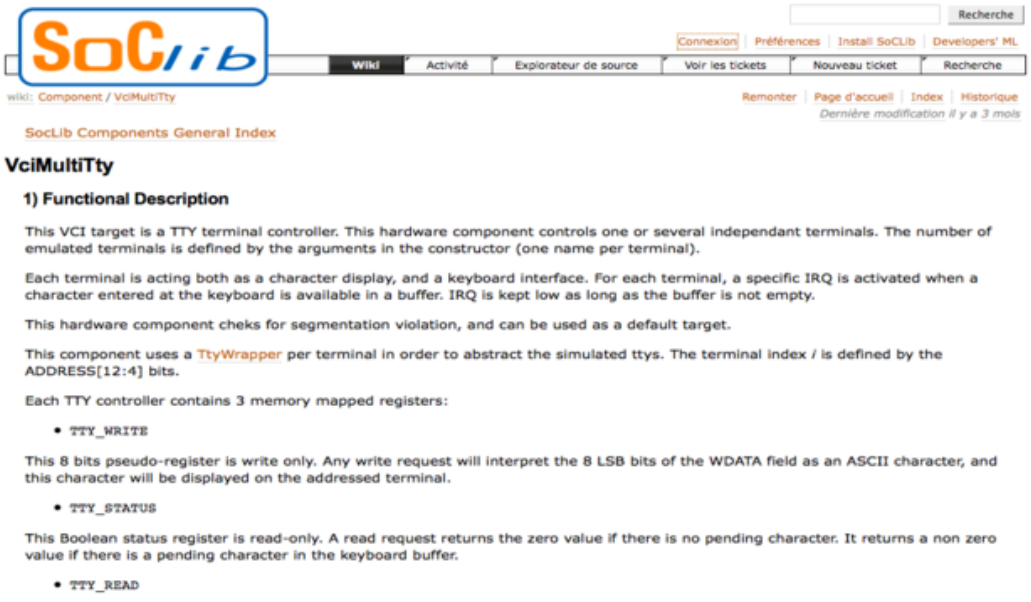
```
// ------- Devce  mapped segments

#define   TIMER_BASE  0xd3200000
#define   TIMER_SIZE   0x00000080

#define   ICU_BASE     0xd2200000
#define   ICU_SIZE     0x00000020

#define   DMA_BASE     0xd1200000
#define   DMA_SIZE     0x00000014

#define   TTY_BASE     0xd0200000
#define   TTY_SIZE     0x00000040

#define   BD_BASE      0xd5200000
#define   BD_SIZE      0x20

#define   FB_XSIZE     512
#define   FB_YSIZE     512
#define   FB_BASE      0x52200000
#define   FB_SIZE      FB_XSIZE*FB_YSIZE*2
```

```
// ------- ROM mapped segments

#define   KTEXT_LMA_BASE 0xbf800000
#define   KTEXT_LMA_SIZE  0x00020000

#define   KDATA_LMA_BASE  0xbf820000
#define   KDATA_LMA_SIZE   0x00020000

#define   BOOT_BASE        0xbfc00000
#define   BOOT_SIZE        0x00001000

// ------- RAM

#define   RAM_BASE         0x7F400000
#define   RAM_SIZE         0x01000000

// ------- Application mapped segments

#define   KTEXT_BASE       0x80000000
#define   KDATA_BASE       0x80020000
#define   KDATA_SIZE       0x003E0000

#define   USR_TEXT_BASE   RAM_BASE
#define   USR_TEXT_SIZE    0x00060000
#define   USR_DATA_BASE   \
          USR_TEXT_BASE + USR_TEXT_SIZE
#define   USR_DATA_SIZE    0x00B9F000
```

# Description des périphériques

https://www.soclib.fr/trac/dev/wiki/Component

pour le TTY :
https://www.soclib.fr/trac/dev/wiki/Component/VciMultiTty



Recherche

Connexion | Préférences | Install SoCLib | Developers' ML

Wiki | Activité | Explorateur de source | Voir les tickets | Nouveau ticket | Recherche

wiki: Component / VciMultiTty

Remonter | Page d'accueil | Index | Historique

*Dernière modification il y a 3 mois*

SocLib Components General Index

**VciMultiTty**

**1) Functional Description**

This VCI target is a TTY terminal controller. This hardware component controls one or several independant terminals. The number of emulated terminals is defined by the arguments in the constructor (one name per terminal).

Each terminal is acting both as a character display, and a keyboard interface. For each terminal, a specific IRQ is activated when a character entered at the keyboard is available in a buffer. IRQ is kept low as long as the buffer is not empty.

This hardware component cheks for segmentation violation, and can be used as a default target.

This component uses a TtyWrapper per terminal in order to abstract the simulated ttys. The terminal index / is defined by the ADDRESS[12:4] bits.

Each TTY controller contains 3 memory mapped registers:

- `TTY_WRITE`

This 8 bits pseudo-register is write only. Any write request will interpret the 8 LSB bits of the WDATA field as an ASCII character, and this character will be displayed on the addressed terminal.

- `TTY_STATUS`

This Boolean status register is read-only. A read request returns the zero value if there is no pending character. It returns a non zero value if there is a pending character in the keyboard buffer.

- `TTY_READ`

# Description des périphériques

Sur la page de description du périphérique on trouve :

The file source:trunk/soclib/soclib/module/connectivity_component/vci_multi_tty/include/soclib/tty.h defines `TTY_WRITE`, `TTY_STATUS`, `TTY_READ` and `TTY_SPAN`.

ficnier : devices.h

```
#ifndef _DEVICES_H_
#define _DEVICES_H_

/* TTY mapped registers offset */
#define TTY_SPAN       4
#define TTY_WRITE_REG  0
#define TTY_STATUS_REG 1
#define TTY_READ_REG   2

#endif
```

# Description de la mémoire pour le linker

fichier kldscript.h

```
#include "segmentation.h"

MEMORY
{
  boot : ORIGIN = BOOT_BASE, LENGTH = BOOT_SIZE
}

SECTIONS
{
 .boot : { *(.boot) } > boot
}
```

# Description de la mémoire pour le linker

## fichier kldscript.h

```
#include "segmentation.h"
MEMORY
{
  boot : ORIGIN = BOOT_BASE, LENGTH = BOOT_SIZE
}
SECTIONS
{
 .boot : { *(.boot) } > boot
}
```

```
#include "segmentation.h"
MEMORY
{
  boot   : ORIGIN = BOOT_BASE, LENGTH = BOOT_SIZE
}
SECTIONS
{
 .boot : { *(.boot) *(.text) *(.boot.*) *(.data) *(.rodata*)  } > boot
}
```

# le code de boot

## fichier : boot.S

```
#include <devices.h>
#include <segmentation.h>

.section .boot ,"ax",@progbits
.ent    boot
.align  2

boot:
    li      $26,    0
    mtc0    $26,    $12     # Status Register
    mfc0    $4,     $0
    la      $5,     str
    j       tty_puts
```

```
//  Function name : fputs
//  Arguments     : $4 <-> processor id, $5 <-> @str
//  Description   : print charecters of string str
//  ------------------------------------------------------------------
tty_puts:
    la      $26,    TTY_BASE
    li      $27,    TTY_SPAN * 4
    multu   $4,     $27
    mflo    $27
    addu    $26,    $26,    $27
loop:
    lbu     $27,    0($5)
    beqz    $27,    deadLoop
    sw      $27,    (TTY_WRITE_REG)($26)
    addiu   $5,     $5,     1
    j       loop
deadLoop:
    j       deadLoop

str:    .asciiz "hello world"

.end boot
```