REDS
Reconfigurable & embedded Digital Systems

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Architecture and Drivers for Smartphones
Hardware Architerure and Interfaces

Cours APS

Salvatore Valenza

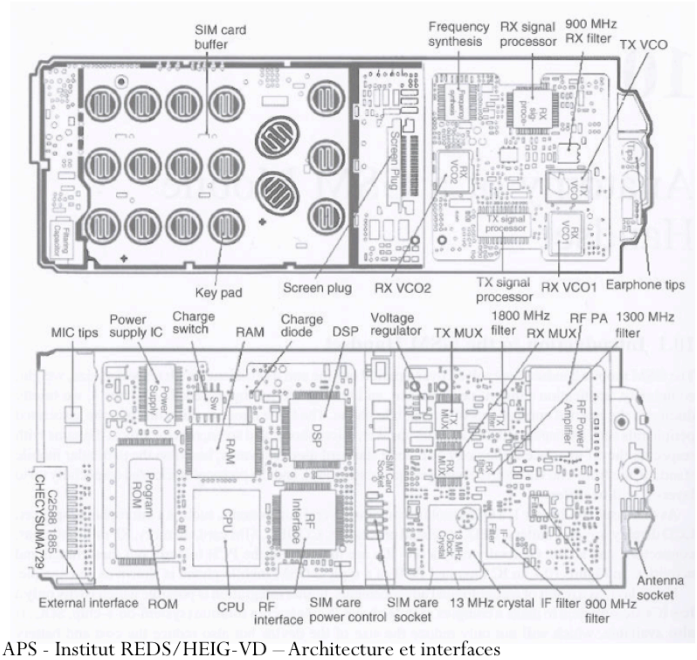
Version 1.0 (2012-2013)

Plan

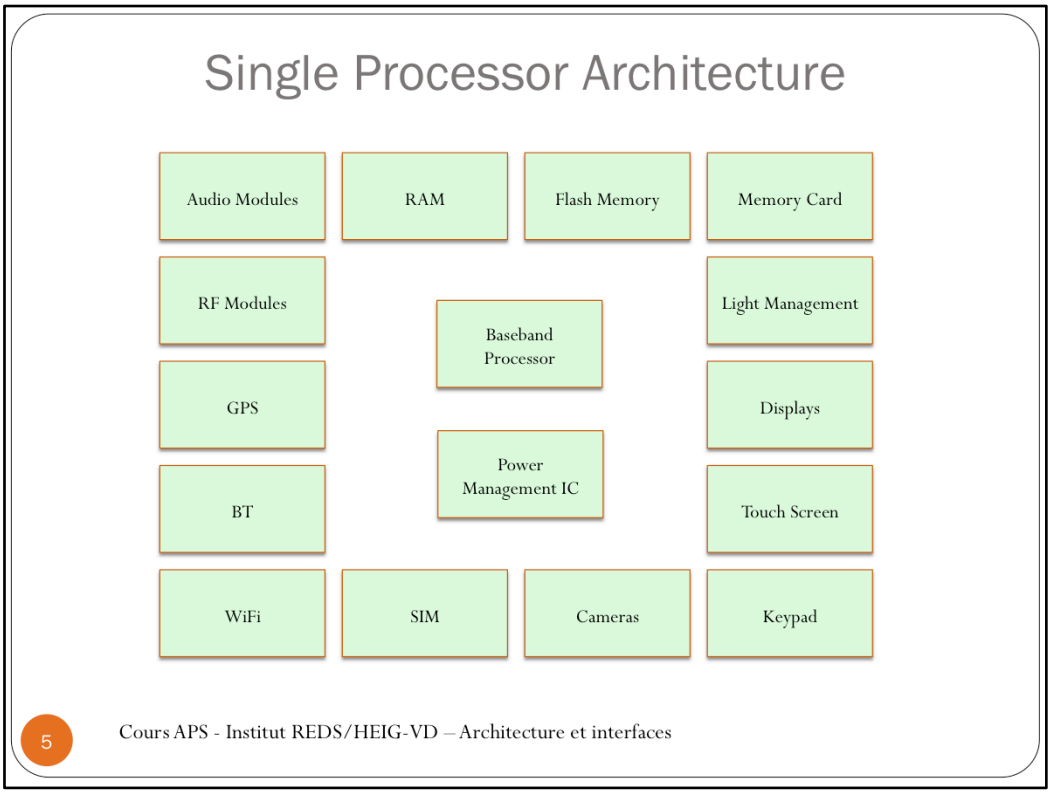
- Architecture matérielle
- Mémoires
- Interfaces principales (rappels)
- Gestion matériel / pilote de périphérique (rappels)

Architecture matérielle

Generic PCB Architecture

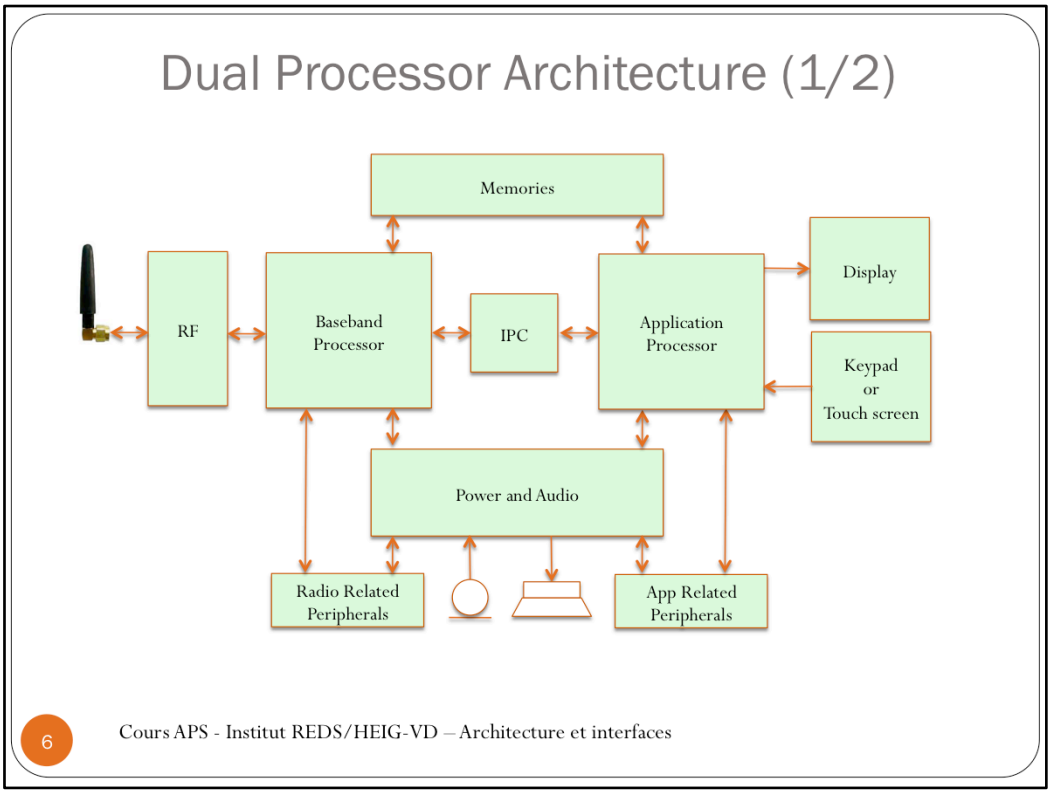


Cours APS - Institut REDS/HEIG-VD – Architecture et interfaces



Dans l'architecture Single Processor le SoC principal gère toutes les périphériques. Le processeur principal peut être un SoC avec un Micro et un DSP pour la gestion des algorithmiques de codification/décodification audio et radio.

Au fur et à mesure que les applications augmentent, cette architecture présente des limites au niveau des performances.

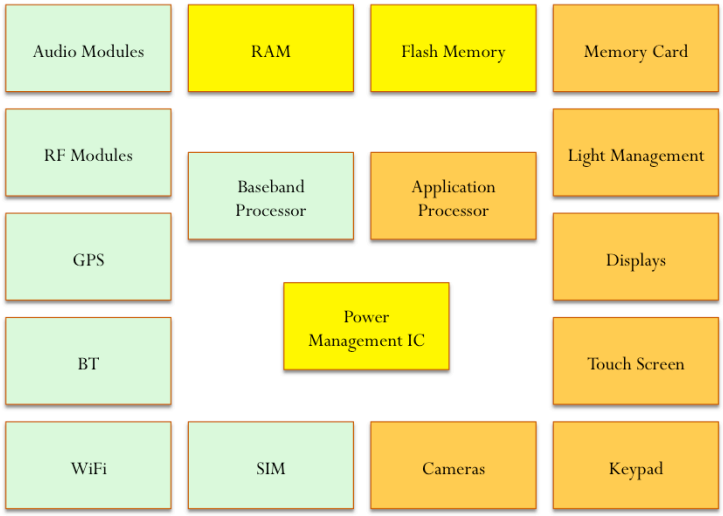


Dans l'architecture Dual Processor on sépare les applications des routines liée à la gestion Radio, grâce à l'introduction d'un Application Processor (AP) et d'un Baseband Processor (BP).

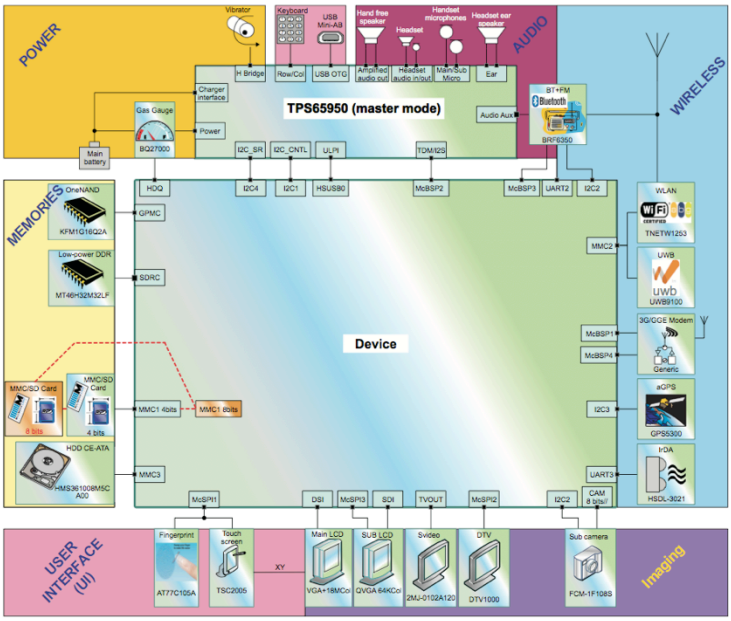
Le BP, d'habitude, utilise un OS propriétaire, par contre l'AP utilise un OS « general pourpose » comme Android, Symbian, Windows etc.

La communication entre AP et BP est gérée avec un IPC (Inter Processor Controller) en utilisant la mémoire partagée et des contrôleurs d'interruptions.

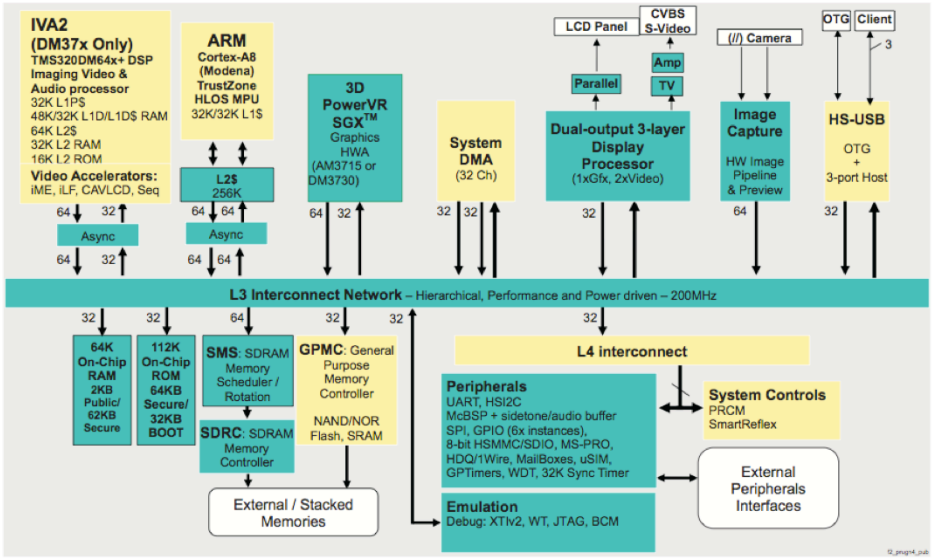
Dual Processor Architecture (2/2)



Example: Application SoC TI OMAP 3730 (1/2)



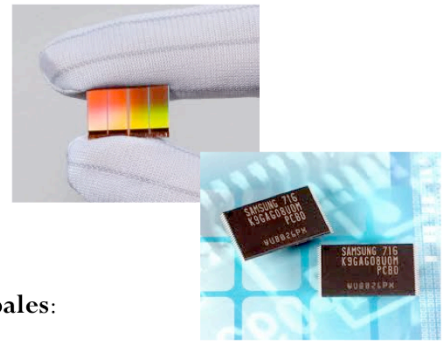
Example: Application SoC TI OMAP 3730 (2/2)



Mémoires

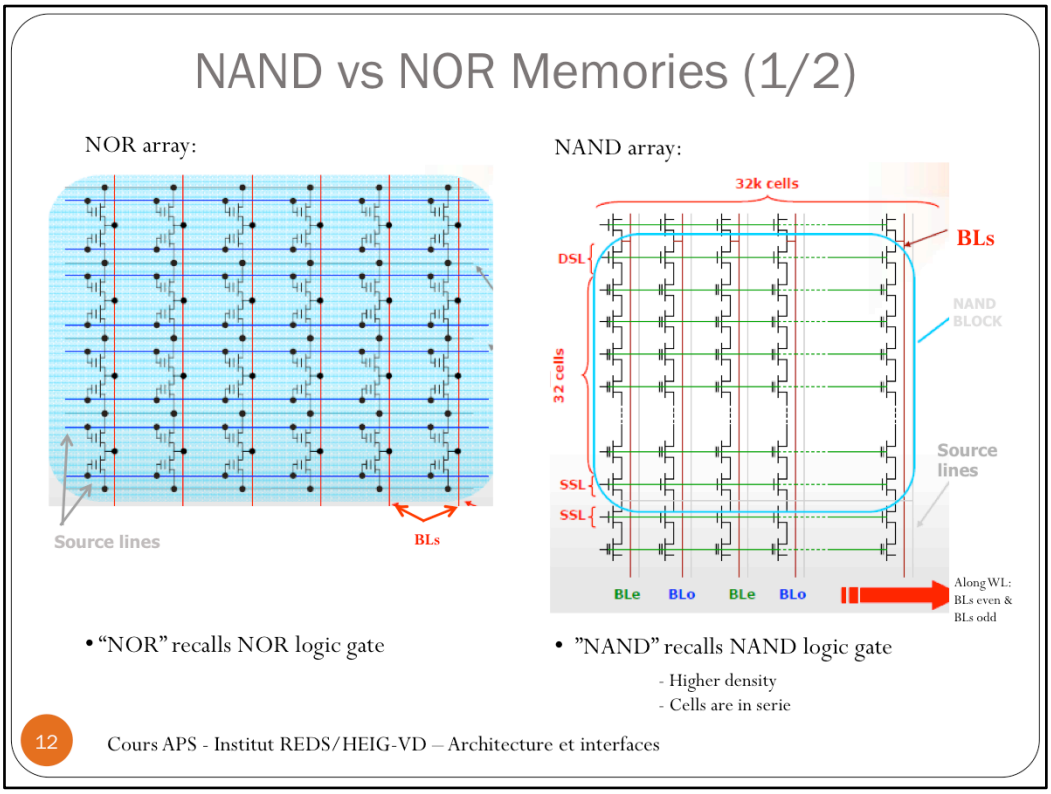
Flash Memory

Elle contiens:
- Boot Loader
- Phone Software
- Product related data
- User's Data
- User's files
- Operator customization



2 méthodes d'exécution principales:
1. Execution In Place (XIP)
2. Execution Off Place

2 types de mémoires:
1. NOR Mémoires
2. NAND Mémoires

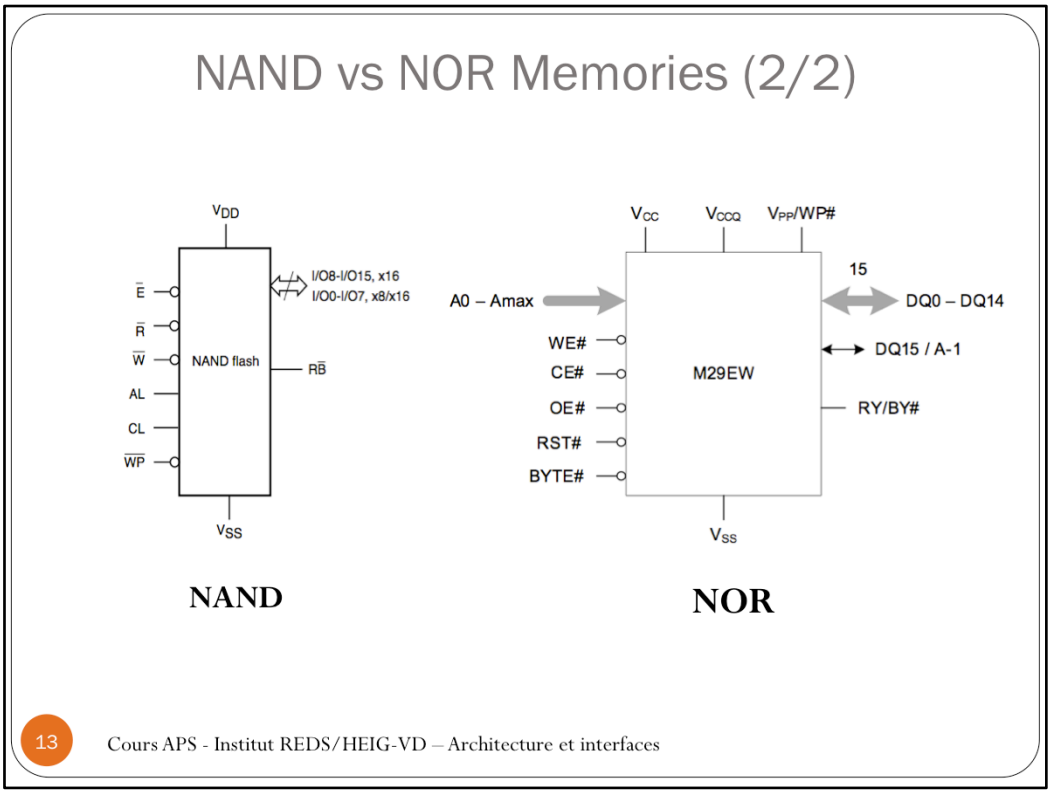


Technogie
La mémoire flash est un type d'EEPROM qui permet la modification de plusieurs espaces mémoires en une seule opération. La mémoire flash est donc plus rapide lorsque le système doit écrire à plusieurs endroits en même temps.La mémoire flash utilise comme cellule de base un transistor MOS possédant une grille flottante enfouie au milieu de l'oxyde de grille, entre le canal et la grille. L'information est stockée grâce au piégeage d'électrons dans cette grille flottante. Deux mécanismes sont utilisés pour faire traverser l'oxyde aux électrons : l'injection d'électrons chauds ;

NOR
Le flash NOR, inventée par Fujio Masuoka, un employé de Toshiba, fut la première à être développée commercialement par Intel en 1988. Les temps d'effacement et d'écriture sont longs mais elle possède une interface d'adressage permettant un accès aléatoire et rapide à n'importe quelle position. Le stockage des données est 100 % garanti par le fabricant.Elle est adaptée à l'enregistrement de données informatiques destinées à être exécutées directement à partir de cette mémoire. Cette caractéristique est appelée XIP (eXecute In Place). La memoire NOR est particulièrment bien adaptée a contenir l'OS par exemple dans les téléphones portables (principal marché des Flash NOR), les décodeurs télés, les cartes mères ou leurs périphériques (imprimantes, appareils photo, etc.) du fait que le code peut y être directement executé (XIP, In Place eXecution).Du fait de son coût, bien plus élevé que celui de la NAND et de sa densité limitée, elle n'est en général pas utilisée pour le stockage de masse.

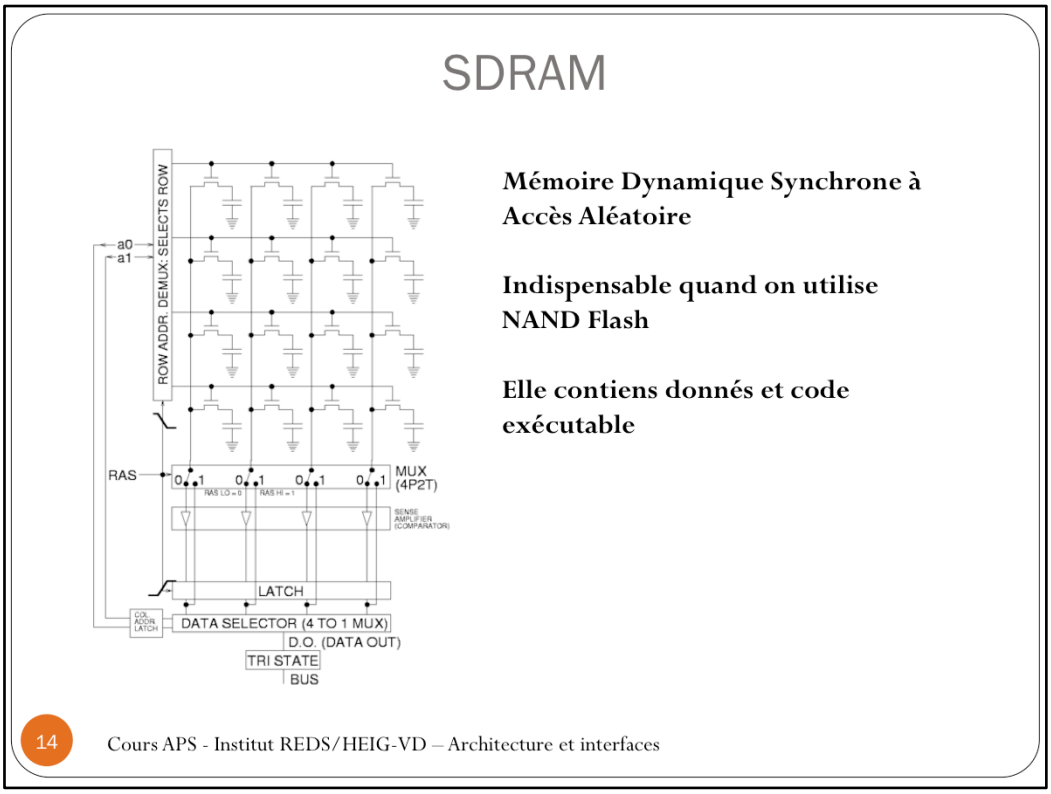
NAND
Le flash NAND suivit en 1989, commercialisée par Toshiba. Elle est plus rapide à l'effacement et à l'écriture, offre une plus grande densité et un coût moins important par bit. Toutefois son interface d'entrée / sortie n'autorise que l'accès séquentiel. Cela tend à limiter — au niveau du système — sa vitesse effective de lecture, et à compliquer le démarrage direct à partir d'une mémoire NAND. De ce fait elle est moins bien adaptée que la NOR pour exécuter du code machine. Du fait de son prix moins élevé, elle est présente dans de nombreux assistants et téléphones portables en utilisant par blocs la mémoire RAM en mode page comme support d'exécution.
Le fabricant en général ne garantit pas le stockage des données à 100 % mais un taux d'erreurs inférieur à une limite donnée. Cette fiabilité limitée nécessite la mise en place d'un système de gestion des erreurs (ECC - Error Code Correction, Bad blocks management, etc.) au niveau de l'application — comme cela est le cas, par exemple, pour les disques durs. Elle est donc utilisée pour le stockage d'informations. Quasiment toutes les mémoires de masse externes Carte MMC, Carte SD et Carte MS utilisent cette technologie.
Samsung débute fin 2009 la production d'une puce de 4 Go de mémoire flash NAND, gravée en 30 nm, et dont la particularité est d'avoir une interface de type DDR (double data rate). Bien que ces dernières souffrent encore de problèmes de performances qui font qu'elles ne sont pas utilisées sur les SSD, ces puces offriraient un débit 3 fois supérieur à celles basées sur une interface SDR (single data rate).

Source: Wikipedia



NOR
Possibilité d'exécution "in place" (SDRAM pas indispensable).

NAND
Pas de « XIP », mais majeur niveau d'intégration et plus à bon marché (avec le même prix de chip on a beaucoup plus de mémoire par rapport à une mémoire NAND.
Par contre on a besoin d'une SDRAM pour la gestion de l'exécution du code.



SDRAM ou Synchronous Dynamic Random Access Memory (en français, Mémoire Dynamique Synchrone à Accès Aléatoire) est un type particulier de mémoire vive ayant une interface de communication synchrone. Jusqu'à son apparition, les mémoires DRAM étaient habituellement asynchrones, cela signifie qu'il fallait attendre un temps donné par les caractéristiques de la mémoire pour obtenir un résultat après une modification de l'état des entrées.

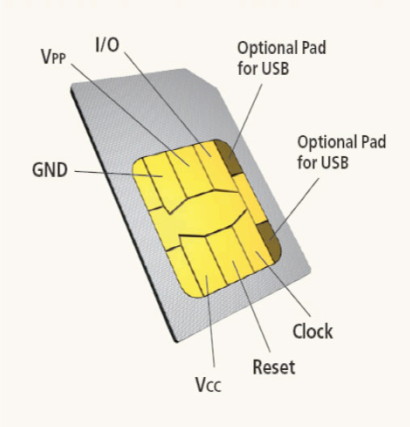
Principe

Contrairement aux mémoires asynchrones, une mémoire SDRAM attend un front d'horloge pour prendre en compte l'état des signaux d'entrées. Cette horloge (habituellement synchrone au Front side bus du processeur) permet de piloter une machine à états finis afin de pipeliner les instructions entrantes.

Le pipelining permet de commencer à traiter une opération avant que l'opération précédente ne soit terminée. Par exemple, après une opération d'écriture "pipeliné", cela permet de commencer le traitement d'une nouvelle opération avant que l'écriture ne soit effective dans la mémoire (plusieurs cycles d'horloge). Dans le cas d'une lecture, cela implique que la donnée n'est disponible qu'après un certain nombre de cycles (pendant lesquels d'autres opérations pourront être commandées), on parle ici de latence de la mémoire, paramètre essentiel au bon fonctionnement du système.

Source: Wikipedia

SIM Card



Subscriber Indentify Module

Elle Contients:

- **IMSI – International Mobile System Identifier**
  - **IMSI = MCC + MNC + MSIN**
  - **MCC – Mobile Country Code**
  - **MNC – Mobile Network Code**
  - **MSIN – Mobile Subscription Identification Number**
- **Encryption keys**
- **User data**

15 Cours APS - Institut REDS/HEIG-VD – Architecture et interfaces

La carte SIM (de l'anglais Subscriber Identity Module) est une puce contenant un microcontrôleur et de la mémoire. Elle est utilisée en téléphonie mobile pour stocker les informations spécifiques à l'abonné d'un réseau mobile, en particulier pour les réseaux de type GSM, UMTS et LTE. Elle permet également de stocker des données et des applications de l'utilisateur, de son opérateur ou dans certains cas de tierces parties.

La carte SIM contient l'identifiant de l'abonné (n° IMSI) et de l'opérateur mobile qui a édité la carte (MCC + MNC).

Généralités

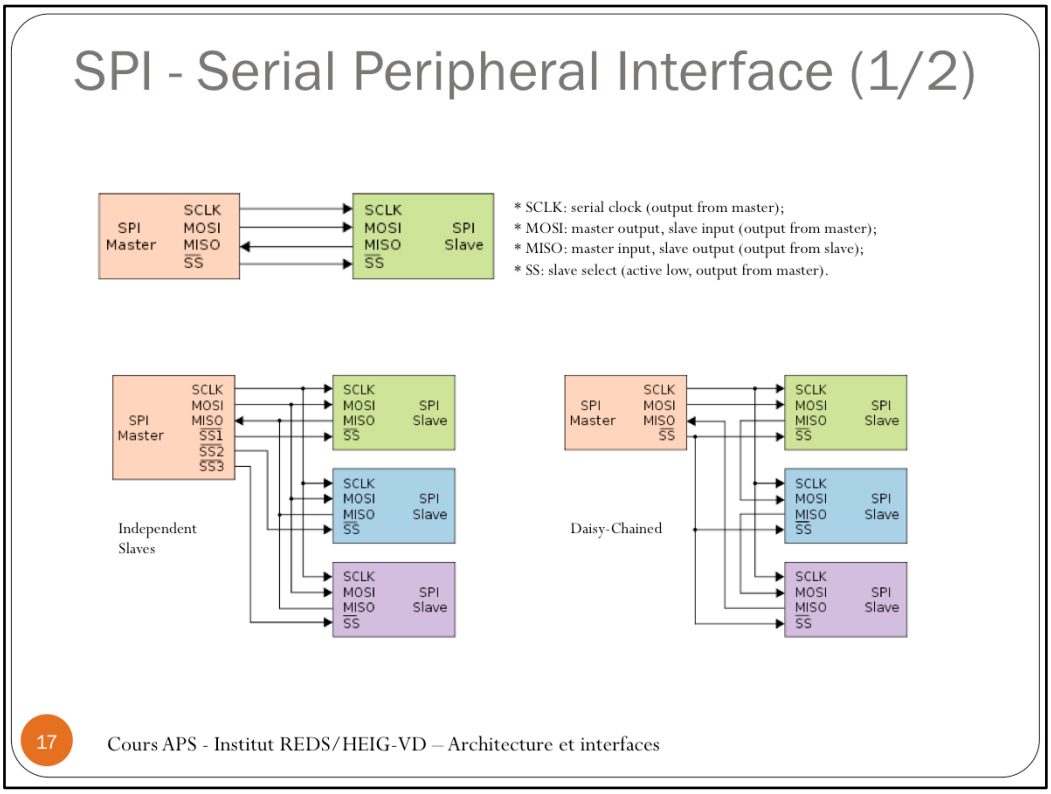
Le choix de l'intégration d'une carte à puce dans les systèmes de téléphonie mobile est basé sur la nécessité de disposer des éléments suivants :

* un élément sécurisé contenant l'identifiant et les données de connexion d'un utilisateur donné ;
* un élément amovible permettant de personnaliser un nouveau téléphone avec les données de connexion ; l'intérêt est de séparer le choix d'un terminal de la notion d'abonnement ;
* un espace de stockage d'information pour les données personnelles de l'abonné (son annuaire), mais également des paramètres de personnalisation de son terminal (paramètres de messagerie, etc.) ;
* un espace pour les applications de l'opérateur de téléphonie.

Il convient de signaler que le terme "carte SIM" est un abus de langage (un raccourci en fait), SIM désignant en fait l'application GSM, définie dans la spécification ETSI/3GPP TS 51.011[1], qui réside sur une carte, nommée UICC (pour « Universal Integrated Circuit Card »).
L'UICC et l'application SIM gèrent l'authentification de l'abonné dans le réseau GSM (l'USIM pour le réseau UMTS) et génèrent des clés qui permettent le chiffrement du flux de données, ceci étant réalisé au sein du terminal mobile.

Une carte SIM donnée est issue d'un seul opérateur ou d'un seul MVNO (français ou étranger) et permet son identification univoque (grâce aux codes MCC + MNC + MSIN intégrés dans la carte).

Interfaces

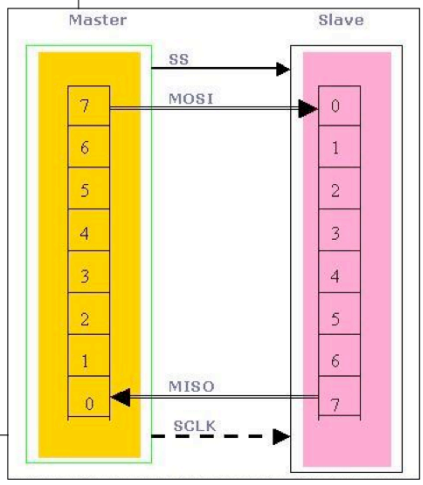


Le bus SPI est très utilisé pour interfacer des périphériques aux processeurs. Tous les périphériques SPI reçoivent une clock du master, un bus d'écriture MOSI (master vers slave), un bus de lecture MISO (slave vers master) et un signal d'enable . Les opérations s'effectuent en mode master /slave. Le master active l'enable du périphérique avec lequel il veut communiquer et envoie une commande en série sur MOSI suivie de lecture ou écriture de data.

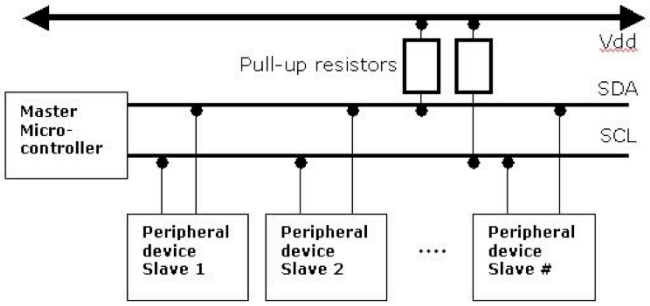
Source: Cours ASP (Chapitre BUS) et Wikipedia

SPI - Serial Peripheral Interface (2/2)

```
unsigned char SPIBitBang8BitsMode0(unsigned char byte)
{
    unsigned char bit;

    for (bit = 0; bit < 8; bit++) {
        /* write MOSI on falling edge of previous clock */
        if (byte & 0x80)
            SETMOSI();
        else
            CLRMOSI();
        byte <<= 1;

        /* read MISO */
        byte |= READMISO();

        SETCLK();

        /* delay between rise and fall of clock  */
        /* gives the h/w time to setup MISO line */
        SPIDELAY(SPISPEED);

        CLRCLK();
    }

    return byte;
}
```

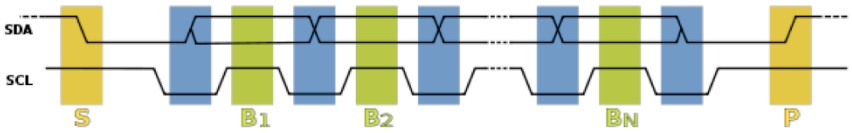


I2C - Inter-Integrated Circuit (1/3)



SDA: Serial DAta line;
SCL: Serial CLock

Fig- 1 (Basic I²C blocks)



I2C - Inter-Integrated Circuit (2/3)

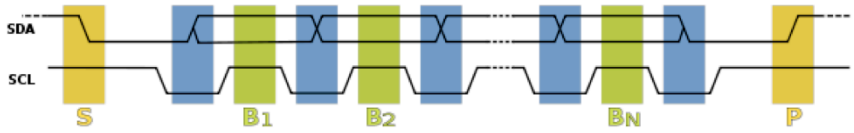
```
/* Global Data */
bool started = false;

void i2c_start_cond(void)
{
        /* if started, do a restart cond */
        if (started) {
                /* set SDA to 1 */
                READSDA();
                I2CDELAY();
                /* Clock stretching */
                while (READSCL() == 0)
                        ;  /* You should add timeout to this loop */
                /* Repeated start setup time, minimum 4.7us */
                I2CDELAY();
        }
        if (READSDA() == 0)
                ARBITRATION_LOST();
        /* SCL is high, set SDA from 1 to 0 */
        CLRSDA();
        I2CDELAY();
        CLRSCL();
        started = true;
}

void i2c_stop_cond(void)
{
        /* set SDA to 0 */
        CLRSDA();
        I2CDELAY();
        /* Clock stretching */
        while (READSCL() == 0)
                ;  /* You should add timeout to this loop */
        /* Stop bit setup time, minimum 4us */
        I2CDELAY();
        /* SCL is high, set SDA from 0 to 1 */
        if (READSDA() == 0)
                ARBITRATION_LOST();
        I2CDELAY();
        started = false;
}
```

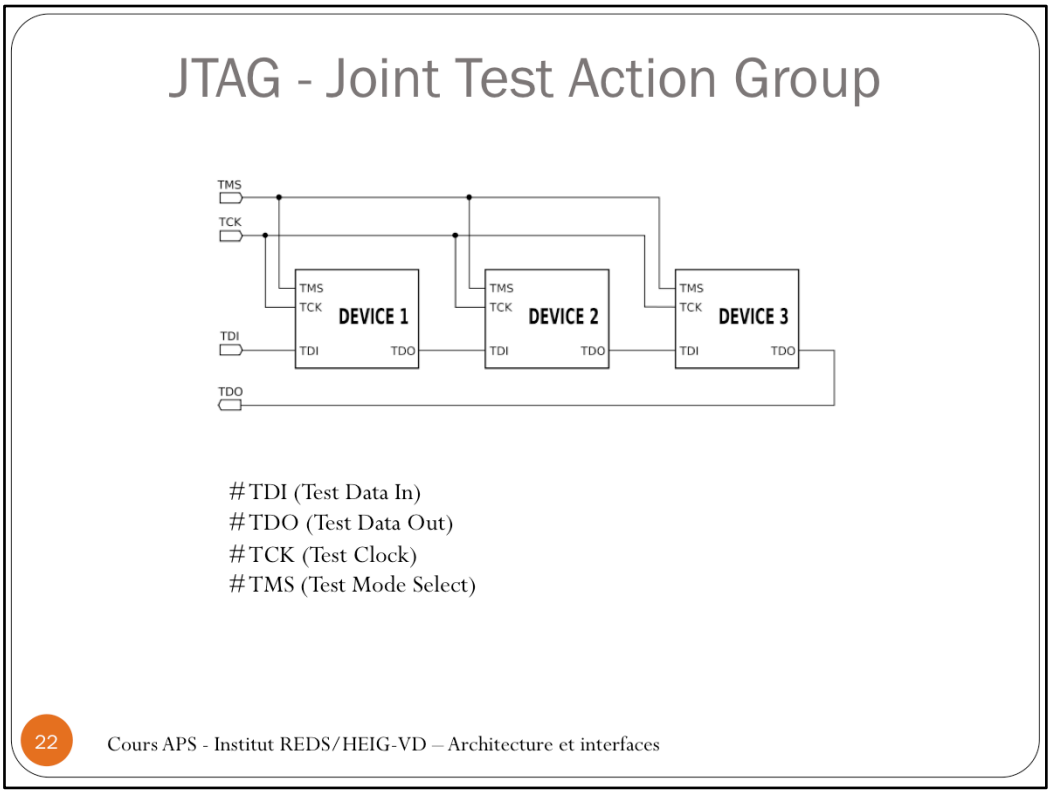
```
/* Write a bit to I2C bus */
void i2c_write_bit(bool bit)
{
        if (bit)
                READSDA();
        else
                CLRSDA();
        I2CDELAY();
        /* Clock stretching */
        while (READSCL() == 0)
                ;  /* You should add timeout to this loop */
        /* SCL is high, now data is valid */
        /* If SDA is high, check that nobody else is driving SDA */
        if (bit && READSDA() == 0)
                ARBITRATION_LOST();
        I2CDELAY();
        CLRSCL();
}

/* Read a bit from I2C bus */
bool i2c_read_bit(void)
{
        bool bit;
        /* Let the slave drive data */
        READSDA();
        I2CDELAY();
        /* Clock stretching */
        while (READSCL() == 0)
                ;  /* You should add timeout to this loop */
        /* SCL is high, now data is valid */
        bit = READSDA();
        I2CDELAY();
        CLRSCL();
        return bit;
}
```

I2C - Inter-Integrated Circuit (3/3)

```
/* Write a byte to I2C bus.
 * Return 0 if ack by the slave */
bool i2c_write_byte(bool send_start,
                    bool send_stop,
                    unsigned char byte)
{
        unsigned bit;
        bool nack;
        if (send_start)
                i2c_start_cond();
        for (bit = 0; bit < 8; bit++) {
                i2c_write_bit((byte & 0x80) != 0);
                byte <<= 1;
        }
        nack = i2c_read_bit();
        if (send_stop)
                i2c_stop_cond();
        return nack;
}
```

```
/* Read a byte from I2C bus */
unsigned char i2c_read_byte(bool nack,
                            bool send_stop)
{
        unsigned char byte = 0;
        unsigned bit;
        for (bit = 0; bit < 8; bit++)
                byte = (byte << 1) |
                       i2c_read_bit();
        i2c_write_bit(nack);
        if (send_stop)
                i2c_stop_cond();
        return byte;
}
```





Debugging

Although it was originally designed for testing printed circuit board assemblies, today JTAG is also used for accessing sub-blocks of integrated circuits, making it an essential mechanism for debugging embedded systems which may not support any other debug-capable communications channel. On most systems, JTAG-based debugging is available from the very first instruction after CPU reset, letting it support development of early boot software which runs before anything is set up. A so-called in-circuit emulator (or more correctly, "JTAG adapter") uses JTAG as the transport mechanism to access on-chip debug modules inside the target CPU. Those modules let software developers debug the software of an embedded system directly at the machine instruction level when needed, or (more typically) in terms of high level language source code.

Debug support is for many software developers the main reason to be interested in JTAG. There are entire debugging architectures built up using JTAG, such as ARM CoreSight and Nexus (plus vendor-specific ones that may not be documented except under NDA) helping move JTAG-centric debugging environments away from early processor-specific designs. Processors can normally be halted, single stepped, or let run freely. Code breakpoints are supported, both for code in RAM (often using a special machine instruction) and in ROM/flash. Data breakpoints are often available, as is bulk data download to RAM. Most designs support “halt mode debugging”, but some allow debuggers to access registers and data busses without needing to halt the core being debugged. Some toolchains can use ARM Embedded Trace Macrocell (ETM) modules to trigger debugger (or tracing) activity on complex hardware events, like a logic analyser programmed to ignore the first seven accesses to a register from one particular subroutine

Storing firmware

Besides debugging, another application of JTAG is allowing device programmer hardware to transfer data into internal non-volatile device memory (e.g. CPLDs). Some device programmers serve a double purpose for programming as well as debugging the device. In the case of FPGAs, volatile memory devices can also be programmed via the JTAG port normally during development work. In addition, newer parts, for instance Xilinx Virtex-5, have internal monitoring capability (temperature, voltage and current) accessible via the JTAG port.

JTAG programmers are also used to write software and data into flash memory. This is usually done using data bus access like the CPU would use, and is sometimes actually handled by a CPU, but in other cases memory chips have JTAG interfaces themselves. Some modern debug architectures, like ARM CoreSight and Nexus, provide internal and external bus master access without needing to halt and take over a CPU. In the worst case, it is usually possible to drive external bus signals using boundary scan support.

As a practical matter, when developing an embedded system, emulating the instruction store is the fastest way to close the edit-compile-test cycle loop. This is because the in-circuit emulator simulating an instruction store can be updated very quickly from the development host, via USB, say. Using e.g. a serial UART port and bootloader to upload firmware makes this edit-compile-test cycle quite slow. Installing firmware via JTAG is intermediate between these extremes, as well as in cost of hardware tools.

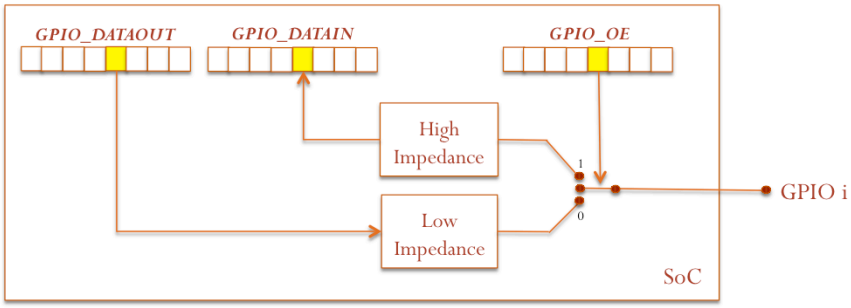
Boundary scan testing

In many ICs today, all the pins that connect to electronic logic are linked together in a set called the Boundary Scan chain. By using JTAG to manipulate the chip's external interface (inputs and outputs to other chips) it is possible to test for certain faults, caused mainly by manufacturing problems. By using JTAG to manipulate its internal interface (to on-chip registers), the combinational logic can be tested.

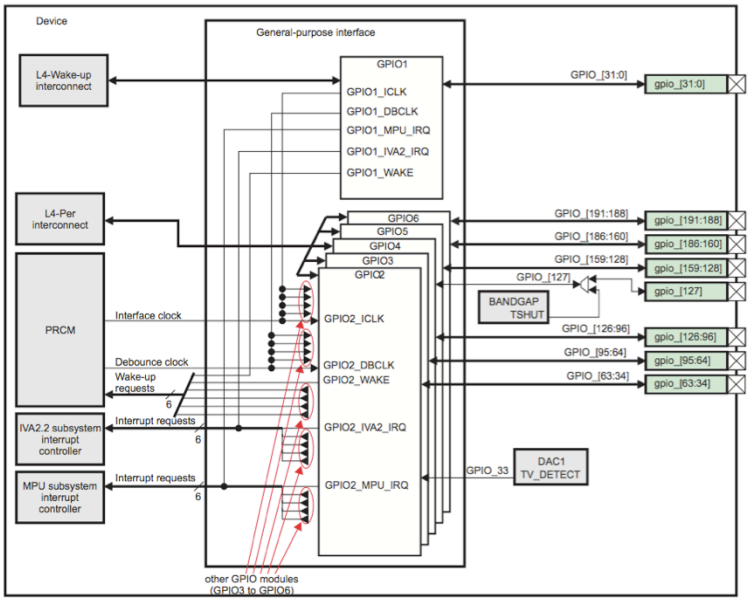
General Purpose Input Output

General Purpose Interface: PINs du SoC que on peut utiliser pour de fonctions spécifiques de la board où le SoC se trouve.

Ils peuvent être configurée soit comme input soit comme output



GPIO: Example OMAP 3730 (1/3)



GPIO: Example OMAP 3730 (2/3)

Module Name	Base Address
GPIO1	0x4831 0000
GPIO2	0x4905 0000
GPIO3	0x4905 2000
GPIO4	0x4905 4000
GPIO5	0x4905 6000
GPIO6	0x4905 8000

Register Name	Type	Register Width (Bits)	Address Offset
GPIO_REVISION	R	32	0x000
GPIO_SYSCONFIG	RW	32	0x010
GPIO_SYSSTATUS	R	32	0x014
GPIO_IRQSTATUS1	RW	32	0x018
GPIO_IRQENABLE1	RW	32	0x01C
GPIO_WAKEUPENABLE	RW	32	0x020
GPIO_IRQSTATUS2	RW	32	0x028
GPIO_IRQENABLE2	RW	32	0x02C
GPIO_CTRL	RW	32	0x030
GPIO_OE	RW	32	0x034
GPIO_DATAIN	R	32	0x038
GPIO_DATAOUT	RW	32	0x03C
GPIO_LEVELDETECT0	RW	32	0x040
GPIO_LEVELDETECT1	RW	32	0x044
GPIO_RISINGDETECT	RW	32	0x048
GPIO_FALLINGDETECT	RW	32	0x04C
GPIO_DEBOUNCENABLE	RW	32	0x050
GPIO_DEBOUNCINGTIME	RW	32	0x054
GPIO_CLEARIRQENABLE1	RW	32	0x060
GPIO_SETIRQENABLE1	RW	32	0x064
GPIO_CLEARIRQENABLE2	RW	32	0x070
GPIO_SETIRQENABLE2	RW	32	0x074
GPIO_CLEARWKUENA	RW	32	0x080
GPIO_SETWKUENA	RW	32	0x084
GPIO_CLEARDATAOUT	RW	32	0x090
GPIO_SETDATAOUT	RW	32	0x094

GPIO: Example OMAP 3730 (3/3)

GPIO_OE

Description	This register is used to enable the pins output capabilities. Its only function is to carry the pads configuration.
Type	RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUTPUTEN																															

Bits	Field Name	Description	Type	Reset
31:0	OUTPUTEN	Output Data Enable 0x0: The corresponding GPIO port is configured as output 0x1: The corresponding GPIO port is configured as input	RW	0xFFFFFFFF

GPIO_DATAOUT

Description	This register is used for setting the value of the GPIO output pins
Type	RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAOUTPUT																															

Bits	Field Name	Description	Type	Reset
31:0	DATAOUTPUT	Output Data	RW	0x00000000

GPIO_DATAIN

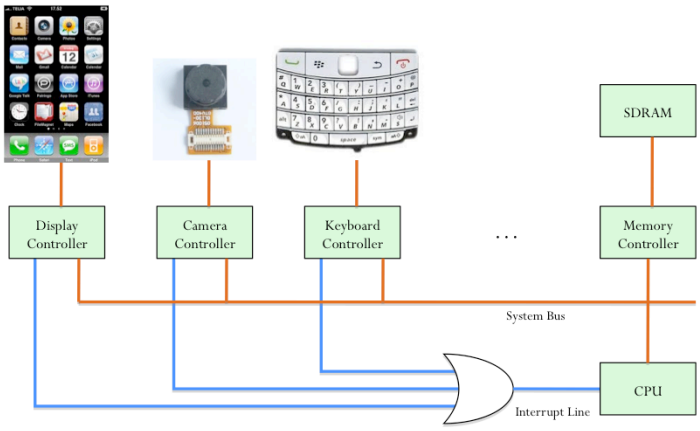
Description	This register is used to register the data that is read from the GPIO pins.
Type	R

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAINPUT																															

Bits	Field Name	Description	Type	Reset
31:0	DATAINPUT	Sampled Input Data	R	0x00000000

Gestion des interactions entre le matériel et le pilote de périphérique

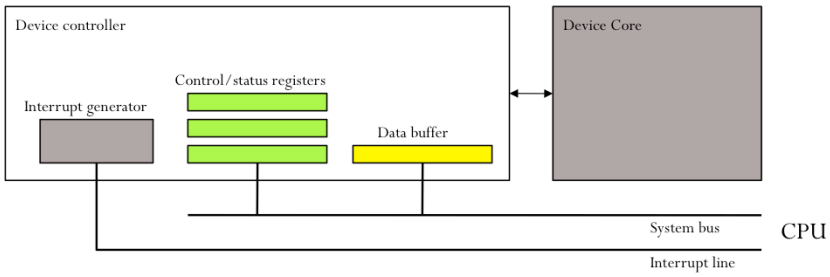
Périphériques vs CPU



Device Controller

Les contrôleurs se composent habituellement de:

- Data buffers
- Registres d'état et de contrôle
- Générateurs d'interruptions



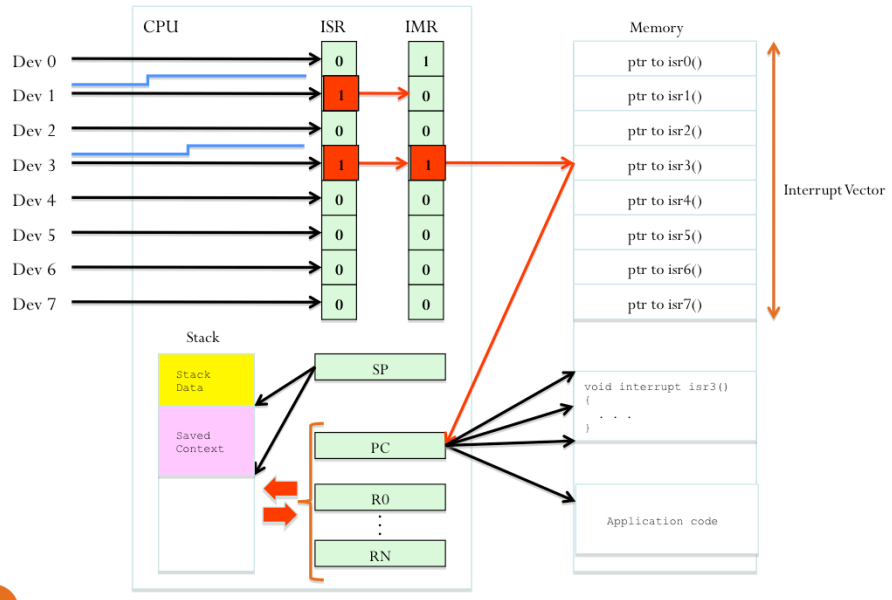
Méthodes de pilote de périphérique

Les pilotes peuvent gérer les interactions avec les contrôleurs de 2 façons principales:

1. Via «polling»: Les pilotes regardent périodiquement l'état des registres de la périphérique pour gérer les data buffers.

2. Via interruptions: Les pilotes sont réveillés par des signaux d'interruption, indiquant que le data buffer est prêt à être lu ou écrit.

Gestion des interruptions



Références

- Texas Instruments, "**AM/DM37x Multimedia Device**", Technical Reference Manual
- Sajal K. Das, "**Mobile Handset Design**", Wiley