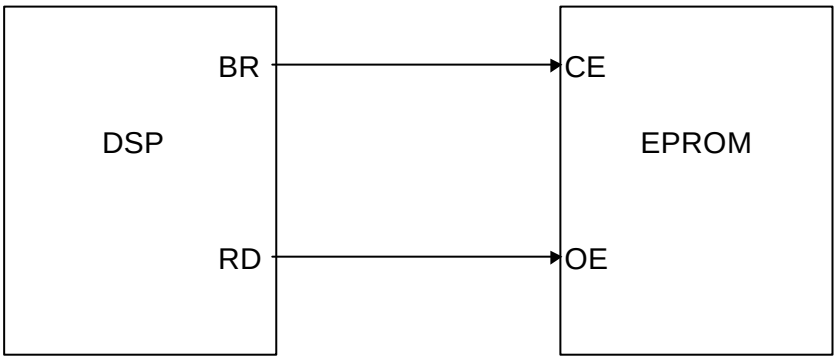
Seqüència de Boot del DSP TMS320C5x:

Boot en mode MC/MP=’0’.

Amb aquesta opció, s’habilita la ROM interna del DSP configurada amb un programa anomenat *Boot Loader* que realitza les següents operacions: Inicialitza el DSP i el configura amb 7 *Wait States* per assegurar que la velocitat dels perifèrics externs com per exemple una EPROM siguin el suficientment ràpids. Reserva un espai de memòria global desde la posició 8000h fins la FFFFh. Llegeix la posició FFFF on s’hi ha de col·locar el BRSW (Boot Request Selection Word). Aquesta paraula li indica al DSP com ha de botar i quina és l’adreça inici de la localització del programa (per default és la 8000h, coincidint amb l’espai reservat). Si dins el BRSW hi gravem el 81h, llavors li indiquem que botarà a partir d’una EPROM externa i el programa comença a la 8000h. El DSP controla la lectura amb el pin BR (Bus Request) pel CE de la EPROM i el pin RD pel OE de la EPROM com es mostra a la següent figura:



COMUNICACIÓ DEL DSP AMB L’EXTERIOR:

El DSP té 64K de I/O i té 16 ports mapejats a memòria (de la 50 a la 5F).
Fent una assignació a alguna d’aquestes adreces el DSP genera un pols a ‘0’ del pin IS (I/O select), i col·loca en la posició adequada els pins r/w, we, rd i strb.

CONNEXIÓ DEL DSP AMB LA FPGA:

La tarja MAGCL està proveïda amb un DSP TMS320C51, i per tant, segueix la rutina de boot anterior.

Els pins que s’han d’utilitzar per fer una comunicació simple són: DADES, ADRECES, rd, we, r/w i is.

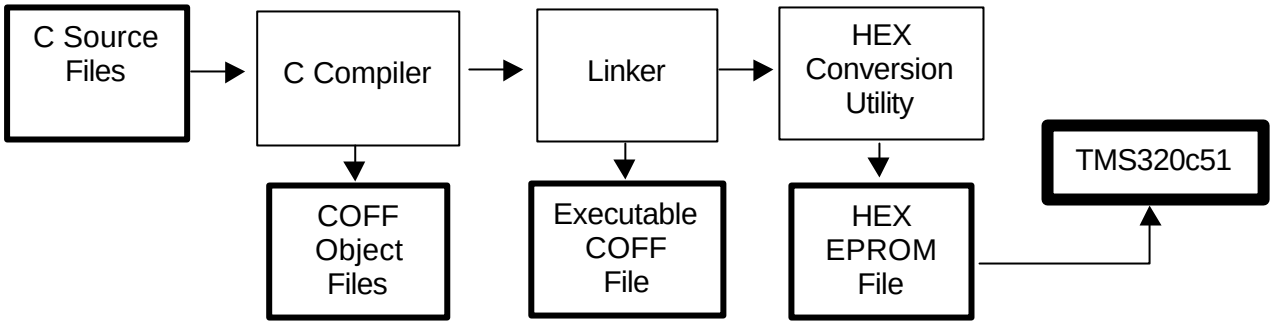
Les Irq’s s’han de deixar a ‘1’.

Els senyals nmi, ready i bio també a ‘1’.

Els altres a ‘Z’.

COMPILACIÓ, LINKATGE I GENERACIÓ DEL FITXER .HEX D'UN PROGRAMA REALITZAT AMB C

El procés que s'ha seguit per arribar a executar un programa dins el DSP ha estat el següent:



A continuació comentarem cada una de les fases amb més detall.

1) *Edició del programa font*

Amb un editor qualssevol s'escriu el programa amb C que volem executar. Per accedir més fàcilment als registres del DSP es disposa d'un fitxer .h (c5xregs.h) on hi ha tots els registres mapejats a memòria i que fins i tot ens permet accedir-hi a nivell de bit.

Per exemple, el programa utilitzat per realitzar les proves ha estat el següent:

```
#include "c5xregs.h"

void main(void)
{
      volatile unsigned int temp0,temp1,temp2;

      /*
      Lectura i escriptura de dades.
      */
       for (;;) {
            temp0 = *PA0;
            temp1 = *PA1;
            /*    tractament de dades;  */
            *PA2 = temp2;
      }
}
```

2) *Compilació del fitxer .c*

Per compilar el fitxer font es disposa de l'utilitat:

dspcl [-options] –g [filenames] [-z [link options]]

A part de compilar fitxers .c, també es poden afegir fitxers .asm que es compilaran sense cap problema. En el nostre cas la comanda sencera ha estat:

dspcl -v51 -g port.c boot.asm -z linkmeu.cmd

L'opció,

- v51: genera un fitxer COFF objecte pel DSP TMS320c51.
- g: genera la taula d'etiquetes del programa pel 'debugger'.
- z: comentada en el següent apartat.

Fitxers,

Port.c: Fitxer amb el programa C font.
Boot.asm: Mòdul que conté les definicions,
__stack - Stack memory area
_c_int0 - Boot function
_var_init - Function which processes initialization tables

3) Linkatge

Un cop compilat el programa s'obtenen uns fitxers .obj que haurem de linkar per obtenir el fitxer COFF executable final.
Es pot fer de dues maneres diferents:

a) utilitzant la comanda,
dsplnk [- options] filenames
b) utilitzant l'opció **–z** dins de la comanda **dspcl**

Nosaltres hem optat per aquesta segona opció. L'opció **–z** va acompanyada per un "Linker command file" on s'especifiquen totes les opcions de linkatge que volem, la configuració del mapa de memòria, les seccions, etc.
Aquest és el fitxer .cmd que hem utilitzat:

```
-c
-l  rts50.lib
-m  port.map
-o  port.out

MEMORY
{
    PAGE 0: PROG    :   origin = 0x2000, length = 0x1000
    PAGE 0: BRSW    :   origin = 0x3fff, length = 0x0001
    PAGE 1: DATA    :   origin = 0x0800, length = 0x1000
}

SECTIONS
{
    boot_sec: {     *(.text)

                cinit = .;  /* Set start address for C init table */

                *(.cinit)   /*include all cinit sections          */

                .+=1;       /*************************************/
                            /* Reserve a single space for the zero */
                            /* word to mark end of C init.         */
                            /*************************************/

        }       fill = 0x0000,      /* Make sure fill value is 0 */
                load = PROG PAGE 0

    brsw:  {        /*************************************/
```

```
           .+=1;                    /* Reserve a word in this section.*/

                    /*************************************/
                    /* Select fill value that corresponds  */
                    /* to correct boot routine select mode */
                    /*************************************/
        }       fill = 0x81,
                load = 0x3fff PAGE 0

    .data   : {} > DATA PAGE 1
    .bss    : {} > DATA PAGE 1
    .const  : {} > DATA PAGE 1
    .sysmem : {} > DATA PAGE 1
    .stack  : {} > DATA PAGE 1
}
```

L'opció,

- c: utilitzem el model d'autoinicialització ROM del TMS320c2x/c2xx/c5x
- l: llibreria pel DSP TMS320c50 (vàlida també pel c51)
- -m: genera un llistat (mapa) de les seccions tant d'entrada com de sortida, incloent forats, del linker.
- -o: indica el nom del fitxer executable de sortida. (per defecte a.out)

Tot seguit, darrera les opcions de linkatge, s'indica quin és la configuració del mapa de memòria que utilitzem. Podem veure que per la secció de programa (PROG) utilitzem l'adreça 2000h com a origen i té una longitud de 1000h. De la mateixa manera s'indica que la secció de dades (DATA) comença a l'adreça 800h i té una longitud de 1000h. Aquest valors depenen del tipus de DSP i del mode de funcionament que s'utilitza. Podeu consultar aquestes opcions al "*User's Guide*" del TMS320c5x (pp. 6-3 a 6-4).
Finalment, observem que tenim una secció anomenada BRSW i que anteriorment hem comentat com a *Boot Routine Selection Word*, que ens permet escollir quin mode de boot utilitzem per el nostre DSP. Depenent del valor que col·loquem a l'adreça que indica aquesta secció el boot es farà des d'una EPROM (el nostre cas), a través del port paral·lel o via port sèrie.

Una cosa a tenir amb compte quan realitzem aquest fitxer.cmd és la següent. El *on-chip boot loader* carrega un únic bloc. Això presenta varis problemes quan volem carregar un codi compilat en C ja que el compilador crea varies seccions o blocs. Algunes aplicacions necessiten que totes les seccions es trobin en el boot per tenir programes completament executables. Per tant, caldrà realitzar alguna modificació en el fitxer .cmd. Aquests canvis són els que s'han fet amb la creació de la nova secció boot_sec que agrupa aquelles seccions necessàries en el boot (en la nostra aplicació, les seccions .text i .cinit).

4) Generació del fitxer .HEX

Un cop generat el fitxer COFF executable final, el següent pas és generar un fitxer que el programador d'EPROM pugui reconèixer i carregar correctament.
Això ho aconseguim amb l'utilitat,

dsphex [-*options*] filename

De la mateixa manera que havíem fet anteriorment, utilitzem un fitxer .cmd on s'indiquen totes les opcions necessàries per generar el fitxer .HEX.

```
/*****************************************************************/
/* DSPHEX command file used to convert the communications kernal to   */
/* an EPROM programmer file in Intel format.                          */
/*****************************************************************/

     port.out        /* Specify COFF file to convert.           */
-i                   /* Intel format.                           */
-map  port.mxp       /* Map file                                */
-o    port.hex

-memwidth 8          /* Set memwidth and romwidth = 8 to create */
-romwidth 8          /* an 8-bit wide eprom programmer file.    */

-boot
-bootorg 0x0000      /* Define address where boot table will    */
/* be located.                          */
ROMS
{
PAGE 0: ROM : origin = 0x0000, length = 0x8000
}

SECTIONS
{
```

L'opció,

-i: Selecciona el format de sortida INTEL.
-map: Genera un map file.
-o: Especifica un nom de sortida.

-memwidth: Defineix l'amplada d'una paraula en el sistema de memòria.
-romwidth: Defineix l'amplada de la memòria ROM.

I les opcions per el boot-loader:

-bootorg: Especifica l'adreça font del *boot loader table*.

Finalment, s'indica quines i on s'han de carregar algunes de les seccions del programa.
En el nostre exemple, havíem definit en el *Linker command file* una secció anomenada boot_sec que és la que col·loquem en el boot i una secció anomenada .brsw que col·loquem a l'adreça 0x7ffe. El perquè d'aquesta posició és degut a l'amplada del sistema de memòria (8 bits) i a l'amplada de les dades (16 bits), que provoca que l'espai adreçable s'expandeixi en un factor de 2. Així doncs, tenim que 0x3fff (adreça origen on hem col·locat la BRSW en el *Linker Command File*) * 2 = 0x7ffe.