Linux et le Système sur Silicium

Patrice Kadionik, Maître de Conférences à l'ENSEIRB (kadionik@enseirb.fr)

Février 2005

Mots Clés

Système embarqué - Linux embarqué - Système sur Silicium (SoC et SoPC) – Codesign – Bloc IP (Intellectual Property)

Introduction

Cet article a pour objectif de présenter comment sont conçus actuellement les circuits numériques complexes. La méthodologie de conception appelée *codesign* est présentée et autorise un développement conjoint du matériel (*hardware*) et du logiciel (*software*). Pour la partie logicielle, Linux pour l'embarqué est de plus en plus utilisé et a déjà fait l'objet de diverses présentations dans Linux Magazine. La mise en oeuvre de Linux embarqué au travers de µClinux est décrite dans le cadre de l'offre de codesign Quartus II d'Altera pour le *System on Programmable Chip* (SoPC). Différents exemples couplant µClinux à un système SoPC seront donnés pour montrer l'intérêt du codesign lors de la conception de systèmes numériques complexes.

La conception des systèmes numériques complexes

Les systèmes numériques deviennent aujourd'hui de plus en plus complexes au niveau intégration et fonctionnalités et l'on est en mesure d'intégrer tout dans un même composant : c'est le concept du *single chip*. Ceci est en fait lié à la loi empirique de Moore qui stipule que pour une surface de silicium donnée, on double le nombre de transistors intégrés tous les 18 mois ! La figure 1 montre cette évolution.

	1998	1999	2001	2002	2004
Technologie	0,25 µm	0,18 µm	0,15 µm	0,13 µm	0,09 µm
Complexité en Millions de transistors	1 M	2-5 M	5-10 M	10-25 M	> 25 M
Loi de Moore					

Figure 1 : Loi de Moore

La loi de Moore a radicalement changé la façon de concevoir les systèmes numériques. On travaille maintenant au niveau système (ou fonctionnalité) et non au niveau porte logique (pour le grand bien des électroniciens).

Les fonctionnalités peuvent être implantées dans des composants spécifiques de type ASIC (*Application Specific intregrated Circuit*). On parle alors de Système sur Silicium SoC (*System on Chip*).

Les fonctionnalités peuvent être implantées dans des composants logiques programmables de type FPGA (*Field Programmable Gate Array*). On parle alors de système SoPC (*System on Programmable Chip*).

Cette évolution de la conception peut être résumée sur la figure 2 :

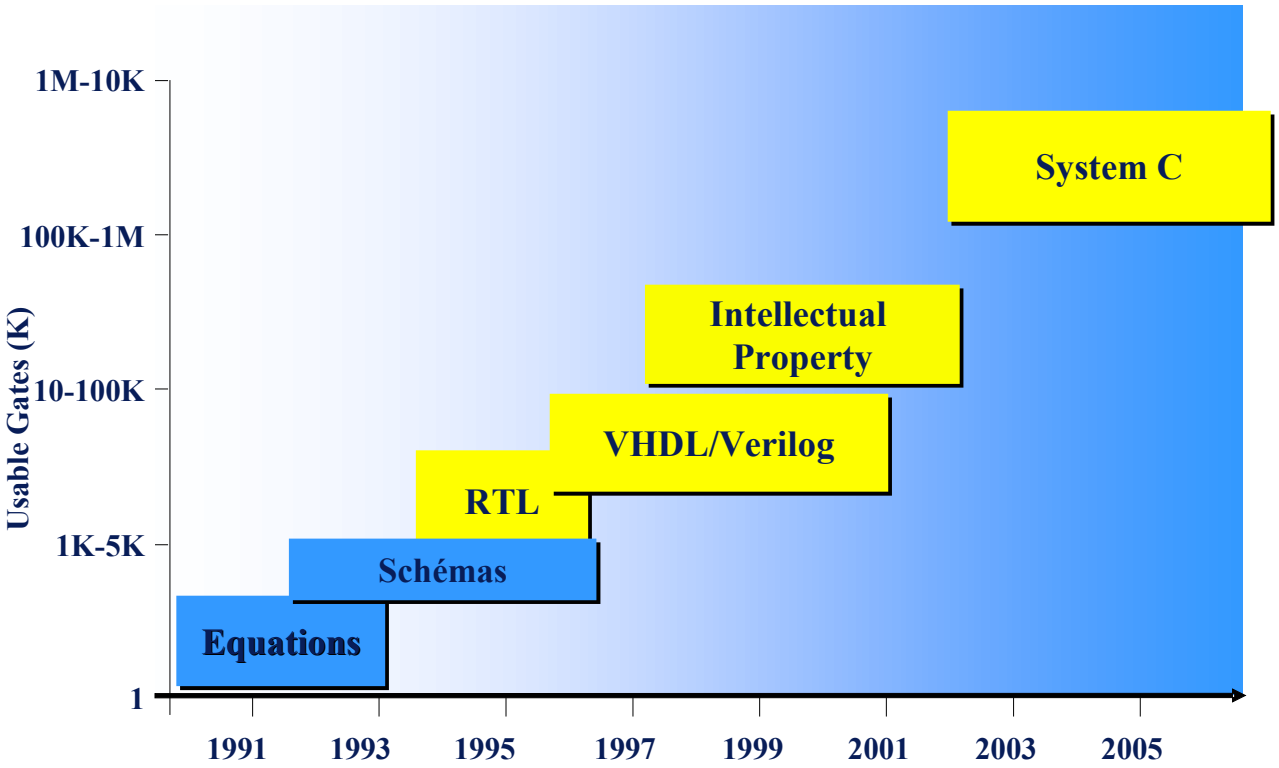

Figure 2 : Evolution de la conception numérique

L'approche « schématique » au niveau portes logiques ou fonctionnalités de base RTL (*Register Transfer Logic*) est délaissée pour la conception des systèmes complexes au profit d'une approche « textuelle » mais reste bien sûr toujours valable pour la conception des systèmes numériques plus modestes.

Pour cela, on utilise des langages de description de matériel comme VHDL (*Very high speed integrated circuit Hardware Description Language*) ou Verilog pour synthétiser une fonctionnalité numérique. Ces langages de description de matériel sont en fait de véritables langages de programmation informatiques, orientés objet et peuvent être utilisés comme tels par les informaticiens. Ils sont utilisés conjointement avec un compilateur ou un simulateur .

Il est d'ailleurs faux de dire compilateur mais plutôt synthétiseur car on génère au final dans la majorité des cas (conception SoPC) un fichier de programmation d'un composant logique programmable ! On synthétise un circuit numérique mais on ne le compile pas, bien que l'on ait un approche logicielle pour concevoir du matériel !

Ces langages ont permis de travailler avec un niveau d'abstraction plus grand laissant les basses besognes au synthétiseur. On a pu rapidement développer des bibliothèques de fonctionnalités comme une interface USB, un contrôleur MAC Ethernet que l'on appelle blocs IP (*Intellectual Property*). Ces blocs IP peuvent être un peu comparés à un composant logiciel comme le *Java Bean*. On peut les acheter ou bien utiliser des blocs IP libres (comme du logiciel libre) dont le site phare de référence est http://www.opencores.org.

On peut ainsi voir la conception d'un système numérique complexe comme un assemblage de blocs IP si bien que les langages de description de matériel sont un peu comme un langage assembleur vis à vis d'un langage plus évolué comme le langage C.

Il est à noter que le nirvana serait de pouvoir générer un circuit numérique à partir d'un fichier source écrit en langage informatique comme le langage C : c'est ce que propose SystemC mais bien des progrès restent encore à faire dans ce domaine...

Les langages de description de matériel sont aussi intéressants pour la facilité de modification et de réutilisation d'un design précédent pour un nouveau design : c'est le *design reuse*. De plus, cela permet de réduire aussi le *Time To Market* (TTM) cher aux gestionnaires !

Il faut noter que lorsque l'on conçoit un système numérique complexe, on met un oeuvre généralement un processeur embarqué. Ce processeur embarqué est :

- Soit un bloc IP : on parle de processeur *softcore*.
- Soit déjà implanté dans le circuit électronique en « dur » : on parle de processeur *hardcore*. Le processeur de ce type est généralement plus performant que le processeur du type précédent.

Le processeur embarqué allie la souplesse du logiciel à l'accélération du temps d'exécution du matériel. Une fonctionnalité particulière peut donc être composée d'une partie matérielle couplée à une fonctionnalité logicielle dédiée : on a donc une conception conjointe matérielle-logicielle ou *codesign*. Le codesign implique donc une conception en même temps du matériel et du logiciel, ce qui est une nouvelle méthodologie par rapport à la méthodologie de conception classique (conception matérielle puis conception logicielle)...

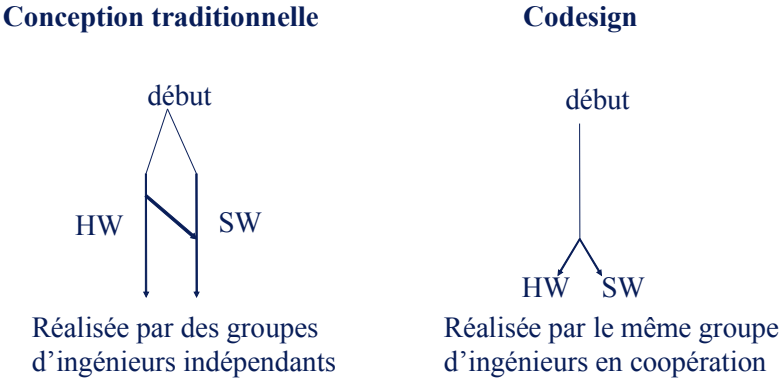

Figure 3 : Conception traditionnelle et codesign

Il faut noter que les processeurs embarqués comme leurs cousins du grand public sont de plus en plus puissants et la puissance CPU en MIPS double les 2 ans (loi empirique de Joy) et sont de plus communicants et l'on estime que l'on a besoin d'une bande passante réseau de 0,3 à 1 Mb/s par MIPS (loi empirique de Ruge). La course à la performance gagne aussi le monde de l'embarqué et des systèmes numériques complexes !

Enfin, qu'en est-il des systèmes mixtes ou purement analogiques ? Des langages de description du matériel apparaissent comme par exemple VHDL-AMS mais restent confinés à la simulation. Il est à noter qu'il commence à y avoir des composants analogiques programmables sur le marché. On peut par exemple citer les pSoC de Cypress qui intègrent dans un même circuit un microcontrôleur, une zone de logique programmable et une zone analogique. La zone analogique comporte des blocs

contenant des composants analogiques élémentaires (capacités, capacités commutées, résistances, amplificateurs...) pour construire des fonctionnalités analogiques ou mixtes...

Les processeurs pour le SoPC

Le choix d'un processeur pour le SoPC peut se faire sur différents critères :

- Processeur hardcore : pour ses performances au détriment de la flexibilité.
- Processeur softcore : pour sa flexibilité de mise à jour au détriment de performances moindres que le précédent. La portabilité vers n'importe quel circuit FPGA est assurée en étant donc circuit FPGA indépendant. Il est aussi possible de migrer vers un circuit de type ASIC en cas d'une production en grande série.

Généralement, on privilégie les processeurs softcore pour s'affranchir des problèmes d'obsolescence et pour pouvoir bénéficier facilement des évolutions apportées en refaisant une synthèse.

Le processeur softcore peut être propriétaire : il est distribué par exemple sous forme d'une *netlist* pour être implantée dans un circuit FPGA. Il est généralement lié à un fondeur de circuit FPGA particulier (comme Altera ou Xilinx). On ne peut pas l'utiliser dans un circuit FPGA autre que celui pour lequel il est prévu. On a donc ici une boîte noire.

Le processeur softcore peut être libre : il est décrit en langage de description de matériel (VHDL, Verilog) dont le code source peut être librement distribué et implanté dans n'importe quel circuit programmable FPGA. On est alors indépendant du type de circuit FPGA.

On trouvera principalement au niveau des processeurs softcores propriétaires :

- Le processeur NIOS et NIOS II d'Altera http://www.altera.com
- Le processeur Microblaze de Xilinx http://www.xilinx.com.
- ...

On trouvera principalement au niveau des processeurs softcores libres :

- Le processeur Leon http://www.gaisler.com/index.html.
- Le processeur OpenRisc http://www.opencores.org/projects.cgi/web/or1k/overview.
- Le processeur F-CPU http://www.f-cpu.org.
- Autres processeurs : clones de 6800, 68HC11, 68K, PIC : http://www.opencores.org/browse.cgi/filter/category_microprocessor
- ...

Ce sont généralement des processeurs 32 bits ayant une architecture de type Harvard avec un jeu d'instructions réduit RISC. Pour plus d'informations, le lecteur pourra se référer à la bibliographie.

Conception d'un système SoPC : le choix d'un circuit FPGA

Quand on désire concevoir un système SoPC complexe, on doit mettre en oeuvre un circuit programmable FPGA. On a alors le choix entre les 2 leaders du marché :

- Les circuits FPGA d'Altera : http://www.altera.com.
- Les circuits FPGA de Xilinx : http://www.xilinx.com.

Le choix se fait alors en fonction de ses besoins, ses habitudes (un étudiant qui a développé sur un circuit Altera aura tendance à les utiliser par la suite dans sa carrière professionnelle) et les outils associés.

Il est clair que le circuit FPGA en lui-même est sans intérêt car l'on a besoin des outils logiciels associés pour pouvoir le mettre en oeuvre. Les 2 fournisseurs proposent chacun leur suite d'outils propriétaires pour la mise en oeuvre de leurs circuits FPGA et le concepteur en est malheureusement tributaire car il n'existe pratiquement rien dans le monde du logiciel libre... Il est à noter que ces outils sont généralement portés sous Windows.

On peut d'ailleurs noter que les outils de conception sont aussi importants voire plus que le composant lui-même, ce qui explique le succès d'Altera dans ce domaine qui a su très tôt proposer une suite d'outils professionnels, simples et utilisables par les PME.

L'ENSEIRB utilise indifféremment les circuits des 2 fournisseurs sachant que dans le cadre de son enseignement des SoPC, c'est l'offre Altera qui a été choisie via le programme universitaire Altera.

Conception d'un système SoPC : l'offre SoPC d'Altera

L'offre SoPC d'Altera s'appelle Quartus II. C'est une offre commerciale de codesign mettant en oeuvre un processeur hardcore ou softcore.
L'offre softcore est la plus populaire et s'appelle NIOS. Une deuxième version plus performante de NIOS est aussi disponible : NIOS II.
Une version gratuite (*Quartus II Web Edition*) mais limitée à un nombre limité de circuits FPGA est disponible au téléchargement (https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp).
Quartus II s'utilise sous Windows mais aussi sous une distribution Linux RedHat 8.0 ou RedHat Enterprise 3.0.

Quartus II permet une conception d'un circuit numérique à partir d'une entrée sous forme :

- D'une schématique traditionnelle.
- D'une netlist standard de type EDIF (*Electronic Design Interchange Format*).
- Textuelle à l'aide d'un langage de description de matériel : VHDL, Verilog et A-HDL (*Altera HDL*).

Le développement est alors classique et résumé sur la figure 4 :

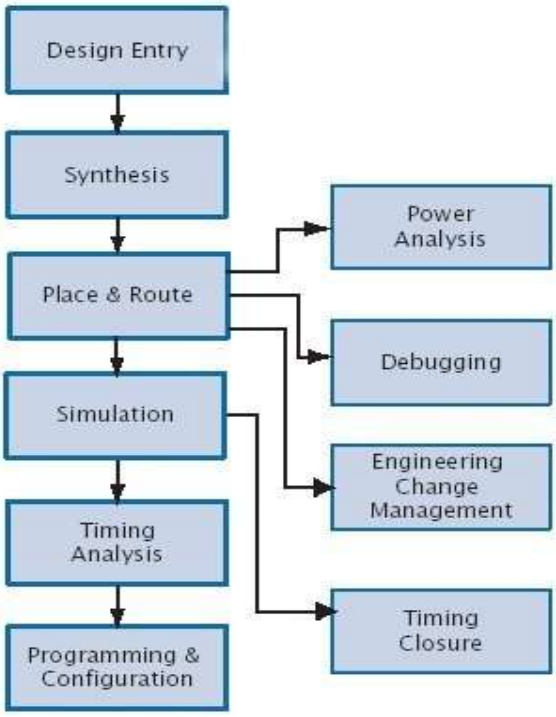

Figure 4 : Flot de conception Quartus II

Après synthèse et routage pour le circuit FPGA ciblé, il est possible de faire un certain nombre de vérifications :

- Simulation post synthèse.
- Etude de timing.
- Etude de la consommation.

et revenir au design si quelque chose ne va pas avant de programmer le circuit FPGA.

La figure 5 présente l'interface graphique de Quartus II :

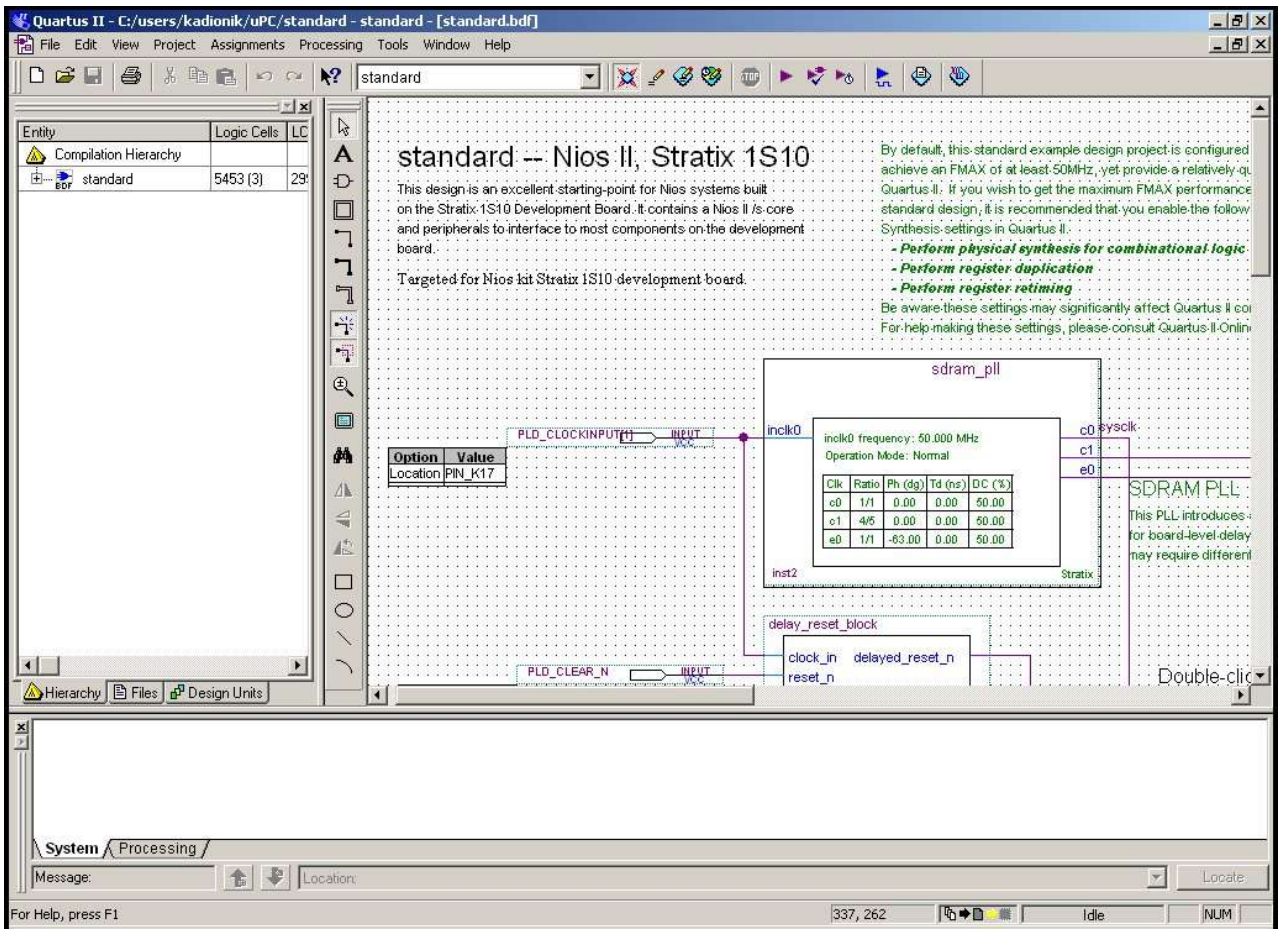

Figure 5 : IDE Quartus II

L'IDE (*Integrated Design Entry*) Quartus II intègre l'outil *SoPC Builder* qui permet de construire un système SoPC. Il permet graphiquement de contruire un microcontrôleur intégrant des périphériques d'E/S divers et variés :

- Processeur NIOS II.
- Contrôleur de SRAM, SDRAM.
- Contrôleur DMA.
- Contrôleur de mémoire CompactFlash.
- Timer.
- Boutons, afficheurs LCD, afficheurs 7 segments.
- Liaison série UART.
- Interface Ethernet.
- ...

Il est possible d'intégrer ses propres périphériques d'E/S à ce moment-là dans le design sous forme d'un bloc IP externe :

- Interface clavier.
- Interface souris.

- Interface VGA.
- ...

On peut ainsi intégrer autant de périphériques que l'on veut, n'étant limité que par le nombre de broches et de cellules logiques du circuit FPGA ciblé.

Le mapping mémoire et le niveau des interruptions sont fixés durant cette phase. La figure 6 montre la mise en oeuvre de l'outil *SoPC Builder*. C'est en fait la première passserelle avec le logiciel embarqué :

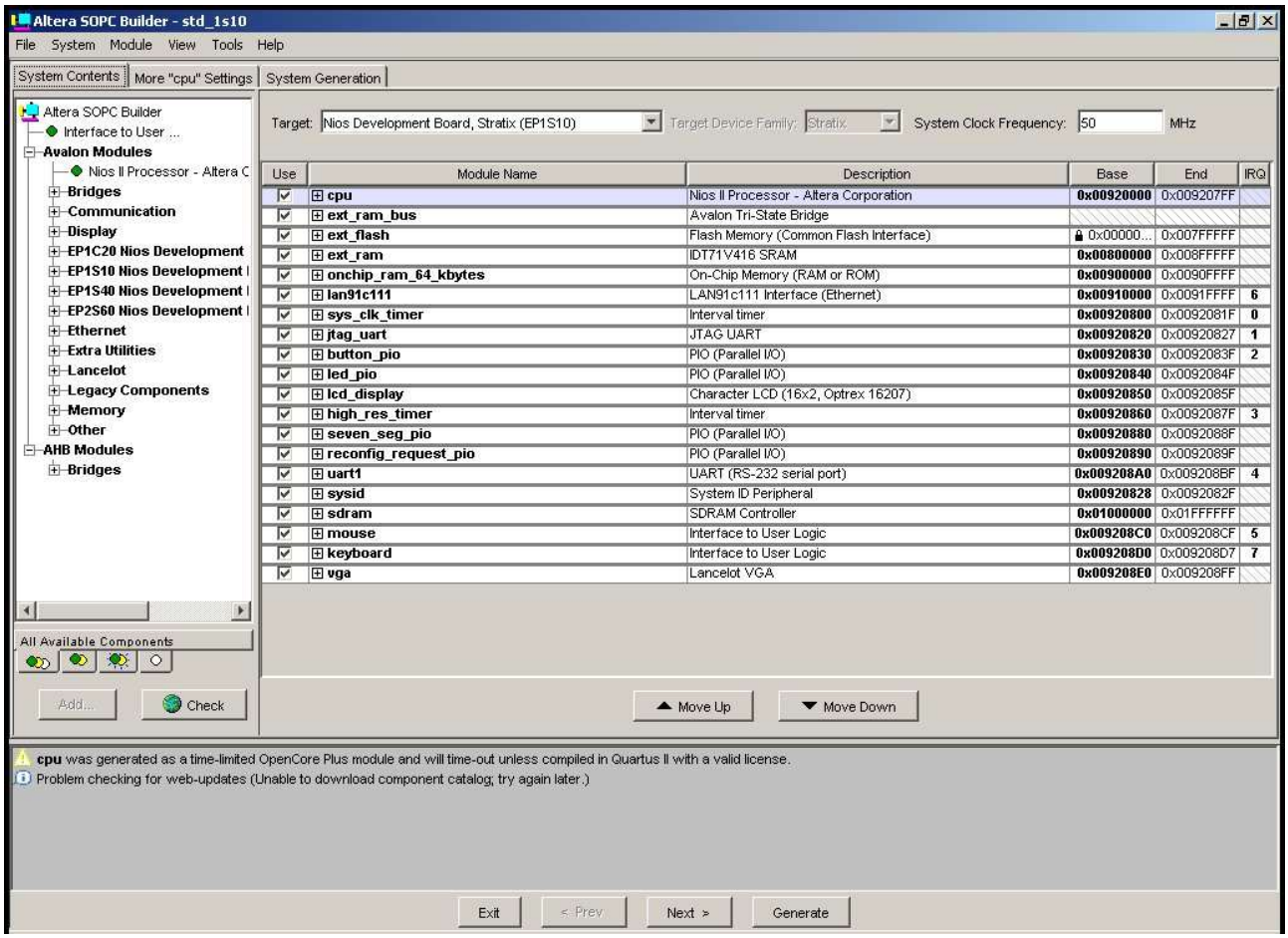

Figure 6 : SoPC Builder et mapping mémoire

A l'issue de la phase de construction du système SoPC, Quartus II génère le projet en conséquence en intégrant tous les modules IP (VHDL ou Verilog). Après synthèse, on a alors le fichier de programmation du circuit FPGA correspondant au design SoPC mais aussi un kit de développement logiciel *cpu_sdk* qui comprend tous les fichiers en langage C .h et .c pour piloter les périphériques d'E/S d'Altera par une application de base ! C'est en fait la deuxième passerelle avec le logiciel embarqué.

L'offre de codesign apparaît au grand jour ici avec la possibilité de développer une partie de l'application par matériel ou de le faire en langage C par logiciel...

La carte cible : carte Altera Stratix

Pour pouvoir réaliser un système SoPC, on a bien sûr besoin de matériels spécifiques... Altera propose des cartes de développement pour mettre en oeuvre notamment son offre de codesign. Ces cartes intègrent toutes un circuit FPGA plus ou moins gros, associé à des périphériques externes (JTAG , SDRAM, interface Ethernet, port série...). La carte choisie ici est une carte moyenne gamme Stratix 1S10 dont la figure 7 donne une vue d'ensemble :

Figure 7 : Carte Altera Stratix 1S10

La carte Stratix 1S10 comprend les fonctionnalités suivantes :

- Circuit FPGS Stratix II EP2S30F672C5.
- 1 Mo de SRAM 16 bits.
- 16 Mo de SDRAM 32 bits.
- 16 Mo de mémoire Flash.
- 1 support CompactFlash type I avec une mémoire CompactFlash de 16 Mo.
- 1 interface Ethernet 10/100 Mb/s.
- 2 ports série (RS-232 DB9).
- 2 ports d'extension pour carte fille maison.
- 1 connecteur JTAG.
- 4 boutons poussoirs.
- 8 leds utilisateurs.
- 2 afficheurs 7 segments.
- 1 afficheur LCD 2x16.
- ...

La carte est proposée pour moins de 1000 USD au catalogue Altera.
Une carte d'évaluation bon marché (*Nios II Evaluation Kit, (*http://www.altera.com/products/devkits/kit-dev_platforms.jsp) est disponible pour un prix plus modeste de 295 USD au catalogue (revendeurs en France : Arrow, Tekelec, EBV...).

Développement d'une application embarquée dans le silicium

Le développement d'une application embarquée dans le système SoPC est alors possible en langage C avec les outils précédents.

Malheureusement, du point de vue logiciel, on ne dépasse pas le stade de la superboucle ; c'est à dire la boucle infinie en concurrence avec les interruptions générées par le système cible.

Il est néanmoins possible de mettre en oeuvre un noyau Temps Réel dans le cas du processeur softcore NIOS :

- Noyau Temps Réel propriétaire microC/OS II (µC/OS II) de la société Micriµm http://www.ucos-ii.com/. Voir la page de l'auteur : http://www.enseirb.fr/~kadionik/embedded/uclinux/nios-uclinux.html#RTOS.
  Il est à noter que microC/OS II est libre de droits dans le cas d'un usage universitaire...
- Noyau Temps Réel propriétaire KROS : http://www.krostech.com/

Dans le cas du processeur de deuxième génération softcore NIOS II, microC/OS II est maintenant directement fourni par Altera à travers son IDE de développement du logiciel.

L'usage d'un noyau Temps Réel avec le processeur NIOS/NIOS II améliore les choses du point de vue de la conception du logiciel (aspect multitâche du noyau) surtout si l'on a en plus des contraintes temporelles dures à respecter. Mais on est encore loin de l'usage de son système d'exploitation Linux préféré...

Linux embarqué dans le silicium !

La société Microtronix (http://www.microtronix.com) a réalisé il y a quelques années le portage de µClinux pour le processeur softcore de première génération NIOS. Ce portage est payant bien que basé sur des projets GPL, non disponile au libre téléchargement et n'a pas été complètement intégré dans le projet µClinux (branche *nios-nommu* du projet µClinux) http://www.uclinux.org.

µClinux est une version modifiée de Linux pour processeur sans MMU courant dans l'embarqué et a déjà été présenté dans Linux Magazine (Linux Magazine, février 2002). µClinux est officiellement reconnu car il est maintenant incorporé dans les sources du noyau standard Linux 2.6 !

Le portage de µClinux pour le processeur NIOS est donc incomplet et l'on pourra trouver des informations sur la compilation du noyau à la page http://www.enseirb.fr/~kadionik/embedded/uclinux/nios-uclinux.html.

On pourra aussi récupérer le portage NIOS à partir de la page du projet CDK4NIOS : http://sourceforge.net/projects/cdk4nios/.

Le portage de µClinux pour le processeur NIOS II est complet sous licence GPL et a encore cette fois été réalisé par Microtronix. Il est disponible au libre téléchargement ici : http://www.niosforum.com/forum/.

µClinux pour NIOS II est une adaptation du noyau Linux 2.6 pour le processeur NIOS II.

Le forum niosforum est LE forum sur la mise en oeuvre des processeurs NIOS et NIOS II et sur le portage µClinux (et microC/OS II).

La mise en oeuvre de µClinux sur le processeur NIOS II nécessite donc :

- L'installation de Quartus II (sous Windows) pour la synthèse matérielle.
- Une carte cible (carte d'évaluation comprenant un circuit FPGA Altera).
- L'installation du portage de µClinux pour le processeur NIOS II.
- Les outils de développement logiciel.

Les outils de développement logiciel sont en fait installés lorsque l'on installe Quartus II.

On a en fait à disposition :

- Un atelier de développement logiciel intégré qui est l'IDE libre Eclipse (http://www.eclipse.org) avec des plugins Altera pour s'interfacer à Quartus II. Eclipse permet de créer un projet µClinux (système de fichiers root et noyau µClinux), de cross-compiler une application µClinux (ou bien de créer un projet microC/OS II).
- Un environnement UNIX sous Windows via le projet Cygwin (http://www.cygwin.com/) qui permet de dialoguer avec la carte cible via la liaison série ou par le port JTAG ou de debugger une application...

Etape 1 : design de référence

Pour le développement codesign NIOS II/µClinux, il convient de définir le design de référence qui servira après synthèse à programmer le composant FPGA de la carte cible, ici la carte Altera Stratix 1S10. Le design de référence choisi est l'exemple Altera *standard* pour la carte 1S10.

On a alors après programmation du circuit FPGA Stratix connecté à l'ensemble des périphériques de la carte cible :

- Processeur Softcore Nios II.
- JTAG.
- 64 Ko de RAM on-chip RAM.
- 1Mo de RAM externe.
- 8 Mo de mémoire FLASH.
- Interface CompactFlash.
- 16 Mo SDRAM.
- 1 UART.
- 2 timers 16 bits.
- 1 interface Ethernet.
- Leds, afficheurs 7 segments, bouttons poussoirs, afficheurs LCD de la carte cible.

Le mapping mémoire de l'espace d'adressage de processeur NIOS II embarqué est celui de la figure 6.

Etape 2 : génération du noyau µClinux

La deuxième étape est la génération du noyau µClinux pour le processeur NIOS II.

Il convient pour cela d'utiliser l'IDE Eclipse. On accède alors à un menu de configuration équivalent à un « *make menuconfig* » pour configurer le noyau avant compilation.

La figure 8 présente la génération du noyau µClinux sous Eclipse.

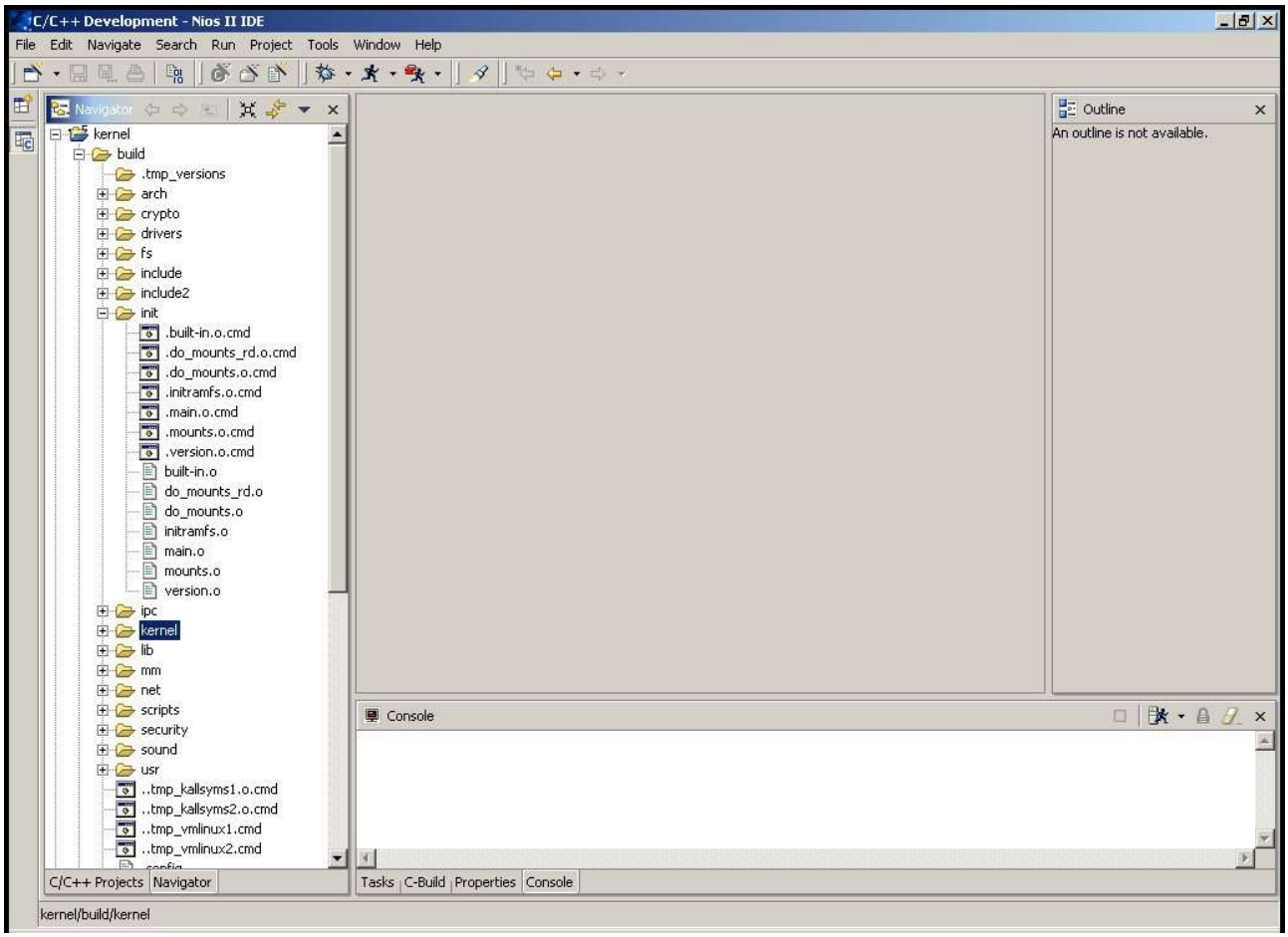

Figure 8 : Génération du noyau µClinux

Etape 3 : construction du système de fichiers root

La troisième étape est la construction du système de fichiers root du noyau µClinux.

Il convient pour cela d'utiliser l'IDE Eclipse.

Un système de fichiers root par défaut est disponible (grâce aux plugins Eclipse d'Altera) qui contient un ensemble de commandes de base Linux. C'est dans ce système de fichiers qu'il convient d'incorporer ses propres applicatifs. La figure 9 présente la construction de ce système de fichiers sous Eclipse.

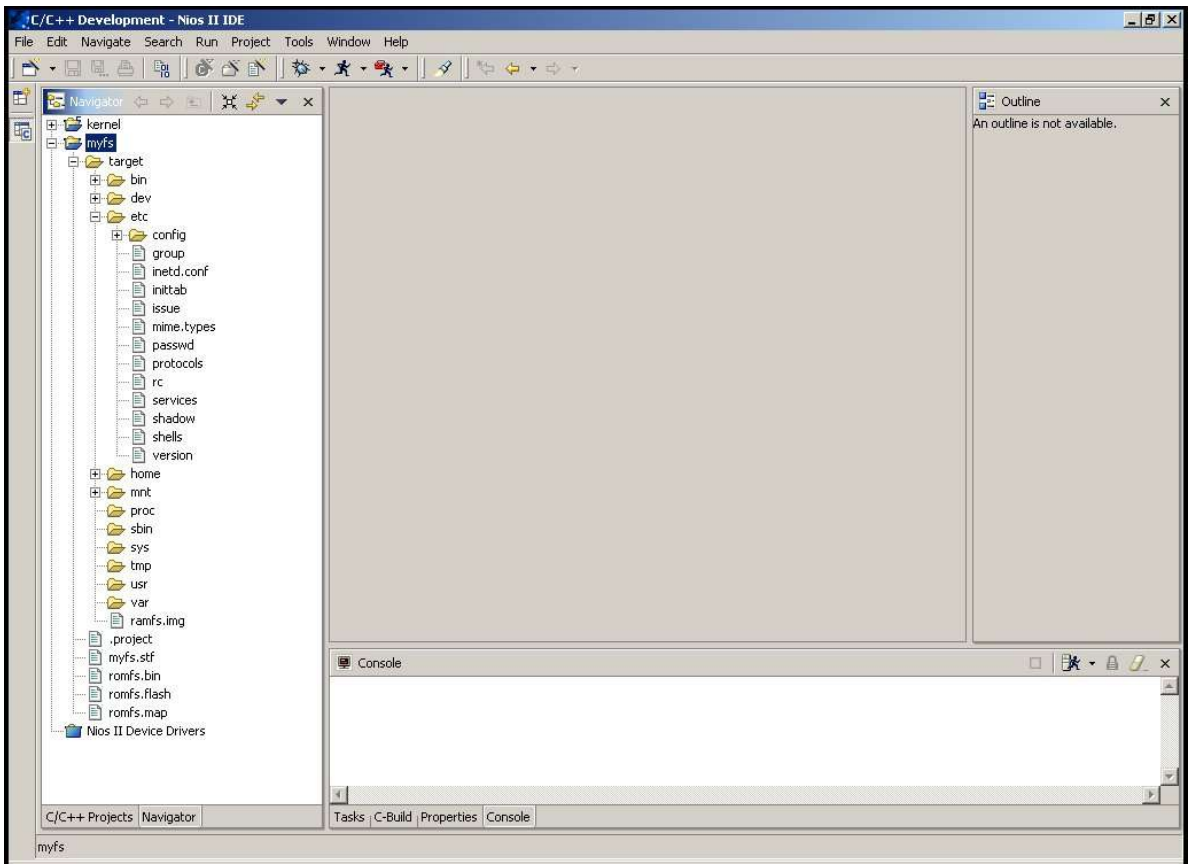

Figure 9 : Construction du système de fichiers root µClinux

Etape 4 : téléchargement et boot

La dernière étape est le téléchargement de l'image binaire contenant le noyau µClinux et le système de fichiers root dans la mémoire FLASH de la carte cible via l'interface JTAG puis la programmation du circuit FPGA (figure 10). Cela se fait en utilisation la fonction programmateur de Quartus II.

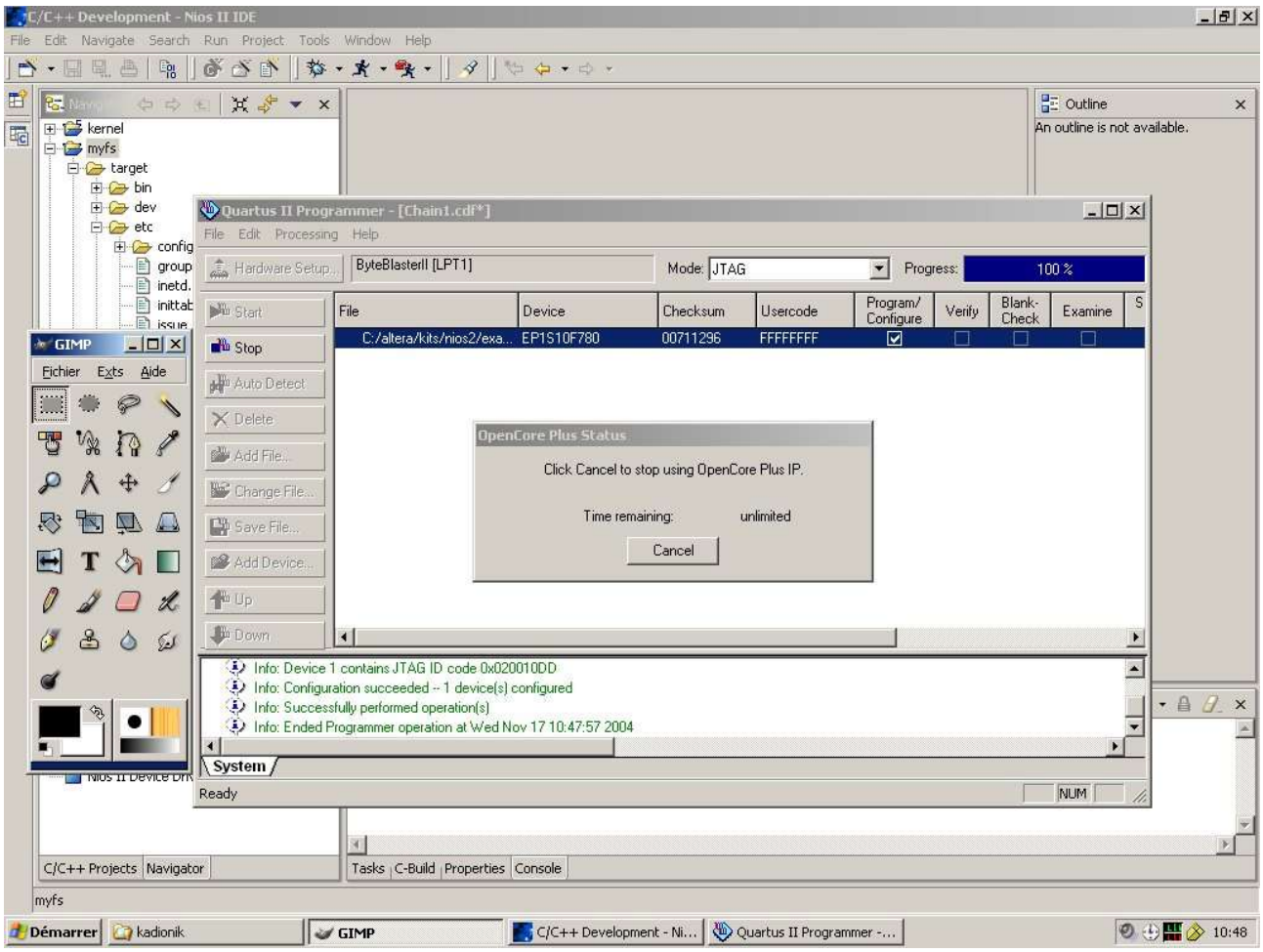

Figure 10 : Programmation du circuit FPGA

Après programmation du circuit FPGA, le processeur NIOS II boote directement sur le programme mis en mémoire FLASH ; c'est à dire ici le noyau µClinux. La figure 11 donne les traces de boot du noyau µClinux sur NIOS II.

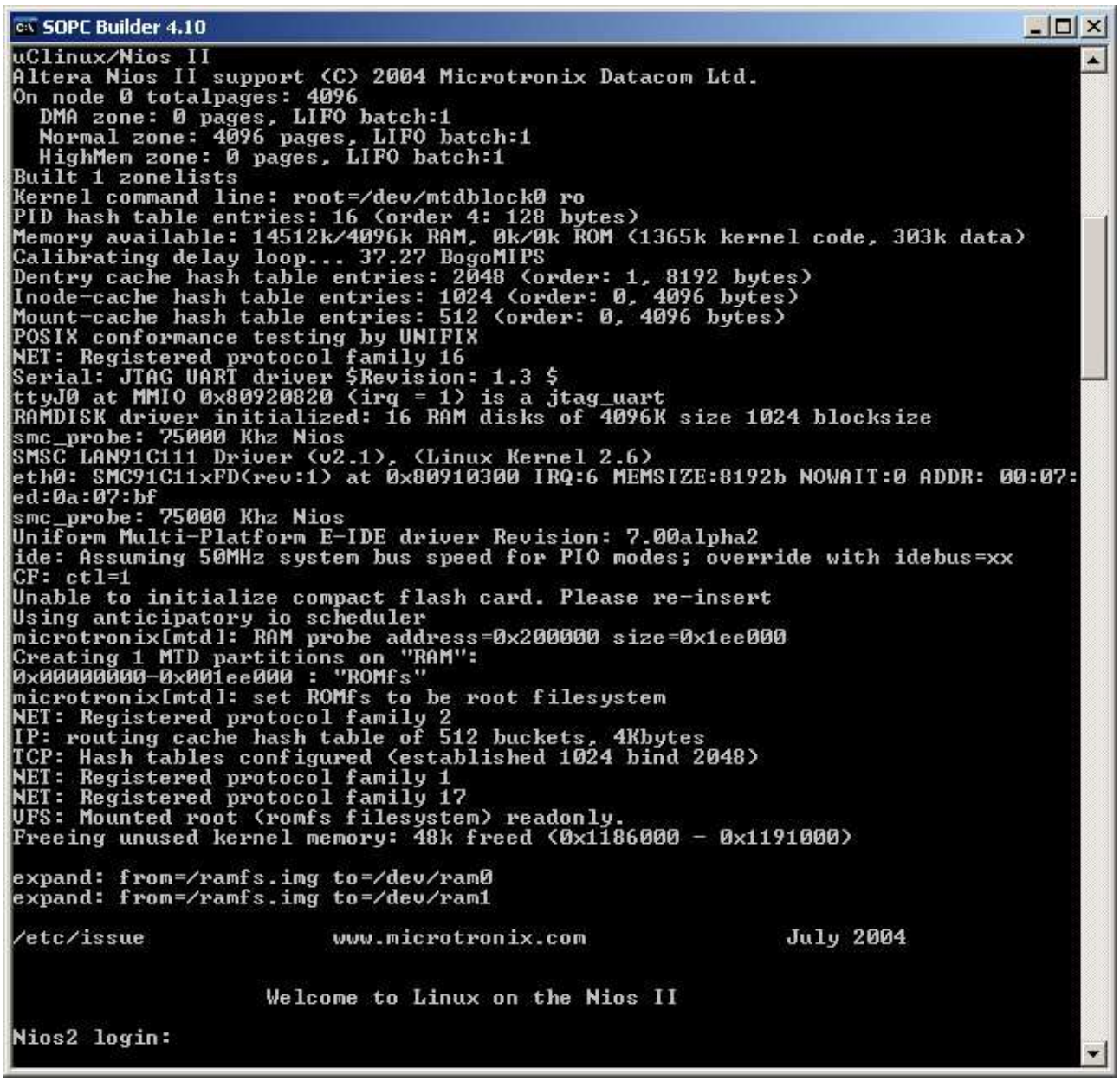

Figure 11 : Boot du noyau µClinux par le processeur softcore NIOS II

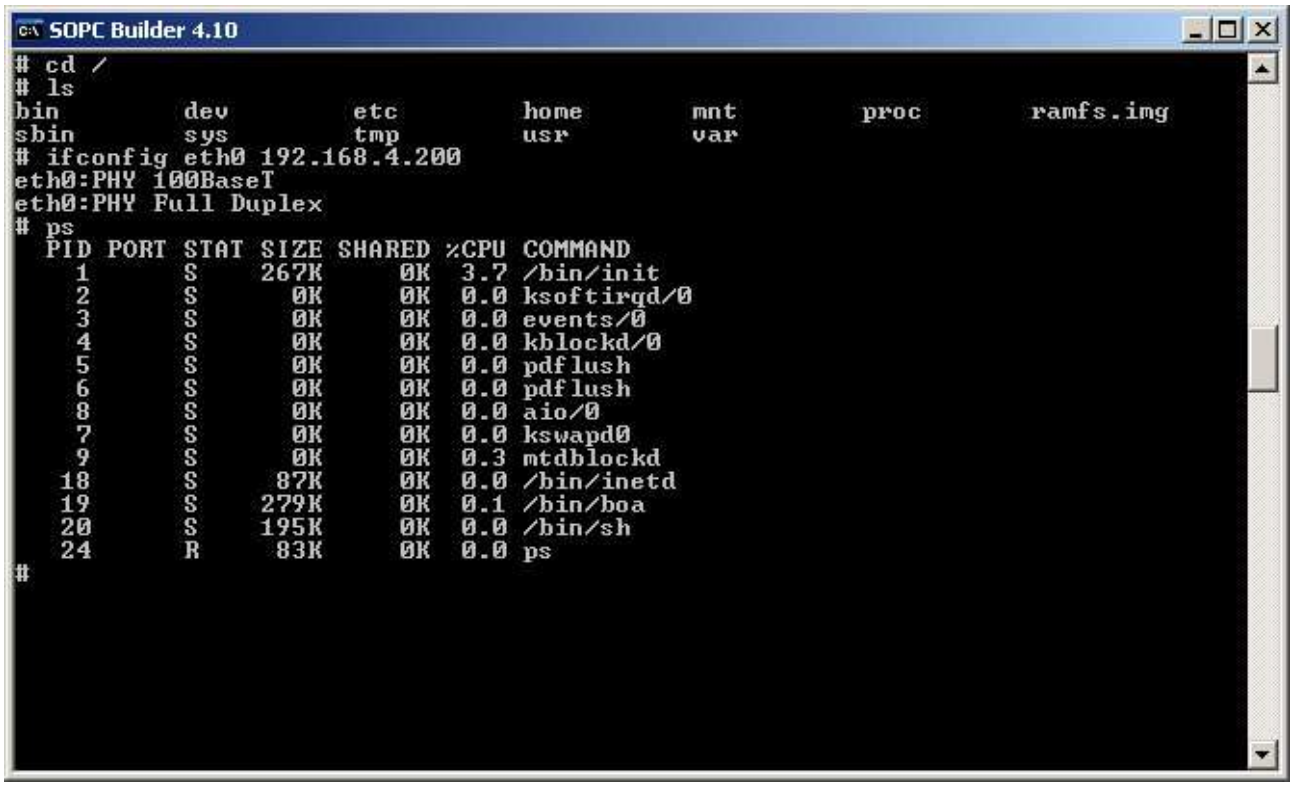

Figure 12 : µClinux en action !

Exemples de mise en oeuvre de Linux embarqué dans un SoPC NIOS II

Quelques exemples de mise en oeuvre de Linux embarqué sur le processeur softcore NIOS II sont présentés. On notera que l'on retrouve en récurrence le besoin de piloter des E/S par logiciel que ce soit localement depuis la carte cible ou bien à distance par le réseau Internet en mettant en oeuvre une connectivité IP. C'est en fait une caractéristique essentielle des systèmes embarqués d'être aujourd'hui massivement communicants en utilisant des outils banalisés de contrôle comme un navigateur web.

Exemple 1 : application de base Hello World

Ce premier exemple a juste pour but de montrer comment on développe une application Linux embarqué µClinux pour la carte cible SoPC NIOS II. L'IDE Eclipse fourni et configuré via les plugins Altera permet de faire cela très facilement. L'auteur émet quand même un petit bémol : tout étant caché ou presque, on n'accède plus à tous les paramètres de contrôle fins : fichier de commandes d'édition liens, utilisation mémoire par l'application µClinux...bref des choses très importantes quand on fait de l'embarqué.

Le développement est simplissime :

1. Création d'un projet Eclipse « *Linux Application Project* »
2. Création du fichier source *hello.c*.
3. Création du fichier *Makefile*.
4. Compilation croisée.

5. Recopie du fichier exécutable *hello* sous */bin* dans le système de fichiers root.
6. Génération du fichier binaire correspondant au système de fichiers root et programmation de la mémoire FLASH avec.
7. Boot du noyau µClinux et tests.

Ceci est bien sûr valable avec n'importe quelle application µClinux et l'on peut utiliser le debugger GNU GDB interfacé à Eclipse pour déverminer une application µClinux s 'exécutant sur la carte cible (par la liaison JTAG).

La figure 13 présente le développement de l'application *hello* sous Eclipse.

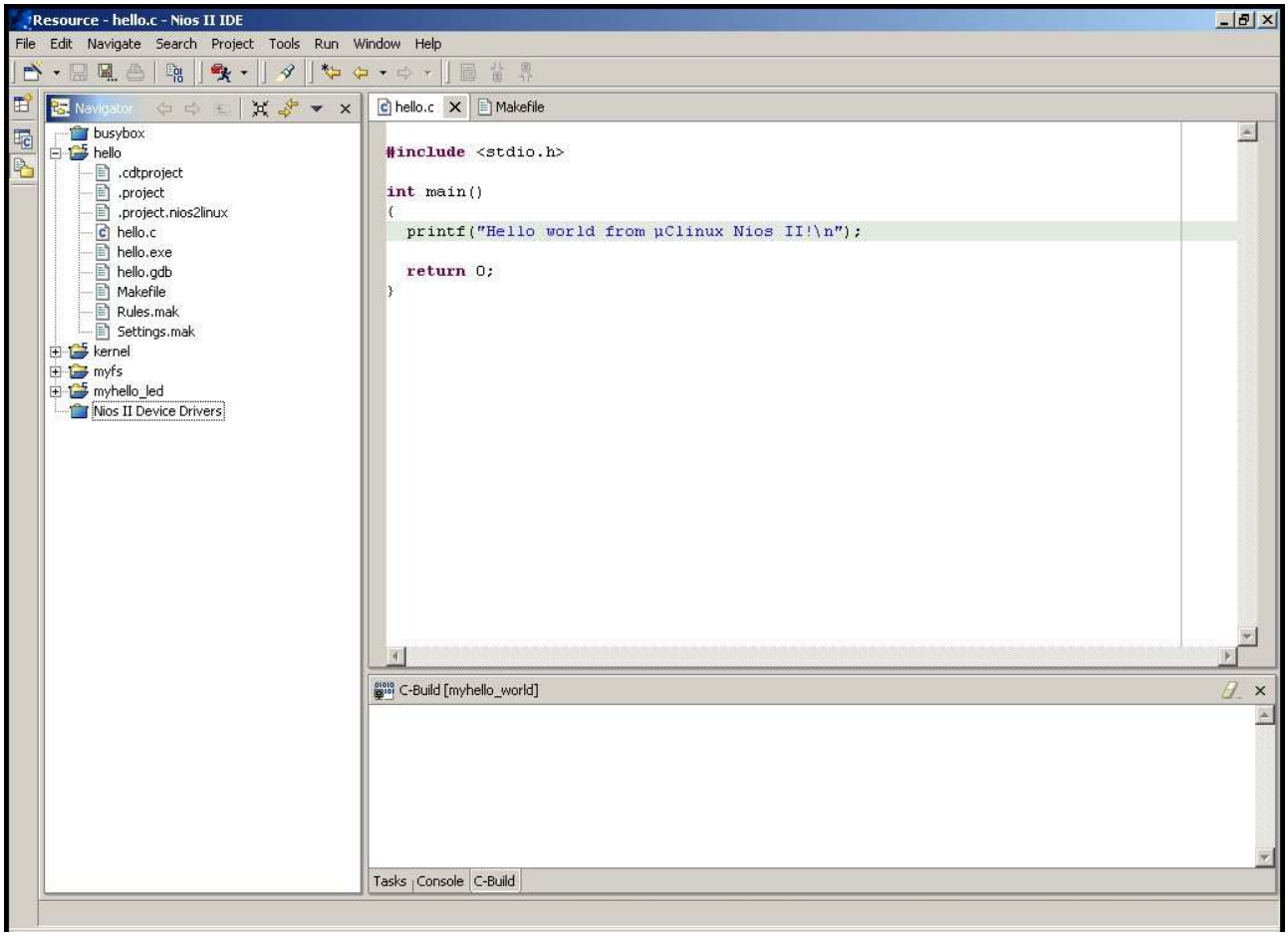

Figure 13 : développement de l'application µClinux NIOS II hello

La figure 14 présente le résultat de l'exécution de l'application µClinux hello sur la carte cible.

Figure 14 : développement de l'application µClinux NIOS II hello

Exemple 2 : serveur web embarqué

Ce deuxième exemple est maintenant un classique du genre. Le but est d'embarquer un serveur web sur la carte cible pour piloter à distance des E/S de la carte.

Pour cela, on embarque le serveur web *boa* (http://www.boa.org/) puis l'on développe des scripts CGI pour piloter les E/S. Les E/S considérées sont ici les 8 leds connectées au port IO *led_pio* du processeur softcore NIOS II. Le mapping mémoire des E/S de la carte cible est donné pour rappel à la figure 6. Il convient comme dans l'exemple 1 de créer une application µClinux *led_enseirb* puis d'intégrer l'exécutable *led_enseirb.cgi* dans le système de fichiers root sous */home/httpd/cgi-bin* pour qu'il soit considéré comme un script CGI par le serveur web *boa*.

Il faut noter que la méthodologie reste la même pour piloter d'autres types de périphériques d'E/S : il suffit de développer le logiciel de contrôle inclus dans le script CGI (qui est en fait un driver dans l'espace utilisateur).

Le fichier source du script CGI *led_enseib.c* est donné ci-après :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// Chemin vers fichier nios2_system.h pour mapping memoire
#include
<C:\altera\kits\nios2\bin\eclipse\workspace\kernel\build\include\nios2_system.h>

int main(void)
{

  unsigned int content_length;
  unsigned char mydata;
  char *cgiinput;
```

```
  // Programmation PIO Leds en sortie, leds eteintes
  np_pio *pio = na_led_pio;
  pio->np_piodirection = 0x03;
  pio->np_piodata = 0x00;

  content_length = 0;
  mydata = 0;

  // Entete reponse HTTP
  printf("HTTP/1.0 200 OK\n");
  printf("Content-type: text/html\n");
  printf("\n");

  // Donnees reponse HTTP
  printf("<html>");
  printf("<title> NIOS II CGI testings </title>\n");

  if ( !(content_length = atoi(getenv("CONTENT_LENGTH"))) ) {
    printf("No Content-Length was sent with the POST request.\n");
    printf("</html>\n");
    exit(1) ;
  }

  if ( !(cgiinput=(char *)malloc(content_length+1)) ) {
    printf("Could not allocate memory for cgiinput.\n");
    printf("</html>\n");
    exit(1) ;
  }

  if (!fread(cgiinput, content_length, 1, stdin)) {
    printf("Could not read CGI input from STDIN.\n") ;
    printf("</html>\n");
    exit(1) ;
  }

  cgiinput[content_length]='\0' ;

  if (strstr(cgiinput, "LED1")) {
    printf("<h1>LED1 set</h1>");
    mydata |= 0x01;
  }
  if (strstr(cgiinput, "LED2")) {
    printf("<h1>LED2 set</h1>");
    mydata |= 0x02;
  }
      . . .
  if (strstr(cgiinput, "LED7")) {
    printf("<h1>LED7 set</h1>");
    mydata |= 0x40;
  }
  if (strstr(cgiinput, "LED8")) {
    printf("<h1>LED8 set</h1>");
    mydata |= 0x40;
  }
  // Pilotage leds
  pio->np_piodata = mydata;

  printf("</html>\n");
  free(cgiinput);
  exit(0);
}
```

On notera que le lien avec les périphériques d'E/S est réalisé grâce à l'usage du fichier

nios2_system.h issu du *cpu_sdk* généré après la synthèse matérielle.

Le serveur web boa a une page d'accueil *index.html* dont le code HTML est le suivant :

```
<html>
<head><title>µClinux NIOS II CGI Testings</title></head>
<body bgcolor="white" text="black">
      <table width="100%">
            <tr valign="top">
                  <td width="80">
                        <FORM METHOD="POST"
                        ACTION="http://192.168.4.150/cgi-bin/led_enseirb.cgi">
                        1<INPUT TYPE="checkbox" NAME="LED1" VALUE="led1"
CHECKED>
                        2<INPUT TYPE="checkbox" NAME="LED2" VALUE="led2">
                        3<INPUT TYPE="checkbox" NAME="LED3" VALUE="led3">
                        4<INPUT TYPE="checkbox" NAME="LED4" VALUE="led4">
                        5<INPUT TYPE="checkbox" NAME="LED5" VALUE="led5">
                        6<INPUT TYPE="checkbox" NAME="LED6" VALUE="led6">
                        7<INPUT TYPE="checkbox" NAME="LED7" VALUE="led7">
                        8<INPUT TYPE="checkbox" NAME="LED8" VALUE="led8">
                        <p>
                        <INPUT TYPE="submit" value="OK">
                        </FORM>
                  </td>
            </tr>
      </table>
</body>
</html>
```

On notera que la carte cible NIOS II est configurée sous µClinux pour posséder l'adresse IP 192.168.4.150.

Le test se fait en utilisant un simple navigateur web. Les figures suivantes présentent le résultat obtenu pour le pilotage à distance par Internet de la carte cible NIOS II/µClinux.

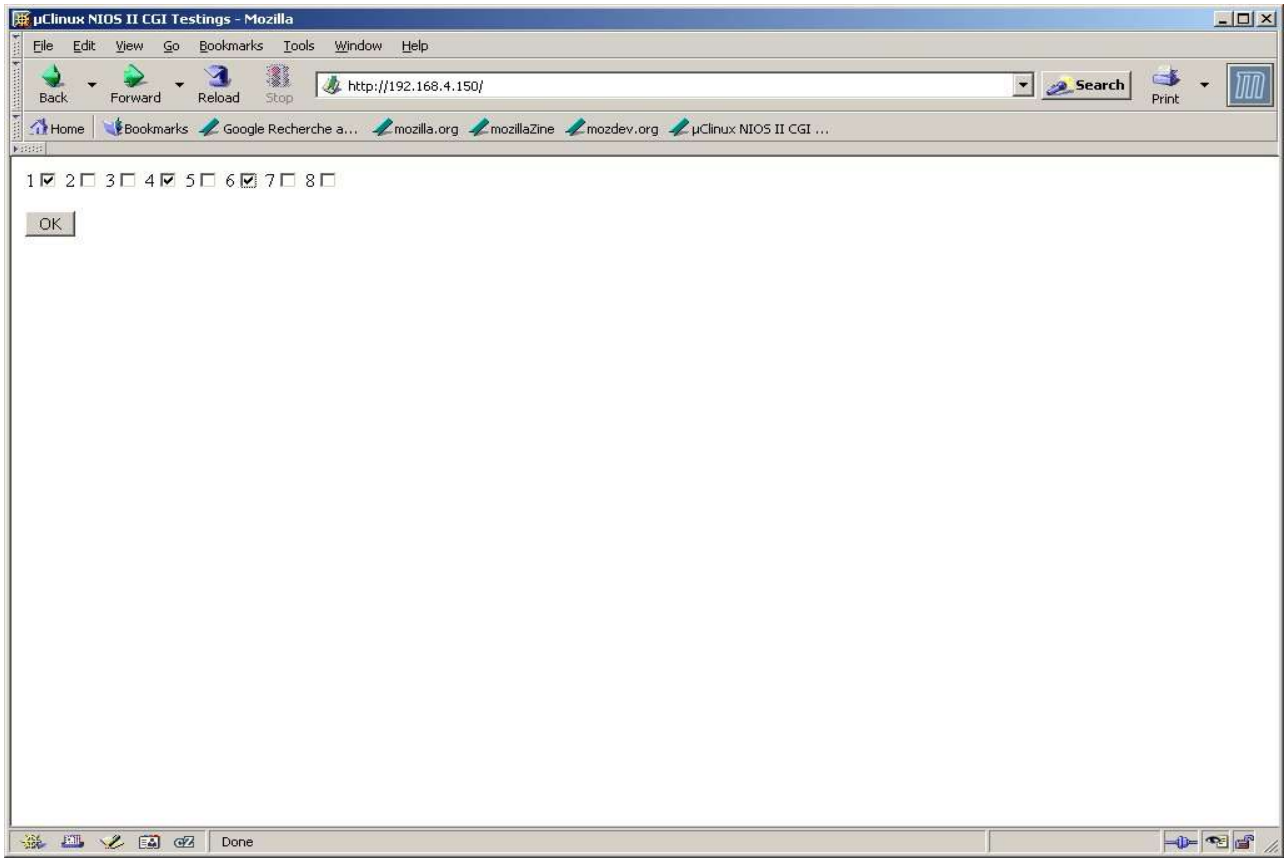

Figure 15 : page d'accueil du serveur web embarqué sur NIOS II

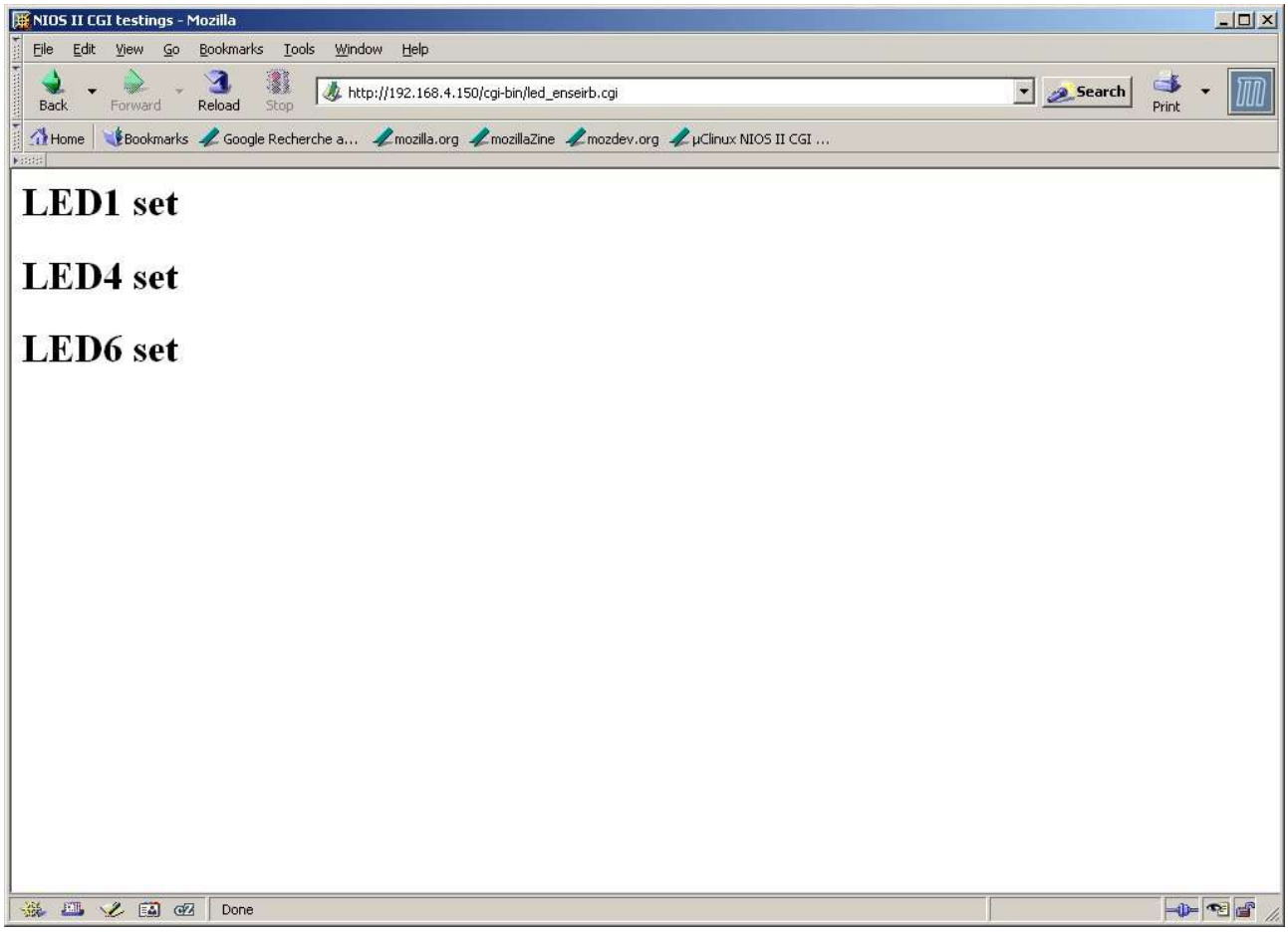

Figure 16 : contrôle des leds 1, 4 et 6 de la carte cible NIOS II/µClinux

Exemple 3 : système d'acquisition et de traitement vidéo

Ce troisième et dernier exemple concerne la réalisation d'un système d'acquisition-traitement d'images en temps réel. Ce travail fait l'objet d'une thèse sur l'implantation matérielle d'algorithmes de traitement d'images.

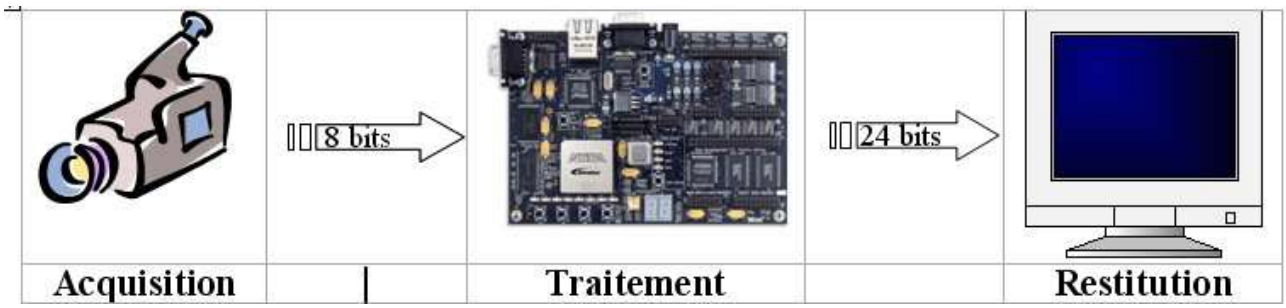

Figure 17 : système d'acquisition, traitement et restitution d'images

Le système réalisé est composé d'une caméra numérique permettant l'acquisition de l'image et la génération des signaux numériques. Ces signaux sont traités par la carte cible Altera Stratix 1S10. Après traitement par la carte cible, les échantillons de l'image traitée ainsi que les signaux de contrôle sont envoyés vers un module contrôleur VGA connecté à un moniteur VGA. L'image restituée a une résolution de 640x480 en niveaux de gris.

Pour cela, un bloc IP pour l'acquisition de l'image de la caméra a été développé en VHDL. Il intègre une mémoire FIFO et un contrôleur DMA connecté au bus du processeur softcore NIOS II de la carte cible (bus *Avalon*).

Le module contrôleur VGA est le module Lancelot (http://www.fpga.nl/lancelot.html) qui est du matériel libre fourni avec un bloc IP en libre. Ce module intègre aussi une mémoire FIFO et un contrôleur DMA connecté au bus du processeur softcore NIOS II de la carte cible.

Les 2 blocs IP ont été intégrés à Quartus II. Les figures 18 et 19 précisent le système SoPC ainsi réalisé :

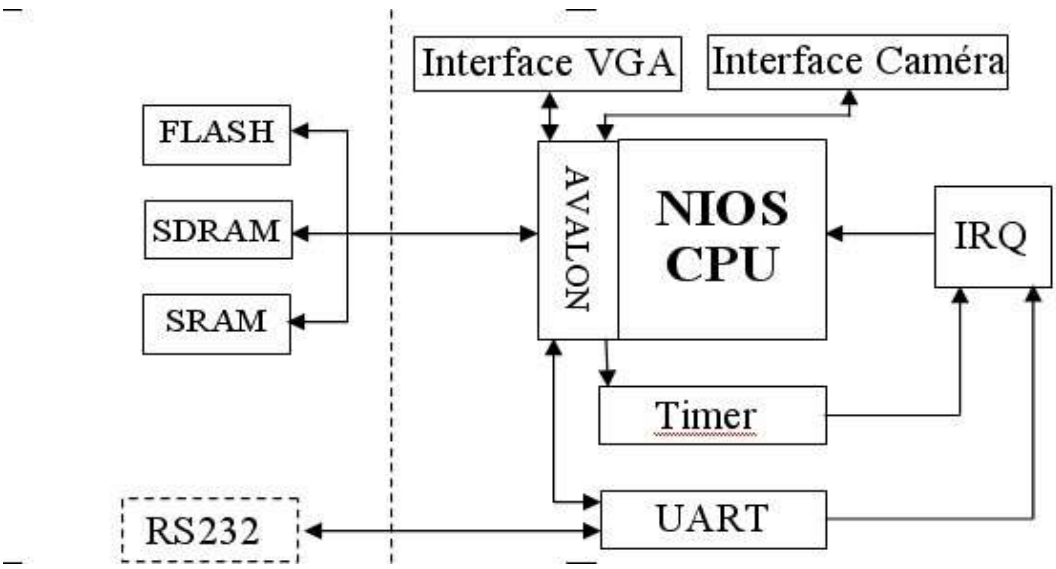

Figure 18 : système SoPC NIOS II pour l'acquisition vidéo

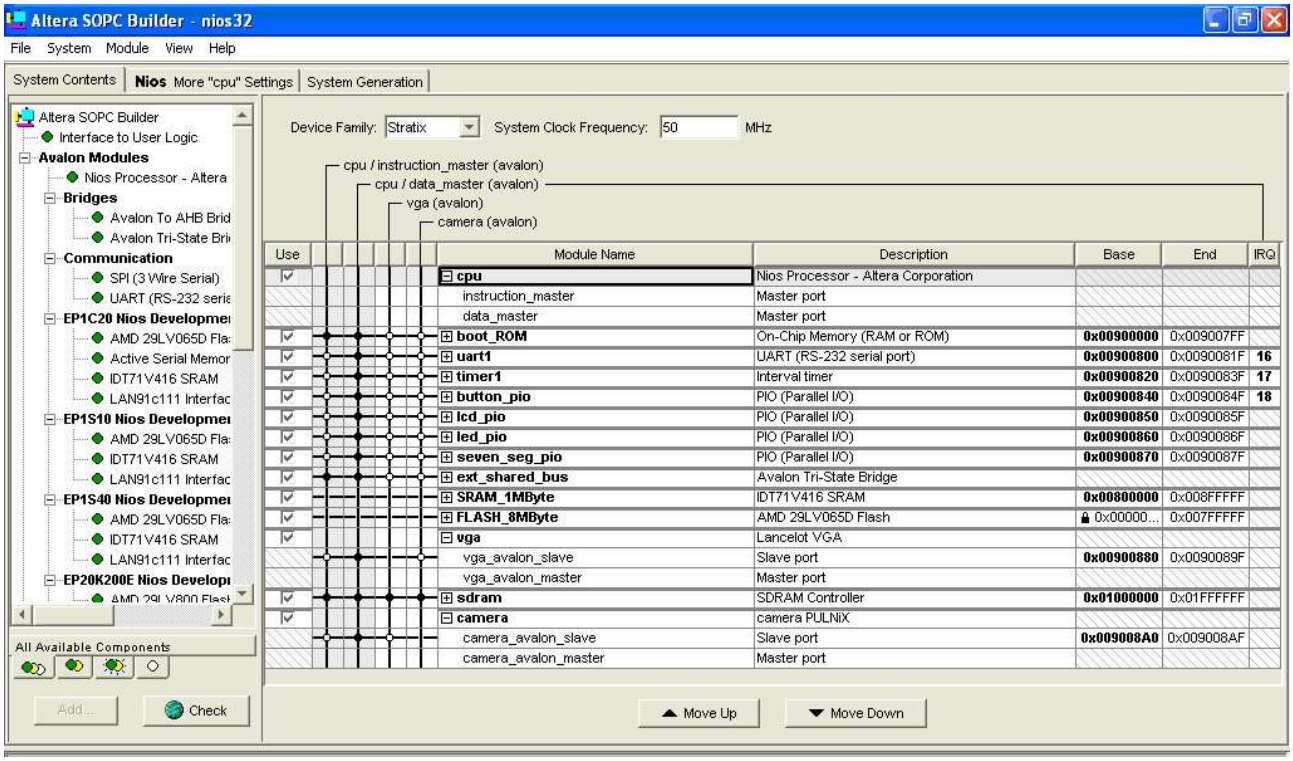

Figure 19 : projet Quartus II SoPC Builer correspondant

Pour le bon fonctionnement du système cible, une synchronisation est nécessaire entre ces deux modules car un seul peut accéder à la mémoire partagée SDRAM à un instant donné. C'est le module VGA ici qui est prioritaire puisqu'une discontinuité de transfert de données entre la mémoire SDRAM et la FIFO du module VGA provoque une perturbation au niveau de l'affichage de l'image.

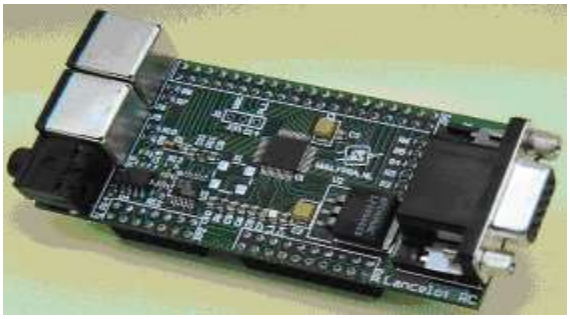

Figure 20 : module VGA Lancelot

Le processeur NIOS II est cadencé à la fréquence de 50 MHz. A cette fréquence, 20 % du temps est consommé pour les transferts DMA caméra et module VGA, ce qui laisse 80% du temps pour le processeur NIOS II pour l'exécution d'un système d'exploitation et des algorithmes de traitement vidéo.

Le noyau Temps Réel microC/OS II a été dans un premier temps adapté à la carte cible pour contrôler le système d'acquisition, traitement et d'acquisition d'images. Un algorithme simple de détection de contours a été implanté sous forme logicielle dans un premier temps.

µClinux a été intégré ensuite au système cible sans problème. Les drivers Linux de contrôle des modules caméra et VGA sont en cours d'écriture. L'idée est d'intégrer un codeur de compression vidéo H.263 modifié pour accepter des accélérations matérielles...

NIOS II, µClinux et le codesign

On a pu voir sur les exemples précédents la mise en oeuvre de µClinux sur le processeur softcore Altera NIOS II. C'est véritablement une offre de codesign que l'on a à disposition et cette approche va révolutionner la conception des systèmes numériques complexes.

L'approche codesign apparaît ici dans le développement conjoint matériel-logiciel :

- On développera une partie de l'application sous forme d'un bloc IP (en VHDL par exemple) pour bénéficier d'une accélération matérielle du traitement. Cela revient en fait à développer un coprocesseur matériel spécifique.
- On développera l'autre partie de l'application sous forme logicielle pour bénéficier de la souplesse de la « logique programmée » exécutée par le processeur softcore embarqué dans le système cible.

L'apport de Linux embarqué (µClinux ici) est indéniable pour différentes raisons :

- On a un système de fichiers à disposition.
- On a un système d'exploitation multitâche.

- On peut réutiliser des briques logicielles issues du logiciel libre. On peut éventuellement intégrer ses modifications pour introduire des accélérations de traitement par matériel (codesign).

Le lien entre le coprocesseur matériel spécifique et le processeur softcore exécutant le noyau µClinux sera établi en développant un driver Linux. Il est à noter que dans le cas de µClinux et en absence de MMU, on peut développer :

- Un driver ou un module Linux.
- Un driver dans l'espace utilisateur (*user space driver*). En absence de MMU, un processus Linux accède directement au matériel : on n'a donc pas de driver Linux à écrire mais on peut planter tout aussi facilement son système... L'absence ou la présence d'une MMU est d'ailleurs un sujet de discorde parmi les acteurs du monde de l'embarqué. Pendant des années, dans l'embarqué, on utilisait des processeurs sans MMU ou bien on la dévalidait s'il y en avait une. Depuis la percée de Linux dans l'embarqué, la tendance s'est inversée car la MMU permet quand même d'avoir un système plus robuste ; ce qui est une qualité que doit posséder tout système embarqué.

Conclusion

La conception des systèmes numériques complexes nécessite maintenant de mettre en oeuvre la méthodologie de *codesign*. L'intégration grandissante sur le silicium a permis d'avoir une approche de conception orientée système et l'on développe maintenant des systèmes sur silicium SoC et SoPC. Ces systèmes SoC et SoPC intégrent généralement un processeur embarqué si bien qu'il semble naturel d'embarquer Linux dans ces systèmes !

La mise en oeuvre de la solution de codesign sur SoPC Quartus II d'Altera a été présentée. Bien sûr, son concurrent Xilinx a aussi sa solution...

La mise en oeuvre de Linux embarqué µClinux couplé au processeur softcore NIOS II d'Altera a été présentée au travers de différents exemples. Il est clair que le couple Linux embarqué-processeur softcore est une solution à explorer et à exploiter dès que l'on a un système numérique complexe à construire, ce qui qui offre de belles perspectives à notre système d'exploitation préféré...

L'auteur en profite enfin pour dédier cet article à ses maîtres qui lui ont tout appris du hard au soft : B. Humbert du Centre de Recherches Nucléaires de Strasbourg, J.L. Pedroza du CENBG de Bordeaux-Gradignan et enfin son collègue P. Nouel de l'ENSEIRB...

Bibliographie

- Embedded system-design with soft and hardcore FPGA's. P. Pelgrims and al. http://emsys.denayer.wenk.be
- Le site d'Altera http://www.altera.com
- Le site de Xilinx http://www.xilinx.com
- Le site de l'auteur http://www.enseirb.fr/~kadionik
- Le site des blocs IP libres opencores.org www.opencores.org
- Le page de l'auteur sur le System On Chip http://www.enseirb.fr/~kadionik/SoC/soc.html
- Le projet µClinux http://www.uclinux.org
- Le site de Microtronix http://www.microtronix.com

- Le forum NIOS forum http://www.niosforum.com/forum
- Rapport de projet avancé, option Systèmes Embarqués de l'ENSEIRB. Linux embarqué sur NIOS II. J. Laboury, P. Mirgoudou, G. Pesquet, L Zanoni. 2005
- Microtronix NIOS II Linux Distribution. Quick Start Guide.
- Microtronix NIOS II Linux Distribution. Reference Guide.
- Interface VGA Lancelot http://www.fpga.nl/lancelot.html
- Hardware Platform Design for Real-Time Video Applications. A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, Ph. Marchegay. International Conference on Microelectronics. 2004.