



Intégration d'un système linux embarqué sur une carte ATNGW100



*CHAPELLE Quentin
LAGUET Benoit
SABATIER Julien
ZEDDA Yannick*

Nous autorisons la diffusion de notre rapport sur l'intranet de l'IUT

Remerciements

Nous tenons à remercier grandement M. DELON, administrateur système à l'IUT pour le temps qu'il nous a consacré ainsi que pour ses conseils avisés.

Sommaire

1. Introduction

2. Études du sujet

- 2.1. Présentation synthétique
- 2.2. Présentation de la carte
- 2.3. Les mémoires et leurs systèmes de fichiers
- 2.4. Alternatives aux flashages
- 2.5. Cross-compilation et intégration

3. Réalisation

- 3.1. Procédure générale
- 3.2. Mise en place du serveur tftp
- 3.3. Création du système avec buildroot
- 3.4. Configuration du système
- 3.5. Flashage de la carte
- 3.6. Scripts Finaux

4. Bilan technique

5. Conclusion

6. English summary

7. Lexique

8. Sources

9. Annexes

1. Introduction

Dans le cadre de ce projet tutoré proposé et encadré par M. DELON David , notre équipe composée de M. CHAPELLE Quentin, M. LAGUET Benoit, M. ZEDDA Yannick et M. SABATIER Julien a réalisé une passerelle à l'aide d'un noyau linux 2.6.27 et d'une carte ATNGW100.

Cette carte produite par le groupe Atmel, d'architecture AVR32, inclut déjà d'origine, un système lui permettant d'être utilisée comme une passerelle. Notre système final pourra être administré par SSH, servir de firewall et embarquera le programme d'analyse de TCPDUMP, le tout avec un espace mémoire réduit.

Notre plus gros défi pour mener à bien ce travail, a été la contrainte matériel. En effet nous avons dû concevoir un système minimaliste n'utilisant qu'une des deux mémoires disponibles c'est à dire, tenant sur 8Mo maximum et pouvant tourner avec seulement 32Mo de mémoire RAM. Le deuxième problème vient toujours du matériel. En effet pour flasher une carte ATNGW100 il faut un certain temps assez contraignant, M. DELON nous a donc demandé de mettre en place un serveur pour démarrer la carte sur le réseau.

Pour tenir compte au mieux de ces contraintes nous allons dans un premier temps analyser et étudier le système, les problèmes et les solutions qui en découlent. Dans un second temps, en tenant compte de cette étude, nous expliquerons plus en détail les étapes de la réalisation. Enfin pour clore notre rapport nous ferons le point sur le travail effectué.

2.1 Présentation synthétique

Notre équipe a pour but de développer un système Linux embarqué exploitant la carte ATNGW100. Une fois le projet terminé, la carte pourra être contactée par SSH et embarquera les fonctionnalités de firewall ainsi que l'outil TCPDUMP. De plus elle devra assurer le rôle de passerelle.

Nous travaillerons ici sur un processeur d'architecture AVR32 et nous avons à notre disposition seulement 8Mo de ROM et 32 de RAM. Plusieurs problèmes se sont alors posés.

Dans un premier temps nous les avons donc étudié pour arriver à trouver des solutions optimales, notamment au niveau du choix du système de fichiers, de la chaîne de compilation croisée et des alternatives aux flashages de la carte.

Après cette phase de recherche nous avons développé un système fonctionnel que nous avons testé à l'aide du démarrage réseau. Une fois réalisé, nous avons pu flasher la carte pour y apposer notre système.

2.2 Présentation de la carte ATNGW100

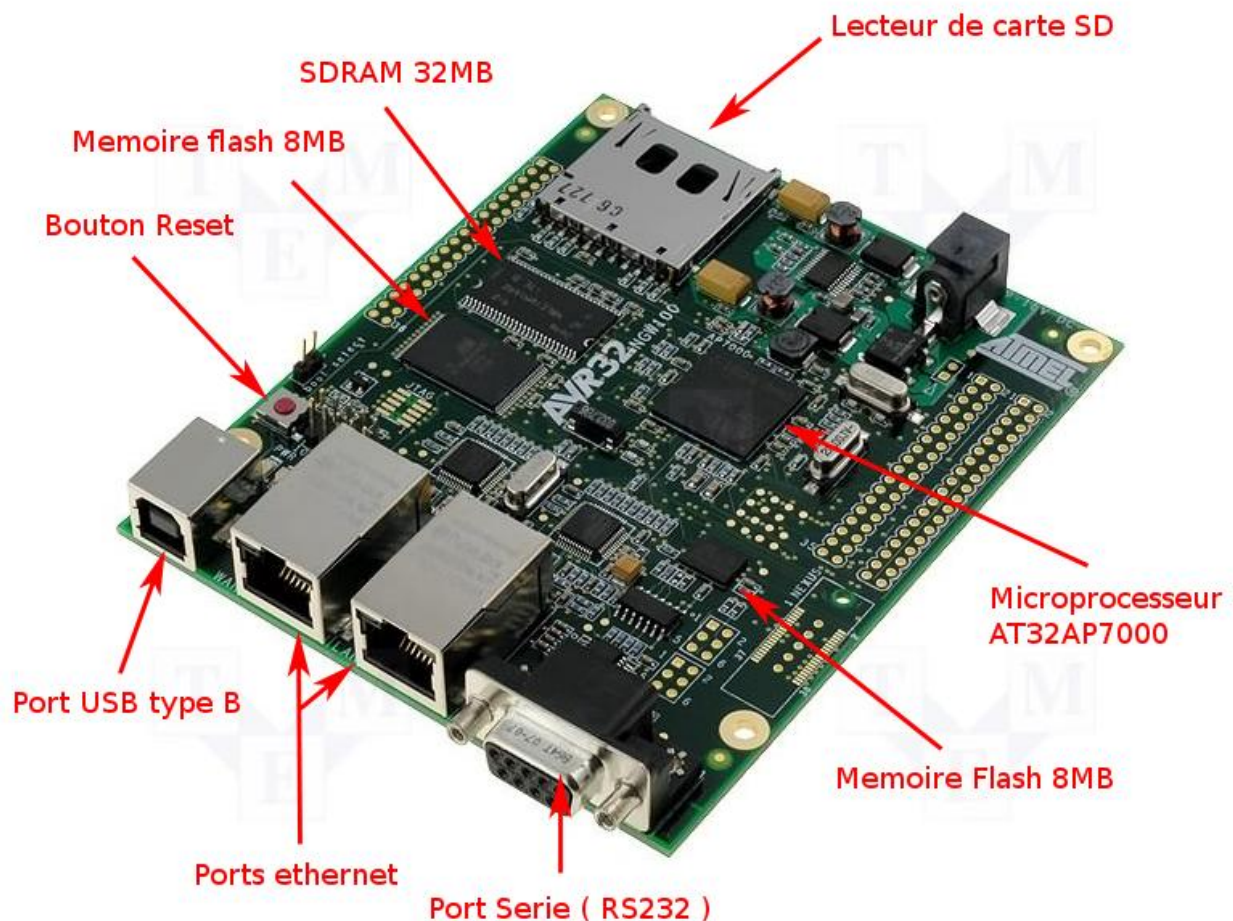


figure 1

La carte Atmel ATNGW100 embarque un microprocesseur AT32AP7000 d'architecture AVR32, et possède deux mémoires mortes (ROM) de 8 Mo. L'une d'entre elle contient le système tandis que l'autre contient le /usr. Cependant dans le cadre de notre projet nous n'avons eu le droit d'utiliser seulement la mémoire contenant actuellement le système. La carte possède aussi deux interfaces Ethernet qui permettent à la carte de jouer son rôle de passerelle et qui nous a permis de démarrer la carte en réseau. Elle possède aussi un port série RS232 qui nous a servi à communiquer via l'outil minicom* et à flasher la carte avec l'aide du logiciel kermi, un port USB, un lecteur de carte SD que nous n'avons pas utilisé ainsi qu'une alimentation.

2.3 La mémoire et son système de fichier

Les mémoires flash :

La carte AT32NGW100 possède deux mémoires flash NOR de 8MO. Le système d'exploitation personnalisé que nous allons créer sera mis sur la mémoire qui possède l'amorceur (U-boot). La mémoire flash est une EEPROM (Electrically-Erasable Programmable Read-Only Memory) rapide et effaçable par secteur complet, qui permet la modification de l'espace mémoire en plusieurs endroits en une seule opération. Ce qui la rend très utile et véloce lors d'une écriture en plusieurs points de la mémoire.

Il y a deux principaux types de mémoires flash qui se différencient aux niveau des portes logiques utilisées pour les cellules de stockage. Nous donc allons maintenant vous présenter ces deux types de mémoires (NOR et NAND), ainsi que leurs différences.

Mémoire flash NOR :

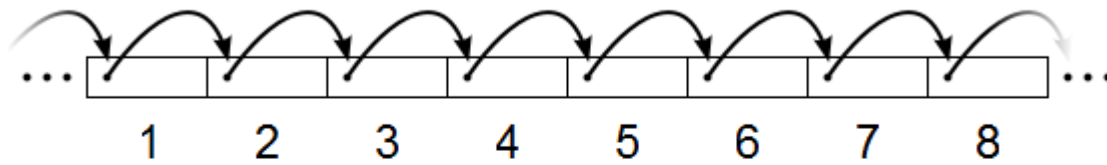
La mémoire flash NOR a un accès aux données direct et rapide XIP (eXecute In Place) grâce à son interface d'adressage, ce qui implique que les temps d'effacement et d'écriture sont un peu longs. Le fabricant garanti le stockage des données à 100%. De fait, la plupart des systèmes d'exploitation des appareils électroniques grand public sont stockés dans une mémoire NOR, téléphones portables, les cartes mères, périphériques (imprimantes, appareils photos, etc...).

Mémoire flash NAND :

La mémoire flash NAND est rapide pour effacer et stocker des données mais l'accès à celles-ci est plus lent (accès séquentiel). Le stockage des données n'est pas garanti 100 % par le fabricant ce qui implique la mise en place d'un système de gestion des erreurs (similaire à ceux des disques durs). Ce qui la rend moins apte à supporter des applications de type XIP. En revanche elle est beaucoup utilisée pour le stockage de données (Carte MMC, Carte SD, Carte MS, etc...).

Il existe des systèmes de fichiers spécialement conçus pour la mémoire flash : JFFS, JFFS2, YAFFS, UBIFS. Ils permettent, entre autres, d'éviter la réécriture répétée sur une même zone, ceci afin de prolonger la durée de vie de la mémoire flash.

Accès séquentiel



Accès direct

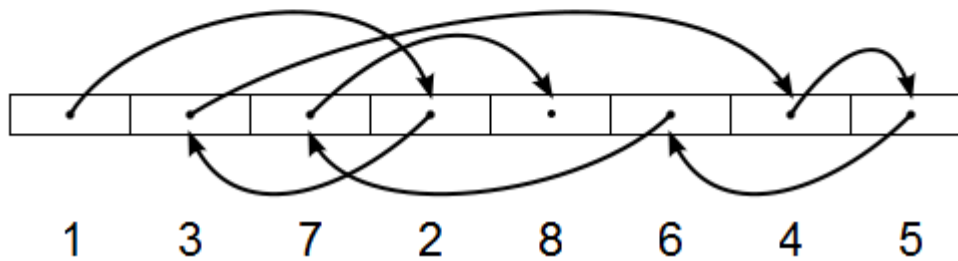


figure 2

Présentation des systèmes de fichiers :

Plusieurs systèmes de fichiers sont utilisés pour les mémoires flash, comme cité précédemment. ATMEL utilise le système JFFS2 (Journaling Flash File System version 2), mais nous avons constaté que le système UBIFS (Unsorted Block Image File System) aurait été plus performant, cependant nous ne pouvions pas l'utiliser car notre amorceur (uboot) n'est pas assez récent pour gérer ce système de fichiers.

Il existe aussi deux autres systèmes de fichiers YAFF2 (Yet Another Flash File System) et logFS mais ceux-ci sont plus développés pour les NAND que les NOR. Nous avons donc gardé le système JFFS2 pour la gestion de la mémoire car nous ne pouvions pas modifier uboot afin d'utiliser UBIFS. Nous allons maintenant vous présenter les caractéristiques du système de fichiers JFFS2.

JFFS2 est la version 2 de JFFS qui révolutionne celui-ci avec un changement du mode d'écriture qui passe d'une écriture circulaire, c'est à dire que le système écrit au fur et à mesure dans une pile en parcourant la mémoire. A un mode d'écriture par blocs, qui permet d'écrire les données dans des blocs de même taille.

Etant donné la taille conséquente des blocs mémoires pour un disque SSD, le système de fichiers JFFS2 possède un garbage collector (ramasse-miettes) afin de fusionner les données des nouveaux blocs et ainsi libérer la place des anciens. JFFS2 construit un index des blocs mémoires au montage ce qui pose problème lorsque la mémoire possède une forte capacité de stockage car il doit pour pouvoir créer l'index, scanner tous les blocs. Dans le cadre de notre projet, cela ne pose pas de problème car nos mémoires ont une petite capacité (8 Mo).

Pour conclure, on peut dire que le système de fichiers JFFS2 est adapté aux mémoires flash de la carte AT32NGW100.

2.4 Alternatives aux flashages

Pour éviter les flashages inutiles, long et à terme détériorant la mémoire, nous avons utilisé les protocoles tftp, dhcp et NFS pour démarrer notre système par le réseau.

fonctionnement Global: Booter en réseau consiste à télécharger en mémoire un noyau présent sur un serveur distant en utilisant les protocoles dhcp et tftp. Ensuite NFS se charge de démarrer et de monter le système de fichier root.

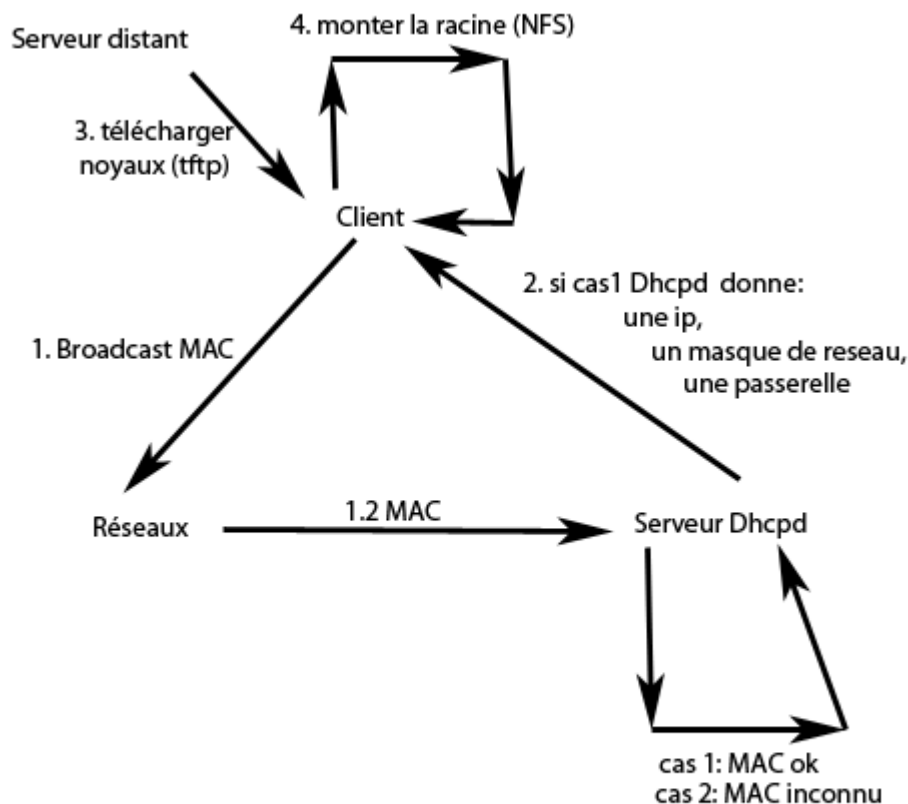


figure 3

Fonctionnement détaillé: Le client va diffuser en broadcast son adresse MAC sur le réseau. Cette adresse est reçue par le serveur DHCP qui, va vérifier dans sa base de données si celle-ci est connue. Si l'opération est un succès, le serveur retourne une adresse IP, un masque réseau et une adresse passerelle qui va permettre au client le téléchargement du noyau à travers tftp. Enfin pour clore le tout le noyau va monter en NFS le répertoire root.

Remarque: pour plus de simplicité nous aurions pu utiliser BOOTP à la place de DHCP. Mais ce dernier est beaucoup plus performant, récent et flexible.

2.5 Cross-compilation et intégration

Définitions Globales Diverses :

Compilateur croisé : (Cross compiler) programme capable de traduire un code source en code objet ayant un environnement d'exécution différent de celui où la compilation est effectuée.

La cross compilation implique l'utilisation de certains outils appelés toolchain

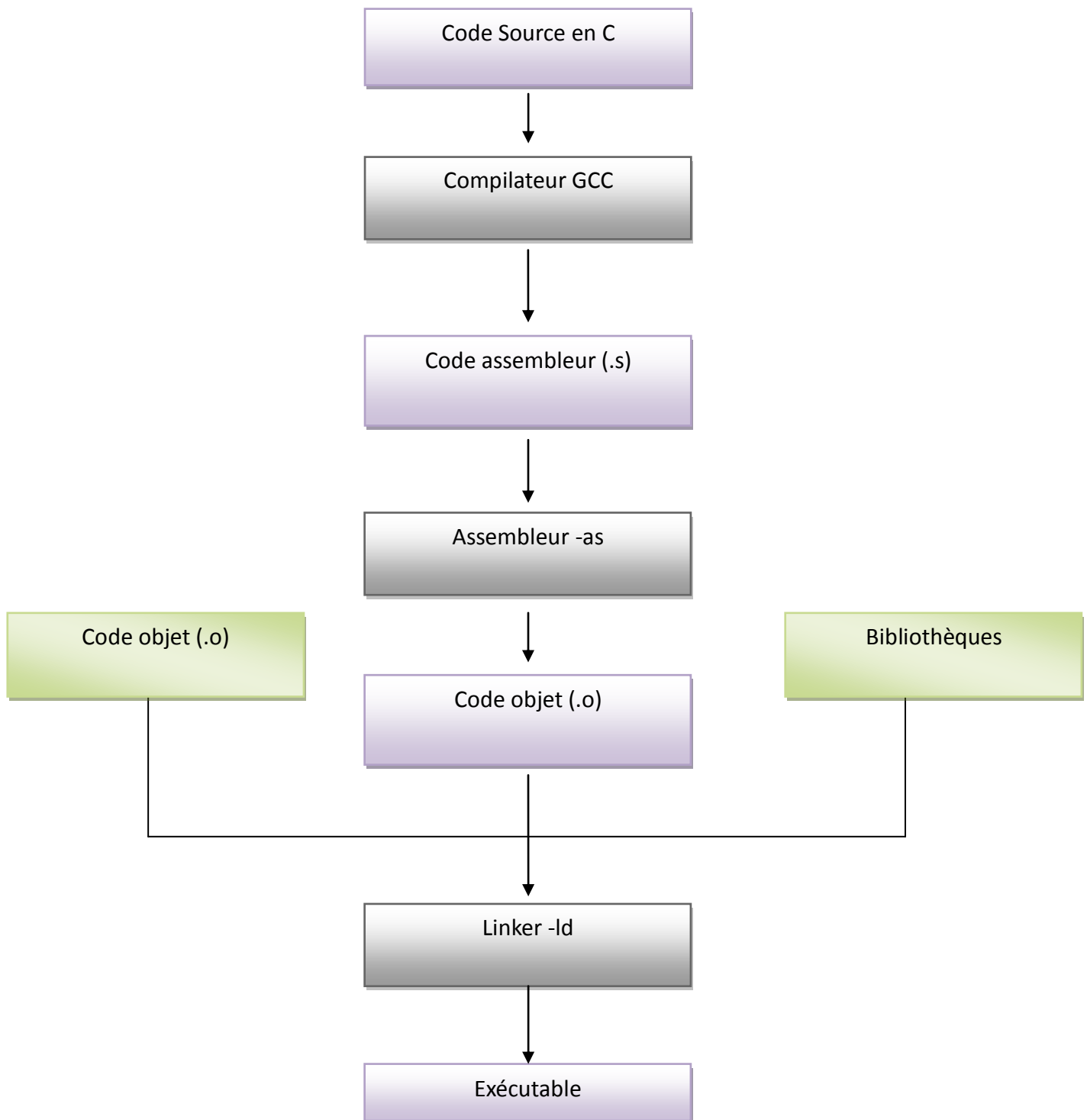


figure 4

3.2 Mise en place d'une solution TFTP

Procédure:

Dans un premier temps nous avons récupéré le système officiel pour faire des tests de notre serveur tftp, dhcp et nfs. Ensuite nous avons agi en deux temps. Tout d'abord nous allons détailler la configuration du serveur puis celle de la carte (U-boot).

1-Configuration du serveur

Tout d'abord nous avons installé les paquets nécessaires (dhcp3-server, nfs-kernel-server et tftpd-hpa).

Ensuite nous avons créé un dossier /tftpboot qui contiendra notre système et un dossier /tftpboot/node1 contenant la racine de notre système.

on configure /etc/inet.conf en vérifiant ou en ajoutant

```
tftp dgram udp wait root /usr/sbin/in.tftpd tftpd -s /tftpboot
```

on ajoute dans le fichier /tftpboot/node1/etc/fstab

```
none /proc proc default 0 0
none /dev/pts exec,dev,suid,rw 0 0
```

on configure le fichier /etc/exports en ajoutant

```
/tftpboot/node1 (rw,no_all_squash,no_root_squash)
```

on va à présent s'occuper de notre serveur DHCP pour cela nous allons modifier le fichier /etc/dhcp3/dhcpd.conf

```
option subnet-mask 255.255.255.0;
server-name "MyDHCP";
subnet 10.0.2.0 netmask 255.255.255.0;
{
    range 10.0.2.200 10.0.2.300;
}
host Atmel
{
    filename "/tftpboot/tftpboot.img";
    server-name "MyDHCP";
    next-server servername;
    hardware ethernet 00:04:25:1c:86:62;
    fixed-address 10.0.2.20;
}
```

il ne faut pas oublier de configurer eth1 dans /etc/network/interfaces

```
auto eth1
    iface eth1 inet static
        address 10.0.2.10
        netmask 255.255.255.0
        network 10.0.2.0
        broadcast 10.0.2.255
```

On configure les interfaces de la carte dans le fichier /tftpboot/node1/etc/network/interfaces

Dans /etc/default/dhcp3 on spécifie l'interface par défaut du serveur dhcp(*interfaces="eth1"*)

2-Configuration de la carte (U-boot)

Pour joindre la carte nous avons utilisé minicom, qui est un outils de communication par port série. En effet à ce niveau d'avancement seul le port série nous permettait d'interagir avec la carte.

Dans ce but nous devons configurer minicom. Pour ce faire nous avons dû lui indiquer la vitesse de communication (115200 bauds), le port séries avec lequel communiquer (/dev/ttyS1) ainsi que les bits de contrôle et la taille des mots (8N1).

(minicom -s) nous allons modifier le port de communication série (serial port setup), la vitesse de communication (Baud).

Une fois cette opération effectuée nous pouvons commencer à nous occuper de u-boot. Pour ce faire, lors du démarrage de la carte il nous faut, pour accéder à l'interface de configuration du chargeur d'amorçage (bootloader) appuyer sur la touche espace au moment opportun. Ensuite avec une série de commandes listées grâce à "help" nous allons modifier le paramétrage de ce dernier.

Tout d'abord nous avons analysé les variables d'environnement par défaut.

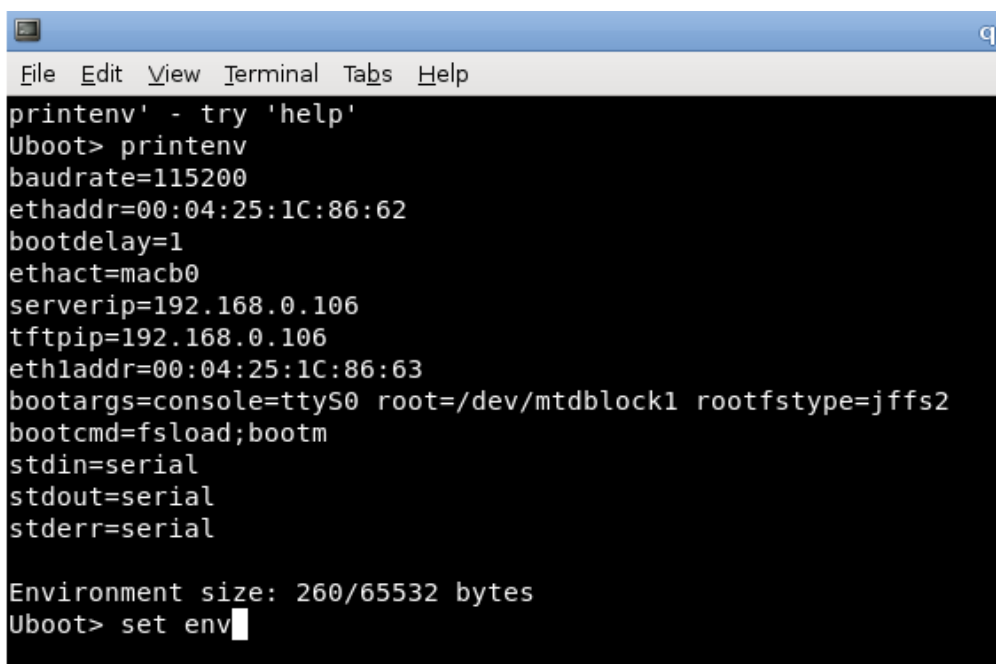
A screenshot of a terminal window titled 'Uboot' with a menu bar containing 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The terminal displays the output of the 'printenv' command, listing various environment variables such as 'baudrate=115200', 'ethaddr=00:04:25:1C:86:62', 'serverip=192.168.0.106', and 'tftpip=192.168.0.106'. At the bottom, it shows 'Environment size: 260/65532 bytes' and the prompt 'Uboot> set env' with a cursor.

figure 5

Nous avons très vite vu que pour un démarrage réseau beaucoup de paramètres étaient à modifier ou à ajouter. Ainsi à l'aide de la commande "setenv" nous avons fait les modifications nécessaires . Voici donc nos nouvelles valeurs.

```
Net: macb0, macb1
Press SPACE to abort autoboot in 10 seconds
Uboot> printenv
baudrate=115200 <- 1 vitesse en baud
ethaddr=00:04:25:1C:86:62 <- 2 adresse mac eth0
ethact=macb0
eth1addr=00:04:25:1C:86:63 <- 2bis adresse mac eth1
tftpip=10.0.02.10 <- 3 adresse IP du serveur tftp
bootdelay=10 <- 4 delais d'attente pour l'accès au bootloader
bootfile=/tftpboot/node1/boot/uImage <- 5 chemin de notre noyau
filesize=131aae
fileaddr=10400000 <- 6 adresse de chargement du système
netmask=255.255.255.0
ipaddr=10.0.2.20 <- 7 adresse ip de la carte
serverip=10.0.2.1 <- 8 adresse ip du serveur
bootcmd=dhcp 10400000 ; bootm <- 9 serie de commandes effectuées au démarrage
bootargs=root=/dev/nfs ip=:::eth0:dhcp nfsroot=10.0.2.10:/tftpboot/node1 console=ttyS0
stdin=serial ^ 10 arguments noyau pour le démarrage
stdout=serial
stderr=serial

Environment size: 543/65532 bytes
Uboot>
```

figure 6

À ce moment précis nous avons rencontré un nouveau problème. En effet le démarrage se retrouve bloqué par un script de démarrage utilisant ifup. Problème venant de notre configuration du fichier /etc/network/interfaces. Le noyau configure lui même les interfaces de la carte grâce aux options présentes dans U-boot ce qui pose des problèmes de conflits. Pour palier à ceci nous avons commenté l'interface eth0 dans notre fichier de configuration.

3.3 Création du système avec buildroot

BuildRoot permet de générer des outils de compilation croisée (toolchain), ainsi que le système de fichiers, l'arborescence et l'image noyau. Ceci est très utile pour travailler sur des systèmes embarqués qui n'utilisent pas de processeurs x86 présents sur les PC classiques.

Notre système ayant pour architecture l'AVR32 la création d'un compilateur adapté est donc nécessaire. BuildRoot va nous permettre de le générer en utilisant la uClibc(~400Ko), qui est une version allégée de la libc(~1700Ko) et donc bien plus adaptée aux systèmes embarqués. De plus il va permettre de simplifier l'installation du système de fichiers et proposer d'implémenter automatiquement des outils tels que busybox.

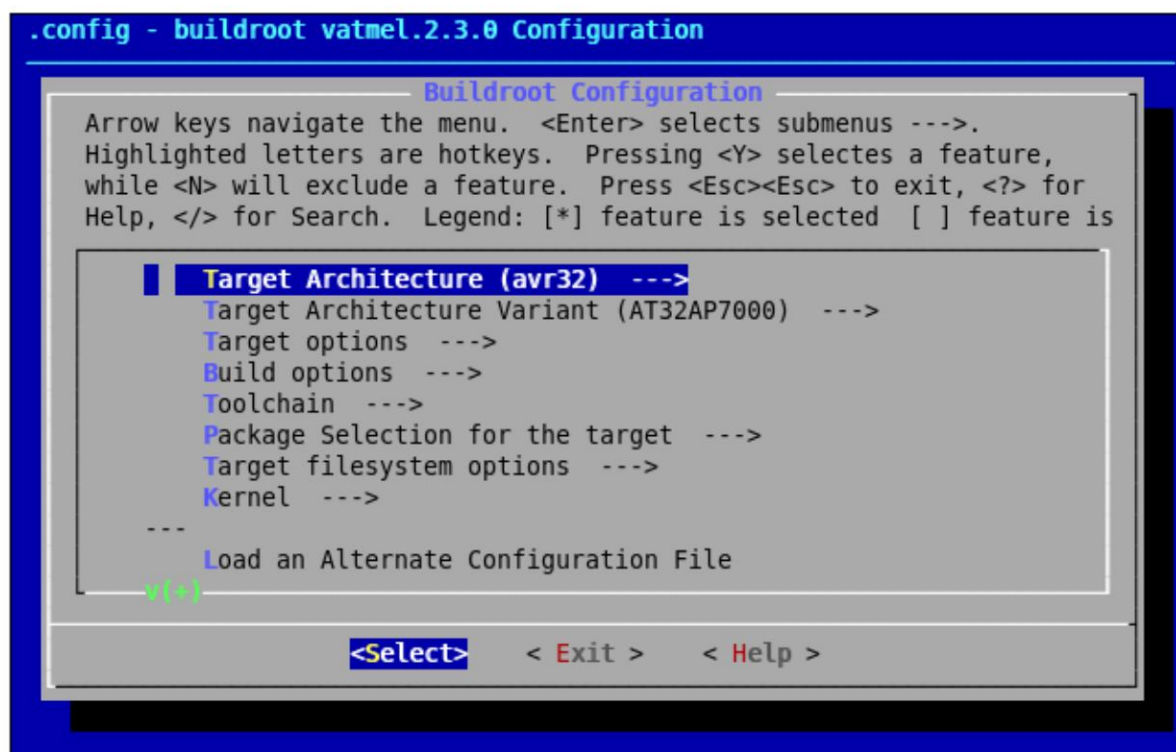
Utilisation de BuildRoot :

BuildRoot dispose d'une interface de configuration, celle-ci est utilisable sans être root.

Pour la lancer :

Mode console (basé sur curses) : `$ make menuconfig`

Mode graphique (basé sur Qt3) : `$ make xconfig`



Mode console de l'interface de configuration figure 7

Une fois que tout est configuré, un fichier .config contenant la description de la configuration est généré. Il sera utilisé par les Makefiles.

Notre configuration :

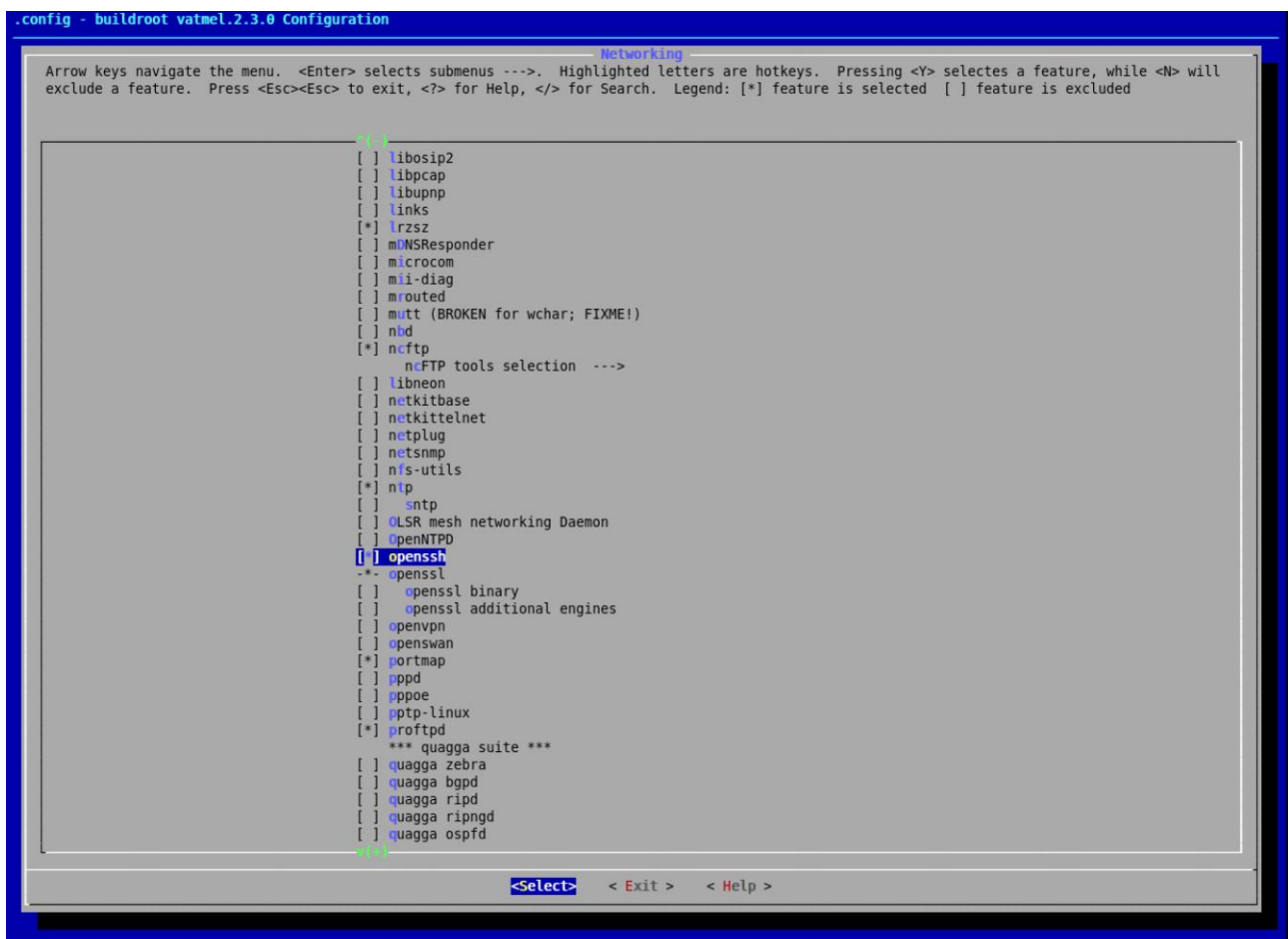
Atmel fournit sur son site une version de BuildRoot permettant de configurer celui-ci pour les cartes qu'il produit (*atstk1002*, *atstk1005*, *atngw100*, *evklcd100*, etc...).

Nous avons donc utilisé la commande : *\$make atngw100_defconfig*

Ceci nous a permis d'ajouter et de supprimer les éléments qui nous intéressaient (binaires, ...) en partant d'une configuration à priori bonne pour la carte ATNGW100.

Le menu de configuration se présente sous forme de cases à cocher, il nous faut donc sélectionner les paquets que l'on veut intégrer à notre Linux embarqué.

Note : il est possible si des paquets ne sont pas proposés de les ajouter manuellement



Sélection des paquets à intégrer figure 8

Nous avons donc vérifié que notre système allait bien contenir nfs, iptables, Dropbear(ssh) et TCPDUMP.

Une fois configuré, il suffit de revenir au menu principal et de sauvegarder la configuration actuelle dans le fichier .config.

Dans le contexte de notre projet, nous ne voulons qu'une seule partition (root). Afin de configurer ceci, il faut se rendre dans le menu de configuration de buildroot et sélectionner une seule partition, comme ceci :

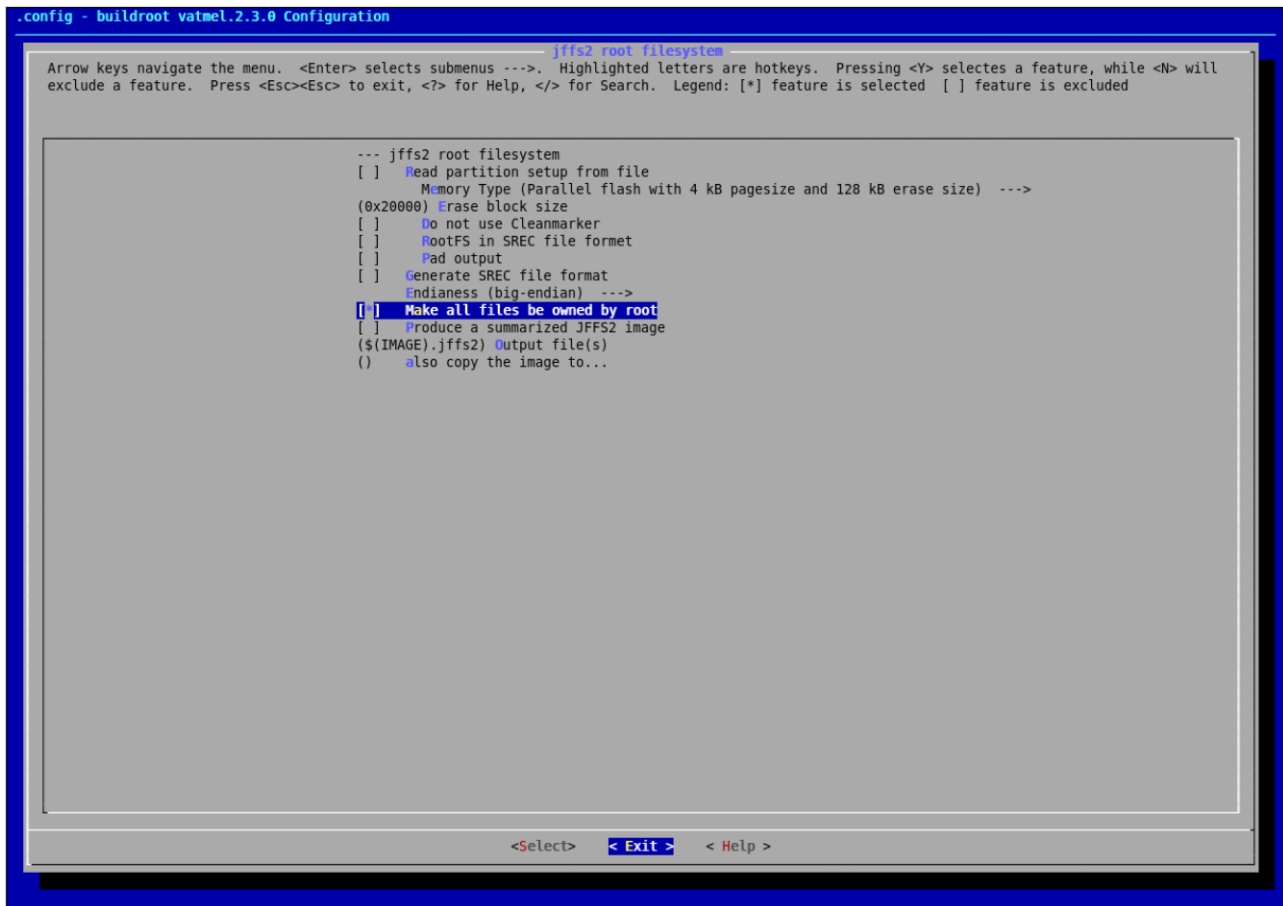


figure 9

En lançant le système sur notre carte, nous avons constaté que les démons telnet et httpd étaient lancés au démarrage (grâce à ps -aux) . Nous avons donc cherché d'où ils venaient car nous ne les avons pas activés dans le menu de configuration de Buildroot. Finalement nous avons trouvé qu'ils étaient gérés par busybox. Il est possible de configurer busybox grâce à un menu de configuration : `$make busybox-menuconfig`

Note : Les fichiers de configuration pris en charge par défaut sont dans le dossier /buildroot-avr32-v2.3.0/target/device/Atmel/atngw100, cependant par défaut le fichier de configuration est généré dans project_build_avr32/atngw100/busybox-1.13.1/ il faut donc le déplacer.

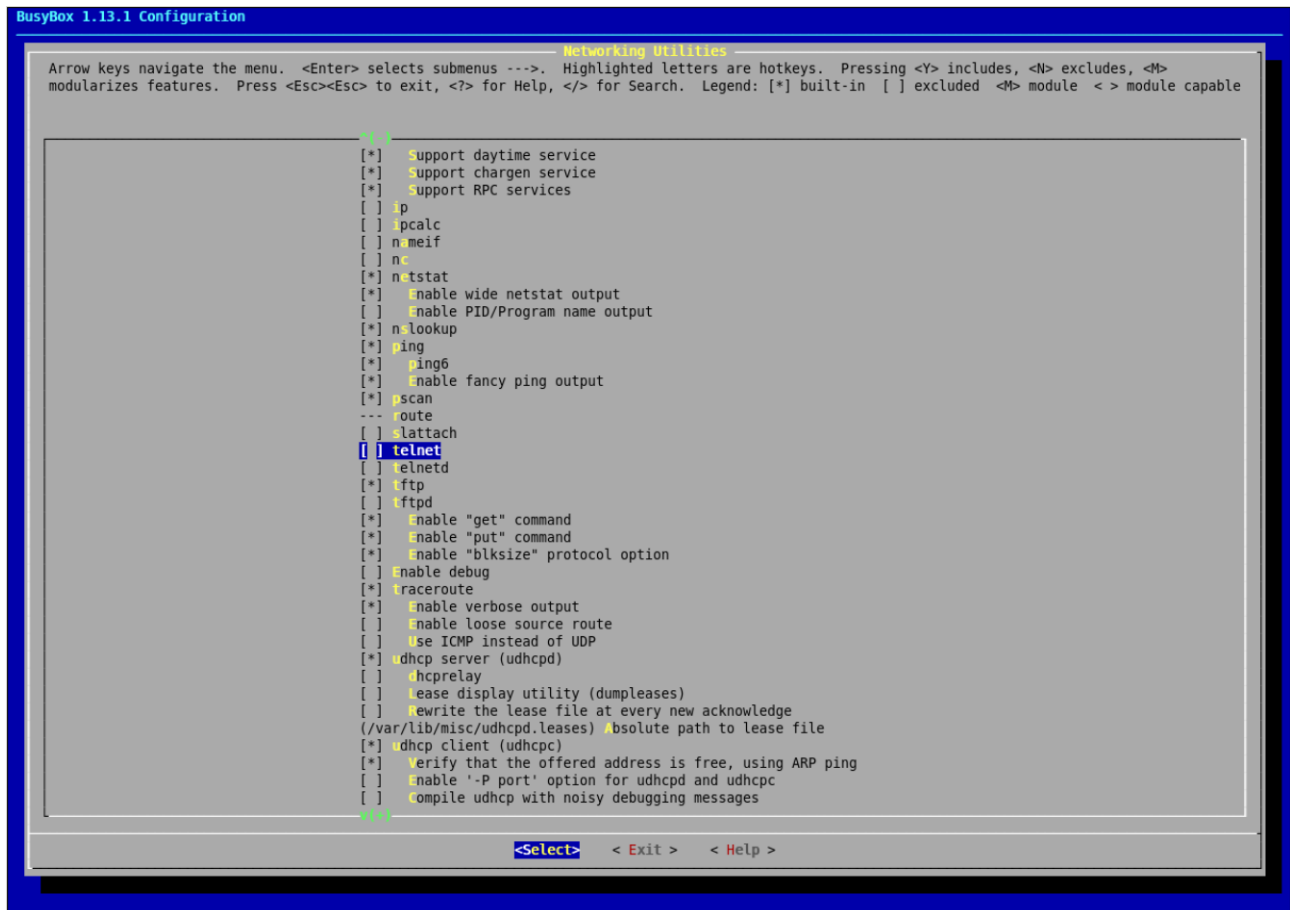


figure 11

Nous avons donc enlevés ces démons car ils ne nous étaient pas utiles dans le cadre de notre projet.

Il est aussi possible de modifier directement le système de fichiers généré grâce au répertoire « project_build_avr32/atngw100/root ». Toute modification apportée dans cette arborescence sera reportée dans le système de fichiers généré après une recompilation.

Note : Il est possible de régénérer totalement l'arborescence et le noyau via les commandes : `$make rootclean $make`. Il est préférable d'utiliser cette commande lorsqu'on retire des paquets, afin de s'assurer de bien supprimer les fichiers correspondants.

Ensuite on tape la commande : `$make source`

Celle-ci permet de télécharger les sources nécessaires aux options que l'on a sélectionnées.

Enfin on fait : `$make`

Afin de lancer la compilation du noyau et la génération des différents outils et système de fichiers.

Après une compilation réussie, il y aura 6 dossiers supplémentaires dans le répertoire de BuildRoot . Voici les plus importants:

binaries/ : Ce répertoire contient les images générées lors de la compilation. Elles sont stockées dans `binaries/<nom du projet>`.

build_avr32/ :Ce répertoire est utilisé pour construire les différentes bibliothèques et applications. Il contient aussi par défaut le répertoire `staging_dir/` qui contient la chaîne de compilation et les bibliothèques.

dl/ :Ce répertoire va contenir toutes les sources téléchargées sous forme d'archives .tar, l'utilisateur n'aura donc à les télécharger qu'une seule fois.

3.4 Configuration du système

Une fois notre système créé grâce à buildRoot et notre serveur tftpbboot mis en place il nous reste encore beaucoup de configuration à effectuer. Dans cette partie nous allons vous exposer toutes les étapes que nous avons suivi pour établir un script lancé au démarrage permettant de configurer le système automatiquement.

1) Configuration de SSH

Notre premier problème fut lors de la génération des clés rsa et dsa. En effet en cours de génération le système s'est bloqué sans nous donner plus d'informations. Pour réussir à obtenir un code d'erreur nous avons modifié le script /etc/init.d/S50dropbear en plaçant un exit 0 au tout début dans l'intention de bloquer son démarrage. Ainsi nous avons réussi à avoir un shell sur notre carte Atmel ce qui nous a permis de laisser les commande de génération (/usr/bin/dropbearkey -t rsa -f /etc/dropbear/dropbear_rsa_host_key) de clé sans redirection des erreurs sur la sortie d'erreur. Nous avons donc découvert qu'il y avait un problème avec /dev/ramdom. Après quelques recherches concernant l'erreur suivante "reading the random source seems to have blocked" nous avons compris que la génération aléatoire de clés rsa ou dsa a besoin de /dev/random pour fonctionner.

En effet /dev/random tout comme /dev/urandom fournissent des interfaces avec le générateur de nombres aléatoires du noyau, ce qui est très utile lorsque l'on a besoin de nombres hautement aléatoires, comme dans notre cas pour générer une clé. Pour fonctionner, il regroupe du bruit* provenant de son environnement et utilise ce dernier pour générer les nombres aléatoires qui seront stockés dans un réservoir d'entropie*. C'est à ce moment que nous avons eu un problème avec /dev/random. En effet si le réservoir d'entropie* est vide ou s'il ne contient pas assez d'informations ce dernier se bloque. Dans notre cas, la carte ne fait pas tourner beaucoup de processus ou de démons donc le bruit capté n'est pas très important et ainsi /dev/random n'a pas assez de données. Pour régler ce problème nous avons fait un lien symbolique de random vers urandom en modifiant le script /etc/init.d/S50dropbear de la façon suivante:

```
# !/bin/bash
...
if [ ! -f /etc/dropbear/dropbear_rsa_host_key ] ; then
    mv /dev/random /dev/chaos
    ln -s /dev/urandom /dev/random
    echo -n "generating rsa key ..."
    usr/bin/dropbearkey -t rsa -f
/etc/dropbear/drop_bear_rsa_host_key > /dev/null 2>&1
    mv /dev/chaos /dev/random
fi
```

En effet /dev/urandom n'est pas bloquant. Si des données lui manquent il les complète avec un algorithme particulier. Avec ces manipulations notre génération de clés rsa et dsa se passe sans soucis mais théoriquement elle sont moins fiables.

Une fois le SSH en place nous pûmes passer à la suite.

2) Configuration de ip_forward et proxy_arp

Pour remplir son rôle, la passerelle doit envoyer sur eth1 ce qu'elle reçoit sur eth0 et vice versa. Pour ce faire nous avons dû modifier deux fichiers noyau de la manière suivante:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
```

3) Configuration de iptables

Notre passerelle doit avoir une fonction de firewall. Chaque accès non autorisé doit être enregistré dans le syslog accessible pas la commande dmesg. Pour ce faire nous avons dû créer différentes règles.

Dans un premier temps nous allons bloquer le ping pour vérifier que notre système de log fonctionne correctement.

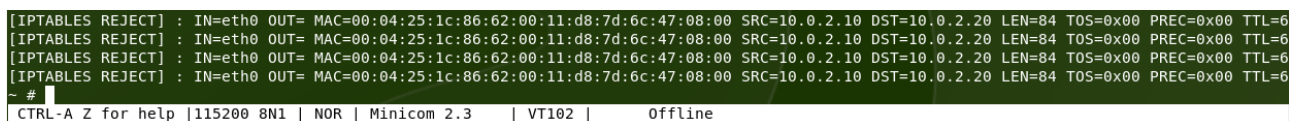
pour ceci voici les règles utilisées:

```
iptables -N LOG_REJECT
iptables -A LOG_REJECT -j LOG --log-prefix '[IPTABLES REJECT] : '
iptables -A LOG_REJECT -j REJECT
```

À présent LOG_REJECT va permettre d'effectuer un REJECT et de loguer tous les messages. Il nous reste donc plus qu'à l'utiliser.

```
iptables -A INPUT -p icmp -j LOG_REJECT
```

Ainsi lorsque qu'un utilisateur extérieur cherche à effectuer un ping de la carte, voici à quoi ressemble notre fichier /var/log/messages (log système)



```
[IPTABLES REJECT] : IN=eth0 OUT= MAC=00:04:25:1c:86:62:00:11:d8:7d:6c:47:08:00 SRC=10.0.2.10 DST=10.0.2.20 LEN=84 TOS=0x00 PREC=0x00 TTL=6
[IPTABLES REJECT] : IN=eth0 OUT= MAC=00:04:25:1c:86:62:00:11:d8:7d:6c:47:08:00 SRC=10.0.2.10 DST=10.0.2.20 LEN=84 TOS=0x00 PREC=0x00 TTL=6
[IPTABLES REJECT] : IN=eth0 OUT= MAC=00:04:25:1c:86:62:00:11:d8:7d:6c:47:08:00 SRC=10.0.2.10 DST=10.0.2.20 LEN=84 TOS=0x00 PREC=0x00 TTL=6
[IPTABLES REJECT] : IN=eth0 OUT= MAC=00:04:25:1c:86:62:00:11:d8:7d:6c:47:08:00 SRC=10.0.2.10 DST=10.0.2.20 LEN=84 TOS=0x00 PREC=0x00 TTL=6
~ #
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.3 | VT102 | Offline
```

figure 12

4) installation de TCPDUMP

Une fois le firewall mis en place nous nous sommes penchés sur l'installation et le fonctionnement de TCPDUMP. A priori aucun problèmes puisqu'à la création du système nous l'avions ajouté. Cependant /usr restait inchangé et ceux malgré la suppression total de telnet et httpd.

Dans un premier temps nous avons pensé à un problème de Buildroot et nous avons donc vérifié plusieurs fois nos fichiers de configuration. Mais, rien à faire, le problème, venait de /ect/fstab. En effet celui-ci étant paramétré pour le système standard, il écrasait notre /usr pour placer celui flashé sur la deuxième mémoire, ce qui par conséquent ne correspondait pas du tout à notre configuration.

5) Configuration des routes

Nous devons dorénavant configurer les routes. Elles permettent d'indiquer aux stations connectées leurs passerelles pour sortir du réseau. Mieux qu'un long discours le schéma ci-dessous expose bien la situation que nous cherchons à atteindre. Pour la suite des opérations nous avons retiré la règle bloquant icmp.

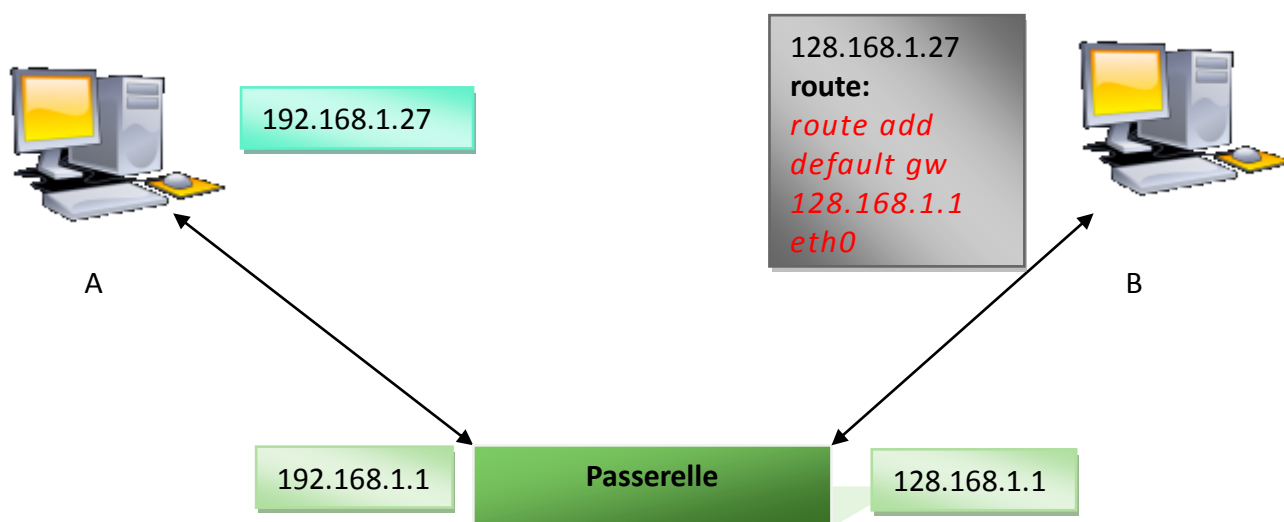


figure 13

Ainsi l'ordinateur A ayant la configuration réseaux suivante:

```
Suffixe DNS propre à la connexion. . . :  
Adresse IPv6 de liaison locale. . . . : fe80::fd29:ac1f:4934:7dab%12  
Adresse IPv4. . . . . : 192.168.1.27  
Masque de sous-réseau. . . . . : 255.255.255.0  
Passerelle par défaut. . . . . : 192.168.1.1
```

figure 14

Arrive à joindre sans problème le pc B

```
C:\Users\leprogr>ping 128.168.1.27  
  
Envoi d'une requête 'Ping' 128.168.1.27 avec 32 octets de données :  
Réponse de 128.168.1.27 : octets=32 temps=1 ms TTL=63  
Réponse de 128.168.1.27 : octets=32 temps=1 ms TTL=63  
Réponse de 128.168.1.27 : octets=32 temps=1 ms TTL=63  
Réponse de 128.168.1.27 : octets=32 temps=1 ms TTL=63  
  
Statistiques Ping pour 128.168.1.27:  
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),  
Durée approximative des boucles en millisecondes :  
    Minimum = 1ms, Maximum = 1ms, Moyenne = 1ms
```

figure 15

Même ssh fonctionne sans problème.

```
quentin@c21-45: ~  
login as: quentin  
quentin@128.168.1.27's password:  
Linux c21-45 2.6.26-2-686 #1 SMP Wed Nov 4 20:45:37 UTC 2009 i686  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
quentin@c21-45:~$
```

figure 16

3.5 Flashage de la carte

Maintenant que notre système est prêt et fonctionnel, nous pouvons le flasher. Comme prévu une seule tentative a suffit pour obtenir un système fonctionnel.

Dans un premier temps nous avons transformé notre arborescence en système de fichier JFFS2.

Ensuite à l'aide de kermi qui est un utilitaire de transfert de fichiers, nous avons commencé la procédure pour flasher la carte.

Dans un premier temps nous avons dû le configurer en modifiant le fichier kermrc. Celui-ci se comporte quasiment comme minicom.

Dans un deuxième temps on lance kermi et on se connecte.

```
kermi  
(/home/user/) c-Kermi>connect
```

Ensuite on redémarre la carte et on entre dans U-boot. Dans un premier temps nous avons enlevé les protections.

```
Uboot> protect off 0X20000 0x7FFFFFFF  
Uboot> erase 0x20000 0x7FFFFFFF
```

A présent nous demandons à U-Boot d'attendre que kermi lui transmette des informations.

```
UBoot> loadb 0x90000000
```

puis avec les touches (`ctrl /-c`) nous mettons la fenêtre de "connexion" en attente pour repasser à celle de kermi. Nous envoyons donc l'image du système de fichier:

```
(/home/user/) c-Kermi> send /tftpboot/node1.img
```

Kermi nous donne alors deux données qui vont nous servir:

```
## Total Size      = 0x00640000 = 6553600 Bytes  
## Start Addr     = 0x90000000
```

Une fois ces données nous nous sommes reconnecté:

```
(/home/user/)c-Kermi>connect
```

Nous avons écrit les données:

```
Uboot> cp.b 0x90000000 0x20000 0x640000
```

et enfin nous avons reprotégé la mémoire.

```
Uboot> protect on all
```


2) Reconfiguration de U-boot

Le flash ne suffit pas à ce que notre système fonctionne. En effet nous avons dû reconfigurer les variables d'environnement de la manière suivante pour que la carte ne démarre plus sur le réseau. En utilisant la méthode précédemment expliquée .

```
Uboot> printenv
baudrate=115200
ethaddr=00:04:25:1C:86:62
ethact=macb0
ethladdr=00:04:25:1C:86:63
bootdelay=3
bootargs=console=ttyS0 root=/dev/mtdblock1 rootfstype=jffs2
bootcmd=fsload 10400000 /boot/uImage;bootm 10400000
stdin=serial
stdout=serial
stderr=serial
```

4.Bilan technique

Le bilan de notre projet est très positif. En effet après une phase de recherche nous avons réussi à mettre en place un environnement de développement réseau nous permettant d'agir rapidement avec la carte.

Nous avons réussi à maîtriser Buildroot afin de créer un système minimaliste d'environ 5,9 Mo fonctionnant sur une architecture AVR32, adoptant l'arborescence et le système de fichier adapté à la carte.

De plus nous avons réussi à créer des scripts de configuration simples à utiliser ce qui permet de changer de système en quelques secondes. Nous avons réglé tous les problèmes posés notamment au niveau de la génération de clés rsa dsa , de la configuration et de l'intégration de iptables.

Enfin nous n'avons eu qu'à effectuer les dernières configurations dues à la migration et à flasher la carte qu'une seule fois comme cela était prévu.

Tous les objectifs ont donc été remplis, notre carte embarque les fonctionnalités de firewall, un TCPCDump et tout ce qui est nécessaire pour enregistrer les logs de Iptables. Le système créé fait 5,9Mo ce qui est très positif par rapport aux 8Mo imposés et démarre en une dizaine de secondes. Elle peut être contactée et utilisée aussi bien par Linux que par Windows et ce sans configurations spécifiques. De plus nous avons développé des scripts qui nous ont permis de mettre en place un nouveau système en quelques secondes.

5.Conclusion

Grâce à ce projet nous avons eût une expérience plus concrète de ce qu'est le travail en équipe. Il nous à permis d'apprendre à nous organiser, à communiquer entre nous et à nous répartir les tâches pour être plus efficace. D'un point de vue technique, ce projet nous a apporté beaucoup de nouvelles connaissances dans le domaine du système embarqué. Nous avons non seulement compris le fonctionnement d'un système Linux de manière générale mais aussi créé et optimisé de nous même un noyau et un système minimaliste, appliqué nos cours de réseau. Ce projet a été très complet, formateur et gratifiant.

6.English Summary

The project that we will introduce to you is supervised by M. DELON David, who is also the final user of our work. Our team is composed of M. CHAPELLE Quentin, M. LAGUET Benoit, M. SABATIER Julien and M. ZEDDA Yannick. The aim of our project was to design a specific Linux operating system for the ATNGW100 board in order to create a gateway. The board on which we had to work, already had an operating system that allows the board to be used like a gateway. But there are some functionalities that our supervisor wanted, so the system that we have to create could be administrated by SSH, used like a firewall and embark a TCP/IP analyser program. Of course, the aim of our project has now been successfully reached.

At the beginning M. DELON helped us to understand what he wanted and also how the board is operating. Indeed, he advised us to use an NFS server and the piece of software named Buildroot. So, in order to achieve our project, we have split our team into pairs to divide up the work. The CHAPELLE Quentin, LAGUET Benoit pair have worked on the deployment of the NFS server that allows us to boot the system on the network and avoid futile flashing of the OS on the board. They also have configured the board to be able to boot on the official operating system for a moment and on our system afterwards. During that time, the SABATIER Julien and ZEDDA Yannick pair have worked on a BuildRoot which can do the cross-compilation that we need to be able to change our OS for the AVR32 architecture, the JFFS2 file system, and the Linux kernel that we need on the ATNGW100 board. When those two phases were finished, we have flashed our OS on the board with the piece of software kermi, and we have set the configuration of the board to use it as a gateway, meaning use of iptables rules, etc... .

During our project, we have encountered some difficulties. Indeed, at the beginning when we configured the board for NFS, the board start-up was blocked by a script that was using ifup. The problem was coming from our interface configuration file. To overcome this problem caused by the kernel which configures the board interfaces, we have just deleted the eth0 interface in our configuration file. The second problem that we have met was on BuildRoot. There were daemons - telnet , httpd- that we didn't want on the system that were installed. With some research, we finally found that those daemons depended on busybox, so we have configured busybox in order to get rid of this problem.

Finally, we have encountered difficulties with the generation of rsa and dsa keys. Indeed the system was blocked on that phase without giving us any information. To bypass that problem, we have changed the script of dropbear to unlock the start-up in order to see error information after. And so, we have found that the problem were /the problem was OR the problems were coming from /dev/random, and the generation of rsa or dsa keys need it.

To conclude, we have reached the aim that M. DELON had fixed for us, which means that we have a gateway that can analyse TCP/IP traffic, act as a firewall, be administered by SSH and in addition our system is smaller than expected.

During that project we have learnt and used a lot of knowledge about embedded systems.

7.Lexique

Bruit: Perturbation aléatoire résultant du fonctionnement du système. Il provient de l'environnement par l'intermédiaire des pilotes de périphériques et d'autres sources.

DHCP: Gère l'attribution et la distribution des adresses IP sur un réseau selon certaines règles pouvant être définies.

DropBear: C'est un client/serveur ssh très léger.

iptables: Logiciel présent dans l'espace utilisateur permettant à l'administrateur réseau de paramétrer les règles du pare-feu netfilter.

Minicom: Application permettant de communiquer via des ports séries.

NFS: c'est un système de fichier distribué qui permet à un utilisateur sur une machine cliente d'accéder, de modifier etc. les fichiers d'une autre machine distante.

Réservoir d'entropie: Espace mémoire plus ou moins grand stockant un nombre aléatoire créé avec le bruit du système. Celui-ci est notamment utilisé par /dev/random et /dev/urandom.

SSH: C'est à la fois un programme et un protocole d'échange de données sécurisé par un système de cryptage rsa ou dsa

TCPDUMP: Utilitaire permettant de surveiller les échanges réseau.

TFTP: Protocole simplifié d'échange de fichiers

Kermit: Protocole de transfert de fichiers avec contrôle d'erreur. C'est aussi un programme faisant de l'émulation de terminal.

Flash: Mémoire se comportant comme de la mémoire RAM mais conservant les données qu'elle emmagasine comme une mémoire ROM.

U-boot: Amorceur de démarrage utilisé sur cette carte.

8. Sources

2.2 Présentation de la carte ATNGW100

- www.atmel.com/

2.3 La mémoire et son système de fichier

- fr.wikipedia.org/wiki/Mémoire_flash

- www.avrfreaks.net/wiki/index.php/Documentation:NGW/NGW100_Memories

- linuxfr.org/2008/04/04/23938.html

- fr.wikipedia.org/wiki/JFFS2

2.4 Alternative aux flashage

- lionel.tricon.free.fr/Documentations/Linux/linux-diskless.html

- www.debian.org/releases/stable/mipsel/ch04s03.html.fr

- www.lri.fr/~quetier/tuto_cluster/cluster

- www.pcinpact.com/forum/sujet_26603.htm

2.5 Cross-compilation et intégration

- fr.wikipedia.org/wiki/compilateur

3.2 Mise en place du serveur tftp

- www.lri.fr/~quetier/tuto_cluster/cluster

- www.pcinpact.com/forum/sujet_26603.htm

3.4 Configuration du système

- pwet.fr/man/linux/fichiers_speciaux/random

3.5 Flashages de la carte

- <http://www.denx.de/wiki/view/DULG/SystemSetup>

9. Annexes

Script de démarrage S50Dropbear: (dans se script on ajoute le lien symbolique a urandom)

```
#!/bin/sh
#
# Starts dropbear sshd.
#

# Make sure the dropbearkey program exists
[ -f /usr/bin/dropbearkey ] || exit 0
start() {
    echo -n "Starting dropbear sshd: "
    # Make sure dropbear directory exists
    if [ ! -d /etc/dropbear ] ; then
        mkdir -p /etc/dropbear
    fi
    # Check for the Dropbear RSA key
    if [ ! -f /etc/dropbear/dropbear_rsa_host_key ] ; then
        mv /dev/random /dev/chaos
        ln -s /dev/urandom /dev/random
        echo -n "generating rsa key... "
        /usr/bin/dropbearkey -t rsa -f
        /etc/dropbear/dropbear_rsa_host_key > /dev/null 2>&1
        mv /dev/chaos /dev/random
    fi

    # Check for the Dropbear DSS key
    #if [ ! -f /etc/dropbear/dropbear_dss_host_key ] ; then
    #    echo -n "generating dsa key... "
    #    /usr/bin/dropbearkey -t dss -f
    /etc/dropbear/dropbear_dss_host_key > /dev/null 2>&1
    #fi
    umask 077
    start-stop-daemon -S -q -p /var/run/dropbear.pid --exec
    /usr/sbin/dropbear
    echo "OK"
}
stop() {
    echo -n "Stopping dropbear sshd: "
    start-stop-daemon -K -q -p /var/run/dropbear.pid
    echo "OK"
}
restart() {
    stop
    start
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart|reload)
```

```

        restart
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart}"
        exit 1
esac

exit $?

```

Script S51init_Atmel: (script de paramétrage spécifique à notre projet)

```

#!/bin/sh

IP_Table () {

    echo 1 > /proc/sys/net/ipv4/ip_forward
    echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp
        echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
        echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp
    echo "Reset des regles de filtrages"
    iptables -F FORWARD
    iptables -F INPUT
    iptables -F OUTPUT
    iptables -F LOG_REJECT
    iptables -X LOG_REJECT

    echo "Parametrage des log"
    iptables -N LOG_REJECT
    iptables -A LOG_REJECT -j LOG --log-prefix '[IPTABLES REJECT] : '
    iptables -A LOG_REJECT -j REJECT

    echo "parametrage de FORWARD"
    iptables -A FORWARD -j ACCEPT
#    iptables -A INPUT -j ACCEPT
#    iptables -A OUTPUT -j ACCEPT
#    iptables -A INPUT -p icmp -j LOG_REJECT
#    iptables -A INPUT -p tcp --dport 22 -j ACCEPT
#    iptables -A INPUT -j REJECT

}

echo "lancement init_Atmel"
IP_Table
echo "init_Atmel ok"

```

Script lancer de notre poste de développement pour compléter le système monté en NFS:

```

#!/bin/sh

cp -R /source/S50dropbear /tftpboot/nodel/etc/init.d
cp -R /source/S51init_Atmel /tftpboot/nodel/etc/init.d
cp -R /source/interfaces /tftpboot/nodel/etc/network

```