CREATION OF A MINIMAL STAND ALONE RTAI SYSTEM

Requirements ::

* I prepared my stand alone RTAI for the following hardware configurations.
* Intel D945GCLF Atom Mini-itx board
* 512MB RAM
* 128MB Transcend DOM (Disk on Module)
* I tried this stand alone RTAI from Ubuntu 8.04 2.6.24 kernel version.

STEP 1 - Configure and compile a reduced version of your Kernel

This step assumed that, you already downloaded the kernel; Downloaded the RTAI;Unzip the files; And apply the RTAI patch to Linux kernel. Please be ensure that, you have already done the steps from STEP:1 to STEP:4 in the file RTAI-TARGET-HOWTO.TXT.

Now we are again compile the kernel. Coz we need to customize more specific for our target board.

The first thing to do is to compile the kernel that fits the target system, as far as processor type, bus and peripherals are concerned.
As root:

```
$ cd /usr/src/linux
$ make xconfig (or menuconfig or config)
```

If you want a reduced version of the kernel,
e.g. size less than 600K, you have to disable virtually everything
except the loadable module support, the IPIPE option,
the ISA bus, and the IDE, floppy, ext2, and /proc support.

Also, at this point, if you are really working with a floppy disk,
and you need an unattended reboot, you should manually edit the file
/usr/src/linux/init/do_mounts.c, search for the "press ENTER" string
and, a few lines below, change the instruction
sys_read(fd, &c, 1); with the instruction sys_read(fd, &c, 0);

Once you have done the above, you have to compile the kernel:

```
$ make clean
$ make
```

copy the configuration file for future reference :

```
$ cp -f .config config-linux.txt
```

STEP 2 - Recompile RTAI for the new kernel

RTAI needs to be recompiled to take into account both the newer
kernel and possible different settings of the target machine,
so, as root:

```
$ cd /usr/src/rtai
$ make xconfig (or menuconfig or config)
$ make clean
$ make
```

```
    $ make install

    and copy the configuration file for future reference :

    $ cp -f .rtai_config config-rtai.txt

    It is a good idea at this point to recompile the code
    from the simulink schemes that need to run on the target machine.


STEP 3 - Download, unpack and install Busybox
---------------------------------------------

    Go to http://busybox.net/downloads/ and get version 1.01,
    version 1.1.13 is ok but is slightly different
    however later version do not compile statically with gcc,
    so it's better to avoid them and just get the 1.01 version.

    As root, download the busybox-x.xx.tar.bz2 file into /usr/src
    ans unpack it using the following commands :

    $ cd /usr/src
    $ tar -xjvf busybox-x.xx.tar.bz2

    Set a symbolic link for later use:

    $ rm -f busybox
    $ ln -fs busybox-x.xx busybox

    then, configure and install busybox:

    $ cd /usr/src/busybox
    $ make config (or make menuconfig)

    In the Build Options, select "Build Busybox as a static binary".
    Be sure to include at least the init, insmod, rmmod, and mount,
    in general, select to only what you need, if unsure, use the default option.
    In version 1.1.13 be sure to set the PREFIX option equal to /mnt/f.
    After the configuration, as root:

    $ make dep
    $ make busybox

    finally, save the config file for future reference:

    $ cp -f .config config-busybox.txt



STEP 4 - Download, unpack and install LILO
---------------------------------------------

    Its time to insert your DOM in your HOST PC. Switch-off your PC.
    I assume that you are running your host Linux from SATA hard disk.
    So atleast you should have one IDE port is free now.
    Now Insert the DOM in your free IDE port.

    GRUB or LILO
    ------------
    Its upto your wish to use. I am not comfortable with LILO. So I
    used GRUB. But here I have given the steps for both LILO & GRUB.
    So please install either GRUB or LILO boot loader.
```

GRUB installation

You can use grub-install command to install GRUB on your DOM from
your Ubuntu Linux.

```
# mke2fs /dev/sda
# mount -t ext2 /dev/sda /mnt
# grub-install --root-directory=/mnt sda
# umount /mnt
```

If you installed GRUB, then skip the next step, LILO installation.
If you dont like GRUB and wish to install install LILO, then continue.

LILO installation

Go to http://freshmeat.net/projects/bin86/ and look for the
latest version of the bin86 assembler and loader, (i use 0.16.17).

As root, download the bin86-x.xx.xx.tar.gz file into /usr/src
ans unpack it using the following commands

```
$ cd /usr/src
$ tar -zxvf bin86-x.xx.xx.tar.gz
```

Then compile the bin86 assembler:

```
$ cd /usr/src/bin86-x.xx.xx
$ mkdir /usr/local/man/man1
$ make && make install
```

If everything goes fine, then it's time to download LILO:
go to http://www.t2-project.org/packages/6.0/lilo.html and look for the
newest stable version of the lilo, (i use version 22.7.1).

As root, download the lilo-xx.x.x.src.tar.gz file into /usr/src
ans unpack it using the following commands :

```
$ cd /usr/src
$ tar -zxvf lilo-xx.x.x.src.tar.gz
```

Set a symbolic link for later use:

```
$ rm -f lilo
$ ln -fs lilo-xx.x.x lilo
```

then, build lilo (you don't have to install it on the system as a
boot loader, you just need to have the /usr/src/lilo/lilo binary)

```
$ make install
```

STEP 5 - Prepare DOM

In what follows, it is assumed that you want to create a
stand alone RTAI system on a DOM. This is rarely useful,
since you can't normally include anything else other than

the bare essential, however, if you learn how to squeeze Linux
on a DOM then you can later put it wherever you want,
using exactly the same procedure with just a few changes.
For example if the destination is a floppy mounted
as /dev/fd0 you can replace the /dev/sda string
with /dev/fd0 in the following instructions.

unmount the DOM :

```
$ umount /dev/sda
$ sync
```

zero out the content of the DOM block by block:

```
$ dd if=/dev/zero of=/dev/sda bs=1MB count=128
```

create an ext2 file system on the DOM:

```
$ /sbin/mke2fs -F -m 0 /dev/sda
```

create a temporary folder if it does not exist,
and remove everything if it exists

```
$ mkdir -p /mnt/f
$ rm -drf /mnt/f/*
```

mount the DOM on /mnt/f :

```
$ mount -t ext2 /dev/sda /mnt/f
$ sync
```

STEP 6 - Populate the root

Again as root, create the classic unix directories :

```
$ cd /mnt/f
$ mkdir boot
$ mkdir initrd
$ mkdir dev
$ mkdir proc
$ mkdir etc
$ mkdir sbin
$ mkdir bin
$ mkdir lib
$ mkdir mnt
$ mkdir usr
$ mkdir tmp

$ mkdir lib/modules
$ mkdir -p var/{log,run}
$ touch var/run/utmp

$ sync
```

STEP 7 - Populate /dev

```
    $ cp -dpR /dev/hda dev
    $ cp -dpR /dev/hda[0-3] dev
    $ cp -dpR /dev/hdb dev
    $ cp -dpR /dev/hdb[0-3] dev
    $ cp -dpR /dev/sda dev
    $ cp -dpR /dev/sda[0-3] dev
    $ cp -dpR /dev/sdb dev
    $ cp -dpR /dev/sdb[0-3] dev
    $ cp -dpR /dev/tty[0-6] dev
    $ cp -dpR /dev/ttyS[0-6] dev
    $ cp -dpR /dev/ram[0-6] dev
    $ cp -dpR /dev/console dev
    $ cp -dpR /dev/zero dev
    $ cp -dpR /dev/null dev
    $ cp -dpR /dev/kmem dev
    $ cp -dpR /dev/mem dev

    $ mkdir dev/rtf
    $ cp -dpR /dev/rtf/[0-9] dev/rtf
    $ cp -dpR /dev/rtf[0-9] dev
    $ cp -dpR /dev/rtai_shm dev

    $ sync
```

If you are creating a bootable flashcard that is seen as, say,
/dev/sdc on the development system and it will be seen as, say,
/dev/hda on the target system, make sure that both
/dev/sdc and /dev/hda are copied to the /mnt/f/dev folder.
If necessary you need to copy these device files from other system,
but they have to be there otherwise step 10 will fail.

STEP 8 - Create the configuration files and populate /etc

I suggest that you create the folder /usr/etc/f
to store the configuration files for your system.

There are 2 very important configuration files that the
system needs to find during the boot process, that is
/etc/fstab and /etc/init.d/rcS :

/etc/fstab

This is the file that tells the filesystems to mount
and where they are located, usually just a few lines are required:

```
    /dev/sda       /           ext2    defaults   1 1
    none           /proc       proc    defaults   0 0
```

If the kernel does not support the /proc filesystem then
just the first line is necessary.

/etc/init.d/rcS

This file (rcS stands for Run Commands Shell) contains
command to be executed by the shell as soon as it starts.
here is an example:

```
#!/bin/sh

/bin/mount -av

sync
insmod /lib/modules/rtai_hal.ko
insmod /lib/modules/rtai_lxrt.ko
insmod /lib/modules/rtai_fifos.ko
insmod /lib/modules/rtai_sem.ko
insmod /lib/modules/rtai_mbx.ko
insmod /lib/modules/rtai_msg.ko
sync

# insert your calls here, e.g:
# /tmp/test -v -f 10

# shut down the system
#poweroff
```

lilo.conf

This part is only needed for those who are using LILO.
If you are using GRUB, please skip this lilo.conf section.

The lilo configuration file it is not needed during the
boot process but it is needed later to properly write the
DOM boot sector:

Here is an example of lilo.conf file for a stand alone
system with both kernel and file system on the same DOM :

```
boot    =/dev/sda
map     =/boot/map
timeout =00
read-write
backup  =/dev/null
compact
image   =/boot/bzImage
label   =RTFLASH
root    =/dev/sda
initrd  =/boot/initrd.img-2.6.24
```

It is a good idea to also store the configuration files:

```
$ cp -f /usr/src/linux/.config /usr/etc/f/config-linux.txt
$ cp -f /usr/src/linux/.config /usr/etc/f/config-rtai.txt
$ cp -f /usr/src/busybox/.config /usr/etc/f/config-busybox.txt
```

At this point we can move the files into the mnt/f/etc folder:

```
$ cd /mnt/f
$ cp /usr/etc/f/* etc
```

    It is always better to make sure that rc is executable:

```
    $ chmod -R 777 etc/*
    $ sync
```

    Finally, if you modified the code of either the linux kernel
    or busybox, it is a good idea to include it here in this folder.

STEP 9 - Populate /bin /sbin and /usr using busybox

    As root, do the following :

```
    $ cd /usr/src/busybox
    $ make PREFIX=/mnt/f install
    $ sync
```

STEP 10 - Create RAM DISK, Copy the kernel and make the disk bootable

    On Ubuntu, you have to create the initial ram disk with the command:

```
    $ update-initramfs -ck x.x.xx
```

    where x.x.xx is the directory under /lib/modules that contains
    the modules for the new kernel.

```
    $ cp /usr/src/linux/arch/x86/boot/bzImage /mnt/f/boot/bzImage
    $ cp /boot/initrd.img-2.6.24 /mnt/f/boot/
    $ cp -dpR /boot/System.map-x.x.xx /mnt/f/boot/
    $ sync
```

    If you are using GRUB, then follow GRUB user's and skip LILO user's section.

GRUB user's

    Now you have to edit the /mnt/f/boot/grub/menu.lst, comment the
    "hiddenmenu" option, set the timeout to something like 10 seconds,
    and physically insert the lines relative to the new kernel,
    resulting in something like this:

```
    title       RTFLASH
    root        (hd0,0)
    kernel      /boot/bzImage ro root=/dev/sda
    initrd      /boot/initrd.img-2.6.24
```

LILO user's

```
    $ /usr/src/lilo/lilo -v -C /etc/lilo.conf -r /mnt/f
    $ sync
```

STEP 11 - Copy rtai related stuff

I hope you have more space left in your DOM. Its better to copy the RTAI and Linux related stuffs from your host to DOM.
It is good, if you will copy the necessary file, libraries, modules from /lib folder from your host Linux.

I copied all the files available in the folder /lib from my host to /mnt/f/.
So the size was ~100MB. After I confirmed that my DOM booted successfully, then I removed the unwanted libraries, modules from the folder /mnt/f/lib/.

Or If you are really working with a standard floppy disk then at this point there will probably be only about 100K left on your floppy disk so, there will be no space left to copy any modules or executables, except maybe for a few ones, so, in that case you can skip this step.

All the rtai modules and the executables that need to run on the target system, have to be compiled for the target system, therefore make sure that rtai has been compiled and installed using the same linux .config file that was used to compile the kernel for the target processor in step 1.

Then,

```
$ cd /mnt/f
$ cp /usr/realtime/modules/rtai_hal.ko lib/modules
$ cp /usr/realtime/modules/rtai_lxrt.ko lib/modules
$ cp /usr/realtime/modules/rtai_fifos.ko lib/modules
$ cp /usr/realtime/modules/rtai_sem.ko lib/modules
$ cp /usr/realtime/modules/rtai_mbx.ko lib/modules
$ cp /usr/realtime/modules/rtai_msg.ko lib/modules
$ sync
```

Also, allow the tmp directory to be accessible, if you want to copy some file over there.

```
chmod -R 777 /mnt/flash/tmp
```

At this point, you should recompile the executables that need to run on the target system, and copy them on the tmp directory along with any other needed file.

STEP 12 - Check size, unmount and reboot

```
$ df -h /dev/sda
$ sync
$ sync
$ umount /dev/sda
```

If you want to test your DOM from your host, then set your bios that, you are booting from DOM first instead of your SATA hard disk.

Or You can directly test the DOM from target board.
now take your device, insert it into the target system, reboot and cross your fingers really hard :-)

If you have any issues, you can reach me by asprakash83 at gmail dot com
More steps are taken from the document "RTAI-TARGET-HOWTO.txt" prepared by
   Giampiero Campa. You can reach him by campa at cemr dot wvu dot edu