Problem:

An application needs to boot from flash disk with a full POSIX compliant real time operating system.

Solution:

If your application requires this type of functionality, it can be achieved by using QNX real time operating system. See the following.

WHAT IS AN EMBEDDED SYSTEM

Often in today's control market, computers are referred to as an embedded computer. Even though this terminology is commonly used, there is still some confusion about what comprises an embedded computer. An embedded computer is basically the same as a desktop with some significant differences.

The first difference is that an embedded system generally does not use a hard drive, a SCSI drive, or another form of non-volatile device for its secondary storage. Embedded systems generally use what is called a solid state drive or SSD to store the operating system and the applications software.

The second difference is that an embedded computer does not necessarily have to have a keyboard or a video monitor attached to the system. Although most embedded computers support a keyboard and a monitor, these functions are also provided through one of the serial ports. This allows a system to be configured and run by connecting a terminal or another computer to the serial port.

The third difference is that an embedded system is usually smaller than a desktop computer. This allows the system to be buried, or embedded, in a cabinet or a piece of equipment. This means that although the computer is providing some control or data gathering functions, it is not obvious that there is even a computer installed in the equipment. As a matter of fact, some system operators are often unaware that when they enter information on a keypad or read information from an LCD display, this is in many respects similar to the computer they have on their desk at home.

WHY USE AN EMBEDDED COMPUTER

If an embedded computer is so similar to a desktop, why use one at all? Using an embedded computer in a system gives several advantages over a desktop computer. The first advantage is that embedded computers are designed to survive in harsh environments. The second advantage is the ability to operate without a hard drive or other type of mechanical device which makes these systems more reliable. The third advantage is the small size of an embedded computer allows the system to be implemented in less space, saving money by requiring less room in the enclosure used.

WHICH OPERATING SYSTEM MAKES SENSE

This question can only be answered by the people doing the system design. Octagon provides DOS as the basic operating system for its embedded computers. DOS is provided with the CPU in the BIOS EPROM. For many applications, DOS is more than adequate. The advantages of using DOS in a system include a wide base of existing software, extensive user knowledge; most personal computers have DOS installed on them, and the programming environment is fairly simple.

Sometimes the application requires a higher performance level than can be achieved with DOS. It is in these instances that the flexibility of an Octagon system can be utilized. Because Octagon CPUs are AT compatible systems, the user is not locked into a particular operation system. The user can select any number of operating systems that will run successfully on an Octagon CPU. One of the most popular operating systems other than DOS that is used in embedded systems is QNX. QNX is a POSIX compliant real time operating system that is designed to be used in high speed and critical control applications. A significant advantage using QNX is the support for embedded systems found in QNX. Octagon currently has several CPUs that can be used to provide an embedded system running QNX. These CPUs are the 5025A, the PC-450, the 7004 and the 5066. The 5025A and the PC-450 embedded file system is currently provided with the Embedded Kit that can be purchased from QNX. The 5066 and the 7004 embedded file system can be obtained from Octagon Systems.

HOW QNX IS EMBEDDED ON AN OCTAGON CPU

QNX is implemented in an Octagon CPU as an embedded system by placing the QNX file system on a solid state disk. This disk is then selected as the bootable drive and the QNX operating system will run when the CPU is powered up.

When a user selects QNX as the operating system for an embedded design, the utilities and file system managers are provided to allow QNX to be configured to support a file system on an SSD. The actual QNX operating system must be purchased from QNX. Octagon, although familiar with QNX, is not a source for the QNX operation system.

The minimum system requirements for the QNX operating system is an Intel 386 compatible processor, and a CPU that has an embedded file system manager developed for it. The software requirements are QNX version 4.22 or greater.

SSDS AND FLASH CHIPS

A Solid State Disk (SSD) is a memory device used to store data just like a hard drive or a floppy drive on a desktop computer. The SSDs on an Octagon CPU can be one of four types. They can be EPROM, flash, SRAM, and DRAM. These SSDs are designated as SSD0, SSD1, and SSD2. These designators define a physical location on the CPU board. A 5025A for example has three SSDs. SSD0 is the EPROM that contains the BIOS and the DOS drive. SSD1 can be an EPROM or flash memory, and SSD2 can be flash memory, an EPROM or an SRAM. The DRAM can be

configured as an additional RAM drive if needed. The user is able to define which SSD is used as the boot device by making an entry into the setup parameters with a utility called setup. This allows the user to boot from the BIOS EPROM or a flash SSD that contains the application program. When an alternate operating system is used this operating system is usually placed in the flash SSD.

The SSDs on an Octagon CPU are mapped into a 32K or a 64K window in memory. The size of the window depends upon the CPU being used. The window size for a particular CPU can be found in the user's manual for that CPU. Flash memory is the most commonly used device in embedded computers. Most of the Octagon systems take advantage of flash memory for storage of the operating system and the applications programs.

QNX AND SSDS ON OCTAGON CPU CARDS

SSDs are referenced in QNX as sockets. This socket represents the logical location of the solid state device. The SSDs are further divided physically into contiguous regions. These regions are composed of memory devices that are the same type. The regions can be further divided into multiple partitions. A partition must be contained within a single region. This means that if you define a partition on a device that has 512k regions, this partition must be less than or equal to 512K.

There are two types of partitions supported by the QNX embedded file system. The first type is the image file system partition and the second type is the embedded file system partition. The image partition allows the SSD to be bootable. This partition stores the boot image for the system. The embedded file system partition stores a QNX compatible file system. For example, the /bin and the /etc/config directories would be placed in this partition. The embedded file system partitions functionality depends upon the type of device it is implemented on. If it is placed on an EPROM, then it is read only. If it is placed on a flash that has an EFSYS file that supports read and writes, it will be a read/write partition.

USING THE EFSYS FILE SYSTEM COMMAND

The embedded file system manager is a stand-alone process similar to the FSYS file manager. It is included when the boot image is built. If the system already has a card information structure (CIS) on the SSD, the EFSYS file system manager will read this from the SSD when the file system manager is started. If there is not a CIS on the SSD, then the file system manager needs to be started with the correct parameters in its command line. There are several parameters that need to be passed to the EFSYS from the command line when it is first started. These are:

- socket number
- Jedec number
- size of the device
- block size
- offset

An example for a 5025A CPU with a 256K flash in SSD1 and a 512K SRAM in SSD2 would be:

```
efsys.5025A -r1,89BD,256k,256k -r2,0,512k,64k
```

This tells the EFSYS that it has a 256K flash as socket one that has a block size of 256k with a Jedec id of 0x89BD. It also tells the EFSYS that there is a 512K SRAM in socket 2 that has a 64K block size. The zero in the SRAM portion tells the EFSYS not to worry about a Jedec id number. The syntax used above only needs to be used when there is no formatted SSD present. If the SSD is already formatted, the EFSYS file system manager can be started with the following command:

```
Efsys.5025A &
```

MOUNTING POINTS FOR AN EFSYS

The standard mount and unmount commands used in QNX to change mount points in a file system cannot be used with the EFSYS. To specify mount point of an EFSYS, the mount point must be defined in the build file used to build the file system image. If the system was to be mounted from the root directory of the EFS partition you would put the following command line in the build file:

```
/bin32/Efsys.5025A      $8000 Efsys.5025A -m/efs
```

This will cause the system to mount from the root of the EFS partition in the SRAM.

CREATING A BOOTABLE SSD

There are eight major steps to creating a bootable SSD on an Octagon CPU.

1. Create a disk boot image using the BUILDQNX command.
2. Create an embeddable boot image using the ROMQNX command
3. Start the EFSYS."OCTAGON CPU" file system manager.
4. Erase or initialize the SSD using the EFSINIT command.
5. Partition the SSD using the MKCIS command.
6. Format the image partition and install the boot image using the MKIMAGE command
7. Format the file system partition using the EFSINIT command
8. Copy the correct files and directory structure to the EFS partition using the CP or the CPBE commands.

The following examples can be used to create a basic embedded file system on the 5025A CPU, the PC-450 mobile industrial computer, and the 5066 CPU. The complete directory structures and the files used for these examples can be found on the Octagon BBS in the "downloads" conference. They are listed in the following format:

```
"CPU NUMBER".TAR
```

5025A CPU EXAMPLE

The following is an example that will make an embedded file system on a 5025A CPU using a 256K flash and a 512K SRAM. The flash will store the boot image. The SRAM will store the file system. The files for this example can be found in the /usr/EKit/examples/5025A subdirectory.

There are several files that need to be examined. The first two are the FLASH.INFO and the SRAM.INFO files. These files help the MKCIS command define what the SSD devices look like. It defines the size, the Jedec number, the boot file used, and the partition type and size. These files are the input to the MKCIS command in the MAKEFILE used to make the flash partitions. Following are the FLASH.INFO and the SRAM.INFO files used with a 5025A CPU, a 256K flash disk, and a 512K SRAM disk.

FLASH.INFO

```
#
#     Info file for Octagon Control Card 5025
#
#     Socket 1 (SSD1)   -     Intel/AMD 28F020 (256kx8 flash)
#           This socket contains a single image file system containing
#           the OS boot image.
#
region
      size 256k
      jedec 0x89bd
      boot_file "/boot/sys/boot.5025a"

      partition
            type image
            attribute largest
```

```
SRAM.INFO
#
#     Info file for Octagon Control Card 5025
#
#     Socket 2 (SSD2)   -     512k SRAM
#           This socket contains a single EFS file system containing QNX
#           executables (non-bootable)
#
region
      size 512k
      type sram

      partition
            type efs
            attribute largest
```

The next file that needs to be examined is the MAKEFILE used to create the EFSYS partitions that are defined in the *.INFO files. This file is usually found in the sub-directory that holds all the information needed to make an EFSYS system for the CPU being used. For example, we keep files for creating a 5025A EFSYS system in the users/EKit/examples/5025A sub-directory. To run this script you need to use the

GMAKE command. It will run the MAKEFILE found in the directory that it resides in and do what is specified in that MAKEFILE.

MAKEFILE

```
#
#     Makefile for bootable Octagon 5025a
#
#     To make a bootable EPROM image suitable for SSD1
#           Efsys.file -f/tmp/rom -s1 &
#           make rom
#           <burn EPROM>
#
#     To make a bootable system with 28F020 in SSD1 and 512k SRAM in SSD2
#           make flash
#           make sram
#

NODE        = 1

FLASH_RAW   = //$(NODE)/dev/skt1
FLASH_IMG   = //$(NODE)/dev/skt1img1

SRAM_RAW    = //$(NODE)/dev/skt2
SRAM_EFS    = //$(NODE)/efs2p1

ROM_RAW = /dev/skt1
ROM_IMG = /dev/skt1img1
ROM_EFS = /efs1p1

BOOT_IMAGES = /boot/sys

BOOTFILE    = $(BOOT_IMAGES)/boot.5025a

EFS_FILES   =/bin/login /bin/sh /bin/tinit /bin/cat /bin/ls       \
            /bin/sin /bin/shutdown /bin/Dev /bin/Dev.con /bin/rtc


# To make a 1MB rom image for SSD1
rom : $(BOOTFILE) rom.info os.rom.image
      mkcis -o$(ROM_RAW) rom.info
      cat os.rom.image >$(ROM_IMG)
      efsinit $(ROM_EFS)
      mkdir $(ROM_EFS)/bin
      mkdir $(ROM_EFS)/etc
      mkdir $(ROM_EFS)/etc/config
      cbpe -f /bin/login $(ROM_EFS)/bin/login
      cbpe -f /bin/sh $(ROM_EFS)/bin/sh
      cbpe -f /bin/tinit $(ROM_EFS)/bin/tinit
      cbpe -f /bin/cat $(ROM_EFS)/bin/cat
      cbpe -f /bin/ls $(ROM_EFS)/bin/ls
      cbpe -f /bin/sin $(ROM_EFS)/bin/sin
      cbpe -f /bin/shutdown $(ROM_EFS)/bin/shutdown
      cbpe -f /bin/Dev $(ROM_EFS)/bin/Dev
      cbpe -f /bin/Dev.con $(ROM_EFS)/bin/Dev.con
      cbpe -f /bin/rtc $(ROM_EFS)/bin/rtc
      cp -t sysinit $(ROM_EFS)/etc/config
      cp -t passwd shadow $(ROM_EFS)/etc
```

```
	romulate /tmp/rom.1


# Then take the output of ROMULATE and burn it into your 1MB eprom
os.rom.image : os.rom.build
	buildqnx n=$(NODE) os.rom.build tmp
	romqnx -c -d0 tmp tmp2
	-rm os.rom.image
	mkimage -d10 -oos.rom.image tmp2
	rm tmp tmp2

flash:	flash.info $(BOOTFILE) os.flash.image
	efsinit $(FLASH_RAW)
	mkcis -o$(FLASH_RAW) flash.info
	cat os.flash.image >$(FLASH_IMG)

os.flash.image : os.flash.build
	buildqnx n=$(NODE) os.flash.build tmp
	romqnx -c -d0 tmp tmp2
	-rm os.flash.image
	mkimage -d10 -oos.flash.image tmp2
	rm tmp tmp2


sram: sram.info
	mkcis -o$(SRAM_RAW) sram.info
	efsinit $(SRAM_EFS)
	mkdir $(SRAM_EFS)/bin
	mkdir $(SRAM_EFS)/etc
	mkdir $(SRAM_EFS)/etc/config
	cbpe -f /bin/login $(SRAM_EFS)/bin/login
	cbpe -f /bin/sh $(SRAM_EFS)/bin/sh
	cbpe -f /bin/tinit $(SRAM_EFS)/bin/tinit
	cbpe -f /bin/cat $(SRAM_EFS)/bin/cat
	cbpe -f /bin/ls $(SRAM_EFS)/bin/ls
	cbpe -f /bin/sin $(SRAM_EFS)/bin/sin
	cbpe -f /bin/shutdown $(SRAM_EFS)/bin/shutdown
	cbpe -f /bin/Dev $(SRAM_EFS)/bin/Dev
	cbpe -f /bin/Dev.con $(SRAM_EFS)/bin/Dev.con
	cbpe -f /bin/rtc $(SRAM_EFS)/bin/rtc
	cp -t etc/config/sysinit $(SRAM_EFS)/etc/config
	cp -t etc/passwd etc/shadow $(SRAM_EFS)/etc

clean:
	-rm tmp tmp2
```

The final file that needs to be examined is the OS.FLASH.BUILD file. This file is used by the BUILDQNX utility. The BUILDQNX utility is used to build a QNX boot image. The boot image is a collection of individual processes combined into a file image. This image is loaded into memory at boot time and is used to start the system. The first line in the OS.FLASH.BUILD file is the first process to which control is transferred when the system is starting up. It is responsible for starting all the rest of the processes listed in the boot image. Processes can be added or removed from the OS.FLASH.BUILD file. The processes included depend upon the services that are needed at startup. There are some processes that have to be in the boot image but others can be started after the system loads by putting them in the

SYSINIT file. The services that are necessary in the boot file are the Proc32 processes manager and the Slib16 shared library service. The EFSYS driver for the CPU used also needs to be in the boot image. This allows the system to mount the EFSYS partition or allows an alternate mount point to be specified. The boot image should contain the minimum number of services needed for the system to start. Listed below is the OS.FLASH.BUILD file used with a 5025A CPU

OS.FLASH.BUILD

```
/boot/sys/Proc32
$ 52000 Proc32 -l $n

/boot/sys/Slib16
$ 1 Slib16

/boot/sys/Slib32
$ 1 Slib32

/bin32/Efsys.5025a
$ 8000 Efsys.5025a -r1,89bd,256k,256k -r2,0,512k -m/

/bin/sinit
$ 1000 sinit TERM=qnx
```

PC-450 EXAMPLE

The following section is an example of an EFSYS system implemented on a PC-450 CPU. There are some differences from the 5025A example already outlined. The first one is in the FLASH.INFO file below. There are two types of partitions defined on the flash SSD. The first one is the boot image that is used to store the boot image created by the BUILDQNX utility. The second one is the EFSYS partition that is used to store the other services that will be used by the operation system once it is running. This is a read/write file system and can be used as a disk. It can also be used as the mount point for the file system if a hard drive is not available.

Note that the BIOS supplied by Octagon resides in the upper 512K of the lower 1 MB of the 2 MB flash disk (SSD1). This area can be overwritten by the EFSYS driver if the command line parameters are not specified correctly. To prevent this, it is important to specify a 512K offset for the EFSYS partition. To do this, you start the EFSYS.PC450 driver with the following parameters the first time you run it.:

```
Efsys.pc450 -r1,89a0,512k &
```

This will protect your BIOS area. Once the CIS is created this way, you can shorten it to EFSYS.PC450 & and leave out the command line parameters. Some users prefer to always include the command line parameters. This keeps the structure of the SSD clear to future users of the system and prevents any unintended erasing of the BIOS. If the BIOS is accidentally erased, you should call the Applications Engineering Group for assistance.

FLASH.INFO

```
#     Info file for Octagon PC450
#
#     Socket 1 (SSD1)   -     Intel 28F016 (2Mx8 flash)
#
#           0000k to 0512k    Image file system with OS image
#           0512k to 1024k    Used by BIOS
#           1024k to 2048k    Embedded File system
#
region
      jedec 0x89a0
      size 512k
      boot_file "/boot/sys/boot.pc450"

      partition
            type image
            size 256k
            attribute largest erase_align
region
      jedec 0x89a0
      size 1M
      offset 1M

      partition
            type efs
            size 512k
            attribute largest erase_align
```

SRAM.INFO

```
#     Info file for Octagon Control Card 5025
#
#     Socket 2 (SSD2)   -     512k SRAM
#           This socket contains a single EFS file system containing QNX
#           executables (non-bootable)
#
region
      size 512k
      type sram

      partition
            type efs
            attribute largest
```

MAKEFILE

```
#
#     Makefile for bootable Octagon PC450
#
NODE        = 1
FLASH_RAW   = //$(NODE)/dev/skt1
FLASH_IMG   = //$(NODE)/dev/skt1img1
BOOT_IMAGES = /boot/sys
BOOTFILE    = $(BOOT_IMAGES)/boot.pc450
EFS_FILES   = /bin/login /bin/sh /bin/tinit /bin/cat /bin/ls      \
            /bin/sin /bin/shutdown /bin/Dev /bin/Dev.con /bin/rtc \
            /bin/Fsys /bin/Fsys.ide /bin/mount /bin/echo
flash: flash.info $(BOOTFILE) os.flash.image
      efsinit /efs1p1
      efsinit $(FLASH_RAW)
      mkcis -o$(FLASH_RAW) flash.info
      cat os.flash.image >$(FLASH_IMG)
      mkdir /efs1p1/bin
      mkdir /efs1p1/etc
      mkdir /efs1p1/etc/config
      cbpe -f /bin/login /efs1p1/bin/login
      cbpe -f /bin/sh /efs1p1/bin/sh
      cbpe -f /bin/tinit /efs1p1/bin/tinit
      cbpe -f /bin/sinit /efs1p1/bin/sinit
      cbpe -f /bin/cat /efs1p1/bin/cat
      cbpe -f /bin/ls /efs1p1/bin/ls
      cbpe -f /bin/sin /efs1p1/bin/sin
      cbpe -f /bin/shutdown /efs1p1/bin/shutdown
      cbpe -f /bin/Dev /efs1p1/bin/Dev
      cbpe -f /bin/Dev.con /efs1p1/bin/Dev.con
      cbpe -f /bin/Fsys /efs1p1/bin/Fsys
      cbpe -f /bin/Fsys.ide /efs1p1/bin/Fsys.ide
      cbpe -f /bin/echo /efs1p1/bin/echo
      cbpe -f /bin/mount /efs1p1/bin/mount
      cbpe -f /usr/EKit/bin/Efsys.new /efs1p1/bin/Efsys.new
      cp -t etc/config/sysinit /efs1p1/etc/config
      cp -t etc/passwd etc/shadow /efs1p1/etc
      cp -t os.flash.image /efs1p1/.boot
os.flash.image : os.flash.build
      buildqnx n=$(NODE) os.flash.build tmp
      romqnx -c -d0 tmp tmp2
```

```
	-rm os.flash.image
	mkimage -oos.flash.image tmp2
	rm tmp tmp2
clean:
	-rm tmp tmp2
```

OS.FLASH.BUILD

```
/boot/sys/Proc32
$ 52000 Proc32 -l $n

/boot/sys/Slib16
$ 1 Slib16

/boot/sys/Slib32
$ 1 Slib32
#add the next three lines to boot from the file system on hd0
/bin/Fsys
$8000 Fsys

/bin/Fsys.ide
$1000 Fsys.ide

/bin/mount
$ 1000 mount -p /dev/hd0 /dev/hd0t77 /

/bin32/Efsys.pc450
#use the -m/ switch to boot from the file system on SSD0B. /
#you must remove the mount command for the hard drive to do this
#$ 8000 Efsys.pc450 -m/
$ 8000 Efsys.pc450

/bin/sinit
$ 1000 sinit TERM=qnx
```

5066 EXAMPLE

The following example is of a flash file system for a 5066 CPU with a 2 MB flash in SSD1. The SSD is formatted as a 2 MB image partition that mounts onto an ATA hard drive. This system could be set up with an EFSYS partition as well. The FLASH.INFO would have to be modified to reflect a 512K image partition and a 1 MB EFSYS partition. The FLASH.INFO file would be similar to the one used in the PC-450 CPU example. This allows the system to be mounted from the EFSYS partition on SSD1.

FLASH.INFO

```
#     Info file for Octagon 5066
#     Socket 1 (SSD1)   -     INTEL 29F016 (2Mx8 flash)
#           0000k to 0128k    Used by BIOS
#           0128k to 2048k    Image
#
region
      jedec 0x01ad
      size 2M
      boot_file "/boot/sys/boot.5066"

      partition
            type image
            attribute largest erase_align
```

MAKEFILE

```
#     Makefile for bootable Octagon 5066
#
NODE        = 1

SOCKET_RAW = skt2
SOCKET_IMG = skt2img1
SOCKET_EFS = efs2p1

FLASH_RAW   = //$(NODE)/dev/$(SOCKET_RAW)
FLASH_IMG   = //$(NODE)/dev/$(SOCKET_IMG)
DEST_EFS    = //$(NODE)/$(SOCKET_EFS)

BOOT_IMAGES = /boot/sys
BOOTFILE    = $(BOOT_IMAGES)/boot.5066

EFS_FILES   = /bin/login /bin/sh /bin/tinit /bin/cat /bin/ls      \
               /bin/sin /bin/shutdown /bin/Dev /bin/Dev.con /bin/rtc


flash:      flash.info $(BOOTFILE) os.flash.image
      efsinit $(FLASH_RAW)
      efsinit $(FLASH_RAW)
      mkcis -o$(FLASH_RAW) flash.info
      cat os.flash.image >$(FLASH_IMG)

os.flash.image : os.flash.build
```

```
	buildqnx n=$(NODE) os.flash.build tmp
	romqnx -c -d0 tmp tmp2
	-rm os.flash.image
	mkimage -oos.flash.image tmp2
	-rm tmp tmp2

clean:
	-rm tmp tmp2
```

OS.FLASH.BUILD

```
/boot/sys/Proc32
$ 52000 Proc32 -l $n

/boot/sys/Slib16
$ 1 Slib16

/boot/sys/Slib32
$ 1 Slib32
#add the next three lines to boot from the file system on hd0
/bin/Fsys
$8000 Fsys

/bin/Fsys.ata
$1000 Fsys.ata

/bin/mount
$ 1000 mount -p /dev/hd0 /dev/hd0t77 /


#use the -m/ switch to boot from the file system on SSD1 /
#you must remove the mount command for the hard drive to do this

/bin32/Efsys.5066

$ 8000 Efsys.5066

/bin/sinit
$ 1000 sinit TERM=qnx
```