NAME : LEKHAN.H. USN : 4NI20EC053

SUB : EMBEDDED SYSTEM COURSE CODE : EC6CO4.

1. Configuring and compiling Uboot and integrating a standlone application with Uboot.

Description of the commands :

1. wget https://mirror.cyberbits.eu/u-boot/u-boot-2010.03.tar.bz2

wget - command that initiates download. The command downloads the u-boot bootloader.

2. sudo apt-get install gcc-arm-none-eabi

The above commands installs the gcc-arm-none-eabi package which provides toolchain for ARM architecture. The toolchain include compilers, linker, assembler and other utilities

3. sudo apt-get install qemu-system.

The above command installs the 'qemu-system' package which provide a system emulator that can simulate different platform. This emulator can be used test and run software on different architecture without the need of physical hardware.

4. sudo apt-get install u-boot tools

It installs 'u-boot-tools' package which provides utilities for working with the u boot bootloader. This contains utilities tools for manipulating u boot image configuring uboot environments and generating u boot scripts

5. nano include/configs/versatile.h

opens the versatile.h file for editing using nano editor in include configs directory.

6. #define CONFIG_ARCH_VERSATILE_QEMU

define configuration options for the CPU - Processor used in ARM software development.

7. make versatilepb_config ARCH=arm CROSS_COMPILE=arm-none-eabi-

The above command configures the linux-kernel build for the ARM based versatile pb development board using the appropriate configured file and cross-compiler

8. make all ARCH=arm CROSS_COMPILE=arm-none-eabi-

This command is used to build the linux kernel for the ARM architecture using cross-compiler. Compile all source file ARM architecture

9. arm-none-eabi-gcc -c -nostdlib mcpu=arm926ej-s hello.c -ohello.o

This command compiles the code in the file "hello.c" into object file named 'hello.o' -c indicates the compiler to generate all object file rather than executable binary -nostdlib tells compiler not to include standart library in output file mcpu=arm926ej-s specifies the target CPU architecture.

10. arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o

This command assembles the ARM assembly code in the file into object file named 'startup.o' mcpu=arm926ej-s specifies CPU found. -g generate debugging information for use with a debugger.

11. arm -none-eabi-ld -T linker.ld hello.o startup.o -o hello.elf

This command links the object file 'hello.o' and 'startup.o' into executable file named 'hello.elf'. The T-linker specifies the linker script to use information about memory layout of the target system.

12. arm-none-eabi-objcopy -O binary hello.elf hello.bin

This command copies the binary data from the executable ELF file 'hello.elf' into binary file named 'hello.bin'. The '-O binary' option specifies the output format binary.

13. mkimage -A arm -C none -O linux -T Kernel -d hello.bin -a 0x00100000 -e 0x00100000 hello.uimg

creates a uimage file "hello_uimg" for an ARM architecture linux kernel using hello.bin as the input, setting the load and entry address to 0x00100000 using the 'mkimage' tool.

14. cat u-boot.bin hello.uimg > flash.bin

Concatenates the file "u-boot.bin" and "hello.uimg" and creates a combined output file "flash.bin" using the 'cat' utilities.

15. printf "bootm 0x%x\n" $(($(stat -c %s u-boot.bin)+65536))

Calculates and print the boot address for the uboot bootloader based on the size of the 'u-boot.bin' file adding 65536 bytes to file size using printf and stat commands. specifies the address of uboot binary file (u-boot.bin) load address to linux kernel.

16. qemu-system-arm -M versatilepb -m 128M -kernel flash.bin -nographic

This command launches the QEMU emulator with board emulating 128Mb of RAM allocated. This kernel loads image from flash.bin using i/p & o/p to terminal

2. Explain the importance of startup code and linker code file in arm board.

Linker and startup code are essential components in the process of building executable program, particularly in embedded system development.

Startup code: The startup code is a small piece of code that runs before the main function of the program executed. The startup code provides the reset vector, initial stack pointer value of a each of the interrupt vector. It is first piece of code to execute after the system reset, without startup code. The program would be not able to execute correctly as the system hardware would not be correctly initialized and program may not be allocate memory correctly.

Linker code: The linker file shows how physical data memory and code memory are swapped into our address space. The linker code that takes a compiled object file generated by the compiles and linker them together into single executable file. It involves references b/w different modules such as function calls and generates a binary file that can be executed on the target platform without linker, it would be challenging to create complex applications as it'll linker together the code for multiple source file and libraries resulting single executable file.

Finally the linker and startup code are critical in the process of building executable program. The linker links together multiple compiled object file and libraries, creating a single executable. and startup code initializes the hardware and executes the program.

KERNEL LINUX

* wget http://ftp.riken.jp/Linux/Kernel.org/linux/kernel/v4.x/linux-4.9.11.tar.gz.

This command helps in downloading the linux, it compress the tarball file of the linux kernel version 4.9.4

* tar -xf linux-4.9.11.tar.gz.

This command will extract all the kernel source file from the tarball file

* sudo apt-get install gcc-arm-none-eabi;

This command will install the GNU compiler collection for the ARM architecture

* sudo apt-get install qemu-system.

This command will installs qemu emulator that allows running virtual machine on the host system.

* sudo apt-get install gcc-arm-linux-gnueabiff

This command will installs the gcc package which will enable the user to compile C & C++ code for arm based linux system.

* sudo apt-get install

This command will install the package from Debian-based linux distribution software respositiory

* sudo apt-get install libncurses-dev.

This command will install the packages of libncurses. It will allow the user to compile program that will use ncurses library

make clean

This will clean all the executables & object files created / generated by previous build.

make distclean

It removes all the files generated by the build process including the configuration files, build artifacts and intermediate files

export ARCH=arm

This command is used to set an environment variable called ARCH to the value arm

export CROSS_COMPILE = arm-linux-gnueabi

This command is used to set an environment variable called CROSS- COMPILE to the value arm-linux-gnueabi

make vexpress-defconfig.

It is used to generate a default configuration file for versatile express development board, which specifies the build options and settings required for particular board.

make menuconfig

This command is used to launch a menu based GUI that allows to config the build settings for the software that will compile

make j2 all.

It is used to build software are compiled in parallel using the multiple cores to run 2 jobs in parallel

qemu-system-arm -M vexpress-a9 -dtb ./dtb/vexpress-v2p-ca9.dtb
-kernel zImage -append "console=ttyAMA0" -nographic

The above command is used to run the linux kernel image on a versatile express development board using QEMU emulator

arm linux-gnueabi/-gcc -static hello.c -o hello

This command will compile a c program "hello.c" into a statically linked binary executable called "hello" using compiler

echo hello | cpio -o -format=newc > initramfs

It is used to create an initial RAM file system archive that contain a single file named "hello" with the contents of hello inside it.

cpio -t < initramfs

This command is used to list the contents of the initramfs file which is a cpio archive containing initial root file system.

qemu-system-arm -M vexpress-a9 -dtb ./dts/vexpress-v2p-ca9.dtb -kernel zImage -initrd initramfs -append "rdinit=hello console=ttyAMA0" -nographic

This command will start QEMU, emulating the vexpress a9 board load linux kernel image and DTB and specifies kernel command line options to the kernel. The kernel will mount the initramfs file system and executes the "/hello" file as initial process. Hence we get object

Root File System (Busy Box)

1. wget http://ftp.riken.jp/linux/kernel/.org/linux/kernel/v4.1/linux-4.4.1 tar.gz

wget command is used to download files from the internet and in that case, it is downloading the file "Linux-4.4.1tar.gz" from the specified URL.

2. wget http://www.busybox.net/downloads/busybox-1.23.1.tar.bz2

downloads the file busybox-1.23.1.tar.bz2" from the specified URL, which is the official website for Busybox download.

3. sudo apt-get install gcc-arm-linux-gnueabi

used to install GNU compiler collection [GCC] for the arm architecture on a linux system.

4. sudo apt-get install qemu-system

used to install the QEMU emulator system on a linux system allowing to run virtual machines & simulate.

5. export ARCH = arm

sets the environment variable "ARCH" to specify the target architecture as ARM

6. export CROSS_COMPILE = arm-linux-gnueabi

sets the environment variable 'CROSS_COMPILE' to specify the cross compiler prefix for ARM.

7. make Vexpress_def config

configures the kernel with the default configuration for the ARM versatile express platform

8. make zImage -j8
Compress the linux kernel image using 8 parallel jobs (-j8)

9. make modules -j8
Builds the kernel modules on parallel using 8 jobs

10. make dtbs
Generates the device tree binary file

11. make defconfig
Generates the default configuration for the kernel based on the existing architecture and platforms.

12. make CROSS_COMPILE = arm_linux_gnueabi
Executes the make command with specified CROSS_COMPILE prefix which indicates that subsequent compilation will be done for the ARM architecture

13. make install CROSS_COMPILE = arm_linux_gnueabi
installs the compiled kernel and associated files using the specified CROSS compiler prefix, typically used for deploying the kernel on an ARM based system

14. sudo mkdir rootfs
Creates a new directory named rootfs with root privileges. This command is typically used to create a directory for a rootfile system

15. sudo mkdir rootfs/lib
Creates a new directory named 'lib' within the 'rootfs' directory with root privileges. This command is commonly used to create a subdirectory for storing library files within a root file system

16. sudo cp -r install/* ~/rootfs/
copies all files and directories from the "install" directory to the roots directory recursively (-r flag). used to populate root file system

17. sudo cp -p /usr/arm-linux-gnueabi/lib/* ~/rootfs/lib/
copies all files from the "/usr/arm-linux-gnueabi/lib/" directory to the "rootfs/lib/" directory while preserving symbolic links

18. sudo mkdir -p ~/rootfs/dev
creats the directory structure for 'dev' within "rootfs" directory including any necessary parent directories, using root privilages.

19. sudo mknod ~/rootfs/dev/tty1 c 4 1
sudo mknod ~/rootfs/dev/tty2 c 4 2
sudo mknod ~/rootfs/dev/tty3 c 4 3
sudo mknod ~/rootfs/dev/tty4 c 4 4
above commands creat character device nodes named "tty1, tty2 tty3 & tty4" in the "rootfs/dev" directory. These device nodes are created using the 'mknod' command with the specified major numbers 4 and respective minor numbers 1, 2, 3, 4.

20 dd -if=/dev/zero of=~/rootfs.ext3 bs=1M count=32
creats a file named "rootfs.ext3" with a size of 32 megabyte with zeros. It uses 'dd' utility to perform low-level data copying and manipulation, in this case creating a blank file for a potential ext3 filesystem.

21. mkfs.ext3 ~/rootfs.ext3
formats the file 'rootfs.ext3' with the ext3 filesystem preparing it for use as a storage device or file system

22. sudo mkdir tmpfs
Create a directory named tmpfs with root privilages used as mount point for a temporary file system.

23. sudo mount -t ext3 agsroot fs.ext3 tmpfs/ -o loop
mounts the file "agsroots.ext3" as an ext3 filesystem to the tmpfs directory using the loopback interface.

24. sudo cp -r rootfs/* tmpfs/
copies all files and directories from the rootfs to the tmpfs recursively (-r flag)

25. sudo umount tmpfs
Unmounts the filesystem mounted at tmpfs directory

26. qemu-system-arm-mvexpress-a9-m 512m -dtb extra-folder/vexpress-ca9.dtb -kernel /extra-folder/zImage -nographic -append "root=/dev/mmcblk0 rw rootwait console=ttyAMA0" -sd agrootfs.ext3
Setup and starts QEMU, emulating vexpress.a9 board with specific memory size, device drive, kernel image and rootfilesystem.

27. qemu -system -arm -m Vexpress -a9 -m 512m -dtb extra folder/vexpress-dtb -kernel extra -folder/zImage -append "root=/dev/mmcblk - sda agrootfs.ext3
open in new window (maybe GUI