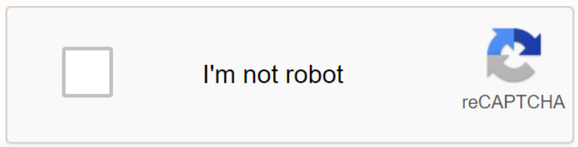

Continue

Raspberry pi 3b startup file

The boot process is the different stages the system goes through from powering up the device to booting the operating system. This article explains the boot process for Raspberry Pi 3 Model B. Let's verify the boot process practically as well. Boot process of Rapsberry Pi 3 Model BRPI3 model B boot sequenceHere are the different stages of the RPI3 Model B boot sequence involves -The boot ROM (the first stage bootloader) is programmed to SoC during the manufacture of RPI. This code looks for the bootcode.bin file (second stage bootloader) in one of the sd card partitions and runs it. The bootcode.bin code looks for the file config.txt for any third stage bootloader info. If nothing was found, it loads the default bootloader start.elf from the SD card and runs it. The start.elf code reads config.txt several times to initialize basic hardware, load dtb and kernel in RAM. The official page to configure config.txt file can be found here. Let's start RPI practically. Download bootcode.bin and start.elf files from Official rpi firmware repo. These files are proprietary GPU firmware files and are necessary to launch RPI3 model B.Copy downloaded files to ~/rpi3/sdCardFiles/ directory.mkdir -p ~/rpi3/sdCardFiles cd ~/rpi3/sdCardFiles cp ~/Downloads/bootcode.bin ~/Downloads/start.elf ./ Copy the file u-boot.bin as compiled from the previous article also to the folder ~/rpi3/sdCardFiles/./ cp ~/rpi3/u-boot/u-boot.bin ./ Add basic configuration to the config.txt configuration fileCreate a minimal konbig.txt configuration in the ~/rpi3/sdCardFiles/ director configuration file with the following lines.cat < eof=> > config.txt arm_64bit=1 kernel=u-boot.bin EOF The kernel=u-boot.bin loads u-boot.bin instead of the actual Linux kernel. The arm_64bit=1 tells the firmware to launch ARM cores in 64 bit mode. Directory ~/rpi3/sdCardFiles/ must have the following files.bootcode.binconfig.txtstart.elfu-boot.binPrepare SD cardCreate three partitions using fdisk and mkfsFirst partition is of size 1GB, type fat32 with label start. Create filesystem with mkfs.vfat.Second partition is of size 512MB, write Linux with label rootfs. Create filesystem with mkfs.ext4.The third partition is of the remaining of SD card storage with the type LinuxOnce SD card partitioning is done, copy the above mentioned files to SD card.cp -r ~/rpi3/sdCardFiles/. <PATH_TO_BOOT_PART_MOUNT_POINT> # Use lsblk command to find mount point Ex: cp -r ~/rpi3/sdCardFiles/. /media/nayab/boot/Power up the board. Connect the USB keyboard to raspberry pi USB. Attach the TV/monitor to raspberry pi with the HDMI cable. Remove the SD card from the system. Insert the card into the Raspberry Pi SD slot and force up the board. You should see the following logs that appear on your TV/monitor.Net: No ethernet found. starts USB... Bus usb@7e980000: scanning bus usb@7e980000 for devices... 3 USB Device (s) found scanning usb for ... 0 Lagringsenhet (er) hittades Hit någon nyckel för att stoppa autoboot: 0 U-Boot> U-Boot> Typ hjälp i U-boot </PATH_TO_BOOT_PART_MOUNT_POINT> </PATH_TO_BOOT_PART_MOUNT_POINT> to see available U-boot commands. Page 2In the previous post we saw the boot process of raspberry pi 3 model b and how to log on to the U-boot console. In this article we will see how to establish TFTP protocol communication between development systems and RPI. The development system acts as a TFTP server and is used to host the files that can be transferred to RPI. After establishing tftp communication, U-boot will be able to download the files from the TFTP server and download into ram. In the upcoming articles we will use this method to download the Linux kernel and Device tree from the TFTP server to Rapsberry Pi. TFTP communication with Raspberry Pi 3 Model BSetup U-bootConnect Rapberry PI to the Ubuntu system using Ethernet cable. Both Raspberry PI and the host system should belong to the same network in order to communicate. Let's setup some environment variables. Power up the Raspberry Pi, press Enter to log on to the U-boot console immediately after powerup. In the console, run the following commands.setenv ipaddr 192.168.1.115 setenv serverip 192.168.1.15 # IP address for host system ethernet interface saveenv The environment variable ipaddr is the IP address of Raspberry Pi. The serverip environment variable is the system IP address in which the TFTP server is running. I use my development system as a TFTP server. This can be any system as long as it is on the same network as Raspberry Pi.The environmental varie) saves environmental data to the SD card so you don't have to enter this data every time the board boots. Set up the serverFriendly use this tutorial to install the TFTP server in your development system (Ubuntu). Note down the tftp directory path. The default directory can be either /srv/tftp or /var/lib/tftpboot. This will be used to store the files that can be transferred to RPI. Change the IP address of your Ubuntu system Ethernet interface IP address to the serverip address. In this case it is 192.168.1.15. The easy way to do this is to use the following command.nmcli con add type ethernet ifname <Ethernet_interface_name> ip4 192.168.1.15/24 Ex : nmcli con add type ethernet ifname enp3s0 ip4 192.168.1.15/24You can find your Ethernet-interface name from the ifconfig -a.Testing connectionFirst change ownership of TFTP directory.sudo chown -R USERNAME TFTP_DIRECTORY # Replace USERNAME and TFTP_DIRECTORY with your Ex: sudo chown -R nayab /srv/tftp/Let's create a dummy file called testFile in the t ftp directory of your host system.echo dummy data > /srv/tftp/testFile Use the following command in the Raspberry Pi U-boot console to download the testFile file from the TFTP server. Make sure that the file is successfully uploaded. At the end of the transfer, you should see the message similar to the following. Loading: ## TFTP communication to transfer files from host system to destination (RPI3 Model B). In this article, we will compile the Linux kernel for Raspberry Pi 3 Model B, load kernel and device tree from U-Boot. Make sure that tftp communicaiton works well before proceeding with the following. Cross compiling and loading Linux kernel for Raspberry Pi 3 Model BLoad kernelRaspberry Pi organization maintains it's own fork of Linux kernel to support all Raspberry Pi variants. Let's download the Linux kernel from offical Raspberry Pi repository.cd ~/rpi3 git clone cd linux git checkout rpi-4.19.y Create a branch for your local development.git checkout -b<new_branch_name>ex: git checkout -b 4.19.y-localCompile for RPILets use the cross-toolchain we compiled here. Let's compile Linux kerenl for arm64. Run the following commands to set arch and CROSS_COMPILE environment variables.export ARCH=arm64 # This should match one of the directories in the 'arch' folder. export CROSS_COMPILE=aarch64-rpi3-linux-gnu-export PATH=$PATH:~/x-tools/aarch64-rpi3-linux-gnu/bin/ This kenel has a pre defined configuration file for Raspberry Pi 3 Model B named bcmrpi3_defconfig present in arch/arm64/configs directory. Now let's generate the .config file.Compile Linux kernel by running the following command. The option allows -j will compile the Linux kernel with the available nproc CPU kernels. Download Linux kernel and device treeAs we will change the Linux kernel and driver source often in the later articles, it will be troublesome to copy the Linux kernel to the SD card every time we change something. So we'll automate the process of loading the kernel and device tree in the following sections. This eliminates the need for removal and insertion of SD cards from/to RPI. After the completion of cross-compilation for linux kernel, let's copy the kernel and drive tree binary to tftp server directory.cp arch/arm64/boot/Image <tftp server=directory=>/kernel.img cp arch/arm64/boot/dts/broadcom/bcm2710-rpi-3-b.dtb <tftp server= directory=>Example:cp arch/arm64/boot/Image /srv/tftp/kernel.img cp arch/arm64/boot/dts/broadcom/bcm2710-rpi-3-b.dtb /srv/tftp/ Load Linux kernel and dtb file from the U-boot consoleNow powerup board, enter the U-boot console and run the following commands.tftp 0x200000 kernel.img tftp 0x200000 bcm2710-rpi-3-b.dtb booti 0x2 000 0000 - 0x200000 Tftp 0x2000000 kernel.img command will load the kernel.img file from the TFTP server directory to the RAM address 0x2000000. The tftp 0x200000 bcm2710-rpi-3-b.dtb command will load bcm2710-rpi-3-b.dtb device tree binary from the TFTP server to the RAM address of 0x200000. The booti command is used to boot the Arm64 Linux image from memory. It has 3 arguments. The first argument is the core load address - 0x2000000. The second argument is the initramfs load address. We don't use initramfs. So passes - (hyphens). The third argument enhetstrådet binär belastning</tftp> </tftp> </new_branch_name> </new_branch_name> - 0x200,000. We can automate the above process of core and drive tree booting every time the board boots up by setting bootcmd env in the U-boot console.setenv bootcmd 'tftp 0x2000000 kernel.img; tftp 0x200000 bcm2710-rpi-3-b.dtb; booti 0x2000000 - 0x200000' saveenv Page 4I the previous articles, we compiled toolchain, U-boot, Linux kernel for Raspberry Pi 3 Model B.In for a Linux system to work fully, a few applications are required. When the Linux kernel boots, it tries to mount the file system and then looks for an init program to perform. If the init program is not found, the kernel panic and boot process will be stopped. A shell to execute the commands and some basic tools are also required to make the Linux system usable. Busybox is a software suite, which generates a minimal root file system with an init application, shell and basic Linux tools. Generate minimal root filesystem for RPI3 Model BGet the busybox source code ~/rpi3/ git clone git://busybox.net/busybox.git cd busybox # Check out to the latest stable version. Can be found with 'git tag --sort=v:refname' git checkout 1_31_1 Change configurationBuild busybox statically (without shared weights) by enabling Settings -> Build statically binary (no shared library)Add cross compiler prefix (aarch64-rpi3-linux-gnu-) by going into Settings -> Cross compiler prefixAdna installation directory by going into Settings -> Destination path for 'do install'. Give it all the way. Ex: /home/USERNAME/rpi3/nfs. Replace username with you Ubuntu system username. Download the patch to fix compile time releaseYou may see the following compilation time error under compilation.date.c:(.text.date_main+0x21c): undefined reference to 'stime' collect2: error: ld returned 1 exit status Note: if build needs additional libraries, put them in CONFIG_EXTRA_LDLIBS. Download and apply the fix to fix the above compile time error.curl --output ./0003-compile-error-fix-stime.patch git apply 0003-compile-error-fix-stime.patch Compile busybox PATHexport=$PATH:~/x-tools/aarch64-rpi3-linux-gnu/bin/ make -j'nproc' Install the minimal filesystemThis will install the minimal root file system to the path ~/rpi3/nfs. Verify the content using the following command. The list should be similar to the following. In the next section we will make the Linux kernel mount the root file system over networks. Page 5I the previous posts we have compiled Linux and busybox filesystem. In this article, we'll install kernel moduels to the busybox filesystem /lib directory. Install Linux kernel modules in busybox filesystem for RPI 3BInstall kernel modulesLets go to the Linux source directory where we recently compiled the Linux kernel. Install the kernel modules to the busybox root filesystem directory with the following commands.cd ~/rpi3/linux export ARCH=arm64 CROSS_COMPILE=aarch64-rpi3-linux-gnu- export export do modules_install INSTALL_MOD_PATH=~/rpi3/nfs/ INSTALL_MOD_PATH refers to the path where we want to install the Linux kernel modules. The ~/rpi3/nfs path is the root file system generated by busybox. Now we have installed the core modules for this root file system. The core modules will be installed in a separate folder in the ~/rpi3/nfs/lib/modules/ path with the name similar to 4.19.115-v8+. Here 4.19.115-v8+ is the Linux kernel version that we compiled. Page 6In the previous articles we have seen how to cross compile linux kernel, load it from the tftp server and how to generate the root file system using busybox. In this article we will make Linux kernel boots like root filesystem over networks. The root file system that we generated through busybox on a development system can be accessed by Raspberry Pi 3B through the network using NFS. The reason behind booting the Raspberry Pi 3B over NFS is to eliminate the process of copying the modified files or kernel to the SD card for every single change we made. Network start your Raspberry Pi 3BSetting NFS server on host systemInstall nfs kernel-server package on you Ubuntu development system by running after command.sudo apt install nfs kernel server When installation is complete, add the following line to /etc/export file. <path_to_busybox_install_directory> <ip_address_of_board>(rw,no_root_squash,no_subtree_check) If you follow root filesystem generation using busybox, we have installed the root file system in the ~/rpi3/nfs path. Replace <path_to_busybox_install_directory>with the absolute path of the root file system and replace <ip_address_of_board>with the RASPBERRY Pi 3B IP address that we set up during TFTP communication provisioning. An example is given below./home/nayab/rpi3/nfs 192.168.115(rw,no_root_squash,no_subtree_check)Restart the NFS serverRun the following command to restart the NFS server on your host system.sudo service nfs-kernel-server restart Set bootargs env board of the U-boot consolePower up the board, stop in the U-boot console and set bootargs env using the following command. The bootargs value is what will be passed as command line arguments to the Linux kernel while booting. setenv bootargs # Reset bootargs setenv bootargs root=/dev/nfs ip=<board_ip>::::thth0 nfsroot=<host_ip>:<path_to_busybox_on_host>,nfsvers=3 rw saveenv An example is given below -setenv bootargs # Reset bootargsenv bootargs root=/dev/dev/dev/nfs ip=192.168..1.115:::thth0 nfsroot=192.168.1.15:/home/nayab/rpi3/nfs,nfsvers=3 rw saveenv Booting the RPI through NFSMake sure that Raspberry Pi is connected via Ethernet to the system where nfs-kernel server is run. Power on the board or run recovery command in the U-boot console. The board must download the Linux kernel and dtb file from the tftp server and then the Linux kernel should start the root filesystem over nfs. If you see the following logs flooding your screen, create a dev directory in the NFS path. Run mkdir -p kommando</path_to_busybox_on_host> </host_ip> </board_ip> </ip_address_of_board> </path_to_busybox_install_directory> </ip_address_of_board> </path_to_busybox_install_directory> </path_to_busybox_install_directory> restart your board.can't open /dev/tty2: No such file or directory can not open /dev/tty3: No such file or directory can not open /dev/tty4: No such file or directory can not open /dev/tty2: No such file or directory can not open /dev/tty4 tty3: No such file or directory can not open /dev/tty4: No such file or directory can not open /dev/tty2: No such file or directory can not open /dev/tty3: No such file or directory can not open /dev/tty4: No such file or directory . . At this stage you should be able to perform commands such as LS, mkdir, etc.. Page 7In the previous articles, we cross-combed the basic software packages, managed to launch Raspberry Pi 3B over the network and be able to perform some basic commands. In this article we will find out how to mount virtual file systems and add /change/remove init scripts. System configuration and init script for Raspberry Pi 3BMount virtual file systemsIf you try to execute the commands such as PS or any other process related command, you will encounter an error that says,PID USER TIME COMMAND ps: can not open '/proc': No such file or directory It is because the proc virtual file system is not mounted. Similarly, there is a sysfs virtual file system that exposes hardware and device driver information from kernel space to user space. Let's create mount points in the root file system before you mount the virtual file systems. Create proc, sys directories in busybox filesystem with the following command.mkdir -p ~/rpi3/nfs/{proc,sys} Now power up the board and mount these virtual file systems using the following commands.mount -t proc nodev/proc mount -t sysfs nodev /sys Run ps command. It should work fine now. The above process of assembly can be automated every time the board boots using init scripts. Let's see the next section on /etc/inittab. InitprogramWhen Linux starts the file system, the first application it executes is the /sbin/init. This Busybox init application checks, if present, the configuration file /etc/inittab for the system configuration and init scripts. If the configuration file is not found, the default configuration will be applied. Configuration file for init programFor more information about inittab configuration file, please read ~ / rpi3 / busybox / example / inittab after busybox compilation tutorial. Let's create /etc/inittab file with the following configuration.cat << EOF >> ~/rpi3/nfs/etc/inittab ::sysinit:/etc/init.d/rcS tty1::askfirst:/bin/sh ::ctrlaltdel:/sbin/reboot ::shutdown:/sbin/swapoff -a ::shutdown:/bin /umount -a r ::restart:/sbin/init tty2::askfirst:/bin/sh tty3::askfirst:/bin/sh tty4::askfirst:/bin/sh EOF The init scriptsThe first line int the inittab file, ::sysinit:/etc/init.d/rcS tells the init program to execute the /etc/init.d/rcS script. We can add our proc and sysfs mounting commands to this file. Each time startar upp, proc och sysfs filesystems kommer att monteras automatiskt av den /etc/init.d/rcS script.mkdir -p ~/rpi3/nfs/etc/init.d/ cat << EOF >> ~/rpi3/nfs/etc/init.d/rcS #!/bin/sh mount -t proc nodev /proc mount -t sysfs nodev /sys LD_LIBRARY_PATH=/lib:$LD_LIBRARY_PATH export LD_LIBRARY_PATH EOF Make the file executable by running after commandchmod +x ~/rpi3/nfs/etc/init.d /rcS Now we'll start raspberry pi and see if we could see any output for the following commands -ps pidof init ls/sys # and ls / proc We can call as many scripts as possible by including the scripts path in / etc / init.d / rcS script. In the next article we will compile a sample binary for Raspberry Pi on the host system and try to perform it on target. Page 8Earlier in Embedded Linux with Raspberry Pi 3B series, we have generated minimal root filesystem statically for Raspberry Pi using the busybox build system. The file system generated in this way has no shared libraries. If we keep cross compiling statically for every program we've written, we'll end up with binaries and the final Embedded Linux system of size much more. The goal of this article is to generate the file system with shared libraries in place, making binaries size smaller. Busybox filesystem with shared librariesSlutes write, compile a simple Hello World! program for Raspeberry Pi 3B and perform on board.cat < eof=> > helloworld.c #include <stdio.h> int main(void) { printf(Hello World!); return 0; } EOF Compile the program using cross-toolchain gcc.export PATH=$PATH:~/x-tools/aarch64-rpi3-linux-gnu/bin/ aarch64-rpi3-linux-gnu-gcc helloworld.c -o helloworld Now try to execute the HelloWorld program on Raspberry Pi by running the following command. You will face the following error / bin / sh: / helloworld: not found It is not like the file is missing. The file is still in your root directory. You can confirm with the ls -l /helloworld command on the board. This is because the linker library (ld-linux-aarch64.so.1) is missing. Let's copy the library from the toolbar path to the file system /lib directory.mkdir -p ~/rpi3/nfs/lib/ cd ~/rpi3/nfs/ ln -s lib lib64 cp ~/x-tools/aarch64-rpi3-linux-gnu/aarch64-rpi3-linux-gnu/sysroot/lib/ld-linux-aarch64.so* ~/rpi3/nfs/lib Try to perform helloworld-binary again. You may need to export LD_LIBRARY_PATH with the command, export LD_LIBRARY_PATH=/lib/ in the board. Let's see if any other shared libaries are missing. Copy all shared libraries reported from the diriectory tool chain to ~/rpi3/lib directory.cp -f ~/x-tools/aarch64-rpi3-linux-gnu/aarch64-rpi3-linux-gnu/sysroot /lib /libc.so* ~/rpi3/nfs/lib cp -f ~/x-tools/aarch64-rpi3-linux-gnu/aarch64-rpi3-linux-gnu/sysroot/lib/libm.so* ~/rpi3/nfs/lib cp -f ~/x-tools/aarch64-rpi3-linux-gnu/aarch64-rpi3-linux-gnu/sysroot/lib/libresolv.so* ~/rpi3 /nfs/lib Cross-compile busybox filesystem againLet's check the size of the busybox binary before deactivating the static building.ls -lh ~/rpi3/nfs/bin/busybox It's about 1.9M size when I checked it. Since the shared libraries are in place now, we can cross-border busybox genom att inaktivera </stdio.h> </stdio.h> -> Build static binary (no shared libraries) options in make menuconfig.cd ~/rpi3/busybox/make menuconfig Save config file, compile filesystem and install.make -j'nproc' do install Now check the size of Busybox binary again. It's 990K for me. The size definitely decreased as it will use the shared libraries we copied above. Page 9Now we have our filesystem with the necessary libraries ready and kernel modules installed to the root filesystem, we will put everything we have compiled / generated so far - bootloader, kernel and root filesystem - in the SD card and boot from it. Boot Raspberry Pi 3B from the SD cardMake a SquashFS image of the root filesystemThe SquashFS is a read-only file system with high compression and reading speed. We will create a SquashFS image of our root filesystem and flash as image to the secondary partition of sd card (we created the second partition with 512MB size). Install the squashfs tool package on your Ubuntu development system with the following commandsudo apt install squashfs-tools Create squashfs image of root directory squashfs ~/rpi3/nfs ~/rpi3/sdCardFiles/rootfs.sqfs -noappend Flash the squashfs image to sd cardoverlay sd card partitions using the following commands first before flashing # Replace <USERNAME>with your Ubuntu system username. sudo umount /media/<USERNAME>/rootfs The dd command will flash the squashfs image to the second partition of the memory card.# The second patition name may vary. Check the 'dmesg' output after inserting the SD card into the system. sudo dd if=~/rpi3/sdCardFiles/rootfs.sqfs of=/dev/mmcblk0p2 bs=1M Re-compile kernel with squashfs as inbuilt modulec sudo dd if=~/rpi3/sdCardFiles/rootfs.sqfs of=/dev/mmcblk0p2 bs=1M Re-squash fs as inbuilt modulec sudo dd if=~/rpi3/sdCardFiles/rootfs.sqfs of=/dev/mmcblk0p2 bs=1M Re-compile kernel with squashfs as built modulec sudod ~/rpi3/linux do menuconfig Make squashfs as inbuilt core module by entering the File system — > Miscellaneous file system – > SquashFS 4.0 - Squashed file system support. Save the config file and recompile the kernel by running for commands.export ARCH=arm64 export CROSS_COMPILE=aarch64-rpi3-linux-gnu-export PATH=$PATH:~/x-tools/aarch64-rpi3-linux-gnu/bin/ make -j'nproc' cp ~/rpi3/linux/linux/linux/arch/arm64/boot/Image /media/<USERNAME>/boot/kernel.img cp ~/rpi3/linux/arch/arm64/boot/dts/broadcom/bcm2710-rpi-3-b.dtb /media/<USERNAME>/boot/ Set bootcmd env on the board U-boot consoleInsert SD card into Raspberry Pi. - Power up the board. Stop at the U-boot console and run the following commands to launch Raspberry Pi 3B completely from SD card.setenv bootargs earlyprintk root=/dev/mmcblk0p2 rootfstype=squashfs rootwait noinitrd setenv bootcmd 'mmc dev 0; fatload mmc 0:1 0x200000kernel.img; fatload mmc 0:1 0x200000 bcm2710-rpi-3-b.dtb; booti 0x2000000 - 0x200000;' saveenv Explanation of the first command following -The bootargs env value is what will be passed as the kernel command line argument while booting. The root=/dev/mmcblk0p2 tells the kernel to look for the root file system from the second partition of the kernel. The rootfstype =squashfs tells the filesystem type. It's squash</USERNAME> </USERNAME> </USERNAME> </USERNAME> </USERNAME> rootwait tells the kernel to wait until the SD card is ready for the root file system. The noinitrd tells us there is no initramfs created. The explanation of the second command as follows -The bootcmd env value is a bunch of U-boot commands that will be executed every time the board boots. Mmc dev 0 selects the first mmc device. Here the SD card. You can check the mmc details using the mmc info command in the U-boot console. The fatload mmc 0:1 0x2000000 kernel.img loads kernel.img image from the first partition of SD card to RAM address 0x2000000The fatload mmc 0:1 0x200000 bcm2710-rpi-3-b.dtb loads the device tree file from the first partition of SD card to the RAM address 0x200000The booti 0x2000000 - 0x200000 boots image (kernel.img) at address 0x2000000 with file dtb (bcm2710-rpi-3-b.dtb) at address 0x200000 with no initramfs(-). The final saveenv command saves the environment variables on SD cards. Restart the boardEnter following command in the U-boot console to restart the Raspberry Pi 3B from the SD card. Congratulations on successfully starting the board from SD card. Page 10In the previous article, we have successfully booted the Embedded Linux system we created by compiling toolchain, U-boot, Linux and root filesystem individually. But there are so many useful programs missing, like ssh server and client applications. In this guide we will use a build system called Buildroot to see how to install other useful applications to the root file system. The Buildroot itself is a tool for generating toolchain, Linux kernel and root file system combiningly. But we will use it to generate the root file system with the other useful apps. Here I am going to take dropbear which is very small ssh server and client program. We will install dropbear in the root filesystem, copy the files to the SD card and start the board. Buildroot for RPI 3B to install appsDownload BuildrootDownload the build system from the official website.cd ~/rpi3/ wget -c takes xvf buildroot-2020.02.2.tar.gz Configure Buildroot-systemGo to the extracted Buildroot directory and configure the following options by running make menuconfigcd ~/rpi3/buildroot-2020.02.2/ makemenuconfig Select Target Options -> Target Architecture -> Aarch64 (Lite endian)Select Toolchain -> Toolchain type -> External tool chain. The total construction time will decrease because we will not compile the tool ring again. Select Toolchain -> Toolchain -> Custom Tool ChainSelect Toolchain -> Toolchain Path. And enter the path to the tool ring. Ex: /home/USERNAME/x-tools/aarch64-rpi3-linux-gnuSelect Toolchain -> Toolchain has SSP supportSelect Tool chain -> Toolchain has RPC supportSelecting Toolchain -> Toolchain prefix -> $(ARCH)-rpi3-linux-gnuSelect Toolchain -> External tool chain kernel headers series -> 4.19.x or later. Select Toolchain -> verktygskedja C-bibliotek -> glibc/eglibcSelect Target-paket -> -> application -> dropbearSelect System configuration -> Root password. Enter the new password. We will use this password to log on to the board console later. Download the patch to fix the compilation problemYou can see the following compilation time error during compilation.date.c:(.text.date_main+0x21c): undefined reference to 'stime' collect2: error: ld returned 1 exit status Note: if building needs additional libraries, put them in CONFIG_EXTRA_LDLIBS. Download the patch to fix the above compile time error.curl --output ~/rpi3/buildroot-2020.02.2/package/busybox/0003-compile-error-fix-stime.patch Generate root file systemEmling the root file system with the following command. The generated new filesystem is present as a tar file in the path ~/rpi3/buildroot-2020.02.2/output/imagesExtract the root filesystem takes file in some other directory.mkdir -p ~/rpi 3 /nfs_tmp tar -C ~/rpi3/nfs_tmp/ -xvf ~/rpi3/buildroot-2020.02.2/output/images/rootfs.tar Install Linux kernel modulesInstall they already arrived Compiled Linux modules to newly created root file system by running under commands.cd ~/rpi3/linux export ARCH=arm64 export CROSS_COMPILE=aarch64-rpi3-linux-gnu-export PATH=$PATH:~/x-tools/a garch64-rpi3-linux-gnu/bin/ make modules_install INSTALL_MOD_PATH=~/rpi3/nfs_tmp/ Insert the SD card into the system and copy the newly generated filesystem into the /dev/mmcblk0p2 partition. Unmount the partition if it is mounted already and then format the partition using the following command.sudo mkfs.ext4 /dev/mmcblk0p2 -L rootfs Delete and insert again into the system so that the mount point is /media/USERNAME/rootfs. Otherwise, it would be different if you mount it manually. Lsblk command helps you find the mounting point of partition.# Copy from rootfs directory to SD card sudo cp -r ~/rpi3/nfs_tmp/. /media/<USERNAME>/rootfs/ Replase UERENAME with yours. The copied root filesystem files have binaries and init scripts to start the SSH server automatically. If you want to add any additional packages, make menuconfig in the directory buildroot, select the required packages and follow the same steps mentioned in this post. Change the U-boot env valuesPower up the board. Stop at the U-boot console and add the following env variables to load RPI from SD card.setenv bootargs earlyprintk root=/dev/mmcblk0p2 rootfs=ext4 rootwait noinitrd setenv bootcmd 'mmc dev 0; fatload mmc 0:1 0x200000kernel.img; fatload mmc 0:1 0x200000 bcm2710-rpi-3-b.dtb; booti 0x2000000 - 0x200000;' saveenv Restart the boardInsert SD card to raspberry pi slot and restart the board. You should be able to log in to Raspberry Pi using SSH from your system. If you can't log in, it's likely that the Ethernet interface isn't up on the board. You can make link up and assign some static IP using the following commands in the baord.ifup -a ifconfig eth0 192.168.1.120 Now you should log on using the following command from your system. </USERNAME> </USERNAME> root password when prompted. Prompts.