SAM9X75-EB - Linux - Getting Started

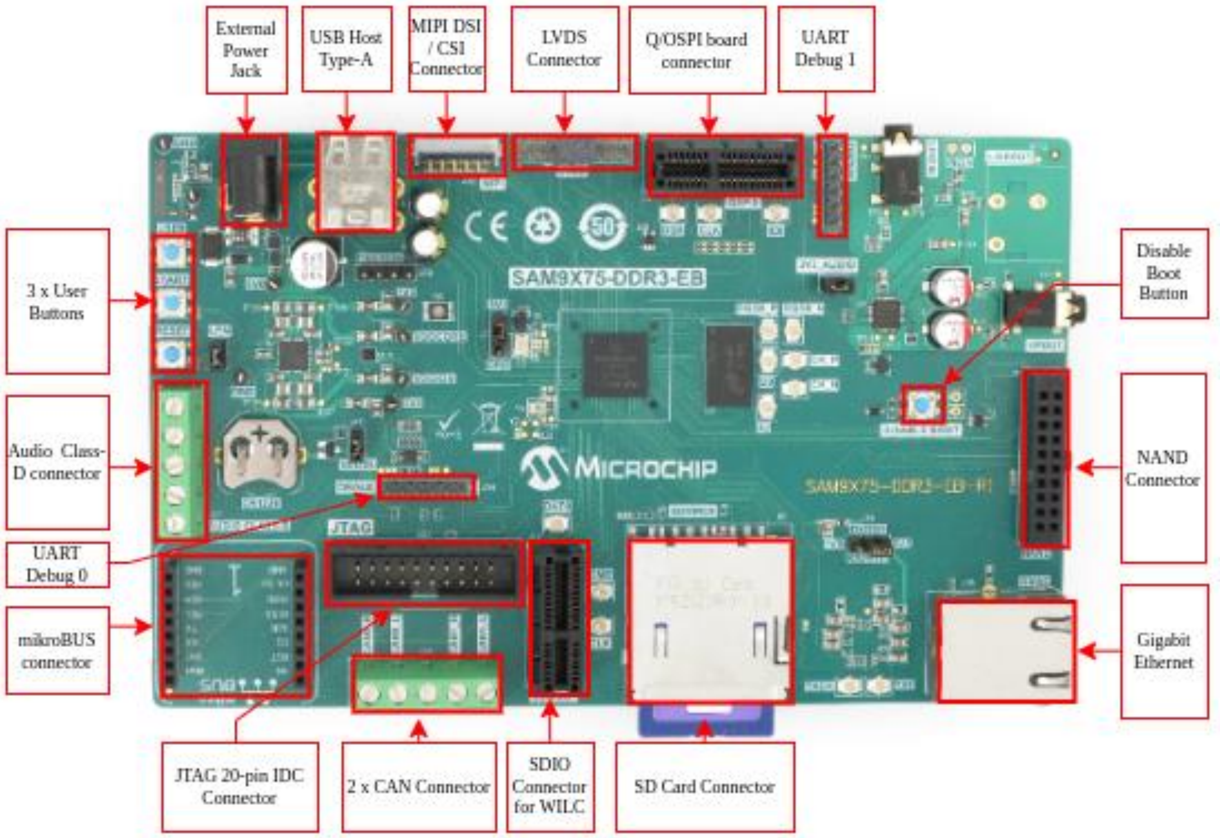
Table of Contents

Hardware Pre-requisite

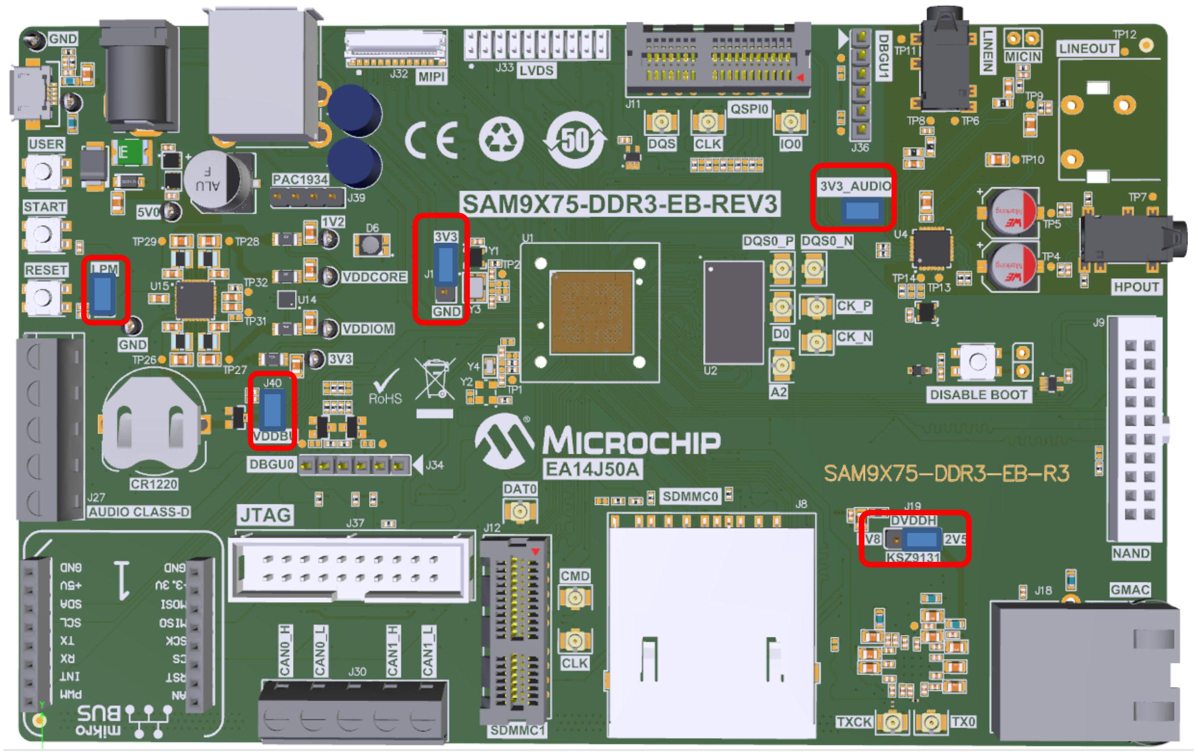
Here is the list of hardware required to complete this setup:

- The SAM9X75 EB board
- a Micro-USB cable
- a Serial to USB cable, also known as USB to TTL serial 3.3V cable (**optional**)

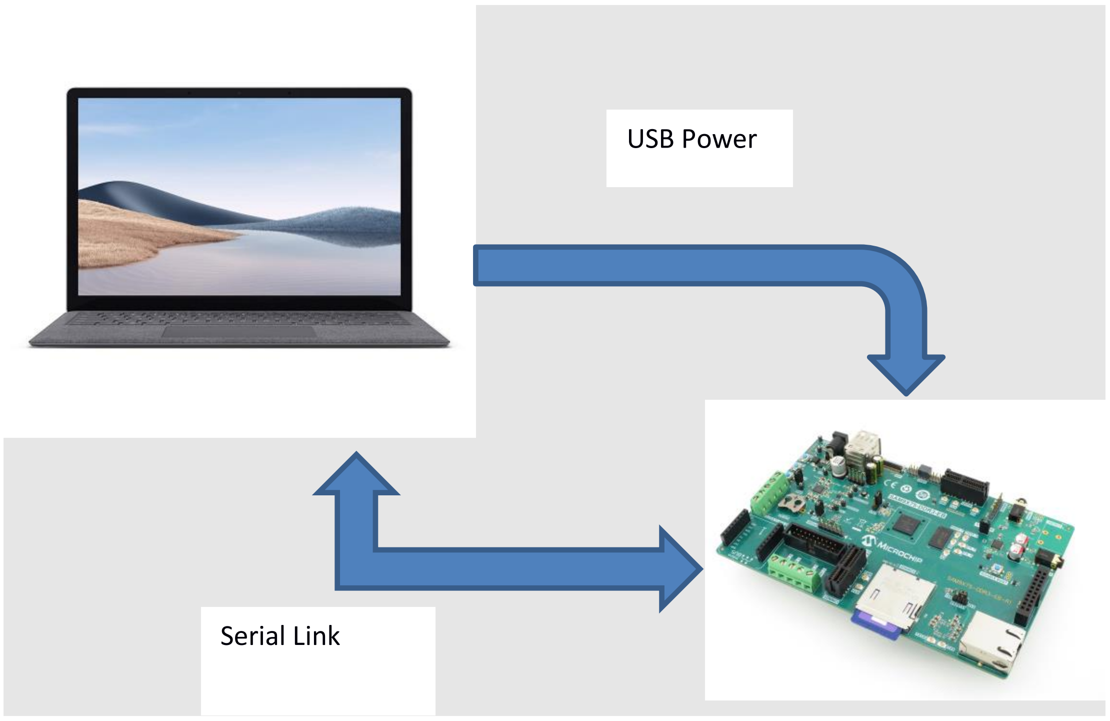
- The picture below shows the main connectors of the SAM9X75 EB board.



- The default jumper settings are shown below:



Hardware Connection



Serial Console

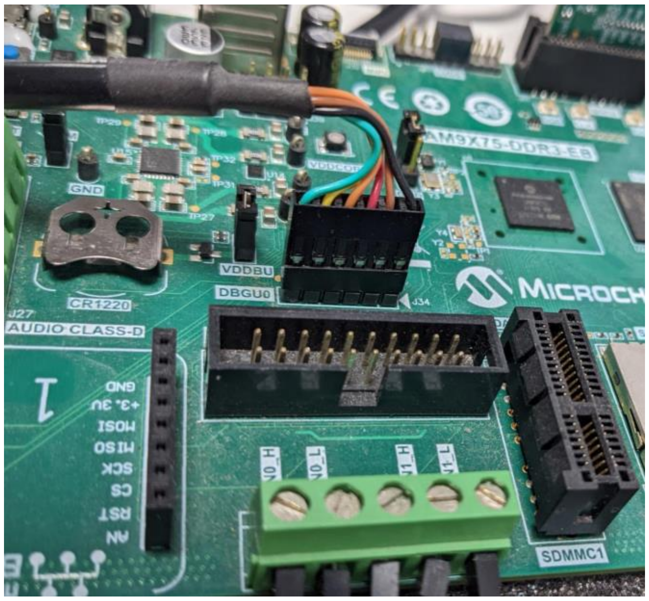
The usual serial communication parameters are 115200 8-N-1 :

Baud rate	115200
Data	8 bits
Parity	None
Stop	1 bit
Flow control	None

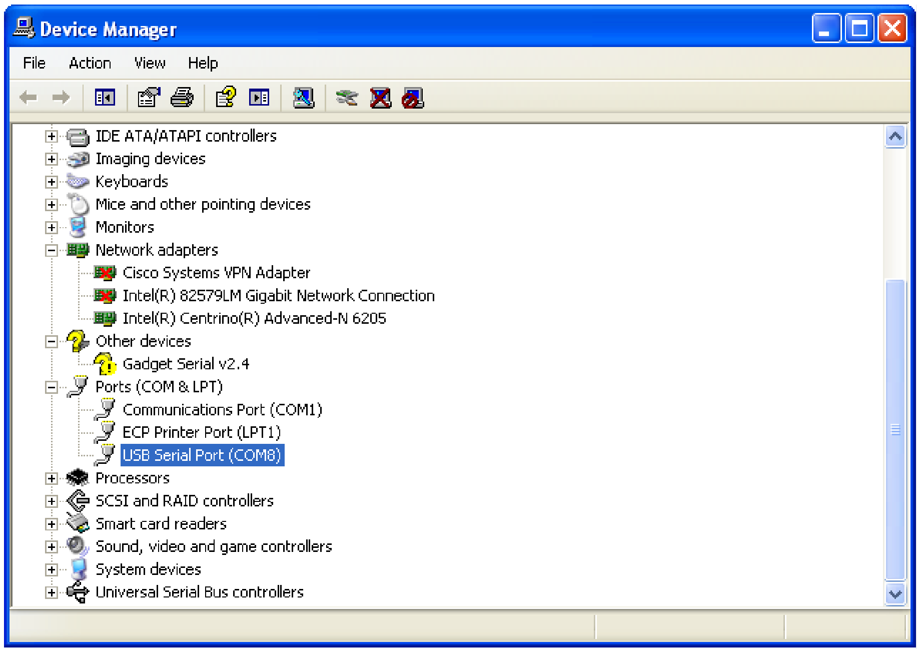
The serial console can be accessed from just one connector. This is from the DEBUG port with the help of a TTL-to-USB serial cable (marked as **J34 - UART Debug**).

Using TTL-to-USB connector (UART Debug J34)

- For Microsoft Windows users: Install the driver of your USB TTL serial cable. FTDI-based ones are the most popular, have a look at this page to get the driver: http://www.ftdichip.com/Drivers/VCP.htm
- Be sure to connect a 3.3V compatible cable and identify its GND pin. Place it properly according to the picture below, ground black wire close to the SDIO connector and connect the cable to the board (**J34**)



For **Microsoft Windows** users: Identify the USB connection that is established, USB Serial Port should appear in Device Manager. The COMxx number will be used to configure the terminal emulator.



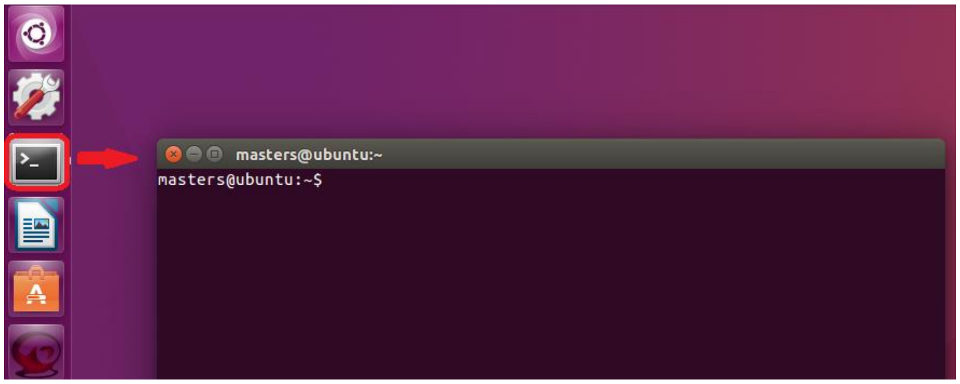
IDE ATA/ATAPI controllers
Imaging devices
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Cisco Systems VPN Adapter
Intel(R) 82579LM Gigabit Network Connection
Intel(R) Centrino(R) Advanced-N 6205
Other devices
Gadget Serial v2.4
Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
USB Serial Port (COM8)
Processors
SCSI and RAID controllers
Smart card readers
Sound, video and game controllers
System devices
Universal Serial Bus controllers

- For Linux users: Identify the serial USB connection by monitoring the last lines of **dmesg** command. The **/dev/ttyUSBx** number will be used to configure the terminal emulator

```
# dmesg
[605576.562740] usb 1-1.1.2: new full-speed USB device number 17 using ehci-pci
[605576.660920] usb 1-1.1.2: New USB device found, idVendor=0403, idProduct=6001
[605576.660933] usb 1-1.1.2: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[605576.660939] usb 1-1.1.2: Product: TTL232R-3V3
[605576.660944] usb 1-1.1.2: Manufacturer: FTDI
[605576.660958] usb 1-1.1.2: SerialNumber: FTGNVZ04
[605576.663092] ftdi_sio 1-1.1.2:1.0: FTDI USB Serial Device converter detected
[605576.663120] usb 1-1.1.2: Detected FT232RL
[605576.663122] usb 1-1.1.2: Number of endpoints 2
[605576.663124] usb 1-1.1.2: Endpoint 1 MaxPacketSize 64
[605576.663126] usb 1-1.1.2: Endpoint 2 MaxPacketSize 64
[605576.663128] usb 1-1.1.2: Setting MaxPacketSize 64
[605576.663483] usb 1-1.1.2: FTDI USB Serial Device converter now attached to
ttyUSB0
```

- A **/dev/ttyUSB0** node has been created.
- Now open your favorite terminal emulator with appropriate settings as indicated in the chapter just above. Below, you can see an example for Ubuntu:

- Launch the terminal on the host Linux PC Ubuntu by clicking on the Terminal icon. You can see in the image below how it would look like:



```
masters@ubuntu:~
masters@ubuntu:~$
```

Launching minicom serial terminal emulator

- Launch a terminal on the Linux host PC and type:

  $ **`minicom -D /dev/ttyUSB0 -b 115200`**

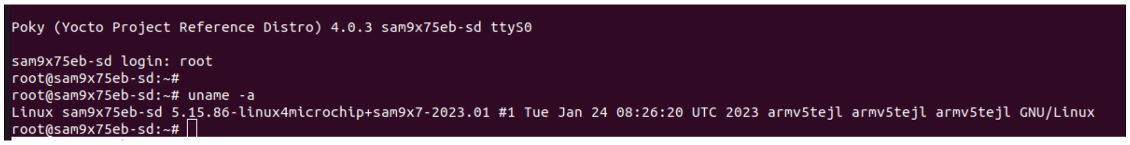
- You have now launched a serial terminal to access the target from the host PC

- If you do not see "sam9x75eb-sd login:" press "enter" and login as "root" (without password).

- To exit minicom type Ctrl-a and Q (not Ctrl-Q). You will be asked if you want to "Leave without reset?" Select "Yes" and hit enter.

- **You can connect to the board, and it will work, if you have the SD-Card with a flashed demo image in the slot. Have a look at the next chapter to see how to flash the demo image.**

- Your terminal will look similar to this:



```
Poky (Yocto Project Reference Distro) 4.0.3 sam9x75eb-sd ttyS0

sam9x75eb-sd login: root
root@sam9x75eb-sd:~#
root@sam9x75eb-sd:~# uname -a
Linux sam9x75eb-sd 5.15.86-linux4microchip+sam9x7-2023.01 #1 Tue Jan 24 08:26:20 UTC 2023 armv5tejl armv5tejl armv5tejl GNU/Linux
root@sam9x75eb-sd:~#
```

Demo

This demo is provided for the SAM9X75EB board only.

Demo archives

Media	Board	Binary	Description
SD Card image **Buildroot** based file system	SAM9X75-EB	linux4sam-buildroot-sam9x75eb-graphics-2023.01.img.bz2 MD5: 12dd680fb109c8c4eeab80a889a12f4b	SAM9X75-EB based graphics demo compiled from tag **sam9x7-early-2.0**
SD Card image **Buildroot** based file system	SAM9X75-EB	linux4sam-buildroot-sam9x75eb-headless-2023.01.img.bz2 MD5: b3370cdc03d63ec1e0f32bb746d21340	SAM9X75-EB based headless demo compiled from tag **sam9x7-early-2.0**
SD Card image **Yocto** based file system	SAM9X75-EB	linux4sam-poky-sam9x75eb-graphics-2023.01.img.bz2 MD5: f7205a573ed97763910f3ec6feb3ba77	SAM9X75-EB based graphics demo compiled from tag **sam9x7-early-2.0**
SD Card image **Yocto based file system**	SAM9X75-EB	linux4sam-poky-sam9x75eb-headless-2023.01.img.bz2 MD5: b8653fe985768deaa1ca92d6110e172f	SAM9X75-EB based headless demo compiled from tag **sam9x7-early-2.0**

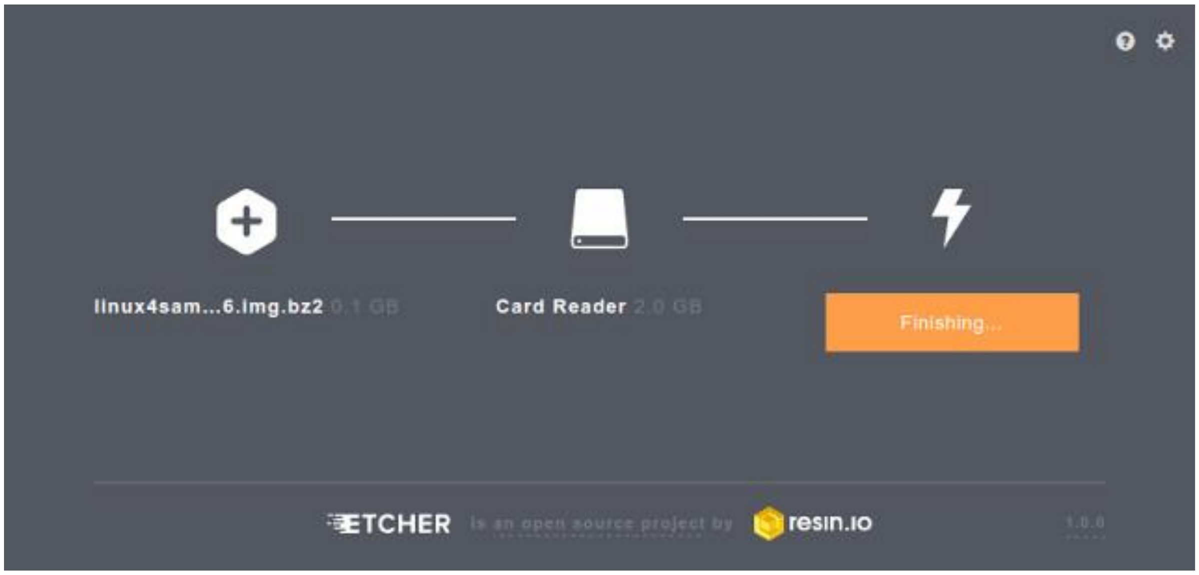
Create an SD card with the demo

It is a multi-platform procedure (Linux & Microsoft Windows)

You need a **1 GB SD card** (or more) and to download the image of the demo. The image is compressed to reduce the amount of data to download. This image contains:

- a FAT32 partition with the AT91Bootstrap, U-Boot and the Linux Kernel (FIT image).
- an EXT4 partition for the rootfs.

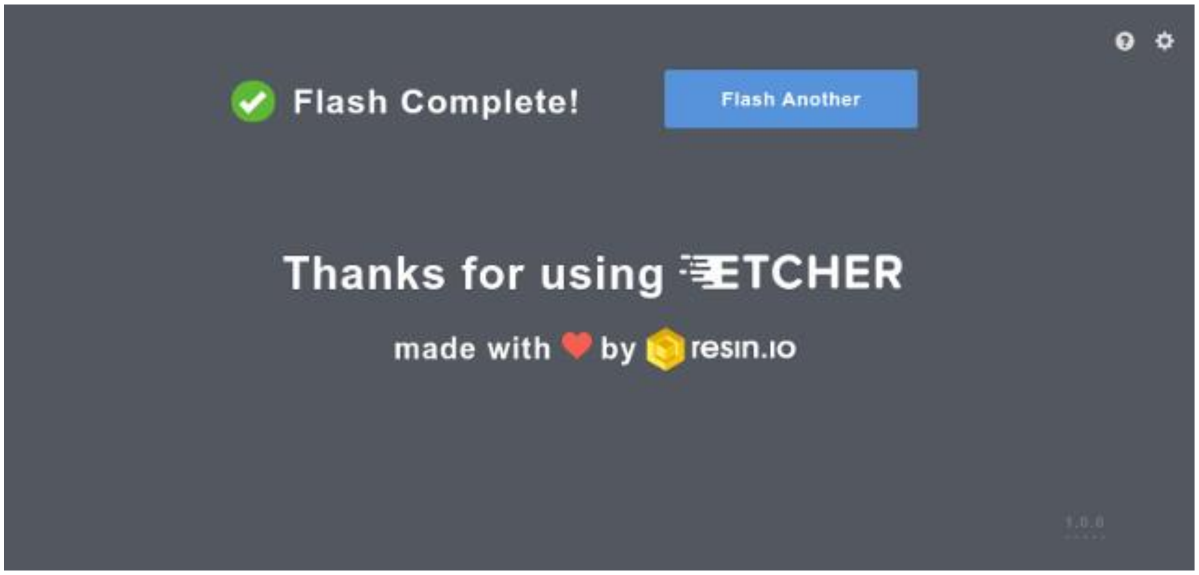
To write the compressed image on the SD card, you will have to download and install Etcher:


linux4sam...6.img.bz2 0.1 GB
Card Reader 2.0 GB
Finishing...
ETCHER is an open source project by resin.io
1.0.0

This tool, which is an Open Source software, is useful since it allows to get a compressed image as input. More information and extra help available here: https://www.balena.io/etcher/

- Insert your SD card and launch **Etcher**:
- Select the demo image. They are marked as "SD Card image" in the demo table above. Note that you can select a compressed image (like the demos available here). The tool can uncompress files on the fly.
- Select the device corresponding to your SD card (Etcher proposes you the devices that are removable to avoid erasing your system disk)
- Click on the Flash! button

On Linux, Etcher finally asks you to enter your root password because it needs access to the hardware (your SD card reader or USB to SD card converter)



- Once writing is done, Etcher asks you if you want to burn another demo image:
- Your SD card is ready!
- Now you can connect with your favorite terminal on the console and send/receive text commands to the board. Look at the previous chapter to see an example about how to start a terminal and connect.

Build From source code

Setup ARM Cross Compiler

- Ubuntu: you can install the ARM Cross Compiler by doing:

```
$ sudo apt-get install gcc-arm-linux-gnueabi
$ export CROSS_COMPILE=arm-linux-gnueabi-
```

- Other distributions: you can download the Linaro cross compiler and set up the environment by doing:

```
$ wget -c https://releases.linaro.org/components/toolchain/binaries/7.3-
2018.05/arm-linux-gnueabi/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-
gnueabi.tar.xz
$ tar xf gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabi.tar.xz
$ export CROSS_COMPILE=`pwd`/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-
gnueabi/bin/arm-linux-gnueabi-
```

Build AT91Bootstrap from sources

This section describes how to get source code from the git repository, how to configure with the default configuration, how to customize AT91Bootstrap based on the default configuration and finally to build AT91Bootstrap to produce the binary. Take the default configuration to download U-Boot from SD Card for example.

Get AT91Bootstrap Source Code

- You can easily download AT91Bootstrap source code on the at91bootstrap git repository.
- The source code is taken from the sam9x7_early branch which is dedicated to sam9x7 early developments.
- To get the source code, you should clone the repository by doing.

```
$ git clone https://github.com/linux4sam/at91bootstrap.git -b sam9x7_early
Cloning into 'at91bootstrap'...
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 11222 (delta 0), reused 1 (delta 0), pack-reused 11221
Receiving objects: 100% (11222/11222), 4.10 MiB | 3.78 MiB/s, done.
Resolving deltas: 100% (8586/8586), done.
$ cd at91bootstrap/
```

Configure AT91Bootstrap

- Assuming you are at the AT91Bootstrap root directory, you will find a configs folder which contains several default configuration files:

```
configs/sam9x75eb_bkptnone_defconfig
configs/sam9x75ebnf_linux_image_dt_defconfig
configs/sam9x75ebnf_uboot_defconfig
configs/sam9x75ebsd1_uboot_defconfig
configs/sam9x75ebsd_uboot_defconfig
```

Tips: nf means to read from nandflash, sd means to read SD-card, emmc means to read the eMMC on the board. Uboot means to load and start u-boot, none means to stop and load nothing, linux_image_dt to load and start the Linux kernel together with a devicetree blob.

- You can configure AT91Bootstrap to load U-Boot binary from SD Card by doing:

```
$ make mrproper
$ make sam9x75ebsd_uboot_defconfig
```

Customize AT91Bootstrap

- If the default configuration does not meet your need, after configuring with the default configuration, you can customize it by doing:

```
$ make menuconfig
```

- Now, in the menuconfig dialog, you can easily add or remove some features to/from AT91Bootstrap as the same way as kernel configuration. Move to “<Exit>” with arrows and press this button hitting the “Enter” key to exit from this screen.

Build AT91Bootstrap

- Then you can build the AT91Bootstrap binary by doing:

```
$ make
```

- If the building process is successful, the final .bin image is **build/binaries/at91bootstrap.bin**.

Build U-Boot from sources

Getting U-Boot sources

You can easily download U-Boot source code from Linux4SAM GitHub U-Boot repository:

- clone the Linux4sam GitHub U-Boot repository. The source code is taken from the sam9x7_early branch which is dedicated to sam9x7 early developments.

```
$ git clone https://github.com/linux4sam/u-boot-at91.git -b sam9x7_early
Cloning into 'u-boot-at91'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 567008 (delta 14), reused 22 (delta 8), pack-reused 566967
Receiving objects: 100% (567008/567008), 121.59 MiB | 3.28 MiB/s, done.
Resolving deltas: 100% (464150/464150), done.
$ cd u-boot-at91
```

Cross-compiling U-Boot

- Before compiling the U-Boot, you need to setup cross compile toolchain in the section.
- Latest versions of U-boot (2018.07 and newer) have a minimum requirement of 6.0 version of the GCC toolchain. We always recommend using the latest versions.
- Once the AT91 U-Boot sources are available, cross-compile U-Boot is made in two steps: configuration and compiling. Check the Configuration chapter in U-Boot reference manual.
- The U-Boot environment variables can be stored in different media. Above config files can specify where to store the U-Boot environment.

```
# To put environment variables in SD/MMC card:


sam9x75eb_mmc_defconfig
```

Here are the building steps:

```
#You can change the config according to your needs.


make sam9x75eb_mmc_defconfig

make
```

The result of these operations is a fresh U-Boot binary called **u-boot.bin** corresponding to the binary ELF file u-boot.

- **u-boot.bin** is the file you should store on the board
- **u-boot** is the ELF format binary file you may use to debug U-Boot through a JTAG link for instance.

Build Kernel from sources

Getting Kernel sources

- To get the source code, you have to clone the repository. The source code is taken from the linux-5.15-mchp+sam9x7 branch which is dedicated to sam9x7 early developments.

```
$ git clone https://github.com/linux4sam/linux-at91.git -b linux-5.15-
mchp+sam9x7
Cloning into 'linux-at91'...
remote: Enumerating objects: 1845, done.
remote: Counting objects: 100% (1845/1845), done.
remote: Compressing objects: 100% (553/553), done.
remote: Total 6456137 (delta 1424), reused 1589 (delta 1292), pack-reused
6454292
Receiving objects: 100% (6456137/6456137), 1.82 GiB | 905.00 KiB/s, done.
Resolving deltas: 100% (5408753/5408753), done.
Checking out files: 100% (61762/61762), done.
$ cd linux-at91
```

Configure and Build the Linux kernel

- Now you have to configure the Linux kernel according to your hardware. We have three at91 SOC default configurations in arch/arm/configs:

  ```
  arch/arm/configs/at91_dt_defconfig
  arch/arm/configs/sama5_defconfig
  arch/arm/configs/sama7_defconfig
  ```

  - at91_dt_defconfig: for at91sam ARM926 series chips
  - sama5_defconfig: for SAMA5 series chips
  - sama7_defconfig: for SAMA7 series chips

- Now we Configure and Build kernel for sam9x75 eb board:

```
$ make ARCH=arm at91_dt_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

- And build the Linux kernel image, before you build you need set up the cross-compile toolchain, check this section.

```
$ make ARCH=arm
[..]
Kernel: arch/arm/boot/Image is ready
Kernel: arch/arm/boot/zImage is ready
```

- Now you have a usable compressed kernel image zImage.
- If you need to recompile the device tree you can run this command:

```
make ARCH=arm dtbs
DTC     arch/arm/boot/dts/at91-sam9x75eb.dtb
```

How to build Buildroot for AT91

Prerequisites

- Host build system should be a Linux system with necessary software installed: http://buildroot.uclibc.org/downloads/manual/manual.html#requirement

- Note You can install missing packages using yum install with Fedora or apt-get install with Ubuntu or Debian. These commands may require root privileges or being in a correct sudoers group.

Get sources

- To get the source code, you have to clone the **buildroot-at91 and buildroot-external-microchip** repositories. buildroot-at91 is a fork of Buildroot with a minimal number of patches. The external tree provides stuff which will not hit the mainline: additional defconfigs and packages dedicated to our demos.
- The source code is taken based on the sam9x7_early branch based on 2022.02-at91 which have all the up-to-date features for the AT91 MPUs.

```
$ git clone https://github.com/linux4sam/buildroot-at91.git
https://github.com/linux4sam/buildroot-at91.git -b sam9x7_early
Cloning into 'buildroot-at91'...
remote: Counting objects: 271126, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 271126 (delta 0), reused 2 (delta 0), pack-reused 271124
Receiving objects: 100% (271126/271126), 61.00 MiB | 2.36 MiB/s, done.
Resolving deltas: 100% (186357/186357), done.

$ ls buildroot-at91/
arch    CHANGES           configs     dl    linux            output    support
utils
board   Config.in         COPYING     docs  Makefile         package   system
boot    Config.in.legacy  DEVELOPERS  fs    Makefile.legacy  README    toolchain


$ git clone https://github.com/linux4sam/buildroot-external-microchip.git
https://github.com/linux4sam/buildroot-external-microchip.git -b sam9x7_early
Cloning into 'buildroot-external-microchip'...
remote: Counting objects: 571, done.
remote: Compressing objects: 100% (159/159), done.
remote: Total 571 (delta 217), reused 288 (delta 170), pack-reused 225
Receiving objects: 100% (571/571), 94.41 KiB | 76.00 KiB/s, done.
Resolving deltas: 100% (332/332), done.

$ ls buildroot-external-microchip/
board      configs  docs           external.mk  patches    system
Config.in  COPYING  external.desc  package      README.md
```

Build the rootfs image

- To build the the rootfs image that we provide for sam9x75 eb, you will have to do:

```
$ cd buildroot-at91
$ BR2_EXTERNAL=../buildroot-external-microchip/ make
sam9x75eb_graphics_defconfig
$ make
```

- Once compilation is done, have a look to output/images to see what has been generated:

```
$ ls -la output/images/
drwxr-xr-x 2 varshini varshini      4096 Jul  5 18:45 .
drwxrwxr-x 6 varshini varshini      4096 Jul  5 18:45 ..
-rwxr-xr-x 1 varshini varshini     18721 Jul  5 18:18 at91bootstrap.bin
-rwxr-xr-x 1 varshini varshini     25141 Jul  5 18:17 at91-sam9x75eb.dtb
-rwxr-xr-x 1 varshini varshini     18721 Jul  5 18:18 boot.bin
-rw-r--r-- 1 varshini varshini  16777216 Jul  5 18:45 boot.vfat
-rw-r--r-- 1 varshini varshini 943718400 Jul  5 18:45 rootfs.ext2
lrwxrwxrwx 1 varshini varshini        11 Jul  5 18:45 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 varshini varshini 354447360 Jul  5 18:45 rootfs.tar
-rw-r--r-- 1 varshini varshini   5427492 Jul  5 18:18 sam9x75eb.itb
-rwxr-xr-x 1 varshini varshini     18721 Jul  5 18:18 sam9x7-sdcardboot-uboot-
4.0.0.bin
-rw-r--r-- 1 varshini varshini 961544192 Jul  5 18:45 sdcard.img
-rw-r--r-- 1 varshini varshini    416580 Jul  5 18:10 u-boot.bin
-rwxr-xr-x 1 varshini varshini     16384 Jul  5 18:11 uboot-env.bin
-rw-r--r-- 1 varshini varshini   5400344 Jul  5 18:17 zImage
```

- All software components are there: booloaders, kernel, device tree and rootfs.
- The SD Card image ready to be flashed: sdcard.img
- If you do a make clean, you will delete the rootfs but also the cross toolchain.
- Have a look at the first and second chapters to see how you can flash the image and then connect to the board using a serial console terminal.

How to build Poky for AT91

Note that building an entire distribution is a lengthy process. It also requires a big amount of free disk space.
The support for Atmel AT91 SoC family is included in a particular Yocto layer: meta-atmel. The source for this layer is hosted on GitHub.

Building environment

A step-by-step comprehensive installation is explained in the guide:
https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html
The following lines have to be considered as an add-on that is AT91 specific or that can facilitate your setup.

Prerequisite

Here are the reference pages for setting up a Yocto building environment: What You Need and How You Get It.

Step by step build procedure

Here is a copy of the procedure available directly in the meta-atmel layer. This file in the meta-atmel layer repository must be considered as the reference and the following copy can be out-of-sync.

```
This layer provides support for Microchip microprocessors (aka AT91)

====================================================================

For more information about the Microchip MPU product line see:

http://www.microchip.com/design-centers/32-bit-mpus

Linux & Open Source on Microchip microprocessors:

http://www.linux4sam.org

Supported SoCs / MACHINE names

==============================

Note that most of the machine names below, have a SD Card variant that can be

built by adding an "-sd" suffix to the machine name.

- SAMA5D2 product family / sama5d2-xplained, sama5d2-xplained-emmc, sama5d27-
som1-ek-sd, sama5d2-ptc-ek, sama5d2-icp, sama5d27-wlsom1-ek-sd

- SAMA5D4 product family / sama5d4ek, sama5d4-xplained

- SAMA5D3 product family / sama5d3xek, sama5d3-xplained
- AT91SAM9x5 product family (AT91SAM9G15, AT91SAM9G25, AT91SAM9X25, AT91SAM9G35
and AT91SAM9X35) / at91sam9x5ek

- AT91SAM9RL / at91sam9rlek

- AT91SAM9G45 / at91sam9m10g45ek

- SAM9X60 / sam9x60ek

- SAMA7G5 / sama7g5ek-sd

-SAM9X75 / sam9x75eb

SAM9X75 / sam9x75eb -sd


Sources

=======

- meta-atmel
```

URI: https://github.com/linux4sam/meta-atmel.git
Branch: sam9x7_early
Tag: sam9x7-early-2.0

```
Dependencies

============

This Layer depends on:
```

- poky
URI: https://git.yoctoproject.org/poky
Branch: kirkstone
Tag:kirkstone-4.0.3

- meta-openembedded
URI: https://git.openembedded.org/meta-openembedded
Branch: kirkstone
Tag:744a4b6eda88b9a9ca1cf0df6e18be384d9054e3

- meta-arm (for optee components)
URI: https://git.yoctoproject.org/meta-arm
Branch: kirkstone
Tag:yocto-4.0.1

```
Build procedure
===============


0/ Create a directory
mkdir my_dir
cd my_dir


1/ Clone yocto/poky git repository with the proper branch ready
git clone git://git.yoctoproject.org/poky -b dunfell


2/ Clone meta-openembedded git repository with the proper branch ready
git clone git://git.openembedded.org/meta-openembedded -b dunfell


3/ Clone meta-aws git repository with the proper branch ready
git clone git://github.com/aws/meta-aws -b dunfell


4/ Clone meta-atmel layer with the proper branch ready
git clone git://github.com/linux4sam/meta-atmel.git -b dunfell


5/ Enter the poky directory to configure the build system and start the build
process
cd poky


6/ Change TEMPLATECONF from .templateconf to:
export TEMPLATECONF=${TEMPLATECONF:-../meta-atmel/conf}
Note: If it is the first time you use Yocto Project templates, and if the
build-microchip directory remains from a previous use, we advice you start
from a fresh directory. Keep your build-microchip/conf/local.conf file for
reference.


7/ Initialize build directory
```

```
source oe-init-build-env build-microchip


8/ To build a small image provided by Yocto Project:

[MACHINE=<machine>] bitbake core-image-minimal


Example for sama5d2-xplained-sd SD card image:

MACHINE=sama5d2-xplained-sd bitbake core-image-minimal


9/ To build the microchip image with no graphics support:

[MACHINE=<machine>] bitbake microchip-headless-image


Example for sama5d2-xplained-sd SD card image:

MACHINE=sama5d2-xplained-sd bitbake microchip-headless-image


10/ To build the microchip image with graphics support (EGT):

[MACHINE=<machine>] bitbake microchip-graphics-image


Example for sama5d2-xplained-sd SD card image:

MACHINE=sama5d2-xplained-sd bitbake microchip-graphics-image
```

Typical bitbake output:

```
Build Configuration:
 BB_VERSION           = "2.0.0"
 BUILD_SYS            = "x86_64-linux"
 NATIVELSBSTRING      = "ubuntu-20.04"
 TARGET_SYS           = "arm-poky-linux-gnueabi"
 MACHINE              = "sam9x75eb-sd"
 DISTRO               = "poky-atmel"
 DISTRO_VERSION       = "4.0.3"
 TUNE_FEATURES        = "arm armv5 thumb dsp"
 TARGET_FPU           = "soft"
 meta
 meta-poky
```

```
meta-yocto-bsp       = "kirkstone:387ab5f18b17c3af3e9e30dc58584641a70f359f"
meta-oe
meta-networking
meta-webserver
meta-python
meta-initramfs       = "kirkstone:744a4b6eda88b9a9ca1cf0df6e18be384d9054e3"
meta-atmel           = "sam9x7_early:5e1090acdf61c070f6fa47e1ba49e0fa702ef1b4"
meta-multimedia      = "kirkstone:744a4b6eda88b9a9ca1cf0df6e18be384d9054e3"
meta-arm
meta-arm-toolchain   = "kirkstone:bafd1d013c2470bcec123ba4eb8232ab879b2660"
```

Tips & tricks

BitBake:

- Yocto reference manual: BitBake brief description
- Yocto reference manutal: BitBake chapter

List tasks provided by a package:
bitbake -c listtasks <package_name>
You can use one of those tasks to have a fine-grained control over the package building.

- http://elinux.org/Bitbake_Cheat_Sheet

The structure of a BitBake file explained: http://docs.openembedded.org/bitbake/html/ch02.html

Feature list

1. AT91Bootstrap
    a. Basic support
    b. PMIC
    c. DDR3
    d. SD-card
    e. CPU clocked at 800MHz

2. U-Boot
    a. Basic support
    b. Gigabit Ethernet
    c. SD-Card
    d. PIT64B
    e. PIO
    f. PMC
    g. NAND Memory access

3. Linux Kernel
    a. Basic support
    b. Serial console
    c. PMC
    d. PIO
    e. SD-Card support
    f. Gigabit Ethernet
    g. Gigabit PTP
    h. MIPI DSI Display
    i. Touchscreen
    j. USB Host & Device
    k. PIT64B
    l. Buttons and LEDs
    m. MIPI CSI Camera
    n. Class-D Speaker
    o. I2S Audio
    p. NAND Memory access
    q. CAN

4. DT-Overlay-AT91
    a. Basic support
    b. MIPI CSI Camera

Known limitations

1. The touch screen works in polling mode.