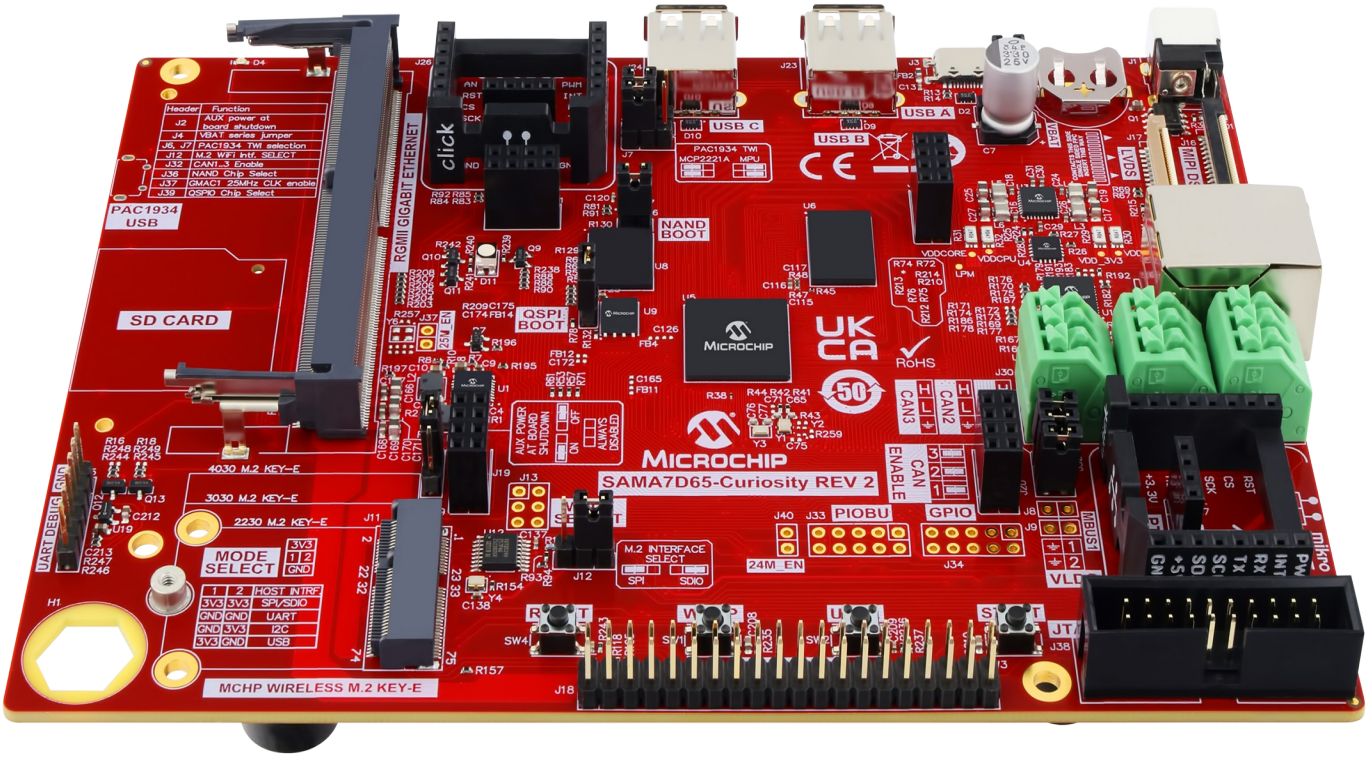
SAMA7D65 - Linux - Getting Started



Document version 1.0.

Table of Contents

Revision History

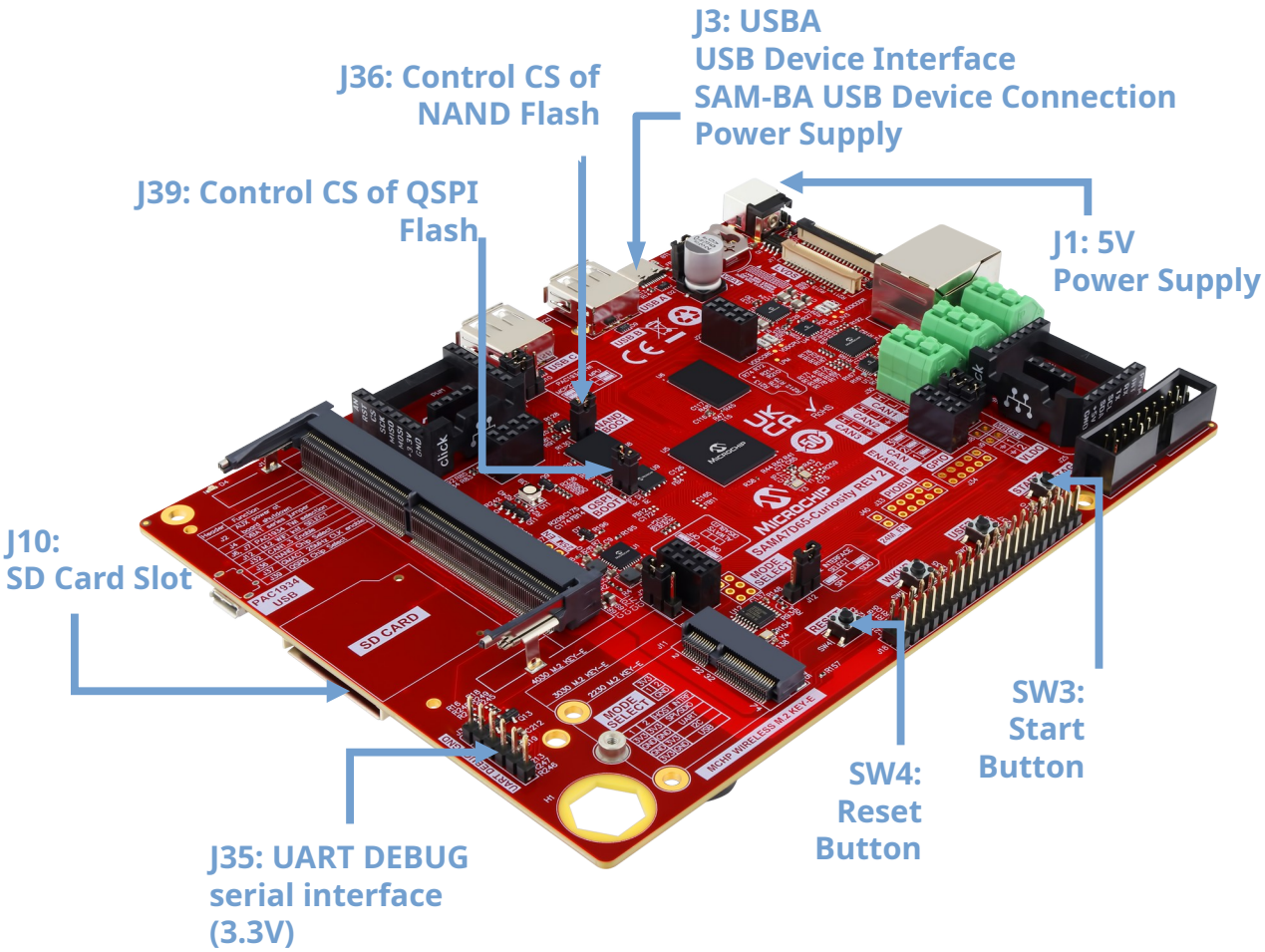
Date	Version	Comments
June 11th, 2024	1.0	Initial document

Hardware Pre-requisites

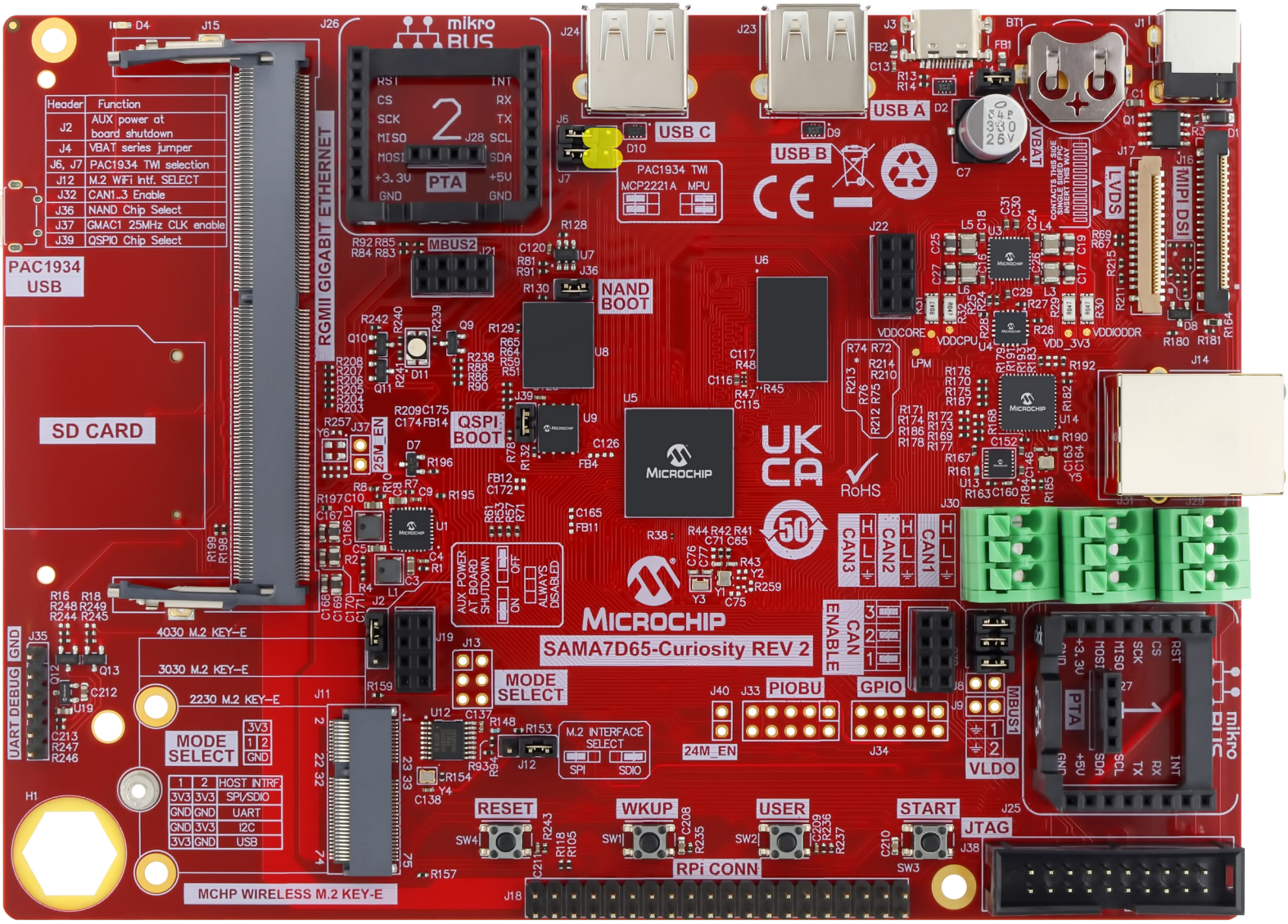
Here is the list of hardware required to complete this setup:

- The SAMA7D65 Curiosity board;
- A power delivery cable (5V) or a USB type-A to type-C 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 SAMA7D65 Curiosity board.



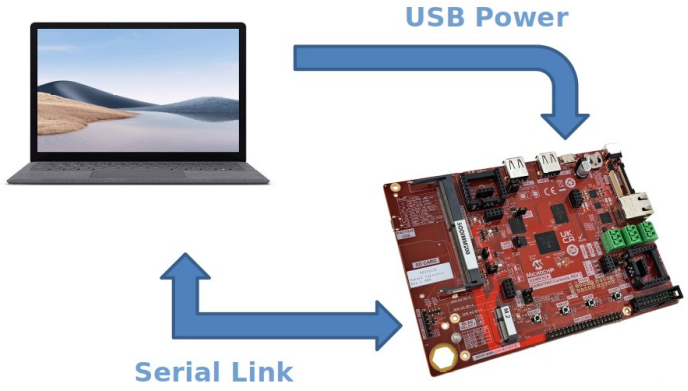
The default jumper settings are shown below:



Hardware Connection

Connect to the board through the serial link to follow the boot process and read the boot log. The Linux console and primary terminal interface is directed to this UART DEBUG serial link.

Power the board by using the dedicated barrel connector (J1: 5V) or using the USB type-A to type-C connector (J3).



Serial Console

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

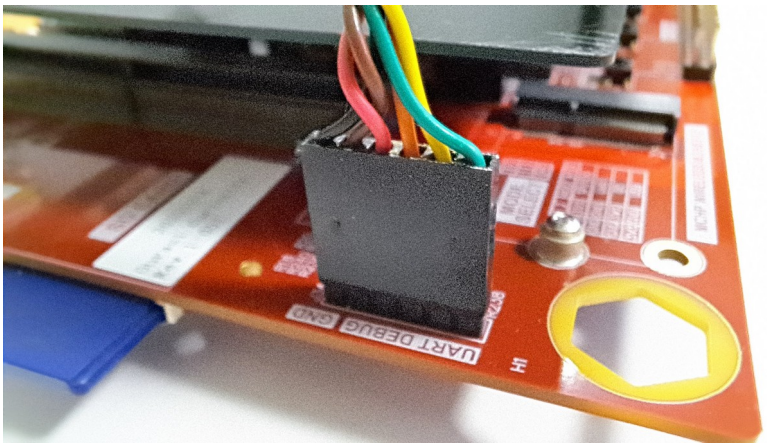
Baud rate	115200
Data	8 bits
Parity	None
Stop	1 bit
Flow control	None

The serial console can be accessed from just one connector. This is from the UART DEBUG port with the help of a TTL-to-USB serial cable (marked as **J35 - UART DEBUG**).

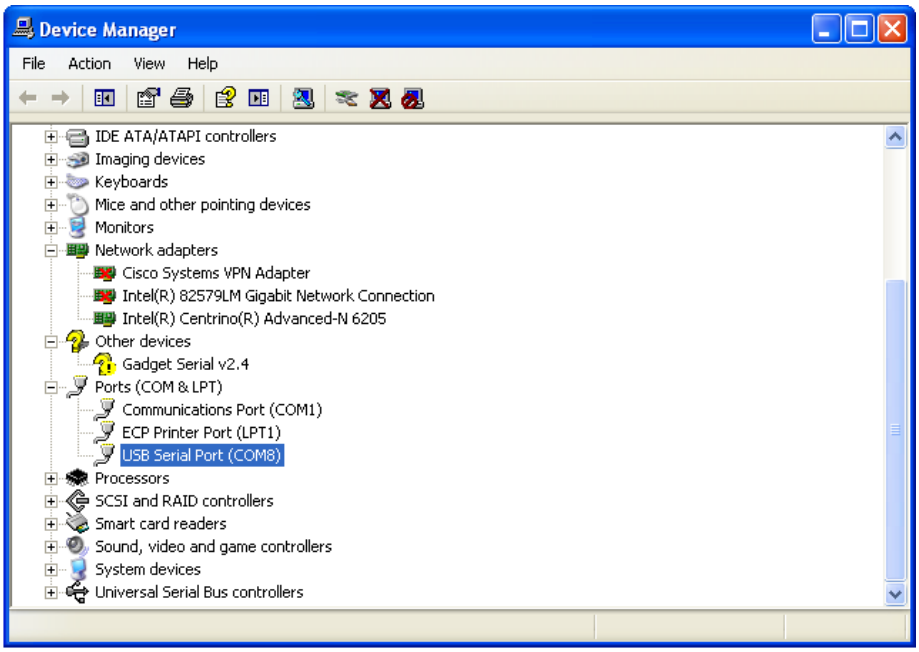
Use TTL-to-USB connector (UART Debug J35)

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 find its GND pin. Place it properly according to the picture below, ground black wire close to the SD CARD connector and connect the cable to the board (**J35**)



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.



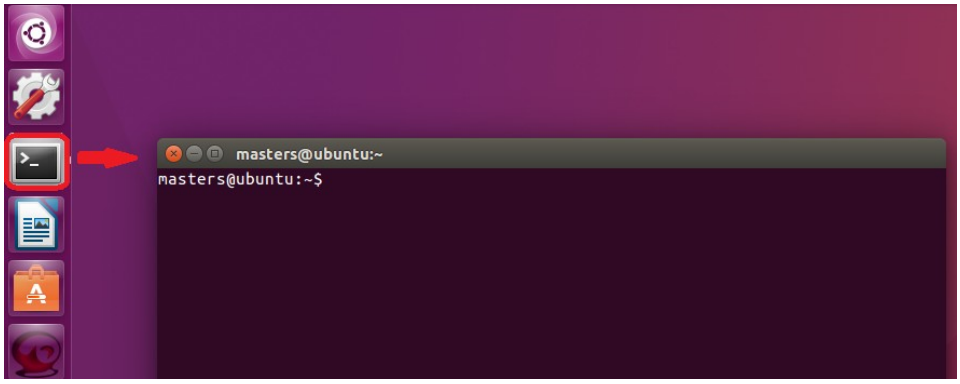
For Linux users: Identify the serial USB connection by monitoring the last lines of **dmesg** command. The **/dev/ttyUSBx** number will be used to configure the terminal emulator:

```
# 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 proper settings as showed 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:



Launching picocom serial terminal emulator

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

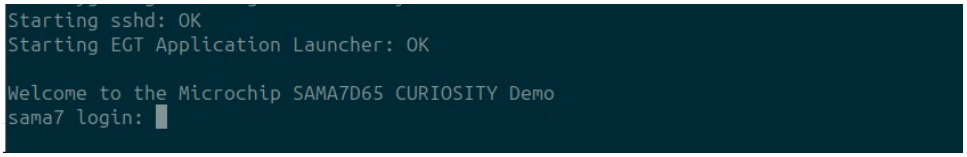
  `$ picocom -b 115200 /dev/ttyUSB0`

- You have now launched a serial terminal to access the target from the host PC
- If you do not see " sama7 login: ” press "enter" and login as “root” (without password).

- To exit picocom hold CTRL-A then press Q.

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

Your terminal will look like this:



Software pre-requisites

Compilation requirements

For cross-compiling bootloaders, the Linux kernel or starting a build-system, minimum requirements are needed on your host machine. A development machine needs the required packages mentioned in the documentation below. Please make sure to follow these links, install mentioned software depending on your distributions, and make sure that they are well installed:

- Build dependencies for the Linux Kernel (also applies to bootloaders)
- Build dependencies for Buildroot
- coreutils and libncurses-dev packages should be installed too.

SAM-BA tool

SAM-BA is required to add a header in at91bootstrap image (for both manual compilation and Buildroot). Download SAM-BA software with the following link:
https://github.com/atmelcorp/sam-ba/releases/tag/v3.8

Uncompress the tgz file in your workspace with:

```
$ tar xvzf sam-ba_v3.8-linux_x86_64
```

Make sure to add the sam-ba application to your $PATH and verify that you have the correct version:

```
$ sam-ba -v

SAM-BA Command Line Tool v3.8

Copyright 2024 Microchip Technology
```

Note: this tool was tested with distribution Ubuntu 22.04 onward.

It is known to **not** work on Ubuntu 20.04 and previous ones, generating such kind of issues:

```
sam-ba: /lib64/libc.so.6: version `GLIBC_2.34' not found (required by sam-ba)
```

Demo

This demo is provided for the SAMA7D65 Curiosity board only.

Demo archives

Media	Board	Binary		Description
Buildroot based file system SD Card Image	SAMA7D65 Curiosity	Headless	linux4sam-buildroot-sama7d65_curiosity-headless-sama7d65-ea-1.0.img.bz2 MD5: a2343e81464dfd96fbeadd826a226c54	Buildroot-at91 based demo compiled from tag: **sama7d65-ea-1.0**
		Graphics	linux4sam-buildroot-sama7d65_curiosity-graphics-ea-1.0.img.bz2 MD5: aac6bd154fde3497b89b30a27a7188f8	
Yocto/Pocky Based Demo SD Card Image	SAMA7D65 Curiosity	Headless	linux4sam-poky-sama7d65_curiosity-headless-sama7d65-ea-1.0.img.bz2 MD5: 930649f5ef961e3982858f379eb3440d	Yocto Project/Poky based demo compiled from tag: **sama7d65-ea-1.0**
		Graphics	linux4sam-poky-sama7d65_curiosity-graphics-sama7d65-ea-1.0.img.bz2 MD5: 759b6a503f168df380e275204226644a	

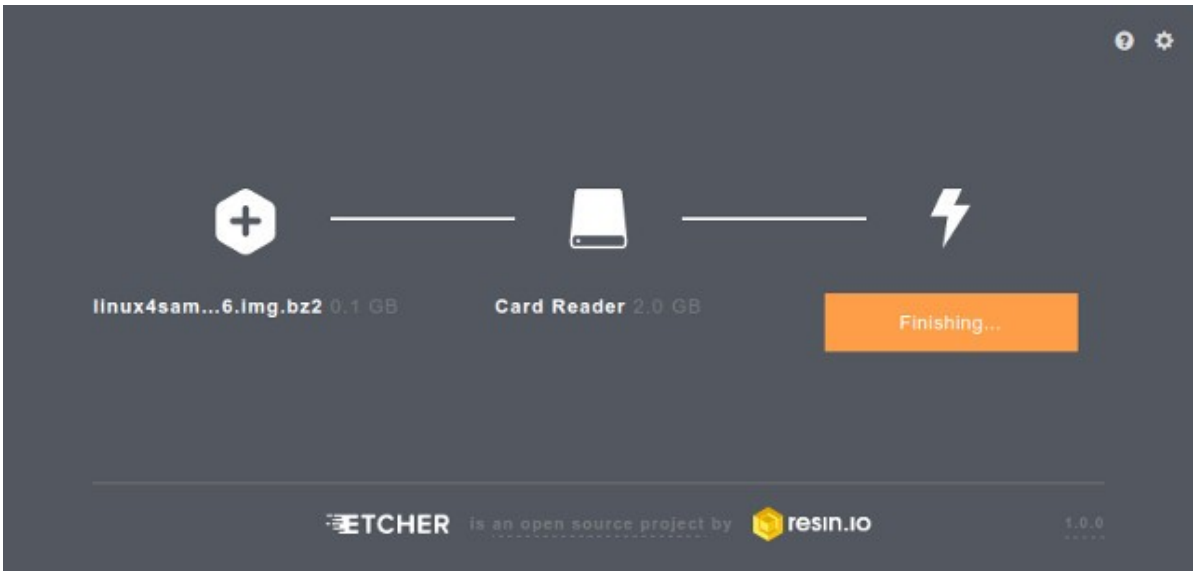
Create an SD card with the demo

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 has:

- a FAT32 partition with the AT91Bootstrap, U-Boot the Linux Kernel and the Device Tree file for the board (.dtb).
- an EXT4 partition for the root filesystem (rootfs).

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



This tool, which is an Open-Source software, is useful since it allows you 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-gnueabihf
$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

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.

Get AT91Bootstrap Source Code

You can download AT91Bootstrap source code from the GitHub repository:
https://github.com/linux4sam/at91bootstrap/tree/sama7d65_ea

The source code is taken from the sama7d65_ea branch which is dedicated to sama7d65 early developments.

To get the source code, you should clone the repository by doing:

```
$ git clone https://github.com/linux4sam/at91bootstrap.git -b sama7d65_ea
Cloning into 'at91bootstrap'...
remote: Enumerating objects: 14778, done.
remote: Counting objects: 100% (7289/7289), done.
remote: Compressing objects: 100% (1581/1581), done.
remote: Total 14778 (delta 5816), reused 6968 (delta 5656), pack-reused 7489
Receiving objects: 100% (14778/14778), 3.84 MiB | 9.19 MiB/s, done.
Resolving deltas: 100% (11611/11611), done.
$
```

Then enter the resulting directory to be ready to use AT91Bootstrap source code:

```
$ cd at91bootstrap/
```

Configure AT91Bootstrap

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

```
$ ls -1 configs/sama7d*
configs/sama7d65_curiosity-bsrsd1_uboot_defconfig
```

Tips: nf means to read the next binary from nandflash, df means to read from SPI/QSPI flash, sd means to read from SD-card, bsr means backup mode is enabled. The string "_uboot_" means to load and start u-boot as the next bootloader.

You can configure AT91Bootstrap to load U-Boot binary from SD Card 1 by doing:

```
$ make mrproper
$ make sama7d65_curiosity-bsrsd1_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

First, make sure that your CROSS_COMPILE environment variable is well defined:

```
$ echo $CROSS_COMPILE
arm-linux-gnueabihf-
$
```

If the resulting line after the "echo" command is empty, please make sure to follow the instructions in the Setup ARM Cross Compiler chapter above.

Then you can build the AT91Bootstrap binary by running "make":

```
$ make
CC
========
arm-linux-gnueabihf-gcc 11

as FLAGS
========
-g -Os -Wall -Idevice/sama7d65 -Iinclude -include config/at91bootstrap-config/autoconf.h -
DJUMP_ADDR=0x66f00000 -DTOP_OF_MEMORY=0x120000 -DMACH_TYPE=9999  -DLINK_ADDR="0x100000" -
DMACH_TYPE=9999 -DTOP_OF_MEMORY=0x120000 -mcpu=cortex-a7 -mtune=cortex-a7 -mfpu=vfpv4-d16

gcc FLAGS
=========
-nostdinc -isystem "/usr/lib/gcc-cross/arm-linux-gnueabi/11/include" -ffunction-sections -g -Os -Wall
-mno-unaligned-access -fno-stack-protector -fno-common -fno-builtin -fno-jump-tables -fno-pie -
Idevice/sama7d65 -Iinclude -Ifs/include -Iconfig/at91bootstrap-config -include config/at91bootstrap-
config/autoconf.h -DAT91BOOTSTRAP_VERSION=\"4.0.7-sama7d65-beta\" -DCOMPILE_TIME="\"2024-01-31
17:06:31\"" -DIMG_ADDRESS= -DIMG_SIZE= -DMMU_TABLE_BASE_ADDR= -DJUMP_ADDR=0x66f00000 -DOF_OFFSET= -
DOF_ADDRESS= -DCMDLINE_FILE="\"\"" -DTOP_OF_MEMORY=0x120000 -DMACH_TYPE=9999  -DCMDLINE="\"\"" -
DIMAGE_NAME="\"u-boot.bin\"" -DCONFIG_DEBUG -DBANNER="""\"\\n\\nAT91Bootstrap \" AT91BOOTSTRAP_VERSION
\" (\" COMPILE_TIME \")\\n\\n\"""" -DBOARD_MAINOSC=24000000 -DMACH_TYPE=9999 -DTOP_OF_MEMORY=0x120000
-mcpu=cortex-a7 -mtune=cortex-a7 -mfpu=vfpv4-d16 -DCONFIG_PUBL -DCONFIG_MEM_CLOCK=533 -DCONFIG_DDR3 -
DCONFIG_DDR_SPEED_1066 -DCONFIG_DDR_8_GBIT -DCONFIG_PMC_MCK_CLK -DBOOTSTRAP_DEBUG_LEVEL=DEBUG_INFO -
DCONFIG_TZC400 -DCONFIG_TZC400_SIMPLE_PROFILE -DCONFIG_WDTS
```

```
ld FLAGS
========
 -Map=./build/binaries/sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea.map --cref -static -z noexecstack
--no-warn-rwx-segments -T elf32-littlearm.lds --gc-sections -Ttext 0x100000

  AS        crt0_gnu.S
  CC        main.c
  CC        device/sama7d65/sama7d65.c
  CC        lib/string.c

[..]

  CC        fs/src/option/ccsbcs.c
  LD        sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea.elf
Size of sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea.bin is 22668 bytes
[Succeeded] It's OK to fit into SRAM area
[Attention] The space left for stack is 108404 bytes
[Succeeded] SAM-BA tool generating bootable binary image available
Generating 'build/binaries/sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea-plaintextimg.bin' version '0.0'
from 'build/binaries/sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea.bin'
Appending 11 bytes of padding to fill the last written page
```

If the building process is successful, the final .bin images are **build/binaries/boot.bin AND build/binaries/boot-plaintextimg.bin**. This last one, boot-plaintextimg.bin, is the one to be used for booting the sama7d65 SoC.

Build AT91Bootstrap for LPDDR2 Boards

Get AT91Bootstrap Source Code

You can download AT91Bootstrap source code from the GitHub repository:
https://github.com/linux4sam/at91bootstrap/commits/at91bootstrap-4.x%2Bsama7d65-ea-1.0/

The source code is taken from the at91boostrap-4.x+sama7d65-ea branch which is dedicated to sama7d65 early development for **LPDDR2** and **SiP** variant **only**.

To get the source code, you should clone the repository by doing:

```
$ git clone https://github.com/linux4sam/at91bootstrap.git -b \
at91bootstrap-4.x+sama7d65_ea at91bootstrap-4.x+sama7d65-ea-1.0
Cloning into 'at91bootstrap-4.x+sama7d65-ea-1.0'...
remote: Enumerating objects: 19145, done.
remote: Counting objects: 100% (4848/4848), done.
remote: Compressing objects: 100% (1314/1314), done.
remote: Total 19145 (delta 3699), reused 4598 (delta 3524), pack-reused 14297
Receiving objects: 100% (19145/19145), 5.84 MiB | 7.86 MiB/s, done.
Resolving deltas: 100% (14693/14693), done.
$
```

Then enter the resulting directory to be ready to use AT91Bootstrap source code:

```
$ cd at91bootstrap-4.x+sama7d65-ea-1.0/
```

Configure AT91Bootstrap

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

```
$ ls -1 configs/ourasi_lpddr2*
configs/ourasi_lpddr2_eb_bkptnone_defconfig
configs/ourasi_lpddr2_ebsd0_uboot_defconfig
configs/ourasi_lpddr2_ebsd1_uboot_defconfig
configs/ourasi_lpddr2_ebsd2_uboot_defconfig
```

Tips: sd means to read from SD-card. The string "_uboot_" means to load and start u-boot as the next bootloader.

You can configure AT91Bootstrap to load U-Boot binary from SD Card 1 by doing:

```
$ make mrproper
$ make ourasi_lpddr2_ebsd1_uboot_defconfig
```

Then follow the steps highlighted here #Build AT91Bootstrap to finish building bootstrap.

Build AT91Bootstrap for SiP Boards

Get AT91Bootstrap Source Code

You can download AT91Bootstrap source code from the GitHub repository:
https://github.com/linux4sam/at91bootstrap/commits/at91bootstrap-4.x%2Bsama7d65-ea-1.0/

The source code is taken from the at91boostrap-4.x+sama7d65-ea branch which is dedicated to sama7d65 early development for **LPDDR** and **SIP** variant **only**.

To get the source code, you should clone the repository by doing:

```
$ git clone https://github.com/linux4sam/at91bootstrap.git -b \
at91bootstrap-4.x+sama7d65_ea at91bootstrap-4.x+sama7d65-ea-1.0
Cloning into 'at91bootstrap-4.x+sama7d65-ea-1.0'...
remote: Enumerating objects: 19145, done.
remote: Counting objects: 100% (4848/4848), done.
remote: Compressing objects: 100% (1314/1314), done.
remote: Total 19145 (delta 3699), reused 4598 (delta 3524), pack-reused 14297
Receiving objects: 100% (19145/19145), 5.84 MiB | 7.86 MiB/s, done.
Resolving deltas: 100% (14693/14693), done.
$
```

Then enter the resulting directory to be ready to use AT91Bootstrap source code:

```
$ cd at91bootstrap-4.x+sama7d65-ea-1.0/
```

Configure AT91Bootstrap

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

```
$ ls -1 configs/ourasi_sip*
configs/ourasi_sip_eb_bkptnone_defconfig
configs/ourasi_sip_ebsd0_uboot_defconfig
configs/ourasi_sip_ebsd1_uboot_defconfig
configs/ourasi_sip_ebsd2_uboot_defconfig
```

Tips: sd means to read from SD-card. The string "_uboot_" means to load and start u-boot as the next bootloader.

You can configure AT91Bootstrap to load U-Boot binary from SD Card 1 by doing:

```
$ make mrproper
$ make ourasi_sip_ebsd1_uboot_defconfig
```

Then follow the steps highlighted here #Build AT91Bootstrap to finish building bootstrap.

Build U-Boot from sources

Get U-Boot sources

You can download U-Boot source code from the GitHub repository:
https://github.com/linux4sam/u-boot-at91/tree/sama7d65_ea

clone the Linux4sam GitHub U-Boot repository. The source code is taken from the sama7d65_ea branch which is dedicated to sama7d65 early developments.

```
$ git clone https://github.com/linux4sam/u-boot-at91.git -b sama7d65_ea
Cloning into 'u-boot-at91'...
remote: Enumerating objects: 893387, done.
remote: Counting objects: 100% (129463/129463), done.
remote: Compressing objects: 100% (27884/27884), done.
remote: Total 893387 (delta 102594), reused 118427 (delta 100790), pack-reused 763924
Receiving objects: 100% (893387/893387), 197.17 MiB | 10.84 MiB/s, done.
Resolving deltas: 100% (739158/739158), done.
Updating files: 100% (17966/17966), done.
```

Then enter the resulting directory to be ready to use U-Boot source code:

```
$ cd u-boot-at91/
```

Cross-compile U-Boot

As mentioned in previous chapters, before compiling any component, you need to setup cross compile toolchain. Make sure that your CROSS_COMPILE environment variable is well defined:

```
$ echo $CROSS_COMPILE
arm-linux-gnueabihf-
$
```

If the resulting line after the “echo” command is empty, please make sure to follow the instructions in the Setup ARM Cross Compiler chapter above.

Once the U-Boot sources are available, cross-compiling U-Boot is made in two steps: configuration and compilation. Check the Configuration chapter in U-Boot reference manual.

Configure U-Boot

Assuming you are at the U-Boot root directory, you will find a configs folder which contains several default configuration files. As the U-Boot environment variables can be stored in different media, config files mentioned hereunder can be used to specify where to store it. To put environment variables in SD/MMC card use the default configuration file sama7d65_curiosity_mmc1_defconfig:

```
$ make sama7d65_curiosity_mmc1_defconfig
```

Customize U-Boot

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 add or remove some features to/from U-Boot as the same way as the kernel configuration. Move to “<Exit>” with arrows and press this button hitting the “Enter” key to exit from this screen.

Build U-Boot

Once the cross-compilation environment is setup and that U-Boot is properly configured, you can start the building process by running make:

```
$ 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 Linux kernel from sources

Get the Linux kernel sources

You can download Linux kernel source code from the GitHub repository:
https://github.com/linux4sam/linux-at91/tree/sama7d65_ea

The source code is taken from the sama7d65_ea branch which is dedicated to sama7d65 early developments.

To get the source code, you should clone the repository by doing:

```
$ git clone https://github.com/linux4sam/linux-at91.git -b sama7d65_ea
Cloning into 'linux-at91'...
remote: Enumerating objects: 9177564, done.
remote: Counting objects: 100% (1500/1500), done.
remote: Compressing objects: 100% (645/645), done.
remote: Total 9177564 (delta 1168), reused 1120 (delta 855), pack-reused 9176064
Receiving objects: 100% (9177564/9177564), 1.55 GiB | 10.46 MiB/s, done.
Resolving deltas: 100% (7729137/7729137), done.
Updating files: 100% (78834/78834), done.
```

Then enter the resulting directory to be ready to use Linux kernel source code:

```
$ cd linux-at91
```

Cross-compile the Linux kernel

As mentioned in previous chapters, before compiling any component, you need to setup cross compile toolchain. Make sure that your CROSS_COMPILE environment variable is well defined:

```
$ echo $CROSS_COMPILE
arm-linux-gnueabihf-
$
```

If the resulting line after the “echo” command is empty, please make sure to follow the instructions in the Setup ARM Cross Compiler chapter above.

Once the Linux kernel sources are available, cross-compiling Linux is made in two steps: configuration and compilation.

Configure the Linux kernel

Now you must configure the Linux kernel according to your hardware. We have three Microchip ScC 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. This is the one chosen for sama7d65 SoC.**

Now we Configure the kernel for sama7d65 boards:

```
$ make ARCH=arm sama7_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/menu.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

Customize the Linux kernel

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

```
$ make ARCH=arm menuconfig
```

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

Build the Linux kernel

Once the cross-compilation environment is setup and that the Linux kernel is properly configured, you can start the building process by running make:

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

You have a usable compressed kernel image **zImage** in the directory **arch/arm/boot/**.

Modify Device Tree

For adapting to changes on the hardware platform, you can change the Device Tree and re-generate a Device Tree Binary (.dtb) file.

If you need to recompile the device tree you can run this command:

```
$ make ARCH=arm dtbs
DTC     arch/arm/boot/dts/at91-sama7d65_curiosity.dtb
DTC     arch/arm/boot/dts/at91-sama7d65eb.dtb
```

How to build DT-Overlay for sama7d65

Get sources

To get the source code, you have to clone the DT Overlay repository from here: https://github.com/linux4sam/dt-overlay-at91/tree/sama7d65_ea

```
  $ git clone https://github.com/linux4sam/dt-overlay-at91.git -b sama7d65_ea
    Cloning into 'dt-overlay-at91'...
    remote: Enumerating objects: 1687, done.
    remote: Counting objects: 100% (281/281), done.
    remote: Compressing objects: 100% (68/68), done.
    remote: Total 1687 (delta 235), reused 245 (delta 213), pack-reused 1406
    Receiving objects: 100% (1687/1687), 543.12 KiB | 2.19 MiB/s, done.
    Resolving deltas: 100% (1003/1003), done.
  $ cd dt-overlay-at91/
```

Build the DT-Overlay

To build the overlays for a board make sure the following steps are done:

- the environment variables ARCH and CROSS_COMPILE are set correctly
- (optional) the environment variable KERNEL_DIR points to Linux kernel and the kernel was already built for the board. This is needed because the DT Overlay repository uses the Device Tree Compiler (dtc) from the kernel source tree. By default, KERNEL_DIR is set to a linux directory that would be under the parent directory in the directory tree: ../linux
- (optional) the environment variable KERNEL_BUILD_DIR that points to where the Linux kernel binary and Device Tree blob, resulting of your compilation of the kernel, are located. By default, KERNEL_BUILD_DIR is set to the same directory as KERNEL_DIR. It shouldn't be changed if you have the habit of compiling your kernel within the Linux source tree

Start the build by running make

```
$ make sama7d65_curiosity.itb
```

Loading DT-Overlay in U-Boot

The FIT image is a placeholder that has the zImage and the base Device Tree, plus additional overlays that can be selected at boot time.

The following steps are required to boot the FIT Image from U-boot:

- Load the FIT image like you would normally load the uImage or zImage
- There is no need to load additional Device Tree Blob, the FIT image includes it
- When booting the FIT image, specify the FIT configuration to use. Several configurations can be appended to the basic configuration, which we name 'kernel_dtb'

Example:

```
fatload mmc 0:1 0x63000000 sama7d65_curiosity.itb; bootm 0x63000000#kernel_dtb
```

This will load the FIT image from address 0x63000000 in memory and then run the configuration named 'kernel_dtb'. This configuration includes the kernel plus the base Device Tree Blob built with the kernel.

Example to load MIPI display Device Tree Overlay:

```
fatload mmc 0:1 0x63000000 sama7d65_curiosity.itb; bootm 0x63000000#kernel_dtb#mipi
```

How to build Buildroot for sama7d65

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 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 changes which will not hit the mainline: additional defconfigs files and packages dedicated to our demos.

```
$ git clone https://github.com/linux4sam/buildroot-at91.git -b 2024.02-mchp
Cloning into 'buildroot-at91'...
remote: Enumerating objects: 463728, done.
remote: Counting objects: 100% (83926/83926), done.
remote: Compressing objects: 100% (30247/30247), done.
remote: Total 463728 (delta 53436), reused 83881 (delta 53426), pack-reused 379802
Receiving objects: 100% (463728/463728), 102.44 MiB | 10.50 MiB/s, done.
Resolving deltas: 100% (313672/313672), done.

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

$ git clone https://github.com/linux4sam/buildroot-external-microchip.git -b sama7d65_ea
Cloning into 'buildroot-external-microchip'...
remote: Enumerating objects: 7753, done.
remote: Counting objects: 100% (7753/7753), done.
remote: Compressing objects: 100% (1634/1634), done.
remote: Total 7753 (delta 5882), reused 7715 (delta 5844), pack-reused 0
Receiving objects: 100% (7753/7753), 1.07 MiB | 5.91 MiB/s, done.
Resolving deltas: 100% (5882/5882), done.

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

$ ls -1 buildroot-external-microchip/configs/sama7d65*
configs/sama7d65_curiosity_graphics_defconfig
configs/sama7d65_curiosity_headless_defconfig
```

Build the rootfs image

To build the the rootfs image that we provide for sama7d65 curiosity, you will have to do:

```
$ cd buildroot-at91
$ BR2_EXTERNAL=../buildroot-external-microchip/ make sama7d65_curiosity_graphics_defconfig
$ make
```

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

```
$ ls -1 output/images/

at91bootstrap.bin
at91-sama7d65_curiosity.dtb
boot.bin
boot-plaintextimg.bin
boot.vfat
rootfs.ext2
rootfs.ext4
rootfs.tar
sama7d65_curiosity_at25ff321a_click1.dtbo
sama7d65_curiosity.itb
sama7d65_curiosity.its
sama7d65_curiosity_lvds.dtbo
sama7d65_curiosity_mipi.dtbo
sama7d65_curiosity_pwm.dtbo
sama7d65_curiosity_tcb_pwm.dtbo
sama7d65-sdcardboot-uboot-4.0.7.bin
sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea.bin
sama7d65-sdcardboot-uboot-4.0.7-sama7d65-ea-plaintextimg.bin
sdcard.img
u-boot.bin
uboot-env.bin
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 sama7d65

Prerequisites

Note that building an entire distribution is a long 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 are hosted on github: https://github.com/linux4sam/meta-atmel

A step-by-step comprehensive installation is explained in this link: https://docs.yoctoproject.org/current/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.

Here are the reference pages for setting up a Yocto building environment: https://docs.yoctoproject.org/current/brief-yoctoprojectqs/index.html#build-host-packages

For instance, on Debian based systems these packages need to be installed on your development host:

```
sudo apt-get install gawk wget git-core git-lfs diffstat unzip texinfo gcc-multilib \
     build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
     xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa\
     libsdl1.2-dev\ pylint3 xterm
```

Build Procedure

```
Sources
=======
- meta-atmel
URI: https://github.com/linux4sam/meta-atmel.git
Branch: sama7d65_ea
Tag: sama7d65-ea-1.0

Dependencies
============
This Layer depends on :
- poky
URI: https://git.yoctoproject.org/poky
Branch: kirkstone
Tag:yocto-4.0.17

- meta-openembedded
URI: https://git.openembedded.org/meta-openembedded
Branch: kirkstone
Tag/commit:8bb16533532b6abc2eded7d9961ab2a108fd7a5b
```

```
- meta-arm (for optee components)
URI: https://git.yoctoproject.org/meta-arm
Branch: kirkstone
Tag:yocto-4.0.2

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 https://git.yoctoproject.org/poky && cd poky && \
git checkout -b kirkstone yocto-4.0.17 && cd -

2/ Clone meta-openembedded git repository with the proper branch ready
git clone git://git.openembedded.org/meta-openembedded && \
cd meta-openembedded && git checkout -b kirkstone 8bb165 && cd -

3/ Clone meta-atmel layer with the proper branch ready
git clone https://github.com/linux4sam/meta-atmel.git && \
cd meta-atmel && git checkout -b sama7d65_ea sama7d65-ea-1.0 && cd -

4/ Clone meta-arm layer with the proper branch ready
git clone https://git.yoctoproject.org/meta-arm && cd meta-arm && \
git checkout -b kirkstone yocto-4.0.2 && cd -

5/ Enter the poky directory to configure the build system and start the build process
cd poky
If not created yet, add a new "build-microchip" directory:
mkdir build-microchip
Else, if it's the first time you use Yocto Project templates, and if the
build-microchip directory remains from a previous use, we advise you to start
from a fresh directory. Keep your build-microchip/conf/local.conf file for
reference.

6/ Inside the .templateconf file, you will need to modify the TEMPLATECONF
variable to match the path to the meta-atmel layer "conf" directory:
export TEMPLATECONF=${TEMPLATECONF:-../meta-atmel/conf}

7/ Initialize build directory
source oe-init-build-env build-microchip

8/ To build a small image provided by Yocto Project:
[MACHINE=] bitbake core-image-minimal

Example for sama7d65-curiosity-sd SD card image:
MACHINE=sama7d65-curiosity-sd bitbake core-image-minimal

9/ To build the microchip image with no graphics support:
[MACHINE=] bitbake microchip-headless-image

Example for sama7d65-curiosity-sd SD card image:
MACHINE=sama7d65-curiosity-sd bitbake microchip-headless-image
```

```
10/ To build the microchip image with graphics support (EGT):
[MACHINE=] bitbake microchip-graphics-image

Example for sama7d65-curiosity-sd SD card image:
MACHINE=sama7d65-curiosity-sd bitbake microchip-graphics-image
```

Once done building, the image will be in ./tmp/deploy/images/sama7d65-curioisty-*/*.wic

Yocto Tips & Tricks

Bitbake:

- Yocto Reference manual: https://docs.yoctoproject.org/ref-manual/#_blank
- Yocto Reference manual Bitbake sub-section: https://docs.yoctoproject.org/bitbake/

  Listing tasks provided by each package:

  ```
  bitbake -c listtasks <package_name>
  ```

- Bitbake task cheat sheet: https://elinux.org/Bitbake_Cheat_Sheet

Feature list

1. AT91Bootstrap
    a. Basic support
        i. PMC
        ii. Serial
        iii. PIO
        iv. PIT64B
    b. Main types of DDR / LPDDR supported
        i. Support for each SiP which RAM size is compatible with a Linux-based system
    c. TZC support (simple, static, no dual world, all access allowed)
    d. SD-Card support
    e. CPU at 800 MHz clock (with DVFS support)
    f. Support for Backup-and-Self-Refresh (BSR)
2. U-Boot
    a. Basic support
        i. PMC
        ii. Serial
        iii. PIO
        iv. PIT64B
    b. Ethernet GMAC support + TFTP functional
    c. SD-Card
    d. USB gadget support
3. DT-Overlay-AT91
    a. Basic support
    b. Overlay for MIPI displays
    c. Overlay for LVDS displays
    d. Overlay for PWM Mikro Bus connection
4. Linux Kernel
    a. Basic support
        i. PMC
        ii. Serial
        iii. PIO
        iv. PIT64B
        v. DMA
        vi. Button and LEDs

   b. Power Management
      i. ULP0, ULP1, BSR modes
      ii. DVFS support with maximum CPU clock 1GHz
   c. PWM / TC
   d. ADC + thermal functions
   e. Crypto engines: TDES, SHA, AES
   f. SD-Card (basic modes)
      i. SDIO mode
   g. GMAC x 2
   h. I2C / SPI / USART (FLEXCOM)
   i. Display sub-system
      i. LCD controller
      ii. LCD backlight
      iii. LVDS controller
      iv. MIPI-DSI controller
      v. Microchip LVDS screen landscape mode
      vi. Microchip MIPI screen portrait mode
      vii. MaXTouch touchscreen controller
   j. USB Host on ports B and C (type-A connector)
   k. USB Gadget on ports B and C (type-A connector)
5. Buildroot
   a. Graphics / Headless
      i. SD-Card Image
   b. Ensemble Graphics Toolkit (EGT) supported
6. Yocto Project
   a. Graphics / Headless
      i. SD-Card Image
   b. Ensemble Graphics Toolkit (EGT) supported

Known Limitations

1. Backup-with-Self Refresh mode (BSR) not supported in buildroot ready-made images version sama7d65-ea-1.0

The buildroot ready made images named linux4sam-buildroot-sama7d65_curiosity-graphics-sama7d65-ea-1.0.img.bz2 and linux4sam-buildroot-sama7d65_curiosity-headless-sama7d65-ea-1.0.img.bz2 don't support BSR mode as this option is not selected in the at91bootstrap booloader binary.

Work around:

Recompile at91boostrap choosing the defconfig file sama7d65_curiosity-bsrsd1_uboot_defconfig and following the procedure described above in this document

or

Use the ready-made Yocto Project images named linux4sam-poky-sama7d65_curiosity-graphics-sama7d65-ea-1.0.img.bz2 and linux4sam-poky-sama7d65_curiosity-headless-sama7d65-ea-1.0.img.bz2 which contain the support for BSR pre-compiled.