TySOM-M-MPFS250T
Linux Guide

Revision 1.1
May 13, 2022

Aldec® Disclaimer: The information provided in this document is provided in connection with Aldec's Hardware products. All solutions, designs, schematics, drawings, boards or other information provided by Aldec to Customer are intellectual property of Aldec, Inc. No license, express or implied, by estoppel, or otherwise, to any intellectual property right is granted to Customer by this document or in connection with the sale of Aldec products.

Export Restriction
Aldec, Inc. Hardware is not to be exported or re-exported, including reference images or accompanying documentation in any form without the appropriate government licenses, if required, and the expressed consent of Aldec, Inc. Purchaser warrants that it is not prohibited from receiving the Hardware under U.S. export laws and that it is not a national of a country subject to U.S. trade sanctions. Purchaser will not use the Hardware in a location that is the subject of U.S. trade sanctions that would cover the Hardware. Purchaser warrants that it is not subject to the U.S. Department of Commerce's table of deny orders or otherwise prohibited from obtaining goods of this sort from the United States.
Resale Restriction
Aldec, Inc. Hardware is not to be resold except within their assigned territory by distributors with a valid written distribution agreement with Aldec, Inc. Purchasers of the Hardware agree not to transfer the Hardware to any third party without express written consent of Aldec, Inc.
2/25/2010

Table of Contents

Table of Figures

1 Introduction

The following document details the process of booting Linux OS on the TySOM-M-MPFS250T rev2 board. The project, along with all the necessary source files, is available on https://github.com/aldec. Additional information on the board's setup and connections is available in the TySOM-M-MPFS250T rev2 Quick Start Guide document. The project's structure is available in Appendix A.

Described steps:

- building Linux OS image using Yocto Project
- generating HSS (Hart Software Services)
- HW generation with HSS loaded in
- uploading a drive image to the eMMC memory
- booting the OS from the memory

Used abbreviations:

- TOP_DIR – location of the main package directory (named 'TySOM-M-MPFS250T')

2 Building Linux OS using Yocto Project

The Linux OS drive image is generated using the Yocto Project.

1. Move into a directory where you keep utilities and make sure it is in the PATH environment variable:

```
cd ~/bin/
```

2. Download the repo script and make it executable:

```
curl https://storage.googleapis.com/git-repo-downloads/repo > repo
chmod a+x repo
```

3. Test if the script is working:

```
repo --help
```

4. Download the PolarFire SoC repository:

```
mkdir yocto-dev
cd yocto-dev
repo init -u https://github.com/polarfire-soc/meta-polarfire-soc-yocto-bsp.git
-b master -m tools/manifests/riscv-yocto.xml
cp <TOP_DIR>/BSP/yocto/2021.08/riscv-yocto-v2021.08.xml .repo/manifests
repo init -u https://github.com/polarfire-soc/meta-polarfire-soc-yocto-bsp.git
-b master -m riscv-yocto-v2021.08.xml
repo sync
repo rebase
```

5. (Optional) For older OS, prepare build tools in proper versions (tar 3.x and gcc 10.x):

```
cd ./openembedded-core/scripts
./install-buildtools --with-extended-buildtools --url http://downloads.yoctoproject.org/releases/yocto/yocto-3.0.2/
```

6. Setting the environment for Yocto:

```
cd ~/bin/yocto-dev
(optional if step 5 was taken) source ../openembedded-core/buildtools/environment-setup-x86_64-pokysdk-linux
source ./meta-polarfire-soc-yocto-bsp/polarfire-soc_yocto_setup.sh
```

7. Adding the TySOM-M BSP layer to Yocto configuration:

```
cd ./build
cp -r <TOP_DIR>/BSP/yocto/2021.08/meta_TySOM-M-MPFS250T_yocto_bsp ~/bin/yocto-dev
bitbake-layers add-layer ../meta_TySOM-M-MPFS250T_yocto_bsp
```

8. Generate the files:

```
bitbake files
MACHINE=tysom-m-mpfs250t bitbake mpfs-dev-cli
```

Once the process is finished, a mpfs-dev-cli-tysom-m-mpfs250t.wic.gz Linux image file should be available in the build directory.

3 Generating HSS

HSS needs to be generated before building the hardware design because it will be embedded in the bitstream.

1. Go to the project's hss directory and run the hss.sh script – it will download the HSS repository, checkout a necessary commit and apply the Aldec HSS patch:

```
cd <TOP_DIR>/BSP/hss/v1_0
./hss.sh
```

2. Go to the newly created hart-software-services directory and copy the board's configuration file:

```
cd hart-software-services
cp boards/tysom-m-mpfs250t/def_config ./.config
```

3. Build the HSS:

```
make BOARD=tysom-m-mpfs250t
```

Once generation is finished the hss.hex and hss.elf files can be found in the Default directory.

4 HW generation and programming the board

In order to load a Linux OS image into the eMMC drive, FPGA hardware needs to be generated with

HSS instead of a bare-metal application embedded in it. It allows to mount the drive as USB storage on a host PC and copy the necessary files.

1. Navigate to <TOP_DIR>/BSP/designs/libero2021.1/tysom_m_mpfs250t_ref_design
2. Run Libero and select Project → Execute Script…

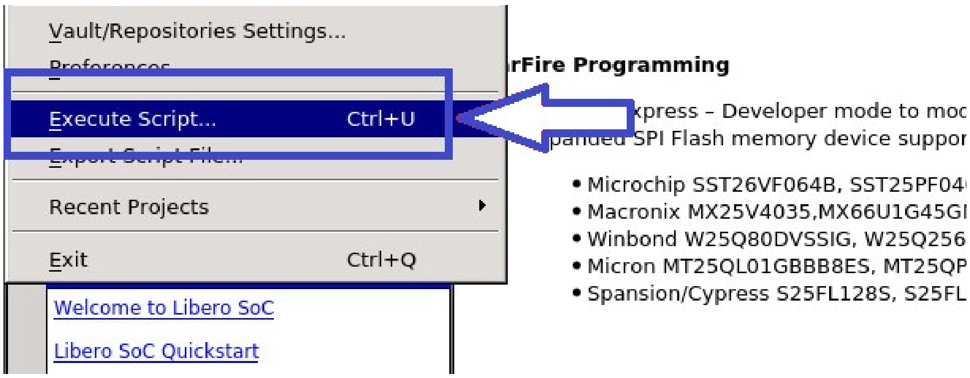

Figure 1: Libero 'Execute Script…' command location

3. Choose either TySOM-M-MPFS250T_SD.tcl or TySOM-M-MPFS250T_eMMC.tcl script and run it. The former will generate a design which uses the SD card slot, while the latter will utilize internal eMMC memory. This generates a block design suitable for the TySOM-M-MPFS250T board. Once the process finishes, an appropriate message will be displayed in the Report window.
4. In the Design Flow tab double click the 'Generate FPGA Array Data' option and wait for it to finish.
5. Double click the 'Configure Design Initialization Data and Memories' option, and select the eNVM tab. Press 'Add→Add Boot Mode 1 Client' and select the hss.hex file build in Chapter 3, in the hss directory.
6. Generate the bitstream.
7. To program the board, double-click the 'Run PROGRAM Action' option in the 'Design Flow' part of Libero. The board is programmed through a Flash PRO programmer, verify if one is connected.

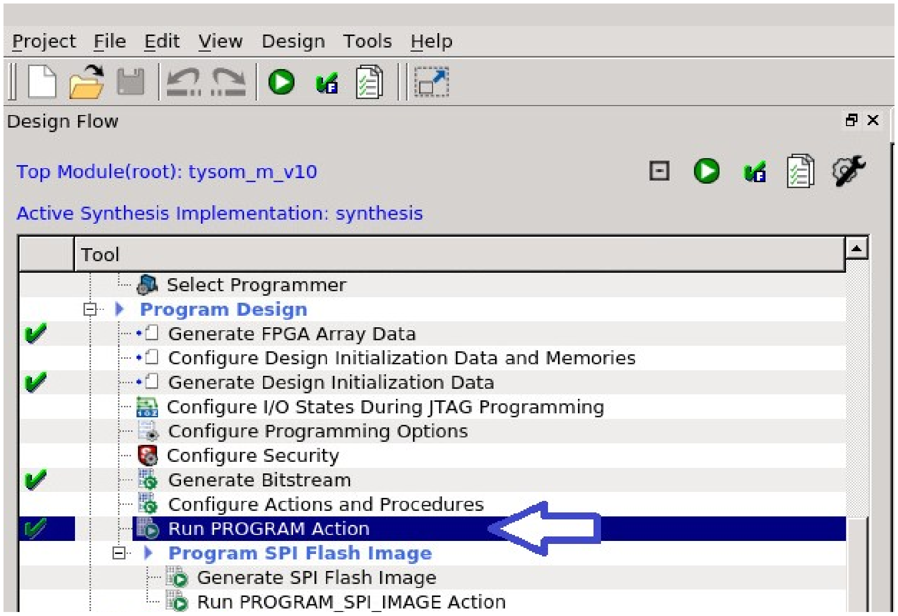

Figure 2: Programming the board

5 Running Linux OS on the board

Linux OS can be loaded either from an SD card, or internal eMMC memory – dependent on which hardware design variant was generated.

5.1 Preparing an SD card

1. Insert the SD card into an appropriate card reader and connect it to the workstation
2. Check the SD card's label with the dmesg command and use it instead of sdX in the following command:

```
zcat ./tmp-glibc/deploy/images/tysom-m-mpfs250t/mpfs-dev-cli-tysom-m-mpfs250t.wic.gz | sudo dd of=/dev/sdX bs=4096 iflag=fullblock oflag=direct conv=fsync status=progress
```

3. Once the transfer is completed plug the SD card into the TySOM-M-MPFS250T rev2 board, connect the UART cable and power the board on. Linux console should be available under UART1, and the HSS console under UART0.

5.2 Programming the internal eMMC memory

Before Linux can be booted, it's image must be copied into the eMMC memory on the board. The HSS software embedded in the bitstream exposes it as a USB mass storage device which can be mounted on the host PC.

1. Connect to the HSS console using any terminal emulation program. For this guide, picocom has been used:

```
picocom -b 115200 /dev/ttyXRUSB0
```

2. Press any key to start the boot process.
3. Once HSS selects the eMMC memory and starts a 5 second timeout countdown press any key to enter the command line interface.

```
[3.20612] HSS_MMCInit(): Attempting to select eMMC ... Passed
Press a key to enter CLI, ESC to skip
Timeout in 5 seconds
.....[7.684133] HSS_ShowTimeout(): Character 13 pressed

[7.690556] HSS_TinyCLI_Parser(): Type HELP for list of commands
```

Figure 3: HSS boot interrupted

4. Run the ‘usbdmsc’ command in the HSS console.

```
[7.697848] HSS_TinyCLI_Parser(): >> usbdmsc
[49.445792] HSS_MMCInit(): Attempting to select SDCARD ... [50.456378]
Failed
[50.465058] HSS_MMCInit(): Attempting to select eMMC ... Passed
Waiting for USB Host to connect... (CTRL-C to quit)
USB Host connected. Waiting for disconnect... (CTRL-C to quit)
 0 bytes written, 2582528 bytes read
```

Figure 4: Running the ‘usbdmsc’ command

5. On the host PC use ‘dmesg | tail’ command to verify if a mass storage device was mounted:

```
[root@est1 ZT]# dmesg | tail
[787329.016190] GPT:Alternate GPT header not at the end of the disk.
[787329.016191] GPT:11253589 != 15273599
[787329.016192] GPT: Use GNU Parted to correct GPT errors.
[787329.016199]  sdb: sdb1 sdb2 sdb3
[787346.816032] GPT:Primary header thinks Alt. header is not at the end of the disk.
[787346.816034] GPT:11253589 != 15273599
[787346.816035] GPT:Alternate GPT header not at the end of the disk.
[787346.816036] GPT:11253589 != 15273599
[787346.816037] GPT: Use GNU Parted to correct GPT errors.
[787346.816045]  sdb: sdb1 sdb2 sdb3
```

Figure 5: Running the ‘dmesg’ command in the host terminal

6. Run the following command to copy the image to the mounted drive, using the appropriate drive label instead of sdX (the image is approximately 5GB in size so it may take up to 30 minutes to load, dependent on the transfer speed):

   ```
   zcat mpfs-dev-cli-tysom-m-mpfs250t.wic.gz | dd of=/dev/sdX bs=4096 iflag=fullblock oflag=direct conv=fsync status=progress
   ```

```
[root@est1 ZT]# zcat mpfs-dev-cli-tysom-m-mpfs250t.wic.gz | dd of=/dev/sdb bs=40
5760286720 bytes (5.8 GB) copied, 1463.303744 s, 3.9 MB/s
1406624+1 records in
1406624+1 records out
5761533952 bytes (5.8 GB) copied, 1463.67 s, 3.9 MB/s
```

Figure 6: Running the 'zcat' command

7. Once the 'zcat' command finishes copying the OS image to the eMMC memory on the board, return to the HSS terminal opened previously using picocom. It should display a number of bytes written to the mounted drive.
8. Press Ctrl+C in order to interrupt the mass storage device mount command.

```
[7.886946] HSS_MMCInit(): Attempting to select eMMC ... Passed
Waiting for USB Host to connect... (CTRL-C to quit)
USB Host connected. Waiting for disconnect... (CTRL-C to quit)
 5761552384 bytes written, 5242880 bytes read
USB Host disconnected...
```

Figure 7: Interrupting the usbdmsc command

9. In order to verify which handle is used for the Linux OS terminal, connect to every other /dev/ttyXRUSBX using picocom. Once the next step finishes, each will display a 'hartX setup completed' message – the Linux OS terminal will be available on the 'hart1' one.

10. Run ‘boot’ command.

```
[321.805695] HSS_TinyCLI_Parser(): >> boot

[345.370419] IPI_QueuesInit(): Initializing IPI Queues (9000 bytes @ 8000e40)...
[345.379186] HSS_PMP_Init(): Initializing PMPs
[345.385002] HSS_BootInit(): Initializing Boot Image..
[345.391512] getBootImageFromMMC_(): Preparing to copy from MMC to DDR ...
[345.399758] getBootImageFromMMC_(): Attempting to read image header (1552 bytes) ...
[345.411693] GPT_ValidateHeader(): Validated GPT Header ...
[345.463911] GPT_ValidatePartitionEntries(): Validated GPT Partition Entries ...
[345.474397] copyBootImageToDDR_(): Copying 470896 bytes to 0xB0000000
[349.470686] copyBootImageToDDR_(): Calculated CRC32 of image in DDR is 3a12a1d7
[349.567026] HSS_BootInit():  boot image passed CRC
[349.573275] HSS_BootInit(): Boot image set name: "PolarFire-SoC-HSS::U-Boot"
[349.581782] HSS_BootInit(): Boot Image registered...
[349.588205] HSS_Boot_RestartCore(): called for all harts
[349.594976] RunStateMachine(): boot_service(u54_1)::Init -> boot_service(u54_1)::SetupPMP
[349.604870] RunStateMachine(): boot_service(u54_2)::Init -> boot_service(u54_2)::SetupPMP
[349.614766] RunStateMachine(): boot_service(u54_3)::Init -> boot_service(u54_3)::SetupPMP
[349.624661] RunStateMachine(): boot_service(u54_4)::Init -> boot_service(u54_4)::SetupPMP
[349.634556] RunStateMachine(): usbdmsc_service::init -> usbdmsc_service::idle
[349.643410] RunStateMachine(): boot_service(u54_1)::SetupPMP -> boot_service(u54_1)::SetupPMPComplete
[349.654346] RunStateMachine(): boot_service(u54_2)::SetupPMP -> boot_service(u54_2)::SetupPMPComplete
[349.665283] RunStateMachine(): boot_service(u54_3)::SetupPMP -> boot_service(u54_3)::SetupPMPComplete
[349.676220] RunStateMachine(): boot_service(u54_4)::SetupPMP -> boot_service(u54_4)::SetupPMPComplete
[349.687157] RunStateMachine(): boot_service(u54_1)::SetupPMPComplete -> boot_service(u54_1)::ZeroInit
[349.698094] RunStateMachine(): boot_service(u54_2)::SetupPMPComplete -> boot_service(u54_2)::ZeroInit
[349.709030] RunStateMachine(): boot_service(u54_3)::SetupPMPComplete -> boot_service(u54_3)::ZeroInit
[349.719967] RunStateMachine(): boot_service(u54_4)::SetupPMPComplete -> boot_service(u54_4)::ZeroInit
[349.730904] RunStateMachine(): boot_service(u54_1)::ZeroInit -> boot_service(u54_1)::Download
[349.741146] RunStateMachine(): boot_service(u54_2)::ZeroInit -> boot_service(u54_2)::Download
[349.751389] RunStateMachine(): boot_service(u54_3)::ZeroInit -> boot_service(u54_3)::Download
[349.761631] RunStateMachine(): boot_service(u54_4)::ZeroInit -> boot_service(u54_4)::Download
[349.771874] RunStateMachine(): boot_service(u54_2)::Download -> boot_service(u54_2)::Idle
[349.781769] RunStateMachine(): boot_service(u54_3)::Download -> boot_service(u54_3)::Idle
[349.791664] RunStateMachine(): boot_service(u54_4)::Download -> boot_service(u54_4)::Idle
[349.827460] RunStateMachine(): boot_service(u54_1)::Download -> boot_service(u54_1)::Wait
[349.837356] boot_download_chunks_onExit(): boot_service(u54_1)::u54_2:sbi_init 80200000
[349.846817] boot_download_chunks_onExit(): boot_service(u54_1)::u54_3:sbi_init 80200000
[349.856279] boot_download_chunks_onExit(): boot_service(u54_1)::u54_4:sbi_init 80200000
[349.865740] boot_download_chunks_onExit(): boot_service(u54_1)::u54_1:sbi_init 80200000
[349.875201] boot_wait_onEntry(): boot_service(u54_1)::Checking for IPI ACKs: - -
[349.884054] boot_wait_handler(): boot_service(u54_1)::Checking for IPI ACKs: ACK/IDLE ACK
[349.893689] RunStateMachine(): boot_service(u54_1)::Wait -> boot_service(u54_1)::Idle
```

Figure 8: HSS console output of the ‘boot’ command

11. It may be necessary to power-cycle the board for the Linux OS to boot properly. Use ‘root’ login and no password:

```
[  OK  ] Started Network Router Discovery Daemon.
         Starting Permit User Sessions...
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started Avahi mDNS/DNS-SD Stack.
[  OK  ] Started Getty on tty1.
[  OK  ] Started Serial Getty on ttyS0.
[  OK  ] Reached target Login Prompts.
[  OK  ] Reached target Multi-User System.
         Starting Update UTMP about System Runlevel Changes...
[  OK  ] Finished Update UTMP about System Runlevel Changes.

OpenEmbedded nodistro.0 tysom-m-mpfs250t ttyS0

tysom-m-mpfs250t login: root
root@tysom-m-mpfs250t:~#
```

Figure 9: Successfully booted Libero Linux OS

Appendix A:

Project directory hierarchy:

- TySOM-M-MPFS250T → Main directory
  - BSP → Board Support Package
    - designs → Libero design for the board
    - doc → Documentation
    - hss → Hart Software Services generation
    - mss → Polarfire SOC MSS Configurator
    - yocto → Yocto Project for building Linux OS

In case of a host connection issue on Linux OS:

The Aldec TySOM-M-MPFS250T rev2 board requires an XR21V1414 USB UART driver (*https://www.maxlinear.com/product/interface/uarts/usb-uarts/xr21v1414*)

Check if the CDC-ACM driver was installed for the Exar USB UART (used by the first revision of the board):

```
ls /dev/tty*
```

If the listed handles contain 'ACM' in their names, the CDC-ACM driver needs to be replaced:

```
rmmod cdc-acm
```

Install the previously downloaded XR21V driver:

```
insmod xr_usb_serial_common.ko
```

Plug the device into a USB port. You should see up to four devices appear, typically /dev/ttyXRUSB[0-3].

In case of a host connection issue on Windows OS:

The Aldec TySOM-M-MPFS250T rev2 board requires an XR21V1414 USB UART driver (*https://www.maxlinear.com/product/interface/uarts/usb-uarts/xr21v1414*)

Check windows detection USB driver USB UART is as below (no driver exist):

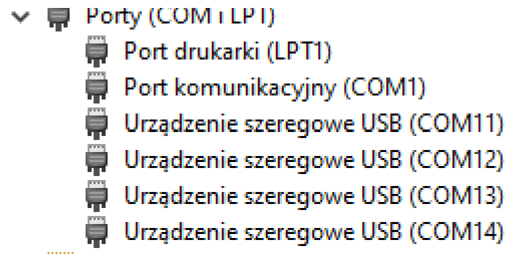

Figure 10: Detection USB UART under Windows 10 without driver installation.

After driver installation detection is as below:

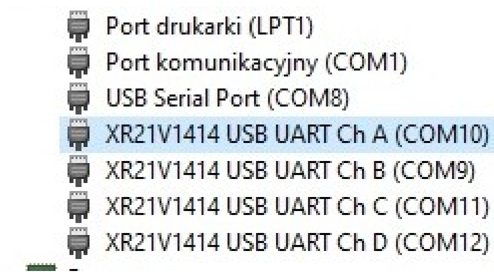

Figure 11: Detection USB UART under Windows 10 after driver installation.

The USB UART console is proposed as follow :

- UART Ch A : is dedicated to HSS console
- UART Ch B : is dedicated to Linux console
- UART Ch C : is dedicated to OpenSBI console
- UART Ch D : is dedicated to RISC-V console

The positions of system detection is randomize and make sure comX is connected to UART Ch A. The boot of HSS wait for any key and enter to accept boot. As you see Figure 11 is not first terminal calculate by windows , important **is chanel A to accept key to boot HSS**.

About Aldec, Inc.

Established in 1984, Aldec Inc. is an industry leader in Electronic Design Verification and offers a patented technology suite including: RTL Design, RTL Simulators, Hardware-Assisted Verification, Design Rule Checking, IP Cores, DO-254 Functional Verification and Military/Aerospace solutions. Continuous innovation, superior product quality and total commitment to customer service comprise the foundation of Aldec's corporate mission. For more information, visit *www.aldec.com*.

Contact:
2260 Corporate Circle
Henderson, NV 89074
USA
Tel: (702) 990-4400
Fax: (702) 990-4414
E-mail: *sales@aldec.com*
Website: *www.aldec.com*