AN13369

i.MX 8 - Boot Time Measurements Methodology

Rev. 1 — 29 April 2022 Application Note

Contents

1 Introduction

This document presents a possible approach to measure the boot time on the i.MX 8 platforms using the GPIO pins.

The main objectives of this document are as follows:

- Modifying the bootloader and system image for measuring
- Setting up the board and the external logic analyzer tool
- Achieving short boot times

1.1 Software environment

Linux BSP release 5.4.70_2.3.0 is used to perform the measurements of i.MX 8MQ, i.MX 8MP, i.MX 8MM, and i.MX 8MN. Linux BSP release 5.10.72_2.2.0 is used to perform the measurement of i.MX 8ULP.

1.2 Hardware setup and equipment

- Development kits:
  - NXP i.MX 8MQ EVK LPDDR4
  - NXP i.MX 8MP EVK LPDDR4
  - NXP i.MX 8MM EVK LPDDR4
  - NXP i.MX 8MN EVK LPDDR4
  - NXP i.MX 8ULP EVK LPDDR4
- Micro SD card. SanDisk Ultra 32 GB Micro SDHC I Class 10 is used for the current experiment.
- Micro-USB cable for the debug port.
- USB Type-C cable for data transfer.
- Logic analyzer with the following minimum requirements to measure the times: 4 channels and 10 MS/s. Saleae Logic 8 or Agilent 16902A is used for the current experiment.
- Breadboard and FPC to 60x0.5 Extension Board (only for i.MX8 MQ).

2 General description

This section describes the general procedure that must be performed to obtain a baseline measurement for a clean system (with no startup optimizations).

2.1 Choosing the GPIO pin for measurement

Use a general-purpose pin to generate a pulse signal at different booting phases. The desired GPIO pin is ideally chosen from those that are not used in either the bootloader or Linux; this can be checked in the associated device tree.

If this is not possible, disable the peripheral module that uses the pin in the next step. It is highly recommended to choose a pin from the expansion connector on the board.

2.2 Updating the device tree at the bootloader and Linux level

After you choose the desired pin, make some modifications at the device-tree level.

Firstly, the functionalities of the desired pin are checked to find out the macros associated with the GPIO functionality. The header files are at different location, depending on the board used.

Secondly, if a pin is used by some other peripheral modules, disable the respective module. This is done by setting the status property to “disabled” in the configuration info for that peripheral module.

Thirdly, the adequate pin muxing is defined in the `pinctrl_hog` section, using the GPIO macro for the chosen pin and the pad-configuration values (`IOMUXC_SW_PAD_CTL_PAD_*`).

2.3 Adding the first pulse generator

The first period measured is between the board POR and the execution of the `board_init_f` function of the SPL part of the bootloader. To generate a pulse, configure the pin as the output. After this, you can drive the pin high for a short time and then you can drive it low. You can do this without delay, because only the rising edge of the pulse is necessary.

2.4 Adding the second pulse generator

The second period measured is between the execution of the `board_init_f` function of the SPL part of the bootloader and before loading the kernel image from the U-Boot console. You can toggle that here using the U-Boot GPIO commands, which can be written in the board configuration file, located in the `/include/configs` folder.

2.5 Adding the third pulse generator

The third period measured is between loading the kernel image from the U-Boot console and starting the `psplash` program, which uses the frame buffer to show the relevant content on the display. To change the GPIO output state, add the `libgpiod` package to the `psplash` Yocto recipe. After adding the `libgpiod` package, the `gpiod` functions are added to the `psplash` code to generate a pulse right before the program uses the frame buffer for the first time.

2.6 Measuring the total time with the logic analyzer

The measurement stage can begin after building and flashing both the bootloader and Linux image to the board.

The boot-time is measured by starting the recording mode on the logic analyzer software and applying the reset button on the board. The recording stops after the third rising edge on the chosen GPIO pin. The elapsed boot time is the time between the rising edge of the nRST signal and the third rising edge of the GPIO pin.

3 Examples

3.1 i.MX 8M Quad

Choosing the pins

The chosen pin is the 19th pin on the J1801 expansion connector on the specified board. In the schematics for the baseboard, the pin is used for the SAI 1 peripheral with the SAI1_RXFS function. Searching the specified pad in the reference manual for the associated pin returns an alternate function of GPIO4_IO[0].

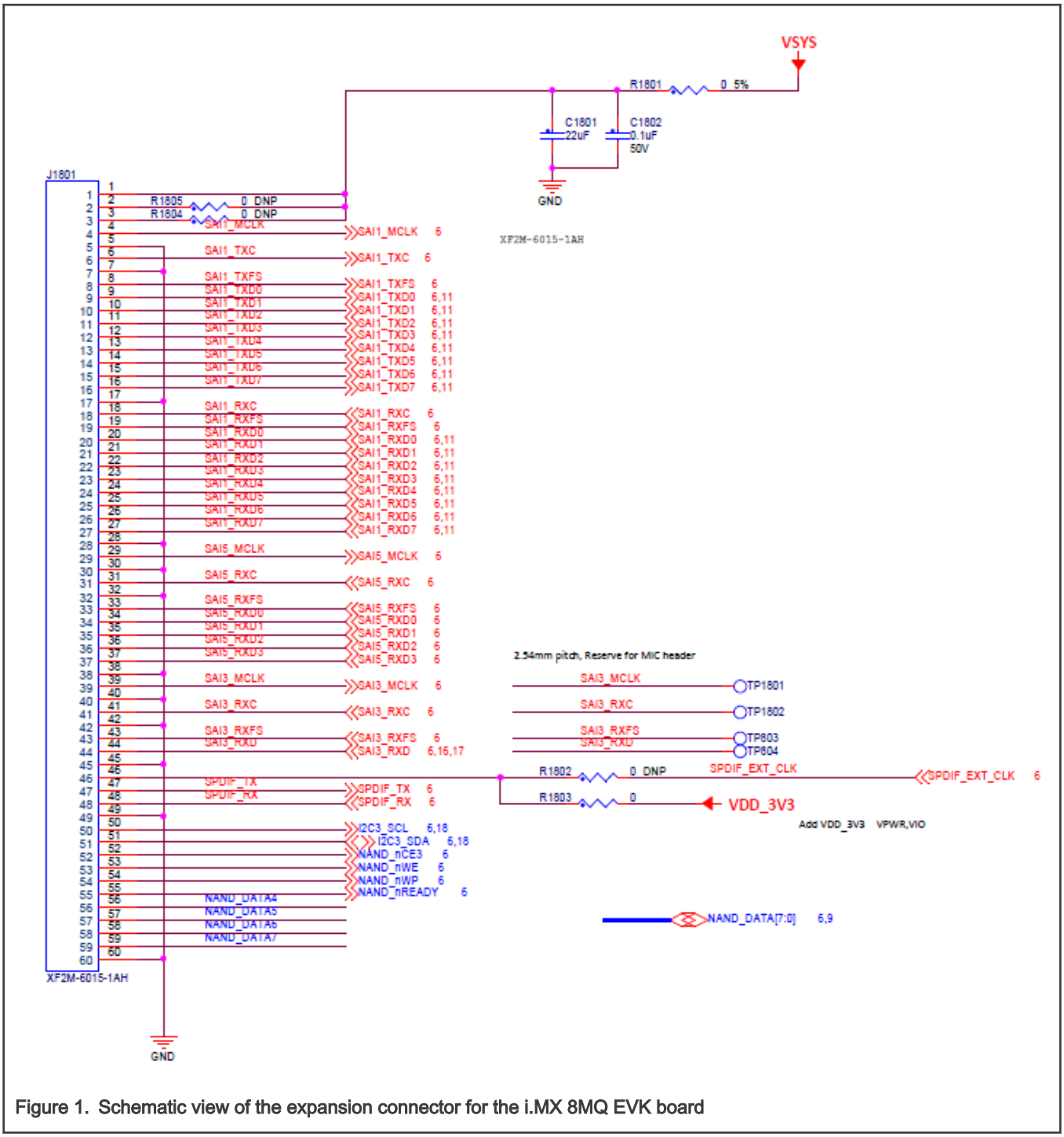


Figure 1. Schematic view of the expansion connector for the i.MX 8MQ EVK board

8.2.5.83 SW_MUX_CTL_PAD_SAI1_RXFS SW MUX Control Register (IOMUXC_SW_MUX_CTL_PAD_SAI1_RXFS)

SW_MUX_CTL Register

Address: 3033_0000h base + 15Ch offset = 3033_015Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved											SION	Reserved	MUX_MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

IOMUXC_SW_MUX_CTL_PAD_SAI1_RXFS field descriptions

Field	Description
31–5 -	This field is reserved. Reserved
4 SION	Software Input On Field Force the select mux mode Input path no matter of MUX_MODE functionality 0 **SION_DISABLED** — Input Path of pad SAI1_RXFS is determined by functionality 1 **SION_ENABLED** — Force Input Path of pad SAI1_RXFS
3 -	This field is reserved. Reserved
MUX_MODE	MUX Mode Select Field Select 1 of 4 iomux modes to be used for pad: SAI1_RXFS 000 **ALT0_SAI1_RX_SYNC** — Select mux mode: ALT0 mux port: RX_SYNC of instance: SAI1 001 **ALT1_SAI5_RX_SYNC** — Select mux mode: ALT1 mux port: RX_SYNC of instance: SAI5 100 **ALT4_CORESIGHT_TRACE_CLK** — Select mux mode: ALT4 mux port: TRACE_CLK of instance: CORESIGHT 101 **ALT5_GPIO4_IO00** — Select mux mode: ALT5 mux port: IO00 of instance: GPIO4

Figure 2. Associated mux control register for the chosen pad (SAI1_RXFS)

Updating the U-Boot Device tree

To get the necessary files, issue the following command:

```
$ bitbake -f -c unpack imx-boot
```

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work/imx8mqevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/dt-bindings/pinctrl/pins-imx8mq.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MQ_IOMUXC_SAI1_RXFS_SAI1_RX_SYNC               0x15C 0x3C4 0x4C4 0x0 0x0
#define MX8MQ_IOMUXC_SAI1_RXFS_SAI5_RX_SYNC               0x15C 0x3C4 0x4E4 0x1 0x1
#define MX8MQ_IOMUXC_SAI1_RXFS_CORESIGHT_TRACE_CLK        0x15C 0x3C4 0x000 0x4 0x0
#define MX8MQ_IOMUXC_SAI1_RXFS_GPIO4_IO0                  0x15C 0x3C4 0x000 0x5 0x0
```

The associated DTS file for the i.MX 8MQ EVK board (<yocto_build_dir>/tmp/work/imx8mqevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mq-evk.dts) shows that the pin is not used, so there is nothing to disable. To use the pin with the GPIO functionality, add the following pin muxing in iomuxc.

```
pinctrl_hog_1: hoggrp-1 {
  fsl,pins = <
   MX8MQ_IOMUXC_SAI1_RXFS_GPIO4_IO0             0x16
  >; };
```

The 0x16 configuration is based on the following pad settings in the IOMUXC_SW_PAD_CTL_PAD_SAI1_RXFS register:

- Drive Strength Field: 45_OHM
- Slew Rate Field: Fast
- Open Drain Enable field: Disabled
- Pull Up Enable Field: Disabled
- Schmitt trigger Enable Field: Disabled
- Lvttl Enable Field: Disabled

Updating the Linux device tree

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work-shared/imx8mqevk/kernel_source/arch/arm64/boot/dts/freescale/pins-imx8mq.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MQ_IOMUXC_SAI1_RXFS_SAI1_RX_SYNC              0x15C 0x3C4 0x4C4 0x0 0x0
#define MX8MQ_IOMUXC_SAI1_RXFS_SAI5_RX_SYNC              0x15C 0x3C4 0x4E4 0x1 0x1
#define MX8MQ_IOMUXC_SAI1_RXFS_CORESIGHT_TRACE_CLK       0x15C 0x3C4 0x000 0x4 0x0
#define MX8MQ_IOMUXC_SAI1_RXFS_GPIO4_IO0                 0x15C 0x3C4 0x000 0x5 0x0
```

The associated DTS file for the i.MX 8MQ EVK (<yocto_build_dir>/tmp/work-shared/imx8mqevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mq-evk.dts) shows that the pin is already used with the SAI1 functionality. Change the status property for the SAI1 from “okay” to “disabled”.

```
&sai1 {
 pinctrl-names = "default", "pcm_b2m", "dsd";
 pinctrl-0 = <&pinctrl_sai1_pcm>;
 pinctrl-1 = <&pinctrl_sai1_pcm_b2m>;
 pinctrl-2 = <&pinctrl_sai1_dsd>;
 assigned-clocks = <&clk IMX8MQ_CLK_SAI1>;
 assigned-clock-parents = <&clk IMX8MQ_AUDIO_PLL1_OUT>;
 assigned-clock-rates = <49152000>;
 clocks = <&clk IMX8MQ_CLK_SAI1_IPG>, <&clk IMX8MQ_CLK_DUMMY>, <&clk IMX8MQ_CLK_SAI1_ROOT>,
<&clk IMX8MQ_CLK_DUMMY>, <&clk IMX8MQ_CLK_DUMMY>, <&clk IMX8MQ_AUDIO_PLL1_OUT>, <&clk
IMX8MQ_AUDIO_PLL2_OUT>;
 clock-names = "bus", "mclk0", "mclk1", "mclk2", "mclk3", "pll8k", "pll11k";
```

```
 fsl,sai-multi-lane;
 fsl,dataline,dsd = <0 0xff 0xff 2 0xff 0x11>;
 dmas = <&sdma2 8 25 0>, <&sdma2 9 25 0>;
 status = "disabled"; }
```

To use the pin with the GPIO functionality, add the following pin muxing to `iomuxc`. If the chosen pad has another pin mux configuration, replace the respective pin muxing with the following to successfully generate the pulse in Linux:

```
pinctrl_hog: hoggrp {
  fsl,pins = <
   MX8MQ_IOMUXC_NAND_READY_B_GPIO3_IO16  0x19
   MX8MQ_IOMUXC_NAND_WE_B_GPIO3_IO17        0x19
   MX8MQ_IOMUXC_NAND_WP_B_GPIO3_IO18        0x19
   MX8MQ_IOMUXC_GPIO1_IO08_GPIO1_IO8        0xd6
   MX8MQ_IOMUXC_GPIO1_IO00_ANAMIX_REF_CLK_32K 0x16
   MX8MQ_IOMUXC_SAI1_RXFS_GPIO4_IO0               0x16
  >; };
```

Adding the first pulse generator in <yocto_build_dir>/tmp/work/imx8mqevk-poky-linux/u-boot-imx/<specified_git_folder>/git/board/freescale/imx8mq_evk/spl.c

Firstly, declare a macro containing the pair of GPIO group and GPIO pin in the file:

```
#define TIMED_GPIO IMX_GPIO_NR(4, 0)
```

Secondly, you need a macro to use a pad as a GPIO. In this case, it already exists:

```
#define USDHC_GPIO_PAD_CTRL (PAD_CTL_PUE | PAD_CTL_DSE1)
```

Now you can add the pulse generation code in the `board_init_f` function, after the BSS clearing part:

```
void board_init_f(ulong dummy) {
int ret;
/* Clear the BSS. */
memset(__bss_start, 0, __bss_end - __bss_start);
gpio_request(TIMED_GPIO, "timed_gpio");
gpio_direction_output(TIMED_GPIO, 1);
gpio_direction_output(TIMED_GPIO, 0);
arch_cpu_init();
```

Adding the second pulse generator in <yocto_build_dir>/tmp/work/imx8mqevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mq_evk.h

The commands responsible for toggling the pin are added in the environment variables for the bootloader, at the load image property. The pin is set before loading the image and reset afterwards:

```
"loadimage=gpio set GPIO4_0;fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image};gpio clear
GPIO4_0\0" \
```

To automate the process, create a patch which contains the U-Boot modifications in the `spl.c`, `imx8mq-evk.dts`, and `imx8mq_evk.h` files. Add the details after the last commands that describe the nature of the patch:

```
$ git add board/freescale/imx8mq_evk/spl.c
$ git add arch/arm/dts/imx8mq-evk.dts
$ git add include/configs/imx8mq_evk.h
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: <yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx. Add the name of the patch into the source location identifier in the Yocto recipe for U-Boot (<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2020.04.bb):

```
SRC_URI = " ...\
file://<patch_name>.patch \
"
```

To check that the patch works after the modifications, apply the following commands, after which the files should contain the following changes:

```
$ bitbake -f -c clean u-boot-imx
$ bitbake u-boot-imx
$ bitbake imx-boot
```

Adding the third pulse generator

For this stage, modify the Yocto recipe extension for psplash so that it can fetch the additional libgpiod library necessary for the toggling of the GPIO (<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend):

```
DEPENDS = "libgpiod"
RDEPENDS_${PN} = "libgpiod"
RDEPENDS_${PN}-dev = "libgpiod"
```

To get the necessary psplash files, issue the following command:

```
$ bitbake -f -c unpack psplash
```

After fetching the necessary files, the makefile (<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/Makefile.am) is modified to include the lgpiod:

```
AM_CFLAGS = $(GCC_FLAGS) -D_GNU_SOURCE -lgpiod
GCC_FLAGS := $(GCC_FLAGS) -Lusr/lib -Iusr/include -lgpiod
LD_FLAGS = $(LD_FLAGS) -lgpiod
```

Modify the source code (<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/psplash.c). The first step is to include the library:

```
#include "gpiod.h"
```

In the main function, declare some additional variables responsible for selecting the bank and the line for the GPIO. The banks are zero-indexed, so the number of the GPIO bank must be decremented by one:

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int gpio_line = 0;
char dev[] = "/dev/gpiochip3";
```

Before sending the first command to the frame buffer (clearing the background), the program should acquire control of the pin, toggle it, and release it.

```
chip = gpiod_chip_open(dev); if (!chip) return -1;
  line = gpiod_chip_get_line(chip, gpio_line); if (!line) {
  gpiod_chip_close(chip); return -1; }
 req = gpiod_line_request_output(line, "SIGNAL", 2); if (req) {
 gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 1) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 1);
  gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 0) != 0){
```

```
  printf("Impossible to change line %d value to %d \n", gpio_line, 0);
  gpiod_chip_close(chip); return -1; }
 gpiod_line_release(line);
 gpiod_chip_close(chip);
```

After applying all the modifications, build the `psplash` and the image using the following commands:

```
$ bitbake psplash
$ bitbake imx-image-core
```

To automate this process, create a patch which contains the modifications in the `Makefile.am` and `psplash.c` files. Add the details after the last commands describing the nature of the patch:

```
$ git add Makefile.am
$ git add psplash.c
$ git commit -s
$ git format-patch HEAD~1
```

The resulting patch is then copied to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-core/psplash/files`. The name of the patch is then added into the source location identifier in the Yocto Recipe extension for psplash (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`).

```
SRC_URI += " \
            file://psplash-start.service \
            file://psplash-basic.service \
            file://psplash-network.service \
            file://psplash-quit.service \
       file://<patch_name>.patch \
            "
```

To check that the patch works after the modifications, apply the following commands, after which the `psplash.c` and `Makefile.am` files should contain the following changes:

```
$ bitbake -f -c clean psplash
$ bitbake psplash
$ bitbake imx-image-core
```

Measuring the total time with the logic analyzer

This part starts by setting up all the preliminary connections, such as the power supply, debug port, download port, and MIPI-DSI to HDMI connections. If the HDMI display is not connected and functional, the `psplash` pulse generation does not work, because no frame buffers exist.

Build both the bootloader and Linux image using `bitbake`. After that, they are written to the board using the UUU program, in which you can choose between writing to the on-board eMMC or the SD card.

The measuring stand is set up by connecting the logic analyzer to the board. The nRST signal is used as a starting reference and it is on the JTAG connector at pin 10.

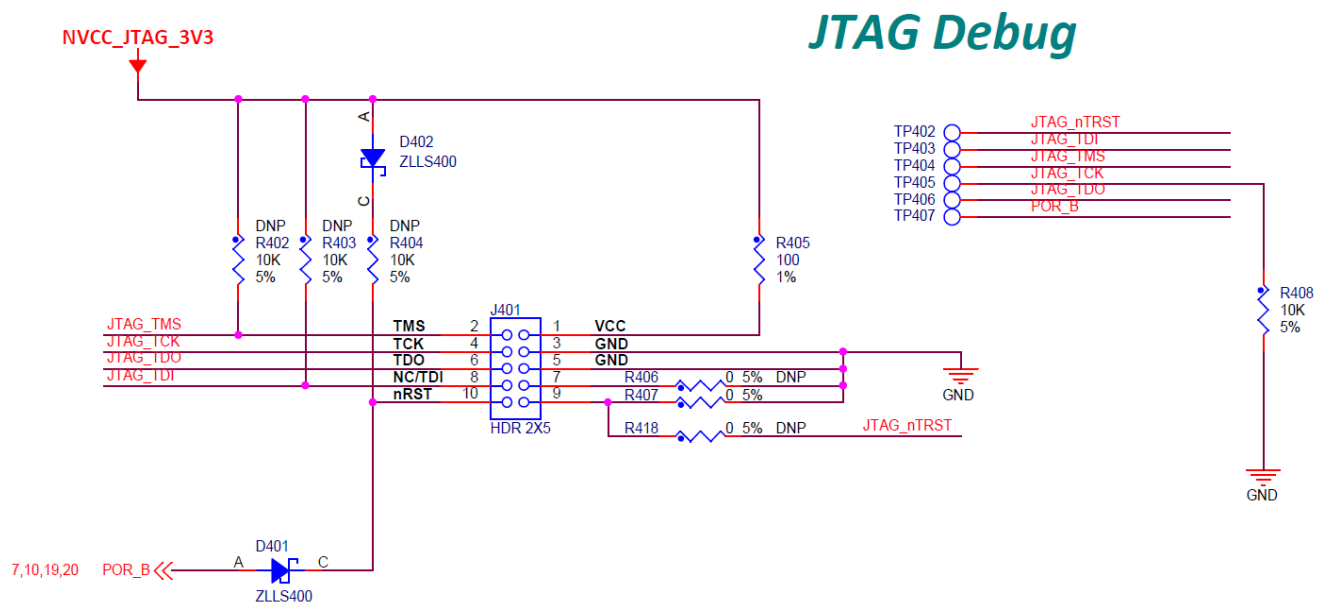


Figure 3. Schematic view of the JTAG connector available on the i.MX 8MQ EVK board

To capture the rising edges of the chosen pin, connect the second probe of the analyzer to the 19th pin on the J1801 expansion connector. The evaluation board features an FPC socket, so plug an FPC to 60x0.5 extension board into the receptacle. This ensures easy access to the required pins.

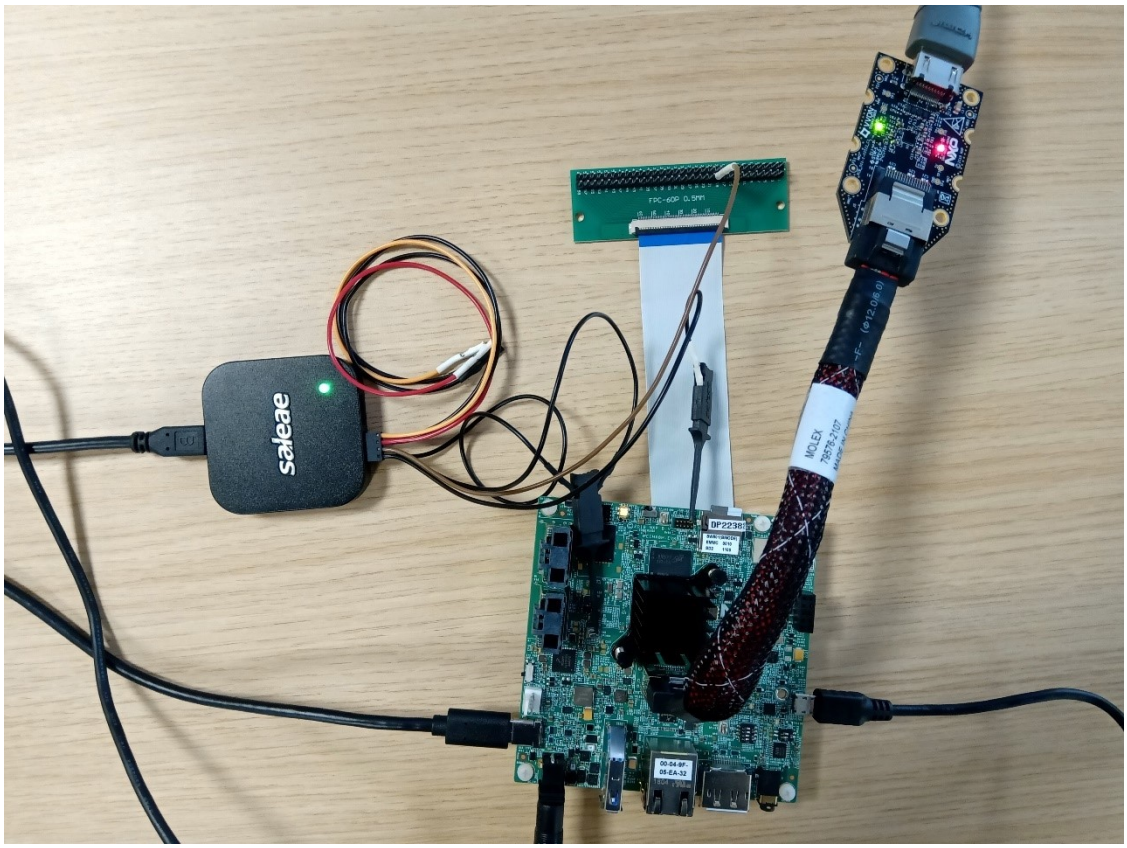


Figure 4. Measurement setup on the i.MX 8MQ EVK board

After checking that both probes are referenced to the board's ground, start the logic analyzer software, where you can set up the probe parameters and the time frame.

Configure the board to start in the internal development mode, boot from the desired storage space, and then power it on. After the `psplash` screen disappears, press the reset button on the board and start the recording on the logic analyzer software.

You should see a rising edge on the nRST pin, followed by three pulses on the chosen pin. At this moment, stop the recording and place the measurement flags to find out the time for each phase. The boot time of the board in this configuration results from the sum of the elapsed time for each stage.

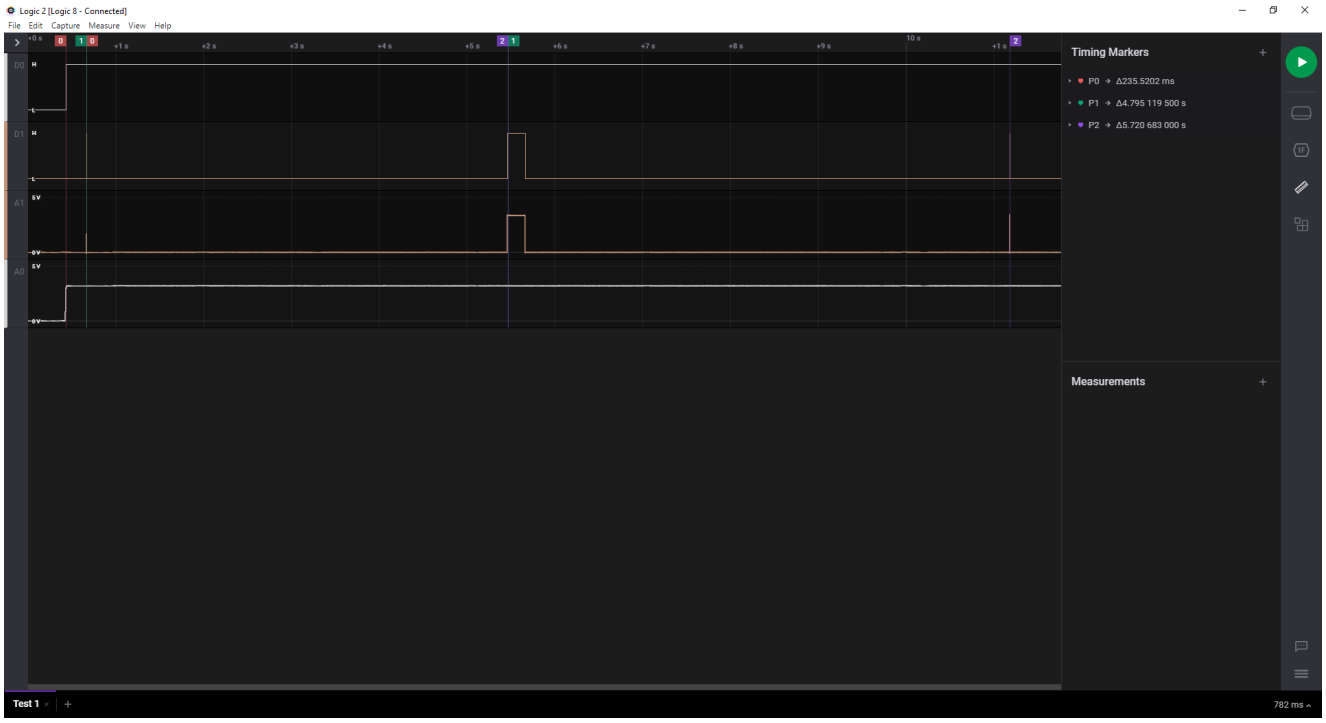


Figure 5. Measurements on the boot-time for i.MX 8MQ EVK board

3.2 i.MX 8M Plus

Choosing the pins

The chosen pin is the 24th pin on the J21 expansion connector on the specified board. In the schematics for the baseboard, the pin is used for the ECSPI 2 peripheral with the ECSPI2_SS0 function. Searching the specified signal in the reference manual for the associated pin returns an alternate function of GPIO5_IO[13].

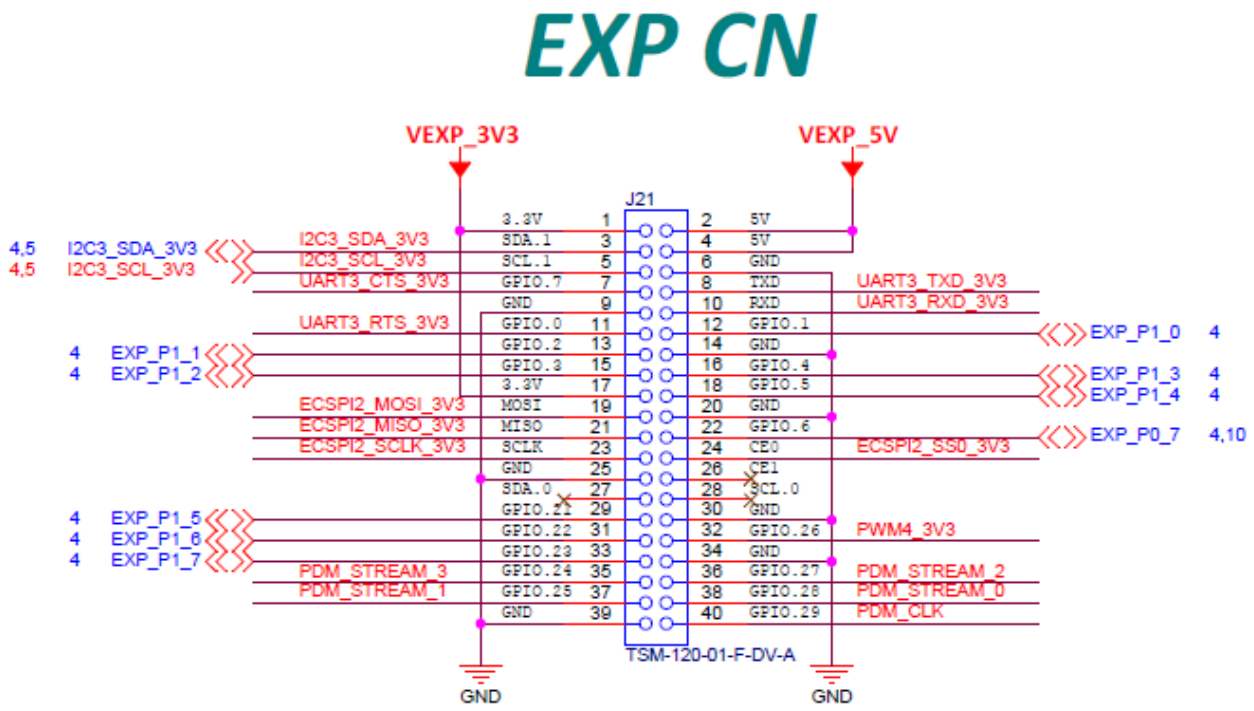


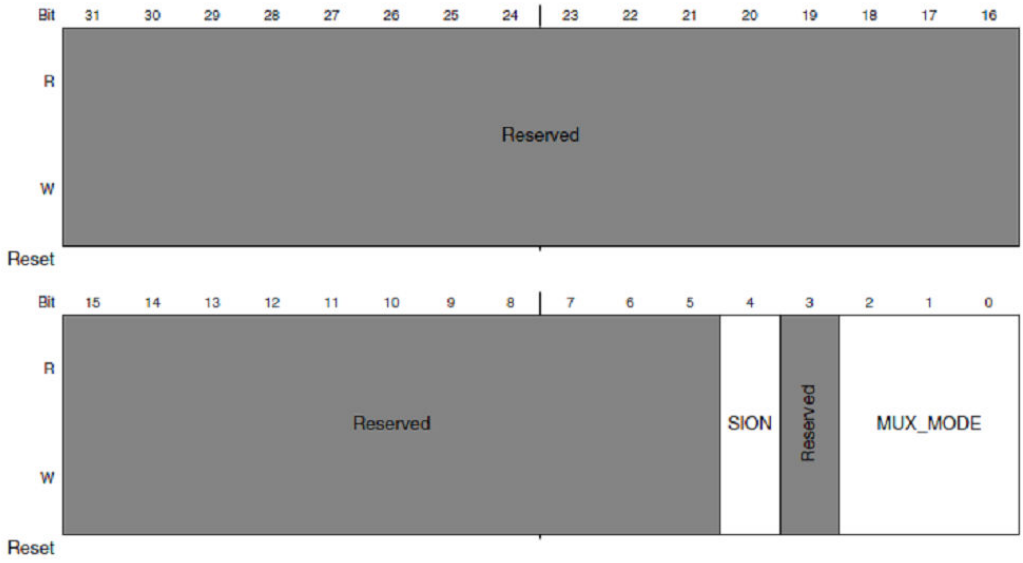
Figure 6. Schematic view of the expansion connector for the i.MX 8MP EVK board

Figure 7. Associated mux control register for the chosen pad (ECSPI2_SS0)

8.2.4.123 SW_MUX_CTL_PAD_ECSPI2_SS0 SW MUX Control Register (IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0)

SW_MUX_CTL Register

Address: 3033_0000h base + 1FCh offset = 3033_01FCh



IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0 field descriptions

Field	Description
31–5 -	This field is reserved. Reserved
4 SION	Software Input On Field. Force the selected mux mode Input path no matter of MUX_MODE functionality. 1 **ENABLED** — Force input path of pad ECSPI2_SS0 0 **DISABLED** — Input Path is determined by functionality
3 -	This field is reserved. Reserved
MUX_MODE	MUX Mode Select Field. Select 1 of 6 iomux modes to be used for pad: ECSPI2_SS0. 000 **ALT0_ECSPI2_SS0** — Select mux mode: ALT0 mux port: ECSPI2_SS0 of instance: ecspi2 001 **ALT1_UART4_RTS_B** — Select mux mode: ALT1 mux port: UART4_RTS_B of instance: uart4 010 **ALT2_I2C4_SDA** — Select mux mode: ALT2 mux port: I2C4_SDA of instance: i2c4 100 **ALT4_CCM_CLKO2** — Select mux mode: ALT4 mux port: CCM_CLKO2 of instance: ccm 101 **ALT5_GPIO5_IO[13]** — Select mux mode: ALT5 mux port: GPIO5_IO13 of instance: gpio5

Updating the U-Boot device tree

Issue the following command to get the necessary files:

```
$ bitbake -f -c unpack imx-boot
```

To update the device tree, find out the macro associated to the desired functionality. This information is in the `<yocto_build_dir>/tmp/work/imx8mpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/`

imx8mp-pinfunc.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MP_IOMUXC_ECSPI2_SS0__ECSPI2_SS0               0x1FC 0x45C 0x574 0x0 0x1
#define MX8MP_IOMUXC_ECSPI2_SS0__UART4_DCE_RTS            0x1FC 0x45C 0x5FC 0x1 0x3
#define MX8MP_IOMUXC_ECSPI2_SS0__UART4_DTE_CTS            0x1FC 0x45C 0x000 0x1 0x0
#define MX8MP_IOMUXC_ECSPI2_SS0__I2C4_SDA                 0x1FC 0x45C 0x5C0 0x2 0x4
#define MX8MP_IOMUXC_ECSPI2_SS0__CCM_CLKO2                0x1FC 0x45C 0x000 0x4 0x0
#define MX8MP_IOMUXC_ECSPI2_SS0__GPIO5_IO13               0x1FC 0x45C 0x000 0x5 0x0
```

The associated DTS file for the i.MX 8MP EVK board (<yocto_build_dir>/tmp/work/imx8mpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mp-evk.dts) shows that the pin is not used, because there is nothing to disable. To use the pin with the GPIO functionality, add the following pin muxing in iomuxc:

```
pinctrl_hog: hoggrp {
   fsl,pins = <
    MX8MP_IOMUXC_ECSPI2_SS0__GPIO5_IO1        0x16
   >; };
```

The 0x16 configuration is based on the following pad settings in the IOMUXC_SW_PAD_CTL_PAD_ECSPI2_SS0 register:

- Drive Strength Field: DSE_X6
- Slew Rate Field: Fast
- Open Drain Enable field: Disabled
- Pull Up/Down Config Field: Weak pull-down
- Input Select Field: CMOS
- Pull Select Field: Pull Disabled

Updating the Linux device tree

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work-shared/imx8mpevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mp-pinfunc.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MP_IOMUXC_ECSPI2_SS0__ECSPI2_SS0               0x1FC 0x45C 0x574 0x0 0x1
#define MX8MP_IOMUXC_ECSPI2_SS0__UART4_DCE_RTS            0x1FC 0x45C 0x5FC 0x1 0x3
#define MX8MP_IOMUXC_ECSPI2_SS0__UART4_DTE_CTS            0x1FC 0x45C 0x000 0x1 0x0
#define MX8MP_IOMUXC_ECSPI2_SS0__I2C4_SDA                 0x1FC 0x45C 0x5C0 0x2 0x4
#define MX8MP_IOMUXC_ECSPI2_SS0__CCM_CLKO2                0x1FC 0x45C 0x000 0x4 0x0
#define MX8MP_IOMUXC_ECSPI2_SS0__GPIO5_IO13               0x1FC 0x45C 0x000 0x5 0x0
```

The associated DTS file for the i.MX 8MP EVK (<yocto_build_dir>/tmp/work-shared/imx8mpevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mp-evk.dts) shows that the pin is already used with the ECSPI2 functionality. Change the status property for the ECSPI2 from “okay” to “disabled”.

```
&ecspi2 {
 #address-cells = <1>;
 #size-cells = <0>;
 fsl,spi-num-chipselects = <1>;
 pinctrl-names = "default";
 pinctrl-0 = <&pinctrl_ecspi2 &pinctrl_ecspi2_cs>;
 cs-gpios = <&gpio5 13 GPIO_ACTIVE_LOW>;
 status = "disabled";
 spidev1: spi@0 {
  reg = <0>;
```

```
  compatible = "rohm,dh2228fv";
  spi-max-frequency = <500000>; }; };
```

To use the pin with the GPIO functionality, add the following pin muxing to `iomuxc`. If the chosen pad has another pin mux configuration, replace the respective pin muxing with the following to successfully generate the pulse in Linux:

```
pinctrl_hog: hoggrp {
  fsl,pins = <
   MX8MP_IOMUXC_HDMI_DDC_SCL__HDMIMIX_HDMI_SCL 0x400001c3
   MX8MP_IOMUXC_HDMI_DDC_SDA__HDMIMIX_HDMI_SDA 0x400001c3
   MX8MP_IOMUXC_HDMI_HPD__HDMIMIX_HDMI_HPD  0x40000019
   MX8MP_IOMUXC_HDMI_CEC__HDMIMIX_HDMI_CEC  0x40000019
   MX8MP_IOMUXC_ECSPI2_SS0__GPIO5_IO13   0x16
  >;
 };
```

Adding the first pulse generator in <yocto_build_dir>/tmp/work/imx8mpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/board/freescale/imx8mp_evk/spl.c

Firstly, declare a macro containing the pair of GPIO group and GPIO pin in the file:

```
#define TIMED_GPIO IMX_GPIO_NR(5, 13)
```

Secondly, create a macro to use a pad as a GPIO. The macro is already created in this case:

```
#define USDHC_GPIO_PAD_CTRL (PAD_CTL_HYS | PAD_CTL_DSE1)
```

Add the pulse generation code in the `board_init_f` function, after the BSS clearing part.

```
void board_init_f(ulong dummy) {
 int ret;
 /* Clear the BSS. */
 memset(__bss_start, 0, __bss_end - __bss_start);
 gpio_request(TIMED_GPIO, "timed_gpio");
 gpio_direction_output(TIMED_GPIO, 1);
 gpio_direction_output(TIMED_GPIO, 0);
 arch_cpu_init();
```

Adding the second pulse generator in <yocto_build_dir>/tmp/work/imx8mpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mp_evk.h

Add the commands responsible for pin toggling into the environment variables for the bootloader at the load image property. The pin is set before image loading and reset afterwards:

```
“loadimage=gpio set GPIO5_13;fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image};gpio clear
GPIO5_13\0” \
```

To automate the process, create a patch that contains the U-Boot modifications in the `spl.c`, `imx8mp-evk.dts`, and `imx8mp_evk.h` files. Add the details after the last commands describing the nature of the patch:

```
$ git add board/freescale/imx8mp_evk/spl.c
$ git add arch/arm/dts/imx8mp-evk.dts
$ git add include/configs/imx8mp_evk.h
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: <yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx. Add the name of the patch into the source location identifier in the Yocto recipe for u-boot (<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2020.04.bb).

```
SRC_URI = " ...\
file://<patch_name>.patch \
 "
```

Apply the following commands to check that the patch works after the modifications, after which the files should contain the following changes:

```
$ bitbake -f -c clean u-boot-imx
$ bitbake u-boot-imx
$ bitbake imx-boot
```

Adding the third pulse generator

For this stage, modify the Yocto recipe extension for psplash so that it can fetch the additional libgpiod library necessary to toggle the GPIO (<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend):

```
DEPENDS = "libgpiod"
RDEPENDS_${PN} = "libgpiod"
RDEPENDS_${PN}-dev = "libgpiod"
```

Issue the following command to get the necessary psplash files:

```
$ bitbake -f -c unpack psplash
```

After fetching the necessary files, modify the makefile (<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/Makefile.am) to include the lgpiod.

```
AM_CFLAGS = $(GCC_FLAGS) -D_GNU_SOURCE -lgpiod
GCC_FLAGS := $(GCC_FLAGS) -Lusr/lib -Iusr/include -lgpiod
LD_FLAGS = $(LD_FLAGS) -lgpiod
```

Modify the source code (<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/psplash.c). The first step is including the library:

```
#include "gpiod.h"
```

Declare some additional variables in the main function, responsible for selecting the bank and the line for the GPIO. The banks are zero-indexed, so the number of the GPIO bank must be decremented by one:

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int gpio_line = 13;
char dev[] = "/dev/gpiochip4";
```

The program should acquire control of the pin, toggle it, and release it before sending the first command to the frame buffer (clearing the background).

```
chip = gpiod_chip_open(dev); if (!chip) return -1;
  line = gpiod_chip_get_line(chip, gpio_line); if (!line) {
  gpiod_chip_close(chip); return -1; }
 req = gpiod_line_request_output(line, "SIGNAL", 2); if (req) {
 gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 1) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 1);
  gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 0) != 0){
```

```
  printf("Impossible to change line %d value to %d \n", gpio_line, 0);
  gpiod_chip_close(chip); return -1; }
 gpiod_line_release(line);
 gpiod_chip_close(chip);
```

After applying all the modifications, you can build the `psplash` and the image using the following commands:

```
$ bitbake psplash
$ bitbake imx-image-core
```

To automate this process, create a patch that contains the modifications in the `Makefile.am` and `psplash.c` files. Add the details after the last commands describing the nature of the patch:

```
$ git add Makefile.am
$ git add psplash.c
$ git commit -s
$ git format-patch HEAD~1
```

The resulting patch is then copied to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-core/psplash/files`. The name of the patch is then added in the source location identifier in the Yocto Recipe extension for psplash (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`).

```
SRC_URI += " \
            file://psplash-start.service \
            file://psplash-basic.service \
            file://psplash-network.service \
            file://psplash-quit.service \
       file://<patch_name>.patch \
            "
```

Apply the following commands to check that the patch works after the modifications, after which the `psplash.c` and `Makefile.am` should contain the following changes:

```
$ bitbake -f -c clean psplash
$ bitbake psplash
$ bitbake imx-image-core
```

Measuring the total time with the logic analyzer

This part starts by setting up all the preliminary connections, such as the power supply, debug port, download port, and HDMI connections. If the HDMI display is not connected and functional, the `psplash` pulse generation does not work, because there are no frame buffers.

Build both the bootloader and Linux images using `bitbake`. After that, write them to the board using the UUU program, in which you can choose between writing to the on-board eMMC or the SD card.

Set up the measuring stand by connecting the logic analyzer to the board. The JTAG_RESET signal is used as a starting reference and it is on the JTAG connector at pin 10.

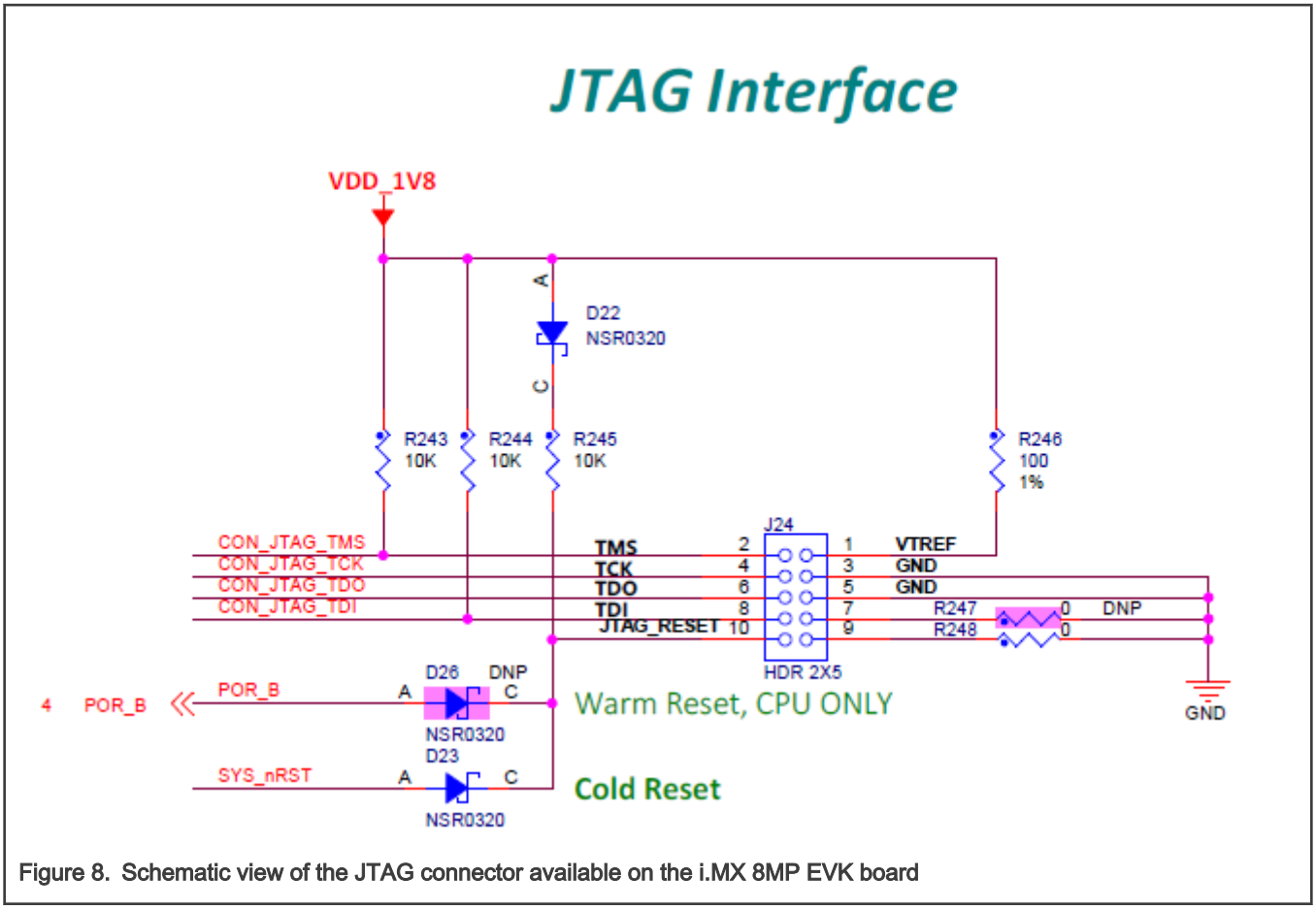


Figure 8. Schematic view of the JTAG connector available on the i.MX 8MP EVK board

To capture the rising edges of the chosen pin, connect the second probe of the analyzer to the 24th pin on the J21 expansion connector.

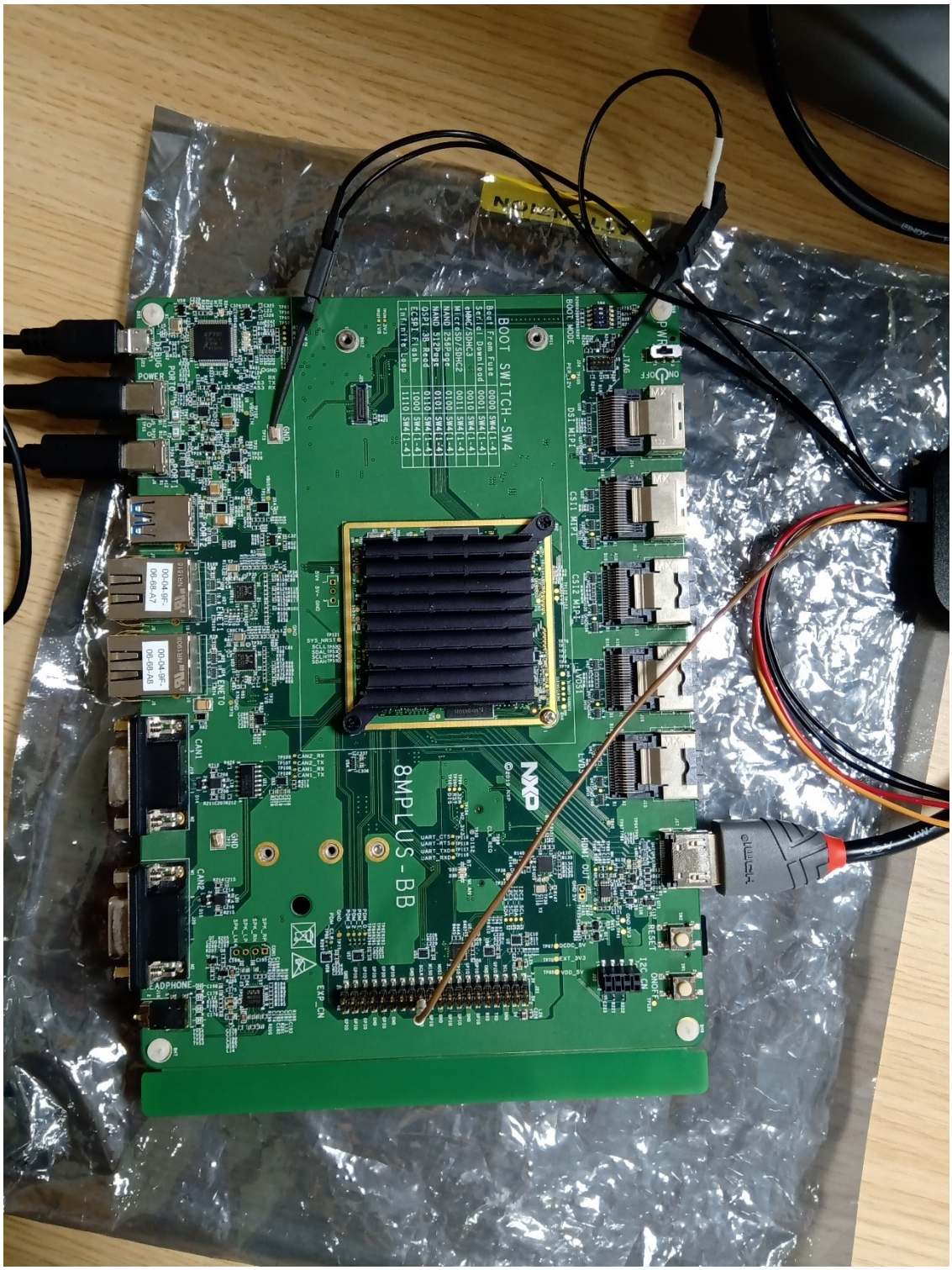


Figure 9. Measurement setup on the i.MX 8MP EVK board

After checking that both probes are referenced to the board's ground, start the logic analyzer software, where the probe parameters and the time frame is set up.

Configure the board to start in the internal development mode, boot from the desired storage space, and then power it on. After the `psplash` screen disappears, press the reset button on the board and start the recording on the logic analyzer software.

You shall see a rising edge on the JTAG_RESET pin, followed by three pulses on the chosen pin. Stop the recording and place the measurement flags to find out the time for each phase. The boot time of the board in this configuration results from the sum of the elapsed time for each stage.

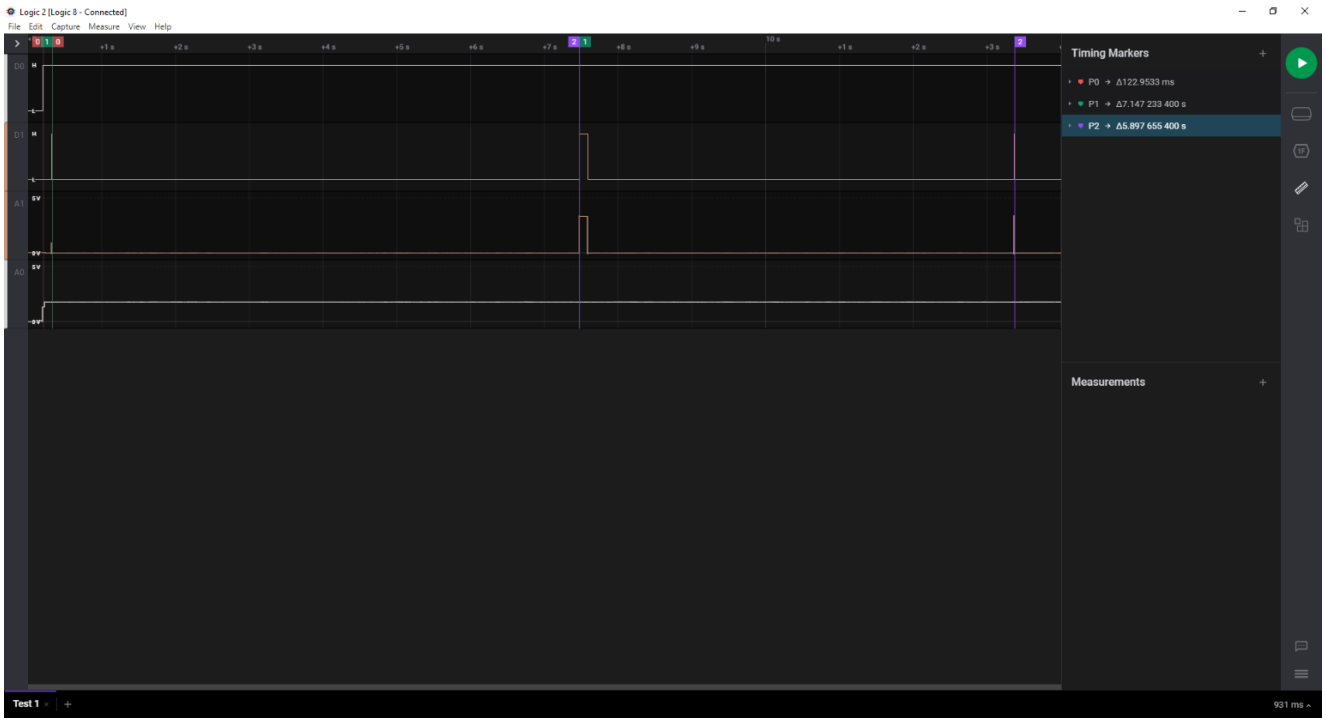


Figure 10. Measurements on the boot-time for i.MX 8MP EVK board

3.3 i.MX 8M Mini

Choosing the pins

The chosen pin is the 24th pin on the J1003 expansion connector on the specified board. In the schematics for the baseboard, the pin is used for the ECSPI 2 peripheral with the ECSPI2_SS0 function. Searching for the specified signal in the reference manual for the associated pin returns an alternate function of GPIO5_IO[13].

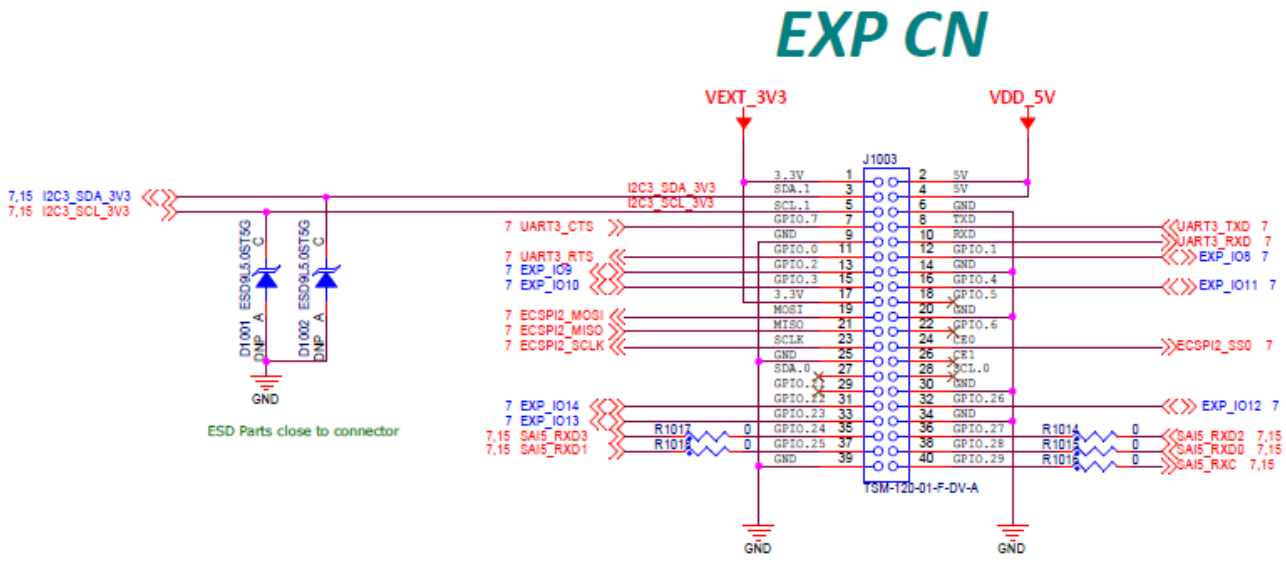
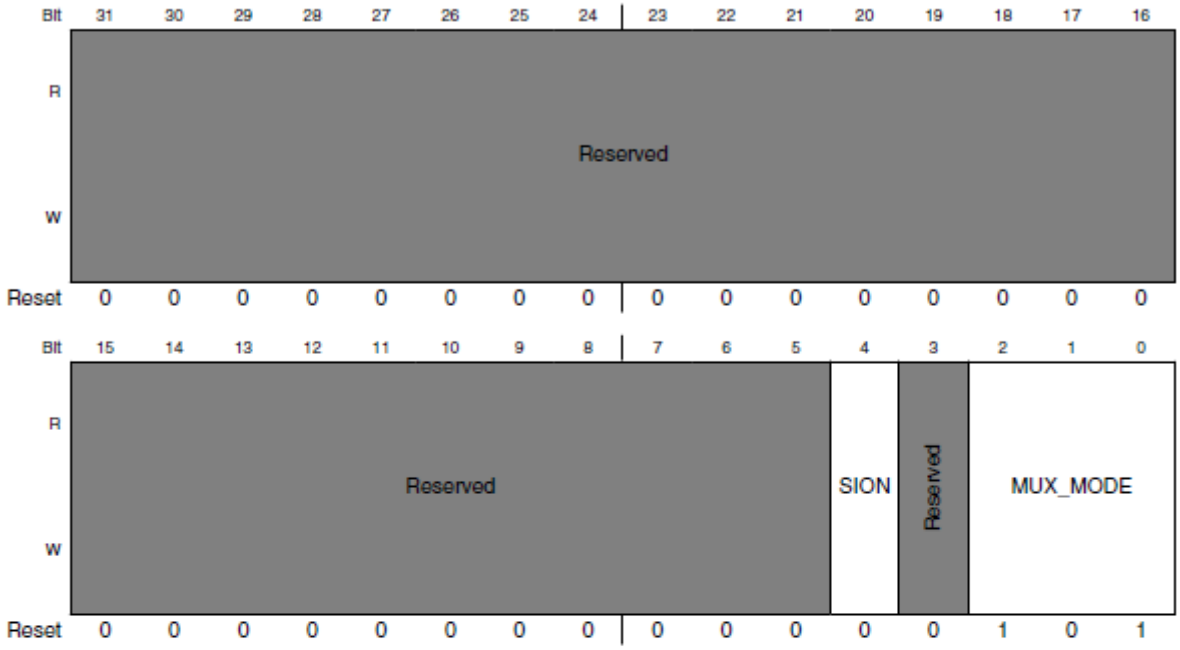


Figure 11. Schematic view of the expansion connector for the i.MX 8MM EVK board

8.2.5.128 Pad Mux Register (IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0)

Address: 3033_0000h base + 210h offset = 3033_0210h



IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0 field descriptions

Field	Description
31–5 -	This field is reserved. Reserved
4 SION	Software Input On Field. Force the selected mux mode input path no matter of MUX_MODE functionality. 1 **ENABLED** — Force input path of pad ECSPI2_SS0 0 **DISABLED** — Input Path is determined by functionality of the selected mux mode (regular).
3 -	This field is reserved. Reserved
MUX_MODE	MUX Mode Select Field. Select 1 of 3 iomux modes to be used for pad: ECSPI2_SS0. **NOTE:** Pad ECSPI2_SS0 is involved in Daisy Chain. 000 **ALT0** — Select signal ECSPI2_SS0 001 **ALT1** — Select signal UART4_RTS_B - Configure register IOMUXC_UART4_RTS_B_SELECT_INPUT for mode ALT1. 101 **ALT5** — Select signal GPIO5_IO13

Figure 12. Associated mux control register for the chosen pad (ECSPI2_SS0)

Updating the U-Boot Device tree

Issue the following command to get the necessary files:

```
$ bitbake -f -c unpack imx-boot
```

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work/imx8mmevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mm-pinfunc.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MM_IOMUXC_ECSPI2_SS0_ECSPI2_SS0                0x210 0x478 0x000 0x0 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_UART4_DCE_RTS_B           0x210 0x478 0x508 0x1 0x1
#define MX8MM_IOMUXC_ECSPI2_SS0_UART4_DTE_CTS_B           0x210 0x478 0x000 0x1 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_GPIO5_IO13                0x210 0x478 0x000 0x5 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_TPSMP_HDATA15             0x210 0x478 0x000 0x7 0x0
```

The associated DTS file for the i.MX 8MM EVK board (<yocto_build_dir>/tmp/work/imx8mmevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mm-evk.dts) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin muxing into iomuxc.

```
pinctrl_hog_1: hoggrp-1 {
fsl,pins = <
MX8MM_IOMUXC_ECSPI2_SS0_GPIO5_IO13 0x16
>;
};
```

The 0x16 configuration is based on the following pad settings in the IOMUXC_SW_PAD_CTL_PAD_ECSPI2_SS0 register:

- Drive Strength Field: X6
- Slew Rate Field: Fast
- Open Drain Enable field: Disabled
- Control IO ports PS: Pull-down resistors
- Hysteresis Enable Field: CMOS
- Pull Resistors Enable Field: Pull Disabled

Updating the Linux device tree

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work-shared/imx8mmevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mm-pinfunc.h file. However, the use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MM_IOMUXC_ECSPI2_SS0_ECSPI2_SS0                0x210 0x478 0x000 0x0 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_UART4_DCE_RTS_B           0x210 0x478 0x508 0x1 0x1
#define MX8MM_IOMUXC_ECSPI2_SS0_UART4_DTE_CTS_B           0x210 0x478 0x000 0x1 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_GPIO5_IO13                0x210 0x478 0x000 0x5 0x0
#define MX8MM_IOMUXC_ECSPI2_SS0_TPSMP_HDATA15             0x210 0x478 0x000 0x7 0x0
```

The associated DTS file for the i.MX 8MM EVK (<yocto_build_dir>/tmp/work-shared/imx8mmevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mm-evk.dts) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin muxing into iomuxc. If the chosen pad has another pin mux configuration, replace the respective pin muxing with the following to successfully generate the pulse in Linux:

```
pinctrl_hog: hoggrp {
fsl,pins = <
MX8MM_IOMUXC_ECSPI2_SS0_GPIO5_IO13 0x16
>;
};
```

Add the first pulse generator into <yocto_build_dir>/tmp/work/imx8mmevk-poky-linux/u-boot-imx/<specified_git_folder>/git /board/freescale/imx8mm_evk/spl.c.

Firstly, declare a macro containing the pair of GPIO group and GPIO pin in the file:

```
#define TIMED_GPIO IMX_GPIO_NR(5, 13)
```

Secondly, there must be a macro to use a pad as a GPIO. In this case, the macro already exists:

```
#define USDHC_GPIO_PAD_CTRL (PAD_CTL_HYS | PAD_CTL_DSE1)
```

Add the pulse generation code into the board_init_f function, after the BSS clearing part.

```
void board_init_f(ulong dummy) {
 int ret;
 /* Clear the BSS. */
 memset(__bss_start, 0, __bss_end - __bss_start);
 gpio_request(TIMED_GPIO, "timed_gpio");
 gpio_direction_output(TIMED_GPIO, 1);
 gpio_direction_output(TIMED_GPIO, 0);
 arch_cpu_init();
```

Adding the second pulse generator in <yocto_build_dir>/tmp/work/imx8mmevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mm_evk.h

The commands to toggle the pin are added in the environment variables for the bootloader, at the load image property. Set the pin before loading the image and reset it afterwards:

```
"loadimage=gpio set GPIO5_13;fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image};gpio clear GPIO5_13\0" \
```

To automate the process, create a patch, which contains the U-Boot modifications in the spl.c, imx8mm-evk.dts, and imx8mm_evk.h files. Add the details after the last commands describing the nature of the patch:

```
$ git add board/freescale/imx8mm_evk/spl.c
$ git add arch/arm/dts/imx8mm-evk.dts
$ git add include/configs/imx8mm_evk.h
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: <yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx. Add the name of the patch into the source location identifier in the Yocto recipe for U-Boot (<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2020.04.bb):

```
SRC_URI = " ...\
file://<patch_name>.patch \
 "
```

Apply the following commands to check that the patch works after the modifications, after which the files should contain the following changes:

```
$ bitbake -f -c clean u-boot-imx
$ bitbake u-boot-imx
$ bitbake imx-boot
```

Adding the third pulse generator

For this stage, modify the Yocto recipe extension for `psplash` so that it can fetch the additional `libgpiod` library necessary to toggle the GPIO (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`):

```
DEPENDS = "libgpiod"
RDEPENDS_${PN} = "libgpiod"
RDEPENDS_${PN}-dev = "libgpiod"
```

Issue the following command to get the necessary `psplash` files:

```
$ bitbake -f -c unpack psplash
```

After fetching the necessary files, modify the makefile (`<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/Makefile.am`) to include the `lgpiod`:

```
AM_CFLAGS = $(GCC_FLAGS) -D_GNU_SOURCE -lgpiod
GCC_FLAGS := $(GCC_FLAGS) -Lusr/lib -Iusr/include -lgpiod
LD_FLAGS = $(LD_FLAGS) -lgpiod
```

Modify the source code (`<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/psplash.c`). The first step is to include the library:

```
#include "gpiod.h"
```

Declare some additional variables in the main function, responsible for selecting the bank and the line for the GPIO. The banks are zero-indexed, so the number of the GPIO bank must be decremented by one:

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int gpio_line = 13;
char dev[] = "/dev/gpiochip4";
```

Before sending the first command to the frame buffer (clearing the background), the program should acquire control of the pin, toggle it, and release it.

```
chip = gpiod_chip_open(dev); if (!chip) return -1;
  line = gpiod_chip_get_line(chip, gpio_line); if (!line) {
  gpiod_chip_close(chip); return -1; }
 req = gpiod_line_request_output(line, "SIGNAL", 2); if (req) {
 gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 1) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 1);
  gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 0) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 0);
  gpiod_chip_close(chip); return -1; }
 gpiod_line_release(line);
 gpiod_chip_close(chip);
```

After applying all the modifications, build the `psplash` and the image using the following commands:

```
$ bitbake psplash
$ bitbake imx-image-core
```

To automate this process, create a patch that contains the modifications in the `Makefile.am` and `psplash.c` files. Add the details after the last commands describing the nature of the patch:

```
$ git add Makefile.am
$ git add psplash.c
```

```
$ git commit -s
$ git format-patch HEAD~1
```

The resulting patch is then copied to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-core/psplash/files`. The name of the patch is then added into the source location identifier in the Yocto Recipe extension for psplash (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`).

```
SRC_URI += " \
            file://psplash-start.service \
            file://psplash-basic.service \
            file://psplash-network.service \
            file://psplash-quit.service \
       file://<patch_name>.patch \
            "
```

Apply the following commands to check that the patch works after the modifications, after which the `psplash.c` and `Makefile.am` should contain the following changes:

```
$ bitbake -f -c clean psplash
$ bitbake psplash
$ bitbake imx-image-core
```

Measuring the total time with the logic analyzer

This part starts by setting up all the preliminary connections, such as the power supply, debug port, download port, and MIPI-DSI to HDMI connections. If the HDMI display is not connected and functional, the `psplash` pulse generation does not work, because there are no frame buffers.

Both the bootloader and the Linux image must be built using `bitbake`. After that, write them to the board using the UUU program, in which you can choose between writing to the on-board eMMC or the SD card.

The measuring stand is set up by connecting the logic analyzer to the board. The nRST signal is used as a starting reference, which can be found on the JTAG connector at pin 10.

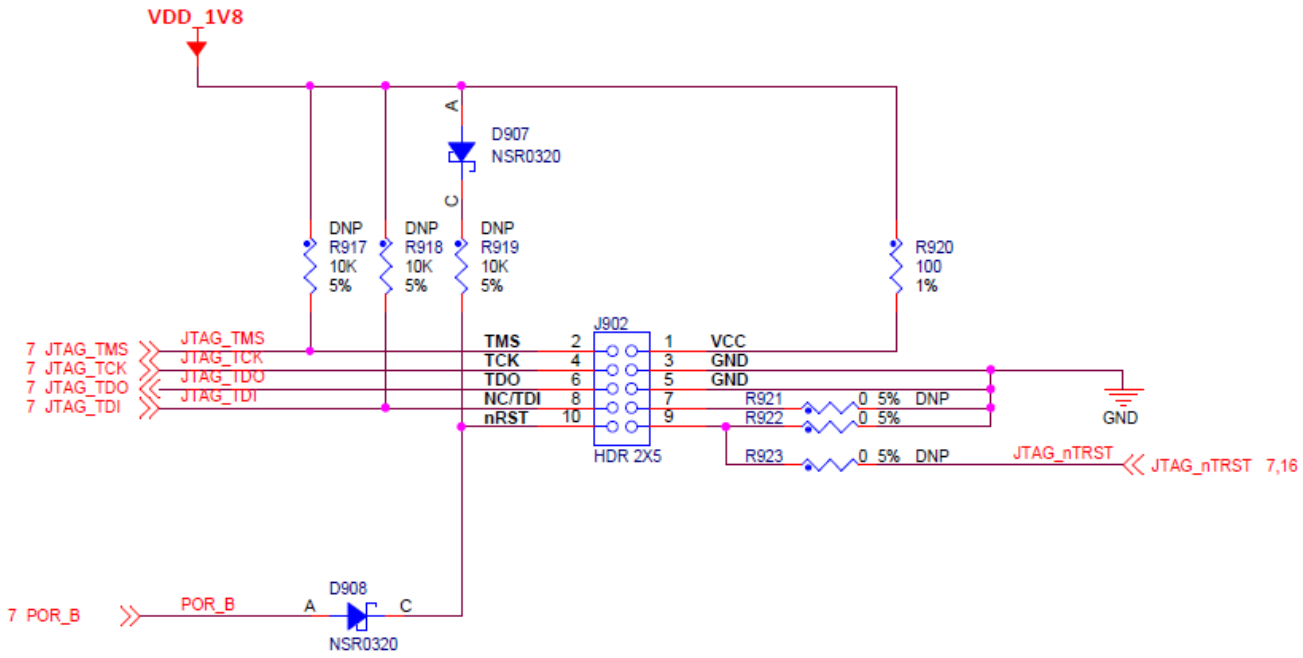


Figure 13. Schematic view of the JTAG connector available on the i.MX 8MM EVK board

To capture the rising edges of the chosen pin, connect the second probe of the analyzer to the 24th pin on the J1003 expansion connector.

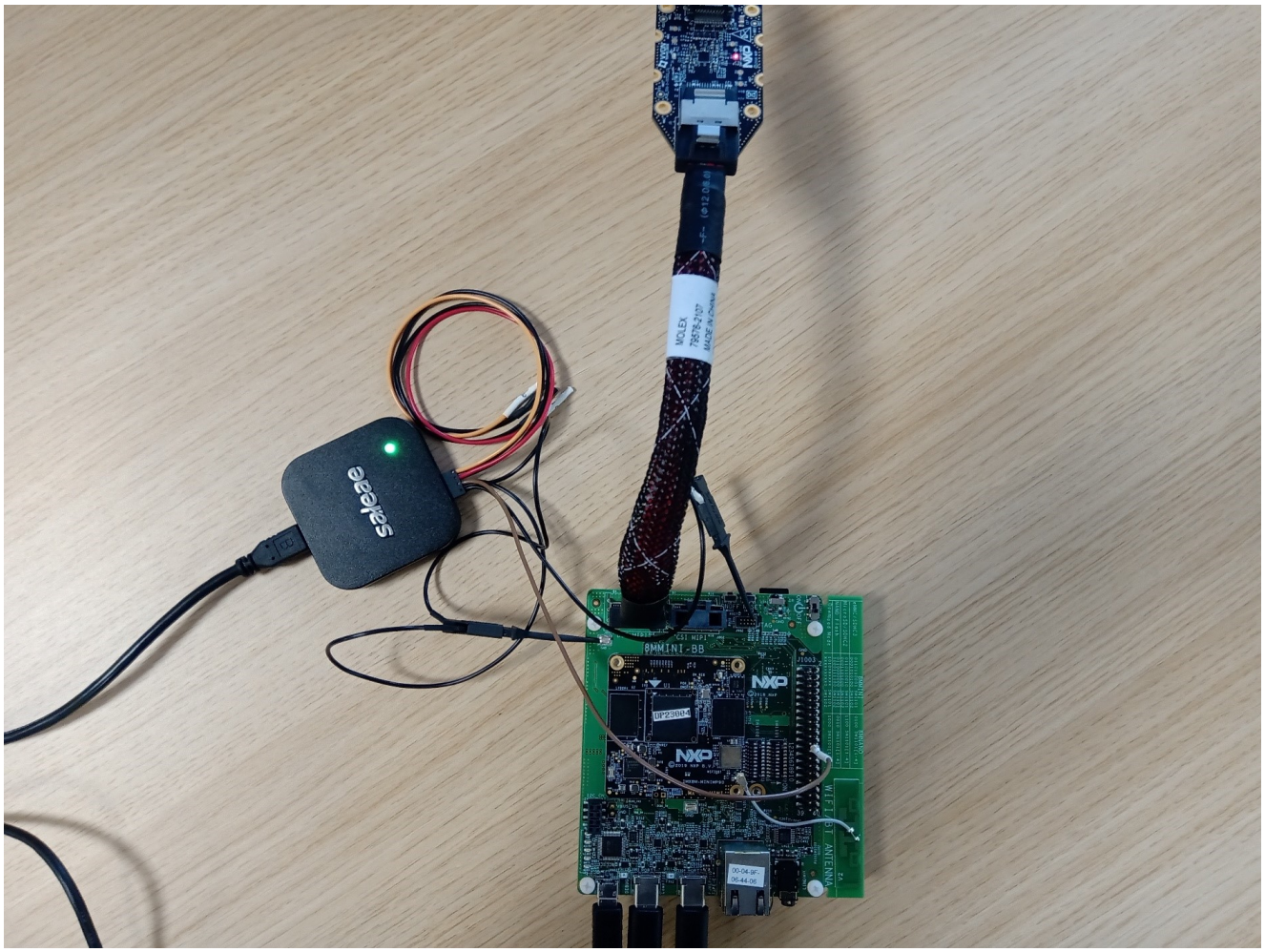


Figure 14. Measurement setup on the i.MX 8MM EVK board

After checking that both probes are referenced to the board's ground, start the logic analyzer software, where you can set up the probe parameters and the time frame.

Configure the board to start in the internal development mode, boot from the desired storage space, and then power it on. After the `psplash` screen disappears, press the reset button on the board and start the recording on the logic analyzer software.

You should see a rising edge on the nRST pin, followed by three pulses on the chosen pin. Stop the recording and place the measurement flags to find out the time for each phase. The boot time of the board in this configuration results from the sum of the elapsed time for each stage.

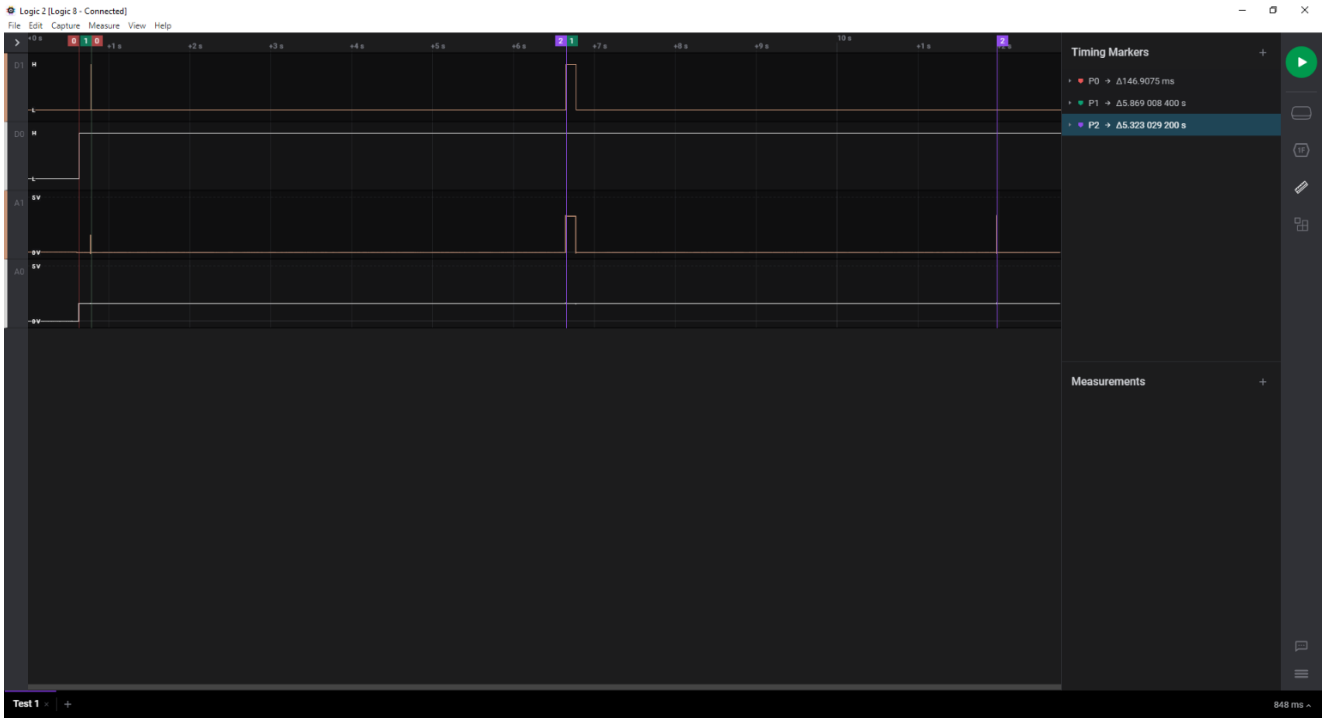


Figure 15. Measurement setup on the i.MX 8MM EVK board

3.4 i.MX 8M Nano

Choosing the pins

The chosen pin is the 24th pin on the J1003 expansion connector on the specified board. In the schematics for the baseboard, the pin is used for the ECSPI 2 peripheral with the ECSPI2_SS0 function. Searching the specified signal in the reference manual for the associated pin returns an alternate function of GPIO5_IO[13].

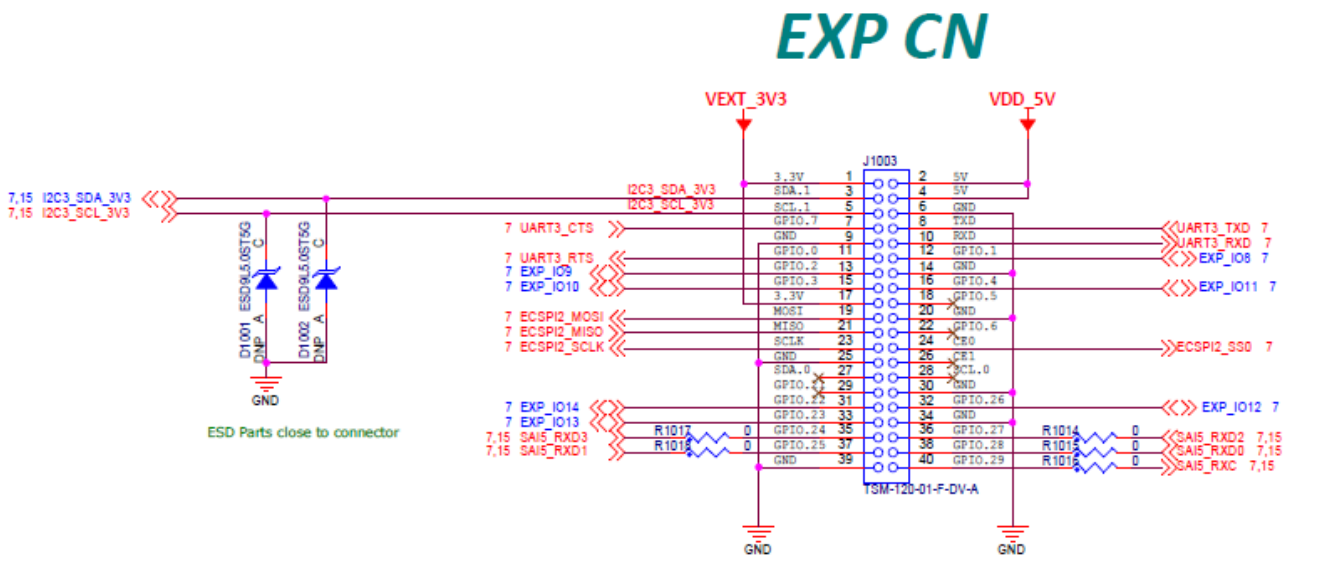
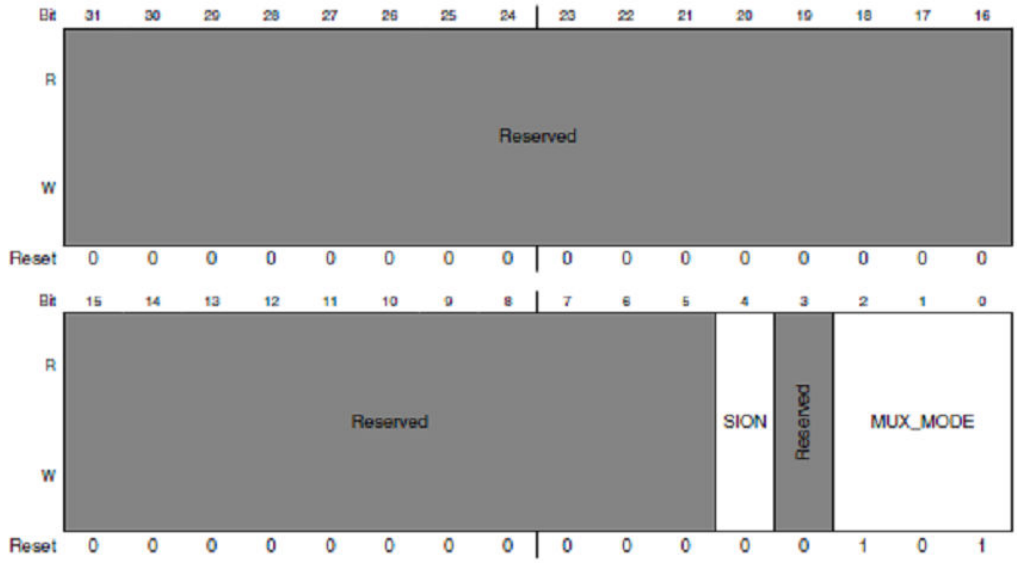


Figure 16. Schematic view of the expansion connector for the i.MX 8MN EVK board

8.2.5.104 Pad Mux Register (IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0)

Address: 3033_0000h base + 210h offset = 3033_0210h



IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0 field descriptions

Field	Description
31–5 -	This field is reserved. Reserved
4 SION	Software Input On Field. Force the selected mux mode input path no matter of MUX_MODE functionality. 1 **ENABLED** — Force input path of pad ECSPI2_SS0 0 **DISABLED** — Input Path is determined by functionality of the selected mux mode (regular).

IOMUXC_SW_MUX_CTL_PAD_ECSPI2_SS0 field descriptions (continued)

Field	Description
3 -	This field is reserved. Reserved
MUX_MODE	MUX Mode Select Field. Select 1 of 4 iomux modes to be used for pad: ECSPI2_SS0. **NOTE:** Pad ECSPI2_SS0 is involved in Daisy Chain. 000 **ALT0** — Select signal ECSPI2_SS0 - Configure register IOMUXC_ECSPI2_SS0_SELECT_INPUT for mode ALT0. 001 **ALT1** — Select signal UART4_RTS_B - Configure register IOMUXC_UART4_RTS_B_SELECT_INPUT for mode ALT1. 010 **ALT2** — Select signal I2C4_SDA - Configure register IOMUXC_I2C4_SDA_SELECT_INPUT for mode ALT2. 101 **ALT5** — Select signal GPIO5_IO13

Figure 17. Associated mux control register for the chosen pad (ECSPI2_SS0)

Updating the U-Boot device tree

Issue the following command to get the necessary files:

```
$ bitbake -f -c unpack imx-boot
```

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work/imx8mnevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mn-pinfunc.h file. The use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MN_IOMUXC_ECSPI2_SS0_ECSPI2_SS0              0x0210 0x0478 0x0570 0x0 0x0
#define MX8MN_IOMUXC_ECSPI2_SS0_UART4_DCE_RTS_B         0x0210 0x0478 0x0508 0x1 0x1
#define MX8MN_IOMUXC_ECSPI2_SS0_UART4_DTE_CTS_B         0x0210 0x0478 0x0000 0x1 0x0
#define MX8MN_IOMUXC_ECSPI2_SS0_I2C4_SDA                0x0210 0x0478 0x058C 0x2 0x5
#define MX8MN_IOMUXC_ECSPI2_SS0_GPIO5_IO13              0x0210 0x0478 0x0000 0x5 0x0
```

The associated DTS file for the i.MX 8MN EVK board (<yocto_build_dir>/tmp/work/imx8mnevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8mn-evk.dts) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin muxing into iomuxc:

```
pinctrl_hog_1: hoggrp-1 {
fsl,pins = <
MX8MN_IOMUXC_ECSPI2_SS0_GPIO5_IO13 0x16
>;
};
```

The 0x16 configuration is based on the following pad settings in the IOMUXC_SW_PAD_CTL_PAD_ECSPI2_SS0 register:

- Drive Strength Field: X6
- Slew Rate Field: Fast
- Open Drain Enable field: Disabled
- Control IO ports PS: Pull-down resistors
- Hysteresis Enable Field: CMOS
- Pull Resistors Enable Field: Pull Disabled

Updating the Linux device tree

To update the device tree, find out the macro associated to the desired functionality. This information is in the <yocto_build_dir>/tmp/work-shared/imx8mnevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mn-pinfunc.h file. The use of this macro is not enough for an adequate pin mux setup, because it requires a sixth value, which represents the pad configuration.

```
#define MX8MN_IOMUXC_ECSPI2_SS0_ECSPI2_SS0                0x210 0x478 0x570 0x0 0x0
#define MX8MN_IOMUXC_ECSPI2_SS0_UART4_DCE_RTS_B           0x210 0x478 0x508 0x1 0x1
#define MX8MN_IOMUXC_ECSPI2_SS0_UART4_DTE_CTS_B           0x210 0x478 0x000 0x1 0x0
#define MX8MN_IOMUXC_ECSPI2_SS0_I2C4_SDA                  0x210 0x478 0x58C 0x2 0x5
#define MX8MN_IOMUXC_ECSPI2_SS0_GPIO5_IO13                0x210 0x478 0x000 0x5 0x0
```

The associated DTS file for the i.MX 8MN EVK (<yocto_build_dir>/tmp/work-shared/imx8mnevk/kernel_source/arch/arm64/boot/dts/freescale/imx8mn-evk.dts) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin muxing into iomuxc. If the chosen pad has another pin mux configuration, the respective pin muxing must be replaced with the following to successfully generate the pulse in Linux:

```
pinctrl_hog: hoggrp {
fsl,pins = <
MX8MN_IOMUXC_ECSPI2_SS0_GPIO5_IO13 0x16
>;
};
```

Adding the first pulse generator in <yocto_build_dir>/tmp/work/imx8mnevk-poky-linux/u-boot-imx/<specified_git_folder>/git/board/freescale/imx8mn_evk/spl.c

Firstly, declare a macro containing the pair of GPIO group and GPIO pin in the file:

```
#define TIMED_GPIO IMX_GPIO_NR(5, 13)
```

There must be a macro for using a pad as a GPIO. In this case, it already exists.

```
#define USDHC_GPIO_PAD_CTRL (PAD_CTL_HYS | PAD_CTL_DSE1)
```

Add the pulse generation code into the `board_init_f` function, after the BSS clearing part:

```
void board_init_f(ulong dummy) {
 int ret;
 /* Clear the BSS. */
 memset(__bss_start, 0, __bss_end - __bss_start);
 gpio_request(TIMED_GPIO, "timed_gpio");
 gpio_direction_output(TIMED_GPIO, 1);
 gpio_direction_output(TIMED_GPIO, 0);
 arch_cpu_init();
```

Adding the second pulse generator in <yocto_build_dir>/tmp/work/imx8mnevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8mn_evk.h

Add the commands to toggle the pin into the environment variables for the bootloader, at the load image property. Set the pin before loading the image and reset it afterwards:

```
"loadimage=gpio set GPIO5_13;fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image};gpio clear
GPIO5_13\0" \
```

To automate the process, create a patch that contains the U-Boot modifications in the `spl.c` , `imx8mn-evk.dts` and `imx8mn_evk.h` files. Add the details after the last commands, describing the nature of the patch:

```
$ git add board/freescale/imx8mn_evk/spl.c
$ git add arch/arm/dts/imx8mn-evk.dts
$ git add include/configs/imx8mn_evk.h
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx`. Add the name of the patch into the source location identifier in the Yocto recipe for U-Boot (`<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2020.04.bb`):

```
SRC_URI = " ...\
file://<patch_name>.patch \
 "
```

To check that the patch works after the modifications, apply the following commands, after which the files should contain the following changes:

```
$ bitbake -f -c clean u-boot-imx
$ bitbake u-boot-imx
$ bitbake imx-boot
```

Adding the third pulse generator

For this stage, modify the Yocto recipe extension for `psplash` so that it can fetch the additional `libgpiod` library to toggle the GPIO (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`):

```
DEPENDS = "libgpiod"
RDEPENDS_${PN} = "libgpiod"
RDEPENDS_${PN}-dev = "libgpiod"
```

Issue the following command to get the necessary `psplash` files:

```
$ bitbake -f -c unpack psplash
```

After fetching the necessary files, modify the makefile (`<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/Makefile.am`) to include the `lgpiod`:

```
AM_CFLAGS = $(GCC_FLAGS) -D_GNU_SOURCE -lgpiod
GCC_FLAGS := $(GCC_FLAGS) -Lusr/lib -Iusr/include -lgpiod
LD_FLAGS = $(LD_FLAGS) -lgpiod
```

Modify the source code (`<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/psplash.c`). The first step is to include the library:

```
#include "gpiod.h"
```

Declare some additional variables in the main function to select the bank and the line for the GPIO. The banks are zero-indexed, so the number of the GPIO bank must be decremented by one.

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int gpio_line = 13;
char dev[] = "/dev/gpiochip4";
```

The program should acquire control of the pin, toggle it, and release it before sending the first command to the frame buffer (clearing the background):

```
chip = gpiod_chip_open(dev); if (!chip) return -1;
  line = gpiod_chip_get_line(chip, gpio_line); if (!line) {
  gpiod_chip_close(chip); return -1; }
 req = gpiod_line_request_output(line, "SIGNAL", 2); if (req) {
 gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 1) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 1);
  gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 0) != 0){
  printf("Impossible to change line %d value to %d \n", gpio_line, 0);
  gpiod_chip_close(chip); return -1; }
 gpiod_line_release(line);
 gpiod_chip_close(chip);
```

After applying all the modifications, build the `psplash` and the image using the following commands:

```
$ bitbake psplash
$ bitbake imx-image-core
```

To automate this process, create a patch that contains the modifications in the `Makefile.am` and `psplash.c` files. Add the details after the last commands, describing the nature of the patch:

```
$ git add Makefile.am
$ git add psplash.c
```

```
$ git commit -s
$ git format-patch HEAD~1
```

The resulting patch is then copied to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-core/psplash/files`. The name of the patch is then added in the source location identifier in the Yocto Recipe extension for psplash (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`).

```
SRC_URI += " \
            file://psplash-start.service \
            file://psplash-basic.service \
            file://psplash-network.service \
            file://psplash-quit.service \
       file://<patch_name>.patch \
            "
```

To check that the patch works after the modifications, apply the following commands, after which the `psplash.c` and `Makefile.am` should contain the following changes:

```
$ bitbake -f -c clean psplash
$ bitbake psplash
$ bitbake imx-image-core
```

Measuring the total time with the logic analyzer

This part starts by setting up all the preliminary connections, such as the power supply, debug port, download port, and MIPI-DSI to HDMI connections. If the HDMI display is not connected and functional, the `psplash` pulse generation does not work, because there are no frame buffers.

Both the bootloader and Linux image must be built using `bitbake`. After that, write them to the board using the UUU program, in which you can choose between writing to the on-board eMMC or the SD card.

Set up the measuring stand by connecting the logic analyzer to the board. The nRST signal is used as a starting reference and it is on the JTAG connector at pin 10.

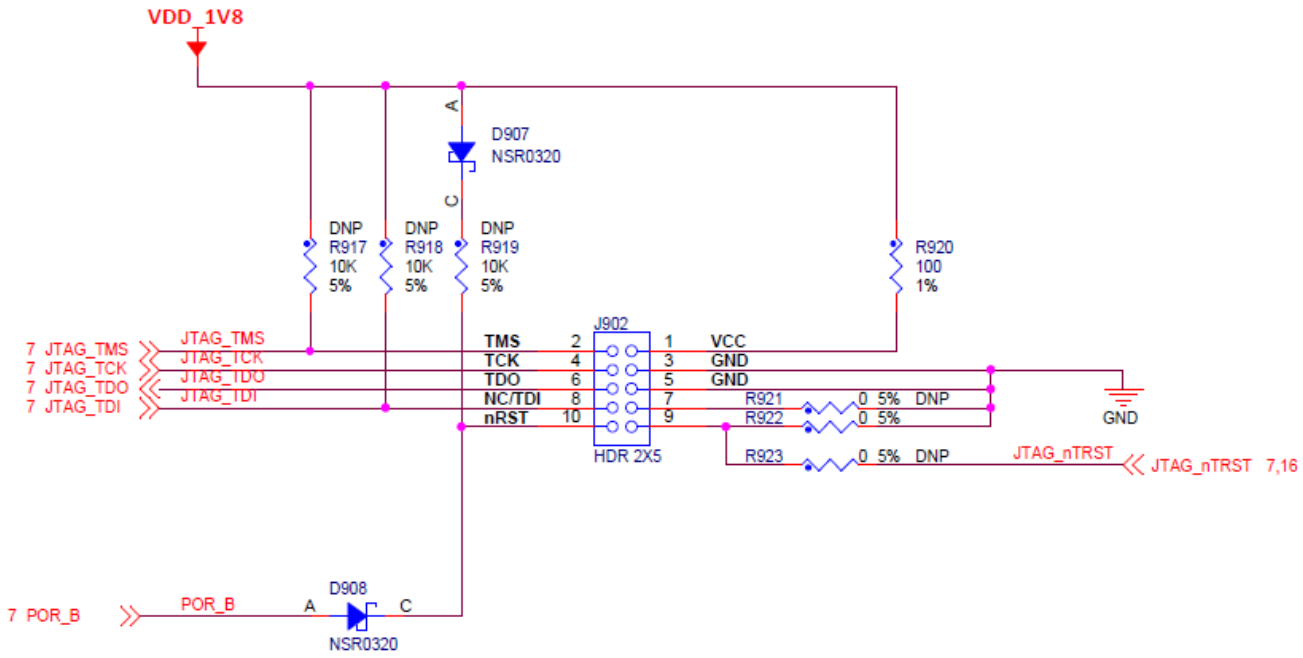


Figure 18. Schematic view of the JTAG connector available on the i.MX 8MN EVK board

To capture the rising edges of the chosen pin, connect the second probe of the analyzer to the 24th pin on the J1003 expansion connector.

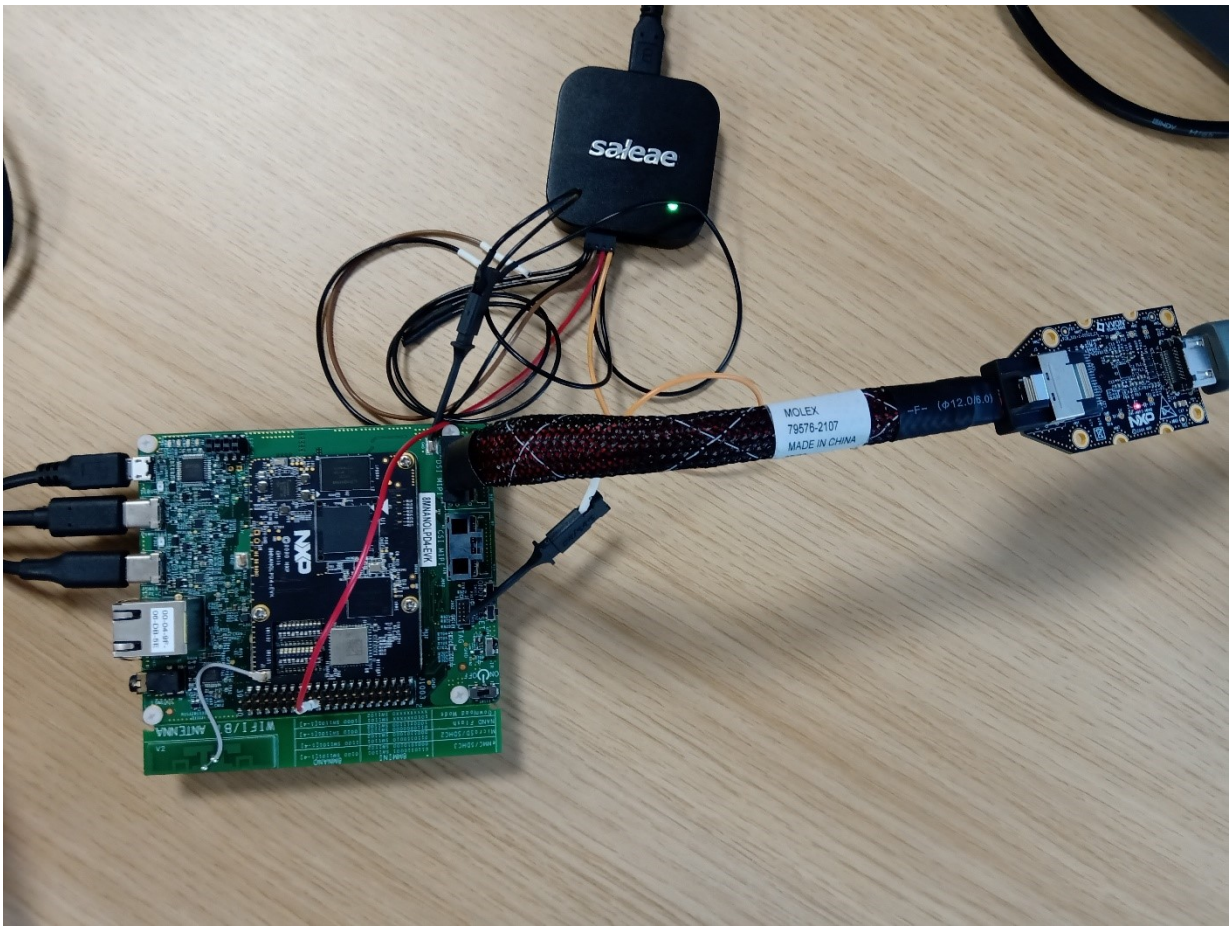


Figure 19. Measurement setup on the i.MX 8MN EVK board

After checking that both probes are referenced to the board's ground, start the logic analyzer software, where you can set the probe parameters and the time frame.

Configure the board to start in the internal development mode, boot from the desired storage space, and then power it on. After the `psplash` screen disappears, press the reset button on the board and start the recording on the logic analyzer software.

You should see a rising edge on the nRST pin, followed by three pulses on the chosen pin. Stop the recording and place the measurement flags to find out the time for each phase. The boot time of the board in this configuration results from the sum of the elapsed time for each stage.

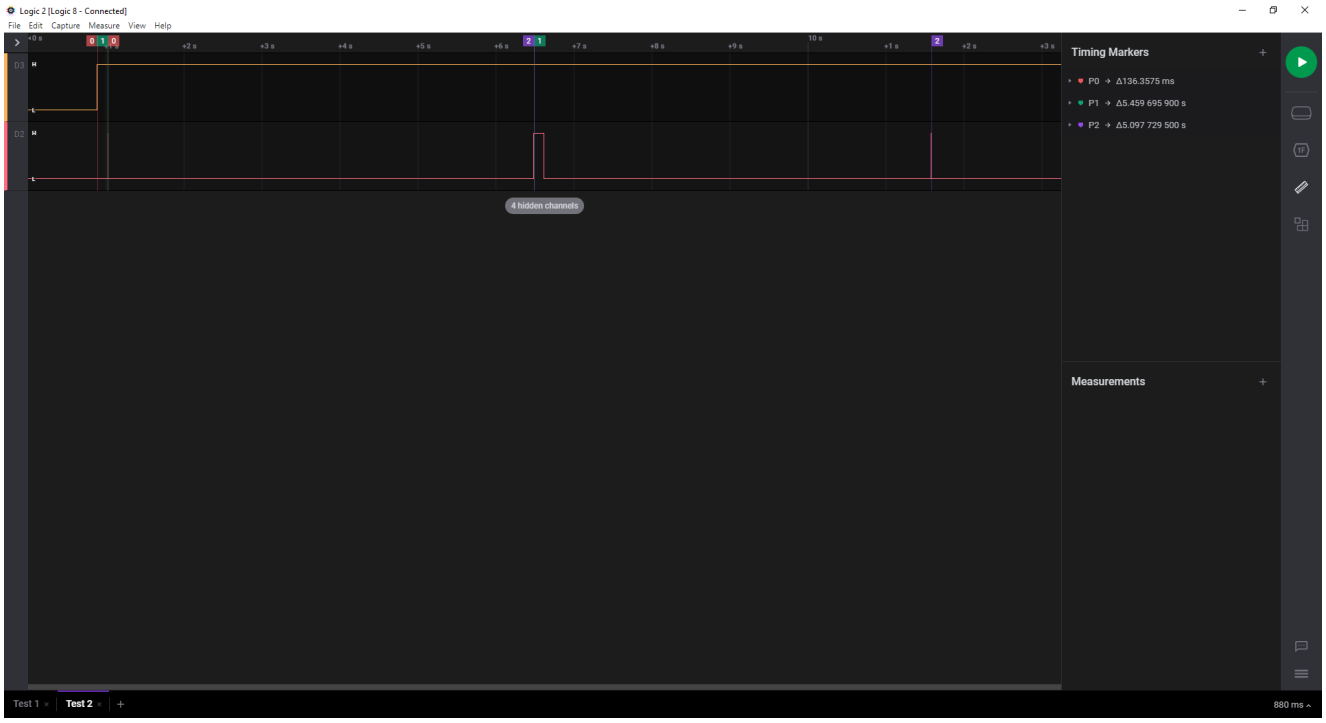


Figure 20. Measurements on the boot time for i.MX 8MN EVK board

3.5 i.MX 8ULP

Choosing the pins

The chosen pin is the 1st pin on the J20 Arduino headers on the specified board. In the schematics for the baseboard, the pin is used for the LPI2C 6 peripheral with the `LPI2C6_SCL` function. Searching the specified signal in the reference manual for the associated pin returns an alternate function of PTF0.

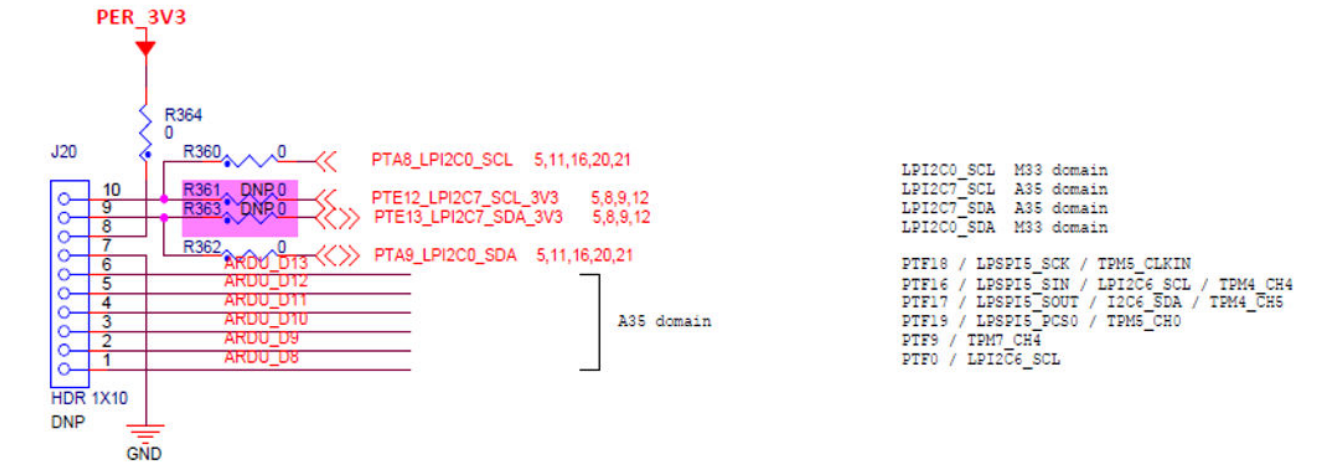


Figure 21. Schematic view of the Arduino headers for the i.MX 8ULP EVK board

Function

IOMUX PCR{DOMAIN}_{PADNAME}, Pad Control Register implements configuration fields for an specific {PADNAME}.

Diagram

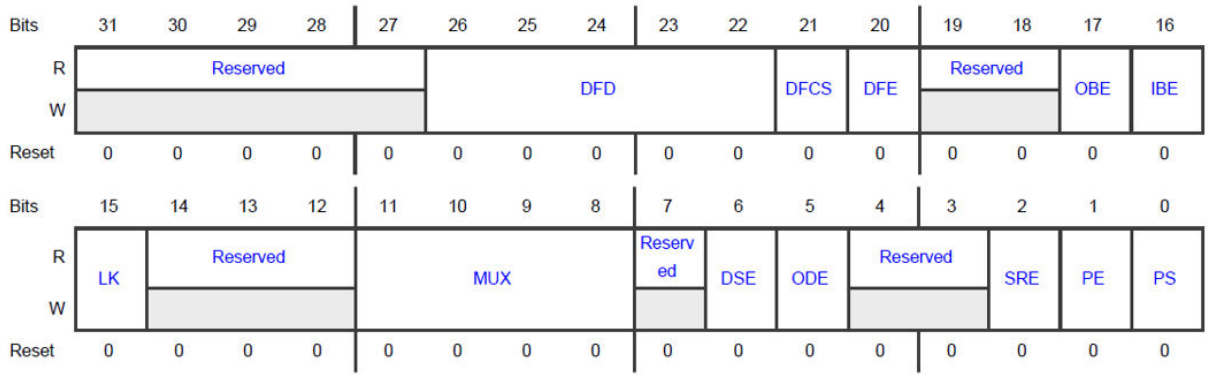


Figure 22. Pad control register for the chosen pad (PTF0)

Updating the U-Boot device tree

Use the following command to get the necessary files:

```
$ bitbake -f -c unpack imx-boot
```

To update the device tree, find out the macro associated to the desired functionality. This information is

in the `<yocto_build_dir>/tmp/work/imx8ulpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8ulp-pinfunc.h` file. The use of this macro is not enough for an adequate pin MUX setup, because it requires a 6th value, which represents the pad configuration.

```
#define MX8ULP_PAD_PTF0__PTF0 0x0100 0x0000 0x1 0x0
#define MX8ULP_PAD_PTF0__FXIO1_D0 0x0100 0x083C 0x2 0x2
#define MX8ULP_PAD_PTF0__LPUART6_CTS_B 0x0100 0x09CC 0x4 0x2
#define MX8ULP_PAD_PTF0__LPI2C6_SCL 0x0100 0x09B8 0x5 0x2
#define MX8ULP_PAD_PTF0__I2S7_RX_BCLK 0x0100 0x0B64 0x7 0x2
#define MX8ULP_PAD_PTF0__SDHC1_D1 0x0100 0x0A68 0x8 0x2
#define MX8ULP_PAD_PTF0__ENET0_RXD1 0x0100 0x0AFC 0x9 0x2
#define MX8ULP_PAD_PTF0__USB1_ID 0x0100 0x0ACC 0xa 0x3
#define MX8ULP_PAD_PTF0__EPDC0_SDOE 0x0100 0x0000 0xb 0x0
#define MX8ULP_PAD_PTF0__DPI0_D23 0x0100 0x0000 0xc 0x0
#define MX8ULP_PAD_PTF0__WUU1_P8 0x0100 0x0000 0xd 0x0
```

The associated DTS file for the i.MX 8ULP EVK board (`<yocto_build_dir>/tmp/work/imx8ulpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/arch/arm/dts/imx8ulp-evk.dts`) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin MUXing into `iomuxc`:

```
pinctrl_hog_1: hoggrp-1 {
fsl,pins = <
MX8ULP_PAD_PTF0__PTF0 0x42
>;
};
```

The `0x42` configuration is based on the following pad settings in the `IOMUX PCR1_PTF0` register:

- Drive strength enable: high drive strength
- Slew rate enable: standard slew rate
- Open drain enable: push-pull output
- Pull select: pull-down selected
- Pull-up enable: pull enabled

Updating the Linux device tree

To update the device tree, find the macro associated to the desired functionality. This information is in the `<yocto_build_dir>/tmp/work-shared/imx8ulpevk/kernel_source/arch/arm64/boot/dts/freescale/imx8ulp-pinfunc.h` file. The use of this macro is not enough for an adequate pin mux setup, because it requires a 6th value, which represents the pad configuration.

```
#define MX8ULP_PAD_PTF0__PTF0 0x0100 0x0000 0x1 0x0
#define MX8ULP_PAD_PTF0__FXIO1_D0 0x0100 0x083C 0x2 0x2
#define MX8ULP_PAD_PTF0__LPUART6_CTS_B 0x0100 0x09CC 0x4 0x2
#define MX8ULP_PAD_PTF0__LPI2C6_SCL 0x0100 0x09B8 0x5 0x2
#define MX8ULP_PAD_PTF0__I2S7_RX_BCLK 0x0100 0x0B64 0x7 0x2
#define MX8ULP_PAD_PTF0__SDHC1_D1 0x0100 0x0A68 0x8 0x2
#define MX8ULP_PAD_PTF0__ENET0_RXD1 0x0100 0x0AFC 0x9 0x2
#define MX8ULP_PAD_PTF0__USB1_ID 0x0100 0x0ACC 0xa 0x3
#define MX8ULP_PAD_PTF0__EPDC0_SDOE 0x0100 0x0000 0xb 0x0
#define MX8ULP_PAD_PTF0__DPI0_D23 0x0100 0x0000 0xc 0x0
#define MX8ULP_PAD_PTF0__WUU1_P8 0x0100 0x0000 0xd 0x0
```

The associated DTS file for the i.MX 8ULP EVK (`<yocto_build_dir>/tmp/work-shared/imx8ulpevk/kernel_source/arch/arm64/boot/dts/freescale/imx8ulp-evk.dts`) shows that the pin is not used, so there is nothing to disable.

To use the pin with the GPIO functionality, add the following pin MUXing into iomuxc. If the chosen pad has another pin MUX configuration, the respective pin MUXing must be replaced with the following to successfully generate the pulse in Linux:

```
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_hog>;
pinctrl_hog: hoggrp {
fsl,pins = <MX8ULP_PAD_PTF0__PTF0 0x42
>;
};
```

Adding the 1st pulse generator in <yocto_build_dir>/tmp/work/imx8ulpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/board/freescale/imx8ulp_evk/spl.c

Add the pulse generation code into the `board_init_f` function, after the BSS clearing part:

```
void board_init_f(ulong dummy)
{
 /* Clear the BSS. */
 memset(__bss_start, 0, __bss_end - __bss_start);
 __raw_writel(0x142, 0x298c0100);
 __raw_writel(0xc0000000, 0x2980007c);
 int val;
 val = __raw_readl(0x2d010044);
 val |= 0x1;
 __raw_writel(val, 0x2d010044);
 val = __raw_readl(0x2d010054);
 val |= 0x1;
 __raw_writel(val, 0x2d010054);
 val = __raw_readl(0x2d010048);
 val |= 0x1;
```

```
__raw_writel(val, 0x2d010048);
val = __raw_readl(0x2d010054);
val |= 0x1;
__raw_writel(val, 0x2d010054);
timer_init();
arch_cpu_init();
```

Adding the 2nd pulse generator in <yocto_build_dir>/tmp/work/imx8ulpevk-poky-linux/u-boot-imx/<specified_git_folder>/git/include/configs/imx8ulp_evk.h

Add the commands to toggle the pin into the environment variables for the bootloader, at the load-image property. Set the pin before loading the image and reset it afterwards:

```
“loadimage=gpio set GPIO3_0;fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${image};gpio clear
GPIO3_0\0” \
```

To automate the process, create a patch that contains the U-Boot modifications in the `spl.c`, `imx8ulp-evk.dts`, and `imx8ulp_evk.h` files. Add the details after the last commands, describing the nature of the patch:

```
$ git add board/freescale/imx8ulp_evk/spl.c
$ git add arch/arm/dts/imx8ulp-evk.dts
$ git add include/configs/imx8ulp_evk.h
$ git commit -s
$ git format-patch HEAD~1
```

Copy the resulting patch to the following location: `<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx`. Add the name of the patch into the source location identifier in the Yocto recipe for U-Boot (`<yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-bsp/u-boot/u-boot-imx_2021.04.bb`):

```
SRC_URI = " ...\
file://<patch_name>.patch \
"
To check that the patch works after the modifications, apply the following commands, after which the
files should contain the following changes:
$ bitbake -f -c clean u-boot-imx
$ bitbake u-boot-imx
$ bitbake imx-boot
```

Adding the 3rd pulse generator

For this stage, modify the Yocto recipe extension for `psplash` so that it can fetch the additional `libgpiod` library to toggle the GPIO (`<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend`):

```
DEPENDS = "libgpiod"
RDEPENDS_${PN} = "libgpiod"
RDEPENDS_${PN}-dev = "libgpiod"
```

Use the following command to get the necessary `psplash` files:

```
$ bitbake -f -c unpack psplash
```

After fetching the necessary files, modify the make file (`<yocto_build_dir>/tmp/work/cortexa35-poky-linux /psplash/<specified_git_folder>/git/Makefile.am`) to include `lgpiod`:

```
AM_CFLAGS = $(GCC_FLAGS) $(EXTRA_GCC_FLAGS) -D_GNU_SOURCE -lgpiod -DFONT_HEADER=\"$(FONT_NAME)-
font.h\" -DFONT_DEF=$(FONT_NAME)_font
GCC_FLAGS := $(GCC_FLAGS) -Lusr/lib -Iusr/include -lgpiod
LD_FLAGS = $(LD_FLAGS) -lgpiod
```

Modify the source code (<yocto_build_dir>/tmp/work/aarch64-poky-linux/psplash/<specified_git_folder>/git/psplash.c). The first step is to include the library:

```
#include "gpiod.h"
```

Declare some additional variables in the main function to select the bank and the line for the GPIO.

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int gpio_line = 0;
int req;
char dev[] = "/dev/gpiochip6";
```

The program should acquire control of the pin, toggle it, and release it before sending the 1st command to the frame buffer (clearing the background):

```
chip = gpiod_chip_open(dev); if (!chip) return -1;
line = gpiod_chip_get_line(chip, gpio_line); if (!line) {
gpiod_chip_close(chip); return -1; }
req = gpiod_line_request_output(line, "SIGNAL", 2); if (req) {
gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 1) != 0){
printf("Impossible to change line %d value to %d \n", gpio_line, 1);
gpiod_chip_close(chip); return -1; } if (gpiod_line_set_value(line, 0) != 0){
printf("Impossible to change line %d value to %d \n", gpio_line, 0);
gpiod_chip_close(chip); return -1; }
gpiod_line_release(line);
gpiod_chip_close(chip);
```

In BSP 5.10.72_2.2.0, the following modification is required to build imx-image-core:

In sources/meta-virtualization/recipes-containers/runc/runc-opencontainers_git.bb:

```
- git://github.com/opencontainers/runc;branch=master \
+ git://github.com/opencontainers/runc;branch=main \
```

After applying all the modifications, build the psplash and the image using the following commands:

```
$ bitbake psplash
$ bitbake imx-image-core
```

To automate this process, create a patch that contains the modifications in the Makefile.am and psplash.c files. Add the details after the last commands, describing the nature of the patch:

```
$ git add Makefile.am
$ git add psplash.c
$ git commit -s
$ git format-patch HEAD~1
```

The resulting patch is then copied to the following location: <yocto_dir>/imx-yocto-bsp/sources/meta-imx/meta-bsp/recipes-core/psplash/files. The name of the patch is then added in the source location identifier in the Yocto Recipe extension for psplash (<yocto_dir>/sources/meta-imx/meta-bsp/recipes-core/psplash/psplash_git.bbappend).

```
SRC_URI += " \
file://psplash-start.service \
file://psplash-basic.service \
file://psplash-network.service \
file://psplash-quit.service \
```

```
file://<patch_name>.patch \
"
```

To check that the patch works after the modifications, apply the following commands, after which the `psplash.c` and `Makefile.am` files should contain the following changes:

```
$ bitbake -f -c clean psplash
$ bitbake psplash
$ bitbake imx-image-core
```

Measuring the total time using the logic analyzer

This part starts by setting up all the preliminary connections, such as the power supply, debug port, download port, and HDMI connections. If the HDMI display is not connected and functional, the `psplash` pulse generation does not work, because there are no frame buffers.

Both the bootloader and the Linux image must be built using `bitbake`. After that, write them to the board using the UUU program.

Set up the measuring stand by connecting the logic analyzer to the board. In the i.MX 8ULP boot-time measurement, Agilent 16902A Logic Analyzer was used. The `JTAG_RESET` signal is used as a starting reference and it is on the JTAG connector at pin 10. Connect this pin to the first probe of the logic analyzer.

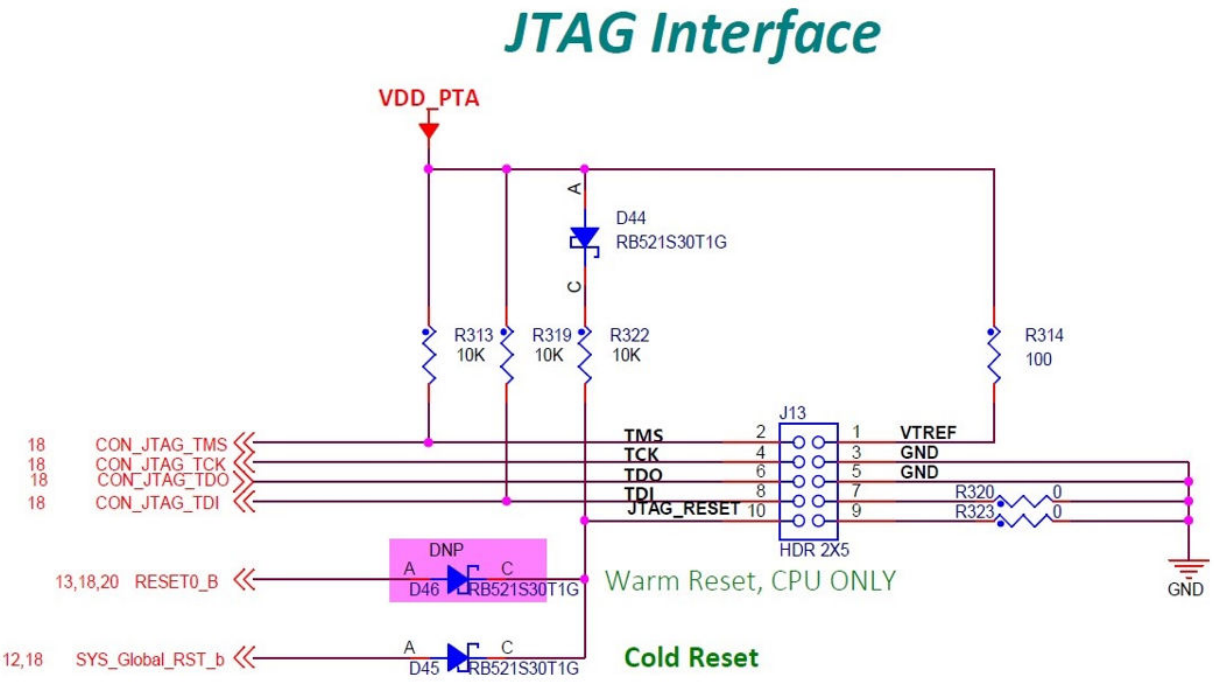


Figure 23. Schematic view of the JTAG connector available on the i.MX 8ULP EVK board

To capture the rising edges of the chosen pin, connect the second probe of the analyzer to the 1st pin on the J20 Arduino headers connector.

After checking that both probes are referenced to the board's ground, start the logic analyzer software, where you can set the probe parameters and the time frame.

Configure the board to boot from eMMC and power it on. After the `psplash` screen disappears, press the reset button on the board and start the recording on the logic analyzer software.

You should see a rising edge on the `JTAG_RESET` pin, followed by three pulses on the chosen pin. Stop the recording and find out the time for each phase. The boot time of the board in this configuration results from the sum of the elapsed time for each stage.

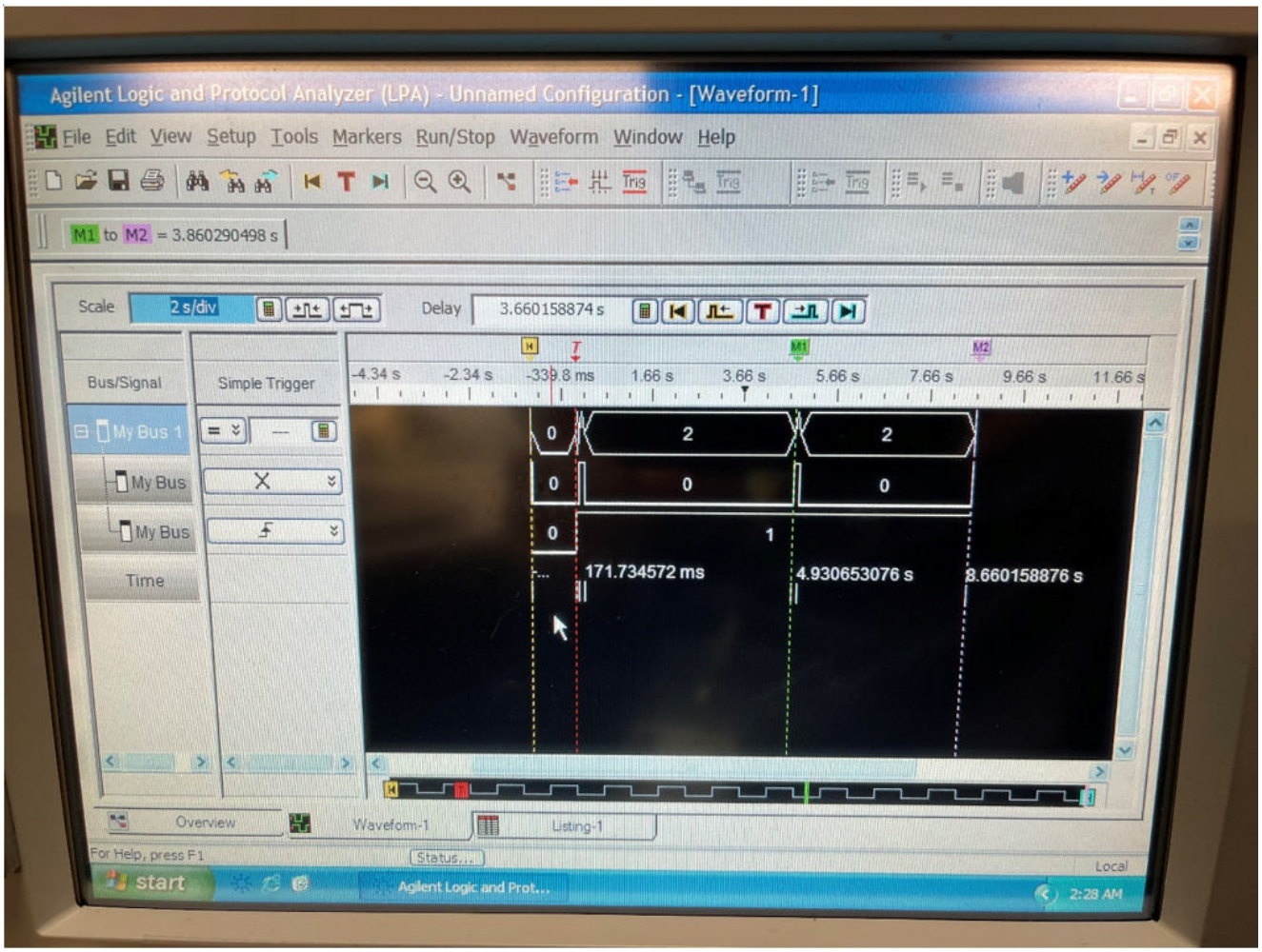


Figure 24. Measurements on the boot time for i.MX 8ULP EVK board

4 Further optimizations

4.1 No delay in kernel loading

Before the U-Boot Kernel image loads according to the autoboot sequence, there is a delay during which you can stop the process. This delay is defined in the u-boot environment variables as `bootdelay` and it may vary between the boards.

To achieve shorter boot times, configure the `bootdelay` to 0 after interrupting the autoboot sequence during the delay:

```
u-boot=> setenv bootdelay 0
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

After measuring the boot times, the total time shall decrease by the default `bootdelay` period.

4.2 Quiet operation

During the Linux start-up sequence, different messages are sent through the debug port, which has a limited speed (serial console). To achieve shorter boot times, suppress some of these messages using the `quiet` parameter in the `mmcargs` environment variable.

```
u-boot=> edit mmcargs
edit: setenv bootargs ${jh_clk} console=${console} root=${mmcroot} quiet
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

4.3 Enabling fast boot and using higher speeds for the storage

For the SD and eMMC storage, you can set higher speeds in the ROM stage of booting, depending on the available configuration. For eMMC, you can activate the fast boot mode. You can set these modifications using the boot fuses. In some cases, there is also the option to bypass the fuses using a predefined GPIO.

The following examples are applicable for the eMMC storage, because no notable decreases in boot time have been observed for the SD.

Case I : Changes can be applied externally by GPIO pins

The prerequisite for this mode is that the `BT_FUSE_SEL` fuse must be intact (0), so that the GPIO boot overrides can work. The board must also boot from the MicroSD card.

- **i.MX 8M Quad**

The GPIO pins that override the fuses are in the i.MX 8MQ reference manual.

Package pin	Direction on reset	eFuse
BOOT_MODE1	Input	Boot mode selection
BOOT_MODE0	Input	
SAI1_RXD0	Input	BOOT_CFG[0]
SAI1_RXD1	Input	BOOT_CFG[1]
SAI1_RXD2	Input	BOOT_CFG[2]
SAI1_RXD3	Input	BOOT_CFG[3]
SAI1_RXD4	Input	BOOT_CFG[4]
SAI1_RXD5	Input	BOOT_CFG[5]
SAI1_RXD6	Input	BOOT_CFG[6]
SAI1_RXD7	Input	BOOT_CFG[7]
SAI1_TXD0	Input	BOOT_CFG[8]
SAI1_TXD1	Input	BOOT_CFG[9]
SAI1_TXD2	Input	BOOT_CFG[10]
SAI1_TXD3	Input	BOOT_CFG[11]
SAI1_TXD4	Input	BOOT_CFG[12]
SAI1_TXD5	Input	BOOT_CFG[13]
SAI1_TXD6	Input	BOOT_CFG[14]
SAI1_TXD7	Input	BOOT_CFG[15]

Figure 25. GPIO overrides for the i.MX 8MQ EVK board

In the board configuration, the onboard eMMC supports the DDR mode, which can lead to shorter boot times. You can set the MMC speed to high and enable the fast boot mode.

		BOOT_CFG[15]	BOOT_CFG[14]	BOOT_CFG[13]	BOOT_CFG[12]	BOOT_CFG[11]	BOOT_CFG[10]	BOOT_CFG[9]	BOOT_CFG[8]
	0x470[15:8]	Infinit-Loop (Debug USE only) 0 - Disable 1 - Enable	001 - SD/eSD			Port Select: 00 - eSDHC1 01 - eSDHC2		Power Cycle Enable '0' - No power cycle '1' - Enabled via	SD Loopback Clock Source Sel (for SDR50 and SDR104 only) '0' - through SD pad '1' - direct
	0x470[15:8]		010 - MMC/eMMC						
	0x470[15:8]		011 - NAND			Pages In Block: 00 - 128 01 - 64 10 - 32 11 - 256		Nand_Row_address_bytes: 00 - 3 01 - 2 10 - 4 11 - 5	
	0x470[15:8]		100 - QSPI			QSPI Instance 0 - QuadSPI0 1 - Reserved	SDR SMP: "000" : Default "001-111"		
	0x470[15:8]		110 - SPI NOR			Port Select: 000 - eCSPI1 001 - eCSPI2			SPI Addressing: 0 - 3-bytes (24-bit) 1 - 2-bytes (16-bit)
	0x470[15:8]	Others - Reserved for future use							

		BOOT_CFG[7]	BOOT_CFG[6]	BOOT_CFG[5]	BOOT_CFG[4]	BOOT_CFG[3]	BOOT_CFG[2]	BOOT_CFG[1]	BOOT_CFG[0]
SD/eSD	0x470[7:0]	Fast Boot: 0 - Regular 1 - Fast Boot	Reserved	Reserved	Bus Width: 0 - 1-bit 1 - 4-bit	Speed 000 - Normal/SDR12 001 - High/SDR25 010 - SDR50 011 - SDR104 101 - Reserved for DDR50 Others - Reserved			Reserved
MMC/eMMC	0x470[7:0]		Bus Width: 000 - 1-bit 001 - 4-bit 010 - 8-bit 101 - 4-bit DDR (MMC 4.4) 110 - 8-bit DDR (MMC 4.4) Else - reserved.			Speed 00 - Normal 01 - High 10 - Reserved for HS200 11 - Reserved		USDHC1 IO VOLTAGE SELECTION 0 - 3.3V 1 - 1.8V	USDHC2 IO VOLTAGE SELECTION 0 - 3.3V 1 - 1.8V
NAND	0x470[7:0]	BT_TOGGLEMODE	BOOT_SEARCH_COUNT: 00 - 2 01 - 2 10 - 4 11 - 8		Toggle Mode 33MHz Preamble Delay, Read Latency: '000' - 16 GPMICLK cycles. '001' - 1 GPMICLK cycles. '010' - 2 GPMICLK cycles. '011' - 3 GPMICLK cycles. '100' - 4 GPMICLK cycles. '101' - 5 GPMICLK cycles. '110' - 6 GPMICLK cycles. '111' - 7 GPMICLK cycles. '1111'- 15 GPMICLK cycles.			Reserved
QSPI	0x470[7:0]	HSPHS: Half Speed Phase Selection 0 : select sampling at non-inverted clock 1: select sampling at inverted clock	HSDLY: Half Speed Delay selection 0 : one clock delay 1: two clock delay	FSPHS: Full Speed Phase Selection 0 : select sampling at non-inverted clock 1: select sampling at inverted clock	FSDLY: Full Speed Delay selection 0 : one clock delay 1: two clock delay	Reserved	Reserved	Reserved	Reserved
SPINOR	0x470[7:0]	CS select (SPI only): 00 - CS#0 (default) 01 - CS#1 10 - CS#2 11 - CS#3		Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Figure 26. Fuse settings for speed and fast boot for the i.MX 8MQ EVK board

To apply the modifications mentioned above, connect the associated pins to a 3v3 power source. The J801 I_2C header provides a solution to this problem on the 5th pin. The 3v3 power source is shared on a breadboard for the desired `BOOT_CFG` pins (7, 6, 5, 2).

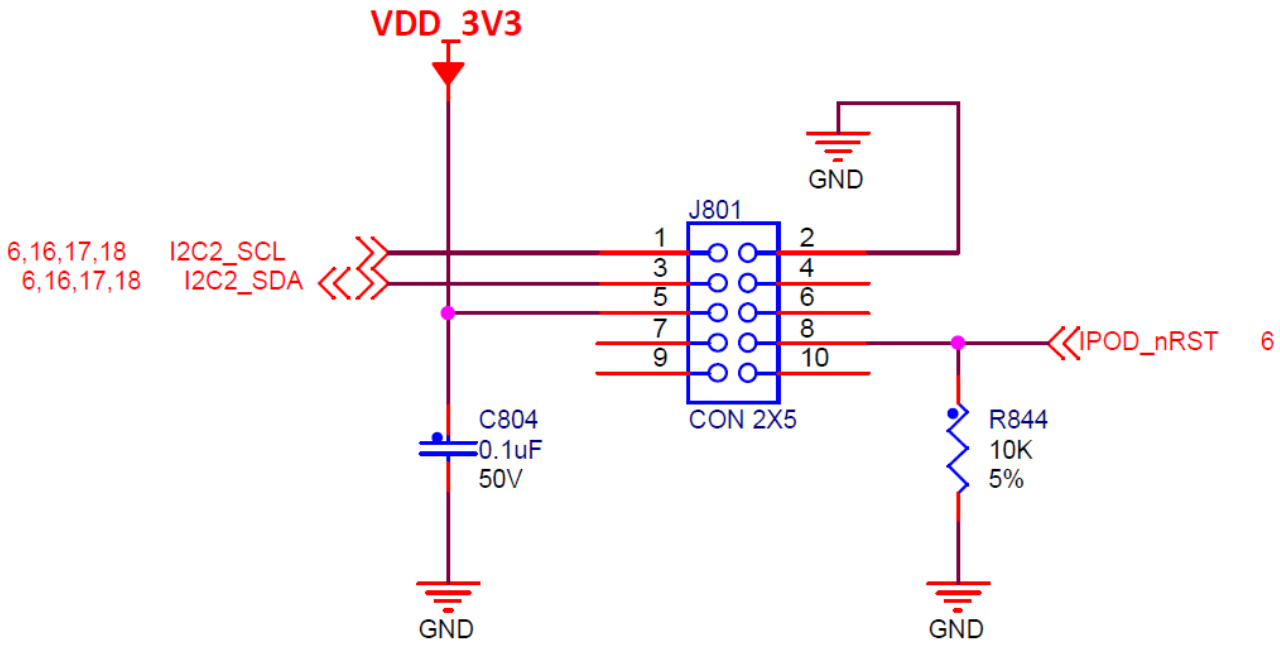


Figure 27. I2C header schematic for the i.MX 8MQ EVK board

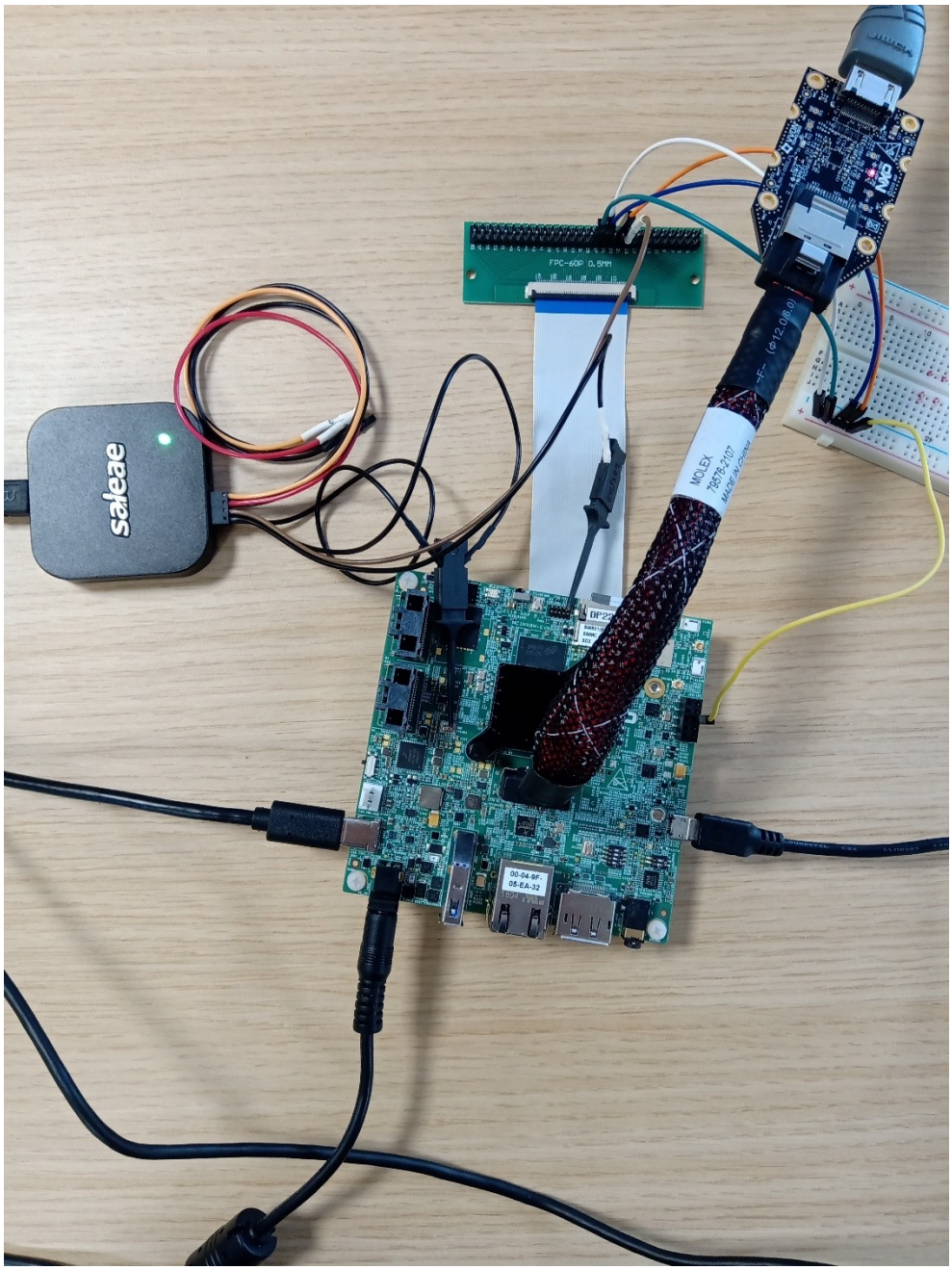


Figure 28. Board setup with breadboard for the i.MX 8MQ EVK board

NOTE

Before you apply any of the following commands, identify the eMMC device in both U-Boot and Linux, because the associated id/name is used in the next steps. The storage mapping for the evaluation board is as follows: eSDHC1 for the eMMC card and eSDHC2 for the MicroSD.

To use fast boot, rebuild the u-boot image using the `flash_evk_emmc_fastboot` target. You can do this by adding the following lines to the `local.conf` file in the `<yocto_build_dir>/conf` directory:

```
IMXBOOT_TARGETS_append = " flash_evk_emmc_fastboot"
```

After this, rebuild the image using the following `bitbake` commands:

```
$ bitbake -f -c clean imx-boot
$ bitbake imx-boot
```

An additional image that contains the `flash_evk_emmc_fastboot` settings should be available in the `<yocto_build_dir>/tmp/deploy/images/imx8mqevk/` directory.

The majority of the MMC settings can be modified in the U-Boot using the `mmc bootbus` command, which sets the `BOOT_BUS_WIDTH` field:

```
u-boot=> mmc bootbus <device_number> 2 1 2
```

Write the Image Vector Table (IVT) to a different offset, as per the reference manual (from 33 kB to 1 kB). You can do this in Linux by enabling the write rights to the bootpartition, writing the new U-Boot image, disabling the rights, and setting it as the bootpartition.

```
root@imx8mqevk:~# echo 0 > /sys/block/mmcblk<device_number>boot<partition_number>/force_ro
root@imx8mqek:~# dd if=<location to flash.bin>/flash.bin of=/dev/
mmcblk<device_number>boot<partition_number> seek=1 bs=1k conv=fsync
root@imx8mqek:~# echo 1 > /sys/block/mmcblk<device_number>boot<partition_number>/force_ro
root@imx8mqevk:~# mmc bootpart enable 1 0 /dev/mmcblk<device_number>
root@imx8mqevk:~# sync
```

- **i.MX 8M Mini**

The GPIO pins that override the fuses are in the i.MX 8MM reference manual.

Package pin	Direction on reset	eFuse
BOOT_MODE1	Input	Boot mode selection
BOOT_MODE0	Input	
SAI1_RXD0	Input	BOOT_CFG[0]
SAI1_RXD1	Input	BOOT_CFG[1]
SAI1_RXD2	Input	BOOT_CFG[2]
SAI1_RXD3	Input	BOOT_CFG[3]
SAI1_RXD4	Input	BOOT_CFG[4]
SAI1_RXD5	Input	BOOT_CFG[5]
SAI1_RXD6	Input	BOOT_CFG[6]
SAI1_RXD7	Input	BOOT_CFG[7]
SAI1_TXD0	Input	BOOT_CFG[8]
SAI1_TXD1	Input	BOOT_CFG[9]
SAI1_TXD2	Input	BOOT_CFG[10]
SAI1_TXD3	Input	BOOT_CFG[11]
SAI1_TXD4	Input	BOOT_CFG[12]

Package pin	Direction on reset	eFuse
SAI1_TXD5	Input	BOOT_CFG[13]
SAI1_TXD6	Input	BOOT_CFG[14]
SAI1_TXD7	Input	BOOT_CFG[15]

Figure 29. GPIO overrides for the i.MX 8MM EVK board

In the board configuration, the on-board eMMC supports the DDR mode, which can lead to shorter boot times. You can set the MMC speed to high and enable the fast boot mode.

		BOOT_CFG[7]	BOOT_CFG[6]	BOOT_CFG[5]	BOOT_CFG[4]	BOOT_CFG[3]	BOOT_CFG[2]	BOOT_CFG[1]	BOOT_CFG[0]
SD/eSD	0x470[7:0]	Fast Boot: 0 - Regular 1 - Fast Boot	Reserved	Reserved	Bus Width: 0 - 1-bit 1 - 4-bit	Speed 000 - Normal/SDR12 001 - High/SDR25 010 - SDR50 011 - SDR104 Others - Reserved			Reserved
MMC/eMMC	0x470[7:0]		Bus Width: 000 - 1-bit 001 - 4-bit 010 - 8-bit 101 - 4-bit DDR (MMC 4.4) 110 - 8-bit DDR (MMC 4.4) Else - reserved.			Speed 00 - Normal 01 - High Others - Reserved		USDHC IO VOLTAGE SELECTION For Normal Boot Mode 0 - 3.3V 1 - 1.8V	Reserved

Figure 30. Fuse settings for speed and fast boot for the i.MX 8MM EVK board

To apply the modifications mentioned above, connect the associated pins to a 3v3 power source. The evaluation board has two sets of ten switches through which the pins can be set or reset: SW1101 and SW1102.

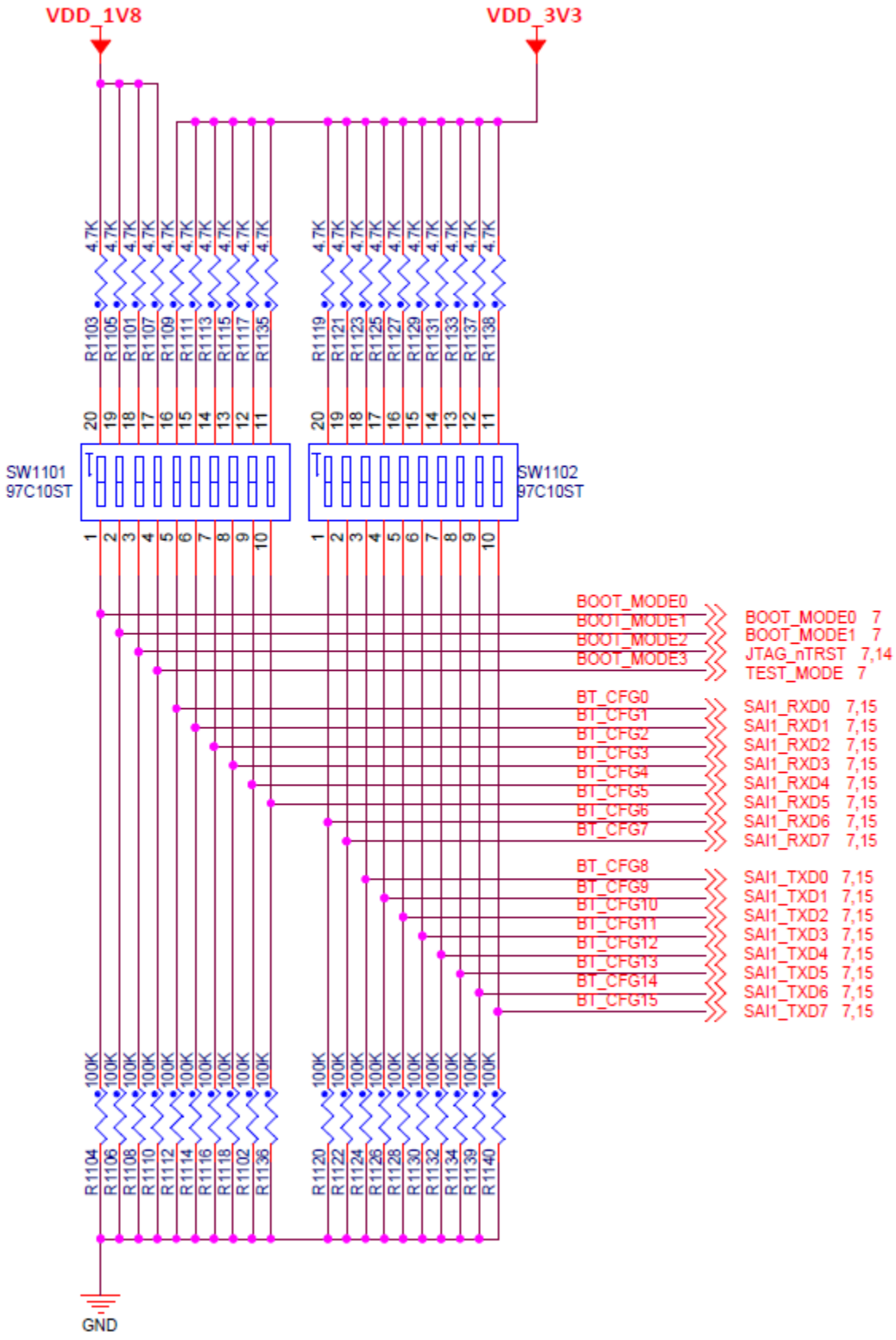


Figure 31. Schematic view of the GPIO override switches on the i.MX 8MM EVK board

NOTE

Before you apply any of the following commands, identify the eMMC device in both U-Boot and Linux, because the associated id/name is used in the next steps. The storage mapping for the development board is as follows: eSDHC2 for the MicroSD card and eSDHC3 for the eMMC.

To use the fast boot, rebuild the U-Boot image using the `flash_evk_emmc_fastboot` target. You can do this by adding the following lines to the `local.conf` file in the `<yocto_build_dir>/conf` directory:

```
IMXBOOT_TARGETS_append = " flash_evk_emmc_fastboot"
```

After this, rebuild the image using the following `bitbake` commands:

```
$ bitbake -f -c clean imx-boot
$ bitbake imx-boot
```

An additional image that contains the `flash_evk_emmc_fastboot` settings should be available in the `<yocto_build_dir>/tmp/deploy/images/imx8mmevk/` directory.

You can modify the majority of the MMC settings in the U-Boot using the `mmc bootbus` command, which sets the `BOOT_BUS_WIDTH` field:

```
u-boot=> mmc bootbus <device_number> 2 1 2
```

Write the Image Vector Table (IVT) to a different offset, as per the reference manual (from 33 kB to 1 kB). You can do this in Linux by enabling the write rights to the bootpartition, writing the new U-Boot image, disabling the rights, and setting it as the bootpartition.

```
root@imx8mmevk:~# echo 0 > /sys/block/mmcblk<device_number>boot<partition_number>/force_ro
root@imx8mmek:~# dd if=<location to flash.bin>/flash.bin of=/dev/
mmcblk<device_number>boot<partition_number> seek=1 bs=1k conv=fsync
root@imx8mmek:~# echo 1 > /sys/block/mmcblk<device_number>boot<partition_number>/force_ro
root@imx8mmevk:~# mmc bootpart enable 1 0 /dev/mmcblk<device_number>
root@imx8mmevk:~# sync
```

Case II : Changes can be applied by writing the boot fuses

- **i.MX 8M Plus**

In the board configuration, the onboard eMMC supports the DDR mode, which can lead to shorter boot times. You can set the MMC speed to high and the fast boot mode can be enabled.

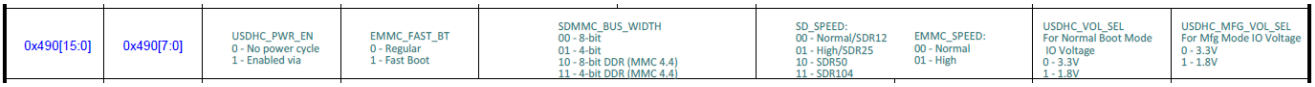


Figure 32. Fuse settings for speed and fast boot for the i.MX 8MP EVK board

NOTE

Before you apply any of the following commands, identify the eMMC device in both U-Boot and Linux, because the associated id/name is used in the next steps. The storage mapping for the evaluation board is as follows: eSDHC2 for the MicroSD card and eSDHC3 for the eMMC.

You can modify the majority of the MMC settings in the U-Boot using the `mmc bootbus` command, which sets the `BOOT_BUS_WIDTH` field:

```
u-boot=> mmc bootbus <device number> 2 1 2
```

NOTE

To apply the modifications mentioned above at the chip level, you must blow the fuses. This process is irreversible, so double check the values before writing the fuses according to the reference manual.

The fuses are written in the u-boot using the `fuse prog` command. This requires the bank and word in which the fuse resides, followed by a word containing the settings for the fuses. Find out the bank and word as follows:

(0x490-0x400)/0x10= 0x9 Hexadecimal = 9 Decimal

9/4 = 2 and the remainder is 1 (Bank = 2 and Word = 1)

The fuses that you must blow are as follows:

- 0x490[6] – Fast Boot
- 0x490[5] – 8-bit DDR
- 0x490[2] – EMMC Speed High

The resulting word to write for the desired configuration is 0x64:

```
u-boot=> fuse prog 2 1 0x64
```

To use the fast boot, there is no additional modification to the U-Boot image, because there is no different offset for the Image Vector Table (IVT).

- **i.MX 8M Nano**

The board configuration shows that the on-board eMMC supports the DDR mode, which can lead to shorter boot times. You can set the MMC speed to high and enable the fast boot mode.

Addr	7	6	5	4	3	2	1	0
0x490[7:0]	USDHC_PWR_EN 0 - No power cycle 1 - Enabled	EMMC_FAST_BT 0 - Regular 1 - Fast Boot	SDMMC_BUS_WIDTH 00 - 8-bit 01 - 4-bit 10 - 8-bit DDR (MMC 4.4) 11 - 4-bit DDR (MMC 4.4)		SD_SPEED: 00 - Normal/SDR12 01 - High/SDR25 10 - SDR50 11 - SDR104 EMMC_SPEED: 00 - Normal 01 - High		USDHC_VOL_SEL For Normal Boot Mode IO Voltage 0 - 3.3V 1 - 1.8V	USDHC_MFG_VOL_SEL For Mfg Mode IO Voltage 0 - 3.3V 1 - 1.8V

Figure 33. Fuse settings for speed and fast boot for the i.MX 8MN EVK board

NOTE

Before applying any of the following commands, you must identify the eMMC device in both U-Boot and Linux, because the associated id/name is used in the next steps. The storage mapping for the evaluation board is as follows: eSDHC2 for the MicroSD card and eSDHC3 for the eMMC.

You can modify the majority of the MMC settings in the U-Boot using the `mmc bootbus` command, which sets the `BOOT_BUS_WIDTH` field:

```
u-boot=> mmc bootbus <device number> 2 1 2
```

NOTE

To apply the modifications mentioned above at the chip level, you must blow the fuses. This process is irreversible, so double check the values before writing the fuses according to the reference manual.

The fuses are written in the U-Boot using the `fuse prog` command, which requires the bank and word in which the fuse resides, followed by a word containing the settings for the fuses. Find out the bank and word as follows:

(0x490-0x400)/0x10= 0x9 Hexadecimal = 9 Decimal

9/4 = 2 and the remainder is 1 (Bank = 2 and Word = 1)

The fuses that you must blow are as follows:

- 0x490[6] – Fast Boot
- 0x490[5] – 8-bit DDR
- 0x490[2] – EMMC Speed High

The resulting word to write for the desired configuration is 0x64:

```
u-boot=> fuse prog 2 1 0x64
```

To use the fast boot, there is no additional modification to the U-Boot image, because there is no different offset for the Image Vector Table (IVT).

5 References

- i.MX 8M Quad Applications Processor Reference Manual (document)
- i.MX 8M Plus Applications Processor Reference Manual (document)
- i.MX 8M Mini Applications Processor Reference Manual (document)
- i.MX 8M Nano Applications Processor Reference Manual (document)

6 Revision history

Table 1. Revision history

Revision number	Date	Substantive changes
1	29 April 2022	Added i.MX8 ULP
0	07 September 2021	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Cadence — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

FeliCa — is a trademark of Sony Corporation.

Freescale — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 29 April 2022
Document identifier: AN13369