

i.MX 6 Series Platform SDK

Version D

1 Introduction

The purposes of the Platform SDK for the i.MX 6 Series are to:

- Provide easily understood driver code that supports the primary features of a peripheral.
- Demonstrate key use cases of the chips.
- Provide unit tests for drivers to demonstrate their main features.
- Provide a simple environment for quick board validation and bring-up.
- Provide a source of entirely non-GPL example driver code.
- Includes consistent, documented register access macros for all hardware IP blocks.
- Uses the IOMUX Tool to generate mux configuration code.
- Provides register definitions for ARM RealView Debugger and Lauterbach TRACE32.

The result is an SDK that provides reusable system code and a large number of drivers with related unit tests. For board bring-up, each module can be individually tested one at a time rather than having to bring up most drives all at once. Based on a common initialization of the i.MX6 and the development board, a test for a specific peripheral is launched with the help of a serial console for human interaction.

Also included in the release package are a Firmware Guide (iMX6FG_RevD.pdf) and IOMUX Tool application (IOMux.exe).

2 Versions

	Version	Release Date	Description
SDK Software	D	29-Jun-2012	Release with complete driver set.
Firmware Guide	D	29-Jun-2012	Detailed driver documentation and use guide for chip peripherals.
IOMUX Tool	3.3.2.1	29-Jun-2012	IOMUX configuration tool Windows application

3 Directory structure

Description	Location
Configuration files used to enable features supported by a board. e.g., <code>mx6dq_sabre_lite.conf</code>	<code>./configs</code>
SDK documentation files. Doxygen output appears in an <code>html</code> child directory.	<code>./doc</code>
Non-driver shared components.	<code>./sdk/common</code>
Source code related to the ARM Cortex-A9 MPCore platform.	<code>./sdk/cortex_a9</code>
Common headers which are chip and board independent.	<code>./sdk/include</code>
Headers specific to the i.MX6 series.	<code>./sdk/include/mx6dq</code> <code>./sdk/include/mx6sdl</code>
Generated register headers.	<code>./sdk/include/mx6dq/registers</code> <code>./sdk/include/mx6sdl/registers</code>
Global library source file such main, system initialization.	<code>./sdk/init</code>
Source code specific to the i.MX6 series.	<code>./sdk/mx6dq/</code> <code>./sdk/mx6sdl/</code>
IOMUX Tool board design files and generated code directories.	<code>./sdk/mx6dq/<board>/</code> <code>./sdk/mx6sdl/<board>/</code>
Parent directory for all drivers, includes, and their unit tests and docs.	<code>./sdk/drivers/</code>
Miscellaneous utility components.	<code>./sdk/utility</code>
Build script and utilities.	<code>./tools</code>
ARM RealView Debugger BCD files	<code>./tools/rvd/registers</code>
Lauterbach TRACE32 peripheral files	<code>./tools/lauterbach</code>

Example for a driver:

Description	Location
TEMPMON driver source code.	<code>./sdk/drivers/tempmon/src/</code>
TEMPMON driver public includes.	<code>./sdk/drivers/tempmon/tempmon.h</code>
TEMPMON driver unit tests.	<code>./sdk/drivers/tempmon/test/</code>

4 System requirements

To build the Platform SDK, either a Linux host or Windows host with Cygwin installed may be used.

4.1 Working with Cygwin

Cygwin can be installed and obtained from here:

<http://www.cygwin.com/>

The installation procedure is documented on this site. Some standard packages are necessary, such as `make`, `bash`, and `perl`, but that list is not thorough.

The `cygpath` program is a utility that converts Windows native filenames to Cygwin POSIX-style pathnames and vice versa. It can be used when a Cygwin program needs to pass a file name to a native Windows program, or expects to get a file name from a native Windows program.

When using the CodeBench toolchain (see below), it is necessary to set this environment variable (sh syntax, csh will use a different syntax):

```
export CYGPATH=cygpath
```

Note: Version 3.81 of the `make` package in Cygwin might cause build failures due to a known problem with that version related to correctly handling Windows pathnames. To avoid this issue, it is recommended to verify that `make` version 3.82 or later is being used. 3.82 is the default version for new installations and upgrades of Cygwin as of this writing.

4.2 Installing the toolchain

The SDK is built using the Mentor Sourcery CodeBench Lite (previously called Sourcery G++ Lite) version of the arm-none-eabi GCC toolchain. The latest version as of this writing is 4.6.3, obtained here:

<https://sourcery.mentor.com/GNUToolchain/release2188>

Versions that are known to work with the SDK are as follows:

Package	gcc version	Date
Sourcery CodeBench Lite 2012.03-56	4.6.3	2012-06-11
Sourcery CodeBench Lite 2011.09-69	4.6.1	2011-12-19
Sourcery G++ Lite 2011.03-42	4.5.2	2011-05-02

The full list of CodeBench Lite releases is available on this page:

<https://sourcery.mentor.com/GNUToolchain/subscription3053?lite=arm&lite=ARM>

Download either the Windows or Linux installer package. The tarball packages do not include all executables.

Install the package under /opt or any other local folder, and make sure that the PATH environment variable allows accessing these executables used for the build process:

```
CC = arm-none-eabi-gcc
CXX = arm-none-eabi-g++
AS = arm-none-eabi-as
AR = arm-none-eabi-ar
LN = arm-none-eabi-ld
```

Example (sh syntax):

```
export PATH=$PATH:<toolchain_install_path>/bin
```

Otherwise, it is possible to redefine the above list in the file `./<sdk_install_path>/make.def` with the complete path to the executable.

4.3 Source code installation

The SDK source code package can be installed anywhere.

It is not recommended to use a Windows unzip program to extract the sources, as this may result in line endings being automatically converted. Instead, the equivalent command of Cygwin or Linux should be used.

Example:

```
tar zxvf mx6series_sdk_release.tar.gz
```

4.4 Build command

To build the SDK, use the `./tools/build_sdk` command.

Note: It is important to run `build_sdk` from the SDK root directory, not from within the `tools` directory.

The detailed usage that is printed for the `-help` option is as follows:

```
Usage:
  build_sdk [-t <target>] [-b <board>] [-v <rev>] [-r <test>] [-n] [-c] [-l]

Generates makefiles for the i.MX SDK project.

Options:
  -t, -target=<target>      Specify the target name. Optional, and the default is all.
  -b, -board=<brd>         Specify the board name. Optional, and the default is all.
  -v, -board_rev=<rev>     Specify the board revision. Optional, and the default is all.
```

```

-r, -test=<name>      Optional argument to select a single test, or ALL for all tests.
Defaults to ALL.
-c, -clean            Optional flag to force a clean build.
-n, -no-build         Don't actually run make.
-l, -list-builds      Optional flag to list target, board, board_rev combinations to be
built.

```

Short options take the same arguments as their respective long options.
The '=' is optional for long options; arguments can be specified as -arg=value or
-arg value. Short options require a space between the option and value.

Generates makefiles for the specified combination of options. If -clean was specified,
'make clean' will be run. Then a regular build will be executed, unless the -no-build
argument is present. Makefiles are generated only for packages that are part of the
selected board.

Valid target and board combinations:

```

target=mx6dq boards:
  sabre_ai      rev a, b
  evb           rev a (MX6QCPUDDR3)
  smart_device  rev a, b
  sabre_lite    rev a, b, c

target=mx6sdl boards:
  sabre_ai      rev a, b
  evb           rev a (MX6QCPUDDR3)
  smart_device  rev a, b

target=all
  builds all valid combinations of target, board, and board_rev
  [-r <test>] is a valid option.
  -clean       Only does clean builds.

```

Examples:

```

build_sdk -target mx6dq -board evb

  Build for the mx6dq evb board with default board rev a, and all tests.

build_sdk -t mx6dq -b sabre_ai -v a -test sdma -clean

  Build the mx6dq sabre_ai board with explicit board rev a, but only build the sdma test.
  Clean before building.

build_sdk -target=mx6dq -board=evb -test=gpmi -no-build

  Generate the makefiles for mx6dq evb and the gpmi test, but don't actually run make.

build_sdk -target=all -test vpu

  Clean build of vpu_test for all valid combinations of target, board, and board_rev.

```

Running build_sdk will automatically generate makefiles for every directory listed in the .conf
file (in the configs directory) associated with the select target and board. A root level
makefile will also be created. However, this makefile cannot be used to build the SDK. The
build_sdk script must always be used to start a build.

The set of boards and revisions supported by this SDK release are listed here:

Name	Description	Revisions
evb	Freescall validation board, labeled MX6QCPUDDR3	A only
sabre_ai	Automotive reference design	A, B
sabre_lite	General purpose reference design	A, B, C
smart_device	Smart device reference design	A, B

The listed tests below are not available for all boards. This is usually due to lack of components or connectors on the given board.

Available tests:

Module	Test Name	Board ¹			
		evb	sabre_ai	sabre_lite	smart_device
Audio, SSI and audiomux	audio	✓	✓	✓	✓
Camera	camera	-	-	-	✓
CAN	flexcan	-	✓	-	-
cpu_utility	cpu_get_cores	✓	✓	✓	✓
ENET for Ethernet	enet	✓	-	-	-
EPIT	epit	✓	✓	✓	✓
GIC, multicore startup	gic_multicore	✓	✓	✓	✓
GIC, software interrupts	gic_sgi	✓	✓	✓	✓
GPPI	gpmi	✓	-	-	-
GPT	gpt	✓	✓	✓	✓
GPU	gpu	✓	-	-	-
HDMI	hdmi	✓	✓	✓	✓
I2C	i2c	✓	✓	✓	✓
IPU for display	ipu	✓	✓	✓	✓
LDB for LVDS display	ipu	✓	✓	✓	✓
KEYPAD	kpp	✓	-	-	-
MIPI	mipi	✓	✓	-	-
OCOTP for e-fuses	ocotp	✓	✓	✓	✓
Low power modes	power_modes	✓	✓	✓	✓
PCIe	pcie	✓	-	-	-
PMU	pmu	✓	✓	✓	✓
PWM	pwm	✓	-	-	-
SATA for Hard Drive	sata	✓	✓	✓	✓
SDMA	sdma	✓	✓	✓	✓
SNVS RTC	snvs_rtc	✓	✓	✓	✓
SNVS SRTC	snvs_srtc	✓	✓	✓	✓
SPI	spi	✓	-	-	-
Timer	microseconds	✓	✓	✓	✓
TEMPMON	tempmon	✓	✓	✓	✓

UART	uart	✓	✓	✓	✓
USB	usb	-	✓	-	-
USDHC for SD/MMC	usdhc	✓	✓	✓	✓
Video decoder	vpu	✓	-	✓	✓
Watchdog	wdog	✓	✓	✓	✓
WEIM	weim	✓	✓	-	-
ALL	ALL ²	✓	✓	✓	✓

¹ '✓' indicates that all boards revisions are supported, otherwise the supported revision is specified.

² The option ALL includes all tests with the possibility to run, re-run, or skip some of tests.

Once the build is finished a completion message will be printed showing where the ELF output file is located. The build produces an ELF file, a binary image, and a map file. Output files can be found in:

```
./output/<cpu>/<board>_rev_<rev>/bin/
```

The output file name contains the targeted CPU, board, board revision, and the name of the chosen test.

Example output files in the bin directory:

```
mx6dq_sabre_ai_rev_a-vpu-sdk.elf
mx6dq_sabre_ai_rev_a-vpu-sdk.bin
mx6dq_sabre_ai_rev_a-vpu-sdk.map
```

Example build completion message:

```
*****
Build has completed. ELF file available at:
=> /home/creed/diag-sdk/output/mx6sdl/evb_rev_b/bin/mx6sdl_evb_rev_b-ALL-
sdk.elf

Build mx6sdl evb rev_b ALL passed!
```

Adding the option `-clean` to the `build_sdk` command line will remove the CPU output directory found under `./output` prior to building. Example:

```
./tools/build_sdk -t mx6dq -b sabre_ai -v a -test sdma -clean
```

5 How to run a test

5.1 Setup required

An i.MX 6Dual/6Quad or i.MX 6Solo/6DualLite board is necessary to run a test. Please refer to the appropriate board user guide to prepare this setup.

The user interaction and output information are available through the serial port UART4 connected to a host running terminal software such as TeraTerm, minicom, HyperTerminal, or a similar program.

The configuration for the terminal is common: 115200 bps 8N1 (8 data bits, no parity, 1 stop bit).

5.2 Download with a JTAG probe

The code can be downloaded and executed through any JTAG tool that supports the Cortex-A9. It is necessary to have a board initialization file that will initialize some of the clocks, the MMDC controller and its interface, as well as the DDR3.

The SDK package contains a debugger initialization script for the ARM RealView Debugger:

```
./tools/rvd/mx6dq_init.inc
```

Run this script in the debugger after opening a connection to the i.MX6 Series device. After the script finishes, DDR3 memory is available for use. At this point you can load the ELF output file from the build and run the test.

5.3 Download with the manufacturing tool

The manufacturing tool can be obtained from the Freescale i.MX Design tool web page:

http://www.freescale.com/webapp/sps/site/overview.jsp?code=IMX_DESIGN

Below is an example of profile that can be added in the `uc1.xml`:

```
<LIST name="DWLD_IN_SDP" desc="Download and execute a binary!">
  <CMD type="find" body="Recovery" timeout="180"/>
  <CMD type="boot" body="Recovery" file="mx6dq_sabre_ai_rev_a-vpu-
sdk.bin">Loading SDK image</CMD>
  <CMD type="jump">Jumping to SDK image.</CMD>
</LIST>
```

5.4 Using SD boot

The SDK binary image can be programmed on a SD card using 2 different methods, depending on the host OS being used.

Using a Windows host:

By using the `cfimager-imx.exe` provided with the release package under folder

```
./tools/windows.
```


Use this command:

```
cfimager-imx.exe -o 0x0 -f  
./output/mx6dq/evb_rev_a/bin/mx6dq_evb_rev_a-ALL-sdk.bin -d  
<your drive letter for the SD card>
```

For example, if the drive letter of the SD card reader is F:, type the following:

```
cfimager-imx.exe -o 0x0 -f  
./output/mx6dq/evb_rev_a/bin/mx6dq_evb_rev_a-ALL-sdk.bin -d f
```

There is also the option of formatting the card first before programming it, though be warned it will take more time for this operation to complete due to the card formatting. To invoke the card formatting, simply append the above command with “-a”, as provided in the following example (again, assuming the drive letter is “f”):

```
cfimager-imx.exe -o 0x0 -f ./output/mx6dq/evb  
rev_a/bin/mx6dq_evb_rev_a-ALL-sdk.bin -d f -a
```

Using a Linux host:

Use the command dd under Linux to format the SD card:

```
dd if=output/mx6dq/evb_rev_a/bin/mx6dq_evb_rev_a-ALL-sdk.bin  
of=/dev/sdx seek=2 skip=2 bs=512 && sync
```

/dev/sdx is the Linux device for your SD card.

The seek option allows to not change the first 1kB of the SD card where a partition table could reside. The skip option is to remove the unnecessary first 1kB from the binary as the seek option is used.

Once the SD card is programmed successfully with a bootable SDK image, follow below instructions on how to run it on the i.MX6DQ EVB board:

- Insert SD card with SDK binary into SD slot.
- Configure boot dip switches for boot from SD:
 - SW1: off-off-off-off-off-ON-off
 - SW2: off-off-off-ON-ON-off-off-off
 - SW5: all off
 - SW8: all off.
- Power on board, and follow on screen prompt commands on host' terminal window.

6 How to add a new driver and test

A new driver should typically be added in a new directory under the `./sdk/drivers/` directory. Public headers go directly into the driver directory root, the source code and private headers into `./sdk/drivers/<new_driver>/src`, and its unit test into `./sdk/new_driver/test`.

In the board config file, such as `mx6dq_evb_rev_a.conf`, the following lines must be added:

```
./src/sdk/<new_driver>/src
./src/sdk/<new_driver>/test
```

Everything contained in those directories will be compiled.

The main test file should contain a test function named: `int32_t new_driver_test(void)`. Hence, when building, the parameter “`– new_driver`” can simply be used.

When using the build ALL option, this test can also be added into the function `void ALL_test(void)` available in `./src/mx6dq/mx6dq.c`.

7 Firmware Guide

The i.MX 6 Series Firmware Guide contains detailed documentation of the peripherals supported by the SDK and the associated drivers. The document **IMX6FG_RevD.pdf** is located in the `doc` folder of release package.

8 Register definitions

Included with the SDK is a full set of register definition C language header files for all modules on the i.MX 6 Series chips. These headers are located under `sdk/include/<chip>/registers`. The macros defined in the header files are generated from the same source material as the register definition sections in the reference manuals and use the same names listed in those sections. In addition, the full documentation of each register and bitfield found in the reference manual is available in the header files as comments.

Further information about the header files can be found in the **Register header quick reference.pdf** file located in the `doc` directory.

To aid in debugging, the SDK includes complete register definitions for both the ARM RealView Debugger and Lauterbach TRACE32. These files are found under `tools/rvd/registers` and `tools/lauterbach`. A PDF describing how to install and use the RealView register definitions is in the `doc` directory.

9 IOMUX Tool

IOMUX Tool helps document and validate the IOMUX configuration for a board. The tool is intended to be used by board designers to generate a design file describing pin mux and pad settings for a board. It can then automatically generate source code (.h and .c files) with functions to setup IOMUXC register configuration. This frees device driver developers from board design details of each pin and pad. The source code generated by the tool is used as-is by the SDK.

The IOMUX Tool executable is available with the release package in the `tools/windows/iomux` folder. Also included in that directory is the IOMUX Tool manual PDF.

IOMUX design files for all boards supported by the SDK are located under the chip- and board-specific source directories at `sdk/<chip>/<board>/`. For instance, the i.MX6DQ SABRE-AI design file is at

`sdk/mx6dq/sabre_ai_rev_b/i.MX6DQ_Sabre_AI_RevB.IomuxDesign.xml`.

10 Doxygen

The SDK comes with a Doxygen configuration file and all source code in the SDK is commented with Doxygen-style documentation comments. This means that Doxygen can be used to generate comprehensive documentation for all source files and functions in the SDK.

Install Doxygen by downloading it from:

<http://doxygen.org/>

Doxygen also depends on the Graphviz project to produce graphs.

<http://www.graphviz.org/>

If the Graphviz tool `dot` is not installed in a directory in your environment `PATH`, you will need to edit the `Doxyfile` configuration file and modify the `DOT_PATH` setting as appropriate. Doxygen will still run and produce documentation without `dot` being available, but the output documentation will contain broken image links to graphs.

To generate the SDK source code documentation, simply execute the `doxygen` command with no arguments from the SDK installation root directory. HTML documentation output will be

placed into the `doc/html` directory. To view the documentation, open `doc/html/index.html` in a browser:

11 Running the video decoder demo

This section described in detail how to setup and run the video display demo on i.MX6 Series EVB / Sabre Lite / Smart Device boards.

Style key for the steps below:

Text for console output.

Text for user input.

Text for important hints.

1. How to build the program?

```
./tools/build-sdk -t mx6dq -b evb -r vpu -c
```

2. How to setup the demo?

- a) Create the image over SD card. Under Linux, using `fdisk/mkfs.vfat/dd` to create a bootable image together with the FAT32 file system on the same SD card.

➤ `sudo fdisk /dev/sdx`, `sdx` is the device name of your SD card.

```
sudo fdisk /dev/sdb
```

```
Command (m for help): m
```

Delete existing partition if there is.

```
Command (m for help): d
```

```
Selected partition 1
```

```
Command (m for help): n
```

```
Command action
```

```
  e   extended
```

```
  p   primary partition (1-4)
```

```
p
```

```
Partition number (1-4): 1
```

```
First cylinder (1-1023, default 1): 256
```

Here the start address should be larger than 2M (space reserved to program the test binary). If one cylinder is 4k, then here 1G is reserved.

```
Last cylinder or +size or +sizeM or +sizeK (256-1023,  
default 1023): 1023
```

```
Command (m for help): w
```

```
The partition table has been altered!
```

Calling ioctl() to re-read partition table.
Syncing disks.

Now there's one partition on the SD card.

cat /proc/partitions

major	minor	#blocks	name
8	0	78125000	sda
8	1	104391	sda1
8	2	78019672	sda2
253	0	75956224	dm-0
253	1	2031616	dm-1
8	16	3872256	sdb
8	17	2904576	sdb1

- Using mkfs.vfat to format the partition:

sudo mkfs.vfat /dev/sdb1

- Copy two video clips to the SD card. *Note that the filename should be less than 8 characters with ".264" extension. The video should be raw H.264 encoded files with no container. The program will find the first valid .264 file for playing.*

- Burn the image to the SD card.

sudo dd if=output/mx6dq/evb_rev_a/bin/mx6dq_evb_rev_a-vpu-sdk.bin of=/dev/sdb seek=2 skip=2 bs=512 && sync

Note that seek=2 skip=2 is mandatory, without them the MBR of the file system will be overwritten.

- b) Put the SD card into SLOT4.
- c) Set the boot switch to boot from SD4.
- d) Plug the HDMI cable of the first display to J5 of the CPU board.
- e) Connect the serial cable for console output and 5V power supply, power on the board.

```
*****
Diagnostics Suite (1.0) for i.MX6DQ T01.0 Sabre-Lite Board Rev A
Build: Mar  8 2012, 16:22:20
Freescall Semiconductor, Inc.
*****

===== Clock frequencies(HZ) =====
Cortex A9 core   : 792,000,000
DDR memory      : 528,000,000
UART2 for debug : 80,000,000
EPIT1 for system timer : 66,000,000
=====

===== DDR configuration =====
data bits: 64, num_banks: 8
```

```
row: 14, col: 10
DDR type is DDR3
Chip select CSD0 is used
Density per chip select: 1024MB
=====

[INFO] Product Info: i.MX6Q
[INFO] VPU firmware version: 2.1.2
      0 - VPU DECODER TEST
      1 - VPU ENCODER TEST
      x - to exit.
```

From this prompt, the user is invited to choose what to do.

3. Limitations

- a) There is no resizing on the video output. For example if you are decoding some video clips with resolution other than 1080p, the video will show on the top-left of the screen with its original size.
- b) Video clips must be raw without container.
- c) Currently only H.264 video decoding is supported. Other formats such as VC1, H.263, MPEG3, MPEG4 may be added in a future release.

12 Known limitations

These are the current set of known bugs and issues in the Platform SDK.

- The TEMPMON driver over-temperature alarm feature does not work.
- The GPMI NAND driver API calls to read and write NAND pages with BCH ECC do not work correctly.
- The CCM driver (ccm_pll.c) does not support getting clocks for all peripherals from its get_peri_clock() API.