Freescale Semiconductor
Application Note

Document Number: AN5285
Rev. 0, 05/2016

MAC57D5xx Start-Up Sequence

by: Manuel Rodriguez

1 Introduction

The MAC57D5xx family is the next generation platform of devices specifically targeted to the instrument cluster market using single and dual high-resolution displays. Leveraging the highly successful MPC56xxS product family, NXP is introducing a multi-core architecture powered by ARM® Cortex®-M (for real time) and Cortex-A processors (for applications and HMI), coupled with 2-D Graphics Accelerators (GPU), Heads Up Display (HUD), Warping Engine, Dual TFT display drive, integrated Stepper Motor Drivers, and a powerful I/O Processor, that will offer leading edge performance and scalability for cost-effective applications.

This application note describes a basic start-up sequence that can be used to bring up the MAC57D5xx family of microcontrollers. This covers starting up the Cortex-M4F core, AIPS permissions initialization, watch dog initialization, Clock and PLL setup, ECC initialization, and data initialization.

An overview of the device architecture is shown in the figure below:

Contents

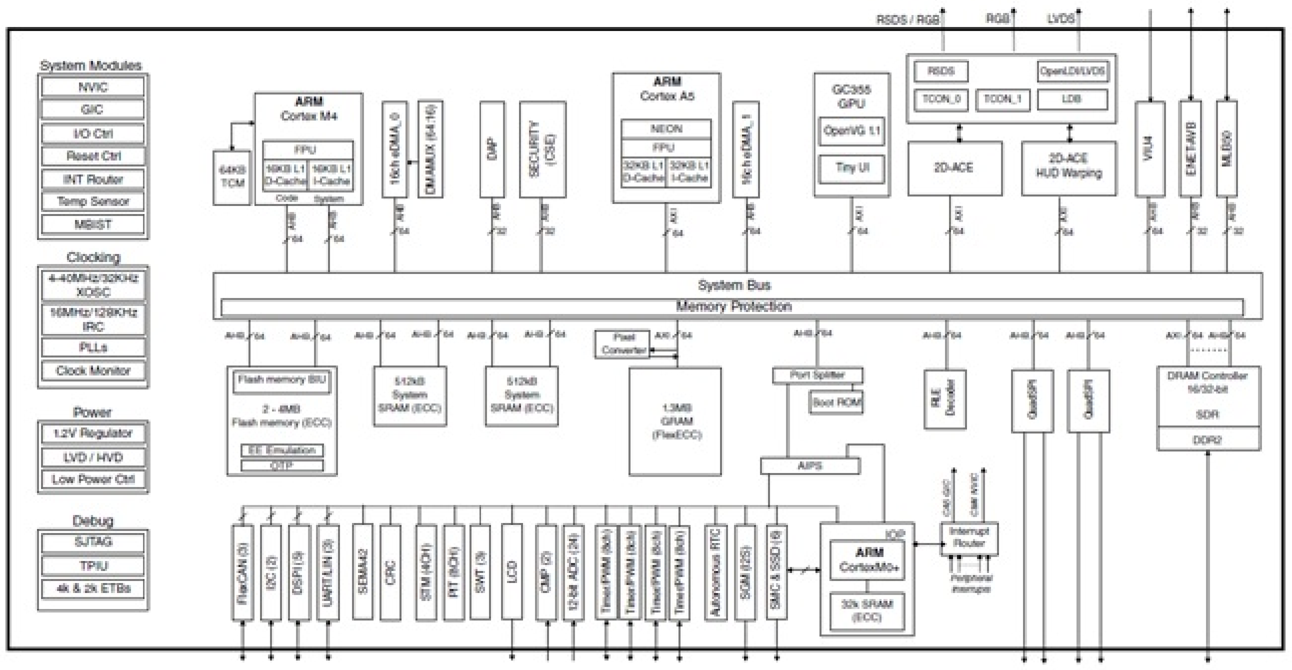

Figure 1. Boot sequence

2 Boot sequence overview

The boot sequence can be split into four major steps:

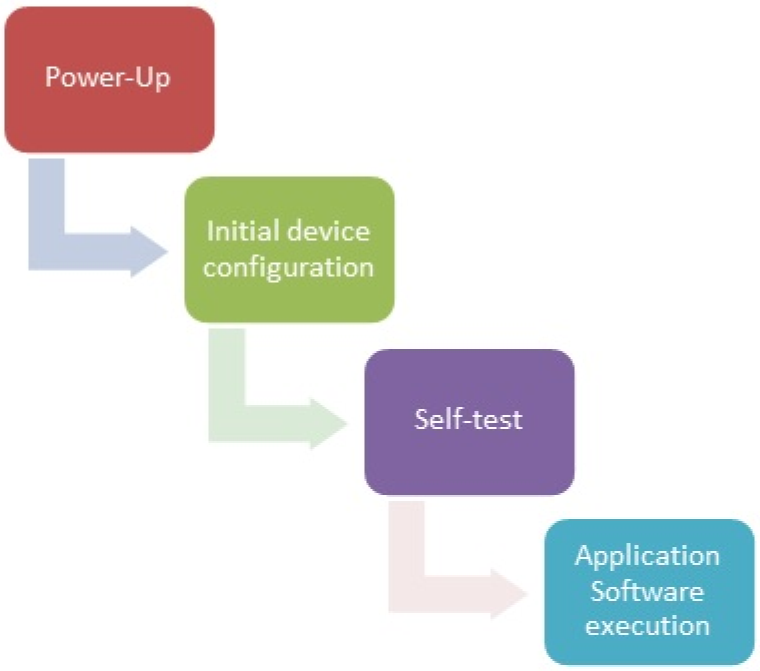

Figure 2. Boot sequence

2.1 Power-up

Once the device is powered-up or after a destructive reset, the Fast Internal 16 MHz RC (FIRC) oscillator is started and used as the clocking source. Before this phase can be exited, the Flash must finish its initialization and signal that it is ready to be accessed.

2.2 Initial device configuration

The System Status and Configuration Module (SSCM) is in charge of loading the factory written Device Configuration Format Records (DCF Records). These records dictate the initial configuration of some modules of the device after reset such as:

- STCU2 – This module is in charge of executing the Self-test of the device.
- PASS – This module is used to implement password-based read and write protection for the Flash blocks.
- Tamper Detect – The Tamper Detection Module provides a type of flash memory erase protection.
- Miscellaneous – This features diverse device configuration such as LCD PAD control and Oscillator Control registers.

The SSCM loads the configuration written at factory at this stage and the initial configuration of the device as well as the life cycle is set. The factory configuration can be seen at the DCF_sheet.xlsx attachment in the Reference Manual of the device.

The configuration written by factory can be overridden by writing to the UTEST memory area the desired DCF records. The UTEST DCF Records area for the MAC57D5xx is 0x1840_0300, the structure of the DCF records is shown below.

Table 1. DCF Record structure

Data	DCF client
32-bit word of data to be loaded to the DCF client	32-bit address of the DCF client to be configured

The SSCM looks at the first word at 0x1840_0300 for a valid DCF start record (Table 2) and if one is found it starts parsing the DCF records until an empty entry is found (0xFFFF_FFFF).

Table 2. DCF start record

Data	DCF client
0x05AA_55AF	0x0000_0000

NOTE

The UTEST DCF Records area is an OTP memory region. Once a DCF Record has been written it cannot be erased. Some DCF Records can be overridden by writing later to the same DCF client.

2.3 Self-test

The Self Test Control Unit (STCU) controls the execution of the built-in self-tests (BISTs). The default STCU configuration is programmed in the MAC57D5xx during factory test and enables BIST execution by default. The execution of the self-tests can be disabled, for information on how to disable the self-tests and an explanation of the kind of tests that are performed please refer to EB833: MAC57D5xx STCU BIST Configuration, available in nxp.com .

The self-tests if enabled are carried out at boot time and the results of these tests can be found at the Status registers of the STCU. For a more thorough explanation please refer to the device Reference Manual.

2.4 Application software execution

At last the Boot Assist Flash (BAF) takes over the boot process, first it sets up an exception vector table to trap any exception that might occur during execution. If any exception occurs during BAF operation, it issues a destructive reset. BAF uses CM4 TCMU for its stack, data and any code that executes from volatile memory.

BAF parses the DCF records area at 0x1840_0300 searching for a DCF start record to override the values programmed by the DCF records written by the SSCM (factory written values).

After writing the DCF records, if any, it searches through the first word location of each Flash memory blocks starting by the 5 x 16 KB blocks and the 4 x 256 KB blocks at last. The blocks are searched in the order shown in following table. Once a boot header is found, no further blocks are searched.

Table 3. Locations of Boot Headers

Search order	Block	Address
1	16 KB Code Flash	18F9_C000h
2	16 KB Code Flash	18FA_000h
3	16 KB Code Flash	18FA_4000h
4	16 KB Code Flash	18FA_8000h
5	16 KB Code Flash	18FA_C000h
6	256 KB Code Flash	1900_000h
7	256 KB Code Flash	1904_000h
8	256 KB Code Flash	1908_000h
9	256 KB Code Flash	190C_000h

A boot header structure is shown in the table below. The first half-word with the value 0x005A is considered valid for booting.

Table 4. Boot Header structure

Address offset	Contents
00h	Boot header start and configuration
04h	Reserved
08h	Secure Boot response timeout
0Ch	Secure Boot image length from start of boot sector (in bytes)
10h	Cortex-M4 entry point
14h	Reserved
18h	Reserved

If a boot header is found, the device starts executing the code set at the entry point field and the BAF finishes its operation.

If the boot header is not present at any flash block and the life cycle of the device is Customer Delivery or Freescale Production then the device attempts to perform a serial boot either via the LINFlexD interface or the FlexCAN interface.

If no valid boot header is found and the life cycle does not match Customer Delivery or Freescale Production then the device enters 'Static mode' in which the device is put in a power safe mode waiting for an external reset.

Once BAF finishes execution it restores any register that it might have accessed with its reset values, it also disables any module it might have enabled. The only registers that are not restored are the ones related to the boot process such as the MC_ME_CADDR0 (holds entry point address for the CM4) and SWT (Watch Dog). The complete list of registers that are not restored as well as its value after BAF is executed can be found in table "Reset values on BAF exit" of the Reference Manual.

3 Cortex-M4F initialization

The main core of the device is the Cortex-M4F, it is the only core available at boot time and after every Power-on-Reset event. The MAC57D5x does not support multicore boot.

Once the Application Software Execution phase starts, the first thing that must be initialized in order for the core to execute routines written in C is the Stack Pointer (SP). The assembly code that achieves this is the following:

```
ldr     r0, =__STACK_ADDRESS
mov     sp, r0
```

Once the SP is initialized the microcontroller can jump to an initialization routine and set the Vector Table Offset Register of the Cortex-M4F. The M4 cache can also be initialized at this stage. This application note contains software examples that performs these steps.

4 System initialization

Some of the basic modules that need to be initialized in almost every application are the following:

- AIPS Peripheral access
- Watchdog timer
- Clock dividers
- PLL configuration
- ECC initialization
- Data initialization

The initialization of each of these modules will be briefly detailed on the following sections. Once these initializations have been carried out the device will be ready to execute the desired application code. This application note contains software example that performs each of these initializations and can be used as a reference.

4.1 AIPS

The AIPS regulates the accesses of "masters" such as cores to the peripheral address space, for example, Flash memory, SRAM memory, and LINFlexD module.

The MAC57D5xx has 15 masters, the table below identifies the master ID with its module.

Table 5. Master ID assignments

Master ID	Master port number	Master name
0x0	M0	Cortex-M4 Code (64-bit AHB)
0x0	M1	Cortex-M4 System (64-bit AHB)
0x1	—	Cortex-M4 Debug
0x2	M2	CSE (64-bit AHB)
0x3	M3	eDMA_0 - 16-Channel DMA2 (64-bit AHB)
0x4	-	Cortex-M0+ System (32-bit AHB)
0x5	-	Cortex-M0+ debug
0x6	M16	Cortex-A5 Core (64-bit AXI)
0x7	M4	Off-platform Master port for MLB50 (64-bit AHB
0x8	M8	ENET Port0 - off-platform (64-bit AXI)
0x9	M9	ENET Port1 - off-platform (64-bit AXI)
0xA	M10	2D-ACE - off-platform (64-bit AXI)
0xB	M11	2D-ACE and HUD (64-bit AXI)
0xC	M12	DAP (64-bit AXI)
0xD	M13	VIU - off-platform (64-bit AHB)
0xE	M14	eDMA_1 - 16-Channel DMA (64-bit AHB)
0xF	M15	GC355 Port (64-bit AXI)

The AIPS registers can be accessed only in supervisor mode by trusted bus masters. Additionally, these registers must be read from or written to only by a 32-bit aligned access.

The Read/Write privileges are managed by the Master Privilege Register A/B. The privileges of all 15 masters have to be set before trying to perform an AHB access from one of these masters.

4.2 Watchdog Timer

The MAC57D5xx has three Software Watchdog Timers or SWT. One for each of the available cores:

- SWT0 (Cortex-M4)
- SWT1 (Cortex-A5)

SWT0 is the only watchdog timer that is enabled with a soft lock after reset. Once the soft lock is enabled, SWT_CR, SWT_TO, SWT_WN, and SWT_SK become read-only registers. To clear the soft lock the service register SWT_SR has to be written with 0xC520 followed by a write with 0xD928, this sequence unlocks the registers and allows them to be written.

To disable the SWT, the SWT_CR[WEN] must be cleared (written to zero). Other functionality can also be configured at the time this bit is written, such as setting the soft/hard locks and the behavior the device should have when an invalid access occurs. For more details please see the Reference Manual of the device.

4.3 Clock dividers

The MAC57D5xx offers the capability of sourcing the clock for each module from different sources (PLL, external oscillator, internal oscillator). At boot the device is clocked from the internal oscillator. If a greater accuracy is desired, the clock source can be changed to work with an external oscillator. The clock tree of the device can be found in the Reference Manual of the device under the Clocking overview chapter, this diagram showcases the different available configurations and clock sources for each module. Each module has its own frequency restrictions the maximum allowed frequency for the available modules is shown below.

Table 6. Maximum system level clock frequencies

Module	Max Frequency [1]
A5 Core	320 MHz
Crossbar and M4 Core	160 MHz
IO Processor	80 MHz
PBRIDGE	80 MHz
QuadSPI	Refer device datasheet for details.
2D-ACE	80 MHz pixel clock
	160 MHz internal logic clock (DMA, Arbitration, Input Buffers and Data path)
GC355 GPU	160 Mpixel/s peak output rate
	320 MHz (GPU dual port RAM and AXI frequency)
SGM	120 MHz
	240 MHz (SGM dual port memory interface)
OpenLDI	560 MHz
ENET	RMII: 50 MHz, MII: 25 MHz
DDR Interface	320 MHz
SDR Interface	160 MHz
TPIU output	80 MHz

1. These figures are for guidance. The specified maximum frequencies should be taken from the data sheet for the devices maskset.

The dividers for the modules that will be used must be configured with respect to the selected clocking source and enabled before trying to use the module.

4.4 PLL configuration

The MAC57D5xx has four instances of the PLL, the PLL_0 can be used in Frequency modulation and Non-FM modes, the remaining PLL's can be used in non-FM mode only.

Coming out of reset the PLL is disabled, if the PLL is required in the application it has to be configured before turning it on. The PLL diagram is shown below:

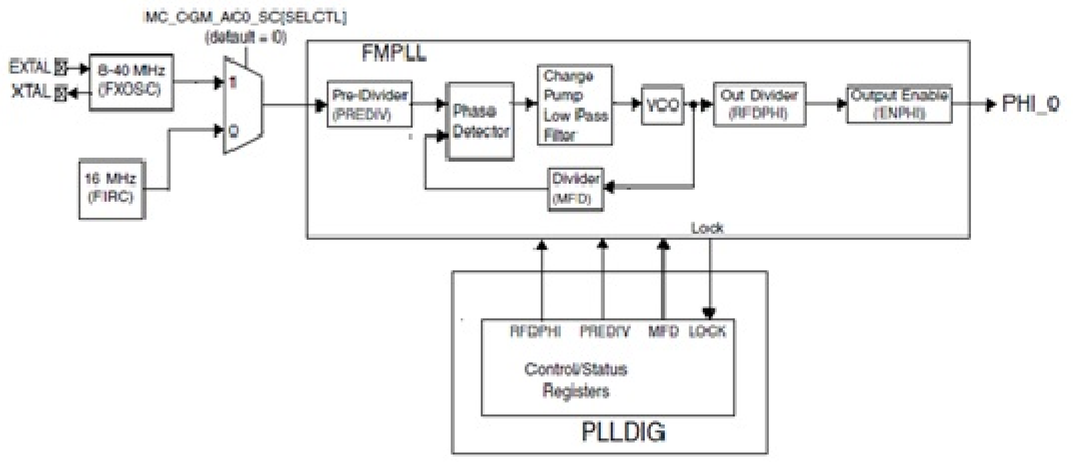

Figure 3. PLL block diagram

The frequency output of the PLL is controlled by the following equation:

$$f_{\text{pll\_phi}} = f_{\text{pll\_ref}} \times \left(\frac{\text{PLLDV[MFD]} + \frac{\text{PLLFD[MFN]}}{(\text{PLLCAL3[MFDEN]} + 1)}}{\text{PLLDV[PREDIV]} \times 2 \times (2^{\text{PLLDV[RFDPHI]}})} \right)$$

Equation 1. PLL output frequency

4.5 ECC initialization

It is essential that each memory address be written to a known value before it is read, to initialize the ECC. Without writing an address to a known value first, a read from this address will most likely generate an uncorrectable ECC event.

The memory sections that are initialized by the example code are the following:

Table 7. ECC memory areas

Memory section	Start	End
Tightly Coupled Memory Lower	0x1E00_0000	0x1E00_7FFF
Tightly Coupled Memory Upper	0x3E00_0000	0x3E00_7FFF
IOP RAM	0x3E40_0000	0x3E40_7FFF
SRAM	0x3EF0_0000	0x3EFF_FFFF

Additionally if the ECC is enabled on the GRAM section this memory area has to be initialized to a known value before use.

NOTE

The SRAM section has to be initialized with 64-bit writes, which can be accomplished by using the DMA.

4.6 Data initialization

Once the memories have been initialized the .bss (zero initialized data) and data (copy from ROM to RAM) sections can be initialized in RAM. The example code provided with this application note uses the data initialization routines provided by each IDE. If a custom initialization routine is required the linker file must be modified to generate the start/end symbols for each section. Data must be initialized before jumping to the main application.

Document Number AN5285
Revision 0, 05/2016