Presentation of MPC5645S MCU - Ultra reliable MCU for automotive and industrial instrument cluster

http://www.alexandre-boyer.fr

Alexandre Boyer
Patrick Tounsi

5e année ESE
October 2016

This document aims at providing basic information for application development on the microcontroller MPC56045S. This microcontroller is the central unit of the instrument cluster used in this lab.
This microcontroller has the same core architecture than the MPC5604B and shares several peripherals. That's why the content of the document is not exhaustive and does not detail every part of the microcontroller (MCU). Some basic peripherals described in the document " Presentation of MPC5604B MCU (Qorivva)" are omitted since they are similar. Moreover, only the peripherals and functions which are required for the lab are presented.
For more technical information about the component, please refer to the datasheet **MPC5645SRM.pdf.** Links to the datasheet will be provided in this document.

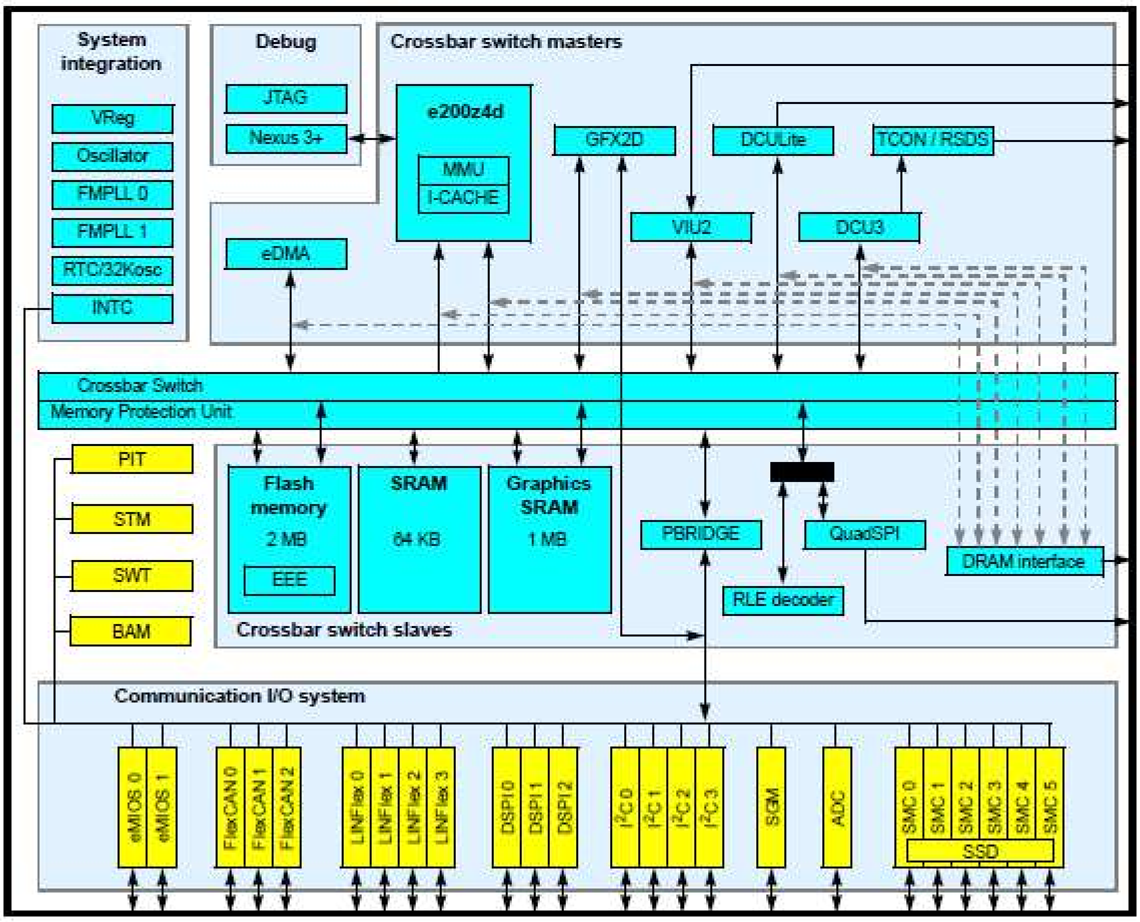
Remark: sometimes, the register names given in the datasheet do not match with those provided by the MCU library **MPC5645S.h**. Don’t hesitate to verify the right name in the library.

I - Presentation of the MCU MPC5645S

MPC5645S is a MCU developed by NXP Semiconductor dedicated to the instrument cluster applications, such as TFT-LCD display, gauge drive, sound generator. It is a 32 bit MCU dedicated to automotive body applications designed in CMOS 90nm technology. Its core is based on a Power Architecture ® and a e200z4d CPU. The version used in the Lab is MPC5645S, which is mounted in a LQFP 176 package.
Its main characteristics are:

- Up to 2MB of ECC Flash memory
- Up to 64 KB of ECC SRAM memory
- Up to 1 MB of Graphic SRAM
- One interface for an external Quad SPI serial Flash memory
- Core frequency up to 125 MHz, two internal PLL
- An interrupt controller (INTC) with 171 selectable priority interrupt vectors (163 peripheral interrupt request sources and 8 software interrupt request sources, 16 priority levels)
- 16 channels for eDMA
- 1 Display Control Unit (DCU3)
- 1 Graphic accelerator
- 1 Video Input Unit (VIU2)
- 4 Stepper Motor Controllers (SMC) with Stepper Stall Detect (SSD)
- 16 channels for 10-bit analog-to-digital converters (ADC)
- 2 serial peripheral interface (DSPI) modules, 3 serial communication interface (LINFlex), 3 CAN modules (FlexCAN)
- Up to 128 configurable general purpose input-output (I/O)
- 4 periodic interrupt timers (PIT) with 32-bit counter resolution
- Device testing based on JTAG bus (IEEE 1149.1)

Fig. 1 presents the block diagram of the MCU.



ADC	– Analog-to-Digital Converter	PBRIDGE	– Peripheral Bridge
BAM	– Boot Assist Module	PIT	– Periodic Interrupt Timer
eDMA	– Enhanced Direct Memory Access Controller	RLE	– Run Length Encoding
DCU3	– Display Control Unit	RSDS	– Reduced-Swing Differential Signal interface
DCULite	– Display Control Unit Lite	RTC	– Real Time Clock
DSPI	– Serial Peripherals Interface	SGM	– Sound Generator Module
eMIOS	– Enhanced Modular Input/Output System	SMC	– Stepper Motor Controller
FlexCAN	– Controller Area Network Controller	SSD	– Stepper Stall Detect
FMPLL	– Frequency-Modulated Phase-Locked Loop	STM	– System Timer Module
GFX2D	– OpenVG Graphics Accelerator	SWT	– Software Watchdog Timer
INTC	– Interrupt Controller	TCON	– Timing Controller
JTAG	– Joint Test Action Group interface	VIU2	– Video Input Unit
MMU	– Memory Management Unit	VReg	– Voltage regulator
QuadSPI	– Quad IO serial flash interface		

Figure 1 - Block diagram of MPC5645S (MPC5645SRM.pdf - p. 58)

II - MPC5645S programming main steps

This part aims at giving the main steps for the programming of the MCU. You are not forced to follow this sequence, it intends only to help you to start with programming.

- Initialization of system clock and modes for system and peripherals (see Chapters 3 and 4 for clock generation, Chapter 5 for mode entry module MC_ME).

The operation mode must be defined at initialization for every peripheral. Enter in RUNx (x = 0 to 3) mode (see Chapter 5 for mode entry module MC_ME)
- Activate/inhibit software watchdog
- Configure input-output pads (direction, alternate function activation, output drive, pull-up, pull-down, filtering) (see chapter 8 for System Integration Unit Lite module SIUL)
- Configure peripherals (clock, interrupt enable, parameters, energy mode…)
- Installation of INTC interrupt handlers
- Enable maskable interrupt requests
- Main program

The register names can be found in the MPC5645S datasheet, but the given names can differ from the actual name defined in the MCU library. Refer to Refer to the header file MPC5645S.h (normally included in your projects) to find the correct names of registers and bits.

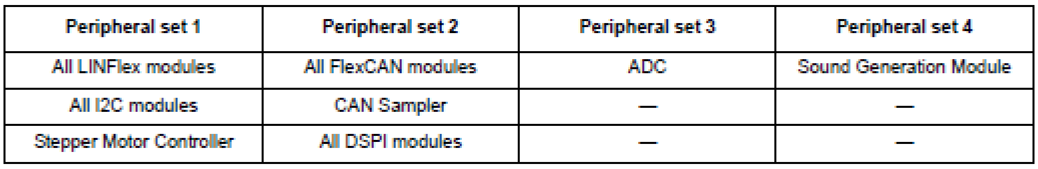
III - Clock generation description

Refer to Chapter 8 – Clock description of MPC5645SRM.pdf for more details about the clock architecture, the different clock sources and Clock generation module (MC_CGM) for more details about the internal clock generation. Only the configuration of the Pierce oscillator (FXOSC) and the primary PLL (FMPLL0) are presented in this document. The activation and selection of clock sources for the system clock are managed by the mode entry MC_ME module (see chapter IV of this document).

1. Clock architecture

The architecture of the internal clock is described in Figure 3. The system clock (sys_clk) can reach up to 125 MHz. It can be built from three selectable sources:
- Fast external quartz oscillator (FXOSC), 4 – 16 MHz
- Fast internal RC oscillator (FIRC), 16 MHz
- Primary Frequency modulated phase locked loop (FMPLL0), synchronized for a 4 to 120 MHz clock reference. It can deliver a clock frequency up to 256 MHz.

Except the peripherals included in one of the four peripheral set (see Figure 2) of those using an auxiliary clock use the system clock as reference clock.



Peripheral set 1	Peripheral set 2	Peripheral set 3	Peripheral set 4
All LINFlex modules	All FlexCAN modules	ADC	Sound Generation Module
All I2C modules	CAN Sampler	—	—
Stepper Motor Controller	All DSPI modules	—	—

Figure 2 – Peripheral sets (MPC5645SRM.pdf - p. 203 – Table. 8-1)

The circuit includes five auxiliary clocks dedicated to certain peripherals. If these peripherals are synchronized by an auxiliary clock, they can operate at a clock rhythm independent from the system clock. The five auxiliary clocks are:
- Auxiliary clock 0: Display Control Unit (DCU3)
- Auxiliary clock 1: eMIOS0
- Auxiliary clock 2: eMIOS1
- Auxiliary clock 3: Quad SPI

- Auxiliary clock 4: DCU Lite

The auxiliary clocks can be provided by the FXOSC, the FIRC, the primary and secondary FMPLL (FMPLL0 and 1).
The quality of clock sources is checked by the Clock Monitor Unit (CMU). This module can detect loss of clock integrity and switch to a SAFE mode in case of clock failure interrupt. It can also be used as frequency meter.

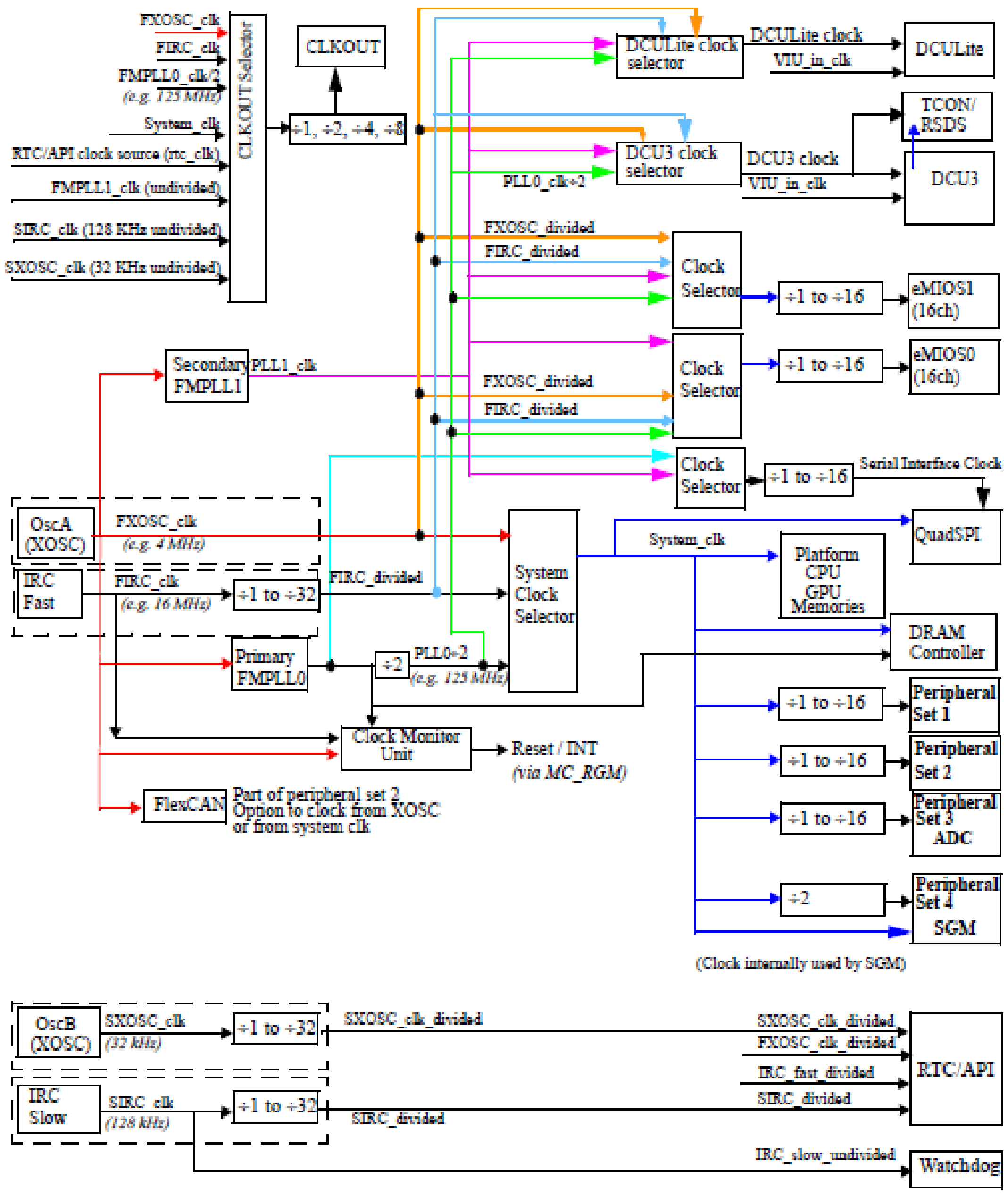
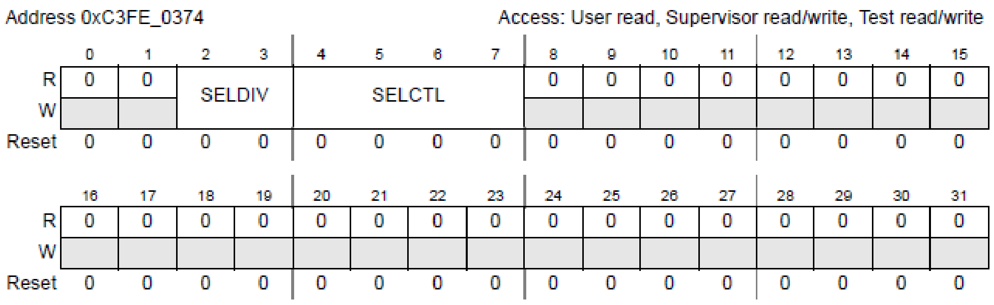


Figure 3 – Clock architecture (MPC5645SRM.pdf - p. 204 – Fig. 8-1)

The selection of a clock source for the system clock is done with the register CGM_OCDS_SC. The field SELCTL selects the clock source while the field SELDIV configures the clock division ratio. The register CGM_SC_SS gives the status of the source of the system clock.


Address 0xC3FE_0374 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

The sources of the auxiliary clock and the dividing ratios are configured by the registers CGM_ACx_DC[0..3], x = 0..4.

1. *Fast external oscillator (FXOSC)*

Refer to Chapter 8.4.1 for more information about FXOSC. This Pierce oscillator uses either an external oscillator circuit or an external quartz crystal. It can provide a clock source for the system clock and an input for the FMPLL0 and 1. The energy management, the activation and the selection of FXOSC as system clock are controlled by the mode entry MC_ME module.
This block does not contain any configuration register. The FXOSC is activated by the bit FXOSCON in the ME.RUN[x] register of the MC_ME module. The availability of a stable oscillator clock is indicated by the status bit S_FXOSC in the register ME_GS of the MC_ME module.

2. *FM PLL0*

Refer to chapter 8.5 for more information about the two FMPLL. Only the primary FMPLL (FMPLL0) is described here, as it is the only PLL that can serves to produce the system clock. Note that FMPLL1 can be used as auxiliary clock source.
The FMPLL enables the generation of high speed clock (up to 256 MHz) from 4-120 MHz clock source, which can be configured by software. FMPLL supports frequency modulation of the system clock in order to reduce electromagnetic interference emission. The modulant signal is a triangular waveform, with frequency up to 100 KHz and modulation depth comprised between 0 and 4 %. The energy management, the activation and the selection of FMPLL as system clock are controlled by the mode entry MC_ME module.
Figure 4 presents the block diagram of the FMPLL. The frequency of the PLL output (PHI) depends on register IDF, ODF and NDIV, according to the following formula: $phi = \dfrac{clkin \times NDIV}{IDF \times ODF}$. The selection of NDIV, IDF and ODF register content must be done with the following constraints:

- The VCO frequency range is between 256 and 512 MHz. If you try to make it operate at lower or larger frequency, the PLL operation could be degraded.
- NDIV values must be ranged between 32 and 96
- IDF can accept any number between 1 and 15
- ODF is coded on 2 bits in order to represent only 4 values: 2, 4, 8 or 16

For example, let’s suppose that the FXOSC is the source generator for the PLL and delivers a 8 MHz clock: clkin = 8 MHz. Let’s suppose that we want to generate a PLL output frequency equal to 45 MHz: phi = 45 MHz. A possible configuration is: NDIV = 90, IDF = 2, ODF = 8. With this configuration, the VCO operates at 360 MHz.

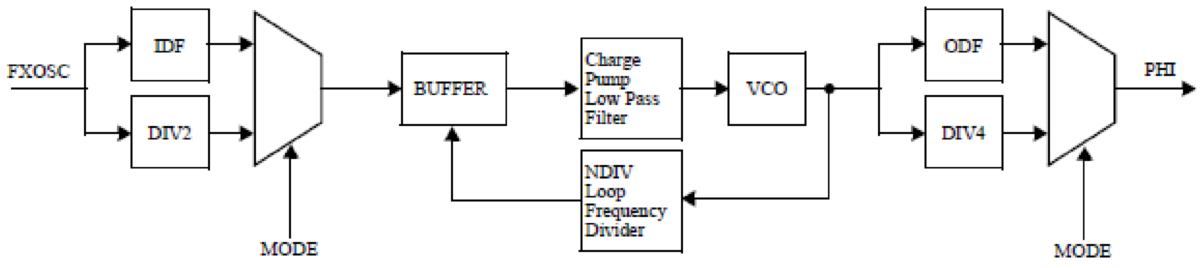


Figure 4 – FMPLL block diagram (MPC5645SRM.pdf - p. 242 – Fig. 8-29)

FMPLL0 proposes also the 1:1 mode. In this configuration, the output frequency is half the input frequency : $phi = \frac{clkin}{2}$.

The configuration of the PLL operation is controlled by the register FMPLL_CR. The register fields IDF, ODF and NDIV sets the PLL output frequency. These values must be changed only when the PLL is not selected as clock source. Setting the bit EN_PLL_SW enables the progressive clock switching which improves the transition to FMPLL as system clock. The bit Mode enables the 1:1 mode. Loss of lock and PLL failure are indicated by the bits UNLOK_ONCE, S_LOCK and PLL_FAIL_FLAG.

Offset 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	IDF				ODF		0	NDIV						
W	▨	▨							▨							
Reset	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	en_pll_sw	mode	unlock_once	0	i_lock	s_lock	pll_fail_mask	pll_fail_flag	0
W	▨	▨	▨	▨	▨	▨	▨			▨	▨	w1c	▨		w1c	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The configuration of the frequency modulation is set by the register FMPLL_MR. Three parameters need to be defined as shown in Fig. 4: the period of the modulant signal (Tmod), the modulation depth (Mod_depth (%) = 100×md/Fmod) and the type of spreading (center spread or down spread), where Fmod is modulation frequency and md the amplitude of frequency excursion. The modulation depth or index is limited to +/-2 % (center spread) ou -4 % (down spread), the maximum modulation frequency is 100 KHz.

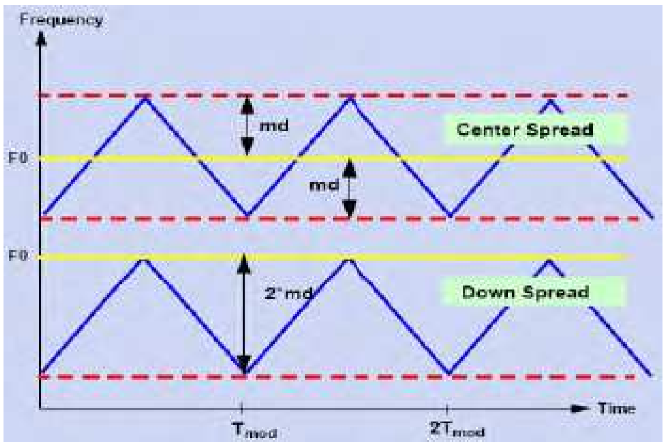
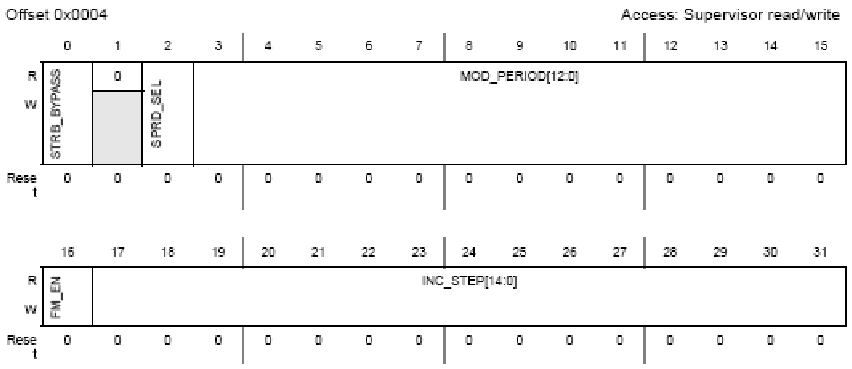


Figure 5 – Frequency modulation principle in FMPLL block (Bolero512K_RM_Rev7_07_2010.pdf - p. 87 – Fig. 3-10)

The field MOD_PERIOD sets the modulant period. Its equivalent binary value is equal to: $MOD\_PERIOD = \dfrac{F_{ref}}{4 \times F_{\text{mod}}}$, where Fref is the frequency at the output of the feedback divider (NDIV). The field INC_STEP sets the modulation index. Its equivalent binary value is equal to: $INC\_STEP = round\left(\dfrac{(2^{15}-1) \times NDIV \times Mod\_depth(\%)}{100 \times 5 \times MODPERIOD}\right)$. The type of spread is defined by the bit SPRD_SEL. If STRB_BYPASS bit is set, the field INC_STEP, MOD_PERIOD and the bit SPRD_SEL must be changed only when the PLL is in powerdown mode. The frequency modulation is enabled by setting the bit FM_EN. The FM must be enabled only when the PLL is active.


Offset 0x0004 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STRB_BYPASS	0	SPRD_SEL	MOD_PERIOD[12:0]												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FM_EN	INC_STEP[14:0]														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After reset, FMPLL is placed in powerdown mode. Its switch on is controlled by software through the MC_ME module. Its switch on is controlled by software through the MC_ME module (ME_<mode>_MC register, FMPLLON bit).
The availability of a stable FMPLL clock is indicated by the status bit S_FMPLL in the register ME_GS of the MC_ME module.

IV - Mode entry module (MC_ME)

This block controls the different modes of the MCU and the transition sequences between the different modes. The notions of modes and transitions between modes are essential to configure the MCU correctly and initiate the user mode, which the normal operation mode.

Refer to chapter 29 – Mode entry module for more details about the MPC5645S's modes.
The MPC5645S has the same operating modes than the MPC5604B.

1. Presentation of the different modes

The MCU proposes different modes corresponding to different usages (system configuration and monitoring, user mode, low power modes…) (refer to table 29-1 p 1065). The embedded software executes only in DRUN, SAFE, TEST and RUN0..RUN3 modes. RESET, DRUN, SAFE and TEST modes are system modes. They are dedicated to the configuration and the monitoring of the system. RUN0..RUN3, HALT0, STOP0 and STANDBY0 are user modes. HALT0, STOP0 and STANDBY0 are low power modes. In the chapter Wakeup Unit, the procedure to exit these low power modes will be detailed. The configuration of the MCU mode depends on the requirements in term of energy management and processing power.

- RESET: the application is not active, the chip configuration is initialized. The system enters in this mode after a reset.

- DRUN: entry mode for the embedded software. It enables the configuration of the system at the start-up. This is the only mode entry to a user mode. If the embedded software does not enable a transition between DRUN mode and a user mode, the main program defined by the user cannot execute. The system enters in this mode after the end of Reset mode, and after software request from RUN0..RUN3, SAFE, TEST modes, and a wake up request from STANDBY mode.

- SAFE: the system enters in this mode after the detection of a recoverable error. The system exits this mode after a reset or DRUN from software.

- TEST: for device self-test. The system enters in this mode from DRUN mode by software request. The system exits this mode after a reset or by software request to come back in DRUN mode.

- RUN0 .. RUN3: these are the embedded software modes where most processing activity is done. 4 RUN modes are provided to enable different power and clock configuration. The system enters in one of these modes after DRUN by software request, interrupt event from HALT0, interrupt or wake up event from STOP0. The system exists one of these modes after reset, entry in SAFE mode after an hardware or software error, HALT0, STANDBY0 or STOP0 by request.

- STOP: Reduced activity low power mode. The wakeup signals are processed rapidly, contrary to HALT mode. By default system clock is FIRC, but it can be switched off. The data and flash memories are powered down but can be activated; the main regulator is switched on. See chapter Wakeup Unit for more details about the exit of STOP mode.

- HALT: Reduced activity low power mode. The clock core is disabled. The analog peripherals can be switched off. The system enters in this mode by software request from RUN0..RUN3 modes. The systems leaves this mode after a reset, after a hardware or software failure to go in SAFE mode, or interrupt event to come back in previous RUN0..RUN3 modes. Contrary to STOP and STANDBY modes, wakeup signals cannot be used to exit from HALT mode.

- STANDBY: This is the most low power mode which ensures a reduced leakage current. Most of the blocks of the MCU are switched off from the power supply to reduce leakage current. Wake up from this mode is quite long. The system enters in this mode by software request from DRUN, RUN0..RUN3 modes. The system leaves this mode after reset, of after wake up event to enter in DRUN mode (see chapter Wakeup Unit). The wakeup from STANDBY0 mode is longer than from STOP0 mode. All the pins are in high impedance mode. Only the reset generation mode, power control unit, wake up unit, 8K RAM, RTC/API, CAN sampler, SIRC, FIRC, FXOSC are powered.

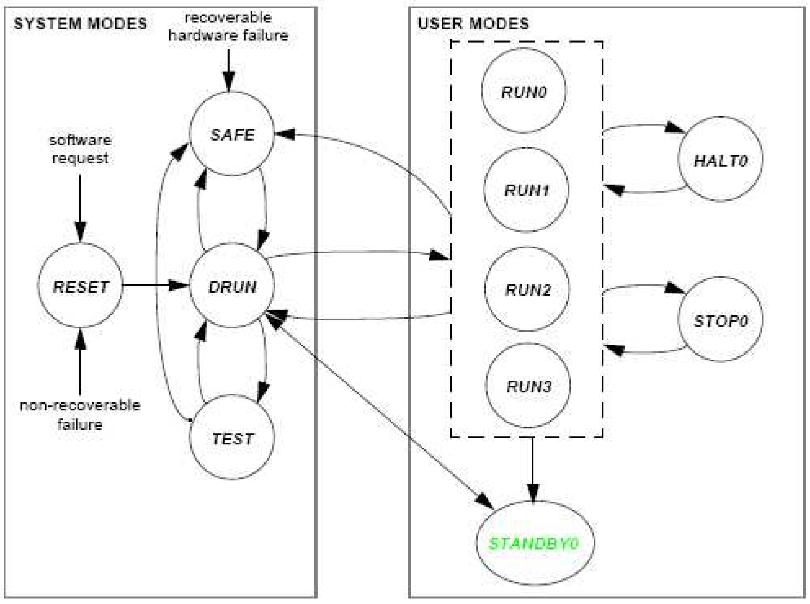
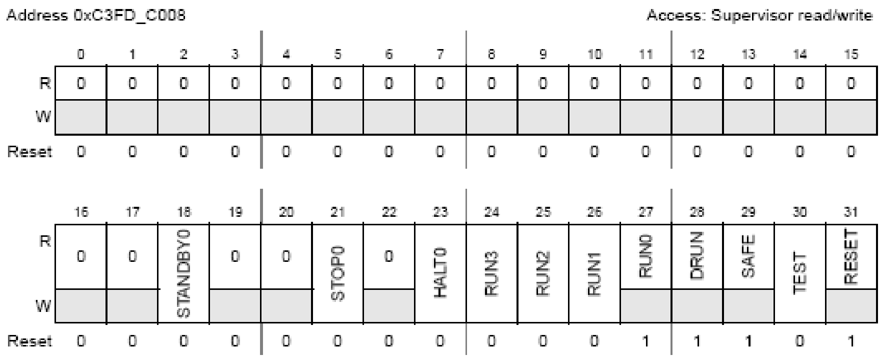


Figure 6 – Mode entry diagram and possible mode transitions (Bolero512K_RM_Rev7_07_2010.pdf - p. 144 – Fig. 5-24)

2. Mode entry module registers

a. Enabling modes

The Mode Enable Register MER (ou ME) allows enabling or disabling some MCU modes (except RESET, DRUN, SAFE and RUN0).



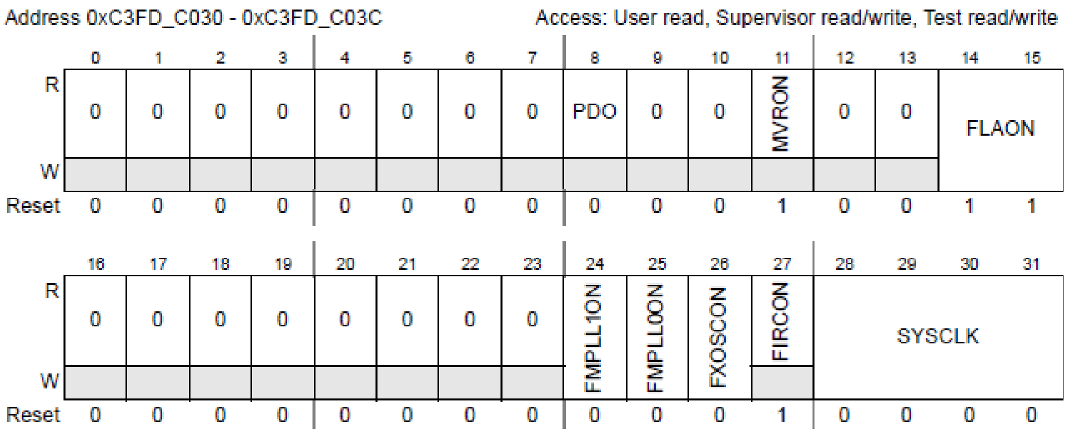
Address 0xC3FD_C008 Access: Supervisor read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STANDBY0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

b. Mode configuration

A mode configuration register is associated to each mode to control the connection or disconnection of some peripherals in the mode, such as the I/O output buffers, internal voltage regulator, data and code flash memory, PLL, fast external crystal and RC oscillators. It specifies also the system clock used by the system (PLL, crystal oscillator, fast RC oscillator…). All these registers have the same structure. The following figure shows the register structure for RUN0 .. RUN3 mode configuration registers, called ME.RUN[0] to ME.RUN[3].



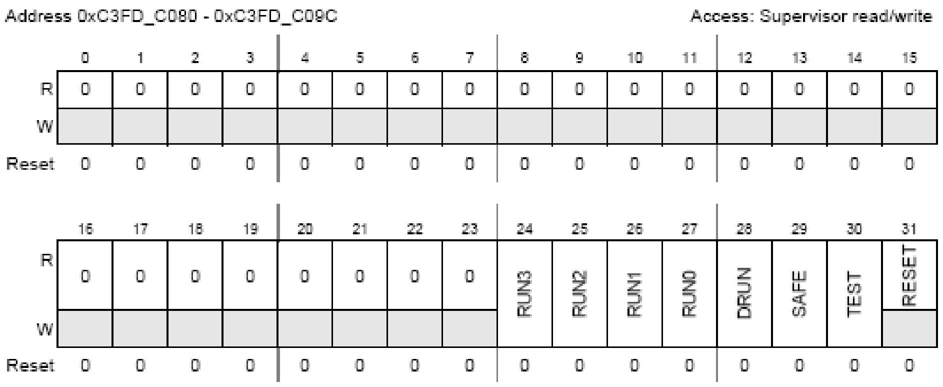
Address 0xC3FD_C030 - 0xC3FD_C03C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	0	0	FLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	FMPLL1ON	FMPLL0ON	FXOSCON	FIRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

c. Peripheral configuration

Up to eight different behaviors can be configured for the peripherals of the MCU in the different run modes. These 8 behaviors are defined by the Run Peripheral Configuration Registers 0 to 7 (RUNPC[0] to RUNPC[7]).
Setting a bit associated to a mode to '0' means that, if this configuration is given to a peripheral, this peripheral will be frozen in with clock gated during this mode. If this bit is set to '1', the peripheral will be active. For example, let's suppose that we define one behavior in RUNPC[0] and we write 0x00000030. If this configuration is associated to one peripheral, this peripheral will be active only in RUN0 and RUN1 mode. In all other modes, it will be frozen.

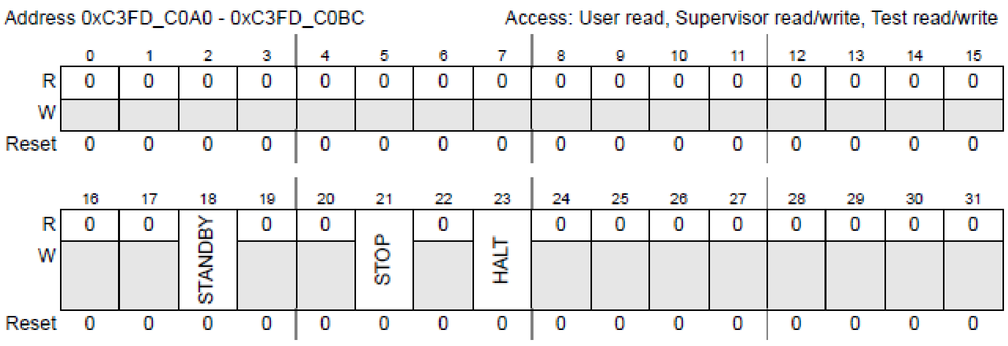


Address 0xC3FD_C080 - 0xC3FD_C09C Access: Supervisor read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

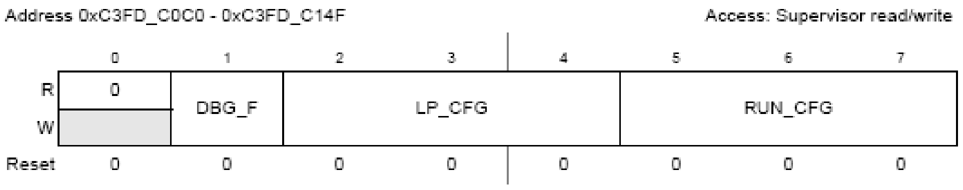
For the 3 non run modes (STANDBY, HALT and STOP), 8 behaviors can also be configures through the registers Low Power Peripheral Configuration LP_PC[0]to LP_PC[7].


Address 0xC3FD_C0A0 - 0xC3FD_C0BC Access: User read, Supervisor read/write, Test read/write

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

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

Once the different possible behaviors have been configured with registers RUNPC[0] to RUNPC[7], these behaviors can be associated to the 144 peripherals of the MCU. 144 registers called Peripheral Control Registers PCTL[0] to PCTL[143] are associated to each peripheral. These registers contains 3 fields: the field RUN_CFG defines which one of the 8 behaviors defined in RUNPC[0] to RUNPC[7] will be associated to the peripheral during the run modes. The field LP_PC defines which one of the 8 behaviors defined in LPPC[0] to LPPC[7] will be associated to the peripheral during the non run modes. The bit DBG_F sets the behavior of the peripheral in Debug mode.


Address 0xC3FD_C0C0 - 0xC3FD_C14F Access: Supervisor read/write

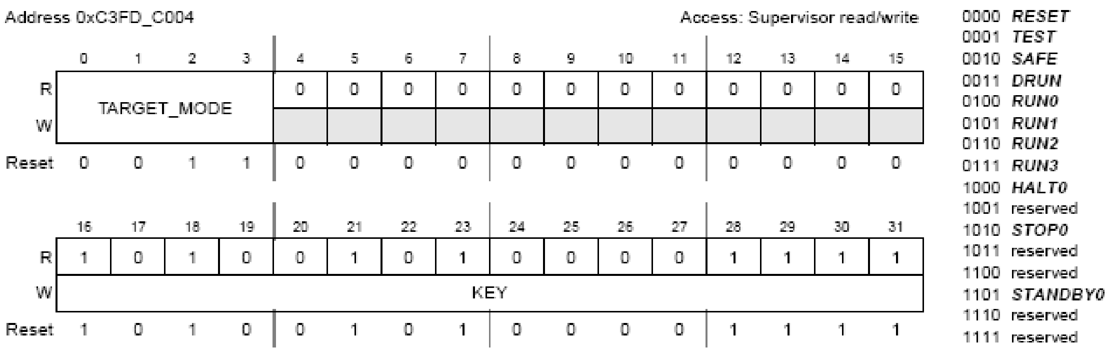
	0	1	2	3	4	5	6	7
R	0	DBG_F	LP_CFG			RUN_CFG		
W								
Reset	0	0	0	0	0	0	0	0

The status of the peripherals is given by the registers PS0, PS1, PS2 and PS3.

<u>Remark:</u> one number from 0 to 143 is associated to each peripheral. The following table (Table 29-2 p 1067) gives the number associated to each peripheral. For example, the number 32 is associated to the ADC0 block, the number 68 to the SIUL module (GPIO). The configuration of the ADC behavior according to the mode will be defined by register PCTL[32] and the configuration of the SIUL behavior by PCTL[68].

d. System mode selection and transition

The Mode Control Register MCTL is used to trigger mode change by software. The TARGET_MODE field defines the target mode to be entered by software request.


Address 0xC3FD_C004 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

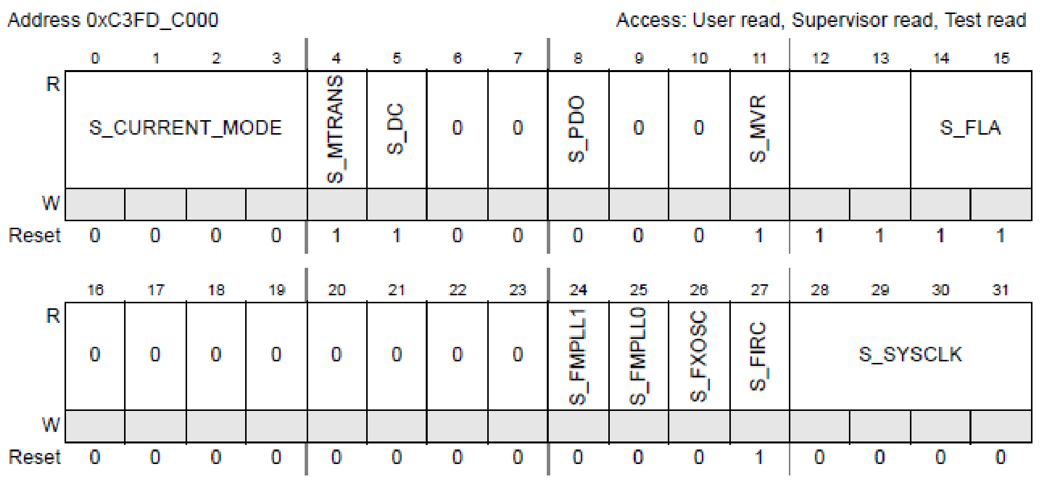
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
W	KEY															
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

0000	*RESET*
0001	*TEST*
0010	*SAFE*
0011	*DRUN*
0100	*RUN0*
0101	*RUN1*
0110	*RUN2*
0111	*RUN3*
1000	*HALT0*
1001	reserved
1010	*STOP0*
1011	reserved
1100	reserved
1101	*STANDBY0*
1110	reserved
1111	reserved

The KEY field is a control key to enable the writing in this register. The KEY is 0x5AF0. A different value is invalid and any writing in the register will be ignored. Actually, two writing of the register have to be done to force the device to enter in the mode defined by TARGET_MODE: first time with the good value of the key, a second time with the inverted value of the key. For example, suppose that we want the system to exit DRUN mode to enter RUN0 mode. The TARGET_MODE field must be equal to ‘0100’. Therefore, the two following lines have to be written in the software:

```
ME.MCTL.R = 0x40005AF0;       /* Enter the target mode and the Key */
ME.MCTL.R = 0x4000A50F;     /* Enter the target mode and the inverted Key */
```

The global mode status of the system is given by the register Glogal Status Register GS. The field S_CURRENTMODE notifies the current device mode. The bit S_MTRANS notifies if a mode transition is on-going. It gives also the status of several MCU peripherals.



Address 0xC3FD_C000 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CURRENT_MODE				S_MTRANS	S_DC	0	0	S_PDO	0	0	S_MVR			S_FLA	
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	S_FMPLL1	S_FMPLL0	S_FXOSC	S_FIRC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

3. *Summary – MCU initialization procedure*

The procedure to initialize the MCU is always the same and describes below. This procedure must be done in DRUN mode.

1. Enables the modes to be used
2. Configure the clock sources for the system clock and auxiliary clocks (see previous chapter)
3. Configure the modes to be used
4. Configure the peripherals
5. Switch from DRUN mode to a user mode (RUN0,1,2,3)

Remark:
When the MCU is in debug mode, the Software Watchdog (SWT) is disabled (See chapter Software Watchdog). In nominal operation, the SWT is activated. The SWT must be stopped or checked regularly to avoid unwanted MCU reset. Refer to chapter Software Watchdog of this document or chapter 4.2 of the MCU datasheet from more information about the configuration of the SWT.

V - Wake up Unit (WKPU)

This block manages the events which trigger a transition from low power modes (HALT0, STOP0 and STANDBY0) to RUN0..3 or DRUN modes. Refer to chapter 49 – Wakeup Unit for more details.

The microcontroller exits a low power mode either after a reset assertion, a interrupt request or a wakeup event (except for Halt0 mode). The wakeup signal can originate from either 24 external sources (specific pads such as CAN or LIN) or internal sources (API and RTC), as shown in the following table. Four interrupts are associated to these events.

Wakeup vector	Wakeup number	Function				Package		
		Port	#1	#2	#3	176	208	416
0	WKUP0	PB1	CANRX_0	—	—	x	x	x
	WKUP1	PB3	LIN_RXD_0	—	—	x	x	x
	WKUP2	PB4	SCK_1	MA0	—	x	x	x
	WKUP3	PB9	SCK_0	eMIOS1[20]	—	x	x	x
	WKUP4	PB10	CANRX_1	I2S_D0	—	x	x	x
	WKUP5	PB12	RXD_1	eMIOS1[19]	CS2_0	x	x	x
	WKUP6	PC0	AN[0]	—	—	x	x	x
	WKUP7	PC10	AN[10]	—	—	x	x	x
1	WKUP8	PF0	eMIOS1[19]	EVTO	DCULITE_B2	x	x	x
	WKUP9	PF2	NMI	—	—	x	x	x
	WKUP10	PF3	eMIOS1[21]	MSEO	DCULITE_B4	x	x	x
	WKUP11	PF8	SDA_0	CS2_1	RXD_1	x	x	x
	WKUP12	PJ4	VIU[0]	eMIOS0[21]	eMIOS0[23]	x	x	x
	WKUP13	PJ6	VIU[2]	eMIOS0[19]	eMIOS0[15]	x	x	x
	WKUP14	PK7	RXD_2	DCULITE_R2	TCON8	—	x	x
	WKUP15	PL0	AN[19]	CANRX_1	SDA_1	—	x	x
2	WKUP16	PL2	AN[17]	CANRX_0	eMIOS1[22]	—	x	x
	WKUP17	PL9	SCK_2	PDI_PCLK	eMIOS1[21]	—	x	x
	WKUP18	PM3	CANRX_2	RXD_3	TCON[4]	—	x	x
	WKUP19	PM10	RXD_2	CANRX_2	eMIOS0[16]	x	—	—
	WKUP20	PN0	DCULITE_HSYNC	—	TCON4	—	—	x
	WKUP21	PN2	DCULITE_R0	RXD_2	VIU[0]	—	—	x
	WKUP22	PN10	DCULITE_G0	RXD_3	VIU[2]	—	—	x
	WKUP23	PP2	DCULITE_B0	CANRX_2	VIU[4]	—	—	x
3	WKUP24	—	API	—	—	—	—	—
	WKUP25	—	RTC	—	—	—	—	—

Several registers are dedicated to the configuration and the management of wakeup events. The register WIFER enabled the different wakeup sources. Writing a ‘1’ in one of the 20 positions of the field IFE[23:0] enables one the external wakeup event (see previous table to find the number associated to a wakeup source).

Address: 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	IFE25	IFE24	IFE23	IFE22	IFE21	IFE20	IFE19	IFE18	IFE17	IFE16	IFE15	IFE14	IFE13	IFE12	IFE11	IFE10	IFE9	IFE8	IFE7	IFE6	IFE5	IFE4	IFE3	IFE2	IFE1	IFE0
W	▨	▨	▨	▨	▨	▨	▨	▨																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The register IRER enables interrupts generation when wakeup events are detected. The register WISR contains the interrupt flags. The wakeup event is activated either on rising or falling edge, depending on the configuration of register WIREER and WIFEER.

VI - GPIO pad configuration (System Integration Unit Lite)

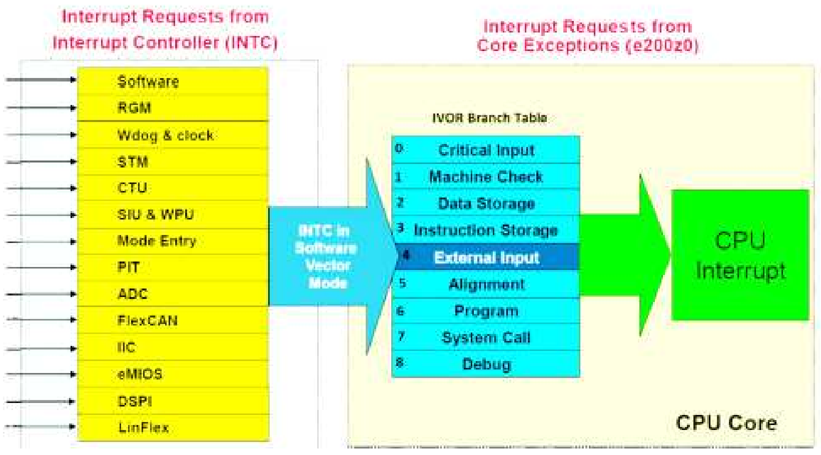
Refer to Chapter 43 – System Integration Unit Lite of the reference manual MPC5645SRM.pdf for the configuration of General Purpose I/O (GPIO) pads and the multiplexing of alternate functions associated to GPIO. The configuration of I/O pads of the MPC5645S is identical to that of the MPC5604B. So you can refer to the document presenting the MPC5604B to have the basic configuration of the SIUL.
The 128 GPIO of the MPC5645S can be configured independently through the PCR[0] to PCR[182] registers (Pad Configuration Register). The association between physical I/O pad and PCR register can be found in Table 5 - p. 105. For example, PCR[0] is associated to the pad PA[0]. Be careful with the MPC5645S mounted in a LQFP176 package: not all the I/O pads are physically connected to the package.
24 I/Os are associated to external interrupt request (EIRQ) inputs: EIRQ[0:23]. The lists of EIRQ input pads can be found in table 43-1 p 1482 of the reference manual MPC5645SRM.pdf.

VII - Interrupt configuration

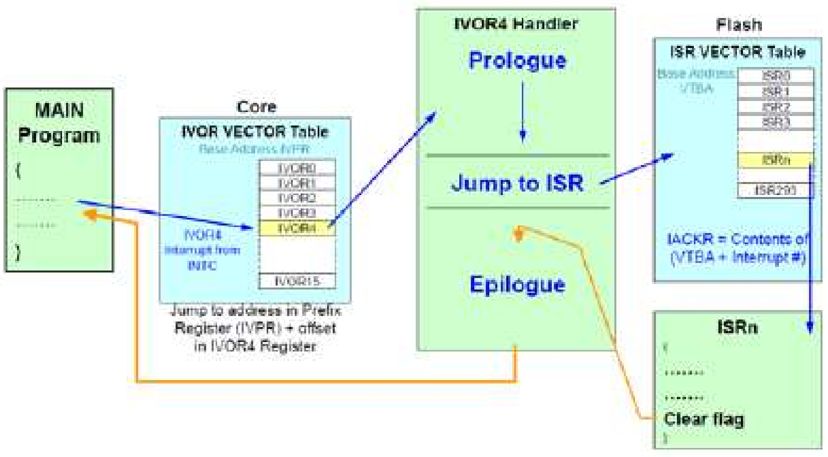
Refer to Chapter 26 – Interrupt Controller (INTC) of the reference manual MPC5645SRM.pdf for the configuration of priority of the different interrupt source.

1. *Presentation of INTC and interrupt vector*

The following figure describes how interrupt requests are handling and the position of the INTC block. In the MCU core (e200z4h), registers called Interrupt Vector Offset Register (IVOR) forms a branching table which handles the different exceptions which occur during the MCU operation. IVOR4 is the register used for interrupt handling.



The INTC module of the MPC5645S manages the ISR based on their programmable priorities and triggers IVOR4 exceptions. The following figure details how an ISR is handled in a mode called software mode (two ISR handling modes are proposed: hardware and software. Only software mode is considered in this document).



The MCU has 171 ISR, refer to Table 26-9 p 942 for the detail about the source of available ISR):

- 163 ISR are associated to peripherals (hardware (HW) triggered ISR)
- 8 ISR which can be configured by software (software (SW) triggered ISR)

Remark: SW triggered ISR are dedicated to:

- In a multiprocessor context, interruption of a processor activity by another processor
- In a program launched by a high level ISR, if a part of the program has a low level priority, it is possible to suspend the execution of this part by a software ISR. It improves the management of dead-lines of operation.

The priority of each ISR can be configured, with a level from 0 (lowest priority) to 15 (highest priority). Most of the HW triggered interrupts are maskable, i.e. it is possible to inhibit the ISR transmission to the INTS by the peripheral, by setting an interrupt enable bit (see configuration registers of each peripheral to know how to mask interrupt). Each time an ISR is launched, a flag bit is set. One flag bit is associated to one ISR source. The flag bits are in interrupt flag registers associated to the peripherals.

Important: don't forget to reset flag bit after ISR triggering. The flag indicates to the INTC that the peripheral sent an ISR. If the flag remains set, no more ISR can be generated. Most of the time, it is necessary to write a '1' in the flag bit to reset it. This is a particularity of NXP MCU.

Table 26-9 p 942 gives the interrupt vector table of the MPC5645S. The address of an interrupt vector is given in the following format:

Base address + Vector number

The vector number starts at 0 (for the software ISR number 0) up to 238 (for ISR launched by the graphical accelerator).
In order to associate an ISR coming from a peripheral or the software and a program to process the ISR, an interrupt handler has to be defined. This interrupt handler writes the address of the interrupt processing program at the interrupt vector address, and manages the ISR priority. We will see how to deal with interrupt handler with hardware or software ISR in the MPC5645S.

2. Enabling maskable interrupt

Maskable interrupt must be enabled at two levels: at local level (i.e. at peripheral level) by a interrupt enable bit associated to ISR source, and at global level. In order to enable ISR in the MCU, you must execute this routine in your program:

```
void enableIrq(void) {
  INTC.CPR.B.PRI = 0;          /* Single Core: Lower INTC's current priority */
  asm(" wrteei 1");              /* Enable external interrupts */
}
```

3. Configuring hardware triggered interrupt

HW triggered interrupt are most of the time maskable interrupt, so the peripheral configuration must enable ISR and the maskable interrupt must enabled at global level. INTC is implemented in several files: INTCInterrupt.h, INTCInterrupt.c, Eceptions.h, Exceptions.c. They contain the routines used to execute the ISR handling procedure. The following function configure the interrupt handler and the interrupt priority:

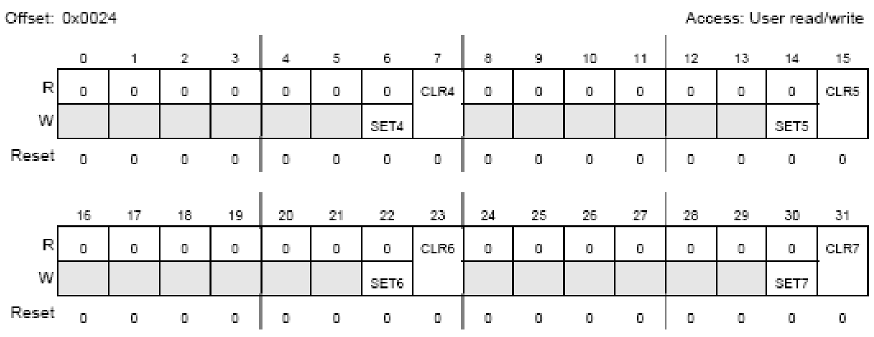
INTC_InstallINTCInterruptHandler(My_ISR_program,vector_number,priority_level);

My_ISR_program is the name of the program that the programmer wants to launch when the ISR is triggered by the peripheral. Vector_number is the number of the interrupt vector associated to the ISR (see Table 26-9 p 942). Priority_level is the level of priority associated to the ISR and ranges from 0 to 15.
For example, let's suppose that you design a program that launches the periodic interrupt timer Timer PIT1. At each time-out of PIT1, you want to trigger an interrupt that launches a function called My_PIT_ISR_function. The vector number of the ISR associated to PIT1 is 60 (according to Table 26-9 p 942). You want to give a priority level equal to 2 to the PIT1 ISR. In order to enable the PIT interrupt, you have to proceed as following:

1. Initialize PIT1 and enable interrupt
2. Interrupt handler for the PIT1 ISR: INTC_InstallINTCInterruptHandler (My_PIT_ISR_function,60,2);
3. Enable maskable interrupt in the MCU: enableIrq();
4. In the function My_PIT_ISR_function, you have to clear the flag associated to PIT1 ISR.

4. Configuring software triggered interrupt

Use the same procedure as HW triggered interrupt to configure SW triggered interrupt.
The only difference relies in the triggering of software interrupt. Hardware interrupt is triggered by a hardware event (external event, time-out of a timer…). A software interrupt is triggered by a program request.
The registers SSCIR[i], i = 0..7, of the INTC modules support the setting or the clearing of software configurable ISR. A couple of 2 bits : SETi/CLRi sets or clear each software ISR. Writing a '1' to SET set the flag bit CLR to '1'.Writing a '0' has no effect. If the CLR bit is set to '1' it indicates that an ISR is pending, like any other flag bit. The flag CLR is cleared by writing a '1'.



Offset: 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VIII - Periodic interrupt Timer (PIT)

Refer to Chapter 32 – Periodic Interrupt Timer of the reference manual MPC5645SRM.pdf for the configuration of timer.
The MCU MPC5645S proposes several timer peripherals dedicated to different uses:

- System Timer Module (STM): it contains a 32 bit running-up counters clocked by the MCU system clock and four 32 bit compare channels with individual interrupts. This block is dedicated to the measurement of code execution time (number of clock cycles).

- Periodic Interrupt Timer (PIT): programmable timers for general purpose time measurements

- Real Time Clock / Autonomous Periodic Interrupt (RTC / API): the RTC is a free counter independent of the operation mode (run or low power mode) used to measure predefined time interval. The RTC contains a 32 bit counter driven either by SIRC, SWOSC or FIRC internal oscillators (see chapter III of this document, Clock Generation Description). It also contains a 10 bit compare channel, able to produce periodic interrupts (API block). The main interest of the API block is to generate periodic wakeup requests to exit from low power mode, or periodic interrupt requests.

- Software Watchdog Time (SWT): it contains a 32 bit timer used to prevent from system lock-up when the software is trapped in a loop or a bus transaction failed.

The operation of these peripherals in the MPC5645S is similar to that of the MPC5604B. So you can refer to the document presenting the MPC5604B to have the basic configuration of these different timers.

IX - FlexCAN module

Refer to Chapter 20 – FlexCAN of the reference manual MPC5645SRM.pdf for its configuration. The FlexCAN module is an integrated CAN controller. This the same peripheral in both MPC5604B and MPC5645S microcontrollers. So you can refer to the document presenting the MPC5604B for more information about the principles and the configuration of the FlexCAN module.

X - DSPI module

Refer to Chapter 10 – Deserial Serial Peripheral Interface of the reference manual MPC5645SRM.pdf for the configuration of DSPI module. The DSPI module is an integrated SPI controller. This the same peripheral in both MPC5604B and MPC5645S microcontrollers. So you can refer to the document presenting the MPC5604B for more information about the principles and the configuration of the DSPI module.

XI - Display Control Unit (DCU3)

Refer to Chapter 11 – Display Control Unit of the reference manual MPC5645SRM.pdf for the configuration of DCU module. This peripheral ensures the control of a TFT LCD panel and the preparation of graphical contents to display stored in internal or external memory. The DCU3 supports various panel sizes, various configuration of interface signals, different color encoding formats for memory usage optimization. The DCU can also display real-time video from an external video source (Parallel Data Interface PDI).
This chapter aims only at describing the main features of this module, its operation principle, how to configure it to manage a TFT-LCD panel, and the basic operation to construct a graphical content. This chapter does not explain in detail how to fetch the graphical content to the different sources, how to decompose a picture in different layers and how to select the best color encoding format to optimize the memory usage.

1. Principles

Figure 7 describes the functional block diagram of DCU3, from graphic or video contents to the signal command to the TFT-LCD panel. The characteristics (timing, polarity of signals…) of the command signal are configured according to the properties of the LCD panel (width, height, refresh frequency…). The role of the DCU3 is to calculate the relevant graphical content for each pixel and display it. The different operations are:

- fetch the source graphics from memory using its internal DMA channels (CH1 to CH4)
- convert the graphic value of each fetched pixel into full quality color format (RGB888), the source graphic can be encoded in various format or according to a Color Look-up Table (CLUT)
- calculate the required pixel value by blending the values of up to four separate graphic layers, with transparency options
- perform a gamma correction on the pixel value if required
- send the pixel value to the TFT LCD display over its data bus
- set flags to indicate end of frame, buffer threshold, and other status changes

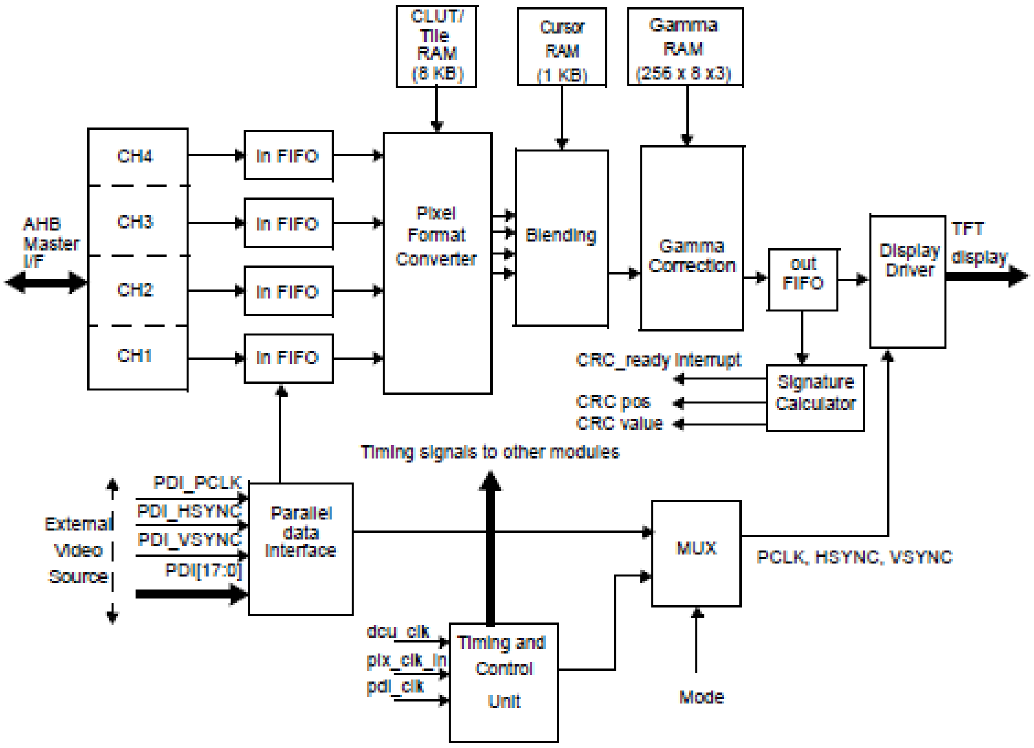


Figure 7 – DCU3 functional block diagram (MPC5645SRM.pdf - p. 337 – Fig. 11-1)

Figure 8 presents the different layers that form the graphical content. The different layers are configured by the user through the register of the DCU3. The graphical content to display is composed of three types of layers:

- 16 graphic layers for the creation of the graphical content, which support different color encoding formats.
- 1 default background layer
- 1 configurable cursor layer with blinking option
- Configurable color look-up tables (CLUT) for user-defined color encoding

These different layers are blended by the DCU3. Various types of blending are proposed and can be configured through the register of the DCU3. The different blending possibilities will not be described in this document. Refer to the MPC5645SRM.pdf reference manual for more information.

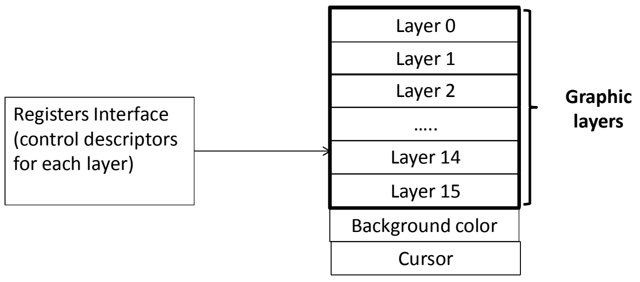


Figure 8 – Decomposition of a graphical content into layers configurable by the user (MPC5645SRM.pdf - p. 337 – Fig. 11-1)

Each layer contains the following information, configured by the registers CTRLDESCL1 to CTRLDESCL7 defined for each layers:

- horizontal and vertical size and position of the graphic
- address of the graphic in memory
- color encoding format and color palettes (if required)

- type and depth of blending
- range of colors identified for chroma blending
- tile size (a tile is a graphic that is repeated horizontally and vertically to fill completely a layer)

2. DCU interface signals

Figure 9 presents the external signals of the DCU module. Here, the PDI signals are not described as they are not used.

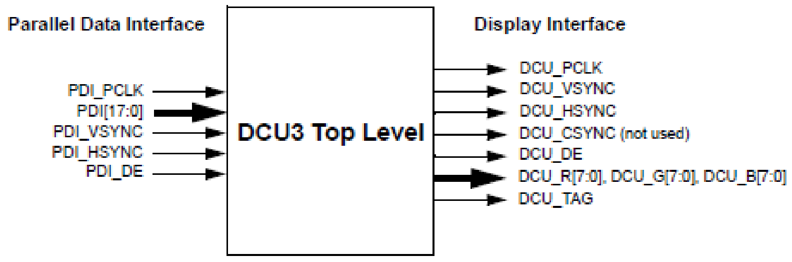


Figure 9 – External signals of the DCU (MPC5645SRM.pdf - p. 339 – Fig. 11-2)

The pixel are encoded in a RGB888 format, i.e. 8 bits are used for red, green and blue. The signals DCU_R[7:0], DCU_G[7:0] and DCU_B[7:0] contains pixel data in parallel mode. The transfer of pixel data and pixel display are synchronized by the clock DCU_PCLK. DCU_HSYNC is the horizontal sync signal which indicates by a pulse the beginning of a new line (the number of pixels per line is given by the width of the LCD panel). DCU_VSYNC is the vertical sync signal which indicates by a pulse the beginning of a new frame, i.e. all the lines have been displayed (the number of lines per frame is given by the height of the LCD panel). The signal DCU_DE means Data Enable and is set to '1' during the display of one line (between two successive DCU_HSYNC pulses). Figure 10 gives a timing diagram of these signals.

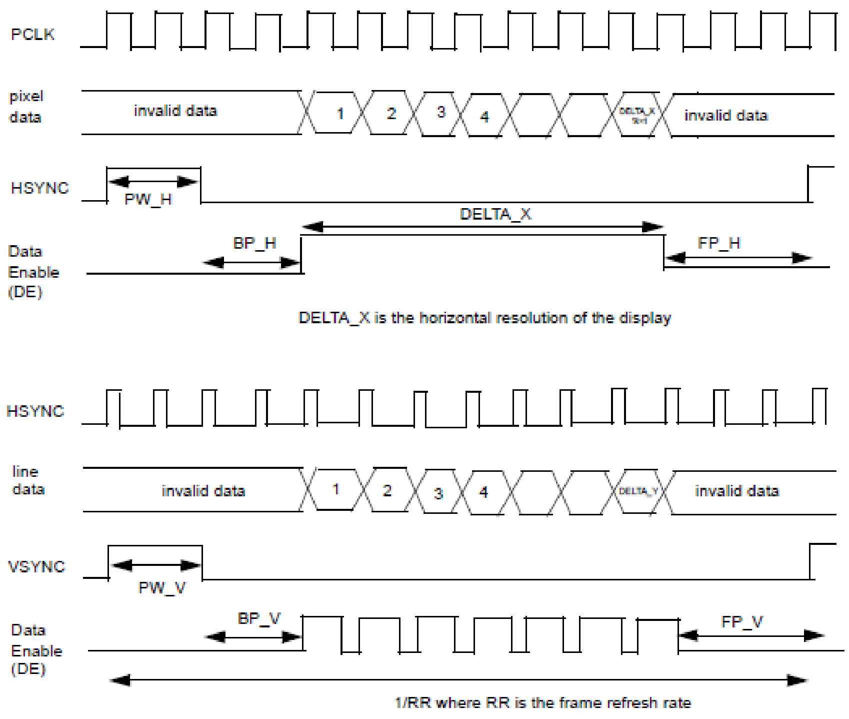
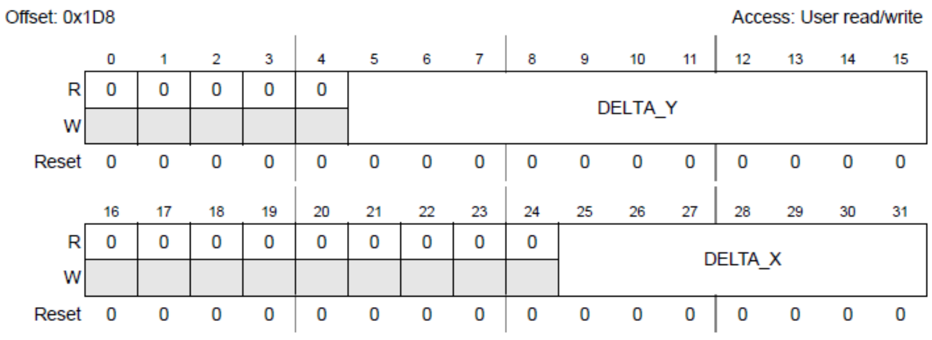


Figure 10 – Pixel data, PCLK, HYNC and VSYNC timing diagram (MPC5645SRM.pdf - p. 412 – Fig. 11-63)

3. Configuration of the interface signals with TFT-LCD panel

First, the pads associated to the interface signals of the DCU3 have to be configured properly. The alternative function 1 (for DCU) has to be selected, through the PCR registers (see VI - SIUL). Don't forget the additional control signals for LCD panel (activation of its power supply, backlight…).
The DCU generates the control signals PCLK, HSYNC, VSYNC, DE emitted in parallel to the pixel data. The characteristics of these signals must comply with the properties of the LCD panel. The following registers are dedicated to the configuration of the interface signals.
The register DISP_SIZE defines the horizontal and vertical pixel resolutions of the LCD panel. DELTA_Y defines the vertical resolution in pixel number. DELTA_X gives the horizontal resolution indirectly: the figure given by DELTA_X must be a multiple of 16. For example, for a 480 x 272 pixels panel, DELTA_Y = 272, while DELTA_X = 30.


Offset: 0x1D8 Access: User read/write

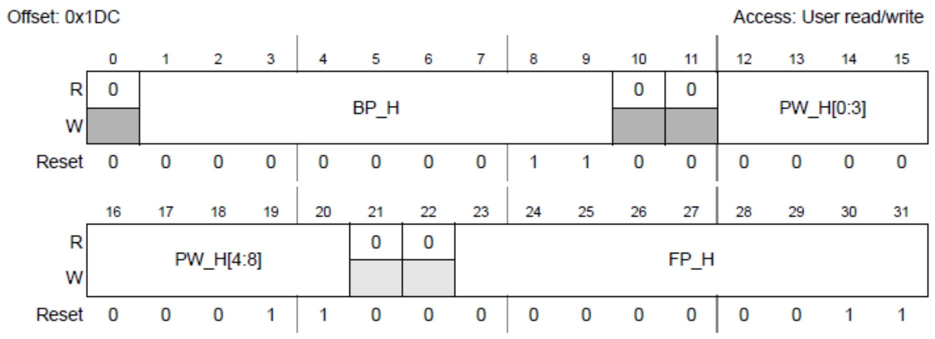
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DELTA_Y										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

The timing parameters of the pixel data transfer must be set properly. They depend on the width and height of the LCD panel, but also on the frame refreshing frequency (in practice between 50 and 60 Hz). For example, the pulse emitted on VSYNC is emitted at each end of frame, while HSYNC at the end of each line of the panel. The first operation is to define the frequency of the pixel clock. It derives from the auxiliary clock 0 (see III. Clock generation description). The register DIV_RATIO defines the division ratio between the auxiliary clock 0 and the pixel clock. The DCU3 clock is equal to the auxiliary clock 0. To divide the auxiliary clock by N, set the register DIV_RATIO to (N-1).
The ratio between the DCU3 clock and the pixel clock must be also selected according to the number of layer to blend. Blending the pixels from several graphic layers consumes several DCU3 clock periods. This number increases with the number of layers to blend (blend stack depth). For example:

- for one or two pixel blending, the minimum DCU3 clock is the same as the pixel clock
- for three pixel blending, the minimum DCU3 clock is twice the pixel clock
- for four pixel blending, the minimum DCU3 clock is three times the pixel clock

The timing characteristics of the HSYNC signal are controlled with the register HSYN_PARA (see Figure 10 for more details about the timing parameters of HSYNC). The PW_H sets the duration of the HSYNC pulse in number of pixel clock periods. The fields BP_H and FP_H define the back-porch pulse width and front-porch pulse width respectively, i.e. the time between the HSYNC pulse and both edges of DE signal. They are given in pixel clock periods.

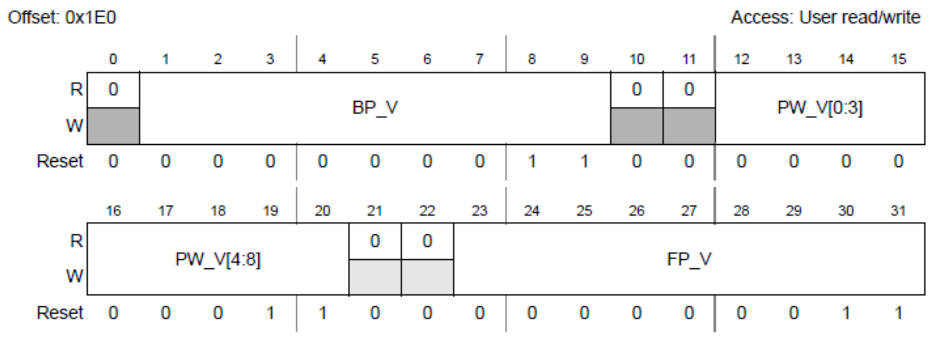


Offset: 0x1DC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	BP_H									0	0	PW_H[0:3]			
W	▨										▨	▨				
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PW_H[4:8]					0	0	FP_H								
W																
Reset	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1

The timing characteristics of the VSYNC signal are controlled with the register VSYN_PARA (see Figure 10 for more details about the timing parameters of VSYNC). The PW_V sets the duration of the VSYNC pulse in number of horizontal line cycles. The fields BP_V and FP_V define the back-porch pulse width and front-porch pulse width respectively, i.e. the time between the VSYNC pulse and both edges of DE signal. They are given in number of horizontal line cycles.



Offset: 0x1E0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	BP_V									0	0	PW_V[0:3]			
W	▨										▨	▨				
Reset	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PW_V[4:8]					0	0	FP_V								
W																
Reset	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1

The register SYN_POL defines the polarity of the interface signal.

Remark: Bypassing the Timing Controller (TCON)

The microcontroller embeds a Timing Controller (TCON) for raw TFT-LCD panels which embeds no TCON. In this situation, the TCON provides directly panel the control signal in RSDS signaling format (Reduced Swing Differential Signal). The TFT-LCD panel used in the instrument cluster of our lab integrates a TCON, so the TCON peripheral of the MPC5645S is useless. However, it is necessary to bypass it, otherwise it will modify the command signal provided by the DCU (HSYN, VSYN and DE are altered). To bypass the TCON, the following procedure must be set:

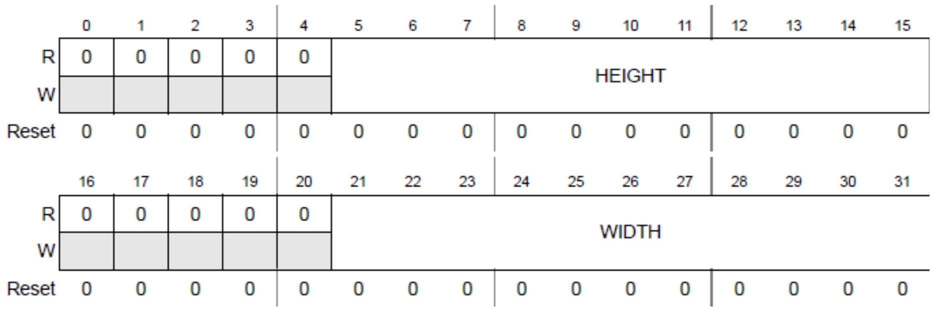
- disable the TCON by writing a '0' to the bit TCON_EN of the register CTRL1 of the TCON peripheral
- bypass the TCON by writing a '1' to the bit TCON_BYPASS of the register CTRL1 of the TCON peripheral

4. Configuration of the graphical layers

The DCU module proposes to construct a graphic from a maximum of 16 layers that can be blended together. Actually, it is only possible to blend 4 superimposed layers. If more than 4 layers are superimposed, only the 4 layers with the highest priority are blended, the others are ignored (Layer0 has the highest priority). Obviously, all the graphical layers are placed above the background of the TFT-LCD panel.

For each graphical layers, 7 registers noted CTRLDESCL1 to CTRLDESCL7 define their properties. Below, a brief description of some of them is proposed. CTRLDESCL1 defines the width and height of the layer in number of pixels. If the width or height of the layer exceeds the actual size of the LCD panel (given by the register DISP_SIZE), the pixels outside the screen will be ignored. As explained in 11.4.4.3 in the MPC5645S reference manual, HEIGHT field can take any values, but there is restriction for WIDTH field depending on the data format of the graphic specified by the layer. This field must always be an integer multiple of the number of pixels that are represented by a 32-bit word except in the special case of 1 bit per pixel where the multiple is 16. Figure 11 gives the WIDTH multiple values according to the encoding format. By default, if RGBA8888 is used (8 bits for red, green, blue and transparency alpha), 32 bits are used to encode one pixel, so the field WIDTH gives directly the width of the layer in pixel number.

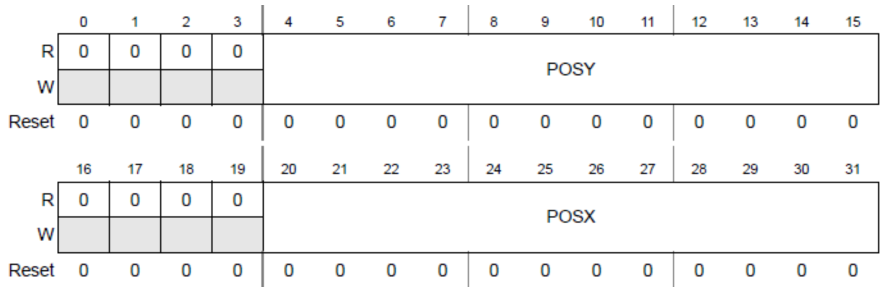
Remark: the transparency is encoded according to a value noted Alpha. If it is coded on 8 bits, this value varies between 0 (fully transparent) to 255 (fully opaque).



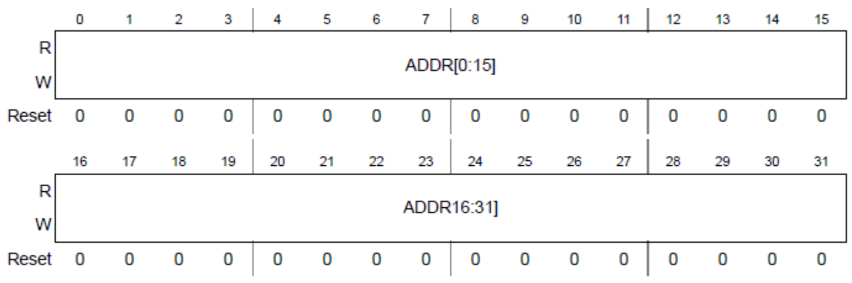
Data format	WIDTH multiples	Example values
1 bpp	16	16, 32, 48, 64, ...
2 bpp	16	16, 32, 48, 64, ...
4 bpp	8	8, 16, 24, 32, ...
8 bpp	4	4, 8, 12, 16, ...
16 bpp	2	2, 4, 6, 8, ...
24 bpp	4 (= 3 whole 32-bit words)	4, 8, 12, 16
32 bpp	1	1, 2, 3, 4, ...
YCbCr422	4	4, 8, 12, 16, ...

Figure 11 – WIDTH multiple values according to the encoding format (MPC5645SRM.pdf - p. 418 – Table 11-59)

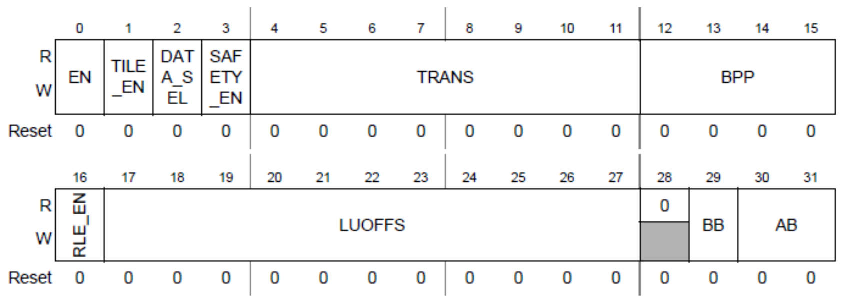
CTRLDESCL2 defines the horizontal and vertical position of the origin of the layer in number of pixels. The origin is the top-left point of the layer. All the pixels outside the screen are ignored.



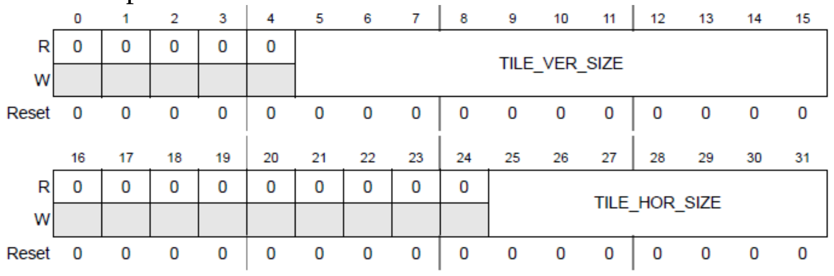
CTRLDESCL3 gives the memory address of the beginning of the layer data (first pixel). Data can be stored in internal Flash, internal SRAM, internal Graphic SRAM, or an external Flash memory, through the serial Quad SPI interface. The memory map of the microcontroller can be found in Table 2-1 p. 87 of the MPC5645S reference manual. Be careful, the memory address of the beginning of the layer must be a multiple of 64 (64 bit aligned).



CTRLDESCL4 gives various graphic options for each layer. EN bit enables the layer. This bit must be set to '1' to be displayed and blended with the other layers. Tile mode is enabled by setting the bit TILE_EN. The Tile data can be either in memory or in CLUT, according to the bit DATA_SEL. The field TRANS can defined a global transparency level (alpha value) for the layer. The field BPP gives the encoding format in terms of bits per pixel (bpp). For example, with BPP = '0110', the encoding format is RGBA8888. LUOFFS field defines the offset of CLUT or Tile associated to the layer in the CLUT/TILE RAM (see Table 11-2 p 340 for the memory address). The field AB defines the Alpha blending, i.e how the transparency levels of superimposed graphical layers are managed.

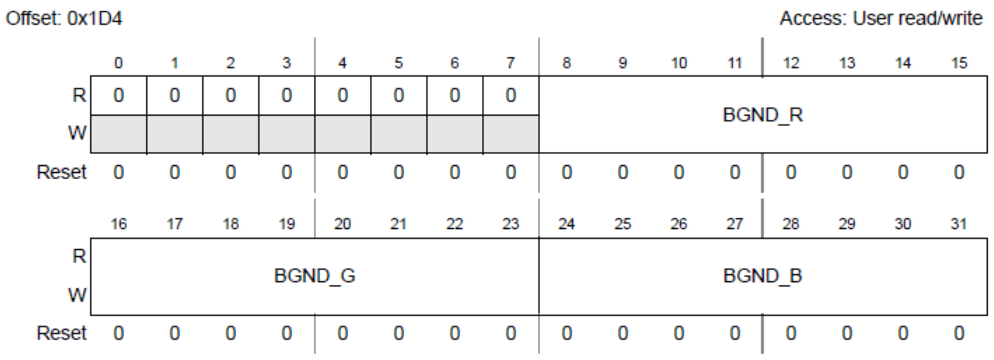


CTRLDESCL5 and CTRLDESC6 deal with Chroma keying, but they are not detailed in this document. CTRLDESCL7 defines the horizontal and vertical size of the Tiles that compose the graphic layer. TILE_VER_SIZE gives the height in pixels, TILE_HOR_SIZE gives the width in multiples of 16 pixels.



5. Configuration of the Background and cursor layers

The background color of the TFT-LCD panel is set by the register BGND. Three fields of 8 bits defines its color in RGB888 format. Obviously, no transparency is associated to the background.


Offset: 0x1D4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	BGND_R							
W	▨	▨	▨	▨	▨	▨	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

A hardware cursor with blinking option can be superimposed on all the other graphic layers. Its properties are configured with registers CTRLDESCCURSOR1 to CTRLDESCCURSOR4. The graphic associated to the cursor is defined by the user and stored in a given area of the memory: the cursor RAM (see Table 11-2 p 340 for the cursor RAM address). The register CTRLDESCCURSOR1 defines the size of the rectangular surface occupied by the cursor. Two fields (HEIGHT and WIDTH) gives the height and width of the cursor in pixel numbers. However, there are some restrictions on the size: the height is limited to 256 pixels and the total number of pixel must not exceed the number of bits of the cursor RAM (its size is limited to 8192 bits). CTRLDESCCURSOR2 configures the horizontal and vertical positions of the cursor. CTRLDESCCURSOR3 enables the cursor and sets its color. CTRLDESCCURSOR4 enables the blinking option and sets the period of blinking in term of number of frame periods.

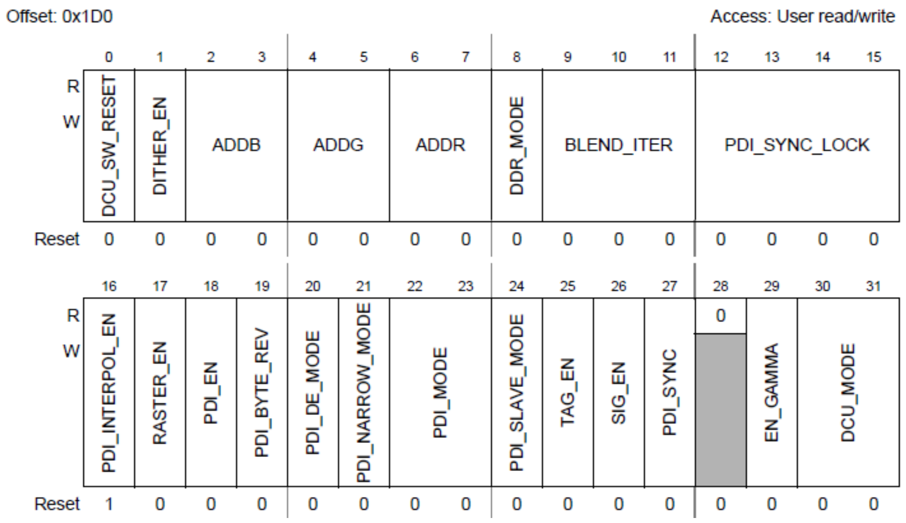
6. Color Look-Up Table (CLUT)

The RAM memory of the microcontroller embeds a short portion dedicated to the definition of a color look-up table (CLUT) (see Table 11-2 p 340 for the its address). 32-bit word coding colors in ARGB8888 format (8 least significant bit for the transparency, followed by 8 bit for the red, 8 bit for the green, and 8 bit for the blue) can be defined and stored by the user. These colors form a table that can be used for the rendering of each graphical layers.
The main interest of the CLUT is to reduce the required size of data stored in memory to encode image. All the picture display on the TFT screen are in RGB888 format. However, it is not necessary to encode data with such a format when only a small amount of colors are used in the picture. The image can be encoded with a smaller amount of bit, which codes an entry of the CLUT, where a color in ARGB8888 format is stored. For each image, the used colors are organized in the CLUT from an offset address, which is defined in the field LUOFFS of CTRLDESC4.

7. Configuration of the operating mode

The DCU3 supports several operating modes, which are activated by the field DCU_MODE of the register DCU_MODE:

- by default, the DCU is in Off mode
- the normal mode, where the different configured layers are blended and displayed
- the color bar mode, which is a test mode (Figure 12)



Offset: 0x1D0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DCU_SW_RESET	DITHER_EN	ADDB		ADDG		ADDR		DDR_MODE	BLEND_ITER			PDI_SYNC_LOCK			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PDI_INTERPOL_EN	RASTER_EN	PDI_EN	PDI_BYTE_REV	PDI_DE_MODE	PDI_NARROW_MODE	PDI_MODE		PDI_SLAVE_MODE	TAG_EN	SIG_EN	PDI_SYNC	0	EN_GAMMA	DCU_MODE	
W													▨			
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is also used to configure several special modes, the PDI operating mode. The bit RASTER_EN must be set to activate the scanning of pixel data, the transfer of pixel data to LCD panel with the other interface signals (PCLK, HSYNC, VSYNC, DE).

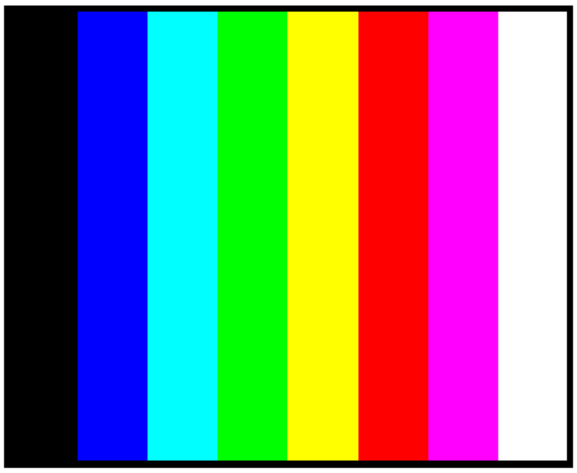
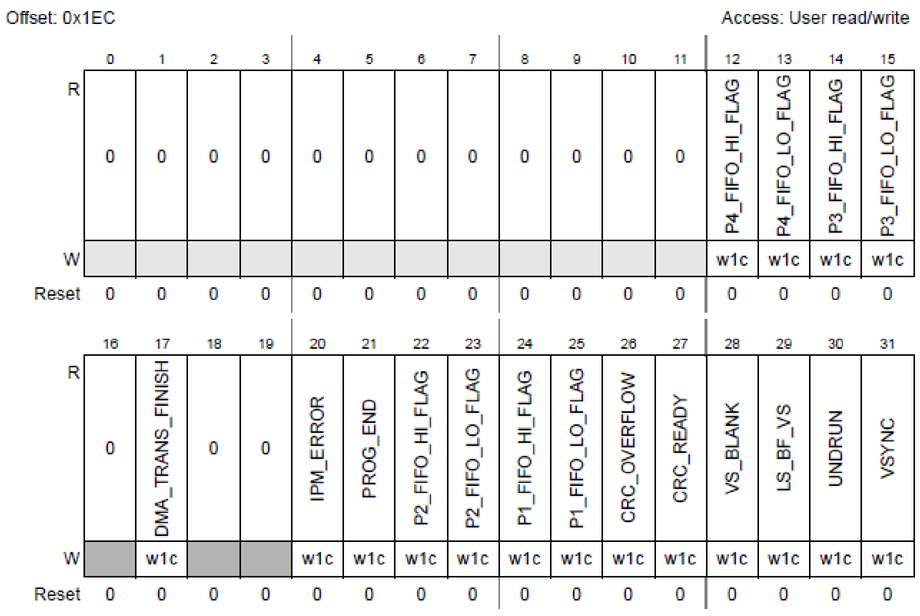


Figure 12 – Example of display in Color bar mode

8. *Timing management*

Changes in the layer configuration can be done at any time by the CPU, independently of the data fetch from memory. In some case, it can produce an incoherent display on the panel. The configuration defined one HSYNC before the end of the vertical blanking period is the configuration used by the DCU3 for the panel refresh phase. Therefore, the DCU3 configuration is completely open during the vertical blanking period. In contrary, control descriptors and some other registers may also be programmed at any time.

To prevent this situation, the DUC3 proposes five timing control flags in order to manage the changes control descriptors, CLUT, tile memory or source graphics. These flags are given in the register INT_STATUS.



Offset: 0x1EC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	P4_FIFO_HI_FLAG	P4_FIFO_LO_FLAG	P3_FIFO_HI_FLAG	P3_FIFO_LO_FLAG
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	DMA_TRANS_FINISH	0	0	IPM_ERROR	PROG_END	P2_FIFO_HI_FLAG	P2_FIFO_LO_FLAG	P1_FIFO_HI_FLAG	P1_FIFO_LO_FLAG	CRC_OVERFLOW	CRC_READY	VS_BLANK	LS_BF_VS	UNDRUN	VSYNC
W		w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The VS_BLANK and LS_BF_VS flags give indication of the start of the vertical blanking period. The VS_BLANK flag is set at the beginning of the vertical blanking period. The LS_BF_VS flag is set a given number of horizontal lines before the start of the vertical blanking period; the given number of lines is defined by the LS_BF_VS bit field in the THRESHOLD register.
The PROG_END flag indicates that the DCU3 has locked the contents of its configuration registers for the new panel refresh period. No further changes are accepted to the DCU3 configuration after this flag is set (until the next vertical blanking period).
The DMA_TRANS_FINISH flag indicates that the DCU3 has completed fetching all data from memory in the current panel refresh cycle. This normally precedes the vertical blanking period and indicates that it is possible to change the contents of a memory that contains graphics used by the DCU3. The VSYNC flag indicates that the DCU3 has begun the next panel refresh period.

The DCU3 uses input and output FIFOs to store incoming data from DMA and data to be displayed. The 4 input FIFOs are not accessible for the user, so that it can become full or empty. However, high and low thresholds can be defined to detect if the FIFOs are nearly full or empty. These thresholds are defined by the register THRESHOLD_INPUT_BUF_1/2. Four couples of flags are associated to each input FIFO to indicate if the FIFO has reached its upper or lower threshold : Pn_FIFO_HI_FLAG and Pn_FIFO_LO_FLAG, with n = 1, 2, 3 or 4.

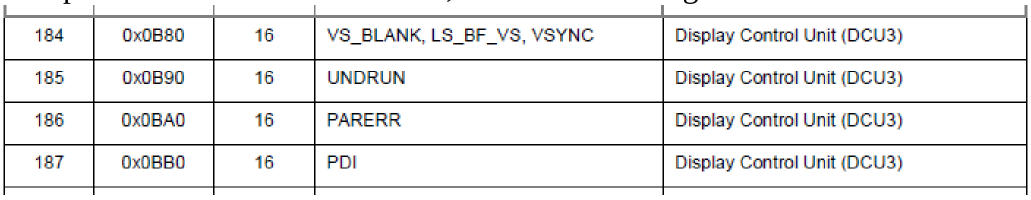
9. Error detection and interrupt generation

The DCU3 asserts error flags when:

- errors are detected in its configuration
- the user attempts to modify the configuration at an invalid point in the panel refresh period
- the DCU3 is unable to access the required source data.

An interrupt is triggered if enabled in the corresponding mask register (INT_MASK register). The flags are registered in PARR_ERR_STATUS and INT_STATUS registers. The first register collects the errors due to the DCU3 configuration and its graphical layers. The second register indicates errors when the DCU3 is unable to access its required source data.

Four interrupt are associated to the DCU3, with the following ISR number:



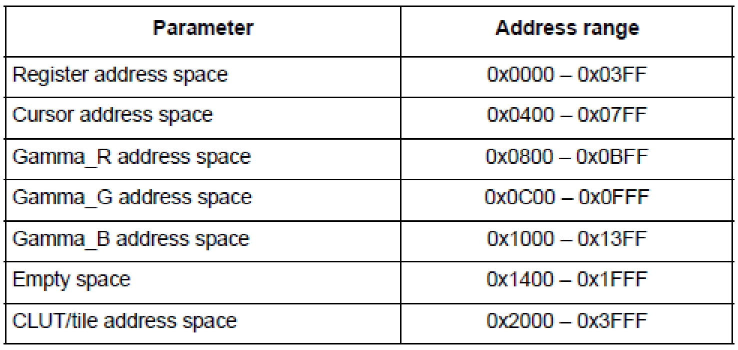
184	0x0B80	16	VS_BLANK, LS_BF_VS, VSYNC	Display Control Unit (DCU3)
185	0x0B90	16	UNDRUN	Display Control Unit (DCU3)
186	0x0BA0	16	PARERR	Display Control Unit (DCU3)
187	0x0BB0	16	PDI	Display Control Unit (DCU3)

Interrupt lines	184. Timing based interrupts	185. Functional interrupts	186. Parameter error interrupts	187. PDI related interrupt
Associated flags	VSYNC LS_BF_VS VS_BLANK PROG_END DMA_TRANS_FI NISH	UNDRUN CRC_READY CRC_OVERFLOW P1_FIFO_HI_FLAG P1_FIFO_LO_FLAG P2_FIFO_LO_FLAG P2_FIFO_HI_FLAG P3_FIFO_HI_FLAG P3_FIFO_LOW_FLAG P4_FIFO_HI_FLAG P4_FIFO_LOW_FLAG IPM_ERROR	Layer Error Signature Calculator Error Display Error HWC_error RLE error	Not described here

10. *Image storage and encoding*

Several memory spaces are available to store the pixel contents of images to be displayed. They can be stored either in on-chip Flash memory, on-chip RAM memory or external Flash memory (the MPC5645S supports interface with external QuadSPI Flash memory). The address of the location of these different memories can be found in Table 2-1 p 87. **When you write data in memory, ensure that you write at a correct location !** Any write in an incorrect location (e.g. in Code Flash or restricted area may lead to a wrong operation of the microcontroller. The DCU includes also some RAM blocks to store some graphical elements, such as the cursor and the CLUT. Table 11-2 p 340 provides the memory location of the hardware cursor and the CLUT. The base address of DCU registers can be found in Table 2-1 p. 87 and is 0xFFE5C000.

Table 11-2. DCU3 memory map



Parameter	Address range
Register address space	0x0000 – 0x03FF
Cursor address space	0x0400 – 0x07FF
Gamma_R address space	0x0800 – 0x0BFF
Gamma_G address space	0x0C00 – 0x0FFF
Gamma_B address space	0x1000 – 0x13FF
Empty space	0x1400 – 0x1FFF
CLUT/tile address space	0x2000 – 0x3FFF

The way the image will be displayed depends on the data encoding of the octets stored in memory and passed to the DCU. This data encoding is defined by the user. As explained in part XI.4, the DCU supports various data encoding. The use of a data format involves various constraints in the memory management. Read carefully the datasheet to ensure that these constraints are observed. In p. 420 of the datasheet, from Table 11-60 to Table 11-70, the storage format for each data encoding is provided. As example, the following figures present the storage format for data in BGRA8888 and & bit per pixel encoding. In BGRA format, each pixel is coded by a 32 bit word stored in memory (e.g. $B_0R_0G_0A_0$), where the most significant octet codes the blue component of the pixel and the least significant octet codes the transparency (alpha component).
In 1 bpp format or monochrome image format, a 32-bit word stores the state of 32 pixels. The two colors that the pixel can take are given by two entries in a color palette stored in the CLUT, as defined by CTRLDESCL4 register of the considered graphical layer. As described by the figure below (Table 11-70), a 32-bit word describes the status of 32 adjacent pixels organized in a row, where the most significant octet [0:7] describes the 8 leftmost pixels (pixel 0 to pixel 7), while the least significant octet [24:31] describes the 8 rightmost pixels (pixel 24 to pixel 31). However, in each octet, the LSB describes the rightmost pixel while the MSB describes the rightmost pixel.

Table 11-60. Data layout for BGRA8888

Address offset	[0:7]	[8:15]	[16:23]	[24:31]	[0:7]	[8:15]	[16:23]	[24:31]
0x00	B0	G0	R0	A0	B1	G1	R1	A1
0x08	B2	G2	R2	A2	B3	G3	R3	A3

Table 11-70. Data layout for 1 bpp

Address offset	[0:7]	[8:15]	[16:23]	[24:31]	[0:7]	[8:15]	[16:23]	[24:31]
0x00	pixel7-pixel0	pixel15-pixel 8	pixel23-pixel 16	pixel31-pixel 24	pixel39-pixel 32	pixel47-pixel 40	pixel55-pixel 48	pixel63-pixel 56
0x08	pixel71-pixel 64	pixel79-pixel 72	pixel87-pixel 80	pixel95-pixel 88	pixel103-pix el96	pixel11-pixel 104	pixel119-pix el112	pixel127-pix el120

11. General procedure to display an image

The following steps describe the typical initialization procedure of the DCU3:

1. After a reset configure the DCU3 peripheral to be active using the mode entry module and configure the DCU3 clock source (auxiliary clock 0) in the MC_CGM
2. If using a panel with an integrated TCON module, disable the TCON signals by setting the TCON_BYPASS bit in the TCON CTRL1 register. Due to the configuration of the TCON module, the DCU3 pixel clock signal will be output as soon as it is selected by the SIUL PCR. This is independent of the DCU3 operating mode.
3. Configure the output ports in the SIUL as required.
4. Configure the timing registers to match the TFT LCD panel requirements
5. Set the background color
6. If necessary, load the initial tile or palette colors into the CLUT/Tile memory
7. Configure the control descriptors for the layers and cursor that are to be used initially

8. Enable the DCU3 in the appropriate mode (DCU_MODE and RASTER_EN bit fields).

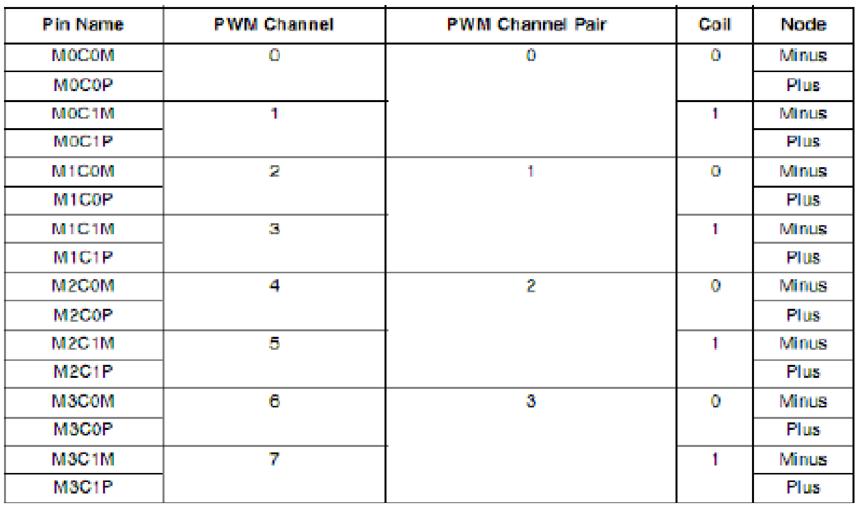
Before enabling the display, the graphical contents have to be stored in on-chip or off-chip memory. The role of the program that manages a LCD screen consists in pointing out the memory address of each graphical layer (through CTRLDESC3 register), modifying the color, the appearance, the position of each graphical layer (through the CTRLDESCX register), or transforming the graphical content stored in memory. Ensure that the graphical content is ready before displaying it.

XII - Stepper Motor Controller (SMC)

Refer to Chapter 41 – Stepper motor controller (SMC) of the reference manual MPC5645SRM.pdf for the configuration of DCU module. You can also refer to the NXP Application note AN4037 - Driving a Stepper Motor using the MPC56xxS SMC Module for more precise information about control of stepper motors with SMC.

The SMC of the microcontrollers used in this lab is not functional, so it is recommended to command the stepper motors of instrument gauges directly with the I/O pads and the PIT timer for synchronization.

The SMC block is a PWM motor controller suitable for driving small stepper motors, which are equivalent to inductive coil. The microcontroller can control up to 4 motors. 16 external pins are associated to the SMC and 8 PWM channels. Each stepper motor is made of 2 coils and has two differential terminals called Plus (P) and Minus (M). The microcontroller pins and PWM channels are resumed in the following table. A H-bridge is associated to each PWM channel pair in order to change the polarity of the current in the coil of the stepper motor. PWM output on M0C0M results in a positive current flow through coil 0 when M0C0P is driven to a logic high state. PWM output on M0C1M results in a positive current flow through coil 1 when M0C1P is driven to a logic high state



Pin Name	PWM Channel	PWM Channel Pair	Coil	Node
M0C0M	0	0	0	Minus
M0C0P				Plus
M0C1M	1		1	Minus
M0C1P				Plus
M1C0M	2	1	0	Minus
M1C0P				Plus
M1C1M	3		1	Minus
M1C1P				Plus
M2C0M	4	2	0	Minus
M2C0P				Plus
M2C1M	5		1	Minus
M2C1P				Plus
M3C0M	6	3	0	Minus
M3C0P				Plus
M3C1M	7		1	Minus
M3C1P				Plus