# DATA SHEET

# SAA6713AH
## XGA analog input flat panel controller

Product specification
Supersedes data of 2002 Jul 16

2004 Apr 05

Philips
Semiconductors

PHILIPS

CONTENTS

## 1 FEATURES

- Integrated triple Analog-to-Digital Converter (ADC) for RGB analog sampling up to 110 MHz
- Integrated PLL for dot clock recovery
- Integrated composite sync slicer
- Integrated sync-on-green separation
- Support of Super Extended Graphics Adapter (SXGA) input mode
- Independent horizontal and vertical arbitrary ratio up and downscaling
- Video mode detection
- Auto-adjustment support for sampling phase and frequency, picture alignment and colour alignment
- Advanced colour adjustment
- Integrated On Screen Display (OSD) controller with predefined and programmable font and bit-mapped graphics, as well as a hardware overlay cursor
- 10-bit gamma correction
- Support for 6-bit and 8-bit panels by temporal dithering
- Freely programmable output timing supports displays of virtually any manufacturer
- Directly interfaces row and column drivers (TCON), versatile timing generation
- Programmable output pin ordering
- Adjustable output pin ordering
- High-speed I$^2$C-bus interface up to 3.4 Mbits/s
- Event driven interrupt generation for easy interfacing with microcontroller software.

## 2 GENERAL DESCRIPTION

The SAA6713AH is a single input single-chip Thin Film Transistor (TFT) display controller IC with analog VGA standard input capabilities. Additionally, the SAA6713AH includes a wide range of functions for processing and the measurement of incoming RGB data according to the requirements of an XGA TFT display.

Covered functions are accurate measurements for the horizontal and vertical input frequencies to determine the incoming video mode and advanced auto-adjustment features that provide all data for a fast and accurate adjustment of frequency, phase and gain settings. The unit is able to generate interrupts for easy interfacing with a system microcontroller with separately maskable interrupt conditions.

The input section handles incoming data up to SXGA resolution that can be downscaled individually in width and height to fit to the connected panel resolution. Independent horizontal and vertical upscaling with enhanced programmable filter possibilities provides the IC's core functionality of high-quality scaling. Picture quality is further supported by an enhanced colour management including a 10-bit gamma correction function. A sophisticated dithering unit allows the use of low-end 6-bit panels while keeping up the high quality image impression.

An advanced OSD generator is integrated with a fixed 12 × 18 ROM font consisting of 179 ANSI characters, 77 Japanese characters, 48 multicolour icons and 48 single colour icons. In addition to these fixed size characters another 112 different border characters can be generated in any desired font size between 8 × 8 and 32 × 32 pixels. Another 38 special characters are provided particularly for multicolour slider icons that can be parametrized in size and style. For higher flexibility of the OSD appearance a downloadable mixed multicolour or single colour font with any programmable character size between 8 × 8 to 32 × 32 pixels and up to four colours per character can be used and displayed together with the predefined ROM characters. A special bitmap organized graphical OSD with up to 16 individual colours allows to include graphic items like company logos, while a double buffered OSD cursor gives the ability to use animated pointers within an on screen menu. The panel timing interface can not only drive today's common timing controller based panel interfaces, but it has also the capability to directly drive the row and column drivers of a panel itself. An adjustable output pin ordering guarantees easy board layout with any type of panel connector.

The SAA6713AH represents a fully integrated single-chip solution for low-end monitors, offering both high quality scaling and an advanced OSD generator.

## 3 QUICK REFERENCE DATA

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| $V_{DDD(IC)}$ | digital supply voltage for internal core on pins $V_{DDD(IC1)}$ to $V_{DDD(IC9)}$ | | 2.3 | 2.5 | 2.7 | V |
| $V_{DDA}$ | analog supply voltage on pins $V_{DDA(R)}$, $V_{DDA(G)}$, $V_{DDA(B)}$, $V_{DDA(ADC)(R)}$, $V_{DDA(ADC)(G)}$ and $V_{DDA(ADC)(B)}$ | | 2.3 | 2.5 | 2.7 | V |
| $V_{DD(PLL)}$, $V_{DDD(PLL)}$, $V_{DDA(PLL)}$ | supply voltage for PLL on pins $V_{DD(PLL)(P)}$, $V_{DDD(PLL)(S)}$ and $V_{DDA(PLL)(S)}$ | | 2.3 | 2.5 | 2.7 | V |
| $V_{DDA(IB)}$ | analog supply voltage for input buffer on pin $V_{DDA(IB)}$ | | 2.7 | 3.0 | 3.3 | V |
| $V_{DDD(EP)}$ | external digital pad supply voltage for pins $V_{DDD(EP1)}$ to $V_{DDD(EP10)}$ | | 3.0 | 3.3 | 3.6 | V |
| $V_{DDA(EP)}$ | external analog pad supply voltage for pin $V_{DDA(EP)}$ | | 3.0 | 3.3 | 3.6 | V |
| $I_{DD(tot)}$ | total supply current | | – | 350 | – | mA |
| $V_i$ | input voltage | note 1 | LVTTL compatible | | | |
| $V_o$ | output voltage for TFT port | | CMOS compatible | | | |
| $T_{amb}$ | ambient temperature | | 0 | – | 70 | °C |

**Note**

1. Pins HSYNC, VSYNC, SDA and SCL are 5 V tolerant inputs.

## 4 ORDERING INFORMATION

| TYPE NUMBER | PACKAGE | | |
|---|---|---|---|
| | NAME | DESCRIPTION | VERSION |
| SAA6713AH/V1 | QFP160 | plastic quad flat package; 160 leads (lead length 1.6 mm); body 28 × 28 × 3.4 mm; high stand-off height | SOT322-2 |

## 5 BLOCK DIAGRAM

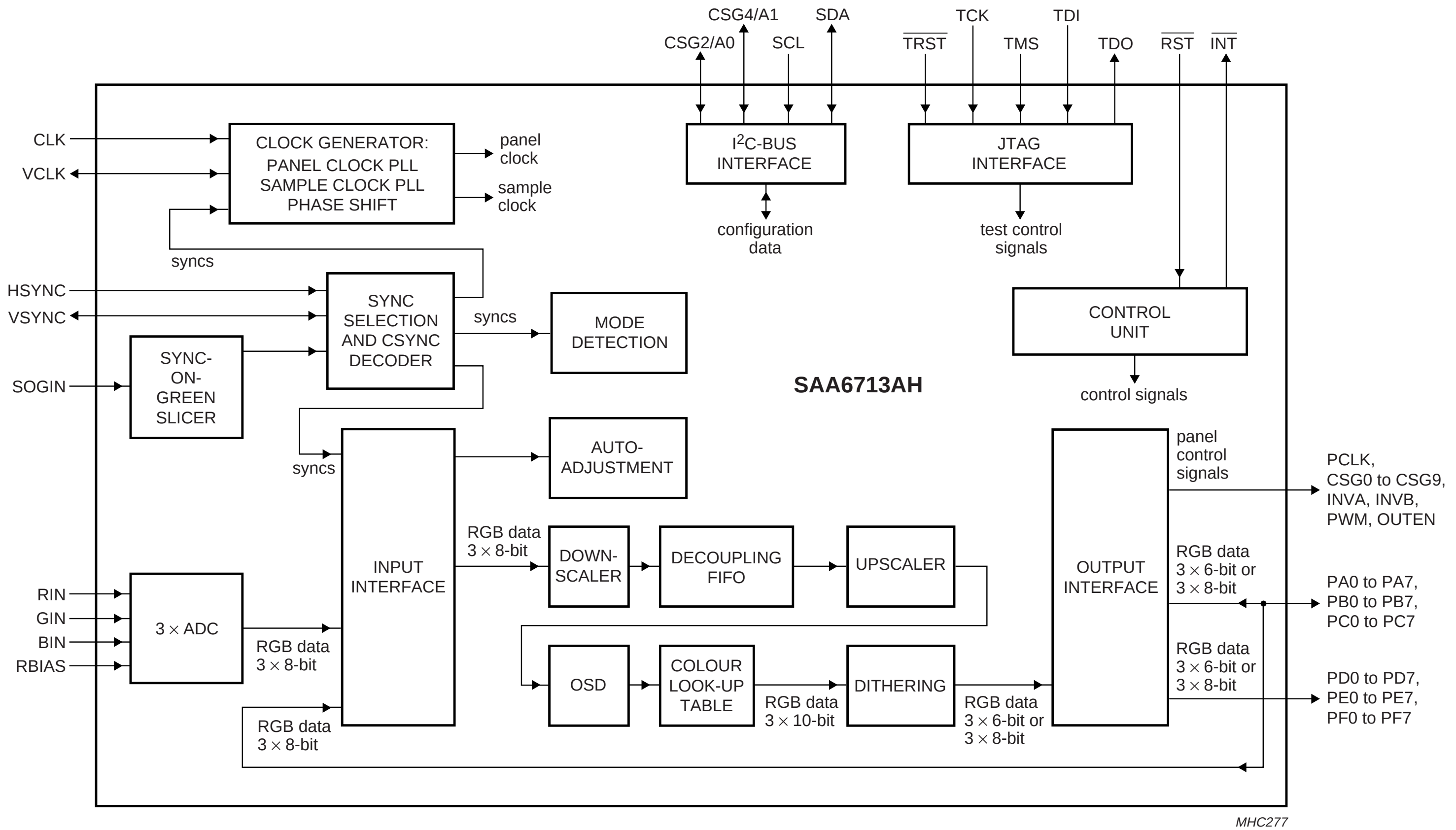



Fig.1 Block diagram.

## 6 PINNING

| SYMBOL | PIN[1] | TYPE | DESCRIPTION |
|---|---|---|---|
| $V_{SSA(BIAS)(B)}$ | 1 | – | analog ground for bias; blue channel |
| BIN | 2 | A | blue colour signal input |
| $V_{DDA(ADC)(B)}$ | 3 | – | analog supply voltage for ADC; blue channel (2.5 V) |
| REF_B | 4 | A | blue channel reference input |
| $V_{SSA(ADC)(B)}$ | 5 | – | analog supply ground for ADC; blue channel |
| $V_{DDA(B)}$ | 6 | – | analog supply voltage; blue channel (2.5 V) |
| $V_{SSA(B)}$ | 7 | – | analog supply ground; blue channel |
| $V_{DDA(IB)}$ | 8 | – | analog supply voltage for input buffers (3.0 V) |
| RBIAS | 9 | A | external bias resistor input |
| $V_{SSA(BIAS)(SOG)}$ | 10 | – | analog ground for bias; sync-on-green |
| SOGIN | 11 | A | sync-on-green input |
| $V_{SSA(BIAS)(G)}$ | 12 | – | analog ground for bias; green channel |
| GIN | 13 | A | green colour signal input |
| $V_{DDA(ADC)(G)}$ | 14 | – | analog supply voltage for ADC; green channel (2.5 V) |
| REF_G | 15 | A | green channel reference input |
| $V_{SSA(ADC)(G)}$ | 16 | – | analog supply ground for ADC; green channel |
| $V_{DDA(G)}$ | 17 | – | analog supply voltage; green channel (2.5 V) |
| $V_{SSA(G)}$ | 18 | – | analog supply ground; green channel |
| $V_{SSA(BIAS)(R)}$ | 19 | – | analog ground for bias; red channel |
| RIN | 20 | A | red colour signal input |
| $V_{DDA(ADC)(R)}$ | 21 | – | analog supply voltage for ADC; red channel (2.5 V) |
| REF_R | 22 | A | red channel reference input |
| $V_{SSA(ADC)(R)}$ | 23 | – | analog supply ground for ADC; red channel |
| $V_{DDA(R)}$ | 24 | – | analog supply voltage; red channel (2.5 V) |
| $V_{SSA(R)}$ | 25 | – | analog supply ground; red channel |
| $V_{DDD(IC1)}$ | 26 | – | internal digital core supply voltage 1 (2.5 V) |
| n.c. | 27 | – | not connected |
| n.c. | 28 | – | not connected |
| $V_{SSD(IC1)}$ | 29 | – | internal digital core supply ground 1 |
| n.c. | 30 | – | not connected |
| n.c. | 31 | – | not connected |
| $V_{DDD(IC2)}$ | 32 | – | internal digital core supply voltage 2 (2.5 V) |
| n.c. | 33 | – | not connected |
| n.c. | 34 | – | not connected |
| $V_{SSD(IC2)}$ | 35 | – | internal digital core supply ground 2 |
| n.c. | 36 | – | not connected |
| RESERVED1 | 37 | – | connect with a pull-up resistor of 51 Ω to $V_{DDE}$ (3.3 V) |
| $V_{DDD(IC3)}$ | 38 | – | internal digital core supply voltage 3 (2.5 V) |
| $V_{SSD(IC3)}$ | 39 | – | internal digital core supply ground 3 |
| n.c. | 40 | – | not connected |

| SYMBOL | PIN(1) | TYPE | DESCRIPTION |
|---|---|---|---|
| SDA | 41 | I/O | serial data input or output ($I^2C$-bus) |
| SCL | 42 | I | serial clock input ($I^2C$-bus) |
| RESERVED2 | 43 | – | connect with a pull-up resistor of 4.7 kΩ to $V_{DDE}$ (3.3 or 5 V) |
| RESERVED3 | 44 | – | connect with a pull-up resistor of 4.7 kΩ to $V_{DDE}$ (3.3 or 5 V) |
| $V_{SSD(IC4)}$ | 45 | – | internal digital core supply ground 4 |
| $V_{DDD(IC4)}$ | 46 | – | internal digital core supply voltage 4 (2.5 V) |
| CLK | 47 | I | master clock input |
| $V_{SSD(EP1)}$ | 48 | – | external digital pad supply ground 1 |
| $V_{DDD(EP1)}$ | 49 | – | external digital pad supply voltage 1 (3.3 V) |
| $\overline{INT}$ | 50 | O | microcontroller interrupt output (active LOW) |
| $\overline{RST}$ | 51 | I | master reset input (active LOW) |
| PCLK | 52 | O | panel clock output |
| CSG0 | 53 | O | control signal generator 0 output |
| CSG1 | 54 | O | control signal generator 1 output |
| CSG2/A0 | 55 | I/O | control signal generator 2 output (CSG2) or $I^2C$-bus slave address input, latched via hardware reset (A0) |
| $V_{SSD(EP2)}$ | 56 | – | external digital pad supply ground 2 |
| $V_{DDD(EP2)}$ | 57 | – | external digital pad supply voltage 2 (3.3 V) |
| PA0 | 58 | I/O | panel data port A bit 0 |
| PA1 | 59 | I/O | panel data port A bit 1 |
| PA2 | 60 | I/O | panel data port A bit 2 |
| PA3 | 61 | I/O | panel data port A bit 3 |
| PA4 | 62 | I/O | panel data port A bit 4 |
| PA5 | 63 | I/O | panel data port A bit 5 |
| PA6 | 64 | I/O | panel data port A bit 6 |
| PA7 | 65 | I/O | panel data port A bit 7 |
| $V_{SSD(EP3)}$ | 66 | – | external digital pad supply ground 3 |
| $V_{DDD(EP3)}$ | 67 | – | external digital pad supply voltage 3 (3.3 V) |
| PB0 | 68 | I/O | panel data port B bit 0 |
| PB1 | 69 | I/O | panel data port B bit 1 |
| $V_{SSD(IC5)}$ | 70 | – | internal digital core supply ground 5 |
| $V_{DDD(IC5)}$ | 71 | – | internal digital core supply voltage 5 (2.5 V) |
| PB2 | 72 | I/O | panel data port B bit 2 |
| PB3 | 73 | I/O | panel data port B bit 3 |
| PB4 | 74 | I/O | panel data port B bit 4 |
| PB5 | 75 | I/O | panel data port B bit 5 |
| PB6 | 76 | I/O | panel data port B bit 6 |
| PB7 | 77 | I/O | panel data port B bit 7 |
| $V_{SSD(EP4)}$ | 78 | – | external digital pad supply ground 4 |
| $V_{DDD(EP4)}$ | 79 | – | external digital pad supply voltage 4 (3.3 V) |
| PC0 | 80 | I/O | panel data port C bit 0 |

| SYMBOL | PIN[1] | TYPE | DESCRIPTION |
|---|---|---|---|
| PC1 | 81 | I/O | panel data port C bit 1 |
| PC2 | 82 | I/O | panel data port C bit 2 |
| PC3 | 83 | I/O | panel data port C bit 3 |
| PC4 | 84 | I/O | panel data port C bit 4 |
| PC5 | 85 | I/O | panel data port C bit 5 |
| PC6 | 86 | I/O | panel data port C bit 6 |
| PC7 | 87 | I/O | panel data port C bit 7 |
| $V_{SSD(EP5)}$ | 88 | – | external digital pad supply ground 5 |
| $V_{DDD(EP5)}$ | 89 | – | external digital pad supply voltage 5 (3.3 V) |
| $V_{SSD(IC6)}$ | 90 | – | internal digital core supply ground 6 |
| $V_{DDD(IC6)}$ | 91 | – | internal digital core supply voltage 6 (2.5 V) |
| PD0 | 92 | O | panel data port D bit 0 |
| PD1 | 93 | O | panel data port D bit 1 |
| PD2 | 94 | O | panel data port D bit 2 |
| PD3 | 95 | O | panel data port D bit 3 |
| PD4 | 96 | O | panel data port D bit 4 |
| PD5 | 97 | O | panel data port D bit 5 |
| PD6 | 98 | O | panel data port D bit 6 |
| PD7 | 99 | O | panel data port D bit 7 |
| $V_{SSD(EP6)}$ | 100 | – | external digital pad supply ground 6 |
| $V_{DDD(EP6)}$ | 101 | – | external digital pad supply voltage 6 (3.3 V) |
| $V_{SSD(IC7)}$ | 102 | – | internal digital core supply ground 7 |
| $V_{DDD(IC7)}$ | 103 | – | internal digital core supply voltage 7 (2.5 V) |
| PE0 | 104 | O | panel data port E bit 0 |
| PE1 | 105 | O | panel data port E bit 1 |
| PE2 | 106 | O | panel data port E bit 2 |
| PE3 | 107 | O | panel data port E bit 3 |
| PE4 | 108 | O | panel data port E bit 4 |
| PE5 | 109 | O | panel data port E bit 5 |
| PE6 | 110 | O | panel data port E bit 6 |
| PE7 | 111 | O | panel data port E bit 7 |
| $V_{SSD(EP7)}$ | 112 | – | external digital pad supply ground 7 |
| $V_{DDD(EP7)}$ | 113 | – | external digital pad supply voltage 7 (3.3 V) |
| PF0 | 114 | O | panel data port F bit 0 |
| PF1 | 115 | O | panel data port F bit 1 |
| PF2 | 116 | O | panel data port F bit 2 |
| PF3 | 117 | O | panel data port F bit 3 |
| PF4 | 118 | O | panel data port F bit 4 |
| PF5 | 119 | O | panel data port F bit 5 |
| PF6 | 120 | O | panel data port F bit 6 |
| PF7 | 121 | O | panel data port F bit 7 |

| SYMBOL | PIN[1] | TYPE | DESCRIPTION |
|---|---|---|---|
| $V_{SSD(IC8)}$ | 122 | – | internal digital core supply ground 8 |
| $V_{DDD(IC8)}$ | 123 | – | internal digital core supply voltage 8 (2.5 V) |
| $V_{SSD(EP8)}$ | 124 | – | external digital pad supply ground 8 |
| $V_{DDD(EP8)}$ | 125 | – | external digital pad supply voltage 8 (3.3 V) |
| CSG3 | 126 | O | control signal generator 3 output |
| CSG4/A1 | 127 | I/O | control signal generator 4 output (CSG4) or $I^2C$-bus slave address input, latched via hardware reset (A1) |
| CSG5 | 128 | O | control signal generator 5 output |
| CSG6 | 129 | O | control signal generator 6 output |
| CSG7 | 130 | O | control signal generator 7 output |
| $V_{SSD(EP9)}$ | 131 | – | external digital pad supply ground 9 |
| $V_{DDD(EP9)}$ | 132 | – | external digital pad supply voltage 9 (3.3 V) |
| CSG8 | 133 | O | control signal generator 8 output |
| CSG9 | 134 | O | control signal generator 9 output |
| VCLK | 135 | I/O | sample clock input or output; configurable as output if generated internally |
| INVA | 136 | O | data inversion output of ports A, B and C |
| INVB | 137 | O | data inversion output of ports D, E and F |
| OUTEN | 138 | O | output enable status output |
| PWM | 139 | O | pulse width modulation for control of backlight brightness output |
| VSYNC | 140 | I/O | vertical sync input or output; configurable as output if decoded from composite sync |
| HSYNC | 141 | I | horizontal and composite sync input |
| $V_{SSD(EP10)}$ | 142 | – | external digital pad supply ground 10 |
| $V_{DDD(EP10)}$ | 143 | – | external digital pad supply voltage 10 (3.3 V) |
| $V_{SSD(IC9)}$ | 144 | – | internal digital core supply ground 9 |
| $V_{DDD(IC9)}$ | 145 | – | internal digital core supply voltage 9 (2.5 V) |
| $V_{SS(PLL)(P)}$ | 146 | – | supply ground for panel clock phase locked loop |
| $V_{DD(PLL)(P)}$ | 147 | – | supply voltage for panel clock phase locked loop (2.5 V) |
| n.c. | 148 | – | do not connect |
| $V_{SSA(PLL)(S)}$ | 149 | – | analog supply ground for sample clock phase locked loop |
| $V_{DDA(PLL)(S)}$ | 150 | – | analog supply voltage for sample clock phase locked loop (2.5 V) |
| $V_{SSD(PLL)(S)}$ | 151 | – | digital supply ground for sample clock phase locked loop |
| $V_{DDD(PLL)(S)}$ | 152 | – | digital supply voltage for sample clock phase locked loop (2.5 V) |
| $\overline{TRST}$ | 153 | I | test reset input for boundary scan test (active LOW); note 2 |
| TCK | 154 | I | test clock input for boundary scan test; note 2 |
| TDI | 155 | I | test data input for boundary scan test; note 2 |
| TMS | 156 | I | test mode select input for boundary scan test or scan test; note 2 |
| TDO | 157 | O | test data output for boundary scan test |

| SYMBOL | PIN[1] | TYPE | DESCRIPTION |
|---|---|---|---|
| $V_{SSA(EP)}$ | 158 | – | external analog pad supply ground |
| $V_{DDA(EP)}$ | 159 | – | external analog pad supply voltage (3.3 V) |
| AGCANA | 160 | – | analog test pad (should be connected to analog ground for application) |

**Notes**

1. For pin type description see Table 1.
2. For board design without boundary scan implementation connect pins $\overline{\text{TRST}}$, TCK, TDI and TMS to ground.

**Table 1** Pin type description

| TYPE | DESCRIPTION |
|---|---|
| A | analog input |
| I | digital input |
| O | digital output |
| I/O | digital input or output |

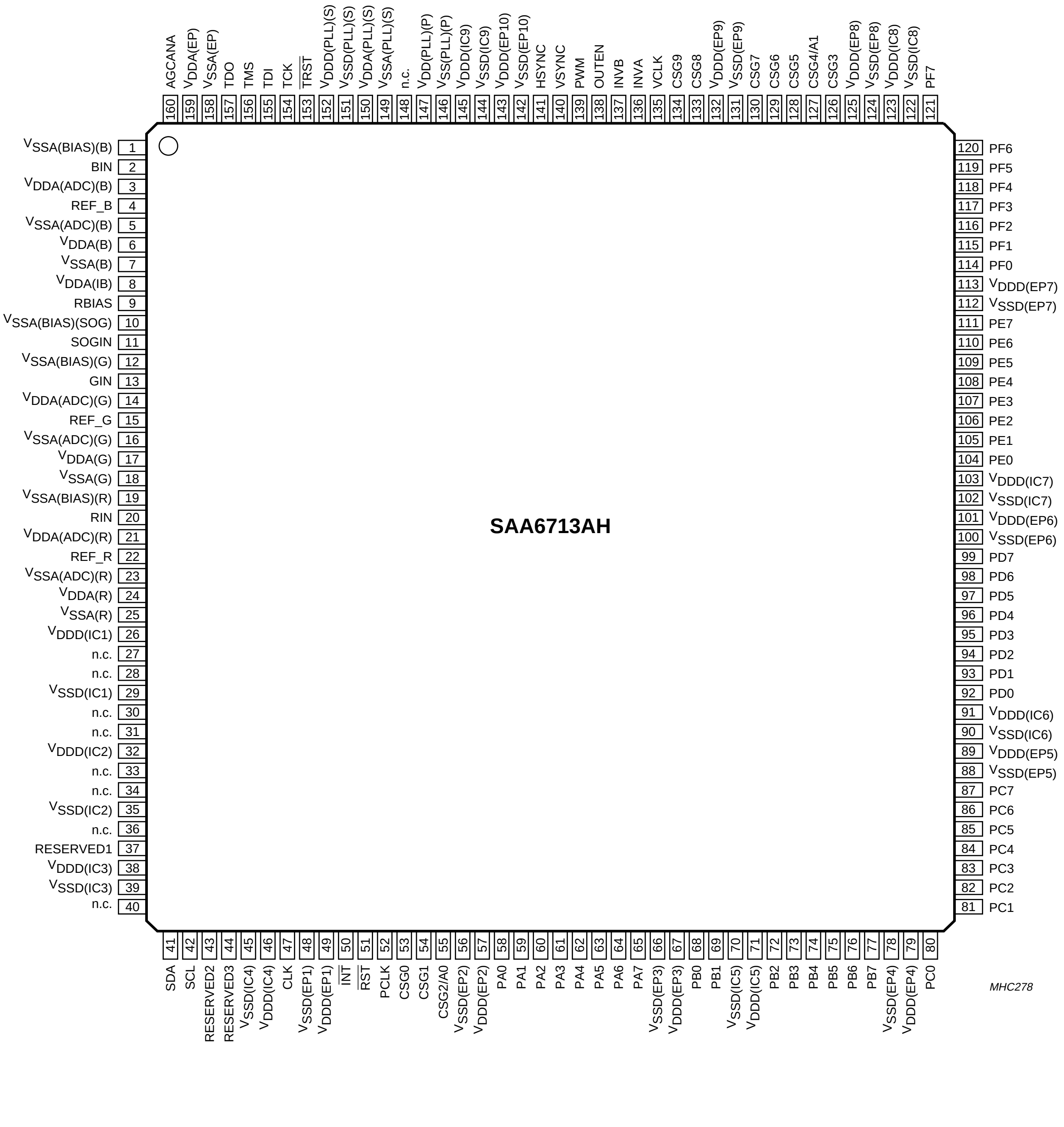



Fig.2 Pin configuration.

## 7 FUNCTIONAL DESCRIPTION

In this chapter detailed information for the general configuration of the SAA6713AH is provided as well as detailed background information belonging to certain submodules of the device. Due to the high complexity of the device functionality this section should be studied very carefully.

### 7.1 Programming registers

#### 7.1.1 CONFIGURATION PARAMETER MAPPING

The SAA6713AH operation is controlled by configuration parameters, that can be multiple-bit words or consist of only a single bit. The configuration parameters are mapped to bits of the 8 bit $I^2C$-bus programming registers, that are accessible via the $I^2C$-bus interface. Read-out data such as measurement results or interrupt states is mapped to readable $I^2C$-bus registers.

The $I^2C$-bus registers are organized in pages. Generally, a register can only be accessed if the particular page is activated with the exception of global registers, so non-global registers are addressed by the $I^2C$-bus subaddress in combination with the matching active page, but global registers are addressed by the subaddress independently of the active page.

The global registers are mapped to $I^2C$-bus subaddresses F8H to FFH. The active page is defined by page_select at subaddress FFH. In general, registers belonging to the same functional unit are mapped onto the same page. The $I^2C$-bus register pages are shown in Table 2.

**Table 2** $I^2C$-bus register pages

| PAGE | FUNCTIONAL UNIT |
|---|---|
| 0 | control unit and clock generator |
| 1 | ADC control |
| 2 | mode detection |
| 3 | auto-adjustment |
| 4 | input interface and picture generator |
| 5 | colour processing |
| 6 | decoupling FIFO |
| 7 | scalers |
| 8 | OSD |
| 9 | OSD colour definition |
| 10 | gamma correction and dithering |
| 11 | TFT output interface |

#### 7.1.2 $I^2C$-BUS INTERFACE

The $I^2C$-bus serial interface consists of two pins: the serial clock pin SCL and the serial data pin SDA.

*7.1.2.1 Transmission bit rate*

The $I^2C$-bus interface supports transmission speeds of up to 3.4 Mbits/s, given that a minimum system clock rate is provided. The required system clock rate depends on the target $I^2C$-bus bit rate, which is the clock rate applied to pin SCL, and the spike suppression mode selected by iic_spike_mode in register IIC_MODE (03H at page 0) as shown in Table 3. If iic_spike_mode is set to 2, a high oversampling rate is used and the most effective spike suppression is provided.

**Table 3** $I^2C$-bus spike suppression modes

| iic_spike_ mode[1:0] | SYSTEM CLOCK | DESCRIPTION |
|---|---|---|
| 00 | >6 × $I^2C$-bus bit rate | 2-out-of-2 filter |
| 01 | >6 × $I^2C$-bus bit rate | 2-out-of-3 majority filter |
| 10 | >16 × $I^2C$-bus bit rate | 4-out-of-4 filter |
| 11 | not used | |

*7.1.2.2 $I^2C$-bus transmission timing*

The SAA6713AH only operates as a slave and the clock pin SCL is exclusively input. Data is transmitted and received at I/O pin SDA. The SDA is an open-drain stage with an external pull-up resistor. When a logic 0 is applied, the bus is pulled to LOW-level by the output buffer. When a logic 1 is applied, the output buffer switches to 3-state and the pull-up resistor pulls the bus up to HIGH-level.

Data transfers are initiated by an $I^2C$-bus master device by sending the start condition, which is a change from HIGH-to-LOW level at SDA when SCL is at HIGH-level (see Fig.3).

Data is transmitted byte wise. Data changes on SDA are allowed only when SCL is at LOW-level and data is sampled on the positive edge of SCL. The first transmitted byte is the recipients $I^2C$-bus device address and the data transfer direction bit. All byte transfers are acknowledged by the recipient by pulling SDA to LOW-level for the following cycle.

If the write mode was selected, the bus master sends a byte containing the starting subaddress and then a series of data bytes. In case the read mode was selected, the addressed slave returns a series of data bytes. A read transfer is preceded by a write transfer that transmits the starting subaddress.

Data transfers are aborted by the stop condition, when SDA is changed by the master from LOW-to-HIGH level when SCL is at HIGH-level (see Fig.4).

#### *7.1.2.3 I²C-bus device address*

Bits A0 and A1 of the I²C-bus device address are externally selected by two input pins CSG2/A0 and CSG4/A1. The device address (byte) of the SAA6713AH is shown in Table 4.

**Table 4** I²C-bus device address byte

| **MSB** | | | | | | | **LSB** |
|---|---|---|---|---|---|---|---|
| **DEVICE ADDRESS BITS** | | | | | | | **R/$\overline{W}$** |
| 0 | 1 | 1 | 1 | 0 | A1 | A0 | 0/1 |

The four possible I²C-bus device addresses are selected via resistor strapping at pins CSG2/A0 and CSG4/A1 (see Table 5).

During the hardware reset (pin $\overline{RST}$ = LOW), pins CSG2/A0 and CSG4/A1 are 3-stated. Their status at the trailing edge of signal $\overline{RST}$ will latch and determine the device address. Pull-up and pull-down resistors (4.7 kΩ suggested) select the address. An internal pull-down resistance of approximately 100 kΩ is provided and eliminates potentially the need for any external strapping resistor. After reset, the pins carry the output of the programmable signal generators.

**Table 5** Device address selection

| **I²C-BUS DEVICE ADDRESS** | **STRAPPING RESISTOR** | |
|---|---|---|
| | **PIN CSG4/A1** | **PIN CSG2/A0** |
| 70H | pull-down | pull-down |
| 72H | pull-down | pull-up |
| 74H | pull-up | pull-down |
| 76H | pull-up | pull-up |

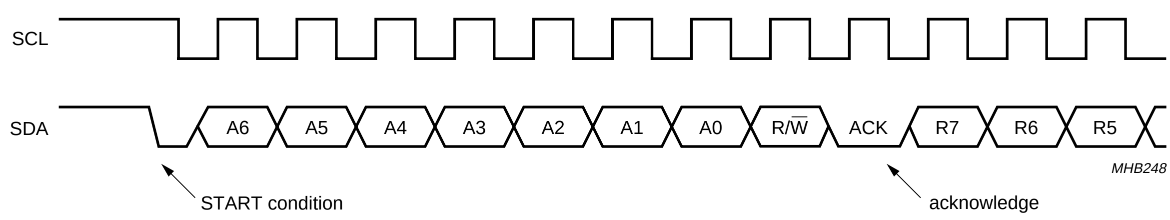



Fig.3 Start of a data transfer.

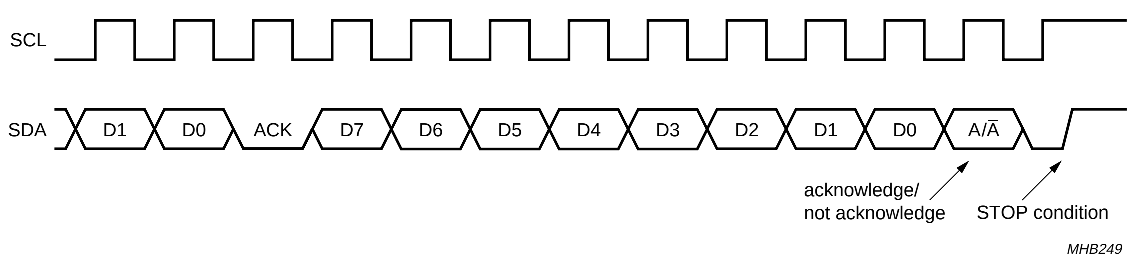



Fig.4 End of a data transfer.

#### 7.1.2.4 *$I^2C$-bus subaddress*

When transmitting a series of data bytes, after a data byte has been written or read, the subaddress for the following byte is automatically updated to allow burst access. During burst access a sequence of data bytes is written or read without repeated device or subaddressing. In general, the $I^2C$-bus auto-increment feature uses the next higher subaddress as the succeeding byte's subaddress.

Auto-incrementing is suppressed for several addresses that provide access to the on-chip parameter RAM. In the event of upscaler register USC_LUT_DATA (02H at page 7) subsequent data is written to the same subaddress and the scaling curve RAM address is incremented instead.

For OSD registers OSDT_PROP2 to OSDT_PROP0, OSDB_DEF and OSDP_DEF (0FH to 11H, 31H and 4CH at page 8) and colour look-up table register CL_VALUE_LO (03H at page 10) different subaddress update modes are selectable and are described in the respective subsection.

#### 7.1.2.5 *Multiple byte parameters*

Parameters or read-out data words consisting of more than 8 bits are mapped into the address space in the order highest byte at the lowest address to lowest byte at the highest address. Multiple byte configuration parameters have to be written lowest address first and only become effective, once the byte of the highest address was written. Multiple byte read registers have to be read-out in the same order.

#### 7.1.2.6 *$I^2C$-bus test register*

Register IIC_TEST (02H at page 0) is a read and write register that can be used to verify correct operations of the $I^2C$-bus. Any programmed value can be read back.

### 7.1.3 $I^2C$-BUS REGISTER LISTING

The global registers are listed in Table 6.

The page-mapped registers are listed for each register page in Tables 7 to 17.

**Table 6** Global configuration registers

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Global control: FAH to FFH** | | | | | | | | | | | |
| GC_MISC0 | FAH | W | 00H | avi_ noclamp_ sog_en | reserved | | | | | | |
| GC_MISC1 | FBH | W | FFH | avi_ noclamp_ pol | reserved | | | | | | |
| GC_RESET | FCH | W | 1FH | | | | reset_ csdec_n | reserved | reset_fclk | reset_bclk | reset_oif |
| GC_INT_MASK | FDH | W | –0–0 0000 | | int_iif_en | | int_mode_ en | int_auto_ en | int_fifo_en | int_osd_en | int_oif_en |
| GC_INT_CLR | FEH | W | –1–1 1111 | | int_iif_clr | | int_mode | int_auto | int_fifo | int_osd | int_oif |
| GC_INT_STAT | FEH | R | –0–0 0000 | | int_iif_stat | | int_mode | int_auto | int_fifo | int_osd | int_oif |
| GC_PAGE | FFH | R/W | ––––– 0000 | | | | | page_select[3:0] | | | |

**Table 7** General control configuration registers (page 0); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Device identification: 00H to 03H** | | | | | | | | | | | |
| DEV_ID_HI | 00H | R | 13H | dev_id[15:8] | | | | | | | |
| DEV_ID_LO | 01H | R | 1CH | dev_id[7:0] | | | | | | | |
| IIC_TEST | 02H | R/W | 00H | iic_test[7:0] | | | | | | | |
| IIC_MODE | 03H | W | ––––– ––00 | | | | | | | iic_spike_mode[1:0] | |
| **Clock distribution: 10H to 12H** | | | | | | | | | | | |
| CD_CLK_EN | 10H | W | ––00 0000 | | | cfgclk_on | osd_ cfgclk_on | aaclk_on | dscclk_on | uscclk_on | osdclk_on |
| CD_CLK_AUTO | 11H | W | ––––– 1111 | | | | | aaclk_auto | dscclk_ auto | uscclk_ auto | osdclk_ auto |
| CD_CLK_MUX | 12H | W | ––11 0110 | | | vclk_in_en | cfgclk_ select | fifo_fclk | frontend_ bclk | bclk_in_en | clk_div4 |
| **Sync distribution: 18H and 19H** | | | | | | | | | | | |
| SYNC_SEL | 18H | W | –––0 0000 | | | | hs_regen_ in_en | vsync_out_ en | reserved | sog_out_ en | sog_en |

–

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SYNC_DIS | 19H | W | –000 0000 | | reserved | mdd_cs_ sog_en | mdd_hs_ regen_on | | reserved | iif_cs_sog_ en | iif_hs_ regen_on |
| **PLL programming: 20H to 29H** | | | | | | | | | | | |
| CD_PLL_CTRL | 20H | W | –010 –000 | | line_pll_ hs_pol | line_pll_ vs_pol | line_pll_en | | pll_src | pll_pre_ div_en | pll_en |
| CD_PLL_P_HI | 21H | W | 00H | pll_pre_div[15:8] | | | | | | | |
| CD_PLL_P_LO | 22H | W | 00H | pll_pre_div[7:0] | | | | | | | |
| CD_PLL_HI | 23H | W | —00 0000 | | | pll_m_div[1:0] | | pll_n_div[11:8] | | | |
| CD_PLL_LO | 24H | W | 00H | pll_n_div[7:0] | | | | | | | |
| CD_LPLL_HI | 25H | W | —00 0000 | | | line_pll_m_div[1:0] | | line_pll_n_div[11:8] | | | |
| CD_LPLL_LO | 26H | W | 00H | line_pll_n_div[7:0] | | | | | | | |
| CD_LPLL_PHA | 27H | W | ——0 0000 | | | | line_pll_phase[4:0] | | | | |
| CD_LPLL_PD | 28H | W | –100 0000 | | phase_ auto | phase_ select | pd_pll_phase[4:0] | | | | |
| CD_PLL_LOCK | 29H | R | —— –XXX | | | | | | phase_ inlock | pll_inlock | llpll_inlock |
| **Interface timing: 34H and 35H** | | | | | | | | | | | |
| IT_CTRL | 34H | W | —— ––11 | | | | | | | adc_pon_ pol | bigger_ out_pol |
| IT_PLL | 35H | W | —— 1111 | | | | | pll_coast_ pol | pll_pon_pol | llpll_coast_ pol | llpll_pon_ pol |

**Note**

1. X = don’t care.

**Table 8** ADC configuration registers (page 1)

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ADC programming: 00H to 06H** | | | | | | | | | | | |
| ADC_CTRL | 00H | W | —— –000 | | | | | | sog_vs_ disable | reserved | sync_on_ green_en |
| ADC_R_BRI | 01H | W | 00H | adc_red_brightness[7:0] | | | | | | | |
| ADC_R_CON | 02H | W | 00H | adc_red_contrast[7:0] | | | | | | | |
| ADC_G_BRI | 03H | W | 00H | adc_green_brightness[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ADC_G_CON | 04H | W | 00H | adc_green_contrast[7:0] | | | | | | | |
| ADC_B_BRI | 05H | W | 00H | adc_blue_brightness[7:0] | | | | | | | |
| ADC_B_CON | 06H | W | 00H | adc_blue_contrast[7:0] | | | | | | | |

**Table 9** Mode detection configuration registers (page 2); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Mode detection: 00H to 0EH** | | | | | | | | | | | |
| MD_CTRL | 00H | W | –000 0000 | | no_vsync_ int_en | clear_int | int_lock | delay_ vsync | h_clocks_ accu | h_clocks_ cont | md_on |
| MD_INT_EN | 01H | W | 0000 0000 | jitter_int_en | v_lines_ int_en | v_clocks_ int_en | h_clocks_ int_en | no_hsync_ int_en | vsync_int_ en | vsync_pol_ int_en | hsync_pol_ int_en |
| MD_POL | 02H | R | –––0 0011 | | | | jitter_ detected | vsync_pol | hsync_pol | no_vsync | no_hsync |
| MD_V_LINE_HI | 03H | R | –––– –000 | | | | | | v_lines[10:8] | | |
| MD_V_LINE_LO | 04H | R | 00H | v_lines[7:0] | | | | | | | |
| MD_H_CLK_HI | 05H | R | 00H | h_clocks[15:8] | | | | | | | |
| MD_H_CLK_LO | 06H | R | 00H | h_clocks[7:0] | | | | | | | |
| MD_V_CLK_HI | 07H | R | 00H | v_clocks[23:16] | | | | | | | |
| MD_V_CLK_MD | 08H | R | 00H | v_clocks[15:8] | | | | | | | |
| MD_V_CLK_LO | 09H | R | 00H | v_clocks[7:0] | | | | | | | |
| MD_INT_HI | 0AH | R | ––00 0000 | | | vsync_int | jitter_int | vsync_pol_ int | hsync_pol_ int | no_vsync_ int | no_hsync_ int |
| MD_INT_LO | 0BH | R | –––– –000 | | | | | | v_lines_int | h_clocks_ int | v_clocks_ int |
| MD_ACT_INT | 0CH | R | X000 0000 | reserved | reserved | reserved | reserved | asog_act_ int | acsvs_act_ int | avs_act_int | ahs_act_int |
| MD_SYNC_ACT | 0DH | R | ––00 0000 | | | reserved | reserved | asog_ active | acsvs_ active | avs_active | ahs_active |
| MD_ACT_IEN | 0EH | W | 0000 0000 | reserved | reserved | reserved | reserved | asog_int_ en | acsvs_int_ en | avs_int_en | ahs_int_en |

**Note**

1. X = don’t care.

**Table 10** Auto-adjustment configuration registers (page 3); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Auto-adjustment: 08H to 16H and 40H to 4DH** | | | | | | | | | | | |
| AA_RC0_HI | 08H | W | —— –XXX | | | | | | ref_col_0[10:8] | | |
| AA_RC0_LO | 09H | W | XX | ref_col_0[7:0] | | | | | | | |
| AA_RR0_HI | 0AH | W | —— XXXX | | | | | ref_row_0[11:8] | | | |
| AA_RR0_LO | 0BH | W | XX | ref_row_0[7:0] | | | | | | | |
| AA_RC1_HI | 0CH | W | —— –XXX | | | | | | ref_col_1[10:8] | | |
| AA_RC1_LO | 0DH | W | XX | ref_col_1[7:0] | | | | | | | |
| AA_RR1_HI | 0EH | W | —— XXXX | | | | | ref_row_1[11:8] | | | |
| AA_RR1_LO | 0FH | W | XX | ref_row_1[7:0] | | | | | | | |
| AA_RCR0 | 10H | W | XX | ref_colour_0[23:16] | | | | | | | |
| AA_RCG0 | 11H | W | XX | ref_colour_0[15:8] | | | | | | | |
| AA_RCB0 | 12H | W | XX | ref_colour_0[7:0] | | | | | | | |
| AA_RCR1 | 13H | W | XX | ref_colour_1[23:16] | | | | | | | |
| AA_RCG1 | 14H | W | XX | ref_colour_1[15:8] | | | | | | | |
| AA_RCB1 | 15H | W | XX | ref_colour_1[7:0] | | | | | | | |
| AA_CTRL | 16H | W | —01 1000 | | | aa_cycles[1:0] | | aa_submode[1:0] | | aa_mode[1:0] | |
| AA_EPR0 | 40H | R | 00H | ref_pixel_red_0[7:0] | | | | | | | |
| AA_EPG0 | 42H | R | 00H | ref_pixel_green_0[7:0] | | | | | | | |
| AA_EPB0 | 44H | R | 00H | ref_pixel_blue_0[7:0] | | | | | | | |
| AA_EPR1 | 41H | R | 00H | ref_pixel_red_1[7:0] | | | | | | | |
| AA_EPG1 | 43H | R | 00H | ref_pixel_green_1[7:0] | | | | | | | |
| AA_EPB1 | 45H | R | 00H | ref_pixel_blue_1[7:0] | | | | | | | |
| AA_ER0_HI | 46H | R | 00H | eval_row_0[15:8] | | | | | | | |
| AA_ER0_LO | 47H | R | 00H | eval_row_0[7:0] | | | | | | | |
| AA_EC0_HI | 48H | R | 00H | eval_col_0[15:8] | | | | | | | |
| AA_EC0_LO | 49H | R | 00H | eval_col_0[7:0] | | | | | | | |
| AA_ER1_HI | 4AH | R | 00H | eval_row_1[15:8] | | | | | | | |
| AA_ER1_LO | 4BH | R | 00H | eval_row_1[7:0] | | | | | | | |
| AA_EC1_HI | 4CH | R | 00H | eval_col_1[15:8] | | | | | | | |
| AA_EC1_LO | 4DH | R | 00H | eval_col_1[7:0] | | | | | | | |

–

**Note**

1. X = don't care.

**Table 11** Input interface configuration registers (page 4)

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Input interface: 00H to 0FH** | | | | | | | | | | | |
| II_CTRL | 00H | W | 0–00 0101 | test_out_ en | | reverse_ field_id | interlace_ on | test_pic_on | ext_select | set to '0' | in_form_on |
| II_SYNC_CTRL | 01H | W | 110– –011 | sync_clk_ edge | ext_clk_ edge | reserved | | | hsync_ edge | vs_pol | hs_pol |
| II_ADC_CTRL | 02H | W | 1001 1111 | delay_vs | convert_2s | reserved | | gainc_en | clamp_en | gainc_pol | clamp_pol |
| II_CLAMP_ON | 03H | W | 02H | clamp_on_delay[7:0] | | | | | | | |
| II_CLAMP_OFF | 04H | W | 05H | clamp_off_delay[7:0] | | | | | | | |
| II_GAINC_ON | 05H | W | 01H | gainc_on_delay[7:0] | | | | | | | |
| II_GAINC_OFF | 06H | W | 01H | gainc_off_delay[7:0] | | | | | | | |
| II_HLEN_HI | 07H | W | —— 0000 | | | | | in_h_length[11:8] | | | |
| II_HLEN_LO | 08H | W | 3CH | in_h_length[7:0] | | | | | | | |
| II_VLEN_HI | 09H | W | —— 0000 | | | | | in_v_length[11:8] | | | |
| II_VLEN_LO | 0AH | W | 28H | in_v_length[7:0] | | | | | | | |
| II_HOFF_HI | 0BH | W | —— 0000 | | | | | in_h_offset[11:8] | | | |
| II_HOFF_LO | 0CH | W | 00H | in_h_offset[7:0] | | | | | | | |
| II_VOFF_HI | 0DH | W | —— 0000 | | | | | in_v_offset[11:8] | | | |
| II_VOFF_LO | 0EH | W | 00H | in_v_offset[7:0] | | | | | | | |
| II_HJIT | 0FH | W | 05H | hs_jitter_th[7:0] | | | | | | | |
| **Picture generator: 10H to 1CH** | | | | | | | | | | | |
| PG_CTRL | 10H | W | 0010 0010 | invert | white_ border | h_ramp_r | h_ramp_g | h_ramp_b | v_ramp_r | v_ramp_g | v_ramp_b |
| PG_HTOTAL_HI | 11H | W | —— 0000 | | | | | h_length_total[11:8] | | | |
| PG_HTOTAL_LO | 12H | W | 51H | h_length_total[7:0] | | | | | | | |
| PG_VTOTAL_HI | 13H | W | —— 0000 | | | | | v_length_total[11:8] | | | |
| PG_VTOTAL_LO | 14H | W | 35H | v_length_total[7:0] | | | | | | | |
| PG_HSTEP1 | 15H | W | 01H | h_step1[7:0] | | | | | | | |
| PG_HINC1 | 16H | W | 02H | h_colour_inc1[7:0] | | | | | | | |
| PG_HSTEP2 | 17H | W | 00H | h_step2[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PG_HINC2 | 18H | W | FFH | h_colour_inc2[7:0] | | | | | | | |
| PG_VSTEP1 | 19H | W | 14H | v_step1[7:0] | | | | | | | |
| PG_VINC1 | 1AH | W | FFH | v_colour_inc1[7:0] | | | | | | | |
| PG_VSTEP2 | 1BH | W | 01H | v_step2[7:0] | | | | | | | |
| PG_VINC2 | 1CH | W | FFH | v_colour_inc2[7:0] | | | | | | | |

**Table 12** Colour processing configuration registers (page 5)

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Colour processing: 00H to 0BH** | | | | | | | | | | | |
| CP_GAIN_Y | 00H | W | 80H | gain_y[7:0] | | | | | | | |
| CP_GAIN_CB | 01H | W | 80H | gain_cb[7:0] | | | | | | | |
| CP_GAIN_CR | 02H | W | 80H | gain_cr[7:0] | | | | | | | |
| CP_OFFS_Y | 03H | W | 00H | offset_y[7:0] | | | | | | | |
| CP_OFFS_CB | 04H | W | 00H | offset_cb[7:0] | | | | | | | |
| CP_OFFS_CR | 05H | W | 00H | offset_cr[7:0] | | | | | | | |
| CP_GAIN_R | 06H | W | 80H | gain_r[7:0] | | | | | | | |
| CP_GAIN_G | 07H | W | 80H | gain_g[7:0] | | | | | | | |
| CP_GAIN_B | 08H | W | 80H | gain_b[7:0] | | | | | | | |
| CP_OFFS_R | 09H | W | 00H | offset_r[7:0] | | | | | | | |
| CP_OFFS_G | 0AH | W | 00H | offset_g[7:0] | | | | | | | |
| CP_OFFS_B | 0BH | W | 00H | offset_b[7:0] | | | | | | | |

**Table 13** Decoupling FIFO configuration registers (page 6)

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Decoupling FIFO: 00H and 01H** | | | | | | | | | | | |
| DF_THLD | 00H | W | 01H | fifo_threshold[7:0] | | | | | | | |
| DF_CTRL | 01H | W | —— ––10 | | | | | | | line_lock | reserved |

–

**Table 14** Scaler configuration registers (page 7); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Upscaler: 00H to 09H, 0DH, 0FH, 11H and 14H to 18H** | | | | | | | | | | | |
| USC_CTRL | 00H | W | 1010 1101 | filter_type[1:0] | | 1 | 0 | 0 | 0 | usc_flip_h | usc_en |
| USC_LUT_ADR | 01H | W | 1100 0000 | v_lut_sel | h_lut_sel | lut_addr[5:0] | | | | | |
| USC_LUT_DATA | 02H | W | XX | lut_data[7:0] | | | | | | | |
| USC_H_INC_HI | 03H | W | —— 0000 | | | | | h_scale_incr[11:8] | | | |
| USC_H_INC_LO | 04H | W | 55H | h_scale_incr[7:0] | | | | | | | |
| USC_H_CORR | 05H | W | –010 0010 | | h_scale_corr[6:0] | | | | | | |
| USC_V_INC_HI | 06H | W | —— 0000 | | | | | v_scale_incr[11:8] | | | |
| USC_V_INC_LO | 07H | W | 0110 0000 | v_scale_incr[7:0] | | | | | | | |
| USC_V_CORR | 08H | W | –000 0000 | | v_scale_corr[6:0] | | | | | | |
| USC_H_PHA | 09H | W | —00 0000 | | | h_phase_off[5:0] | | | | | |
| USC_V_PHA_0 | 0DH | W | —00 0000 | | | v_phase_off_0[5:0] | | | | | |
| USC_V_PHA_1 | 0FH | W | —00 0000 | | | v_phase_off_1[5:0] | | | | | |
| USC_PHA_SEL | 11H | W | —— –000 | | | | | | v_phase_off_sel[1:0] | | set to ‘0’ |
| Reserved | 14H to 18H | – | – | | | | | | | | |
| **Downscaler: 40H to 44H** | | | | | | | | | | | |
| DS_EN | 40H | W | —— ––10 | | | | | | | flip_h | dsc_en |
| DS_HSC | 41H | W | –011 0011 | | dsc_h_incr[6:0] | | | | | | |
| DS_HSC_CO | 42H | W | –001 0100 | | dsc_h_incr_corr[6:0] | | | | | | |
| DS_VSC | 43H | W | –011 0000 | | dsc_v_incr[6:0] | | | | | | |
| DS_VSC_CO | 44H | W | –000 0000 | | dsc_v_incr_corr[6:0] | | | | | | |

**Note**

1. X = don’t care.

–

**Table 15** Definition of OSD configuration registers (pages 8 and 9); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Control registers (page 8)** | | | | | | | | | | | |
| OSD TEXT: 00H TO 1FH | | | | | | | | | | | |
| OSDT_CTRL0 | 00H | W | X000 0000 | areafill_ start | window_ shadow | h_flip | v_flip | rotate_right | zoom[1:0] | | text_on |
| OSDT_CTRL1 | 01H | W | —— 0011 | | | | | txt_ shadow_ style | 0 | 1 | 1 |
| OSDT_BGA | 02H | W | 7FH | bg_alpha[7:0] | | | | | | | |
| OSDT_FGA | 03H | W | 7FH | fg_alpha[7:0] | | | | | | | |
| OSDT_WX | 04H | W | 28H | text_column[7:0] | | | | | | | |
| OSDT_WY | 05H | W | 1EH | text_row[7:0] | | | | | | | |
| OSDT_PX_HI | 06H | W | —— –000 | | | | | | x_position[10:8] | | |
| OSDT_PX_LO | 07H | W | 00H | x_position[7:0] | | | | | | | |
| OSDT_PY_HI | 08H | W | —— –000 | | | | | | y_position[10:8] | | |
| OSDT_PY_LO | 09H | W | 00H | y_position[7:0] | | | | | | | |
| OSDT_WSHAD | 0AH | W | –000 –000 | | window_shadow_height[2:0] | | | | window_shadow_width[2:0] | | |
| OSDT_BDLY | 0BH | W | 3CH | blink_delay[7:0] | | | | | | | |
| OSDT_CURX | 0CH | R/W | 00H | cursor_column[7:0] | | | | | | | |
| OSDT_CURY | 0DH | R/W | 00H | cursor_row[7:0] | | | | | | | |
| OSDT_MASK | 0EH | W | 1111 1111 | blink_mask | shadow_ mask | bg_mask | fg_mask | code_mask | write_mode[2:0] | | |
| OSDT_PROP2 | 0FH | W | –000 0000 | | blink[1:0] | | shadow | bg_trans | fg_trans | bg_alpha | fg_alpha |
| OSDT_PROP1 | 10H | W | 0001 1110 | bg_colour[2:0] | | | fg_colour[2:0]/palette[2:0] | | | ROM | charcode [8] |
| OSDT_PROP0 | 11H | W | 00H | charcode[7:0] | | | | | | | |
| OSDT_FR_X | 12H | W | ––-0 1100 | | | | font_horizontal_resolution[4:0] | | | | |
| OSDT_FR_Y | 13H | W | ––-1 0010 | | | | font_vertical_resolution[4:0] | | | | |
| OSDT_SC_HI | 14H | W | —— ––0 | | | | | | | | sc_ startcode [8] |
| OSDT_SC_LO | 15H | W | 00H | sc_startcode[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_CC_HI | 16H | W | 0—— ——0 | single_ char_def | | | | | | | define_ charcode [8] |
| OSDT_CC_LO | 17H | W | 00H | define_charcode[7:0] | | | | | | | |
| OSDT_CMASK | 18H | W | FFH | definition_mask[7:0] | | | | | | | |
| OSDT_CDEF | 19H | W | 00H | char_definition[7:0] | | | | | | | |
| OSDT_FAULX | 1AH | W | XX | fill_area_upper_left_corner_x[7:0] | | | | | | | |
| OSDT_FAULY | 1BH | W | XX | fill_area_upper_left_corner_y[7:0] | | | | | | | |
| OSDT_FABRX | 1CH | W | XX | fill_area_bottom_right_corner_x[7:0] | | | | | | | |
| OSDT_FABRY | 1DH | W | XX | fill_area_bottom_right_corner_y[7:0] | | | | | | | |
| OSDT_SLP1 | 1EH | W | 0001 0001 | slider_border[3:0] | | | | slider_offset[3:0] | | | |
| OSDT_SLP0 | 1FH | W | ——0 0001 | | | | slider_style | slider_gap[3:0] | | | |
| OSD BITMAP: 20H TO 31H | | | | | | | | | | | |
| OSDB_CTRL0 | 20H | W | –000 0000 | | bitmap_ behind | h_flip | v_flip | rotate_right | zoom[1:0] | | bitmap_on |
| OSDB_CTRL1 | 21H | W | –XX0 0011 | | bpp[1:0] | | bg_trans | fg_trans | 0 | 1 | 1 |
| OSDB_BGA | 22H | W | 7FH | bg_alpha[7:0] | | | | | | | |
| OSDB_FGA | 23H | W | 7FH | fg_alpha[7:0] | | | | | | | |
| OSDB_SX_HI | 24H | W | —— –100 | | | | | | width[10:8] | | |
| OSDB_SX_LO | 25H | W | 00H | width[7:0] | | | | | | | |
| OSDB_SY_HI | 26H | W | —— –000 | | | | | | height[10:8] | | |
| OSDB_SY_LO | 27H | W | 04H | height[7:0] | | | | | | | |
| OSDB_PX_HI | 28H | W | —— –000 | | | | | | x_position[10:8] | | |
| OSDB_PX_LO | 29H | W | 00H | x_position[7:0] | | | | | | | |
| OSDB_PY_HI | 2AH | W | —— –000 | | | | | | y_position[10:8] | | |
| OSDB_PY_LO | 2BH | W | 00H | y_position[7:0] | | | | | | | |
| OSDB_CX_HI | 2CH | W | —— –000 | | | | | | cursor_column[10:8] | | |
| OSDB_CX_LO | 2DH | W | 00H | cursor_column[7:0] | | | | | | | |
| OSDB_CY_HI | 2EH | W | —— –000 | | | | | | cursor_row[10:8] | | |
| OSDB_CY_LO | 2FH | W | 00H | cursor_row[7:0] | | | | | | | |
| OSDB_MASK | 30H | W | XX | definition_mask[7:0] | | | | | | | |
| OSDB_DEF | 31H | W | XX | pixel_definition[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSD POINTER: 40H TO 4CH | | | | | | | | | | | |
| OSDP_CTRL0 | 40H | W | 0000 0000 | autosel_en | buffer_sel | h_flip | v_flip | rotate_right | zoom[1:0] | | pointer_on |
| OSDP_CTRL1 | 41H | W | —00 0011 | | | anim_int_en | bg_trans | fg_trans | 0 | 1 | 1 |
| OSDP_BGA | 42H | W | FFH | bg_alpha[7:0] | | | | | | | |
| OSDP_FGA | 43H | W | FFH | fg_alpha[7:0] | | | | | | | |
| OSDP_AD | 44H | W | 1EH | anim_delay[7:0] | | | | | | | |
| OSDP_DW | 45H | W | —— ––00 | | | | | | | defwidth[1:0] | |
| OSDP_PX_HI | 46H | W | —— –000 | | | | | | x_position[10:8] | | |
| OSDP_PX_LO | 47H | W | 00H | x_position[7:0] | | | | | | | |
| OSDP_PY_HI | 48H | W | —— –000 | | | | | | y_position[10:8] | | |
| OSDP_PY_LO | 49H | W | 00H | y_position[7:0] | | | | | | | |
| OSDP_CX | 4AH | W | ——0 0000 | | | | cursor_column[4:0] | | | | |
| OSDP_CY | 4BH | W | ——0 0000 | | | | cursor_row[4:0] | | | | |
| OSDP_DEF | 4CH | W | 00H | pixel_definition[7:0] | | | | | | | |
| **Colour definitions (page 9)** | | | | | | | | | | | |
| OSD TEXT COLOURS: 00H TO 92H | | | | | | | | | | | |
| OSDT_FGC0R | 00H | W | 00H | osd_text_foreground_colour0_red[7:0] | | | | | | | |
| OSDT_FGC0G | 01H | W | 00H | osd_text_foreground_colour0_green[7:0] | | | | | | | |
| OSDT_FGC0B | 02H | W | 00H | osd_text_foreground_colour0_blue[7:0] | | | | | | | |
| OSDT_FGC1R | 03H | W | FFH | osd_text_foreground_colour1_red[7:0] | | | | | | | |
| OSDT_FGC1G | 04H | W | 00H | osd_text_foreground_colour1_green[7:0] | | | | | | | |
| OSDT_FGC1B | 05H | W | 00H | osd_text_foreground_colour1_blue[7:0] | | | | | | | |
| OSDT_FGC2R | 06H | W | 00H | osd_text_foreground_colour2_red[7:0] | | | | | | | |
| OSDT_FGC2G | 07H | W | FFH | osd_text_foreground_colour2_green[7:0] | | | | | | | |
| OSDT_FGC2B | 08H | W | 00H | osd_text_foreground_colour2_blue[7:0] | | | | | | | |
| OSDT_FGC3R | 09H | W | 00H | osd_text_foreground_colour3_red[7:0] | | | | | | | |
| OSDT_FGC3G | 0AH | W | 00H | osd_text_foreground_colour3_green[7:0] | | | | | | | |
| OSDT_FGC3B | 0BH | W | FFH | osd_text_foreground_colour3_blue[7:0] | | | | | | | |
| OSDT_FGC4R | 0CH | W | FFH | osd_text_foreground_colour4_red[7:0] | | | | | | | |
| OSDT_FGC4G | 0DH | W | FFH | osd_text_foreground_colour4_green[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_FGC4B | 0EH | W | 00H | osd_text_foreground_colour4_blue[7:0] | | | | | | | |
| OSDT_FGC5R | 0FH | W | 00H | osd_text_foreground_colour5_red[7:0] | | | | | | | |
| OSDT_FGC5G | 10H | W | FFH | osd_text_foreground_colour5_green[7:0] | | | | | | | |
| OSDT_FGC5B | 11H | W | FFH | osd_text_foreground_colour5_blue[7:0] | | | | | | | |
| OSDT_FGC6R | 12H | W | FFH | osd_text_foreground_colour6_red[7:0] | | | | | | | |
| OSDT_FGC6G | 13H | W | 00H | osd_text_foreground_colour6_green[7:0] | | | | | | | |
| OSDT_FGC6B | 14H | W | FFH | osd_text_foreground_colour6_blue[7:0] | | | | | | | |
| OSDT_FGC7R | 15H | W | FFH | osd_text_foreground_colour7_red[7:0] | | | | | | | |
| OSDT_FGC7G | 16H | W | FFH | osd_text_foreground_colour7_green[7:0] | | | | | | | |
| OSDT_FGC7B | 17H | W | FFH | osd_text_foreground_colour7_blue[7:0] | | | | | | | |
| OSDT_BGC0R | 18H | W | 00H | osd_text_background_colour0_red[7:0] | | | | | | | |
| OSDT_BGC0G | 19H | W | 00H | osd_text_background_colour0_green[7:0] | | | | | | | |
| OSDT_BGC0B | 1AH | W | 00H | osd_text_background_colour0_blue[7:0] | | | | | | | |
| OSDT_BGC1R | 1BH | W | FFH | osd_text_background_colour1_red[7:0] | | | | | | | |
| OSDT_BGC1G | 1CH | W | 00H | osd_text_background_colour1_green[7:0] | | | | | | | |
| OSDT_BGC1B | 1DH | W | 00H | osd_text_background_colour1_blue[7:0] | | | | | | | |
| OSDT_BGC2R | 1EH | W | 00H | osd_text_background_colour2_red[7:0] | | | | | | | |
| OSDT_BGC2G | 1FH | W | FFH | osd_text_background_colour2_green[7:0] | | | | | | | |
| OSDT_BGC2B | 20H | W | 00H | osd_text_background_colour2_blue[7:0] | | | | | | | |
| OSDT_BGC3R | 21H | W | 00H | osd_text_background_colour3_red[7:0] | | | | | | | |
| OSDT_BGC3G | 22H | W | 00H | osd_text_background_colour3_green[7:0] | | | | | | | |
| OSDT_BGC3B | 23H | W | FFH | osd_text_background_colour3_blue[7:0] | | | | | | | |
| OSDT_BGC4R | 24H | W | FFH | osd_text_background_colour4_red[7:0] | | | | | | | |
| OSDT_BGC4G | 25H | W | FFH | osd_text_background_colour4_green[7:0] | | | | | | | |
| OSDT_BGC4B | 26H | W | 00H | osd_text_background_colour4_blue[7:0] | | | | | | | |
| OSDT_BGC5R | 27H | W | 00H | osd_text_background_colour5_red[7:0] | | | | | | | |
| OSDT_BGC5G | 28H | W | FFH | osd_text_background_colour5_green[7:0] | | | | | | | |
| OSDT_BGC5B | 29H | W | FFH | osd_text_background_colour5_blue[7:0] | | | | | | | |
| OSDT_BGC6R | 2AH | W | FFH | osd_text_background_colour6_red[7:0] | | | | | | | |
| OSDT_BGC6G | 2BH | W | 00H | osd_text_background_colour6_green[7:0] | | | | | | | |
| OSDT_BGC6B | 2CH | W | FFH | osd_text_background_colour6_blue[7:0] | | | | | | | |
| OSDT_BGC7R | 2DH | W | FFH | osd_text_background_colour7_red[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_BGC7G | 2EH | W | FFH | osd_text_background_colour7_green[7:0] | | | | | | | |
| OSDT_BGC7B | 2FH | W | FFH | osd_text_background_colour7_blue_7[7:0] | | | | | | | |
| OSDT_P0C0R | 30H | W | 00H | osd_palette0_colour0_red[7:0] | | | | | | | |
| OSDT_P0C0G | 31H | W | 00H | osd_palette0_colour0_green[7:0] | | | | | | | |
| OSDT_P0C0B | 32H | W | 00H | osd_palette0_colour0_blue[7:0] | | | | | | | |
| OSDT_P0C1R | 33H | W | FFH | osd_palette0_colour1_red[7:0] | | | | | | | |
| OSDT_P0C1G | 34H | W | 00H | osd_palette0_colour1_green[7:0] | | | | | | | |
| OSDT_P0C1B | 35H | W | 00H | osd_palette0_colour1_blue[7:0] | | | | | | | |
| OSDT_P0C2R | 36H | W | 00H | osd_palette0_colour2_red[7:0] | | | | | | | |
| OSDT_P0C2G | 37H | W | FFH | osd_palette0_colour2_green[7:0] | | | | | | | |
| OSDT_P0C2B | 38H | W | 00H | osd_palette0_colour2_blue[7:0] | | | | | | | |
| OSDT_P0C3R | 39H | W | 00H | osd_palette0_colour3_red[7:0] | | | | | | | |
| OSDT_P0C3G | 3AH | W | 00H | osd_palette0_colour3_green[7:0] | | | | | | | |
| OSDT_P0C3B | 3BH | W | FFH | osd_palette0_colour3_blue[7:0] | | | | | | | |
| OSDT_P1C0R | 3CH | W | FFH | osd_palette1_colour0_red[7:0] | | | | | | | |
| OSDT_P1C0G | 3DH | W | FFH | osd_palette1_colour0_green[7:0] | | | | | | | |
| OSDT_P1C0B | 3EH | W | 00H | osd_palette1_colour0_blue[7:0] | | | | | | | |
| OSDT_P1C1R | 3FH | W | 00H | osd_palette1_colour1_red[7:0] | | | | | | | |
| OSDT_P1C1G | 40H | W | FFH | osd_palette1_colour1_green[7:0] | | | | | | | |
| OSDT_P1C1B | 41H | W | FFH | osd_palette1_colour1_blue[7:0] | | | | | | | |
| OSDT_P1C2R | 42H | W | FFH | osd_palette1_colour2_red[7:0] | | | | | | | |
| OSDT_P1C2G | 43H | W | 00H | osd_palette1_colour2_green[7:0] | | | | | | | |
| OSDT_P1C2B | 44H | W | FFH | osd_palette1_colour2_blue[7:0] | | | | | | | |
| OSDT_P1C3R | 45H | W | FFH | osd_palette1_colour3_red[7:0] | | | | | | | |
| OSDT_P1C3G | 46H | W | FFH | osd_palette1_colour3_green[7:0] | | | | | | | |
| OSDT_P1C3B | 47H | W | FFH | osd_palette1_colour3_blue[7:0] | | | | | | | |
| OSDT_P2C0R | 48H | W | 40H | osd_palette2_colour0_red[7:0] | | | | | | | |
| OSDT_P2C0G | 49H | W | 40H | osd_palette2_colour0_green[7:0] | | | | | | | |
| OSDT_P2C0B | 4AH | W | 40H | osd_palette2_colour0_blue[7:0] | | | | | | | |
| OSDT_P2C1R | 4BH | W | 80H | osd_palette2_colour1_red[7:0] | | | | | | | |
| OSDT_P2C1G | 4CH | W | 00H | osd_palette2_colour1_green[7:0] | | | | | | | |
| OSDT_P2C1B | 4DH | W | 00H | osd_palette2_colour1_blue[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_P2C2R | 4EH | W | 00H | osd_palette2_colour2_red[7:0] | | | | | | | |
| OSDT_P2C2G | 4FH | W | 80H | osd_palette2_colour2_green[7:0] | | | | | | | |
| OSDT_P2C2B | 50H | W | 00H | osd_palette2_colour2_blue[7:0] | | | | | | | |
| OSDT_P2C3R | 51H | W | 00H | osd_palette2_colour3_red[7:0] | | | | | | | |
| OSDT_P2C3G | 52H | W | 00H | osd_palette2_colour3_green[7:0] | | | | | | | |
| OSDT_P2C3B | 53H | W | 80H | osd_palette2_colour3_blue[7:0] | | | | | | | |
| OSDT_P3C0R | 54H | W | 80H | osd_palette3_colour0_red[7:0] | | | | | | | |
| OSDT_P3C0G | 55H | W | 80H | osd_palette3_colour0_green[7:0] | | | | | | | |
| OSDT_P3C0B | 56H | W | 00H | osd_palette3_colour0_blue[7:0] | | | | | | | |
| OSDT_P3C1R | 57H | W | 00H | osd_palette3_colour1_red[7:0] | | | | | | | |
| OSDT_P3C1G | 58H | W | 80H | osd_palette3_colour1_green[7:0] | | | | | | | |
| OSDT_P3C1B | 59H | W | 80H | osd_palette3_colour1_blue[7:0] | | | | | | | |
| OSDT_P3C2R | 5AH | W | 80H | osd_palette3_colour2_red[7:0] | | | | | | | |
| OSDT_P3C2G | 5BH | W | 00H | osd_palette3_colour2_green[7:0] | | | | | | | |
| OSDT_P3C2B | 5CH | W | 80H | osd_palette3_colour2_blue[7:0] | | | | | | | |
| OSDT_P3C3R | 5DH | W | 80H | osd_palette3_colour3_red[7:0] | | | | | | | |
| OSDT_P3C3G | 5EH | W | 80H | osd_palette3_colour3_green[7:0] | | | | | | | |
| OSDT_P3C3B | 5FH | W | 80H | osd_palette3_colour3_blue[7:0] | | | | | | | |
| OSDT_P4C0R | 60H | W | 00H | osd_palette4_colour0_red[7:0] | | | | | | | |
| OSDT_P4C0G | 61H | W | 00H | osd_palette4_colour0_green[7:0] | | | | | | | |
| OSDT_P4C0B | 62H | W | 00H | osd_palette4_colour0_blue[7:0] | | | | | | | |
| OSDT_P4C1R | 63H | W | 3FH | osd_palette4_colour1_red[7:0] | | | | | | | |
| OSDT_P4C1G | 64H | W | 3FH | osd_palette4_colour1_green[7:0] | | | | | | | |
| OSDT_P4C1B | 65H | W | 3FH | osd_palette4_colour1_blue[7:0] | | | | | | | |
| OSDT_P4C2R | 66H | W | 7FH | osd_palette4_colour2_red[7:0] | | | | | | | |
| OSDT_P4C2G | 67H | W | 7FH | osd_palette4_colour2_green[7:0] | | | | | | | |
| OSDT_P4C2B | 68H | W | 7FH | osd_palette4_colour2_blue[7:0] | | | | | | | |
| OSDT_P4C3R | 69H | W | FFH | osd_palette4_colour3_red[7:0] | | | | | | | |
| OSDT_P4C3G | 6AH | W | FFH | osd_palette4_colour3_green[7:0] | | | | | | | |
| OSDT_P4C3B | 6BH | W | FFH | osd_palette4_colour3_blue[7:0] | | | | | | | |
| OSDT_P5C0R | 6CH | W | 00H | osd_palette5_colour0_red[7:0] | | | | | | | |
| OSDT_P5C0G | 6DH | W | 00H | osd_palette5_colour0_green[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_P5C0B | 6EH | W | 00H | osd_palette5_colour0_blue[7:0] | | | | | | | |
| OSDT_P5C1R | 6FH | W | 7FH | osd_palette5_colour1_red[7:0] | | | | | | | |
| OSDT_P5C1G | 70H | W | 00H | osd_palette5_colour1_green[7:0] | | | | | | | |
| OSDT_P5C1B | 71H | W | 00H | osd_palette5_colour1_blue[7:0] | | | | | | | |
| OSDT_P5C2R | 72H | W | 00H | osd_palette5_colour2_red[7:0] | | | | | | | |
| OSDT_P5C2G | 73H | W | 7FH | osd_palette5_colour2_green[7:0] | | | | | | | |
| OSDT_P5C2B | 74H | W | 00H | osd_palette5_colour2_blue[7:0] | | | | | | | |
| OSDT_P5C3R | 75H | W | 00H | osd_palette5_colour3_red[7:0] | | | | | | | |
| OSDT_P5C3G | 76H | W | 00H | osd_palette5_colour3_green[7:0] | | | | | | | |
| OSDT_P5C3B | 77H | W | 7FH | osd_palette5_colour3_blue[7:0] | | | | | | | |
| OSDT_P6C0R | 78H | W | C0H | osd_palette6_colour0_red[7:0] | | | | | | | |
| OSDT_P6C0G | 79H | W | C0H | osd_palette6_colour0_green[7:0] | | | | | | | |
| OSDT_P6C0B | 7AH | W | C0H | osd_palette6_colour0_blue[7:0] | | | | | | | |
| OSDT_P6C1R | 7BH | W | 80H | osd_palette6_colour1_red[7:0] | | | | | | | |
| OSDT_P6C1G | 7CH | W | 80H | osd_palette6_colour1_green[7:0] | | | | | | | |
| OSDT_P6C1B | 7DH | W | 80H | osd_palette6_colour1_blue[7:0] | | | | | | | |
| OSDT_P6C2R | 7EH | W | E0H | osd_palette6_colour2_red[7:0] | | | | | | | |
| OSDT_P6C2G | 7FH | W | E0H | osd_palette6_colour2_green[7:0] | | | | | | | |
| OSDT_P6C2B | 80H | W | E0H | osd_palette6_colour2_blue[7:0] | | | | | | | |
| OSDT_P6C3R | 81H | W | 00H | osd_palette6_colour3_red[7:0] | | | | | | | |
| OSDT_P6C3G | 82H | W | 00H | osd_palette6_colour3_green[7:0] | | | | | | | |
| OSDT_P6C3B | 83H | W | 00H | osd_palette6_colour3_blue[7:0] | | | | | | | |
| OSDT_P7C0R | 84H | W | C0H | osd_palette7_colour0_red[7:0] | | | | | | | |
| OSDT_P7C0G | 85H | W | C0H | osd_palette7_colour0_green[7:0] | | | | | | | |
| OSDT_P7C0B | 86H | W | C0H | osd_palette7_colour0_blue[7:0] | | | | | | | |
| OSDT_P7C1R | 87H | W | E0H | osd_palette7_colour1_red[7:0] | | | | | | | |
| OSDT_P7C1G | 88H | W | E0H | osd_palette7_colour1_green[7:0] | | | | | | | |
| OSDT_P7C1B | 89H | W | E0H | osd_palette7_colour1_blue[7:0] | | | | | | | |
| OSDT_P7C2R | 8AH | W | 80H | osd_palette7_colour2_red[7:0] | | | | | | | |
| OSDT_P7C2G | 8BH | W | 80H | osd_palette7_colour2_green[7:0] | | | | | | | |
| OSDT_P7C2B | 8CH | W | 80H | osd_palette7_colour2_blue[7:0] | | | | | | | |
| OSDT_P7C3R | 8DH | W | 00H | osd_palette7_colour3_red[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDT_P7C3G | 8EH | W | 00H | osd_palette7_colour3_green[7:0] | | | | | | | |
| OSDT_P7C3B | 8FH | W | 00H | osd_palette7_colour3_blue[7:0] | | | | | | | |
| OSDT_SCR | 90H | W | 00H | osd_shadow_colour_red[7:0] | | | | | | | |
| OSDT_SCG | 91H | W | 00H | osd_shadow_colour_green[7:0] | | | | | | | |
| OSDT_SCB | 92H | W | 00H | osd_shadow_colour_blue[7:0] | | | | | | | |
| OSD BITMAP COLOURS: 93H TO C2 | | | | | | | | | | | |
| OSDB_C0R | 93H | W | 00H | osd_bitmap_colour0_red[7:0] | | | | | | | |
| OSDB_C0G | 94H | W | 00H | osd_bitmap_colour0_green[7:0] | | | | | | | |
| OSDB_C0B | 95H | W | 00H | osd_bitmap_colour0_blue[7:0] | | | | | | | |
| OSDB_C1R | 96H | W | FFH | osd_bitmap_colour1_red[7:0] | | | | | | | |
| OSDB_C1G | 97H | W | 00H | osd_bitmap_colour1_green[7:0] | | | | | | | |
| OSDB_C1B | 98H | W | 00H | osd_bitmap_colour1_blue[7:0] | | | | | | | |
| OSDB_C2R | 99H | W | 00H | osd_bitmap_colour2_red[7:0] | | | | | | | |
| OSDB_C2G | 9AH | W | FFH | osd_bitmap_colour2_green[7:0] | | | | | | | |
| OSDB_C2B | 9BH | W | 00H | osd_bitmap_colour2_blue[7:0] | | | | | | | |
| OSDB_C3R | 9CH | W | 00H | osd_bitmap_colour3_red[7:0] | | | | | | | |
| OSDB_C3G | 9DH | W | 00H | osd_bitmap_colour3_green[7:0] | | | | | | | |
| OSDB_C3B | 9EH | W | FFH | osd_bitmap_colour3_blue[7:0] | | | | | | | |
| OSDB_C4R | 9FH | W | FFH | osd_bitmap_colour4_red[7:0] | | | | | | | |
| OSDB_C4G | A0H | W | FFH | osd_bitmap_colour4_green[7:0] | | | | | | | |
| OSDB_C4B | A1H | W | 00H | osd_bitmap_colour4_blue[7:0] | | | | | | | |
| OSDB_C5R | A2H | W | 00H | osd_bitmap_colour5_red[7:0] | | | | | | | |
| OSDB_C5G | A3H | W | FFH | osd_bitmap_colour5_green[7:0] | | | | | | | |
| OSDB_C5B | A4H | W | FFH | osd_bitmap_colour5_blue[7:0] | | | | | | | |
| OSDB_C6R | A5H | W | FFH | osd_bitmap_colour6_red[7:0] | | | | | | | |
| OSDB_C6G | A6H | W | 00H | osd_bitmap_colour6_green[7:0] | | | | | | | |
| OSDB_C6B | A7H | W | FFH | osd_bitmap_colour6_blue[7:0] | | | | | | | |
| OSDB_C7R | A8H | W | FFH | osd_bitmap_colour7_red[7:0] | | | | | | | |
| OSDB_C7G | A9H | W | FFH | osd_bitmap_colour7_green[7:0] | | | | | | | |
| OSDB_C7B | AAH | W | FFH | osd_bitmap_colour7_blue[7:0] | | | | | | | |
| OSDB_C8R | ABH | W | 40H | osd_bitmap_colour8_red[7:0] | | | | | | | |
| OSDB_C8G | ACH | W | 40H | osd_bitmap_colour8_green[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDB_C8B | ADH | W | 40H | osd_bitmap_colour8_blue[7:0] | | | | | | | |
| OSDB_C9R | AEH | W | 80H | osd_bitmap_colour9_red[7:0] | | | | | | | |
| OSDB_C9G | AFH | W | 00H | osd_bitmap_colour9_green[7:0] | | | | | | | |
| OSDB_C9B | B0H | W | 00H | osd_bitmap_colour9_blue[7:0] | | | | | | | |
| OSDB_C10R | B1H | W | 00H | osd_bitmap_colour10_red[7:0] | | | | | | | |
| OSDB_C10G | B2H | W | 80H | osd_bitmap_colour10_green[7:0] | | | | | | | |
| OSDB_C10B | B3H | W | 00H | osd_bitmap_colour10_blue[7:0] | | | | | | | |
| OSDB_C11R | B4H | W | 00H | osd_bitmap_colour11_red[7:0] | | | | | | | |
| OSDB_C11G | B5H | W | 00H | osd_bitmap_colour11_green[7:0] | | | | | | | |
| OSDB_C11B | B6H | W | 80H | osd_bitmap_colour11_blue[7:0] | | | | | | | |
| OSDB_C12R | B7H | W | 80H | osd_bitmap_colour12_red[7:0] | | | | | | | |
| OSDB_C12G | B8H | W | 80H | osd_bitmap_colour12_green[7:0] | | | | | | | |
| OSDB_C12B | B9H | W | 00H | osd_bitmap_colour12_blue[7:0] | | | | | | | |
| OSDB_C13R | BAH | W | 00H | osd_bitmap_colour13_red[7:0] | | | | | | | |
| OSDB_C13G | BBH | W | 80H | osd_bitmap_colour13_green[7:0] | | | | | | | |
| OSDB_C13B | BCH | W | 80H | osd_bitmap_colour13_blue[7:0] | | | | | | | |
| OSDB_C14R | BDH | W | 80H | osd_bitmap_colour14_red[7:0] | | | | | | | |
| OSDB_C14G | BEH | W | 00H | osd_bitmap_colour14_green[7:0] | | | | | | | |
| OSDB_C14B | BFH | W | 80H | osd_bitmap_colour14_blue[7:0] | | | | | | | |
| OSDB_C15R | C0H | W | 80H | osd_bitmap_colour15_red[7:0] | | | | | | | |
| OSDB_C15G | C1H | W | 80H | osd_bitmap_colour15_green[7:0] | | | | | | | |
| OSDB_C15B | C2H | W | 80H | osd_bitmap_colour15_blue[7:0] | | | | | | | |
| OSD POINTER COLOURS: C3H TO CEH | | | | | | | | | | | |
| OSDP_C0R | C3H | W | 00H | osd_pointer_colour0_red[7:0] | | | | | | | |
| OSDP_C0G | C4H | W | 00H | osd_pointer_colour0_green[7:0] | | | | | | | |
| OSDP_C0B | C5H | W | 00H | osd_pointer_colour0_blue[7:0] | | | | | | | |
| OSDP_C1R | C6H | W | FFH | osd_pointer_colour1_red[7:0] | | | | | | | |
| OSDP_C1G | C7H | W | 00H | osd_pointer_colour1_green[7:0] | | | | | | | |
| OSDP_C1B | C8H | W | 00H | osd_pointer_colour1_blue[7:0] | | | | | | | |
| OSDP_C2R | C9H | W | 00H | osd_pointer_colour2_red[7:0] | | | | | | | |
| OSDP_C2G | CAH | W | FFH | osd_pointer_colour2_green[7:0] | | | | | | | |
| OSDP_C2B | CBH | W | 00H | osd_pointer_colour2_blue[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OSDP_C3R | CCH | W | 00H | osd_pointer_colour3_red[7:0] | | | | | | | |
| OSDP_C3G | CDH | W | 00H | osd_pointer_colour3_green[7:0] | | | | | | | |
| OSDP_C3B | CEH | W | FFH | osd_pointer_colour3_blue[7:0] | | | | | | | |

**Note**

1. X = don’t care.

**Table 16** Colour look-up table and dithering configuration registers (page 10)

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Colour look-up table: 00H to 03H** | | | | | | | | | | | |
| CL_CTRL | 00H | W | —00 0000 | | | write_ hsynced | quick_prog | red_prog | green_prog | blue_prog | cc_on |
| CL_INDEX | 01H | W | 00H | colour_index[7:0] | | | | | | | |
| CL_VALUE_HI | 02H | W | —— ––00 | | | | | | | colour_value[9:8] | |
| CL_VALUE_LO | 03H | W | 00H | colour_value[7:0] | | | | | | | |
| **Temporal dithering: 80H to 83H** | | | | | | | | | | | |
| DT_CTRL | 80H | W | 1––– 1––– | dither_ bypass | | | | dither_out_ bits | | | |
| DT_COLMAP | 81H | W | 11–– ––– | dither_colmap[1:0] | | | | | | | |
| DT_MODE | 82H | W | 00–– –100 | dither_ rand_mode | dither_ rand_mono | | | | dither_idx_ofs_reg[2:0] | | |
| DT_NOISE | 83H | W | 0––– ––0– | dither_ add_noise | | | | | | dither_ noise_mag | |

**Table 17** Output interface configuration registers (page 11); note 1

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Output interface: 01H to 39H, 40H to 4AH, 51H to 59H, 61H to 6AH, 71H to 7AH, 81H to 8AH, 91H to 9AH, A1H to AAH, B1H to BAH, C1H to CAH, D1H to DEH, E1H to EEH and F0H to F7H** | | | | | | | | | | | |
| OI_WX_HI | 01H | W | —— –000 | | | | | | wait_column[10:8] | | |
| OI_WX_LO | 02H | W | 02H | wait_column[7:0] | | | | | | | |
| OI_INVA_DEL | 03H | W | 0000 0000 | pin_drv_inva[2:0] | | | inversion_A_pin_delay[4:0] | | | | |
| OI_INVB_DEL | 04H | W | 0000 0000 | pin_drv_invb[2:0] | | | inversion_B_pin_delay[4:0] | | | | |
| OI_PSX_HI | 05H | W | —— –000 | | | | | | picture_start_x[10:8] | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_PSX_LO | 06H | W | 09H | picture_start_x[7:0] | | | | | | | |
| OI_PSY_HI | 07H | W | —— –000 | | | | | | picture_start_y[10:8] | | |
| OI_PSY_LO | 08H | W | 07H | picture_start_y[7:0] | | | | | | | |
| OI_ASX_HI | 09H | W | —— –000 | | | | | | active_start_x[10:8] | | |
| OI_ASX_LO | 0AH | W | 07H | active_start_x[7:0] | | | | | | | |
| OI_ASY_HI | 0BH | W | —— –000 | | | | | | active_start_y[10:8] | | |
| OI_ASY_LO | 0CH | W | 05H | active_start_y[7:0] | | | | | | | |
| OI_PEX_HI | 0DH | W | —— –000 | | | | | | picture_end_x[10:8] | | |
| OI_PEX_LO | 0EH | W | 54H | picture_end_x[7:0] | | | | | | | |
| OI_PEY_HI | 0FH | W | —— –000 | | | | | | picture_end_y[10:8] | | |
| OI_PEY_LO | 10H | W | 3EH | picture_end_y[7:0] | | | | | | | |
| OI_AEX_HI | 11H | W | —— –000 | | | | | | active_end_x[10:8] | | |
| OI_AEX_LO | 12H | W | 56H | active_end_x[7:0] | | | | | | | |
| OI_AEY_HI | 13H | W | —— –000 | | | | | | active_end_y[10:8] | | |
| OI_AEY_LO | 14H | W | 40H | active_end_y[7:0] | | | | | | | |
| OI_FY_HI | 15H | W | —— –000 | | | | | | last_line[10:8] | | |
| OI_FY_LO | 16H | W | 46H | last_line[7:0] | | | | | | | |
| OI_FX_HI | 17H | W | —— –000 | | | | | | blank_line_length[10:8] | | |
| OI_FX_LO | 18H | W | 5EH | blank_line_length[7:0] | | | | | | | |
| OI_ALX_HI | 19H | W | —— –000 | | | | | | active_line_length[10:8] | | |
| OI_ALX_LO | 1AH | W | 5CH | active_line_length[7:0] | | | | | | | |
| OI_PX_HI | 1BH | W | —— –000 | | | | | | picture_line_length[10:8] | | |
| OI_PX_LO | 1CH | W | 5AH | picture_line_length[7:0] | | | | | | | |
| OI_WM | 1DH | W | —— ––01 | | | | | | | wait_mode[1:0] | |
| OI_B0R | 1EH | W | —00 0011 | | | MSB_align | swap | inv | port_A_conf[2:0] | | |
| OI_B0G | 1FH | W | —00 0001 | | | MSB_align | swap | inv | port_B_conf[2:0] | | |
| OI_B0B | 20H | W | —00 0000 | | | MSB_align | swap | inv | port_C_conf[2:0] | | |
| OI_B1R | 21H | W | —00 0111 | | | MSB_align | swap | inv | port_D_conf[2:0] | | |
| OI_B1G | 22H | W | —00 0101 | | | MSB_align | swap | inv | port_E_conf[2:0] | | |
| OI_B1B | 23H | W | —00 0100 | | | MSB_align | swap | inv | port_F_conf[2:0] | | |
| OI_PAD | 24H | W | 0000 0000 | pin_drv_pa[2:0] | | | pin_delay[4:0] | | | | |
| OI_PBD | 25H | W | 0000 0000 | pin_drv_pb[2:0] | | | pin_delay[4:0] | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_PCD | 26H | W | 0000 0000 | pin_drv_pc[2:0] | | | pin_delay[4:0] | | | | |
| OI_PDD | 27H | W | 0000 0000 | pin_drv_pd[2:0] | | | pin_delay[4:0] | | | | |
| OI_PED | 28H | W | 0000 0000 | pin_drv_pe[2:0] | | | pin_delay[4:0] | | | | |
| OI_PFD | 29H | W | 0000 0000 | pin_drv_pf[2:0] | | | pin_delay[4:0] | | | | |
| OI_CTRL0 | 2AH | W | –10– 0100 | | ivsl1 | ivsl0 | | 0 | OI_enable | power_ down | blank_ mode |
| OI_CTRL1 | 2BH | W | –000 0000 | | PCLK_pin_delay[4:0] | | | | | double_ pixel | PCLK_pol |
| OI_BOC_R | 2CH | W | 00H | border_colour_red[7:0] | | | | | | | |
| OI_BOC_G | 2DH | W | FFH | border_colour_green[7:0] | | | | | | | |
| OI_BOC_B | 2EH | W | 00H | border_colour_blue[7:0] | | | | | | | |
| OI_BLC_R | 2FH | W | FFH | blank_colour_red[7:0] | | | | | | | |
| OI_BLC_G | 30H | W | 00H | blank_colour_green[7:0] | | | | | | | |
| OI_BLC_B | 31H | W | 00H | blank_colour_blue[7:0] | | | | | | | |
| OI_G0ASX_HI | 32H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G0ASX_LO | 33H | W | 01H | point1_x[7:0] | | | | | | | |
| OI_G0ASY_HI | 34H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G0ASY_LO | 35H | W | 01H | point1_y[7:0] | | | | | | | |
| OI_G0AEX_HI | 36H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G0AEX_LO | 37H | W | 25H | point2_x[7:0] | | | | | | | |
| OI_G0AEY_HI | 38H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G0AEY_LO | 39H | W | 02H | point2_y[7:0] | | | | | | | |
| OI_G0AC | 40H | W | —— 0100 | | | | | pol_CSG 0A+0B | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G0BSX_HI | 41H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G0BSX_LO | 42H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G0BSY_HI | 43H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G0BSY_LO | 44H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G0BEX_HI | 45H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G0BEX_LO | 46H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G0BEY_HI | 47H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G0BEY_LO | 48H | W | 00H | point2_y[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_G0BC | 49H | W | —— –000 | | | | | | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G0BD | 4AH | W | X000 0000 | pin_drv_csg0[2:0] | | | pin_delay[4:0] | | | | |
| OI_G1ASX_HI | 51H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G1ASX_LO | 52H | W | 03H | point1_x[7:0] | | | | | | | |
| OI_G1ASY_HI | 53H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G1ASY_LO | 54H | W | 01H | point1_y[7:0] | | | | | | | |
| OI_G1AEX_HI | 55H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G1AEX_LO | 56H | W | 05H | point2_x[7:0] | | | | | | | |
| OI_G1AEY_HI | 57H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G1AEY_LO | 58H | W | 46H | point2_y[7:0] | | | | | | | |
| OI_G1AC | 59H | W | —— 0000 | | | | | pol_CSG 1A+1B | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G1BSX_HI | 61H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G1BSX_LO | 62H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G1BSY_HI | 63H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G1BSY_LO | 64H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G1BEX_HI | 65H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G1BEX_LO | 66H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G1BEY_HI | 67H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G1BEY_LO | 68H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G1BC | 69H | W | —— –000 | | | | | | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G1BD | 6AH | W | X000 0000 | pin_drv_csg1[2:0] | | | pin_delay[4:0] | | | | |
| OI_G2SX_HI | 71H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G2SX_LO | 72H | W | 06H | point1_x[7:0] | | | | | | | |
| OI_G2SY_HI | 73H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G2SY_LO | 74H | W | 05H | point1_y[7:0] | | | | | | | |
| OI_G2EX_HI | 75H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G2EX_LO | 76H | W | 56H | point2_x[7:0] | | | | | | | |
| OI_G2EY_HI | 77H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G2EY_LO | 78H | W | 40H | point2_y[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_G2C | 79H | W | ——0 1000 | | | | invol_ CSG3 | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G2D | 7AH | W | X000 0000 | pin_drv_csg2[2:0] | | | pin_delay[4:0] | | | | |
| OI_G3SX_HI | 81H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G3SX_LO | 82H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G3SY_HI | 83H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G3SY_LO | 84H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G3EX_HI | 85H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G3EX_LO | 86H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G3EY_HI | 87H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G3EY_LO | 88H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G3C | 89H | W | —— 0000 | | | | | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G3D | 8AH | W | X000 0000 | pin_drv_csg3[2:0] | | | pin_delay[4:0] | | | | |
| OI_G4SX_HI | 91H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G4SX_LO | 92H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G4SY_HI | 93H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G4SY_LO | 94H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G4EX_HI | 95H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G4EX_LO | 96H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G4EY_HI | 97H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G4EY_LO | 98H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G4C | 99H | W | ——0 0000 | | | | invol_ CSG5 | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G4D | 9AH | W | X000 0000 | pin_drv_csg4[2:0] | | | pin_delay[4:0] | | | | |
| OI_G5SX_HI | A1H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G5SX_LO | A2H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G5SY_HI | A3H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G5SY_LO | A4H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G5EX_HI | A5H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G5EX_LO | A6H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G5EY_HI | A7H | W | —— –000 | | | | | | point2_y[10:8] | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_G5EY_LO | A8H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G5C | A9H | W | —— 0000 | | | | | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G5D | AAH | W | X000 0000 | pin_drv_csg5[2:0] | | | pin_delay[4:0] | | | | |
| OI_G6SX_HI | B1H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G6SX_LO | B2H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G6SY_HI | B3H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G6SY_LO | B4H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G6EX_HI | B5H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G6EX_LO | B6H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G6EY_HI | B7H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G6EY_LO | B8H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G6C | B9H | W | —— 0000 | | | | | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G6D | BAH | W | X000 0000 | pin_drv_csg6[2:0] | | | pin_delay[4:0] | | | | |
| OI_G7SX_HI | C1H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G7SX_LO | C2H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G7SY_HI | C3H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G7SY_LO | C4H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G7EX_HI | C5H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G7EX_LO | C6H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G7EY_HI | C7H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G7EY_LO | C8H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G7C | C9H | W | —— 0000 | | | | | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G7D | CAH | W | X000 0000 | pin_drv_csg7[2:0] | | | pin_delay[4:0] | | | | |
| OI_G8SX_HI | D1H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G8SX_LO | D2H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G8SY_HI | D3H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G8SY_LO | D4H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G8EX_HI | D5H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G8EX_LO | D6H | W | 00H | point2_x[7:0] | | | | | | | |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_G8EY_HI | D7H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G8EY_LO | D8H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G8C | D9H | W | 0000 0000 | skip_mode | point3_ toggle | point3_ on/off | point3_ frm/line | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G8D | DAH | W | X000 0000 | pin_drv_csg8[2:0] | | | pin_delay[4:0] | | | | |
| OI_G8SPX_HI | DBH | W | —— –000 | | | | | | point3_x[10:8] | | |
| OI_G8SPX_LO | DCH | W | 00H | point3_x[7:0] | | | | | | | |
| OI_G8SPY_HI | DDH | W | —— –000 | | | | | | point3_y[10:8] | | |
| OI_G8SPY_LO | DEH | W | 00H | point3_y[7:0] | | | | | | | |
| OI_G9SX_HI | E1H | W | —— –000 | | | | | | point1_x[10:8] | | |
| OI_G9SX_LO | E2H | W | 00H | point1_x[7:0] | | | | | | | |
| OI_G9SY_HI | E3H | W | —— –000 | | | | | | point1_y[10:8] | | |
| OI_G9SY_LO | E4H | W | 00H | point1_y[7:0] | | | | | | | |
| OI_G9EX_HI | E5H | W | —— –000 | | | | | | point2_x[10:8] | | |
| OI_G9EX_LO | E6H | W | 00H | point2_x[7:0] | | | | | | | |
| OI_G9EY_HI | E7H | W | —— –000 | | | | | | point2_y[10:8] | | |
| OI_G9EY_LO | E8H | W | 00H | point2_y[7:0] | | | | | | | |
| OI_G9C | E9H | W | 0000 0000 | skip_mode | point3_ toggle | point3_ on/off | point3_ frm/line | pol | frame/line | point2_ tog/reset | point1_ tog/set |
| OI_G9D | EAH | W | X000 0000 | pin_drv_csg9[2:0] | | | pin_delay[4:0] | | | | |
| OI_G9SPX_HI | EBH | W | —— –000 | | | | | | point3_x[10:8] | | |
| OI_G9SPX_LO | ECH | W | 00H | point3_x[7:0] | | | | | | | |
| OI_G9SPY_HI | EDH | W | —— –000 | | | | | | point3_y[10:8] | | |
| OI_G9SPY_LO | EEH | W | 00H | point3_y[7:0] | | | | | | | |
| OI_PWM0 | F0H | W | 00H | PWM[7:0] | | | | | | | |
| OI_PWM1 | F1H | W | ——X X000 | | | | PWM_HS_ sync | PWM_pol | PWM_DIV[2:0] | | |
| OI_FCR | F2H | W | 00H | frame_col[23:16] | | | | | | | |
| OI_FCG | F3H | W | 00H | frame_col[15:8] | | | | | | | |
| OI_FCB | F4H | W | FFH | frame_col[7:0] | | | | | | | |
| OI_FC_EN | F5H | W | —— ——0 | | | | | | | | enable_ frame_ generator |

| REGISTER | ADR | R/W | RESET | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| OI_PWMD | F6H | W | 000– ——— | pin_drv_pwm[2:0] | | | | | | | |
| OI_WC | F6H | R | 00H | wait_count[7:0] | | | | | | | |
| OI_PCLKD | F7H | W | 0000 00—— | pin_drv_pclk[2:0] | | | pin_drv_outen[2:0] | | | | |

**Note**

1. X = don’t care.

### 7.2 Device ID

The readable parameter device_id contains the IC version code. The current version returns the code 131CH.

### 7.3 Initialization

The external Power-on reset is active LOW and applied to pin $\overline{\text{RST}}$.

Front-end, back-end and the output interface can be switched into the reset state individually by the I²C-bus programming using reset_fclk, reset_bclk and reset_oif at register GC_RESET (FCH). Each domain reset is active if the corresponding programming bit is set to logic 1.

### 7.4 Clock management

All clock management configuration registers are mapped to register page 0.

A block diagram of the clock distribution is given in Fig.5. The clock source for the decoupling FIFO is selected by fifo_fclk. If fifo_fclk is set to logic 1, the front-end clock is applied to the decoupling FIFO; otherwise the back-end clock is used.

The decoupling FIFO always has to be supplied with the clock signal of the higher clock rate.

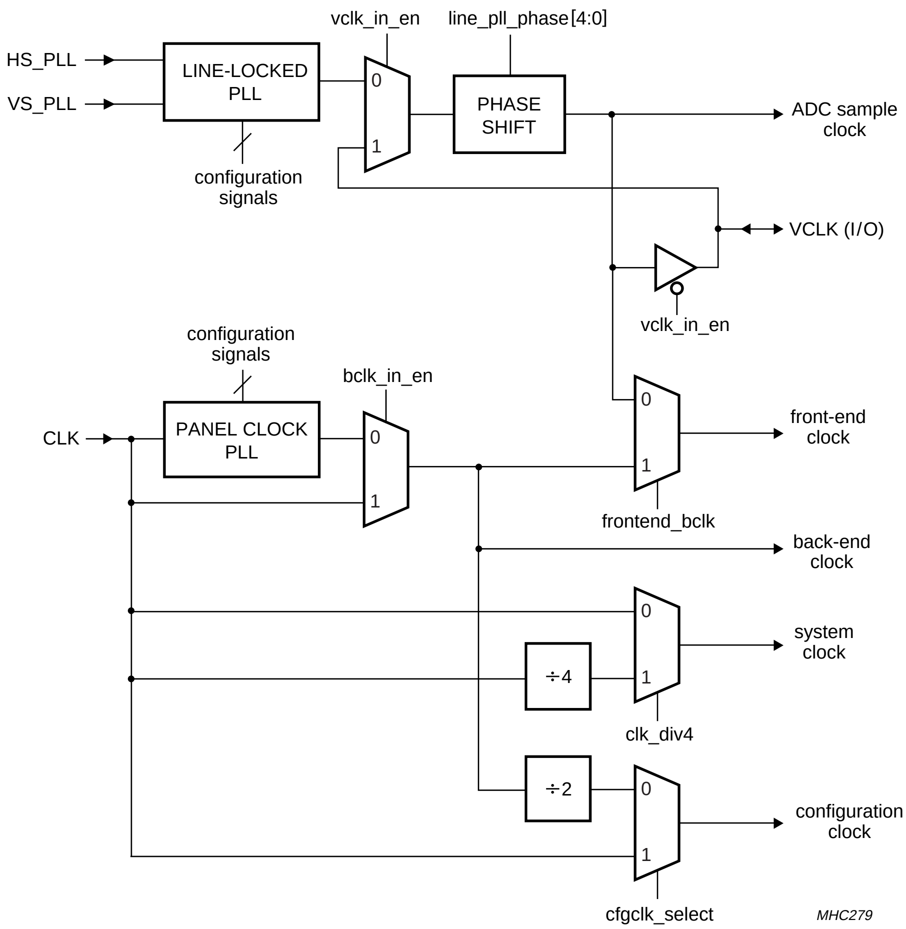



Fig.5 Clock distribution.

### 7.4.1 CLOCK SIGNALS

#### *7.4.1.1 System clock*

The system clock is applied to pin CLK and is used to drive the internal control structures and block configuration, and serves as input for the panel clock PLL. The maximum clock rate is 50 MHz.

The system clock is directly taken from pin CLK if clk_div4 is set to logic 0; otherwise the system clock is derived from the clock signal at pin CLK additionally divided by 4 as shown in Table 18.

**Table 18** System clock switching modes

| clk_div4 | SYSTEM CLOCK | DESCRIPTION |
|---|---|---|
| 0 | CLK | direct input |
| 1 | ¼CLK | divided by 4 |

#### *7.4.1.2 Back-end clock*

The back-end clock is the pixel clock used in data processing behind the decoupling FIFO. Possible clock rates lie between 5 and 100 MHz in case of single pixel panel output, but it is identical with the panel clock; if using double pixel mode it equals twice the panel clock.

The clock signal is generated by the panel clock PLL based on the system clock if bclk_in_en is set to logic 0; otherwise the signal applied externally to pin CLK is used as system clock (see Table 19).

**Table 19** Back-end clock switching modes

| bclk_in_en | BACK-END CLOCK | DESCRIPTION |
|---|---|---|
| 1 | CLK | external clock |
| 0 | PLL clock | internal clock generation |

#### *7.4.1.3 Front-end clock*

The front-end clock is the pixel clock of the input section and is generated by the line PLL for the analog RGB input. The front-end clock rate can be up to 110 MHz.

Pin VCLK is switched as output for the used clock signal.

An externally generated clock signal can also be connected to pin VCLK if vclk_in_en is set to logic 1. Alternatively, the back-end clock can be selected as front-end clock, which is particularly needed if the picture generator is used without an external clock source. Front-end clock modes are shown in Table 20.

**Table 20** Front-end clock switching modes; note 1

| frontend_bclk | vclk_in_en | FRONT-END CLOCK | DESCRIPTION |
|---|---|---|---|
| 1 | X | back-end clock | initialization |
| 0 | 1 | VCLK | external clock generation |
| 0 | 0 | line PLL clock | internal clock generation |

**Note**

1. X = don’t care.

#### *7.4.1.4 Configuration clock*

The internal configuration clock is driving the configuration parameters section of all modules. It is usually running at half the back-end clock frequency. If somehow the back-end clock is not usable for the configuration, the system clock could be used to drive the configuration clock instead. The selection of the configuration clock source could either be done automatically monitoring the back-end clock or forced manually if this is desired. For power saving issues the configuration clock is powered-down during inactive periods when no data is received or requested via the I$^2$C-bus interface.
See Table 21 for configuration clock switching options.

**Table 21** Configuration clock switching modes

| cfgclk_select | CONFIGURATION CLOCK | DESCRIPTION |
|---|---|---|
| 0 | half back-end clock | application (stable back-end clock) |
| 1 | CLK | initialization |

#### 7.4.2 CLOCK ACTIVATION CONTROL

The clock signals of auto-adjustment, downscaler, upscaler and OSD module are powered-down automatically during inactivity if programming bits aaclk_auto, dscclk_auto, uscclk_auto and osdclk_auto respectively in register CD_CLK_AUTO (11H) are set to logic 1. Otherwise the clock signals are switched on and off according to the state of bits aaclk_on, dscclk_on, uscclk_on and osdclk_on respectively in register CD_CLK_EN (10H).

The general configuration and the OSD configuration clock signal are also powered-down during inactivity unless forced active, when cfgclk_on or osd_cfgclk_on respectively (CD_CLK_EN, 10H) is set to logic 1.

When automatic activation is selected, each clock signal is active during either power-on or the programmable reset of the specific domain and whenever the concerned module is activated.

#### 7.4.3 PLL PROGRAMMING

The SAA6713AH contains two PLLs:

- Line-locked PLL generating the sample clock from the hsync signal (see Fig.6)
- PLL running on the system clock generating the panel clock (see Fig.7).

The PLL programming registers are mapped to register page 0.

The PLLs are activated by pll_en and line_pll_en and the back-end clock PLL pre-divider by pll_pre_div_en at register CD_PLL_CTRL (20H).

Bits line_pll_vs_pol and line_pll_hs_pol define the polarity of the vertical and horizontal sync inputs. Each bit has to be set to logic 1 in case of positive (active HIGH) polarity of the corresponding sync signal; otherwise to logic 0.

The outputs for the pre-divider, n-divider and m-divider ratios are set accordingly to bits pll_pre_div, pll_m_div, pll_n_div, line_pll_m_div and line_pll_n_div at registers CD_PLL_P_HI to CD_LPLL_LO (21H to 26H).

The pll_n_div is a programmable divider between 100 to 4096. The relation between hsync and pll_clk is: pll_clk = pll_n_div × hsync. The frequency of the oscillator should be selected at minimum two times pll_clk.

The pll_m_div is a programmable divider between ‘00’ = 1, ‘01’ = 2, ‘10’ = 2, ‘11’ = 4 and limits the current controlled oscillator tuning range.

The line PLL clock is finally phase shifted as defined in steps of 11.25 degrees by line_pll_phase at register CD_LPLL_PHA (27H).

For the auto-adjustment phase distortion measurement register CD_LPLL_PD contains an alternative phase value pd_pll_phase for the line PLL. Parameter phase_auto enables switching between both phase values controlled by the auto-adjustment if set to logic 1, or manual selection by phase_select.

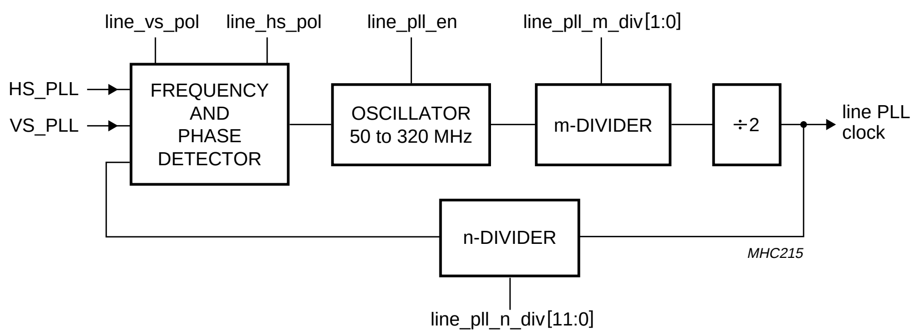



Fig.6 Line PLL block diagram.

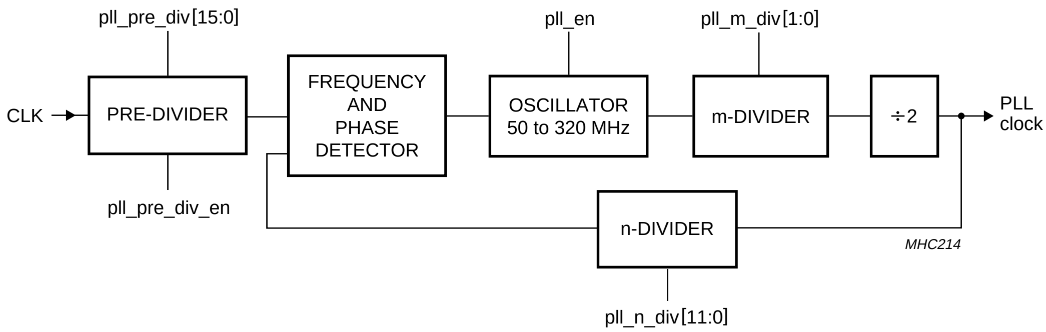



Fig.7 PLL block diagram.

### 7.5 Synchronization pulse distribution

The line-locked PLL, input interface and mode detection are provided with horizontal and vertical synchronization pulse signals (HSYNC and VSYNC). Signal switching is controlled by configuration registers SYNC_SEL (18H at page 0) and SYNC_DIS (19H at page 0). A composite sync decoder and hsync regeneration can be inserted. Possible selections and the concerned configuration parameters are shown in Fig.8 and described more detailed in the Sections 7.5.1 to 7.5.5.

#### 7.5.1 COMPOSITE SYNC INPUT

The composite sync decoder input is selected by sog_en between the separated SOG provided by the sync-on-green slicer and a composite sync applied at input pin HSYNC (see Table 22). The sync-on-green slicer has to be enabled by setting sync_on_green_en in register ADC_CTRL (00H at page 1) to logic 1.

To provide a stable hsync during the vsync, the sync-on-green slicer might have to be disabled during the vsync which is performed automatically if sog_vs_disable is set to logic 1; otherwise the sync-on-green slicer is constantly enabled.

The composite sync decoder will regenerate hsync and vsync signals for internal use. Figure 9 shows the composite sync modes that can be used. The maximum number of equalizing pulses (csync-3 and csync-4) may not exceed 30.

**Table 22** Composite sync decoder input selection

| sog_en | CSYNC |
|---|---|
| 1 | SOG |
| 0 | HSYNC |

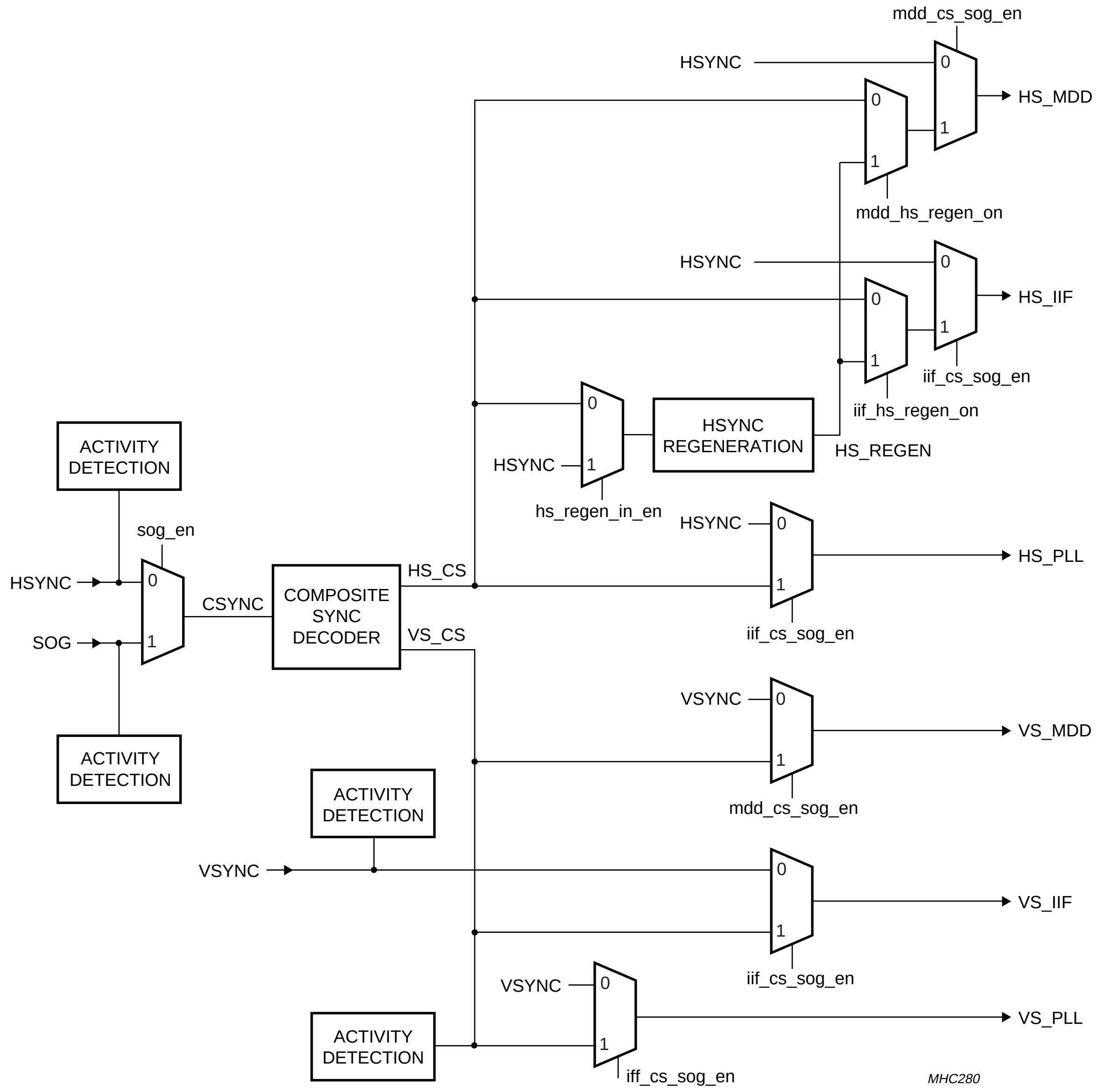



Fig.8 Synchronization pulse distribution.

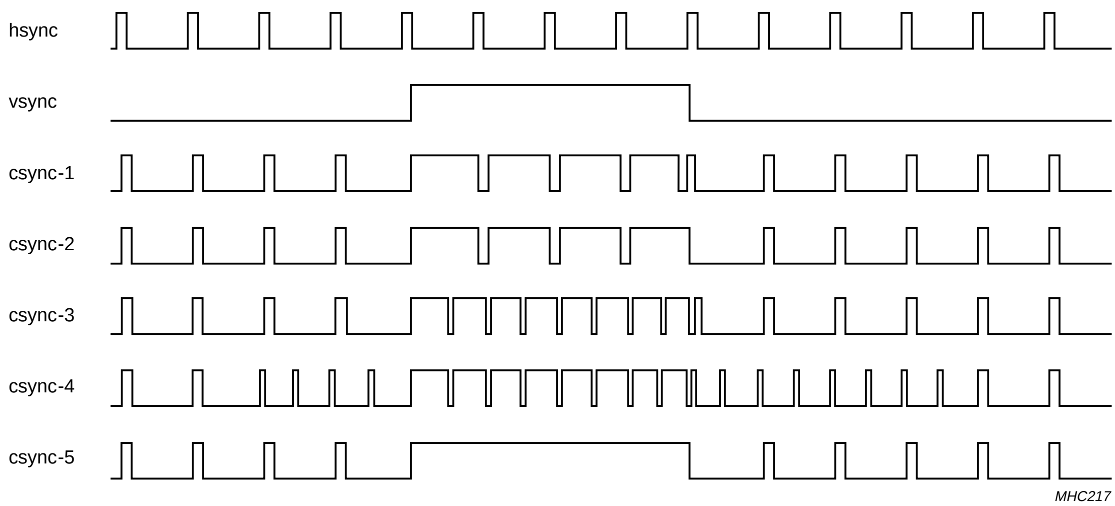



Fig.9 Supported composite sync modes.

### 7.5.2 HSYNC REGENERATION

The hsync regeneration reproduces a regular hsync, e.g. in case of equalizing pulses or an absent hsync during vsync. The input selection is shown in Table 23.

**Table 23** Hsync regeneration input selection

| hs_regen_in_en | HS_REGEN |
|---|---|
| 0 | HS_CS |
| 1 | HSYNC |

### 7.5.3 SELECTION OF SYNCS FOR LINE-LOCKED PLL

The source signals of the line-locked PLL are selected according to Table 24 as either HSYNC and VSYNC from the input pins or the composite sync decoder outputs HS_CS and VS_CS.

**Table 24** Line-locked PLL sync selection

| iif_cs_sog_en | HS_PLL | VS_PLL |
|---|---|---|
| 0 | HSYNC | VSYNC |
| 1 | HS_CS | VS_CS |

### 7.5.4 SELECTION OF SYNCS FOR MODE DETECTION AND INPUT INTERFACE

The output selection for input interface and mode detection allows to choose between the input signals HSYNC and VSYNC, composite sync decoder outputs HS_CS and VS_CS. The regenerated hsync HS_REGEN can be selected as source (see Tables 25 and 26).

**Table 25** Mode detection sync selection; note 1

| mdd_cs_ sog_en | mdd_hs_ regen_on | HS_MDD | VS_MDD |
|---|---|---|---|
| 0 | X | HSYNC | VSYNC |
| 1 | 0 | HS_CS | VS_CS |
| 1 | 1 | HS_REGEN | |

**Note**

1. X = don’t care.

**Table 26** Input interface sync selection; note 1

| iif_cs_sog_en | iif_hs_ regen_on | HS_IIF | VS_IIF |
|---|---|---|---|
| 0 | X | HSYNC | VSYNC |
| 1 | 0 | HS_CS | VS_CS |
| 1 | 1 | HS_REGEN | |

**Note**

1. X = don't care.

#### 7.5.5 PIN VSYNC CONFIGURATION

Besides serving as input for an external vertical synchronization pulse VSYNC can be switched as output of the vsync internally derived from (not shown in Fig.8):

- Sync-on-green slicer (SOG)
- Composite sync decoder (VS_CS).

The I/O direction of pin VSYNC is selected by vsync_out_en of register SYNC_SEL (18H at page 0). In case of output mode, the source is selected by sog_out_en of register SYNC_SEL according to Table 27.

**Table 27** Pin VSYNC switching modes; note 1

| vsync_out_en | sog_out_en | DIRECTION | VSYNC |
|---|---|---|---|
| 0 | X | input | external |
| 1 | 1 | output | SOG |
| 1 | 0 | | VS_CS |

**Note**

1. X = don't care.

### 7.6 Interrupt generation

An interrupt signal is provided at output pin $\overline{\text{INT}}$ (active LOW). The state of $\overline{\text{INT}}$ is based on mode detection, auto-adjustment, OSD, decoupling FIFO and output interface interrupt conditions shown in Table 28.

**Table 28** Interrupt conditions and description

| INTERRUPT | SUBMODULE | DESCRIPTION |
|---|---|---|
| int_mode | mode detection | change of input video mode detected |
| int_auto | auto-adjustment | auto-adjustment finished |
| int_fifo | decoupling FIFO | FIFO overflow |
| int_osd | OSD | end of programmed OSD frame sequence |
| int_oif | output interface | FIFO underflow |
| int_iif | input interface | line jitter occurs (hsync jitter detection) |

Interrupt output pin $\overline{\text{INT}}$ is set LOW (active) whenever one or more of the interrupt flags is HIGH. The interrupt flags are set HIGH, when the corresponding interrupt condition is met:

- The mode detection interrupt flag is set HIGH when one of the mode measurement bits toggles or a value changes significantly at the vsync or in case of vsync or hsync jitter, depending on which of the conditions are enabled (see Section 7.10.1).
- The auto-adjustment interrupt flag is set HIGH in the moment an auto-adjustment measurement finishes, indicating the result values can be read out.
- The decoupling FIFO interrupt flag is set HIGH whenever the decoupling FIFO is full, indicating that the output timing is too slow and a change of the timing is required; otherwise a corrupt output picture will occur.
- The OSD interrupt flag is set HIGH every time a pointer animation frame sequence ends to allow to switch the displayed icon and program the icon for the next turn (see Section 7.13.3).
- The output interface interrupt flag is set HIGH when the pixel stream to the output interface is broken, indicating that the output pixel or line rate is too fast.
- The hsync jitter interrupt flag (int_iif) is set HIGH when line jitter at the analog video input occurs more than a number of times specified in the register II_HJIT, indicating that the other clock edge should be used to sample the hsync and vsync signal.

The interrupt flags are accessible at the global interrupt state register GC_INT_STAT (FEH) and are readable. The flags are only cleared (set to LOW) if a logic 1 is written into the corresponding bit in GC_INT_STAT.

The interrupt conditions are maskable by the corresponding programming bit in GC_INT_MASK (FDH); a logic 1 is enabling the particular interrupt condition.

### 7.7 Triple analog-to-digital converter

The integrated triple ADC samples analog RGB signals of up to 110 MHz with a resolution of 8 bits per colour component and provide automatic brightness and contrast control (see Fig.11). The sample clock is generated by the line-locked PLL (see Section 7.4.3), but can also be applied externally. The triple ADC is automatically enabled, when analog RGB is selected as input source.

The time frames for the ADC automatic brightness and gain control are defined by clamp and gain correction pulses generated by the input interface. During these times the ADCs adjust brightness and gain according to the programmable brightness and contrast values defined by adc_red_brightness to adc_blue_contrast at registers ADC_R_BRI to ADC_B_CON (01H to 06H at page 1), that have to be provided in 2s-complement form between –128 (80H) and +127 (7FH).

Not all combinations of contrast and brightness settings are allowed. Combining very low contrast (low gain) together with low brightness (more black than black) is not allowed. These combinations would result in a very low input DC level, which would result in the clamp circuit going out of saturation. This would lead to unpredictable behaviour of the clamp level. The allowed region for the gain value is limited between 27 and 110.

The clamp and gain correction pulse generation is programmed via registers II_ADC_CTRL and II_CLAMP_ON to II_GAINC_OFF (02H to 06H at page 4).

Clamp pulse generation is enabled by clamp_en. The beginning of the clamp pulse CLAMP is marked by clamp_on_delay as an offset to the second edge of the hsync pulse, the end by clamp_off_delay as shown in Fig.10. The polarity of CLAMP is given with clamp_pol; logic 1 is HIGH active and logic 0 is LOW active. During the clamp pulse, that should fall into the hsync backporch, the ADCs each match the sampled black level output value to the value given by adc_red_brightness, adc_green_brightness and adc_blue_brightness respectively.

The gain correction pulse GAINC is the delayed hsync. The first edge of the hsync is delayed by gainc_on_delay and the second edge by gainc_off_delay (see Fig.10). The polarity is programmed by gainc_pol; logic 1 is HIGH active and logic 0 is LOW active. The gain correction pulse generation is enabled by setting gainc_en. During gain correction the ADC inputs are connected to a reference voltage and by gain adjustment the output is matched to adc_red_contrast, adc_green_contrast and adc_blue_contrast.

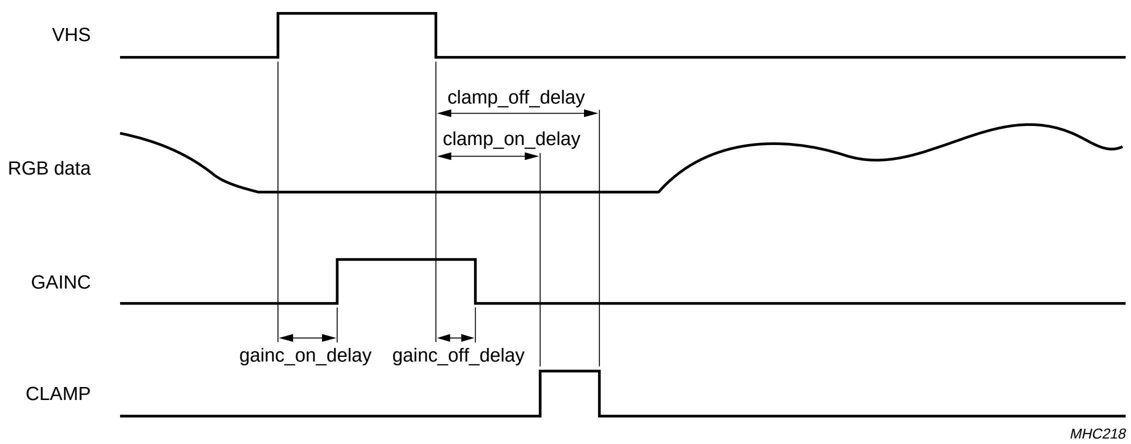



Fig.10 Gain adjustment and clamp pulse generation.

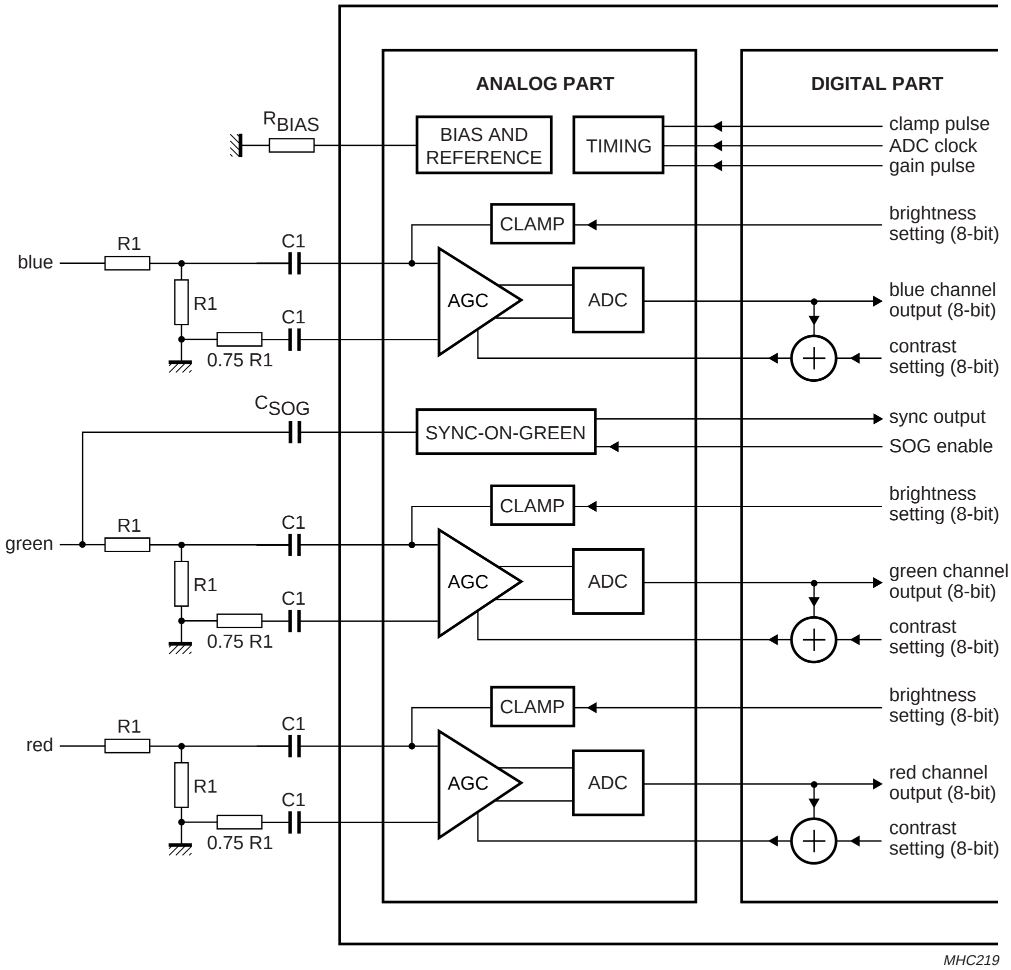



Fig.11 Analog video input block diagram.

### 7.8 Input interface

The input interface selects video data either provided by the ADCs or externally applied and extracts the input picture for processing. The sample window position and size is programmable, using vertical and horizontal synchronization signals as reference. Alternatively, the picture generator can generate different test pictures with programmable size and horizontal and vertical blanking length.

All input interface programming registers are mapped to the $I^2C$-bus configuration register page 4.

#### 7.8.1 INPUT SELECTION

The input source is selected by ext_select (register II_CTRL, address 00H) as shown in Table 29. In case of parallel RGB input, the R component has to be provided at ports PA7 (MSB) to PA0 (LSB) in 8-bit format (range 0 to 255), G and B component similarly at ports PB7 to PB0 and ports PC7 to PC0, respectively. The input source can only be changed in a functional reset (see Section 7.3).

The clock signal edge used to sample the data inputs is specified by ext_clk_edge. If ext_clk_edge is set to logic 1 data is sampled on the rising front-end clock edge; otherwise on the falling front-end clock edge. If convert_2s is set to logic 1 the incoming data is expected to be in 2s-complement form from –128 (80H) to +127 (7FH); otherwise input data is treated as unsigned values from 0 to 255. Data from the internal ADCs is always in 2s-complement form.

To enable the input interface in_form_on has to be set to logic 1; otherwise no data will be provided for processing. If the picture generator is active, the input formatter will always provide generated data.

**Table 29** Input source selection

| ext_select | INPUT SOURCE |
|---|---|
| 1 | parallel RGB |
| 0 | analog RGB |

#### 7.8.2 SYNCHRONIZATION SIGNALS

The synchronization pulses are used to identify the frame outline. The sync signals for the input interface are provided by the sync distribution. The complete description of sync switching options is given in Section 7.5. If analog or parallel RGB input mode is used, the vertical synchronization pulse (vsync) is connected to pin VSYNC and the horizontal synchronization pulse (hsync) to pin HSYNC. A composite synchronization signal is connected to pin HSYNC. Pin VSYNC can then serve as an output for the generated vertical synchronization pulse.

The polarities of hsync and vsync are defined by vs_pol and hs_pol. In case of active HIGH polarity, the corresponding bit has to be set to logic 1; otherwise to logic 0.

If sync_clk_edge is set to logic 1 all synchronization signals are sampled with the rising front-end clock signal edge; otherwise with falling edge.

If delay_vs is set to logic 1, the vsync is delayed in relation to the hsync to prevent line jitter if both occur at the same time, which is monitored by the mode detection.

#### 7.8.3 DEFINITION OF SAMPLE WINDOW

The sample window is defined by in_v_offset, in_h_offset, in_v_length and in_h_length. The vertical offsets are measured from the trailing edge of the vsync pulse. The horizontal offsets are measured from either the first edge of the hsync pulse if hsync_edge is set to logic 1, or the second edge if hsync_edge is set to logic 0. Figure 12 shows the horizontal offset for the case hsync_edge is set to logic 0. If both offsets are set to value 0H, sampling will start with the first pixel in the first line (see Fig.13).

The length defines width and height of the sampled frame. The vertical sample offset and length are given in lines and the horizontal offset and length are measured in pixels.

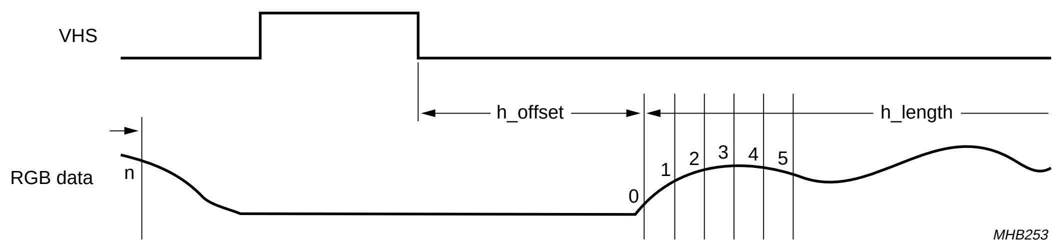



Fig.12 RGB data sampling (hsync_edge = 0).

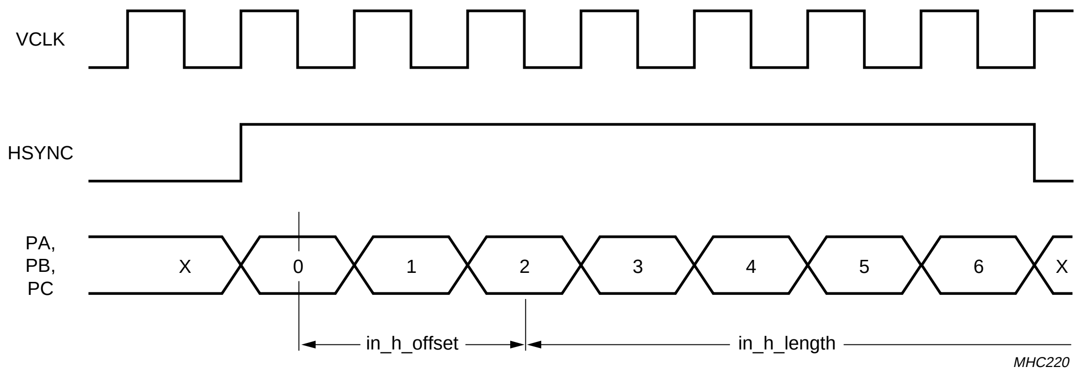



Fig.13 RGB input port timing using a data enable signal (connected to HSYNC).

### 7.8.4 INTERLACED INPUT

Sampling of interlaced RGB data is enabled by interlace_on. The polarity of the input fields is determined by the number of hsyncs within a frame. The even fields are expected to contain an additional line if reverse_field_id is set to logic 0, or to contain one line less if reverse_field_id is set to logic 1.

In the interlaced input mode, the vertical sampling length parameter in_v_length has to be programmed with the length of the actual field (is half the frame length).

For de-interlacing, the upscaler has to be programmed accordingly.

### 7.8.5 PICTURE GENERATOR

The input interface contains a picture generator, that can be used to apply simple test pictures instead of using the ADC. A front-end clock has to be provided (see Section 7.4.1).

The picture generator is active when test_pic_on is logic 1. It generates a picture of the size defined by in_h_length and in_v_length with additional blanking. The total line length and number of lines are defined by h_length_total and v_length_total. The input interface sample offset is without effect when using the picture generator.

The picture generator consists of a border generation, a vertical and a horizontal ramp and ripple generator, that work independently. The two ramp and ripple generators can be activated separately for each RGB colour component. If h_ramp_r, h_ramp_g, h_ramp_b, v_ramp_r, v_ramp_g or v_ramp_b are set to logic 1, the corresponding ramp and ripple pattern is applied to the corresponding colour component; otherwise the pattern does not contribute to the colour component. If white_border is logic 1, then the border generator is activated for all colours. The border, horizontal and vertical ramp and ripple generator outputs are added up for each colour component. Additionally, all colour components are bit reversed if invert is set to logic 1.

Both ramp and ripple pattern generators work in the same way, only the horizontal generator is based on the column position and the vertical generator on the line number. The ramp and ripple generation is shown in Fig.14 for the example of the horizontal generator.

The first step size (h_step1 or v_step1) defines the interval after which the increment value (h_colour_inc or v_colour_inc) is added to the current colour. If the second step size (h_step2 or v_step2) is set to 0, the increment is repeatedly added after the first step size interval. If the second step size is not 0, after the increment value was added the second step size defines the position where the decrement value is subtracted from the current colour. After this the first step size and the increment is applied again and so on. Range over or underflows are not suppressed and cause the colour values to wrap around.

7.8.6 HSYNC JITTER DETECTION

For certain sampling phases the hsync is sampled at its edge and thus unstable. This jitter is detected and another sampling clock edge can be used to avoid it. To detect hsync sample jitter the interval between hsyncs in sample clock cycles is monitored. If the length varies, hsync jitter is detected. As the sample jitter can only change the line length by a maximum of two cycles, only the lowest two bits of the line length have to be considered. If the current line length differs from the previous line, line jitter occurred. The differences of line lengths within a frame are accumulated and the hsync jitter interrupt may be generated when a certain level (hs_jitter_th) is exceeded. During normal operation the jitter detection is only active during the sampled area of the input frame, because the clock rate of the PLL generated sample clock might slightly vary during vsyncs. The detection circuit is active at all times during reset or when the input interface is disabled. For the interrupt a state and an enable register exists, as well as a clear flag. The interrupt is level-based, so every frame after a certain number of occurrences until the next vsync the interrupt state is set to logic 1. The jitter detection does not work correctly without a vsync signal.

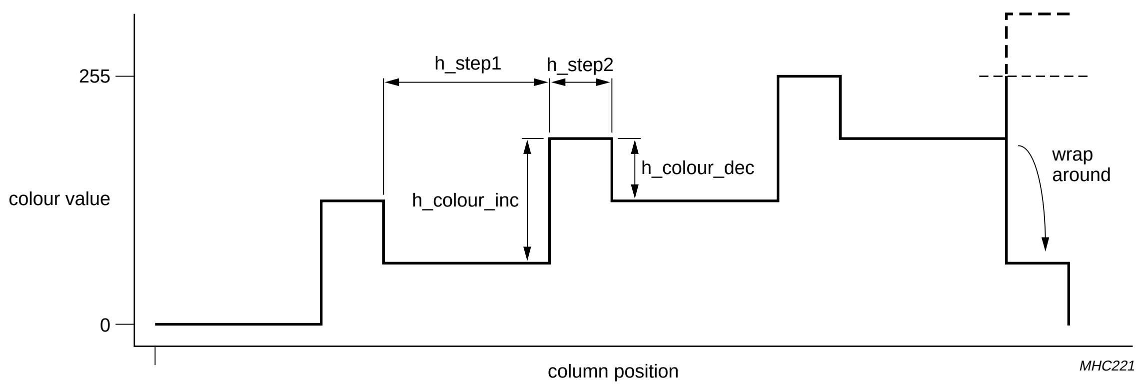



Fig.14 Picture generator ramp and ripple pattern (horizontal generator).

### 7.9 Colour processing

The colour processing performs brightness and contrast adjustment. A programmable offset and gain factor is applied to each RGB colour component. Additional gain and offset values can be applied to the pixel data, not affecting R, G and B components separately, but all components at the same time. Luminance and chrominance of the pixel data can be directly adjusted, which allows true brightness, contrast and colour saturation using single parameters. Register CP_GAIN_Y controls the contrast and CP_OFFS_Y controls the brightness level; both without affecting the colour temperature. Registers CP_GAIN_CB, CP_OFFS_CB, CP_GAIN_CR and CP_OFFS_CR specify gain and offset values for the red and blue saturation of the RGB data. The colour saturation can be shifted simply by using both gain values.

The gain and offset values are specified by the 8-bit configuration registers CP_GAIN_Y to CP_OFFS_B (address 00H to 0BH at page 5). The offset values offset_y, offset_cb and offset_cr for Y-$C_B$-$C_R$ and offset_r, offset_g and offset_b for RGB colour space are given in the range from −128 (80H) to +127 (7FH) in 2s-complement form. The gain factors gain_y, gain_cb and gain_cr as well as gain_r, gain_g and gain_b are given in unsigned form, 128 (80H) representing a factor of 1.0.

### 7.10 RGB mode detection and auto-adjustment

The SAA6713AH can be used to build up auto-scan systems using an external microcontroller. Therefore, information about the input resolution and timing are measured by the SAA6713AH that can be read out via the $I^2C$-bus.

Provided information can be divided into mode detection information to determine the actual RGB input mode and various auto-adjustment features to support the adjustment of the setting of the SAA6713AH to the new mode.

#### 7.10.1 MODE DETECTION

The mode detection determines mode characteristics of the selected video input. The information is provided at the readable $I^2C$-bus registers and changes in the values can trigger the interrupt. All the mode detection $I^2C$-bus registers are mapped to register page 2. The mode detection uses the back-end clock and cannot run without a present back-end clock. The mode detection is enabled by setting md_on to logic 1.

The source of the synchronization pulse signals used by the mode detection is selected by the sync distribution as described in Section 7.5.4 (HS_MDD and VS_MDD).

The absence of synchronization pulses is indicated by the flags no_vsync and no_hsync. If the corresponding synchronization signal cannot be detected, the flags are set to logic 1; otherwise to logic 0. It should be noted that the hsync is considered undetected, whenever there are more than 65536 back-end clock cycles between two hsyncs.

The bits vsync_pol and hsync_pol provide the polarities of the synchronization signals applied. If the synchronization signal is active HIGH, the corresponding flag is set to logic 1; otherwise the flag is set to logic 0.

The flag jitter_detected is set to logic 1, when the active edge of hsync and vsync coincide indicating a possible jitter of the syncs, which would lead to an incorrect or unstable result for the number of hsyncs between vsyncs; otherwise the flag is set to logic 0. If a possible jitter between hsync and vsync is detected, a delayed vsync can be used for the measurements instead, which is selected by setting delay_vsync to logic 1; otherwise the original vsync is used.

The value of v_lines reports the number of lines within a frame up to a maximum of 2048 lines and v_clocks gives the length of the input frame in back-end clock cycles with a maximum of $2^{24}$ clock cycles. The horizontal period in back-end clock cycles is given by h_clocks, which can be determined in different measurement modes. If h_clocks_accu and h_clocks_cont are both set to logic 0, the h_clocks value is determined once per frame in the middle of the frame. If h_clocks_accu is logic 1, then h_clocks gives the accumulated length of 16 lines also measured in the middle of the frame. If h_clocks_accu is logic 0, but h_clocks_cont is set to logic 1, then the h_clocks measurement is performed every line of the frame, including the vertical blanking and vsync time. The maximum horizontal period is 65536 back-end clock cycles.

The measurement results can be used to generate a mode detection interrupt. Each flag or value can be individually enabled for interrupt generation by setting the corresponding interrupt enable bit jitter_int_en, v_lines_int_en, h_clocks_int_en, v_clocks_int_en, no_vsync_int_en, no_hsync_int_en, vsync_pol_int_en or hsync_pol_int_en to logic 1. Changes of v_lines, v_clocks and h_clocks only cause an interrupt if the difference between new and old value is greater than four.

Additionally, the mode detection interrupt can be generated on the falling edge of every vsync, which is enabled if vsync_int_en is set to logic 1.

The states of each interrupt condition vsync_int, jitter_int, vsync_pol_int, hsync_pol_int, no_vsync_int, no_hsync_int, v_lines_int, h_clocks_int and v_clocks_int can be read out at registers MD_INT_HI and MD_INT_LO (0AH and 0BH). Whenever an interrupt condition is met, the particular flag is set to logic 1. If clear_int at MD_CTRL (00H) is programmed with logic 1, all interrupt flags are cleared.

If int_lock is set to logic 1, all flags and values are frozen in the moment an interrupt occurs until clear_int is set to logic 1 the next time.

#### 7.10.2 SYNC ACTIVITY DETECTION

Activity detection for AVI horizontal and vertical sync is provided. Moreover, the vertical sync output of the CSYNC slicer and the sync-on-green signal from the sync separator are checked permanently for activity. An interrupt may be generated on any change of activity. Interrupts can be masked with a set of interrupt enable bits. Writing a logic 1 to the existing clear_int bit will clear this interrupts.

For activity bits, logic 0 means inactive and logic 1 means active. For sync interrupt bits, logic 0 means disabled and logic 1 means enabled. For sync active interrupt bits, logic 0 means no interrupt and logic 1 means interrupt pending.

The sync-on-green activity detection is only an indicator that the digital output of the sync slicer is active. The line-locked PLL with its lock flag should be used to distinguish a real sync-on-green from disturbances resulting from the image data on the green channel.

**Table 30** Line PLL lock

| llpll_inlock | FUNCTION |
|---|---|
| 0 | line PLL out-of-lock |
| 1 | line PLL in lock |

#### 7.10.3 AUTO-ADJUSTMENT

There are four auto-adjustment modes:

- Active area detection
- Brightest and lowest pixel search
- Pixel measurement
- Phase distortion measurement.

The programming registers for all four modes are shared. Bit aa_mode selects the auto-adjustment mode according to Table 31.

**Table 31** Auto-adjustment modes

| aa_mode[1:0] | FUNCTION |
|---|---|
| 00 | active area detection |
| 01 | brightest and lowest pixel search |
| 10 | pixel measurement |
| 11 | phase distortion measurement |

In each mode, reference colours or reference coordinates have to be programmed (into bits ref_colour_0, ref_colour_1 or ref_row_0, ref_col_0, ref_row_1, ref_col_1 respectively). The auto-adjustment is activated by writing to the AA_CTRL register and started synchronized to the beginning of the next frame. The function is then applied for a number of frames defined in aa_cycles. After performing the auto-adjustment for this number of frames, an interrupt can be generated. The different aa-functions have two further aa_submode bits to control the functionality of each auto-adjustment mode.

##### *7.10.3.1 Active area detection*

With the active area detection feature it is possible to measure the number of blanking pixels and lines between the synchronization pulses and the active video.

To distinguish between blanking and active video the threshold colour values ref_colour_0 and ref_colour_1 have to be defined. Parameter ref_colour_0 is used to determine the start of the active video area. If the sample value of at least one of the three colour components is above this value the pixel is treated as upper left corner of active video.

Its coordinates are stored in eval_col_0 (x-coordinate) and eval_row_0 (y-coordinate). All pixels are also compared with the ref_colour_1 values. If one of the current colour values is bigger, the coordinates are saved in eval_row_1 and eval_col_1. At the end this value defines the lower, right corner of the active area. The values are kept in eval_row_0/1 and eval_col_0/1 until another mode or another area detection without resample is started. It is also possible to start measuring with the preceded values in the resample submodes. In submodes without resample, the results will not be smaller than the preceding values.

There are two different modes available:

- In the enhanced mode all input data is used for measurement (see Fig.15)
- In the non-enhanced mode only one input line, defined by ref_row_1 and one input column, defined by ref_col_1, is used (see Fig.16).

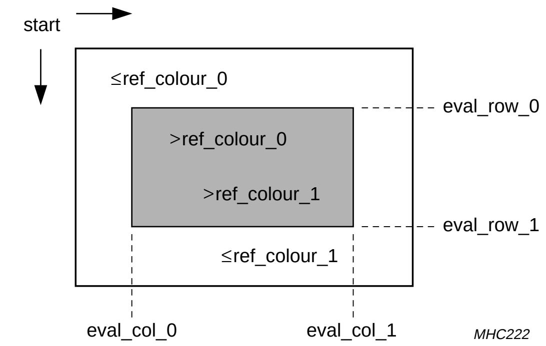
The needed sample offsets for the input interface can be directly obtained by reading out the eval_row_0 and eval_col_0 values. The number of active pixels per line and lines per field is generated by subtracting the eval_row_0, eval_col_0 value from the eval_row_1, eval_col_1 value.

Dependent on the sampling settings of the input interface, the eval_row_0 and eval_col_0 values usually correspond to the horizontal and vertical backporch of the incoming video signal and the active video, of course, meets the active area of that stream. The calculated active pixels per line value can be used for adjusting the line-locked PLL which is generating the ADC sample clock in a way that this value matches the number of expected active pixels per line in the actual graphics mode.

There are four submodes available, as shown in Table 32. In the non-enhanced modes, the active area detection is not performed over the whole frame as in the enhanced modes, but only within the line and within the column programmed by ref_row_0 and ref_col_0.

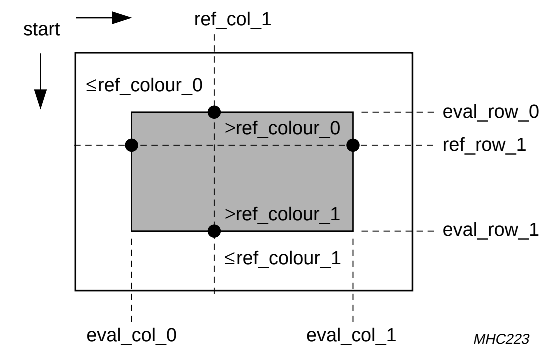
For correct results, these reference values have to be previously set inside the active area of the picture. If a submode with initial values is selected eval_col_0/1 and eval_row_0/1 are reset to their initial values before evaluation.

The brightest and lowest pixel value inside the active area is available in ref_pixel_red_0/1, ref_pixel_green_0/1 and ref_pixel_blue_0/1. The evaluated values for the lowest colour value pixel cannot be lower than the lowest threshold value.




Active area: shaded.

Fig.15 Enhanced mode.




Active area: shaded.

Fig.16 Non-enhanced mode.

**Table 32** Mode 00 (active area detection)

| BIT | DESCRIPTION |
|---|---|
| **Input values** | |
| aa_submode[1:0] | submode<br>00: enhanced mode with resampling, no initial values, evaluation over full frame<br>01: non-enhanced mode with resampling, no initial values, evaluation in one row and column<br>10: enhanced mode with initial values<br>11: non-enhanced mode with initial values |
| aa_cycles[1:0] | measurement interval<br>00: 1 frame<br>01: 4 frames<br>10: 8 frames<br>11: until next change in position value |
| ref_col_0[10:0] | reference column (for non-enhanced mode) |
| ref_row_0[11:0] | reference row (for non-enhanced mode) |
| ref_colour_0[23:0] | threshold value of backporch colour (upper left) |
| ref_colour_1[23:0] | threshold value of frontporch colour (lower right) |
| **Output values** | |
| eval_col_0[15:0] | active area upper left corner column; will be set to FFFF before evaluation in mode without resample |
| eval_row_0[15:0] | active area upper left corner row; will be set to FFFF before evaluation in mode without resample |
| eval_col_1[15:0] | active area lower right corner column; will be set to 0000 before evaluation in mode without resample |
| eval_row_1[15:0] | active area lower right corner row; will be set to 0000 before evaluation in mode without resample |
| ref_pixel_red_0[7:0] | maximum red component colour value over the whole frame |
| ref_pixel_green_0[7:0] | maximum green component colour value over the whole frame |
| ref_pixel_blue_0[7:0] | maximum blue component colour value over the whole frame |
| ref_pixel_red_1[7:0] | minimum red component colour value within active area |
| ref_pixel_green_1[7:0] | minimum green component colour value within active area |
| ref_pixel_blue_1[7:0] | minimum blue component colour value within active area |

#### 7.10.3.2 Brightest and lowest pixel search

The brightest and lowest pixel search determines the position of the brightest pixels and the lowest pixels in a predefined area. Therefore, the area is scanned from the upper left to the lower right corner. The pixel value and the position values are readable. Four submodes are available to search independently for RGB minimum and maximum values (see Table 33).

**Table 33** Mode 01 (minimum and maximum search)

| BIT | DESCRIPTION |
|---|---|
| **Input values** | |
| aa_submode[1:0] | submode<br>00: maximum of red and green<br>01: maximum of blue<br>10: minimum of red and green<br>11: minimum of blue |
| aa_cycles[1:0] | measurement interval<br>00: 1 frame<br>01: 4 frames<br>10: 8 frames<br>11: 16 frames |
| ref_col_0[10:0] | search area upper left corner column |
| ref_row_0[11:0] | search area upper left corner row |
| ref_col_1[10:0] | search area lower right corner column |
| ref_row_1[11:0] | search area lower right corner row |
| **Output values** | |
| eval_col_0[15:0] | pixel position 0 column (according to submode) |
| eval_row_0[15:0] | pixel position 0 row (according to submode) |
| eval_col_1[15:0] | pixel position 1 column (according to submode) |
| eval_row_1[15:0] | pixel position 1 row (according to submode) |
| ref_pixel_red_0[7:0] | red channel colour value at evaluated position 0 |
| ref_pixel_green_0[7:0] | green channel colour value at evaluated position 0 |
| ref_pixel_blue_0[7:0] | blue channel colour value at evaluated position 0 |
| ref_pixel_red_1[7:0] | red channel colour value at evaluated position 1 |
| ref_pixel_green_1[7:0] | green channel colour value at evaluated position 1 |
| ref_pixel_blue_1[7:0] | blue channel colour value at evaluated position 1 |

#### *7.10.3.3 Pixel measurement*

For exact measurements within the incoming video stream, two reference pixel positions can be defined with ref_row_0, ref_col_0 and ref_row_1, ref_col_1. The R, G and B components of this pixel are sampled and available at ref_pixel_red_0/1, ref_pixel_green_0/1 and ref_pixel_blue_0/1. The reference pixel colour values can be used for fine tuning the external PLL in frequency and phase and for colour gain adjustment.

Three submodes are available to output the maximum value, the minimum value or the mean value at the dedicated position (see Table 34).

To simplify the measurements, the values can be taken as a single snapshot representing the momentary value of the pixel at the reference position or they can be build up over several frames, which is activated by programming the number of frames to bits aa_cycles.

**Table 34** Mode 10 (pixel measurement)

| BIT | DESCRIPTION |
|---|---|
| **Input values** | |
| aa_submode[1:0] | submode<br>00: maximum of pixel at dedicated positions<br>01: minimum of pixel at dedicated positions<br>10: mean value of pixel at dedicated positions<br>11: mean value of pixel at dedicated positions |
| aa_cycles[1:0] | measurement interval<br>00: 1 frame<br>01: 4 frames<br>10: 8 frames<br>11: 16 frames |
| ref_col_0[10:0] | pixel position 0 column |
| ref_row_0[11:0] | pixel position 0 row |
| ref_col_1[10:0] | pixel position 1 column |
| ref_row_1[11:0] | pixel position 1 row |
| **Output values** | |
| ref_pixel_red_0[7:0] | red channel colour value at position 0 |
| ref_pixel_green_0[7:0] | green channel colour value at position 0 |
| ref_pixel_blue_0[7:0] | blue channel colour value at position 0 |
| ref_pixel_red_1[7:0] | red channel colour value at position 1 |
| ref_pixel_green_1[7:0] | green channel colour value at position 1 |
| ref_pixel_blue_1[7:0] | blue channel colour value at position 1 |

#### *7.10.3.4 Phase distortion measurement*

To help adjusting the phase for the ADCs, the SAA6713AH has a built-in phase distortion measurement which is calculating a 30-bit indicator value of a defined area of the video signal (see Table 35). The area for phase distortion measurements may contain active video or blanking. The area is defined by applying the upper left and lower right corner of the area to ref_col_0, ref_row_0 and ref_col_1, ref_row_1 respectively. Assuming a stable input picture with different pixel values inside the measurement window, the phase adjustment can be done by shifting the ADC phases and reading out the phase distortion indicator value of bits eval_row_0 and eval_col_0 for the maximum distortion value and eval_row_1 and eval_col_1 for the minimum distortion value. The best sampling phase corresponds to the highest value of the phase distortion indicator.

This phase distortion value could also be used for the frequency adjustment by sweeping the input frequency around the assumed target frequency.

When auto-adjustment is in phase distortion mode, a signature is calculated for each frame. Changes inside the input stream can be detected by reading the signature bits ref_pixel_blue_0, ref_pixel_red_1, ref_pixel_green_1 and ref_pixel_blue_1. For still pictures, the signature does not change.

**Table 35** Mode 11 (phase distortion measurement)

| BIT | DESCRIPTION |
|---|---|
| **Input values** | |
| aa_submode[1:0] | not used |
| aa_cycles[1:0] | measurement interval<br>00: 1 frame<br>01: 4 frames<br>10: 8 frames<br>11: 16 frames |
| ref_col_0[10:0] | measurement area upper left corner column |
| ref_row_0[11:0] | measurement area upper left corner row |
| ref_col_1[10:0] | measurement area lower right corner column |
| ref_row_1[11:0] | measurement area lower right corner row |
| **Output values** | |
| eval_col_0[15:0] | maximum of distortion[15:0] |
| eval_row_0[15:0] | maximum of distortion[29:16] |
| eval_col_1[15:0] | minimum of distortion[15:0] |
| eval_row_1[15:0] | minimum of distortion[29:16] |
| ref_pixel_blue_0[5:0] | signature[29:24] |
| ref_pixel_red_1[7:0] | signature[23:16] |
| ref_pixel_green_1[7:0] | signature[15:8] |
| ref_pixel_blue_1[7:0] | signature[7:0] |

#### 7.10.3.5 *How to use auto-adjustment*

**Table 36** Auto-adjustment steps

| STEP | ACTION |
|---|---|
| 1 | program position values according to mode |
| 2 | program mode, submode and cycle values to start auto-adjustment |
| 3 | wait until interrupt appears |
| 4 | read the according values |

### 7.11 Decoupling FIFO

The decoupling FIFO allows an output line generation independent of the input line timing. The FIFO holds 1280 pixels, and either buffers incoming data when the vertical upscaling does not require any or holds back a line to be able to provide a continuous data stream in case of vertical downscaling.

The FIFO output is locked after every line if line_lock is set to logic 1; otherwise after every frame and only released if the FIFO level exceeds the threshold level, given by fifo_threshold in units of 8 pixels.

### 7.12 Scaling

The SAA6713AH features separate scaling engines for upscaling and downscaling, for both horizontal and vertical processing. Two separate scaling units are implemented to perform upscaling and downscaling.

#### 7.12.1 DOWNSCALING

The downscaling engine is used for reducing the incoming RGB data stream, i.e. for displaying high resolution input frames on panels with a smaller resolution. The scaling ratio can be programmed independently for both horizontal and vertical downscaling units. The algorithm uses pixel accumulation, achieving a minimum scaling factor of $\frac{1}{64}$. If the downscaler is used, it must be enabled by setting dsc_en to logic 1.

Setting-up the desired downscaling ratios is achieved by programming the scaling increments dsc_v_incr, dsc_v_incr_corr, dsc_h_incr and dsc_h_incr_corr. This must be done for both vertical and horizontal scaling.

$$\text{incr} = \frac{\text{number\_of\_output\_pixels}}{\text{number\_of\_input\_pixels}} \times 64 = \text{xx.yy}$$

Where xx is equivalent to dsc_v_incr or dsc_h_incr and yy is the fraction of the result in $\frac{1}{100}$.

This is the value for programming the increment correction values dsc_v_incr_corr and dsc_h_incr_corr.

Example: from SXGA to XGA.

Horizontal: $\frac{1024}{1280} \times 64 = 51.20$

This means dsc_h_incr = 51 and dsc_h_incr_corr = 20.

Vertical: $\frac{768}{1024} \times 64 = 48.00$

This means dsc_v_incr = 48 and dsc_v_incr_corr = 00.

#### 7.12.2 UPSCALING

The upscaling engine is used for enlarging the incoming video frames. The magnification can be programmed individually for horizontal and vertical scaling. The maximum scaling factor for both directions is 64.

The implemented filter algorithm uses interpolation with pixel enhancement, based on a free programmable transition function. Therefore, it is possible to define the transition between two calculated pixels to obtain different sharpness characteristics. This transition function must be defined in the 48 × 8 bits look-up table, with a number ranging from 0 to 64. Different functions can be programmed for horizontal and vertical scaling.

The upscaler must be activated by usc_en. To set up the zoom factor, the scaling increments v_scale_incr, v_scale_corr, h_scale_incr and h_scale_corr must be programmed.

$$\text{incr} = \frac{\text{number\_of\_output\_pixels}}{\text{number\_of\_input\_pixels}} \times 64 = \text{xx.yy}$$

Where xx is equivalent to v_scale_incr or h_scale_incr and yy is the fraction of the result in $\frac{1}{100}$.

This is the value for programming the increment correction values v_scale_corr and h_scale_corr.

Example: from XGA to SXGA.

Horizontal: $\frac{1280}{1024} \times 64 = 80.00$

This means h_scale_incr = 80 and h_scale_corr = 00.

Vertical: $\frac{1024}{768} \times 64 = 85.33$

This means v_scale_incr = 85 and v_scale_corr = 33.

**Remark:** The last digit must be rounded up: 85.33 results in 1023.96 lines, but the upscaler will display only 1023 lines.

#### 7.12.3 HORIZONTAL FLIPPING

The SAA6713AH provides the possibility to flip horizontally the incoming picture. As flipping needs a line memory, both the downscaler and the upscaler have a flip programming register. When using the downscaler flip mode (flip_h = 1), no vertical downscaling can be performed. This is to be used when upscaling and flipping have to be programmed.

In case downscaling and flipping shall be performed, flipping has to be done inside the upscaler by setting usc_h_flip to logic 1.

### 7.13 On screen display

The on screen display consists of three different and independent parts: OSD text, OSD bitmap and OSD pointer, where the OSD text is used as the ‘main’ OSD part to build an application specific On Screen Menu (OSM). The bitmap part of the OSD is intended to be used for company logo or can be used as the backdrop of an OSM with up to 16 individual colours. As an addition for the graphical user interface in the OSM, the OSD pointer part allows a hardware cursor that is overlaid over picture data and the other OSD data. Its intention is to be used as a mouse pointer for selecting and modifying OSM items.

Each of the three OSDs can be zoomed independently with pixel repetition by the factors 1, 2, 3 and 4 and can be rotated by 90 degrees clockwise, horizontally and vertically mirrored, if desired. All colour information used by the three OSD parts are organized in global colour tables (palettes) which define a certain colour each with 24-bit RGB data. These colour and palette registers are located at register page 9 (OSD colours).

#### 7.13.1 OSD TEXT

The OSD text is a character based approach and consists of a window definition RAM, a font definition RAM and a font definition ROM. The window definition RAM gives the information about the data that is going to be displayed. It is organized as a character-based matrix that is free definable in terms of width and height (registers OSDT_WX and OSDT_WY) as long as the resulting number of elements does not exceed the maximum number of 1024 elements. Each element of this window matrix can directly accessed using the cursor registers OSDT_CURX and OSDT_CURY. The display position where the OSD text window is displayed in the picture, can be freely defined via the registers OSDT_PX and OSDT_PY.

In Fig.17 an example of an 11 × 5 character window is shown that uses a total number of 55 elements. It should be noted that the parameters OSDT_WX and OSDT_WY are given in CHARACTER units, whether the offset of the window is given in PIXEL units. The real size of the OSD text window depends on the actual defined font resolution (OSDT_FR_X and OSDT_FR_Y), the actual zoom factor (zoom[1:0] value of 1, 2, 3 or 4 in register OSDT_CTRL0) and the rotate flag (rotate_right in register OSDT_CTRL0). So, the overall size of the OSD text in pixel is derived by calculating OSDT_WX × ZOOM × OSDT_FR_X respectively OSDT_WY × ZOOM × OSDT_FR_Y.
In addition to this nominal window size, the optional window shadow feature (bit window_shadow) will extend the active OSD text area by the defined width and height (OSDT_WSHAD) multiplied with the actual zoom factor.

Keep in mind that during rotation of the OSD, the core OSD text height and width will be visible exchanged, but the anchor position and the window shadow will not be seen (see Fig.18). From the application (software) point of view, the OSD programming does not change no matter if horizontal, vertical or flip flags are used or not. Only the display position registers (anchor) OSDT_PX and OSDT_PY must be chosen in a way that the now transposed OSD text window fits still in the picture. All matrix and font based accesses are automatically transposed, not even the index of the elements (cursor) has to be considered.

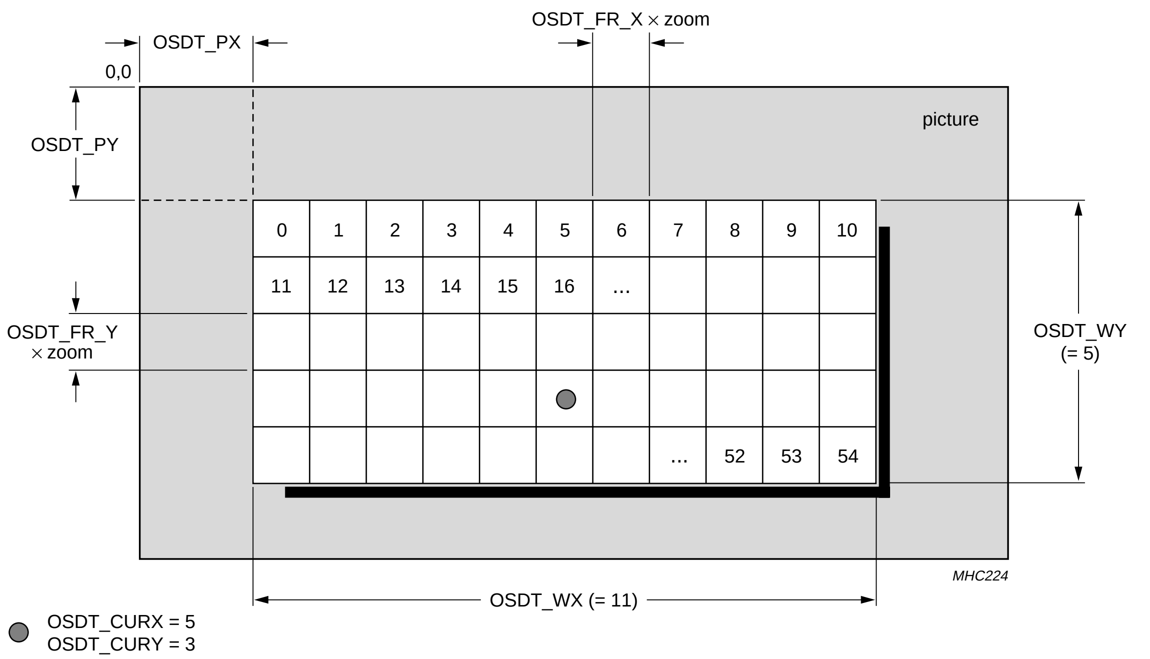



Fig.17  OSD window definition.

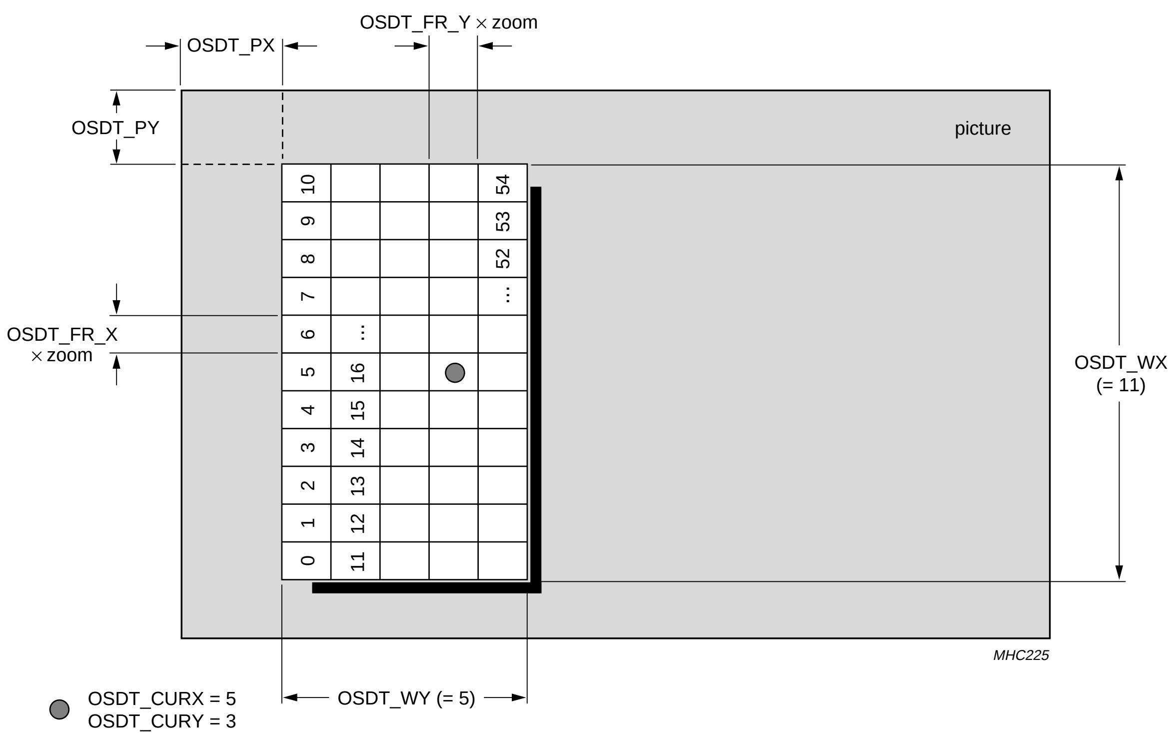



Fig.18  OSD window horizontal, vertical, flip, rotate.

To allow easy access to the window definition when writing data to the OSD text, the cursor will perform an auto-increment function to the next right element (or to a new line, if the line ends) each time an element is written to the RAM which is allowing an $I^2C$-bus burst transmission to define the window contents. The actual cursor values can be read back at any time.

Each element of the OSD text window consist of 23 bits. They represent the property of an OSD text character. The elements are accessible via the OSDT_PROP2, OSDT_PROP1 and OSDT_PROP0 registers (see Table 37). All information encoded with these OSDT_PROP registers is valid for one character only, so the look of an OSD text can be changed mainly on a character base. The colour definition elements bg_colour[2:0] and fg_colour[2:0]/palette[2:0] are not defining a colour directly but are assigning a value from the global OSD colour tables which are defined within register page 9. For single colour characters, the user can select one of the eight possible foreground colours and one of eight possible background colours. For multicolour characters one out of eight possible colour palettes is chosen; each defining four colours (1 background and 3 foreground) to be used within this character. The information whether a character is a multicolour character or a single colour character is derived from the charcode and the value of sc_startcode (OSDT_SC_HI and OSDT_SC_LO) that defines the multi or single character mapping inside the font RAM.

**Table 37** OSD property registers

| REGISTER | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| OSDT_PROP2 | | blink[1:0] | | shadow | bg_trans | fg_trans | bg_alpha | fg_alpha |
| OSDT_PROP1 | bg_colour[2:0] | | | fg_colour[2:0]/palette[2:0] | | | ROM | charcode[8] |
| OSDT_PROP0 | charcode[7:0] | | | | | | | |

**Table 38** OSD property registers bit description

| BIT | DESCRIPTION |
|---|---|
| blink[1:0] | defines the blink mode of the character<br>00: blinking is off<br>01: blinking of foreground only<br>10: character is inverse, no blinking<br>11: blinking by inversion of foreground and background colour |
| shadow | if set to logic 1, the character will be displayed with an 1 pixel horizontal and vertical shadow |
| bg_trans | if set to logic 1, the background colour is displayed transparent |
| fg_trans | if set to logic 1, the foreground colour is displayed transparent |
| bg_alpha | if set to logic 1, the background will not be displayed solid but alpha-blended with picture data using the global definable background alpha-blending factor (OSDT_BGA) |
| fg_alpha | if set to logic 1, the foreground will not be displayed solid but alpha-blended with picture data using the global definable foreground alpha-blending factor (OSDT_FGA) |
| bg_colour[2:0] | defines which of the 8 definable text background colours is used for this character |
| fg_colour[2:0]/ palette[2:0] | shared programming bits; fg_colour defines which of the 8 text foreground colours is used for this character (only valid if charcode points to a single colour character) and palette defines used multicolour palette (only valid if charcode points to a multicolour character) |
| ROM | if set to logic 1, the character is selected from ROM; if set to logic 0, then the character is selected from RAM |
| charcode[8:0] | indicates the desired character inside the font ROM/GEN or the font RAM |

Different property register write modes can be selected, allowing to accelerate the I$^2$C-bus programming of OSD windows with characters sharing the contents of one or more property registers. Parameter write_mode of register OSDT_MASK (see Table 39) controls which of the three property registers OSDT_PROP2 to OSDT_PROP0 have to be updated until the character information is internally written into the window RAM. The registers are activated by write_mode according to Table 41. Only once all activated registers have been updated via the I$^2$C-bus, the character is written into the window RAM and the cursor position defined by OSDT_CURX and OSDT_CURY is advanced to the next window element. The information of an inactive property register is still included in the character definition, but the register does not have to be rewritten for every new character definition.

Example 1: for a single-coloured ASCII character-based OSD, write_mode is set to ‘001’ and property registers OSDT_PROP2 and OSDT_PROP1 have only to be defined initially, all window elements are then defined by consecutive writing of OSDT_PROP0. With every write operation to OSDT_PROP0 a new window element is defined.

The I$^2$C-bus burst access is also supported for the property registers specified by parameter write_mode as specified in Table 41. The active property register values of consecutive window elements can be transmitted in the I$^2$C-bus burst mode without the requirement of repeating device addressing and the transmission of the subaddress for every character or property register.

Example 2: for a multi-coloured OSD, write_mode is set to 6 to activate only OSDT_PROP1 and OSDT_PROP0. OSDT_PROP2 is set initially. The OSD can then be programmed with one I$^2$C-bus write burst consisting of device addressing byte, the OSDT_PROP1 subaddress followed by OSDT_PROP1 value of first character, OSDT_PROP0 value of first character, OSDT_PROP1 value of second character, OSDT_PROP0 value of second character, OSDT_PROP1 value of third character etc. After each transmission of an OSDT_PROP0 value the character definition is transferred into the window RAM. The masking bits (see Table 40) are used as a data filter that specifies which parts of the complete OSDT_PROP word (23 bits) are written to the RAM and which are masked out. Each attribute will only be updated in the OSD text window RAM element if its mask bit is set to logic 1. If that is not the case, the window RAM will ignore this part of the OSDT_PROP register and will keep up its previously defined value for this part at the selected OSD text window element.

**Table 39** OSDT_MASK register

| REGISTER | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| OSDT_MASK | blink_mask | shadow_mask | bg_mask | fg_mask | code_mask | write_mode[2:0] | | |

**Table 40** OSDT_MASK register bit description

| BIT | DESCRIPTION |
|---|---|
| blink_mask | 1: blink[1:0] property will be written according to actual OSDT_PROP2 settings<br>0: blink[1:0] property will not be modified |
| shadow_mask | 1: shadow property will be written according to actual OSDT_PROP2 settings<br>0: shadow property will not be modified |
| bg_mask | 1: all background information will be written according to actual property settings (OSDT_PROP2: bg_trans, bg_alpha and OSDT_PROP1: bg_colour)<br>0: the background property will not be modified |
| fg_mask | 1: all foreground information will be written according to actual property settings (OSDT_PROP2: fg_trans, fg_alpha and OSDT_PROP1: fg_colour)<br>0: the foreground property will not be modified |
| code_mask | 1: the charcode property will be written according to actual property settings (OSDT_PROP1: ROM, charcode[8] and OSDT_PROP0: charcode[7:0])<br>0: the charcode property will not be modified |
| write_mode[2:0] | write mode selection (see Table 41) |

**Table 41** Write mode selection

| write_ mode[2] | write_ mode[1] | write_ mode[0] | ACTIVE $I^2$C-BUS REGISTERS | $I^2$C-BUS SUBADDRESS AUTO-INCREMENT HANDLING |
|---|---|---|---|---|
| 0 | 0 | 0 | – | – |
| 0 | 0 | 1 | OSDT_PROP0 | burst access to OSDT_PROP0 |
| 0 | 1 | 0 | OSDT_PROP1 | burst access to OSDT_PROP1 |
| 0 | 1 | 1 | OSDT_PROP1 and OSDT_PROP0 | sequential access to OSDT_PROP1 → OSDT_PROP0 → OSDT_PROP1 → etc. |
| 1 | 0 | 0 | OSDT_PROP2 | burst access to OSDT_PROP2 |
| 1 | 0 | 1 | OSDT_PROP2 and OSDT_PROP0 | sequential access to OSDT_PROP2 → OSDT_PROP0 → OSDT_PROP2 → etc. |
| 1 | 1 | 0 | OSDT_PROP2 and OSDT_PROP1 | sequential access to OSDT_PROP2 → OSDT_PROP1 → OSDT_PROP2 → etc. |
| 1 | 1 | 1 | OSDT_PROP2 to OSDT_PROP0 | sequential access to OSDT_PROP2 → OSDT_PROP1 → OSDT_PROP0 → OSDT_PROP2 → etc. |

The combination of the mask bits and the write_mode already provides a powerful way to speed any OSM drawings by minimizing the needed $I^2$C-bus transmissions, but there is even more hardware support for defining an area inside the OSD text window which has the same element property for all elements within its boundaries.

An area can be defined using the upper-left and bottom-right cursor coordinates inside the OSD text window matrix using the OSDT_FAULX, OSDT_FAULY, OSDT_FABRX and OSDT_FABRY registers. The execution of the writing is initiated by writing a logic 1 to areafill_start (register OSDT_CTRL0) and as before, the current value of the complete 23-bit OSDT_PROP word is written to each element of the defined area. Of course, the mask bits are still valid and can be used also during an areafill execution. So, this function can not only be used to overwrite and clear areas inside the OSM, it can also be used to highlight or blink certain areas in the OSM. It should be noted that it might be needed to set the write_mode to ‘000’ if you want to change any of the OSDT_PROP settings previous to an areafill and assure that no write and cursor auto-increment is done accidentally.

As described before: all definitions of the OSD window elements are just defining the property of a character and are pointing to a font definition by the charcode attribute. The real character contents are taken from either the font ROM/GEN part or the font RAM part of the OSD text indexed by that charcode.

The font definition ROM/GEN is already providing a large amount of predefined fonts as illustrated in Fig.19.

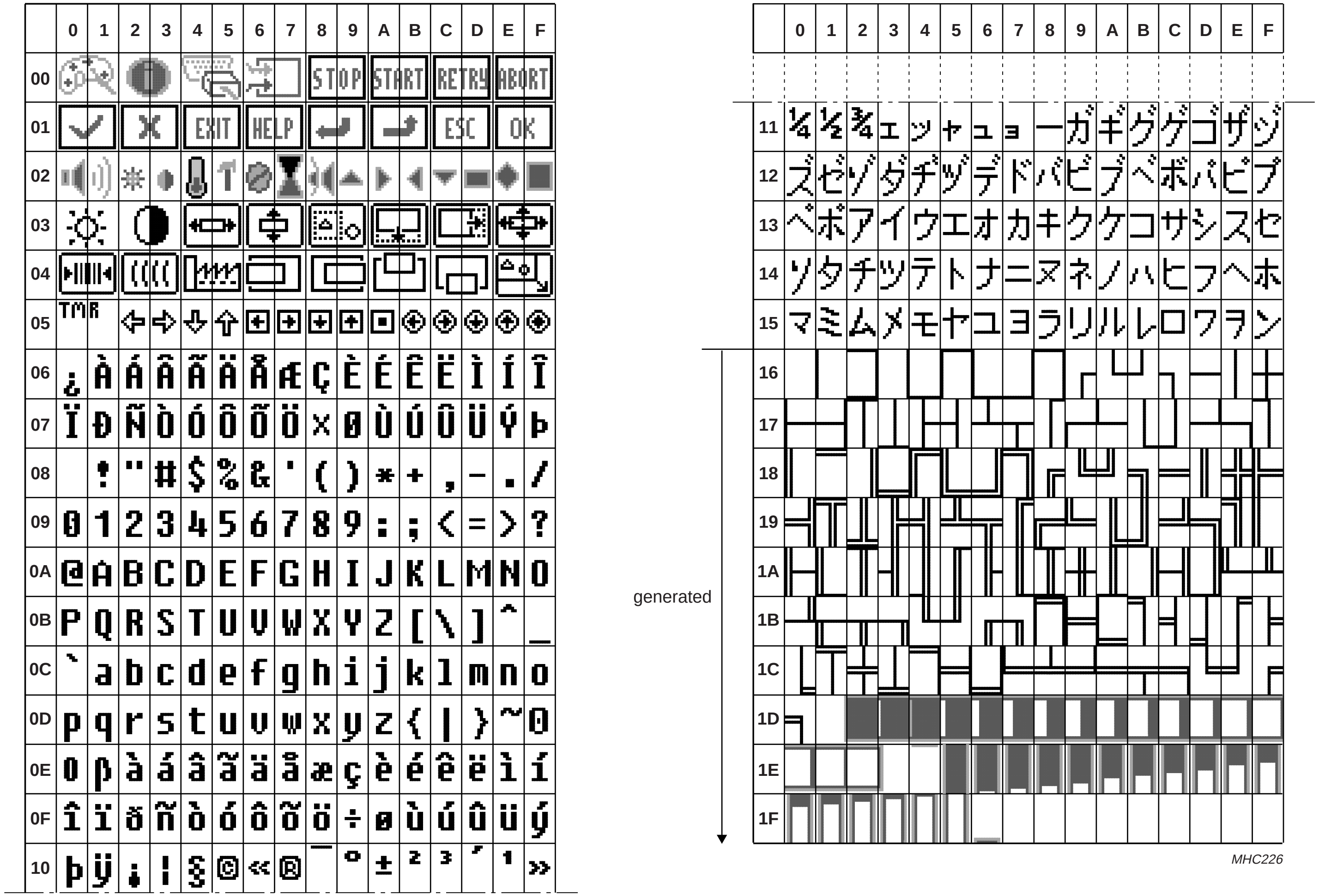



Fig.19 ROM/GEN character codes.

The character codes from 000H to 15FH (see Table 42) are ROM defined characters with the natural resolution of $12 \times 18$ pixels, so the font resolution defined via OSDT_FR_X and OSDT_FR_Y should be set to 12 and 18 to achieve optimal viewing results. If the actual font resolution is defined greater than $12 \times 18$, all ROM characters will be centred automatically inside the programmed font size; if it is less than $12 \times 18$ the characters will be cropped. This handling allows ROM and RAM characters to be displayed together in one OSD, even if the RAM font size does not fit the ROM size of $12 \times 18$ pixels. Looking at Fig.19 it is easily seen that the ROM is using six different subareas for the charcode.

The addresses from 160H to 1F6H are mapped to the internal character generators (GEN). Despite the real ROM definitions these characters do not have a native resolution instead they will always be displayed in the actual defined font resolution (OSDT_FR_X and OSDT_FR_Y) itself.

While the border characters of the font GEN are kept fixed and just adapted to the used font size, the slider parts are generated based on its parameters OSDT_SLP1 and OSDT_SLP0 (see Table 43).

**Table 42** ROM mapping

| ADDRESS (HEX) | CONTENTS |
|---|---|
| 000 to 01F | multicolour, dual character symbols |
| 020 to 02F | multicolour, single character symbols |
| 030 to 04F | single colour, dual character symbols |
| 050 to 05F | single colour, single character symbols |
| 060 to 112 | single colour, ANSI like character set with ASCII mapping (ASCII code + 60H) |
| 113 to 15F | single colour, basic Japanese font set |
| 160 to 1D1 | single colour, border and line characters |
| 1D1 to 1F6 | multicolour generated slider parts |

**Table 43** Slider property registers

| REGISTER | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| OSDT_SLP1 | slider_border[3:0] | | | | slider_offset[3:0] | | | |
| OSDT_SLP0 | | | | slider_style | slider_gap[3:0] | | | |

**Table 44** Slider property registers bit description

| BIT | DESCRIPTION |
|---|---|
| slider_offset[3:0] | distance from the character border to the generated slider in pixel |
| slider_border[3:0] | thickness of slider border in pixel |
| slider_gap[3:0] | gap between slider border and slider core in pixel |
| slider_style | 0: fill; middle slider parts are solidly filled from left to right and from bottom to top<br>1: peak; middle slider parts are created with a single marker at reference position |

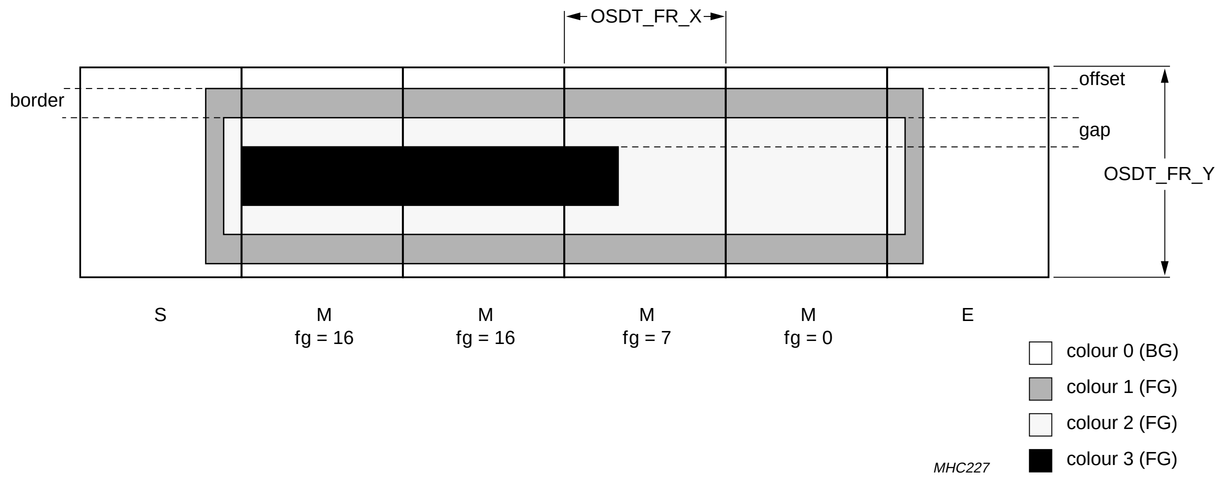



Fig.20 Multicolour slider generation.

When putting together a slider with multiple characters (see Fig.20), a slider always consists of three basic parts: a start part (S), a middle part (M) with different fill grades (fg) and an ending part (E).

The start part is always located at address 1D2H (horizontal) respectively 1F6H (vertical) and the end part is always located at 1E3H respectively 1E4H.

The characters in between the start and the end character correspond to different fill grades of the middle part that are needed to give a subcharacter precision in a custom slider. In this system the middle part uses a natural resolution of 16 pixel per character which is resulting in 17 different characters from empty to full. If the display size is different from the natural resolution, either some fill grades will be skipped or some fill grades will be doubled within the 17 characters that are reserved to represent the middle parts. Anyhow mathematically, it is always correct to use the slider resolution of 16 to calculate the fill grade of the last partly filled slider part and use this value directly to index to the correct middle part. Using this approach results in an overall slider resolution equal to the number of used middle characters multiplied by 16 to graphically display any values within an OSM either with a fill bar or a single marker.

To achieve more flexibility in the OSD look, 4 kbyte of user definable RAM can be used in addition to the ROM/GEN characters. This font definition RAM can contain a downloadable mixed multicolour or single colour font which is in terms of character size freely programmable via OSDT_FR_X and OSDT_FR_Y registers but has to be between $8 \times 8$ and $32 \times 32$ pixels. This font resolution is valid for all characters inside the RAM, no matter if they are defined in single colour or multicolour but a single colour pixel can be stored with one while a multicolour pixel occupies two bits inside the font RAM. For single colour characters a pixel of value ‘0’ will be displayed as background and a pixel data of value ‘1’ will be displayed as foreground according to the defined colour values in the OSDT_PROP registers and the OSD colour definitions on register page 9. For multicolour characters always two bits are taken for each pixel that directly map to the four colour value inside the selected multicolour palette which is also defined in register page 9 and is selected within OSDT_PROP.

The font RAM (see Table 45) can store a maximum number of 512 $8 \times 8$ single colour characters which corresponds to the 9-bit charcode in the OSDT_PROP registers. Using multicolour definitions that need two bits per pixel and/or larger font resolution reduces of course the number of possible characters to be stored in the font RAM.

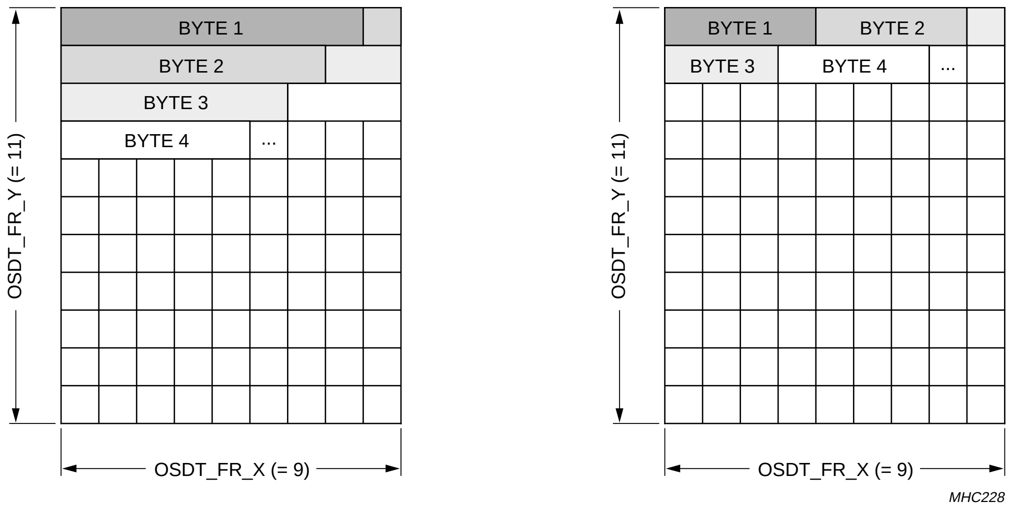
The multicolour font definitions have to start always from address 0 and the start of the single colour definitions is indicated by the configurable single colour start code (OSDT_SC register). This means all the characters with charcodes below the OSDT_SC value are treated as multicolour and all charcodes above this value are handled as single colour characters. To use only multicolour characters in the RAM font set this value must be set to an unreachable value. Due to the RAM size of 4 kbyte a multicolour 8 × 8 font with 2 bits per pixel can hold 256 characters as maximum so it is enough to set only bit 8 in OSDT_SC_HI register to logic 1. To use only single colour definitions set the complete OSDT_SC pointer simply to 000H.

A character is stored in the font RAM using the OSDT_CC_HI, OSDT_CC_LO, OSDT_CMASK and OSDT_CDEF registers where OSDT_CC gives the charcode of the character to be defined and OSDT_CDEF and OSDT_CMASK are used for the data bits and the according mask using 8 bits at a time.

The data format written to OSDT_CDEF has to be MSB aligned representing the following pixel of the character with the pixel sequence processed from top to bottom and left to right. As mentioned before, a single colour pixel is represented with one bit while a multicolour pixel needs two bits to be described. So each definition of a character will need multiple writes to OSDT_CDEF (8 bits at a time) until the whole character is completed. The total number of bytes to be transmitted is depending on the defined font size and if the character uses a single or multicolour definition.

**Table 45** Examples of possible font RAM configurations

| FONT SIZE | STORABLE CHARACTERS | |
|---|---|---|
| | COLOUR | MAXIMUM NUMBER |
| 8 × 8 | single colour only | 512 |
| | multicolour only | 256 |
| | mixed example | 256 single colour and 128 multicolour |
| 12 × 16 | single colour only | 170 |
| | multicolour only | 85 |
| | mixed example | 100 single colour and 35 multicolour |
| 12 × 18 | single colour only | 151 |
| | multicolour only | 75 |
| | mixed example | 121 single colour and 15 multicolour |
| 9 × 13 | single colour only | 280 |
| | multicolour only | 140 |
| | mixed example | 200 single colour and 40 multicolour |
| 24 × 24 | single colour only | 56 |
| | multicolour only | 28 |
| | mixed example | 40 single colour and 8 multicolour |
| 32 × 32 | single colour only | 32 |
| | multicolour only | 16 |
| | mixed example | 24 single colour and 4 multicolour |




a. Single colour definition sequence (needs $9 \times 11$ = 99 bits $\rightarrow$ 13 bytes).

b. Multicolour definition sequence (needs $9 \times 11 \times 2$ = 198 bits $\rightarrow$ 25 bytes).

Fig.21 Data format of the OSD font RAM.

It should be noted that the characters are not stored byte-aligned in the internal RAM due to the programmable font sizes (see Fig.21). Due to this, a bit-exact address is internally needed that points to the first bit of a character inside an 8-bit data word. This internal character base address is calculated new, each time data is written to either the OSDT_CC_HI or OSDT_CC_LO register and is incremented by 8 bits on each write to OSDT_CDEF. During each write the actual value of the OSDT_CMASK register is used. So for an $8 \times 8$ single colour character to be defined, simply set OSDT_CC to the desired charcode, set all bits of OSDT_MASK to logic 1 and write 8 times to OSDT_CDEF to define the complete character. For a $9 \times 9$ single colour character (needs 81 bits) you have to do 11 writes to OSDT_CDEF, even if only the MSB of the 11th transmission is used and the remaining 7 bits stay unused. One could now either use masked writing to mask those bits 6 to 0 out (so the following character definition is not touched) or immediately continue with the definition of the next character and simply connect the next character data to bit 6 down to bit 0. Because the IC cannot determine which functionality is desired, the user can select this by setting the OSDT_CC register. If an automatic masking at the end of each character is desired, the flag single_char_def (OSDT_CC_HI[7]) can be set to logic 1. This means that before the next character definition the desired charcode must be written again, because the internal bit address does not longer match with the following character base address. Otherwise, if this bit is kept to logic 0, the internal bit address is continued over character boundaries allowing multiple bit-packed character transmission in a sequence. Only the last definition might need a manual masked writing in case of an address space overflow or if any needed data is present at higher charcodes.

To reduce the programming time for the font RAM, an auto-increment function is used internally so that an I$^2$C-bus burst transmission can be used to transmit as many 8-bit data words as needed, even configuring all characters in one continuous burst. To allow this, the SAA6713AH register auto-increment is re-addressing OSDT_CDEF after each write to OSDT_CDEF.

Any changes to the OSD text RAM definitions can also be made while the OSD is displayed. So the usable character set is only limited by the size of the external microcontrollers ROM. Just keep in mind that due to internal address calculation, the font size (OSDT_FR_X and OSDT_FR_Y) and the single colour start code (OSDT_SC_LO and OSDT_SC_HI) must be defined prior to any font definition, in order that the character data will not look disturbed.

### 7.13.2 OSD BITMAP

The OSD bitmap part can be used for displaying pixel based multicolour graphics along with the regular text based OSM (see Fig.22). Its display position can be defined anywhere in the picture (OSDB_PX and OSDB_PY) and like the OSD text it can be zoomed, flipped and rotated according to the settings within its control register OSDB_CTRL0. It is allowed to overlap with a displayed OSD text window if desired. In this event the bitmap_behind flag (OSDB_CTRL0[6]) defines whether the bitmap part appears on top or behind the OSD text information. Two separate alpha-blending values (bg_alpha and fg_alpha) define a blending value for the OSD bitmap and separate transparent flags provide the possibility to define either the foreground or the background from transparent to solid. The display colours are again defined on a separate osd_bitmap palette inside the osd_colour definition page 9 so each pixel can use one of the defined 16 colours where bitmap colour 0 is always treated as background. The OSD bitmap uses an internal memory of 4 kbyte RAM in which the graphic pixels are stored (see Table 46). It can be parametrized freely in width (OSDB_SX), height (OSDB_SY) and colour depth (OSDB_CTRL1[6:5], bits per pixel) with the restriction that the needed memory size still fits into the 4 kbyte memory address space.

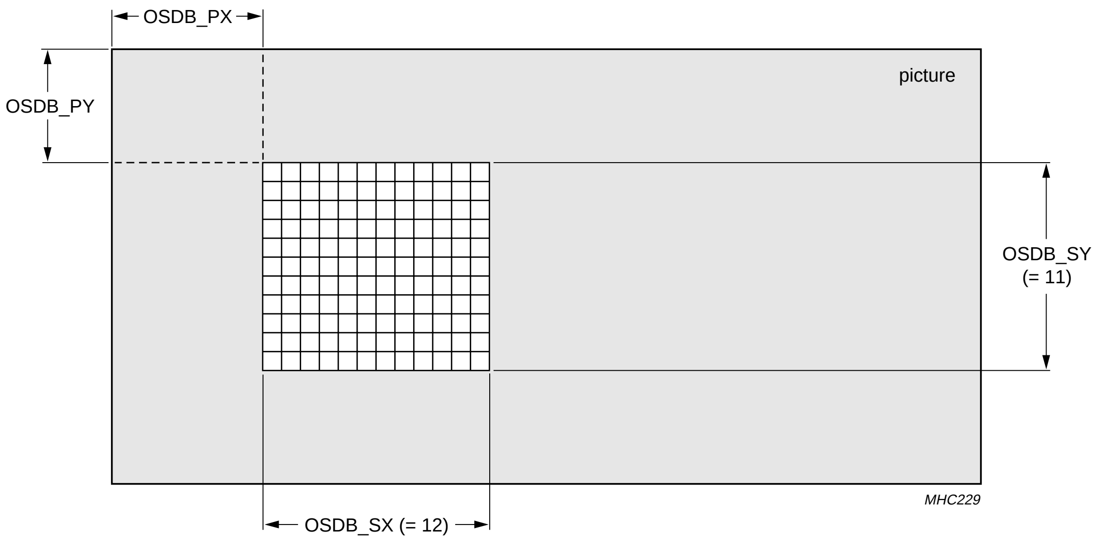



Fig.22 OSD bitmap structure.

**Table 46** Bitmap RAM configurations; note 1

| BPP CODE | USED BITS PER PIXEL | USED COLOURS | DISPLAYABLE PIXELS | EXAMPLE WINDOW SIZES (NOT ZOOMED) |
|---|---|---|---|---|
| 00 | 1 | 2 | 32768 | 256 × 128; 181 × 181 |
| 01 | 2 | 4 | 16384 | 256 × 64; 128 × 128 |
| 1X | 4 | 16 | 8192 | 256 × 32; 90 × 90 |

**Note**

1. X = don’t care.

The access to the graphic memory is based on a masked writing with pixel exact addressing (see Fig.23) via the write cursor (OSDB_CX, OSDB_CY) always configuring 8 bits at a time (OSDB_DEF) with data being processed from left to right and top to bottom. Using the 8 corresponding mask bits (OSDB_MASK) any pixel within the OSD bitmap can directly be accessed and redefined without changing neighbouring pixel also during display time allowing software guided animations for fancy start-up screens. Both the OSDB_DEF and OSDB_MASK are always MSB aligned which means that bit 7 will be written to the pixel address that is referenced by the OSD bitmap cursor (OSDB_CX, OSDB_CY). Depending on the selected bitmap size and the number of colours to be used this pixel address will probable not be byte aligned but the user does not have to take care of any internal alignments. The next 8 bits that are written to OSDB_DEF will be written bit wise starting with the MSB from the given cursor location.

In order to speed up the OSD bitmap definitions the internal RAM address is incremented by 8 bits always when a write to OSDB_DEF happened. Together with a stop of the SAA6713AH register auto-increment at this register, this allows a fast burst configuration of multiple pixel up to a complete OSD bitmap definition setting the cursor to (0,0), the mask to FFH and writing all needed data bytes in a single burst. The number of needed byte transmissions is derived by multiplying the total number of pixels to be configured with the used bits per pixel and dividing this result by 8 bits. When overwriting parts of the bitmap image the user must handle the OSDB_MASK flags for the remaining bits that shall not overwrite any data by himself.

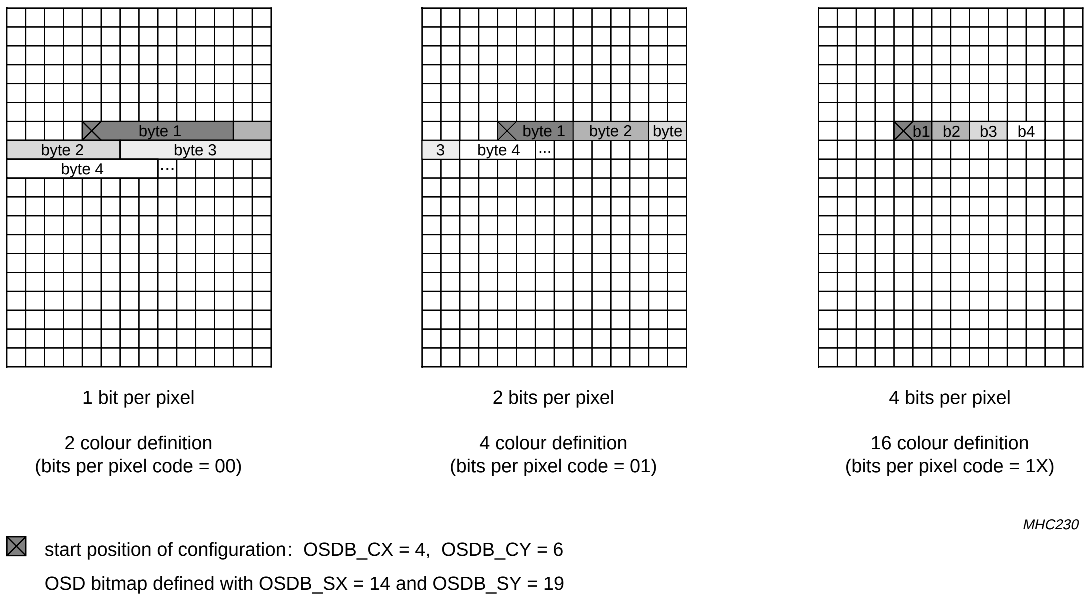



Fig.23 Data format of the OSD bitmap RAM.

#### 7.13.3 OSD POINTER

The OSD pointer icon is a four colour 32 × 32 pixel structure and is intended to be used as a cursor on top of an OSM (see Fig.24). It is very much alike the OSD bitmap part allowing individual positioning (OSDP_PX_HI, OSDP_PX_LO, OSDP_PY_HI and OSDP_PY_LO), zooming, flipping and rotating (OSDP_CTRL0) but will always be displayed on top of any OSD text or OSD bitmap window. It is fixed in its resolution and always uses 2 bits per pixel allowing 4 possible colours. Those colours are again definable via a palette in the OSD colour settings (register page 9) where colour 0 is always treated as background colour. The foreground and the background colours can be displayed from solid to almost transparent with an individual alpha-blending factor (OSDP_FGA and OSDP_BGA) or fully transparent using fg_trans and bg_trans flags inside the OSDP_CTRL1 register.

For animation purpose of the icon, it is double buffered and able to generate a frame based switching interrupt. The buffer to be displayed can either be selected manually via the buffer_sel flag (OSDP_CTRL0[6]) or can be switched automatically on each generated interrupt (OSDP_CTRL0[7]). During the display of one buffer all writes are redirected to the inactive buffer. The period of the animation interrupt can be adjusted with the OSDP_AD register that defines the number of frames between two interrupts. The interrupt generation itself can be enabled with the anim_int_en flag (OSDP_CTRL1[5]).

The definition of the OSD pointer RAM is similar to the definitions of the OSD bitmap RAM. The data is written MSB-aligned to the OSDP_DEF register using 2 bits per pixel. It is written starting from the pixel-exact coordinates given with the OSD pointer cursor (OSDP_CX and OSDP_CY). Instead of using a masked writing the definition width giving the number of pixels that are used from the OSDP_DEF register and written from the given start position can be set via the OSDP_DW register. As in the preceding OSD units also the OSD pointer uses an auto-increment always setting the cursor to the following definition position on each write to OSDP_DEF where the increment is depending on the actual used defwidth. Together with stopped SAA6713AH register auto-increment at OSDP_DEF, this allows a fast burst definition mode that needs 256 I$^2$C-bus byte transmissions to define a complete pointer buffer (see Table 47).

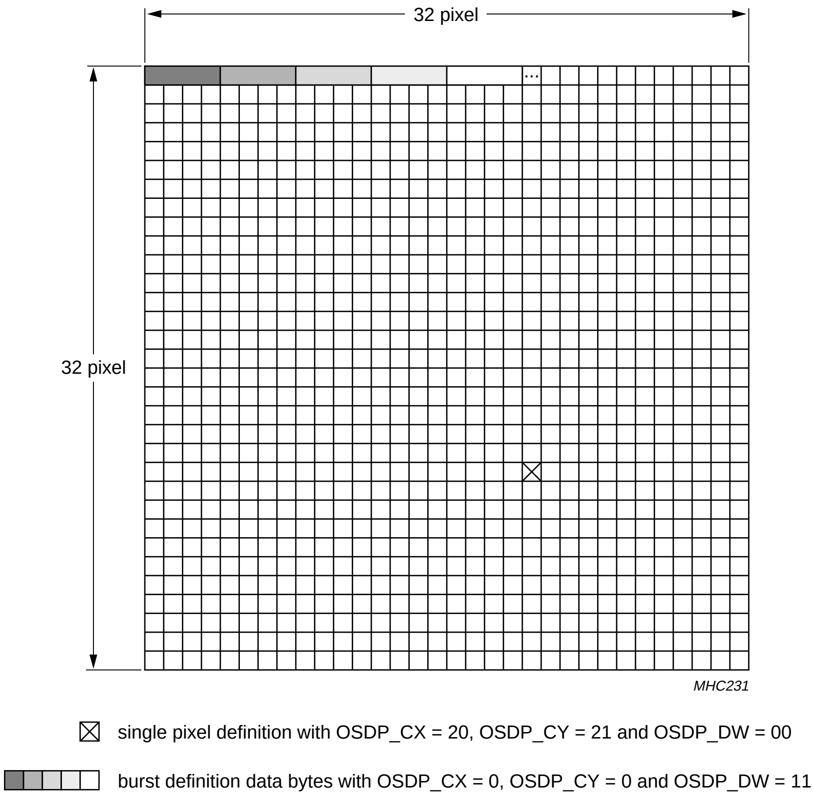



Fig.24 Data format of the OSD pointer.

**Table 47** OSD pointer definition width

| defwidth[1:0] | PIXELS TO BE DEFINED | USED BITS FROM OSDP_DEF | NEEDED TRANSMISSIONS FOR A COMPLETE BUFFER |
|---|---|---|---|
| 00 | 1 | 7 and 6 | 1024 |
| 01 | 2 | 7 to 4 | 512 |
| 10 | 3 | 7 to 2 | 342 |
| 11 | 4 | 7 to 0 | 256 |

#### 7.13.4 HOW TO USE OSD

##### *7.13.4.1 How to create a simple single colour OSD text*

1. Define the desired font size you want to use (OSDT_FR_X and OSDT_FR_Y).
2. If RAM font is needed: set OSDT_SC to logic 0, set OSDT_CC_HI to logic 0, set OSDT_MASK to FFH and define as many characters as wished by sending the needed number of data bytes to OSDT_CDEF preferable using an I$^2$C-bus burst transmission.
3. Define the OSD text window size (OSDT_WX and OSDT_WY), set the cursor to OSDT_CURX = 0 and OSDT_CURY = 0.
4. Set OSDT_MASK to FFH forcing all data to be written, all data to be configured.
5. Define the window content by all three OSDT_PROP registers defining the attributes, colours and charcodes. Use an I$^2$C-bus burst transmission to speed up the programming.
6. Set the desired position and orientation and enable the OSD text with text_on flag that resides in OSDT_CTRL0.

##### *7.13.4.2 How to make changes to a displayed OSD text*

1. Just set the cursor to the desired position and set the desired mask and write mode.
2. Overwrite the character by writing the new OSDT_PROP registers defining new attributes, colours or charcodes.

##### *7.13.4.3 How to create fade-in and fade-out effects*

1. Define the desired elements of the OSD text window to be alpha-blended.
2. Modify the values of OSDT_BGA every few frames in the desired direction by a certain value.

##### *7.13.4.4 How to display a company logo*

1. Define the OSD bitmap part in the needed resolution and the available colour depth.
2. Set the OSDB cursor to 0,0; set OSDB_MASK to FFH.
3. Send all needed bytes with the correct used bits per pixel to OSDB_DEF register, preferable in a burst sequence and turn the OSD bitmap on.

##### *7.13.4.5 How to use pointer animation*

1. Set the OSDP cursor to 0,0 and OSDP_DW to ‘11’.
2. Define the desired animation speed via OSDP_AD, enable the pointer animation interrupt and enable automatic switching.
3. On each interrupt send a 256 byte burst containing the next picture of the animations to OSDP_DEF. It should be noted that this must be finished before the next interrupt arrives.

##### *7.13.4.6 Remarks on the configuration of the OSD*

The three OSD parts can be used independently. If all three parts are turned off, the whole OSD module will be bypassed and clocked down to reduce the power consumption.

Most of the registers of the OSD can be reprogrammed during processing except some needed definition parameters e.g. the resolution and sizes that need to be defined at start-up in order to guarantee correct address calculations.

Before defining the font RAM a valid font size, a valid charcode and a valid sc_startcode must be defined.
A burst definition with new address calculations to the OSD font RAM is only possible either in the multicolour or the single colour area of the memory. So if both areas are to be defined you should define the RAM in two bursts, one for the multicolour and one for the single colour characters. With some effort it is of course possible to write down a user-packed byte burst to speed up the software init that includes all the multicolour and single colour information and create the corresponding font size afterwards.

If something is not displayed as expected, you should carefully check the write mode. Data will only be accepted when all of the corresponding OSDT_PROP registers are written.

To speed up clears or highlighting, the areafill function should be used. By setting the areafill_start bit, an area of the text window within the defined area boundaries is overwritten using the actual settings of OSDT_PROP[2:0] registers and the OSDT_MASK register.

The text shadow is generated over the whole OSD and just displayed in the enabled characters. So it is **not** character bound. This means that a neighbouring 'non-shadowed' character can throw part of its shadow in the shadow allowed character. Also the text shadow is only able to work correct if the whole OSD is inside the picture boundaries and will be turned off automatically if this is not the case. The generated shadow is treated as background, so the shadow is alpha-blended with the background alpha-blending factor, but the shadow is **never** displayed transparent.

During any configuration with the cursors in either the OSD window or one of the pixel addresses be aware that the cursor will 'wrap around' if the calculated address exceeds the physical memory address space.

### 7.14 Colour look-up table

The colour look-up table unit (or gamma correction unit) performs gamma correction and colour component brightness and contrast adjustment. Each 8-bit RGB component value is mapped to a programmable 10-bit value by using it as an index for a look-up table that returns the corresponding image value. The colour components are processed by three independent tables.

The output value for each index value is programmed by writing the 8-bit index to register CL_INDEX and then programming registers CL_VALUE_HI (02H) and CL_VALUE_LO (03H) with the 10-bit image value. Each of the three look-up tables is individually activated for programming by setting red_prog, geen_prog or blue_prog respectively to logic 1.

The activated tables are updated with the new value pair when the lower byte of CL_VALUE_LO (03H) was written. To support quick programming of consecutive values, the index value is incremented after every completed write, so CL_INDEX does not have to be reprogrammed for every data pair. Also the $I^2C$-bus subaddress auto-increment is overridden when writing to CL_VALUE_LO. Instead the subaddress for the next write is determined according to register CL_CTRL. If quick_prog is set logic 0 the subaddress for the next write is set back to 02H (CL_VALUE_HI); otherwise it remains 03H (CL_VALUE_LO), which allows sequential writes of the lower byte only.

As the look-up tables can only be either written or read at the same time, during write operations with activated colour look-up the tables are bypassed. To avoid any influence on the output picture, write_hsynced can be set to logic 1 to update the look-up tables only during horizontal blanking, which slows down programming speed.

Colour look-up is enabled by setting cc_on to logic 1; otherwise the colour look-up tables are in bypass mode and the image values consist of the original value in the upper eight bits and both LSBs are set to logic 0. Programming of the look-up tables is possible in bypass mode or during data processing.

### 7.15 Dithering unit

The dither unit improves the visual quality of displays with only 6-bit or 8-bit physical colour resolution to a virtual colour depth of 10 bits. This is achieved through temporal variation of the physically possible colour values.
To reduce artefacts of the temporal variation neighbouring pixels follow different sequences of variation. The dithering unit registers are mapped to page 10, registers 80H to 83H.

Dithering is switched on if dither_bypass is set to logic 0; otherwise the dithering unit is bypassed. The colour depth of the target display is selected by dither_out_bits. For an 8-bit panel dither_out_bits is set to logic 1; for a 6-bit panel the programming bit is set to logic 0.

Bits dither_idx_ofs_reg[2:0] give a choice of variation sequences (see Table 48). Best quality is expected for most displays with the setting random.

**Table 48** Dithering sequences; note 1

| dither_idx_ofs_reg[2:0] | SEQUENCE |
|---|---|
| 000 | constant zero |
| 001 | 2 × 2 Bayer |
| 010 | 4 × 4 Bayer |
| 011 | 5 × 5 special |
| 1XX | random |

**Note**

1. X = don't care.

Additionally, the unit adds LSB noise to the 10-bit colour values from the colour look-up table, when enabled by dither_add_noise = 1, which improves visual display quality of certain 10-bit displays (e.g. plasma displays). The noise includes only one LSB if dither_noise_mag is set to logic 0; otherwise two LSBs.

Configuration parameters dither_colmap, dither_rand_mono and dither_rand_mode are for test purposes and should be left in their reset values.

### 7.16 Output interface

The Output Interface (OIF) provides picture data and command signals to the display. Programming the output interface, the output frame geometry can be defined. As most displays require continuous data stream during one frame or line, it is possible to define wait points. There are different possibilities how to map the RGB data to the output ports PA to PF.

The SAA6713AH does not have particular output ports for panel signals VSYNC, HSYNC or DE. Instead, there are in total 10 Configurable Signal Generator (CSG) outputs which are driven by free programmable CSGs.

All output interface programming registers are mapped to the $I^2C$-bus configuration register page 11.

#### 7.16.1 DEFINITION OF THE OUTPUT FRAME GEOMETRY

The total output frame area (main frame) is defined by blank_line_length and last_line (registers OI_FX and OI_FY). It consists of the visible data (active frame) and the invisible data (blanking); see Fig.25 and Table 49.

The active frame is divided into border and picture area. The picture area includes the data from the input stream. The border area is around the picture area. If no border is needed, the register values of picture and active area have to be equal. The active frame can be put anywhere inside the main frame except in the first row or in the first column of the main frame.

The geometric values for the frames/areas depend on the display and the timing. If the picture values are not correct data may be lost or missing data will be replaced by border colour. The serial output begins in the upper left corner of the main frame in row 1 and column 1. The active frame starts in row active_start_y and in column active_start_x (point a in Fig.25). Values 0 are not allowed and the active frame or the picture area cannot start in column one. The picture area has to be contained in the active frame, at maximum it may be identical to the active frame.

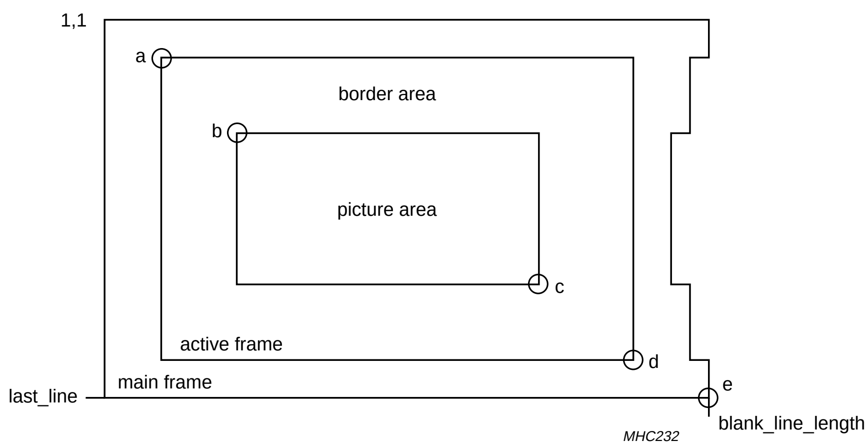



Fig.25 Output frame set-up.

**Table 49** Programmable geometric values

| POINT | HORIZONTAL | VERTICAL |
|---|---|---|
| a | active_start_x (OI_ASX) | active_start_y (OI_ASY) |
| b | picture_start_x (OI_PSX) | picture_start_y (OI_PSY) |
| c | picture_end_x (OI_PEX) | picture_end_y (OI_PEY) |
| d | active_end_x (OI_AEX) | active_end_y (OI_AEY) |
| e | blank_line_length (OI_FX) | last_line (OI_FY) |

To make timing adaption of output to input frame easier, each area has its own line length, i.e. the blanking behind the border area can be freely adjusted. As some panels are sensitive to different line lengths, they should differ as little as possible. All line length values have to be greater or equal to active_end_x. Parameter active_end_x must be greater or equal to picture_end_x. The according programming registers are listed in Table 50.

Border and blanking colour are freely programmable as described in Table 51.

**Table 50** Line length values

| AREA | LINE LENGTH |
|---|---|
| Blanking | blank_line_length (OI_FX) |
| Border | active_line_length (OI_ALX) |
| Picture | picture_line_length (OI_PX) |

**Table 51** Border and blanking colour

| AREA | REGISTER |
|---|---|
| Blanking | OI_BLC_R, OI_BLC_G and OI_BLC_B |
| Border | OI_BOC_R, OI_BOC_G and OI_BOC_B |

### 7.16.2 WAIT MODES

It is not necessary to match the output timing exactly to the input timing. The output timing can be a little faster. In this event it may happen that no valid data is available at the OIF. As the output stream to the panel should not be interrupted during the output of a frame or line, a line wise and a frame wise wait mode is available.

The wait_column (register OI_WX) has to be programmed. During output of lines inside the picture area, the output stream stops at this defined wait_column and waits until new picture data is available. If fieldwise wait mode is programmed, the output only stops at the wait_column of the first line of the picture area. The wait_column must be located in front of the active area (see Fig.26). Additionally, there is a free-running mode without any wait point.

The wait modes are programmed in register OI_WM according to Table 52.

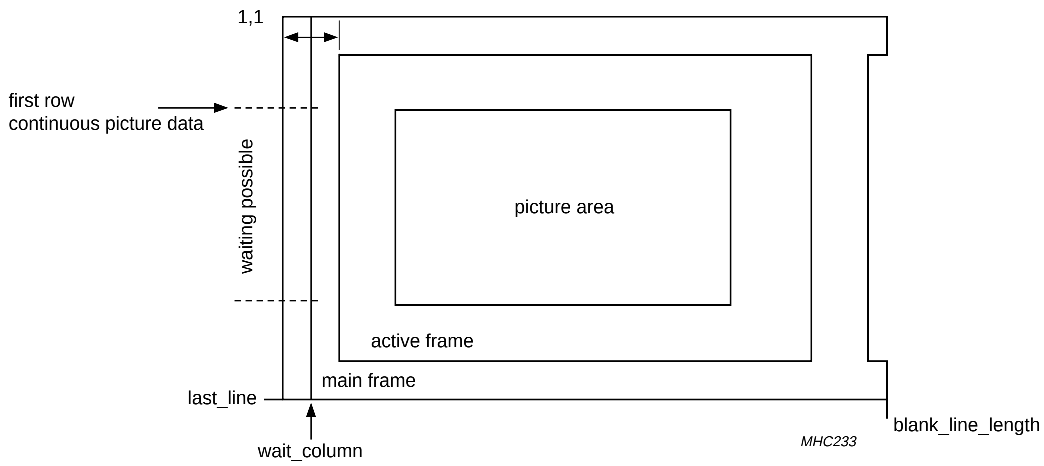



Fig.26 Wait column.

**Table 52** Wait modes

| wait_mode[1:0] | MODE | ACTION |
|---|---|---|
| 11 | free-running | no waiting |
| 10 | | |
| 01 | row stop | waiting in each row of picture area |
| 00 | one stop | waiting in first row of picture area once each frame |

7.16.3 DATA TO OUTPUT MAPPING

Each colour of each pixel is handled separately. In double pixel mode there are 6 bytes (red, green and blue of pixel 0 and pixel 1 from which pixel 0 arrives first). In single pixel mode there are 3 bytes (red, green and blue of pixel 0). Each of the six output ports (see Fig.27) can be connected to each colour and can be inverted, swapped and aligned to the MSB (sensible to drive 6-bit panels).

Registers OI_B0R, OI_B0G, OI_B0B, OI_B1R, OI_B1G and OI_B1B have to be programmed according to Table 53.

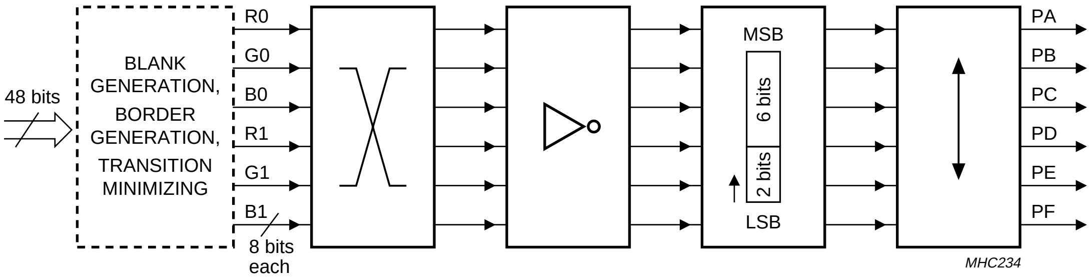



Fig.27 Data to output mapping schema.

**Table 53** Data to output mapping; note 1

| BIT | FUNCTION | ACTION |
|---|---|---|
| MSB_align | alignment | If this bit is set to logic 1, then the lower 6 bits are aligned to MSB. |
| swap | swapping | If this bit is set to logic 1, then swap [7:0] to [0:7]; LSB becomes MSB. |
| inv | inversion | If this bit is set to logic 1, then bit wise inversion of the colour component. |
| port_x_conf[2:0] | allocation | Output port Px gets data byte with 6 bytes in double pixel mode or 3 bytes in single pixel mode (for x = A to F):<br>11X: 0R<br>101: 0G<br>100: 0B<br>01X: 1R<br>001: 1G<br>000: 1B |

**Note**

1. X = don’t care.

#### 7.16.4 CONFIGURABLE SIGNAL GENERATORS

There are 10 configurable signal generators available. The functionality is particularly designed to drive displays directly without the use of a Timing Controller (TCON). For operation with hsync, vsync and data enable only three generators are needed.

All CSGs have the same basic structure (see Fig.28). There are two programmable action points: the start point (a) and the end point (b). The start point describes the upper left corner of the operation window. The end point the lower right corner. When the row and line counter values of the output interface are equal to the action point values, the output becomes HIGH or LOW according to the set-up. The possible actions for the start point are set or toggle and for the end point reset or toggle. The output signal can be inverted additionally.

Two modes are available: frame or line based. In frame based mode the signal only changes in the upper, left and the lower, right corner of each frame. In line based mode the signal changes every line at the beginning and at the end of the operation window. It is also possible to use just one action point, e.g. to toggle the output each line or just once in a frame. To disable an action point a logic 0 has to be programmed to the y-value.

CSG0 and CSG1 are driven by two separate signal generators (CSG0A or CSG0B and CSG1A or CSG1B respectively) allowing a more complex signal to be generated.

The operation window upper left corner (a) is defined by the OI_GxSX and OI_GxSY registers and the lower right corner (b) is described by the OI_GxEX and OI_GxEY registers (x out of 0 to 9). The action is defined by programming the configuration register OI_GxC. The corners (a and b) of the operation window are the action points. In each action point the output signal is modified as described in the configuration register. At the first action point (a) the output will be set or toggled. At the second action point the output signal will be reset or toggled again.

Example: In order to define a DE signal, the CSG window is set to the active area. Bit frame or bit line of the concerned CSG control register is set to line mode and the CSG signal is set to logic 1 at point 1 and set to logic 0 at point 2 (see Figs 29 and 30).

**Important programming hint:** The horizontal start values (x-values) of the action points describe the offset from the beginning of the line. If you want to start e.g. CSG0 at (2,3) you have to program the values (1,3). If you want to stop the signal after (12,14) you have to program the values (12,14) so the signal changes its value at the end of position 12 (edge to position 13). An offset of 0 is not allowed. Avoid using the same column as wait column.

There are 4 groups of CSGs. The CSGs of each group have some other additional features.

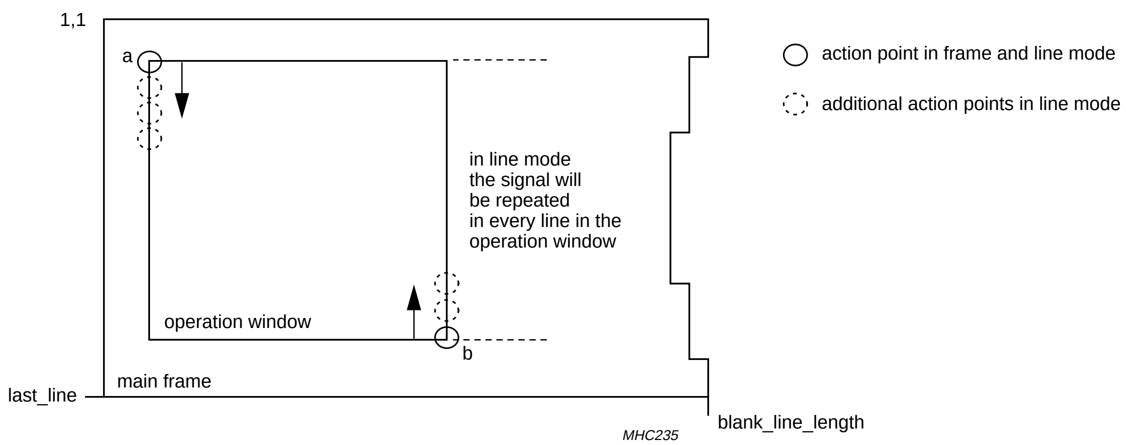



Fig.28 Output frame set-up.

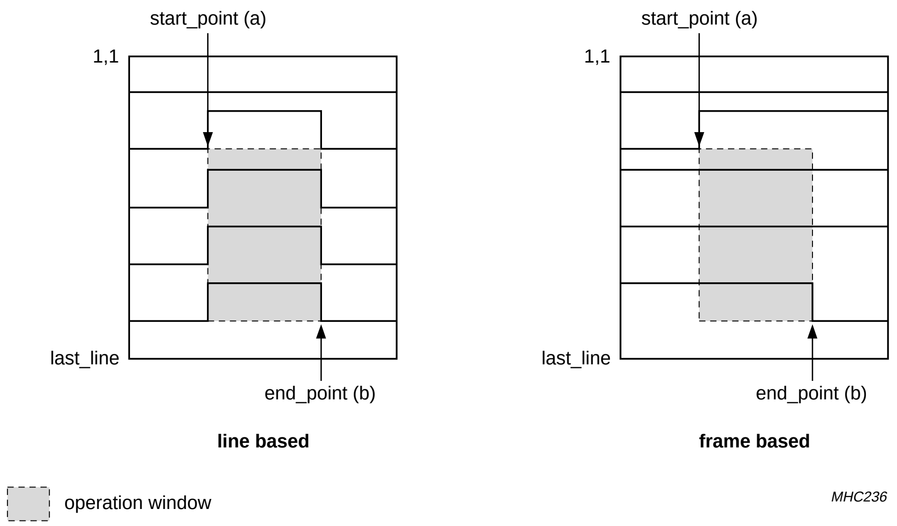



Fig.29 Examples for signal generator outputs.

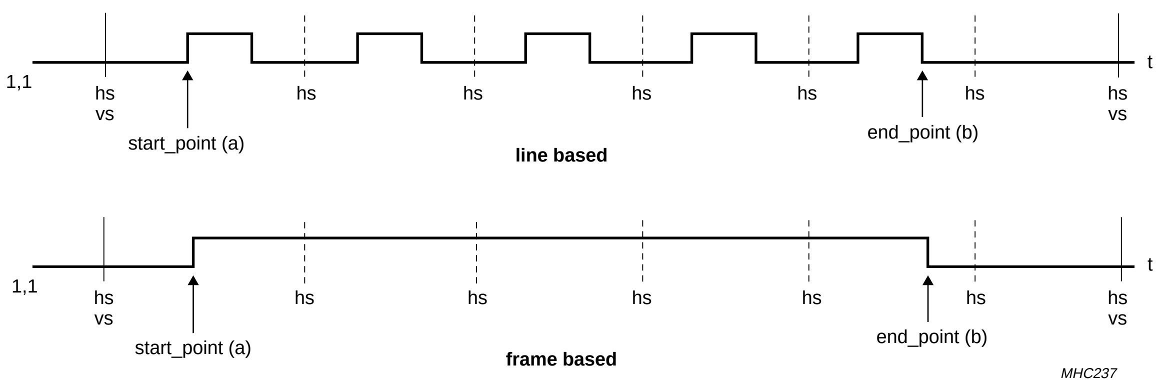



Fig.30 Examples for signal generator outputs.

#### 7.16.5 SPECIAL FEATURES AND CSG GROUPS

##### *7.16.5.1 CSG0 and CSG1*

These CSGs are two CSGs with one output (see Fig.31). They are splitted in CSGXa and CSGXb. The a and b part are equal and the programming is as two separated CSGs. So one can generate signals with four events each line/frame.

##### *7.16.5.2 CSG2 to CSG5*

Normal CSGs with two action points. Additionally, CSG2 plus CSG3 and CSG4 plus CSG5 can cooperate like CSG0/1 a/b on outputs CSG3 and CSG5.

##### *7.16.5.3 CSG6 and CSG7*

No special or additional features. Two action points.

##### *7.16.5.4 CSG8 and CSG9*

These CSGs have an additional action point. The signal can be set, reset or toggled in this point. The execution of action point 0 can be depressed only in line mode for every second line. The execution of action point 1 and 2 is not influenced by skip_mode.

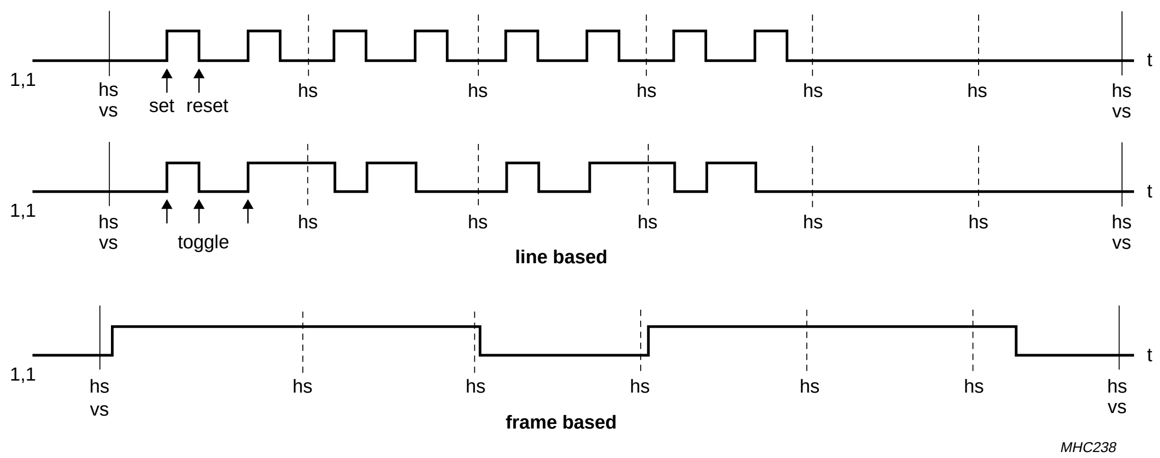



Fig.31 Examples for CSG0/1 outputs.

#### 7.16.6 Transition minimizing

The transition minimizing programming is done in the OI_CTRL0 register in the OIF. This section describes how the OIF pixel input operates with INVA and INVB outputs for various values of ivsl0 and ivsl1 (register OI_CTRL0). All modes are designed for double pixel handling.

##### *7.16.6.1 Bit ivsl0 = 1 and bit ivsl1 = 0*

INVA operates with input pixel 0 (means the first of a couple). This inversion operation considers a total of 18 bits: the 6 high order bits of each colour component of input pixel 0 (R2 to R7, G2 to G7 and B2 to B7). The input data of time (n – 1) is compared to the data at time n. If 10 or more bits within the 18 bits have changed from LOW to HIGH or from HIGH to LOW, then INVA toggles between HIGH and LOW: if INVA was HIGH at (n – 1) it goes LOW, and if it was LOW at (n – 1), it toggles HIGH. If 9 or fewer bits within the 18 bits have changed from HIGH to LOW or from LOW to HIGH, then INVA does not toggle. When INVA is HIGH, all bits (24 bits) of pixel 0 to output (means data before ‘data to output mapping’) are inverted.

INVB operates with input pixel 1 (means the second of a couple). This inversion operation considers a total of 18 bits: the 6 high order bits of each colour component of input pixel 1 (R2 to R7, G2 to G7 and B2 to B7). The input data of time (n – 1) is compared to the data at time n. If 10 or more bits within the 18 bits have changed from LOW to HIGH or from HIGH to LOW, then INVB toggles between HIGH and LOW: if INVB was HIGH at (n – 1), it goes LOW, and if it was LOW at (n – 1), it toggles HIGH. If 9 or fewer bits within the 18 bits have changed from HIGH to LOW or from LOW to HIGH, then INVB does not toggle. When INVB is HIGH, all bits (24 bits) of pixel 1 to output are output inverted.

##### *7.16.6.2 Bit ivsl0 = 1 and bit ivsl1 = 1*

INVA and INVB both operate with input pixel 0 and 1.This inversion operation considers a total of 36 bits, the 6 high order bits of each colour component of pixel 0 and 1 (0R2 to 0R7, 0G2 to 0G7, 0B2 to 0B7, 1R2 to 1R7, 1G2 to 1G7 and 1B2 to 1B7). The input data of time (n – 1) is compared to the data at time n. If 19 or more bits within the 36 bits have changed from LOW to HIGH or from HIGH to LOW, then both INVA and INVB toggle between HIGH and LOW. When INVA and INVB are HIGH at (n – 1), they go LOW, and when they are LOW at (n – 1), they toggle HIGH. If 18 or fewer bits within the 36 bits have changed from HIGH to LOW or from LOW to HIGH, then INVA and INVB do not toggle. When INVA and INVB are both HIGH, all bits (48 bits) are inverted.

Because there is no previous data for the first data in every column (horizontal period), the above noted toggle operations for INVA and INVB, as well as the data inversion operations are not performed. In the event of first data output of every column, INVA and INVB are set to LOW, and data is not inverted.

##### *7.16.6.3 Bit ivsl0 = 0 and bit ivsl1 = 0*

For input pixel, data inversion is similar to when ivsl0 = 1, ivsl1 = 0, with input pixel 0 and 1 being separated, and the outputs being driven according to the results of calculations.

For INVA and INVB signals, the calculations are similar to when ivsl0 = 1, ivsl1 = 0, but the INVA and INVB outputs are driven as logical opposites.

##### *7.16.6.4 Bit ivsl0 = 0 and bit ivsl1 = 1*

The INVA and INVB signals are always driven LOW and data inversion operations are not performed.

#### 7.16.7 Background and emergency frame generator

The output interface includes a simple frame generator. It may be useful when the front-end receives no signal, so no front-end clock is available. The generated frame has the same dimensions as the picture area. The frame colour is programmable (OI_FCx). The on screen display is still working. The generator may be switched on via the OI_FC_EN register.

#### 7.16.8 GLOBAL CONTROL

The output interface has four global modes, which can be programmed with bits OI_enable, power_down and blank_mode (register OI_CTRL0) according to Table 54.

The blank colour is programmable via bits blank_colour_red, blank_colour_green and blank_colour_blue.

**Table 54** Global modes; note 1

| OI_ enable | power_ down | blank_ mode | MODE | ACTION |
|---|---|---|---|---|
| 1 | 0 | 1 | blank | all colours replaced by blank colour values |
| X | 1 | X | Power-down | all registers set to default, like soft reset; all outputs LOW |
| 0 | 0 | X | disable | all outputs reset but incoming data queued to trash |
| 1 | 0 | 0 | normal | normal operation |

**Note**

1. X = don't care.

#### 7.16.9 PANEL CLOCK

The output interface can handle single and double pixel mode (bit double_pixel in register OI_CTRL1). In single pixel mode one pixel (24 bits) is available each cycle at the output ports. The panel clock PCLK is the same as the back-end clock. In double pixel mode 2 pixels (48 bits) are available at the output ports. The PCLK in double pixel mode changes every second cycle of the back-end clock. The panel clock polarity can be inverted by setting PCLK_pol of register OI_CTRL1 to logic 1. At the beginning of each frame the PCLK is synced again. It is very important that the number of pixels in a double pixel frame is even.

The horizontal sync signal of the VGA video input source may be used as a reference clock for the panel PLL (see Table 55). This allows more stable locking of the panel timing to the source timing. In this mode the PLL will be 'coasted' during vertical sync when a composite sync or sync-on-green is enabled (iif_cs_sog_en = 1).

**Table 55** Panel PLL

| pll_src | FUNCTION |
|---|---|
| 0 | pre-divided clock |
| 1 | HS_PLL |

#### 7.16.10 HOW TO START THE OUTPUT INTERFACE

**Table 56** Starting output interface

| STEP | ACTION |
|---|---|
| 1 | set-up frame geometry |
| 2 | set-up signal generators |
| 3 | set-up wait column and wait mode |
| 4 | set-up PCLK and pixel mode |
| 5 | enable output interface |

#### 7.16.11 PROGRAMMABLE OUTPUT DRIVE STRENGTH

For all data and control signals of the output interface (PA[7:0], PB[7:0], PC[7:0], PD[7:0], PE[7:0], PF[7:0], CSG[9:0], INVA, INVB, OUTEN and PWM) a programmable output drive strength up to 15 mA is provided (in 8 steps and starting at 2.9 mA); see Table 57.

For the PCLK output, a programmable output drive up to 30 mA is provided (in 8 steps and starting at 5.8 mA); see Table 57.

Individual drive strength programming is possible for each 8-bit group of data signals (see Table 58). The drive strength of control and clock signals are programmable individually. This is necessary to drive the multiple source and gate drivers directly.

**Table 57** Programmable drive strength

| DS2 | DS1 | DS0 | DATA AND CONTROL OUTPUTS (mA) | PCLK OUTPUT (mA) |
|---|---|---|---|---|
| 0 | 0 | 0 | 2.9 | 5.8 |
| 0 | 0 | 1 | 3.4 | 6.8 |
| 0 | 1 | 0 | 4 | 8 |
| 0 | 1 | 1 | 5 | 10 |
| 1 | 0 | 0 | 6 | 12 |
| 1 | 0 | 1 | 8 | 16 |
| 1 | 1 | 0 | 11 | 22 |
| 1 | 1 | 1 | 15 | 30 |

**Table 58** Output interface drive strength

| BIT | DESCRIPTION | REMARK |
|---|---|---|
| pin_drv_inva[2:0] | output drive strength for INVA | from 2.9 mA (reset) to 15 mA; see Table 57 |
| pin_drv_invb[2:0] | output drive strength for INVB | |
| pin_drv_pa[2:0] | output drive strength for PA | |
| pin_drv_pb[2:0] | output drive strength for PB | |
| pin_drv_pc[2:0] | output drive strength for PC | |
| pin_drv_pd[2:0] | output drive strength for PD | |
| pin_drv_pe[2:0] | output drive strength for PE | |
| pin_drv_pf[2:0] | output drive strength for PF | |
| pin_drv_csg0[2:0] | output drive strength for CSG0 | |
| pin_drv_csg1[2:0] | output drive strength for CSG1 | |
| pin_drv_csg2[2:0] | output drive strength for CSG2 | |
| pin_drv_csg3[2:0] | output drive strength for CSG3 | |
| pin_drv_csg4[2:0] | output drive strength for CSG4 | |
| pin_drv_csg5[2:0] | output drive strength for CSG5 | |
| pin_drv_csg6[2:0] | output drive strength for CSG6 | |
| pin_drv_csg7[2:0] | output drive strength for CSG7 | |
| pin_drv_csg8[2:0] | output drive strength for CSG8 | |
| pin_drv_csg9[2:0] | output drive strength for CSG9 | |
| pin_drv_pwm[2:0] | output drive strength for PWM | |
| pin_drv_outen[2:0] | output drive strength for OUTEN | |
| pin_drv_pclk[2:0] | output drive strength for PCLK | from 5.8 mA (reset) to 30 mA; see Table 57 |

#### 7.16.12 ADJUSTABLE OUTPUT DELAYS

Every output pin, except pin PWM, can be delayed. The delay increment is 0.36 ns. The programming value is 5-bit wide (see Table 59).

**Table 59** Data to output mapping

| REGISTER | BIT | OUTPUT |
|---|---|---|
| OI_INVA_DEL | inversion_A_pin_delay[4:0] | INVA |
| OI_INVB_DEL | inversion_B_pin_delay[4:0] | INVB |
| OI_PAD | pin_delay[4:0] | PA |
| OI_PBD | pin_delay[4:0] | PB |
| OI_PCD | pin_delay[4:0] | PC |
| OI_PDD | pin_delay[4:0] | PD |
| OI_PED | pin_delay[4:0] | PE |
| OI_PFD | pin_delay[4:0] | PF |
| OI_CTRL1 | PCLK_pin_delay[4:0] | PCLK |
| OI_G0BD | pin_delay[4:0] | CSG0 |
| OI_G1BD | pin_delay[4:0] | CSG1 |
| OI_G2D | pin_delay[4:0] | CSG2 |
| OI_G3D | pin_delay[4:0] | CSG3 |
| OI_G4D | pin_delay[4:0] | CSG4 |
| OI_G5D | pin_delay[4:0] | CSG5 |
| OI_G6D | pin_delay[4:0] | CSG6 |
| OI_G7D | pin_delay[4:0] | CSG7 |
| OI_G8D | pin_delay[4:0] | CSG8 |
| OI_G9D | pin_delay[4:0] | CSG9 |

#### 7.16.13 PULSE WIDTH MODULATION

A pulse width modulated signal can be generated for brightness control of the panel. The pulse width and the pre-divider value can be programmed. The PWM can be synced with the h-gate. The logical polarity can be inverted.

The PWM runs with the system clock and can be divided by the pre-divider. A period depends on 256 cycles.

The configuration registers for the PWM are OI_PWM0 and OI_PWM1.

#### 7.16.14 RESET BEHAVIOUR

A hardware reset forces all true bidirectional pins (PAx, PBx, PCx, VCLK, VSYNC and SDA) to input. Their output functionality must be explicitly invoked by software. CSG2/A0 and CSG4/A1 are input during the hardware reset for latching in the configuration data and switched to output immediately after hardware reset.

## 8 BOUNDARY SCAN TEST

The SAA6713AH has built-in logic and 5 dedicated pins to support boundary scan testing which allows board testing without special hardware (nails). The SAA6713AH follows the *“IEEE Std. 1149.1 - Standard Test Access Port and Boundary-Scan Architecture”* set by the Joint Test Action Group (JTAG) chaired by Philips.

The 5 special pins are: Test Mode Select (TMS), Test Clock (TCK), Test Reset ($\overline{\text{TRST}}$), Test Data Input (TDI) and Test Data Output (TDO).

The Boundary Scan Test (BST) functions BYPASS, EXTEST, INTEST, SAMPLE, CLAMP and IDCODE are all supported (see Table 60). Details about the JTAG BST-TEST can be found in the specification *“IEEE Std. 1149.1”*.

A file containing the detailed Boundary Scan Description Language (BSDL) description of the SAA6713AH is available on request.

### 8.1 Initialization of boundary scan circuit

The Test Access Port (TAP) controller of an IC should be in the reset state (TEST_LOGIC_RESET) when the IC is in the functional mode. This reset state also forces the instruction register into a functional instruction such as IDCODE or BYPASS.

To solve the power-up reset, the standard specifies that the TAP controller will be forced asynchronously to the TEST_LOGIC_RESET state by setting pin $\overline{\text{TRST}}$ to LOW.

### 8.2 Device identification codes

A device identification register is specified in *“IEEE Std. 1149.1b-1994”*. It is a 32-bit register which contains fields for the specification of the IC manufacturer, the IC part number and the IC version number. Its biggest advantage is the possibility to check for the correct ICs mounted after production and determination of the version number of ICs during field service.

When the IDCODE instruction is loaded into the BST instruction register, the identification register will be connected between pins TDI and TDO of the IC. The identification register will load a component specific code during the CAPTURE_DATA_REGISTER state of the TAP controller and this code can subsequently be shifted out. At board level this code can be used to verify component manufacturer, type and version number. The device identification register contains 32 bits, numbered 31 to 0, where bit 31 is the most significant bit (nearest to TDI) and bit 0 is the least significant bit (nearest to TDO); see Fig.32.

**Table 60** BST instructions supported by the SAA6713AH

| INSTRUCTION | DESCRIPTION |
|---|---|
| BYPASS | This mandatory instruction provides a minimum length serial path (1 bit) between TDI and TDO when no test operation of the component is required. |
| EXTEST | This mandatory instruction allows testing of off-chip circuitry and board level interconnections. |
| SAMPLE | This mandatory instruction can be used to take a sample of the inputs during normal operation of the component. It can also be used to preload data values into the latched outputs of the boundary scan register. |
| CLAMP | This optional instruction is useful for testing when not all ICs have BST. This instruction addresses the bypass register while the boundary scan register is in external test mode. |
| IDCODE | This optional instruction will provide information on the components manufacturer, part number and version number. |
| INTEST | This optional instruction allows testing of the internal logic (no customer support available). |
| USER1 | This private instruction allows testing by the manufacturer (no customer support available). |

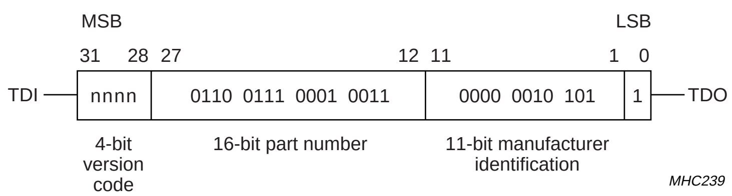



Fig.32 32 bits of identification code.

## 9 LIMITING VALUES

In accordance with the Absolute Maximum Rating System (IEC 60134)

| SYMBOL | PARAMETER | CONDITIONS | MIN. | MAX. | UNIT |
|---|---|---|---|---|---|
| $V_{DDD(IC)}$ | digital supply voltage for internal core on pins $V_{DDD(IC1)}$ to $V_{DDD(IC9)}$ | | –0.5 | +3.3 | V |
| $V_{DDA}$ | analog supply voltage on pins $V_{DDA(R)}$, $V_{DDA(G)}$, $V_{DDA(B)}$, $V_{DDA(ADC)(R)}$, $V_{DDA(ADC)(G)}$ and $V_{DDA(ADC)(B)}$ | | 0 | 2.8 | V |
| $V_{DD(PLL)}$, $V_{DDD(PLL)}$, $V_{DDA(PLL)}$, | supply voltage for PLL on pins $V_{DD(PLL)(P)}$, $V_{DDD(PLL)(S)}$ and $V_{DDA(PLL)(S)}$ | | 0 | 2.8 | V |
| $V_{DDA(IB)}$ | analog supply voltage for input buffer on pin $V_{DDA(IB)}$ | | 0 | 3.3 | V |
| $V_{DDD(EP)}$ | external digital pad supply voltage for pins $V_{DDD(EP1)}$ to $V_{DDD(EP10)}$ | | –0.5 | +4.2 | V |
| $V_{DDA(EP)}$ | external analog pad supply voltage for pin $V_{DDA(EP)}$ | | 0 | 3.6 | V |
| $V_n$ | voltage on | | | | |
| | digital input pins SDA and SCL (5 V tolerant) | | –0.5 | +5.8 | V |
| | digital input pins | note 1 | –0.5 | $V_{DDD(EP)}$ + 0.5 | V |
| | analog input and output pins | | –0.5 | $V_{DDA}$ + 0.5 | V |
| $P_{tot}$ | total power dissipation | | – | 1.7 | W |
| $T_{stg}$ | storage temperature | | –25 | +150 | °C |
| $T_j$ | junction temperature | | – | 125 | °C |
| $T_{amb}$ | ambient temperature | | 0 | 70 | °C |
| $V_{esd}$ | electrostatic discharge voltage | note 2 | –1500 | +2000 | V |
| | | note 3 | –150 | +150 | V |

**Notes**

1. May not exceed 4.2 V; including outputs in 3-state mode; only when supply voltages are present.
2. Human body model: C = 100 pF; R = 1.5 kΩ.
3. Machine model: C = 200 pF; L = 0.75 μH; R = 0 Ω.

## 10 THERMAL CHARACTERISTICS

| SYMBOL | PARAMETER | CONDITIONS | VALUE | UNIT |
|---|---|---|---|---|
| $R_{th(j-a)}$ | thermal resistance from junction to ambient | in free air | 26 | K/W |

## 11 CHARACTERISTICS

$T_{amb}$ = 25 °C; unless otherwise specified.

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| **Supplies** | | | | | | |
| DIGITAL SUPPLY FOR INTERNAL CORE: PINS $V_{DDD(IC1)}$ TO $V_{DDD(IC9)}$ | | | | | | |
| $V_{DDD(IC)}$ | supply voltage | | 2.3 | 2.5 | 2.7 | V |
| $I_{DDD(IC)}$ | supply current | note 1 | – | 90 | – | mA |
| $P_{DDD(IC)}$ | power dissipation | note 1 | – | 225 | – | mW |
| ANALOG SUPPLY FOR COLOUR CHANNELS AND ADCS: PINS $V_{DDA(R)}$, $V_{DDA(G)}$, $V_{DDA(B)}$, $V_{DDA(ADC)(R)}$, $V_{DDA(ADC)(G)}$ AND $V_{DDA(ADC)(B)}$ | | | | | | |
| $V_{DDA}$ | supply voltage | | 2.3 | 2.5 | 2.7 | V |
| $I_{DDA}$ | supply current | note 1 | – | 200 | – | mA |
| $P_{DDA}$ | power dissipation | note 1 | – | 500 | – | mW |
| SUPPLY FOR PLL: PINS $V_{DD(PLL)(P)}$, $V_{DDA(PLL)(S)}$ AND $V_{DDD(PLL)(S)}$ | | | | | | |
| $V_{DD(PLL)}$ | supply voltage | | 2.3 | 2.5 | 2.7 | V |
| $I_{DD(PLL)}$ | supply current | note 1 | – | 5 | – | mA |
| $P_{DD(PLL)}$ | power dissipation | note 1 | – | 13 | – | mW |
| ANALOG SUPPLY FOR INPUT BUFFER: PIN $V_{DDA(IB)}$ | | | | | | |
| $V_{DDA(IB)}$ | supply voltage | | 2.7 | 3.0 | 3.3 | V |
| $I_{DDA(IB)}$ | supply current | note 1 | – | 2 | – | mA |
| $P_{DDA(IB)}$ | power dissipation | note 1 | – | 6 | – | mW |
| DIGITAL SUPPLY FOR PADS: PINS $V_{DDD(EP1)}$ TO $V_{DDD(EP10)}$ | | | | | | |
| $V_{DDD(EP)}$ | supply voltage | | 3.0 | 3.3 | 3.6 | V |
| $I_{DDD(EP)}$ | supply current | note 1 | – | 50 | – | mA |
| $P_{DDD(EP)}$ | power dissipation | note 1 | – | 165 | – | mW |
| ANALOG SUPPLY FOR PAD: PIN $V_{DDA(EP)}$ | | | | | | |
| $V_{DDA(EP)}$ | supply voltage | | 3.0 | 3.3 | 3.6 | V |
| $I_{DDA(EP)}$ | supply current | note 1 | – | 1 | – | mA |
| $P_{DDA(EP)}$ | power dissipation | note 1 | – | 3 | – | mW |
| **Analog front-end inputs** | | | | | | |
| ANALOG VIDEO INPUTS: PINS RIN, GIN AND BIN | | | | | | |
| $V_{i(p-p)}$ | input voltage (peak-to-peak value) | note 2 | 0.2 | – | 0.5 | V |
| $C_i$ | input capacitance | | – | 850 | – | fF |
| SYNC-ON-GREEN SLICER INPUT: PIN SOGIN | | | | | | |
| $V_{i(p-p)}$ | input voltage (peak-to-peak value) | | 0.1 | – | 0.4 | V |

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| ANALOG-TO-DIGITAL CONVERTER | | | | | | |
| $f_{pixel}$ | sample clock of ADC | | 25 | – | 110 | MHz |
| N | resolution of ADC | | – | 8 | – | bits |
| $LE_{dc(i)}$ | DC integral linearity error | | – | 0.8 | – | LSB |
| $LE_{dc(d)}$ | DC differential linearity error | | – | 1.6 | – | LSB |
| ENOB | effective number of bits | $f_{pixel}$ = 110 MHz | 6.6 | 7 | – | bits |
| CONTROL LOOPS FOR CONTRAST AND BRIGHTNESS | | | | | | |
| M | matching of contrast and clamp settings among the three channels | | – | 1 | – | % |
| $B_{loop}$ | bandwidth of contrast and clamp loops | | – | 500 | – | Hz |
| $N_{clamp}$ | required width of clamp pulse | | 40 | – | – | pixels |
| $N_{gainc}$ | required width of gain control pulse | | 96 | – | – | pixels |
| **Digital inputs** | | | | | | |
| CLOCK, RESET AND BST INPUTS: PINS CLK, $\overline{RST}$, TCK, TDI, TMS AND $\overline{TRST}$ | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.7 | V |
| $V_{IH}$ | HIGH-level input voltage | | 1.7 | – | $V_{DDD(EP)}$ | V |
| $I_{IL}$ | LOW-level input current | | – | – | 1 | μA |
| $I_{IH}$ | HIGH-level input current | | – | – | 1 | μA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| HORIZONTAL SYNC INPUT: PIN HSYNC (5 V TOLERANT) | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.8 | V |
| $V_{IH}$ | HIGH-level input voltage | | 2 | – | 5.5 | V |
| $I_{IL}$ | LOW-level input current | | – | – | 1 | μA |
| $I_{IH}$ | HIGH-level input current | | – | – | 1 | μA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| **Output pins** | | | | | | |
| DATA PORTS AND PANEL CONTROL SIGNALS: PINS PD0 TO PD7, PE0 TO PE7, PF0 TO PF7, CSG0, CSG1, CSG3, CSG5 TO CSG9, INVA, INVB, PCLK, PWM AND OUTEN | | | | | | |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 16 mA | – | – | 0.4 | V |
| $V_{OH}$ | HIGH-level output voltage | $I_{OH}$ = –16 mA | $V_{DDD(EP)}$ – 0.4 | – | – | V |
| $I_{pu}$ | pull-up current | $V_i$ = 0 | –23 | –50 | –65 | μA |
| | | $V_{DD}$ < $V_i$ <$V_{DDD(EP)}$; note 3 | – | 0 | – | μA |
| GENERAL CONTROLS: PINS $\overline{INT}$ AND TDO | | | | | | |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 4 mA | – | – | 0.4 | V |
| $V_{OH}$ | HIGH-level output voltage | $I_{OH}$ = –4 mA | $V_{DDD(EP)}$ – 0.4 | – | – | V |

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| **Input or output pins** | | | | | | |
| DATA PORTS, PANEL CONTROL SIGNALS, SAMPLE CLOCK AND VERTICAL SYNC PULSES: PINS PA0 TO PA7, PB0 TO PB7, PC0 TO PC7, CSG2/A0 AND CSG4/A1 | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.8 | V |
| $V_{IH}$ | HIGH-level input voltage | | 2.0 | – | 3.6 | V |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 16 mA | 0 | – | 0.4 | V |
| $V_{OH}$ | HIGH-level output voltage | $I_{OH}$ = –16 mA | $V_{DDD(EP)}$ – 0.4 | – | $V_{DDD(EP)}$ | V |
| $I_{pu}$ | pull-up current | $V_i$ = 0 | –23 | –50 | –65 | µA |
| | | $V_{DD} < V_i < V_{DDD(EP)}$; note 3 | – | 0 | – | µA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| VERTICAL SYNC INPUT OR OUTPUT: PIN VSYNC | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.8 | V |
| $V_{IH}$ | HIGH-level input voltage | | 2.0 | – | 5.5 | V |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 2 mA | 0 | – | 0.4 | V |
| $V_{OH}$ | HIGH-level output voltage | $I_{OH}$ = –2 mA | $V_{DDD(EP)}$ – 0.4 | – | $V_{DDD(EP)}$ | V |
| $I_{pd}$ | pull-down current | $V_i$ = 5 V; note 4 | 18 | 50 | 53 | µA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| SAMPLE CLOCK: PIN VCLK | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.7 | V |
| $V_{IH}$ | HIGH-level input voltage | | 1.7 | – | $V_{DDD(EP)}$ | V |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 4 mA | 0 | – | 0.4 | V |
| $V_{OH}$ | HIGH-level output voltage | $I_{OH}$ = –4 mA | $V_{DDD(EP)}$ – 0.4 | – | $V_{DDD(EP)}$ | V |
| $I_{OZ}$ | 3-state output leakage current | $V_{IH} = V_{DDD(EP)}$; $V_{IL}$ = 0 | – | – | 1 | µA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| **$I^2C$-bus interface** | | | | | | |
| CLOCK INPUT: PIN SCL | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.8 | V |
| $V_{IH}$ | HIGH-level input voltage | 5 V tolerant | 2.0 | – | 5.5 | V |
| $V_{hys}$ | hysteresis voltage | | 0.3 | – | – | V |
| $I_{IL}$ | LOW-level input current | | – | – | 1 | µA |
| $I_{IH}$ | HIGH-level input current | | – | – | 1 | µA |
| $C_i$ | input capacitance | | – | – | 8 | pF |
| DATA INPUT AND OUTPUT: PIN SDA | | | | | | |
| $V_{IL}$ | LOW-level input voltage | | 0 | – | 0.8 | V |
| $V_{IH}$ | HIGH-level input voltage | 5 V tolerant | 2.0 | – | 5.5 | V |
| $V_{hys}$ | hysteresis voltage | | 0.3 | – | – | V |
| $V_{OL}$ | LOW-level output voltage | $I_{OL}$ = 4 mA | 0 | – | 0.4 | V |

**Notes**

1. 1024 × 768 at 60 Hz with input pattern Grill_33.
2. Pin connected to video source via a 6 dB/75 Ω attenuator.
3. Leakage current due to external voltage higher than internal $V_{DD}$.
4. Minimum value for $V_i$ = 4.5 V; maximum value for $V_i$ = 5.5 V.

## 12 TIMING CHARACTERISTICS

| SYMBOL | PARAMETER | CONDITIONS | MIN. | TYP. | MAX. | UNIT |
|---|---|---|---|---|---|---|
| **System clock input at pin CLK** | | | | | | |
| $T_{cy}$ | system clock cycle time | | 20 | – | 41.66 | ns |
| **Analog video interface;** see Fig.33 | | | | | | |
| $T_{cy}$ | analog video clock cycle time | | 9.1 | – | – | ns |
| $t_{su}$ | video data set-up time | | 4 | – | – | ns |
| $t_h$ | video data hold time | | 3 | – | – | ns |
| $t_{VSYNC}$ | vertical sync length | | $2/f_{VCLK}$ | – | – | ns |
| $t_{HSYNC}$ | horizontal sync length | | $2/f_{VCLK}$ | – | – | ns |
| **Parallel video interface;** see Fig.34 | | | | | | |
| $T_{cy}$ | parallel video clock cycle time | | 9.1 | – | – | ns |
| $t_{su}$ | video data set-up time | | 0 | – | – | ns |
| $t_h$ | video data hold time | | 5 | – | – | ns |
| **Panel interface;** see Fig.35 | | | | | | |
| $T_{cy}$ | panel clock cycle time | | 15.4 | – | 40 | ns |
| $t_{out1}$ | undelayed PCLK to output delay time; single pixel mode | $C_L$ = 15 pF | –2.5 | – | +0.2 | ns |
| $t_{out2}$ | undelayed PCLK to output delay time; double pixel mode | $C_L$ = 15 pF | –3.5 | – | –0.8 | ns |
| $t_{del}$ | output delay increment | | 200 | 500 | 800 | ps |

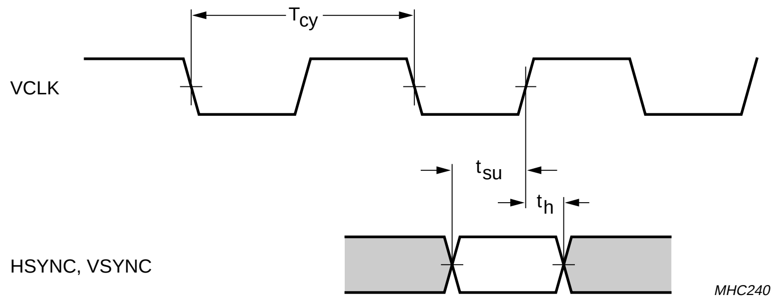



Fig.33 Analog video interface timing.

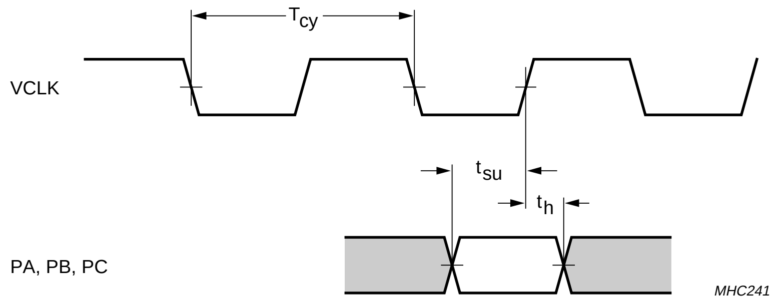



Fig.34 Parallel video interface timing.

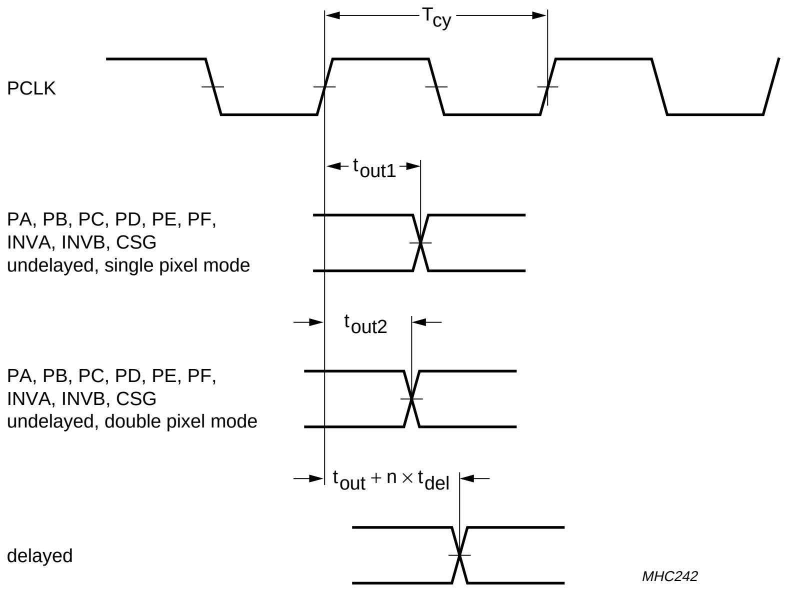



Fig.35 Panel interface timing.

## 13 APPLICATION INFORMATION

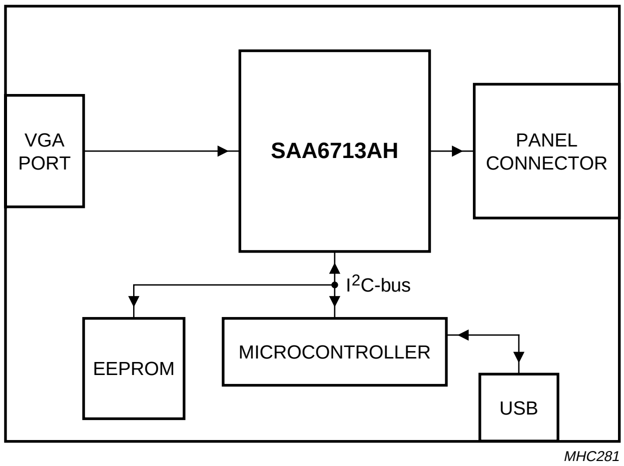
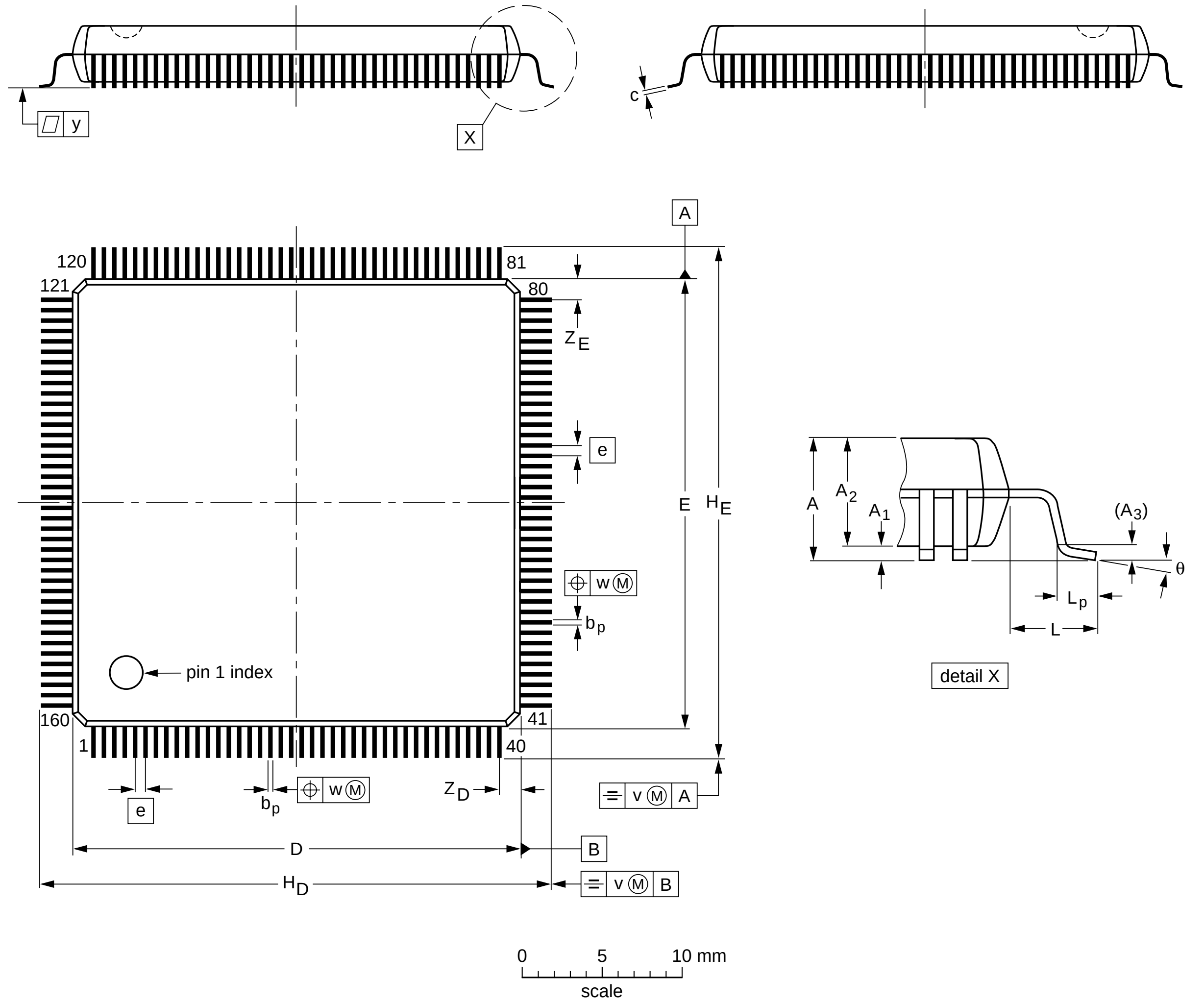



Fig.36 Application board block diagram.

## 14 PACKAGE OUTLINE

**QFP160: plastic quad flat package;**
**160 leads (lead length 1.6 mm); body 28 x 28 x 3.4 mm; high stand-off height** **SOT322-2**




**DIMENSIONS (mm are the original dimensions)**

| UNIT | A max. | $A_1$ | $A_2$ | $A_3$ | $b_p$ | c | D(1) | E(1) | e | $H_D$ | $H_E$ | L | $L_p$ | v | w | y | $Z_D$(1) | $Z_E$(1) | θ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mm | 4.07 | 0.50<br>0.25 | 3.6<br>3.2 | 0.25 | 0.38<br>0.22 | 0.23<br>0.13 | 28.1<br>27.9 | 28.1<br>27.9 | 0.65 | 31.45<br>30.95 | 31.45<br>30.95 | 1.6 | 1.03<br>0.73 | 0.3 | 0.13 | 0.1 | 1.5<br>1.1 | 1.5<br>1.1 | $7^o$<br>$0^o$ |

**Note**

1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

| OUTLINE VERSION | REFERENCES | | | | EUROPEAN PROJECTION | ISSUE DATE |
|---|---|---|---|---|---|---|
| | IEC | JEDEC | JEITA | | | |
| SOT322-2 | 135E12 | MS-022 | | | | ~~00-01-19~~<br>03-02-25 |

## 15 SOLDERING

### 15.1 Introduction to soldering surface mount packages

This text gives a very brief insight to a complex technology. A more in-depth account of soldering ICs can be found in our *"Data Handbook IC26; Integrated Circuit Packages"* (document order number 9398 652 90011).

There is no soldering method that is ideal for all surface mount IC packages. Wave soldering can still be used for certain surface mount ICs, but it is not suitable for fine pitch SMDs. In these situations reflow soldering is recommended.

### 15.2 Reflow soldering

Reflow soldering requires solder paste (a suspension of fine solder particles, flux and binding agent) to be applied to the printed-circuit board by screen printing, stencilling or pressure-syringe dispensing before package placement. Driven by legislation and environmental forces the worldwide use of lead-free solder pastes is increasing.

Several methods exist for reflowing; for example, convection or convection/infrared heating in a conveyor type oven. Throughput times (preheating, soldering and cooling) vary between 100 and 200 seconds depending on heating method.

Typical reflow peak temperatures range from 215 to 270 °C depending on solder paste material. The top-surface temperature of the packages should preferably be kept:

- below 220 °C (SnPb process) or below 245 °C (Pb-free process)
  - for all BGA and SSOP-T packages
  - for packages with a thickness ≥ 2.5 mm
  - for packages with a thickness < 2.5 mm and a volume ≥ 350 $mm^3$ so called thick/large packages.
- below 235 °C (SnPb process) or below 260 °C (Pb-free process) for packages with a thickness < 2.5 mm and a volume < 350 $mm^3$ so called small/thin packages.

Moisture sensitivity precautions, as indicated on packing, must be respected at all times.

### 15.3 Wave soldering

Conventional single wave soldering is not recommended for surface mount devices (SMDs) or printed-circuit boards with a high component density, as solder bridging and non-wetting can present major problems.

To overcome these problems the double-wave soldering method was specifically developed.

If wave soldering is used the following conditions must be observed for optimal results:

- Use a double-wave soldering method comprising a turbulent wave with high upward pressure followed by a smooth laminar wave.
- For packages with leads on two sides and a pitch (e):
  - larger than or equal to 1.27 mm, the footprint longitudinal axis is **preferred** to be parallel to the transport direction of the printed-circuit board;
  - smaller than 1.27 mm, the footprint longitudinal axis **must** be parallel to the transport direction of the printed-circuit board.

  The footprint must incorporate solder thieves at the downstream end.
- For packages with leads on four sides, the footprint must be placed at a 45° angle to the transport direction of the printed-circuit board. The footprint must incorporate solder thieves downstream and at the side corners.

During placement and before soldering, the package must be fixed with a droplet of adhesive. The adhesive can be applied by screen printing, pin transfer or syringe dispensing. The package can be soldered after the adhesive is cured.

Typical dwell time of the leads in the wave ranges from 3 to 4 seconds at 250 °C or 265 °C, depending on solder material applied, SnPb or Pb-free respectively.

A mildly-activated flux will eliminate the need for removal of corrosive residues in most applications.

### 15.4 Manual soldering

Fix the component by first soldering two diagonally-opposite end leads. Use a low voltage (24 V or less) soldering iron applied to the flat part of the lead. Contact time must be limited to 10 seconds at up to 300 °C.

When using a dedicated tool, all other leads can be soldered in one operation within 2 to 5 seconds between 270 and 320 °C.

### 15.5 Suitability of surface mount IC packages for wave and reflow soldering methods

| PACKAGE[1] | SOLDERING METHOD | |
|---|---|---|
| | WAVE | REFLOW[2] |
| BGA, LBGA, LFBGA, SQFP, SSOP-T[3], TFBGA, VFBGA | not suitable | suitable |
| DHVQFN, HBCC, HBGA, HLQFP, HSQFP, HSOP, HTQFP, HTSSOP, HVQFN, HVSON, SMS | not suitable[4] | suitable |
| PLCC[5], SO, SOJ | suitable | suitable |
| LQFP, QFP, TQFP | not recommended[5][6] | suitable |
| SSOP, TSSOP, VSO, VSSOP | not recommended[7] | suitable |
| PMFP[8] | not suitable | not suitable |

**Notes**

1. For more detailed information on the BGA packages refer to the *"(LF)BGA Application Note*" (AN01026); order a copy from your Philips Semiconductors sales office.
2. All surface mount (SMD) packages are moisture sensitive. Depending upon the moisture content, the maximum temperature (with respect to time) and body size of the package, there is a risk that internal or external package cracks may occur due to vaporization of the moisture in them (the so called popcorn effect). For details, refer to the Drypack information in the *"Data Handbook IC26; Integrated Circuit Packages; Section: Packing Methods"*.
3. These transparent plastic packages are extremely sensitive to reflow soldering conditions and must on no account be processed through more than one soldering cycle or subjected to infrared reflow soldering with peak temperature exceeding 217 °C ± 10 °C measured in the atmosphere of the reflow oven. The package body peak temperature must be kept as low as possible.
4. These packages are not suitable for wave soldering. On versions with the heatsink on the bottom side, the solder cannot penetrate between the printed-circuit board and the heatsink. On versions with the heatsink on the top side, the solder might be deposited on the heatsink surface.
5. If wave soldering is considered, then the package must be placed at a 45° angle to the solder wave direction. The package footprint must incorporate solder thieves downstream and at the side corners.
6. Wave soldering is suitable for LQFP, TQFP and QFP packages with a pitch (e) larger than 0.8 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.65 mm.
7. Wave soldering is suitable for SSOP, TSSOP, VSO and VSSOP packages with a pitch (e) equal to or larger than 0.65 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.5 mm.
8. Hot bar or manual soldering is suitable for PMFP packages.

## 16 DATA SHEET STATUS

| LEVEL | DATA SHEET STATUS[1] | PRODUCT STATUS[2][3] | DEFINITION |
|---|---|---|---|
| I | Objective data | Development | This data sheet contains data from the objective specification for product development. Philips Semiconductors reserves the right to change the specification in any manner without notice. |
| II | Preliminary data | Qualification | This data sheet contains data from the preliminary specification. Supplementary data will be published at a later date. Philips Semiconductors reserves the right to change the specification without notice, in order to improve the design and supply the best possible product. |
| III | Product data | Production | This data sheet contains data from the product specification. Philips Semiconductors reserves the right to make changes at any time in order to improve the design, manufacturing and supply. Relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). |

**Notes**

1. Please consult the most recently issued data sheet before initiating or completing a design.
2. The product status of the device(s) described in this data sheet may have changed since this data sheet was published. The latest information is available on the Internet at URL http://www.semiconductors.philips.com.
3. For data sheets describing multiple type numbers, the highest-level product status determines the data sheet status.

## 17 DEFINITIONS

**Short-form specification** — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

**Limiting values definition** — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 60134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

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

## 18 DISCLAIMERS

**Life support applications** — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

**Right to make changes** — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status ‘Production’), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

## 19 PURCHASE OF PHILIPS I$^2$C COMPONENTS

Purchase of Philips I$^2$C components conveys a license under the Philips' I$^2$C patent to use the components in the I$^2$C system provided the system conforms to the I$^2$C specification defined by Philips. This specification can be ordered using the code 9398 393 40011.

# *Philips Semiconductors – a worldwide company*

**Contact information**

For additional information please visit **http://www.semiconductors.philips.com**. Fax: **+31 40 27 24825**
For sales offices addresses send e-mail to: **sales.addresses@www.semiconductors.philips.com**.



Printed in The Netherlands R21/02/pp100 Date of release: 2004 Apr 05 Document order number: 9397 750 12102

*Let's make things better.*

**Philips Semiconductors**

**PHILIPS**