## 27.4.5 Interface Registers

### OCDS Control and Status Register

The OCDS Control and Status (OCS) register is cleared by Debug Reset.

The OCS register can only be written when the OCDS is enabled.

If OCDS is being disabled, the OCS register value will not change.When OCDS is disabled the OCS suspend control is ineffective.

**OCS**

**OCDS Control and Status Register (00E8$_H$) Debug Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | SUSSTA | SUS_P | SUS | | | | 0 | | | | | | | |
| r | | rh | w | rw | | | | r | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | 0 | |
| r | | | | | | | | | | | | | | rw | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **SUS** | 27:24 | rw | **OCDS Suspend Control**<br>Controls the sensitivity of STMx to the suspend signal coming from CPUx (CPUxSUSOUT)<br>$0_H$ Will not suspend<br>$1_H$ Reserved, do not use this combination<br>$2_H$ 64-bit counter will be stopped<br>**others**, Reserved, do not use this combination |
| **SUS_P** | 28 | w | **SUS Write Protection**<br>SUS is only written when SUS_P is 1, otherwise unchanged. Read as 0. |
| **SUSSTA** | 29 | rh | **Suspend State**<br>$0_B$ Module is not (yet) suspended<br>$1_B$ Module is suspended |
| **0** | 2:0 | rw | **Reserved**<br>Read as 0; must be written with 0. |
| **0** | 23:3, 31:30 | r | **Reserved**<br>Read as 0; must be written with 0. |

**Table 2 Access Mode Restrictions of OCS sorted by descending priority**

| Mode Name | Access Mode | | Description |
|---|---|---|---|
| write 1 to **SUS_P** | rw | SUS | |
| (default) | r | SUS | |

### Access Enable Register 0

The Access Enable Register 0 controls write access for transactions with the on chip bus master TAG ID 000000B to 011111B (see On Chip Bus chapter for the products TAG ID <-> master peripheral mapping). The BPI_FPI is prepared for a 6-bit TAG ID. The registers ACCEN0 / ACCEN1 are providing one enable bit for each possible 6-bit TAG ID encoding.

Mapping of TAG IDs to ACCEN0.ENx: EN0 -> TAG ID 000000B, EN1 -> TAG ID 000001B , …,EN31 -> TAG ID 011111B.

**ACCEN0**

**Access Enable Register 0** **($00FC_H$)** **Application Reset Value: $FFFF\ FFFF_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **EN31** | **EN30** | **EN29** | **EN28** | **EN27** | **EN26** | **EN25** | **EN24** | **EN23** | **EN22** | **EN21** | **EN20** | **EN19** | **EN18** | **EN17** | **EN16** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **EN15** | **EN14** | **EN13** | **EN12** | **EN11** | **EN10** | **EN9** | **EN8** | **EN7** | **EN6** | **EN5** | **EN4** | **EN3** | **EN2** | **EN1** | **EN0** |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

| Field | Bits | Type | Description |
|---|---|---|---|
| **ENn (n=0-31)** | n | rw | **Access Enable for Master TAG ID n**<br>This bit enables write access to the module kernel addresses for transactions with the Master TAG ID n<br>$0_B$ Write access will not be executed<br>$1_B$ Write access will be executed |

### Access Enable Register 1

The Access Enable Register 1 controls write access for transactions with the on chip bus master TAG ID 100000B to 111111B (see On Chip Bus chapter for the products TAG ID master peripheral mapping).

Mapping of TAG IDs to ACCEN1.ENx: EN0 -> TAG ID 100000B, EN1 -> TAG ID 100001B, …, EN31 -> TAG ID 111111B.

**ACCEN1**

**Access Enable Register 1** **($00F8_H$)** **Application Reset Value: $0000\ 0000_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | | | | | |
| r | | | | | | | | | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | | | | | | | | |
| r | | | | | | | | | | | | | | | |

| Field | Bits | Type | Description |
|---|---|---|---|
| **0** | 31:0 | r | **Reserved**<br>Read as 0; should be written with 0. |

### Kernel Reset Register 0

The Kernel Reset Register 0 is used to reset the related module kernel. To reset a module kernel it is necessary to set the RST bits by writing with ´1´ in both Kernel Reset Registers related to the module kernel that should be reset (kernel 0 or kernel 1). In order support modules with two kernel the BPI_FPI provides two set of kernel reset registers. The RST bit will be re-set by the BPI with the end of the BPI kernel reset sequence.

Kernel Reset Register 0 includes a kernel reset status bit that is set to ´1´ by the BPI_FPI in the same clock cycle the RST bit is re-set by the BPI_FPI. This bit can be used to detect that a kernel reset was processed. The bit can be re-set to ´0´ by writing to it with ´1´.

**KRST0**

**Kernel Reset Register 0** **(00F4$_H$)** **Application Reset Value: 0000 0000$_H$**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| r | | | | | | | | | | | | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | RSTSTAT | RST |
| r | | | | | | | | | | | | | | rh | rwh |

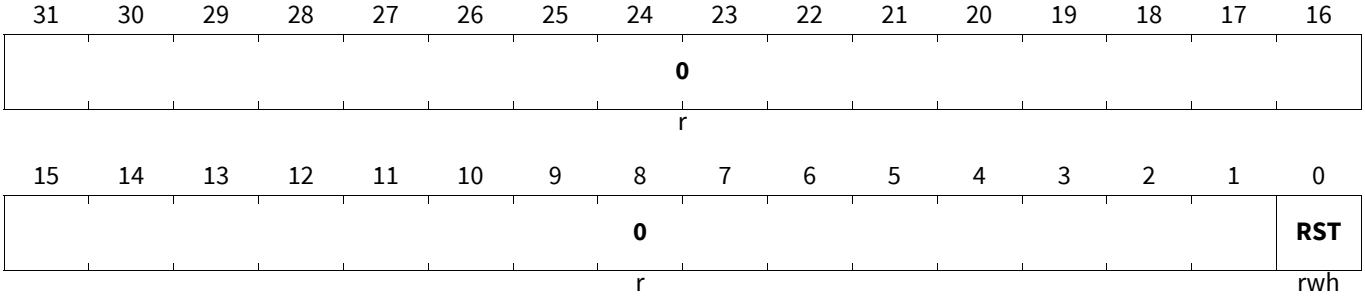
| Field | Bits | Type | Description |
|---|---|---|---|
| **RST** | 0 | rwh | **Kernel Reset**<br>This reset bit can be used to request for a kernel reset. The kernel reset will be executed if the reset bits of both kernel registers are set.<br>The RST bit will be cleared (re-set to ´0´) by the BPI_FPI after the kernel reset was executed.<br>$0_B$ No kernel reset was requested<br>$1_B$ A kernel reset was requested |
| **RSTSTAT** | 1 | rh | **Kernel Reset Status**<br>This bit indicates wether a kernel reset was executed or not. This bit is set by the BPI_FPI after the execution of a kernel reset in the same clock cycle both reset bits.<br>This bit can be cleared by writing with ´1´ to the CLR bit in the related KRSTCLR register.<br>$0_B$ No kernel reset was executed<br>$1_B$ Kernel reset was executed |
| **0** | 31:2 | r | **Reserved**<br>Read as 0; should be written with 0. |

### Kernel Reset Register 1

The Kernel Reset Register 1 is used to reset the STM kernel. STM kernel registers related to the Debug Reset (Class 1) are not influenced. To reset the STM kernel it is necessary to set the RST bits by writing with ´1´ in both Kernel Reset registers. The RST bit will be cleared with the end of the BPI kernel reset sequence.

### KRST1

**Kernel Reset Register 1** **(00F0$_H$)** **Application Reset Value: 0000 0000$_H$**



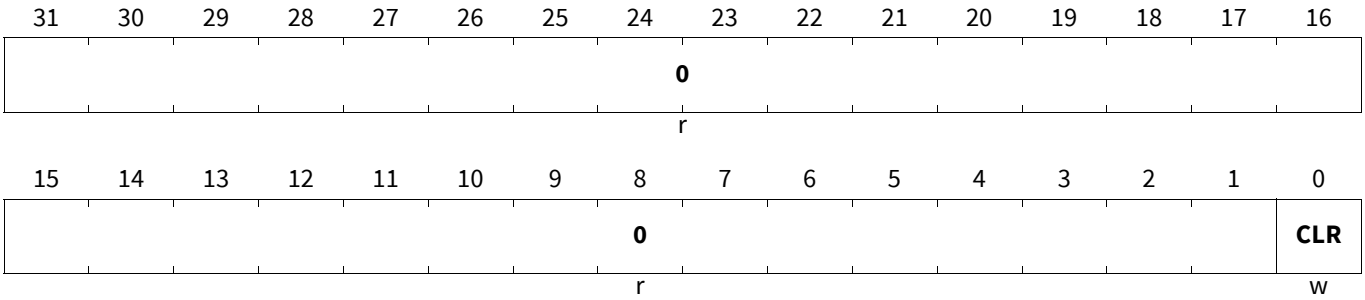

| Field | Bits | Type | Description |
|---|---|---|---|
| **RST** | 0 | rwh | **Kernel Reset**<br>This reset bit can be used to request for a kernel reset. The kernel reset will be executed if the reset bits of both kernel reset registers is set.<br>The RST bit will be cleared (re-set to ´0´) after the kernel reset was executed.<br>$0_B$ No kernel reset was requested<br>$1_B$ A kernel reset was requested |
| **0** | 31:1 | r | **Reserved**<br>Read as 0; should be written with 0. |

### Kernel Reset Status Clear Register

The Kernel Reset Register Clear register is used to clear the Kernel Reset Status bit (<>_KRST0.RSTSTAT).

### KRSTCLR

**Kernel Reset Status Clear Register** **(00EC$_H$)** **Application Reset Value: 0000 0000$_H$**




| Field | Bits | Type | Description |
|---|---|---|---|
| **CLR** | 0 | w | **Kernel Reset Status Clear**<br>Read always as 0.<br>$0_B$ No action<br>$1_B$ Clear Kernel Reset Status KRST0.RSTSTAT |
| **0** | 31:1 | r | **Reserved**<br>Read as 0; should be written with 0. |

## 27.5 IO Interfaces

The table below list of interfaces of the STM to other modules.

**Table 3 List of STM Interface Signals**

| Interface Signals | I/O | Description |
|---|---|---|
| sx_fpi | | **FPI slave interface** |
| sx_irq_stm | | **STM Interrupt Socket** |
| SR0_INT | out | **System Timer Service Request 0** |
| SR1_INT | out | **System Timer Service Request 1** |

## 27.6 Revision History

**Table 4 Revision History**

| Reference | Change to Previous Version | Comment |
|---|---|---|
| **V9.2.3** | | |
| **Page 23** | Previous versions removed from revision history. | |
| **V9.2.4** | | |
| **Page 19** | Description of bit field “SUS” of register “OCDS Control and Status Register” updated. | |

# 28 Generic Timer Module (GTM)

GTM general Architecture

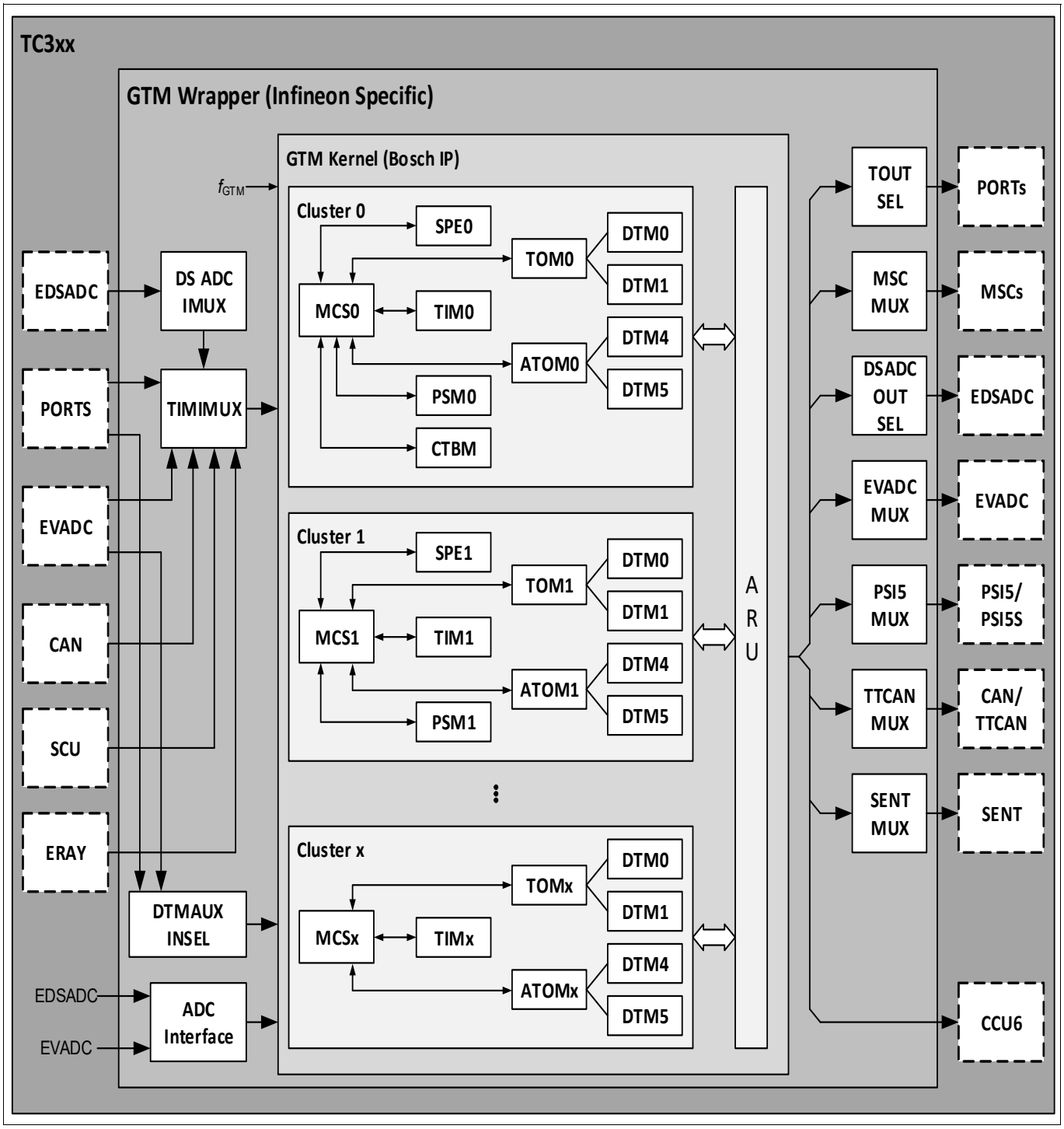



**Figure 4 GTM Architecture block diagram**

*Note: The GTM system runs with the GTM global clock $f_{GTM}$ (referred in this chapter as SYS_CLK)*