Freescale Semiconductor
Application Note

Document Number: AN4048
Rev. 0, 31/August/2010

Using the Development Trigger Semaphore Module

As Implemented on the MPC564xA

by: Randy Dees
Applications Engineering
Austin, Texas
USA

1 Introduction

As automotive powertrain and other engine control applications (such as those used in hybrid vehicles) become more complex, the need for more advanced tools and development processes increases. The MPC564xA[1] implements new features specifically for use in advanced automotive development processes. One of the new features is a module specifically intended to enable the user software to signal to an external tool that data is available. This application note describes the operation of the Development Trigger Semaphore (DTS) module and its primary intended use. There are other ways that this module may be used, but only the primary intended use is discussed in this application note.

The calibration process of a new engine requires real-time access to calibration tables and the ability to update these tables in real time[2]. Another important aspect of the development process requires real-time access to data measurements, for example, sensor data or data calculations. The DTS module specifically targets real-time access to measurements available in the CPU that affect the operation of the engine. Generally, these data measurements are time-aligned or angle-aligned to the engine. The DTS module allows up to thirty-two triggered sets of measurements to be signaled to an external tool. When the data

1. The features described in this application note are not available on revision 1 of the MPC564xA, but are available on revision 2.
2. See Freescale document AN4030, "Real-Time MMU Manipulation on the e200*zx* Power Architecture™ Core," for information on a method for real-time switching of calibration tables.

Contents

is available, a trigger signals the external tool. The tool can then retrieve the data. Since the tool is notified that new data is available via an external signal, the tool is not required to continuously poll internal device registers or memory locations.

It is the user's responsibility to ensure that the tool has time to retrieve the data prior to that particular trigger being set a second time. It is permissible to have multiple triggers active at the same time or for a second trigger to be set before a previous trigger has been serviced, as long as it is not the same trigger (unless it is acceptable to the tool to not receive every data set).

2 Overview

The Development Trigger Semaphore (DTS) module provides a 32-bit register of semaphores and an identification register. The identification register can be used as part of a triggered data acquisition protocol between an embedded controller and an external tool, either for calibration or for rapid prototyping.[3]

The DTS_SEMAPHORE register, along with the DTS Trigger Output ($\overline{\text{DTO}}$), provides a mechanism to indicate to the tool that the calibration variables (or sets of measurements), from one to thirty-two, have been updated with new values and are available for access. Data coherency is maintained by limiting the setting of bits in the DTS_SEMAPHORE to only the CPU and DMA. The CPU and DMA cannot clear any bit in this register. Only a tool access via Nexus read/write access through the JTAG port can clear bits in this register. Access permissions to the DTS_SEMAPHORE register are based on the crossbar (XBAR) master identification number for the port.

The DTS_STARTUP register provides a mechanism for the tool to notify software running on the CPU that a tool is connected and can provide information about either the type of tool or the options that can be used by the software.

The DTS_ENABLE register enables the DTS feature.

2.1 Development Trigger Semaphore module block diagram

The Development Trigger Semaphore (DTS) module consists of three registers and a small amount of combinational logic to generate a trigger output signal (DTS Trigger Out, or $\overline{\text{DTO}}$).

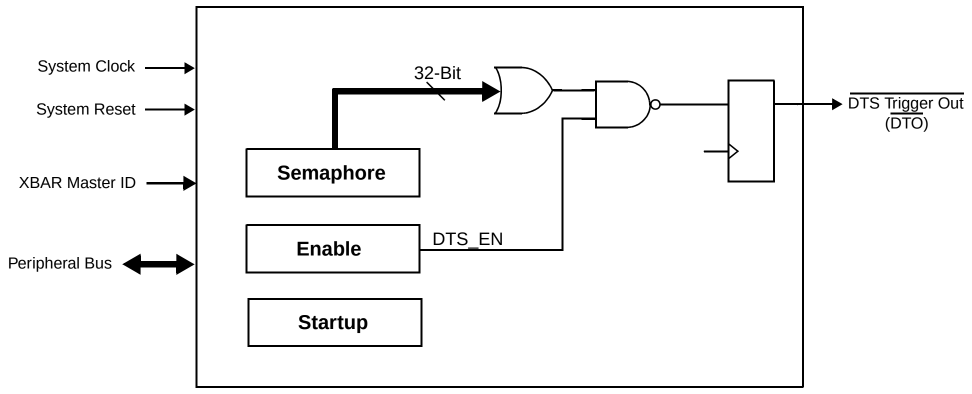

Figure 1. DTS block diagram

The DTS Trigger Out ($\overline{\text{DTO}}$) is asserted when any bit in the DTS_Semaphore register is set. The $\overline{\text{DTO}}$ signal connects to one of the $\overline{\text{EVTO}}$ inputs of the Nexus port controller (NPC). The other $\overline{\text{EVTO}}$ inputs to the NPC are connected to the other Nexus modules in the device.

NOTE

When the DTS module is enabled (DTS_ENABLE[DTS_EN]=0b1), the Nexus $\overline{\text{EVTO}}$ function of the $\overline{\text{EVTO}}$ pin is disabled and $\overline{\text{EVTO}}$ becomes the $\overline{\text{DTO}}$. Unlike the $\overline{\text{EVTO}}$

3. This triggered data acquisition uses the Nexus read/write access via the JTAG interface of the Nexus debug port and is different than the data acquisition protocol that uses the Nexus auxiliary port and is defined in either the IEEE-ISTO 5001-2003 or IEEE-ISTO 5001-2010 Nexus standard. The IEEE-ISTO 5001 Nexus data acquisition is also supported on the e200z4 and e200z7 cores.

function that only asserts for one clock, the $\overline{\text{DTO}}$ function remains asserted until the tool reads the DTS_SEMAPHORE register.

The timing of the $\overline{\text{DTO}}$ signal and its effect on the $\overline{\text{EVTO}}$ output are shown in the following figure.

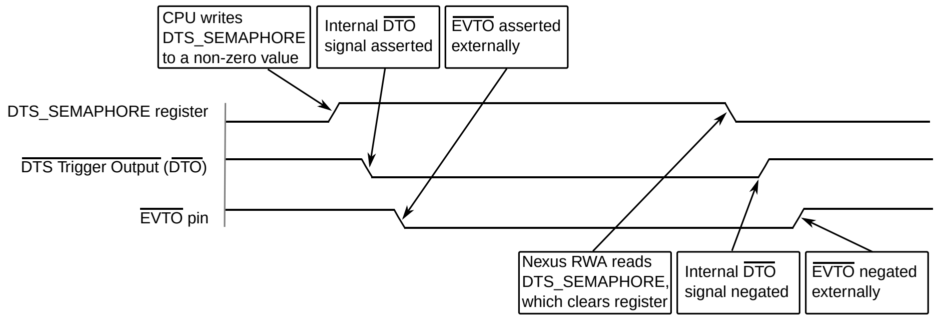
CPU writes DTS_SEMAPHORE to a non-zero value
Internal DTO signal asserted
EVTO asserted externally
DTS_SEMAPHORE register
DTS Trigger Output (DTO)
EVTO pin
Nexus RWA reads DTS_SEMAPHORE, which clears register
Internal DTO signal negated
EVTO negated externally

Figure 2. $\overline{\text{DTS trigger-out}}$ timing

2.2 DTS device connections

The DTS module connects to the peripheral bridge (PBRIDGE) for access to the registers. The PBRIDGE is then connected to a slave port of the crossbar bus interface (XBAR). Connected to the XBAR master ports are the core (one or more — one e200z4 Power Architecture core on the MPC564xA, each with a master access port for the separate instruction and load/store buses), the DMA (one or more eDMA modules depending on the device), the FlexRay modules, and an external bus interface[4].

Some of the registers have limited access, as shown in the DTS register access section. Access is based on the master ID of the module through the XBAR.

NOTE

Nexus read/write accesses use the load/store bus of the core to perform accesses, but Nexus accesses have a different master ID than normal core load/stores.

Access to the DTS_SEMAPHORE register is limited to either the e200*zx* cores and the eDMA module to set bits only (based on the XBAR master ID).[5] Only an access via a Nexus read/write access from an external tool through the Nexus/JTAG port of the device can clear bits in the DTS_SEMAPHORE register (cleared automatically by the read through the Nexus/JTAG port)[6]. Similarly, the DTS_ENABLE and DTS_STARTUP registers can only be written via a Nexus read/write access.

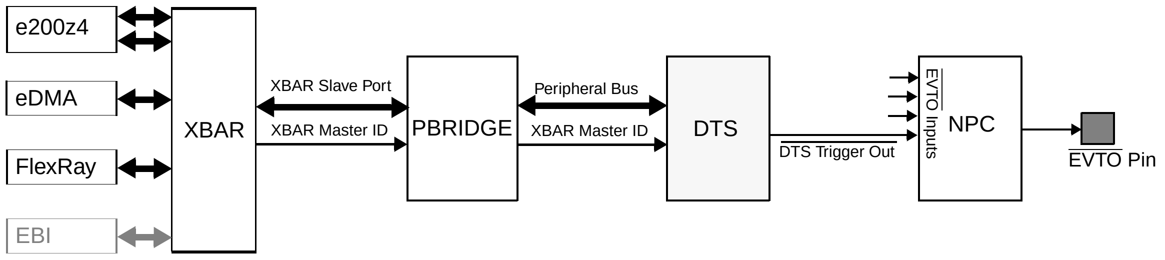

Figure 3. MPC564xA DTS internal device connections

2.2.1 DTS register access

A summary of accesses to all DTS registers by bus masters is provided in DTS register access. Note that only proper 32-bit accesses are valid. The effects of write accesses that are not 32-bit are not defined.

4. The external bus interface master port is used for internal testing of the device and is not accessible to the user.
5. On multi-core devices or devices with multiple eDMA modules, any core and any eDMA module can set bits in the DTS_Semaphore register.
6. On multi-core devices, the Nexus read/write access can use any core for the access.

Table 1. DTS register access

Register	32-bit read				32-bit write			
	RWA[1]	CPU Core	DMA[2]	Other[3]	RWA[1]	CPU Core	DMA[2]	Other[3]
DTS_ENABLE	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_STARTUP	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_SEMAPHORE	Data and Clear[4]	Data	Data	Data	No effect	Bit OR	Bit OR	No effect

1. Nexus read/write access via an external tool.
2. Direct Memory Access module.
3. Other master ports on the XBAR are connected to FlexRay and the external bus interface (not accessible).
4. A read of the DTS_SEMAPHORE register by any Nexus read/write access module is destructive and clears all bits in the register.

2.2.2 MPC564xA crossbar ports and port master identifiers

All internal accesses in the MPC564xA are through a crossbar (XBAR) switch matrix. The XBAR allows multiple concurrent accesses between masters connected on the XBAR and slaves also connected to the XBAR. In the MPC564xA, the master ports are the e200z4 core (connected to two XBAR ports), the eDMA module, the FlexRay, and a master port for an external bus master access. The table below shows the different XBAR master IDs for each of the XBAR ports. These master IDs are used to control access to the DTS registers.

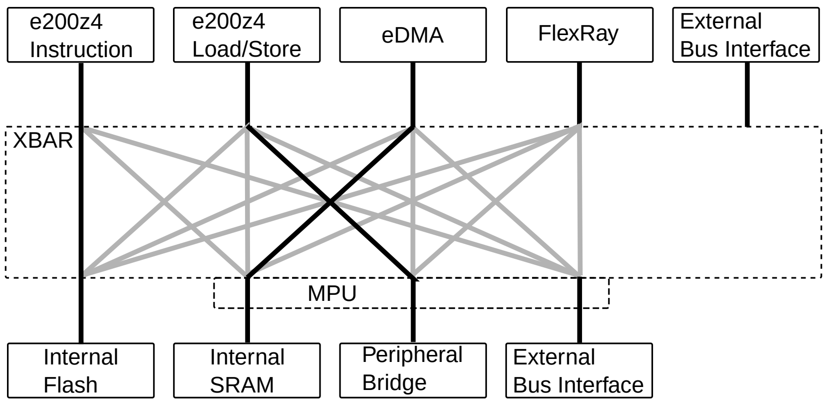

Figure 4. MPC564xA XBAR connections block diagram

Figure 4 shows all of the master and slave ports of the XBAR bus, as well as all of the possible paths through the XBAR. Up to four paths can be in use at the same time. Three example concurrent paths are highlighted in the figure showing the following concurrent transactions:

- e200z4 core instruction fetch from the internal flash
- e200z4 core load or store access from/to a peripheral on the peripheral bridge
- eDMA access to or from the internal SRAM

NOTE

The XBAR master ID should not be confused with the master port number of the XBAR. These are separate. The master IDs are shown in this document since they are used by the access logic of the DTS module, but this is transparent to the user. Some peripherals allow the user to control access to the peripheral based on the XBAR master ID and therefore are not transparent to the user.

Table 2. MPC564xA crossbar master IDs and master port numbers

Module type	Description	XBAR master port	Master ID
CPU Core	e200z4 core (instruction bus)	0	0
	e200z4 core (load/store bus)	1	0
	Nexus RWA[1]	1	8
DMA	Enhanced direct memory access	4	3
FlexRay	FlexRay master access	6	6
EBI[2]	External bus interface	7	7

1. The Nexus port uses the core load/store bus, but has its own master ID when used for access.
2. The external bus master port is not accessible, but is implemented and used for internal factory test.

NOTE

Tools must access the DTS registers (ENABLE, STARTUP, and SEMAPHORE) through the Nexus read/write access mechanism of either core. JTAG accesses through the core will appear as if the access is via the core, and therefore will not have the same level of access as a Nexus read/write access. For additional information about JTAG core accesses versus Nexus read/write accesses, see Appendix B of Freescale document AN2614, "MPC553x, MPC555x, and MPC556x Family Nexus Interface Connector."

3 Development Trigger Semaphore module memory map

The table below shows the the memory map of the Development Trigger Semaphore module registers. Only three 32-bit registers are implemented. The rest of the memory map (0xC3F9_C00C thorugh 0xC3F9_FFFF) is reserved.

Table 3. Development Trigger Semaphore Module Memory Map

Address	Register Name	Register Description	Size (bits)	Access
0xC3F9_C000	DTS_ENABLE	DTS Output Enable register	32	Restricted R/W[1]
0xC3F9_C004	DTS_STARTUP	DTS Startup register	32	Restricted R/W[1]
0xC3F9_C008	DTS_SEMA-PHORE	DTS Semaphore register	32	Restricted R/W[1]
0xC3F9_C00C – 0xC3F9_FFFF	—	Reserved	—	—

1. Only certain types of accesses are allowed. See DTS register access.

3.1 DTS Enable register (DTS_ENABLE)

The DTS Enable register (DTS_ENABLE) controls the DTS Trigger Output ($\overline{\text{DTO}}$) and whether $\overline{\text{DTO}}$ is active on the $\overline{\text{EVTO}}$ output of the device. Table 4 shows the format of the DTS_ENABLE register. Only one bit is implemented. Access to the DTS_SEMAPHORE and DTS_STARTUP registers are unaffected by the state of this bit.

Table 4. DTS enable (DTS_ENABLE)

Address 0xC3F9_C000		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DTS_EN
	W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. DTS_ENABLE field descriptions

Field	Description
DTS_EN	DTS Enable. Controls whether the $\overline{\text{DTO}}$ signal is routed to the $\overline{\text{EVTO}}$. 0: DTS output disabled. 1: DTS output enabled. Any bit set in the DTS_SEMAPHORE register will assert the $\overline{\text{DTO}}$ signal.

The DTS Enable bit is cleared by a device reset (either the assertion of the external $\overline{\text{RESET}}$ or an internally generated reset). A JTAG reset does not change the state of this register. The DTS_ENABLE register is a 32-bit register that can be read by the e200z*x* core, but can only be written by a Nexus read/write access (RWA).

3.2 DTS Startup register (DTS_STARTUP)

The DTS Startup register (DTS_STARTUP) is used for tool detection and startup information exchange between the tool and software running on the MCU. Table 6 shows the format of the DTS_STARTUP register.

- A device reset (either from the $\overline{\text{RESET}}$ pin or an internally generated reset) clears all bits in the register.
- A JTAG reset does not change the contents of the register.
- A core, DMA, or Nexus RWA 32-bit read access returns the register contents.
- Only a Nexus RWA 32-bit write access can update the contents of this register.

Table 6. DTS Start Up register (DTS_STARTUP)

Address 0xC3F9_C004		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R	AD31	AD30	AD29	AD28	AD27	AD26	AD25	AD24	AD23	AD22	AD21	AD20	AD19	AD18	AD17	AD16
	W																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
	W																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7. DTS_STARTUP register field descriptions

Field	Description
AD[31:0]	Application Dependent register bits — the bits have no defined meaning to the MCU and are used to exchange information between an external tool and the software running on the target CPU at startup time.

3.3 DTS Semaphore register (DTS_SEMAPHORE)

The following table shows the format of the DTS Semaphore (DTS_SEMAPHORE) register. This register stores the currently active triggers for the tool. When any of the trigger bits are set and the DTS_ENABLE[DTS_EN] is set, the DTS module will assert the DTS_Trigger_Out, which drives the $\overline{\text{EVTO}}$ pin.

- All register bits are set to one by a device reset.
- A JTAG reset does not change the state of this register.
- The register can be accessed, with restrictions, by any core, DMA, or any Nexus RWA.
- For the core or DMA, only 32-bit write or read accesses are valid.
- A core or DMA valid read access returns the current value of the register and leaves the register unchanged.
- The effect of a core or DMA valid write access and Nexus RWA read access is shown in the DTS register access section of this application note. The effect of other types of access and access by other bus masters is also discussed in that section.

Table 8. DTS Semaphore register (DTS_SEMAPHORE)

Address 0xC3F9_C008		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	R	ST31	ST30	ST29	ST28	ST27	ST26	ST25	ST24	ST23	ST22	ST21	ST20	ST19	ST18	ST17	ST16
	W																
	Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST6	ST5	ST4	ST3	ST2	ST1	ST0
	W																
	Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 9. DTS_SEMAPHORE register field descriptions

Field	Description
ST[31:0]	When a core or eDMA writes a logical one to a bit, the bit is set. A write of zero by the core or DMA does not change the state of the bit. A read of this register returns the current value and does not affect the state of the register. A Nexus read via a tool returns the current value and automatically clears all bits in the register. 0: Trigger is not active for this channel (No flag). 1: Trigger is set for this channel, data is available if software is using the DTS_SEMAPHORE register for triggered data acquisition (flag is set).

4 Using the DTS module

The Development Trigger Semaphore module is intended to allow software that is running in the MCU to notify external tools that specific data is available for those tools. The DTS_SEMAPHORE register allows for up to thirty-two different data sets to be identified.

The use of the DTS module can be divided into three phases: initialization, synchronization, and acquisition.

4.1 Trigger initialization and synchronization

The startup and synchronization sequence can be as simple or as complicated as the need requires. Figure 5 shows an example of a typical startup sequence with a synchronization hand-shake and one "raster" of data becoming available (raster trigger 8 — 0x0000_0100).

- The DTS_STARTUP register is cleared by a power-on reset or any CPU reset.
- The tool writes a value, other than 0x0, to the DTS_STARTUP register.
- The CPU (user application software) then reads the value of the DTS_STARTUP register. Based on this value, different initialization settings can be selected. The bits can be used for any application-specific definitions.
- Since the DTS_SEMAPHORE register is cleared when the tool reads the current value, the tool should perform all necessary initialization before reading this register. The application software can then check that the DTS_SEMAPHORE register was cleared by the tool. This is done to determine whether it is safe to start using it for its intended raster trigger semaphore function.
- In addition, an optional handshake from the CPU could be used to inform the tool that the user software has detected that the tool is attached and that the CPU has performed the proper initialization for the tool by writing a predefined value to the DTS Semaphore register. (The example shown in the figure above uses 0xAAAA_AAAA — all A's was used since it is unrealistic that sixteen channels could be enabled very quickly after a startup that follows a reset.)

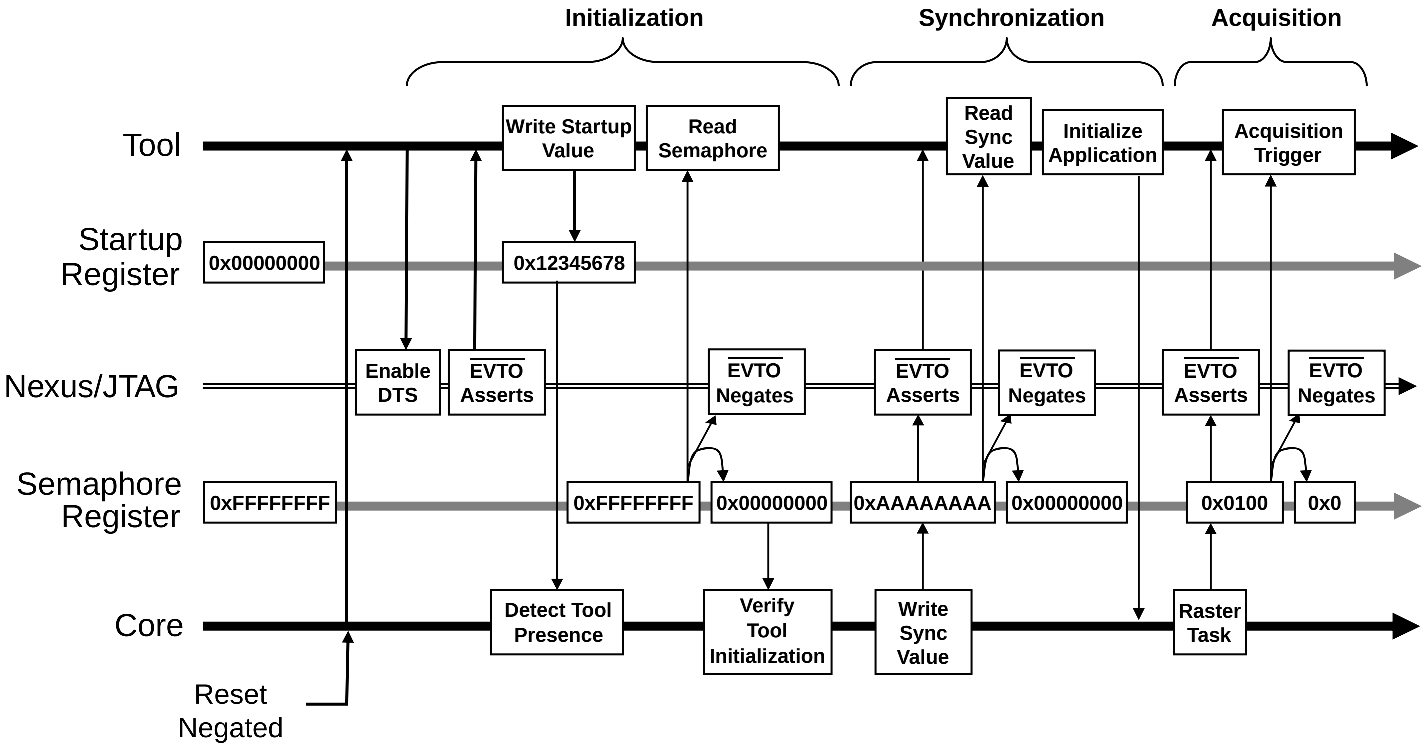

Figure 5. DTS usage overview

The acquisition phase is described in the next section.

4.2 Trigger-based acquisition

The raster trigger semaphore function is normally used to enable up to thirty-two signals from the application software to be detected by an external tool. A raster trigger informs the tool that information (for example, calibration data measurements) is available from the user application. It is assumed that the tool knows the meaning and requirements of each of the defined trigger rasters. An example use is as follows:

1. An application generates data measurements at some defined time. The values are then stored in a predefined memory address for that trigger. The bit for the particular raster trigger is set in the DTS_SEMAPHORE register.
2. The bit set in the DTS Semaphore register causes the $\overline{\text{EVTO}}$ ($\overline{\text{DTO}}$) pin to be asserted (low).
3. When the tool detects $\overline{\text{EVTO}}$ to be asserted, the tool reads the DTS_SEMAPHORE register (which clears all bits in the register). The tool then processes each of the raster triggers that were set and transfers all data measurements for each of the triggered "channels."
4. One example of handling the trigger raster is that the tool reads data at an address that has been defined in memory for that raster trigger. The data could be any number of variables (and any size) as defined by the application software. Possible data measurements could include analog-to-digital converter values from sensors, such as oxygen level, or even digital values, such as wheel speed calculation or spark actual dwell time.
5. If multiple bits have been set, then the tool should process the other rasters that have been set in the DTS_SEMAPHORE register.

The CPU application software should perform a "blind" write to the DTS_SEMAPHORE register of the bits (set the bits) that represent the trigger rasters that are available.

```
/***************************************************************************/
/* Example setting DTS_Semaphore register.                                 */
/***************************************************************************/
DTS.SEMAPHORE.R = 0x00000100; /* set semaphore trigger 8 */
```

NOTE

A read/modify/write operation should not be used since it is possible that the tool could read (which clears the register) while the modify operation is being performed. By writing ones only to the desired bits, this preserves any clear that occurs, but sets the desired bits (since the CPU can never clear any bit in the DTS_SEMAPHORE register).

Appendix A Development Trigger Semaphore header file

The code listing below shows the header file information for the Development Trigger Semaphore module.

```
/***************************************************************************/
/* FILE NAME: MPC564xA_DTS.h COPYRIGHT (c) Freescale 2010 */

/* VERSION:  0.0                                    All Rights Reserved     */
/*                                                                         */
/* DESCRIPTION:                                                            */
/* This file contains all of the register and bit field definitions for   */
/* MPC564xA DTS module.                                                */
/*=========================================================================*/
/* UPDATE HISTORY                                                          */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                   */
/* ---   -----------  ---------    ---------------------                   */
/* 0.0   R. Dees      03/JAN/10    Initial version.                        */
/***************************************************************************/

/***************************************************************************/
/*                     MODULE : Development Trigger Semaphore Module       */
/***************************************************************************/

    struct DTS_tag {
        union {
            vuint32_t R;
            struct {
                vuint32_t RSRVD:31;        /*   */
                vuint32_t DTS_EN:1; /* Enable for the DTS Module */
            } B;
        } ENABLE;                 /* DTS_ENABLE @BaseAddress */

        union
        {
        vuint32_t R;
            struct {
                vuint32_t AD31:1;       /* Startup register MSB */
                vuint32_t AD30:1;       /* */
                vuint32_t AD29:1;        /* */
                vuint32_t AD28:1;        /* */
                vuint32_t AD27:1;        /* */
                vuint32_t AD26:1;        /* */
                vuint32_t AD25:1;        /* */
                vuint32_t AD24:1;        /* */
                vuint32_t AD23:1;        /* */
                vuint32_t AD22:1;        /* */
                vuint32_t AD21:1;        /* */
                vuint32_t AD20:1;        /* */
                vuint32_t AD19:1;        /* */
                vuint32_t AD18:1;        /* */
                vuint32_t AD17:1;        /* */
                vuint32_t AD16:1;        /* */
                vuint32_t AD15:1;        /* */
                vuint32_t AD14:1;        /* */
                vuint32_t AD13:1;        /* */
                vuint32_t AD12:1;        /* */
                vuint32_t AD11:1;        /* */
                vuint32_t AD10:1;        /* */
                vuint32_t AD9:1;        /* */
                vuint32_t AD8:1;        /* */
                vuint32_t AD7:1;        /* */
                vuint32_t AD6:1;        /* */
                vuint32_t AD5:1;        /* */
                vuint32_t AD4:1;        /* */
                vuint32_t AD3:1;        /* */
                vuint32_t AD2:1;        /* */
                vuint32_t AD1:1;        /* */
                vuint32_t AD0:1;        /* Startup Register LSB */
            } B;
        } STARTUP;                /* DTS_STARTUP @BaseAddress + 0x4*/

        union
        {
        vuint32_t R;
            struct {
                vuint32_t ST31:1;        /* Semaphore register MSB */
                vuint32_t ST30:1;        /* */
                vuint32_t ST29:1;        /* */
                vuint32_t ST28:1;        /* */
                vuint32_t ST27:1;        /* */
                vuint32_t ST26:1;        /* */
                vuint32_t ST25:1;        /* */
                vuint32_t ST24:1;        /* */
                vuint32_t ST23:1;        /* */
                vuint32_t ST22:1;        /* */
                vuint32_t ST21:1;        /* */
                vuint32_t ST20:1;        /* */
                vuint32_t ST19:1;        /* */
                vuint32_t ST18:1;        /* */
                vuint32_t ST17:1;        /* */
                vuint32_t ST16:1;        /* */
                vuint32_t ST15:1;        /* */
                vuint32_t ST14:1;        /* */
                vuint32_t ST13:1;        /* */
                vuint32_t ST12:1;        /* */
                vuint32_t ST11:1;        /* */
                vuint32_t ST10:1;        /* */
                vuint32_t ST9:1;        /* */
                vuint32_t ST8:1;        /* */
                vuint32_t ST7:1;        /* */
                vuint32_t ST6:1;        /* */
                vuint32_t ST5:1;        /* */
                vuint32_t ST4:1;        /* */
                vuint32_t ST3:1;        /* */
                vuint32_t ST2:1;        /* */
                vuint32_t ST1:1;        /* */
                vuint32_t ST0:1;        /* Semaphore Register LSB */
            } B;
        } SEMAPHORE;                /* DTS_SEMAPHORE @BaseAddress + 0x8 */
    };

/* Define instances of modules */
#define DTS     (*( volatile struct DTS_tag *)     0xC3F9C000)
```

Appendix B DTS system level block diagram

The figure below shows a combination of the DTS block diagram and the system device connections.

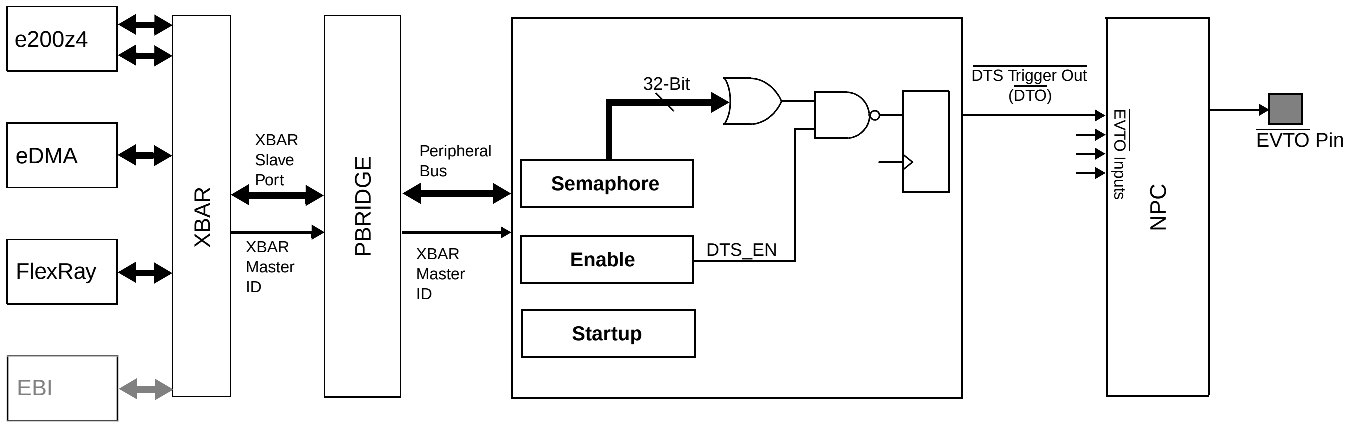

Figure B-1. MPC564xA system block diagram

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4048
Rev. 0, 31/August/2010