Freescale Semiconductor
Application Note

Document Number: AN4380
Rev. 0, 10/2011

SATA Temperature Sensor (i.MX53)

by *Multimedia Applications Division*
Freescale Semiconductor, Inc.
Austin, TX

1 Introduction

SATA module provides the feature to measure the die temperature during its operation. The die's temperature is measured through voltage differences within the internal analog circuitry. The measured voltages are converted through ADC within SATA PHY block in order for the user to obtain the temperature data in the data register.

In order to obtain usable temperature, user must sample the voltage data 80 times each for two different sampling modes then apply the obtained data to the formula. The details on the procedures and formula will be shown in the following sections.

2 SATA Registers Used to Obtain Temperature Data

Obtaining the die temperature data requires the user to access internal SATA control and status registers. The internal SATA registers are not mapped to i.MX's register memory addresses. Therefore, to access these internal registers located in SATA, i.MX's SATA registers

Contents

(SATA_P0PHYCR and SATAP0PHYSR) are required.

2.1 SATA_P0PHYCR (SATA PHY control register)

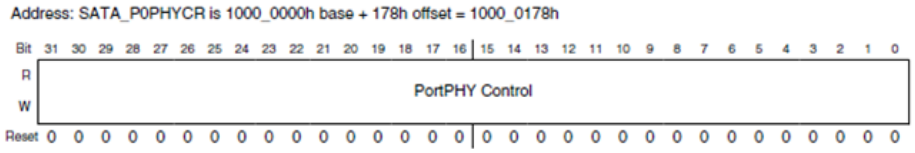

Figure 1. SATA PHY Control Register

Table 1. SATA_P0PHYCR Field Descriptions

Field	Description
31-0	PortPHY Control.

The register is used for Port PHY control. This register is also used to specify the address of the internal SATA register, specify the command and to write specific value to the internal registers. This register supports only 32-bit write access, so if the user wishes to retain command word but change one of the bits, 32-bit word must be written into the register in order to toggle only a few bits. In other words, bit wise operation cannot be performed.

2.2 SATA_P0PHYSR (SATA PHY Status Register)

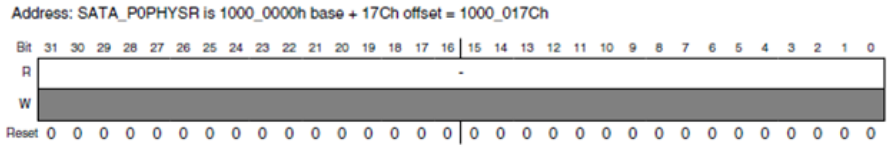

Figure 2. SATA PHY Status Register

Table 2. SATA_P0PHYSR Field Descriptions

Field	Description
31-0	PortPHY Status.

This register is used to monitor PHY status. This is an important register when the user is reading the status of the internal SATA register or read out result of the ADC within the SATA PHY. This is the register where the user will obtain the base data for the temperature calculation.

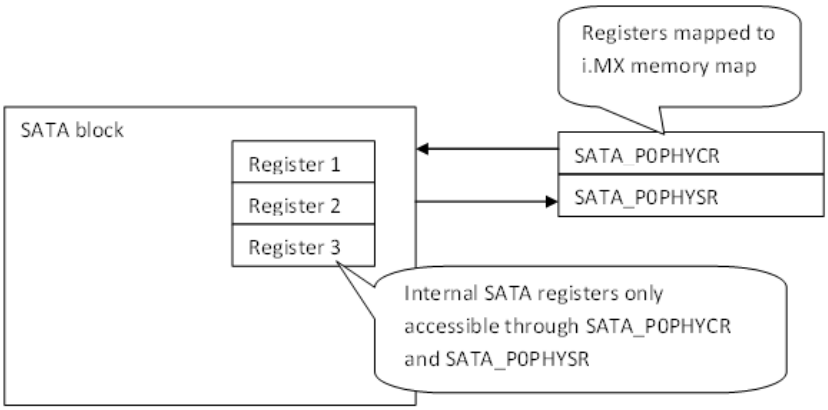

Figure 3. Internal Registers vs. i.MX Mapped SATA Register

2.3 Internal SATA PHY Registers Required for Temperature Data Calculation

Table 3 lists the registers required in order to obtain temperature data.

Table 3. Registers Required for Temperature Data

Register Name	Abbreviation	Address
DAC Control Register	DAC_CTL	0x0008
Resistor Tuning Control	Register	0x0009
ADC Output Register	ADC_OUT	0x000A
MPLL Test Register	MPLL_TST	0x0017

The internal SATA register addresses are 16 bit wide.

2.3.1 DAC Control Register

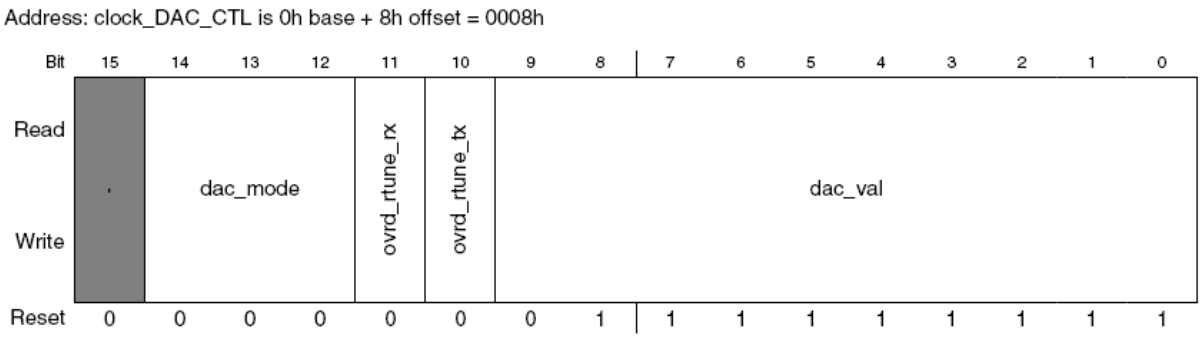

Figure 4. DAC Control Register

Table 4. DAC_CTL Field Descriptions

Field	Description
15	Reserved.
14–12 dac_mode	DAC output mode: 000 Powers down DAC 001 Reserved 010 High-range margining (VP25 x 418e-6 res) 011 Low-range margining (VP25 x 279e-6 res) 100 100% range DAC, 0% offset 101 36% range DAC, 0% offset 110 36% range DAC, 33% offset 111 36% range DAC, 66% offset
11 ovrd_rtune_rx	Writes DAC_VAL[5:0] to the Rx rtune bus.
10 ovrd_rtune_tx	Writes DAC_VAL[5:0] to the Tx rtune bus.
9–0 dac_val	Digital value to be used for DAC.

2.3.2 Resistor Tuning Control Register

Address: clock_RTUNE_CTL is 0h base + 9h offset = 0009h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	-					adc_trig	rtune_trig	rtune_dis	cmp_invert	dac_chop	rsc_x4	sel_atbp	pwron_lcl	frc_pwron	mode	
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figure 5. Resistor Tuning Control Register

Table 5. RTUNE_CTL Field Descriptions

Field	Description
15–11	Reserved.
10 adc_trig	Triggers ADC conversion.
9 rtune_trig	Triggers manual resistor calibration.

Table 5. RTUNE_CTL Field Descriptions

Field	Description
8 rtune_dis	Disables automatic resistor recalibrations.
7 cmp_invert	Inverts output of comparator (to reverse successive approximation register (SAR) feedback loop).
6 dac_chop	Polarity of chop control for DAC.
5 rsc_x4	Sets x4 in rescal circuitry.
4 sel_atbp	Selects atb_s_p for A/D measurement.
3 pwron_lcl	Value of power-on to force.
2 frc_pwron	Overrides internal power-on.
1-0 mode	Resistor tune SAR mode: 00 Normal restune 01 ADC 10 Rx Resistor test 11 Tx Resistor test

2.3.3 ADC Output Register

Address: clock_ADC_OUT is 0h base + Ah offset = 000Ah

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	-					fresh	value									
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6. ADC Output Register

Table 6. ADC_OUT Field Descriptions

Field	Description
15-11	Reserved.

Table 6. ADC_OUT Field Descriptions

Field	Description
10 fresh	Flag indicates that a new A/D conversion result is present.
9–0 value	A/D conversion result. Based on RTUNE_CTL.MODE, this value is the result of either the last conversion (MODES 0 or 1) or the current Tx/Rx cal value (MODES 3/2).

2.3.4 MPLL Test Register

Address: clock_MPLL_TEST is 0h base + 17h offset = 0017h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	ovrd_ctl	gearshift_val	reset_val	meas_iv											meas_gd	atb_sense
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7. MPLL Test Register

Table 7. MPLL_TEST Field Descriptions

Field	Description
15 ovrd_ctl	Overrides MPLL reset and gearshift controls.
14 gearshift_val	Value to override for mpll_gearshift.
13 reset_val	Value to override for mpll_reset.
12–2 meas_iv	Measures various MPLL controls: Bit 2: Measures dcc_vcntrl_p on atb_sense_p Bit 3: Measures dcc_vcntrl_m on atb_sense_m Bit 4: Measures 1-V supply voltage on atb_sense_m Bit 5: Measures vp_cp voltage on atb_sense_p; gd on atb_sense_m Bit 6: Measures VCO supply voltage on atb_sense_p; gd on atb_sense_m Bit 7: Measures clock tree supply voltage on atb_sense_p; gd on atb_sense_m Bit 8: Measures vp16 on atb_sense_p; gd on atb_sense_m Bit 9: Measures vref on atb_sense_p; gd on atb_sense_m Bit 10: Measures vcntrl on atb_sense_m Bit 11: Measures copy of bias current in oscillator on atb_force_m Bit 12: Enables phase linearity testing of phase interpolator and VCO

Table 7. MPLL_TEST Field Descriptions

Field	Description
1 meas_gd	Measures Ground For correct measurements, this field must be set when various meas_iv bits are set.
0 atb_sense	Hooks up ATB sense lines.

2.4 How to Access Internal SATA Registers Through i.MX SATA Register

All temperature reading operation must be executed through the usage of i.MX's SATA registers: SATA_P0PHYCR and SATA_P0PHYSR. SATA_P0PHYCR and SATAP0PHYSR are the two registers that are mapped to i.MX's SATA register locations. All of SATA's internal status and control registers located within SATA block must be accessed and modified through SATA_P0PHYCR and SATA_P0PHYSR. This rule, also applies to all of the registers relevant to temperature data read operation.

Even though the word length of SATA_P0PHYCR and SATAP0PHYSR are 32bit wide, the actual data address word for internal SATA register is 16 bits. The upper 16bits of SATA_P0PHYCR is used for the following actions.

- Bit 16 (0x0001_0000): this is to specify in SATA_P0PHYCR register that the user is writing a 16bit-word as an internal SATA register address value for capturing the data stored in that particular address location (LATCH_ADDR).
- Bit 17 (0x0002_0000): this is to specify in SATA_P0PHYCR register that the user is writing a 16bit-word as the data value to be written into internal SATA register (LATCH_DATA).
- Bit 18 (0x0004_0000): this is the command to write the data into the current captured internal SATA address (WRITE COMMAND).
- Bit 19 (0x0008_0000): this is the command to read the content of the current captured internal SATA address (READ COMMAND).

Likewise, SATA_P0PHYSR is 32bit wide and the lower 16bits are normally used for read register for the data captured from internal SATA registers. The upper 16bits are used for status check. Bit 18 is used for acknowledge flag. This bit is used to check whether the command is sent correctly, whether the address is ready to accept data or the data is ready to be read.

2.4.1 Specify Address and Reading the Contents of the Internal SATA Registers

Below, example is given how to specify the internal address and read the value stored in the register corresponding to the address value. In this example, MPLL_TST (address = 0x0017) address is accessed, and the value in MEAS_IV bit is stored into a temporary location.

2.4.1.1 STEP 1

Write internal SATA register address (value=0x0017) into SATA_P0PHYCR (i.MX register address = 0x1000_0178)

```
/*pseudocode*/
mpll_tst = 0x0017;
SATA_P0PHYCR = mpll_tst; /*write address value to P0PHYCR and goto next step*/
```

2.4.1.2 STEP 2

Capture the internal SATA register address for MPLL_TST, and check for ACK bit in SATA_P0PHYSR.

```
/*pseudocode*/
SATA_P0PHYCR = (mpll_tst | 0x10000);  /*bit 16 is used to capture the address*/
while( ((SATA_P0PHYSR >> 18)&0x1)!= 0x1);  /*dummy loop to capture acknowledge*/
SATA_P0PHYCR = mpll_tst & 0xffff;/*deassert bit16 (capture address bit)*/
while(((SATA_P0PHYSR>>18)&0x1) == 0x1); /*dummy loop to capture acknowledge deassertion*/
/*ready for next step*/
```

2.4.1.3 STEP 3

Send read command, then read the retrieved data from SATA_P0PHYSR.

```
/*pseudocode*/
SATA_P0PHYCR = 0x80000;/*read command sent to SATA block*/
while( ((SATA_P0PHYSR >>18)&0x1) != 0x1);  /*dummy loop to capture acknowledge*/
Readout = SATA_P0PHYSR & 0xffff; /*read  data state of the internal SATA register 0x0017*/
SATA_P0PHYCR = 0x00000000;/*deassert bit16 (capture address bit)*/
while(((SATA_P0PHYSR >>18)0x1) == 0x1); /*dummy loop to capture acknowledge deassertion*/
/*end of read register data*/
```

2.4.2 Specify Address and Writing Data Into the Internal SATA Registers

On this section, an example and pseudo code for write operation onto the same internal address (0x0017) is presented.

2.4.2.1 STEP 1

Write internal SATA register address (value=0x0017) into SATA_P0PHYCR (i.MX register address = 0x1000_0178)

```
/*pseudocode*/
mpll_tst = 0x0017;
SATA_P0PHYCR = mpll_tst; /*write address value to P0PHYCR and goto next step*/
```

2.4.2.2 STEP 2

Capture the internal SATA register address for MPLL_TST, and check for ACK bit in SATA_P0PHYSR.

```
/*pseudocode*/
SATA_P0PHYCR = (mpll_tst | 0x10000);  /*bit 16 is used to capture the address*/
while( ((SATA_P0PHYSR >> 18)&0x1)!= 0x1);  /*dummy loop to capture acknowledge*/
SATA_P0PHYCR = mpll_tst & 0xffff;/*deassert bit16 (capture address bit)*/
while(((SATA_P0PHYSR>>18)&0x1) == 0x1); /*dummy loop to capture acknowledge deassertion*/
/*ready for next step*/
```

2.4.2.3 STEP 3

```
Temp_write = 512; /*this is the actual data to be written into MPLL_TST*/
Temp_write |= 0x20000;/*write command sent to SATA block*/
SATA_P0PHYCR = Temp_write;
while( ((SATA_P0PHYSR >>18)&0x1) != 0x1);  /*dummy loop to capture acknowledge*/
Temp_write &= 0xffff/*deassert  capture address write*/
SATA_P0PHYCR = Temp_write;
while(((SATA_P0PHYSR >>18)0x1) == 0x1); /*dummy loop to capture acknowledge deassertion*/
Temp_write |= 0x40000;/*write command*/
while( ((SATA_P0PHYSR >>18)&0x1) != 0x1);  /*dummy loop to capture acknowledge*/
SATA_P0PHYCR = 0x0;/*deassert ack*/
while(((SATA_P0PHYSR >>18)0x1) == 0x1); /*dummy loop to capture acknowledge deassertion*/
/*end of pseudocode*/
```

3 SATA Temperature Accessing/Flow-Chart

SATA temperature monitor was designed to work when SATA is powered up. If a user wishes to measure the temperature but wishes not to use the SATA module for interfacing SATA devices, the SATA module must still be powered up through VP and VPH pins according to its prescribed voltage requirements. In addition to powering up SATA, reference clock (25MHz to 156.25MHz) must be supplied externally to SATA through external reference clock or USB PHY PLL. When the clock is supplied through USB PHY or external reference, SATA_REXT can be left unconnected. This section assumes that the SATA module is turned on properly prior to the usage, therefore the details on SATA setup prior to its usage is not going to be covered in this section.

In order to read the temperature off of the SATA block, it requires the user to take seven major steps. Details on each step are provided in the following paragraphs.

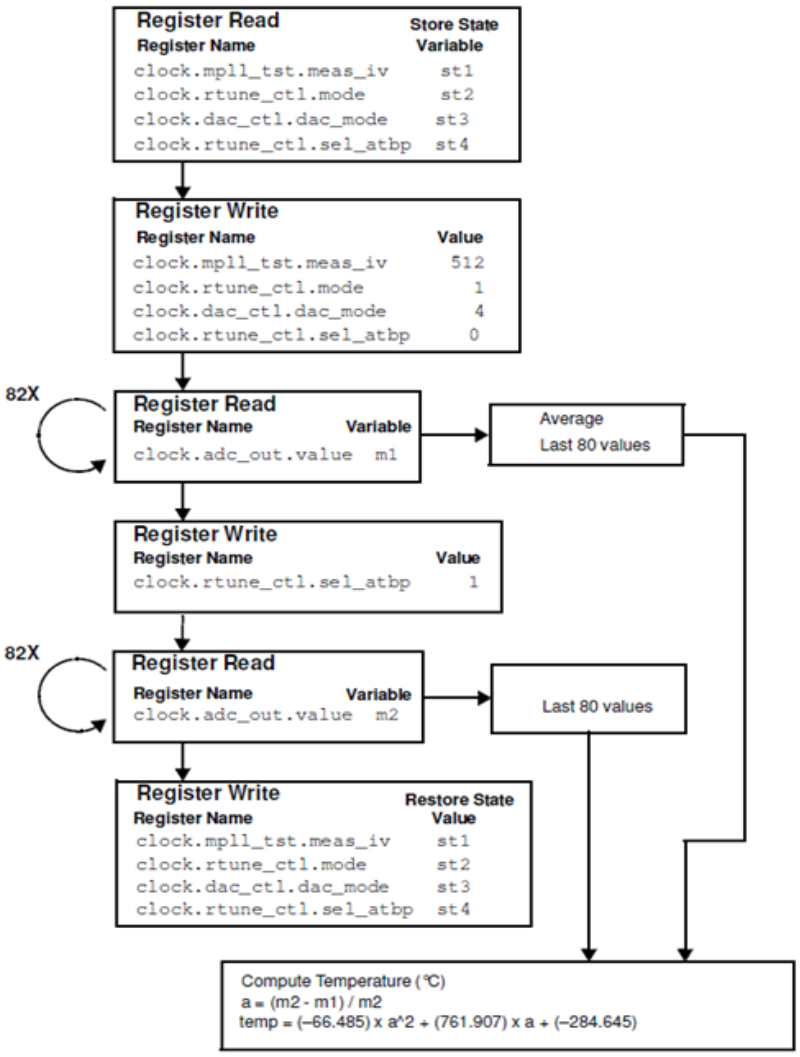

Figure 8. Temperature Measurement Flowchart

3.1 STEP 1: READ INTERNAL SATA Registers

The first step is to save the state of the registers the users are about to modify. Save these registers in a temporary location until temperature reading procedure is done. After temperature is obtained, the user is required to restore the register values before exiting the subroutine.

- Store MPLL_TST.MEAS_IV[12:2] into a temporary memory location
- Store RTUNE_CTL.MODE[1:0] into a temporary memory location
- Store DAC_CTL.DAC_MODE[14:12] into a temporary memory location
- Store RTUNE_CTL.SEL_ATBP[4] into a temporary memory location

On important note is that only to save those bits listed above and not the entire register, since SATA module may modify other bits during its operation. In other words, only to save the state of the register bits listed

above and only restore those bits listed above before exiting the subroutine. Failing to do so may cause erroneous SATA operation.

3.2 STEP 2: WRITE to Internal SATA Registers

On this step, the user is required to write the setup for each of the control registers that were accessed in the previous step. The actual values of the internal SATA registers are written below.

- Set MPLL_TST.MEAS_IV[12:2] = 0x200 (or 512)
- Set RTUNE_CTL.MODE[1:0] = 0x1
- Set DAC_CTL.DAC_MODE[14:12] = 0x4
- Set RTUNE_CTL.SEL_ATBP[4] = 0

One important note is that only modify those bits listed above; user should take extra care not to overwrite or modify other bits within the registers, since SATA and other software components (such as kernel of an OS) may modify other bits within the SATA module. It is a good programming exercise to only modify the bits of interest before updating the i.MX's SATA register.

3.3 STEP 3: 80 Consecutive Raw Data Read and Averaging

On this step, the raw data is retrieved from the ADC_OUT register of the internal SATA registers. The raw data must be read 82 times. The first and the second reads are done as dummy reads in order to ensure the data retrieved are not garbage data. After 80 consecutive read, the raw data are averaged, then stored as the first intermediate value. In the Figure2, this intermediate value is named as "m1."

3.4 STEP 4: Switch RTUNE_CTL.SEL_ATBP[4] to 1

On this step, bit 4 of RTUNE_CTL register must be set. This is to switch the measuring mode to ATB_SENSE. This prepares the system for the STEP 5.

3.5 STEP 5: 80 Consecutive Raw Data Read and Averaging

This step is similar to STEP 3. The number of read and procedure should be the same as the one described in STEP3. Only this time, the intermediate after averaging to be stored into the intermediate variable named "m2."

3.6 STEP 6: Calculate the temperature value

Using two averaged intermediate values "m1" and "m2," apply those values onto the following formula.

```
a = (m2 – m1)/m2

Temp = (-66.485) * (a^2) + (761.907)*a + -284.645
```

The calculated value "Temp" is the die temperature.

3.7 STEP 7

Restore register bits stored in STEP 1 and exit subroutine.

4 Limitation of the Temperature Readouts and Troubleshooting

Although the temperature data provides useful operational parameter, one must realize that the value gained in this procedure is only an approximation of the temperature within the die. The accuracy of the values calculated is dependent on the calculation method applied within the software and its precision. Also, the temperature value fluctuate greatly by the source voltage/current levels, number of modules and processes present, along with the surrounding air and junction temperature. Because of all the factors mentioned above, the values obtained through the temperature sensor should only be used as a reference point rather than to be taken as an absolute value.

5 Conclusion

Even with the limitations mentioned above, the temperature data on the die gives the user valuable information. This data point can be used to dynamically adjust the clock speed and/or to trigger the cooling mechanism. It can also be used to adjust the current source level after detecting certain temperature threshold. Whatever the application and its product specification requirements, this function can be a useful features to the users.

6 Revision History

Table 8 provides a revision history for this application note.

Table 8. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	10/2011	Initial Release.

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
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: AN4380
Rev. 0
10/2011