AN3202
Application note

How to configure the SPEAr600 general purpose timers (GPTs)

Introduction

This application note provides information about how to configure the general purpose timers (GPTs) integrated in the SPEAr600 embedded MPU family.

General purpose timers (GPTs) play an important role in any system as they provide a means of calculating time for controlling the execution of various operations. In case of an operating system, they are used for the system tick generation, usually every 10 ms; in other applications they can be used to get a finer granularity for controlling the timing of events.

The purpose of this application note is to explain how to read the free running timer counter and configure the clock source of the various GPTs that are integrated in the SPEAr600 architecture.

Contents

1 General purpose timers (GPTs) in SPEAr600

In the SPEAr600 architecture, there are five different GPT blocks located in the various subsystems. Each timer block consists of two independent channels, each one with a 16-bit counter register.

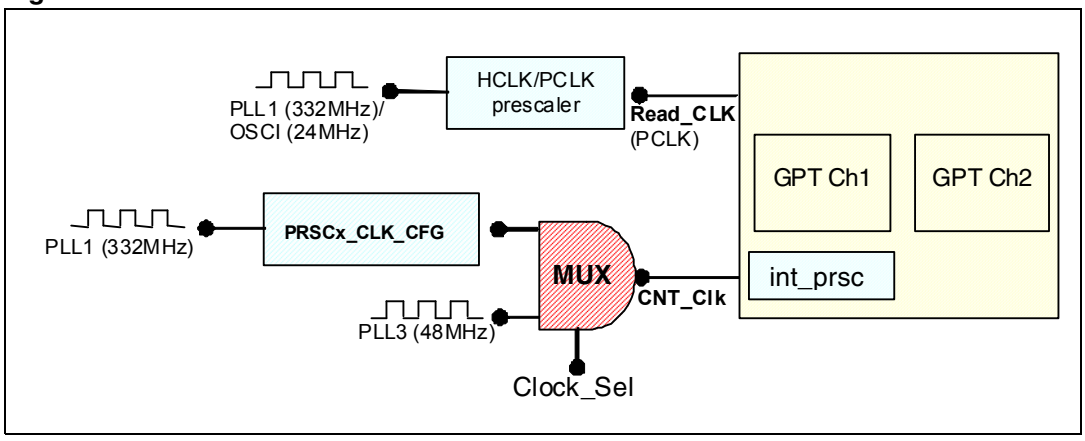
Table 1. GPTs in SPEAr600

	Subsystem	Base address
GPT1	ARM1	0xF000_0000
GPT2	ARM2	0xF000_0000
GPT3	Basic	0xFC80_0000
GPT4	Application 1	0xD800_0000
GPT5	Application 2	0xD808_0000

Each timer has a *READ_Clk*, input which is the APB clock (PCLK), and a *CNT_Clk*, which can be selected by the user from a list of clock sources.

- *READ_Clk* (PCLK): When SPEAr600 is in *normal* mode, it takes the input from PLL1 divided by a programmable prescaler, whose reset values impose the ratio 1:2:4 to the core_clk, HCLK and PCLK clocks. When SPEAr600 is in *slow* mode, it takes directly the input from the OSCI signal.
- *CNT_Clk*: The clock source can be selected as either a fixed 48 MHz or the PLL1 itself divided by a programmable prescaler, which is defined in the *PRSC1_CLK_CFG* register (0xFCA8_0044) for GPT1/GPT2/GPT3, *PRSC2_CLK_CFG* register (0xFCA8_0048) for GPT4 and *PRSC3_CLK_CFG* register (0xFCA8_004C) for GPT5. The *CNT_Clk* may then be further divided by a GPT internal 4-bit prescaler able to divide up to 256 times (‘/256’).

Figure 1. GPT clock sources



The following table describes the clock selectors (Clock_Sel) for each GPT.

Table 2. GPTx clock source selector

	Register	Address	Value
GPT1	PRPH_CLK_CFG [08]	0xFCA8_0028 (bit8)	0: PLL3 48 MHz 1: PLL1 (*PRSC1_CLK_CFG*)
GPT2	PRPH_CLK_CFG[09]	0xFCA8_0028 (bit9)	0: PLL3 48 MHz 1: PLL1 (*PRSC1_CLK_CFG*)
GPT3	PRPH_CLK_CFG[10]	0xFCA8_0028 (bit10)	0: PLL3 48 MHz 1: PLL1 (*PRSC1_CLK_CFG*)
GPT4	PRPH_CLK_CFG[11]	0xFCA8_0028 (bit11)	0: PLL3 48 MHz 1: PLL1 (*PRSC2_CLK_CFG*)
GPT5	PRPH_CLK_CFG[12]	0xFCA8_0028 (bit12)	0: PLL3 48 MHz 1: PLL1 (*PRSC3_CLK_CFG*)

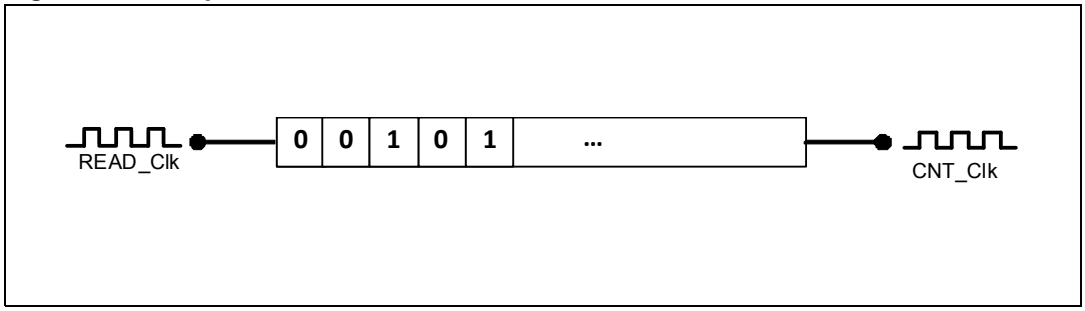
The SPEAr600 GPTs always generate precise alarm interrupts, for example in the case of a system tick for a RTOS. Nevertheless, as you can see in *Section 2: Reading a free-running timer counter*, GPTs can return unpredictable read values when they are running and the input clock is asynchronous (or not in phase).

2 Reading a free-running timer counter

When the GPT interrupt is enabled, the interrupts generated at each timer wrap-around condition are always triggered at the right frequency, however reading the timer counter when the timer itself is active and free-running may present some difficulties which are described below.

In a simplified scenario, a hardware timer block can be seen just as a simple counter register with two input clocks: *CNT_Clk* for incrementing/decrementing the counter and *READ_Clk* for synchronizing the READ accesses of the bus the timer is connected to.

Figure 2. Simplified timer

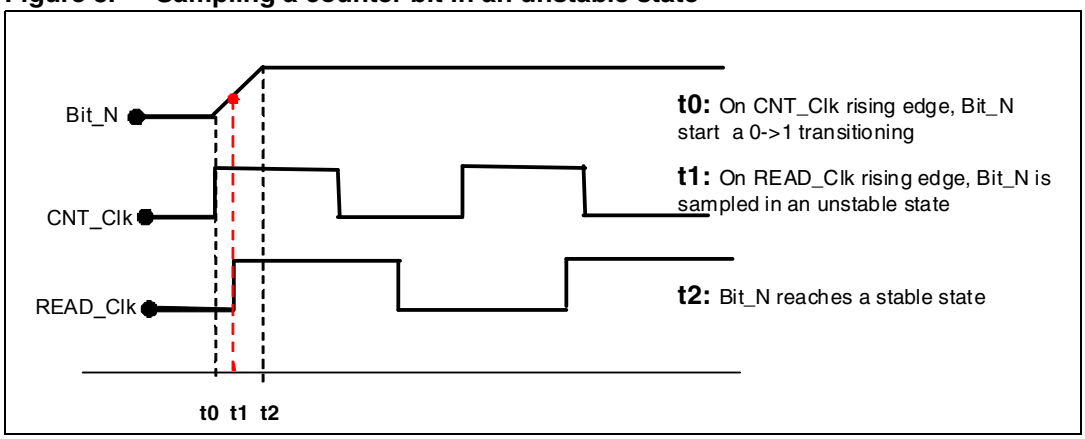


The two clocks can be either synchronous, coming from the same source PCLK, or completely asynchronous, for example coming from two different sources.

When the two clocks involved in the scenario are asynchronous, then the value retrieved by the CPU in a read counter operation is unpredictable, and might be completely different from the real value in the register.

The situation is due to the fact that the *READ_Clk* is sampling the counter bits while they are in a transitioning, unstable phase.

Figure 3. Sampling a counter bit in an unstable state



The above scenario may take place during any kind of transition (0->1 or 1->0) and for any bit in the register.

If one of the bits impacted has a large weight (significant position) in the counter, then the difference between the value returned in the read transaction and the real value of the counter can be very large.

Let's take as an example a counting down 16-bit counter transitioning from the value **1000_0000_0000_0000** (0x8000) to **0111_1111_1111_1111** (0x7FFF). Since the transition time of the 16 bits can be slightly different between each other, then the 16-bit counter value could be read by the CPU randomly as 0x0000 or 0xFFFF leading to a big difference from its real value.

A similar scenario may also occur in case the two clocks are synchronous, but not in phase. In this case, in fact, the *READ_Clk* may sample the bit during its unstable state period.

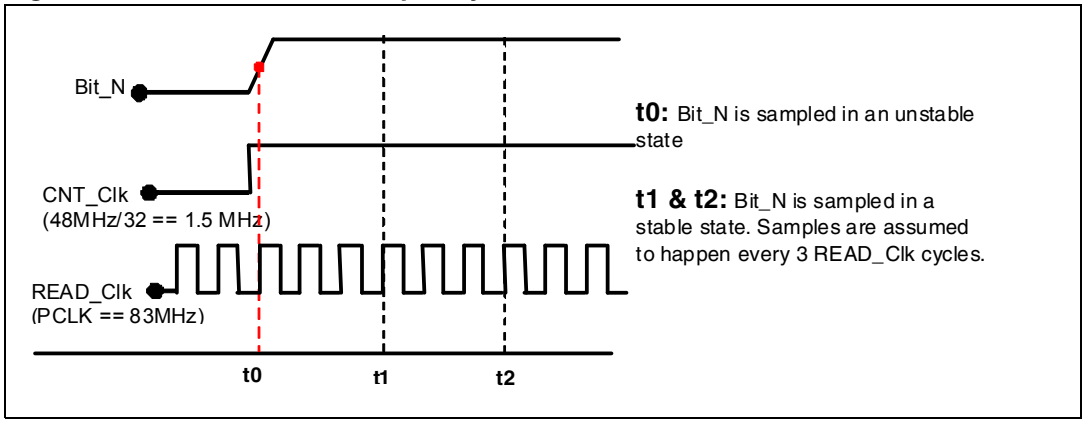
So, the two clocks must be synchronous and in phase.

3 Scenario with slow CNT_Clk and fast READ_Clk

In certain cases, for example when the timer is used by an operating system to generate the system tick, the *CNT_Clk* (after prescaling) is usually much slower than *READ_Clk*. For example, let's suppose you need to generate a tick every 10 ms; the GPT with a clock source of 48 MHz might be programmed using a '/32' internal prescaler and a counter equal to 15000.

This results in a great number (around 60) *READ_Clk* '*sampling cycles*' for every single *CNT_Clk* cycle. Or, in other words, *CNT_Clk* is about 60 times slower than *READ_Clk*.

Figure 4. CNT_Clk at low frequency



Let's see what happens if the CPU does three consecutive read operations instead of a single one. Since the bit instability lasts much less than the *CNT_Clk* time period, we can say that, out of 3 *READ_Clk* edges, only one will ever fall into the bit instability window. The other two are stable.

Moreover, since *CNT_Clk* is about 60 times slower than *READ_Clk,* the two stable read operations return counter values that differ by 1 in the worst case, which is when there is a *CNT_Clk* rising edge between the first and third read operations. Of course, interrupts should be disabled during the reads.

So, reading three times the counter and discharging the unstable value (if any) is a valid workaround that can be used for all GPTs of SPEAr600 in similar scenarios. In particular, this method might be used for **GPT1**, **GPT2** and **GPT3**.

In general, this workaround is valid when the minimum period of *CNT_Clk* is greater than 3 times the read_cycle_time. The read_cycle_time depends on the CPU frequency, and also on the way the reads are implemented, so they should be carefully evaluated.

4 How to configure CNT_Clk and READ_Clk to be synchronous

This method, which is very simple to implement, is nevertheless guaranteed to work for **GPT4** and **GPT5** only.

The most common configuration is when SPEAr600 is in normal mode with system clocks fed by PLL1. In case the system is set in this mode, you can just select PLL1 as CNT_Clk to guarantee the synchronicity between CNT_Clk and READ_Clk.

To set the input clock source of GPTx to PLL1 you need to use PRPH_CLK_CFG register (0xFCA8_0028). There are five different bits, one for each GPT block.

- For **GPT4**: PRPH_CLK_CFG [11] = 1
- For **GPT5**: PRPH_CLK_CFG [12] = 1

In case SPEAr600 enters the slow mode, for example to save power after detecting a period of inactivity, the HCLK/PCLK system clocks are directly fed from the OSCI at 30 MHz. In this mode READ_Clk (OSCI) and CNT_Clk (PLL1) become asynchronous again.

5 Summary

A general purpose timer can be seen as a simple counter with two clocks in input: *READ_Clk* (for the slave interface) and *CNT_Clk* (for incrementing/decrementing the counter).

The *CNT_Clk* for the GPT in SPEAr600 can be selected between a fixed 48 MHz source and PLL1, which is also the source clock for the rest of the system. The *READ_Clk* is derived from PLL1 in normal mode (PCLK) and from the 30 MHz OSCI in slow mode.

Having a fixed clock source different from the system clock has the advantage of eliminating the need for reconfiguring the GPT registers if the system clock frequency is slowed down. However, it introduces the possibility of obtaining an unpredictable result when reading the timer value, due to the non-synchronous operation of the two clocks.

In case CNT_Clk is much slower than READ_Clk, three consecutives read of the counter (3-reads workaround) guarantees to have at least two stable values with a maximum difference of 1.

The following table summarizes the suggested solutions for this issue:

Table 3. Summary of the solutions

	Subsystem	Solution in normal mode	Solution in slow mode
GPT1	ARM1	3-reads workaround	3-reads workaround
GPT2	ARM2	3-reads workaround	3-reads workaround
GPT3	Basic	3-reads workaround	3-reads workaround
GPT4	Application1	Keep *READ_Clk* and *CNT_Clk* synchronous	3-reads workaround
GPT5	Application2	Keep *READ_Clk* and *CNT_Clk* synchronous	3-reads workaround

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
03-May-2010	1	Initial release.