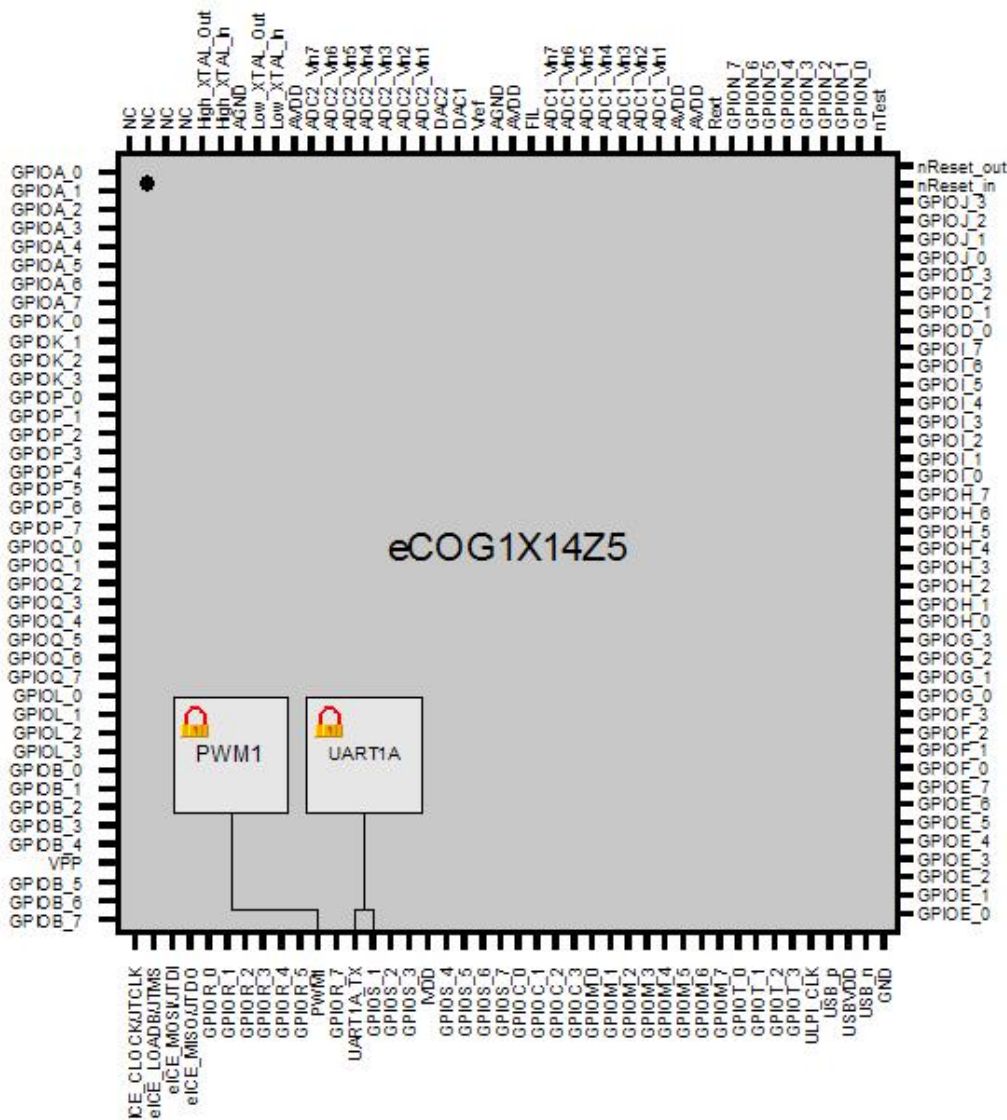




# AN050 – eCOG1X Low Power Mode

## Version 1.1

This application note describes some power saving features of the eCOG1X microcontroller, and provides an example showing how to put the device into its lowest power consumption mode (deep sleep mode).



## Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



## Revision History

Version	Date	Notes
V1.0	08/05/2007	First release.
V1.1	03/04/2008	Added clk_en bit set and debugging description section.

## Contents

1	Introduction .....	5
2	Glossary .....	5
3	Embedded Flash Memory Modes .....	6
3.1	Fast Mode.....	6
3.2	Slow Mode.....	6
3.3	Stop Mode .....	6
3.4	Power Consumption Summary .....	6
4	Flash Timer.....	7
4.1	Flash Timer Registers .....	7
4.2	Flash Timer Clock.....	8
4.3	Flash Mode Changes .....	8
4.4	Flash Operation in Slow Mode .....	9
4.5	Changing from Fast to Slow Mode .....	9
5	Deep Sleep Mode .....	10
5.1	Debugging with Deep Sleep Mode .....	10
6	Example Application .....	11
7	Guidelines for Achieving Minimum Power Consumption .....	11

## 1 Introduction

The eCOG1X has a number of power saving features which can be used to reduce the overall power consumption of the device. To reduce the power consumption to a minimum in low-power standby operation, the eCOG1X has a “Deep Sleep” mode in which all internal clocks are stopped and only leakage current is drawn by the device.

One significant step required to achieve minimum power consumption and Deep Sleep mode is to power down the flash memory within the eCOG1X. This process must be handled carefully, since in normal operation the eCOG1X is executing code from the flash.

## 2 Glossary

A table of abbreviations used in this document.

eCOG1X	Cyan Technology target micro controller
SSM	System Support Module

The on-chip I/O registers may be accessed as complete registers, or as named bit fields within the registers. The include file registers.h contains structure definitions for all on-chip registers and bit fields. To access any register, use the structure prefix **rg**. To access a bit field within a register, use the structure prefix **fd**. For example:

```
// No timeouts on DUART A
rg.duart.a_tmr_cfg = 0;
// Enable DUART A transmitter
fd.duart.ctrl.a_tx_en = 1;
```

### 3 Embedded Flash Memory Modes

The Embedded Flash Memory module of the eCOG1X can be placed in a number of operating modes, each with different power consumption and timing requirements. These are described briefly below.

#### 3.1 Fast Mode

In Fast mode, the flash memory has its fastest access time, but it also requires the largest power supply current.

#### 3.2 Slow Mode

In Slow mode, the power consumption is decreased significantly. The access time is increased such that the CPU clock speed must be reduced or the device must be configured for a large number of wait states on flash memory read cycles.

#### 3.3 Stop Mode

In Stop mode, the flash memory draws only leakage current. However, it cannot be accessed in this mode for either instruction fetch or data read cycles.

#### 3.4 Power Consumption Summary

This table gives a summary of the power consumption and access times in the different modes.

Mode	Cycle Time	Recovery Time	Total Power (Typical)
Fast	49ns cycle time 34ns access time	None	43.2mW when active 1.32mW when inactive
Slow	60µs cycle time 2µs access time	2µs from Slow to Fast	2.05mW when active 0.144mW when inactive
Stop	n/a	20µs from Stop to Slow or Stop to Fast	Leakage current only

## 4 Flash Timer

An on-chip hardware timer is used within the flash controller block to time the changes between the different flash operating modes. The times used to change from Fast to Slow mode and from Slow to Stop mode are arbitrary and up to the application.

The recovery time to change from Stop to Slow mode or from Stop to Fast mode is constrained by the hardware and is specified as a minimum of 20µs. The recovery time to change from Slow to Fast mode is a minimum of 2µs.

### 4.1 Flash Timer Registers

Four registers are used to specify the time delays for the transitions between the different power states of the flash:

Timer Register	Function
<i>flash.tmo_fast_slow</i>	Timer register that specifies the flash idle time before the flash is moved from the Fast mode to the Slow mode. Delay = (N+1) x flash_tmr clock period.
<i>flash.tmo_slow_stop</i>	Timer register that specifies the flash idle time before the flash is moved from the Slow mode to the Stop mode. Delay = (N+1) x flash_tmr clock period.
<i>flash.recover_slow</i>	Timer register that specifies the recovery time from Slow mode when the flash is accessed. This must be set so that the delay of (N+1) x flash_tmr clock period ≥ 2µs.
<i>flash.recover_stop</i>	Timer register that specifies the recovery time from Stop mode when the flash is accessed. This must be set so that the delay of (N+1) x flash_tmr clock period ≥ 20µs.

Two further register bit fields within the *rg.flash.cfg* register are used to control the mode in which the flash memory is operating.

The first is *fd.flash.cfg.slow\_mode*; this bit field is used to change the state of the flash between Fast and Slow modes. When set to '1', the flash is immediately changed to the Slow mode. When set to '0', the next flash access begins the slow-to-fast recovery delay time and the flash changes back to the Fast mode when the delay timer expires.

<i>slow_mode</i>	Function
0 (default)	When accessed, the flash memory operates in Fast mode.
1	When accessed, the flash memory operates in Slow mode.

The second is *fd.flash.cfg.pwr\_save*; this bit field controls the lowest power operating mode to which the flash can power down automatically.

<i>pwr_save</i>	Function
00 (default)	The flash memory does not power down automatically but remains in Fast mode.
01	The flash memory can power down automatically from Fast to Slow mode.
10	The flash memory can power down automatically from Fast to Slow mode and then from Slow to Stop mode.
11	Reserved – do not use.

## 4.2 Flash Timer Clock

The flash memory delay timer is driven from a peripheral clock called *flash\_tmr* in the SSM. This is configured by selecting a clock source and a divisor from the associated ripple counter. The register field *fd.ssm.clk\_src2.flash\_tmr* selects the clock source and has one of the following values:

0 (default)	Disabled	Disables the clock source for the flash timer
1	high_ref_clk	Selects the high reference oscillator
2	high_pll_clk	Selects the high PLL clock
4	low_ref_clk	Selects the low reference oscillator
5	low_pll_clk	Selects the low PLL clock
7	relax_osc_clk	Selects the relaxation oscillator

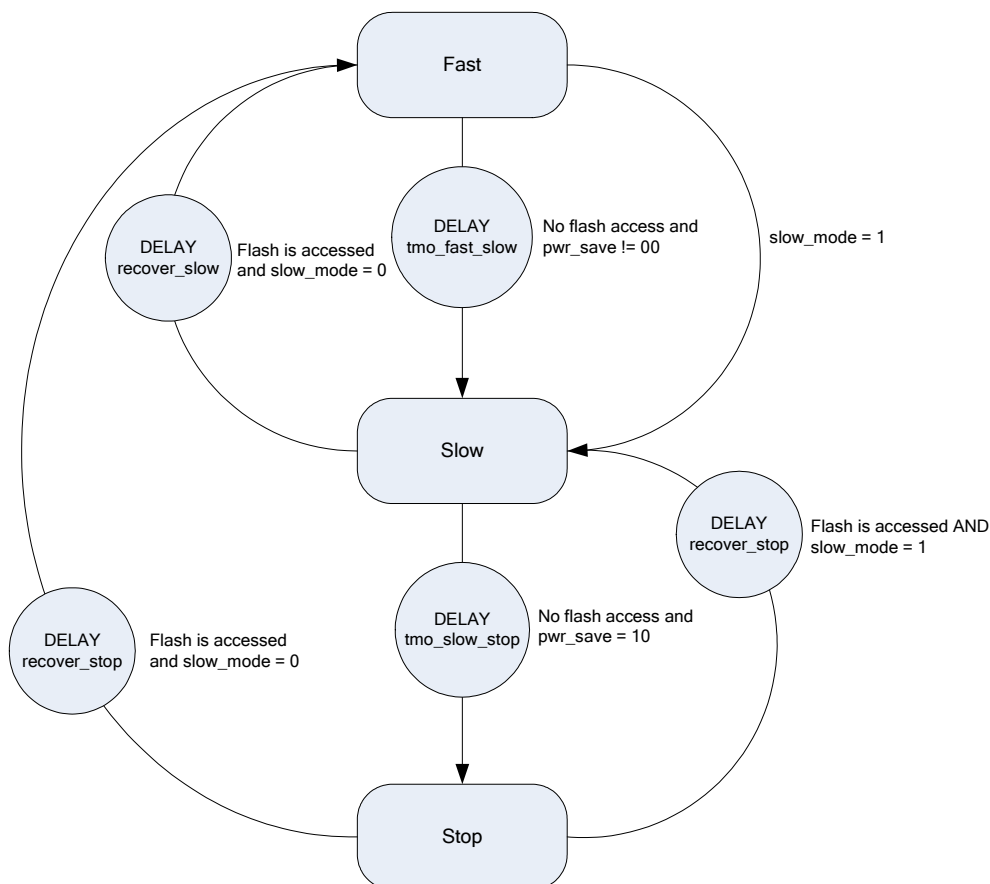
The register bit field *fd.ssm.clk\_div6.flash\_tmr* selects one output from the clock divider chain for the clock source selected by the *fd.ssm.clk\_src2.flash\_tmr* bit field. The smallest division ratio is  $\div 2$ , selected by setting the field to 15 (0xf), and the largest division ratio is  $\div 2^{16}$ , selected by setting the field to zero.

The *flash\_tmr* clock is enabled by setting the *fd.ssm.clk\_en2.flash\_tmr* bit field.

When setting the clock source and divider, it is important to ensure that the appropriate divider chain is released from reset. The reset clear bits for the five divider chains can be found in bits 0 to 4 of the *rg.ssm.rst\_clr1* register.

## 4.3 Flash Mode Changes

The following diagram illustrates the flash memory operating modes and the mechanisms for changing between them.



## 4.4 Flash Operation in Slow Mode

When the flash memory is operating in Slow mode, the access time of the flash is increased as outlined above. The flash timer does not ensure automatically that these timing requirements are met and so the application must set the wait states to meet these requirements before the flash memory is set to Slow mode.

The flash memory access timing is controlled by the **wait\_states** and **data\_release** bit fields in the **mmu.flash\_ctrl** register. These timing parameters are in units of the current CPU clock period. The value of the **data\_release** bit field must always be less than or equal to the value in the **wait\_states** bit field, otherwise a lockup occurs when accessing the flash.

The value N in the **fd.mmu.flash\_ctrl.wait\_states** field specifies one less than the total flash access cycle time. In Slow mode, this should be set to give a total memory access time of 60µs. This is used as the starting value for a wait state down counter, used to time each read access to the flash memory.

The value N in the **fd.mmu.flash\_ctrl.data\_release** field specifies the count value in the wait states down counter at which the read data is sampled. It is recommended this is set to give a release time of at least 5µs in Slow mode.

With a 500kHz CPU clock (2µs clock period) the following values should be used:

**fd.mmu.flash\_ctrl.wait\_states** = 29 (60µs cycle time)  
**fd.mmu.flash\_ctrl.data\_release** = 27 (6µs release time)

Note that if the application uses the low reference clock as the CPU clock source, then the fastest CPU clock speed of 16kHz meets exactly the timing requirements for Slow mode and can be used with zero wait states.

## 4.5 Changing from Fast to Slow Mode

Typically, applications require only Fast or Slow mode. If both modes are required in the same application, then either the cycle and access times must be set for the worst case (Slow mode) or the application must control the switch between the modes and update the times appropriately.

To change from Fast to Slow mode, the following sequence must be observed:

1. Set the cycle and access times for Slow mode.  
(Set them simultaneously as they are in the same register.)
2. Set the **slow\_mode** bit field.

To change from Slow to Fast mode the, following sequence must be observed:

1. Clear the **slow\_mode** bit field
2. Set the cycle and access times for Fast mode.

When using the flash timer to control the flash power state in combination with the deep sleep mode, it is recommended that the **pwr\_save** bit field is set to Stop mode before the *sleep* instruction and set back to Fast mode after the *sleep* instruction.

## 5 Deep Sleep Mode

In combination with the automatic power down of the flash memory, it is possible to configure the eCOG1X to stop the clock from which it is running and so enter a completely static state.

The only clock source that can be used in this way is the relaxation oscillator. When the CPU enters the sleep state and the flash automatically powers down to Stop mode, the relaxation oscillator can then be stopped. When a wakeup event occurs, the relaxation oscillator is restarted and execution resumes.

With the CPU clock source set to the relaxation oscillator, the ***fd.ssm.clk\_dis1.relax\_osc*** bit field must be set. Normally this disables the relaxation oscillator, but since it is selected as the CPU clock source, it remains enabled. The ***fd.ssm.clk\_dis2.relax\_osc\_suspend*** bit field must also be set. This configures the relaxation oscillator such that when the CPU is in Sleep state and the flash has automatically changed to the Stop mode, the oscillator is disabled.

Note that when disabling other clock sources and peripheral clocks in this process, the peripheral clock for the flash timer is left enabled. The following example code disables all the clocks appropriately and leaves the flash timer running:

```
// Disable all the clocks except the flash power down timer
// This also enables the relax_osc_suspend mode
rg.ssm.clk_dis2 = (0x7FFF ^ SSM_CLK_DIS2_FLASH_TMR_MASK);
rg.ssm.clk_dis1 = 0xFFFF;
```

When a *sleep* instruction is executed and the ***fd.ssm.sleep.evening*** field is set, the CPU enters the sleep state. Following this, as the CPU has now stopped performing instruction fetch read cycles from flash memory, the flash timer automatically powers down the flash memory to Stop mode. Once the flash memory is in Stop mode, the relaxation oscillator is disabled automatically, and the eCOG1X enters a completely static state. This gives the lowest possible power supply current.

When a wakeup event is received from a GPIO interrupt, the relaxation oscillator is enabled and execution restarts in the same way as waking from normal sleep mode.

This mode is advantageous as it allows a fast start up from sleep mode, but retains the low power consumption associated with using a low frequency clock. If the relaxation oscillator frequency is adjusted to 10MHz with an external tuning resistor, then the wakeup time is approximately 5µs. If the relaxation oscillator frequency is at its default value (approximately 1MHz) with no external tuning resistor, then the wakeup time is approximately 50µs.

A simple test using the eCOG1X processor daughter board from the development kit shows that it requires a standby current of approximately 5µA on both the 1.8V and 3.3V supplies in this deep sleep mode.

### 5.1 Debugging with Deep Sleep Mode

Note that it is not possible to maintain debug access through the eICE interface when the eCOG1X enters deep sleep mode, because all the device clocks are stopped. If a CyanIDE debug session is in progress when the eCOG1X enters deep sleep mode, then the next eICE transaction fails and the CyanIDE debug session stops with an error.

To prevent this, it is recommended that two builds of the application project are used, one for debugging and one for the final application.

Applications built from one of the CyanIDE templates or example projects always have two build configurations, labelled *Release* and *Debug*. The difference between the two is that the *Release* build has the symbol "NDEBUG" defined. The presence of this defined symbol can be used to change slightly the behaviour of the deep sleep function between the two builds and so allow the application to be debugged.

The change between the two builds is to enable the ***fd.ssm.clk\_dis2.relax\_osc\_suspend*** bit only in the release build, such that when the eCOG1X executes the *sleep* instruction in the *Debug* build the relaxation oscillator remains enabled.

## 6 Example Application

The example code provided with the application note in the file *AN050SW.zip* demonstrates the deep sleep mode operation.

A GPIO input is configured on PortR\_7 to act as a wake up input in to the eCOG1X; a falling edge on this input wakes the eCOG1X from deep sleep. On the eCOG1X Development Board Rev. B, this input is connected to dip switch SW3\_4, when the LED and Switch Jumper is connected across J2 and J27.

A PWM timer output signal is generated on PortR\_6; this can be used to confirm that the internal clocks are running or have stopped. On the eCOG1X Development Board Rev. B, this output signal is connected to LED7, when the LED and Switch Jumper is connected across J2 and J27.

All other port pins are configured as GPIO outputs and driven high to prevent them floating and drawing current from the 3.3V supply.

The two builds (*Release* and *Debug*) demonstrate the use of the NDEBUG symbol to allow debugging with CyanIDE in the *Debug* build. To switch between the two build variants, select *Build->Configurations* from the main menu, then select the required build. The currently selected build is displayed in the CyanIDE title bar, after the project path.

## 7 Guidelines for Achieving Minimum Power Consumption

Drive all unused port pins with GPIO to prevent them floating.

Ensure that all peripheral input pins (for example the MISO pin of the SPI interfaces) are either permanently driven, or have bias resistors to prevent them floating when undriven.

On the 100QFN and 208BGA parts with USB (such as the eCOG1X14Z5 used on the standard Development Kit Daughter Board), there is a dedicated high-speed clock input ULPI\_CLK that must not be left floating. If this pin is not used in the target system, it should be pulled low or tied to GND.