XS1-L Active Energy Conservation

Version 1.0

XMOS

Publication Date: 2010/04/27

1 PLL and Clock Divider Overview

The XS1-L provides the option of using Active Energy Conservation (AEC). AEC reduces the XCore clock frequency to reduce dynamic power dissipation when all active threads are paused. AEC can reduce dynamic power dissipation by up to 80% when paused with no loss of functionality. AEC can be used to reduce the clock frequency to practically eliminate dynamic power consumption. The XCore memory is retained and the reference clock continues to operate when the XCore clock is slowed. When an event occurs and wakes up one of the threads, the XCore promptly returns to full operating speed.

2 System Clock Dividers

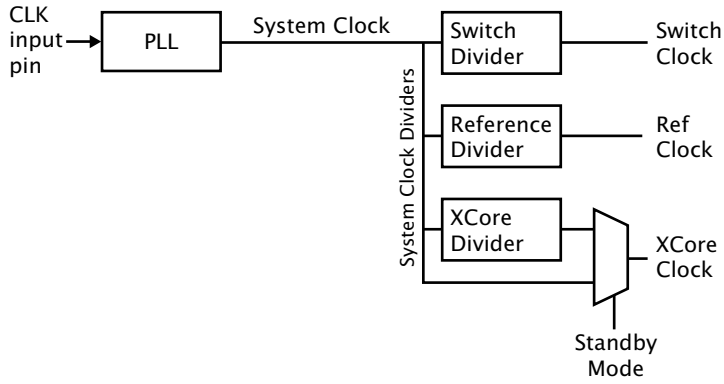

Figure 1: PLL and System Clock Dividers

The XS1-L system clock is obtained from the CLK input pin using the PLL as shown in Figure 1. Three clock dividers are used to obtain the switch, reference and XCore clocks. Typical operating frequencies might be: the CLK input pin operating at 20MHz; the system clock at 400MHz, the reference clock at 100MHz and both the switch and XCore clocks operating at 400MHz.

The XCore clock either uses the system clock directly (active mode) or uses the output of the XCore clock divider (standby mode). When AEC is enabled, standby mode is automatically selected when all of the active threads are paused waiting for resources. Alternatively, the code may choose not to use AEC. Instead the code may enable the XCore clock divider for all, part or none of the code's execution time.

3 Using AEC

To enable AEC, the XCore standby clock divider must be set and AEC mode enabled.

Code Snippet: Enabling AEC with Standby Frequency of 100MHz

```
// Set XCore standby clock to 100MHz from 500MHz System frequency
#define STANDBY_CLOCK_DIVIDER   (5-1)
#define XCORE_CTRL0_CLOCK_MASK  0x30
#define XCORE_CTRL0_ENABLE_AEC  0x30

void enableAEC (unsigned standbyClockDivider)
{
  unsigned xcore_ctrl0_data;

// Set standby divider
  write_pswitch_reg(get_core_id(),
              XS1_PSWITCH_PLL_CLK_DIVIDER_NUM,
              standbyClockDivider);

  // Modify the clock control bits
  xcore_ctrl0_data = getps(XS1_PS_XCORE_CTRL0);
  xcore_ctrl0_data &= 0xffffffff - XCORE_CTRL0_CLOCK_MASK;
  xcore_ctrl0_data +=  XCORE_CTRL0_ENABLE_AEC;
  setps(XS1_PS_XCORE_CTRL0, xcore_ctrl0_data);
}

int main (void) {
  ...
  enableAEC ( STANDBY_CLOCK_DIVIDER);
  ...
}
```

When any active thread has fast mode enabled standby mode will not be entered.

3.1 Timers

The reference clock continues to operate at the same frequency when standby mode is entered and exited. Timer operations can be freely used when AEC is used. If a timer operation causes the last active thread to pause, standby mode is entered if AEC is enabled. Equally if a timer expires when the XCore is in standby, the thread waiting for the timer will be woken up, and the XCore will return to the XCore active mode frequency.

A number of standby mode frequency clock pulses are required to bring the XCore

into active mode. See section 4.3 (Wake-up Response Delay) for more details.

Code Snippet: Timer used to control standby mode

```
// Pause until next bit needs outputting
// XCore may enter standby here
bitRateTimer when timerafter(nextBitTime) :> void;
```

3.2 Channels

In systems with more than one XS1-L XCore, an XCore with AEC enabled can be placed into standby waiting for channel input from another XCore via a link.

With the receiving XCore in standby mode, data continues to be transferred at full speed over the link. This is because the link speed is determined by the switch clock and the link's configuration, not the XCore clock. However, when the channel's data is transferred from the switch to the XCore channel end, the data transfer rate slows to the standby frequency. When the data reaches the channel end the XCore returns to active mode. Channel communication continues to be fully functionally during standby mode, albeit at a slower rate until the XCore exits standby.

The XCore will not enter standby when threads are paused waiting for channel communication across a single XCore.

Code Snippet: Channel End and Timer used to control standby mode

```
// XCore may enter standby within this select
select {
  // Respond to a command from the master XCore
  case controlChan :> command :
    ...
    break ;

  // Alternatively continue when the next bit needs outputting
  case bitRateTimer when timerafter ( nextBitTime ) :> void :
    break ;
}
```

3.3 Ports

Signals on port pins can be used to enter and exit standby when AEC is enabled. This may be done by performing a conditional input on that port.

The ports remain fully operational so long as the XCore standby frequency is the same as, or greater than the reference clock frequency. Dynamic power can be significantly reduced in this way whilst maintaining full system functionality.

When very low dynamic power dissipation is required, the standby frequency may be lower than the reference clock. When this is the case, buffered ports cannot be used during standby and any unbuffered port that is used must be clocked from the XCore clock.

Standby Frequency	Port Use Restrictions during standby
$f_{STANDBY} >= f_{REF}$	None
$f_{STANDBY} < f_{REF}$	Must be unbuffered and clocked off the XCore clock

Table 1: Port restrictions during standby

Code Snippet: Port used to control standby mode ($f_{STANDBY} = f_{REF}$)

```
// Pause until 1 bit port sleepPort negated
sleepPort when pinsneq (0) :> void;
```

Code Snippet: Port used to control standby mode ($f_{STANDBY} < f_{REF}$)

```
// assembler macro for setting a clock
#define SETCLK(res, src) asm("setclk res[%0], %1" : : "r"(res), "r"(src));
...
// Use XCore clock for sleepPortClock
// And attach sleepPort to sleepPortClock
stop_clock(sleepPortClk);
SETCLK(sleepPortClk, XS1_CLK_XCORE);
set_port_clock(sleepPort, sleepPortClk);
start_clock(sleepPortClk);
...
// Pause until 1 bit port sleepPort negated
sleepPort when pinsneq (0) :> void;
```

3.4 Divide Unit

When a divide (or remainder) operation is performed on a thread, that thread will pause until the result is available. Even if AEC is enabled, the XCore will not enter standby mode if there is an active divide operation.

4 Selecting the Optimal XCore Standby Frequency

The XCore standby frequency is determined by the system clock frequency and the XCore clock divider (See Figure 1). The following factors need to be considered when chosing the standby frequency.

- Is full port operation required? If so the minimum standby frequency is the reference frequency.
- If a port is used with low standby frequencies, the standby frequency must be fast enough to sample the wake-up signal on the port.
- Is the reduction in dynamic power enough for the application?
- Is the wake-up time adequate for the application?

4.1 Edge Detection on Ports

For a signal to be detected on an input port, a port clock must occur whilst the signal is asserted. If $F_{STANDBY} < F_{REF}$ the XCore standby clock will be used to sample the port (see section 3.3). So if an input signal is asserted for 1us, for the port to detect that signal the standby frequency must be above 1MHz. This becomes important when AEC is enabled if a port is being used to exit standby. If the system clock is 400MHz and the XCore clock divider is set to 400, the port will be clocked at 1MHz, and input signals asserted for 1us or less will not be reliably detected.

The following equation shows how to determine if the XCore clock divider is too large to allow detection of a signal. XCoreClockDivider must meet the following condition in order to detect a signal:

$$\text{XCoreClockDivider} < \text{SignalDuration (s)} / F_{SYSTEM}\ \text{(Hz)}$$

4.2 Dynamic Power

The dynamic power of the XCore reduces linearly as the XCore clock frequency reduces. See *Estimating Power Consumption For XS1-L Devices* [1] for details. An example from that document has a 500MHz XS1-L1 device consuming 20mA of static current and 180mA of dynamic current. 20+180=200mA.

Using AEC to reduce the XCore clock to match the 100MHz reference during standby, the standby current is reduced to 20+(180/5)=56mA. This is a substantial saving to be made when all threads are paused whilst full functionality is maintained.

Where AEC is used to further reduce standby current with the loss of full port functionality, if the XCore frequency is reduced to 1MHz (an XCore clock divide value of 400), the standby current is reduced to 20+(180/500)=20.4mA. Clearly static current dominates at this frequency so there is little to be gained by reducing the XCore frequency further.

Mode	XCore Clock Divider	XCore Freq (MHz)	Functionality	Dynamic Current (mA)	Static Current (mA)	Total Current (mA)
active	1	500	Full	180	20	200
standby	5	100	Full	36	20	56
standby	500	1	Unbuffered, XCore clocked ports only	0.36	20	20.4

Table 2: Example Comparison of Current Draw for Different XCore Frequencies

4.3 Wake-up Response Delay

When a resource brings the XCore out of standby mode, the XCore is clocked for a few cycles at the standby frequency before active mode is entered. This means that the response time to resource events take longer when coming out of standby mode (wake-up) than if they are in active mode when the event occurs. For very slow standby frequencies, it is possible that the response time may be too long for the application to tolerate. To calculate the wake up time, the number of standby clock cycles need to be known.

Table 3 shows the worst case delay for entering active mode from standby mode.

Resource	XCore Standby Clock Delay	XCore Active Clock Delay[1]	Notes
Timer	2	2	
Channel	7+tokens (1 or 4)	2	If the a word is input, 4 tokens are required, if a character is input, only 1 token is required.
Port	5+delay_flops (0-5)	2	The port signal is passed through a flexible number of delay flops (0-5). This delay is set using set_pad_delay()

Table 3: Worst Case Wake-up Delays

1. In addition to the delay caused by clocking the XCore at the standby clock frequency, two active clock ticks are also lost when waking up and entering active mode.

4.4 Examples of wake-up delay calculation

- A timer is used to exit a 1MHz standby mode to run in active mode at 400MHz. Using Table 3, a delay of two XCore standby clocks is required to exit standby. Two 1MHz cycles corresponds to a delay of 2us before active mode is entered. Two active (400MHz) cycles are also required.

  delay = (2*1us)+(2*2.5ns) = 2.005us

- A port with a channel inputting a 32 bit word is used to exit a 10MHz standby mode to run in active mode at 500MHz. Using Table 3, a 32 bit word consists of 4 tokens, so, a delay of 11 XCore standby clocks is required to exit standby, + 2 active clocks.

  delay = (11*100ns)+(2*2ns)= 1.104us

- A port with a port delay of 2 is used to exit a 100MHz standby mode to run in active mode at 500MHz. Using Table 3, with a port delay of 2, a delay of 7 XCore standby clocks is required to exit standby, + 2 active clocks.

  delay = (7*10ns)+(2*2ns)= 74ns

5 Additional Power Saving Methods

5.1 Reducing the Switch Clock

If you are in a system with a single XCore, then you do not require the inter-core communications provided by the switch. The switch contains some configuration registers, but once the configuration has been performed the switch's clock frequency can be reduced. Up to 25mA of dynamic current can be saved in this way. See *Estimating Power Consumption For XS1-L Devices* [1] for more details.

Alternatively, if inter-core communications are required, but latency is not important, the switch's clock frequency can be reduced to, say, 100MHz for a 20mA current saving.

Code Snippet: Reducing the Switch Clock frequency

```
// Reduce the switch clock frequency from 500MHz to 1MHz
#define SWITCH_CLOCK_DIVIDER (500-1)
write_sswitch_reg(get_core_id(),
            XS1_SSWITCH_CLK_DIVIDER_NUM,
            SWITCH_CLOCK_DIVIDER);
```

5.2 Permanently Enabling Standby Mode

If an application does not require the full processing power of the XS1-L device, rather than using AEC, the XCore operating frequency may be reduced permanently so long as it exceeds the reference clock. This will mean that the current draw is more constant than when AEC is used. The XCore operating frequency can be reduced by either enabling the standby mode permanently, or by reducing the PLL output frequency (see *XS1-L Clock Frequency Control* [2]).

Code Snippet: Permanently Enabling Standby Mode

```
// Reduce XCore clock to 100MHz from 500MHz
#define STANDBY_CLOCK_DIVIDER (5-1)
#define XCORE_CTRL0_CLOCK_MASK 0x30
#define XCORE_CTRL0_ENABLE_STANDBY 0x10

unsigned xcore_ctrl0_data;

// Set standby divider
write_pswitch_reg(get_core_id(),
            XS1_PSWITCH_PLL_CLK_DIVIDER_NUM,
            STANDBY_CLOCK_DIVIDER);
// Modify the clock control bits
xcore_ctrl0_data = getps(XS1_PS_XCORE_CTRL0);
xcore_ctrl0_data &= 0xffffffff - XCORE_CTRL0_CLOCK_MASK;
xcore_ctrl0_data +=  XCORE_CTRL0_ENABLE_STANDBY;
setps(XS1_PS_XCORE_CTRL0, xcore_ctrl0_data);
```

5.3 Fully Functional Ports and Low XCore Clock

During standby, port operation is restricted when the XCore clock is slower than the reference clock. If fully functional ports are required during standby, but to save dynamic current an XCore clock frequency of below 100MHz is required, it is possible to reduce the reference clock to match the desired XCore clock. If this is done the ports will remain operational during standby mode.

This should only be done with caution however. If the reference clock is changed from 100MHz then all timer operations will require different values to represent the same time. Also the resolution of time is greatly reduced with lower reference clocks. For example, for a reference clock of 10MHz, timers only have a resolution of 100ns.

6 AEC Related Registers

Register Name	Type	No.	Description	XC Access Function
XS1_PS_XCORE_CTRL0	Processor Status Register	2	[4]=1: Enable XCore clk divider [4]=0: Disable XCore clk divider [5]: 1 divider in AEC mode [5]: 0 divider in standby Other bits should not be modified when changing the XCore clock divider mode.	setps(); getps();
XS1_PSWITCH_PLL_CLK_DIVIDER_NUM	Processor Status Register	6	[15:0] XCore clock is divided down from the system clock by this value+1 when the divider is enabled.	write_pswitch_reg(); read_pswitch_reg();
XS1_SSWITCH_CLK_DIVIDER_NUM	System Switch Register	7	[15:0] Switch clock is divided down from the system clock by this value+1.	write_sswitch_reg(); read_sswitch_reg();
XS1_SSWITCH_REF_CLK_DIVIDER_NUM	System Switch Register	8	[15:0] The reference clock is divided down from the system clock by this value+1	write_sswitch_reg(); read_sswitch_reg();

Table 4: AEC Related Registers

Code Snippet: Examples of accessing the AEC registers from XC

```
write_pswitch_reg(get_core_id(),
                  XS1_PSWITCH_PLL_CLK_DIVIDER_NUM,
                  STANDBY_CLOCK_DIVIDER);

data = read_sswitch_reg(get_core_id(), XS1_SSWITCH_CLK_DIVIDER_NUM);

setps(XS1_PS_XCORE_CTRL0, xcore_ctrl0_data);
```

7 Document History

Date	Release	Comment
2010-04-27	1.0	First release

Bibliography

[1] XMOS Limited. Power Consumption For XS1-L Devices. Website, 2010. http://www.xmos.com/published/xs1l_power.

[2] XMOS Limited. XS1-L Clock Frequency Control. Website, 2010. http://www.xmos.com/published/xs1l_clk.

Disclaimer