A-Series Support Library

This library provides support for accessing the available functionality of the XMOS A-Series devices.

Features

- ADC support.
- Sleep support.
- Watchdog timer support.

Software version and dependencies

This document pertains to version 2.0.1 of this library. It is known to work on version 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_logging (>=2.0.0)
- lib_xassert (>=2.0.0)

1 A-Series ADC API

Type	**at_adc_bits_per_sample_t**
Description	Valid bits_per_sample values (8, 16 or 32).
Values	ADC_8_BPS Samples will be truncated to 8 bits. ADC_16_BPS Samples will be placed in the MSB 12 bits of the half word. ADC_32_BPS Samples will be placed in the MSB 12 bits of the word.

Type	**at_adc_config_t**
Description	Configuration structure for ADCs:.

Fields	`char input_enable`
	An array to determine which inputs are active.
	Each non-zero input will be enabled.
	
	`at_adc_bits_per_sample_t bits_per_sample`
	Select how many bits to sample per ADC.
	
	`unsigned int samples_per_packet`
	Number of samples per packet.
	Must be >0 and <=XS1_MAX_SAMPLES_PER_PACKET.
	
	`int calibration_mode`
	When set the ADCs will sample a 0.8V reference rather than the external voltage.

Function	**at_adc_enable**
Description	Configure and enable the requested ADCs. Will also perform the calibration pulses so that the ADCs are ready to provide data. adc_enable() also checks that the configuration is valid and will raise a trap if attempting to incorrectly configure the ADCs.

Type	`void at_adc_enable(tileref periph_tile,`
	`chanend adc_chan,`
	`out port trigger_port,`
	`const_adc_config_ref_t config)`

Continued on next page

Parameters	`periph_tile`
	The identifier of the tile containing the ADCs
	
	`adc_chan` The chanend to which all ADC samples will be sent.
	
	`trigger_port`
	The port connected to the ADC trigger pin.
	
	`config` The configuration to be used.

Returns	ADC_OK on success and one of the return codes in adc_return_t on an error.

Function	**at_adc_disable_all**
Description	Disable all of the ADCs.
Type	`void` `at_adc_disable_all(tileref periph_tile)`

Function	**at_adc_trigger**
Description	Causes the ADC to take one sample. This function is intended to be used with `adc_read()`. If used with `adc_read_packet()` then this function must be called enough times to ensure that an entire data packet will be available before the `adc_read_packet()` is called.
Type	`void` `at_adc_trigger(out port trigger_port)`
Parameters	`trigger_port` The port connected to the ADC trigger pin.

Function	**at_adc_trigger_packet**
Description	Trigger the ADC enough times to complete a packet.

Type	`void`
	`at_adc_trigger_packet(out port trigger_port,`
	`const_adc_config_ref_t config)`

Parameters	`trigger_port` The port connected to the ADC trigger pin. `config` The ADC configuration.

Function	at_adc_read
Description	A selectable function to read an ADC sample from the chanend. Any control tokens due to packetization will be discarded silently. Note that the adc_trigger function must have been called before this function will return any data. Note that the configuration must be the same as that used when enabling the ADCs.

Type	`void at_adc_read(chanend adc_chan,`
	`const_adc_config_ref_t config,`
	`unsigned int &data)`

Parameters	`adc_chan` The chanend to which all ADC samples will be sent. `config` The ADC configuration. `data` The word to place the data in.

Function	at_adc_read_packet
Description	A selectable function to read a packet of ADC samples from the chanend. Note that the adc_trigger_packet function must have been called before this function will return any data. Note that the configuration must be the same as that used when enabling the ADCs.

Type	`void at_adc_read_packet(chanend adc_chan,`
	`const_adc_config_ref_t config,`
	`unsigned int data[])`

Parameters	`adc_chan` The chanend to which all ADC samples will be sent. `config` The ADC configuration. `data` The buffer to place the returned data in. Each sample will be placed in a separate word. The buffer must be big enough to store all the data that will be read (samples_per_packet words).

2 A-Series Sleep API

Type	**at_wake_sources_t**
Description	Enumerated type containing possible wake sources from sleep mode Each source type can be enabled or disabled. RTC and WAKE_PIN_x may be used together however, WAKE_PIN_LOW or HIGH are mutually exclusive, i.e. enabling wake on WAKE_PIN_LOW will disable WAKE_PIN_HIGH

Values	`RTC` Enable wake from RTC.
	
	`WAKE_PIN_LOW`
	Wake when wake pin is low.
	
	`WAKE_PIN_HIGH`
	Wake when wake pin is high.

Macro	**at_pm_memory_read**
Description	Reads sleep memory and copies to array/structure up to 128B.
Parameters	x Structure or array that sleep memory is copied too

Macro	**at_pm_memory_write**
Description	Reads sleep memory and copies to array/structure up to 128B.
Parameters	x Structure or array that is copied into sleep memory

Function	**at_pm_memory_is_valid**
Description	Function that tests to see if the deep sleep memory contents are valid. Use before reading sleep memory to see if it has been previously initialised. Note that the chip initialises this to zero on reset.
Type	`char at_pm_memory_is_valid(void)`
Returns	boolean result. 1 = Valid, 0 = Invalid.

Function	at_pm_memory_validate
Description	Function that sets the validity of the sleep memory to valid Use only after a write to sleep memory contents. Note that it defaults to invalid on reset.
Type	`void at_pm_memory_validate(void)`

Function	at_pm_memory_invalidate
Description	Function that sets the validity of the sleep memory to invalid Note that it defaults to invalid after power-on reset.
Type	`void at_pm_memory_invalidate(void)`

Function	at_pm_enable_wake_source
Description	Function that enables the chip to be woken by specific sources Each wake source type can be enabled or disabled. RTC and WAKE_PIN_x may be used together however, WAKE_PIN_LOW or HIGH are mutually exclusive i.e. enabling wake on WAKE_PIN_LOW will disable WAKE_PIN_HIGH and vice versa. A single wake source can only be passed at a time. To enable multiple sources, please call the function multiple times for each wake source.
Type	`void` `at_pm_enable_wake_source(at_wake_sources_t wake_source)`
Parameters	`wake_source` enumerated type at_wake_sources_t specifying wake source to enable

Function	at_pm_set_wake_time
Description	Function that sets the wake time in milliseconds, measured by the RTC clock. It is recommended to reset the RTC before setting the wake time to avoid issues with overflow if the application has been running for some time before. The time may be up to about 4E6 seconds from reset, or approx 48 days before overflow occurs.
Type	`void` `at_pm_set_wake_time(unsigned int alarm_time)`
Parameters	`alarm_time` absolute time to set alarm/wake up in milliseconds

Function	at_pm_set_min_sleep_time
Description	Function that sets the minimum time to stay asleep in milliseconds. Default time on power up is 2^16 sleep clocks, or about 2s. Note this function truncates to the value to the nearest power of 2, so is +100% - 50% accurate, due to hardware. This setting is not critical but can be used, for example, to ignore pin events until a certain time.
Type	`void` `at_pm_set_min_sleep_time(unsigned int min_sleep_time)`
Parameters	`min_sleep_time` minimum time asleep in milliseconds

Function	at_pm_sleep_now
Description	Function that instructs the chip to go to sleep immediately. Sleep is a very deep state that switches off everything except the RTC and deep sleep memory, so the application should have exited gracefully before this function is called, including all peripheral functions. This must be the last function to be called before sleep. When waking, the chip will go through the reset sequence. Use deep sleep memory to steer the application at boot time, to recover from sleep, and determine state.
Type	`void at_pm_sleep_now(void)`

Function	at_rtc_read
Description	Function that reads the RTC value. Takes the counter and scales to milliseconds. The time may be up to about 4E6 seconds from reset, or approx 48 days before overflow occurs.
Type	`unsigned int at_rtc_read(void)`
Returns	time in milliseconds.

Function	at_rtc_reset
Description	Function that clears the RTC value. Sets the time to zero.
Type	`void at_rtc_reset(void)`

3 A-Series Watchdog-Timer API

Function	at_watchdog_enable
Description	Function that enables the watchdog timer. On overflow, it reset the xCORE tile only. The analog tile is not reset. By default, the WDT is disabled by the chip.
Type	`void at_watchdog_enable(void)`

Function	at_watchdog_disable
Description	Function that disables the watchdog timer. By default, the WDT is disabled by the chip.
Type	`void at_watchdog_disable(void)`

Function	at_watchdog_set_timeout
Description	Function that sets the overflow time in milliseconds, from now. Calling this function will automatically clear the WDT and start counting from zero ms. It is a 16b counter which allows up to 65535 milliseconds, or about 65 seconds.
Type	`void` `at_watchdog_set_timeout(unsigned short milliseconds)`
Parameters	`milliseconds` watchdog overflow time in milliseconds

Function	at_watchdog_kick
Description	Function that kicks the watchdog timer, i.e. sets it counting from zero. This function should be periodically called to prevent overflow and system reset, during normal operation. It returns the current time in the watchdog timer, to allow the application to see how close it is to a reset caused by WDT overflow
Type	`unsigned short at_watchdog_kick(void)`
Returns	WDT time in milliseconds.

APPENDIX A - Known Issues

There are no known issues with this library.

APPENDIX B - A-Series support library change log

B.1 2.0.1

- Update to source code license and copyright

B.2 2.0.0

- Changes to dependencies:
  - lib_logging: Added dependency 2.0.0
  - lib_xassert: Added dependency 2.0.0