MAX32600

Firmware Developer's Guide

April 2015

maxim integrated™

Rev. 4.1, April 2015

CONTENTS

TABLES

FIGURES

1 Overview

This manual will provide the basic building blocks to develop firmware for the MAX32600 series of analog integrated microcontrollers featuring the ARM Cortex-M3 CPU. The code referenced in this manual is provided by Maxim Integrated in full source format.

2 Module Documentation

2.1 ADC

Enumerations

- enum mxc_adc_clk_mode
- enum mxc_adc_mode_t
- enum mxc_adc_range_t
- enum mxc_adc_bi_pol_t
- enum mxc_adc_avg_mode_t
- enum mxc_adc_strt_mode_t
- enum mxc_adc_pga_mux_ch_sel_t
- enum mxc_adc_pga_mux_diff_t
- enum mxc_adc_pga_gain_t
- enum mxc_adc_spst_sw_ctrl_t
- enum mxc_adc_scan_cnt_t

Functions

- void ADC_Enable (void)
- void ADC_Disable (void)
- void ADC_SetMode (mxc_adc_mode_t adc_mode, mxc_adc_avg_mode_t decimation_mode, uint32_t decimation_rate, mxc_adc_bi_pol_t bipolar_enable, mxc_adc_range_t bipolar_range)
- void ADC_SetRate (uint32_t pga_acq_cnt, uint32_t adc_acq_cnt, uint32_t pga_trk_cnt, uint32_t adc_slp_cnt)
- void ADC_SetMuxSel (mxc_adc_pga_mux_ch_sel_t mux_ch_sel, mxc_adc_pga_mux_diff_t mux_diff)
- void ADC_SetPGAMode (uint32_t pga_bypass, mxc_adc_pga_gain_t pga_gain)
- void ADC_SetScanCount (mxc_adc_scan_cnt_t scan_cnt)
- void ADC_SetScanDesc (uint8_t scan_desc_index, uint8_t mux_diff, uint8_t pga_gain, uint8_t mux_ch_sel)
- void ADC_SetStartMode (mxc_adc_strt_mode_t adc_strt_mode)
- int32_t ADC_CaptureConfig (adc_transport_t ∗transport, uint16_t ∗buf1, uint32_t buf1_samples, uint16_t ∗buf2, uint32_t buf2_samples, void(∗done_cb)(int32_t exit_status, void ∗done_arg), void ∗done_cb_arg, uint8_t stop)
- int32_t ADC_CaptureStart (adc_transport_t ∗transport)
- int32_t ADC_CaptureStop (adc_transport_t ∗transport)
- uint16_t ADC_ManualRead (void)

2.1.1 Detailed Description

This is the high level API for the analog-to-digital converter module of the MAX32600 family of ARM Cortex based embedded microcontrollers.
Refer to ADC_SetRate().

Full Rate Mode with PGA enabled:
For sample rate above 162KHz, the sample rate is adjusted by increasing the pga_trk_cnt.
$T_s = (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c} - pga\_acq\_cnt - adc\_acq\_cnt - 8$
$T_c =>$ ADC clock rate, typically 8MHz
$pga\_trk\_cnt =>$ A rounded up and down integer to determine two closest sample rates achievable to the desired.
The user will need to select which on is desired. For sample rates above 162KHz the PGA and ADC are enabled for the duration of the data collection. The total current draw is sum of the ADC and PGA maximum current.

Full Rate Burst Mode with PGA enabled:
Burst reduced effective sample rate by the decimation rate. $2^{adc\_brst\_cnt}$ samples are collected and averaged to improve the SNR of the output. Only the averaged data sample is output so the data rate is reduced by $2^{adc\_brst\_cnt}$
$T_s = 2^{adc\_brst\_cnt} \cdot (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c \cdot 2^{adc\_brst\_cnt}} - pga\_acq\_cnt - adc\_acq\_cnt - 8$

Full Rate Scan Mode with PGA enabled:
When the channel scanning is enabled, the ADC cycles through N_{scan} different input based on the 8 different scan descriptors. Similar to burst, the sample rate per channel is reduced by the number of scan channels.
$T_s = N_{scan} \cdot (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c \cdot N_{scan}} - pga\_acq\_cnt - adc\_acq\_cnt - 8$

Full Rate Scan Burst Mode with PGA enabled:
When both scanning and averaging are enabled, the ADC collects $2^{adc\_brst\_cnt}$ of each channel before moving to the next channel. Thus the maximum possible sample rate is reduced by $N_{scan} \cdot 2^{adc\_brst\_cnt}$
$T_s = N_{scan} 2^{adc\_brst\_cnt} (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c \cdot N_{scan} \cdot 2^{adc\_brst\_cnt}} - pga\_acq\_cnt - adc\_acq\_cnt - 8$

Low Power Mode with PGA enabled:
For sample rates below 162KHz the low-power mode can be utilized. The pga_trk_cnt is set to the minimum and the sample rate is adjusted by increasing the adc_slp_cnt
$T_s = (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + adc\_slp\_cnt + 40) \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - pga\_trk\_cnt - pga\_acq\_cnt - adc\_acq\_cnt - 40$

Low Power Mode Burst with PGA enabled:
The decimation filter can be used in the low power mode. The effective sample rate is a function of the decimation rate and the sleep counter. With $decimation\_mode$ being set in ADC_SetMode(), the adc_brst_cnt defines the length of the decimation filter. The sample rate is controlled by adjusting the adc_slp_cnt
$T_s = \left[2^{adc\_brst\_cnt} (pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) + adc\_slp\_cnt + 32\right] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - 2^{adc\_brst\_cnt} (pga\_trk\_cnt - pga\_acq\_cnt - adc\_acq\_cnt + 8) - 32$

Low Power Scan with PGA enabled:
When the channel scanning is enabled, the ADC cycles through N_{scan} different input based on the 8 different scan descriptors. The maximum potential sample rate per channel is reduced by the number of scan channels.

$T_s = [N_{scan}\,(pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) + adc\_slp\_cnt + 32] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - N_{scan}\,(pga\_trk\_cnt - pga\_acq\_cnt - adc\_acq\_cnt + 8) - 32$

Low Power Scan Burst with PGA enabled:
When both scanning and averaging are enabled, the ADC collects $2^{adc\_brst\_cnt}$ of each channel before moving to the next channel. Thus the maximum possible sample rate is reduced by $N_{scan} \cdot 2^{adc\_brst\_cnt}$
$T_s = \left[2^{adc\_brst\_cnt} N_{scan}\,(pga\_trk\_cnt + pga\_acq\_cnt + adc\_acq\_cnt + 8) + adc\_slp\_cnt + 32\right] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - N_{scan} \cdot 2^{adc\_brst\_cnt}\,(pga\_trk\_cnt - pga\_acq\_cnt - adc\_acq\_cnt + 8) - 32$

Full Rate Mode with PGA bypass:
For sample rates above 333KHz, the sample rate is adjusted by increasing the pga_trk_cnt

$T_s = (pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c} - adc\_acq\_cnt - 7$
T_c => ADC clock rate, typically 8MHz. For example, if pga_trk_cnt is 9 and adc_acq_cnt is 0, $T_s = 16 \cdot T_c$. Which results in the target maximum sampling rate of 500Ksps.

Full Rate Burst Mode with PGA bypass:
Burst reduced effective sample rate by decimation rate. $2^{adc\_brst\_cnt}$ samples are collected and averaged to improve the SNR of the output. Only the averaged data sample is output so the data rate is reduced by $2^{adc\_brst\_cnt}$.

$T_s = 2^{adc\_brst\_cnt}(pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{2^{adc\_brst\_cnt} T_c} - adc\_acq\_cnt - 7$

Full Rate Scan Mode with PGA bypass:
For sample rates above 333KHz, the sample rate is adjusted by increasing the pga_trk_cnt.

$T_s = (pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{T_c} - adc\_acq\_cnt - 7$

Full Rate Burst Mode with PGA bypass:
Burst reduced effective sample rate by the decimation rate. $2^{adc\_brst\_cnt}$ samples are collected and averaged to improve the SNR of the output. Only the averaged data sample is output so the data rate is reduced by $2^{adc\_brst\_cnt}$.

$T_s = 2^{adc\_brst\_cnt} \cdot (pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{2^{adc\_brst\_cnt} T_c} - adc\_acq\_cnt - 7$

Full Rate Scan Mode with PGA bypass:
Scan mode divides the sample rate across N_{scan} channels without additional overhead.

$T_s = N_{scan} \cdot (pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{N_{scan} T_c} - adc\_acq\_cnt - 7$

Full Rate Scan and Bypass Mode with PGA bypass:
Burst adds an additional reduction in sample rate.

$T_s = N_{scan} 2^{adc\_brst\_cnt}(pga\_trk\_cnt + adc\_acq\_cnt + 7) \cdot T_c$
$pga\_trk\_cnt = \frac{T_{s\_desired}}{N_{scan} 2^{adc\_brst\_cnt} T_c} - adc\_acq\_cnt - 7$

Low Power Mode with PGA bypass:
The PGA wake and track counters are not used if the PGA is bypassed. Thus the sample rate calculation is different.

$T_s = (adc\_acq\_cnt + adc\_slp\_cnt + 24) \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - adc\_acq\_cnt - 24$
In PGA bypass mode, the low power mode can be used for $Ts <= 3\mu$s.

Low Power Mode and Decimation Filter with PGA bypass:
The decimation filter can be used in the low power mode with the PGA in bypass mode. The effective sample rate is a function of the decimation rate and hte sleep counter. With $decimation\_mode$ being set to 1 in ADC_SetMode(), the adc_brst_cnt defines the length of the decimation filter. The sample rate is controlled by adjusting the adc_slp_cnt
$T_s = \left[\left(2^{adc\_brst\_cnt} - 1\right)(pga\_trk\_cnt + adc\_acq\_cnt + 7) + adc\_slp\_cnt + adc\_acq\_cnt + 24\right] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - \left(2^{adc\_brst\_cnt} - 1\right)(pga\_trk\_cnt + adc\_acq\_cnt + 7) - adc\_acq\_cnt - 24$

Low Power Scan Mode with PGA bypass:
Low power scan mode is like low power burst except a different channel is sampled each time.

$T_s = [(N_{scan} - 1)(pga\_trk\_cnt + adc\_acq\_cnt + 7) + adc\_slp\_cnt + adc\_acq\_cnt + 24] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - (N_{scan} - 1)(pga\_trk\_cnt + adc\_acq\_cnt + 7) - adc\_acq\_cnt - 24$

Low Power Scan Mode with Burst and PGA bypass:
In this mode, a burst of each channel is take.

$T_s = \left[\left(2^{adc\_brst\_cnt} N_{scan} - 1\right)(pga\_trk\_cnt + adc\_acq\_cnt + 7) + adc\_slp\_cnt + adc\_acq\_cnt + 24\right] \cdot T_c$
$adc\_slp\_cnt = \frac{T_{s\_desired}}{T_c} - \left(2^{adc\_brst\_cnt} N_{scan} - 1\right)(pga\_trk\_cnt + adc\_acq\_cnt + 7) - adc\_acq\_cnt - 24$

2.1.2 Enumeration Type Documentation

enum mxc_adc_avg_mode_t

Defines Decimation Filter Modes.

Enumerator

MXC_E_ADC_AVG_MODE_FILTER_BYPASS Decimation Filter ByPassed
MXC_E_ADC_AVG_MODE_FILTER_OUTPUT Output Average Only
MXC_E_ADC_AVG_MODE_FILTER_OUTPUT_RAW Output Average and Raw Data (Test Mode Only)

enum mxc_adc_bi_pol_t

Defines ADC Range Control.

Enumerator

MXC_E_ADC_BI_POL_UNIPOLAR Uni-polar operation (0 -> Vref)
MXC_E_ADC_BI_POL_BIPOLAR Bi-polar operation see ADC Range Control

enum mxc_adc_clk_mode

Defines ADC Clock Divider Options, Must Run at 8MHz.

Enumerator

MXC_E_ADC_CLK_MODE_FULL	Do not divide input clock
MXC_E_ADC_CLK_MODE_HALF	Divide input clock by 2
MXC_E_ADC_CLK_MODE_THIRD	Divide input clock by 3
MXC_E_ADC_CLK_MODE_FOURTH	Divide input clock by 4
MXC_E_ADC_CLK_MODE_SIX	Divide input clock by 6
MXC_E_ADC_CLK_MODE_EIGHTH	Divide input clock by 8
MXC_E_ADC_CLK_MODE_TWELVE	Divide input clock by 12

enum mxc_adc_mode_t

Defines ADC Modes.

Enumerator

MXC_E_ADC_MODE_SMPLCNT_FULL_RATE	Single Mode Full Rate
MXC_E_ADC_MODE_SMPLCNT_LOW_POWER	Single Mode Low Power
MXC_E_ADC_MODE_CONTINUOUS_FULL_RATE	Continuous Mode Full Rate
MXC_E_ADC_MODE_CONTINUOUS_LOW_POWER	Continuous Mode Low Power
MXC_E_ADC_MODE_SMPLCNT_SCAN_FULL_RATE	Single Mode Full Rate with Scan Enabled
MXC_E_ADC_MODE_SMPLCNT_SCAN_LOW_POWER	Single Mode Low Power with Scan Enabled
MXC_E_ADC_MODE_CONTINUOUS_SCAN_FULL_RATE	Continuous Mode Full Rate with Scan Enabled
MXC_E_ADC_MODE_CONTINUOUS_SCAN_LOW_POWER	Continuous Mode Low Power with Scan Enabled

enum mxc_adc_pga_gain_t

Defines the PGA Gain Options.

Enumerator

MXC_E_ADC_PGA_GAIN_1	PGA Gain = 1
MXC_E_ADC_PGA_GAIN_2	PGA Gain = 2
MXC_E_ADC_PGA_GAIN_4	PGA Gain = 4
MXC_E_ADC_PGA_GAIN_8	PGA Gain = 8

enum mxc_adc_pga_mux_ch_sel_t

Defines Mux Channel Select for the Positive Input to the ADC.

Enumerator

MXC_E_ADC_PGA_MUX_CH_SEL_AIN0	Single Mode Input AIN0+; Diff Mode AIN0+/AIN8-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN1	Single Mode Input AIN1+; Diff Mode AIN1+/AIN9-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN2	Single Mode Input AIN2+; Diff Mode AIN2+/AIN10-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN3	Single Mode Input AIN3+; Diff Mode AIN3+/AIN11-

MXC_E_ADC_PGA_MUX_CH_SEL_AIN4	Single Mode Input AIN4+; Diff Mode AIN4+/AIN12-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN5	Single Mode Input AIN5+; Diff Mode AIN5+/AIN13-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN6	Single Mode Input AIN6+; Diff Mode AIN6+/AIN14-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN7	Single Mode Input AIN7+; Diff Mode AIN7+/AIN15-
MXC_E_ADC_PGA_MUX_CH_SEL_AIN8	Single Mode Input AIN8+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN9	Single Mode Input AIN9+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN10	Single Mode Input AIN10+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN11	Single Mode Input AIN11+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN12	Single Mode Input AIN12+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN13	Single Mode Input AIN13+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN14	Single Mode Input AIN14+
MXC_E_ADC_PGA_MUX_CH_SEL_AIN15	Single Mode Input AIN15+
MXC_E_ADC_PGA_MUX_CH_SEL_VSSADC	Positive Input VSSADC
MXC_E_ADC_PGA_MUX_CH_SEL_TMON_R	Positive Input TMON_R
MXC_E_ADC_PGA_MUX_CH_SEL_VDDA4	Positive Input VDDA/4
MXC_E_ADC_PGA_MUX_CH_SEL_PWRMON_TST	Positive Input PWRMAN_TST
MXC_E_ADC_PGA_MUX_CH_SEL_AIN0DIV	Positive Input Ain0Div
MXC_E_ADC_PGA_MUX_CH_SEL_OUTA	Positive Input OpAmp OUTA
MXC_E_ADC_PGA_MUX_CH_SEL_OUTB	Positive Input OpAmp OUTB
MXC_E_ADC_PGA_MUX_CH_SEL_OUTC	Positive Input OpAmp OUTC
MXC_E_ADC_PGA_MUX_CH_SEL_OUTD	Positive Input OpAmp OUTD
MXC_E_ADC_PGA_MUX_CH_SEL_INAPLUS	Positive INA+
MXC_E_ADC_PGA_MUX_CH_SEL_SNO_OR	Positive SNO_or
MXC_E_ADC_PGA_MUX_CH_SEL_SCM_OR	Positive SCM_or
MXC_E_ADC_PGA_MUX_CH_SEL_TPROBE_SENSE	Positive TPROBE_sense
MXC_E_ADC_PGA_MUX_CH_SEL_VREFDAC	Positive VREFDAC
MXC_E_ADC_PGA_MUX_CH_SEL_VREFADJ	Positive VREFADJ
MXC_E_ADC_PGA_MUX_CH_SEL_VDD3XTAL	Positive Vdd3xtal

enum mxc_adc_pga_mux_diff_t

Decoded with the MUX Channel Select to enable Differential Mode Input to the ADC.

Enumerator

MXC_E_ADC_PGA_MUX_DIFF_DISABLE	Differential Mode Disabled
MXC_E_ADC_PGA_MUX_DIFF_ENABLE	Differential Mode Enabled

enum mxc_adc_range_t

Defines ADC Range Control.

Enumerator

MXC_E_ADC_RANGE_HALF	Bi-polar Operation (-Vref/2 -> Vref/2)
MXC_E_ADC_RANGE_FULL	Bi-polar Operation (-Vref -> Vref)

enum mxc_adc_scan_cnt_t

Defines the number of channels to scan when Scan Mode is enabled.

Enumerator

MXC_E_ADC_SCAN_CNT_1	Number of Channels to Scan = 1
MXC_E_ADC_SCAN_CNT_2	Number of Channels to Scan = 2
MXC_E_ADC_SCAN_CNT_3	Number of Channels to Scan = 3
MXC_E_ADC_SCAN_CNT_4	Number of Channels to Scan = 4
MXC_E_ADC_SCAN_CNT_5	Number of Channels to Scan = 5
MXC_E_ADC_SCAN_CNT_6	Number of Channels to Scan = 6
MXC_E_ADC_SCAN_CNT_7	Number of Channels to Scan = 7
MXC_E_ADC_SCAN_CNT_8	Number of Channels to Scan = 8

enum mxc_adc_spst_sw_ctrl_t

Defines the Switch Control Mode.

Enumerator

MXC_E_ADC_SPST_SW_CTRL_SOFTWARE	Switch Control Mode = Software
MXC_E_ADC_SPST_SW_CTRL_PULSETRAIN	Switch Control Mode = Pulse Train

enum mxc_adc_strt_mode_t

Defines ADc StartMode Modes.

Enumerator

MXC_E_ADC_STRT_MODE_SOFTWARE	StarMode via Software
MXC_E_ADC_STRT_MODE_PULSETRAIN	StarMode via PulseTrain

2.1.3 Function Documentation

int32_t ADC_CaptureConfig (adc_transport_t ∗ *transport*, uint16_t ∗ *buf1*, uint32_t *buf1_samples*, uint16_t ∗ *buf2*, uint32_t *buf2_samples*, void(∗)(int32_t exit_status, void ∗done_arg) *done_cb*, void ∗ *done_cb_arg*, uint8_t *stop*)

This function will setup a single and re-usable capture object for the ADC input port. For the most efficient usage, the buffer sizes should be a multiple of 3/4 ADC FIFO size.

Parameters

transport	pointer to state structure. This API will populate the required fields of the struct, no initialization necessary.
buf1	pointer to first RAM allocated capture buffer location, needs to be at least samples∗2 bytes long.
buf1_samples	number of samples to place in buf1
buf2	(OPTIONAL) pointer to second buffer for double-buffer style capture
buf2_samples	(OPTIONAL) number of samples to place in buf2
done_cb	(OPTIONAL) pointer to a callback function to be called by ISR when one buffer is full.
done_cb_arg	
stop	stop the capture when all buffers are full or continue forever filling each buffer until ADC_CaptureStop() is called.

Returns

0 on success

int32_t ADC_CaptureStart (adc_transport_t ∗ *transport*)

Start a capture process using a previously allocated handle from ADC_CaptureConfig().

Parameters

transport	Return handle from a call to ADC_CaptureConfig().

Returns

0 => Success. Non zero => error condition.

int32_t ADC_CaptureStop (adc_transport_t ∗ *transport*)

Stop a running analog capture that was started without the stop bit set. Capture will complete to the next end of buffer condition and callback function will be called with 'stop_bit' set.

Parameters

transport	Return handle from a call to ADC_CaptureConfig().

Returns

0 => Success. Non zero => error condition.

uint16_t ADC_ManualRead (void)

Trigger and read a single sample from the ADC. This function will place the ADC into 'full power' mode, trigger the capture and return the collected data.

Returns

16-bit data value.

void ADC_SetMode (mxc_adc_mode_t *adc_mode,* mxc_adc_avg_mode_t *decimation_mode,* uint32_t *decimation_rate,* mxc_adc_bi_pol_t *bipolar_enable,* mxc_adc_range_t *bipolar_range*)

Setup ADC Configuration.

Parameters

adc_mode	ADC Operation Mode
decima-tion_mode	Turns on/off decimation averaging filter
decima-tion_rate	Decimation Filter rate if enabled

bipolar_enable	ADC bipolar operation control
bipolar_range	ADC Range Control when in bi-polar mode of operation

void ADC_SetMuxSel (mxc_adc_pga_mux_ch_sel_t *mux_ch_sel*, mxc_adc_pga_mux_diff_t *mux_diff*)

Setup ADC Input Mux.

Parameters

mux_ch_sel	Selection of Input Mux to ADC. Decoded in concert with mux_diff
mux_diff	Select differential or single ended input mode

void ADC_SetPGAMode (uint32_t *pga_bypass*, mxc_adc_pga_gain_t *pga_gain*)

Setup PGA Configuration.

Parameters

pga_bypass	When set to 1, the PGA is in Bypass Mode
pga_gain	When the PGA is enabled, sets the PGA gain

void ADC_SetRate (uint32_t *pga_acq_cnt*, uint32_t *adc_acq_cnt*, uint32_t *pga_trk_cnt*, uint32_t *adc_slp_cnt*)

Set ADC Sample Rate.

Parameters

pga_acq_cnt	PGA Acquisition Count
adc_acq_cnt	ADC Acquisition Count
pga_trk_cnt	PGA Tracking Count
adc_slp_cnt	ADC Sleep Count

void ADC_SetScanCount (mxc_adc_scan_cnt_t *scan_cnt*)

Setup ADC Scan Count.

Parameters

scan_cnt	Number of channels to scan, see enumeration for correct values.

void ADC_SetScanDesc (uint8_t *scan_desc_index*, uint8_t *mux_diff*, uint8_t *pga_gain*, uint8_t *mux_ch_sel*)

Setup Scan Mode Channel Configuration.

Parameters

scan_desc_index	Specifies the number of the scan channel being configured
mux_diff	Selects differential or single ended input mode for the selected channel
pga_gain	Selects PGA input gain for the selected channel
mux_ch_sel	Selects the input mux for the selected channel

void ADC_SetStartMode (mxc_adc_strt_mode_t *adc_strt_mode*)

Setup ADC Start Mode.

Parameters

adc_strt_mode	Data Collection Start Mode: 'software' or 'pulse train' control

2.2 AES

Macros

- #define MXC_AES_DATA_LEN (128 / 8)
- #define MXC_AES_KEY_128_LEN (128 / 8)
- #define AES_ECBEncrypt(ptxt, ctxt, mode) AES_ECBOp(ptxt, ctxt, mode, MXC_E_AES_ENCRYPT)
- #define AES_ECBDecrypt(ctxt, ptxt, mode) AES_ECBOp(ctxt, ptxt, mode, MXC_E_AES_DECRYPT)
- #define AES_ECBEncryptAsync(ptxt, mode, callback) AES_AsyncSetup(ptxt, mode, MXC_E_AES_ENCRYPT_A callback)
- #define AES_ECBDecryptAsync(ctxt, mode, callback) AES_AsyncSetup(ctxt, mode, MXC_E_AES_DECRYPT_A callback)

Enumerations

- enum mxc_aes_mode_t
- enum mxc_aes_dir_t
- enum mxc_aes_ret_t

Functions

- mxc_aes_ret_t AES_SetKey (const uint8_t ∗key, mxc_aes_mode_t mode)
- mxc_aes_ret_t AES_ECBOp (const uint8_t ∗in, uint8_t ∗out, mxc_aes_mode_t mode, mxc_aes_dir_t dir)
- mxc_aes_ret_t AES_GetOutput (uint8_t ∗out)
- mxc_aes_ret_t AES_AsyncSetup (const uint8_t ∗in, mxc_aes_mode_t mode, mxc_aes_dir_t dir, void(∗cb)(void))

2.2.1 Detailed Description

This is the high level API for the MAX32600 AES encryption engine.

– Key/data format in memory –

These functions expect that key and plain/ciphertext will be stored as a byte array in LSB .. MSB format.

As an example, given the key 0x139a35422f1d61de3c91787fe0507afd, the proper storage order is:

uint8_t key[16] = {0xfd, 0x7a, 0x50, 0xe0, 0x7f, 0x78, 0x91, 0x3c, 0xde, 0x61, 0x1d, 0x2f, 0x42, 0x35, 0x9a, 0x13};

This is the same order expected by the underlying hardware.

2.2.2 Macro Definition Documentation

#define AES_ECBDecrypt(***ctxt, ptxt, mode*** **) AES_ECBOp(ctxt, ptxt, mode, MXC_E_AES_DECRYPT)**

Decrypt a block of ciphertext with the loaded AES key, blocks until complete.

Parameters

ctxt	Pointer to ciphertext output array (always 16 bytes)
ptxt	Pointer to plaintext input array (always 16 bytes)
mode	Selects key length, valid modes found in mxc_aes_mode_t

#define AES_ECBDecryptAsync(***ctxt, mode, callback*** **) AES_AsyncSetup(ctxt, mode, MXC_E_AES_DECRYPT_ASYNC, callback)**

Starts encryption of a block, enables interrupt, and returns immediately. Use AES_GetOuput() to retrieve result after interrupt fires.

Parameters

ctxt	Pointer to ciphertext output array (always 16 bytes)
mode	Selects key length, valid modes found in mxc_aes_mode_t
callback	Function to be called when AES operation complete, prototype is void callback(void)

#define AES_ECBEncrypt(***ptxt, ctxt, mode*** **) AES_ECBOp(ptxt, ctxt, mode, MXC_E_AES_ENCRYPT)**

Encrypt a block of plaintext with the loaded AES key, blocks until complete.

Parameters

ptxt	Pointer to plaintext input array (always 16 bytes)
ctxt	Pointer to ciphertext output array (always 16 bytes)
mode	Selects key length, valid modes found in mxc_aes_mode_t

#define AES_ECBEncryptAsync(***ptxt, mode, callback*** **) AES_AsyncSetup(ptxt, mode, MXC_E_AES_ENCRYPT_ASYNC, callback)**

Starts encryption of a block, enables interrupt, and returns immediately. Use AES_GetOuput() to retrieve result after interrupt fires.

Parameters

ptxt	Pointer to plaintext input array (always 16 bytes)
mode	Selects key length, valid modes found in mxc_aes_mode_t
callback	Function to be called when AES operation complete, prototype is void callback(void)

2.2.3 Enumeration Type Documentation

enum mxc_aes_dir_t

Direction select.

Enumerator

MXC_E_AES_ENCRYPT	Encrypt (blocking)
MXC_E_AES_ENCRYPT_ASYNC	Encrypt (interrupt-driven)
MXC_E_AES_DECRYPT	Decrypt (blocking)
MXC_E_AES_DECRYPT_ASYNC	Decrypt (interrupt-driven)

enum mxc_aes_mode_t

Key size selection (bits)

Enumerator

MXC_E_AES_MODE_128	128-bit key
MXC_E_AES_MODE_192	192-bit key
MXC_E_AES_MODE_256	256-bit key

enum mxc_aes_ret_t

Standardized return codes for the AES module.

Enumerator

MXC_E_AES_ERR	Error
MXC_E_AES_OK	No error
MXC_E_AES_BUSY	Engine busy, try again later

2.2.4 Function Documentation

mxc_aes_ret_t AES_ECBOp (const uint8_t ∗ *in*, uint8_t ∗ *out*, mxc_aes_mode_t *mode*, mxc_aes_dir_t *dir*)

Encrypt/decrypt an input block with the loaded AES key.

Parameters

in	Pointer to input array (always 16 bytes)
out	Pointer to output array (always 16 bytes)

mxc_aes_ret_t AES_GetOutput (uint8_t ∗ *out*)

Read the AES output memory, used for asynchronous encryption, and clears interrupt flag.

Parameters

out	Pointer to output array (always 16 bytes)

mxc_aes_ret_t AES_SetKey (const uint8_t ∗ *key*, mxc_aes_mode_t *mode*)

Configure AES block with keying material.

Parameters

key	128, 192, or 256 bit keying material
mode	Selects key length, valid modes found in mxc_aes_mode_t

2.3 AFE

Enumerations

- enum mxc_opamp_pos_in_t
- enum mxc_opamp_neg_in_t
- enum mxc_opamp_neg_pad_t
- enum mxc_lpc_pos_in_t
- enum mxc_lpc_neg_in_t
- enum mxc_opamp_mode_t
- enum mxc_afe_in_mode_opamp_t
- enum mxc_afe_led_cfg_port_t
- enum mxc_afe_en_wud_comp_t
- enum mxc_afe_in_mode_comp_t
- enum mxc_afe_bias_mode_comp_t
- enum mxc_afe_tmon_current_t
- enum mxc_afe_ref_volt_sel_t
- enum mxc_afe_dac_ref_t
- enum mxc_afe_hyst_comp_t
- enum mxc_afe_scm_or_sel_t
- enum mxc_afe_sno_or_sel_t
- enum mxc_afe_dacx_sel_t
- enum mxc_afe_close_spst_t
- enum mxc_afe_gnd_sel_opamp_t
- enum mxc_afe_p_in_sel_opamp_t
- enum mxc_afe_n_in_sel_opamp_t
- enum mxc_afe_dac_sel_t
- enum mxc_afe_npad_sel_t
- enum mxc_afe_pos_in_sel_comp_t
- enum mxc_afe_neg_in_sel_comp_t

Functions

- void AFE_OpAmpSetup (uint8_t opamp_index, mxc_opamp_mode_t mode, mxc_opamp_pos_in_t pos_input, mxc_opamp_neg_in_t neg_input, mxc_afe_in_mode_opamp_t opamp_inmode)
- void AFE_OpAmpEnable (uint8_t opamp_index)
- void AFE_OpAmpDisable (uint8_t opamp_index)
- void AFE_OpAmpSetupInt (uint8_t opamp_index, void(∗intr_cb)(void ∗arg), void ∗cb_arg)
- void AFE_LPCSetup (uint8_t lpc_index, mxc_lpc_pos_in_t pos_input, mxc_lpc_neg_in_t neg_input, mxc_afe_in_mode_comp_t cmp_inmode)
- void AFE_LPCEnable (uint8_t lpc_index)
- void AFE_LPCDisable (uint8_t lpc_index)
- void AFE_LPCConfig (uint8_t lpc_index, mxc_afe_bias_mode_comp_t cmp_bias, mxc_afe_hyst_comp_t hysteresis, uint8_t tp_polarity)
- void AFE_LPCSetupInt (uint8_t lpc_index, void(∗intr_cb)(void ∗arg), void ∗cb_arg)
- void AFE_ADCVRefEnable (mxc_afe_ref_volt_sel_t adc_refsel)

- void AFE_ADCVRefDisable (uint8_t fast_power_down)
- void AFE_DACVRefEnable (mxc_afe_ref_volt_sel_t dac_refsel, mxc_afe_dac_ref_t dacsel)
- void AFE_DACVRefDisable (uint8_t fast_power_down)
- void AFE_VRefExtBandgapSetup (uint8_t v1extadj)
- void AFE_SetSwitchState (uint8_t switch_index, mxc_afe_close_spst_t state)
- void AFE_SetSwitchMode (uint8_t switch_index, mxc_adc_spst_sw_ctrl_t switch_mode)
- void AFE_LEDConfig (uint8_t port_index, mxc_afe_led_cfg_port_t led_cfg)
- void AFE_GndSwitchSet (uint8_t opamp_index, mxc_afe_gnd_sel_opamp_t state)
- void AFE_NpadSetup (uint8_t opamp_index, mxc_opamp_neg_pad_t npad_select)

2.3.1 Detailed Description

This is the high level API for the analog front end module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.3.2 Enumeration Type Documentation

enum mxc_afe_bias_mode_comp_t

LPC Bias.

Enumerator

MXC_E_AFE_BIAS_MODE_COMP_0	BIAS 0.52uA Delay 4.0us
MXC_E_AFE_BIAS_MODE_COMP_1	BIAS 1.4uA Delay 1.7us
MXC_E_AFE_BIAS_MODE_COMP_2	BIAS 2.8uA Delay 1.1us
MXC_E_AFE_BIAS_MODE_COMP_3	BIAS 5.1uA Delay 0.7us

enum mxc_afe_close_spst_t

Selection for state of Switch.

Enumerator

MXC_E_AFE_CLOSE_SPST_SWITCH_OPEN	Switch is OPEN
MXC_E_AFE_CLOSE_SPST_SWITCH_CLOSE	Switch is CLOSED

enum mxc_afe_dac_ref_t

Selection for DAC VOltage Reference, REFADC or REFDAC.

Enumerator

MXC_E_AFE_DAC_REF_REFADC	DAC Voltage Reference = REFADC
MXC_E_AFE_DAC_REF_REFDAC	DAC Voltage Reference = REFDAC

enum mxc_afe_dac_sel_t

MUX Selection for DAC_sel.

Enumerator

MXC_E_AFE_DAC_SEL_DAC0	DAC_or = DAC0
MXC_E_AFE_DAC_SEL_DAC1	DAC_or = DAC1
MXC_E_AFE_DAC_SEL_DAC2P	DAC_or = DAC2P

MXC_E_AFE_DAC_SEL_DAC3P DAC_or = DAC3P

enum mxc_afe_dacx_sel_t

Selection for MUX DACx_sel.

Enumerator

MXC_E_AFE_DACX_SEL_P dacx = DACOP
MXC_E_AFE_DACX_SEL_N dacx = DACON

enum mxc_afe_en_wud_comp_t

Setup of Wake Up Detector for LPCs.

Enumerator

MXC_E_AFE_EN_WUD_COMP_IDLE IDLE
MXC_E_AFE_EN_WUD_COMP_FALLING_EDGE Activate WUD for falling edges
MXC_E_AFE_EN_WUD_COMP_RISING_EDGE Activate WUD for rising edges

enum mxc_afe_gnd_sel_opamp_t

Switch to Connect Positive Pad to GND.

Enumerator

MXC_E_AFE_GND_SEL_OPAMP_SWITCH_OPEN Positive Pad GND Switch OPEN
MXC_E_AFE_GND_SEL_OPAMP_SWITCH_CLOSED Positive Pad GND Switch CLOSED

enum mxc_afe_hyst_comp_t

Selection for LPC Hysteresis.

Enumerator

MXC_E_AFE_HYST_COMP_0 LPC Hysteresis = 0 mV
MXC_E_AFE_HYST_COMP_1 LPC Hysteresis = 7.5 mV
MXC_E_AFE_HYST_COMP_2 LPC Hysteresis = 15 mV
MXC_E_AFE_HYST_COMP_3 LPC Hysteresis = 30 mV

enum mxc_afe_in_mode_comp_t

LPC InMode.

Enumerator

MXC_E_AFE_IN_MODE_COMP_NCH_PCH InMode: both Nch and Pch
MXC_E_AFE_IN_MODE_COMP_NCH InMode: only Nch
MXC_E_AFE_IN_MODE_COMP_PCH InMode: only Pch

enum mxc_afe_in_mode_opamp_t

OpAmp InMode.

Enumerator

MXC_E_AFE_IN_MODE_OPAMP_NCH_PCH InMode: both Nch and Pch

MXC_E_AFE_IN_MODE_OPAMP_NCH InMode: only Nch
MXC_E_AFE_IN_MODE_OPAMP_PCH InMode: only Pch

enum mxc_afe_led_cfg_port_t

Defines Configure Options for the LED Ports.

Enumerator

MXC_E_AFE_LED_CFG_PORT_OPAMP_A_C LED Sink Port 0 with OpAmp A, LED Sink Port 1 with OpAmp C
MXC_E_AFE_LED_CFG_PORT_OPAMP_B_D LED Sink Port 0 with OpAmp B, LED Sink Port 1 with OpAmp D
MXC_E_AFE_LED_CFG_PORT_DISABLED Disable LED Sink Port 0,Disable LED Sink Port 1

enum mxc_afe_n_in_sel_opamp_t

MUX Selection for OpNsel.

Enumerator

MXC_E_AFE_N_IN_SEL_OPAMP_INMINUS OpNsel = INx-
MXC_E_AFE_N_IN_SEL_OPAMP_OUT OpNsel = OUTx
MXC_E_AFE_N_IN_SEL_OPAMP_SCM_OR OpNsel = SCM_or
MXC_E_AFE_N_IN_SEL_OPAMP_SCM_OR_AND_INMINUS OpNsel = SCM_or also output on INx-

enum mxc_afe_neg_in_sel_comp_t

MUX Selection for CmpNSel.

Enumerator

MXC_E_AFE_NEG_IN_SEL_COMP_INMINUS CmpNSel = INx-
MXC_E_AFE_NEG_IN_SEL_COMP_SNO CmpNSel = SNO
MXC_E_AFE_NEG_IN_SEL_COMP_DAC0 CmpNSel = dac0
MXC_E_AFE_NEG_IN_SEL_COMP_DAC2P CmpNSel = DAC2P
MXC_E_AFE_NEG_IN_SEL_COMP_LED_OBS_PORT CmpNSel = LED Observation Port

MXC_E_AFE_NEG_IN_SEL_COMP_DAC0_AND_INMINUS CmpNSel = dac0 also output on INx-
MXC_E_AFE_NEG_IN_SEL_COMP_DAC2P_AND_INMINUS CmpNSel = DAC2 also output on INx-
MXC_E_AFE_NEG_IN_SEL_COMP_DAC2P_AND_SNO CmpNSel = DAC2 also output on SNO

enum mxc_afe_npad_sel_t

MUX Selection for NPAD_sel.

Enumerator

MXC_E_AFE_NPAD_SEL_HIZ NPAD_Sel = HIZ
MXC_E_AFE_NPAD_SEL_LED_OBS_PORT NPAD_Sel = LED Observe Port
MXC_E_AFE_NPAD_SEL_DAC_OR NPAD_Sel = DAC_or

MXC_E_AFE_NPAD_SEL_DAC_OR_AND_LED_OBS_PORT NPAD_Sel = DAC_or and LED Observe Port

enum mxc_afe_p_in_sel_opamp_t

MUX Selection for OpPsel.

Enumerator

MXC_E_AFE_P_IN_SEL_OPAMP_INPLUS OpPsel = INx+
MXC_E_AFE_P_IN_SEL_OPAMP_DAC_OR OpPsel = DAC_or
MXC_E_AFE_P_IN_SEL_OPAMP_SNO_OR OpPsel = SNO_or
MXC_E_AFE_P_IN_SEL_OPAMP_DAC_OR_AND_INPLUS OpPsel = DAC_or also output on INx+

enum mxc_afe_pos_in_sel_comp_t

MUX Selection for CmpPSel.

Enumerator

MXC_E_AFE_POS_IN_SEL_COMP_INPLUS CmpPSel = INx+
MXC_E_AFE_POS_IN_SEL_COMP_SCM CmpPSel = SCM
MXC_E_AFE_POS_IN_SEL_COMP_DAC1 CmpPSel = dac1
MXC_E_AFE_POS_IN_SEL_COMP_DAC3P CmpPSel = DAC3P
MXC_E_AFE_POS_IN_SEL_COMP_LED_OBS_PORT CmpPSel = LED Observe Port
MXC_E_AFE_POS_IN_SEL_COMP_DAC1_AND_INPLUS CmpPSel = dac1 also output on INx+
MXC_E_AFE_POS_IN_SEL_COMP_DAC3P_AND_INPLUS CmpPSel = DAC3P also output on INx+
MXC_E_AFE_POS_IN_SEL_COMP_DAC1_AND_SCM CmpPSel = dac1 also output on SCM

enum mxc_afe_ref_volt_sel_t

REFADC and REFDAC Voltage Select.

Enumerator

MXC_E_AFE_REF_VOLT_SEL_1024 Voltage Reference = 1.024 V
MXC_E_AFE_REF_VOLT_SEL_1500 Voltage Reference = 1.5 V
MXC_E_AFE_REF_VOLT_SEL_2048 Voltage Reference = 2.048 V
MXC_E_AFE_REF_VOLT_SEL_2500 Voltage Reference = 2.5 V

enum mxc_afe_scm_or_sel_t

Selection for MUX for SCM_or_sel.

Enumerator

MXC_E_AFE_SCM_OR_SEL_HIZ SCM_or = HIZ
MXC_E_AFE_SCM_OR_SEL_SCM0 SCM_or = SCM0
MXC_E_AFE_SCM_OR_SEL_SCM1 SCM_or = SCM1
MXC_E_AFE_SCM_OR_SEL_SCM2 SCM_or = SCM2
MXC_E_AFE_SCM_OR_SEL_SCM3 SCM_or = SCM3

enum mxc_afe_sno_or_sel_t

Selection for MUX for SNO_or_sel.

Enumerator

MXC_E_AFE_SNO_OR_SEL_HIZ	SNO_or = HIZ
MXC_E_AFE_SNO_OR_SEL_SNO0	SNO_or = SNO0
MXC_E_AFE_SNO_OR_SEL_SNO1	SNO_or = SNO1
MXC_E_AFE_SNO_OR_SEL_SNO2	SNO_or = SNO2
MXC_E_AFE_SNO_OR_SEL_SNO3	SNO_or = SNO3

enum mxc_afe_tmon_current_t

TMON Current Value.

Enumerator

MXC_E_AFE_TMON_CURRENT_VAL_0	TMON Current 4uA
MXC_E_AFE_TMON_CURRENT_VAL_1	TMON Current 60uA
MXC_E_AFE_TMON_CURRENT_VAL_2	TMON Current 64uA
MXC_E_AFE_TMON_CURRENT_VAL_3	TMON Current 120uA

enum mxc_lpc_neg_in_t

LPC Negative Input.

Enumerator

MXC_E_AFE_LPC_NEG_IN_PAD_INxN	LPC Negative Input = PAD_INxN
MXC_E_AFE_LPC_NEG_IN_SNOx	LPC Negative Input = SNOx
MXC_E_AFE_LPC_NEG_IN_DAC0P	LPC Negative Input = DAC0p
MXC_E_AFE_LPC_NEG_IN_DAC0N	LPC Negative Input = DAC0n
MXC_E_AFE_LPC_NEG_IN_DAC2P	LPC Negative Input = DAC2p
MXC_E_AFE_LPC_NEG_IN_LED_OBS_PORT	LPC Negative Input = LED Observation Port
MXC_E_AFE_LPC_NEG_IN_DAC0P_AND_INxN	LPC Negative Input = DAC0p output on INxN
MXC_E_AFE_LPC_NEG_IN_DAC0N_AND_INxN	LPC Negative Input = DAC0n output on INxN
MXC_E_AFE_LPC_NEG_IN_DAC2P_AND_INxN	LPC Negative Input = DAC2p output on INxN
MXC_E_AFE_LPC_NEG_IN_DAC2P_AND_SNO	LPC Negative Input = DAC2p output on SNO

enum mxc_lpc_pos_in_t

LPC Positive Input.

Enumerator

MXC_E_AFE_LPC_POS_IN_PAD_INxP	LPC Positive Input = PAD_INxP
MXC_E_AFE_LPC_POS_IN_SCMx	LPC Positive Input = SCMx
MXC_E_AFE_LPC_POS_IN_DAC1P	LPC Positive Input = DAC1p
MXC_E_AFE_LPC_POS_IN_DAC1N	LPC Positive Input = DAC1n
MXC_E_AFE_LPC_POS_IN_DAC3P	LPC Positive Input = DAC3p
MXC_E_AFE_LPC_POS_IN_LED_OBS_PORT	LPC Positive Input = LED Observation Port

MXC_E_AFE_LPC_POS_IN_DAC1P_AND_INxP LPC Positive Input = DAC1P output on INxP

MXC_E_AFE_LPC_POS_IN_DAC1N_AND_INxP LPC Positive Input = DAC1N output on INxP

MXC_E_AFE_LPC_POS_IN_DAC3P_AND_INxP LPC Positive Input = DAC3P output on INxP

MXC_E_AFE_LPC_POS_IN_DAC1P_AND_SCM LPC Positive Input = DAC1P output on SCM

MXC_E_AFE_LPC_POS_IN_DAC1N_AND_SCM LPC Positive Input = DAC1N output on SCM

enum mxc_opamp_mode_t

OpAmp vs Comparator Mode.

Enumerator

MXC_E_AFE_OPAMP_MODE_OPAMP MODE = OpAmp Mode

MXC_E_AFE_OPAMP_MODE_COMP MODE = Comparator Mode

enum mxc_opamp_neg_in_t

OpAmp Negative Input.

Enumerator

MXC_E_AFE_OPAMP_NEG_IN_PAD_INxN OpAmp Negative Input = PAD_INxN

MXC_E_AFE_OPAMP_NEG_IN_OUTx OpAmp Negative Input = OUTx

MXC_E_AFE_OPAMP_NEG_IN_SCM_HIZ OpAmp Negative Input = SCM_HIZ

MXC_E_AFE_OPAMP_NEG_IN_SCM0 OpAmp Negative Input = SCM0

MXC_E_AFE_OPAMP_NEG_IN_SCM1 OpAmp Negative Input = SCM1

MXC_E_AFE_OPAMP_NEG_IN_SCM2 OpAmp Negative Input = SCM2

MXC_E_AFE_OPAMP_NEG_IN_SCM3 OpAmp Negative Input = SCM3

MXC_E_AFE_OPAMP_NEG_IN_SCM_HIZ_INxN OpAmp Negative Input = SCM_HIZ output to INxN

MXC_E_AFE_OPAMP_NEG_IN_SCM0_INxN OpAmp Negative Input = SCM0 output to INxN

MXC_E_AFE_OPAMP_NEG_IN_SCM1_INxN OpAmp Negative Input = SCM1 output to INxN

MXC_E_AFE_OPAMP_NEG_IN_SCM2_INxN OpAmp Negative Input = SCM2 output to INxN

MXC_E_AFE_OPAMP_NEG_IN_SCM3_INxN OpAmp Negative Input = SCM3 output to INxN

enum mxc_opamp_neg_pad_t

OpAmp negative PAD Selection.

Enumerator

MXC_E_AFE_OPAMP_NEG_PAD_OFF OpAmp Negative Pad Selection = OFF

MXC_E_AFE_OPAMP_NEG_PAD_LED_OBS_PORT OpAmp Negative Pad Selection = LED Observe Port

MXC_E_AFE_OPAMP_NEG_PAD_DAC0P OpAmp Negative Pad Selection = DAC0p

MXC_E_AFE_OPAMP_NEG_PAD_DAC0N OpAmp Negative Pad Selection = DAC0n

MXC_E_AFE_OPAMP_NEG_PAD_DAC1P OpAmp Negative Pad Selection = DAC1p

MXC_E_AFE_OPAMP_NEG_PAD_DAC1N OpAmp Negative Pad Selection = DAC1n

MXC_E_AFE_OPAMP_NEG_PAD_DAC2P OpAmp Negative Pad Selection = DAC2p
MXC_E_AFE_OPAMP_NEG_PAD_DAC3P OpAmp Negative Pad Selection = DAC3p
MXC_E_AFE_OPAMP_NEG_PAD_DAC0P_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC0p and LED Obs Port
MXC_E_AFE_OPAMP_NEG_PAD_DAC0N_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC0n and LED Obs Port
MXC_E_AFE_OPAMP_NEG_PAD_DAC1P_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC1p and LED Obs Port
MXC_E_AFE_OPAMP_NEG_PAD_DAC1N_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC1n and LED Obs Port
MXC_E_AFE_OPAMP_NEG_PAD_DAC2P_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC2p and LED Obs Port
MXC_E_AFE_OPAMP_NEG_PAD_DAC3P_AND_LED_OBS_PORT OpAmp Negative Pad Selection = DAC3p and LED Obs Port

enum mxc_opamp_pos_in_t

OpAmp Positive Input.

Enumerator

MXC_E_AFE_OPAMP_POS_IN_PAD_INxP OpAmp Positive Input = PAD_INxP
MXC_E_AFE_OPAMP_POS_IN_DAC0P OpAmp Positive Input = DAC0p
MXC_E_AFE_OPAMP_POS_IN_DAC0N OpAmp Positive Input = DAC0n
MXC_E_AFE_OPAMP_POS_IN_DAC1P OpAmp Positive Input = DAC1p
MXC_E_AFE_OPAMP_POS_IN_DAC1N OpAmp Positive Input = DAC1n
MXC_E_AFE_OPAMP_POS_IN_DAC2P OpAmp Positive Input = DAC2p
MXC_E_AFE_OPAMP_POS_IN_DAC3P OpAmp Positive Input = DAC3p
MXC_E_AFE_OPAMP_POS_IN_SNO_HIZ OpAmp Positive Input = SNO_HIZ
MXC_E_AFE_OPAMP_POS_IN_SNO0 OpAmp Positive Input = SNO0
MXC_E_AFE_OPAMP_POS_IN_SNO1 OpAmp Positive Input = SNO1
MXC_E_AFE_OPAMP_POS_IN_SNO2 OpAmp Positive Input = SNO2
MXC_E_AFE_OPAMP_POS_IN_SNO3 OpAmp Positive Input = SNO3
MXC_E_AFE_OPAMP_POS_IN_DAC0P_INxP OpAmp Positive Input = DAC0p output to INxP
MXC_E_AFE_OPAMP_POS_IN_DAC0N_INxP OpAmp Positive Input = DAC0n output to INxP
MXC_E_AFE_OPAMP_POS_IN_DAC1P_INxP OpAmp Positive Input = DAC1p output to INxP
MXC_E_AFE_OPAMP_POS_IN_DAC1N_INxP OpAmp Positive Input = DAC1n output to INxP
MXC_E_AFE_OPAMP_POS_IN_DAC2P_INxP OpAmp Positive Input = DAC2p output to INxP
MXC_E_AFE_OPAMP_POS_IN_DAC3P_INxP OpAmp Positive Input = DAC3p output to INxP

2.3.3 Function Documentation

void AFE_ADCVRefDisable (uint8_t *fast_power_down*)

Disable the internal ADC Voltage Reference, switches ADC reference voltage to external source.

Parameters

fast_power_down	Enable or disable the pull-down resistor on the voltage reference

void AFE_ADCVRefEnable (mxc_afe_ref_volt_sel_t *adc_refsel*)

Enable the internal ADC Voltage Reference.

Parameters

adc_refsel	Value set to the internal ADC reference.

void AFE_DACVRefDisable (uint8_t *fast_power_down*)

Disable the internal DAC Voltage Reference, switches DAC reference voltage to external source.

Parameters

fast_power_down	Enable or disable the pull-down resistor on the voltage reference

void AFE_DACVRefEnable (mxc_afe_ref_volt_sel_t *dac_refsel*, mxc_afe_dac_ref_t *dacsel*)

Enable the internal DAC Voltage Reference.

Parameters

dac_refsel	Value set to the internal DAC reference.
dacsel	Select internal source for the DAC reference.

void AFE_GndSwitchSet (uint8_t *opamp_index*, mxc_afe_gnd_sel_opamp_t *state*)

Setup of GND Switch on Positive Pad of Opamp.

Parameters

opamp_index	Index of OpAmp to use.
state	Open or close the switch to ground.

void AFE_LEDConfig (uint8_t *port_index*, mxc_afe_led_cfg_port_t *led_cfg*)

Setup of LED current sink port. See hardware documentation for details on the current sink ports designed for driving LEDs.

Parameters

port_index	LED drive port number.

led_cfg	Configuration value.

void AFE_LPCConfig (uint8_t *lpc_index,* mxc_afe_bias_mode_comp_t *cmp_bias,* mxc_afe_hyst_comp_t *hysteresis,* uint8_t *tp_polarity*)

Configuration of Low Power Comparators (LPC)

Parameters

lpc_index	Index for the comparator to use.
cmp_bias	Set the power usage and reaction speed time
hysteresis	Set the magnitude of the comparator hysteresis
tp_polarity	Vp > Vn + Vhys = 0, Vp < Vn - Vhys = 1.

void AFE_LPCDisable (uint8_t *lpc_index*)

Power Disable of LPCs.

Parameters

lpc_index	Index for the comparator to use.

void AFE_LPCEnable (uint8_t *lpc_index*)

Power Enable of LPCs.

Parameters

lpc_index	Index for the comparator to use.

void AFE_LPCSetup (uint8_t *lpc_index,* mxc_lpc_pos_in_t *pos_input,* mxc_lpc_neg_in_t *neg_input,* mxc_afe_in_mode_comp_t *cmp_inmode*)

Setup of Low Power Comparators (LPC)

Parameters

lpc_index	Index for the comparator to use.
pos_input	MUXs of possible inputs for positive side of the comparator.
neg_input	MUXs of possible inputs for negative side of the comparator.
cmp_inmode	Comparator modes.

void AFE_LPCSetupInt (uint8_t *lpc_index,* void(∗)(void ∗arg) *intr_cb,* void ∗ *cb_arg*)

Setup and activate an interrupt for a Low Power Comparator (LPC)

Parameters

lpc_index	Index for the comparator to use.
intr_cb	Function pointer to callback function. If NULL pointer is passed, CPU interrupt is not activated and PMU interrupt will be activated
cb_arg	Pointer passed to the callback function.

void AFE_NpadSetup (uint8_t *opamp_index,* mxc_opamp_neg_pad_t *npad_select*)

NPAD Select for OpAmp.

Parameters

opamp_index	Index of OpAmp to use.
npad_select	Select connection for the negative pad on the opamp.

void AFE_OpAmpDisable (uint8_t *opamp_index*)

Power Disable of OPAMPs.

Parameters

opamp_index	Index of OpAmp to use.

void AFE_OpAmpEnable (uint8_t *opamp_index*)

Power Enable of OPAMPs.

Parameters

opamp_index	Index of OpAmp to use.

void AFE_OpAmpSetup (uint8_t *opamp_index,* mxc_opamp_mode_t *mode,* mxc_opamp_pos_in_t *pos_input,* mxc_opamp_neg_in_t *neg_input,* mxc_afe_in_mode_opamp_t *opamp_inmode*)

Setup of OPAMPs.

Parameters

opamp_index	Index of OpAmp to use.
mode	OpAmp = 0, Comparator = 1
pos_input	MUXs of possible inputs for positive side of the opamp.
neg_input	MUXs of possible inputs for the negative side of the opamp.
opamp_inmode	Set mode for the input pads.

void AFE_OpAmpSetupInt (uint8_t *opamp_index,* void(∗)(void ∗arg) *intr_cb,* void ∗ *cb_arg*)

Setup and activate an interrupt for an OpAmp in comparator mode.

Parameters

opamp_index	Index for the opamp comparator to use.
intr_cb	Function pointer to callback function. If NULL pointer is passed, CPU interrupt is not activated and PMU interrupt will be activated
cb_arg	Pointer passed to the callback function.

void AFE_SetSwitchMode (uint8_t *switch_index,* mxc_adc_spst_sw_ctrl_t *switch_mode*)

Setup Switch Control Mode.

Parameters

switch_index	Specifies the number of the switch being configured
switch_mode	Switch mode: 'software' or 'pulse train' control

void AFE_SetSwitchState (uint8_t *switch_index,* mxc_afe_close_spst_t *state*)

Setup of Switch.

Parameters

switch_index	Index of switch to set
state	Open or close switch.

void AFE_VRefExtBandgapSetup (uint8_t *v1extadj*)

Setup external bandgap reference voltage and adjust voltage 1.024V (2's complement)
$1024 = Vrefadj \frac{87+v1extadj}{104*v1extadj}$
Where $Vrefadj$ is on the external pin.

Parameters

v1extadj	Adjustment value

2.4 CLKMAN

Enumerations

- enum mxc_clkman_clk_t
- enum mxc_clkman_crypt_clk_t
- enum mxc_clkman_pll_input_select_t
- enum mxc_clkman_pll_divisor_select_t
- enum mxc_clkman_stability_count_t
- enum mxc_clkman_system_source_select_t
- enum mxc_clkman_adc_source_select_t
- enum mxc_clkman_wdt_source_select_t
- enum mxc_clkman_clk_scale_t
- enum mxc_clkman_clk_gate_t

Functions

- int32_t CLKMAN_HFXConfig (uint8_t hfx_bypass, uint8_t hfx_gm_adjust, uint8_t hfx_dc_control)
- void CLKMAN_HFXEnable (void)
- int32_t CLKMAN_HFXDisable (void)
- int32_t CLKMAN_PLLConfig (mxc_clkman_pll_input_select_t pll_input_select, mxc_clkman_pll_divisor_select_ pll_divisor_select, mxc_clkman_stability_count_t pll_stability_count, uint8_t pll_bypass, uint8_t pll_8mhz_enable)
- void CLKMAN_PLLEnable (void)
- int32_t CLKMAN_PLLDisable (void)
- void CLKMAN_TrimRO_Start (void)
- void CLKMAN_TrimRO_Stop (void)
- int32_t CLKMAN_SetSystemClock (mxc_clkman_system_source_select_t system_source_select)
- void CLKMAN_WaitForSystemClockStable (void)
- void CLKMAN_USBClockEnable (void)
- void CLKMAN_USBClockDisable (void)
- int32_t CLKMAN_CryptoClockConfig (mxc_clkman_stability_count_t crypto_stability_count)
- void CLKMAN_CryptoClockEnable (void)
- void CLKMAN_CryptoClockDisable (void)
- int32_t CLKMAN_SetADCClock (mxc_clkman_adc_source_select_t adc_source_select, mxc_adc_clk_mode adc_clk_mode)
- void CLKMAN_ADCClockDisable (void)
- int32_t CLKMAN_SetWatchdogClock (uint8_t index, mxc_clkman_wdt_source_select_t watchdog_source_select)
- int32_t CLKMAN_WatchdogClockDisable (uint8_t index)
- void CLKMAN_SetClkScale (mxc_clkman_clk_t device_clk, mxc_clkman_clk_scale_t clk_scale)
- void CLKMAN_SetCryptClkScale (mxc_clkman_crypt_clk_t device_clk, mxc_clkman_clk_scale_t clk_scale)
- void CLKMAN_SetRTOSMode (uint8_t enable)

2.4.1 Detailed Description

This is the high level API for the clock management module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.4.2 Enumeration Type Documentation

enum mxc_clkman_adc_source_select_t

Defines clock source selections for analog to digital converter clock.

Enumerator

MXC_E_CLKMAN_ADC_SOURCE_SELECT_SYSTEM	Clock select for system clock frequency
MXC_E_CLKMAN_ADC_SOURCE_SELECT_PLL_8MHZ	Clock select for 8MHz phase locked loop output
MXC_E_CLKMAN_ADC_SOURCE_SELECT_HFX	Clock select for high frequency crystal oscillator
MXC_E_CLKMAN_ADC_SOURCE_SELECT_24MHZ_RO	Clock select for 24MHz ring oscillator

enum mxc_clkman_clk_gate_t

Defines Setting of the Clock Gates .

Enumerator

MXC_E_CLKMAN_CLK_GATE_OFF	Clock Gater is Off
MXC_E_CLKMAN_CLK_GATE_DYNAMIC	Clock Gater is Dynamic
MXC_E_CLKMAN_CLK_GATE_ON	Clock Gater is On

enum mxc_clkman_clk_scale_t

Defines clock scales for various clocks.

Enumerator

MXC_E_CLKMAN_CLK_SCALE_DISABLED	Clock disabled
MXC_E_CLKMAN_CLK_SCALE_ENABLED	Clock enabled
MXC_E_CLKMAN_CLK_SCALE_DIV_2	Clock scale for dividing by 2
MXC_E_CLKMAN_CLK_SCALE_DIV_4	Clock scale for dividing by 4
MXC_E_CLKMAN_CLK_SCALE_DIV_8	Clock scale for dividing by 8
MXC_E_CLKMAN_CLK_SCALE_DIV_16	Clock scale for dividing by 16
MXC_E_CLKMAN_CLK_SCALE_DIV_32	Clock scale for dividing by 32
MXC_E_CLKMAN_CLK_SCALE_DIV_64	Clock scale for dividing by 64
MXC_E_CLKMAN_CLK_SCALE_DIV_128	Clock scale for dividing by 128
MXC_E_CLKMAN_CLK_SCALE_DIV_256	Clock scale for dividing by 256

enum mxc_clkman_clk_t

Selects an internal module clock for clock scaling configuration.

Enumerator

MXC_E_CLKMAN_CLK_SYS	Main System clock
MXC_E_CLKMAN_CLK_GPIO	GPIO module clock

MXC_E_CLKMAN_CLK_PT Pulse Train engine clock
MXC_E_CLKMAN_CLK_SPI0 SPI instance 0 module clock
MXC_E_CLKMAN_CLK_SPI1 SPI instance 1 module clock
MXC_E_CLKMAN_CLK_SPI2 SPI instance 2 module clock
MXC_E_CLKMAN_CLK_I2CM I2C Master module clock (for all instances)
MXC_E_CLKMAN_CLK_I2CS I2C Slave module clock
MXC_E_CLKMAN_CLK_LCD_CHPUMP LCD Charge pump clock
MXC_E_CLKMAN_CLK_PUF Reserved
MXC_E_CLKMAN_CLK_PRNG PRNG module clock
MXC_E_CLKMAN_CLK_WDT0 Watchdog Timer 0 clock
MXC_E_CLKMAN_CLK_WDT1 Watchdog Timer 1 clock
MXC_E_CLKMAN_CLK_RTC_INT_SYNC RTC synchronizer clock (required for cross-clock-domain register updates)
MXC_E_CLKMAN_CLK_DAC0 Clock for DAC 0
MXC_E_CLKMAN_CLK_DAC1 Clock for DAC 1
MXC_E_CLKMAN_CLK_DAC2 Clock for DAC 2
MXC_E_CLKMAN_CLK_DAC3 Clock for DAC 3

enum mxc_clkman_crypt_clk_t

Selects a TPU module clock for crypto ring-oscillator clock scaling configuration.

Enumerator

MXC_E_CLKMAN_CRYPT_CLK_AES AES engine clock
MXC_E_CLKMAN_CRYPT_CLK_MAA Modular Arithmetic Accelerator (MAA) clock
MXC_E_CLKMAN_CRYPT_CLK_PRNG Pseudo-random number Generator (PRNG) clock

enum mxc_clkman_pll_divisor_select_t

Defines clock input frequency for the phase locked loop.

Enumerator

MXC_E_CLKMAN_PLL_DIVISOR_SELECT_24MHZ Input frequency of 24MHz
MXC_E_CLKMAN_PLL_DIVISOR_SELECT_12MHZ Input frequency of 12MHz
MXC_E_CLKMAN_PLL_DIVISOR_SELECT_8MHZ Input frequency of 8MHz

enum mxc_clkman_pll_input_select_t

Defines clock input selections for the phase locked loop.

Enumerator

MXC_E_CLKMAN_PLL_INPUT_SELECT_HFX Input select for high frequency crystal oscillator
MXC_E_CLKMAN_PLL_INPUT_SELECT_24MHZ_RO Input select for 24MHz ring oscillator

enum mxc_clkman_stability_count_t

Defines terminal count for PLL stable.

Enumerator

MXC_E_CLKMAN_STABILITY_COUNT_2_8_CLKS Clock stable after $2^{\wedge}8 = 256$ clock cycles

MXC_E_CLKMAN_STABILITY_COUNT_2_9_CLKS Clock stable after 2^9 = 512 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_10_CLKS Clock stable after 2^10 = 1024 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_11_CLKS Clock stable after 2^11 = 2048 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_12_CLKS Clock stable after 2^12 = 4096 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_13_CLKS Clock stable after 2^13 = 8192 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_14_CLKS Clock stable after 2^14 = 16384 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_15_CLKS Clock stable after 2^15 = 32768 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_16_CLKS Clock stable after 2^16 = 65536 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_17_CLKS Clock stable after 2^17 = 131072 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_18_CLKS Clock stable after 2^18 = 262144 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_19_CLKS Clock stable after 2^19 = 524288 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_20_CLKS Clock stable after 2^20 = 1048576 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_21_CLKS Clock stable after 2^21 = 2097152 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_22_CLKS Clock stable after 2^22 = 4194304 clock cycles
MXC_E_CLKMAN_STABILITY_COUNT_2_23_CLKS Clock stable after 2^23 = 8388608 clock cycles

enum mxc_clkman_system_source_select_t

Defines clock source selections for system clock.

Enumerator

MXC_E_CLKMAN_SYSTEM_SOURCE_SELECT_24MHZ_RO_DIV_8 Clock select for 24MHz ring oscillator divided by 8 (3MHz)
MXC_E_CLKMAN_SYSTEM_SOURCE_SELECT_24MHZ_RO Clock select for 24MHz ring oscillator
MXC_E_CLKMAN_SYSTEM_SOURCE_SELECT_HFX Clock select for high frequency crystal oscillator
MXC_E_CLKMAN_SYSTEM_SOURCE_SELECT_PLL_48MHZ_DIV_2 Clock select for 48MHz phase locked loop output divided by 2 (24MHz)

enum mxc_clkman_wdt_source_select_t

Defines clock source selections for watchdog timer clock.

Enumerator

MXC_E_CLKMAN_WDT_SOURCE_SELECT_SYSTEM Clock select for system clock frequency
MXC_E_CLKMAN_WDT_SOURCE_SELECT_RTC Clock select for 8MHz phase locked loop output

MXC_E_CLKMAN_WDT_SOURCE_SELECT_24MHZ_RO Clock select for high frequency crystal oscillator

MXC_E_CLKMAN_WDT_SOURCE_SELECT_NANO Clock select for 24MHz ring oscillator

2.4.3 Function Documentation

int32_t CLKMAN_CryptoClockConfig (mxc_clkman_stability_count_t *crypto_stability_count*)

Configures but does not enable the crypto clock.

Parameters

crypto_stability_count	Number of clocks before crypto clock is stable.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_HFXConfig (uint8_t *hfx_bypass*, uint8_t *hfx_gm_adjust*, uint8_t *hfx_dc_control*)

Configures but does not enable the high frequency external oscillator circuitry.

Parameters

hfx_bypass	1 for crystal receiver bypass, 0 for no bypass.
hfx_gm_adjust	High frequency crystal gain adjust.
hfx_dc_control	High frequency crystal dc control.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_HFXDisable (void)

Disables the high frequency crystal receiver.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_PLLConfig (mxc_clkman_pll_input_select_t *pll_input_select*, mxc_clkman_pll_divisor_select_t *pll_divisor_select*, mxc_clkman_stability_count_t *pll_stability_count*, uint8_t *pll_bypass*, uint8_t *pll_8mhz_enable*)

Configures but does not enable the phase locked loop circuitry.

Parameters

pll_input_select	Phase locked loop clock input.

pll_divisor_select	Input clock frequency for the phase locked loop.
pll_stability_count	Number of clocks before phase locked loop is stable.
pll_bypass	1 for high frequency oscillator output for 48MHz clock, 0 for phase locked loop output.
pll_8mhz_enable	1 for enable 8MHz phase locked loop output, 0 for disable.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_PLLDisable (void)

Disables the phase locked loop circuitry.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_SetADCClock (mxc_clkman_adc_source_select_t *adc_source_select*, mxc_adc_clk_mode *adc_clk_mode*)

Sets the analog to digital converter clock source if the source is valid.

Parameters

adc_source_select	Analog to digital converter clock source.
adc_clk_mode	Divide the clock source into the ADC if source is too high. ADC must run at 8MHz.

Returns

0 => Success. Non zero => error condition.

void CLKMAN_SetClkScale (mxc_clkman_clk_t *device_clk*, mxc_clkman_clk_scale_t *clk_scale*)

Set the system clock scale.

Parameters

device_clk	device enum for clock scale setup
clk_scale	System clock scale.

void CLKMAN_SetCryptClkScale (mxc_clkman_crypt_clk_t *device_clk*, mxc_clkman_clk_scale_t *clk_scale*)

Set the TPU clock scale.

Parameters

device_clk	device enum for clock scale setup
clk_scale	System clock scale.

void CLKMAN_SetRTOSMode (uint8_t *enable*)

Set RTC clock for systick counter, allowing systick to operate in full clockgating powersaving mode.

Parameters

enable	1 enable 0 disable

int32_t CLKMAN_SetSystemClock (mxc_clkman_system_source_select_t *system_source_select*)

Sets the system clock source if the source is valid. Make sure HFX is stable before switching system clock.

Parameters

system_source_select	System clock source.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_SetWatchdogClock (uint8_t *index,* mxc_clkman_wdt_source_select_t *watchdog_source_select*)

Sets the watchdog clock source if the source is valid for the watchdog specified.

Parameters

index	Index of watchdog to set clock.
watchdog_source_select	Watchdog clock source.

Returns

0 => Success. Non zero => error condition.

int32_t CLKMAN_WatchdogClockDisable (uint8_t *index*)

Disable the watchdog clock source for the watchdog specified.

Parameters

index	Index of watchdog to disable.

Returns

0 => Success. Non zero => error condition.

2.5 DAC

Enumerations

- enum mxc_dac_pwr_mode_t
- enum mxc_dac_op_mode_t
- enum mxc_dac_interp_mode_t
- enum mxc_dac_start_mode_t

Functions

- void DAC_Enable (uint32_t dac_index, mxc_dac_pwr_mode_t power_mode)
- void DAC_Disable (uint32_t dac_index)
- void DAC_SetRate (uint32_t dac_index, uint16_t rate, mxc_dac_interp_mode_t interp_mode)
- void DAC_SetStartMode (uint32_t dac_index, mxc_dac_start_mode_t start_mode)
- int32_t DAC_PatternConfig (uint32_t dac_index, dac_transport_t ∗dac_handle, const void ∗data, uint32_t samples, uint16_t loops, void(∗done_cb)(int32_t exit_status, void ∗done_cb_arg), void ∗done_cb_arg)
- int32_t DAC_PatternStart (dac_transport_t ∗transport)
- void DAC_PatternStop (dac_transport_t ∗transport)
- int32_t DAC_SetOutput (uint8_t dac_index, uint32_t value)
- int32_t DAC_SetOutputRaw (uint8_t dac_index, uint32_t value)

2.5.1 Detailed Description

This is the high level API for the digital-to-analog converter module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

When using the DAC in periodic output mode to push wave-form patterns out, the following equation will apply for setting the pattern output rate
$T_{out} = \frac{N_s * N_i * (N_r + 2)}{F_c}$

Where:
N_s => Number of samples in a period as set in DAC_PatternConfig()
N_i => Interpolation rate set in DAC_SetRate()
N_r => DAC rate as set in DAC_SetRate()
F_c => DAC clock rate, normally system clock. Can be divided by using CLOCKMAN clock dividers.

2.5.2 Enumeration Type Documentation

enum mxc_dac_interp_mode_t

Defines the DAC Interpolation Options.

Enumerator

MXC_E_DAC_INTERP_MODE_DISABLED	DAC Interpolation is Disabled
MXC_E_DAC_INTERP_MODE_2_TO_1	DAC Interpolation 2:1
MXC_E_DAC_INTERP_MODE_4_TO_1	DAC Interpolation 4:1

MXC_E_DAC_INTERP_MODE_8_TO_1 DAC Interpolation 8:1

enum mxc_dac_op_mode_t

Defines the DAC Operational Modes.

Enumerator

MXC_E_DAC_OP_MODE_FIFO	DAC OpMode FIFO
MXC_E_DAC_OP_MODE_DACSMPLCNT	DAC OpMode Sample Count
MXC_E_DAC_OP_MODE_DAC_REG	DAC OpMode DAC_REG Control
MXC_E_DAC_OP_MODE_CONTINUOUS	DAC OpMode Continuous

enum mxc_dac_pwr_mode_t

Defines the DAC power modes. Intermediate values are only applicable to 12-bit DAC instances.

Enumerator

MXC_E_DAC_PWR_MODE_OFF	Power Level OFF (Disabled)
MXC_E_DAC_PWR_MODE_LVL0	Power Level 0 (48uA)
MXC_E_DAC_PWR_MODE_LVL1	Power Level 1 (130uA)
MXC_E_DAC_PWR_MODE_LVL2	Power Level 2 (210uA)
MXC_E_DAC_PWR_MODE_FULL	Power Level FullPwr (291uA)

enum mxc_dac_start_mode_t

Defines the DAC Start Modes.

Enumerator

MXC_E_DAC_START_MODE_FIFO_NOT_EMPTY	Start on FIFO Not Empty
MXC_E_DAC_START_MODE_ADC_STROBE	Start on ADC generated Start Strobe
MXC_E_DAC_START_MODE_DAC_STROBE	Start on DAC generated Start Strobe

2.5.3 Function Documentation

void DAC_Disable (uint32_t *dac_index*)

Power down the selected DAC instance.

Parameters

dac_index	DAC index number

void DAC_Enable (uint32_t *dac_index*, mxc_dac_pwr_mode_t *power_mode*)

Sets the power mode for the selected DAC instance.

Parameters

dac_index	DAC index number

power_mode	Set or disable DAC power

int32_t DAC_PatternConfig (uint32_t *dac_index,* dac_transport_t ∗ *dac_handle,* const void ∗ *data,* uint32_t *samples,* uint16_t *loops,* void(∗)(int32_t exit_status, void ∗done_cb_arg) *done_cb,* void ∗ *done_cb_arg*)

This will setup a single and re-usable output object for the DAC output port. The return pointer is allocated with malloc() and can be later released using free(). Calling this function does not access the DAC hardware directly; instead, this function loads all the information into a RAM buffer for later use by the PMU to transfer the pattern into the DAC FIFO. For cases where multiple patterns are used in an application, this function can be called separately for each different pattern. Then, the handle of the desired pattern can be passed to DAC_PatternStart() to start that pattern.

Parameters

dac_index	DAC index number
dac_handle	Pointer to an uninitialized state structure, this function will fill-in the correct fields.
data	Pointer to the beginning of the sample pattern in memory, needs to be at least samples∗2 bytes long
samples	number of samples to output in a single pass (UI/wave)
loops	number of times to repeat pattern, 0: forever or until DAC_PatternStop() is called
done_cb	(OPTIONAL, set to 'NULL' if not used) pointer to a callback function to be called when the pattern output has completed
done_cb_arg	(OPTIONAL, set to 'NULL' if not used) pointer to data to be given to the 'done' callback function
exit_status	If done_cb callback function is used, exit_status => 0; pattern pushed to DAC FIFO to completion without error. exit_status => -X; PMU or FIFO error.

Returns

0 on success.

int32_t DAC_PatternStart (dac_transport_t ∗ *transport*)

Starts a DAC pattern output process using a handle previously returned by DAC_PatternConfig(). The pattern is transmitted to the selected DAC FIFO using a dynamically allocated PMU channel. The PMU channel will free itself once the pattern is stopped, either because DAC_PatternStop() was called or because the number of output loops defined by the 'loops' parameter in DAC_PatternStart() have completed. Once the pattern output has started, the CPU is not needed to continue the pattern generation, which leaves the CPU free to perform other tasks or wait in sleep mode (LP2) until the pattern has completed.

Parameters

transport	Return handle from a call to DAC_PatternConfig().

Returns

0 => Success. Non zero => error condition.

void DAC_PatternStop (dac_transport_t ∗ *transport*)

Stop a running DAC output pattern that was configured with a loop count of 0 (repeat indefinitely).

Parameters

transport	Return handle from a call to DAC_PatternConfig().

int32_t DAC_SetOutput (uint8_t *dac_index*, uint32_t *value*)

Directly set the DAC to an exact value.

Parameters

dac_index	DAC index number
value	Exact value to set the DAC output DC level

Returns

0 => Success. Non zero => error condition.

int32_t DAC_SetOutputRaw (uint8_t *dac_index*, uint32_t *value*)

Directly set the DAC to an exact value.

Parameters

dac_index	DAC index number
value	FIF0 value to set the DAC output DC level

Returns

0 => Success. Non zero => error condition.

void DAC_SetRate (uint32_t *dac_index*, uint16_t *rate*, mxc_dac_interp_mode_t *interp_mode*)

Set values in the DAC registers related to output rates for use in a periodic mode.

Parameters

dac_index	DAC index number
rate	Delay between output samples in the output FIFO, see hardware docs
interp_mode	Level of interpolation between real points

void DAC_SetStartMode (uint32_t *dac_index*, mxc_dac_start_mode_t *start_mode*)

Set the start mode on the selected DAC instance. The start mode determines which source will trigger the start of the DAC pattern output.

Parameters

dac_index	DAC index number
start_mode	Device or module that will start the DAC output

2.6 Flash Controller

Functions

- int32_t FLC_Erase (uint32_t address, uint8_t erase_code, uint8_t unlock_key)
- int32_t FLC_WriteBlock (uint32_t address, const void ∗data, uint32_t length, uint8_t unlock_key)

2.6.1 Detailed Description

This is the high level API for the internal flash controller module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.6.2 Function Documentation

int32_t FLC_Erase (uint32_t *address,* uint8_t *erase_code,* uint8_t *unlock_key*)

This function will erase a single page of flash, 1 page is 2K bytes. Keys needed for flash are in the hardware specific register file "flc_regs.h".

Parameters

address	Start address that needs to be erased, must be aligned with 0x800
erase_code	Flash erase code; defined as 'MXC_V_FLC_ERASE_CODE_PAGE_ERASE' for page erase
unlock_key	Key necessary for accessing flash; defined as 'MXC_V_FLC_FLSH_UNLOCK_KEY'

Returns

0 => Success. Non zero => error condition.

int32_t FLC_WriteBlock (uint32_t *address,* const void ∗ *data,* uint32_t *length,* uint8_t *unlock_key*)

This function writes data to the flash device through flash controller.

Parameters

address	Start address that needs to be written, must be aligned with 4 bytes
data	Pointer to the buffer containing data to write
length	Size of the data to write in bytes, must be multiple of 4 bytes
unlock_key	Key necessary for accessing flash; defined as 'MXC_V_FLC_FLSH_UNLOCK_KEY'

Returns

0 => Success. Non zero => error condition.

2.7 GPIO

Enumerations

- enum gpio_int_mode_t
- enum gpio_in_mode_t
- enum gpio_out_mode_t

Functions

- void GPIO_SetOutMode (uint8_t port, uint8_t pin, gpio_out_mode_t val)
- void GPIO_SetFuncSel (uint8_t port, uint8_t pin, uint8_t val)
- void GPIO_SetInMode (uint8_t port, uint8_t pin, gpio_in_mode_t val)
- void GPIO_SetOutVal (uint8_t port, uint8_t pin, uint32_t val)
- uint32_t GPIO_GetInVal (uint8_t port, uint8_t pin)
- void GPIO_SetIntMode (uint8_t port, uint8_t pin, gpio_int_mode_t val, void(∗gpio_irq_cb)(void))

2.7.1 Detailed Description

This is the high level API for the general-purpose input/output module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.7.2 Enumeration Type Documentation

enum gpio_in_mode_t

Controls how the logical input value is determined for a given GPIO pin. This input value will also be used for interrupt detection if that has been enabled for the GPIO pin in question.

Enumerator

MXC_E_GPIO_IN_MODE_NORMAL Normal mode: low logic level translates to 0, high logic level translates to 1
MXC_E_GPIO_IN_MODE_INVERTED Inverted mode: low logic level translates to 1, high logic level translates to 0
MXC_E_GPIO_IN_MODE_ALWAYS_ZERO A zero value will always be returned for this pin's input regardless of the voltage level present at the pin
MXC_E_GPIO_IN_MODE_ALWAYS_ONE A one value will always be returned for this pin's input regardless of the voltage level present at the pin

enum gpio_int_mode_t

Defines the condition needed to generate an interrupt for a given GPIO pin.

Enumerator

MXC_E_GPIO_INT_MODE_DISABLE no interrupt generation
MXC_E_GPIO_INT_MODE_FALLING_EDGE falling edge detect
MXC_E_GPIO_INT_MODE_RISING_EDGE rising edge detect
MXC_E_GPIO_INT_MODE_ANY_EDGE any edge detect
MXC_E_GPIO_INT_MODE_LOW_LVL low level detect
MXC_E_GPIO_INT_MODE_HIGH_LVL high level detect

enum gpio_out_mode_t

Output drive mode of output. Defines drive state as modified by output setting of 0/1.

Enumerator

MXC_E_GPIO_OUT_MODE_TRISTATE	Wk1/Hiz
MXC_E_GPIO_OUT_MODE_OPEN_DRAIN	Hiz/Dr0
MXC_E_GPIO_OUT_MODE_OPEN_DRAIN_W_PULLUP	Wk1/Dr0
MXC_E_GPIO_OUT_MODE_TS	Unused
MXC_E_GPIO_OUT_MODE_NORMAL_TRISTATE	HiZ/Hiz
MXC_E_GPIO_OUT_MODE_NORMAL	Dr1/Dr0
MXC_E_GPIO_OUT_MODE_SLOW_TRISTATE	HiZ/Hiz
MXC_E_GPIO_OUT_MODE_SLOW	Dr1/Dr0
MXC_E_GPIO_OUT_MODE_FAST_TS	HiZ/Hiz
MXC_E_GPIO_OUT_MODE_FAST	Dr1/Dr0

2.7.3 Function Documentation

uint32_t GPIO_GetInVal (uint8_t *port*, uint8_t *pin*)

Returns the logical input value (as configured by GPIO_SetInMode()) for the selected GPIO pin.

Parameters

port	desired gpio port.
pin	desired gpio pin.

Returns

current value on this pin, as defined by GPIO_IN_MODE

void GPIO_SetFuncSel (uint8_t *port*, uint8_t *pin*, uint8_t *val*)

Sets the GPIO function select for the given GPIO pin. This setting is only effective if the GPIO pin in question is in 'GPIO mode', which means that the GPIO pin has not been requested for use by a higher priority function such as SPI, I2C, UART, LCD, etc.

Parameters

port	Selects GPIO port (starting at 0 for P0, 1 for P1, and so on)
pin	Selects pin within the selected GPIO port, from 0 to 7.
val	select field for selection of one a parameterized number of functions (Max 16, Min 2) which can control pad if owned by GPIO function.

See also

IOMUX matrix (GPIO: firmware control; Pulse Train: pulse train; tmr: 32-bits Timer)

```
*
*   val-=>   0       1       2       3       4       5       6       7
*          ------  ------  ------  ------  ------  ------  ------  ------
*   P0.0:   gpio    pt0     pt0     pt4     tmr0    tmr1    tmr2    tmr3
*   P0.1:   gpio    pt1     pt4     pt0     tmr1    tmr2    tmr3    tmr0
*   P0.2:   gpio    pt2     pt1     pt5     tmr2    tmr3    tmr0    tmr1
*   P0.3:   gpio    pt3     pt5     pt1     tmr3    tmr0    tmr1    tmr2
*   P0.4:   gpio    pt4     pt2     pt6     tmr0    tmr1    tmr2    tmr3
*   P0.5:   gpio    pt5     pt6     pt2     tmr1    tmr2    tmr3    tmr0
*   P0.6:   gpio    pt6     pt3     pt7     tmr2    tmr3    tmr0    tmr1
*   P0.7:   gpio    pt7     pt7     pt3     tmr3    tmr0    tmr1    tmr2
*   P1.0:   gpio    pt0     pt0     pt4     tmr0    tmr1    tmr2    tmr3
*   P1.1:   gpio   pt1    pt4    pt0    tmr1   tmr2   tmr3   tmr0
*   P1.2:   gpio   pt2    pt1    pt5    tmr2   tmr3   tmr0   tmr1
*   P1.3:   gpio   pt3    pt5    pt1    tmr3   tmr0   tmr1   tmr2
*   P1.4:   gpio   pt4    pt2    pt6    tmr0   tmr1   tmr2   tmr3
*   P1.5:   gpio   pt5    pt6    pt2    tmr1   tmr2   tmr3   tmr0
*   P1.6:   gpio   pt6    pt3    pt7    tmr2   tmr3   tmr0   tmr1
*   P1.7:   gpio   pt7    pt7    pt3    tmr3   tmr0   tmr1   tmr2
*   P2.0:   gpio   pt0    pt0    pt4    tmr0   tmr1   tmr2   tmr3
*   P2.1:   gpio   pt1    pt4    pt0    tmr1   tmr2   tmr3   tmr0
*   P2.2:   gpio   pt2    pt1    pt5    tmr2   tmr3   tmr0   tmr1
*   P2.3:   gpio   pt3    pt5    pt1    tmr3   tmr0   tmr1   tmr2
*   P2.4:   gpio   pt4    pt2    pt6    tmr0   tmr1   tmr2   tmr3
*   P2.5:   gpio   pt5    pt6    pt2    tmr1   tmr2   tmr3   tmr0
*   P2.6:   gpio   pt6    pt3    pt7    tmr2   tmr3   tmr0   tmr1
*   P2.7:   gpio   pt7    pt7    pt3    tmr3   tmr0   tmr1   tmr2
*   P6.0:   gpio   pt0    pt0    pt4    tmr0   tmr1   tmr2   tmr3
*   P6.1:   gpio   pt1    pt4    pt0    tmr1   tmr2   tmr3   tmr0
*   P6.2:   gpio   pt2    pt1    pt5    tmr2   tmr3   tmr0   tmr1
*   P6.3:   gpio   pt3    pt5    pt1    tmr3   tmr0   tmr1   tmr2
*   P6.4:   gpio   pt4    pt2    pt6    tmr0   tmr1   tmr2   tmr3
*   P6.5:   gpio   pt5    pt6    pt2    tmr1   tmr2   tmr3   tmr0
*   P6.6:   gpio   pt6    pt3    pt7    tmr2   tmr3   tmr0   tmr1
*   P6.7:   gpio   pt7    pt7    pt3    tmr3   tmr0   tmr1   tmr2
*   P7.0:   gpio   pt0    pt0    pt4    tmr0   tmr1   tmr2   tmr3
*   P7.1:   gpio   pt1    pt4    pt0    tmr1   tmr2   tmr3   tmr0
*   P7.2:   gpio   pt2    pt1    pt5    tmr2   tmr3   tmr0   tmr1
*   P7.3:   gpio   pt3    pt5    pt1    tmr3   tmr0   tmr1   tmr2
*   P7.4:   gpio   pt4    pt2    pt6    tmr0   tmr1   tmr2   tmr3
*   P7.5:   gpio   pt5    pt6    pt2    tmr1   tmr2   tmr3   tmr0
*   P7.6:   gpio   pt6    pt3    pt7    tmr2   tmr3   tmr0   tmr1
*   P7.7:   gpio   pt7    pt7    pt3    tmr3   tmr0   tmr1   tmr2
*
*
```

void GPIO_SetInMode (uint8_t *port*, uint8_t *pin*, gpio_in_mode_t *val*)

This function will set gpio input mode.

Parameters

port	desired gpio port.
pin	desired gpio pin.
val	gpio input mode

void GPIO_SetIntMode (uint8_t *port*, uint8_t *pin*, gpio_int_mode_t *val*, void(∗)(void) *gpio_irq_cb*)

This function will set gpio interrupt mode and register interrupt callback function.

Parameters

port	desired gpio port.
pin	desired gpio pin.
val	gpio interrupt mode
gpio_irq_cb	gpio interrupt callback function. If NULL, no ARM interrupt is generated. Useful for configuring internal signaling to the PMU.

void GPIO_SetOutMode (uint8_t *port*, uint8_t *pin*, gpio_out_mode_t *val*)

Sets the output mode for the selected GPIO pin.

Parameters

port	desired gpio port.
pin	desired gpio pin.
val	gpio output mode.

Returns

0 => Success. Non zero => error condition.

void GPIO_SetOutVal (uint8_t *port*, uint8_t *pin*, uint32_t *val*)

This function will set gpio out value.

Parameters

port	desired gpio port.
pin	desired gpio pin.
val	set 1 to high level; 0 to low level.

2.8 I2C Master

Enumerations

- enum i2cm_speed_t

Functions

- int32_t I2CM_Init (uint8_t index, i2cm_speed_t speed)
- int32_t I2CM_Read (uint8_t index, uint8_t addr, const uint8_t ∗cmd_data, uint32_t cmd_data_bytes, uint8_t ∗read_data, uint32_t read_data_bytes)
- int32_t I2CM_Write (uint8_t index, uint8_t addr, const uint8_t ∗cmd_data, uint32_t cmd_data_bytes, const uint8_t ∗write_data, uint32_t write_data_bytes)
- int32_t I2CM_WriteAsync (uint8_t index, uint8_t addr, const uint8_t ∗cmd, uint32_t cmd_bytes, const uint8_t ∗data, uint32_t data_bytes, void(∗tx_done)(int32_t ret_status))
- int32_t I2CM_ReadAsync (uint8_t index, uint8_t addr, const uint8_t ∗cmd, uint32_t cmd_bytes, uint8_t ∗data, uint32_t data_bytes, void(∗rx_handler)(int32_t rx_bytes))

2.8.1 Detailed Description

This is the high level API for the inter-integrated circuit master controller module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.8.2 Enumeration Type Documentation

enum i2cm_speed_t

speed option for i2c master

Enumerator

MXC_E_I2CM_SPEED_100KHZ	100KHz
MXC_E_I2CM_SPEED_400KHZ	400KHz
MXC_E_I2CM_SPEED_1MHZ	1MHz

2.8.3 Function Documentation

int32_t I2CM_Init (uint8_t *index*, i2cm_speed_t *speed*)

This function initialize the I2C master device.

Parameters

index	index of I2C master.
speed	speed of the I2C clock. The output is only correct with a I2Cm clock of 24MHz.

Returns

0 => Success. Non zero => error condition.

int32_t I2CM_Read (uint8_t *index*, uint8_t *addr*, const uint8_t ∗ *cmd_data*, uint32_t *cmd_data_bytes*, uint8_t ∗ *read_data*, uint32_t *read_data_bytes*)

This function performs a read transaction over I2C. A read transaction consists of a write from cmd_data of length cmd_data_bytes followed by a read of length read_data_bytes to read_data. If cmd_data_bytes is 0 then there will be neither an initial write nor a repeated start condition transmitted.

Parameters

index	index of I2C master.
addr	address of I2C slave, driver will take care of the read/write bit.
cmd	command data buffer.
cmd_bytes	number of command data bytes to write to slave.
data	read data buffer.
data_bytes	number of bytes to read from slave following the command write (and repeated start).

Returns

number of bytes read.

int32_t I2CM_ReadAsync (uint8_t *index*, uint8_t *addr*, const uint8_t ∗ *cmd*, uint32_t *cmd_bytes*, uint8_t ∗ *data*, uint32_t *data_bytes*, void(∗)(int32_t rx_bytes) *rx_handler*)

This function performs an asyncronized read transaction over I2C. A read transaction consists of a write from cmd_data of length cmd_data_bytes followed by a read of length read_data_bytes to read_data. If cmd_data_bytes is 0 then there will be neither an initial write nor a repeated start condition transmitted.

Parameters

index	index of I2C master.
addr	address of I2C slave, driver will take care of the read/write bit.
cmd_data	command data buffer.
cmd_data_bytes	number of command data bytes to write to slave.
read_data	read data buffer.
read_data_bytes	number of bytes to read from slave following the command write (and repeated start).

Returns

0 => Success. Non zero => error condition.

int32_t I2CM_Write (uint8_t *index*, uint8_t *addr*, const uint8_t ∗ *cmd_data*, uint32_t *cmd_data_bytes*, const uint8_t ∗ *write_data*, uint32_t *write_data_bytes*)

This function performs a write transaction over I2C. A write transaction consists of a write from cmd_data of length cmd_data_bytes followed by a write from write_data of length write_data_bytes. If either cmd_data_bytes or write_data_bytes is 0 then there will be only a single write with no repeated start condition.

Parameters

index	index of I2C master.
addr	address of I2C slave, driver will take care of the read/write bit.
cmd	command data buffer.
cmd_bytes	number of command data bytes to write to slave.
data	write data buffer.
data_bytes	number of bytes to write to slave following the command write (and repeated start).

Returns

number of bytes written.

int32_t I2CM_WriteAsync (uint8_t *index*, uint8_t *addr*, const uint8_t ∗ *cmd*, uint32_t *cmd_bytes*, const uint8_t ∗ *data*, uint32_t *data_bytes*, void(∗)(int32_t ret_status) *tx_done*)

This function performs an asyncronized write transaction over I2C. A write transaction consists of a write from cmd_data of length cmd_data_bytes followed by a write from write_data of length write_data_bytes. If either cmd_data_bytes or write_data_bytes is 0 then there will be only a single write with no repeated start condition.

Parameters

index	index of I2C master.
addr	address of I2C slave, driver will take care of the read/write bit.
cmd	command data buffer.
cmd_bytes	number of command data bytes to write to slave.
data	write data buffer.
data_bytes	number of bytes to write to slave following the command write (and repeated start).

Returns

0 => Success. Non zero => error condition.

2.9 ICC

Macros

- #define ICC_Flush()

Functions

- void ICC_Enable (void)
- void ICC_Disable (void)

2.9.1 Detailed Description

This is the high level API for the instruction cache controller module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.10 IO MUX

Enumerations

- enum ioman_mapping_t

Functions

- uint32_t IOMAN_SPI0 (ioman_mapping_t map, uint8_t core_io, uint8_t ss0, uint8_t ss1, uint8_t ss2, uint8_t ss3, uint8_t ss4, uint8_t sr0, uint8_t sr1, uint8_t quad, uint8_t fast)
- uint32_t IOMAN_SPI1 (ioman_mapping_t map, uint8_t core_io, uint8_t ss0, uint8_t ss1, uint8_t ss2, uint8_t ss3, uint8_t ss4, uint8_t sr0, uint8_t sr1, uint8_t quad, uint8_t fast)
- uint32_t IOMAN_SPI2 (ioman_mapping_t map, uint8_t core_io, uint8_t ss0, uint8_t ss1, uint8_t ss2, uint8_t ss3, uint8_t ss4, uint8_t sr0, uint8_t sr1, uint8_t quad, uint8_t fast)
- uint32_t IOMAN_UART0 (ioman_mapping_t map, uint8_t tr, uint8_t cts, uint8_t rts)
- uint32_t IOMAN_UART1 (ioman_mapping_t map, uint8_t tr, uint8_t cts, uint8_t rts)
- uint32_t IOMAN_I2CM0 (ioman_mapping_t map, uint8_t mstr_io)
- uint32_t IOMAN_I2CM1 (ioman_mapping_t map, uint8_t mstr_io)
- uint32_t IOMAN_I2CS0 (ioman_mapping_t map, uint8_t slave_io)
- uint32_t IOMAN_CRNT (uint8_t p0, uint8_t p1, uint8_t p2, uint8_t p3, uint8_t p4, uint8_t p5, uint8_t p6, uint8_t p7)
- uint32_t IOMAN_CRNTMode (uint8_t p0, uint8_t p1, uint8_t p2, uint8_t p3, uint8_t p4, uint8_t p5, uint8_t p6, uint8_t p7)
- uint32_t IOMAN_LCD (uint32_t m, uint32_t s0, uint32_t s1)

2.10.1 Detailed Description

High level API to program the IO pin matrix manager of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.10.2 Enumeration Type Documentation

enum ioman_mapping_t

Pin mapping define values common to all modules.

Enumerator

MXC_E_IOMAN_MAPPING_A	Pin Mapping 'A'
MXC_E_IOMAN_MAPPING_B	Pin Mapping 'B'
MXC_E_IOMAN_MAPPING_C	Pin Mapping 'C'
MXC_E_IOMAN_MAPPING_D	Pin Mapping 'D'
MXC_E_IOMAN_MAPPING_E	Pin Mapping 'E'
MXC_E_IOMAN_MAPPING_F	Pin Mapping 'F'
MXC_E_IOMAN_MAPPING_G	Pin Mapping 'G'
MXC_E_IOMAN_MAPPING_H	Pin Mapping 'H'

2.10.3 Function Documentation

uint32_t IOMAN_CRNT (uint8_t *p0*, uint8_t *p1*, uint8_t *p2*, uint8_t *p3*, uint8_t *p4*, uint8_t *p5*, uint8_t *p6*, uint8_t *p7*)

Set the pin mapping for current drive module.

Parameters

p0	Request pin pair for current drive port 0
p1	Request pin pair for current drive port 1
p2	Request pin pair for current drive port 2
p3	Request pin pair for current drive port 3
p4	Request pin pair for current drive port 4
p5	Request pin pair for current drive port 5
p6	Request pin pair for current drive port 6
p7	Request pin pair for current drive port 7

uint32_t IOMAN_CRNTMode (uint8_t *p0*, uint8_t *p1*, uint8_t *p2*, uint8_t *p3*, uint8_t *p4*, uint8_t *p5*, uint8_t *p6*, uint8_t *p7*)

Set the mode value for selected port(s) pin(s) in the current drive module.

Parameters

p0	Set current mode for pin selected in port 0
p1	Set current mode for pin selected in port 1
p2	Set current mode for pin selected in port 2
p3	Set current mode for pin selected in port 3
p4	Set current mode for pin selected in port 4
p5	Set current mode for pin selected in port 5
p6	Set current mode for pin selected in port 6
p7	Set current mode for pin selected in port 7

uint32_t IOMAN_I2CM0 (ioman_mapping_t *map*, uint8_t *mstr_io*)

Set the pin mapping for I2C master 0 module.

Parameters

map	Set the pin mapping for all configured I2CM pins
mstr_io	Request master mode for SCK and SDA pins.

uint32_t IOMAN_I2CM1 (ioman_mapping_t *map*, uint8_t *mstr_io*)

Set the pin mapping for I2C master 1 module.

Parameters

map	Set the pin mapping for all configured I2CM pins
mstr_io	Request master mode for SCK and SDA pins.

uint32_t IOMAN_I2CS0 (ioman_mapping_t *map*, uint8_t *slave_io*)

Set the pin mapping for I2C slave module.

Parameters

map	Set the pin mapping for all configured I2C slave pins
slave_io	Request slave mode for SCK and SDA pins.

uint32_t IOMAN_LCD (uint32_t *m*, uint32_t *s0*, uint32_t *s1*)

Set the pin mapping of the LCD module.

Parameters

s0	Set LCD SEG mode for GPIO[55:24]
s1	Set LCD SEG mode for GPIO[63:56]

uint32_t IOMAN_SPI0 (ioman_mapping_t *map*, uint8_t *core_io*, uint8_t *ss0*, uint8_t *ss1*, uint8_t *ss2*, uint8_t *ss3*, uint8_t *ss4*, uint8_t *sr0*, uint8_t *sr1*, uint8_t *quad*, uint8_t *fast*)

IOMUX mappings (7x7/WLP) .

GPIOs	Priority_1	Priority_2	Priority_3	Priority_4	Priority_5	Priority_6	Priority_7
P0.0	SPI0A_SCK	SPI1A_SS1	SPI1A_SR0			CUR_0_DRAIN	UART0D_RX
P0.1	SPI0A_MOSI	SPI1A_SS2	SPI1A_SR1			CUR_0_SRC	UART0D_TX
P0.2	SPI0A_MISO	SPI1A_SS3	SPI1A_SDIO2			CUR_1_DRAIN	UART0D_CTS
P0.3	SPI0A_SS0	SPI1A_SS4	SPI1A_SDIO3			CUR_1_SRC	UART0D_RTS
P0.4	SPI1A_SCK	SPI0A_SS1	SPI0A_SR0			CUR_2_DRAIN	I2CM0D/SD_SDA
P0.5	SPI1A_MOSI	SPI0A_SS2	SPI0A_SR1			CUR_2_SRC	I2CM0D/SD_SCL
P0.6	SPI1A_MISO	SPI0A_SS3	SPI0A_SDIO2			CUR_3_DRAIN	I2CM1D/SH_SDA
P0.7	SPI1A_SS0	SPI0A_SS4	SPI0A_SDIO3			CUR_3_SRC	I2CM1D/SH_SCL
P1.0	UART0A_RX			SPI0B_SCK	SPI1B_SS1	SPI1B_SR0	
P1.1	UART0A_TX			SPI0B_MOSI	SPI1B_SS2	SPI1B_SR1	
P1.2	UART1A_RX	UART0A_CTS		SPI0B_MISO	SPI1B_SS3	SPI1B_SDIO2	
P1.3	UART1A_TX	UART0A_RTS		SPI0B_SS0	SPI1B_SS4	SPI1B_SDIO3	
P1.4	I2CM0A/SA_SDA			SPI1B_SCK	SPI0B_SS1	SPI0B_SR0	
P1.5	I2CM0A/SA_SCL			SPI1B_MOSI	SPI0B_SS2	SPI0B_SR1	
P1.6	I2CM1A/SE_SDA	UART1A_CTS		SPI1B_MISO	SPI0B_SS3	SPI0B_SDIO2	UART1D_RX
P1.7	I2CM1A/SE_SCL	UART1A_RTS	SPI2A_SR0	SPI1B_SS0	SPI0B_SS4	SPI0B_SDIO3	UART1D_TX
P2.0	SPI2AB_SCK	UART0B_RX					
P2.1	SPI2AB_MOSI	UART0B_TX					
P2.2	SPI2AB_MISO	I2CM0B/SB_SDA					

P2.3	SPI2AB_SS0	I2CM0B/SB_SCK					
P2.4	LCD_COM0	UART1B_RX	UART0B_CTS	SPI2AB_SS1	SPI2B_SR0		
P2.5	LCD_COM1	UART1B_TX	UART0B_RTS	SPI2AB_SS2	SPI2AB_SR1		
P2.6	LCD_COM2	I2CM1B/SF_SDA	UART1B_CTS	SPI2AB_SS3	SPI2AB_SDIO2		
P2.7	LCD_COM3	I2CM1B/SF_SCL	UART1B_RTS	SPI2AB_SS4	SPI2AB_SDIO3		

IOMUX mappings (12x12)

GPIOs	Priority_1	Priority_2	Priority_3	Priority_4	Priority_5	Priority_6	Priority_7
P0.0	SPI0A_SCK	SPI1A_SS1	SPI1A_SR0				UART0D_RX
P0.1	SPI0A_MOSI	SPI1A_SS2	SPI1A_SR1				UART0D_TX
P0.2	SPI0A_MISO	SPI1A_SS3	SPI1A_SDIO2				UART0D_CTS
P0.3	SPI0A_SS0	SPI1A_SS4	SPI1A_SDIO3				UART0D_RTS
P0.4	SPI1A_SCK	SPI0A_SS1	SPI0A_SR0				I2CM0D/S_SDA
P0.5	SPI1A_MOSI	SPI0A_SS2	SPI0A_SR1				I2CM0D/S_SCL
P0.6	SPI1A_MISO	SPI0A_SS3	SPI0A_SDIO2				I2CM0H/S_SDA
P0.7	SPI1A_SS0	SPI0A_SS4	SPI0A_SDIO3				I2CM0H/S_SCL
P1.0	UART0A_RX			SPI0B_SCK	SPI1B_SS1	SPI1B_SR0	
P1.1	UART0A_TX			SPI0B_MOSI	SPI1B_SS2	SPI1B_SR1	
P1.2	UART1A_RX	UART0A_CTS		SPI0B_MISO	SPI1B_SS3	SPI1B_SDIO2	
P1.3	UART1A_TX	UART0A_RTS		SPI0B_SS0	SPI1B_SS4	SPI1B_SDIO3	
P1.4	LCD_COM0	UART1B_RX	UART0B_CTS	SPI2AB_SS1	SPI2B_SR0		
P1.5	LCD_COM1	UART1B_TX	UART0B_RTS	SPI2AB_SS2	SPI2B_SR1		
P1.6	LCD_COM2	I2CM0F/S_SDA	UART1B_CTS	SPI2AB_SS3	SPI2B_SDIO2		
P1.7	LCD_COM3	I2CM0F/S_SCL	UART1B_RTS	SPI2AB_SS4	SPI2B_SDIO3		
P2.0	SPI2AB_SCK	UART0B_RX					
P2.1	SPI2AB_MOSI	UART0B_TX					
P2.2	SPI2AB_MISO	I2CM0B/S_SDA					
P2.3	SPI2AB_SS0	I2CM0B/S_SCL					
P2.4	I2CM0A/S_SDA			SPI1B_SCK	SPI0_SS1	SPI0B_SR0	
P2.5	I2CM0A/S_SCL			SPI1B_MOSI	SPI0_SS2	SPI0B_SR1	
P2.6	I2CM0E/S_SDA	UART1A_CTS		SPI1B_MISO	SPI0_SS3	SPI0B_SDIO2	UART1D_RX
P2.7	I2CM0E/S_SCL	UART1A_RTS	SPI2A_SR0	SPI1B_SS0	SPI0_SS4	SPI0B_SDIO3	UART1D_TX
P3.0	LCD_SEG0						

P3.1	LCD_SEG1						
P3.2	LCD_SEG2						
P3.3	LCD_SEG3						
P3.4	LCD_SEG4						
P3.5	LCD_SEG5						
P3.6	LCD_SEG6						
P3.7	LCD_SEG7						
P4.0	LCD_SEG8						
P4.1	LCD_SEG9						
P4.2	LCD_SEG10						
P4.3	LCD_SEG11						
P4.4	LCD_SEG12						
P4.5	LCD_SEG13						
P4.6	LCD_SEG14						
P4.7	LCD_SEG15						
P5.0	LCD_SEG16						
P5.1	LCD_SEG17						
P5.2	LCD_SEG18						
P5.3	LCD_SEG19						
P5.4	LCD_SEG20						
P5.5	LCD_SEG21						
P5.6	LCD_SEG22						
P5.7	LCD_SEG23						
P6.0	LCD_SEG24	SPI0C_SCK	SPI1C_SS1	SPI1C_SR0		CUR_0_DRAIN	
P6.1	LCD_SEG25	SPI0C_MOSI	SPI1C_SS2	SPI1C_SR1		CUR_0_SRC	
P6.2	LCD_SEG26	SPI0C_MISO	SPI1C_SS3	SPI1C_SDIO2		CUR_1_DRAIN	
P6.3	LCD_SEG27	SPI0C_SS0	SPI1C_SS4	SPI1C_SDIO3		CUR_1_SRC	
P6.4	LCD_SEG28	SPI1C_SCK	SPI0C_SS1	SPI0C_SR0		CUR_2_DRAIN	
P6.5	LCD_SEG29	SPI1C_MOSI	SPI0C_SS2	SPI0C_SR1		CUR_2_SRC	
P6.6	LCD_SEG30	SPI1C_MISO	SPI0C_SS3	SPI0C_SDIO2		CUR_3_DRAIN	
P6.7							

P7.4	LCD_SEG36	I2CM0C/S_SDA				CUR_6_DRAIN	
P7.5	LCD_SEG37	I2CM0C/S_SCL				CUR_6_SRC	
P7.6	LCD_SEG38	I2CM1G/S_SDA	UART1C_CTS			CUR_7_DRAIN	
P7.7	LCD_SEG39	I2CM1G/S_SCL	UART1C_RTS			CUR_7_SRC	

Set the pin mapping of SPI0 module

Parameters

map	Select pinmapping for all enabled SPI pins
core_io	Request SPI mode for SCLK, SDIO(0) and SDIO(1)
ss0	Request slave select 0 active out
ss1	Request slave select 1 active out
ss2	Request slave select 2 active out
ss3	Request slave select 3 active out
ss4	Request slave select 4 active out
sr0	Request sr0 for flow control
sr1	Request sr1 for flow control
quad	Request quad IO
fast	Request fast mode

uint32_t IOMAN_SPI1 (ioman_mapping_t *map*, uint8_t *core_io*, uint8_t *ss0*, uint8_t *ss1*, uint8_t *ss2*, uint8_t *ss3*, uint8_t *ss4*, uint8_t *sr0*, uint8_t *sr1*, uint8_t *quad*, uint8_t *fast*)

Set the pin mapping of SPI1 module.

Parameters

map	Select pinmapping for all enabled SPI pins
core_io	Request SPI mode for SCLK, SDIO(0) and SDIO(1)
ss0	Request slave select 0 active out
ss1	Request slave select 1 active out
ss2	Request slave select 2 active out
ss3	Request slave select 3 active out
ss4	Request slave select 4 active out
sr0	Request sr0 for flow control
sr1	Request sr1 for flow control
quad	Request quad IO
fast	Request fast mode

uint32_t IOMAN_SPI2 (ioman_mapping_t *map*, uint8_t *core_io*, uint8_t *ss0*, uint8_t *ss1*, uint8_t *ss2*, uint8_t *ss3*, uint8_t *ss4*, uint8_t *sr0*, uint8_t *sr1*, uint8_t *quad*, uint8_t *fast*)

Set the pin mapping of SPI2 module.

Parameters

map	Select pinmapping for all enabled SPI pins
core_io	Request SPI mode for SCLK, SDIO(0) and SDIO(1)
ss0	Request slave select 0 active out
ss1	Request slave select 1 active out
ss2	Request slave select 2 active out
ss3	Request slave select 3 active out
ss4	Request slave select 4 active out
sr0	Request sr0 for flow control
sr1	Request sr1 for flow control
quad	Request quad IO
fast	Request fast mode

uint32_t IOMAN_UART0 (ioman_mapping_t *map*, uint8_t *tr*, uint8_t *cts*, uint8_t *rts*)

Set the pin mapping of the UART0 module.

Parameters

map	Set the pin mapping for all configured UART pins
tr	Request TX and RX pins
cts	Request CTS pin
rts	Request RTS pin

uint32_t IOMAN_UART1 (ioman_mapping_t *map*, uint8_t *tr*, uint8_t *cts*, uint8_t *rts*)

Set the pin mapping of the UART1 module.

Parameters

map	Set the pin mapping for all configured UART pins
tr	Request TX and RX pins
cts	Request CTS pin
rts	Request RTS pin

2.11 LCD Display Driver

Typedefs

- typedef int32_t(∗ lcd_display_callback_t) (uint8_t position, uint8_t display_char)

Enumerations

- enum lcd_duty_t

Functions

- void LCD_Enable (void)
- void LCD_Disable (void)
- int32_t LCD_Init (uint8_t segments, uint8_t gnd_enable, uint8_t frame_rate, lcd_duty_t duty_cycle, int32_t(∗LCD_DisplayChar_cb)(uint8_t position, uint8_t ch), uint8_t max_length)
- void LCD_SetAdjust (uint8_t value)
- void LCD_Clear (void)
- int32_t LCD_Display (uint8_t ∗msg)
- void LCD_Write (uint8_t address, uint8_t data)
- void LCD_Hold (void)
- void LCD_Update (void)

2.11.1 Detailed Description

This is the high level API for the liquid-crystal display driver module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.11.2 Typedef Documentation

typedef int32_t(∗ lcd_display_callback_t) (uint8_t position, uint8_t display_char)

This function is to be defined in whole by the external device specific driver code. This is where the device specific driver should decode from the position and display_char which segments of LCD RAM do write and then call lcd_write() correctly.

Parameters

position	position of display.
display_char	character to display.

2.11.3 Enumeration Type Documentation

enum lcd_duty_t

Supported duty cycles for the LCD Controller.

Enumerator

LCD_DUTY_STATIC	Each pin is dedicated to a single LCD segment.
LCD_DUTY_50	1/2 duty cycle. Each pin can drive two LCD segments.

LCD_DUTY_33	1/3 duty cycle. Each pin can drive three LCD segments.
LCD_DUTY_25	1/4 duty cycle. Each pin can drive four LCD segments.

2.11.4 Function Documentation

int32_t LCD_Display (uint8_t ∗ *msg*)

Send a full string to the LCD device and activate in an atomic way.

Parameters

msg	string do display in whole.

int32_t LCD_Init (uint8_t *segments,* uint8_t *gnd_enable,* uint8_t *frame_rate,* lcd_duty_t *duty_cycle,* int32_t(∗)(uint8_t position, uint8_t ch) *LCD_DisplayChar_cb,* uint8_t *max_length*)

Initialize the controller for use with a specific LCD display.

Parameters

segments	Number of segments driven by the controller.
gnd_enable	Connect Radj to ground if true.
frame_rate	Display frame rate.
duty_cydle	See lcd_duty_t} for duty cycle enumerations.
LCD_DisplayCHar_cb	Callback function to write character to display.
max_length	maximum lcd display length

Returns

0 for success

void LCD_SetAdjust (uint8_t *value*)

Set the adjustment resistor. This will change the contrast in the LCD controller waveform generator.

Parameters

value	resistor value (0-255, four bits of resolution).

void LCD_Write (uint8_t *address,* uint8_t *data*)

Write a byte to a specific LCD address. This is the function that the device specific function lcd_display_char() should use.

Parameters

address	LCD RAM address.
data	LCD RAM data.

2.12 Peripheral Management Unit

Functions

- int8_t PMU_GetChannel (void)
- int32_t PMU_Start (int8_t channel, const void ∗opcode, void(∗intr_cb)(int32_t exit_status, void ∗cb_arg), void ∗cb_arg)
- void PMU_SetCounter (int8_t channel, int8_t counter_num, uint16_t value)
- void PMU_Stop (int8_t channel)

2.12.1 Detailed Description

This is the high level API for the peripheral management unit module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.12.2 Function Documentation

int8_t PMU_GetChannel (void)

Request and lock a PMU channel. The channel will be automatically free when the PMU program completes with the 'stop' bit set in the final op-code of the PMU program.

Returns

The next available channel in a stack fashion, or -1 if none are available

void PMU_SetCounter (int8_t *channel,* int8_t *counter_num,* uint16_t *value*)

Set a loop counter value on a channel.

Parameters

channel	Channel number to set the value on.
counter_num	Counter number for the channel, (0 or 1).
value	Loop count value.

int32_t PMU_Start (int8_t *channel,* const void ∗ *opcode,* void(∗)(int32_t exit_status, void ∗cb_arg) *intr_cb,* void ∗ *cb_arg*)

Start a PMU program on a channel.

Parameters

channel	Channel that will run the PMU program.
opcode	Pointer to the first opcode of the PMU program.
intr_cb	Callback function to be called for every opcode that has the interrupt bit set. The arguments to the callback function include a void pointer specified in the next arg to this func cb_arg, and a single bit value of 1 if the interrupt opcode has the stop bit set indicating completion of this PMU program. If the channel was allocated with PMU_GetChannel(), having the 'stop' bit set in an opcode will automatically free the channel.
cb_arg	Pointer to be passed to the interrupt callback function.

Returns

0 => Success. Non zero => error condition.

void PMU_Stop (int8_t *channel*)

Stop a running channel. This will de-assert the enable bit on the channel and stop the running PMU program at the current op-code. The PMU interrupt will not get set and the int_cb function will not be called.

Parameters

channel	Channel to stop.

2.13 Power Management

Enumerations

- enum pwrseq_vdd3_trip_point_t
- enum mxc_pwr_mode_t
- enum mxc_pwr_enable_t
- enum mxc_pwr_trickle_charger_t
- enum mxc_pwr_device_t
- enum mxc_pwr_event_t
- enum mxc_pwrman_pad_mode_t

Functions

- void PWR_SetTripPointVDD3 (uint32_t vdd3, void(∗trippoint)(void))
- void PWR_EnableGPIO (void)
- void PWR_DisableGPIO (void)
- void PWR_SetGPIOWUD (uint8_t port, uint8_t pin, uint8_t act_high)
- void PWR_ClearAllGPIOWUD (void)
- void PWR_ClearGPIOWUD (uint8_t port, uint8_t pin)
- void PWR_SetCompWUD (uint8_t index, uint8_t rising_edge)
- void PWR_ClearCompWUD (uint8_t index)
- void PWR_ClearAllCompWUD (void)
- void PWR_ClearFlags (void)
- void PWR_SetMode (mxc_pwr_mode_t mode, void(∗wakeup)(void))
- void PWR_Sleep (void)
- void PWR_Init (void)
- void PWR_Enable (mxc_pwr_enable_t module)
- void PWR_Disable (mxc_pwr_enable_t module)
- void PWR_SetTrickleCharger (mxc_pwr_trickle_charger_t decode)
- void PWR_EnableDevRun (mxc_pwr_device_t device)
- void PWR_EnableDevSleep (mxc_pwr_device_t device)
- void PWR_DisableDevRun (mxc_pwr_device_t device)
- void PWR_DisableDevSleep (mxc_pwr_device_t device)
- void PWR_EnableAllWakeupEvents (void)
- void PWR_DisableAllWakeupEvents (void)
- void PWR_EnableWakeupEvent (mxc_pwr_event_t event)
- void PWR_DisableWakeupEvent (mxc_pwr_event_t event)
- void PWR_SetGPIOWeakDriver (uint8_t port, uint8_t pin, uint8_t act_high)

2.13.1 Detailed Description

This is the high level API for the power management module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.13.2 Enumeration Type Documentation

enum mxc_pwr_device_t

Defines devices to enable/disable.

Enumerator

MXC_E_PWR_DEVICE_LDO LDO regulator power switch
MXC_E_PWR_DEVICE_CHZY CHZY regulator power switch
MXC_E_PWR_DEVICE_RO Relaxation oscillator power switch
MXC_E_PWR_DEVICE_NR Nano ring oscillator power switch
MXC_E_PWR_DEVICE_RTC Real-time clock power switch
MXC_E_PWR_DEVICE_SVM3 VDD3 system voltage monitor power switch
MXC_E_PWR_DEVICE_SVM1 VREG18 system voltage monitor power switch
MXC_E_PWR_DEVICE_SVMRTC VRTC system voltage monitor power switch
MXC_E_PWR_DEVICE_SVMVDDA3 VDDA3 system voltage monitor power switch

enum mxc_pwr_enable_t

Defines modules to enable/disable.

Enumerator

MXC_E_PWR_ENABLE_AFE AFE Powered
MXC_E_PWR_ENABLE_IO I/O Active
MXC_E_PWR_ENABLE_USB USB Powered
MXC_E_PWR_ENABLE_STATIC_PULLUPS Static Pullups Enabled

enum mxc_pwr_event_t

Defines event masks to enable/disable.

Enumerator

MXC_E_PWR_EVENT_SYS_REBOOT Firmware reset event
MXC_E_PWR_EVENT_POWER_FAIL Power fail event
MXC_E_PWR_EVENT_BOOT_FAIL Boot fail event
MXC_E_PWR_EVENT_COMP_FLAG Comparator wakeup event
MXC_E_PWR_EVENT_IO_FLAG GPIO wakeup event
MXC_E_PWR_EVENT_VDD3_RST VDD3 reset comparator event
MXC_E_PWR_EVENT_VDD3_WARN VDD3 warning comparator event
MXC_E_PWR_EVENT_VDD1_RST VREG18 reset comparator event
MXC_E_PWR_EVENT_VDD1_LOW_RST VREG18 reset low comparator event
MXC_E_PWR_EVENT_VDD1_WARN VREG18 warning comparator event
MXC_E_PWR_EVENT_VRTC_WARN VRTC comparator event
MXC_E_PWR_EVENT_POR3Z_FAIL POR3 and POR3_lite event
MXC_E_PWR_EVENT_RTC_CMPR0 RTC cmpr0 event
MXC_E_PWR_EVENT_RTC_CMPR1 RTC cmpr1 event
MXC_E_PWR_EVENT_RTC_PRESCALE_CMP RTC prescale event
MXC_E_PWR_EVENT_RTC_ROLLOVER RTC rollover event
MXC_E_PWR_EVENT_BROWNOUT RTC brownout event
MXC_E_PWR_EVENT_USB_PLUG RTC usb plug inserted event
MXC_E_PWR_EVENT_USB_REMOVE RTC usb plug removed event
MXC_E_PWR_EVENT_VDD22_RST VDD22 reset comparator event
MXC_E_PWR_EVENT_VDD195_RST VDD195 reset comparator event

enum mxc_pwr_mode_t

Defines power modes.

Enumerator

MXC_E_PWR_MODE_LP0	ARM deep sleep mode without data retention (WFE)
MXC_E_PWR_MODE_LP1	ARM deep sleep mode with data retention (WFE)
MXC_E_PWR_MODE_LP2	ARM sleep mode (WFI)
MXC_E_PWR_MODE_LP3	No sleep mode

enum mxc_pwr_trickle_charger_t

Enumerator

MXC_E_PWR_TRICKLE_CHARGER_NO_DIODE_W_250_OHM	Trickle charger with no diode and 250 ohm resistor
MXC_E_PWR_TRICKLE_CHARGER_NO_DIODE_W_2K_OHM	Trickle charger with no diode and 2k ohm resistor
MXC_E_PWR_TRICKLE_CHARGER_NO_DIODE_W_4K_OHM	Trickle charger with no diode and 4k ohm resistor
MXC_E_PWR_TRICKLE_CHARGER_DIODE_W_250_OHM	Trickle charger with diode and 250 ohm resistor
MXC_E_PWR_TRICKLE_CHARGER_DIODE_W_2K_OHM	Trickle charger with diode and 2k ohm resistor
MXC_E_PWR_TRICKLE_CHARGER_DIODE_W_4K_OHM	Trickle charger with diode and 4k ohm resistor

enum mxc_pwrman_pad_mode_t

Defines PAD Modes for Wake Up Detection.

Enumerator

MXC_E_PWRMAN_PAD_MODE_CLEAR_SET	WUD Mode for Selected PAD = Clear/Activate
MXC_E_PWRMAN_PAD_MODE_ACT_HI_LO	WUD Mode for Selected PAD = Set WUD Act Hi/Set WUD Act Lo
MXC_E_PWRMAN_PAD_MODE_WEAK_HI_LO	WUD Mode for Selected PAD = Set Weak Hi/ Set Weak Lo
MXC_E_PWRMAN_PAD_MODE_NONE	WUD Mode for Selected PAD = No pad state change

enum pwrseq_vdd3_trip_point_t

Defines the Supply Voltage Monitor Trip Setting for VDD3.

Enumerator

MXC_E_VDD3_1_764V_TRIP_POINT	VDD3 trip point = 1.764V
MXC_E_VDD3_1_798V_TRIP_POINT	VDD3 trip point = 1.798V
MXC_E_VDD3_1_833V_TRIP_POINT	VDD3 trip point = 1.833V
MXC_E_VDD3_1_870V_TRIP_POINT	VDD3 trip point = 1.870V
MXC_E_VDD3_1_908V_TRIP_POINT	VDD3 trip point = 1.908V
MXC_E_VDD3_1_948V_TRIP_POINT	VDD3 trip point = 1.948V
MXC_E_VDD3_1_989V_TRIP_POINT	VDD3 trip point = 1.989V
MXC_E_VDD3_2_032V_TRIP_POINT	VDD3 trip point = 2.032V

MXC_E_VDD3_2_077V_TRIP_POINT	VDD3 trip point = 2.077V
MXC_E_VDD3_2_125V_TRIP_POINT	VDD3 trip point = 2.125V
MXC_E_VDD3_2_174V_TRIP_POINT	VDD3 trip point = 2.174V
MXC_E_VDD3_2_226V_TRIP_POINT	VDD3 trip point = 2.226V
MXC_E_VDD3_2_280V_TRIP_POINT	VDD3 trip point = 2.280V
MXC_E_VDD3_2_337V_TRIP_POINT	VDD3 trip point = 2.337V
MXC_E_VDD3_2_397V_TRIP_POINT	VDD3 trip point = 2.397V
MXC_E_VDD3_2_460V_TRIP_POINT	VDD3 trip point = 2.460V
MXC_E_VDD3_2_526V_TRIP_POINT	VDD3 trip point = 2.526V
MXC_E_VDD3_2_597V_TRIP_POINT	VDD3 trip point = 2.597V
MXC_E_VDD3_2_671V_TRIP_POINT	VDD3 trip point = 2.671V
MXC_E_VDD3_2_749V_TRIP_POINT	VDD3 trip point = 2.749V
MXC_E_VDD3_2_833V_TRIP_POINT	VDD3 trip point = 2.833V
MXC_E_VDD3_2_921V_TRIP_POINT	VDD3 trip point = 2.921V
MXC_E_VDD3_3_015V_TRIP_POINT	VDD3 trip point = 3.015V
MXC_E_VDD3_3_116V_TRIP_POINT	VDD3 trip point = 3.116V
MXC_E_VDD3_3_223V_TRIP_POINT	VDD3 trip point = 3.223V
MXC_E_VDD3_3_339V_TRIP_POINT	VDD3 trip point = 3.339V
MXC_E_VDD3_3_462V_TRIP_POINT	VDD3 trip point = 3.462V
MXC_E_VDD3_3_595V_TRIP_POINT	VDD3 trip point = 3.595V
MXC_E_VDD3_3_730V_TRIP_POINT	VDD3 trip point = 3.739V
MXC_E_VDD3_3_895V_TRIP_POINT	VDD3 trip point = 3.895V
MXC_E_VDD3_4_064V_TRIP_POINT	VDD3 trip point = 4.064V

2.13.3 Function Documentation

void PWR_ClearCompWUD (uint8_t *index*)

Clears WUD for designated comparator.

Parameters

index	Comparator index.

void PWR_ClearGPIOWUD (uint8_t *port*, uint8_t *pin*)

Clears WUD for designated GPIO port and pin.

Parameters

port	Port index.
pin	Pin index.

void PWR_DisableDevRun (mxc_pwr_device_t *device*)

Disable a module in run mode.

Parameters

device	Device to disable in run mode.

void PWR_DisableDevSleep (mxc_pwr_device_t *device*)

Disable a module in sleep mode.

Parameters

device	Device to disable in sleep mode.

void PWR_DisableWakeupEvent (mxc_pwr_event_t *event*)

Disables wakeup event to wake up power sequencer for a given event.

Parameters

event	Event mask to disable.

void PWR_EnableDevRun (mxc_pwr_device_t *device*)

Enable a module in run mode.

Parameters

device	Device to enable in run mode.

void PWR_EnableDevSleep (mxc_pwr_device_t *device*)

Enable a module in sleep mode.

Parameters

device	Device to enable in sleep mode.

void PWR_EnableWakeupEvent (mxc_pwr_event_t *event*)

Enables wakeup events to wake up power sequencer for a given event.

Parameters

event	Event mask to enable.

void PWR_SetCompWUD (uint8_t *index,* uint8_t *rising_edge*)

Sets up WUD for designated comparator.

Parameters

index	Comparator index.

rising_edge	1 for rising edge, 0 for falling edge.

void PWR_SetGPIOWeakDriver (uint8_t *port*, uint8_t *pin*, uint8_t *act_high*)

This function will set gpio in tristate with a 1 MEG pulldown.

Parameters

port	desired gpio port.
pin	desired gpio pin.
act_high	set 1 to high level; 0 to low level.

void PWR_SetGPIOWUD (uint8_t *port*, uint8_t *pin*, uint8_t *act_high*)

Sets up WUD for designated GPIO port and pin.

Parameters

port	Port index.
pin	Pin index.
act_high	1 for active high, 0 for active low.

void PWR_SetMode (mxc_pwr_mode_t *mode*, void(∗)(void) *wakeup*)

Sets processor mode.

Parameters

mode	Sleep mode.
wakeup	Callback function for return from LP1.

void PWR_SetTrickleCharger (mxc_pwr_trickle_charger_t *decode*)

Set up trickle charger super-cap.

Parameters

decode	Trickle charger resistor and diode.

void PWR_SetTripPointVDD3 (uint32_t *vdd3*, void(∗)(void) *trippoint*)

Sets the Voltage Trip Point for VDD3.

Parameters

vdd3	Use the pwrseq_vdd3_trip_point_t enumerations.
trippoint	Callback function for when the VDD3 voltage trippoint is reached .

2.14 PRNG

Functions

- int PRNG_Init (void)
- uint16_t PRNG_GetSeed (void)
- void PRNG_AddUserEntropy (uint8_t entropy)

2.14.1 Detailed Description

This is the high level API for the MAX32600 PRNG module.

Note

> The PRNG hardware does not produce true random numbers. The output should be used as a seed to an approved random-number algorithm, per a certifying authority such as NIST or PCI. The approved algorithm will output random numbers which are cerfitied for use in encryption and authentication algorithms.

2.14.2 Function Documentation

void PRNG_AddUserEntropy (uint8_t *entropy*)

Add user entropy to the PRNG entropy source.

Parameters

entropy	This value will be mixed into the PRNG entropy source

uint16_t PRNG_GetSeed (void)

Retrieve a seed value from the PRNG.

Note

> The PRNG hardware does not produce true random numbers. The output should be used as a seed to an approved random-number algorithm, per a certifying authority such as NIST or PCI. The approved algorithm will output random numbers which are certified for use in encryption and authentication algorithms.

Returns

> This function returns a 16-bit seed value

int PRNG_Init (void)

Initialize required clocks and enable PRNG module.

Note

> Function will set divisors to /1 if they are found disabled. Otherwise, it will not change the divisor.

Returns

> <0 if error, otherwise success

2.15 Pulse Train

Functions

- void PT_Init (void)
- void PT_SetPulseTrain (uint8_t index, uint32_t rate_control, uint8_t mode, uint32_t pattern)
- void PT_Start (void)
- void PT_Stop (void)
- void PT_Resync (uint32_t resync_pts)

2.15.1 Detailed Description

This is the high level API for the pulse train module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.15.2 Function Documentation

void PT_Resync (uint32_t *resync_pts*)

Resynchronize individual pulse trains together.

Parameters

resync_pts	Mask of pulse train modules that need to be re-synced by bit number. Bit0->pt0, Bit1->pt1... etc.

void PT_SetPulseTrain (uint8_t *index,* uint32_t *rate_control,* uint8_t *mode,* uint32_t *pattern*)

This function configures pulse train module.

Parameters

index	pulse train index
rate_control	pulse train output rate
mode	sets either square wave or pulse train mode; for pulse train mode, defines pulse train length, also, pattern parameter will be used. (range 1-32)
pattern	no effect in square wave mode; in pulse train mode, it contains the repeating pattern that will be shifted out as the pulse train output stream

2.16 Real-time Clock

Functions

- void RTC_Enable (void)
- void RTC_Disable (void)
- void RTC_SetVal (uint32_t value)
- void RTC_SetPrescale (mxc_rtc_prescale_t prescale)
- uint32_t RTC_GetVal (void)
- mxc_rtc_prescale_t RTC_GetPrescale (void)
- int8_t RTC_SetAlarm (uint32_t value, void(∗alarm_callback)(void))
- void RTC_ClearAlarm (int8_t alarm)
- int8_t RTC_SetContAlarm (mxc_rtc_prescale_t mask, void(∗alarm_callback)(void))
- void RTC_ClearContAlarm (int8_t alarm_num)
- void RTC_SetOvrfInt (void(∗overflow_cb)(void))

2.16.1 Detailed Description

This is the high level API for the real-time clock module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.16.2 Function Documentation

void RTC_ClearAlarm (int8_t *alarm*)

Clear the alarm set by RTC_SetAlarm()

Parameters

alarm	rtc_alarm returned by RTC_SetAlarm();

mxc_rtc_prescale_t RTC_GetPrescale (void)

Get the current value of the real-time clock prescale ticks.

Returns

mxc_rtc_prescale_t current real-time timer prescale.

uint32_t RTC_GetVal (void)

Get the current value of the real-time clock counter.

Returns

uint32_t current real-time timer value.

int8_t RTC_SetAlarm (uint32_t *value,* void(∗)(void) *alarm_callback*)

Set a one shot alarm at desired real-time clock timer match.

Parameters

value	rtc timer match value
alarm_callback	callback function when the rtc reached the value being

Returns

alarm number, -1 for error

int8_t RTC_SetContAlarm (mxc_rtc_prescale_t *mask*, void(∗)(void) *alarm_callback*)

Set a continuous alarm at desired real-time clock prescale mask value.

Parameters

mask	rtc timer prescale mask value
alarm_callback	callback function when the rtc reached the value being

Returns

alarm number, -1 for error

void RTC_SetPrescale (mxc_rtc_prescale_t *prescale*)

Set the prescale of real-time clock, the value of which it ticks as related to the RTC crystal.

Parameters

prescale	prescale will determine the accuracy of rtc.

void RTC_SetVal (uint32_t *value*)

Set and start the real-time clock.

Parameters

value	value wanted to be set for real-time timer.

2.17 SPI

Functions

- void SPI_Config (spi_slave_t *slave, uint8_t port)
- void SPI_ConfigClock (spi_slave_t *slave, uint8_t clk_high, uint8_t clk_low, uint8_t alt_clk_high, uint8_t alt_clk_low, uint8_t polarity, uint8_t phase)
- void SPI_ConfigSlaveSelect (spi_slave_t *slave, uint8_t slave_select, uint8_t polarity, uint8_t act_delay, uint8_t inact_delay)
- void SPI_ConfigPageSize (spi_slave_t *slave, spi_page_size_t size)
- void SPI_ConfigSpecial (spi_slave_t *slave, spi_flow_ctrl_t flow_ctrl, uint8_t polarity, uint8_t ss_sample_mode_en, uint8_t out_val, uint8_t drv_mode, uint8_t three_wire)
- uint8_t SPI_WaitFlowControl (spi_slave_t *slave)
- int32_t SPI_TransmitAsync (spi_slave_t *slave, const uint8_t *tx_buf, uint32_t tx_size, void(*tx_handler)(int32_ ret_status), uint8_t *rx_buf, uint32_t rx_size, void(*rx_handler)(uint32_t ret_size), spi_size_unit_t unit, spi_mode_t mode, uint8_t alt_clk, uint8_t last)
- int32_t SPI_Transmit (spi_slave_t *slave, const uint8_t *tx_buf, uint32_t tx_size, uint8_t *rx_buf, uint32_t rx_size, spi_size_unit_t unit, spi_mode_t mode, uint8_t alt_clk, uint8_t last)

2.17.1 Detailed Description

This is the high level API for the serial peripheral interface module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.17.2 Function Documentation

void SPI_Config (spi_slave_t * *slave,* uint8_t *port*)

Initialize SPI slave handle.

Parameters

slave	Pointer to spi_slave_t
port	Port to configure (0, 1, etc...)

void SPI_ConfigClock (spi_slave_t * *slave,* uint8_t *clk_high,* uint8_t *clk_low,* uint8_t *alt_clk_high,* uint8_t *alt_clk_low,* uint8_t *polarity,* uint8_t *phase*)

Set up SPI clocks.

Parameters

slave	Pointer to spi_slave_t
clk_high	Number of system clock ticks that SPI clock will be high
clk_low	Number of system clock ticks that SPI clock will be low
alt_clk_high	Number of system clock ticks that SPI clock will be high

alt_clk_low	Number of system clock ticks that SPI clock will be low
polarity	Clock polarity
phase	Clock phase

void SPI_ConfigPageSize (spi_slave_t ∗ *slave*, spi_page_size_t *size*)

Set up SPI page size.

Parameters

slave	Pointer to spi_slave_t
unit	Page size

void SPI_ConfigSlaveSelect (spi_slave_t ∗ *slave*, uint8_t *slave_select*, uint8_t *polarity*, uint8_t *act_delay*, uint8_t *inact_delay*)

Set up SPI slave select signals.

Parameters

slave	Pointer to spi_slave_t
slave_select	Slave select index
polarity	Polarity of slave select signal (0: active low; 1: active high)
act_delay	Delay between slave select assert and active SPI clock
inact_delay	Delay between active SPI clock and slave select deassert

void SPI_ConfigSpecial (spi_slave_t ∗ *slave*, spi_flow_ctrl_t *flow_ctrl*, uint8_t *polarity*, uint8_t *ss_sample_mode_en*, uint8_t *out_val*, uint8_t *drv_mode*, uint8_t *three_wire*)

Set up SPI special configuration.

Parameters

slave	Pointer to spi_slave_t
flow_ctrl	Flow control mode
polarity	Flow control polarity (0: active low; 1: active high)
ss_sample_mode	When asserted SDIO is driven prior to slave select assertion
out_val	Output value for SDIO prior to slave select assertion
drv_mode	Select which SDIO is driven prior to slave select assertion (0: MOSI, 1: MISO)
three_wire	Enable 3 wire mode (MISO and MOSI tied together)

int32_t SPI_Transmit (spi_slave_t ∗ *slave*, const uint8_t ∗ *tx_buf*, uint32_t *tx_size*, uint8_t ∗ *rx_buf*, uint32_t *rx_size*, spi_size_unit_t *unit*, spi_mode_t *mode*, uint8_t *alt_clk*, uint8_t *last*)

Read from and/or write to a SPI slave.

Parameters

slave	Pointer to spi_slave_t
tx_buf	Pointer to the buffer containing data to send
tx_size	Size of the data to send (maximum is 32)
ret_status	Argument to the callback function for return status; 0 => success.
rx_buf	Pointer to the buffer of receiving data
rx_size	Size of the data to read (maximum is 32)
unit	Unit for the size parameters
mode	Mode for the SPI transaction
alt_clk	Use alternate clock if asserted
last	If asserted SPI port will be cleaned up and slave select deasserted

Returns

0 => Success. Non zero => error condition.

int32_t SPI_TransmitAsync (spi_slave_t ∗ *slave*, const uint8_t ∗ *tx_buf*, uint32_t *tx_size*, void(∗)(int32_t ret_status) *tx_handler*, uint8_t ∗ *rx_buf*, uint32_t *rx_size*, void(∗)(uint32_t ret_size) *rx_handler*, spi_size_unit_t *unit*, spi_mode_t *mode*, uint8_t *alt_clk*, uint8_t *last*)

Read from and/or write to a SPI slave, return immediately, let the transmission be handled by the ISR and hardware FIFOs.

Parameters

slave	Pointer to spi_slave_t
tx_buf	Pointer to the buffer containing data to send
tx_size	Size of the data to send
tx_handler	Callback function to be called when data is finished being written to FIFO
ret_status	Argument to the callback function for return status; 0 => success.
rx_buf	Pointer to the buffer of receiving data
rx_size	Size of the data to read
rx_handler	Callback function to be called when data is finished being read from FIFO
ret_size	Size of the data read
unit	Unit for the size parameters
mode	Mode for the SPI transaction
alt_clk	Use alternate clock if asserted
last	If asserted SPI port will be cleaned up and slave select deasserted

Returns

0 => Success. Non zero => error condition.

uint8_t SPI_WaitFlowControl (spi_slave_t ∗ *slave*)

Get the current MISO status.

Parameters

slave	Pointer to spi_slave_t

Returns

current MISO status (0: active low; 1: active high)

2.18 TMON

Functions

- void TMON_Enable (uint8_t tmon_select)
- void TMON_Disable (void)
- void TMON_SetCurrent (mxc_afe_tmon_current_t tmon_current_val)

2.18.1 Detailed Description

This is the high level API for the internal temperature monitor.

2.18.2 Function Documentation

void TMON_Enable (uint8_t *tmon_select*)

Enable TMON.

Parameters

tmon_select	Internal TMON Circuit = 0, External TMON Circuit to AIN1+ = 1

void TMON_SetCurrent (mxc_afe_tmon_current_t *tmon_current_val*)

Setup of TMON current.

Parameters

tmon_current_val	TMON current value

2.19 Timer

Enumerations

- enum tmr_period_unit_t

Functions

- int32_t TMR32_TicksToPeriod (uint32_t ticks, uint8_t prescale, uint32_t ∗period, tmr_period_unit_t ∗units)
- int32_t TMR16_TicksToPeriod (uint16_t ticks, uint8_t prescale, uint32_t ∗period, tmr_period_unit_t ∗units)
- int32_t TMR32_PeriodToTicks (uint32_t period, tmr_period_unit_t unit, uint32_t ∗ticks, uint8_t ∗prescale)
- int32_t TMR16_PeriodToTicks (uint32_t period, tmr_period_unit_t unit, uint16_t ∗ticks, uint8_t ∗prescale)
- int32_t TMR32_Config (tmr32_config_t ∗cfg, int32_t index, mxc_tmr_mode_t mode, uint32_t ticks, uint8_t prescale, uint8_t duty_cycle, uint8_t polarity)
- int32_t TMR16_Config (tmr16_config_t ∗cfg, int32_t index, uint8_t sub_index, mxc_tmr_mode_t mode, uint32_t ticks, uint8_t prescale)
- int32_t TMR32_Start (tmr32_config_t ∗cfg, void(∗cb_fn)(uint32_t ticks))
- int32_t TMR16_Start (tmr16_config_t ∗cfg, void(∗cb_fn)(void))
- int32_t TMR32_Stop (tmr32_config_t ∗cfg)
- int32_t TMR16_Stop (tmr16_config_t ∗cfg)

2.19.1 Detailed Description

This is the high level API for the general purpose timer module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.19.2 Enumeration Type Documentation

enum tmr_period_unit_t

unit option for tick passing to timer_setup()

Enumerator

MXC_E_TMR_PERIOD_UNIT_NANOSEC nanosecond
MXC_E_TMR_PERIOD_UNIT_MICROSEC microsecond
MXC_E_TMR_PERIOD_UNIT_MILLISEC millisecond
MXC_E_TMR_PERIOD_UNIT_SEC second

2.19.3 Function Documentation

int32_t TMR16_Config (tmr16_config_t ∗ *cfg*, int32_t *index*, uint8_t *sub_index*, mxc_tmr_mode_t *mode*, uint32_t *ticks*, uint8_t *prescale*)

This function will allocate and configure a handle for the 16-bit timer. Only MXC_E_TMR_CTRL_ONE_SHOT and MXC_E_TMR_CTRL_CONTINUOUS modes are available for 16-bit timers.

Parameters

cfg	pointer to tmr16_config_t
index	timer index (-1 for auto-assign).
sub_index	timer sub index (0 or 1).
mode	timer mode.
ticks	period in ticks.
prescale	clock prescale value.

Returns

0 => Success. Non zero => error condition.

int32_t TMR16_PeriodToTicks (uint32_t *period*, tmr_period_unit_t *unit*, uint16_t ∗ *ticks*, uint8_t ∗ *prescale*)

Converts a period and units to a number of ticks and a prescale value for the 16-bit timer.

Parameters

period	period value.
unit	period units.
ticks	calculated number of ticks.
prescale	calculated timer clock prescale value.

Returns

0 => Success. Non zero => error condition.

int32_t TMR16_Start (tmr16_config_t ∗ *cfg*, void(∗)(void) *cb_fn*)

This function will start the timer using the configuration handle allocated and returned by TMR16_Config.

Parameters

cfg	pointer to tmr16_config_t
cb_fn	callback function.

Returns

0 => Success. Non zero => error condition.

int32_t TMR16_Stop (tmr16_config_t ∗ *cfg*)

This function will stop the timer using the configuration handle allocated and returned by TMR16_Config.

Parameters

cfg	pointer to tmr16_config_t

Returns

0 => Success. Non zero => error condition.

int32_t TMR16_TicksToPeriod (uint16_t *ticks*, uint8_t *prescale*, uint32_t ∗ *period*, tmr_period_unit_t ∗ *units*)

Converts a number of ticks and a prescale value to a period and units for the 16-bit timer.

Parameters

ticks	number of ticks.
prescale	timer clock prescale value.
period	calculated period value.
units	calculated period units.

Returns

0 => Success. Non zero => error condition.

int32_t TMR32_Config (tmr32_config_t * *cfg*, int32_t *index*, mxc_tmr_mode_t *mode*, uint32_t *ticks*, uint8_t *prescale*, uint8_t *duty_cycle*, uint8_t *polarity*)

This function will allocate and configure a handle for the 32-bit timer.

Parameters

cfg	pointer to tmr32_config_t
index	timer index (-1 for auto-assign).
mode	timer mode.
ticks	period in ticks.
prescale	clock prescale value.
duty_cycle	duty cycle (only used for TMR_CTRL_PWM mode).
polarity	polarity control for pad.

Returns

0 => Success. Non zero => error condition.

int32_t TMR32_PeriodToTicks (uint32_t *period*, tmr_period_unit_t *unit*, uint32_t * *ticks*, uint8_t * *prescale*)

Converts a period and units to a number of ticks and a prescale value for the 32-bit timer.

Parameters

period	period value.
unit	period units.
ticks	calculated number of ticks.
prescale	calculated timer clock prescale value.

Returns

0 => Success. Non zero => error condition.

int32_t TMR32_Start (tmr32_config_t * *cfg*, void(*)(uint32_t ticks) *cb_fn*)

This function will start the timer using the configuration handle allocated and returned by TMR32_Config.

Parameters

cfg	pointer to tmr32_config_t
cb_fn	callback function with number of ticks for capture modes.

Returns

0 => Success. Non zero => error condition.

int32_t TMR32_Stop (tmr32_config_t ∗ *cfg*)

This function will stop the timer using the configuration handle allocated and returned by TMR32_Config.

Parameters

cfg	pointer to tmr32_config_t

Returns

0 => Success. Non zero => error condition.

int32_t TMR32_TicksToPeriod (uint32_t *ticks*, uint8_t *prescale*, uint32_t ∗ *period*, tmr_period_unit_t ∗ *units*)

Converts a number of ticks and a prescale value to a period and units for the 32-bit timer.

Parameters

ticks	number of ticks.
prescale	timer clock prescale value.
period	calculated period value.
units	calculated period units.

Returns

0 => Success. Non zero => error condition.

2.20 UART

Functions

- int32_t UART_Config (uint32_t uart_index, uint32_t baud, uint8_t parity_enable, uint8_t parity_mode, uint8_t flow_control)
- int32_t UART_Write (uint32_t uart_index, const uint8_t ∗data, uint32_t length)
- int32_t UART_WriteAsync (uint32_t uart_index, const uint8_t ∗data, uint32_t length, void(∗tx_done_cb)(int32_t ret_code))
- int32_t UART_Read (uint32_t uart_index, uint8_t ∗data, uint32_t length)
- void UART_ReadAsync (uint32_t uart_index, uint8_t ∗data, uint32_t length, void(∗rx_cb)(int32_t ret_code))
- int32_t UART_Poll (uint32_t uart_index)

2.20.1 Detailed Description

This is the high level API for the universal asynchronous receiver/transmitter module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.20.2 Function Documentation

int32_t UART_Config (uint32_t *uart_index,* uint32_t *baud,* uint8_t *parity_enable,* uint8_t *parity_mode,* uint8_t *flow_control*)

Setup a uart for transfer with standard UART parameters.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)
baud	Baud rate (57600, 9600, etc..) The baud rate is calculated as a relation to system clock. If the system clock changes, it will affect the baud rate.
parity_enable	Enable or disable parity. (TRUE/FALSE)
parity_mode	odd = 0; even = 1.
flow_control	Enable or disable hardware flow control. (TRUE/FALSE)

Returns

0 => Success. Non zero => error condition.

int32_t UART_Poll (uint32_t *uart_index*)

Check to see if the UART has any bytes in its receive FIFO.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)

Returns

Number of bytes in receive FIFO.

int32_t UART_Read (uint32_t *uart_index,* uint8_t ∗ *data,* uint32_t *length*)

Read data from the UART RX fifo.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)
data	Pointer to location data will be placed.
length	Maximum number of bytes to read.

Returns

Number of bytes read.

void UART_ReadAsync (uint32_t *uart_index*, uint8_t ∗ *data*, uint32_t *length*, void(∗)(int32_t ret_code) *rx_cb*)

Read data from the UART receiver in an asynchronous manner. This function will return immediately and it will setup the UART for interrupt level time to populate read_count bytes into the data_buffer and call the rx_cb function when read_count bytes are received.

If this function is called with NULL as rx_cb, it will clear any rx callback and disable interrupt level reading from UART FIFO.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)
data	Pointer to received data allocation
length	Number of bytes to populate in data_buffer
rx_cb	Function called at interrupt level when read_count bytes are populated into data_buffer
ret_code	Number of bytes received or -1 for error condition

int32_t UART_Write (uint32_t *uart_index*, const uint8_t ∗ *data*, uint32_t *length*)

Write data to the UART TX fifo. Returns after all data has been submitted to the FIFO.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)
data	Pointer to the transmit data buffer.
length	Length of data buffer and number of bytes to transmit.

Returns

number of bytes written to fifo.

int32_t UART_WriteAsync (uint32_t *uart_index*, const uint8_t ∗ *data*, uint32_t *length*, void(∗)(int32_t ret_code) *tx_done_cb*)

Write data to the UART TX fifo asynchronously Returns immediately and allows data to be sent to TX FIFO as allowed.

Parameters

uart_index	Index of UART to configure. (0,1, etc...)
data	Pointer to the transmit data buffer.

length	Length of data buffer and number of bytes to transmit.
tx_done_cb	Callback for when all data has been sent to the FIFO (optional; NULL is valid)
ret_code	Callback argument; number of bytes sent or -1 for error condition

Returns

number of bytes written to fifo.

2.21 WDT

Functions

- int32_t WDT_EnableInt (uint8_t index, mxc_wdt_period_t int_period, void(∗int_cb_fn)(void), uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_DisableInt (uint8_t index, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_EnableWait (uint8_t index, mxc_wdt_period_t wait_period, void(∗prewin_cb_fn)(void), uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_DisableWait (uint8_t index, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_EnableReset (uint8_t index, mxc_wdt_period_t rst_period, uint8_t reboot, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_DisableReset (uint8_t index, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_Start (uint8_t index, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_Reset (uint8_t index, uint8_t unlock_key, uint8_t lock_key)
- int32_t WDT_Stop (uint8_t index, uint8_t unlock_key, uint8_t lock_key)

2.21.1 Detailed Description

This is the high level API for the watchdog timer interface module of the MAX32600 family of ARM Cortex based embedded microcontrollers.

2.21.2 Function Documentation

int32_t WDT_DisableInt (uint8_t *index*, uint8_t *unlock_key*, uint8_t *lock_key*)

Disables the interrupt timeout for the watchdog specified.

Parameters

index	Index of watchdog to disable.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_DisableReset (uint8_t *index*, uint8_t *unlock_key*, uint8_t *lock_key*)

Disables the reset timeout for the watchdog specified.

Parameters

index	Index of watchdog to disable.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_DisableWait (uint8_t *index,* uint8_t *unlock_key,* uint8_t *lock_key*)

Disables the pre-window timeout for the watchdog specified.

Parameters

index	Index of watchdog to disable.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_EnableInt (uint8_t *index,* mxc_wdt_period_t *int_period,* void(∗)(void) *int_cb_fn,* uint8_t *unlock_key,* uint8_t *lock_key*)

Configures and enables the interrupt timeout for the watchdog specified.

Parameters

index	Index of watchdog to configure and enable.
int_period	Interrupt period.
int_cb_fn	Interrupt callback function.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_EnableReset (uint8_t *index,* mxc_wdt_period_t *rst_period,* uint8_t *reboot,* uint8_t *unlock_key,* uint8_t *lock_key*)

Configures and enables the reset timeout for the watchdog specified.

Parameters

index	Index of watchdog to configure and enable.
rst_period	Reset period.
reboot	0 => reboot system. 1 => reset system.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_EnableWait (uint8_t *index,* mxc_wdt_period_t *wait_period,* void(∗)(void) *prewin_cb_fn,* uint8_t *unlock_key,* uint8_t *lock_key*)

Configures and enables the pre-window timeout for the watchdog specified.

Parameters

index	Index of watchdog to configure and enable.
wait_period	Pre-window period.
prewin_cb_fn	Pre-window callback function.

unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_Reset (uint8_t *index,* uint8_t *unlock_key,* uint8_t *lock_key*)

Feeds the watchdog specified.

Parameters

index	Index of watchdog to feed.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_Start (uint8_t *index,* uint8_t *unlock_key,* uint8_t *lock_key*)

Starts the watchdog specified.

Parameters

index	Index of watchdog to start.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.

int32_t WDT_Stop (uint8_t *index,* uint8_t *unlock_key,* uint8_t *lock_key*)

Stops the watchdog specified.

Parameters

index	Index of watchdog to stop.
unlock_key	Key to unlock watchdog.
lock_key	Key to lock watchdog.

Returns

0 => Success. Non zero => error condition.